

ESHBACH'S HANDBOOK OF ENGINEERING FUNDAMENTALS

FOURTH EDITION

Editor

BYRON D. TAPLEY

Department of Aerospace Engineering
and Engineering Mechanics
The University of Texas at Austin
Austin, Texas

Managing Editor

THURMAN R. POSTON

Postscript, Consultants to Publishers
Temenos
Snohomish, Washington

11
726



A WILEY-INTERSCIENCE PUBLICATION

John Wiley & Sons, Inc.

NEW YORK / CHICHESTER / BRISBANE / TORONTO / SINGAPORE

R4-3-90

REF
TA151
.E8
1989

Copyright © 1990 by John Wiley & Sons, Inc.

All rights reserved. Published simultaneously in Canada.

Reproduction or translation of any part of this work beyond that permitted by Section 107 or 108 of the 1976 United States Copyright Act without the permission of the copyright owner is unlawful. Requests for permission or further information should be addressed to the Permissions Department, John Wiley & Sons, Inc.

Library of Congress Cataloging in Publication Data:

Eshbach, Ovid W. (Ovid Wallace), 1893–1958.

[Handbook of engineering fundamentals]

Eshbach's handbook of engineering fundamentals.—4th ed./
editor, Byron D. Tapley; managing editor, Thurman R. Poston.

p. cm.

“A Wiley-Interscience publication.”

Bibliography: p.

1. Engineering—Handbooks, manuals, etc. I. Tapley, Byron D.
II. Poston, Thurman R. III. Title. IV. Title: Handbook of
engineering fundamentals.

TA151.E8 1989

620—dc19

88-20581

CIP

ISBN 0-471-89084-7

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Chapter Overview

The first sections, on architecture (Sections 9.1 and 9.2), present a complete discussion of computer architecture—the functional description of a computer's operation—from the point of view of the programmer. A single architecture can be implemented in many ways, using different technologies. The IBM System/370 is an example of a single architecture that has been implemented in many embodiments. These appear virtually identical to the programmer but have different costs and speeds. The sections discuss how registers and main memory can be addressed, how the instructions manipulate the information in the registers and main memory, and how the computer's "interrupt" system permits it to interact with external signals that signify the occurrence of events to which the computer must respond.

Section 9.3 discusses the preparation of sequences of CPU instructions to carry out specified functions. This article on computer programming is included in this section on computer design because it is important that the hardware designer understand how the computer instructions are used. Section 9.4 introduces the logic of switching devices—those building blocks from which all computer components are built. Techniques are given for realizing and minimizing logical functions. Section 9.5 describes the design of arithmetic-logic units and how they perform complex operations (e.g., floating-point multiplication) by executing series of simple operations such as addition, under the management of the control unit. Sections 9.6–9.8 present the electronic technologies and physical structures used to embody main memory, describe how main memory and attached storage devices are organized and how they can present the user with a "virtual memory" that is larger than the available main memory, and explore the concerns specific to microprocessor systems. A microprocessor is a CPU on a silicon chip. Some require more than one chip, and some include memory and I/O control on the chip. Section 9.9 contains an extensive analysis of control units. A frequently used control unit design uses very regular structures called "microprograms." (These must not be confused with programs for microprocessors, which are almost identical to programs for larger CPUs.)

Sections 9.10–9.12 include a general systems description of I/O: the interface among devices, controller, main memory, and CPU, and the interchange of control signals that takes place during an I/O operation. The technology of attached storage devices and techniques used for I/O in microprocessor systems are also analyzed. Section 9.13 describes buses, cables carrying data, and control signals between (and within) components of a computer. Alternative bus strategies are analyzed in this section.

A glossary follows.

9.1 ARCHITECTURE: REGISTERS, ADDRESSING, AND INSTRUCTIONS

James E. Thornton

9.1.1 Background

A computer architecture must be defined independently of any specific implementation since there are often several different implementations over a period of time. There are numerous ways to view an architecture, ranging from user application programs written in a high-level language to specific instruction sets, including those privileged ones used by the operating system. We choose the latter for this description. In order to provide enough understanding of the architecture, reference will be made to some implementation detail and occasionally to some alternative implementations.

