Instruction codes

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

- It is usually divided into parts
  1. operation code
  2. number of bits

- **The operation code** of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.

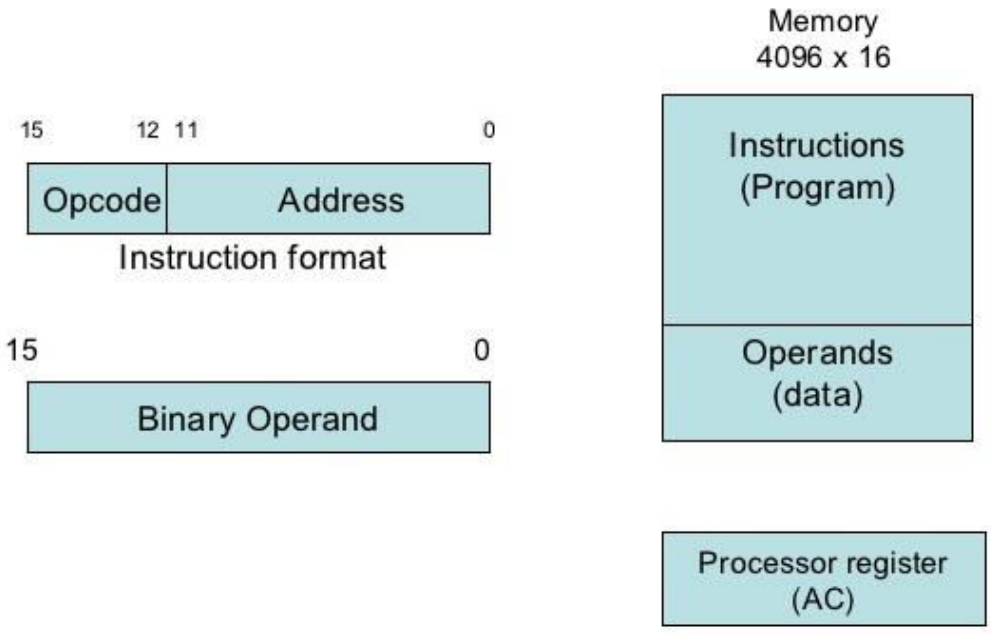
- **The number of bits** required for the operation code of an instruction depends on the total number of operations available in the computer.

- The operation code must consist of at least n bits for a given 2^n operations.

- The operation part of an instruction code specifies the operation to be performed. This operation must be performed on some data stored in processor registers or in memory.

Stored Program Organization

- Instruction code format is divided in two parts.
- The first part specifies the operation to be performed and the second specifies an address.



- Instructions are stored in one section of memory and data in another. For a memory unit with 4096 words we need 12 bits to specify an address since 2^{12} = 4096.
- If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.

- The operation is performed with the memory operand and the content of AC. For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register. They do not need an operand from memory.

INDIRECT ADDRESS

- Consider the following instruction format for direct and indirect addressing mode.

- It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by 1. The mode bit is 0 for a direct address and 1 for an indirect address.The instruction format for direct and indirect addressing mode is shown below:

