EXPERIMENT NO. 05

DATE OF PERFORMANCE: **GRADE:**

DATE OF ASSESSMENT: **SIGNATURE OF LECTURER/ TTA:**

AIM: Introduction to basic computer organization and design.

THEORY:

we introduce a basic computer and show how its operation can be computer specified with register is defined by its internal transfer registers, statements.The organization and control of structure, the command the set of instructions that It uses. The design of the computer is then carried out in detail. Although the basic computer presented in this chapter is very small compared to commercial computers, It has the advantage of being simple enough so we can demonstrate the design process without too many complications.

Instruction Codes:

The organization and control of structure, the command the set of instructions that It uses. The design of the computer is then carried out in detail. Although the basic computer presented in this chapter is very small compared to commercial computers, It has the advantage of being simple enough so we can demonstrate the design process without too many complications.

It can be instructed as to what specific sequence of operations it must perform. program. The user of a computer can control the process by means of 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 microoperations 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.

Operation Code:

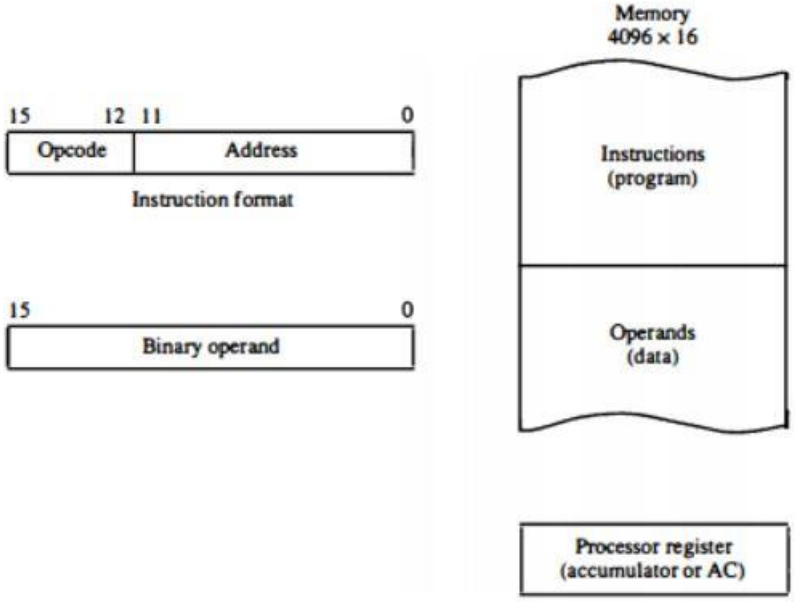
An instruction code is a group of bits that instruct the computer to perform a specific operation. It is usually divided into parts, each having its own particular interpretation. The most basic part of an instruction code is its operation part. The operation code of an instruction is a group of bits

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" (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.

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 212 = 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.

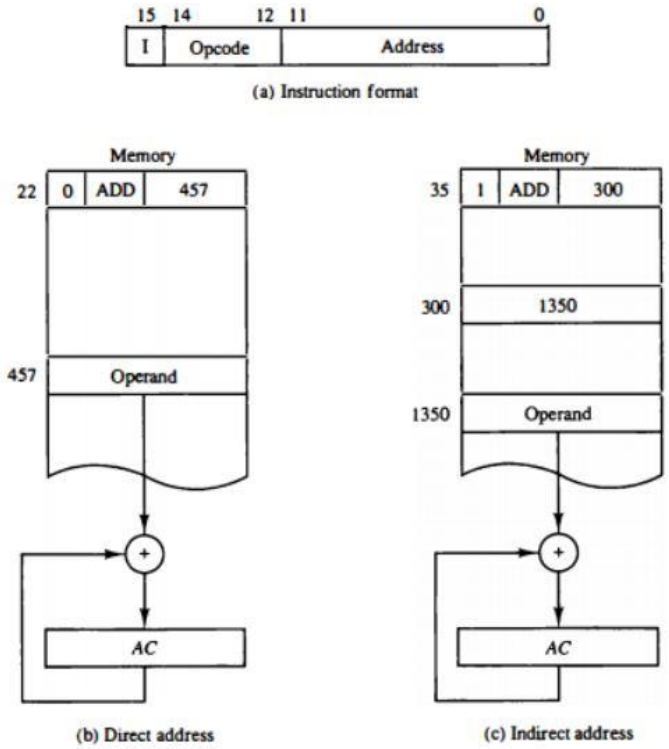
Computers that have a single-processor register usually assign to it the name accumulator and label it AC. The operation is performed with the memory operand and the content of AC.

