# ANALOG AND DIGITAL COMPUTER TECHNOLOGY

Norman R. Scott
Professor of Electrical Engineering
University of Michigan

McGRAW-HILL BOOK COMPANY, INC.
NEW YORK TORONTO LONDON 1960

Valve Exhibit 1099
Valve v. Immersion

ANALOG AND DIGITAL COMPUTER TECHNOLOGY

Copyright © 1960 by the McGraw-Hill Book Company, Inc. Printed in the United States of America. All rights reserved. This book, or parts thereof, may not be reproduced in any form without permission of the publishers. *Library of Congress Catalog Card Number* 60-8843

THE MAPLE PRESS COMPANY, YORK, PA.

55815

# Preface

The scope of this book might be more precisely indicated by the title, "An Introduction to the Application and the Engineering Design of Direct-coupled Electronic Differential Analyzers and of Electronic Digital Computers." The first four chapters treat the d-c electronic differential analyzer, which is loosely referred to as the "analog computer," and the last seven treat the digital computer. In each of these two major parts of the book, the computer is first treated as a tool, and methods of using it to solve problems are introduced; then, its design is approached from an engineering point of view, and the appropriate engineering technology is introduced. The book is intended primarily for use by electrical engineers, and this double approach to each class of computer arises from the fact that the electrical engineer's relation to computers is unique, since he alone both uses and designs computers.

The book is an outgrowth of a course that has been presented at the University of Michigan since 1952. The course and the book are intended for beginning graduate students in electrical engineering or for seniors who have completed their undergraduate programs in mathematics and in electronics. The course from which the book grew is a one-semester (15 weeks) course with three 1-hour meetings per week. It is not possible to cover all the material in the book in such a course, but by choosing topics from both the analog and the digital parts, an instructor can present the most important aspects of each in a one-semester course. Such a course provides a broad background in computers for the student whose main interest is in another field and also furnishes a good foundation for the more advanced courses to be taken by the student who wishes to specialize in computers. It is also possible, of course, to teach a two-semester course from the book. Moreover, it is my hope that the book will be of use to the practicing engineer who is not taking a formal course but who needs to acquire a knowledge of computers.

Many books could be written about the topics that have been omitted from this one. Among these topics are repetitive differential analyzers, digital differential analyzers, analog-to-digital and digital-to-analog

converters, digital input-output devices, cryotrons, masers, parametrons, and many others. For the omission of these subjects I make no apologies. I have included what I felt needed to be included and omitted what I felt did not need to be included.

My thanks go to many people who have assisted me in the preparation of this book. The drawings were done by John Schick, Carl Dubois, and Guillermo Gonzales. Wendell Erickson assisted in the proofreading. Henry Ruston examined the whole manuscript with the eye of a friendly critic and made many helpful suggestions. Edna Neyer did a superb job of typing the manuscript. The most important assistance of all was given by my wife, who read the entire manuscript and helped me to polish the grammar and the style throughout.

*Norman R. Scott*

# Contents


**Preface** . . . . . v

**Chapter 1. Some Introductory Comments** . . . . . 1

1.1 A perspective of computation . . . . . 1
1.2 Two classes of computers . . . . . 2
The digital computer . . . . . 2
The analog computer . . . . . 3
1.3 The origin of the differential analyzer . . . . . 4
1.4 Electronic circuits of the differential analyzer . . . . . 6
The addition circuit . . . . . 6
The circuit for integrating . . . . . 8

**Chapter 2. Solution of Problems on Electronic Differential Analyzers** 12

2.1 Linear differential equations . . . . . 12
Potentiometer loading . . . . . 25
Minimal integrator setups . . . . . 26
Using the computer to generate simple forcing functions . . . . . 34
Generation of transfer functions using computer components . . . 35
2.2 Nonlinear differential equations . . . . . 40
Principle of the servo multiplier . . . . . 40
Some servo-multiplier applications . . . . . 41
Generation of inverse functions . . . . . 44
2.3 Other applications . . . . . 46
Integration with respect to variables other than time . . . . . 46
Evaluation of definite integrals . . . . . 47
Computer setups involving constraints . . . . . 49
Partial differential equations . . . . . 52

**Chapter 3. Computer Representation of Nonlinear Functions** . . . 58

3.1 Introduction . . . . . 58
3.2 Classification of nonlinear function generators . . . . . 59
3.3 Functional approximation . . . . . 60
Curve fitting with linear segments . . . . . 62
Curve fitting with polynomials . . . . . 65
Other types of approximations . . . . . 69
3.4 Some types of approximate function generators . . . . . 70
Use of diode switches for function generation . . . . . 70
Function generation with linear potentiometers . . . . . 82

3.5 Some examples of function generators based upon the true function . . 88
Tapered function potentiometers . . . . . 88
Mechanically driven curve followers . . . . . 91
Photoformers . . . . . 93
3.6 Generation of functions of two or more variables . . . . . 95
Three-dimensional cams . . . . . 95
Use of Laplace's equation . . . . . 95
Solution of the partial differential equation . . . . . 98
Line-segment function generators with variable bias . . . . . 98
Linear interpolation between $f(x,y_i)$ and $f(x,y_{i+1})$ . . . . . 100
3.7 Multipliers . . . . . 101
Instrumentation of a physical law . . . . . 102
Special-function devices . . . . . 104
Variable-transmission networks . . . . . 107
Modulation processes . . . . . 107
3.8 Simulation of time delay . . . . . 112

**Chapter 4. Computing Amplifiers** . . . . . 120

4.1 Introduction . . . . . 120
4.2 Frequency characteristics of ideal computing elements . . . . . 120
4.3 Sources of error . . . . . 122
4.4 Errors in computer solutions . . . . . 129
4.5 Automatic drift stabilization . . . . . 133
4.6 Reduction of drift voltages in d-c amplifiers . . . . . 140
4.7 Stabilization of feedback amplifiers . . . . . 145

**Chapter 5. Introduction to the Digital Computer** . . . . . 164

5.1 Quantized and digitized variables . . . . . 164
5.2 Representation of numbers in the computer . . . . . 165
5.3 The organization of digital computers . . . . . 169
5.4 The physical structure of the computer . . . . . 170

**Chapter 6. Solving Problems on the Digital Computer** . . . . . 173

6.1 Introduction . . . . . 173
6.2 A four-address iterative program . . . . . 174
6.3 Instruction modification and resetting . . . . . 177
6.4 Three-address instructions . . . . . 180
6.5 Modes of computation . . . . . 182
6.6 Scale factors . . . . . 184
6.7 Flow diagrams as an aid to programming . . . . . 188
6.8 One- and two-address instructions . . . . . 189
6.9 Minimum-access programming . . . . . 190
6.10 Special operations to facilitate computation . . . . . 191
6.11 Use of subroutines . . . . . 195
6.12 A complete order list . . . . . 197
6.13 A single-address program . . . . . 200
6.14 Structure of instruction words . . . . . 203
6.15 Automatic programming . . . . . 205
The tools of automatic programming . . . . . 207
Machine aids to error location . . . . . 210

**Chapter 7. Number Systems and Codes** . . . . . 217

7.1 Historical introduction . . . . . 217
7.2 Choice of radix . . . . . 218
Radix conversion . . . . . 220
7.3 Binary representations of numbers in arbitrary radix . . . . . 223
7.4 Representation of negative numbers . . . . . 227
7.5 Coded-decimal systems . . . . . 230
Weighted codes . . . . . 231
Excess-3 code, a modified weighted system . . . . . 234
A negative-radix code . . . . . 235
The reflected code . . . . . 237
7.6 Use of redundancy to avoid errors . . . . . 239
Parity checks . . . . . 240
The Hamming code . . . . . 242
Arithmetic checks . . . . . 243

**Chapter 8. Mathematical Logic and Switching Networks** . . . . 247

8.1 Introduction . . . . . 247
8.2 The logic of classes . . . . . 248
8.3 The logic of propositions . . . . . 251
8.4 Boolean algebra in switching networks . . . . . 255
Dual networks . . . . . 259
Inverse networks . . . . . 260
8.5 Manipulative techniques of Boolean algebra . . . . . 260
Canonical forms of Boolean functions . . . . . 263
8.6 Types of switching networks . . . . . 265
Multiterminal networks . . . . . 265
Combinational and sequential networks . . . . . 266
Iterative networks . . . . . 267
8.7 Some physical characteristics of switching elements . . . . . 268
8.8 Some comparisons of circuit costs . . . . . 272
Complete decoding nets . . . . . 273
8.9 Combinational circuits: Tools, techniques, and tricks . . . . . 278
Tables of combinations . . . . . 280
Function maps . . . . . 283
Nonnormal forms . . . . . 286
Quine's method of simplification . . . . . 288
Algebraic and "topological" methods . . . . . 292
Circuit realization by Shannon's method . . . . . 297
Circuit realization by matrix methods . . . . . 300
8.10 Sequential circuits . . . . . 305
Synchronous and asynchronous systems . . . . . 307
Methods of analysis and synthesis . . . . . 308

**Chapter 9. Logic Circuits for Arithmetic and Control** . . . . . 317

9.1 Introduction . . . . . 317
9.2 Circuits for adding . . . . . 317
Two-input adders (half-adders) . . . . . 323
Carry propagation . . . . . 326
Adders of the accumulator type . . . . . 329
9.3 Subtractors . . . . . 330

| | | |
|---|---|---|
| | Half-subtractors | 331 |
| 9.4 | Addition and subtraction of positive and negative numbers | 334 |
| 9.5 | Sign conventions and manipulations | 337 |
| 9.6 | Simple control circuits | 341 |
| | Shift registers | 342 |
| | Counters | 344 |
| | Pulse distributors (timing-pulse generators) | 345 |
| 9.7 | Methods of binary multiplication | 348 |
| 9.8 | Multiplication of complements | 353 |
| 9.9 | Methods of binary division | 357 |
| 9.10 | Square root | 362 |
| 9.11 | Round-off methods | 364 |
| 9.12 | Description of the logic of the MIC | 366 |
| | MIC characteristics and organization | 366 |
| | Control unit | 367 |
| | Instruction storage | 371 |
| | Operation-decoding circuits | 371 |
| | Locating a cell in memory | 372 |
| | Execution of operations | 375 |
| | Arithmetic circuits | 377 |
| | Communicating with the computer | 382 |
| **Chapter 10.** | **Electronic Circuits for Logic** | 388 |
| 10.1 | Introduction | 388 |
| 10.2 | Survey of gate circuits | 388 |
| | Vacuum-tube gates | 388 |
| | Diode gates | 390 |
| | Transistor gates | 391 |
| | Temporary storage in logic circuits | 392 |
| 10.3 | Dynamic circuitry | 393 |
| | Diode AND gates | 393 |
| | Diode OR gates | 398 |
| | A complete circuit | 399 |
| | Timing problems | 401 |
| | Delay lines | 403 |
| 10.4 | Static circuitry using vacuum tubes | 407 |
| 10.5 | Types of vacuum-tube flip-flops | 410 |
| 10.6 | Computer uses of flip-flops | 411 |
| 10.7 | Circuit design considerations for flip-flops | 413 |
| 10.8 | Switching considerations in flip-flop | 417 |
| 10.9 | Use of transistors in computer circuits | 418 |
| 10.10 | Transistors as switches | 419 |
| 10.11 | D-C conditions in the NOR circuit | 423 |
| 10.12 | Transient conditions in transistor switches | 427 |
| 10.13 | Logical design with the NOR circuit | 431 |
| **Chapter 11.** | **Storage in the Digital Computer** | 442 |
| 11.1 | The storage hierarchy | 442 |
| 11.2 | Properties of common types of storage systems | 442 |
| 11.3 | The storage property | 444 |
| 11.4 | Storage using magnetic cores | 445 |

11.5 Some engineering problems in core memories . . . . . 450
11.6 Properties of magnetic materials . . . . . 453
11.7 Ferrite and metal cores: manufacture and characteristics . . . . 459
11.8 Multidimensional selection . . . . . 462
11.9 Matrix switches . . . . . 463
11.10 Completely external selection . . . . . 468
11.11 Speeding up core memories . . . . . 471
Effect of core dimensions . . . . . 471
Partial switching . . . . . 472
Thin films . . . . . 472
Nondestructive sensing . . . . . 473
11.12 Magnetic-core shift registers . . . . . 474
11.13 Magnetic-core logic . . . . . 475
11.14 Multihole cores . . . . . 480
11.15 Electrostatic storage tubes . . . . . 483
11.16 Secondary emission and its effect upon target potentials . . . . . 484
11.17 Redistribution storage . . . . . 487
11.18 Reliability of redistribution storage . . . . . 490
11.19 Storage by potential of target elements . . . . . 491
Equilibrium storage . . . . . 491
Bistable storage with a holding beam . . . . . 492
11.20 Storage by magnetic recording on moving surfaces . . . . . 494
11.21 Some engineering considerations in digital magnetic recording . . . 495
11.22 Storage using magnetic tape . . . . . 500
11.23 Other forms of memory . . . . . 501

**Appendix** . . . . . 510

**Index** . . . . . 515

CHAPTER 1

# Some Introductory Comments

## 1.1 A perspective of computation

The history of machine aids to computation is almost as long as the history of mathematics itself. For many centuries man has used various devices, both physical and symbolic, to help him in carrying out computations. Indeed, the word "calculation" and its variants are derived from the Latin word "calculus", or stone, reflecting the Romans' use of stones or pebbles as counters in executing computations. Tracing the development of mathematics and of computational aids through the centuries is an absorbing study, and the reader is urged to do some collateral browsing among the many fine works that have been published on this subject.[1–3] Our concern here, however, is solely with the modern high-speed electronic devices that have come into use since about 1945, and our historical references will be limited to those which bear directly upon these recent developments.

It is not an overstatement to say that high-speed electronic computers are revolutionizing man's approaches to problem solving. A broad variety of computers now exist that allow computations to be carried out several thousand to several million times as rapidly as was possible with the most elaborate devices in use even as recently as 1940. The present availability of high-speed computing devices has the obvious implication that problems heretofore solved by slow and tedious hand or semimechanized methods can now be solved much more rapidly. A somewhat subtler and much more significant implication is that problems never before attacked because of their size have now been brought within the realm of practicality and can be attacked with genuine hope of success. Furthermore, new mathematical methods of problem solving particularly applicable to high-speed computers have been developed, and applied mathematics has received both new stimulus and new stature.

A good example of the change that has occurred in problem-solving techniques is found in the aircraft industry. In the 1930s and early 1940s, each new aircraft design was first constructed and then evaluated

by being put through an elaborate sequence of ground tests, followed by an equally elaborate series of flight tests. This procedure was slow, inefficient, and often hazardous. The modern procedure consists in simulating the behavior of the aircraft components and of the complete system by means of computers before the aircraft is built. Many aircraft are fully "flight-tested" in this way before they are built. Of course, the same elaborate tests are still conducted upon the completed aircraft, but the simulation phase allows unsuitable designs to be eliminated or modified before they are constructed.[4]

Many other areas for application of computers may be mentioned. It now appears feasible to use computers as control elements in a feedback loop around large continuous-process production systems such as oil refineries or chemical plants.[5,6] Problems of inventory control, which have been "solved" in the past by inadequate combinations of personal judgment, past experience, and overstocking, can now be solved much more efficiently by use of computers.[7] Manipulations of immense amounts of data, such as those obtained from a population census or from radar trackings of earth satellites, can now be performed quickly enough to provide significant data before the problem has changed.[8] Computers are being adopted in business and commerce for file maintenance where large amounts of data must be filed in such a way as to be readily accessible for reference and up-dating.[9] In fact, there is currently much interest in the use of computers in what is known as "information retrieval", the generalized library-search problem.[10] Another nonmathematical problem is automatic language translation, a task which appears formidable but in which there have already been some preliminary partial successes.[11]

This list of areas for application of computers is by no means exhaustive and is meant merely to indicate the variety of uses to which computers can be put. The reader will find it entertaining to add to this list, especially if he should encounter it a few years after its publication.

## 1.2 Two classes of computers

Although there are many ways in which electronic computers may be classified—according to their intended application, their size, their speed, or their internal organization—the principal classification that we shall make in this book divides computing devices into the two broad classes of analog computers and digital computers.

**The digital computer.** The digital computer may be defined simply as the computer that does arithmetic. It manipulates numbers and the individual digits that constitute these numbers and often executes such basic processes as addition, subtraction, multiplication, and division in

very much the same way as a human arithmetician. A simple example of a digital device is the abacus, which is widely used in one form or another in many Asiatic countries. It should be noted that the abacus is, strictly speaking, not a computer but merely an aid to computation, since it is the human operator who does the computing. The abacus merely provides a convenient means of recording and modifying intermediate results during the computation. Figure 1.1 shows photographs of a Chinese abacus (suan pan) on the left and a Japanese abacus (soroban) on the right. The Chinese version, it will be observed, has two beads above the bar on each rod and five below; the Japanese version shown has one bead above and five below. Each rod corresponds to a decimal-digit position. Each bead above the bar contributes a weight of 5 in

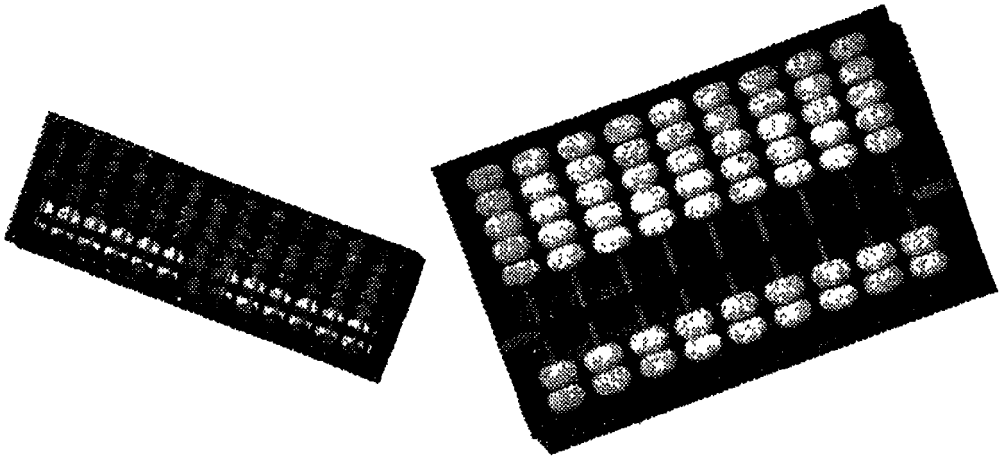


Fig. 1.1 A Chinese abacus (left) and a Japanese abacus (right)

that position when it is moved down to the bar, and each bead below contributes a weight of 1 when it is moved up to the bar.

Needless to say, the high-speed electronic digital computers that we shall treat in the latter part of this book are vastly more sophisticated devices than the abacus. However, like the abacus, they are digital devices and manipulate numbers. Even the directions that the operator gives to the digital computer are numbers, and it is principally in the methods of storing, manipulating, and using these numerical "instructions" that the sophistication of the digital computer is found.

Modern though the digital computer is, the fundamental principles of its organization were set forth long before the era of electronics by the English mathematician Charles Babbage in the first half of the nineteenth century.[12] Babbage's designs were beyond the capabilities of the industrial technology of his day, however, and it is only in recent years that his work has appeared in its full significance.

**The analog computer.** Broadly speaking, the term "analog computer" means any device in which the variables are analogous to those

of the problem to be solved and behave in the same way, permitting measurements of the analog to be substituted for measurement of the problem variables. Although the term "analog" carries of itself no connotation of continuity, "analog computer" has come to mean the computer that manipulates continuous variables. A simple example of an analog computing element is the common slide rule. Here the distances along the two sticks are analogs of the logarithms of the numbers to be multiplied or divided. The sum of these distances is analogous to the sum of the logarithms, and, since the sticks are calibrated in antilogarithms, the device functions as a direct-reading analog multiplier or divider. (The fact that input and output data are in digital form does not alter the essentially analog nature of the device.)

Some analog computers perform a direct simulation of a physical system, with each section of the computer identifiable as the analog of some particular physical portion of the system being studied. A good example is the a-c network analyzer used by many electric utilities to aid them in planning system operation and growth. Individual generators, loads, transformers, and transmission lines are simulated at higher frequencies (e.g., 10 kc) by lumped electric resistances and reactors, and measurements on the model can then be converted to yield information about the system.

The direct-analogy type of computer is a less versatile device, however, than the mathematical analog computer, in which individual parts are the analogs of terms in the equations describing the system under study. The mathematical analog computer contains devices that perform various simple mathematical operations such as addition, multiplication, and integration. Suitable combinations of these devices allow systems of equations to be set up, and the solutions are obtained by making direct measurements at points of interest. The analog computers to be described in this book are of this type. In fact, we shall restrict our attention to a particular class of mathematical analog computers: those which contain integrating devices and which can, therefore, be used in solving differential equations. The great majority of analog machines are of this type. These computers have come to be known as "differential analyzers", for this was the name given to the first such machine, an entirely mechanical device built by Dr. Vannevar Bush at the Massachusetts Institute of Technology and completed in 1931.[13]

## 1.3 The origin of the differential analyzer

The integrating element used in the Bush differential analyzer is an interesting device. Its basic features are shown in Fig. 1.2. It follows readily from the diagram that the total angle of rotation of the output

shaft is given by

$$\theta = \frac{1}{R} \int x \, dy \tag{1.1}$$

Thus, if means are provided to move the small wheel across the large wheel in such a way that the distance $x$ is always proportional to the quantity to be integrated and if the angular rotation of the large disk is made proportional to the variable of integration, then the total rotation of the output shaft is proportional to the integral $\int x \, dy$.

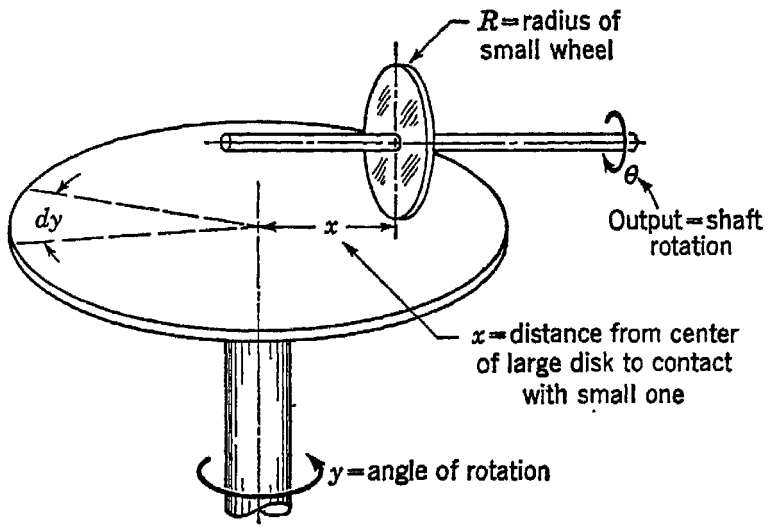



Fig. 1.2 The disk integrator

The success of the Bush integrator was largely due to its use of an ingenious torque-amplifying mechanism that enabled the integrator output shaft to drive other loads without slipping. This amplifier is less interesting for our present purposes, however, than the principle on which the integrator operates. The Bush integrator is actually an adaptation of an integrator invented by James Thomson, the brother of Lord Kelvin, about 1875.[14] Kelvin, in applying the integrator, was led to set forth the fundamental philosophy upon which all differential analyzers now operate. He was working upon the differential equation

$$\frac{d}{dx}\left[\frac{1}{P(x)}\frac{dy}{dx}\right] + y = 0 \tag{1.2}$$

When integrated twice, this equation yields

$$y = \int P(x)(c - \int y \, dx) \, dx \tag{1.3}$$

In Kelvin's words,[15] "Now let my brother's integrator be applied to find $c - \int_0^x u_1 \, dx$, and let its result feed . . . continuously a second machine,

which shall find the integral of the product of its result into $P(x)\,dx$. The second machine will give out continuously the value of $u_2$. Use again the same process with $u_2$ instead of $u_1$, and then with $u_3$ and so on.

"After thus altering $u_1$ into $u_2$ by passing it through the machine, then $u_2$ into $u_3$ by a second passage through the machine, and so on, the thing will . . . become refined into a solution which will be more and more nearly rigorously correct the oftener we pass it through the machine. If $u_{n+1}$ does not differ sensibly from $u_n$, then each is sensibly a solution.

"So far I had gone and was feeling satisfied, feeling I had done what I wished to do for many years. But then came a pleasing surprise. Compel agreement between the function fed into the double machine and that given out by it."

It is this last statement that sums up the basic philosophy of the method of interconnecting computing elements to solve differential equations. At the beginning of a succession of integrations, the unknown variable that is sought as a solution is regarded as an available input. The output of the last integrator, which would be the desired solution if the first integrator had been fed the solution, is then applied to the first integrator as its input in order to "compel agreement between the function fed in . . . and that given out."

Kelvin goes on to say, "Thus I was led to a conclusion which was quite unexpected; and it seems to me remarkable that the general differential equation of the second order with variable coefficients may be rigorously, continuously, and in a single process solved by a machine."

## 1.4 Electronic circuits of the differential analyzer

In Chap. 2 we shall study applications of this method in solving differential equations by means of the electronic differential analyzer. First, however, let us study briefly the action of electronic devices that can be used to perform the basic processes of addition and integration, which are the fundamental operations performed in the electronic differential analyzer. In this machine voltages are used as the analogs of the dependent variables, and time is used as the variable of integration. The choice of voltage as the analog of the dependent variables reflects the fact that, in the present state of electronic technology, voltage-sensitive and voltage-measuring devices are simpler and more easily constructed than current-sensitive and current-measuring devices. This may not always be so, however, and the reader is cautioned not to feel that voltages will necessarily always be the best choice.

**The addition circuit.** Addition of voltages can be accomplished by means of a resistor network, as shown in Fig. 1.3. By summing the currents into the common node, we see that

$$\frac{e_1 - e_o}{R_1} + \frac{e_2 - e_o}{R_2} + \cdots + \frac{e_n - e_o}{R_n} = \frac{e_o}{R_0} + \frac{e_o}{R_L} \tag{1.4}$$

$$e_o = \frac{(e_1/R_1) + (e_2/R_2) + \cdots + (e_n/R_n)}{(1/R_1) + (1/R_2) + \cdots + (1/R_n) + (1/R_0) + (1/R_L)} \tag{1.5}$$

If the load resistance $R_L$ is of constant value, then the output voltage $e_o$ is proportional to the weighted sum of the input voltages $e_1, e_2, \ldots, e_n$. However, depending on the requirements of the problem being solved, the output voltage may be applied to one or several other points in the computer; so the effective value of $R_L$ is not fixed. To make $e_o$ independent of $R_L$, one could make $R_1, R_2, \ldots, R_n \gg R_L$, or $R_0 \ll R_L$, or both. Unfortunately, either of these steps makes $e_o$ very small in comparison with the input voltages, and an amplifier must be used to restore $e_o$ to

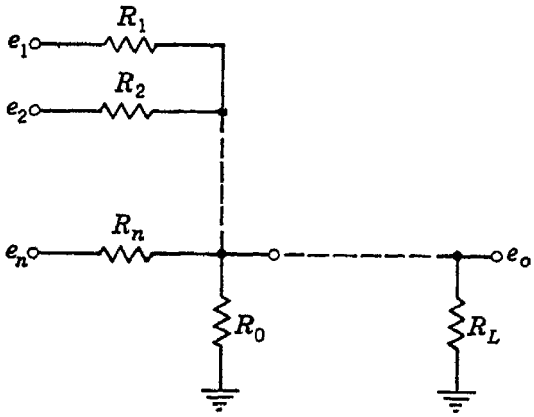



Fig. 1.3 Voltage summation by a resistor network

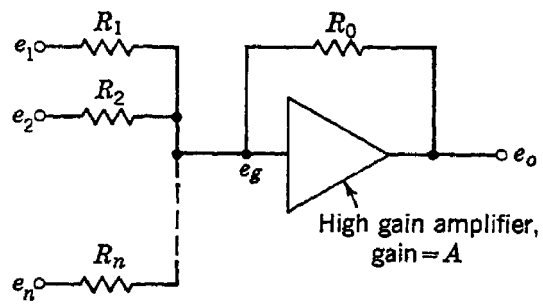



Fig. 1.4 Summing network using an amplifier

their order of magnitude. Simple amplification of $e_o$ also proves unsatisfactory, however, since it makes $e_o$ directly proportional to the amplifier gain, which in turn varies both rapidly with supply-voltage changes and slowly with component aging. The best way to use the amplifier is to employ it in a current-feedback circuit, as shown in Fig. 1.4; this makes the resistive network a true current-summing system independent of the load resistance $R_L$.

Let us assume the use of a high-quality amplifier for which the input impedance approximates infinity. The voltage at the amplifier input terminal we shall call $e_g$. (All voltages are taken to be measured with ground as the reference.) Two equations may be written, one summing the currents at the amplifier input terminal and the other relating amplifier input and output voltages:

$$\frac{e_1 - e_g}{R_1} + \frac{e_2 - e_g}{R_2} + \cdots + \frac{e_n - e_g}{R_n} = \frac{e_g - e_o}{R_0} \tag{1.6}$$

$$e_o = Ae_g \tag{1.7}$$

The first of these may be rearranged to give

$$\frac{e_1}{R_1} + \frac{e_2}{R_2} + \cdots + \frac{e_n}{R_n} = e_g\left(\frac{1}{R_0} + \frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_n}\right) - \frac{e_o}{R_0} \tag{1.8}$$

Replacing $e_g$ by $e_o/A$ gives

$$\frac{e_1}{R_1} + \frac{e_2}{R_2} + \cdots + \frac{e_n}{R_n} = -e_o\left[\frac{1}{R_0} - \frac{1}{A}\left(\frac{1}{R_0} + \frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_n}\right)\right] \tag{1.9}$$

Finally,

$$e_o = -\frac{(e_1/R_1) + (e_2/R_2) + \cdots + (e_n/R_n)}{(1/R_0) - (1/A)[(1/R_0) + (1/R_1) + (1/R_2) + \cdots + (1/R_n)]} \tag{1.10}$$

If $A$ is made to be a large number, of the order of $10^4$ to $10^6$, then $e_o$ is given more simply by

$$e_o = -\frac{R_0}{R_1}e_1 - \frac{R_0}{R_2}e_2 - \cdots - \frac{R_0}{R_n}e_n \tag{1.11}$$

This result has several interesting features. (1) The output voltage is a simple weighted sum of the input voltages, and the weighting factors $-R_0/R_j$ are independently adjustable by varying the input resistors. (2) Although an amplifier is used, its gain $A$ does not appear in the final expression for $e_o$ if $A$ is sufficiently large to permit neglect of the second term in the denominator of Eq. (1.10). (3) The circuit produces a sign reversal, so each weighting factor is negative. [The reader with some knowledge of feedback systems will recognize that, if the feedback system of Fig. 1.4 is to be stable, $A$ must actually be a negative real number. It was not this fact, however, that produced the negative signs in Eq. (1.11), since $A$ does not appear in that equation. The sign reversal occurs simply because, if $e_g$ is forced to be approximately zero, as it is here, then currents flowing into the common node due to potentials of one polarity at the input must flow out through $R_0$ to a potential of the opposite polarity at the output.]

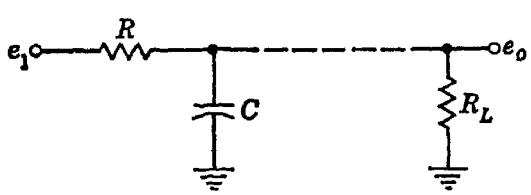


Fig. 1.5 *RC* integrator

**The circuit for integrating.** Integration of voltages can be accomplished by means of an *RC* network, as shown in Fig. 1.5. The transfer function of this network is readily expressed in terms of Laplace-trans-

form-operator notation:

$$\frac{E_o(s)}{E_1(s)} = \frac{\dfrac{R_L \cdot (1/sC)}{R_L + (1/sC)}}{R + \dfrac{R_L \cdot (1/sC)}{R_L + (1/sC)}} \tag{1.12}$$

$$= \frac{1}{RC} \cdot \frac{1}{s + (1/C)(1/R + 1/R_L)} \tag{1.13}$$

If $R$, $R_L$, and $C$ are all made very large, then the transfer function may be approximated by

$$\frac{E_o(s)}{E_1(s)} = \frac{1}{RCs} \tag{1.14}$$

or

$$e_o = \frac{1}{RC} \int e_1 \, dt \tag{1.15}$$

Making $R_L$ large causes it to have a negligible shunting effect upon the capacitance. Making $R$ and $C$ large causes the current flow through $R$ to be approximately

$$i_R = \frac{e_1}{R} \tag{1.16}$$

Thus the voltage across $C$ is given approximately by

$$e_o = \frac{1}{C} \int \frac{e_1}{R} \, dt \tag{1.17}$$

which is the result found in Eq. (1.15).

The principal defect in this circuit is that the very circumstances that permit it to yield a good approximation to the integral make its output voltage so small as to be unusable without amplification. As with the resistive network for adding voltages, the best way to use an amplifier is to incorporate it in a feedback circuit. Figure 1.6 shows the arrangement usually employed. Again we shall assume that a high-quality amplifier of very large input impedance is employed. The following equations are obtained:

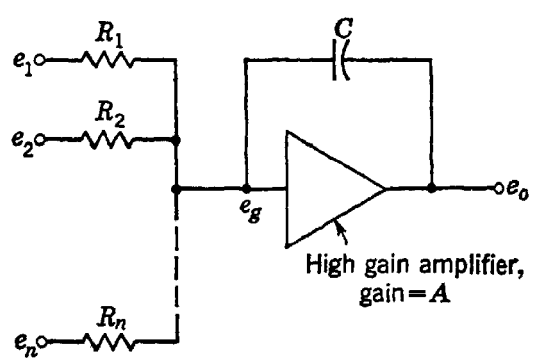



Fig. 1.6 Integrating network using an amplifier

$$\frac{E_1(s) - E_g(s)}{R_1} + \cdots + \frac{E_n(s) - E_g(s)}{R_n} = \frac{E_g(s) - E_o(s)}{1/sC} \tag{1.18}$$

and

$$E_o(s) = A \cdot E_g(s) \tag{1.19}$$

Proceeding as before, we obtain

$$\frac{E_1(s)}{R_1} + \cdots + \frac{E_n(s)}{R_n} = E_g(s)\left(sC + \frac{1}{R_1} + \cdots + \frac{1}{R_n}\right) - E_o(s) \cdot sC \tag{1.20}$$

$$\frac{E_1(s)}{R_1} + \cdots + \frac{E_n(s)}{R_n} = -E_o(s)\left[sC - \frac{1}{A}\left(sC + \frac{1}{R_1} + \cdots + \frac{1}{R_n}\right)\right] \tag{1.21}$$

and finally

$$E_o(s) = -\frac{[E_1(s)/R_1] + \cdots + [E_n(s)/R_n]}{sC - (1/A)[sC + (1/R_1) + \cdots + (1/R_n)]} \tag{1.22}$$

If $A$ is taken here also to be a large number, of the order of $10^4$ to $10^6$, then we obtain the simpler expression

$$E_o(s) = -\frac{E_1(s)}{sCR_1} - \frac{E_2(s)}{sCR_2} - \cdots - \frac{E_n(s)}{sCR_n} \tag{1.23}$$

or

$$e_o = -\frac{1}{R_1C}\int e_1\,dt - \frac{1}{R_2C}\int e_2\,dt - \cdots - \frac{1}{R_nC}\int e_n\,dt \tag{1.24}$$

The same points that were noted for the adding circuit may be observed here: (1) the output voltage is a simple weighted sum of the integrals of the input voltages, and the weighting factors $-1/R_jC$ are independently adjustable by varying the input resistors; (2) the gain $A$ does not appear in the final expression for $e_o$; and (3) a sign reversal is inherent in the circuit, so each weighting factor is negative.

These two circuits, one for adding voltages and one for adding and integrating voltages, are the fundamental building blocks used in the electronic differential analyzer. Subtraction is performed by letting the two input quantities have opposite signs. Simple sign reversal is performed by an adding circuit in which $R_0 = R_1$. Multiplication by a fixed quantity can be performed incidentally in either the adder or the integrator by an appropriate setting of the weighting factor. All these techniques and many others, such as multiplication and division by variable quantities, are elaborated in the next three chapters, in which we examine in detail the use of the fundamental principles presented in this chapter.

## REFERENCES

1. Dantzig, Tobias, "Number, the Language of Science," 4th ed., The Macmillan Company, New York, 1954. Also available in paperback, published by Doubleday Anchor Books, Garden City, N.Y., 1956.

2. Struik, Dirk J., "A Concise History of Mathematics," Dover Publications, New York, 1948.
3. Bell, E. T., "The Development of Mathematics," 2d ed., McGraw-Hill Book Company, Inc., New York, 1945.
4. Wadel, L. B., and C. C. Wan, Aircraft performance studied on an electronic analog computer, *Proc. Western Joint Computer Conf., 1955*, pp. 78–82.
5. Engel, H. L., Computing-control applied to a fractionating column, *Control Eng.*, vol. 4, no. 9, September, 1957, pp. 144–147.
6. Salzer, J. M., Controlling a process, *Control Eng.*, vol. 3, no. 12, December, 1956, pp. 95–99.
7. Hurd, C. C., Applications of electronic data-processing machines, in E. M. Grabbe (ed.), "Automation in Business and Industry," John Wiley & Sons, Inc., New York, 1957.
8. Quarles, D. A., Jr., Preparations for tracking artificial earth-satellites at the Vanguard Computing Center, *Proc. Eastern Joint Computer Conf., Dec. 9-13, 1957*, pp. 58–64.
9. Adams, C. W., Processing business data, *Control Eng.*, vol. 3, no. 6, June, 1956, pp. 105–112.
10. Perry, J. W., M. M. Berry, F. U. Luehrs, Jr., and A. Kent, Automation of information retrieval, *Proc. Eastern Joint Computer Conf., Dec. 8–10, 1954*, pp. 68–73.
11. Booth, Andrew Donald, and William N. Locke, "Machine Translation of Languages," John Wiley & Sons, Inc., New York, 1955.
12. Bowden, B. V. (ed.), "Faster than Thought," Pitman Publishing Corporation, New York, 1953, pp. 341–409.
13. Bush, V., The differential analyzer: A new machine for solving differential equations, *Franklin Inst. J.*, vol. 212, no. 4, October, 1931, pp. 447–488.
14. Thomson, James, On an integrating machine having a new kinematic principle, *Proc. Roy. Soc. London*, vol. 24, 1876, pp. 262–265.
15. Thomson, Sir William, Mechanical integration of the linear differential equations of the second order with variable coefficients, *Proc. Roy. Soc. London*, vol. 24, 1876, pp. 269–271.

CHAPTER 2

# Solution of Problems on Electronic Differential Analyzers

## 2.1 Linear differential equations[1,2]

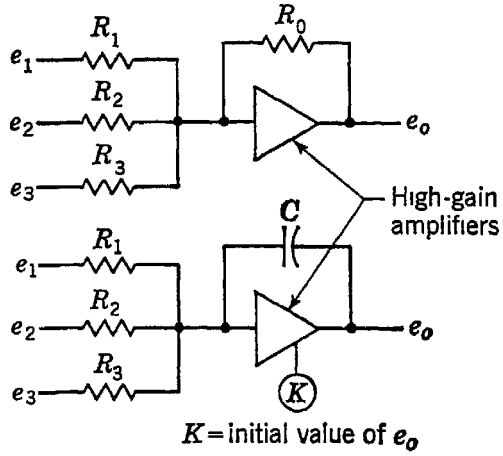
As we have already seen, high-gain d-c amplifiers in conjunction with resistors or capacitors for their input and feedback elements can be used to perform the simple operations of addition, sign inversion, multiplication by a constant, and integration. Since these are the operations that characterize the linear differential equation, an analogy to such an equation can be set up by an appropriate interconnection of computing amplifiers arranged to perform these functions. Voltages produced at various points in the computer setup are analogs of terms in the differential equation, and time is the analog of the independent variable. Linear computing elements and their transfer functions are shown in Fig. 2.1.

In order that the computer setup may be more than a mere analogy to the equation and may provide useful quantitative data, a very careful correspondence must be established between the quantities in the equation and the analogous quantities in the computer. Proportionalities between the voltages in the computer and the dependent variables in the problem must be chosen, and so must the relation between time in the computer solution and time or another independent variable in the equation of the problem. The selection of these scale factors is governed largely by the performance capabilities of the computer components. We shall consider these matters in some detail after first examining the procedure for setting up the computer analog of the differential equation.

As an example of the procedure, let us take the differential equation describing the motion of a mass-spring-damper system with a forcing function $f(t)$ applied:

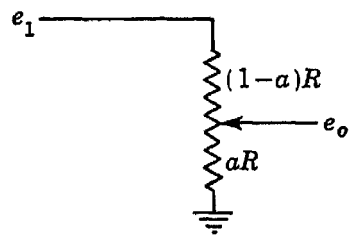
$$M\ddot{x} + B\dot{x} + Kx = f(t) \tag{2.1}$$

Using only the components of Fig. 2.1, we must evidently set up our analogy by performing successive integrations of the highest-order deriv-




$$e_o = -\frac{R_0}{R_1}e_1 - \frac{R_0}{R_2}e_2 - \frac{R_0}{R_3}e_3 = -R_0\sum_{j=1}^{n}\frac{e_j}{R_j}$$

$$e_o = -\frac{1}{R_1C}\int_0^t e_1dt - \frac{1}{R_2C}\int_0^t e_2dt - \frac{1}{R_3C}\int_0^t e_3dt + K = -\frac{1}{C}\sum_{j=1}^{n}\int_0^t \frac{e_j}{R_j}dt + K$$




$$e_o = ae_1$$

Fig. 2.1 Linear computing elements

ative of the dependent variable. If the analog of this quantity were available as a voltage input, two integrations would be required to produce a voltage analogous to $x$. The equation itself furnishes the information needed to determine $\ddot{x}$, and we observe that

$$\ddot{x} = -\frac{B}{M}\dot{x} - \frac{K}{M}x + \frac{1}{M}f(t) \tag{2.2}$$

A possible computer setup for this equation is shown in Fig. 2.2.

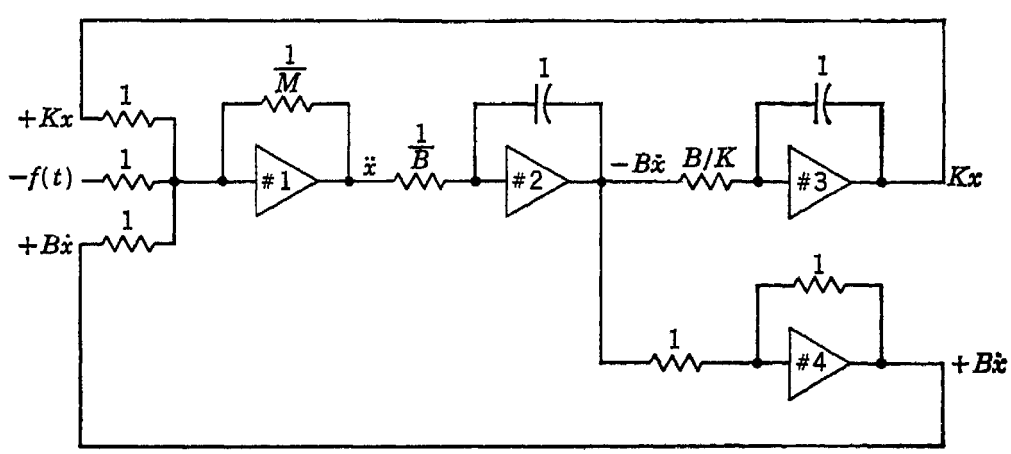



Fig. 2.2 Mass-spring-damper analog

In this setup amplifier 1 performs the summation of the three inputs, their multiplication by $1/M$, and a sign inversion to produce a voltage proportional to $\ddot{x}$ in accordance with Eq. (2.2). Amplifier 2 integrates this quantity, multiplies it by $B$, and inverts its sign to form $-B\dot{x}$.

Amplifier 3 performs another integration, sign inversion, and constant multiplication to form $Kx$, one of the inputs needed for 1. Amplifier 4 merely inverts $-B\dot{x}$ to give the proper sign for another of the inputs to 1.

The solution of the problem is started by simultaneously applying $f(t)$ and disconnecting any voltage sources that were present to establish initial charges on the integrating capacitors. Such voltage sources ordinarily consist of circuits like that shown in Fig. 2.3. With the switch in the RESET position, a voltage of $-aE$ volts is developed across the integrating capacitor in a few seconds, and, when the switch is put in OPERATE, the integration commences from this initial value.

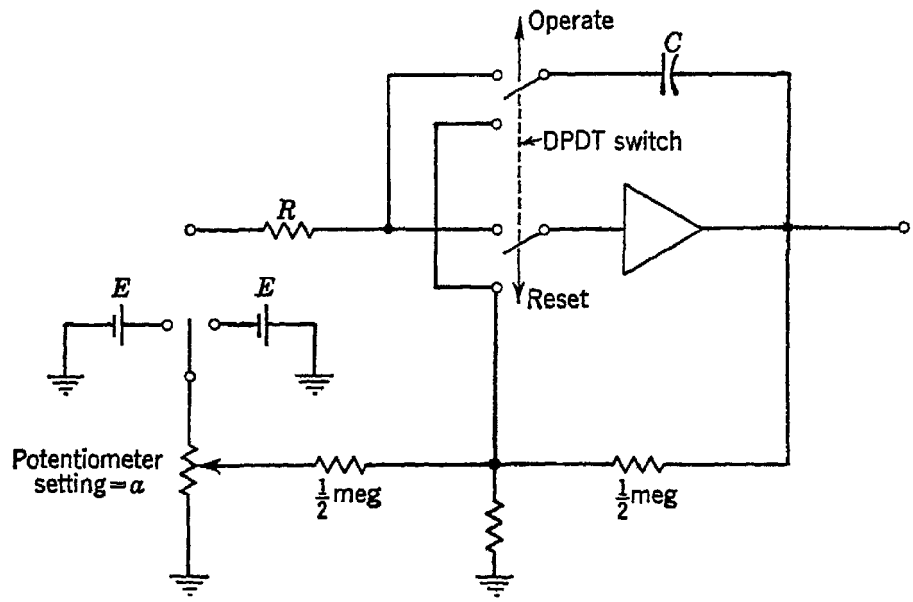



Fig. 2.3 Circuit for placing initial charge on integrating capacitors

The steps taken in the foregoing example may be stated in general terms as follows:

1. Solve the equations of the system for the highest-order derivative of each of the dependent variables.

2. Assuming that each of these highest-order-derivative quantities is available as an input, show a chain of integrators from left to right to produce each dependent variable at the right of the diagram.

3. Before the first integrator in each chain, show a summing amplifier to add those terms whose sum, according to the equation, should equal the negative of the first integrand. (Alternatively, this sum can be formed by the first integrator, as in the circuits of Fig. 2.4.)

4. Proceeding from left to right, obtain the quantities required as inputs to each chain of integrators by tapping them off wherever they are found. Adjust magnitudes by inclusion of potentiometers and amplifiers and by suitable choice of input and feedback elements.

5. Show the initial value of the integral on the diagram beside each integrator that is to have a nonzero initial condition.

Although this procedure is a useful working method, it does not in general yield a computer circuit that is optimum in the sense of requiring a minimum number of computing amplifiers or a minimum number of integrators. The problem of minimization has been studied by several workers, but no general solution has been found. It should also be noted

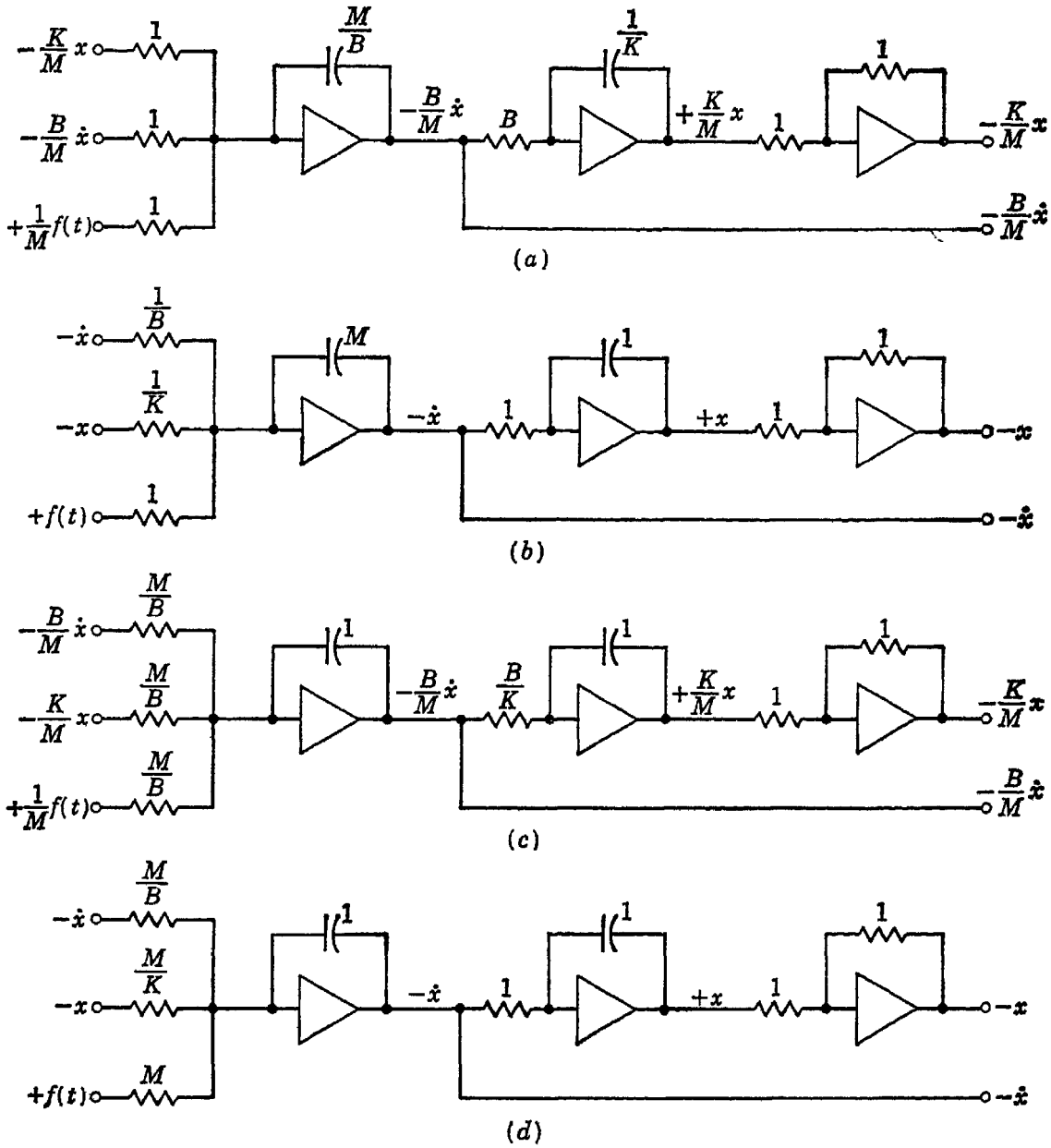



Fig. 2.4 Some equivalent setups for the mass-spring-damper problem

that there is no unique computer setup for any equation and that, theoretically at least, there exists an infinite number of setups satisfying any equation. This is trivially evident from the fact that any voltage can be inverted and reinverted without affecting the solution.

Even without the inclusion of redundant amplifiers, there are many different ways to set up a problem on a computer. Although under ideal

circumstances these would all be equivalent, in practice some of them are preferred to others. As an example, the circuit of Fig. 2.2 can be implemented in many ways that use only three amplifiers instead of four. Some of these are shown in Fig. 2.4.

Although any self-consistent system of units can be used to give the values of the resistors and capacitors, in the present state of the art it has been found most generally convenient to let the values of the resistors be given in megohms and the capacitors in microfarads. The magnitudes in Figs. 2.2 and 2.4 may be interpreted this way. Circuits *a* and *b* of Fig. 2.4 have the disadvantage that they use capacitors of arbitrary size. Although it is possible to use high-precision decade capacitors for these circuits, their cost is very high; therefore, circuits *c* and *d*, which use the same values of capacitance for all integrators, are much better. Precision resistors are much less expensive than precision capacitors; hence, adjustment of coefficients is better done by using different sizes of resistors than by changing integrating capacitors. Circuits *c* and *d* differ principally in the magnitudes of the voltages appearing at corresponding points. This difference is superficial, however, since the magnitudes of these voltages can be altered by changing the initial-condition settings and the forcing function. Circuit *d* is slightly simpler than circuit *c* in that it uses only three odd-sized resistors, those at the input of the first integrator.

A common feature of all the circuits of Fig. 2.4, which may constitute a disadvantage by contrast with Fig. 2.2, is that $\ddot{x}$ does not exist in the circuit as a measurable quantity. To be sure, the three voltages whose sum is $-\ddot{x}$ (or proportional to it) do appear separately, but the processes of adding and integrating are performed in the same computer amplifier, and the output is $+\dot{x}$. If $\ddot{x}$ is required, then a fourth amplifier can be used, possibly in a circuit like that of Fig. 2.2. It is often possible to save one or more amplifiers in computer setups in which not every quantity need be formed explicitly.

In order to simplify our diagrams of computer setups, we shall henceforth use a more generalized set of symbols, often used for computers that have resistors and capacitors built in to allow a limited choice of fixed multiplying constants. Commonly such computers use 1-megohm resistors or 1-$\mu$f capacitors as feedback elements and perhaps half a dozen input resistors ranging from 1 megohm down to $\frac{1}{10}$ megohm. When these resistors are used singly, multiplying factors of 1, 2, 4, and 10 can commonly be obtained, and, when they are used in parallel, any additive combination of these factors can be obtained. One widely used commercial computer employs as input resistors three 1-megohm, two $\frac{1}{4}$-megohm, and two $\frac{1}{10}$-megohm resistors. These may be used for summing different inputs or may be connected in combinations to give

integer multiplying factors of 1 to 31. The symbols of Fig. 2.5 are commonly used for such computers. The number adjacent to each input lead indicates the factor by which that input is multiplied, and the circled number dangling from the integrator denotes the initial-value setting for the integrator output in volts.

Let us turn now to the problem of selecting the scale factors relating problem quantities to their analogs in the computer. We shall distinguish between (1) magnitude scaling, the selection of scale factors for the magnitudes of the dependent variables, and (2) time scaling, the selection of a scale factor for the independent variable. If it were not for the

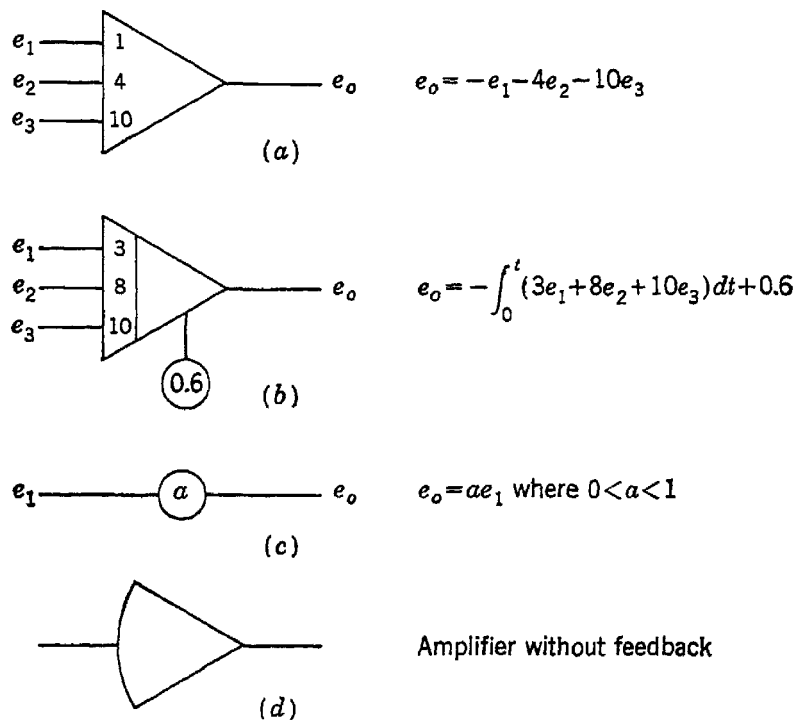



Fig. 2.5 A commonly used set of symbols

limitations imposed by the elements of the computer system, the choice of the scale factors could be completely arbitrary. Although some of these limitations will be discussed in detail in later chapters, they will be described briefly here to show their effect on scale factors. They are as follows:

1. Maximum-voltage limitation. A well designed computing amplifier has a wide dynamic range over which its output voltage is proportional to its input voltage. At the extremes of this range, however wide it may be, the proportionality fails as the amplifier saturates. Operation beyond the linear range is to be avoided because of the errors it introduces.

2. Minimum-voltage limitation. Even the best computers generate error voltages and random noise voltages. With good design these can

be kept low, but in any case the computer user should not allow any of the voltage magnitudes of the problem to be so small as to be comparable to the error and noise voltages for more than a very short time.

3. Slow-speed limitation. Slowed-down time scales require that the computer operate longer to solve a problem. Since error voltages, as well as signals, are integrated, the error tends to build up. Moreover, higher-order derivatives are small in magnitude if the problem is slowed down, and the minimum-voltage limitation comes into play.

4. High-speed limitation. At high speeds, time lags occur as a result of phase shifts in computing amplifiers, electromechanical multipliers, and recording instruments. The presence of phase shifts within the computer has the effect of displacing the roots of the characteristic equation as well as introducing spurious roots, with the result that the computer setup becomes an increasingly poorer analog of the problem being solved. Furthermore, higher-order derivatives become large in magnitude, and the maximum-voltage limitation takes effect.

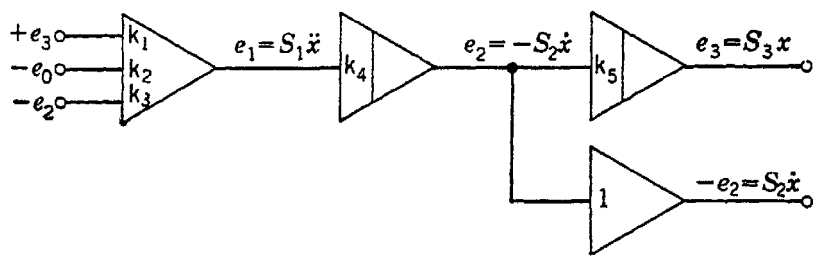



Fig. 2.6 Generalized setup for the mass-spring-damper problem

In summary, then, the voltages used must be neither too big nor too small, and the time scale must be neither too rapid nor too slow.

The maximum output voltage for linear operation of a computer amplifier usually falls within the range of 80 to 100 volts for vacuum-tube amplifiers and somewhat lower for voltage amplifiers using transistors; $\pm 100$ volts is a typical dynamic range for commercially available American computers. Scaling for a particular computer is usually done most conveniently in terms of the voltages produced in that computer. One starts from the initial rough "road map", or wiring diagram for solving the problem. Each amplifier in the circuit produces a voltage that is the analog of a quantity in the problem being solved. The selection of proportionality factors relating these voltages to problem variables is a necessary step in putting a problem on the computer. Of course, the more one knows about the range of magnitude of the variables in the problem, the easier it is to make a good selection of magnitude scale factors. However, it is unwise to spend a great deal of effort trying to determine these ranges closely. It is better practice to put the problem on the computer with roughly chosen scale factors, for then the voltages

of the various amplifiers can be measured and the initial choices of scale factors refined, if necessary.

Let us follow through a complete determination of magnitude scale factors for the mass-spring-damper system described by Eqs. (2.1) and (2.2). A setup diagram is shown in Fig. 2.6. As yet the scale factors $S_j$ and the various transfer constants $k_j$ are undetermined. To find them, we first write the equation of the computer setup

$$e_1 = -k_1 e_3 + k_2 e_0 + k_3 e_2 \tag{2.3}$$

The following relations are then substituted into this equation:

$$\begin{aligned} e_1 &= S_1 \ddot{x} \\ e_2 &= -S_2 \dot{x} \\ e_3 &= S_3 x \\ e_0 &= S_4 f(t) \end{aligned} \tag{2.4}$$

The result is

$$S_1 \ddot{x} = -k_1 S_3 x - k_3 S_2 \dot{x} + k_2 S_4 f(t)$$

or

$$\ddot{x} = -k_1 \frac{S_3}{S_1} x - k_3 \frac{S_2}{S_1} \dot{x} + k_2 \frac{S_4}{S_1} f(t) \tag{2.5}$$

For this equation to be the same as Eq. (2.2), it is necessary that

$$\frac{K}{M} = k_1 \frac{S_3}{S_1}$$

$$\frac{B}{M} = k_3 \frac{S_2}{S_1}$$

and

$$\frac{1}{M} = k_2 \frac{S_4}{S_1}$$

Thus $k_1$, $k_2$, and $k_3$ can be found as soon as the scale factors $S_j$ have been determined. $k_4$ and $k_5$ are found from the relations between $e_1$ and $e_2$ and between $e_2$ and $e_3$, which are as follows:

$$\begin{aligned} e_2 &= -k_4 \int e_1 \, dt \\ -S_2 \dot{x} &= -k_4 \int S_1 \ddot{x} \, dt \\ S_2 &= k_4 S_1 \end{aligned}$$

and

$$\begin{aligned} e_3 &= -k_5 \int e_2 \, dt \\ S_3 x &= -k_5 \int - S_2 \dot{x} \, dt \\ S_3 &= k_5 S_2 \end{aligned}$$

The next step, which is the determination of numerical values for the scale factors and transfer constants, can be carried out for any equation in the form of Eq. (2.1) if numerical values are given for the coefficients

and for the forcing function. Consider the following equation as an example:

$$45\ddot{x} + 9\dot{x} + 900x = 400 \tag{2.6}$$

i.e., let $M = 45$, $B = 9$, $K = 900$, and $f(t) = 400$. Furthermore, let us suppose that $x(0)$ is given as 2.0 and $\dot{x}(0)$ as 0. The reader is undoubtedly well aware of the formal classical methods of solution of this problem, which yield an answer in closed form. These same methods can, of course, be used to determine maximum values of $x$, $\dot{x}$, and $\ddot{x}$ from which scale factors can then be found. But solving the problem "by hand" in order to be able to put it on the computer defeats the whole purpose of using the computer, so we seek instead for easy ways of making rough estimates of the magnitudes of the variables. In making these rough estimates we may make use of previous experience, approximations, tricks, or short cuts of any kind so long as they do not do obvious violence to the problem. Generally speaking, we should choose our approximations in such a way as to make our rough estimates of the variables err on the side of being larger than they are in actuality. Thus we may find an upper bound (although not a least upper bound) for each of the variables, and if scale factors are chosen to accommodate these upper bounds, then no amplifiers will be saturated with the first choice of magnitude scale factors.

With these considerations in mind, we may note that our problem is in the form of the familiar differential equation of simple harmonic motion with damping. If we neglect the damping, we find the natural frequency $\omega_0$ to be approximately $\sqrt{900/45} \cong 4.5$ radians/sec. This lies well within the usable frequency range of most computers, so no time-scale change need be made. $x$ is initially 2 and ultimately $400/900$. Since $\dot{x}(0) = 0$, the maximum value of $x$ to be expected is 2. To be on the safe side, let us scale for $x_{max} = 2.5$. Then $\dot{x}_{max} \cong 4.5 \times 2.5$, or 11.25, which we may round off to 12 for convenience. Similarly $\ddot{x}_{max} \cong \omega^2 x_{max}$, or 50. Assuming 100 volts to be the largest voltage each of the amplifiers can produce, we choose scale factors in such a way that 100 volts corresponds to the largest value of each of the quantities in the problem. Thus

$$\begin{aligned} S_1 &= 100/50 = 2 \\ S_2 &= 100/12 = 8.33 \\ S_3 &= \frac{100}{2.5} = 40 \end{aligned} \tag{2.7}$$

The scale factor $S_4$ for the forcing function may be conveniently taken as $1/4$, making the magnitude of $e_0$ be 100 volts. The various amplifier coefficients $k_j$ can now be found as follows:

$$
\begin{aligned}
k_1 &= \frac{K}{M}\frac{S_1}{S_3} = \frac{900}{45}\frac{2}{40} = 1 \\
k_2 &= \frac{S_1}{S_4 M} = \frac{2}{\frac{1}{4} \times 45} = 0.178 \\
k_3 &= \frac{B}{M}\frac{S_1}{S_2} = \frac{9}{45}\frac{2}{8.33} = 0.048 \\
k_4 &= \frac{S_2}{S_1} = \frac{8.33}{2} = 4.17 \\
k_5 &= \frac{S_3}{S_2} = \frac{40}{8.33} = 4.80
\end{aligned}
\tag{2.8}
$$

It is of interest to note that the product $k_1k_4k_5$, which may be termed the "loop gain", is independent of the scale factors, since

$$
\begin{aligned}
k_1k_4k_5 &= \frac{K}{M}\frac{S_1}{S_3}\frac{S_2}{S_1}\frac{S_3}{S_2} \\
&= \frac{K}{M}
\end{aligned}
$$

The setup as it would appear after magnitude scaling is shown in Fig. 2.7. Note that the integrator factors 4.17 and 4.80 are obtained by the

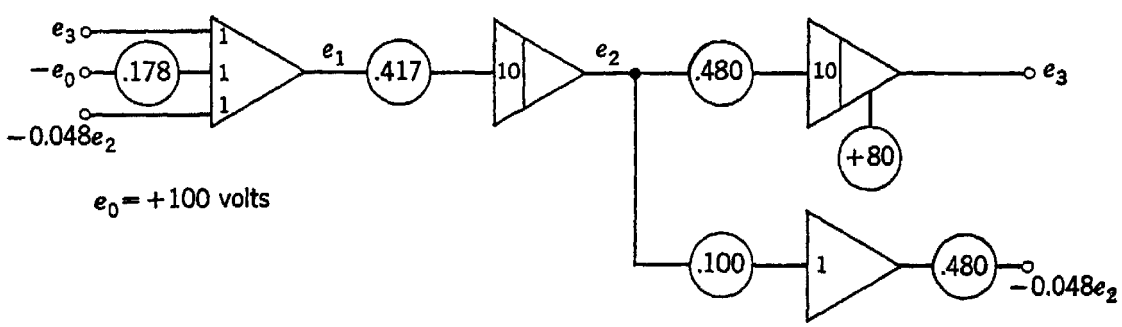



Fig. 2.7 Setup for Fig. 2.6 modified to show magnitude scaling for Eq. (2.6)

combined use of a potentiometer and an amplifier gain of 10. Also, the factor $k_3$ is obtained here not through use of a fractional gain adjustment in the summing amplifier but rather through the use of two potentiometers associated with the inverter. Two potentiometers are used to get a more accurate setting than could be obtained with only one, and they are placed on opposite sides of the inverting amplifier for isolation, since in this way the second potentiometer does not load the first.

If the computer is capable of operating at the same frequencies as those in the problem, then our scaling task is completed. However, in most cases it is necessary to adjust the rate at which the problem is solved on the computer as well as the magnitudes of the voltages generated. If the frequencies of the problem (either natural frequencies or forcing-function frequencies) lie outside the range that the computer

components can transmit faithfully, then it becomes necessary to solve the problem at a different rate on the computer. Since the elapsed time in the computer solution is then no longer the same as the elapsed time in the problem, we introduce a time scale factor $S_t$ to relate $t$, the independent variable of the problem, to $T$, the independent variable of the computer setup. Then

$$tS_t = T$$

The effect of this change upon derivatives may be seen as follows:

$$\frac{dx}{dt} = \frac{dx}{d(T/S_t)} = S_t \frac{dx}{dT}$$

and

$$\frac{d^n x}{dt^n} = S_t{}^n \frac{d^n x}{dT^n}$$

and the computer analog is slower than the original problem by a factor of $S_t$.

If no change of time scale is made, the computer is said to be solving the problem in "real time". Where the frequencies of the problem are higher than those to which the computer will respond adequately, a change in time scale is made to allow the computer to solve the problem more slowly. If the computer is used merely for solving problems, some latitude in time scaling is possible, but, if it must communicate with other equipment, as it must if it is incorporated in an airplane navigation system or in a production-process controller, then it is forced to operate in real time. The design requirements upon computers in such environments can be very severe.

In the problem under consideration, let us change the forcing function to $f(t) = 400 \sin 377t$. The frequency of 377 radians/sec is too high for the computer by a factor of about 50. To slow the solution down by a factor of 50, we let $S_t = 50$. From the electrical point of view it is evident that, if all integrators have their coefficients reduced by a factor $S_t$, then the computer solution will be slowed down by a factor $S_t$, since all integrating capacitors will charge $1/S_t$ times as fast as before. Forcing functions should also have their frequencies reduced by a factor $S_t$ and their magnitudes adjusted as required by the system equations. Thus Eq. (2.5) becomes

$$\frac{d^2x}{dT^2} = -\frac{1}{2{,}500} k_1 \frac{S_3}{S_1} x - \frac{1}{50} k_3 \frac{S_2}{S_1} \frac{dx}{dT} - \frac{1}{2{,}500} k_2 \frac{S_4}{S_1} \times 100 \sin \frac{377}{50} T \quad (2.9)$$

where $\dot{x}$ and $\ddot{x}$, denoting differentiation with respect to $t$, have been replaced by $dx/dT$ and $d^2x/dT^2$, denoting differentiation with respect to the scaled time variable $T$.

Since

$$\frac{S_3}{S_1} = k_4 k_5$$

and

$$\frac{S_2}{S_1} = k_4$$

this equation may be rewritten in the form

$$\frac{d^2x}{dT^2} = -k_1 \frac{k_4}{50}\frac{k_5}{50} x - k_3 \frac{k_4}{50}\frac{dx}{dT} - k_2 \frac{S_4}{S_1}\frac{1}{25} \sin \frac{377}{50} T \qquad (2.10)$$

A setup for this problem that includes both magnitude scaling and time scaling appears in Fig. 2.8. [The reader should satisfy himself that magnitude scale factors chosen with $f(t) = 400$ will still be satisfactory for $f(t) = 400 \sin 377t$.]

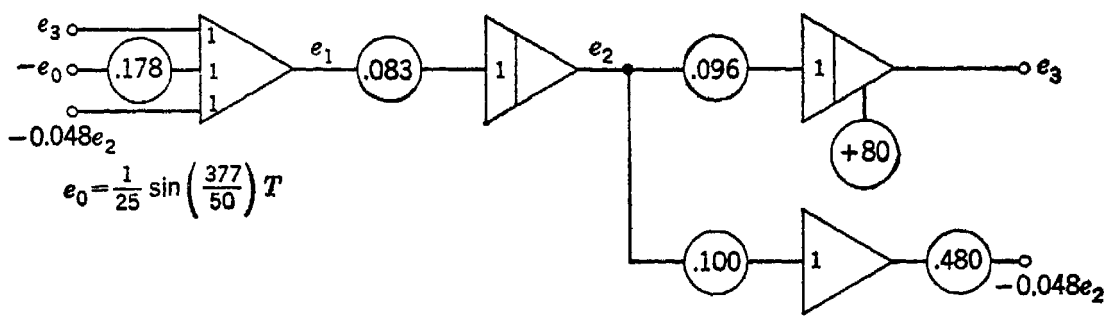



Fig. 2.8 Setup for solving $45\ddot{x} + 9\dot{x} + 900x = 400 \sin 377t$, showing both magnitude scaling and time scaling

It is sometimes possible to choose time scale factors in such a way that all time constants in the scaled system are of the order of 1 sec. To accomplish this it is necessary to be able to adjust the time scaling so that the coefficients of each order of derivative in the scaled equation are approximately the same magnitude. As an example consider the equation

$$\ddot{x} + 10\dot{x} + 100x = f(t) \qquad (2.11)$$

If we let $S_t = 10$, we have

$$\dot{x} = 10 \frac{dx}{dT}$$

and

$$\ddot{x} = 100 \frac{d^2x}{dT^2}$$

Thus,

$$100 \frac{d^2x}{dT^2} + 100 \frac{dx}{dT} + 100x = f\left(\frac{T}{10}\right)$$

$$\frac{d^2x}{dT^2} + \frac{dx}{dT} + x = \frac{1}{100} f\left(\frac{T}{10}\right)$$

The 10:1 time-scale change has reduced both the natural frequencies of the system and the frequency of the forcing function by a factor of 10.

The general procedure consists in determining $m$, the number of orders by which the highest-order derivative and the lowest-order derivative differ, and $n$, the number of orders of magnitude by which the coefficient of the lowest-order term exceeds that of the highest-order term. The scale factor $S_t$ is then given by

$$S_t = 10^{n/m}$$

In the example of Eq. (2.11), $m = 2$ and $n = 2$, giving $S_t = 10$ as an appropriate time scale factor.

As another example, consider a mass-spring-damper system having the differential equation

$$0.08\ddot{x} + 0.32\dot{x} + 7.5x = 0$$

To choose a time scale, observe that the second-order-derivative term is approximately two orders of magnitude smaller than the 0th-order term. Therefore, a 10:1 slowdown will produce time constants of approximately 1 sec, or system frequencies of about 1 radian/sec. Thus

$$8.0\frac{d^2x}{dT^2} + 3.2\frac{dx}{dT} + 7.5x = 0$$

$$\frac{d^2x}{dT^2} = -0.4\frac{dx}{dT} - 0.9375x \tag{2.12}$$

If the coefficients of derivatives of all orders can be made to be of the same order of magnitude by a change of time scale, then magnitude scaling need be done only on the dependent variable and not upon its derivatives. This is so because the natural frequencies of the scaled system are then approximately 1 radian/sec, and all derivatives are approximately the same size as the dependent variable. For instance, in the present problem, let the initial value of $x$ be 10 and the initial value of $\dot{x}$ be 0. Then the machine variable $X$ is related to the problem variable $x$ by

$$X = 10x$$

Thus we obtain

$$\frac{d^2(X/10)}{dT^2} = -0.4\frac{d(X/10)}{dT} - 0.9375\frac{X}{10}$$

or

$$\frac{d^2X}{dT^2} = -0.4\frac{dX}{dT} - 0.9375X \tag{2.13}$$

It will be noted that Eq. (2.13) is the same as Eq. (2.12). This is so because we have, in effect, applied the same scale factor to the dependent variable and all its derivatives. This type of magnitude scaling is adjusted merely by changing the initial-condition settings of the integrators and the magnitudes of the forcing functions. Thus the appearance of a linear homogeneous differential equation is unaffected by

changes of magnitude scale. A setup diagram for this problem appears in Fig. 2.9.

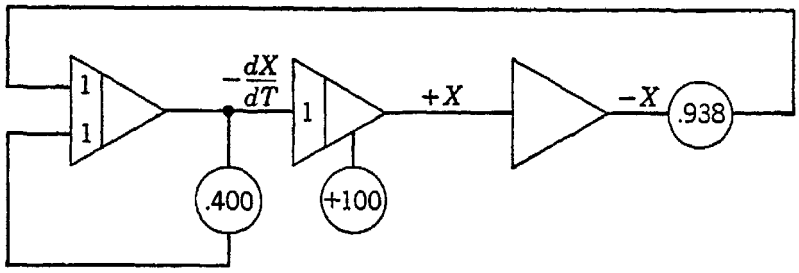



Fig. 2.9 Setup for Eq. (2.13)

**Potentiometer loading.**[3,4] Although a more detailed discussion of the electrical characteristics of computer components is presented in Chaps. 3 and 4, it is appropriate to observe at this point that merely setting the dial of a potentiometer to the value shown by the symbol on the setup diagram does not ensure that the desired ratio will be obtained in the actual setup. This is so because the potentiometer tap supplies current to other parts of the computer, and therefore the current through the upper part of the potentiometer is greater than that in the lower part. Consequently, the output voltage is a smaller fraction of the input voltage than is indicated by the dial setting.

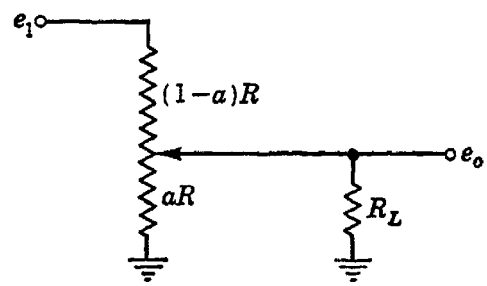



Fig. 2.10 Circuit to show effect of potentiometer loading

Consider the loaded potentiometer of Fig. 2.10. Although the potentiometer is set to $a$, the ratio $e_o/e_1$ differs from $a$ because of the loading introduced by $R_L$. If we wish to set $e_o/e_1 = k$, we calculate the proper value of $a$ to be used as follows:

$$k = \frac{aRR_L/(aR + R_L)}{aRR_L/(aR + R_L) + (1 - a)R}$$

from which we find that

$$a^2 + a\left(\frac{R_L}{kR} - 1\right) - \frac{R_L}{R} = 0$$

$$a = \frac{1}{2}\left[1 - \frac{R_L}{kR} + \sqrt{\left(1 - \frac{R_L}{kR}\right)^2 + 4\frac{R_L}{R}}\right]$$

Evidently the loading effect is minimized by keeping $R_L$ large. In cases where it is necessary for the potentiometer to drive many input resistors to computing amplifiers, the effective $R_L$ may become very low; it is then desirable to isolate the potentiometer from its loads by inclusion

of an amplifier between them, if the sign inversion can be accommodated without too much difficulty. The other way to reduce the loading effect is to let $R$, the potentiometer resistance, be small. This, however, places a heavy load on the sources supplying the potentiometer. From this standpoint, a high-resistance potentiometer would be desirable. A compromise value, in the range of 20,000 to 50,000 ohms, is usually used.

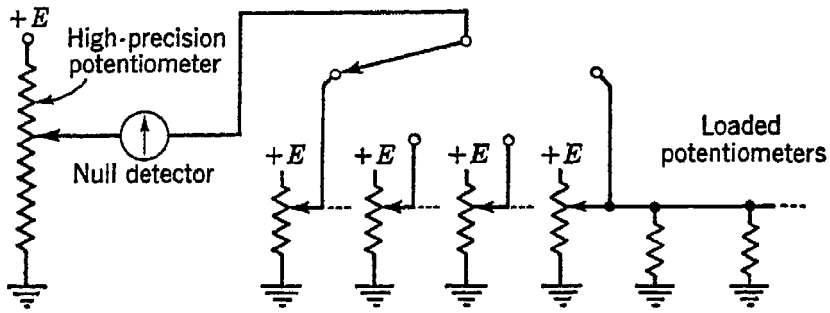



Fig. 2.11 A method of setting potentiometers

In many computers the calculation of the proper dial setting for the potentiometers is obviated by the provision of a single high-precision potentiometer against which the variously loaded potentiometers of the setup can be balanced one at a time, as in Fig. 2.11. The desired value of the transfer ratio is first set on the high-precision potentiometer, and then the loaded potentiometer is adjusted until the null detector is at zero.

**Minimal integrator setups.** So far we have said nothing about attempting to minimize the number of components used in a computer setup. Of course, economy alone is often a sufficient justification for using the simplest possible circuit, but there are other reasons for avoiding more complex circuits. The principal trouble is the error that occurs in larger systems. One source of such errors is the accumulation of phase shifts and drifts that results from assembling a large number of components. A more subtle source of error is occasionally introduced by using more than the required minimum number of integrators, as will be seen from the example that follows.

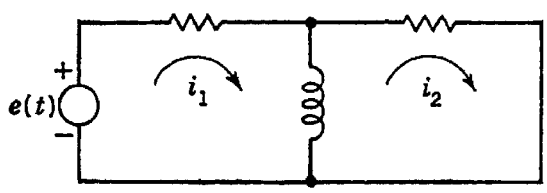



Fig. 2.12 A two-mesh network

Consider the electrical network of Fig. 2.12, in which, to simplify our work, all circuit elements have a magnitude of 1 unit. The differential equations for the mesh currents are

$$e(t) = \left(\frac{di_1}{dt} + i_1\right) - \frac{di_2}{dt}$$
$$0 = -\frac{di_1}{dt} + \left(\frac{di_2}{dt} + i_2\right) \tag{2.14}$$

By proceeding in the usual manner we obtain the setup diagram of Fig. 2.13, which requires two amplifiers for integrating, two for summing, and two for inverting. If the integrating amplifiers are identical and all components are ideal, this setup will indeed be a true analog of the network of Fig. 2.12. However, if the two integrators are not identical, an interesting complication arises, since the characteristic equation for the computer setup is different from the characteristic equation of the original network. Writing Eqs. (2.14) in Laplace-transform notation

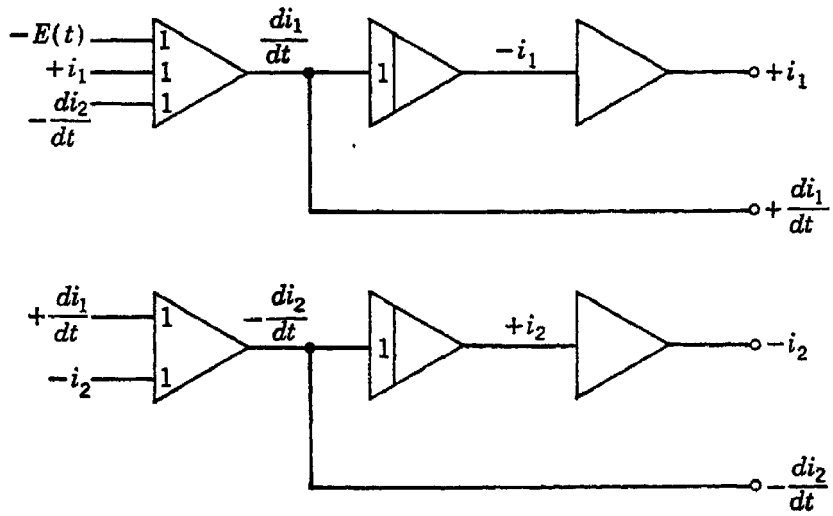



Fig. 2.13 Setup diagram for network of Fig. 2.12

enables us to write the characteristic equation of the network by setting the determinant of the coefficients equal to zero:

$$\left.\begin{aligned} E(s) &= (s+1)I_1(s) - sI_2(s) \\ 0 &= -sI_1(s) + (s+1)I_2(s) \end{aligned}\right\} \rightarrow \begin{vmatrix} s+1 & -s \\ -s & s+1 \end{vmatrix} = 0 \qquad (2.15)$$

from which Eq. (2.16) follows:

$$2s + 1 = 0 \qquad (2.16)$$

Now let us suppose that the integrator in the $i_1$ chain of Fig. 2.13 has a multiplying factor of $1/k$ instead of 1. As the reader may verify, the equations of this altered setup yield the following characteristic equation:

$$\begin{vmatrix} 1+ks & -s \\ -ks & 1+s \end{vmatrix} = 0$$
$$s(k+1) + 1 = 0 \qquad (2.17)$$

If $k = 1$, the computer setup yields the same characteristic equation as the network, but, if $k \neq 1$, the transient response will be

$$i = K\epsilon^{-t/(k+1)}$$

instead of $\quad i = K\epsilon^{-t/2}$

This difference arises, of course, from the fact that the computer setup employs two integrators to perform a function carried out in the network with only one. The total inductor current $i_1 - i_2$ equals the integral of one quantity, the voltage across the inductor. In the computer, however, $i_1$ is produced by one integrator and $i_2$ by another. If these are not perfectly matched, $i_1 - i_2$ will not be correct. It is as if the current in loop 1 produced in loop 2 a voltage different from that which an equal current in loop 2 would produce in loop 1 or, in other words, as if the inductance element were no longer bilateral. For suppose a current $i_1$ produces in the inductance a voltage of $k(di_1/dt)$ but that $i_2$ produces $di_2/dt$. The network equations become

$$\begin{aligned} E(s) &= (1 + ks)I_1(s) - sI_2(s) \\ 0 &= -ksI_1(s) + (s + 1)I_2(s) \end{aligned} \tag{2.18}$$

and the characteristic equation is

$$\begin{vmatrix} 1 + ks & -s \\ -ks & s + 1 \end{vmatrix} = s(k + 1) + 1 = 0 \tag{2.19}$$

which is the characteristic equation of the two-integrator computer setup.

Mere dislocation of the roots of the characteristic equation, however troublesome it may be, is likely to occur even in the optimum computer setup as a result of component inaccuracies. These component inaccuracies become even more troublesome, however, in setups that employ redundant integrators, since in such cases there is a very real possibility that the degree of the characteristic equation may be increased, with the result that not only are the desired roots moved but new ones are introduced. As an example, suppose that, in the setup of Fig. 2.13, the summing resistor to which $-di_2/dt$ is fed at the input of the upper chain is of such a value that this input is multiplied by a factor $k$ instead of by unity. Then

$$\begin{aligned} sI_1(s) &= -I_1(s) + ksI_2(s) + E(s) \\ -sI_2(s) &= -sI_1(s) + I_2(s) \end{aligned}$$

are the system equations, from which it follows that the characteristic equation is

$$\begin{vmatrix} s + 1 & -ks \\ -s & s + 1 \end{vmatrix} = s^2(1 - k) + 2s + 1 = 0 \tag{2.20}$$

The roots of this equation are

$$s = \frac{-1 \pm \sqrt{k}}{1 - k}$$

for $k \neq 1$. If $k$ is precisely unity, then no trouble arises, since the char-

acteristic equation is of the first degree, with its root where it belongs, at $s = -\frac{1}{2}$. If $k$ is slightly less than unity, the trouble is not severe, since one of the two roots is a large negative number, corresponding to an exponential term that diminishes very rapidly and can hence be neglected, and the other root lies very close to the desired location $s = -\frac{1}{2}$ since

$$\frac{-1+\sqrt{k}}{1-k} = -\frac{1-\sqrt{k}}{(1+\sqrt{k})(1-\sqrt{k})} = -\frac{1}{1+\sqrt{k}} \cong -\tfrac{1}{2}$$

As $k$ decreases toward zero, these two roots approach each other and merge at $s = -1$. If $k$ could go negative (perhaps because of some gross blunder in connecting up the system), the two roots would become complex conjugates and the system would display a damped oscillatory response.

Values of $k$ substantially less than unity are improbable, but $k$ is as likely to be slightly larger as to be slightly smaller than unity. The roots of the second-degree equation are then found to be (1) a large positive number, and (2) a number slightly to the right of $s = -\frac{1}{2}$. The large positive root has a disastrous effect on the computer solution, since it corresponds to an extremely rapidly growing exponential term. Instabilities such as this have often been encountered in practice,[5] and a number of only partially satisfactory artifices have been used to reduce their effects. Such tricks as the introduction of judiciously placed additional damping to counteract the effect of spurious roots fail to recognize that the additional roots are present because the computer setup uses a number of energy-storage elements (the integrators) that exceeds the degree of the characteristic equation of the system being simulated. It is, therefore, possible for the computer setup to be unstable, although, as our examples showed, it need not necessarily be unstable.

The problem then reduces to a determination of the number of integrators to be used. If the characteristic equation of the simulated system is known, the number of integrators to be used is equal to its degree. Determination of the characteristic equation is laborious, however, and we prefer merely to determine its degree without being obliged to write the entire equation. The following steps enable us to do this:

1. Count the total number of energy-storage elements (inductances and capacitances) in the system.
2. Subtract from this count the total number of

   *a.* Independent cut-sets formed solely by capacitances
   *b.* Independent cut-sets formed solely by inductances
   *c.* Independent tie-sets formed solely by capacitances
   *d.* Independent tie-sets formed solely by inductances

The result of these two steps is the number of independent energy-storage elements that the simulated system possesses.

The terms "cut-set" and "tie-set" may be unfamiliar to the reader, since they are rather new even to circuit theorists. Briefly, a tie-set is any set of branches forming a closed path within the network, and a cut-set is any set of branches that divides the network into two parts when all the branches of the set are cut. These terms are discussed in greater detail by Guillemin.[6]

As an example, consider the network of Fig. 2.14. This circuit has ten energy-storage elements, but, since it has a cut-set formed of inductances, a tie-set of inductances, and a tie-set of capacitances, only seven

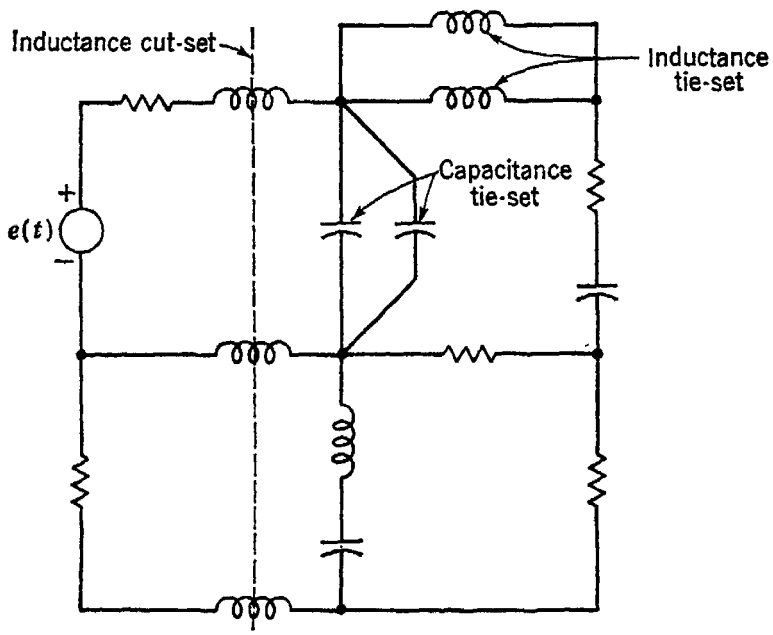



Fig. 2.14 A network having a seventh-degree characteristic equation

of the ten can be regarded as independently contributing to the degree of the characteristic equation. Some other examples are shown in Fig. 2.15. In Fig. 2.15*a*, there is a loop formed by capacitances, and there is also a cut-set formed by capacitances, so the degree of the characteristic equation is 2 less than the number of energy-storage elements. The circuit of Fig. 2.15*b* has a capacitance cut-set, an inductance cut-set, and an inductance tie-set. Since it has eleven energy-storage elements, the degree of its characteristic equation is $11 - 3 = 8$.

Although this procedure can be applied to any network, in practice its application may sometimes be simplified by preliminary modification of the circuit. It sometimes happens that a branch is present that contains more than one inductance or more than one capacitance in series. Likewise, there may be a node pair between which are connected in parallel more than one inductance or more than one capacitance. Such series or parallel combinations of like elements may advantageously be com-

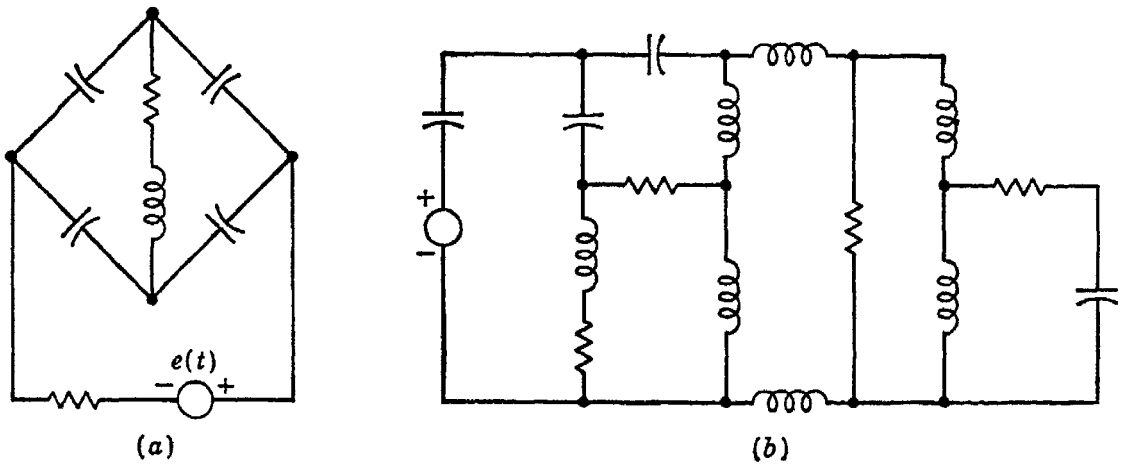



Fig. 2.15 (*a*) A network having a third-degree characteristic equation, (*b*) a network having an eighth-degree characteristic equation

bined into single elements before the count is made to determine the degree. For example, the circuits on the left in Fig. 2.16 may be modified to the forms on the right before making the count. In the circuit of Fig. 2.16*a*, the loop formed by $L_1$ and $L_2$ is fairly obvious, but the cut-sets formed by $S_1$ and $S_2$ or by $S_1$ and $S_3$ are less apparent, and the count is

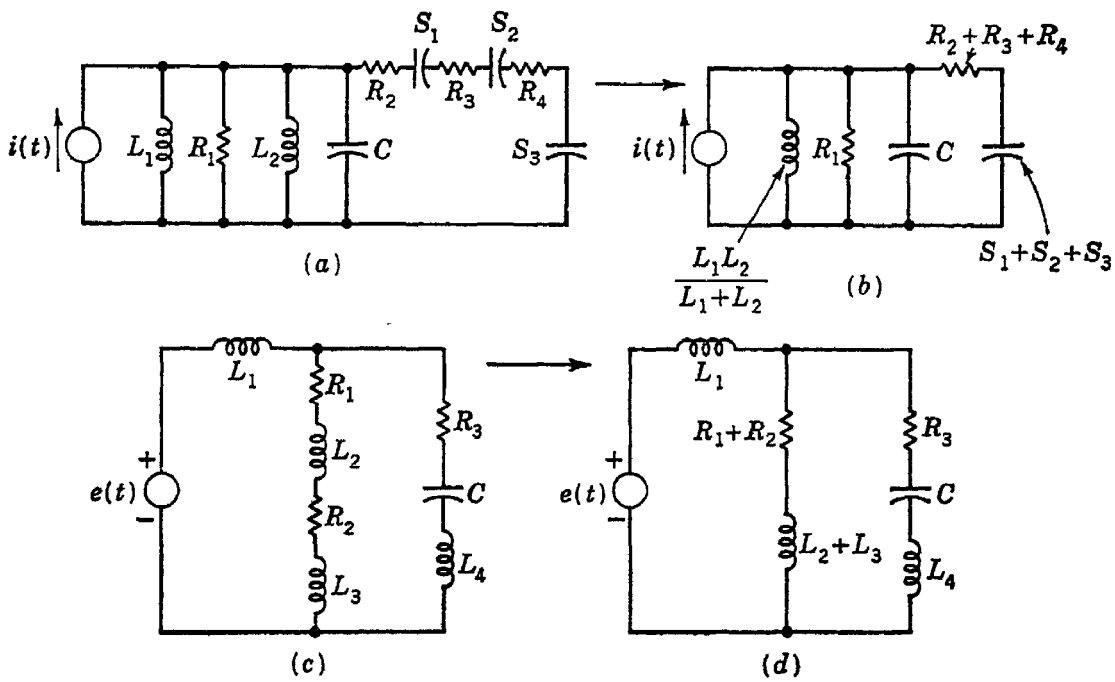



Fig. 2.16 Network modifications before determining reactive-element count

more easily made on the simplified circuit in Fig. 2.16*b*. Similarly, in Fig. 2.16*c*, the cut-set formed by $L_2$ and $L_3$ is not obvious, and Fig. 2.16*d* is easier to analyze. (Of course, the circuit simplification is superfluous if one is careful to use in each tie-set or cut-set at least one circuit element not used in any previously counted set. Thus in Fig. 2.16*a*, $S_1$ and

$S_2$ form a capacitance cut-set, $S_1$ and $S_3$ form another, and $S_2$ and $S_3$ form a third. Only two of these contribute to the count, however, since the third is not independent. Similarly, in Fig. 2.16*c*, $L_1$, $L_2$, and $L_4$ form a cut-set; so do $L_1$, $L_3$, and $L_4$; and $L_2$ and $L_3$. Here again, only two of the three cut-sets are independent.)

One question remains: Now that we know the degree of the characteristic equation, how shall we set up the computer to use no more than this number of integrators? The answer should by now be evident. Since, in Fig. 2.12, the single current through the inductor was replaced by two currents, $i_1$ and $i_2$, the single voltage drop was replaced by the sum $L(di_1/dt) - L(di_2/dt)$, and two integrators instead of one were required in the computer setup. Had a single current $i_3 = i_1 - i_2$ been used instead, the voltage term would have been $L(di_3/dt)$, and only a single integrator would have been used. Generalizing, then, it may be said that, if the equations for the system are written on a mesh basis, the currents through any reactive elements common to two or more meshes should be treated as branch currents rather than as sums of mesh currents. Likewise, if the equations are written on a node basis, the voltage across a reactive element common to two node pairs should be given an identity of its own rather than being written as the difference of two node-pair voltages. In addition it can be seen that each inductance cut-set or capacitance cut-set gives rise to a current equation that expresses the current in one element of the set as the sum of those in all the others. Each inductance tie-set or capacitance tie-set gives rise to an equation giving the voltage across one element of the set as the sum of the voltages across all the others. Each such algebraic equation thus permits us to use a summing amplifier instead of an integrator to find the current or voltage of one energy-storage element.[7] Figure 2.12 is redrawn in this light in Fig. 2.17, and the resulting equations are

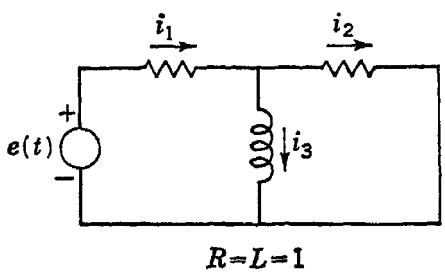



Fig. 2.17 Branch-current version of Fig. 2.12

$$e(t) = i_1 + \frac{di_3}{dt} \tag{1}$$

$$0 = i_2 - \frac{di_3}{dt} \tag{2}$$

$$i_1 = i_2 + i_3 \tag{3}$$

The setup diagram of Fig. 2.18 follows readily. This setup is not only incapable of producing spurious roots but also, since it contains only four amplifiers instead of the six used in Fig. 2.13, much less likely to produce erroneous responses due to root dislocation.

Occasionally it may happen that the minimum integrator circuit satisfies the differential equation and yet produces only derivatives of the desired results. In this case integrators may be added to integrate the derivatives, without endangering the circuit stability. For example, consider the equations

$$\begin{aligned} \ddot{x}_1 + \dot{x}_1 + (x_1 - x_2) &= f(t) \\ \ddot{x}_2 + \dot{x}_2 + (x_2 - x_1) &= 0 \end{aligned} \tag{2.21}$$

A circuit with only three integrators can be evolved if it is noted that $x_1 - x_2$ occurs in each equation. Thus we may integrate $\dot{x}_1 - \dot{x}_2$ in one

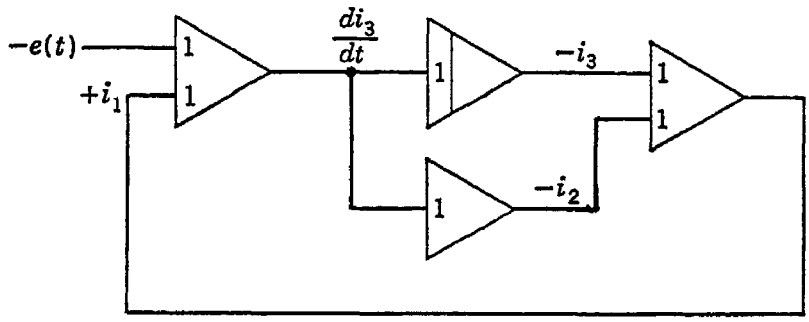



Fig. 2.18 Single-integrator computer setup for Fig. 2.17

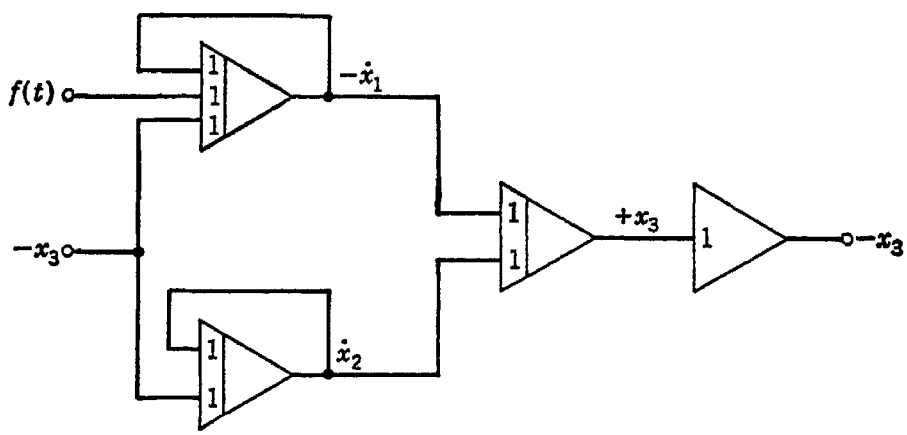



Fig. 2.19 A computer setup for Eqs. (2.21)

integrator instead of separately integrating $\dot{x}_1$ and $\dot{x}_2$ and then forming their difference.

Let $\quad x_1 - x_2 = x_3$

Then, $\quad \ddot{x}_1 = -\dot{x}_1 - x_3 + f(t)$

$\quad -\ddot{x}_2 = \dot{x}_2 - x_3$

The setup diagram of Fig. 2.19 follows. This circuit uses only three integrators, as it should, but does not produce either $x_1$ or $x_2$ as an output. These could be formed either by adding two more integrators to integrate $-\dot{x}_1$ and $\dot{x}_2$ or by adding one integrator to form $+x_1$ and a summing amplifier to combine this with $-x_3$ to form $-x_2$. Although such a circuit would then have more than the minimum number of inte-

grators, these would be used in a way that would not endanger the stability of the setup, since the extra integrators would not be part of the basic closed loop needed to satisfy the differential equation.

It may occasionally happen that a set of differential equations can be satisfied by a computer setup employing fewer integrators than the degree of the characteristic equation. In such a case it is usually found that more integrators must be added, as in the circuit of Fig. 2.19, to produce all the desired outputs. If fewer integrators suffice, this is satisfactory. The important criterion is that the number of integrators within the closed-loop part of the computer setup should not exceed the degree of the characteristic equation. As an example of this situation, Eqs. (2.21) may be added to give

$$(\ddot{x}_1 + \ddot{x}_2) + (\dot{x}_1 + \dot{x}_2) = f(t)$$

for which the setup of Fig. 2.20 may be used. Of course, it may reasonably be argued that this setup does not really satisfy Eqs. (2.21), since it does not produce directly either $x_1$ or $x_2$ or any of their derivatives.

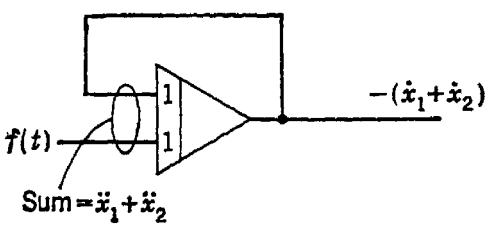



Fig. 2.20 A single-integrator setup derived from Eqs. (2.21)

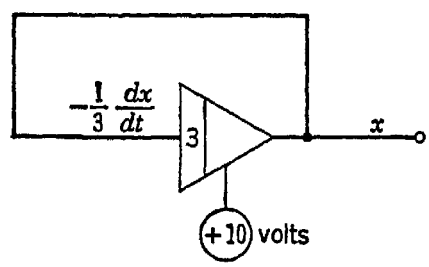



Fig. 2.21 A setup to generate $x = \epsilon^{-3t}$

**Using the computer to generate simple forcing functions.** It frequently happens that a problem to be solved on the computer requires forcing functions not readily available in the laboratory. Such functions as exponentials, ramps, parabolic functions, hyperbolic functions, low-frequency sinusoids, and others can easily be produced by auxiliary computer setups. The general procedure to be followed in deriving the appropriate setup is to differentiate the desired function repeatedly, seeking a differential equation relating higher-order to lower-order derivatives. For example, to produce

$$x = 10\epsilon^{-3t} \text{ volts}$$

we form

$$\frac{dx}{dt} = -30\epsilon^{-3t}$$

and thus

$$\frac{dx}{dt} = -3x$$

The setup diagram of Fig. 2.21 follows. The setups of Fig. 2.22 can be derived in the same way. No differential equation is needed for 2.22*a*

and $b$, of course, since these circuits do not involve a closed loop. In practice the circuits of 2.22$c$ and $d$ would employ double-gang potentiometers for setting the factor $a$ in the two amplifiers; $a$ would then be easily adjustable and directly readable.

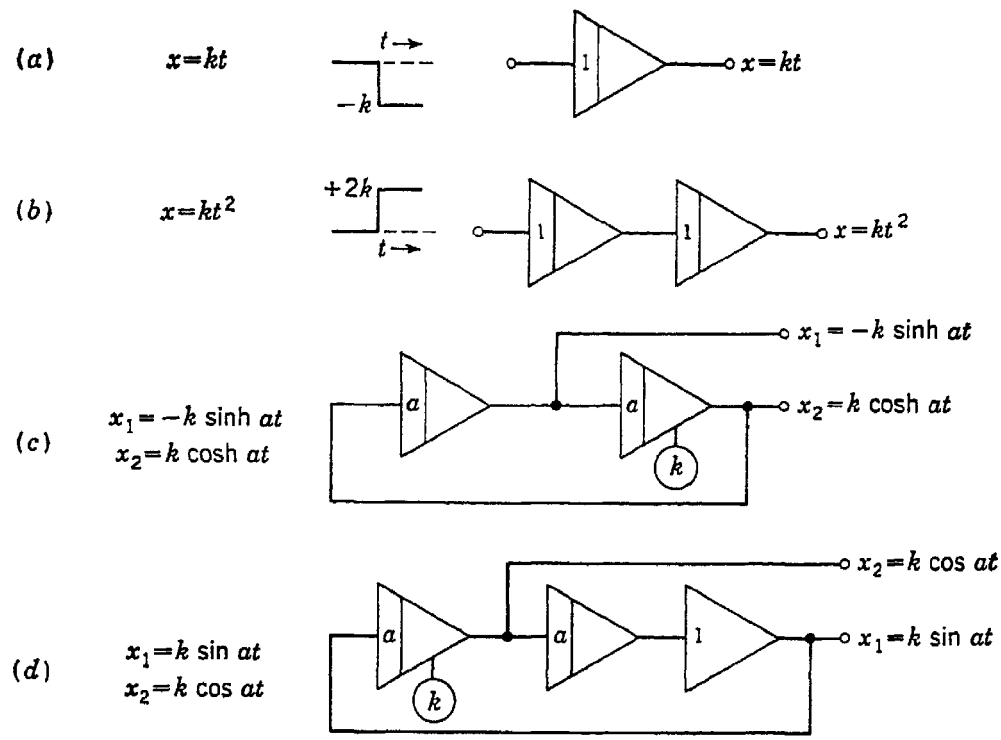



Fig. 2.22 Some simple functions which can be generated with linear computing components

**Generation of transfer functions using computer components.** The analog computer is sometimes used for problems other than solving a set of differential equations. Computer components can be combined to form a wide range of transfer characteristics, and the resulting systems, together with appropriate input and output conversion devices, can be incorporated into nonelectronic environments. Computing elements have been used to form special transfer functions for such applications as oil-refinery process control, machine-tool control, and autopilot guidance of aircraft.

There are two ways to use computing components to form transfer functions. One is to use a single amplifier with complex $RC$ networks for its feedback path and its input path. Proper choices for these networks and for the magnitudes of their components permit formation of a wide variety of transfer functions. The other is to use amplifiers connected simply as integrators, adders, or inverters, but to combine them with potentiometers in a network that yields the desired transfer function. The first method is more economical in its use of computing ampli-

fiers, but in practice is often more difficult to use because of the wide range of values of resistance and capacitance that must be available if more than a very few transfer functions are to be realized. Catalogues of circuits formed in each of these ways have been published.[8,9] A few representative samples of each will be considered here.

Following Godet,[8] we note that an amplifier with generalized input and feedback networks can be represented as in Fig. 2.23. In any practical system both $e_g$ and $i_g$ closely approximate zero, so we may write

$$\begin{aligned} e_1 &= i_1 Z_1 \\ e_o &= i_o Z_0 \\ i_1 &= -i_o \end{aligned} \tag{2.22}$$

and $$\frac{e_o}{e_1} = -\frac{Z_0}{Z_1} \quad \text{or} \quad \frac{E_o(s)}{E_1(s)} = -\frac{Z_0(s)}{Z_1(s)}$$

where $Z_0$ and $Z_1$ are the short-circuit transfer impedances of the two networks, that is, the ratios of input voltage to output current with the

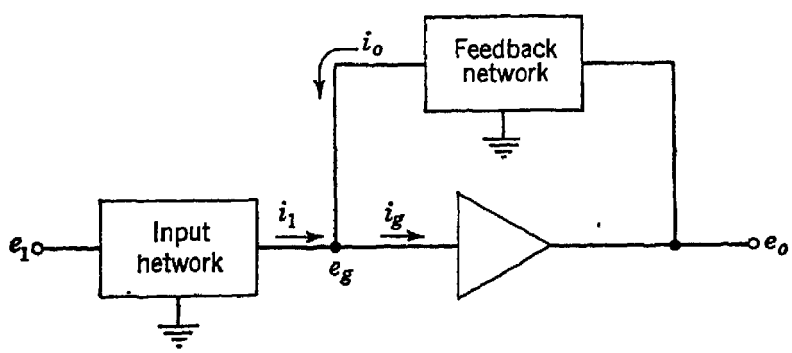



Fig. 2.23 Relations in circuit comprising a high-gain amplifier with input and feedback elements

output short-circuited. Determining the transfer function for a given $Z_1$ and $Z_0$ is simple and straightforward, but the converse problem, that of determining the $Z_1$ and $Z_0$ networks to give a desired transfer function, is a typical synthesis problem requiring the use of considerable engineering skill. Fortunately, this exercise is obviated in many cases by the existence of extensive tabulations of the short-circuit transfer impedances of simple $RC$ networks, and from these a suitable choice can often be made to yield the desired transfer function.

A partial list of three- and four-element impedance functions is given in Table 2.1. These can be used to determine a network to produce a desired transfer function as follows:

1. Rewrite the desired transfer function $e_O/e_1$ in the form $-Z_0/Z_1$ where $Z_0$ and $Z_1$ are each in a form corresponding to one of the short-circuit transfer impedances of Table 2.1.

2. Use the network corresponding to $Z_0$ as the feedback path and the network corresponding to $Z_1$ as the input path. Adjust magnitudes of the $R$'s and $C$'s according to the last column.

As an example, suppose it is desired to form the transfer function

$$\frac{E_o(s)}{E_1(s)} = -\frac{3 + 6s}{s + s^2}$$

After some manipulating, this becomes

$$\frac{E_o(s)}{E_1(s)} = -\frac{3(1 + 2s)}{s(1 + s)} = -\frac{(1 + 2s)/(s/3)}{1 + s}$$

from which the complete network is readily found to be as shown in Fig. 2.24.

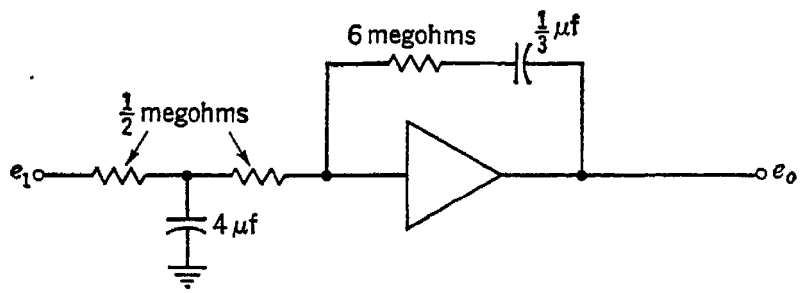



Fig. 2.24 Network to realize transfer function $\frac{E_o(s)}{E_1(s)} = -\frac{3 + 6s}{s + s^2}$

In the second way of synthesizing a transfer function with computing components,[9] the desired relation of output and input quantities is expressed as a differential equation, and a computer setup implementing it is then evolved in the usual way. Consider again the transfer function

$$\frac{E_o(s)}{E_1(s)} = -\frac{3 + 6s}{s + s^2}$$

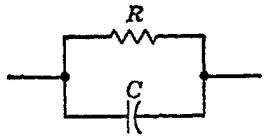
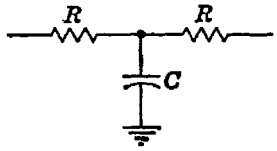
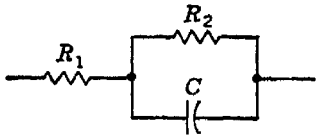
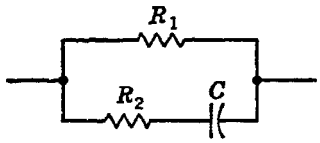
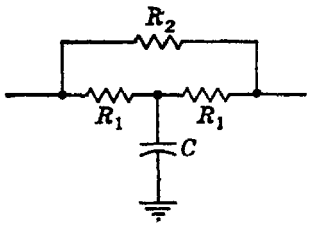
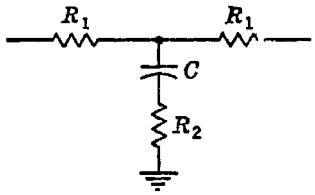
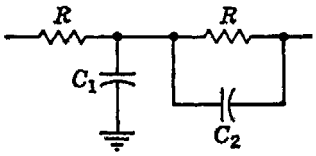
This is rewritten

$$\begin{aligned}(s + s^2)E_o(s) &= -(3 + 6s)E_1(s)\\ s^2E_o(s) &= -(3 + 6s)E_1(s) - sE_o(s)\\ E_o(s) &= -\left(\frac{3}{s^2} + \frac{6}{s}\right)E_1(s) - \frac{1}{s}E_o(s)\end{aligned}$$

from which the setup of Fig. 2.25 follows.

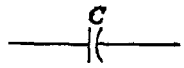
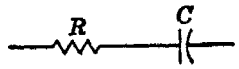
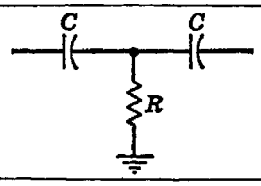
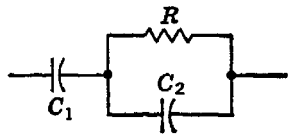
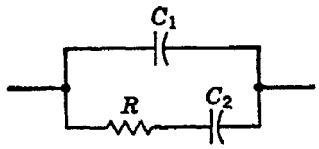
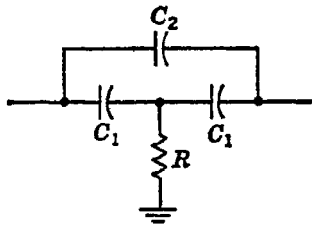
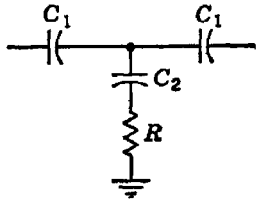
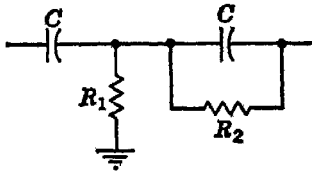
Although this method requires more amplifiers (and usually more resistors and capacitors), it has the advantage of being much more flexible. Many more derivative quantities are formed explicitly in setups of this type and are available to be recorded or otherwise used. Of course, outputs can also be taken from various points in the feedback and input

**Table 2.1***

| Transfer impedance function | Network | Relations | Inverse relations |
|---|---|---|---|
| $A(s)$ | | $A = R$ | $R = A$ |
| $\dfrac{A}{1 + sT}$ |  | $A = R$<br>$T = RC$ | $R = A$<br>$C = \dfrac{T}{A}$ |
| $A(1 + sT)$ |  | $A = 2R$<br>$T = \dfrac{RC}{2}$ | $R = \dfrac{A}{2}$<br>$C = \dfrac{4T}{A}$ |
| $A\dfrac{1 + s\theta T}{1 + sT}$<br>$\theta < 1$ |  | $A = R_1 + R_2$<br>$T = R_2C$<br>$\theta = \dfrac{R_1}{R_1 + R_2}$ | $R_1 = A\theta$<br>$R_2 = A(1 - \theta)$<br>$C = \dfrac{T}{A(1 - \theta)}$ |
| |  | $A = R_1$<br>$T = (R_1 + R_2)C$<br>$\theta = \dfrac{R_2}{R_1 + R_2}$ | $R_1 = A$<br>$R_2 = \dfrac{A\theta}{1 - \theta}$<br>$C = \dfrac{T(1 - \theta)}{A}$ |
| $A\dfrac{1 + sT}{1 + s\theta T}$<br>$\theta < 1$ |  | $A = \dfrac{2R_1R_2}{2R_1 + R_2}$<br>$T = \dfrac{R_1C}{2}$<br>$\theta = \dfrac{2R_1}{2R_1 + R_2}$ | $R_1 = \dfrac{A}{2(1 - \theta)}$<br>$R_2 = \dfrac{A}{\theta}$<br>$C = \dfrac{4T(1 - \theta)}{A}$ |
| |  | $A = 2R_1$<br>$T = \left(R_2 + \dfrac{R_1}{2}\right)C$<br>$\theta = \dfrac{2R_2}{2R_2 + R_1}$ | $R_1 = \dfrac{A}{2}$<br>$R_2 = \dfrac{A\theta}{4(1 - \theta)}$<br>$C = \dfrac{4T(1 - \theta)}{A}$ |
| |  | $A = 2R$<br>$T = \dfrac{R}{2}(C_1 + C_2)$<br>$\theta = \dfrac{2C_2}{C_1 + C_2}$ | $R = \dfrac{A}{2}$<br>$C_1 = \dfrac{2T(2 - \theta)}{A}$<br>$C_2 = \dfrac{2T\theta}{A}$ |

* Portions of Table of Transfer Impedances for Operational Amplifiers from F. R. of the publisher, McGraw-Hill

**Table 2.1 (*Continued*)**

| Transfer impedance function | Network | Relations | Inverse relations |
|---|---|---|---|
| $\frac{1}{sB}$ |   | $B = C$ | $C = B$ |
| $\frac{1}{sB}(1 + sT)$ |   | $B = C$<br>$T = RC$ | $R = \frac{T}{B}$<br>$C = B$ |
| $\frac{1}{sB}\frac{1 + sT}{sT}$ |   | $B = \frac{C}{2}$<br>$T = 2RC$ | $R = \frac{T}{4B}$<br>$C = 2B$ |
| $\frac{1}{sB}\frac{1 + sT}{1 + s\theta T}$<br>$\theta < 1$ |   | $B = C_1$<br>$T = R(C_1 + C_2)$<br>$\theta = \frac{C_2}{C_1 + C_2}$ | $R = \frac{T(1 - \theta)}{B}$<br>$C_1 = B$<br>$C_2 = \frac{B\theta}{1 - \theta}$ |
| |   | $B = C_1 + C_2$<br>$T = RC_2$<br>$\theta = \frac{C_1}{C_1 + C_2}$ | $R = \frac{T}{B(1 - \theta)}$<br>$C_1 = B\theta$<br>$C_2 = B(1 - \theta)$ |
| $\frac{1}{sB}\frac{1 + s\theta T}{1 + sT}$<br>$\theta < 1$ |   | $B = C_2$<br>$T = RC_1\frac{2C_2 + C_1}{C_2}$<br>$\theta = \frac{2C_2}{2C_2 + C_1}$ | $R = \frac{T\theta^2}{4B(1 - \theta)}$<br>$C_1 = \frac{2B(1 - \theta)}{\theta}$<br>$C_2 = B$ |
| |   | $B = \frac{C_1^2}{2C_1 + C_2}$<br>$T = RC_2$<br>$\theta = \frac{2C_1}{2C_1 + C_2}$ | $R = \frac{T\theta^2}{4B(1 - \theta)}$<br>$C_1 = \frac{2B}{\theta}$<br>$C_2 = \frac{4B(1 - \theta)}{\theta^2}$ |
| |   | $B = \frac{R_1}{R_1 + R_2}C$<br>$T = R_2C$<br>$\theta = \frac{2R_1}{R_1 + R_2}$ | $R_1 = \frac{T\theta^2}{2B(2 - \theta)}$<br>$R_2 = \frac{T\theta}{2B}$<br>$C = \frac{2B}{\theta}$ |

Bradley and R. McCoy, Driftless d-c amplifier, *Electronics*, April, 1952, by permission Publishing Company, Inc.

networks of Fig. 2.23, but these points are much more easily overloaded than points in networks like that of Fig. 2.25, in which amplifiers in the circuit provide driving power. A further advantage of the active-circuit

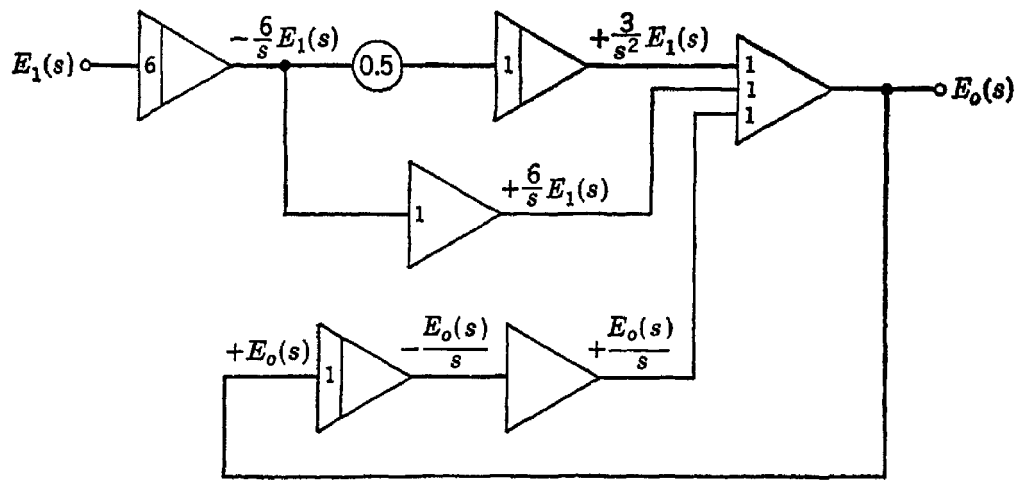



Fig. 2.25 Alternate method of forming $\frac{E_o(s)}{E_1(s)} = -\frac{3+6s}{s+s^2}$

type of simulator is that it is possible to set initial conditions on the various integrals in the transfer function.

## 2.2 Nonlinear differential equations

Our discussions to this point, although ranging over a variety of topics, have been confined to a rather restricted class of functions, namely, those described by linear differential equations. However important such functions may be, a computer system incapable of handling nonlinear functions is severely restricted in its range of application. The nonlinearities encountered are of all different types; one that arises so often that it has required special attention from computer designers is simple multiplication, forming the product of two variables. Many different devices have been used to perform this operation; these are described in more detail in Chap. 3. For the present we shall discuss the use of only one of them, the servo multiplier, since this is the device that the reader is most likely to meet in practice and since in any case the principles of its use can be readily adapted to the faster all-electronic multipliers that will ultimately replace it.

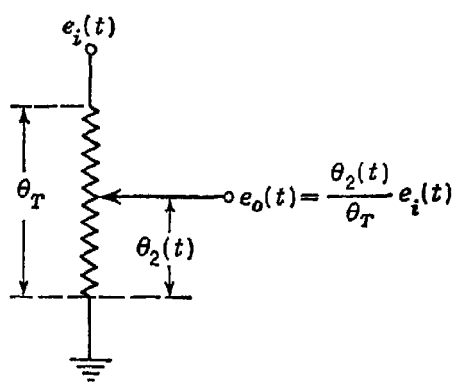



Fig. 2.26 Multiplier based on the use of a potentiometer

**Principle of the servo multiplier.** The servo multiplier is based on the principle of potentiometric multiplication, as shown in Fig. 2.26. If

$e_1(t)$ is a scaled analog of one of the variables to be multiplied, and if $\theta_2(t)/\theta_T$ (the fractional rotation of the potentiometer) is a scaled analog of the other, then the output voltage $e_o(t)$ is proportional to their product. The difficulty with this simple multiplier is the problem of accurately positioning the potentiometer arm. A suitably designed servomechanism

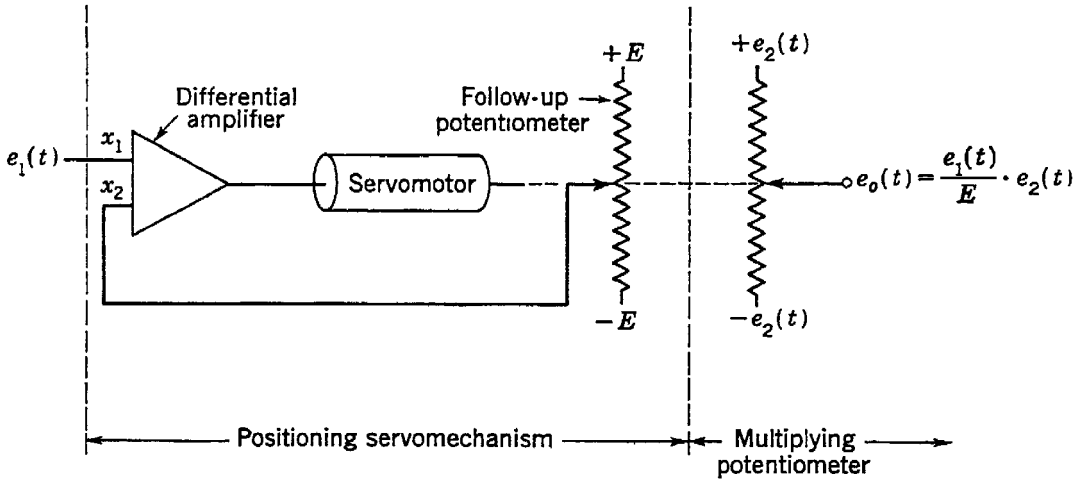



Fig. 2.27 Servo-multiplier block diagram

can be used to do this, as shown in Fig. 2.27. The differential amplifier has the characteristic that, if $x_1 > x_2$, one polarity of output voltage is produced, whereas, if $x_2 > x_1$, the other polarity is produced. The servo motor is connected so that, if $x_1 > x_2$, the resulting polarity of voltage causes the servo motor to rotate in the direction that advances the follow-up potentiometer arm toward the $+E$ end. Thus the difference between $x_1$ and $x_2$ is reduced, and the motor stops when they are equal. The potentiometer has then been advanced by a fraction $e_1(t)/E$ of its total possible rotation from the center position, and the multiplying potentiometer, which is mechanically coupled and, therefore, advanced by the same amount, produces an output voltage of $[e_1(t)/E]e_2(t)$ volts.

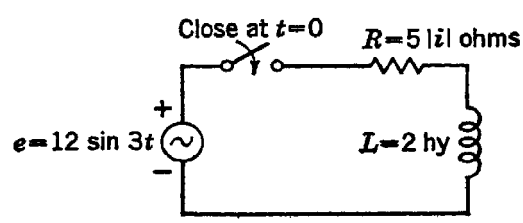



Fig. 2.28 An example of a system requiring use of a multiplier in its setup

**Some servo-multiplier applications.** As an example of the use of a servo multiplier, consider the circuit of Fig. 2.28, which shows a voltage source driving an $RL$ circuit with a nonlinear resistance that may be represented by $R = 5|i|$ ohms. The differential equation is

$$12 \sin 3t = 5|i|i + 2\frac{di}{dt}$$

No change in time scale need be made, since the terms on the right have coefficients of approximately equal magnitude and since the frequency of

the forcing function is well within the range of all computer components needed. To determine approximate magnitude scaling, note that, at the instant the inductor voltage is zero, the entire source voltage appears across $R$. The maximum current therefore cannot exceed $i = \sqrt{12/5}$ amp. To simplify the scale factors, choose $i_{max} = 2$ amp, giving the scale factor $M_i = 1/2$ machine unit* per ampere, or $M_i = 50$ volts/amp if a machine of $\pm 100$ volts' dynamic range is used. The scaled equation is

$$\frac{dI}{dt} = 3 \sin 3t - 5I|I|$$

Since quantities of 3 and 5 machine units (M.U.) are too large for the computer, this is rewritten as

$$\frac{1}{5}\frac{dI}{dt} = \frac{3}{5} \sin 3t - I|I|$$

from which the setup of Fig. 2.29 follows.

Some care must be used in setups involving servo multipliers to make sure that the correct polarities are given to the voltages applied to the

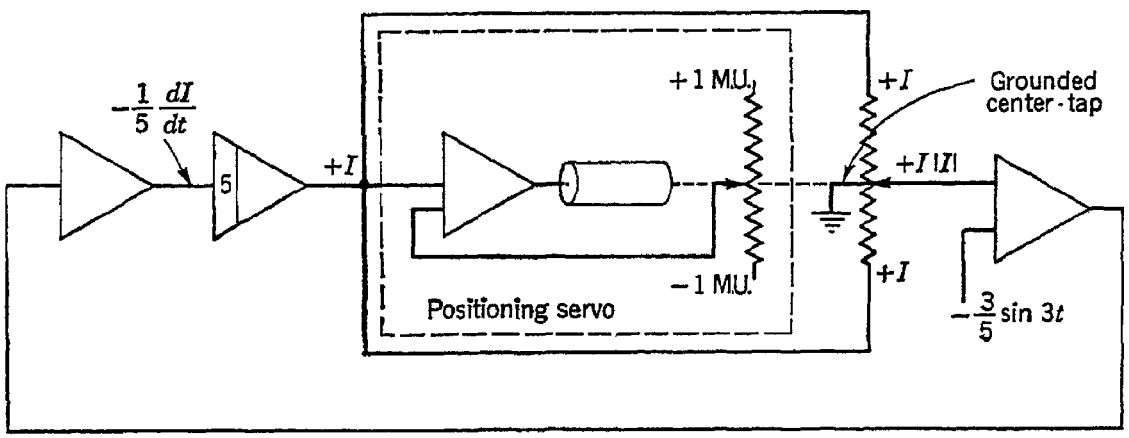


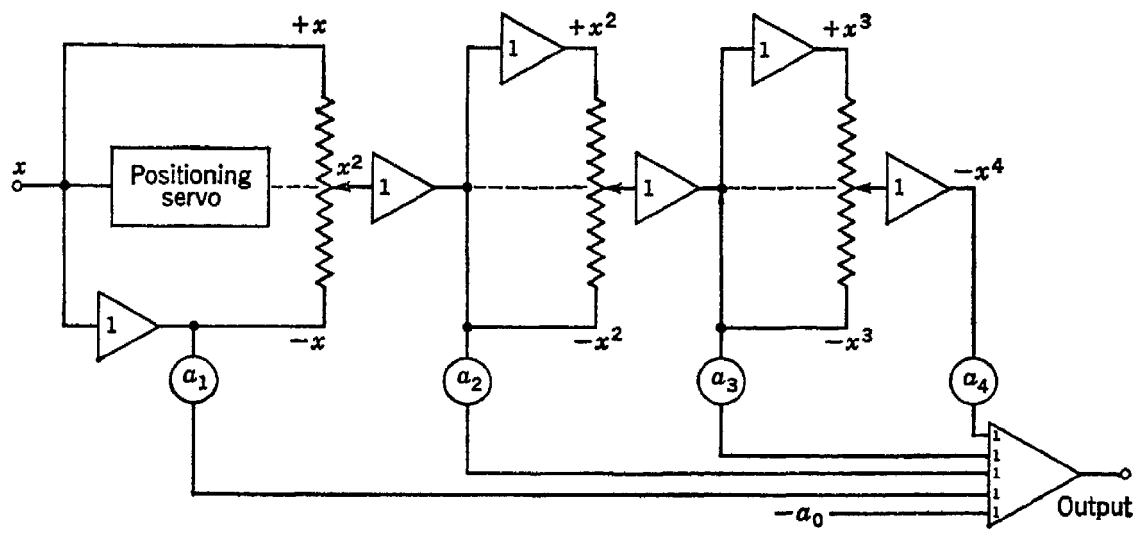

Fig. 2.29 Computer setup for the circuit of Fig. 2.28

multiplying potentiometer. In the setup of Fig. 2.29, it will be seen that, when $I$ is positive, the two servo-driven potentiometers are tapped in the upper half of their ranges. To get a positive value for the output, it is therefore necessary that $+I$ be applied to the upper end of the multiplying potentiometer. On the other hand, when $+I$ is negative and the potentiometer taps are in the lower half of the range, a negative output is obtained by putting $+I$ on the lower end also. Since the same voltage is applied to both ends of the potentiometer, it is necessary to ground the

* Scaling is sometimes done in terms of "machine units" (M.U.), where 1 M.U. is taken to be the maximum output of the amplifiers. Scaling in terms of machine units allows the troublesome factors of $1/100$ to be dropped from the output expressions of the servo multipliers. However, since the various signals in the computer are usually measured directly in volts, scaling is usually done in terms of volts per problem unit.

center-tap in this case. The reader may verify that the multiplier output would have been $+I^2$ if the upper and lower ends of the multiplying potentiometer had been connected respectively to $+I$ and $-I$, but that reversal of both these polarities would have produced $-I^2$. Using $-I$ on both ends would produce $-I|I|$.

If one of the input variables always has the same sign, it is possible to improve the sensitivity of the servo somewhat by grounding one end of the follow-up potentiometer instead of its center. Each increment of angle of servo rotation is now produced with only half as big a signal as when the center-tap was grounded. Thus, for a positive-polarity multiplier input to the servo amplifier, the lower end of the follow-up potentiometer could be grounded, and the full +100 volts (1 M.U.) would be




Output $= [a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4]$
(machine units)

Fig. 2.30 Servo-multiplier system to form a fourth-degree polynomial

applied to the upper end. It is interesting to note in this connection that there is no compelling reason to restrict servo-multiplier operation to the $\pm 100$-volt dynamic range to which computing amplifiers are limited. Input and output voltages, as well as potentiometer terminal voltages, can all have values above 100 without impairing the servo-multiplier operation. None of these voltages, however, can be allowed to overdrive an associated computing amplifier.

Multipliers can be used to perform many functions besides simple multiplication. The use of a multiplier to square an input may already have suggested to the reader the possibility of using successive multipliers to form higher powers of the input and combining various of these to produce a polynomial. This can easily be done, especially if the servo system is able to drive many potentiometers. Such a setup is shown in Fig. 2.30. A polynomial generator is a useful way of generating $f(x)$ if $f(x)$ can be expanded in a power series.

Amplifiers are needed between successive stages of the polynomial generator to provide signal inversion and to reduce the loading on the multiplying potentiometers. Gains of greater than unity may be needed at some of the inputs of the output summing amplifier, if any of the coefficients $a_j$ are larger in magnitude than unity. If any $a_j$ is negative, the output of the second amplifier in the corresponding stage should be used instead of the first.

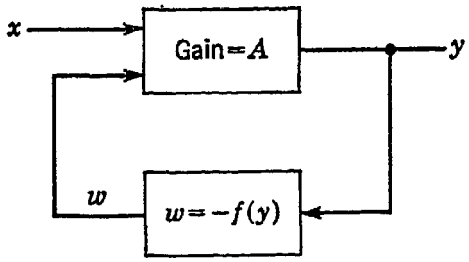



Fig. 2.31 Amplifier with negative feedback

**Generation of inverse functions.** A useful type of computing circuit is based upon the fact that, when a negative feedback network is applied to a high-gain amplifier, the system's transmission characteristic becomes the inverse of that of the feedback network. From Fig. 2.31 we have

$$\begin{aligned} y &= A(x + w) \\ w &= -f(y) \\ y &= Ax - Af(y) \\ Af(y) + y &= Ax \end{aligned}$$

Now, if $Af(y)$ is very much larger in magnitude than $y$, then $y$ may be neglected, and we obtain approximately

$$\begin{aligned} f(y) &= x \\ y &= f^{-1}(x) \end{aligned}$$

where $f^{-1}(x)$ denotes the function that is the inverse of $f(x)$. (The following function pairs, for instance, are inverses: $\epsilon^x$ and $\ln x$, $\sin x$ and $\arcsin x$, $x^n$ and $x^{1/n}$.)

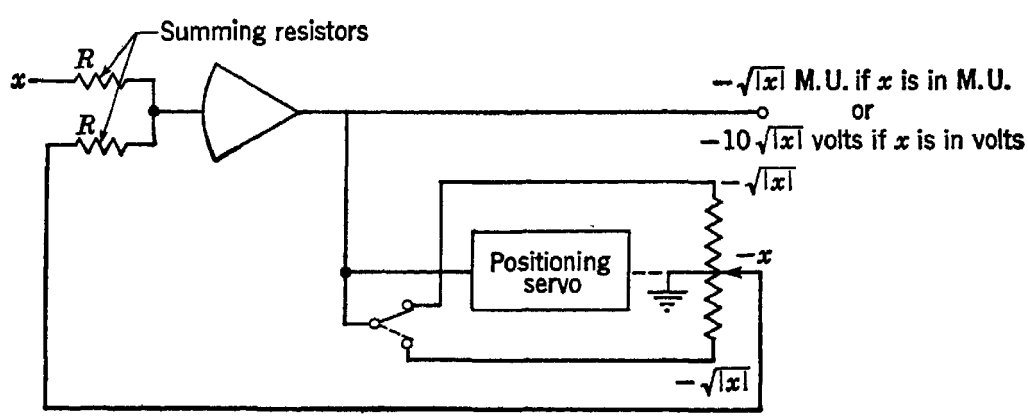



Fig. 2.32 A square-root setup using a squaring network

As an example, consider the use of a squaring network in the feedback path around a computing amplifier that has no other feedback. (A common symbol for such an amplifier is the triangle with curved base.) The circuit is that of Fig. 2.32. Evidently the output $\sqrt{x}$ appears as a volt-

age of polarity opposite to that of $x$. If $x$ is always positive, the lower position of the switch is the correct one; if $x$ is always negative, the upper position is correct. The reader may verify this, keeping in mind that the high-gain computing amplifier produces a polarity reversal. It is also possible to leave both ends of the multiplying potentiometer permanently connected to the servo input and in this way to allow $x$ to range over both positive and negative values.

The principle of inversion by feedback can also be used to perform division by using a multiplying circuit in the feedback path, as in Fig.

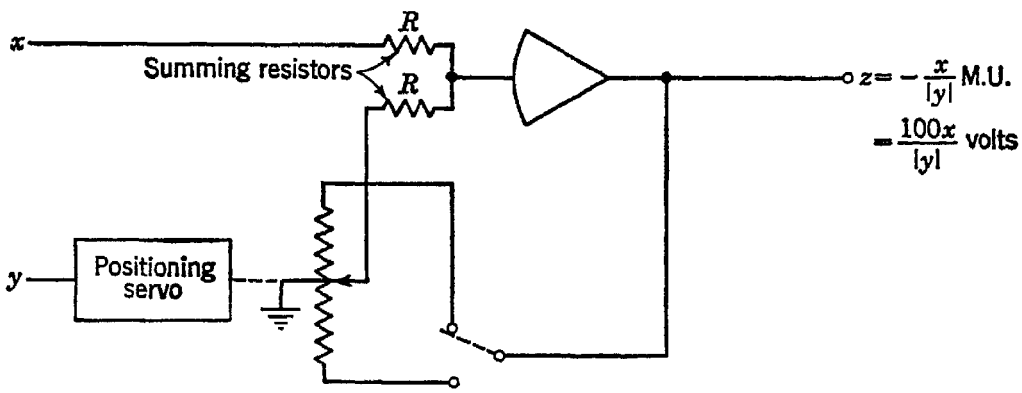



Fig. 2.33 A divider setup using a multiplier network

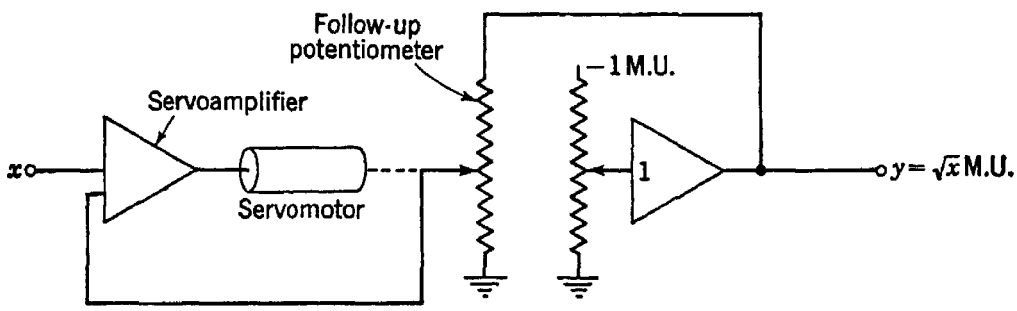



Fig. 2.34 A less satisfactory square-root circuit

2.33. The switch is used in the upper position for positive values of $y$ and in the lower position for negative values. The circuit produces an output opposite in sign to $x$ and equal in magnitude to $x/y$. Instabilities will be encountered in this circuit in the vicinity of $y = 0$ or, more generally, any time $x$ exceeds $y$. Even if $x$ and $y$ approach zero together, output fluctuations are likely, since the feedback circuit is unable to respond to the random noise that becomes the principal component of the amplifier input.

Of course, it is not necessary to use the principle of inversion by feedback to perform function generation, but circuits based on other techniques are often much less satisfactory. A square-root circuit, for example, can be set up as in Fig. 2.34, but suffers from the disadvantage of variable sensitivity of the servo because the voltage applied to the follow-up potentiometer is not constant.

## 2.3 Other applications

The methods described in the preceding sections of this chapter by no means exhaust the ways in which computers and computing components can be used. Many other ingenious techniques have been employed, and we shall conclude this chapter by examining a representative sampling of these methods, which may be helpful in suggesting other applications to the reader.

**Integration with respect to variables other than time.** The variable of integration in electronic differential analyzers is time. The fact that the integrators integrate only with respect to time is not a limitation in solving problems where time is the independent variable or where the independent variable can be treated as though it were proportional to time. However, when the variable of integration cannot be so treated, more subtle means must be used. In machines equipped with multipliers, the technique employed consists in integrating the product of $f(x)$ and $dx/dt$:

$$\int f(x)\,dx = \int \left[f(x)\,\frac{dx}{dt}\right] dt$$

If $dx/dt$ is readily available or if it can be formed without differentiating $x$, this method can be easily implemented. The use of differentiators is undesirable, as we shall see in detail in Chap. 4, since they amplify noise disturbances rather than smooth them as integrators do. However, if there is no other way to form $dx/dt$, an approximate differentiator having adjustable cutoff frequency can be used.

Figure 2.35 shows a circuit using this method of integration to form the frequency-modulated wave

$$e = 50 \sin\left(t + 0.2 \sin \frac{t}{2}\right) \qquad \text{volts}$$

First let $$x = t + 0.2 \sin \frac{t}{2}$$

Then $$e = 50 \sin x$$

$$\frac{d^2e}{dx^2} = -50 \sin x = -e$$

To integrate with respect to $x$, we need $dx/dt$, which is

$$\frac{dx}{dt} = 1 + 0.1 \cos \frac{t}{2}$$

The derivative $dx/dt$ is readily formed by adding a constant to a cosine wave.

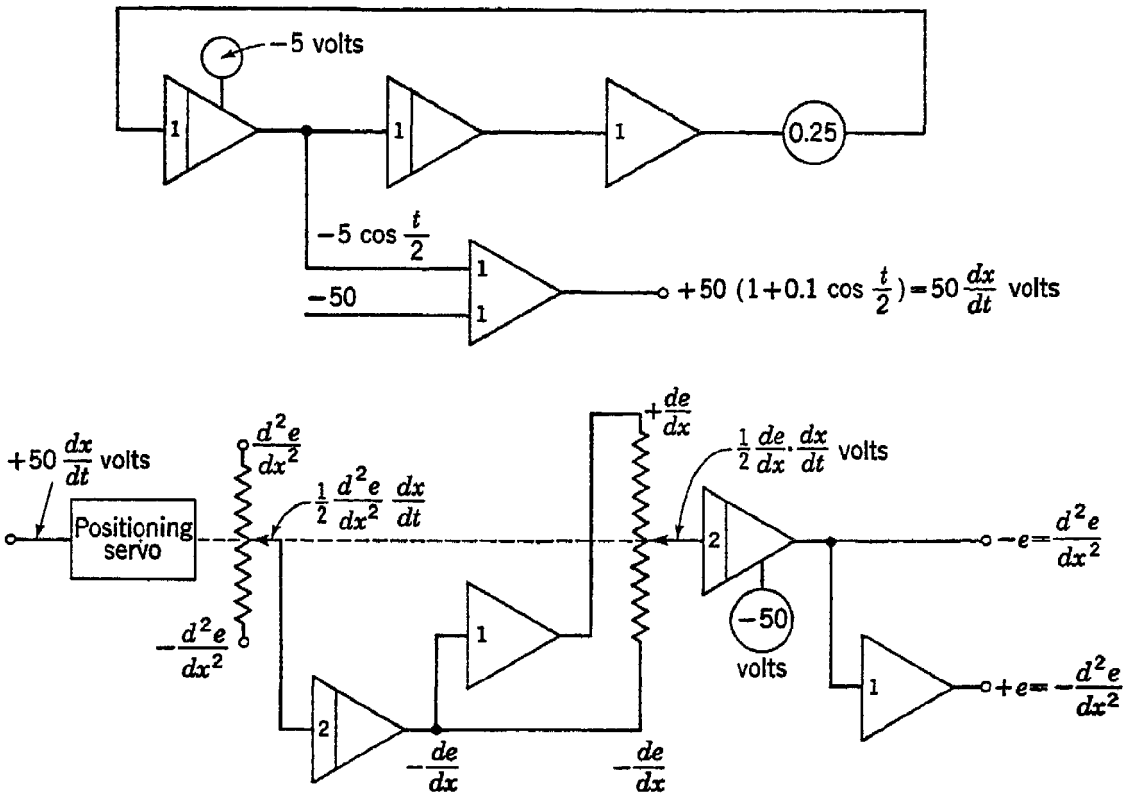



Fig. 2.35 An example of generalized integration using a multiplier

As another example, consider the differential equation

$$\frac{d^2x}{dt^2} + \epsilon^{2x} \cos x = C_0$$

Let

$$y = \epsilon^{2x} \cos x$$

It may then be shown, by differentiating twice, that

$$\frac{d^2y}{dx^2} = 4\frac{dy}{dx} - 5y$$

The setup of Fig. 2.36 follows.

**Evaluation of definite integrals.** It is often possible to use the differential analyzer to evaluate definite integrals. Consider, for example, the equation

$$f(t) = \int_0^\infty \epsilon^{-\omega^2} \cos \omega t \, d\omega$$

Now

$$\frac{df(t)}{dt} = -\int_0^\infty \omega\epsilon^{-\omega^2} \sin \omega t \, d\omega$$

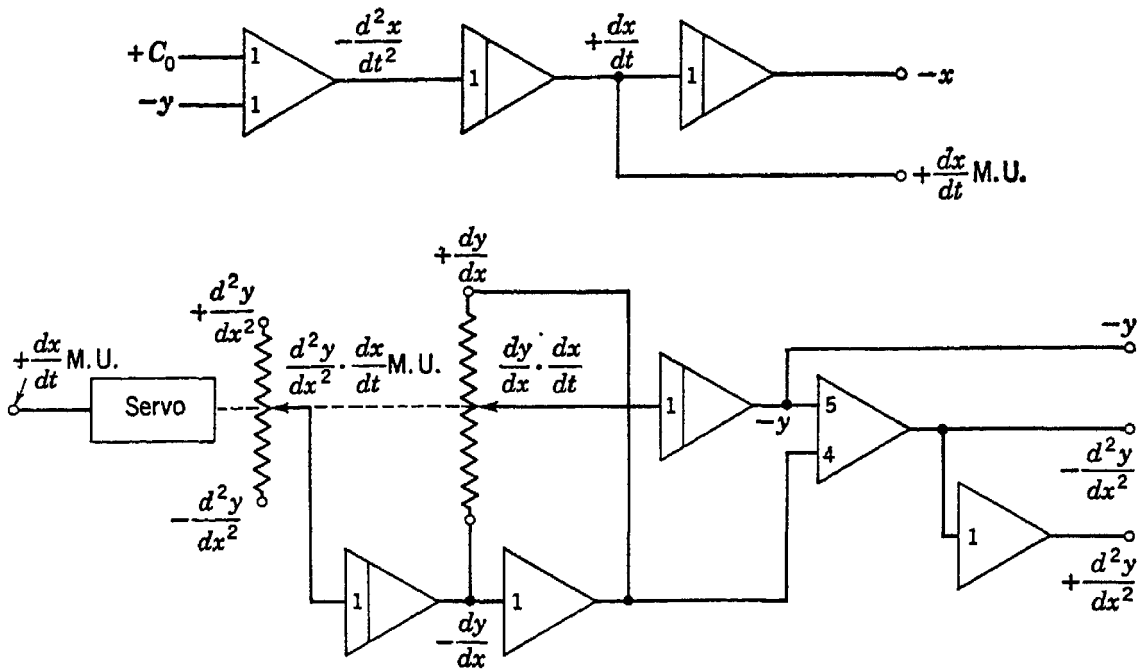



Fig. 2.36 Another example of generalized integration

Integrating by parts,

$$\int_0^\infty \omega\epsilon^{-\omega^2} \sin \omega t \, d\omega = \frac{t}{2} \int_0^\infty \epsilon^{-\omega^2} \cos \omega t \, d\omega$$

that is,

$$\frac{df(t)}{dt} = -\frac{t}{2} f(t)$$

The setup of Fig. 2.37 follows.

The use of a repetitive computer to evaluate integrals of the form

$$F(y) = \int_0^{T \text{ or } t} f(t)K(y,t) \, dt$$

has been described by Macnee,[10] and the general problem of evaluating

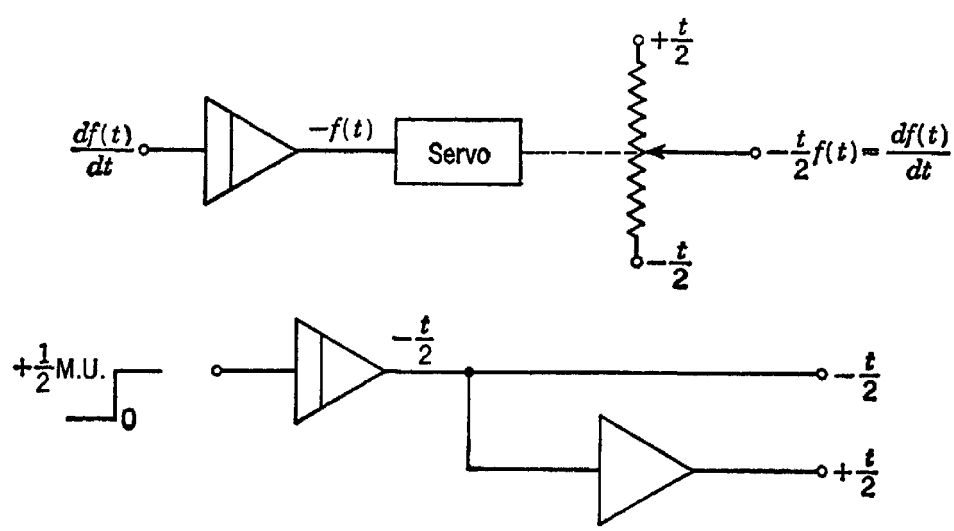



Fig. 2.37 Circuit to form $\int_0^\infty \epsilon^{-\omega^2} \cos \omega t \, d\omega$

transform integrals (Fourier, Laplace, Hilbert transforms, etc.) has been discussed by Wallman.[11]

**Computer setups involving constraints.** In setting up problems on a computer one occasionally encounters a situation in which the coordinates of the problem are not independent of one another but are related by what may be called an "equation of constraint". Such a relation may be regarded as acting to reduce the number of degrees of freedom of the circuit, and it is always possible to choose a smaller number of independent coordinates to describe the constrained system than would be needed for the system without constraints. Often, however, the computer setup representing the reduced system lacks a great deal of the flexibility present in a setup in which each coordinate is represented, even if the coordinates are not all independent.

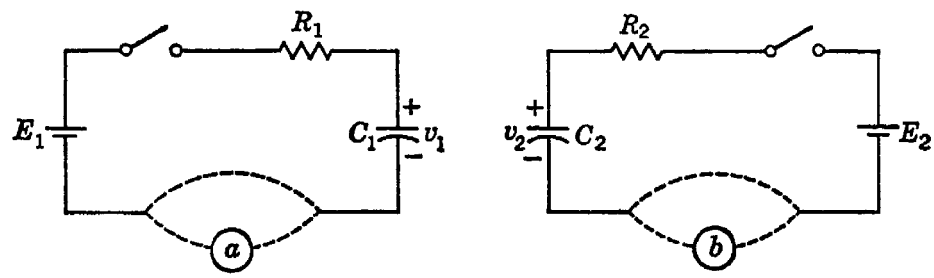



Fig. 2.38 A constrained system. Switches close simultaneously, and capacitors are assumed initially uncharged

A simple example will help to make the procedures evident. Consider the two circuits of Fig. 2.38, with elements $a$ and $b$ short-circuited. The equations may be written in terms of the capacitor voltages $v_1$ and $v_2$ as follows:

$$E_1 = R_1C_1 \frac{dv_1}{dt} + v_1 \tag{1}$$

$$E_2 = R_2C_2 \frac{dv_2}{dt} + v_2 \tag{2}$$

Now suppose that a constraint is introduced, perhaps in the form of suitably varying voltages $a$ and $b$, which has the effect of maintaining a proportion between $v_1$ and $v_2$

$$v_2 = kv_1$$

This constraint relation may be used to reduce the number of coordinates from 2 to 1, giving

$$E_1 + E_2 = (R_1C_1 + kR_2C_2)\frac{dv_1}{dt} + (1 + k)v_1$$

or

$$\frac{dv_1}{dt} = \frac{E_1 + E_2}{R_1C_1 + kR_2C_2} - \frac{1 + k}{R_1C_1 + kR_2C_2} v_1$$

from which the setup diagram of Fig. 2.39 follows.

Although this setup is a mathematically correct equivalent of the constrained system of Fig. 2.38, it has at least two shortcomings: (1) it forms only one of the variables of the original system, and (2) the constraint relationship is awkward to modify. Of course, in this particular problem, it is simple enough to form $v_2$ by adding another potentiometer and another amplifier, but the coordinates eliminated are not always so easily restored. The difficulty of adjusting $k$ is perhaps more significant in this example. $k$ appears in both numerator and denominator of the coefficient of $v_1$, as well as in the denominator of the forcing function; and, although both the inputs to the summing amplifier can be computed for any value of $k$ and set accordingly, it would be much more desirable if a setup could be devised having a single potentiometer by which $k$ could be adjusted at all its appearances in the setup.

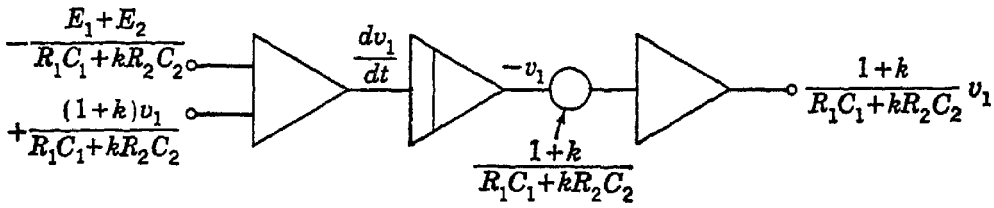



Fig. 2.39 Setup diagram for circuit of Fig. 2.38

Such difficulties as these can be overcome by a computer setup that includes an explicit representation of the constraint equation instead of employing that equation implicitly to reduce the number of coordinates.[12] The constraint equation can usually be put in the form

$$f(x_1,x_2, \ldots ,x_n) = 0$$

where $x_1, x_2, \ldots, x_n$ are the $n$ coordinates of the system; then a computer loop with large negative feedback can be used to enforce the relation. The equations for the system of Fig. 2.38 can be written as follows:

$$R_1C_1 \frac{dv_1}{dt} = E_1 - v_1 \tag{1}$$

$$R_2C_2 \frac{dv_2}{dt} = E_2 - v_2 \tag{2}$$

$$v_2 - kv_1 = 0 \tag{3}$$

and the setup of Fig. 2.40 can be made. In this setup the upper chain of computing amplifiers is a representation of the left part of the circuit of Fig. 2.38, and the lower chain represents the right part. The high-gain amplifier, together with its inputs and outputs, represents the constraint equation. Since the total current supplied to the input point of the high-gain amplifier is $v_1/R_1 - v_2/R_2$, any deviation of this quantity

from zero is amplified and fed back in the proper sense to reduce the difference. If $R_1$ and $R_2$ are potentiometers, then $k = R_2/R_1$ and can easily be adjusted to be either greater or less than unity. (Too low a setting of the potentiometers is inadvisable, for it might overload the amplifiers. The reader can easily devise other ways of adjusting $k$.) The proper sense for the feedback can be determined by noting that, if $v_1$ becomes too large, point $A$ goes negative and $B$ positive, since each amplifier produces a sign inversion. To reduce $v_1$ and increase $v_2$ evidently requires a positive input to the $v_1$ chain and a negative input to the $v_2$ chain. Hence $A$ and $B$ are connected as shown in Fig. 2.40. (The reader may also observe that, by reversing the signs of the quantities in the $v_2$ chain, it is possible to dispense with the two inverting amplifiers associated with the high-gain amplifier.)

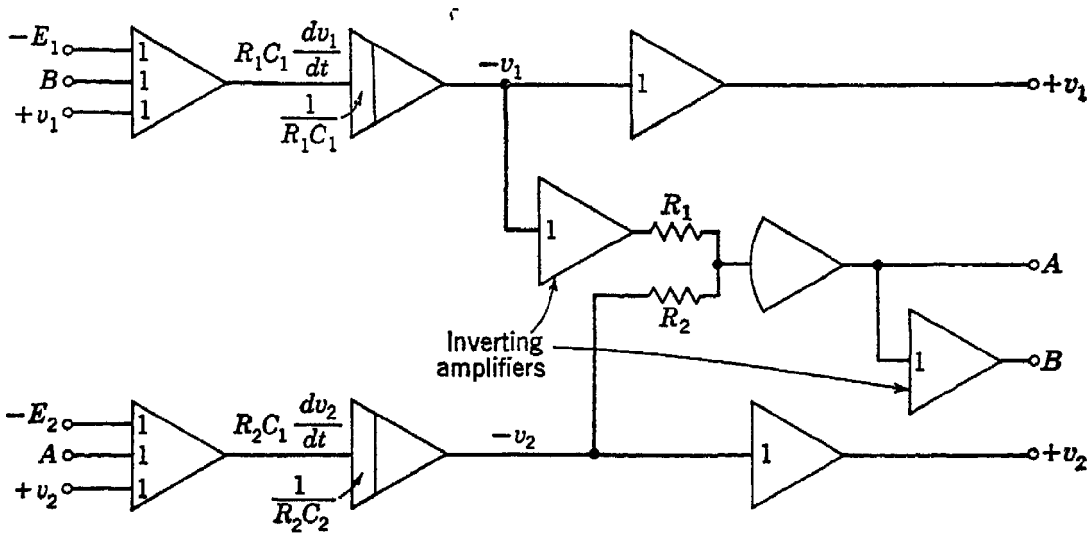



Fig. 2.40 Setup diagram for circuit of Fig. 2.38, with the constraint explicitly represented

The setup of Fig. 2.40 is not a unique way of representing the constraint explicitly. It would also be possible to sense the integrator inputs and let the high-gain feedback circuit determine from the difference in these inputs what the corrected inputs should be to enforce the constraint upon the integrator outputs. It is usually more desirable to sense the constrained quantities directly than to sense some derivative. The more integrators there are between the sensing point and the point to be constrained, the greater is the chance for cumulative integrator drifts to cause deviations from the relation of constraint.

As with any high-gain feedback system, this type of circuit may have enough phase shift to produce a spurious high-frequency oscillation. A small integrating capacitor (100 to 200 $\mu\mu$f) around the high-gain amplifier usually suffices to suppress this oscillation without noticeably affecting the response of the feedback loop.

**Partial differential equations.** All the systems considered so far in this chapter have involved only ordinary differential equations. Such equations, requiring integration with respect to only one variable, are the only differential equations for which true analogs can be constructed using electronic integrators, which integrate only with respect to time. However, it is often possible to construct an approximate analog for a system described by a partial differential equation by replacing differentiation by differencing. If this is done for differentiation with respect to all the independent variables except one, conventional integrators can be used to integrate with respect to this one variable, and summing amplifiers can perform the differencing.

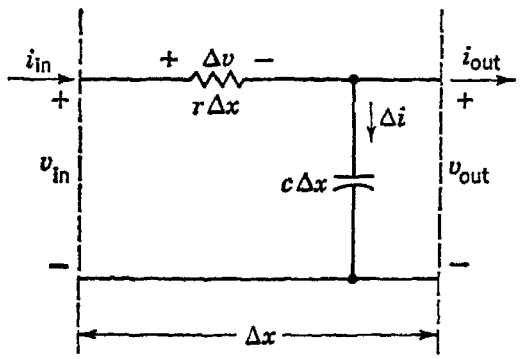



Fig. 2.41 A section of continuous line having series resistance of $r$ ohms per unit length and shunt capacitance of $c$ farads per unit length

We shall demonstrate the method for a simple problem in electric-wave propagation along a continuously distributed transmission line having series resistance and shunt capacitance, as shown in Fig. 2.41. For each incremental section of this line we have

$$\Delta i = c\,\Delta x\,\frac{\partial v_{\text{out}}}{\partial t}$$

and

$$\Delta v = -r\,\Delta x\,i_{\text{in}}$$

In the limit, as $\Delta x \to 0$, these equations become

$$\frac{\partial i}{\partial x} = c\,\frac{\partial v}{\partial t}$$

and

$$\frac{\partial v}{\partial x} = -ri$$

giving finally the partial differential equations

$$\frac{\partial^2 v}{\partial x^2} = -rc\,\frac{\partial v}{\partial t}$$

and

$$\frac{\partial^2 i}{\partial x^2} = -rc\,\frac{\partial i}{\partial t}$$

Having derived the partial differential equation, we now replace it by a difference-differential equation. To do this we consider the potentials at equally spaced positions $x_n$ along the line and note that

$$\left.\frac{\partial v}{\partial x}\right|_{x=x_n} \cong \frac{v_{n+1} - v_n}{\Delta x}$$

where $\Delta x$ is the spacing interval. Similarly,

$$\left.\frac{\partial^2 v}{\partial x^2}\right|_{x=x_n} \cong \frac{\dfrac{v_{n+1}-v_n}{\Delta x} - \dfrac{v_n - v_{n-1}}{\Delta x}}{\Delta x} = \frac{v_{n+1} - 2v_n + v_{n-1}}{(\Delta x)^2}$$

Since these approximations become exact in the limit as $\Delta x \to 0$, it is apparent that the computer approximation will be improved by increasing the number of positions described along the line. Surprisingly good results are often obtained, however, with as few as five or six points.

Our equation now becomes

$$(v_{n+1} - 2v_n + v_{n-1})\frac{1}{(\Delta x)^2} = -rc\frac{dv_n}{dt}$$

The procedure now is to implement the differential equation for each of the positions $x_n$, using as inputs the required outputs $v_{n+1}$ and $v_{n-1}$ from other integrating chains. The form of the resulting network is shown in Fig. 2.42.

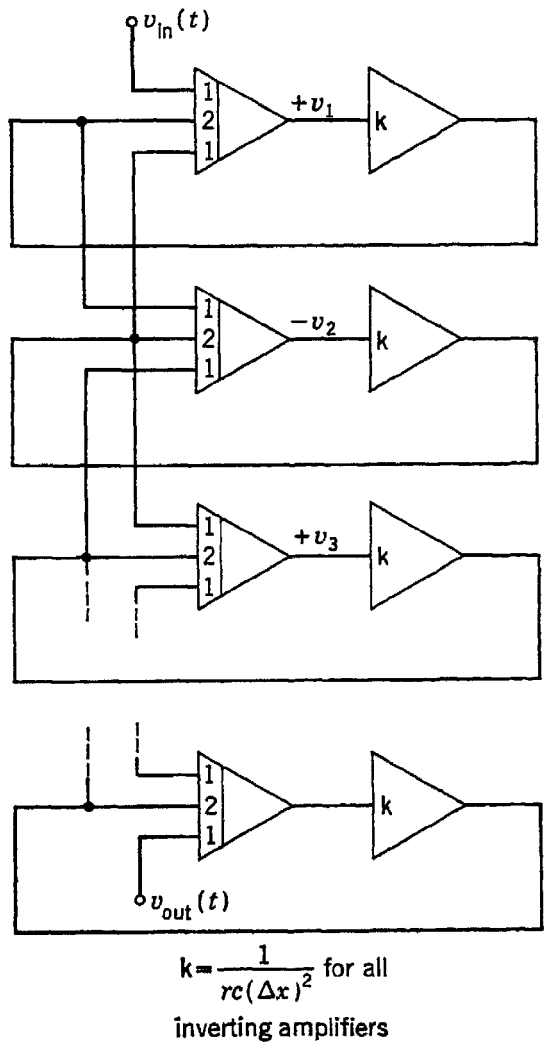



Fig. 2.42 A computer setup for approximating the partial differential equation of the transmission line of Fig. 2.41

Several points of interest may be noted about this circuit. (1) The voltage $v_{in}(t)$ represents the excitation applied at the 0th point on the line. The last point on the line will be either free or driven and, hence, will require either an integrator loop to represent it or a driving excitation $v_{out}(t)$ applied to the integrator chain representing the next-to-last point. (2) We observe that any arbitrary initial distribution of voltage on the line can be represented by appropriate initial charges on the integrating capacitors. In fact, arbitrary nonuniformities of the line can also be represented, all that is necessary being a proper adjustment of the product $rc$ in each of the inverting amplifiers. (3) Finally, the signs of the voltages alternate from point to point in the computer setup in order to save inverting amplifiers that would otherwise be needed.

This technique has been extensively investigated by Professor R. M. Howe and his associates at the University of Michigan, and many references describing it may be found in the literature.[13]

## REFERENCES

*Items Referred to by Number:*


1. Meneley, C. A., and C. D. Morrill, Application of electronic differential analyzers to engineering problems, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1487–1496.
2. Shannon, C. E., Mathematical theory of the differential analyzer, *J. Math. and Phys.*, vol. 20, no. 4, December, 1941, pp. 337–354.
3. Parker, G. M., The loading error in linear potentiometers, *Electronic Eng.*, vol. 23, no. 12, December, 1951, p. 489.
4. Gilbert, J., Here's a shortcut in compensating pot loading errors, *Control Eng.*, vol. 2, no. 2, February, 1955, pp. 36–40.
5. Walters, L. G., Hidden regenerative loops in electronic analog computers, *IRE Trans. on Electronic Computers*, vol. EC-2, no. 2, June, 1953, pp. 1–4.
6. Guillemin, E. A., "Introductory Circuit Theory," John Wiley & Sons, Inc., New York, 1953, chap. 1.
7. Otterman, J., On the loop- and node-analysis approaches to the simulation of electrical networks, *IRE Trans. on Electronic Computers*, vol. EC-7, no. 3, September, 1958, pp. 199–206.
8. Godet, S., Synthesis of input and output networks for feedback amplifiers, *REAC Inform. Bull.* 4, Apr. 28, 1950, Reeves Instrument Corp., New York.
9. Heizer, L. E., and S. J. Abraham, Transfer function simulation by means of amplifiers and potentiometers, *J. ACM*, vol. 3, no. 3, July, 1956, pp. 186–198.
10. Macnee, A. B., A high speed product integrator, *Rev. Sci. Instr.*, vol. 24, no. 3, March, 1953, pp. 207–211.
11. Wallman, H., An electronic integral-transform computer and the practical solution of integral equations, *J. Franklin Inst.*, vol. 250, 1950, pp. 45–61.
12. Greenwood, D. T., The representation of constraints by means of an electronic differential analyzer, *IRE Trans. on Electronic Computers*, vol. EC-5, no. 3, September, 1956, pp. 111–113.
13. Howe, R. M., and V. S. Haneman, Jr., The solution of partial differential equations by difference methods using the electronic differential analyzer, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1497–1508.

*General:*

Johnson, C. L., "Analog Computer Techniques," McGraw-Hill Book Company, Inc., New York, 1956.

Korn, Granino A., and T. M. Korn, "Electronic Analog Computers," 2d ed., McGraw-Hill Book Company, Inc., New York, 1956.

Wass, C. A. A., "Introduction to Electronic Analogue Computers," McGraw-Hill Book Company, Inc., New York, and Pergamon Press Ltd., London, 1955.

## PROBLEMS

**2.1** Show a complete computer setup diagram for solving the following circuit to determine the voltage across $R$. Include magnitude and time scaling, if necessary. Avoid using a divider.

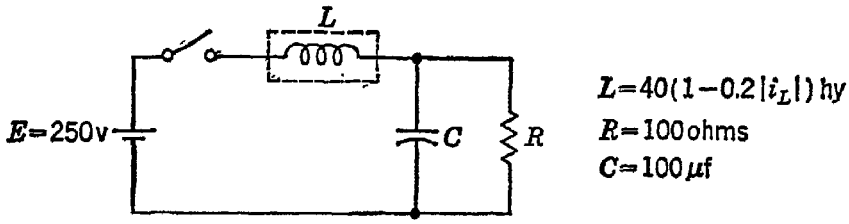



Fig. P2.1

**2.2** The following is a modified form of the Van der Pol equation, describing a self-starting oscillation in a system with nonlinear damping:

$$5\ddot{x} + 50(|x| - 6)\dot{x} + 713x = 0$$

Determine suitable scale factors and show a computer setup to solve this equation. Assume initial conditions to be zero. (Oscillations will start, since the system is in an unstable equilibrium.)

**2.3** The differential equation of the angular displacement of an undamped simple pendulum is

$$\frac{1}{2}mr^2\left(\frac{d\theta}{dt}\right)^2 + mgr(1 - \cos\theta) = \text{constant}$$

Assuming the length $r$ to be 3 ft and the pendulum bob to weigh 1 lb, show a computer setup to solve for $\theta$ with provision for changing the constant. Include suitable scaling.

**2.4** Determine the differential equations one would solve to generate the functions:

(*a*) $y = \tan t$

(*b*) $y = \sec t$

and show the complete setup diagrams for each.

**2.5** Show a computer setup to form the function

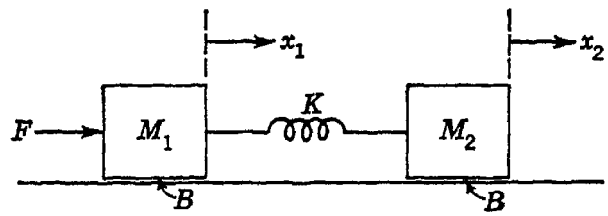
$$f(t) = \int_0^\infty \epsilon^{-\omega t}\cos\omega t\, d\omega$$

**2.6** Show a computer setup to form the function

$$f(t) = 100\epsilon^{-(t^2)/3} \quad \text{volts} \quad \text{for } -3 < t < +3 \text{ sec}$$

Include all necessary scale factors and initial conditions. Adjust the scaling so that the multiplier operates over at least 50 per cent of its full range.

**2.7** Two masses $M_1$ and $M_2$ are coupled by a spring of stiffness $K$ and slide on a surface with viscous-friction coefficient $B$. Show a computer setup to solve for $x_1$ and $x_2$, the two displacements, during the first second after application of $F$, a unit-step force function. Determine new scale factors for solution during the first 10 sec.




$M_1=0.1$
$M_2=0.2$
$B=0.005$
$K=2.0$

Fig. P2.7

**2.8** A certain computer setup requires a forcing function

$$e(t) = 80 \sin\left(3t - \frac{\pi}{6}\right) \qquad \text{volts} \qquad t \geq 0$$

Show a computer setup to generate this function, being careful to include proper initial conditions and to avoid overdriving amplifiers.

**2.9** Determine the transfer characteristics of each of the following networks. Which of them would be of use in solving a linear second-order differential equation with constant coefficients?

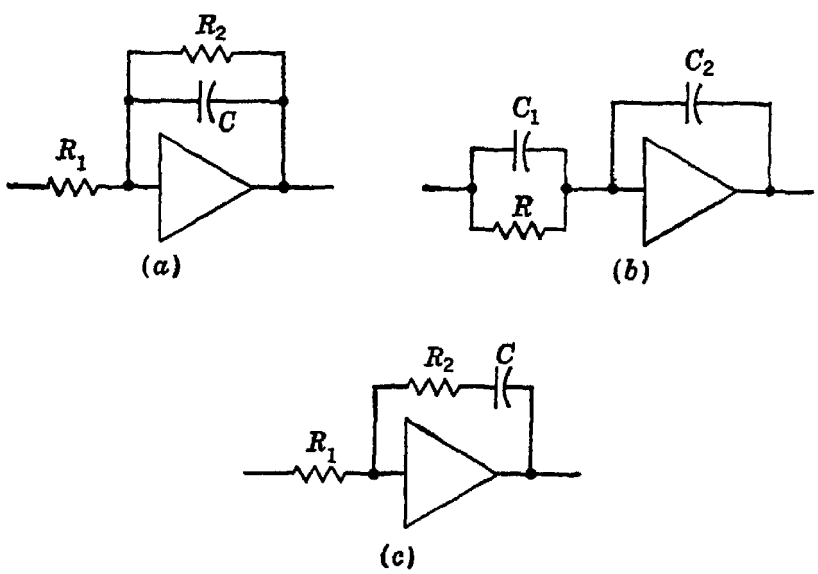



Fig. P2.9

**2.10** Show a computer setup to solve the following differential equation:

$$\dot{x} - (\dot{x} - \tfrac{1}{3}\dot{x}^3) + x = 0$$

At $t = 0$, $x = +1$ and $\dot{x} = 0$. Choose magnitude scale factors appropriate to a computer having $\pm 80$ volts dynamic range.

**2.11** Discuss the behavior of each of the following servo-multiplier connections. (Some of them are quite useless.) In each case assume that the servomotor drives the potentiometer pick-off up if $x > z$, down if $x < z$, and not at all if $x = z$.

**2.12** Determine a suitable time-scale change for each of the following equations and form the modified equation incorporating this change:

(a) $$147 \frac{d^2x}{dt^2} + 23.7 \frac{dx}{dt} + 1.03x = 19 \sin \frac{t}{10}$$

(b) $$0.02\frac{d^2x}{dt^2} + 0.01\frac{dx}{dt} + 0.25x = 3$$

(c) $$\frac{d^2x}{dt^2} + 10\left(\frac{dx}{dt}\right)^2 + 100x = 0$$

(There is no unique answer to a time-scaling problem, so consider the possibilities carefully and be able to support your selection.)

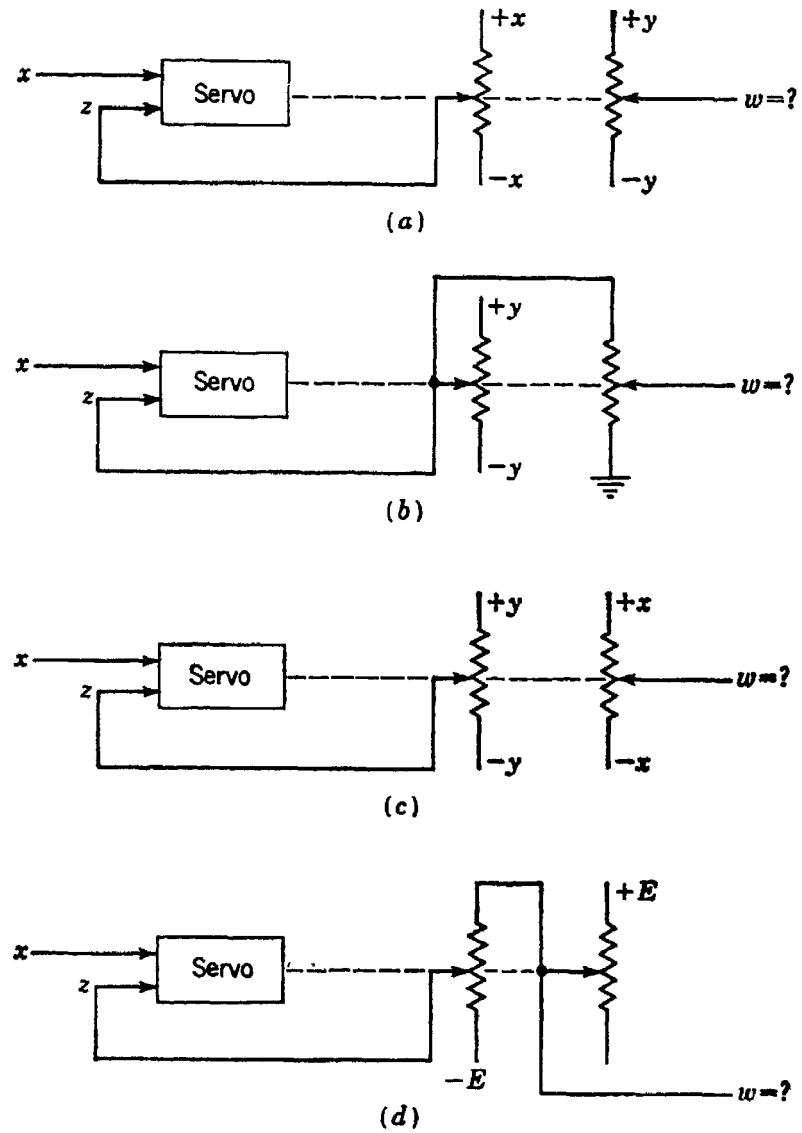



Fig. P2.11

**2.13** Show a complete computer setup to solve for the currents $i_1$ and $i_2$. Use no more than the minimal number of integrators. Determine proper scaling. Include a circuit to generate the forcing function. The switch closes at $t = 0$.

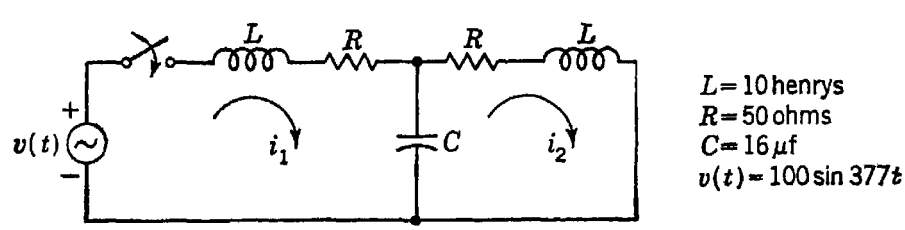



Fig. P2.13

CHAPTER 3

# Computer Representation of Nonlinear Functions

## 3.1 Introduction

In the preceding chapter we have studied the fundamental techniques for placing problems on the electronic differential analyzer and have discussed only to the minimum extent some of the characteristics and limitations of the ways in which this can be done. We turn now to a more detailed consideration of the problems involved in representing nonlinear functions on the computer. The term "nonlinear" is used here in its broadest mathematical sense, referring to all differential equations except the purely linear, i.e., those consisting of a sum of terms each involving only the dependent variable or one of its derivatives raised only to the first power. Our discussion will also cover situations in which purely linear systems are driven by forcing functions that are either impossible or inconvenient to produce by combinations of linear computing elements (integrators, adders, and potentiometers).

This treatment of nonlinear function generators is not intended to catalogue the many ingenious devices and techniques that computer engineers have evolved over the years, although of course many references to specific techniques will be made both to illustrate principles and to familiarize the reader with current practices he is likely to encounter. Our purpose is rather to examine the principles fundamental to the major types of function generator so that the reader may more easily appraise the techniques he meets in practice or even evolve superior techniques for his own applications.

The criteria upon which any function generator, or indeed any computer, must ultimately be judged are such compelling ones as accuracy, speed, reliability, power consumption, initial cost, simplicity, size, weight, and environmental sensitivity. These variables are rarely "free" variables, and the application for which the system is intended places constraints upon them. The engineer has the task of first discovering

whether the constraints are compatible and then, if they are, devising a system which not only meets the requirements imposed but also is optimum in some sense. If the constraints are initially incompatible—for example, a demand for high speed and accuracy together with a requirement of low cost—the engineer must be able to determine this and find the minimum extent to which the constraints must be relaxed in order to permit a physically realizable system. This is the art of any kind of engineering, and there is no short cut to skill in its practice.

## 3.2 Classification of nonlinear function generators

Although the art of engineering, even in respect to so limited a topic as nonlinear function generators, is too vast to be covered in a chapter or even in a volume, there are many important points that can be made. Classifying function generators according to their basic principles is a helpful step toward understanding them. We shall classify them here neither according to the mathematical nature of the function to be generated nor according to the electrical or mechanical nature of the various types of device that can be used, but rather into approximate-function generators and true-function generators, according to whether their operation is based upon an approximation to the function or upon the true function itself. These terms are not meant to carry any connotation of relative inaccuracy and accuracy; accuracy is determined largely by the physical implementation of the method. With a perfectly accurate set of components, however, an approximate-function generator would yield at best an approximation, whereas a true-function generator would yield the function itself. We may classify further as follows:

*A*. Approximate-function generators
   1. Interpolators. These methods proceed from exact values of the function at certain points to approximate values of the function at intermediate points.
      *a*. Curve fitting by analog methods
         (1) By linear segments
         (2) By higher-order segments
      *b*. Digital-to-analog conversion at specific points, followed by analog curve fitting
   2. Explicit generators of an approximating function

*B*. Generators based upon the true function
   1. Curve followers. A graph or cam or other representation of the function is traced, and an electrical reproduction is formed.
   2. Explicit generators of the function. A transfer characteristic identical with that desired is synthesized from elements such as computer amplifiers, square-law diodes, etc.

So far, no mention has been made of multiplication, since this can be regarded simply as a special type of nonlinear function. It should be possible to place any multiplier in one of the general categories we have just established. Nevertheless, multipliers are such useful and versatile elements in a computer that we shall find it helpful to devote some study to the special problems that arise in connection with them. This treatment is presented in Sec. 3.7.

## 3.3 Functional approximation

The necessity for approximating one function by another arises in connection with both analog and digital computers. Because this is so, we shall digress somewhat at this point to discuss some of the mathematical methods of functional approximation.

Functional approximations are needed in such situations as the following:

1. Values of $y = f(x)$ are known only at certain discrete values of $x$, either equally or unequally spaced, and it is desired to generate the function for other values of the argument. If nothing else is known about the function, the problem is virtually meaningless, but a meaningful answer is possible if it is known, for instance, that the function is continuous and varies smoothly between the given points or if it is known that the function can be represented by an $n$th-degree polynomial and the values of the function are known for $n + 1$ values of the argument. In general, the more that is known about the function, the easier it becomes to approximate it closely or even to form it exactly.

2. A graph of the function $y = f(x)$ is known, and from this it is desired to form the analogous voltages $Y = f(X)$ for an analog computer or numerical values $y$ for a digital computer. One suitable analog approach is to form a function, possibly a polynomial or a sequence of straight-line segments, which can easily be generated by the computer and which approximates the function to within some known tolerance. A similar approach is usually used on the digital computer. Although it is theoretically possible to store in the computer memory a complete table of $f(x)$ for all values of $x$, in practice it saves storage space to store a functional relationship that can generate $f(x)$ whenever a value of the argument $x$ is supplied.

3. An explicit equation for the function is known but is too complicated to be implemented on the computer directly. Instead it is desired to form a function simpler and better suited to the computer which approximates the function to within some predetermined tolerance.

Several methods are available for prescribing the manner in which the approximating function is to fit the true function. The following are typical:

1. Prescribing the maximum allowable absolute error. This method ensures merely that the maximum value of the difference between the approximating function and the true function will never exceed a certain amount. It does not ensure that the approximating function will at any point equal the true function, and it is possible for the approximating function to be always larger or always smaller than the true function.

2. Minimizing the maximum allowable absolute error. This method ensures that, for a function of given form (e.g., an $n$th-degree polynomial), no other function of the same form will have a smaller maximum error. Approximating functions developed in this way are usually (but not necessarily) oscillatory; that is, in some regions they are larger than the function approximated and in some smaller. The average error for oscillatory functions tends to be smaller than for nonoscillatory functions.

3. Equal-ripple approximation. If we define the error as the true-function value minus the approximating-function value, then, for an equal-ripple approximation, all maxima of the positive error and all maxima of the negative error are equal in magnitude within a given interval of the argument.

4. Minimizing the percentage error. This method ensures that the ratio of the error to the true-function value will be a minimum over some interval for a particular approximating function.

5. Least-squares fit. With this method, the approximating function fits the given function in such a way that the integral of the square of the error over a given interval is minimized. This is one of the most useful methods of fitting an approximating function. It tends to minimize large errors at the expense of small errors. For this reason and also because it is insensitive to the sign of the error, the method tends to give an equal-ripple type of approximation. It tends to give large errors in the vicinity of discontinuities in the given function, since the value of the least-squares approximant at the point of discontinuity is the arithmetic mean of the true function's values on either side of the discontinuity. It may be noted in passing that the familiar Fourier series forms a least-squares approximation to a periodic function.

In trying to visualize the various kinds of fit the reader may already have grasped an important limitation of approximating functions, namely that, even when the error is small in some sense, the derivative of the approximant may be very different from the derivative of the true function. Only if the approximant is formed with differentiability included as an explicit requirement is there any assurance of differentiability. Otherwise, approximants formed on an equal-ripple or a least-squares basis may give very bad approximations to the derivative, since such approximants usually cross and recross the true function. Integration of approximating functions is much safer, since integration tends to smooth out differences rather than emphasize them as differentiation does.

In numerical analysis it is customary to consider the error to have two components, known as the "truncation error" and the "round-off error". This classification excludes what may be called mistakes, due either to human blunder or to equipment malfunction. We shall of necessity assume that mistakes are either not present at all or so egregious as to be recognized and removed. Truncation error derives its name from circumstances in which the approximant is truncated after a finite number of terms or steps, although inclusion of an infinite number would have given the correct result. Since not all approximants are of this type, it is better to regard truncation errors as comprising all errors other than round-off errors. Round-off errors are errors that result from dropping the less significant portions of a quantity, for whatever reason. The same terms might be somewhat loosely used in connection with analog computers, where we might regard "round-off" error as error due to imprecision of equipment and "truncation" error as due to imprecision of the algorithm used to form the approximant.