Instruction Format

Direct Addressing Mode

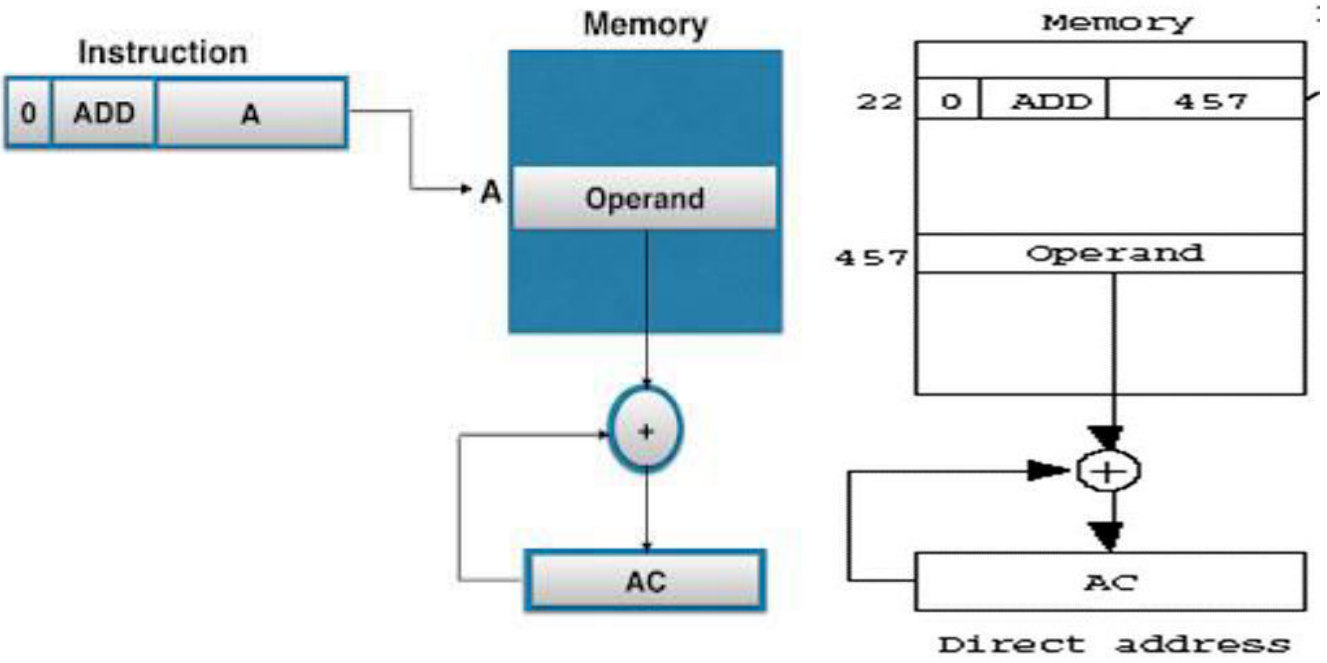
- is also known as “Absolute Addressing Mode”.In this mode the address of data(operand) is specified in the instruction itself.That is, in this type of mode, the

operand resides in memory and its address is given directly by the address field of the instruction.

For example: Consider the instruction:

ADD A - Means add contents of cell A to accumulator .

It Would look like as shown below:



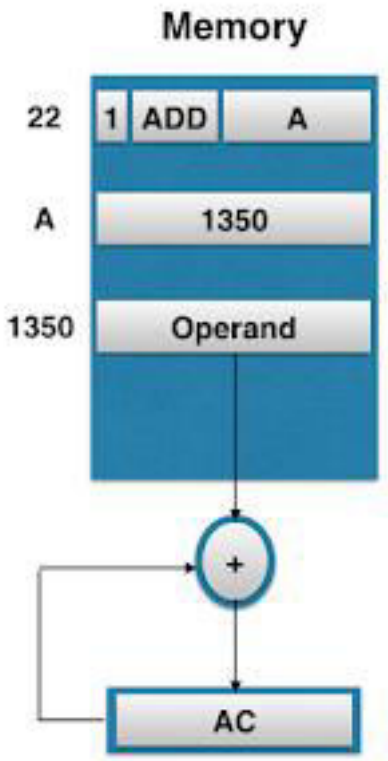
Indirect Addressing

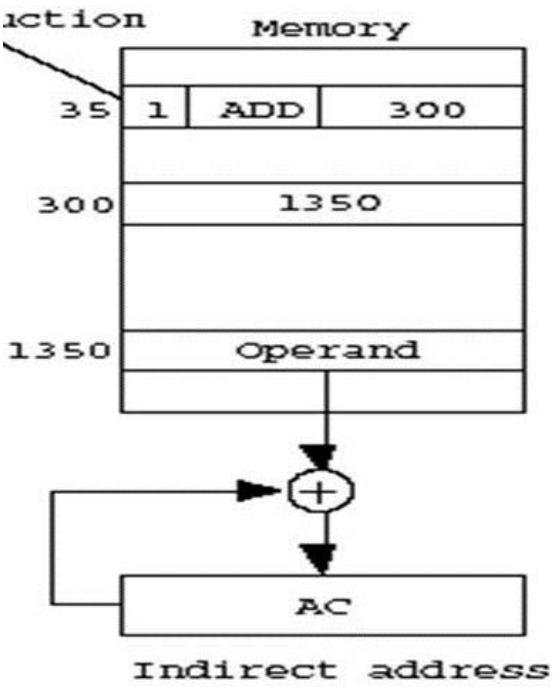
- In this mode, the address field of instruction gives the memory address where on, the operand is stored in memory.

