SCIENTIFIC AMERICAN

Microprogramming

Author(s): David A. Patterson

Source: *Scientific American*, Vol. 248, No. 3 (March 1983), pp. 50-57

Published by: Scientific American, a division of Nature America, Inc.

Stable URL: https://www.jstor.org/stable/10.2307/24968851

JSTOR is a not-for-profit service that helps scholars, researchers, and students discover, use, and build upon a wide range of content in a trusted digital archive. We use information technology and tools to increase productivity and facilitate new forms of scholarship. For more information about JSTOR, please contact support@jstor.org.

Your use of the JSTOR archive indicates your acceptance of the Terms & Conditions of Use, available at https://about.jstor.org/terms

Scientific American, a division of Nature America, Inc. is collaborating with JSTOR to digitize, preserve and extend access to *Scientific American*

This content downloaded from 145.15.244.233 on Mon, 31 Mar 2025 10:28:27 UTC
All use subject to https://about.jstor.org/terms

Qualcomm Ex. 1025 - 1/9

Microprogramming

In most modern computers the routing of information is controlled at the lowest level by a microprogram: a set of stored instructions that functions in place of a completely "hard-wired" control system

by David A. Patterson

A fundamental issue in the design of any computer is how to control, or steer, the electrical signals that represent information. In the arithmetic and logic unit, where the actual processing of information is done, signals must be routed between various counters, adders and other components. The control system must also mediate the transfer of information between the central processor, the main memory units and the various input and output devices. In one approach the control system is completely "hard-wired," that is, it is laid down permanently in the processor's electrical circuitry. A second approach is more flexible and in many cases less expensive. The essential idea is to reduce the complexity of the control system by recording the detailed instructions for controlling the computer in a coded form. In other words, the sequence of paths a signal is to follow is embodied in a program, which is stored in a separate memory unit incorporated into the processor.

In the hierarchy of programs that operate a computer the instructions executed by the control system occupy the lowest and most elementary level; each instruction specifies a single functional state of the machine. Because the control instructions are responsible for such fine details, the task of defining and encoding them is termed microprogramming, thereby distinguishing it from the writing of the higher-level programs known generically as software. A set of control instructions—a microprogram—is said to be written in microcode. The idea of microprogramming was conceived more than 30 years ago, soon after the advent of the first computers. At the time the hardware needed to implement the idea did not yet exist. The method has been adopted, however, in most computers being built today.

In recent years a good deal of confusion has arisen about the meaning of the term microprogramming, owing largely to the advent of the microprocessor, the "computer on a chip" that is at the heart of the latest products of the progressive miniaturization of silicon-based semiconductor technology. It must be emphasized that microprogramming a computer is not the same as programming a microcomputer; in principle any computer, from the largest "mainframe" system to the smallest personal computer, can be designed with a microprogrammed control system. To avoid such confusion microprograms are sometimes classified as firmware, thereby signifying their intermediate status between hardware and software. For clarity I shall adopt the following convention: the term macroinstruction will refer to a software instruction and the term microinstruction will refer to a firmware instruction.

Before microprogramming can be explained it is necessary to review some of the basic principles of computer hardware and computer programming. In many ways a computer is like a simple calculator. Numbers are introduced into both devices by means of a keyboard, and they are read out on a display unit. Some calculators have a memory key to record a number for later retrieval; saving and restoring numbers is also the purpose of the memory unit in a computer. A computer memory, however, is capable of handling millions of numbers. To keep track of so many numbers each one is assigned a specific address.

Both a calculator and a computer must be told what to do. Giving a computer a command is analogous to pressing an operator key on a calculator. For example, almost every computer has the macroinstructions ADD, SUB, MULT and DIV, which correspond exactly to the keys marked $+$, $-$, $\times$ and $\div$ on a calculator. Similarly, the macroinstruction LOAD performs the same function as the memory key on a calculator.

Unlike a simple calculator, however, a computer is preeminently a decision-making machine. To understand the distinction consider the sequence of operations represented by the last few sections of Form 1040, the U.S. Individual Income Tax Return [*see top illustration on page 53*]. A computer programmer faced with the chore of completing the form might begin by making a flow chart, which serves as a kind of blueprint for constructing a program [*see illustration at bottom left on page 53*]. A taxpayer working with a simple calculator could follow the flow chart, carrying out the instructions in sequence until he reached the diamond-shaped "decision" step. He would then compare the numbers in the preceding two boxes and, depending on which number was greater, choose the appropriate path to continue his calculations.