**Curve fitting with linear segments.** The first class of approximants that we shall examine are more properly known as interpolating functions than as approximating functions, since they are formed in such a way as (1) to pass through a given set of points $y = f(x)$ corresponding to certain discrete and often equispaced values of the argument $x$, and (2) to be continuous in the intervals between these discrete values of $x$, so as to afford a means of evaluating $y = f(x)$ at intermediate points. Such functions are often used for interpolating between numerical entries in a tabulation, e.g., tables of logarithms. However, they are also useful in the approximation of functions that are known for all values of the argument but not easily reproduced within the computer.

The simplest interpolating function is linear; it assumes a straight-line variation of the function between the given points. Thus, if $x_j$ and $x_{j+1}$ are two adjacent values of the argument and if $f(x_j)$ and $f(x_{j+1})$ are the corresponding function values, we write

$$f(x) = f(x_j) + (x - x_j)\frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j} \qquad x_j \leq x \leq x_{j+1} \tag{3.1}$$

as the linear interpolation between $x_j$ and $x_{j+1}$. Although this approximation has been widely used in analog-computer work because it is so simple to implement, it has some fundamental limitations that should be noted. (1) Although the interpolating function itself is continuous, its first derivative is a staircase function with a step discontinuity at each $x_j$. Some types of function generator deliberately introduce some higher-order curvature in the neighborhood of each of the points separating the straight-line segments, but, although this is qualitatively the right thing to do, it is difficult to do precisely. (2) Between points of inflection of a

curve being approximated, the linear approximants all lie on the same side of the curve, as shown in Fig. 3.1. To the left of the point of inflection at $x = x_i$, all the linear sections lie below $f(x)$, and to the right of $x = x_i$, they all lie above $f(x)$. This means, for example, that, if $y = f(x)$ is concave upward and has no inflection points, then the linear approximants are always larger than the function being approximated, except at the defining points $x = x_j$, where the function and its approximant are equal.

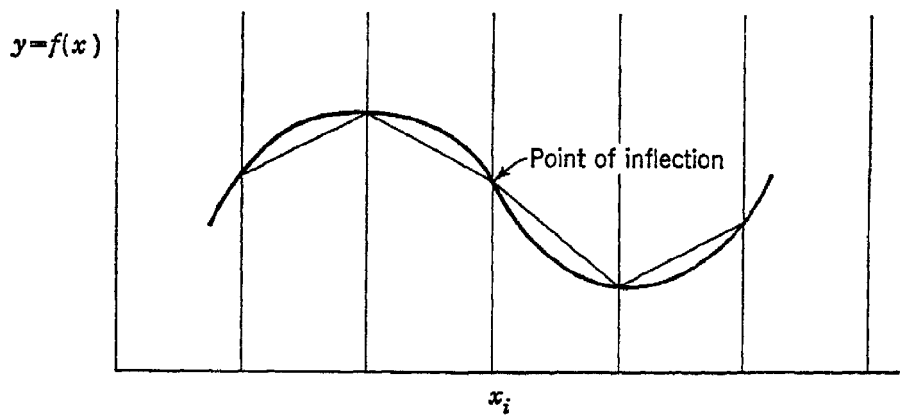



Fig. 3.1 Linear approximations to a curve

Linear approximations of the type just described have no inherent error-minimizing properties. Of course, the maximum error can be kept less than any desired amount if the defining abscissas are made close enough. This suggests that the most flexible function generator of this type should permit the spacing of the $x_j$ to be unequal, so that the defining points may be closely spaced where the function is changing slope rapidly and widely spaced where the slope is nearly constant.

With any physical function-generating device, the number of straight-line segments available determines the accuracy of the approximant, although deciding how best to use these segments is not a simple problem. For instance, for a fixed number of segments, the best approximation, in either a minimum-absolute-error or a least-squares-error sense, is one in which the straight-line segments do not necessarily originate or terminate on the function being approximated. Consider the parabola $y = a^2 - (x - a)^2$ of Fig. 3.2. The reader may verify that the maximum error of the approximation of Fig. 3.2*a* is $0.25a^2$ and that, for the approximants of Fig. 3.2*b* and *c*, which were selected to minimize the maximum error, the maximum errors are $0.172a^2$ for 3.2*b* and $0.125a^2$ for 3.2*c*.

We have, then, a general problem of functional approximation, which may be stated as follows: Given a certain number of straight-line segments of adjustable length and slope to be fitted together in continuous

fashion to approximate a given continuous function over a given interval, what should be the starting point of the first segment and what should be the length and slope of each segment in order to minimize the error in some sense? No general solution of this problem is known. This lack,

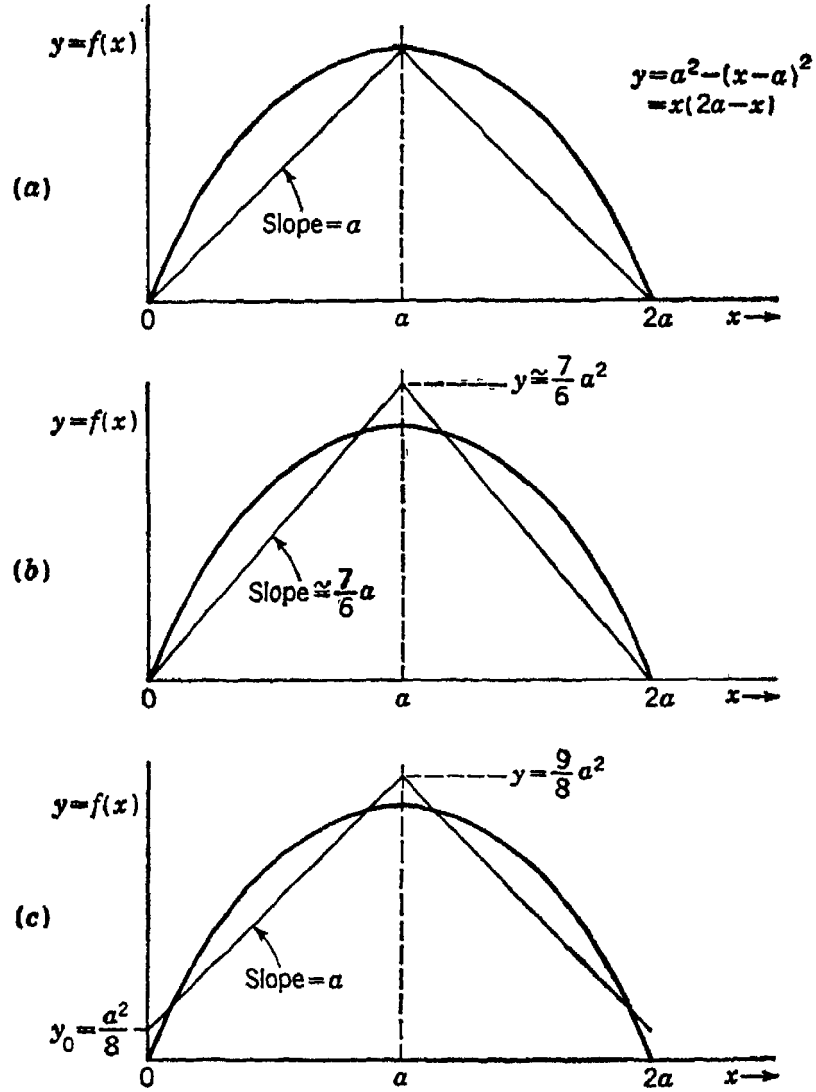



Fig. 3.2 Possible ways of approximating a curve by means of linear segments: (*a*) linear approximants originating and terminating on the curve to be approximated, (*b*) linear approximants having one terminus on the curve to be approximated, and (*c*) linear approximants neither originating nor terminating on the curve to be approximated

however, is not ordinarily a handicap to the analog-computer engineer, since the imprecisions in present-day electronic differential analyzers are large enough to mask the inaccuracies due to improper choices of segment length and slope. Another way of putting this is to say that the number of line segments provided in most function generators is large enough to allow most functions to be fitted to within the tolerances of the computing equipment. If enough segments are available, moderately

crude methods may be used to select their length and slope without the necessity for involved calculations. However, for computing equipment of high precision or for applications in which the number of segments is to be minimized, a careful error analysis must be made in fitting the approximant.

The problem of approximating a given curve to within a specified tolerance by means of straight-line segments can readily be solved graphically. Thus, if $\varepsilon$ represents the absolute value of the maximum allowable error, then the curves $(1 + \varepsilon)f(x)$ and $(1 - \varepsilon)f(x)$ are the bounds of the region within which the approximating straight-line segments may lie. Any

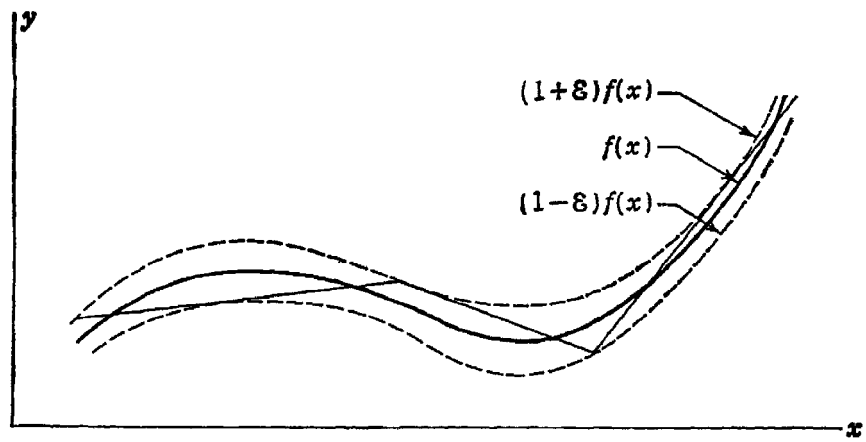



Fig. 3.3 Graphic method of fitting straight-line approximants to within a specified tolerance $\varepsilon$

arrangement of straight-line segments within the region may be used, but it is usually desirable to use as few as possible. This can be done by selecting the starting point on one boundary and drawing the longest possible straight line within the region. The end of this line is the start of the next, and the process is continued throughout the desired range of variation of the independent variable. Figure 3.3 illustrates the procedure.

**Curve fitting with polynomials.** Approximating functions that provide a better fit than the simple linear function just described can be formed from polynomials that equal the given function at some or all of a selected group of coordinates. A well known theorem due to Weierstrass states that, if $n + 1$ values of a function are given, there exists a unique polynomial of degree $n$ that passes through all these points. A second theorem states that, if a function is continuous over a closed interval, it can be approximated to any specified tolerance over the whole interval by some polynomial. These theorems are of interest to the user of analog-computer equipment as well as to the user of digital computers, since they indicate the feasibility of polynomial generators as a means of approximating arbitrary continuous functions.

The problems involved in selecting the points through which to pass the polynomial approximation are much the same as with the linear approximation. For a given set of $n + 1$ points, the choice is whether to pass a polynomial of degree $n$ through all of them or to fit various sections with polynomials of lower degree. If the maximum number of points that can be selected is fixed by the design of the function generator but the abscissas can be chosen at will, then the problem of picking these points in such a way as to minimize the error in some sense can be very difficult. If the number of points can be made arbitrarily large, then,

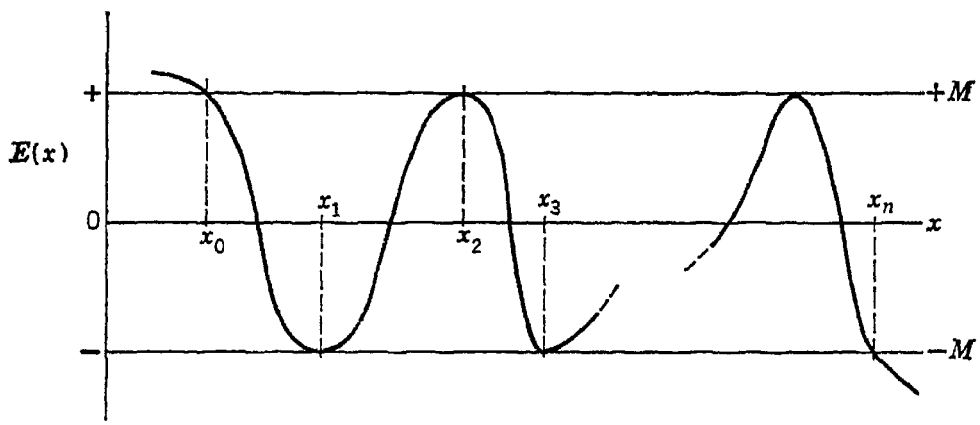



Fig. 3.4 An equal-ripple error

by the second theorem of Weierstrass, the function can be approximated arbitrarily closely. However, complete freedom to use any large number of points is a luxury rarely afforded the computer user.

It is at least theoretically possible to pick the abscissas of the $n + 1$ points in such a way as to give an equal-ripple approximation, i.e., an approximation such that the maximum positive errors and the maximum negative errors are equal in magnitude. Consider a curve of the equal-ripple error $E(x)$ as shown in Fig. 3.4. Clearly,

$$E(x_0) = -E(x_1) = E(x_2) = \cdots = (-1)^n E(x_n)$$

Now, if the polynomial is designated by $P(x)$ and the function to be approximated by $f(x)$, then

$$\begin{aligned} & f(x_i) - P(x_i) = (-1)^i M && i = 0, 1, \ldots, n \\ \text{and} \quad & f'(x_i) - P'(x_i) = 0 && i = 1, 2, \ldots, n-1 \end{aligned} \tag{3.2}$$

provided $f(x)$ is continuous and differentiable. $M$ is equal in magnitude to the maximum error and has whichever sign is appropriate. $x_0$ and $x_n$ are considered to be the known end points of the interval. There are thus $2n$ equations to be solved for $n$ coefficients of the polynomial, $n - 1$ points $x_1, \ldots, x_{n-1}$, and $M$. The equations are not likely to be linear,

however, and, even if they were, the labor of calculation might well be so prodigious as to require a digital computer. Some digital-computer establishments have "mini-max" programs available, for use in minimizing the maximum error of the approximant, and the analog-computer user may well find it entirely practical to avail himself of their help in setting up his function generator. One such program, for example, makes a selection, from among a set of $N$ points, of a subset $n + 2 \leq N$ that allows determination of a mini-max polynomial of degree $n$ and of the maximum error of its fit.

If the abscissas at which the approximating polynomial is to equal the desired function have been selected but the polynomial coefficients have not been evaluated, we can choose among several methods of determining those coefficients. One obvious method is to form the set of $n + 1$ equations

$$\begin{aligned} a_0x_0^n + a_1x_0^{n-1} + \cdots + a_{n-1}x_0 + a_n &= f(x_0) \\ a_0x_1^n + a_1x_1^{n-1} + \cdots + a_{n-1}x_1 + a_n &= f(x_1) \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \end{aligned} \tag{3.3}$$

which can then be solved for the coefficients $a_j$. This task is usually no small one, and a digital computer might well be needed to carry it out. Another technique, which does not necessitate explicit formation of the polynomial, is to use a method such as Newton's formula, which is based on tables of differences. Let us define the first divided difference of a function, relative to the arguments $x_0$ and $x_1$, to be

$$f(x_0,x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \tag{3.4}$$

Then a linear approximation to $f(x)$ between $x_j$ and $x_{j+1}$ may be written

$$f(x) \cong f(x_j) + (x - x_j)f(x_j,x_{j+1}) \tag{3.5}$$

Similarly, the second divided difference $f(x_0,x_1,x_2)$ is defined as

$$\begin{aligned} f(x_0,x_1,x_2) &= \frac{[f(x_2) - f(x_1)]/(x_2 - x_1) - [f(x_1) - f(x_0)]/(x_1 - x_0)}{x_2 - x_0} \\ &= \frac{f(x_1,x_2) - f(x_0,x_1)}{x_2 - x_0} \end{aligned} \tag{3.6}$$

More generally,

$$f(x_0,x_1, \ldots ,x_k) = \frac{f(x_1,x_2, \ldots ,x_k) - f(x_0,x_1, \ldots ,x_{k-1})}{x_k - x_0} \tag{3.7}$$

Newton's formula then is

$$\begin{aligned} f(x) \cong f(x_0) &+ (x - x_0)f(x_0,x_1) + (x - x_0)(x - x_1)f(x_0,x_1,x_2) \\ &+ \cdots + (x - x_0)(x - x_1) \cdots (x - x_{n-1})f(x_0,x_1, \ldots ,x_n) \end{aligned} \tag{3.8}$$

The various divided differences can be formed from a tabulation of the values $f(x_j)$, as in Table 3.1, and a group of multipliers can be connected

**Table 3.1**
**Table of divided differences**

| | | | | | |
|---|---|---|---|---|---|
| $x_0$ | $f(x_0)$ | | | | |
| | | $f(x_0,x_1)$ | | | |
| $x_1$ | $f(x_1)$ | | $f(x_0,x_1,x_2)$ | | |
| | | $f(x_1,x_2)$ | | $f(x_0,x_1,x_2,x_3)$ | |
| $x_2$ | $f(x_2)$ | | $f(x_1,x_2,x_3)$ | | |
| | | $f(x_2,x_3)$ | | $f(x_1,x_2,x_3,x_4)$ | etc. |
| $x_3$ | $f(x_3)$ | | $f(x_2,x_3,x_4)$ | | |
| $\cdot$ | $\cdot$ | $f(x_3,x_4)$ | $\cdot$ | $f(x_2,x_3,x_4,x_5)$ | |
| $\cdot$ | $\cdot$ | $\cdot$ | $\cdot$ | $\cdot$ | |
| $\cdot$ | $\cdot$ | $\cdot$ | $\cdot$ | $\cdot$ | |
| $\cdot$ | $\cdot$ | $\cdot$ | $\cdot$ | $\cdot$ | |
| $\cdot$ | $\cdot$ | | $\cdot$ | | |
| $x_n$ | $f(x_n)$ | | | | |

to generate the polynomial, as shown symbolically in Fig. 3.5. The top row of summing amplifiers forms the quantities $x - x_j$, which then go to the multipliers, which form $(x - x_0)(x - x_1)$, $(x - x_0)(x - x_1)(x - x_2)$, etc. The factors $f(x_0,x_1)$, $f(x_0,x_1,x_2)$, etc., are set on the potentiometers. Additional amplifiers that may be needed to adjust the signs or magnitudes of the various terms would be inserted between the potentiometers and the final summing amplifier.

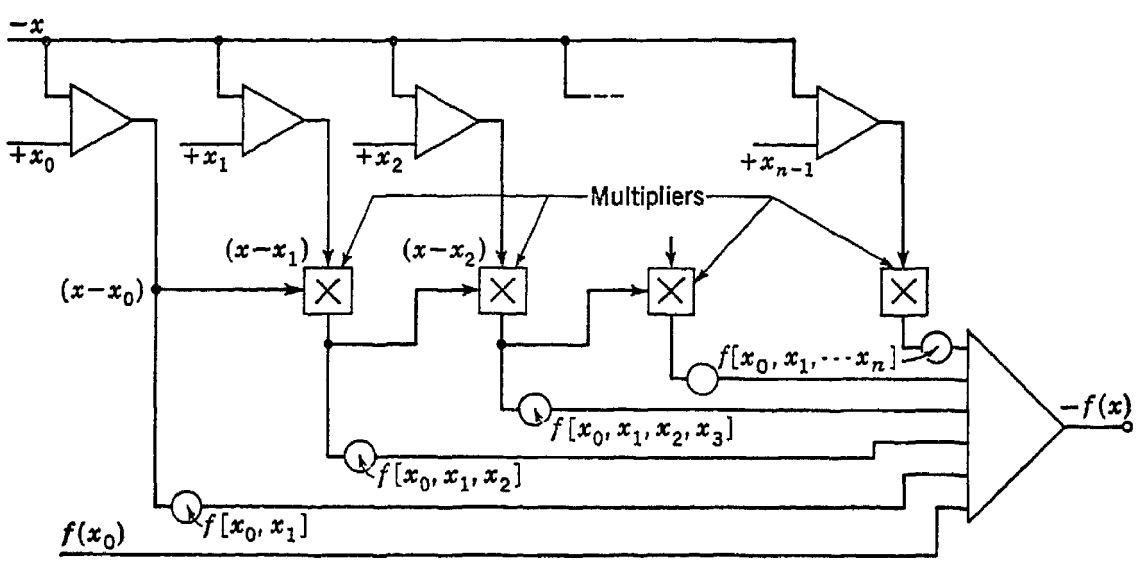



Fig. 3.5 A polynomial generator based on Newton's formula

Use of tables of differences to evaluate the polynomial is usually easier than direct solution of equations (3.3) to determine the coefficients. If the polynomial is needed explicitly, it is always possible to expand Eq. (3.8) into polynomial form.

If the coefficients are available, a system similar to that of Fig. 3.5 can be used to generate the polynomial. Such a system is shown in Fig. 3.6. As in Fig. 3.5, the potentiometers may have to be supplemented by amplifiers to change the signs or the magnitudes. Since the method shown in Fig. 3.6 involves a common input to all the multipliers, it is particularly well adapted to devices whose construction is such as to allow multiplication of many variables by a common variable. A good example is the servo multiplier driving several multiplying potentiometers. Such a circuit is shown in Fig. 2.30.

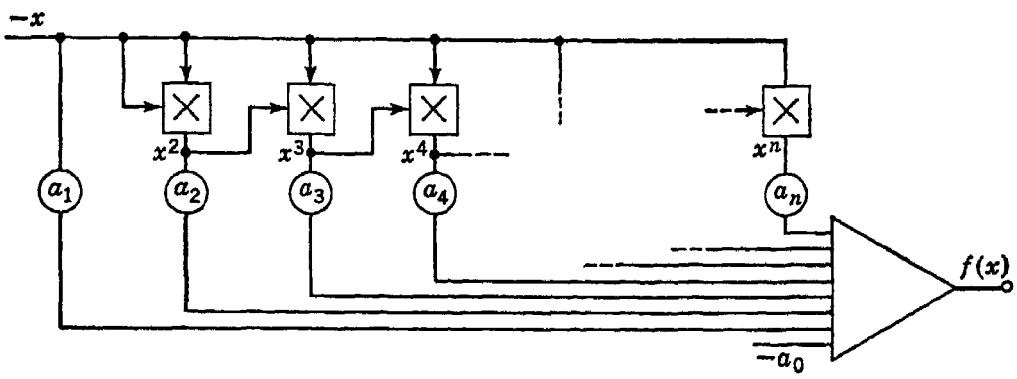



Fig. 3.6 A polynomial generator based directly on the polynomial

Other methods of using Newton's formula are also possible. For example, in one such system,[1] a linear interpolation between $n$th-order differences at two adjacent points is integrated $n$ times to give an $(n + 1)$st-degree polynomial approximating the function.

**Other types of approximation.** Before we leave the subject of functional approximation, it should be noted that a great many ways exist in which to form interpolants or approximants. There are many formulas, such as Newton's, Stirling's, and Bessel's, that can be used with tables of differences based on equal increments of the argument. These are useful principally for interpolating in tables of functions, since, as we have seen, better approximations can generally be obtained if the defining abscissas are not equally spaced. Lagrange's formula for interpolation, based upon the values of the function at the various abscissas instead of upon their differences, may prove useful in some cases. Several types of polynomials giving approximations in a least-squares sense, such as Legendre, Laguerre, Hermite, and Tchebycheff polynomials, are well known and extensively described in the literature. Fourier series, as approximants to periodic functions, are widely used by electrical engineers and are probably familiar to the reader. The possibility of using approximants based on continued-fraction expansions might be attractive if simple dividers existed. A detailed discussion of these and other techniques is beyond the scope of this work, however, and the reader is

referred to some of the excellent reference works that treat these topics more fully.*

## 3.4 Some types of approximate function generators

Having considered some of the mathematical problems involved in approximating functions, let us now examine some typical devices that generate these functions. We have already seen that a group of multipliers can be interconnected in such a way as to form a polynomial approximation, and some computer installations have employed this technique. However, the only simple device yet developed for multiplying each of several variables by a common variable factor is the servo multiplier (Fig. 2.30), which unfortunately has several limitations. Its speed of response is too slow for many real-time problems, and, if a high-order polynomial is to be formed, a large number of potentiometers and isolating amplifiers must be used. In addition, a powerful servomotor may be needed to drive the mechanical load of the multiplying potentiometers. Thus a servo-multiplier type of polynomial generator is one of the more expensive function generators. Simpler, more versatile, and less expensive function generators of other types have come into widespread use, although the principle of the common-factor multiplier system would be very useful if more suitable multipliers were available.

**Use of diode switches for function generation.**[2–6] One of the most attractive devices used for function generation employs biased diodes to switch from one linear-segment approximant to the next. Such devices, either semiconductor or thermionic, provide one of the most flexible ways of nonlinearizing a circuit; usually they are inserted in either the input path or the feedback path of a computer amplifier or both, in order to alter its transfer characteristic. We shall first examine circuits in which the diodes are used simply for limiting and then study ways in which diodes can be used to generate arbitrary functional relations between output and input. For the present we shall assume that the diodes are ideal switches, characterized by no current flow when reverse voltage is applied and no voltage drop when forward current flows. It is a fairly easy matter to modify the idealized transfer characteristic to take into account the departure of practical diodes from this ideal.

Diodes can be used as limiters either in series with one of the current paths to the amplifier input or in shunt. The four basic arrangements and the transfer characteristics of each are shown in Fig. 3.7.

In 3.7*a* the output voltage varies only in a range corresponding to input voltages that do not allow either diode to conduct. When the

* For example, see the book by Hildebrand, listed among the general references at the end of this chapter.

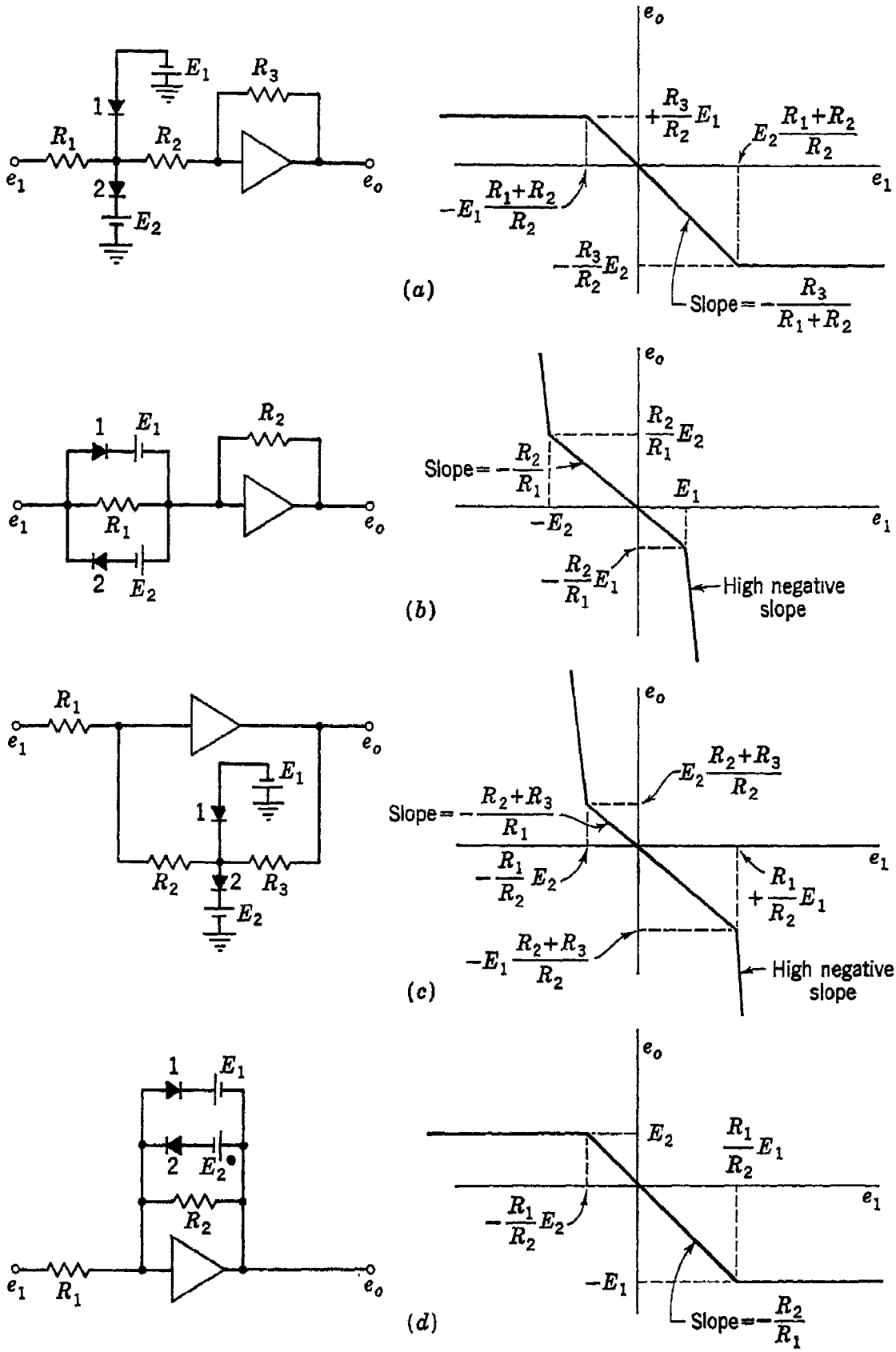



Fig. 3.7 Basic diode limiter circuits: (*a*) input shunt limiting, (*b*) input series limiting, (*c*) output shunt limiting, (*d*) output series limiting

junction of the diodes becomes more positive than the cathode return potential of diode 2 or more negative than the anode return potential of diode 1, then the junction potential is clamped. Additional current produced by changes in $e_1$ is diverted through the conducting diode instead of flowing through $R_2$; therefore, no change in output voltage occurs. Note that, by reversing either $E_1$ or $E_2$, the active range of the transfer characteristic can be restricted to a region in only one quadrant instead of two as shown. (Of course, the anode of diode 1 must be at a lower potential than the cathode of diode 2 if there is to be any active range at all.)

Similar analyses account for the characteristics in 3.7*b*, *c*, and *d*. However, it will be observed that in *b* and *c* the effect of diode conduction is

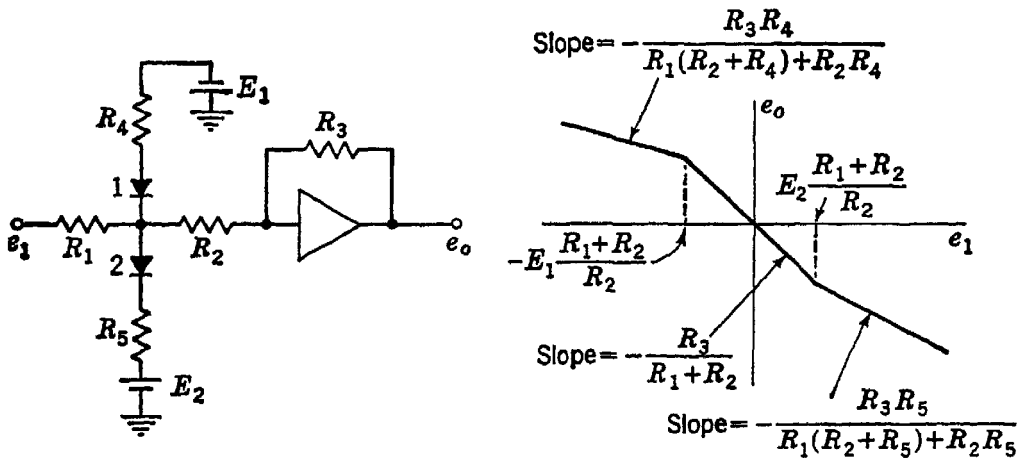



Fig. 3.8 Effect of resistance in series with the diodes of Fig. 3.7*a*

to make the over-all gain very high, rather than nearly zero as in *a* and *d*. This is so because in *b* the input resistor is shorted out by diode conduction and current into the summing point can increase without limit, and in *c* the feedback path is in effect opened by conduction of one of the diodes, since further increases in output-voltage magnitude do not increase the current fed back to the summing point at the amplifier input terminal.

A great many variations of these basic circuits are possible and can be used to produce many different transfer characteristics. For example, resistances in series with the diodes will make the change in slope smaller in all circuits 3.7*a*, *b*, *c*, and *d*. Thus, for circuit 3.7*a*, we would have the characteristic shown in Fig. 3.8.

Diode conduction has had the effect of reducing the (negative) value of the slope below its value in the nonconducting range, but the presence of the resistors keeps the junction point from being solidly clamped in potential, so that some change in output still occurs. The same effect can be achieved with the circuit of *d* by putting resistances in series with the diodes.

In the circuits of Fig. 3.7*b* and *c*, diode conduction gives the curve a negative slope steeper than in the normal range, and inclusion of series resistances can be used to make this slope less steep. Thus we see that it is a simple matter to shape the transfer characteristic of a single amplifier so that at each break the new slope can be anything between zero and a very large negative value. These results are summarized in Fig. 3.9.

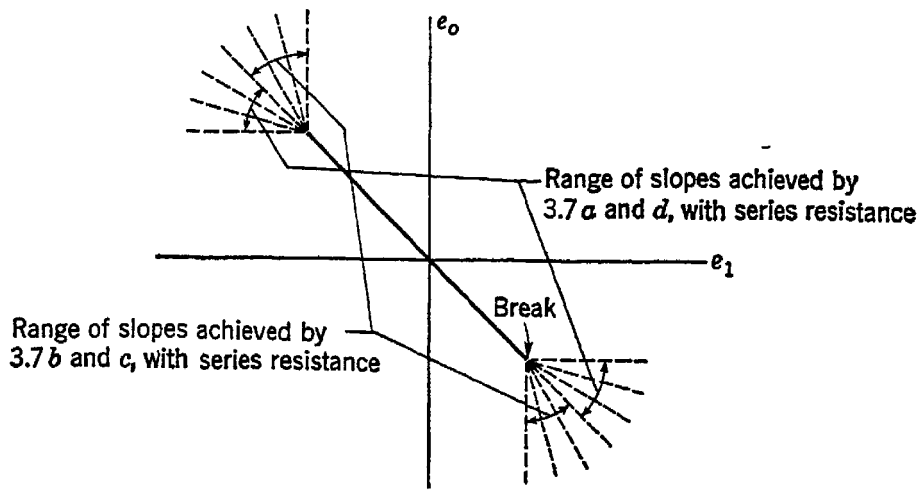



Fig. 3.9 Summary of effects of resistances in series with the diodes of Fig. 3.7

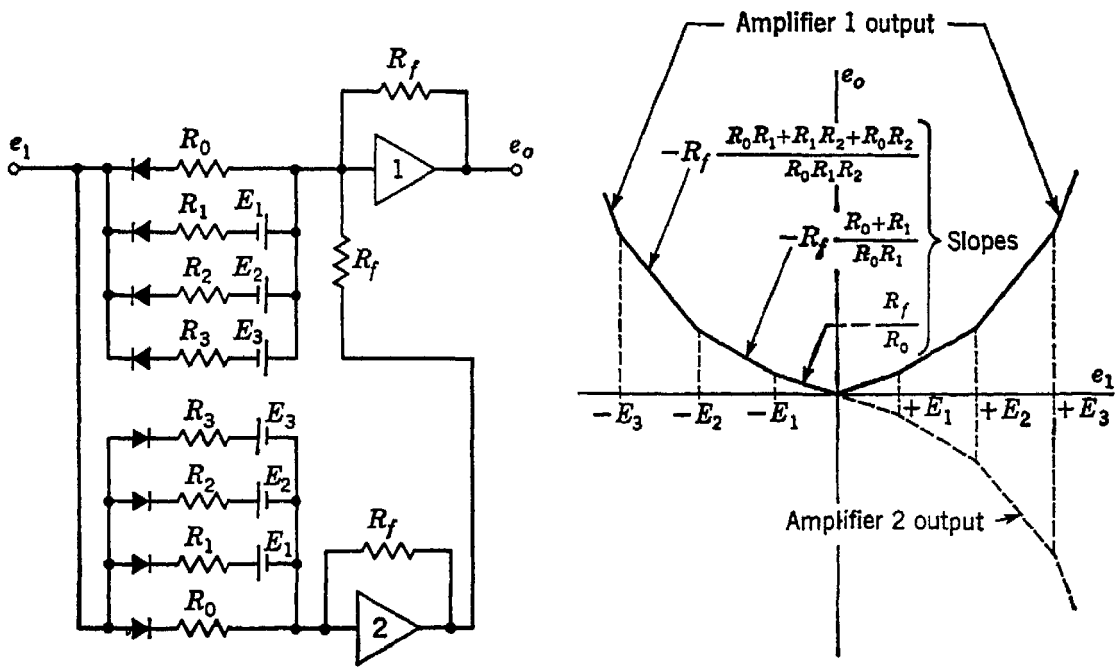



Fig. 3.10 A diode parabola generator

Use of many diodes biased at different voltages makes it possible to synthesize a curve by approximating it with straight-line segments. By using inverters and by adding together appropriate combinations of linear and nonlinear curves, it is possible to generate any single-valued transfer characteristic within the range of the amplifiers. For example, a parabolic transfer characteristic can be produced as shown in Fig. 3.10.

As $e_1$ goes negative, more and more of the input diodes of amplifier 1 conduct, making the effective input resistance progressively smaller. Thus the output voltage becomes larger more and more rapidly, as shown in the second quadrant. Amplifier 2 is similarly affected for positive voltages. Its output is added as an input to amplifier 1, giving an overall approximation to the desired parabola. The accuracy of the approximation is enhanced, of course, by using many diode circuits arranged to cut in or out at closely spaced voltages.

In circuits like these, any small forward voltage drop while the diodes are conducting or any slight reverse current while they are supposedly cut off tends to be obscured, in the former case by the additive voltage drop across the series resistors and in the latter case by the much larger currents in conducting paths in parallel. For the precisions involved in most analog-computer work, these deviations from the ideal can be quite safely neglected. If, however, in some special situation it should be necessary to take these effects into account, measurements can be made on the diodes used, and a small constant resistance can be determined to approximate the conducting diode and a large constant resistance to approximate the nonconducting diode. Of course, diode forward and reverse resistances are both functions of current, but correction for this nonlinearity would be a second-order correction which can be neglected.

The example in Fig. 3.10 suggests that it should be possible to make use of all four combinations of the two polarities of the input variable and the two polarities of diode connection to give four types of line segments. The four possibilities are shown in Fig. 3.11. The polarity of the break-point potential $E$, as well as the diode connection, has been reversed in $a$ and $b$ and also in $c$ and $d$ to suggest that slopes in the four quadrants may be obtained. Note that $-e_1$ does not mean a negative voltage, but simply a voltage of polarity opposite to $e_1$, whether this is positive or negative. Resistor $R_G$ acts to keep the summing node at ground potential when the input diode is cut off. A very versatile function generator can thus be created using (1) amplifiers to supply both polarities of input, (2) a group of diodes whose connections can be reversed, and (3) a group of bias voltages having adjustable magnitudes and reversible polarities. Such generators are not restricted to monotonic functions but can have slopes that increase, decrease, or change sign from one break point to the next. The slope of any line segment is determined by all the input paths that are conducting in that interval; that is,

$$\text{Slope} = R_f\left(\pm\frac{1}{R_1} \pm \frac{1}{R_2} \pm \frac{1}{R_3} \pm \cdots\right) \tag{3.9}$$

The minus sign is used with each summing resistor that is driven directly

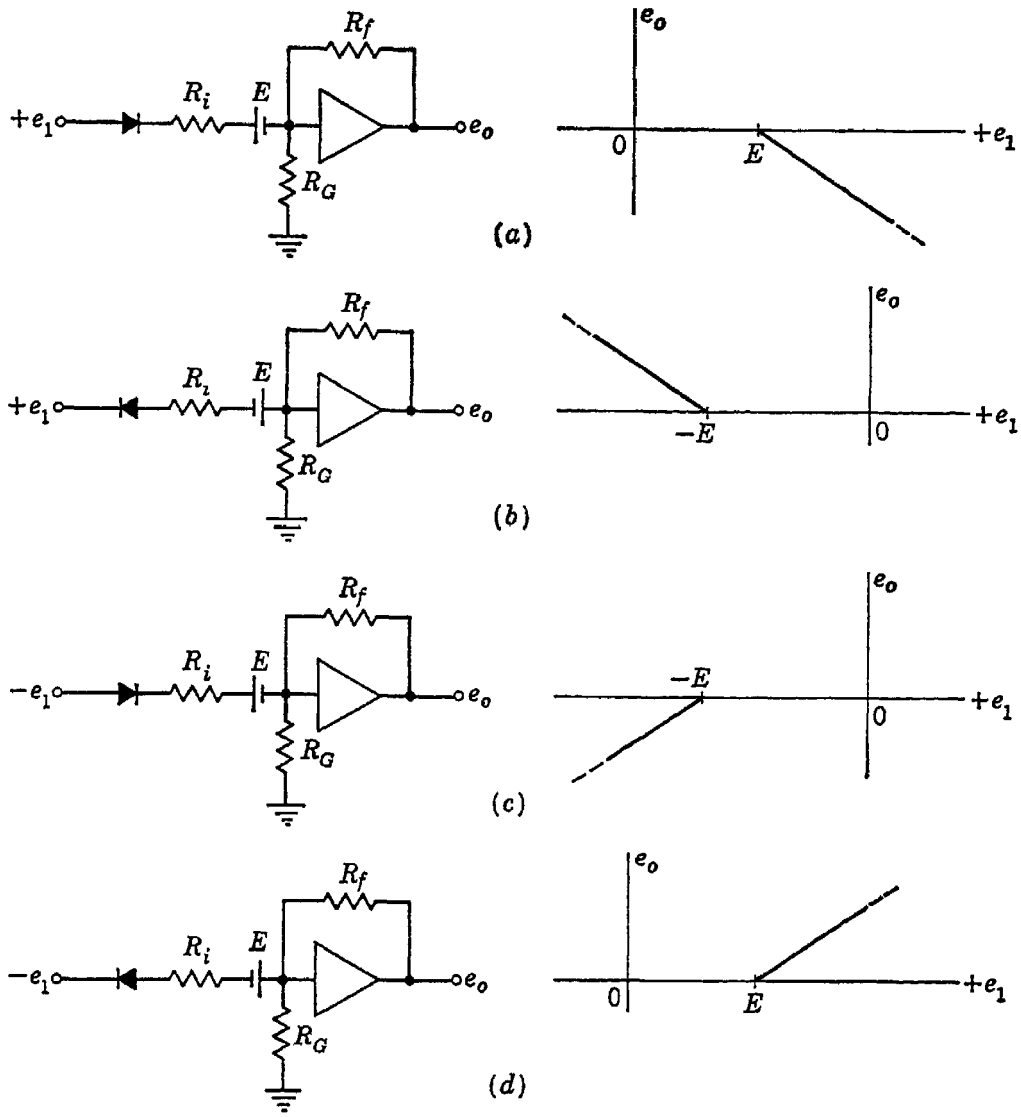



Fig. 3.11 Generators for four basic types of line segments

by the independent variable and the plus sign with each resistor driven through a sign-inverting amplifier by the independent variable.

For example, a circuit to generate a semicircle might be arranged as in Fig. 3.12. Segment 1 has the steepest slope, which is $+R_f/R_1$. As succeeding diodes driven by $+x$ come into conduction, each one contributes a negative component to the slope, until finally segment 8 has a slope of

$$R_f\left[\frac{1}{R_1} - \left(\frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4} + \frac{1}{R_5} + \frac{1}{R_6} + \frac{1}{R_7} + \frac{1}{R_8}\right)\right]$$

Note that, in order to minimize the maximum error, a bias voltage $E_1$ has been inserted in series with $R_1$, so that a slight offset is introduced at $x = 0$.

Provision of suitable bias power supplies may be an awkward problem with diode limiter circuits, since they may have to be isolated from

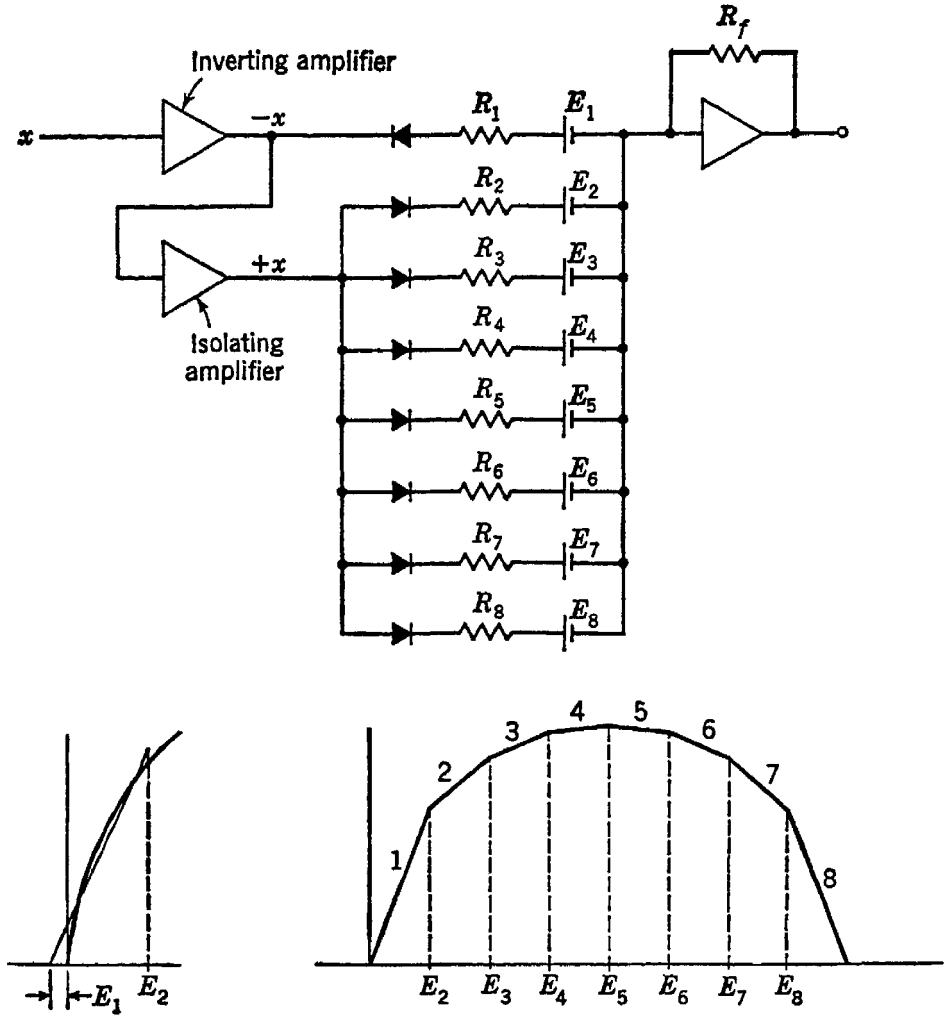



Fig. 3.12 A semicircle generator

ground. Dry cells or dry batteries can be used, with potentiometers to adjust the bias voltage, but they have many disadvantages, and electronically regulated power supplies are often preferable. Isolating them from ground, however, may introduce noise problems as well as shock hazards. A connection that avoids these difficulties is shown in Fig. 3.13. Current flows to the summing point from the signal source when the anode potential of the diode goes above ground in Fig. 3.13*b*. The signal voltage at which this happens is determined by $R_1$, $R_2$, the potentiometer setting, and the polarity of the potentiometer voltage. The potentiometer setting is not directly proportional to the break-point voltage unless the potentiometer has very low resistance, and in that case, of course, the power loss in the potentiometer is large. For this reason it is not practical to calculate the setting needed for a particular break point. Instead, a high-precision potentiometer is adjusted to the proper voltage, and the output of the function generator is balanced against it with a null voltmeter until the desired break point is obtained.

In the circuit of Fig. 3.13*b*, the slope is determined by $R_1$, $R_2$, $R_3$, $R_4$,

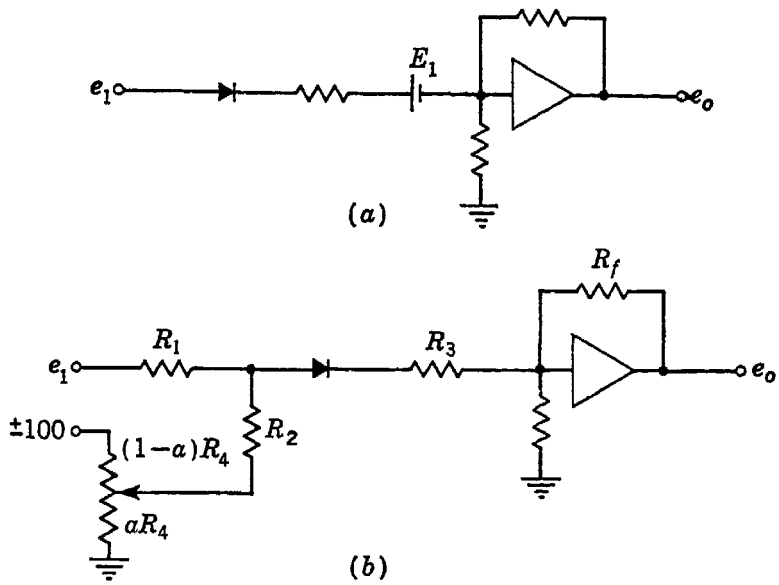



Fig. 3.13 A method of furnishing bias voltage: (a) simplified circuit, (b) circuit using grounded power supply

$R_f$, and the potentiometer setting $a$. To minimize any effects of the slope setting upon the break-point voltage, $R_3$ is used to adjust the slope, in a circuit such as that of Fig. 3.14. Like the break-point potentiometer, the slope potentiometer is also not direct-reading and can be adjusted by first setting the desired break point and then altering the slope until, with an appropriate input signal $e_1$, the desired output voltage is obtained. A few readjustments of both potentiometers may be needed because of interactions between them, although, theoretically at least, the break-point setting is not affected by the adjustment of $R_3$.

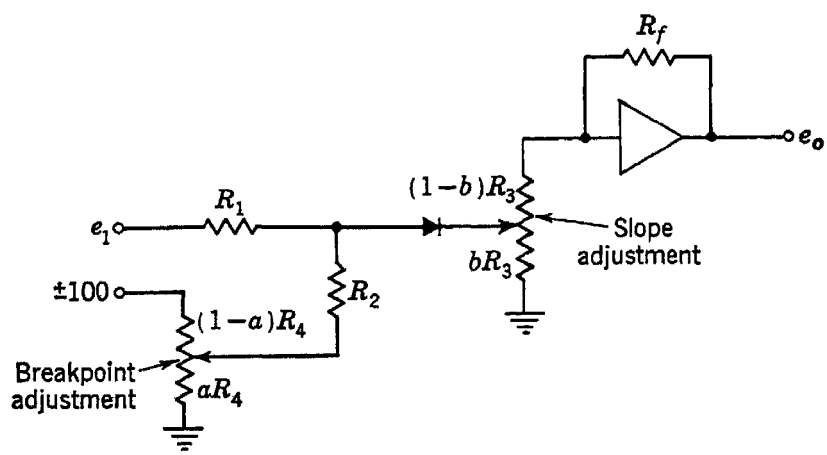



Fig. 3.14 Circuit showing line-segment break point and slope adjustments

Although the foregoing discussion has been largely concerned with input series limiting, the other forms are equally usable, and the reader can gain added understanding by devising circuits analogous to those of Figs. 3.10 to 3.14 for the other types of limiters, i.e., input shunt, output

shunt, or output series limiters. Many ingenious versions of these circuits have been devised by various computer laboratories and manufacturers. For example, an interesting input shunt limiter circuit is shown in Fig. 3.15. This is a function generator used at the Jet Propulsion Laboratory of California Institute of Technology and described by Bruns.[6] The 10k potentiometers control the break-point voltage, and the 200k units control the slope. With the 200k potentiometer set at its electrical center, the currents flowing to the two amplifiers are

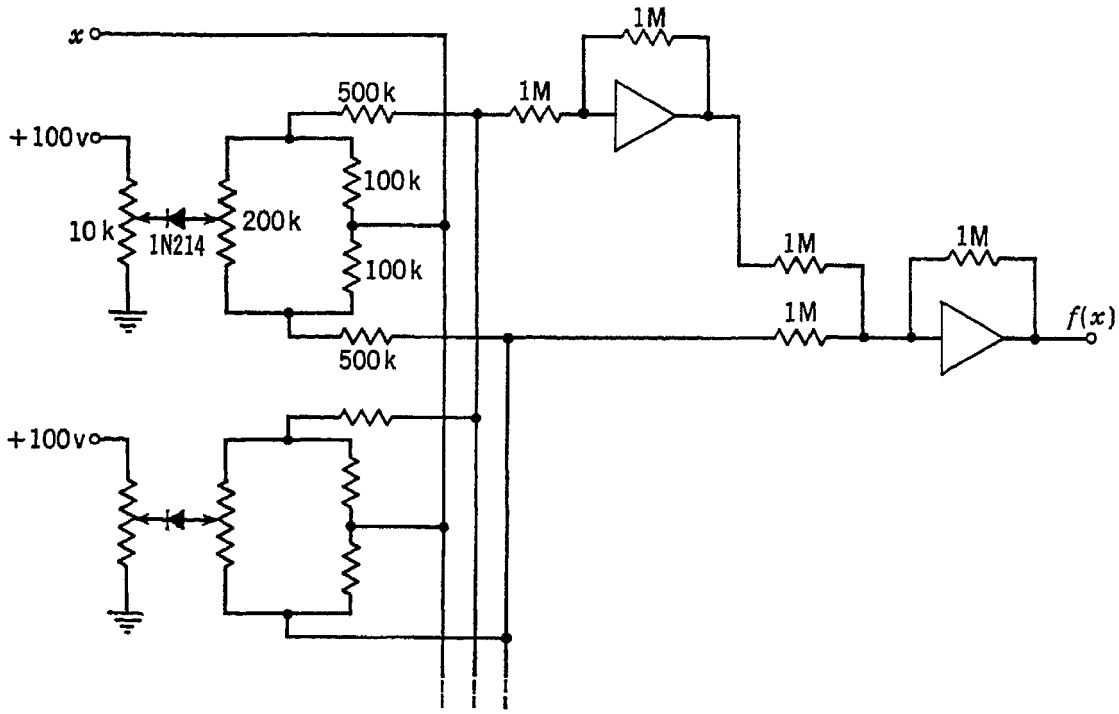



Fig. 3.15 Practical circuit for input shunt limiting

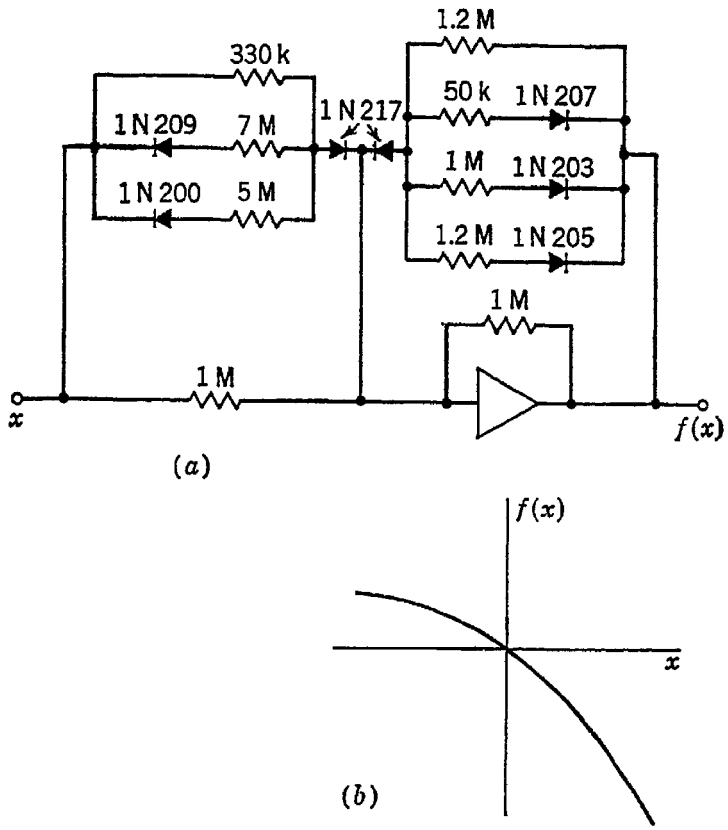
identical, and there is no contribution to the output even if the diode conducts. Unbalancing the resistive bridge by changing the potentiometer setting increases the current to one amplifier and decreases that to the other, causing the output voltage to rise or fall with increasing $x$, depending on the direction of potentiometer offset. Other segments are controlled by similar circuits which supply currents to the two amplifiers. Similar circuits are also used in certain commercially produced computers, but a unique feature of this one is its successful employment of a silicon junction diode as the switching element.

The art of manufacturing semiconductor diodes has advanced to such a point that it is now possible to construct diodes with specified Zener break-down voltages anywhere within a wide range. This gives rise to the interesting possibility of building fixed function generators by using the Zener voltages as the break points. The circuit of Fig. 3.16, also due

to Bruns,[6] approximates the function

$$f(x) = -\left(\frac{x^2}{200\sqrt{2}} + \frac{x}{\sqrt{2}}\right) \tag{3.10}$$

As the input voltage $x$ goes positive, the 1N200 diode breaks down and conducts in the reverse direction, followed at a higher voltage by the 1N209. Thus the effective input impedance is lowered, and the output




| Diode type | Zener voltage |
|---|---|
| 1 N 200 | 7.5 |
| 1 N 203 | 13.0 |
| 1 N 205 | 20.0 |
| 1 N 207 | 30.0 |
| 1 N 209 | 43.0 |
| 1 N 217 | 230.0 |

Fig. 3.16 Function generator using Zener voltages for break points

increases in magnitude more rapidly. Of the 1N217s, only the left one is conducting, the Zener voltages for these units being high enough so that only forward conduction can occur for the voltage levels used with the circuit. When $x$ goes negative, the right 1N217 conducts, and the 1N203, the 1N205, and the 1N207 successively break down, thus lowering the feedback impedance and decreasing the slope of the output.

Although the break-point voltages in such a function generator are determined by the Zener voltages of the diodes, the slopes can be adjusted by potentiometers as in the more flexible types of function generators. The simplicity of this type of function generator is a strong point in its favor, although temperature instability of the Zener voltages must be guarded against. The development is promising and may prove very useful to computer users.

Before leaving the subject of diodes, let us examine a number of other useful functional relations easily realized with diodes. Such effects as dead space, Coulomb friction, and backlash are often present in physical systems and are easily simulated on the computer. Figure 3.17 shows a

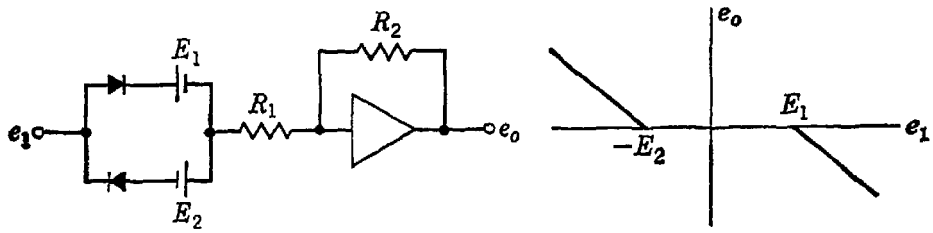



Fig. 3.17 Dead-space simulator

useful dead-space simulator. Of course, the slopes of the two active portions can be made unequal by putting suitable values of unbalancing resistors in series with the diodes.

Coulomb friction is another effect easily simulated. This is the familiar dry friction, first studied by Coulomb, who observed that two surfaces in contact cannot slide relative to each other as a result of applied forces less than a certain value, but, once in motion, slide with a frictional force approximately independent of their relative velocities. This phenomenon is unlike viscous friction, which is characterized by a force proportional to the relative velocity. An amplifier operating "wide-open", that is, without feedback, gives a good analogy to Coulomb friction. If the amplifier alone does not saturate abruptly enough or at the desired level, limiting diodes can be used at the output, as shown in Fig. 3.18.

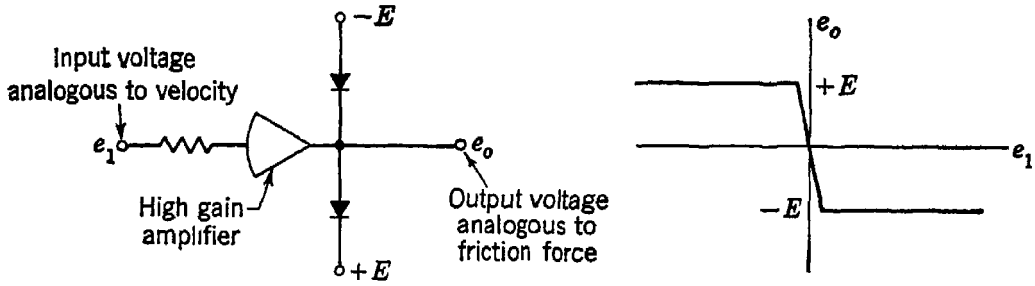



Fig. 3.18 Coulomb-friction simulator

In this circuit the friction after the motion starts is the same as the limiting value at which it just starts. More complex circuits can be devised in which the sliding friction is less than the sticking friction; this is the situation more commonly encountered in practice.

The backlash simulator is a circuit in which the output ceases to follow the input for a certain interval whenever the rate of change of the input changes sign. The transfer characteristic of this circuit displays a hyster-

esis loop, as shown in Fig. 3.19. The circuit of Fig. 3.20 will provide this behavior. Diode 1 conducts if $-(e_1 - e_o) > E$ and continues to conduct until $-(e_1 - e_o) = E$. If the $RC$ time constant of the integrator is made small (usually by making $R$ small), the integrator output builds up very rapidly to the value $e_o = E + e_1$. Similarly, diode 2 conducts if $-(e_1 - e_o) < -E$ and continues to conduct until $-(e_1 - e_o) = -E$. Thus, along the part of the curve corresponding to increasing values of $e_1$ we have

$$e_o = e_1 - E \tag{3.11}$$

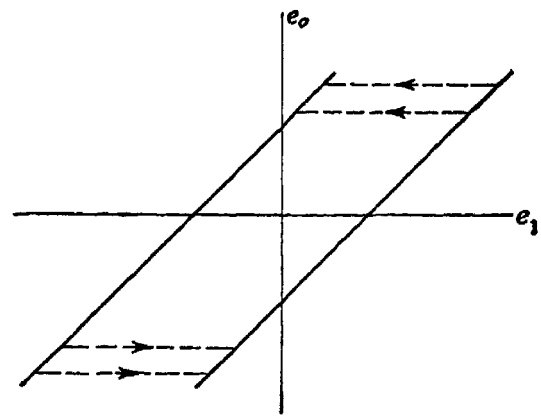



Fig. 3.19 Transfer characteristic exhibiting hysteresis

and along the part corresponding to decreasing values of $e_1$ we have

$$e_o = e_1 + E \tag{3.12}$$

In the intermediate range of inputs where

$$-E < -(e_1 - e_o) < E \tag{3.13}$$

neither diode conducts, and, since there is no current into the integrator, its output voltage remains constant at the last value it had before the rate of change of $e_1$ changed sign. The integrator thus acts as a memory

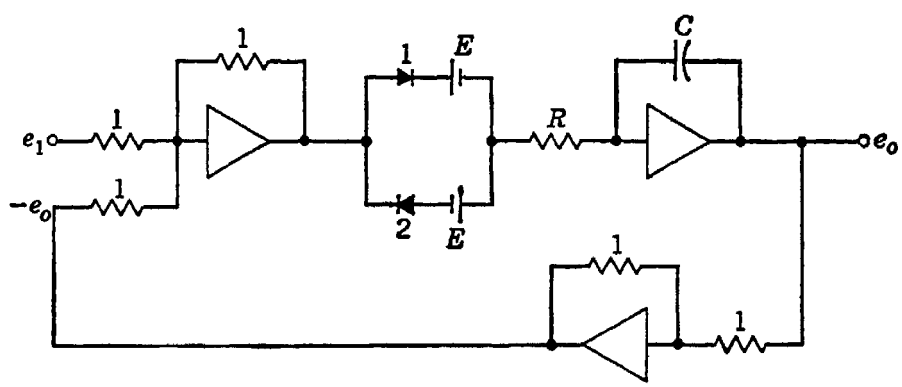



Fig. 3.20 A hysteresis simulator

or storage device to retain the output voltage while $e_1$ traverses a dead zone. The total change in $e_1$ must be $2E$ volts before $e_o$ begins to follow the new direction of change of $e_1$.

It should be noted that, although this circuit simulates very well the backlash in a gear train, it does not display the same behavior as a mag-

netic hysteresis loop. The magnetic hysteresis curve can have minor loops, since there are no dead zones in which the input can vary without affecting the output.

Circuits similar to these but using the saturation characteristics of the amplifier instead of diodes for limiting have been described by Howe.[7]

**Function generation with linear potentiometers.** Another method of generating an approximating function is by the use of linear potentiometers. There are at least three ways in which this can be done:

1. Linear potentiometers and fixed resistors of suitable sizes can be combined in iterative networks to fit a function expressed as a continued fraction.[8]

2. Combinations of one or more linear potentiometers, with or without tap points, and one or more fixed resistors can sometimes be arranged to approximate a function.

3. Tapped linear potentiometers with appropriate voltages introduced at the tap points can approximate a function.

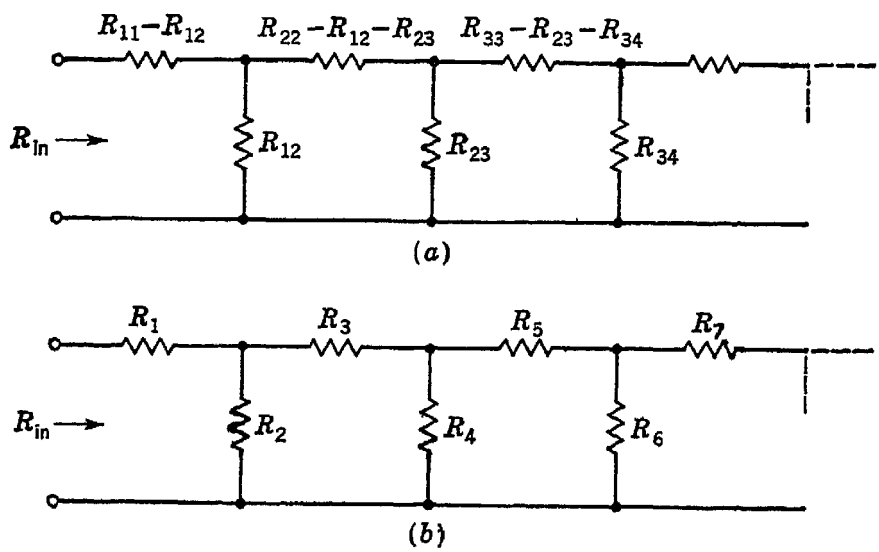



Fig. 3.21 Ladder networks

The iterative-network method is based upon the continued-fraction expansion for the input impedance of a ladder network. From the networks of Fig. 3.21 it can readily be shown that

$$R_{in} = R_{11} - \cfrac{R_{12}^2}{R_{22} - \cfrac{R_{23}^2}{R_{33} - \cfrac{R_{34}^2}{R_{44} - \cdots}}} \tag{3.14}$$

for the network of Fig. 3.21*a*, and that

$$R_{in} = R_1 + \cfrac{1}{\cfrac{1}{R_2} + \cfrac{1}{R_3 + \cfrac{1}{\cfrac{1}{R_4} + \cfrac{1}{R_5 + \cfrac{1}{\cfrac{1}{R_6} + \cdots}}}}} \tag{3.15}$$

for the network of Fig. 3.21*b*.

It is often possible to express a function in continued-fraction form,* and the individual parts of the resulting expression may then be identified with resistances in one of the networks of Fig. 3.21. An example is a circuit to form $x/\tan x$ from the expansion:

$$\frac{x}{\tan x} = 1 - \cfrac{x^2}{3 - \cfrac{x^2}{5 - \cfrac{x^2}{7 - \cdots}}} \tag{3.16}$$

The network of Fig. 3.22 follows at once, since $x$ can be identified as the mutual resistance between adjacent loops, and 1, 3, 5, . . . are the self-resistances of the loops. The resulting network has an input resistance equal to the desired function when the resistance values of Fig. 3.22

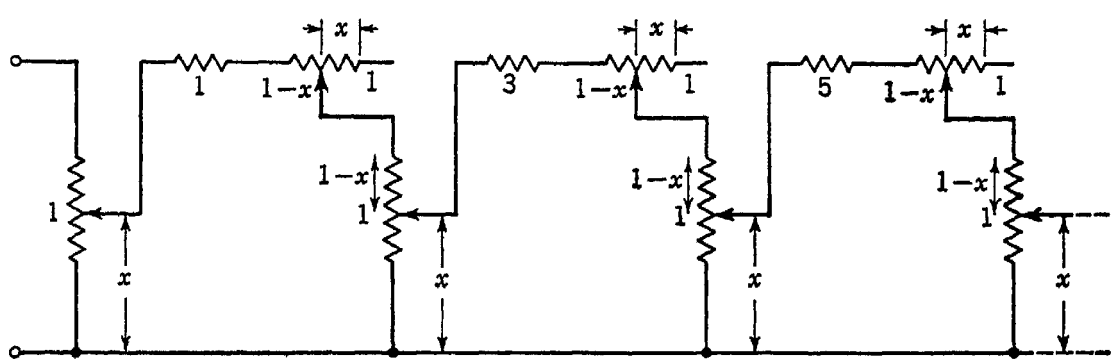



Fig. 3.22 Iterative network to form $\dfrac{x}{\tan x}$

are in ohms. These resistances might be better expressed in megohms and the whole network used as a variable feedback resistance around a computer amplifier having 1-megohm input impedance. Then the overall transfer characteristic $e_o/e_1$ will be the negative of $x/\tan x$.

* See pp. 395–412 of Hildebrand, listed among the general references at the end of this chapter.

The continued-fraction method is not well suited to general-purpose computers since it lacks flexibility. A large variety of sizes of potentiometers and resistors would be needed to set up more than a few functions. Furthermore the method has the speed limitations of all electromechanical systems in which a common shaft drives many elements. Although the convergence of a continued-fraction form may be more rapid than that of a simple polynomial, thus requiring fewer potentiometers than a polynomial function generator, this advantage is offset in practice by the common necessity that the potentiometers have different resistance values.

A second method of function generation with linear potentiometers consists in employing potentiometers and resistors in perfectly general configurations to produce an input resistance that matches the desired function either exactly or approximately. Although a few resistance networks have been developed to produce one or another function, general methods of synthesis of such systems have not been developed, nor, indeed, have such networks been extensively analyzed. Any given network can, of course, be analyzed by a straightforward application of simple network theory, but the work may be tedious and arduous for all but the simplest systems, and the results are usually not easy to generalize. Another analytic procedure is to examine systematically the functions that can be generated by networks of resistances and linear potentiometers, starting with the simplest possible systems and then elaborating them. For example, Fig. 3.23 shows the six possible connections of the three terminals of a single linear potentiometer, together with a sketch of $R_{in}(x)$ for each. (Other connections are possible in which one potentiometer terminal is left free or connected to one of the others.) Each of these circuits can be modified in such ways as the following:

1. Connect terminals 1 and 2.
2. Connect terminals 2 and 4.
3. Connect a fixed resistance in series with terminal 1 or 3.
4. Connect a fixed resistance between terminals 1 and 2.
5. Connect a fixed resistance between terminals 2 and 4.
6. Connect a fixed resistance between terminals 1 and 3.
7. Carry out a nontrivial combination of any of the foregoing.

A systematic study of even these few combinations is a large task, and, if we enlarge the scope of our study to include any number and size of fixed resistances and of linear potentiometers, the magnitude of the cataloguing problem becomes enormous. Therefore, although a few simple and sometimes useful functions are known, no extensive tabulations have been made. A technique for dealing with such networks in a more sophisticated mathematical manner has been developed by Levenstein.[9]

It is also possible sometimes to devise resistance networks that have

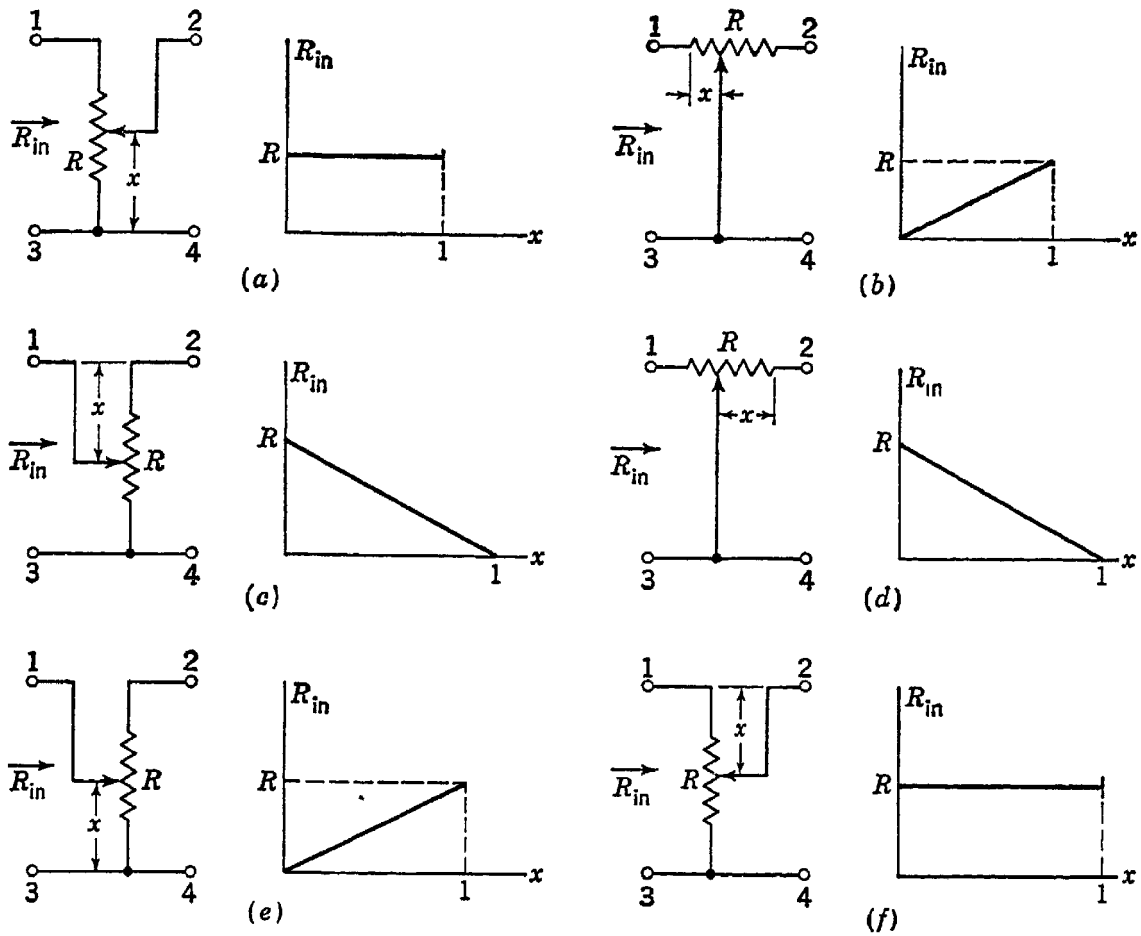



Fig. 3.23 Six linear potentiometer connections and their input resistance curves

transfer functions approximating the desired function.[10] Some examples are shown in Fig. 3.24.

Another example of the use of resistance networks is shown in Fig. 3.25, in which simultaneous adjustments of the input and feedback resistances used with a computing amplifier yield a transfer ratio approximating the function tan $\theta$. Except in the neighborhood of 90°, where amplifier saturation introduces large errors, this network has angular errors of less than 2′.[11]

A third method of using potentiometers for function generation is to apply fixed voltages at tap points along the resistance winding. Then, in the absence of loading, the voltage between any adjacent pair of taps varies linearly along the potentiometer. This method has its obvious drawbacks but, if used with care, it can be very helpful. The voltage gradient between two adjacent tap points must not be permitted to be so large as to endanger the potentiometer. Effects of loading cannot be ignored, and departures from linearity will be found to be different over different parts of the approximation. Nevertheless, the method has been widely used. The voltages applied to the tap points can be obtained from individual potentiometers placed in parallel across the positive and

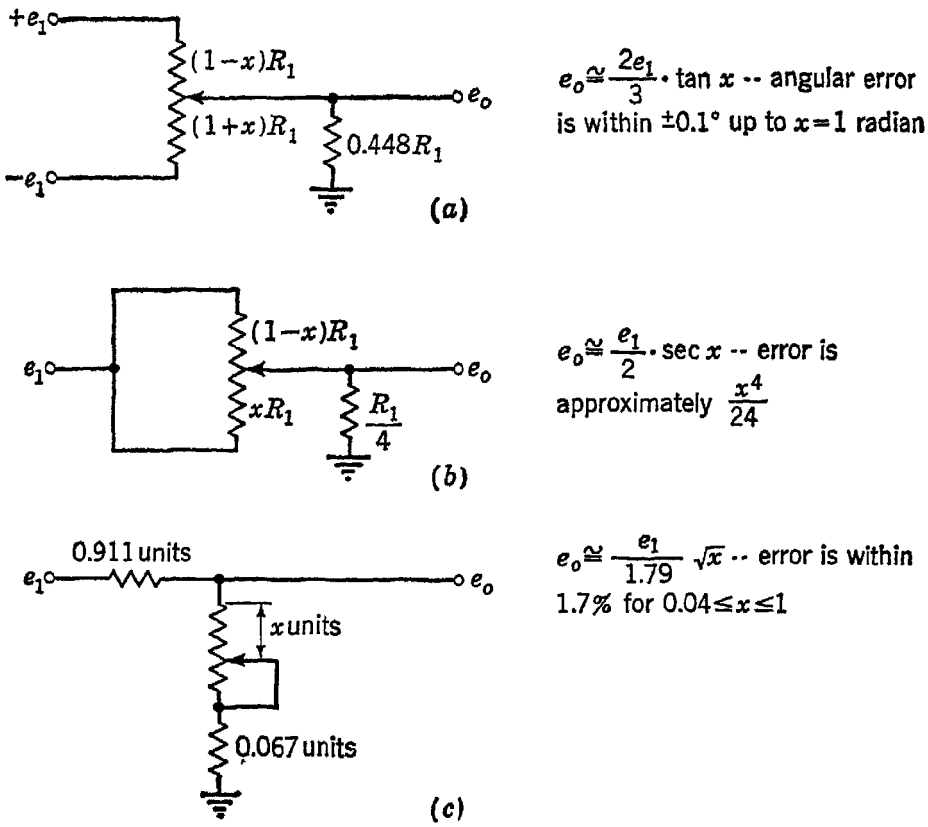



Fig. 3.24 Some networks of resistances and linear potentiometers for function generation

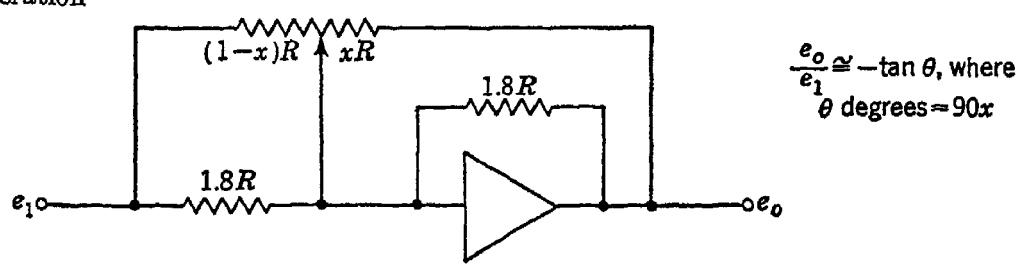



Fig. 3.25 An approximating circuit for tan $\theta$

negative reference supplies, as in Fig. 3.26*a*, from a series group of potentiometers, as in 3.26*b*, or from variable resistances in series with the tap points, as in 3.26*c*.

Setting up the proper voltages on a tapped potentiometer can best be done by measuring and adjusting them. Although the proper settings for the individual voltage-adjusting potentiometers can be calculated, the burden of the arithmetic is considerable. Furthermore, since these adjustment potentiometers are seldom precisely calibrated devices, they could not be set to the calculated values without being measured. The voltage adjustments are of course not independent, and this may necessitate considerable readjustment before the final settings are obtained. One ingenious way in which this problem is sometimes avoided is to start from one end of the tapped potentiometer and adjust each voltage to its proper value while the next tap beyond is held at its proper value by

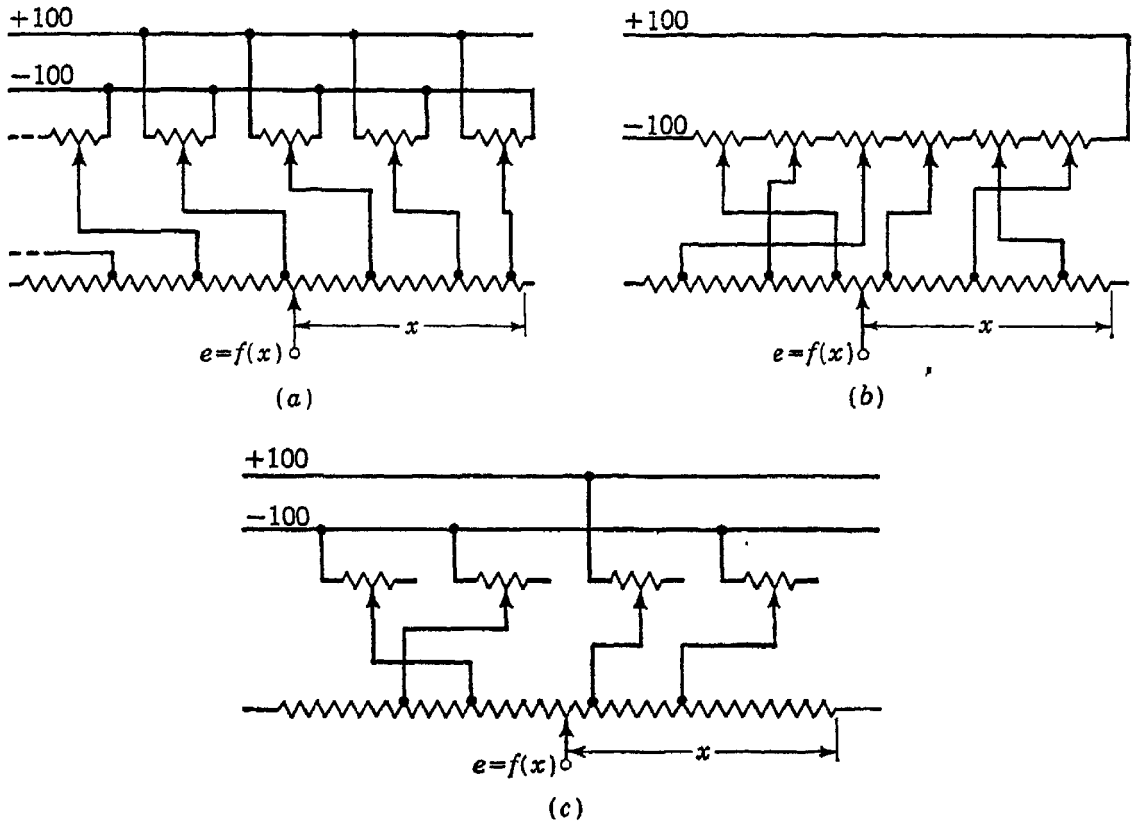



Fig. 3.26 Ways of providing tap voltages

being driven from a very low-impedance source, e.g., one of the computer amplifiers.

As already mentioned, load current drawn by the potentiometer contact arm causes the voltage output to be a nonlinear function of the shaft displacement, in some such manner as the example of Fig. 3.27. This

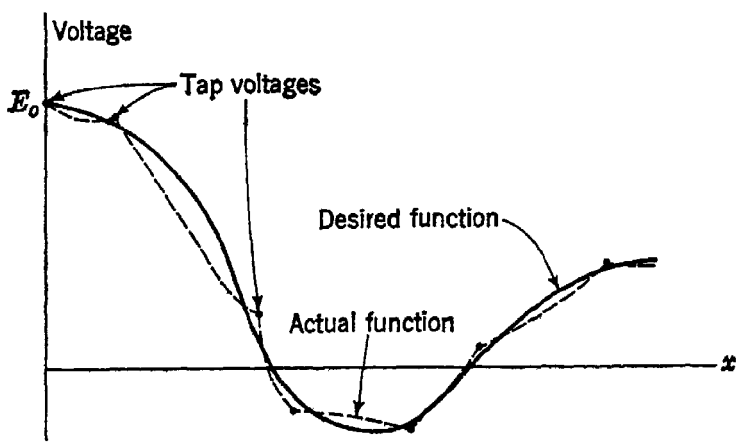



Fig. 3.27 Departure from piecewise linearity due to loading

effect has been analyzed by Korn.[12] Between tap points the voltage is smaller in absolute value than the voltage corresponding to a truly linear variation from one tap to the next. We have already noted that, for optimum approximation by means of linear segments, the break points should not lie directly on the curve being approximated. The loading

effect will cause the optimum tap voltages to be slightly different from those which would be obtained if the approximating segments were truly linear. With a large enough number of tap points, however, the errors due to loading can be made very small.

## 3.5 Some examples of function generators based upon the true function

The devices discussed in the preceding sections all generate approximations to the desired function even if the apparatus itself is free from errors. A second class of function generators is based upon the true function and would produce that function exactly if the equipment were error-free. Of course, the total error is the sum of what we have loosely called "round-off" error and "truncation" error, i.e., errors due to the equipment and to the mathematical approximant, respectively; and the so-called "true-function" generators are not necessarily superior to the "approximate-function" generators. Classification of function generators by their mathematical properties, however, is a more fundamental method than classification by physical properties and should help the reader to evaluate more readily the new techniques and devices he will meet.

**Tapered function potentiometers.** A logical sequel to the treatment of linear potentiometers in Sec. 3.4 is consideration of nonlinear potentiometers. A wide variety of functions can be formed by making the resistance between the contact arm and the fixed terminals vary nonlinearly with $x$, the coordinate of the contact-arm displacement. This nonlinear distribution of potentiometer resistance can be effected in several ways.

1. The turns of resistance wire around the form that holds them can be spaced more sparsely or more densely in the $x$ direction.
2. The cross-sectional area of the resistance wire or the properties of the material composing it can be altered from region to region.
3. The form holding the wire can be shaped so that individual turns vary in length.

Each of these methods makes the increment of resistance per increment of $x$ vary along the potentiometer. Method 3 is the most versatile of these, since the shaped resistance card is easily constructed in a wide range of shapes.

Figure 3.28 shows the pertinent coordinates on a shaped resistance card to be used in a function potentiometer. Evidently we can write

$$\frac{\Delta R}{\Delta x} = 2ry \tag{3.17}$$

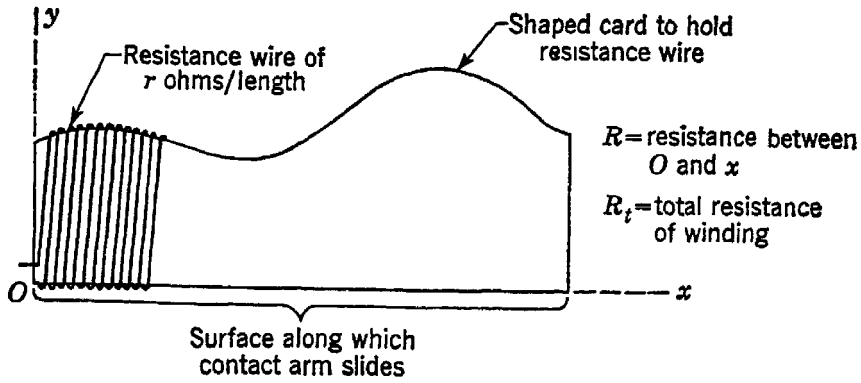



Fig. 3.28 A shaped resistance card

With $E$ volts applied across the potentiometer, the contact-arm voltage, which is to be the desired function of $x$, is given by

$$e_o = f(x) = \frac{E}{R_t} R \tag{3.18}$$

Now

$$\frac{df(x)}{dx} = \frac{E}{R_t} \frac{dR}{dx} \tag{3.19}$$

$$\cong \frac{E}{R_t} 2ry \tag{3.20}$$

Thus

$$y = \frac{R_t}{2rE} \frac{df(x)}{dx} \tag{3.21}$$

That is, the distance $y$ across the card at any point $x$ should be made proportional to the derivative of the desired function at that point.

Such potentiometers are not difficult to make if the function is reasonably well behaved, that is, if it is continuous and if its second derivative is not so large that the turns of wire slip over one another. The mechanics of potentiometer construction are adequately covered elsewhere in the literature and will not be treated here.[13,14] Some sources of potentiometer error should be noted, however:

1. End resistance, a small resistance that may exist between the external terminal and the point of contact of the wiper arm with the last turn of resistance wire that it touches
2. Undesired nonlinearities, which may be due to slight errors in shaping the card, to uneven wire tension during winding, to variation of the wire characteristics from point to point, or to other construction faults
3. Change of resistance with temperature, which may be reduced by use of resistance wire having a low temperature coefficient of resistivity
4. "Granularity", the small stepwise changes of output voltage caused when the wiper arm breaks contact with one turn and makes contact with another

Granularity can be reduced by making the wire size extremely small, but there are limits to this method, for very fine wire is difficult to wind and such windings are fragile and short-lived. Alternatively, the card can be made very large, but there are practical limits to this method, too.

Shaped cards are widely used to generate sine and cosine functions, the potentiometer arm being driven by a servomechanism such as that employed in the multipliers mentioned in Chap. 2. Such a card is shown in Fig. 3.29. It is ordinarily wrapped around in a circle so that points $a$ and $e$ are joined, and $x$ is the angular displacement of the ganged contact arms.

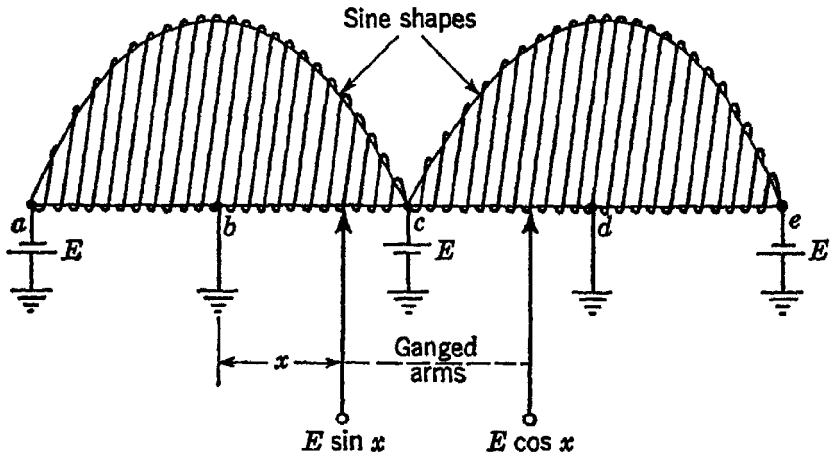



Fig. 3.29 A shaped resistance card to form $E \sin x$ and $E \cos x$

Another method of function generation uses linear potentiometers with a mechanical linkage establishing the functional relation between the wiper-arm position and the independent-variable input. A simple example is the sine (or cosine) generator of Fig. 3.30, in which the distance $x$ and, hence, the voltage at the wiper are made to vary as $\sin \theta$.

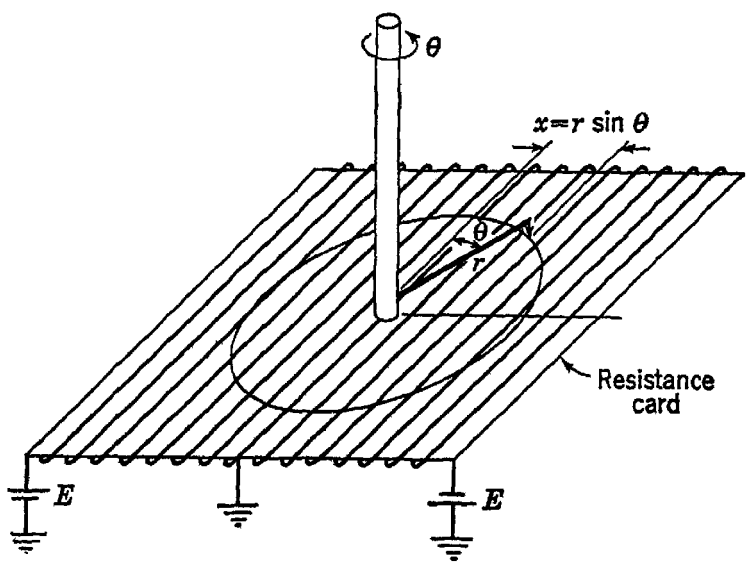



Fig. 3.30 Mechanical linkage used with a linear potentiometer to form $E \sin \theta$

**Mechanically driven curve followers.** Several devices for function generation have been developed on the principle of the mechanical cam. In all of these an output element is positioned in space by the physical dimension of the cam at each value of the input. In the purely mechanical cam, the entire process is mechanical—input, positioning, and output. For electronic-computer applications, various parts of the mechanical system have been replaced by electrical devices of one kind or another; typically, the principal remaining mechanical element is a contour of some sort having the shape of the desired function. This contour might be a conductor or a graph or a photographic film.

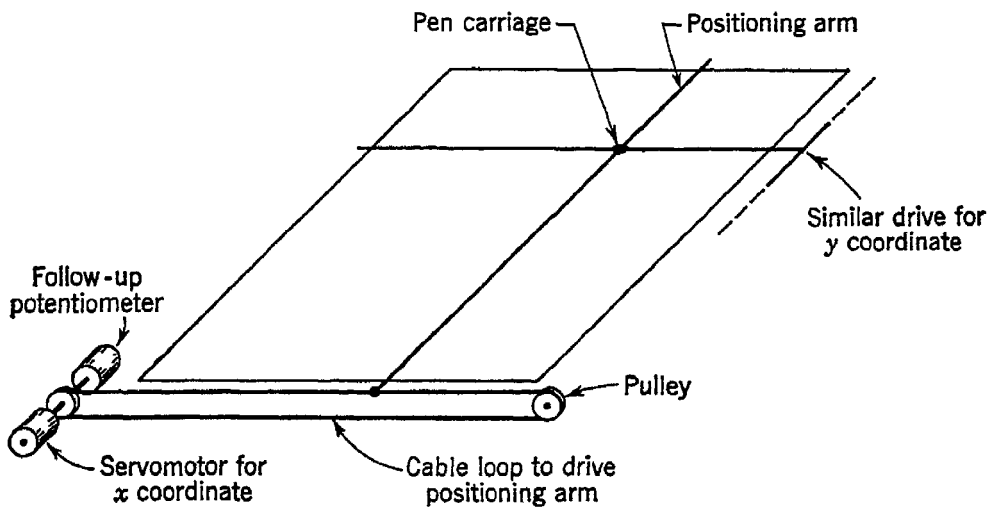



Fig. 3.31 Sketch of an $xy$ plotting table

Adaptations of $xy$ output plotting tables have been successfully used as function generators. A drawing of such a table is shown in Fig. 3.31. As in the servo multiplier, the amplifier that drives the servomotor is supplied by the coordinate input voltage and by the voltage from the follow-up potentiometer and drives the positioning arm to the point at which these two voltages become equal. One such servo system drives the $x$ arm and another the $y$ arm. The pen, which is positioned at the crosspoint of the two arms, draws out the curve of $y$ versus $x$. Modification of an output table for function generation consists in removing the servo system for the $y$ coordinate and mounting a linear resistance element along the under side of the $x$ arm so that it makes contact with a wire glued onto the table in the shape of the desired function. If a voltage is applied across the resistance element, then the output voltage $e_o = Kf(x)$ is obtained from the contour wire. The same method can be used with output plotters in which the paper is wrapped around a drum instead of being laid out on a flat board. With such a device the servo that rotates the drum is used for the input, and the servo that imparts motion parallel to the drum axis is disabled. The resistance element is

mounted parallel to the drum axis and in contact with the contour wire fastened to the drum surface, as in Fig. 3.32. Any good glue or cement may be used to secure the wire to the paper.

This system could be altered or improved in various obvious ways. Among its major shortcomings are the awkward and messy process of gluing the wire in the desired shape and the inconvenience of storing such functions for use at other times. Furthermore, contact between

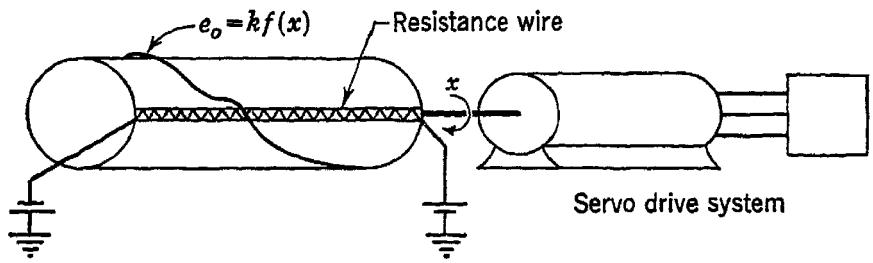



Fig. 3.32 Drum-type function generator

the contour wire and the resistance element is not always satisfactory, and improvement here is desirable. This contact problem on the "home-made" potentiometer can be alleviated by using instead the follow-up potentiometer (ordinarily a high-quality device) that is associated with the $y$-axis servo drive. This requires, however, that the $y$ arm be so positioned that the $x$ arm, the $y$ arm, and the function curve all cross at the same point. Some sensing means must, therefore, be incorporated to provide an input to the $y$-axis servo so that it may be properly positioned. This sensing element should be lightweight, so that it can be easily carried on a low-friction carriage at the crossing of the $x$ and $y$ arms. As the $x$ input varies, the carriage will in general be carried away from the curve. The sensing element should develop a voltage output that is a measure of the position error in the $y$ direction, and this voltage should in turn drive the $y$ servo in such a direction as to reduce the error.

Photoelectric sensing elements have been used in curve followers with mechanical differential analyzers,[15,16] and similar devices are also commercially available for use with electronic differential analyzers. Another curve-following technique that has been successfully used employs an inductive pickup coil to sense the field in the vicinity of a function curve drawn in conducting ink and excited with a radio-frequency signal. The circuit of the sensing element is shown in Fig. 3.33. When the sensing element is positioned directly over the conducting strip representing the function, equal voltages are induced in the two halves of the pickup coil, and $E_{out} = 0$. When the element is slightly displaced, $E_{out}$ increases in

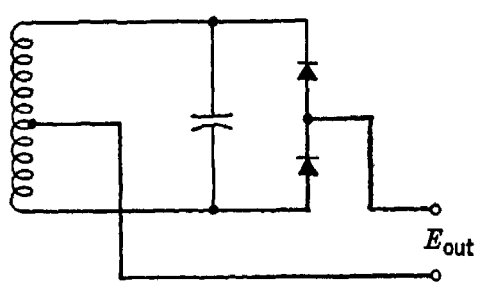



Fig. 3.33 Sensing element for an r-f curve follower

magnitude, its polarity depending on the direction of displacement from the conducting strip. For small displacements $E_{out}$ is approximately proportional in magnitude to the distance between the sensing element and the strip and is thus capable of furnishing an input to the $y$-positioning servo amplifier.

**Photoformers.**[17–20] The photoformer is an almost completely electrical curve follower. Several versions of it have been built, which differ in detail but employ the same principles. The independent-variable analog voltage $x$ deflects an electron beam in a cathode-ray tube, moving the bright spot on the phosphor. A mask formed to the desired-function shape partially covers the face of the tube and prevents the light of the spot from falling upon an external photoelectric tube unless the beam is sufficiently deflected in the $y$ direction. The $y$ deflecting voltage is developed by an amplifier so biased that, in the absence of an input signal, the output voltage fully deflects the beam in the $y$ direction. An input voltage to the amplifier is developed by the photocell whenever the cathode-ray-tube spot is above the edge of the mask and illuminates the cell. This is an excellent example of a feedback system and, like all such systems, must be designed for stability. Stabilization of feedback amplifiers is discussed in Chap. 4; for the present it will suffice to say that the total phase shift around the loop involving photocell, amplifier, and electron beam must be 180°, so that, if the beam rises above the edge of the mask, the photocell and amplifier must respond by producing a voltage just sufficient to pull it down to the mask. Similarly, if the beam falls below the mask, the resultant deflection voltage must elevate it. The $y$ deflecting voltage is a measure of the $y$ dimension of the mask and can be used as the output after being amplified to the scale of computer voltages.

An interesting photoformer circuit is shown in Fig. 3.34. This circuit, which is used in the computer of the University of Michigan's Electrical Engineering Department, is designed to be attached to a conventional oscilloscope. The vertical-deflection amplifier in the oscilloscope is not used, and the photoformer is connected directly to the vertical-deflection plates. The first 6SN7 is connected as a differential amplifier and provides enough output to furnish the vertical-deflection voltage for the oscilloscope. Another 6SN7 differential amplifier, followed by a 6SF5 triode, supplies the drive for the parallel-connected 6SN7 cathode-follower output stage. A polarity-reversing switch is included to allow either $+f(x)$ or $-f(x)$ to be taken as the output quantity. (This switch of course does not change the polarity of the signal applied to the oscilloscope, for this would destroy the stability of the photoformer feedback loop.)

Function masks for photoformers can be prepared simply by cutting

out the function shape with scissors from a piece of opaque paper. The accuracy of this technique is scarcely likely to be very high, however, and more elaborate procedures involving photographic reduction of a large drawing of the function are sometimes used. The area above the

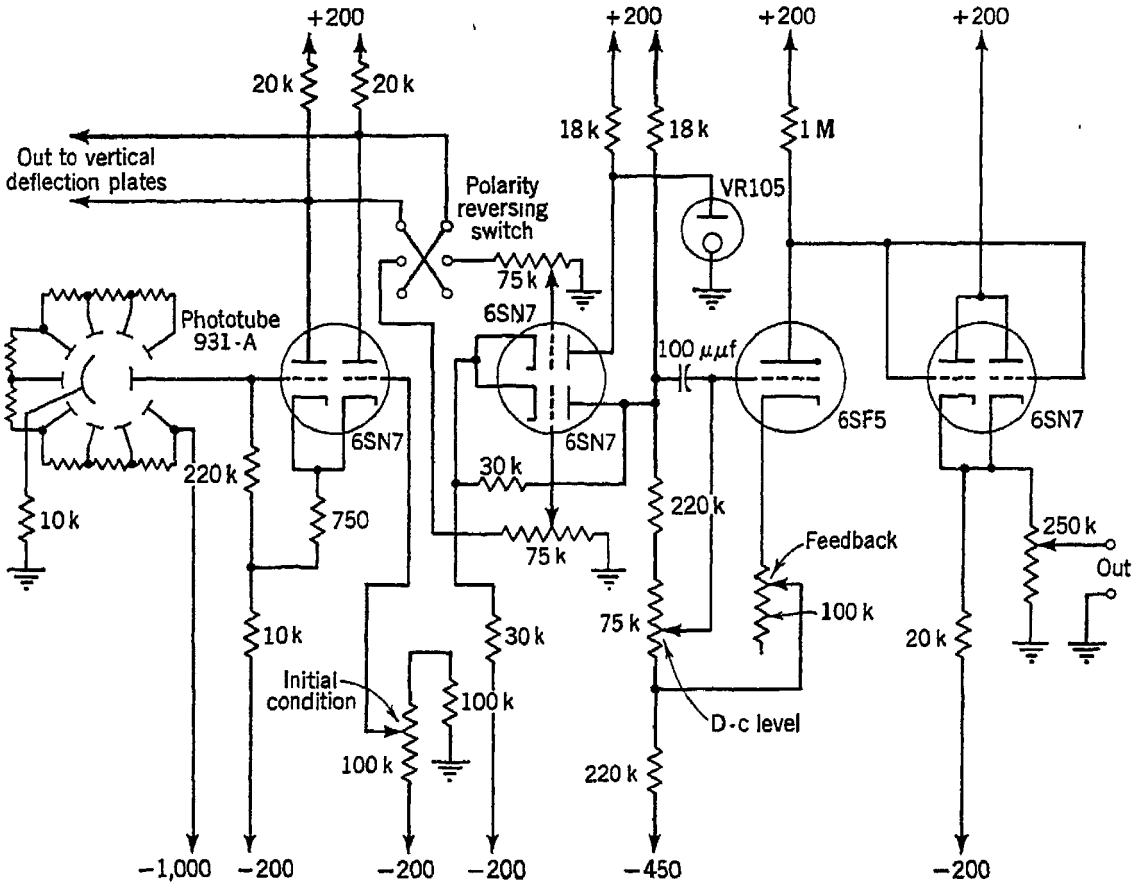



Fig. 3.34 A photoformer circuit used at the University of Michigan

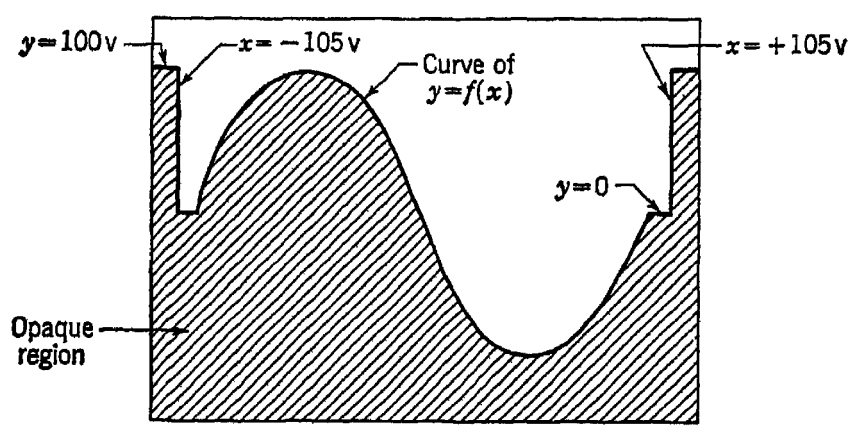



Fig. 3.35 A photoformer mask, with calibration steps

curve is blacked in, so that, in the photographic negative, the area below is black. A negative of the proper size can be made into a slide to be placed in the photoformer. Steps on the sides of the mask can be used to provide calibration points. A typical mask is shown in Fig. 3.35.

## 3.6 Generation of functions of two or more variables

The mathematical theory of the approximation of functions is not nearly so well developed for functions of two or more variables as it is for functions of only one variable. Similarly, the techniques available to the analog-computer engineer or user for the generation of such functions are not at present so versatile, simple, or inexpensive as those for functions of a single variable. Some available methods are the following:

1. Use of three-dimensional mechanical cams
2. Solution of Laplace's equation by means of membranes, electrolytic tanks, or conducting surfaces
3. Analog-computer solution of the partial differential equations that give rise to the desired function
4. Use of diode-type line-segment function generators with bias voltages controlled by the second variable (Meissinger's method)
5. Linear interpolation in $y$ between approximate or exact curves for $f(x,y)$, $f(x,y_1)$, $f(x,y_2)$, etc.

**Three-dimensional cams.** The three-dimensional cam is a surface shaped so that its elevation $z$ above a reference surface is at each point the desired function $f(x,y)$ of the two input variables $x$ and $y$. Although the reference surface may be a plane, it is often the axis of a cylinder. Then the angular rotation of the cylinder is one input variable, and the axial position of the sensing element is the other. The variation in the radial displacement of the sensing element can be converted to electrical form by a suitable mechanical linkage driving a potentiometer. For a special-purpose computer in which the desired function is to be a permanent part, such a cam may be a very suitable element, for it can be made to occupy only a few cubic inches and can be machined to very close tolerances. However, for general-purpose use, the high cost of cams and the fact that a different cam is needed for each function make their use impractical. For either type of computer the speed with which the cam mechanism can operate may be too slow for the requirements of the problem. Furthermore, the class of functions that can be reproduced by cams is restricted to those which do not have large slopes, since a sensing element riding on the surface cannot follow abrupt changes in elevation.

**Use of Laplace's equation.** A class of function generators that has proved useful in a laboratory environment is based upon devices that in effect simulate Laplace's equation over a region. If the regions can be made small and if their boundary conditions are properly adjusted, solutions of Laplace's equation may constitute good approximations to the

desired function. The use of such devices is well known,* one of the most familiar examples in electrical engineering being the simulation of potential distributions in electron tubes with arbitrary element configurations and potentials. In the stretched-rubber-membrane analogy, the height of the membrane is analogous to electric potential, and any desired

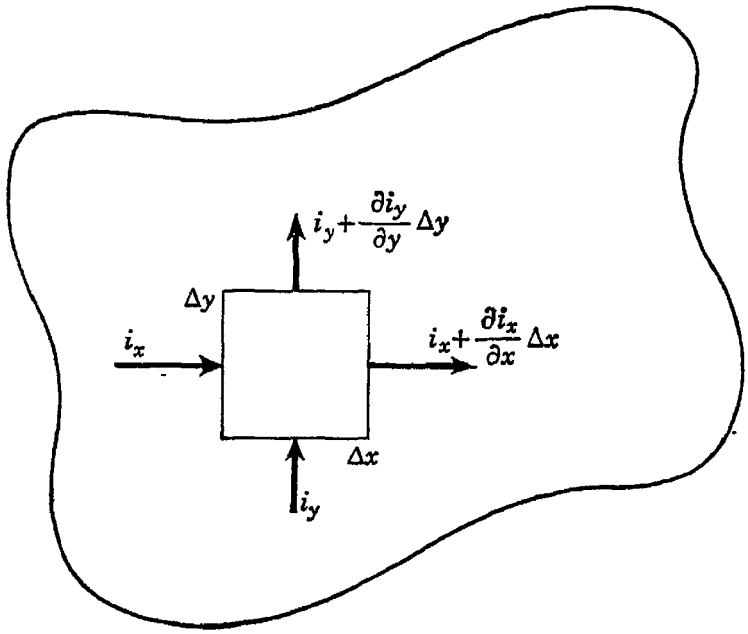



Fig. 3.36 Current flow across an element of a uniform surface

potential distribution is simulated by distorting the sheet. Balls rolling down the sheet may be used to trace electron trajectories. Neither the rubber membrane nor the electrolytic tank is well suited for use with the general-purpose differential analyzer, however, and the resistive surface has been found to be more satisfactory.[21]

Consider an element of area $\Delta x\ \Delta y$ on a uniformly resistive surface with arbitrary boundary potentials, as in Fig. 3.36. Evidently,

$$\left(i_x + \frac{\partial i_x}{\partial x}\Delta x\right)\Delta y + \left(i_y + \frac{\partial i_y}{\partial y}\Delta y\right)\Delta x - i_x\,\Delta y - i_y\,\Delta x = 0 \quad (3.22)$$

and

$$\frac{\partial i_x}{\partial x} + \frac{\partial i_y}{\partial y} = 0 \quad (3.23)$$

But the current density $i_x$ is given by

$$i_x = -\frac{1}{r}\frac{\partial v}{\partial x}$$

where $r$ is the surface resistance in ohms per square, and $v$ is the potential at any point. Substituting this result and a similar result for $i_y$ gives

* See pp. 337–380 of the book by Soroka, listed among the general references at the end of this chapter.

Laplace's equation:

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = 0 \tag{3.24}$$

The uniformly resistive sheet, therefore, constitutes a true-function generator when the desired function satisfies Laplace's equation and constitutes an approximate-function generator for other functions. The use of the device as an approximate-function generator is its more interesting application in general-purpose-computer work, since more often than not the function to be generated does not satisfy Laplace's equation.

The usual way of using resistive sheets for functional approximation is to plot contours of constant $f(x,y)$ on the resistive surface and then to paint over these lines with a conducting silver paint.[22] When the lines are driven by voltages proportional to the constant value of $f(x,y)$ along each one, the surface between two adjacent lines performs an electrical interpolation between the line potentials. An infinite-impedance probe, positioned by $x$ and $y$ input servo systems, detects the interpolated voltage at the desired point. The method has both mathematical and physical shortcomings, which may, however, be sufficiently minimized to be negligible in many instances. From a mathematical point of view, the difficulty is that the interpolation is by Laplace's equation rather than by the equation of the function. Although for some functions it might be possible to perform a mathematical analysis of the error due to this source, in practice it usually suffices to prepare the surface with several equipotential contours and then to scan it with the probe to see whether the maximum errors are within the desired limits. If in some region of the surface the error is too large, the equipotential line passing through the point of maximum error should be silvered and driven with its proper voltage. It is also sometimes possible to improve the interpolation accuracy by driving the equipotential lines with variable voltages instead of fixed voltages, another function generator being used to provide an appropriately varying voltage.

The physical shortcomings of the system are (1) the difficulty of preparing uniform plates of the right resistivity, (2) the difficulty of measuring the voltage without scratching the surface or drawing current, and (3) the difficulty of making independent adjustment of the exciting voltages for the silvered lines. Although resistive plates can be made by the user, the process is exacting. Such plates are now commercially available with resistivities of about 20,000 ohms per square and uniform within 5 per cent or better. Too low a resistivity is undesirable because of the load placed upon the driving sources for the equipotential lines, and too high a resistivity is undesirable because the loading caused by the probe alters the output voltage. Probe loading can be virtually

eliminated, however, by a circuit such as that of Fig. 3.37, in which the second amplifier supplies the current to the summing point of the first and no current is required from the probe.

Ideally the voltages for driving the equipotentials should be obtained from very low-impedance sources, but as a matter of practical necessity they are more usually obtained from potentiometers placed across the reference voltage of the computer. Thus, as each line is set to its correct

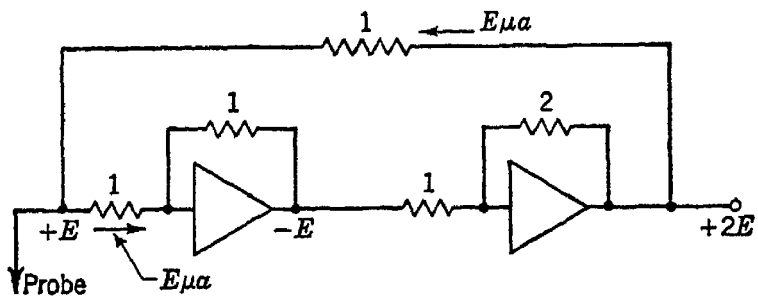



Fig. 3.37 An "infinite-impedance" probe

potential, the load drawn from neighboring lines varies and their potentials change. Several adjustments of voltage may be necessary before the right potentials are obtained on all lines.

**Solution of the partial differential equation.** Another method that could, at least in theory, be used for generating a function of two variables consists in determining the partial differential equation for which the desired function is the solution and then making a computer setup for this equation with its proper initial or boundary conditions. This method is rarely practical, however, because of the difficulty of setting up partial differential equations on the computer. Moreover, for functions in which the two independent variables are not proportional to time, the computer setup can be greatly complicated by the necessity of performing generalized integration with respect to $x$ and $y$ instead of integration with respect to time. However, the method might prove useful for a partial differential equation that is separable into ordinary differential equations and in which generalized integration is either not necessary or not difficult.

**Line-segment function generators with variable bias.** One of the more useful methods is due to Meissinger;[23] this consists of the use of a conventional diode-type line-segment generator with the bias voltages for the segments controlled by one of the variables or by some function of one of the variables. The method is not completely general, but does allow the generation of a wide range of functions. In effect it causes a straight-line-segment approximation of $f(x,y_0)$ to be moved to new positions for other values of $y$ in such a way that the individual straight-line segments are displaced parallel to themselves and changed

in length. In the simplest case, the break points move along straight lines radiating from the origin; other paths are possible, however.

Consider first the elementary diode-function-generator circuit of Fig. 3.38. If now all the diodes are biased by an additional voltage of $\Delta E_x$ volts, then the curve of $z$ versus $x$ is translated in the $x$ direction by $\Delta E_x$ volts, since all the break points are moved by equal amounts and

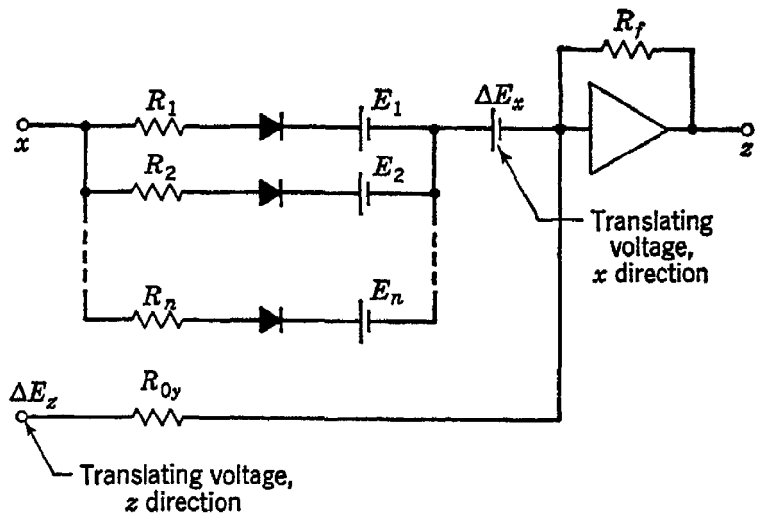



Fig. 3.38 Methods of translating the $z$ versus $x$ curve by means of bias changes

the slopes are unaffected. Likewise, the $z$ versus $x$ curve can be translated in the $z$ direction by adding a constant component $\Delta E_z$ to the input. By a combination of these actions it is possible to translate a curve by arbitrary amounts and directions and thus to generate a family of curves. If the translations are made proportional to a second independent variable $y$, then the output $z$ will be a function of both $x$ and $y$. Very few functions, however, have so simple a manner of variation with respect to $y$; therefore, a more versatile procedure is desirable. A combination of translation and "stretching" of the curve can be accomplished as shown in Fig. 3.39. It is easily seen that $z_0$, the output for $x = 0$, is given by

$$z_0 = -yR_f\left(\frac{1}{R_{0y}} + \sum \frac{P_j}{R_j}\right) \tag{3.25}$$

where $P_j$ is the fractional setting of the $j$th potentiometer measured from the $x$ end, and the summation is extended over all input branches that are conducting at $x = 0$. (The potentiometer resistances are assumed low compared with the summing resistances.) Furthermore, the abscissa $x_j$, of the $j$th break point, is determined by

$$(y - x_j)P_j + x_j = 0 \tag{3.26}$$

Therefore,

$$x_j = -\frac{P_j}{1 - P_j}y \tag{3.27}$$

Since the zero intercept of the curve and the abscissas of the break

points are both proportional to $y$ and the slopes of the segments are unaffected, it is evident that the break points move out from the origin along straight lines and that the line segments are displaced parallel to themselves and altered in length.

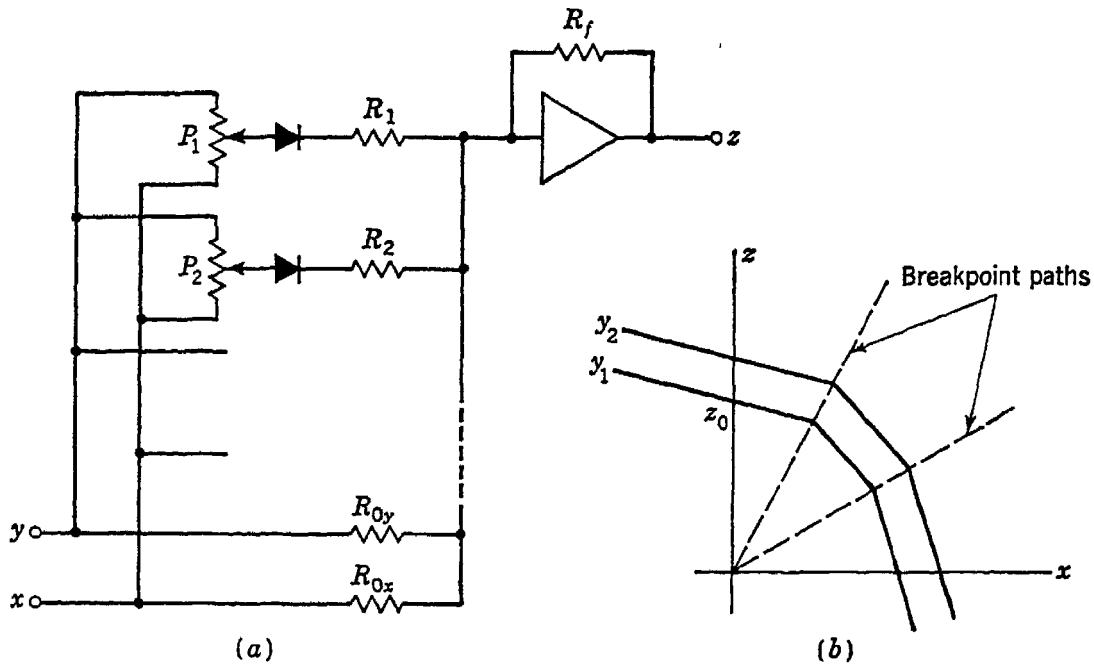



Fig. 3.39 Translation and stretching by bias change

More general modifications of the curves are also possible. For instance, the point from which the break-point paths radiate need not be the origin, but can be placed at any point by appropriate choice of the zero-input bias levels of $x$ and $y$. The break-point paths can be made curved instead of straight by supplying a bias voltage $f_1(y)$ to the potentiometers and a different voltage $f_2(y)$ to the zero-offset resistor $R_{0y}$. These voltages can in turn be provided by single-input-variable function generators. If the same voltage is applied to all the break-point potentiometers, then all the break-point positions move in proportion to changes in that voltage; but if a more flexible adjustment of the curve is desired, each break-point potentiometer can be driven from a separate function generator. This last technique is in effect an example of the fifth of the methods listed at the beginning of this section.

The reader may have observed that, throughout these adjustments of the curve, only translations and stretchings have occurred. Since the slopes are not altered, no rotation of the curves takes place. For some functions, this is not a handicap, however. A more detailed discussion of methods of curve fitting appropriate to this type of function generator may be found in the literature.[28]

**Linear interpolation between $f(x,y_j)$ and $f(x,y_{j+1})$.** A method of replacing a single function of two variables by several functions of a

single variable involves forming the family of functions $f(x,y_0)$, $f(x,y_1)$, $f(x,y_2)$, etc. The input $x$ is applied simultaneously to function generators set up to form the individual curves of the family, and the outputs are applied to some device permitting linear interpolation between the curves to give an output approximating $f(x,y)$. Any single-variable function generator can be used to form the individual curves of the family. A suitable device might be a group of diode line-segment function generators or a modified plotting table with several shaped wires and a single resistance element in contact with all of them. The interpolator between these curves might be a servo-driven tapped potentiometer, with its taps supplied from the curves of the set. The spacing of the taps, of course, would have to be made proportional to the spacings of the $y_j$ chosen for the various curves, and the usual precautions in regard to loading would have to be observed.

## 3.7 Multipliers[24]

Although the multiplier is simply a device for generating a particular function of two (or more) independent variables, it is so important a component in analog-computer work that it merits separate consideration. Any of the systems of the preceding section can be set up in such a way as to perform multiplication, but the high degree of flexibility that allows these systems to generate a wide variety of functions is not needed if only the multiplying function is to be performed, and designers have preferred to forego this flexibility in order to produce multipliers of better speed and accuracy. Even so, the design of multipliers optimum in every sense is a difficult engineering problem, as is evidenced by the wide variety of techniques that have been developed, each having its own characteristic advantages and disadvantages.

The nature of the functional relationship $z = xy$ is shown graphically in Fig. 3.40, in which contours of the $z$ surface have been sketched. These contours are hyperbolas, with $z$ positive in the first and third quadrants of the $xy$ plane and negative in the second and fourth. Any intersection of a plane $y$ = constant or $x$ = constant with the $z$ surface is a straight line. The general-purpose multiplier should be capable of reproducing this surface within the desired limits of accuracy and speed and over a wide range of $x$ and $y$. Preservation of the proper sign for $z$ is implied in this requirement, but it should be noted that not all multiplying systems do this. Failure to preserve the sign is not necessarily a disadvantage, since some applications may not require "four-quadrant" multiplication, or the multiplier may have advantages that outweigh the inconvenience of adding circuitry solely to preserve the sign.

The simplest multipliers multiply a variable by a constant and thus

produce a straight line on the $z$ surface. These can always be implemented by voltage-divider networks or by amplifiers, either alone or in combination, and represent a degenerate case which need not concern us further.

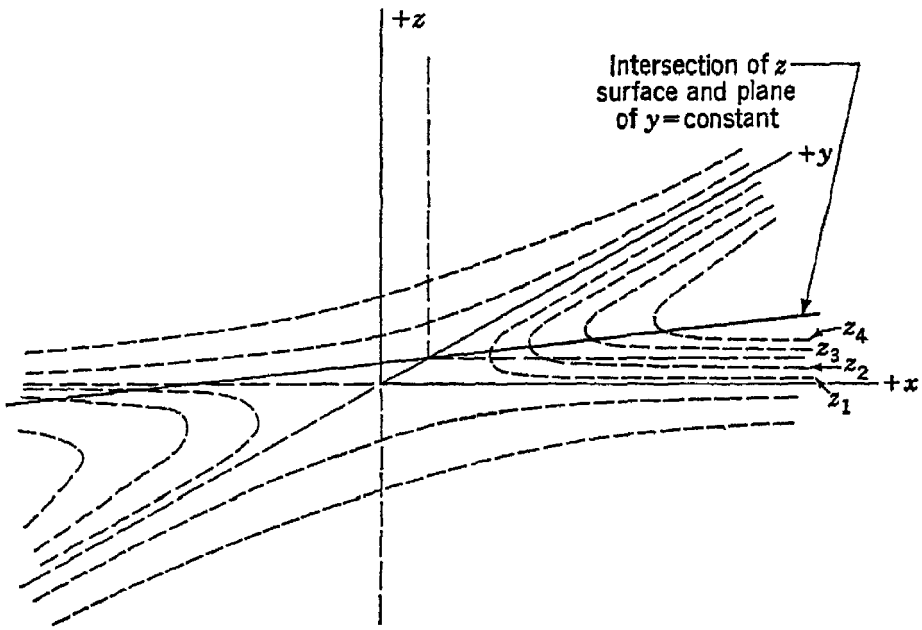



Fig. 3.40 Contours of $z = xy$

General-purpose multipliers are conveniently classified, as suggested by C. M. Edwards,[25] as representing:

1. Instrumentation of a physical law
2. Special-function devices (instrumentation of a mathematical law)
3. Variable-transmission networks
4. Modulation processes

In the subsequent pages we shall describe these classifications together with some representative examples.

**Instrumentation of a physical law.** There are in nature many laws involving product relations, such as $F = MA$ or $E = IR$. The designer of multipliers (or indeed of dividers) can occasionally use one of these laws to good advantage if he can devise a physical system constrained to act in accordance with the law and having two of the three quantities readily adjustable as inputs and the third readily measurable as an output. Although Newton's law and Ohm's law exhibit the required product relation, the physical difficulties of adjustment and measurement of the variables render them less suitable than other laws involving quantities more susceptible of manipulation.

A well known multiplier of this type is the wattmeter, which is based upon the physical law that the force on a current-carrying conductor in

a magnetic field is proportional to the current in the conductor, the magnetic field strength, and the sine of the angle between them. The input quantities in this analog are the magnetic field and the current in the moving coil. By proper engineering design of the electromagnet, the field can be made directly proportional over a wide range to the line current in the system under measurement. The current in the moving coil

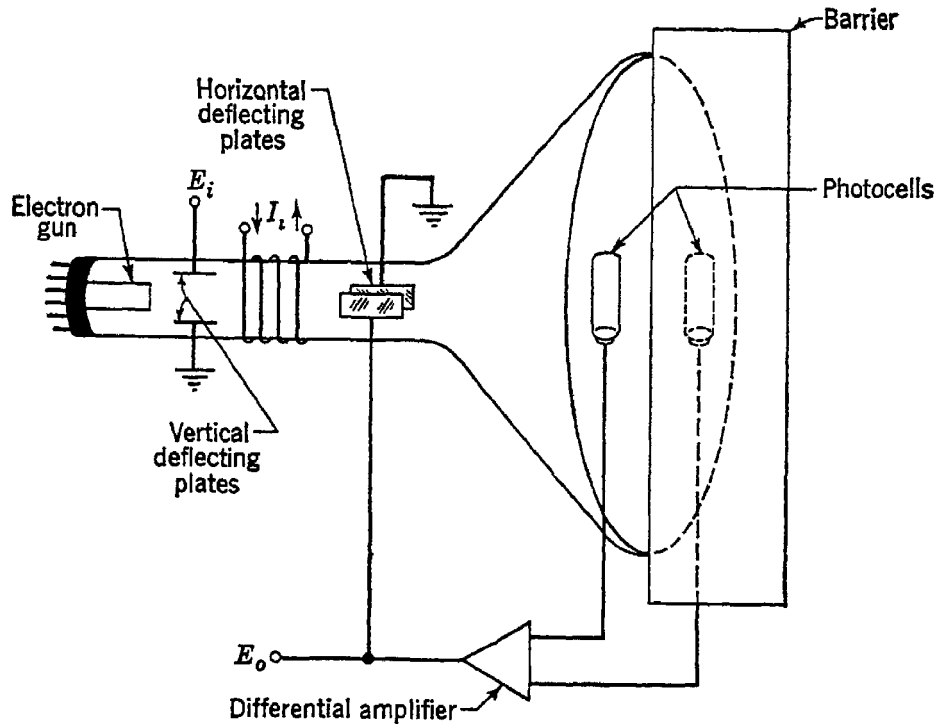



Fig. 3.41 The crossed-fields electron-beam multiplier

is made proportional to the line voltage, and the angle between field and current is held constant. The output force is measured by its effect in extending a spring, the extension being directly proportional to the force.

It often happens that a given physical law can be instrumented in several different ways, and the designer who has found one such method should not feel that he has exhausted the possibilities. The force law upon which the wattmeter is based is a case in point, and at least one device exists that obeys the same law but is very differently constructed. This is the crossed-fields electron-beam multiplier.[26] The elements of this system are shown in Fig. 3.41. One input is impressed as a voltage $E_i$ on the vertical deflecting plates and the other as a current $I_i$ on the coil wound around the barrel of the tube. The beam electrons acquire a vertical velocity proportional to $E_i$ in passing through the vertical deflecting plates. The axial magnetic field then imparts a horizontal deflection proportional to the vertical velocity and to the magnetic field, that is, proportional to $E_i$ and $I_i$. Two photocells on opposite sides of a barrier at the face of the tube detect the sense of the lateral deflection

of the beam and drive a differential amplifier connected to the horizontal plates so as to return the beam to the center line of the screen. Thus $E_o$, the voltage required to do this, is directly proportional to $E_i$ and $I_i$, that is, to their product.

Although these two devices are based upon the same law, the crossed-fields multiplier is by far the more useful for computer applications because it converts the force that is the measure of the product into electrical form rather than into a mechanical displacement. The crossed-fields multiplier also exhibits the useful property of being usable as a divider. If the output of the differential amplifier is applied to the vertical deflecting plates and the inputs are applied to the coil and to the horizontal deflecting plates, then the output voltage of the amplifier is proportional to the horizontal-deflection voltage divided by the input current to the coil.

**Special-function devices.** Just as certain laws of physics provide a basis for devices to perform multiplication, so certain mathematical

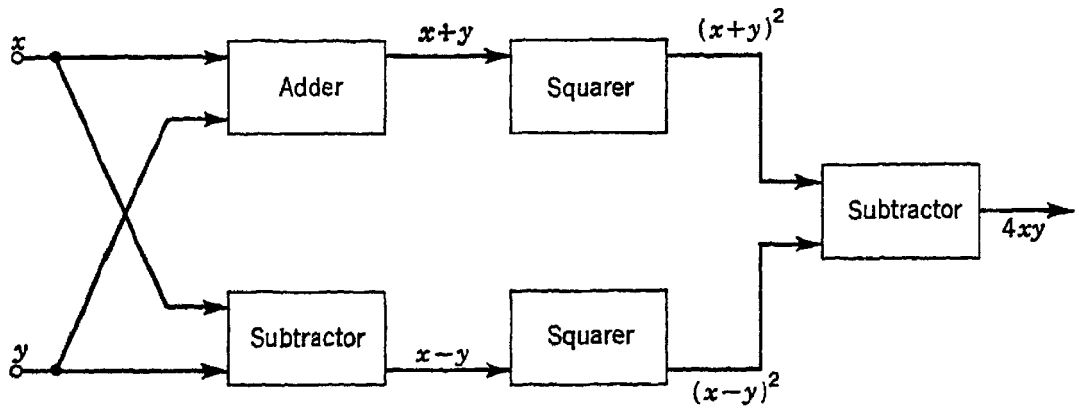



Fig. 3.42 The "quarter-squares" multiplier

principles suggest ways of multiplying. Many of these principles have been employed in one device or another, and many more are waiting to be studied and used. Application of a mathematical principle to the design of a multiplier is a stimulating problem, for mathematical techniques, unlike physical laws, are not restricted in the physical nature of the devices used to implement them.

An example is the so-called "quarter-squares" principle, which allows formation of a product by the relation

$$xy = \tfrac{1}{4}[(x + y)^2 - (x - y)^2] \tag{3.28}$$

A block diagram of a typical system implementing this relation is shown in Fig. 3.42. The principle has been used in many systems taking various physical forms, most of which differ basically in the devices they use to generate $(x + y)^2$ and $(x - y)^2$. Among these devices are the following:

1. Conventional vacuum tubes operated under conditions in which the plate current is a quadratic function of the grid voltage or of the plate voltage
2. Function generators such as those of Secs. 3.4 and 3.5
3. Specially constructed vacuum tubes having a square-law characteristic
4. Use of nonlinear conductive materials, such as thyrite, that display a square-law characteristic[27]
5. Devices based on the square-law relation of the areas of similar geometric figures

Over a limited range of their characteristics some vacuum tubes exhibit a square-law relation between plate current and grid voltage or between plate current and plate voltage,[28] and multipliers of moderate accuracy

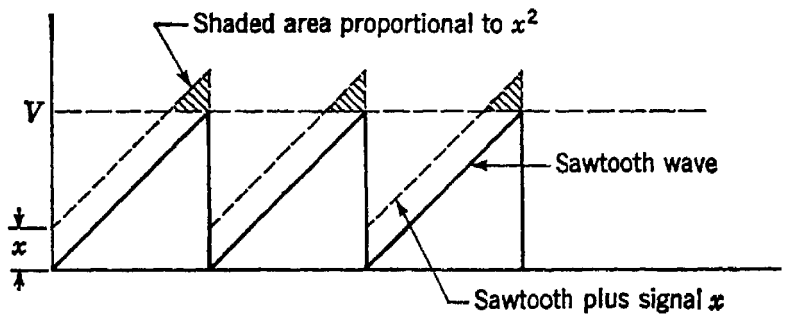



Fig. 3.43 Principle of the similar-triangle squarer

have often been built around such tubes. Careful adjustment of the bias conditions may be necessary to eliminate terms other than the quadratic. Multigrid tubes have also been used as squarers, with the signal applied to two grids that affect the plate current in the same manner.

Although the use of a general-purpose function generator in a multiplier is not ordinarily justifiable, it is sometimes possible to make use of the function-generator principle in a less versatile device intended only as a squarer. The biased-diode line-segment function generator is one simple device that has been used in this way. One system that has been described[29] uses a single squarer time-shared at 50 kc between the two quantities to be squared, thus obviating the usual necessity for carefully matching the characteristics of two squarers.

Specially constructed vacuum tubes have also been used to perform the squaring operation. In one such device a flat sheet of electrons is deflected across a target having parabolic apertures. The output current is then proportional to the square of the input signal.[30,31]

Squarers have also been built using the principle that the areas of similar triangles are proportional to the squares of their corresponding sides.[32] Consider the waves shown in Fig. 3.43. If a saw-tooth wave

of peak value $V$ and a positive voltage $x$ are added and if the resulting wave is clipped so that only those parts above $V$ are retained, then the average value of the remaining part (shaded in the figure) is proportional to $x^2$.

If the squaring device can accept inputs of either sign, then the quarter-squares multiplier gives four-quadrant operation. With squarers that can accept only positive inputs, the multiplier is limited to the portion of the $xy$ plane in which both $x + y \geq 0$ and $x - y \geq 0$, that is, to a 90° sector centered on the positive $x$ axis. In this case additional circuitry is needed to furnish four-quadrant operation. For instance, the

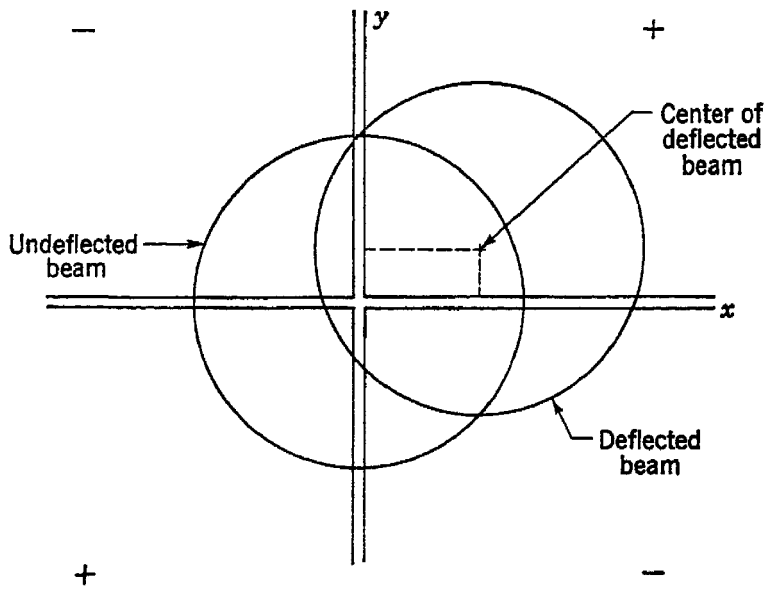



Fig. 3.44 Principle of the Angelo multiplier tube

system might be arranged to handle $|x + y|$ and $|x - y|$ instead of the signed quantities.

Another mathematical technique that has attractive features for multiplication is the use of logarithms. A logarithmic multiplier makes use of the simple process of addition and is not restricted to multiplying only two quantities. The practical difficulty, of course, lies in the problem of forming quantities proportional to logarithms and antilogarithms. Here, too, it is possible to use conventional function generators or to make use of devices having approximately logarithmic characteristics.[33,34] As was pointed out in Chap. 2, it is not necessary to form both functions explicitly, since, if one is available, its inverse can be formed by placing it in a feedback loop around a high-gain computing amplifier.

Geometrical principles have been used for multiplication in a tube devised by Angelo.[35] A circular beam of electrons is directed toward a target that is divided into four electrically isolated quadrants, as shown in Fig. 3.44. When no deflecting voltages are applied, the beam falls

equally on the four quadrants. The output current, which is taken as the sum of the first and third quadrant currents less the sum of the second and fourth quadrant currents, is zero for this condition, but, if the beam is given $x$ and $y$ deflections, it is a simple exercise in geometry to show that the net current is proportional to the product of the deflections.

**Variable-transmission networks.** In the variable-transmission type of multiplier one variable controls the transmission of another through a network. If the output is linearly related to the input so that $e_o = k_1 e_i$ and if $k_1$ is directly proportional to a second variable $e_c$, then $e_o = k_2 e_c e_i$. The most familiar example of this type of multiplier is the servo-driven potentiometer, in which the shaft is positioned in proportion to one variable and the other variable is applied across the potentiometer, so that the output voltage is proportional to the product of the two. The variable-transmission multiplier often uses two or more identical networks, one in a feedback path to aid in positioning the control and the others as multipliers. It is possible, of course, to switch a single network back and forth between the feedback circuit and the multiplying circuit and thereby eliminate the need for careful matching of two networks, but this switching must be done at a frequency much higher than the signal frequencies and introduces more problems than it solves.

**Modulation processes.** Although in a broad sense all forms of multiplication are basically modulation processes, there are several types of multiplier that have so much in common with the modulation techniques of radio communication that they deserve to be treated separately. Amplitude modulation, frequency modulation, and various types of pulse modulation can be employed, or combinations of these. A wide variety of systems employing these techniques have been described in the literature.[36–41] The details of their circuits will not concern us here, but the more significant features of the systems will be pointed out.

Systems involving amplitude modulation or frequency modulation typically employ double modulation, the carrier being modulated by one of the variables and the modulated carrier then being modulated by the second. Both modulations may be of the same type, either AM or FM, or one may be AM and the other FM. After detection by a system suitable to the forms of modulation used, an output is obtained proportional to the product of the two modulating signals. Block diagrams of an AM-AM system and an FM-AM system are shown in Fig. 3.45. It is of course obvious that, although the phase discriminator used in the FM-AM system can be the familiar Foster-Seeley circuit used in FM receivers, the amplitude-limiting process used in FM receivers has no place here. A block diagram of an FM-FM system might also have been included, but the practical difficulties involved make this unlikely to be useful.

The high state of development reached by pulse techniques in recent years opens up a wide variety of possibilities for their use in multipliers. One example is what might be called a "sampling multiplier".[42] In its simplest form, a reference sweep $e_1 = k_1t$ is continuously compared with $k_2x$, a voltage proportional to one of the factors. At the instant they become equal, a linear sweep $e_3 = k_3yt$, rising at a rate proportional to the other variable, is sampled. At the sampling instant $e_3 = (k_2k_3/k_1)xy$.

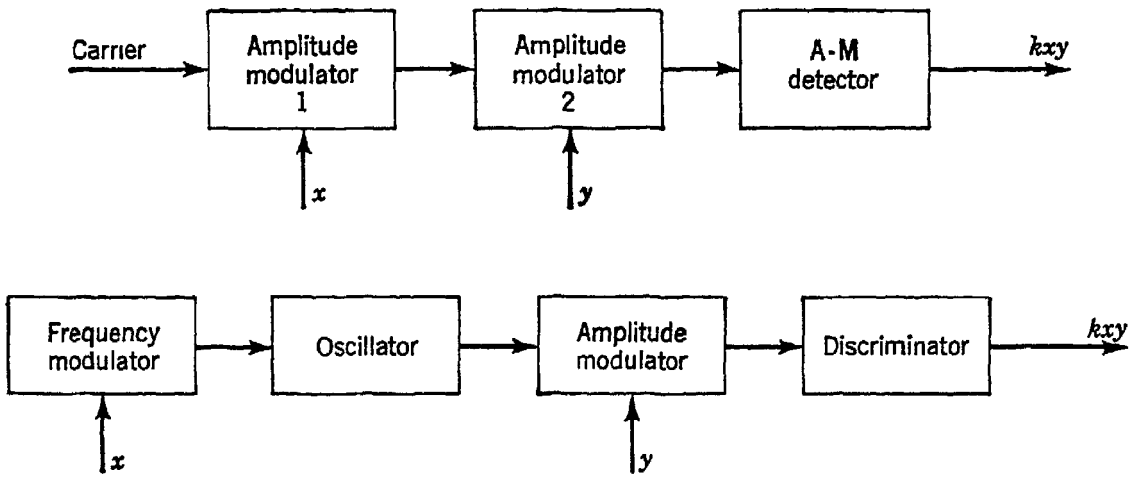



Fig. 3.45 Multipliers using (*a*) an AM-AM system, and (*b*) an AM-FM system

One pulse-modulation multiplier extremely simple in concept produces rectangular pulses with amplitude proportional to one variable and duration proportional to the other. The average value of the pulse stream is, then, proportional to the product of the two variables. Although this idea had been in circulation for a number of years, the practical implementation of it in devices of high accuracy did not occur until the invention of the precision switching method of Goldberg.[43] The system has come to be known as the "time-division" multiplier, and is one of the most accurate of the all-electronic high-speed multipliers. Methods of implementing the system differ in detail but have the following elements in common:

1. An integrator whose input is switched alternately from a positive value to a negative value of different magnitude
2. A bistable element ("flip-flop") that switches state when the integrator output reaches a certain magnitude
3. An electronic switch, controlled by the flip-flop, to drive the integrator
4. An electronic switch, controlled by the flip-flop, to drive the output of the multiplier

A block diagram of the interconnection of these components is shown in Fig. 3.46.

Switch 1 produces a positive output while the flip-flop is in one state and an equal negative output while the flip-flop is in the other state. The magnitudes of these outputs must be larger than the maximum magnitude of $x$ to be encountered. While the switch output and $x$ are of the same sign, the integrator output changes rapidly; while they are of opposite sign, it changes slowly. Thus the voltage excursion required to trip the flip-flop is more rapid in one direction than in the other, and the time durations of the two states of the flip-flop are unequal. (The

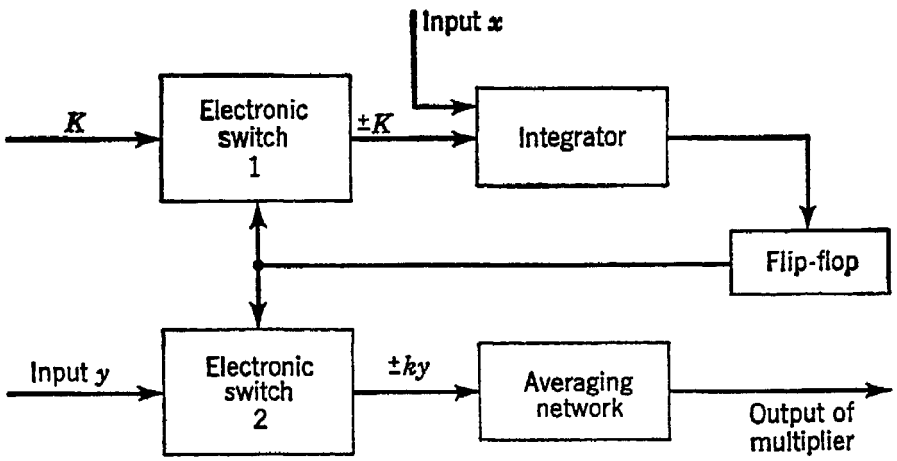



Fig. 3.46 Time-division multiplier

flip-flop is assumed balanced with respect to ground, so that successive triggering voltage levels are equal in magnitude and opposite in sign.) Consequently, switch 2, which produces a positive output proportional to $y$ while the flip-flop is in one state and an equal negative output while the flip-flop is in the other state, produces an unsymmetrical wave whose average value is proportional to $xy$. This can be demonstrated as follows:

Let $K$ = magnitude of output of switch 1
$E$ = total voltage excursion needed to trip the flip-flop
$T_1$ = time during which integrator input is $x + K$
$T_2$ = time during which integrator input is $x - K$

Then
$$T_1 = \frac{E}{x + K} \quad \text{and} \quad T_2 = \frac{-E}{x - K} = \frac{E}{K - x} \tag{3.29}$$

$$T_1 + T_2 = \frac{E}{K + x} + \frac{E}{K - x} \tag{3.30}$$

$$= \frac{2EK}{K^2 - x^2} \tag{3.31}$$

Let $ky$ = magnitude of output of switch 2, which is of one sign during $T_1$ and of opposite sign during $T_2$. Then the average of the output of

switch 2 is

$$\text{Average output} = \frac{kyT_1 - kyT_2}{T_1 + T_2} \tag{3.32}$$

$$= ky\,\frac{E/(K+x) - E/(K-x)}{2EK/(K^2 - x^2)} \tag{3.33}$$

$$= -ky\,\frac{2x}{2K} = -\frac{k}{K}\,xy \tag{3.34}$$

If $x$ and $y$ are constant, the waveforms of Fig. 3.47 are obtained. In this sketch $x$ was taken to be a negative quantity and $y$ and $K$ positive.

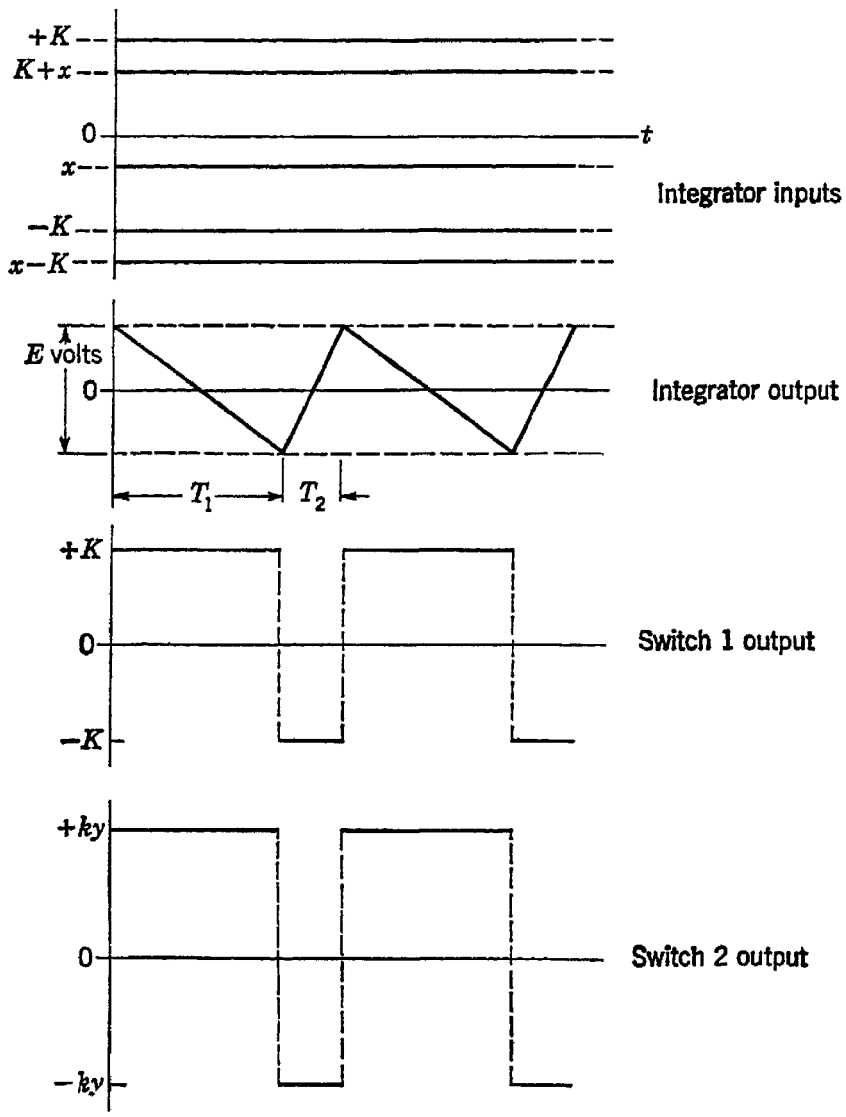



Fig. 3.47 Waveforms in the time-division multiplier

Thus $K + x$ is a small positive quantity, and while the integrator input has this value its output voltage decreases slowly to the negative trigger level of the flip-flop. When the switch output is changed in sign, the integrator input is a large negative quantity, and its output rises rapidly during the interval $T_2$. During $T_1$ the output of switch 2 is $ky$, and

during $T_2$ it is $-ky$. Thus the output of the multiplier is proportional to the product of $x$ and $y$, the sign being determined by the sign of $K$, the input to switch 1. The system evidently provides four-quadrant multiplication.

The frequency of the wave is

$$f = \frac{1}{T_1 + T_2} = \frac{K^2 - x^2}{2EK} \tag{3.35}$$

For $x = 0$, $T_1 = T_2$ and $f$ is maximum. The lowest frequency of operation, corresponding to full-scale value for $x$, should be high compared with the highest frequency of $x$ or $y$. Some form of low-pass filter

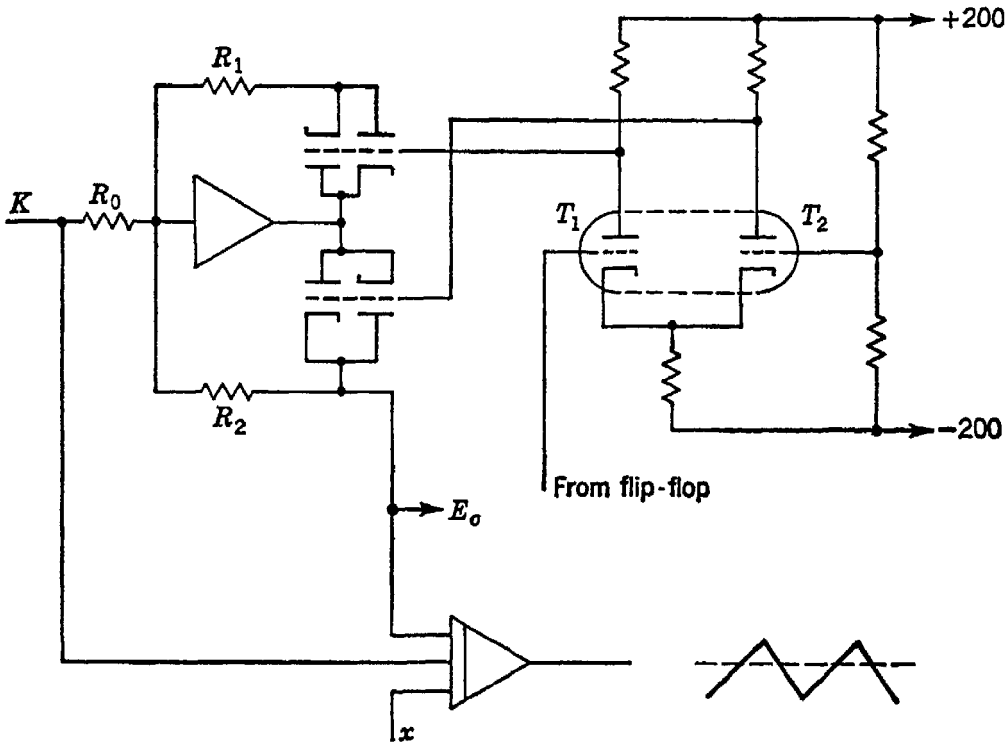



Fig. 3.48 An electronic switch due to Morrill and Baum

or averaging network is necessary at the output of switch 2 to eliminate the switching-frequency component and its harmonics in the multiplier output.

The basic switch circuit used by Morrill and Baum[44] in their multiplier is shown in Fig. 3.48. The flip-flop drives the double-triode amplifier $T_1T_2$, which in turn drives the triodes in series with $R_1$ and $R_2$ so that one or the other of these resistors is connected as the feedback path for the computing amplifier. While $R_1$ conducts, $E_o = 0$ and the integrator input is $x + K$. While $R_2$ conducts, $E_o = -2K$, since $R_2/R_o = 2$ and the total integrator input is thus $x - K$.

Through the use of high-precision switches[45,46] the time-division multi-

plier can be made to operate with very high accuracy and speed, although the equipment cost may not be justifiable where such high performance is not required. Goldberg[43] has reported accuracies of 0.01 per cent with a maximum multiplier switching frequency of 2,000 cycles and 0.1 per cent with a maximum frequency of 20,000 cycles. More recent improvements in electronic switches have made it possible to obtain these accuracies at higher frequencies.

### 3.8 Simulation of time delay[47–54]

The occasion sometimes arises in analog-computer work for producing an exact replica of a signal at a time $\tau$ sec after the original. A common situation of this sort is the simulation of systems involving "transport lag", in which a change in a quantity being transported (e.g., liquids in a chemical-processing pipeline) is not detected until a later point in the system. There are several ways in which such a problem can be treated, but none is suitable for all ranges of delay. An obvious method is to plot the function on a plotting table and retrace it with a curve follower after the proper time interval. Manual curve following is suitable only for slowly varying functions, and an automatic device would be desirable for following rapidly varying functions. If the original curve can be drawn in conducting ink, then the curve follower can be a high-frequency detector such as that described in Sec. 3.5 in the article on mechanically driven curve followers. A photoelectric follower is another possibility, although the design problems involved are difficult.

Another method is to record the function on magnetic tape or on a magnetic drum and to read it back after the proper time interval. This technique requires that the signal be modulated as it is recorded, since a constant level of magnetization would be read back as zero, in the absence of flux changes. "Chopping" of the signal (i.e., amplitude modulation of a square wave) is one way of doing this, if the chopping frequency is well above the signal frequencies and yet low enough to be faithfully recorded. An amplitude-modulation system requires careful control of the spacing between the magnetic surface and the recording and playback heads. Frequency modulation might also be employed. The requirement in this case is for precise control of the velocity of the magnetic medium. A very useful characteristic of systems that record and play back is the possibility of altering the time lag by varying the spacing of the recording and reading elements.

A third technique consists in delaying the electrical signal by means of electrical circuits, instead of transforming it into a visual or magnetic recording and then retransforming it after the proper interval. This method is best suited to delays of seconds or less, since the amount of

equipment required for longer delays is prohibitive unless a very small bandwidth is acceptable. It is well known that the curve of phase shift versus frequency must be linear if all frequency components of a signal are to be delayed by the same time intervals. Such a relation cannot be achieved in practice, since it corresponds to an all-pass filter, a device that cannot exist because of the ultimate effect of stray shunting capacitances at high frequency. However, it is possible to approximate such a phase characteristic over a limited band of frequencies. This is a typical approximation problem of network synthesis, and circuit theorists have developed powerful techniques for handling it. Treatment of these techniques is beyond the scope of this work, but the interested reader will find a large literature available.

In brief, the network-synthesis technique consists in devising transfer characteristics that have the desired slope for their phase shift versus frequency curve at $f = 0$ and for which the first several derivatives of this slope are zero. If the desired curve of phase shift versus frequency is $\phi = \omega T$, then the slope $T$ is the time delay of the system. Generally speaking, the higher this slope, the lower will be the frequency at which an approximating circuit of given form departs from the ideal by a given amount, as shown in Fig. 3.49. The curve approximating $\phi_1$ differs from it by $\Delta$ degrees at $f_1$, and the same deviation between $\phi_2$ and its approximant occurs at a higher frequency $f_2$. In order to achieve a deviation of $\Delta$ degrees from curve $\phi_1$ at a higher frequency than $f_1$, a higher-order approximant is needed, and a more complicated circuit must be used.

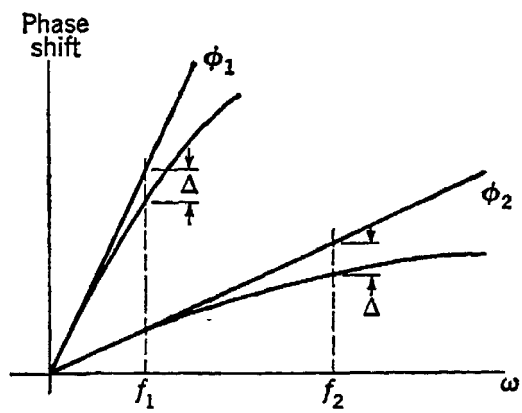


Fig. 3.49 Two phase-shift curves and their approximants

The transfer function $H$ of a delay network is conveniently expressed as a ratio of polynomials in $s$, the complex frequency variable. Several of these follow, as given by Cunningham:[47]

First-order approximant: $$H = -\frac{s - a}{s + a} \tag{3.36}$$

Second-order:

$$H = -\frac{s^2 - 2as + (a^2 + b^2)}{s^2 + 2as + (a^2 + b^2)} \tag{3.37}$$

Third-order:

$$H = \frac{[s^2 - 2as + (a^2 + b^2)](s - c)}{[s^2 + 2as + (a^2 + b^2)](s + c)} \tag{3.38}$$

Fourth-order:

$$H = \frac{[s^2 - 2as + (a^2 + b^2)][s^2 - 2cs + (c^2 + d^2)]}{[s^2 + 2as + (a^2 + b^2)][s^2 + 2cs + (c^2 + d^2)]} \tag{3.39}$$

These expressions are for networks having poles of $H$ at $-a \pm jb$ and $-c \pm jd$ and zeros at $+a \pm jb$ and $+c \pm jd$. The networks for the first two approximants are shown in Fig. 3.50.

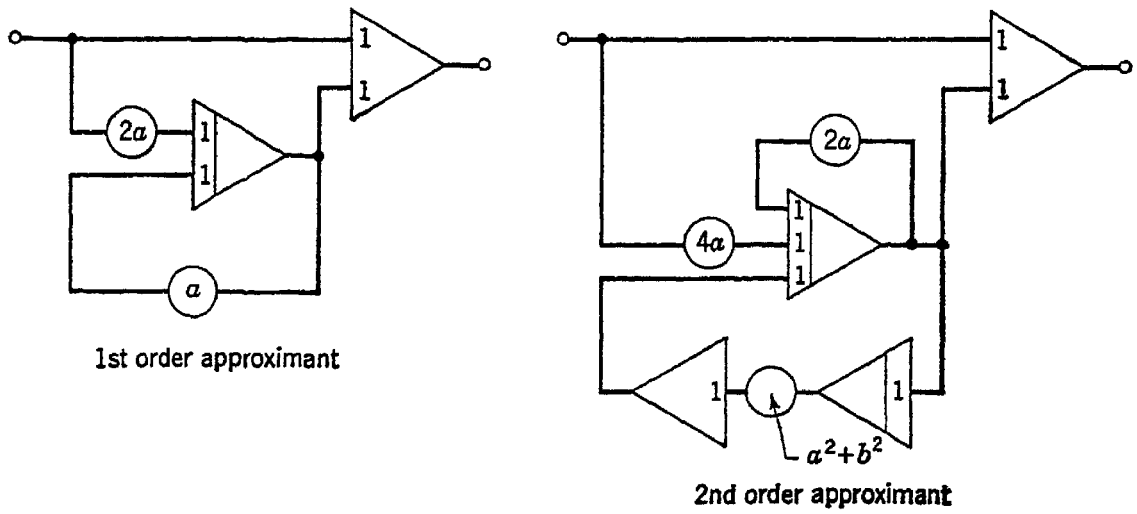



Fig. 3.50 Time-delay networks

In this connection it is of interest to note that a continued-fraction expansion of a ratio of polynomials is realizable as a network having successive feedback loops within feedback loops. The transfer function of the second-order approximant may be expressed in continued-fraction form as follows:

$$H = -1 + \cfrac{1}{\cfrac{s}{4a} + \cfrac{1}{2 - \cfrac{1}{\cfrac{a}{a^2 + b^2}s + \cfrac{1}{2}}}} \tag{3.40}$$

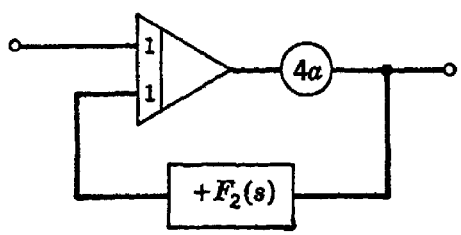



Fig. 3.51 Network for $-F_1(s)$

The fractional part of this may be written

$$F_1(s) = \frac{1}{s/4a + F_2(s)} \tag{3.41}$$

The transfer characteristic $-F_1(s)$ is realizable by the network of Fig. 3.51. $F_2(s)$ in turn is given by the expression

$$F_2(s) = \frac{1}{2 + F_3(s)} \tag{3.42}$$

where $$F_3(s) = -\frac{1}{[a/(a^2 + b^2)]s + \frac{1}{2}} = -\frac{(a^2 + b^2)/a}{s + (a^2 + b^2)/2a} \tag{3.43}$$

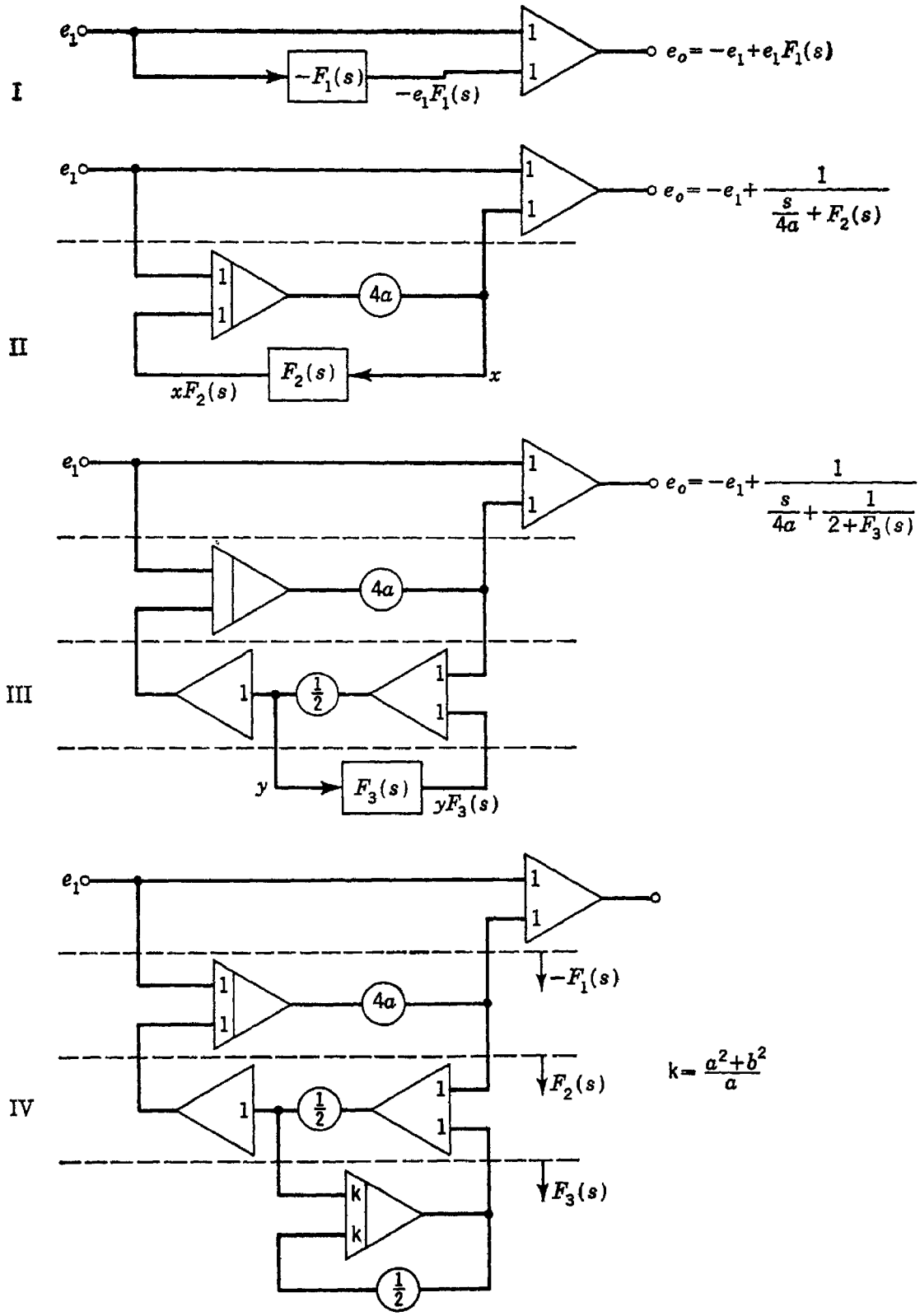



Fig. 3.52 Successive steps in a continued-fraction transfer-function realization

Construction of the complete network by successive inclusion of these internal loops is shown in Fig. 3.52. It will be noted that five amplifiers are used, while the network of Fig. 3.50 used only four. Thus, although this method is systematic, it is not necessarily optimum.

## REFERENCES

*General:*

Achieser, N. I., "Theory of Approximation," Frederick Ungar Publishing Co., New York, 1956.

Hildebrand, F. B., "Introduction to Numerical Analysis," McGraw-Hill Book Company, Inc., New York, 1956.

Soroka, Walter W., "Analog Methods in Computation and Simulation," McGraw-Hill Book Company, Inc., New York, 1954, chaps. 6, 8, and 9.

Johnson, Clarence L., "Analog Computer Techniques," McGraw-Hill Book Company, Inc., New York, 1956, chaps. 7 and 8.

Korn, Granino A., and T. M. Korn, "Electronic Analog Computers," 2d ed., McGraw-Hill Book Company, Inc., New York, 1956, chap. 6.

*Items Referred to by Number:*

1. Moshos, G. J., Analog interpolator for automatic control, *J. ACM*, vol. 2, no. 2, April, 1955, pp. 83–91.
2. Burt, E. G. C., and O. H. Lange, Function generators based on linear interpolation with applications to analogue computing, *Monograph* 137M, *Proc. IEE London*, vol. 103, pt. C, March, 1956, pp. 51–58.
3. Role of diodes in an electronic differential analyzer, *Goodyear Aircraft Corp. Rept.* GER-4798, Apr. 14, 1952.
4. Morrill, C. D., and R. V. Baum, Diode limiters simulate mechanical phenomena, *Electronics*, vol. 25, no. 11, November, 1952, pp. 122–126.
5. Updike, O. L., Jr., and J. H. McLeod, Jr., A comparison of diode switching methods for analog function generation, *Proc. 1956 Natl. Simulation Conf.*, pp. 35.1–35.6.
6. Bruns, R. A., An improved diode function generator for analog computers, *Proc. 1956 Natl. Simulation Conf.*, pp. 32.1–32.7.
7. Howe, R. M., Representation of nonlinear functions by means of operational amplifiers, *IRE Trans. on Electronic Computers*, vol. EC-5, no. 4, December, 1956, pp. 203–206.
8. Bolie, V. W., Synthesis of function generators by continued fraction theory, *Tele-Tech & Electronic Inds.*, vol. 15, no. 3, March, 1956, p. 88.
9. Levenstein, H., Generating nonlinear functions with linear potentiometers, *Tele-Tech & Electronic Inds.*, vol. 12, no. 10, October, 1953, p. 76.
10. Greenwood, Ivan A., Jr., J. Vance Holdam, Jr., and Duncan MacRae, Jr. (eds.), "Electronic Instruments," McGraw-Hill Book Company, Inc., New York, 1948.
11. Seay, P. A., New circuit computes tangents, *Control Eng.*, vol. 1, no. 3, November, 1954, pp. 59–60.
12. Korn, G. A., Design and construction of universal function generating potentiometers, *Rev. Sci. Instr.*, vol. 21, no. 1, January, 1950, pp. 77–81.
13. Dole, F. E., chap. 8 in Blackburn, John F. (ed.), "Components Handbook," McGraw-Hill Book Company, Inc., New York, 1949, pp. 243–287.
14. Duncan, D. C., Characteristics of precision servo computer potentiometers, *Tele-Tech*, vol. 11, no. 11, pp. 52–54+, November, 1952.
15. Blackett, P. M. S., and F. C. Williams, An automatic curve follower for use with the differential analyzer, *Proc. Cambridge Phil. Soc.*, vol. 35, 1939, pp. 494–505.

16. Williams, F. C., A reversible head for the automatic curve following device, *Proc. Cambridge Phil. Soc.*, vol. 35, 1939, pp. 506–511.
17. Sunstein, D. E., Photoelectric waveform generator, *Electronics*, vol. 22, no. 2, February, 1949, pp. 100–103.
18. Mynall, D. J., Electrical analogue computing III, *Electronic Eng.*, vol. 19, no. 8, August, 1947, pp. 259–262.
19. MacKay, D. M., A high-speed electronic function generator, *Nature*, vol. 159, no. 4038, pp. 406–407.
20. Mynall, D. J., Electronic function generator, *Nature*, vol. 159, no. 4048, p. 743.
21. Uniformly conductive surfaces, *Univ. Mich. Eng. Research Inst. Rept. on Project* M992-4, September, 1953.
22. Larrowe, V., and M. Spencer, Use of semi-conducting surfaces in analog function generation, *Proc. 1956 Natl. Simulation Conf.*, pp. 33.1–33.6.
23. Meissinger, H. F., An electronic circuit for the generation of functions of several variables, *1955 IRE Conv. Record*, pt. 4, pp. 150–161.
24. Czajkowski, Z., Electronic methods of analogue multiplication, *Electronic Eng.*, vol. 28, July, 1956, pp. 283–287; August, 1956, pp. 352–355.
25. Edwards, C. M., Survey of analog multiplication schemes, *J. ACM*, vol. 1, January, 1954, pp. 27–35.
26. Macnee, A. B., An electronic differential analyzer, *Proc. IRE*, vol. 37, pp. 1315–1324, 1949.
27. Kovach, L. D., and W. Comley, An analog multiplier using thyrite, *IRE Trans. on Electronic Computers*, vol. EC-3, no. 2, June, 1954, pp. 42–45.
28. Chance, Britton, F. C. Williams, V. W. Hughes, D. Sayre, and E. F. MacNichol, Jr. (eds.), "Waveforms," McGraw-Hill Book Company, Inc., New York, 1949, pp. 678–693.
29. Chance, B., F. C. Williams, C. Yang, J. Busser, and J. Higgins, A quarter-square multiplier using a segmented parabolic characteristic, *Rev. Sci. Instr.*, vol. 22, no. 9, September, 1951, pp. 683–688.
30. Miller, J. A., A. S. Soltes, and R. E. Scott, Wide-band analog function multiplier, *Electronics*, vol. 28, no. 2, February, 1955, pp. 160–163.
31. Soltes, A. S., A wide-band square-law computing amplifier, *IRE Trans. on Electronic Computers*, vol. EC-3, no. 2, June, 1954, pp. 37–41.
32. Norsworthy, K. H., A simple electronic multiplier, *Electronic Eng.*, vol. 26, February, 1954, pp. 72–75.
33. Savant, C. J., Jr., and R. C. Howard, Multiplier for analog computers, *Electronics*, vol. 27, no. 9, September, 1954, pp. 144–147.
34. Savant, C. J., and R. C. Howard, A function generator for the solution of engineering design problems, *IRE Trans. on Electronic Computers*, vol. EC-3, no. 4, September, 1954, pp. 34–38.
35. Angelo, E. J., Jr., An electron-beam tube for analog multiplication, *Rev. Sci. Instr.*, vol. 25, no. 3, March, 1954, pp. 280–284.
36. Chance, Britton, F. C. Williams, V. W. Hughes, D. Sayre, and E. F. MacNichol, Jr. (eds.), "Waveforms," McGraw-Hill Book Company, Inc., New York, 1949, pp. 674–676.
37. Garrett, D. E., and F. G. Cole, A general purpose electronic wattmeter, *Proc. IRE*, vol. 40, no. 2, February, 1952, pp. 165–171.
38. Sydnor, R. L., T. R. O'Meara, and J. Strathman, Analog multipliers and squarers using a multigrid modulator, *IRE Trans. on Electronic Computers*, vol. EC-5, no. 2, June, 1956, pp. 82–85.
39. McCool, W. A., An AM-FM electronic analog multiplier, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1470–1476.

40. Weibel, E. S., An electronic analog multiplier using carriers, *IRE Trans. on Electronic Computers*, vol. EC-6, no. 1, March, 1957, pp. 30–33.
41. Meyer, M. A., and H. W. Fuller, Two new electronic analog multipliers, *Rev. Sci. Instr.*, vol. 25, no. 12, December, 1954, pp. 1166–1172.
42. Freeman, H., and E. Parsons, A time-sharing analog multiplier, *IRE Trans. on Electronic Computers*, vol. EC-3, no. 1, March, 1954, pp. 11–17.
43. Goldberg, E. A., A high-accuracy time-division multiplier, *RCA Rev.*, vol. 13, no. 3, September, 1952, pp. 265–274.
44. Morrill, C. D., and R. V. Baum, A stabilized electronic multiplier, *Proc. Natl. Electronics Conf.*, vol. 8, 1952, pp. 710–715.
45. Blanyer, C. G., Precision modulators and demodulators, *J. ACM*, vol. 2, no. 4, October, 1955, pp. 229–242.
46. Edwards, C. M., Precision electronic switching with feedback amplifiers, *Proc. IRE*, vol. 44, no. 11, November, 1956, pp. 1613–1620.
47. Cunningham, W. J., Time-delay networks for an analog computer, *IRE Trans. on Electronic Computers*, vol. EC-3, no. 4, December, 1954, pp. 16–18.
48. Morrill, C. D., A sub-audio time delay circuit, *IRE Trans. on Electronic Computers*, vol. EC-3, no. 2, June, 1954, pp. 45–49.
49. Thomson, W. E., Time-delay circuits, *IRE Trans. on Electronic Computers*, vol. EC-4 no. 1, June, 1955, p. 74.
50. Thomson, W. E., Delay networks having maximally flat frequency characteristics, *Proc. IEE London*, vol. 96, pt. III, November, 1949, pp. 487–490.
51. Single, C. H., An analog for process lags, *Control Eng.*, vol. 3, no. 10, October, 1956, pp. 113–114.
52. Teasdale, R. D., Time-domain approximation by use of Pade approximants, *1953 IRE Conv. Record*, pt. 5, pp. 89–94.
53. Janssen, J. M. L., Discontinuous low-frequency delay line with continuously variable delay, *Nature*, vol. 169, Jan. 26, 1952, p. 148.
54. Irvine, N. L., Simulation of dead time, *Proc. 1956 Natl. Simulation Conf.*, pp. 25.1–25.3.

## PROBLEMS

**3.1** Show a computer setup to solve the equation

$$3\frac{d^2x}{dt^2} + 47\left(\frac{dx}{dt}\right)^2 + 2 \times 10^4 x = f\left(\frac{dx}{dt}\right)$$

where

$$f\left(\frac{dx}{dt}\right) \begin{cases} 1 \text{ for } \dfrac{dx}{dt} > 0 \\ = 0 \text{ for } \dfrac{dx}{dt} = 0 \\ -1 \text{ for } \dfrac{dx}{dt} < 0 \end{cases}$$

At $t = 0$, $x = 10$ and $dx/dt = 0$.

**3.2** A continued-fraction expansion of $\log(1 + x)$ is

$$\log(1 + x) = \cfrac{x}{1 + \cfrac{1^2 \cdot x}{2 + \cfrac{1^2 \cdot x}{3 + \cfrac{2^2 \cdot x}{4 + \cfrac{2^2 \cdot x}{5 + \cfrac{3^2 \cdot x}{6 \cdots}}}}}}$$

Develop an iterative network by means of which this function can be approximated.

**3.3** Repeat Prob. 3.2 for the function

$$(x + c^2)^{1/2} = c + \cfrac{x}{2c + \cfrac{x}{2c + \cfrac{x}{2c + \cdots}}}$$

**3.4** The function $y = 50 \sin x$, $0 \leq x \leq \pi$ is to be fitted to within 3 units by straight-line segments. What is the smallest number of segments that can be used? Specify the positions of such a set. (Use a graphic solution, plotting upper and lower error bounds.)

**3.5** Using Newton's formula, form a table of divided differences for $y = \sin x$, evaluated at 10° intervals from 0° to 90° with three-figure accuracy. From this table derive a setup for a servo-multiplier type of polynomial generator modeled after Fig. 3.5.

**3.6** The limiter characteristics shown in Fig. 3.7 are based upon the assumption that the diodes are perfect. If instead the diodes are all characterized by a small forward resistance $R_f$ and a large but finite back resistance $R_b$, show what the effect would be upon each of the characteristics of Fig. 3.7.

**3.7** For the circuit of Fig. 3.13*b*, derive an expression for the value of $e_1$ at which the break point in $e_o$ occurs. What is the slope $de_o/de_1$ in the region of diode conduction? If $R_1 = R_2 = 1$ megohm and $R_4 = 20{,}000$ ohms, what should be the value of $a$ to give a break point at $e_1 = +50$ volts?

**3.8** (*a*) Derive the slope expression for the circuit of Fig. 3.14. (*b*) Verify the slopes given in Fig. 3.8.

**3.9** Determine suitable values for the hysteresis simulator of Fig. 3.20 in order to form the characteristic below:

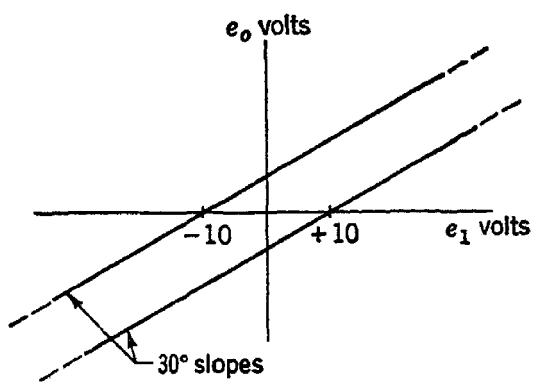



Fig. P3.9

**3.10** Show that the crossed-fields multiplier (Fig. 3.41) can be used as a divider by connecting the amplifier to the vertical deflecting plates, the divisor to the deflecting coil, and the dividend to the horizontal deflection plates.

**3.11** Sketch waveforms for the time-division multiplier like those of Fig. 3.47, using the other three polarity combinations of $x$ and $y$ in order to verify that the device is a four-quadrant multiplier.

CHAPTER 4

# Computing Amplifiers

## 4.1 Introduction

In this chapter we shall be principally concerned with the computing amplifier and its associated input and feedback elements, which together function to perform the linear operations of addition, sign inversion, constant multiplication, integration, and differentiation. We shall examine in turn what the desired ideal behavior of these devices is, what sort of departures from this ideal occur in practice, what errors are introduced into computer solutions by these departures, and finally what design techniques can be used to approximate the ideal.

## 4.2 Frequency characteristics of ideal computing elements

The building blocks from which computer setups are constructed are adders and integrators, and it is to these two components that we shall devote our major consideration. As we have seen, the adder can perform not only the addition of several inputs but also the multiplication of each of them by a different constant, and the integrator can perform both these tasks and integrate the sum with respect to time. (In feedback circuits of the type studied here, the sign-inversion property is inherent in both adders and integrators.)

The unavoidable errors in computing elements must be carefully analyzed. These may be considered to be either static or dynamic. Dynamic errors are best treated by the frequency-domain approach, and we shall examine the shapes of the ideal amplitude and phase-shift characteristics of computing components as functions of frequency. Departures of the shape of the actual characteristic from that of the ideal are the source of dynamic errors. All errors due to other sources will be regarded as static errors.

The frequency-domain behavior of the ideal adder is easily specified, since the output of the adder should not vary with frequency in any way. The behavior of the ideal integrator as a function of frequency is readily

found by integrating $A \sin \omega t$ to obtain $(-A/\omega) \cos \omega t$. The amplitude function is inversely proportional to frequency, and the phase function has a 90° lag. Similarly, the ideal differentiator has an amplitude function directly proportional to frequency and a phase function that has a 90° lead. Physically realizable components must depart from these ideals, since infinite gain is not attainable either at zero frequency as required by the ideal integrator or at infinite frequency as required by the ideal differentiator and since even the adder cannot maintain constant gain out to infinite frequency because stray shunting capacitance

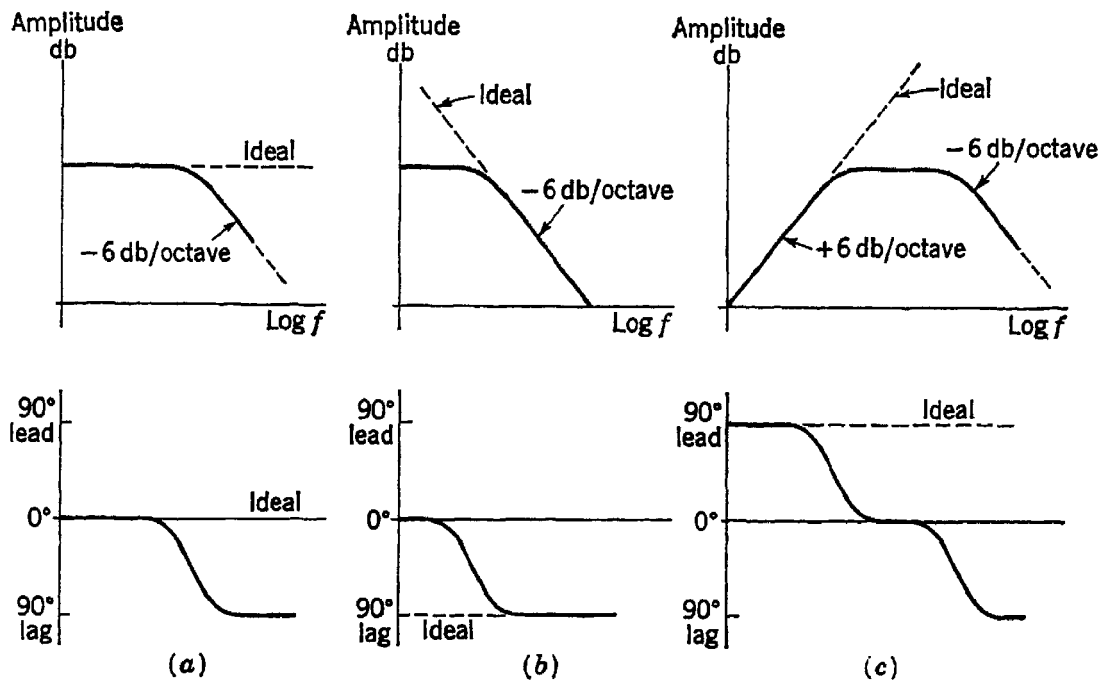



Fig. 4.1 Amplitude and phase characteristics, ideal and physically realizable: (*a*) adder, (*b*) integrator, (*c*) differentiator

must ultimately short-circuit the output at a high frequency. Figure 4.1 shows the ideal amplitude and phase characteristics of these three elements together with curves showing the simplest physically realizable approximation to them. (The phase characteristics are shown relative to the inherent 180° phase shift.) The ideal adder should produce an output uniform over the entire frequency spectrum and shifted in phase from the input by 180°. Because of the presence of shunting capacitances, the output of the physically realizable adder must fall off at −6 db/octave at high frequencies and in most cases will fall off even more rapidly at still higher frequencies as the shunting capacitances of the individual stages of the amplifier begin to contribute their −6 db/octave to the output slope. Each such −6-db/octave slope component contributes a phase lag of 90° to the phase characteristic. A good adder will have constant output up to very high frequencies.

The ideal amplitude response for an integrator is shown in Fig. 4.1*b*. Infinite gain is not possible, and the best realizable approximation at low frequencies is a response with a high constant value. That is, at low frequencies the amplitude response must depart from the ideal −6-db/octave slope. Similarly, the ideal phase response should display a lag of 90° from the 180° value that characterizes the amplifier circuit. At the low frequency at which the amplitude response leaves its −6-db/octave slope, the phase response leaves its 90° lag value. At high frequencies, the integrator amplitude-response curve will fall off even more rapidly than −6 db/octave, just as was observed in the adder.

A differentiator should have the +6-db/octave slope shown in Fig. 4.1*c*. The practical device cannot have infinite gain and must, therefore, level off and ultimately drop with at least −6-db/octave slope. Again, each 6-db/octave change in slope is accompanied by a 90° change in the phase response. As with the other circuits, the curve must ultimately fall at rates even faster than −6 db/octave, if multistage amplifiers are used.