For example: Consider the instruction:

ADD (A) Means adds the content of cell pointed to contents of A to Accumulator.

It look like as shown in diagram below:





COMPUTER REGISTERS

- The following table shows list of registers.

List of BC Registers

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

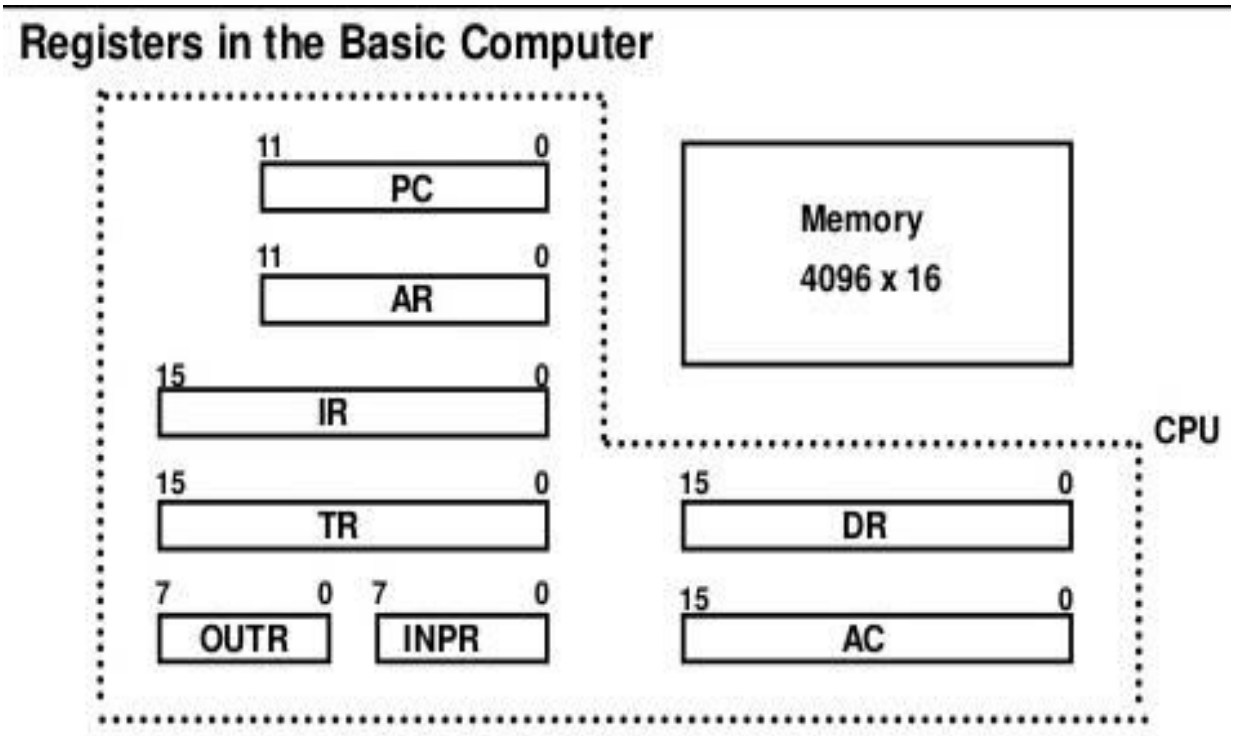
- The data register (DR) holds the operand read from memory. The accumulator (AC) register is a generalpurpose processing register.

- The instruction read form 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 since this is the width of a memory address. The program counter (PC) also has 12 bits and it holds address of the next instruction to be read from memory after the current the instruction is executed.