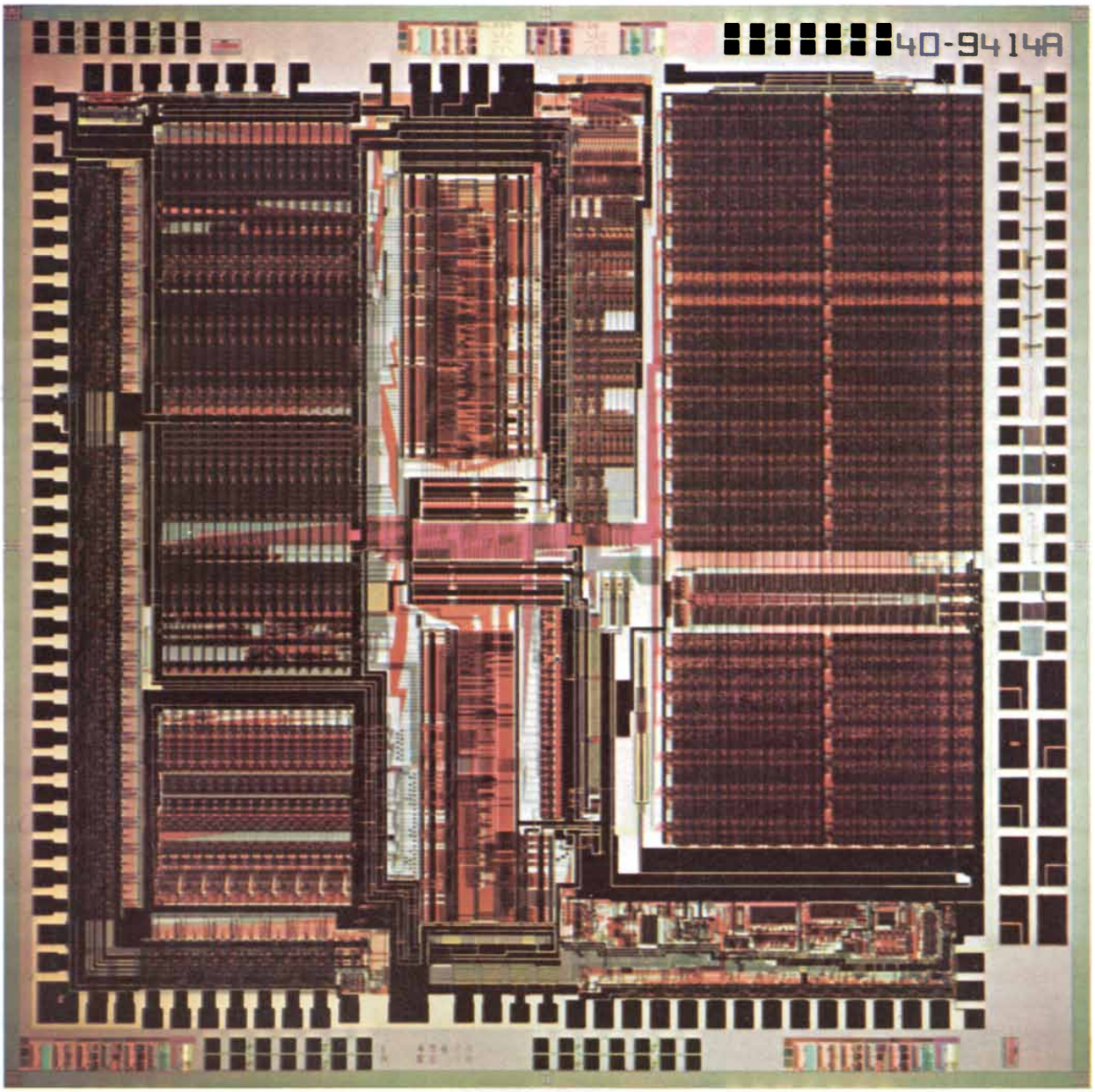
For a taxpayer working with a computer rather than a calculator the programmer could go a step further: he could get the computer to make the decision by itself. Following the flow chart, he would first instruct the computer to add successively the numbers entered on lines 50 through 58 of the tax form, storing the sum of each addition at a specific address in the computer's memory; in this case the addresses of the numbers are designated $L50$, $L51$, $L52$ and so on, and they are added successively to an "accumulator" address designated A [*see illustration at bottom right on page 53*]. The total for this sequence of steps would then be stored at address $L59$. Next the programmer would do the same for lines 60 through 66, storing the total for the second sequence at address $L67$. At this point (step 19) he would give the computer the following macroinstruction: "JUMPIF $A < L59$, 23." This macroinstruction, called a conditional jump, tells the computer to begin executing a new sequence of macroinstructions at step 23 if the number at the accumulator address is less than the number at address $L59$. If it is not, the program would continue with step 20.

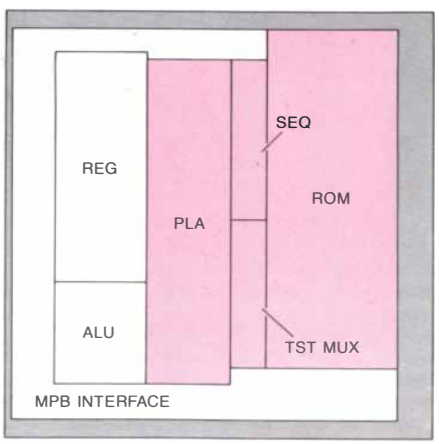
It is the conditional-jump macroinstruction, corresponding to the diamond-shaped box in the flow chart, that endows the computer with the ability to

© 1983 SCIENTIFIC AMERICAN, INC

This content downloaded from
145.15.244.233 on Mon, 31 Mar 2025 10:28:27 UTC
All use subject to https://about.jstor.org/terms

Qualcomm Ex. 1025 - 2/9





MICROPROGRAMMED MICROPROCESSOR is at the heart of the Hewlett-Packard Company's HP 9000 family of computers. The microelectronic circuit is the densest of its kind ever made: it has the equivalent of some 450,000 transistors on a square chip of semiconducting silicon 6.3 millimeters on a side. The standard spacing between the signal-carrying lines is one micrometer. The map at the left shows the seven major subsections of the chip. The colored area identifies those subsections that constitute the processor's control system and the white area corresponds to the data path. The control system consists of four parts: a read-only memory (*ROM*), where the microinstructions for controlling the flow of information throughout the computer are stored; a programmed-logic array (*PLA*), where most of the microinstructions are decoded before being transmitted to their various destinations; a sequencing device (*SEQ*), where the order of the microinstructions is arranged, and a test multiplexer (*TST MUX*), where certain conditional microinstructions are decoded. The three other subsections are an array of memory registers (*REG*), where the data being handled by the computer are stored in the form of "words," each of which is 32 bits long; an arithmetic and logic unit (*ALU*), where the actual processing of the 32-bit words is done, and a memory-processor-bus (*MPB*) interface, which oversees communications between the processor chip and the rest of the computer. The rectangular shapes at the top, bottom and right edges of the chip include test devices, special resolution patterns and alignment marks that are unrelated to the operation of the microprocessor.

© 1983 SCIENTIFIC AMERICAN, INC

This content downloaded from
145.15.244.233 on Mon, 31 Mar 2025 10:28:27 UTC
All use subject to https://about.jstor.org/terms

Qualcomm Ex. 1025 - 3/9

make decisions, provided of course that the programmer can represent the decisions as simple arithmetic tests. The programmer builds a program by constructing a maze of conditional jumps that leads each input number to its correct destination. On the average one out of 10 macroinstructions in a computer program is a conditional-jump macroinstruction; many computers available today can execute a macroinstruction in less than a millionth of a second, so that a computer typically makes more than 100,000 decisions of this kind per second. The macroinstructions for making the decisions are themselves encoded as numbers and stored along with the other numbers handled by the computer in the main memory. The principle of the stored program, the invention of which was a milestone in the development of the modern digital computer, makes it possible to change the function of a computer by changing the contents of its memory unit instead of by changing its hardware.

