UNIT II
BASIC COMPUTER ORGANIZATION AND DESIGN

Instruction Codes

The organization of the computer is defined by its internal registers, the timing and control structure, and the set of instructions that it uses. The design of the computer is then carried out in detail. The internal organization of a digital system is defined by the sequence of micro operation it performs on data stored in its registers. The general-purpose digital computer is capable of executing various micro operations and, in addition, can be instructed as to what specific sequence of operations it must perform. The user of a computer can control the process by means of a program. A program is a set of instructions that specify the operations, operands, and the sequence by which processing has to occur. The data- processing task may be altered by specifying a new program with different instructions or specifying the same instructions with different data.

A computer instruction is a binary code that specifies a sequence of micro operations for the computer. Instruction codes together with data are stored in memory. The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro operations. Every computer has its own unique instruction set. The ability to store and execute instructions, the stored program concept, is the most important property of a general- purpose computer.

An instruction code is a group of bits that instruct the computer to perform a specific operation. It is usually divided into parts, each having its own particular interpretation. The most basic part of an instruction code is its operation part. The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement. The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.

The operation code must consist of at least n bits for a given 2^n (or less) distinct operations. As an illustration, consider a computer with 64 distinct operations, one of them being an ADD operation. The operation code consists of six bits, with a bit configuration 110010 assigned to the ADD operation. When this operation code is decoded in the control unit, the computer issues control signals to read an operand from memory and add the operand to a processor register.

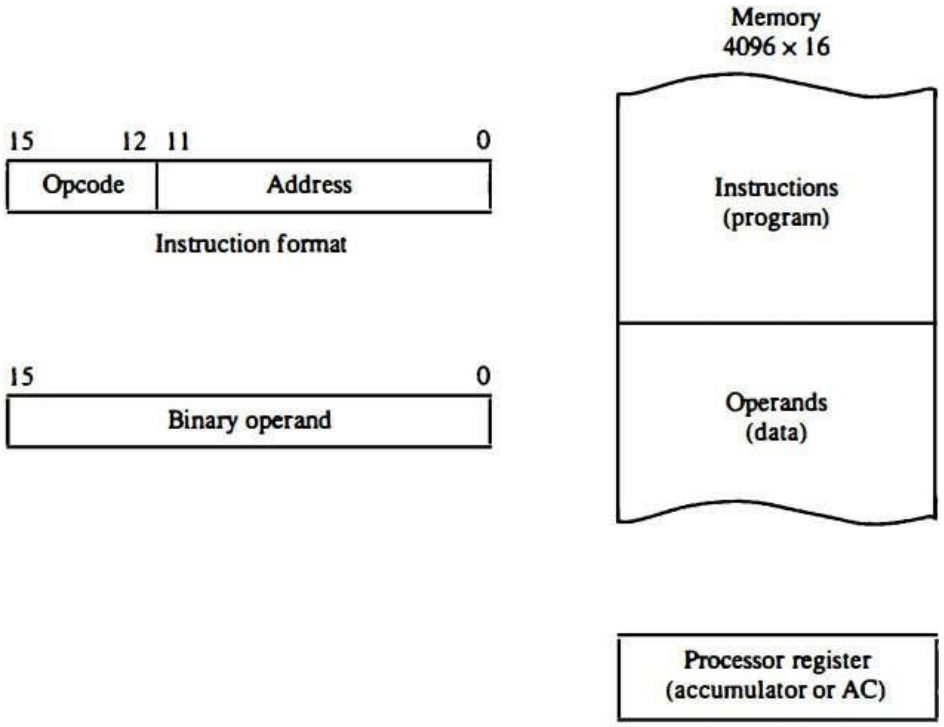
At this point we must recognize the relationship between a computer operation and a micro operation. An operation is part of an instruction stored in computer memory. It is a binary code that tells the computer to perform a specific operation. The control unit receives the instruction from memory and interprets the operation code bits. It then issues a sequence of control signals to initiate micro operations in internal computer registers. For every operation code, the control issues a sequence of micro operations needed for the hardware implementation of the specified operation. For this reason, an operation code is sometimes called a macro operation because it specifies a set of micro operations. The operation part of an instruction code specifies the operation to be performed. This operation must be performed on some data stored in processor registers or in memory. An instruction code must therefore specify not only the operation but also the registers or the memory words where the operands

are to be found, as well as the register or memory word where the result is to be stored. Memory words can be specified in instruction codes by their address. Processor registers can be specified by assigning to the instruction another binary code of k bits that specifies one of 2* registers. There are many variations for arranging the binary code of instructions, and each computer has its own particular instruction code format. Instruction code formats are conceived by computer designers who specify the architecture of the computer.

Stored Program Organization

The simplest way to organize a computer is to have one processor register and an instruction code format with two parts. The first part specifies the operation to be performed and the second specifies an address. The memory address tells the control where to find an operand in memory. This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.

Figure depicts this type of organization. Instructions are stored in one section of memory and data in another. For a memory unit with 4096 words we need 12 bits to specify an address since $2^{12} = 4096$. If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand. The control reads a 16-bit instruction from the program portion of memory. It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory. It then executes the operation specified by the operation code.



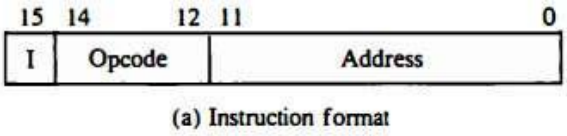
Computers that have a single-processor register usually assign to it the name accumulator and label it AC. The operation is performed with the memory operand and the content of AC . If an operation in an instruction code does not need an operand from memory, the rest of the bits in

the instruction can be used for other purposes. For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register. They do not need an operand from memory. For these types of operations, the second part of the instruction code (bits 0 through 11) is not needed for specifying a memory address and can be used to specify other operations for the computer.

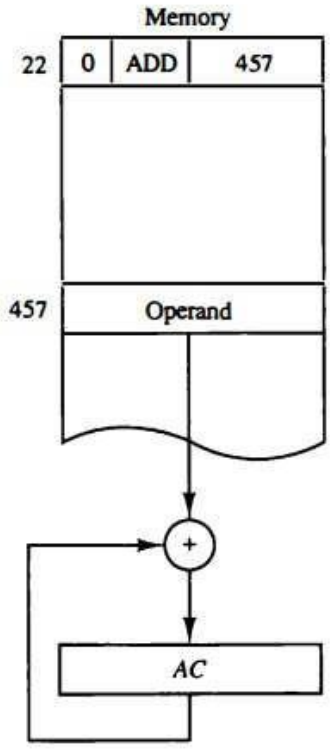
Indirect Address

It is sometimes convenient to use the address bits of an instruction code not as an address but as the actual operand. When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand. When the second part specifies the address of an operand, the instruction is said to have a direct address. This is in contrast to a third possibility called indirect address, where the bits in the second part of the instruction designate an address of a memory word in which the address of. the operand is found. One bit of the instruction code can be used to distinguish between a direct and an indirect address.

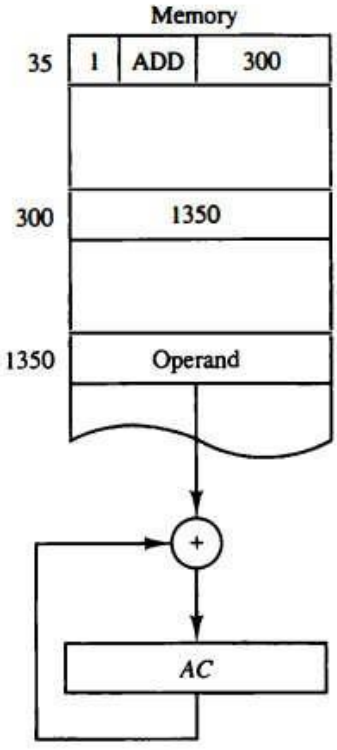
As an illustration of this configuration, consider the instruction code format shown in Figure (a). It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I. The mode bit is 0 for a direct address and 1 for an indirect address. A direct address instruction is shown in Figure (b). It is placed in address 22 in memory.



(a) Instruction format



(b) Direct address



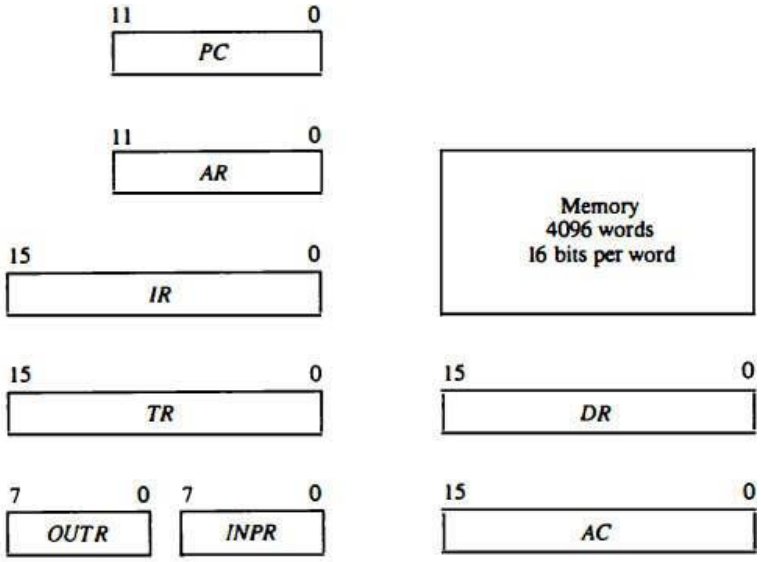
(c) Indirect address

The I bit is 0, so the instruction is recognized as a direct address instruction. The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457. The control finds the operand in memory at address 457 and adds it to the content of AC . The instruction in address 35 shown in Figure (c) has a mode bit I=1. Therefore, it is recognized as an indirect address instruction. The address part is the binary equivalent of 300. The control goes to address 300 to find the address of the operand. The address of the operand in this case is 1350. The operand found in address 1350 is then added to the content of AC. The indirect address instruction needs two references to memory to fetch an operand. The first reference is needed to read the address of the operand; the second is for the operand itself. We define the effective address to be the address of the operand in a computation-type instruction or the target address in a branch-type instruction. Thus the effective address in the instruction of Figure (b) is 457 and in the instruction of Figure (c) is 1350.

The direct and indirect addressing modes are used in the computer presented in this chapter. The memory word that holds the address of the operand in an indirect address instruction is used as a pointer to an array of data. The pointer could be placed in a processor register instead of memory as done in commercial computers.

