# Unit : 2
# Basic Computer Organization and Design

## Instruction Code :

An instruction code is a group of bits that instruct the computer to perform a specific operation.


## Operation Code :

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 2n (or less) distinct operations.

## Accumulator (AC)

Computers that have a single-processor register usually assign to it the name accumulator (AC) accumulator and label it AC.

The operation is performed with the memory operand and the content of AC.

# Stored Program Organization :

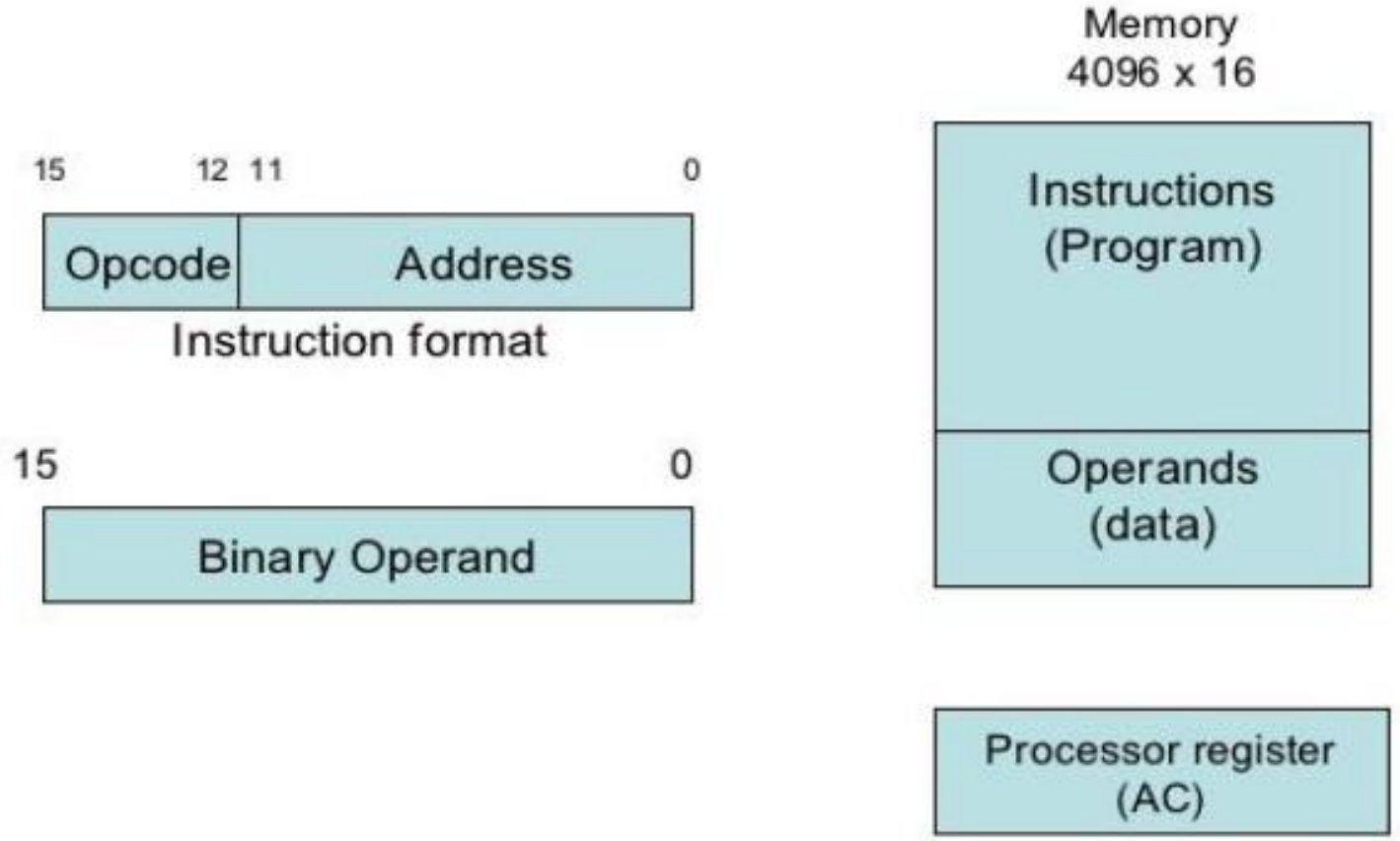


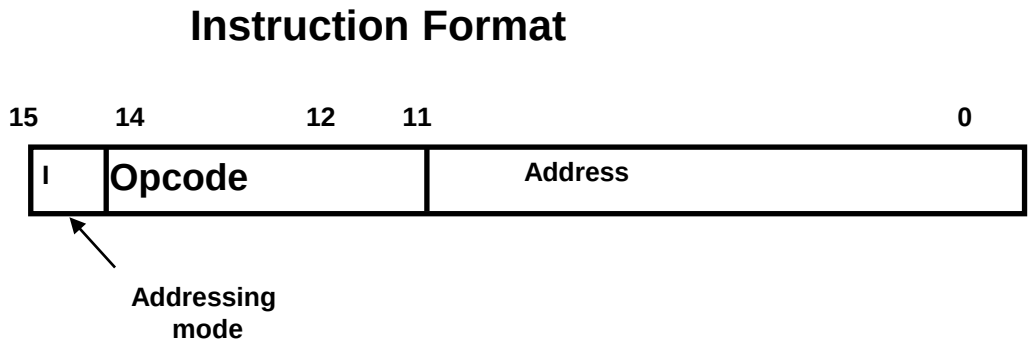

**Figure 2.1: Stored Program Organization**

# INSTRUCTION FORMAT

- A computer instruction is often divided into **two parts**

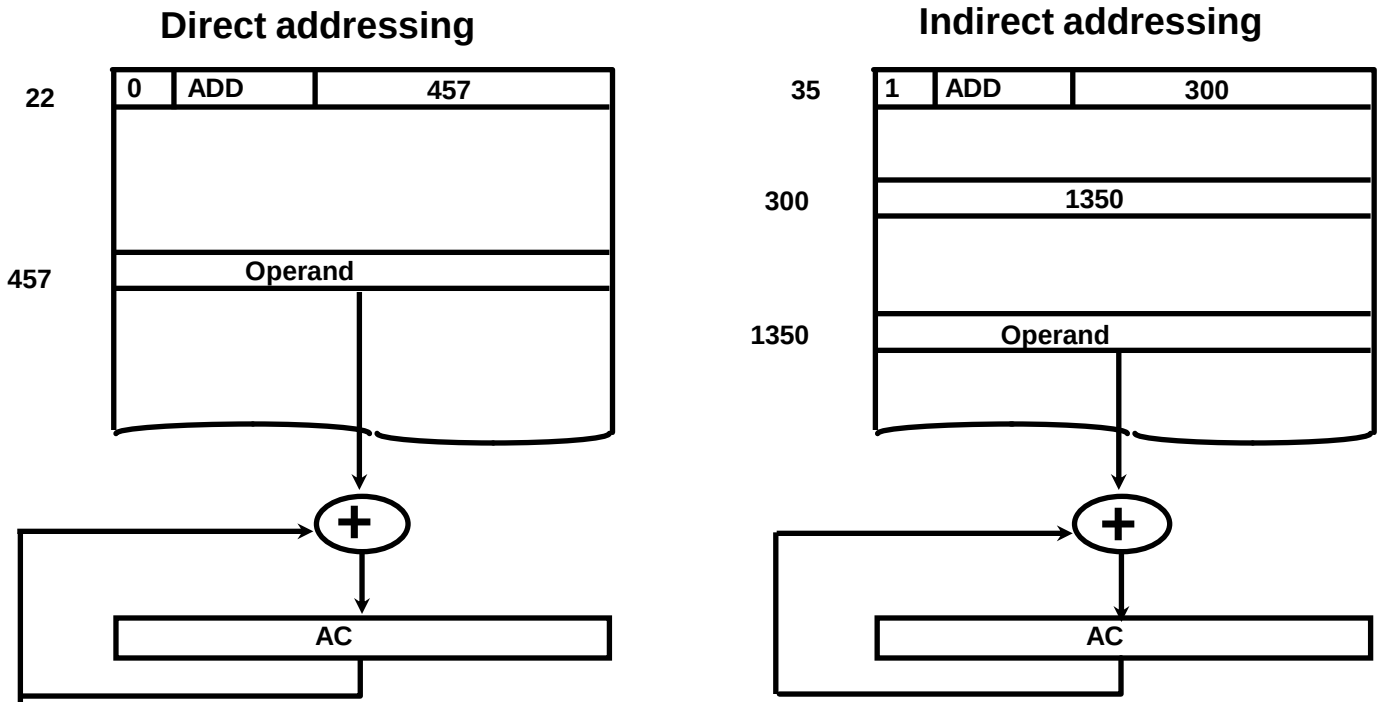
  1. An ***opcode*** (Operation Code) that specifies the operation for that instruction
  2. An ***address*** that specifies the registers and/or locations in memory to use for that operation

- In the Basic Computer, since the memory contains 4096 (= $2^{12}$) words, we needs 12 bit to specify which memory address this instruction will use.
- 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



# ADDRESSING MODES :

- The address field of an instruction can represent either
  - **Direct address**: the address in memory of the data to use (the address of the operand), or
  - **Indirect address**: the address in memory of the address in memory of the data to use




- **Effective Address (EA)**
  - The address, that can be directly used without modification to access an operand for a computation-type instruction, or as the **target address** for a branch-type instruction

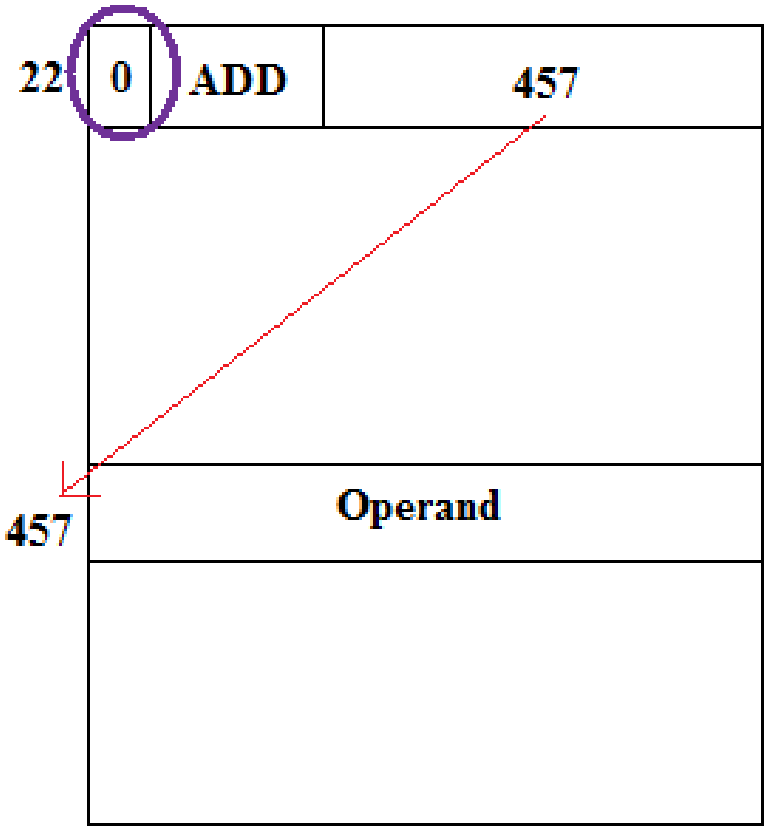
# Direct address :

Occurs When the Operand Part Contains the Address of Needed Data.

1. Address part of IR is placed on the bus and loaded back into the AR

2. Address is selected in memory and its Data placed on the bus to be loaded into the Data Register to be used for requested instructions

# Direct address



# Indirect address :

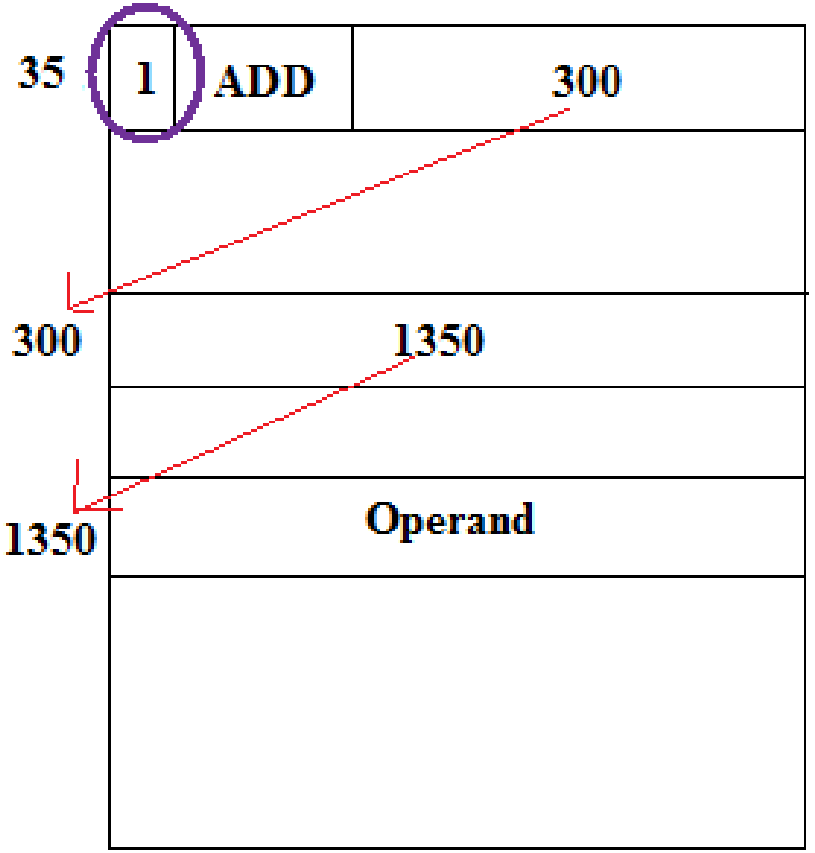
Occurs When the Operand Contains the Address of  the Address of Needed Data.