During execution, a program views the architecture as a set of **registers**, a main memory with various **addressing modes**, and a processing unit **instruction set** (see the introduction to this chapter for a summary of computer organization). For very simple machines, there may be no privileged instructions or addressing modes. The machine executes programs for which all the resources of the machine are available (i.e., the entire main memory). For machines that are shared by many programs operating one at a time (multiprogramming), the architecture needs to provide privileged facilities for allocating and managing resources.

9.1.2 Registers and Local Memory

One distinguishing feature of a computer architecture is the register structure of the central processing unit. A register is a group of flip-flops or latches operating together to temporarily store data or control information.

All operations in a computer system result in modifications of bits in registers or main memory. An instruction may cause one or more operations to take place, generally in a specific sequence. At the completion of each instruction, the information stored in registers and main memory is available for subsequent instructions or other access.

Some processors make use of a single register for temporarily storing data. This register is typically called the **accumulator** (see Fig. 9.2). It operates in conjunction with the arithmetic-logic unit

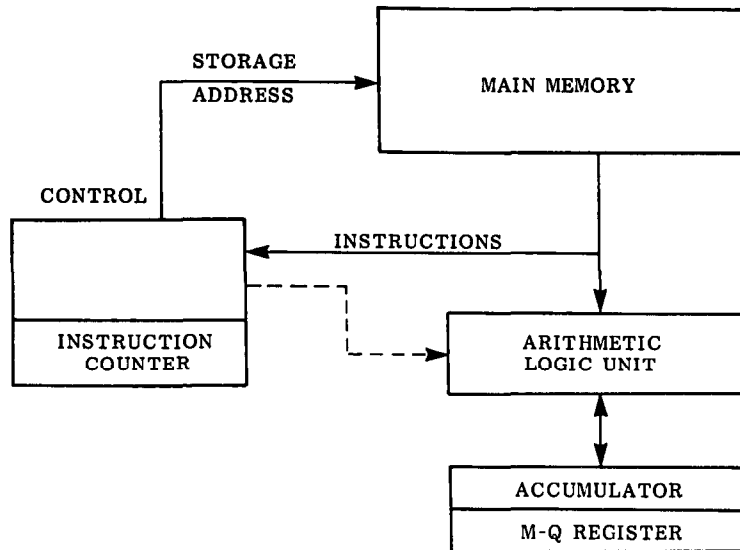


Fig. 9.2 Simple register configuration.

(ALU) component of the processing unit. The accumulator can, for example, receive a number from main memory and add it to the one already stored in it, clear its contents, and transfer its contents to main memory, among other properties. Most arithmetic operations store the results in the accumulator. In conjunction with the accumulator, an additional register is often used in order to implement multiply and divide operations. These operations usually require the additional register to hold partial results during the iterative execution of the instruction. This register has been called the Q register or the M-Q register since it holds the multiplier or quotient.

The length of the accumulator and other registers used to store data is determined by the largest number that needs to be held or processed. Since a fraction can be infinitely long, the choice of length is arbitrary. The choice sets the maximum integer size and determines the precision of the central processing unit. (See Section 9.3.6 for a discussion of scaling, which enables larger integers to be represented.) The relationship of the length of the accumulator to the length of a memory word is subject to varying implementation choices. The architecture defines data types (see later discussion) that may be shorter in length than a memory word or, alternatively, may require more than one memory word. For simplicity we may think of the accumulator length and memory word length as equal, but this is not necessary. The accumulator and memory, in any event, must accommodate the data types defined by the architecture.