As computers get faster they can execute more macroinstructions and make more decisions in a given time span; hence it becomes possible to run increasingly complex programs. Under the circumstances mistakes are difficult to avoid. Programmers prefer a term whose connotations are not quite the same as those of "mistake," however: they say that a program has a "bug." To appreciate the special sense of this usage, consider again the illustration of the income-tax form on the opposite page. Line 68 tells the taxpayer what to do if the amount on line 67 is larger than the amount on line 59, and line 71 explains what to do if line 59 is larger than line 67, but nowhere in the form or in the accompanying instructions is there any indication of what to do if the amounts on lines 59 and 67 are equal! Evidently the Internal Revenue Service has overlooked a bug in its program. It is precisely this kind of detail that a programmer must keep in mind when he is writing and "debugging" a computer program. The difficulty of the task is compounded when it comes to the fine details of writing and debugging a microprogram.

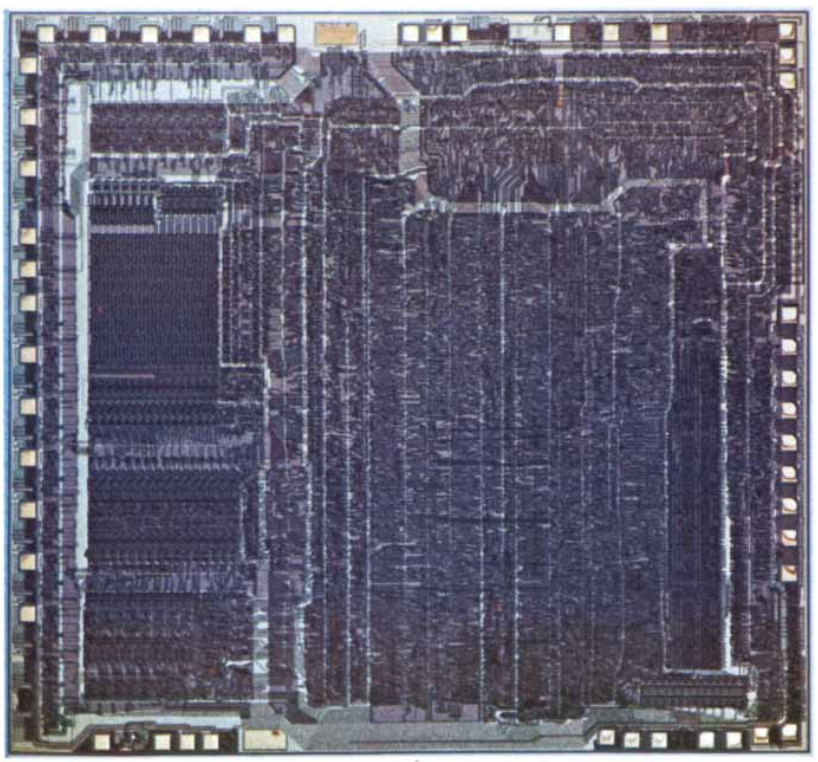
To see how decisions are made and macroinstructions are executed by a computer one must look deeper into the operation of the central processing unit. The processor consists of two major parts: the control system, which makes the decisions and issues the resulting commands to the rest of the hardware, and the arithmetic and logic unit, or data path, where calculations are actually carried out. At the heart of the arithmetic and logic unit is an adder, and in principle all the arithmetic functions can be reduced to repetitive addition steps. Each part of the processor has its own array of short-term memory devices, called registers.

The organization of the computer at this level resembles plumbing. The wires connecting the hardware devices can be thought of as pipes channeling the flow of information. The control system directs the flow by sending an "on" or an "off" signal to open or close each of a large number of valvelike electronic devices called gates [*see top illustration on page 54*]. Because the gates must be opened and closed at the right moment to get the information to the right place, the control signals must be precisely timed. The rhythm of the entire processor is therefore synchronized to an internal clock. Each macroinstruction corresponds to a sequence of clock cycles, with each clock cycle marking a single transfer of information along the data path. The function of the control system is to supply the control signals during the right clock cycles.

In a hard-wired control system a network of electronic logic must be devised to recognize each macroinstruction in the computer's repertory. The macroinstruction itself is a pattern of binary digits, embodied in a pattern of signals sent to the processor. The control system must somehow transform this pattern into the quite different set of signals needed to open and close various gates in the data path. Furthermore, the signals issued by the control system must be timed to coincide with the appropriate clock cycles. Each macroinstruction must give rise to a different sequence of control signals, and so the system becomes more complex as the number of macroinstructions increases. Changes can be made and bugs can be eliminated only by rewiring.

In the early days of the computer the design of the control hardware was a major problem. In comparison the task of designing the other components of the hardware was fairly easy. A teletypewriter could serve as an input-and-output device, and the repetitiveness of the memory structures and of the circuit elements in the data path simplified their design and construction. The hardware needed to count clock cycles and to issue control signals at the appropriate times made the control system quite irregular, and that in turn made the job of designing this part of the central processor much harder.

In 1949 Maurice V. Wilkes of the University of Cambridge set out to find a better way to organize the control function. After two years of study he concluded that the best approach is to



NONMICROPROGRAMMED MICROPROCESSOR is distinguished by its highly irregular, hard-wired control section, which occupies the large area at the right in this photograph. The two smaller and more regular areas at the left correspond to the *REG* and *ALU* subsections in the illustration on the preceding page. The irregularity of such a hard-wired control system makes it difficult to design and modify. At the time this chip was made (about five years ago) the designers thought that microprogramming would have required a much larger chip.

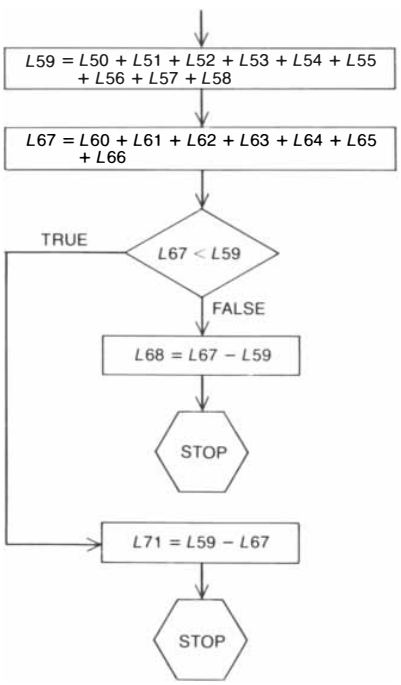
© 1983 SCIENTIFIC AMERICAN, INC

This content downloaded from
145.15.244.233 on Mon, 31 Mar 2025 10:28:27 UTC
All use subject to https://about.jstor.org/terms

