MICROPROGRAMMED VERSUS HARDWIRED CONTROL UNITS: HOW COMPUTERS REALLY WORK

Richard R. Eckert
Department of Computer Science
State University of New York
Binghamton, NY 13901

Introduction

For too many students of computer science, the level of understanding of how digital computers really execute programs never quite makes it above that of "black magic." Many of them know what the basic parts of the computer are and what kinds of actions occur in each. But, in too many cases, they cannot put it all together and formulate a clear "big picture" of what is really going on inside the machine. In this article we present a simple computer architecture and describe in detail two alternative ways in which its control section may be organized.

The Basic Computer

Every student of computer science knows that all traditional digital computers have two principal functional parts: the data path section in which processing occurs and the control section which is responsible for decoding instructions and issuing the correct sequence of control signals to make the processing happen in the data path. Basically there are two types of control units: hard-wired controllers and microprogrammed controllers. In order to appreciate the difference and see how computers really work, we present a very simple computer. A block diagram of its data path section is shown in Figure 1.

A single 12-bit-wide bus provides for exchange of information between pairs of registers within the data path section. The registers and the 256 X 12 bit RAM memory are controlled by 16 control signals. Most of the registers have Load (L) and Enable (E) signals. An active L signal to a register causes the contents of the bus to be clocked into that register on the next rising pulse from the system clock. An active E signal enables the tristate outputs of the register, thereby making its contents available to the bus. Therefore, a register transfer from, for example, register A to register B would require active EA and LB control signals.

Processing of data is done by the Arithmetic-Logic-Unit (ALU), a circuit that is capable of adding or subtracting the 12-bit numbers contained in its two input registers: the accumulator (ACC) and register B. The operation performed by the ALU is selected by the Add (A) or Subtract (S) control signals. The accumulator also contains a single flip-flop that is set whenever its contents are negative (i.e., whenever the leading bit is set--meaning a negative two's complement number). The value of this "negative flag" provides input to the controller/sequencer, and, as we shall see, permits implementation of conditional branching instructions.

The machine's RAM memory is accessed by first placing the 8-bit address in the Memory Address Register (MAR). An active Read (R) control signal to the RAM will then cause the selected word from the RAM to appear in the Memory Data Register (MDR). An active Write (W) signal, on the otherhand, will cause the word contained in the MDR to be stored in the RAM at the address specified by the MAR. Since there are no input or output ports in this simple computer, all I/O is memory mapped. In other words, several memory locations are reserved for input/output devices. Memory reads from any of those locations will cause data from the corresponding input device to appear in the MDR; memory writes to them will cause data in the MDR to be sent to the corresponding output device. A word stored in any given memory location may be data to be manipulated by the computer or a coded instruction that specifies an action to be taken.

The data path section also contains a Program Counter (PC) whose function it is to point to the address in RAM of the next instruction to be executed. The Increment Program Counter (IP) control signal causes the contents of the PC to increase by one. Since, as we shall see, instructions on this machine are one word long, this provides a simple mechanism for sequential instruction execution. In addition there is an Instruction Register