## 4.3 Sources of error

The nonideal frequency characteristics of the preceding section have several causes, and there are also several sources of static errors. The following are the principal sources of errors:

1. Finite rather than infinite amplifier gain
2. Amplifier fall-off and phase shift at high frequencies
3. Shunt capacitance associated with input or feedback resistances
4. Leakage resistance associated with input or feedback capacitances
5. Stray capacitance to ground at the summing node
6. Dielectric absorption in capacitors
7. Drift voltages in the amplifier
8. Grid current at the amplifier input
9. Incorrect values for input and feedback impedances

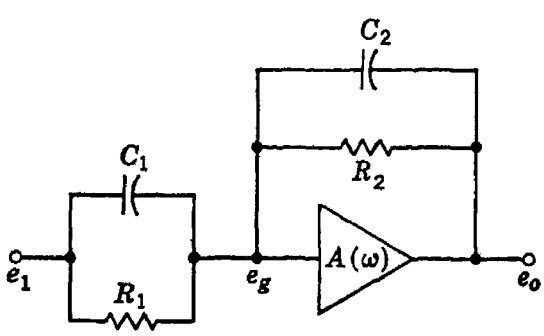


Fig. 4.2 Generalized adder, integrator, or differentiator

The first five of these items contribute to dynamic errors in the computer solution and the remainder to the static error.

Let us examine first the behavior of the first four items on this list. The symbols for our analysis are shown in Fig. 4.2. If the circuit is an adder, then $C_1$ and $C_2$ are small stray shunting capacitances. If the circuit is an integrator, $C_1$ is a small stray capacitance and $R_2$ is a very

large value corresponding to the leakage resistance of the integrating capacitor. If the circuit is a differentiator, then $R_1$ is the leakage resistance of the differentiating capacitor and $C_2$ is a small stray capacitance. For our present purposes we may safely neglect the minute current that flows into the amplifier input and write in Laplace-transform notation

$$\frac{E_1(s) - E_g(s)}{R_1(1/sc_1)/(R_1 + 1/sc_1)} = \frac{E_g(s) - E_o(s)}{R_2(1/sc_2)/(R_2 + 1/sc_2)} \tag{4.1}$$

and

$$E_o(s) = A(s)E_g(s) \tag{4.2}$$

Eliminating $E_g(s)$, we obtain

$$\frac{E_o(s)}{E_1(s)} = -\frac{Z_2/Z_1}{1 - (1/A)(1 + Z_2/Z_1)} \tag{4.3}$$

where $$Z_1 = \frac{R_1(1/sc_1)}{R_1 + 1/sc_1} \quad \text{and} \quad Z_2 = \frac{R_2(1/sc_2)}{R_2 + 1/sc_2}$$

If for the moment we regard the gain $A$ as being infinite, we may write

$$\frac{E_o(s)}{E_1(s)} = -\frac{R_2}{R_1}\frac{1 + sT_1}{1 + sT_2} \quad \text{for the adder} \tag{4.4}$$

$$\frac{E_o(s)}{E_1(s)} = -\frac{1}{R_1C_2}\frac{(1 + sT_1)T_2}{1 + sT_2} \quad \text{for the integrator} \tag{4.5}$$

and

$$\frac{E_o(s)}{E_1(s)} = -R_2C_1\frac{1 + sT_1}{(1 + sT_2)T_1} \quad \text{for the differentiator} \tag{4.6}$$

where $T_1 = R_1C_1$ and $T_2 = R_2C_2$. For the adder both $T_1$ and $T_2$ should be small. The integrator requires $T_1$ small and $T_2$ large, and the differentiator should have $T_1$ large and $T_2$ small. If these conditions are fulfilled, then Eqs. (4.4), (4.5), and (4.6) become the expressions for the ideal adder, integrator, and differentiator, respectively. It is interesting to note that, if $T_1$ and $T_2$ are equal, the circuit functions as an adder no matter what their value.

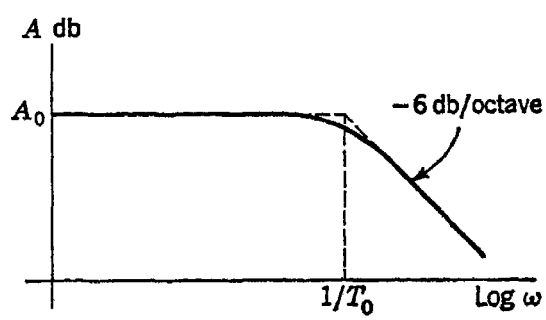


Fig. 4.3 Amplitude characteristic of a simple amplifier

Let us next replace the gain $A(s)$ by an approximate expression $A(s) = -A_0/(1 + sT_0)$. This expression represents a d-c amplifier with the response curve of Fig. 4.3. As we shall see later, most amplifiers will fall off more rapidly than −6 db/octave at high frequencies. These more rapid fall-offs must be taken into account when we consider the stabilization of the feedback amplifier, but they may be safely neglected in our error analysis.

The transfer ratio for the adder now becomes

$$\frac{E_o(s)}{E_1(s)} = -\frac{R_2}{R_1}\frac{(1 + sT_1)/(1 + sT_2)}{1 + \dfrac{1 + sT_0}{A_0}\left(1 + \dfrac{R_2}{R_1}\dfrac{1 + sT_1}{1 + sT_2}\right)} \tag{4.7}$$

Since $T_1$ and $T_2$ are both due to stray capacitances, we may assume them small. This enables us to neglect terms of the form $s^2T_1T_2$ and also to replace $1/(1 + sT_2)$ by $1 - sT_2$, the first two terms of its binomial expansion, obtaining

$$\frac{E_o(s)}{E_1(s)} = -\frac{R_2}{R_1}\frac{1 + s(T_1 - T_2)}{1 + [(1 + sT_0)/A_0]\{1 + (R_2/R_1)[1 + s(T_1 - T_2)]\}} \tag{4.8}$$

Ordinarily $A_0 \gg (1 + R_2/R_1)$. With this further approximation and again neglecting terms in $s^2$, $s^3$, etc., the expression may be put in the form

$$\frac{E_o(s)}{E_1(s)} = -\frac{R_2}{R_1}\frac{1}{1 + s[(T_0/A_0)(1 + R_2/R_1) - (T_1 - T_2)]} \tag{4.9}$$

which shows that the effective high-frequency time constant for the adder is $(T_0/A_0)(1 + R_2/R_1) - (T_1 - T_2)$. To make this quantity small, $R_2/R_1$ should be small, $T_1 - T_2$ should be either zero or positive, $T_0$ should be small, and $A_0$ should be large.

Let us evaluate the adder time constant for a reasonable set of parameter values, as listed below:

$A_0 = 10^5$ (100-db voltage gain)
$T_0 = 1/10$ sec
$R_1 = 1$ megohm
$R_2 = 2$ megohms
$T_1 = 10^{-5}$ sec
$T_2 = 2 \times 10^{-5}$ sec

The effective time constant is, therefore, $1.3 \times 10^{-5}$ sec; i.e., the adder response is down 3 db at $\omega = 1/1.3 \times 10^{-5}$ radians, or $f \cong 12{,}000$ cycles.

The step-function response of the adder is also of interest, and can be obtained as follows:

$$E_o(s) = -\frac{R_2}{R_1}\frac{1}{1 + sT_A}E_1(s) \tag{4.10}$$

$$E_o(s) = -\frac{R_2}{R_1}\frac{1}{1 + sT_A}\frac{1}{s} \tag{4.11}$$

$$e_o(t) = -\frac{R_2}{R_1}[U(t) - \epsilon^{-t/T_A}] \tag{4.12}$$

where $U(t)$ is the unit step function and $1/s$ is its Laplace transform. $T_A$ represents the effective adder time constant. The curve of Eq. (4.12) is shown in Fig. 4.4, with its sign reversed to facilitate comparison with the input.

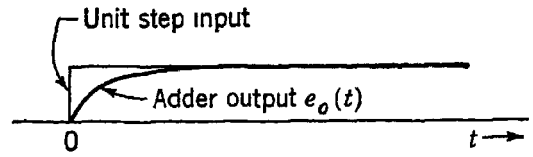



Fig. 4.4 Response of an adder to a unit step

A similar analysis will now be made for the integrator. The transfer ratio for the integrator may be written

$$\frac{E_o(s)}{E_1(s)} = \frac{1}{Z_1(s)} \frac{A(s)}{1 - A(s)} \frac{1}{[1/Z_2(s)] + [1/Z_1(s)][1/1 - A(s)]} \tag{4.13}$$

Now, if

$$Z_2(s) = \frac{R_2}{1 + sR_2C_2}$$

$$Z_1(s) = \frac{R_1}{1 + sR_1C_1}$$

and

$$A(s) = -\frac{A_0}{1 + sT_0}$$

then

$$\frac{E_o(s)}{E_1(s)} = -\frac{1 + sR_1C_1}{R_1} \frac{A_0}{1 + A_0 + sT_0} \frac{1}{(1/R_2) + sC_2 + (1/R_1 + sC_1)(1 + sT_0)/(1 + A_0 + sT_0)} \tag{4.14}$$

If $A_0 \gg 1$, $C_1/A_0 \ll 1$, $T_0/A_0R_1 \ll 1$, and second-order terms in $s$ are neglected, this becomes

$$\frac{E_o(s)}{E_1(s)} = -\frac{1}{R_1C_2} \frac{1}{1 + s(T_0/A_0 - R_1C_1)} \frac{1}{s + 1/R_2C_2 + 1/A_0R_1C_2} \tag{4.15}$$

The corresponding expression for the integrator with no stray input shunting capacitance and no shunt leakage conductance in its feedback path is

$$\frac{E_o(s)}{E_1(s)} = -\frac{1}{R_1C_2} \frac{1}{1 + sT_0/A_0} \frac{A_0R_1C_2}{1 + sA_0R_1C_2} \tag{4.16}$$

We may note two differences between these expressions. The presence of leakage in the integrating capacitor changes the low-frequency time constant from $T_L = A_0R_1C_2$ to a value given by

$$\frac{1}{T'_L} = \frac{1}{R_2C_2} + \frac{1}{A_0R_1C_2}$$

The second difference is the effect of $C_1$, the input shunting capacitance, which reduces the high-frequency time constant from $T_0/A_0$ to $T_0/A_0 - R_1C_1$. To obtain an idea of typical magnitudes of these effects, let

$A_0 = 10^5$, $T_0 = 1/10$, $R_1 = 1$ megohm, $R_2 = 500$ megohms, $C_1 = 10$ $\mu\mu$f, and $C_2 = 1$ $\mu$f. Then

$$\frac{E_o(s)}{E_1(s)} = -\frac{1}{1 + s(10^{-6} - 10^{-5})}\frac{1}{s + 2 \times 10^{-3} + 10^{-5}} \tag{4.17}$$

For these values the effect of the leakage resistance upon the low-frequency time constant $T_L$ is very large, since it is increased by a factor of almost 200. This effect emphasizes the importance of using high-quality integrating capacitors. The high-frequency time constant is apparently reduced from $10^{-6}$ to $-9 \times 10^{-6}$, suggesting that the over-all frequency characteristic of the system will rise at high frequencies. This anomalous behavior will not actually occur, of course, because the terms neglected in deriving Eq. (4.17) become significant at high frequencies. Nevertheless, Eq. (4.17) does demonstrate that a small amount of capacitance shunting $R_1$ can have a beneficial effect upon the integrator's high-frequency response.

Either Eq. (4.15) or (4.16) may be expressed as follows:

$$\frac{E_o(s)}{E_1(s)} = -\frac{1}{K}\frac{1}{1 + sT_H}\frac{T_L}{1 + sT_L} \tag{4.18}$$

where $T_H$ is the effective high-frequency time constant and $T_L$ the effective low-frequency time constant. Let us now examine the integrator errors resulting from failure of $T_H$ to be zero and of $T_L$ to be infinite. We shall consider a unit step function to be applied as the input. Then

$$E_o(s) = -\frac{1}{K}\frac{1}{1 + sT_H}\frac{T_L}{1 + sT_L}\frac{1}{s} \tag{4.19}$$

The corresponding partial-fraction expansion is

$$E_o(s) = -\frac{T_L}{K}\left(\frac{1}{s} + \frac{T_H}{T_L - T_H}\frac{1}{s + 1/T_H} - \frac{T_L}{T_L - T_H}\frac{1}{s + 1/T_L}\right) \tag{4.20}$$

from which it follows that

$$e_o(t) = -\frac{T_L}{K}\left[U(t) + \frac{T_H}{T_L - T_H}\epsilon^{-t/T_H} - \frac{T_L}{T_L - T_H}\epsilon^{-t/T_L}\right] \tag{4.21}$$

A sketch of this function (or rather of its negative) is shown in Fig. 4.5.

The failure of $T_H$ to be zero means that the integrator output does not contain enough high frequencies to allow it to exhibit the abrupt change in slope required for faithful integration of the step. The initial slope of the actual response is zero, instead of $1/K$ as ideally it should be. The effect of a finite rather than an infinite value for $T_L$ is to make the response

level off at an ultimate limit of $T_L/K$ instead of increasing uniformly forever. Of course, the level $T_L/K$ is ordinarily far beyond the output voltage attainable with a computing amplifier, and in practice we are more concerned with small deviations from the ideal that occur within the amplifier's dynamic range.

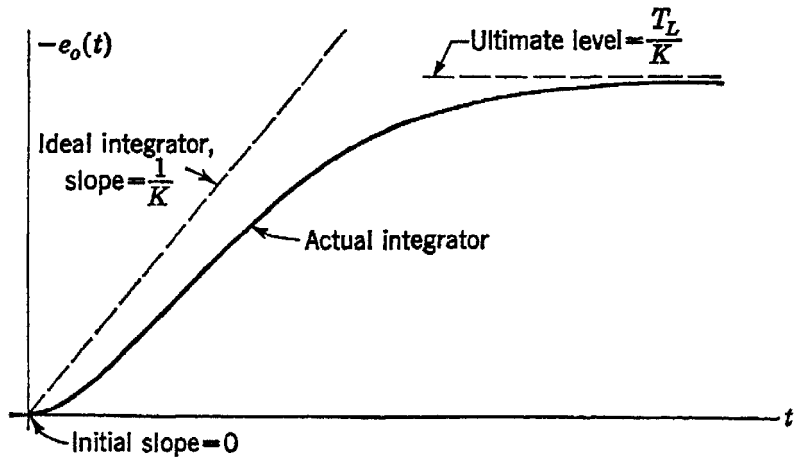



Fig. 4.5 Response of an integrator to a unit step

To complete our study of the basic circuits, let us briefly examine the differentiator, simplifying our analysis by ignoring leakage resistance. The transfer ratio may be written

$$\frac{E_o(s)}{E_1(s)} = -sC_1 \frac{A_0}{1 + A_0 + sT_0} \frac{R_2}{1 + sC_2R_2 + sC_1R_2(1 + sT_0)/(1 + A_0 + sT_0)} \tag{4.22}$$

If we take $A_0 \gg 1$, this may be approximated as

$$\frac{E_o(s)}{E_1(s)} = -sC_1 \frac{1}{1 + sT_0/A_0} \frac{R_2}{1 + sC_2R_2 + (sC_1R_2/A_0)[1 + s(T_0 - T_0/A_0) - s^2T_0^2/A_0]} \tag{4.23}$$

which may be further simplified by neglecting terms in $s^2$ and $s^3$ in the second factor of the denominator:

$$\frac{E_o(s)}{E_1(s)} = -sC_1 \frac{1}{1 + sT_0/A_0} \frac{R_2}{1 + s(R_2C_2 + R_2C_1/A_0)} \tag{4.24}$$

Ideally, both $T_0/A_0$ and $R_2C_2 + R_2C_1/A_0$ should be very small. The initial departure from the +6-db/octave slope of the ideal differentiator will be controlled by the larger of these two quantities.

An analysis of the step-function response of the differentiator of Eq. (4.24) can be carried out in much the same manner as for the adder and

the integrator. Although differentiators are not sufficiently useful to warrant presenting such an analysis here, the reader is urged to carry it through for himself to develop his understanding of the device. The principal objection to the use of differentiators is that they tend to exaggerate irregularities in a function rather than to smooth them as integrators do. In some cases they can produce spurious responses that totally conceal the desired response (e.g., see Prob. 4.2). Noise signals are amplified by differentiators and tend to mask the desired signal as well as to introduce d-c errors when rectified by tubes or transistors in

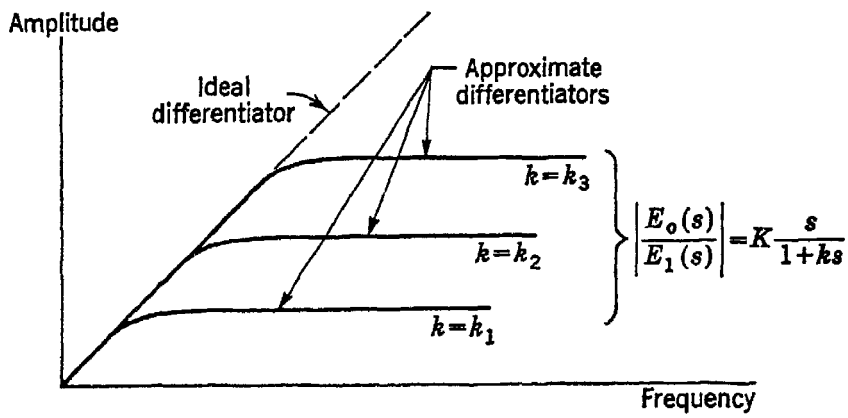



Fig. 4.6 Amplitude versus frequency curves for approximate differentiators

the amplifier. Differentiators are not suited for use with line-segment function generators, because the discontinuous slope of their output produces staircase functions and impulse functions upon successive differentiation.

Although the use of differentiators is usually inadvisable, it occasionally happens that it cannot be avoided. For example, in generalized integration with respect to a variable other than time, we integrate the product of $f(x)\,dx/dt$, and it may be that $dx/dt$ can be formed only by differentiating $x$. An approximate differentiator with adjustable maximum gain is useful in such cases. Figure 4.6 shows the amplitude versus frequency plot for an approximate differentiator. The transfer ratio of such a system is

$$\frac{E_o(s)}{E_1(s)} = K\frac{s}{1+ks}$$

From Eq. (4.24) we see that one way of realizing an approximate differentiator is to shunt $R_2$, the feedback resistance, with a capacitance $C_2$. Then, if $C_1/A_0 \ll C_2$ and if $s(T_0/A_0) \ll 1$, Eq. (4.24) becomes

$$\frac{E_o(s)}{E_1(s)} = -sC_1R_2\frac{1}{1+sR_2C_2}$$

Thus adjustment of $C_2$ controls the maximum gain, which is $C_1/C_2$, and the "corner" frequency, i.e., the frequency at which the output voltage is 3 db below its ultimate level. This frequency, in radians, is $1/R_2C_2$.

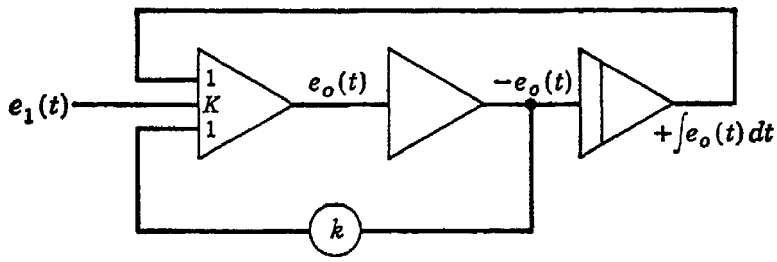



Fig. 4.7 A circuit for an approximate differentiator

The difficulty of obtaining well-shielded calibrated variable capacitors may make this circuit inconvenient to use. The variable capacitor can be avoided by putting the transfer ratio in the form

$$\frac{E_o(s)}{E_1(s)} = K\frac{s}{1 + (1 - k)s}$$

and then finding a computer setup to implement this function. Such a setup is shown in Fig. 4.7. As $k$ approaches unity, the maximum gain and the corner frequency both rise toward limits imposed by the computing amplifiers. In practice the noise level of the circuit determines the maximum value for $k$. Since $k$ can in some systems be allowed to be very nearly unity, a finer adjustment may be needed than can be provided by only a potentiometer. This adjustment can be provided by putting a fixed resistor in series with the potentiometer winding, as shown in Fig. 4.8.

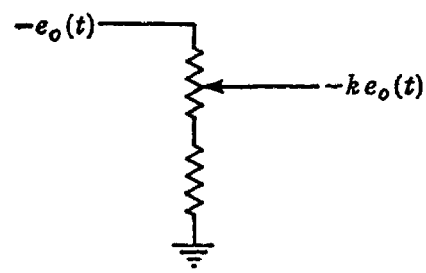



Fig. 4.8 Fine-adjustment circuit for $k$ near unity

The results derived in Eqs. (4.9), (4.15), and (4.24) of this section are due originally to P. C. Dow.[1]

## 4.4 Errors in computer solutions

The treatment of adder and integrator errors in the preceding section furnishes the basis on which to analyze the errors arising in computer setups for the solution of ordinary linear differential equations with constant coefficients. A general approach to such an error analysis is due originally to Macnee[2] and has been supplemented by recent work of Marsocci[3] and of Dow.[1] The dynamic errors, to which we shall limit our discussion in this section, arise from the circumstance that practical adders and integrators have characteristics that depart from the ideal. Each addition or integration in the equation to be solved is implemented

in the computer by a device that is more complex in its behavior with respect to frequency than the process it is approximating. The differential equation implemented by the computer is, therefore, of higher order than that of the problem. It is desirable to know something of the differences between the solution of this "machine equation" and the solution of what we shall call the "problem equation".

Let us consider the differential equation

$$+a_m \frac{d^m x}{dt^m} - a_{m-1} \frac{d^{m-1}x}{dt^{m-1}} + a_{m-2} \frac{d^{m-2}x}{dt^{m-2}} - \cdots \pm a_1 \frac{dx}{dt} \mp a_0 x = f(t) \quad (4.25)$$

The setup of Fig. 4.9 is one straightforward way of implementing this equation. It is assumed that all the coefficients $a_j$ of Eq. (4.25) are positive and that the signs of the terms alternate. This assumption considerably facilitates the manipulations to follow without seriously

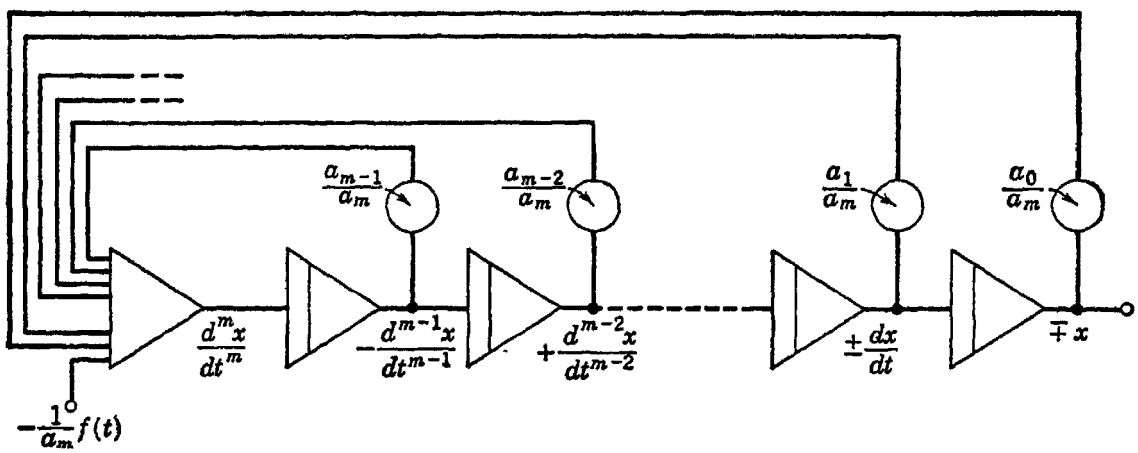



Fig. 4.9 Computer setup for Eq. (4.25)

affecting the validity of the results, since at most one more adder would be required to invert the signs of terms that did not fit the assumed pattern of alternation. It is also assumed that the quantities $a_j/a_m$ are less than unity in magnitude or that, if they are not, an appropriate multiplying factor can be incorporated in the corresponding adder input without affecting the high-frequency performance of the adder.

If we let $I(s)$ be the integrator transfer ratio and $A(s)$ be the adder transfer ratio, then the adder output $E(s)$ will be

$$E(s) = A(s) \left\{ E(s) \left[ I(s) \frac{a_{m-1}}{a_m} + I^2(s) \frac{a_{m-2}}{a_m} + \cdots + I^{m-1}(s) \frac{a_1}{a_m} + I^m(s) \frac{a_0}{a_m} \right] - \frac{F(s)}{a_m} \right\} \quad (4.26)$$

Ideally, $A(s) = -1$ and $I(s) = -1/s$. If we substitute these functions into Eq. (4.26) and set $F(s) = 0$ to make the equation homogeneous,

we obtain

$$1 = -\left(-\frac{1}{s}\frac{a_{m-1}}{a_m} + \frac{1}{s^2}\frac{a_{m-2}}{a_m} - \frac{1}{s^3}\frac{a_{m-3}}{a_m} + \cdots \pm \frac{1}{s^{m-1}}\frac{a_1}{a_m} \mp \frac{1}{s^m}\frac{a_0}{a_m}\right) \quad (4.27)$$

Rearranging this, we get the familiar form of the characteristic equation

$$s^m - s^{m-1}\frac{a_{m-1}}{a_m} + s^{m-2}\frac{a_{m-2}}{a_m} - \cdots \pm s\frac{a_1}{a_m} \mp \frac{a_0}{a_m} = 0 \quad (4.28)$$

Now let us see the effect of imperfect components by letting

$$I(s) = -\frac{T_L}{1 + sT_L}\frac{1}{1 + sT_H} \qquad \text{and} \qquad A(s) = -\frac{1}{1 + sT_A}$$

where $T_L$ and $T_H$ are respectively the low-frequency and high-frequency time constants of the integrator and $T_A$ is the time constant of the adder. Substitution of these expressions will yield the following characteristic equation:

$$\left[s^2T_H + s\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L}\right]^m - \frac{a_{m-1}}{a_m}\left[s^2T_H + s\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L}\right]^{m-1} + \frac{a_{m-2}}{a_m}\left[s^2T_H + s\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L}\right]^{m-2} - \cdots \mp \frac{a_0}{a_m} = -sT_A\left[s^2T_H + s\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L}\right]^m \quad (4.29)$$

This equation is similar to Eq. (4.28), with $s$ replaced by $s^2T_H + s(1 + T_H/T_L) + 1/T_L$ and the right side set equal to $-sT_A[s^2T_H + s(1 + T_H/T_L) + 1/T_L]$. Equation (4.28) has $m$ roots, but the actual machine characteristic equation (4.29) has $2m + 1$ roots. Thus the effect of the imperfect components is to produce a machine equation very different from the problem equation, and a reasonably good solution can be obtained only if $m$ of the roots of (4.29) lie close to the $m$ roots of (4.28) and if the $m + 1$ extraneous roots have negative real parts so large in magnitude that the resulting exponential terms are quickly damped out and therefore make only a minor contribution to the output of the computer.

It is evident that Eq. (4.29) will approximate (4.28) if $T_L$ is very large and $T_H$ and $T_A$ very small. Following Macnee's procedure, let us denote (4.28) by $C\{s\} = 0$. Then (4.29) becomes

$$C\left\{s^2T_H + s\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L}\right\} = -sT_A\left[s^2T_H + s\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L}\right]^m \quad (4.30)$$

If we denote the roots of (4.28) by $s_n$, then the left side of (4.30) is zero whenever $s^2T_H + s(1 + T_H/T_L) + 1/T_L = s_n$. A power-series expansion of the left side of (4.30) in the vicinity of its zeros gives

$$C\left\{s^2T_H + s\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L}\right\} = \left[s^2T_H + s\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L} - s_n\right] C'\{s\} + \text{higher-order terms} \tag{4.31}$$

Now, if the errors are to be minimized, we must have

$$\frac{1}{T_L} \ll s_n \ll \frac{1}{T_H} \text{ or } \frac{1}{T_A} \tag{4.32}$$

Making use of this fact and letting $s = s_n + e_n$, where $e_n$ represents the error, we may substitute (4.31) into (4.30) to obtain

$$\left[s^2T_H + s\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L} - s_n\right] C'\{s\} \cong -sT_A\left[s^2T_H + s\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L}\right]^m$$

$$\left[(s_n + e_n)^2T_H + (s_n + e_n)\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L} - s_n\right] C'\{s\} \cong -T_A(s_n + e_n)\left[(s_n + e_n)^2T_H + (s_n + e_n)\left(1 + \frac{T_H}{T_L}\right) + \frac{1}{T_L}\right]^m$$

$$\left[(s_n + e_n)^2T_H + e_n + \frac{1}{T_L}\right] C'\{s\} \cong -T_As_n(s_n{}^2T_H + s_n)^m \cong -T_As_n{}^{m+1}(s_nT_H + 1)^m$$

$$\left(T_Hs_n{}^2 + e_n + \frac{1}{T_L}\right) C'\{s\} \cong -T_As_n{}^{m+1}$$

$$e_n = -\frac{1}{T_L} - \frac{T_As_n{}^{m+1}}{C'\{s\}} - T_Hs_n{}^2 \tag{4.33}$$

Equation (4.33) gives the deviations of $m$ of the roots of (4.29) from the desired roots of (4.28) that they approximate. The locations of the $m + 1$ extraneous roots may be approximated by assuming $s \gg s_n$. This makes $s^m \gg s^{m-1} \gg s^{m-2} \cdots$, and Eq. (4.29) then simplifies to

$$(s^2T_H + s)^m \cong -sT_A(s^2T_H + s)^m \tag{4.34}$$

or

$$s^m(1 + sT_H)^m(1 + sT_A) \cong 0$$

$m$ of the roots of this equation are zero or approximately so. These correspond to the $m$ roots of (4.28), which we have here assumed small.

One of the remaining roots is at $s = -1/T_A$, and the other $m$ are at $s = -1/T_H$.

To summarize this discussion, we may note that imperfect adders and integrators introduce errors (1) by displacing the roots of the characteristic equation from their true locations, and (2) by introducing extra roots that give rise to spurious responses. The effect of the root displacement is to introduce a damping factor, so that in place of each response term of the form $k\epsilon^{s_n t}$ there is a term $k\epsilon^{s_n t}\epsilon^{e_n t}$, where $e_n$ is the real quantity given by Eq. (4.33). A good test of the quality of computer amplifiers consists in setting up the equation of simple harmonic motion

$$\ddot{x} + \omega^2 x = 0 \tag{4.35}$$

The roots of the characteristic equation lie at $s = \pm j\omega$, and the system is very sensitive to the presence of damping, since any value of $e_n \neq 0$ should be evidenced by a change of amplitude if the computer is allowed to run for long enough. Macnee has shown, for instance, that, with $T_H = 0$ and $T_L = \infty$ (i.e., perfect integrators), $T_A = (1/2\pi) \times 10^{-4}$, and $\omega = 10\pi$, the amplitude will increase by 1 per cent after $4/\pi$ sec. However, high-quality commercially available computing equipment will do much better than this.

Although the Macnee analysis is entirely satisfactory for our purposes since it shows clearly the effects of the imperfect components, a more exact analysis is sometimes needed, and for this purpose Marsocci's treatment is useful, since it makes no assumptions about the relative magnitudes of $T_L$, $T_H$, and $T_A$. Another method, due to Dow, makes it possible to determine the $m$th-order machine equation that has the $m$ altered roots for its solution. Comparison of this equation with the $m$th-order problem equation can give useful information about the machine errors.

## 4.5 Automatic drift stabilization[4,5]

The requirement that computer amplifiers respond to d-c levels at their inputs and not merely to changes in these levels gives rise to one of the major design problems of the analog computer, namely, the reduction of the effects of drift voltages. Any change in the d-c operating conditions of circuits that transmit d-c quantities is indistinguishable from a change in the d-c input signals and, therefore, constitutes a source of error. For example, in the simple circuit of Fig. 4.10, the voltage across the load resistor varies when the

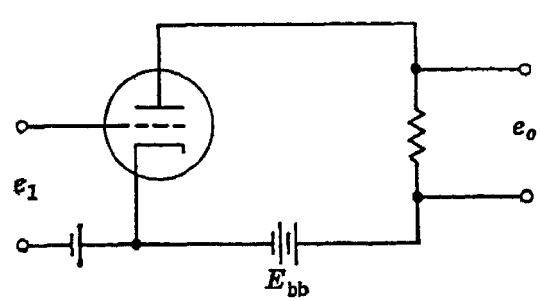


Fig. 4.10 A d-c amplifier

input signal changes, but it is also affected by changes in plate supply voltage ($E_{bb}$), cathode emission, and transconductance. Ordinarily all these quantities and others, too, are varying slowly, thus producing a drift of output voltage away from its correct value. Drifts in the first stages of an amplifier are amplified in all the succeeding stages and can grow to a sizable value at the output point.

Drift voltages can be kept to a minimum by careful design, but they can never be completely eliminated. Two alternative techniques are available for minimizing the effects of what might be called the "irreducible" drift voltages, that is, drift voltages that remain after a reasonable engineering effort aimed at their elimination. One technique is to make a manual adjustment of bias voltage to set the output to zero with zero input, after the initial "warm-up" drift, and then to rely on this adjustment during at least a single run of the computer. A warm-up time of 15 to 30 min is usually sufficient, since subsequent drifts are less pronounced. If the amplifier and power supplies are well designed, the drift that occurs between manual adjustments may be within the desired error limits in many situations. After the initial warm-up, it is desirable to be able to operate the computer for several hours without repeating the drift-balancing adjustment. The other technique is to use an auxiliary amplifier that increases the loop gain at low frequencies and therefore acts to reduce the effect of the drift voltages. This method is widely used, and we shall examine it in some detail.

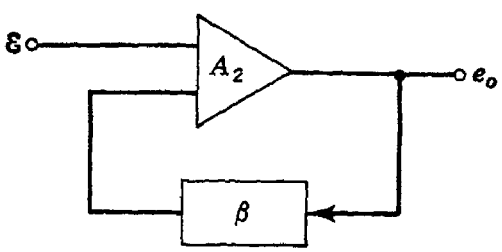


Fig. 4.11 Generalized feedback circuit

First let us consider the generalized feedback circuit of Fig. 4.11. The output voltage $e_o$ can be found from the relation

$$e_o = \varepsilon A_2 + e_o A_2 \beta \tag{4.36}$$

Thus,

$$e_o = \frac{\varepsilon A_2}{1 - A_2\beta} \tag{4.37}$$

The cumulative effect of all the drift-voltage components within the amplifier $A_2$ can be regarded as the same as the effect due to a single equivalent drift voltage $\varepsilon$ acting at the input of the amplifier. If the product $A_2\beta \gg 1$, then $|e_o| = \varepsilon/\beta$. Now let an auxiliary drift-free amplifier of gain $A_1$ be inserted ahead of $A_2$ as shown in Fig. 4.12.

Evidently

$$e_o = \frac{\varepsilon A_2}{1 - A_1 A_2 \beta} \tag{4.38}$$

For $A_1A_2\beta \gg 1$, this reduces to $|e_o| = \varepsilon/A_1\beta$. The inclusion of $A_1$ has thus reduced the output drift voltage to $1/A_1$ times its former value.

The crux of this technique lies in the provision of amplifier $A_1$, which is free from drift effects. The usual technique for eliminating effects of

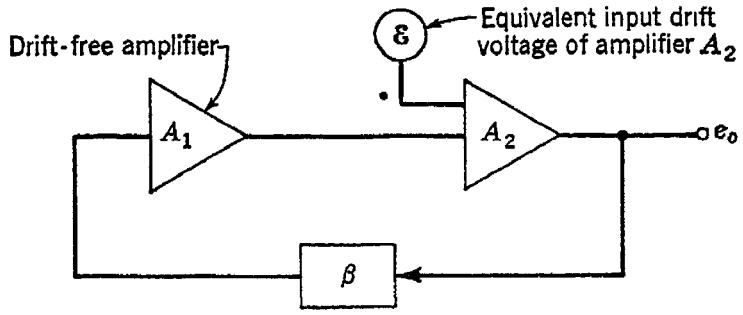



Fig. 4.12 Inclusion of a drift-free amplifier

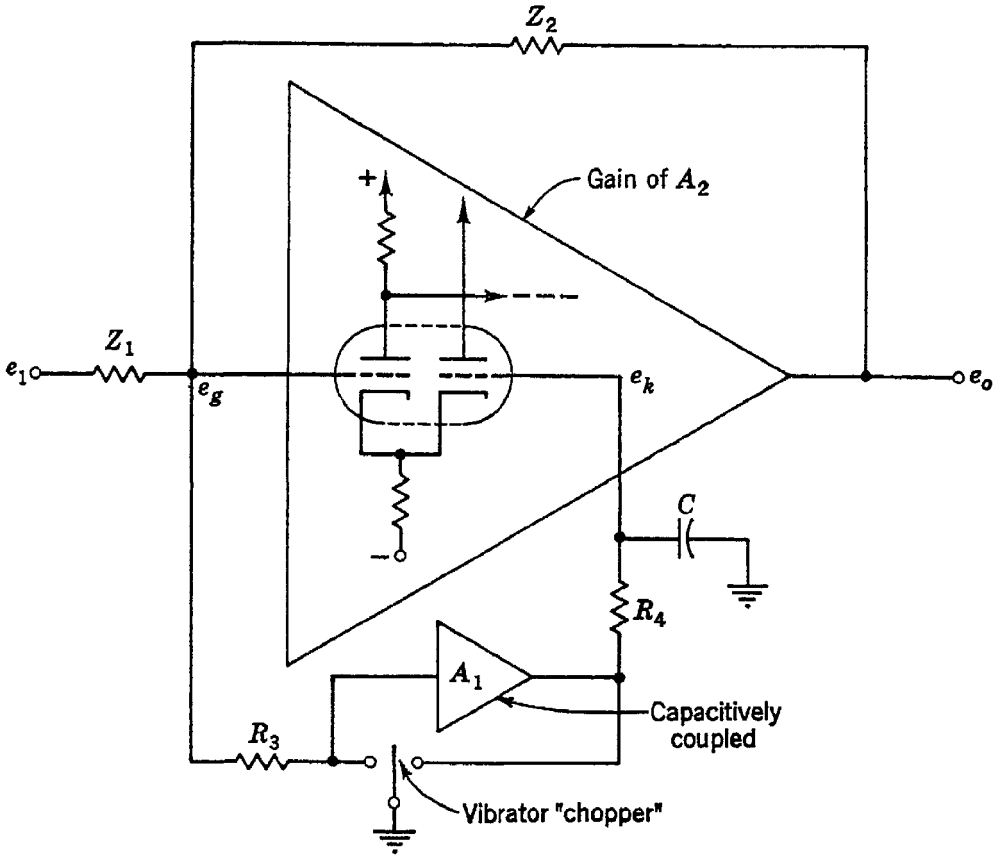



Fig. 4.13 The Goldberg drift-stabilization system

drift in $A_1$ is to construct it as a capacitively coupled amplifier. Then, since it will not respond to d-c inputs, the d-c input signal is amplitude-modulated upon an a-c carrier. The amplified carrier is demodulated at the output before the signal is applied to $A_2$. The circuit of Fig. 4.13 shows the essentials of Goldberg's widely used method of accomplishing this. Because of the feedback through $Z_2$, drift voltages produced anywhere within $A_2$ are detectable at the summing node. The voltage from

this point is applied through $R_3$ to one contact of the vibrator, which is intermittently grounded. The resulting "chopped" wave is amplified in $A_1$ and demodulated at the other chopper contact before being filtered by $R_4C$ and applied to amplifier $A_2$ at the grid of a differential-amplifier stage.

Let us first analyze the behavior of the circuit with $A_2$ acting alone and the $A_1$ circuit disconnected. As usual,

$$\frac{e_1 - e_g}{Z_1} = \frac{e_g - e_o}{Z_2} \tag{4.39}$$

and, in addition,

$$e_o = (e_g + \varepsilon)A_2 \tag{4.40}$$

From these we readily obtain

$$e_o = -e_1 \frac{Z_2/Z_1}{1 - (1/A_2)[(Z_1 + Z_2)/Z_1]} - \varepsilon \frac{(Z_1 + Z_2)/Z_1}{1 - (1/A_2)[(Z_1 + Z_2)/Z_1]} \tag{4.41}$$

If the first stage of the $A_2$ circuit is a differential amplifier and if the output $e_k$ of the $A_1$ circuit is now connected to it, we find

$$e_o = (e_g + \varepsilon)A_2 - e_k A_2 \tag{4.42}$$

where $e_k = A_1 e_g$. Substituting these relations into (4.39), we find that

$$e_o = -e_1 \frac{Z_2/Z_1}{1 - [1/A_2(1 - A_1)][(Z_1 + Z_2)/Z_1]} - \varepsilon \frac{(Z_1 + Z_2)/Z_1}{1 - A_1 - (1/A_2)[(Z_1 + Z_2)/Z_1]} \tag{4.43}$$

Comparison of Eqs. (4.43) and (4.41) shows the improvement afforded by the auxiliary amplifier. Not only is the first term of (4.43) more nearly equal in magnitude to $-e_1 Z_2/Z_1$ than is the first term of (4.41), but the second term, which represents the drift, has been substantially reduced, since its denominator is approximately $A_1$ times as great as that in (4.41).

The reader will note that the circuit of Fig. 4.13 represents an arrangement slightly different from that of Fig. 4.12, since in 4.13 the feedback from the output of the d-c amplifier to its input is still present in addition to the component that is amplified in the drift-free amplifier. Thus the effective forward gain in 4.13 is $(1 - A_1)A_2$, whereas in 4.12 it is $A_1A_2$. Both circuits, however, show clearly the benefits of augmenting the loop gain with drift-free amplification.

Figure 4.14 shows the relative phases and polarities of the waveforms in the auxiliary-amplifier circuit. $e_g$ is assumed to be positive, and is alternately applied to $A_1$ and grounded through $R_3$. The amplified wave

(with d-c level removed) is applied to the other chopper contact, where it is grounded during those half-cycles when the input is not. The low-pass filter $R_4C$ serves to remove high-frequency components, so that a substantially d-c signal may be applied as $e_k$ to the differential amplifier in $A_2$. Since the purpose of the auxiliary amplifier is to augment the gain of the loop, it is evident that signals transmitted through the $A_1$ circuit should produce a voltage at the output of the differential amplifier that is in phase with the voltage produced at that point by the action

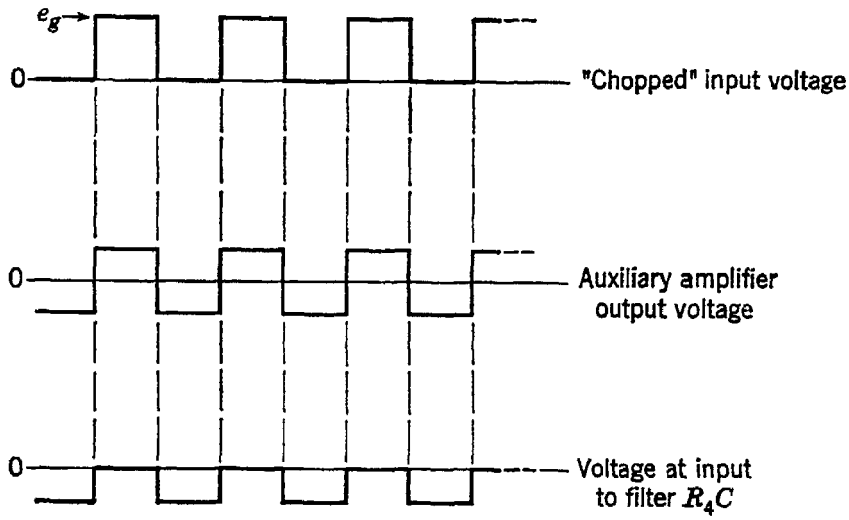



Fig. 4.14 Auxiliary amplifier waveforms

of the summing-node voltage upon the other grid of the differential amplifier. Since the two grids cause opposite effects upon the differential-amplifier output, the two signals must be of opposite polarity. If the auxiliary amplifier produces no net phase reversals, then the relative phases are as shown in Fig. 4.14, and a positive voltage $e_g$ at the summing node results in formation of a negative voltage $e_k$. If the auxiliary amplifier produced a net phase reversal, then the combination of $A_1$, $A_2$, and the feedback element $Z_2$ would be unstable. [$A_1$, as used in the derivation of Eq. (4.43), includes the polarity-reversing effect of the chopper as well as the gain of the capacitively coupled amplifier and is, therefore, a negative number.]

It is interesting to note that the addition of the "chopper stabilizer" has the additional advantage of raising the effective d-c gain of the computer amplifier by a large factor. This is of particular benefit in integrators, where the principal departure from ideal behavior is a leveling off of the gain at low frequencies. $\omega_u$, the break point for the unstabilized case, occurs at the point where the real and imaginary parts of the denominator of Eq. (4.41) are equal; i.e.,

$$\omega_u \cong \frac{1}{A_2R_1C_2}$$

where $R_1$ is the input resistance and $C_2$ the feedback capacitance. Similarly,

$$\omega_s \cong \frac{1}{A_2(1 - A_1)R_1C_2}$$

Since $A_1$ is a negative number, we may write

$$\frac{\omega_s}{\omega_u} = \frac{1}{1 + |A_1|}$$

as a measure of the improvement of the integrator behavior.

Curves showing the relative amplitude responses of $A_1$ and $A_2$, separately and in combination, are shown in Fig. 4.15. It will be observed

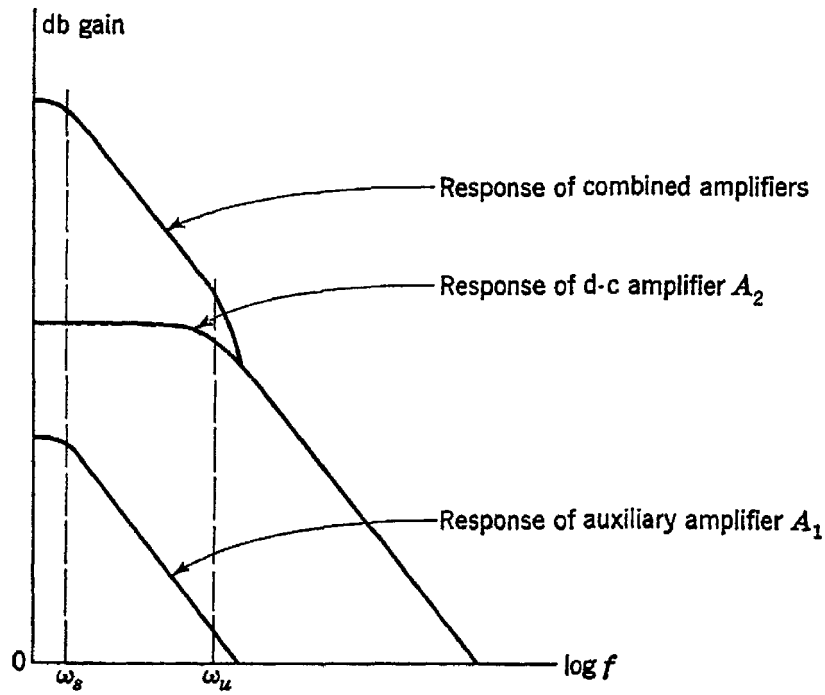



Fig. 4.15 Amplitude responses of $A_1$ and $A_2$

that the gain of the chopper amplifier starts to fall at a lower frequency than does that of the $A_2$ amplifier. This effect is due to the low-pass filtering action of the $R_4C$ circuit in Fig. 4.13, which is introduced to remove the chopper frequency and its harmonics as well as noise caused by the chopper. A filter that passed a wider band could be used with a high-frequency chopper, but reliable mechanical choppers have not been available for frequencies of operation above a few hundred cycles per second. The following values are representative of the frequency-response curves for typical commercially available computing equipment:

D-c amplifier:

| | |
|---|---|
| Zero-frequency voltage gain | 90 db |
| 3-db down frequency | 1.0 cycles |

Auxiliary amplifier:

| | |
|---|---|
| Zero-frequency voltage gain | 60 db |
| 3-db down frequency | 0.02 cycles |

Such a system would have an over-all gain of 150 db at zero frequency.

The reader might ask why it is necessary to include a direct-coupled amplifier in the system at all. In theory it should be possible to use the modulator-amplifier-demodulator system for all frequencies or else to use this method only for the low frequencies and to place a conventional capacitively coupled amplifier in tandem. These arrangements are shown in Fig. 4.16. In Fig. 4.16*a* all frequencies are modulated upon

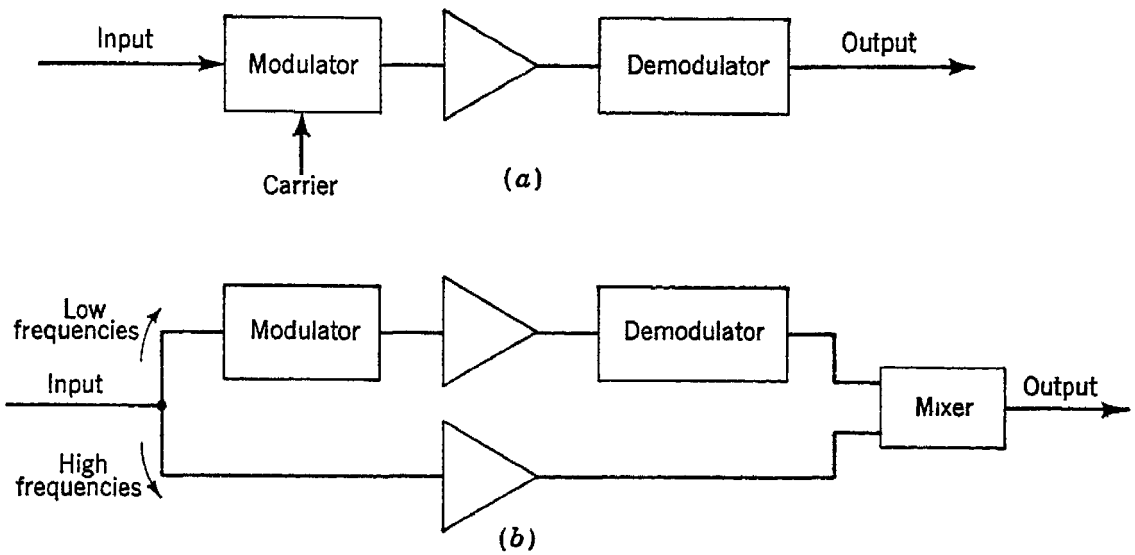



Fig. 4.16 Systems using capacitively coupled amplifiers

the carrier. Thus the carrier frequency must be high with respect to the highest frequency to be encountered in the use of the computer. Electromechanical modulators are not suitable for such use, and more complex electronic modulators would be needed. In *b* a vibrator chopper may be used, since only low frequencies are transmitted in the modulator channel and high frequencies pass through the other. Mixing at high levels is a problem in this system, as is matching of the gains of the tandem channels, and use of the chopper-stabilizer to increase the low-frequency loop gain has been found to be the most satisfactory method of reducing effects of drift. However, a very interesting circuit of this type which appears to have good possibilities has been described by Buckerfield.[6]

Although most computers that employ drift stabilization use an auxiliary amplifier for each computing amplifier, it is possible to use a single auxiliary amplifier time-shared by several computing amplifiers.[7]

## 4.6 Reduction of drift voltages in d-c amplifiers[8–10]

Although the drift-stabilization technique of the previous section is a useful way of reducing the effects of drift voltages, it should supplement rather than obviate a careful design effort to reduce the drift voltages themselves. Drift components are due principally to the following causes:

1. Changes in d-c supply voltages
2. Changes in voltage drops due to grid currents; integration of grid current
3. Changes in cathode emission
4. Changes in component values
5. Rectification of noise voltages
6. Changes in thermal potentials

The effects of all these variations are most severe when they occur in the input stages of the amplifier, since they are then amplified by all the following stages. Therefore, great care is to be used in the design of the input stages. Within the amplifier the principal circuit-design problems arise in connection with items 2 and 3 of the foregoing list. Item 1 is of equal importance from the point of view of the over-all system, but

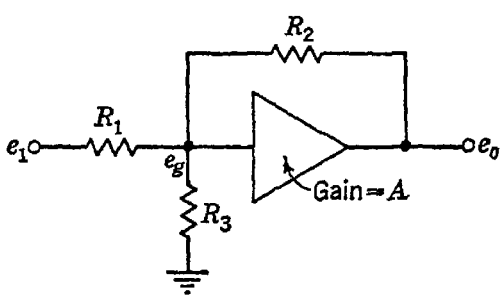



Fig. 4.17 Amplifier with finite input resistance

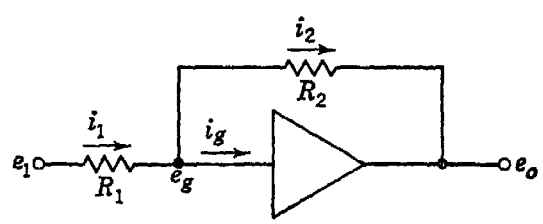



Fig. 4.18 Circuit for calculation of zero offset due to grid current

the design of voltage-stabilized power supplies is beyond the scope of this work, and the reader is referred to the literature for some very fine treatments of this topic.[11]

In the error analyses of Secs. 4.3 and 4.4, the amplifier was assumed to be a device with infinite input impedance, and the current in the series impedance to the summing node was taken to be equal to the current in the feedback impedance. It is readily seen that this approach is ordinarily justified in the analysis of dynamic errors. Let $R_3$ of Fig. 4.17 be an equivalent input resistance (assumed constant) of the amplifier. Then it follows that

$$e_o = -e_1 \frac{R_2/R_1}{1 - (1/A)[1 + R_2(1/R_1 + 1/R_3)]}$$

$R_3$ may well be many times greater than $R_2$ or $R_1$, so that it has negligible effect upon the transfer characteristic. Its effect upon amplifier drift may be more pronounced, however, since $R_3$ is seldom a constant value. $R_3$ is due mainly to current flow to the grid of the first tube of the amplifier. If this current were constant, it would produce a constant voltage drop in $R_1$ and $R_2$, and a steady output voltage would be observed with zero input. This voltage can be calculated from the circuit of Fig. 4.18 and is found to be

$$e_o = \frac{i_g R_2}{1 - (1/A)(1 + R_2/R_1)} \tag{4.44}$$

Thus, for moderately large values of $A$, $e_o \cong i_g R_2$. A simple adjustment of bias conditions within the amplifier could introduce a compensating offset to cancel this quantity. Unfortunately, $i_g$ is made up of many components and does not remain constant, so that it is not possible to eliminate its effect in this manner. The following is a list of sources of grid current in vacuum tubes:[12]

1. Initial velocity of electrons emitted from the cathode in the presence of zero field or slightly retarding field
2. Positive ions produced in the residual atmosphere by accelerated electrons
3. Positive ions emitted from the cathode
4. Thermionic emission from the grid when heated by radiation from the cathode
5. Photoelectric emission from the grid under external illumination and illumination from the hot cathode
6. Ionization of residual atmosphere by soft X rays emitted at the anode
7. Positive ions emitted at the anode
8. Secondary electron emission from the grid due to positive-ion bombardment
9. Secondary electron emission from the cathode due to positive-ion bombardment
10. Leakage current

Although many of these components are not large enough in magnitude to be troublesome in computer amplifiers, the variable nature of the grid current is evident. The following steps are helpful in reducing the grid current:

1. Component 1 can be reduced by operating with a negative bias of 2 volts or more.
2. Components 2, 6, 7, 8, and 9 can be reduced by operating at low screen and anode potentials.

3. Components 1, 3, 4, and 5 can be reduced by operating at reduced heater voltage.

4. Component 10 can be reduced by cleaning the tube base or removing it altogether and by operating the tube in a clean, dry atmosphere.

By choosing an input-stage tube type with low grid current and observing some or all of the foregoing precautions, the designer can keep the grid current in a well designed amplifier at less than $10^{-9}$ amp.

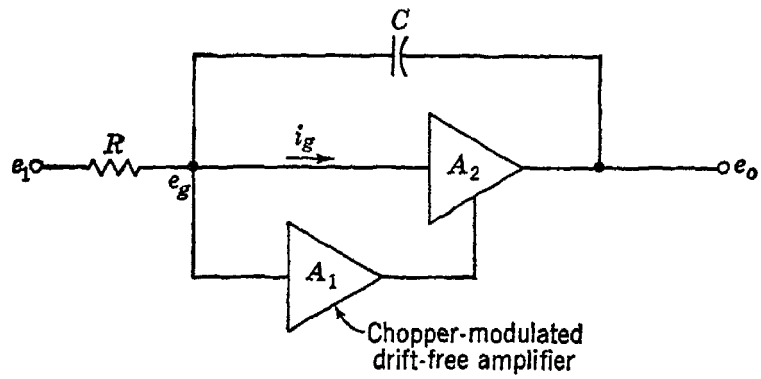



Fig. 4.19 Circuit to show effects of grid current when drift stabilization is used

An amplifier that exhibits no drift effects but has a constant grid current will display drift due to the integration of this grid current when it is connected as an integrator. As shown in Eq. (4.44), the grid current flows almost entirely in the feedback element. $10^{-8}$ amp of grid current will cause a 1-volt output in 100 sec with a 1-$\mu$f integrating capacitor. It is a simple matter to show that this drift is not affected by the usual chopper-stabilizer circuit, such as that of Fig. 4.19.

We may write

$$[E_1(s) - E_g(s)]\frac{1}{R} = I_g(s) + sC[E_g(s) - E_o(s)]$$

and

$$E_o(s) = A_2[E_g(s) + \varepsilon] + A_1A_2E_g(s)$$

From these we may obtain

$$E_o\left[\frac{1}{A_2(1 + A_1)} - \frac{sCR}{1 + sCR}\right] = \frac{E_1}{1 + sCR} - \frac{I_gR}{1 + sCR} + \frac{\varepsilon}{1 + A_1}$$

$$E_o \cong -\frac{E_1}{sCR} + \frac{I_g}{sC} - \frac{\varepsilon}{1 + A_1}\cdot\frac{1 + sCR}{sCR} \qquad \text{if } A_1A_2 \gg 1$$

Thus, even with the drift stabilizer, the term $I_g/sC$ is present. A system that avoids this difficulty is due to Hamer,[13] and is shown in Fig. 4.20.

We may write

$$E_o(s) = A_2[E_g(s) + \varepsilon] + A_1A_2E_g'(s)$$

and

$$[E_1(s) - E_g(s)]\frac{1}{R} = I_g(s) + sC[E_g(s) - E_o(s)]$$

and $$[E_1(s) - E_g'(s)]\frac{1}{R} = sC[E_g'(s) - E_o(s)]$$

These can be solved, to yield approximately

$$E_o \cong -\frac{E_1}{sCR} + \frac{I_g}{sC}\frac{1}{1 + A_1} - \frac{\varepsilon}{1 + A_1}\cdot\frac{1 + sCR}{sCR}$$

The output voltage contains a term proportional to the integral of the grid current but $1/(1 + A_1)$ times the size of the corresponding term in the former case.

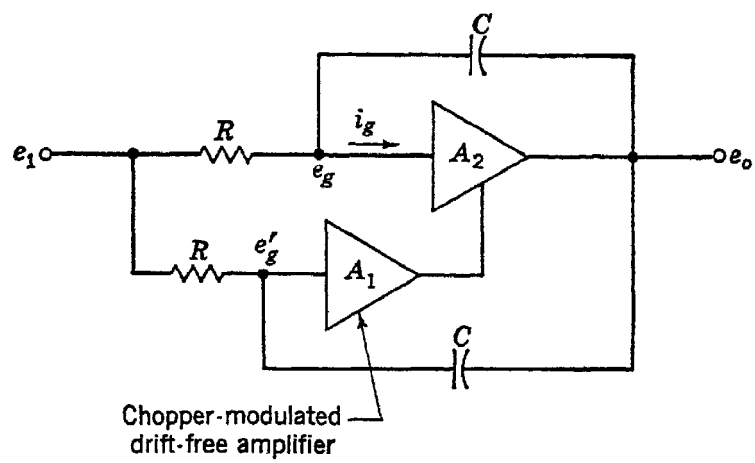



Fig. 4.20 Circuit for stabilization against drift due to integration of grid current

Another approach to the grid-current problem, which can be used if a drift-stabilizing amplifier is employed, consists in inserting a blocking capacitor between the summing node and the input grid of amplifier $A_2$. (A d-c path from grid to ground must be provided, of course.) The blocking capacitor has negligible effect upon the gain, since at low frequencies the gain through the auxiliary amplifier is large and at high frequencies the blocking-capacitor impedance is negligible. A good-quality capacitor must be used, so that absorption effects will not delay recovery of the amplifier after an overload. (When an overload occurs, the feedback path is, in effect, opened. The summing node is, therefore, no longer held at ground potential, and the blocking capacitor can receive a substantial charge.)

Circuit precautions may also be taken to reduce the effect of changes in cathode emission. Several ways exist for doing this;[14] two of these will be briefly described here. One technique, due to S. E. Miller,[15] is shown in Fig. 4.21. Two triodes in one envelope are used, rather than two separate triodes, in order to increase the likelihood that any emission changes that do occur will be the same for both tubes. Any such change can be regarded as introducing approximately equal voltages $\Delta E$ in series

with each cathode. If the plate current of tube 1 is not to change, the voltage $e_k$ must change by $-\Delta E$ volts in order to keep the effective grid-cathode and plate-cathode potentials at their original values. This

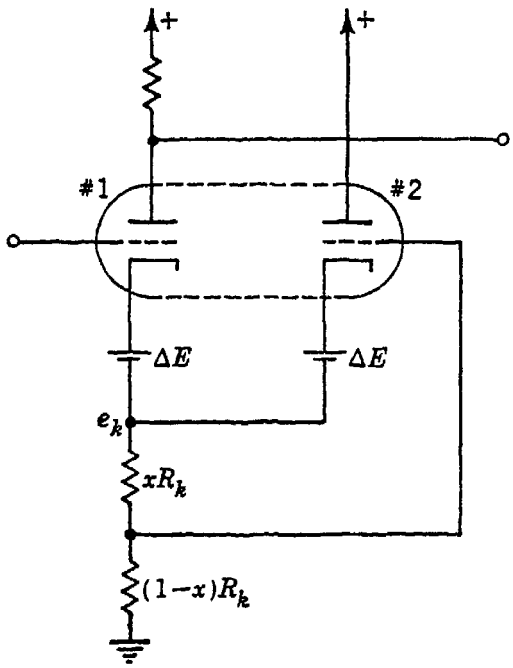



Fig. 4.21 Cancellation of drift due to emission changes

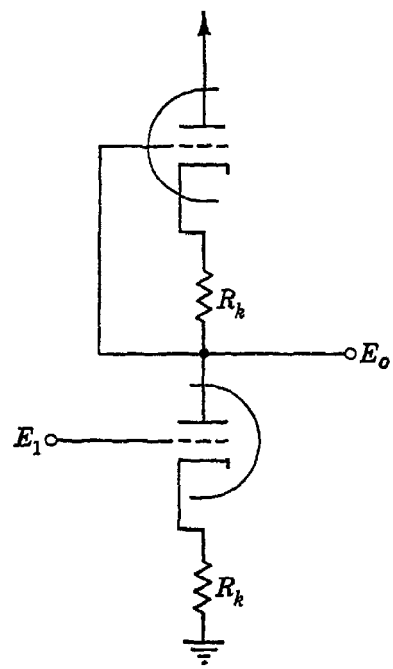



Fig. 4.22 Series triode circuit for cancellation of drift due to emission changes

change of $-\Delta E$ volts must be due only to a change in $i_{b2}$, the plate current of tube 2. Since the grid bias of tube 2 changes by $-\Delta E(1 - x)$ volts and since its effective plate-cathode voltage does not change, we see that

$$\Delta i_{b2} \cong -g_m \Delta E(1 - x) \tag{4.45}$$

Since

$$\Delta i_{b2} = -\frac{\Delta E}{R_k}$$

we have now

$$(1 - x)R_k = \frac{1}{g_m} \tag{4.46}$$

as the condition to be fulfilled to stabilize the plate current of tube 1 against emission changes.

Another method, shown in Fig. 4.22, uses two triodes, also in one envelope, in a series connection. If the two tubes are identical, $E_o$ is exactly midway between ground and the plate supply potential when the input-signal voltage is zero. If identical emission changes occur in the two tubes, $E_o$ is still symmetrically located with respect to these two levels and is therefore unaffected by the emission changes.

Emission changes caused by changes in the heater voltage are best prevented by controlling the heater-voltage source. Where the heater

current required is not excessive, it may be possible to supply the heaters in series from the regulated d-c power source.

Balanced amplifiers can also be used to good advantage in the early stages of computer amplifiers, since they provide a good degree of protection against variations in cathode emission or in plate-supply potential. A typical circuit arrangement is shown in Fig. 4.23. The balanced output can be applied to another balanced amplifier or either output voltage may be used alone, depending on whether a phase inversion is desired. A variation of this circuit is shown in Fig. 4.24. This version is a useful way of inserting the output of the drift-stabilizing amplifier. In effect, the drift stabilizer drives a cathode follower, the output of which is applied to the cathode of the triode amplifier.

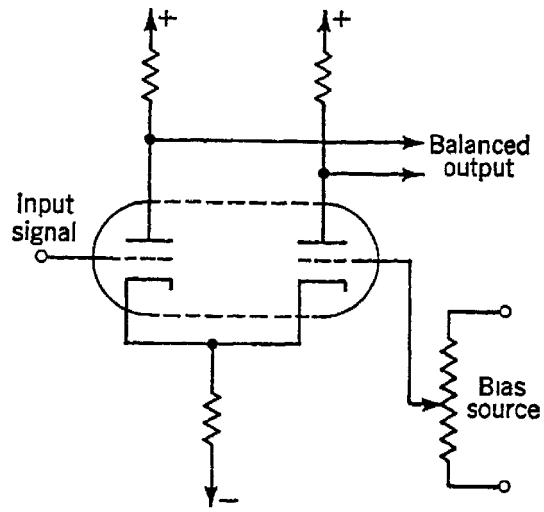



Fig. 4.23 Differential amplifier

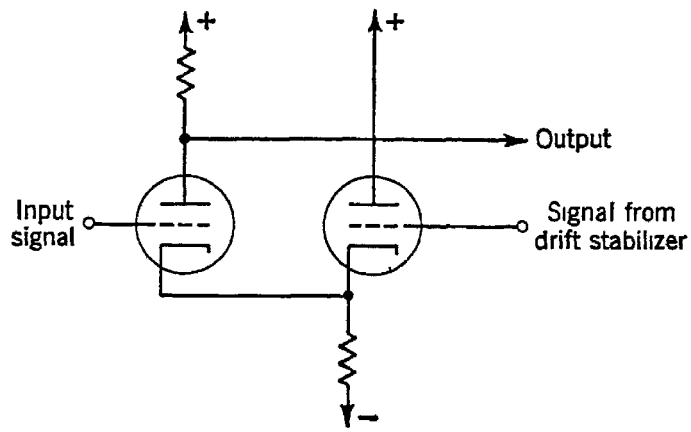



Fig. 4.24 Differential amplifier with one input applied through a cathode follower

## 4.7 Stabilization of feedback amplifiers[16–19]

Like any high-gain amplifier to which feedback is applied, the d-c computer amplifier requires great care in the design of its behavior at high frequencies to prevent it from becoming unstable and breaking into oscillation. The amplitude and phase responses must be controlled at frequencies far above the range of useful amplification, and lack of awareness of this necessity can lead to designs that exhibit unexpected instabilities.

Since our attention will be focused upon the transfer function, the ratio of output voltage to input voltage, we start by considering the nature of

this function. To determine the function for a given network, we consider the network to be driven only at its input terminals and solve for the output voltage. A set of mesh equations would be written as follows:

$$\begin{aligned} E_1 &= Z_{11}I_1 + Z_{12}I_2 + \cdots + Z_{1n}I_n \\ 0 &= Z_{21}I_1 + Z_{22}I_2 + \cdots + Z_{2n}I_n \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ 0 &= Z_{n1}I_1 + Z_{n2}I_2 + \cdots + Z_{nn}I_n \end{aligned} \tag{4.47}$$

where $Z_{ij} = sL_{ij} + R_{ij} + \frac{1}{sC_{ij}}$. If $i = j$, then $Z_{ij}$ is the total self-impedance of the $i$th loop, and if $i \neq j$, then $Z_{ij}$ is the mutual impedance of the $i$th and $j$th loops. In matrix notation we might write

$$[E] = [Z][I] \tag{4.48}$$

where $[E]$ and $[I]$ are column matrices and $[Z]$ is the square matrix of impedance operators

$$[Z] = \begin{bmatrix} Z_{11} & Z_{12} & \cdots & Z_{1n} \\ Z_{21} & Z_{22} & \cdots & Z_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ Z_{n1} & Z_{n2} & \cdots & Z_{nn} \end{bmatrix} \tag{4.49}$$

For the special case in which the network consists only of bilateral elements, $[Z]$ is a symmetrical matrix; that is, $Z_{ij} = Z_{ji}$. However, for the general case in which nonbilateral elements, such as tubes or transistors, are employed, $Z_{ij} \neq Z_{ji}$.

Although the operator $s$ in these expressions may be regarded in the limited sense as being merely a short-hand notation for the differentiation operator $\frac{d}{dt}$, it is useful to regard it in the much more general sense of being a complex frequency $s = \sigma + j\omega$. For the special case in which $\sigma$ is taken to be zero, $s$ becomes simply $j\omega$, and the generalized network equations (4.47) reduce to the steady-state equations describing the circuit when it is excited by a sinusoidal driving source.

With the network equations written in terms of $s$, it is evident that in a network of linear elements the transfer function will be a ratio of polynomials in $s$. This is readily demonstrated, for, if $I_k$ is the current in the output element and if $Z_0$ is the impedance of that element, then

$$I_k = \frac{E_1 \det |Z_{1k}|}{\det |Z|} \tag{4.50}$$

and

$$G(s) = \frac{E_o}{E_1} = \text{transfer ratio} = \frac{Z_0 \det |Z_{1k}|}{\det |Z|} \tag{4.51}$$

Here $\det |Z|$ is the determinant formed from the impedance matrix $[Z]$,

and det $|Z_{1k}|$ is the determinant formed by deleting the first row and $k$th column of det $|Z|$. Both numerator and denominator of Eq. (4.51) are polynomials in $s$, since they are obtained by multiplying together factors of the form $Z_{ij} = sL_{ij} + R_{ij} + 1/sC_{ij}$. Then we may write

$$G(s) = \frac{A_0 s^m + A_1 s^{m-1} + \cdots + A_{m-1}s + A_m}{B_0 s^n + B_1 s^{n-1} + \cdots + B_{n-1}s + B_n} \tag{4.52}$$

Both polynomials may be written in factored form to give

$$G(s) = \frac{A_0}{B_0}\frac{(s - s_1)(s - s_3)(s - s_5)\cdots}{(s - s_2)(s - s_4)(s - s_6)\cdots} \tag{4.53}$$

Now, if the complex variable $s$ assumes any of the values $s_1, s_3, s_5, \ldots$ that appear in the numerator, then $G(s)$ becomes zero. Hence these values of $s$ are called "zeros" of the transfer function $G(s)$. Similarly, if $s$ assumes any of the values $s_2, s_4, s_6, \ldots$ that appear in the denominator, $G(s)$ becomes infinite. Such points are called "poles" of $G(s)$.

In the most general case the values $s_j$ in Eq. (4.53) are complex numbers, and $s$ also may take on complex values. Therefore $G(s)$ is itself a complex quantity and may be described by a magnitude and an angle. In the study of feedback networks there is use for a polar plot of $G(s)$ corresponding to a variation of $s$ from $s = -j\infty$ along the axis of imaginaries (i.e., the real-frequency axis) to $s = +j\infty$. This polar plot is equivalent to a mapping process that maps the real-frequency axis of the $s$ plane into a closed curve in the $G(s)$ plane. If we denote the real and imaginary parts of $s_j$ by $\sigma_j$ and $\omega_j$, respectively, then we may write

$$\frac{|G(s)|}{A_0/B_0} = \left\{\frac{[\sigma_1^2 + (\omega - \omega_1)^2][\sigma_3^2 + (\omega - \omega_3)^2]\cdots}{[\sigma_2^2 + (\omega - \omega_2)^2][\sigma_4^2 + (\omega - \omega_4)^2]\cdots}\right\}^{1/2} \tag{4.54}$$

and

$$\arg G(s) = \left(\tan^{-1}\frac{\omega_1 - \omega}{\sigma_1} + \tan^{-1}\frac{\omega_3 - \omega}{\sigma_3} + \cdots\right) - \left(\tan^{-1}\frac{\omega_2 - \omega}{\sigma_2} + \tan^{-1}\frac{\omega_4 - \omega}{\sigma_4}\cdots\right) \tag{4.55}$$

Thus it can be seen that at any frequency $\omega$, $|G(s)|$ is proportional to the product of the distances from each of the zeros of $G(s)$ to the point $\omega$ on the $j\omega$ axis and inversely proportional to the product of the distances from the poles. Similarly, the angle of $G(s)$ for any $\omega$ is the sum of the angles from the zeros to the point $\omega$ on the $j\omega$ axis minus the sum of the angles from the poles to that point.

As a simple example, consider the network of Fig. 4.25$a$. $G(s)$ is easily found to be

$$\frac{s + 1/R_1C}{s + 1/[R_1R_2C/(R_1 + R_2)]}$$

and therefore has a zero at $s = -1/R_1C$ and a pole at

$$s = -\frac{1}{[R_1R_2/(R_1 + R_2)]C}$$

as shown in Fig. 4.25*b*. To obtain the plot of $G(s)$, we allow $s$ to move along the $j\omega$ axis from $\omega = -\infty$ to $\omega = +\infty$. The circular plot of Fig. 4.25*c* is the result, as we can show by carrying out the processes of

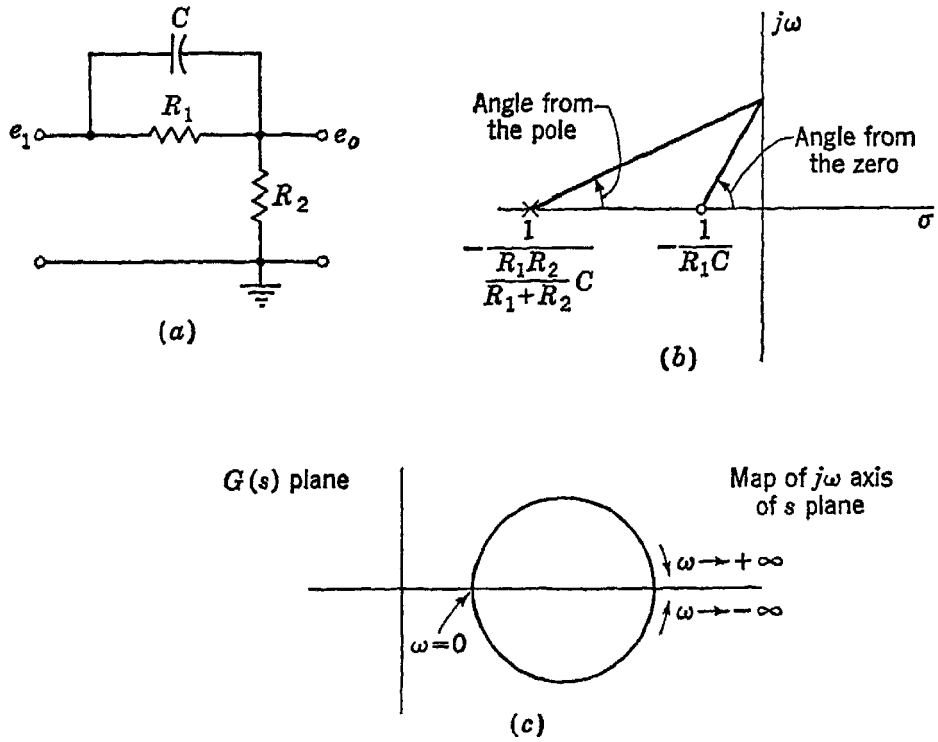



Fig. 4.25 A network, its pole and zero locations, and an $s$-plane plot of its transfer function

Eqs. (4.54) and (4.55). The intercepts on the axis of reals are $+1$ at $\omega = \pm\infty$ and $+R_2/(R_1 + R_2)$ at $\omega = 0$. Thus the circle diameter is $R_1/(R_1 + R_2)$.

It is of interest to note that, if a pole is near the $j\omega$ axis, the gain will be very large at frequencies near the pole, and the phase shift will be changing rapidly. A zero near the axis causes small gain at frequencies near it and also produces rapid phase shift but in the opposite sense. It is thus possible to draw good qualitative conclusions about the behavior of $G(s)$ at real frequencies by examining the positions of its poles and zeros in the $s$ plane.

Besides providing information about the gain (or attenuation) and phase-shift characteristics, the pole and zero locations of the transfer function also enable us to determine the stability of a network. A well-known network theorem states that a network is stable if its transfer function has no poles in the right half of the $s$ plane. As we have already

seen, the transfer function of a feedback network in which the forward gain is $A$ and the fraction of output fed back is $\beta$ is given by

$$G(s) = \frac{A}{1 - A\beta} \tag{4.56}$$

Poles of this function can occur only at poles of $A$ or at zeros of $1 - A\beta$. If we assume that the amplifier $A$ is stable in the absence of feedback, then $A$ has no poles in the right half plane. Any poles of $G(s)$ are therefore due to zeros of $1 - A\beta$. From Eq. (4.56) it is evident that, if $A\beta = 1$, $G(s)$ becomes infinite; even with zero input an output is obtained, since the system is unstable and breaks into oscillation. Thus, if a plot of $A\beta$ passes through the point (1,0) in the $G(s)$ plane, the system is

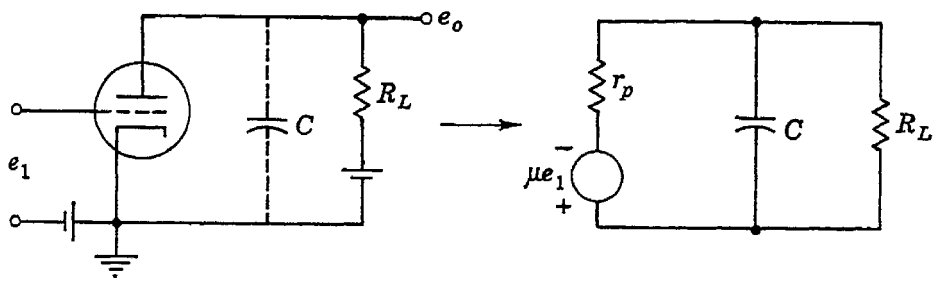



Fig. 4.26 A simple amplifier

unstable. Actually, a more general condition than this is established by the Nyquist criterion, which states that if a polar plot of $A\beta$ merely encloses the point (1,0) without passing through it, the system is unstable. This follows from a theorem from complex-variable theory to the effect that, if the $j\omega$ axis from $-\infty$ to $+\infty$ is mapped into another complex plane by a mapping function, such as $1 - A\beta$, the number of counterclockwise encirclements of the origin made by the map is equal to the number of poles minus the number of zeros that the mapping function has in the right half of the $s$ plane. Now if $\beta$ is a passive network, as it often is in practice, then $\beta$ is stable and therefore has no poles in the right half plane. In fact, even if $\beta$ were an active network, it would probably have been designed to be stable. Thus in most practical circuits neither $A$ nor $1 - A\beta$ can have poles in the right half plane. Any encirclements of the origin made by the $1 - A\beta$ map of the $j\omega$ axis, therefore, indicate zeros of $1 - A\beta$, i.e., poles of $A/(1 - A\beta)$, and the feedback system is then unstable. It is often simpler to map $A\beta$; then instability is indicated by any encirclements of the point (1,0).

Some examples may serve to clarify the application of the Nyquist criterion. Consider first the amplifier stage of Fig. 4.26. The transfer function for this amplifier is

$$G(s) = -\frac{\mu/r_pC}{s + 1/[r_pR_L/(r_p + R_L)]C} \tag{4.57}$$

The network has a real pole (i.e., a pole on the real axis) at

$$s = -\frac{1}{[r_p R_L/(r_p + R_L)]C}$$

and has no zeros. Thus the transformation of the $j\omega$ axis by the mapping function of Eq. (4.57) yields the circular map in the $G(s)$ plane shown in Fig. 4.27.

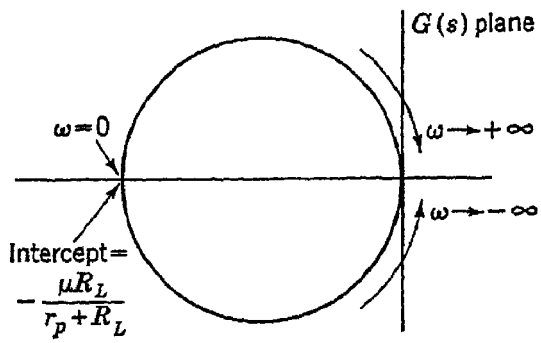



Fig. 4.27 $G(s)$ map of $j\omega$ axis for the circuit of Fig. 4.26

Now let us modify the circuit of Fig. 4.26 by tapping off a fraction $\beta$ of its output voltage and feeding this back in series with the input as shown in Fig. 4.28. [We shall suppose the voltage divider $R_1, R_2$ to have so high a resistance that its effect upon $G(s)$ as given by Eq. (4.57) may be neglected.] The product of $\beta$ and the forward-gain expression of Eq. (4.57) yields a plot that is the same as that of Fig. 4.27 but reduced in magnitude as shown in Fig. 4.29. Since the polar plot

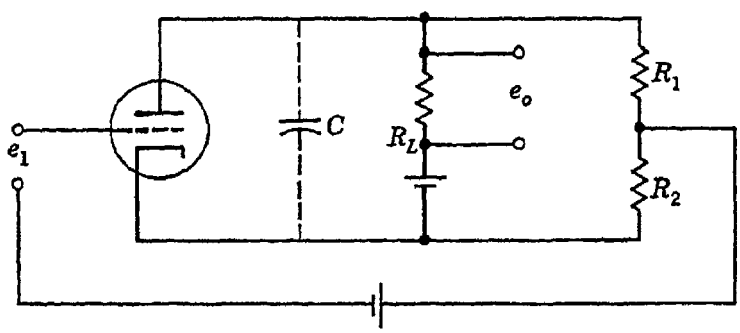



Fig. 4.28 An amplifier with feedback

does not encircle the critical point (1,0), the amplifier with feedback is stable.

The transfer function of two cascaded stages like that of Fig. 4.26 can be found by considering a pole of order 2 to exist at

$$s = -\frac{1}{[r_p R_L/(r_p + R_L)]C}$$

and proceeding as before to evaluate Eqs. (4.54) and (4.55) or, alternatively, by simply multiplying by itself the vector whose tip traces out $G(s)$ in Fig. 4.27. In either case the map in Fig. 4.30 is obtained. If we now place resistive feedback around the two stages in the same way as in Fig. 4.28, then the resulting plot will be that of Fig. 4.30 with all radii diminished in the ratio $R_2/(R_1 + R_2)$. Now, unless $R_2/(R_1 + R_2)$ is

very small, so that $[R_2/(R_1 + R_2)][\mu R_L/(r_p + R_L)]^2 < 1$, the circuit will be unstable, since the polar plot will encircle the point (1,0). Physically, this means that since we have two stages of amplification, the net phase shift near $\omega = 0$ is zero, and only a very small amount of feedback can be tolerated without making the amplifier unstable.

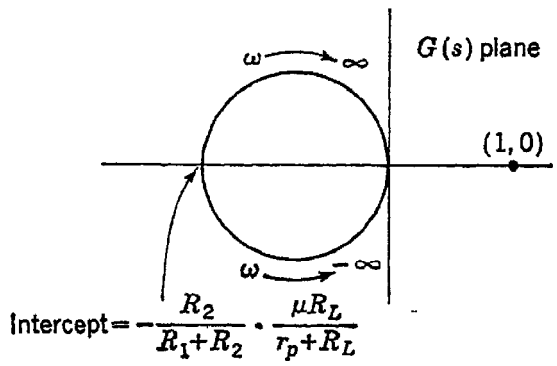



Fig. 4.29 $G(s)$ map of $j\omega$ axis for the circuit of Fig. 4.28

One interesting consequence of the Nyquist criterion is that it shows that it is possible for a network to be "conditionally" stable. One often encounters the oversimplified statement that the output should not be allowed to be in phase with the input at any frequencies for which the gain exceeds unity. For computer amplifiers this is sometimes stated by saying that the phase of the output must not change by as much as 180° before the gain has dropped below unity. Although this statement is substantially correct for the simple feedback paths and circuit configurations usually found in computer amplifiers, it

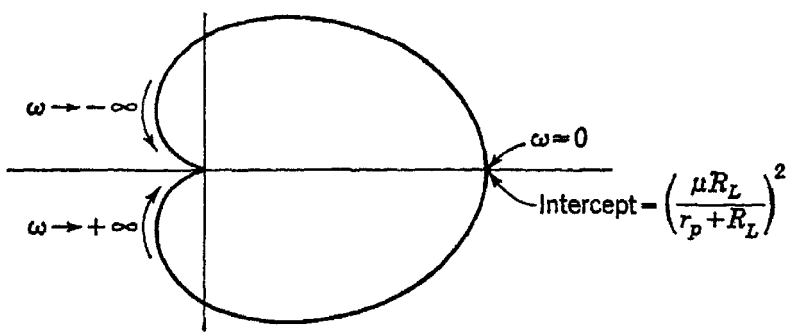



Fig. 4.30 $G(s)$ map of $j\omega$ axis for two cascaded stages

is by no means generally applicable. The $A\beta$ plot in Fig. 4.31 is an example. As drawn, the plot does not encircle the critical point. (Let a radius vector from the critical point follow the curve; it will be seen that this vector does not make a net rotation in going along the full curve.) Thus, although there are regions of the plot for which the gain exceeds unity when output and input are in phase, the Nyquist criterion indicates that the system is stable. However, this stability is conditional, in the sense that a radial shrinkage of the whole plot, caused perhaps by tube aging or by low transconductance during the initial warm-up when power is applied, would make the curve encircle the critical point and therefore become unstable. Conditional stability is usually regarded as undesirable

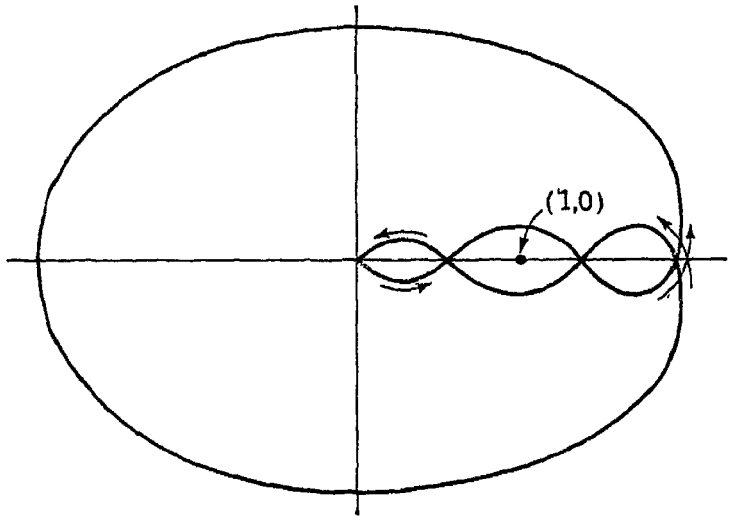



Fig. 4.31 Nyquist plot of a conditionally stable feedback system

because of the special care that must be taken to ensure that the stability is maintained.

Let us consider now the Nyquist plot for three identical stages of d-c amplification with resistive feedback. Such a plot is shown in Fig. 4.32. Although as it is drawn the plot does not enclose the point (1,0), it would evidently do so if either $A$ or $\beta$ or both were sufficiently large. In fact it is easy to see that three identical stages do not constitute a suitable

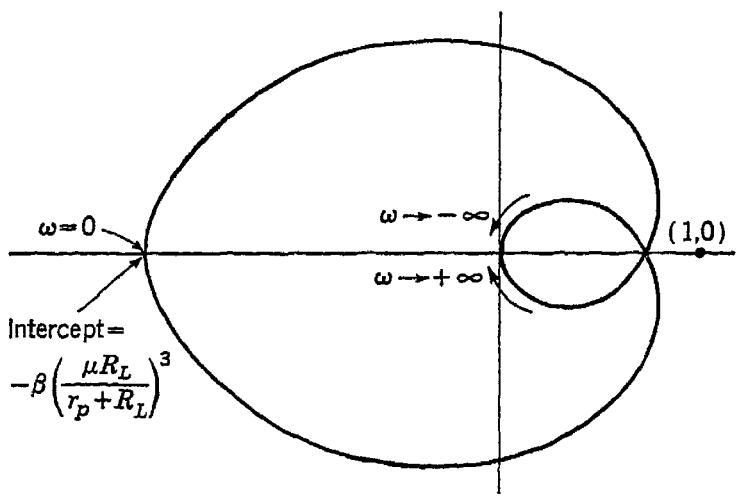



Fig. 4.32 Nyquist plot for a three-stage d-c amplifier with resistive feedback

computing amplifier, since, for any except small feedback ratios $\beta$, the system would be unstable. Since the drop-off of gain with frequency is due only to the stray shunting capacitance $C$ in each stage (Fig. 4.26), it is apparent that, at the frequency at which each stage is producing a 60° phase shift from the zero-frequency phase angle, the gain of each stage is half its zero-frequency value. Thus the over-all amplifier gain is $A/8$ when the over-all phase shift is 180°, where $A$ is the d-c

gain of the amplifier. For practical values of $A$ ($10^4$ to $10^6$), $\beta$ must therefore be very small if $A\beta/8$ is to be less than unity in magnitude.

It is, therefore, necessary to modify the amplifier circuit in such a way that the high-frequency portion of the Nyquist plot will not enclose the critical point even if $A$ is large and $\beta = 1$. In other words, we must shape the high-frequency response of the amplifier in such a way as to ensure stability. One way of doing this is to concentrate the drop-off in gain in either one or two stages instead of distributing it equally among all three. With only one $RC$ network causing the drop-off, zero gain

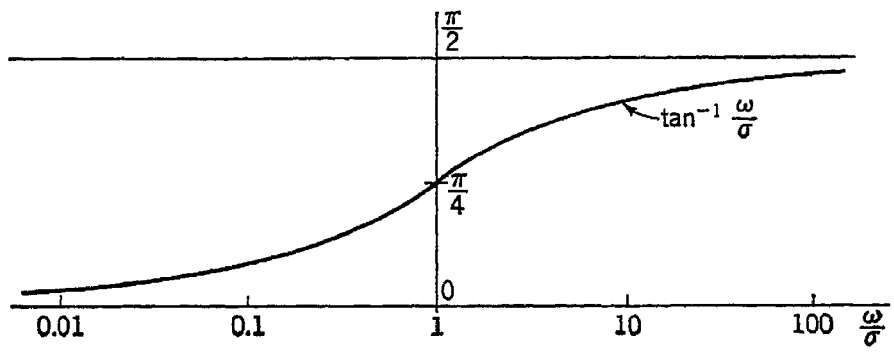



Fig. 4.33 $\tan^{-1}\omega/\sigma$ on a logarithmic frequency scale

would be reached at 90° phase shift, but with two, although the phase shift from the low-frequency value could become 180°, the gain would then be zero. In either case the $A\beta$ plot cannot enclose the critical point, and the amplifier is stable. In order to produce the desired phase shift, an interstage lead network is often used to compensate over a portion of the frequency spectrum for the lag caused by the stray shunting capacitances. The network of Fig. 4.25 is an example. From its $G(s)$ plot it can be seen that the phase shift advances from zero through a maximum lead and then back to zero as $\omega$ goes from zero to $+\infty$. The magnitude of the phase shift can be calculated by applying Eq. (4.55) to the expression for $G(s)$ for the network. Thus the lead angle $\theta$ is given by

$$\theta = \tan^{-1}(\omega R_1 C) - \tan^{-1}\left(\omega \frac{R_1 R_2}{R_1 + R_2} C\right) \tag{4.58}$$

or, in somewhat more general terms, by

$$\theta = \tan^{-1}\frac{\omega}{-\sigma_0} - \tan^{-1}\frac{\omega}{-\sigma_p} \tag{4.59}$$

where $\sigma_0$ is the position of the (real) zero and $\sigma_p$ the position of the (real) pole.

On a logarithmic scale of frequency the inverse tangent curve appears as in Fig. 4.33. Equation (4.59) indicates that $\theta$ can be plotted as the

difference of two of these curves, one having $\sigma = 1/R_1C$ and the other having $\sigma = 1/[R_1R_2/(R_1 + R_2)]C$. Such a curve is shown in Fig. 4.34.

Evidently, if the "center frequencies" $\sigma_0$ and $\sigma_p$ are made close to each other, the maximum lead angle is small, but if they are widely separated,

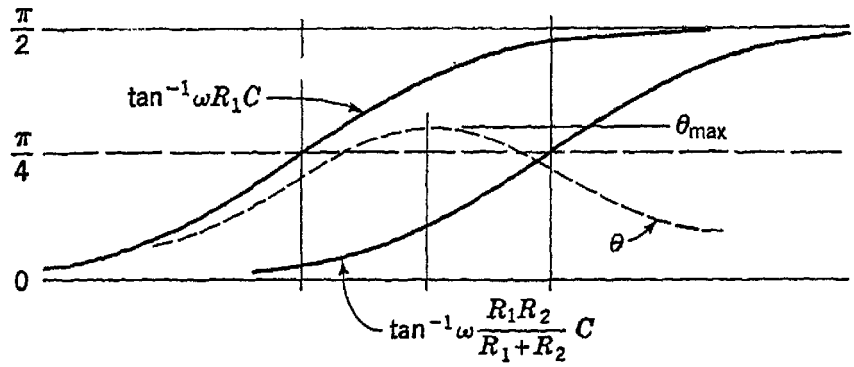



Fig. 4.34 Phase-shift curve for the lead network of Fig. 4.25

the maximum lead angle will approach 90°. The frequency $\omega_m$, of maximum lead, is the geometric mean of the center frequencies; that is, $\omega_m = \sqrt{\sigma_0\sigma_p}$. It follows that the maximum angle of lead $\theta_m$ is

$$\theta_m = \tan^{-1}\sqrt{\frac{\sigma_p}{\sigma_0}} - \tan^{-1}\sqrt{\frac{\sigma_0}{\sigma_p}} \tag{4.60}$$

$$= 2\tan^{-1}\sqrt{\frac{\sigma_p}{\sigma_0}} - \frac{\pi}{2} \tag{4.61}$$

For the network of Fig. 4.25 we obtain

$$\omega_m = \frac{1}{R_1C}\sqrt{\frac{R_1 + R_2}{R_2}} \tag{4.62}$$

and

$$\theta_m = 2\tan^{-1}\sqrt{\frac{R_1 + R_2}{R_2}} - \frac{\pi}{2} \tag{4.63}$$

One obvious way to make use of a lead network is shown in Fig. 4.35. Here the network of Fig. 4.25 has been used as the interstage coupling. Note that this network also provides a means of adjusting the grid-bias voltage of the second tube by the voltage-divider action of $R_1$ and $R_2$. (The alternative view may also be taken that, since $R_1$ and $R_2$ are ordinarily present to form a d-c voltage divider for bias setting, a lead network incorporating them can be formed merely by adding capacitor $C$.)

A somewhat subtler approach may be taken by recognizing that each pole in the transfer function produces a phase lag and each zero a phase lead. Thus lag due to poles may be canceled by introducing zeros into the transfer function. If additional poles are introduced at the same time, they should lie to the left of the zeros by the proper amount to furnish the desired maximum lead angle. The synthesis of $RC$ transfer functions has received much attention in the periodical literature in the

last few years and will not be treated here, with the exception of the network shown in Fig. 4.36. If $C_1 \gg C_s$ and if $R_e = r_pR_L/(r_p + R_L) \gg R_1$, then, as the reader may show, the transfer characteristic will have poles at approximately $s = -1/R_eC_1$ and $s = -1/R_1C_s$ and a zero at approximately $s = -1/R_1C_1$. In the absence of the compensating network $R_1C_1$, only a single pole is present, located at $s = -1/R_eC_s$. The zero and the left-hand pole together produce a phase lead like that of Fig. 4.25, and the right-hand pole contributes a lag. The over-all phase shift might be as shown in Fig. 4.37. Although the phase of the output starts to lag at lower frequencies than it does without the compensating network $R_1C_1$, the lag at high frequencies is not so great as it is without this branch. At high frequencies $C_1$ may be considered a virtual short circuit. $C_s$ is thus effectively shunted by the low resistance $R_2$, and its phase-shifting effect is relegated to much higher frequencies than without $R_1C_1$.

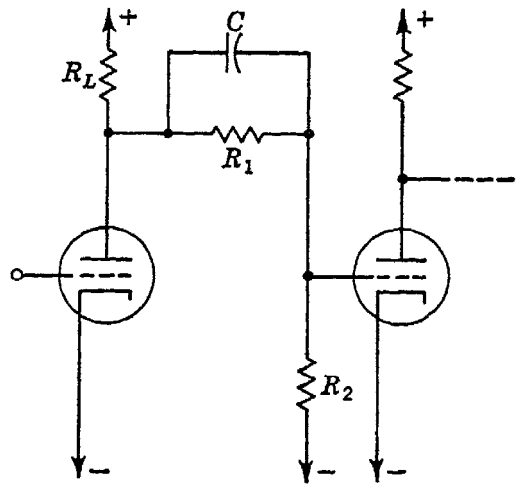



Fig. 4.35 Incorporation of a lead network in the interstage coupling

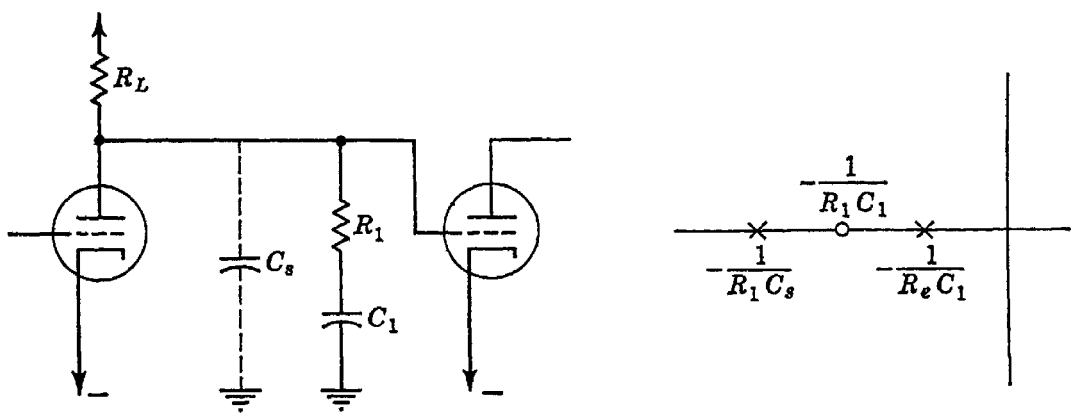



Fig. 4.36 An interstage network and its pole-zero pattern

In order to put the detailed considerations of the foregoing pages into a somewhat more down-to-earth perspective, let us consider the problems involved in the design of a high-performance computer amplifier. Let us suppose that we desire an open-loop gain of 120 db with no more than 3-db drop-off at 10 cycles. If only one time constant is responsible for the drop in gain, the curve must ultimately fall at a rate asymptotic to a −6-db/octave slope (−20 db/decade). At an angular frequency $\omega_1 = 1/T_1$, where $T_1$ is the controlling time constant, the sloping asymptote intersects the extended horizontal part of the gain curve at a point

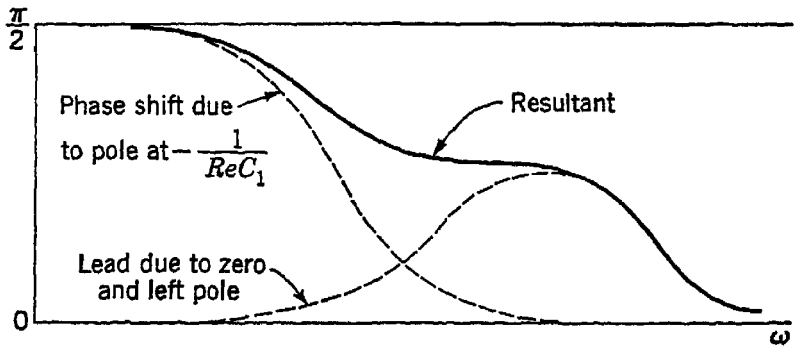



Fig. 4.37 Phase shift for the network of Fig. 4.36

3 db above the actual curve, as shown in Fig. 4.38.* Each such drop-off at −6 db/octave can be drawn in asymptotically, the break points occurring for each at $\omega_k = 1/T_k$ and the actual curve being 3 db below the asymptotes at their intersection. To each such gain asymptote there corresponds a 90°-phase asymptote, so that, to a fair degree of approximation, a curve that is falling at −6 db/octave has a 90° phase shift, one falling at −12 db/octave a 180° phase shift, and so on.† If the response of our hypothetical amplifier drops at only −6 db/octave, it will not fall to 0 db until 10 mc, a bandwidth that is extremely difficult to attain. If we allow the drop-off to be at −12 db/octave, we encounter a phase shift of 180° while the amplifier gain is still above 0 db, and the system becomes unstable. One solution is to reduce the low-frequency gain, so that −6-db/octave drop-off to zero can be achieved in a reasonable bandwidth. Each such decrease of 6 db reduces the bandwidth by 1 octave. Another solution is to reduce the region of uniform gain at low frequencies. Thus, reducing the 3-db down point from 10 cycles to 1 cycle reduces the zero-db gain point from 10 mc to 1 mc. Both these solutions are typical of low-cost, low-performance computers, but if the maximum gain is to be achieved over as wide a range as possible, more elegant measures must be employed. To keep the gain flat (to within 3 db) out to 10

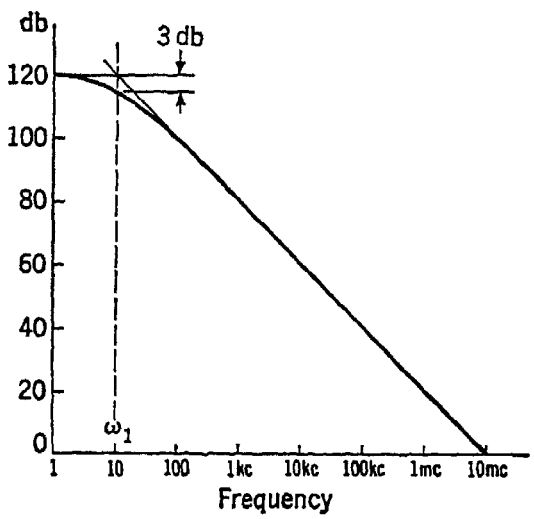



Fig. 4.38 A hypothetical gain versus frequency curve

* See Truxal,[16] pp. 344–360, for a good discussion of the method of asymptotes.

† This approximation is often good enough, but it can be carelessly used. For a more precise statement of the relations between attenuation and phase, see Bode,[16] pp. 303–336.

cycles and at the same time to keep the frequency of 0-db gain substantially lower than 10 mc, the curve must drop with an average slope of more than −6 db/octave, but the phase shift must not reach 180° until the gain is below 0 db. Since one time constant will not result in steep enough slope, two would be used in this example. Since two can give 180° phase shift, one or more phase-lead networks would be incorporated. The design is carried out to leave reasonable gain and phase margins. Gain margin is the amount by which the gain would have to be increased at the frequency of 180° phase shift to produce instability, and phase margin is the amount by which the phase lag would have to be increased at 0-db gain to produce instability. Margins of 10 db for the gain and 30° for the phase shift are typical.

An interesting approach to the problem of optimizing gain and bandwidth and simultaneously ensuring stability has been reported by Deering,[20] who has provided a path from the amplifier input that shunts the first two stages and goes directly to the third through capacitive coupling. At frequencies at which the phase shift of the first two stages is large enough to endanger stability, the shunt path predominates. This allows substantial improvement of the bandwidth, since no artificial drop-offs need be incorporated.

Two examples exhibiting many of the design features we have discussed are shown in Figs. 4.39 and 4.40. These circuits were developed under the direction of Prof. R. M. Howe at the University of Michigan. Figure 4.39 shows a drift-stabilized amplifier, the upper portion being the d-c amplifier and the lower portion the auxiliary drift-free amplifier. The d-c amplifier uses in its first stage a 5691 connected as a cathode-coupled differential amplifier. The signal input is applied to one section through a 500-kilohm grid-current-limiting resistor, and the output of the auxiliary amplifier is applied to the other grid. Preliminary manual balancing is done by adjusting the rheostats in the plate circuit of the right section of the first 5691 tube with the auxiliary amplifier disabled. The output of this stage is taken from the left plate and is applied through a combination voltage-divider and lead network to the left half of a second 5691. Approximately 30 per cent of the first-stage output signal is lost in the voltage divider. Another lead network (this one consisting of 50 $\mu\mu$f, paralleled by 4 megohms, the combination being in series with 2 megohms) connects the output of the left section of the second 5691 to the grid of the right section. A high-frequency degenerative feedback path from plate to grid of the left section of this tube is also incorporated to help shape the high-frequency gain and phase shift of this stage. The shaping action is similar in principle to that performed by the circuit of Fig. 4.36. However, it can be shown that, when the $R_1C_1$ circuit is connected back to the grid as in Fig. 4.39 instead of being grounded as in

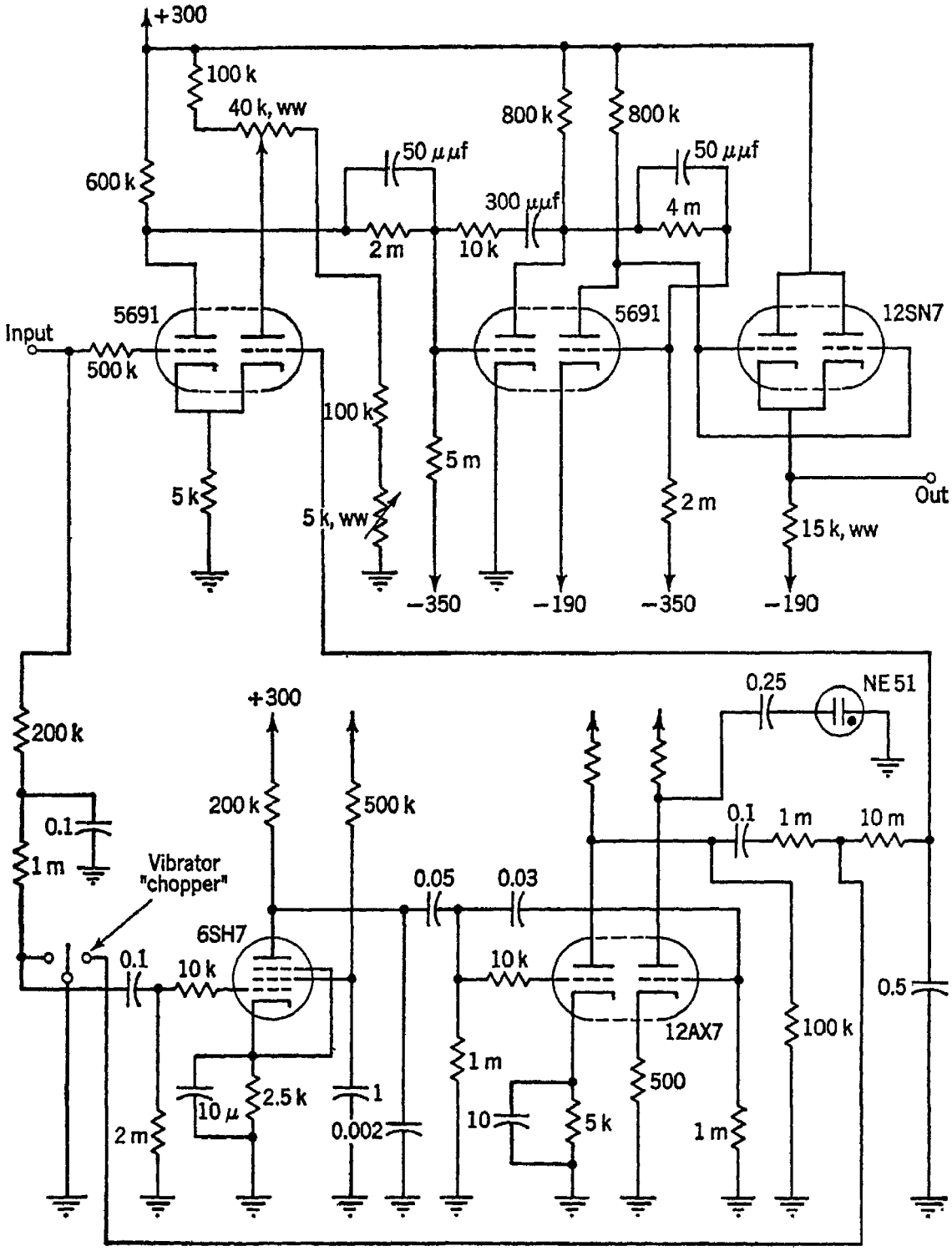



Fig. 4.39 A chopper-stabilized computer amplifier

Fig. 4.36, the same shaping effect can be achieved with a much smaller capacitance, and there is less loading of the amplifier. The output of the right section is applied to a 12SN7, with the two sections parallel-connected as a cathode follower. Note that from input terminal to output terminal the amplifier has three net phase reversals.

The auxiliary-amplifier circuit uses a pentode followed by one section of a 12AX7. This amplifier is of course capacitively coupled throughout. *RC* filters at both the input and the output are included to reduce chopper-frequency components to a minimum. The second section of the 12AX7 is not part of the amplifier but is used to drive a neon lamp that is an overload indicator. During normal operation of the system the drift voltage that is "chopped" and amplified is of small magnitude. If for any reason the output of the d-c amplifier becomes unable to follow changes in its input, the feedback loop is in effect interrupted and the summing node is no longer held at ground potential. A large voltage is produced at the output of the auxiliary amplifier and at the output of the right section of the 12AX7. This voltage, which is at the chopper frequency, is capacitively coupled to a neon indicator. The circuit is adjusted so that the neon light glows whenever the summing-node potential exceeds a magnitude of a few millivolts with respect to ground. This type of overload indicator is a standard device on analog computers that use drift-stabilizing amplifiers. Amplifiers that do not use drift stabilization often have a neon indicator shunted across a fraction of the output voltage so that, when the output magnitude reaches 100 volts, the indicator lights up. However, since a heavily loaded amplifier may saturate below 100 volts, such an indicator would not be of any use. The circuit in Fig. 4.39 detects saturation at any level, whether due to overloading of the amplifier or to component deterioration.

When amplifier saturation does occur, the filter capacitors receive a much larger charge than during normal operation. After correction of the overload condition and before resumption of the computation, sufficient time must be allowed to restore these capacitors to their normal condition. Time constants of several seconds are common in the filters, and dielectric absorption effects may add substantial time delay. As much as a minute is sometimes necessary for the filter capacitors to return to their normal conditions.

The second circuit, shown in Fig. 4.40, is a somewhat higher-performance circuit, which was designed for low drift so that drift stabilization would be unnecessary. This circuit also uses a cathode-coupled differential amplifier followed by two triode stages and a cathode follower. The 5751 gives an amplification of about 45 from input grid to left plate. Amplifier zero-set is controlled by the rheostats in the plate circuit of the second section of this tube. The output of this stage is coupled through a lead network to the first 12AX7 stage. In addition, a small amount of regeneration is introduced by coupling from the output of the lead network back to the grid of the right section of the differential amplifier. The 100-kilohm resistor and the 2.4-megohm resistor function as a volt-

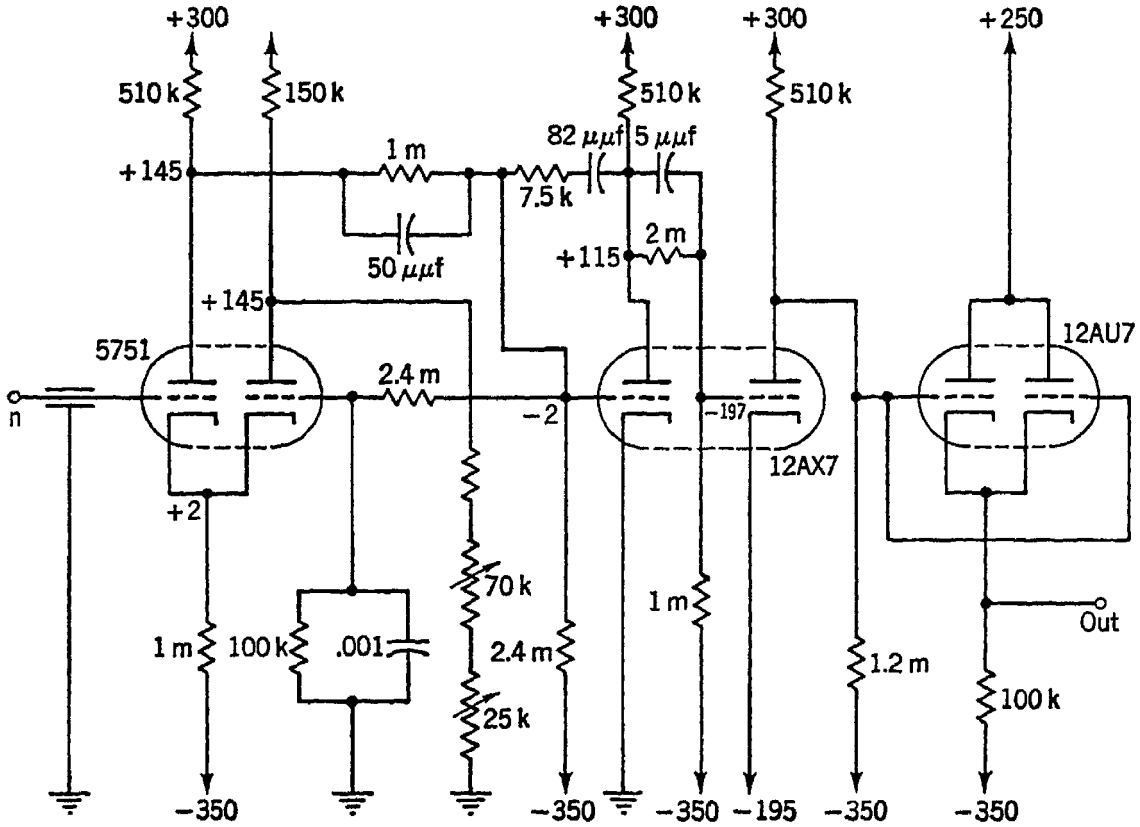



Fig. 4.40 A low-drift unstabilized amplifier

age divider, and the 0.001-μf capacitor decreases the regeneration as the frequency increases. The second stage (left half of the 12AX7) has a gain of about 50. High-frequency degenerative feedback is introduced in this stage by the series resistor and capacitor from plate back to grid. Another voltage divider, with a small amount of lead due to the 5-μμf capacitor, couples this stage to the second half of the 12AX7, the output of which drives a cathode follower. The over-all gain of this amplifier is about $10^5$. Its open-loop gain versus frequency curve is shown in Fig. 4.41.

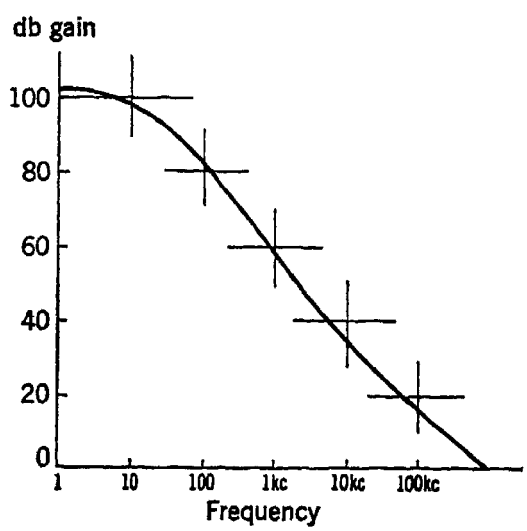



Fig. 4.41 Open-loop gain of the amplifier of Fig. 4.40

In our discussions of stabilization of feedback amplifiers we have tacitly assumed that $\beta$, the feedback fraction, was a real number and that its value could be readily obtained. Neither of these assumptions is valid except in rather simple situations, and it is wise for us to examine our feedback

circuit with some care. The circuit of Fig. 4.42 is our reference. Three equivalent forms in which the transfer ratio may be stated are as follows:

$$\begin{aligned}\frac{e_o}{e_1} &= \frac{KZ_2/(Z_1+Z_2)}{1-[KZ_2/(Z_1+Z_2)](Z_1/Z_2)}\\ &= \frac{Z_2}{Z_1+Z_2}\,\frac{K}{1-KZ_1/(Z_1+Z_2)}\\ &= -\frac{Z_2}{Z_1}\,\frac{K}{K-(Z_1+Z_2)/Z_1}\end{aligned}$$

The first of these is in the form $A/(1-A\beta)$, and we may identify $A$ as $KZ_2/(Z_1+Z_2)$ and $\beta$ as $Z_1/Z_2$. This is a shunt feedback system, with input current and feedback current being applied to the same point. We may consider $\beta$ to be the ratio of current fed back per unit of output

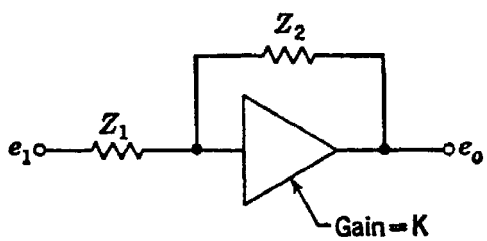



Fig. 4.42 Basic feedback circuit

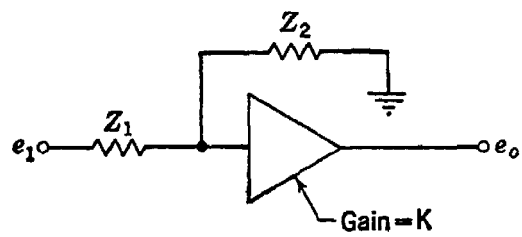



Fig. 4.43 Circuit of Fig. 4.42 modified to determine forward gain

voltage to current input per unit of input voltage. The value $KZ_2/(Z_1+Z_2)$, which is analogous to the forward gain $A$, is equivalent to the gain of the circuit when $Z_2$ is disconnected as a feedback element and grounded as shown in Fig. 4.43.

For a study of the circuit with regard to the Nyquist stability criterion, the three expressions for over-all gain are practically the same. One may plot $KZ_1/(Z_1+Z_2)$ and examine it for encirclement of the point (1,0), or one may plot $K$ and examine it for encirclement of the point $\left(\frac{Z_1+Z_2}{Z_1}, 0\right)$. The usefulness of the first form lies in its close analogy with the familiar $A/(1-A\beta)$.

It is natural to inquire whether the stability of the feedback amplifier is different for the adder and integrator connections. If we plot $K$, then for the adder the critical point has the coordinates $\left(\frac{R_1+R_2}{R_1}, 0\right)$, and for the integrator the coordinates are $\left(1, -j\frac{1}{\omega CR_1}\right)$. In the integrator case, the critical point is somewhat below the real axis and approaches it as the frequency increases. For the high frequencies at which the plot of $K$

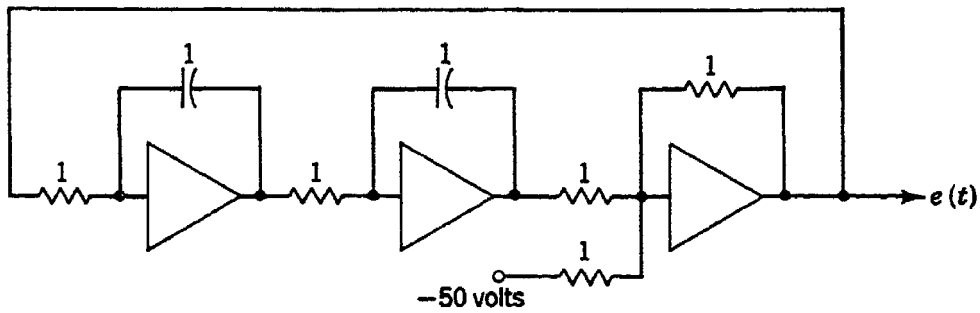
comes near the critical point, the point will usually be found so close to the real axis as to be considered exactly upon it, for all practical purposes. Thus an amplifier stable with resistive feedback may also be expected to be stable when the feedback is capacitive.

## REFERENCES


1. Dow, P. C., An analysis of certain errors in electronic differential analyzers, Ph.D. dissertation, University of Michigan, 1957. Also published in *IRE Trans. on Electronic Computers*, vol. EC-6, no. 4, December, 1957, pp. 255–260; vol. EC-7, no. 1, January, 1958, pp. 17–22.
2. Macnee, A. B., Some limitations on the accuracy of electronic differential analyzers, *Proc. IRE*, vol. 40, no. 3, March, 1952, pp. 303–308.
3. Marsocci, V. A., An error analysis of electronic analog computers, *IRE Trans. on Electronic Computers*, vol. EC-5, no. 4, December, 1956, pp. 207–212.
4. Goldberg, E. A., Stabilization of d-c amplifiers, *RCA Rev.*, vol. 11, no. 2, June, 1950, pp. 296–300.
5. Bradley, F. R., and R. McCoy, Driftless d-c amplifier, *Electronics*, vol. 25, no. 4, April, 1952, pp. 144–148.
6. Buckerfield, P. S. T., The parallel-T d.c. amplifier: A low-drift amplifier with wide frequency response, *Proc. IEE London*, vol. 99, pt. II, 1952, pp. 497–506.
7. Slaughter, D. W., Time-shared amplifier stabilizes computers, *Electronics*, vol. 27, no. 4, April, 1954, pp. 188–190.
8. Harris, E. J., and P. O. Bishop, The design and limitations of d.c. amplifiers, *Electronic Eng.*, vol. 21, 1949, September, pp. 332–335; October, pp. 355–359.
9. Kandiah, K., and D. E. Brown, High-gain d.c. amplifiers, *Proc. IEE London*, vol. 99, pt. II, 1952, pp. 314–326. Discussion, pp. 344–348.
10. Hamer, H., and J. D. Kennedy, Testing of operational amplifiers, *J. ACM*, vol. 2, no. 2, April, 1955, pp. 92–94.
11. Greenwood, Ivan A., Jr., J. Vance Holdam, Jr., and Duncan MacRae, Jr. (eds.), "Electronic Instruments," McGraw-Hill Book Company, Inc., New York, 1948, chaps. 15 and 16.
12. Yarwood, J., and D. H. LeCroisette, D-C amplifiers, methods of amplifying and measuring small direct currents and potentials, *Electronic Eng.*, vol. 26, 1954, pp. 14–19, 64–70, 114–117.
13. Hamer, H., A stabilized driftless analog integrator, *IRE Trans. on Electronic Computers*, vol. EC-3, no. 4, December, 1954, pp. 19–20.
14. Valley, George E., Jr., and Henry Wallman, "Vacuum Tube Amplifiers," McGraw-Hill Book Company, Inc., New York, 1948, chap. 11.
15. Miller, S. E., Sensitive d-c amplifier with a-c operation, *Electronics*, vol. 14, no. 11, November, 1941, pp. 27–31.
16. Bode, H. W., "Network Analysis and Feedback Amplifier Design," D. Van Nostrand Company, Inc., Princeton, N.J., 1945.
17. Thomason, J. G., "Linear feedback analysis," McGraw-Hill Book Company, Inc., New York, and Pergamon Press, Ltd, London, 1955.
18. Stewart, J. L., "Circuit Theory and Design," John Wiley & Sons, Inc., New York, 1956, chap. 12.
19. Truxal, J. G., "Automatic Feedback Control System Synthesis," McGraw-Hill Book Company, Inc., New York, 1955, chaps. 2–6.
20. Deering, C. S., A wide-band direct-coupled operational amplifier, *Proc. 1956 Natl. Simulation Conf.*, pp. 31.1–31.6.

## PROBLEMS

4.1



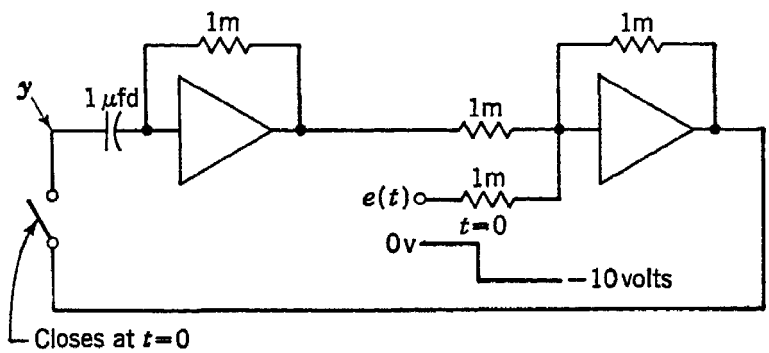

The setup above is frequently used to test the operational amplifiers of a differential analyzer. For a certain set of amplifiers the recorded output is found to be

$$e(t) = K\epsilon^{-t/100} \cos 0.99\, t \qquad \text{volts} \qquad t \geq 0$$

*a.* What would $e(t)$ be if the computer elements were perfect?

*b.* If the error is due entirely to insufficient gain in the amplifiers and if they are identical, what is the open-loop gain of an amplifier?

4.2




With perfect components, this setup would solve the differential equation $\dot{y} - y = -10$. Assume that the adder is perfect and that the differentiator transfer function is

$$\frac{E_o(s)}{E_1(s)} = -\frac{s}{(1 + sT)^2}$$

Find the characteristic equation for the system and show that for small $T$ the roots are approximately $s_1 = 1 + 2T$ and $s_2 = 1/T^2$. (The existence of this large positive root demonstrates the reason that practical differentiators cannot be used to solve equations whose characteristic functions have roots in the right half of the complex plane.) Find the paths followed by the roots as $T$ is allowed to increase without limit. At what value of $T$ does the system become oscillatory?

**4.3** Use the same differentiator to solve the equation $\dot{y} + y = 0$. Find the characteristic equation and determine its roots for small $T$. What is the largest value $T$ can have and still keep the error within 1 per cent after 1 sec computation?

**4.4** Although a chopper-stabilizer amplifier ideally has zero output for zero input, it inevitably produces a small d-c output because of contact potentials and amplification of stray pickup at the vibrator frequency. Let this voltage, referred to the stabilizer input point, be called $\delta$ and let the equivalent drift of $A_2$, referred to its input, be called $\varepsilon$. In terms of $Z_1$, $Z_2$, $A_1$, and $A_2$, determine the output voltage components due to $\delta$ and to $\varepsilon$. If $\varepsilon = 100\delta$, show that very little improvement in the zero-offset voltage is obtained by making $|A_1| > 100$.

CHAPTER 5

# Introduction to the Digital Computer

## 5.1 Quantized and digitized variables

The transition in thinking involved in leaving the discussion of analog computers and taking up digital computers may be more difficult for the reader whose background is in the physical sciences than it is for the reader accustomed to dealing with purely numerical procedures, such as those encountered in business. The term "analog computer" has come

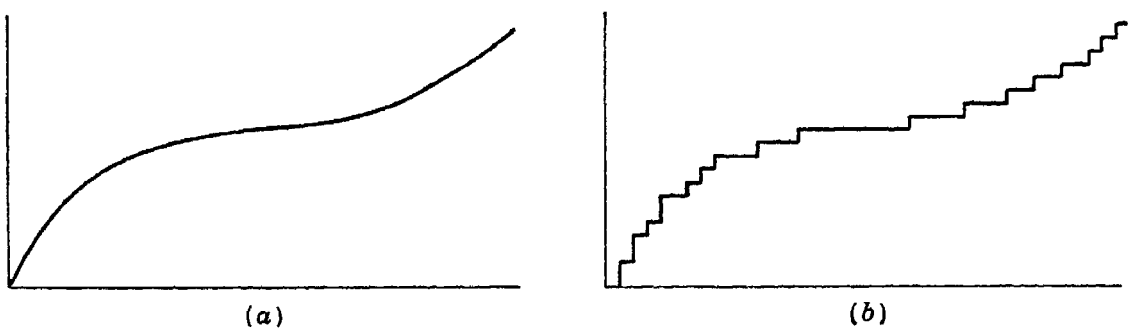



Fig. 5.1 A continuous function, before and after being quantized

to mean the computer which deals with continuous processes. The electronic differential analyzer, mechanizing as it does the familiar differential equations of engineering, is a natural and easily understood tool for the engineer. The digital computer, by contrast, is a discontinuous computer, and, when it solves problems involving continuous functions, it does so by approximating them with discontinuous functions. This is so because the digital computer solves problems by manipulating numbers, and, since the numbers it handles can have at most a finite number of digits, a continuous function cannot be described except by approximation.

This essentially "granular" nature of functional representation in the digital computer requires that any continuous functions be quantized before being digitized. Thus the continuous curve in Fig. 5.1*a* might be quantized as shown in Fig. 5.1*b*.

The quantum employed can be no smaller than the smallest number representable with the digits available but is often taken larger than this. Corresponding to each quantized section of the function is a numerical value, and it is the digits composing these numbers that the machine manipulates in solving a problem in the physical sciences.

Business problems, it may be noted, do not generally deal with truly continuous functions. Such quantities as units of production, prices, hours worked, income taxes, insurance premiums, stock numbers, inventories, and similar concerns of commerce and industry are by their nature quantized, and the digital representation of these quantities is a statement of the number of quanta that compose them.

Since the digital computer is a number-manipulating machine, let us examine some of the methods by which numbers can be represented in computers. In written form numbers are customarily written as a succession of symbols or marks. The sequence

$$a_n\, a_{n-1}\, a_{n-2} \cdots a_2\, a_1\, a_0$$

would be such a succession. Each $a_k$ in the sequence is one of the $A_k$ marks that are allowable in that position in the sequence. Although there are number systems in which the number of marks allowable in the various positions is different (for example, the numerical form for dates, such as 4/25/53, or the British coinage system), the more widely used systems employ the same number of marks, and in fact the same set of marks, for each position in the sequence. In this case, the total possible numbers that $n$ marks can represent is $A^n$, where $A$ is the number of marks allowable in each of the $n$ positions in the sequence. $A$ is called the base or radix of the number system, and the sequence

$$a_n\, a_{n-1} \cdots a_3\, a_2\, a_1\, a_0$$

is interpreted to mean

$$a_nA^n + a_{n-1}A^{n-1} + \cdots + a_3A^3 + a_2A^2 + a_1A^1 + a_0A^0$$

## 5.2 Representation of numbers in the computer

The marks of the written form are replaced by some other representation when the number is placed in a computing machine. Considerable freedom of choice exists in the form and manner of this representation. Thus, 1,000 different numbers (successions of marks) could be represented by 1,000 different levels of voltage above reference on a single wire, by 500 levels on one wire and 2 on another, or by 10 levels on each of three wires. If the time coordinate is used, pulse sequences or pulse durations may be used to represent numbers. Thus numbers might be defined as vectors in a three-dimensional space having coordi-

nates of voltage level, space distribution (the various wires), and time distribution (the pulse sequences). As another example, in desk calculators, the numbers are represented by teeth on gear wheels.

Although in our daily life we use the decimal number system, that is, the system using radix 10, there are many practical reasons for employing in computing machines a number system based on the radix 2. Many physical elements useful for number representation are by their nature binary (two-state) devices. Thus, a switch is either open or closed; a hole in a card is either punched or not punched; a vacuum tube "flip-flop" has either tube 1 on and tube 2 off or 1 off and 2 on. In each case,

**Table 5.1**
**Some decimal-binary equivalences**

| *Decimal* | *Binary* |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 10 |
| 3 | 11 |
| 4 | 100 |
| 5 | 101 |
| 10 | 1010 |
| 16 | 10000 |
| 27 | 11011 |

no other state exists. Many other elements are not intrinsically binary devices but operate with maximum reliability when they are used in a binary mode. Thus, a vacuum tube can carry a wide range of plate currents, and in theory ten values of plate current could be chosen to correspond to the ten decimal digits. However, reduction of the current as the tube ages makes identification of the various levels difficult, and it is better practice to operate even such versatile devices as tubes or transistors at only two levels, i.e., on and off, in order to make more certain the distinction between the allowable states.

Two widely used ways of representing numbers in a binary system of notation are

1. The pure binary system
2. The binary-coded decimal-digit system

In both systems the only marks employed are 0 and 1. In the pure binary system, a sequence $a_n\, a_{n-1} \cdots a_2\, a_1\, a_0$ is interpreted to mean $a_n \times 2^n + a_{n-1} \times 2^{n-1} + \cdots + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$. Some decimal numbers and their binary equivalents are shown in Table 5.1. Arithmetic operations are carried out in much the same way with binary numbers as with decimal numbers. To facilitate the work we might use

the simple addition and multiplication tables of Table 5.2. The following two examples indicate the manner of application of these tables:

$$\begin{array}{r} 1001 \\ +\ 101 \\ \hline 1110 \end{array} \qquad \begin{array}{r} 1010 \\ \times\ \ 101 \\ \hline 1010 \\ 10100\phantom{0} \\ \hline 110010 \end{array}$$

Although binary arithmetic is simpler than decimal arithmetic (compare the sizes of the binary addition and multiplication tables with the

**Table 5.2**
**Binary addition and multiplication tables**

| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 10 |

| × | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

decimal tables), a much longer sequence of binary symbols than of decimal symbols is required to represent a given number. With $n$ decimal digits, numbers up to $10^n - 1$ can be written. To represent this number in binary form we must have $m$ binary digits ("bits"), where $m$ is found from the condition that

$$2^m \geq 10^n \tag{5.1}$$

$$m \log 2 \geq n \tag{5.2}$$

$$m \geq \frac{n}{0.3010} \tag{5.3}$$

Thus the number of binary digits required to represent the largest $n$-digit decimal number is approximately $3\frac{1}{3}$ times $n$ [$m$, as found from Eq. (5.3), must be rounded off to the next higher integer].

To convert a decimal integer to its pure binary form, divide the number repeatedly by 2, writing the remainder digits down separately. The remainder digits will be the binary equivalent, with the last remainder digits being the most significant digits of the result. The following example demonstrates the method:

| | *Decimal form* | *Remainder* |
|---|---|---|
| | 331 | |
| ÷2 | 165 | 1 |
| ÷2 | 82 | 1 |
| ÷2 | 41 | 0 |
| ÷2 | 20 | 1 |
| ÷2 | 10 | 0 |
| ÷2 | 5 | 0 |
| ÷2 | 2 | 1 |
| ÷2 | 1 | 0 |
| ÷2 | 0 | 1 |

The remainder digits, taken from bottom to top, form the binary number 101001011.

In the first step in this method, we note whether the decimal number is odd or even. If it is odd, then the binary equivalent must include $1 \times 2^0$; if it is even, the binary form will include $0 \times 2^0$. We then divide by 2, and determine, according to whether the result is odd or even, whether the binary form should include $1 \times 2^1$ or $0 \times 2^1$. The process is repeated until we have divided out all the 2's in the decimal number.

A similar process can be performed for decimal fractions. Thus at each step we multiply the decimal fraction by 2, writing down the integral part separately. The resulting sequence of digits will be the binary equivalent, with the first digits being the most significant digits of the result. The following example demonstrates the method:

| | *Decimal form* | *Digit* |
|---|---|---|
| | 0.890625 | |
| ×2 | 0.781250 | + 1 |
| ×2 | 0.56250 | + 1 |
| ×2 | 0.1250 | + 1 |
| ×2 | 0.250 | + 0 |
| ×2 | 0.50 | + 0 |
| ×2 | 0.00 | + 1 |

The integer digits, taken from top to bottom, form the binary number 0.111001.

These methods are easily extended to conversion from any radix to any other radix.

Another binary form of representation is the binary-coded decimal-digit form. Here each decimal digit is replaced by the four binary digits that are its equivalent. Thus the binary sequence 011100101001 is equivalent to the decimal number 729. This system has the advantage of being quite close in form to the decimal system we are used to while at the same time employing binary elements. However, it uses its binary elements inefficiently, since four binary elements are capable of representing 16 different numbers, and here they are used in only 10 of these possible configurations. In the example above, 12 bits were needed to represent three decimal digits, but in pure binary form, only 10 bits would be required.

The foregoing introductory comments on binary number systems are intended simply to give the reader an appreciation of the type of manipulations that the machine performs when it executes the programs which we shall study in the next chapter. A more general treatment of number systems will be found in Chap. 7.

## 5.3 The organization of digital computers

Having discussed briefly the manner in which numbers are represented in computers, let us now examine the organization of the structure that manipulates them. Certain functional elements are common to all digital calculations, whether they are done by an electronic calculator or by "hand". A block diagram of these elements and their interrelations is shown in Fig. 5.2.

In a hand calculation, the input device consists of sheets of paper stating the steps to be carried out and giving the initial data for the calculation. The operator, acting as the control over the system, stores these sheets beside his calculator and proceeds with the computation. In executing the computation, he transfers data from the sheets to the calculator, and he also records on paper the numbers that the calculator produces. Some of the numbers so recorded are final results, but others are partial results that must later be reentered on the calculator. As the computation proceeds, the operator observes the progression of steps and makes simple decisions about the steps that should follow or about whether his computation is complete. (These decisions are often so simple and instinctive that the operator may scarcely be aware of them, but, as we shall see in the next chapter, each such decision must be explicitly indicated in the program of instructions for the automatic computer.) The final output results are recorded on paper for whatever use may be desired.

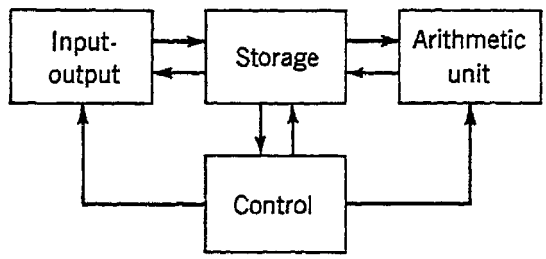



Fig. 5.2 Block diagram of an organization of elements for digital computation

These same functions are performed in the automatic digital computer but at enormously faster speeds, with electromechanical devices for input and output and electronic and magnetic devices for storage, arithmetic, and control. The control unit is analogous to the operator in the hand calculating system and directs the action of the input-output, storage, and arithmetic units in a manner that is dependent upon the information it receives from the storage unit.

To make calculations automatic without the use of impractically complex equipment, it is necessary to restrict operations to the simplest type of arithmetic and to execute complicated calculations by performing a large number of simple operations. Thus the only arithmetical operations performed by most computers are addition, subtraction, multiplication, and division. Some computers do not include division in this

basic list, and it is not even necessary to include multiplication, since both multiplication and division can be effected by doing enough additions or subtractions. Thus the first problem of the "programmer" who is preparing a problem for calculation on a machine is to break the problem down into a succession of simple operations that the machine can perform.

## 5.4 The physical structure of the computer

Before studying the preparation of the problem for the machine, let us define the nature of some of the functional elements of Fig. 5.2. The storage device is ordinarily a hierarchy of devices, each capable of storing a quantity of numbers, which are usually of the same length. In a typical computer the storage might consist of a device capable of storing 10,000 numbers of 40 binary digits, any of which could be taken quickly out of storage, followed by a larger storage capacity (e.g., 100,000 numbers), followed perhaps by an even larger storage unit (e.g., 1,000,000 numbers). In general, the larger the storage capacity of the unit in this hierarchy, the longer will be the time required to extract any desired number from it. Each storage cell (that is, each position in which a number may be located) carries its own identifying number, so that numbers desired may be located by specifying the numbers ("addresses") of the cells in which they are located.

Instructions to the machine consist of combinations of these addresses with code numbers designating the arithmetical operations to be performed. One method of constructing these instructions is the "four-address code". Each instruction consists of a sequence of four addresses, $\alpha$, $\beta$, $\gamma$, and $\delta$, together with an operation-code symbol. In this system, the control unit calls forth from storage first the number at the address $\alpha$, then the number at $\beta$, combines them in the arithmetic unit as directed by the operation code, inserts the result in the memory at address $\gamma$, and then calls forth the next instruction from address $\delta$. Since the "instruction" is simply a sequence of digits, it is indistinguishable from a number except by the way in which it is employed by the machine. Thus it may be inserted in storage cells to be used when needed, or may even be inserted in the arithmetic unit to be changed to a different instruction by addition of suitable numbers to its $\alpha$, $\beta$, $\gamma$, $\delta$, or operation code.

Storage systems can be made in many different physical forms; these will be treated in some detail in Chap. 11. At this point, however, to give the reader some grasp of their nature, we shall describe one of the simpler, widely used storage devices, the magnetic drum. The drum is a rotating cylinder with a magnetic coating on its surface. Mounted close to it are "writing" heads, which can be driven by currents of either

polarity to magnetize an adjacent small spot on the drum surface with flux in one or the other direction. As these magnetized spots pass under "reading" heads (often the same heads that are used for writing), they induce a voltage of one polarity or the other in the head winding. Thus the binary digits 0 and 1 can be stored as flux patterns on the drum surface. They can be read back as often as desired with no loss in output and are changed only by writing a different pattern of 0's and 1's over them. Figure 5.3 shows a head-and-drum configuration.

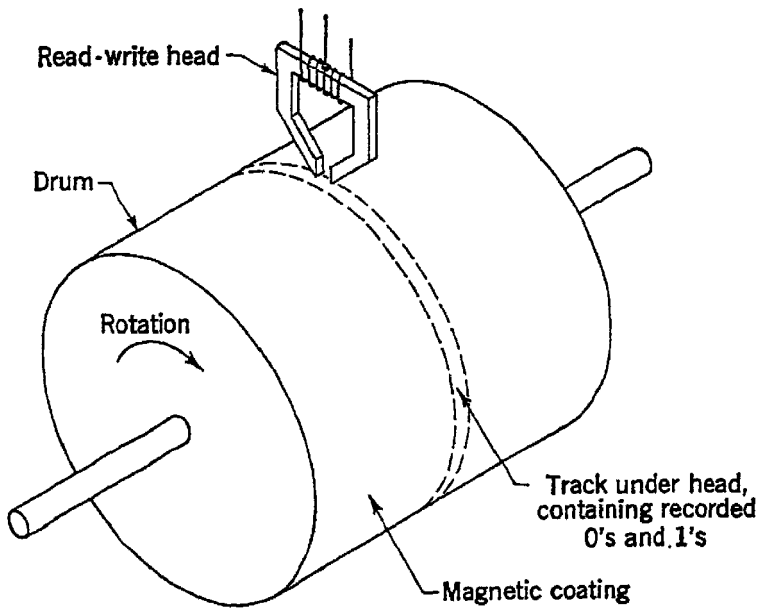



Fig. 5.3 A magnetic recording head-and-drum configuration

To record as many as 50 to 100 or more digits per inch around the drum circumference requires a small air gap (about 0.001 in.) in the head. Thus to get any substantial flux through the drum surface requires a head-to-drum spacing of under 0.001 in. Drum diameters of 8 in. are common, and the tolerance on this dimension is often put at $\pm 0.0001$ in.

A "track" is the part of the drum surface that passes under a head. If the digits are recorded sequentially along a track, then a single storage "cell" is that part of a track in which the digits of a single number or instruction are recorded. To distinguish one cell from another, an identifying mark is recorded opposite the beginning of each cell on another track, and a group of digits numbering each circumferential cell location is placed on another track. A third indexing track has 1's recorded all the way around to mark locations in the storage tracks where digits are recorded. The layout for these tracks and a storage track is shown in Fig. 5.4.

As the drum rotates, the cell addresses are read from the address track and stored temporarily, perhaps in a bank of flip-flops. The address that the control is seeking has already been similarly stored, and sensing

circuits determine when these two are the same. A signal results which allows the next word on the desired track to be read out and routed to the part of the machine determined by the digits of the operation code. Transfers to the drum are effected in much the same way, except that writing circuits instead of reading circuits are activated.

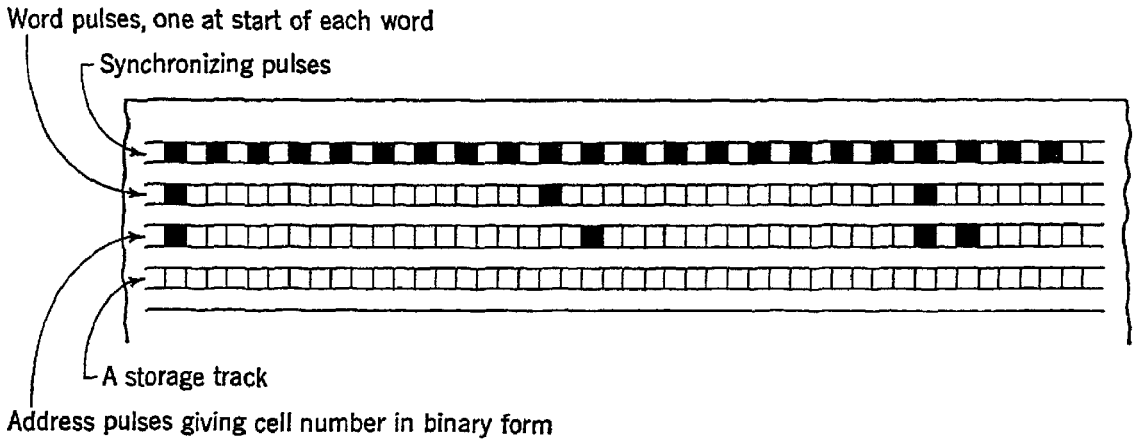



Fig. 5.4 Layout of indexing tracks and a storage track on the drum surface

The entire operation of the computer consists, then, of transmitting digits along various paths to devices that can perform binary addition, complementation, sensing, and comparison. The paths chosen, the direction of flow, and the state of activation of the circuits are fixed by the designer for each of the operation codes that he builds into the machine. The problem of designing these devices and their circuit interconnections will be treated in some detail in Chaps. 8 to 10. By contrast with the engineer, however, the programmer is not interested in the details about paths taken through the computer by the various digits but only in the end result accomplished when the machine processes an instruction in the manner indicated by the operation code. In the next chapter we shall discuss the computer from the programmer's point of view, and then turn to a treatment of the engineering of the computer.

CHAPTER 6

# Solving Problems on the Digital Computer

## 6.1 Introduction

As indicated in the preceding chapter, the first step in preparing a problem for the digital computer is to resolve it into a succession of simple steps that the machine can do. This often requires that the programmer be a skilled mathematician who can restate complex problems so that the machine can handle them. Since the machine handles only quantized and digitized representations of variables, the programmer must replace continuous processes such as those of calculus with stepwise processes expressed as difference equations. Furthermore, since there is an upper limit to the number of digits in each group that a computer manipulates, machine precision is limited, though frequently high. To estimate the maximum error occurring in a machine solution of a problem may require very careful study of the interplay between the finite precision of the computer and the limited accuracy of the approximation used in digitizing the problem.

The more purely mathematical aspects of problem preparation will not concern us here, and the reader whose interest is in that direction will find adequate treatments elsewhere in the literature.[1–3] Our present concern is more with the mechanics of program preparation.[4–8]

We shall begin by considering some examples of programming with the four-address code. From the discussion in the preceding chapter it will be recalled that, in arithmetical operations, at least, the $\alpha$ digits of an instruction specify the cell number (address) of the first operand, the $\beta$ digits the address of the second operand, the operation-code digits the manner of combining these operands, the $\gamma$ digits the address at which the result is to be placed, and the $\delta$ digits the address at which the digits forming the next instruction are to be found. Since the "instruction" is simply a sequence of digits, it is indistinguishable from a number except by the way in which it is employed by the machine.* Accordingly, it

* This is true of most computers, although some have been built which provide one storage and processing facility for numbers and another for instructions.

may be inserted in storage cells to be used when needed or may even be inserted in the arithmetic unit to be changed to a different instruction by addition of suitable numbers to its $\alpha$, $\beta$, $\gamma$, or $\delta$ digits or to its operation code. In speaking broadly of these digit groups, without distinguishing between numbers and instructions, we shall use the name "word".

Let us now define some simple operations that a typical machine might perform. We employ parentheses to designate the word at a given address; that is, the notation "$(j)$" will mean "the word contained in the storage cell designated $j$". Our operations are as follows:

| Operation | Abbreviation | Meaning |
|---|---|---|
| Addition | ad | $(\alpha) + (\beta)$ is placed in $\gamma$. Next instruction is $(\delta)$. |
| Subtraction | su | $(\alpha) - (\beta)$ is placed in $\gamma$. Next instruction is $(\delta)$. |
| Multiplication | mu | $(\alpha) \times (\beta)$ is placed in $\gamma$. Next instruction is $(\delta)$. |
| Division | dv | $(\beta) \div (\alpha)$ is placed in $\gamma$. Next instruction is $(\delta)$. |
| Comparison of magnitudes | cm | If $\lvert(\alpha)\rvert < \lvert(\beta)\rvert$, next instruction is $(\gamma)$. If not, next instruction is $(\delta)$. |
| Read-out | ro | $\alpha$ = number of words to be read out<br>$\beta$ = location of first word to be read out<br>$\gamma$ = number of output unit<br>$(\delta)$ = next instruction |

## 6.2 A four-address iterative program

As an example of the use of these operations, we shall set up a sequence of instructions for taking the square root of a number $x$ by a successive-approximation process derived from Newton's formula:

$$y_{n+1} = \frac{y_n + x/y_n}{2} \tag{6.1}$$

$y_n$ is the $n$th approximation to the square root of $x$, and $y_{n+1}$ is a better approximation. This process converges fairly rapidly, even for a poor choice of $y_0$, as the reader may verify by trying a few iterations with $x = 25$ and $y_0 = 1$.

We shall find it necessary to store three types of quantities during the calculation: (1) the instructions, (2) constants needed in the calculation, and (3) "temporaries", numbers that change during the calculation or arise during it. Since we do not know in advance how many of each are needed, we shall use a block addressing system in which addresses of instructions are designated with reference to a starting point A, addresses

of constants with reference to a starting point B, and addresses of temporaries with reference to a starting point C. Thus, an address such as 003C means the address that equals 3 + C, whatever the numerical value ultimately given to C. When the program of instructions has been fully written out, we can assign numerical values to A, B, and C and either add them in ourselves or program the computer to perform this operation.

**Table 6.1**
**Storage locations of constants and temporaries**

| Constants | | Temporaries | |
|---|---|---|---|
| Cell number | Contents | Cell number | Contents |
| 000B | $x$ | 000C | $y_n$ (initially 1) |
| 001B | 0 | 001C | $x/y_n$ |
| 002B | 2 | 002C | $y_n + x/y_n$ |
| 003B | $\varepsilon$ | 003C | $\frac{1}{2}(y_n + x/y_n) = y_{n+1}$ |
| | | 004C | $y_{n+1}^2$ |
| | | 005C | $y_{n+1}^2 - x$ |

The constants required for this problem will be the number $x$ whose square root we wish to determine, the number zero, the number 2, and a precision constant $\varepsilon$ to determine whether we have calculated enough digits. Table 6.1 lists the constants and temporaries to be stored and the cell addresses in which we store them.

**Table 6.2**
**Program of instructions for single square root**

| Instruction location | Instruction | | | | | Explanation |
|---|---|---|---|---|---|---|
| | $\alpha$ | $\beta$ | $\gamma$ | $\delta$ | Operation | |
| 000A | 000C | 000B | 001C | 001A | dv | Calculates $x/y_n$ |
| 001A | 000C | 001C | 002C | 002A | ad | Calculates $y_n + x/y_n$ |
| 002A | 002B | 002C | 003C | 003A | dv | Calculates $y_{n+1}$ |
| 003A | 003C | 003C | 004C | 004A | mu | Calculates $y_{n+1}^2$ |
| 004A | 004C | 000B | 005C | 005A | su | Calculates $y_{n+1}^2 - x$ |
| 005A | 005C | 003B | 006A | 007A | cm | Is $\lvert y_{n+1}^2 - x\rvert < \varepsilon$? |
| 006A | 1 | 003C | 1 | | ro | Reads out result |
| 007A | 001B | 003C | 000C | 000A | ad | Substitutes $y_{n+1}$ for $y_n$ and starts over |

A possible set of instructions to perform the calculation is shown in Table 6.2. These instructions are shown in the form in which the programmer might write them, but of course when they are placed in the machine, each of the letters A, B, and C is replaced by a numerical value, and the mnemonic abbreviations ad, dv, etc., are replaced by binary digit groups.

We note that, although many iterations may be required to give an answer of the desired precision, it is necessary to program only one complete iteration. The comparison (or "jump") instruction gives the machine a limited decision-making ability that enables it to "decide" whether another iteration is or is not necessary. The test here consists in determining whether the square of $y_{n+1}$ differs from $x$ by less than some predetermined amount. If, on the basis of the comparison order (the instruction in location 005A), another iteration is needed, instruction 007A makes a modification of the contents of 000C so that instructions 000A, 001A, and 002A will carry out the calculation of $y_2, y_3, \ldots$ on subsequent iterations. In fact, it is for repetitive calculations like this, where the machine uses the same set of instructions over and over, modifying them slightly in order to reuse them, that the automatic high-speed computer finds its greatest application.

Our final step is to assign values to A, B, and C. The number A would ordinarily be zero, unless the computer performs some preliminary work, such as decimal-to-binary conversion, before starting the problem itself. Such a routine might occupy, for instance, the first ten storage cells (000 through 009), in which case we set A = 010. Thus 000A becomes 010, and 007A becomes 017. The constants are stored starting at B = 018 and going through cell 021, and the temporaries start from C = 022 and go through cell 027.

The questions naturally arise: How are the numbers and instructions placed in the computer storage initially, and how is the machine started? Although the procedure differs in detail from one computer to another, the basic principle is that certain instruction data can be set into the computer manually at the start to control the insertion of the data for the problem. The instructions and numbers for the problem are recorded on a medium such as punched paper tape or cards, which can be "read" by the computer. The manual controls in a typical computer might then be set to require that the first word read in be treated as an instruction. This first word on the tape might be a digit group corresponding to an instruction to "read in $\alpha$ words from input reader $\beta$, store them at consecutive memory locations starting at cell $\gamma$, and take the next instruction from cell $\delta$." Pushing the START button initiates the whole sequence of operations. In the programs of this chapter, no initial read-in instructions are included, since these are a minor part of the program and depend very much upon the input facilities associated with each computer. Instead it is assumed that all input data are stored in specified memory cells at the start.

Methods of halting the machine also differ. Some machines include a specific HALT instruction. Others are so arranged that the machine will halt after executing any instruction that has a minus sign affixed. Most

machines will also halt when certain unusual conditions arise during a calculation, such as an arithmetical operation that exceeds the digital capacity of the machine. In programs in this chapter, it will be assumed that the computer halts upon finishing any instruction that carries a minus sign.

## 6.3 Instruction modification and resetting

The program we have set up is satisfactory if we have to take the square root of only one number. However, we can demonstrate another useful characteristic of computers by generalizing the problem to the determination of the square roots of several numbers, say, 50. This characteristic is the ability of the machine to modify its own program. We shall store our constants and temporaries as listed in Table 6.3.

**Table 6.3**
**Storage locations of constants and temporaries**
**(for program in Table 6.4)**

| Constants | | Temporaries | |
|---|---|---|---|
| Cell number | Contents | Cell number | Contents |
| 000B | $x_1$ | 000C | $x_j$ |
| 001B | $x_2$ | 001C | $y_n$ |
| ... | ... | 002C | $x_j/y_n$ |
| ... | ... | 003C | $y_n + x_j/y_n$ |
| ... | ... | 004C | $\frac{1}{2}(y_n + x_j/y_n)$ |
| 049B | $x_{50}$ | 005C | $y_{n+1}^2$ |
| 050B | 0 | 006C | $y_{n+1}^2 - x_j$ |
| 051B | 1 | 007C | $j$ |
| 052B | 2 | 008C | $\sqrt{x_1}$ |
| 053B | $\varepsilon$ | ... | ... |
| 054B | 50 | ... | ... |
| 055B | 0,0,1,0,0 | ... | ... |
| 056B | 0,1,0,0,0 | 057C | $\sqrt{x_{50}}$ |

The set of instructions to perform the calculation appears in Table 6.4. Through the use of JUMP instructions and instruction modifiers, only 16 instructions are needed to program the square roots of 50 numbers, each of which will require several iterations. Instructions 002A through 005A calculate an approximation to the square root, and 006A, 007A, and 008A determine whether that approximation is close enough. If it is not, 009A inserts this value as a starting point for calculation of a better approximation, starting again from 003A. If it is, 010A stores this value, and 011A determines whether all square roots have been taken. If all square roots have been taken, 012A reads out the desired numbers. If not, 013A adds 1 to the index $j$, and 014A and 015A change 001A and

010A, respectively, to call for the correct addresses on the next sequence of instructions.

The instructions in cells 014A and 015A deserve special comment, since they demonstrate a very important characteristic to which we have already referred, namely, the ability of the machine to modify its own instructions. Instruction 015A adds to the digits in 010A the digits that are stored in 055B and puts the result back into cell 010A. The digits

**Table 6.4**
**Program for 50 square roots**

| Instruction location | Instruction | | | | | Explanation |
|---|---|---|---|---|---|---|
| | $\alpha$ | $\beta$ | $\gamma$ | $\delta$ | Operation | |
| 000A | 050B | 051B | 007C | 001A | ad | Sets $j = 1$ |
| 001A | 050B | 000B | 000C | 002A | ad | Transfers $x_j$ to 000C |
| 002A | 050B | 051B | 001C | 003A | ad | Sets $y_0 = 1$ |
| 003A | 001C | 000C | 002C | 004A | dv | $x_j/y_n$ |
| 004A | 002C | 001C | 003C | 005A | ad | $y_n + x_j/y_n$ |
| 005A | 052B | 003C | 004C | 006A | dv | $\frac{1}{2}(y_n + x_j/y_n)$ |
| 006A | 004C | 004C | 005C | 007A | mu | $y_{n+1}^2$ |
| 007A | 005C | 000C | 006C | 008A | su | $y_{n+1}^2 - x_j$ |
| 008A | 006C | 053B | 010A | 009A | cm | Is $\lvert y_{n+1}^2 - x_j \rvert < \varepsilon$? |
| 009A | 004C | 050B | 001C | 003A | ad | Substitutes $y_{n+1}$ for $y_n$ |
| 010A | 004C | 050B | 008C | 011A | ad | Puts $\sqrt{x_j}$ in 007C $+ j$ |
| 011A | 007C | 054B | 013A | 012A | cm | Is $j < 50$? |
| 012A | 50 | 008C | 1 | | ro | Reads out results |
| 013A | 007C | 051B | 007C | 014A | ad | Adds 1 to $j$ |
| 014A | 001A | 056B | 001A | 015A | ad | Instruction modifier |
| 015A | 010A | 055B | 010A | 001A | ad | Instruction modifier |

originally stored in 010A constituted the eleventh instruction in our list, but it is apparent that this instruction should be different each time it is performed, since the first square root is to be stored in 008C, the second in 009C, and so on. If we add 1 unit to the $\gamma$ part of the instruction in cell 010A after each execution of this instruction, then the instruction calls for the correct storage location of the next square root. This addition is accomplished by the addition of the digits in cell 055B, which are indicated in the table of constants as being "0, 0, 1, 0, 0". This grouping signifies that the $\alpha$, $\beta$, $\delta$, and operation-code parts of the word in 055B are all zero, while the $\gamma$ part is $00 \cdots 1$. Thus, adding such a group to an instruction increases its $\gamma$ address by 1 each time the addition is performed. A similar process is carried out by instruction 014A, which increases the $\beta$ address of instruction 001A by adding the modifier digits stored in cell 056B.

(The exact form in which the modifier digits should be arranged for insertion into the computer will depend upon the instruction structure

of the machine. For instance, the ten most significant binary digits of an instruction might specify $\alpha$, the next ten $\beta$, the next ten $\gamma$, the next ten $\delta$, the next four the operation code, and the last the sign. If the computer is one in which all number words are regarded as fractions—a common type—then the modifier digits in cell 055B can be written as the binary equivalent of the single number $2^{-30}$ and the content of 056B as $2^{-20}$.)

One shortcoming of our program as it now stands is that it does not reset itself at the start to begin with the proper sequence of instructions. It frequently happens that, after a program has been partially run on a machine, a programming error or a machine malfunction will cause the computer to stop. It is often easier to start the program over again than to correct the error where it occurred and continue calculation from there. But, if we attempt to start our present program again, we note that, although the first instruction sets the index $j$ to 1, as it should, instructions 001A and 010A have been modified by the previous partial performance of the program and do not start now at cells corresponding to $x_j = x_1$. Thus it would be desirable at the beginning of our program always to reset 001A and 010A to their correct initial values. We can do this if we wish by inserting the desired instructions after 000A and then renumbering all subsequent instructions and all references to these instructions. However, by using another method, we can demonstrate an advantage of the four-address code, namely, the ease with which omitted instructions can be inserted. First let us store in cell 057B the desired initial setting for instruction 001A; i.e., let

$$(057\text{B}) = 050\text{B}\ 000\text{B}\ 000\text{C}\ 002\text{A}\ \text{ad}$$

and also let (058B) = 004C 050B 008C 011A ad, which is the desired initial setting of 010A. Then let us change the $\delta$ address of 000A to be 016A instead of 001A. Now at the end of the program we append the following instructions:

| Instruction location | $\alpha$ | $\beta$ | $\gamma$ | $\delta$ | Operation | Explanation |
|---|---|---|---|---|---|---|
| 016A | 057B | 050B | 001A | 017A | ad | Sets (001A) for $j = 1$ |
| 017A | 058B | 050B | 010A | 001A | ad | Sets (010A) for $j = 1$ |

Thus, whenever the program is started off with the first instruction, control jumps to 016A, and the group of two instructions there (together with the first) sets $j = 1$ at all its appearances in the program and then transfers control back to instruction 001A. The program has now been made automatically resetting.

The process of instruction modification carried out in the foregoing programs is typical of the way in which the process is handled in what are known as "ambiguous-word" machines. In such machines a word-length group of digits might be either a number or an instruction, and the group itself has no identifying characteristics to designate it as being uniquely one or the other. The difference arises only in the manner of use of the word. A word that is taken from storage and treated as an instruction is an instruction, and a word treated as a number is a number. The same word, in fact, may be on one occasion an instruction and on another a number. Thus the words in cells 001A and 010A in our square-root program are instructions most of the time—that is, whenever they are dissected to find their address components—but they are numbers when instructions 014A and 015A are executed, since they are sent to the arithmetic unit and added to the modifier digits. The term "ambiguous" is somewhat unfortunate, since it carries a connotation of doubt about what is meant. Actually, of course, there is never any doubt which interpretation the computer should place upon a given word at a given point in its operation, since the program of instructions must fully specify what the machine is to do.

### 6.4 Three-address instructions

The reader may already have observed that, in the programs we have seen in the four-address form, the $\delta$ addresses are more often than not a sequence of consecutive numbers. This suggests that it might be more practical to have the machine generate its own $\delta$ address merely by adding 1 to the value of the current $\delta$ address upon completion of each instruction. Then it would no longer be necessary to specify the $\delta$ address as part of each instruction. This would represent a small saving in work for the programmer but would result in a more important saving in storage space in the computer, since the instruction word could now be shorter. The instruction with only three addresses is slightly less flexible than the four-address instruction, since an extra instruction must be inserted wherever instructions are not taken in sequence, whereas, in the four-address code, the $\delta$ address can be used to depart from the sequence. As in the four-address code, changes in sequence can be accomplished by the comparison or JUMP instruction. The comparison instruction might now be interpreted as "If $|(\alpha)| < |(\beta)|$, set the instruction counter to $\gamma$. Otherwise add 1 to the content of the instruction counter."

In a three-address code, the $\alpha$, $\beta$, and $\gamma$ parts of the instruction word specify the storage locations of the two operands and of the result. (This is true, at least, of arithmetical operations, but as we shall see soon, there are certain "logical" operations that do not involve "operands", and for

these a different interpretation is placed upon the $\alpha$, $\beta$, and $\gamma$ components.) The program of instructions is stored sequentially in the memory system, and a counter originally set to the location of the first instruction word is automatically advanced by 1 upon the completion of each instruction. The contents of this counter then indicate the location of the next instruction to be performed.

Let us now modify our four-address square-root program to the three-address form. At the same time, since it is awkward to have our instruction-resetting instructions appended at the end of the program, we shall incorporate them at the beginning and make whatever consequent address changes we need in the rest of the program. We must store two additional words, which are the initial settings for the instructions that are modified during the program. These we put in cells 057B and 058B, as follows:

$$(057B) = 050B\ 000B\ 000C\quad \text{ad}$$
$$(058B) = 004C\ 050B\ 008C\quad \text{ad}$$

Moreover, our instruction-modifier digits in cells 055B and 056B will be different, since the three-address instruction has only four components ($\alpha$, $\beta$, $\gamma$, and operation code) instead of five. These cells now store

$$(055B) = 0, 0, 1, 0$$
$$(056B) = 0, 1, 0, 0$$

The modified program appears in Table 6.5.

This program differs from that in Table 6.4 in that the instruction-resetting instructions have been included at the start of the program as instructions 001A and 002A. Thus corresponding steps in the two programs are not stored at the same locations. The other changes result from using a three-address instruction form, with a slightly different type of comparison instruction. The three-address comparison instruction is less versatile than the four-address one, since, depending upon the outcome of the comparison, control either does or does not jump, whereas, in the four-address form, control can be made to jump upon either outcome of the comparison. Therefore, in the three-address form it is sometimes necessary to use a forced jump, as in instruction 019A of Table 6.5. This instruction compares 1 with 2 and always transfers control to instruction 003A.

One further advantage of the four-address code that should be pointed out is that omissions in programming are more easily inserted with a four-address code than with a three-address code. An example of this is the modification we found desirable to make in the program of Table 6.4 in order to make the program always start from the same point. The

**Table 6.5**
**Three-address square-root program including self-resetting of modified instructions**

| Instruction location | Instruction | | | | Explanation |
|---|---|---|---|---|---|
| | $\alpha$ | $\beta$ | $\gamma$ | Operation | |
| 000A | 050B | 051B | 007C | ad | Set $j = 1$ |
| 001A | 057B | 050B | 003A | ad | |
| 002A | 058B | 050B | 013A | ad | |
| 003A | 050B | 000B | 000C | ad | Transfers $x_j$ to 000C |
| 004A | 050B | 051B | 001C | ad | Sets $y_0 = 1$ |
| 005A | 001C | 000C | 002C | dv | $x_j/y_n$ |
| 006A | 002C | 001C | 003C | ad | $y_n + x_j/y_n$ |
| 007A | 052B | 003C | 004C | dv | $\frac{1}{2}(y_n + y_j/y_n)$ |
| 008A | 004C | 004C | 005C | mu | $y_{n+1}^2$ |
| 009A | 005C | 000C | 006C | su | $y_{n+1}^2 - x_j$ |
| 010A | 006C | 053B | 013A | cm | Is $\lvert y_{n+1}^2 - x_j \rvert < \varepsilon$? |
| 011A | 004C | 050B | 001C | ad | Substitutes $y_{n+1}$ for $y_n$ |
| 012A | 051B | 052B | 005A | cm | Forced jump to 005A |
| 013A | 004C | 050B | 008C | ad | Stores $\sqrt{x_j}$ in 007C $+ j$ |
| 014A | 007C | 054B | 016A | cm | Is $j < 50$? |
| 015A | 50 | 008C | 1 | ro | Reads out results |
| 016A | 007C | 051B | 007C | ad | Adds 1 to $j$ |
| 017A | 003A | 056B | 003A | ad | Instruction modifier |
| 018A | 013A | 055B | 013A | ad | Instruction modifier |
| 019A | 051B | 052B | 003A | cm | Forced jump to 003A |

instruction-resetting instructions had to be included after instruction 000A and before 001A. A jump out of the sequence to the two instructions at the end was easily made by means of a change in the $\delta$ address of 000A. With the three-address form, however, no $\delta$ address is available, and to jump out of the sequence we must use a comparison instruction. This must be put in place of the instruction in 000A, which must therefore go at the end of the program together with the instructions to be added. After the added instructions another FORCED JUMP instruction must be included to return control to the beginning of the program.

## 6.5 Modes of computation

An upper limit ordinarily exists for the number of digits contained in any number produced or encountered by a computing machine. There are two ways in which this limited number of digits can be used for representing numbers within the machine:

1. Fixed-point representation. The radix point occurs at the same position (frequently the extreme right or the extreme left) in all numbers handled by the machine.

2. Floating-point representation. All numbers in the machine consist of two parts, one an exponent for the base of the number system and the other a fraction having the radix point followed by the most significant nonzero digit of the number.

In fixed-point operation with the radix point at the right, all numbers are treated as integers. Any fractions must be multiplied by an appropriate scale factor to make them integers. Since arithmetical operations may produce result numbers having more digits than the original operands, the possibility of overflow beyond the digital capacity of the machine must always be recognized. With the radix point always at the right, any overflow is always at the left, and the overflow digits that are lost are the most significant digits. On the other hand, placing the radix point at the left allows those overflows which arise during multiplication to take place at the right, and the lost digits are then the least significant. For this reason, the radix point is usually at the left in fixed-point operations, and any integers must be scaled down to be fractions before they can be placed in the machine.[9]

Although floating-point arithmetic does not eliminate the difficulties due to overflow, it simplifies the scaling problem, since the digits of the word are used in such a way as to give an extremely wide range to the size of numbers represented. Thus the programmer is spared the necessity of keeping careful control over the sizes of the numbers in his problem. A further advantage is that all numbers are represented with the same number of significant digits. As an example, consider a machine with a word length of 20 bits. If we use the sign and eleven of these to represent the fractional part (sometimes called the mantissa) and the remaining eight to represent the exponent of 2 and its sign, then numbers $x$ can be represented that lie in the range

$$\tfrac{1}{2} \times 2^{-127} \leq |x| \leq (1 - 2^{-11}) \times 2^{+127}$$

The machine automatically normalizes all numbers so that the most significant 1 lies adjacent to the radix point and the amount of the shift is incorporated in the exponent part of the word. Thus the numbers 50 and 70 would be represented in binary floating-point form in the machine as 0.11001000000,00000110 and 0.10001100000,00000111, respectively. (The comma would not appear in the machine and is inserted here only for the reader's convenience in separating the fraction and the exponent.) The machine performs the multiplication of these numbers by multiplying the fractions, adding the exponents, and renormalizing, to give 0.11011010110,00001100, which is readily seen to be the floating-point equivalent of 3,500. The machine must perform some additional manipulations if it is required to add two numbers with different exponents,

since the smaller of them must first be shifted far enough to the right to make the exponents the same. Before adding the floating-point equivalents of 50 and 70, the machine would first put the quantity 50 in the form 0.01100100000,00000111. Then it would form the sum by adding the fractional parts to give 0.11110000000,00000111.

Floating-point facilities can be very useful for problems in which the number magnitudes vary widely. However, it is difficult to achieve the maximum effectiveness of floating-point arithmetic unless it is incorporated as part of the machine's electronic structure. Floating-point arithmetic executed by appropriate programming of a fixed-point machine slows down machine speed by factors variously estimated between 5 and 20.[10,11]

## 6.6 Scale factors

In the programs we have studied so far, no scale factors were included. Let us now examine the programming complications that arise because, with fixed-radix-point operation, the numbers handled must be scaled in magnitude to fall within the limits imposed by the construction of the machine. If the radix point lies at the left, as it does in many computers, all numbers handled by the machine must be fractions. Thus a number $x$ such that $r^m \leq x < r^{m+1}$ must be shifted $m + 1$ places to the right, i.e., must be multiplied by $r^{-(m+1)}$ before it can be represented in the machine. ($r$ is the radix of the number system used in the computer.) The number the machine processes is not $x$ but $r^{-(m+1)}x$, and the shift factor $r^{-(m+1)}$ must be taken into account in any operations involving $x$.

Treatment of scaled numbers that are to be multiplied is straightforward, since the exponents in the scale factors are simply added. Thus, if $y$ lies between $r^n$ and $r^{n+1}$, its representation is $yr^{-(n+1)}$. The machine representation of $xy$ is $xyr^{-(m+n+2)}$, and, if $xy$ is to be obtained, this number must be shifted $m + n + 2$ places to the left. Division is similar, except that the exponents are subtracted. Here, however, another complication arises, since the quotient of the scaled numbers may be a number larger than unity. Since a number larger than unity cannot be formed by the machine, steps should be taken to prevent it from arising. This can be done by shifting the numerator number one place to the right before performing the division, provided that both numerator and denominator are already in such a form that their most significant nonzero digits occur in the first place to the right of the radix point.

In addition or subtraction, the numbers must first be scaled by the same factor. If $x$ and $y$ are as we have defined them and if $m > n$, it is necessary to shift $y$ to the right by $m + 1$ places before adding it to $xr^{-(m+1)}$. If $y$ has already been shifted $n + 1$ places to insert it in the

machine, a further shift of $m - n$ places to the right is needed. As in division, the possibility of an overflow exists and can be avoided by shifting each of the numbers to be added one place to the right before adding or subtracting.

The situation that occurs in the digital computer when the result of some program step is a number exceeding the digit capacity of the machine is similar to that which arises in the analog computer when one of the computing amplifiers is driven into saturation. In either machine the cause is some nonvalid operation, such as dividing by too small a quantity, multiplying by too large a quantity, or adding quantities which although individually within machine range, in combination exceed it. As we have seen, analog computers frequently contain automatic sensing devices to detect amplifier saturation and signal the operator. Digital machines also are usually provided with overflow indicators. Some of these stop the computation immediately, others turn on a control-panel warning light and continue computation, and others are arranged to jump control to a program step which starts the machine on a subroutine (see Sec. 6.11) to correct the overflow condition or to give the programmer information which he needs regarding the overflow.

The programmer's responsibility in scale-factoring extends beyond merely ensuring that the input data are properly scaled. He must also ensure that his program takes into account the scale factors attached to numbers to be added, subtracted, multiplied, or divided, and, most important of all, he should anticipate overflows by including appropriate shifting operations wherever they are needed. However, precautions against overflow should be taken only at points in the problem where examination shows overflow to be possible. Unnecessary shifting operations use valuable machine time for no purpose.

Let us study a simple example of scaling by modifying the program of Table 6.5 for some specific numbers. Let it be required to evaluate to four-decimal accuracy the square roots of the integers 2 to 51, and let us assume further that in the computer to be used all numbers must be decimal fractions. Evidently the numbers 2 to 51 must be scaled down to meet this requirement. Different scale factors might be used for the various numbers, but the programming is simpler if all numbers to be treated in the same way carry the same scale factor. The largest possible scale factor should be used to avoid loss of significant digits due to the limited length of number representations in the machine. (For example, if numbers in the machine can be only five digits long and the number 106.52 is to be inserted, a scale factor of $10^{-3}$ would result in .10652, but a smaller factor such as $10^{-4}$ would give .01065, in which one significant figure has been lost.) Accordingly, a suitable scale factor for the integers 2 to 51 is $10^{-2}$. Moreover, the constants 1 and 2 could

have factors of $10^{-1}$. This is not a good choice for the number 1, however, since the program requires the repeated addition of this number to itself to form the index $j$, which can grow as large as 50. $j$, therefore, must carry a scale factor of $10^{-2}$, and so must the number 50. To determine the precision constant $\varepsilon$ we note that, if $y_n^2$ differs from $x_j$ by less than $10^{-8}$, then $y_n$ is correct to four decimal places. However, since $x_j$ is scaled by $10^{-2}$, our precision constant should be $10^{-10}$.

A number of other precautions must also be observed. To avoid producing during the calculation any numbers as large as unity, we must

**Table 6.6**
**Storage locations for scaled square-root problem**

| Constants | | Temporaries | |
|---|---|---|---|
| Cell number | Contents | Cell number | Contents |
| 000B | $2 \times 10^{-2}$ | 000C | $x_j \times 10^{-2}$ |
| 001B | $3 \times 10^{-2}$ | 001C | $y_n$ |
| ... | ... | 002C | $x_j \times 10^{-2}/y_n$ |
| ... | ... | 003C | $\frac{1}{2}y_n$ |
| ... | ... | 004C | $\frac{1}{2}x_j \times 10^{-2}/y_n$ |
| 049B | $51 \times 10^{-2}$ | 005C | $\frac{1}{2}(y_n + x_j \times 10^{-2}/y_n)$ |
| 050B | $52 \times 10^{-2}$ | 006C | $y_{n+1}^2$ |
| 051B | 0 | 007C | $y_{n+1}^2 - x_j \times 10^{-2}$ |
| 052B | $1 \times 10^{-2}$ | 008C | $j \times 10^{-2}$ |
| 053B | $\frac{1}{2}$ | 009C | $10^{-1}\sqrt{2}$ |
| 054B | $1 \times 10^{-10}$ | 010C | $10^{-1}\sqrt{3}$ |
| 055B | $50 \times 10^{-2}$ | ... | ... |
| 056B | 0,0,1,0 | ... | ... |
| 057B | 0,1,0,0 | ... | ... |
| 058B | 051B 000B 000C ad | 058C | $10^{-1}\sqrt{51}$ |
| 059B | 051B 001B 001C ad | | |
| 060B | 005C 051B 009C ad | | |

inspect each step in the program. Thus, to avoid an overflow in the division of $x_j \times 10^{-2}$ by $y_n$, it is necessary for $y_n$ to be greater than $x_j \times 10^{-2}$. From the nature of the iterative formula it can be seen that, if $x$ is a fraction (as we have scaled it to be) and if $y_0$, the first approximation, is greater than $x$, then all subsequent $y$'s will also exceed $x$. Thus $x_j \times 10^{-2}/y_n$ would always be less than 1, so no overflow could occur in division. However, this quantity must next be added to $y_n$, and an overflow might occur here. We may avoid this possibility by observing that $y_n + x_j \times 10^{-2}/y_n$ will always be less than 2, since $y_n$ and $x_j \times 10^{-2}/y_n$ are both fractions. Therefore, instead of forming $y_n + x_j \times 10^{-2}/y_n$ first and then dividing the sum by 2, let us multiply each term by $\frac{1}{2}$ first and then add the results to get $\frac{1}{2}(y_n + x_j \times 10^{-2}/y_n)$. To start each calculation of $\sqrt{x_j}$, it is now necessary to choose a number larger than $x_j \times 10^{-2}$

as our $y_0$. A possible choice for $y_0$ would be $(1 + x_j) \times 10^{-2}$. We could form this in each case by adding $10^{-2}$ to the current value of $x_j \times 10^{-2}$, but in the present problem an easier way might be to use directly the stored value $x_{j+1} \times 10^{-2}$. This will require storing $x = 52 \times 10^{-2}$ in addition to the 50 quantities whose square roots are to be calculated.

Table 6.6 lists the storage locations for constants and temporaries, and Table 6.7 shows the program.

**Table 6.7**
**Square-root program including scaling**

| Instruction location | Instruction | | | | Explanation |
|---|---|---|---|---|---|
| | $\alpha$ | $\beta$ | $\gamma$ | Operation | |
| 000A | 051B | 052B | 008C | ad | Sets $j = 1$ |
| 001A | 051B | 058B | 004A | ad | |
| 002A | 051B | 059B | 005A | ad | |
| 003A | 051B | 060B | 015A | ad | |
| 004A | 051B | 000B | 000C | ad | Transfers $x_j \times 10^{-2}$ to 000C |
| 005A | 051B | 001B | 001C | ad | Sets $y_0 = x_{j+1} \times 10^{-2}$ |
| 006A | 001C | 000C | 002C | dv | $(x_j \times 10^{-2})/y_n$ |
| 007A | 001C | 053B | 003C | mu | $\frac{1}{2}y_n$ |
| 008A | 002C | 053B | 004C | mu | $x_j \times 10^{-2}/2y_n$ |
| 009A | 003C | 004C | 005C | ad | $\frac{1}{2}(y_n + x_j \times 10^{-2}/y_n)$ |
| 010A | 005C | 005C | 006C | mu | $y_{n+1}^2$ |
| 011A | 006C | 000C | 007C | su | $y_{n+1}^2 - x_j \times 10^{-2}$ |
| 012A | 007C | 054B | 015A | cm | Is $\lvert y_{n+1}^2 - x_j \times 10^{-2}\rvert < \varepsilon$? |
| 013A | 005C | 051B | 001C | ad | Substitutes $y_{n+1}$ for $y_n$ |
| 014A | 051B | 052B | 006A | cm | Forced jump to 006A |
| 015A | 005C | 051B | 009C | ad | Stores $\sqrt{x_j} \times 10^{-1}$ |
| 016A | 008C | 055B | 018A | cm | Is $j < 50$? |
| 017A | 50 | 009C | 1 | ro | Reads out results |
| 018A | 008C | 052B | 008C | ad | Adds 1 to $j$ |
| 019A | 004A | 057B | 004A | ad | Instruction modifier |
| 020A | 005A | 057B | 005A | ad | Instruction modifier |
| 021A | 015A | 056B | 015A | ad | Instruction modifier |
| 022A | 051B | 052B | 004A | cm | Forced jump to 004A |

One shortcoming of the program of Table 6.7 lies in the choice of $(1 + x_j) \times 10^{-2}$ as the initial approximation to the square root of $x_j \times 10^{-2}$. The only possible advantage of this as the initial approximation is the ease with which it is obtained. A much better initial approximation to $\sqrt{x_j \times 10^{-2}}$ would be $\sqrt{x_{j-1}} \times 10^{-1}$, that is, the last previously calculated square root. By starting with an approximation which is much closer to the final value, many iterations can be saved, and the problem will require much less machine time. Revision of Table 6.7 in this manner is left as an exercise for the reader. (The first square root

must start from an initial approximation found in some other manner, since no previous square root is available for use in the first calculation.)

## 6.7 Flow diagrams as an aid to programming

In the simple examples just given it was a fairly easy matter to visualize the various cycles through which the calculation passed and, hence, to program repetitions of these cycles. In more complicated problems, however, it is often helpful to make a diagram to show the "flow" of information and control through the various stages of the computation.

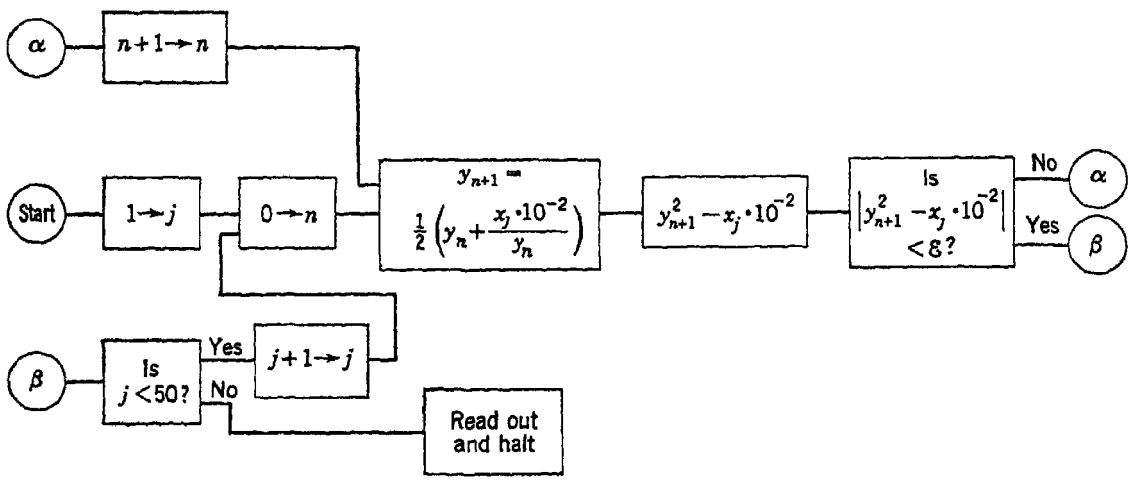



Fig. 6.1 A flow diagram for the square-root problem

The method described here is a somewhat simplified version of that originally suggested by von Neumann and Goldstine of the Institute for Advanced Study.[12]

The flow diagram is an interconnection of boxes that indicate steps or groups of steps in the computation. Each box may be either an

1. Operation box—within which is written the calculation to be carried out at this point in the routine, or an

2. Alternative box—which contains a question to be answered by the machine. The path divides into two possible paths leading out, and control follows one or the other path depending on whether the answer to the question is yes or no.

We shall show operation boxes as rectangles and alternative boxes as triangles with the input at the vertex and the alternative outputs at the base. Remote connections will be shown as circles carrying letter designations. Using these conventions, we may draw a flow diagram for the program as in Fig. 6.1.

The notation" $j + 1 \rightarrow j$" means "$j + 1$ replaces $j$". Topologically, this diagram is identical with two loops, one within the other, as shown in Fig. 6.2. The inner loop corresponds to a calculation of

$y_{n+1} = \frac{1}{2}(y_n + x_j/y_n)$ and must be traversed many times before a single value of $\sqrt{x_j}$ has been calculated. The outer loop corresponds to a complete calculation of $\sqrt{x_j}$ and is traversed once for every $x_j$.

Note that it is not necessary to show a box for every program step. A box indicates a related group of steps that occur together. No distinction in form is made between boxes which operate on indices and boxes which operate on problem data, although some programmers distinguish an index operation by a square box set up on its corner.

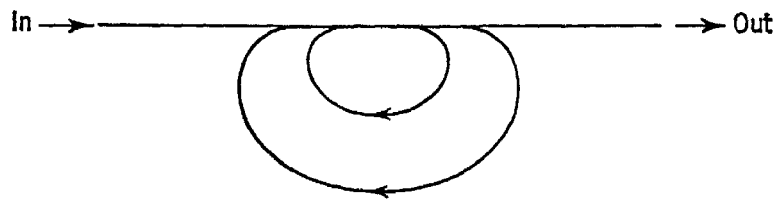



Fig. 6.2 Topological equivalent of Fig. 6.1

Careful preparation of a flow diagram is an important step, since it can produce a visualization of the sequence of calculations which will greatly aid the programmer in seeing his way through a maze of computational loops. The flow diagram is, in effect, a restatement of the problem in a language one step closer to its final expression in machine language.

## 6.8 One- and two-address instructions

The programs we have considered up to this point have been made up of instructions that contained either three or four addresses and a group of digits specifying the operation to be performed. Although such instructions have some advantageous features, they are not necessarily the best forms, and computers have also been built using two-address and single-address codes, each of which may have advantages for some particular circumstances.

Machines that use three-address and four-address codes have arithmetic units that start off fresh with each new instruction; i.e., when the results of a previous instruction have been stored in the memory, the arithmetic unit is cleared and ready for the operands of the next instruction. In contrast, computers using single-address and two-address codes employ an accumulator, which holds the result of a previous instruction unless it is specifically cleared. In this respect they are like desk calculating machines. In the single-address code, each instruction consists of an address and an operation code. Depending on the operation specified, the address may be that of a number to be added into the accumulator or that to which the contents of the accumulator are to be transferred.