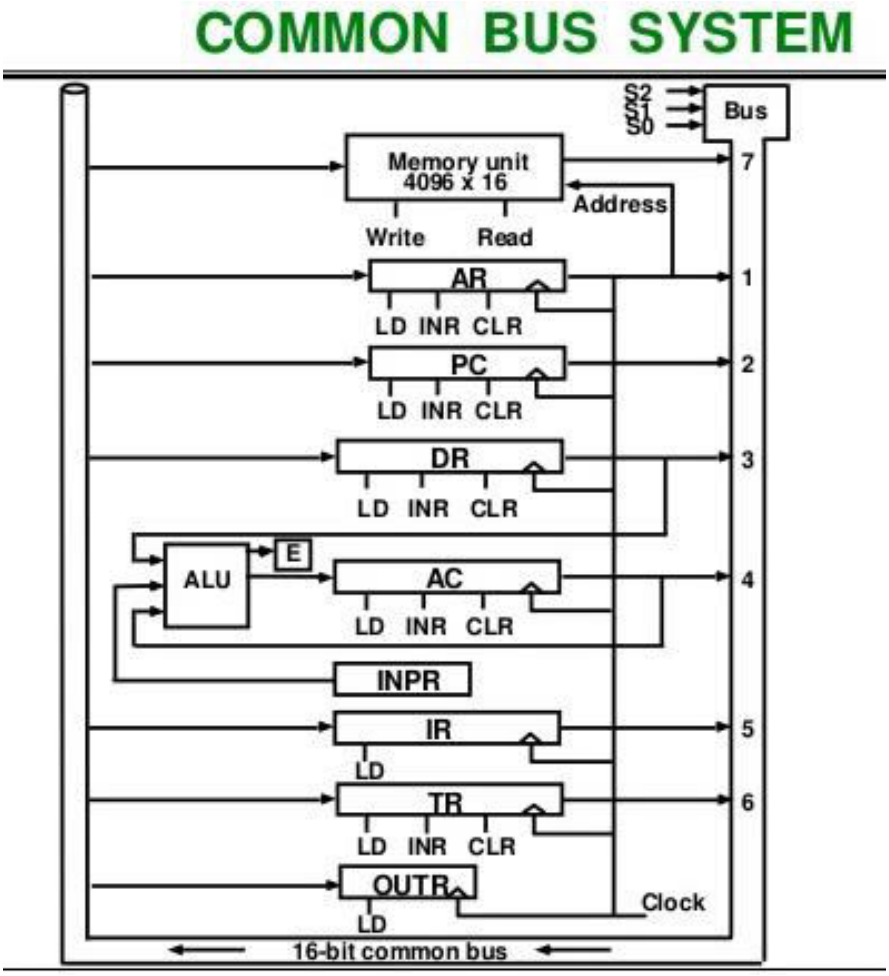
- Two registers are used for input and output. The input register (INPR) receives an 8-bit

character from an input device. The output register (OUTR) holds an 8-bit character for an output device.



Common Bus System

- The connection of the registers and memory of the basic computer to a common bus system is shown in following diagram.



- 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 S2S1, 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 S2 S1 S0= 011 since this is the binary value of decimal 3.

- The lines from the common bus are connected to the inputs of each registerand the data input 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 S2 S1 S0 = 111.

- 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 or PC receives 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 (LSB) in the bus.

- INPR receives a character from an input device which is then transferred to AC. OUTR receives a character from AC and delivers it to an output device. There is no transfer from OUTR to any of the other registers.

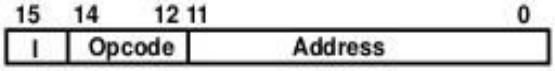
Computer Instruction

- The basic computer has three instruction code formats, as shown in following diagram. Each format has 16 bits.

BASIC COMPUTER INSTRUCTIONS

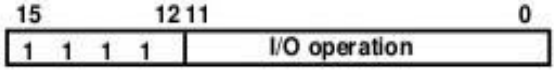
• **Basic Computer Instruction Format**

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



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

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



- The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.

- **A memory reference instruction** uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address.

- **The register-reference instructions** are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction. A register-reference instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation.

- **An input-output instruction** does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.

- The instructions for the computer are listed in Table below

TABLE 5-2 Basic Computer Instructions

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

- The hexadecimal code is equal to the equivalent hexadecimal number of the binary code used for the instruction. By using the hexadecimal equivalent we reduced the 16 bits of an instruction code to four digits.

- A memory-reference instruction has an address part of 12 bits. The address part is denoted by three x's and stand for the three hexadecimal digits corresponding to the 12-bit address.

- The last bit of the instruction is designated by the symbol I. When I = 0, the last four bits of an instruction have a hexadecimal digit equivalent from 0 to 6 since the last bit is 0. When I = 1, the hexadecimal digit equivalent of the last four bits of the instruction ranges from 8 to E since the last bit is I.

- **Register-reference instructions** use 16 bits to specify an operation. The leftmost four bits are always 0111, which is equivalent to hexadecimal 7.