Computer Registers

Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time. The control reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it, and so on. This type of instruction sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed. It is also necessary to provide a register in the control unit for storing the instruction code after it is read from memory. The computer needs processor registers for manipulating data and a register for holding a memory address. These requirements dictate the register configuration shown in Figure.



The registers are also listed in Table together with a brief description of their function and the number of bits that they contain. The memory unit has a capacity of 4096 words and each word contains 16 bits. Twelve bits of an instruction word are needed to specify the address of an operand. This leaves three bits for the operation part of the instruction and a bit to specify a direct or indirect address. The data register (DR) holds the operand read from memory. The accumulator (AC) register is a general-purpose processing register. The instruction read from memory is placed in the instruction register (IR). The temporary register (TR) is used for holding temporary data during the processing.

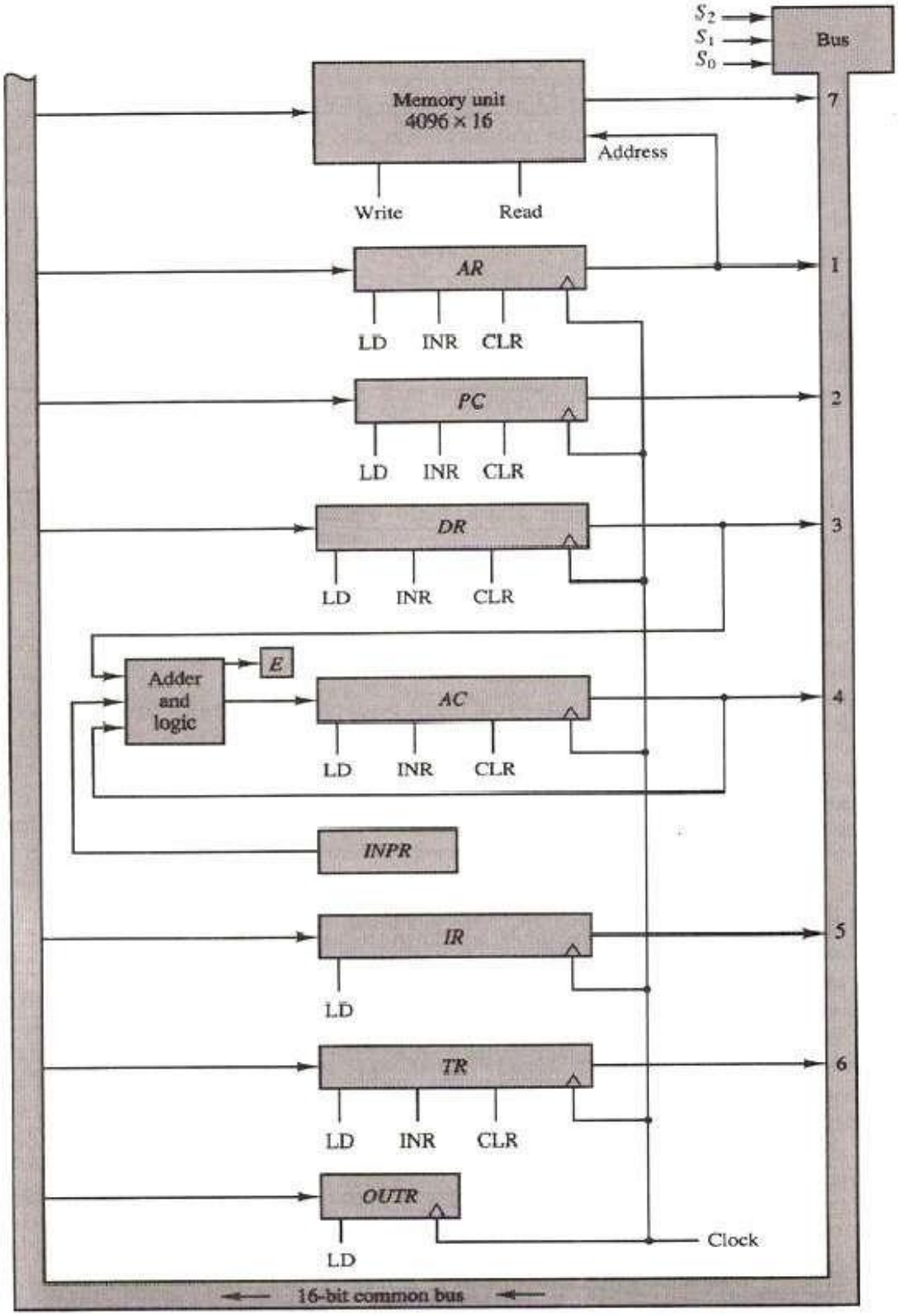
Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

The memory address register (AR) has 12 bits since this is the width of a memory address. The program counter (PC) also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed. The PC goes through a counting sequence and causes the computer to read sequential instructions previously stored in memory. Instruction words are read and executed in sequence unless a branch instruction is encountered. A branch instruction calls for a transfer to a nonconsecutive instruction in the program. The address part of a branch instruction is transferred to PC to become the address of the next instruction. To read an instruction, the content of PC is taken as the address for memory and a memory read cycle is initiated. PC is then incremented by one, so it holds the address of the next instruction in sequence. Two registers are used for input and output. The input register (IN PR) receives an 8-bit character from an input device. The output register (OUTR) holds an 8-bit character for an output device.

Common Bus System

The basic computer has eight registers, a memory unit, and a control unit. Paths must be provided to transfer information from one register to another and between memory and registers. The number of wires will be excessive if connections are made between the outputs of each register and the inputs of the other registers. A more efficient scheme for transferring information in a system with many registers is to use a common bus. The connection of the registers and memory of the basic computer to a common bus system is shown in Figure. The outputs of seven registers and memory are connected to the common bus. The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S2 , S1, and S0 . The number along each output shows the decimal equivalent of the required binary selection.

Unit II


S_2
S_1
S_0
Bus
Memory unit
4096 × 16
Address
Write
Read
7
AR
LD INR CLR
1
PC
LD INR CLR
2
DR
LD INR CLR
3
Adder and logic
E
AC
LD INR CLR
4
INPR
IR
LD
5
TR
LD INR CLR
6
OUTR
LD
Clock
16-bit common bus

For example, the number along the output of DR is 3. The 16-bit outputs of DR are placed on the bus lines when S2S1S0 = 011 since this is the binary value of decimal 3. The lines from the common bus are connected to the inputs of each register and the data inputs of the memory. The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition. The memory receives the contents of the bus when its write input is activated. The memory places its 16-bit output onto the bus when the read input is activated and S2S1S0 =111. Four registers, DR, AC, IR, and TR, have 16 bits each. Two registers, AR memory address and PC, have 12 bits each since they hold a memory address. When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to 0's. When AR or PC receive information from the bus, only the 12 least significant bits are transferred into the register. The input register INPR and the output register OUTR have 8 bits each and communicate with the eight least significant bits in the bus. 1NPR is connected to provide information to the bus but OUTR can only receive information from the bus. This is because INPR receives a character from an input device which is then transferred to AC. OUTR receives a character from AC and delivers it to an output device. There is no transfer from OUTR to any of the other registers. The 16 lines of the common bus receive information from six registers and the memory unit. The bus lines are connected to the inputs of six registers and the memory. Five registers have three control inputs: LD (load), INR (increment), and CLR (clear). This type of register is equivalent to a binary counter with parallel load and synchronous clear.

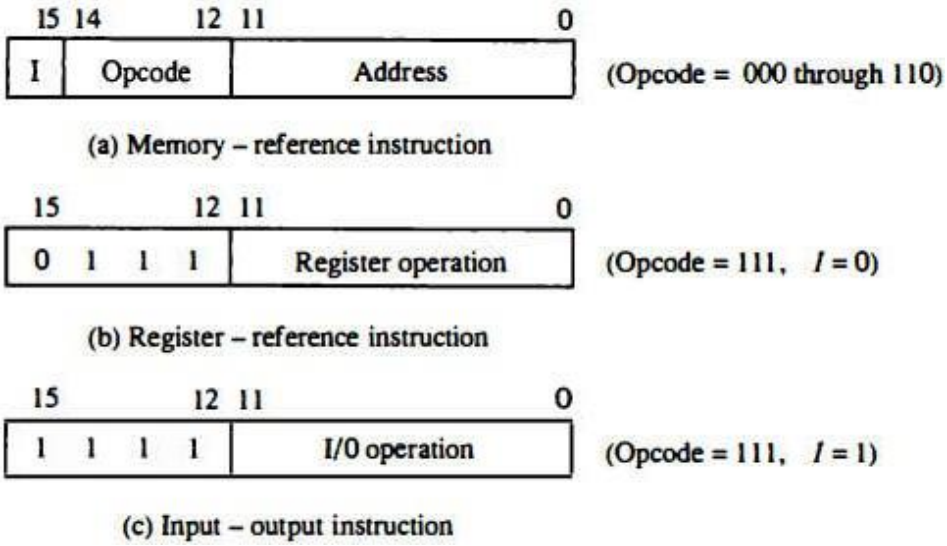
The increment operation is achieved by enabling the count input of the counter. Two registers have only a LD input. The input data and output data of the memory are connected to the common bus, but the memory address is connected to AR . Therefore, AR must always be used to specify a memory address. By using a single register for the address, we eliminate the need for an address bus that would have been needed otherwise. The content of any register can be specified for the memory data input during a write operation. Similarly, any register can receive the data from memory after a read operation except AC.

The 16 inputs of AC come from an adder and logic circuit. This circuit has three sets of inputs. One set of 16-bit inputs come from the outputs of AC . They are used to implement register micro operations such as complement AC and shift AC. Another set of 16-bit inputs come from the data register DR. The inputs from DR and AC are used for arithmetic and logic micro operations, such as add DR to AC or AND DR to AC. The result of an addition is transferred to AC and the end carry-out of the addition is transferred to flip-flop E (extended AC bit). A third set of 8-bit inputs come from the input register INPR. Note that the content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle. The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into AC.

Computer Instructions

The basic computer has three instruction code formats, as shown in Figure. Each format has 16 bits. The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered. A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode 1. 1 is equal to 0 for direct address and to 1 for indirect address. The register-reference instructions are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction. A

register-reference instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed. Similarly, an input-output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.