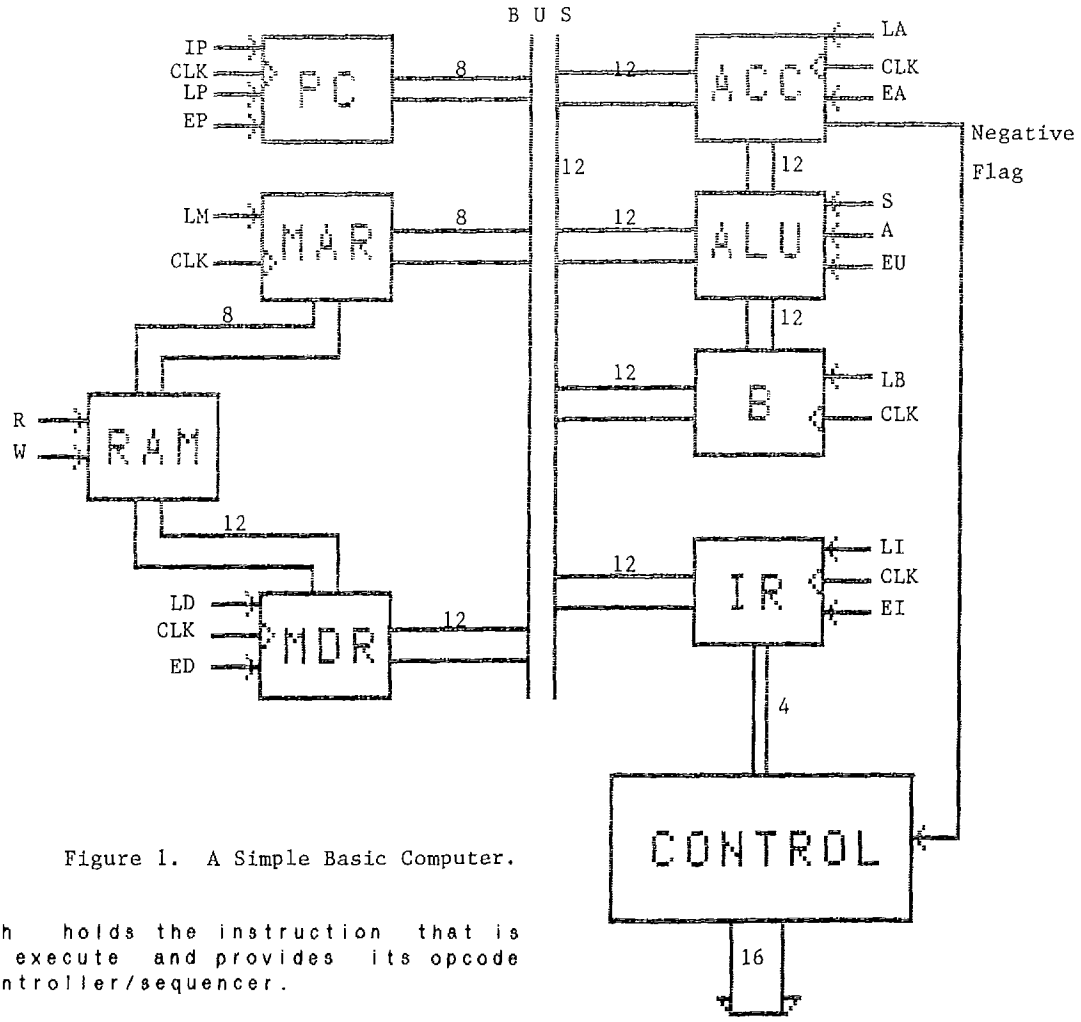

Figure 1. A Simple Basic Computer.

(IR) which holds the instruction that is about to execute and provides its opcode to the controller/sequencer.

The Computer's Instruction Set

An instruction on our simple computer consists of one 12-bit word. The leading four bits form the operation code (opcode) which specifies the action to be taken, and the remaining 8 bits, when used, indicate the memory address of one of the instruction's operands. For those instructions that have two operands, the other operand is always contained within the accumulator.

Table 1 gives eight instructions that form the instruction set we have chosen for our machine. Also shown in the table is the sequence of control signals necessary for execution of each of the instructions in the machine's instruction set and for fetching the next instruction. In each case the register transfers required for execution of each step are shown. For example, in the case of the LDA (load accumulator) instruction, the first step consists of copying the address of the operand, contained in the least significant 8 bits of the instruction register, to the memory address register. Thus the EI (enable IR) and LM (load MAR) control signals are active. The next step is to read the operand from memory into the memory data register. An active R (memory read) signal performs that task. The last step required to execute the LDA instruction is to copy the contents of the memory data register to the accumulator. Active ED (enable MDR) and LA (load accmulator) do the trick.

The Hard-Wired Control Unit

Figure 2 is a block diagram showing the internal organization of a hard-wired control unit for our simple computer. Input to the controller consists of the 4-bit opcode of the instruction currently contained in the Instruction Register and the negative flag from the accumulator. The controller's output is a set of 16 control signals that go out to the various registers and to the memory of the computer, in addition to a HLT signal that is activated whenever the leading bit of the op-code is one. The controller is composed of the following functional units: A ring counter, an instruction decoder, and a control matrix.

The ring counter provides a sequence of six consecutive active signals that cycle

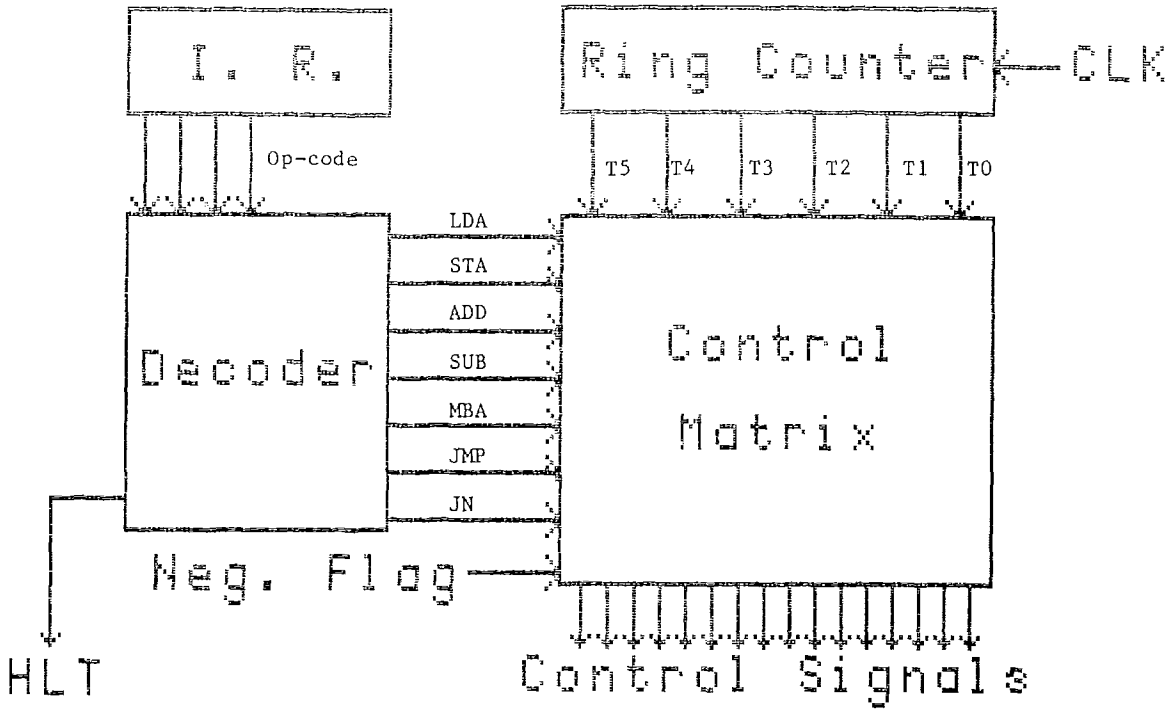

Figure 2. A block diagram of the basic computer's hard-wired control unit.

continuously. Synchronized by the system clock, the ring counter first activates its T0 line, then its T1 line, and so forth. After T5 is active, the sequence begins again with T0. Figure 3 shows how the ring counter might be organized internally.