- **The input-output instructions** also use all 16 bits to specify an operation. The last four bits are always 1111, equivalent to hexadecimal F.

Instruction Set Completeness

- The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

  1. Arithmetic, logical, and shift instructions
  2. Instructions for moving information to and from memory and processor registers
  3. Program control instructions together with instructions that check statues conditions
  4. Input and output instructions

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.

- There are two major types of control organization

1. Hardwired control

- In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.

- Requires changes in the wiring among the various components if the design has to be modified or changed.

2. Micro programmed control

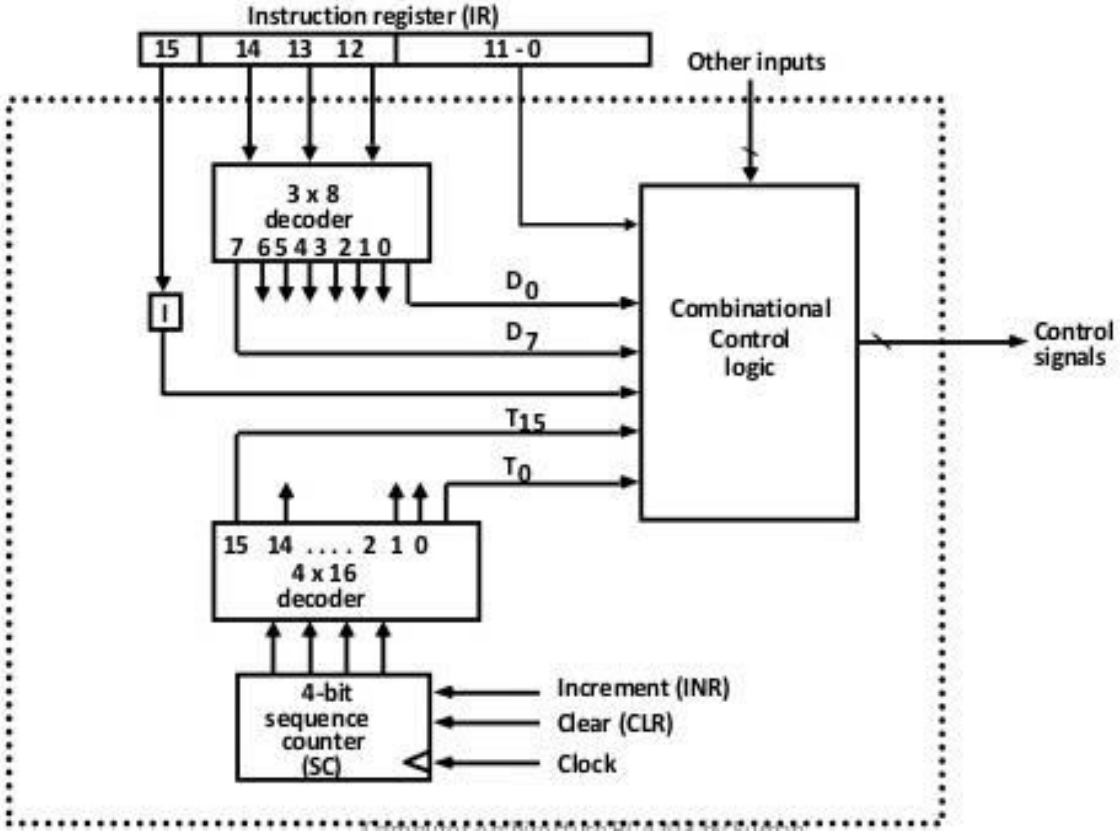
- The control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro operations.

- Required changes or modifications can be done by updating the micro program in control memory

➢ The block diagram of the control unit is shown below

Hardwired Timing And Control Unit

Control unit of Basic Computer



➢ It consists of two decoders, a sequence counter, and a number of control logic gates.

➢ An instruction read from memory is placed in the instruction register (IR).

- The operation code in bits 12 through 14 are decoded with a 3 × 8 decoder. The eight outputs of the decoder are designated by the symbols D0 through D7.

- Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates.