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.

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 Fig. 5-2(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 Fig. 5-2(b). It is placed in address 22 in memory. 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 Fig. 5-2(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 Fig. 5-2(b) is 457 and in the instruction of Fig 5-2(c) is 1350.



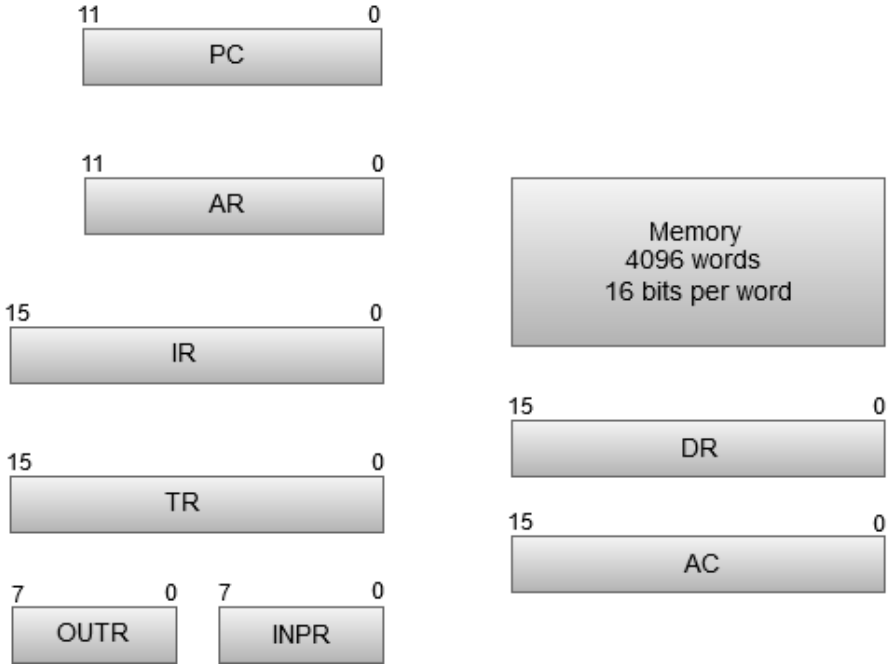
Computer Registers:

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 Fig. 5-3. The registers are also listed in Table 5-1 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

Register and Memory Configuration of a basic computer:



Common Bus System:

The basic computer has

1. **eight registers,**
2. **a memory unit, and**
3. **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. We can construct a bus system using multiplexers or three-state buffer gates. We just have to connect the registers and memory of the basic computer to a common bus system.

- **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. 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.**