1. Address part of IR is placed on the bus and loaded back into the AR

2. Address is selected in memory and placed on the bus to be loaded Back into the AR

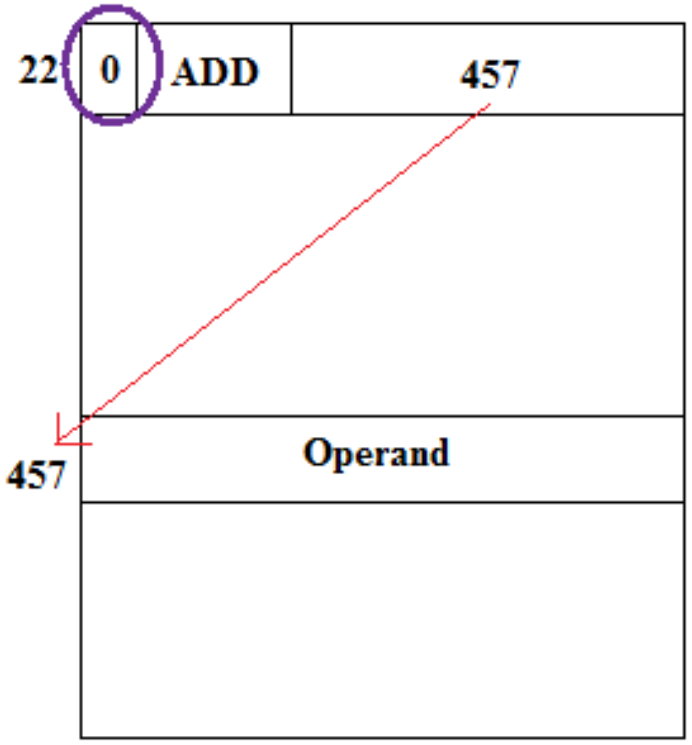
3. New Address is selected in memory and placed on the bus to be loaded into the DR to use later

# Indirect address

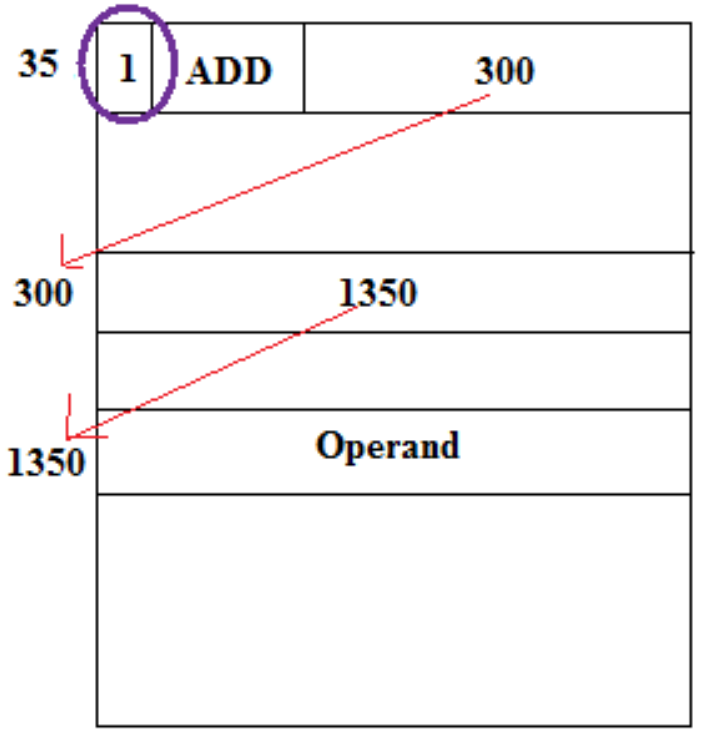


# Effective address:

- **Effective address**: Address where an operand is physically located

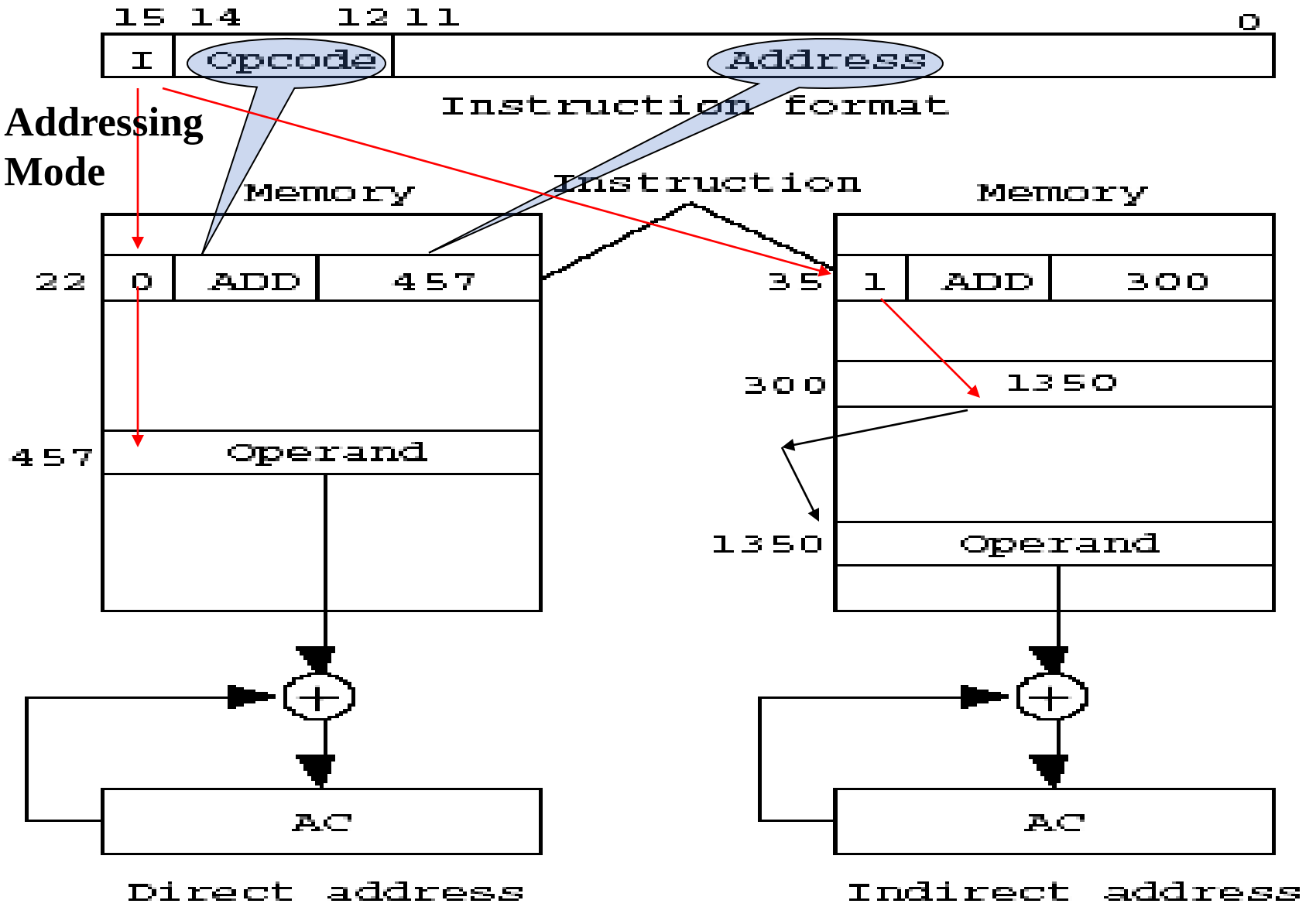



**Effective address: 457**




**Effective address: 1350**

# Direct and Indirect addressing example



# Registers of basic computer :

- It is 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

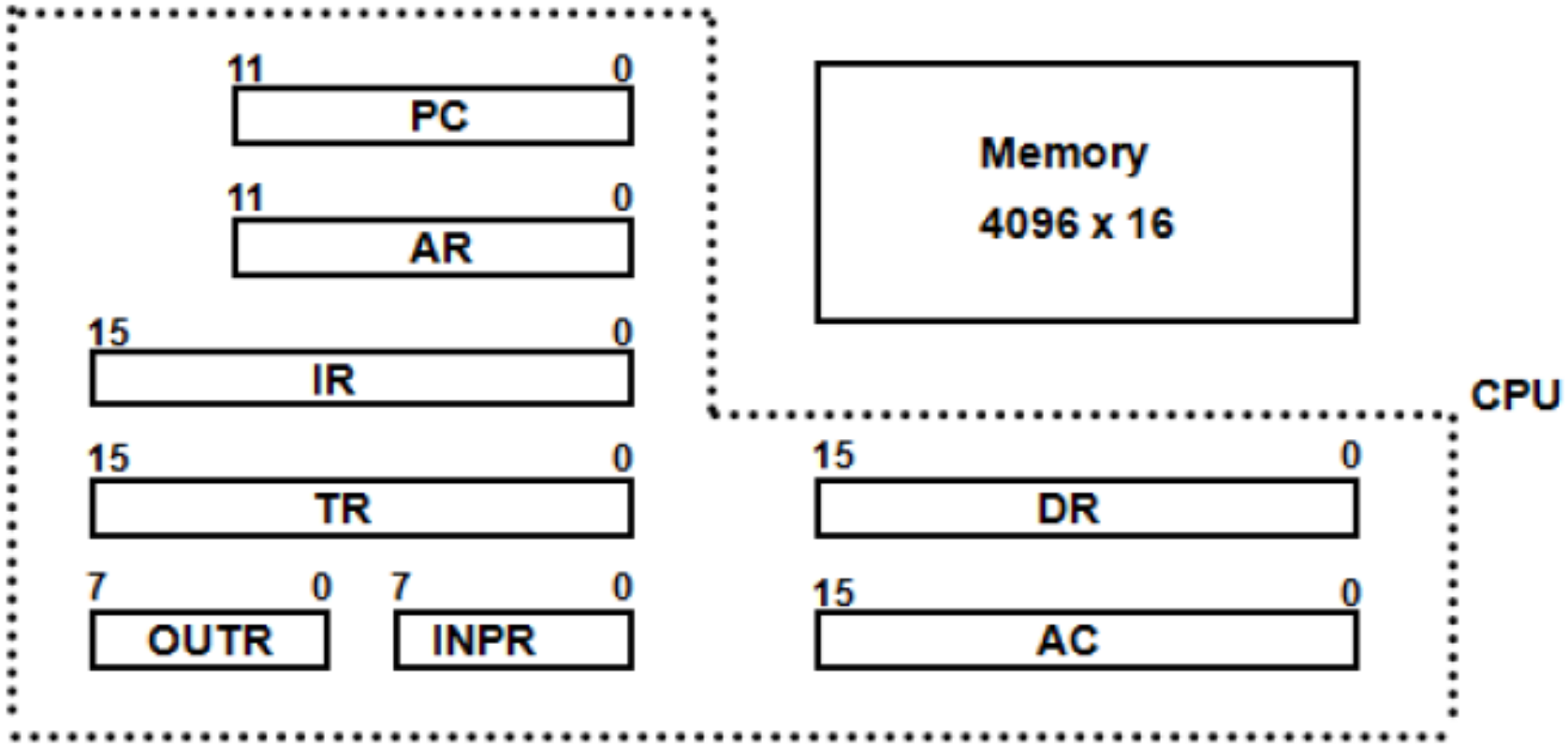



**Figure : Basic Computer Register and Memory**

- 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.
- The memory address register (AR) has 12 bits.
- 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.

- 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.

- Two registers are used for input and output. The input register (INPR) receives an 8-bit character from an input device. The output register (OUTR) holds an 8-bit character for an output device.

| Register Symbol | 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 |

**Table 2.1: List of Registers for Basic Computer**

# Common Bus System for basic computer register :

- 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 register.

- The number of wires will be excessive if connections are between the outputs of each register and the inputs of the other registers. An efficient scheme for transferring information in a system with many register 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.