Qualcomm Ex. 1025 - 4/9

	50	**Balance.** Subtract line 49 from line 40 and enter difference (but not less than zero) . ▶			50	
Other Taxes (Including Advance EIC Payments)	51	Self-employment tax *(attach Schedule SE)*			51	
	52	Minimum tax *(attach Form 4625)*			52	
	53	Alternative minimum tax *(attach Form 6251)*			53	
	54	Tax from recapture of investment credit *(attach Form 4255)*			54	
	55	Social security (FICA) tax on tip income not reported to employer *(attach Form 4137)*			55	
	56	Uncollected employee FICA and RRTA tax on tips *(from Form W–2)*			56	
	57	Tax on an IRA *(attach Form 5329)*			57	
	58	Advance earned income credit (EIC) payments received *(from Form W–2)*			58	
06	59	**Total tax.** Add lines 50 through 58 ■			59	
Payments Attach Forms W–2, W–2G, and W–2P to front.	60	Total Federal income tax withheld	60			
	61	1982 estimated tax payments and amount applied from 1981 return	61			
	62	Earned income credit. If line 33 is under $10,000, see page 15 of Instructions	62			
	63	Amount paid with Form 4868	63			
	64	Excess FICA and RRTA tax withheld (two or more employers)	64			
	65	Credit for Federal tax on special fuels and oils *(attach Form 4136)*	65			
	66	Regulated Investment Company credit *(attach Form 2439)*	66			
	67	**Total.** Add lines 60 through 66 ▶			67	
Refund or Amount You Owe	68	If line 67 is larger than line 59, enter amount **OVERPAID** ▶			68	
	69	Amount of line 68 to be **REFUNDED TO YOU** ▶			69	
	70	Amount of line 68 to be applied to your 1983 estimated tax ▶	70			
	71	If line 59 is larger than line 67, enter **AMOUNT YOU OWE.** Attach check or money order for full amount payable to Internal Revenue Service. Write your social security number and "1982 Form 1040" on it. ▶ (Check ▶ ☐ if Form 2210 (2210F) is attached. See page 16 of Instructions.) ▶ $			71	

FAMILIAR FORM serves as an example of a sequence of operations that can be done by a computer program. The illustration reproduces the last 22 lines of Form 1040, the U.S. Individual Income Tax Return for 1982, prepared by the Internal Revenue Service.



FLOW CHART serves as a kind of blueprint of the sequence of operations needed to complete the last 22 lines of the income-tax form. The amount entered on each line of the form is stored at a specific address in the computer's memory; in this case the addresses of the numbers are designated *L50*, *L51*, *L52* and so on. The diamond-shaped box is computer programmer's symbol for a "decision" step.

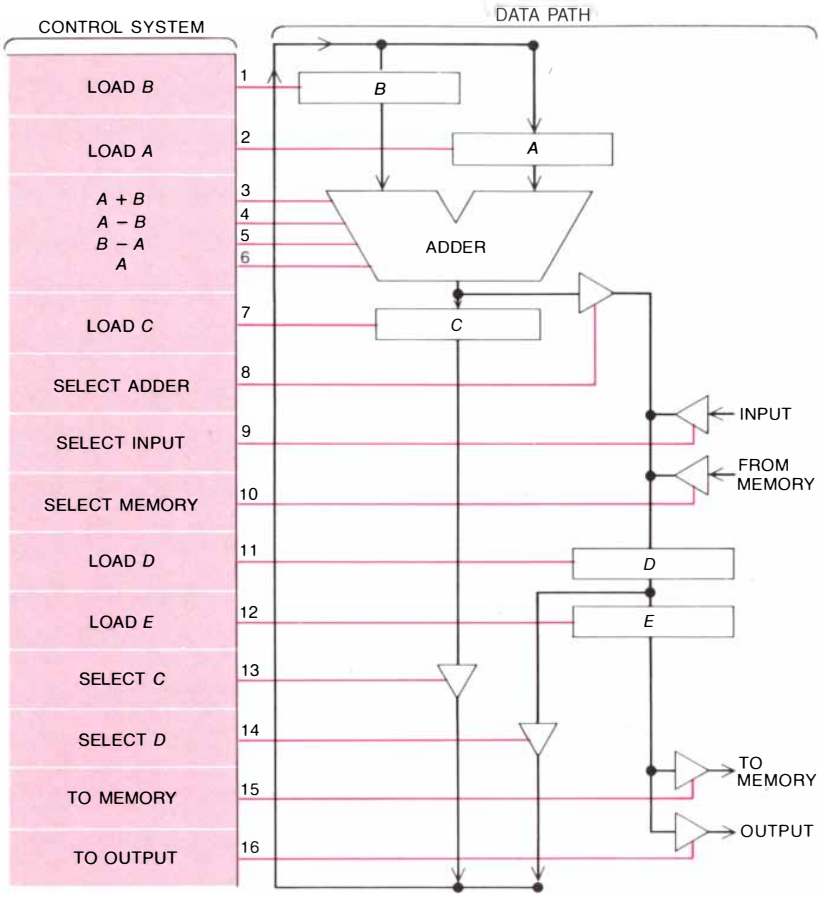
ADDRESSES	INSTRUCTIONS	COMMENTS
1	LOAD *L50*, *A*	*A* = *L50*
2	ADD *L51*, *A*	*A* = *L50* + *L51*
3	ADD *L52*, *A*	*A* = *L50* + *L51* + *L52*
4	ADD *L53*, *A*	*A* = *L50* + *L51* + *L52* + *L53*
5	ADD *L54*, *A*	*A* = *L50* + *L51* + *L52* + *L53* + *L54*
6	ADD *L55*, *A*	*A* = *L50* + *L51* + *L52* + *L53* + *L54* + *L55*
7	ADD *L56*, *A*	*A* = *L50* + *L51* + *L52* + *L53* + *L54* + *L55* + *L56*
8	ADD *L57*, *A*	*A* = *L50* + *L51* + *L52* + *L53* + *L54* + *L55* + *L56* + *L57*
9	ADD *L58*, *A*	*A* = *L50* + *L51* + *L52* + *L53* + *L54* + *L55* + *L56* + *L57* + *L58*
10	STORE *A*, *L59*	*L59* = *L50* + *L51* + *L52* + *L53* + *L54* + *L55* + *L56* + *L57* + *L58*
11	LOAD *L60*, *A*	*A* = *L60*
12	ADD *L61*, *A*	*A* = *L60* + *L61*
13	ADD *L62*, *A*	*A* = *L60* + *L61* + *L62*
14	ADD *L63*, *A*	*A* = *L60* + *L61* + *L62* + *L63*
15	ADD *L64*, *A*	*A* = *L60* + *L61* + *L62* + *L63* + *L64*
16	ADD *L65*, *A*	*A* = *L60* + *L61* + *L62* + *L63* + *L64* + *L65*
17	ADD *L66*, *A*	*A* = *L60* + *L61* + *L62* + *L63* + *L64* + *L65* + *L66*
18	STORE *A*, *L67*	*L67* = *L60* + *L61* + *L62* + *L63* + *L64* + *L65* + *L66*
19	JUMP IF *A* < *L59*, 23	GO TO ADDRESS 23 IF *A* < *L59* (SAME AS *L67* < *L59*)
20	SUB *A*, *L59*	(*L67* ≥ *L59*) THEREFORE *L67* − *L59*
21	STORE *A*, *L68*	*L68* = *L67* − *L59*
22	STOP	FINISHED
23	LOAD *L59*, *A*	(*L67* < *L59*) THEREFORE *L59* − *L67*
24	SUB *L67*, *A*	*A* = *L59* − *L67*
25	STORE *A*, *L71*	*L71* = *A* (= *L59* − *L67*)
26	STOP	FINISHED