The type of instruction is recognized by the computer control from the four bits in positions 12 through 15 of the instruction. If the three opcode bits in positions 12 through 14 are not equal to 1 11, the instruction is a memory-reference type and the bit in position 15 is taken as the addressing mode I. If the 3-bit opcode is equal to 111, control then inspects the bit in position 15. If this bit is 0, the instruction is a register-reference type. If the bit is 1, the instruction is an input-output type. Note that the bit in position 15 of the instruction code is designated by the symbol I but is not used as a mode bit when the operation code is equal to 111. Only three bits of the instruction are used for the operation code. It may seem that the computer is restricted to a maximum of eight distinct operations. However, since register-reference and input-output instructions use the remaining 12 bits as part of the operation code, the total number of instructions can exceed eight. In fact, the total number of instructions chosen for the basic computer is equal to 25. The instructions for the computer are listed in Table 5-2. The symbol designation is a three-letter word and represents an abbreviation intended for programmers and users. The hexadecimal code is equal to the equivalent hexa-decimal number of the binary code used for the instruction. By using the hexadecimal equivalent we reduced the 16 bits of an instruction code to four digits with each hexadecimal digit being equivalent to four bits. A memory-reference instruction has an address part of 12 bits. The address part is denoted by three x's and stand for the three hexadecimal digits corresponding to the 12-bit address. The last bit of the instruction is designated by the symbol I. When 1 = 0, the last four bits of an instruction have a hexadecimal digit equivalent from 0 to 6 since the last bit is 0. When 1 = 1, the hexadecimal digit equivalent of the last four bits of the instruction ranges from 8 to E since the last bit is 1. Register-reference instructions use 16 bits to specify an operation. The leftmost four bits are always 0111, which is equivalent to hexadecimal 7.

Symbol	Hexadecimal code $I = 0$	Hexadecimal code $I = 1$	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

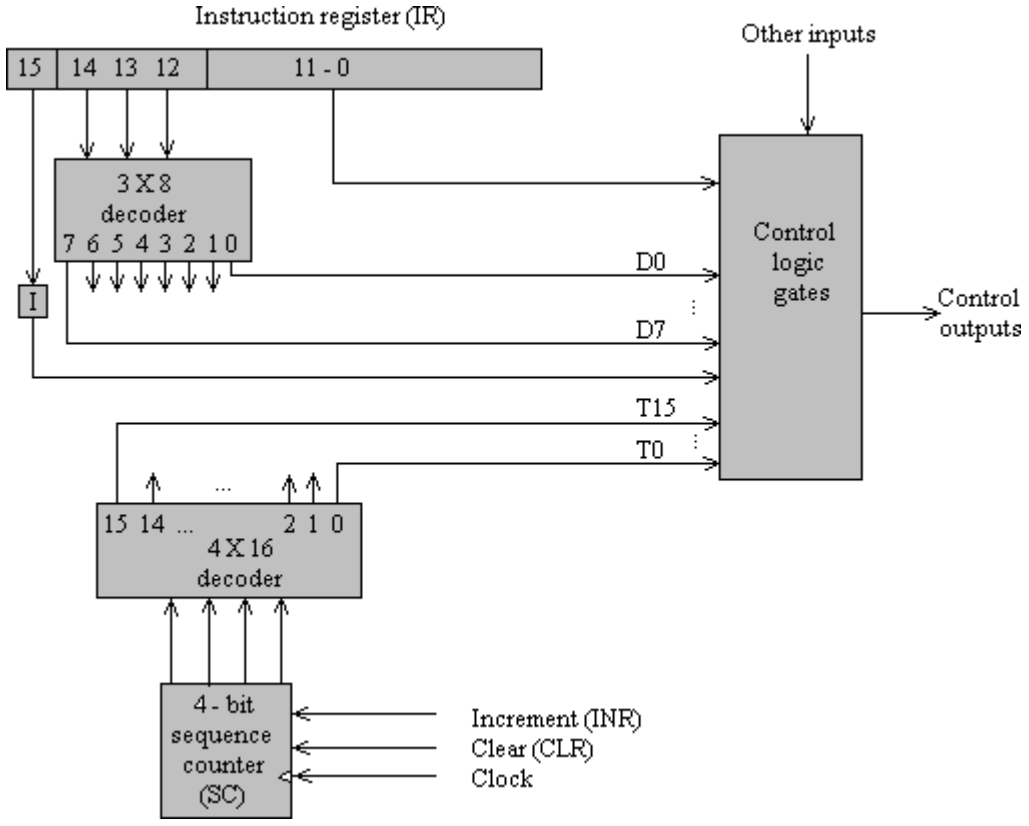
The other three hexadecimal digits give the binary equivalent of the remaining 12 bits. The input-output instructions also use all 16 bits to specify an operation. The last four bits are always 1111, equivalent to hexadecimal F.

Timing and Control

The timing for all registers in the basic computer is controlled by a master clock generator. The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit. The clock pulses do not change the state of a register unless the register is enabled by a control signal. The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and micro operations for the accumulator. There are two major types of control organization: hardwired control and micro programmed control. In the hardwired organization,

the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation. In the micro programmed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro operations. A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed. In the micro programmed control, any required changes or modifications can be done by updating the micro program in control memory. A hardwired control for the basic computer is presented in this section.

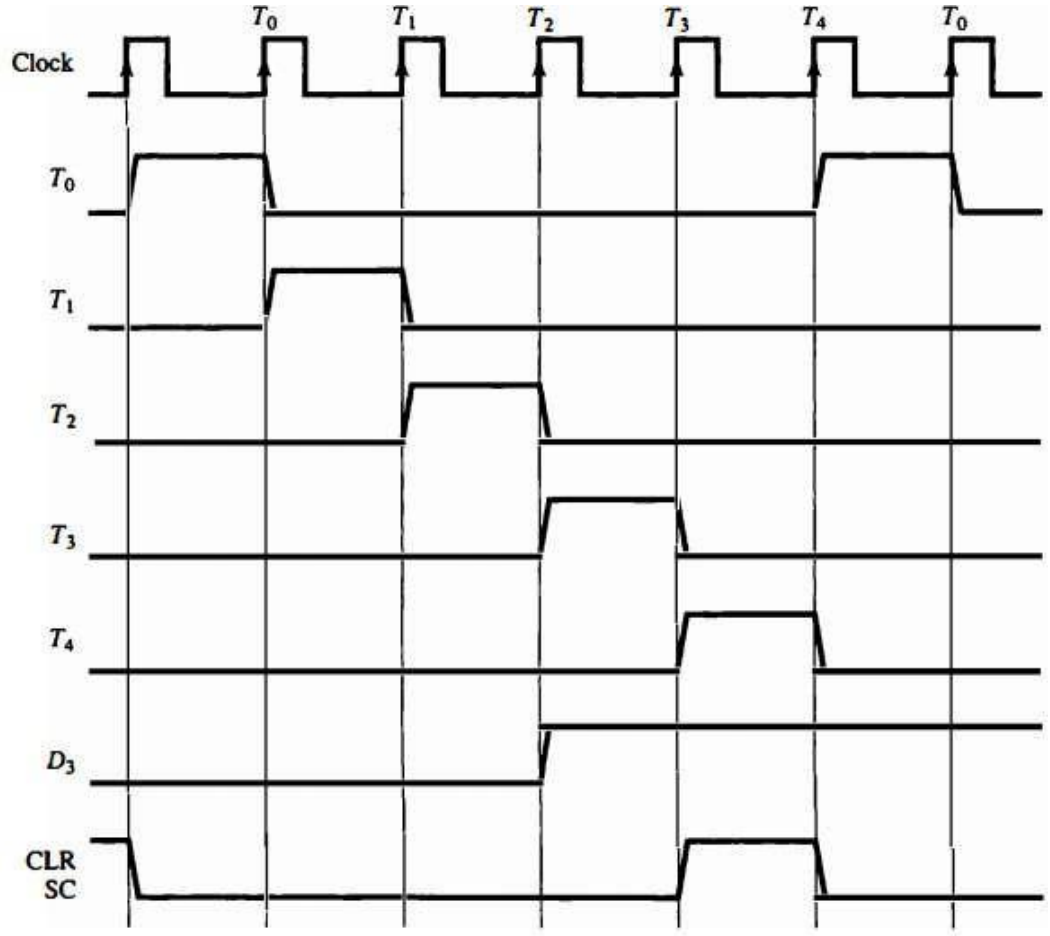
The block diagram of the control unit is shown in Figure. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register (IR). The instruction register is shown again in Figure, where it is divided into three parts: the I bit, the operation code, and bits 0 through 11. The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D0 through D7 . The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T0 through T15.



The sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in awhile, the counter is cleared to 0, causing the next active timing signal to be T0 . As an

example, consider the case where SC is incremented to provide timing signals T0, T2, T3, and T4 in sequence. At time T4 , SC is cleared to 0 if decoder output D3 is active.

The timing diagram shows the time relationship of the control signals. The sequence counter SC responds to the positive transition of the clock. Initially, the CLR input of SC is active. The first positive transition of the clock dears SC to 0, which in turn activates the timing signal To out of the decoder. To is active during one clock cycle. To in the diagram will trigger only those registers whose control inputs are connected to timing signal To. SC is incremented with every positive clock transition, unless its CLR input is active. This produces the sequence of timing signals To, T1, T2, T3, T4 and so on, as shown in the diagram, If SC is not cleared, the timing signals will continue with T5, T6, upto T15 and back to To.



The last three waveforms in Figure show how SC is cleared when D3 T4 = 1. Output D3 from the operation decoder becomes active at the end of timing signal T2 . When timing signal T4 becomes active, the output of the AND gate that implements the control function D3 T4 becomes active. This signal is applied to the CLR input of SC. On the next positive clock transition (the one marked T 4 in the diagram) the counter is cleared to 0. This causes the timing signal To to become active instead of T5 that would have been active if SC were incremented instead of cleared.

Instruction Cycle

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases. In the basic computer each instruction cycle consists of the following phases:

1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Fetch and Decode

Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal T 0 . After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T 0 , T i, T 2 , and so on. The micro operations for the fetch and decode phases can be specified by the following register transfer statements.

T0: AR ← PC
T1: IR ← M[AR], PC ← PC + 1
T 2 : D0 , . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)

Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T0. The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1. At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program. At time T2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR. Note that SC is incremented after each clock pulse to produce the sequence T0, Tl and T2. Figure shows how the first two register transfer statements are implemented in the bus system. To provide the data path for the transfer of PC to AR we must apply timing signal T0 to achieve the following connection:

1. Place the content of PC onto the bus by making the bus selection inputs S2S1S0 equal to 010.
2. Transfer the content of the bus to AR by enabling the LD input of AR.

