# CHAPTER 3

# BASIC COMPUTER ORGANISATION AND DESIGN

LH- 8 Hours

Prepared By: Rolisha Sthapit

# CONTENTS

## INTRODUCTION: Description of Basic Computer

- We introduce here a basic computer whose operation can be specified by the register transfer statements. Internal organization of the computer is defined by the sequence of microoperations it performs on data stored in its registers. Every different processor type has its own design (different registers, buses, microoperations, machine instructions, etc). Modern processor is a very complex device. It contains:

– Many registers

– Multiple arithmetic units, for both integer and floating point calculations

– The ability to pipeline several consecutive instructions for execution speedup

## Continue

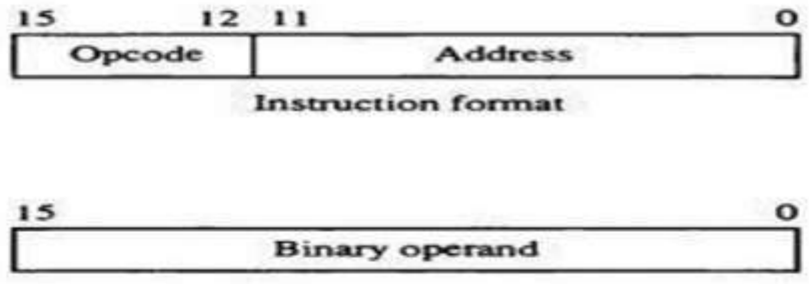
- However, to understand how processors work, we will start with a simplified processor model. M. Morris Mano introduces a simple processor model; he calls it a —Basic Computer. The Basic Computer has two components, a processor and memory.
- The memory has 4096 words in it
  - –4096 = $2^{12}$, so it takes 12 bits to select an address in memory
- Each word is 16 bits long

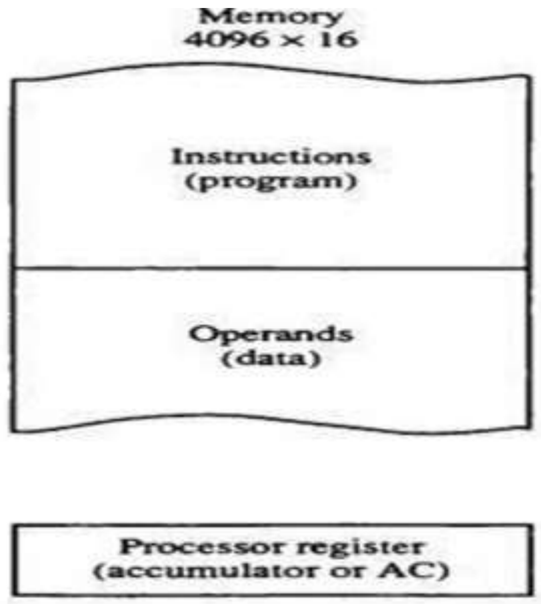
## 3.1 Instruction Code and Stored Program Organisation

- An **instruction code** is a group of bits that instructs the computer to perform a specific operation. It is usually divided into parts each having one particular interpretation. Most basic part is operation (**operation code**).
- Operation code is group of bits that defines operations as add, subtract, multiply, shift, complement etc. The operation part of an instruction code specifies the operation to be performed. This operation must be performed on some data stored in processor register or in memory.
- An instruction code therefore specify not only the operation but also the register or the memory word where the operand (data on which operation is performed) are to be found.

## Stored Program Organization:

- The program (instruction) as well as data (operand) is stored in the same memory. If the instruction needs data, the data is found in the same memory and accessed. This feature is called stored program organization.





# Instruction Formats

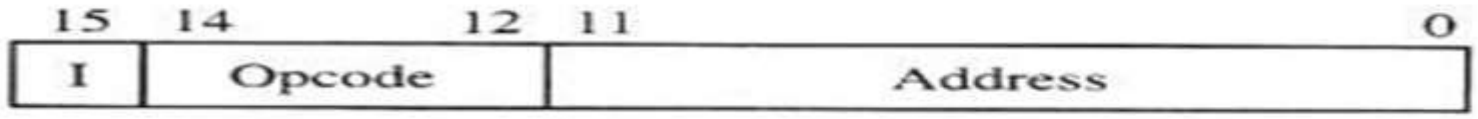
Instruction Format of Basic Computer:

A computer instruction is often divided into two parts

– An *opcode* (Operation Code) that specifies the operation for that instruction

– An *address* that specifies the registers and/or locations in memory to use for that operation.

In the Basic Computer, since the memory contains 4096 (= $2^{12}$) words, we needs 12 bit to specify the memory address that is used by this instruction. In the Basic Computer, bit 15 of the instruction specifies the *addressing mode* (0: direct addressing, 1: indirect addressing). Since the memory words, and hence the instructions, are 16 bits long, that leaves 3 bits for the instruction's opcode.



(a) Instruction format

## Addressing Modes:

Addressing Modes:

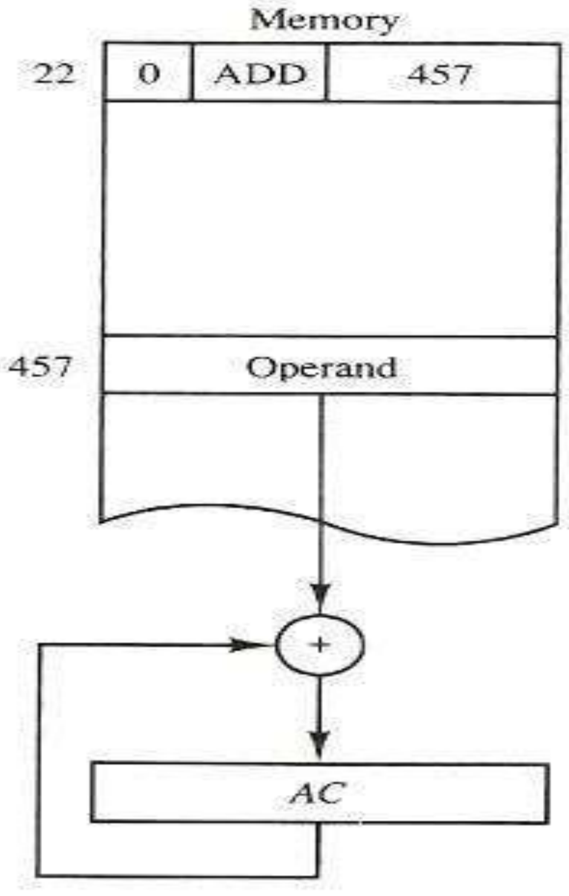
The address field of an instruction can be represented in two ways

– Direct address: the address operand field is effective address (the address of the operand)

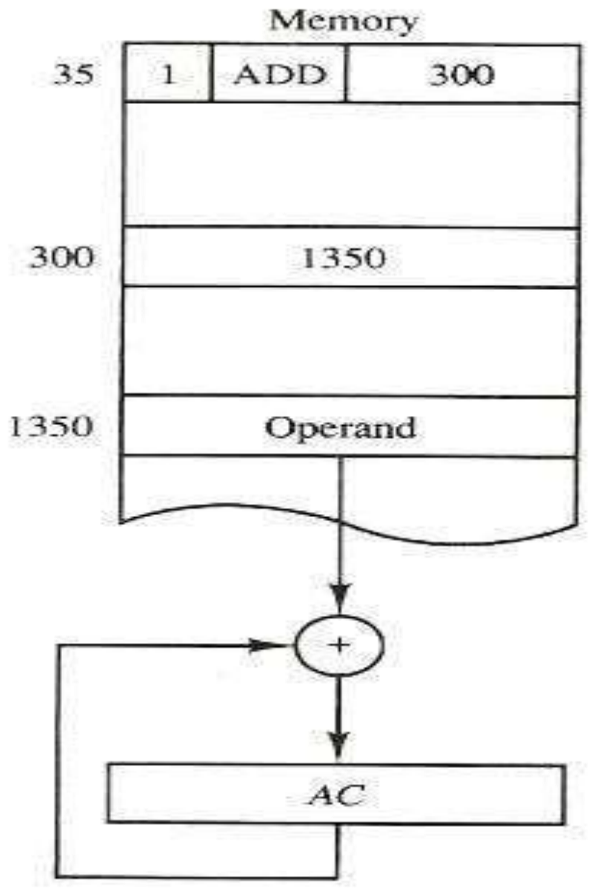
– Indirect address: the address in operand field contains the memory address where effective address resides.

Note: Effective Address (EA): The address, where actual data resides is called effective address.

## Continue:



(b) Direct address



(c) Indirect address

## Continue

- A direct address instruction is shown in figure 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 the 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 indirect address instruction. The address part is the binary equivalent of 300. The control goes to address 300 to find the address of an operand. The address of an 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 and second is for the operand itself. The effective address of direct addressing mode is 457 and that of indirect addressing mode is 1350.

## 3.2 Computer Registers

- Computer instructions are normally stored in the consecutive memory locations and are executed sequentially one at a time. Thus computer needs processor registers for manipulating data and holding memory address which are shown in the following table:

| Symbol | Size | Register Name | Description |
|---|---|---|---|
| | | | |
| 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 |
| | | | |

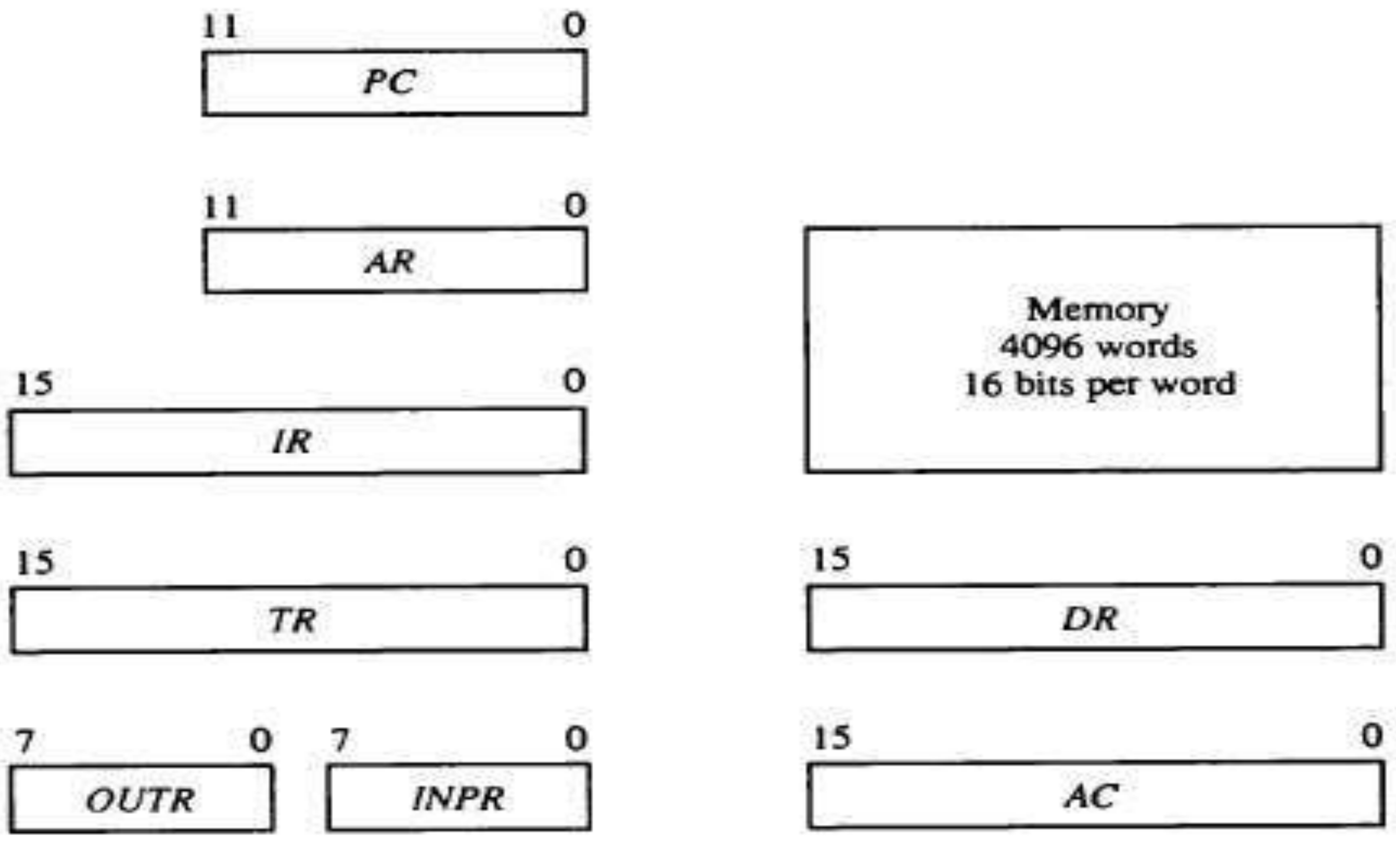


**Figure 5-3** Basic computer registers and memory.

- Since the memory in the Basic Computer only has 4096 (=$2^{12}$) locations, PC and AR only needs 12 bits. Since the word size of Basic Computer only has 16 bit, the DR, AC, IR and TR needs 16 bits. The Basic Computer uses a very simple model of input/output (I/O) operations.