PROGRAM OF MACROINSTRUCTIONS for completing the income-tax form follows the flow chart. Each macroinstruction is itself stored at a memory address (*first column at left*). The numbers at addresses *L50*, *L51*, *L52* and so on are added successively to an "accumulator" address designated *A*. Steps 1 through 10 correspond to the topmost box in the flow chart and and steps 11 through 18 correspond to the second box from the top. The conditional-jump macroinstruction at step 19 ("JUMPIF *A* < *L59*, 23") is equivalent to the diamond-shaped decision box. Steps 20 and 21 correspond to the box on the "False" path of the flow chart and steps 23, 24 and 25 correspond to the box on the "True" path. The comments at right are notes to remind the programmer of what is going on; they do not affect the execution of the program.

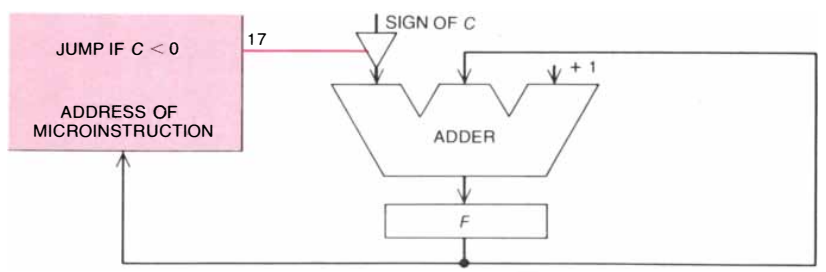
© 1983 SCIENTIFIC AMERICAN, INC

This content downloaded from 145.15.244.233 on Mon, 31 Mar 2025 10:28:27 UTC
All use subject to https://about.jstor.org/terms

Qualcomm Ex. 1025 - 5/9



IDEALIZED CENTRAL PROCESSOR relies on 16 control wires (*colored lines*) to guide the electrical signals representing information through a simplified data path (*black lines*). In this case the arithmetic and logic unit has only one special-purpose device: an adder, which receives numbers from two input wires, adds or subtracts them in accordance with commands received over control wires *3* through *6* and sends the result through an output wire. The rectangular shapes are registers, short-term memory components that can hold one word of information at a time. The triangular shapes are gates, devices that pass the information along when the associated control wire is on and block the information when the control wire is off. In a nonmicroprogrammed processor each of the commands listed in the control section is defined by specific control hardware. In the microprogrammed version of the same processor each command is stored as a separate microinstruction in the microprogram memory. A sample microprogram for this simplified processor is given in the top illustration on the opposite page.



ADDITIONAL HARDWARE IS NEEDED to enable the microprogrammed version of the simplified processor depicted at the top of the page to make decisions of the conditional-jump type. Register *F* holds the address in microprogram memory of the microinstruction currently in control of the data path. The function of the extra adder is to add one ("+1") to the address of the current microinstruction, thereby transmitting the address of the next sequential microinstruction in the microprogram. The adder also feeds information from the data path back into the control system. The sign of the number in register *C* goes through a gate controlled by wire *17* into the third input of the adder. When this wire is off, the microinstruction at $F + 1$, the next sequential address, is executed (since $0 + F + 1$ is equivalent to $F + 1$). The adder adds either $0 + F + 1$ or $1 + F + 1$ depending on whether number in register *C* is respectively positive or negative. Thus with the aid of control wire *17* the microprogrammer can conditionally jump over a microinstruction, depending on the sign of the number in *C*. The microprogram for the conditional-jump operation is given in the bottom illustration on the opposite page.

think of the control system as a matrix, or rectangular grid, with each row of squares corresponding to a clock cycle and each column associated with a control wire. For example, the simplified processor shown in the top illustration at the left has 16 control wires, and so the control matrix corresponding to it is 16 squares wide. Choosing a sequence of operations then becomes a simple matter of putting a suitable combination of binary symbols in the squares: for each control wire that must be on during a particular clock cycle write a 1 in the corresponding square, and for each control wire that must be off write a 0.

The hardware equivalent of the control matrix is a simple, repetitive memory structure. The content of each cell in a row determines the state of the corresponding control line for the duration of one clock cycle. The pattern of digits that constitutes a macroinstruction serves merely to select the appropriate row (or sequence of rows) in the control memory. In other words, the macroinstruction becomes an address designating the row. Designing a control system then becomes more a matter of software than of hardware; the complexity lies in choosing the right pattern of 1's and 0's to store in the memory rather than in specifying the right combination of hard-wired circuits to generate the control signals.