The instruction decoder takes its four-bit input from the opcode field of the instruction register and activates one and only one of its 8 output lines. Each line corresponds to one of the instructions in the computer's instruction set. Figure 4 shows the internal organization of this decoder.

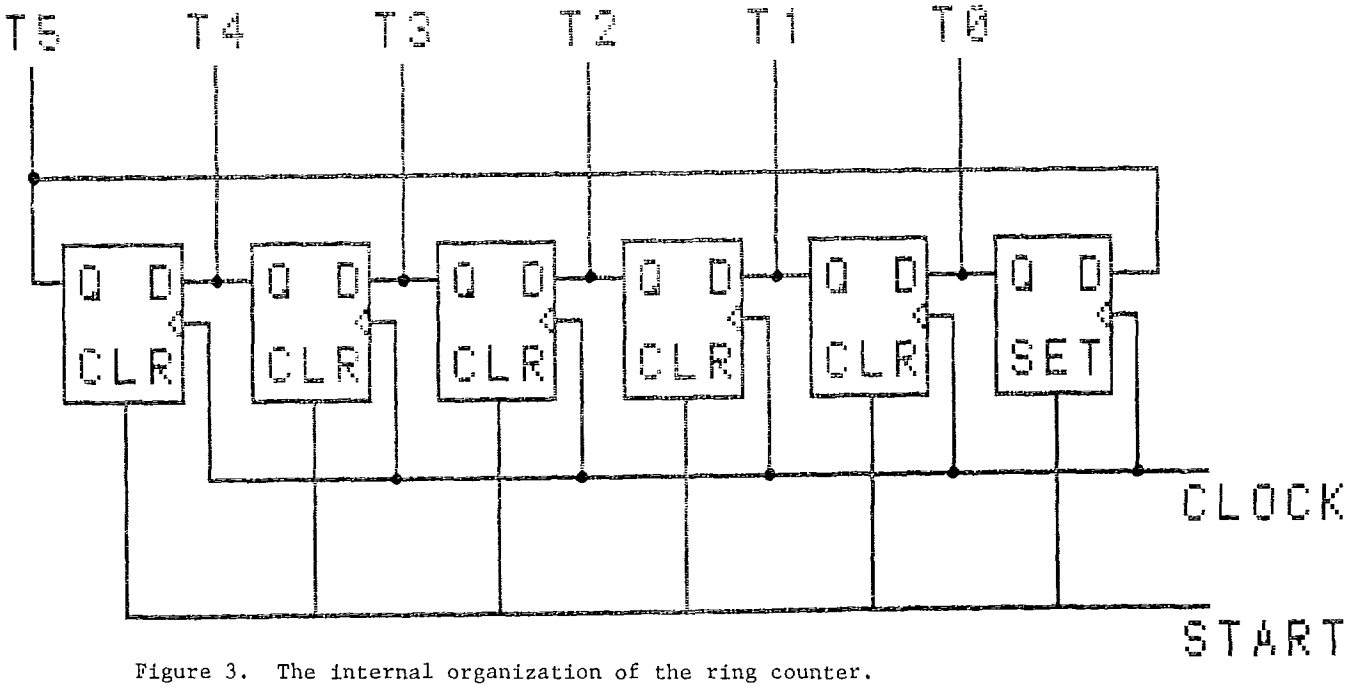

Figure 3. The internal organization of the ring counter.