–Input devices are considered to send 8 bits of character data to the processor

–The processor can send 8 bits of character data to output devices

The Input Register (INPR) holds an 8-bit character gotten from an input device and the Output Register (OUTR) holds an 8-bit character to be sent to an output device.

Q) Why the address bus (AR) and PC of basic computer is 12 bit?

Q) What is the use of PC?

## Common Bus System

- The basic computer has eight registers, a memory unit, and a control unit. These registers, memory and control unit are connected using a path (bus) so that information can be transferred to each other. If separate buses are used for connecting each registers, it will cost high. The cost and use of extra buses can be reduced using a special scheme in which many registers use a common bus, called *common bus system*

## Continue….

- The registers in the Basic Computer are connected using a bus. This gives a savings in circuitry over complete connections between registers. Three control lines, S2, S1, and S0 control which register the bus selects as its input.

| $S_2$ $S_1$ $S_0$ | | | Register |
|---|---|---|---|
| 0 | 0 | 0 | X (nothing) |
| 0 | 0 | 1 | AR |
| 0 | 1 | 0 | PC |
| 0 | 1 | 1 | DR |
| 1 | 0 | 0 | AC |
| 1 | 0 | 1 | IR |
| 1 | 1 | 0 | TR |
| 1 | 1 | 1 | Memory |

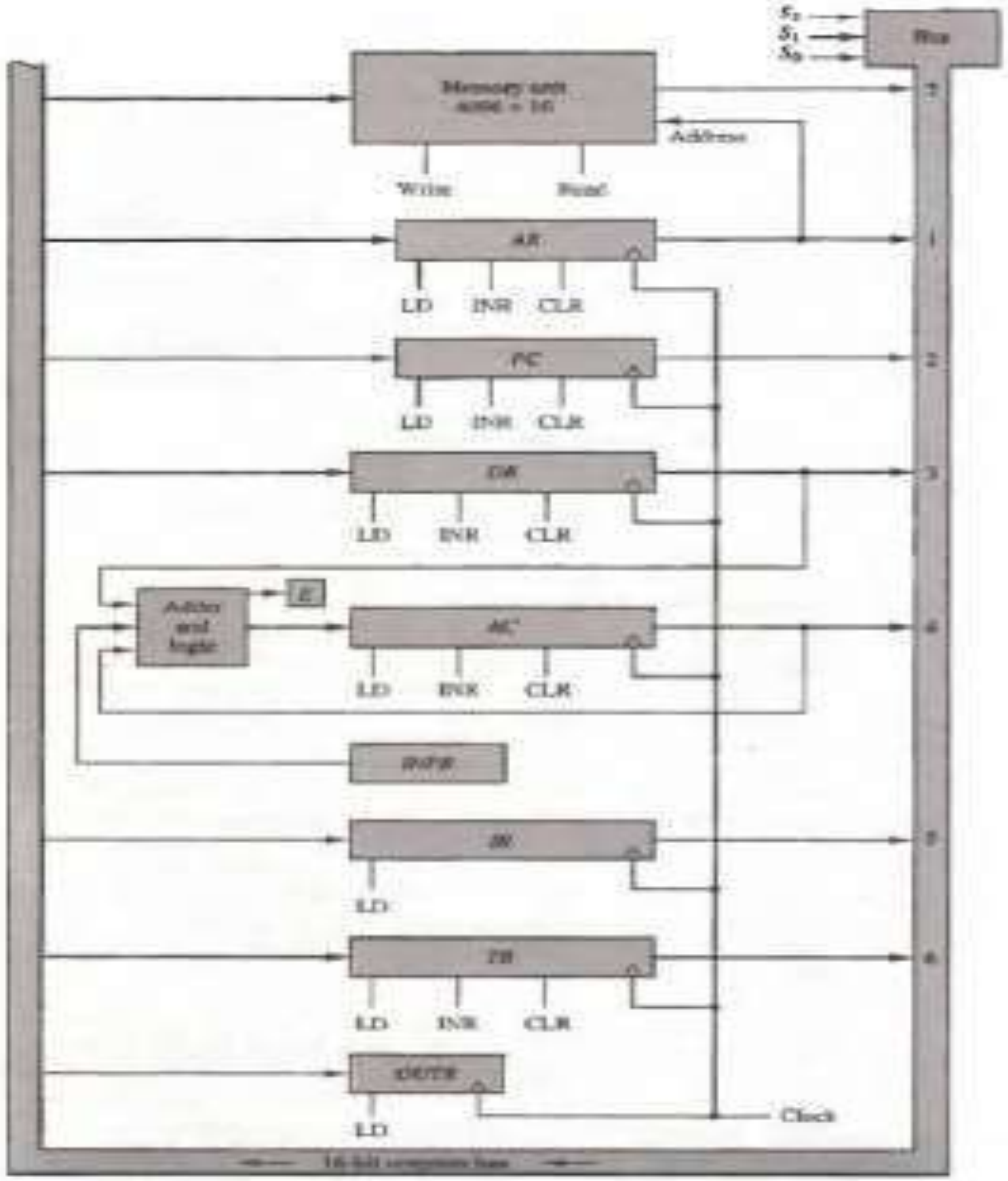


**Figure 5-4** Basic computer registers connected to a common bus.

- 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 the value of S2, S1, and S0 is 111.
- Four registers DR, AC, IR and TR have 16 bits each and two registers AR and PC have 12 bits each since they hold a memory address.
- When the content of AR and PC are applied to the 16-bit common bus, the four MSBs are set to 0’s.
- When AR and 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.

- INPR is connected to provide information to the bus but OUTR can only receive information from the bus.
- Five registers have three control inputs: LD (load), INR (increment) and CLR(Clear).
- Two registers have only a LD input.
- AR must always be used to specify memory address; therefore memory address is connected to AR. The 16 inputs of AC come from an Adder and Logic Circuit. This circuit has three inputs.
- Set of 16- bit inputs come from the outputs of AC.
- Set of 16-bit inputs come from the data register DR.
- Set of 8-bit inputs come from the input register INPR.

- 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).
- 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.

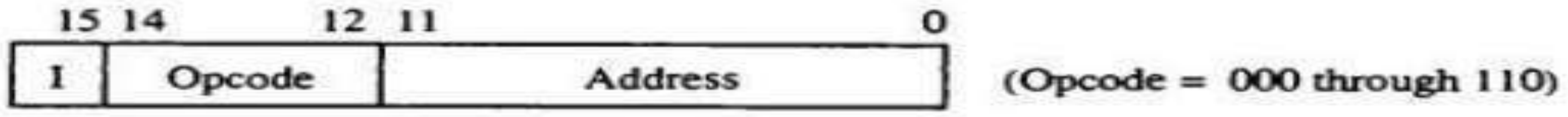
IMP ✓

## 3.3 Basic Computer Instructions

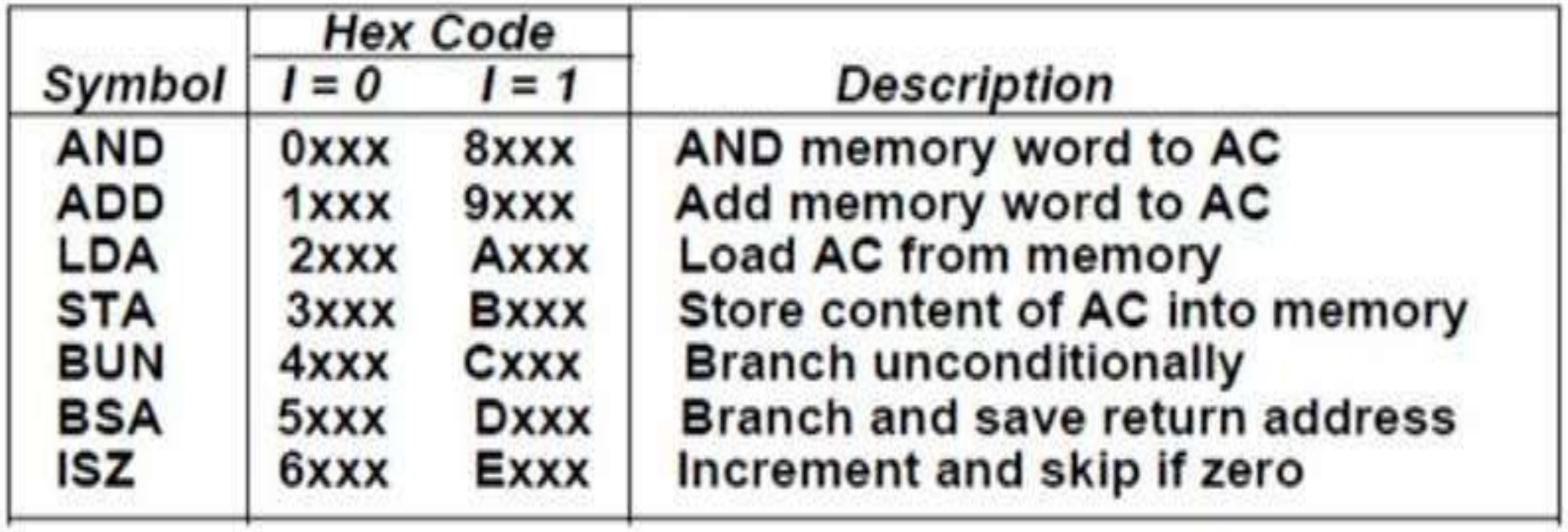
The basic computer has 3 instruction code formats. Type of the instruction is recognized by the computer control from 4-bit positions 12 through 15 of the instruction.