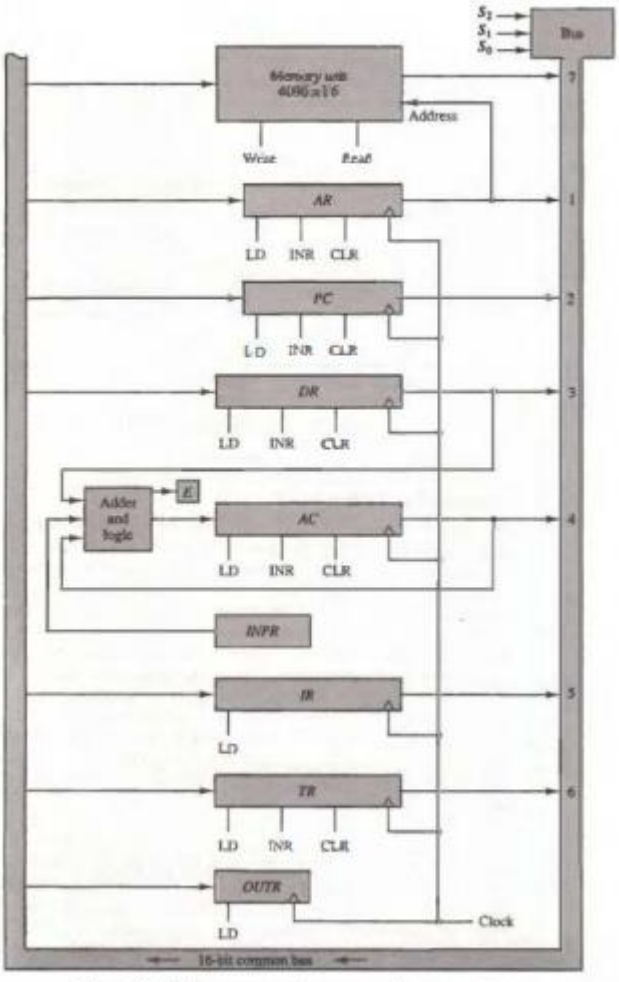
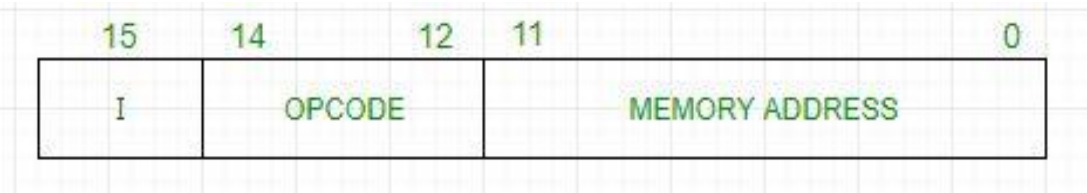


Figure 5-4 Basic computer registers connected to a common bus.

Computer Instructions:

The basic computer has 16-bit instruction register (IR) which can denote either memory reference or register reference or input-output instruction.

1) **Memory Reference – These instructions refer to memory address as an operand. The other operand is always accumulator. Specifies 12-bit address, 3-bit opcode (other than 111) and 1-bit addressing mode for direct and indirect addressing.**



Example –
IR register contains = 0001XXXXXXXXXXX, i.e. ADD after fetching and decoding of instruction we find out that it is a memory reference instruction for ADD operation.

Hence, DR ← M[AR]

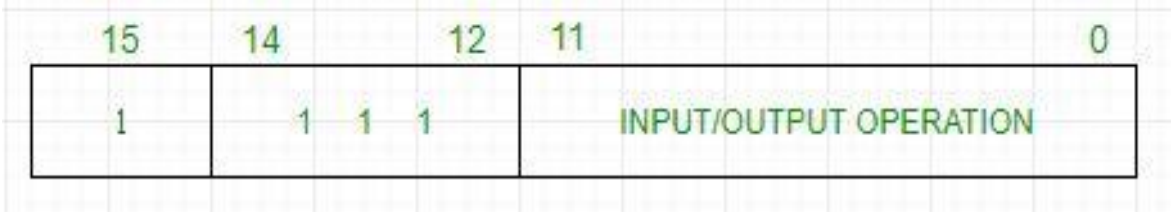
AC ← AC + DR, SC ← 0

2) **Register Reference – These instructions perform operations on registers rather than memory addresses. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 0 (differentiates it from input/output instructions). The rest 12 bits specify register operation.**

Example –
IR register contains = 0111001000000000, i.e. CMA after fetch and decode cycle we find out that it is a register reference instruction for complement accumulator.

Hence, AC ← ~AC

3) **Input/Output – These instructions are for communication between computer and outside environment. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 1 (differentiates it from register reference instructions). The rest 12 bits specify I/O operation.**



Example –
IR register contains = 1111100000000000, i.e. INP after fetch and decode cycle we find out that it is an input/output instruction for inputing character. Hence, INPUT character from peripheral device.