Other registers are also utilized to store control information in the processing unit. These are generally not "visible" to the architecture but are described here to provide a description of execution of instructions. The implementation of these control registers may vary. The principal one is the **instruction counter**, which holds the main-memory address of the current instruction being executed. To begin a program, this register is set to the location of the first instruction in the program, either by an operator in a simple system or by the operating system. Subsequently, the register is incremented during the normal course of execution of each instruction and in preparation for obtaining the next instruction from main memory. Each instruction is brought from main memory into the processing unit and resides in a register (the instruction register) during the course of instruction execution.

In general, an instruction set may include instructions of different lengths for reasons to be explained later. The length in bits of an instruction is often a multiple of the length of a memory word since efficiency in storing programs in memory is a major objective.

The execution of a single instruction (see also Section 9.9.1) involves a number of elementary steps such as:

- Increment instruction counter.
- Read next instruction from main memory.
- Interpret the instruction.
- Read operand (if required).

FUNCTION	REGISTER 1	REGISTER 2
----------	------------	------------

Fig. 9.3 Instruction format.

Perform the arithmetic function.

Store result (if required). (In branch-type instructions, the result may be a modification of the contents of the instruction counter.)

The accumulator represents all that is necessary to describe the registers of a simple architecture. The instruction set for such an architecture can also be quite simple. In general, an instruction will contain the function code, identifying the function to be performed, and an address in main memory. The address may refer to data to be fetched or stored or to a new location for the next instruction.

If main-memory access time were as rapid as that of registers, there would perhaps be no need to create a more complex architecture. Instructions would then execute as rapidly with operands from main memory as they do with operands from intimately located high-speed registers. However, historically, this factor plus other ease-of-use factors have combined to generate more registers and more architectural structure.

For instance, in the matter of addressing main memory for data, it is convenient to hold an index to be added to the base address of an array of data. The index provides a means to traverse the array without modifying the base address. In this case a more complex instruction may be added to the instruction set, or a program loop using simple instructions may be used. In either case the instruction itself now must contain some reference to the index register. In Fig. 9.3, the instruction contains three fields, identifying the function to be executed, the register containing the base address, and the register containing the index (in general, there may be more than one).

An architecture may utilize multiple base registers and multiple index registers for program efficiency or ease of use. For example, an architecture with only one base register may require reloading from main memory often to deal with multiple arrays of data. While this gets back to the implementation question of why registers are required at all, the architecture has been adjusted or enlarged to accommodate implementation efficiency. At this point, it is convenient to refer to the multiple index and base registers (as well as other registers used to store data) as a *local memory*. Local memory may be any size, and a field (or fields) of some instructions is used to refer to the desired word, just as a field in an instruction may refer to main memory. The local memory shown in Fig. 9.4 is that of the IBM System/370 architecture. For clarity, these registers are shown in two groups, one for indices, integers, and addresses, the other for floating-point numbers. These two groups may contain registers of different length, but this is not necessary. The arithmetic-logic unit utilizes the integer group of registers for fixed-point operations and the other register group for floating-point operations. In general, variable-field-length operations make references to main memory for data types that do not have a fixed length, such as alphanumeric text.

The registers remain an area where the user program can be entirely free of interaction with the operating system. If a user program is interrupted, the contents of these registers are stored in main memory and reloaded when the program is restarted. The operating system may be required to assist in saving the registers. Interaction with main memory, on the other hand, is subject to operating system control over the physical location and protection of a main-memory word. In Fig. 9.4 a virtual address translation is shown as one means of accomplishing this control. This is discussed in detail in the following section.

At this point, the nature of the information being stored or processed is not being examined. This is treated in sections that follow on the instruction set and data types.

9.1.3 Addressing

Increasing complexity of the central processing unit instruction set derives in a major way from the methods employed in addressing main memory and local memory. The address may be carried directly in the instruction or may be constructed by references to local-memory registers. Instructions may reference more than one address; however, we treat first the concepts of addressing memory and then the modes by which the addresses are constructed.

Main Memory

Traditional methods of addressing main memory use an address that identifies a specific word in real memory. (Reference is made to "real" memory here to differentiate from "virtual" memory, which is discussed later.) Differences in the implementation of main memory may require fetching or storing a