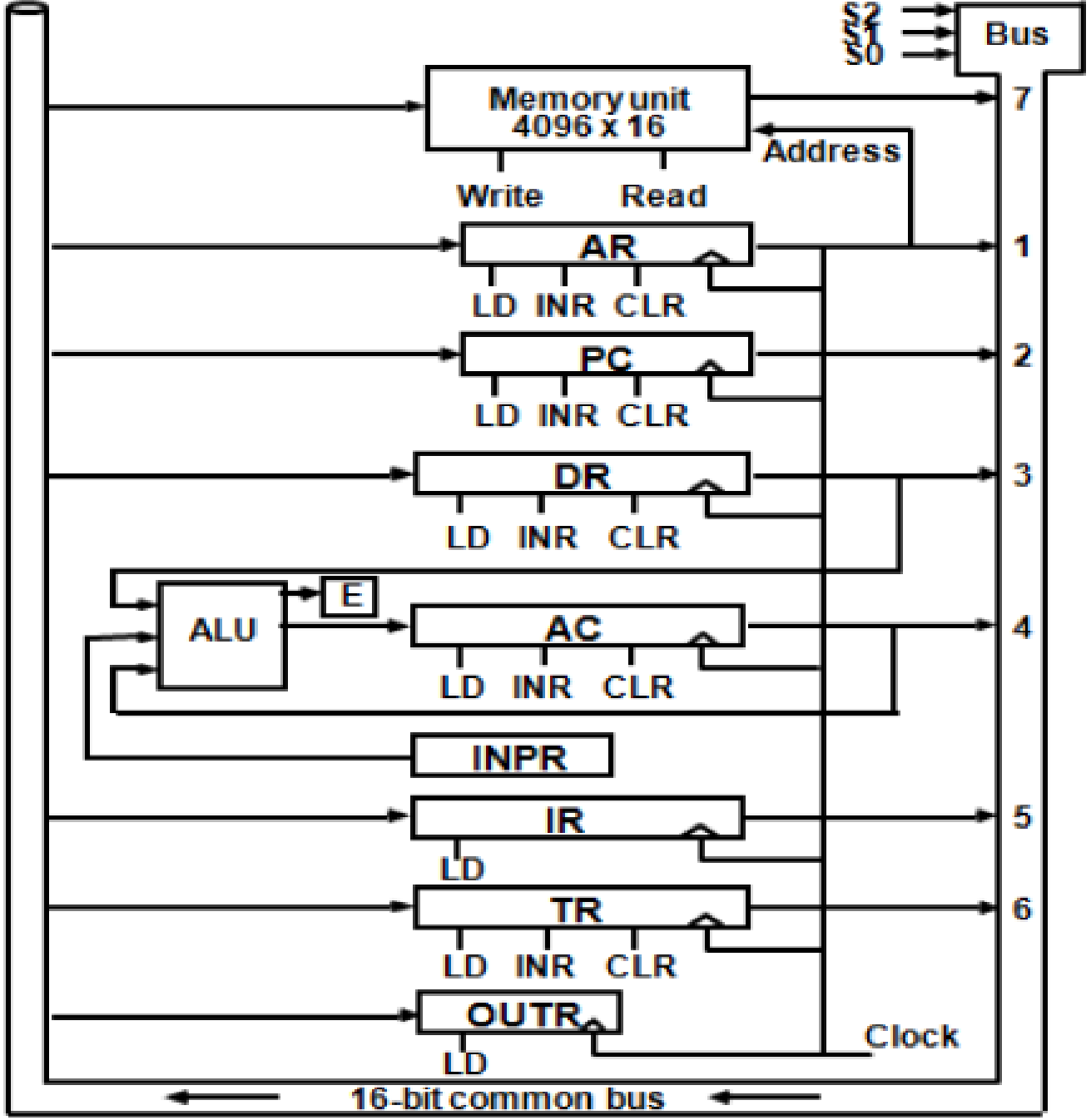



Figure 2.5: Basic computer registers connected to a common bus

One Address Instruction

$X = (A+B) * (C+D)$

AC=A+B

Load A $AC \leftarrow M[A]$

Add B $AC \leftarrow AC + M[B]$

Store T

Load C $AC \leftarrow M[C]$

Add D $AC \leftarrow AC + M[D]$

MUL T

Store X

- 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.

- 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 S2 S1 S0 = 1 1 1.
- Four registers, DR, AC, IR, and TR have 16 bits each.
- Two registers, AR 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 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 a 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 sets of inputs.
    1. Set of 16-bit inputs come from the outputs of AC.
    2. Set of 16-bits come from the data register DR.
    3. 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.

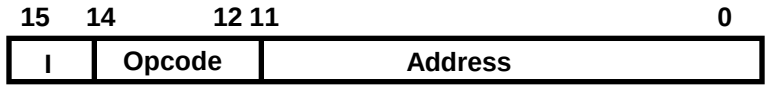
- Three control lines, $S_2$, $S_1$, and $S_0$ control which register the bus selects as its input

| $S_2$ | $S_1$ | $S_0$ | Register |
|---|---|---|---|
| 0 | 0 | 0 | x |
| 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 |

# BASIC COMPUTER INSTRUCTIONS :

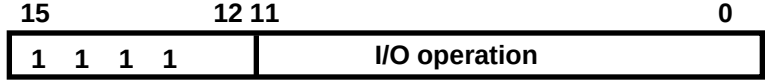
- **Basic Computer Instruction Format**

Memory-Reference Instructions (OP-code = 000 ~ 110)




Register-Reference Instructions (OP-code = 111, I = 0)


Input-Output Instructions (OP-code =111, I = 1)



# BASIC COMPUTER INSTRUCTIONS

| 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 |
| 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 |
| 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 |

# Basic working principle of the Control Unit with timing diagram :

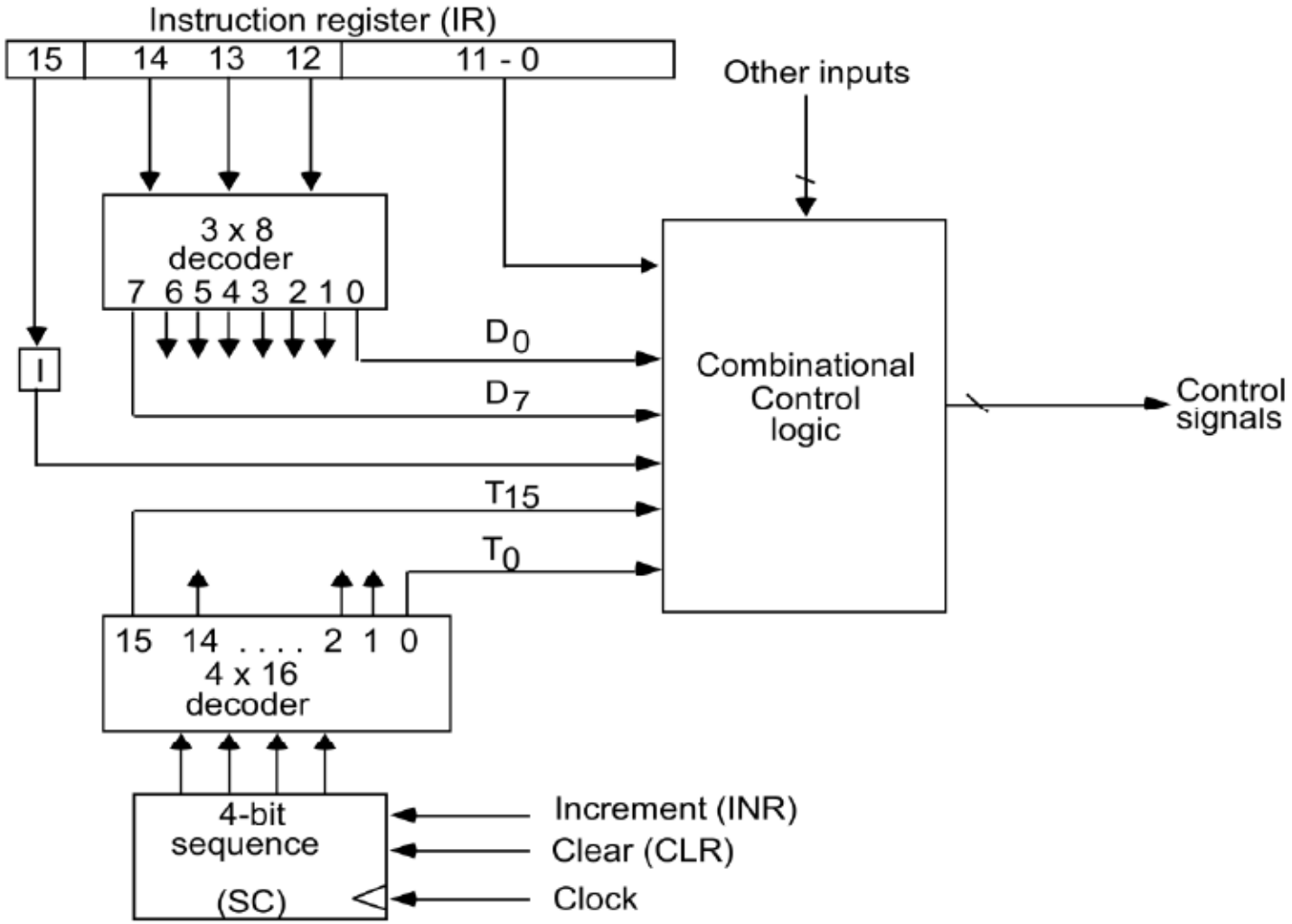



**Figure 2.7: Control unit of basic computer**

- Components of Control unit are
    1. Two decoders
    2. A sequence counter
    3. Control logic gates
- An instruction read from memory is placed in the instruction register (IR). In control unit the IR is divided into three parts: I bit, the operation code (12-14)bit, and bits 0 through 11.
- The operation code in bits 12 through 14 are decoded with a 3 X 8 decoder.
- Bit-15 of the instruction is transferred to a flip-flop designated by the symbol I.
- The eight outputs of the decoder are designated by the symbols D0 through D7. 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 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 4 X 16 decoder. Once in awhile, the counter is cleared to 0, causing the next timing signal to be T0.
- As an example, consider the case where SC is incremented to provide timing signals T0, T1, T2, T3 and T4 in sequence. At time T4, SC is cleared to 0 if decoder output D3 is active. This is expressed symbolically by the statement

$$D3T4: SC \leftarrow 0$$

## Timing Diagram:

- The timing diagram figure 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 clears SC to 0, which in turn activates the timing T0 out of the decoder. T0 is active during one clock cycle.
- The positive clock transition labeled T0 in the diagram will trigger only those registers whose control inputs are connected to timing signal T0.
- SC is incremented with every positive clock transition, unless its CLR input is active.

- This procedures the sequence of timing signals T0, T1, T2, T3 and T4, and so on. If SC is not cleared, the timing signals will continue with T5, T6, up to T15 and back to T0.
- The last three waveforms shows how SC is cleared when D3T4 = 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 D3T4 becomes active.

- This signal is applied to the CLR input of SC. On the next positive clock transition the counter is cleared to 0. This causes the timing signal T0 to become active instead of T5 that would have been active if SC were incremented instead of cleared.