The next clock transition initiates the transfer from PC to AR since T0 = 1. In order to implement the second statement

T1: IR ← M[AR], PC ← PC + 1

it is necessary to use timing signal T1 to provide the following connections in the bus system.

1. Enable the read input of memory.
2. Place the content of memory onto the bus by making S2S1S0 = 111.

3. Transfer the content of the bus to IR by enabling the LD input of IR.
4. Increment PC by enabling the INR input of PC.

The next clock transition initiates the read and increment operations since T1 = 1 .

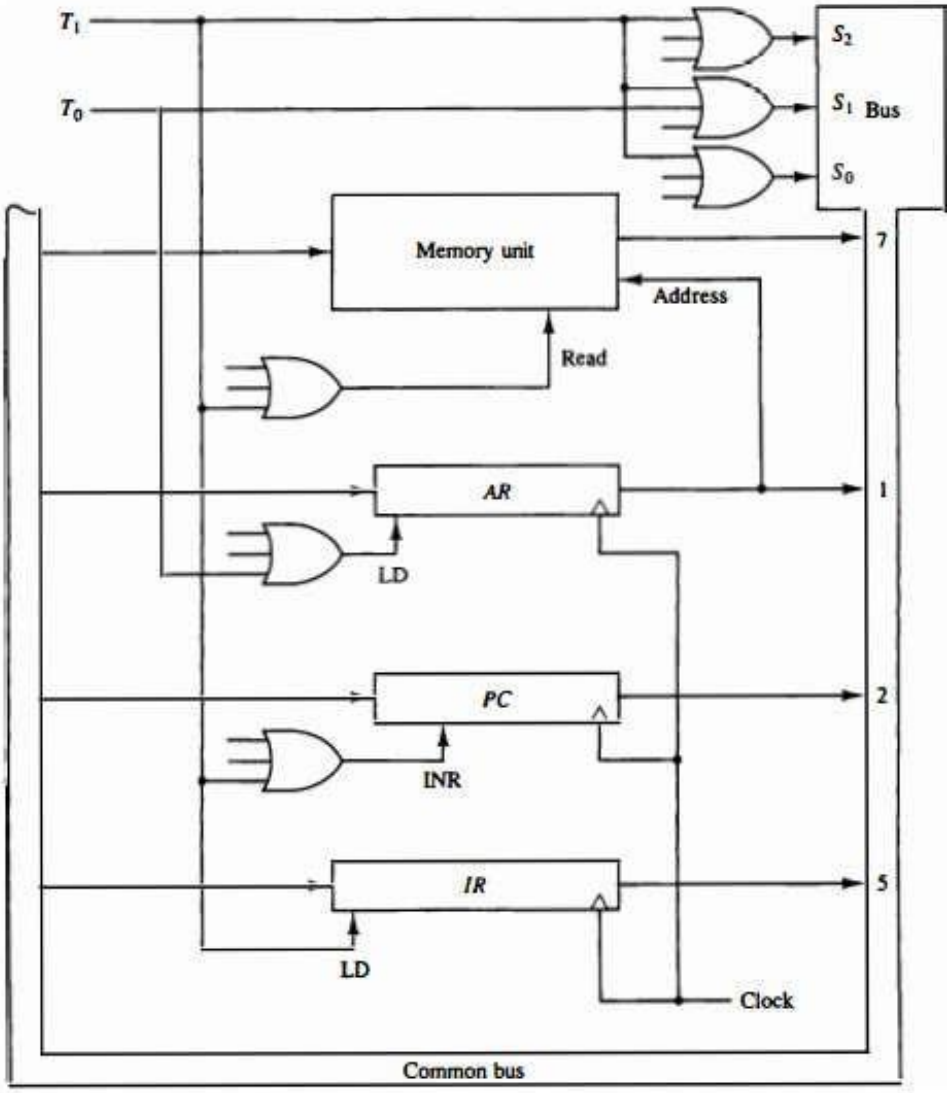
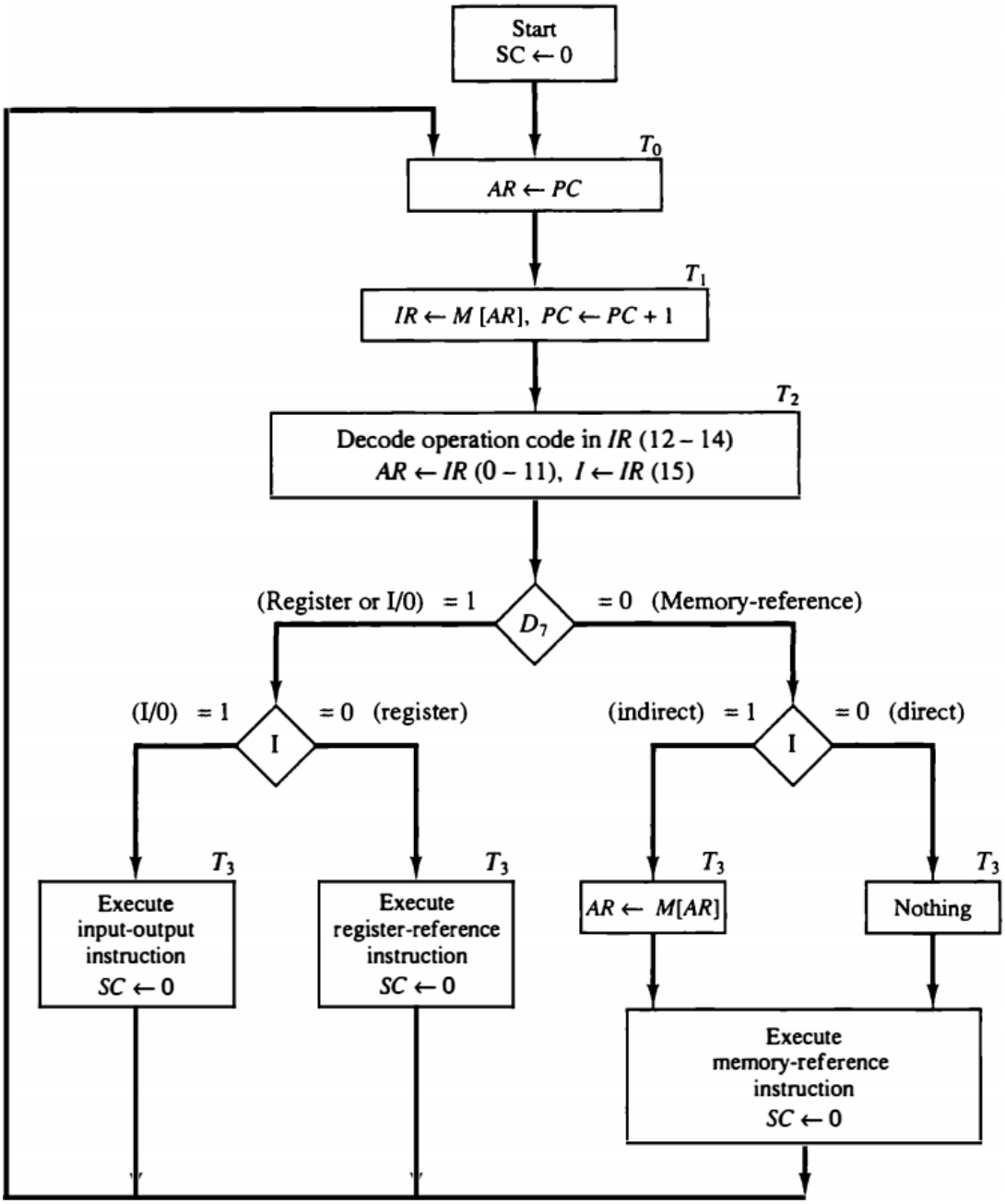


Figure duplicates a portion of the bus system and shows how To and T1 are connected to the control inputs of the registers, the memory, and the bus selection inputs. Multiple input OR gates are included in the diagram because there are other control functions that will initiate similar operations.

Determine the Type of Instruction

The timing signal that is active after the decoding is T3 . During time T3, the control unit determines the type of instruction that was just read from memory. The flowchart presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding. Decoder output D7 is equal to 1 if the operation code is equal to binary 111. If D7 = 1, the instruction must be a register-reference or input-output type. If D7 = 0, the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.



Control then inspects the value of the first bit of the instruction, which is now available in flip-flop I. If D7 = 0 and I = 1, we have a memory-reference instruction with an indirect address. It is then necessary to read the effective address from memory. The micro operation for the indirect address condition can be symbolized by the register transfer statement AR ← M[AR]. Initially, AR holds the address part of the instruction. This address is used during the memory read operation. The word at the address given by AR is read from memory and placed on the common bus. The LD input of AR is then enabled to receive the indirect address that resided in the 12 least significant bits of the memory word. The three instruction types are subdivided into

four separate paths. The selected operation is activated with the clock transition associated with timing signal T3. This can be symbolized as follows:

D7'IT3 : AR ← M[AR]
D7'I'T3 : Nothing
D7I'T3 : Execute a register-reference instruction
D71T3 : Execute an input-output instruction

When a memory-reference instruction with I = 0 is encountered, it is not necessary to do anything since the effective address is already in AR. However, the sequence counter SC must be incremented when D7'T 3 = 1, so that the execution of the memory-reference instruction can be continued with timing variable T4 . A register-reference or input-output instruction can be executed with the clock associated with timing signal T3 . After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with T0 = 1. Note that the sequence counter SC is either incremented or cleared to 0 with every positive clock transition. We will adopt the convention that if SC is incremented, we will not write the statement SC ← SC + 1, but it will be implied that the control goes to the next timing signal in sequence. When SC is to be cleared, we will include the statement SC ← 0.

Register-Reference Instructions

Register-reference instructions are recognized by the control when D7 = 1 and I = 0. These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions. These 12 bits are available in IR(0-11). They were also transferred to AR during time T2. The control functions and micro operations for the register-reference instructions are listed in Table. These instructions are executed with the clock transition associated with timing variable T3. Each control function needs the Boolean relation D7I'T3, which we designate for convenience by the symbol r. The control function is distinguished by one of the bits in IR{ 0-11). By assigning the symbol B, to bit i of IR, all control functions can be simply denoted by rB.