- **Memory-Reference Instructions**
- **Register-Reference Instructions**
- **Input-Output Instructions**

1. **Memory-Reference Instructions:**

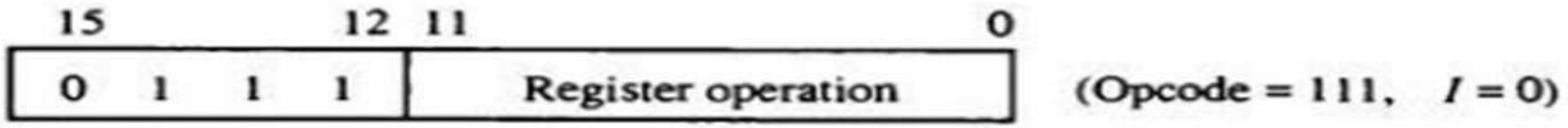


(a) Memory – reference instruction

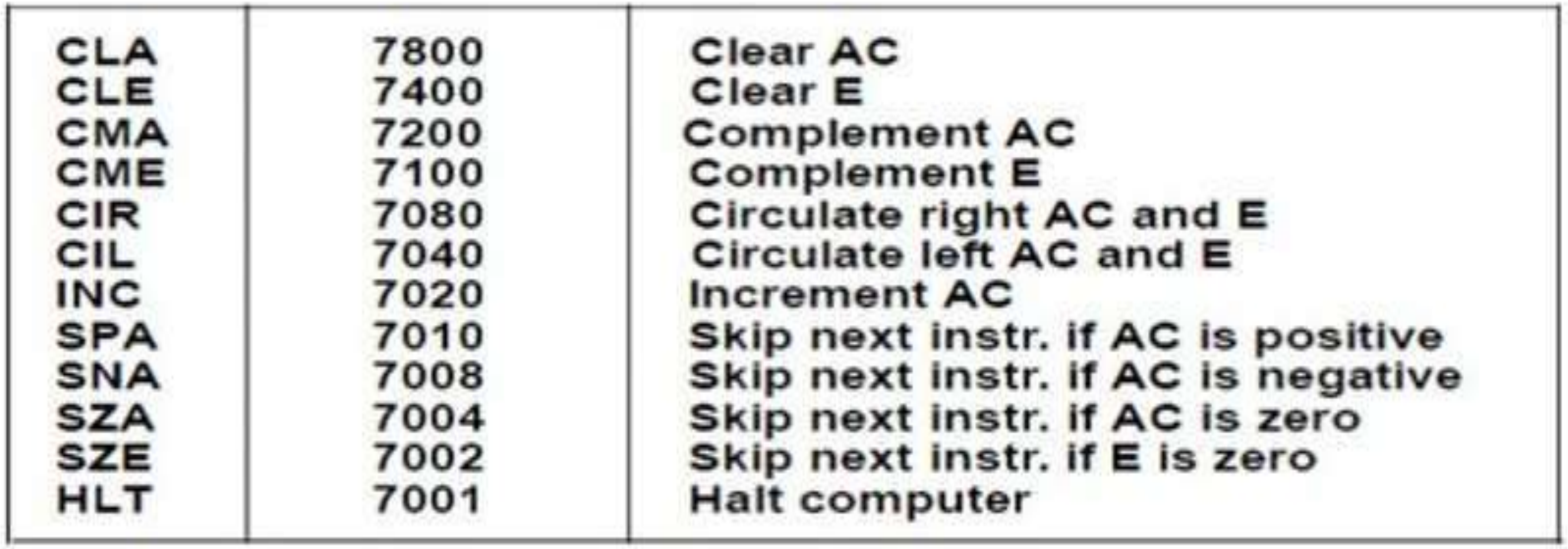


| Symbol | Hex Code I = 0 | Hex Code I = 1 | Description |
|---|---|---|---|
| AND | 0xxx | 8xxx | AND memory word to AC |
| ADD | 1xxx | 9xxx | Add memory word to AC |
| LDA | 2xxx | Axxx | Load AC from memory |
| STA | 3xxx | Bxxx | Store content of AC into memory |
| BUN | 4xxx | Cxxx | Branch unconditionally |
| BSA | 5xxx | Dxxx | Branch and save return address |
| ISZ | 6xxx | Exxx | Increment and skip if zero |

## 2. **Register-Reference Instructions**

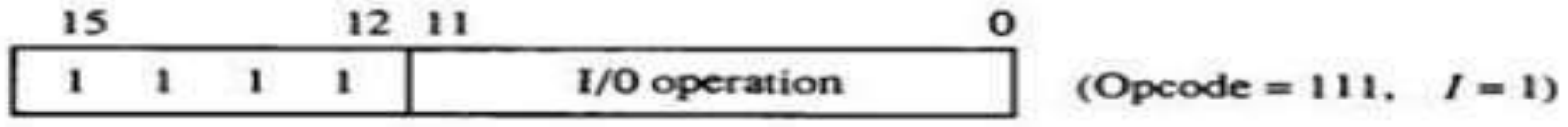


(b) Register – reference instruction



| | | |
|---|---|---|
| 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 instr. if AC is positive |
| SNA | 7008 | Skip next instr. if AC is negative |
| SZA | 7004 | Skip next instr. if AC is zero |
| SZE | 7002 | Skip next instr. if E is zero |
| HLT | 7001 | Halt computer |

## 3. **Input-Output Instructions**



(c) Input – output instruction

| | | |
|---|---|---|
| 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 |

## Instruction Set Completeness

An instruction set is said to be complete if it contains sufficient instructions to perform operations in following categories:

Functional Instructions

- Arithmetic, logic, and shift instructions
- Examples: ADD, CMA, INC, CIR, CIL, AND, CLA

Transfer Instructions

- Data transfers between the main memory and the processor registers
- Examples: LDA, STA

Control Instructions

- Program sequencing and control
- Examples: BUN, BSA, ISZ

Input/output Instructions

- Input and output
- Examples: INP, OUT

***Instruction set of Basic computer is complete*** because:

- ADD, CMA (complement), INC can be used to perform addition and subtraction and CIR (circular right shift), CIL (circular left shift) instructions can be used to achieve any kind of shift operations. Addition subtraction and shifting can be used together to achieve multiplication and division. AND, CMA and CLA (clear accumulator) can be used to achieve any logical operations.
- LDA instruction moves data from memory to register and STA instruction moves data from register to memory.
- The branch instructions BUN, BSA and ISZ together with skip instruction provide the mechanism of program control and sequencing.
- INP instruction is used to read data from input device and OUT instruction is used to send data from processor to output device.

## 3.3Timing and Control Unit

**Control Unit**

Control unit (CU) of a processor translates from machine instructions to the control signals for the microoperations that implement them. There are two types of control organization:

a) Hardwired Control

b) Microprogrammed Control

*a) Hardwired* Control

- CU is made up of sequential and combinational circuits to generate the control signals.
- If logic is changed, we need to change the whole circuit.
- Expensive
- Fast

*b) Microprogrammed Control*

- A control memory on the processor contains microprograms that activate the necessary control signals.
- If logic is changed, we only need to change the microprogram.
- Cheap
- Slow

# Control Unit of a Basic Computer (Hardwired Control)

The block diagram of a hardwired control unit is shown below. It consists of two decoders, a sequence counter, and a number of control logic gates.

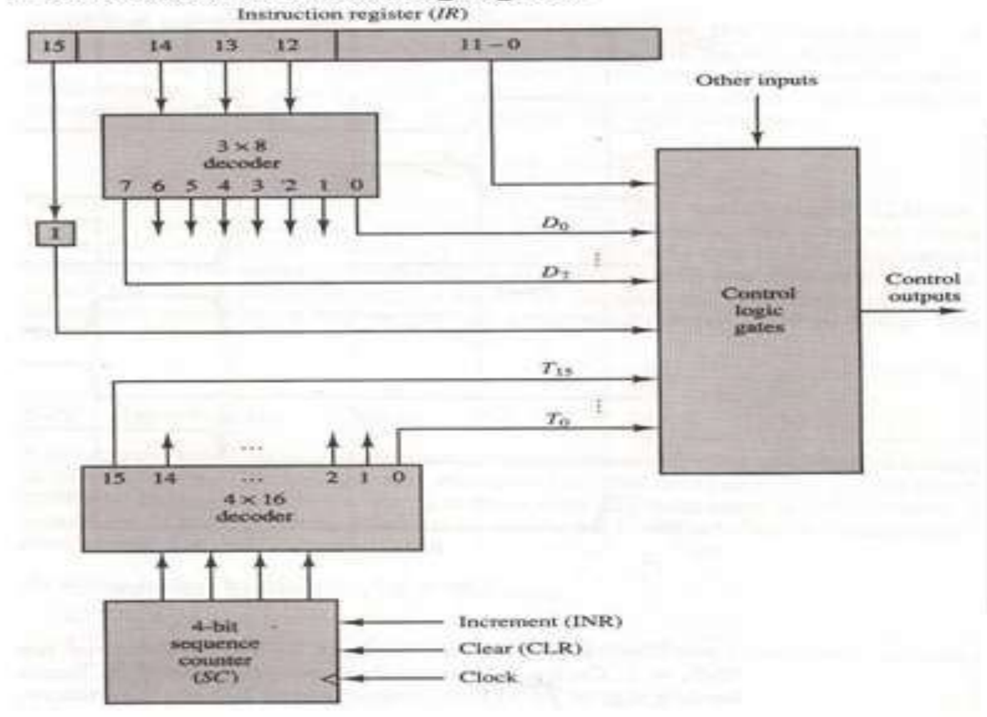
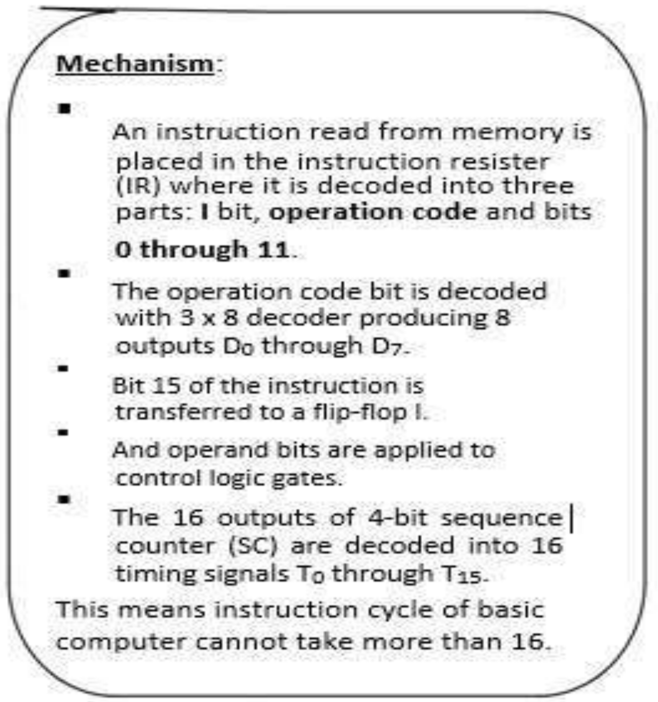


Fig: Control unit of a basic computer



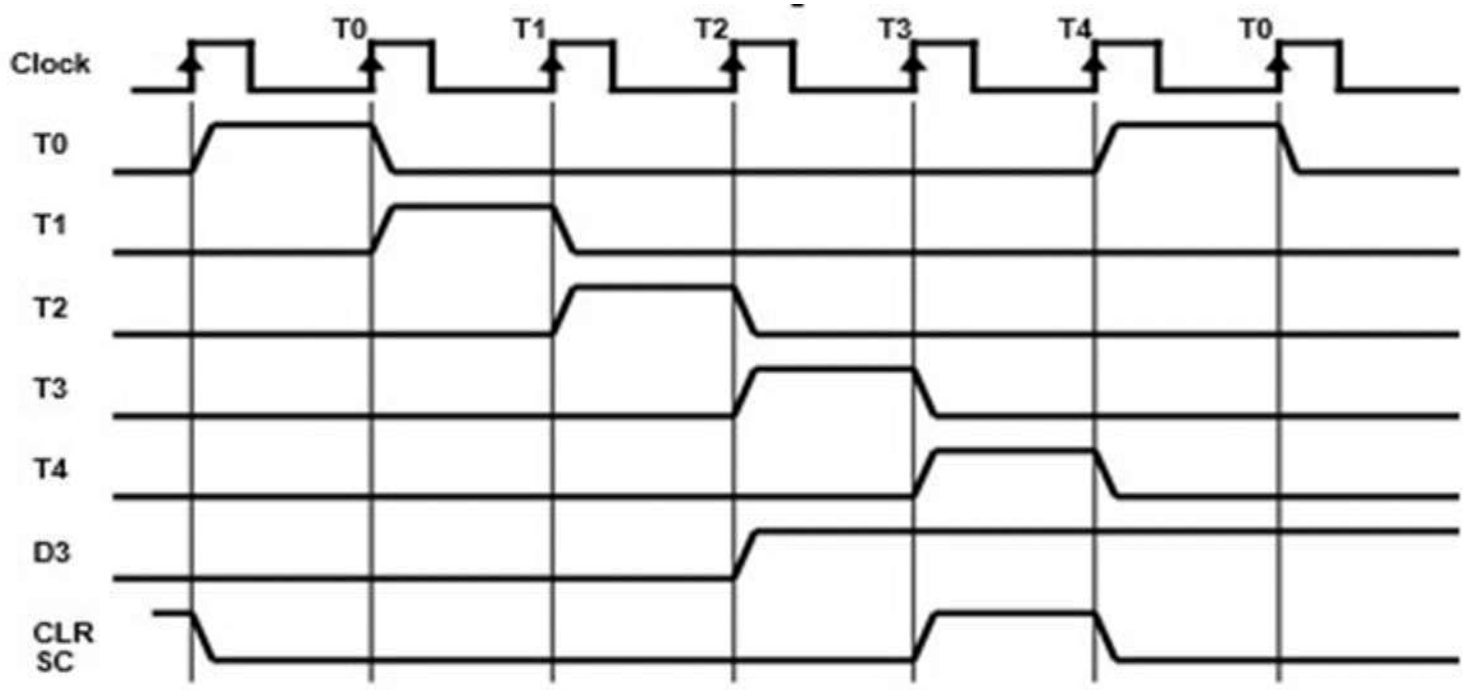
**Mechanism**:

- An instruction read from memory is placed in the instruction resister (IR) where it is decoded into three parts: **I** bit, **operation code** and bits **0 through 11**.
- The operation code bit is decoded with 3 x 8 decoder producing 8 outputs $D_0$ through $D_7$.
- Bit 15 of the instruction is transferred to a flip-flop I.
- And operand bits are applied to control logic gates.
- The 16 outputs of 4-bit sequence counter (SC) are decoded into 16 timing signals $T_0$ through $T_{15}$.