Such machines usually contain a register in addition to the accumulator register into which one of the operands may be inserted for multiplication or division. With a single-address code, an automatically sequenced counter stores the address of the next instruction, as with the three-address code.

Instructions in the two-address code may use the second address to specify the location of the next instruction or may serve as double-operation instructions, such as "add the contents of cell $\alpha$ into the accumulator and then add the contents of cell $\beta$ into the accumulator." (In some machines, each instruction word actually consists of two separate single-address instructions that are performed in sequence. Although such words do contain two addresses, they are not properly examples of a two-address code.)

Although, on superficial inspection, the single-address code might appear to require even more detail in programming than three- or four-address codes, it is at no disadvantage in such a problem as that of adding a column of numbers, since it is not necessary to clear the accumulator after each addition. Thus the number of instructions for a problem is not necessarily greater in the single-address code—although for many problems it will be. For instance, to add two numbers stored in A and B and to put the result in C the following instructions would be performed:

1. Clear accumulator.
2. Add contents of A into accumulator.
3. Add contents of B into accumulator.
4. Transfer contents of accumulator to C, and clear.

Since the accumulator accumulates the results of previous operations, it need be cleared only at the end of the series of operations that produce the desired total. Thus to add a number into the accumulator takes only one instruction, whereas to clear the accumulator, add $n$ numbers, and store the result takes $n + 2$ instructions. This means that, although a three-address instruction specifies three times as much information as a single-address instruction, the single-address machine is by no means at a 3:1 disadvantage in the number of instructions needed for a program. Furthermore, the control circuits of the single-address machine can be simpler than those of the multiaddress machine.

## 6.9 Minimum-access programming[13–15]

Machines that use a single-address code or a three-address code are at some disadvantage timewise in comparison with two- or four-address machines if their storage systems are of the cyclical type. In a cyclical memory the stored information is continuously circulating and is available only at the time it passes the read-out point in the memory. In

reading out the next instruction from the memory there is a delay which averages half the memory recirculation time. This delay can be reduced with the four-address code or with the two-address code that specifies the next instruction location if the next instruction is always located in a cell that passes the read-out point immediately after the completion of the current instruction. This process is known as minimum-access, or optimum, programming. It can be used with any type of address code to shorten the time required to locate operands or to store results, but can reduce instruction-read-out delay only with four-address codes or the type of two-address code described above. (It is possible to renumber the storage cells in such a way as to reduce instruction-read-out delay with single-address machines.[15]) Minimum-access programming is, of course, unnecessary with memory systems of the noncyclical type, in which all memory cells are equally accessible at any time.

## 6.10 Special operations to facilitate computation

Although computers have been built that use no more than the few operations we have employed up to now, practical large-scale high-speed machines include in their structure many more special-purpose operations that the machine can execute. Although all these special functions can be programmed from a very limited set of basic operations, the cost of this programming in computation time and in program storage space might be prohibitive. Conversely, however, the designer must be sure that the added complexity of equipment required for special operations is justified by savings in computation time and storage space.

Special operations built into general-purpose machines are often nonarithmetical in nature; that is, their function is to facilitate control operations such as scale-factoring, index tabulation, jumps, and subroutine entry or exit. Arithmetical operations such as square-rooting are not usually included in general-purpose machines, although a special-purpose machine designed for problems in which the square root often arises might well have such an operation built permanently into its repertory of operations. However, many machines do include two or three different types of multiplication operations. One of these is "double-precision" multiplication, in which all $2n$ digits resulting from the multiplication of two numbers $n$ digits long are preserved. Two storage cells in the memory are required to store this number. In some machines, one transfer instruction transfers the double-length result to two adjacent storage cells. In some other machines, the double-length product is formed by one instruction, but its two parts must be transferred separately to two arbitrary storage locations. Still other machines do not have a double-length register in their arithmetic unit and therefore form

the double-length result in two separate operations: high-order multiplication, which preserves the $n$ most significant digits of the result, and low-order multiplication, which preserves the $n$ least significant digits of the result. The other, more common type of multiplication is "rounded" multiplication, in which the $n$-digit result comprises the $n - 1$ most significant digits of the $2n$-digit result and in which the $n$th digit has been rounded according to the magnitude of the $(n + 1)$st digit.

Another special operation is a SHIFT instruction, by means of which any word can be shifted right or left a desired number of places. This instruction is extremely useful in scaling magnitudes of numbers. In some machines the digit group that specifies how many places the operand is to be shifted is stored in a cell whose address is given by the instruction. However, many machines state the shift constant explicitly in the body of the shift instruction, thus saving a memory access at the expense of the additional circuitry required to treat the address digits as a shift constant.

Comparison orders (also known as conditional orders or jumps) are of several types. With three- or four-address codes, the contents of $\alpha$ and of $\beta$ are compared either as they stand, including signs, or on a magnitude basis only. Both types of comparison instruction are often available in a given machine. With a single-address code two numbers can be compared by subtracting one from the other and determining whether the result is positive or negative. Two types of comparison are also available here: one in which a positive result transfers control to the address specified in the comparison instruction and another in which a negative result effects this jump. Some machines also have a JUMP ON ZERO operation, which transfers control only if the accumulator content is zero. (The equivalent in three- or four-address machines would be JUMP ON EQUALITY.)

Many other instructions more elaborate than these are provided in most machines. For instance, a facility for indexing is often furnished. In one widely used method of indexing, the machine contains, in addition to the usual instruction counter, an index or base counter that can be augmented under program control. Such a counter can be used, for example, to count the number of repetitions of a given group of instructions, and this count can be added automatically to specified address components of instructions to accomplish automatic instruction modification. In machines having this facility, each address in an instruction word can be treated either as absolute or as relative to either the base counter or the instruction counter. An absolute address is taken as it stands and is interpreted by the control as the address of the storage cell that is to be located. A relative address, however, is added to the contents of the counter to which it is relative, and the resulting number is

treated as the address of the cell to be located. Which interpretation the control is to place upon the addresses in an instruction can be indicated by a group of digits that are part of each instruction word. In a three-address machine, four digits $a\ b\ c\ d$ would be included in each instruction word. $a$, $b$, or $c$ equal to zero would indicate, respectively, that $\alpha$, $\beta$, or $\gamma$ is to be considered absolute, whereas $a$, $b$, or $c$ equal to 1 would indicate, respectively, that $\alpha$, $\beta$, or $\gamma$ is to be treated as relative. $d = 0$ would indicate that any relative address is relative to the base counter, $d = 1$ that any relative address is relative to the instruction counter. These relative-addressing facilities provide considerable flexibility in programming, especially when used in conjunction with the BASE instruction.

BASE is the instruction that augments the contents of the base counter. It also compares the *addresses* $\alpha'$ and $\beta'$. (The prime indicates the effective address, that is, the address as it stands plus the contents of the counter to which it is relative, if any.) If the address $\alpha'$ is a smaller number than the address $\beta'$, the base counter is set to $\alpha'$ and the instruction counter to $\gamma'$. But, if $\alpha' \geq \beta'$, the base counter is set to zero and the instruction counter is increased by 1. In this instruction, $\alpha$ and $\beta$ can be relative only to the base counter and $\gamma$ only to the instruction counter. By means of the BASE instruction a sequence of operations can be repeated any desired number of times. Thus, suppose $\alpha$ is treated as relative and $\beta$ and $\gamma$ as absolute. If $\alpha$ is 1 and $\beta$ is 10 and if the base counter is initially at zero, a BASE instruction at the end of the sequence will compare 1 to 10, set the base counter to 1, and return control to $\gamma$, the first instruction of the sequence to be repeated. The next execution of the BASE instruction compares $\alpha$ plus the base-counter contents, a sum of 2, to 10. The base counter is set now to 2, and the sequence is repeated. The sequence continues to be repeated until its tenth repetition produces $\alpha' = \beta'$, which resets the base counter to zero and passes control to the next instruction following the sequence.

The base counter of the preceding paragraph is only one form of indexing counter. A similar device is known in some machines as a "B box" or "B register" and in others simply as an index register. Such registers may be simply counters, or they may be more elaborate registers that store not only the total count but also a number indicating the amount to be added to the counter each time it is augmented and another number indicating the limit set upon the total count before change of control is effected. Some machines have many indexing registers, each of which can be called into action as needed in different parts of the program. Although the various indexing techniques differ in detail, the base counter and the base instruction are representative of the principles of indexing.

As an example of the use of the BASE instruction, the program of Table

6.7 has been modified and is shown in Table 6.8. The storage locations referred to in Table 6.8 are the same as those of Table 6.6, although not all the items listed in Table 6.6 are required for the present program.

The BASE instruction and the control digits simplify the problem of instruction modification, as can be seen by comparing the program of Table 6.8 with that of Table 6.7. Eight fewer instructions are required when the BASE instruction is used; moreover, several constants and temporaries required for the program of Table 6.7 are not used here.

**Table 6.8**
**Square root of 50 numbers using BASE instruction**

| Instruction location | Instruction | | | | | Explanation |
|---|---|---|---|---|---|---|
| | $\alpha$ | $\beta$ | $\gamma$ | Control digits | Operation | |
| 000A | 001 | 000 | 000 | 0000 | ba | Sets base counter to 0 |
| 001A | 051B | 000B | 000C | 0101 | ad | Transfers $x_j \times 10^{-2}$ to 000C |
| 002A | 051B | 001B | 001C | 0101 | ad | Sets $y_0 = x_{j+1} \times 10^{-2}$ |
| 003A | 001C | 000C | 002C | 0000 | dv | $x_j \times 10^{-2}/y_n$ |
| 004A | 001C | 053B | 003C | 0000 | mu | $\frac{1}{2}y_n$ |
| 005A | 002C | 053B | 004C | 0000 | mu | $x_j \times 10^{-2}/2y_n$ |
| 006A | 003C | 004C | 005C | 0000 | ad | $y_{n+1} = \frac{1}{2}(y_n + x_j \times 10^{-2}/y_n)$ |
| 007A | 005C | 005C | 006C | 0000 | mu | $y_{n+1}^2$ |
| 008A | 006C | 000C | 007C | 0000 | su | $y_{n+1}^2 - x_j \times 10^{-2}$ |
| 009A | 007C | 054B | 012A | 0000 | cm | Is $\lvert y_{n+1}^2 - x_j \times 10^{-2}\rvert < 1 \times 10^{-10}$? |
| 010A | 005C | 051B | 001C | 0000 | ad | Substitutes $y_{n+1}$ for $y_n$ |
| 011A | 051B | 052B | 003A | 0000 | cm | Forced jump to 003A |
| 012A | 005C | 051B | 009C | 0011 | ad | Stores $\sqrt{x_j} \times 10^{-1}$ |
| 013A | 001 | 050 | 001A | 1001 | ba | Has cycle been done 50 times? |
| 014A | 050 | 009C | 001 | 0000 | ro | Reads out 50 results |

Substantial savings in storage space are thus made, as well as a saving in computation time by use of a shorter program.

The instructions eliminated from Table 6.7 by use of the base operation are those which keep track of the index $j$, those which modify instructions, those which set instructions at the start, and the forced jump at the end of the list. The control digits 0101 in instruction 001A indicate that the $\beta$ address is relative to the base counter, i.e., that the actual $\beta$ address the control seeks out is the sum of 000B and the base-counter content. Since the base counter is advanced by 1 each time a base operation is done, the effective $\beta$ address of 000A is also advanced by 1, but without changing the digits stored in 000A. Thus, when the base counter is set back to zero after the 50th execution of the base operation, all instructions that contained components relative to the base counter are automatically reset to their proper initial values. The BASE

instruction and the relative-addressing digits make possible substantial savings in the purely administrative parts of a program that must be recycled a given number of times.

Another useful instruction is the FILE instruction, which can be used to facilitate the transfer of control from one sequence of instructions to another and back again. Let us again assume that instructions are of three-address form and contain a group of control digits $a\ b\ c\ d$. If $d = 0$, the FILE instruction causes the address in the instruction counter increased by 1 to be stored in the $\gamma$ part of the word located at address $\alpha'$ and causes the instruction counter to be set to $\gamma'$. (The $\beta$ digits play no part in the FILE instruction.) Thus control is transferred to the sequence of instructions starting at $\gamma'$. If the $\gamma$ part of the last instruction in the sequence of instructions starting at $\gamma'$ was the place in which the instruction-counter contents were originally filed, then this instruction may be chosen to be one that resets the instruction counter and so returns control to the point in the main program from which it departed at the FILE instruction. If $d = 1$, the FILE instruction operates similarly upon the base counter, transferring its contents increased by 1 to the $\alpha$ part of the word located at $\alpha'$ and resetting the instruction counter to $\gamma'$. Since a BASE instruction can be used to apply a sequence of instructions to each of several sets of data in turn, the second form of FILE order used with a BASE instruction makes it possible to use the BASE instruction for both an interior loop and an exterior loop in a flow diagram. Thus, suppose a problem requires that a sequence of instructions be applied to each of several sets of data, and suppose part of this sequence requires the application of a subset of instructions to each of several other sets of data. Each traversal of the exterior loop in the flow diagram requires several traversals of the interior loop, and proper use of the BASE and FILE instructions can allow traversals of both loops to be tallied by the base counter.

## 6.11 Use of subroutines

Early in the use of high-speed digital computers it became apparent that calculations that occur often either in the same problem or in different problems should not have to be reprogrammed every time they are encountered. It is convenient to have available sets of instructions for performing such recurrent calculations. These precoded programs are known as subroutines and have been divided into two categories by Wilkes, Wheeler, and Gill:[6]

1. Open subroutines—programs of instructions that are inserted by the programmer directly into the main body of instructions for a problem wherever they are needed. Each time such a subroutine is inserted into

the program all its addresses must be adjusted to conform to the main program. Although such a routine does save some programming time and effort, it does not save any storage capacity.

2. Closed subroutines—programs of instructions that are not inserted by the programmer directly into the main body of the program but are stored separately in the computer memory. At each point in the main program where the subroutine calculation must be performed, control is transferred to the subroutine, and at the end of the subroutine, control is transferred back to the main program. The programmer must include the proper transfer instructions in the main body of the program, but, since the subroutine appears in the memory only once, this system is economical of storage capacity. Use of special instructions such as FILE facilitates the application of the subroutine to many different sets of data and simplifies the transfers of control.

Let us examine a simple example of subroutine entry and departure. We may assume that the subroutine has been stored at some section of the memory, say, in cells 475 through 500. At each point in the main program at which we desire to perform the subroutine, control must be transferred to cell 475. To do this it is necessary to reset the instruction counter. This can be done by inserting a FILE instruction at the point of departure from the main program, as shown in the following example:

| | *Main program* | | *Subroutine* |
|---|---|---|---|
| ... | | 475 | First instruction of subroutine |
| ... | | ... | |
| ... | | ... | |
| 75 | Last instruction before subroutine | ... | |
| 76 | 500—475 FILE | 500 | Forced jump to $\gamma$ (last instruction of subroutine) |
| 77 | First instruction after subroutine | | |
| ... | | | |
| ... | | | |
| ... | | | |
| 113 | Last instruction before next repetition of subroutine | | |
| 114 | 500—475 FILE | | |
| 115 | First instruction after subroutine | | |

Instruction 76 files in the $\gamma$ part of cell 500 the current value of the instruction counter plus 1, that is, 77, and sets the instruction counter to 475, so that 475 is the next instruction performed. The instruction in cell 500 is a forced jump, e.g., "Compare ¼ to ½. If ¼ is smaller, do

next the instruction in cell $\gamma$." This has the effect of returning program control to cell 77. The same subroutine is used again after instruction 113, and control is subsequently returned to cell 115.

Of course, this example is somewhat oversimplified, for an actual problem would require that a new set of numbers be inserted as input data into the subroutine each time it is to be used. Here we have shown only the steps involved in transferring control to and from the subroutine.

Extensive collections of subroutines have been worked up by various computation laboratories. Such subroutines as the following are in the subroutine libraries of many computing centers:

1. Floating-point arithmetic
2. Division
3. Power series
4. Logarithms
5. $n$th root
6. Trigonometric functions

In some computers, frequently used subroutines are left permanently recorded in a comparatively slow-speed memory and are called forth into the high-speed memory when they are needed. Other machines with more limited storage capacity have the subroutine inserted by the operator for the problem in which it is needed. The subroutine can be permanently recorded on magnetic tape or punched paper tape or can be written in a notebook to be punched on tape when needed.

## 6.12 A complete order list

As an example of some of the orders and order structures discussed in this chapter, in this section we shall examine in detail the command list for the MIC, the University of Michigan Instructional Computer. In addition the reader may find in the literature many discussions concerning other computers which will be helpful in broadening his understanding of the structure and behavior of computer systems.[16–24] (The references cited here are brief descriptions from the user's point of view. Much more detailed descriptions may be found in the programming and operating manuals published by the various government and university computing laboratories and by computer manufacturers. References giving engineering features of computers are cited in Chap. 9.)

The MIC is basically a single-address machine, but instructions are packed two to a memory cell, and the instructions in each pair must be processed in sequence. Thus, the order code takes on some of the aspects

of a two-address code, although it has considerably more versatility than that of the usual two-address machine.

The MIC has 1,024 addressable storage locations, 32 of which are input-output stations, such as electric typewriters, tape readers, and punches, or other optional apparatus. It has an accumulator, which is a closed loop in which inserted information circulates until removed, and another loop, called the multiplier loop, which is used in some operations. Words are 31 bits in length. All numbers are treated as fractions, 30 bits representing the magnitude and 1 bit the sign. Each 15-bit instruction uses 10 bits to designate the address (except for a few cases in the following list), 1 bit to indicate whether the address is relative or absolute, and 4 bits to designate the operation. The machine operations are as follows:

1. ADD. ($\alpha'$) is added to (accumulator). Result is in accumulator.

2. SUBTRACT. ($\alpha'$) is subtracted from (accumulator). Result is in accumulator.

3. MULTIPLY. ($\alpha'$) is multiplied by (accumulator). Rounded high-order product is placed in accumulator, and low-order product is in multiplier loop at finish.

4. DIVIDE. (accumulator) is divided by ($\alpha'$). Result is in accumulator and remainder in multiplier loop at finish.

5. SHIFT. (accumulator) is shifted by the number of places specified by the five low-order binary digits of the $\alpha'$ digits. The fifth binary digit from the left of $\alpha'$ specifies the direction—a 1 for a right shift and a 0 for a left. Overflow digits that result from a left shift are placed in the multiplier loop, but overflow digits on a right shift are lost.

6. LOGICAL TRANSFER. Those digits of a number $A$ (which must previously have been inserted in the multiplier loop) that correspond in position to 1's in a number $B$ (which is in cell $\alpha'$) are inserted into a number $C$ (which must previously have been inserted in the accumulator loop). The result is in the accumulator loop, and the number $B$, that is, ($\alpha'$), is in the multiplier loop at the finish.

7. COMPARE. If (accumulator) is positive, machine performs next instruction in sequence. If (accumulator) is negative, the instruction counter is set to the number $\alpha'$.

8. BASE. This instruction must always be the left instruction of an instruction pair. It adds 1 to (base counter) and compares this count with the number $\alpha'$. If $\alpha' >$ (base counter), the instruction in the right-hand position is processed. This instruction must be a SET INSTRUCTION COUNTER instruction, so as to return control to the beginning of the group to be repeated. If $\alpha' \leq$ (base counter), 1 is added to (instruction counter). The computer then skips the right-hand instruction and clears the base counter to zero.

9. SET BASE COUNTER. Base counter is set to the number $\alpha$. If the

control digit in the instruction word is 1, then (base counter) is added to $\alpha$, and this sum is placed in the base counter.

10. FILE BASE COUNTER. (base counter) is inserted into right-hand $\alpha$ position of cell $\alpha'$. The operation-code digits of the right-hand instruction of cell $\alpha'$ are set to correspond to SET BASE COUNTER.

11. FILE INSTRUCTION COUNTER. (instruction counter) is inserted into the right hand $\alpha$ position of cell $\alpha'$. The operation-code digits of the right-hand instruction of cell $\alpha'$ are set to correspond to SET INSTRUCTION COUNTER. (Since the instruction counter is advanced by 1 at the start of each instruction pair, this instruction actually files 1 plus the address of the current instruction.) This instruction must always be in the left-hand position, and the right-hand instruction must be a SET INSTRUCTION COUNTER.

12. LOAD ACCUMULATOR. ($\alpha'$) is transferred to accumulator loop.

13. STORE ACCUMULATOR. (accumulator) is stored in cell $\alpha'$.

14. SET INSTRUCTION COUNTER. Instruction counter is set to the number $\alpha'$.

15. STORE MULTIPLIER. (multiplier loop) is stored in cell $\alpha'$.

The occasional necessity for placing instructions in a certain order and position makes this order list somewhat more restrictive than that of the usual single-address machine. However, this necessity arises from making the machine more versatile than the conventional single-address computer. Some instructions in the list (e.g., the BASE instruction and LOGICAL TRANSFER) involve the statement of more than one address and, hence, can be employed only by using more than one single-address instruction. This form of instruction code is also more versatile than the usual two-address instruction, which has only one operation code to specify how to use two addresses. Some two-address machines exist that can do such operations as "ADD-ADD", i.e., "Add ($\alpha$) to (accumulator) and then add ($\beta$) to (accumulator)." However, in the MIC the same result is achieved by juxtaposing two add instructions without the need of a new operation code for ADD-ADD. In fact, except for a few restrictions on the logical operations, any two instructions in the MIC can be placed in a pair, giving the equivalent of such special two-address instructions as "ADD-SUBTRACT", "ADD-MULTIPLY", "DIVIDE-ADD", and others, without requiring the large number of special operation codes that these would need.

Operation of the MIC is initiated by inserting the first instructions on four rows of manually set switches and then pressing the START button. The first instructions can call for transfers, from tape to internal storage, of the program to be performed, and can end with an instruction that transfers control to the program called in, so that operation then proceeds automatically.

## 6.13 A single-address program

As an example of the techniques used in programming for a single-address machine, the square-root problem has been recoded for the MIC, as shown in Tables 6.9 and 6.10.

Since single-address instructions are quite different from the three-address and four-address codes we have been using, a detailed discussion of the program is in order. The first instruction, in the left half of cell 000A, sets the base counter to zero initially. The right half of this cell contains a useless instruction, which acts merely as a filler. This wasted instruction calls for the contents of the accumulator to be shifted zero

**Table 6.9**
**Storage locations for MIC program (Table 6.10)**

| Constants | | Temporaries | |
|---|---|---|---|
| Cell number | Contents | Cell number | Contents |
| 000B | $2 \times 10^{-2}$ | 000C | $y_n$ |
| 001B | $3 \times 10^{-2}$ | 001C | $x_j \times 10^{-2}/2y_n$ |
| ... | ... | 002C | $y_{n+1}^2 - x_j \times 10^{-2} - \varepsilon$ |
| ... | ... | | |
| ... | ... | | |
| 049B | $51 \times 10^{-2}$ | | |
| 050B | $52 \times 10^{-2}$ | | |
| 051B | 0 | | |
| 052B | ½ | | |
| 053B | $\varepsilon$ | | |

positions. Since this instruction does nothing, it takes very little time, and the instruction counter then advances to allow the two instructions in 001A to be executed. (The filler allows the instruction "001B 1 la" to appear in the left half of 001A. It is necessary that it be in the left position, since a later command, in cell 011A, transfers control back to this instruction; the machine always executes the left instruction first and thus would take an erroneous action if this instruction were placed in the right half of 000A.) The two instructions in 001A transfer (001B) to 000C as a starting point for the calculation of each square root. Since 001B is made relative to the base counter, in the third instruction (left half of 001A) subsequent executions of this instruction transfer (002B), (003B), etc. The fifth instruction (left instruction in 002A) is also made relative to the base counter, so as to call for successive $x_j$ on successive cycles. This instruction, and all those through the left half of 005A, are arithmetic and transfer operations used to calculate $y_{n+1}$, the next approximation to the square root, which is then stored in 000C by the

**Table 6.10**
**MIC program for square root of 50 numbers**

| Instruction location | Left half | | | Explanation | Right half | | | Explanation |
|---|---|---|---|---|---|---|---|---|
| | $\alpha$ | Relative bit | Operation | | $\alpha$ | Relative bit | Operation | |
| 000A | 000 | 0 | sbc | Sets base counter to 0 | 000 | 0 | sr | Does nothing, fast |
| 001A | 001B | 1 | la | Takes $y_0$ | 000C | 0 | sa | Sets $y_0$ |
| 002A | 000B | 1 | la | Takes $x_j \times 10^{-2}$ | 000C | 0 | dv | $x_j \times 10^{-2}/y_n$ |
| 003A | 052B | 0 | mu | $x_j \times 10^{-2}/2y_n$ | 001C | 0 | sa | Stores $x_j \times 10^{-2}/2y_n$ |
| 004A | 000C | 0 | la | Loads $y_n$ | 052B | 0 | mu | $y_n/2$ |
| 005A | 001C | 0 | ad | $\frac{1}{2}(y_n + x_j \times 10^{-2}/y_n)$ | 000C | 0 | sa | $y_{n+1} \rightarrow y_n$ |
| 006A | 000C | 0 | mu | $y_{n+1}^2$ | 000B | 1 | su | $y_{n+1}^2 - x_j \times 10^{-2}$ |
| 007A | 053B | 0 | su | $y_{n+1}^2 - x_j \times 10^{-2} - \varepsilon$ | 002C | 0 | sa | Stores $y_{n+1}^2 - x_j \times 10^{-2} - \varepsilon$ |
| 008A | 051B | 0 | la | Clears accumulator (loads 0) | 002C | 0 | su | Loads $\varepsilon - (y_{n+1}^2 - x_j \times 10^{-2})$ |
| 009A | 002A | 0 | cm | Jumps if negative | 000 | 0 | sr | Does nothing, fast |
| 010A | 000C | 0 | la | Loads $\sqrt{x_j} \times 10^{-1}$ | Output | 0 | sa | Prints $\sqrt{x_j} \times 10^{-1}$ |
| 011A | 050 | 0 | ba | 50 cycles executed? | 001A | 0 | sic | Unconditional jump |
| 012A | | | HALT | End of program | | | | |

right half of 005A, to be ready for the start of the next iteration. 006A and 007A then calculate and store $y_{n+1}^2 - x_j \times 10^{-2} - \varepsilon$. 008A loads the accumulator with the negative of $y_{n+1}^2 - x_j \times 10^{-2} - \varepsilon$, the sign reversal being necessary because the comparison instruction jumps control only if the accumulator content is negative. For the numbers and scale factors chosen, $y_{n+1}^2 - x_j \times 10^{-2}$ is always positive, and $\varepsilon - (y_{n+1}^2 - x_j \times 10^{-2})$ is always negative until $\varepsilon > y_{n+1}^2 - x_j \times 10^{-2}$, so that control always jumps to the left half of 002A until the calculation has gone to the desired precision.

Whether control jumps or not, the right half of 009A is always executed. Therefore, this also must be a useless instruction, as is the right half of 000A. 010A transfers the last value of $y_n$ to an output unit, and 011A,

**Table 6.11**
**Comparison of three-address and single-address programs**

| | Three-address | Single-address |
|---|---|---|
| Number of instructions: | 14 | 25 |
| Number of storage cells for instructions: | 14 | 13 |
| Number of memory accesses in interior loop: | 35 | 23 |
| Number of memory accesses in exterior loop: | 14 | 8 |

which contains the BASE and SET INSTRUCTION COUNTER instructions, recycles the program the desired 50 times. After the 50th complete execution of the program, the HALT instruction is executed.

It is interesting to compare this program with the three-address form of the same routine given in Table 6.8. Table 6.11 compares some of the important aspects of the two.

Many more single-address than three-address instructions are required, but they can be stored in fewer cells, since they are packed two to a cell. A more important comparison is the number of memory accesses required, and here it is evident that the single-address program—in this case, at least—is superior. This is so because the result of one instruction is frequently an operand required in the next. To store this result in the memory and then call it forth again, as is necessary in many three-address machines, is wasteful of machine time. In the single-address computer, on the other hand, the result is in the accumulator at the end of the operation, where it is immediately available if needed for the next step in the routine. Noting this economy, the designers of some three-address machines have been led to include in their structure an accumulator register which has an address, just as does a memory cell, but which is physically located in the arithmetic unit rather than in the storage unit, so that access to it is much quicker than to a memory cell.

## 6.14 Structure of instruction words

Although our discussions of programming have been concerned only to a slight extent with the structure of the computing machine, some mention should be made at this point of the interrelations among such design factors as word length, storage capacity, number of addresses stated in an instruction, computing precision, speed of computation, cost of equipment, and ease of programming. Most of these factors vary from computer to computer, and the optimum choice for each of them may depend not only upon the nature of the problem but also upon the significance attached to them by the user. Thus it is difficult to optimize all these factors except in well specified situations.

In a machine that has a storage capacity of $2^m$ words, each address in an instruction must have $m$ bits, so that any storage cell can be specified. If there are $k$ addresses in the instruction and if $2^p$ different operations are available to the programmer, then the length of an instruction word is $km + p$ bits. If number words are taken to be the same length as instruction words, then $n$, the number of decimal digits that each word can represent, is found from

$$10^n = 2^{km+p}$$
$$n \cong \tfrac{3}{10}(km + p)$$

For example, in a four-address machine with 1,024 storage cells and 16 operations, $n$ would be slightly more than 13 decimal digits.

If each number in a computer is to be only a few decimal digits in length, this word length may be too short for anything but single-address instructions. For example, the Whirlwind computer at MIT carried slightly less than five decimal digits in a word 16 binary digits in length. With a storage capacity of 2,048 words, 11 of these bits were needed to specify a single address, leaving 5 available for operation codes and control functions. Whirlwind's comparatively short word length was chosen largely because the machine was originally intended to be used in real-time applications where high speed of operation was more important than high precision. For computations of a more general nature, however, it was found necessary to perform double-precision arithmetic with the computer to obtain greater accuracy. Operation of the machine was much slower in this mode.

There are many machines that use single-address instructions but employ enough digits in their numbers to make the over-all word length sufficient to contain two complete instructions. As we have already seen, the MIC is a machine of this type, and several other examples are to be found among the commercially available machines produced by various manufacturers.

The relative advantages of single-address and two-, three-, or four-address codes cannot be stated precisely except in reference to specific problems. Although a few generalizations can be made, they very quickly enter a realm of subjective evaluation. Thus it is probably safe to say that programs written for a single-address machine usually require more instructions than programs for a comparable three-address machine. Also, it is safe to say that the slight added convenience afforded by having a fourth address is seldom worth the cost in equipment needed to provide it. The arithmetic unit and the storage unit are relatively unaffected by the order structure, but the control unit is more complex in computers that use more addresses per instruction, since more circuitry must be provided to treat the various sections of the instruction in different ways. Although some machines have been built using two-address instructions, this compromise has not been widely adopted, except in machines employing cyclic memory systems, e.g., drums. The statement is sometimes made that the three-address system is simpler and easier for the newcomer to learn than the single-address system. This certainly is difficult either to prove or to disprove, and the reader who has studied the examples in this chapter may already have formed his own opinion. A very interesting case study of the relative advantages of the four types of order structure in reference to a particular problem has been reported by Carter and Ellis.[25]

Although so far we have regarded number words and instruction words as being the same length, this is by no means necessary. However, if number words are made shorter than instruction words, then the machine is not being used to its full calculating precision, and nothing is gained except a minor increase in computing speed made possible by having fewer digits to manipulate in arithmetic operations. It is sometimes practical to let the instruction word be shorter than the number word in situations where the number must be long enough to give high precision but need not be long enough for the word to accommodate another address or another instruction. In this case the computer simply disregards certain blank digit positions in the instruction word.

The precision to which numbers are represented in the machine increases with the length of the number word, but it should be pointed out that frequently computations are carried out to greater precision than that of the input data. The most efficient way to use the binary digits of a number word in fixed-point form is to employ pure binary representation of the numbers, each bit having a weight twice that of the bit to its right. For such a representation, as we have seen, $n$, the number of decimal digits equivalent to $m$ binary digits, is given by $n = 0.3m$. However, it is rarely convenient to present the input data to the computer in pure binary form, and the more usual procedure is to

give the computer the data in decimal form and have it compute the binary equivalents. The individual decimal digits are presented as groups of four binary digits, this much of the decimal-to-binary conversion being easily accomplished by the input typewriter or other device. The $m$ binary digits of the word can accommodate only $[m/4]$ decimal digits coded in this fashion,* but after converting the decimal numbers to binary form, the computer treats each number as the equivalent of $\frac{3}{10}m$ decimal digits. Thus the calculating precision is greater than the input-data precision in the ratio $10^{3m/10-[m/4]}$, or $10^{m/20}$ if $m/4$ is an integer. For example, a 40-bit word can contain 10 decimal digits each coded as a 4-bit group, but is equivalent to 12 decimal digits when used as a pure binary group. Thus calculations are performed to a precision 100 times as great as that of the input numbers. This is simply to say that after conversion the numbers are represented more compactly than before.

Some computers do not make the conversion from binary-coded decimal-digit form to pure binary form but employ binary-coded decimal arithmetic. One advantage of the conversion, however, is that the increased computing precision affords decreased sensitivity to round-off error. Furthermore, computers that operate in a pure binary mode are structurally somewhat simpler than those using a binary-coded decimal form throughout. These advantages must be weighed against the disadvantage of having to convert numbers from one form to the other at input and output. The present trend is to use a pure binary form in scientific computers, where the input and output data are usually small in quantity and the calculations extensive, but to use binary-coded decimal form in computers for business applications, where small amounts of computation are usually performed upon large amounts of data and conversions would take a large proportion of the computing time.

## 6.15 Automatic programming[26–30]

As familiarity with computers has grown in recent years, it has become apparent that much of the work of the programmer is purely routine and consists of such simple drudgery as counting program steps, assigning cell numbers, and recognizing that certain operations always follow certain other operations. This work and much more besides can be performed by the computer itself, thus freeing the programmer for more creative work. Moreover, even a highly skilled programmer makes occasional errors that may go undetected until the program is run on the computer. There are, thus, strong incentives for the development

* The symbol $[x]$ means "the greatest integer not greater than $x$," i.e., the integral part of $x$.

of automatic means for preparing error-free programs, means for using the computer to help locate program errors, and means for communicating with the computer in a language simpler than its own instruction code. These are the general areas encompassed by "automatic programming".

Broadly speaking, we may define automatic programming as the technique by which one program operates upon another program. Let us designate the program that does this operating, whatever its nature, as the "internal program", since it is normally placed in the computer and is for all practical purposes an internal part of the computer. The program that is operated upon we shall term the "external program", since it is supplied from outside the computer as a set of data to be manipulated. The internal programs can often be classified as either interpretive routines or translation routines. Interpretive routines are routines that "interpret", into machine language, instructions and numbers written in a form different from that which the machine employs in its internal structure. The interpreted instruction, which may consist of several instructions in the machine language, is then executed, and the process is repeated for the next instruction of the external program. Of course, the external program must be in the form required for the interpreter routine, but this form can be very different from the machine language as well as much simpler. By contrast with interpreters, translation programs operate upon an entire program in nonmachine language and translate it into a new program in machine language, which is then ready to be run on the computer. The terms "translate" and "interpret" have about the same distinction in programming as they have in their more common usage; "translation" refers to an entire operation, upon a whole book or a whole program, and "interpretation" refers to a running process, handled a little at a time, upon successive sentences in speech or successive steps in a program.

One of the most important applications of translators or interpreters is in helping persons unfamiliar with a machine's complicated instruction code to put their problems onto the machine quickly and easily.[31] Many computer laboratories and companies have devised routines that will accept information written in mathematical notation or even in simple English and operate upon it to produce instructions in machine language. Several examples of such systems have been described in detail in the literature and will not be treated here.[32,33] All ranges of complexity are possible for conversion routines (i.e., interpreters or translators), from simple routines that merely convert decimal input data to binary form on through to routines that convert instructions written for one machine (whether real or hypothetical) into instructions that can be executed by another.

Another technique of automatic programming is the compiling routine.

The function of the compiler is to form a complete machine program from a set of input data equivalent to a flow diagram. The diagram itself, in the present state of the art, is not a suitable form for the input; a set of simple statements describing the diagram and meaningful to the compiler must first be prepared. A primary objective, of course, is to be able to use for these simple statements a set of symbols and connectives that are familiar to the user. Thus some compilers accept mathematical symbols as input, others are coded to operate upon English-language words used in business and commerce, and still others might be devised to accept any particular set of symbols or characters that would be useful for other situations. (Of course, the compiler "recognizes" characters only by being programmed to respond in specified ways to the various binary-code groups that are furnished to it from the input keyboard via cards or tapes or other media.) So far, the compiler appears to have much in common with the translation routine, but it goes beyond the simple translation program by functioning to assign storage locations, call forth required subroutines from storage, prepare subroutine entries and exits, arrange various control transfers, and finally record the complete program, usually on a magnetic tape. Thus compilers are much more versatile than translation programs, translation and sometimes interpretation being simply parts of a compiling routine.

In one method of using the compiler, the program loops are all "straightened out", so that there are no branch points or points of reentry. In this method subroutines are included in open form wherever they are needed in the program. This may save a substantial amount of time in running the program, since it is now unnecessary to perform the subroutine entry and exit each time it is to be used. The price paid for this is the need for large storage space to accommodate the straightened-out program. Thus machines with large magnetic-tape storage capacity are best suited for use with compiling routines. For machines with limited storage capacity the interpretive routine is more appropriate, since only the current step of the external program instead of the entire external program need be stored in addition to the internal program. The price paid for the privilege of using only a small storage capacity is the increased time required for solution, since each instruction of the external program must be reinterpreted each time it is encountered instead of being translated once and for all.

**The tools of automatic programming.** Automatic programming has developed a terminology of its own, which includes such expressions as "pseudo orders", "symbolic addresses", "floating addresses", and many others. A few comments about these may help to clarify to the reader the manner in which internal programs function. To begin with, pseudo orders are orders written in pseudo code, the form of instruction

code that the internal program accepts and converts into machine code. The pseudo code often contains commands that do not exist in the machine code and must, therefore, be made up of many machine instructions collected as a subroutine. For example, the pseudo code might contain the pseudo order SQR, meaning "take the square root." This one instruction in pseudo code is then converted by the internal program into the full group of machine instructions required to carry it out. Thus, one of the conspicuous advantages of the pseudo code is that programs written in it are much shorter than the equivalent machine-coded programs.

The length of words in the pseudo code need not be the same as the word length in machine code. In fact, words need not even be all the same length. Of course, it is possible to construct machines that do not use a fixed internal word length, but the present trend is to incorporate this flexibility, if it is needed, in the internal program rather than in the machine structure. Systems that employ variable word length ordinarily include a space character at the end of each word to indicate to the internal program that subsequent characters constitute another word. In fixed-word-length systems each line of coding may be taken as constituting a single word, and the act of returning the carriage of the input typewriter can generate a symbol recorded on the tape to signify start of the next character.

"Symbolic addresses" or "floating addresses" are another technique for simplifying the programmer's work.[34,35] A very simple form of this, known as block coding, was used in the programming examples in the early part of this chapter, where addresses were followed by a letter designator that could subsequently be given a numerical value and added to the numerical part of the address to give a true machine address. In those examples we tacitly recognized that not all addresses can be assigned until the program has been written in full. Even assigning block addresses may be more work than necessary, since, as Wilkes has pointed out, only a few of the orders in most programs need be referred to by their storage locations, and numbering every order is too stringent a requirement, especially in view of the problem of renumbering that arises when an instruction is to be inserted.

In its most general form the floating address is simply a tag or label for each word that allows the programmer to refer to the word by its label, which does not change, instead of by its storage location, which may not be known and which furthermore is subject to change. The internal program does the bookkeeping of assigning actual machine storage locations to these floating-address quantities. (Of course, the only advantage in using a floating-address label for a word instead of using the word itself is that the label can be much shorter than the

word and therefore easier for the internal program to manipulate and identify.)

A simple example of the floating-address technique is the floating-address input-translation program used with the MIDAC.* This program can accept pseudo orders having floating addresses with the following designations:

| | |
|---|---|
| a00, a01, a02, . . . | a39 |
| b00, . . . | b39 |
| c00, . . . | c39 |
| d00, . . . | d39 |
| e00, . . . | e39 |

Each of these designators can be the initial address in a block of up to 100 words, in which addresses are indicated as 1d00, 2d00, etc.

A typical sequence of input data to the translation program might be as shown in Table 6.12.

**Table 6.12**
**A typical sequence of input data to the floating-address input-translation program of MIDAC**

| Guides used by the translation program | Program and constants to be translated |
|---|---|
| ac43 | |
| fa a01 | c02 c02 e01 ml |
| | e01 c02 1e01 ml |
| | c01 1e01 2e01 ml |
| | 1c01 e01 3e01 ml |
| fa c01 | $1 \times 2^{-44}$ |
| | $25 \times 2^{-44}$ |
| fa c02 | $413 \times 2^{-44}$ |
| fa e01 | |

The groups written to the left of the vertical line are guides followed by the translation program in translating to machine language the pseudo-code instructions and constants to the right of the line. The translation routine first reads in "ac43", which signifies "Assign storage cell number 43 as the location for the next word of the translated program." It then reads in "fa a01", which signifies that the programmer has given the floating address a01 to the instruction word on this line. The translation program keeps a cross index between floating addresses and machine addresses and records for later reference the correspondence between a01

* MIDAC, University of Michigan Digital Automatic Computer, was a three-address computer having characteristics almost identical to those of SEAC, the Bureau of Standards Eastern Automatic Computer. It was a serial binary machine having a word length of 44 bits and sign.

and 43. The second, third, and fourth instructions carry no explicit floating-address designators and are assigned addresses 44, 45, and 46 respectively. The numerical quantities next listed have floating addresses c01 and 1c01 (the latter stated only implicitly), and these are assigned addresses 47 and 48, the next addresses available in sequence. The next quantity, which has floating address c02, is assigned to cell number 49, and the floating addresses e01, 1e01, etc., are assigned the next cells in sequence. Furthermore, once these assignments have been made, the translation program processes the entire sequence of instructions, referring to its cross index in order to replace each floating address in the body of an instruction by the actual cell number just assigned. In addition, the program performs a number of other functions, such as converting numerical quantities to binary form. (Each of the numerical quantities of the program in Table 6.12 would have to carry a designator, omitted here for simplicity, to distinguish it from an instruction word so that the translation routine would process it properly.)

Since the actual cell numbers are not assigned until the external program is processed by the translation program, it is a simple matter to insert an omitted instruction at any point in the program. The omitted instruction is written at the end of the program and given a floating address, for example, d24. Then the instruction following the spot where the omission occurred is labeled with the next address in the block, that is, 1d24, and the translation program can then assign sequential storage cells to the entire program including the added instruction in its proper place.

**Machine aids to error location.**[36,37] Although, ideally, both the programmer and the computing machine should function without error, this ideal seems unlikely to be realized. Most new programs require "debugging" to eliminate their errors, and occasional machine malfunctions must also be corrected. Most computers have some limited facilities built in to aid in this process, such as sets of switches that allow certain key information in the machine to be displayed by lights or printed out, but these facilities are usually inadequate. Furthermore, tying the machine down to manual operating speeds under the control of console switches is extremely wasteful of expensive computer time. As an alternative, a variety of programmed methods have been devised to help locate errors by giving the programmer information about his program, either before, during, or after its run. These programs and others also designed to assist in the actual operation of the computer are known as service routines. In addition, a class of programs known as diagnostic routines gives information about the way in which various parts of the computer itself are functioning.[38]

As an improvement upon mere manual manipulation of console controls to read out key information about the status of the program, some

computers have a built-in "automonitor" facility. When operating in the automonitor mode, the computer performs an instruction and then prints out such information as the contents of the instruction counter and any other indexing element, the instruction just performed, and the result of that instruction. The automonitor may be arranged to process each instruction in this way, or it may perform the print-out only on certain predesignated instructions. The designator, sometimes called a "break-point", is a single bit appended to those instruction words for which the automonitor facility should function. Automonitors can be built in as apparatus or can be inserted as programs. The advantage of the break-point automonitor is that it is selective and provides only the information the programmer wants. Full information about each step in the program can be a tremendous volume of data to sift in searching for an error, but judiciously inserted break points can be very helpful by furnishing the programmer information about branches, indices, and other significant features. Furthermore, break-point automonitoring takes less computer time than automonitoring every instruction, and it is usually desirable to get an inoperative program off the computer as quickly as possible. Study of programming errors is best done by giving the programmer some information about his program that he can analyze at his desk. Tying up the computer for "debugging" is an expensive luxury.

People who work with computers have a whimsical tendency to speak of them in anthropomorphic terms. Machines are often called "giant brains" (although not by those who know them well) and are frequently said to have "memory". In fact, there is even a class of programs known as "post mortems", used, of course, to analyze programs that are "dead". The term "post-mortem" is loosely used to describe any program used in the analysis of another program that fails to operate properly. For example, one such routine used in several computing centers is the "changed-word post-mortem". This routine makes it possible to determine which words of the program have changed during the course of the computation. A study of this information can be very helpful to the programmer in trying to locate an error.

Other helps may be provided by such means as directory print-outs, analyzer routines, correctors, comparators, and others. A directory is a cross index between floating addresses of the external program and actual machine addresses that correspond to them. The information this provides is very useful to the programmer, especially in a computing center that does not have a highly developed system of service routines and in which the programmer is not far removed from the machine and its own internal language. Analyzer routines, another aid, operate upon a program to prepare a list showing each instruction, its location in storage,

and the address of each instruction that refers to it. A corrector is a routine that allows words to be substituted, one for one, for specified words in storage. This routine is better adapted to machines using tape inputs than to those using punched-card inputs. It is not needed for punched-card-input machines, since insertions of this type can be performed rather easily with cards. Finally, the comparator routine compares two programs word by word and records any disagreements between them.

The list of service routines given here is by no means exhaustive, but it is representative of those in use at the major computing centers. Most computing centers are engaged in research efforts of their own to improve programming techniques, and the reader interested in this field will find it stimulating and challenging.

## REFERENCES

1. Hartree, D. R., "Numerical Analysis," Oxford University Press, New York, 1955.
2. Hildebrand, F. B., "Introduction to Numerical Analysis," McGraw-Hill Book Company, Inc., New York, 1956.
3. Householder, Alston S., "Principles of Numerical Analysis," McGraw-Hill Book Company, Inc., New York, 1953.
4. Thomas, W. H., Fundamentals of digital computer programming, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1245–1249.
5. Van der Pol, W. L., A simple electronic digital computer, *Appl. Sci. Research*, sec. B, vol. 2, 1952, p. 367.
6. Wilkes, M. V., D. J. Wheeler, and S. Gill, "The Preparation of Programs for an Electronic Digital Computer," Addison-Wesley Publishing Company, Cambridge, Mass., 1951.
7. Burks, A. W., The logic of programming electronic digital computers, *Ind. Math.*, vol. 1, 1950, pp. 36–52.
8. McCracken, D. D., "Digital Computer Programming," John Wiley & Sons, Inc., New York, 1957.
9. Lubkin, S., Decimal point location in computing machines, *MTAC*, vol. 3, no. 21, January, 1948, pp. 44–50.
10. Brooker, R. A., and D. J. Wheeler, Floating operations on the EDSAC, *MTAC*, vol. 7, no. 41, January, 1953, pp. 37–47.
11. Samelson, K., and F. L. Bauer, Optimale Rechengenauigkeit bei Rechenanlagen mit gleitendem Komma, *Z. angew. Math. u. Phys.*, vol. 4, 1953, pp. 312–316.
12. Goldstine, H. H., and J. von Neumann, "Planning and Coding of Problems for an Electronic Computing Instrument," Institute for Advanced Study, Princeton, N.J., 1947.
13. Alway, G. G., Optimum coding, *Natl. Phys. Lab., Proc. 1953 Symposium on Automatic Digital Computation*, pp. 65–69.
14. "Type 650 Magnetic Drum Data Processing Machine Manual of Operation," International Business Machines Corp., New York, 1955.
15. Perry, D. P., Minimum access programming, *MTAC*, vol. 6, no. 39, July, 1952, pp. 172–182.
16. Bloch, R. M., R. V. D. Campbell, and M. Ellis, The logical design of the Raytheon computer, *MTAC*, vol. 3, no. 24, October, 1948, pp. 286–295.

17. Bloch, R. M., R. V. D. Campbell, and M. Ellis, General considerations for the Raytheon computer, *MTAC*, vol. 3, no. 24, October, 1948, pp. 317–323.
18. Huskey, H. D., Characteristics of the Institute for Numerical Analysis Computer, *MTAC*, vol. 4, no. 30, April, 1950, pp. 103–108.
19. Booth, A. D., The development of A. P. E. (x). C., *MTAC*, vol. 8, no. 46, April, 1954, pp. 98–105.
20. The incorporation of subroutines into a complete problem on the NBS Eastern Automatic Computer, *MTAC*, vol. 4, no. 31, July, 1950, pp. 164–168.
21. Leiner, A. L., Provision for expansion in the SEAC, *MTAC*, vol. 5, no. 36, October, 1951, pp. 232–237.
22. Greig, J., The Circle computer, *MTAC*, vol. 7, no. 44, October, 1953, pp. 249–255.
23. Estrin, G., The electronic computer at the Institute for Advanced Study, *MTAC*, vol. 7, no. 42, April, 1953, pp. 108–114.
24. Buchholz, W., The system design of the IBM type 701 computer, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1262–1275.
25. Carter, W. C., and M. Ellis, A comparison of order structures for automatic digital computers, *J. Operations Research Soc. Am.*, vol. 2, no. 1, 1954, pp. 41–58.
26. Adams, C. W., Developments in programming research, *Proc. Eastern Joint Computer Conf., Nov. 7–9, 1955*, pp. 75–79.
27. Hopper, G. M., and J. M. Mauchly, Influence of programming techniques on the design of computers, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1250–1254.
28. Hopper, G. M., Automatic coding for digital computers, *Conf. on High-speed Computers, Louisiana State Univ., 1955*. Reprints published by Remington-Rand Corporation.
29. Mutch, E. N., and S. Gill, Conversion routines, *Natl. Phys. Lab., Proc. 1953 Symposium on Automatic Digital Computation*, pp. 74–80.
30. Rutishauser, H., Some programming techniques for the ERMETH, *J. ACM*, vol. 2, no. 1, January, 1955, pp 1–4.
31. Perkins, R., EASIAC, a pseudo-computer, *J. ACM*, vol. 3, no. 2, April, 1956, pp. 65–72.
32. A group of seven papers describing PACT 1, a compiler program for use with the IBM type 701 computer, *J. ACM*, vol. 3, no. 4, October, 1956, pp. 265–313.
33. Steel, T. B., Jr., PACT 1A, *J. ACM*, vol. 4, no. 1, January, 1957, pp. 8–11.
34. Rochester, N., Symbolic programming, *IRE Trans. on Electronic Computers*, vol. EC-2, no. 1, March, 1953, pp. 10–15. (*Note:* Tables I and II of this paper are incorrect as they stand and were revised in a subsequently distributed correction sheet. Notwithstanding this, the paper is significant and worth while.)
35. Wilkes, M. V., The use of a "floating address" system for orders in an automatic digital computer, *Proc. Cambridge Phil. Soc.*, vol. 44, 1953, p. 84.
36. Gill, S., The diagnosis of mistakes in programs on the EDSAC, *Proc. Roy. Soc. London*, vol. A206, 1951, pp. 538–554.
37. Gill, S., Getting programmes right, *Natl. Phys. Lab., Proc. 1953 Symposium on Automatic Digital Computation*, pp.80–83.
38. Wheeler, D. J., and J. E. Robertson, Diagnostic programs for the ILLIAC, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1320–1325.

## PROBLEMS

The following programming problems are intended to give the reader some practice is setting up programs and some experience in dealing with a variety of programming situations. Although some of the problems make specific reference to operations

available with the MIC, as listed in the text, any of the problems may be set up for machines with which the reader has had more opportunity to become familiar. In each case it is recommended that alternative methods of programming the problem be studied and that a flow diagram be drawn before the program is written.

**6.1** Using the series expansion

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

program the calculation of sin 0.01 to five-decimal-digit precision. (It may help to recall that, with the alternating series, the last term calculated always exceeds the error caused by omitting all subsequent terms.) It is recommended that this program be done by an iterative procedure that calculates each term from the term before. Generalize your program to calculate values of sin $x$ for $x = 0.01, 0.02, \ldots, 0.50$. Generalize it still further by including scaling necessary to allow $x$ to go up to 1.99.

**6.2** Using series expansions for sin $x$ and cos $x$, program the calculation of tan $x$ to five-decimal-digit precision for $x = 10°, 20°, \ldots, 80°$.

**6.3** Use the LOGICAL TRANSFER, ADD, and SHIFT operations to find the sum of the binary digits of the word in cell 0 of the memory.

**6.4** The MIC word has 30 bits and a sign. Treating the 28 most significant bits as seven groups of 4 bits, each group being a binary code for a decimal integer, program the conversion of such a decimal number to binary form. That is, regard the number (in cell 0, for example) as being

$$a_6 \times 10^6 + a_5 \times 10^5 + a_4 \times 10^4 + a_3 \times 10^3 + a_2 \times 10^2 + a_1 \times 10^1 + a_0$$

*Suggestion:* A simple method of polynomial evaluation on the digital computer is to evaluate the following form:

$$\{[(10a_6 + a_5) \times 10 + a_4] \times 10 + a_3\} \times 10 + a_2 \cdots$$

**6.5** Devise a program to read 50 words from input into storage. Then read them in again, one at a time, checking to see if they were read in correctly the first time. If the two read-ins agree, print out a symbol that indicates this; if they do not, print out both the correct and incorrect words. (Character print-out on the MIC is accomplished by a STORE ACCUMULATOR instruction, the address being that of the typewriter. Code groups that must be in the accumulator for each desired character are listed in Table 9.10.)

**6.6** Program the matrix multiplication $C = A \cdot B$, where $A$ is a matrix of $m$ rows and $n$ columns and $B$ is a matrix of $n$ rows and $p$ columns. *Note:* A matrix is a rectangular, but not necessarily square, array of numbers, for example,

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix}$$

The product of two matrices is a matrix $C$ in which each element $c_{ij}$ is given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

If 4,096 memory cells are available for program and numbers and if $m$, $n$, and $p$ are equal, what is the largest value they could have?

**6.7** Program the division of two numbers by means of the iterative formula for evaluating $1/a$:

$$y_{n+1} = y_n(2 - ay_n)$$

In order for this process to converge, it is necessary for $y_0$ to lie between 0 and $2/a$. (Some computers, in which circuit complexity is kept to a minimum, use this process instead of having a built-in division operation.)

**6.8** The number in cell 100 is to be divided by the number in cell 101, in a machine with radix point fixed at the left. Devise a program to test these two numbers to determine whether an overflow will occur upon division. If it will, program the minimum possible shift to avoid overflow, perform the division, put the result into cell 102, and put the number of binary places that the result is shifted into cell 103. If no overflow will occur, put the result in 102 and zero in cell 103.

**6.9** Devise a program to determine whether the $\alpha$ part of the instruction in cell 050 is an odd or an even number.

**6.10** Five numbers are stored in cells 100 to 104. Construct a program that stores them in order of decreasing magnitude in cells 105 to 109. Store the instructions sequentially, with the first in cell 000. Use storage cells 110 and following for storage of temporaries and any constants needed. Repetitive cycles should be programmed with the base operation.

There are at least two ways to do this sorting:

1. Determine the correct position of each number by determining how many numbers exceed it. 105 plus this number of numbers is then the correct storage cell for the result.
2. Compare $x_j$ with $x_{j+1}$, exchanging their positions if $x_j < x_{j+1}$. Advance $j$ by 1 if no exchange is needed; reset it to 1 after each exchange.

Suggest some other procedures that might also be employed and make estimates of which method

*a.* Uses the smallest amount of machine operating time
*b.* Requires the least storage space in the memory

The complexities in the programming of this simple problem point up the fact that the so-called "general-purpose" computer is really best suited for arithmetical computations and not nearly so well suited for logical computations.

**6.11** The MIC memory has 992 cells. Devise a program that treats the contents of *all* cells as numbers, adds them (ignoring overflow) and puts the result in cell 991. Store the program starting at cell 000.

**6.12** Program the MIC to perform the addition of two numbers without employing any of the arithmetic commands, i.e., $+$, $-$, $\times$, or $\div$. (There are at least two ways in which this can be done by taking full advantage of the LOGICAL TRANSFER operation.)

**6.13** Repeat the preceding problem for subtraction instead of addition.

**6.14** Program the MIC to carry out the operation performed by its JUMP command but without making use of this command. (*Hint:* What other MIC command effects a conditional transfer of control?)

**6.15** Many types of programs are used to test the proper operation of a computer and to exercise particular sections of it to see whether they are functioning properly. Devise an "inchworm" program for memory test, to be stored initially in consecu-

tively numbered cells at the low end of the memory, that executes the following sequence of steps:

1. Transfers its last word from cell $K$ to cell $K + 1$
2. Transfers its next-to-last word from cell $K - 1$ to cell $K$
3. Proceeds in this manner to transfer every one of its words to the next-higher-numbered cell
4. Repeats the whole process until it has again transferred itself one position forward
5. Keeps on going until the whole program has been transferred to the other end of the memory
6. Finally types out the code group "1111 · · · 111" (a full computer word) and halts

(To do this program on the MIC, it is important to know that addresses 0000 to 0991 refer to memory cells. Addresses 0992 to 1001 refer to toggle-switch banks from which information can be read but into which the computer cannot write. Type-out is done by the instruction "STORE ACCUMULATOR in cell 1005".)

Many interesting variants of this problem can be devised. For example, it can be programmed to advance two steps and retreat one. If it is desired only to ensure that each cell is used once, each word can be moved by a distance equal to the full program length at each step. Make estimates of the running time required on the computer for which you do this problem. Comment on the possibility of arranging for the machine to erase its entire memory after completion of the transfer.

**6.16** Write a program for the MIC, to be stored in consecutive cells starting at cell 0000, that will add the contents of *all* even-numbered cells commencing with cell 0000 and going through the entire MIC memory (992 cells). The result is to be stored in cell 1005. For the purposes of this program the machine may be assumed to disregard overflow.

(It is possible to write a program to do this that will occupy no more than four storage cells.)

**6.17** The only JUMP operation in the MIC command list transfers control if the accumulator contains a negative number. Using MIC instructions, program the following types of jump:

*a.* JUMP ON ZERO. Transfers control if accumulator content is zero.

*b.* JUMP ON PLUS. Transfers control if accumulator content is positive ("positive" means $\geq 0$).

*c.* JUMP ON EVEN. Transfers control if accumulator content (regarded as an integer) is an even number.

Assume that the number to be tested is in cell 000. Store any constants you need starting at cell 001, and list what they are and where they are. Put your program in cell 010 and immediately following locations. Cell 100 is the location to which control is to be transferred.

CHAPTER 7

# Number Systems and Codes

## 7.1 Historical introduction[1]

The number concept is so deeply ingrained in our present civilization and culture that we employ it without a thought about its origins or its validity. Yet any parent has an opportunity to observe in his own children some of the awkward fumblings with the idea of number which must be typical of those made by the human race some ten or more thousands of years ago. To abstract from collections of two blocks, two stones, two people, or two fingers the concept of "two" that is characteristic of all of them is a difficult step. In fact some aboriginal languages are known in which this abstraction has not been made at all, and different sets of number words are used for describing different kinds of quantities. (The English language still retains, although without much distinction as to their application, several words that denote 2: pair, brace, couple, team, and others.)

It would appear that man's inherent concept of number extends only to such numbers as 3 or 4 or 5. It is easy to tell at a glance the difference between two sets of objects one having three members and the other four, but to detect the difference between sets of nine and ten members can only be done by arranging one-to-one correspondences or by counting. Both these techniques are historically ancient; notched tally sticks dating back to paleolithic times have been found in Europe. Counting is a somewhat more advanced technique, requiring as it does names for the various numbers and rules for combining these names. The development of counting led to the use of a radix, or base, for the number system, and in different cultures different bases have been used. The radix most often used in ancient times and in present-day tribal cultures is 10, and it is this radix that our present culture employs. The choice of 10 may well stem from finger counting using ten fingers, although many cultures have used other number systems, such as the quinary system (base 5), perhaps based on the fingers of one hand, or the biquinary system, based on the two groups of five fingers, or the vigesimal system, based on the

twenty fingers and toes. The base 60 was used in the Sumerian civilization of about 2000 B.C., and vestiges of it are still present in our use of 360 degrees in a circle, 60 minutes in a degree or an hour, and 60 seconds in a minute.

An important feature of the ancient Sumerian number system is that it was a positional system, like our own, in which the weight to be attached to a symbol was indicated by its position in the sequence. Unfortunately, this positional notation was hampered by lack of a symbol for zero; so a group of digits contained in itself no indication of whether two successive digits had weights differing by just one power of the base or whether they might differ by two or more powers. Lack of a symbol for zero made it necessary to interpret such numbers with reference to the context. Thus 63 might mean $6 \times 60^1 + 3 \times 60^0 = 363$, or $6 \times 60^2 + 3 \times 60^0 = 21{,}603$. Strangely enough, the symbol for zero is a comparatively recent invention, coming from India some time in the early centuries of the Christian era by way of the Arab lands and finally to Europe. Like many small inventions, however, it was of tremendous importance, since it gave to number systems a flexibility and ease of manipulation that removed calculation from the secret domain of ancient priesthoods and made it ultimately available to all of literate mankind.

With these ideas as a background, let us now examine some of the concepts and procedures involved in the representation of numbers in a modern digital computer. Although we have already had a preliminary glance at these matters in Chap. 5, it is desirable now to go into more detail as we approach the problem of machine design. Such questions as the following will concern us:

1. What choices exist for the radix of the number system used in the computer? What set of marks should be employed for the radix $r$ digits, and what representation should be used for them in the computer?
2. How are negative numbers to be represented and manipulated?
3. How are fractions, integers, and mixed numbers to be handled?

After some exploration of these general aspects of number systems, we shall consider in detail several binary modes of representation that might be useful for computers.

## 7.2 Choice of radix

The most widely used number systems are positional systems; that is, the positions of the various digits indicate, by general agreement, the significance to be attached to each digit. This makes it possible to express quantities in much more compact notation than would be necessary if an identifying tag were to be attached to each digit. Thus, by general agreement in the U.S., the notation 4/12/56 signifies the fourth

month (April), the 12th day of that month, and the 56th year of the present century. In Britain, however, the same notation would be taken to indicate the fourth day of the twelfth month (December), 1956. Positional notations are useful, therefore, when there is general agreement about the significance of the various positions. Fortunately, this is the case with the familiar decimal system, and a sequence

$$a_n a_{n-1} a_{n-2} \cdots a_2 a_1 a_0 . a_{-1} a_{-2} \cdots \tag{7.1}$$

is always understood to mean

$$a_n \times 10^n + a_{n-1} \times 10^{n-1} + \cdots + a_2 \times 10^2 + a_1 \times 10 + a_0 + a_{-1} \times 10^{-1} + a_{n-2} \times 10^{n-2} + \cdots \tag{7.2}$$

Note that, if the weights to be attached to each $a_k$ are explicitly stated, then there is no need to write the sequence in any particular order. The sequence

$$a_1 \times 10 + a_n \times 10^n + a_3 \times 10^3 \text{ etc.} \tag{7.3}$$

has exactly the same meaning as the ordered sequence (7.2) if all terms are included. However, if, by generally adopted convention, we always put the sequence in the same order, the weighting factors associated with each digit need not be explicitly stated. In that case it becomes necessary, as we have seen, to include a digit for each position, whether any units of that weight are included in the number or not. Thus $7 \times 10^3 + 3 \times 10^0$ is written in positional notation as 7003 instead of 73, in order to preserve the positional significance of the digits. In writing numbers down manually, we do not ordinarily include zeros to the left of the leftmost nonzero digit in the integral part or to the right of the rightmost nonzero digit in the fractional part. However, most computers have a storage device for each digit position, and, since this device must always be in one of its possible stable states, it always indicates a digit explicitly. Thus the number 546 would be represented in a machine of five-decimal-digit capacity as 00546.

Another convention, to which we have already alluded, is that the radix point divides a number into an integral part given by the digits to its left and a fractional part given by the digits to its right. If the radix point does not appear explicitly, it is assumed to lie immediately to the right of the rightmost digit.

Although 10 is the radix most commonly used in our civilization, there is no fundamental reason, other than the familiarity resulting from long usage, for using 10 rather than some other radix, particularly in the design of a machine, which has no prejudices or habits to influence it. We have already seen that the radix 2 is widely employed in computers because

of the existence of apparatus with two stable states to represent the two digits, and it is by no means impossible that other radices may come to have as much importance in computers. In fact, some computers employ radices of 8, 16, or 32 in their input-output systems.

One possible reason for using a radix $r > 10$ is that a number can be expressed in such a radix with fewer digits than in radix 10. $n$ decimal digits are capable of representing $10^n$ different numbers. To represent these numbers in radix $r$ will require $m$ digits, where

$$r^m \geqq 10^n$$

and

$$m = \left[\!\left[\frac{n}{\log_{10} r}\right]\!\right] \tag{7.4}$$

The symbol $[\![x]\!]$ here means $x$ if $x$ is an integer and the next larger integer otherwise; that is, $[\![x]\!]$ is the smallest integer not smaller than $x$. For instance, if $n = 10$ and $r = 14$, $m = 9$; that is, the information conveyed by ten decimal digits can be conveyed by nine radix 14 digits. This should not be taken to imply, however, that a large radix is necessarily desirable. The difficulty with using a radix $r > 10$ is that $r$ distinct marks must be available in order to write numbers in this form. Thus we must either invent some new marks or borrow some marks ordinarily used for nonnumerical purposes. This latter procedure is the one more often used, and it is common to employ letters of the alphabet.

Some computers that use a base 16 notation have employed as marks the familiar decimal digits 0 to 9, augmented by the lower-case letters *a*, *b*, *c*, *d*, *e*, and *f* to indicate, respectively, the numbers 10 to 15. Some equivalences between decimal notation and base 16 (hexadecimal or sexadecimal) are given below:

| *Decimal* | *Base* 16 |
|---|---|
| 27 | 1*b* |
| 467 | 1*d*3 |
| 511 | 1*ff* |

Although the borrowing of alphabetic characters for numerical purposes is not easy to become accustomed to, the alternative of inventing characters seems even less desirable. It has been proposed that new symbols be devised that bear some similarity to the numbers they represent, but this proposal has never been adopted.

**Radix conversion.** Conversion of numbers from decimal to binary form has already been described in Chap. 5. Let us now generalize the procedure to show how conversion from any base $r_1$ to any other base $r_2$ can be accomplished. Given a number $N_0$ expressed in base $r_1$, we wish

to find the digits $a_j$ that express $N_0$ in base $r_2$, that is,

$$N_0 = a_n r_2^n + a_{n-1} r_2^{n-1} + \cdots + a_1 r_2 + a_0 \tag{7.5}$$

This polynomial may be written in the form

$$N_0 = (\{[(a_n r_2 + a_{n-1}) r_2 + a_{n-2}] r_2 + \cdots) r_2 + a_0 \tag{7.6}$$

This is equivalent to $N_0 = N_1 r_2 + a_0$, where $N_1$ is a polynomial $N_1 = N_2 r_2 + a_1$ and is one degree lower than the polynomial $N_0$. Each $a_j$ is thus seen to be the remainder left after dividing $N_j$ by $r_2$.

As an example, let us convert 6437 (decimal) to radix 6.

$$\begin{aligned}
6{,}437 \div 6 &= 1{,}072 + 5 \text{ remainder} \\
1{,}072 \div 6 &= 178 + 4 \text{ remainder} \\
178 \div 6 &= 29 + 4 \text{ remainder} \\
29 \div 6 &= 4 + 5 \text{ remainder} \\
4 \div 6 &= 0 + 4 \text{ remainder}
\end{aligned}$$

Thus 6437 (decimal) = 45445 (base 6). The reverse conversion can also be done in the same way, but care must be used because arithmetic in base 6 is less familiar. Let us first form an addition table, a multiplication table, and a table of multiplication by tens in base 6 (Table 7.1).

**Table 7.1**
**Base 6 arithmetic tables**

| + | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | 2 | 3 | 4 | 5 | 10 |
| 2 | 2 | 3 | 4 | 5 | 10 | 11 |
| 3 | 3 | 4 | 5 | 10 | 11 | 12 |
| 4 | 4 | 5 | 10 | 11 | 12 | 13 |
| 5 | 5 | 10 | 11 | 12 | 13 | 14 |

| × | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| 2 | 0 | 2 | 4 | 10 | 12 | 14 |
| 3 | 0 | 3 | 10 | 13 | 20 | 23 |
| 4 | 0 | 4 | 12 | 20 | 24 | 32 |
| 5 | 0 | 5 | 14 | 23 | 32 | 41 |

| Base 10 | Base 6 |
|---|---|
| 10 | 14 |
| 20 | 32 |
| 30 | 50 |
| 40 | 104 |
| 50 | 122 |
| 60 | 140 |
| 70 | 154 |
| 80 | 212 |
| 90 | 230 |

We may now proceed to divide 45445 (base 6) repeatedly by 14, the base 6 equivalent of decimal 10. In the first step we find

```
      2551
14/45445
     32
     134
     122
      124
      122
        25
        14
        11 → = 7 (decimal)
```

The second step is

$$
\begin{array}{r}
144 \\
14\overline{)2551} \\
\underline{14\phantom{51}} \\
115\phantom{1} \\
\underline{104\phantom{1}} \\
111 \\
\underline{104} \\
3 \rightarrow 3 \text{ (decimal)}
\end{array}
$$

Then

$$
\begin{array}{r}
6 \\
14\overline{)144} \\
\underline{140} \\
4 \rightarrow 4 \text{ (decimal)}
\end{array}
$$

and

$$
\begin{array}{r}
0 \\
14\overline{)6} \rightarrow 6 \text{ (decimal)}
\end{array}
$$

The foregoing procedure for integers is easily modified for fractions, for, if $N_0$ is a fraction, we may write

$$
\begin{aligned}
N_0 &= a_{-1}r_2^{-1} + a_{-2}r_2^{-2} + \cdots + a_{-n}r_2^{-n} \\
&= r_2^{-1}\{a_{-1} \cdots [a_{-(n-2)} + r_2^{-1}(a_{-(n-1)} + a_{-n}r_2^{-1})]\}
\end{aligned} \tag{7.7}
$$

Multiplying $N_0$ by $r_2$ gives a mixed number with integral part $a_{-1}$ and fractional part $.a_{-2}a_{-3} \cdots a_{-n}$. Successive multiplications of the fractional parts yield the desired digits. As an example, we may convert 0.739 (decimal) to base 6 as follows:

$$
\begin{aligned}
6 \times 0.739 &= 0.434 + 4 \\
6 \times 0.434 &= 0.604 + 2 \\
6 \times 0.604 &= 0.624 + 3 \\
6 \times 0.624 &= 0.744 + 3
\end{aligned}
$$

In this example, the base 6 equivalent is a nonterminating fraction, the first four digits of which are 0.4233. The conversion of this number back to decimal is as follows:

$$
\begin{array}{r}
0.4233 \\
\times\ \ 14 \\
\hline
2\ 5420 \\
4\ 233\phantom{0} \\
\hline
11.2150
\end{array}
\qquad 11 \text{ (base 6)} = 7 \text{ (base 10)}
$$

$$
\begin{array}{r}
0.215 \\
\times\ 14 \\
\hline
1\ 312 \\
2\ 15\phantom{2} \\
\hline
3.502
\end{array}
\qquad 3 \text{ (base 6)} = 3 \text{ (base 10)}
$$

```
 0.502
 ×14
 3 212
 5 02
12.232        12 (base 6) = 8 (base 10)
```

Thus, the first three figures of the decimal equivalent of 0.4233 (base 6) are 0.738. If more figures had been calculated for the base 6 equivalent of 0.739, the result of converting back to base 10 would have been closer to the original number.

### 7.3 Binary representations of numbers in arbitrary radix

For machines that must communicate with a human operator (and, of course, some need not, since they are incorporated in systems where they communicate only with other machines), the use of the decimal system has much to recommend it. However, in the present state of computer technology and in its foreseeable future, all machines, whatever radix they employ, will use a number representation based upon binary symbols. This is so because the simplest, cheapest, and most dependable devices for digit representation are binary elements, and there is no prospect of the early development of comparable "decimal" devices, that is, devices that can produce on a single output lead any one of ten easily distinguishable and stable signals. Thus, whatever the radix adopted for convenience in communicating with the machine, individual digits or groups of digits will have binary representations.

Perhaps the most obvious way to represent a decimal digit with 1's and 0's is to let the representation of the digit be a group of 1's and 0's totaling ten in number and having 1's equal in total number to the digit represented. Thus the decimal number 17 would be represented as

$$\underbrace{0000000001}_{\text{Representation of 1}}\underbrace{0001111111}_{\text{Representation of 7}}$$

In terms of equipment or storage space such a system is highly inefficient, since it employs 20 binary digits to represent only 100 different numbers and the same 20 digits are capable of representing $2^{20}$ numbers, a figure somewhat in excess of 1 million. However, for some applications, the manipulative advantages of this system may outweigh this inefficiency. For example, addition of numbers in this system can be accomplished by a counter that counts the 1's in the least significant position of the addend and then augments this count by counting the 1's in the same position of the augend. A count beyond 10 generates a carry.

To subtract, the subtrahend 1's are counted down from the count of minuend 1's in each position. A count below zero must generate a borrow from the next more significant digit of the minuend and add 10 to the negative result digit by interchanging 0's and 1's in it.

A variation of this system again uses ten binary digits to represent a decimal digit, but only one of them is a 1 and all others are 0. The ten binary-digit positions are weighted to correspond to the ten decimal digits, giving the following form for representation of the decimal digits:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Weights: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Representation of 3: | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Representation of 6: | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Etc. | | | | | | | | | | |

This system was employed in the ENIAC, the first electronic digital computer. A bank of ten flip-flops was used to store one decimal digit, with the one flip-flop that was in the ON state designating the particular decimal digit stored.

Now, if we use binary characters to represent each radix $r$ digit, it is apparent that, although the number of radix $r$ digits required goes down as $r$ goes up, the number of binary digits required per radix $r$ digit goes up, since more marks must be represented. What choice of radix $r$ involves the smallest number of binary digits? If we use for each radix $r$ digit a total of $r$ binary digits, in a method analogous to one of the two types of decimal-digit representation we have just considered, then the number of binary digits required is

$$\text{Required bits} = \left[\!\!\left[\frac{n}{\log_{10} r}\right]\!\!\right] r \tag{7.8}$$

where $n$ is the number of decimal digits to be represented and the brackets have the same meaning as in Eq. (7.4).

The quantity $\log_{10} r/r$ has frequently been suggested as a figure of merit for number systems, and it is easily shown that this quantity has a maximum for $r = \epsilon$.

$$\frac{d}{dr}\left(\frac{\log_{10} r}{r}\right) = \frac{r(1/r)\log_{10} \varepsilon - \log_{10} r}{r^2} = 0$$

$$\log_{10} \varepsilon = \log_{10} r$$

$$r = \varepsilon \tag{7.9}$$

On this basis it may be reasonably argued that a number base of 2 or 3, both close to $\epsilon$, affords the greatest economy in digits. However, to represent the $r$ digits in a radix $r$ system by $r$ binary digits is highly inefficient, and this figure of merit unduly penalizes radices not close to $\epsilon$.

A better way is to use $k$ binary digits, where $k$ is given by

$$2^k \geq r$$
$$k = \llbracket \log_2 r \rrbracket \tag{7.10}$$

Now we have in the representation of $n$ decimal digits a number of binary digits given by

$$\text{Required bits} = \llbracket \log_2 r \rrbracket \left\llbracket \frac{n}{\log_{10} r} \right\rrbracket \tag{7.11}$$

This is not a continuous function, since it takes on only integer values and, furthermore, is defined only for integer values of $r$. Therefore we cannot find its minimum by the usual process of equating its derivative

**Table 7.2**
**Determination of number of binary digits needed for radix $r$ representation of ten decimal digits**

| $r$ | $\llbracket \log_2 r \rrbracket$ | $\log_{10} r$ | $\left\llbracket \frac{10}{\log_{10} r} \right\rrbracket$ | Required bits | $\left\llbracket \frac{10}{\log_{10} r} \right\rrbracket r$ |
|---|---|---|---|---|---|
| 2 | 1 | 0.3010 | 34 | 34 | 68 |
| 3 | 2 | 0.4771 | 21 | 42 | 63 |
| 4 | 2 | 0.6021 | 17 | 34 | 68 |
| 5 | 3 | 0.6990 | 15 | 45 | 75 |
| 6 | 3 | 0.7782 | 13 | 39 | 78 |
| 7 | 3 | 0.8451 | 12 | 36 | 84 |
| 8 | 3 | 0.9031 | 12 | 36 | 96 |
| 9 | 4 | 0.9542 | 11 | 44 | 99 |
| 10 | 4 | 1.0000 | 10 | 40 | 100 |
| 11 | 4 | 1.0414 | 10 | 40 | 110 |
| 12 | 4 | 1.0792 | 10 | 40 | 120 |
| 13 | 4 | 1.1139 | 9 | 36 | 117 |
| 14 | 4 | 1.1461 | 9 | 36 | 126 |
| 15 | 4 | 1.1761 | 9 | 36 | 135 |
| 16 | 4 | 1.2041 | 9 | 36 | 144 |

to zero, but must evaluate the function for a particular $n$ and various values of $r$ to see where the minimum lies. Let us choose $n = 10$. Then the tabulated values that enable us to find the number of binary digits in each case are given in Table 7.2.

The smallest number of binary digits results from using either 2 or 4 as a radix, either choice requiring only 34 bits for the representation of ten decimal digits. However, choosing as the radix the numbers 7, 8, 13, 14, 15, or 16 requires only 36 bits, a small increase over the minimum

of 34. Radix 3 shows up rather badly, since it involves 42 binary digits. It is interesting to observe, however, by comparison with the rightmost column of Table 7.2, how much more efficient it is to use a weighted binary representation of each radix $r$ digit than to use $r$ binary digits in an unweighted scheme. Even with the poorest choice of radix, that is, $r = 5$, only 45 binary digits are required in the weighted binary coding method, which is fewer than the smallest number needed for the best choice of radix ($r = 3$) in the unweighted binary representation. Of course, the maximum theoretical efficiency (i.e., the minimum number of binary digits in the representation) would occur for $r$ an integral power of 2 and $n/\log_{10} r$ an integer. For such a case we would have

$$\begin{aligned} \text{Required bits} &= \log_2 r \frac{n}{\log_{10} r} \\ &= 3.32n \end{aligned} \tag{7.12}$$

The use of radix 5 for the representation of ten decimal digits is an example of inefficient use of digits. Table 7.2 shows that for this choice

$$m = \left[\!\left[\frac{10}{\log_{10} 5}\right]\!\right] = 15 \tag{7.13}$$

and

$$k = [\![\log_2 5]\!] = 3 \tag{7.14}$$

Thus, $mk = 45$, a figure much larger than the minimum of $3.32n = 33.2$ (or 34 when increased to an integer).

As an example of greater efficiency, consider $n = 9$ and $r = 8$. For these choices,

$$m = \left[\!\left[\frac{9}{\log_{10} 8}\right]\!\right] = \left[\!\left[\frac{9}{0.903}\right]\!\right] = [\![9.96]\!] \tag{7.15}$$

$$m = 10$$

and

$$k = [\![\log_2 8]\!]$$

$$k = 3 \tag{7.16}$$

Thus, $mk = 30$ and $3.32n = 29.88$ (or 30 when increased to an integer).

Systems in which the radix $r$ is an integral power of 2 have the useful property that, when each radix $r$ digit of any number in this radix is expressed as a binary number, the resulting sequence of bits is the binary representation of the original number. For example, in radix 8, the decimal number 133 is written 205. The binary representations of these three digits are, respectively, 010, 000, and 101, which, written together, form 010000101, which is the pure binary form for 133.

If a radix $r > 10$ is used, the number in radix $r$ has fewer radix $r$ digits than it had decimal digits originally. This may be an illusory advantage, since the digits must ultimately be put into a binary form. The radix 16

is sometimes used, however, because, if 4 bits each are used for the representation of the ten decimal digits, six of the 16 possible combinations of 4 bits are unused. Transforming to radix 16 allows more efficient use of the available binary digits.

Of course, it is not necessary to treat the entire number at one time when converting to another radix. The number can be broken into sections and each of these converted. The ultimate in this direction is the system in which each digit in the original number is converted to the new radix. Roughly speaking, the more sections into which the number is converted, the larger will be the number of digits in the result. For example, a six-decimal-digit number may be expressed in radix 5 by nine digits. Treating the number as two 3-digit groups requires five digits for each group, or ten altogether. Three 2-digit groups, however, require nine total, and six 1-digit groups require twelve.

## 7.4 Representation of negative numbers

There are two ways of representing negative numbers in the computer. One of these is to express the number as a magnitude with a minus sign attached, and the other is to use complementation. A frequent practice is to use the signed-magnitude form for communicating with the machine, that is, for input, output, and storage, and to use the complement form for addition and subtraction, since the use of complements simplifies the machine by permitting an adder to be used for either addition or subtraction. We shall postpone until Chap. 9, however, consideration of how machine structure and arithmetical processes are affected by the choice of mode of representation of negative numbers, and we shall examine at this point the rules for performing complementation and some illustrations of their application.

If all numbers handled by the machine are fractions, then the radix complement (R.C.) of a negative number is formed by adding 1 to the negative number. (This is the ten's complement in the decimal system, the two's complement in the binary.) If numbers can have $n$ digits to the left of the radix point, the radix complement of a negative number is formed by adding the radix raised to the $n$th power to the negative number. This is equivalent to subtracting the least-significant nonzero digit from the radix and all other more significant digits from the radix minus 1.

In diminished radix complementation (D.R.C.), the negative number is added to a number that is 1 unit less in the least significant position than the radix power used for radix complementation. (The result is the nine's complement in the decimal system, the one's complement in the binary.) This is equivalent to subtracting each digit of the number from the radix minus 1.

Consider the following examples in radix 10:

| *Radix complementation* | *Diminished radix complementation* |
|---|---|
| 1. Fractions: $x = -0.95$ | $x = -0.95$ |
| R.C. of $x = 1 - 0.95$ | D.R.C. of $x = (1 - 0.01) - 0.95$ |
| $= 0.05$ | $= 0.99 - 0.95$ |
| | $= 0.04$ |
| 2. Numbers $> 1$ (in a machine assumed to carry two digits to left of decimal point): | |
| $x = -74$ | $x = -74$ |
| R.C. of $x = 100 - 74$ | D.R.C. of $x = (100 - 1) - 74$ |
| $= 26$ | $= 99 - 74$ |
| | $= 25$ |
| $x = -36.82$ | $x = -36.82$ |
| R.C. of $x = 100 - 36.82$ | D.R.C. of $x = (100 - 0.01) - 36.82$ |
| $= 63.18$ | $= 99.99 - 36.82$ |
| | $= 63.17$ |
| $x = -4$ | $x = -4$ |
| R.C. of $x = 100 - 4$ | D.R.C. of $x = (100 - 1) - 4$ |
| $= 96$ | $= 95$ |

The subtraction of $x_2$ from $x_1$ can be performed by adding its radix complement (or its diminished radix complement) to $x_1$. Since the computer can carry no more than $n$ digits to the left of the radix point, $r^n$ cannot be represented in the machine, and neither can 1 in machines using only fractions. Thus $x_1 - x_2$ is handled by forming instead $x_1 + (r^n - x_2)$. Since $r^n - x_2$ is a positive number smaller than $r^n$, it can be represented in the machine. But, since $r^n$ itself cannot be so represented, the result of forming $x_1 + (r^n - x_2)$ is identical with $x_1 - x_2$. Thus suppose $x_1 = 74$ and $x_2 = 33$ in a two-digit computer. $x_1 - x_2$ is found by adding $x_1$ to $100 - 33$ to give $74 + 67 = 141$. However, the leftmost digit of this result represents an overflow beyond the machine capacity and hence is lost. The remaining digits 41 are the correct result for $74 - 33$. This procedure is equivalent to ignoring the left-hand carry, if it occurs.

In using diminished radix complementation, $x_1 - x_2$ is handled by forming $x_1 + (r^n - 1 - x_2)$ in the case of integers, or $x_1 + (r^n - r^{-m} - x_2)$ in the case of numbers having a fractional part of $m$ digits. The result will be in error by $r^n - 1$ (or $r^n - r^{-m}$); so it is necessary to subtract this quantity from the quantity $x_1 + (r^n - 1 - x_2)$. However, since the machine can carry only $n$ digits ($n + m$ for numbers with fractional parts), the same result can be obtained by adding 1 unit in the least-significant-digit position. An exception occurs if $|x_2| \geq |x_1|$; in this case

the result is $r^n - 1 - (x_2 - x_1)$, which we recognize as being the diminished radix complement of $x_2 - x_1$. For all cases in which $x_1 - x_2$ is a positive number, it will be observed that $x_1 + (r^n - 1 - x_2)$ produces a left carry beyond the machine capacity. The correct result is obtained by adding this carry on the right. Thus, let $x_1 = 74$ and $x_2 = 33$; then $x_1 + (r^n - 1 - x_2) = 74 + (100 - 1 - 33) = 74 + 66 = 140$. The leftmost digit here is a carry beyond the machine capacity, which we should add on the right instead to get 41, the correct result. Again, let $x_1 = 74$ and $x_2 = 83$; then

$$x_1 + (r^n - 1 - x_2) = 74 + (100 - 1 - 83) = 74 + 16 = 90$$

The absence of the carry digit shows this result to be a complement. The correct answer, therefore, is $-9$, the number of which 90 is the diminished radix complement.

The following examples demonstrate the principles for fixed-point binary arithmetic with the binary point at the left.

| *Radix complementation* | | *Diminished radix complementation* | |
|---|---|---|---|
| 1*a*. | $x_1 = .1001$ | 1*b*. | $x_1 = .1001$ |
| | $x_2 = -.0110$ | | $x_2 = -.0110$ |
| | R.C. of $x_2 = .1010$ | | D.R.C. of $x_2 = .1001$ |
| | $x_1 = .1001$ | | $x_1 = .1001$ |
| | $x_1 - x_2 = 1.0011$ | | $x_1 - x_2 = 1.0010$ |
| Neglecting left carry gives .0011 | | Adding carry on right gives .0011 | |
| 2*a*. | $x_1 = .1011$ | 2*b*. | $x_1 = .1011$ |
| | $x_2 = -.1101$ | | $x_2 = -.1101$ |
| | R.C. of $x_2 = .0011$ | | D.R.C. of $x_2 = .0010$ |
| | $x_1 = .1011$ | | $x_1 = .1011$ |
| | $x_1 - x_2 = .1110$, which | | $x_1 - x_2 = .1101$, which |
| is the 2's complement of $-.0010$ | | is the 1's complement of $-.0010$ | |

When a number is placed in the computer, the sign is represented by a digit, 0 for + and $(r - 1)$ for −. Treatment of this sign digit is discussed in Sec. 9.8.

We may note that, in binary numbers, the diminished radix complement is formed by replacing all 1's by 0's and all 0's by 1's, and the radix complement is formed by leaving the least significant 1 unchanged and changing all 1's and 0's lying to the left. Since it treats all digit positions in the same way, the diminished-radix-complement scheme is easily mechanized and is, therefore, preferred by some computer designers. However, it has the disadvantage that it gives rise to two possible representations of the number zero, one being the usual 0.0000 and the other its one's complement, 1.1111. Only the complement form can be pro-

duced by a machine that uses an adder in its arithmetic unit, and only the all-0 form by a machine using a subtractor. For instance, if $x = 0.1001$ is subtracted from itself by adding its one's complement, the result is 1.1111; if it is subtracted directly from itself, the result is 0.0000.

## 7.5 Coded-decimal systems[2,3]

When each separate decimal digit in the original number is given its own representation, the number is then said to be in a coded-decimal form. A tremendous number of such coded forms are available, even if we consider only the binary codes. A minimum of 4 bits is needed to represent any one of the decimal digits 0 to 9, and these may be arranged in 16 combinations, as follows:

| | | | |
|---|---|---|---|
| 0000 | 0100 | 1000 | 1100 |
| 0001 | 0101 | 1001 | 1101 |
| 0010 | 0110 | 1010 | 1110 |
| 0011 | 0111 | 1011 | 1111 |

Any group among these 16 can be selected as the representation of decimal 0, any one of the remaining 15 as the representation of decimal 1, and so on, giving a total of 16!/6! different codes that can be used to represent the ten decimal digits. This modest-looking number is actually 29,059,430,400 when written out, or about $2.9 \times 10^{10}$.

**Table 7.3**
**Decimal digits in the 8,4,2,1 binary code**

| | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |

One may argue that these codes are not really all "different", since many can be derived from others by complementing digits or by rearranging them. If the four digits are designated $x_1$, $x_2$, $x_3$, and $x_4$, we see that there are 24 different sequences ($4 \times 3 \times 2 \times 1$) in which they can be arranged. (For example, in Table 7.3, any one of the four columns of binary digits can be put on the left, any one of the remaining three next, and so on.) Moreover, any or all of them can be complemented, giving (for each of the 24 sequences) 16 possible ways of distributing comple-

mentation symbols. Thus from a given code, $16 \times 24 = 384$ codes can be derived. This reduces the number of fundamentally different codes to $16!/384 \times 6! \cong 7.6 \times 10^7$—which nevertheless affords the thoughtful designer a moderately wide range of choice.

**Weighted codes.** The best known of the binary codes is that in which the successive digits, usually from right to left, represent weights equal to successive powers of 2, and the decimal number equals the sum of the products of these weights by the binary digit corresponding, as shown in Table 7.3.

Such a code is additive, for the sum of the representations of two digits is the representation of their sum. This, and its familiarity, are the principal advantages of this code.

Codes in which successive digits have weights of 1, 2, 4, and 8 are not the only weighted codes possible. If the weights chosen are positive numbers, their sum must be not greater than 15 or less than 9. Moreover, one of them must be 1 and another must be either 1 or 2. The possible positive weights are listed in Table 7.4.

**Table 7.4**
**Possible positive weights for the binary coding of decimal digits**

| | | | |
|---|---|---|---|
| 3 | 3 | 2 | 1 |
| 4 | 2 | 2 | 1 |
| 4 | 3 | 1 | 1 |
| 5 | 2 | 1 | 1 |
| 4 | 3 | 2 | 1 |
| 4 | 4 | 2 | 1 |
| 5 | 2 | 2 | 1 |
| 5 | 3 | 1 | 1 |
| 5 | 3 | 2 | 1 |
| 5 | 4 | 2 | 1 |
| 6 | 2 | 2 | 1 |
| 6 | 3 | 1 | 1 |
| 6 | 3 | 2 | 1 |
| 6 | 4 | 2 | 1 |
| 7 | 3 | 2 | 1 |
| 7 | 4 | 2 | 1 |
| 8 | 4 | 2 | 1 |

If some of the digits are allowed to have negative weights, several more weights are possible. However, it is not necessary that any of the weights be 1, provided there are present positive and negative weights whose sum is 1. Neither is it necessary for the sum of the weights to be as large as 9. Some examples of systems including negative weights are

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 4 | −2 | −1 | 6 | 4 | 3 | −2 | 5 | 2 | 2 | −1 |
| 8 | −4 | −2 | 1 | 4 | 3 | 2 | −1 | 5 | 3 | 2 | −1 |
| 7 | 3 | −2 | 1 | 5 | 4 | −3 | 2 | | | | |
| 7 | 4 | −2 | 1 | 8 | −4 | 3 | 2 | | | | |

An example of a weighted code is the 7,4,2,1 code, in which the digit representations are as shown in Table 7.5. An interesting feature of this code is that only fourteen 1's appear in the table. This code is useful for electrical devices that must be in the ON, or power-consuming, state to represent a 1, since it results in minimum power dissipation in the relays or dynamic flip-flops or similar elements that represent the digits.

**Table 7.5**
**Decimal digits in the 7,4,2,1 code**

| | 7 | 4 | 2 | 1 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 1 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 1 |
| 9 | 1 | 0 | 1 | 0 |

Another weighted code is the 4,3,1,1 system. Digit representations might be as in Table 7.6.

**Table 7.6**
**Decimal digits in the 4,3,1,1 code**

| | 4 | 3 | 1 | 1 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| | or | | | |
| | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 | 1 |
| | or | | | |
| | 0 | 1 | 1 | 0 |
| | or | | | |
| | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 1 | 1 |
| | or | | | |
| | 1 | 0 | 0 | 1 |
| | or | | | |
| | 1 | 0 | 1 | 0 |
| 6 | 1 | 0 | 1 | 1 |
| 7 | 1 | 1 | 0 | 0 |
| 8 | 1 | 1 | 0 | 1 |
| | or | | | |
| | 1 | 1 | 1 | 0 |
| 9 | 1 | 1 | 1 | 1 |

Since there are two ways to represent 1, three for 4, three for 5, and two for 8, there are 36 codes possible with these weights. Since half of these can be obtained from the others by interchange of the last two columns, there are 18 fundamentally different codes possible with these weights.

As another example, consider the 8,4,−2,−1 system shown in Table 7.7. This system has the useful property that the nine's complement of numbers represented in it can be formed by replacing all 1's by 0's and all 0's by 1's.

**Table 7.7**
**Decimal digits in 8,4,−2,−1 code**

| | 8 | 4 | −2 | −1 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 1 | 0 |
| 3 | 0 | 1 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 0 | 1 | 0 |
| 7 | 1 | 0 | 0 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 1 | 1 | 1 |

Several other systems also share this property. Table 7.8 shows a set of choices of representation in the 4,3,1,1 system, which allows nine's complementation by binary-digit complementation.

**Table 7.8**
**A code that yields 9's complementation by bit complementation**

| | 4 | 3 | 1 | 1 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 1 | 1 |
| 6 | 1 | 0 | 1 | 1 |
| 7 | 1 | 1 | 0 | 0 |
| 8 | 1 | 1 | 1 | 0 |
| 9 | 1 | 1 | 1 | 1 |

The only weighted systems that have this property are those in which the sum of the weights is exactly 9. The positive-weight systems for which this sum is 9 are

3 3 2 1
4 2 2 1
4 3 1 1
5 2 1 1

However, not all codes with these weights have this property. A necessary and sufficient condition is that the representations of each digit and of its nine's complement be symmetrically situated with respect to the center of the table of 16 combinations of 4 binary digits, as in the example in Table 7.9. From among the positive-weight systems whose sum is 9, a total of 14 codes can be chosen that allow nine's complementation by complementation of the binary digits.

**Table 7.9**
**An example of symmetry allowing nine's complementation by bit complementation**

| | 4 2 2 1 | |
|---|---|---|
| 0 | 0 0 0 0 | |
| 1 | 0 0 0 1 | |
| | 0 0 1 0 | |
| 3 | 0 0 1 1 | |
| 2 | 0 1 0 0 | |
| | 0 1 0 1 | |
| | 0 1 1 0 | |
| 5 | 0 1 1 1 | |
| | · · · · · · | Axis of symmetry |
| 4 | 1 0 0 0 | |
| | 1 0 0 1 | |
| | 1 0 1 0 | |
| 7 | 1 0 1 1 | |
| 6 | 1 1 0 0 | |
| | 1 1 0 1 | |
| 8 | 1 1 1 0 | |
| 9 | 1 1 1 1 | |

**Excess-3 code, a modified weighted system.** If the decimal digits 0 to 9 are represented, in that order, by the middle ten entries in the table of 16 combinations of 4 binary digits, the resulting code is one called the "excess-3 code", since each decimal digit is represented by a binary group whose value (in an 8,4,2,1 system) is 3 larger than the decimal digit. The code groups are shown in Table 7.10. This is another

**Table 7.10**
**Excess-3 code**

| | |
|---|---|
| 0 | 0 0 1 1 |
| 1 | 0 1 0 0 |
| 2 | 0 1 0 1 |
| 3 | 0 1 1 0 |
| 4 | 0 1 1 1 |
| 5 | 1 0 0 0 |
| 6 | 1 0 0 1 |
| 7 | 1 0 1 0 |
| 8 | 1 0 1 1 |
| 9 | 1 1 0 0 |

system in which nine's complementation is performed by interchanging 1's and 0's. For use of this system in arithmetical operations, however, one precaution must be observed: the excess-3 quantity associated with each digit makes a correction necessary. For instance, in adding two excess-3 numbers, an excess-6 result is produced, from which 3 must be subtracted to give the desired excess-3 result. Similarly, in subtraction, 3 must be added to the excess-0 result to give an excess-3 number. Addition and subtraction are performed just as in the conventional 8,4,2,1 system, except that a carry produced in the fourth binary-digit position always indicates a carry into the next decimal-digit position.

The following examples demonstrate arithmetical operations in the excess-3 system:

1.
```
   1 → 0100
  +3 → 0110
  ---   ----
   4    1010
        -11 ← correction
        ----
        0111 ← excess-3 code for 4
```

2.
```
   6 → 1001
  +7 → 1010
  ---   -----
  13   10011
       ↑
       └──── carry
```

A carry digit that occurs in excess-3 addition can be regarded as having a weight of 16. If 10 of these 16 units are added as a carry in the next decimal-digit column and 6 are simply dropped, then the sum is deficient by 3 units instead of being in excess by 3. Thus, to return to excess-3 code when a carry occurs, we augment the next decimal-digit position by 0001 (or, if there were no digits there at all, by 0100), and the result in the position producing the carry by 0011. Continuing example 2, we get

```
          0011 ← result, neglecting carry
  +0100   0011 ← correction
  -----   ----
   0100   0110 ← excess-3 code for 13
```

3.
```
  18 →    0100 1011
 -11 → -0100 0100
  ---   -----  ----
   7     0000 0111
              +11 ← correction
         ---------
         0000 1010 ← excess-3 code for 7
```

In subtraction, 3 must be added to return to excess-3 code.

**A negative-radix code.** A code in which negative numbers as well as positive numbers can be represented is obtained by using as a radix

the number $-2$. The columnar weights and the digit representations are as shown in Table 7.11. The 16 combinations now represent 16 decimal numbers from $-10$ to $+5$ inclusive. In order to represent all the decimal digits from 0 to 9, it is necessary to add a fifth binary digit, having a weight of $+16$.

**Table 7.11**
**A negative-radix code**

| | $(-2)^3$ | $(-2)^2$ | $(-2)^1$ | $(-2)^0$ |
|---|---|---|---|---|
| | $-8$ | 4 | $-2$ | 1 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| $-2$ | 0 | 0 | 1 | 0 |
| $-1$ | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 2 | 0 | 1 | 1 | 0 |
| 3 | 0 | 1 | 1 | 1 |
| $-8$ | 1 | 0 | 0 | 0 |
| $-7$ | 1 | 0 | 0 | 1 |
| $-10$ | 1 | 0 | 1 | 0 |
| $-9$ | 1 | 0 | 1 | 1 |
| $-4$ | 1 | 1 | 0 | 0 |
| $-3$ | 1 | 1 | 0 | 1 |
| $-6$ | 1 | 1 | 1 | 0 |
| $-5$ | 1 | 1 | 1 | 1 |

In performing arithmetic in this system, special provision must be made for handling carry digits. Evidently a carry generated in the $j$th digit position will cancel a 1, if one is present, in the $(j + 1)$st position. If neither addend nor augend has a 1 in the $(j + 1)$st position, a carry is added into both the $(j + 1)$st and the $(j + 2)$nd positions. This procedure obtains because each weight is one-half the sum of the two weights to its left.

Addition of negative numbers obeys the same rules, and subtraction is handled in the same way if the subtrahend is represented as a negative number and then added. However, if a positive number is to be subtracted, subtraction is performed in the usual way (see Sec. 9.3, Subtractors) except that the borrow digit required in the $j$th position is added to the $(j + 1)$st position instead of being subtracted from it.

Some examples follow:

1.
$$\begin{array}{rl} 2 \rightarrow & 0110 \\ \underline{+3} \rightarrow & \underline{0111} \\ 5 \rightarrow & 0101 \end{array}$$

2.
$$\begin{array}{rrl} 5 \rightarrow & 0101 & \\ \underline{+(-3)} \rightarrow & \underline{1101} & \\ 2 & 11 & \leftarrow \text{carry digits} \\ & \underline{0000} & \leftarrow \text{sum digits} \\ 2 \leftarrow & 0110 & \leftarrow \text{result} \end{array}$$

3.
$$\begin{array}{rcrl} 5 & \rightarrow & 0101 & \\ -(+3) & \rightarrow & -0111 & \\ \hline 2 & & 0100 & \leftarrow \text{borrow digits} \\ & & 0010 & \leftarrow \text{difference digits} \\ \hline 2 & \leftarrow & 0110 & \leftarrow \text{result} \end{array}$$

**The reflected code.**[4] For analog-to-digital conversion, it is desirable to employ a binary code in which the representations of successive decimal digits differ by only 1 binary digit. This is so because it is not possible to make two or more digit indicators change state in exact synchronism, and the digital reading of a device in which more than 1 bit

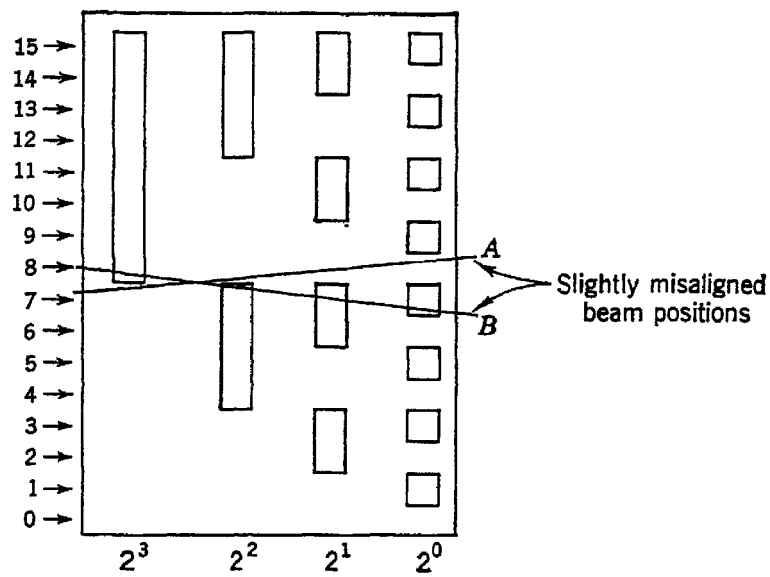



Fig. 7.1 A binary coding mask

changes at a time will be in error during the transition. For example, one analog-to-digital converter uses a beam of electrons deflected vertically by an amount proportional to the analog quantity to be digitalized. This beam strikes a plate having holes arranged in binary sequence, and the presence or absence of beam current at four collector plates behind this coding mask forms a binary number to represent the quantity (see Fig. 7.1).

In the conventional 8,4,2,1 code, all four digits change as the beam moves from 7 to 8. A beam misaligned as at $A$ will momentarily miss all holes as it is deflected upward and give an erroneous output of 0, while the beam at $B$ will momentarily strike holes in all four columns and give an erroneous output of 15. Thus the range of error can be as great as the entire range of the converter when the 8,4,2,1 code is used. However, with a code in which only one digit changes and in which the digits are not weighted, the maximum error is $\frac{1}{16}$ of the total range,

for 4 binary digits in the output, or, for an $n$-digit code, $1/2^n$ of the total range.

The reflected code and the 8,4,2,1 code are shown in Table 7.12 for comparison. In each column of the 8,4,2,1 code the entries alternate

**Table 7.12**
**8,4,2,1 code and reflected code**

| 8 4 2 1 | | Reflected |
|---|---|---|
| 0 0 0 0 | 0 | 0 0 0 0 |
| 0 0 0 1 | 1 | 0 0 0 1 |
| 0 0 1 0 | 2 | 0 0 1 1 |
| 0 0 1 1 | 3 | 0 0 1 0 |
| 0 1 0 0 | 4 | 0 1 1 0 |
| 0 1 0 1 | 5 | 0 1 1 1 |
| 0 1 1 0 | 6 | 0 1 0 1 |
| 0 1 1 1 | 7 | 0 1 0 0 |
| 1 0 0 0 | 8 | 1 1 0 0 |
| 1 0 0 1 | 9 | 1 1 0 1 |
| 1 0 1 0 | 10 | 1 1 1 1 |
| 1 0 1 1 | 11 | 1 1 1 0 |
| 1 1 0 0 | 12 | 1 0 1 0 |
| 1 1 0 1 | 13 | 1 0 1 1 |
| 1 1 1 0 | 14 | 1 0 0 1 |
| 1 1 1 1 | 15 | 1 0 0 0 |

between 0 and 1 in groups of one for the first column, two for the second, four for the next, and so on. In the reflected code the entries in each column are symmetrical with respect to the mid-point of each group of four in the first column, eight in the second, and so on.

The translations between these codes may be formalized as follows:

Let $C_p$ = $p$th digit of the conventional binary number, i.e., the binary number in which successive digits have weights that are successive integral powers of 2

$R_p$ = $p$th digit of the reflected binary number

$C = C_nC_{n-1}C_{n-2} \cdots C_2C_1C_0$ = conventional binary number

$R = R_nR_{n-1}R_{n-2} \cdots R_2R_1R_0$ = reflected binary number

Then,

$$\left.\begin{aligned} C_n &= R_n \\ C_{n-1} &= R_n + R_{n-1} \\ C_{n-2} &= R_n + R_{n-1} + R_{n-2} \\ &\cdots\cdots\cdots\cdots \end{aligned}\right\} \rightarrow C_p = \sum_{j=p}^{n} R_j \tag{7.17}$$

and

$$\left.\begin{aligned} R_n &= C_n \\ R_{n-1} &= C_n + C_{n-1} \\ R_{n-2} &= C_{n-1} + C_{n-2} \\ &\cdots\cdots\cdots\cdots \end{aligned}\right\} \rightarrow R_p = C_p + C_{p+1} \tag{7.18}$$

where addition is defined as binary addition without carry; that is,

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 + 0 = 1 \\ 1 + 1 &= 0 \end{aligned} \tag{7.19}$$

## 7.6 Use of redundancy to avoid errors

It is well known that redundancy in a message makes possible a reduction in the number of errors that occur in its transmission or reception. Everyone has had the experience of having to repeat his words, sometimes several times, to a listener who could not understand them the first time, perhaps because of high noise level. The transmission of messages in the presence of noise, the measurement of redundancy in messages, and the recovery of information contaminated by noise or other errors are subjects encompassed by the new field of information theory, which was originally put on a sound mathematical basis by C. E. Shannon. A simple summary of the subject is not possible, but one of the key points of information theory is the conclusion that it is theoretically possible to reduce the number of errors in any message to as low a level as desired by inclusion of sufficient redundancy. We shall not attempt an information-theoretic approach to the subject here, but we shall present some redundant codes and consider their use in reduction of errors.

A simple example of the use of redundancy can be made of any code using 4 binary digits to represent one of ten decimal digits. Four bits can represent 16 different things, and in any such code used to represent the ten decimal digits there are six nonvalid combinations. Appearance of one of these six is, therefore, evidence of one or more errors. However, with this method errors that transform a valid combination into another valid combination go undetected. It can easily be seen by examination of the code groups in the 8,4,2,1 system that of the 40 possible errors only ten will produce nonvalid combinations, so that reliance upon this method alone would detect only 25 per cent of the single errors that might occur, if all errors are equally probable.[5]

A more sophisticated system using 5 binary digits in such a way that any single error produces a nonvalid combination is the 2-out-of-5 code. Each decimal digit is represented by 5 binary digits, two and only two of which are 1. The number of such combinations is

$$\binom{5}{2} = \frac{5!}{(5-2)! \times 2!} = 10$$

A possible assignment of code groups is shown in Table 7.13. Many

other arrangements are possible, of course, but we shall shortly see an advantage to this one.

In a 2-out-of-5 code a single error introduces either a 1 in place of a 0 or a 0 in place of a 1. In either case the resulting digit group no longer has two and only two 1's, and logical circuits can easily be designed to detect this condition. Two errors, however, may go undetected, since, if both a 1 and a 0 change, there are still two 1's in the resulting group.

**Table 7.13**
**A 2-out-of-5 code**

| | |
|---|---|
| 0 | 01100 |
| 1 | 10001 |
| 2 | 10010 |
| 3 | 00011 |
| 4 | 10100 |
| 5 | 00101 |
| 6 | 00110 |
| 7 | 11000 |
| 8 | 01001 |
| 9 | 01010 |

Another code that detects a single error is the biquinary or 2-out-of-7 code. Here 7 binary digits are used, and are treated as a group of two and a group of five. In each group one and only one digit must be a 1. This can be considered a form of weighted code. The weights and the code groups are shown in Table 7.14.

**Table 7.14**
**A biquinary code**

| | 5 0 | 4 3 2 1 0 |
|---|---|---|
| 0 | 0 1 | 0 0 0 0 1 |
| 1 | 0 1 | 0 0 0 1 0 |
| 2 | 0 1 | 0 0 1 0 0 |
| 3 | 0 1 | 0 1 0 0 0 |
| 4 | 0 1 | 1 0 0 0 0 |
| 5 | 1 0 | 0 0 0 0 1 |
| 6 | 1 0 | 0 0 0 1 0 |
| 7 | 1 0 | 0 0 1 0 0 |
| 8 | 1 0 | 0 1 0 0 0 |
| 9 | 1 0 | 1 0 0 0 0 |

This system of number representation was used in the Bell Telephone Laboratories relay computer.

**Parity checks.** One widely used application of redundancy is the addition of an extra digit to any binary group, the extra digit being a 1 or a 0, as required, to make the total number of 1's in the augmented group always an even number (or always an odd number). This check digit establishes the parity of the binary group, and, if the parity (that is,

the evenness of the totality of 1's) fails, then an error is indicated. Evidently two errors go undetected in this system, but if the probability of two errors is small enough, this simple parity check is satisfactory.

As an example, consider the addition of a parity-check digit to the digits in a 7,4,2,1 code (Table 7.15).

**Table 7.15**
**7,4,2,1 code with parity-check digit**

| | Check | 7 4 2 1 |
|---|---|---|
| 0 | 0 | 0 0 0 0 |
| 1 | 1 | 0 0 0 1 |
| 2 | 1 | 0 0 1 0 |
| 3 | 0 | 0 0 1 1 |
| 4 | 1 | 0 1 0 0 |
| 5 | 0 | 0 1 0 1 |
| 6 | 0 | 0 1 1 0 |
| 7 | 1 | 1 0 0 0 |
| 8 | 0 | 1 0 0 1 |
| 9 | 0 | 1 0 1 0 |

With the exception of the code for zero, these groups are the same as those listed earlier for the 2-out-of-5 code, which allows us to treat the 2-out-of-5 code as a weighted code with a parity check, if suitable provisions are made for handling the group for zero.

One important problem in the use of the parity check is the determination of the number of digits to which the check should apply. Although the simple parity check will detect a single error in a group of digits of any size, the probability of a double error goes up as the length of the checked group increases, and there is an optimum length of checked group for any particular probability distribution and error limit.

By means of cross parity checks, a single error occurring among several groups of digits can be not only detected but also located, and, once having been located, can be corrected by being changed (i.e., from 1 to 0 or from 0 to 1). This is done by applying parity checks to both the rows and the columns of an array of binary digits. As an example, the representations (in the 8,4,2,1 system) of the digits 78465 are shown one below the other, with check digits for rows and for columns:

row-check digits

10111 ↔ 7
11000 ↔ 8
10100 ↔ 4
00110 ↔ 6
00101 ↔ 5
11000 ← column-check digits

If any single digit in this array is changed, two parity checks will fail, one for the row and one for the column in which the error occurred. This of course makes it possible to locate and, therefore, to correct the error.

**The Hamming code.**[6] The next step in multiple parity checks was taken by R. W. Hamming of Bell Telephone Laboratories, who devised a simpler system than the one we have just described for correcting single errors and who also developed methods for detecting and for correcting multiple errors. In the Hamming code, check digits are distributed throughout the message group in such a way as to provide parity checks on various digit positions. To correct an error, the parity checks are made in order, a successful check being designated by a 0 and a failure by a 1. The resulting binary number formed by writing these designators from right to left is the position of the digit in error. If this "checking number" is zero, then there is no error.

We can form such a code by noting first that any digit position whose binary representation has a 1 on the right must cause the first parity check to fail if it contains an error. Such positions are the following:

| *Binary representation* | *Decimal number* | |
|---|---|---|
| 1 | 1 | Positions to which first parity check applies |
| 11 | 3 | |
| 101 | 5 | |
| 111 | 7 | |
| 1001 | 9 | |
| 1011 | 11 | |
| 1101 | 13 | |
| Etc. | ⋮ | |

The second check must apply to positions having a 1 in the second digit of their binary representations:

| *Binary representation* | *Decimal number* | |
|---|---|---|
| 10 | 2 | Positions to which second parity check applies |
| 11 | 3 | |
| 110 | 6 | |
| 111 | 7 | |
| 1010 | 10 | |
| 1011 | 11 | |
| Etc. | ⋮ | |

Thus we see that the

First check digit checks positions 1, 3, 5, 7, 9, 11, 13, 15, 17, . . . .

Second check digit checks positions 2, 3, 6, 7, 10, 11, 14, 15, 18, . . . .

Third check digit checks positions 4, 5, 6, 7, 12, 13, 14, 15, 20, . . . .
Fourth check digit checks positions 8, 9, 10, 11, 12, 13, 14, 15, 24, . . . .
Etc.

If positions 1, 2, 4, 8, etc., are used for the check digits, the check digits are then independent of one another and can be very easily determined.

Let us construct a Hamming code with three check digits. With three digits in our checking number we can specify any one of seven digit positions in addition to the all-zero setting, which indicates no error. Thus, three check digits can check at most a total of seven digits, and four of the seven are then available for the message. The decimal number 10 would be represented as follows:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Digit position: | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Use: | Check | Check | Message | Check | Message | Message | Message |
| Digits: | 1 | 0 | 1 | 1 | 0 | 1 | 0 |

Suppose an error changes this group to 1010010. The first parity check is satisfied, and the second is satisfied, but the third is not. The resulting check number, therefore, is 100, the binary representation of 4, which is the number of the position in error. The fourth digit is now changed to correct the error.

A double error in this code causes some of the parity checks to fail, but, given a double error, changing the digit specified by the check number will certainly not eliminate all errors and may even increase the number of errors, even though the "corrected" group may pass the checking test. For example, if the digit group 0010101110, which contains check digits in positions 1, 2, 4, and 8, is altered by a double error to 0001101110, the check number is 111 (decimal 7). Changing the digit in the seventh position gives 0001100110, for which the check number is found to be zero. Thus, not only has the double error gone undetected, but another has been introduced by the checking. To detect the presence of a double error, we may add a parity check to the Hamming-coded group. This check will always fail in the presence of a single error, and the other checks will give the error location as before. However, in the presence of a double error this check will be satisfied and the other checks will show an error but will not specify the error positions correctly.

**Arithmetic checks.** The redundant codes of the previous sections are useful for checking transfers of data from one point to another within a digital system. For instance, if a number is stored in the computer memory and is subsequently read back, a parity check can give a reasonably good test of whether the complete transfer was effected without error. However, parity checks are not invariant under arithmetical operations, as the reader may easily verify, and more elaborate means are necessary if arithmetic is to be checked as well as transfers.

A process that can be used for checking addition is known, in its decimal form, as "casting out nines". The method consists in adding the digits of the addend, retaining only the part of the sum that exceeds a multiple of 9, adding the augend digits in the same way, adding these two result digits in the same way, and comparing the result with that obtained by summing the digits of the sum of addend and augend, again casting out the largest multiple of 9. For example,

| | |
|---|---|
| 2,794 | Sum of digits after casting out nines = 4 |
| 617 | Sum of digits after casting out nines = 5 |
| 3,411 | Sum = 0 |
| | (after casting out nines) |

Sum of digits of 3,411 after casting out nines = 0

In most computers, all the more complex operations of arithmetic are performed by repeated additions in combination with operations of shifting and complementation. Thus, a check on the operation of addition is, in effect, a check on most arithmetical operations. Of course, another way to check the operation of any part of the computer is to duplicate that part of the system and to require that the duplicated portions be in agreement before the next operation is begun. Whether it is better to duplicate equipment or to provide checking equipment is a matter that must be decided in each instance by such engineering considerations as the relative cost and reliability of the two methods.

It can be shown that the method of casting out nines can be used for numbers expressed in any radix $r \neq 2$. Thus it might be termed "casting out $(r - 1)$'s". For example, in a computer built by the Raytheon Corporation, 35-digit binary numbers are treated as seven groups of 5 binary digits, each such group being equivalent to one radix 32 digit. Before an arithmetical operation on such a number, a 5-bit check group is formed by the process of casting out 31's, and the check is performed in the same way as with casting out nines.

It should be noted that the method of casting out $(r - 1)$'s does not detect all errors. Any error or combination of errors that has the effect of changing the sum of the digits by a whole number of $(r - 1)$'s will still pass the check. For example, in casting out nines, a single error that changes a 0 to a 9 or a 9 to a 0 remains undetected by the check, and so does a double error that interchanges two digits.

A more elaborate system of checks has been proposed by H. L. Garner.[7,8] In this system each number carries with it a group of check digits that are the remainders after division of the number by 2, 3, 5, 7, 11, 13, and successive relatively prime numbers. Only enough check digits are needed to ensure that the product of the prime numbers used

shall exceed the largest number to be represented. In fact, once the check digits have been formed in this way, they constitute a unique representation of the number, and the number digits themselves can be dispensed with. Addition, subtraction, and multiplication of numbers can easily be performed by carrying out these same operations on corresponding digits of the check-digit groups. Division and comparison, however, are not so readily executed.

## REFERENCES

*Items Referred to by Number:*

1. Dantzig, Tobias, "Number, the Language of Science," 4th ed., The Macmillan Company, New York, 1954; also Doubleday Anchor Books, Garden City, N.Y., 1956.
2. White, G. S., Coded decimal number systems for digital computers, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1450–1452.
3. Staff of the Harvard University Computation Laboratory, "Synthesis of Electronic Computing and Control Circuits," Harvard University Press, Cambridge, Mass., 1951, pp. 145–157.
4. Foss, F. A., Use of a reflected code in digital control systems, *IRE Trans. on Electronic Computers*, vol. EC-3, no. 4, December, 1954, pp. 1–6.
5. Kautz, W. H., Optimized data encoding for digital computers, *1954 IRE Conv. Record*, pt. 4, pp. 47–57.
6. Hamming, R. W., Error detecting and error correcting codes, *Bell System Tech. J.*, vol. 29, no. 2, April, 1950, pp. 147–160.
7. Garner, H. L., Generalized parity checking, *IRE Trans. on Electronic Computers*, vol. EC-7, no. 3, September, 1958, pp. 207–213.
8. Garner, H. L., The residue number system, *IRE Trans. on Electronic Computers*, vol. EC-8, no. 2, June, 1959, pp. 140–147.

*Supplementary References:*

Brooks, F. P., Jr., Multi-case binary codes for non-uniform character distribution, *IRE Natl. Conv. Record, March, 1957*, vol. 5, pt. 2, pp. 63–68.

Gilbert, E. N., and E. F. Moore, Variable-length binary encodings, *Bell System Tech. J.*, vol. 38, no. 4, July, 1959, pp. 933–967.

Huffman, D. A., A method for the construction of minimum-redundancy codes, *Proc. IRE*, vol. 40, no. 9, September, 1952, pp. 1098–1101.

Lee, C. Y., Some properties of nonbinary error-correcting codes, *IRE Trans. on Inform. Theory*, vol. IT-4, no. 2, June, 1958, pp. 77–82.

Sacks, G. E., Multiple error correction by means of parity checks, *IRE Trans. on Inform. Theory*, vol. IT-4, no. 4, December, 1958, pp. 145–147.

## PROBLEMS

**7.1** Express the following decimal numbers in bases 16, 7, and 2. (*a*) 347, (*b*) 916, (*c*) 512. Use *a*, *b*, *c*, *d*, *e*, and *f* as the base 16 digits corresponding respectively to 10 to 15.

**7.2** Express $\pi$ in octal form to five significant figures. Also express $\pi$ in binary form to 14 figures, and show that the same binary form is obtained if each octal digit is replaced by its binary equivalent.

**7.3** Perform $A \cdot B$ and $A/B$, where $A$ and $B$ are

(*a*) $\left.\begin{array}{l} A = 176 \\ B = 24 \end{array}\right\}$ both octal numbers

(*b*) $\left.\begin{array}{l} A = 431 \\ B = 132 \end{array}\right\}$ both radix 5 numbers

(It may help to prepare addition and multiplication tables for the two radices before carrying out the indicated operations.)

**7.4** *a.* A 10-bit group, which contains error-locating digits of the Hamming code, is received in the following form:

0010100110

Determine whether the group has been correctly received and, if it has not, determine the correct group. Assume that at most a single error has occurred.

*b.* Express each of the excess-3 binary-coded decimal digits in a form that includes Hamming check digits.

**7.5** Encode each of the decimal digits 0 to 9 in 4-bit groups having the following weights:

| | |
|---|---|
| *a.* 8, 4, −2, −1 | *f.* 4, 3, 2, −1 |
| *b.* 8, −4, −2, 1 | *g.* 5, 4, −3, 2 |
| *c.* 7, 3, −2, 1 | *h.* 8, −4, 3, 2 |
| *d.* 7, 4, −2, 1 | *i.* 5, 2, 2, −1 |
| *e.* 6, 4, 3, −2 | *j.* 5, 3, 2, −1 |

Which of these sets of weights makes possible a code that yields nine's complements when 0's and 1's are exchanged?

Find another set of weights, including at least one negative weight, in terms of which the decimal digits can be represented by 4-bit groups.

**7.6** The code shown in Table 7.5 has only fourteen 1's. Show that no other code can have fewer than fourteen 1's. How many codes can have exactly fourteen 1's?

**7.7** Derive Eqs. (7.17) and (7.18).

**7.8** Prove that the following procedure converts binary integers to decimal integers, where the binary integer is $x_n x_{n-1} x_{n-2} \cdots x_1 x_0$ and $j$ is a variable index. Modify this procedure to apply to mixed binary numbers of the form $x_n x_{n-1} \cdots x_1 x_0 . x_{-1} \cdots x_{-m}$.

This procedure is an easily remembered method for binary-to-decimal conversion.

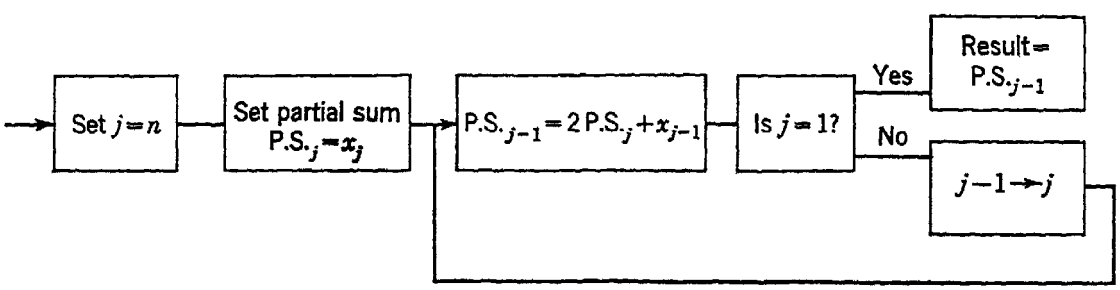



Fig. P7.8

CHAPTER 8

# Mathematical Logic and Switching Networks

## 8.1 Introduction[1–6]

That mathematics is a logical science is a fact that the average individual never questions. The assertion, however, that logic is a mathematical science is a new idea to many people. It is even a relatively new idea among mathematicians, since a large part of the development of mathematical logic has taken place in a little more than the last hundred years. Mathematical logic, or symbolic logic, as it is often called, is a means of using symbols to represent propositions (i.e., statements or assertions) and the relations among these propositions. It enables us to make systematic manipulations of complexes of statements and connectives in order to determine the implications of these complexes or the conditions determining their truth or falsity. Also, in the words of Alfred North Whitehead and Bertrand Russell, whose *Principia Mathematica* is the major work in this field, "The adaptation of the rules of the symbolism to the processes of deduction aids the intuition in regions too abstract for the imagination readily to present to the mind the true relation between the ideas employed."

Symbolic logic is of fundamental importance in digital computation, since the digital computer is essentially a "logic machine", which can perform in a very short time a vast number of simple logical operations upon complexes of propositions and determine the proposition implied by these complexes. Here the propositions are statements about the values of numbers, and the logical operations are the ordinary operations of arithmetic. The proposition implied is the numerical result that we seek. In a sense, the digital computer is actually an analog machine, since the numbers and the arithmetical operations are analogs, respectively, of propositions and of logical connectives. Both the designer and the user of the digital computer are, in effect, logicians, and a knowledge of mathematical logic is a virtual necessity to both if they are to realize their maximum effectiveness.

## 8.2 The logic of classes

The fundamental concepts of the logic of classes provide a convenient starting point for an examination of the techniques of symbolic logic. The logic of classes is concerned with aggregates of elements that have at least one characteristic in common. In fact, the common characteristic (or characteristics) constitutes a definition of the extent and composition of the class. Classes may have elements in common, and these common elements may be considered to constitute other classes. The logic of classes is a systematization of the complex linguistic statements that would be required to indicate the interrelations of classes that have various characteristics extending over various sets of their elements.

We define the following symbols:

| | |
|---|---|
| $x \in A$ | $x$ is a member of the class $A$; that is, $x$ is one of the elements having those characteristics which are required for membership in the class $A$. |
| $A \leq B$ | The class $A$ is included in the class $B$; i.e., every element of $A$ is also a member of $B$. |
| $A \cap B$ | The intersection of $A$ and $B$, i.e., the class of all elements that are members *both* of $A$ and of $B$. The symbol $\cap$ is called "cap". |
| $A \cup B$ | The union of $A$ and $B$, i.e., the class of all elements that are members of $A$ or of $B$ or of both. The symbol $\cup$ is called "cup". |
| $\sim A$ | The complement or negation of $A$, i.e., the class of all elements that are not members of $A$. |

Before considering the interrelations among various classes, we must first define the "universal" class that we are considering. For instance, if $A$ comprises the class of positive integers, we cannot define $\sim A$ until the universal class has been defined. Is $\sim A$ the class of negative integers or the class of positive numbers not integers or perhaps the class of fire engines in Milwaukee? Any of these might be considered the complement of $A$, depending upon the particular universe chosen. If our universe is the class of all the positive numbers, then $\sim A$ is the second of the foregoing possibilities. Any classification of elements into $A$ must be associated either explicitly or implicitly with a classification of other elements into $\sim A$.

If, for instance, our universal class is the class of positive (i.e., nonnegative) integers and if $A$ is the class of positive integers smaller than 101, then $\sim A$ is the class of positive integers greater than 100. If $B$ is the class of positive integers smaller than 200, then $A \leq B$. If $C$ is the

class of positive integers exactly divisible by 25, then $A \cup C$ is the class consisting of the integers 0 through 100 and every 25th succeeding integer. On the other hand, $A \cap C$ is the class consisting of the integers 0, 25, 50, 75, and 100.

Such relations may be conveniently visualized with the help of a Venn diagram. All elements of the universal class are considered to be points

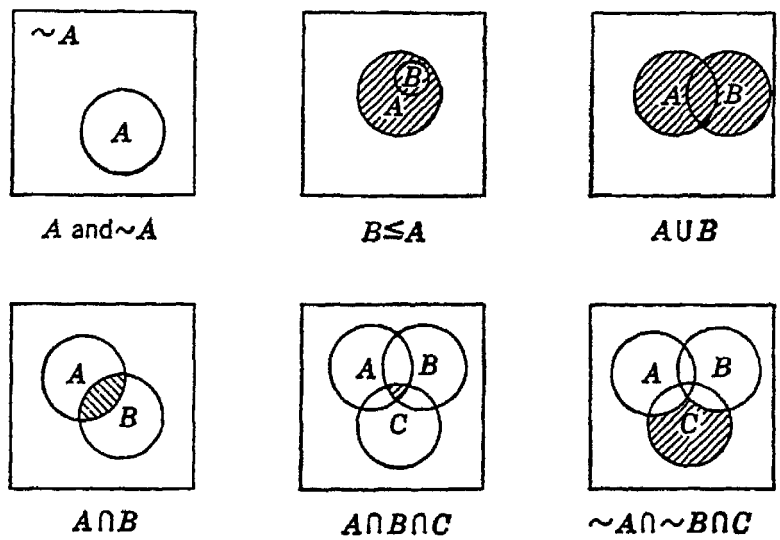



Fig. 8.1 Venn diagrams

lying within a bounding square, and classes of elements in the universal class are delineated by circles inside the square. Some examples are shown in Fig. 8.1.

The operations of intersection and union are commutative, distributive, and associative, as follows:

$$\left.\begin{aligned} A \cup B &= B \cup A \\ A \cap B &= B \cap A \end{aligned}\right\} \quad \text{commutative property} \qquad (8.1)$$

$$\left.\begin{aligned} A \cup (B \cup C) &= (A \cup B) \cup C \\ A \cap (B \cap C) &= (A \cap B) \cap C \end{aligned}\right\} \quad \text{associative property} \qquad (8.2)$$

$$\begin{aligned} A \cup (B \cap C) &= (A \cup B) \cap (A \cup C) \\ A \cap (B \cup C) &= (A \cap B) \cup (A \cap C) \end{aligned} \quad \text{distributive property} \qquad (8.3)$$

It can be readily verified (e.g., by a Venn diagram) that

$$A \cup (B \cap C) \neq (A \cup B) \cap C$$

Accordingly, when $\cap$ and $\cup$ operations appear in the same expression, it is necessary to indicate their scope by parentheses and brackets. In the absence of parentheses, the intersection operation takes precedence over union. Such an expression as $A \cup B \cap C$, therefore, is not ambiguous but is interpreted as meaning $A \cup (B \cap C)$.

If we denote the universal class, i.e., the class containing all elements in the universe of discourse, by 1, and the null class, the class containing no elements, by 0, we may state the following four relations:

| | | |
|---|---|---|
| $A \cup \sim A = 1$ | The class of all elements that are members of $A$ or of $\sim A$ or of both is the universal class. | (8.4) |
| $A \cap \sim A = 0$ | The class of all elements that are members both of $A$ and of $\sim A$ is the null class. | (8.5) |
| $A \cup 1 = 1$ | The class of all elements that are members of $A$ or of the universal class or of both is the universal class. | (8.6) |
| $A \cap 0 = 0$ | The class consisting of all elements that are members both of $A$ and of the null class is the null class. | (8.7) |

It also follows that

$$\sim(\sim A) = A \tag{8.8}$$
$$\sim 0 = 1 \tag{8.9}$$
$$\sim 1 = 0 \tag{8.10}$$
$$\sim(A \cup B) = \sim A \cap \sim B \tag{8.11}$$
$$\sim(A \cap B) = \sim A \cup \sim B \tag{8.12}$$

As an example of the algebra of classes, the following problem, attributed by Birkhoff and MacLane to Lewis Carroll, may be considered:

> In a very hotly fought battle, at least 70 per cent of the combatants lost an eye, at least 75 per cent an ear, at least 80 per cent an arm, and at least 85 per cent a leg. How many lost all four members?

Let $A$, $B$, $C$, and $D$ represent the four classes described in the problem. The class of those who lost all four members is $A \cap B \cap C \cap D$. The number of members in this class can be found from the fact that the number of members in the intersection of two classes equals the sum of the number in each class minus the number in their union. (This may be verified with a little thought, supplemented, if need be, by a Venn diagram.)

Evidently,

$$A \cap B \cap C \cap D = [(A \cap B) \cap C] \cap D$$

$$\begin{aligned}
&\text{Minimum \% number in } A \cap B = 70 + 75 - 100 = 45 \\
&\text{Minimum \% number in } (A \cap B) \cap C = 45 + 80 - 100 = 25 \\
&\text{Minimum \% number in } [(A \cap B) \cap C] \cap D = 25 + 85 - 100 = 10
\end{aligned} \tag{8.13}$$

Thus 10 per cent is the minimum number who certainly lost all four members, although the problem does not preclude the possibility that as many as 70 per cent might have lost all four.

Throughout the relations of the algebra of classes, a similarity may be noted between the operations of intersection and union on the one hand and the arithmetical operations of multiplication and addition on the other. Thus, for instance, the distributive property

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

has the arithmetical analog $x(y + z) = xy + xz$. This similarity must not be carried too far, since, although the union operation is distributive over intersection, i.e., $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$, it is not true in the ordinary algebra of everyday life that $x + yz = (x + y)(x + z)$. However, as we shall see, an algebra exists in which such an equality holds. This becomes apparent in the form in which it is encountered in the logic of propositions, to which we turn now.

## 8.3 The logic of propositions

The theorems of symbolic logic apply with equal validity either to the logic of classes or to the logic of propositions. Only a slight change of viewpoint is necessary. Thus, in the logic of classes, we considered whether elements were members of particular classes or not, whereas in the logic of propositions we consider whether propositions are true or false. A different set of symbols is often used in propositional logic, and the following are frequently met:

| | |
|---|---|
| $x \vee y$ | disjunction, the proposition that "either $x$ or $y$ or both" |
| $x \wedge y$ | conjunction, the proposition that "both $x$ and $y$" |
| $x'$ or $\bar{x}$ | negation, the proposition that "not $x$" |

The variables $x$ and $y$ are propositions, such as "Today is Tuesday" or "$Q$ is an integer" or "A pulse is present on wire 6".

If we now assign to true propositions a value of 1 and to false propositions a value of 0, the terms disjunction, conjunction, and negation may be defined by "truth tables" (Table 8.1), which are tabulations of the "truth values" of complete propositions in terms of the truth values of their components.

Negation of $x$, or, in other words, the proposition "$y = \bar{x}$", is equivalent to the assertion, "If $x$ is true, $y$ is not true, and if $x$ is not true, then $y$ is true." Disjunction of $x$ and $y$ is the proposition that "$x$ is true or $y$ is true or both are true," and the conjunction of $x$ and $y$ is the propo-

**Table 8.1**
**Truth-table definitions**

Negation (a)

| $x$ | $y = \bar{x}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

Disjunction (b)

| $x$ | $y$ | $x \vee y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Conjunction (c)

| $x$ | $y$ | $x \wedge y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

sition that "both $x$ and $y$ are true." Thus, if $x$ is the proposition "Tomorrow is Thursday," $\bar{x}$ is the proposition "Tomorrow is not Thursday." Again, if $x$ is the proposition "There is a pulse on wire 2 at time $t_4$" and $y$ is the proposition "Flip-flop $A$ is in the ON state at time $t_4$," then $x \vee y$ is the proposition "Either there is a pulse on wire 2 at time $t_4$, or flip-flop $A$ is in the ON state at time $t_4$, or there is a pulse on wire 2 at time $t_4$ and flip-flop $A$ is in the ON state at time $t_4$." $x \wedge y$, similarly, is the proposition "There is a pulse on wire 2 at time $t_4$ and flip-flop $A$ is in the ON state at time $t_4$."

**Table 8.2**
**The functions of two variables**

| Variables | Null function | Dual stroke $x \downarrow y$ | | $\bar{x}$ | | $\bar{y}$ | Inequivalence $x \oplus y$ | Sheffer stroke $x\|y$ | Conjunction $x \wedge y$ | Equivalence $x \equiv y$ | $y$ | Material implication, $x \rightarrow y$ | $x$ | $y \rightarrow x$ | Disjunction $x \vee y$ | Identity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x$ $y$ | (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) | (10) | (11) | (12) | (13) | (14) | (15) | (16) |
| 0 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

These are only a few of the possible functions that we might define, since 16 different configurations of truth values are possible with two variables. These are listed in Table 8.2.

Function 15 is the function we have already defined as disjunction, and 9 is evidently conjunction. Function 8 is known as the Sheffer stroke function, and 2 is the dual stroke function, sometimes called the dagger function or the Peirce function. Function 12 is material impli-

cation. The implication function means "$x$ implies $y$"; that is, either $\bar{x}$ is true or $y$ is true. Function 10 is equivalence, that is, $x \equiv y$; and 7 is inequivalence, $x \not\equiv y$, sometimes written $x \oplus y$.

It is of interest to note that these functions are not all independent. In fact, any two (except 1 and 16) can be taken as primitive and all the rest expressed in terms of them. For example, in terms of disjunction and conjunction, negation of $x$ may be defined as the quantity $y$ satisfying the relations $x \vee y = 1$ and $x \wedge y = 0$. Again, in terms of negation and conjunction, the Sheffer stroke function is $\overline{x \wedge y}$. Some familiarity with the manipulations may be gained by selecting various pairs of functions and attempting to define others in terms of them. Perhaps even more interesting is the fact that either the Sheffer stroke or the dual stroke is sufficient by itself to generate all the other functions. Thus, for example, the reader can easily demonstrate such statements as

$$p \wedge q = (p|q)|(p|q)$$

and

$$\bar{p} = p|p$$

Statement of the other functions in terms of the stroke functions is left as an exercise. (Some further comments in regard to the use of the stroke and dual stroke functions will be found at the end of Chap. 10 in connection with the transistor NOR element.)

We have already observed, in connection with the logic of classes, the similarity between certain operations of logic and certain operations of arithmetic. This similarity also arises in connection with propositional logic, and assigning numerical values 1 and 0 to the concepts of truth and falsity, respectively, permits us to replace the disjunction symbol $\vee$ wherever it appears by the addition symbol $+$ and the conjunction symbol $\wedge$ by the multiplication symbol $\times$ (times) or $\cdot$ (center dot), which may, as in conventional algebra, be omitted. In doing this, we must remember that only two numerical values, 1 and 0, are admissible for any of the propositions. Thus it is quite proper to assert that

$$1 + 1 + \cdots + 1 = 1$$

since the truth value of the disjunction of any number of true propositions is simply 1. With this convention in mind we may state several logical relationships, as follows:

$$\begin{aligned} x + x &= x \\ x \cdot x &= x \end{aligned} \tag{8.14}$$

$$\begin{aligned} x + y &= y + x \\ x \cdot y &= y \cdot x \end{aligned} \qquad \text{commutativity} \tag{8.15}$$

$$\begin{aligned} x + (y + z) &= (x + y) + z \\ x \cdot (y \cdot z) &= (x \cdot y) \cdot z \end{aligned} \qquad \text{associativity} \tag{8.16}$$

$$\begin{aligned} x \cdot (y + z) &= (x \cdot y) + (x \cdot z) \\ x + y \cdot z &= (x + y) \cdot (x + z) \end{aligned} \qquad \text{distributivity} \tag{8.17}$$

$$\begin{aligned} x \cdot 0 &= 0 \\ x + 0 &= x \end{aligned} \tag{8.18}$$

$$\begin{aligned} x \cdot 1 &= x \\ x + 1 &= 1 \end{aligned} \tag{8.19}$$

$$\begin{aligned} x + \bar{x} &= 1 \\ x \cdot \bar{x} &= 0 \end{aligned} \tag{8.20}$$

$$\begin{aligned} x + x \cdot y &= x \\ x \cdot (x + y) &= x \end{aligned} \tag{8.21}$$

$$\begin{aligned} (x + \bar{y}) \cdot y &= x \cdot y \\ x \cdot \bar{y} + y &= x + y \end{aligned} \tag{8.22}$$

$$\begin{aligned} \overline{x + y} &= \bar{x} \cdot \bar{y} \\ \overline{x \cdot y} &= \bar{x} + \bar{y} \end{aligned} \qquad \text{De Morgan's theorems} \tag{8.23}$$

These are theorems* in Boolean algebra, which was devised by George Boole about 1850. This algebra is a particularly simple system, since it deals with only two values, 0 and 1, and two operations, + and × (or ·) and has no coefficients or exponents. It is because of this simplicity that some of the theorems at first appear unusual, those, for instance, given by Eqs. (8.21). However, with only a little practice, it is easy to accustom oneself to the manipulations.

We obtain the same set of theorems for our Boolean algebra if we reverse our assignment of values to the concepts of truth and falsity and let truth be represented by 0 and falsity by 1. However, the operation of disjunction is now properly represented by the symbol × and conjunction by +. Therefore, before interpreting a Boolean expression we must know which assignment of truth value it is based upon. The logician customarily uses 1 for truth and 0 for falsity and, therefore, reads "+" as "or" and "×" as "and". The electrical engineer, however, when he applies the algebra to networks of switches to determine their states, finds it convenient to be able to use either value for truth and must be prepared, therefore, to read "+" as "and" and "×" as "or" when truth is assigned a value of 0.

* Strictly speaking, we should call some of them postulates.

## 8.4 Boolean algebra in switching networks[7–10]

A two-valued algebra is particularly convenient for describing networks of switches, each of which must be in one or the other of two states, open or closed. The propositions whose truth or falsity we seek to determine are of the form: "The circuit is closed between points $P_1$ and $P_2$." If we designate truth by 1 and falsity by 0, then the proposition "The circuit is closed between points $P_1$ and $P_2$" takes on a particular truth value for each combination of open and closed settings of the switches in the circuit. The list of these truth values corresponding to all such combinations is called the "transmission function" of the circuit. For example, consider a series connection of two switches $A$ and $B$, a circuit and a truth table for which are shown in Fig. 8.2. In

| $A$ | $B$ | $T$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Fig. 8.2 Transmission function of two switches in series

the table an entry of 0 in the $A$ or $B$ column signifies that that switch is open and a 1 shows that it is closed. For each of the four possible combinations of $A$ and $B$ open or closed, the transmission function $T$ is 0 or 1 depending on the truth or falsity of the proposition "The circuit is closed between points $P_1$ and $P_2$." It is apparent that the transmission function of this circuit may be written as $T = A \cdot B$ and is in fact identical with logical conjunction, since $T$ is true if and only if both $A$ and $B$ are true.

Similarly, for $A$ and $B$ in parallel, the circuit will be closed when either $A$ or $B$ or both are closed. The transmission function for this network is $T = A + B$. Proceeding in this way, we may write down altogether 16 networks and switching functions for two variables $A$ and $B$. These are listed in Fig. 8.3. In this figure the symbol $\circ\!\!-\!\!\!-\!\!\bar{A}\!\!-\!\!\!-\!\!\circ$ is used to mean a switch that is open when $A$ is closed and closed when $A$ is open; i.e., when the variable $A$ takes on the value 0, $\bar{A}$ takes on the value 1, and vice versa.

(It is of interest to note that, although 16 functions are listed, only six different circuit configurations exist, all 16 functions being obtainable from these six by relettering the switches.)

The reader may already have noted that, in choosing to study propositions of the form "The circuit is closed between points $P_1$ and $P_2$" and in assigning to truth a value of 1 and to falsity the value 0, we made entirely arbitrary choices. We might equally well have chosen to state

propositions regarding switching networks in the form "The circuit is open between points $P_1$ and $P_2$" or to have assigned 0 and 1, respectively, as values for truth and falsity. What would be the consequences of such choices? As far as the algebra itself is concerned, the answer is

| $A$ | $B$ | $T_0$ | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ | $T_9$ | $T_{10}$ | $T_{11}$ | $T_{12}$ | $T_{13}$ | $T_{14}$ | $T_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

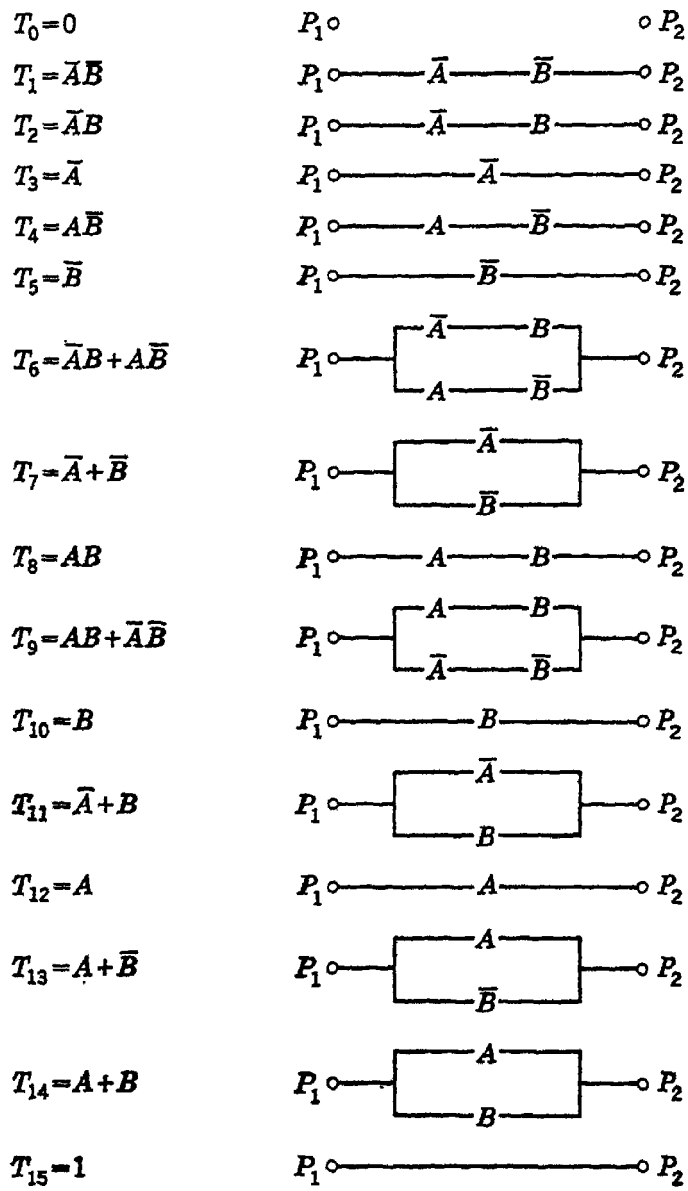



Fig. 8.3 The switching functions of two variables

that these choices would leave the algebra, its manipulations, and its theorems completely unaltered. The algebra is unaffected by the interpretations we choose to place upon its variables and their values and simply constitutes a set of rules for manipulating them. From the cir-

cuit point of view, however, we see that these choices have a profound effect upon the circuit represented by a given function and upon the function representing a given circuit.

To explore this situation let us consider a series connection of switches $A$ and $B$ and describe it in the following four ways:

*a.* Truth = 1 and falsity = 0. The proposition we examine is the assertion "The circuit is closed between points $P_1$ and $P_2$."

*b.* Truth = 0 and falsity = 1. Same proposition.

*c.* Truth = 1 and falsity = 0. The proposition we now examine is the assertion "The circuit is open between points $P_1$ and $P_2$."

*d.* Truth = 0 and falsity = 1. Same proposition as in *c*.

The truth functions for the four cases are tabulated in Table 8.3.

**Table 8.3**
**Truth functions for four types of circuit descriptions**

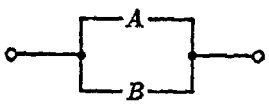
| | (*a*) | | (*b*) | | (*c*) | | (*d*) |
|---|---|---|---|---|---|---|---|
| $A$ $B$ | Truth function | $A$ $B$ | Truth function | $A$ $B$ | Truth function | $A$ $B$ | Truth function |
| 0 0 | 0 | 0 0 | 0 | 0 0 | 0 | 0 0 | 0 |
| 0 1 | 0 | 0 1 | 1 | 0 1 | 1 | 0 1 | 0 |
| 1 0 | 0 | 1 0 | 1 | 1 0 | 1 | 1 0 | 0 |
| 1 1 | 1 | 1 1 | 1 | 1 1 | 1 | 1 1 | 1 |

Functions of type *a* are what we have already called transmission functions. They are analogous to admittance functions of electrical network theory, since a high admittance, like a high value of transmission, indicates easy flow of current, whereas low admittance and low transmission indicate low current. In switching networks, of course, the transmission is "all or nothing", that is, 1 or 0, whereas in electrical networks the admittance function has a wide range for both its resistive and reactive components. Functions of type *b*, on the other hand, are known as hindrance functions and are analogous to impedance functions. A switching circuit with low hindrance, that is, 0, like an *RLC* network with low impedance, admits a large current; one with high hindrance, that is, 1, like an *RLC* network with high impedance, admits low current. We may now observe that functions of type *c* are the same as those of type *b*, since the assertion "It is true that the circuit is closed" is the same as "It is false that the circuit is open," and the same numerical value, 0, is used for each of these statements. By the same reasoning we see that functions of type *d* are the same as those of type *a*. Thus we see that there are two distinct methods, transmission and hindrance, for describing switching networks. Neither one is "better" than the other; and in switching networks, unlike electrical networks, in which there may be fairly clear-cut reasons for preferring admittances to imped-

ances or impedances to admittances in a given circuit situation, there are rarely compelling reasons other than personal preference for choosing either transmission or hindrance in preference to the other.

The transmission function for the series connection of $A$ and $B$ is $T = A \cdot B$; the hindrance function $H$ for the same circuit is $H = A + B$. As an exercise, the reader will find it worth while to form the hindrance

**Table 8.4**
**Summary of transmission and hindrance**

| | Hindrance | Transmission |
|---|---|---|
| Closed circuit<br>Open circuit | $H = 0$<br>$H = 1$ | $T = 1$<br>$T = 0$ |
| Series circuit<br> <br> | Closed if and only if $A$ and $B$ are closed<br>Hindrance = 0 if and only if both $A$ *and* $B$ are 0<br>$\therefore H = A + B$ | Closed if and only if $A$ and $B$ are closed<br>Transmission = 1 if and only if both $A$ and $B$ are 1<br>$\therefore T = A \cdot B$ |
| Parallel circuit<br>  <br> | Closed if $A$ *or* $B$ *or* both are closed<br>Hindrance = 0 if $A$ *or* $B$ *or* both are 0<br>$\therefore H = A \cdot B$ | Closed if $A$ *or* $B$ *or* both are closed<br>Transmission = 1 if $A$ *or* $B$ *or* both are 1<br>$\therefore T = A + B$ |
| $+$<br>$\cdot$ | $+$ corresponds to a series circuit (AND)<br>$\cdot$ corresponds to a parallel circuit (OR) | $+$ corresponds to a parallel circuit (OR)<br>$\cdot$ corresponds to a series circuit (AND) |

function for each of the circuits of Fig. 8.3. It is apparent that the operation of conjunction in propositional logic has as its analog in switching algebra the operation "$\cdot$" when the transmission basis is used and the operation "$+$" when the hindrance basis is used. In fact, given either a $T$ function or an $H$ function for a given network, we may form the other by replacing all "$+$" signs by "$\cdot$" and all "$\cdot$" by "$+$". In addition, any 1's in the original function should be changed to 0's and any 0's to 1's.

Similarly, disjunction is represented by "$+$" when the transmission basis is used and by "$\cdot$" when the hindrance basis is used. The two methods are summarized in Table 8.4.

As examples of the use of these methods, Fig. 8.4 shows two simple networks of switches together with the $H$ and $T$ functions for each.

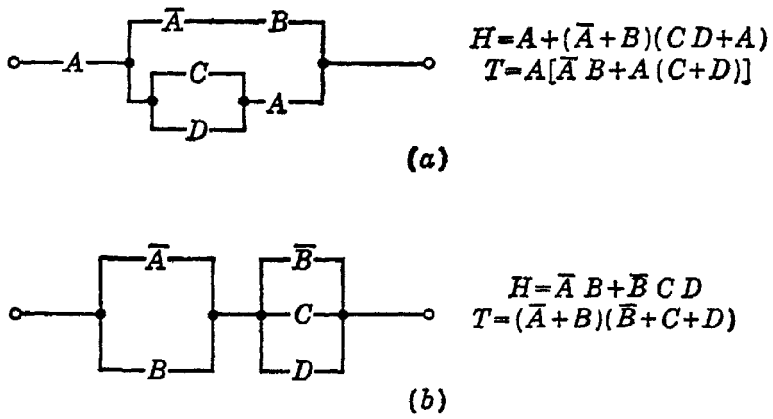



Fig. 8.4 Two networks and their $H$ and $T$ functions

Conversely, a switching function corresponds to two different circuits, depending on whether it is interpreted as a transmission function or as a hindrance function. The function $AB + C(B + D)$ has the circuit forms shown in Fig. 8.5.

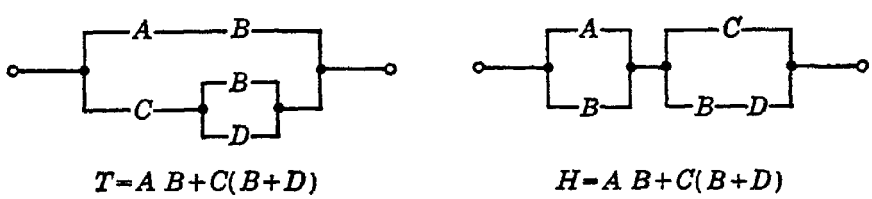



Fig. 8.5 Two circuits described by the same function, interpreted (left) as a $T$ function and (right) as an $H$ function

**Dual networks.** Another useful concept in switching circuitry is that of duality. Just as two electrical networks are said to be duals if the impedance function of one is of the same form as the admittance function of the other, so two switching networks are duals if the hindrance function of one is the same as the transmission function of the other. The two switching networks of Fig. 8.5 are examples of duals. The dual of a series circuit is a parallel circuit composed of the same switches, and the dual of a parallel circuit is a series circuit of the same switches. In general we may say that, if circuits 1 and 2 are duals, then $T_1 = H_2$ and $T_2 = H_1$.

The familiar topological procedure of electrical network theory for finding the dual of a planar network can also be used for planar switching networks. The connecting leads for the two-terminal network are assumed to be extended infinitely, a dot is placed on each side of this dividing line and in each mesh, and these dots are then connected by lines passing through the various switches. The resulting network, with a switch in each line that passed through a switch, is the dual net. An example of this method is shown in Fig. 8.6. The original network has

terminals $P_1$ and $P_2$ and its dual has terminals $Q_1$ and $Q_2$. The derived dual is shown "straightened out" in Fig. 8.6*b*.

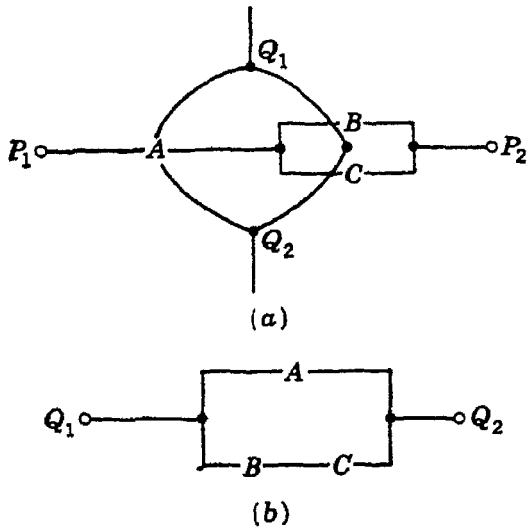
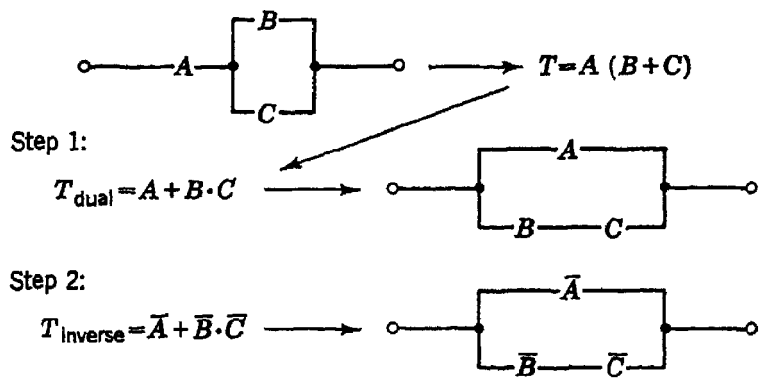



Fig. 8.6 Topological method of forming a dual network

**Inverse networks.** Dualizing a network is a step in forming the inverse (or "negative") of a given network. By "inverse network" we mean the network that is open when the given network is closed and closed when the given network is open. The two steps in forming the inverse of a given $H$ function or $T$ function are:

1. Form the dual of the given function.
2. Complement (negate) all variables.

As an example, consider the network of Fig. 8.7. The transmission functions for the original network and for the derived network are shown in the figure.




| $A$ | $B$ | $C$ | $T$ | $T_{\text{inverse}}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

Fig. 8.7 Forming the inverse of a network

## 8.5 Manipulative techniques of Boolean algebra

Let us turn now to consideration of the uses for the various algebraic relations stated in Eqs. (8.14) to (8.23). These relations, together with

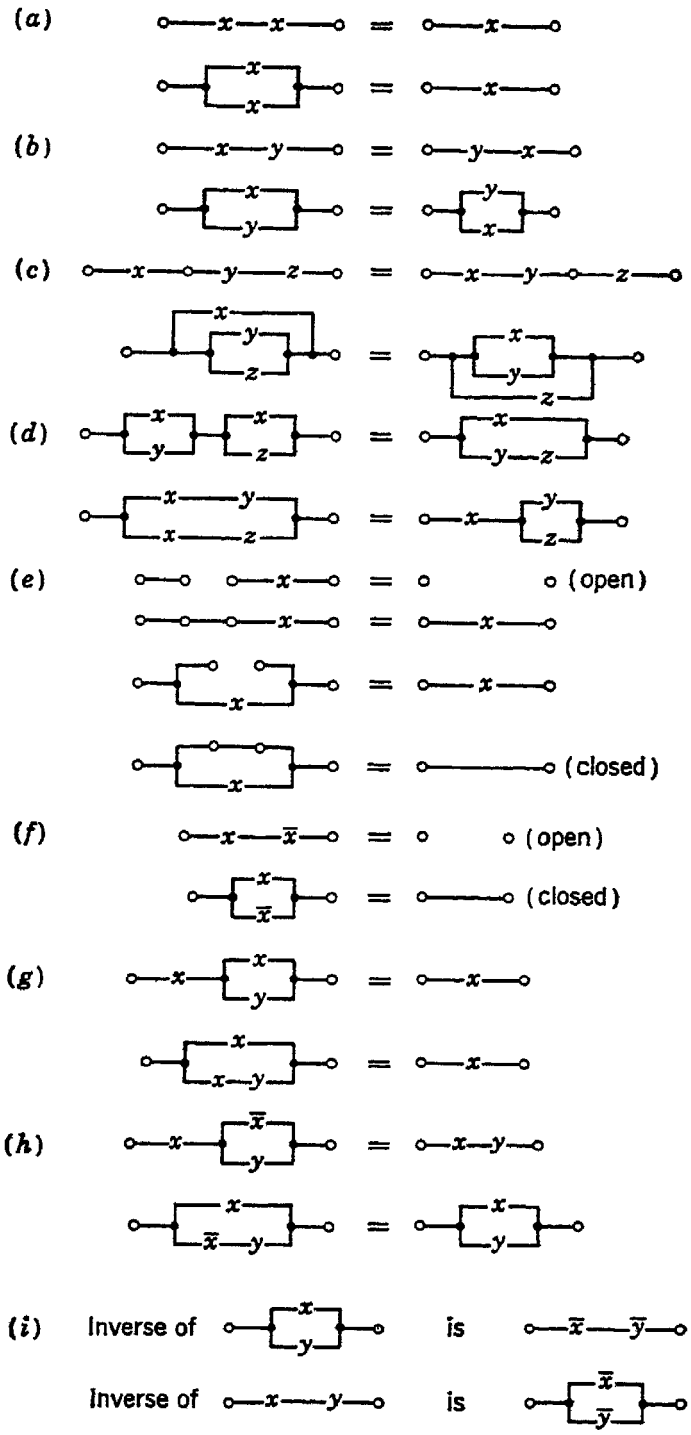



Fig. 8.8 Circuits demonstrating some relationships in switching algebra

others that can be derived from them, are valuable for manipulating switching functions in performing both analysis and synthesis of switching networks. We shall first discuss the significance of each of the relations and then apply them to network manipulations. Circuits corresponding to the following descriptions are shown in Fig. 8.8.

*a.* $x + x = x$ and $xx = x$. These relations state that any number of identical switches in series or in parallel may be replaced by a single switch.

*b.* $x + y = y + x$ and $xy = yx$. The order of series switches or of parallel switches may be altered without affecting the switching functions.

*c.* $x + (y + z) = (x + y) + z$ and $x(yz) = (xy)z$. Series switches or parallel switches may be grouped in any way.

*d.* $xy + xz = x(y + z)$ and $(x + y)(x + z) = x + yz$. A switch common to each of several parallel groups that are in series may be removed and placed in parallel with the entire group. A switch common to each of several series groups that are in parallel may be removed and placed in series with the entire group.

*e.* $x \cdot 0 = 0$, $x + 0 = x$, $x \cdot 1 = x$, and $x + 1 = 1$. A switch in series with an open switch is an open circuit. A switch in parallel with a closed switch is a closed circuit. A switch in series with a closed switch or in parallel with an open switch has the same $T$ or $H$ function as the switch alone.

*f.* $x + \bar{x} = 1$ and $x\bar{x} = 0$. A switch and its inverse in series form an open circuit. A switch and its inverse in parallel form a closed circuit.

*g.* $x + xy = x$ and $x(x + y) = x$. A switch in series with a parallel circuit containing the same switch may be replaced by the switch alone. A switch in parallel with a series circuit containing the same switch may be replaced by the switch alone.

*h.* $x(\bar{x} + y) = xy$ and $x + \bar{x}y = x + y$. If a switch is in parallel with its inverse and another in series, the inverse switch may be replaced by a closed path. If a switch is in series with its inverse and another in parallel, the inverse switch may be replaced by an open path.

*i.* $\overline{x + y} = \bar{x}\bar{y}$ and $\overline{xy} = \bar{x} + \bar{y}$. The inverse of parallel switches is a series connection of their inverses. The inverse of series switches is a parallel connection of their inverses.

In each of the foregoing statements, either equation may be identified with either verbal statement, depending on whether the transmission or the hindrance basis is used.

The following example illustrates the use of the relations in simplifying functions. Let it be required to simplify the function

$$S = BC(\bar{D}A + \bar{B} + C) + \bar{C}(D + \bar{A})(B + C) \tag{8.24}$$

where $S$ may be either a $T$ function or an $H$ function. From the distributive property we may write

$$S = BC\bar{D}A + BC\bar{B} + BCC + \bar{C}DB + \bar{C}DC + \bar{C}\bar{A}B + \bar{C}\bar{A}C$$

Since $\cdot$ is commutative, we may write

$$S = ABC\bar{D} + B\bar{B}C + BCC + B\bar{C}D + C\bar{C}D + \bar{A}B\bar{C} + AC\bar{C}$$

But $B\bar{B} = C\bar{C} = 0$ and $CC = C$

so $S = ABC\bar{D} + BC + B\bar{C}D + \bar{A}B\bar{C}$

Also, $BC + ABC\bar{D} = BC$

so $S = BC + B\bar{C}D + \bar{A}B\bar{C}$

Factoring out $B$ and $\bar{C}$ gives

$$S = B[C + \bar{C}(\bar{A} + D)]$$

But $C + \bar{C}(\bar{A} + D) = \bar{A} + C + D$

Finally, $$S = B(\bar{A} + C + D) \tag{8.25}$$

**Canonical forms of Boolean functions.** It is often useful to express Boolean functions in what is called the canonical (or standard or normal) form. These canonical forms, which are useful both in comparing functions and in simplifying them, are as follows:

1. A sum of products, corresponding to the disjunctive normal form in propositional logic
2. A product of sums, the conjunctive normal form

These forms are said to be fully expanded* if each term in the first (or each factor in the second) contains one and only one appearance of every variable, either with or without a bar (negation symbol). The expressions are otherwise said to be in contracted form. To express a function in the fully expanded form, it may be necessary to add terms of the form $x\bar{x}$, which are identically 0, or factors of the form $x + \bar{x}$, which are identically 1. Consider, for instance, the Boolean function $A + \bar{B}C$. To express this function as a sum of products, we first note that it is already in this form, but not fully expanded; i.e., not all variables appear in each term. Therefore, we multiply each term by a factor that includes the missing letters as follows:

$$A + \bar{B}C = A(B + \bar{B})(C + \bar{C}) + (A + \bar{A})\bar{B}C \tag{8.26}$$

$$= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C \tag{8.27}$$

This is one of the two canonical forms. We note that it contains five of the eight possible products of the three variables $A$, $B$, and $C$. These eight are $ABC$, $AB\bar{C}$, $A\bar{B}C$, $A\bar{B}\bar{C}$, $\bar{A}BC$, $\bar{A}B\bar{C}$, $\bar{A}\bar{B}C$, and $\bar{A}\bar{B}\bar{C}$. It can be shown that there is only one possible way to expand completely a Boolean function as a sum of products; therefore, the canonical form we have found in Eq. (8.27) is unique.

The other canonical form is also the only completely expanded form of its kind and consists of a product of some, at least, of the eight fac-

* The adjective "developed" is sometimes used instead of "fully expanded"

tors $A + B + C$, $A + B + \bar{C}$, $A + \bar{B} + C$, $A + \bar{B} + \bar{C}$, $\bar{A} + B + C$, $\bar{A} + B + \bar{C}$, $\bar{A} + \bar{B} + C$, and $\bar{A} + \bar{B} + \bar{C}$. It can be found either from the original function or from the sum-of-products form. To proceed from the sum-of-products form, we write down the sum of all the product terms not present in the original expanded canonical form, then interchange + and × throughout, and negate every symbol. Thus, to get the product-of-sums canonical form of $A + \bar{B}C$, we write first $\bar{A}BC + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C}$. Interchanging + and × gives

$$(\bar{A} + B + C)(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + \bar{C}) \tag{8.28}$$

and negating all symbols gives

$$(A + \bar{B} + \bar{C})(A + \bar{B} + C)(A + B + C) \tag{8.29}$$

The same result could have been obtained by first using Eq. (8.17) as follows:

$$A + \bar{B}C = (A + \bar{B})(A + C) \tag{8.30}$$

Now we add null terms to get

$$A + \bar{B}C = (A + \bar{B} + C\bar{C})(A + B\bar{B} + C) \tag{8.31}$$

Applying the same theorem again, we get

$$A + \bar{B}C = [(A + \bar{B}) + C][(A + \bar{B}) + \bar{C}][(A + C) + B][(A + C) + \bar{B}] \tag{8.32}$$

which is, of course,

$$A + \bar{B}C = (A + \bar{B} + C)(A + \bar{B} + \bar{C})(A + B + C) \tag{8.33}$$

which is the same as (8.29).

In general, to obtain one expanded canonical form from the other, we first write the absent product terms (or sum factors), then interchange + and × throughout, and finally negate all symbols. After being put in the same canonical form, two Boolean functions can readily be compared term by term to determine whether they are identical. The general procedure for obtaining a canonical form is to make use of one of the following theorems for the expansion of a Boolean function:

$$f(x) = \bar{x} \cdot f(0) + x \cdot f(1) \tag{8.34}$$

$$f(x) = [\bar{x} + f(1)][x + f(0)] \tag{8.35}$$

These theorems also hold for many variables. Thus,

$$\begin{aligned} f(x,y,z) &= \bar{x} \cdot f(0,y,z) + x \cdot f(1,y,z) \\ &= \bar{x}[\bar{y} \cdot f(0,0,z) + y \cdot f(0,1,z)] + x[\bar{y} \cdot f(1,0,z) + y \cdot f(1,1,z)] \end{aligned} \tag{8.36}$$

$$\begin{aligned} f(x,y,z) = {} & \bar{x}\bar{y}\bar{z} \cdot f(0,0,0) + \bar{x}\bar{y}z \cdot f(0,0,1) \\ & + \bar{x}y\bar{z} \cdot f(0,1,0) + \bar{x}yz \cdot f(0,1,1) \\ & + x\bar{y}\bar{z} \cdot f(1,0,0) + x\bar{y}z \cdot f(1,0,1) \\ & + xy\bar{z} \cdot f(1,1,0) + xyz \cdot f(1,1,1) \end{aligned} \tag{8.37}$$

Applying this formula to our example $A + \bar{B}C$, we get

$$\begin{aligned} A + \bar{B}C = {} & \bar{A}\bar{B}\bar{C} \cdot 0 + \bar{A}\bar{B}C \cdot 1 + \bar{A}B\bar{C} \cdot 0 + \bar{A}BC \cdot 0 \\ & + A\bar{B}\bar{C} \cdot 1 + A\bar{B}C \cdot 1 + AB\bar{C} \cdot 1 + ABC \cdot 1 \end{aligned} \tag{8.38}$$

$$= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C \tag{8.39}$$

as before.

## 8.6 Types of switching networks

The only switching circuits we have mentioned so far are extremely simple systems. (1) They are two-terminal circuits, or, to use a better characterization, they are what we might call 1-to-1 terminal networks, since they establish connection from one terminal to one terminal. (2) They are what is known as combinational circuits, since their state (1 or 0) at any time depends only upon the combination of the states of the switches at that time and not in any way upon the sequence of states by which that combination was reached. (3) Definite combinations of input-variable states are required for each of the network states in contrast with certain less rigorously specified networks in which the network state depends only on the number of input variables in the 1 state or upon their relative positions in the network.

Some examples may help to clarify the reader's understanding of all these types of networks.

**Multiterminal networks.** Figure 8.9 shows block diagrams of the 1-to-1, 1-to-$n$, $n$-to-$n$, and $n$-to-$m$ terminal networks. A switching function can be written relating each terminal on the left to each terminal on the right. (Note that these are not distinguishable as to "input" and "output" in the sense of electrical network theory.) The variables in each of these functions will be some, at least, of the input variables $A_1 \cdots A_k$. The number of such functions that can be written for the general $n$-to-$m$ terminal net is $n \cdot m$, and the problem of synthesizing such a network can be formally solved by a straightforward construction of the series-parallel networks corresponding to each of these functions. Such a procedure, although it does yield *a* network, may reasonably be expected not to yield the optimum network. Although it is sometimes

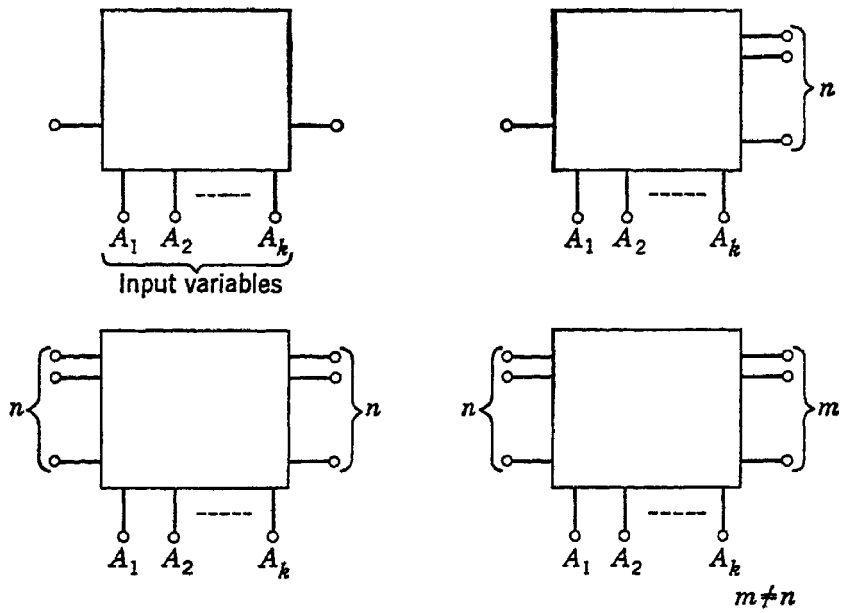



Fig. 8.9 Switching-network terminal arrangements

possible to reduce the total number of switches by allowing various of the functions to share switches, other requirements upon the circuit may make this impossible. An example is shown in Fig. 8.10, where the two terminals on the left in *a* have been reduced to one in *b* by making both paths share the same switch $A$, since $A$ is a series element in each of them. The network of *b* is not equivalent in every sense to that of *a*, since the

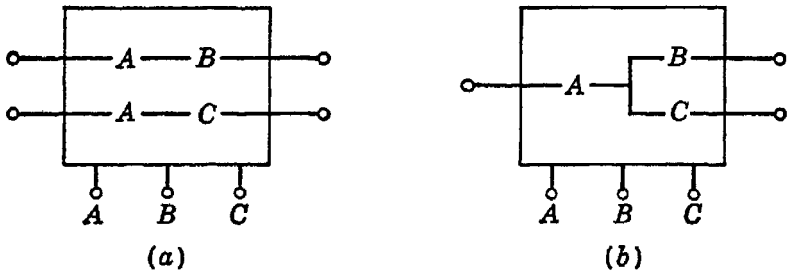



Fig. 8.10 A multiterminal-network simplification

two terminals on the left are now electrically identical instead of separate. Furthermore, the two terminals on the right of *b* can be electrically connected if $B$ and $C$ are both closed, a condition that did not exist in *a*.

**Combinational and sequential networks.** The distinction between combinational circuits and sequential circuits is very important, from both theoretical and practical points of view. The combinational circuit, that is, the circuit in which the present output is a function only of the present inputs, is straightforward both to analyze and to synthesize, the only real problem being to optimize the circuit in some manner. The output of the sequential circuit, on the other hand, depends not only upon the present inputs but also upon their previous history. Analysis and synthesis of such networks are laborious and involved, and although

important advances have been made in this area in recent years, much remains to be done. Sequential circuits are discussed in more detail in Sec. 8.10.

**Iterative networks.** A class of combinational circuits occasionally encountered is the class of iterative networks, so called because they are implemented by iterating a sequence of identical networks. Conditions

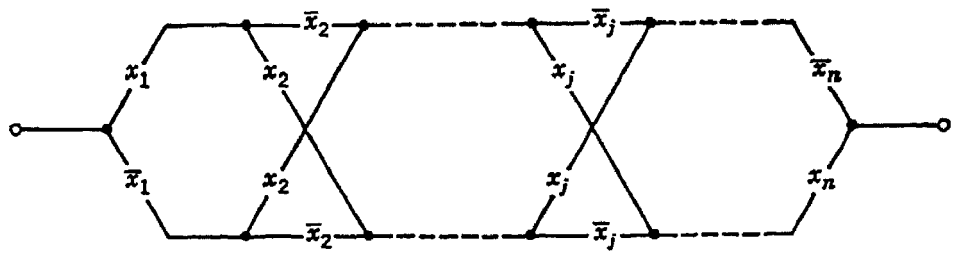



Fig. 8.11 Iterative circuit closed when an odd number of relays are activated

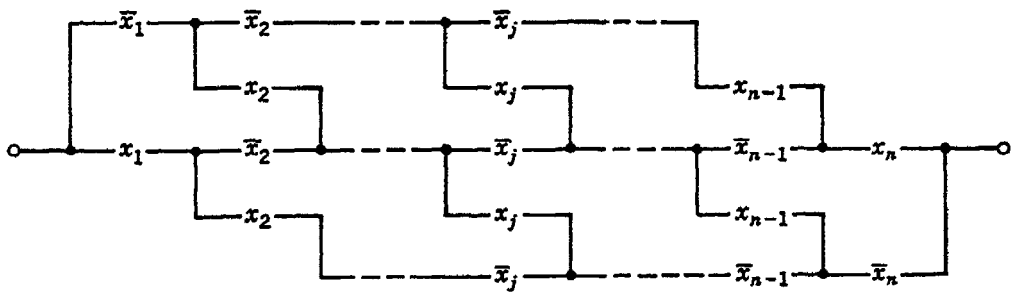



Fig. 8.12 Iterative circuit closed when exactly two adjacent relays are activated

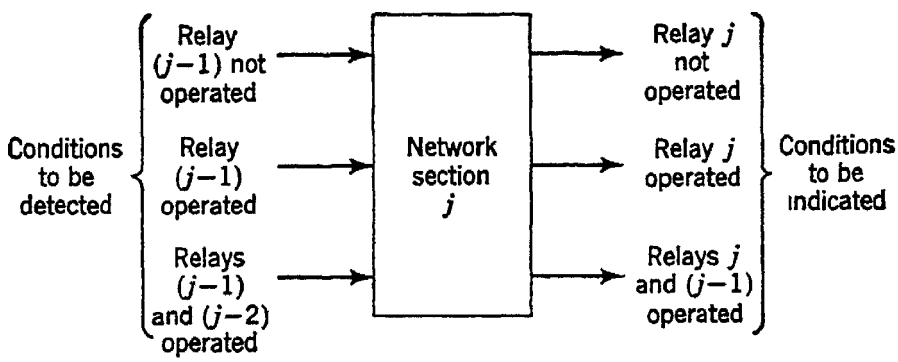



Fig. 8.13 Example of the design technique for iterative networks

for closing such circuits are stated in terms of the total number of elements activated or in terms of their relative positions rather than by specifying the particular variables that should be 1 for circuit closure. For example, Fig. 8.11 shows a circuit that is closed whenever an odd number of relays is activated, and Fig. 8.12 shows a circuit that is closed whenever exactly two adjacent relays are activated anywhere in the chain. The iterated section of each network is formed of switches $x_j$ and $\bar{x}_j$. The terminating sections are somewhat simpler. Design of such iterated sections proceeds from statements of conditions to be sensed by the section and conditions to be indicated by the section. For instance, Fig. 8.13 shows the conditions for a section of the circuit of Fig. 8.12.

From this it is a straightforward matter (in this simple example, at least) to derive the circuit of Fig. 8.12.

## 8.7 Some physical characteristics of switching elements

In any discussion of switching theory a point is reached at which it is necessary to take into account certain ways in which switching devices differ. This usually occurs when comparisons must be made between various circuits for implementing a given switching function. Switching devices differ so widely in their properties that no simple formalism, i.e., structure of definitions and theorems, has been constructed that is equally applicable to comparisons of all of them. However, it is possible to establish two classes of switching devices within each of which adequate formalisms either do exist or can exist and within each of which circuit comparisons can be made in a precise manner. The choice of which of these two classes of switching elements is to be used in a given physical system (e.g., a telephone exchange, a computer, or a process controller) is seldom made on the basis of any rigorous circuit comparisons of the number of elements used but is more often determined by such practical engineering considerations as speed, life, reliability, etc.

The principal difference between these two classes is electrical. One class consists of switches that either establish a connection between two points or do not. Interconnections of switches form switching networks, and for any two points $a$, $b$ in such a network it is meaningful to investigate the truth of the proposition "The circuit is closed between $a$ and $b$." Current can flow in either direction along any closed path in such a network, and so the terms "input" and "output" have no meaning when applied to the closed path. We regard the inputs instead as being the signals that activate the switches, and we regard the outputs as being the logical functions that express the conditions under which each pair of terminals is connected.

The most common element in this first class is the electromagnetic relay, in which a magnetic field produced by a current in a solenoid moves an armature carrying springs with electrical contacts arranged variously to close and to open ("make" and "break") when the energizing current flows. The circuit of Fig. 8.14, for example, shows two relays with a switching network constructed on their contacts. The two variables $A$ and $B$ are considered to be the inputs. We may regard the functions $A \cdot B$ and $A \cdot \bar{B}$ as being the outputs, in the sense that these are the switching functions realized by the circuit.

The other class of switching devices consists of active elements that produce an electrical voltage (with reference to ground) on a single terminal when a specified combination of such voltages is present at the

input terminals of the device. This single terminal may properly be termed the output terminal, and the logical function stating the conditions under which the signal is present at that terminal may be regarded as the output function realized at it. For any such output terminal $a$ the proposition whose truth may be meaningfully examined is the assertion "A signal is present at terminal $a$."

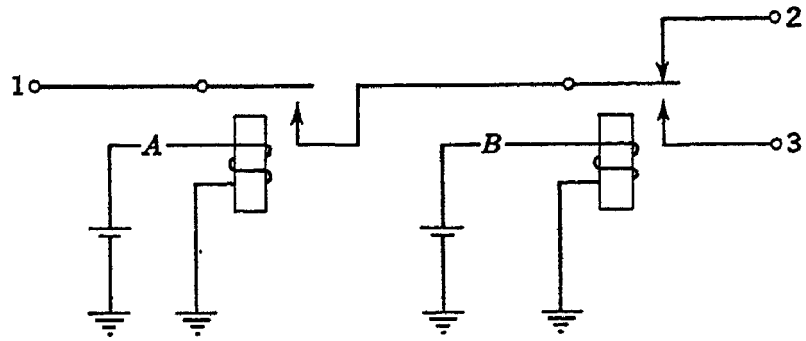



Fig. 8.14 A simple relay circuit

Most types of switching devices fall in this second class, including logical elements made of vacuum tubes, semiconductor diodes, transistors, and magnetic cores. (This should not be construed as minimizing the importance of relays, which have probably been used for logical functions in greater quantity than all these other devices together and which will continue to find application for low-speed uses.) The logical func-

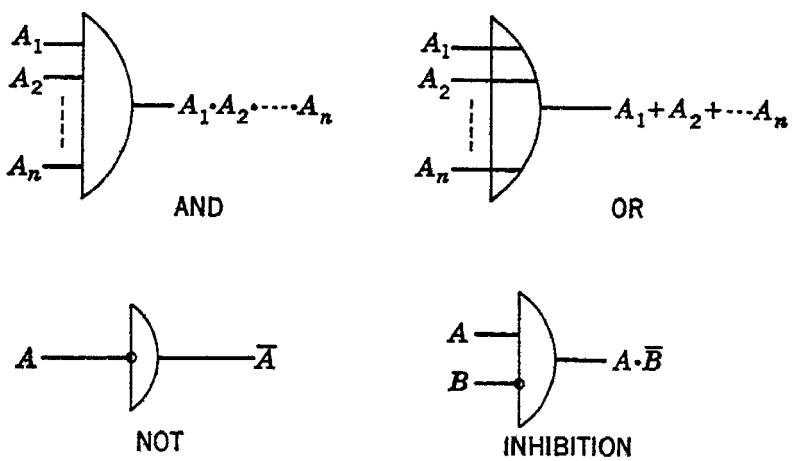



Fig. 8.15 A set of symbols for voltage-producing logical elements (see Fig. 8.52 for another set)

tions most often constructed of these devices are AND, OR, and NOT, for which the symbols of Fig. 8.15 are often used. The AND element produces an output signal if and only if signals are present at all of $A_1$, $A_2$, . . . $A_n$. The OR device produces an output signal if at least one of $A_1$, $A_2$, . . . $A_n$ is present. The NOT device produces a signal if and only if $A$ is not present. The inhibition symbol shows an inhibiting

input to an AND element (it is usually not shown with an OR element, for electrical reasons), which has the effect of preventing any output signal when $B$ is present.

It is characteristic of this class of active logical elements that the input signals of each element are output signals of others. The logical function (AND, OR, NOT) realized by each element is implemented by the electrical devices and connections represented by each of the symbols of

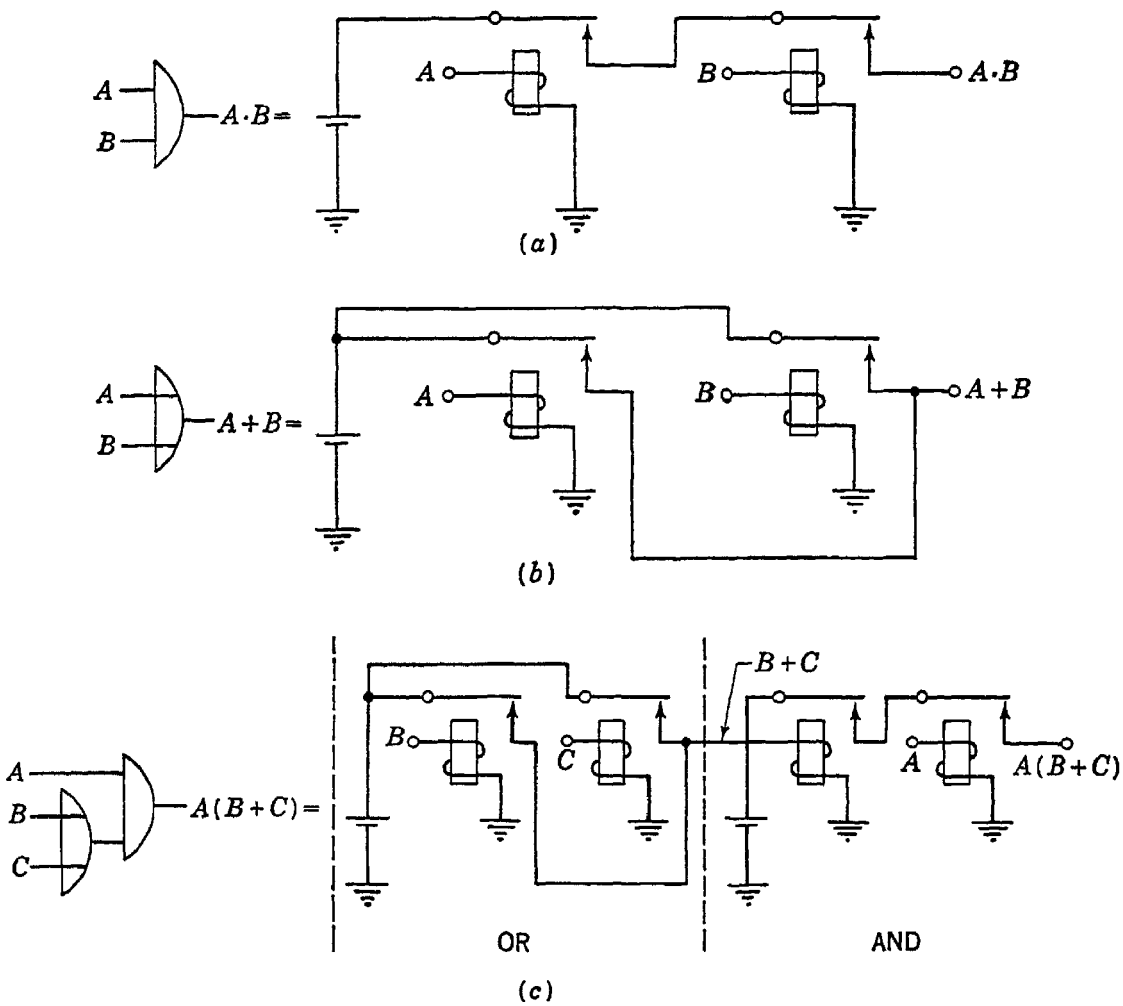



Fig. 8.16 Relay-circuit equivalents of the type of logical elements shown in Fig. 8.15

Fig. 8.15. In relay networks, however, it is not the device that performs the logical function but the interconnection of its contacts, and it is not necessary for every relay contact to be in series with a relay coil. Although it is possible to regard the relay as a special type of active element, circuits constructed of relays in this manner are generally less useful than circuits that employ relays in a manner more appropriate to their structure. Thus, each of the semicircle symbols of Fig. 8.15 can be regarded as a relay network, where the input terminals are connections to relay coils and the outputs are the terminals of contact networks in series with voltage sources.