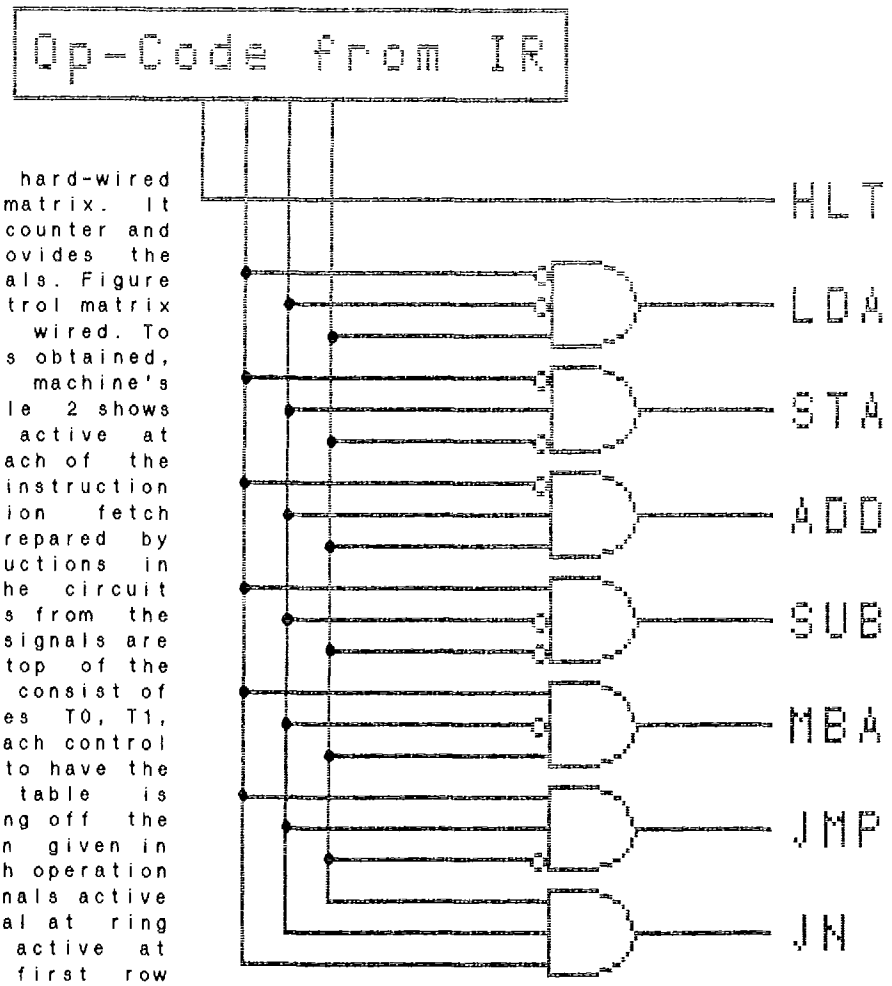

Figure 4. The hard-wired instruction decoder.

The most important part of the hard-wired controller is the control matrix. It receives input from the ring counter and the instruction decoder and provides the proper sequence of control signals. Figure 5 is a diagram of how the control matrix for our simple machine might be wired. To understand how this diagram was obtained, we must look carefully at the machine's instruction set (Table 1). Table 2 shows which control signals must be active at each ring counter pulse for each of the instructions in the computer's instruction set (and for the instruction fetch operation). The table was prepared by simply writing down the instructions in the left-hand column. (In the circuit these will be the output lines from the decoder.) The various control signals are placed horizontally along the top of the table. Entries into the table consist of the moments (ring counter pulses T0, T1, T2, T3, T4, or T5) at which each control signal must be active in order to have the instruction executed. This table is prepared very easily by reading off the information for each instruction given in Table 1. For example, the Fetch operation has the EP and LM control signals active at ring count 0, the R signal at ring count 1, and ED, LI, and IPC active at ring count 2. Therefore the first row (Fetch) of Table 2 has T0 entered below EP and LM, T1 below R, and T2 below IP, ED, and LI.

Once Table 2 has been prepared, the logic required for each control signal is easily obtained. For each an AND operation is performed between any active ring counter (Ti) signals that were entered into the signal's column and the corresponding instruction contained in the far left-hand column. If a column has more than one entry, the output of the ANDs are ORed together to produce the final control signal. For example, the LM column has the following entries: T0 (Fetch), T3 associated with the LDA instruction, and T3 associated with the STA instruction. Therefore, the logic for this signal is:

LM = T0 + T3*LDA + T3*STA

This means that control signal LM will be activated whenever any of the following conditions is satisfied: (1) ring pulse T0 (first step of an instruction fetch) is active; (2) an LDA instruction is in the IR and the ring counter is issuing pulse 3; or (3) an STA instruction is in the IR and the ring counter is issuing pulse 3.

The entries in the JN (Jump Negative) row of this table require some further explanation. The LP and EI signals are active during T3 for this instruction if and only if the accumulator's negative flag has been set. Therefore the entries that appear above these signals for the JN instruction are T3*NF, meaning that the state of the negative flag must be ANDed in for the LP and EI control signals.

Figure 6 gives the logical equations required for each of the control signals used on our machine. These equations have been read from Table 2, as explained above. The circuit diagram of the control matrix (Figure 5) is constructed directly from these equations.

It should be noticed that the HLT line from the instruction decoder does not enter the control matrix. Instead this signal goes directly to circuitry (not shown) that will stop the clock and thus terminate execution.

A Microprogrammed Control Unit

As we have seen, the controller causes instructions to be executed by issuing a specific set of control signals at each beat of the system clock. Each set of control signals issued causes one basic operation (micro-operation), such as a register transfer, to occur within the data path section of the computer. In the case of a hard-wired control unit the control matrix is responsible for sending out the required sequence of signals.

An alternative way of generating the control signals is that of microprogrammed control. In order to understand this method it is convenient to think of sets of control signals that cause specific micro-operations to occur as being "microinstructions" that could be stored in a memory. Each bit of a microinstruction might correspond to one control signal. If the bit is set it means that the control signal will be active; if cleared the signal will be inactive. Sequences of microinstructions could be stored in an internal "control" memory. Execution of a machine language instruction could then be caused by fetching the proper sequence of microinstructions from the control memory and sending them out to the data path section of the computer. A sequence of microinstructions that implements an instruction on the external computer is known as a microroutine. The instruction set of the computer is thus determined by the set of microroutines, the "microprogram," stored in the controller's memory. The control unit of a microprogram-controlled computer is essentially a computer within a computer.