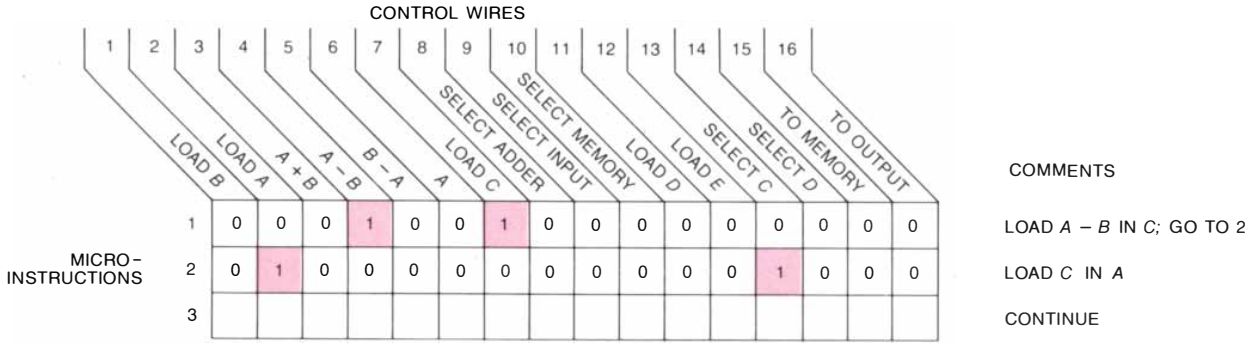
Wilkes recognized the similarities between his approach and conventional programming, and so he borrowed terms from programming to describe his ideas, in each case adding the prefix "micro-" to indicate the elementary nature of the control function. Hence "microprogramming" and its family of related terms. In particular Wilkes named each row of squares in the control matrix a microinstruction, and each sequence of rows that executes a single macroinstruction he called a microprogram. The memory structure designed to hold the microprogram is called the microprogram memory. The concept of microprogramming made it easier to understand the control function and, by replacing the complex control circuitry with a repetitive array of memory cells, made it easier to build the hardware. Most important, it gave the computer a new flexibility: the control system could now be changed without redesigning the hardware.

Consider a sample microprogram for the simplified processor discussed above [*see top illustration on opposite page*]. Suppose the operation controlled by the microprogram is to subtract the number stored in memory register *B* from the one in the accumulator register *A* and then to store the result in *A*. By examining the data path in the hardware diagram one can see that the device labeled "adder" is capable not only of

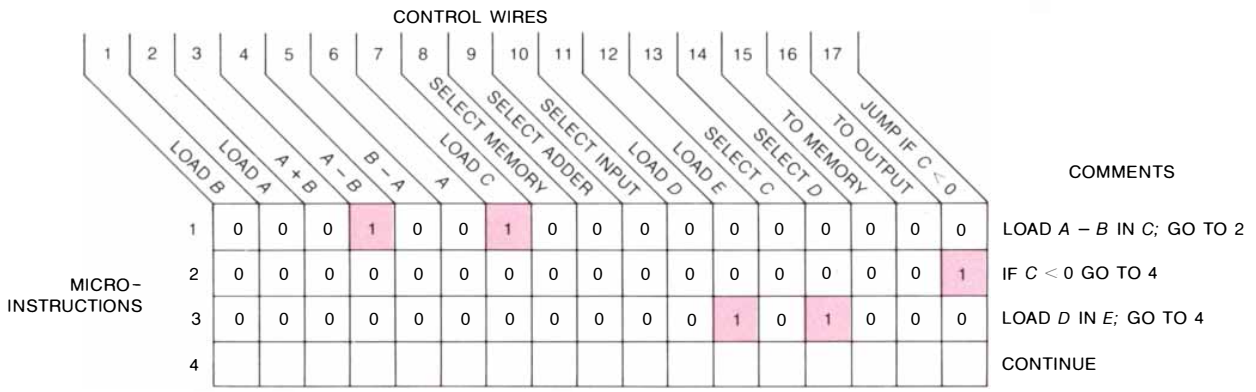
© 1983 SCIENTIFIC AMERICAN, INC

This content downloaded from
145.15.244.233 on Mon, 31 Mar 2025 10:28:27 UTC
All use subject to https://about.jstor.org/terms

Qualcomm Ex. 1025 - 6/9



SAMPLE MICROPROGRAM for a single macroinstruction that can be executed by the simplified processor shown in the top illustration on the opposite page consists of a sequence of rows in a control matrix, or rectangular grid, in which each row corresponds to a microinstruction and each column is associated with a control wire. For each control wire that must be on during a particular clock cycle a 1 is written in the corresponding square and for each control wire that must be off a 0 is written. (Here "on" is also signified by color and "off" by white.) The operation controlled by the microprogram is to subtract the number stored in memory register *B* from the one in the accumulator register *A* and then store the result in *A*. As in the case of macroinstructions, the comments at the right are just reminders.



DECISION-MAKING MICROPROGRAM utilizes the augmented control hardware shown in the bottom illustration on the opposite page. The operation controlled by the microprogram is to load the number in register *D* into register *E* if *A* is greater than or equal to *B*. The conditional jump calls for an additional square to give the command to jump to microinstruction *4* if the number in register *C* is negative (column 17). Thus the conditional-jump macroinstruction comes down to a choice between microinstruction *3* and microinstruction *4*.

adding the two numbers but also of subtracting either one from the other. Before the result can be put back in *A*, however, it must be held temporarily in register *C;* hence register *C* must be loaded and then unloaded into the data path that leads back to *A*. The subtraction operation calls for two microinstructions. The first microinstruction tells the adder to subtract the number in *B* from the one in *A* and to put the result into *C;* accordingly control wires 4 ("*A* − *B*") and 7 ("LOAD *C*") are turned on. The next microinstruction moves the information from *C* into *A* by opening a gate in the wire from *C* in order to pass the information along to *A;* hence control wires 13 ("SELECT *C*") and 2 ("LOAD *A*") are turned on.

Now consider a somewhat more complicated microprogram for the same processor. The second microprogram calls for a decision-making process similar to the conditional jump in the income-tax program. The decision comes down to a choice between two microinstructions. For this purpose one more square must be added to the microinstructions and one more control wire must be added to the processor. The augmented control hardware is shown in the bottom illustration on the opposite page and the revised microprogram in the bottom illustration on this page.

The operation controlled by the second microprogram is to load the number in register *D* into register *E* if *A* is greater than or equal to *B*. First *B* is subtracted from *A* and the result is stored in *C*. If the number in *C* is negative, it follows that *A* is less than *B* and so the decision is to jump over the next microinstruction and resume the microprogram at microinstruction 4. If the result of the subtraction is a positive number or zero, on the other hand, *A* is greater than or equal to *B*, and the microprogram proceeds to the next sequential microinstruction, which loads *E* from *D*. A square is added to the microinstructions to give the command to jump to microinstruction 4 if *C* is negative, and a corresponding control wire is added to the processor's hardware.