Figure 8.16 shows relay equivalents for the AND and OR symbols of Fig. 8.15, together with a simple function network drawn in symbol form and in equivalent relay form. Note that in this network two relays are used for the OR function and two for the AND function and that the output $B + C$ from the first two relays drives a relay coil in the second pair. A better circuit arrangement of relays is shown in Fig. 8.17; this circuit uses only three relays instead of four and has only six contact springs instead of eight, and it has the further advantage that the only time delay in establishing the closed path is that required for relays $A$ and ($B$ or $C$) to operate after the energizing currents are applied. In the

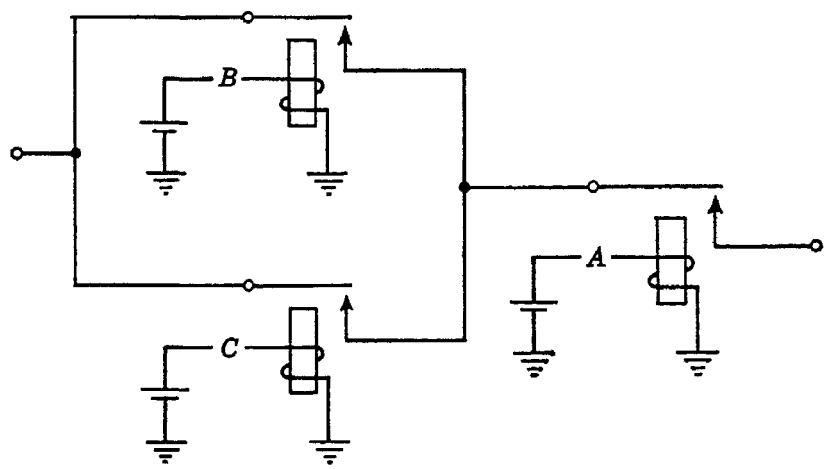



Fig. 8.17 An improvement over the circuit of Fig. 8.16c

circuit of Fig. 8.16 the third relay from the left cannot be energized until either $B$ or $C$ has operated after being energized; thus its time delay must be added to the delay of whichever relay $A$, $B$, or $C$ is slowest in order to ascertain the over-all delay in establishing the closed circuit.

It is evident that, although the semicircle symbols can be given a relay embodiment, the relay circuit obtained by replacing each symbol by its relay equivalent is often slower and less economical of equipment than necessary. This is one of the principal reasons for retaining a separate symbolism for relay circuits. The other reason is that the semicircle symbol is not capable of representing the many ingenious ways of connecting relays, such as introducing shunt paths around coils, coils in series, coils in parallel, networks including coils and contacts, and polarizing relays, which are sensitive to the current direction.

All the structure of Boolean algebra treated in the first part of this chapter is equally applicable to both these classes of switching devices. The differences between them must be considered only when it is important to distinguish between the statement "The circuit is closed between points $a$ and $b$," and the statement "A signal is present at point $a$," or when circuits must be compared to determine relative numbers of components used.

## 8.8 Some comparisons of circuit costs

In comparing the costs of various relay networks for implementing a function, the significant quantities are (1) the number of relays used, (2) the total number of contact springs on all the relays used, and (3) the maximum number of springs on any one relay. No general method exists for minimizing all these quantities in any arbitrary network, but for a few simple cases the theory has been fairly well developed.

One of these cases is what we may call the "binary decoding network". Such a network has $n$ input variables and $2^n$ output terminals, any one of which may be connected to a common terminal by a suitable combination of the variables. One form of network to perform this decoding is shown in Fig. 8.18. This network contains three relays, each of which

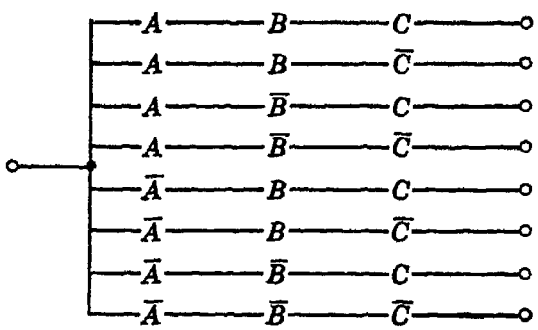



Fig. 8.18 The binary decoding network

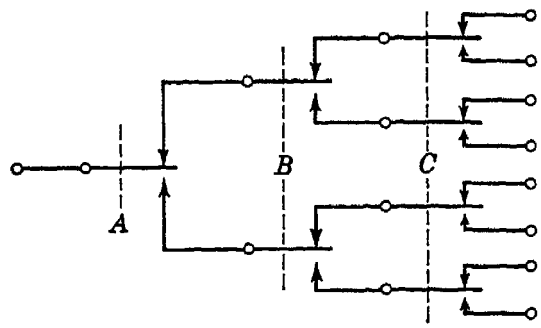



Fig. 8.19 The (binary) tree

carries eight pairs of contact springs, giving a total of 48 springs. A much more economical form of this circuit is the configuration of Fig. 8.19, which is called a "tree". This circuit also has three relays, but the $A$ relay carries only three springs, the $B$ relay six, and the $C$ relay 12. These "transfer" contacts effect a considerable economy in contact springs, the total number for an $n$-relay tree being $3(2^n - 1)$ instead of $n \times 2^{n+1}$ as required by the circuit of Fig. 8.18. A disadvantage of this form of tree is the unequal numbers of springs on the various relays. The more heavily loaded relays will operate more slowly and less reliably unless their magnetic circuits are designed for heavier loads, and in this case their power consumption is greater. It is possible to rearrange the circuit by a process called "folding", which gives more uniform spring loads for the various relays. This technique has been studied by Shannon[11] for relays and by Burks et al.[12] for more general networks. A folded tree that performs the same function as the network of Fig. 8.18 is shown in Fig. 8.20. Again 21 springs are needed but these are distributed three to the $A$ relay, nine to the $B$ relay, and nine to the $C$ relay. The spring load on the most heavily loaded relay has been reduced at the expense of the load on more lightly loaded relays.

**Complete decoding nets.** An interesting example of a well-developed formalism for comparing networks has been developed by Professor A. W. Burks and his associates at the University of Michigan.[13] This method applies to what are called "complete" decoding nets, constructed of active circuit devices rather than of relays. The input variables for such a net are considered to be grouped into $d$ sets in such a way that one and only one variable in each set can be 1 at a given time, and a total of $d$ variables each equal to 1 constitutes an allowable input combination. A net that forms a unique switching function for each of these

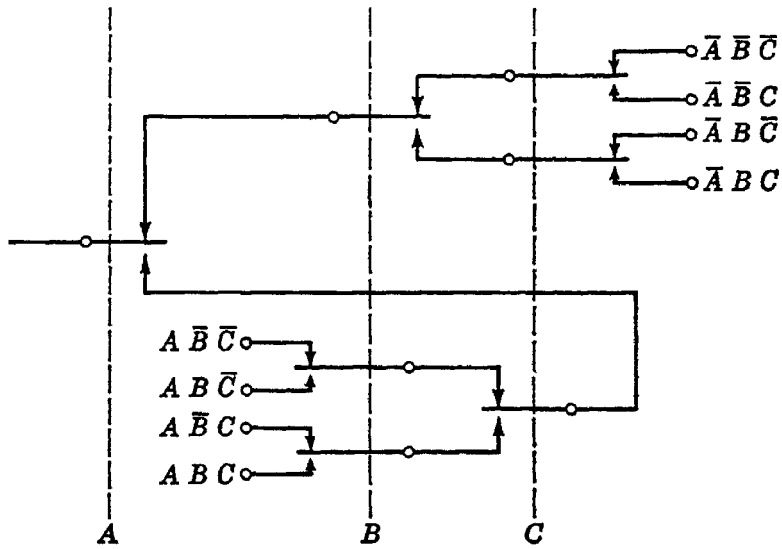



Fig. 8.20 A folded tree

allowable input combinations is a complete decoding net. A $b$-$d$ complete decoding net is defined as a complete decoding net with $d$ sets of $b$ variables each, which realizes $b^d$ switching functions. (The three-variable binary tree of Fig. 8.18 may be regarded as a 2-3 complete decoding net, since it has three sets of variables $A, \bar{A}$; $B, \bar{B}$; $C, \bar{C}$, where one and only one variable in each set can be 1 at a given time; three of the variables must be 1 at a given time; and $2^3$ switching functions are realized.)

Let us now define a multiplicative switch as follows:

*Definition* 1. The $i_1, i_2, \ldots, i_d$ multiplicative switch is a net having (*a*) $i_1 + i_2 + \cdots + i_d$ input variables grouped into $d$ sets of $i_1, i_2, \ldots, i_d$ members, respectively, with allowable input combinations consisting of exactly one variable equal to 1 in each of the $d$ sets; (*b*) $i_1 i_2 \cdots i_d$ realizable switching functions; (*c*) $i_1 i_2 \cdots i_d$ logic elements, each being a $d$-input conjunction; and (*d*) for each logic element $d$ inputs, one from each of the $d$ sets.

Some simple multiplicative switches are shown in Fig. 8.21. Figure 8.21*a* shows a switch for which $i_1 = 2$ and $i_2 = 3$. There are $i_1 + i_2 = 5$

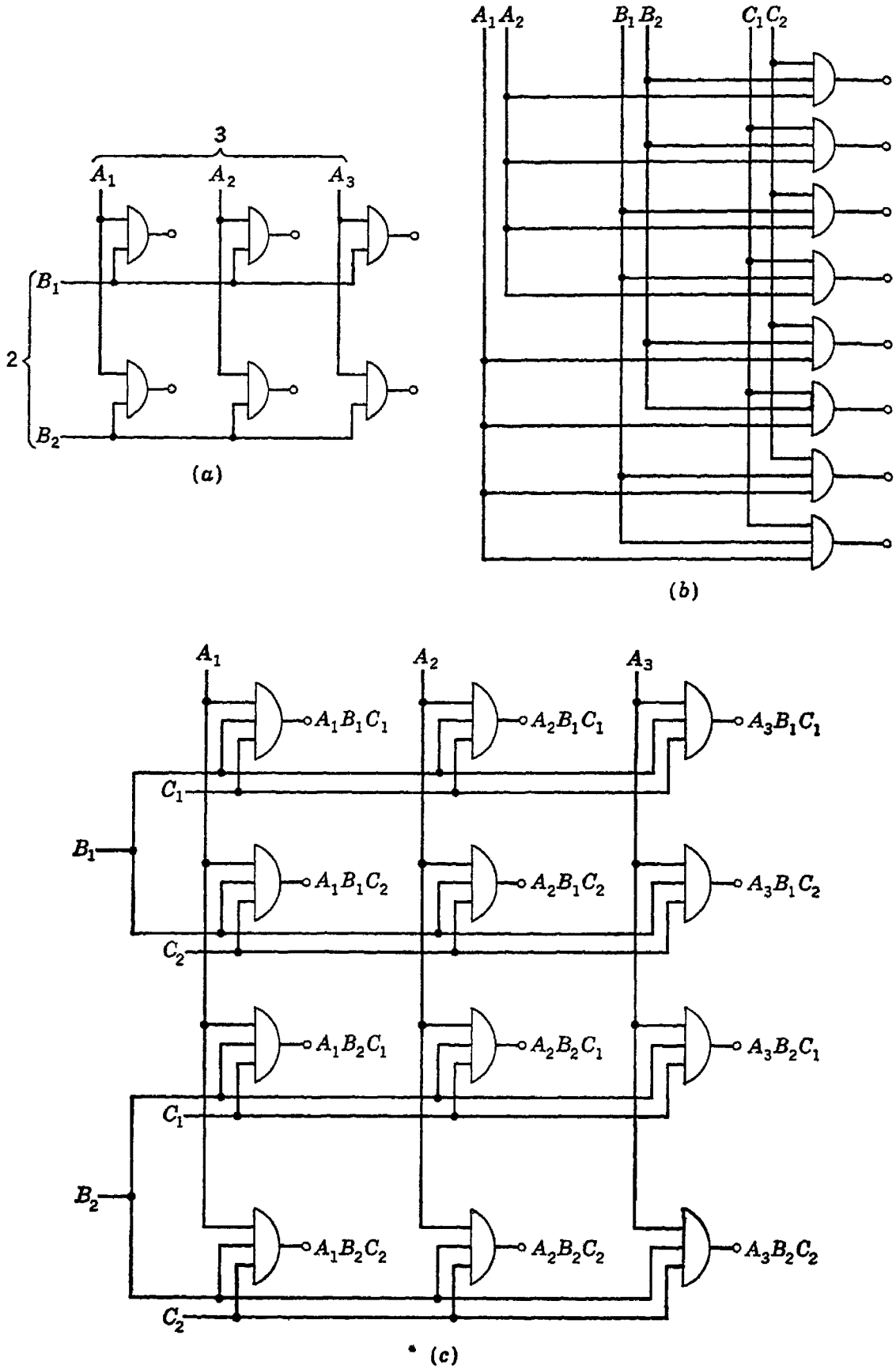



Fig. 8.21 Some simple multiplicative switches: (*a*) a 2,3 multiplicative switch, (*b*) a 2,2,2 ms., (*c*) a 2,2,3 ms.

input variables, $i_1 i_2 = 6$ realizable switching functions, and also six logic elements, each a two-input conjunction. Figure 8.21*b* shows a 2,2,2 multiplicative switch, which is also a 2-3 complete decoding net, since it has three groups of two variables each. Figure 8.21*c* shows a 2,2,3 multiplicative switch, with 7 input variables, 12 realizable functions, and 12 three-input conjunctions. A symbol for the multiplicative switch is shown in Fig. 8.22*a*, and the three multiplicative switches of Fig. 8.21 are shown symbolically in Fig. 8.22*b*, *c*, and *d*.

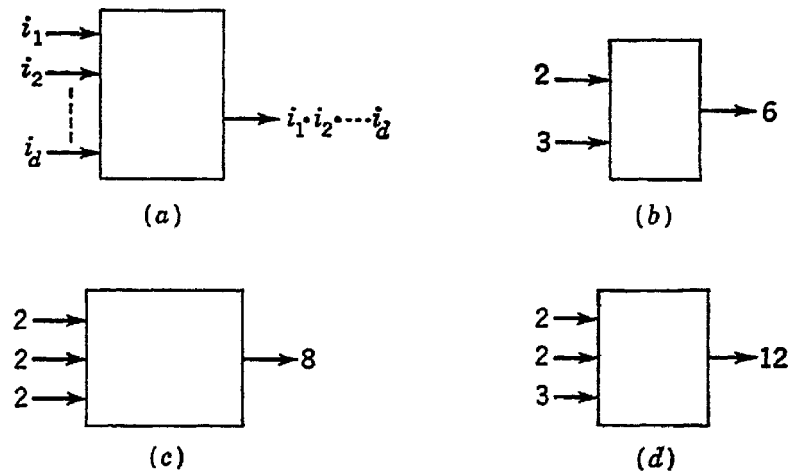



Fig. 8.22 Symbols for multiplicative switches

We continue our definitions as follows:

*Definition* 2. A multiplicative switch network (abbreviated msn.) is an interconnection of multiplicative switches (abbreviated ms.) such that some of the input variables for some of the ms. are outputs of other ms.

*Definition* 3. A *b-d* msn. is a msn. with *d* input groups, each containing *b* variables.

*Definition* 4. A *b-d* exponential switch is a ms. with *d* input groups of *b* variables each.

*Definition* 5. A *b-d* tree is a msn. that is a cascade of (1) a $b,b$ ms. driving (2) a $b^2,b$ ms. driving $\cdots$ $(d-1)$ a $b^{d-1},b$ ms. (Each $b^n,b$ ms. realizes $b^{n+1}$ switching functions.)

*Definition* 6. A *b-d* balanced msn. consists of (1) a $b^{[d/2]},b^{[(d+1)/2]}$ ms., each $b^x$ input of which is the output of (2) a $b^{[x/2]},b^{[(x+1)/2]}$ ms., and so on down to the simplest ms.

Some examples of these definitions appear in Fig. 8.23.

It is possible to make a comparison of the relative costs of these networks in terms of the equipment required for each, subject to some simplifying assumptions. For some important types of electronic devices used to implement switching functions, a significant part of the cost lies in the apparatus through which each input variable is connected to a logical element (an AND device, for example). Therefore one measure of

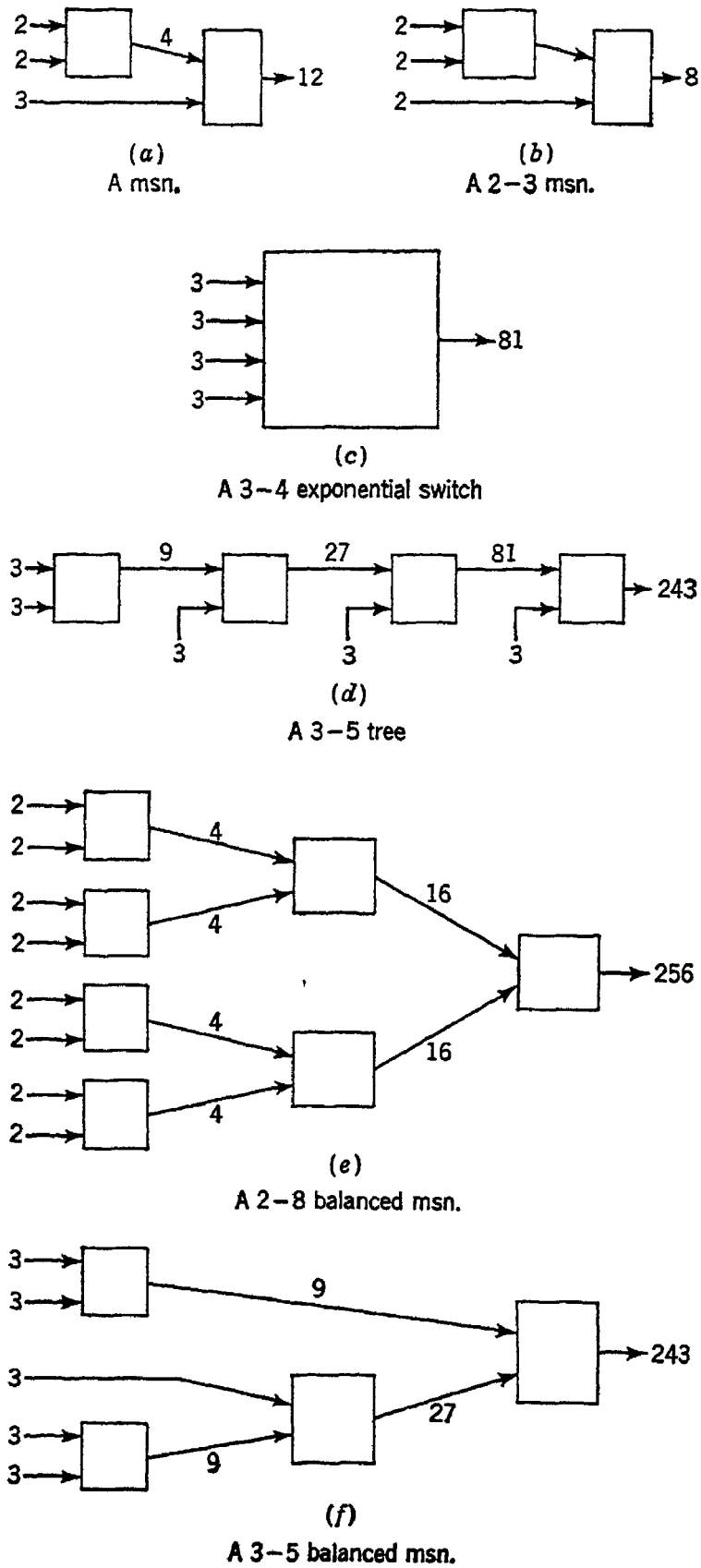



Fig. 8.23 Examples of definitions 2, 3, 4, 5, and 6: (*a*) a multiplicative switch network, (*b*) a 2-3 msn., (*c*) a 3-4 exponential switch, (*d*) a 3-5 tree, (*e*) a 2-8 balanced msn., (*f*) a 3-5 balanced msn.

cost is what may be called the "element-input count" for the circuit, that is, the summation over all the logic elements of the number of inputs to each. (This measure is not valid for relay circuits, as we have seen.)

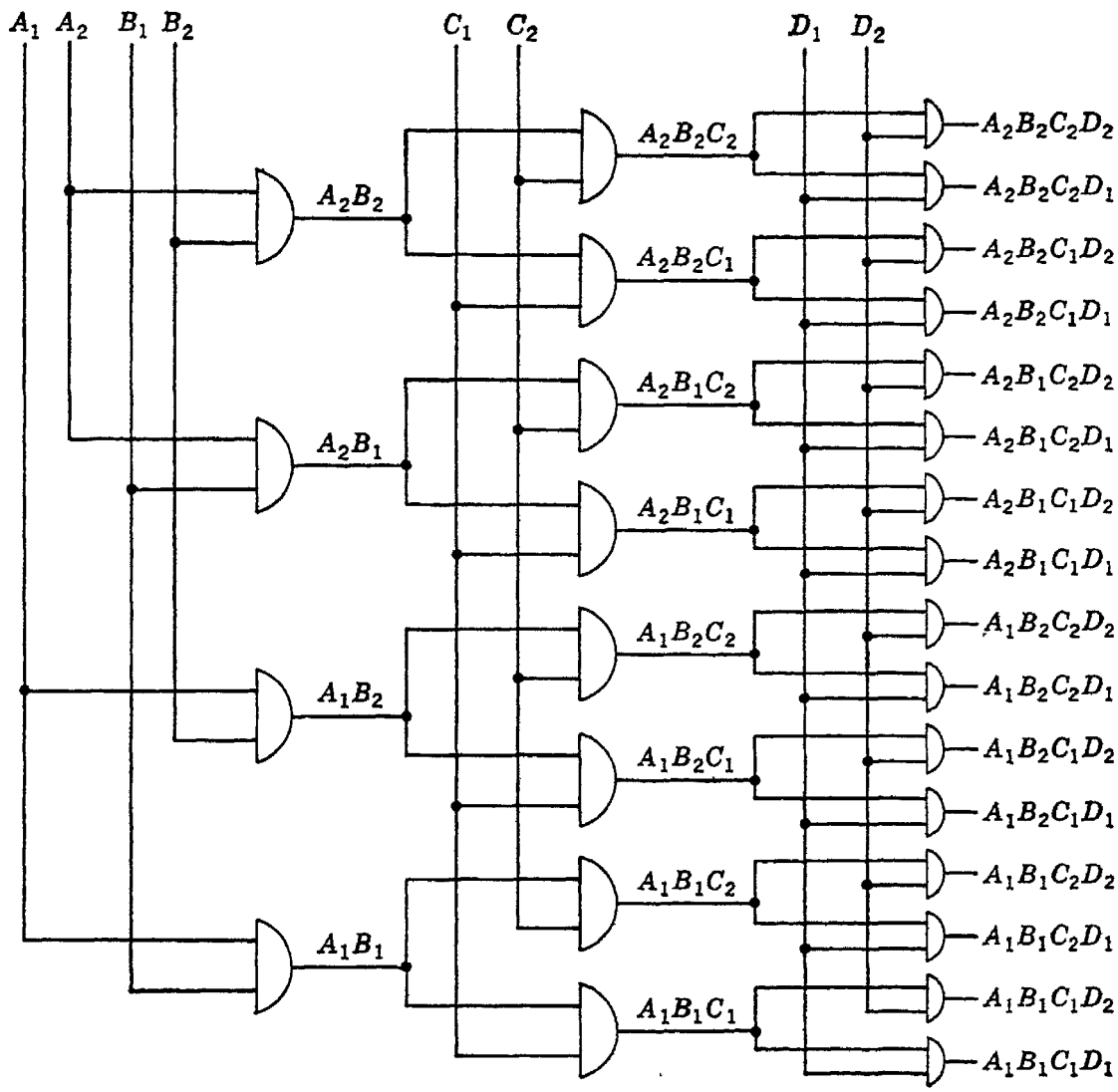



Fig. 8.24 The 2-4 tree

Burks has shown that $C_E$, the element-input count for the $b$-$d$ exponential switch, and $C_T$, the element-input count for the $b$-$d$ tree, are given by:

$$C_E = db^d \tag{8.40}$$

$$C_T = \frac{2b^{d+1} - 2b^2}{b - 1} \tag{8.41}$$

For small values of $b$ and $d$, $C_E = C_T$; but for larger values, $C_E > C_T$, and the tree is markedly lower in cost. It is of interest to note that the most efficient network, as measured by the element-input count, is the balanced msn., for which the count $C_B$ proves to be even lower than $C_T$ for all except small values of $b$ and $d$. No simple closed formula for $C_B$ is known, but a recursive relationship has been found and studied.

As an example, consider networks of the three types for the $b = 2$, $d = 4$ configuration. The exponential switch requires 16 conjunction elements, each with 4 inputs, giving $C_E = 64$. (The reader may draw the network.) The 2-4 tree, shown in Fig. 8.24, has $C_T = 56$, as can be verified from the formula or by direct count. The third form, the 2-4 balanced msn., is shown in Fig. 8.25; it has an element-input count of $C_B = 48$.

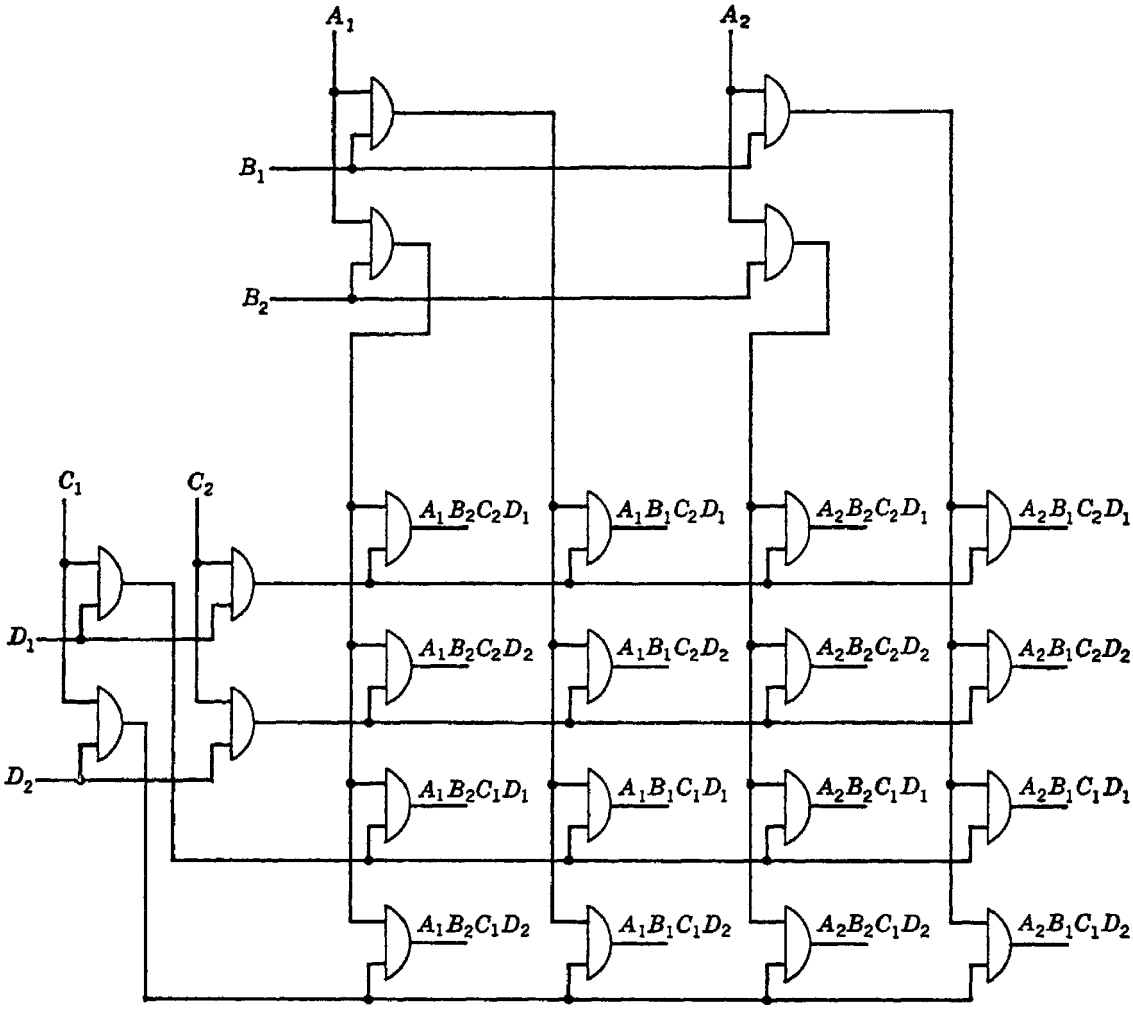



Fig. 8.25 The 2-4 balanced multiplicative switch network

## 8.9 Combinational circuits: Tools, techniques, and tricks

Let us turn now to a more general examination of the techniques useful in analysis and synthesis of combinational networks. Even though combinational networks as a class are much simpler to handle than sequential networks, practical difficulties in manipulating functions of more than a few variables are imposed by the astronomically large number of switching functions that can be formed from as few as six or seven variables.

We have already seen that, with $n = 2$ variables, there are $2^n$ or 4 different ways to assign 0 or 1 as the values of the variables. A switching function is defined by stating the condition of the two-terminal network (0 or 1) for each of the $2^n$ possible combinations of values of the variables. There are thus $2^{2^n}$ such functions possible with $n$ variables. Let us tabulate this number for a few values of $n$, as shown in Table 8.5. Merely

**Table 8.5**
**Number of switching functions of $n$ variables**

| $n$ | $2^{2^n}$ |
|---|---|
| 1 | 4 |
| 2 | 16 |
| 3 | 256 |
| 4 | 65,536 |
| 5 | 4,294,967,296 |

tabulating these functions becomes a physical impossibility for comparatively small values of $n$. Sir Arthur S. Eddington once estimated that the total number of protons and electrons in the universe was $\frac{3}{2} \times 2^{256} \times 136$.[14] This number is smaller than the number of switching functions of nine variables; so there is not enough matter in the universe even to represent each of these functions, let alone write them all down on paper.

It is interesting to note that not all of the $2^{2^n}$ functions of $n$ variables are different in the sense of being realized by different topological networks of switches. We have already seen (Sec. 8.4) that the 16 functions of two variables can be realized by only six different networks if the switches in these networks are appropriately relabeled. The number of these symmetry types of functions of $n$ variables has been investigated by D. Slepian, who has determined the figures given in Table 8.6.[15] For

**Table 8.6**
**Number of symmetry types of functions of $n$ variables**

| Number of variables $n$ | Number of symmetry types $N$ |
|---|---|
| 1 | 3 |
| 2 | 6 |
| 3 | 22 |
| 4 | 402 |
| 5 | 1,228,158 |
| 6 | 400,507,806,843,728 |

five variables there are about 4,000 times as many functions as there are symmetry types, and for six variables there are roughly $10^4$ times as many functions as symmetry types. Thus it might appear that the work

involved in manipulating, simplifying, synthesizing, or comparing networks would be greatly reduced by working only with the symmetry types. This would indeed be the case were it not for the enormous number of such functions. Thus, for six variables, we find that treating the symmetry types alone still involves us with more than $4 \times 10^{14}$ different networks.

Fortunately it is almost never necessary to make an extensive search through the functions of several variables. The important problem in combinational circuits is that of minimizing the circuit in some sense. The mere synthesis of a network that realizes a given transmission function is readily solved by a circuit having a series connection of contacts (or an AND device) for each product in the function and a parallel connection of contacts (or an OR device) for each sum in the function. Minimizing is generally very difficult, however, since so many different criteria for minimality may be used. Depending upon the type of circuit devices used, one may wish to minimize contact springs, transfer contacts, back or front contacts, cathodes, grids, transistors, power dissipation, time delay, or variously weighted combinations of these and other factors as well. As might be suspected, the minimization problem has been only partially solved. Later in this section, when we discuss Quine's method of simplification, we shall see one method of realizing a near-minimal contact network, after we have first examined some of the manipulative tools useful in network minimization.

**Tables of combinations.** One approach to network synthesis is based upon a complete tabulation of the values of the switching function for each combination of values of the input variables. Consider the example in Table 8.7. From such a table it is a simple matter to write

**Table 8.7**
**A switching function in tabular form**

| $x$ | $y$ | $z$ | $f(x,y,z)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

down a sum of products (expanded disjunctive normal form) of terms that give the switching function a value of 1; or a network path can also be diagrammed for each 1 in the function-value column. If $f(x,y,z)$ is a transmission function, we can show a network as in Fig. 8.26. An alge-

braic expression written in this manner is

$$f(x,y,z) = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} \tag{8.42}$$

Even without having defined a criterion for the simplest network, we strongly suspect that this network could be simplified, since it is likely that a reduction of the number of appearances of each variable is closely related to the minimization of some physical quantity associated with the

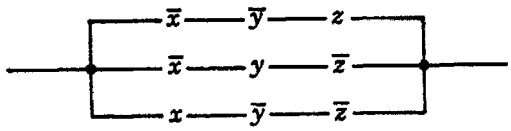



Fig. 8.26 Network corresponding to the transmission function of Table 8.7

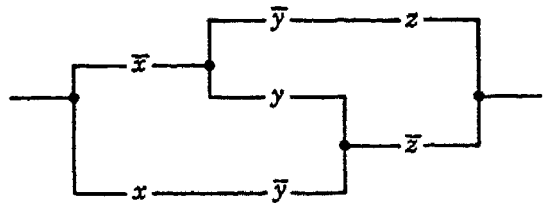



Fig. 8.27 A simplification of Fig. 8.26

elements used as switches, whether this quantity is number of cathodes, relay springs, tube envelopes, or some other. Thus, replacing the two switches $\bar{x}$ by a single switch and the two switches $\bar{z}$ by a single switch gives the network of Fig. 8.27. An interesting feature of this network is that, although it is an easy matter to write a Boolean expression for the transmission of the circuit, it is by no means clear how the circuit might be obtained from the Boolean expression. This circuit is an example of a class of circuits called bridges, since it can be redrawn in bridge form as shown in Fig. 8.28. The bridge is a particularly desirable form of network, since the function that it implements is realized with fewer total appearances of variables than there would be in an arrangement consisting only of series and parallel elements. (Of course, it is trivially evident that any network has a bridge equivalent, since a single contact can be replaced by five contacts in bridge form. The bridges of interest are those that have fewer contacts than the equivalent series-parallel circuit, and it is these that we shall henceforth mean whenever we use the term "bridge".) Unfortunately, in the present state of switching theory there is no general method of determining whether there is a bridge equivalent to a particular two-terminal circuit.[16] Synthesis methods that yield a bridge if one is possible are not well developed. Since

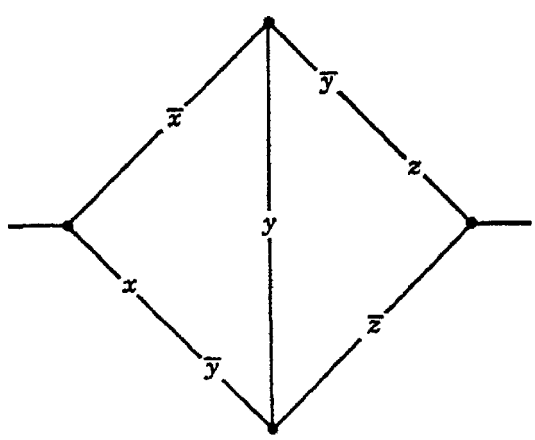



Fig. 8.28 The bridge form of Fig. 8.27

Boolean expressions contain only sums and products of variables, they do not show any shunt paths that might simplify the network, and the designer must develop his own skill at recognizing situations in which a bridge circuit might be used.

The type of equipment used for the switches will also affect the determination of the optimum network. For example, the two networks of Fig. 8.29 have the same element counts and the same Boolean expressions; yet, if the switches are relays, the arrangement on the right is to be

Fig. 8.29 Two alternative forms of a circuit

preferred. The arrangement at the left requires a separate single-pole single-throw switch for each variable; the other can employ two single-pole double-throw switches. Since a single-throw switch uses a spring for each contact, the circuit at the left has eight springs. For the double-throw switch, however, the two common contacts are carried on a single spring. Thus the circuit at the right requires only six relay contact springs. Recognition of situations of this type, too, depends on the designer's skill, and there are as yet no systematic methods for obtaining a circuit with a minimum relay-spring count.

Synthesis of a switching network is sometimes simplified by supplementary knowledge that certain combinations of input signals will never occur in the system under design. In this case the designer is at liberty to use these invalid input combinations in any way he sees fit to simplify his network. As a simple example, consider the transmission function $f(x,y,z)$, which has the values given in Table 8.8 for various combinations

**Table 8.8**
**Example of a function with invalid input combinations**

| $x$ | $y$ | $z$ | $f(x,y,z)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

of $x$, $y$, and $z$. No value of $f(x,y,z)$ is specified for $\bar{x}\bar{y}z = 1$ or $\bar{x}y\bar{z} = 1$; so we may choose $f(x,y,z)$ to be either 0 or 1 for each of these input conditions. Since these two input conditions never arise, we don't care what the network transmission is. In fact, such input conditions are

commonly known as "don't care" conditions. Examination of the table reveals that, if we let $f(0,0,1) = 1$ and $f(0,1,0) = 0$, then we have simply $f(x,y,z) = z$.

The foregoing example was simple enough to be solved almost at a glance, but if the number of don't care conditions is large, investigation of all possible ways of assigning values to the function under these conditions can be a sizable job. For instance, suppose a certain two-terminal circuit has four variables $w$, $x$, $y$, and $z$. If $n$ of the 16 input combinations are invalid, then $2^n$ switching functions of four variables must be investigated for minimality.

**Function maps.** Another method of displaying a Boolean function is to map it in a coordinate system in which the axes are the variables or

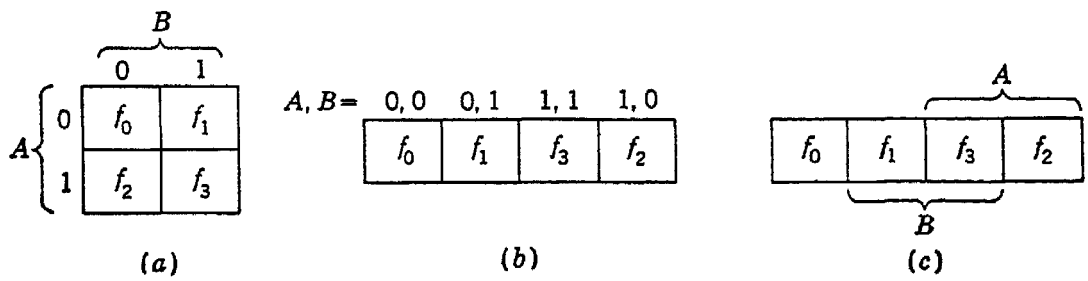



Fig. 8.30 Equivalent map forms for two variables

functions of the variables and the corresponding points in the space thus defined are assigned values of 1 or 0 as determined by the switching function. A version of this method was developed at the Computation Laboratory of Harvard University.[17] Another method, which is more widely used, was proposed originally by Veitch[18] and improved upon by Karnaugh.[19] The method may be best explained by starting from the two-variable rectangular function chart of Fig. 8.30*a*, which may be rearranged into the equivalent forms of Fig. 8.30*b* and *c*. The $f_i$ are either 0 or 1; so any of the 16 functions of $A$ and $B$ can be displayed. Note that the map in Fig. 8.30*b* is arranged so that each input combination $A,B$ differs in only one digit position from the adjacent input combinations and that the two ends of the chart are in this sense adjacent, as though the chart were wrapped around a cylinder. The map of Fig. 8.30*c* is the same as that of *b*, the brackets indicating the squares over which the indicated input variable has a value of 1.

Such maps are easily extended to three variables and to four variables in a manner that still preserves the adjacency condition just described. Figure 8.31 shows such maps. A five-variable map cannot be drawn as a planar figure preserving the adjacency condition and must be represented by 2 four-variable maps, over one of which the fifth variable is 0 and over the other of which it is 1. A six-variable map consists of 4

four-variable maps, for which four sheets of transparent plastic in a cubical array are sometimes used. This is about the practical limit to the technique, although it is certainly not impossible to use two such cubes for seven variables or four of them for eight.

Karnaugh calls each square containing a 1 in such maps a product square, or simply a $p$ square, since it corresponds to the logical product of all the variables, each appearing either with or without a bar. A function that has been mapped can be written down in sum-of-products form (expanded disjunctive normal form) by writing the algebraic sum

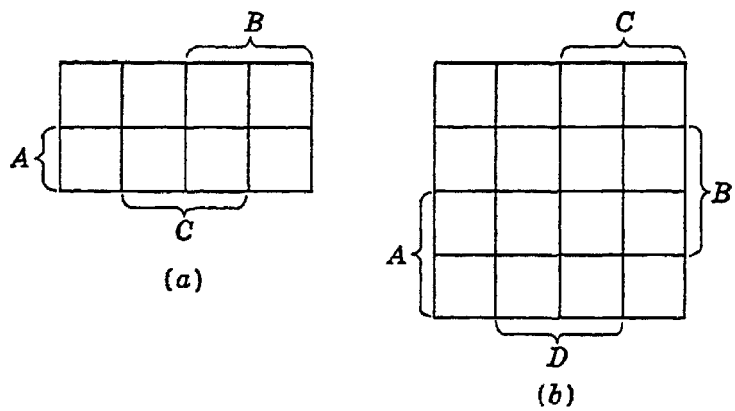



Fig. 8.31 Three- and four-variable maps

of all input combinations yielding $p$ squares. The conjunctive normal form can be obtained by writing the algebraic sum of all input combinations not yielding $p$ squares and then negating this sum by De Morgan's theorem. Of course, removing parentheses from this result by carrying out the indicated product operations gives the disjunctive normal form again.

Adjacent $p$ squares have common factors in their algebraic representations; therefore, the function that they jointly represent can be expressed more simply than in an expanded normal form. The map method is helpful in determining this factorization. A group of $p$ squares over which $m$ of the variables are not fixed is said to be an $m$-dimensional subcube. The larger the number of variables that have constant value over a given subcube, the smaller will be the dimensionality of that subcube; i.e., the smaller will be the number of variables needed to represent the function specified by the subcube. Some subcubes of one, two, and three dimensions are shown in Fig. 8.32. It will be observed that an $m$-dimensional subcube has $2^m$ squares.

In order to achieve a near-minimal sum-of-products form for the algebraic representation of a function, we should select a set of as few subcubes as possible that include all the $p$ squares and no others and have as high a dimensionality as possible. The algebraic sum of the

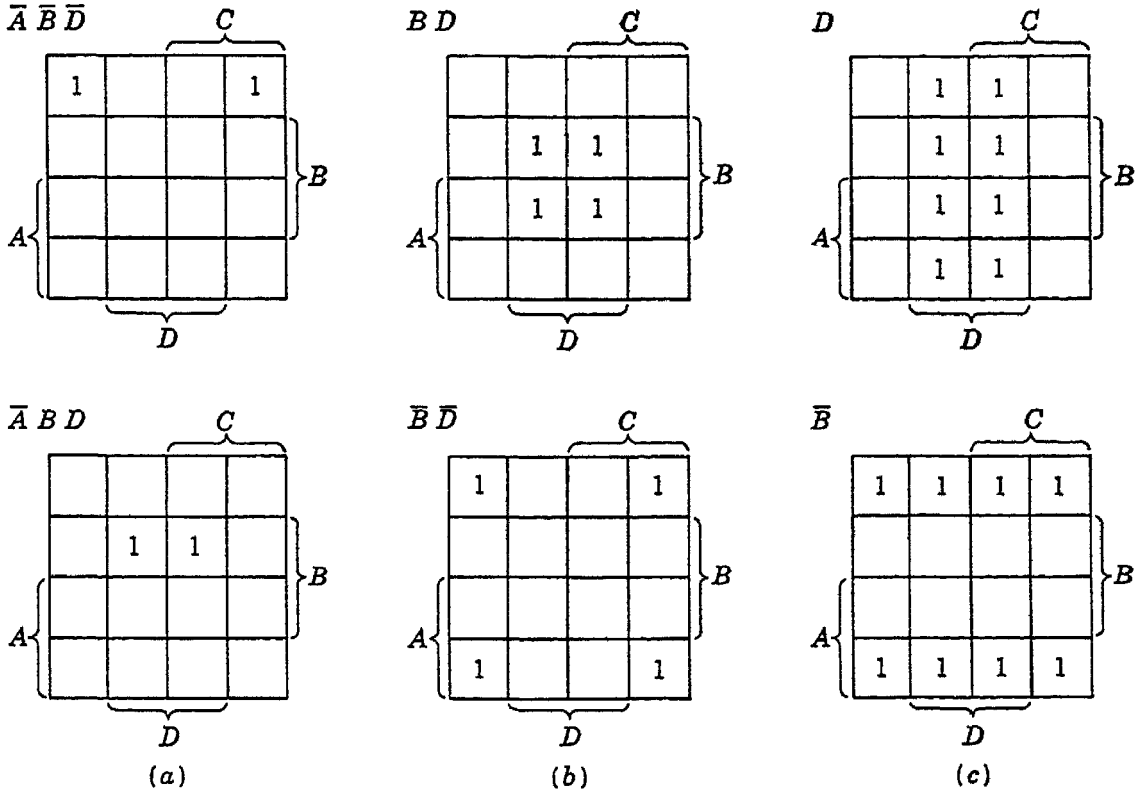
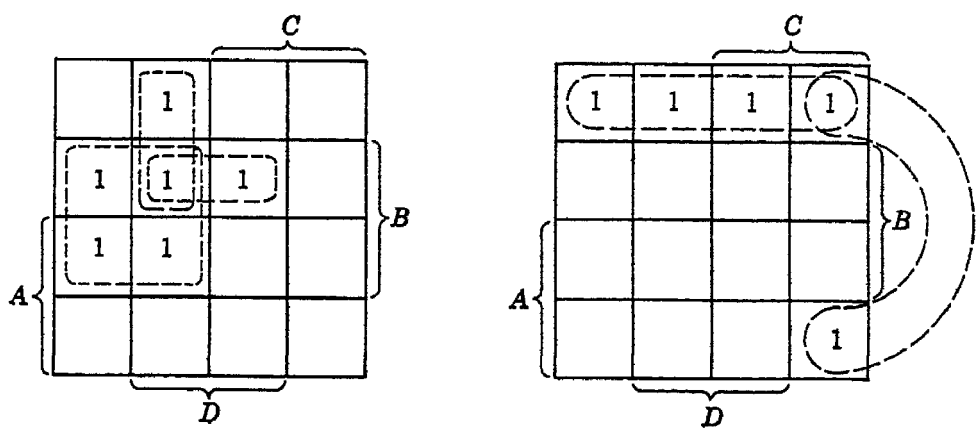



Fig. 8.32 (*a*) One-dimensional subcubes, (*b*) two-dimensional subcubes, (*c*) three-dimensional subcubes




(*a*) Function $= B\,\bar{C} + \bar{A}\,\bar{C}\,D + \bar{A}\,B\,D$ (*b*) Function $= \bar{A}\,\bar{B} + \bar{B}\,C\,\bar{D}$

Fig. 8.33 Synthesis by the map method

expressions defining these subcubes then constitutes the desired algebraic representation of the function. Examples are shown in Fig. 8.33. In Fig. 8.33*a* three subcubes have been indicated by dotted lines, and the corresponding algebraic function is written below the map. The same procedure has been followed in Fig. 8.33*b*. In each case factoring is possible and can be done either by inspection of the map or by algebraic manipulation of the sum-of-products expression. Factored expressions

for Fig. 8.33*a* are

$$B(\bar{C} + \bar{A}D) + \bar{A}\bar{C}D$$
$$\bar{A}D(B + \bar{C}) + B\bar{C}$$
$$\bar{C}(B + \bar{A}D) + \bar{A}BD$$

For Fig. 8.33*b* we obtain $\bar{B}(\bar{A} + C\bar{D})$.

A good procedure to follow in using the map method is to choose first subcubes that include any $p$ squares that can be included in only one way as parts of larger subcubes. For example, in Fig. 8.34, the four $p$ squares in the center make an attractive starting point, but, if we choose this subcube $BD$, we find that each square in it is included when we select also the subcubes $\bar{A}\bar{C}D$, $\bar{A}BC$, $ACD$, and $AB\bar{C}$, which include the $p$ squares on the edges of the map. Thus the subcube $BD$ is implied by the other four.*

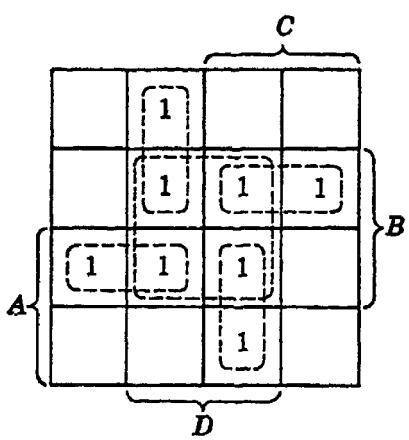



Fig. 8.34 An example showing a poor choice of subcubes

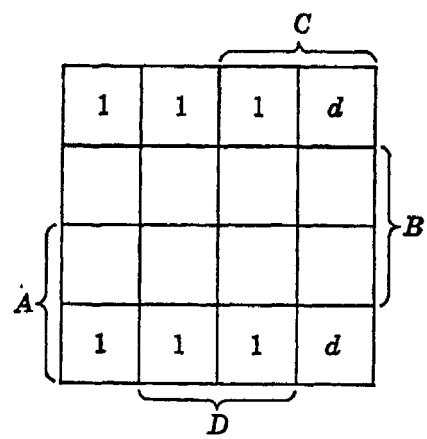



Fig. 8.35 A function map including "don't care" conditions

Don't care conditions are easily handled by the map method, since the display on the map makes it easy to see what value should be assigned to the don't cares to maximize the size of the subcubes. Figure 8.35 shows an example, the don't cares being marked by $d$. Evidently the $d$'s should be set equal to 1 to give the simple function $\bar{B}$.

**Nonnormal forms.** The manipulative techniques of the preceding few pages have all been concerned with functions in the disjunctive normal form (sum of products). An important point to bear in mind about minimizing techniques based on the normal form is that they may yield results that are not at all optimal from a circuit point of view. As Quine puts it,[20]

"This is not the most general form of the simplification problem from the point of view of engineering, since it can happen that some short

* In terminology to be introduced a few pages subsequently, $BD$ is thus not part of the core.

nonnormal formula represents a still cheaper circuit than any normal equivalent. But it will be more than enough to occupy us on the present occasion."

This last comment is directed at the general lack of methods for achieving functional simplification except in special cases. Since the final determination of the cost of a switching circuit depends so much upon the type of device used to implement the circuit, it seems probable that no single method of functional simplification will be found that will apply to all types of switching apparatus. Instead it appears that the most useful methods will yield near-minimal forms, which can then be further manipulated into forms appropriate to the physical characteristics of the apparatus used. The sum-of-products form is one of the more useful near-minimal forms, since it corresponds to what switching engineers call two-stage or two-level logic, i.e., circuit configurations in which groups of inputs are conjoined in AND elements, the outputs of which are then disjoined in OR elements, as in the example of Fig. 8.36. However, it is often possible to rearrange a sum-of-products form into factored arrangements that involve fewer total appearances of variables.

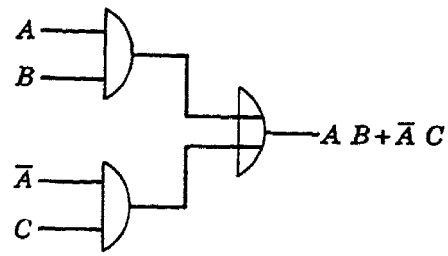


Fig. 8.36 An example of two-level logic

Should the minimization problem be that of finding an expression with the fewest possible total appearances of variables but in factored form instead of in sum-of-products form, then there is no practical method of solution. Of course, a given function can be rearranged in only a finite number of ways without introducing redundancies, and it should in theory be possible to exhaust all these arrangements in the search for a simplest. However, introduction of redundancies may aid in factorization, as in the following example given by Burkhart.[21] Consider the function

$$f(x_1,x_2,x_3,x_4,x_5,x_6) = x_1x_2x_3 + x_1x_3x_4x_5 + x_1x_2x_4x_6 + x_4x_5x_6 \quad (8.43)$$

This function is in the simplest sum-of-products form, but can be factored into

$$f = x_1x_3(x_2 + x_4x_5) + x_4x_6(x_1x_2 + x_5) \quad (8.44)$$

which is an expression containing ten total appearances of variables. If, however, some redundant factors are introduced into the original function as follows

$$f = x_1x_1x_2x_3 + x_1x_3x_4x_5 + x_1x_2x_4x_6 + x_4x_4x_5x_6 \quad (8.45)$$

then the following factorization is possible:

$$f = x_1x_2(x_1x_3 + x_4x_6) + x_4x_5(x_1x_3 + x_4x_6) \tag{8.46}$$
$$= (x_1x_2 + x_4x_5)(x_1x_3 + x_4x_6) \tag{8.47}$$

This expression contains only eight total appearances of variables.

Even without introducing redundancies, the method of exhausting all possibilities is not practical except for very simple functions. Allowing introduction of redundancy actually complicates the problem because of the infinite variety of ways in which it can be done and the lack of any systematic procedure for choosing the most suitable.

**Quine's method of simplification.**[20,22] Most of the methods that have been developed for simplifying functions in normal form are heavily indebted to the basic work done by Quine. Quine's method consists of finding all the "prime implicants" of the function and then selecting from this set a minimal set sufficient to represent the given function. A prime implicant $P$ of a function $F$ is defined as a conjunction of letters that implies $F$ and contains no shorter conjunction of letters also implying $F$. ("Implies" refers to the material-implication function of Table 8.2; i.e., $P$ implies $F$ if there is no way of assigning 1's and 0's to the letters of $P$ in such a way that $P = 1$ and $F = 0$.) Quine has shown that any simplest normal form consists of a sum of prime implicants. The following definitions will be useful in stating the method:

1. A fundamental formula is a conjunction of literals in which no letter appears twice. By a "literal" we mean any letter, either complemented or not.

2. A normal formula is a sum (disjunction) of fundamental formulas. These fundamental formulas we shall call clauses.

3. The consensus of two clauses that oppose each other in exactly one letter is the conjunction of all the other letters in both clauses. Examples are:

| | |
|---|---|
| $wx\bar{y} + wz\bar{x}$ | Consensus is $w\bar{y}z$ |
| $wx\bar{y} + w\bar{x}$ | Consensus is $w\bar{y}$ |
| $w + \bar{w}\bar{x}$ | Consensus is $\bar{x}$ |

4. A clause $R$ will be said to subsume (i.e., include) a clause $Q$ if and only if all the literals of $Q$ are among those of $R$.

Quine has shown that, by repeatedly adding the consensus of clauses and dropping any subsuming clauses until no further changes can be made, one obtains the disjunction of all the prime implicants and nothing but prime implicants. As an example of this procedure, we shall consider the following function due to Quine:

$$pqrs + \bar{p}\bar{q}rs + pq\bar{r}s + p\bar{q}\bar{r}s + pq\bar{r}\bar{s} + p\bar{q}\bar{r}\bar{s} \tag{8.48}$$

Add all possible consensuses of clauses, to get

$$pqrs + \bar{p}\bar{q}rs + pq\bar{r}s + p\bar{q}\bar{r}s + pq\bar{r}\bar{s} + p\bar{q}\bar{r}\bar{s} + pqs + p\bar{r}s + pq\bar{r} + p\bar{q}\bar{r} + p\bar{r}\bar{s} \quad (8.49)$$

Drop subsuming clauses, to get

$$\bar{p}\bar{q}rs + pqs + p\bar{r}s + pq\bar{r} + p\bar{q}\bar{r} + p\bar{r}\bar{s} \quad (8.50)$$

Again add all possible consensuses:

$$\bar{p}\bar{q}rs + pqs + p\bar{r}s + pq\bar{r} + p\bar{q}\bar{r} + p\bar{r}\bar{s} + p\bar{r} \quad (8.51)$$

and drop subsuming clauses:

$$\bar{p}\bar{q}rs + pqs + p\bar{r} \quad (8.52)$$

Since the process can no longer be applied, this last result gives all the prime implicants of the given function.

Having determined the list of all the prime implicants, one must select from this list a minimum set that implies the given function. It occasionally happens that there are certain members of the list of all prime implicants that must be included in any normal form equivalent to the given function. These particular prime implicants comprise what is known as the core. To test a prime implicant for membership in the core, we assign it a value of 1 (i.e., we assume it true) and then, subject to this assignment, examine the remaining clauses to determine whether they constitute a true formula. If they do not, then the prime implicant from which we started is part of the core. As an example of the procedure, consider the following complete set of prime implicants:

$$pq + qs + q\bar{r} + \bar{q}r\bar{s} + pr\bar{s} \quad (8.53)$$

If we assume $pq = 1$, then the rest of the formula becomes

$$s + \bar{r} + r\bar{s} \quad (8.54)$$

But this is equivalent to

$$s + \bar{r} + r \quad (8.55)$$

which is always a true function. Thus $pq$ is not part of the core. Proceeding similarly, one readily finds the core to be

$$qs + q\bar{r} + \bar{q}r\bar{s} \quad (8.56)$$

This does not necessarily complete the finding of the simplest normal form, however, since there may be clauses in the formula that are not implied by the core clauses. Having found the core, we then proceed merely by inspection to find the shortest disjunction of clauses to be added to the core clauses in order to complete the simplest normal form. In the case of the function we have just examined, either the clause $pq$

or the clause $pr\bar{s}$ must be added to the core. The clause $pq$ is shorter, so finally we find the following formula as the simplest normal form in this case:

$$pq + qs + q\bar{r} + \bar{q}r\bar{s} \tag{8.57}$$

It is interesting to note that, although these procedures are theoretically capable of reducing any function to its simplest normal form, carrying them out for functions of more than a few variables may be a lengthy

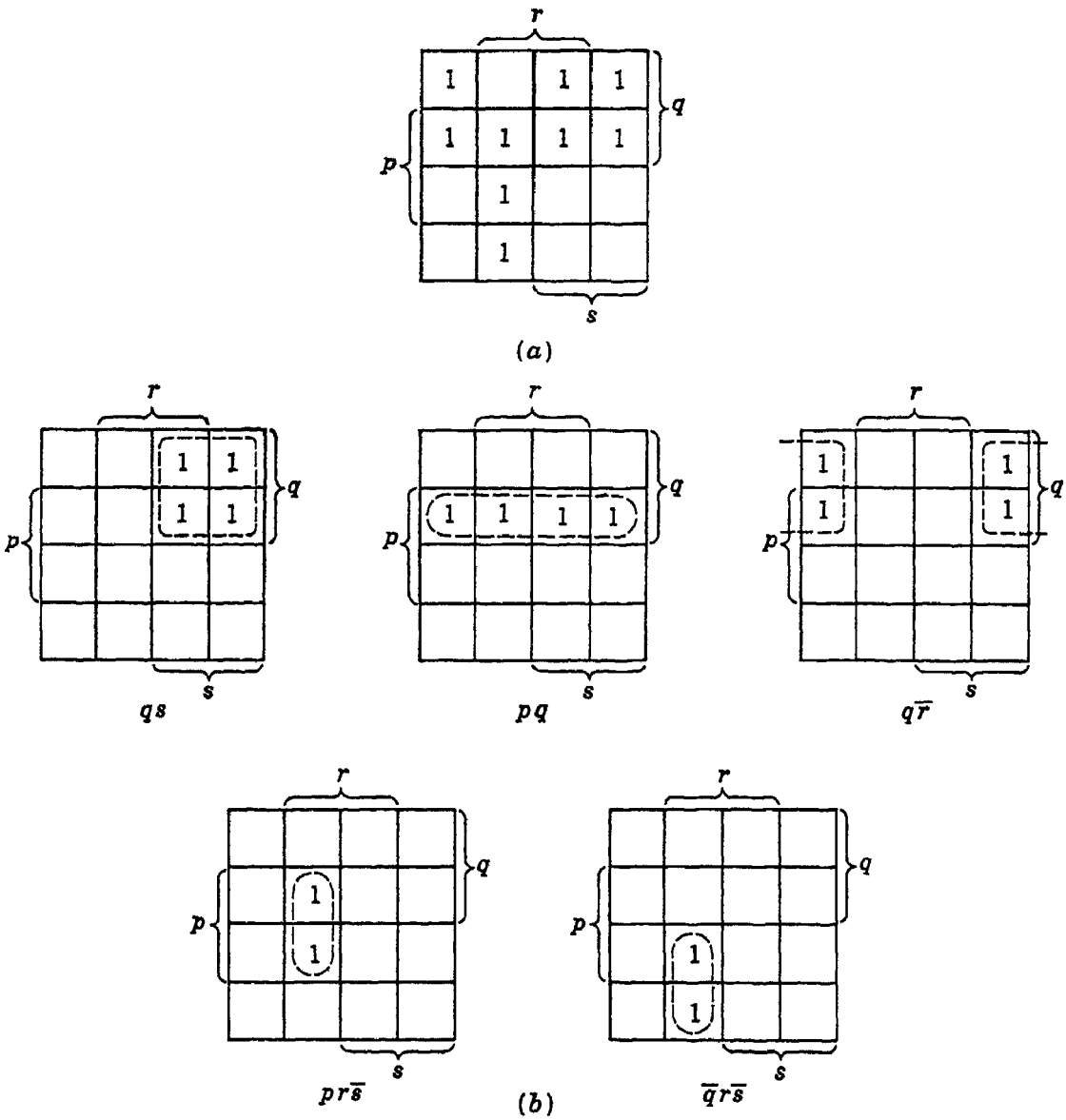



Fig. 8.37 Map of a function and its prime implicants

process. Indeed, it has been shown that, for as few as five variables, the number of prime implicants of a formula can exceed the number of fully developed clauses, that is, clauses that contain one appearance of each of the variables.

The prime implicants and the core of a function of a small number of

variables may often be found merely by inspection of the map of the function. As an example, consider the function

$$f(p,q,r,s) = \bar{p}\bar{q}r\bar{s} + p\bar{q}r\bar{s} + \bar{p}q\bar{r}\bar{s} + \bar{p}q\bar{r}s + pqrs + pqr\bar{s} + pq\bar{r}s + pq\bar{r}\bar{s} + \bar{p}qrs \quad (8.58)$$

The map of this function is shown in Fig. 8.37*a*, and separate maps of the prime implicants appear in 8.37*b* [these are the prime implicants given in Eq. (8.53)]. Certain squares of the function are part of only one prime implicant, and these prime implicants are thus part of the core. By inspection we find that the core members are those given by Eq. (8.56). These are shown in Fig. 8.38. The single square that is not part of any of the core prime implicants may be included by adding the term $pq$, which yields the result of Eq. (8.57).

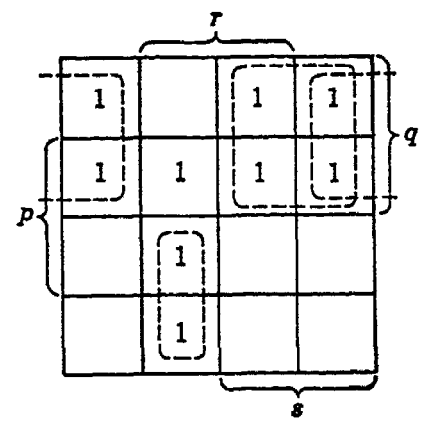



Fig. 8.38 Core prime implicants for the function of Eq. (8.53)

Another method of finding the complete set of prime implicants is due to Nelson.[23] The steps are as follows:

1. The function is put into conjunctive normal form, and all factors of the form $A + \bar{A}$ are deleted.
2. The function is then multiplied out into the disjunctive normal form.
3. All clauses containing a literal and its complement are deleted.
4. All repetitious literals are deleted.
5. All subsuming clauses are deleted.

As an example, consider the function

$$f(x,y,z) = x\bar{z} + \bar{x}y \quad (8.59)$$

First we transform to conjunctive normal form as follows:

$$\begin{aligned} f(x,y,z) &= (x\bar{z} + \bar{x})(x\bar{z} + y) \\ &= (x + \bar{x})(\bar{x} + \bar{z})(x + y)(y + \bar{z}) \\ &= (\bar{x} + \bar{z})(x + y)(y + \bar{z}) \end{aligned} \quad (8.60)$$

Multiplying out gives

$$\begin{aligned} f(x,y,z) &= (\bar{x}x + \bar{x}y + x\bar{z} + y\bar{z})(y + \bar{z}) \\ &= x\bar{x}(y + \bar{z}) + \bar{x}y + \bar{x}y\bar{z} + xy\bar{z} + x\bar{z} + y\bar{z} \end{aligned} \quad (8.61)$$

Steps 3, 4, and 5 then yield

$$f(x,y,z) = \bar{x}y + x\bar{z} + y\bar{z} \quad (8.62)$$

This result may be readily verified by Quine's method or by the use of a map.

**Algebraic and "topological" methods.** Following Quine's work, there have recently been developed a number of rather similar systematic methods for simplifying switching functions expressed in the sum-of-products normal form. All these methods ultimately consist of algebraic manipulation of the switching function. However, as an aid to visualization, some of them borrow the language and concepts of topology and have come to be known as topological methods. This is a somewhat unfortunate terminology, since it seems to suggest that there are theorems and results known in topology which are useful in switching theory but which are not readily established by an algebraic approach. Such is not the case, and theorems concerning topological properties of polyhedra (analogous to switching functions) are all proved by use of algebra.

The principal appeal of the "topological" approach is that it offers a set of familiar terms and concepts that can be identified with various kinds of switching functions and for which no adequate algebraic terminology has existed. Thus the terms of a switching function of $n$ variables, expressed in expanded disjunctive normal form, can be put into a one-to-one correspondence with a set of vertices of an $n$-dimensional cube.[24] We may identify edges, faces, and cubes of various orders with terms of the switching function in which appear respectively all but one of the variables, all but two of the variables, and so forth. However, the familiar names thus imparted are at best only names, and, although they may help the user to visualize his work, they do not help him to do it.

The algebraic steps that must ultimately be executed are facilitated by the use of a set of consistent algebraic terms. The previous lack of such a terminology is easily remedied by making the following definitions:

*Definition* 1. A product term is a conjunction of some, at least, of the switching variables describing a function. The variables may appear either complemented or uncomplemented, but no variable may appear both complemented and uncomplemented in the same product term.

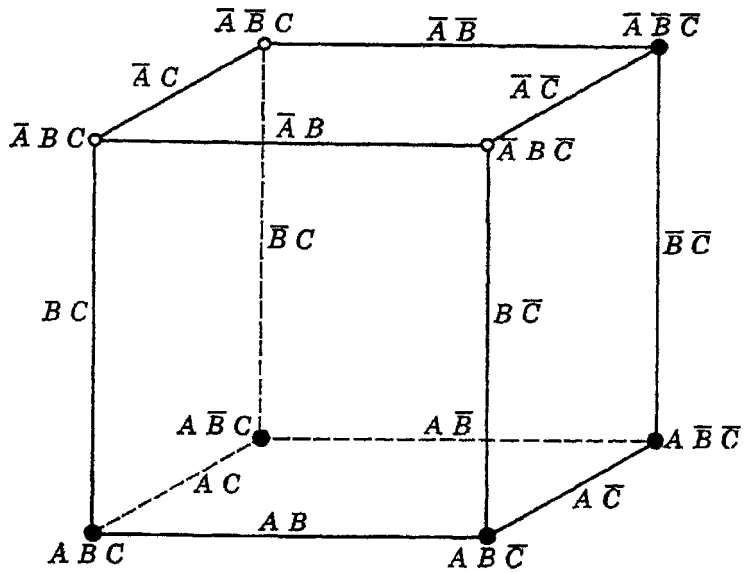
*Definition* 2. A complete product term is a conjunction of all the variables, each appearing either complemented or uncomplemented. Such a term will also be called a product term of order zero or, more simply, a $p_0$ term.

*Definition* 3. The sum of two complete product terms that differ only in one variable may be represented by a product term with one less variable. Such a term is called a product term of order 1 or, more simply, a $p_1$ term. In general, the sum of two product terms of order $j$ that differ only in the complementations of one variable may be represented by a single product term of order $j + 1$.

Having stated these definitions we may now identify $p_0$ terms with the topologist's "vertices", $p_1$ terms with his "1-cells" or edges, and, in general, $p_j$ terms with his "$j$-cubes" or "$j$-cells". Figure 8.39 shows

an example of correspondence between these two notations. Each $p_0$ term corresponds to a vertex of the three-dimensional cube, each $p_1$ term to an edge, and each $p_2$ term to a face. The $p_3$ term (or in general the $p_n$ term when there are $n$ variables) corresponds to the entire hypercube, i.e., to the identity function.

Topological approaches have been reported in the literature by Roth,[25] by Urbano and Mueller,[26] and by Harris.[27] These three works have




$$f(A, B, C) = A\,B\,C + A\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,\bar{B}\,C + \bar{A}\,\bar{B}\,\bar{C}$$

Fig. 8.39 A function of three variables and its topological representation

much in common with one another as well as with more directly algebraic treatments reported by Muller[28] and by McCluskey.[29] The synthesis problem these methods attack is that of finding a minimal-sum representation of the function. The following are typical of what have been regarded as minimal sums:

1. Sum-of-products functions that have the fewest terms of all equivalent sums and, in the event there is more than one such function, those of this class which have the fewest total appearances of variables (McCluskey).

2. Sum-of-products functions that have the least total of $+$ and $\times$ operations (Urbano and Mueller).

The solution to the problem consists in finding, by one means or another, a set of $p_j$ terms that are few in number and for which the order $j$ of each is as large as possible. This is topologically equivalent to finding a set of subcubes that (1) "cover" all the desired vertices and no others, (2) are as large as possible, and (3) are irredundant, i.e., no cube can be deleted from the set without leaving at least one vertex

uncovered. It is fairly easy to see by inspection of Fig. 8.39 that two "cubes" are involved in the function depicted, one being the two-dimensional cube with vertices $ABC$, $AB\bar{C}$, $A\bar{B}\bar{C}$, and $A\bar{B}C$ which form the surface $A$, and the other being the one-dimensional cube with vertices $A\bar{B}\bar{C}$ and $\bar{A}\bar{B}\bar{C}$ which form the edge $\bar{B}\bar{C}$. The function is then expressed simply as $A + \bar{B}\bar{C}$.

It is evident that two $p_0$ terms that differ only in one variable form an edge of the hypercube, which may be described with one less variable. Two edges that differ in only one variable define a surface (sometimes called a 2-cell), which in turn requires one less variable to define it. The problem of finding the largest dimensionality of cube that includes a given vertex is the same as that of determining the highest-order $p_j$ term that includes (or implies) a given $p_0$ term. (The resulting set of higher-order $p$ terms are Quine's *prime implicants.*) We may start with a given $p_0$ term and compare it pairwise with all other given $p_0$ terms to find whether any are present that differ from it in only one variable. This is topologically equivalent to determining for each given vertex whether any other given vertices lie at a "distance" of 1 from the given vertex. One way in which this can be done is by McCluskey's orderly arrangement of Quine's method. Each $p_0$ term in the function is represented by an $n$-bit character. Each bit position in this character corresponds to one of the $n$ variables, with 1 representing an uncomplemented variable and 0 a complemented variable.

The process then carried out is essentially the following:

1. Set $j = 0$.
2. Compare $p_j$ terms two at a time and form any possible $p_{j+1}$ terms.
3. Retain all $p_j$ terms not yielding $p_{j+1}$ terms.
4. If step 3 yielded no $p_{j+1}$ terms, sum all retained terms of all orders to get the result. Otherwise, advance $j$ by 1 and return to step 2.

The work is arranged by first listing all given $p_0$ terms in binary form in a sequence starting with a term having no 1's, then a group of terms each having a single 1, a group of terms each having two 1's, etc. The pairwise comparison need be performed only between pairs from adjacent groups, since no other pairs can differ in precisely one position. The process is repeated for the resulting $p_1$ terms, $p_2$ terms, etc., until no higher-order terms are formed. To preserve the positional significance of the binary notation, each variable removed in forming a $p_{j+1}$ term from two $p_j$ terms is replaced in McCluskey's method by a dash. (Roth uses an $x$ for this purpose, and Harris preserves a numerical value for the character by using a 2 and interpreting the resulting digit group as a ternary number which he then converts to decimal form.)

An example will clarify the procedure. Let us consider the function of five variables that has unity transmission for the following rows of

the table of combinations:

$$T = 1 \text{ for rows } 1, 2, 7, 8, 9, 10, 11, 15, 16, 17, 18, 23, 25, 26, 31 \quad (8.63)$$

This is sometimes written:

$$T = \Sigma(1,2,7,8,9,10,11,15,16,17,18,23,25,26,31) \quad (8.64)$$

which is simply a short way of writing the fully expanded disjunctive normal formula

$$\begin{aligned} T = {} & x_5x_4x_3x_2\bar{x}_1 + x_5x_4x_3\bar{x}_2x_1 + x_5x_4\bar{x}_3\bar{x}_2\bar{x}_1 + x_5\bar{x}_4x_3x_2x_1 \\ & + x_5\bar{x}_4x_3x_2\bar{x}_1 + x_5\bar{x}_4x_3\bar{x}_2x_1 + x_5\bar{x}_4x_3\bar{x}_2\bar{x}_1 + x_5\bar{x}_4\bar{x}_3\bar{x}_2\bar{x}_1 \\ & + \bar{x}_5x_4x_3x_2x_1 + \bar{x}_5x_4x_3x_2\bar{x}_1 + \bar{x}_5x_4x_3\bar{x}_2x_1 + \bar{x}_5x_4\bar{x}_3\bar{x}_2\bar{x}_1 \\ & + \bar{x}_5\bar{x}_4x_3x_2\bar{x}_1 + \bar{x}_5\bar{x}_4x_3\bar{x}_2x_1 + \bar{x}_5\bar{x}_4\bar{x}_3\bar{x}_2\bar{x}_1 \end{aligned} \quad (8.65)$$

The literal expression for the function is first written as the group of binary $p_0$ terms shown in section I of Table 8.9. The first $p_0$ term, $p_0 = 1$, is compared with those in the next group for which $p_0 = 9$, 10, 17, and 18, forming the $p_1$ terms designated as 1,9; 1,17; 2,10; 2,18; 8,9; 8,10; 16,17; and 16,18 in section II. A check mark is placed beside each $p_0$ term that is used in this manner. Completion of this process for column I yields the terms of column II, also arranged for convenience into a group of terms containing no 1's, a group containing one 1, and so forth. The terms of column II are now compared in the same manner, each one from one group being compared with each in the next lower group and a check being placed beside each $p_1$ term that yields a $p_2$ term. Only terms having dashes in the same position can be compared. (This is equivalent to saying that the numerical designators for one $p_j$ term must be exactly a power of 2 less than those of a $p_j$ term from the group below, if a $p_{j+1}$ term is to result. For example, the 1,9 term and the 17,25 term, whose designators differ by 16, yield a $p_3$ term, but not the 1,9 term and the 9,11 term.) Some duplications will occur, since terms with two dashes can be formed in two ways, terms with three dashes in three ways, etc. Thus the 1,9,17,25 term in column III is obtained from the 1,9 and 17,25 terms of column II or from the 1,17 and 9,25 terms.

When the entire process has been carried as far as possible, the terms remaining unchecked form a near-minimal representation of the function. For some purposes this may be a satisfactory point at which to leave the process, but it is usually desirable to examine the resulting set of terms to see whether any are superfluous, i.e., whether any might be deleted without altering the function. Both Quine and McCluskey have shown a tabular method of doing this, as shown in Table 8.10 for the function of Table 8.9. On the left are listed the unchecked $p$ terms from Table 8.9 and across the top the binary designators of the $p_0$ terms.

The marks opposite each $p_j$ term indicate $p_0$ terms that it represents. We must now select the minimum number of $p_j$ terms that yields at least one mark in each column. If any column in the given table has only one mark, the $p_j$ term that gives it is essential to the representation of the function; i.e., such a term is part of the core.

**Table 8.9**
**Example of McCluskey's tabular method of function simplification**

I

| $p_0$ | |
|---|---|
| 1 | 0 0 0 0 1 ✓ |
| 2 | 0 0 0 1 0 ✓ |
| 8 | 0 1 0 0 0 ✓ |
| 16 | 1 0 0 0 0 ✓ |
| 9 | 0 1 0 0 1 ✓ |
| 10 | 0 1 0 1 0 ✓ |
| 17 | 1 0 0 0 1 ✓ |
| 18 | 1 0 0 1 0 ✓ |
| 7 | 0 0 1 1 1 ✓ |
| 11 | 0 1 0 1 1 ✓ |
| 25 | 1 1 0 0 1 ✓ |
| 26 | 1 1 0 1 0 ✓ |
| 15 | 0 1 1 1 1 ✓ |
| 23 | 1 0 1 1 1 ✓ |
| 31 | 1 1 1 1 1 ✓ |

II

| $p_1$ | |
|---|---|
| 1, 9 | 0 — 0 0 1 ✓ |
| 1,17 | — 0 0 0 1 ✓ |
| 2,10 | 0 — 0 1 0 ✓ |
| 2,18 | — 0 0 1 0 ✓ |
| 8, 9 | 0 1 0 0 — ✓ |
| 8,10 | 0 1 0 — 0 ✓ |
| 16,17 | 1 0 0 0 — |
| 16,18 | 1 0 0 — 0 |
| 9,11 | 0 1 0 — 1 ✓ |
| 9,25 | — 1 0 0 1 ✓ |
| 10,11 | 0 1 0 1 — ✓ |
| 10,26 | — 1 0 1 0 ✓ |
| 17,25 | 1 — 0 0 1 ✓ |
| 18,26 | 1 — 0 1 0 ✓ |
| 7,15 | 0 — 1 1 1 ✓ |
| 7,23 | — 0 1 1 1 ✓ |
| 11,15 | 0 1 — 1 1 |
| 15,31 | — 1 1 1 1 ✓ |
| 23,31 | 1 — 1 1 1 ✓ |

III

| $p_3$ | |
|---|---|
| 1, 9,17,25 | — — 0 0 1 |
| 2,10,18,26 | — — 0 1 0 |
| 8, 9,10,11 | 0 1 0 — — |
| 7,15,23,31 | — — 1 1 1 |

Unchecked terms: $\bar{x}_5x_4x_3x_2 + \bar{x}_5x_4x_3x_1 + x_5\bar{x}_4\bar{x}_2\bar{x}_1 + x_3x_2\bar{x}_1 + x_3\bar{x}_2x_1 + x_5\bar{x}_4x_3 + \bar{x}_3\bar{x}_2\bar{x}_1$

We mark such a term with an asterisk and project all the marks in its row to a common row at the bottom. This common row is simply a tally of $p_0$ terms implied by the asterisked $p_j$ terms. If, after placing asterisks in this manner and forming the common row, we find that there are no blanks in the common row, then the asterisked terms together comprise the simplest normal form. However, it may happen (as it does in Table 8.10) that the common row is not complete. Some of the $p_j$ terms not in the core must be added to the core to obtain the simplest normal form. However, since there is a choice of ways in which to do this, we should use the highest-order $p_j$ terms that will fill out the common row. For the function of Table 8.10 there are two ways in which this can be done with

**Table 8.10**
**Table of prime implicants used in finding core terms**

| | 1 | 2 | 7 | 8 | 9 | 10 | 11 | 15 | 16 | 17 | 18 | 23 | 25 | 26 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * — — 0 0 1 | X | | | | X | | | | | X | | | X | | |
| * — — 0 1 0 | | X | | | | X | | | | | X | | | X | |
| * 0 1 0 — — | | | | X | X | X | X | | | | | | | | |
| * — — 1 1 1 | | | X | | | | | X | | | | X | | | X |
| · 1 0 0 0 — | | | | | | | | | X | X | | | | | |
| 1 0 0 — 0 | | | | | | | | | X | | X | | | | |
| 0 1 — 1 1 | | | | | | | X | X | | | | | | | |
| Common row:† | X | X | X | X | X | X | X | X | | X | X | X | X | X | X |

† $p_0$ terms implied by core terms.

$p_j$ terms of order 1. Thus the given function has two equally minimal forms:

$$T = x_3x_2\bar{x}_1 + x_3\bar{x}_2x_1 + x_5\bar{x}_4x_3 + \bar{x}_3\bar{x}_2\bar{x}_1 + \bar{x}_5x_4x_3x_2 \tag{8.66}$$

and

$$T = x_3x_2\bar{x}_1 + x_3\bar{x}_2x_1 + x_5\bar{x}_4x_3 + \bar{x}_3\bar{x}_2\bar{x}_1 + \bar{x}_5x_4x_3x_1 \tag{8.67}$$

**Circuit realization by Shannon's method.**[11] After having obtained a formula in simplest normal form, the designer must next realize this function with a network. As we have seen, the straightforward series-parallel realization may not be best, and, in the case of contact networks, some method capable of yielding bridge circuits is desirable. Two such methods are described here. The first is due to Shannon. In this method one forms two networks, called the $M$ network and the $N$ network, as shown in Fig. 8.40. The $M$ network involves only $m$ of the variables and has one input terminal and $2^m$ output terminals. These terminals are totally disjunctive; that is, no output terminal connects to any other output terminal through the $M$ network under any conditions. The $N$ network uses only the remaining variables of the given transmission function and is such as to realize all possible functions of these variables. If terminal $j$ of the $M$ network is connected to terminal $k$ of the $N$ network, then, as Shannon has shown, the transmission from $a$ to $b$ is given by

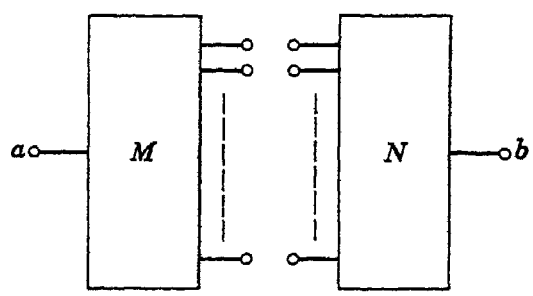



Fig. 8.40 Shannon's network-synthesis method

$$T_{ab} = \sum_{j,k} T_{aj}T_{kb} \tag{8.68}$$

where $T_{aj}$ is the transmission from $a$ to $j$, $T_{kb}$ is the transmission from

$k$ to $b$, and the summation is extended over all connections of any terminal $j$ to any terminal $k$.

In the case of the symmetrical switching function of Eq. (8.42) for which we have already found a bridge circuit by inspection, we can find one also by Shannon's method. The $M$ and $N$ networks and their interconnections are shown in Fig. 8.41. This circuit may be redrawn to give the more conventional appearance of a bridge, as shown in Fig. 8.42.

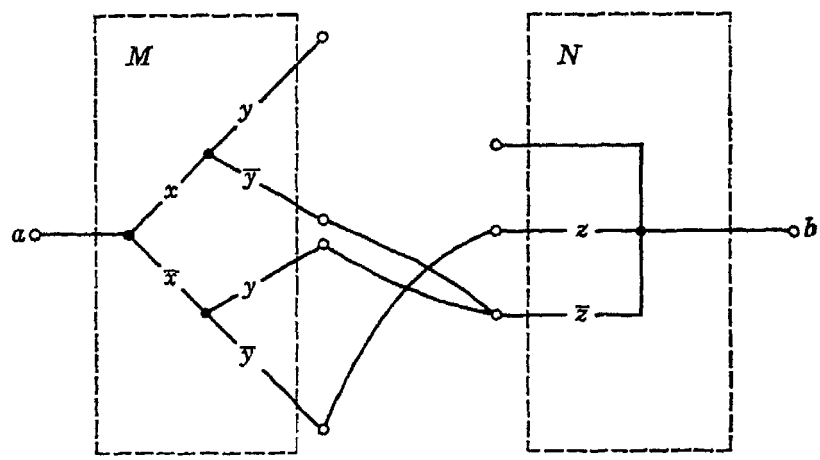



Fig. 8.41 Shannon's method applied to Eq. (8.42)

Shannon has shown that, if $n$ is the total number of variables in the switching function, then $m$, the number of variables to be used in the $M$ network, is given by the following table:

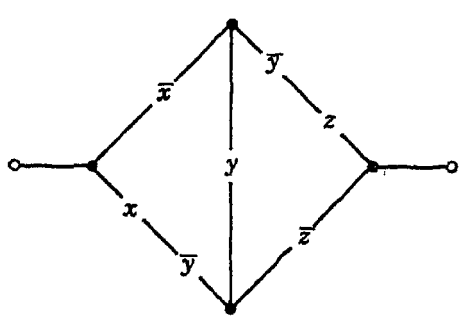



Fig. 8.42 Rearrangement of Fig. 8.41

| | | |
|---|---|---|
| For | $3 < n \leq 6$ | $m = 1$ |
| | $6 < n \leq 11$ | $m = 2$ |
| | $11 < n \leq 20$ | $m = 3$ |
| | $20 < n \leq 37$ | $m = 4$, etc. |

It has also been shown that any switching function of $n$ variables can be constructed using no more than $2^{n-1} + 16$ contacts. (Shannon's original value for this limit was $2^{n-1} + 18$. This followed from a division of the $n$ total variables in such a way that $n - 2$ of them were used in the $M$ network and the remaining two in the $N$ network. A complete tree for $n - 2$ variables uses $2^{n-1} - 2$ contacts. Shannon's $N$ network for realizing all functions of two variables employed 20 contacts, but it has since been shown that this can be done with a network using only 18 contacts, as shown in Fig. 8.43. This is a minor but nonetheless interesting improvement.)

Once a choice of the value $m$ has been made for a given function, the problem still remains of deciding which $m$ variables should be selected

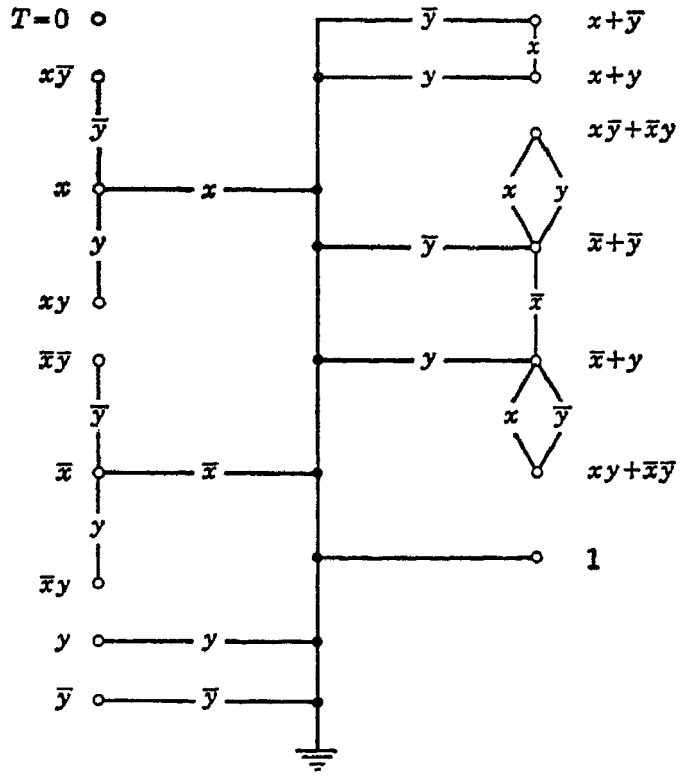



Fig. 8.43 Minimal network realizing all 16 functions of two variables

for the $M$ network. Unfortunately there exists no formal procedure for optimizing this choice. In broad terms, one uses Eq. (8.37) or one of its related forms to expand the function about some subset $m$ of its variables, seeking to find a particular set of $m$ variables that makes the transmissions $T_{aj}$ the same for a large set of terms in the expanded function. Consider, for example, the function

$$f(x,y,z) = x\bar{y} + \bar{x}y + z \tag{8.69}$$

Expanding this about $x$ and $y$, we obtain

$$f(x,y,z) = \bar{x}\bar{y}f(0,0,z) + \bar{x}yf(0,1,z) + x\bar{y}f(1,0,z) + xyf(1,1,z) \tag{8.70}$$
$$= \bar{x}\bar{y}z + \bar{x}y + x\bar{y} + xyz \tag{8.71}$$

The $M$ and $N$ networks for this form of the function are shown in Fig. 8.44.

Alternatively, the function may be expanded about $x$ and $z$ or about $y$ and $z$. Networks of different appearance (but equivalent behavior) will be found. The reader is urged to derive these networks as an exercise. It will be observed that the resulting networks all contain redundancies and can be simplified. Indeed, none of the resulting networks contains as few switches as the network that would have been derived by a direct series-parallel implementation of Eq. (8.69). This serves to emphasize that, although the Shannon method is capable of yielding bridge networks, it makes no pretension of yielding minimal networks.

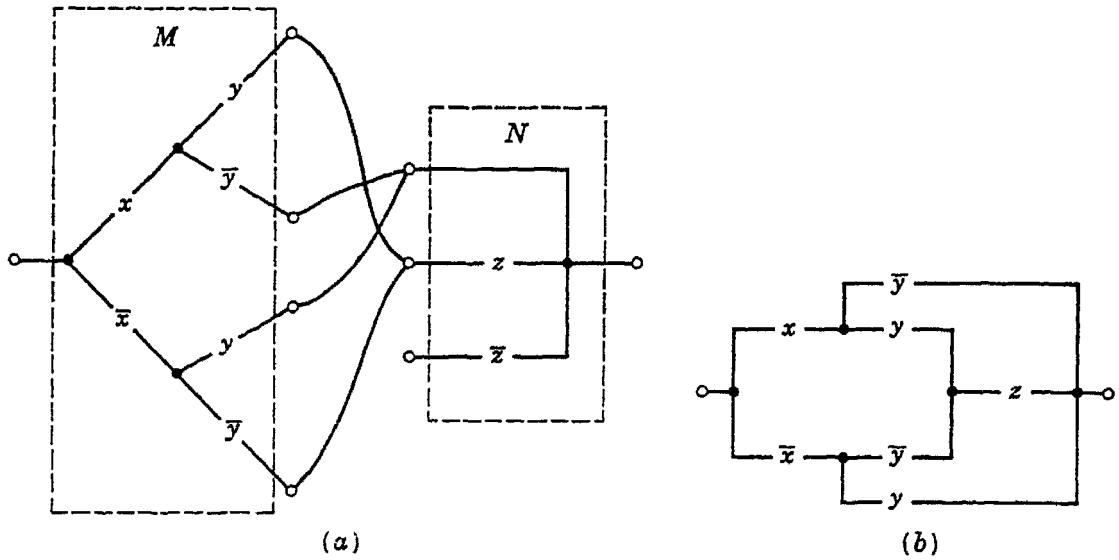



Fig. 8.44 (*a*) *M* and *N* networks corresponding to Eq. (8.71), (*b*) same, straightened out

**Circuit realization by matrix methods.**[30,31] A second method which may be used to derive a contact network from an algebraic expression and which is capable of yielding bridge circuits is the matrix method. A path-closing device (i.e., a switch or a relay) is characterized by two terminal points between which a path does or does not exist as determined by the logic of the device. An interconnection of such devices, therefore, possesses an array of terminal points connected directly or indirectly through various switching devices. It is apparent that a convenient way to present information about the states of all possible paths among a set of $n$ terminals is to arrange the transmission functions (or the hindrances) in an $n \times n$ matrix in such a way that $t_{ij}$ specifies the transmission between terminals $i$ and $j$. (Voltage-producing elements, by contrast, present only a single terminal and a common reference. A matrix for such a system would consist of only a single row or column with a switching function entered opposite each terminal number.)

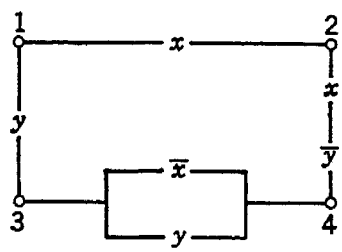
Several types of matrices have been defined for use with multiterminal networks. One of these, called the "output matrix", is the matrix we have just described; it gives the array of transmission functions from each terminal by all possible paths to each other terminal. Figure 8.45 shows a simple circuit and its output matrix. Since each terminal is always connected to itself, the elements on the major diagonal are all 1, and, since the elements are bilateral, the matrix is symmetrical.

Although the output matrix states the function to be synthesized, it is not in a form that yields a circuit configuration directly. For this we seek what is called a "primitive connection matrix" $P$, a matrix giving the transmission functions directly between all nodes of the circuit, whether terminal nodes or nonterminal nodes. The elements $t_{ij}$ of this

matrix are the transmissions directly from node $i$ to node $j$ and not via any other nodes. For example, we designate the node in the direct branch from terminal 2 to terminal 4 in Fig. 8.45 as node 5 and obtain the primitive connection matrix below:

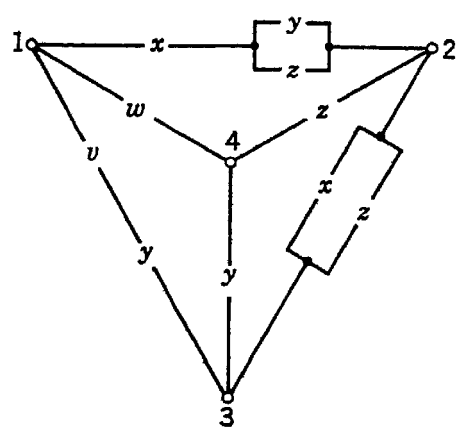
$$P = \begin{bmatrix} 1 & x & y & 0 & 0 \\ x & 1 & 0 & 0 & x \\ y & 0 & 1 & \bar{x}+y & 0 \\ 0 & 0 & \bar{x}+y & 1 & \bar{y} \\ 0 & x & 0 & \bar{y} & 1 \end{bmatrix} \tag{8.72}$$

Every element $t_{ij}$ of the primitive connection matrix $P$ consists of a sum of one or more individual variables, either complemented or not. Since




$$F = \begin{bmatrix} 1 & x & y & y+x \\ x & 1 & xy & x \\ y & xy & 1 & \bar{x}+y \\ y+x & x & \bar{x}+y & 1 \end{bmatrix}$$

Fig. 8.45 A four-terminal network and its output matrix




$$C = \begin{bmatrix} 1 & x(y+z) & vy & w \\ x(y+z) & 1 & x+z & z \\ vy & x+z & 1 & y \\ w & z & y & 1 \end{bmatrix}$$

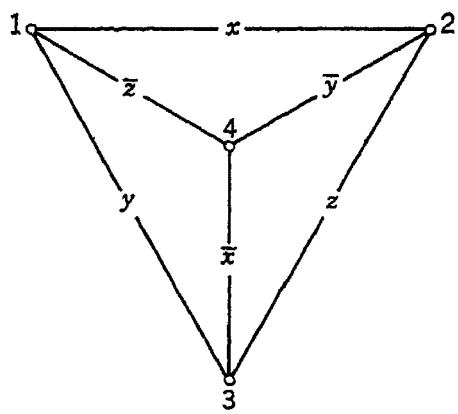
Fig. 8.46 A network and a connection matrix

a node is any junction of two or more elements and since the primitive connection matrix includes all nodes, it follows that this matrix must contain no product terms.

A less fully developed form of matrix is one called simply a connection matrix. In such a matrix not all the nonterminal nodes are included, but only enough of them to allow construction of a matrix all of whose elements are switching functions of two-terminal circuits. A circuit and one of its possible connection matrices are shown in Fig. 8.46. Although this circuit contains six nodes, including the terminals 1, 2, and 3, only

four of them are included in the connection matrix shown. Some of the elements of the connection matrix, therefore, contain products.

It is useful to be able to expand an output matrix into a connection matrix by introducing nonterminal nodes or to reduce a primitive connection matrix by eliminating some of the nonterminal nodes. First let us consider elimination of nodes. Briefly, to remove from the matrix any nonterminal node $r$, we add to each element $t_{ij}$ the product $t_{ir}t_{rj}$ and then delete the $r$th row and the $r$th column from the matrix. We thus




$$C = \begin{bmatrix} 1 & x+\bar{y}\bar{z} & y+\bar{x}\bar{z} \\ x+\bar{y}\bar{z} & 1 & z+\bar{x}\bar{y} \\ y+\bar{x}\bar{z} & z+\bar{x}\bar{y} & 1 \end{bmatrix}$$

Fig. 8.47 A four-node circuit and its reduced connection matrix

substitute for each element $t_{ij}$, which represented a connection from node $i$ to node $j$ directly, an element $t_{ij} + t_{ir}t_{rj}$ which represents the original direct connection paralleled by a connection via node $r$. When this has been done for each $t_{ij}$, the connections to node $r$ have been completely included in other entries, and the row and column of node $r$ may be eliminated. (This does not mean that the node has been eliminated from the circuit. In fact, a circuit drawn from a partially reduced connection matrix may have more nodes than the circuit from which the primitive connection matrix was formed.) A matrix successively reduced in this way until all nonterminal nodes have been deleted is called a reduced connection matrix.

Nodes that have been removed may be reinserted by essentially the reverse process. Deletion of a node is evidenced by the presence of a product term in a matrix element, and, to restore the node, the product term is eliminated, one factor being placed in the same row in a new column and the other in the same column in a new row. Of course, the factors must be located in positions where they maintain the symmetry of the matrix. An example is the expansion of the matrix

$$C = \begin{bmatrix} 1 & xy \\ xy & 1 \end{bmatrix} \tag{8.73}$$

which can be done in either of the following two ways:

$$C = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ x & y & 1 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 & y \\ 0 & 1 & x \\ y & x & 1 \end{bmatrix} \tag{8.74}$$

The only difference between these two forms is in the matter of the numerical designation of the nodes. The circuit in each case is a series connection of $x$ and $y$.

If a node is a junction of switching paths connecting to only two other nodes, its elimination will create product terms in two symmetrically located elements of the partially reduced matrix. If a node connects to three other nodes, its elimination will produce six product terms, and, although it can be restored by the process just described, more care is necessary. Consider the circuit of Fig. 8.47, from which node 4 may be eliminated to give the reduced connection matrix shown. We may expand this matrix again by splitting product terms. First let us do this in an inefficient manner, to see the results of an inferior way of expanding. If each product term is split with no reference to any others, we obtain the expanded matrix of Eq. (8.75). Canceled terms in the matrix are those which were split to form a new row and a new column.

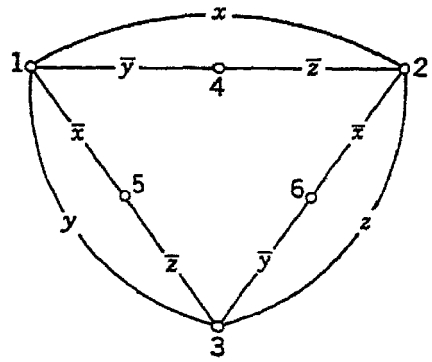



Fig. 8.48 Network drawn from the matrix of Eq. (8.75)

$$\begin{array}{c} \begin{array}{cccccc} (1) & (2) & (3) & (4) & (5) & (6) \end{array} \\ \left[\begin{array}{ccc|c|c|c} 1 & x + \cancel{\bar{y}\bar{z}} & y + \cancel{\bar{x}\bar{z}} & \bar{y} & \bar{x} & 0 \\ x + \cancel{\bar{y}\bar{z}} & 1 & z + \cancel{\bar{x}\bar{y}} & \bar{z} & 0 & \bar{x} \\ y + \cancel{\bar{x}\bar{z}} & z + \cancel{\bar{x}\bar{y}} & 1 & 0 & \bar{z} & \bar{y} \\ \hline \bar{y} & \bar{z} & 0 & 1 & 0 & 0 \\ \hline \bar{x} & 0 & \bar{z} & 0 & 1 & 0 \\ \hline 0 & \bar{x} & \bar{y} & 0 & 0 & 1 \end{array}\right] \end{array} \tag{8.75}$$

The network corresponding to this matrix appears in Fig. 8.48. Although this network is electrically identical with the network from which we started, it uses nine contacts, which is three more than the original circuit used.

A better way to expand the matrix is to note that in row 1 the two product terms have a common factor. The same is true of the two product terms in column 3. Thus, if we split these terms properly, we need make only one entry for each common factor instead of two. The matrix

may thus be expanded as shown in Eq. (8.76). The four-node circuit

$$C = \begin{array}{c} \begin{array}{cccc} (1) & (2) & (3) & (4) \end{array} \\ \begin{bmatrix} 1 & x + \cancel{\bar{y}\bar{z}} & y + \cancel{\bar{x}\bar{z}} & \bar{z} \\ x + \cancel{\bar{y}\bar{z}} & 1 & z + \bar{x}\bar{y} & \bar{y} \\ y + \cancel{\bar{x}\bar{z}} & z + \cancel{\bar{x}\bar{y}} & 1 & \bar{x} \\ \bar{z} & \bar{y} & \bar{x} & 1 \end{bmatrix} \end{array} \tag{8.76}$$

that follows from this matrix is the original six-contact circuit. Of course, the circuit of Fig. 8.48 could be rearranged in such a way as to eliminate the superfluous branches and nodes, but it is desirable to employ the matrix in such a way as to make this circuit rearrangement unnecessary.

As a final example of the use of matrices, consider again the transmission function

$$T = x\bar{y}\bar{z} + \bar{x}y\bar{z} + \bar{x}\bar{y}z \tag{8.77}$$

The connection matrix for this function is shown in Eq. (8.78) and its expansion in Eq. (8.79).

$$C = \begin{bmatrix} 1 & x\bar{y}\bar{z} + \bar{x}y\bar{z} + \bar{x}\bar{y}z \\ x\bar{y}\bar{z} + \bar{x}y\bar{z} + \bar{x}\bar{y}z & 1 \end{bmatrix} \tag{8.78}$$

$$C = \begin{array}{c} \begin{array}{cccccc} (1) & (2) & (3) & (4) & (5) & (6) \end{array} \\ \begin{bmatrix} 1 & \cancel{x\bar{y}\bar{z}} + \cancel{\bar{x}y\bar{z}} + \cancel{\bar{x}\bar{y}z} & \bar{x} & x & 0 & 0 \\ \cancel{x\bar{y}\bar{z}} + \cancel{\bar{x}y\bar{z}} + \cancel{\bar{x}\bar{y}z} & 1 & \cancel{y\bar{z}} + \cancel{\bar{y}z} & \cancel{\bar{y}\bar{z}} & \bar{y} & \bar{z} \\ \bar{x} & \cancel{y\bar{z}} + \cancel{\bar{y}z} & 1 & 0 & z & y \\ x & \cancel{\bar{y}\bar{z}} & 0 & 1 & \bar{z} & 0 \\ 0 & \bar{y} & z & \bar{z} & 1 & 0 \\ 0 & \bar{z} & y & 0 & 0 & 1 \end{bmatrix} \end{array} \tag{8.79}$$

The network of Fig. 8.49 follows at once. This network is only superficially different from that of Fig. 8.42, the difference being due to the fact that the function is totally symmetrical; i.e., if $x$ is replaced by $y$, $y$ by $z$, and $z$ by $x$ throughout the original function, the function is unchanged.

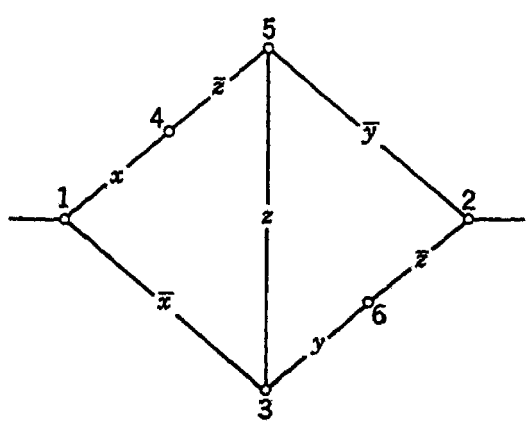



Fig. 8.49 Network drawn from the expanded matrix of Eq. (8.79)

A significant feature of this circuit is that it is a bridge circuit, i.e., a non-series-parallel circuit. As Hohn[30,31] puts it, the matrix method "is not prejudiced in favor of series-parallel circuits." Therefore, it has possibilities of being a

very powerful tool for network synthesis. As yet, however, little is known about the minimizing properties of the matrix method or about ways in which it should be used to give near-minimal circuits. Its usefulness as a tool should be greatly extended by the development of methods for handling large switching matrices on digital computers, since a considerable amount of work and care are involved in manipulating matrices for even just a few variables.

Published papers have carried the subject further than we need go here, and the reader who is interested in a fuller exploration of the topic will find it well discussed in the literature.

## 8.10 Sequential circuits

Our considerations of switching circuits up to this point have been limited entirely to combinational circuits: circuits in which the value of the switching function at any time is wholly determined by the values of the input variables at that time. It is always possible to analyze and to synthesize such circuits, although, as we have seen, methods of optimizing them are as yet very poorly developed. Sequential circuits, i.e., circuits in which the value of the switching function at any time is determined in part by the values of the input variables and in part by their previous history, are much more difficult to treat. There presently exist no straightforward methods of analysis and synthesis that can be applied to all sequential systems, and the problem of optimization can be treated only in certain special cases. Thus sequential circuits present to the engineer and the mathematician problems of much greater challenge and complexity than those of combinational circuits.

In its most general form the theory of sequential circuits becomes the same as the theory of automata and as such has been treated with a high degree of mathematical sophistication and abstraction.[32,33] These treatments, however, have not been concerned with details of switching circuitry. For our purposes, it is better to proceed from an elementary description of the sequential circuit to a discussion of some of the simpler techniques for handling it.

The basic distinction between combinational and sequential circuits is that the sequential circuit possesses "memory" and the combinational circuit does not. Figure 8.50 shows two simple examples of sequential circuits. In Fig. 8.50*a*, if switch $x_1$ is open or closed, the circuit from $a$ to $b$ has the same transmission as $x_1$ provided $x_2$ has never been closed. However, if $x_2$ is closed, relay $y$ operates and holds itself closed through the circuit on its own contacts even after $x_2$ has been opened. Thus the contact in series with $x_1$ is held open. Circuit 8.50*b* is an active circuit, which produces an output signal if it receives an input signal $x_1$ and if

the last previous signal applied to the memory element was $x_3$. If the last previous memory input was $x_2$, then an inhibiting signal is applied to the AND gate and $x_1$ produces no output. [The electrical nature of the memory element in this circuit need not concern us here. There exist many different ways of implementing it, and the reader with some background in electronic circuitry will recognize that the electronic flip-flop, with SET ($S$) and RESET ($R$) inputs, is one way in which it might be done.]

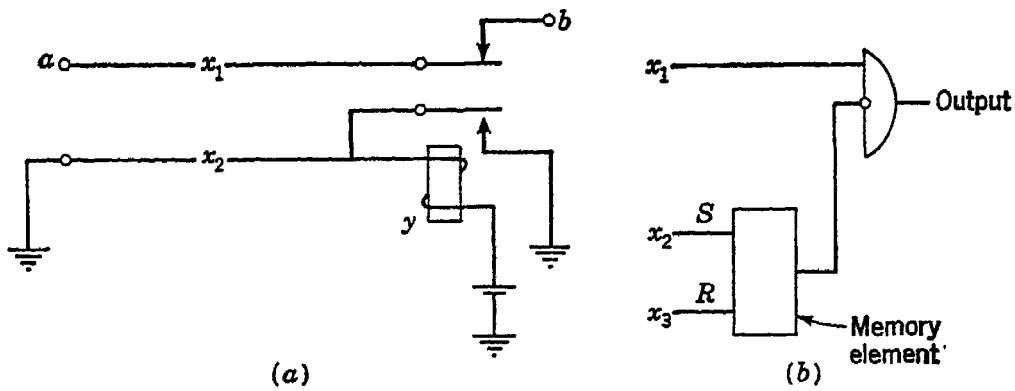



Fig. 8.50 Two sequential circuits

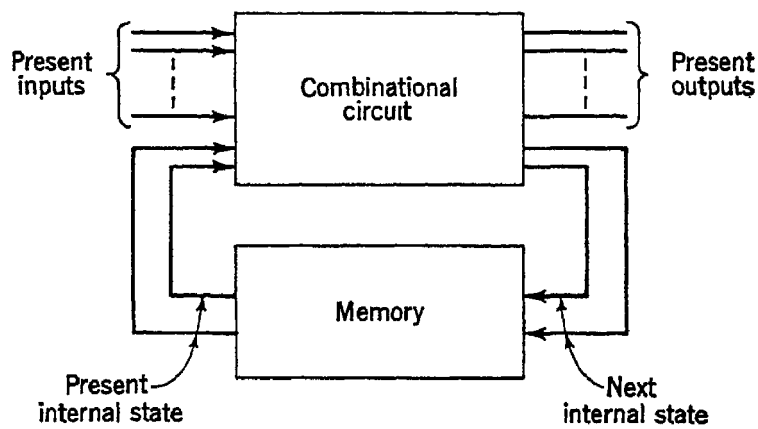



Fig. 8.51 Block diagram of the generalized sequential circuit

The memory elements of a sequential circuit make it possible for the circuit to have different internal states $S$. If there are $p$ memory elements (assumed binary), then $2^p$ internal states can be defined. Not all of these are necessarily possible states for the network, however, since its structure may be such as to prevent certain states from being reached. The output of a sequential circuit (or, more properly, the output combination, since there may be several outputs) is determined by its present internal state and by its present input combination. Likewise, the present internal state and the present input combination determine what the next internal state will be. A block diagram of these relations is shown in Fig. 8.51. If all the memory elements are gathered into the box labeled "memory", the remaining parts of the system, in the upper box,

form a purely combinational circuit that has as inputs the present inputs of the system and the present internal state and as outputs the present output of the system and the next internal state.

**Synchronous and asynchronous systems.** Before proceeding with our discussion of analysis and synthesis of sequential circuits, we shall find it helpful to distinguish between two classes of circuitry: synchronous circuits and asynchronous circuits. The distinguishing characteristic of the synchronous circuit is that its input combination, output combination, and internal state are sensed only at certain definite moments of time. Commonly, such circuits are active (or voltage-producing) circuits rather than passive (or path-closing) circuits, and the sensing is done by applying an enabling signal to each logic element of the system at each of these definite moments. Although these sensing times may be unequally spaced, they usually are not. Many synchronous systems employ a 1-megacycle oscillator to apply "clock" pulses of a few tenths of a microsecond duration and 1 $\mu$sec spacing to all parts of the system. (Higher and lower frequencies have also been used.) When the sensing times are equispaced, the memory elements are often merely delay devices, which produce an output replica of their input after a delay of $\delta$ clock periods, $\delta$ being a positive integer. Of course, the logic elements also have time delays, since no physical device can produce its output simultaneously with its input. Although these delays must ultimately be taken into account when the electrical devices are assembled in the circuit, theoretical treatment is facilitated by assuming that all delays are incorporated in the memory elements and that logic elements act without delay.

The symbols commonly used for synchronous circuits are those used with dynamic electronic circuitry (see Chap. 9) and are the same as those of Fig. 8.15. When these symbols are used to represent synchronous circuits, it is assumed that clock signals are applied to each gate element, so that outputs from each gate are obtained only at clock times.

The simplest distinction that can be drawn between synchronous circuits and asynchronous circuits is that the operations of synchronous circuitry proceed in synchronism with externally furnished clock pulses whereas those of asynchronous circuitry proceed at their own speed without external clock signals. Asynchronous circuits sometimes generate timing signals to mark the end of certain operations, but sometimes no timing signal is used and subsequent operations are always delayed long enough to ensure completion of current operations. In most large digital systems certain portions operate synchronously, i.e., with externally furnished timing signals, and other parts operate asynchronously. However, in common usage, "synchronous circuitry" is taken to mean circuitry in which each elementary group of three or four AND gates and a single OR gate has its output timed by a clock pulse, and "asynchronous

circuitry" means circuitry in which no timing signals are applied at the elementary logic level. Asynchronous circuitry using active (voltage-producing) elements often uses d-c levels of voltage instead of pulse-or-no-pulse signals. Such circuitry is called d-c circuitry or static circuitry. The symbols of Fig. 8.52 are often used with asynchronous active circuitry. All signals are binary, having at any time one or the other of two voltage levels, one corresponding to 1 and the other to 0. The device

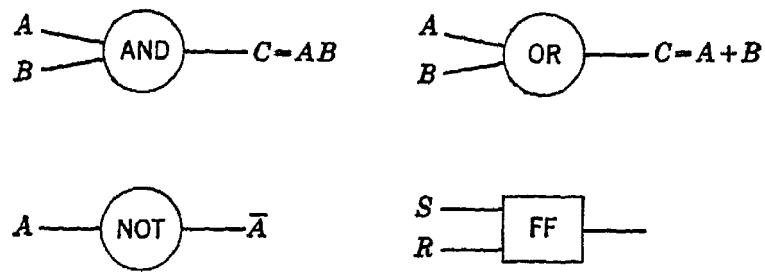



Fig. 8.52 A set of symbols often used for asynchronous circuits

marked FF is a flip-flop memory element. A signal on its $S$ input sets its output to the 1 level, and a signal on its $R$ input resets its output to the 0 level. Only one input can be applied at a time, and the device remains in the state to which it was last set until an appropriate input changes it.

Relays constitute the other principal type of asynchronous circuitry. With relays in particular, and with asynchronous circuitry in general, it is necessary to distinguish between the state of excitation of an element and its output state. A short time may elapse between application of an exciting signal (or removal of such a signal) and the resulting change in the element's output. This time differs not only from one element to another but also for an individual element as it is affected by changes of temperature, humidity, voltage, etc. Consequently, if the states of excitation of $n$ elements are changed simultaneously, it is quite unlikely that their outputs will also change simultaneously and quite likely that $n - 1$ output combinations will occur before the desired final output. Some of these undesired chance outputs may put the circuit into other undesired conditions. The circuit designer tries to avoid such operating hazards, i.e., conditions in which undesired random transient outputs can occur. A similar hazard, called a "race condition", occurs when two or more memory elements change excitation simultaneously. The undesired internal states through which the system passes as the output states of the elements change may cause the system to end in different ultimate internal states depending on the outcome of the "race" between the memory elements. Methods of dealing with these hazards have been investigated by Huffman.[34]

**Methods of analysis and synthesis.**[35,36] Since both the present output combination and the next internal state of a sequential circuit are

determined by the present input combination and the present internal state, we may tabulate the behavior of a given circuit by listing all combinations of inputs and internal states and, corresponding to each of these, the output combination and the next internal state. If there are $n$ input variables and $s$ internal-state variables, such a tabulation will have $2^{n+s}$ rows. If there are also $m$ output variables, then the table will have $m + s$ columns. Such a table, when completely filled out, constitutes a complete description of the circuit and is a formal solution of the analysis

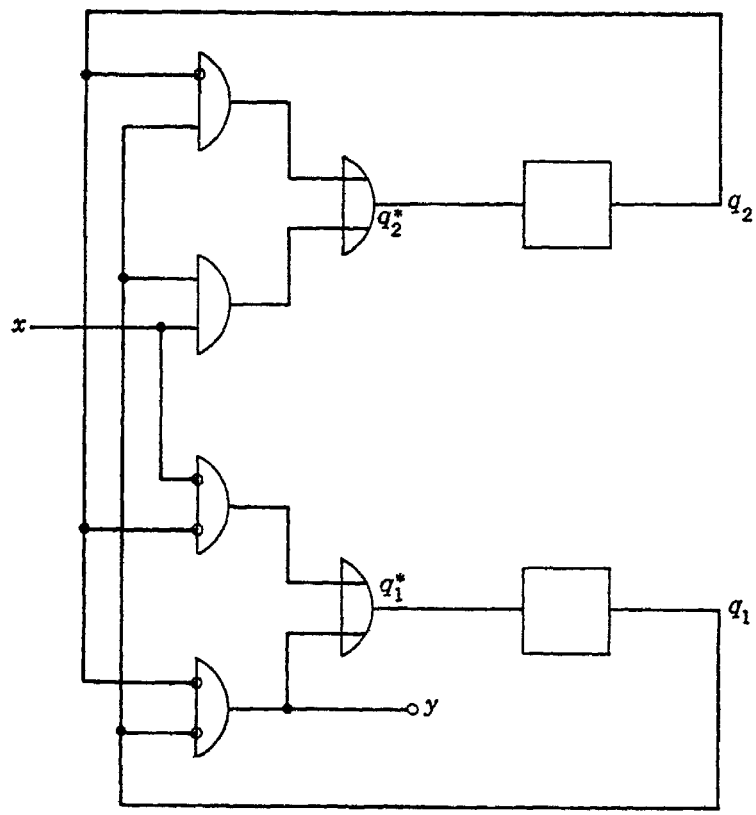



Fig. 8.53 A sequential circuit

problem. Such a complete tabulation can always be made if the internal structure of the system is fully known, but, if only the terminals of the system are available for study, it is not always possible to ascertain the details of the system's internal behavior. However, a complete description of the terminal behavior for all possible inputs and input sequences may also be regarded as a solution of the analysis problem. The terminal behavior may be conveniently described by what are called "state diagrams".

To demonstrate sequential-network analysis, we borrow an example (Fig. 8.53) from Mealy.[35] This is an active (voltage-producing) synchronous system. The rectangular boxes are 1-unit delay elements, and it is assumed that the two gates at the lower left, which are shown as having only inhibitory inputs, will deliver an output pulse whenever no

inputs are present. $q_1$ and $q_2$ are the present internal-state variables, and $q_1^*$ and $q_2^*$ are the next internal-state variables. It can readily be seen that

$$\begin{aligned} q_1^* &= \bar{q}_1\bar{q}_2 + \bar{x}\bar{q}_2 \\ q_2^* &= q_1\bar{q}_2 + xq_1 \\ y &= \bar{q}_1\bar{q}_2 \end{aligned} \tag{8.80}$$

For this circuit, $n = 1$, $s = 2$, and $m = 1$. A tabular description of the circuit will, therefore, have eight rows and three columns. Such a table is shown as Table 8.11.

**Table 8.11**

| $q_1$ | $q_2$ | $x$ | $q_1^*$ | $q_2^*$ | $y$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 |

A state diagram corresponding to this circuit is shown in Fig. 8.54. Each circle carries a binary designator $q_1q_2$. The circles are connected by arrows indicating possible changes of state. Each arrow carries a number pair $a/b$, where $a$ is the new input and $b$ is the new output. Each arrow corresponds to one or more rows of the table. Thus, if $q_1q_2 = 00$, then $y = 1$, and the next state is 10 whether $x = 0$ or 1. This information is given either by the first two rows of the table or by the arrow from 00 to 10 on the diagram.

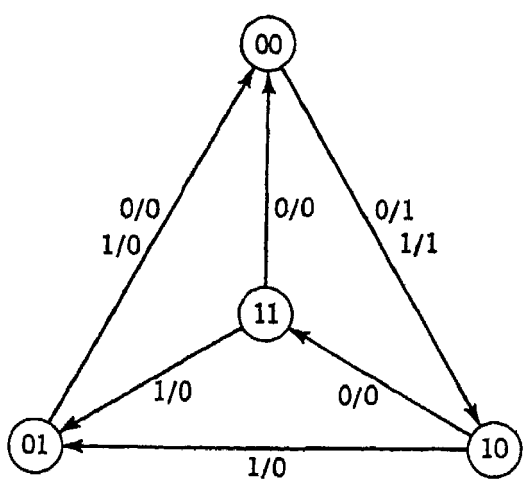


Fig. 8.54 State diagram for the circuit of Fig. 8.53

Not every combination of internal-state variables need give an internal state that is admissible for a given system. It may happen, for instance, that a table of the form of Table 8.11 contains descriptions of internal states that cannot be reached by any sequence of inputs. It may also happen that two systems with different numbers of internal states give identical output sequences whenever they receive identical input sequences. This is roughly what we mean when we say that two cir-

cuits are equivalent. The concept of equivalence is used in simplifying a state diagram by eliminating states from a circuit while maintaining its equivalence to a desired circuit.

The synthesis problem for sequential circuits consists in finding an optimum state diagram or table. Once we know all possible combinations of input variables and present-state variables and the corresponding combinations of output variables and next-state variables, the problem reduces to design of a combinational circuit. The difficult step is the formation of a state diagram or table that satisfies the requirements imposed. At present there is no suitable formalism for accomplishing this step, and the designer is on his own. However, it is not necessary that this step be done efficiently, since there exist well-formulated methods for reducing a state diagram to an equivalent that is simpler in the sense of having fewer internal states. (Of course, it is by no means clear that the circuit having the fewest internal states will use the least equipment in its physical realization. However, for $s$ states we need at least $p$ memory elements, where $p$ is the least integer not smaller than $\log_2 s$. Thus the memory elements will be used more effectively if $s$ is slightly less than a power of 2 than if it is slightly greater than a power of 2.)

Formal methods for reducing state diagrams have been described in detail by Huffman[36] and by Mealy,[35] and the reader is urged to refer to their very lucid papers for detailed descriptions of these methods. Such discussion is beyond the scope of this work, especially since the original papers are so well presented. Huffman's method, it may be noted, is directed specifically toward asynchronous relay circuits, with brief comments upon asynchronous active circuits. Mealy's work, on the other hand, deals with synchronous circuits but is broad enough to be applicable also to asynchronous circuits. (A less formalized and more intuitive method for designing asynchronous circuits is the sequence diagram, which has been fully described by Keister, Ritchie, and Washburn.[7–10])

Other significant work in the field of sequential circuits has been done by Moore,[37] who developed formal concepts of equivalence independently of Huffman, and by Burks and Wang,[33] who have formalized the concept of admissibility and have given useful matrix procedures for sequential systems.

## REFERENCES

*Items Referred to by Number:*

1. Hilbert, D., and W. Ackerman, "Principles of Mathematical Logic" (1928), reprinted in English translation by Chelsea Publishing Company, New York, 1950.
2. Langer, Susanne K., "An Introduction to Symbolic Logic," Dover Publications, New York, 2d ed., 1953.

3. Stabler, E. R., "An Introduction to Mathematical Thought," Addison-Wesley Publishing Company, Cambridge, Mass., 1953.
4. Rosenbloom, Paul C., "The Elements of Mathematical Logic," Dover Publications, New York, 1950.
5. Birkhoff, G., and S. MacLane, "A Survey of Modern Algebra," The Macmillan Company, New York, 1941.
6. Copi, I. M., "Symbolic Logic," The Macmillan Company, New York, 1954.
7. Keister, W., A. E. Ritchie, and S. H. Washburn, "The Design of Switching Circuits," D. Van Nostrand Company, Inc., Princeton, N.J., 1951.
8. Phister, M., Jr., "Logical Design of Digital Computers," John Wiley & Sons, Inc., New York, 1958.
9. Caldwell, S. H., "Switching Circuits and Logical Design," John Wiley & Sons, Inc., New York, 1958.
10. Culbertson, J. T., "Mathematics and Logic for Digital Devices," D. Van Nostrand Company, Inc., Princeton, N.J., 1958.
11. Shannon, C. E., "The synthesis of two-terminal switching circuits," *Bell System Tech. J.*, vol. 28, January, 1949, pp. 59–98.
12. Burks, A. W., R. McNaughton, C. H. Pollmar, D. W. Warren, and J. B. Wright, The folded tree, *J. Franklin Inst.*, vol. 260, no. 1, July, 1955, pp. 9–24; no. 2, August, 1955, pp. 115–126.
13. Burks, A. W., R. McNaughton, C. H. Pollmar, D. W. Warren, and J. B. Wright, Complete decoding nets: General theory and minimality, *J. Soc. Ind. Appl. Math.*, vol. 2, no. 4, December, 1954, pp. 201–243.
14. Eddington, Sir Arthur S., "Fundamental Theory," Cambridge University Press, New York, 1948, pp. 265–283.
15. Slepian, D., On the number of symmetry types of Boolean functions of $n$ variables, *Can. J. Math.*, vol. 5, no. 2, 1953, pp. 185–193.
16. Miller, R. E., Formal analysis and synthesis of bilateral switching networks, *IRE Trans. on Electronic Computers*, vol. EC-7, no. 3, September, 1958, pp. 231–244.
17. Harvard Computation Laboratory, Synthesis of electronic computing and control circuits, *Ann. Harvard Univ. Computation Lab.*, vol. 27, 1951, pp. 50–67.
18. Veitch, E. W., A chart method for simplifying truth functions, *Proc. Meeting ACM, May 2–3, 1952*, pp. 127–133.
19. Karnaugh, M., The map method for synthesis of combinational logic circuits, *Trans. AIEE*, vol. 72, pt. I, 1953, pp. 593–599.
20. Quine, W. V., The problem of simplifying truth functions, *Am. Math. Monthly*, vol. 59, no. 8, October, 1952, pp. 521–531.
21. Burkhart, W. H., Theorem minimization, *Proc. Meeting ACM, May 2–3, 1952*, pp. 259–263.
22. Quine, W. V., A way to simplify truth functions, *Am. Math. Monthly*, vol. 62, no. 10, November, 1955, pp. 627–631.
23. Nelson, R. J., Simplest normal truth functions, *J. Symbolic Logic*, vol. 20, no. 2, June, 1955, pp. 105–108.
24. Lee, C. Y., Switching functions on an $n$-dimensional cube, *Trans. AIEE*, vol. 73, pt. I, 1954, pp. 289–291.
25. Roth, J. P., Algebraic topological methods for the synthesis of switching circuits in $n$ variables, *IAS Electronic Computer Project, tech. rept.* 56-02, April, 1956, Princeton, N.J.
26. Urbano, R. H., and R. K. Mueller, A topological method for determination of the minimal forms of a Boolean function, *IRE Trans. on Electronic Computers*, vol. EC-5, no. 3, September, 1956, pp. 126–132.
27. Harris, B., An algorithm for determining minimal representations of a logic func-

tion, *IRE Trans. on Electronic Computers*, vol. EC-6, no. 2, June, 1957, pp. 103–108.

28. Muller, D. E., Application of Boolean algebra to switching circuit design and to error detection, *IRE Trans. on Electronic Computers*, vol. EC-3, no. 3, September, 1954, pp. 6–12.
29. McCluskey, E. J., Jr., Minimization of Boolean functions, *Bell System Tech. J.*, vol. 35, no. 6, November, 1956, pp. 1417–1444.
30. Hohn, F. E., and L. R. Schissler, Boolean matrices and the design of combinational relay switching circuits, *Bell System Tech. J.*, vol. 34, no. 1, January, 1955, pp. 177–202.
31. Hohn, F. E., A matrix method for the design of relay circuits, *IRE Trans. on Circuit Theory*, vol. CT-2, no. 2, June, 1955, pp. 154–161.
32. Shannon, C. E., and J. McCarthy (eds.), "Automata Studies," Princeton University Press, Princeton, N.J., 1956.
33. Burks, A. W., and H. Wang, The logic of automata, *J. ACM*, vol. 4, no. 2, April, 1957, pp. 193–218; vol. 4, no. 3, July, 1957, pp. 279–297.
34. Huffman, D. A., The design and use of hazard-free switching networks, *J. ACM*, vol. 4, no. 1, January, 1957, pp. 47–62.
35. Mealy, G. H., A method for synthesizing sequential circuits, *Bell System Tech. J.*, vol. 34, no. 5, September, 1955, pp. 1045–1079.
36. Huffman, D. A., The synthesis of sequential switching circuits, *J. Franklin Inst.*, vol. 257, no. 3, March, 1954, pp. 161–190; vol. 257, no. 4, April, 1954, pp. 275–303.
37. Moore, E. F., Gedanken-experiments on sequential machines, in Shannon, C. E., and J. McCarthy (eds.), "Automata Studies," Princeton University Press, Princeton, N.J., 1956, pp. 129–153.

*References for Supplementary Study:*

Shannon, C. E., A symbolic analysis of relay and switching circuits, *Trans. AIEE*, vol. 57, 1938, pp. 713–723.

Riordan, J., and C. E. Shannon, The number of 2-terminal series and parallel networks, *J. Math. and Phys.*, vol. 21, no. 2, 1942, pp. 83–93.

Burks, A. W., and J. B. Wright, Theory of logical nets, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1357–1365.

Lee, C. Y., and W. H. Chen, Several-valued combinational switching circuits, *Trans. AIEE*, vol. 75, pt. I, 1956, pp. 278–283.

Seshu, S., On electrical circuits and switching circuits, *IRE Trans. on Circuit Theory*, vol. CT-3, no. 3, September, 1956, pp. 172–178.

McCluskey, E. J., Jr., Detection of group invariance or total symmetry of a Boolean function, *Bell System Tech. J.*, vol. 35, no. 6, November, 1956, pp. 1445–1453.

Gilbert, E. N., Lattice theoretic properties of frontal switching functions, *J. Math. and Phys.*, vol. 33, no. 1, April, 1954, pp. 57–67.

Gardner, Martin, "Logic Machines and Diagrams," McGraw-Hill Book Company, Inc., New York, 1958.

Hohn, F. E., S. Seshu, and D. D. Aufenkamp, The theory of nets, *IRE Trans. on Electronic Computers*, vol. EC-6, September, 1957, pp. 154–161.

Aufenkamp, D. D., and F. E. Hohn, Analysis of sequential machines, *IRE Trans. on Electronic Computers*, vol. EC-6, December, 1957, pp. 276–285.

Berlin, R. D., Synthesis of N-valued switching circuits, *IRE Trans. on Electronic Computers*, vol. EC-7, no. 1, March, 1958, pp. 52–56.

Epstein, G., Synthesis of electronic circuits for symmetric functions, *IRE Trans. on Electronic Computers*, vol. EC-7, no. 1, March, 1958, pp. 57–60.

Ginsburg, S., On the reduction of superfluous states in a sequential machine, *J. ACM*, vol. 6, no. 2, April, 1959, pp. 259–282.

Abhyankar, S., Minimal "sum of products of sums" expressions of Boolean functions, *IRE Trans. on Electronic Computers*, vol. EC-7, no. 4, December, 1958, pp. 268–276.

Gazalé, M. J., Irredundant disjunctive and conjunctive forms of a Boolean function, *IBM J. Research Develop.*, vol. 1, no. 2, April, 1957, pp. 171–176.

## PROBLEMS

**8.1** Express each of the eight areas in the diagram in terms of $\cap$, $\cup$, and $\sim$ operations among $A$, $B$, and $C$.

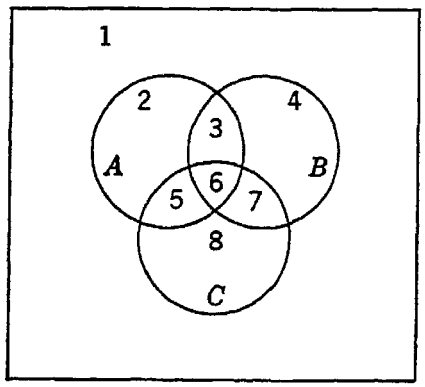



Fig. P8.1

**8.2** Demonstrate by means of the Venn diagram that

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

and

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

**8.3** Express each of the functions of two variables by means of (*a*) the stroke function, and (*b*) the dual stroke function.

**8.4** Simplify each of the following networks by (*a*) rearranging the network, and

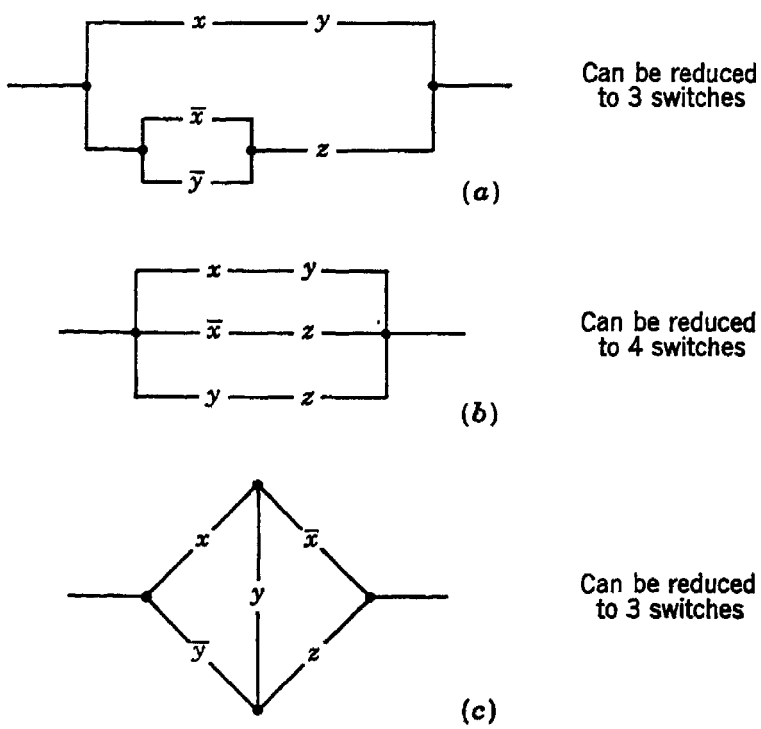



Fig. P8.4

(*b*) writing the transmission or hindrance function for each and simplifying by means of the logical identities in Eqs. (8.14) to (8.23).

**8.5** Consider 2 two-terminal networks, one having transmission $T = T_1$, and the other having hindrance $H = \bar{T}_1$. What is the transmission function of the combined network when these two networks are connected (*a*) in parallel, and (*b*) in series?

**8.6** Express each of the following functions in the fully expanded disjunctive normal form (sum of products) and in the fully expanded conjunctive normal form:

(*a*) $f = AC + AD + AB + B\bar{D}$

(*b*) $f = \bar{B}\bar{D} + \bar{A}C\bar{D} + \bar{A}BC + A\bar{B}C$

(*c*) $f = (A + \bar{D})(B + \bar{C})(\bar{A} + BC)$

(*d*) $f = A\bar{B} + B\bar{C}(A + \bar{D})(\bar{A} + \bar{C})$

**8.7** The reflected code was described in Sec. 7.5. This code is only one of many 4-bit codes having the property that a change of 1 in the decimal number causes only one digit of the binary-code group to change. Successive changes in the decimal number can be represented by a path on the Karnaugh map. Thus the code of Table 7.12 maps as follows:

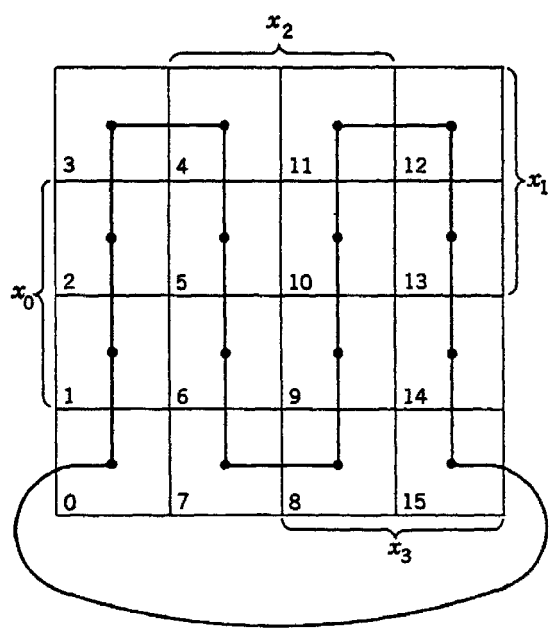



Fig. P8.7

Codes in which only one binary digit changes at a time are called "unit distance" codes, and their maps consist of successions of horizontal and vertical segments, each 1 unit long.

Invent some unit distance codes with the aid of the map.

How many 4-bit unit distance codes are there?

**8.8** Each of the following functions can be expressed in two different ways as a sum of products with eight literals appearing. Find both forms for each function.

(*a*) $f(A,B,C,D) = ABC + \bar{A}BC\bar{D} + \bar{A}\bar{C}\bar{D} + A\bar{B}CD + \bar{B}\bar{C}D$

(*b*) $f(A,B,C,D) = A\bar{C}\bar{D} + ABC\bar{D} + BCD + \bar{A}\bar{B}\bar{D} + \bar{A}\bar{B}CD$

**8.9** Use the Karnaugh map to find a minimum sum-of-products form for each of the following functions. [The numerical designators indicate those rows in the table of combinations of *ABCD* for which the function is 1, as in Eq. (8.64).]

(a) $f(A,B,C,D) = \Sigma 2,3,4,6,7,8,12$
(b) $f(A,B,C,D) = \Sigma 4,6,7,9,10,11,12,14,15$
(c) $f(A,B,C,D) = \Sigma 0,3,4,5,6,7,8,13,14$
(d) $f(A,B,C,D) = \Sigma 0,2,4,6,8,9,10,13,14,15$

**8.10** Using the Karnaugh map, diagram a function of four variables that has more prime implicants than the number of terms in its fully expanded sum-of-products form.

**8.11** Simplify each of the following functions, making optimum use of the don't care conditions:

(a) $f = ABC + BC\bar{D} + \bar{A}B\bar{C}\bar{D} + AB\bar{C}D$ Don't care $= \bar{A}BCD$
(b) $f = \bar{A}C\bar{D} + B\bar{C}D$ Don't cares $= \bar{A}BCD + ABC\bar{D}$
(c) $f = A\bar{B}\bar{C} + ABC + \bar{A}B\bar{C}$ Don't cares $= AB\bar{C} + \bar{A}\bar{B}C$

**8.12** We have seen that the product-of-sums form can be obtained from the fully expanded sum-of-products form by negating the sum of the missing terms. Show that the Karnaugh map can be used to get results in product-of-sums form by selecting subcubes of zeros and labeling them with the complements of the labels used for the sum-of-products map. Each subcube, of course, must be written as a sum instead of as a product.

Using this method, find the product-of-sums form for each of the following functions:

(a) $f(A,B,C) = \Sigma 1,2,3,4,6$
(b) $f(A,B,C,D) = \Sigma 4,6,7,9,10,11,12,14,15$
(c) $f(A,B,C,D) = \Sigma 3,4,6,7,10,12,14,15$

**8.13** Use McCluskey's method to find a minimum sum-of-products form for each of the following functions:

(a) $f = \Sigma 2,6,7,10,20,21,22,23$
(b) $f = \Sigma 2,6,7,10,11,17,19,20,21,22,23,26,27,31$
(c) $f = \Sigma 1,3,6,7,8,12,14,16,18,22,23,24,28,30,31$

**8.14** For the binary tree of Fig. 8.19, show that the total number of contacts is $2^n - 1$, if the tree has $n$ relays.

**8.15** Use the matrix method to find networks of switches to realize each of the following functions:

(a) $f = w\bar{x}yz + \bar{w}xyz$
(b) $f = xy\bar{z} + x\bar{y}z + \bar{x}yz$
(c) $f = xyz + x\bar{y}\bar{z} + \bar{x}\bar{y}z$

**8.16** Expand the function of Eq. (8.69) about the variables $x$ and $z$ and carry out the Shannon network-synthesis procedure. Repeat, expanding about the variables $y$ and $z$.

CHAPTER 9

# Logic Circuits for Arithmetic and Control

## 9.1 Introduction

Let us turn now from our consideration of the more abstract phases of logic and switching theory to the discussion of basic logic circuits used to implement the computer. We shall find it helpful to consider these circuits as being

1. Either arithmetic circuits or control circuits
2. Either serial circuits or parallel circuits
3. Either synchronous circuits or asynchronous circuits

In practice these alternatives are not always mutually exclusive, and it is sometimes difficult to classify a particular circuit or group of circuits. Nevertheless, there are enough cases in which the distinction is clear-cut to make this classification useful for an initial approach to the subject. The distinctions as well as the border-line areas will be clarified in the following sections by consideration of some typical circuits.

The arithmetic circuits to be considered in this chapter will be restricted to purely binary operations. An understanding of circuits for binary arithmetic will make it a straightforward matter for the reader to handle circuits for doing arithmetic in other radices and other codes. The differences between binary arithmetic and other kinds of arithmetic are differences not so much of principle as of degree. The reader who wishes to delve more deeply will find decimal arithmetic well described in the literature, and he may also find it interesting to investigate the devising of circuits that can perform arithmetic in some of the other number systems discussed in Chap. 7.

## 9.2 Circuits for adding

One might reasonably argue that the most basic and important circuit in a computer is the circuit that performs addition and that everything else is peripheral equipment existing merely to present the proper num-

bers to the adder and remove the results. Of course, there are computing systems for which this argument is not valid, since their basic operation is logical rather than arithmetical. In such computers the fundamental process is manipulation rather than calculation, and operations of sorting, merging, classifying, comparing, and tallying occur more often than addition, subtraction, multiplication, or division. Nevertheless, the adder is a sufficiently important circuit to warrant our considering it first, and it is also a convenient reference for discussion of the serial-parallel and synchronous-asynchronous classifications.

In performing the pencil-and-paper addition of two numbers $x$ and $y$, we first add $x_0$, the least-significant digit of $x$, to $y_0$, the least-significant digit of $y$, to obtain $S_0$, the least-significant digit of the sum, and $c_1$, the carry digit into position 1. In succeeding steps we add $x_j$, $y_j$, and $c_j$ to obtain $S_j$ and $c_{j+1}$. Direct mechanization of this process requires an adder that has three input variables and two outputs. The table of combinations is shown in Table 9.1. This table is equivalent to the Boolean

**Table 9.1**
**Sum and carry digits**

| $x_j$ | $y_j$ | $c_j$ | $S_j$ | $c_{j+1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

expressions

$$\begin{aligned} S_j &= \bar{x}_j\bar{y}_jc_j + \bar{x}_jy_j\bar{c}_j + x_j\bar{y}_j\bar{c}_j + x_jy_jc_j \\ c_{j+1} &= y_jc_j + x_jc_j + x_jy_j \end{aligned} \tag{9.1}$$

Several choices now exist for ways in which to implement this circuit. The most obvious way (and, therefore, probably not the best) is to assemble AND and OR elements in exactly the configuration called for by Eq. (9.1) in its present form. Although this procedure gives a circuit that meets the requirements, it is not an efficient circuit, since (1) both equations can be factored, and (2) some logic elements can be shared between the circuits for $S_j$ and for $c_{j+1}$. Possible arrangements for factoring and for element sharing can be found either by direct algebraic manipulation of the expressions for $S_j$ and $c_{j+1}$ or by other methods such as the use of a Karnaugh map. One possible factorization appears in

Eq. (9.2):

$$\begin{aligned} S_j &= c_j(x_j y_j + \bar{x}_j \bar{y}_j) + \bar{c}_j(x_j \bar{y}_j + \bar{x}_j y_j) \\ c_{j+1} &= c_j(x_j + y_j) + x_j y_j \end{aligned} \tag{9.2}$$

A configuration for implementing these equations with path-closing elements (relays) is shown in Fig. 9.1. This circuit uses 16 switches per stage, which can be grouped as eight transfer contacts. It also requires two relays per stage, one for $x_j$ and one for $y_j$.

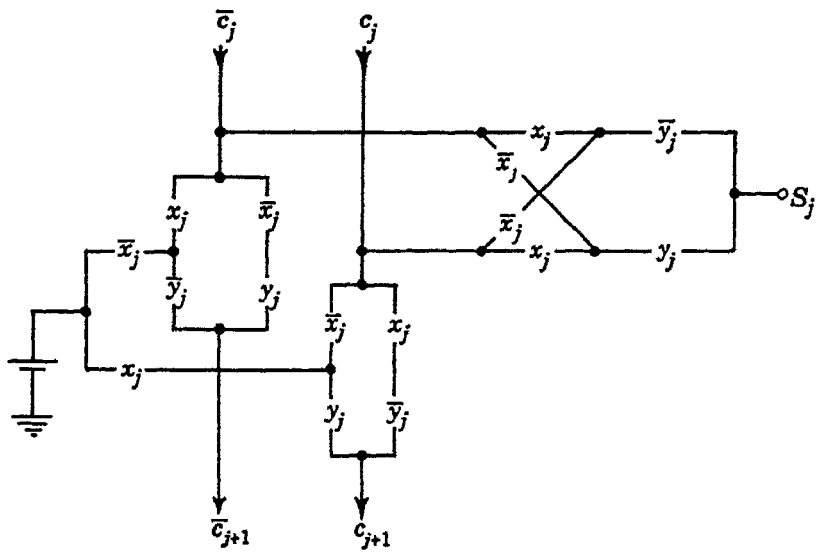



Fig. 9.1 A relay adder stage

There are two ways in which this adder stage can be used. (1) In the serial mode, the digits of $x$ and $y$ are presented sequentially, $x_0$ and $y_0$ being applied together, then $x_1$ and $y_1$, etc. In this mode the carry digit that is formed when $x_j$ and $y_j$ are applied must be stored and applied to the $c_j$ and $\bar{c}_j$ inputs during the time when $x_{j+1}$ and $y_{j+1}$ are applied. (2) In the parallel mode, the circuit uses as many adder stages as there are digit positions in each number. All the $n$ digits of $x$ and the $n$ digits of $y$ are applied simultaneously, and the sum appears in parallel on the $n + 1$ output terminals.

Several points should be noted about the operation of the circuit in these two modes. First, the parallel mode is faster, because all digits are applied at the same time, and, if the operating times for all relays were the same, the outputs would also occur simultaneously after a delay equal to this operating time. As a practical matter the relays will operate with slightly different speeds, and the states of the output leads will not represent the true sum until all energized relays have operated. However, the time required for even the slowest relay to operate should be much less than the time required for the $n$ sequential relay operations needed in the serial mode. If all relays were identical, the parallel mode would be $n$ times as fast. However, it would require almost $n$ times as

much equipment as the serial mode (the first stage is simplified by absence of carry inputs).

In either the parallel or the serial mode the circuit can operate either synchronously or asynchronously. The basic distinction here is that the synchronous circuit is externally timed and the asynchronous circuit is self-timed. In any case, timing signals are necessary to distinguish one time interval from the next. Thus, if $x_0$ and $y_0$ are applied to the serial

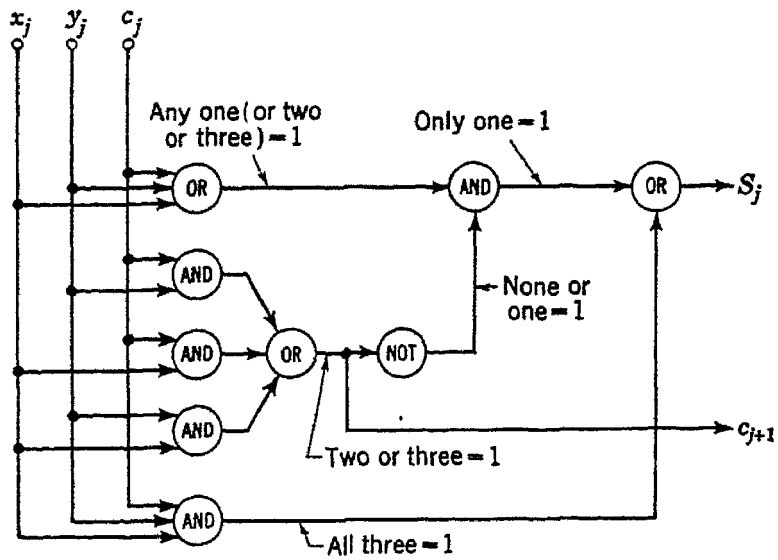



Fig. 9.2 A three-input adder

adder, $S_0$ is formed when the relays have operated. $x_0$ and $y_0$ are then removed, the relays release, and $x_1$ and $y_1$ are applied. However, if $x_0 = x_1$ and $y_0 = y_1$, there is no way to tell from mere observation of the output whether any or many input digits have been applied. In the synchronous mode the input signals are applied at regularly spaced instants of time, and the circuit outputs are sensed at these same instants (or slightly later, to allow for the relay operation time). Thus a succession of addend and augend zeros would be detected as a succession of sum zeros instead of as only one continuous zero. In the asynchronous mode supplementary circuits detect when the relay action has been completed, then connect the resulting output, and finally connect the next inputs.

The relay circuit described in the preceding pages is only one of a great many different ways of organizing the structure of an adder. Let us now consider some of the other ways of constructing adders. In the following discussion we shall assume the use of active or voltage-producing elements rather than passive or path-closing elements, since such devices are of more interest to the computer engineer.

Binary adders may be classified as three-input adders or as two-input adders. The three-input adder accepts as inputs the addend digit $x_j$,

the augend digit $y_j$, and the carry digit $c_j$, and it produces as outputs a sum digit $S_j$ and a carry digit $c_{j+1}$ to be added into the next position. The two-input adder can accept only two of the three digits $x_j$, $y_j$, and $c_j$; therefore, it must add two of them to obtain a partial sum and then add this to the third to give the final sum $S_j$ and the carry $c_{j+1}$. This second addition is usually carried out in a second two-input adder, and, since two of these devices thus combine to make a whole adder, each of them is commonly called a "half-adder".

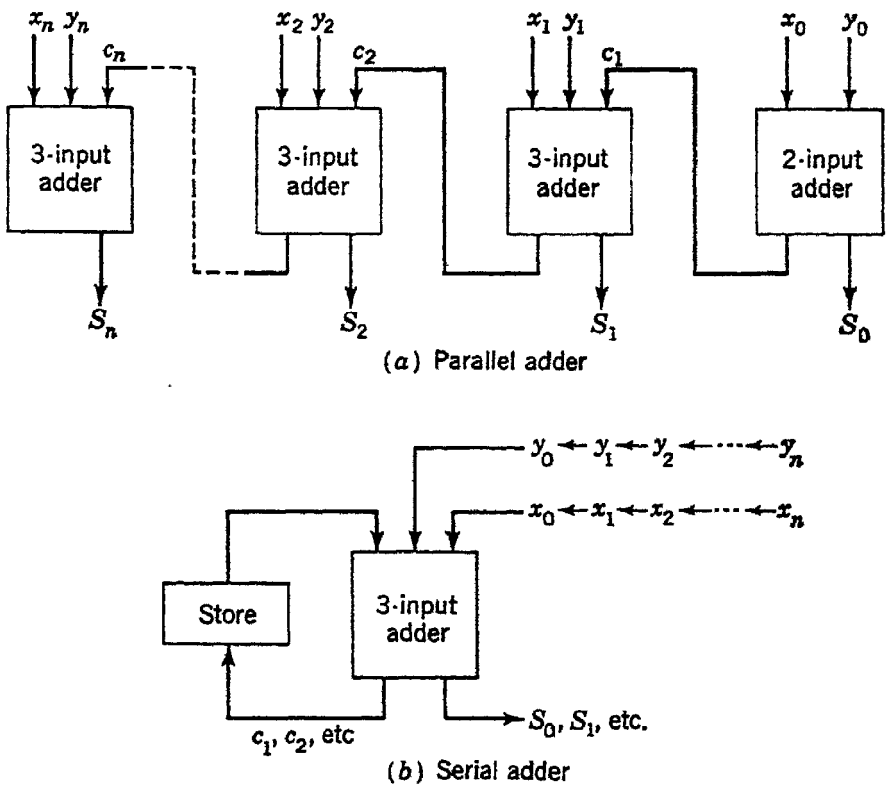



Fig. 9.3 Parallel and serial adders

A three-input adder in near-minimal form can be devised by first noting from Table 9.1 that $S_j = 1$ if either exactly one or all three of the inputs equal 1, and that $c_{j+1} = 1$ if either two or three of the inputs equal 1. On the basis of this observation the circuit of Fig. 9.2 can readily be formed. The verbal statements written in the figure state the input conditions under which a signal representing the digit 1 is present at each of the indicated points.

If this circuit is used for serial addition, then $c_{j+1}$ must be stored in some manner and applied to the input marked $c_j$ at the time the next pair of input digits $x_j$ and $y_j$ are applied. If it is used as one stage of a parallel adder, then $c_{j+1}$ is applied to the $c_j$ terminal of the adder in the next-more-significant-digit position. These two configurations are shown in Fig. 9.3.

Although the three-input adder of Fig. 9.2 can be implemented with either synchronous or asynchronous circuitry, the symbols of Fig. 9.2 are more commonly used when the elements are asynchronous and the alternative symbolism in Fig. 9.4 is usually used with synchronous circuitry.

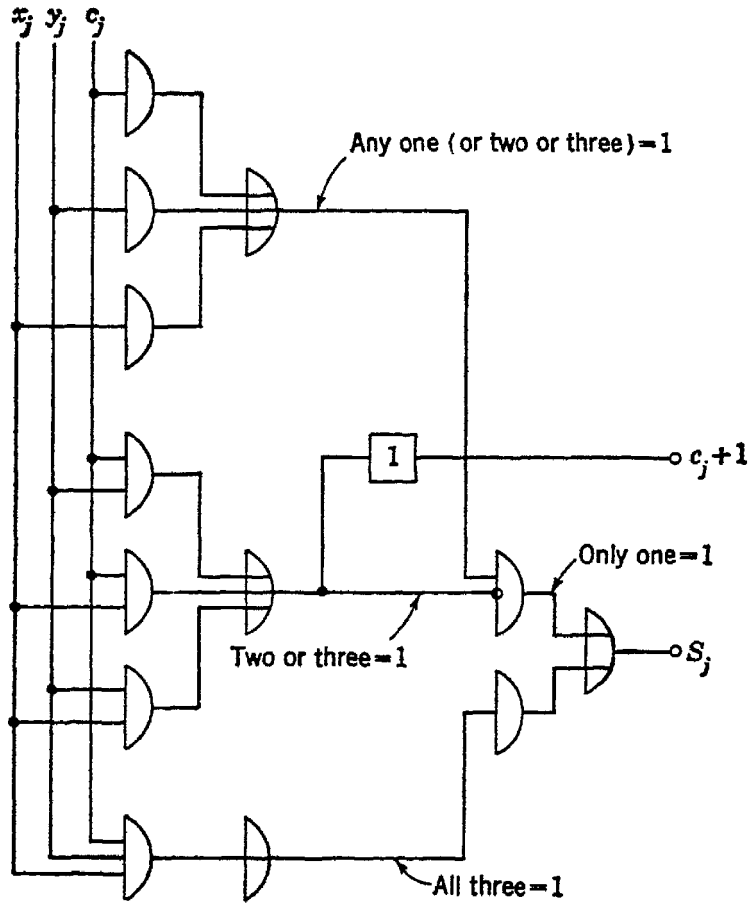



Fig. 9.4 Three-input serial adder using synchronous circuitry

If the timing signals are periodic, then the storage elements can be merely pieces of electrical delay line which produce an output replica of their input after an integral number of clock pulse periods. The symbol used for these delays is a rectangular box with the delay interval (in pulse periods) written in it. (Sometimes, if the delay is exactly one pulse period, the numeral 1 is omitted.) The symbols used in Fig. 9.4 are those introduced in Chap. 8 in Fig. 8.15. For the purposes of the logic circuit it is assumed that all elements except the delay element act instantaneously. As a practical matter, of course, they do not act instantaneously. However, compensation for the operating delay can be accomplished by adjustment of the electrical circuit without affecting the logic circuit. We shall treat this electrical adjustment in the next chapter and in the meantime assume that the logic elements, in the synchronous case at least, are instantaneous devices.

**Two-input adders (half-adders).** Let us now examine the use and design of a simpler adder having only two inputs. We shall start again from the basic equations for the sum and carry digits:

$$S_j = x_j\bar{y}_j\bar{c}_j + \bar{x}_jy_j\bar{c}_j + \bar{x}_j\bar{y}_jc_j + x_jy_jc_j \tag{9.3}$$
$$c_{j+1} = x_jy_j + x_jc_j + y_jc_j \tag{9.4}$$

The expression for $S_j$ may be factored as follows:

$$S_j = (x_j\bar{y}_j + \bar{x}_jy_j)\bar{c}_j + (x_jy_j + \bar{x}_j\bar{y}_j)c_j \tag{9.5}$$

Now we define

$$s_j = x_j\bar{y}_j + \bar{x}_jy_j = x_j \oplus y_j \tag{9.6}$$

Thus,

$$S_j = s_j\bar{c}_j + \bar{s}_jc_j \tag{9.7}$$
$$= s_j \oplus c_j \tag{9.8}$$

The expression for $c_{j+1}$ we manipulate as follows:

$$c_{j+1} = x_jy_j + x_j(y_j + \bar{y}_j)c_j + (x_j + \bar{x}_j)y_jc_j \tag{9.9}$$
$$= x_jy_j + (x_j\bar{y}_j + \bar{x}_jy_j)c_j \tag{9.10}$$
$$= x_jy_j + (x_j \oplus y_j)c_j \tag{9.11}$$
$$= x_jy_j + s_jc_j \tag{9.12}$$

These equations are implemented in the logic circuit of Fig. 9.5. An element to perform the ring-sum operation $\oplus$ (exclusive OR, or sum

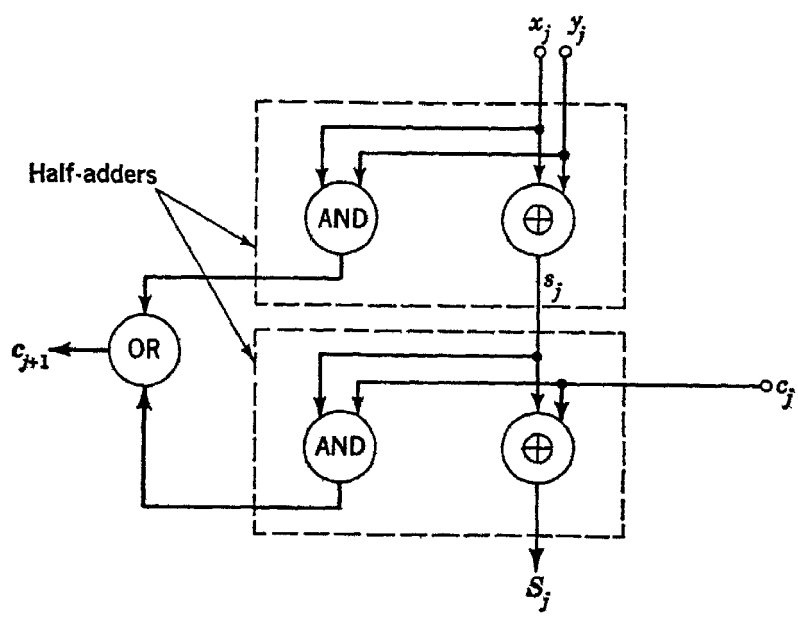



Fig. 9.5 An adder stage

modulo 2) is used in this diagram as a convenient way of showing the circuit action. The circuit contains two identical subcircuits and an additional OR element. Each of these subcircuits is, of course, a half-adder or two-input adder. (In using the word "adder", the reader is cautioned to distinguish carefully among the ring-sum element or modulo

2 adder, the half-adder, the complete adder stage or three-input adder, and the entire adder circuitry of a computer, which is sometimes taken to mean not only the $n$ complete stages but also the addend, augend, and sum registers.)

An alternative manipulation of the equations for sum and carry digits is as follows:

$$S_j = (x_j\bar{c}_j + \bar{x}_jc_j)\bar{y}_j + (x_jc_j + \bar{x}_j\bar{c}_j)y_j \tag{9.13}$$

$$= (x_j \oplus c_j)\bar{y}_j + (\overline{x_j \oplus c_j})\bar{y}_j \tag{9.14}$$

Now let $$r_j = x_j \oplus c_j \tag{9.15}$$

Then, $$S_j = r_j\bar{y}_j + \bar{r}_jy_j \tag{9.16}$$

$$= r_j \oplus y_j \tag{9.17}$$

Similarly, $$c_{j+1} = x_jy_j(c_j + \bar{c}_j) + x_jc_j + (x_j + \bar{x}_j)y_jc_j \tag{9.18}$$

$$= (x_j\bar{c}_j + \bar{x}_jc_j)y_j + x_jc_j \tag{9.19}$$

$$= r_jy_j + x_jc_j \tag{9.20}$$

A circuit implementing these equations is shown in Fig. 9.6. Since Eqs. (9.8) and (9.17) are similar in form and so are Eqs. (9.12) and (9.20),

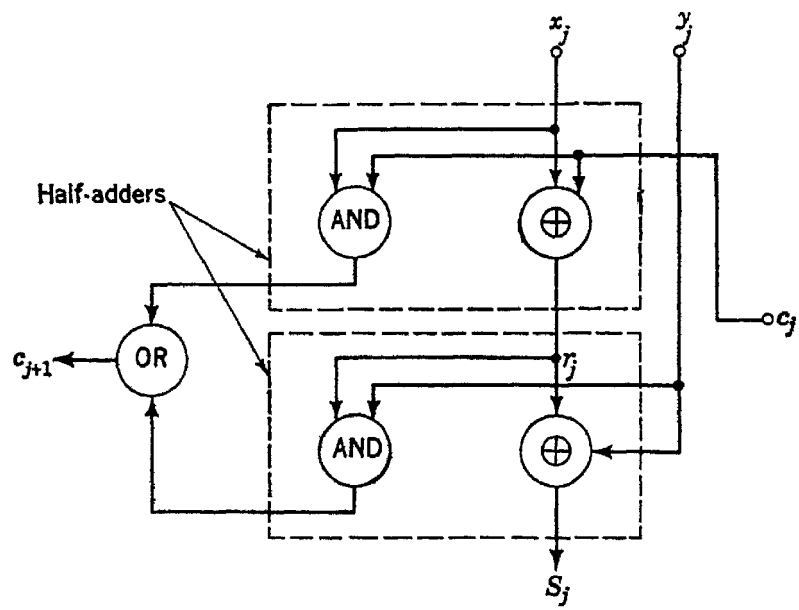



Fig. 9.6 A variant of the two-half-adders circuit

it is not surprising that the two adder circuits look very similar. Each consists of two half-adders supplemented by an OR element. The differences lie solely in the inputs to the ring-sum elements.

Some other logic circuits for implementing the half-adder using AND, OR, and NOT elements are shown in Fig. 9.7. All of these are easily obtained from the Boolean expressions.

A combination of two half-adders to make a full adder (using synchronous circuitry) is shown in Fig. 9.8. A chain of these adders can be connected for parallel addition in the same way as the three-input adder of Fig. 9.3. Terminals $x_j$, $y_j$, and $c_j$ would be used as input terminals in

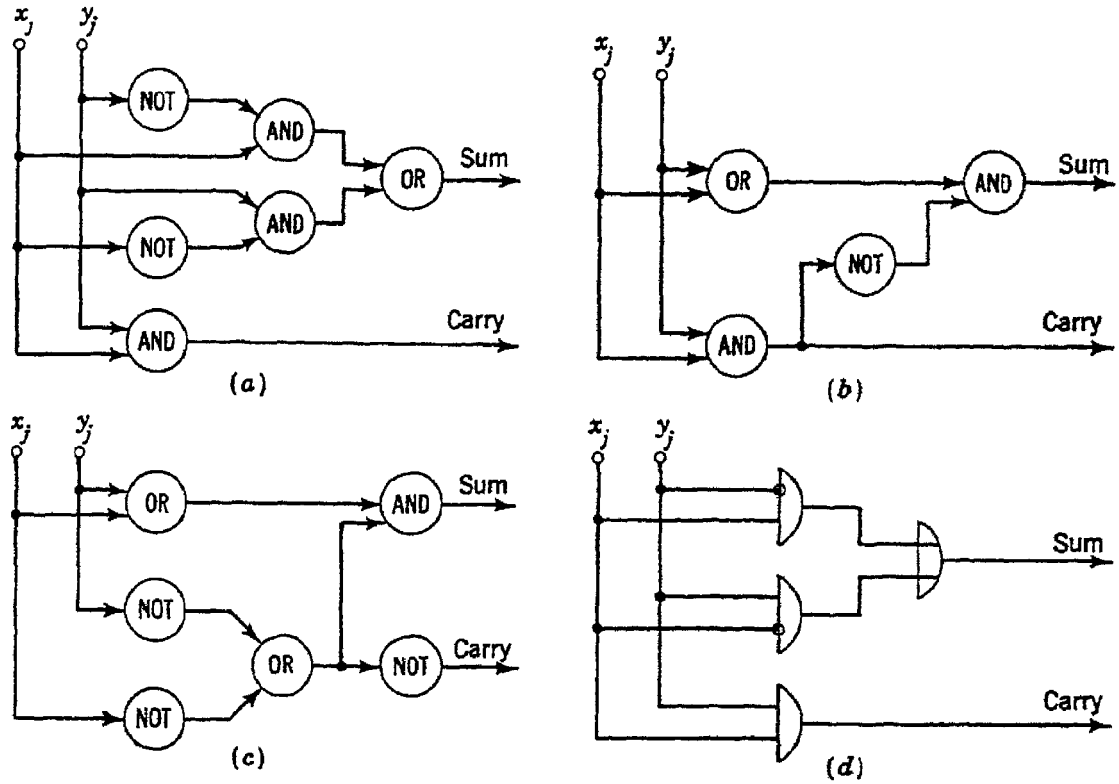



Fig. 9.7 Some half-adders

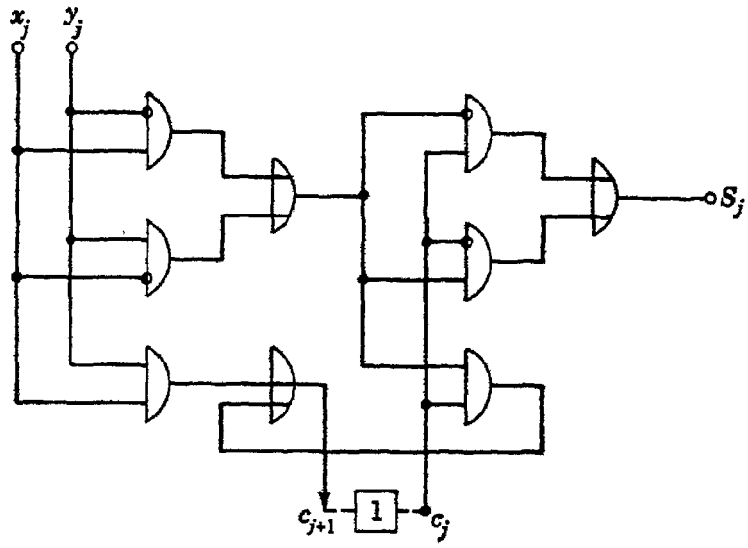



Fig. 9.8 Full adder made of two half-adders

this connection, and $S_j$ and $c_{j+1}$ would be the output terminals. To use the adder for serial addition it is necessary to modify it by connecting points $c_{j+1}$ and $c_j$ through an electrical delay of one pulse period, as shown by the broken lines in Fig. 9.8. The reader will find it helpful to trace the action of the circuit in the serial mode by determining the pulse sequences produced at points within it by various addend and augend sequences.

**Carry propagation.** At a casual glance it might appear that two $n$-digit binary numbers could be added in the parallel mode $n$ times as rapidly as in the serial mode. A more careful consideration shows, however, that this is not so and that, although $n$ adders operating in parallel upon the $n$ bits of the addend and the $n$ bits of the augend do indeed form the sum more rapidly than a single adder operating sequentially, the speed gain is seldom as great as $n$ times. This is so because there is in each adder a slight time delay between reception of its three inputs and formation of its sum and carry outputs. Although two of the three inputs to each adder are from external sources and can be applied simultaneously to all stages, the third input to each stage is obtained from the next less significant stage. Thus it is possible that the addend and the augend may be such as to form a carry in the least significant position, which then produces carries successively in each more significant position. (The addition of 1010101 to 0101011, for example, produces this effect.) In such an extreme case the last stage does not produce its correct output until the carry signal has propagated through all the less significant stages.

The simplest way to treat the phenomenon of carry propagation in the design of the parallel adder is to employ the conservative method of always waiting long enough for a full-length carry propagation to occur before using the output signals. This method has been used in many parallel computers, but, as faster machines are built, the time lost in this method can be appreciable, since full-length carries occur only infrequently. Investigations carried out at the Institute for Advanced Study by Burks, Goldstine, and von Neumann have shown that, in adding random numbers of 40-bit length, the average maximum length of carry of 1's is only 4.6 positions. A system that detects the completion of all carries and then senses the outputs should allow, for 40-bit numbers, an eightfold increase in speed over a system that always allows time for 40-position carry propagation.

In order to study the requirements upon such a circuit, let us first examine in some detail the conditions that give rise to a carry. The general condition for the creation of a carry $c_{j+1}$ in the $j$th position is

$$c_{j+1} = x_j y_j + x_j c_j + y_j c_j \tag{9.21}$$

Now $c_{j+1}^G$, the condition for the creation of a carry $c_{j+1}$ in the $j$th position independently of the presence of a carry into that position, is

$$c_{j+1}^G = x_j y_j \tag{9.22}$$

Thus $c_{j+1}^P$, the condition under which a carry is propagated through the

$j$th position, is

$$c_{j+1}^P = c_{j+1}\overline{[c_{j+1}^G]} \tag{9.23}$$
$$= (x_jy_j + x_jc_j + y_jc_j)(\overline{x_jy_j}) \tag{9.24}$$
$$= x_j\bar{y}_jc_j + \bar{x}_jy_jc_j \tag{9.25}$$
$$= (x_j \oplus y_j)c_j \tag{9.26}$$

It is evident that the conditions for $c_{j+1}^G$ and for $c_{j+1}^P$ are mutually exclusive; that is, an adder stage that generates a carry cannot also propagate a carry. Thus, in each adder stage, the following mutually exclusive possibilities exist:

1. No carry occurs, either because $x_j = y_j = 0$ or because $x_j \neq y_j$ and $c_j = 0$.
2. A carry of 1 is produced immediately because $x_j = y_j = 1$.
3. A carry of 1 is produced after an input of $c_j = 1$. For this to occur, we must have $x_j \neq y_j$.

Since no adder stage can participate twice in the carry function, completion of all carries in the parallel adder can be sensed by examining the carry outputs of all stages. This may be facilitated by encoding the carry output in a 1-out-of-2 code, in which one and only one of two carry leads must be energized after formation of the carry digit. When one of these two output leads has been energized in every stage of the adder, the carry is complete. A block diagram of such a scheme is shown in Fig. 9.9. The extra circuitry to be included in the adder to implement this system may be determined from the expression for $c_{j+1}^0$, the zero-carry output condition:

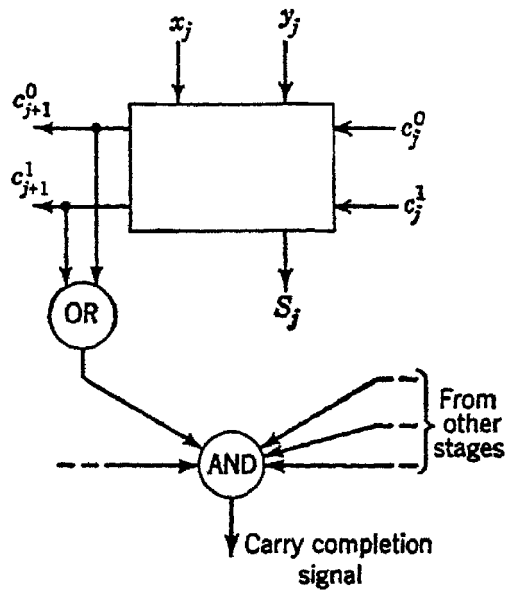



Fig. 9.9 Block diagram of circuit that senses completion of carries

$$c_{j+1}^0 = \bar{x}_j\bar{y}_j + (x_j \oplus y_j)c_j^0 \tag{9.27}$$
$$= \overline{(x_j + y_j)} + (x_j \oplus y_j)c_j^0 \tag{9.28}$$

The complete adder stage appears in Fig. 9.10. The rightmost portion of this circuit is simply the three-input adder of Fig. 9.5, and the left part is the circuitry needed for carry-completion sensing.

An equivalent arrangement in slightly different form, due to Gilchrist, Pomerene, and Wong,[1] is shown in Fig. 9.11. Their investigations of carry propagation showed that for circuits of this type, which form both

0 and 1 carries, the average maximum length of carry (whether 0 or 1) in addition of 40-bit random numbers is 5.6 positions.

The circuits of Fig. 9.10 and Fig. 9.11 are particularly well suited for use in systems with static direct-coupled circuitry, since signals in such systems are produced as voltage levels. Thus there is no problem of synchronizing the signals presented to the AND gate that generates the carry-completion signal (see Prob. 9.3).

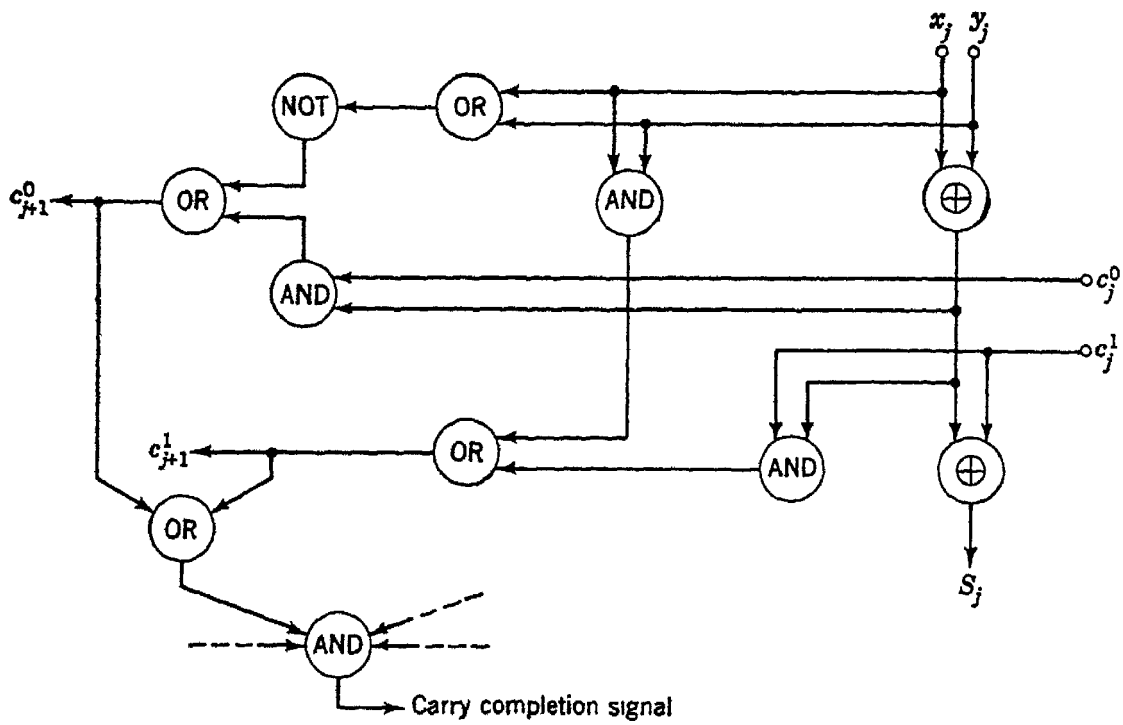



Fig. 9.10 Adder stage incorporating signals for sensing carry completion

Other solutions are also possible for the problem of carry propagation in parallel addition. In one such solution, each adder stage is a two-input adder (half-adder) instead of a three-input adder, and each stage forms a digit of the partial sum and a carry digit. Addition is performed sequentially; addend and augend are applied as the first set of inputs, the partial-sum digits and the carry digits are applied as the next set of inputs, and subsequent partial sums and carries are applied in turn until no further changes result.

It is interesting to note that the relay adder of Fig. 9.1 is, in effect, free from delays due to carry propagation. This is so because the action of the relays in each stage is independent of that of earlier stages and is determined only by digits $x_j$ and $y_j$. Transmission of electrical signals through the network formed by the relay contacts takes place in a time that is negligibly short in comparison with the relay operating time. Consequently, the sum digits are all available after a time equal to the operating time of the slowest relay.

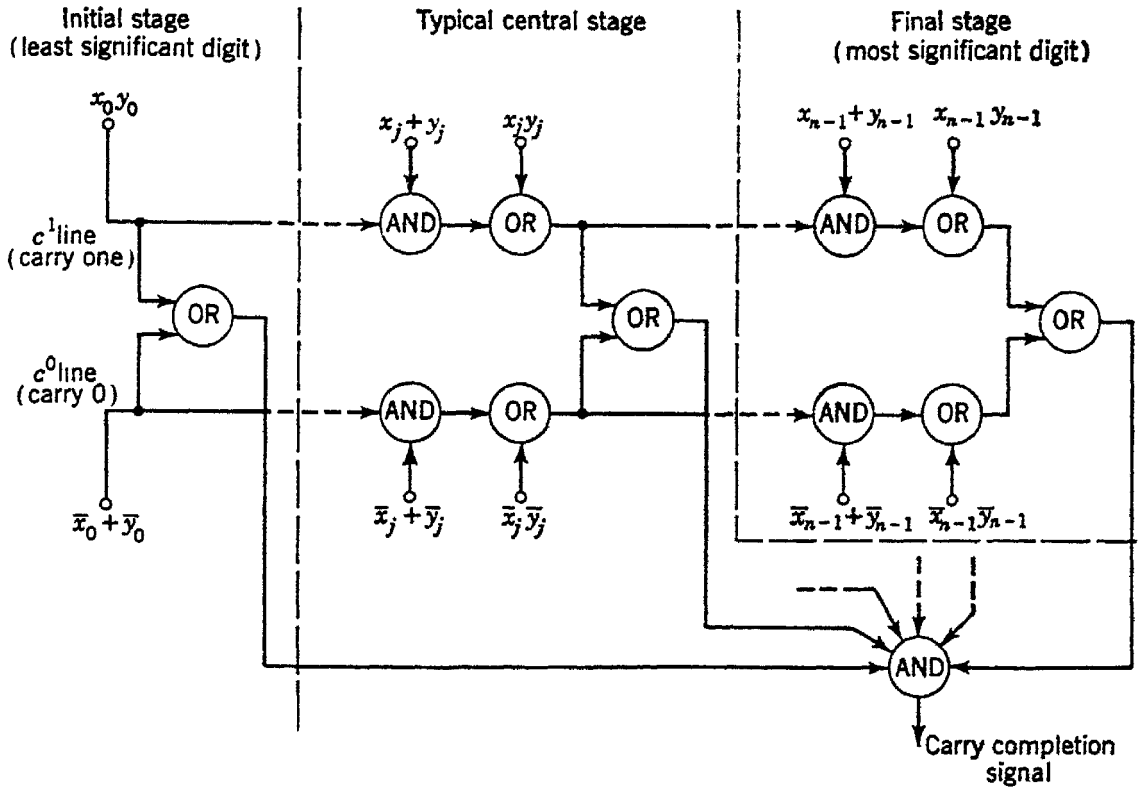



Fig. 9.11 Circuit for sensing completion of carries

**Adders of the accumulator type.** All the adder circuits we have so far considered have been constructed from logic devices—AND, OR, NOT, and RING SUM elements. In many computers the electronic circuitry corresponds closely to these logic elements. However, it is also possible to construct adders in which this close correspondence does not appear, even though the results of the circuit behavior can be expressed in terms of Boolean algebra. An example of such a system is the accumulator-type adder, each stage of which consists of a flip-flop, or bistable element. (Electronic circuits for the flip-flop are discussed in Chap. 10.) Each flip-flop has input leads that allow the insertion of a digit of the addend, as well as a complementing input which reverses the state of the flip-flop if it receives a 1 signal. An adder stage constructed around a flip-flop is shown in Fig. 9.12.

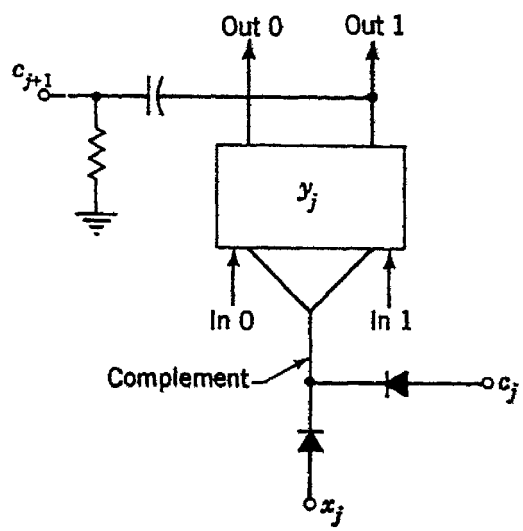



Fig. 9.12 A stage of an accumulator adder

In this adder the augend digits $x_j$ are applied in the form of short

pulses. Such pulses might be obtained, for example, from a register of flip-flops upon application of a single complementing pulse to all stages of the register. This $x_j$ pulse applied to the complement input of the $y_j$ flip-flop makes that flip-flop assume a state $S$ given by

$$S = x_j \oplus y_j \tag{9.29}$$

The transition of the 1 output lead from the $y_j$ flip-flop is then differentiated by the $RC$ circuit. A positive pulse appearing at $c_{j+1}$ is evidence of a carry. (This is equivalent to assuming that, when the flip-flop is storing the digit 1, the 1 output lead is at the lower of its two stable potentials.) The pulse from $c_{j+1}$ is then applied to the carry input of the next more significant adder stage. The timing of the circuit must be such that the $x_j$ pulse has been removed from its input before the $c_j$ pulse is applied. If this is done, then the state assumed by the $y_j$ flip-flop is given by

$$S = x_j \oplus y_j \oplus c_j \tag{9.30}$$

and the circuit forms the sum correctly. However, if the $c_j$ and $x_j$ pulses are not separated far enough in time, the two diodes function simply as an OR element instead of as a RING SUM element, and the circuit does not operate correctly. In practice, delay elements are sometimes needed in the carry lines between stages.

Many variations of the accumulator principle are possible, but this one example should suffice to demonstrate the fundamental features of the method.

## 9.3 Subtractors

Circuits for subtraction can be devised that are analogous to the adders of the previous section. The truth table for a complete subtractor appears in Table 9.2, and the equivalent Boolean expressions are given in Eqs. (9.31).

$$\begin{aligned} d_j &= \bar{m}_j\bar{s}_j b_j + \bar{m}_j s_j \bar{b}_j + m_j\bar{s}_j\bar{b}_j + m_j s_j b_j \\ b_{j+1} &= b_j\bar{m}_j + b_j s_j + \bar{m}_j s_j \end{aligned} \tag{9.31}$$

We may well anticipate a similarity between subtractor and adder circuits, since the entries in the $d_j$ column of Table 9.2 are precisely those of the sum column of Table 9.1, and the $b_{j+1}$ and $c_{j+1}$ columns differ only in that the first four entries of one are the last four of the other.

Figure 9.13 shows a three-input subtractor, the circuit for which follows almost directly from the entries of Table 9.2. Like the three-input adder, this circuit can be used for either serial or parallel operation. In the parallel connection, each $b_{j+1}$ terminal is connected to the $b_j$ terminal of the subtractor in the next more significant digit position. For

serial operation the $b_{j+1}$ terminal is connected to the $b_j$ terminal through a one-pulse-period delay element (or through an appropriate storage and timing circuit if asynchronous circuitry is used).

**Table 9.2**
**Subtractor truth table**

| Minuend $m_j$ | Subtrahend $s_j$ | Borrow $b_j$ | Difference $d_j$ | Next borrow $b_{j+1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

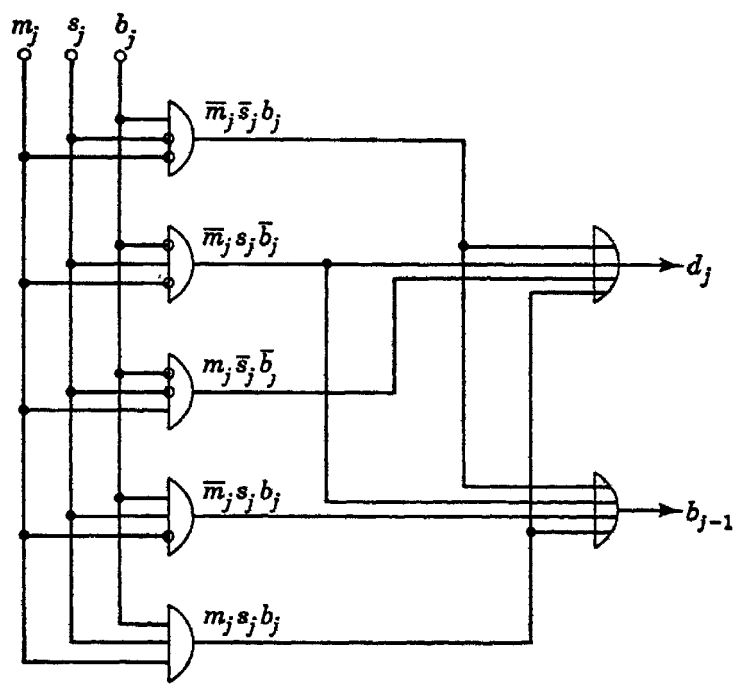



Fig. 9.13 A three-input subtractor

The subtractor of Fig. 9.13 is a circuit somewhat more complex than the adder of Fig. 9.4, in that it has a larger total number of gate inputs. The circuit of Fig. 9.13 has 23 total gate inputs; that of Fig. 9.4 has only 18. The reader may find it stimulating to devise other arrangements equivalent to Fig. 9.13 which may be simpler.

**Half-subtractors.** Subtraction, like addition, can be performed in two steps if two-input circuits are used. The table that describes such a circuit appears in Table 9.3. Figure 9.14 shows two ways in which two

half-subtractors can be interconnected to form a full subtractor. The reader may verify the functioning of these circuits much as we verified the two arrangements of half-adders in Fig. 9.5.

**Table 9.3**
**Truth table for the half-subtractor**

| $m_j$ | $s_j$ | $d_j$ | $b_{j+1}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

The circuit of the half-subtractor can be realized in several ways. Two possibilities are shown in Fig. 9.15. A complete subtractor made of two half-subtractors is shown in Fig. 9.16. Like the adder of Fig. 9.8,

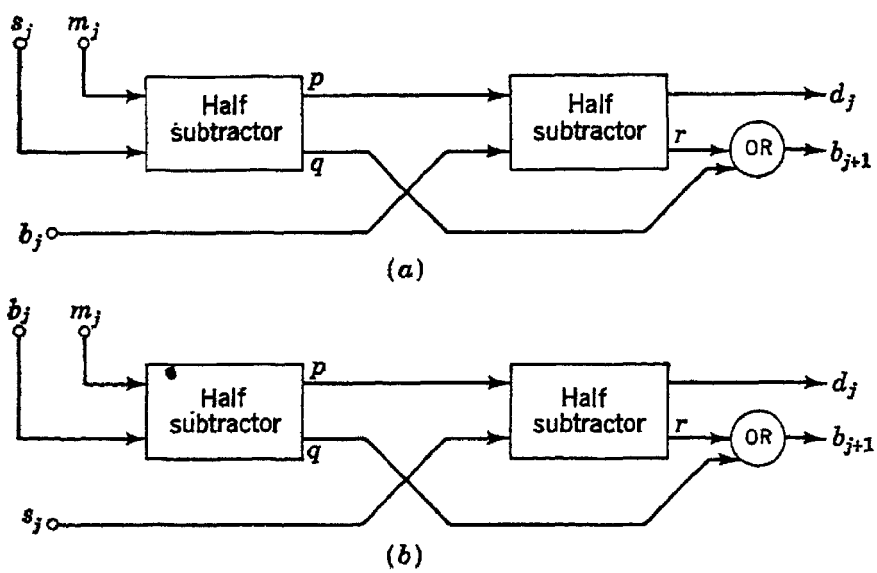



Fig. 9.14 Two ways of connecting two half-subtractors to make a whole subtractor

this circuit can be repeated in a chain to form a parallel subtractor, each $b_{j+1}$ output terminal being connected to the $b_j$ input terminal of the subtractor in the next more significant digit position, or it may be used with serial inputs if the $b_{j+1}$ terminal is connected back to the $b_j$ terminal through a one-pulse-period delay.