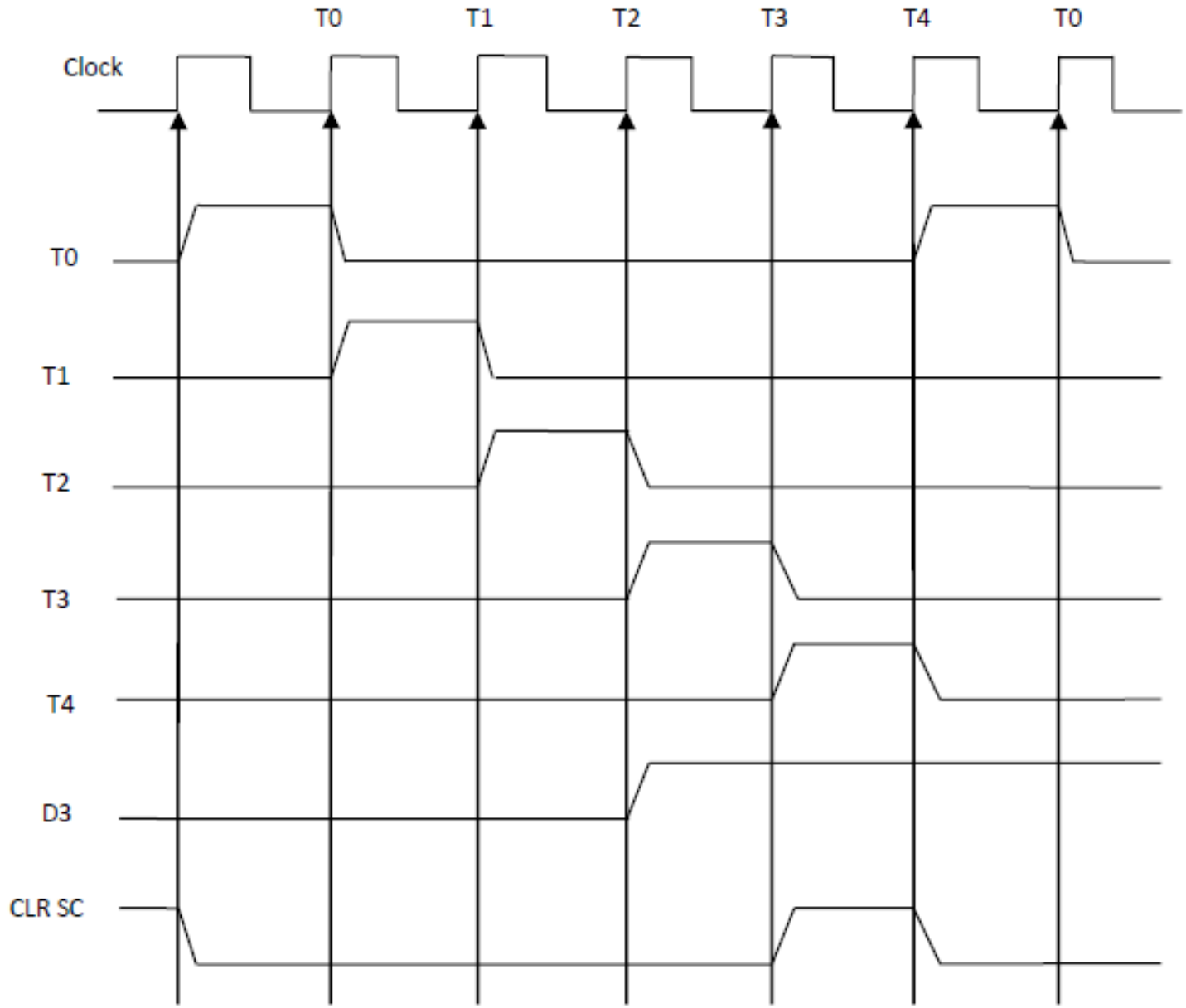



**Figure 2.8: Example of control timing signals**

## Instruction Execution Cycle :

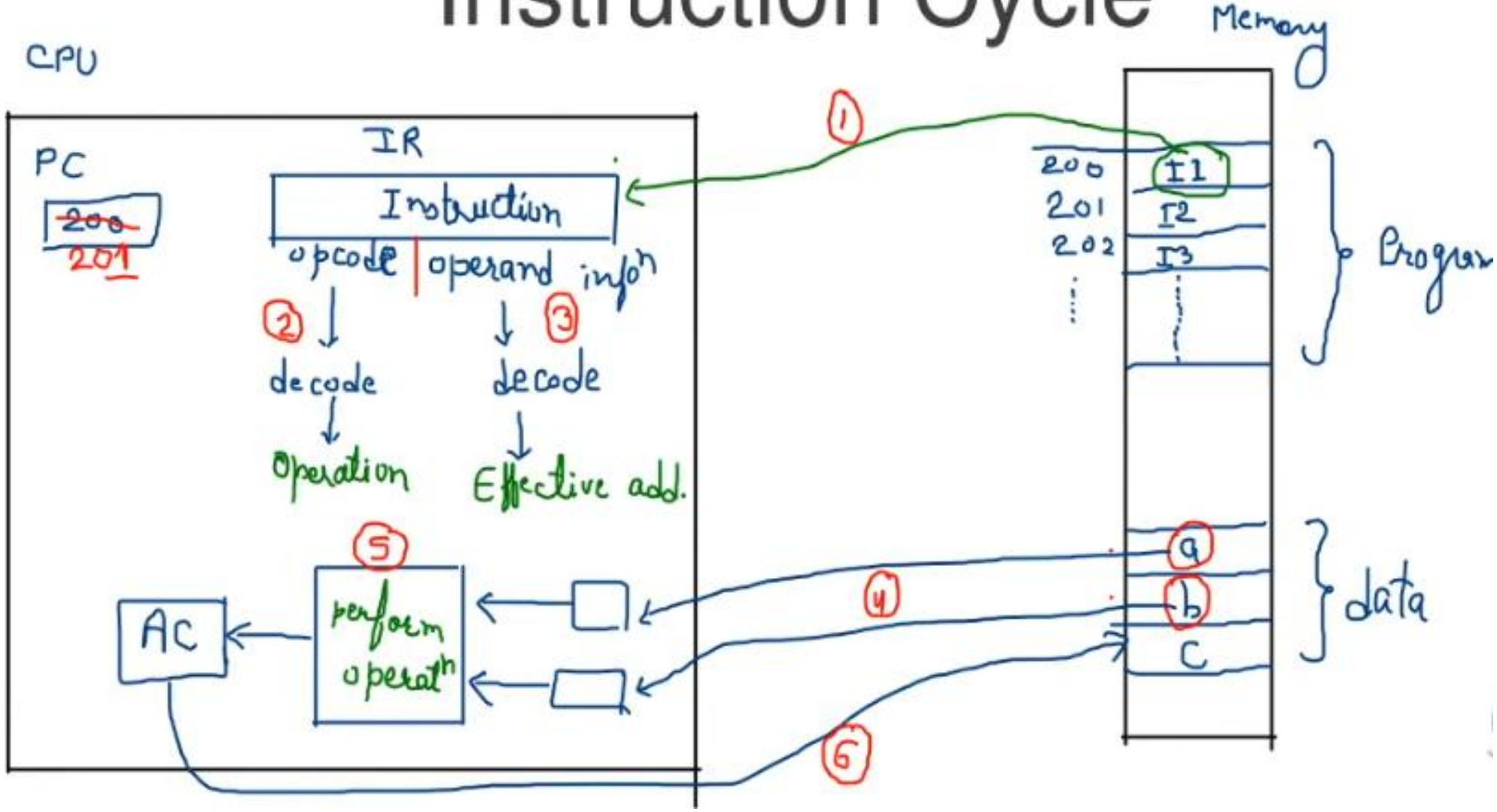
- A program residing in the memory unit of the computer consists of a sequence of instructions. In the basic computer each instruction cycle consists of the following phases:

  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

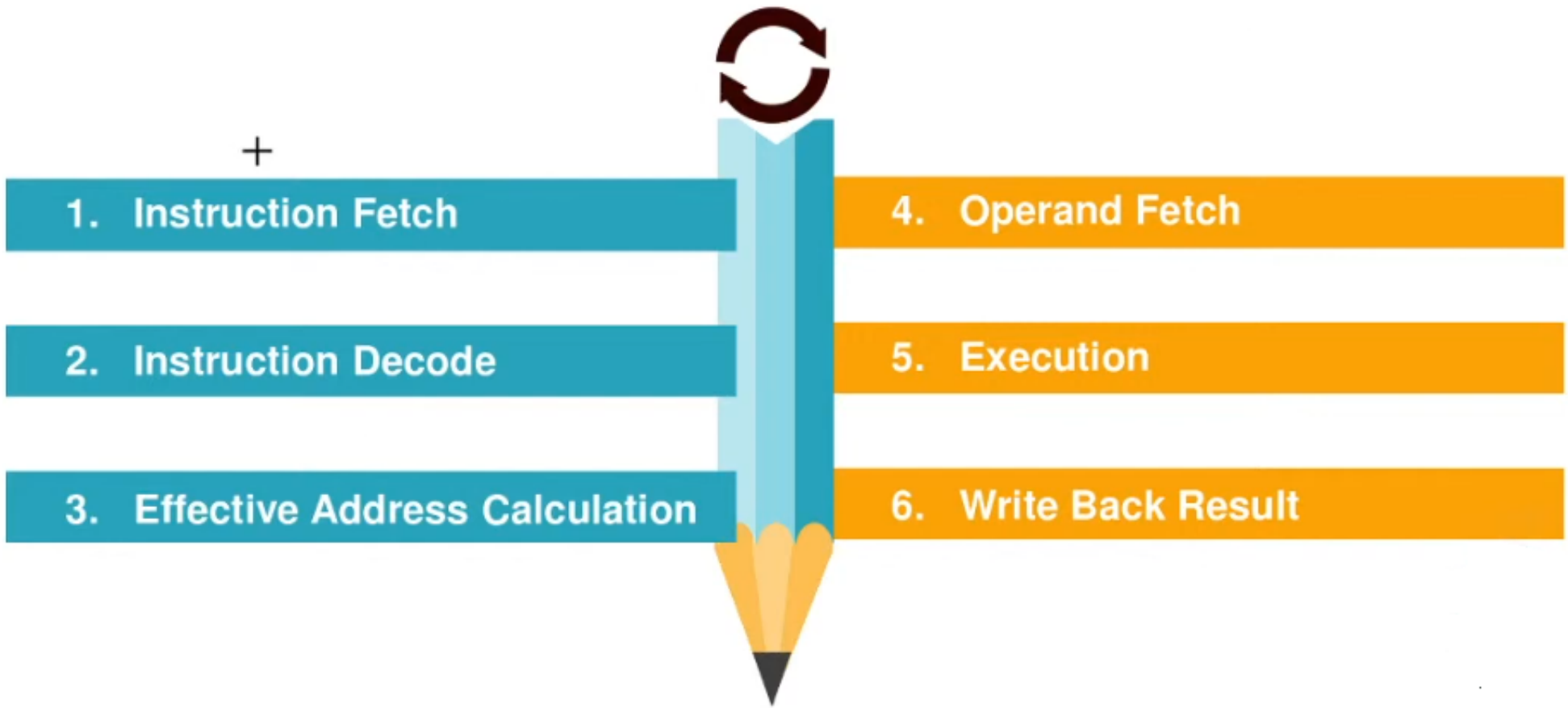
- After an instruction is executed, the cycle starts again at step 1, for the next instruction
- This process continues unless a HALT instruction is encountered.

- *Note*: Every different processor has its own (different) instruction cycle

# Instruction Cycle



# Instruction Cycle



- **Fetch and Decode**

- Initially, the program counter PC is loaded with the address of first instruction in the program.
- The sequence counter SC is cleared to 0, providing a decoded timing signal T0.
- After each clock pulse, SC is incremented by 1.
- The following microoperation for fetch & decode phases can be specified by the following resister transfer statements.

T0: AR ← PC ($S_0S_1S_2$=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)

- Only AR is connected to address inputs of memory, AR ← PC with timing signal T0.
- The instruction read from memory, IR ← M [AR] with timing signal T1 & at the same time PC ← PC + 1
- At the time T2, the operation code in IR is decoded, the indirect bit I is transferred to flip-flop & address part of instruction is transferred to AR.
- SC is incremented after each clock cycle pulse to produce the sequence T0, T1, T2.

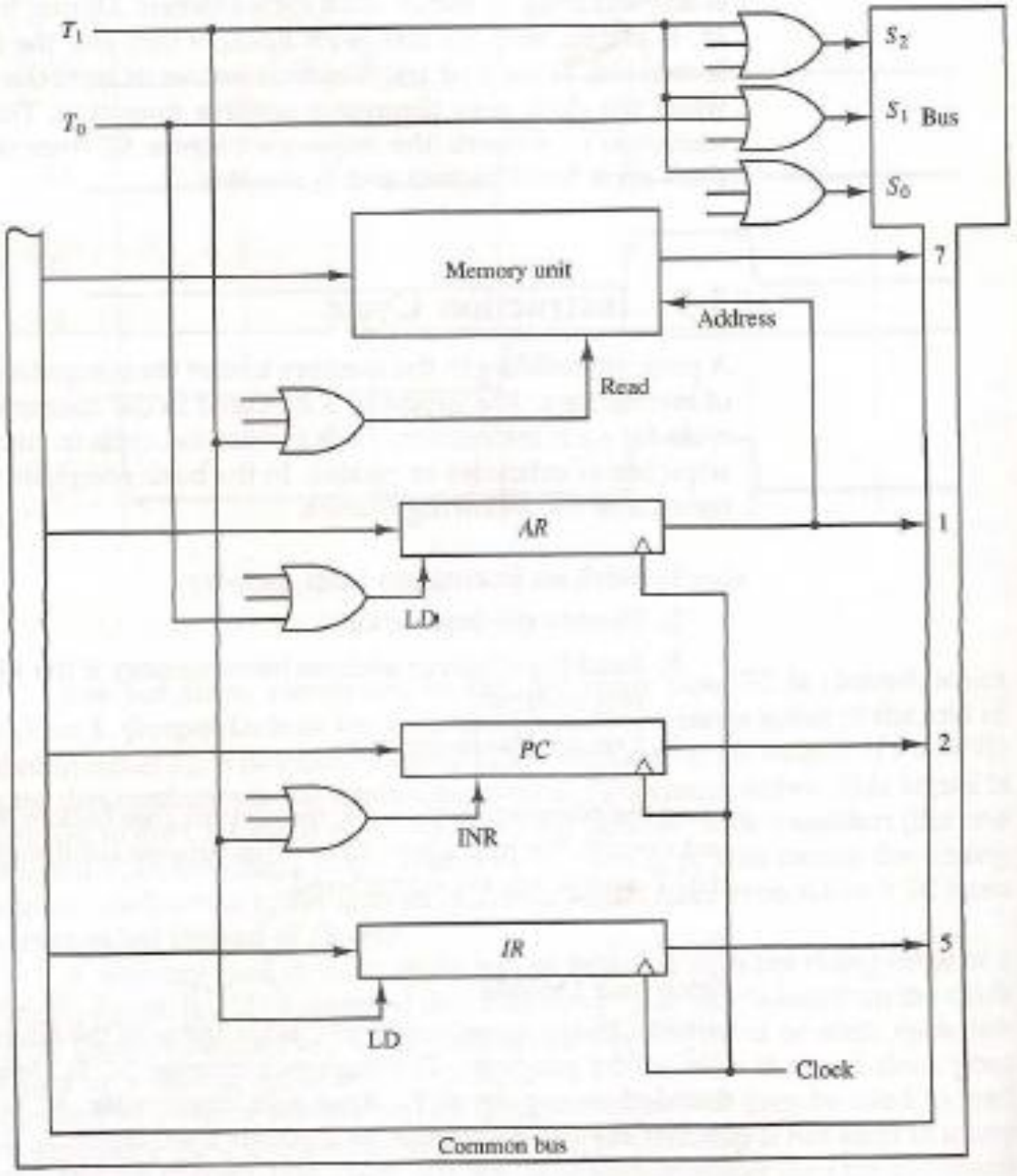
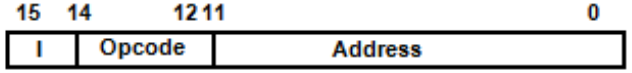
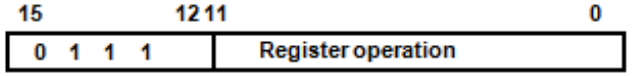



Figure 5-8 Register transfers for the fetch phase.

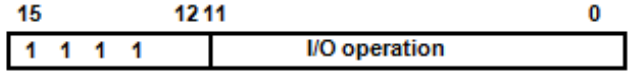
## Memory-Reference Instructions (OP-code = 000 ~ 110)