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

For example, the instruction CLA has the hexadecimal code 7800, which gives the binary equivalent 0111 1000 0000 0000. The first bit is a zero and is equivalent to I'. The next three bits constitute the operation code and are recognized from decoder output D7. Bit 11 in IR is 1 and is recognized from Bn . The control function that initiates the micro operation for this instruction is D7I'T3B11 = rB11 . The execution of a register-reference instruction is completed at time T3. The sequence counter SC is cleared to 0 and the control goes back to fetch the next instruction with timing signal To. The first seven register-reference instructions perform clear, complement, circular shift, and increment micro operations on the AC or E registers. The next four instructions cause a skip of the next instruction in sequence when a stated condition is satisfied. The skipping of the instruction is achieved by incrementing PC once again (in addition, it is being incremented during the fetch phase at time T1). The condition control statements must be recognized as part of the control conditions. The AC is positive when the sign bit in AC(15) = 0; it is negative when AC(15) = 1. The content of AC is zero {AC = 0) if all the flip-flops of the register are zero. The HLT instruction clears a start-stop flip-flop S and stops the sequence counter from counting. To restore the operation of the computer, the start-stop flip-flop must be set manually.

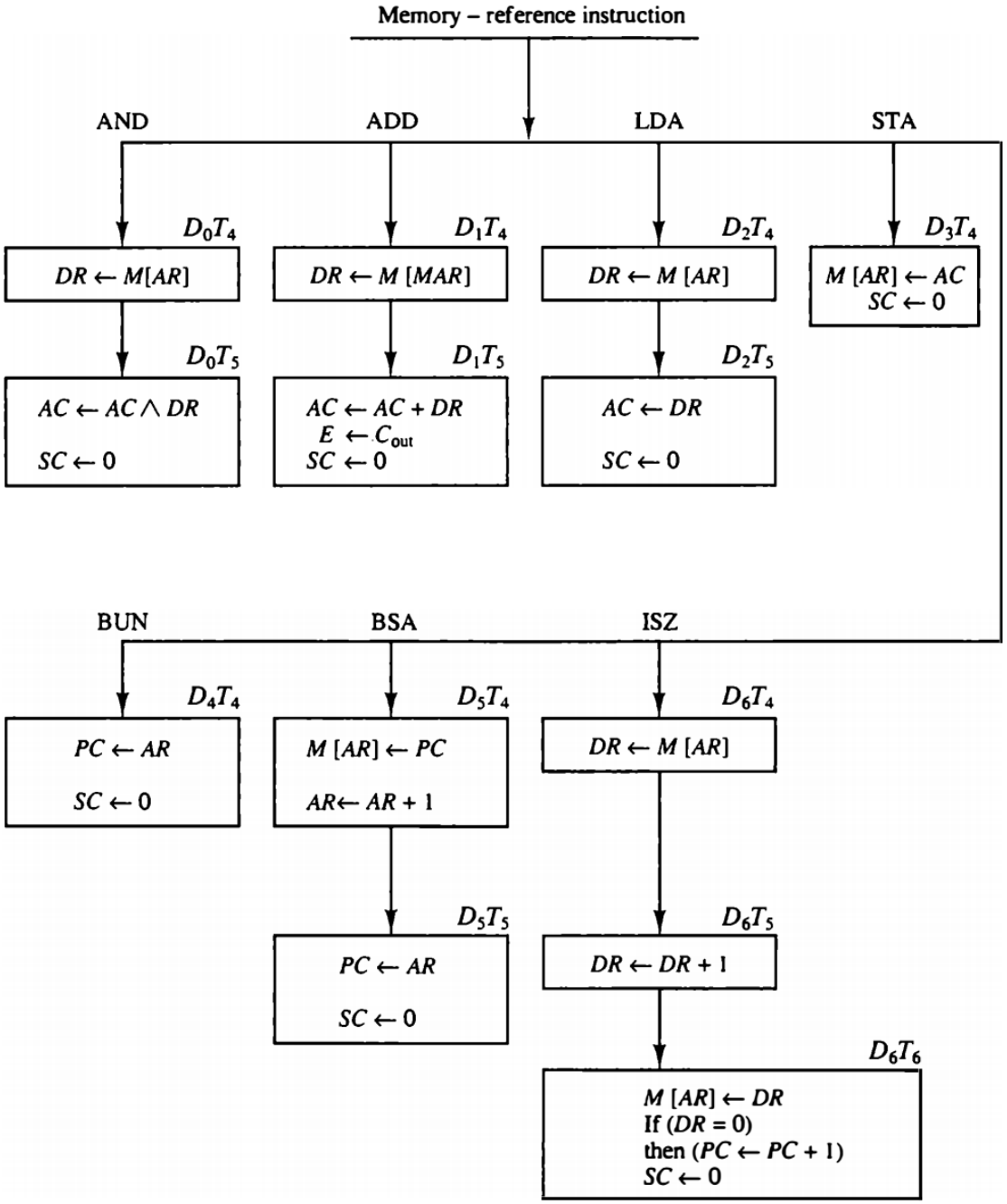
Memory Reference Instructions

In order to specify the micro operations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely. Table lists the seven memory-reference instructions. The decoded output D, for i = 0, 1, 2, 3, 4, 5, and 6 from the operation decoder that belongs to each instruction is included in the table. The effective address of the instruction is in the address register AR and was placed there during timing signal T2 when I = 0, or during timing signal T3 when I = 1. The execution of the memory-reference instructions starts with timing signal T4. The symbolic description of each instruction is specified in the table in terms of register transfer notation. The actual execution of the instruction in the bus system will require a sequence of micro operations. This is because data stored in memory cannot be processed directly. The data must be read from memory to a register where they can be operated on with logic circuits.

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

A flowchart showing all micro operations for the execution of the seven memory-reference instructions is shown in Fig. 5-11. The control functions are indicated on top of each box. The

micro operations that are performed during time T 4 , T 5 , or T 6 depend on the operation code value. This is indicated in the flowchart by six different paths, one of which the control takes after the instruction is decoded. The sequence counter SC is cleared to 0 with the last timing signal in each case. This causes a transfer of control to timing signal To to start the next instruction cycle. Note that we need only seven timing signals to execute the longest instruction (ISZ). The computer can be designed with a 3-bit sequence counter.

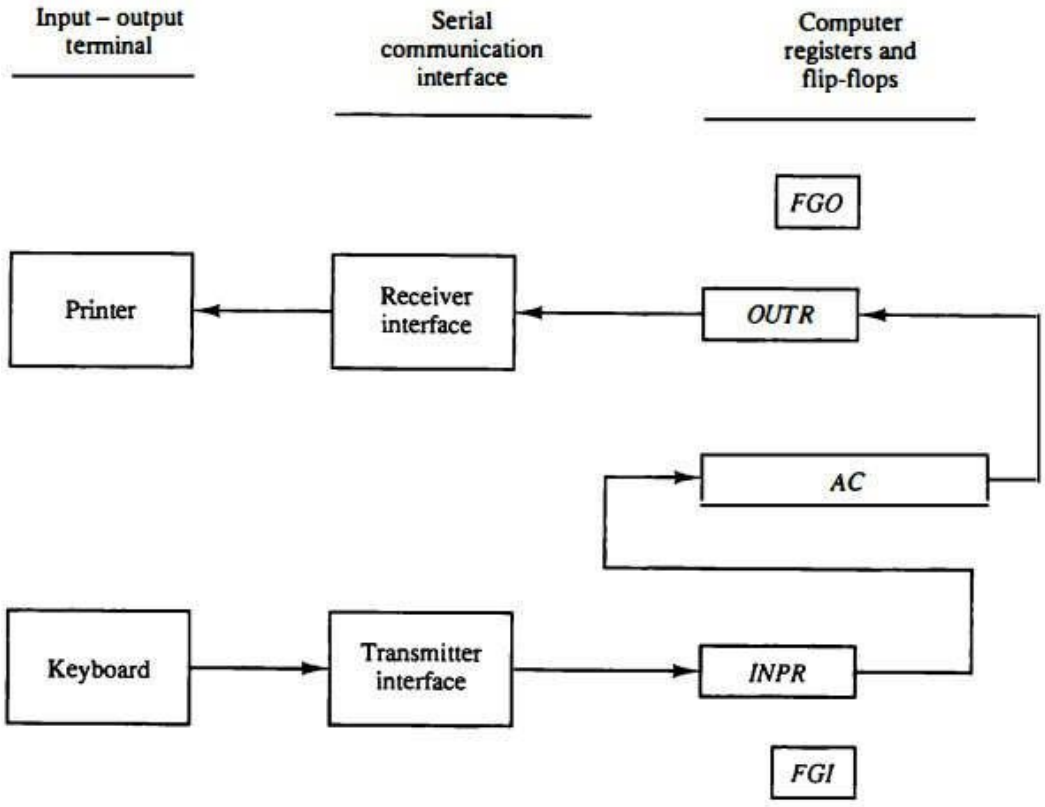


Input-Output and Interrupt

A computer can serve no useful purpose unless it communicates with the external environment. Instructions and data stored in memory must come from some input device. Computational results must be transmitted to the user through some output device. Commercial computers include many types of input and output devices. To demonstrate the most basic requirements for input and output communication, we will use as an illustration a terminal unit with a keyboard and printer.

Input-Output Configuration

The terminal sends and receives serial information. Each quantity of information has eight bits of an alphanumeric code. The serial information from the keyboard is shifted into the input register INPR. The serial information for the printer is stored in the output register OUTR. These two registers communicate with a communication interface serially and with the AC in parallel. The input-output configuration is shown in Figure.



The transmitter interface receives serial information from the keyboard and transmits it to INPR. The receiver interface receives information from OUTR and sends it to the printer serially. The input register INPR consists of eight bits and holds an alphanumeric input information. The 1-bit input flag FGI is a control flip-flop. The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer. The flag is needed to synchronize the timing rate difference between the input device and the computer. The process of information transfer is as follows. Initially, the input flag FGI is cleared to 0. When a key is struck in the keyboard, an 8-bit alphanumeric code is

shifted into INPR and the input flag FGI is set to 1. As long as the flag is set, the information in INPR cannot be changed by striking another key. The computer checks the flag bit; if it is 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0. Once the flag is cleared, new information can be shifted into INPR by striking another key. The output register OUTR works similarly but the direction of information flow is reversed. Initially, the output flag FGO is set to 1. The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1. The computer does not load a new character into OUTR when FGO is 0 because this condition indicates that the output device is in the process of printing the character.