The set of instructions incorporated in16 bit IR register are:

1. **Arithmetic, logical and shift instructions (and, add, complement, circulate left, right, etc)**
2. **To move information to and from memory (store the accumulator, load the accumulator)**
3. **Program control instructions with status conditions (branch, skip)**
4. **Input output instructions (input character, output character)**

Symbol	Hexadecimal code		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 AC content in memory
BUN	4xxx	Cxxx	Branch Unconditionally
BSA	5xxx	Dxxx	Branch and Save Return Address
ISZ	6xxx	Exxx	Increment and skip if 0
CLA	7800		Clear AC

CLE	**7400**	**Clear E(overflow bit)**
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 > 0**
SNA	**7008**	**Skip next instruction if AC < 0**
SZA	**7004**	**Skip next instruction if AC = 0**
SZE	**7002**	**Skip next instruction if E = 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**

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 microoperations for the accumulator.

There are two major types of control organization:

1. **hardwired control and**
2. **microprogrammed 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 microprogrammed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations. 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 microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.

The block diagram of the control unit is shown in Fig. 5.6.

It consists of two decoders,

1. **a sequence counter, and**
2. **a number of control logic gates.**

The instruction register is shown again in Fig. 5.6, where it is divided into three parts:

1. **the 1 bit,**
2. **the operation code, and**
3. **bits 0 through 11.**

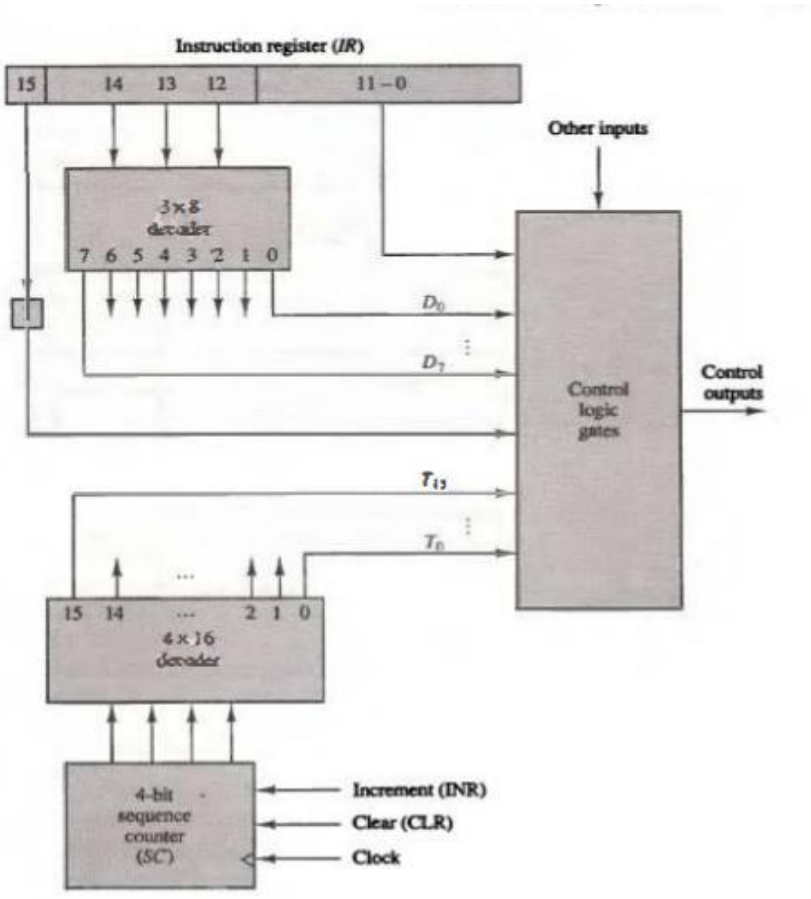


Figure 5-6 Control unit of basic computer.

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 D_0 through D_7. 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 T_0 through T_{15}.

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 a while, the counter is cleared to 0, causing the next active timing signal to be T_0.

As an example, consider the case where SC is incremented to provide timing signals T_0, T_1, T_2, T_3, and T_4 in sequence. At time T_4, SC is cleared to 0 if decoder output D_3 is active. This is expressed symbolically by the statement

$$D_3T_4: SC \leftarrow 0$$

The timing diagram of Fig. 5-7 shows the time relationship of the control signals.