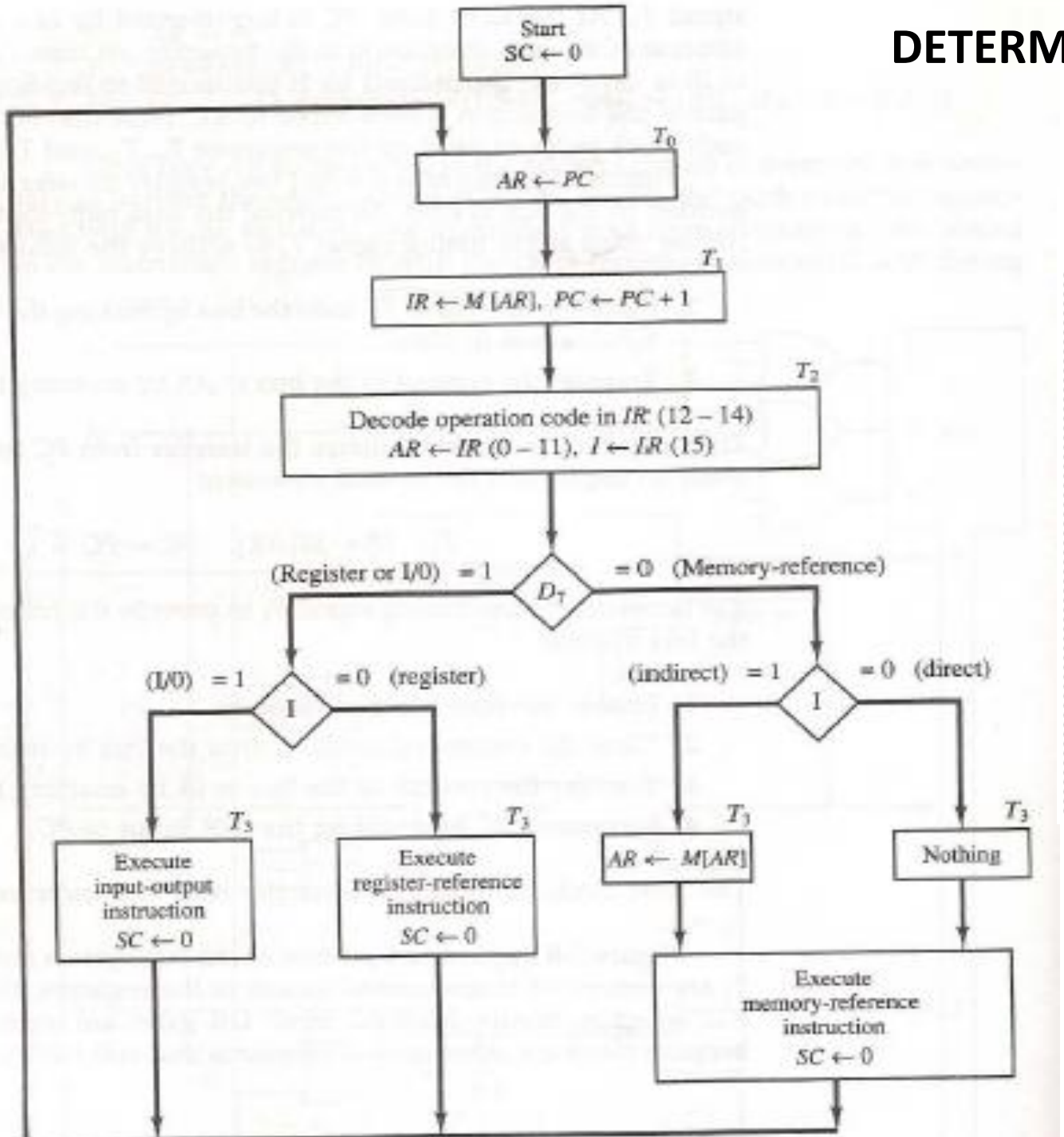

## Register-Reference Instructions (OP-code = 111, I = 0)

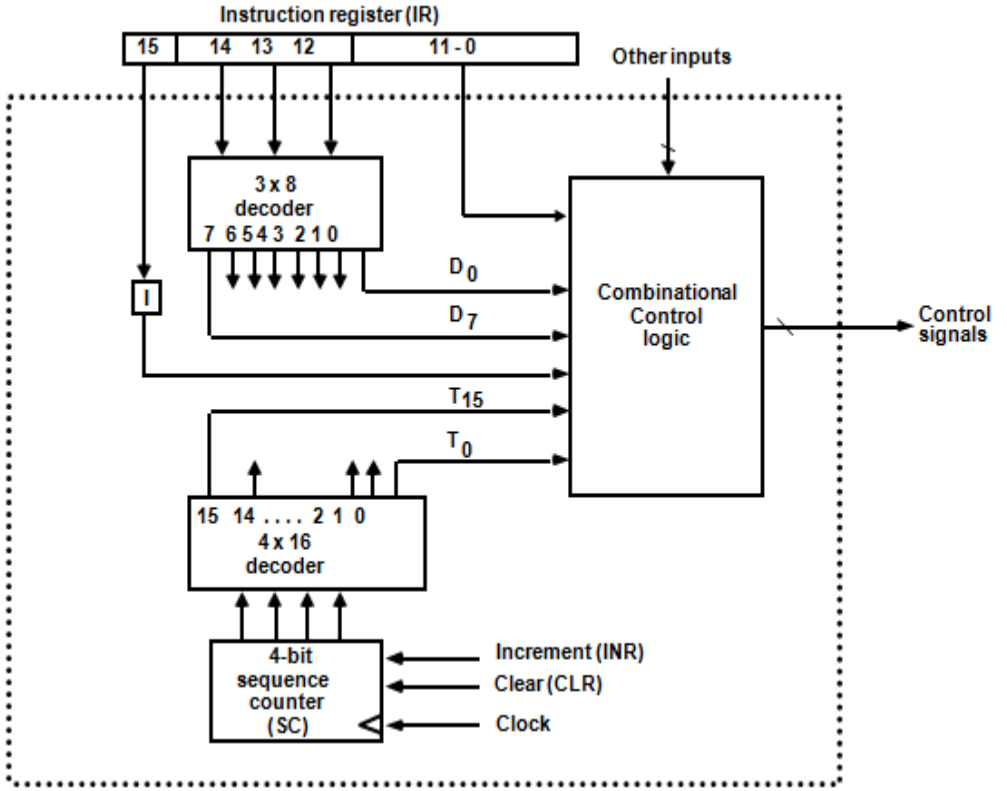



## Input-Output Instructions (OP-code =111, I = 1)



# DETERMINE THE TYPE OF INSTRUCTION



- 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= 1**, the instruction must be **register-reference** or **input-output type.**
- If **D7 = 0**, the operation code must be one of the other seven values 110, specifying a memory reference instruction. (000 – 110)
- Control then inspects the value of the first bit of the instruction, which 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 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 | I | |
|---|---|---|
| 0 | 1 | **D'7 I T3 : AR ← M[AR]** |
| 0 | 0 | **D'7 I 'T3 : Nothing** |
| 1 | 0 | **D7 I 'T3 : Execute a register-reference instr.** |
| 1 | 1 | **D7 I T3 : Execute an input-output instr.** |

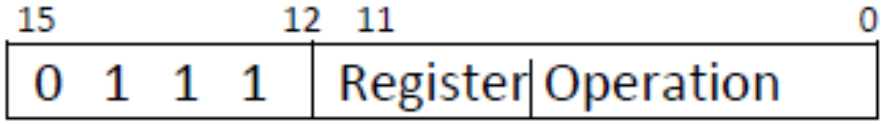
- When a memory-reference instruction with **I = 0** is encountered, it is not necessary to do anything since the **effective address is already in AR**.

- However, the sequence counter SC must be incremented when D’7 I T3 = 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. SC is either incremented or cleared to 0 with every positive clock transition.

# REGISTER REFERENCE INSTRUCTIONS :

- When the register-reference instruction is decoded, D7 bit is set to 1.
- Each control function needs the Boolean relation D7 I' T3




- There are 12 register-reference instructions listed below:

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

## REGISTER REFERENCE INSTRUCTIONS :

Register Reference Instructions are identified when

- $D_7 = 1$, $I = 0$
- Register Ref. Instr. is specified in $b_0 \sim b_{11}$ of IR
- Execution starts with timing signal $T_3$

$r = D_7 I' T_3$ => Register Reference Instruction

$B_i = IR(i)$ , i=0,1,2,...,11

- These 12 bits are available in IR (0-11). They were also transferred to AR during time T2.
- These instructions are executed at timing cycle T3.

- 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 condition is satisfied. The skipping of the instruction is achieved by incrementing PC.

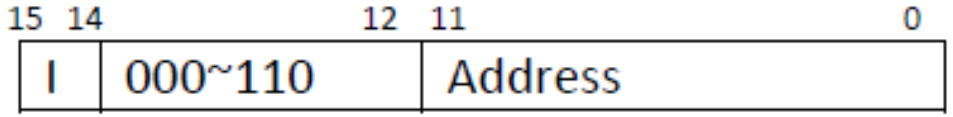
- The condition control statements must be recognized as part of the control conditions. The AC is positive when the sign bit in AC(15) = 0; it is negative when AC(15) = 1.
- The content of AC is zero (AC = 0) if all the flip-flops of the register are zero.

- 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 :

- When the memory-reference instruction is decoded, D7 bit is set to 0.




- The following table lists seven memory-reference instructions.

| Symbol | Operation Decoder | Symbolic Description |
|---|---|---|
| AND | $D_0$ | AC ← AC ∧ M[AR] |
| ADD | $D_1$ | AC ← AC + M[AR], E ← $C_{out}$ |
| LDA | $D_2$ | AC ← M[AR] |
| STA | $D_3$ | M[AR] ← AC |
| BUN | $D_4$ | PC ← AR |
| BSA | $D_5$ | M[AR] ← PC, PC ← AR + 1 |
| ISZ | $D_6$ | M[AR] ← M[AR] + 1, if M[AR] + 1 = 0 then PC ← PC+1 |

- 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.

## AND to AC

- This is an instruction that performs 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.

$$D0T4: DR \leftarrow M[AR]$$
$$D0T5: AC \leftarrow AC \wedge DR, SC \leftarrow 0$$

## 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 Cout is transferred to the E (extended accumulator) flip-flop.

$$D1T4: DR \leftarrow M[AR]$$
$$D1T5: AC \leftarrow AC + DR, E \leftarrow Cout, SC \leftarrow 0$$

## LDA: Load to AC

- This instruction transfers the memory word specified by the effective address to AC.

D2T4: DR ← M[AR]

D2T5: AC ← DR, SC ← 0

## STA: Store AC

- This instruction stores the content of AC into the memory word specified by the effective address.

D3T4: M[AR] ← AC, SC ← 0

## BUN: Branch Unconditionally

- This instruction transfers the program to instruction specified by the effective address.
- The BUN instruction allows the programmer to specify an instruction out of sequence and the program branches (or jumps) unconditionally.

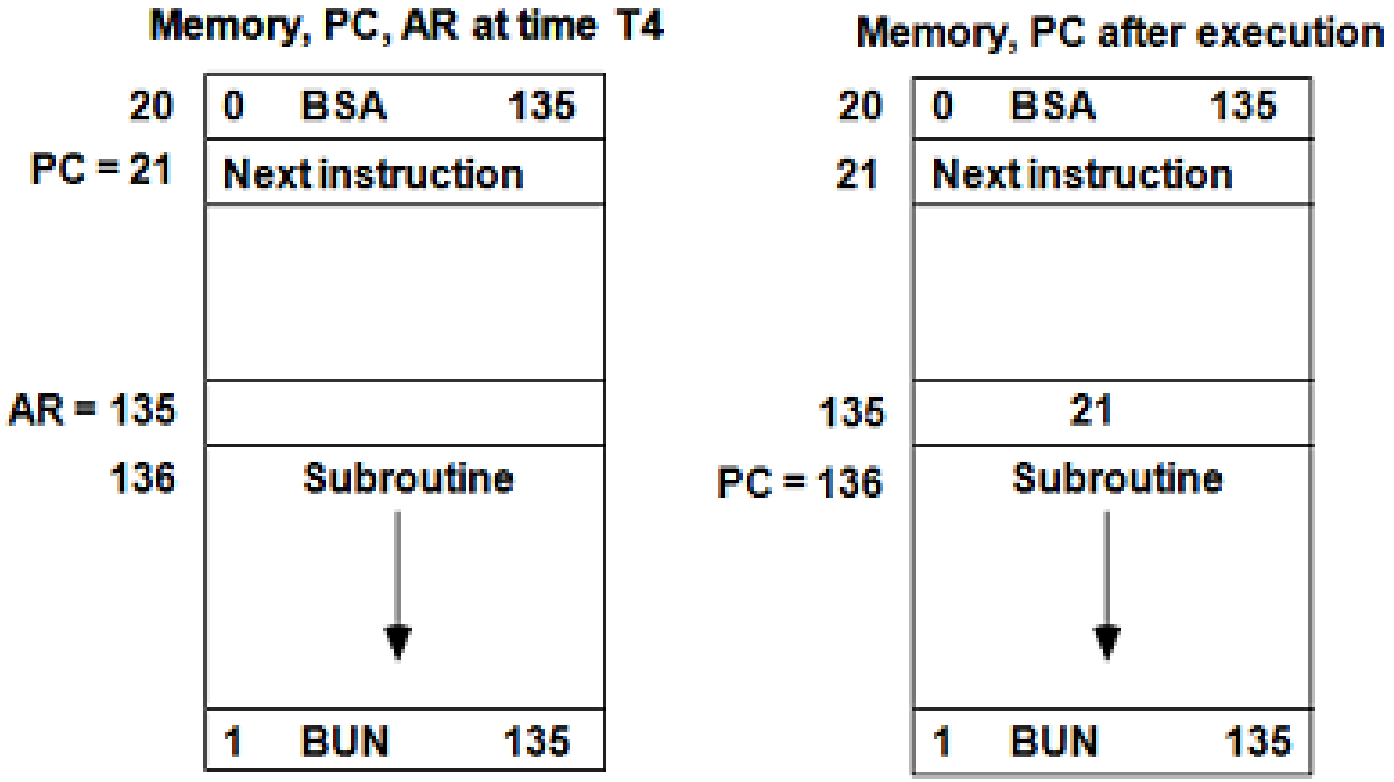
D4T4: PC ← AR, SC ← 0

## 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.

$M[AR] \leftarrow PC, PC \leftarrow AR + 1$

$M[135] \leftarrow 21, PC \leftarrow 135 + 1 = 136$



- It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the basic computer.
- To use the memory and the bus properly, the BSA instruction must be executed with a sequence of two microoperations:

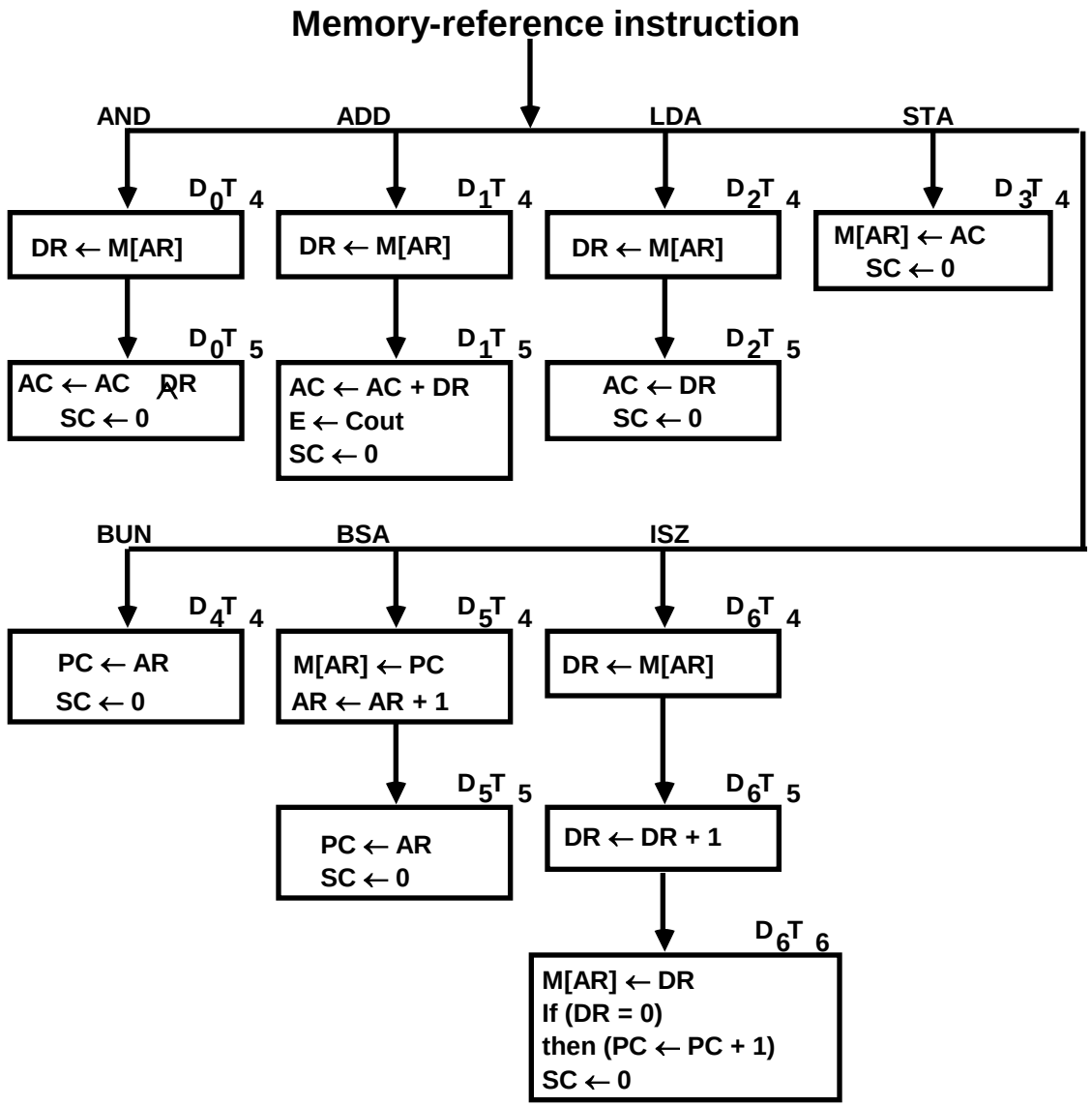
D5T4: M[AR] ← PC, AR ← AR + 1
D5T5: PC ← AR, SC ← 0

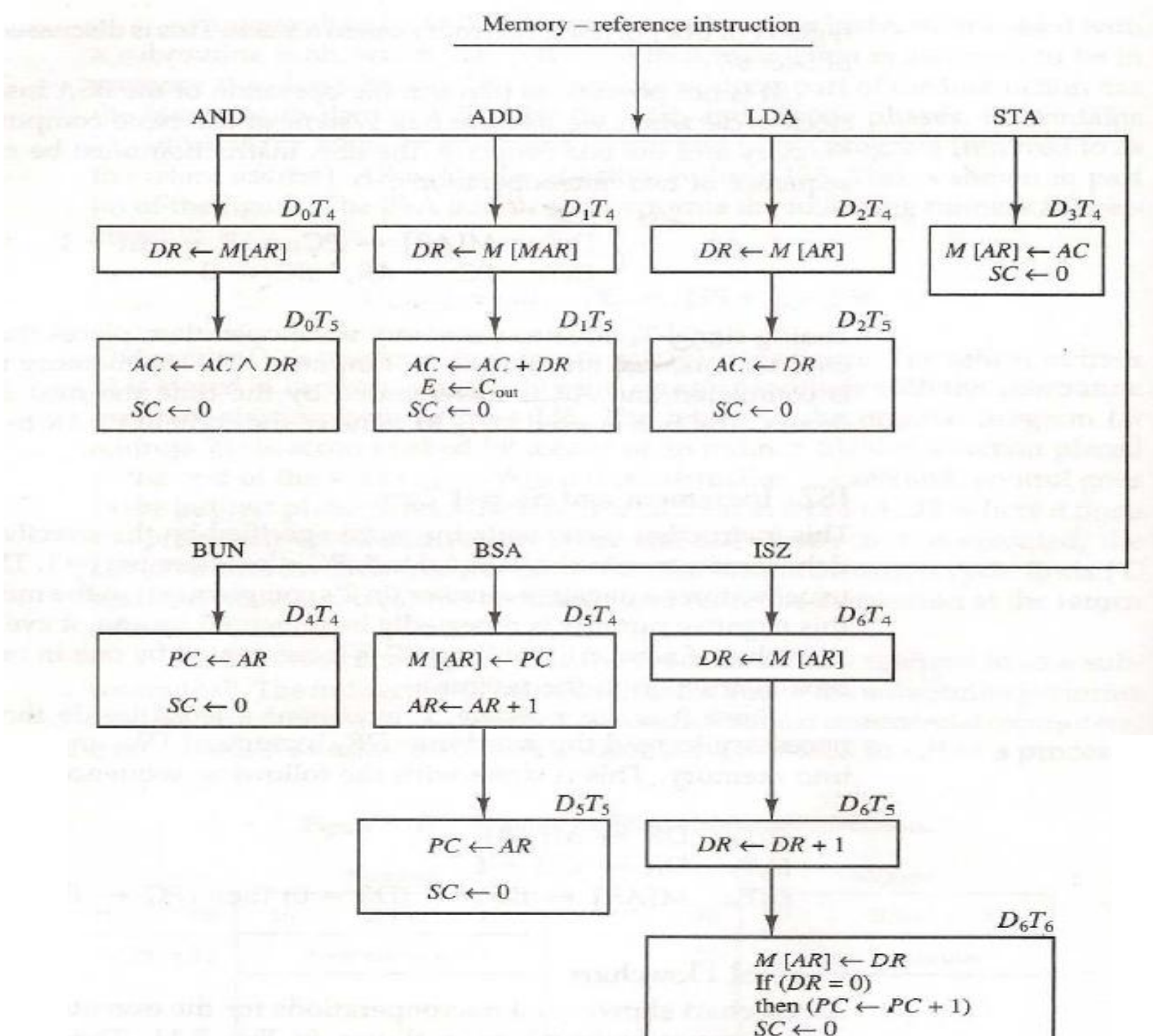
## ISZ: Increment and Skip if Zero

- These instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1.
- 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.

D6T4: DR ← M[AR]
D6T5: DR ← DR + 1
D6T4: M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0

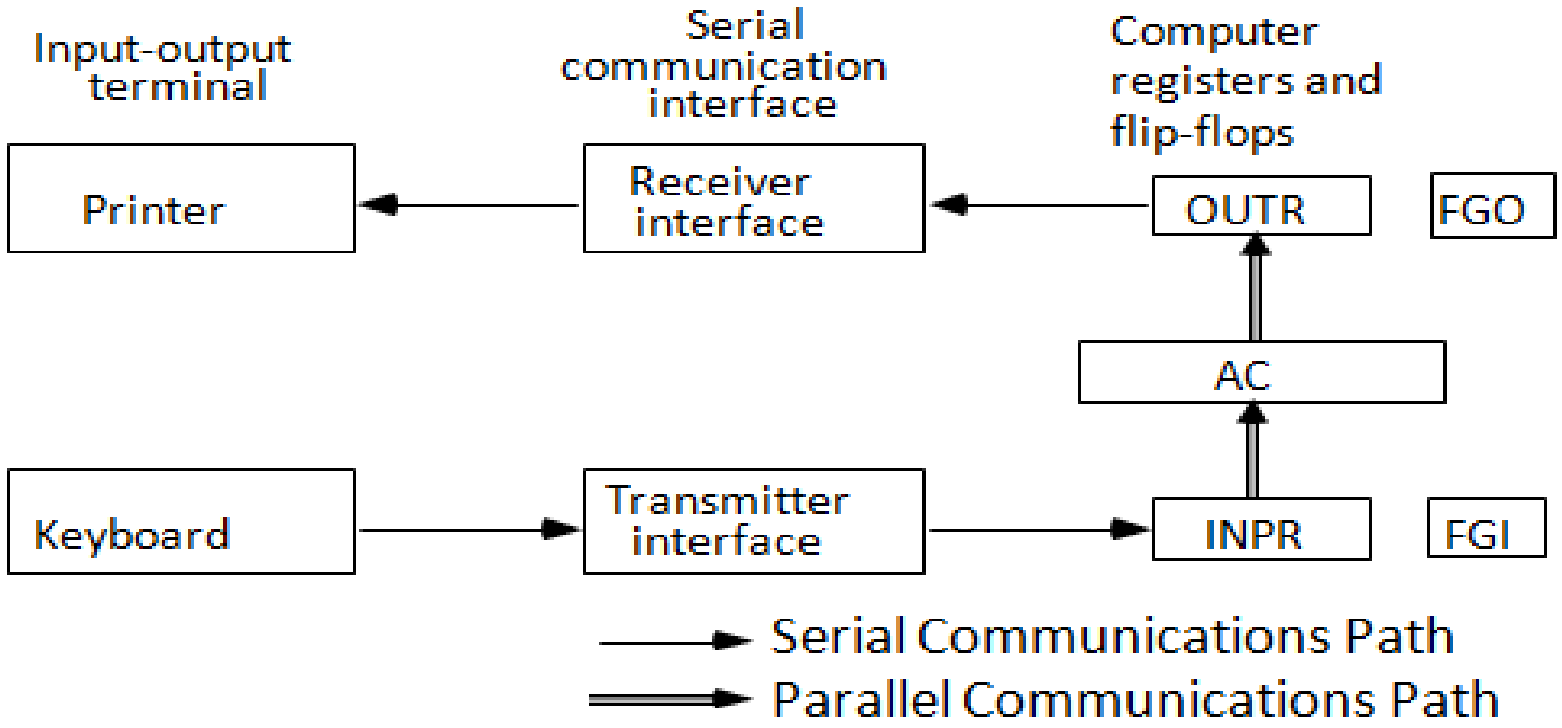
# FLOWCHART FOR MEMORY REFERENCE INSTRUCTIONS :




Memory – reference instruction
AND
ADD
LDA
STA
$D_0T_4$
$DR \leftarrow M[AR]$
$D_1T_4$
$DR \leftarrow M[MAR]$
$D_2T_4$
$DR \leftarrow M[AR]$
$D_3T_4$
$M[AR] \leftarrow AC$
$SC \leftarrow 0$
$D_0T_5$
$AC \leftarrow AC \wedge DR$
$SC \leftarrow 0$
$D_1T_5$
$AC \leftarrow AC + DR$
$E \leftarrow C_{out}$
$SC \leftarrow 0$
$D_2T_5$
$AC \leftarrow DR$
$SC \leftarrow 0$
BUN
BSA
ISZ
$D_4T_4$
$PC \leftarrow AR$
$SC \leftarrow 0$
$D_5T_4$
$M[AR] \leftarrow PC$
$AR \leftarrow AR + 1$
$D_6T_4$
$DR \leftarrow M[AR]$
$D_5T_5$
$PC \leftarrow AR$
$SC \leftarrow 0$
$D_6T_5$
$DR \leftarrow DR + 1$
$D_6T_6$
$M[AR] \leftarrow DR$
If $(DR = 0)$
then $(PC \leftarrow PC + 1)$
$SC \leftarrow 0$

# Input-Output Configuration of basic computer :

- A computer can serve no useful purpose unless it communicates with the external environment.
- To exhibit the most basic requirements for input and output communication, we will use a terminal unit with a keyboard and printer.




***INPR*** Input register - 8 bits
***OUTR*** Output register - 8 bits
***FGI*** Input flag - 1 bit
***FGO*** Output flag - 1 bit
***IEN*** Interrupt enable - 1 bit

**Input-output configuration**

- The terminal sends and receives serial information
- The serial info. from the keyboard is shifted into INPR
- The serial info. for the printer is stored in the OUTR
- INPR and OUTR communicate with the terminal serially and with the AC in parallel.
- The flags are needed to *synchronize* the timing difference between I/O device and the computer.

• The terminal sends and receives serial information and 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 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 1-bit input flag FGI is a control flip-flop. It 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:

## *The process of input information transfer:*

- 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 process of outputting information:*

- 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 microoperations for the input-output instructions are listed below.

| | | |
|---|---|---|
| INP | AC(0-7) ← INPR, FGI ← 0 | Input char. to AC |
| OUT | OUTR ← AC(0-7), FGO ← 0 | Output char. from AC |
| SKI | if(FGI = 1) then (PC ← PC + 1) | Skip on input flag |
| SKO | if(FGO = 1) then (PC ← PC + 1) | Skip on output flag |
| ION | IEN ← 1 | Interrupt enable on |
| IOF | IEN ← 0 | Interrupt enable off |

**Table 2.2: Input Output Instructions**

# 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 microoperations for the input-output instructions are listed below.

$D_7IT_3 = p$
$IR(i) = B_i$, i = 6, …, 11

| | | | |
|---|---|---|---|
| | p: | SC ← 0 | Clear SC |
| INP | $pB_{11}$: | AC(0-7) ← INPR, FGI ← 0 | Input char. to AC |
| OUT | $pB_{10}$: | OUTR ← AC(0-7), FGO ← 0 | Output char. from AC |
| SKI | $pB_9$: | if(FGI = 1) then (PC ← PC + 1) | Skip on input flag |
| SKO | $pB_8$: | if(FGO = 1) then (PC ← PC + 1) | Skip on output flag |
| ION | $pB_7$: | IEN ← 1 | Interrupt enable on |
| IOF | $pB_6$: | IEN ← 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 2.2 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.