A macroinstruction is said to be "interpreted" by the microcode to yield the control signals that actually manipulate the flow of information in the data path. Thus the sequence of microinstructions in the second example above is the microcode interpretation of the conditional-jump macroinstruction written as "JUMPIF" at step 19 of the income-tax program. The microprogram deals with the simple arithmetic test presented by the programmer by directing the hardware to test the number in the *C* register to see if it is negative, and if it is, to execute the appropriate conditional-jump microinstruction. For the hardware to perform such a test it must feed information from the data path back into the control system. The hardware needed to determine whether the number in *C* is negative is simply a wire from *C* that represents the sign of the number in that register: the wire is off (0) when the sign is positive (implying that *C* is greater than or equal to zero), and it is on (1) when the sign is negative (implying that *C* is less than zero).

The substitution of simple, repetitive memory structures for complex hard-wired control circuits yields two main advantages: it makes it easier to understand and build the control sys-

© 1983 SCIENTIFIC AMERICAN, INC

This content downloaded from
145.15.244.233 on Mon, 31 Mar 2025 10:28:27 UTC
All use subject to https://about.jstor.org/terms

Qualcomm Ex. 1025 - 7/9

tem, and it makes it easier to modify the system. One can correct a mistake in a microprogrammed control system simply by changing the contents of the memory. For example, suppose one decides to change the preceding microprogram from one that loads E from D if A is less than B to one that loads E from D if B is less than A. The first microinstruction in the original microprogram loads C with the result of $A - B$. If one turns off control wire 4 and turns on control wire 5, however, then the microinstruction would load C with the result of $B - A$. In either case the next microinstruction determines whether the number in C is less than zero and jumps if it is. Hence one can correct the operation by changing only two control wires in one microinstruction. With a nonmicroprogrammed control system one would have to change the structure of the hardware to make the same correction.

Besides reducing the cost of correcting the control system the introduction of microprogramming makes it possible to change the control system completely by changing the total contents of the microprogram memory. Thus a single hardware system can be made to serve many different functions.

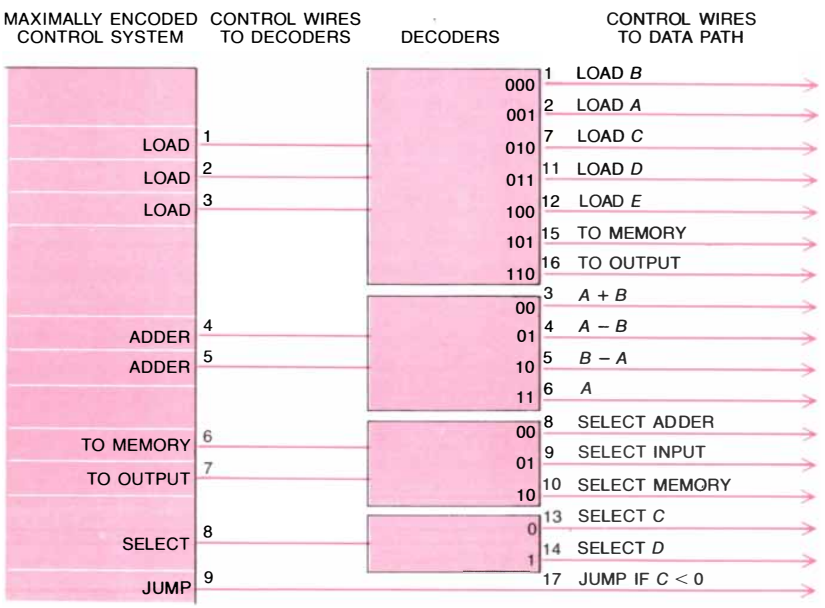
The advantages of microcode were apparent almost from the beginning. Wilkes's idea, however, was somewhat ahead of the hardware technology. Microprogramming calls for fast and inexpensive memory structures, and they were not available in the 1950's. Furthermore, it was generally thought at the time that any computer designed to take advantage of this ingenious approach would be slower than a machine with a hard-wired control system. As a result microprogramming remained little more than an intellectual curiosity for more than a decade.

It was not until the early 1960's that microprogramming emerged into the limelight, largely as the outcome of a major business decision by the International Business Machines Corporation. In the late 1950's IBM had in effect become many small companies, each one marketing its own type of computer. The computers differed not only in size, speed and price but also in the set of macroinstructions provided to operate them. Because of the differences in macroinstruction sets, programs devised for one type of IBM computer could not be run on another IBM computer. Obviously it would have been much easier to sell computers that operated with standardized software and that differed only in size, speed and price. The problem was to convince the managers of the subsidiary "companies" that it was possible for both the designers of the fastest computers and the designers of the cheapest computers to build efficient machines that relied on the same set of macroinstructions.

As it happened, the managers of the subsidiaries agreed to the innovative concept of a family of software-compatible computers only because they were persuaded that microprogramming made such a plan feasible. If one could microprogram the same hardware to execute different sets of macroinstructions, it was reasoned, then one should be able to microprogram different computers to execute the same macroinstruction set. The perceived marketing advantages of the family concept were so great that IBM made a major investment in the development of memory technologies that could lead to fast and inexpensive microprogram memories. Finally, in 1964 IBM announced a family of seven software-compatible computer models that varied in speed by a factor of 300 and in cost by a factor of 100.

Almost as an afterthought it was decided at IBM that microprogramming could help with the difficult task of making the programs for the old computers compatible with the new ones. Accordingly several models in the new IBM family ended up with two microprograms: one for the new, improved set of macroinstructions and another for the old macroinstruction set. This approach enabled customers to run their old programs without modification on the new hardware. The method proved so popular that for the first few years it was difficult to say whether the new IBM machines were spending more time running the microprogram for the new macroinstructions or the microprogram that emulated the old ones.

Most major computer manufacturers soon followed the lead of IBM in relying on microcode to produce a family of software-compatible machines that vary over a wide range of speeds and prices. The main difficulty encountered along the way was in duplicating the expensive memory technology invented by the engineers at IBM. Rapid progress in microelectronic technology eventually solved the problem. Every year for more than a decade the semiconductor industry was able to double the amount of information that could be stored on a single memory chip. This technology solidified the position of microprogramming as the control method of choice through the 1970's. The exceptions were "supercomputers," the fastest and most expensive processors. In that case special-purpose control is still



NUMBER OF CONTROL WIRES can be reduced by further encoding the control signals and introducing suitable decoding devices. In general this approach involves finding two wires that are rarely both on for the same microinstruction and eliminating one of them, making the other serve a dual function. This schematic hardware diagram depicts a maximally encoded version of the processor shown in the two illustrations on page 54. The control wires to the data path have been rearranged to emphasize the wires that have been encoded together in the new control system. The additional encoding generally results in a control system that is slower than a minimally encoded one (in other words, one in which each microprogram square is matched by one control wire), because fewer operations can proceed simultaneously and because the control signals must pass through the decoders. In addition the resulting microprogram tends to have a larger number of microinstructions. Because each microinstruction has fewer binary digits, however, the cost of microprogram memory can be significantly reduced.