This means instruction cycle of basic computer cannot take more than 16.

# Timing Signal

- Generated by 4-bit sequence counter and 4x16 decoder.
- The SC can be incremented or cleared.
- Example: $T_0$, $T_1$, $T_2$, $T_3$, $T_4$, $T_0$, $T_1$ . . .
- Assume: At time T4, SC is cleared to 0 if decoder output D3 is active:
- $D_3T_4$: SC$\longleftarrow$ 0



## 3.5 Instruction Cycle of Basic Computer

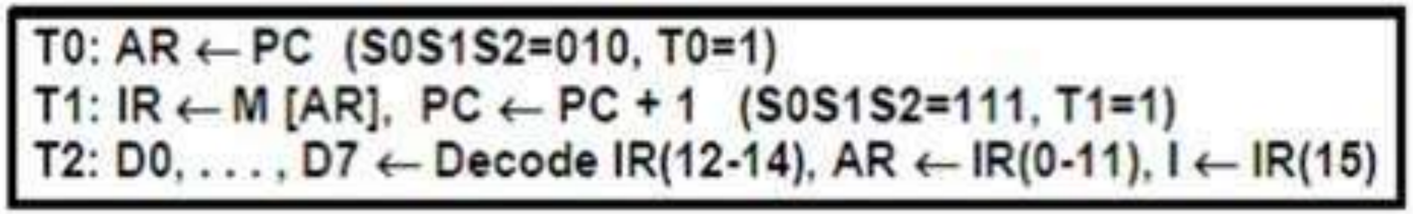
Processing required for complete execution of an instruction is called instruction cycle.

In Basic Computer, a machine instruction is executed in the following cycle:

- Fetch an instruction from memory
- Decode the instruction
- Read the effective address from memory if the instruction has an indirect address
- Execute the instruction
- Upon the completion of step 4, control goes back to step 1 to fetch, decode and execute the next instruction. This process is continued indefinitely until HALT instruction is encountered.

## Fetch and Decode

Sequence of steps required for fetching instruction from memory to CPU internal register is known as fetch cycle. The microoperations for the fetch and decode phases can be specified by the following resister transfer statements: (first two steps for fetch and third step for decode)



T0: AR ← PC (S0S1S2=010, T0=1)
T1: IR ← M [AR], PC ← PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)

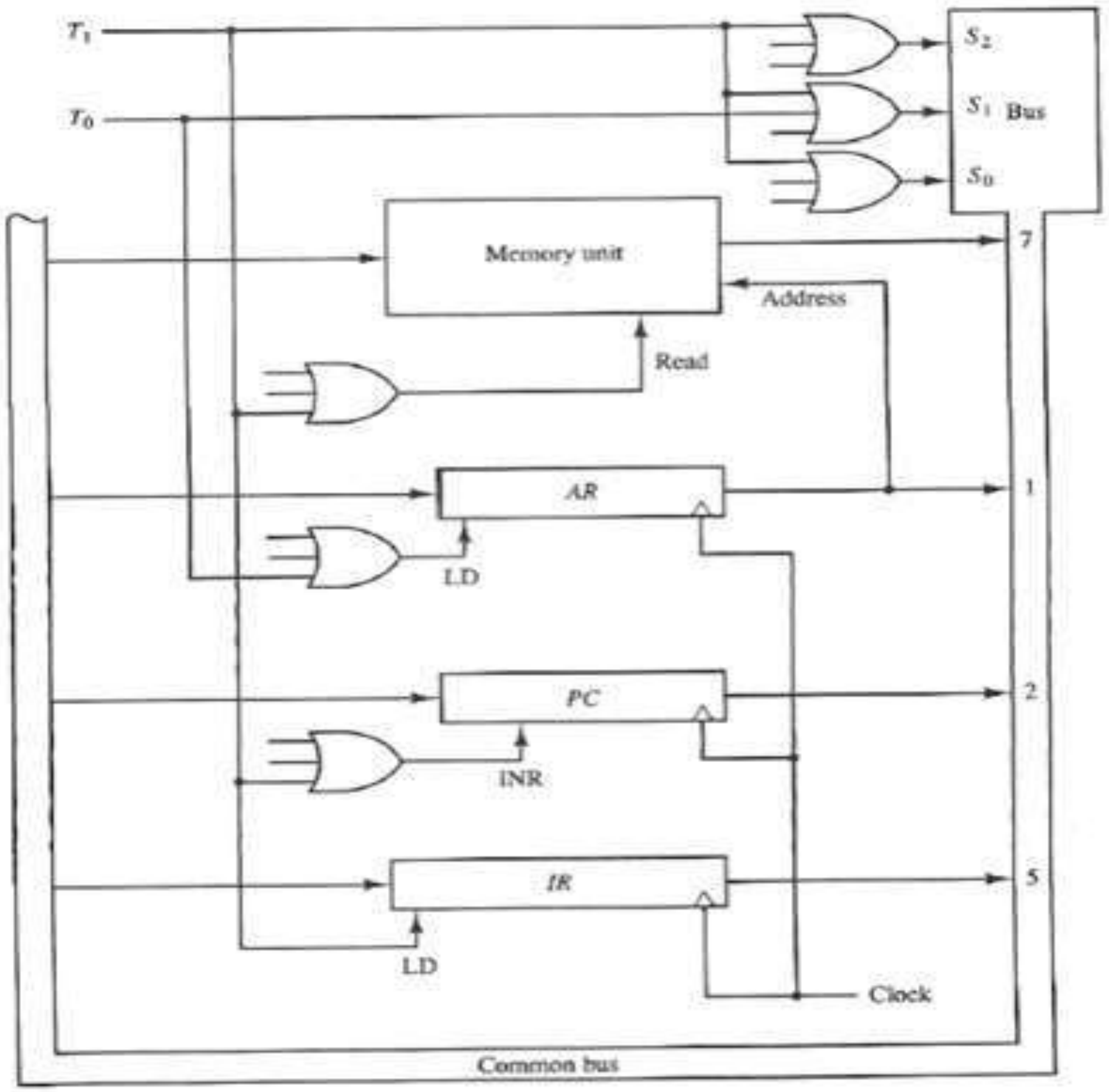


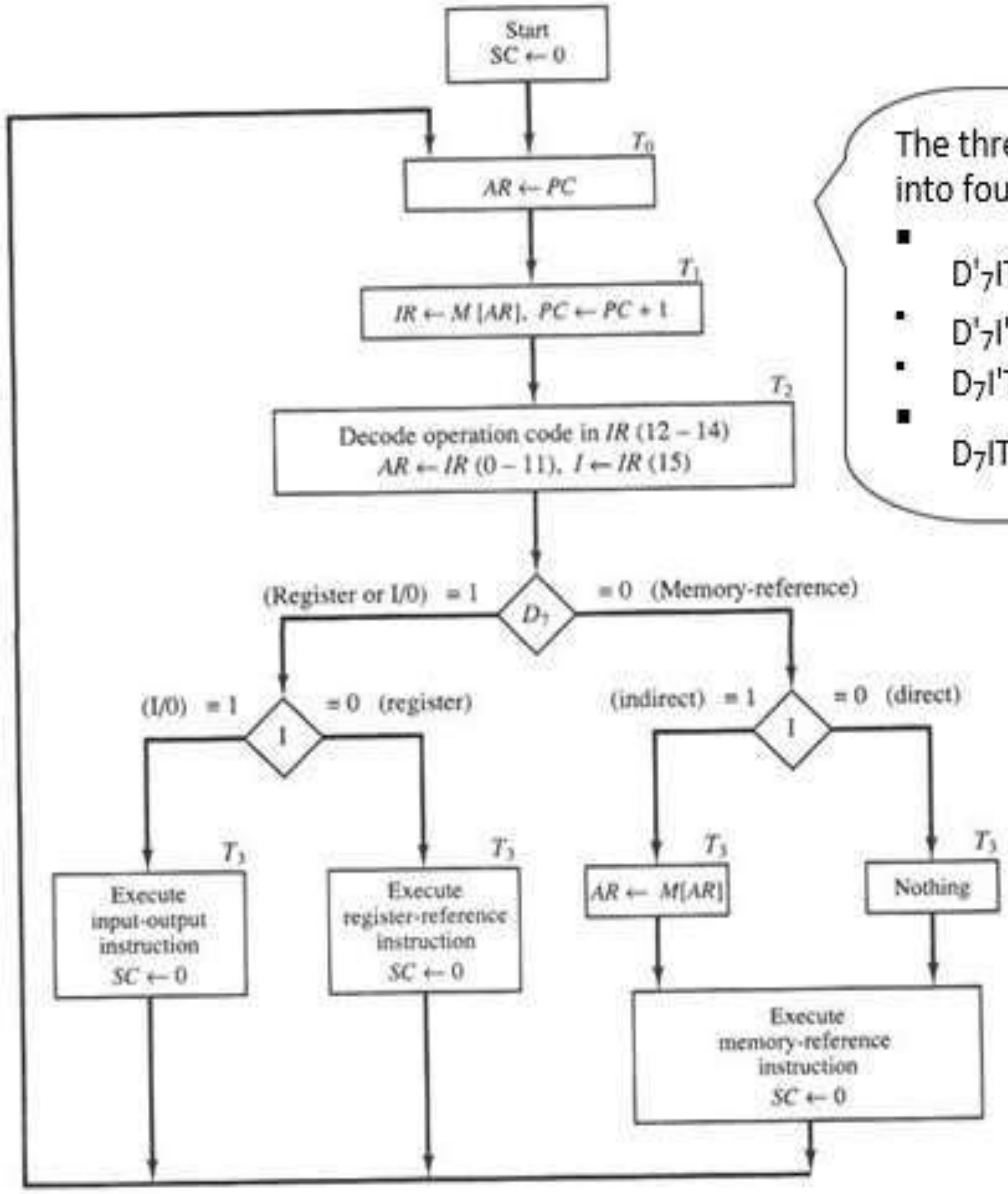
Fig: Resister transfers for the fetch phase

It is necessary to transfer the address from PC to AR during clock transition associated with the timing signal $T_0$. The instruction read from memory is then placed in IR with clock transition associated with the timing signal $T_1$. At the same time, PC is incremented by one to prepare for the next instruction in the program. At time $T_2$, the opcode 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**: SC is incremented after each clock pulse to produce the sequence $T_0$, $T_1$ and $T_2$.

- **Determine the type of the instruction**

The timing signal that is active after decoding is $T_3$. During time $T_3$, the control unit determines the type of instruction that was just read from memory. Following flowchart presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after decoding.



The three instruction types are subdivided into four separate paths:

- $D'_7IT_3$: AR ← M[AR] (Indirect address)
- $D'_7I'T_3$: Nothing
- $D_7I'T_3$: Execute register-reference instrs.
- $D_7IT_3$: Execute input-output instructions.

Fig: Flowchart for instruction cycle (Initial configuration)

Then, among decoded, D7 determines which type of instruction.

1) If D7 = 1, it will be either register-reference or input-output instruction.

- If I = 1, input-output instruction is executed during T3.
- If I = 0, register-reference instruction is executed during T3.

2) If D7 = 0, it will be memory-reference instruction.

- If I = 1, indirect addressing mode instruction during T3.
- If I = 0, direct addressing mode instruction during T3.

The SC is reset after executing each instruction.