Input-Output Instructions

Input and output instructions are needed for transferring information to and from AC register, for checking the flag bits, and for controlling the interrupt facility. Input-output instructions have an operation code 1111 and are recognized by the control when D7 = 1 and I = 1. The remaining bits of the instruction specify the particular operation. The control functions and micro operations for the input-output instructions are listed in Table. These instructions are executed with the clock transition associated with timing signal T3. Each control function needs a Boolean relation D7IT3, which we designate for convenience by the symbol p. The control function is distinguished by one of the bits in IR(6-11). By assigning the symbol B, to bit i of IR, all control functions can be denoted by pBi for i = 6 though 11. The sequence counter SC is cleared to 0 when p = D7IT3 = 1.

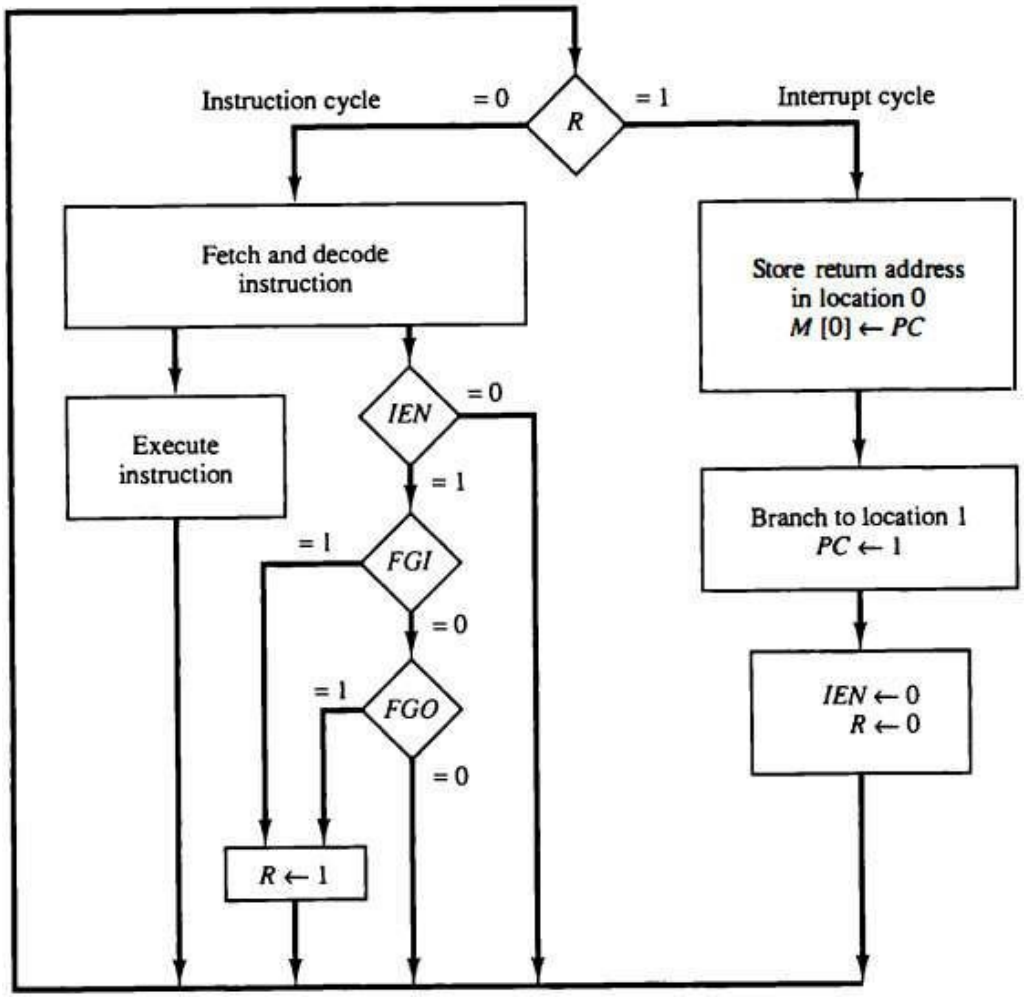
$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in IR(6–11) that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$	Output character
SKI	pB_9:	If ($FGI = 1$) then ($PC \leftarrow PC + 1$)	Skip on input flag
SKO	pB_8:	If ($FGO = 1$) then ($PC \leftarrow PC + 1$)	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

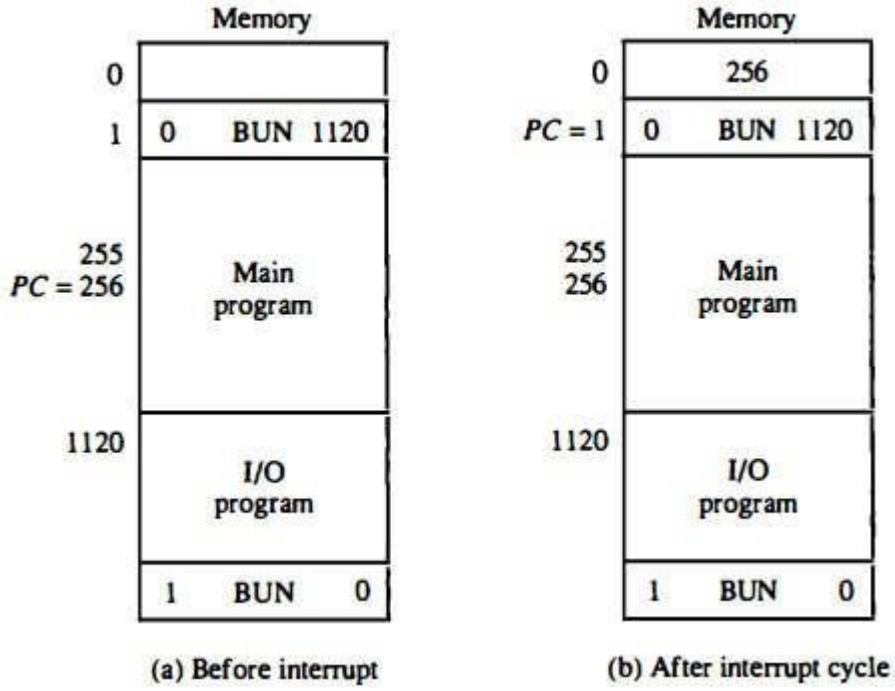
The INP instruction transfers the input information from INPR into the eight low-order bits of AC and also clears the input flag to 0. The OUT instruction transfers the eight least significant bits of AC into the output register OUTR and clears the output flag to 0. The next two instructions in Table check the status of the flags and cause a skip of the next instruction if the flag is 1. The instruction that is skipped will normally be a branch instruction to return and check the flag again. The branch instruction is not skipped if the flag is 0. If the flag is 1, the branch instruction is skipped and an input or output instruction is executed. The last two instructions set and clear an interrupt enable flip-flop IEN. The purpose of IEN is explained in conjunction with the interrupt operation.

Program Interrupt

The process of communication just described is referred to as programmed control transfer. The computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer. The difference of information flow rate between the computer and that of the input-output device makes this type of transfer inefficient. This means that at the maximum rate, the computer will check the flag multiple times between each transfer. The computer is wasting time while checking the flag instead of doing some other useful processing task. An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer. In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility. While the computer is running a program, it does not check the flags. However, when a flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that a flag has been set. The computer deviates momentarily from what it is doing to take care of the input or output transfer. It then returns to the current program to continue what it was doing before the interrupt. The interrupt enable flip-flop IEN can be set and cleared with two instructions. When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer. When IEN is set to 1 (with the ION instruction), the computer can be interrupted. These two instructions provide the programmer with the capability of making a decision as to whether or not to use the interrupt facility.



The way that the interrupt is handled by the computer can be explained by means of the flowchart of Figure. An interrupt flip-flop R is included in the computer. When R = 0, the computer goes through an instruction cycle. During the execute phase of the instruction cycle IEN is checked by the control. If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle. If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information. In this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN = 1, flip-flop R is set to 1. At the end of the execute phase, control checks the value of R, and if it is equal to l,it goes to an interrupt cycle instead of an instruction cycle.



The interrupt cycle is a hardware implementation of a branch and save return address operation. The return address available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted. This location may be a processor register, a memory stack, or a specific memory location. Here we choose the memory location at address 0 as the place for storing the return address. Control then inserts address 1 into PC and clears IEN and R so that no more interruptions can occur until the interrupt request from the flag has been serviced. An example that shows what happens during the interrupt cycle is shown in Figure. Suppose that an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255. At this time, the return address 256 is in PC. The programmer has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1. This is shown in Figure (a). When control reaches timing signal T 0 and finds that R = 1, it proceeds with the interrupt cycle. The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0. At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC. The branch instruction at address 1 causes the program to transfer to the input-output service program at address 1120. This program checks the flags, determines which flag is set, and then transfers the required input or

output information. Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted. This is shown in Figure (b). The instruction that returns the computer to the original place in the main program is a branch indirect instruction with an address part of 0. This instruction is placed at the end of the I/O service program. After this instruction is read from memory during the fetch phase, control goes to the indirect phase (because I = 1) to read the effective address. The effective address is in location 0 and is the return address that was stored there during the previous interrupt cycle. The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

MICRO-PROGRAMMED CONTROL

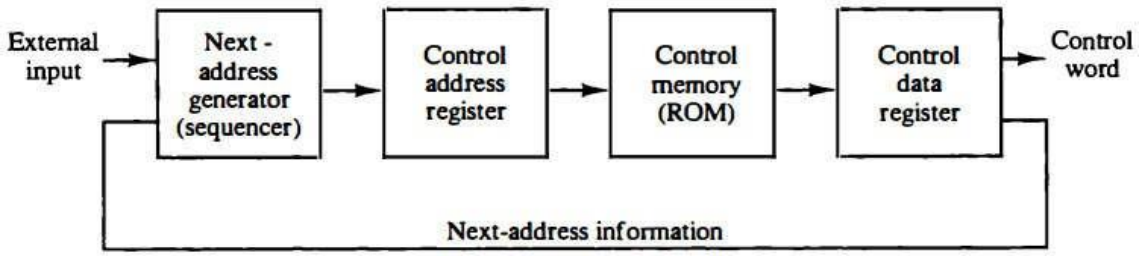
Control Memory

The function of the control unit in a digital computer is to initiate sequences of micro-operations. The number of different types of micro-operations that are available in a given system is finite. The complexity of the digital system is derived from the number of sequences of micro-operations that axe performed. When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired. Micro programming is a second alternative for designing the control unit of a digital computer. The principle of micro programming is an elegant and systematic method for controlling the micro-operation sequences in a digital computer.