© 1983 SCIENTIFIC AMERICAN, INC

This content downloaded from 145.15.244.233 on Mon, 31 Mar 2025 10:28:27 UTC
All use subject to https://about.jstor.org/terms

Qualcomm Ex. 1025 - 8/9

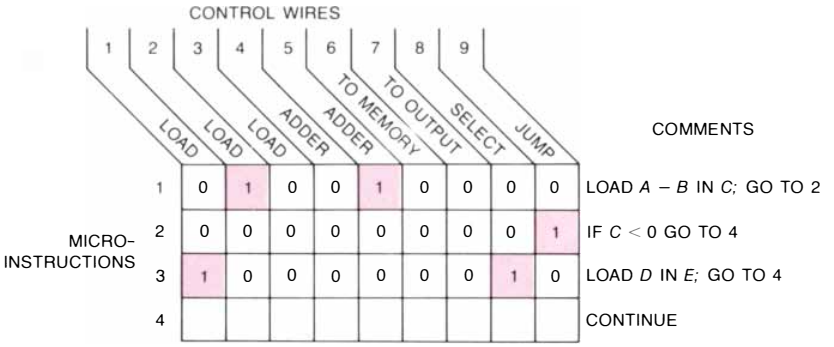
preferred for its greater speed, even though it is harder to build.

When microprocessors began to appear on the scene a decade or so ago, the question naturally arose of how to design their control systems. Repeating history, the first microprocessors were built with nonmicroprogrammed control systems. As the technology has matured, however, the advantages of microprogramming have persuaded most microcomputer designers to choose microcode.

In the meantime one kind of microprogramming has evolved away from the simple matrix concept in which each square is matched to an individual control wire. The improvements have come mainly from the effort to reduce the cost of the microprogram memory. In the execution of any single microinstruction it is generally the case that only a few components of the hardware are active; the control wires leading to all the other elements are off. As a result a microprogram tends to have islands of 1's scattered in a sea of 0's. If the same end could be achieved with less memory (that is, with fewer 0's), the result would be a less expensive memory. The solution is to supply the same information to the data path by further encoding the control signals. In general the additional encoding step involves finding two wires that are rarely both on for the same microinstruction and eliminating one of them, making the other serve a dual function. This approach tends to result in a larger number of microinstructions, but each microinstruction has fewer binary digits. A maximally encoded set of microinstructions is said to be "vertical," because the resulting microprograms are usually tall and narrow. A minimally encoded set of microinstructions is said to be "horizontal," because the resulting microprograms are usually short and wide. Minimally encoded microinstructions generally lead to faster computers, since more operations can be done simultaneously.

The growing importance of microprogramming has also stimulated interest in finding ways to enable computer users to write their own microcode. The potential advantages of transforming the computer from a general-purpose problem solver to one specifically tailored to a given application has made user microprogramming an attractive option. So far the best results obtained by users who have written their programs in microcode have shown an improvement in speed by as much as a factor of 10. In practice the gain in speed depends on the particular data path of the microprogrammed computer and on the application; in some cases the speed advantage is much less, and occasionally it may even disappear.

In most microprogrammed computers the microinstructions are stored in a form of read-only memory, whose contents are determined when the memory array is manufactured. Another kind of microprogrammed computer is capable of changing the contents of its microprogram memory in as little as a few thousandths of a second, while the computer is in operation. Such a machine is said to have a writable control store. The key to this approach is to have many different microcoded applications available to be loaded only when necessary. Endowing a computer with a writable control store is a way of removing the hardware barrier from user microprogramming.



MAXIMALLY ENCODED MICROPROGRAM for the control system on the opposite page is said to be "vertical," because it usually has a larger number of shorter microinstructions, thereby making it taller and narrower than the corresponding "horizontal" microprogram. In this case the encoding was done in such a way as to require no additional microinstructions.

In spite of these efforts microprogramming has remained so difficult that few users have taken full advantage of its potential. Therefore some investigators have attacked the problem of making it easier to write microcode. In many ways the situation is similar to one that arose earlier in the history of computer programming. Then the problem was the difficulty of arranging thousands of macroinstructions into a program that would operate without fail. This problem inspired programmers to write special programs to help write the application programs. The special programs translate high-level computer languages into the simpler macroinstructions a computer can execute. The high-level languages such as BASIC and Pascal are easier to understand and use, thereby making the task of programming less difficult, albeit at some sacrifice in the speed of the program.

For the past few years workers in microprogramming have been trying to create analogous translation programs for microcode. The problem is harder in this case because there are many more potential microinstructions than there are ordinary macroinstructions. A microprogrammed processor might have 50 control wires; since each wire can have one of two states (on or off), the total number of possible microinstructions would be 2^{50}, or approximately 10^{15}. Moreover, since the speed of the microprogram determines the speed of the computer, the demand for efficiency is much higher. The result has been the invention of high-level microprogramming languages aimed at balancing ease of microprogramming with efficiency. Recent developments have been encouraging, but general acceptance among the practitoners of microprogramming is still several years away.

Meanwhile investigators at the IBM Thomas J. Watson Research Center and my colleagues and I in the department of electrical engineering and computer science of the University of California at Berkeley have taken a different approach. We start by discarding the conventional macroinstruction set. We then write new programs that translate the standard high-level programming languages directly into microcode. Machines based on this principle are known generically as reduced-instruction-set computers, or RISC's. To test our ideas our group at Berkeley has designed our own microprocessor, which we call RISC I. In addition we have simplified the translation step by maximally encoding the microinstructions to achieve a highly vertical type of microprogram. In effect, all programs then become microprograms.

Recent tests by our group and by the workers at IBM have shown that a reduced-instruction-set computer would be from three to five times as fast as a conventional complex-instruction-set computer for many applications. Workers at Stanford University and at several industrial laboratories have begun projects to investigate this approach. If the preliminary findings are substantiated, a simpler type of computer might emerge, representing an unexpected advance in the continuing exploitation of the principle of microprogramming.

© 1983 SCIENTIFIC AMERICAN, INC

This content downloaded from
145.15.244.233 on Mon, 31 Mar 2025 10:28:27 UTC
All use subject to https://about.jstor.org/terms

Qualcomm Ex. 1025 - 9/9