## Register-Reference Instructions

Register Reference Instructions are identified when

- - D7 = 1, I = 0
- - Register Ref. Instr. is specified in b0 ~ b11 of IR
- - Execution starts with timing signal T3
- Let r = D7 I'T3 => Common to all Register Reference Instruction
- Bi = IR (i), i=0, 1, 2... 11. [Bit in IR(0-11) that specifies the operation]

TABLE 5-3 Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0\text{–}11)$ 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 |

## Memory-reference instructions

- Once an instruction has been loaded to IR, it may require further access to memory to perform its intended function (direct or indirect).
- The effective address of the instruction is in the AR and was placed their during:

- Time signal T2 when I = 0 or

- Time signal T3 when I = 1

- Execution of memory reference instructions starts with the timing signal T4.
- ⍰ Described symbolically using RTL.

TABLE 5-4 Memory-Reference Instructions

| 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 $PC \leftarrow PC + 1$ |

**AND to AC**

This instruction performs the AND logical operation on pairs of bits on AC and the memory word specified by the effective address. The result is transferred to AC. Microoperations that execute these instructions are:

| | | |
|---|---|---|
| $D_0T_4$: | $DR \leftarrow M[AR]$ | //Read operand |
| $D_0T_5$: | $AC \leftarrow AC \wedge DR, SC \leftarrow 0$ | //AND with AC |

**ADD to AC**

| | | |
|---|---|---|
| $D_1T_4$: | $DR \leftarrow M[AR]$ | //Read operand |
| $D_1T_5$: | $AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$ | //Add to AC and stores carry in E |

**LDA: Load to AC**

| | | |
|---|---|---|
| $D_2T_4$: | $DR \leftarrow M[AR]$ | //Read operand |
| $D_2T_5$: | $AC \leftarrow DR, SC \leftarrow 0$ | //Load AC with DR |

**STA: Store AC**

$D_3T_4$: $M[AR] \leftarrow AC, SC \leftarrow 0$ // store data into memory location

**BUN: Branch Unconditionally**

$D_4T_4$: $PC \leftarrow AR, SC \leftarrow 0$ //Branch to specified address

**BSA: Branch and Save Return Address**

$D_5T_4$: $M[AR] \leftarrow PC,\ AR \leftarrow AR + 1$ // save return address and increment AR

$D_5T_5$: $PC \leftarrow AR, SC \leftarrow 0$ // load PC with AR

**ISZ: Increment and Skip-if-Zero**

$D_6T_4$: $DR \leftarrow M[AR]$ //Load data into DR

$D_6T_5$: $DR \leftarrow DR + 1$ // Increment the data

$D_6T_4$: $M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

// if DR=0 skip next instruction by incrementing PC

## Input-Output Configuration

The terminal sends and receives serial information. Each quantity of information has 8 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.

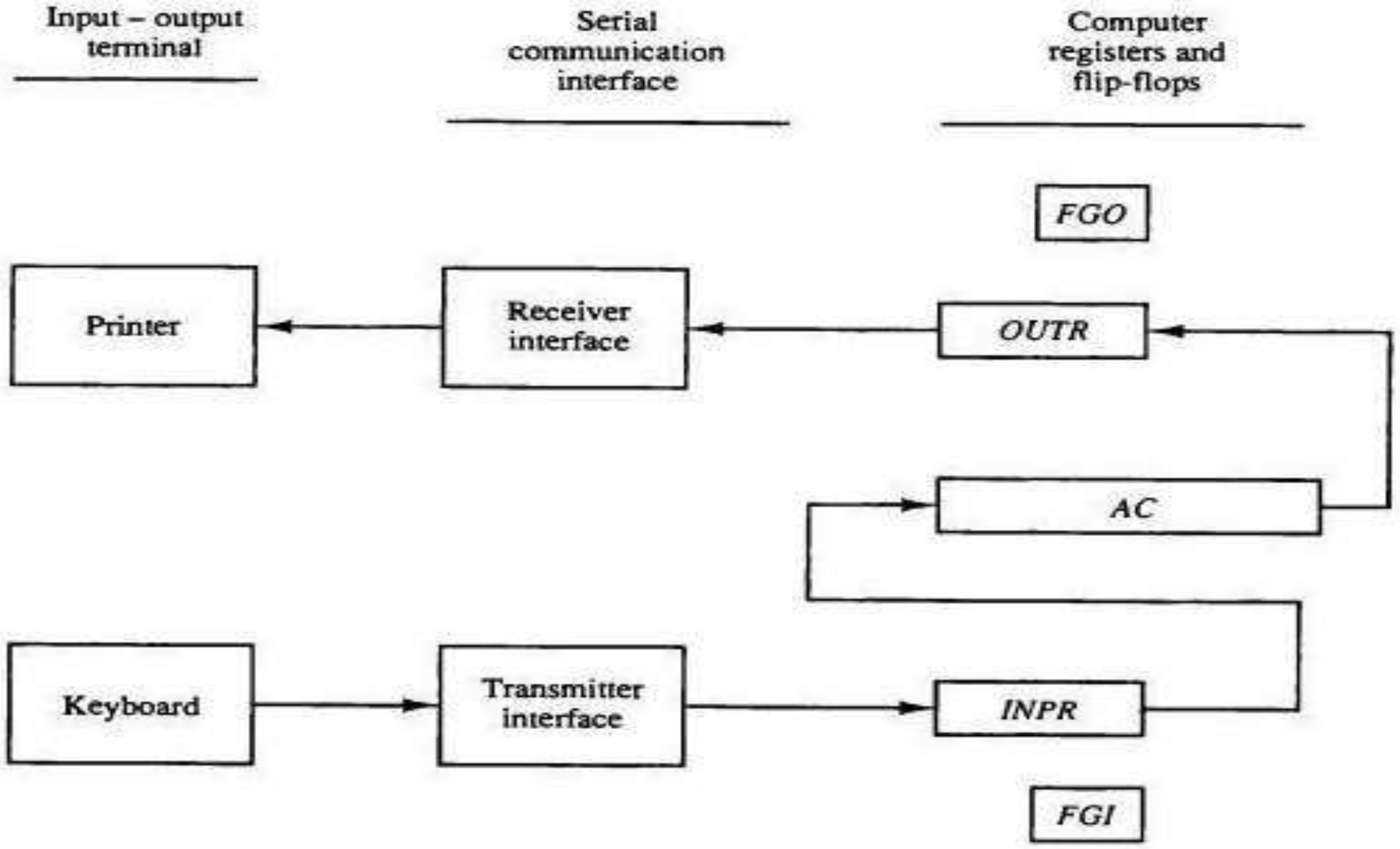


Fig: Input-Output Configuration

Scenario1: 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 control checks the flag bit, if 1, contents of INPR is transferred in parallel to AC and FGI is cleared to 0. Once the flag is cleared, new information can be shifted into INPR by striking another key.

Scenario2: OUTR works similarly but the direction of information flow is reversed. Initially FGO is set to 1. The computer checks the flag bit; if it is 1, the information 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 operation is completed, it sets FGO to 1.

**Input-output Instructions :** I/O instructions are needed to transferring information to and form AC register, for checking the flag bits and for controlling the interrupt facility.

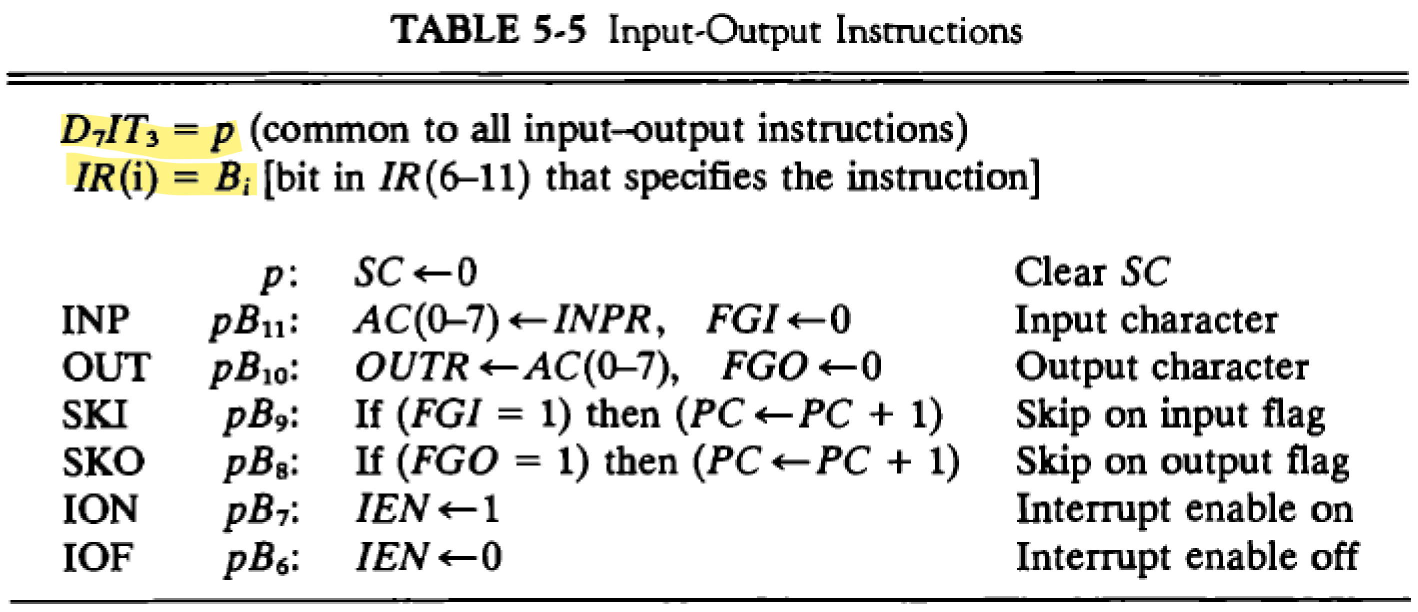


**TABLE 5-5** Input-Output Instructions

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

| | | | |
|---|---|---|---|
| | $p$: | $SC \leftarrow 0$ | Clear $SC$ |
| INP | $pB_{11}$: | $AC(0\text{–}7) \leftarrow INPR, \quad FGI \leftarrow 0$ | Input character |
| OUT | $pB_{10}$: | $OUTR \leftarrow AC(0\text{–}7), \quad FGO \leftarrow 0$ | 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 |

## Program Interrupt

- Input and Output interactions with electromechanical peripheral devices require huge processing times compared with CPU processing times – I/O (milliseconds) versus CPU (nano/micro-seconds)
- Interrupts permit other CPU instructions to execute while waiting for I/O to complete
- The I/O interface, instead of the CPU, monitors the I/O device.
- When the interface founds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU
- Upon detecting an interrupt, the CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing.