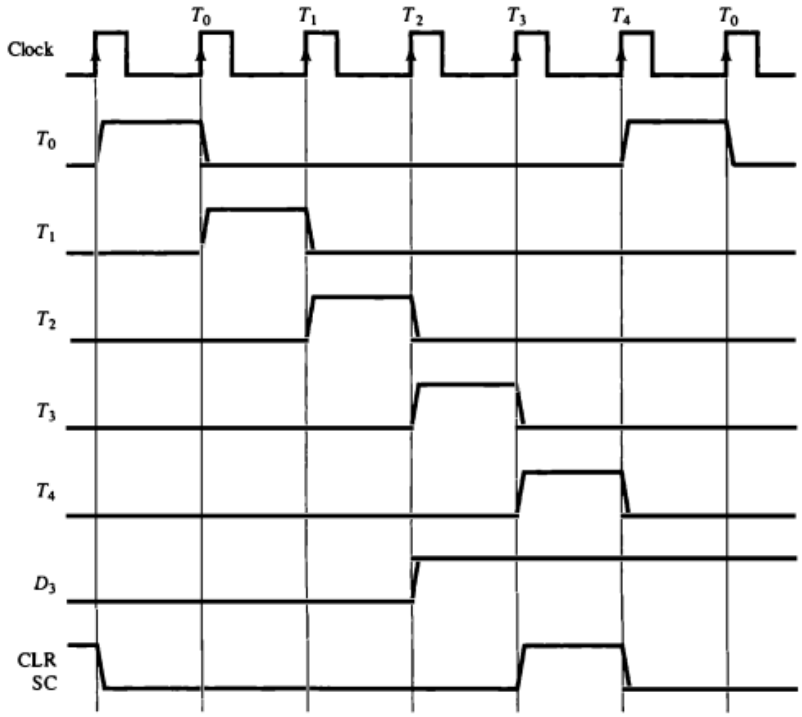


Figure 5-7 Example of control timing signals.

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 clears SC to 0, which in tum activates the timing signal T_0 out of the decoder. T_0 is active during one clock cycle. The positive clock transition labelled T_0 in the diagram will trigger only those registers whose control inputs are transition, to timing signal T_0. SC is incremented with every positive clock transition unless its CLR input is active. This produces the sequence of timing signals T_0, T_1, T_2, T_3 ,T_4 and so on, as shown in the diagram. (Note the the relationship between the timing signal and and its corresponding positive clock transition.) If SC is not cleared, the timing signals will continue with T5, T_6 up to T_{15} and back to T_0

The last three waveforms in Fig. 5-7 show how SC is cleared when $D_3T_4 = 1$. Output D_3 from the operation decoder becomes active at the end of timing signal T_2. When timing signal T_4 becomes active, the output of the AND gate that implements the control function D_3T_4 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 T_0 to become active instead of T_5 that would have been active if SC were incremented instead of cleared.

A memory read or write cycle will be initiated with the rising edge of a timing signal. It will be assumed that a memory cycle time is less than the clock cycle time. According to this assumption, a memory read or write cycle ini tiated by a timing signal will be completed by the time the next clock goes through its positive transition. The clock transition will then be used to load the memory word into a register. This timing relationship is not valid in many computers because the

memory cycle time is usually longer than the processor clock cycle. In such a case it is necessary to provide wait cycles in the processor until the memory word is available. To facilitate the presentation, we will assume that a wait period is not necessary in the basic computer.

To fully comprehend the operation of the computer, it is crucial that one understands the timing relationship between the clock transition and the timing signals. For example, the register transfer statement

$$T_0: AR \leftarrow PC$$

specifies a transfer of the content of PC into AR if timing signal T_0 is active. T_0 is active during an entire clock cycle interval During this time the content of PC is placed onto the bus (with $S_2S_1S_0$ = 010) and the LD (load) input of AR is enabled. The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition. This same positive clock transition increments the sequence counter SC from 0000 to 0001 . The next clock cycle has T_1 active and T_0 inactive.

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 subcycles 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.