Many other interesting arrangements of these circuits can be made. Figure 9.17, for instance, shows a circuit in which a half-adder and a half-subtractor are connected in such a way as to form a full subtractor. It is even possible to connect two half-adders in such a way that the resulting circuit is a full subtractor. Such a circuit appears in Fig. 9.18.

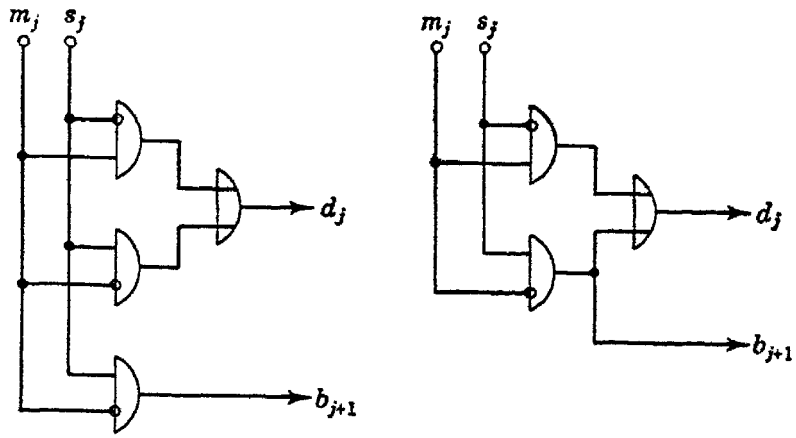



Fig. 9.15 Two circuits for half-subtractors

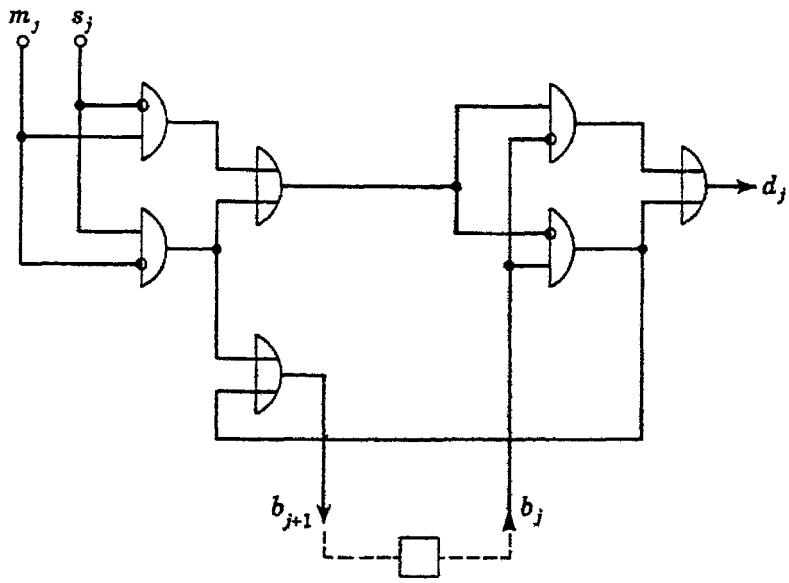



Fig. 9.16 A full subtractor made of two half-subtractors

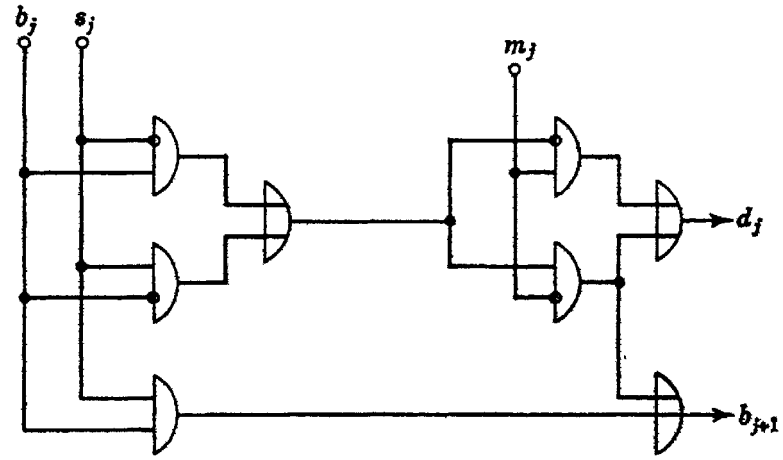



Fig. 9.17 A full subtractor made of a half-adder and a half-subtractor

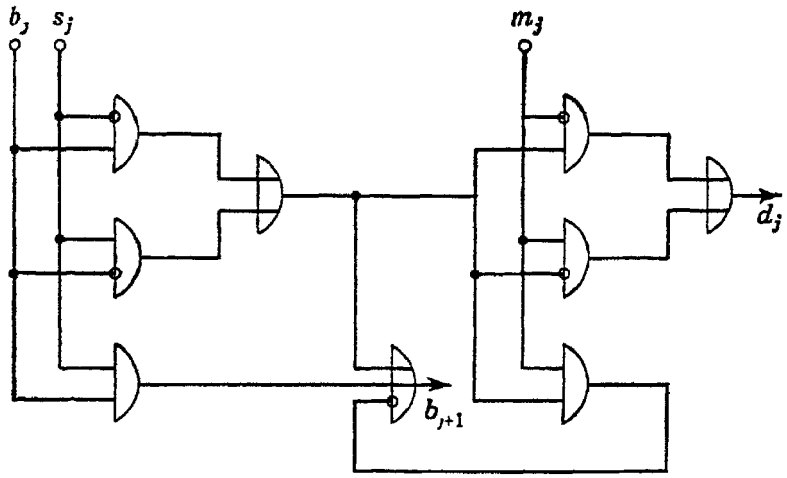



Fig. 9.18 A full subtractor using two half-adders

## 9.4 Addition and subtraction of positive and negative numbers

In building an arithmetic unit for a computer from the adders and subtractors of the preceding section, the designer must exercise care in arranging the manipulation of the numbers. Thus, if two positive numbers are to be added, it is certainly possible to use an adder circuit. The adder may also be used to add two negative numbers, if a negative sign is affixed to their sum. However, both these operations can also be carried out by a subtractor if complements are used. The addition of a positive and a negative number or the subtraction of a positive number from a positive number can also be done using either an adder or a subtractor.

Let us make a brief examination of these possibilities. We shall first show how an adder can be used to perform both addition and subtraction. Addition of positive numbers is straightforward, of course, since this is precisely what an adder does best. Addition of negative numbers is also simple if each number is presented as a magnitude and a minus sign. Auxiliary sensing circuits must be provided to detect the presence of the two minus signs and to affix a minus sign to the sum of the magnitudes. Addition of a positive number and a negative number or, what is the same thing, subtraction of a positive number from a positive number can be done with an adder if the negative number is put in complement form and added. As we have seen in Chap. 7, either the two's complement or the one's complement may be used. Although the one's complement is slightly easier to implement, since all digit positions are treated in the same way, it has the disadvantage of requiring an end-around carry. Two's complementation avoids the end-around carry at the expense of slightly greater complexity in the circuits that perform the complementation.

Consider three full-adders connected for parallel addition, as shown in

Fig. 9.19. If two positive numbers or two negative numbers are being added, then the appearance of a 1 on line $c_3$ is evidence of overflow; i.e., the sum is too large to be expressed by 3 bits. However, if a positive number is being added to the two's complement of a negative number, the appearance of a 1 on $c_3$ indicates that the result of the process exceeds zero, and the binary group $S_2S_1S_0$ constitutes the correct 3-bit output. Appearance of a 0 on $c_3$ (when a positive number and a two's complement number are being added) indicates that the result is itself the two's complement of a negative number, and the group $S_2S_1S_0$ must be complemented to obtain the magnitude of the negative result.

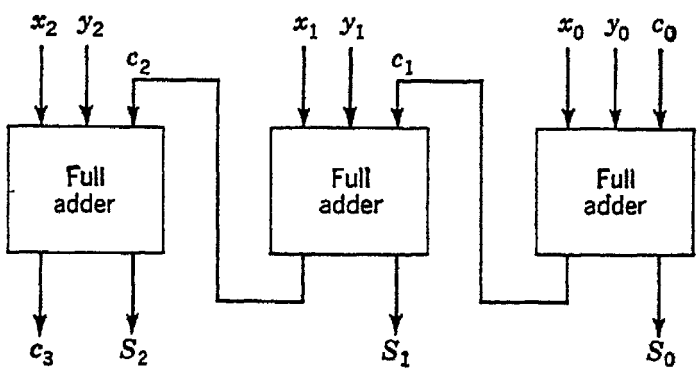



Fig. 9.19 A three-digit parallel adder

If one's complementation is being used, the appearance of a 1 on $c_3$ indicates that the result is positive. However, since the group $S_2S_1S_0$ is deficient by 1 unit, the 1 on $c_3$ can be used to make up this deficiency by adding it in at $c_0$ through an end-around carry. That is, in forming $x - y$ by one's complementation in an $n$-digit system, we form $x + 2^n - 1 - y$. If $x > y$, then $2^n - 1 + x - y \geq 2^n$, and a carry from the $n$th digit position is generated. Adding this in on the right forms $2^n + x - y$, but, since $2^n$ cannot be represented in an $n$-digit system, this quantity appears as $x - y$.

If a 0 appears on $c_3$, then the result is itself the complement of a negative number, and the group $S_2S_1S_0$ must be complemented to obtain the magnitude of the negative result. An exception occurs, however, with one's complementation, when $c_3 = 0$ and all the sum digits are 1. This condition occurs whenever a number is added to its own one's complement and is an indication that the correct sum is zero. (It can also occur whenever a number consisting of all 1's is added to a number that is all 0's, but this situation is easily distinguished.)

It is interesting to note that, if one's complementation is employed, there are two ways to represent the number zero. One of these, called "plus zero", consists of all 0's. The other, called "minus zero", is the one's complement of plus zero and consists of all 1's. Arithmetic units

that use an adder can generate only the minus-zero form, and the plus-zero form occurs only if it is inserted from external sources.

Now let us turn to the use of a subtractor, again taking a simple 3-bit unit, as shown in Fig. 9.20. A positive number is readily subtracted from a positive number by applying the subtrahend digits to the $s$ inputs of the subtractor and the minuend digits to its $m$ inputs. A positive result is manifested by the appearance of a 0 on the $b_3$ line; a negative result is indicated by a 1 at that point. If the result is negative, the digits $d_2d_1d_0$ constitute the two's complement of the magnitude of the result and must be complemented if the magnitude is desired. If the

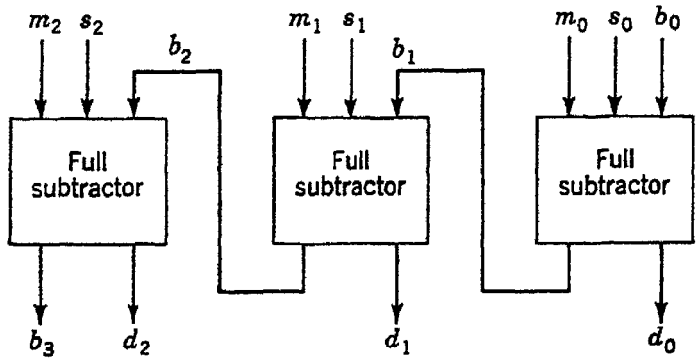



Fig. 9.20 A three-digit parallel subtractor

two's complement form is desired, the digits can be used as they stand. Conversion of the two's complement form to one's complement form is accomplished by subtracting 1 unit from the group. One way of doing this is by what might be called an "end-around borrow", the borrow signal from the highest-order subtractor being applied to $b_0$, the borrow input for the lowest-order subtractor.

If a negative number is to be subtracted from a negative number, the subtrahend digits can be applied to the $m$ inputs and the minuend digits to the $s$ inputs, neither number being complemented. The result can then be put in magnitude form (if it is not in that form already) and a negative sign affixed if the minuend was larger or a positive sign if the subtrahend was larger.

Just as an adder can be used to subtract by adding the complement, so a subtractor can be used to add by subtracting the complement. This method can be used to subtract a negative number from a positive number or, what is the same thing, to add two positive numbers. Of course, the subtractor can be used directly to add a positive number and a negative number without complementation if the positive number is applied to the minuend inputs and the negative number to the subtrahend inputs. (The negative number is assumed to be a signed magnitude.) In adding two positive numbers by means of a subtractor, either may be complemented initially. If the complemented number is then treated as sub-

trahend, then the digits of the result always form the correct sum. If the complemented number is treated instead as the minuend, the digits of the result form the complement of the correct sum. For example, let $x_1$ and $x_2$ be two positive numbers to be added. We complement one of them, say $x_2$, to form $2^n - x_2$ (two's complement). Then, if $2^n - x_2$ is subtracted from $x_1$, we form $x_1 + x_2 - 2^n$, which is indistinguishable from $x_1 + x_2$ in a machine of $n$-digit capacity. If, however, $x_1$ is subtracted, we form $2^n - (x_1 + x_2)$, which is the two's complement of the desired sum $x_1 + x_2$.

If a positive number is to be subtracted from a positive number or a negative number from a negative number, the orthodox way of carrying out the operation with a subtractor is simply to apply the subtrahend digits and the minuend digits respectively to the subtrahend and minuend inputs of the subtractor. This process can also be carried out using complements, although there is no advantage to doing so. The complement of the subtrahend magnitude and the complement of the minuend magnitude are applied respectively to the subtrahend and minuend inputs. If the result appears in complement form, it should be recomplemented and given a plus sign; if the result appears uncomplemented, it should be given a minus sign and otherwise left unaltered.

In practice most designers have preferred to build arithmetic units that contain either an adder or a subtractor but not both. The adder is more often used than the subtractor, probably because addition is a commutative operation, which allows addend and augend to be applied to either input, whereas subtraction is not commutative, and a subtractor's associated control circuitry must, therefore, be somewhat more complex in order to apply minuend and subtrahend to the proper inputs in all cases.

## 9.5 Sign conventions and manipulations

In the preceding sections we have regarded numbers as being represented variously as signed magnitudes, one's complements, or two's complements. The various representations cannot be mixed indiscriminately in the computer, and some standard conventions must be adopted in any system or section of a system. One commonly used practice is to store numbers in the memory as signed magnitudes and to sense the sign of each number to determine whether it must be complemented before the current program step is executed. This means that, in a completely serial arithmetic unit, in which the digits of a number are presented sequentially, starting with the least significant, the sign digit must appear first. In the parallel arithmetic unit, the sign digit may be at either end of the number, since all digits are presented simultaneously.

Let us explore in some detail the considerations involved in sign determination for a system in which numbers are stored as signed magnitudes and in which subtraction is accomplished by addition of the two's complement. Even in doing addition, complementation may be necessary, since either the addend or the augend or both may be negative, and their signs will determine the action to be taken. The steps involved in adding $A_1$ and $A_2$ for each possible sign combination are shown in Table 9.4.

**Table 9.4**
**Steps involved in addition**
**($A_1$ and $A_2$ are regarded as signed magnitudes)**

| Sign of $A_1$ | Sign of $A_2$ | Steps in addition |
|---|---|---|
| + | + | Add directly and put + sign on result. |
| + | − | Complement $A_2$ and add. If result is +, affix + sign. If not, recomplement and affix − sign. |
| − | + | Complement $A_1$ and add. If result is +, affix + sign. If not, recomplement and affix − sign. |
| − | − | Add directly and put − sign on result. |

The various possibilities involved in subtracting $A_2$ from $A_1$ are shown in Table 9.5.

**Table 9.5**
**Steps involved in subtraction of $A_2$ from $A_1$**
**($A_1$ and $A_2$ are regarded as signed magnitudes)**

| Sign of $A_1$ | Sign of $A_2$ | Steps in subtraction |
|---|---|---|
| + | + | Complement $A_2$ and add. If result is +, affix + sign. If not, recomplement and affix − sign. |
| + | − | Add directly and put + sign on result. |
| − | + | Add directly and put − sign on result. |
| − | − | Complement $A_1$ and add. If result is +, affix + sign. If not, recomplement and affix − sign. |

Which, if either, of the two operands should be complemented is a function of the signs of the operands and the nature of the operation to be executed, i.e., addition or subtraction. Table 9.6 summarizes this information, taken from the lists of possibilities in Tables 9.4 and 9.5.

The circuit of Fig. 9.21 shows one way of generating these signals. In a serial computer the "complement $A_1$" or "complement $A_2$" signals would have to be present all during the time that digits of the number were being presented to the complementer. This could be done by letting the output signals set one or the other of two flip-flops, but it is usually more convenient to let the input signals exist during the full word length, since information about the signs is needed in determining the

sign of the result. The circuit of Fig. 9.21 can also be used to supply inputs to the sign-determining circuit of Fig. 9.22, which sets the sign of the result for the four cases in which the sign is not a function of the relative magnitudes of the operands.

**Table 9.6**
**Conditions for complementing operands**
**($A_1$ and $A_2$ are regarded as signed magnitudes)**

| Operation ($A_1 \pm A_2$) | Sign of $A_1$ | Sign of $A_2$ | Complement $A_1$? | Complement $A_2$? |
|---|---|---|---|---|
| + | + | + | No | No |
| + | + | − | No | Yes |
| + | − | + | Yes | No |
| + | − | − | No | No |
| − | + | + | No | Yes |
| − | + | − | No | No |
| − | − | + | No | No |
| − | − | − | Yes | No |

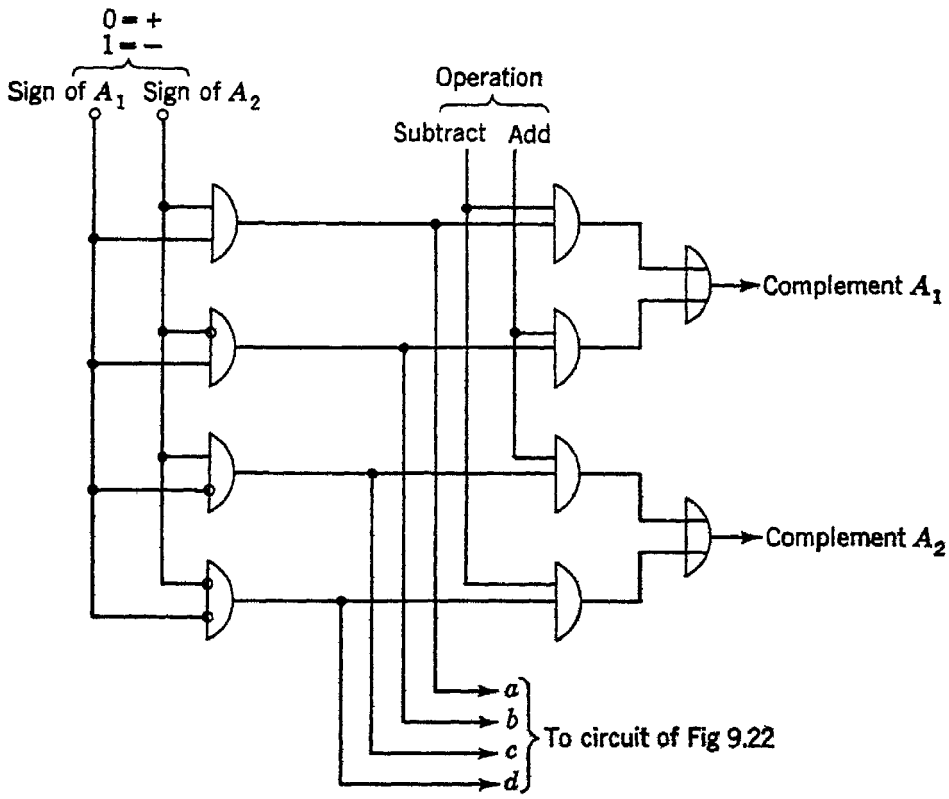



Fig. 9.21 A circuit for generating complementation signals

The two's complement, it will be recalled, can be formed by leaving unchanged all digits on the right end of a number up to and including the least significant 1 and by changing 1's to 0's and 0's to 1's in the more significant positions. This is an awkward operation to implement

directly, because different rules apply to the various digit positions, and for this reason one's complementation is often preferred by designers. On the other hand, since one's complementation can result in an end-around carry, which requires another addition time, the two's complement

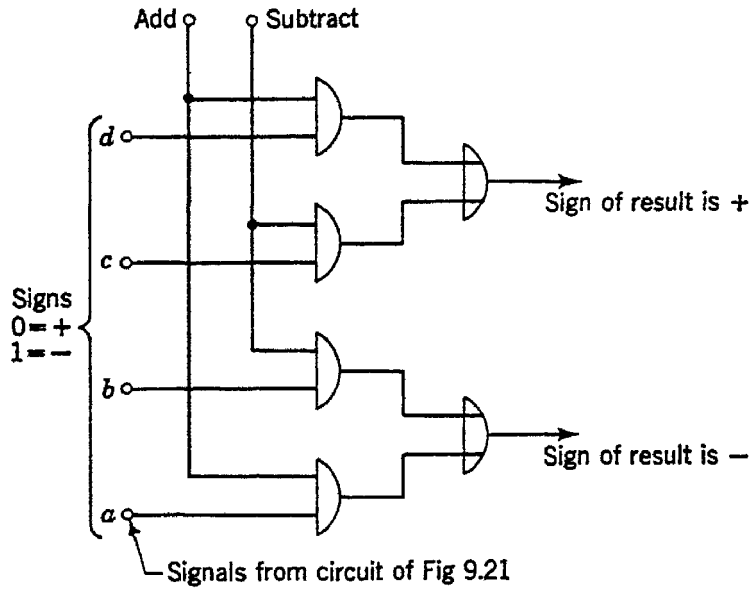



Fig. 9.22 Sign-determining circuit

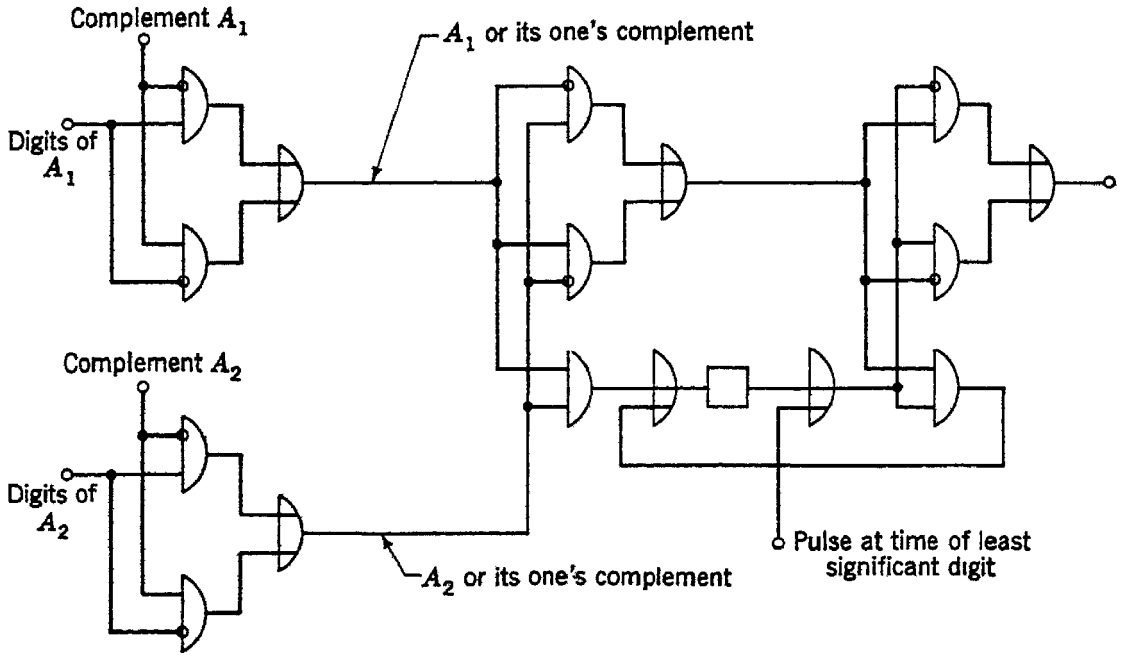



Fig. 9.23 Serial complementer and adder

method is preferred by users. The two methods are easily reconciled, however, since the two's complement is readily formed from the one's complement by adding 1 in the least significant position of the one's complement. The one's complement, of course, is formed simply by changing 1's to 0's and 0's to 1's throughout the number. It is not neces-

sary to use an extra adder, since the computer's adder can be modified to accept the 1 in the least significant position while it is adding the two operands. A circuit for complementing and adding is shown in Fig. 9.23. The reader may verify the behavior of the circuit for simple sequences of $A_1$ and $A_2$ digits with or without complementation.

## 9.6 Simple control circuits

Before continuing with some of the more involved computer operations, such as multiplication, division, shifting, and others, let us first examine some simple control circuits that will be useful in these operations. The simplest and most widely used control element in a computer is the flip-flop, a device having two states, called "SET" and "RESET", or "ON" and

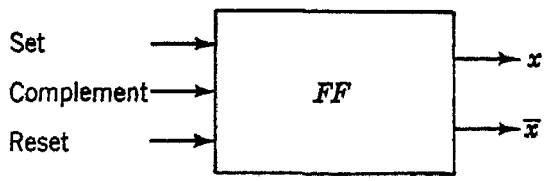



Fig. 9.24 Asynchronous flip-flop symbol

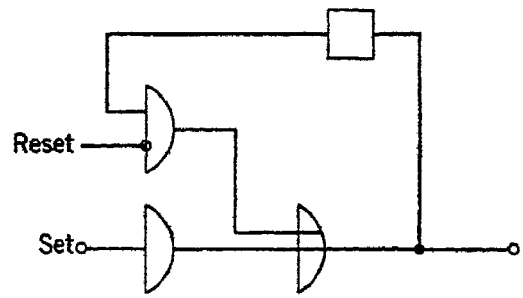



Fig. 9.25 Dynamic flip-flop

"OFF", or "1" and "0". A flip-flop is in a sense a 1-bit memory element, but, in addition to having memory, it has the ability to supply energy to control other circuits. For static-type circuitry the most common flip-flop is the Eccles-Jordan circuit or one of its descendants, consisting of a back-to-back connection of two tubes or two transistors. Such a flip-flop commonly has two outputs $x$ and $\bar{x}$. If output $x$ is in the state corresponding to 1 (either the higher or the lower of the two possible voltage levels), then output $\bar{x}$ will be in the state corresponding to 0. In a well-designed flip-flop either output or both can supply energy to several AND and OR elements. The symbol of Fig. 9.24 is often used for the asynchronous flip-flop. (This is a more general form of the symbol in Fig. 8.52.)

With dynamic circuitry the flip-flop takes the form of a pulse-circulating circuit, as shown in Fig. 9.25. A SET pulse applied to the OR gate appears at its output and is applied to a one-pulse-period delay, which feeds back to another input of the OR gate where it is readmitted. Thus a single SET pulse generates a train of output pulses. The device is then said to be SET, or ON, or "in the 1 state". A pulse applied to the RESET lead inhibits the action of the gate and prevents the passage of the recirculating pulse, thus turning the flip-flop OFF. Of course, the output pulse is slightly delayed with respect to the input pulse, and the delay

line causes some attenuation. Methods of meeting these electrical problems are discussed in the next chapter; for our present purposes we may ignore these effects and assume all gates to be without delay and all delay elements without attenuation.

**Shift registers.** A storage register to hold a group of binary digits can be constructed from a group of flip-flops. The utility of such a register is greatly enhanced, however, if each flip-flop can accept information

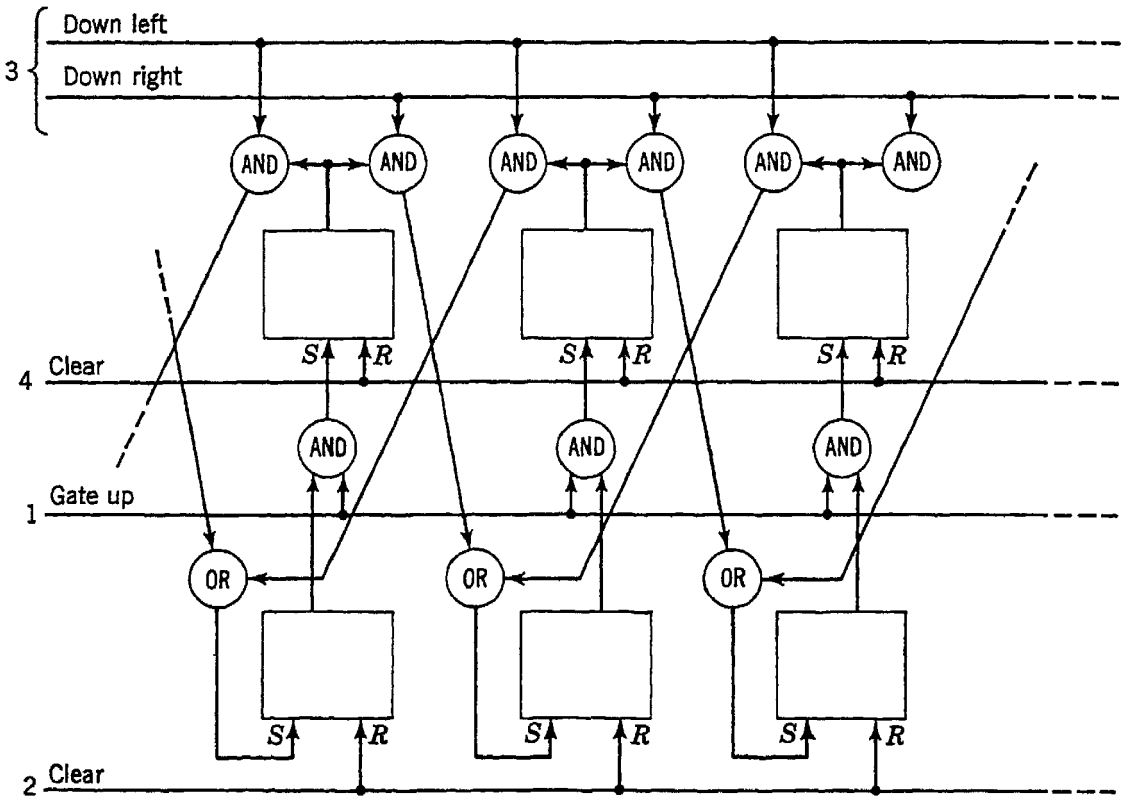



Fig. 9.26 A two-rank shifting register

from or deliver information to the flip-flops on either side of it. Such a register is a very versatile device, a bidirectional shift register. The design of such a device is somewhat complicated by the fact that most types of flip-flop cannot simultaneously accept and deliver information. Thus the shift register must operate either (1) by starting from the end toward which information is to be shifted, clearing each stage in turn, and transferring its contents to the adjacent previously cleared stage; or (2) by simultaneously clearing all stages and storing their contents temporarily in elements which then discharge the information to the adjacent stage to right or left as desired. The second method is the faster and is commonly used in both serial and parallel computers.

A type of shifting system used in some computers employs two registers, which we may call the upper and lower registers. Such a system appears in Fig. 9.26. A four-step process shifts information in the lower register

one position right or left, the upper register being used as a temporary store. These steps are as follows:

1. The contents of the lower register are gated directly up to the upper register.
2. All lower-register stages are reset to zero.
3. The contents of the upper register are gated down to the lower register either one position to the right or one position to the left as desired.
4. The upper register stages are reset to zero.

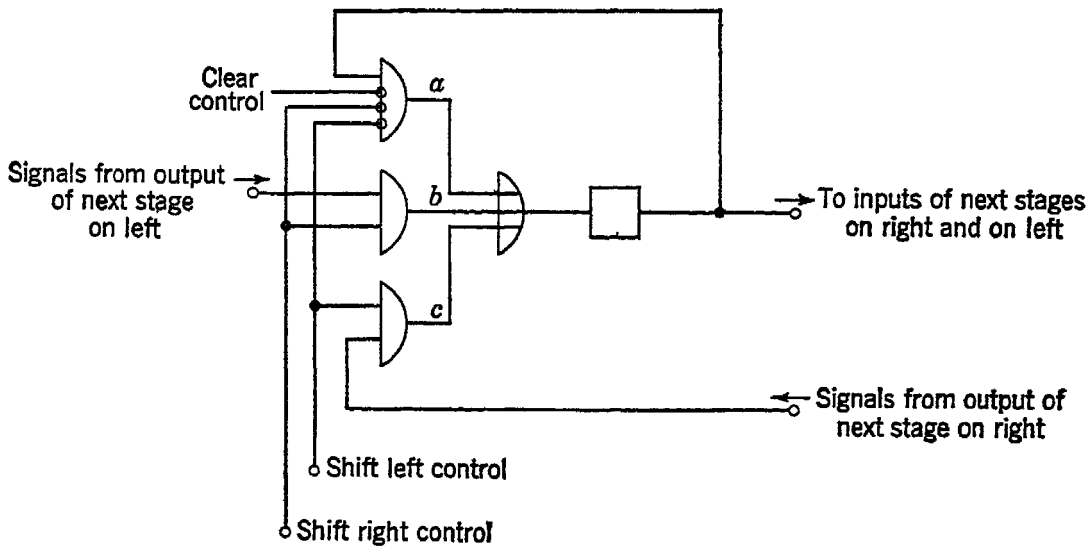



Fig. 9.27 A stage of a bidirectional shift register using dynamic flip-flops

Some types of flip-flop produce an output only when they are changing state. With such devices the upper rank of flip-flops in the circuit of Fig. 9.26 can be replaced by electrical delay lines having a delay time longer than the flip-flop resettling time. The flip-flops are first cleared, and, as a result, 0 or 1 signals are applied to each delay input. The delay outputs are then gated right or left as desired.

The dynamic flip-flop of Fig. 9.25 may also be connected as a bidirectional shift register. One stage of such a circuit is shown in Fig. 9.27. A pulse impressed on the point marked SHIFT RIGHT inhibits the recirculation gate *a* and opens gate *b* from the next stage on the left. Similarly a SHIFT LEFT pulse inhibits the recirculation gate and opens gate *c* from the next stage on the right. In this circuit the delay element that is part of the flip-flop is used as the temporary store that holds the flip-flop information while the stage is being cleared.

A complementing circuit could be included in Fig. 9.27, but is not a necessary part of the shift register. If all stages of a shift register are to be complemented, each stage may include a complementer, in which case complementation requires only one pulse period; or a single complementer

may be used at the output, $n$ pulse periods being needed to shift the full contents of an $n$-stage register through the complementer and back in at the other end of the register. The action of the complementer, of course, is simply replacement of 1's by 0's and 0's by 1's.

**Counters.** A great many ingenious electronic circuits for counting are well described in the literature, and it is not our purpose to review them here. Instead we shall limit our discussions to one or two simple logic circuits with which counting can be done, so that the reader will have some grasp of the organization of this important computer component.

One type of binary counter consists of an interconnection of stages each of which produces a single output pulse after receiving two input pulses. Figure 9.28 shows an example of this type of counter. The first input

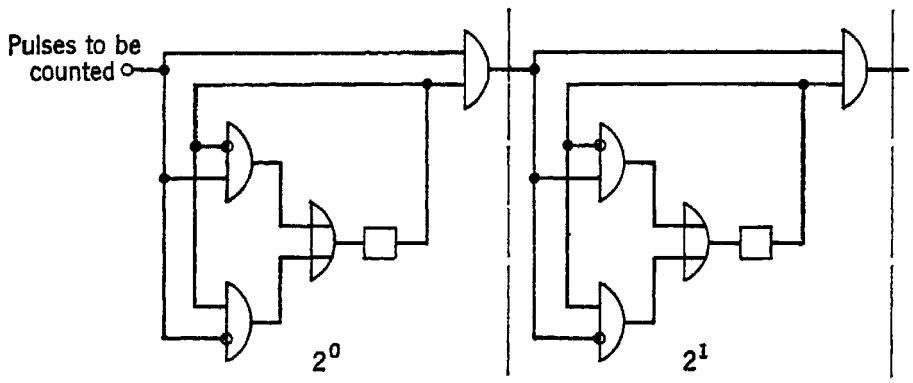



Fig. 9.28 A binary counter

pulse puts the first stage in the 1 state (pulse circulating). A second pulse at any later clock pulse time turns this flip-flop off and admits a pulse to the second flip-flop. This process is repeated in successive stages along the chain. At any time the total count of input pulses is the binary sequence formed by the digits stored in the successive stages, the digit in the input stage being the least significant.

This type of counter can also be implemented with static (d-c) flip-flops, if capacitive coupling is included between the stages. Only the changes of level are then transmitted, and only one of the two changes affects the next stage.

Counters like the preceding will count randomly spaced pulses (which, however, must be in synchronism with clock pulses in synchronous circuits). If the pulses to be counted are always spaced by the same number $k$ of clock pulses, a counter can be devised using much less circuitry, as shown in Fig. 9.29. Let us designate the pattern of pulses stored in the delay line as $x_{k-1}, x_{k-2} \ldots x_1 x_0$, with the $x_0$ pulse emerging from the right end of the delay line at the instant an input pulse is presented to be counted. If $x_0$ is 0, the input pulse is admitted and $x_0$ becomes 1. If $x_0$ is 1, no pulse is admitted and $x_0$ becomes 0. However,

a carry signal is then produced which is delayed so as to arrive in synchronism with $x_1$ and the same process takes place over again. The reader will recognize the circuit before the delay line as a half-adder circuit, which adds 1 to the binary number $x_{k-1}x_{k-2} \ldots x_1x_0$ each time an input pulse occurs. Thus after each input pulse the digit configuration in

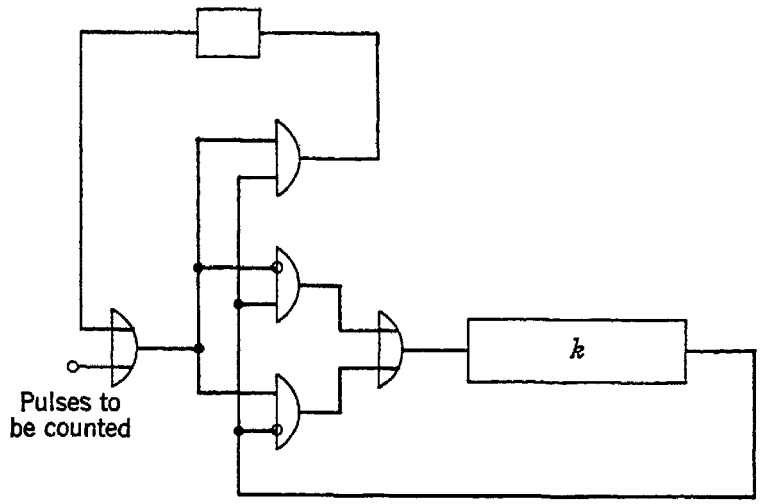



Fig. 9.29 A counter for $k$-spaced pulses

the delay line is the binary representation of the number of input pulses counted. It is evident that $2^k$ pulses are required to cycle this counter through its full count and back to zero.

**Pulse distributors (timing-pulse generators).** In synchronous computers it is frequently necessary to distribute successive clock pulses among a group of $k$ control wires in such a way that wire 1 receives pulse 1,

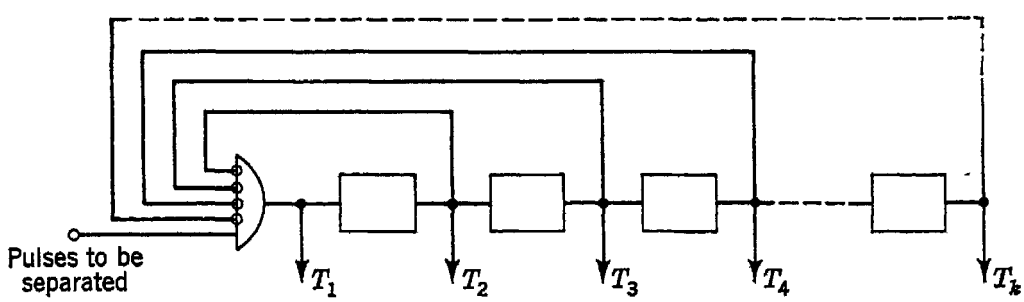



Fig. 9.30 A pulse distributor

pulse $1 + k$, pulse $1 + 2k$, etc., wire 2 receives pulses 2, $2 + k$, $2 + 2k$, etc., on through wire $k$, which receives pulses $k$, $2k$, $3k$, etc. In a serial computer, $k$ might be the number of pulses assigned to a word. One way of performing this separation is shown in Fig. 9.30. The first pulse passes through the input gate at time $T_1$ and on down the chain of 1-unit delay elements. It emerges from these in succession at times $T_2, T_3, \ldots, T_k$ and at each emergence also inhibits the input gate. $T_{k+1}$, which is the next $T_1$, is not inhibited, however, and passes through the gate to start the cycle over.

This circuit requires $k - 1$ delay units, however, as well as a very complicated input gate. The large number of delay units required would be of no concern if the physical devices used were characterized by zero attenuation and zero cost. Because of the actual attenuation, however, output pulses become weaker as they progress down the chain of delays.

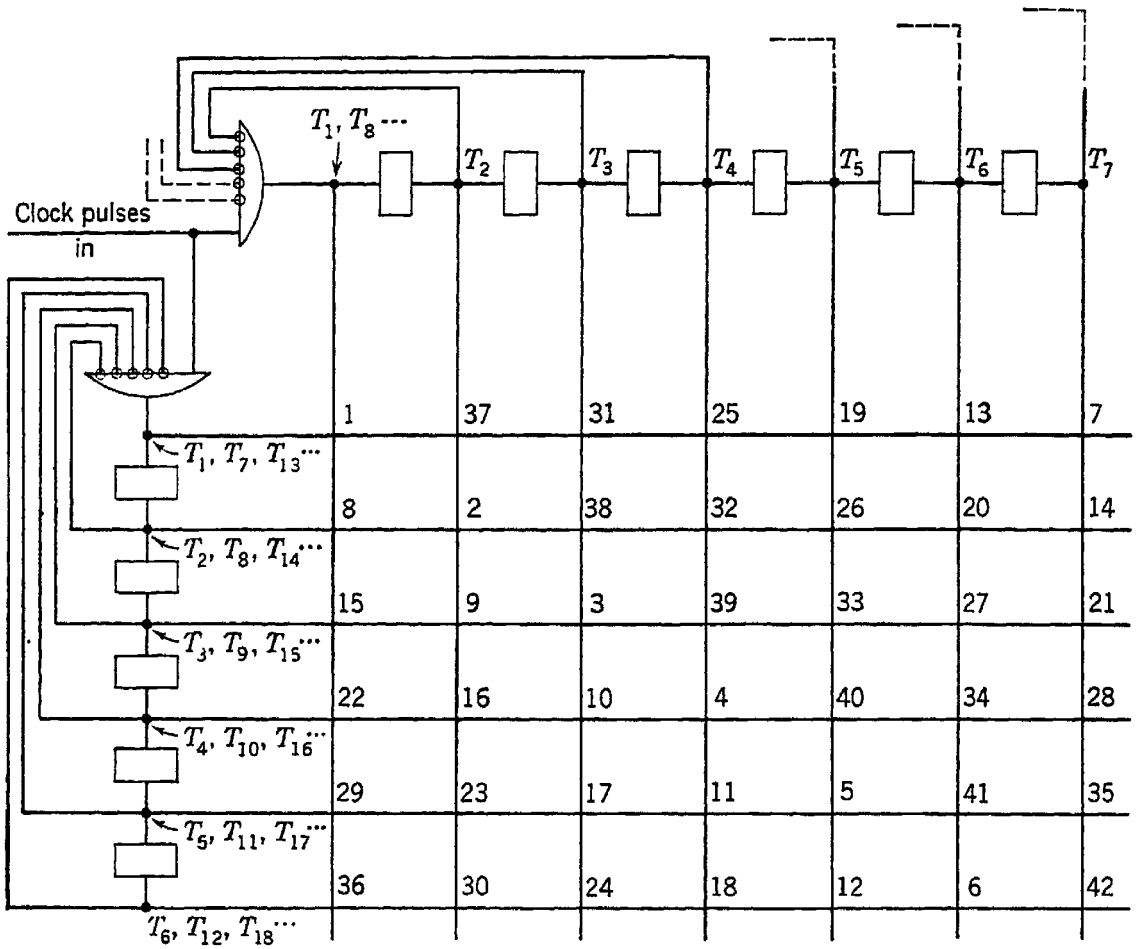



Fig. 9.31 A two-chain pulse distributor

Inhibition of the gate becomes marginal, and so does operation of other circuits driven from the outputs of this circuit. Amplifiers may be incorporated in the chain, but a better way is to use another chain of delay elements. The same function can be performed much more economically if $k$ can be factored into two integers about equal to $\sqrt{k}$. Thus, for $k = 42$, we may use the circuit of Fig. 9.31.

At each numbered intersection, an AND gate can be driven from the two lines to produce a single pulse in synchronism with the clock pulse having the same number.

A three-chain distributor for up to 60 pulses can be made up of elements shown in Fig. 9.32.

The output lead in each chain that is pulsed at each clock pulse $T_1$, $T_2$, . . . , $T_k$ is shown in Table 9.7.

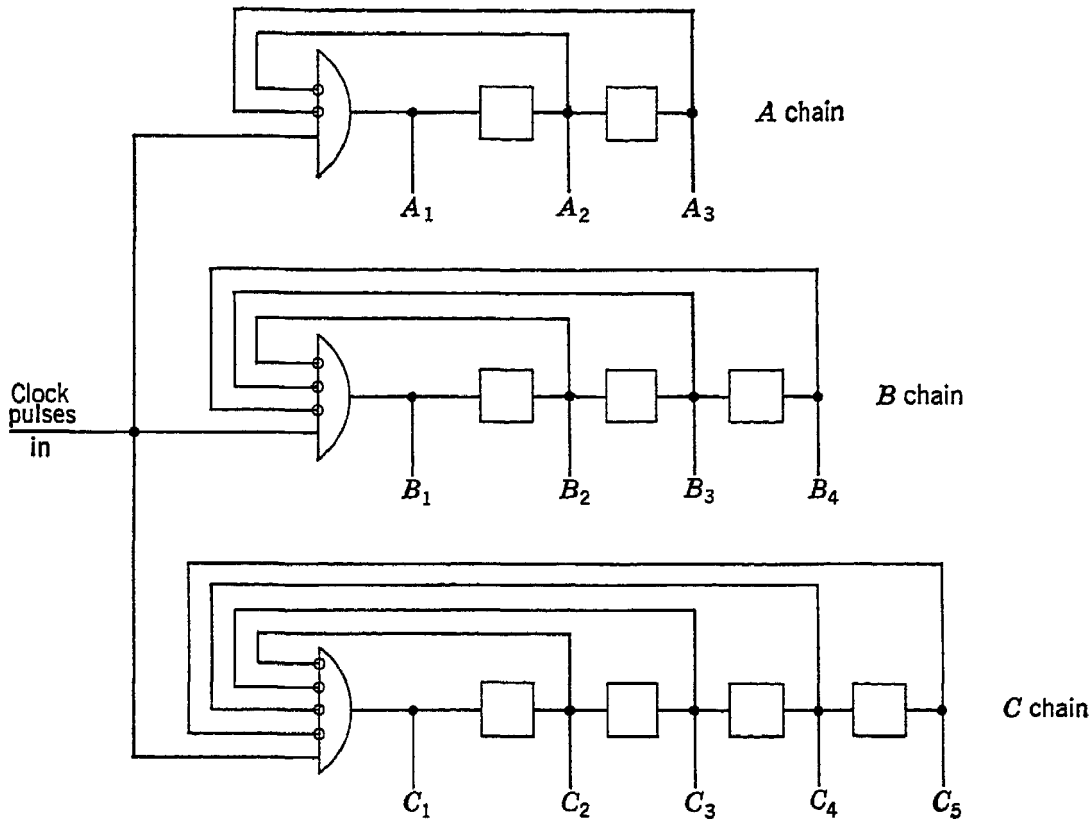



Fig. 9.32 A three-chain pulse distributor

**Table 9.7**
**Output states of three pulse-distribution chains**

| | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ | $T_9$ | $T_{10}$ | $T_{11}$ | $T_{12}$ | $T_{13}$ | . . . |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A$ | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 | 1 | . . . |
| $B$ | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | . . . |
| $C$ | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | . . . |

The table can readily be extended, and will be seen to repeat itself only after 60 pulses. For each of the 60 pulses a unique combination of three leads is activated, one in each of the three chains. Thus, a suitably connected three-input AND gate can be used to produce the desired single pulse.

It may be necessary to reset some of the chains if the product of the number of pulses produced by all of the chains exceeds the number of pulses desired in each cycle of operation. For example, to use chains of three, four, and five, which have a basic cycle of 60 pulses, to time a machine cycle of 48 pulses, we must reset the $C$ chain so that $T_{49}$ and $T_1$ are identical. In the $A$ and $B$ chains, $T_{49}$ already occurs on lead 1, since both 3 and 4 are factors of 48. Resetting of the $C$ chain may be done as shown in Fig. 9.33. Here $T_{48}$ is produced by a three-input AND gate driven by $A_3$, $B_4$, and $C_3$. $T_{48}$, delayed by one period, inhibits the pas-

sage of the pulse that would normally have inhibited the admission of $T_{49}$ to the $C$ chain. Thus $T_{49}$ is admitted as $T_1$.

In general the number of pulses $T_j$ to be produced in each chain should be such that the product of all the $T_j$ should be $\geq k$, where $k$ is the number of pulses to be separated. No two of the $T_j$ should have a common factor

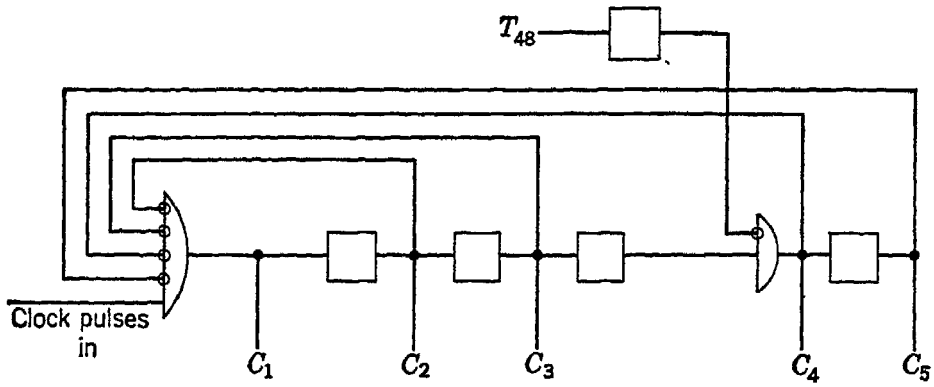



Fig. 9.33 Resetting a pulse-distributor chain

(other than 1) or ambiguity of output will result. The simplest circuit results when $T_aT_b \cdots T_n = k$, since then the $(k + 1)$st pulse arrives in each chain as $T_1$, and it is unnecessary to reset any chains. Any chain for which $T_j$ is not a factor of $k$ must be reset after $k$ pulses to make the $(k + 1)$st pulse be $T_1$.

## 9.7 Methods of binary multiplication[2]

Through the use of some of the circuits described in the preceding section, it is a fairly straightforward matter to construct circuits for performing multiplication. In the binary number system multiplication is a particularly simple process, since the only digits are 0 and 1 and the complete multiplication table takes the form shown in Table 9.8. The multi-

**Table 9.8**
**Binary multiplication table**

| × | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

plication of two multidigit binary numbers is usually executed in just the same way as the pencil-and-paper multiplication of two decimal numbers. First the multiplicand is multiplied successively by the individual digits of the multiplier, starting with the least significant. Then the sum of these partial products, each shifted one position left of the previous partial product, is formed to get the final product. In the computer, it is usually more convenient to add each partial product in as it is formed instead of retaining all partial products and adding them at the end, but

this is merely a manipulative convenience. A more significant feature of binary multiplication is that the simplicity of the binary multiplication table makes formation of the partial products particularly easy. Since each multiplier digit is either 1 or 0, each partial product is either the multiplicand itself or 0. Thus one way to perform multiplication is by a suitable interconnection of the following circuits:

1. An adder that adds the multiplicand to the partial sum if and only if the current multiplier digit is a 1.
2. A shifting circuit that shifts the partial sum one position to the right after each multiplier digit is sensed. (The same result could be obtained by shifting the multiplicand one position left instead.)
3. A shifting circuit that shifts the multiplier one position to the right after each step to put the next more significant digit in position to be sensed.

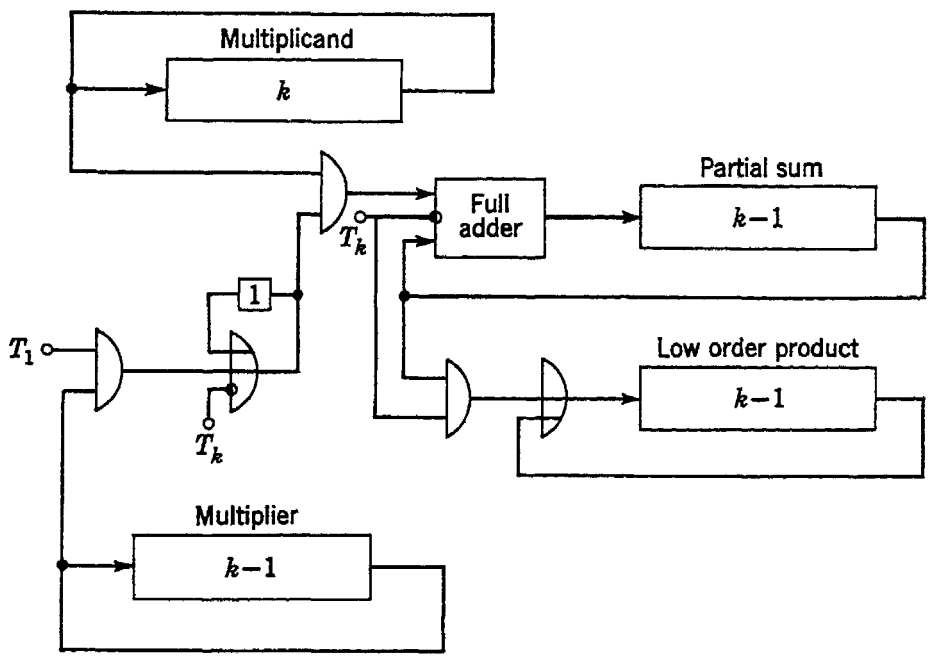



Fig. 9.34 A serial synchronous multiplier

Perhaps the simplest multiplier using these principles is the serial synchronous multiplier shown in Fig. 9.34. The multiplier digits are stored in a recirculating delay element having a total delay of $k - 1$ pulse periods, and the multiplicand is stored in a $k$-pulse delay element. $T_1$ and $T_k$ timing pulses, obtained from a pulse-distributor circuit, mark the beginning and the end of each group of $k$ pulses. The multiplicand and the multiplier are each assumed to be initially in such a position in the delay elements that, at the time of the first $T_1$, when the multiplication begins, the least-significant bit of each emerges from its delay element. If the multiplier digit is a 1, the flip-flop is set and emits a train of pulses until it is reset by a $T_k$ pulse. If the multiplier bit is a 0, the flip-flop is

not set. Thus, the flip-flop controls the adder input gate to admit the multiplicand to the adder or to block it, as determined by the current multiplier bit. The adder thus adds either the multiplicand or zero to the partial sum. A minor-cycle counter, not shown in the diagram, allows the operation to be repeated until each multiplier digit has been examined.

In this method of multiplying, the right shifts of the multiplier and of the partial sum are accomplished by storing these components in delay elements that are one pulse period shorter than the $k$ pulse periods assigned to each word. Thus in $k$ pulse periods the contents of such a loop circulate completely and move one digit position more. If the least-significant bit emerges from such a line at the first $T_1$, then the next more significant bit emerges in turn at each following $T_1$, thus effecting the right shift.

Using delay units $k - 1$ pulses long for words having a cycle length of $k$ pulses does not mean that one digit is left over when a word is inserted, since it is common practice in a serial computer to employ one or two blank spaces at the end of each word. Thus the actual word length is at most $k - 1$ pulses, and this full group can be stored in the multiplier loop. When the multiplicand is passed through the adder into the partial-sum loop, all its $k - 1$ digits are inserted, but the adder is inhibited by $T_k$, so that the least-significant bit emerging from the partial-sum delay line is prevented from reentering. However, $T_k$ also admits this digit to another $(k - 1)$-pulse-period delay element where the low-order digits of the product are stored. (A common practice is to insert the low-order digits, as they are formed, into the multiplier loop in place of the low-order bits that have already been sensed and hence can be discarded. Although this saves a loop, it also means that the multiplier is destroyed and must be read out of storage again if it is needed in the next step of the program.) After $k - 1$ full word cycles of $k$ pulses each, the $k - 1$ most significant bits of the product are stored in the partial-sum line and the $k - 1$ least-significant bits are in the low-order product line or in the multiplier line. The reader will find it helpful to trace the pulse patterns in the circuit for some simple case such as $k = 5$ and numbers of 4 binary digits.

It is also possible to build a parallel multiplier based upon the same digit-by-digit inspection of the multiplier digits. In such a system the addition of the multiplicand to the partial sum is done in parallel. Parallel computers often employ three registers in adding, one each for addend, augend, and sum. For multiplication one more register is needed for the multiplier. It may be assumed that the multiplicand is in the addend register and that the augend register is initially empty. As before, the first (least-significant) digit of the multiplier is sensed, perhaps by a right

shift to put it in a flip-flop which drives a gate that controls admission of the multiplicand to the adder. Then the addition process takes place, followed by a right shift and return of the partial-sum digits to the augend register. In this method also it is common practice after each step to insert the least-significant bit of the partial sum into the vacated position in the multiplier register.

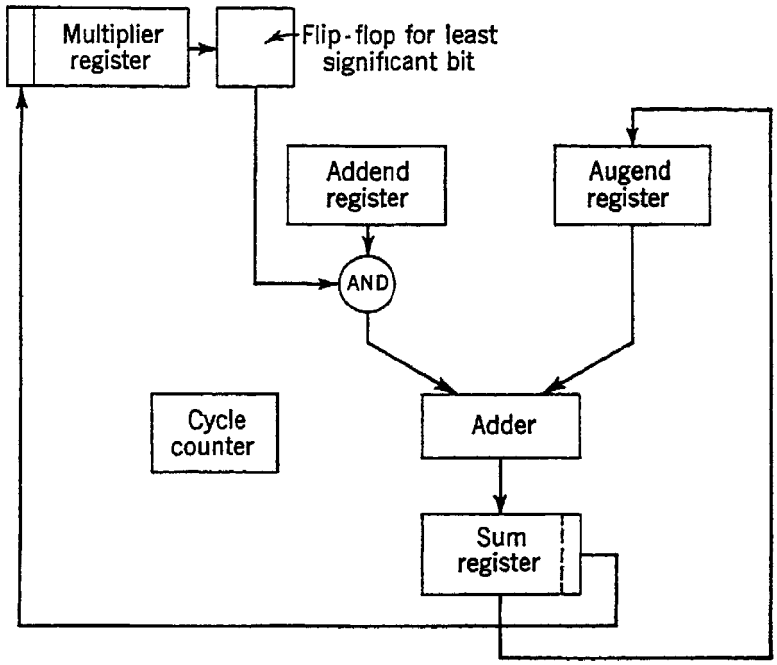



Fig. 9.35 A parallel multiplier

A block diagram for such a multiplying system is shown in Fig. 9.35. The following steps carry the system through the multiplication of two $k$-digit numbers:

1. The cycle counter and the augend register are cleared. The multiplier is inserted into its register and the multiplicand into the addend register.

2. The multiplier contents are shifted one position right, either the multiplicand or zero is added to the content of the augend register, and the sum appears in the sum register.

3. The sum register is shifted one position right, and the emerging digit is inserted at the left of the multiplier register.

4. The contents of the sum register are transferred without alteration of position to the augend register.

5. The cycle-counter content is augmented by 1 and compared with $k$. If $k$ is larger, step 2 and subsequent steps are repeated. If it is not, the multiplication has been completed.

Such a sequence of steps is typical of most of the more complex operations performed by a computer. It is usually possible to resolve complex

operations into a succession of shifts, additions, complementations, digit sensings, etc., and thus to prepare what has been called a "microprogram" for the computer. There has been much interest recently among computer engineers in the design of microprogrammed computers,[3,4] in which the microprograms are wired in on easily replaced matrix-connection panels; this allows simple alteration of the machine's order structure. As yet, however, the subject has been of academic rather than practical interest, and the sequencing of operations in most computers is carried out by various combinations of counters and control elements that are called into play by the operation-code digits of each instruction.

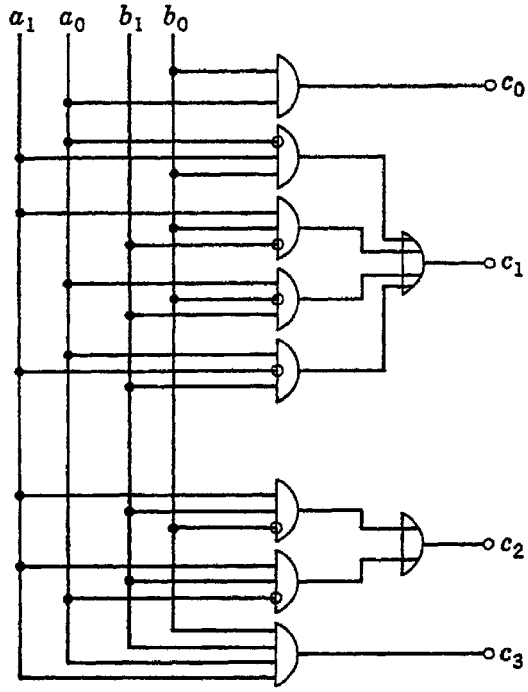


Fig. 9.36 A simultaneous binary multiplier for two-digit factors

The parallel circuit of Fig. 9.35 is evidently a much faster multiplier than the serial device of Fig. 9.34. However, it is not $k$ times as fast, because parallel addition of two $k$-bit numbers is not $k$ times as fast as serial addition, assuming the same type of adder circuit. The speed advantage of the parallel multiplier, like that of the parallel adder, can be enhanced if carry-completion sensing is employed. It is then unnecessary to allow time at each addition for carries to propagate the full length of the adder; only the time actually taken by the carries at each step need be allowed.

Even faster than the parallel multiplier is a class of circuits that might be called "simultaneous multipliers". In such circuits all the digits of the multiplicand and of the multiplier are applied simultaneously, and, after a delay for signal transmission or transient decay, all the digits of the product are presented simultaneously. A truly simultaneous multiplier is ordinarily not practical except for small numbers, because of the large amount of equipment it requires. However, with slight modifications, the technique can be very useful. If two $n$-digit numbers are to be multiplied, a Boolean expression can be written for each of the $2n$ digits of the product. These expressions become increasingly more complicated for the higher-order digits of the product, since the $j$th digit of the low-order half of the product is a function of $2j$ digits, and the $n$ high-order digits are functions of all $2n$ digits of the multiplier and the multiplicand. Each such function can be expressed in one of the canonical forms, for

instance, the disjunctive normal form, and can then be realized with a group of AND gates and a group of OR gates. As a simple example, consider a binary multiplier that accepts 2-bit multiplicands and multipliers and forms a 4-bit product. The product $c_3c_2c_1c_0$ can be expressed in terms of the factors $a_1a_0$ and $b_1b_0$ by the following relations:

$$\begin{aligned} c_0 &= a_0b_0 \\ c_1 &= \bar{a}_0a_1b_0 + a_1b_0\bar{b}_1 + a_0\bar{a}_1b_1 + a_0\bar{b}_0b_1 \\ c_2 &= a_1b_1\bar{b}_0 + a_1b_1\bar{a}_0 \\ c_3 &= a_1a_0b_1b_0 \end{aligned} \tag{9.32}$$

A circuit that implements these expressions is shown in Fig. 9.36.

## 9.8 Multiplication of complements[5–8]

The foregoing discussions of arithmetical operations have all been based upon the assumption that numbers are stored in the memory in the form of a sign and a magnitude; in that case negative numbers produced in complement form as the result of an operation are recomplemented before being placed in storage. In a perfectly random sequence of arithmetical operations, the probability is ½ that such a group will have to be recomplemented when it is called forth from storage to be used again. Since the same probability of recomplementation is involved if the number is stored in complement form, it is apparent that, on the average, the computer should perform its operations with the same speed whether numbers are stored as complements or as signed magnitudes. However, if all numbers are stored as signed magnitudes, it is clear that half of the recomplementations performed to put numbers into this form before storage are actually unnecessary. Therefore, storage of negative results in the complement form in which they are produced should result in a slight increase in computer speed. Storage in signed-magnitude form is probably more common than storage in complement form, however, since the speed benefit is not very great and the circuitry for execution of multiplication and division becomes somewhat more complex than for numbers in signed-magnitude form. Nevertheless, the procedures are of sufficient interest to justify consideration here.

First let us recall that, if an integer $x$ having $n$ nonsign digits is expressed in radix-complement form, the nonsign digits of its radix complement are given by $r^n - |x|$. The complete radix complement, including sign digit, is given by $r^{n+1} - |x|$. Similarly, the radix complement of an $n$-digit fraction $x$ is given by $r^0 - |x| = 1 - x$, and, if the sign digit is included, the group is given by $r - |x|$. The following examples illustrate these rules.

1. $x = -43$ (decimal)

   Nonsign digits of radix complement of $x$ are given by

   $$100 - 43 = 57$$

   Radix complement including sign: $1000 - 43 = 957$
   (The initial 9 is interpreted as a minus sign.)

2. $x = -0.43$ (decimal)

   Nonsign digits of radix complement of $x$ are given by

   $$1 - 0.43 = 0.57$$

   Radix complement including sign: $10 - 0.43 = 9.57$
   (The initial 9 is interpreted as a minus sign.)

3. $x = -101$ (binary)

   Nonsign digits of radix complement of $x$ are given by

   $$1000 - 101 = 011$$

   Radix complement including sign: $10000 - 101 = 1011$
   (The 1 at the left is interpreted as a minus sign.)

4. $x = -.101$ (binary)

   Nonsign digits of radix complement of $x$ are given by

   $$1 - .101 = .011$$

   Radix complement including sign: $10 - .101 = 1.011$
   (The 1 at the left is interpreted as a minus sign.)

In multiplying two $n$-digit integers, a result having $2n$ digits may be produced. In multiplying a positive number $x$ by a negative number $y$ in radix-complement form, we generate $x(r^{n+1} + y) = xr^{n+1} + xy$. The correct result, however, would be $r^{2n+1} + xy$. To get the correct result we must add $r^{2n+1} - xr^{n+1}$ to the result generated by direct multiplication of $x(r^{n+1} + y)$. Consider the following example:

$$\begin{array}{rrl}
-23 = y \rightarrow & 977 & \\
\times\ \ 37 = x \rightarrow & 037 & \\
\hline
161 & 6839 & \\
69 & 2931 & \\
\hline
-851 & 36{,}149 & \text{before correction}
\end{array}$$

$$\begin{aligned}
\text{Correction} &= r^{2n+1} - xr^{n+1} \\
&= 10^5 - 37 \times 10^3 \\
&= 63{,}000 \\
\text{Correct result} &= 63{,}000 + 36{,}149 = 99{,}149
\end{aligned}$$

The most significant digit of the five-digit group is the sign, and the four digits at the right are the complement of the nonsign digits. The group 99,149 is thus seen to be the radix complement of $-851$. Of course, machine execution of the process is carried out somewhat differently, since the double-length product is formed modulo $10^5$. It is sufficient, therefore, merely to subtract the quantity $xr^{n+1}$ from the uncorrected result. It is even possible, for a positive multiplier and a negative multiplicand, to insert the correction at each step, if, as each multiplier digit is multiplied by the $n + 1$ digits of the multiplicand, sign included, only the nonsign digits are preserved. Consider the following multiplication:

```
 − 23 →  977
× 37 →  037
    (a) 6839  ———7 × 977
    (b)  839  ———nonsign digits retained
    (c)  9839 ———negative partial product of (b) after right shift
    (d) 2931  ———3 × 977
    (e)  931  ———nonsign digits retained
    (f) 19149 ———sum of (c) and (e)
    (g)  9149 ———nonsign digits retained
    (h)  99149———result (g) after right shift
```

The correct result is generated without a final correction.

For the more general case in which the multiplier also may be negative, we may observe that the last step adds in the product of the multiplicand and the multiplier sign digit, which is $r - 1$. The correction subtracts $r$ times the multiplicand, so that these two steps may be combined into a single step in which the complement of the multiplicand is added if the multiplier sign is negative. For example,

```
     37 → 037
× −23 → 977
        (a) 259   ———7 × 037
        (b) 0259  ———nonsign digits retained and shifted right
        (c) 259   ———7 × 037
        (d) 2849  ———sum of (b) and (c)
        (e) 02849———result (d) after right shift
        (f) 963   ———complement of multiplicand
        (g) 99149———sum of (e) and (f)
```

```
   −37 →  963
× −23 →  977
         ----
    (a) 6741  ———7 × 963
    (b)  741  ———nonsign digits retained
    (c) 9741  ———negative partial product of (b) after right shift
    (d) 6741  ———7 × 963
    (e)  741  ———nonsign digits retained
    (f) 17151 ———sum of (c) and (e)
    (g)  7151 ———nonsign digits retained
    (h) 97151 ———negative partial product of (g) after right shift
    (i)  037  ———complement of multiplicand
    (j) 00851 ———sum of (h) and (i), with overflow dropped
```

The reader will find it informative to adapt these procedures to binary integers and fractions.

An interesting procedure, applicable to binary numbers, is due to A. D. Booth.[9] In this procedure the multiplier digits are examined two at a time, starting from the least-significant digit, which we shall call $x_0$, and proceeding toward the sign digit, which we shall call $x_n$. The rules are as follows for each digit $x_j$:

1. If $x_j = x_{j+1}$, shift the existing sum of partial products one place to the right.
2. If $x_j = 0$ and $x_{j+1} = 1$, subtract the multiplicand $y$ from the existing sum of partial products and shift the result one place to the right.
3. If $x_j = 1$ and $x_{j+1} = 0$, add the multiplicand $y$ to the existing sum of partial products and shift the result one place to the right.
4. Do not shift after the last step.

For example, let us multiply $-.101$ by $-.011$; that is,

$$\text{Multiplicand } y = -.101 = 1.0110 \text{ in two's complement form}$$
$$\text{Multiplier } x = -.011 = 1.1010 \text{ in two's complement form}$$
$$x = x_4x_3x_2x_1x_0$$

1. $x_0 = 0$ and $x_1 = 1$; $\therefore$ subtract $y$ from 0 (existing sum of partial products) to get 0.101. Shift right one place to get 0.0101, new sum of partial products.
2. $x_1 = 1$ and $x_2 = 0$; $\therefore$ add $y$ to 0.0101 to get 1.1011. Shift right one place to get 1.11011.
3. $x_2 = 0$ and $x_3 = 1$; $\therefore$ subtract $y$ from 1.11011 to get 0.01111. Shift right one place to get 0.001111.
4. $x_3 = x_4$. $\therefore$ neither add nor subtract. Do not shift, since this is last step. Result is 0.001111.

Proof of the validity of this method will make an interesting exercise for the reader.

## 9.9 Methods of binary division

The process of division is the inverse of the process of multiplication, and it is not surprising to find that division can be executed in the computer in a manner roughly inverse to that by which multiplication is performed. The usual method for computers having a built-in division operation consists of a succession of subtractions and shifts. The divisor is first subtracted from the dividend and then, after being shifted one binary position to the right, is subtracted from the remainder. For each subtraction that leaves a positive remainder, a 1 is entered as the quotient digit. If a negative remainder occurs, a 0 is entered and the divisor is added back in before the next shift and subtraction. In this manner the computer proceeds to generate as many quotient digits as required.

The following simple numerical example will help to illustrate the principles. Let it be required to divide 0.10101 by 0.11000 in a machine carrying 5 bits and sign in each number. In the first step we subtract the divisor, using two's complementation:

```
1.  0.10101     dividend
    1.01000     complement of divisor
    -------
    1.11101
    ↑
    └──── shows result is negative; ∴ first quotient digit is 0.
```

Since the result is negative, we add the divisor back in again and then shift the divisor right one position, complement it, and add the complement as follows:

```
2.  1.11101     result of step 1
    0.11000     divisor
    -------
    0.10101     dividend
    0.01100     divisor shifted one position right
    1.10100     complement of shifted divisor
    -------
    0.01001     sum of dividend and complement of shifted divisor
    ↑
    └──── shows result is positive; ∴ next quotient digit is 1.
3.  0.01001     result of step 2
    0.00110     divisor shifted another position right
    1.11010     complement of shifted divisor
    -------
    0.00011     sum of result of step 2 and complement of shifted divisor
    ↑
    └──── shows result is positive; ∴ next quotient digit is 1.
```

4. 0.00011 result of step 3
   0.00011 divisor shifted another position right
   1.11101 complement of shifted divisor
   0.00000 sum of result of step 3 and complement of shifted divisor
   ↑— shows result is positive; ∴ next quotient digit is 1.
5. 0.00000 result of step 4
   0.00001 divisor shifted another position right
   1.11111 complement of shifted divisor
   1.11111 sum of result of step 4 and complement of shifted divisor
   ↑— shows result is negative; ∴ next quotient digit is 0.

In this example the binary points of both dividend and divisor were taken to be in the same position, i.e., at the left. Thus the initial subtraction of divisor from dividend determined whether the result of the division would be a fraction or a number $\geq 1$. In computers in which all numbers are assumed to be fractions, a zero appearing in the leftmost position of the result of the first step is an indication of overflow.

The quotient obtained in the foregoing example is 0.1110. When carrying out the process by hand, we see that step 4 leaves no remainder and we know, therefore, that all succeeding quotient digits are 0. Although it is possible to design the computer to sense a zero remainder and terminate the division at that point, this facility is not usually provided since zero remainders occur so seldom, and the computer continues to calculate the terminal zeros one by one in the manner of step 5 of the example.

It is possible to shorten the process somewhat at each step that yields a negative result after a subtraction of the divisor. In such a case (according to the above procedure) we should add the divisor in again, shift it one position to the right, and subtract it again. Since the net result of these three steps is merely to add half the divisor, it is usually quicker to get the next digit by omitting these three steps and directly adding half the divisor. Thus, division can be performed by shifting the divisor one position right after each step and then either adding this shifted quantity to the dividend or subtracting it, according to whether the last previously determined digit was a 0 or a 1.

Division can also be executed by shifting the dividend (or the partial remainder) to the left at each step instead of shifting the divisor right. In this case the short cut of the preceding paragraph is implemented by either adding or subtracting the divisor from the partial remainder. We now repeat our previous example using this method:

```
1.  0.10101     dividend
    1.01000     complement of divisor
    -------
    1.11101
    ↑
    └── shows result is negative; ∴ first quotient digit is 0.
2.   1.11010    previous result shifted one place left
     0.11000    divisor (to be added)
    --------
    10.10010
     ↑
     └── shows result is positive; ∴ next quotient digit is 1.
3.   1.00100    previous result shifted one place left
     0.11000    divisor (to be subtracted)
     1.01000    complement of divisor
    --------
    10.01100
     ↑
     └── shows result is positive; ∴ next quotient digit is 1.
4.   0.11000    previous result shifted one place left
     0.11000    divisor (to be subtracted)
     1.01000    complement of divisor
    --------
    10.00000
     ↑
     └── shows result is positive; ∴ next quotient digit is 1.
```

In machines in which all numbers are to be regarded as fractions, it is necessary for the programmer to arrange that the divisor exceed the dividend in magnitude in order to make the result of a division be a fraction. As we have seen, the result of the first step can constitute an indication of whether the divisor exceeds the dividend or an overflow will occur. On the other hand, if all numbers in a machine are to be regarded as integers or mixed numbers, it is necessary that the dividend exceed the divisor in order for the quotient to have an integral part. Division in integer machines without restriction upon relative sizes of dividend and divisor involves a further complication, however, since it is desirable to shift the divisor to the left until its most significant 1 is in the same digit position as the most significant 1 of the dividend before starting the subtract-and-shift sequence. The following example, in which 60 is divided by 5, makes the process clearer.

$$(60/5)_{\text{decimal}} = (111100/101)_{\text{binary}} = 1100$$

Division without first shifting divisor:

```
1.        111100
           -101
         --------
         110111
```

```
2.        110111
           -101
          ------
          110010
. . .       . . .
12.       000101
           -101
          ------
          000000
```

Division with divisor shifted left:

```
   111100
  -101000
  -------
    10100 → first digit = 1
   -10100
   ------
    00000 → second digit = 1
   - 1010
  -------
 Overdraft → third digit = 0
      0000
    - 101
  -------
 Overdraft → fourth digit = 0
```

In the first case, in which the successive subtractions are performed without first shifting the divisor left, the total number of these subtractions executed before the result goes negative is the integer part of the quotient. This process, of course, is a highly inefficient way of doing division, and the second process shown is much better. Three left shifts have aligned the first 1 of the divisor with the first 1 of the dividend, and the sign of the difference now determines the first quotient digit. This first quotient digit lies to the left of the radix point a number of positions 1 greater than the number of shifts required to align the first 1's of divisor and dividend. It can be seen that the number of subtractions involved in the second method is of the order of the base 2 logarithm of the number of subtractions required by the first method. Some machines have a built-in facility for automatically aligning the divisor and dividend while keeping track of the shifts. This facility is a normal part of machines having built-in floating-point arithmetic. In the usual fixed-point machine, however, it is the programmer's responsibility to control the relative sizes of dividend and divisor so as to avoid overflow. The following procedure is usually used:

1. Dividend and divisor are assumed to have their binary points in the same position (whether they actually do or not). If the divisor then

exceeds the dividend in absolute value, division proceeds. The binary point of the quotient is at the left if the binary points of dividend and divisor were actually in the same position. If they were not, then the binary point of the quotient lies to the right by $k - j$ positions, where $k$ and $j$ are the actual binary-point positions of divisor and dividend, respectively, measured from the right of the $n$-bit group.

2. If the divisor does not exceed the dividend in absolute value, with points assumed in the same position, then an overflow signal is activated, since, as we have seen, this condition can lead to the necessity for repeated subtractions of the divisor from the same dividend position, and must be corrected by left shift of the divisor first.

To facilitate the execution of both multiplication and division and to provide added flexibility in the matter of binary-point location, many machines employ a double-length register to hold the $2n$ bits of a product or of a dividend. The location of the binary point is entirely under the programmer's control. Thus, if two fractions each having $n$ bits are multiplied together, the result is a $2n$-bit number with the binary point at the left. If the factors are mixed numbers, the binary point in the product is displaced to the right by a number of positions equal to the sum of the number of positions by which the binary points in each of the factors are displaced to the right. That is, if two 10-bit numbers, each having its binary point at the left, are multiplied together, the result is a 20-bit number also having its point at the left. If one factor has 3 bits to the left of the point and 7 to the right and the other has 4 bits to the left of the point and 6 to the right, the 20-bit product is interpreted as having 7 bits to the left of its binary point and 13 to its right. If the two factors are integers, then the binary point of the product lies at the extreme right. Thus the binary-point location is not restricted by the machine structure.

If a double-length register is used for division, a $2n$-bit dividend can be inserted into it, and the divisor is subtracted initially from its $n$ high-order positions. As we have seen, in division as well as in multiplication, the binary-point location is under the programmer's control.

The reader may by now have drawn the obvious inference that the distinction between "integer" machines and "fractional" machines is largely artificial and depends more upon manner of use than upon internal structure. About the only structural difference is in the matter of whether an addition adds a number into the low-order half of the double-length register or into the high-order half. If numbers are added into the low-order half, they are in effect being treated as integers; if they are added into the high-order half, they are being treated as fractions. However, by appropriate shifting of the contents of the double-length register, a number can be added into any section of the double-length number.

Thus it is possible, in the final analysis, to program any computer as an integer machine, a fraction machine, or a mixed-number machine.

## 9.10 Square root

The facility for taking square roots is not ordinarily provided as a built-in operation in the general-purpose computer, since the necessity for taking the square root arises relatively infrequently in most problems. The extra apparatus required for automatic square-rooting is not usually justifiable, and the time lost in performing a programmed square root by means of a subroutine is usually inconsequential. However, for some computer applications a built-in square-root operation is useful, and the procedure described in the following paragraphs may be employed.

This procedure, which applies to binary numbers, is a direct adaptation of the conventional "long-hand" method of square-rooting. Three registers are used: (1) a double-length accumulator, which initially contains the number whose square root is to be found, (2) an $n$-digit nonshifting register to contain the $n$ digits of the root, and (3) an $n$-digit shifting register initially containing a 1 in its most significant position. The content of the accumulator after the $j$th step will be designated $x_j$. The content of the nonshifting register after the $j$th step will be designated $A_j$, where $A_j$ is regarded as a fraction and is given by

$$A_j = 0 \,.\, a_0 a_1 \cdots a_j \tag{9.33}$$

The successive accumulator contents $x_{j+1}$ are found from the recursion relation*

$$x_{j+1} = 2x_j - a_{j+1}(2A_j + 2^{-(j+1)}a_{j+1}) \tag{9.34}$$

Each digit $a_{j+1}$ is either a 0 or a 1. To determine which is correct, at each step the trial subtraction

$$2x_j - (2A_j + 2^{-(j+1)}) \tag{9.35}$$

is performed. If the result is negative, then $x_{j+1} = 2x_j$, and the digit $a_{j+1}$ of $A_{j+1}$ is 0. If the result is positive, then $x_{j+1}$ is the result of the trial subtraction, and the digit $a_{j+1} = 1$.

The following example shows the step-by-step contents of the three registers in taking the square root of the 8-bit group 0.01111001, the binary equivalent of $^{121}/_{256}$.

* In terms of radix $r$, this relation is

$$x_{j+1} = rx_j - a_{j+1}(2A_j + r^{-(j+1)}a_{j+1})$$

| | Accumulator | $A_j$ register | $2^{-(j+1)}$ register |
|---|---|---|---|
| 1. | $x_0 = 0.01111001$ | $A_0 = 0.0000$ | $2^{-1} = 0.1000$ |
| | $2x_0 = 0.11110010$ | | |
| | $2x_0 - (2A_1 + 2^{-1}) = 0.01110010$ | | |
| | Positive; | $\therefore A_1 = 0.\underline{1}000$ | |
| 2. | $x_1 = 0.01110010$ | $0.\underline{1}000$ | $2^{-2} = 0.0100$ |
| | $2x_1 = 0.11100100$ | | |
| | $2x_1 - (2A_1 + 2^{-2}) = -0.01011100$ | | |
| | Negative; | $\therefore A_2 = 0.\underline{1}000$ | |
| 3. | $x_2 = 0.11100100$ | $0.\underline{1}000$ | $2^{-3} = 0.0010$ |
| | $2x_2 = 1.11001000$ | | |
| | $2x_2 - (2A_2 + 2^{-3}) = 0.10101000$ | | |
| | Positive; | $\therefore A_3 = 0.\underline{1}010$ | |
| 4. | $x_3 = 0.10101000$ | $0.\underline{1}010$ | $2^{-4} = 0.0001$ |
| | $2x_3 = 1.01010000$ | | |
| | $2x_3 - (2A_3 + 2^{-4}) = 0.00000000$ | | |
| | Positive; | $\therefore A_4 = 0.\underline{1011}$ | |

The foregoing procedure can also be adapted for use with decimal computers. In this case it is more convenient to form the digit $a_{j+1}$ by taking it to be the number of subtractions of successive odd integers that can be performed without overdrawing the partial result. This method is based on the fact that the sums of the successive odd integers are equal to the squares of the successive integers. Thus,

$$\begin{aligned} 1 + 3 &= 2^2 \\ 1 + 3 + 5 &= 3^2 \\ 1 + 3 + 5 + 7 &= 4^2 \\ 1 + 3 + 5 + 7 + 9 &= 5^2 \\ &\cdots\cdots\cdots \end{aligned} \tag{9.36}$$

As an example, we may take the square root of 625 by first subtracting odd integers from the leftmost two-digit group as follows:

$$\begin{array}{r} 06\ 25 \\ -1\phantom{\ 25} \\ \hline 05\ 25 \\ -3\phantom{\ 25} \\ \hline 02\ 25 \end{array}$$

Since two subtractions are all that can be performed without yielding a negative result, we take 2 as the most significant digit of the result. We double this and shift it one position left, as in the long-hand method, to get 40. Our next trial subtrahend is 41, followed by 43, and so on.

These steps are as follows:

$$
\begin{array}{r}
0225 \\
-41 \\
\hline
0184 \\
-43 \\
\hline
0141 \\
-45 \\
\hline
0096 \\
-47 \\
\hline
0049 \\
-49 \\
\hline
0000
\end{array}
$$

The five subtractions performed before overdrawing indicate that the next digit is 5. Since the last partial result was precisely zero, all further digits of the square root are zero, and our answer is, not unexpectedly, 25.

## 9.11 Round-off methods[10–12]

In the course of the various arithmetical manipulations of numbers within a computer, it frequently becomes necessary to discard some of the less significant digits of a number. If at each time this is done the discarded digits are merely ignored without any account being taken of their magnitude, then the digits retained always represent numbers smaller in magnitude than the original numbers. In the course of a long calculation the resulting errors, which are all of the same sign, would tend to accumulate rapidly and reduce the accuracy of the result. It is customary, therefore, to employ some form of round-off technique that modifies the retained digits in a manner which is determined by the discarded digits and which gives smaller errors than would be produced by merely disregarding them.

The necessity for dropping digits at the right of a number can arise in such situations as the following:

1. In addition, when two numbers to be added have a sum that exceeds machine capacity, each must first be shifted one position to the right, and the rightmost digit of each is dropped.

2. In multiplication of two numbers of $n$ bits each, a product of length $2n$ bits is produced. Although it is sometimes desirable to retain all $2n$ bits, frequently only $n$ are needed, and the $n$ least significant are discarded.

3. In division of two numbers of $n$ bits each, a quotient of infinite length may result. Only the first $n$ bits (or sometimes $2n$) are retained, and the less significant digits are discarded.

If digits are discarded with no round-off of retained digits, the retained digits always form a number of smaller magnitude than the original number (unless all discarded digits are 0's). The mean error is thus seen to be $-1/2^{n+1}$, where $n$ is the number of digits retained and numbers are assumed to be fractions. If we apply statistical analysis, we find that the variance is $1/(3 \times 2^{2n})$. By the use of round-off, both the mean error and the variance can be reduced. The following are the two most frequently used round-off techniques:

1. The last digit retained is made to be a 1, regardless of the value of the digits discarded.

2. A 1 is added to the $(n + 1)$st position of the original number, and the first $n$ bits of the result are retained.

Either of these procedures has a mean error of zero, but the variance of the first is $1/(3 \times 2^{2n})$, and the variance of the second is $1/(12 \times 2^{2n})$, since the maximum possible error of the first is twice as great as that of the second. The first method, of course, is much simpler to implement, since it requires an operation upon only a single digit position, whereas the second method requires an addition operation in the $(n + 1)$st position that may result in a carry, which in an extreme case could be propagated through all $n$ positions. The necessity for performing a complete addition in the second method means that the round-off must be executed in circuits capable of addition. This can complicate machine structure, if the circuits that produce numbers to be rounded off are required also to perform the rounding; an alternative is to slow machine operation, and to transfer such numbers to an adder for rounding. Although method 2 does require more complex circuits, it is more widely used than the method of forcing a final 1, because of its smaller variance.

In any case, whether numbers are rounded by adjusting the final digit or merely by discarding digits without compensation, the digits retained constitute an approximation to some more precise number. Continued arithmetical operations and further round-offs performed upon such approximate numbers generate other approximate numbers, and in general the errors caused by round-off tend to grow. In the early days of the large high-speed computer, it was feared that round-off errors would accumulate so rapidly as to destroy the validity of most long computations. Although these early fears have not been borne out in most long problems, the computer user must nevertheless be on his guard with each new problem lest some unsuspected statistical variation in the numbers of his problem give rise to a rapidly accumulating round-off error. One way of protecting against round-off error accumulation is to carry many more digits in each number than are required for the precision of the result. This, of course, is very inefficient. By the same token, however, machines with short word lengths are likely to produce large round-off errors at an

early point in their operation, and it may be necessary to resort to double-precision operation as a countermeasure.

## 9.12 Description of the logic of the MIC

Any structure as large and complicated as a digital computer can be assembled in a great variety of ways to meet the various design requirements that may be imposed. It is not possible to discuss many of the designs or even to suggest more than a few of them. Nevertheless, although no computer can be considered typical in all respects, most engineers approaching the subject of computer structure for the first time find it helpful to analyze in some detail the structure of at least one computer, if only to satisfy themselves that computers are really possible. The computer to be discussed here is the MIC, the University of Michigan Instructional Computer, which we have already encountered in Chap. 6 in connection with programming. The MIC, although not a large machine, is rather versatile and contains structural elements found in one form or another in most large computers.

**MIC characteristics and organization.** The essential characteristics of the MIC are listed in Table 9.9.

**Table 9.9**
**MIC Characteristics**

| | |
|---|---|
| Logic | Serial, binary |
| Number size | 30 bits plus sign |
| Instruction type | Single-address; two instructions per word |
| Memory | 1,024 words of magnetic drum storage |
| Circuitry | Dynamic, with 1-megacycle clock frequency |
| Command list | 16 possible arithmetical and logical operations* |

* For list, see Chap. 6.

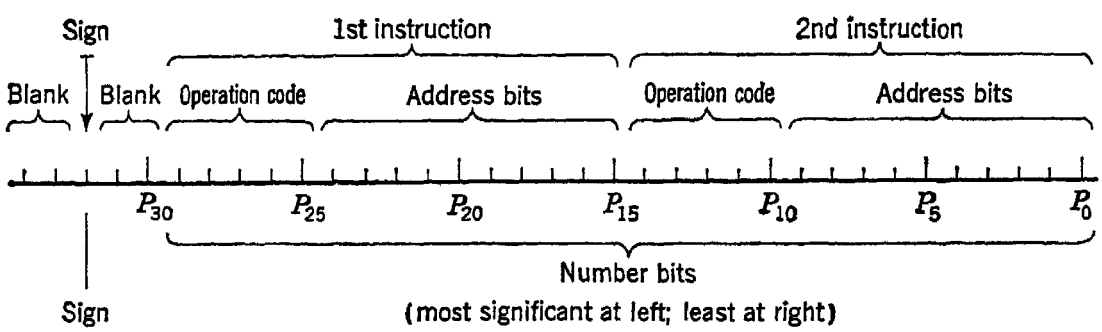



Fig. 9.37 MIC word structure

The word structure of the MIC, for numbers and for instructions, is shown in Fig. 9.37. Although information is typed into the machine starting from the left end of the words shown in Fig. 9.37, all other transfers of words within the machine take the rightmost bits first and the

leftmost bits last. In all transfers of information, the word to be transferred is inserted first into a shift register and thence to its ultimate destination. Thus the basic organization of the machine may be represented as shown in the block diagram of Fig. 9.38. The dashed lines are control lines, and the heavy lines are paths for word transmission.

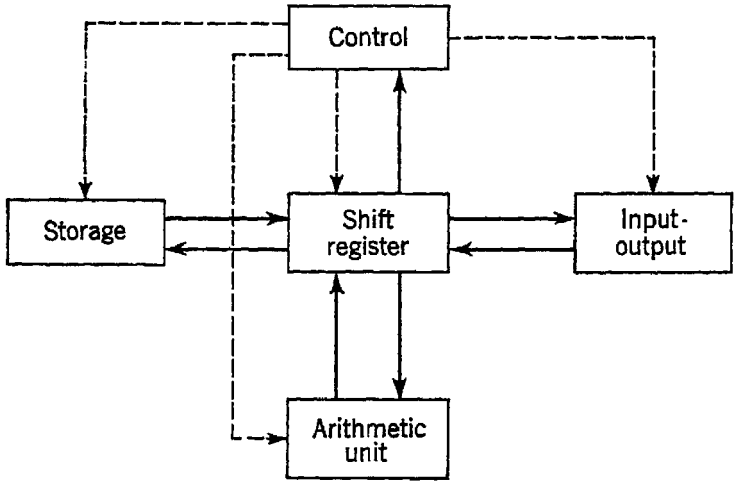



Fig. 9.38 MIC organization

The operation of the system is controlled by four timing phases, some fixed in length and some variable. The phases and their functions are as follows:

Phase 1. The instruction word in the shift register is transferred to the control unit. This phase has a duration of 1 minor cycle, that is, 35 clock pulse periods.

Phase 2.0. The cell specified by the address in the first instruction in the control unit is located in storage, and its contents are inserted into the shift register or the shift register contents are inserted into the cell, depending upon the nature of the operation. With the magnetic drum memory, this may require many minor cycles.

Phase 3.0. The operation specified by the first instruction in the control unit is executed. This may require up to 32 minor cycles.

Phase 2.1. Same as phase 2.0, but uses address specified by second instruction.

Phase 3.1. The operation specified by the second instruction in the control unit is executed.

Phase 4. The next instruction is acquired from storage and inserted in the shift register.

**Control unit.** The major parts of the control unit are shown in Fig. 9.39. As the instruction word enters the control unit during phase 1, the address digits of the second instruction enter first, followed by the operation-code digits of the second instruction and then by the digits of the first instruction. At the end of one minor cycle, the second instruc-

tion is circulating in the 15-bit loop, while the digits of the first instruction are "staticized", that is, held in fixed positions, in the flip-flops that constitute the operation-code register and the address register. At the end of phase 3.0, the second instruction is transferred from the recirculating loop to the decoding registers, where it in turn is processed. The phase control produces the timing signals to control these transfers by means of the circuit of Fig. 9.40. This circuit consists essentially of four flip-flops,

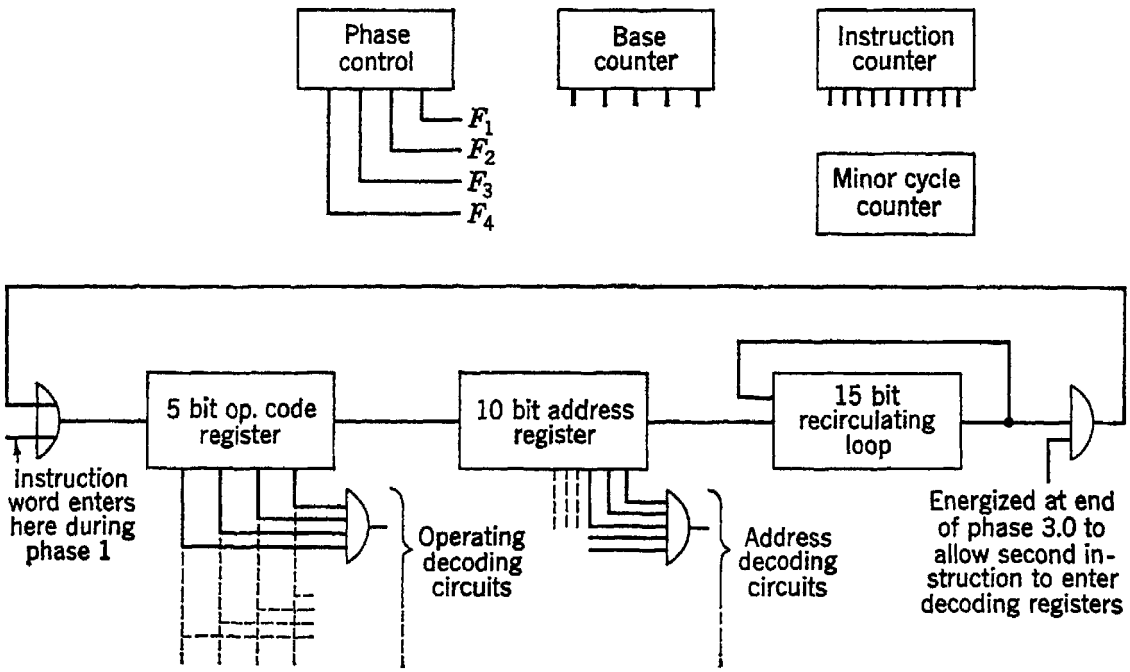



Fig. 9.39 Major components of the MIC control unit

only one of which is in the ON state at any time. The phase-control pulses applied to this circuit form a continuous stream to keep the ON stage turned on. Each interruption of this pulse stream turns off the stage that was on and turns on the next stage in the chain. Thus phase 1 ($F_1$) is followed by $F_2$, $F_3$, $F_4$, $F_1$, etc. However, since a full cycle of operation repeats $F_2$ and $F_3$ (we have already spoken of these as "phase 2.0" and "phase 3.0", the first phase 2 and the first phase 3, and "phase 2.1" and "phase 3.1", the second phase 2 and the second phase 3), this basic chain of four flip-flops must be augmented by the "two counter" shown below the chain. This counter is turned on by the $F_1$ signal as the machine enters phase 1. It remains on through phase 2 and through phase 3 and for one clock pulse beyond the end of phase 3. Thus, when the stream of phase-control pulses is interrupted for one pulse time to mark the end of phase 3, a single pulse from the two counter is admitted to the $F_2$ stage to turn it on again. The two counter is now turned off by the simultaneous inhibitions effected by $F_2$ and by the last $F_3$ pulse delayed one pulse period. Thus, after the next $F_3$, flip-flop $F_4$ is acti-

vated. The reader will find it helpful to lay out a timing diagram showing pulse sequences at the various points in the circuit.

When the machine is first turned on, the POWER ON wire admits a single pulse to the $F_3$ flip-flop to turn it on. Computation starts when the START button is pressed and the circuit in the lower part of Fig. 9.40

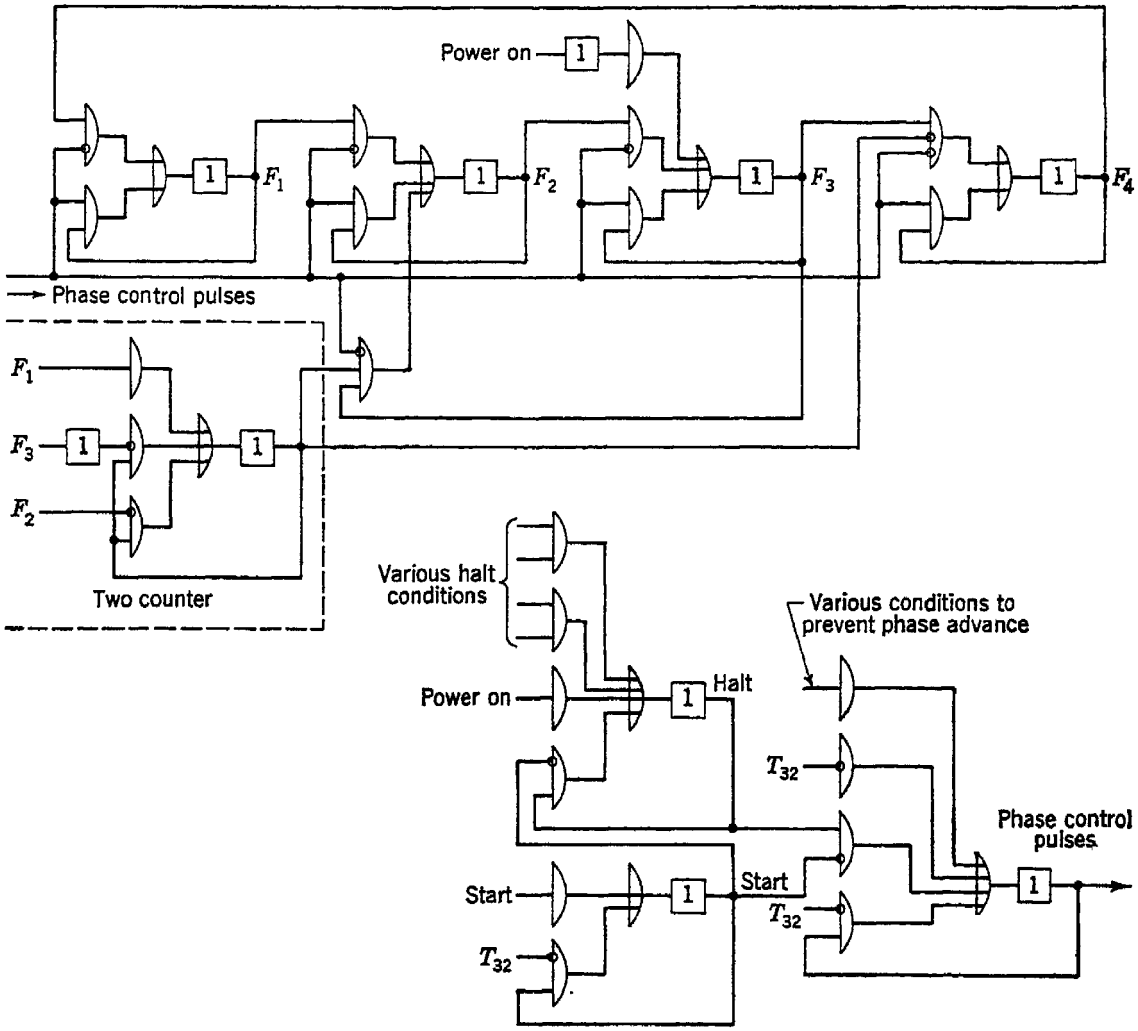



Fig. 9.40 Phase-control circuit

allows the machine to enter phase 4. This circuit produces a continuous stream of phase-control pulses when power is turned on, since the HALT flip-flop is on. The START switch turns on the START flip-flop, which then turns off the HALT flip-flop. When the next $T_{32}$ timing pulse is applied, the phase-control flip-flop is turned off for one pulse time, allowing the phase to advance to $F_4$. Thereafter, the phase will advance following each $T_{32}$ pulse, unless the computer is in a state in which the phase should not be advanced. For example, while the machine is locating a memory cell, several minor cycles may elapse before the cell has been located and the next phase can commence. A similar situation occurs during such arithmetic operations as multiplication or division,

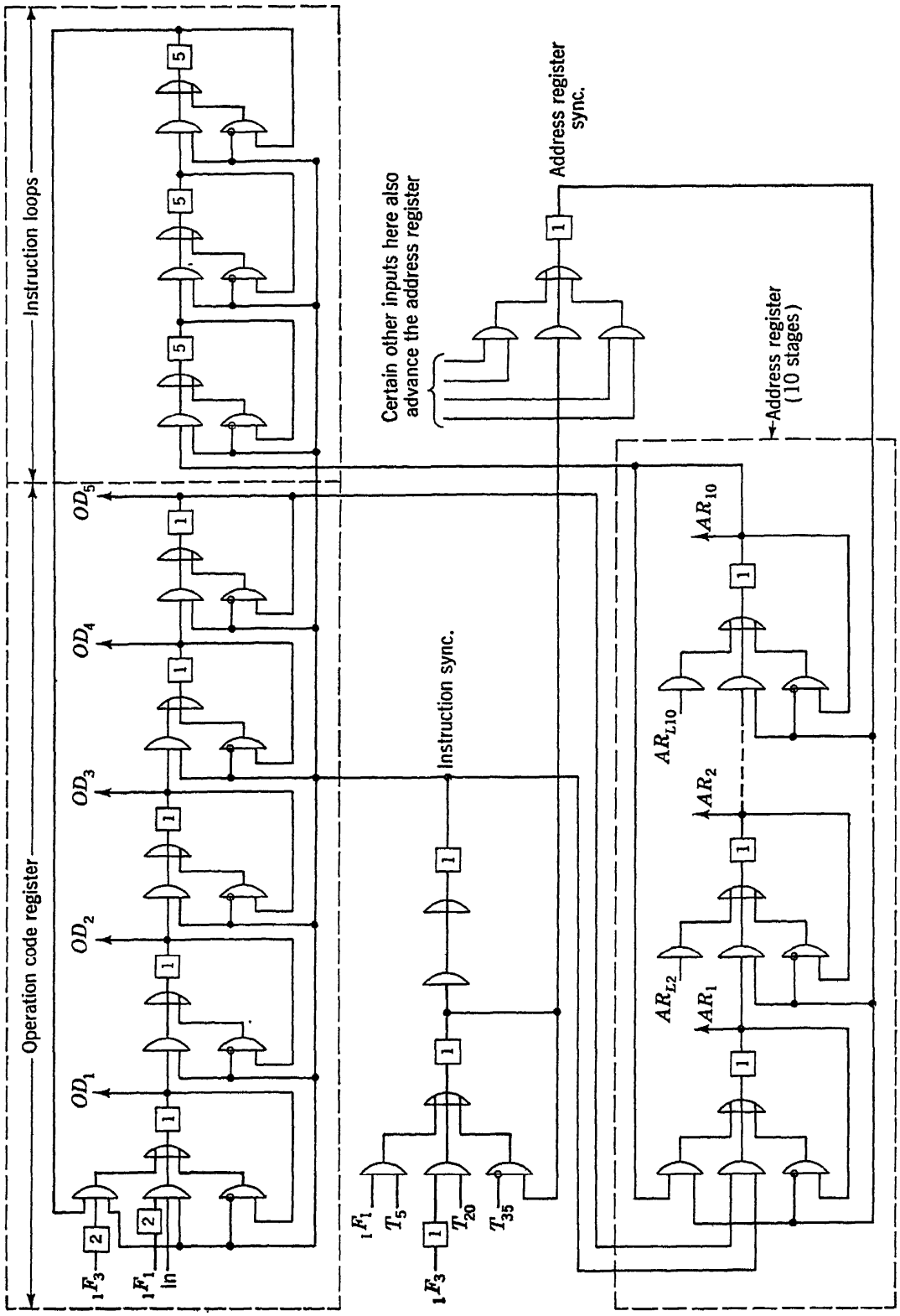



Fig. 9.41 Instruction register

which also require several minor cycles. Any such situation generates a signal that is applied to the phase-control flip-flop to keep it in the ON state and thus prevent phase advance.

**Instruction storage.** During phase 1, an instruction word is obtained from storage and placed in the instruction-storage circuits. These circuits, which were shown in block-diagram form in Fig. 9.39, are shown in more detail in Fig. 9.41. The INSTRUCTION SYNC circuit at center left produces a 30-pulse train during phase 1 or a 15-pulse train during the first minor cycle of phase 3. (This cycle is denoted by ${}_1F_3$ on the diagram.) Information entering the left end of the operation-code register during phase 1 is gated by the 30-pulse train through this register, through the 10-stage address register, and into the three successive 5-bit loops at the upper right. Thus, at the end of phase 1, the right instruction of the word transferred from the shift register is stored in the three 5-bit loops, the address digits of the left instruction are in the address register, and the operation-code digits are in the operation-code register. After execution of the left instruction during phases 2 and 3, the INSTRUCTION SYNC generator produces a 15-pulse train which advances the right instruction from the three 5-bit loops into the address register and the operation-code register, where it is then ready for execution during the second phases 2 and 3. (The left instruction is advanced into the three loops by the 15-pulse train, but is not used again.)

**Operation-decoding circuits.** During phase 3, the operation specified by 4 of the 5 bits of the operation code is executed. The fifth bit, it will be recalled from Chap. 6, indicates whether the address bits are absolute or relative to the base counter and, hence, is not involved in determining the operation. The four leftmost positions of the operation-code register, designated $OD_1$, $OD_2$, $OD_3$, and $OD_4$, drive 16 flip-flops, one for each of the machine's 16 commands. Some of these decoding circuits are shown in Fig. 9.42. The ADD circuit, for example, is turned on at $T_{35}$ of the first minor cycle of phase 3 (${}_1F_3$) if flip-flops $OD_1$, $OD_2$, $OD_3$, and $OD_4$ are all set to the zero state. It remains on until $T_{35}$ of the last minor cycle of phase 3, at which time both the lower and the upper AND gates are inhibited and the pulse cannot continue to circulate. The SUBTRACT flip-flop and the MULTIPLY flip-flop (as well as the DIVIDE flip-flop, which is not shown) are also energized from $T_{35}$ of the first minor cycle of phase 3 until $T_{35}$ of the last minor cycle of phase 3. The SHIFT flip-flop is turned on at $T_2$ of phase 2 and is turned off at $T_{35}$ of the last minor cycle of phase 3. The SHIFT flip-flop is turned on in phase 2 instead of in phase 3, since no memory access is required in shifting and execution of the operation can, therefore, start earlier. (It will be recalled that, in the MIC SHIFT instruction, the address bits form the shift constant rather than an address.) If a right shift is to be performed, the fifth bit in the address

register is a 1; therefore, the SHIFT RIGHT flip-flop will be turned on (it is also turned on when the MULTIPLY flip-flop is on). When the fifth bit in the address register is a 0, indicating a left shift, the SHIFT LEFT flip-flop is turned on by the SHIFT flip-flop.

The rest of the 16 decoding circuits are not shown, since they do not differ in principle from those of Fig. 9.42. In each case the decoding

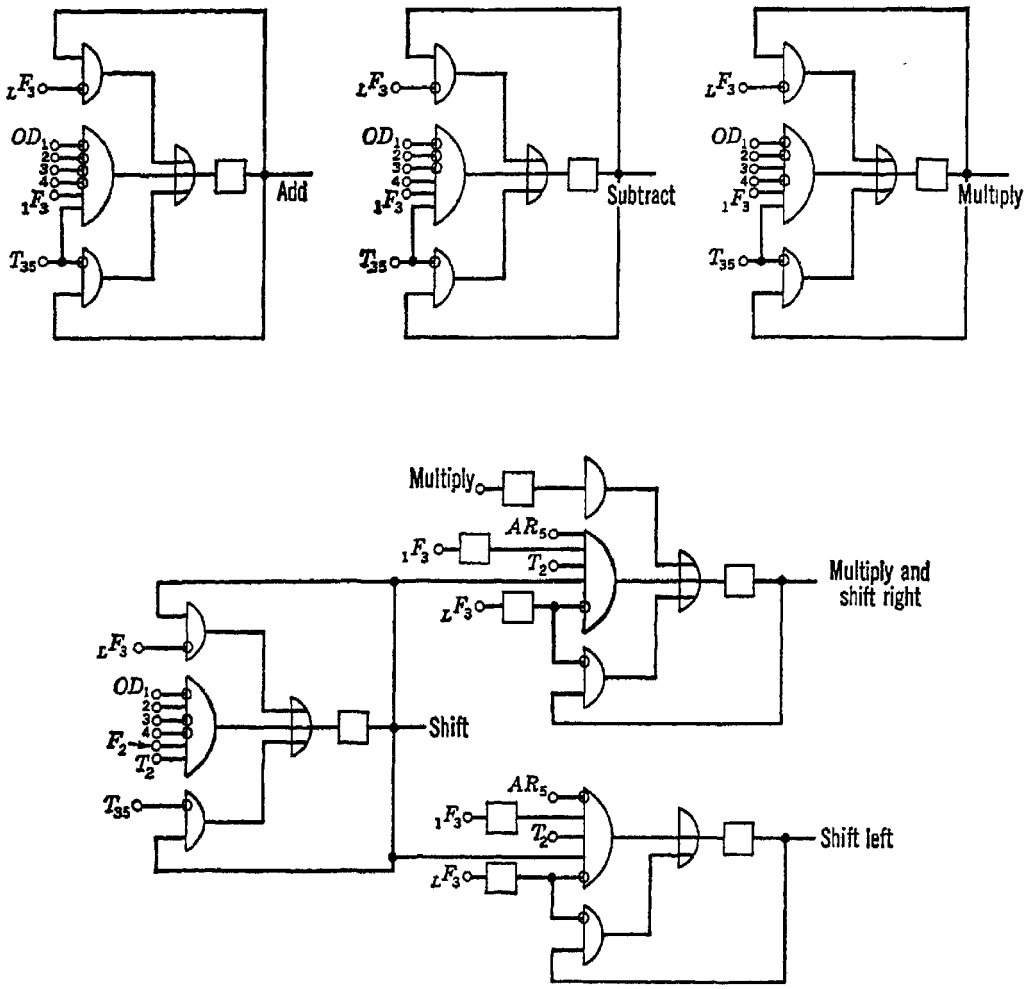



Fig. 9.42 Typical operation-decoding circuits

circuit produces a train of pulses in response to a particular set of operation-code digits, and this train in turn controls other elements of the computer in such a way as to execute the basic steps needed to carry out the desired operation.

**Locating a cell in memory.** Communication with the computer memory often involves difficult problems of synchronization. Even with noncyclic memories, such as the magnetic-core memory, there are only certain times at which access to the memory is possible, and the computer may have to wait for several microseconds until access can be obtained. These problems are even more severe in computers with cyclic memories (e.g., the magnetic drum), since the waiting time may be long and the

data rate to and from the memory may be asynchronous with respect to data in the computer proper. The main memory of the MIC is a magnetic drum, and, to help solve the timing problems, several timing tracks are used in addition to the address and information tracks. A portion of the drum tracks is shown symbolically in Fig. 9.43. The principal timing track is the drum digit-synchronizing track, on which pulses are recorded with uniform spacing around the entire circumference. Pulses on all other drum tracks are in synchronism with these drum digit-synchronizing pulses. Since each word is 31 bits long and 32 words are stored

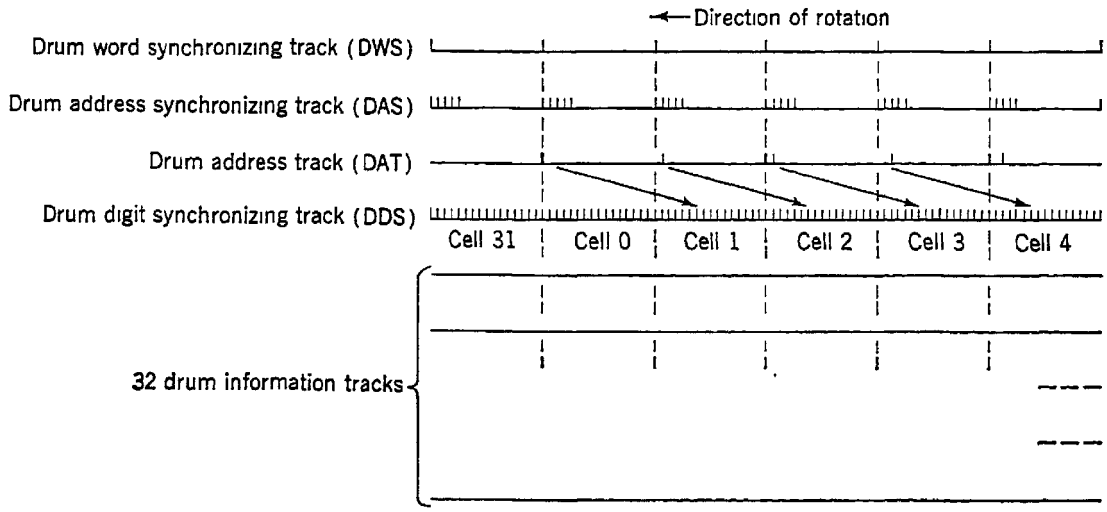



Fig. 9.43 Drum layout

circumferentially in each track, a total of $31 \times 32 = 992$ drum digit-synchronizing pulses are present. The drum word-synchronizing track contains a single pulse for each 31 on the drum digit-synchronizing track to mark the beginning of each word. The drum address-synchronizing track contains groups of five pulses at the beginning of each word interval. These pulses time the read-outs from the drum address track, which contains groups of binary digits corresponding to track positions 0 to 31. (It will be noted that the cell specified by each group of drum-address digits is the cell following the address group and not the cell concurrent with the address group. This is so because the current cell has gone partly by the read-out point before the address has been fully read. The address is, therefore, made to refer to the next complete cell.)

The logic circuitry for the drum appears in Fig. 9.44. It includes an address comparator (AC), which compares the digit group being read from the drum address track with the digit group stored in positions 6 to 10 of the address register. Any disagreement between these digit groups puts flip-flop AC in the ON state and inhibits the setting of the FAVORABLE COMPARISON flip-flop (FCFF). However, if the digit groups agree, FCFF is turned on by the next drum word-synchronizing pulse. It remains on for the duration of one drum word time (approximately 1 msec) and is

turned off by the next drum word-synchronizing pulse, which also turns on the TRANSFER END flip-flop (TEFF), marking completion of the transfer of a word to or from the drum. TEFF is turned off at the next $T_{32}$.

MAP is the "memory access package". This circuit forms a flip-flop that is turned on at $T_3$ of the first minor cycle of phase 4 ($_1F_4$) or of phase 2 ($_1F_2$), except during the operations SHIFT, SET INSTRUCTION COUNTER, SET BASE COUNTER, or BASE. It is turned off by the first minor

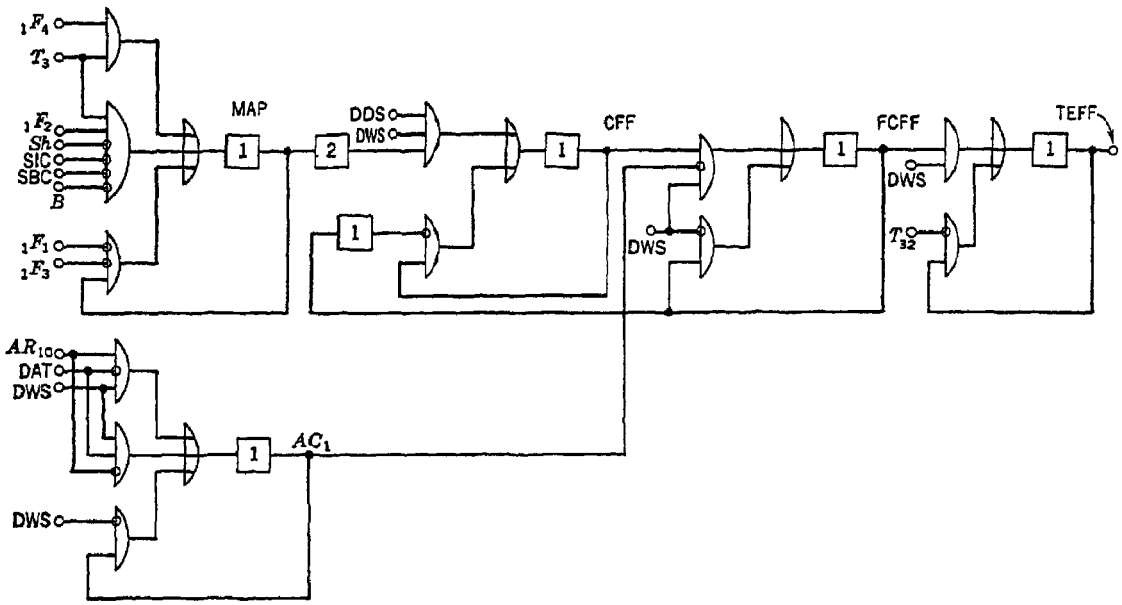



Fig. 9.44 Logic circuits for locating a drum-storage position

cycle of phase 1 or of phase 3 ($_1F_1$ or $_1F_3$). The output from this circuit turns on the COMPARISON flip-flop (CFF) when a drum word-synchronizing pulse and a drum digit-synchronizing pulse are simultaneously received. The signal from CFF and a drum word-synchronizing pulse together turn on FCFF when the desired cell has been found on the drum.

While these actions are taking place, positions 6 to 10 of the address register are being circulated by being shifted one position for each drum address-synchronizing pulse. Thus the $AR_{10}$ input to the address comparator receives one at a time the digits of the address to be found, and these are then compared with the digits being read from the drum address track. The 5 bits stored in address register positions 1 to 5 ($AR_1$ to $AR_5$) are in the meantime controlling the setting of relays to choose one of the 31 tracks on the drum. (Only 31 tracks are used on the drum, since 32 of the possible 1,024 addresses are reserved for input and output equipment.)

When the desired address has been located and FCFF is turned on, transfer to or from the drum can commence. In each case the content of the shift register is shifted one position right for each drum digit-synchronizing pulse. If the operation involves transfer from the drum,

each drum information digit, synchronized by a drum digit-synchronizing pulse, is admitted in turn at the left end of the shift register ($SR_0$); if the transfer is to the drum, successive digits from the right end of the shift register ($SR_{30}$) are gated to the drum. The shift register and its associated circuits are not shown here, since their manner of operation is relatively straightforward.

**Execution of operations.** We have already mentioned that operations are executed during phase 3 of the instruction cycle. Since some

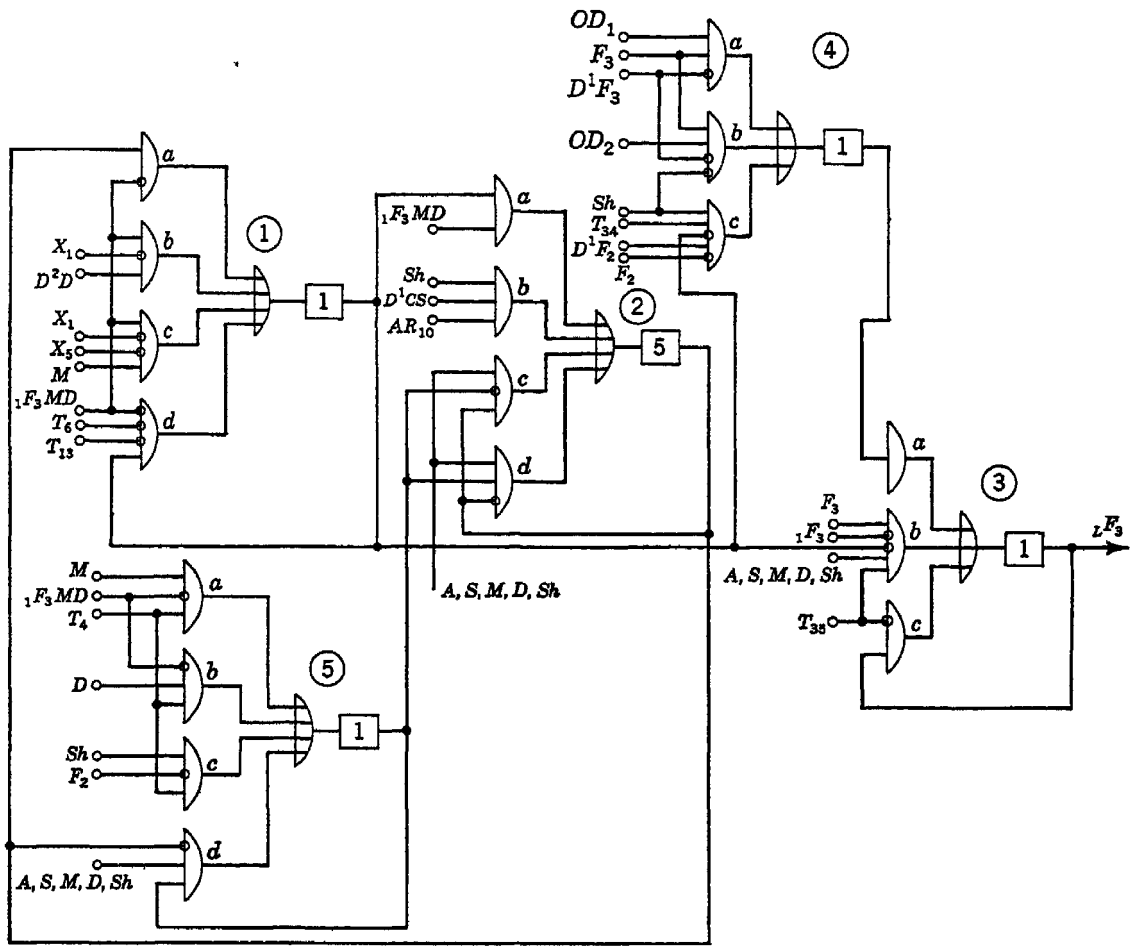



Fig. 9.45 Timing circuit for controlling duration of phase 3

operations are accomplished comparatively quickly and others in a much longer time, it is desirable to let the duration of this phase be determined by the operation to be performed, so as to minimize the time taken. Thus, in addition or subtraction, phase 3 has a duration of two minor cycles; one cycle is used to perform the addition, and the second is provided for complementation of the result, if necessary. Multiplication requires 30 minor cycles, and division 31. Shifting requires one minor cycle for each position of shift.

Certain logical operations take only one minor cycle. The circuitry that performs this timing is shown in Fig. 9.45. Gating circuit 3 in this

figure forms a flip-flop which is turned on only during the last minor cycle of phase 3. Circuit 4 provides one way of turning on circuit 3. Gate *a* of circuit 4 is driven by $OD_1$ (digit 1 of the operation code) and by $F_3$ and is inhibited by $D^1F_3$ (phase 3 pulses delayed by one clock period). Thus, if $OD_1$ is a 1 (as it will be during the operations STORE MULTIPLIER LOOP, FILE INSTRUCTION COUNTER, FILE BASE COUNTER, SET INSTRUCTION COUNTER, SET BASE COUNTER, BASE, and COMPARE), flip-flop 3 is at once turned on; therefore, the entire duration of phase 3 is only one minor cycle. Gate *b* of circuit 4 performs a similar function for the operations LOGICAL TRANSFER, LOAD ACCUMULATOR LOOP, and STORE ACCUMULATOR LOOP.

Circuit 3 can also be turned on either from its own gate *b*, or from gate *c* of circuit 4. In either case the flip-flop formed by circuit 1 must be off. The primary function of circuit 1 is to sense the presence of any 1's in the 5-bit loop formed by circuit 2 and its delay element. This loop initially contains the digit group 11100 if a multiplication is to be performed, or 11101 for a division. For a shift operation, it contains the shift constant. Circuit 5 generates one pulse per minor cycle, which is subtracted from the content of the 5-bit loop. When finally the 5-bit loop contains all 0's, the flip-flop of circuit 1 remains off during the next minor cycle. Gate *b* of circuit 3 then admits a single pulse at $T_{35}$, which allows the flip-flop of circuit 3 to be turned on. The next $T_{35}$ turns it off again, and its ON interval marks the last minor cycle.

The 5-bit loop of circuit 2 is initially loaded either from gates *b* or *c* of circuit 1 or from the address register by way of gate *b* of circuit 2. When a shift is to be executed, the shift constant, which was initially in positions 6 to 10 of the address register, is gated into this loop through its gate *b* by the shift signal and by $D^1CS$ (countersynchronizing pulses, delayed one clock period). The countersynchronizing pulses also shift address-register digits 6 to 10 around a five-position path and back to their original places in the address register. Since no memory access is required by the shift operation, phase 2 is used instead to load the shift constant into circuit 2 and its loop, so that the contents of this loop are available at the start of phase 3. The loading is done in synchronism with $D^1CS$, a pulse train which lasts from $T_5$ to $T_9$ of phase 2. Thus, during phase 3, the least-significant digit of the shift constant is emerging from the 5-bit loop at $T_5$. At this same instant, a pulse from circuit 5 is being applied to gates *c* and *d* of circuit 2. If the digit emerging from the loop is a 1, then gates *c* and *d* are both inhibited and flip-flop 5 is also kept in the OFF state. If the emerging digit is a zero, however, gate *d* admits a pulse and flip-flop 5 is turned on, producing a pulse at $T_6$, and the action of the circuit repeats. By tracing a few pulse sequences, the reader may verify that the combined action of circuits 2 and 5 has the

effect of reducing the content of the 5-bit loop by 1 for each minor cycle. When the content of this loop has finally been reduced to zero, there are no further inputs to the flip-flop of circuit 1, which then remains off and allows flip-flop 3 to signal the last minor cycle of phase 3.

If a multiplication is to be performed, the 5-bit loop of circuit 2 is loaded from gate *c* of circuit 1; for a division, gate *b* is used; 30 minor cycles are needed for multiplication and 31 for division. During the first minor cycle of phase 3 in multiplication, gate *c* admits pulses except when $X_1$ or $X_5$ occurs. $X_1$ and $X_5$ are derived from one of the pulse-distributor chains. $X_1$ occurs at $T_5$, $T_{10}$, $T_{15}$, etc., and $X_5$ at $T_4$, $T_9$, etc. Thus pulses appear at gate *a* of circuit 2 at $T_2$, $T_3$, and $T_4$. (They appear again every five pulse periods during this minor cycle, but these subsequent reappearances have no effect.) The output of the 5-bit loop at $T_5$, $T_6$, $T_7$, $T_8$, $T_9$ is therefore 0,0,1,1,1, which forms the binary equivalent of 28, with the least-significant digit appearing at $T_5$. Gate *b* of circuit 1 functions in similar fashion during division and admits a group of pulses forming the binary equivalent of 29. In each case the digit group represents a number two less than the number of minor cycles required, since the first minor cycle of phase 3 commences before the 5-bit loop is loaded, and the last occurs after its contents have been reduced to zero. Thus the required 30 minor cycles for multiplication and 31 for division are correctly timed.

**Arithmetic circuits.** The principal circuits that may be regarded as forming the arithmetic unit of the machine include the accumulator loop, the multiplier loop, the adder, two complementers, and various gates and flip-flops for controlling the flow of information to and from these circuits. The accumulator loop shown in Fig. 9.46 normally contains 35 clock periods of delay, with gating circuits interspersed at various points in the loop. Information entering this loop normally comes from the adder via gates *b* or *c* of circuit 1. Information that is to be circulated in the loop without being shifted relative to the timing pulses passes through gate *b* of circuit 2. If information is to be shifted left, a SHIFT LEFT signal inhibits gate *b* and enables *d*, making the loop cycle time be 36 clock periods. Thus, in one minor cycle of 35 periods, the loop contents are not quite completely circulated and lie finally one position to the left of their initial position. In a similar manner, a SHIFT RIGHT signal inhibits *b* and enables *c*, shortening the loop by one period and causing the loop contents to be circulated completely and moved one position more in each minor cycle. Although this is a slow shift, requiring one minor cycle per position of shift, it is more economical of equipment than a bidirectional shift register would be.

The functions of the other gating circuits in the accumulator loop can best be understood by examining first the complementing and adding

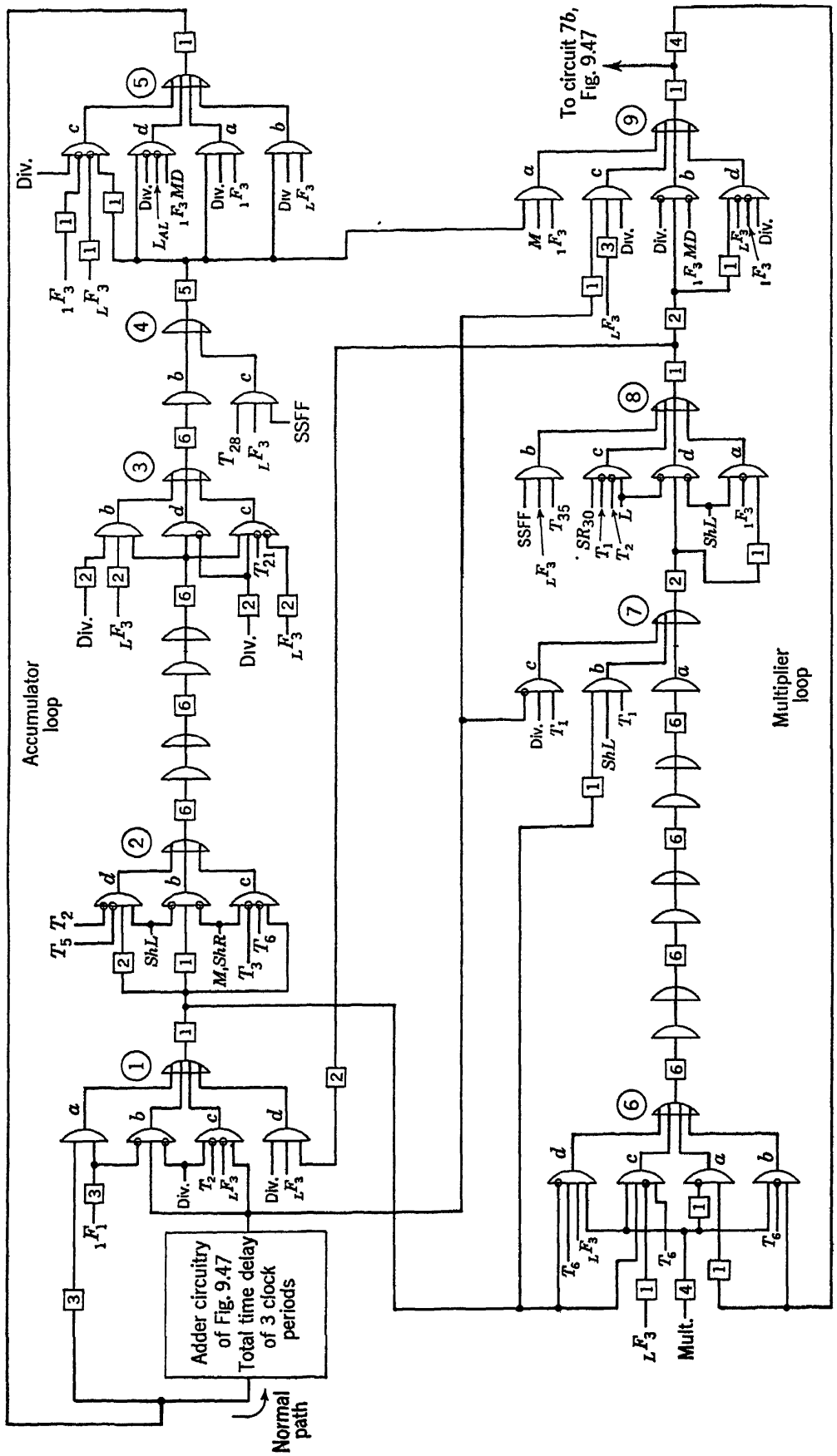



Fig. 9.46 Multiplier and accumulator loops

circuits of Fig. 9.47. Circuits 4 and 9 in this figure together form a conventional three-input adder. This adder, together with circuit 3 and a 1-unit delay element, are directly in the accumulator loop except when they are bypassed during the first minor cycle of phase 1. (See circuit 1*a* of the accumulator loop, Fig. 9.46.) Circuit 3 of Fig. 9.47 acts either to recirculate accumulator digits without change or to recirculate them after complementation. Circuits 1 and 2 control the complementing action of circuit 3. Circuit 1 examines the sign digit, which is emerging from the adder at $T_2$. If this is a 1 (signifying a negative number), the flip-flop of circuit 1 is turned on, and pulses from it are applied to circuit 2 at $T_3$ to $T_{33}$. The flip-flop of circuit 2 cannot be turned on, however, until a 1 from the accumulator loop appears at its input. The first 1 that appears sets this flip-flop but also passes unaltered through gate *d* of circuit 3. The pulses from circuit 2, however, change all more significant 1's to 0's and 0's to 1's, by the action of gate *c* of circuit 3. Thus, true radix complementation is accomplished.

Both addition and subtraction are performed by admitting the content of the shift register to the adder circuit in synchronism with the circulating content of the accumulator. Circuit 8 of Fig. 9.47 performs this gating by passing the content of shift-register stage 30 ($SR_{30}$) to the adder at times determined by circuits 6 and 7. The flip-flop of circuit 11 determines whether the shift-register contents are complemented while being gated to the adder. Circuit 10 examines the sign bit in $SR_0$ and produces an output pulse if the sign bit is 1 and an addition is to be performed (gate *a*) or if the sign bit is 0 and a subtraction is to be performed (gate *b*). Circuit 11 then produces a string of pulses at $T_3$ to $T_{34}$ to complement the content of the shift register as it passes through circuit 8. Since the complementation performed by circuit 8 is one's complementation (exchange of 1's and 0's), a 1 must be added in the least-significant-digit position to form the radix complement. This step is performed by gate *a* of circuit 9, which admits a pulse to the carry circuit at $T_3$.

In multiplication of two numbers, the multiplier, which was initially in the accumulator loop, is gated into the multiplier loop by the action of circuits 5*d* and 9*a* (Fig. 9.46) during the first minor cycle of phase 3. The accumulator loop is shortened by 1 unit (circuit 2*b* and *c*), and the multiplicand is added into this loop each time a 1 is detected in the multiplier. The multiplier loop is also shortened 1 unit (circuit 6*a* and *b*), thus shifting the multiplier one position right for each minor cycle. In successive minor cycles the successive digits of the multiplier, starting with the least significant, emerge from circuit 9 of the multiplier loop and are applied to circuit 7*b* of Fig. 9.47 at $T_2$. Flip-flop 7 of Fig. 9.47 then gates the multiplicand from the shift register into the adder if the multiplier digit was a 1 or blocks it if it was a 0. The multiplier digit that was

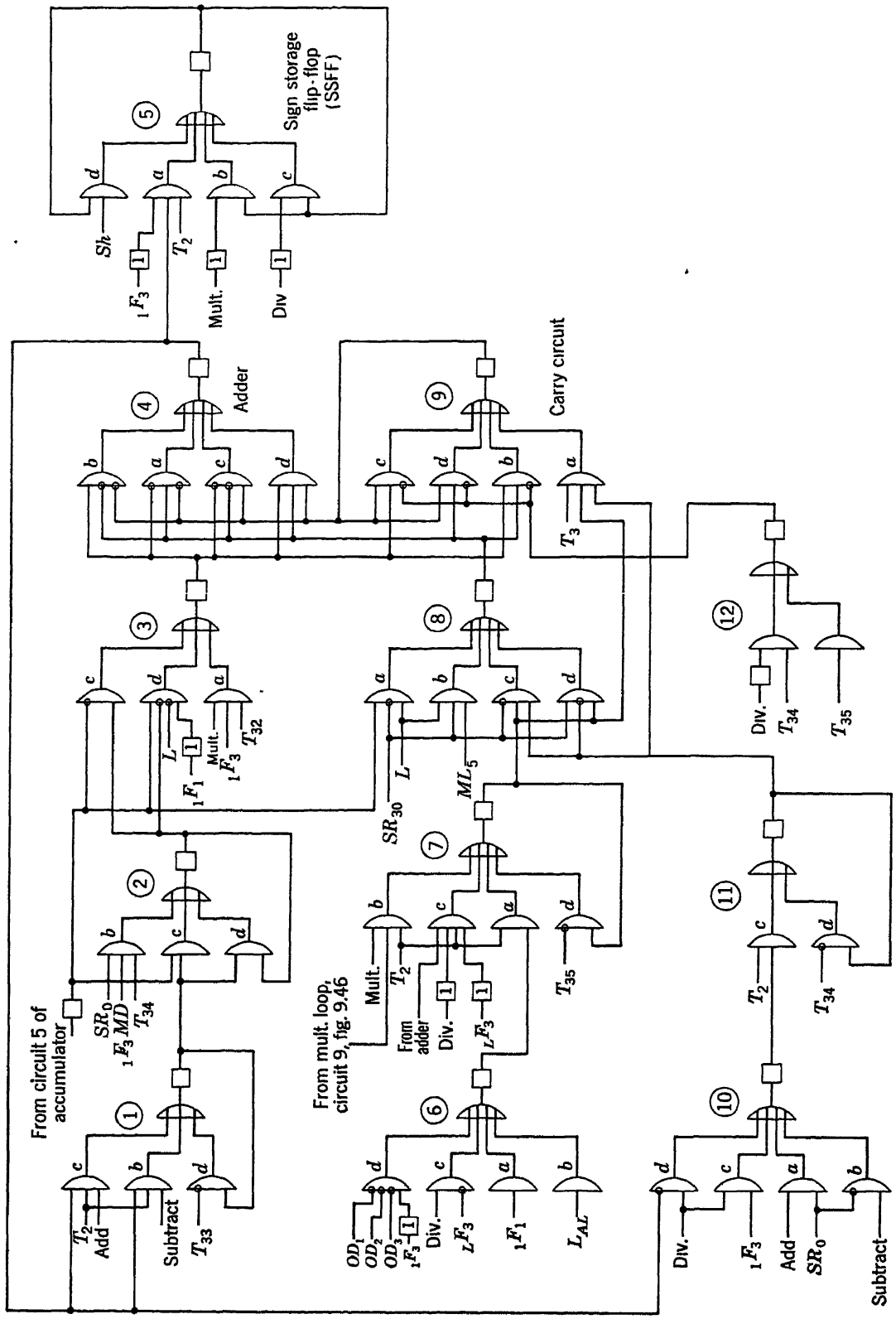



Fig. 9.47 Arithmetic circuits

sampled at $T_2$ is blocked from reentering the multiplier loop at $T_6$ by circuit 6*b* (Fig. 9.46). In its place the least-significant bit from the accumulator loop is admitted by circuit 6*c* at $T_6$, except in the last minor cycle of phase 3. At $T_6$ of the last minor cycle of phase 3, the digit entering circuit 6 is complemented at gate *d*. This is necessary, because a round-off operation, performed during the first minor cycle of phase 3, altered this digit. The round-off was performed by adding 1 to the most significant digit of the first partial product by admitting a pulse at gate 3*a* of Fig. 9.47 at $T_{32}$ of the first minor cycle of phase 3. The ultimate effect of this step is to round the high-order part of the double-length product, while altering the most significant digit of the low-order product. After the correction has been made, the accumulator loop stores the rounded high-order product, and the multiplier loop stores the low-order product.

The division circuitry employs the subtract-and-add algorithm described in Sec. 9.9 of this chapter. The dividend is initially in the accumulator loop, and the divisor may be assumed to be in the shift register. Circuits 6*c* and 7 of Fig. 9.47 cause the content of the shift register to be transmitted to the adder through circuit 8*c* and *d*. Circuits 10*c* and 11 complement the output of the shift register during the first minor cycle of phase 3. Thereafter, the complementation of the shift register output is determined by circuit 10*d*, which prevents complementation if the remainder in the accumulator is negative or causes it to occur if the remainder is positive. The digit that gives this information is the digit emerging from the adder delay element (circuit 4) at $T_1$. After the first minor cycle of phase 3, the accumulator loop is lengthened 1 unit by circuits 5*c* and *d* (Fig. 9.46), so that subsequent subtractions or additions of the divisor find the remainder moved successively one position further left in each minor cycle. The digit that gives the sign of the remainder also determines the next less significant digit of the quotient. This is done by gate 7*c* of Fig. 9.46, which admits a 1 to the multiplier loop if the sign digit of the remainder is 0 but otherwise admits a 0. After having been sensed, the remainder sign digit is suppressed in the accumulator loop by circuit 3*c*, which is inhibited at $T_{21}$, the time of arrival of the sign digit. This gate is bypassed during the last minor cycle of phase 3. The multiplier loop is also lengthened 1 unit in division (by circuits 9*b* and *d*, Fig. 9.46), so that successive quotient digits are placed one position to the right of the previously determined digits in each minor cycle. The final step of the division operation transfers the quotient digits from the multiplier loop to the accumulator loop while allowing the remainder digits to pass from the accumulator into the multiplier loop. To do this, gate 1*c* is blocked during the last minor cycle of phase 3 ($_LF_3$) and information emerging from the adder is

admitted to the multiplier loop at gate 9*c*. Similarly, gate 9*d* blocks circulation of the quotient digits in the multiplier loop, and gate 1*d* admits them to the accumulator. The unit of delay added to the accumulator loop during most of the division operation is bypassed during ${}_LF_3$ by circuit 5*b*. Also during ${}_LF_3$, the divisor is added to the remainder if the remainder was negative. Gate *c* of circuit 7 detects this condition and turns on flip-flop 7 to allow the divisor to enter the adder.

Sign determination in multiplication and division is performed by circuits 2 and 3 of Fig. 9.47. Gate 2*b* admits the sign digit from the shift register at $T_{34}$ of the first minor cycle of multiplication or division. This digit arrives at gates 3*c* and 3*d* in synchronism with the sign digit in the accumulator loop. If the signs are different, a 1 is passed by circuit 3, but, if they are the same, a 0 is passed (i.e., no pulse). This resultant sign digit is stored by circuit 5 upon its arrival there at $T_2$. If a multiplication is being performed, the sign digit from SSFF is admitted to the accumulator by gate 4*d* (Fig. 9.46) at $T_{27}$ of ${}_LF_3$. In division or in shifting, the sign digit is admitted at $T_{28}$ by gate 4*c*, the difference in timing being due to the difference in accumulator-loop length. The proper sign is also inserted in the multiplier loop at the end of phase 3. Gate 8*b* performs this function at the last timing pulse ($T_{35}$) of the last minor cycle of phase 3.

**Communicating with the computer.** All the logic circuitry so far considered has been concerned with the manipulation of data inside the machine. The questions naturally arise: How is this information placed in the computer initially, and how is it removed? Like most computers, the MIC can employ many different types of input and output devices, such as electric typewriters, paper-tape punches and readers, paper-card punches and readers, and magnetic tapes. We shall consider here only the paper-tape input-output system, since its characteristics exemplify many of the problems encountered in the other systems.

Although data can be fed directly from an electric typewriter to the computer, this mode of input is not so fast as reading into the computer from a previously prepared tape. The tape can be prepared by any electric typewriter having the same code. Each time a typewriter key is depressed, the tape punch mechanism punches a group of holes in the tape and advances it for the next character. Each group may have as many as seven holes and always has one hole in a fixed position to indicate presence of a character on the tape. Table 9.10 shows the other six hole locations and the binary code for each of the characters on the MIC keyboard. It will be noted that, for the first 32 characters on the list, the sixth (leftmost) hole is not punched. Thus 5 bits suffice to specify any one of these 32 characters. A binary number to be inserted in the machine may be divided into 5-bit groups, each corresponding to one of

these 32 keyboard characters. Each key that is depressed then punches the corresponding hole group on the tape, the most significant being typed first and the least significant last.

**Table 9.10**
**MIC typewriter code**

| Hole group | Binary code | Keyboard character | Hole group | Binary code | Keyboard character |
|---|---|---|---|---|---|
| | 000000 | 0 | 00 0 | 011010 | q |
| 0 | 000001 | 1 | 00 00 | 011011 | r |
| 0 | 000010 | 2 | 000 | 011100 | s |
| 00 | 000011 | 3 | 000 0 | 011101 | t |
| 0 | 000100 | 4 | 0000 | 011110 | u |
| 0 0 | 000101 | 5 | 00000 | 011111 | v |
| 00 | 000110 | 6 | 0 | 100000 | Tape feed |
| 000 | 000111 | 7 | 0 0 | 100001 | w |
| 0 | 001000 | 8 | 0 0 | 100010 | x |
| 0 0 | 001001 | 9 | 0 00 | 100011 | y |
| 0 0 | 001010 | a | 0 0 | 100100 | z |
| 0 00 | 001011 | b | 0 0 0 | 100101 | = |
| 00 | 001100 | c | 0 00 | 100110 | + |
| 00 0 | 001101 | d | 0 000 | 100111 | – |
| 000 | 001110 | e | 0 0 | 101000 | Upper case |
| 0000 | 001111 | f | 0 0 0 | 101001 | < |
| 0 | 010000 | g | 0 0 0 | 101010 | · |
| 0 0 | 010001 | h | 0 0 00 | 101011 | * |
| 0 0 | 010010 | i | 0 00 | 101100 | Lower case |
| 0 00 | 010011 | j | 00 0 | 110100 | Back space |
| 0 0 | 010100 | k | 00 0 0 | 110101 | Carriage return |
| 0 0 0 | 010101 | l | 0 0000 | 101111 | / |
| 0 00 | 010110 | m | 0 00 0 | 101101 | Space |
| 0 000 | 010111 | n | 00 0 | 110001 | Tabulate |
| 00 | 011000 | o | 000 | 111000 | Stop code |
| 00 0 | 011001 | p | 000000 | 111111 | Code delete |

After the tape is prepared, it is read into the computer. However, since the computer operates upon numbers by treating first the least-significant digits, whereas the tape presents first the most significant digits, the machine must rearrange the sequence of the 5-bit groups. This is accomplished by admitting each 5-bit group from the tape reader to the last five stages of the shift register ($SR_{26}$ to $SR_{30}$) and then performing a 25-position right shift with the output of $SR_{30}$ entering $SR_1$. This shifts the information five positions to the left and leaves it in $SR_{21}$

to $SR_{25}$. (The long right shift is necessary because the MIC shift register is not bidirectional.) Six such operations place a 30-bit group in the shift register, with the most significant digit in $SR_1$ and the least significant in $SR_{30}$. The sign character, which precedes the six number groups on the tape, is admitted directly to $SR_0$, the sign position of the shift register.

The machine communicates with its input-output devices whenever it is performing an instruction requiring such communication. Thus, in doing such operations as STORE CONTENT OF ACCUMULATOR LOOP, LOAD ACCUMULATOR LOOP, FILE INSTRUCTION COUNTER, and others, the digits of the operation code activate a READ flip-flop or a WRITE flip-flop. The address digits in turn determine what input-output device is involved (tape, typewriter, toggle-switch banks, etc.) and whether characters to be transferred are 6-bit groups or 5-bit groups. It will be recalled that, of the 1,024 addresses that can be specified with ten binary digits, only 992 are used for cells in the computer memory. The remaining 32, which are those in which the five most significant digits are all 0, specify input-output devices. Ten of these 32 are for toggle-switch registers on which small programs can be inserted manually. Another address calls for transfer to or from the keyboard of the typewriter in 5-bit groups, another to or from tape in 5-bit groups, and two others for transfer to or from either keyboard or tape in 6-bit groups. The 5-bit mode is referred to as "base 32", and the 6-bit mode is called "alphanumeric", since it makes possible the use of any keyboard character, whether alphabetic or numerical. When an instruction contains an address specifying the alphanumeric mode, the transfer to or from the shift register involves the last six stages ($SR_{25}$ to $SR_{30}$), and the long right shift is 24 positions. Five such operations completely load (or unload) the shift register in the alphanumeric mode.

The reader who has successfully followed through this maze of logic circuitry will have acquired some insight into the complex problems of sequencing and timing involved in the design of the digital computer. Since this was, after all, one of the principal reasons for presenting this discussion, further details concerning the MIC would be superfluous, and the reader might better turn to consideration of the logic of machines to which he is closer.

It should be noted finally that, although enough of the logic of the MIC has been described to show the major features of its design, several circuits and machine operations have been mentioned only incidentally or not at all. Operations not described include the following: COMPARE, LOGICAL TRANSFER, BASE, SET BASE COUNTER, SET INSTRUCTION COUNTER, FILE BASE COUNTER, FILE INSTRUCTION COUNTER, STORE ACCUMULATOR LOOP, and STORE MULTIPLIER LOOP. Circuits that have not been dis-

cussed include the timing-pulse generator, the base counter, the instruction counter, the overflow detector, shift-register synchronizers, and typewriter-and-tape synchronizers. The relative-addressing system has been omitted, and also the manual controls and the operator's console. Discussion of these topics lies far beyond the scope of this chapter or indeed of this book.

## REFERENCES

*References cited by Number:*


1. Gilchrist, B., J. H. Pomerene, and S. Y. Wong, Fast carry logic for digital computers, *IRE Trans. on Electronic Computers*, vol. EC-4, no. 4, December, 1955, pp. 133–136.
2. Estrin, G., B. Gilchrist, and J. H. Pomerene, A note on high-speed digital multiplication, *IRE Trans. on Electronic Computers*, vol. EC-5, no. 3, September, 1956, p. 140.
3. Wilkes, M. V., and J. B. Stringer, Micro-programming and the design of the control circuits in an electronic digital computer, *Proc. Cambridge Phil. Soc.*, vol. 49, 1953, pp. 230–238.
4. Wilkes, M. V., W. Renwick, and D. J. Wheeler, The design of the control unit of an electronic digital computer, *Proc. IEE London*, vol. 105, pt. B, no. 20, March, 1958, pp. 121–128.
5. Lehman, M., High-speed digital multiplication, *IRE Trans. on Electronic Computers*, vol. EC-6, no. 3, September, 1957, pp. 204–205.
6. Robertson, J. E., Two's complement multiplication in binary parallel digital computers, *IRE Trans. on Electronic Computers*, vol. EC-4, no. 3, September, 1955, pp. 118–119.
7. Burks, A. W., H. H. Goldstine, and J. von Neumann, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," Institute for Advanced Study, Princeton, N.J., 1946, pp. 13–19.
8. Lehman, M., Short-cut multiplication and division in automatic binary digital computers, *Proc. IEE London*, vol. 105, pt. B, no. 23, September, 1958, pp. 496–504.
9. Booth, A. D., A signed binary multiplication technique, *Quart. J. Mech. and Appl. Math.*, vol. 4, pt. 2, June, 1951, pp. 236–240.
10. Richards, R. K., "Arithmetic Operations in Digital Computers," D. Van Nostrand Company, Inc., Princeton, N.J., 1955, pp. 174–176.
11. Burks, A. W., H. H. Goldstine, and J. von Neumann, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," Institute for Advanced Study, Princeton, N.J., 1946, pp. 23–29.
12. Buchholz, W., The system design of the IBM type 701 computer, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1262–1275.


*Supplementary References:*


Wilkes, M. V., "Automatic Digital Computers," John Wiley & Sons, Inc., New York, 1956, chaps. 2 and 6.

Richards, R. K., "Arithmetic Operations in Digital Computers," D. Van Nostrand Company, Inc., Princeton, N.J., 1955, chaps. 4–9.

Bowden, B. V. (ed.), "Faster than Thought," Pitman Publishing Corporation, New York, 1953.

Weinberger, A., and J. L. Smith, A one-microsecond adder using one-megacycle circuitry, *IRE Trans. on Electronic Computers*, vol. EC-5, no. 2, June, 1956, pp. 65–73.

Brown, W., J. DeTurk, H. Garner, and E. Lewis, "The MIDSAC Computer," University of Michigan Press, Ann Arbor, Mich., 1954. (Contains a description of a high-speed serial-parallel multiplier.)

Felker, J. H., Typical block diagrams for a transistor digital computer, *Trans. AIEE*, vol. 71, pt. I, 1952, pp. 175–182.

## PROBLEMS

**9.1** For each of the following round-off methods, determine the maximum and minimum errors of an $n$-digit binary fraction and find the variance of the error distribution:

*a.* No rounding; succeeding digits are simply dropped.

*b.* The last retained digit is always made a 1.

*c.* 1 is added into the $(n + 1)$st position of the original number, and the first $n$ bits of the result are retained.

Consider the error to be defined as the true representation of the number minus its machine representation. In finding maximum and minimum errors, retain the sign of the error.

**9.2** Design a carry-completion sensing circuit for use with a parallel adder constructed with dynamic circuitry. Use two-level elements (AND followed by OR) with the basic full-adder circuit shown in Fig. 9.8. Assume that each AND-OR cascade has a time delay of exactly one pulse period. Addend and augend digits are applied for as many pulse periods as are needed to complete the operation. If the average longest carry is 5.6 positions, what is the average number of pulse periods needed to complete an addition?

**9.3** Show logic circuits, using dynamic-circuitry symbols, to perform each of the following functions:

*a.* Admit only the first pulse of a train of successive pulses.

*b.* Admit only the last pulse of a train of successive pulses.

*c.* Admit every third pulse of a train of successive pulses.

*d.* Reject every third pulse of a train of successive pulses.

*e.* Admit every third pulse from a sequence of pulses that do not necessarily occur at successive clock pulse times, although they do occur in synchronism with clock pulses.

*f.* Produce a single pulse corresponding to the first pulse on the input line and thereafter produce a pulse corresponding to each absent pulse on the input line.

*g.* Arrange two flip-flops so that one or the other is on at any time but never both. Each missing pulse in a stream of input pulses turns off the flip-flop that was on and turns on the flip-flop that was off.

**9.4** Devise a switching circuit (using dynamic-circuitry symbols) for converting (*a*) reflected-code groups to conventional binary code, and (*b*) conventional binary-code groups to reflected code. Do this both for serial and for parallel presentation of the digits.

**9.5** Devise a logic circuit around a delay element having a total delay of three clock pulse periods that will count *down* one every time an input pulse occurs. That is, the pattern of 1's and 0's (pulses or no pulses) will form a binary number equal to 7 minus the number of input pulses. Assume that the input pulses are separated by three pulse periods (or any integral multiple of 3).

**9.6** A serial-parallel adder system is to be built from three-input adders that have a delay time of $\delta$ clock periods ($\delta$ a fraction) between the arrival of the input-digit signals and the emergence of the sum and carry signals. The general scheme is shown in the diagram. The least-significant $n$ digits are presented to the $n$ adders through staggered delays. One pulse period later, the next more significant $n$ digits are presented on the same inputs, and simultaneously the carry from the first addition is presented at the input to the low-order adder stage.

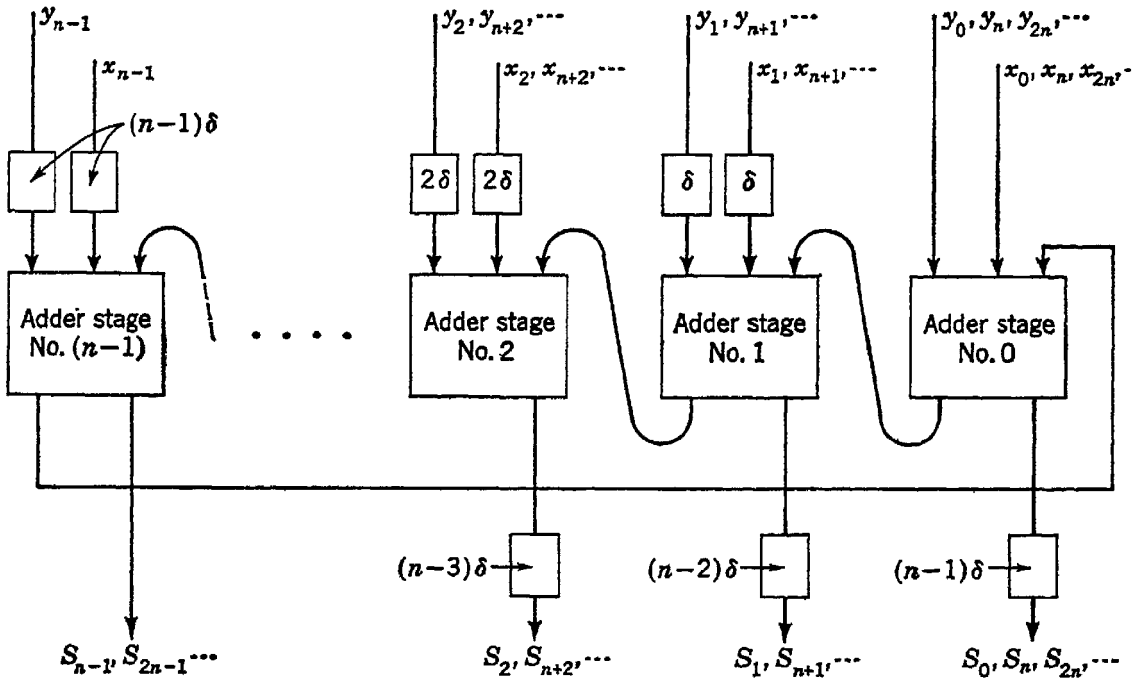



Fig. P9.6

If $\delta = \frac{1}{4}$ pulse period, what is the total time required to add two 40-bit binary numbers in this manner? If $\delta = \frac{1}{10}$, what time is required?

If $a + 1 > 1/\delta \geq a$, where $a$ is an integer, what is the optimum number of adder stages to use? What magnitude of compensating delays should be used, and where should they be put in the circuit?

CHAPTER 10

# Electronic Circuits for Logic

## 10.1 Introduction

In this chapter are presented the fundamentals of a number of methods of performing logic by means of electronic devices and circuits. Although it was convenient in many places in Chap. 8 to demonstrate the realization of logical functions by using networks of switches as examples, in practice simple contact networks are not used for the logic of high-speed digital computers because they are too slow. Rather, much faster-acting electronic devices are used to open and close circuits or to produce signals in a manner dictated by the logical interconnection of the devices. Devices that will be considered in this chapter are vacuum tubes, semiconductor diodes, and transistors.

Elements for performing logical functions are usually called "gates". When it is necessary to distinguish among them, they are called "OR gates", "AND gates", "INVERTING gates", and other names that indicate their action. In the early days of computer technology, the term "gate" was reserved for a device that performed the AND function, and OR devices were known as "buffers". The reader may encounter this terminology in the early literature, but modern practice is to use the word "buffer" to refer to an intermediate storage device by means of which two nonsynchronous parts of a digital system can communicate with each other.

In the next section we shall consider briefly the operation of gates constructed from vacuum tubes, from diodes, and from transistors, and in subsequent sections we shall examine in more detail the behavior of some complete gating systems based upon these devices.

## 10.2 Survey of gate circuits

**Vacuum-tube gates.** Although in computers of modern design vacuum tubes are seldom used as gates, such use was common in machines

designed and built during the first decade of the electronic computer (1945–1955). These machines will be in use for some years to come, and the reader should understand something of their operation. Circuits using vacuum tubes as gates also serve as models for some types of transistor gating devices.

Five vacuum-tube gating devices are shown in Fig. 10.1. Two of these use triodes, and two use diodes. Accompanying each circuit is a

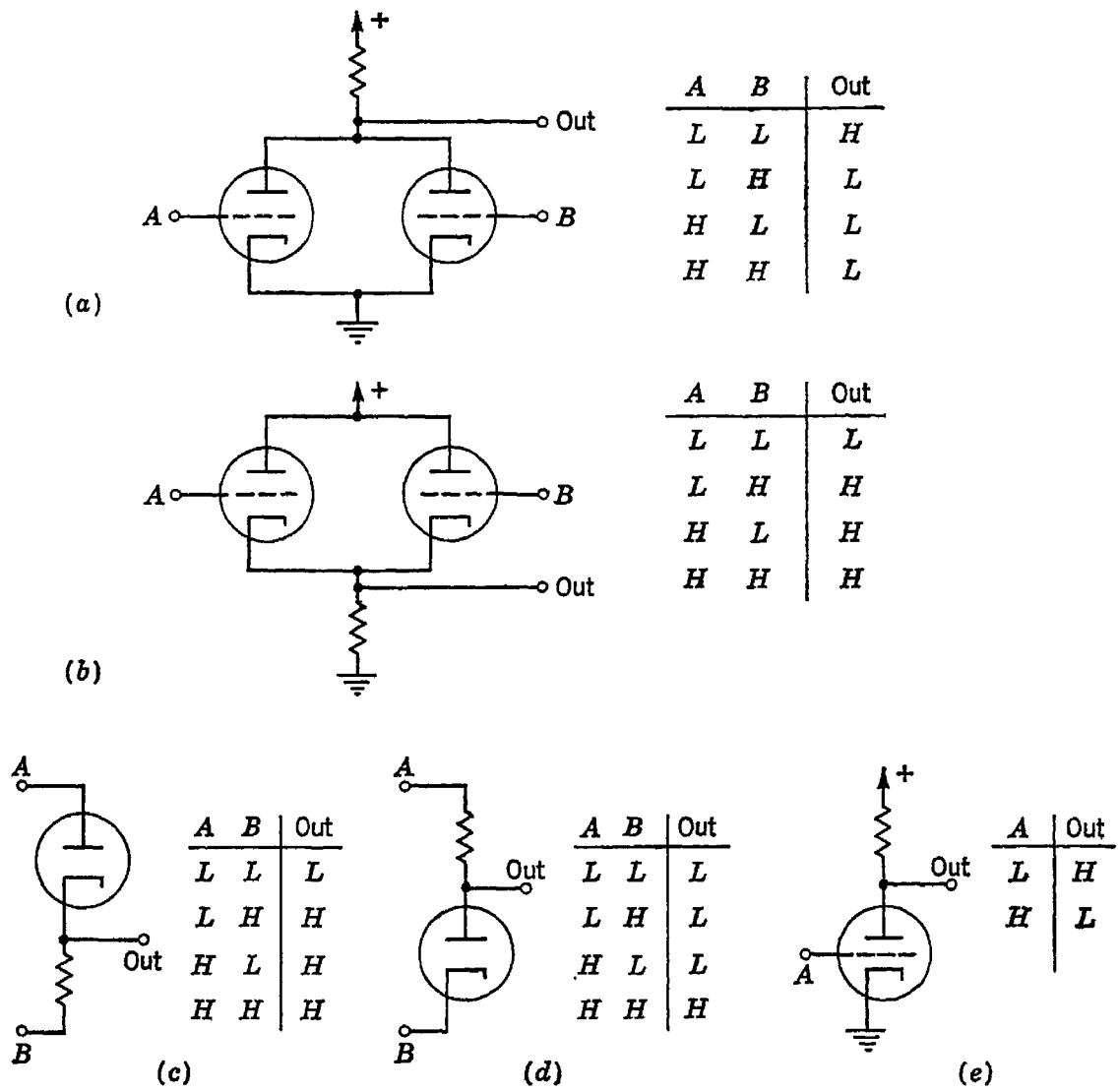
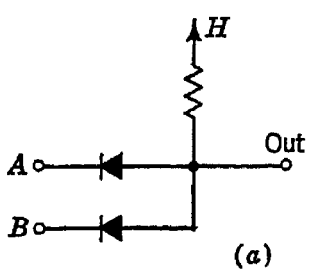



Fig. 10.1 Vacuum-tube gating devices

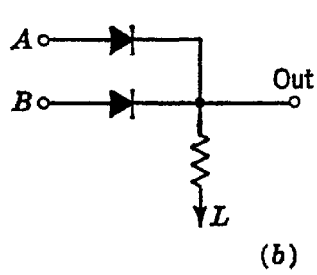
table of combinations showing the conditions under which the output is at high potential ($H$) or at low potential ($L$) for various combinations of high and low levels on the inputs. It is apparent that each of these circuits (except the inverter) can be regarded as realizing either of two logical functions, depending on whether the higher potential level is assigned a truth value of 1 or a truth value of 0. Thus, if $L = 0$ and $H = 1$, the circuit of Fig. 10.1$a$ is a realization of the dual stroke function,

10.1*b* represents the OR function, 10.1*c* also represents the OR function, and 10.1*d* realizes the AND function. With the other assignment of truth values, 10.1*a* becomes the Sheffer stroke function, 10.1*b* the AND function, 10.1*c* the AND function, and 10.1*d* the OR function. Figure 10.1*e*, of course, is an inverter under either assignment of truth values.

In all these circuits the question of what constitutes a "high" or a "low" potential level depends on several factors. Of course, the two levels are high and low only with respect to each other, and the entire circuit may operate well above ground or well below. Thus, in the triode circuit of Fig. 10.1*a*, the low level at the input is a level low enough to cut off the tube, perhaps −10 volts, and the high level at the input is in the vicinity of ground potential. The high and low levels at the output, however, are determined by the plate supply voltage, the plate load resistance, and the characteristics of the triode. In Fig. 10.1*b*, the low level is approximately ground potential, and the high level is anything




| A | B | Out |
|---|---|---|
| L | L | L |
| L | H | L |
| H | L | L |
| H | H | H |




| A | B | Out |
|---|---|---|
| L | L | L |
| L | H | H |
| H | L | H |
| H | H | H |

Fig. 10.2 Basic diode gates

above this, the circuit acting as a cathode follower. The triode circuit of Fig. 10.1*a* is used where it is desired to change the voltage levels of the signals, and the circuit of Fig. 10.1*b* is used to provide power gain without change of voltage level. The diode circuits are useful where the output point is not required to supply much power and no change of voltage level is needed.

**Diode gates.** Although diode gates can be made of either thermionic or semiconductor diodes, in modern circuits the semiconductor diode is used almost exclusively. The two basic circuits of the diode gate are shown in Fig. 10.2. If we identify the high level of potential *H* with the binary digit 1 and the low level *L* with the binary digit 0, then the gate of Fig. 10.2*a* performs the AND function and the gate of 10.2*b* performs the OR function. Interchanging these assignments interchanges the functions performed by the gates.

The chief difference between the gates of Fig. 10.1*c* and *d* and those of Fig. 10.2 is that, in the former, one end of the resistor is used as a signal input point, whereas, in the circuit of Fig. 10.2, each signal is applied through a diode, and the resistor terminal is fixed to a reference potential. Although it is possible with either type of circuit to include additional

signal input points by connecting in another diode for each input, it is generally preferable with multi-input gates to have all input points be of the same type. Thus the circuit of Fig. 10.2 is commonly used for gating together many signals. (For electrical reasons which we shall examine shortly, it is seldom practical to gate together more than six or seven signals in one circuit. In situations that require more gate inputs than this, the usual procedure is to perform the gating in two or more smaller groups.)

**Transistor gates.** There are several ways in which gate circuits can be constructed from transistors. One of the simplest circuits forms the

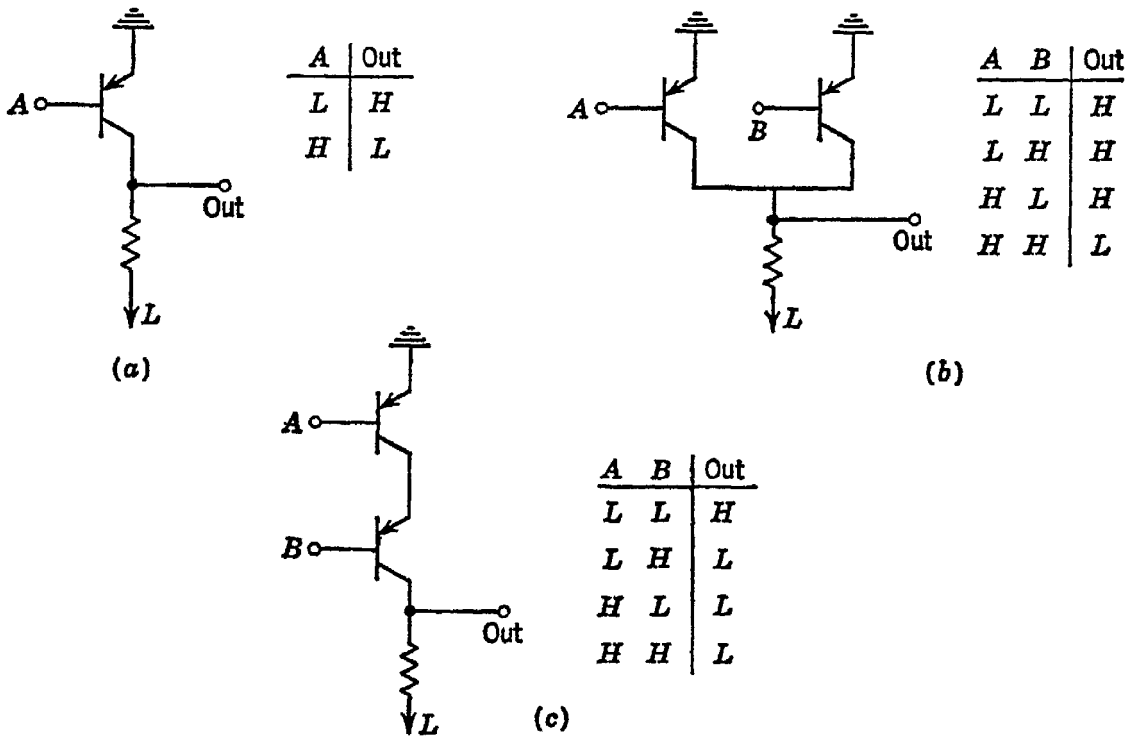



Fig. 10.3 Basic circuits for direct-coupled transistor logic

basis of what is called "direct-coupled transistor logic", or DCTL for short. Examples of this type of logic are shown in Fig. 10.3. All the circuits in this figure are constructed around the grounded-emitter switch of Fig. 10.3*a*. This circuit is simply an inverter. When the base is driven to a sufficiently negative potential, both the emitter-base diode and the collector-base diode conduct, and the transistor is driven into saturation. The voltage between emitter and collector drops to something in the vicinity of 30 mv. When the base current is removed by raising the base potential, the transistor is cut off, the collector current falls to a few microamperes, and the output voltage drops to $E_{cc}$, a typical value for which might be $-3$ volts.

Two (or more) of these grounded-emitter switches, with the collectors

tied to a common resistance, give the circuit of Fig. 10.3*b*. If the low-potential and high-potential signal levels are identified respectively with binary 0 and 1, this circuit performs the Sheffer stroke function. Of course, $A|B = \overline{A \cdot B}$; so this circuit is sometimes called a NOT AND (or NAND) circuit. If the output is followed by an inverter, then the combined circuit performs the AND function. Opposite identification of signal levels with the binary digits 0 and 1 makes the circuit perform the dual stroke or NOR function.

Figure 10.3*c* shows a series connection of transistors. Only if inputs $A$ and $B$ are both at a low level will the two transistors conduct, thus raising the output voltage from approximately $E_{cc}$ to almost ground level. If $L$ and $H$ are identified respectively with 0 and 1, then this circuit performs the dual stroke function (that is, NOT OR or NOR); with the opposite identification, it realizes the Sheffer stroke. This circuit could also be followed by an inverter to make the combined circuit perform the OR function (for $L = 0$ and $H = 1$).

It is interesting to note in this regard that the convention determining whether the low level or the high level is to be equivalent to the binary digit 1 is entirely at the designer's choice. He may even wish to change this choice from point to point in the system and may do so provided he keeps careful track of the choices made for the various signals. Accordingly, the circuit of Fig. 10.3*b* may be considered to be an AND circuit if $L = 0$ and $H = 1$ at the input and if at the output $L$ is treated as 1 and $H$ as 0. Such changes in identification often make it possible to construct systems using fewer inverters than would otherwise be required.

**Temporary storage in logic circuits.** In addition to gating devices, a digital system requires the use of temporary-storage elements as part of its logic circuitry. Any sequential circuit must include such devices, as means for responding to its past input. Also, many circuits intended to be purely combinational have time delays in their operation that make it necessary to store some of their inputs or outputs until a time at which they can be used. Several such "combinational" circuits, having different time delays, may have to drive the same gating circuit, and the temporary-storage devices provide means for making such signals available at the same time. The two types of storage elements commonly used are (1) flip-flops based on the Eccles-Jordan circuit, and (2) short sections of electrical delay line. When delay lines are used, the signals are available only at the moments that they emerge from the lines, and careful synchronization is necessary. On the other hand, if Eccles-Jordan flip-flops are used, the timing problem is to ensure that, before the outputs are used, all flip-flop action that should occur actually has occurred. Each system has its own set of problems, its advantages and disadvantages, its enthusiastic proponents, and its violent opponents.

## 10.3 Dynamic circuitry

Having briefly examined the three common types of logic elements, let us now study in some detail a full electronic circuit based on one of them. This section describes the use of diode logic in conjunction with amplifying elements (tubes or transistors), pulse transformers, and electrical delay lines. This electronic implementation of logic is a synchronous system and is known as "dynamic circuitry" or sometimes as "a-c circuitry". It is the form of circuitry used in most of the logic structures presented in Chap. 9.

**Diode AND gates.** Consider first the behavior of a simple diode AND gate, as shown in Fig. 10.4. The reference potential to which

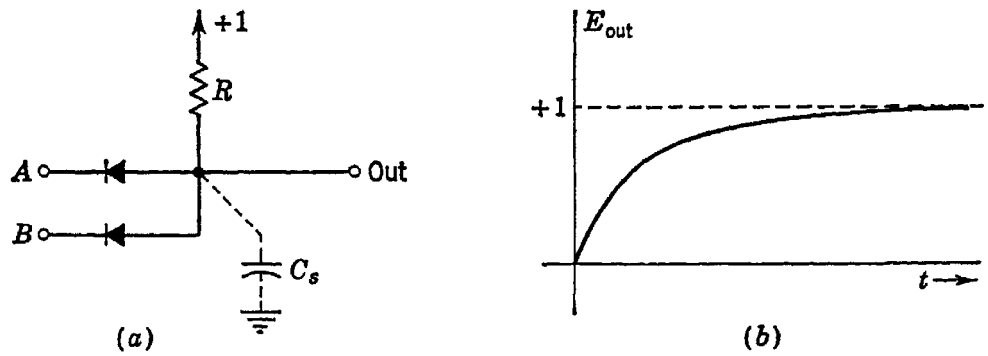



Fig. 10.4 Diode AND gate and its output voltage

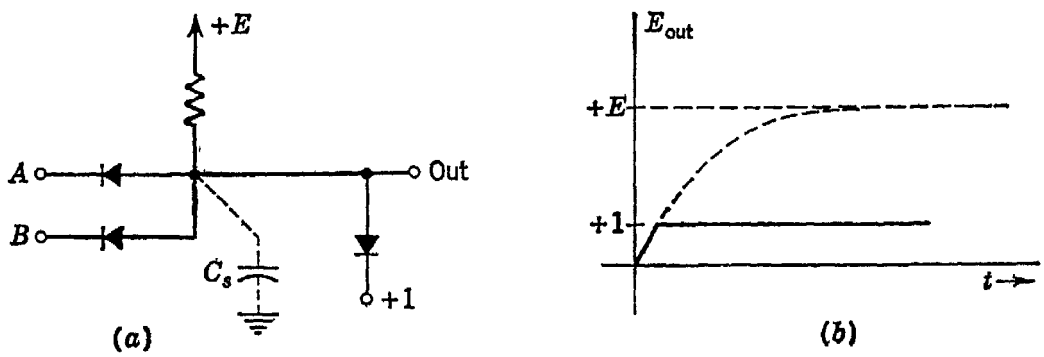



Fig. 10.5 Improved AND gate

resistor $R$ is returned is $+1$ unit, and the input signals $A$ and $B$ are assumed to be either at 0 or at $+1$. For the present the diodes are regarded as perfect; that is, they have no voltage drop while they support a forward current and no current while they support a reverse voltage. With this assumption, the only defect in the circuit is the presence of the stray capacitance $C$, which prevents the output signal from changing abruptly from 0 to 1 when both diodes have been cut off by input signals of 1, since the capacitance requires a theoretically infinite time to charge to $+1$ through resistor $R$.

A means of speeding up the rise of the output signal is to return resistor $R$ to $+E$, a much higher potential than $+1$, and to clamp the output

when it reaches +1. These modifications are shown in Fig. 10.5. When both $A$ and $B$ receive signals of +1, the output voltage rises as the stray capacitance charges toward $+E$. As soon as the output voltage reaches +1 (the return potential for the clamp diode), the clamp diode conducts and prevents further rise of output voltage. The reader may easily verify that the time of rise to +1 volt is given by

$$t = RC \ln \frac{E}{E - 1} \tag{10.1}$$

and that the initial rate of rise of the output voltage is $E/RC$ volts/sec.

When the signal is removed from any input lead, the corresponding input diode conducts and allows the stray capacitance to discharge toward 0 through the impedance of the signal source. To speed up this drop in output potential, the input leads can be returned to reference

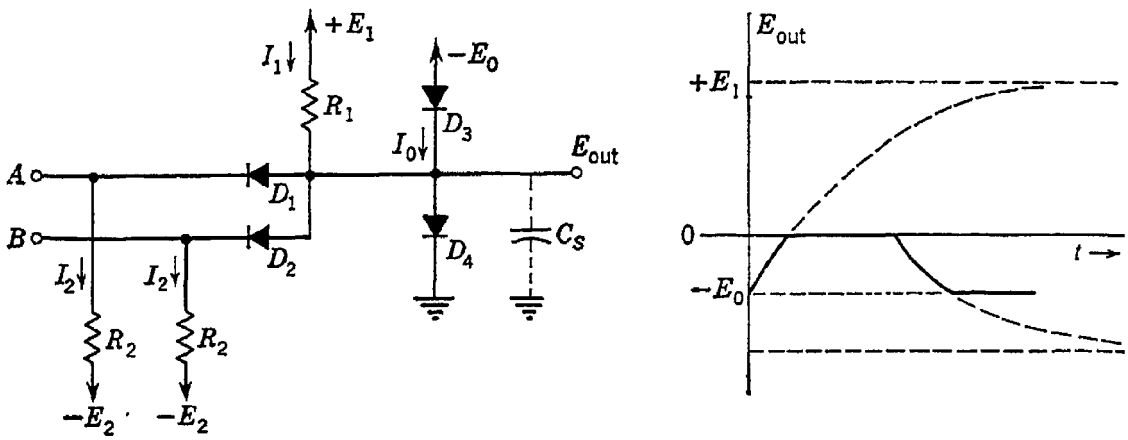



Fig. 10.6 AND gate with "pull-up" and "pull-down" circuits

levels of $-E_2$ volts, as shown in the circuit of Fig. 10.6. In this circuit the voltages and resistances are so adjusted that, in the absence of signals $A$ and $B$, diodes $D_1$, $D_2$, and $D_3$ are all conducting. $D_3$ acts as a clamp to hold the output voltage at $-E_0$ while it is conducting. If $D_1$ and $D_2$ are also conducting, then the signal input points $A$ and $B$ are also at $-E_0$ volts. Raising one of these leads above this potential cuts off the diode in that line, but, if $D_3$ still conducts, then $E_{out}$ is unchanged. In order for the circuit to function as an AND device, it is necessary that $E_{out}$ change only if all input diodes are cut off by input signals. That is, the current $I_0$ must be larger than $(n - 1)I_2$ but not so large as $nI_2$, where $n$ is the total number of input points. Then each input signal that is applied will furnish current to the corresponding resistor $R_2$, and the current in clamp diode $D_3$ will decrease by $I_2$ amp. The current $I_1$ remains unchanged as long as $D_3$ conducts, since the voltage across $R_1$ is fixed at $E_1 + E_0$ volts. Thus, if as many as $n - 1$ input diodes are cut off by the presence of signals, the current in diode $D_3$ is reduced to

$I_2 - I_1$ amp. If all $n$ diodes are cut off by input signals, then the current in $D_3$ is reduced to zero, and the active part of the circuit is simply the portion shown in Fig. 10.7. The output voltage is now given by

$$E_{out} = (E_1 + E_0)(1 - \epsilon^{-t/R_1C_S}) - E_0 \tag{10.2}$$

If diode $D_4$ is returned to ground (as shown in Fig. 10.6), $E_{out}$ can rise only as far as ground potential, since $D_4$ conducts at that point and bypasses the current from $R_1$, which would otherwise charge $C_S$. The time of rise for $E_{out}$ from $-E_0$ to 0 is found from Eq. (10.2). It is

$$t = R_1C_S \ln \frac{E_1 + E_0}{E_1} \tag{10.3}$$

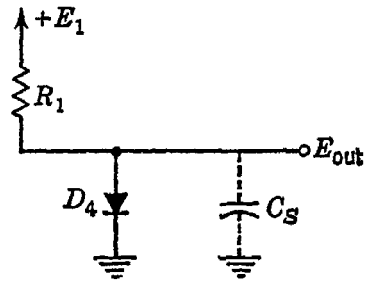



Fig. 10.7 Active portions of AND gate during pulse rise

The charging current that flows into $C_S$ during the pulse rise is $C_S\, dE_{out}/dt$ and is given by

$$i_c = \frac{E_1 + E_0}{R_1} \epsilon^{-t/R_1C_S} \tag{10.4}$$

The initial value of this current is simply $I_1$, the current flowing in resistor $R_1$ before all input diodes are cut off. A fairly good approximation to the rise time is obtained if one assumes the charging current to be constant at this initial value. This approximation yields*

$$t \cong \frac{E_0C_S}{I_1} = R_1C_S \frac{E_0}{E_1 + E_0} \tag{10.5}$$

This result is somewhat optimistic, and, if a pessimistic approximation is preferred, one can assume instead that the charging current is constant at its lowest value. This gives

$$t \cong R_1C_S \frac{E_0}{E_1} \tag{10.6}$$

Some idea of typical magnitudes may be obtained by considering a circuit in which $I_1 = 2$ ma, $-E_0 = -10$ volts, and $C_S = 20\ \mu\mu$f. Using our optimistic approximation, we find

$$t \cong \frac{10 \times 20 \times 10^{-12}}{2 \times 10^{-3}} = 10^{-7} \text{ sec} \tag{10.7}$$

If $E_1$ is taken as $+100$, then $R_1 = 55{,}000$ ohms. A precise value for $t$ is found from Eq. (10.3):

$$\begin{aligned} t &= 55 \times 10^3 \times 20 \times 10^{-12} \ln {}^{110}\!/_{100} \\ &= 1.05 \times 10^{-7} \text{ sec} \end{aligned} \tag{10.8}$$

* It is interesting to observe that this expression is also obtained by substituting for $\ln [(E_1 + E_0)/E_1]$ in Eq. (10.3) the first term of the series expansion of the logarithm.

The rate at which $E_{out}$ falls when one or more input signals are removed can be found by a similar analysis. As soon as any input signal is removed, the corresponding input diode conducts, and $C_S$ starts to discharge. The voltage $-E_2$, to which the resistors $R_2$ are returned, aids in this discharge, but it may be noted that, because $+E_1$ is connected to $C_S$ (through $R_1$), the voltage toward which $C_S$ discharges is not so negative

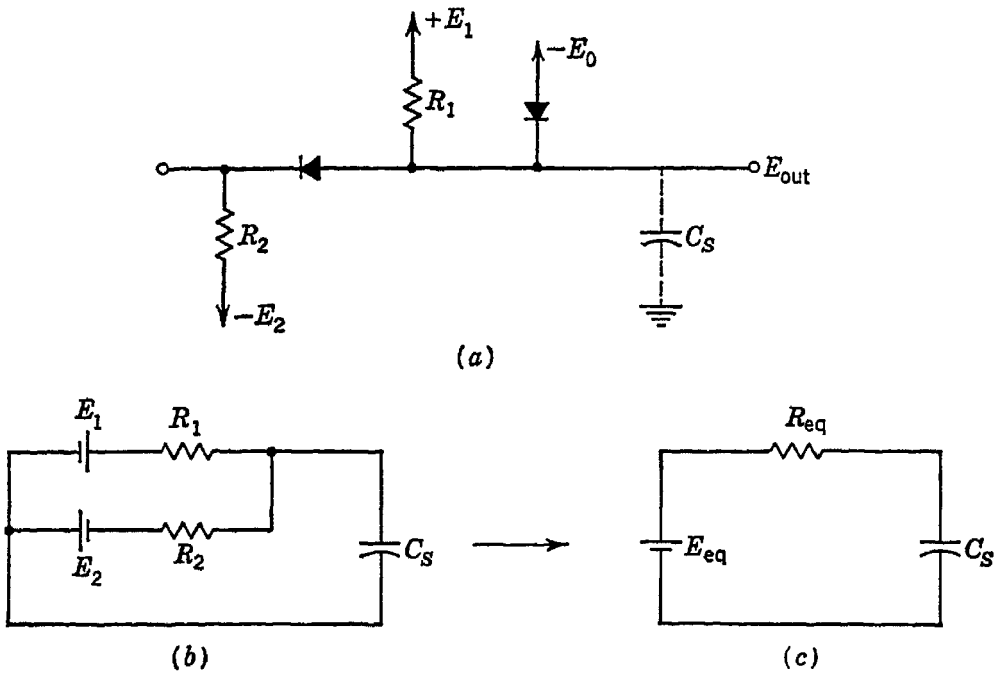



Fig. 10.8 Active portions of AND gate during pulse fall

as $-E_2$. The active portion of the circuit during the fall of $E_{out}$ is shown in Fig. 10.8, with the input terminal assumed floating.