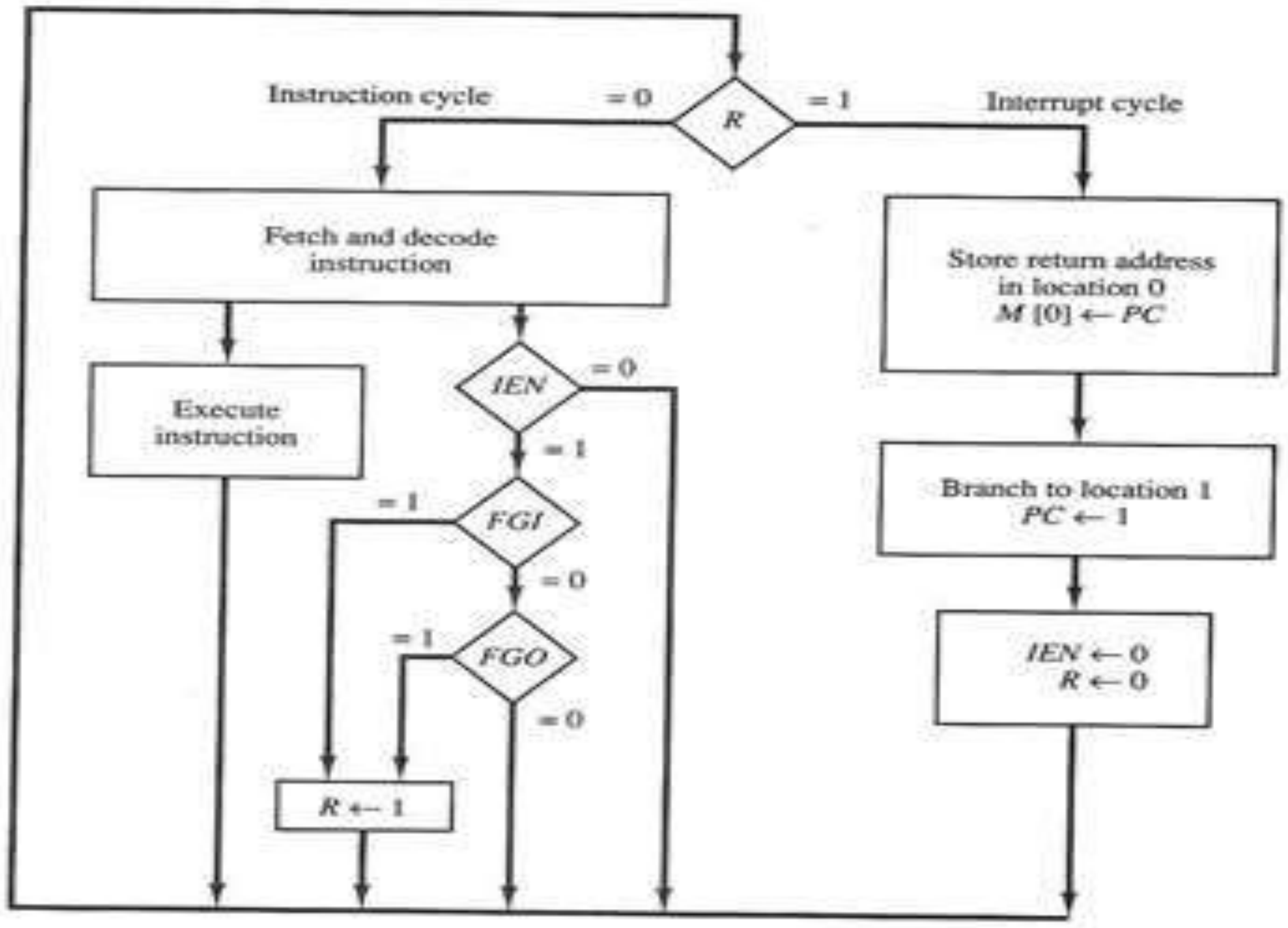


Fig: flowchart of interrupt cycle

## Description

- The flowchart consists of R – an interrupt flip flop where if R=0 it proceeds to instruction cycle and if R=1, it proceeds to interrupt cycle.
- When R=0, the computer goes through an instruction cycle. During the execution phase of the instruction cycle, IEN (Interrupt enable flip flop) 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= 1, control checks the flag bit, If both flags are 0, it indicates that neither the input nor the output registers are ready for the 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 1, it goes to an interrupt cycle instead of an instruction cycle.

# Continue…..

- When R=1, the interrupt instruction executes the following steps:

a) Save the current program and return to the interrupt where PC proceed to zero memory.

$$M[0] \leftarrow PC$$

b) Control then issues address 1 into PC. PC <- 1

c) Finally, interrupt is finished or cleared then IEN=0, R=0, further proceed to instruction cycle. IEN <- 0, R<-0

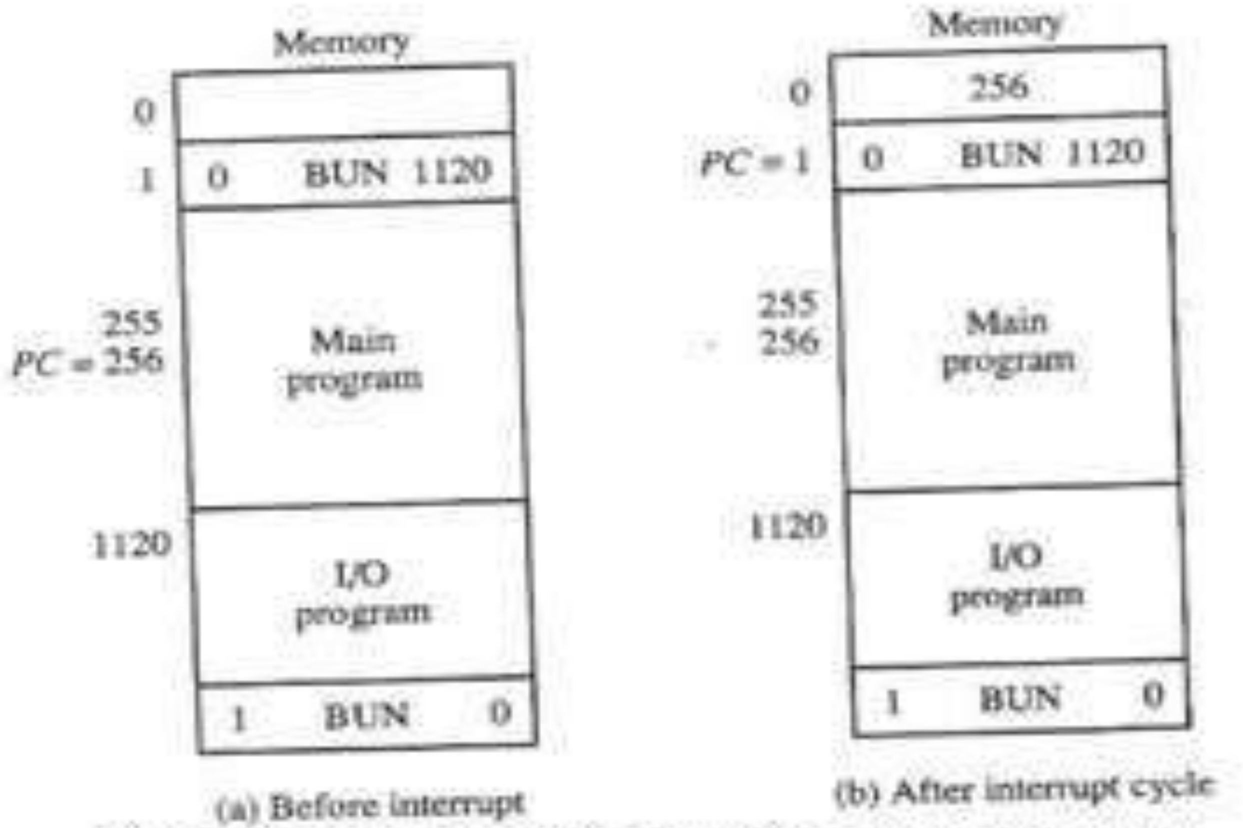
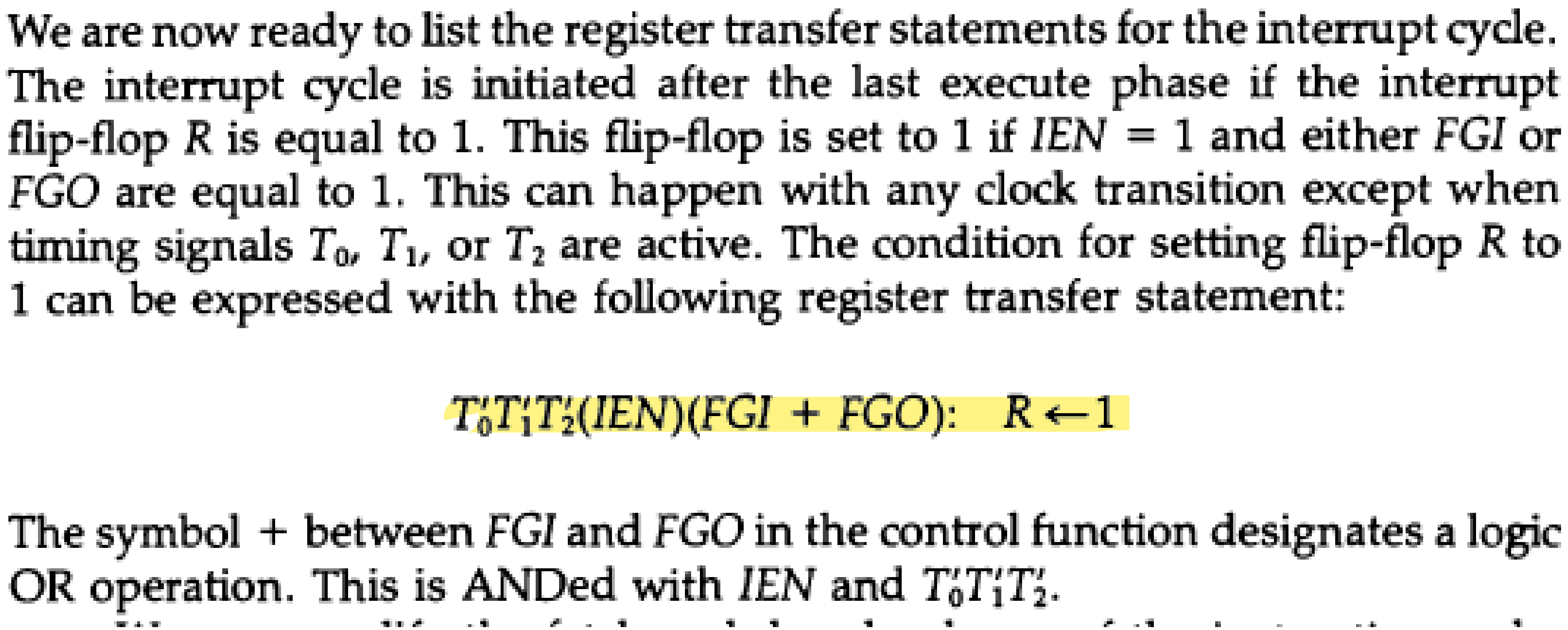


Fig: Demonstration of interrupt cycle

An example that shows what happens during the interrupt cycle is shown in Fig. 5-14. 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 Fig. 5-14(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 Fig. 5-14(b).

# Register Transfer in Interrupt Cycle



We are now ready to list the register transfer statements for the interrupt cycle. The interrupt cycle is initiated after the last execute phase if the interrupt flip-flop $R$ is equal to 1. This flip-flop is set to 1 if $IEN = 1$ and either $FGI$ or $FGO$ are equal to 1. This can happen with any clock transition except when timing signals $T_0$, $T_1$, or $T_2$ are active. The condition for setting flip-flop $R$ to 1 can be expressed with the following register transfer statement:

$$T_0'T_1'T_2'(IEN)(FGI + FGO): \quad R \leftarrow 1$$

The symbol + between $FGI$ and $FGO$ in the control function designates a logic OR operation. This is ANDed with $IEN$ and $T_0'T_1'T_2'$.