Figure 7 is a block diagram of a microprogrammed control unit that may be used to implement the instruction set of the computer we described above. The heart of this controller is the control ROM memory in which 24-bit long microinstructions are stored. Each is

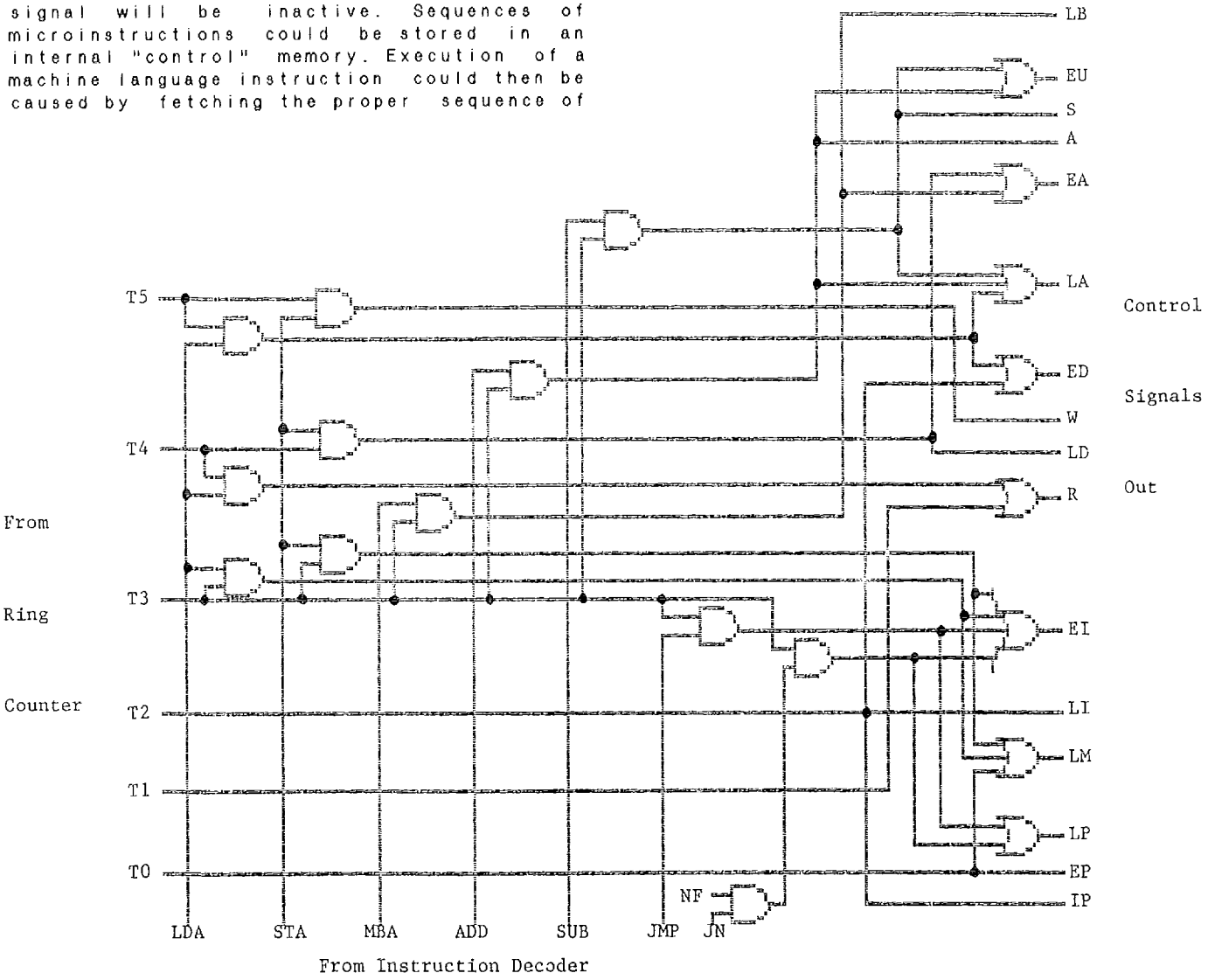

Figure 5. The hard-wired control matrix.

```
IP = T2
LP = T3*JMP + T3*NF*JN
EP = T0
LM = T0 + T3*LDA + T3*STA
R = T1 + T4*LDA
W = T5*STA
LD = T4*STA
ED = T2 + T5*LDA
LI = T2
EI = T3*LDA + T3*STA + T3*JMP + T3*NF*JN
LA = T5*LDA + T3*ADD + T3*SUB
EA = T4*STA + T3*MBA
A = T3*ADD
S = T3*SUB
EU = T3*ADD + T3*SUB
LB = T3*MBA
```

Figure 6. The logical equations required for each of the hardwired control signals on the basic computer. The machine's control matrix is designed from these equations.

composed of two main fields: a 16-bit wide control signal field and an 8-bit wide next-address field. Each bit in the control signal field corresponds to one of the control signals discussed above. The next-address field contains bits that determine the address of the next microinstruction to be fetched from the control ROM. We shall see the details of how these bits work shortly.

Words selected from the control ROM feed the microinstruction register. This 24-bit wide register is analogous to the outer machine's instruction register. Specifically, the leading 16 bits (the control-signal field) of the microinstruction register are connected to the control-signal lines that go to the various components of the external machine's data path section.