# Interrupt Cycle :

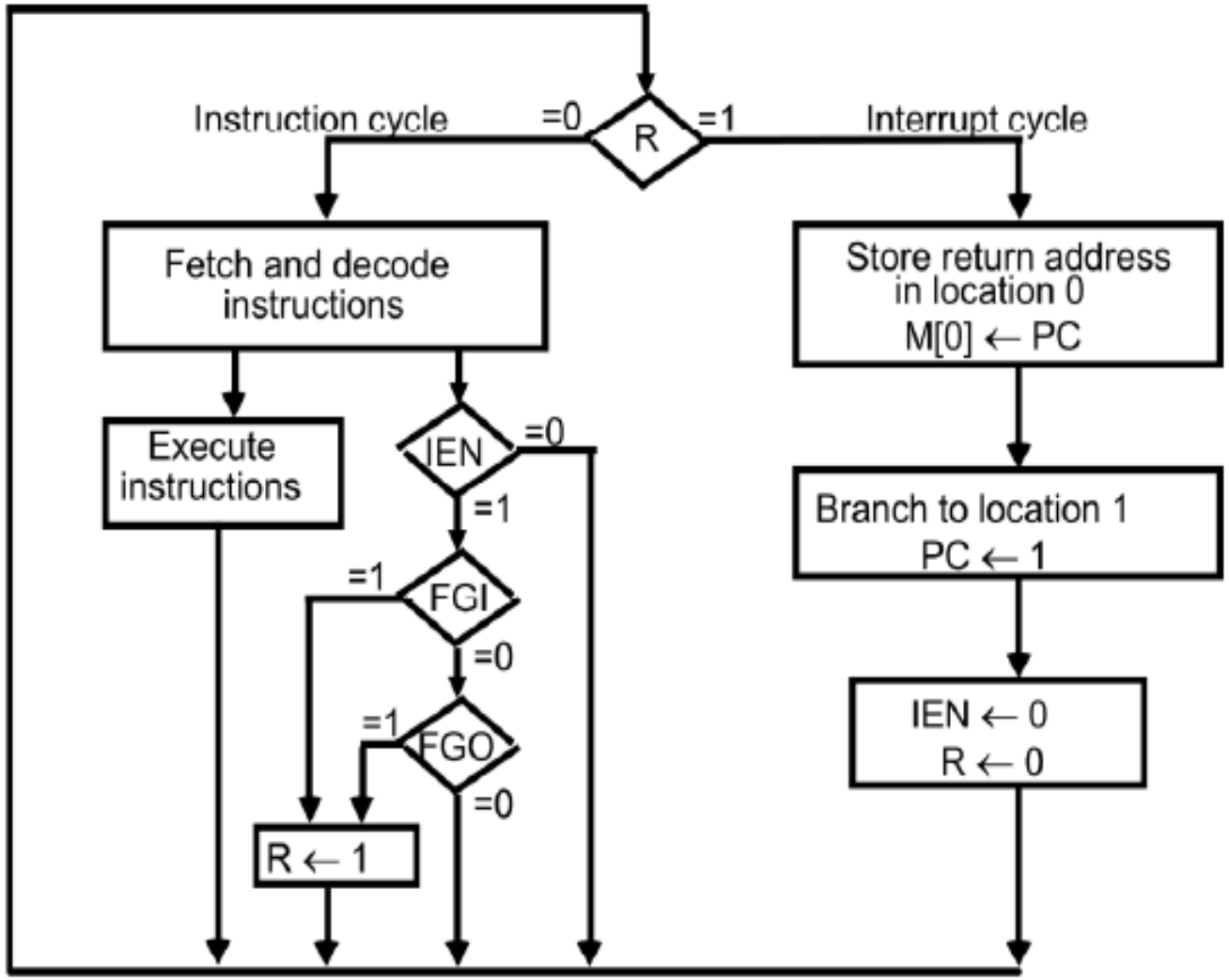



Figure 2.13: Flowchart for interrupt cycle

- 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 1, it goes to an interrupt cycle instead of an instruction cycle.

## *Interrupt 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

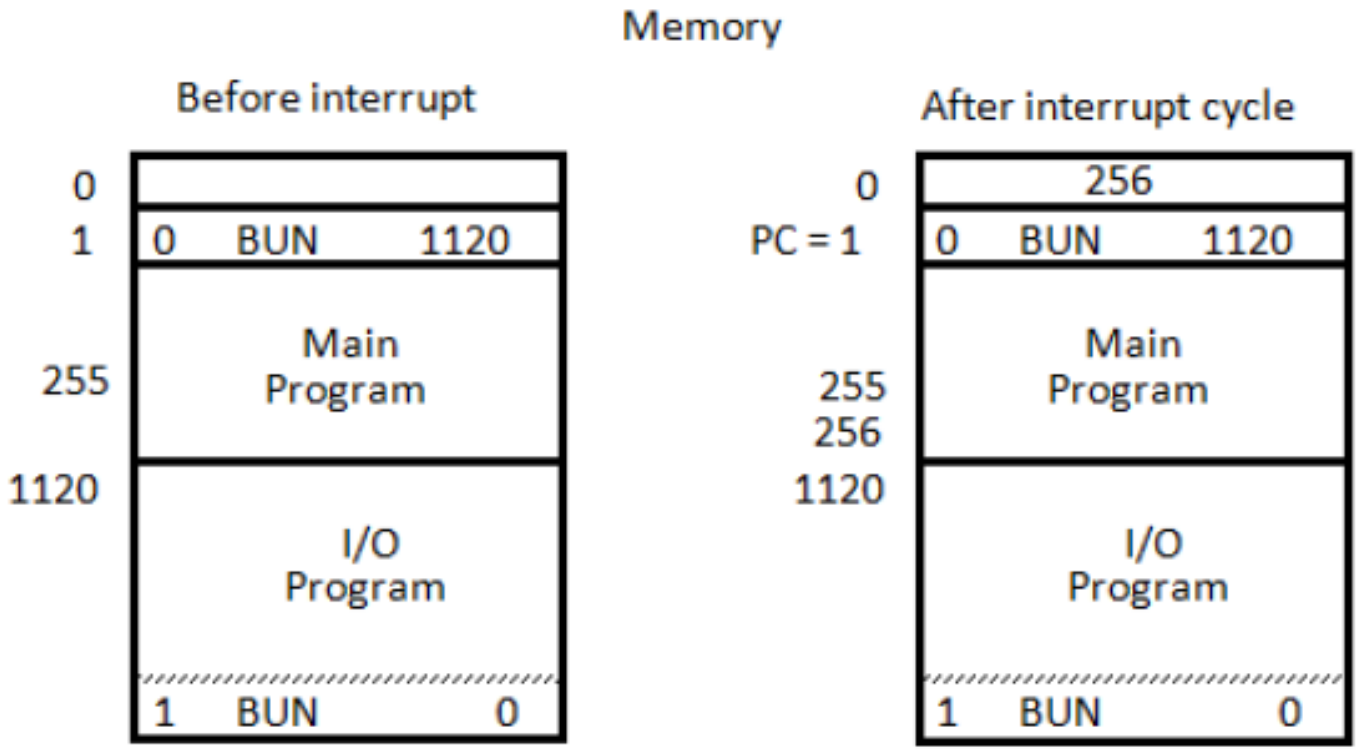



Figure 2.14: Demonstration of the interrupt cycle

- Suppose that an interrupt occurs and R = 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.

- 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.

- 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.

• The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

# Complete Computer Description: Flowchart for basic computer

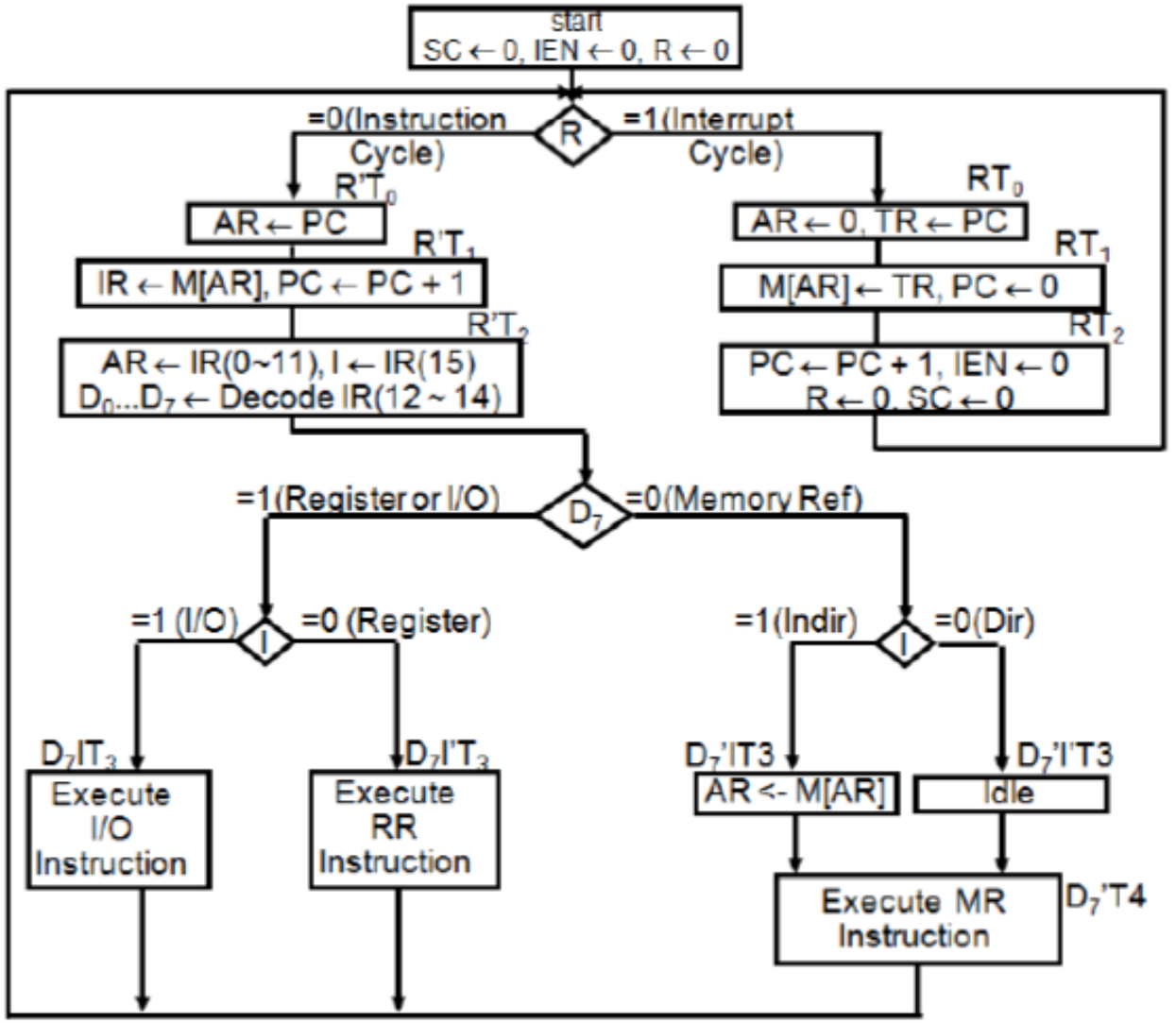
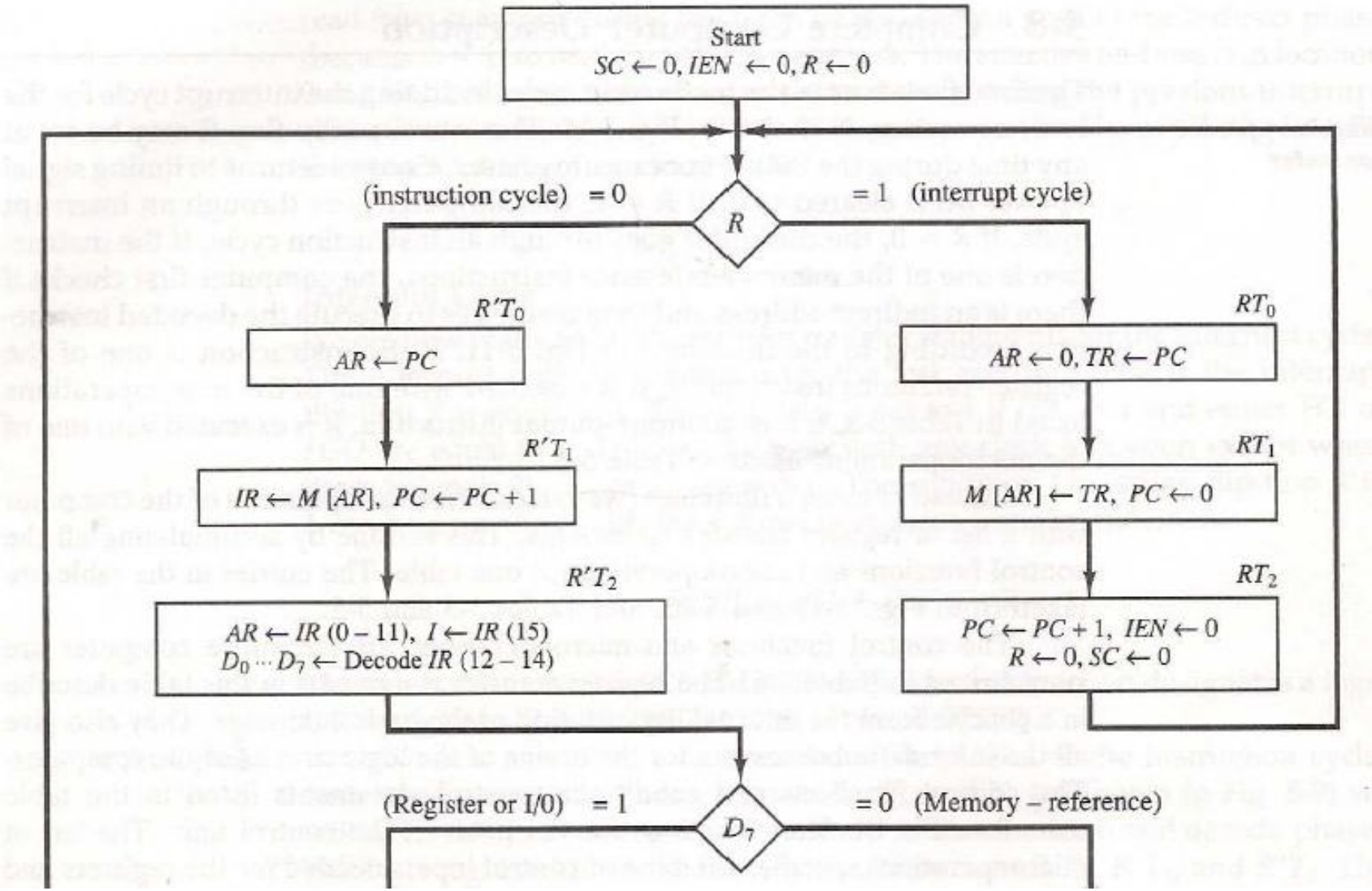