We now modify the fetch and decode phases of the instruction cycle. Instead of using only timing signals $T_0$, $T_1$, and $T_2$ (as shown in Fig. 5-9) we will AND the three timing signals with $R'$ so that the fetch and decode phases will be recognized from the three control functions $R'T_0$, $R'T_1$, and $R'T_2$. The reason for this is that after the instruction is executed and $SC$ is cleared to 0, the control will go through a fetch phase only if $R = 0$. Otherwise, if $R = 1$, the control will go through an interrupt cycle. The interrupt cycle stores the return address (available in $PC$) into memory location 0, branches to memory location 1, and clears $IEN$, $R$, and $SC$ to 0. This can be done with the following sequence of microoperations:

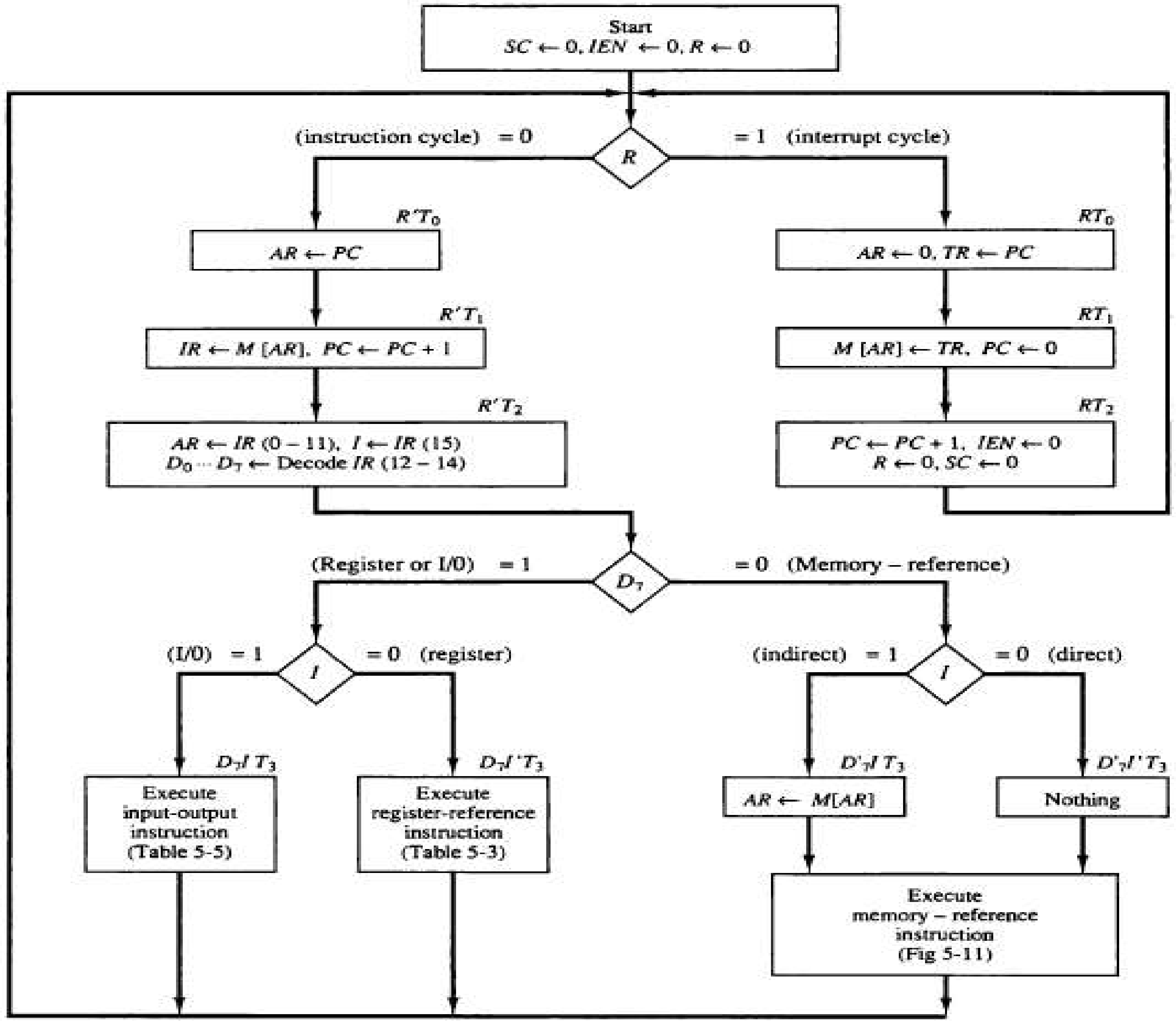
✓
$$\begin{aligned}
&RT_0: \quad AR \leftarrow 0, \quad TR \leftarrow PC\\
&RT_1: \quad M[AR] \leftarrow TR, \quad PC \leftarrow 0\\
&RT_2: \quad PC \leftarrow PC + 1, \quad IEN \leftarrow 0, \quad R \leftarrow 0, \quad SC \leftarrow 0
\end{aligned}$$

During the first timing signal $AR$ is cleared to 0, and the content of $PC$ is transferred to the temporary register $TR$. With the second timing signal, the return address is stored in memory at location 0 and $PC$ is cleared to 0. The third timing signal increments $PC$ to 1, clears $IEN$ and $R$, and control goes back to $T_0$ by clearing $SC$ to 0. The beginning of the next instruction cycle has the condition $R'T_0$ and the content of $PC$ is equal to 1. The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.

## Design of Basic Computer

The basic computer consists of the following hardware components:

- ➢A memory unit with 4096 words of 16 bits each.
- ➢Nine registers: AR(Address Reg.), PC (Program Counter), DR(Data Reg.), AC (Accumulator), IR (Instruction Reg.), TR (Temp. Reg.),
- ➢OUTR (Output Reg.), INPR (Input Reg.), and SC (Sequence Counter).
- ➢Flip-flops: lEN (Interrupt Enable), FGI (Input Flag), and FGO (Output Flag).
- ➢Two decoders: a 3 x 8 operation decoder and a 4 x 16 timing decoder
- ➢A 16-bit common bus.
- ➢Control logic gates.
- ➢Adder and logic circuit connected to the input of AC.


Start
$SC \leftarrow 0, IEN \leftarrow 0, R \leftarrow 0$
(instruction cycle) = 0
$R$
= 1 (interrupt cycle)
$R'T_0$
$AR \leftarrow PC$
$R'T_1$
$IR \leftarrow M[AR], PC \leftarrow PC + 1$
$R'T_2$
$AR \leftarrow IR(0-11), I \leftarrow IR(15)$
$D_0 \cdots D_7 \leftarrow$ Decode $IR(12-14)$
$RT_0$
$AR \leftarrow 0, TR \leftarrow PC$
$RT_1$
$M[AR] \leftarrow TR, PC \leftarrow 0$
$RT_2$
$PC \leftarrow PC + 1, IEN \leftarrow 0$
$R \leftarrow 0, SC \leftarrow 0$
(Register or I/0) = 1
$D_7$
= 0 (Memory – reference)
(I/0) = 1
$I$
= 0 (register)
(indirect) = 1
$I$
= 0 (direct)
$D_7IT_3$
Execute input-output instruction (Table 5-5)
$D_7I'T_3$
Execute register-reference instruction (Table 5-3)
$D'_7IT_3$
$AR \leftarrow M[AR]$
$D'_7I'T_3$
Nothing
Execute memory – reference instruction (Fig 5-11)

# Control Functions and Microoperations of Basic Computer

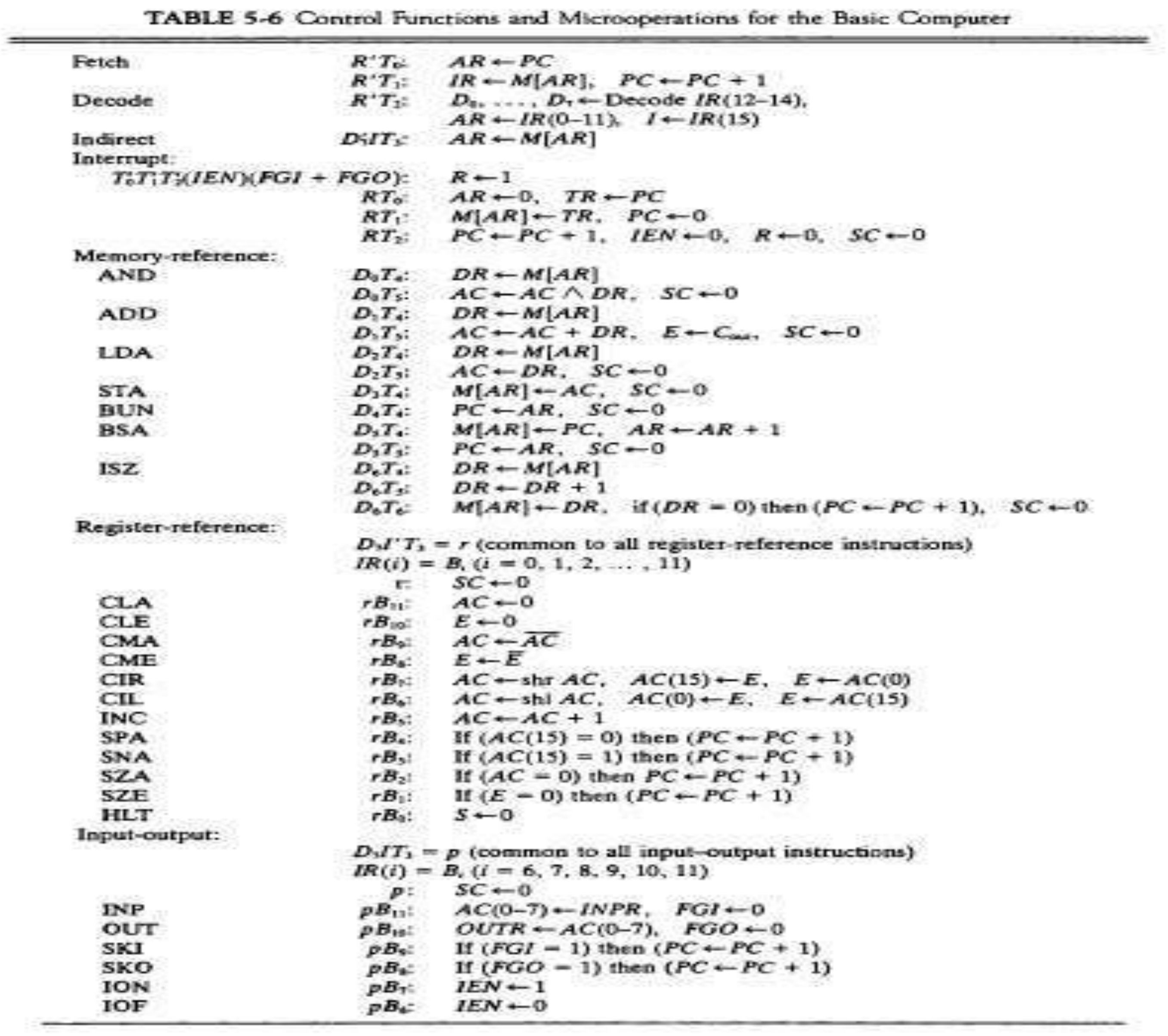


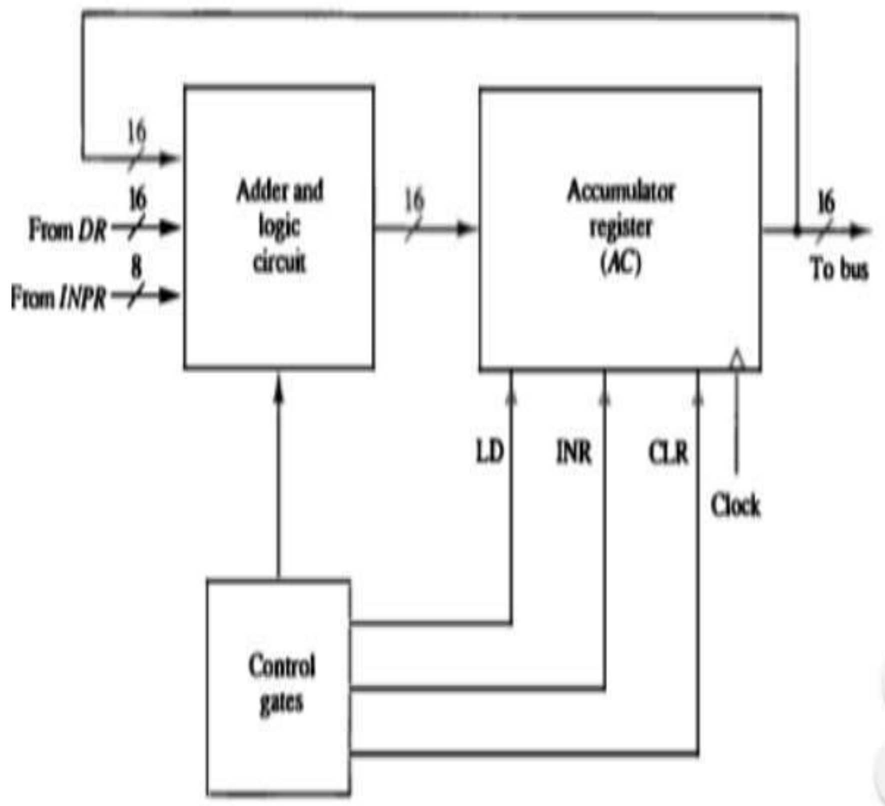
TABLE 5-6 Control Functions and Microoperations for the Basic Computer