The Thevenin equivalent resistance and voltage are

$$R_{eq} = \frac{R_1R_2}{R_1 + R_2} \tag{10.9}$$

and

$$E_{eq} = \frac{E_1R_2 - E_2R_1}{R_1 + R_2} \tag{10.10}$$

Thus, during the fall of the output voltage,

$$E_{out} = \frac{E_1R_2 - E_2R_1}{R_1 + R_2}(1 - \epsilon^{-t/R_{eq}C_S}) \tag{10.11}$$

and the time of fall to $-E_0$ volts is

$$t = \frac{R_1R_2}{R_1 + R_2} C_S \ln \frac{E_1R_2 - E_2R_1}{E_1R_2 - E_2R_1 + E_0(R_1 + R_2)} \tag{10.12}$$

We have already considered rise time with $I_1 = 2$ ma, $-E_0 = -10$ volts, $C_S = 20$ $\mu\mu$f, and $E_1 = 100$ volts. Now suppose further that

$I_2 = 3$ ma and $-E_2 = -100$. Then $R_1 = 55{,}000$ ohms and

$$R_2 = \frac{(100 - 10)}{(3 \times 10^{-3})} = 30{,}000 \text{ ohms}$$

$E_{eq}$ is found to be $-29.4$ volts, and, from Eq. (10.12), the fall time is found to be $1.61 \times 10^{-7}$ $\mu$sec. Thus the fall time is substantially greater than the rise time. Of course, the case considered was the worst, for only one input signal was removed. If several input signals had been removed, then several input diodes would have provided parallel paths through which $C_S$ could have discharged more rapidly.

Approximations of the type used in calculating rise time [Eqs. (10.5) and (10.6)] may not be valid in calculating fall time, since the voltage $E_{eq}$ toward which $C_S$ discharges is ordinarily not large compared with $E_0$. Therefore, it may not be valid to assume that the fall in potential is approximately constant at its initial rate as $E_{out}$ drops from 0 to $-E_0$ volts. The reader will find it helpful to derive an equation for the fall time that is valid for $|E_{eq}| \gg E_0$.

The larger the currents that charge and discharge $C_S$, the shorter will be the rise and fall times, respectively. The current limits are set (1) by the amount of power loss that can be tolerated in $R_1$ and $R_2$, or (2) by the maximum current that the diodes can handle without exceeding some maximum forward voltage drop and without being damaged. Call the maximum permissible diode current $I_m$. Then, since $I_2$ must exceed $I_1$ for proper circuit functioning, we must have

$$I_m > I_2 > I_1 \tag{10.13}$$

for protection of the signal input diodes. Since the maximum current in the clamp diode $D_3$ is $nI_2 - I_1$, we must also have

$$I_m > (nI_2 - I_1) \tag{10.14}$$

To ensure that the clamp diode $D_3$ will hold the output voltage down effectively, its current should exceed some minimum value, say $I_a$, when $n - 1$ input signals are applied. Therefore,

$$I_2 = I_1 + I_a \tag{10.15}$$

As a limiting case, let

$$I_m = nI_2 - I_1 \tag{10.16}$$

i.e., let $D_3$ be allowed to carry $I_m$ amp in the absence of any input signals. Then

$$nI_2 = I_m + (I_2 - I_a)$$

$$I_2 = \frac{I_m - I_a}{n - 1} \tag{10.17}$$

and
$$I_1 = \frac{I_m - nI_a}{n - 1} \tag{10.18}$$

If we insert this result in Eq. (10.5), the approximate formula for the rise time, we obtain

$$\text{Rise time } t = E_0 C_S \frac{n - 1}{I_m - nI_a} \tag{10.19}$$

Even without including the effect of added capacitance, we can see that the rise time becomes greater as more input circuits are added to the gate, if the clamp diode is presumed to carry the maximum permissible current whatever the number of input circuits. In practice, of course, the increase in rise time due to limitations imposed by restricted maximum current in the clamping diode can easily be overcome by paralleling two or more diodes at $D_3$.

**Diode OR gates.** Similar application of clamping diodes and large return potentials can be made in the OR gate, as shown in Fig. 10.9. Input diodes $D_1$ and $D_2$ are normally nonconducting. Diode $D_3$ is normally conducting, since $E_2 > E_0$. Thus, in the absence of input signals $A$ or $B$, $E_{out}$ is held at $-E_0$ volts. A signal applied to one of the input leads causes its diode to conduct, permitting $C_S$ to charge from $-E_0$ volts toward the potential of the pulsed input lead. Since the input pulse must supply current both to $R_3$ and to $C_S$, it is desirable to keep $R_3$ as large as possible in order to avoid undue loading of the signal source. On the other hand, since $R_3$ provides the path through which $C_S$ discharges at the termination of the input signal, it is desirable on this account to keep $R_3$ small. A practical design must represent a compromise between these two conflicting criteria.

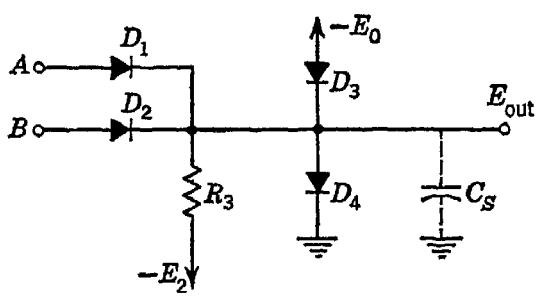


Fig. 10.9 Improved OR gate

A disadvantage of the circuit of Fig. 10.9 is that the output voltage rises only toward the potential of the pulsed input lead instead of toward a much higher "pull-up" voltage as in the AND gate. This disadvantage can be overcome by driving an OR gate only from an AND-type input, as shown in Fig. 10.10. In this circuit the OR diodes $D_5$ and $D_6$ are each driven by a circuit that is merely an AND gate with only one signal input lead. Thus, when $D_1$, for instance, is cut off by the presence of an input signal, both $C_{S1}$ and $C_{S3}$ charge toward $+E_1$ volts through resistor $R_1$. In this circuit, diode $D_3$ would be returned to a potential slightly more negative than $-E_0$, so $D_5$ would be slightly back-biased in the absence of input signals. $C_{S1}$ would charge through $R_1$ until $D_5$ started to con-

duct, and from this point on the two capacitors would charge in parallel. Diode $D_8$ performs its usual function of limiting the swing of the output.

The output voltage of the circuit in Fig. 10.10 may be expected not to rise quite so quickly as the output voltage of the simple AND gate of Fig. 10.6, since the combined capacitance $C_{S1} + C_{S3}$ is likely to be larger than the capacitance of the simple AND gate. However, this increased capacitance can be compensated for by decreasing the size of resistor $R_1$ and thus allowing greater charging current. The fall in output at the end of the input signal may be expected to be somewhat more rapid, however, since the stray capacitances can discharge through the two effectively

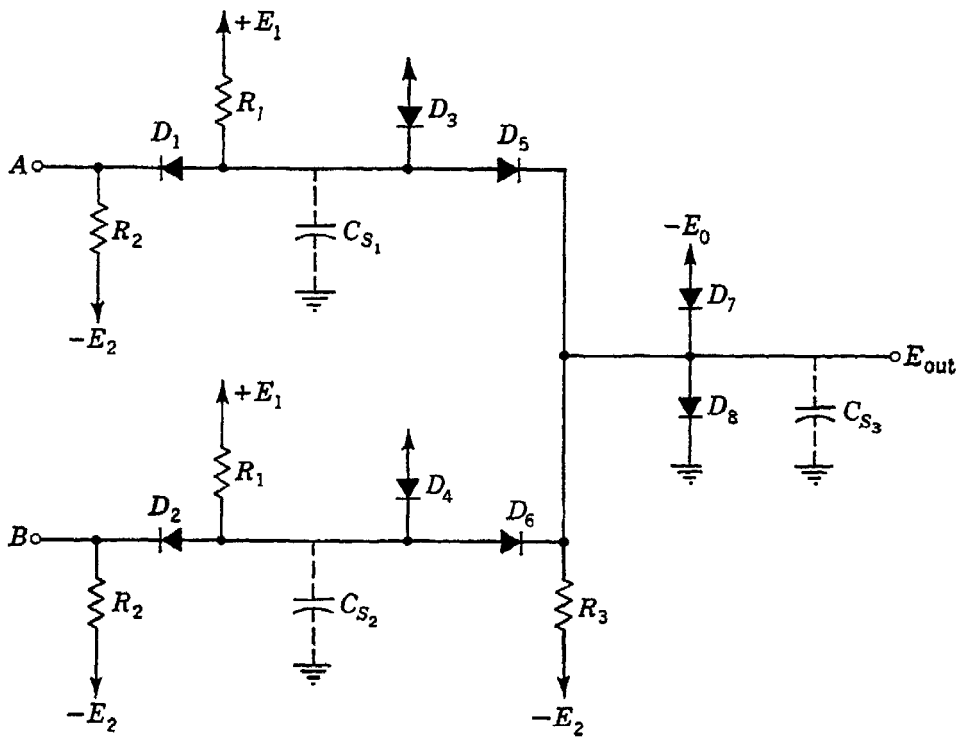



Fig. 10.10 OR gate driven by AND-type inputs

parallel paths provided by $R_2$ and $R_3$, both of which are returned to $-E_2$ volts.

**A complete circuit.** This type of circuitry has been carried to a high degree of perfection in the packaged gate circuits used in the SEAC (Bureau of Standards Eastern Automatic Computer)[1] and in many other computers built by other government and university laboratories and by commercial concerns. Figure 10.11 shows the circuit of the SEAC package. The logical configuration of each of these packages is AND followed by OR. The output from the diode logic circuit drives a vacuum-tube amplifier that is transformer-coupled, for impedance matching, to output points which drive inputs of other similar units.

It will be noted that the pull-down resistors for the input-signal leads do not appear at the inputs but are instead connected at the outputs of

each circuit. This is the function of the 1.25-kilohm resistor returned to −65 volts from the positive output lead of the package. If the logic of the over-all system is such as to require any particular circuit to drive a large number of AND diodes in other circuits, then additional resistances may be placed in parallel with the 1.25-kilohm resistor to enable the pull-down circuit to furnish this drive. The positive output lead is normally

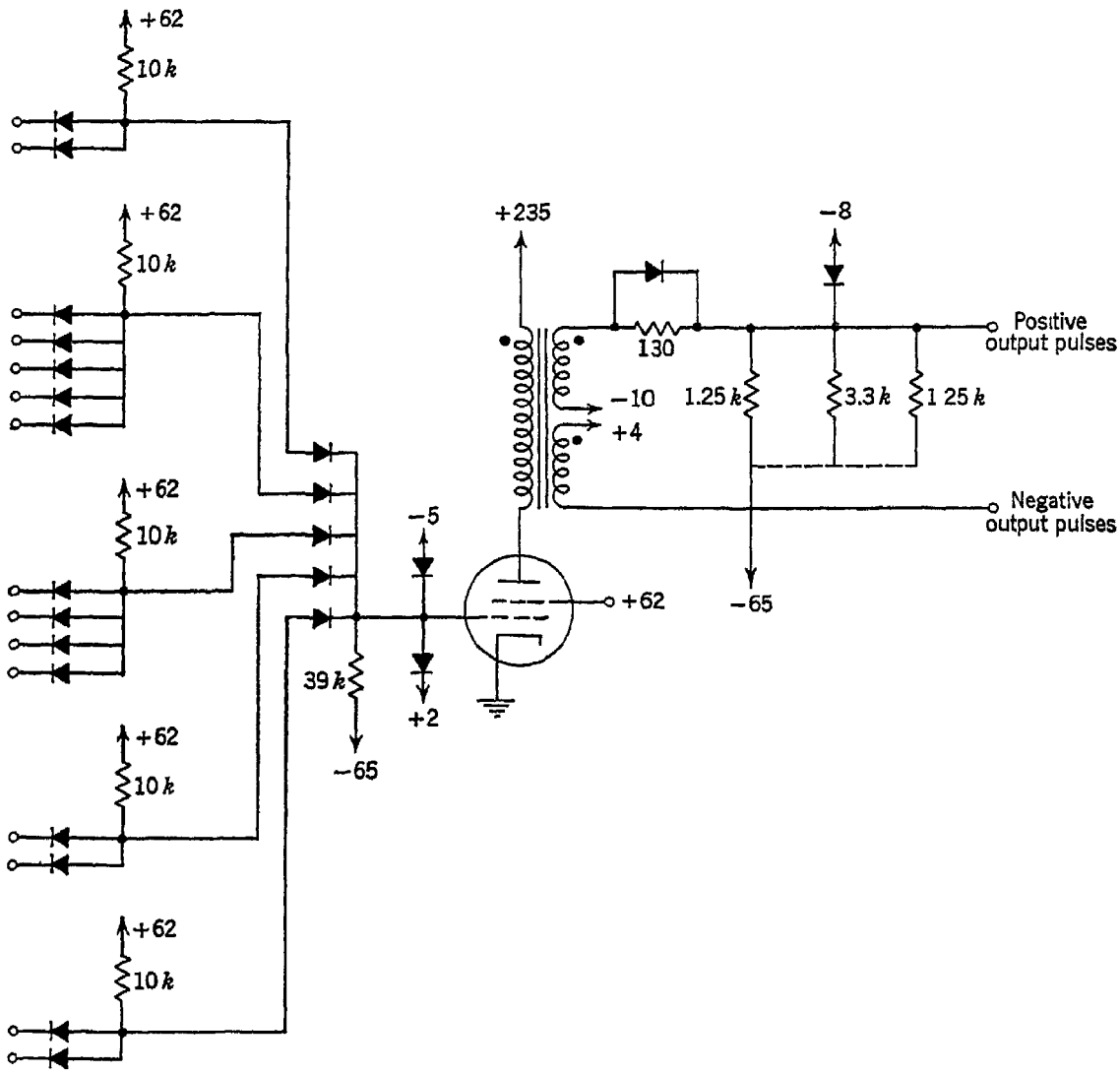



Fig. 10.11 SEAC logic element

at −8 volts, in the absence of input signals, because of the clamping diodes returned to −8. (Several diodes are usually used in parallel here to carry the heavy current.)

The action of the over-all circuit can readily be followed. Suppose three of the diodes of the five-legged AND gate of Fig. 10.11 are connected to outputs of other similar circuits. (The two unconnected diodes of this gate play no part in the circuit action.) If all three of these inputs are pulsed simultaneously, the corresponding OR diode passes a signal to the grid of the vacuum tube, which rises from −5 volts to +2. The tube

conducts heavily, and the dotted ends of the transformer windings are driven positive with respect to their undotted ends. Thus positive and negative pulses appear at the positive output point and the negative output point, respectively. (The 130-ohm resistor, which is present to damp out the backswing of transformer voltage, is shunted by diodes during this part of the cycle.) The transformer design is such that a reasonably flat-topped output pulse about 0.5 $\mu$sec in duration is obtained, although, in any case, the pulse terminates when the grid returns to $-5$ volts at the end of coincidence of signals at the AND gate input.

NOT action is obtained in this circuit by the use of the negative output pulses. In the absence of signals, the negative output lead is at a potential of $+4$ volts. If this point is connected to an AND diode, that diode is normally nonconducting and plays no part in the action of the AND circuit. However, a pulse at this negative output lead drives it down to a potential of perhaps $-6$ volts and causes the AND diode that is connected to conduct. When one of the AND diodes is made to conduct by the presence of a negative signal, the cutting off of the other AND diodes by positive signals cannot make the AND output rise. The negative pulse thus inhibits the action of the AND gate. In the circuits of Chap. 9 this inhibiting signal is indicated by the small circle around one of the input leads to an AND gate.

**Timing problems.** In any logic circuitry using pulses, a problem is encountered when the pulses are not in exact time synchronism with one

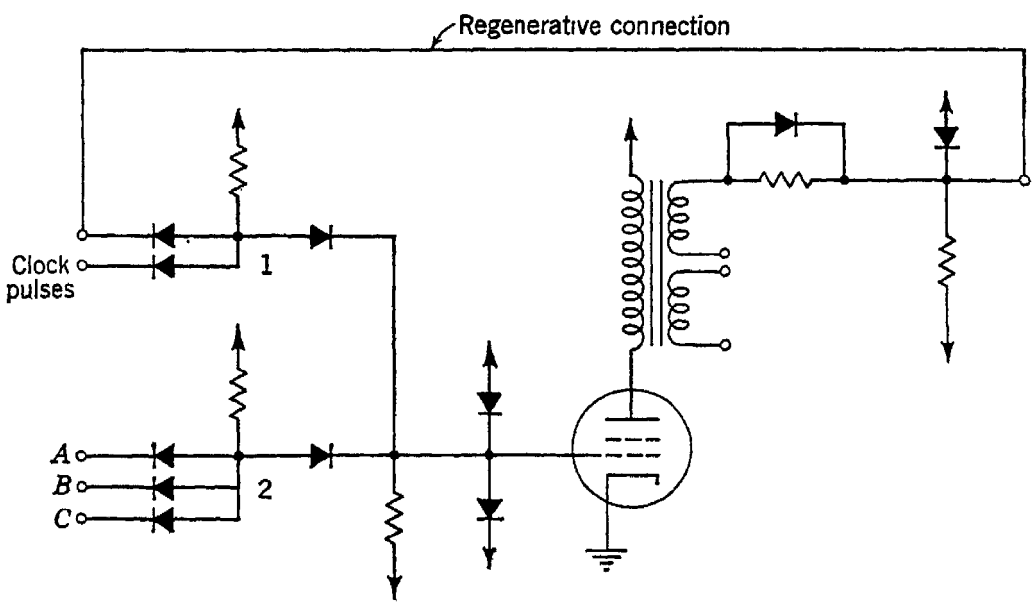



Fig. 10.12 Circuit with regenerative broadening

another at the input to an AND element. Two means are used to meet this difficulty. (1) One input to each AND gate is derived from a master synchronizing-pulse source, or "clock". (2) A technique known as "regenerative broadening" is used to ensure that the output pulse is a

replica of the clock pulse. This method is illustrated in Figs. 10.12 and 10.13.

If the $A$ signal, the $B$ signal, and the clock pulse are slightly out of phase with one another, as shown in Fig. 10.13, then the AND output will be high only during the narrow overlap of these signals. This narrowing cannot be tolerated, since a narrow signal from one gate might totally miss another narrow pulse with which it is gated. The remedy lies in the use of the regenerative connection. If $A$, $B$, and the clock overlap, then the tube conducts, and the output pulse is applied to gate 1 to be gated against the clock pulse. Both gate 1 and gate 2 are now acting to keep the tube in conduction, so that even when gate 2 goes off at the end of the overlap of $A$, $B$, and the clock, gate 1 keeps the grid of the tube high until the end of the clock pulse. If the region of overlap of $A$ and $B$ pulses overlaps the beginning of the clock pulse, the grid is driven positive by gate 2 at the start of the clock pulse and is driven negative when gate 1 is turned off at the end of the clock pulse. Thus a full-length pulse is applied to the grid, and a full-length pulse is produced at the pulse-transformer output.

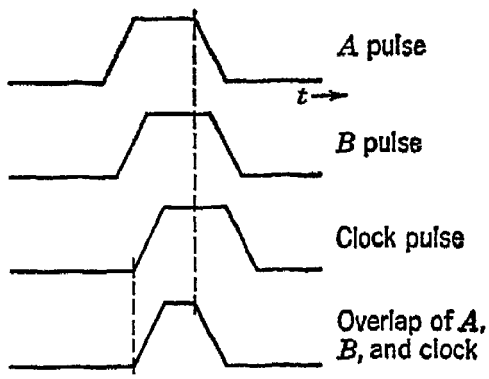



Fig. 10.13 Narrowing of output pulse when input pulses do not coincide

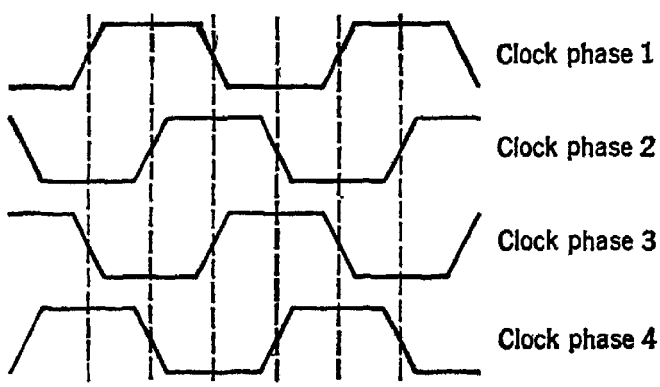



Fig. 10.14 A four-phase clock

Although full-length pulses are produced, there is a slight time delay, so that the output pulse lags the input pulse by a small amount. In the SEAC circuit, the pulses occur at a rate of 1 megacycle. Each pulse has a nominal duration of 0.5 $\mu$sec. The time delay in each package unit is in the vicinity of 0.15 $\mu$sec. This delay time consists principally of the time required to charge up the stray capacitances and the time required to build up the field in the pulse transformer. Since the output lags the

input, it is apparent that the same clock pulse that timed the input cannot be gated with the output in a subsequent circuit. Instead, one of the succeeding clock pulses must be used. A common way of minimizing the waiting time until another clock pulse arrives is to generate clock pulses in four phases, as shown in Fig. 10.14. If the basic clock pulse period is 1 $\mu$sec, clock phases 2, 3, and 4 produce their pulses $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$ $\mu$sec after clock phase 1, respectively. Thus, if the input signals to a gate are timed by pulses from clock phase 1, the output signals can be timed at the next gate by pulses from clock phase 2, and so on. Without these separate clock phases, the output from a gate circuit would have to be delayed the rest of a full clock period before being applied to another gate input. Systems using single-phase clocks are somewhat slower, therefore, than systems using several clock phases, but the timing problems are not so thorny.

**Delay lines.** The temporary-storage element used with dynamic circuitry is the electrical delay line. Such lines are used to perform two functions:

1. The logical function of storing information until needed. In this case the lines are used as the storage elements in a sequential switching system.
2. The electrical function of delaying a signal either to put it in synchronism with a signal that lags it or to allow formation of a long pulse.

An example of the use of a delay line in a purely logical manner is the flip-flop circuit shown in block-diagram form in Fig. 10.15. This diagram represents a circuit like that shown in Fig. 10.11 but with its positive output terminal connected back to one of its AND inputs through a delay line. Clock pulse inputs to the AND gates are implicit in the diagram, as are the vacuum tube and the pulse transformer. Since the output pulses from the gate circuit are slightly delayed with respect to the input pulses, the delay line should have a delay of less than one clock period in order to make the total delay around the loop equal to exactly one clock period.

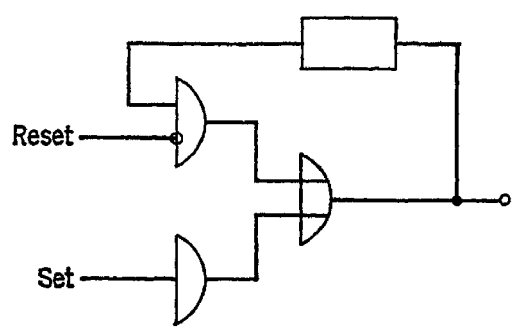


Fig. 10.15 The dynamic flip-flop

An example of the purely electrical use of delay lines is the formation of long pulses for inhibiting an AND gate. If an inhibiting pulse is to be successful in blocking the action of an AND gate, it must commence slightly before the signals it is to inhibit and end slightly after them. Then there will be no possibility that a narrow spike might slip through the AND gate and give an erroneous output. However, since the negative output pulses, which are used for inhibition, are of the same duration as

the positive pulses, these negative pulses must be stretched. This is done by applying the inhibiting pulse directly to one leg of the AND gate it is to inhibit and indirectly, through a delay line, to another leg of the same gate, as shown in Fig. 10.16. Then diode 3 conducts for a full pulse period, and diode 4 also conducts for a full pulse period, starting before the end of conduction in diode 3. Thus signals $A$ and $B$ can be initiated any time within the interval of conduction of diode 3 without causing a signal to appear at the AND-gate output.

Before we leave the subject of delay lines, some brief comments are in order in regard to the electrical characteristics and engineering problems encountered in delay lines. Viewed in its simplest light, the electrical problem of designing a delay line is that of designing a transmission system with suitable impedance characteristics and with acceptable distortion and attenuation. In any transmission system distortion can be

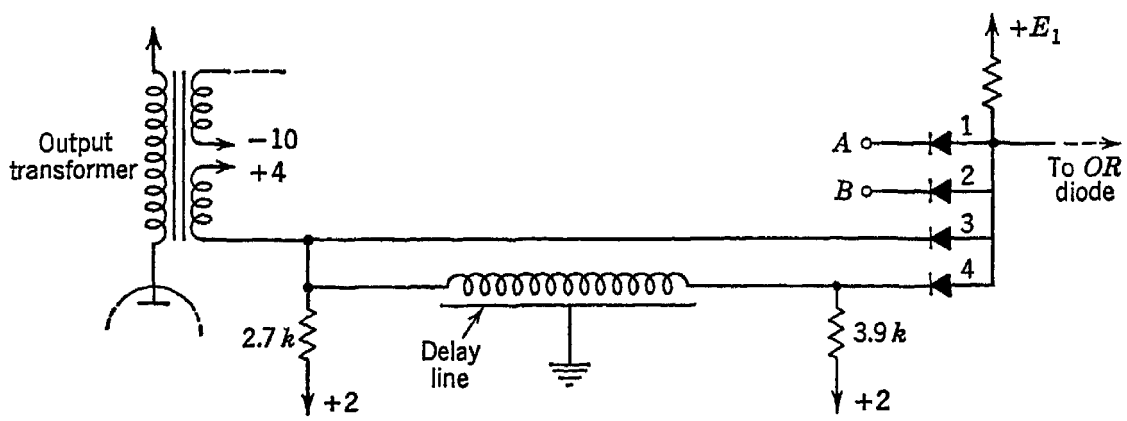



Fig. 10.16 Use of delay line to stretch an inhibiting pulse

avoided by making the amplitude response be constant as a function of frequency and by making the phase shift versus frequency curve have a constant slope. The effect produced by change of amplitude with frequency is called amplitude distortion; change of the phase-shift curve from constant slope is called phase distortion, or, sometimes, delay distortion. A square pulse transmitted through a system with only amplitude distortion will be symmetrically distorted about its mid-point in time, since its component sine waves are all in phase and undergo only changes of amplitude relative to one another but no changes in phase. Phase distortion, however, produces an asymmetry, since the high- and low-frequency components are delayed unequally. The output may be oscillatory followed by a rounded pulse, or a rounded pulse followed by an oscillatory portion, depending on whether the high-frequency components arrive first or last.

Electrical delay lines may be either the distributed-parameter type or the lumped-constant type. The lumped-constant delay line is simply a

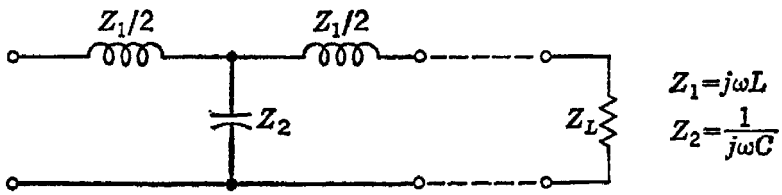



Fig. 10.17 Lumped-constant delay-line section

low-pass filter made up of sections such as that shown in Fig. 10.17. The characteristic impedance $Z_0$ of a line composed of such sections is given by

$$Z_0 = \sqrt{Z_1 Z_2 + \frac{Z_1^2}{4}}$$
$$= \sqrt{\frac{L}{C}\left(1 - \frac{\omega^2 LC}{4}\right)} \tag{10-20}$$

Cutoff occurs at the frequency at which $Z_0$ ceases to be resistive and becomes reactive, i.e., at $\omega_0 = 2/\sqrt{LC}$. Thus we may write

$$Z_0 = \sqrt{\frac{L}{C}}\sqrt{1 - \left(\frac{\omega}{\omega_0}\right)^2} \tag{10.21}$$

Figure 10.18 shows a plot of $Z_0$ versus $\omega$, from which it is evident that a resistance $R = \sqrt{L/C}$ is a matched termination only at frequencies $\omega \ll \omega_0$.

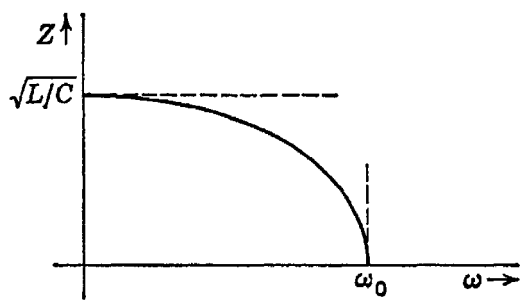



Fig. 10.18 $Z_0$ versus $\omega$ for filter section of Fig. 10.17

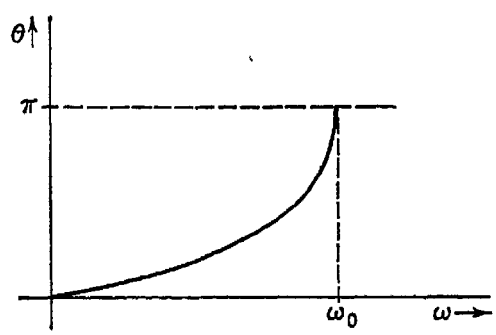



Fig. 10.19 Phase shift $\theta$ versus $\omega$ for filter section of Fig. 10.17

For such a line terminated in its characteristic impedance, the phase shift per section is easily shown to be

$$\theta = 2 \sin^{-1} \frac{\omega}{\omega_0} \tag{10.22}$$

A sketch of this function is shown in Fig. 10.19. It is evident from the shape of these two curves that only for $\omega \ll \omega_0$ does this delay section show a constant-amplitude response and a phase shift proportional to frequency.

Although complete design of a delay line is an advanced problem in

filter theory, it is possible nevertheless to make some approximations that indicate typical magnitudes. Suppose it is desired to transmit a pulse with a rise time of 0.1 $\mu$sec. A fairly good rule of thumb indicates that $1/T_r = 10^7$ cycles should be adequate bandwidth. Since the delay time for any frequency component is $d\theta/d\omega$, we have

$$\text{Delay time } T_d = \frac{2}{\omega_0} \frac{1}{\sqrt{1 - (\omega/\omega_0)^2}} \tag{10.23}$$

Evidently the delay increases rapidly as $\omega \rightarrow \omega_0$; so components in this range arrive too late to assist in the initial rise of the output voltage. Since we have required frequency components up to 10 megacycles in the output, these must be in the low-frequency part of the passband if they are to help in the rise of the pulse. Let us, therefore, take

$$\omega_0 = 2\pi \times 10^8 \text{ radians/sec } (f_0 = 100 \text{ mc})$$

Then

$$\omega_0 = \frac{2}{\sqrt{LC}} \qquad \text{and} \qquad LC \cong 10^{-17} \tag{10.24}$$

If we now choose $Z_0 = 100$ ohms, we find that $C = 31.6$ $\mu\mu$f and $L = 0.316$ $\mu$h. For low frequencies the delay is given approximately by $2/\omega_0$, which for this line is found to be about 0.003 $\mu$sec. Thus a delay of 0.15 $\mu$sec would require about 50 such sections.

Lumped-constant delay lines are useful where the characteristic impedance must be made low, where the voltage is high, or where simplicity of construction is important. They usually have lower attenuation than distributed lines, because a lower-loss dielectric can be used in the capacitors.

The distributed-constant delay line is also a low-pass filter. In one typical construction, a solenoid having the desired inductance is wound over an insulating form, then covered with an insulating layer, and then finally wrapped with a conducting layer that is grounded. Such lines often have considerable delay distortion, however, since the effective inductance drops with increasing frequency. This is because flux linkages are not in phase along the line when the wavelength becomes comparable to the line length. The phase characteristic can be equalized by making the effective capacitance increase with frequency to offset the drop in inductance. Such an effect is obtained by including an isolated conducting strip in the line in addition to the grounded conductor. Since the strip is isolated, the net current flow to it is zero. Therefore, at low frequencies where there is negligible phase difference along the portion of line opposite a given strip, there will be no current flow from any turn of the coil to the strip although there is capacitive current flow to the

grounded conductor. At high frequencies, however, the additional strip acts like a grounded conductor, and its capacitance adds to that of the grounded conductor, thus raising the effective capacitance of the system.

The delay lines used in most present-day dynamic circuitry have an attenuation characteristic such that delays of more than four or five clock pulse periods cannot be safely obtained without interposing a gating and amplifying package to restore the pulse shape. Thus, for example, in the circuits of MIC discussed in Chap. 9, the multiplier loop and the accumulator loop were both made up of 5-$\mu$sec delay sections with gating packages intervening.

## 10.4 Static circuitry using vacuum tubes

The concepts of dynamic circuitry discussed in the previous section contrast sharply with those of static circuitry. The most obvious difference, of course, is that, in dynamic circuitry, information is stored as a circulating pulse in a loop consisting of a delay line, an amplifier tube, and a pulse transformer, whereas, in static circuitry, information is stored as a voltage level in an Eccles-Jordan flip-flop. However, merely stating this difference gives a very incomplete picture of the differences in computer circuitry that result from use of these two systems, and some further comments upon them are in order.

Dynamic circuitry, by virtue of the fact that information is both stored and transmitted in pulse form, must be used in the synchronous mode. Since the dynamic flip-flop circulates a pulse around a loop, this pulse can be detected and used only as it passes the read-out point of the loop. Static circuitry, on the other hand, can be used either synchronously or asynchronously, since the voltage level of a flip-flop is continuously available for read-out. When the static flip-flop is used as the basic information-storing element of a digital system operating synchronously, it is common practice to let the transmission of information from one circuit to another be by means of pulses rather than by voltage levels and to use a transformer or a capacitor as the coupling element. However, direct coupling can be used with synchronous static circuitry and is the usual method of coupling for asynchronous static circuitry.

Some methods of interconnecting static circuitry are shown in Fig. 10.20. Figure 10.20*a* shows a static flip-flop supplying a d-c voltage level to one leg of a diode gate. (The other legs would be similarly excited.) The pull-up voltage for the gate is a clock pulse, and the output signal, in this case capacitively coupled, is also obtained as a pulse. Figure 10.20*b* shows a similar circuit in which the output is coupled through a transformer. Both these systems have the advantage, as compared with the use of a fixed voltage for the pull-up, of requiring less power. Further-

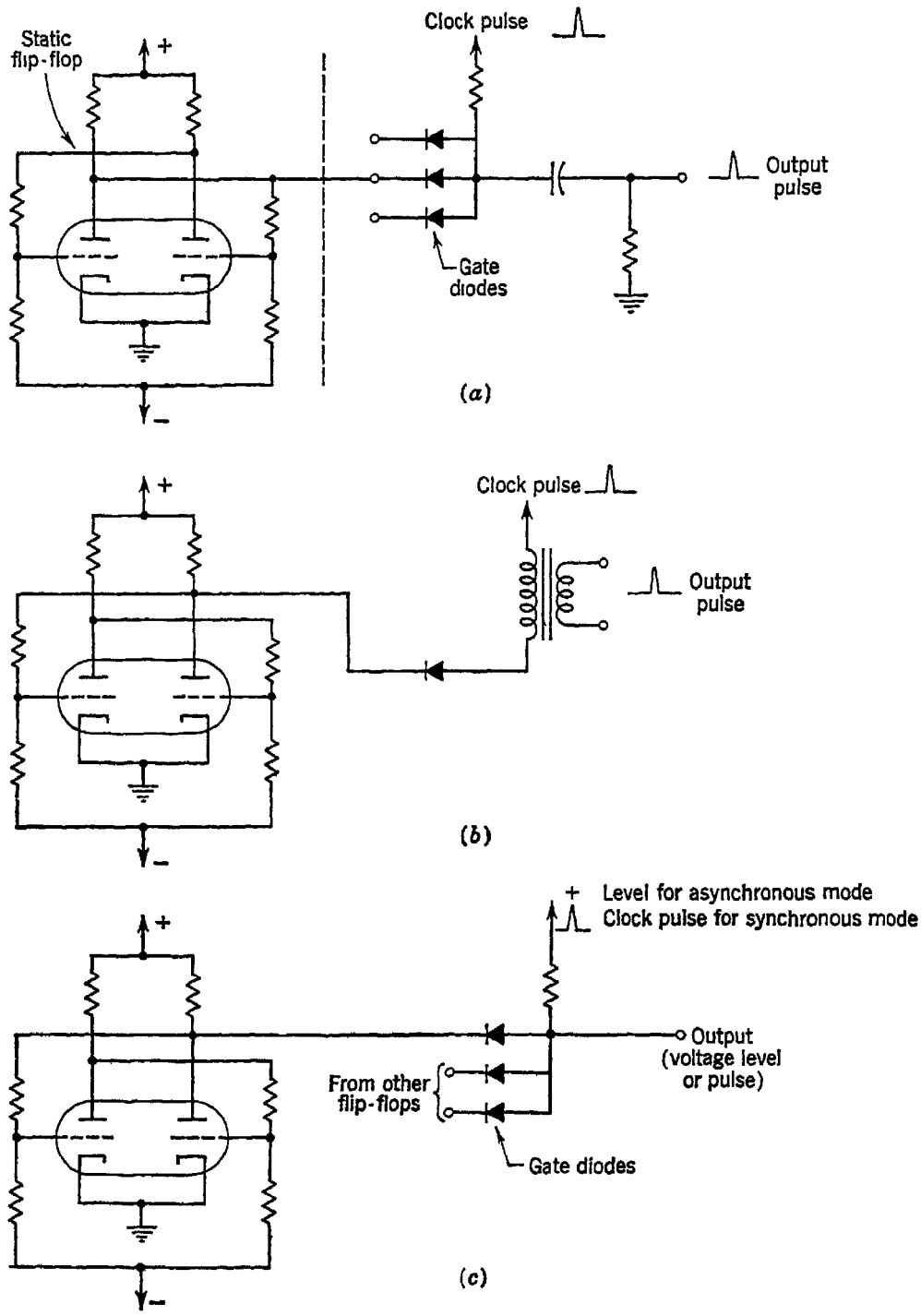



Fig. 10.20 Interconnections of static circuits: (*a*) an example of synchronous-static circuitry with capacitive coupling, (*b*) an example of synchronous static circuitry with transformer coupling, (*c*) an example of direct-coupled static circuitry, either synchronous or asynchronous

more, the use of transformer coupling allows better matching of the circuit to its load. Figure 10.20*c* shows a similar circuit which is direct-coupled. Use of a fixed level for the pull-up voltage of the diode gate allows the circuitry to be used asynchronously; a clock pulse applied at the same point would result in a synchronous circuit.

Another contrast between dynamic and static circuitry lies in the method of obtaining an impedance match between the high impedance of the vacuum tube and the comparatively low impedance of the load circuit. In dynamic circuitry this matching is accomplished by the use of a pulse transformer, and the tube is thus able to work into a well-matched load even under conditions in which the required output power may be comparatively large. Direct-coupled static circuitry cannot make use of matching transformers, however, and with this type of circuitry the impedance matching, whenever it is necessary, is accomplished by the use of cathode followers. Of course, cathode followers to supply the load power are not always needed, since some flip-flop circuits in any given system are only lightly loaded. In such cases it is often possible to design the flip-flop so that it will still flip reliably and rapidly even when it is driving an asymmetrical load, provided the load is sufficiently small.

The fact that all elements of a dynamic circuitry system operate in synchronism with a basic clock signal has had the effect that in most such systems only one basic circuit element is used throughout the entire control and arithmetic portions of the system. An example of such a circuit unit is the SEAC circuit package shown in Fig. 10.11. Indeed, it is even possible (although impractical) to build the entire computer, except for input and output devices, from such a basic circuit element.[2] Several machines have been built that employ the same dynamic circuit package through most of the arithmetic and control circuitry, the different circuit configurations being achieved by appropriate choices of input diodes and by suitable interconnection of the elements, as shown, for example, in the control circuitry of the MIC in Chap. 9.

Asynchronous static circuitry, on the other hand, is not required to fit into such a rigid timing pattern as that imposed upon dynamic circuitry; as a consequence it is common practice to use many different kinds of electronic circuits. Each circuit type is designed to do a specific task, and the asynchronous static system often includes many circuit devices in addition to the flip-flops and cathode followers already mentioned. One typical list of circuit elements includes such devices as complementing flip-flops, noncomplementing flip-flops, flip-flops with tapped outputs, desensitized flip-flops for actuation by mechanical switches, inverter circuits, cathode followers, diode gates, and thyratron units. The variety of these devices available to the designer points up at once the difficulty of carrying out a straightforward design technique that leads from Boolean expressions directly to electronic circuits. Such a procedure is more directly applicable to dynamic circuitry simply because of the uniformity of the circuit elements. However, it can reasonably be argued that the designer of an asynchronous static system is in an

advantageous position because the existence of many different special-purpose circuit elements allows him to obtain a more efficient design by using the optimum element at each point in the system.

A final point of comparison between dynamic circuitry and asynchronous static circuitry is the matter of relative operating speeds. Of course, as better components are developed, both types of circuitry will be made to operate faster and faster. However, from a philosophical point of view, it appears that the speed advantage will always lie with the asynchronous system, since each operation can take place as soon as the preceding operation has been completed, whereas in synchronous circuitry each stage must await the next clock pulse before commencing its action. Although the difference may appear trivial, it is interesting to note that all the highest-speed computers under development in the United States (as of 1958) use some form of asynchronous circuitry.

## 10.5 Types of vacuum-tube flip-flops

The basic circuit for the Eccles-Jordan flip-flop is shown in Fig. 10.21. The behavior of this circuit depends upon the nature of the impedances $Z_1$ and $Z_2$, which determines whether the circuit will be astable, monostable, or bistable. The bistable circuit has two stable states: (1) tube $A$ conducting and $B$ cut off, and (2) tube $B$ conducting and $A$ cut off. A single input pulse (for example, a negative pulse applied to the grid of the conducting tube) is required to shift the circuit from one stable state to another, and two pulses carry the circuit through a complete cycle. The circuit will be bistable only if both $Z_1$ and $Z_2$ provide through paths for direct current. Typically, $Z_1$ and $Z_2$ are identical resistances or identical parallel $RC$ branches.

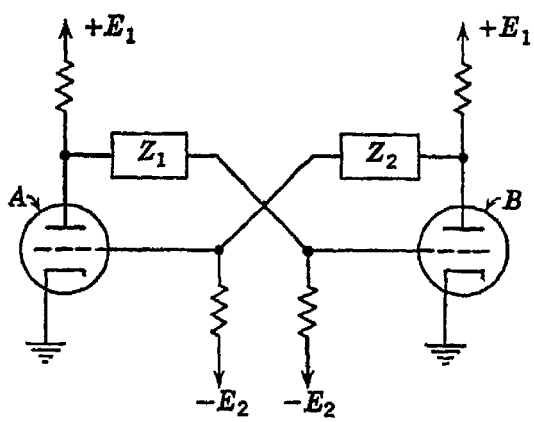


Fig. 10.21 Generalized Eccles-Jordan circuit

The monostable circuit has only one stable state and always returns to this state after having been triggered temporarily to the other. In order for the circuit to be monostable, it is necessary that either $Z_1$ or $Z_2$, but not both, should have a through path for direct current. For example, if $Z_1$ is a parallel $RC$ circuit and $Z_2$ is a pure capacitance, the quiescent grid voltage of tube $A$ is $-E_2$ volts. If $-E_2$ is a sufficiently negative voltage, then tube $A$ is cut off and $B$ conducts. One way in which the states of conduction may be temporarily interchanged, how-

ever, is by applying a short-duration negative pulse to the grid of tube $B$ to cut it off. The rise in its plate potential causes a charging current to flow through capacitor $Z_2$ and the resistor connected to the grid of $A$, thus raising the grid potential of $A$ and allowing it to conduct. However, as the charging current decays, the grid potential of $A$ falls, and $A$ will continue to conduct only as long as its grid potential remains above the cutoff potential. Thus each input pulse causes the monostable circuit to execute a complete cycle. The duration of the cycle is controlled principally by the time constant of the capacitive element $Z_2$ and the resistive path in series with it.

In contrast with these devices, the astable circuit has no d-c path in either $Z_1$ or $Z_2$ and oscillates from one unstable condition to the other at a rate determined largely by the time constants of the charging paths of the capacitors. No input pulses are required to shift this circuit from one state to the other.

## 10.6 Computer uses of flip-flops

Of the three types of flip-flop elements, the bistable circuit is by far the most useful in the digital computer. The well-designed bistable flip-flop constitutes a high-speed, rapid-access, 1-bit storage device. It may be used simply for the retention of information until it is needed, or it may be used as a trigger circuit, exerting on-off control over other circuits in accordance with the information it is storing. The use of the circuit determines its design, since input and output conditions vary in different applications. For example, as a counter the circuit requires symmetrical input pulses, but as a trigger or as a stage in a shift register it requires separate inputs for information pulses and shift pulses, as shown in Fig. 10.22. The diodes in these circuits serve both to isolate the flip-flop from its drivers and load and to steer the pulses to the proper points. Thus, in the circuit of Fig. 10.22*a*, the input diodes ensure that the flip-flop is not connected to the source of pulses except during the application of a pulse, and they also direct the pulse to the more positive plate to start the flipping action. (Without the diodes, the pulse would be applied to both the high plate and the low plate at the same time, thus making flipping much less certain.)

The monostable flip-flop, or "one-shot multivibrator", is occasionally useful in computer circuitry. It can be used as a delay element for the short-time storage of 1 bit or as a trigger circuit of fixed duration. A typical circuit is shown in Fig. 10.23. This circuit is somewhat different from that of Fig. 10.21, since the grid of tube $A$ has a pull-up resistor returned to the positive supply voltage instead of a pull-down returned to the negative supply. Thus tube $A$ is normally conducting and tube $B$ is

off. A negative-going input pulse of sufficient magnitude turns $A$ off and $B$ on. As the capacitor at the plate of $B$ charges through the ½-megohm resistor, grid $A$ rises above the cutoff voltage, thus allowing $A$ to conduct again and cutting off $B$. While $B$ conducts, input pulses have no effect upon the circuit. During this interval, either the low

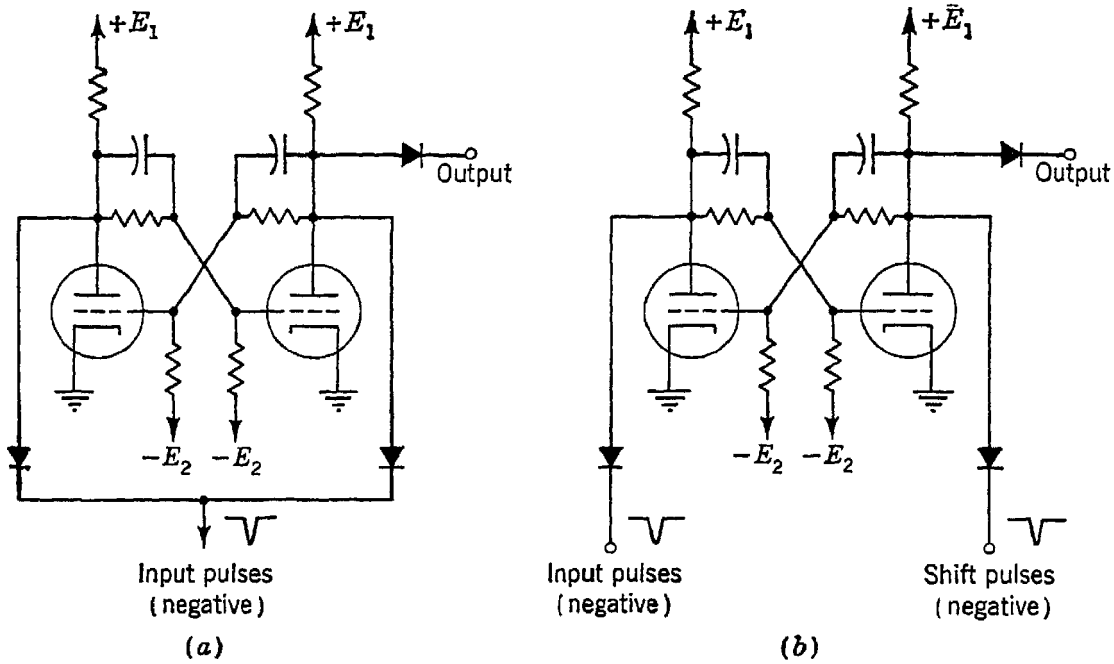



Fig. 10.22 Two flip-flop applications: (a) counter stage, (b) shift-register stage

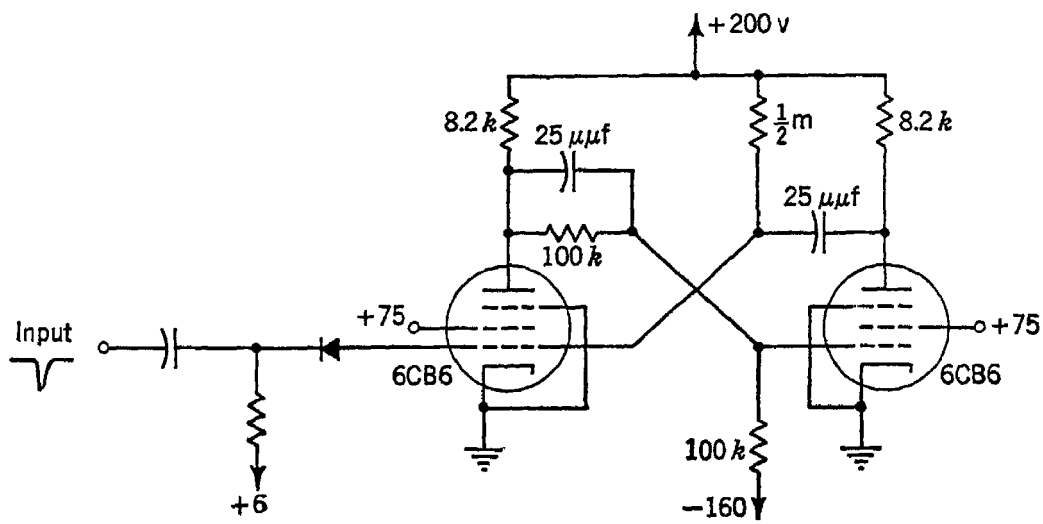



Fig. 10.23 A monostable flip-flop

potential of plate $B$ or the high potential of plate $A$ can be used as a control over another circuit.

In the astable flip-flop, the impedances $Z_1$ and $Z_2$ are pure capacitances. Their charging currents, flowing through the pull-down resistors connected to the two grids, are continually changing, and the grid potentials therefore cannot reach steady values. The astable flip-flop is not of

much use to the computer engineer, since its state is so largely a function of its own parameters and so little a function of any external information signals. The circuit does make a useful frequency divider, since it can be synchronized at a submultiple of the frequency of an external wave, but this function is more useful in such fields as television and radar than in computer design.

## 10.7 Circuit design considerations for flip-flops

There are three problems in the design of a satisfactory flip-flop:

1. The d-c problem—that of choosing resistors, voltages, and tolerances in such a way as to minimize the effect of changes in the tube characteristics and to maintain the stability of the circuit
2. The switching problem—that of choosing the circuit, the operating points, the component sizes, and the layout in such a way as to minimize the duration of the transients as the circuit switches from one state to the other
3. The loading problem—that of choosing circuit constants such that loading will have a minimum effect on d-c stability and on switching speed

These problems are usually attacked in the order listed, although they cannot be regarded as independent. For instance, from the d-c point of view (problem 1), the plate load resistors should be as large as possible to minimize power dissipation, but, from the switching point of view (problem 2), they should be small in order to maximize charging currents and switching speed. Again, minimum dissipation requires small voltages, but for fast waveforms large voltages should be used, especially if the circuit uses clamping diodes to limit voltage swings.

A useful method of solving the d-c design problem has been devised by Rubinoff[3] and by Ritchie.[4] To present the essential features of this method, we may start by referring to Fig. 10.24. The voltage divider connected to the conducting plate is designated the $C$ divider; that connected to the nonconducting plate is labeled the $NC$ divider. The design procedure is based upon the premise that the grid voltage of the tube that is cut off should be held below the cutoff potential by some minimum amount, while the grid voltage of the conducting tube should be approximately at the level of the cathode. Most tubes have their maximum transconductance when the grid is at the cathode potential (or even slightly positive), and designing for this condition ensures that the plate current will change at the most rapid possible rate in response to a triggering signal at the grid. Any conditions that tend to raise the grid voltage of the nonconducting tube or to lower the grid voltage of the conducting

tube lead toward instability, the tendency of the flip-flop to change state without being triggered. For the two voltage dividers of Fig. 10.24, the conditions that lead toward instability are as follows:

| *C divider* | *NC divider* |
|---|---|
| $E_1$ high | $E_1$ low |
| $E_2$ low | $E_2$ high |
| $R_1$ low | $R_1$ high |
| $R_2$ low | $R_2$ high |
| $R_3$ high | $R_3$ low |

These conditions allow us to set up two of the equations needed. Let $\sigma$ be the fractional variation to be allowed for the supply voltages and let $\rho$ be the fractional change allowed for the resistances. Let us require that the open-circuit grid voltage of the conducting tube be at least as positive

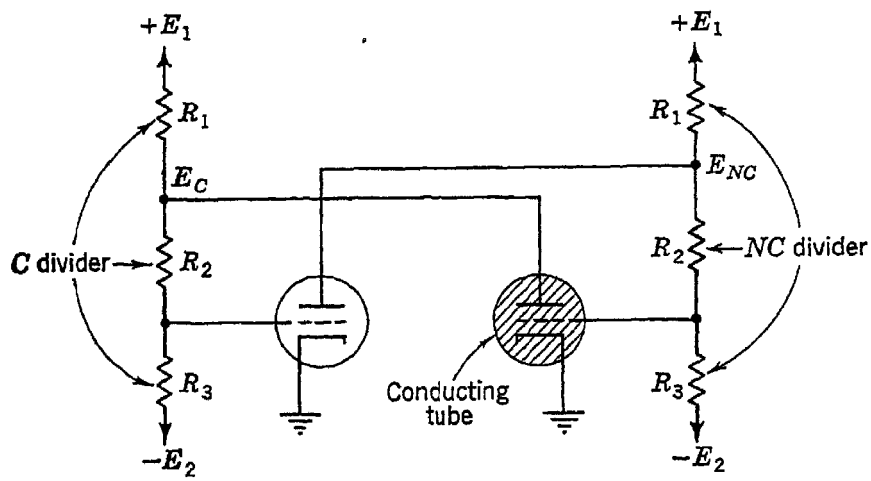



Fig. 10.24 Flip-flop, showing symbols used in d-c design method

as $+e_1$ volts and that the grid voltage of the nonconducting tube be at least as negative as $-e_2$ volts, where $e_1$ and $e_2$ are determined by the type of tube used. Furthermore, we shall require that, as the tubes age, the minimum permissible plate current $I_b$ shall be drawn with a plate voltage of no more than $E_b$ volts. From these conditions the reader can easily derive Eqs. (10.25) and (10.26):

$$e_{g_C} = \frac{E_1R_3(1-\sigma)(1-\rho) - E_2(R_1+R_2)(1+\sigma)(1+\rho)}{R_3(1-\rho) + (R_1+R_2)(1+\rho)} \geq e_1 \qquad (10.25)$$

and

$$e_{g_{NC}} = \frac{E_bR_3(1+\rho) - E_2R_2(1-\sigma)(1-\rho)}{R_2(1-\rho) + R_3(1+\rho)} \leq -e_2 \qquad (10.26)$$

These equations can be rearranged into the forms of Eqs. (10.27) and (10.28):

$$L_1 = a_1\frac{R_3}{R_1} + b_1\frac{R_2}{R_1} + b_1 \geq 0 \qquad (10.27)$$

$$L_2 = a_2\frac{R_3}{R_1} + b_2\frac{R_2}{R_1} \leq 0 \qquad (10.28)$$

The constants $a_1$, $a_2$, $b_1$, and $b_2$ are

$$a_1 = E_1(1 - \sigma)(1 - \rho) - e_1(1 - \rho) \tag{10.29}$$
$$a_2 = (E_b + e_2)(1 + \rho) \tag{10.30}$$
$$b_1 = -E_2(1 + \rho)(1 + \sigma) - e_1(1 + \rho) \tag{10.31}$$
$$b_2 = -E_2(1 - \sigma)(1 - \rho) + e_2(1 - \rho) \tag{10.32}$$

Equations (10.27) and (10.28) specify regions in the $(R_2/R_1, R_3/R_1)$ plane that must overlap, as shown in Fig. 10.25, if a physically realizable solu-

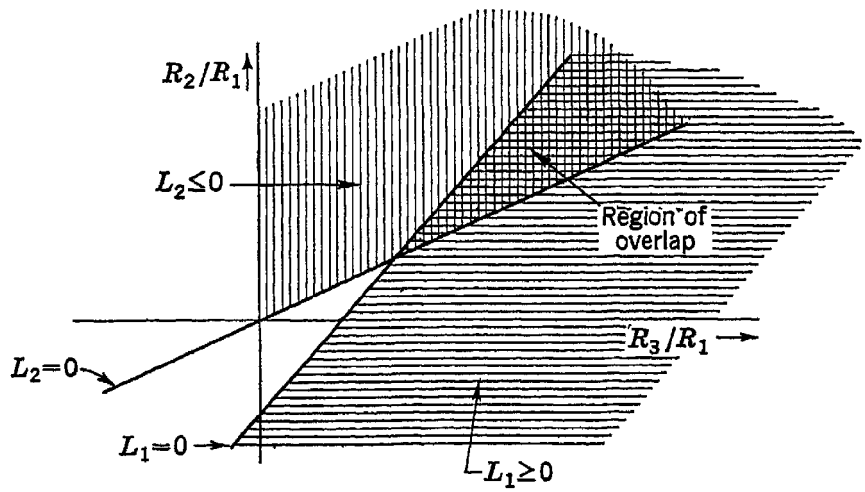



Fig. 10.25 Regions specified by Eqs. (10.27) and (10.28)

tion is to be obtained. Overlap occurs only if the slope of the line $L_1 = 0$ exceeds that of the line $L_2 = 0$, i.e., if

$$\frac{a_1 b_2}{a_2 b_1} > 1 \tag{10.33}$$

The various operating voltages and tolerances must be chosen in such a way as to satisfy this condition.

Assuming that the condition has been satisfied, we may now proceed to determine suitable values for the resistances. To do this three equations are needed, two of which are obtained by noting that the optimum point at which to operate in the region of overlap in Fig. 10.25 is at the intersection of the two lines. From a d-c point of view, any point within the region of overlap is suitable; however, the point corresponding to the lowest resistances, which is the point of intersection, gives the most rapid switching rate, since it allows the largest currents to flow to charge and discharge the circuit capacitances. The coordinates of the point of intersection may be expressed in terms of the following two equations:

$$\frac{R_2}{R_1} = \frac{1}{a_1 b_2 / a_2 b_1 - 1} = K_1 \tag{10.34}$$
$$\frac{R_3}{R_1} = -\frac{b_2/a_2}{a_1 b_2 / a_2 b_1 - 1} = K_2 \tag{10.35}$$

If these two equations are supplemented by an equation giving $R_1$, the problem is solved. Such an equation is obtained from the limiting conditions on the $C$ divider and on the tube. These conditions are shown in

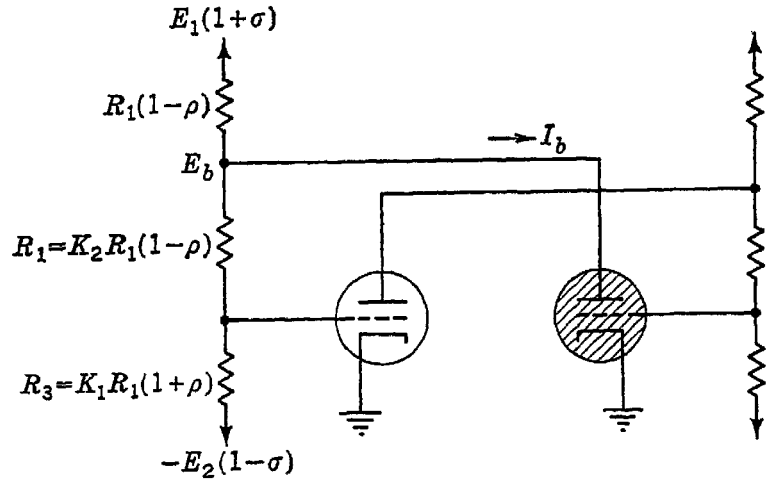



Fig. 10.26 Limiting conditions

Fig. 10.26. The Thevenin-theorem equivalent for the source supplying the current $I_b$ is a generator having voltage and resistance given by Eqs. (10.36) and (10.37):

$$E_{eq} = E_1(1 + \sigma)\alpha + E_2(1 - \sigma)(\alpha - 1) \tag{10.36}$$

$$R_{eq} = R_1(1 - \rho)\alpha \tag{10.37}$$

The quantity $\alpha$ in these equations is given by

$$\alpha = \frac{K_1 + K_2 + \rho(K_1 - K_2)}{(1 - \rho) + K_1 + K_2 + \rho(K_1 - K_2)} \tag{10.38}$$

$R_{eq}$ may also be expressed as

$$R_{eq} = \frac{E_{eq} - E_b}{I_b} \tag{10.39}$$

Thus we have finally

$$R_1 = \frac{E_{eq} - E_b}{I_b} \frac{1}{(1 - \rho)\alpha} \tag{10.40}$$

This result, together with Eqs. (10.34) and (10.35), permits the three resistances to be calculated in terms of the voltages $E_1$ and $E_2$ and the limits $E_b$, $I_b$, and $e_2$ determined by the tube type. The choice of supply voltages may often be dictated by considerations of the over-all design of the system of which the flip-flop is a part. However, if the supply voltages can be chosen to optimize flip-flop behavior, it is evident that the most rapid switching will result from the use of the largest possible voltages, since these allow the largest charging currents. Making both $E_1$ and $E_2$ large tends to increase the quantity $a_1b_2/a_2b_1$ of Eq. (10.33), thus making the point of intersection of the curves of Fig. 10.25 occur at a

value corresponding to lower resistance values for $R_2$ and $R_3$. At the same time, however, the designer must be prepared to accept increased power dissipation in the circuit. Thus, in practical circuits for computer use, the total voltage across the divider $R_1R_2R_3$ is typically of the order of 400 volts and seldom exceeds 500.

## 10.8 Switching considerations in the flip-flop

The rate at which a flip-flop switches over to its new state may be increased by inserting "speed-up" capacitors in parallel with the cross-over resistors $R_2$. Since these capacitors cannot be charged instantaneously, they allow voltage changes at the plates to be transferred to the opposite grids with only small attenuation. Besides this function, however, the speed-up capacitors perform the important function of memory. During the process of transition from one state to another, the two tubes, one with current increasing and one with current decreasing, reach a condition in which the two currents are equal. Without the "inertial" effect provided by the capacitors, the circuit would be as likely to fall back to its initial state as it would be to continue to the new state. Thus the capacitors store information about the direction of the transition. In order to minimize the effects of stray capacitances, the speed-up capacitors should be made large in comparison to the stray capacitances. However, if the speed-up capacitors are increased to improve the switching rate, the designer must be prepared to pay a price in slower recovery rate, since the larger capacitors require a longer time to charge up between switching operations. Twenty-five-$\mu\mu$f capacitors are commonly used in this application.

In order to make the voltages at the plates change rapidly, the circuit should be designed to carry high currents and should be laid out in such a way as to keep stray capacitances small. The tubes should have a short grid base; i.e., only a small excursion of grid voltage should be needed to swing the plate current from fully off to fully on. The tubes should also have high transconductance in the neighborhood of zero grid volts. (Many tubes have their maximum transconductance with the grid slightly positive, and flip-flops are often designed to have the grid of the conducting tube slightly positive to take advantage of this condition.)

A complete analysis of the switching process in the flip-flop is an extremely intricate problem that has never been completely solved. Nevertheless, practical designs of flip-flops can be carried out with the aid of design techniques such as those described here. Some organizations have designed flip-flops successfully with the aid of computers, since in this way a large body of arithmetical computations can be carried out easily and a wide range of parameters explored quickly in order to select

the optimum combination for a given design specification. In fact, for a simultaneous solution of the d-c problem, the switching problem, and the loading problem, the use of the computer is a necessity. Less-than-optimum designs can be achieved without the use of the computer, and the loading problem can always be solved by the straightforward method of employing cathode followers with the flip-flop.

## 10.9 Use of transistors in computer circuits

Before the development of reliable semiconductor diodes, the two circuit functions of logic and temporary storage were both performed by vacuum tubes in most computers. As better semiconductor diodes became available in the early 1950s, they gradually supplanted tubes as logic elements in new computer designs. Tubes continued to be used as amplifiers and in flip-flop circuits. The development of transistors, however, made available devices upon which entirely new systems of computer circuitry could be based, as well as systems adapted from the older vacuum-tube circuitry or from the vacuum-tube and semiconductor-diode circuitry.

From the simplest point of view, the transistor may be regarded simply as a device somewhat resembling a vacuum tube and having characteristics that permit it to be used in place of a vacuum tube after suitable circuit modifications, which are required by its low impedance level and different d-c operating conditions. To use the transistor merely as a replacement for the vacuum tube, however, is to use it very ineffectively, and the best transistor circuits are those designed from the start to take the best advantage of the transistor's desirable characteristics and to minimize the effects of its undesirable characteristics.

Many different classes of transistor circuitry for computers have been devised. The following partial list indicates the range of circuit techniques:

1. Dynamic circuitry, employing diode logic, point-contact transistor amplifier, and pulse transformer for output[5,6]
2. Diode logic, operated either synchronously (with clocked pull-up voltage) or asynchronously, together with transistor flip-flops based on the Eccles-Jordan circuit
3. Direct-coupled transistor logic, using transistors for gating and using a simplified Eccles-Jordan flip-flop with two transistors, two resistors, and one supply voltage[7,8]
4. The NOR circuit, based on the common emitter switch[9,10]
5. "Dual-range" circuitry, in which each flip-flop uses both an npn transistor and a pnp transistor.[11] The two transistors are on or off together, and two pairs of output levels are obtained.

The most useful forms of transistor circuitry for computers have been found to be those in which the transistor is used as a switch, with a d-c connection from the output of one unit to the input of the next. (This d-c connection is sometimes paralleled by a capacitor, as we shall see shortly, but this is for speed-up and not for d-c isolation.) The usefulness of transistors as switches is due primarily to the successful quantity production of reliable and reproducible junction transistors. Although the point-contact transistor can have values of $\alpha$ substantially larger than unity and $\alpha$ for junction transistors is always less than unity, the characteristics of point-contact transistors vary much more from unit to unit than those of junction transistors. Therefore, circuits designed to utilize the higher $\alpha$ of the point-contact transistor must be such that circuit behavior is relatively independent of individual transistor characteristics. Dynamic circuitry techniques, adapted from the vacuum-tube circuit of Fig. 10.11, have been used for this purpose and have been described in the literature.[5,6] A complete computer, the TRADIC, was built at the Bell Telephone Laboratories using such a circuit as the basic element. However, these circuits are much more complicated than those which use modern junction transistors, and they are now largely of historical interest.

## 10.10 Transistors as switches[12]

The common-emitter connection of the transistor makes an excellent switch and provides the basis for several important classes of transistor

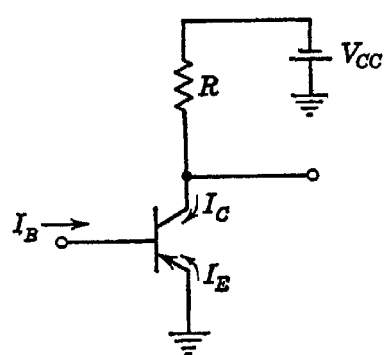



Fig. 10.27 The common emitter switch

logic systems. The basic circuit is shown in Fig. 10.27. When the transistor is turned off (by application of zero or slightly positive base current), its collector current is very small, and the impedance from collector to ground is very large. However, when the transistor is turned on (by application of a negative base current), the collector current can be very large, and the impedance from collector to ground drops from a value of several megohms to a value of the order of 1 ohm. The voltage drop

from collector to emitter when the transistor is turned on may be a few tenths of a volt or less if the transistor is conducting heavily.

We have already observed earlier in this chapter (Fig. 10.3) one possible way of using the common emitter switch in a logic configuration. Another logic system using the common emitter switch is shown in Fig. 10.28. If all the input leads are at their high levels, the output lead is at its low level, but if any input lead is at a low level, the output potential rises. Identifying "high" with "0" and "low" with "1", we see that this device performs the NOR function, i.e., the dual stroke function. This circuit is commonly referred to as "the transistor NOR circuit". It will be recalled from Chap. 8 that the stroke and the dual stroke are functionally complete; i.e., all other logical functions of two variables

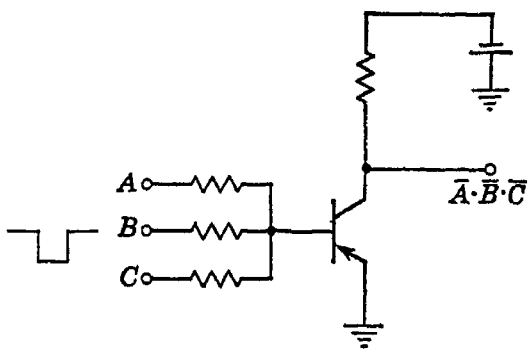



Fig. 10.28 A resistor-transistor logic circuit

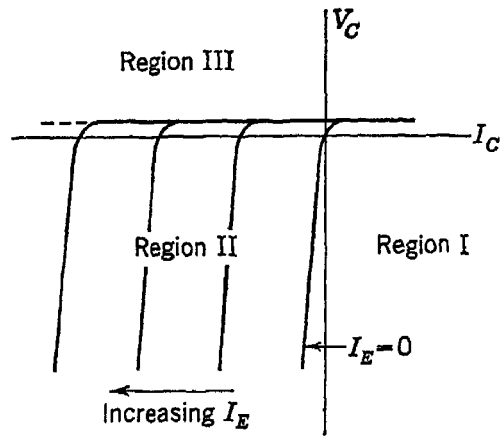



Fig. 10.29 Regions of operation, shown on the grounded-base collector characteristics

can be synthesized from stroke elements alone or from dual stroke elements alone. The use of NOR logic in digital systems is becoming increasingly popular.

In considering the operation of a transistor as a switching device, it is convenient to regard it as having three regions of operation. These are usually defined as follows:

Region I—the region of collector current cutoff, or collector voltage saturation. Both the emitter-base junction and the collector-base junction are biased in the reverse direction.

Region II—the active region. The emitter junction is forward-biased and the collector junction is reverse-biased.

Region III—the region of collector current saturation, or collector voltage cutoff. Both the emitter junction and the collector junction are forward-biased.

The boundaries of these regions are especially distinct on the family of collector characteristics for the common base connection, as shown in Fig. 10.29.

A detailed insight into the operation of the common emitter switch is provided by the Ebers-Moll equations,[12] which relate the emitter current and the collector current to the voltages of the emitter junction and the collector junction. The equations are

$$I_E = -\frac{I_{EO}}{1 - \alpha_N\alpha_I}(\epsilon^{qV_E/kT} - 1) + \frac{\alpha_I I_{CO}}{1 - \alpha_N\alpha_I}(\epsilon^{qV_C/kT} - 1) \quad (10.41)$$

$$I_C = +\frac{\alpha_N I_{EO}}{1 - \alpha_N\alpha_I}(\epsilon^{qV_E/kT} - 1) - \frac{I_{CO}}{1 - \alpha_N\alpha_I}(\epsilon^{qV_C/kT} - 1) \quad (10.42)$$

The transistor parameters appearing in these equations are as follows:

$\alpha_N$ = the normal alpha, i.e., the current gain $\partial I_C/\partial I_E$ with the emitter serving as emitter and the collector as collector

$\alpha_I$ = the inverted alpha, i.e., the current gain $\partial I_C/\partial I_E$ with the emitter serving as collector and the collector as emitter

$I_{EO}$ = the saturation current of the emitter junction with zero collector current

$I_{CO}$ = the saturation current of the collector junction with zero emitter current

As usual, $q$ is the magnitude of the electron charge ($1.6 \times 10^{-19}$ coulombs), $k$ is the Boltzmann constant ($1.38 \times 10^{-23}$ joules/°K), and $T$ is the temperature in degrees Kelvin.

In region II, where the collector is back-biased, $|\epsilon^{qV_C/kT}| \ll 1$, and these equations can be solved to give

$$I_C = -\alpha_N I_E + I_{CO} \quad (10.43)$$

If $\epsilon^{qV_C/kT}$ is not negligible, then

$$I_C = -\alpha_N I_E - I_{CO}(\epsilon^{qV_C/kT} - 1) \quad (10.44)$$

Equation (10.44) is valid in all regions, and (10.43) is the approximation commonly used to describe the transistor behavior in region II.

In region I, where both collector and emitter are back-biased, we may regard $\epsilon^{qV/kT}$ as negligible at both junctions and obtain

$$I_E = \frac{I_{EO}}{1 - \alpha_N\alpha_I} - \frac{\alpha_I I_{CO}}{1 - \alpha_N\alpha_I} \quad (10.45)$$

Ebers and Moll have shown that $\alpha_I I_{CO} = \alpha_N I_{EO}$ and (10.45) thus may be written

$$I_E = I_{EO}\frac{1 - \alpha_N}{1 - \alpha_N\alpha_I} \quad (10.46)$$

Similarly, we obtain

$$I_C = I_{CO}\frac{1 - \alpha_I}{1 - \alpha_N\alpha_I} \quad (10.47)$$

With the transistor in the grounded emitter connection of Fig. 10.27, it is appropriate to regard the base current as an independent variable. Since the sum of the three currents entering the transistor is zero, we may use Eqs. (10.41) and (10.42) to obtain

$$I_C = \frac{\alpha_N}{1 - \alpha_N} I_B - \frac{I_{CO}}{1 - \alpha_N} (\epsilon^{qV_C/kT} - 1) \tag{10.48}$$

From this equation it can be seen that the collector current can be reduced to zero only by applying a small positive current $I_B$, given by

$$I_B = \frac{I_{CO}}{\alpha_N} (\epsilon^{qV_C/kT} - 1) \tag{10.49}$$

For $I_B = 0$, we obtain

$$I_C = -\frac{I_{CO}}{1 - \alpha_N} (\epsilon^{qV_C/kT} - 1) \tag{10.50}$$

If $V_C$ is sufficiently negative, this becomes approximately

$$I_C = \frac{I_{CO}}{1 - \alpha_N} \tag{10.51}$$

Under this condition, a transistor having $I_{CO} = 2\ \mu a$ and $\alpha_N = 0.9$ would have $I_C = 20\ \mu a$, a very small value in comparison with the collector current that can flow with the transistor turned on.

In considering the transistor's behavior in region III, it is convenient, as Ebers and Moll point out, to regard the currents as independent variables and to express the junction voltages in terms of them. Thus we find

$$V_E = \frac{kT}{q} \ln\left(-\frac{I_E + \alpha_I I_C}{I_{EO}} + 1\right) \tag{10.52}$$

$$V_C = \frac{kT}{q} \ln\left(-\frac{I_C + \alpha_N I_E}{I_{CO}} + 1\right) \tag{10.53}$$

The voltage $V_{CE}$ from collector to emitter is found from these two equations and from the relation $\alpha_I I_{CO} = \alpha_N I_{EO}$.

$$V_{CE} = V_C - V_E = \frac{kT}{q} \ln \frac{I_{CO} - I_C(1 - \alpha_N) + \alpha_N I_B}{I_{CO} + \alpha_N I_C(1 - \alpha_I)/\alpha_I + (\alpha_N/\alpha_I) I_B} \tag{10.54}$$

This equation shows that the general shape of the collector characteristics for the grounded emitter connection is as shown in Fig. 10.30. The voltage $V_{CE}$ becomes zero at a current given by

$$I_C = -\frac{I_B}{1 + (\alpha_I/\alpha_N)(1 - \alpha_N)/(1 - \alpha_I)} \tag{10.55}$$

and, at $I_C = 0$, the voltage $V_{CE}$ is given by

$$V_{CE} = \frac{kT}{q} \ln \alpha_I \frac{1 + I_{CO}/\alpha_N I_B}{1 + (\alpha_I/\alpha_N)(I_{CO}/I_B)} \tag{10.56}$$

$V_{CE}$ approaches $+\infty$ as the denominator of Eq. (10.54) goes to zero, i.e., at

$$I_C = -\frac{I_B}{1 - \alpha_I} - \frac{\alpha_I I_{CO}}{\alpha_N(1 - \alpha_I)} \tag{10.57}$$

$V_{CE}$ approaches $-\infty$ as the numerator goes to zero, i.e., at

$$I_C = \frac{\alpha_N}{1 - \alpha_N} I_B + \frac{I_{CO}}{1 - \alpha_N} \tag{10.58}$$

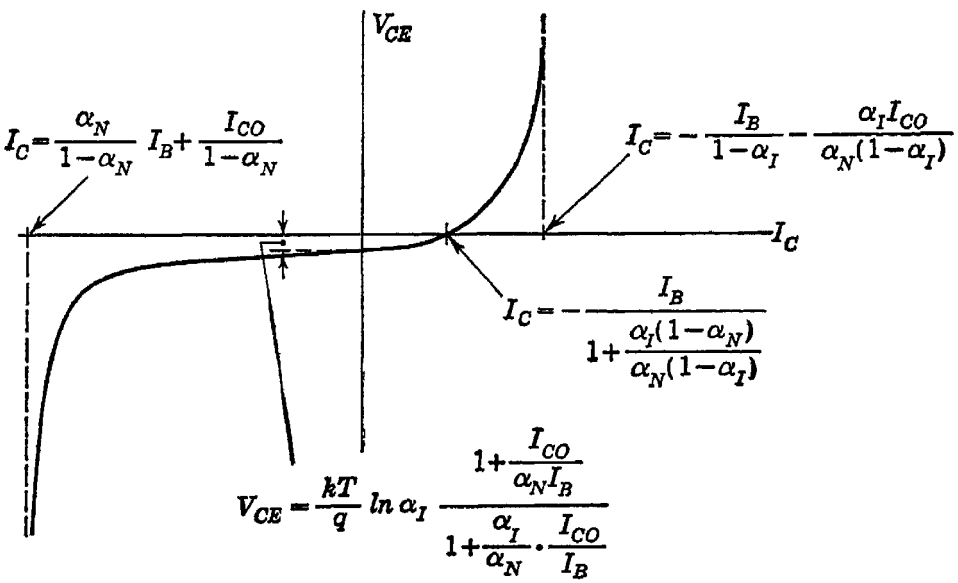



Fig. 10.30 Shape of the collector characteristic for the grounded-emitter connection

The output impedance of the transistor is found from

$$\begin{aligned}\frac{dV_{CE}}{dI_C} &= -\frac{kT}{q}\left[\frac{1 - \alpha_N}{I_{CO} - I_C(1 - \alpha_N) + \alpha_N I_B} + \frac{(\alpha_N/\alpha_I)(1 - \alpha_I)}{I_{CO} + \alpha_N I_C(1 - \alpha_I)/\alpha_I + (\alpha_N/\alpha_I) I_B}\right] \\ &\cong -\frac{kT}{q}\left\{\frac{1 - \alpha_N}{\alpha_N[I_B - I_C(1 - \alpha_N)/\alpha_N]} + \frac{1 - \alpha_I}{I_B + I_C(1 - \alpha_I)}\right\}\end{aligned} \tag{10.59}$$

Along the low-slope portion of the $V_{CE}$ versus $I_C$ curve, corresponding approximately to the boundary of region III, this impedance may easily be as low as 1 ohm.

## 10.11 D-C conditions in the NOR circuit[9,10]

The simple inverter of Fig. 10.27 must be augmented by interstage coupling resistances when it is used to perform NOR logic. These resist-

ances provide compensation for the differences in d-c level between the output of one logic stage and the input of the next, in a manner very similar to the interstage coupling networks used in the analog-computer amplifiers discussed in Chap. 4. The resistances also control the base-current drive and permit isolation between several inputs to the same base. The NOR logic circuit is shown in Fig. 10.31.

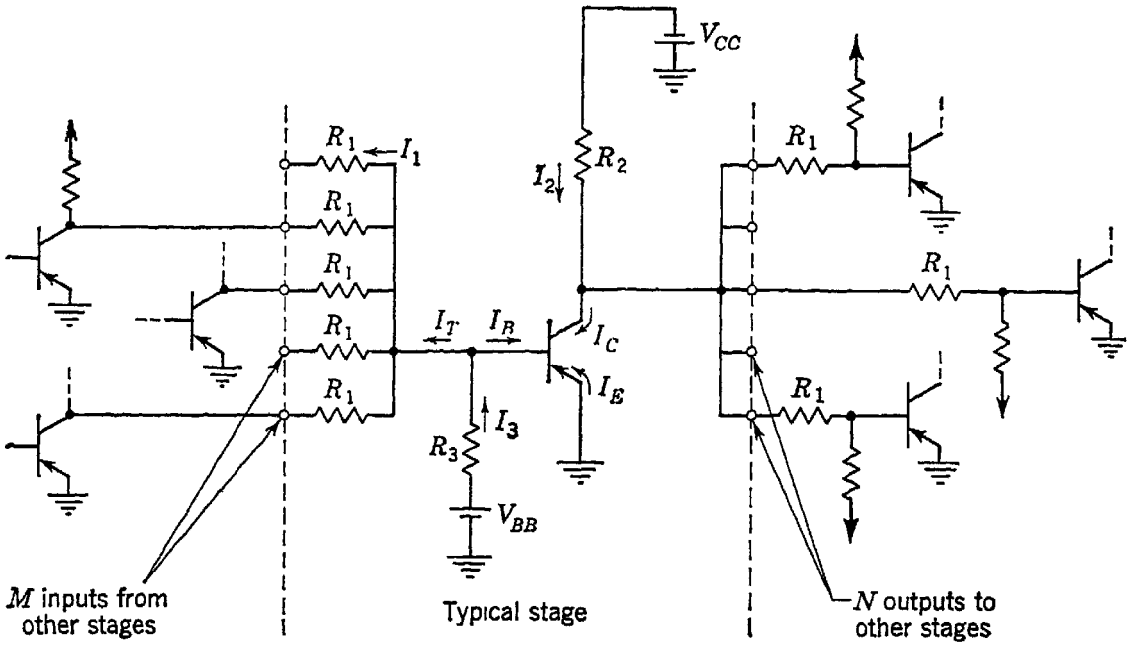



Fig. 10.31 The transistor NOR circuit

The function of $V_{BB}$ and $R_3$ is to supply a small current into the base when the transistor is cut off. To determine the required value of $I_B$ under these conditions, we note that the collector is heavily back-biased and that Eq. (10.48) may, therefore, be rewritten in the form

$$I_C = \frac{\alpha_N}{1 - \alpha_N} I_B + \frac{I_{CO}}{1 - \alpha_N} \tag{10.60}$$

The smallest collector current that can flow with reverse collector bias is $I_{CO}$. Thus

$$I_{CO} = \frac{\alpha_N}{1 - \alpha_N} I_B + \frac{I_{CO}}{1 - \alpha_N} \tag{10.61}$$

$$I_B = -I_{CO} \tag{10.62}$$

Since $I_{CO}$ is a negative number (for a pnp transistor), $I_B$ should be positive. At this point we should note one of the unfortunate limitations of the transistor, namely, its temperature sensitivity. The parameter $I_{CO}$ is highly temperature-sensitive, and in germanium transistors it approximately doubles for each 8°C rise in temperature. Thus the base bias circuit should be chosen to supply the value of $I_{CO}$ corresponding to the

highest temperature of operation. For example, if $I_{CO}$ is measured at 25°C and if the circuit is to operate satisfactorily at 65°C, the required maximum base bias current is

$$I_B = -I_{CO} \times 2^{(65-25)/8} = -32I_{CO} \tag{10.63}$$

When the transistor is cut off, its base potential is of the order of 0.1 volt above ground, as can be seen from Eq. (10.52). For $I_C = I_{CO}$ and $I_B = -I_{CO}$, this equation becomes

$$V_{BE} = -\frac{kT}{q}\ln\left(-\frac{\alpha_I I_{CO}}{I_{EO}} + 1\right) \tag{10.64}$$

$$= -\frac{kT}{q}\ln(1 - \alpha_N) \tag{10.65}$$

For a transistor having $\alpha_N = 0.9$, $V_{BE} = 0.06$ volt at room temperature. Since $I_3 = I_T + I_B$, we must now find $I_T$ before we can determine $R_3$. $I_T$ is determined by the resistances $R_1$ and by the voltages of the saturated transistors to which the $R_1$'s are connected. $R_1$ in turn is determined by the conditions required to turn the transistor on.

In the fully ON state, the collector current of the transistor will be required to be at least as great as some minimum value $I_{C\,\min}$. The base current $I_{B\,\min}$ required to give this value of $I_C$ at voltages $V_{CE}$ less than any desired magnitude can be read directly from a family of collector characteristics (grounded emitter connection) for the transistor. Since the transistor must be turned fully on by any input signal, a preceding stage that turns the transistor on must supply a current $I_1$ given by

$$I_1 = \frac{V_{BE} - V_{\text{out min}}}{R_1} = -kI_{B\,\min} + I_3 - (M-1)\frac{V_{BE} - V_{CE}}{R_1} \tag{10.66}$$

In this equation $V_{BE}$ is the base voltage of the ON transistor and $V_{CE}$ is the largest (in magnitude) of the collector voltages of the ON transistors connected to $M - 1$ of the resistors $R_1$. $V_{BE} - V_{CE}$ can usually be approximated safely as a few tenths of a volt, and $I_3$ is closely equal to $V_{BE}/R_3$. The constant $k$ is a safety factor by means of which tolerance to gain deterioration of the transistor can be incorporated in the circuit.

The heaviest output current that any stage is required to deliver could occur if each of its $N$ outputs simultaneously supplied the full input current to each of $N$ stages. Summing the currents at the collector node, we obtain

$$NI_1 - I_{CO} + \frac{-V_{CC} - V_{\text{out min}}}{R_2} = 0 \tag{10.67}$$

$$V_{\text{out min}} = R_2(NI_1 - I_{CO}) - V_{CC} \tag{10.68}$$

where $V_{CC}$ is regarded as a positive number and the reversed polarity is

indicated by the minus sign. We also note that

$$V_{\text{out min}} = V_{BE} - I_1R_1 \tag{10.69}$$

Thus, $R_2(NI_1 - I_{CO}) - V_{CC} = V_{BE} - I_1R_1$ (10.70)

and

$$N = \frac{1}{I_1}\left(\frac{V_{BE} + V_{CC}}{R_2} + I_{CO}\right) - \frac{R_1}{R_2} \tag{10.71}$$

This equation may be simplified by choosing $V_{CC} \gg V_{BE}$ and by selecting $-V_{CC}/R_2$ to be equal to the desired minimum value of $I_C$ for the ON transistor. We then find

$$N = -I_C\left(\frac{1}{I_1} - \frac{R_1}{V_{CC}}\right) \tag{10.72}$$

$I_1$, as given by Eq. (10.66), may be substituted in (10.72) to give

$$N = -I_C\left[\frac{1}{-kI_{b\,\text{min}} + I_3 - (M-1)(V_{BE} - V_{CE})/R_1} - \frac{R_1}{V_{CC}}\right] \tag{10.73}$$

Many of the quantities in this equation can be readily determined. Thus $V_{BE} - V_{CE}$ can be safely taken to be $-0.2$ volt. $I_C$ and $I_{b\,\text{min}}$ can be found from the collector characteristics. $V_{CC}$ should be taken as large as possible without danger of exceeding the reverse breakdown voltage of the collector junction. $I_3$, which must equal the sum of the base bias current and the currents through all $M$ resistors $R_1$ to ON transistors, may be taken to be two or three times the magnitude of $I_{CO}$. A reasonable value for $k$ is 2 to 3. The number of inputs $M$ should, preferably, be at least as large as 5, since there are many occasions for gating five or more signals together in most digital systems. Although $M$ can be chosen smaller than 5, such a choice requires more transistors per level of logic as well as more levels of logic for all but the simplest logical functions. On the other hand, if $M$ is made as large as 20, for instance, many of the inputs will be unused in most systems. In the absence of circuit restrictions that fix $M$, the designer might well choose $M = 10$.

Once these choices have all been made, the number $N$ of outputs from a stage is a function only of the input resistance $R_1$. It may be observed qualitatively that, if $R_1$ is chosen too large, the current $I_T$ (Fig. 10.31) will not change much when a single one of the driving transistors changes its state unless the number of inputs $M$ is decreased. Likewise, if $R_1$ is made too small, the collector bias supply of an OFF transistor cannot draw enough current to hold succeeding stages ON unless $N$ is made small. Various values of $R_1$ can be substituted into Eq. (10.73) to determine the manner of variation of $N$, and then a suitable combination of $N$ and $R_1$ can be chosen.

One last step in the design of the circuit is to compute the power dissipation within the transistor to see whether it is within the specified

limit. The power dissipation $P$ is given by

$$P = V_{BE}I_B + V_{CE}I_C \tag{10.74}$$

The largest base current occurs when every one of the $M$ input leads is connected to an OFF transistor and each such OFF transistor drives only the one transistor under consideration. Then

$$I_B = -M\frac{V_{CC} + V_{BE}}{R_1 + R_2} \cong -\frac{MV_{CC}}{R_1 + R_2} \tag{10.75}$$

If sufficient data on the transistor are available, it may be possible to calculate $V_{BE}$ directly by the use of Eq. (10.52). If not, one can use instead a conservative estimate of the largest value that $V_{BE}$ will have. If the power dissipation is well within the allowable limit when computed using $V_{BE}$ as large as perhaps $-\frac{3}{4}$ volt, then the circuit design may be taken to be sufficiently conservative. However, if the power so computed appears only marginally safe, additional measurements or calculations to find $V_{BE\ \max}$ more precisely may be warranted.

The foregoing discussion does not include any consideration of circuit changes due to resistor variations or supply-voltage drifts. In practice, allowance would have to be made for such changes before one could assemble with confidence a large number of these components into a system. It is possible to make a trial-and-error analysis, in which components and voltages are allowed to change by arbitrary amounts in the most disadvantageous direction and the circuit behavior is then recalculated to ascertain whether it is still satisfactory. This procedure may be long and arduous, however, and it may be more convenient to write a set of equations in which tolerance limits appear explicitly and can be calculated, much as the calculations for the Eccles-Jordan vacuum-tube flip-flop were carried out in Sec. 10.7. Use of a digital computer may be desirable for carrying out the calculations or for computing the behavior of the circuit.[13]

## 10.12 Transient conditions in transistor switches[14,15]

The complete design of a transistor switch must include careful attention not only to the d-c operating conditions but also to the transient conditions that exist during transistor turn-on and turn-off. Operating conditions that are entirely satisfactory from the d-c point of view may result in slow switching unless the transient conditions are also carefully controlled.

The turn-on and turn-off transients are due to basically different mechanisms in the transistor. The principal factor contributing to the turn-on delay is the drop in $\alpha$ at high frequencies; the main cause of

turn-off delay is minority carrier storage in the base region. Let us first examine the turn-on transient for the common emitter circuit of Fig. 10.32. As we have already seen, the collector current in the active region is given by

$$I_C = -\alpha_N I_E + I_{CO} \tag{10.76}$$

Neglecting $I_{CO}$ and replacing $I_E$ by $-(I_B + I_C)$, we get

$$I_C = \frac{\alpha_N}{1 - \alpha_N} I_B \tag{10.77}$$

To calculate $I_C$ in response to a step increase in $I_B$, we must note that $\alpha_N$ is a function of frequency

$$\alpha_N = \frac{\alpha_0}{1 + j\omega/\omega_N} \tag{10.78}$$

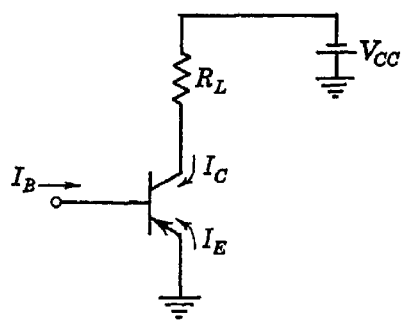



Fig. 10.32 The common emitter circuit

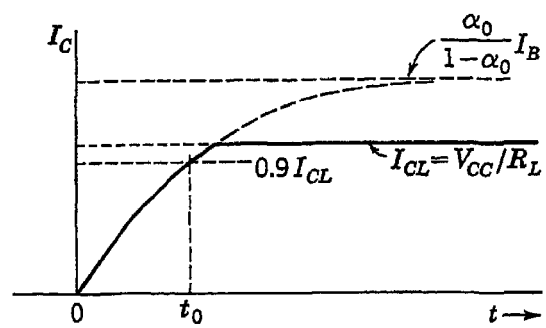



Fig. 10.33 Transistor turn-on

where $\omega_N$ is the angular frequency at which $\alpha_N$ has dropped 3 db. Thus

$$I_C = \frac{\alpha_0}{(1 + j\omega/\omega_N)[1 - \alpha_0/(1 + j\omega/\omega_N)]} I_B \tag{10.79}$$

$$= \frac{\alpha_0}{1 - \alpha_0} \frac{1}{1 + j\omega/\omega_N(1 - \alpha_0)} I_B \tag{10.80}$$

Thus, as a function of time,

$$I_C = \frac{\alpha_0}{1 - \alpha_0} I_B(1 - \epsilon^{-\omega_N(1-\alpha_0)t}) \tag{10.81}$$

The time required for the collector current to reach a value of $I_C$ amp can be found from this equation, and is

$$t = \frac{1}{\omega_N(1 - \alpha_0)} \ln \frac{I_B}{I_B - I_C(1 - \alpha_0)/\alpha_0} \tag{10.82}$$

Of course, if $I_B$ is large, the transistor will be driven heavily into saturation, and the ultimate value of the collector current will be determined by the external circuit. If $I_{CL}$ is this ultimate value of collector current and if we define the rise time $t_0$ to be the time required for the collector cur-

rent to reach 0.9 of its limiting value, then

$$t_0 = \frac{1}{\omega_N(1 - \alpha_0)} \ln \frac{I_B}{I_B - 0.9I_{CL}(1 - \alpha_0)/\alpha_0} \tag{10.83}$$

The rise time can be decreased by increasing $I_B$, increasing $\alpha_0$, or increasing $\omega_N$. Figure 10.33 shows a typical curve of $I_C$ versus time.

The foregoing analysis is based upon the assumption that $\alpha$ cutoff is the only effect limiting the rise rate of $I_C$. Although this assumption is usually valid, it ignores the effect of the collector-junction capacitance. If the load impedance is sufficiently low, the time constant formed by the collector capacitance and the load impedance may be assumed to have negligible effect on the rise rate, but for high load impedances this effect would also have to be considered.

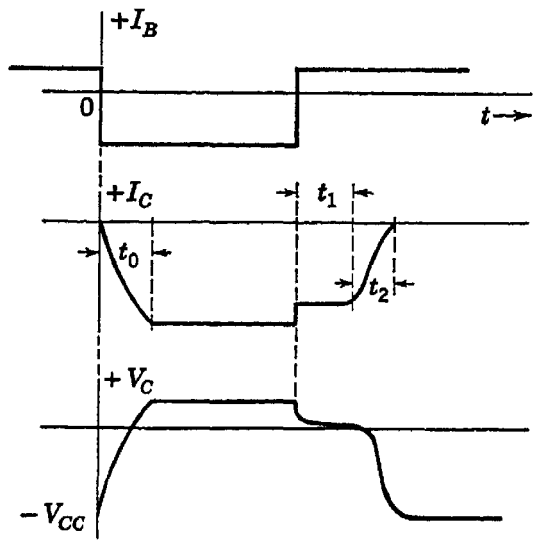



Fig. 10.34 Turn-on and turn-off transients, where $t_0$ = turn-on time, $t_1$ = storage time, $t_2$ = decay time

The turn-off transient has two distinct regions: (1) the storage time, and (2) the decay time. The storage time is the interval during which minority carriers are swept out of the base region. The collector impedance remains low as long as the minority-carrier density at the collector junction exceeds zero. Thus the collector current remains substantially unchanged during this time. The voltage across the collector junction, which was slightly positive while the transistor was in region III, diminishes only slightly during the storage interval. As the collector impedance begins to rise, after the minority-carrier density at the collector junction has reached zero, the collector current decreases and the collector voltage approaches $V_{CC}$. Figure 10.34 shows the turn-on and turn-off transients in response to changes in $I_B$.

The more heavily the transistor has been driven into saturation, the longer will be the storage time before the collector impedance begins to recover. This is the cause of one of the design dilemmas facing the engineer who wishes to build a system of transistor switches around transistors having a wide range of values of $\beta = \alpha_N/(1 - \alpha_N)$. If design conditions are chosen so that transistors having a low value for $\beta$ are barely driven into saturation, then transistors for which $\beta$ is three or four times as large will be driven heavily into saturation and will consequently have a much longer recovery time. Since $\beta$ is very sensitive to changes in $\alpha_N$, it is not easy to select transistors having approximately

the same values for $\beta$ or to be sure that these values will remain the same in use. One way of avoiding this problem is to use limiting diodes, which prevent the transistor from going into saturation.[16]

Moll[15] has shown that the storage time for a transistor in the grounded emitter connection is

$$t_1 = \frac{\omega_N + \omega_I}{\omega_N \omega_I (1 - \alpha_N \alpha_I)} \ln \frac{I_{B1} - I_{B2}}{I_{C1}(1 - \alpha_N)/\alpha_N - I_{B2}} \tag{10.84}$$

As usual, $\alpha_N$ and $\alpha_I$ are the normal $\alpha$ and the inverse $\alpha$, and $\omega_N$ and $\omega_I$ are their respective cutoff frequencies in radians. $I_{B1}$ and $I_{B2}$ are the base currents before and after the switching pulse has been applied, and $I_{C1}$ is the initial value of $I_C$ with the transistor on.

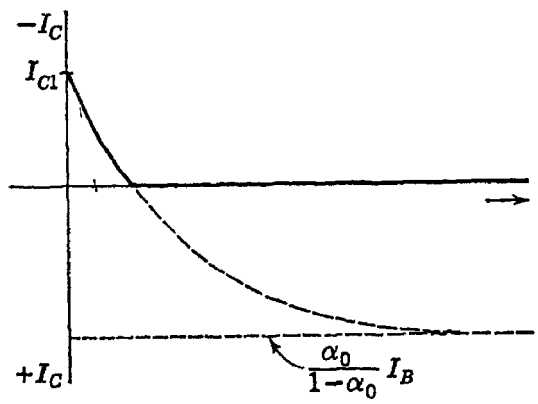



Fig. 10.35 Decay of collector current from initial value of $I_{C1}$ when pulse of $I_B$ amp flows into base

The second part of the turn-off transient commences when the minority carrier density in the base has reached zero at the collector junction. The transistor is then effectively out of saturation and operating in region II, and the collector current falls exponentially. If the base current has been merely reduced to zero, then the collector current also drops toward zero. However, if a reverse current is allowed to flow in the emitter-base junction,

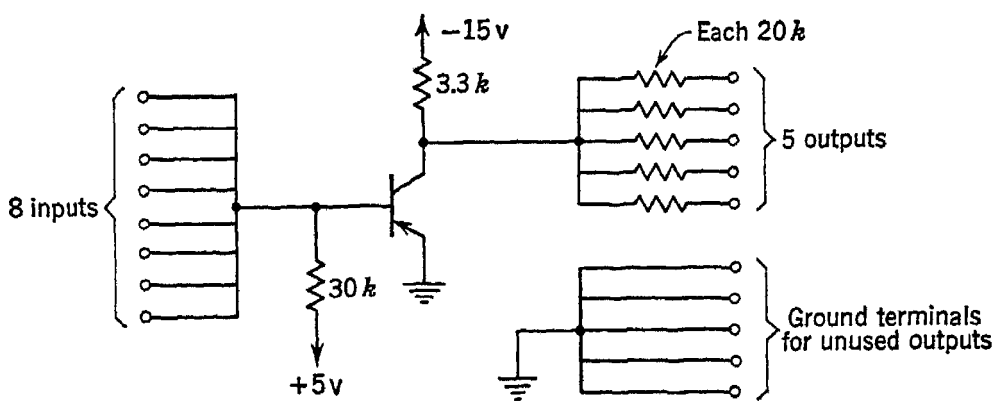



Fig. 10.36 A NOR module

then the collector current will decrease more rapidly, along a curve that approaches a reverse value of $I_B\alpha_0/(1 - \alpha_0)$. The collector current will not actually reverse, of course, but will reach zero more quickly than if $I_B = 0$. Figure 10.35 shows $I_C$ as a function of time during the decay interval.

The time constant for this curve can be found from the expression for the frequency dependence of $\alpha_N$, as given in Eq. (10.78).

$$\frac{\alpha_N}{1 - \alpha_N} = \frac{\alpha_0 \omega_N}{\omega_N(1 - \alpha_0) + j\omega} \tag{10.85}$$

and the time constant is

$$T = \frac{1}{\omega_N(1 - \alpha_0)} \tag{10.86}$$

The curve of Fig. 10.35 may, therefore, be expressed as

$$I_C = -\left(I_{C1} + \frac{\alpha_0}{1 - \alpha_0} I_B\right) \epsilon^{-\omega_N(1-\alpha_0)t} + \frac{\alpha_0}{1 - \alpha_0} I_B \tag{10.87}$$

This equation can be solved for $t$ to give

$$t = \frac{1}{\omega_N(1 - \alpha_0)} \ln \frac{-I_{C1} - I_B\alpha_0/(1 - \alpha_0)}{I_C - I_B\alpha_0/(1 - \alpha_0)} \tag{10.88}$$

Thus $t_2$, the time required for $I_C$ to fall to $\frac{1}{10}$ its initial value of $-I_{C1}$, is

$$t_2 = \frac{1}{\omega_N(1 - \alpha_0)} \ln \frac{I_{C1} + I_B\alpha_0/(1 - \alpha_0)}{0.1I_{C1} + I_B\alpha_0/(1 - \alpha_0)} \tag{10.89}$$

($I_{C1}$ is taken as negative for a pnp transistor.)

Many computer laboratories find the NOR circuit a convenient module for construction of experimental circuits and have built up assemblies of these units with all leads brought out to pin jacks on patch boards so that logic circuits may be readily assembled. A typical NOR circuit is shown in Fig. 10.36; this is used in the Digital Computer Engineering Laboratory at the University of Michigan. Five such NOR circuits can readily be assembled on a plug board about 3 in. square.

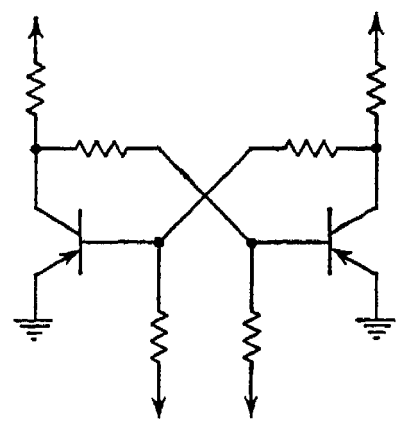


Fig. 10.37 A flip-flop made from NOR circuits

A simple flip-flop circuit can be made from two NOR circuits, as shown in Fig. 10.37, and more complex structures can also be easily assembled.

## 10.13 Logical design with the NOR circuit[17,18]

In previous chapters logic structures have been assembled from AND, OR, and NOT elements. The use of the NOR element or its dual, the NAND element, may seem at first somewhat clumsy, but there is no fundamental reason that one cannot develop a high degree of skill in the use of these circuits. First let us recall the definitions of the dagger function (or dual stroke) and the stroke function, as shown in Table 10.1. Depending upon our assignment of truth values to the high and low levels of potential, the common emitter circuit of Fig. 10.28 may be regarded as

performing either of these two functions. If the higher level of potential is designated as 1 and the lower as 0, the circuit performs the stroke function, sometimes also called the NOT AND or NAND function. (This assignment of values is sometimes called "positive logic".) Letting the higher level be 0 and the lower level be 1 ("inverted logic") causes the circuit to perform the dual stroke (NOR) function.

**Table 10.1**
**The dagger function and the stroke function**

| $A$ | $B$ | $A \downarrow B$ (NOR) | $A \mid B$ (NAND) |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

Since many logical functions are expressed in the so-called canonical forms, either disjunctive or conjunctive, it is desirable to be able to convert these forms for use with stroke elements. The reader may easily verify that

$$A + B = \overline{A \downarrow B} = \bar{A} \mid \bar{B} \tag{10.90}$$

and

$$A \cdot B = \bar{A} \downarrow \bar{B} = \overline{A \mid B} \tag{10.91}$$

In addition to these relations, the following useful rules should also be verified:

$$A \downarrow (B \downarrow C) \neq (A \downarrow B) \downarrow C \tag{10.92}$$
$$A \mid (B \mid C) \neq (A \mid B) \mid C \tag{10.93}$$
$$A \downarrow B = B \downarrow A \tag{10.94}$$
$$A \mid B = B \mid A \tag{10.95}$$
$$A \downarrow (B + C) = \bar{A} \cdot \bar{B} \cdot \bar{C} = (A \downarrow B) \cdot (A \downarrow C) \tag{10.96}$$
$$A \mid (B + C) = \bar{A} + (\bar{B} \cdot \bar{C}) = (A \mid B) \cdot (A \mid C) \tag{10.97}$$
$$A \downarrow (B \cdot C) = \bar{A} \cdot (\bar{B} + \bar{C}) = A \downarrow B + A \downarrow C \tag{10.98}$$
$$A \mid (B \cdot C) = \bar{A} + \bar{B} + \bar{C} = A \mid B + A \mid C \tag{10.99}$$
$$A \downarrow (B \mid C) = \bar{A} \cdot B \cdot C \tag{10.100}$$
$$A \mid (B \downarrow C) = \bar{A} + B + C \tag{10.101}$$
$$A \mid B = \overline{\bar{A} \downarrow \bar{B}} \tag{10.102}$$
$$A \downarrow B = \overline{\bar{A} \mid \bar{B}} \tag{10.103}$$
$$(\bar{A} \mid \bar{B}) \mid (A \mid B) = (A \equiv B) \tag{10.104}$$
$$(A \mid \bar{B}) \mid (\bar{A} \mid B) = (A \not\equiv B) = A \oplus B \tag{10.105}$$
$$(A \downarrow B) \downarrow (\bar{A} \downarrow \bar{B}) = A \oplus B \tag{10.106}$$
$$(A \downarrow \bar{B}) \downarrow (\bar{A} \downarrow B) = (A \equiv B) \tag{10.107}$$

The reader will observe that the stroke and dagger functions are commutative but are not otherwise readily manipulable. Neither the stroke

nor the dagger is associative, and neither of these functions is distributive over the other or over disjunction or conjunction.

So far we have treated the stroke and dagger functions as binary operators, i.e., as functions having two arguments. However, since the electronic circuits for performing these functions are readily extended to

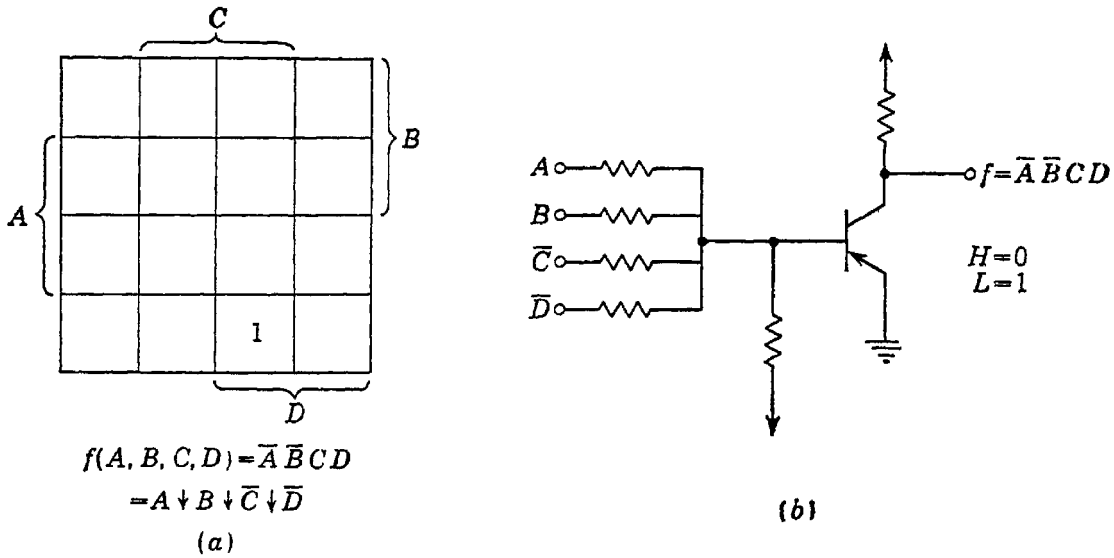



Fig. 10.38 A one-square function and its NOR realization

have many inputs, it is convenient to modify our notation to apply to such circuits. Probably the simplest way to extend our notation is to define an expression such as

$$A_1 \downarrow A_2 \downarrow A_3 \downarrow \cdots \downarrow A_n \tag{10.108}$$

as denoting the function performed by a single NOR circuit with $n$ inputs, i.e.,

$$A_1 \downarrow A_2 \downarrow A_3 \cdots \downarrow A_n = \bar{A}_1\bar{A}_2\bar{A}_3 \cdots \bar{A}_n \tag{10.109}$$

Likewise,

$$A_1|A_2|A_3 \cdots |A_n = \bar{A}_1 + \bar{A}_2 + \bar{A}_3 \cdots + \bar{A}_n$$

It should be noted that without this definition a cascading of stroke or dagger functions without parentheses is meaningless, since the functions are not associative.

Subject to this extension of the notation, we can now show that any function of $n$ variables can be realized by at most a three-level cascade of stroke or dagger circuits, where the inputs to the first level are the variables or their complements.

Consider, for example, a function of $n$ variables whose map contains only one square (i.e., the table of combinations contains only a single 1). Such a function is expressible as a single dagger function among the negations of the variables identifying the square. The function in Fig. 10.38 is an example.

If the function map consists of 2, 4, 8, . . . , or $2^i$ adjacent squares, again only a single NOR circuit is needed. The inputs will be the complements of the variables defining the set of adjacent squares. Figure 10.39 shows an example of such a function.

If a function map consists of a group of squares or subcubes no two of which are adjacent, then each subcube may be expressed as a NOR function and the outputs of the various NOR circuits may be disjoined by use of the identity

$$A + B = \overline{A \downarrow B} \tag{10.110}$$

The disjunction operation requires a two-level cascade of NOR elements. Since we have already seen (Chap. 8) that any function can be expressed as a disjunction of conjunctions (sum of products), it is now apparent that any function can be expressed by at most a NOR-NOR-NOR circuit.

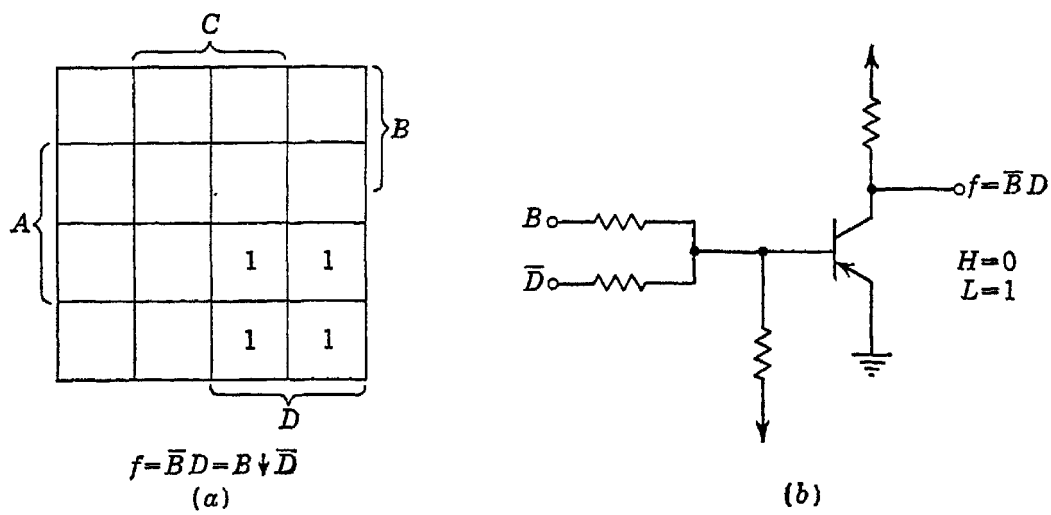



Fig. 10.39 Another function and its NOR realization

By a similar line of reasoning, it is apparent that any function can be alternatively expressed by at most a three-level cascade of NAND circuits. To show this, we first observe that any function may be expressed in the conjunctive normal form. Each factor of the conjunction is a disjunction of one or more literals, and each such disjunction may be expressed, by Eq. (10.90), as a stroke among the complements of the literals. Then the conjunction may be performed, by Eq. (10.91), by two more levels of stroke functions.

The general procedures for converting functions from disjunctive or conjunctive normal forms to forms using the stroke only or the dagger only are tabulated in Table 10.2. It will be observed that if a function in conjunctive normal form is to be expressed with stroke functions, the complementation of the whole expression requires a stroke element at the output point. Likewise, if a function in disjunctive normal form is to

be expressed with dagger functions, the final complementation of the whole expression requires a dagger element at the output. In each case this final level of logic is unnecessary, since one less level is needed if the realization in terms of stroke functions starts from the disjunctive normal form or if the realization in terms of dagger elements starts from the conjunctive normal form.

If the function has the same number of literals whether written in minimum sum-of-products form or in minimum product-of-sums form,

**Table 10.2**
**Procedures for converting functions to stroke or dagger form**

| | *Function initially in disjunctive normal form* | *Function initially in conjunctive normal form* |
|---|---|---|
| To express in terms of stroke function, i.e., NAND: | 1. Bracket each term of the disjunction<br>2. Complement any literal which appears by itself, i.e., not conjoined with other literals<br>3. Replace all + and · symbols by \| symbols | 1. Bracket each factor of the conjunction<br>2. Complement all literals of each factor, unless the factor consisted of only one literal<br>3. Replace all + and · symbols by \| symbols<br>4. Place a complement bar over the entire function |
| To express in terms of dagger function, i.e., NOR: | 1. Bracket each term of the disjunction<br>2. Complement all literals of each conjunction, unless the conjunction consisted of only one literal<br>3. Replace all + and · symbols by ↓ symbols<br>4. Place a complement bar over the entire function | 1. Bracket each factor of the conjunction<br>2. Complement any literal which appears by itself, i.e., not disjoined with other literals<br>3. Replace all + and · symbols by ↓ symbols |

then it will be found that the dagger circuit derived from the sum-of-products form is the same as the stroke circuit derived from the product-of-sums form. Likewise, the dagger circuit derived from the product-of-sums form is the same as the stroke circuit derived from the sum-of-products form. The latter circuit is a two-level circuit, however, while the former uses three levels.

If the function has fewer literals in one minimum normal form than in the other, it may be desirable from the standpoint of circuit minimization to start from the form using the fewer literals. Thus the function

$$f(A,B,C) = A + BC$$

also can be written

$$f(A,B,C) = (A + B)(A + C)$$

In this example the stroke function circuit derived from the sum-of-products form will require fewer total input resistors than the dagger circuit derived from the product-of-sums form.

As an example, let us consider the equations for the sum and carry digits in a three-input adder stage:

$$S_j = x_j\bar{y}_j\bar{c}_j + \bar{x}_jy_j\bar{c}_j + \bar{x}_j\bar{y}_jc_j + x_jy_jc_j \tag{10.111}$$

$$c_{j+1} = x_jy_j + x_jc_j + y_jc_j \tag{10.112}$$

The maps of these functions appear in Fig. 10.40. Equation (10.111) may be rewritten as follows:

$$S_j = \overline{(\bar{x}_j \downarrow y_j \downarrow c_j) \downarrow (x_j \downarrow \bar{y}_j \downarrow c_j) \downarrow (x_j \downarrow y_j \downarrow \bar{c}_j) \downarrow (\bar{x}_j \downarrow \bar{y}_j \downarrow \bar{c}_j)} \tag{10.113}$$

and thus a circuit for $S_j$ would appear as shown in Fig. 10.41. (In this figure the box with the dagger inside is introduced as a compact symbol

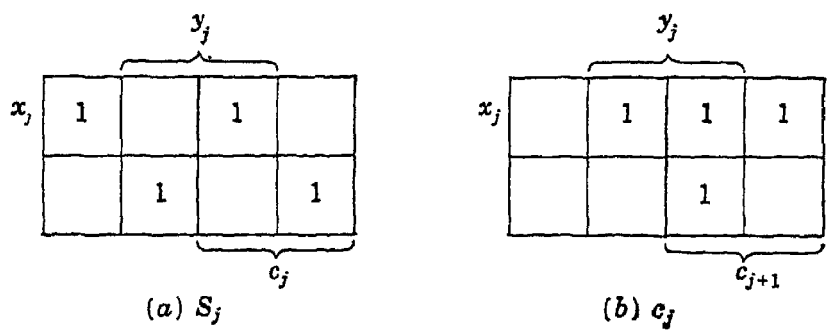



Fig. 10.40 Maps of sum and carry functions

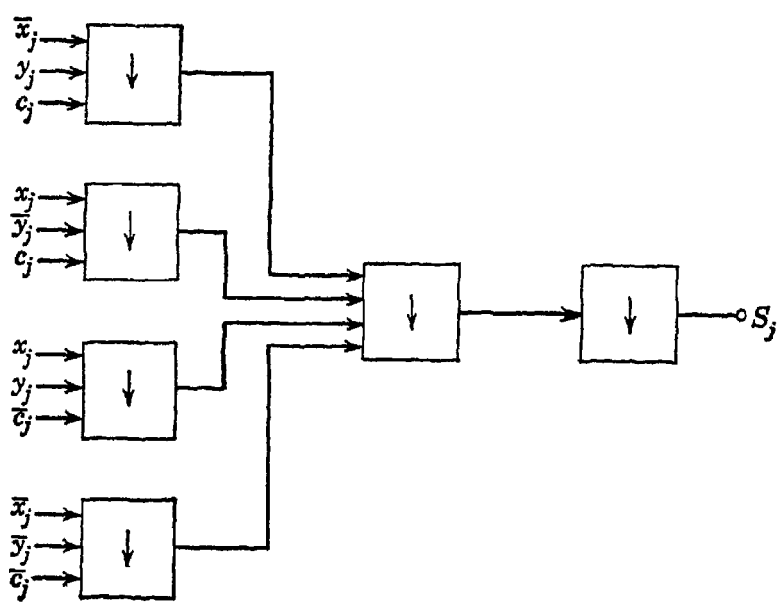



Fig. 10.41 A three-level NOR circuit for the binary-sum digit $S_j$

for the transistor NOR circuit.) If the variables are not available at the input in both complemented and uncomplemented forms, an additional level of NOR circuits will be needed to furnish whichever form is not available.

Now let us derive a circuit starting from the conjunctive normal form. We may write

$$S_j = (x_j + y_j + c_j)(\bar{x}_j + y_j + \bar{c}_j)(\bar{x}_j + \bar{y}_j + c_j)(x_j + \bar{y}_j + \bar{c}_j) \quad (10.114)$$

From this we find

$$S_j = \overline{x_j + y_j + c_j} \downarrow \overline{\bar{x}_j + y_j + \bar{c}_j} \downarrow \overline{\bar{x}_j + \bar{y}_j + c_j} \downarrow \overline{x_j + \bar{y}_j + \bar{c}_j} \quad (10.115)$$

$$= \overline{\overline{x_j \downarrow y_j \downarrow c_j}} \downarrow \overline{\overline{\bar{x}_j \downarrow y_j \downarrow \bar{c}_j}} \downarrow \overline{\overline{\bar{x}_j \downarrow \bar{y}_j \downarrow c_j}} \downarrow \overline{\overline{x_j \downarrow \bar{y}_j \downarrow \bar{c}_j}} \quad (10.116)$$

$$= (x_j \downarrow y_j \downarrow c_j) \downarrow (\bar{x}_j \downarrow y_j \downarrow \bar{c}_j) \downarrow (\bar{x}_j \downarrow \bar{y}_j \downarrow c_j) \downarrow (x_j \downarrow \bar{y}_j \downarrow \bar{c}_j) \quad (10.117)$$

This function can be realized as shown in Fig. 10.42.

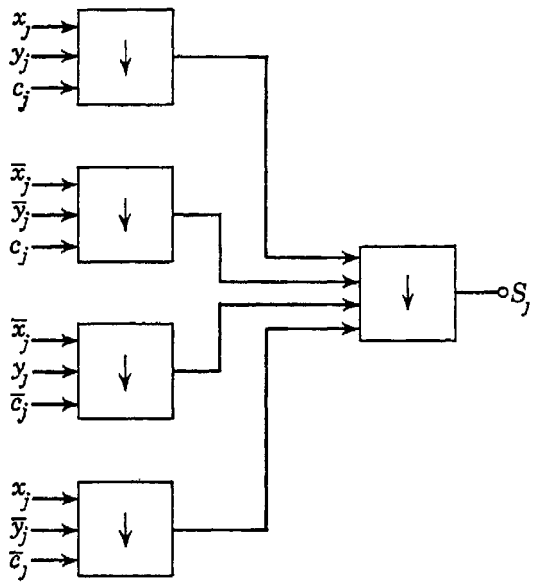



Fig. 10.42 A two-level NOR circuit for the binary-sum digit $S_j$

It will be noted that Fig. 10.42 can be obtained from Fig. 10.41 by changing the complementation symbols of all inputs of Fig. 10.41 and then deleting the final NOR circuit, which acts as a complementer. The reader is cautioned against making a general rule of this procedure, however. One can say in general that if, in a NOR or NAND cascade, one level performs only complementation, then that level can be eliminated by a different choice of distribution of complementation symbols on the input variables. Consider, for example, the function

$$f(A,B,C) = AB + \bar{A}C \quad (10.118)$$

$$= \overline{AB \downarrow \bar{A}C} \quad (10.119)$$

$$= \overline{(\bar{A} \downarrow \bar{B}) \downarrow (A \downarrow \bar{C})} \quad (10.120)$$

This function has the circuit realization shown in Fig. 10.43. Alternatively, we may write

$$f(A,B,C) = (A + C)(\bar{A} + B) \tag{10.121}$$
$$= \overline{A + C} \downarrow \overline{\bar{A} + B} \tag{10.122}$$
$$= \overline{\overline{(A \downarrow C)}} \downarrow \overline{\overline{(\bar{A} \downarrow B)}} \tag{10.123}$$
$$= (A \downarrow C) \downarrow (\bar{A} \downarrow B) \tag{10.124}$$

The circuit realization for this function appears in Fig. 10.44. It will be observed that, although Fig. 10.44 does not contain the complementing NOR element of Fig. 10.43, the inputs of Fig. 10.44 are not simply the complements of those of Fig. 10.43.

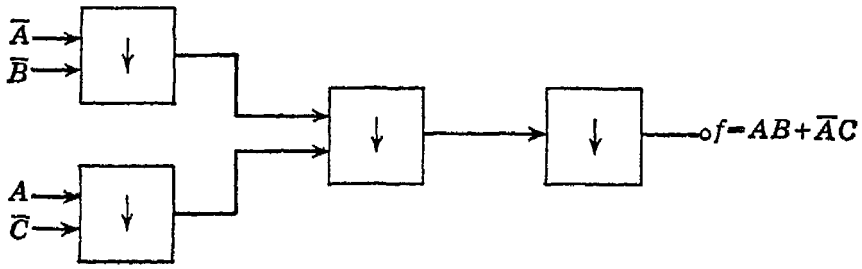



Fig. 10.43 A three-level NOR circuit for $f = AB + \bar{A}C$

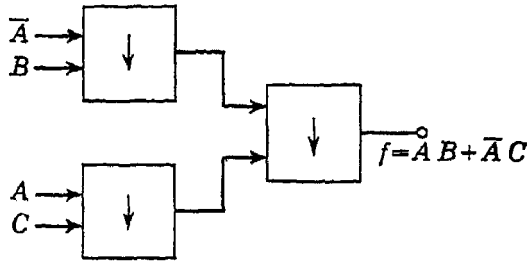



Fig. 10.44 A two-level NOR circuit for $f = AB + \bar{A}C$

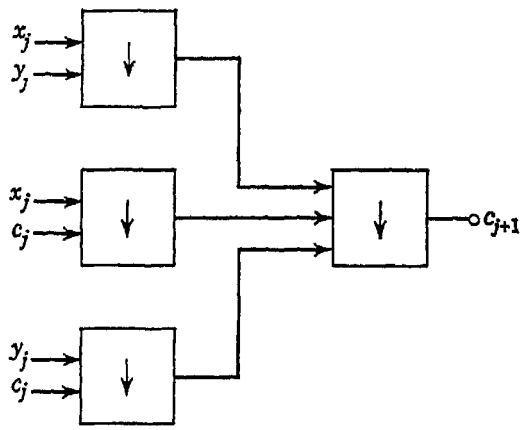



Fig. 10.45 A two-level NOR circuit for the carry digit $c_{j+1}$

Now let us complete the three-input adder circuit by constructing a NOR realization of the carry digit $c_{j+1}$. Equation (10.112) may be rewritten as follows:

$$c_{j+1} = (x_j + y_j)(x_j + c_j)(y_j + c_j) \tag{10.125}$$
$$= \overline{x_j + y_j} \downarrow \overline{x_j + c_j} \downarrow \overline{y_j + c_j} \tag{10.126}$$
$$= \overline{\overline{x_j \downarrow y_j}} \downarrow \overline{\overline{x_j \downarrow c_j}} \downarrow \overline{\overline{y_j \downarrow c_j}} \tag{10.127}$$
$$= (x_j \downarrow y_j) \downarrow (x_j \downarrow c_j) \downarrow (y_j \downarrow c_j) \tag{10.128}$$

A two-level NOR circuit for implementing this function is shown in Fig. 10.45.

## REFERENCES

*Items Referred to by Number:*


1. Elbourn, R. D., and R. P. Witt, Dynamic circuit techniques used in SEAC and DYSEAC, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1380–1387.
2. Burks, A. W., and I. M. Copi, The logical design of an idealized general-purpose computer, *Univ. Mich. Eng. Research Inst. Rept. No.* 1828-11-T, Oct. 30, 1954.
3. Rubinoff, M., Notes on the design of Eccles-Jordan flip-flops, *Trans. AIEE*, vol. 71, pt. I, July, 1952, pp. 215–220.
4. Ritchie, D. K., The optimum dc design of flip-flops, *Proc. IRE*, vol. 41, no. 11, November, 1953, pp. 1614–1617.
5. Vogelsong, J. H., A transistor pulse amplifier using external regeneration, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1444–1450.
6. Simkins, Q. W., and J. H. Vogelsong, Transistor amplifiers for use in a digital computer, *Proc. IRE*, vol. 44, no. 1, January, 1956, pp. 43–55.
7. Harris, J. R., Direct-coupled transistor logic circuitry, *IRE Trans. on Electronic Computers*, vol. EC-7, no. 1, March, 1958, pp. 2–6.
8. Easley, J. W., Transistor characteristics for direct-coupled transistor logic circuits, *IRE Trans. on Electronic Computers*, vol. EC-7, no. 1, March, 1958, pp. 6–16.
9. Rowe, W. D., and G. H. Royer, Transistor NOR circuit design, *Trans. AIEE*, vol. 76, pt. I, July, 1957, pp. 263–267.
10. Marcovitz, M. W., and E. Seif, Analytical design of resistor-coupled transistor logical circuits, *IRE Trans. on Electronic Computers*, vol. EC-7 no. 2, June, 1958, pp. 109–119.
11. Rubinoff, M., A new family of transistor switching circuits, *Trans. AIEE*, vol. 76, pt. I, July, 1957, pp. 286–289.
12. Ebers, J. J., and J. L. Moll, Large-signal behavior of junction transistors, *Proc. IRE*, vol. 42, no. 12, December, 1954, pp. 1761–1772.
13. Domenico, R. J., Simulation of transistor switching circuits on the IBM 704, *IRE Trans. on Electronic Computers*, vol. EC-6, no. 4, December, 1957, pp. 242–247.
14. Kingston, R. H., Switching time in junction diodes and junction transistors, *Proc. IRE*, vol. 42, no. 5, May, 1954, pp. 829–834.
15. Moll, J. L., Large-signal transient response of junction transistors, *Proc. IRE*, vol. 42, no. 12, December, 1954, pp. 1773–1784.
16. Linvill, J. G., Nonsaturating pulse circuits using two junction transistors, *Proc. IRE*, vol. 43, no. 7, July, 1955, pp. 826–834.
17. Grisamore, N. T., L. S. Rotolo, and G. Y. Uyehara, Logical design using the stroke function, *IRE Trans. on Electronic Computers*, vol. EC-7, no. 2, June, 1958, pp. 181–183.
18. Kintner, P. M., Dual-polarity logic as a design tool, *IRE Trans. on Electronic Computers*, vol. EC-8, no. 2, June, 1959, pp. 227–228.


*Supplementary References:*


Henle, R. A., and J. L. Walsh, The application of transistors to computers, *Proc. IRE*, vol. 46, no. 6, June, 1958, pp. 1240–1254.

Prom, G. J., and R. L. Crosby, Transistor switching circuits for high-speed computers, *IRE Trans. on Electronic Computers*, vol. EC-5, no. 4, December, 1956, pp. 192–196.

Morris, E. F., and T. E. Wohr, Automatic implementation of computer logic, *Communs. ACM*, vol. 1, no. 5, May, 1958, pp. 14–20.

Wallmark, J. T., and S. M. Marcus, Integrated devices using direct-coupled unipolar

transistor logic, *IRE Trans. on Electronic Computers*, vol. EC-8, no. 2, June, 1959, pp. 98–107.

Cloot, P. L., A basic transistor circuit for the construction of digital-computing systems, *Proc. IEE London*, vol. 105, pt. B, no. 21, May, 1958, pp. 213–220.

## PROBLEMS

**10.1** Gating devices, such as those shown in Fig. 10.1, should have only two levels for their output voltages. Practically, however, the circuit of Fig. 10.1*a* may have a range of voltages over which its output may be considered to be at the low level. The upper limit of this range results when only one triode conducts and the lower limit when all conduct. If type 6J6 triodes are used with $E_{bb} = 250$ volts and $R_L = 10$ kilohms, what will be the respective output levels for one, two, three, and four triodes conducting, assuming that the grids swing from cutoff to ground level? What would be the effect upon the spread of these levels if (*a*) the value of $R_L$ were raised to 25 kilohms, or (*b*) the grids were driven to +6 volts instead of merely to ground? Comment on the advantages and disadvantages of these two courses of action.

**10.2** A situation analogous to that of Prob. 10.1 occurs in the circuit of Fig. 10.1*b*. Here, however, the spread of voltages occurs around the high level. If type 6J6 triodes are used with $E_{bb} = 250$ volts and $R_k = 1{,}000$ ohms, what will be the output levels for one, two, three, and four triodes conducting, assuming that the grids swing from cutoff to +5 volts?

**10.3** Assume that the diode of Fig. 10.1*d* can be approximated as a capacitance of 10$\mu\mu$f when reverse potentials are applied across it. If $R_L = 50$ kilohms and terminal $A$ is at ground level, what will be the output voltage as a function of time if a +10-volt step is applied to point $B$? ($B$ is initially at ground level.)

**10.4** Assume, in the AND gate shown, that the input diodes are perfect but the clamp diode has an effective forward resistance of 100 ohms. What will be the output voltages corresponding to zero, one, two, three, and four of the input diodes cut off by input signals?

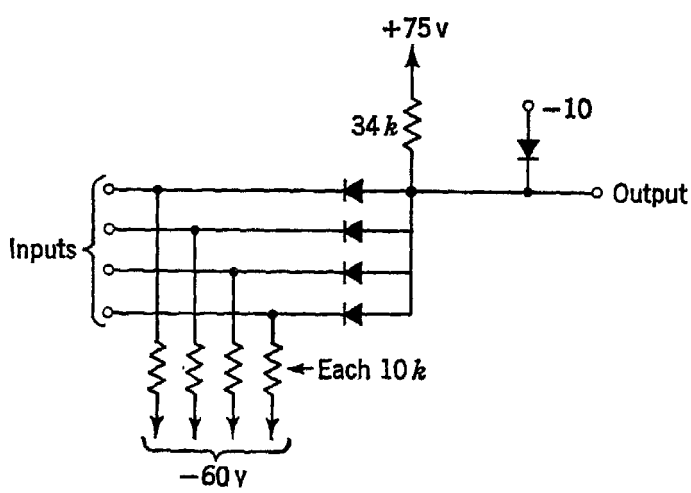



Fig. P10.4

**10.5** A clock signal and two input pulses $A$ and $B$, as shown, are applied to a set of the AND inputs of Fig. 10.12. Assuming that time delays in the circuit can be neglected, show to the same time scale the shape of the output pulse (*a*) when the regeneration connection is not present, and (*b*) when the regeneration connection is present. What effect would inclusion of the time delays have on your sketch?

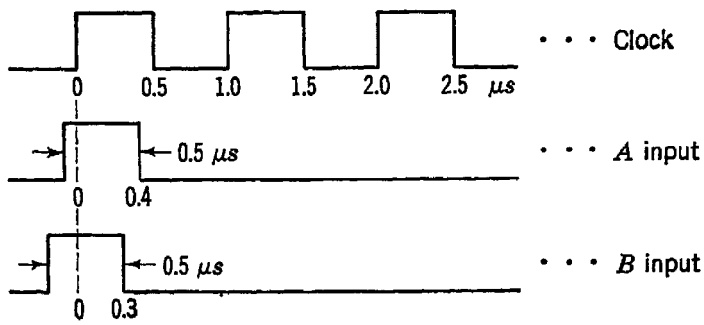



Fig. P10.5

**10.6** Derive Eqs. (10.22) and (10.23).

**10.7** Use the design method described in Sec. 10.7 to design a flip-flop having the following specifications:

| | |
|---|---|
| Tube | 6J6 dual triode |
| $+E_1$ | +250 volts |
| $-E_2$ | −150 volts |
| $\rho$ | 5 per cent |
| $\sigma$ | 5 per cent |
| $+e_1$ | +5 volts |
| $-e_2$ | −10 volts |
| $I_b$ | 8 ma |
| $E_b$ | 150 volts |

What would be the effect (qualitative) upon the switching rate if the foregoing specifications were altered to call for $I_b = 10$ ma minimum with no more than 150 volts for $E_b$?

**10.8** For the grounded-emitter switch of Fig. 10.32, calculate the turn-on time for the following values of base current $I_B$: (*a*) ½, (*b*) 1, (*c*) 5, and (*d*) 10 ma. Assume $\alpha_N = 0.9$, $\omega_N = 10^7$, $V_{CC} = 10$ volts, $R_L = 1{,}000$ ohms. What is the final value of $I_C$ in each case?

Repeat these calculations for a transistor having $\alpha_0 = 0.98$ and $\omega_N = 75 \times 10^7$, values more typical of computer transistors.

CHAPTER 11

# Storage in the Digital Computer

## 11.1 The storage hierarchy

The storage function in a digital system is organized in a hierarchy of levels, from the fastest and most complex at one extreme to the slowest and least complex at the other. The fastest elements, and also the most complex and expensive, are the vacuum-tube or transistor flip-flops used for storage of single bits of information. Single flip-flops are used at many points to control other circuits, and groups of flip-flops are often arranged in registers to hold a single word or a few words. The flip-flop has such high cost per stored bit, however, that its use for bulk storage is avoided. Bulk storage of program and data is accomplished at the next level by means of a moderately large array of passive elements, which in turn transfers its contents at intervals to or from a larger and slower storage system. This in its turn may be backed up by other successively slower levels of storage. In this chapter we shall direct most of our attention to the various systems used for the bulk-storage function.

## 11.2 Properties of common types of storage systems

The names of many of the more widely used types of storage systems are probably already familiar to the reader. A tabulation of the more important properties of these systems provides a useful point of departure for a detailed study of the engineering considerations in their design.

Probably the most significant characteristic of a memory system is the speed with which a word can be read out from or written into storage. The total time for reading or writing usually has two components: the time required for establishing connections to the desired cell, and the time required for action of the individual bit-storing elements of the cell. In a "random-access" memory, all cells are equally accessible, and the time required for establishing connections to the desired cell is the same for all cells. In a "cyclic" memory, information circulates in a loop (or in many loops), and each cell is accessible only as it passes the reading or writing point. Although the cyclic system has the advantage of simpler

access circuitry than the random-access system, the access time is ordinarily longer, since on the average half a cycle must elapse before the desired cell passes the read-write point.

In some types of storage system, the act of reading the information from a cell puts all bit positions of the cell in the same state, thus destroying the stored information. If it is desired to retain the information in storage, systems characterized by this destructive read-out must use extra circuitry or time or both to restore the memory cell to its previous

**Table 11.1**
**Characteristics of storage systems**

| | Typical access time | Cyclic? | Random access? | Volatile? | Destructive read-out? |
|---|---|---|---|---|---|
| Magnetic drums | 1–30 msec | Yes | No | No | No |
| Electrostatic tubes | 10 μsec | No | Yes | Yes | Yes |
| Acoustic delay lines | 50–100 μsec | Yes | No | Yes | No |
| Magnetic cores | 1–10 μsec | No | Yes | No | Yes, in most present systems (1959) |
| Magnetic tape | Note 1 | Note 2 | No | No | No |
| Punched tape | Up to several seconds | Note 2 | No | No | No |
| Punched cards | Up to several minutes | Note 2 | No | No | No |
| Photographic film | .....Note 3..... | | | No | No |

1. Although several minutes may be required for complete search of a full 2,400-ft reel of tape, systems using tapes ordinarily compute while searching.

2. Although these systems are not ordinarily cyclic, they have the cyclic system's characteristic that the individual cells are not equally accessible.

3. Photographic films are suitable for permanent storage, since stored information cannot be readily altered. They can be organized for random-access or for cyclic reading. Access times of 1 to 30 msec are realizable in cyclic systems.

condition. The magnetic core, which is at present the most widely used high-speed storage device, is characterized by destructive read-out when it is used in the simplest manner, and efforts are under way in many laboratories to find practical ways of using cores in a nondestructive mode.

Storage systems in which the stored information gradually deteriorates with time are said to be "volatile" systems. If information is to be retained in storage for longer than the decay time during which the pattern is recognizable, additional circuitry or time or both must be provided to regenerate the stored information. Furthermore, when the power is turned off, all information stored in a volatile system vanishes. By contrast, a nonvolatile system, such as a magnetic drum or magnetic cores, retains its stored information after removal of power.

In Table 11.1, several types of memory systems are listed, and their properties are tabulated.

## 11.3 The storage property

A memory element for the storage of a binary digit can be made from any material or device that displays a hysteresis loop in its response versus excitation characteristic. Thus, a toggle switch is a memory device, but

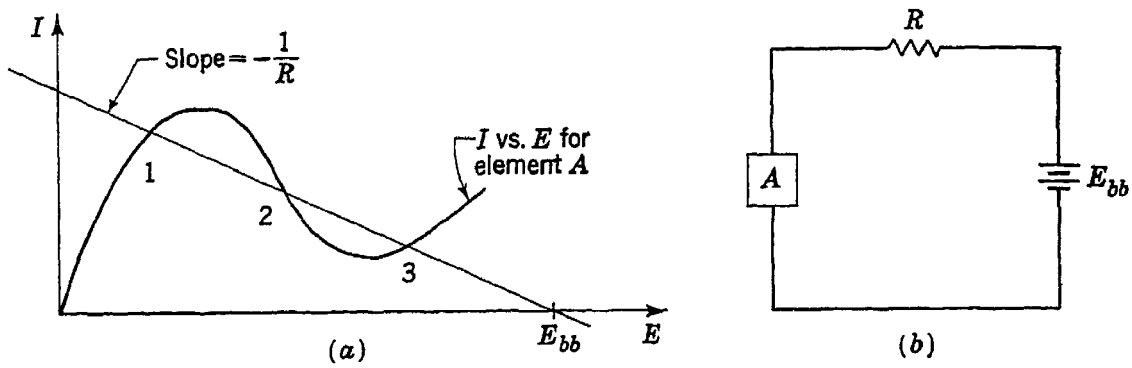



Fig. 11.1 A bistable circuit

a simple push-button switch is not. Permanent magnetic materials are obviously well suited for storage and have come to be the most widely used form of memory system. Examples of many other devices displaying hysteresis loops will occur to the reader, and the exercise of "inventing" storage devices will provide considerable insight into the subject. Patterns of charge placed on an insulating surface in a cathode-ray tube have been used for storage. Dielectric materials that display a hysteresis loop in the curve of electric displacement density $D$ versus electric intensity $E$ have been used experimentally. Systems of capacitors, each either charged or uncharged, have been used, and there are undoubtedly other devices or physical phenomena not yet explored that can provide bases for storage systems.[1]

It is interesting to note that an element having a hysteretic response can be constructed from a linear element in combination with an element having a suitable nonlinearity. For instance, certain circuits can be made to have an $I$ versus $E$ curve like that shown in Fig. 11.1$a$. If such a device is placed in series with a suitable resistor $R$ and d-c supply $E_{bb}$, the combined circuit has three points of equilibrium, marked on the curve. However, point 2 represents an unstable equilibrium, since any minute perturbation of the current from the conditions at point 2 will cause increasing changes, which will cause the circuit to change to point 1 or to point 3, depending upon the direction of the initial perturbation. Points 1 and 3 are both points of stable equilibrium. Once placed in either of these two conditions, the circuit will remain in that condition

until a disturbing force is applied sufficient to carry it beyond the point of unstable equilibrium. The conventional Eccles-Jordan flip-flop circuit can be analyzed in this manner, and many flip-flops have been described in the literature that employ a single active element (tube or transistor) having a characteristic like that shown in Fig. 11.1.[2] Thus the engineer seeking new types of storage elements should not overlook the possibilities inherent in elements that have interesting nonlinearities.

## 11.4 Storage using magnetic cores[3–8]

Certain magnetic materials display a nearly rectangular hysteresis loop in their $B$-$H$ characteristic, as shown in Fig. 11.2. In such materials the remanent flux density $B_r$ is only slightly less than the maximum flux density $B_m$. The ratio $B_r/B_m$ is in the range 0.92 to 0.98 for most of the "square-loop" materials. These materials are also characterized by steep slopes in the vicinity of $B = 0$; that is, their incremental permeabilities are high, and only a small change in magnetic intensity $H$ is required to produce a large change in $B$. A toroidal core of such a material, linked by suitable windings, makes an excellent element for storage of a single bit of information. The configuration shown in Fig. 11.3 might be used. This figure shows a single input winding through which current passes in one direction to store or "write" a 1 and in the other direction to write a 0. Reading out the information stored in the core is complicated by the fact that a magnetic flux manifests itself only when it is changing.* Thus read-out could be accomplished by applying a current to the input winding in the 0 direction. If the core is in the 1 state, this 0 current will reverse its magnetization, and the resulting flux change will induce a voltage in the output winding. The polarity of this voltage will be such as to make terminal 2 positive with respect to terminal 1. If the core is already in the 0 state, the current in the 0 direction will cause a

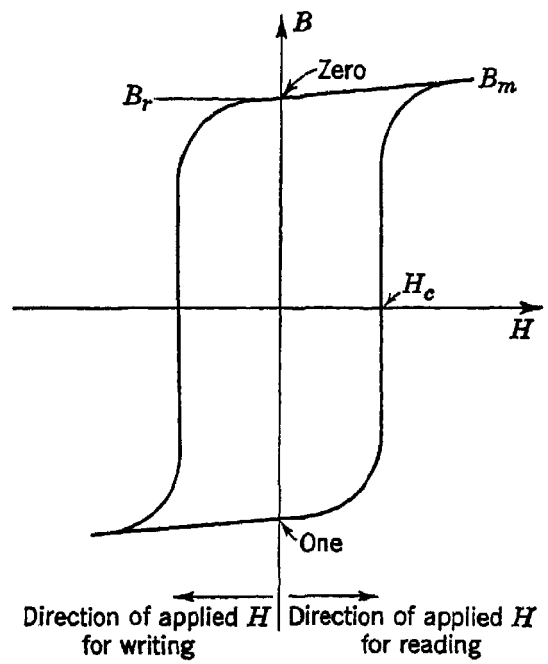



Fig. 11.2 A "rectangular" hysteresis loop

* An exception is the Hall effect: if a conductor carries a current in a magnetic field, a voltage is produced at right angles to both and proportional to both. The effect is small in most conducting materials and has not been used for magnetic-core read-out.

small flux change, from $B_r$ to $B_m$, and then back to $B_r$ as the current falls back to zero. The resulting small voltage is of the same polarity as for the 1, but is much smaller in magnitude and easily distinguishable.

The principal difficulty met in using this simple storage element in the way just described is the problem of selecting the desired core, for reading or writing, from among an array of cores. In an array containing $n^2$ cores, each would have a separate input winding, and external circuitry would need to be furnished to select one such winding from among $n^2$. It is possible, however, to effect a considerable reduction in the external

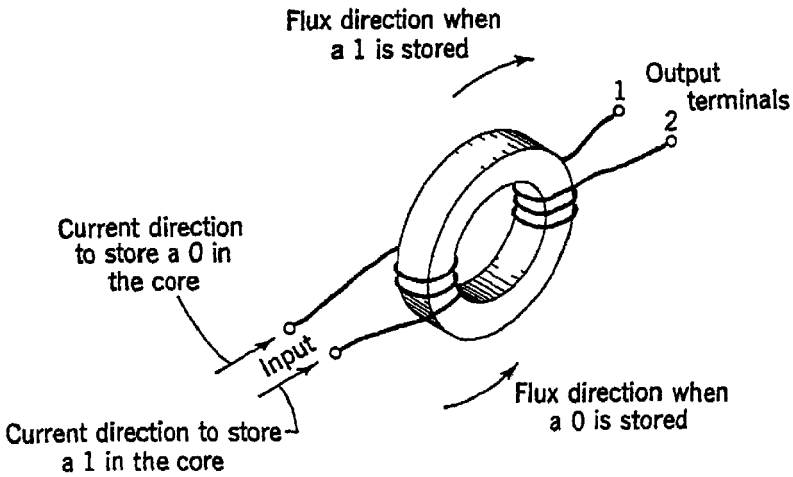



Fig. 11.3 A magnetic-core storage element

selecting circuitry by taking advantage of the rectangularity of the hysteresis loop and allowing the cores themselves to participate in the selection function. For cores with nearly rectangular hysteresis loops, it is possible to choose a value of current $I_s$ in the input winding large enough to switch the flux density completely from $-B_r$ to $+B_m$ but small enough so that $I_s/2$ units applied to the input will not carry the core beyond the knee of the hysteresis loop. If each core is linked by two windings, as shown in Fig. 11.4, then application of $I_s/2$ units of current to a given row and a given column will create enough magnetic field strength $H$ to switch only the core at the intersection of the selected row and column. The other cores in the selected row and column are only partially excited and ideally should undergo no flux changes. A square array of $n^2$ cores requires $n$ row drives and $n$ column drives, and in each case a selection of one out of $n$ drive wires must be made. This is a considerably simpler problem than that of furnishing $n^2$ drive wires and making a selection externally of one out of $n^2$.

Since only the one core selected can undergo a flux reversal, a sense winding common to all cores is used. If all cores had perfectly rectangular loops, the presence of any voltage at the ends of the sense winding would indicate a flux reversal in the selected core. Unfortunately, the

cores are not perfectly rectangular, and each of the partially excited cores on the selected row and on the selected column undergoes a slight flux change. To prevent these slight changes from inducing voltages of the same polarity in the sense winding, the winding may be passed through the array in some such pattern as that shown in Fig. 11.4. This pattern allows the sense winding to link half the cores in each row and in each column in one direction and the other half in the opposite direction, thus

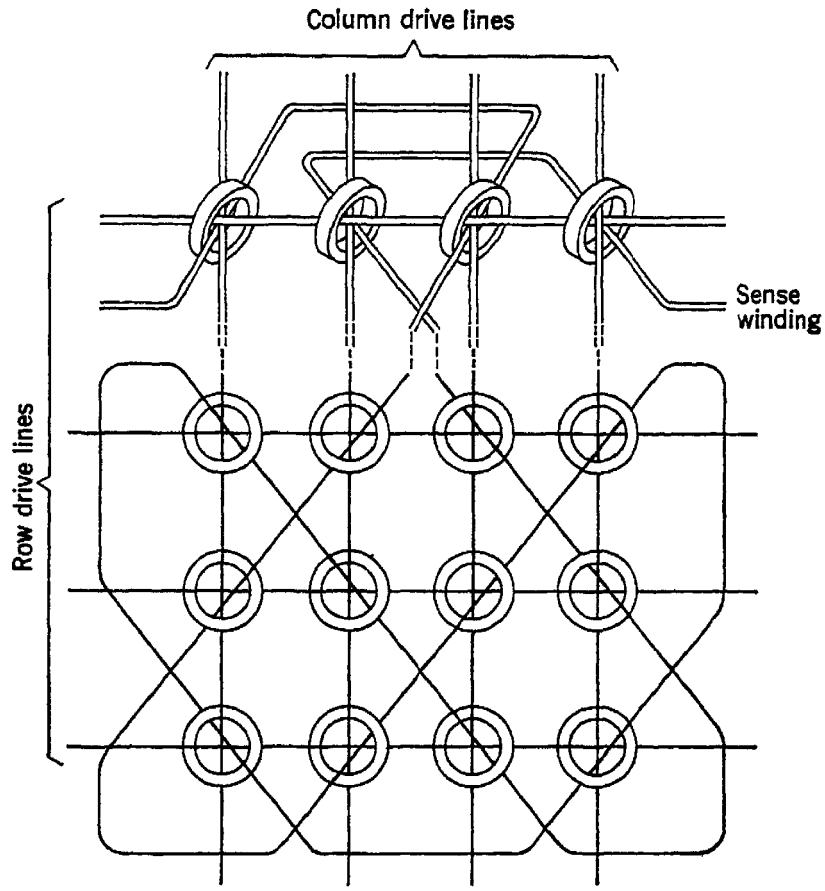



Fig. 11.4 A 4 × 4 coincident-current matrix

greatly reducing the net voltage contributed by the disturbed cores. It also minimizes the direct coupling between the drive wires and the sense winding. This coupling, which has both inductive and capacitive components, is due simply to the proximity of the drive wires and the sense winding. It will be noted that the sense winding in Fig. 11.4 passes through the cores in such a way as to be always *behind* the drive wires. If the sense winding is inadvertently placed behind the drive wires in some cores and in front of them in others, then some drive wires link the sense winding and can cause appreciable stray-voltage pickup.

The selection system in Fig. 11.4 allows selection of one core at a time from among those in the matrix. For high-speed computer applications it is desirable to read all the bits in a word simultaneously (i.e., in parallel)

rather than serially. A widely used way of accomplishing parallel read-out of a word is to employ as many core memory planes as there are digits in the word. One plane stores the least-significant digits of all words, the next plane the next more significant digits, and so forth. A single word has its bits stored at coordinates $(j,k)$ in all the planes. The "stack" of planes is excited by connecting all the $j$th rows in series and all

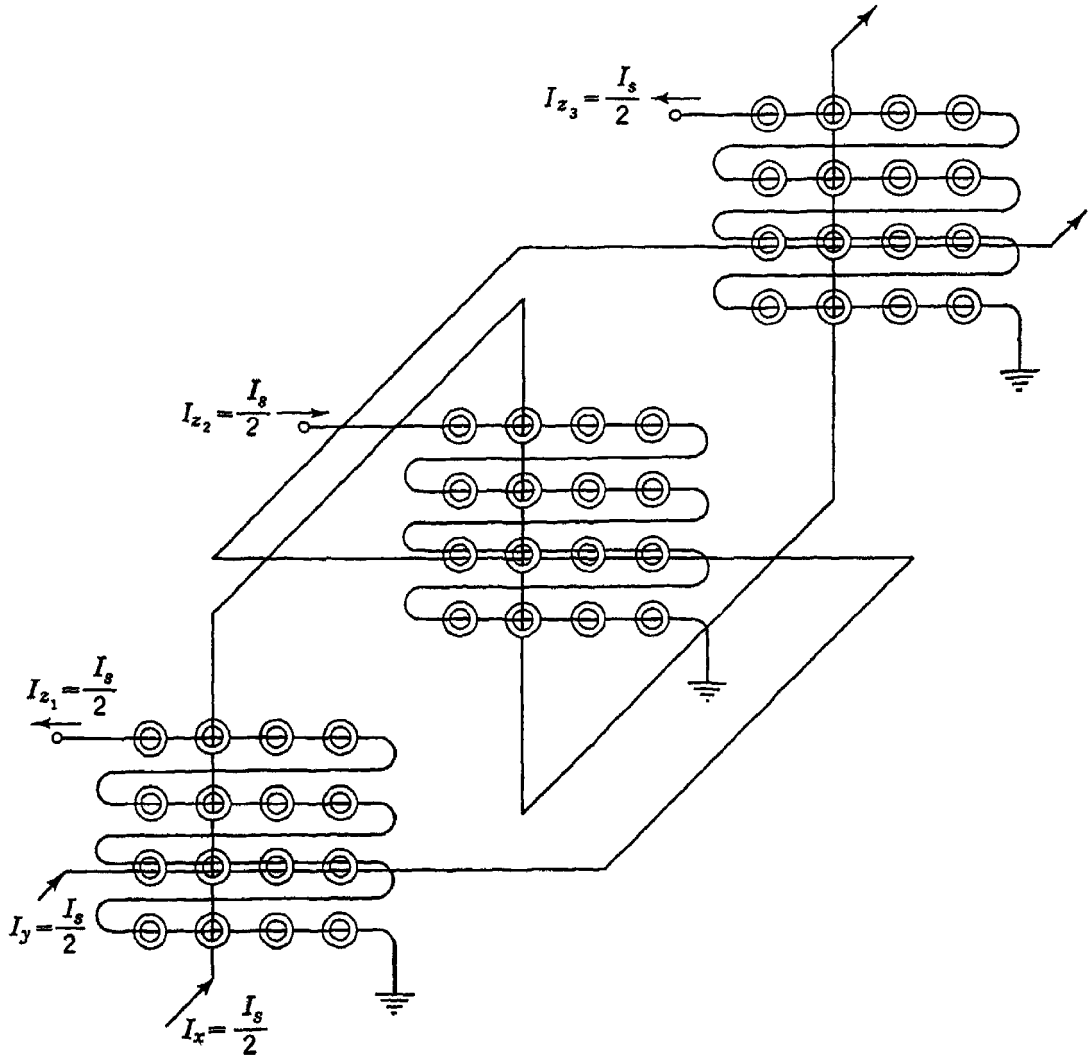
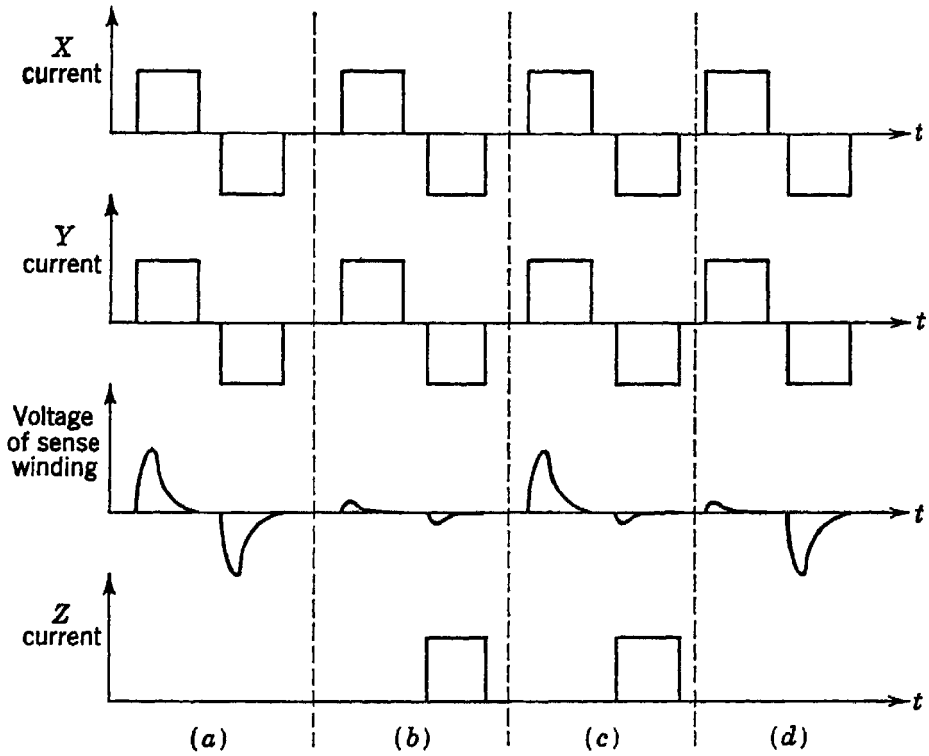



Fig. 11.5 Organization of memory planes for parallel read-out

the $k$th columns in series, as shown in Fig. 11.5, so that selection of a row and of a column simultaneously selects one core in each plane. Each plane has its own sense winding, following the pattern of Fig. 11.4. (These windings have been deliberately omitted from Fig. 11.5 to avoid confusion. Similarly, the $X$ and $Y$ drive lines have been shown for only one line in each coordinate.)

As pointed out earlier, the act of reading information from a core element destroys that information if the core is switched by the reading excitation. In order to restore the information, it is common practice to reverse the $X$ and $Y$ drive currents after read-out and to apply $-I_s/2$

units of current to the selected lines. Any core that was switched from one remanent flux state to the other by the reading current will then be switched back to its original state during the writing portion of the cycle. Unfortunately, cores that were not switched by the reading current, as well as those that were switched, will be set into the opposite state by the writing currents. To prevent this undesired switching, the $Z$ windings shown in Fig. 11.5 are driven with a current of $I_s/2$ units in such a




(a) Read 1, restore 1
(b) Read 0, restore 0
(c) Read 1, restore 0
(d) Read 0, restore 1

Fig. 11.6 Excitation patterns of $X$, $Y$, $Z$, and sense windings

direction as to oppose the switching during the writing part of the cycle. Only those planes from which a 0 was obtained in the reading part of the cycle are driven by an inhibiting current in the $Z$ winding during writing. A bank of flip-flops, each connected to one of the $Z$-driver circuits, receives the information obtained on read-out and allows the proper $Z$ drivers to be activated during the writing cycle. If new information is to be written into the cores, this information can be stored in the flip-flop bank instead of the digits obtained during the reading portion of the cycle. Excitation patterns for the various windings are shown in Fig. 11.6. The duration of the read-write cycle varies from one computer to another, the fastest (as of 1958) being about 6 $\mu$sec and the slowest 10 to 20 $\mu$sec. Some machines, expected to be completed in 1960, will probably achieve 2-$\mu$sec cycles, and research efforts directed toward a cycle time of 0.1 $\mu$sec are under way in several laboratories.

## 11.5 Some engineering problems in core memories

Although the discussion of the preceding section suffices for a simple understanding of the elementary principles of core storage, a more detailed description of the phenomena involved will give the reader a better grasp of some of the difficult engineering problems encountered in the design of magnetic-core systems. Nonrectangularity of the hysteresis loops is a characteristic that creates several engineering problems, since cores subjected to partial excitations will traverse small minor loops. Not only does each such traversal require energy from the driving source, but in addition the small flux changes induce undesired voltages in the sense winding. In any but the smallest matrices, the energy consumed by all the partially excited cores on the selected row and column is substantially greater than that required to switch the selected core, and the total voltage induced by all the partially excited cores would mask the voltage produced by the selected core were it not for the partial cancellation resulting from the alternating pattern of the sense winding.

Figure 11.7 shows a hysteresis loop and some of the minor loops that are encountered.[9] A core through which the last previous current was a full $I_s$ units in the writing direction is left at $a$ on the loop (Fig. 11.7$c$). Application of a half-selecting current $I_s/2$ in the reading direction will drive it momentarily to $b$ and then back to $c$ at the end of the current pulse. The writing action during the next half of the cycle will drive it to $d$ and back to $e$. Thus, each time a core is half-selected by receiving a current of $I_s/2$ on only one of its two coordinate lines, it traverses a slightly different hysteresis loop. The magnitude of the disturbing voltages induced in the sense winding depends upon the action cycle to which the cores have been subjected. It will also be observed that each HALF-READ HALF-WRITE cycle to which a core is subjected leaves it at a slightly lower remanent flux density. For a core to be usable in a storage array, this "walking up the loop" must converge toward a fairly high value of $B_r$ and must not continue to reduce $B_r$ toward zero. The decrease in the remanent flux density also means that, when a core storing a 1 is fully selected, the induced voltage in the sense winding is appreciably less than from an undisturbed core resting at $a$.

A similar family of minor loops is also observed in the neighborhood of the remanent flux density representing a stored 0. It is impossible, however, for a core to be left at $j$ after a complete READ-WRITE cycle, since the last current applied is a net current of $I_s/2$ units in the writing direction. Thus point $l$ represents the state of a core in which the last previous action was the storing of a 0. Successive READ-WRITE cycles reduce the remanent flux density. The result is that, when the core is fully selected to read its stored information, the net change in flux is larger than if it

were at point $j$. Traversal of the minor loops and the resulting decrease in $B_r$ thus make the 1 voltage smaller and the 0 voltage larger and so complicate the problem of discriminating between them.

Fortunately, the distinction between the 1 voltage and the 0 voltage need not be made on the basis of amplitude alone, since the 1 voltage

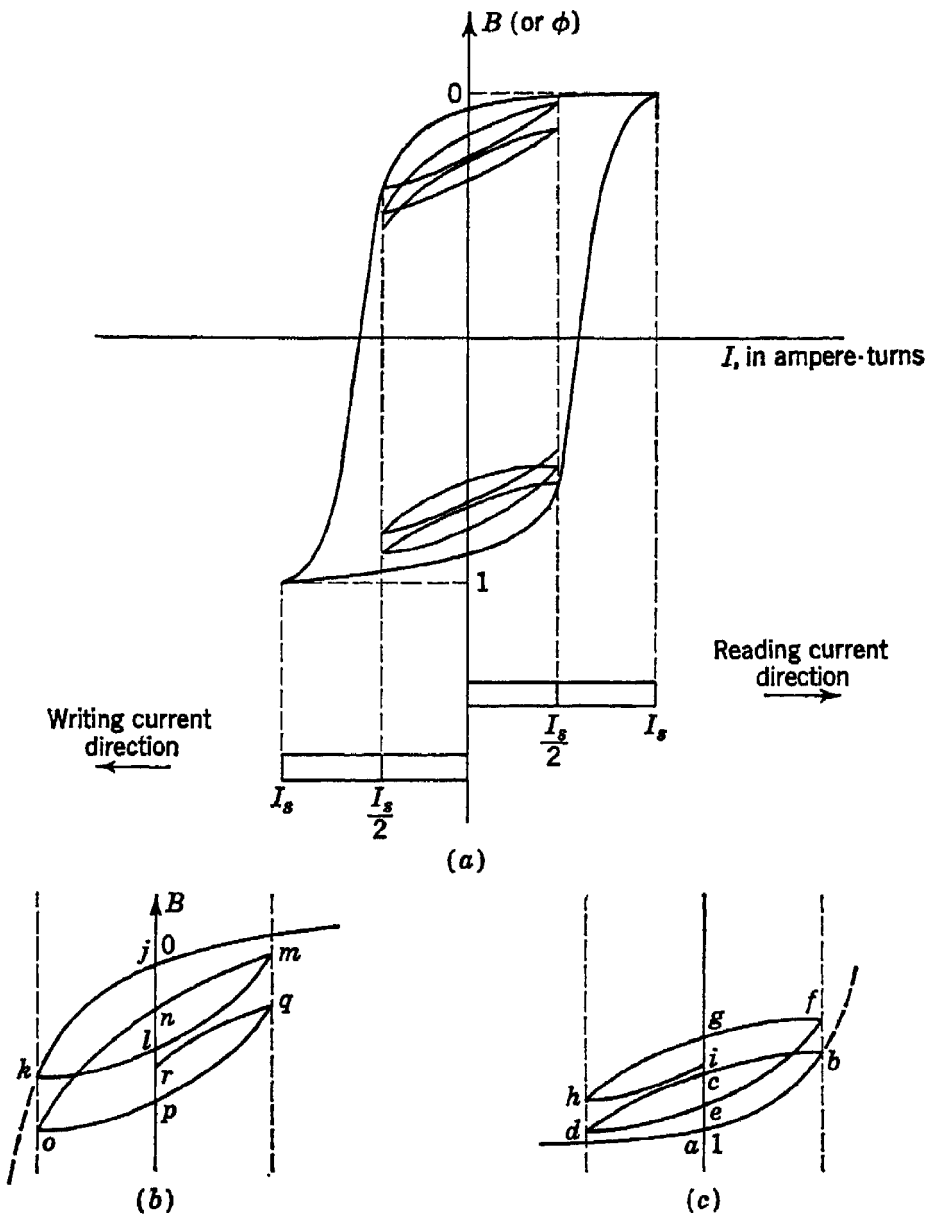



Fig. 11.7 Minor loops traversed by cores: (a) major loop and minor loops, (b) minor loops in vicinity of stored 0, (c) minor loops in vicinity of stored 1

reaches its peak value somewhat later than the 0 voltage does. The reasons for this lie in the nature of the flux-reversal process. Typical observed voltage waveforms for a 1 and for a 0 are shown in Fig. 11.8.[10] Thus, if the sensing amplifier is normally gated OFF and is gated ON for only a short interval at the time when the 1 voltage should be at its peak, the discrimination between 1 and 0 is made much more easily than by

merely comparing peak amplitudes. For cores with switching times of a few microseconds, the "strobing" or gating pulse applied to the sensing amplifier typically has a duration of 0.1 μsec. By means of the strobing technique, it is possible to achieve a 10:1 amplitude discrimination between 1's and 0's that differ in peak amplitudes by only 3:1 or 4:1.

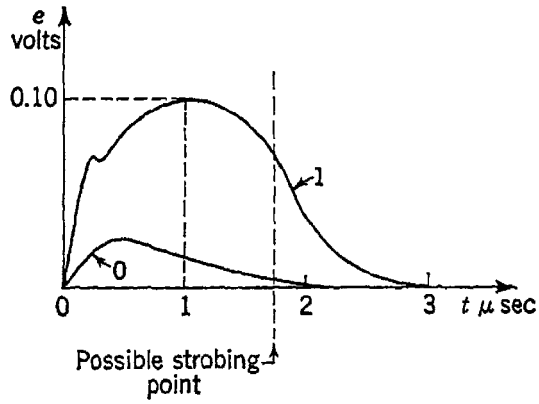


Fig. 11.8 0- and 1-voltage waveforms

In large arrays, strobing is complicated by the time delay in the long sense winding, and the optimum time for application of the strobing pulse varies from core to core, depending upon the core's position in the array. In the largest core memory in existence (in 1960), each plane is 256 × 256 cores and is made up of 16 smaller planes, each 64 × 64 and each having its own sense winding brought out separately to reduce the time delay. (This memory system, which has 37 planes, was constructed at the Lincoln Laboratory of the Massachusetts Institute of Technology.[11])

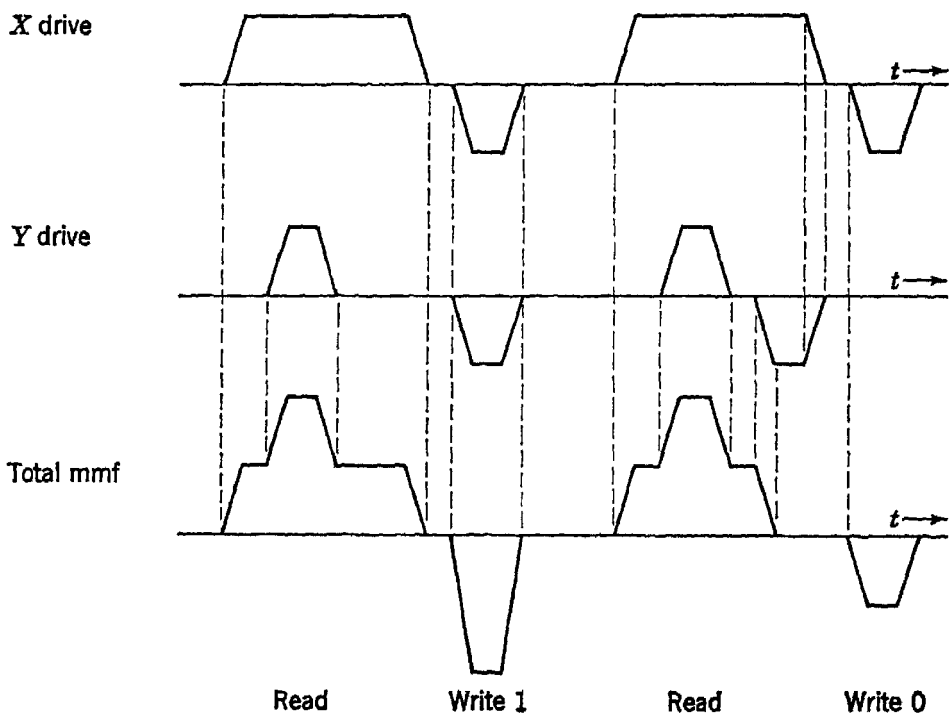


Fig. 11.9 An excitation cycle to reduce disturbing voltages

Some core storage systems use a driving cycle that greatly simplifies the wiring of core matrices and the problems of disturbing voltages, but at the expense of an increase in cycle time.[12] The $X$ and $Y$ excitations are shown in Fig. 11.9. The $X$ drive is applied first; after transient disturb-

ances produced by the $X$ drive have substantially diminished, the $Y$ drive is applied. Strobing of the sensing amplifier occurs after application of the $Y$ drive; therefore, the disturbances produced by the $X$ drive do not appear in the output. Thus there is no need to interlace the sense winding in the manner of Fig. 11.4, which achieves cancellation of both $X$ and $Y$ disturbances. The sense winding can now be placed in parallel with the $X$-drive lines; this greatly simplifies the wiring of the matrices. Since the $Y$ drives are now the only drives that produce disturbances, it is advantageous to make the array merely rectangular, instead of square, by putting more cores on each $X$ line and fewer on the $Y$ lines. A $64 \times 64$ array could be conveniently replaced by an array $32 \times 128$. Each $X$-drive wire would pass through a row of 128 cores, and each $Y$-drive wire through a column of 32. In the $64 \times 64$ array, 123 cores would contribute disturbance voltage components on the sense winding, but in the equivalent nonsquare rectangular array, only 31 cores would contribute disturbances. In practice, the $Y$ drives are separately controlled for the various planes. In this way, 1's or 0's can be selectively written without the use of an inhibiting winding.

## 11.6 Properties of magnetic materials[13–16]

Square-loop magnetic materials used as switches and as storage elements in computers are of two types: ferrites and metals. The ferrites are ceramic-like crystalline materials formed at high temperatures from pulverized mixtures of iron oxide and certain other metallic oxides. The metallic materials are high-permeability alloys of such metals as iron, cobalt, nickel, and molybdenum. These alloys, when rolled as thin as ¼ to ⅛ mil (0.00025 to 0.000125 in.) and subjected to careful annealing, display a highly rectangular hysteresis loop. The metals, as a class, display considerably higher saturation flux densities than do the ferrites. Their coercivity, the magnetic intensity $H_c$ required to reduce the flux density to zero, is lower than that of the ferrites, with the result that the loops for the two materials have the relative shapes shown in Fig. 11.10. The saturation flux density $B_s$ is in the range from 8,000 to 16,000 gauss for typical magnetic metal alloys, and in the vicinity of 2,000 gauss for typical ferrites. The coercivity $H_c$

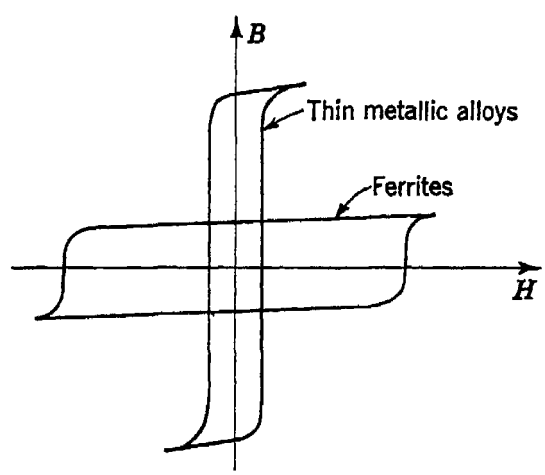



Fig. 11.10 Comparative hysteresis loops for metals and ferrites

of the metal alloys is around 0.1 to 0.2 oersted; for ferrites the values are approximately ten times as great, the range being about 0.5 to 2 oersteds.

Although much of our discussion in this chapter has made reference to "square-loop" materials, it should be pointed out that the loops may be

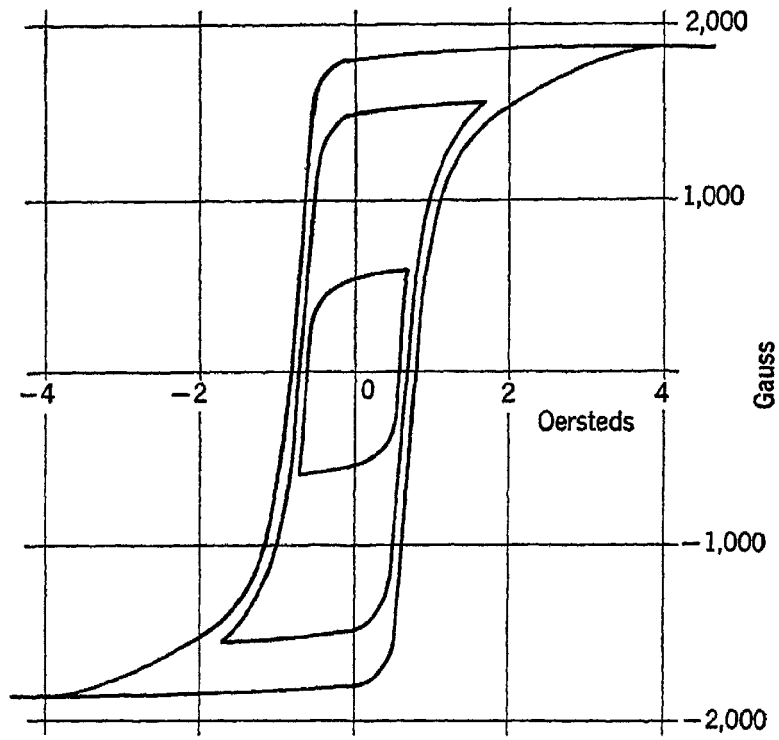



Fig. 11.11 A family of hysteresis loops for a typical ferrite

far from square and yet be usable. Each material has a family of hysteresis loops corresponding to different values of maximum applied magnetic field. Figure 11.11 shows such a family for one typical ferrite material. It will be noted that the outermost loop, which is the saturation loop, has a long "tail" at high flux densities. Such a loop is not suited for use in a coincident-current selection system, since application of half the maximum magnetic field is more than enough to cause switching of the flux density $B$ from one direction to the other. Instead, one of the interior loops is used. The best loop to use is that which allows the maximum ratio between the flux change under full selection and the flux change under half-selection. Such a choice provides the largest possible voltage read-out for a stored 1 per unit of disturbing voltage.

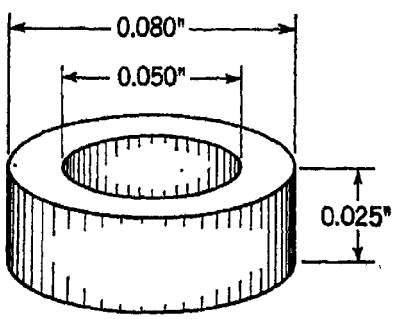


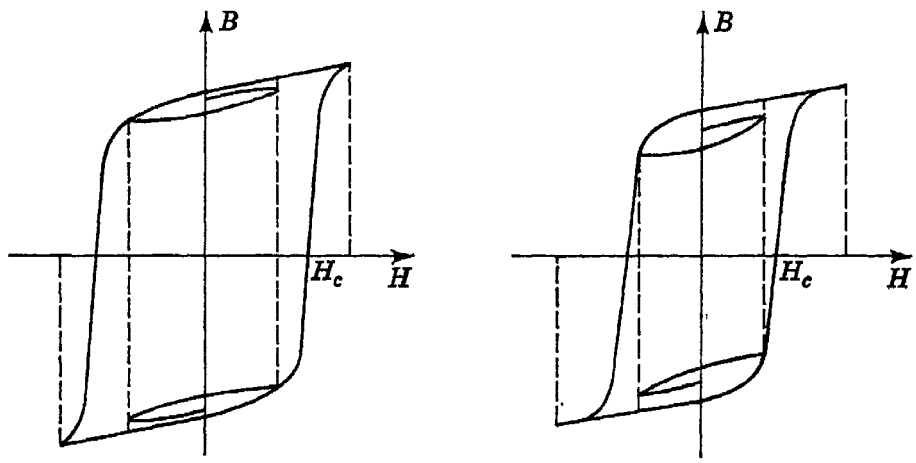

Fig. 11.12 Dimensions of a typical ferrite memory core

It is of interest to determine the approximate energy loss, due to hysteresis, for a typical ferrite core. Let us assume a core of the dimen-

sions shown in Fig. 11.12. Many storage systems have been built using cores of these dimensions. A smaller size, 0.050 in. OD and 0.030 in. ID, is used in some higher-speed systems. Let us also, for simplicity, assume that the core is operated on a perfectly rectangular loop having $B_m$ of 1,500 gauss and $H_c$ of 1.5 oersteds. The energy loss per cycle is given by

$$W = 4B_m H_c \qquad \text{joules/unit volume} \tag{11.1}$$

where $B_m$ is in webers per square meter and $H_c$ is in amperes per meter.




(a) Room temperature (b) Elevated temperature

Fig. 11.13 Effects of temperature rise

Substituting, we obtain

$$W = 4 \times 1{,}500 \times 10^{-4} \frac{1.5}{4\pi \times 10^{-3}} \tag{11.2}$$
$$= 72.7 \text{ joules/m}^3$$

The volume of our core is

$$\text{Volume} = \frac{\pi}{4}(0.080^2 - 0.050^2) \times 0.025 \times 2.54^3 \times 10^{-6} \tag{11.3}$$
$$= 1.254 \times 10^{-9} \text{ m}^3$$

Thus the total energy loss per cycle, due to hysteresis, is $0.0914 \times 10^{-6}$ joules. Under extreme conditions such a core might make a complete traversal of its hysteresis loop once every 4 $\mu$sec. The average energy dissipation would then be 22.8 mw. Although this amount of power represents a minor burden upon any practical driving circuit, it nevertheless can contribute to a marked temperature rise of the core.

If the properties of magnetic materials were invariant with temperature, the energy loss in the cores would be a negligible factor in the design of core storage systems. However, all magnetic materials display a gradual decrease in their magnetic properties with increasing temperature

until a temperature is reached beyond which the magnetic properties vanish completely. This temperature, known as the Curie temperature, varies from material to material and falls in the range from 300°C to 700°C for most "square-loop" ferrites. Although the rate of decrease in magnetic properties becomes greater as the Curie temperature is approached, it is significant even at room temperatures. As the temperature rises, the hysteresis loop shrinks in both directions, as shown in Fig. 11.13. If the drive currents are not reduced to compensate for this shrinking, then the half-selecting currents take the core further around

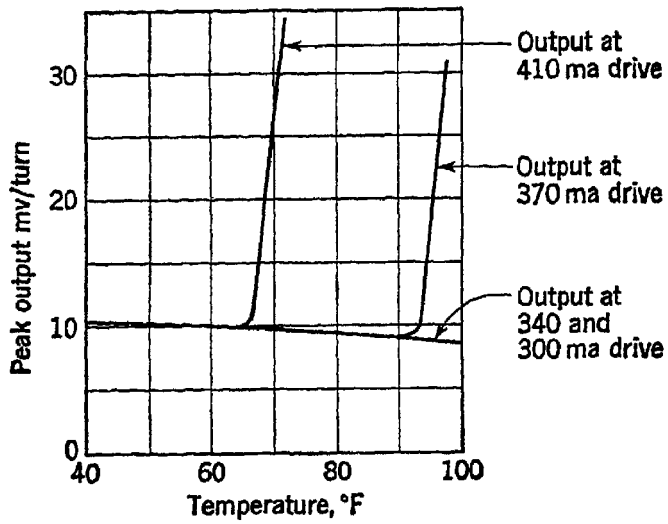



Fig. 11.14 Output voltage for stored 0 as a function of temperature

the knee of the loop, thus producing larger flux changes and larger disturbing voltages, both from half-selection of 1's and from full selection of 0's. The curves of Fig. 11.14 show the magnitude of the peak 0 output for a typical 0.080 in. core designed to switch at a nominal driving current of 400 ma. Under conditions of heavy drive (410 ma), this core develops very rapidly increasing 0-output voltage at temperatures above 65°F. The effect is deferred to higher temperatures when the drive current is lower, as shown by the curves for 370, 340, and 300 ma.

The curves of peak 1-output voltage are somewhat more complex than those of the 0 signal, as can be seen in Fig. 11.15. As would be expected, the ultimate deterioration in magnetic properties at high temperatures does indeed result in a reduction of output voltage. However, the curves for drive currents of 370, 340, and 300 ma exhibit regions in which the output voltage rises as the temperature increases. This effect may be attributed to several causes, among which two predominate. (1) The change in magnetic properties with temperature is by no means linear, and the permeability first rises to a peak and then drops rapidly as the temperature increases. (2) As the hysteresis loop shrinks and the coercivity $H_c$ decreases, the value of $H$ produced by a fixed driving cur-

rent exceeds $H_c$ by larger and larger amounts. It has been well verified both theoretically and experimentally that the time required for flux reversal is inversely proportional to $H - H_c$, the difference between the applied field and the coercivity. Thus, although shrinking of the loop decreases $\phi$, the amount of flux reversed, the phenomenon of overdrive sensitivity at least partially compensates by reducing the switching time so that $d\phi/dt$, the output voltage generated, may first rise and then fall as the temperature increases.

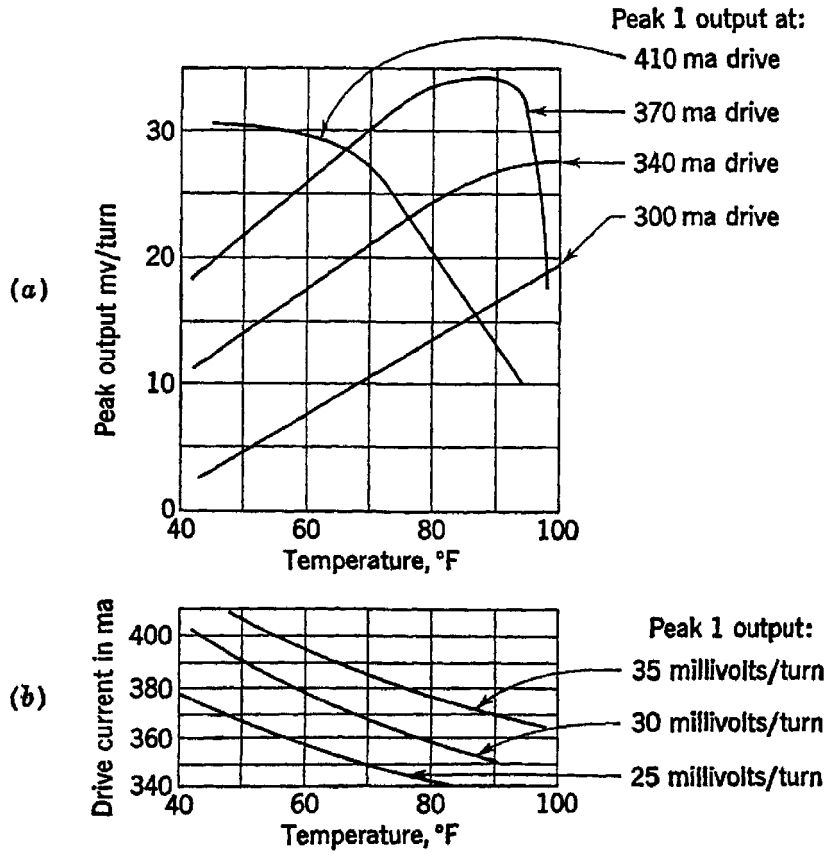



Fig. 11.15 Output voltage for stored 1 as a function of temperature

The effect of increasing drive is shown by the switching-voltage waveforms of Fig. 11.16. The curves numbered 1, 2, 3, and 4 show in that order the output voltages resulting from application of successively larger applied magnetic fields. The areas under these curves are approximately the same, since $\int_0^\infty e\,dt = \phi$, and the flux switched is the same in each case. The axes in Fig. 11.16 are not marked in numbers, since the voltage and the switching time both depend upon the material of the core. Curve 1 might be expected to have a duration of 5 to 10 $\mu$sec for typical ferrite cores.

The relation between switching time $T$ and applied magnetic field $H$ is

$$\frac{1}{T} = k(H - H_c) \tag{11.4}$$

where $H_c$ is the coercivity and $k$ is a constant that depends upon the type of material. For a given material the plot of $1/T$ versus $H$ is a straight line, as shown in Fig. 11.17. The curve has not been extended to the origin, since Eq. (11.4) does not hold for values of $H$ that are close to $H_c$.

The ability to switch cores more rapidly by overdriving them is of little use for the cores in a coincident-current storage array, since there the maximum $X$- or $Y$-driving current must be less than that required to switch the core. However, as we shall see in Sec. 11.9, there are applications of magnetic cores in which it is possible to make good use of the phenomenon of overdrive sensitivity.