Figure 2.15: Flowchart for computer operation

# Design of Basic Computer :



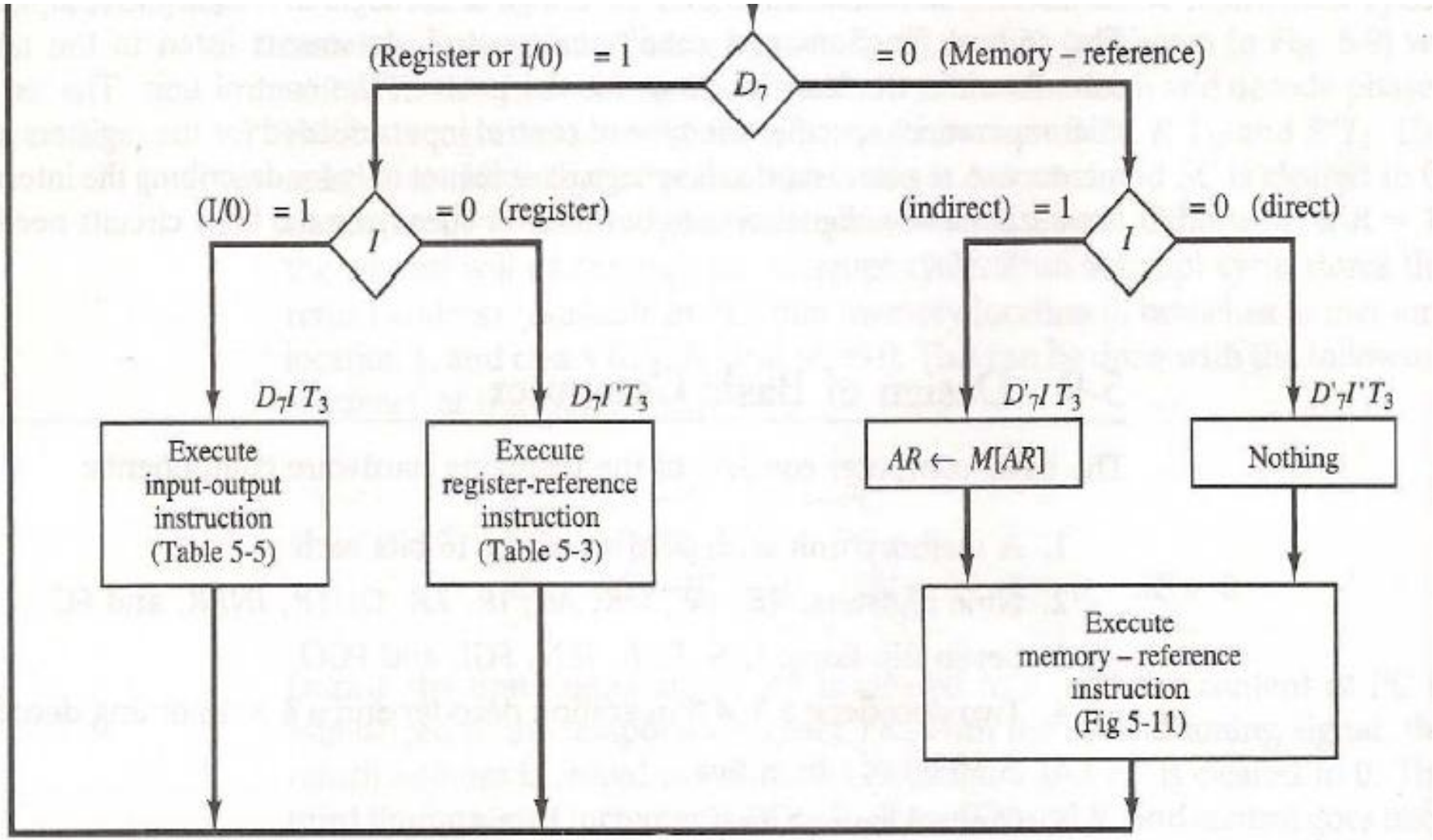



**Figure 5-15** Flowchart for computer operation.

The basic computer consists of the following hardware components:

1. A memory unit with 4096 words of 16 bits each
2. Nine registers: *AR*, *PC*, *DR*, *AC*, *IR*, *TR*, *OUTR*, *INPR*, and *SC*
3. Seven flip-flops: *I*, *S*, *E*, *R*, *IEN*, *FGI*, and *FGO*
4. Two decoders: a 3 × 8 operation decoder and a 4 × 16 timing decoder
5. A 16-bit common bus
6. Control logic gates
7. Adder and logic circuit connected to the input of *AC*

# Design of Accumulator Unit:

- The circuits associated with the AC register are shown in figure 2.16.

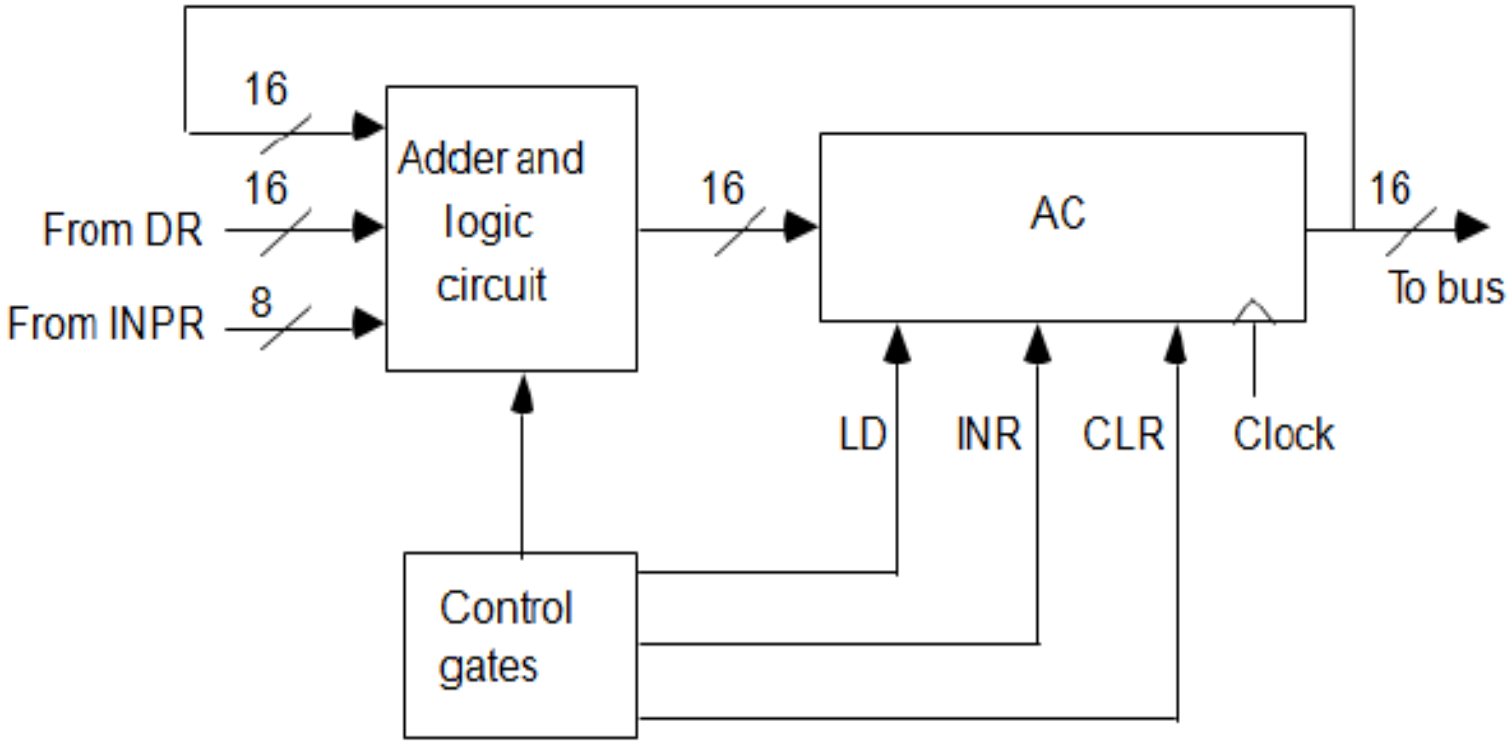



Figure 2.16: Circuits associated with AC

- 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.
- In order to design the logic associated with AC, it is necessary to extract all the statements that change the content of AC.

D0T5: AC ← AC ∧ DR AND with DR
D1T5: AC ← AC + DR Add with DR
D2T5: AC ← DR Transfer from DR
pB11: AC(0-7) ← INPR Transfer from INPR
rB9: AC ← AC' Complement
rB7: AC ← shr AC, AC(15) ← E Shift right
rB6: AC ← shl AC, AC(0) ← E Shift left
rB11: AC ← 0 Clear
rB5: AC ← AC + 1 Increment

- The gate structure that controls the LD, INR, and CLR inputs of AC is shown in figure

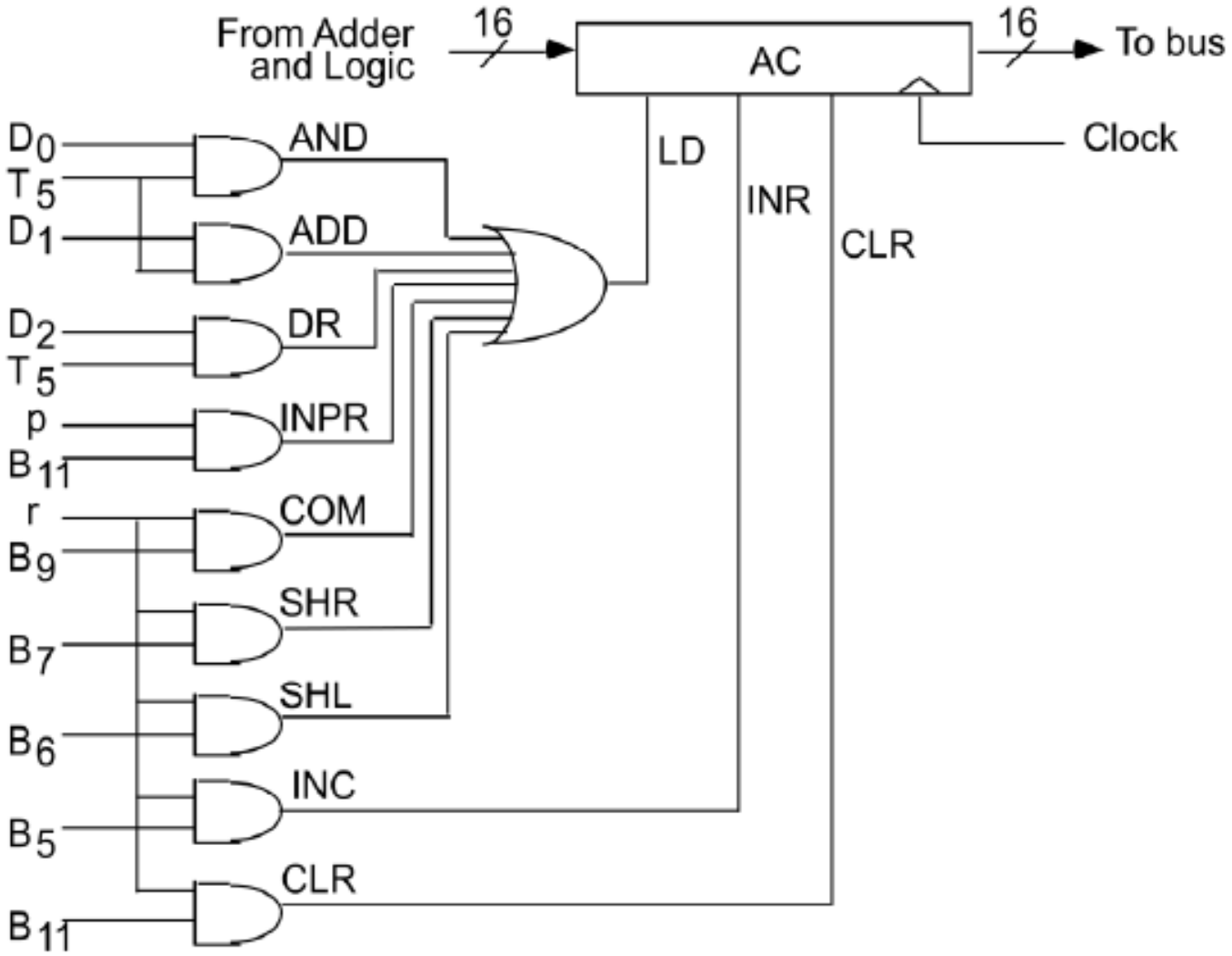



**Figure 2.17: Gate structure for controlling the LD, INR, and CLR of AC**

- The gate configuration is derived from the control functions in the list above.
- The control function for the clear microoperation is rB11, where r = D7I’T3 and B11 = IR (11).
- 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 microoperation is connected to the INR input of the register.
- The other seven microoperations 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 are marked with a symbolic name and used in the design of the adder and logic circuit.