Determine the Type of Instruction:

The timing signal that is active after the decoding is T3• During time T,, the control unit determines the type of instruction that was just read from memory. The flowchart of Fig. 5-9 presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding. The three possible instruction types available in the basic computer are specified in Fig. 5-5.

Decoder output D, is equal to 1 if the operation code is equal to binary Il l. From Fig. 5-5 we determine that if D7 = I, the instruction must be a

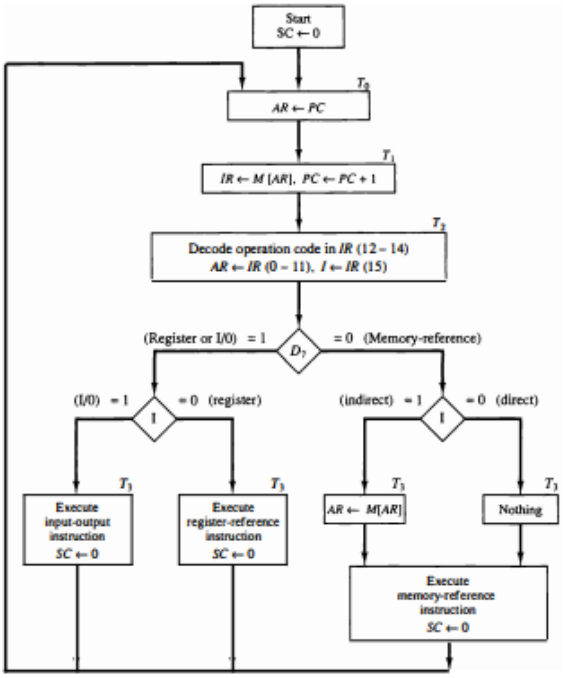


Figure 5-9 Flowchart for instruction cycle (initial configuration).

register-reference or input-output type. If D, = 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 microoperation 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:

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 D;T, = 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.

The register transfers needed for the execution of the register-reference instructions are presented in this section. The memory-reference instructions are explained in the next section. The input-output instructions are included in Sec. 5-7.

Register-Reference Instructions:

Register-reference instructions are recognized by the control when 07 = 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 microoperations for the register-reference instructions are. listed in Table 5-3. 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;. For example, the instruction CLA has the hexadecimal code 7800 (see Table 5-2), which gives the binary equivalent 011I 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 I and is recognized from 811• The control function that initiates the rnicrooperation 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 T0•

$D_7I'T_3 = r$ (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:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow$ 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

The first seven register-reference instructions perform clear, complement, circular shift, and increment microoperations 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(IS) = 0; it is negative when AC(IS) = I. 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 microoperations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely. Some instructions have an ambiguous description. This is because the explanation of an instruction in words is usually lengthy, and not enough space is available in the table for such a lengthy explanation.

We will now show that the function of the memory-reference instructions can be defined precisely by means of register transfer notation.

Table 5-4 lists the seven memory-reference instructions

The decoded 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 microoperations. 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. We now explain the operation of each instruction and list the control functions and microoperations needed for their execution.
A flowchart that summarizes all the microoperations is presented at the end of this section.

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

AND to AC:

This is an instruction that perform the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC.

The microoperations that execute this instruction are:

D0T4: DR <- M [AR]

D0T5: AC <- AC ∧ DR, SC <--- 0

The control function for this instruction uses the operation decoder D0 since this output of the decoder is active when the instruction has an AND operation whose binary code value 000. Two timing signals are needed to execute the instruction. The clock transition associated with timing signal T4 transfers the operand from memory into DR . The clock transition associated with the next timing signal T5 transfers to AC the result of the AND logic operation between the contents of DR and AC. The same clock transition clears SC to 0, transferring control to timing signal T0 to start a new instruction cycle.

ADD to AC

This instruction adds the content of the memory word specified by the effective address to the value of AC . The sum is transferred into AC and the output carry C_{out} is transferred to the E (extended accumulator) flip-flop. The microoperations needed to execute this instruction are

D_1T_4: DR← M[AR]

D_1T_5: AC← AC + DR, E← C_{out} , SC ← 0

Same Two timing signals, T, and T5, are used again but with operation decoder D1 instead of D0, which was used for the AND instruction. After the instruction is fetched from memory and decoded, only one output of the operation decoder will be active, and that output determines the sequence of microoperations that the control follows during the execution of a memory-reference instruction.