| | | |
|---|---|---|
| Fetch | $R'T_0$: | $AR \leftarrow PC$ |
| | $R'T_1$: | $IR \leftarrow M[AR], \quad PC \leftarrow PC + 1$ |
| Decode | $R'T_2$: | $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, |
| | | $AR \leftarrow IR(0\text{–}11), \quad I \leftarrow IR(15)$ |
| Indirect | $D_7'IT_3$: | $AR \leftarrow M[AR]$ |
| Interrupt: | | |
| $T_0'T_1'T_2'(IEN)(FGI + FGO)$: | | $R \leftarrow 1$ |
| | $RT_0$: | $AR \leftarrow 0, \quad TR \leftarrow PC$ |
| | $RT_1$: | $M[AR] \leftarrow TR, \quad PC \leftarrow 0$ |
| | $RT_2$: | $PC \leftarrow PC + 1, \quad IEN \leftarrow 0, \quad R \leftarrow 0, \quad SC \leftarrow 0$ |
| Memory-reference: | | |
| AND | $D_0T_4$: | $DR \leftarrow M[AR]$ |
| | $D_0T_5$: | $AC \leftarrow AC \wedge DR, \quad SC \leftarrow 0$ |
| ADD | $D_1T_4$: | $DR \leftarrow M[AR]$ |
| | $D_1T_5$: | $AC \leftarrow AC + DR, \quad E \leftarrow C_{out}, \quad SC \leftarrow 0$ |
| LDA | $D_2T_4$: | $DR \leftarrow M[AR]$ |
| | $D_2T_5$: | $AC \leftarrow DR, \quad SC \leftarrow 0$ |
| STA | $D_3T_4$: | $M[AR] \leftarrow AC, \quad SC \leftarrow 0$ |
| BUN | $D_4T_4$: | $PC \leftarrow AR, \quad SC \leftarrow 0$ |
| BSA | $D_5T_4$: | $M[AR] \leftarrow PC, \quad AR \leftarrow AR + 1$ |
| | $D_5T_5$: | $PC \leftarrow AR, \quad SC \leftarrow 0$ |
| ISZ | $D_6T_4$: | $DR \leftarrow M[AR]$ |
| | $D_6T_5$: | $DR \leftarrow DR + 1$ |
| | $D_6T_6$: | $M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$ |
| Register-reference: | | |
| | $D_7I'T_3 = r$ (common to all register-reference instructions) | |
| | $IR(i) = B_i \; (i = 0, 1, 2, \ldots, 11)$ | |
| | $r$: | $SC \leftarrow 0$ |
| CLA | $rB_{11}$: | $AC \leftarrow 0$ |
| CLE | $rB_{10}$: | $E \leftarrow 0$ |
| CMA | $rB_9$: | $AC \leftarrow \overline{AC}$ |
| CME | $rB_8$: | $E \leftarrow \overline{E}$ |
| CIR | $rB_7$: | $AC \leftarrow \text{shr } AC, \quad AC(15) \leftarrow E, \quad E \leftarrow AC(0)$ |
| CIL | $rB_6$: | $AC \leftarrow \text{shl } AC, \quad AC(0) \leftarrow E, \quad E \leftarrow AC(15)$ |
| INC | $rB_5$: | $AC \leftarrow AC + 1$ |
| SPA | $rB_4$: | If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$ |
| SNA | $rB_3$: | If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$ |
| SZA | $rB_2$: | If $(AC = 0)$ then $PC \leftarrow PC + 1)$ |
| SZE | $rB_1$: | If $(E = 0)$ then $(PC \leftarrow PC + 1)$ |
| HLT | $rB_0$: | $S \leftarrow 0$ |
| Input-output: | | |
| | $D_7IT_3 = p$ (common to all input–output instructions) | |
| | $IR(i) = B_i \; (i = 6, 7, 8, 9, 10, 11)$ | |
| | $p$: | $SC \leftarrow 0$ |
| INP | $pB_{11}$: | $AC(0\text{–}7) \leftarrow INPR, \quad FGI \leftarrow 0$ |
| OUT | $pB_{10}$: | $OUTR \leftarrow AC(0\text{–}7), \quad FGO \leftarrow 0$ |
| SKI | $pB_9$: | If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$ |
| SKO | $pB_8$: | If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$ |
| ION | $pB_7$: | $IEN \leftarrow 1$ |
| IOF | $pB_6$: | $IEN \leftarrow 0$ |

## Design of Accumulator Logic

- The circuits associated with the AC register are shown in Fig. The adder and logic circuit has three sets of inputs.
- One set of 16 inputs comes from the outputs of AC.
- Another set of 16 inputs comes from the data register DR.
- A third set of eight inputs comes from the input register INPR.
- The outputs of the adder and logic circuit provide the data inputs for the register. In addition, it is necessary to include logic gates for controlling the LD, INR, and CLR in the register and for controlling the operation of the adder and logic circuit.

Figure 5-19 Circuits associated with AC.



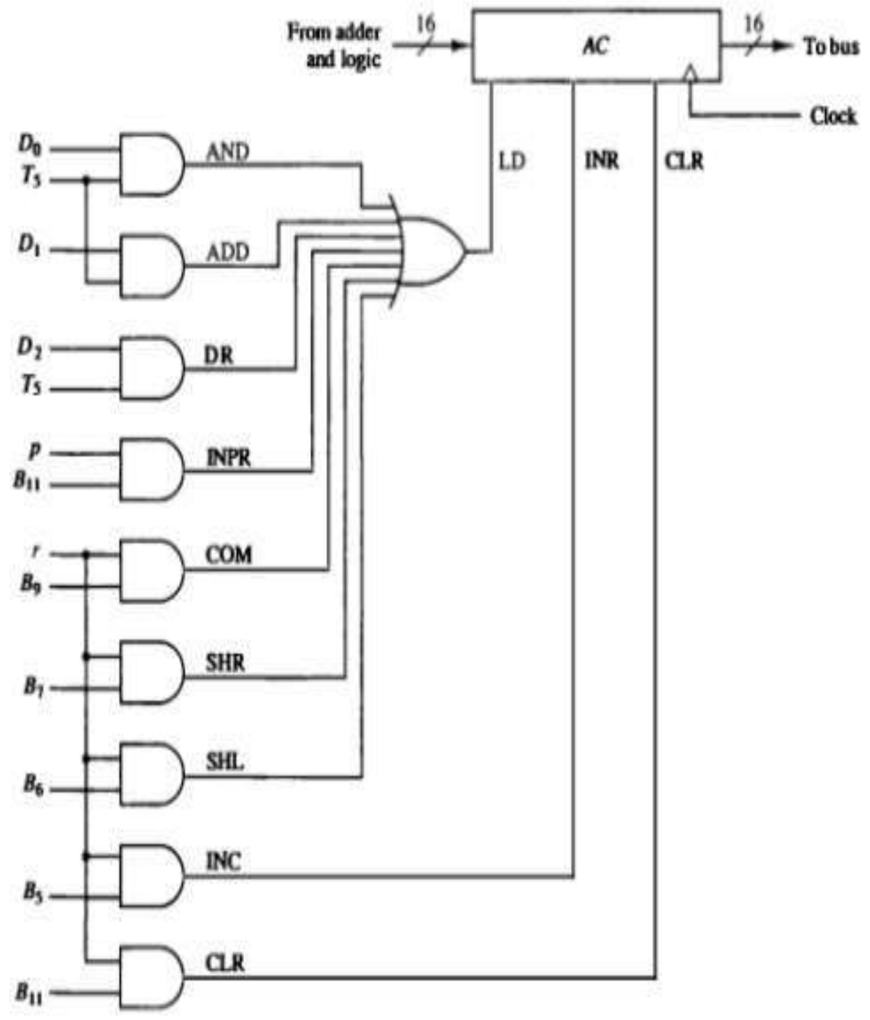
In order to design the logic associated with *AC*, it is necessary to go over the register transfer statements in Table and extract all the statements that change the content of *AC*.

| | | |
|---|---|---|
| $D_0T_5$: | $AC \leftarrow AC \wedge DR$ | AND with *DR* |
| $D_1T_5$: | $AC \leftarrow AC + DR$ | Add with *DR* |
| $D_2T_5$: | $AC \leftarrow DR$ | Transfer from *DR* |
| $pB_{11}$: | $AC(0\text{–}7) \leftarrow INPR$ | Transfer from *INPR* |
| $rB_9$: | $AC \leftarrow \overline{AC}$ | Complement |
| $rB_7$: | $AC \leftarrow \text{shr } AC, \quad AC(15) \leftarrow E$ | Shift right |
| $rB_6$: | $AC \leftarrow \text{shl } AC, \quad AC(0) \leftarrow E$ | Shift left |
| $rB_{11}$: | $AC \leftarrow 0$ | Clear |
| $rB_5$: | $AC \leftarrow AC + 1$ | Increment |

# Control of AC Register – Gate Structure

- The gate structure that controls the LD, INR, and CLR inputs of AC is shown in Fig.
- The output of the AND gate that generates this control function is connected to the CLR input of the register.
- Similarly, the output of the gate that implements the increment micro operation is connected to the INR input of the register.
- The other seven micro operations are generated in the adder and logic circuit and are loaded into AC at the proper time.
- The outputs of the gates for each control function is marked with a symbolic name. These outputs are used in the design of the adder and logic circuit.

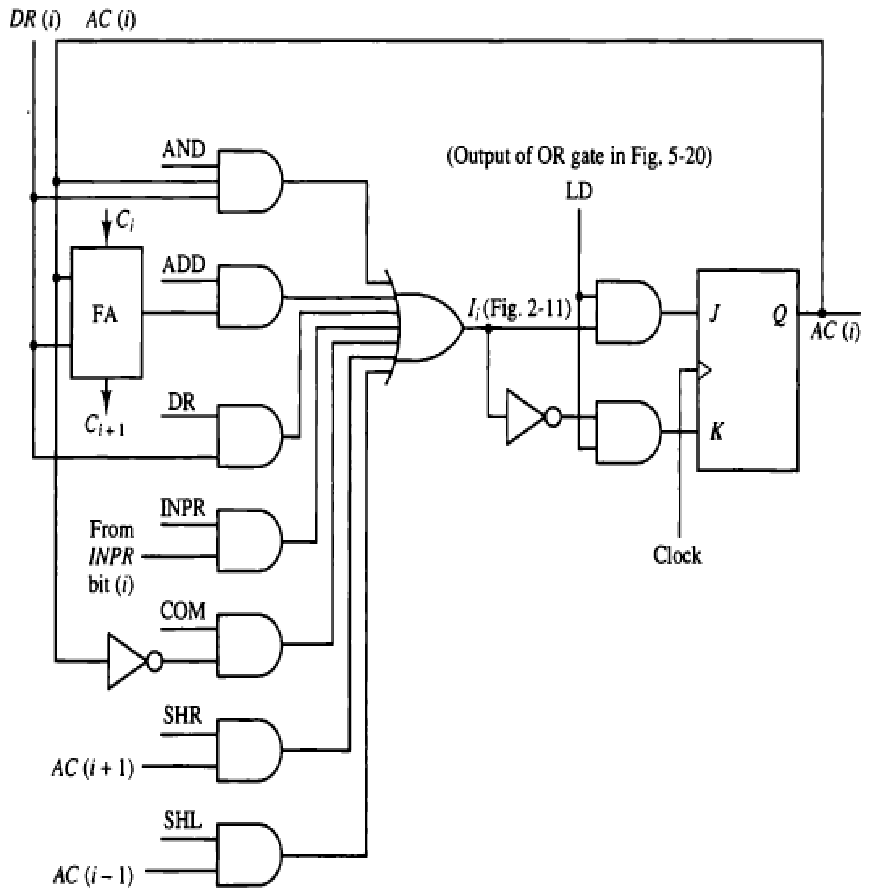
Figure 5-20 Gate structure for controlling the LD, INR, and CLR of AC.



# Adder and Logic Circuit

The adder and logic circuit can be sub-divided into 16 stages with each stage corresponding to one bit of AC. The load (LD) input is connected to the inputs of the AND gate.

Figure 5-21 One stage of adder and logic circuit.


DR (i)
AC (i)
AND
$C_i$
ADD
FA
$C_{i+1}$
DR
INPR
From INPR bit (i)
COM
SHR
AC (i + 1)
SHL
AC (i − 1)
(Output of OR gate in Fig. 5-20)
LD
$I_i$ (Fig. 2-11)
J
Q
K
AC (i)
Clock