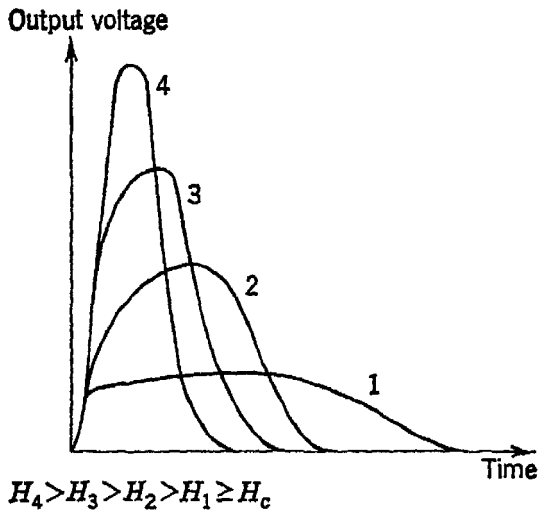



Fig. 11.16 Switching-voltage waveforms for various driving currents

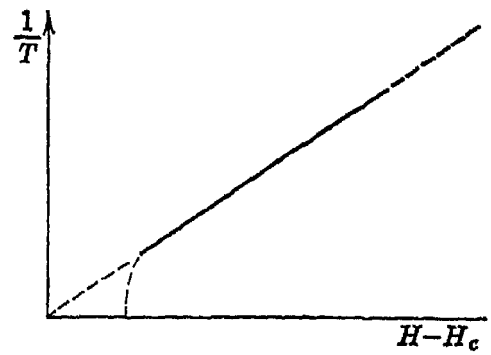



Fig. 11.17 Inverse switching time as a function of excess applied field

It is appropriate to conclude this section with a caution to the reader against trying to read into a hysteresis loop more information than it actually provides. Although the hysteresis loop is a plot of flux density $B$ against magnetic intensity $H$, most published loops are taken at low frequencies, often 60 cycles, and yield very little information about the manner in which $B$ changes when $H$ is changed very rapidly. As we have seen, the flux density does not reverse instantaneously as soon as $H$ exceeds $H_c$. Indeed, if $H$ changes rapidly enough, it can far exceed the coercivity before $B$ has had time to change. Thus, a loop that appears narrow at d-c or low frequencies can appear very broad at high frequencies, as shown in Fig. 11.18. It may even be questioned whether the concept of the hysteresis loop has any meaning at all at high frequencies, since $H$ could conceivably be varied in a rather arbitrary manner once the flux reversal had started and thus produce erratic paths such as the zig-zag path in the figure. It should be noted finally that mere squareness of the corners of a low-frequency hysteresis loop does not indicate that $B$ will change abruptly, since the rate of flux reversal is determined by $H - H_c$, in the manner shown in Eq. (11.4).

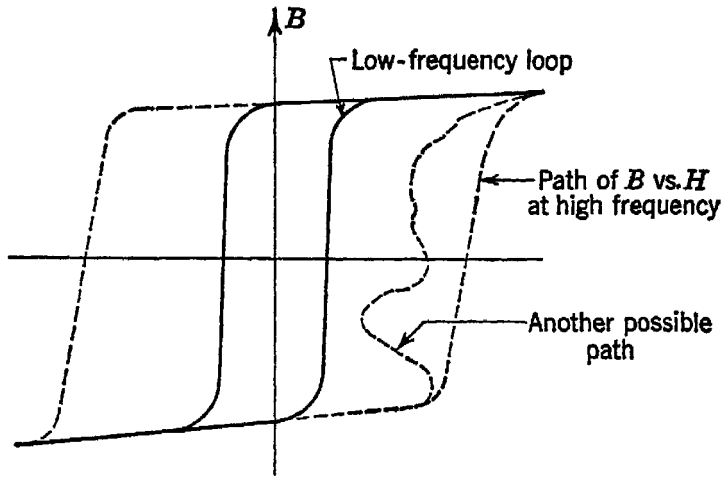



Fig. 11.18 Broadening of the hysteresis loop at high frequencies

## 11.7 Ferrite and metal cores: manufacture and characteristics[17–19]

The manufacturing of ferrite cores is a process requiring care, cleanliness, and precision at each step. In the initial step the various oxides to be used (for example, MgO, $MnO_2$, $Fe_2O_3$) are measured in proper proportions and then mixed by being wet-milled in a rotating vessel containing porcelain balls and enough water to make a thin mud. After several hours of mixing, this wet mixture is oven-dried at about 200°C, and the dry material is powdered. The material is heated in air for an hour at 1050°C, then ground to a very fine powder, commonly in a steel ball mill. This milling step may last for several days. After this the powder is sifted and graded. Two grades of powder may be mixed together so that small particles will occupy the interstices among the larger ones. An organic material is then added to act as a binder so that the powder can be molded into the desired shape. After another mixing step, the material is molded in a die press. Most laboratories use a press that is an adaptation of a pharmaceutical pill-making machine. The "green" cores are then heated in air to about 300°C to burn off the organic binder, and the final firing takes place at a carefully controlled temperature around 1300°C to 1400°C. Although the final firing can be done in an atmosphere of air, it has been found that at some temperatures the material oxidizes and at some it is reduced. The rate of passing through this region of change, around 600°C, affects the ultimate magnetic properties, and better control can be exerted if an inert atmosphere, e.g., argon or nitrogen, is used for this step. The peak temperature is held for about 2 hr, and then the cores come slowly down to room temperature in the oven after it is turned off.

Thin metallic tapes made of magnetic alloys were first developed in

Germany during World War II for use in magnetic amplifiers. The materials currently used for such tapes bear many trade names, but consist of two principal families. One of these is a group having approximately equal proportions of iron and nickel, and the other is a group containing approximately 4 per cent molybdenum, 79 per cent nickel, and 17 per cent iron. These materials are first reduced in thickness to ¼ to ⅛ mil (0.00025 to 0.000125 in.) by cold-rolling. This process has two desirable effects: (1) eddy currents are greatly reduced in cores built up of layers of this material, and (2) the grain structure of the metal is oriented parallel to the direction of rolling, making the material more readily magnetized in this direction. The material, which at this stage may be about 2 in. wide, is then slit into narrow strips perhaps ⅛ to ¼ in. wide. Such a strip is then wrapped around a ceramic bobbin a few sixteenths of an inch in diameter, and the end of the last turn is tacked in place with a small spot weld. Depending on the characteristics desired, the number of turns may be as small as 1 or as large as 40 or more. Since both the wrapping process and the spot weld produce mechanical strains that interfere with the magnetization, the completed assembly is then annealed to relieve these strains. This annealing is done under carefully controlled conditions, sometimes in an atmosphere of dry hydrogen, at a temperature of about 1000°C. The result is a core material having a highly rectangular hysteresis loop, high saturation flux density, and low coercive force, as shown in Fig. 11.10.

Several interesting points of comparison may be made between ferrite cores and magnetic-alloy tape cores. The switching characteristics are of perhaps the greatest interest. Figure 11.19[19] shows curves of $H_m$, the maximum applied field, versus $1/T$, the reciprocal of the switching time in microseconds, plotted for one of the earlier ferrite materials and for two different thicknesses of molybdenum-permalloy tapes. Each of these curves may be expressed by the equation

$$H_m = \frac{k}{T} + H_0 \tag{11.5}$$

where $T$ is the switching time and $H_0$ is the threshold at which irreversible motion of the domain walls commences. $H_0$ is not much less than the coercivity $H_c$, and for our purposes we may regard them as the same. Thus Eq. (11.5) is simply a restatement of Eq. (11.4). It is of interest to note that the switching coefficient $k$, which is the slope of the lines, is not much different for the ferrite shown and for the metal cores. In fact, it has been found that, for most ferrites, $k$ lies within a factor of 2 of the value shown in Fig. 11.19.

It may be observed from Fig. 11.19 that, for the same value of applied field, the metal tape core switches faster than the ferrite core. However,

if the applied field is limited for each material to twice $H_c$ ($\cong H_0$), then the ferrite switches more rapidly. This can be seen by rearranging Eq. (11.5) as follows:

$$T = \frac{k}{H_m - H_0} \tag{11.6}$$

Now, if $k$ is approximately the same for the two materials and if $H_m = 2H_c \cong 2H_0$, we have

$$T = \frac{k}{H_c} \tag{11.7}$$

Thus, under these conditions, the material with the higher coercivity

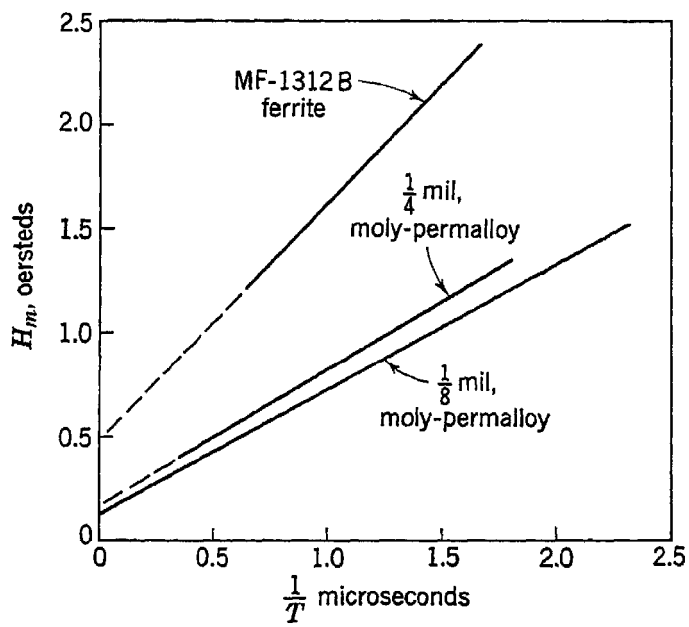



Fig. 11.19 Comparative switching characteristics (from D. R. Brown, D. A. Buck, and N. Menyuk, A comparison of metals and ferrites for high-speed pulse operation, *Trans. AIEE*, vol. 73, pt. I, pp. 631–634, by permission of *AIEE*).

switches more rapidly. Since this is the condition under which the coincident-current storage system operates, much effort has been directed to developing ferrites having higher coercivities. The slower switching of the metal-tape cores in this mode of operation is one reason for their unsuitability in memory arrays. The other principal reason is their high cost, which is due partly to the need for individual fabrication and partly to the nonuniformity of the cores, with the resulting necessity for selection to obtain sufficiently uniform batches.

Energy losses per unit volume are approximately the same for ferrites and for metals. Brown, Buck, and Menyuk[19] report a total loss, due to all sources, of approximately 1,170 ergs/cm$^3$, or 117 joules/m$^3$, for metals and approximately 800 ergs/cm$^3$, or 80 joules/m$^3$, for ferrites.

Because the flux density is larger in the metals than in the ferrites, metal cores can be smaller than ferrite cores for the same flux change. Thus metal cores show lower power dissipation per core than do ferrite cores. Metal cores are easily cooled, because of their high thermal conductivity. Moreover, the Curie point of the metals is usually at a higher temperature than that of the ferrites, and the metals therefore exhibit less temperature sensitivity.

## 11.8 Multidimensional selection

The principle of selection of one core from among many by coincidence of the currents in two driving windings may already have suggested to the reader the possibility of using coincidence among three or four or many wires for selection in cores with sufficiently distinct thresholds. This idea has often been suggested and is worth examining in some detail. In order to provide the maximum possible difference in excitation between the selected core and the unselected cores, let us arrange that each of the $M$ selecting wires linking each core will normally carry a bias current of $-k$ units, where $k$ is a fraction. Let the total driving current for full selection be 1 unit of current. To select a particular core, let the bias current in one wire of each of the $M$ coordinates be replaced by a selecting current of $1/M$ units. The one core that is fully selected thus receives 1 unit of current. A core with $M - 1$ excitations receives a total drive of $(M - 1)/M - k$ units of current. If the total excitation of such a core, which is the most highly excited nonselected core, is made the same in magnitude as the net bias in a core having no excitations, then

$$Mk = \frac{M-1}{M} - k \tag{11.8}$$

$$k(M+1) = \frac{M-1}{M} \tag{11.9}$$

$$k = \frac{M-1}{M+1}\frac{1}{M} \tag{11.10}$$

The ratio $R$ of the current in the most highly excited nonselected core to the current in the selected core is

$$R = \frac{M-1}{M} - k \tag{11.11}$$

$$= \frac{M-1}{M} - \frac{M-1}{M+1}\frac{1}{M} \tag{11.12}$$

$$= \frac{M-1}{M+1} \tag{11.13}$$

The data in Table 11.2 show the effect of increasing $M$.

Such calculations as these provide an interesting mathematical pastime but little else of real value. Cores with thresholds sufficiently sharp to allow reliable use of values of $M > 2$ do not exist. Even if they did, however, the problems of wiring multidimensional arrays become enormous even for small values of $M$. The one thing that a magnetic-core element does better than it does anything else is to "remember". To require of a core not only that it remember but also that it discriminate between $M$ and $M - 1$ excitations is to impose difficult requirements both upon the

**Table 11.2**
**Bias current and selection ratio for multidimensional selection**

| $M$ | $k$ | Total bias | $(M-1)/(M+1)$ |
|---|---|---|---|
| 2 | $1/6$ | $-1/3$ | $1/3$ |
| 3 | $1/6$ | $-1/2$ | $1/2$ |
| 4 | $3/20$ | $-3/5$ | $3/5$ |
| 5 | $2/15$ | $-2/3$ | $2/3$ |

core and upon the circuit designer. Thus it is common practice to use the cores in no greater than a 2:1-selection mode and to perform the rest of the selection, or all of it, in other circuits. In the next section we shall examine some core matrix switches used in external selection.

## 11.9 Matrix switches[4,20,21]

As we have already seen, the use of the two-dimensional coincident-current principle considerably simplifies the selection problem in large arrays. For instance, an $m \times n$ rectangular array would require a total of $mn$ drive wires if all selection were performed externally, but, through the use of two-dimensional coincident-current drive, the number of drive wires becomes $m + n$, a substantially smaller number. A matrix $64 \times 64$, a common size, would require 4,096 wires for external selection, but needs only 128 when the cores themselves participate. Of course, the problem of making two selections of one drive line out of 64 is by no means trivial. One typical solution is to use the coincident-current principle in an $8 \times 8$ array to select one out of 64. Thus the selection problem becomes four selections, each of one wire among 8, as shown in Fig. 11.20. Each of the 64 cores in an $8 \times 8$ array drives one of the 64 lines in one coordinate of the large array.

Although the driving matrix that excites the main storage matrix employs the coincident-current principle, its design problems are in many ways simpler than those of a storage matrix. There is no sense winding threading all cores, since each core has its own separate output winding. Likewise, there is no inhibiting winding. Furthermore, each core in the

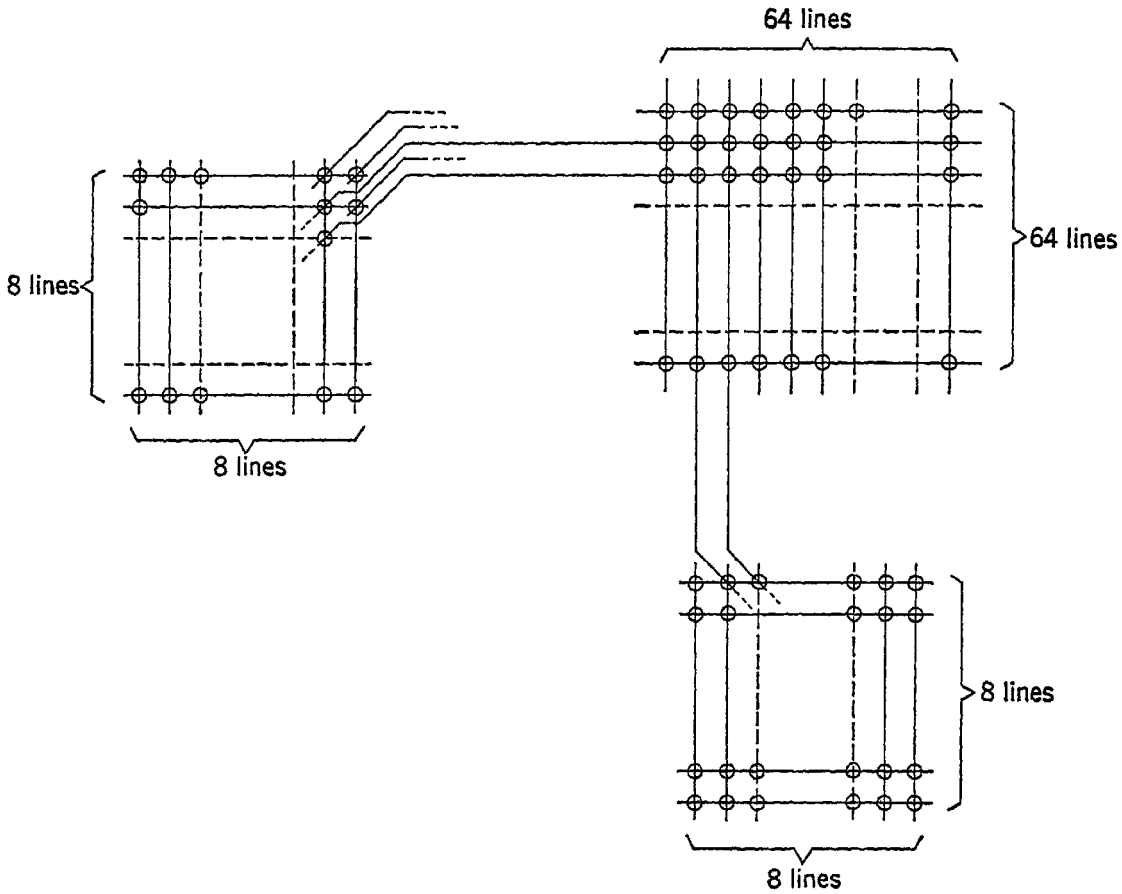



Fig. 11.20 Matrix drive applied to storage matrix

driving array executes a complete traversal of its hysteresis loop each time it is selected, and all cores normally rest in the same state. For this reason it is practical to apply a bias current to all cores in order to overdrive them upon selection. As can be seen from Fig. 11.21, if a bias of $I_b$ units is applied in the reverse direction and if $I_m$ is the forward current required for full selection, then each selecting wire can carry $I_b + I_m/2$ amp. The one core selected is driven by $I_m + I_b$ amp, and no other core receives enough excitation to switch. Of course, some overdrive is necessary in order for the output winding of the selected core to be able to furnish current to its load, the line of cores in the memory matrix. A simplified picture of the circuit action is obtained by regarding the core as a transformer requiring a magnetizing current of $I_m$ amp. Any load ampere-turns must be provided by an equal number of primary

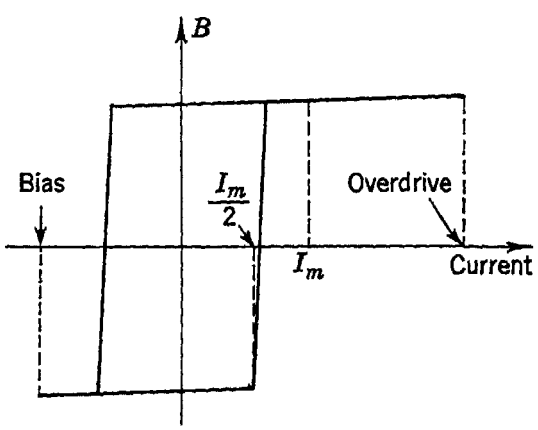



Fig. 11.21 Use of bias current to obtain overdrive

ampere-turns. Thus, if each selection winding has $n_1$ turns and if the output winding has $n_2$ turns and if a secondary current of $I_s$ amp flows, then the primary current $I_p$ in each winding must be at least

$$I_p = \frac{I_m}{2} + \frac{1}{2}\frac{n_1}{n_2} I_s \tag{11.14}$$

Any primary current in excess of this value constitutes overdrive, which is effective in switching the core more rapidly.

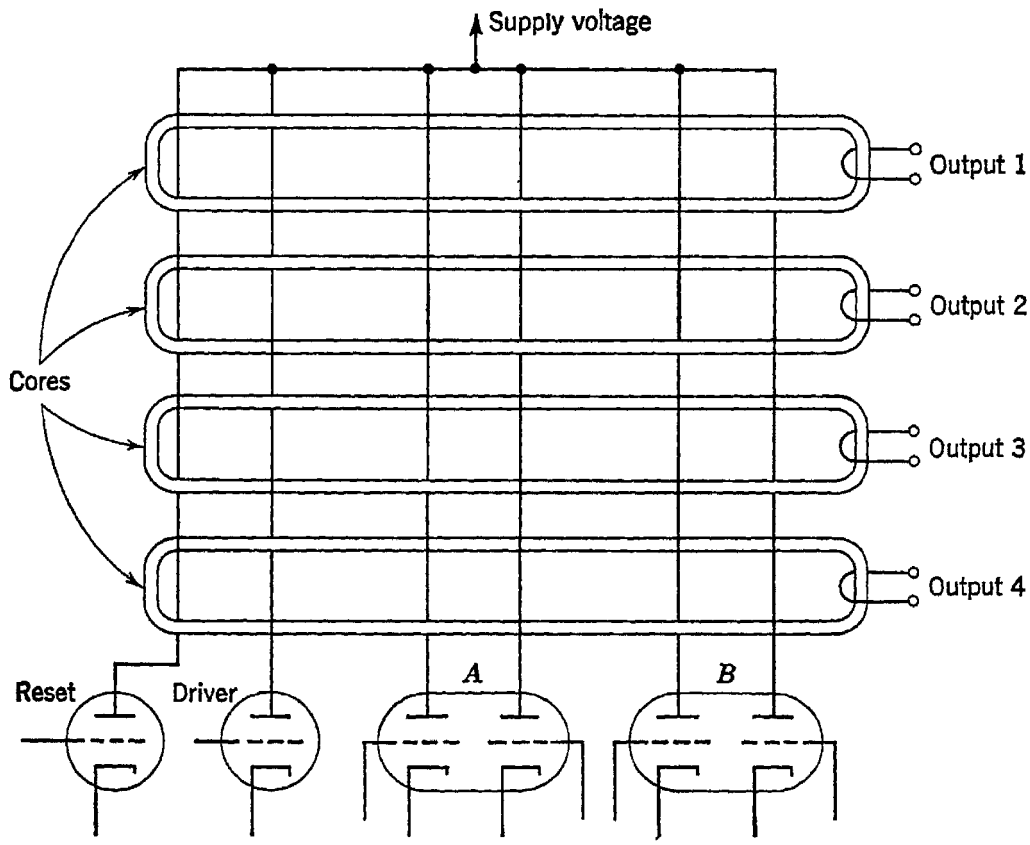



Fig. 11.22 Selection using multiwinding cores

The secondary current $I_s$ is the current in a row or column of the memory matrix. The cores on this line present a highly variable load which depends upon the pattern of stored 0's and 1's and upon the previous history of half-drives of each core. In order to stabilize this load, each line is terminated in a resistance $R_0$, selected so that its time constant $L/R_0$, as calculated from the effective inductance of the line of cores, will be substantially less than the period of the driving cycle. Without this resistance the current in the selected line of the memory array would rise and fall slowly, and the duration of the driving period would have to be greatly extended. The power delivered to these terminating resistances may be 10 to 100 times as much as that dissipated in the memory cores themselves.

A square array of cores operating in the two-dimensional coincident-

current mode is not the only way of using cores to perform selections. Another way is to employ multiple coincidence, using cores with multiple windings that are connected to the binary-signal sources in such a way that each different set of binary input data energizes a different core. Such an arrangement is shown in Fig. 11.22. The double triodes may be regarded as flip-flops storing binary information. The core windings are connected in such a way that, for each of the four combinations of states of flip-flops $A$ and $B$, a different one of the four cores receives no excitation and all others receive two units of excitation. The driver tube excites all cores but in the direction opposite to the mmf (magnetomotive force) applied by $A$ and $B$. Thus $A$ and $B$ act to inhibit all cores but one, and the driver switches that one core and produces a voltage on its output winding. The core thus selected must be reset before it is again selected, and, to accomplish this, the RESET tube is gated ON after the driving tube has been turned off.

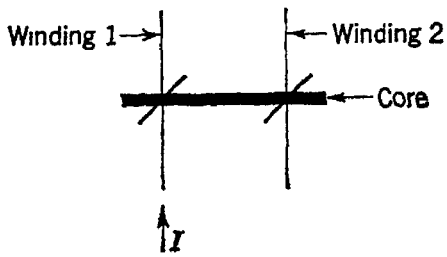


Fig. 11.23 The mirror symbol for multiwinding cores

Diagrams like that of Fig. 11.22 become quite complicated for larger arrays, and a convenient symbolism devised by R. P. Mayer is often used instead. Each core is represented simply by a heavy straight line and the windings by lines crossing it at right angles. A short diagonal stroke at the crossing indicates that the winding links the core once. The direction of the stroke indicates the sense of the winding. Figure 11.23 shows the symbol for a core with two windings. To find the direction of flux in the core due to current in a winding, one regards the diagonal stroke as a mirror that "reflects" a current in one direction or the other along the winding line into a flux directed along the core. Thus the current $I$ flowing upward in winding 1 of Fig. 11.23 produces a flux to the right in the core. To find the polarity of the voltage induced in winding 2 by a current increase in winding 1, one imagines an oppositely directed flux reflected from the mirror into a voltage directed along the line representing the winding. An increase of current $I$ flowing up in winding 1 thus makes the lower end of winding 2 positive with respect to the upper end. In Fig. 11.24 the circuit of Fig. 11.22 is redrawn using this notation.

Many variations of the multiple-coincidence circuit are possible. Figure 11.25 shows a circuit in which each tube of each flip-flop is coupled to all cores. The double diagonal strokes indicate windings having twice as many turns as those represented by single strokes. It can be seen that each flip-flop contributes both to the drive of the selected core and to the inhibition of the unselected cores. The reader will find it helpful to

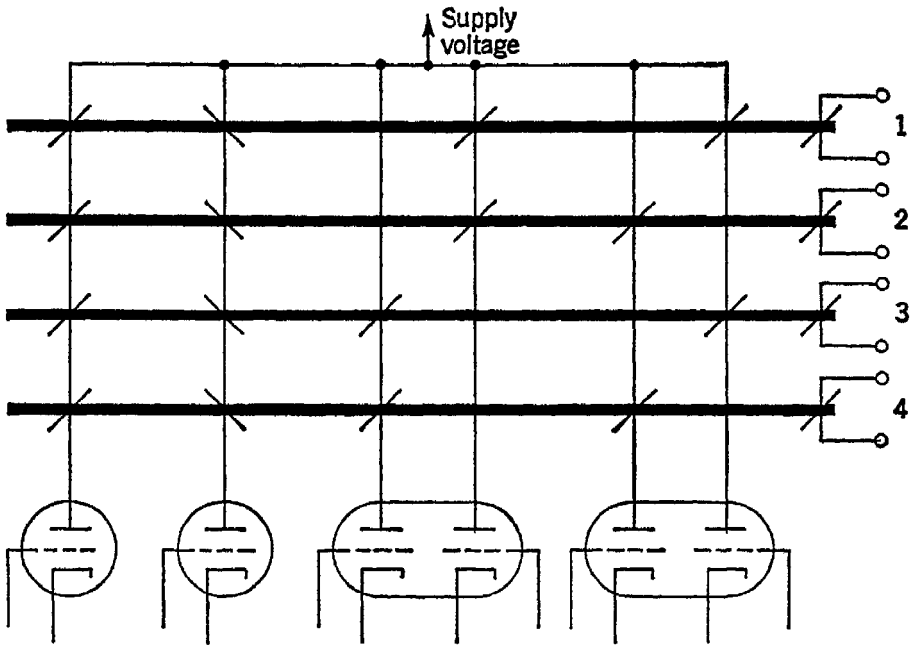



Fig. 11.24 Circuit of Fig. 11.22 redrawn with mirror symbols

tabulate the total excitation applied to each of the four cores for each of the four combinations of flip-flop states. It will be found that the selected core receives 2 units of driving excitation, two cores each receive 1 unit of inhibiting excitation, and one core receives 4 units of inhibiting excitation in this four-output circuit. If this type of circuit is generalized to have $n$ flip-flops, it will be observed that, for each combination of flip-flop states, one core will receive $n$ units of driving excitation, and the others will receive $n - j$ units of driving excitation and $2j$ units of inhibiting excitation, where $j$ ranges over the values 1 to $n$. An advantage of this circuit is that a separate driver tube is not needed, and the total required drive is divided equally among the flip-flops. A separate RESET circuit is also unnecessary. However, the absence of a separate driver somewhat complicates the timing by making it necessary for all flip-flops to be set simultaneously to their desired states when the selection is to be made.

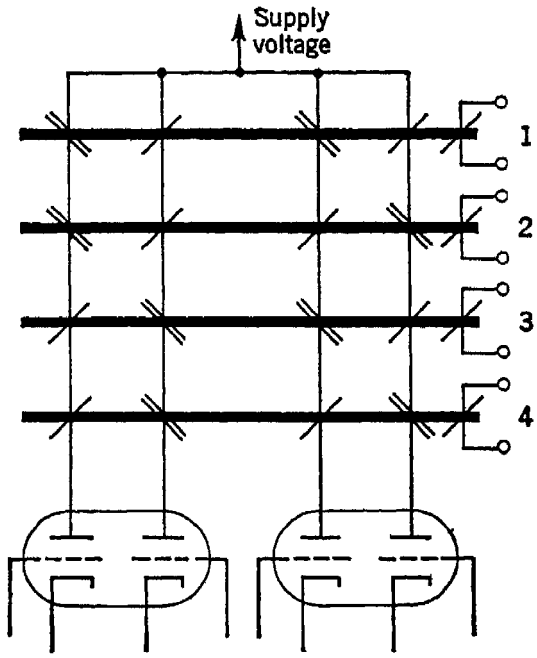



Fig. 11.25 Drive-sharing arrangement

Another arrangement, shown in Fig. 11.26, returns the common plate-supply lead through a winding linking all cores. The differences in

excitation among the various cores are the same as in Fig. 11.25, but the amounts of the excitations are different, the selected core being more heavily driven in the circuit of Fig. 11.26.

Another variation employs the basic circuit of Fig. 11.25, but, instead of always employing a 2:1 turns ratio between inhibiting and driving windings, it uses a ratio determined by the number of input flip-flops. If there are $n$ flip-flops and $2^n$ outputs, each inhibiting winding has $n - 1$ times the number of turns in the driving windings. Thus, for each combination of flip-flop states, one core receives $n$ units of driving excitation,

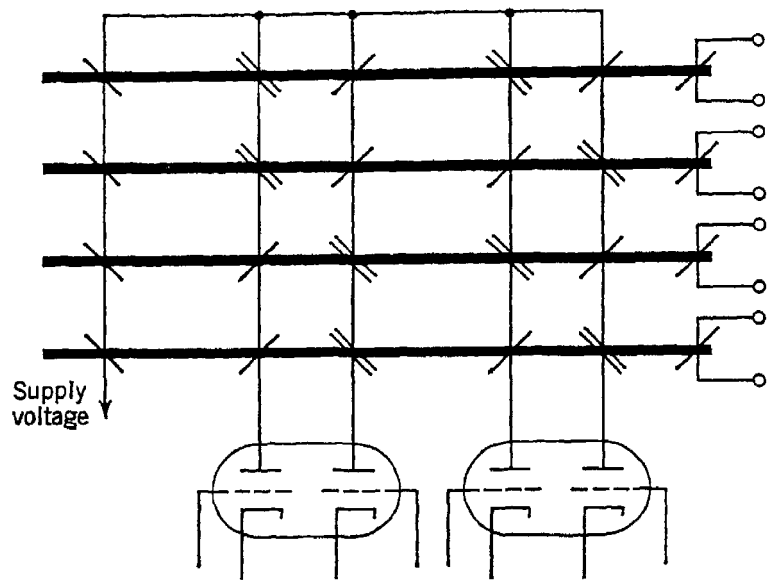



Fig. 11.26 Supply current returned through a driving winding

and the others receive $n - j$ units of driving excitation and $(n - 1)j$ units of inhibiting excitation, where $j$ ranges over the values 1 to $n$. From this it follows that one core receives $+n$ units of excitation, some none, some $-n$, and so forth, down to one that receives $-n(n - 1)$.

## 11.10 Completely external selection

In the previous section we examined several ways of performing selection by circuitry external to the memory cores. It is reasonable to extend these methods in such a way as to allow the entire selection operation to be performed outside the memory cores. If this extension is made, the cores no longer operate in a 2:1 coincident-current mode and are freed from the limitations imposed by such operation. With completely external selection, the cores serve only their principal function of storing.

Several advantages of external selection may be noted. Since the memory cores need no longer discriminate between half-excitation and full excitation and instead receive no excitation whatever unless they are selected, it is practical to use large excitation currents that overdrive the

selected cores on read-out and switch them more rapidly. Higher switching speed allows the cycle time to be less and provides a larger output voltage, since the rate of change of flux is greater. The sense winding can link the cores in a much simpler manner than in the coincident-current matrices, since there are no half-selected cores and, therefore, no problem of canceling out disturbing signals. Furthermore, the cores need not be so rectangular or so uniform as they must be for the coincident-current

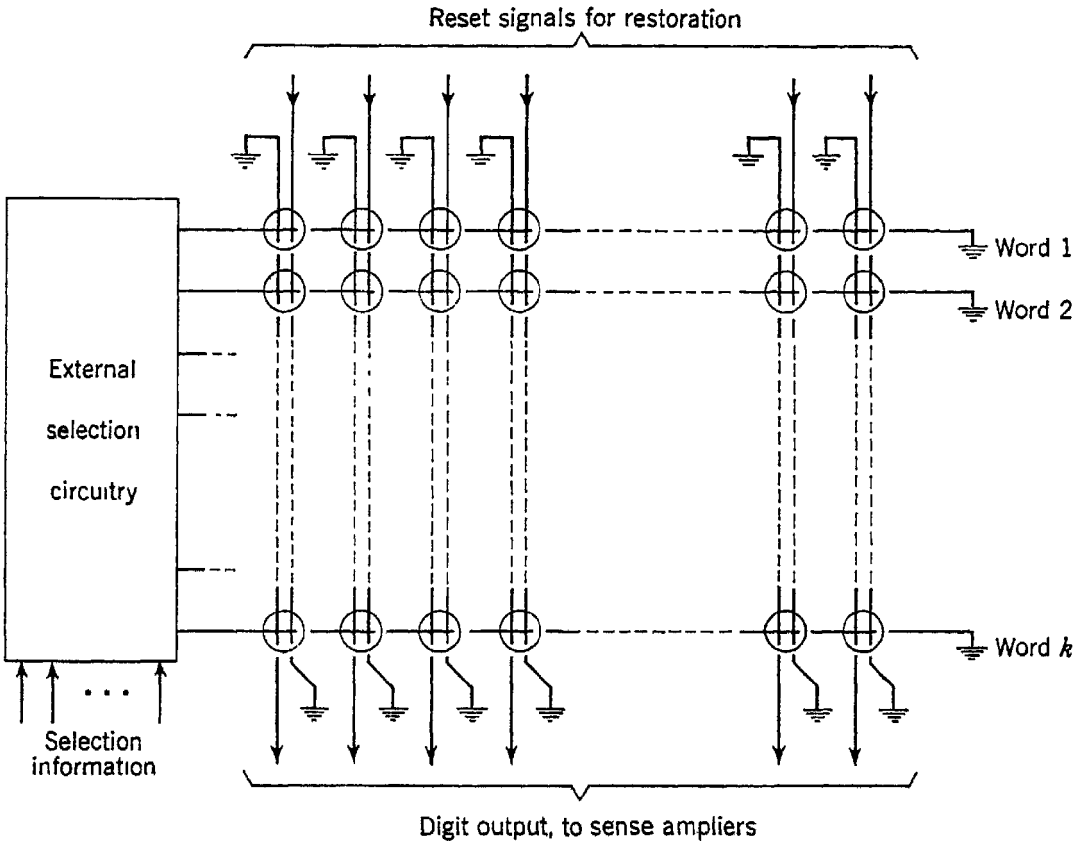



Fig. 11.27 Elementary word-organized memory plane

system. The major disadvantage is the greatly increased complexity of the external-selection circuitry.

As might be expected, some new problems not encountered in the coincident-current system are met in the external-selection method. The core planes are wired differently, in a pattern sometimes called a "word-organized" arrangement. All the digits of a word are stored in a row of cores on one plane. A single drive wire through the row excites all the cores of one word when it is driven by the external selecting circuitry. Each column of cores in the same digit position is linked by a separate sense winding, each winding in turn driving its own sense amplifier, just as in the coincident-current array. Each core must also be linked by a separate inhibiting or RESET winding to provide for restoration of information after read-out. Figure 11.27 shows such a configuration.

The major problem encountered in such an array is that the load on each drive line is highly variable and depends upon the pattern of information stored. If all the cores are switched, the external-selection circuitry must drive the required current through the resulting high back emf caused by the flux reversal in all cores. Similarly, if only part of the cores are switched or none at all, the selection circuitry is required to produce essentially the same current against a much lower back emf. Although it is possible to convert the driving circuit to a current source by putting a high resistance in series with each driving line, as in the

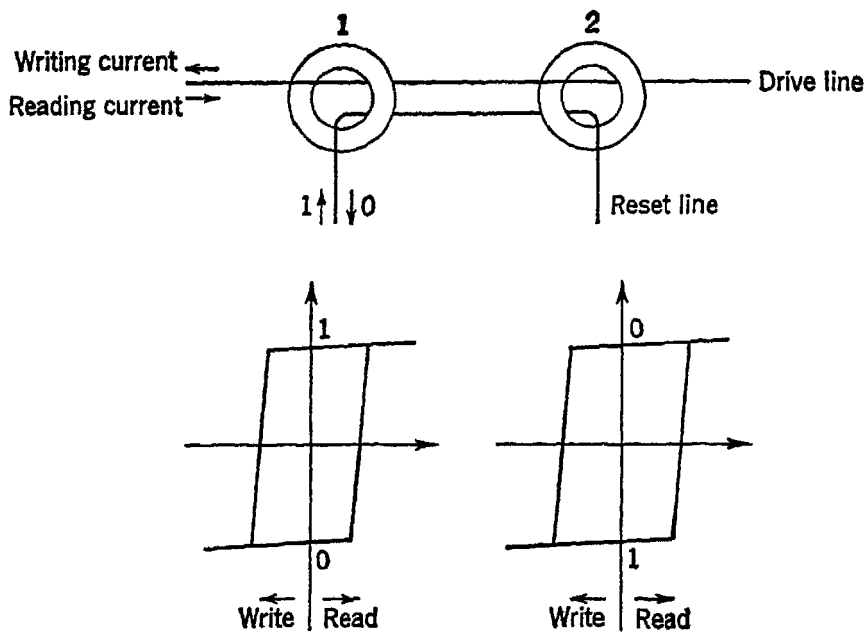



Fig. 11.28 A two-core-per-bit cell

coincident-current system, the required resistance would be very large, because of the wide variation in the total back emf. A better method is to stabilize the load by using two cores for each stored bit. The two cores are always magnetized oppositely, and one or the other of them is always switched by each reading or writing signal. The two-core configuration for storing one bit is shown in Fig. 11.28. Assume the cores both initially store 0; i.e., core 1 has negative remanent flux and core 2 positive. A reading pulse switches core 1 but simply drives core 2 further into saturation. Both cores are left at the top of their loops at the end of the reading pulse. To restore the cell to its initial state, a writing pulse is applied in the opposite direction on the drive line, and the RESET line receives an inhibiting pulse in such a direction as to prevent the switching of core 2 and to aid the switching of core 1. If the digit 1 is to be stored after the read-out, then the RESET line is driven in the opposite direction. Since the RESET line links cores in nonselected words as well as in the selected word, the RESET current must be smaller than the value required to switch the flux in order to prevent accidental switching of cores in non-

selected words. Therefore, the magnitude of the writing current is also restricted.

The two-core-per-bit memory cell was first proposed in 1952 by K. H. Olsen at the Lincoln Laboratories of MIT. However, since there have been available in the United States cores of high quality for use in coincident-current systems, the word-arrangement memory has not been much used. Interest in it has recently revived, however, as demands for higher speeds of computation have made its shorter cycle times appear more attractive. It is currently being used (as of 1960) in several digital systems and in at least one general-purpose computer. It is interesting to note that a word-arrangement memory system is employed in the BESM computer at the Institute of Exact Mechanics and Computing Technique of the Soviet Academy of Sciences. The development of the two-core-per-bit cell, apparently independently, is attributed by the Russians to Professor L. I. Gutenmacher. This memory was incorporated in the BESM computer in 1956.

## 11.11 Speeding up core memories

The never-ending quest for higher speeds of operation of digital computers puts heavy demands upon the designer of storage systems, since in most computers it is the memory system rather than the control circuits or the arithmetic circuits that sets the maximum speed of operation. Efforts are continually being made to make storage systems operate more rapidly. In this section we shall examine briefly some of the techniques that have been used.

**Effect of core dimensions.** The external magnetic field $H$ about a long straight conductor carrying a current $I$ is found from the relation

$$I = 2\pi r H \tag{11.15}$$

where $r$ is the distance from the center of the wire. If such a wire passes through a core, as shown in Fig. 11.29, the field will be higher at the inside radius $r_i$ than at the outside radius $r_o$. Thus, as the driving current is applied and rises from its initial value to its final value, the threshold value of $H$ required for switching is reached at the inside of the core before it is reached at the outside, and the switching of the outside parts of the core is therefore initiated later. If the current pulse rose perfectly vertically, this time lag would not exist, but, for any finite rate of rise, the lag does exist. One obvious way to reduce this lag is to make the core walls thinner by reducing the outside radius of the core. An inci-

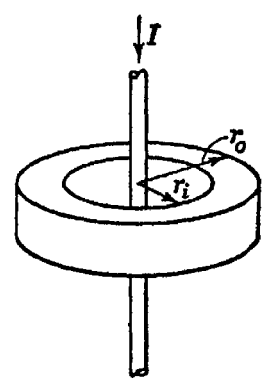



Fig. 11.29 Current-carrying conductor passing through a core

dental advantage of this step is that it makes the hysteresis loop of the core more nearly rectangular, since a smaller range of current change is now required to switch the core from one direction of magnetization to the other, as shown in Fig. 11.30.

Over-all reduction of core dimensions has another desirable effect. Since the ratio of core surface area to core volume is increased as the dimensions are reduced, the energy dissipated within the core is more readily removed from it. Thus a small core can be driven more often than a large one for the same rise in temperature. Moreover, a small core does not require so great a driving current as a large core, since the threshold field $H$ is reached at a smaller current in cores of smaller radius. Very small cores are extremely difficult to assemble in arrays, however, and, if their walls are too thin, they are very fragile. In addition, the reduction of cross section reduces the total flux, and smaller output voltages are obtained unless the switching speed is somehow simultaneously increased. Although very small cores have been made experimentally, the smallest cores used in any large quantities are 0.050 in. OD and 0.030 in. ID.

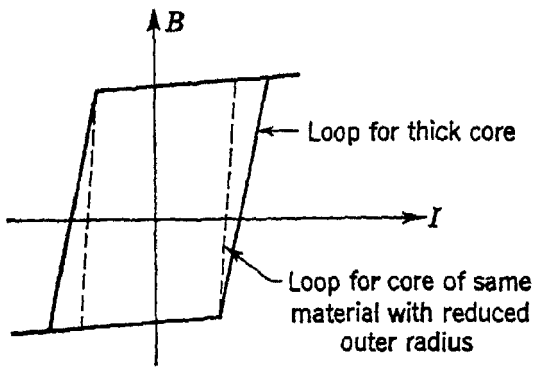



Fig. 11.30 Effect of reducing core-wall thickness

**Partial switching.** Several observers have reported that it is possible to apply short-duration high-intensity driving pulses of current to a core without totally reversing its magnetization. Newhouse[22] has used pulses of 100 m$\mu$sec duration that exceed the coercivity by several times and has observed that, although a disturbance is produced, the final flux is the same as the original. McMahon[23] has used somewhat longer pulses, having carefully controlled magnitudes, to reverse the flux in the inside part of the core while leaving the outer parts unaffected. This partial switching takes much less time than flux reversal throughout the entire core. This method has approximately the same electrical effect as reducing the wall thickness, without loss of core mechanical strength.

**Thin films.** In the search for magnetic materials that can switch rapidly, several laboratories have studied the use of very thin metallic films of magnetic material. Such films are laid down by vacuum deposition of metallic vapors. Thicknesses from several hundred angstroms (A) to about 10,000 A have been studied. Very thin films appear to display no magnetic properties at all, but films of about 2,000 A thickness have been found to have characteristics much superior to ½-mil tape-

wound cores of the same material. Pohm and Rubens[24] have described an experimental storage array using thin-film magnetic elements that are capable of switching in as little as 0.1 $\mu$sec.

**Nondestructive sensing.** A prominent disadvantage of all the core storage systems that we have so far considered is the necessity for rewriting information destroyed by reading. If it were possible to read information from a memory register without removing the information from the memory, operation of the computer could be more rapid, since in most programs there are many accesses to memory performed simply for reading and not for writing. This problem of nondestructive read-out has proved to be very thorny, and the simplest methods that have been developed are not nearly so simple as one might wish.

One class of nondestructive read-out methods uses a magnetic field at right angles to the direction of the flux that stores the information, as shown in Fig. 11.31.[25] The effect of the quadrature field is simply to rotate the direction of magnetization without causing appreciable change in the length of the flux vector. The component of flux along the principal axis of the core is thus reduced in magnitude. The net change in this principal component of flux is opposite in sign for a stored 0 and for a stored 1. Thus an output winding coupled to this principal field will produce either a small positive voltage or a small negative voltage, depending upon the stored information. Upon removal of the quadrature field, the principal flux returns to its original direction without change of magnitude.

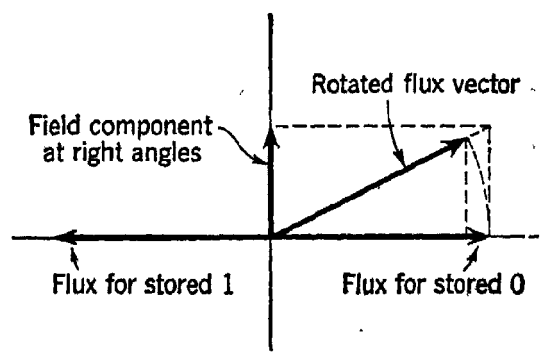


Fig. 11.31 Effect of quadrature field

The difficult problem in this method is establishing the quadrature field. The method works when the core is mounted in the gap of an electromagnet, but this is hardly practical for large memories. Metal tape cores have been used with a current flowing around the turns of the tape to produce the quadrature field, but this method is also impractical except in the laboratory.

Another method of nondestructive read-out is the application of short-duration, high-intensity current pulses, as described by Newhouse.[22] Such pulses produce no permanent change of magnetization and leave the core at its original remanent-flux value. The transient flux change is found to be greater for cores in which the remanent flux is negative than for those in which it is positive. Thus the voltage produced by the sensing pulse is greater for a stored 1 than for a stored 0.

## 11.12 Magnetic-core shift registers[26–29]

Most of the discussion in the preceding sections has been directed toward the application of magnetic cores as storing elements in random-access memories. Although this has been a major application of cores, it is by no means the only application, and in this section and the next we shall examine some others.

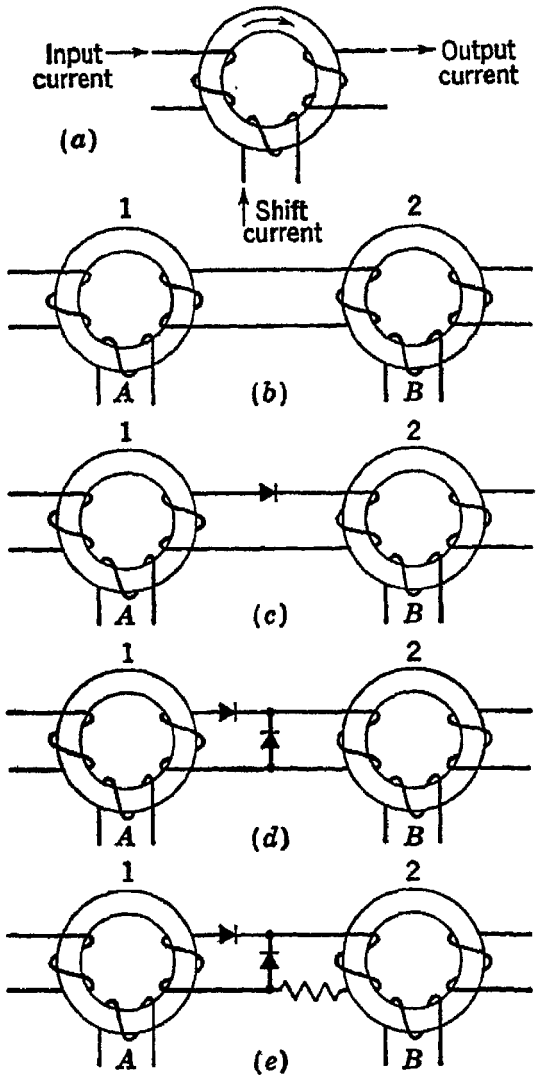



Fig. 11.32 The magnetic-core shift register

Cores can be very easily interconnected to form a shift register, and their use in this manner preceded their application in coincident-current storage systems by about two years. The behavior of the core shift register may be readily understood by reference to the step-by-step development of Fig. 11.32. The basic element, a three-winding core, is shown in 11.32*a*. An input current pulse magnetizes the core in the direction shown. At a later time a pulse of shift current reverses the flux in the core. This flux reversal induces a voltage that makes the upper output terminal momentarily positive, and an output current would flow in the indicated direction if this winding were connected to another, as shown in Fig. 11.32*b*. The element in *b* is not satisfactory, however, since the flux reversal due to an input pulse also induces a voltage in the output winding. This voltage is opposite in polarity to that which occurs when a shift pulse is applied. To prevent this pulse from being coupled to core 2 and thence to all succeeding cores in the register, a diode is inserted in the coupling path, as shown in *c*.

One more spurious signal remains to be blocked. This is the voltage produced at the input winding when the shift pulse is applied. The flux reversal produced by a shift pulse makes the lower terminal of the input winding go positive. The series diode of Fig. 11.32*c* allows this current to pass; so some other means must be provided to prevent information

from flowing back to the left. The shunting diode $D_2$ of $d$ serves this purpose, since it provides a low-impedance path shunting the output winding of core 1. The voltage produced at the input winding of core 2 by the shift pulse applied to $B$ thus causes a current to circulate through $D_1$, $D_2$, and the input winding. This current opposes the switching of core 2, and, in order to reduce it, the resistance shown in $e$ is inserted.

The circuit shown in Fig. 11.32$e$ represents a single stage of a magnetic-core shift register, although it uses two cores. Two cores per stage are needed in this shift register because there is no time delay between cores. In this register all the $A$ windings of the No. 1 cores in the successive stages are connected in series, and all the $B$ windings of the No. 2 cores are connected in series. The $A$ and $B$ shift lines are pulsed alternately, and input signals to the first core in the chain are applied in synchronism with $B$ SHIFT pulses. Simultaneous application of SHIFT pulses to core 1 and to core 2 is not possible, since each core would then receive a SET pulse on its input winding at the same time that it received a RESET pulse on its SHIFT winding.

The second core in each stage may be regarded simply as a temporary-storage element to hold the information while the first core is being reset in preparation for accepting new information. Other means of temporary storage can be used. For example, Fig. 11.33 shows a one-core-per-bit shift register that uses interstage capacitors as temporary-storage devices. The capacitor charges rapidly through the diode whenever a shift pulse is applied and then discharges somewhat later through the input winding of the next core to the right. The shunt diode is not needed, since the capacitor provides a low-impedance path for currents that try to feed information back to the left.

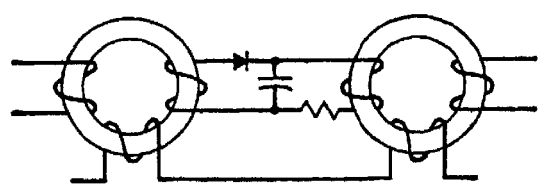


Fig. 11.33 A one-core-per-bit shift register

## 11.13 Magnetic-core logic[30,31]

Besides being a very useful memory element, the magnetic core also finds applications in performing logic. We have already seen one method of using a core as an AND element in the coincident-current selection system used in storage matrices. The OR function can be implemented by placing several windings on a core, any one of which can SET the core when it is driven by a suitable current pulse. Negation, or at least inhibition, can be performed by means of an oppositely excited winding having a few more turns than a SET winding. Thus it is seen to be possible in principle, at least, to construct entire logic networks from cores.

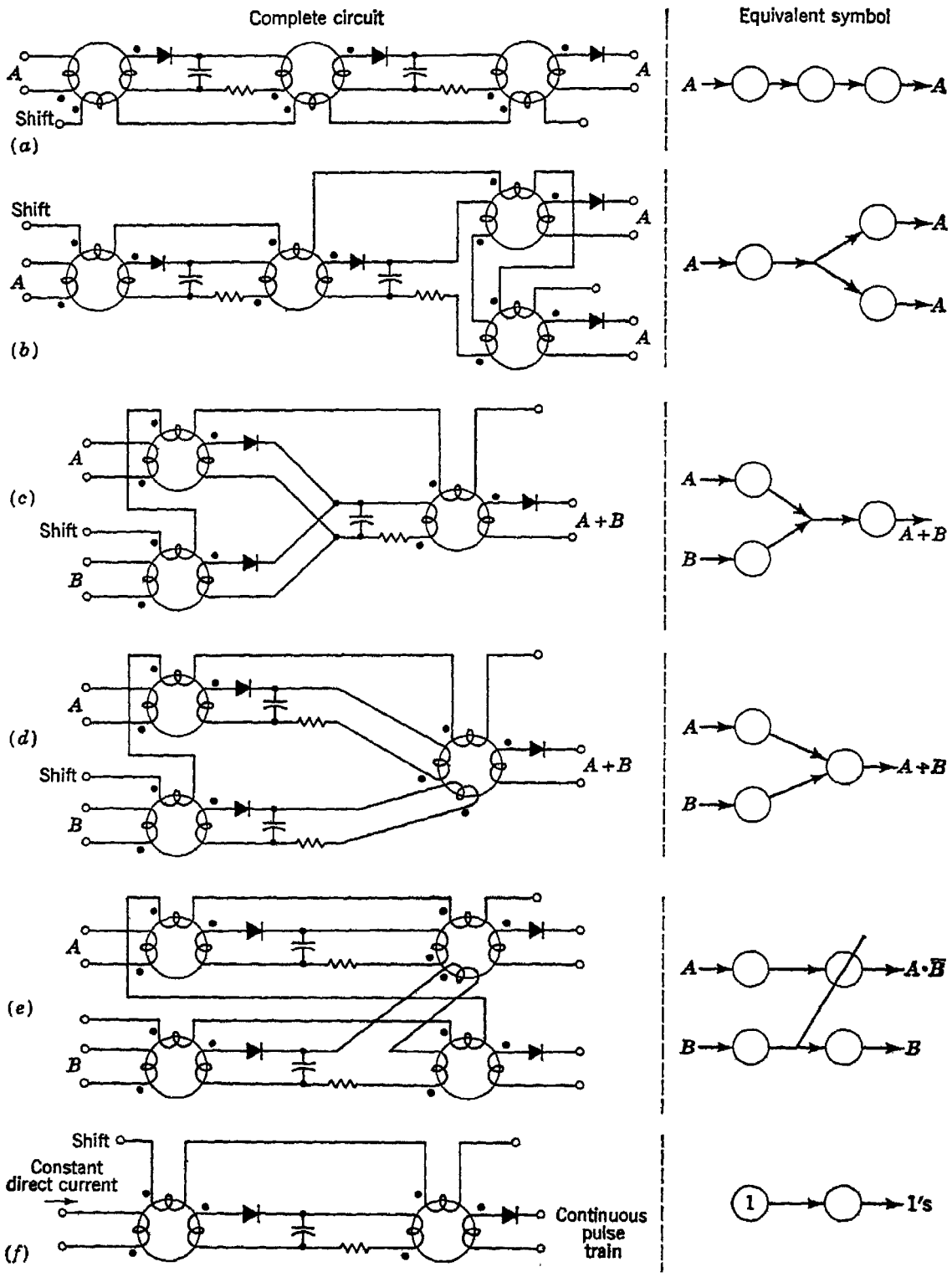



Fig. 11.34 Symbols for logic devices based on the core shift register: (*a*) shift register, (*b*) branch circuit, (*c*) an OR circuit, (*d*) another OR circuit, (*e*) inhibiting circuit, (*f*) 1's generator

In practice, however, this simple picture requires some elaboration. Since magnetic cores are not capable of producing continuous output signals but instead produce voltages only when flux is changing, they must be operated synchronously, with clock signals distributed to all parts of the system. Thus the output signal from a core connected to implement some particular logical function is stored in another core and is

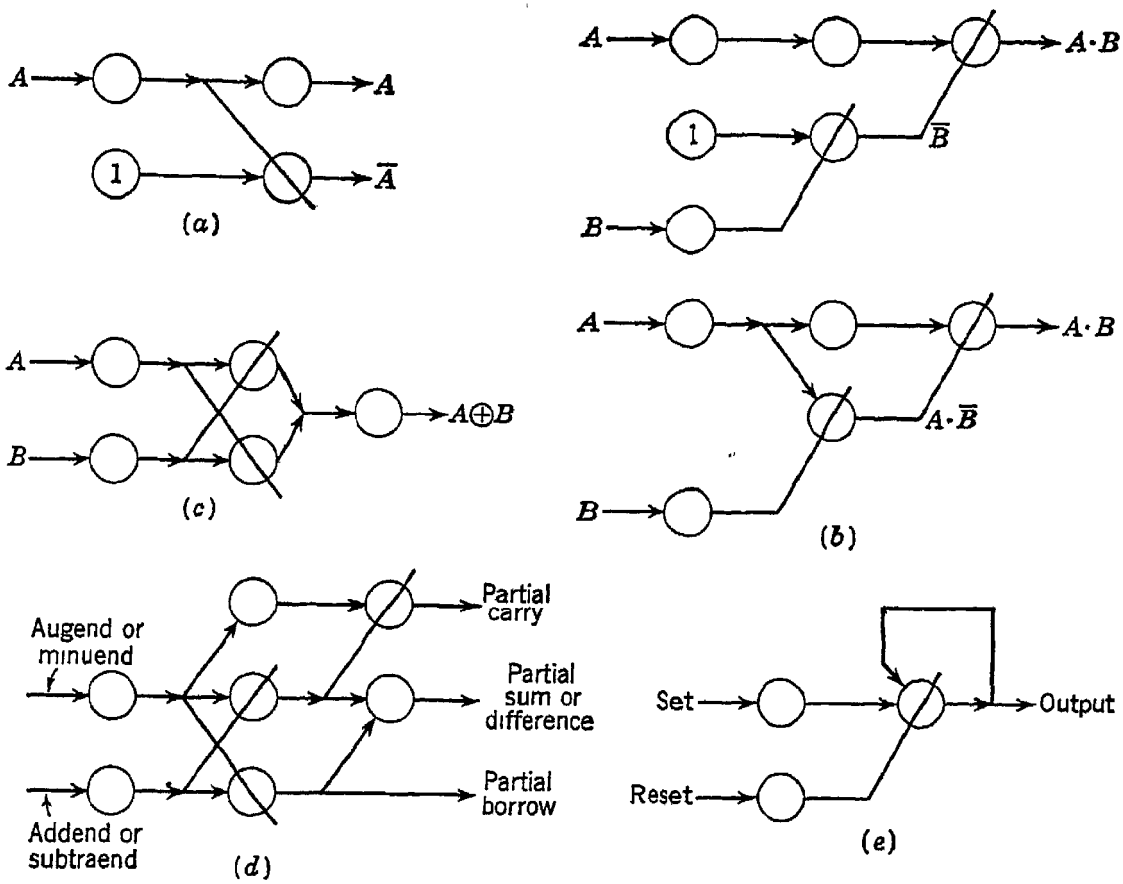



Fig. 11.35 Some computer circuits using shift-register core logic: (*a*) complementer, (*b*) AND circuits, (*c*) RING SUM element (exclusive OR), (*d*) half-adder or half-subtractor, (*e*) a flip-flop

used at the time of a later clock pulse. The AND function is not usually implemented by application of half-amplitude pulses to a core, since the currents in the various parts of a core network performing logic are less readily controlled in amplitude than those in the rows and columns of a memory matrix. Instead the AND can be realized by use of the identity

$$A \cdot B = \overline{(\bar{A} + \bar{B})} \tag{11.16}$$

One straightforward and simple method of performing logic with cores is a direct adaptation of the shift registers described in the previous section. This method has been described by Guterman, Kodis, and Ruhman,[82] and we shall adopt their convenient notation, shown in Fig. 11.34. Figure 11.34*a* shows the basic shift register; *b* is a branching circuit by

means of which a single output drives several inputs in series; *c* and *d* show two ways of performing the OR function. The circuit in *c* uses the diodes to perform the OR operation; the circuit in *d* employs separate input windings. The circuit in *d* is somewhat the more complex of the two, but it allows both $A$ and $B$ signals as well as their logical sum to be separately available. In *e* an inhibiting circuit is shown. The inhibiting winding has a few more turns than the other windings and is excited in opposition to the other input windings. Besides these basic circuits for logic, it is convenient to have a circuit for generating a continuous sequence of pulses, that is, 1's. Such a circuit appears in *f*. From these elemental circuits it is possible to construct all the logic circuits required in a computer. Examples of some circuits constructed in this way are shown in Fig. 11.35. The reader is urged to trace the behavior of each of these circuits in detail to satisfy himself that they perform the functions indicated.

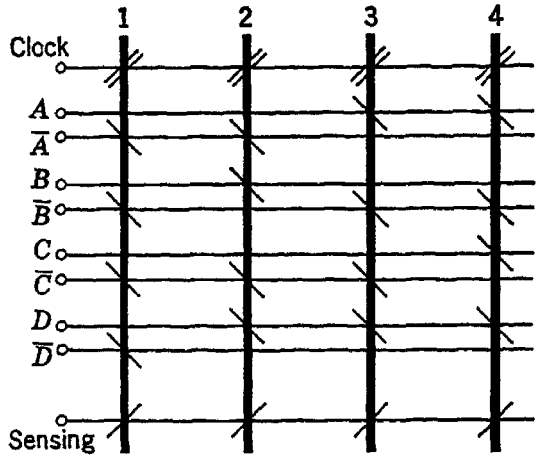


Fig. 11.36 Inhibit-core-logic realization of the function $X = ABCD + A\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}B\bar{C}\bar{D}$

Several somewhat different ways of using cores to perform logic have been proposed. One of these, called "inhibit-core logic",[33] makes extensive use of the identity of Eq. (11.16). Variables must be available in both complemented and uncomplemented forms. The circuit of Fig. 11.36 shows an example of this type of circuitry. All currents are controlled to be half-switching currents. Thus a core will be switched by a clock pulse only if it has no inhibiting currents in any of its windings. The cores in Fig. 11.36 individually perform the following functions:

$$\begin{aligned} X_1 &= \overline{\bar{A} + \bar{B} + \bar{C} + \bar{D}} = ABCD \\ X_2 &= \overline{\bar{A} + B + \bar{C} + D} = A\bar{B}C\bar{D} \\ X_3 &= \overline{A + \bar{B} + \bar{C} + D} = \bar{A}BC\bar{D} \\ X_4 &= \overline{A + \bar{B} + C + D} = \bar{A}B\bar{C}\bar{D} \end{aligned} \tag{11.17}$$

The sense winding, common to all cores, then performs the disjunction of these functions, to yield as its output

$$X = ABCD + A\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}B\bar{C}\bar{D} \tag{11.18}$$

Magnetic cores have also been used in logic circuits in a manner rather different from either of the foregoing circuits. In this mode of operation

the core behaves as a device to store information between pulses and as a magnetic amplifier to produce amplified output pulses when driven by clock pulses. Most of the logic is performed outside the cores, by means of diodes, and the only logical function the core itself performs is complementation or inversion. The two basic circuits, the inverting amplifier and the noninverting amplifier, are shown in Fig. 11.37.[34,35]

Since the inputs of each of these circuits are obtained from outputs of others, the clock pulses are seen to be connected through a series path consisting of an output winding in series with a parallel combination of

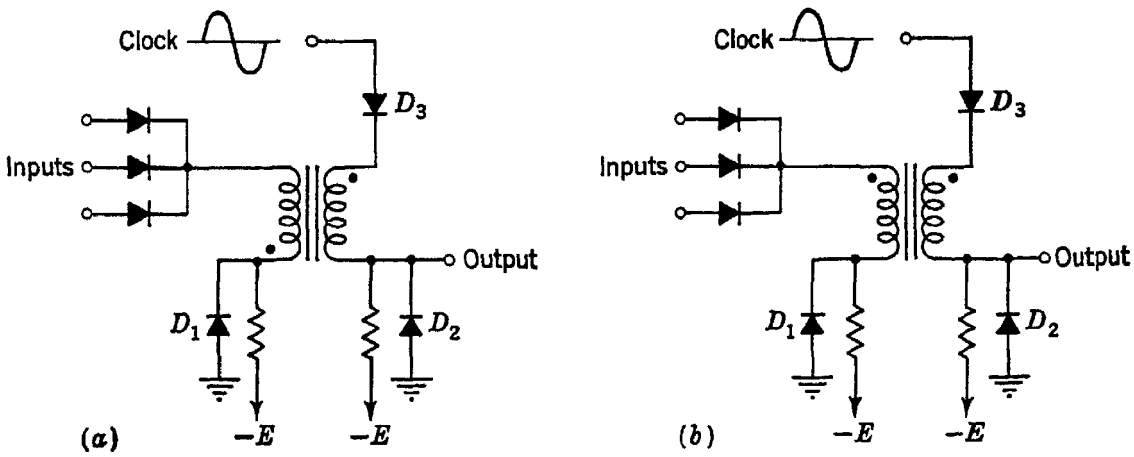



Fig. 11.37 Basic circuits for magnetic-amplifier core logic: (*a*) inverting amplifier, (*b*) noninverting amplifier

one or more input windings. The flow of current through the input windings is determined by the state of the core on which the output winding is located. If the core is already in the state to which the positive clock pulse would drive it, then only a small flux change occurs and only a small back voltage is produced. Consequently a large current flows in the output winding and in the input windings of cores connected to it. On the other hand, if the core is in the opposite state, the positive clock pulse reverses its magnetization, producing a large back emf. Thus the output winding in effect presents a large impedance to the clock signals, and only small currents can flow in input windings of cores connected to it. The principal difference between the inverting and noninverting amplifiers is in the polarity of the input winding.

A two-phase clock is used, with successive stages in an interconnection of amplifiers being driven from alternate clock phases. Each amplifier receives its input signals while its clock input is negative and delivers its output to succeeding stages while its clock input is positive. If one regards the presence of a signal as a 1 and its absence as a 0, then the noninverting amplifier functions as an OR element.

The inverting amplifier may be treated in several ways. If on the input side absence of a pulse is regarded as a 1 and on the output side

the presence of a pulse is regarded as a 1, then this circuit performs the AND function. If these conventions are both reversed, the circuit is an OR element. However, if presence of a pulse is treated on both sides as a 1, the inverter performs the function

$$\text{Output} = \bar{A}\bar{B}\bar{C} = \overline{A + B + C} \tag{11.19}$$

where $A$, $B$, and $C$ represent inputs. If absence of pulses is treated on both sides as a 1, the function realized is

$$\text{Output} = \overline{ABC} = \bar{A} + \bar{B} + \bar{C} \tag{11.20}$$

It is evident that these circuits, as well as the other systems of core logic, are sufficient for construction of the logic of a digital system.

## 11.14 Multihole cores[36,37]

The multihole core is a new type of magnetic-core element, which has attracted much interest among computer engineers since it can be used as

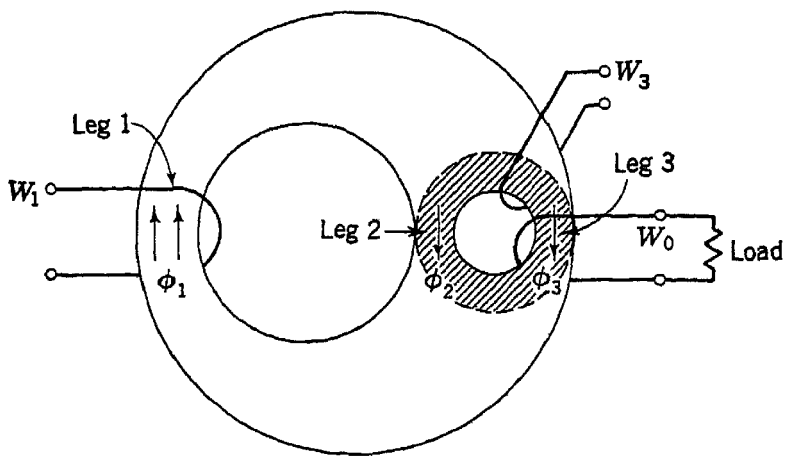



Fig. 11.38 A two-hole core, or transfluxor

an analog device, as a memory element with nondestructive read-out, or as a device for performing complex logical functions in a single element. A simple form of this versatile device is shown in Fig. 11.38.

An important property of this two-hole core is the ability of a setting pulse on one winding to block subsequent transfer of energy between other windings. This blocking action need not be total, however, and partial blocking can be used to control the amount of transfer. In Fig. 11.38, legs 2 and 3 of the core are assumed to have equal cross sections, the sum of these being somewhat less than that of leg 1. A sufficiently strong current pulse in winding $W_1$ can produce saturation flux densities in legs 2 and 3. The return path for fluxes $\phi_2$ and $\phi_3$ is by way of leg 1. If a square-loop material is used, legs 2 and 3 remain virtually saturated

after termination of the pulse on $W_1$. When the core is in this state, windings $W_3$ and $W_0$ are almost totally uncoupled from each other, at least for all values of current less than a critical maximum. A current in $W_3$ that would tend to decrease the flux in leg 2 or in leg 3 would have to increase the flux in the other leg by the same amount, since lines of magnetic flux are continuous. However, neither flux can increase, since both legs are saturated. Thus no flux change occurs, and no voltage is induced in $W_0$.

It is possible now to unblock the core by applying a current in $W_1$ opposite in direction to the current that blocked it. As the current in

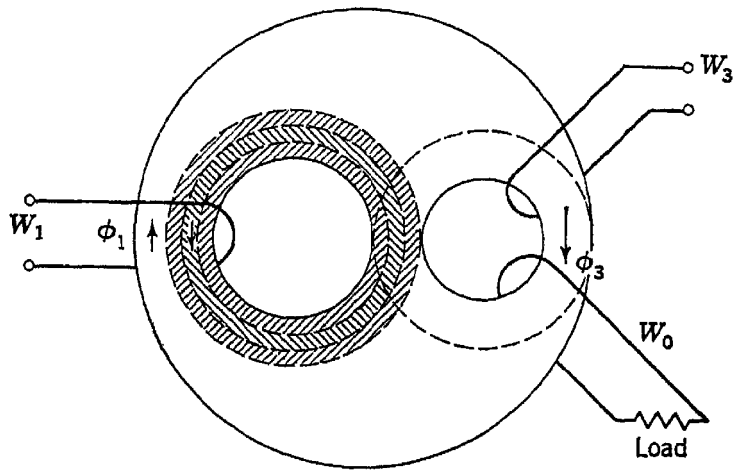



Fig. 11.39 Partial unblocking by partial reversal of flux in legs 1 and 2

$W_1$ is increased, the coercivity of the material will be exceeded first in a narrow region around the large hole, and part of the flux in legs 1 and 2 will be reversed. As the current continues to increase, this region will become broader until all the flux in leg 2 has been reversed, as shown in Fig. 11.39. The core is now totally unblocked, since an alternating current applied to $W_3$ can reverse the flux in legs 2 and 3 on alternate half-cycles. These flux reversals induce voltage in $W_0$ and thus couple $W_3$ and $W_0$. In effect, flux is transferred back and forth between legs 2 and 3 when the core is unblocked, and for this reason the device has been called a "transfluxor" by its inventors, Dr. J. A. Rajchman and Dr. A. W. Lo.

It will be observed that, if the current pulse applied to $W_1$ is sufficiently large, flux in leg 3 as well in leg 2 can be reversed. Thus the core again becomes partially blocked, since only that part of the flux which is oppositely directed (up and down) in the two legs is available for coupling $W_3$ and $W_0$. It will also be noted that it is possible to unblock the core by application of sufficiently large current pulses to $W_3$, since flux can then be reversed around the long path formed by legs 1 and 3. This type of unblocking may ordinarily be regarded as a spurious action, although in some applications it could be deliberately utilized.

A core that has been only partially blocked behaves as an analog or continuous device, since the amount of flux that can be transferred between legs 2 and 3 varies directly with the strength of the unblocking current applied to $W_1$. The device, therefore, has potential applications in digital-to-analog conversion or even in analog-to-digital conversion.

The two-hole core can be used for nondestructive read-out in a coincident-current storage matrix. The two states of the core corresponding to a stored 0 and a stored 1 are the blocked and the unblocked states, respectively. $X$- and $Y$-coordinate drive lines for writing link the large

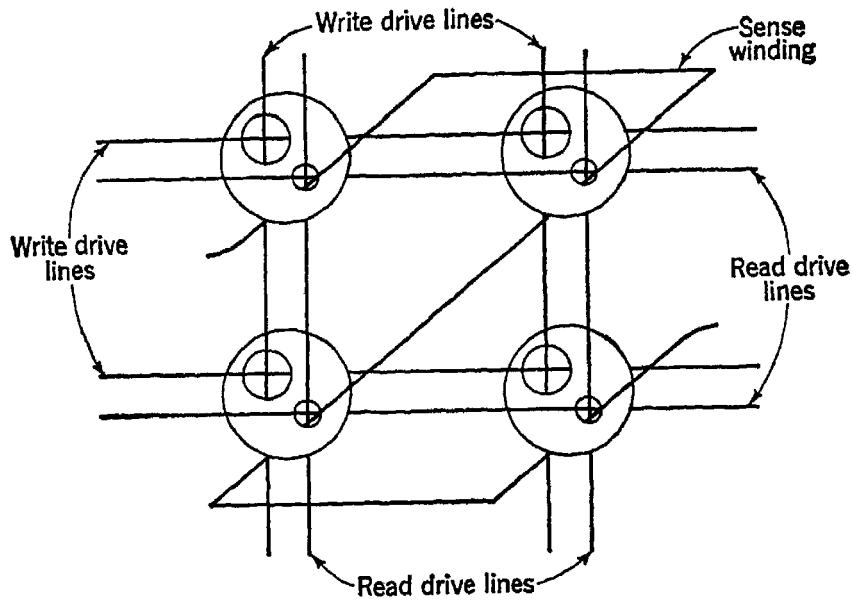



Fig. 11.40 An arrangement of two-hole cores for nondestructive read-out

hole. By proper choice of driving currents, a single excitation can be made to have essentially no effect, although excitation on both wires will switch the state of the core. The sense winding and the $X$- and $Y$-coordinate lines for reading link the small holes. The reading currents are half the amplitude required to exchange flux around legs 2 and 3. The configuration is shown in Fig. 11.40. An inhibiting winding, not shown, could link all the large holes in each plane if the memory were organized in the manner shown in Fig. 11.5.

Another version of this device, using cores with three holes, has been employed in a memory system having access time of ½ $\mu$sec.[38]

The use of the multihole core for performing logic has many attractive features. A great variety of flux paths can be established by using holes of various sizes, shapes, locations, and quantities. By means of these complex flux patterns it is possible to execute complex logical operations in a single core. The two arrangements in Fig. 11.41 show simple examples of "flux logic". The three-hole core on the left can produce an output signal if and only if input $A$ has been applied and input $B$ has been

applied later. The five-hole core performs the AND function, since it can produce an output only if all four signals $A$, $B$, $C$, and $D$ have been applied, in any order. A study of the flux patterns in the legs to verify the operation of these devices is left as an exercise for the reader.

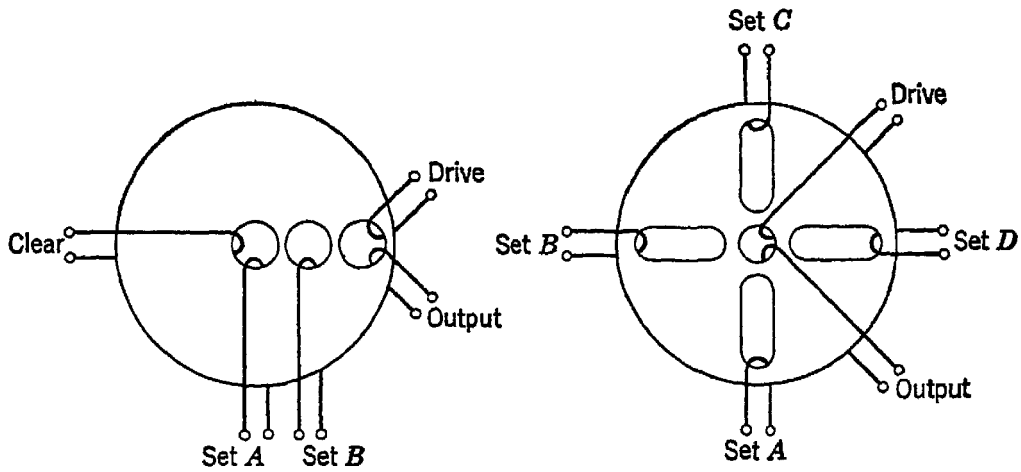



Fig. 11.41 Logic devices using multihole cores

Many other core configurations more complex than these have been investigated.[39–41] Unfortunately, one of the current shortcomings of the flux-logic method is lack of a suitable formalism for representing and manipulating the flux patterns and their logic. Several investigators have been studying this problem, however, and it is to be hoped that some useful synthesis procedures will be evolved.

## 11.15 Electrostatic storage tubes

Magnetic-core elements, which we have examined at some length in the first part of this chapter, have become the most widely adopted form of high-speed storage elements. Before the development of suitable ferrite materials, however, most high-speed computers used electrostatic storage tubes. In this form of memory, information is stored as a pattern of charges on an insulating surface inside a vacuum tube, and selection of a cell in which charge is to be inserted or detected is accomplished by directing an electron beam to the desired spot. This principle has been used as the basis for a variety of memory systems, some employing commercially available cathode-ray oscilloscope tubes as the storage devices and some using tubes especially designed to facilitate information storage. Inadequate reliability of early electrostatic storage systems resulted in the general adoption of the magnetic core in the middle 1950s. However, continuing work in storage tubes has brought about improvements, and, as of 1960, at least two large fast computers using electrostatic storage were being built in the United States.

Electrostatic memories are normally operated as parallel systems in a manner similar to the way in which core planes are used. A word is stored with each of its digits in a different tube, one tube storing the least-significant digits of all words, another all the next more significant digits, and so on. The coordinates of the storage spots in which the digits of a given word are placed are the same in all the tubes. The same deflecting voltages may then be applied to the beams of all the tubes, simultaneously selecting the same spot in all the tubes for parallel reading or writing of all the digits in a word. A typical electrostatic memory is that used in the ILLIAC, the University of Illinois computer, which was built in 1950. This system has 40 tubes for storing 40 binary digits. Each tube has 1,024 spots, giving a memory capacity of 1,024 words. The memory-access cycle allows one access, for reading or for writing, every 20 $\mu$sec.

## 11.16 Secondary emission and its effect upon target potentials

Secondary emission is the fundamental electronic phenomenon utilized in electrostatic storage tubes. Thus, to understand the action of such tubes, one should have some knowledge of the factors affecting secondary emission and of the variations in potential of insulated surfaces from which secondary emission takes place.

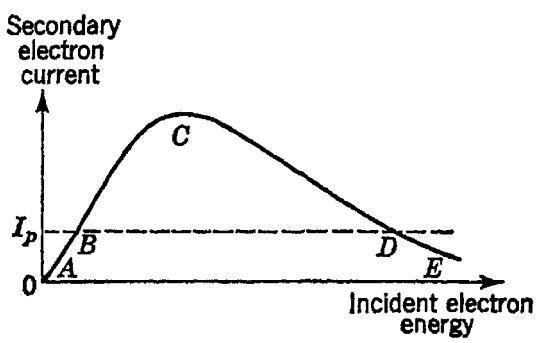



Fig. 11.42 Secondary electron current versus incident electron energy

Extensive measurements of various materials have shown that, if an electron beam falls upon a surface, the relation between secondary electron current (flowing to a collector electrode) and incident electron energy is generally as in Fig. 11.42. In the region $AB$ of this curve, incident electrons have enough energy to cause a small amount of secondary emission. This gradually increases with increasing energy of the incident electrons until at $B$, on the average, each incident (primary) electron causes the emission of one secondary electron. With further increase in primary-electron energy, each primary electron causes, on the average, the release of more than one secondary electron, and the curve continues to rise, reaching a maximum at $C$. However, further increase of primary electron energy causes the incident electrons to penetrate the target surface more deeply. A large part of the primary-electron energy is dissipated in collisions from which secondary electrons cannot escape, since they have insufficient energy to pass through the surface layers that

the high-velocity primary electrons penetrated. The curve thus falls, passing again through a point of equal primary and secondary electron currents at $D$ and falling further ($DE$) at high primary-electron energies.

This description is, of course, merely a statement of the observed facts regarding the gross electron flow from a bombarded surface, and does not explain the basic mechanism of secondary emission. Although much theoretical and experimental work has been done in an attempt to establish the detailed mechanism, there is as yet no simple and general theory.[42] For our purposes, however, a knowledge of the exact mechanism is not so important as an understanding of the target-potential variations resulting from secondary emission.

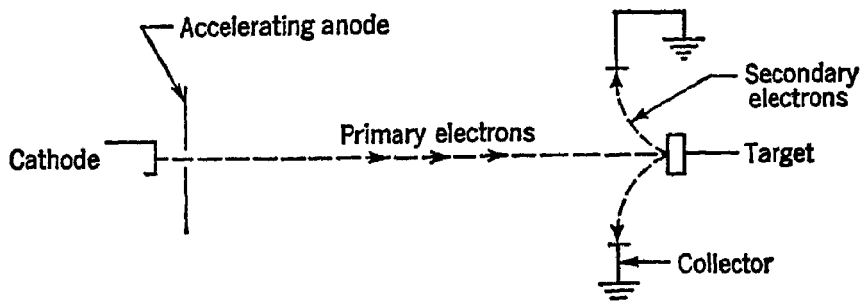



Fig. 11.43 Electrode structure showing an electron gun, an isolated target, and a collector electrode for secondary electrons

It is found that various metallic elements have a value of $\delta_{max}$ between 0.5 and 1.5, where $\delta = I_{secondary}/I_{primary}$. Such compounds as NaF, KCl, NaBr, $Cs_2O$, $BaF_2$, and BaO show $\delta_{max}$ in the range from 4 to 7; for glass, values of 2.1 to 2.9 have been found.[43] If $\delta_{max} > 1$, we may identify the two points at which $\delta = 1$ as the first crossover potential $V_p^{I}$ and the second crossover potential $V_p^{II}$. For compounds such as the foregoing, $V_p^{I}$ has been found to lie in the range of 10 to 50 volts; and $V_{p\ max}$, the voltage corresponding to $\delta_{max}$, is as low as 375 for some materials and as high as 2,000 for others. $V_p^{II}$ has been found to be in the range of 2,000 to 3,000 volts for those few materials for which it has been measured.

An isolated surface, bombarded with primary electrons of energy such that $\delta < 1$, will acquire electrons and, therefore, drop in potential; if $\delta > 1$, the surface will lose electrons and rise in potential. These potential changes continue until the surface reaches an equilibrium potential, which equals the cathode potential in the first case ($\delta < 1$) and approximates the potential of the secondary-current collector electrode in the second case ($\delta > 1$).

Consider the structure in Fig. 11.43. We shall assume that the cathode potential is adjustable and that the accelerating-anode potential is fixed. It is evident that the energy with which electrons strike the target depends only on the difference between the cathode and target

potentials. However, the picture is complicated by the variation of the target potential as it loses or acquires electrons. Suppose that initially the target potential exceeds the cathode potential by less than $V_p^{\mathrm{I}}$, the voltage of the first crossover. Then, whatever the initial voltages may have been, the target acquires electrons and drops in potential until its potential equals that of the cathode. This is shown in Fig. 11.44.

Any initial combination of target potential and cathode potential that gives a point lying between the solid and the dotted lines will result in a drop in target potential down to the line $V_T - V_k = 0$. (The cathode potential $V_k$ is negative. For convenience, the curves of Figs. 11.44, 11.45, and 11.46 use the magnitude $-V_k$ of the cathode potential as the abscissa.) If the point is initially below the line $V_T - V_k = 0$, no electrons can reach the target, and the target potential will remain unchanged (except for changes caused by leakage). If the point is above the dotted line and within the region $0 \leq V_k \leq V_p^{\mathrm{I}}$, then $\delta$ exceeds 1, and the target might be expected to rise in potential. However, since under these conditions the target potential is above the collector potential, secondary electrons encounter a retarding field and are returned to the target. The effective value of $\delta$ is thus less than 1, and the target potential falls to the cathode potential.

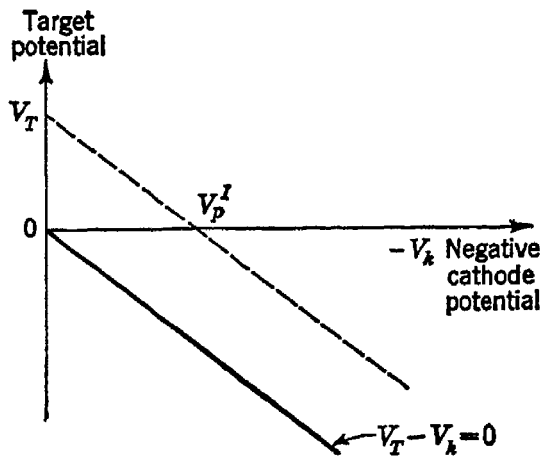



Fig. 11.44 Target potential versus negative cathode potential: Part I. (Figures 11.44, 11.45, and 11.46 are adapted from Figures 3 and 4 of M. Knoll and B. Kazan, "Storage Tubes and Their Basic Principles," 1952, by permission of the publisher, John Wiley & Sons, Inc.)

For all other initial combinations of target potential and cathode potential such that $-V_k \leq V_p^{\mathrm{II}}$, the final target potential will be in the neighborhood of the collector potential. Thus, points that lie below $V_T = 0$ and above the dotted line correspond to values of $\delta > 1$, and the target potential rises. Points above $V_T = 0$ again result in possible secondary electrons being reflected back to the target by the retarding field of the collector. The curve of final target potential for these conditions takes the form shown between $V_p^{\mathrm{I}}$ and $V_p^{\mathrm{II}}$ in Fig. 11.45. The slight upward curvature is due to the fact that secondary electrons are emitted with some initial velocities and can, therefore, reach the collector even if the target potential is slightly above the collector potential.

For values of $-V_k$ that exceed $V_p^{\mathrm{II}}$, the line $V_T - V_k = V_p^{\mathrm{II}}$ represents the ultimate equilibrium value of the target potential. All points lying above this line correspond to values of $\delta < 1$, since the primary

electrons have energy greater than the second crossover potential. The target potential, therefore, drops to this line from such points or rises to it from points below it, for which $\delta > 1$. (The line is shown somewhat less than 45° below the horizontal, in agreement with experimental observation.)

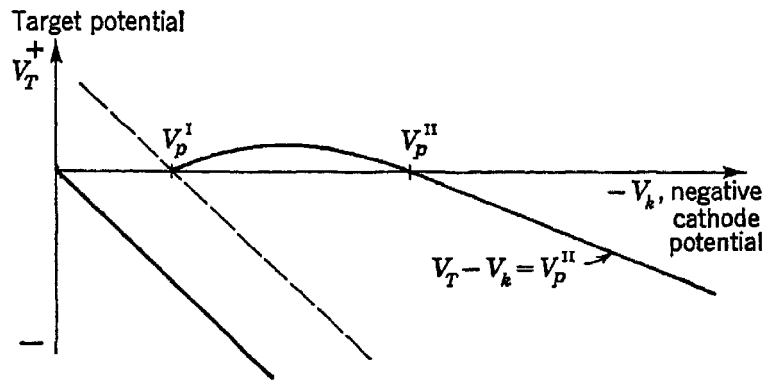



Fig. 11.45 Target potential versus negative cathode potential: Part 2

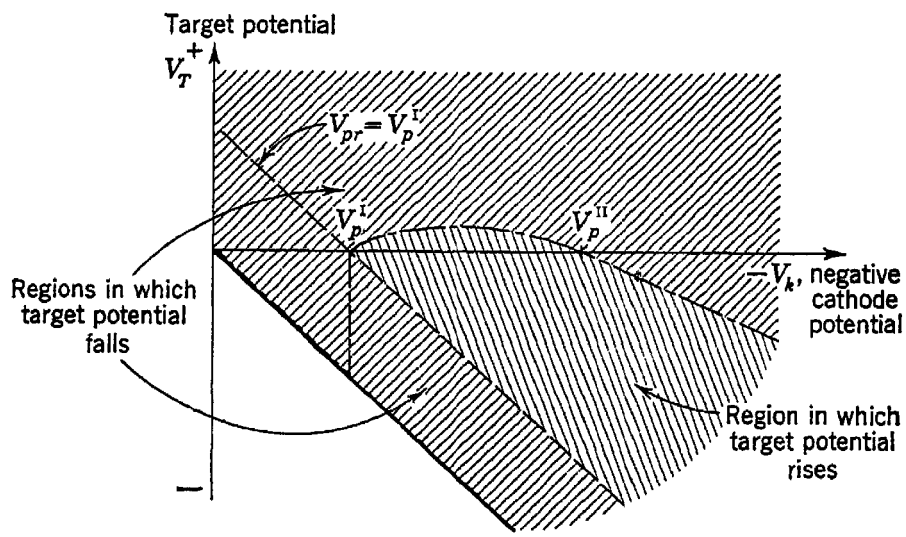



Fig. 11.46 Target potential versus negative cathode potential (complete)

The complete curve, as shown by Knoll and Kazan,[44] is given in Fig. 11.46.

## 11.17 Redistribution storage

Digital storage devices based on secondary emission may be broadly divided into two types: those which use storage by charge redistribution, and those which use storage by potential of target elements. In the redistribution system, charges are redistributed on each storage spot of the target in one or the other of two patterns identified with the binary digits 0 and 1. In the potential systems, it is the voltage of each storage spot on the target that is identified with one or the other binary digit. In both systems the target surface is made of an insulating material, and

the redistributions of charge and the changes in potential result from secondary emission caused by an incident electron beam.

Typical redistribution systems employ standard cathode-ray tubes, the phosphor coating forming the target surface on which charges are redistributed. Although the electronic phenomena at the target surface are extremely difficult to analyze quantitatively, it is possible to give an approximate qualitative description of the storage mechanism. The target is assumed to be initially at the collector potential (usually ground), and the cathode potential is sufficiently below it to ensure a value of $\delta > 1$. When the beam is directed to a particular target area, that region loses electrons and rises in potential a few volts, perhaps 5 or so. (Much

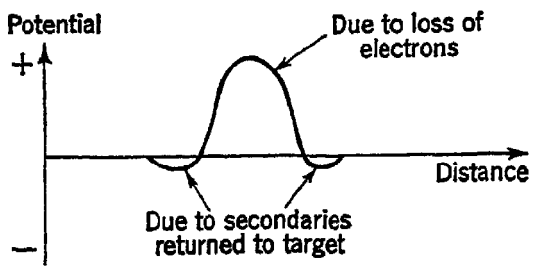



Fig. 11.47 Potential hump due to secondary-electron emission

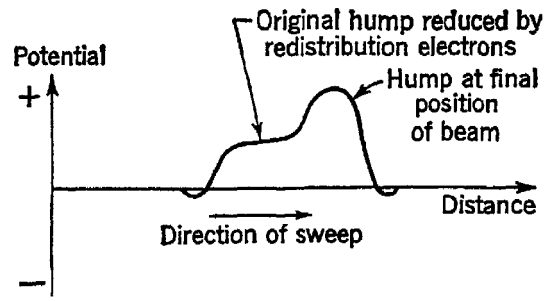



Fig. 11.48 Effect of moving the electron beam

larger rises are unlikely because the rise creates a retarding field, and most of the secondary electrons have low emission velocities that are not enough to overcome more than a few volts of potential difference.) Many of the secondaries are emitted in the wrong direction and with too low an energy to reach the collector. These electrons fall back on the target, thus effecting a redistribution of charge. A potential "hump" may be said to have been formed on the target surface, as shown in Fig. 11.47.

If the target surface is the phosphor on the inside face of a conventional cathode-ray tube, the net change in charge may be detected by capacitive coupling to a conducting surface placed against the face of the tube on the outside. A fine-mesh wire screen is suitable for this purpose and allows the pattern to be observed visually as well as electrically.

In the method originally used by Williams,[45] a second pattern to represent the other binary digit is formed by a slight movement of the beam on the target. Secondaries falling back onto the target cause the potential hump to appear as shown in Fig. 11.48.

When the "dot" pattern of Fig. 11.47 is placed on top of a dot already on the screen, the potential of the capacitively coupled backplate at first goes abruptly negative as the electron beam strikes the target. This negative pulse decays exponentially as the secondary-electron current comes to equilibrium with the primary-electron current. Finally, a posi-

tive pulse occurs when the beam is turned off, and the space-charge cloud of electrons disperses to the collector.

On the other hand, when a dot is placed on the first part of the "dash" of Fig. 11.48, there is a net loss of electrons from the target surface as the

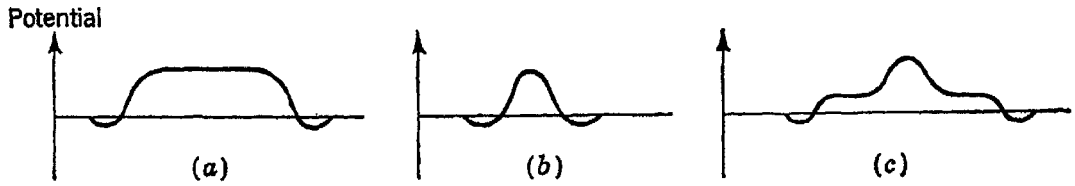



Fig. 11.49 Potential distributions with a focus-defocus pattern: (*a*) defocused, (*b*) focused, (*c*) focused spot on top of defocused

original hump is restored, and a positive pulse is observed at the backplate, preceded by a small negative pulse when the beam is first turned on. Since the dot has now replaced the dash, secondary electrons having reduced the hump at the end, it is necessary to deflect the beam to regenerate the dash if the stored information is to be retained.

The dot-dash configuration is only one of many possible patterns. The most desirable patterns are those which

1. Produce the largest signals, i.e., cause the greatest change of charge when one pattern reads the other
2. Give the greatest freedom from spurious signals due to blemishes and the greatest reliability in writing

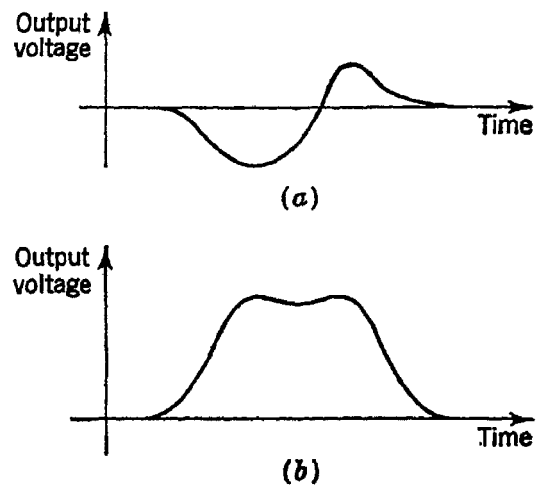



Fig. 11.50 Output pulses resulting from a circle-dot pattern. (*a*) Circle reads circle. (*b*) Circle reads dot.

One configuration that has been found effective is a focus-defocus pattern,[45,46] with the sharply focused beam creating one pattern and the defocused beam the other. The potential distributions on the target might be represented as in Fig. 11.49. The defocused beam is used for reading. If the distribution written by the defocused beam represents a 0 and that written by the focused beam a 1, we see that reading a 0 produces a small negative pulse and that reading a 1 produces a large positive pulse, as in the dot-dash configuration.

A circle-dot arrangement has also been found to be highly effective.[47] The circle is generated by superimposing 90°-out-of-phase sine waves of voltage on the deflecting voltages. Using the circle as the reading pattern and either the dot or the circle as the stored information, the output pulses of Fig. 11.50 are obtained.

## 11.18 Reliability of redistribution storage

Since some of the secondary electrons inevitably fall back upon other information-storing areas of the target, the stored information will eventually be altered, and erroneous indications will be obtained. For this reason it is necessary to regenerate periodically the entire pattern of charge stored on the target. For quantitative comparison of storage systems, we may define the "read-around ratio" to be the number of times a spot can be read in succession without causing a neighboring spot to be in error. High read-around ratios, of the order of several hundred, can be obtained by spacing the spots far apart, but this, of course, reduces the number of spots available. Close spot spacings will allow large information-storage capacity, but the read-around ratio may then be so low that the storage pattern is being regenerated most of the time and is available for read-out during only a small part of the time. For example, in a system with 1,000 information-storage spots and a read-around ratio of 50, 50 read-outs are possible before the pattern must be regenerated. If regeneration of a spot takes the same time as read-out from a spot, then regeneration of the entire memory takes 20 times as long as the time available for read-out. Of course, only spots in the neighborhood of frequently read spots need be regenerated, but systematic regeneration of the entire pattern is the only practical scheme.

Since there is some defocusing when the beam is deflected near the edge of the target area, the read-around ratio is somewhat lower for spots in this position than for those near the center. Inequality of secondary-emission characteristics may cause some spots to have much lower read-around ratios than others. The lowest value of read-around ratio for any spot determines the effective read-around ratio for the system.

Blemishes on the screen of the cathode-ray tube that cause no trouble in oscillography may be very troublesome if the tube is used for storage, since blemishes give rise to spurious signals. These signals are due to the different secondary-emission characteristics of occasional impurities in the phosphor, such as carbon spots, or of the glass behind the phosphor, which the electron beam may strike through pin-point holes in the phosphor. Blemishes can be avoided by careful production control and by careful selection of blemish-free tubes, or by moving the raster so that no spot falls on a blemish. The effect of hole-type blemishes can be reduced by using a somewhat lower accelerating potential for the beam. Williams and Kilburn[45] point out that carbon particles have $\delta < 1$ for all primary-electron energies and, therefore, fall toward cathode potential when the beam strikes them. For primary electrons of greater than about 1,000-volts energy, corresponding to $V_p^{II}$, glass also has $\delta < 1$, and spots of glass struck by electrons through holes in the phosphor fall toward $V_p^{II}$,

i.e., about 1,000 volts. In either case the potential of a blemish is substantially below that of normal storage areas. This negative potential sets up a barrier that interferes with redistribution of secondary electrons and may make it impossible to write a dash on the screen. Moreover, as the beam scans across a blemish, a spurious signal is produced. In practice it has been found satisfactory to operate with an accelerating potential of about 1,400 volts for the beam. Signal-to-noise ratios of 5:1 or 10:1 have been reported under this condition.[45] Both the circle-dot and the focus-defocus patterns have been found to be comparatively free of spurious signals due to blemishes.[46,47]

Spot spacing on the face of the tube affects reliability of the storage system by its influence upon redistribution of secondary electrons. Redistribution, as has been observed, changes rapidly as the spot spacing is increased from 1 spot diameter to 2 but does not change much beyond that. The total number of spots depends upon the redistribution, the spot diameter, and the maximum possible deflection angle. The possible spot density, in spots per square inch, decreases as the tube size is increased, assuming the same electron gun and accelerating voltage. This decrease is due largely to the increased spreading of the beam. Redistribution distances are not very different in large tubes and in small, since the secondary-electron velocity distribution is relatively unchanged for the same phosphor and the same accelerating voltage. As a consequence, the spot density does not drop as rapidly as the surface area increases, with the result that larger tubes can store more spots, but not in direct proportion to their areas.[47] Systems storing 1,024 spots on a 3-in. cathode-ray tube are quite common. Selection of one out of 32 horizontal deflection voltages and one out of 32 vertical deflection voltages permits the electron beam to be positioned to any spot.

## 11.19 Storage by potential of target elements

Two methods have been used for storage of information by establishment of a particular potential on a target element. These are

1. Equilibrium storage. Each spot is made to assume an equilibrium position (on the curve of Fig. 11.46) that corresponds to the digit being represented.

2. Bistable storage using a holding beam. Each spot is made to assume either the collector potential or the potential of the cathode of an auxiliary electron beam which is provided to maintain the potential distribution on the target.

**Equilibrium storage.** In equilibrium storage the target elements can be made to assume any potential along the equilibrium curve. Therefore, this method is capable of storing continuously varying as well

as digital information, although two potential levels are all that are usually used in computer applications. For example, to store binary information on one of the straight-line segments of the equilibrium curve, the cathode potential may be made negative with respect to the collector by one or the other of two potentials less than $V_p^{\mathrm{I}}$ (or greater than $V_p^{\mathrm{II}}$). Then, when the beam is turned on, the target spot assumes the cathode potential. Alternatively, the curved portion of the equilibrium curve may be used by setting the cathode at a potential between $V_p^{\mathrm{I}}$ and $V_p^{\mathrm{II}}$ volts below the collector. The collector itself is then set to one or the other of two potentials, which represent the digits 1 and 0, and the beam is turned on to write the digit at the desired location.

This method was used in one of the early storage tubes built at MIT and described by Forrester.[48] In this tube the cathode is maintained at −2,000 volts, and the collector is set at either plus or minus 100 volts. With the collector at +100 volts, a target spot at which the beam is directed rises in potential to +100 volts, if enough beam current is provided and enough time is allowed. Conversely, with −100 volts on the collector, the target spot drops to −100 volts. In order to avoid redistribution effects, the tube contains a barrier grid placed close to the target surface. Most secondary electrons are drawn to or through this grid, and the few that fall back on the target land close to the initiating spot and have little effect on neighboring spots. The barrier grid becomes an effective collector electrode, and it is to this electrode that the ±100 volt potentials are applied.

To read out information in this tube, the collector is set to zero volts and the beam is turned on to the desired spot. An output voltage pulse of the order of 0.1 volt is observed at the capacitively coupled backplate. The pulse is negative if the spot was positively charged and positive if the spot was negatively charged. Since read-out is destructive, it is necessary to rewrite the information. If a positive output pulse occurs, the collector is then raised to +100 volts before the beam is turned off; if a negative pulse occurs, the collector is dropped to −100 volts before the beam is turned off, to regenerate the stored information. It should be noted that, in tubes of this type, it is also possible to write information by pulsing the backplate instead of the collector, although it is convenient to have separate electrodes for reading and for writing.

**Bistable storage with a holding beam.** For long-time storage, a holding beam to maintain the information pattern on the target has been found helpful. The holding beam is broadly focused and floods the entire target area with low-energy electrons that hold spots at either the collector potential or the potential of the cathode of the holding beam. The collector may be assumed to be at ground potential and the holding-beam cathode somewhat more than $V_p^{\mathrm{I}}$ volts below ground. The com-

bined effects of leakage across the surface and of secondary-electron "splatter" tend to make spots on the target change potential. A spot that was initially at the potential of the holding-beam cathode tends to rise in potential but cannot do so, since it is struck by holding-beam electrons for which $\delta < 1$. Similarly, a spot that was initially at collector potential tends to fall in potential but is prevented from doing so by the holding beam, which in this case is characterized by $\delta > 1$.

An example of such a tube is the MIT electrostatic storage tube formerly used in the Whirlwind I computer[49] (and later replaced by magnetic cores). Two electron guns are used, one for holding and one for reading and writing. The collector electrode, a 100-mesh wire screen 0.015 in. in front of the target, is held at ground potential. The holding-gun cathode is at $-100$ volts and the READ-WRITE-gun cathode at $-2{,}000$. The target is a mosaic of beryllium on a mica sheet 0.005 in. thick. A silver film on the back serves as a backplate. Since the mosaic elements are small, the beam covers many of them in reading or in writing. To put a target area at collector potential it is enough to direct the writing beam upon it. If the area was below collector potential, $\delta > 1$ and the potential rises. To put a target area at holding-gun-cathode potential, the backplate is first pulsed positively 100 volts. Areas initially at collector potential are now at $+100$, due to capacitive coupling with the backplate, and, when the writing beam strikes them, they fall to collector potential. Upon removal of the $+100$-volt backplate pulse, these areas fall to $-100$ volts.

Read-out is accomplished by pulsing the backplate positively with a 50-volt pulse applied through the output resistor and then observing the polarity of the signal obtained at the backplate when the beam is directed to the desired spot. A positive pulse indicates that the spot was negative, and a negative pulse that the spot was positive. In order to avoid erasure of stored information, the reading-beam current must be small enough not to shift the target area potential so far that the holding beam will complete the shifting of the spot when the 50-volt pulse is removed from the backplate. If target elements do not shift completely to the collector potential during reading, then the holding beam will restore them to their original potential after removal of the 50-volt backplate pulse.

In practice, the READ-WRITE beam of the MIT tube was intensity-modulated at 10 megacycles during read-out to facilitate separation of the signal pulse from the 50-volt sensing pulse. If the 10-megacycle voltage at the backplate was observed to be in phase with the 10-megacycle component of electron current in the beam, then the spot was evidently negative; if these were out of phase, the spot was evidently positive (i.e., at collector potential).

Another tube employing bistable storage and a holding beam is the selectron, designed by J. A. Rajchman.[50] This tube employs individual cathodes and individual storage elements and, hence, has no deflection system. It does have a selection system, however, which allows the beam from any cathode to be directed at its target when desired. In principle the selectron is a very elegant device from both a mathematical and an engineering point of view. In practice, however, its complexity and the large driving power required for its operation have made it much less attractive than some of the simpler systems. The selectron is no longer being manufactured.

## 11.20 Storage by magnetic recording on moving surfaces

Although there are many incentives for using very high-speed memories such as magnetic cores or electrostatic tubes, the high cost of these devices makes it impractical to use them for all the storage functions in a computer. High-speed computers usually contain, as auxiliary storage, some form of magnetic drum, tape, or disk,[51] and for slow computers the magnetic drum often provides the principal storage.

Storage systems using magnetic recording have at least two features in common:

1. Access to the desired storage cell is accomplished by a mechanical movement of the storage medium relative to the read-out element.
2. This mechanical motion of magnetized regions in the storage medium past the conductors of the read-out element induces the output-signal voltages.

Mechanical access has the conspicuous advantage that it is simple and inexpensive by comparison with electrical-access systems. However, it has the prominent disadvantage that it is much slower than electrical access. For example, magnetic drums are often rotated by 60-cycle induction motors at speeds slightly under 3,600 rpm. A single rotation requires about 17 msec; thus, on the average, the access time is approximately 8.5 msec. This time is of the order of $10^3$ times as great as the access time of a magnetic-core memory.

We have already seen, in relation to locating a cell in memory, in Sec. 9.12, an example of the way in which the surface of a typical magnetic drum is laid out into data-storing and data-locating tracks. Most drum surfaces are organized in ways not greatly different from this. The principal differences among drum systems lie in the drum dimensions, the rotation speeds, the information-storing density, the mode of recording, and the type of selection system. Memory drums have been built which are only 2 in. in diameter and which spin at more than 20,000 rpm. At the other extreme are drums 3 ft in diameter that spin at only 750 rpm.

Obviously, a drum cannot be both large in diameter and fast, because of the severe mechanical problems that would be imposed by such a combination. Most drums have diameters in the range of 8 to 10 in. Axial lengths range from 2 or 3 in. to 2 or 3 ft. A common speed is 3,500 to 3,600 rpm, although one computer, of which nearly 2,000 have been built, uses a speed of 12,500 rpm.

Information-storing density is determined by the number of digits stored per inch on a track around the drum circumference and by the number of tracks per inch along the drum axis. Pulse-packing densities on a track range from 50 pulses/in. to 600 to 800 pulses/in., and the number of tracks per axial inch usually falls between 5 and 20. Densities as low as 50 pulses/in. represent extremely conservative design, and 80 to 100 pulses/in. is the region in which most packing densities fall. Greater densities require very small gaps in the magnetic recording head and very close spacing between the head and the magnetic medium. Densities as great as 600 to 800/in. can be achieved only with the head directly in contact with the magnetic medium, and, to minimize the mechanical wear in such a system, it is necessary that the velocity of the medium past the head be low. Thus the price for high density is slow speed.

Drum speeds, dimensions, and packing densities commonly fall within ranges in which the upper and lower limits are in a ratio of about 10:1. A "typical" drum with characteristics approximately in the center of these ranges might have the following specifications:

| | |
|---|---|
| Diameter | 10 in. |
| Length | 18 in. |
| Speed | 3,550 rpm |
| Pulses per inch | 100 |
| Tracks per inch | 10 |

Such a drum can store approximately 565,000 binary digits.

## 11.21 Some engineering considerations in digital magnetic recording[52–56]

The magnetic surface of a digital recording medium should be a material having high remanent-flux density and low coercivity. Almost every conceivable method has been used to put such surfaces on drums. Magnetic alloys have been electroplated on drums; thin sleeves of magnetic alloy have been slipped over drums by shrink-fitting; drums have been wrapped with magnetic wire, which is then ground smooth; and both the black and the red iron oxides have been suspended in lacquer and slowly sprayed onto drum surfaces. In each case the net result consists of a thin magnetic cylinder over a rigid and rugged mechanical structure. Adja-

cent to the magnetic surface are mounted the recording heads that establish flux patterns in the tracks around the drum.

Figure 11.51*a* shows a magnetic head mounted next to the magnetic surface of a drum. Figure 11.51*b* shows an enlarged view of the pole-tip region and of the flux pattern, and *c* shows the flux pattern of a magnetized part of the surface after it has passed by the head. To establish a large flux density in the magnetic surface, the spacing between the head

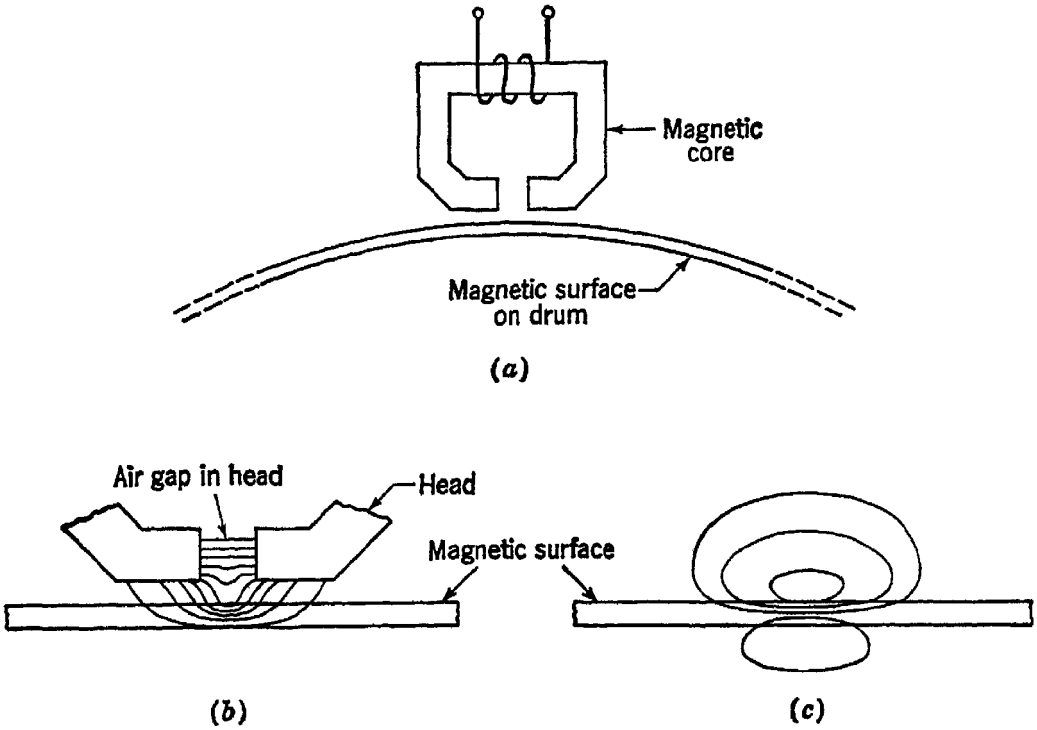



Fig. 11.51 (*a*) A READ-WRITE element situated next to a magnetic drum surface, (*b*) approximate flux configuration under the READ-WRITE head, (*c*) approximate flux configuration of a magnetized spot away from the head

and the surface should be small. (If the surface velocity is not so large as to produce severe wear at the interface, the head can be placed directly against the surface.) It might appear desirable also that the air gap in the head be long compared with the head-to-surface distance so that a large part of the flux in the head would pass through the surface. However, a long air gap lengthens the magnetized region of the surface and lowers packing density. In practice the long gap is not necessary, since it is possible to cause complete saturation of the surface with heads having gaps of the order of 0.0005 to 0.001 in. and spaced from the surface by about the same distance. It should be noted that the head is made of a soft magnetic material with low remanent-flux density. Moreover, the head itself should not be driven into saturation, since driving currents in excess of those which saturate the head produce leakage flux which may alter the magnetization of adjacent spots. It should also be pointed out

that the surface may move a distance as great as several gap lengths during the writing pulse.

A plot of the longitudinal component of magnetic flux density $B$ against distance along the track, for the ideal and actual cases, is shown in Fig.

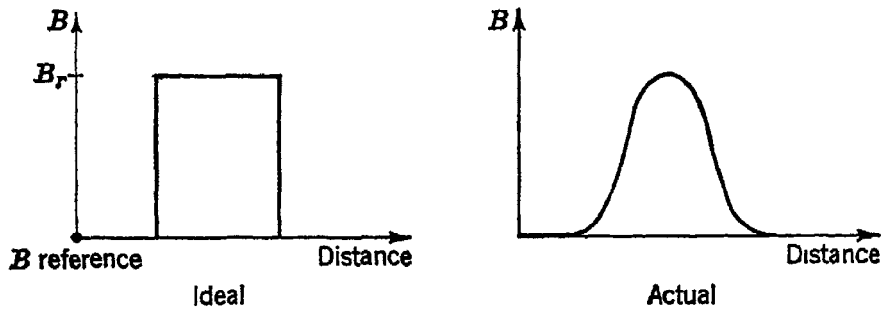



Fig. 11.52 Ideal and actual flux distributions for a magnetized spot on a track

11.52. Among the factors that cause the flux distribution to be rounded are the following:

1. Nonuniformity of the flux density in the head-to-surface space
2. Nonrectangularity of the hysteresis loop of the surface material
3. Motion of the surface during the nonzero rise and fall times of the current pulse

Since the voltage induced in the coil of the magnetic head as a magnetized region passes under the head is proportional to the time rate of change of the flux, the waveform of the voltage is approximately as shown in Fig. 11.53, if the flux starts to fall as soon as it has reached its peak. A sequence of magnetized spots, with the flux density returned to the reference level after each spot and no spacing between, gives rise to the flux distribution and read-out voltage shown in Fig. 11.54. These curves

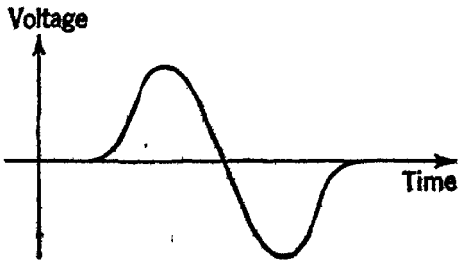



Fig. 11.53 Voltage induced in the head by the actual flux distribution of Fig. 11.52

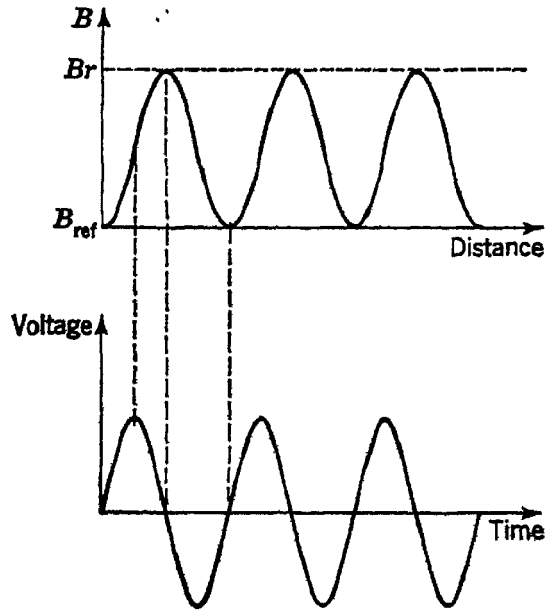



Fig. 11.54 Flux distribution and read-out voltages for a sequence of magnetized spots.

are seen to be closely sinusoidal. The maximum possible peak-to-peak range for the curve of flux density is from positive saturation to negative saturation. Whatever the reference flux density, whether it be zero or $-B_r$ or some other level, it is evident that, if the magnetized regions are made to overlap, the resultant flux-distribution curve will have a smaller amplitude. This situation is shown in Fig. 11.55. Such overlapping can be caused by driving the heads into saturation (an avoidable malpractice) or by packing the pulses too closely along the track. The limit to the pulse-packing density is imposed by the spurious output voltages produced by variations in the head-to-surface spacing, nonuniformities in the magnetic surfaces, and intertrack interference.

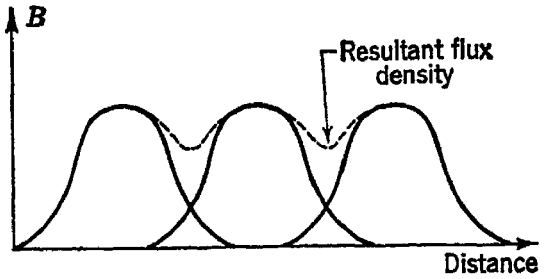



FIG. 11.55 Reduction of flux-density variation due to overlapping of magnetized regions

Several different methods can be used to identify the binary digits 0 and 1 of a digit group with flux levels or flux changes along the track. Many of these methods may be classified broadly as return-to-zero (RZ) systems or non-return-to-zero (NRZ) systems. In the RZ methods the flux density is always returned to the reference level after each digit has been recorded. In one variant of the RZ method the reference level for the flux density is zero, and 1's are recorded by one polarity of $B_r$ and 0's by the other. In the other version the reference level is $-B_r$. 1's are recorded by driving the surface to $+B_r$, 0's by leaving the surface at $-B_r$. This method requires the use of an auxiliary track with synchronizing pulses recorded on it to allow the system to discriminate between 0's and regions of no information.

Variants of the NRZ method also exist. In one of these the flux density is $+B_r$ while 1's are being recorded and $-B_r$ while 0's are being recorded. In another variant it is the change of polarity of the flux density rather than the polarity itself that is significant; each change, from $+B_r$ to $-B_r$ or from $-B_r$ to $+B_r$, indicates the presence of a 1. Both these methods also require a synchronizing track.

The four methods are summarized in Fig. 11.56. No one is outstandingly superior to the others, and all have been used successfully. In some applications the use of a synchronizing track may be undesirable, and the use of the first of the RZ methods would be indicated. With this method, however, it is not easy to replace old information merely by recording new information, and a separate erasing head is needed. The NRZ methods have the advantage of requiring a lower cutoff frequency for the head and the sensing and driving circuits. In the RZ methods

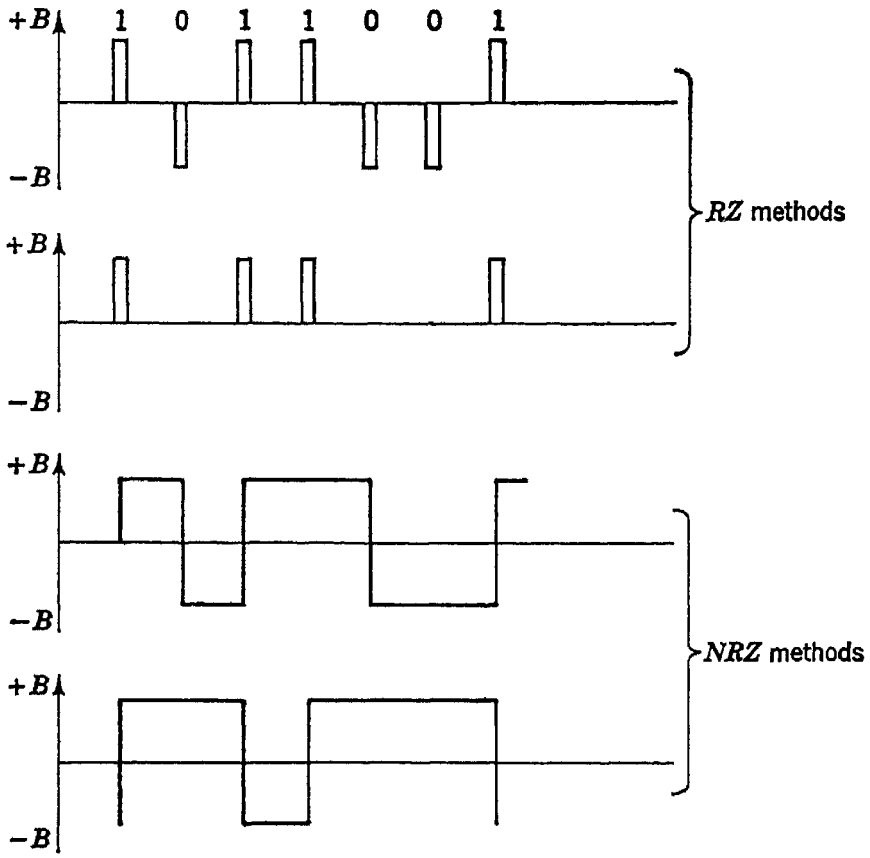



Fig. 11.56 Return-to-zero (RZ) and non-return-to-zero (NRZ) recording methods

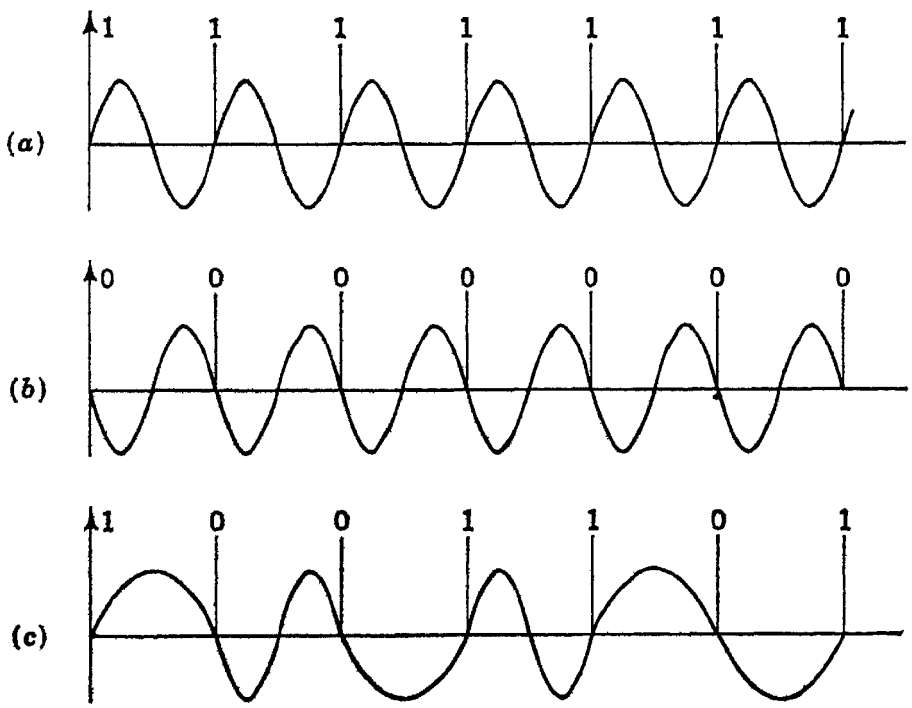



Fig. 11.57 Some typical flux distributions with a phase-modulation type of recording

the worst digit group, a sequence of 1's, requires a full cycle of flux variation for each digit. In the NRZ methods the worst digit groups require a full cycle of flux variation for each two digits. The need for continuous current in the driving heads during recording with the NRZ system means that transformer coupling cannot be used. Thus, heads for NRZ recording usually have multiturn drive windings so that vacuum tube or transistor drivers of reasonable size can be used to provide the needed mmf.

The RZ and NRZ methods of recording may be regarded as amplitude-modulation methods. It is also possible to record information by a phase-modulation method, with positive rate of change of flux, at the strobing instant, representing one binary digit and a negative rate of change of flux representing the other.[57] Figure 11.57 shows flux distributions for a sequence of 1's, a sequence of 0's, and a mixed sequence, using the phase-modulation method.

## 11.22 Storage using magnetic tape[58–63]

Magnetic-tape units are often used with digital computers as a form of supplemental storage. A single reel of tape can store several million binary digits, and large computers sometimes have as many as 10 or 20 tape units. The most efficient way to use tapes is to transfer information to or from them in large blocks rather than by individual words. Tapes on which large quantities of output data from the computer have been recorded may be removed and placed on separate "off-line" equipment for preparation of printed records or for other types of processing.

The use of magnetic tape involves engineering problems somewhat different from those encountered in connection with magnetic drums. Except for a very short piece of tape, which might be connected end to end, tape is not continuous and cannot be continuously in motion. Starting and stopping the tape quickly involves some very severe mechanical problems, but systems are in production in which tapes can be brought up to a speed of 100 in./sec or halted from that speed in well under 10 msec. Since the tape travels a few tenths of an inch during acceleration or deceleration, blank spaces must be left between recorded blocks of data. To increase the speed of information transfer, several channels are recorded in parallel. One widely used tape system has seven parallel channels for recording the 6 bits of an alphanumeric character and 1 bit for parity-checking the group. Such tapes are approximately ¾ in. wide. High-speed tapes 3 in. wide with 36 channels are also in use.

A serious problem in the use of magnetic tapes is the presence of minor blemishes or pinholes in the magnetic surface which can cause "dropouts" of recorded digits. It is customary to inspect each new tape minutely

before it is used. Blemishes detected in this inspection can be identified and avoided during recording. However, new flaws are likely to be produced during use of the tape as it is subjected to the mechanical strains of starting, stopping, and flexing. Moreover, particles of dust on the tape can also cause dropouts. For these reasons parity checking is usually used in tape transfers.

Most magnetic tapes are made of a thin plastic material coated with a magnetic iron oxide. The total thickness of the coated tape is about 0.003 in. Some tapes have a thin plastic coating over the magnetic surface to reduce friction with the head and to prevent deterioration of the magnetic coating. Metal tapes with an electroplated magnetic surface have also been used, but these are somewhat bulkier and less flexible than the plastic tapes.

Information on a tape can be located in either of two ways: (1) by the use of block addresses permanently recorded on the tape, or (2) by merely counting blocks from a reference point. The first method requires precise control to ensure that new information erases old information and is written into the same place on the tape. The second method requires less precise control, since there are no permanently recorded addresses assigning a given section of tape to a block. Information on the tape is altered by being read from the tape, altered in the computer, and stored in an erased section of the tape.

## 11.23 Other forms of memory

Many types of storage system other than those treated in the foregoing portions of this chapter have been used in computers, and new systems are constantly undergoing development. Some of these other systems (for example, the acoustic delay-line memory) are of historical importance; many involve highly ingenious engineering techniques; still others are based upon physical phenomena that are only partially understood. By and large, these systems have in common the characteristic that, as of the date of publication of this work, they are not in significant current use. For the reader who wishes to pursue the topic further, the following list of memory systems and bibliographical references is included:

1. Acoustic delay lines[64–72]
2. Magnetostrictive delay lines[73–77]
3. Diode-capacitor storage[78]
4. Ferroelectric-capacitor storage[79]
5. Spin-echo storage[80,81]
6. Photographic storage[82]

## REFERENCES

*Items Referred to by Number:*


1. Eckert, J. P., Jr., A survey of digital computer memory systems, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1393–1406.
2. Lo, A. W., Transistor trigger circuits, *Proc. IRE*, vol. 40, no. 11, November, 1952, pp. 1531–1541.
3. Forrester, J. W., Digital information storage in three dimensions using magnetic cores, *J. Appl. Phys.*, vol. 22, January, 1951, pp. 44–48.
4. Rajchman, J. A., Static magnetic matrix memory and switching circuits, *R C A Rev.*, vol. 13, no. 2, June, 1952, pp. 183–201.
5. Papian, W. N., A coincident-current magnetic memory cell for the storage of digital information, *Proc. IRE*, vol. 40, no. 4, April, 1952, pp. 475–478.
6. Rajchman, J. A., A myriabit magnetic core matrix memory, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1407–1421.
7. Foss, E. D., and R. S. Partridge, A 32,000-word magnetic-core memory, *IBM J. Research and Develop.*, vol. 1, no. 2, April, 1957, pp. 102–109.
8. Papian, W. N., New ferrite-core memory uses pulse transformers, *Electronics*, March, 1955, pp. 194–197.
9. Freeman, J. R., Pulse responses of ferrite memory cores, *Proc. WESCON Computer Sessions, 1954*, pp. 50–61.
10. Wylen, J., Pulse response characteristics of rectangular-hysteresis-loop ferromagnetic materials, *Trans. AIEE*, vol. 71, pt. I, 1953, pp. 648–656.
11. Mitchell, J. L., and K. H. Olsen, TX-O, a transistor computer with a 256 by 256 memory, *Proc. Eastern Joint Computer Conf., Dec. 10–12, 1956*, pp. 93–101.
12. Stuart-Williams, R., M. Rosenberg, and M. A. Alexander, Recent advances in coincident-current magnetic memory techniques, paper presented at *1954 Meeting ACM*.
13. Menyuk, N., and J. B. Goodenough, Magnetic materials for digital-computer components I: A theory of flux reversal in polycrystalline ferromagnetics, *J. Appl. Phys.*, vol. 26, no. 1, January, 1955, pp. 8–18.
    Menyuk, N., Magnetic materials for digital computer components II: Magnetic characteristics of ultra-thin molybdenum-permalloy cores, *J. Appl. Phys.*, vol. 26, no. 6, June, 1955, pp. 692–697.
14. Williams, H. J., Magnetic domains, *Bell Lab. Record*, vol. 30, October, 1952, pp. 385–396.
15. Littman, M. F., Ultrathin magnetic alloy tapes with rectangular hysteresis loops, *Trans. AIEE*, vol. 71, pt. II, July, 1952, pp. 220–223.
16. Sands, E. A., Behavior of rectangular hysteresis loop magnetic materials under current pulse conditions, *Proc. IRE*, vol. 40, no. 10, October, 1952, pp. 1246–1250.
17. Albers-Schoenberg, E., Ferrites for microwave circuits and digital computers, *J. Appl. Phys.*, vol. 25, no. 2, February, 1954, pp. 152–154.
18. Harvey, R. L., I. J. Hegyi, and H. W. Leverenz, Ferromagnetic spinels for radio frequencies, *R C A Rev.*, vol. 11, no. 3, September, 1950, pp. 321–363.
19. Brown, D. R., D. A. Buck, and N. Menyuk, A comparison of metals and ferrites for high-speed pulse operation, *Trans. AIEE*, vol. 73, pt. I, January, 1955, pp. 631–635.
20. Constantine, G., Jr., A load-sharing matrix switch, *IBM J. Research and Develop.*, vol. 2, no. 3, July, 1958, pp. 204–211.
21. Rajchman, J. A., Ferrite apertured plate for random-access memory, *Proc. Eastern Joint Computer Conf., Dec. 10–12, 1956*, pp. 107–115.

22. Newhouse, V. L., The utilization of domain-wall viscosity in data-handling devices, *Proc. IRE*, vol. 45, no. 11, November, 1957, pp. 1484–1491.
23. McMahon, R. E., Impulse switching of ferrites, *Proc. Eastern Joint Computer Conf., Dec. 3–5, 1958*, pp. 31–33.
24. Pohm, A. V., and S. M. Rubens, A compact coincident-current memory, *Proc. Eastern Joint Computer Conf., Dec. 10–12, 1956*, pp. 120–123.
25. Buck, D. A., and W. I. Frank, Nondestructive sensing of magnetic cores, *Trans. AIEE*, vol. 72, pt. I, January, 1954, pp. 822–830.
26. Wang, A., and W. D. Woo, Static magnetic storage and delay line, *J. Appl. Phys.*, vol. 21, January, 1950, pp. 49–54.
27. Wang, A., Magnetic delay line storage, *Proc. IRE*, vol. 39, no. 4, April, 1951, pp. 401–407.
28. Kodis, R. D., S. Ruhman, and W. D. Woo, Magnetic shift register using one core per bit, *1953 IRE Conv. Record*, pt. 7, pp. 38–42.
29. Sands, E. A., An analysis of magnetic shift register operation, *Proc. IRE*, vol. 41, no. 8, August, 1953, pp. 993–999.
30. Karnaugh, M., Pulse-switching circuits using magnetic cores, *Proc. IRE*, vol. 43, no. 5, May, 1955, pp. 570–583.
31. Loev, D., W. Miehle, J. Paivinen, and J. Wylen, Magnetic core circuits for digital data-processing systems, *Proc. IRE*, vol. 44, no. 2, February, 1956, pp. 154–162.
32. Guterman, S., R. D. Kodis, and S. Ruhman, Logical and control functions performed with magnetic cores, *Proc. IRE*, vol. 43, no. 3, March, 1955, pp. 291–298.
33. Andrews, L. J., A technique for using magnetic cores as logical elements, *Proc. Eastern Joint Computer Conf., Dec. 10–12, 1956*, pp. 39–46.
34. Torrey, R. D., and T. H. Bonn, A 2.5-megacycle ferractor accumulator, *Proc. Eastern Joint Computer Conf., Dec. 10–12, 1956*, pp. 50–53.
35. Ramey, R. A., The single-core magnetic amplifier as a computer component, *Trans. AIEE*, vol. 71, pt. I, 1952, pp. 442–446.
36. Rajchman, J. A., and A. W. Lo, The transfluxor: A magnetic gate with stored variable setting, *R C A Rev.*, vol. 16, no. 2, June, 1955, pp. 303–311.
37. Rajchman, J. A., and A. W. Lo, The transfluxor, *Proc. IRE*, vol. 44, no. 3, March, 1956, pp. 321–332.
38. Lawrence, W. W., Jr., Recent developments in very-high-speed magnetic storage techniques, *Proc. Eastern Joint Computer Conf., Dec. 10–12, 1956*, pp. 101–103.
39. Abbott, H. W., and J. J. Suran, Multihole ferrite core configurations and applications, *Proc. IRE*, vol. 45, no. 8, August, 1957, pp. 1081–1093.
40. Crane, H. D., A high-speed logic system using magnetic elements and connecting wire only, *Proc. IRE*, vol. 47, no. 1, January, 1959, pp. 63–73.
41. Lockhart, N. F., Logic by ordered flux changes in multipath ferrite cores, *1958 IRE Natl. Conv. Record*, pt. 4, pp. 268–278.
42. Bruining, H., "Physics and Applications of Secondary Electron Emission," 1954, McGraw-Hill Book Company, Inc., New York, and Pergamon Press, Ltd., London, pp. 159–176. (Contains an excellent bibliography of over 400 items.) See also Zworykin, V. K., and G. A. Morton, "Television," John Wiley & Sons, Inc., New York, 1940, pp. 28–34.
43. Bruining, *op. cit.*, pp. 39–53.
44. Knoll, M., and B. Kazan, "Storage Tubes and their Basic Principles," John Wiley & Sons, Inc., New York, 1952, pp. 1–13.
45. Williams, F. C., and T. Kilburn, A storage system for use with binary-digital computing machines, *Proc. IEE London*, vol. 96, pt. II, 1949, pp. 183–202. Same paper also appears in vol. 96, pt. III, pp. 81–100. Discussion, vol. 97, pt. III, pp. 453–454.

46. Williams, F. C., T. Kilburn, C. N. W. Litting, D. B. G. Edwards, and G. R Hoffman, Recent advances in cathode-ray tube storage, *Proc. IEE London*, vol. 100, pt. II, 1953, pp. 523–539.
47. Eckert, J. P., H. Lukoff, and G. Smoliar, A dynamically regenerated electrostatic storage system, *Proc. IRE*, vol. 38, no. 5, May, 1950, pp. 498–510.
48. Forrester, J. W., High-speed electrostatic storage, *Proc. Symposium on Large Scale Digital Calculating Machines, Ann. Harvard Univ. Computation Lab.*, vol. 16, 1948, pp. 125–129.
49. Dodd, S. H., H. Klemperer, and P. Youtz, Electrostatic storage tube, *Elec. Engr.*, vol. 69, 1950, pp. 990–995.
50. Rajchman, J. A., The selective electrostatic storage tube, *R C A Rev.*, vol. 12, March, 1951, pp. 53–97.
51. Noyes, T., and W. E. Dickinson, The random-access memory accounting machine II: The magnetic-disk random-access memory, *IBM J. Research and Develop.*, vol. 1, no. 1, January, 1957, pp. 72–75.
52. Fuller, H. W., P. A. Husman, and R. C. Kelner, Techniques for increasing storage density of magnetic drum systems, *Proc. Eastern Joint Computer Conf., Dec. 8–10, 1954*, pp. 16–21.
53. Hong, K., Magnetic drum components for high storage density, *Trans. AIEE*, vol. 77, pt. I, November, 1958, pp. 667–672.
54. Hoagland, A. S., High-resolution magnetic recording structures, *IBM J. Research and Develop.*, vol. 2, no. 2, April, 1958, pp. 90–104.
55. Hoagland, A. S., Magnetic drum recording of digital data, *Trans. AIEE*, vol. 73, pt. I, September, 1954, pp. 381–385.
56. Hoagland, A. S., Magnetic data recording theory: Head design, *Trans. AIEE*, vol. 75, pt. I, November, 1956, pp. 506–512.
57. Williams, F. C., T. Kilburn, and G. E. Thomas, Universal high-speed digital computers: A magnetic store, *Proc. IEE London*, vol. 99, pt. II, no. 68, April, 1952, pp. 94–106.
58. Smith, J. Ernest, A new large-scale data-handling system, DATA-matic 1000, *Proc. Eastern Joint Computer Conf., Dec. 10–12, 1956*, pp. 22–28.
59. Lawrance, R. B., R. E. Wilkins, and R. A. Pendleton, Apparatus for magnetic storage on three-inch wide tapes, *Proc. Eastern Joint Computer Conf., Dec. 10–12, 1956*, pp. 84–90.
60. Thompson, B. W., and D. F. Eldrige, Achieving maximum pulse packing densities and transfer rates, *1958 IRE WESCON Conv. Record*, pt. 4, pp. 48–53.
61. Skov, R. A., Pulse time displacement in high-density magnetic tape, *IBM J. Research and Develop.*, vol. 2, no. 2, April, 1958, pp. 130–141.
62. The following group of papers appeared in *Proc. IEE London*, vol. 103, pt. B, suppl. 3, *Conv. on Digital-computer Tech., April, 1956:*
    Wilkes, M. V., Introduction to session on Magnetic tape: Input, output, and auxiliary storage, pp. 331–332.
    Kilburn, T., G. R. Hoffman, and P. Wolstenholme, Reading of magnetic records by reluctance variation, pp. 333–336.
    Wilkes, M. V., and D. W. Willis, A magnetic-tape auxiliary storage system for the EDSAC, pp. 337–345.
    Robinson, A. A., F. McAulay, A. H. Banks, and D. Hogg, A magnetic-tape digital-recording equipment, pp. 346–353.
    Discussion on foregoing group of papers, pp. 354–356.
63. Mee, C. D., Magnetic tape for data recording, *Proc. IEE London*, vol. 105, pt. B, no. 22, July, 1958, pp. 373–382.

64. Huntington, H. B., A. G. Emslie, and V. W. Hughes, Ultrasonic delay lines I, *J. Franklin Inst.*, vol. 245, no. 1, January, 1948, pp. 1–23.
Emslie, A. G., H. B. Huntington, H. Shapiro, and A. E. Benfield, Ultrasonic delay lines II, *J. Franklin Inst.*, vol. 245, no. 2, February, 1948, pp. 101–115.
65. Wilkes, M. V., and W. Renwick, An ultrasonic memory unit for the EDSAC, *Electronic Eng.*, vol. 20, July, 1948, pp. 208–213.
66. Auerbach, I. L., J. P. Eckert, R. F. Shaw, and C. B. Sheppard, Mercury delay line memory using a pulse rate of several megacycles, *Proc. IRE*, vol. 37, no. 8, August, 1949, pp. 855–861.
67. Arenberg, D. L., Ultrasonic solid delay lines, *J. Acoust. Soc. Am.*, vol. 20, January, 1948, pp. 1–26.
68. Beveridge, H. N., and W. W. Keith, Piezoelectric transducers for ultrasonic delay lines, *Proc. IRE*, vol. 40, no. 7, July, 1952, pp. 828–835.
69. Huntington, H. B., On ultrasonic propagation through mercury in tubes, *J. Acoust. Soc. Am.*, vol. 20, July, 1948, pp. 424–432.
70. Spaeth, D. A., T. F. Rogers, and S. J. Johnson, Fused-quartz ultrasonic delay line memory, *Electronics*, vol. 26, no. 12, December, 1953, pp. 151–153.
71. Pennell, E. S., Vitreous silica for ultrasonic delay line applications, *Proc. Natl. Electronics Conf., 1952*, vol. 8, pp. 799–810.
72. Newman, E. A., D. O. Clayden, and M. A. Wright, The mercury-delay-line storage system of the ACE pilot model electronic computer, *Proc. IEE London*, vol. 100, pt. II, August, 1953, pp. 445–452.
73. Epstein, H., and O. Stram, Magnetostrictive sonic delay line, *Rev. Sci. Instr.*, vol. 24, March, 1953, pp. 231–232.
74. Chaplin, G. B. B., R. E. Hayes, and A. R. Owens, A transistor digital fast multiplier with magnetostrictive storage, *Proc. IEE London*, vol. 102, pt. B, July, 1955, pp. 412–425.
75. Robbins, R. C., and R. Millership, Applications of magnetostriction delay lines, *Natl. Phys. Lab. Proc. 1953 Symposium on Automatic Digital Computation*, pp. 199–212.
76. Fairclough, J. W., A sonic delay-line storage for a digital computer, *Proc. IEE London*, vol. 103, pt. B, suppl. 3, *Conv. on Digital-computer Tech., April, 1956*, pp. 491–496.
77. Scarrott, G. G., and R. Naylor, Wire-type acoustic delay lines for digital storage, *Proc. IEE London*, vol. 103, pt. B, suppl. 3, *Conv. on Digital-computer Tech., April, 1956*, pp. 497–508.
78. Holt, A. W., An experimental rapid access memory using diodes and capacitors, *Proc. ACM Toronto meeting, Sept. 8–10, 1952*, pp. 133–141.
79. Anderson, J. R., Ferroelectric materials as storage elements for digital computers and switching systems, *Elec. Eng.*, vol. 71, October, 1952, pp. 916–922.
80. Anderson, A. G., R. L. Garwin, E. L. Hahn, J. W. Horton, G. L. Tucker, and R. M. Walker, Spin echo serial storage memory, *J. Appl. Phys.*, vol. 26, no. 11, November, 1955, pp. 1324–1338.
81. Fernbach, S., and W. G. Proctor, Spin-echo memory device, *J. Appl. Phys.*, vol. 26, no. 2, February, 1955, pp. 170–181.
82. King, G. W., G. W. Brown, and L. N. Ridenour, Photographic techniques for information storage, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1421–1428.

*Supplementary References on Magnetic Cores:*

Albers-Schoenberg, E., Ferrites for microwave circuits and digital computers, *J. Appl. Phys.*, vol. 25, no. 2, February, 1954, pp. 152–154.

Hegyi, I. J., Ferromagnetic spinels with rectangular hysteresis, loops, *J. Appl. Phys.*, vol. 25, no. 2, February, 1954, pp. 176–178.

Galt, J. K., J. Andrus, and H. G. Hopper, Motion of domain walls in ferrite crystals, *Rev. Modern Phys.*, vol. 25, January, 1953, pp. 93–97.

Van Sant, O. J., Jr., Considerations for the selection of magnetic core materials for digital computer elements, *1954 IRE Conv. Record*, pt. 4, pp. 109–115.

Haynes, M. K., Multidimensional magnetic memory selection systems, *IRE Trans. on Electronic Computers*, vol. EC-1, December, 1952, pp. 25–29.

Guterman, S., and R. D. Kodis, Magnetic core selection systems, *1954 IRE Conv. Record*, pt. 4, pp. 116–123.

Bartik, W. J., and T. H. Bonn, A small coincident-current magnetic memory, *IRE Trans. on Electronic Computers*, vol. EC-5, no. 2, June, 1956, pp. 73–78.

Fresh, D. L., Methods of preparation and crystal chemistry of ferrites, *Proc. IRE*, vol. 44, no. 10, October, 1956, pp. 1303–1311.

Merwin, R. E., The IBM 705 EDPM memory system, *IRE Trans. on Electronic Computers*, vol. EC-5, no. 4, December, 1956, pp. 219–223.

Wier, J. M., A high-speed permanent storage device, *IRE Trans. on Electronic Computers*, vol. EC-4, no. 1, March, 1955, pp. 16–20.

Robinson, A. A., V. L. Newhouse, M. J. Friedman, D. G. Bindon, and I. P. V. Carter, A digital store using a magnetic core matrix, *Proc. IEE London*, vol. 103, pt. B, suppl. 3, *Conv. on Digital-computer Tech., April, 1956*, pp. 295–301.

Scarrott, G. G., W. J. Harwood, and K. C. Johnson, The design and use of logical devices using saturable magnetic cores, *Proc. IEE London*, vol. 103, pt. B, suppl. 3, *Conv. on Digital-computer Tech., April, 1956*, pp. 302–312.

Discussion on Rapid-access storage, including the use of magnetic cores for storage and switching, *Proc. IEE London*, vol. 103, pt. B, suppl. 3, *Conv. on Digital-computer Tech., April, 1956*, pp. 327–330.

McNamara, F., The noise problem in a coincident-current core memory, *IRE Trans. on Instrumentation*, vol. I-6, no. 2, June, 1957, pp. 153–156.

McMahon, R. E., Transistorized-core memory, *IRE Trans. on Instrumentation*, vol. I-6, no. 2, June, 1957, pp. 157–160.

Bobeck, A. H., New concept in large-size memory arrays: The twistor, *J. Appl. Phys.*, vol. 29, no. 3, March, 1958, pp. 485–486.

Kaufman, M. M., and V. L. Newhouse, Operating range of a memory using two ferrite plate apertures per bit, *J. Appl. Phys.*, vol. 29, no. 3, March, 1958, pp. 487–488.

Minnick, R. C., and R. L. Ashenhurst, Multiple coincidence magnetic storage systems, *J. Appl. Phys.*, vol. 26, no. 5, May, 1955, pp. 575–579.

Blois, M. S., Jr., Preparation of thin magnetic films and their properties, *J. Appl. Phys.*, vol. 26, no. 8, August, 1955, pp. 975–980.

Thorenson, R., and W. R. Arsenault, A new nondestructive read for magnetic cores, *Proc. 1955 Western Joint Computer Conf.*, pp. 111–116.

Papoulis, A., The nondestructive read-out of magnetic cores, *Proc. IRE*, vol. 42, no. 8, August, 1954, pp. 1283–1288.

Gianola, U. F., and T. H. Crowley, The LADDIC: A magnetic device for performing logic, *Bell System Tech. J.*, vol. 38, no. 1, January, 1959, pp. 45–72.

Chen, M. C., A magnetic core parallel adder, *IRE Trans. on Electronic Computers*, vol. EC-7, no. 4, December, 1958, pp. 262–264.

*Proc. IEE*, pt. B, suppl. 7, *Conv. on Ferrites, 1957*. Contains several papers on square-loop ferrite materials and their applications in storage and in logic circuits.

*Supplementary References on Electrostatic Storage:*

Haeff, A. V., A memory tube, *Electronics*, September, 1947, pp. 80–83.

Pensak, L., The graphechon: A picture storage tube, *R C A Rev.*, vol. 10, March, 1949, pp. 59–73.

Dyall, W. T., G. R. Fadner, and M. D. Harsh, Development of an improved graphechon storage tube, *Proc. Natl. Electronics Conf.*, vol. 8, 1952, pp. 535–542.

Parker, C. V., Charge storage in cathode-ray tubes, *Proc. IRE*, vol. 39, no. 8, August, 1951, pp. 900–907.

McConnell, R. A., Video storage by secondary emission from simple mosaics, *Proc. IRE*, vol. 35, no. 11, November, 1947, pp. 1258–1264.

Harrington, J. V., Storage of small signals on a dielectric surface, *J. Appl. Phys.*, vol. 21, no. 10, October, 1958, p. 1048.

Edwards, D. B. G., The design and operation of a parallel-type cathode-ray-tube storage system, *Proc. IEE London*, vol. 103, Pt. B, Suppl. 3, *Conv. on Digital-computer Tech., April, 1956*, pp. 319–326.

*Supplementary References on Drums:*

Hendrickson, A. P., G. I. Williams, and J. L. Hill, Message storage and processing with a magnetic drum system, *Proc. Eastern Joint Computer Conf., Dec. 8–10, 1954*, pp. 74–78.

McGuigan, J. H., Combined reading and writing on a magnetic drum, *Proc. IRE*, vol. 41, no. 10, October, 1953, pp. 1438–1444.

Seader, L. D., Magnetic-recording-head selection switch, *IBM J. Research and Develop.*, vol. 2, no. 1, January, 1958, pp. 36–42.

Merry, I. W., and B. G. Maudsley, The magnetic-drum store of the computer Pegasus, *Proc. IEE London*, vol. 103, pt. B, suppl. 3, *Conv. on Digital-computer Tech., April, 1956*, pp. 197–202.

Stephen, J. H., and E. H. Cooke-Yarborough, An interleaved-digit magnetic-drum store for a transistor digital computer, *Proc. IEE London*, vol. 103, pt. B, suppl. 3, *Conv. on Digital-computer Tech., April, 1956*, pp. 382–389.

Eadie, D., EDVAC drum memory phase system of magnetic recording, *Elec. Engr.*, vol. 72, no. 7, July, 1953, pp. 590–595.

## PROBLEMS

**11.1** A device having the volt-ampere characteristic shown is used in the circuit of Fig. 11.1*b*. What is the lowest value of the resistor $R$ that will allow the circuit to have two points of stable equilibrium? Plot a volt-ampere characteristic ($I$ versus $E_{bb}$) for the device and resistor $R$ in series, for $R = 4{,}000$ ohms.

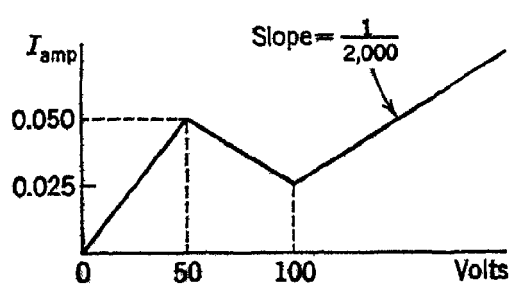



Fig. P11.1

**11.2** A coincident-current magnetic-core-memory plane having $n^2$ cores requires $2n$ driving wires. Each set of $n$ of these driving lines may in turn be excited from a coincident-current matrix of smaller size, whose lines are in turn driven by still smaller matrices. Carrying this principle to its extreme, how many cores would have to be provided in auxiliary matrices and how many wires would be driven to make a single selection for a memory plane having 65,536 cores?

Show a block diagram for the system, with 16 flip-flops driving the lowest level of the cascaded decoding matrices.

**11.3** Magnetic-core matrices using the coincident-current principle can be utilized for logic functions as well as for storage. For example, if $x$ and $y$ are two binary digits, each encoded in a 1-out-of-2 form, the following circuit functions as a half-adder. Using this same technique, show a $5 \times 5$ core array linked by five sense windings arranged to form the modulo 5 sum of two base 5 numbers. Show another array wired to form the mod 5 product. (See Appendix for discussion of the notation "mod 5".)

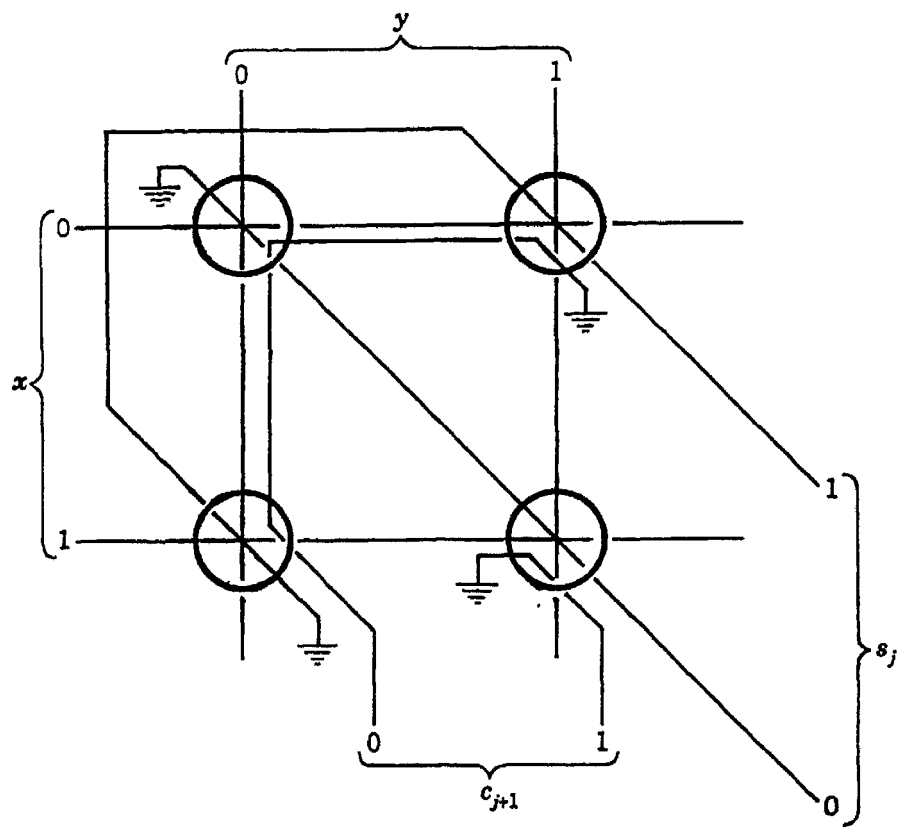



Fig. P11.3

**11.4** Using the circuit principle exhibited in Fig. 11.24, show a magnetic-core system that converts from binary to decimal. The binary information may be assumed to be in the 8,4,2,1 code, each bit being stored in a flip-flop. The "decimal" output should be in a 1-out-of-10 binary code.

**11.5** For the circuit of Fig. 11.25, tabulate the total excitation applied to each of the four cores for each of the four combinations of flip-flop states.

**11.6** A certain ferrite core material is found to have a perfectly rectangular $B$-$H$ loop when measurements are made upon samples of minute dimensions. The saturation flux density $B_s$ is 2,000 gauss, and the coercive force $H_c$ is 2.0 oersteds. When this material is made into a core of 0.080 in. OD and 0.050 in. ID, what is the incre-

mental permeability of the resulting core? If the core switches from $-B_s$ to $+B_s$ in 2 $\mu$sec, what is the average induced voltage per turn?

**11.7** Draw a timing diagram for a two-core-per-bit magnetic shift register. Plot the following as functions of time, one below the other, for an input pulse sequence of 1,1,0,1:

Input current
*A* shift current
*B* shift current
Flux in core 1 of first stage
Flux in core 2 of first stage
Flux in core 1 of second stage
Flux in core 2 of second stage

**11.8** Why should the shift register of Fig. 11.31*e* not be connected as shown below?

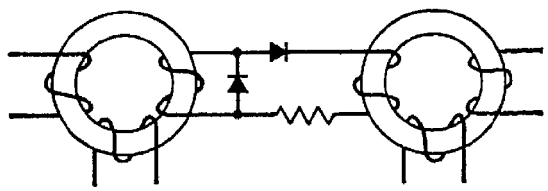


Fig. P11.8

# Appendix

## Congruences

An important branch of number theory is the congruence relationship. Computer engineers find the concepts and terminology of congruences particularly useful in describing operations performed upon numbers in the digital computer.

Two integers $a$ and $b$ are said to be "congruent modulo $m$" if $a - b$ is exactly divisible by $m$, where $m$, which is called the "modulus", is also an integer. This relationship is usually written

$$a \equiv b \bmod m \tag{A.1}$$

For example,

$$5 \equiv 2 \bmod 3$$

and

$$10 \equiv 6 \bmod 2 \tag{A.2}$$

Congruence modulo $m$ is a relationship that divides the integers into $m$ "residue classes". Thus congruence mod 3 divides the integers into classes as follows, where any member of one residue class is congruent mod 3 to any other member of the same class:

$$\begin{array}{l} \ldots, -9, -6, -3, 0, 3, 6, 9, 12, \ldots \\ \ldots, -8, -5, -2, 1, 4, 7, 10, 13, \ldots \\ \ldots, -7, -4, -1, 2, 5, 8, 11, 14, \ldots \end{array} \tag{A.3}$$

The $m$ least positive residues (in this case 0, 1, and 2) are often used as representatives of the $m$ residue classes, since any integer is congruent mod $m$ to one of these.

Any digital device represents numbers in residue form, since the device has only a finite number of digit positions. For example, an automobile odometer usually has five digit dials (in the integer part). Thus $x$, the number of miles traveled, and $x_i$, the number of miles indicated, are related by

$$x \equiv x_i \bmod 10^5$$

where $x_i$ is the least positive residue of $x$ mod $10^5$. Similarly, a binary digital computer with $n$ bits in a number word represents integers by

their least positive residue mod $2^n$. (The fractional form in which most computers manipulate numbers can be regarded as the result of a scale change performed after reduction of the number mod $2^n$.) Consequently, some of the manipulative rules for congruences are of particular interest to the computer engineer.

An important rule is that congruences with respect to the same modulus may be added. Thus, if

$$a \equiv b \bmod m \tag{A.4}$$

and

$$c \equiv d \bmod m \tag{A.5}$$

it is also true that

$$a + c \equiv b + d \bmod m \tag{A.6}$$

This can be easily demonstrated, since the congruence of $a$ and $b$ implies that

$$a = b + k_1 m \tag{A.7}$$

where $k_1$ is an integer. Likewise,

$$c = d + k_2 m \tag{A.8}$$

Since equalities are additive, we may write

$$a + c = b + d + (k_1 + k_2)m \tag{A.9}$$

But $k_1 + k_2$ is an integer; therefore,

$$a + c \equiv b + d \bmod m \tag{A.10}$$

Similar arguments show that congruences may be subtracted or multiplied. Congruences may not be divided, however, since congruence is defined as a relation among integers.

Let us define "addition modulo $m$" to be conventional addition of two integers $a$ and $b$ followed by reduction of the sum to the least positive residue. Thus the sum of 6 and 8 mod 10 is 4. The ring sum symbol $\oplus$ is often used to denote this operation. Thus we would write

$$6 \oplus 8 = 4 \tag{A.11}$$

The modulus for this operation is often implicit in the context, but it should be stated explicitly if there is any possibility of confusion.

If

$$X = x_n x_{n-1} \cdots x_2 x_1 x_0 \tag{A.12}$$

and

$$Y = y_n y_{n-1} \cdots y_2 y_1 y_0 \tag{A.13}$$

where $x_j$ and $y_j$ are the individual digits, we may write their sum $S$ as follows:

$$S = s_n s_{n-1} \cdots s_2 s_1 s_0 \tag{A.14}$$

Each of the sum digits $s_j$ is given by

$$s_j = x_j \oplus y_j \oplus c_j \tag{A.15}$$

where $c_j$ is the carry digit into the $j$th position, and the modulus is the base of the number system in which $X$ and $Y$ are represented.

Parity digits can be readily defined using congruence notation. If $p_x$ is the digit (either 1 or 0) required to make the total number of 1's even in the group that consists of the binary number $X$ and its parity digit $p_x$, we may write

$$p_x \oplus x_n \oplus x_{n-1} \oplus \cdots \oplus x_2 \oplus x_1 \oplus x_0 = 0 \tag{A.16}$$

or

$$p_x = x_n \oplus x_{n-1} \oplus \cdots \oplus x_2 \oplus x_1 \oplus x_0 \tag{A.17}$$

The reader can easily verify that in general $p_s$, the parity digit for the sum $S$ of two numbers $X$ and $Y$, is not equal to the sum of $p_x$ and $p_y$. However, it is easy to find a relation between $p_s$ and $p_x \oplus p_y$. First we note that

$$p_s = s_n \oplus s_{n-1} \oplus \cdots \oplus s_2 \oplus s_1 \oplus s_0 \tag{A.18}$$

but

$$s_j = x_j \oplus y_j \oplus c_j \tag{A.19}$$

Therefore,

$$p_s = (x_n \oplus y_n \oplus c_n) \oplus (x_{n-1} \oplus y_{n-1} \oplus c_{n-1}) \oplus \cdots \oplus (x_0 \oplus y_0) \tag{A.20}$$

Since the $\oplus$ operator is commutative, we may write

$$p_s = (x_n \oplus x_{n-1} \oplus \cdots \oplus x_0) \oplus (y_n \oplus y_{n-1} \oplus \cdots \oplus y_0) \oplus c_n \oplus c_{n-1} \oplus \cdots \oplus c_1 \tag{A.21}$$

which is

$$p_s = p_x \oplus p_y \oplus c_n \oplus c_{n-1} \oplus \cdots \oplus c_1 \tag{A.22}$$

That is, the parity digit $p_s$ should be equal to the sum mod $b$ of $p_x$, $p_y$, and any carry digits produced in the addition. [$b$ is the base of the number system, and Eq. (A.17) may be regarded as defining the parity digit for any base and not merely for binary numbers.]

The importance of Eq. (A.22) is that it makes possible a parity check on the arithmetical operations of a computer as well as on transfers of data from one part of the computer to another. Since all arithmetical operations within the computer are performed by successions of additions, it is possible to use parity digits to check the arithmetic completely. This result is due to H. L. Garner.

## REFERENCES

Birkhoff, G., and S. MacLane, "A Survey of Modern Algebra," The Macmillan Company, New York, 1948, pp. 23–29.

Vinogradov, I. M., "Elements of Number Theory," Dover Publications, New York, 1954, chaps. 3, 4, and 5.

Garner, H. L., Generalized parity checking, *IRE Trans. on Electronic Computers*, vol. EC-7, no. 3, September, 1958, pp. 207–213.

Garner, H. L., The residue number system, *IRE Trans. on Electronic Computers*, vol. EC-8, no. 2, June, 1959, pp. 140–147.

## PROBLEMS

**A.1** A binary number $X$ has associated with it a parity check digit $p_x$. State whether $p_x$ should be altered to maintain the parity check under each of the following circumstances:

*a.* $X$ has an even number of digits to the left of its least-significant 1 and is radix-complemented.

*b.* $X$ has an odd number of digits to the left of its least-significant 1 and is radix-complemented.

*c.* $X$ has an even number of digits and is diminished-radix complemented.

*d.* $X$ has an odd number of digits and is diminished-radix complemented.

*e.* $X$ is shifted two places to the left.

*f.* $X$ is added to itself.

*g.* $X$ is divided by 4.

In *e*, *f*, and *g*, the various possible patterns of overflow digits should be considered.

How will the answers to the preceding questions be affected if the parity bit is chosen so as to make the total number of 1's odd instead of even?

**A.2** Form $p_x$ and $p_y$ for each of the following pairs of numbers. Add the numbers and form $p_s$. Check your addition by keeping a count of the number of carries and using Eq. (A.22).

(*a*) $\left.\begin{array}{l} X = 3{,}489{,}172 \\ Y = 1{,}924{,}387 \end{array}\right\}$ decimal numbers

(*b*) $\left.\begin{array}{l} X = 0174226 \\ Y = 3421770 \end{array}\right\}$ octal numbers

(*c*) $\left.\begin{array}{l} X = 0101101101 \\ Y = 0011110011 \end{array}\right\}$ binary numbers

**A.3** For the binary case the parity digit $p_x$ is sometimes taken to be the number $p_x$ satisfying the relation

$$p_x \oplus x_n \oplus x_{n-1} \oplus \cdots \oplus x_1 \oplus x_0 = 1$$

Show whether Eq. (A.22) still holds under these circumstances.

**A.4** The residue code is one in which decimal integers are represented by their least positive residues with respect to a set of relatively prime moduli. For example, the number 123 may be encoded as the group 4,3,0,1 where 4,3,0, and 1 are respectively the residues mod 7,5,3, and 2.

*a.* Encode the numbers −33 and +89 in this code.

*b.* Find the code for their sum (1) by adding the uncoded numbers and encoding the sum, and (2) by adding the coded representations employing addition modulo $m$. Note that, although this code possesses the superficial advantage of being free from carry generation in addition, it is difficult to determine the sign of an encoded number. Comparison of the magnitudes of two encoded numbers is also difficult, and division can be accomplished only with the aid of additional rules and manipulations.

# Index


Abacus, 3
Absolute address, 192
Absorption, dielectric, 122, 143, 159
Accumulator, as adder, 329–330
  addressable, 202
  MIC, 377–381
  in single-address machines, 189
Adder, accumulator type, 329–330
  analog, frequency characteristic, 120–125
    step-function response, 124–125
  digital, 317–330
  half, 320–325
  MIC, 377–381
  modulo 2, 323–324
  relay, 319
  sum and carry digits, 318
  three-input, 320–322
  two-input, 320–325
Addition, analog, circuit for, 6–8
  symbol, 17
Address, definition of, 170
  relative, 192–193
Alternative box, 188
Ambiguous-word machines, 180
Amplifier, analog computer, auxiliary, 134–139, 157–159
    error in, 120–133
  drift, 122, 140–145
  fall-off and phase shift, 122
  grid current, 122, 140–143
  saturation, 17
Analog computer defined, 3–4
Analyzer routine, 211
AND element, definition of, 269
AND gate (*see* Gates)
Angelo, E. J., Jr., 106
Angelo multiplier tube, 106
Arithmetic checks, 243–245
Arithmetic unit, 169
Associativity, of $\cap$ and $\cup$, 249
  lack of, for $|$ and $\downarrow$, 432–433
  of $+$ and $\cdot$, 254
Asymptotes, method of, 155–157
Asynchronous systems, 307–308, 317, 320
Automata, theory of, 305
Automatic programming, 205–212
Automonitor, 211

*B* box (*B* register), 193
Babbage, Charles, 3
Backlash (*see* Diode switch)
Base counter, 192–195
Base instruction, 193
Base number system (*see* Radix)
Baum, R. V., 111
BESM, 471
Binary-coded decimal digit, 168, 223–224, 230–239
Binary counter, 344–395
Binary numbers, conversion of, to decimal numbers, 247
  systems of notation, 166
  (*See also* Binary-coded decimal digit; Complementation)
Birkhoff, G., 250
Block addressing, 174
Boole, George, 254
Boolean algebra, 254–265
Booth, A. D., 356
Borrow, end-around, 336
Borrow function, 330
Breakpoint, in digital programs, 211
  in diode line segment generator, 76–79
Bridge circuit, 281–282, 297–298, 300, 304

Bruns, R. A., 78–79
Buckerfield, P. S. T., 139
Buffer, 388
Burkhart, W. H., 287
Burks, A. W., 272–273, 311, 326
Bush, Vannevar, 4
Bush differential analyzer, 4
Bush integrator, 4–5

Calculating precision, 204
Canonical forms, 263
Cap symbol, $\cap$, 248
Carroll, Lewis, 250
Carry, end-around, 229, 334, 340
  propagation of, 326
  sensing of completion of, 327–329
Carter, W. C., 204
Cathode emission, effect of changes in, reduction of, 143–145
Characteristic equation, degree of, 29–32
  root dislocation, 27–29, 131–133
  spurious roots, 29, 131–133
Chopper stabilizer (auxiliary amplifier), 134–139, 157–159
Circuits, bridge, 281–282, 297–298, 300, 304
  combinational, 278–305
  sequential (*see* Sequential circuits)
Clause, 288
Clock phases, 402–403
Clock pulses, 307, 322, 401
Code, binary, for decimal digits, 168, 223–239
  excess-3, 234–235
  four-address, 170, 174
  Hamming error-correcting, 242–243
  MIC keyboard, 382–384
  operation, 170
  reflected, 237–239
  unit-distance, 315
Coercivity, 453–458, 460–461
Coincident-current selection, 446–448
Combinational circuits, 278–305
  (*See also* Switching networks)
Commutativity, of $\cap$ and $\cup$, 249
  of $\downarrow$ and $|$, 432
  of $+$ and $\cdot$, 253
Comparator routine, 212
Comparison instruction, 174, 176
  (*See also* Jump instruction)
Compiler, 206–207
Complementation, 227–230, 334–341, 353–357
  MIC, 377–382
Complementing codes, 233–235
Complete product term, 292
Computing elements, analog, linear, circuits of, 13
  generalized symbols for, 17
Congruence, 510–513
Conjunction, definition of, 251–252
  symbol, 253
Conjunctive normal form, 263, 284
Consensus, 288
Constraints, inclusion in analog setups, 49–51
Continuous computer, 4, 164
Control digits, 193–195
Control unit, 169
Conversion routine, 206
Core switching function, 289–290, 296–297
Cores (*see* Magnetic cores)
Corrector routine, 212
Coulomb friction (*see* Diode switch)
Counter, base, 192–195
  binary, 344–395
  instruction, 180–181
  with static circuitry, 412
Crossed-fields electron-beam multiplier, 103
Cunningham, W. J., 113
Cup symbol, $\cup$, 248
Curve followers, 91–94

Dagger function (*see* Dual stroke)
Dead space (*see* Diode switch)
Decimal-to-binary conversion, 167–168, 205
Deering, C. S., 157
Delay, analog simulation of, 112–116
  in synchronous systems, 307, 322, 392
Delay lines, characteristics of, 403–407
De Morgan's theorems, 254, 284
Diagnostic routine, 210
Dielectric absorption, 122, 143, 159
Differences, tables of, 67–69
Differential equations, linear, computing elements used for, 13
  nonlinear, use of multiplier, 40–45

Differential equations, partial, setup on analog computer, 52–54
Differentiator, approximate, 128–129
  transfer ratio, 127
  use of, 46, 163
Digital computer, block diagram, 169
  definition of, 2–3
Digitized functions, 165–169
Digits, control, 193–195
Diode switch, backlash simulator, 80–82
  bias controlled by second variable, 98–100
  bias supplies for, 75–77
  break-point adjustment, 77–79
  Coulomb-friction simulator, 80
  dead-space simulator, 80
  effect of resistance, 72–73
  four-quadrant operation, 74–75
  in parabola generator, 73
  in semicircle generator, 76
  series and shunt limiting, 70–73
Direct-coupled transistor logic, 391
Directory, 211
Discontinuous computer, 164
Disjunction, definition of, 251–252
  symbol, 253
Disjunctive normal form, 263, 280, 284
Distributivity, of $\cap$ and $\cup$, 249
  lack of, for $\downarrow$ and $|$, 433
  of $+$ and $\cdot$, 254
Division, analog, 45
  digital, 357–362
  MIC, 381–382
"Don't care" conditions, 282–283, 286
Double-precision multiplication, 191
Dow, P. C., 129, 133
Drift voltages, reduction of, 140–145
  sources of, 140
  stabilization against, 133–139, 157–160
Drum, magnetic, description of, 170–172
    layout, MIC, 373
    typical specifications, 495
    (*See also* Magnetic recording)
Dual networks, definition of, 259
  topological method of obtaining, 259–260
Dual stroke, conversion from normal form, 435
  definition of, 252, 432
  extension of notation, 433
  rules for manipulating, 432
Dual stroke, transistor circuit, 391–392, 423–427
  use in logical design, 431–438
  vacuum-tube implementation, 389–390
Dynamic circuitry, design considerations in, 393–407
Dynamic flip-flop, 341–343, 349, 368–382, 403

Ebers-Moll equations, 421–423
Eddington, Sir Arthur S., 279
Edwards, C. M., 102
Electrostatic storage, 483–494
  by potential of target elements, 491–494
  by redistribution of charge, 487–489
Element-input count, 277–278
Ellis, M., 204
ENIAC, 224
Error, in analog-computer solutions, 129–133
  in computing amplifiers, 120–133
  detection and correction of, 243
  equal-ripple, 61, 66
  round-off, 62, 88
  truncation, 62, 88
Excess-3 code, 234–235
Exclusive OR, 323
Expansion theorems, Boolean, 264–265
Exponential switch, 275–278
External program, 206
External selection, core memory, 468–471

Ferrites, compared with metals, 453–454, 460–462
  manufacture of, 459
  properties of, 453–459
File instruction, 195
Fixed-point numbers, 182
Flip-flop, dynamic, 341–343, 349, 368–382, 403
  made from NOR circuits, 431
  memory element in sequential circuit, 308
  in static circuitry, 341, 407–410
  switching, 417, 418
  as temporary storage, 392, 442
  types of, 341–342, 410–413

Flip-flop, vacuum-tube circuit, d-c design, 413–417
Floating addresses, 208
Floating-point numbers, 183–184
Flow diagrams, 188–189
Forcing functions, generation on analog computer, 34–35
Forrester, J. W., 492
Four-address code, definition of, 170
  operations with, 174
Four-quadrant multipliers, 101
Function generators, nonlinear, classification of, 59
  diode line segment (*see* Diode switch)
Function maps, 283–286, 315–316
Functional approximation, by interpolating functions, 62
  need for, 60
  other types, 69
  types of fit, 61
Fundamental formula, 288

Garner, H. L., 244
Gates, AND, 389–401
  diode, 390–391
  transistor, 391–392
  vacuum-tube, 388–390
Gilchrist, B., 327
Gill, S., 195
Godet, S., 36
Goldberg, E. A., 108, 135
Goldstine, H., 188, 326
Gray code (*see* Reflected code)
Grid current, 122, 140–143
  integration of, 142
  sources of, 141
Gutenmacher, L. I., 471

Halt instruction, 176–177
Hamer, H., 142
Hamming, R. W., 242
Hamming error-correcting code, 242–243
Harris, B., 293
Hazard, 308
High-order multiplication, 192
Hindrance function, 255–259
Hohn, F. E., 304
Howe, R. M., 54, 157
Huffman, D. A., 308, 311

Hysteresis, for memory, 444
  simulation of, 80–82
Hysteresis loops, family of, 454
  losses due to, 455
  for metals and ferrites, 453
  minor, 450–452
  square, 445
  variation of, with core thickness, 472
    with temperature, 455–457

ILLIAC, 484
Implicant, prime, 288–297
Implication, material, 252
Index register, 193
Inequivalence, definition of, 252
Infinite-impedance probe, 97–98
Inhibit-core logic, 478
Inhibit current, magnetic-core plane, 448–449
Inhibition of gate, circuit for, 400–401
  symbol, 269–270
Initial charge, 14, 54
Input-output, digital, 169, 382–384
Insertion of omitted instructions, 179
Instruction, comparison, 174, 176
  file, 195
  halt, 176–177
  jump (*see* Jump instruction)
  omitted, insertion of, 179
  three-address, 180, 202
  two-address, 189–191, 204
Instruction counter, 180–181
Instruction modification, 177–180
Instruction structure, 203–204
Integration, definite, 47–49
  generalized, 46–47, 128
Integrator, circuit, 8–10
  frequency characteristic, 122, 125–127
  minimum number of, 26–34
  step-function response, 126–127
  symbol, 17
Internal program, 206
Internal state, 308–311
Interpolating functions, linear approximations, 62–65
  Newton's formula, 67–69
  polynomials, 65–69
Interpretive programs, 206
Intersection, 248
Interstage coupling, 155, 423–424

Invalid combinations (*see* "Don't care" conditions)
Inverse networks, 260
Inverted logic, 432
Iterative networks, function generation using, 82–83
  switching circuits, 267–268

Jump instruction, 176, 177, 180
  on equality, 192
  forced, 181
  on zero, 192

Karnaugh, M., 283–286, 315–316
Kazan, B., 486–487
Keister, W., 311
Kelvin, Lord, 5–6
Kilburn, T., 490
Knoll, M., 486–487
Korn, G. A., 87

Laplace's equation used in function generators, 95–97
Lead network, 153–160
Literal, definition of, 288
Lo, A. W., 481
Logarithmic multiplier, 106
Logic, of classes, 247–251
  by means of magnetic cores, 475–480, 482–483
  of propositions, 251–255
Low-order multiplication, 192

*M* network, 297–300
McCluskey, E. J., 293–297
MacLane, S., 250
McMahon, R. E., 472
Macnee, A. B., 48, 129, 133
Magnetic-core memory, 445–462, 468–473, 482
Magnetic cores, 445–484
  logic by means of, 475–480, 482–483
  matrix switches using, 463–468
  multidimensional selection of, 462–463
  nondestructive read-out of, 473, 482
  partial switching of, 472
  shift registers using, 474–478
Magnetic cores, switching time, 457–458, 460–461
  thin films for, 472–473
  (*See also* Coincident-current selection; Ferrites; Hysteresis loops; Memory; Memory properties)
Magnetic recording, 494–501
  on drums, 494–495
  on tape, 500–501
  (*See also* Drum)
Marsocci, V. A., 129, 133
Material implication, 252
Matrix, connection, 301
  core memory, 446–449
  expansion of, 302–304
  methods of use, 300–305
  output, 300
  primitive connection, 300
  switching, 463–471
Mayer, R. P., 466
Mealy, G. H., 309, 311
Meissinger, H. F., 95
Memory, acoustic delay line, 501
  diode-capacitor, 501
  electrostatic, 483–494
  ferroelectric capacitor, 501
  magnetic-core, 445–462, 468–473, 482
  magnetic recording, 494–501
  magnetostrictive delay line, 501
  photographic, 501
  spin-echo, 501
Memory properties, 442–443
MIC, accumulator, 377–381
  characteristics, 366
  command list, 197–199
  description of logic, 366–385
  drum layout, 373
  instruction register, 370
  keyboard code, 382–384
  memory accessing, 372–375
  operation decoding, 371–372
  organization, 377
  phase control, 369
  phase-3 timing, 375–377
  square-root program, 200–202
  word structure, 366
Microprogramming, 352
MIDAC, 209
Miller, S. E., 143
Minimization, integrators, 26–34
  switching circuits, 281, 284–297

Minimum-access programming, 190–191
Mirror symbol, 466
Modulus, 510–513
Moore, E. F., 311
Morrill, C. D., 111
Mueller, R. K., 293
Muller, D. E., 293
Multiaperture device (transfluxor), 480–483
Multiplication (digital), 348–353
  of complements, 353–357
  double-precision, 191
  high-order, 192
  low-order, 192
  rounded, 192
Multiplicative switch, 273–278
Multipliers, analog, 101–112
Multiterminal switching networks, 265–266

*N* network, 297–300
NAND (*see* Sheffer stroke)
Negation, logical, definition of, 251
Negative numbers, radix complement, 227–230
  diminished, 227–230
Nelson, R. J., 291
Newhouse, V. L., 472
Newton's formula, for functional approximation, 67–69
  for square root, 174
NOR function, 432
  conversion from normal form, 435
  extension of notation, 433
  rules for manipulating, 432
  transistor circuit, 391–392, 423–427
  use in logical design, 431–438
  vacuum-tube implementation, 389–390
Normal formula, 288
NOT circuit, 390–392, 401
Notation, positional, 219
NRZ recording (*see* Magnetic recording)
Null class, 250
Numerical symbols, 220
Nyquist criterion, 149–153, 161–162

Olsen, K. H., 471
One's complement (*see* Complementation)
Operation box, 188
Operation code, 170
OR circuit, 388–389, 398–401
OR element, definition of, 269
Overdrive, magnetic core, 457, 464–465
Overflow, 185
Overload indicator, 159

*p* square, 284–286
Parallel operation, 317, 319, 321
Parity checks, 240–245, 500, 512
Phase-modulation recording (*see* Magnetic recording)
Phase shifts, effect of, 18
Photoelectric sensing, 92
Photoformers, 93–94
Photographic memory, 501
Pohm, A. V., 473
Poles (*see* Transfer functions, of linear networks)
Polynomial generator, 68–69
  using servo multiplier, 43
Pomerene, J. H., 327
Positional notation, 219
Positive logic, 432
Post-mortem, 211
Potentiometers, follow-up, 41–43
  loading of, 25–26, 87
  multiplying, 40–45
  for scale-factor adjustment, 21
  sources of error in, 89–90
  tapered, 88–90
  used in function generation, 82–88
Precision, calculating, 204
Prime implicant, 288–297
Probe, infinite-impedance, 97–98
Product term, 292
Programming, automatic, 205–212
  minimum-access, 190–191
Pseudo orders, pseudo code, 207
Pulse distributor, 345–348

Quantized functions, 164–165
Quarter-squares multiplier, 104–105
Quine, W. V., 280, 286, 288, 291–292, 294–295

Race condition, 308
Radix, conversion of, 220–223

Radix, definition of, 165
in early cultures, 217–218
negative, 235–237
optimum choice of, 224–227
point, 219
Radix complement, 227–229
Rajchman, J. A., 481, 494
Read-around ratio, 490
Reading head, 171
Redundancy, aid in factorization, 286
in digital codes, 239–245
Reflected code, 237–239
Regenerative broadening, 401–403
Relative address, 192–193
Residue class, 510
Residue number system, 244–245
Resistive sheets, function generation using, 96–98
Ring sum, in adder, 323–324
addition mod $m$, 511
definition of, 252–253
Ritchie, A. E., 311
Ritchie, D. K., 413
Roth, J. P., 293
Round-off, 364–366
in MIC multiplication, 381
Round-off errors, 62, 88
Rounded multiplication, 192
Rubens, S. M., 473
Rubinoff, M., 413
Russell, B., 247
RZ recording (*see* Magnetic recording)

Scale factors, in digital computers, 184–187
magnitude, 19
time, 22
SEAC, 399–400
Secondary emission, 484–487
Selectron, 494
Sense winding, 446–453
Sequential circuits, 305–311
block diagram, generalized, 306
state diagram, 310–311
Serial operation, 317, 319, 321
Servo multiplier, 40–45
used to form polynomial, 43
Setup procedure, linear differential equations, 12–15
minimization of amplifiers, 15
Shannon, C. E., 239, 272, 297–300
Sheffer stroke, conversion from normal form, 435
definition of, 252, 432
extension of notation, 433
rules for manipulating, 432
transistor circuit, 391–392, 423–427
used in logical design, 431–438
vacuum-tube implementation, 389–390
Shift, executed in accumulator, 377
register, 342–344, 367
in scaling, 184–185
Single-address instructions, 189–191, 197–204
comparison with three-address, 202
Spin-echo memory, 501
Spurious roots, introduction of, 18
Square root, analog circuit for, 44–45
built-in, digital, 362–364
program, four-address, 175–179
with indexing, 193–195
with scaling, 185–188
single-address MIC, 200–202
three-address, 180–182
by successive approximations, 174
Squarers, 105–106
Stability, Nyquist, 145–162
conditional, 151–152
Stabilization, drift-voltage, 133–139
high-frequency, 145–162
Starting, digital computer, 176
Static circuitry, counter with, 412
flip-flop in, 341, 407–410
Storage (*see* Memory; Memory properties)
Storage cell, 170
Storage hierarchy, 170, 442
Storage unit, 169
Stroke function (*see* Sheffer stroke)
Subcube, 284–286
Subroutines, 191, 195–197
Subsuming clause, 288
Subtractor, digital, difference and borrow functions of, 330–331
half, 331–334
three-input, 330–331
Switching functions, canonical forms of, 263
of $n$ variables, 279
nonnormal forms of, 286–288
simplification of, 262–263, 280–297

Switching functions, symmetry types, 279–280
of two variables, 256
Switching networks, active, 268–269, 300, 307, 309, 320
binary decoding, 272
bridge, 281–282, 297–298, 300, 304
combinational, analysis and synthesis of, 278–305
combinational and sequential, definition of, 266–277
complete decoding, 273–278
iterative, 267–268
multiterminal, 265–266
path-closing, 268, 300, 307, 320
of relays, 270–271
sequential, analysis and synthesis of, 305–311
tree, 272–278
Switching time, magnetic core, 457–458, 460–461
Symbolic addresses, 208
Symmetry types, 255, 279–280
Synchronous systems, 307–308, 317, 320

Table of combinations, 280
Thomson, James, 5
Three-address instructions, 180
comparison with single-address, 202
Time, fall, in AND circuit, 396
rise, in AND circuit, 395, 398
Time constant, adder, 124
integrator, high-frequency, 125
low-frequency, 125
Time delay, analog simulation of, 112–116
Time-division multiplier, 108–112
Timing-pulse generator, 345–348
Topological method, of network dualization, 259–260
of network simplification, 292–294
Track, 171
Track layout on drum, 172
TRADIC, 419
Transfer functions, generation on analog computer, 35–40
of linear networks, 145–148
Transfer impedances, short-circuit, table of, 38–39
Transfluxor, 480–483
Transistors, decay time of, 429–431
Transistors, storage time in, 429–430
as switches, 419–423
turn-on and turn-off, 427–431
types of logic circuitry, 418
Transmission function, 255–259
Tree, 272–278, 298
Truncation errors, 62, 88
Truth tables, 251–252
Truth value, 251
Two-address instructions, 189–191, 204
Two-level logic, 287
Two's complement (*see* Complementation)

Union, 248
Unit-distance code, 315
(*See also* Reflected code)
Universal class, 248
Urbano, R. H., 293

Veitch, E. W., 283
Venn diagram, 249
Voltage (*see* Drift voltages)
von Neumann, J., 188, 326

Wallman, H., 49
Wang, H., 311
Washburn, S. H., 311
Wattmeter multiplier, 102–103
Weierstrass theorems, 65–66
Weighted codes, 231
Wheeler, D. J., 195
Whirlwind, 203, 493
Whitehead, A. N., 247
Wilkes, M. V., 195, 208
Williams, F. C., 488, 490
Wong, S. Y., 327
Word, definition of, 174
length of, factors in, 203–204
structure of, in MIC, 366
Word-organized memory, 469
Writing head, 170

Zero, numerical symbol for, 218
plus or minus, 335
Zeros (*see* Transfer functions, of linear networks)