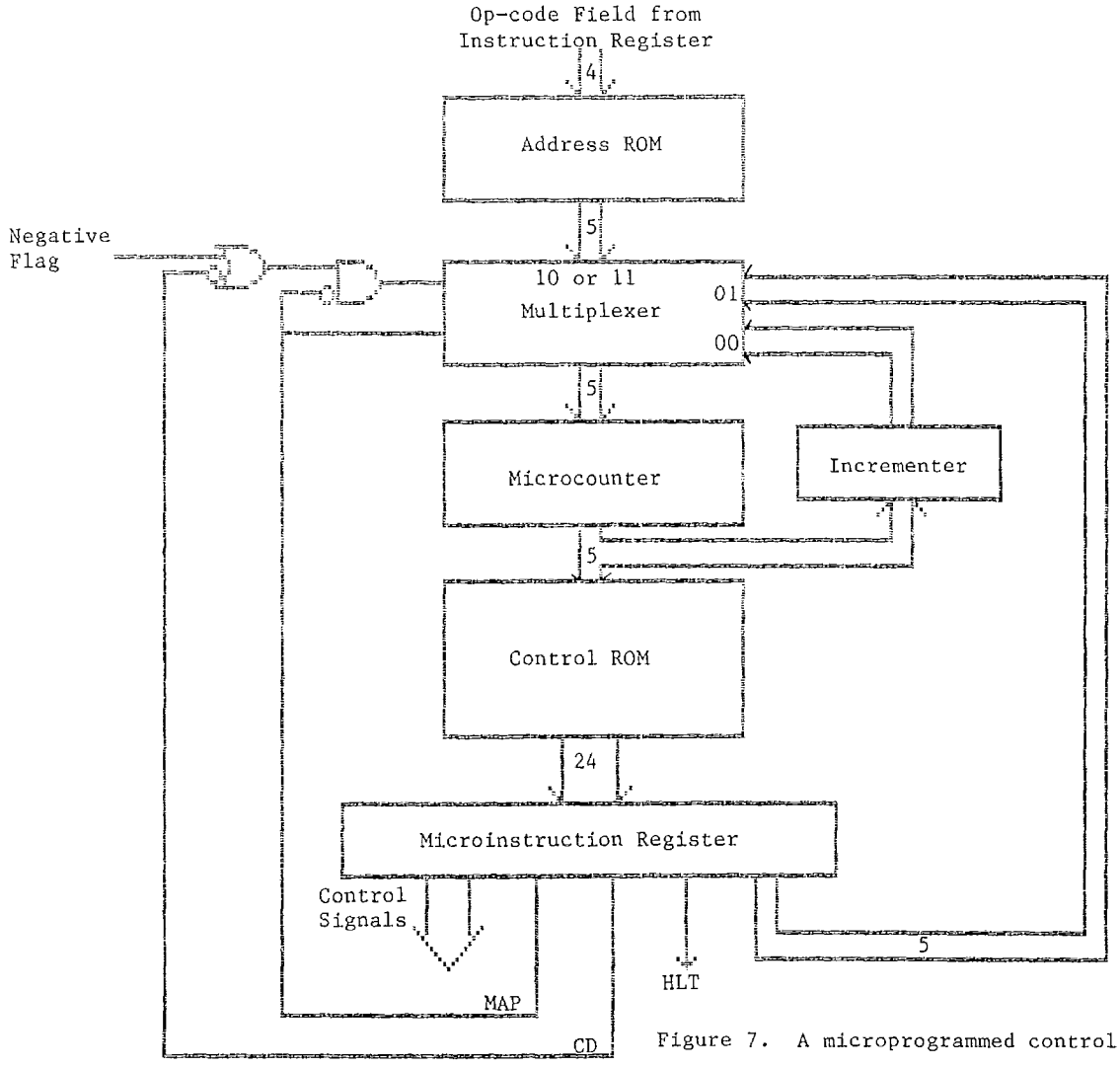

Figure 7. A microprogrammed control unit for the basic computer.

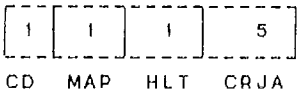

Figure 8. Next Address field of the microinstruction register. CD is the condition bit, MAP causes the address of the next microinstruction to be obtained from the address ROM, HLT stops the clock, and CRJA is the control ROM jump address field.

Addresses provided to the control ROM come from a microcounter register, which is analogous to the external machine's program counter. The microcounter, in turn receives, its input from a multiplexer which selects from: (1) the output of an address ROM, (2) a current-address incrementer, or (3) the address stored in the next-address field of the current microinstruction. The logic that selects one of these three alternatives will be explained shortly.

The controller's address ROM is fed by the outer computer's instruction register. The address ROM maps the op-code of the instruction currently contained in the op-code field of the instruction register to the starting address of the corresponding microroutine in the control ROM. Address zero of the address ROM contains the control-ROM address of the fetch routine; each other address in the address-ROM corresponds to one of the op-codes of the computer's instruction set. Table 3 shows the contents of the address ROM for the instruction set of our simple computer. To see how the address ROM works, let us assume that an ADD instruction has been fetched into the outer computer's instruction register. Since the op-code of the ADD instruction is 3, the number stored at location 3 of the address ROM (a nine) is the starting