LDA: Load to AC

This instruction transfers the memory word specified by the effective address to AC . The microoperations needed to execute this instruction are

D_2T_4: DR← M [AR]

D_2T_5: AC← DR , ← 0

Looking back at the bus system shown in Fig. 5-4 we note that there is no direct path from the bus into AC . The adder and logic circuit receive information from DR which can be transferred into AC . Therefore, it is necessary to read the memory word into DR first and then transfer the content of DR into AC . The reason for not connecting the bus to the inputs of AC is the delay

encountered in the adder and logic circuit. It is assumed that the time it takes to read from memory and transfer the word through the bus as well as the adder and logic circuit is more than the time of one clock cycle. By not connecting the bus to the inputs of AC we can maintain one clock cycle per microoperation.

STA: Store AC

This instruction stores the content of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation:

D3T4: M [AR] ← AC, SC ← 0

BUN: Branch Unconditionally

This instruction transfers the program to the instruction specified by the effective address. Remember that PC holds the address of the instruction to be read from memory in the next instruction cycle. PC is incremented at time T1 to prepare it for the address of the next instruction in the program sequence. The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally. The instruction is executed with one microoperation:

D4T4: PC ← AR, SC ← 0

The effective address from AR is transferred through the common bus to PC.Resetting SC to 0 transfers control to T0• The next instruction is then fetched and executed from the memory address given by the new value in PC .

BSA: Branch and Save Return Address:

This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address. The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine. This operation was specified in Table 5-4 with the following register transfer:

M[AR] <-- PC, PC <-- AR + I

A numerical example that demonstrates how this instruction is used with a subroutine is shown in Fig. 5-10. The BSA instruction is assumed to be in memory at address 20. The I bit is 0 and the address part of the instruction has the binary equivalent of 135. After the fetch and decode phases, PC contains 21, which is the address of the next instruction in the program (referred to as the return address). AR holds the effective address 135. This is shown in part (a) of the figure. The BSA instruction performs the following numerical operation:

M[135] <-- 21, PC <-- 135 + 1 = 136

The result of this operation is shown in part (b) of the figure. The return address 21 is stored in memory location 135 and control continues with the subroutine program starting from address 136. The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine. When this instruction is executed, control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21. When the BUN instruction is executed, the effective address 21 is transferred to PC . The next instruction cycle finds PC with the value 21, so control continues to execute the instruction at the return address.

ISZ: Increment and Skip if Zero

This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. The programmer usually stores a negative number (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time PC is incremented by one in order to skip the next instruction in the program.

Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory. This is done with the following sequence of microoperations:

D6T4: DR <-- M [AR]

D6T5: DR <-- DR + 1

D,T,: M [AR] <-- DR,

if (DR = 0) then (PC ← PC + 1), SC ← 0

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.

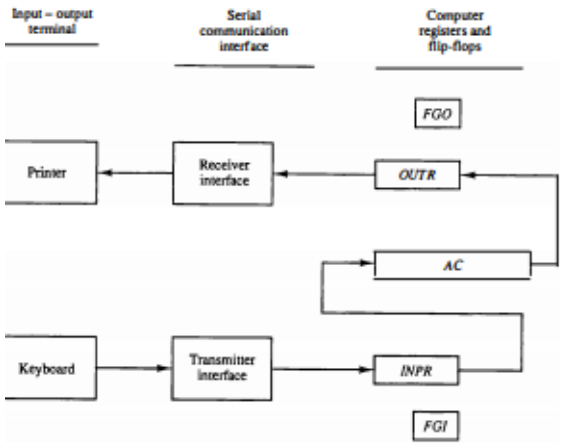
Input-Output Configuration

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 Fig. 5-12. 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 operation of the serial communication interface is explained in Sec. 11-3.

Input register:

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

Figure 5-12 Input-output configuration.



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.

Output register:

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.