The control function that specifies a micro-operation is a binary variable. When it is in one binary state, the corresponding micro-operation is executed. A control variable in the opposite binary state does not change the state of the registers in the system. The active state of a control variable may be either the 1 state or the 0 state, depending on the application. In a bus-organized system, the control signals that specify micro-operations are groups of bits that s elect the paths in multiplexers, decoders, and arithmetic lope units. The control unit initiates a series of sequential steps of micro-operations. During any given time, certain micro-operations are to be initiated, while others remain idle. The control variables at any given time can be represented by a string of l's and 0's called a control word. As such, control words can be programmed to perform various operations on the components of the system. A control unit whose binary control variables are stored in memory is called a micro programmed control unit. Each word in control memory contains within it a microinstruction. The microinstruction specifies one or more micro-operations for the system. A sequence of microinstructions constitutes a micro program. Since alterations of the micro program are not needed once the control unit is in operation, the control memory can be a read-only memory (ROM). The content of the words in ROM are fixed and cannot be altered by simple programming since no writing capability is available in the ROM. ROM words are made permanent during the hardware production of the unit. The use of a micro program involves placing all control variables in words of ROM for use by the control unit through successive read operations. The content of the word in ROM at a given address specifies a microinstruction.

A more advanced development known as dynamic micro programming permits a micro program to be loaded initially from an auxiliary memory such as a magnetic disk. Control units that use dynamic micro programming employ a writable control memory. This type of memory can be used for writing (to change the micro program) but is used mostly for reading. A memory that is part of a control unit is referred to as a control memory. A computer that employs a micro programmed control unit will have two separate memories: a main memory and a control memory. The main memory is available to the user for storing the programs. The

contents of main memory may alter when the data are manipulated and every time that the program is changed. The user's program in main memory consists of machine instructions and data. In contrast, the control memory holds a fixed micro program that cannot be altered by the occasional user. The micro program consists of microinstructions that specify various internal control signals for execution of register micro-operations. Each machine instruction initiates a series of microinstructions in control memory. These microinstructions generate the micro- operations to fetch the instruction from main memory; to evaluate the effective address, to execute the operation specified by the instruction, and to return control to the fetch phase in order to repeat the cycle for the next instruction. The general configuration of a micro programmed control unit is demonstrated in the block diagram of Figure.



The control memory is assumed to be a ROM, within which all control information is permanently stored. The control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction read from memory. The microinstruction contains a control word that specifies one or more micro-operations for the data processor. Once these operations are executed, the control must determine the next address. The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory. For this reason it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction. The next address may also be a function of external input conditions. While the micro-operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction. Thus a microinstruction contains bits for initiating micro-operations in the data processor part and bits that determine the address sequence for the control memory.

The next address generator is sometimes called a micro program sequencer, as it determines the address sequence that is read from control memory. The address of the next microinstruction can be specified in several ways, depending on the sequencer inputs. Typical functions of a micro program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address, or loading an initial address to start the control operations. The control data register holds the present microinstruction while the next address is computed and read from memory. The data register is sometimes called a pipeline register. It allows the execution of the micro-operations specified by the control word simultaneously with the generation of the next microinstruction. This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register. The system can operate without the control data register by applying a single-phase clock to the address register. The control word and next-address information are taken directly from the control memory. It must be realized that a ROM operates as a combinational circuit, with the address value as the input and the corresponding word as the output. The content of the specified word in ROM remains in the output wires as long as its address value remains in the address register. No read signal is needed as in a random-access memory. Each clock pulse will execute the micro-operations specified by the control word and also

transfer a new address to the control address register. In the example that follows we assume a single-phase clock and therefore we do not use a control data register. In this way the address register is the only component in the control system that receives clock pulses. The other two components: the sequencer and the control memory are combinational circuits and do not need a clock.

The main advantage of the micro programmed control is the fact that once the hardware configuration is established, there should be no need for further hardware or wiring changes. If we want to establish a different control sequence for the system, all we need to do is specify a different set of micro-instructions for control memory. The hardware configuration should not be changed for different operations; the only thing that must be changed is the micro program residing in control memory.

It should be mentioned that most computers based on the reduced instruction set computer (RISC) architecture concept use hardwired control rather than a control memory with a micro program.

Address Sequencing

Microinstructions are stored in control memory in groups, with each group specifying a routine. Each computer instruction has its own micro program routine in control memory to generate the micro-operations that execute the instruction. The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another. To appreciate the address sequencing in a micro program control unit, let us enumerate the steps that the control must undergo during the execution of a single computer instruction.

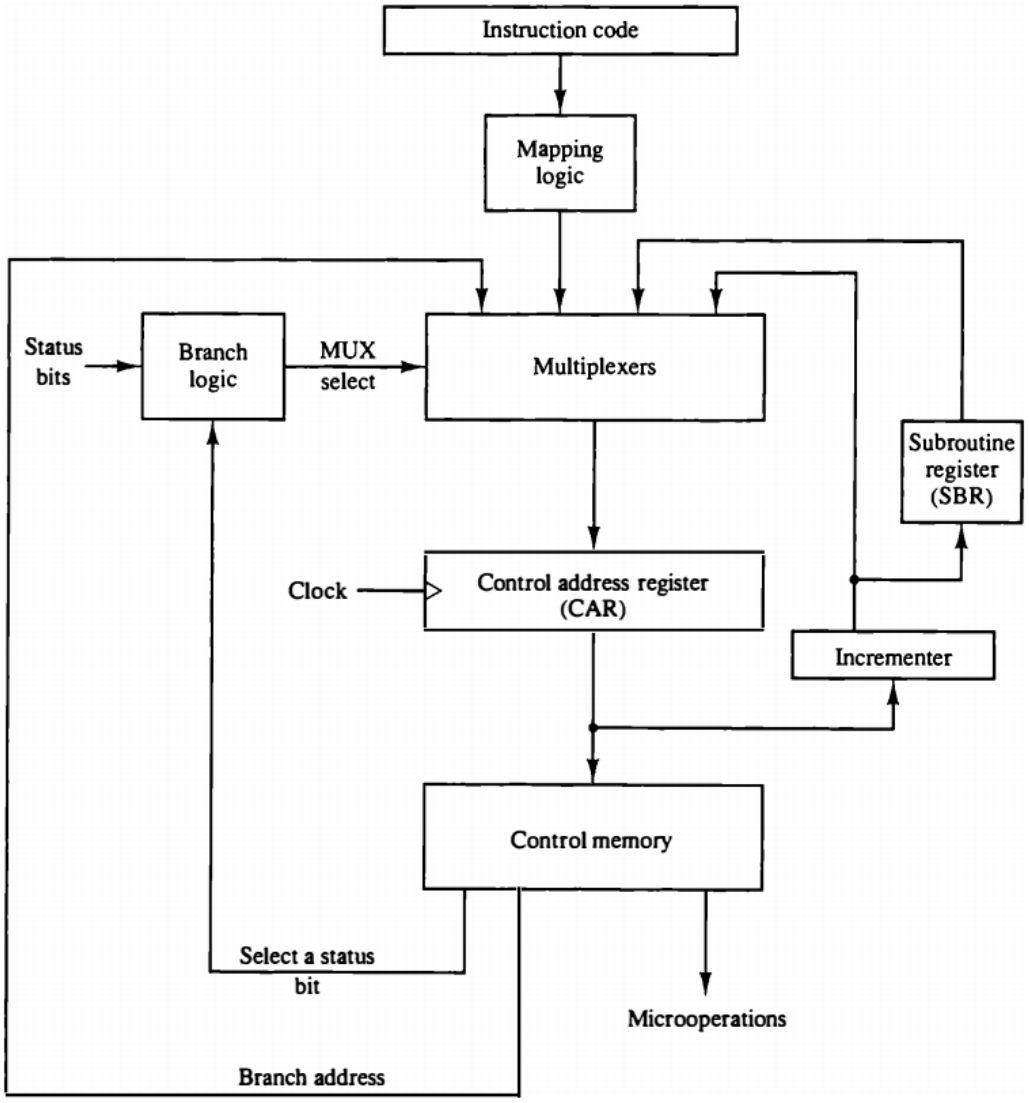
An initial address is loaded into the control address register when power is turned on in the computer. This address is usually the address of the first microinstruction that activates the instruction fetch routine. The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions. At the end of the fetch routine, the instruction is in the instruction register of the computer. The control memory next must go through the routine that determines the effective address of the operand. A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers. The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction. When the effective address computation routine is completed, the address of the operand is available in the memory address register.

The next step is to generate the micro-operations that execute the instruction fetched from memory. The micro-operation steps to be generated in processor registers depend on the operation code part of the instruction. Each instruction has its own micro program routine stored in a given location of control memory. The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a mapping process. A mapping procedure is a rule that transforms the instruction code into a control memory address. Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register, but sometimes the sequence of micro-operations will depend on values of certain status bits in processor registers. Micro programs that employ subroutines will require an external register for storing the return address. Return addresses cannot be stored in ROM because the unit has no writing capability.

When the execution of the instruction is completed, control must return to the fetch routine. This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine. In summary, the address sequencing capabilities required in a control memory are:

1. Incrementing of the control address register.
2. Unconditional branch or conditional branch, depending on status bit conditions.
3. A mapping process from the bits of the instruction to an address for control memory.
4. A facility for subroutine call and return.