TABLE 1

AN INSTRUCTION SET FOR THE BASIC COMPUTER

Instruction Mnemonic	Op-Code	Execution Action	Register Transfers	Active Control Signals
LDA (Load A)	1	A <-- (Mem)	1. MAR <-- IR 2. MDR <-- M(MAR) 3. A <-- MDR	EI, LM R ED, LA
STA (Store A)	2	(Mem) <-- A	1. MAR <-- IR 2. MDR <-- A 3. M(MAR) <-- MDR	EI, LM EA, LD W
ADD (Add B to A)	3	A <-- A + B	1. A <-- ALU(add)	A, EU, LA
SUB (Subtract B from A)	4	A <-- A - B	1. A <-- ALU(sub)	S, EU. LA
MBA (Move A to B)	5	B <-- A	1. B <-- A	EA, LB
JMP (Jump to Address)	6	PC <-- Mem	1. PC <-- IR	EI, LP
JN (Jump if Negative)	7	PC <-- Mem if negative flag is set	1. PC <-- IR if NF set	NF: EI, LP
HLT (Terminate)	8-15	Stop clock		
"Fetch"		IR <-- next instruction	1. PC <-- MAR 2. MDR <-- M(MAR) 3. IR <-- MDR; PC <-- PC + 1	EP, LM R ED, LI, IP

TABLE 2

A MATRIX OF TIMES AT WHICH EACH CONTROL SIGNAL MUST BE ACTIVE IN ORDER TO EXECUTE THE HARD-WIRED BASIC COMPUTER'S INSTRUCTIONS

Control Signal:	IP	LP	EP	LM	R	W	LD	ED	LI	EI	LA	EA	A	S	EU	LB
Instruction																
"Fetch"	T2		T0	T0	T1			T2	T2							
LDA				T3	T4			T5		T3	T5					
STA				T3		T5	T4			T3		T4				
MBA												T3				T3
ADD											T3		T3		T3	
SUB											T3			T3	T3	
JMP		T3								T3						
JN		T3*NF								T3*NF						

address in the control ROM of the microroutine that implements the ADD instruction.

Details of a microinstruction's next address field are shown in Figure 8. The 5-bit CRJA (Control ROM Jump Address) subfield holds a microinstruction address. Thus the address of the next microinstruction may be obtained from the current microinstruction. This permits branching to other sections within the microprogram. The combination of the MAP bit, the CD (condition) bit, and the negative flag from the accumulator of the external machine provide input to the logic that feeds the select lines of the multiplexer and thereby determine how the address of the next microinstruction will be obtained.

If the MAP bit is one, the logic attached to the multiplexer's select lines produces a 10 or a 11 (depending on the value of the CD bit and the negative flag), either of which selects the address ROM. Therefore, the address of the microroutine corresponding to the instruction in the outer machine's instruction register will be channeled to the control ROM. It should be clear that the MAP bit must be set in the last microinstruction of the "fetch" microroutine, since it is at that moment that we want the newly-fetched instruction to be executed.

If the MAP bit is zero and the CD bit is zero, (unconditional branch), the multiplexer logic produces a 01, which selects the CRJA field of the current

TABLE 3

THE BASIC COMPUTER'S ADDRESS ROM

Instruction Mnemonic	Address-ROM Address (Instruction Op-Code)	Address-ROM Contents (Control-ROM Micro-Routine Start Address)
"Fetch"	0	00
LDA	1	03
STA	2	06
ADD	3	09
SUB	4	0A
MBA	5	0B
JMP	6	0C
JN	7	0D
Avaible for New Instructions	8-E	10-1E
HLT	F	1F

instruction. Therefore the next instruction will come from the address contained in the current instruction's next-address field. With MAP=0 and CD=1 (conditional branch), the logic that feeds the multiplexer will produce either a 00 or a 01, depending on the value of the negative flag. If the flag is set, it is a 01, which selects the jump address contained in the current microinstruction. If the negative flag is cleared, the select lines to the multiplexer receive a 00, which causes the incrementer to be selected. The next microinstruction will come from the next address in sequence. It should be noticed that with this scheme, if we are not doing branching, the CRJA field should contain the address of the next microinstruction and the CD bit should be cleared. This will cause a "branch to the next microinstruction" to occur. The one exception to this rule is the case of the last microinstruction within a microroutine. Normally we would then want to branch back to the "fetch" microroutine. Since this routine starts at control-ROM location 00000, that address should be contained in the CRJA field and CD should be 0.

The HLT bit is used to terminate execution. If it is set, the clock that synchronizes activities within the entire machine is stopped.

Notice that the microcounter is triggered by a rising clock edge, and the microinstruction register by a falling edge. Thus we see that on each positive edge the microcounter receives the address of the next microinstruction and presents it to the control ROM, which has until the next negative edge to output the addressed control word to the microinstruction register. Since all operations in the data path section are positive-edge triggered, there is adequate time for the signals specified in the control word contained in the microinstruction register to go out to all sections of the external machine. The sequence of latching the address of microinstruction i+1 into the microcounter while microinstruction i executes (positive edge) and then presenting the control word of microinstruction i+1 to the microinstruction register (negative edge) continues until a set HLT bit stops the clock.

Table 4 shows a microprogram which, when loaded into the control ROM, will implement the instruction set of the computer we have been describing. For each microinstruction the control ROM address has been expressed in hexadecimal, and the contents in binary. The order of the bits in the control signal field is the same as that shown in Table 2: IP, LP, EP, LM, R, W, LD, ED, LI, EI, LA, EA, A, S, EU, LB, reading from left to right. The last four columns of Table 4 express the status of the CD, MAP, and HLT bits, and the Control ROM Jump Address, expressed in hexadecimal. In order to clarify how the

TABLE 4

A MICROPROGRAM THAT IMPLEMENTS THE BASIC COMPUTER'S INSTRUCTION SET

Microroutine Name (Mnemonic)	Address-ROM Address (Op-code)	Micro-Instruction Address	Control Signals: ILELRWLELELEASEL PPPM DDIIAA UB	CD bit	MAP bit	HLT bit	Address of Next Micro-Instruction
"Fetch"	0	00	0011000000000000	0	0	0	01
		01	0000100000000000	0	0	0	02
		02	1000000110000000	0	1	0	XX
LDA	1	03	0001000001000000	0	0	0	04
		04	0000100000000000	0	0	0	05
		05	0000000100100000	0	0	0	00
STA	2	06	0001000001000000	0	0	0	07
		07	0000001000010000	0	0	0	08
		08	0000010000000000	0	0	0	00
ADD	3	09	0000000000101010	0	0	0	00
SUB	4	0A	0000000000100110	0	0	0	00
MBA	5	0B	0000000000010001	0	0	0	00
JMP	6	0C	0100000001000000	0	0	0	00
JN	7	0D	0000000000000000	1	0	0	0F
		0E	0000000000000000	0	0	0	00
		0F	0100000001000000	0	0	0	00
Available for Expansion	8-E	10-1E					
HLT	F	1F	0000000000000000	0	0	1	XX

microprogram works, a description is now given of the "fetch" and JN (jump on negative) microroutines.

The "fetch" microroutine occupies control ROM addresses 0, 1, and 2. The active EP and LM control-signal bits in its first microinstruction cause a register transfer from the program counter to the memory address register to occur. The MAR will now contain the address in RAM of the next instruction. Since CD and MAP are both zero (unconditional branch), the next microinstruction will come from the address stored in the CRJA field (01) -- the next consecutive location. The microinstruction stored at that location has only the R bit active. Thus the word stored in the memory location being accessed by the MAR (presumably the next instruction) will be gated to the Memory Data Register (MDR). The zeroes in CD and MAP again cause the next microinstruction to be fetched from the address specified in the CRJA field, a 02 here. Active control signal bits for that microinstruction are ED, LI, and IP. The first two transfer the word in the MDR to the Instruction Register, and the last increments the program counter. The new instruction is safely in the IR, and the PC is pointing to the next instruction in sequence. We have completed an instruction fetch. Since the MAP field in the last microinstruction of this "fetch" microroutine is equal to 1, the address of the next micrpinstruction is determined by the address ROM, which, in turn, depends upon the opcode of the instruction that has just been placed in the instruction register.

When the JN instruction is executed, control is supposed to be transferred to the address specified by the least significant eight bits of the number contained in the instruction register if the negative flag is set. If the negative flag is not set, execution should continue with the next instruction in sequence. Let us see how the microroutine stored at control-ROM locations 0D, 0E, and 0F implement this conditional jump. In the first microinstruction none of the control signal bits is set. Thus nothing will occur in the data path section of the computer. However, the fact that the CD bit is set means that IF THE NEGATIVE FLAG IS SET, the next microinstruction will be fetched from the control-ROM address specified in the CRJA field (a 0F in this case). The microinstruction stored at that location has the EI and LP control signal bits set. Thus the contents of the instruction register (the least significant eight bits) will be transferred to the program counter. The zeroes stored in the CD and MAP bits cause the next microinstruction to be fetched from the address contained in the CRJA field--a 00 in this case. This is the start of the "fetch" microroutine. Thus we see that if the negative flag is set, the JN microroutine places the jump address in the program counter and transfers to the fetch routine. When that fetch is performed, control will have been transferred to the jump address.

If, on the otherhand, the negative flag is NOT SET when the JN microroutine executes, then the set CD bit in its first microinstruction causes the current address stored in the microcounter to be incremented. Thus the next microinstruction would be fetched from location 0E. That microinstruction also has no active control signal bits, but with CD=0 and CRJA=00, the next microinstruction will be the first one in the "fetch" routine. Notice that in this case, the JN instruction simply returns us to the next fetch. Since the program counter has not been altered, that fetch will be from the next sequential memory location, as usual.

Hardwired vs. Microprogrammed Computers

It should be mentioned that most computers today are microprogrammed. The reason is basically one of flexibility. Once the control unit of a hard-wired computer is designed and built, it is virtually impossible to alter its architecture and instruction set. In the case of a microprogrammed computer, however, we can change the computer's instruction set simply by altering the microprogram stored in its control memory. In fact, taking our basic computer as an example, we notice that its four-bit op-code permits up to 16 instructions. Therefore we could add seven more instructions to the instruction set by simply expanding its microprogram. To do this with the hard-wired version of our computer would require a complete redesign of the controller circuitry.

Another advantage to using micprogrammed control is the fact that the task of designing the computer in the first place is simplified. The process of specifying the architecture and instruction set is now one of software (microprogramming) as opposed to hardware design. Nevertheless, for certain applications hard-wired computers are still used. If speed is a consideration, hard-wiring may be required since it is faster to have the hardware issue the required control signals than to have a "program" do it.

Summary

The intent of this article has been to present, through example, the distinction between hardwired and microprogrammed computers. In the process it is hoped that the reader has gained insight into what really occurs inside a digital computer as a program executes.