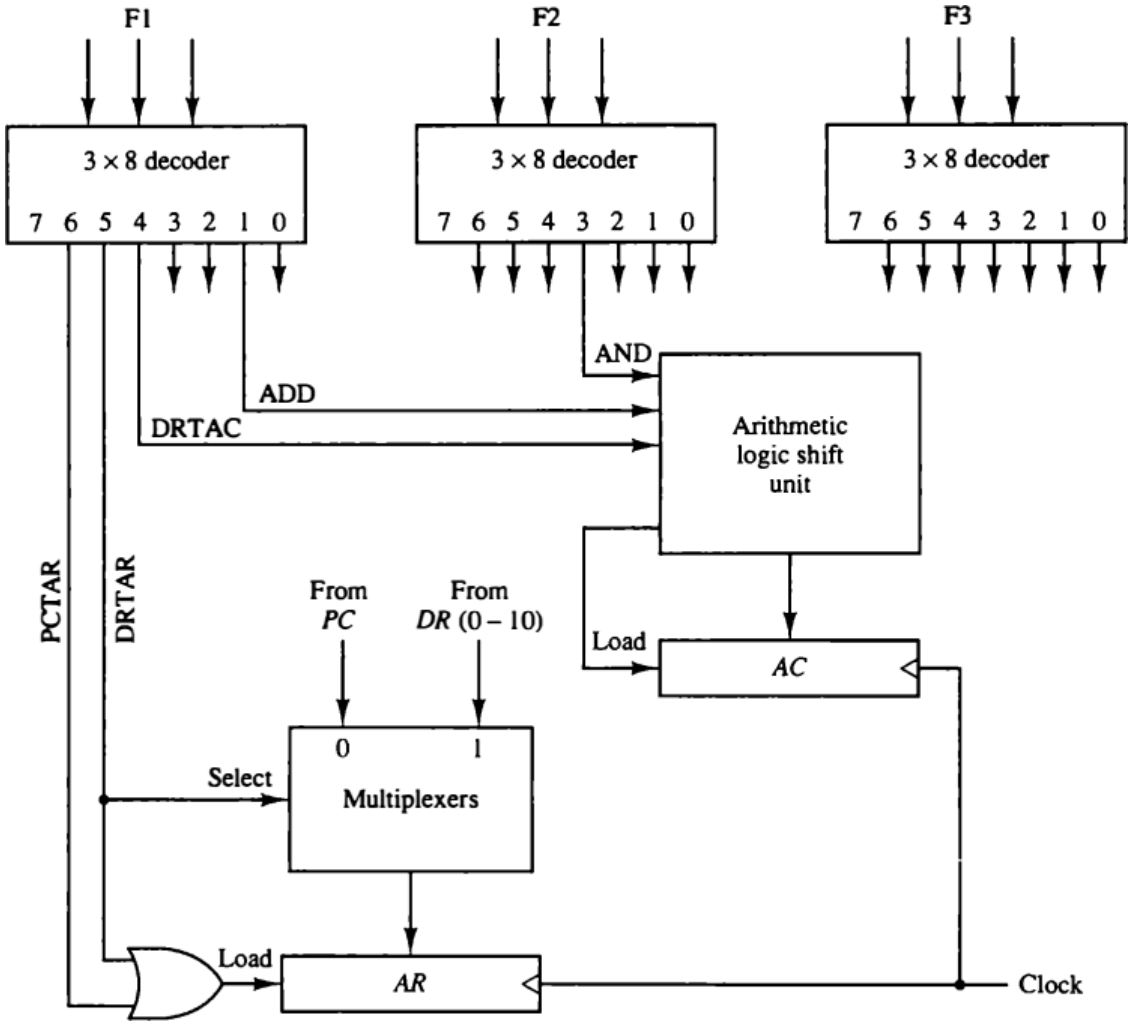
Figure shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address. The microinstruction in control memory contains a set of bits to initiate micro-operations in computer registers and other bits to specify the method by which the next address is obtained.



The diagram shows four different paths from which the control address register (CAR) receives the address. The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence. Branching is achieved by specifying the branch address in one of the fields of the microinstruction. Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition. An external address is transferred into control memory via a mapping logic circuit. The return address for a subroutine is stored in a special register whose value is then used when the micro program wishes to return from the subroutine.

Design of Control Unit

The bits of the microinstruction are usually divided into fields, with each field defining a distinct, separate function. The various fields encountered in instruction formats provide control bits to initiate micro-operations in the system, special bits to specify the way that the next address is to be evaluated, and an address field for branching. The number of control bits that initiate micro-operations can be reduced by grouping mutually exclusive variables into fields and encoding the k bits in each field to provide 2^k micro-operations. Each field requires a decoder to produce the corresponding control signals. This method reduces the size of the microinstruction bits but requires additional hardware external to the control memory. It also increases the delay time of the control signals because they must propagate through the decoding circuits. The encoding of control bits was demonstrated in the programming example of the preceding section. The nine bits of the micro-operation field are divided into three subfields of three bits each. The control memory output of each subfield must be decoded to provide the distinct micro-operations. The outputs of the decoders are connected to the appropriate inputs in the processor unit. Figure shows the three decoders and some of the connections that must be made from their outputs.



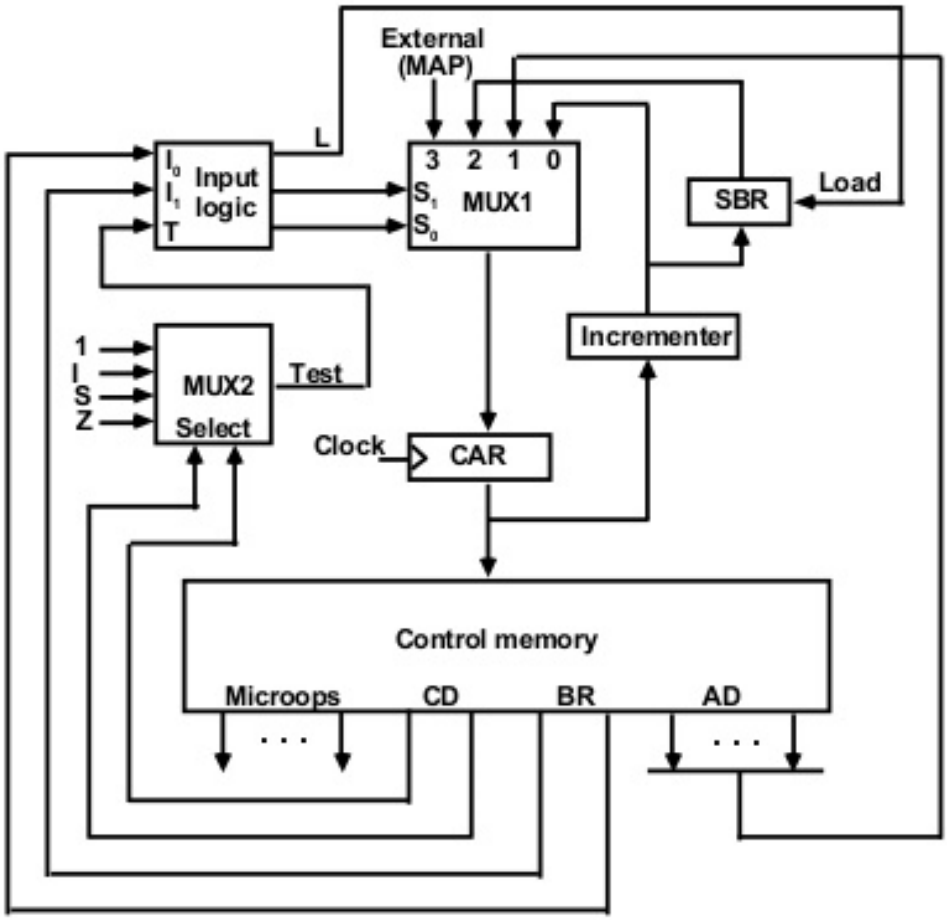
Each of the three fields of the microinstruction presently available in the output of control memory are decoded with a 3x8 decoder to provide eight outputs. Each of these outputs must be connected to the proper circuit to initiate the corresponding micro-operation as specified in Table. For example, when F1 = 101 (binary 5), the next clock pulse transition transfers the content of DR(0-10) to AR (symbolized by DRTAR in Table). Similarly, when F1 = 110 (binary 6) there is a transfer from PC to AR (symbolized by PCTAR). As shown in Figure, outputs 5 and 6 of decoder F1 are connected to the load input of AR so that when either one of these outputs is active, information from the multiplexers is transferred to AR. The multiplexers select the information from DR when output 5 is active and from PC when output 5 is inactive. The transfer into AR occurs with a clock pulse transition only when output 5 or output 6 of the decoder are active. The other outputs of the decoders that initiate transfers between registers must be connected in a similar fashion. The arithmetic logic shift unit can be designed. Instead of using gates to generate the control signals, the inputs will now come from the outputs of the decoders associated with the symbols AND, ADD, and DRTAC, respectively, as shown in Figure. The other outputs of the decoders that are associated with an AC operation must also be connected to the arithmetic logic shift unit in a similar fashion.

Micro program Sequencer

The basic components of a micro programmed control unit are the control memory and the circuits that select the next address. The address selection part is called a micro program sequencer. A micro program sequencer can be constructed with digital functions to suit a particular application. However, just as there are large ROM units available in integrated circuit packages, so are general-purpose sequencers suited for the construction of micro program control units. To guarantee a wide range of acceptability, an integrated circuit sequencer must provide an internal organization that can be adapted to a wide range of applications. The purpose of a micro program sequencer is to present an address to the control memory so that a microinstruction may be read and executed. The next-address logic of the sequencer determines the specific address source to be loaded into the control address register. The choice of the address source is guided by the next-address information bits that the sequencer receives from the present microinstruction. Commercial sequencers include within the unit an internal register stack used for temporary storage of addresses during micro program looping and subroutine calls. Some sequencers provide an output register which can function as the address register for the control memory.

The block diagram of the micro program sequencer is shown in Figure. The control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it. There are two multiplexers in the circuit. The first multiplexer selects an address from one of four sources and routes it into a control address register CAR. The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit. The output from CAR provides the address for the control memory. The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register SBR. The other three inputs to multiplexer number 1 come from the address field of the present microinstruction, from the output of SBR, and from an external source that maps the instruction. Although the diagram shows a single subroutine register, a typical sequencer will have a register stack about four to eight levels deep. In this way, a number of subroutines can be active at the same time. A push and pop operation, in conjunction with a stack pointer, stores and retrieves the return address during the call and return microinstructions.

The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer. If the bit selected is equal to 1, the T (test) variable is equal to 1; otherwise, it is equal to 0. The T value together with the two bits from the BR (branch) field go to an input logic circuit. The input logic in a particular sequencer will determine the type of operations that are available in the unit.



Typical sequencer operations are: increment, branch or jump, call and return from subroutine, load an external address, push or pop the stack, and other address sequencing operations. With three inputs, the sequencer can provide up to eight address sequencing operations. Some commercial sequencers have three or four inputs in addition to the T input and thus provide a wider range of operations. Note that the incrementer circuit in the sequencer of Figure is not a counter constructed with flip-flops but rather a combinational circuit constructed with gates. A combinational circuit incrementer can be designed by cascading a series of half-adder circuits. The output carry from one stage must be applied to the input of the next stage. One input in the first least significant stage must be equal to 1 to provide the increment-by-one operation.