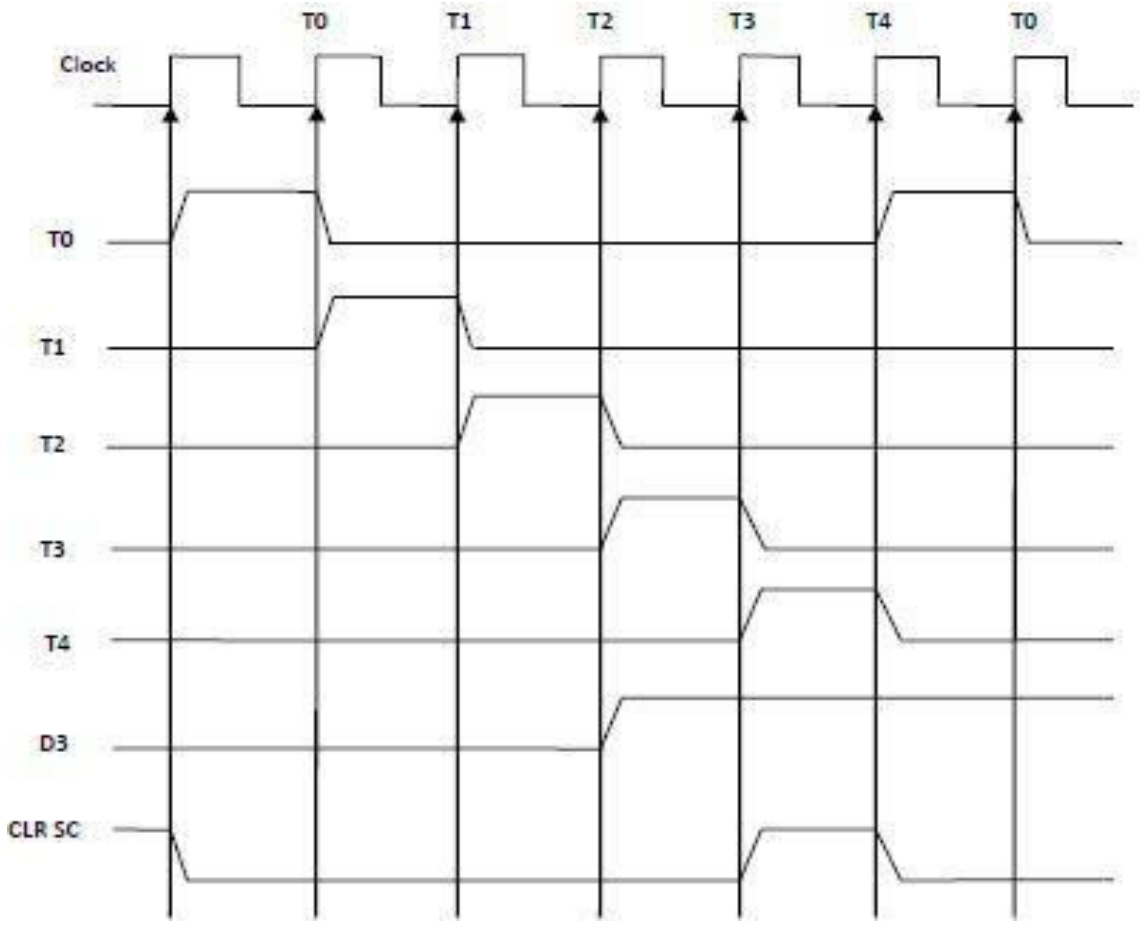
- The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T0 through T15.

- **The sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 × 16 decoder. Once in a while, the counter is cleared to 0, causing the next active timing signal to be T0.**

- 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

  D3 T4: SC ← 0

- The timing diagram below shows the time relationship of the control signals.



- Initially, the CLR input of SC is active.
- SC is incremented with every positive clock transition, unless its CLR input is active. This produces the sequence of timing signals T0, T1, T2, T3, T4, and so on.
- If SC is not cleared, the timing signals will continue with T5, T6, up to T15 and back to T0.

Instruction Cycle

- The program is executed in the computer by going through a cycle for each instruction.

- Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases.

- In the basic computer each instruction cycle consists or the following phases:

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

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

Fetch and Decode

- The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

T_0: AR ← PC

T_1: IR ← M[AR], PC ← PC + 1

T2: D0,…, D1 ← Decode IR (12-14), AR ← IR(0 −11), 1 ← IR (15)

- AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T0.

- The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1. At the same time, PC is incremented by one.

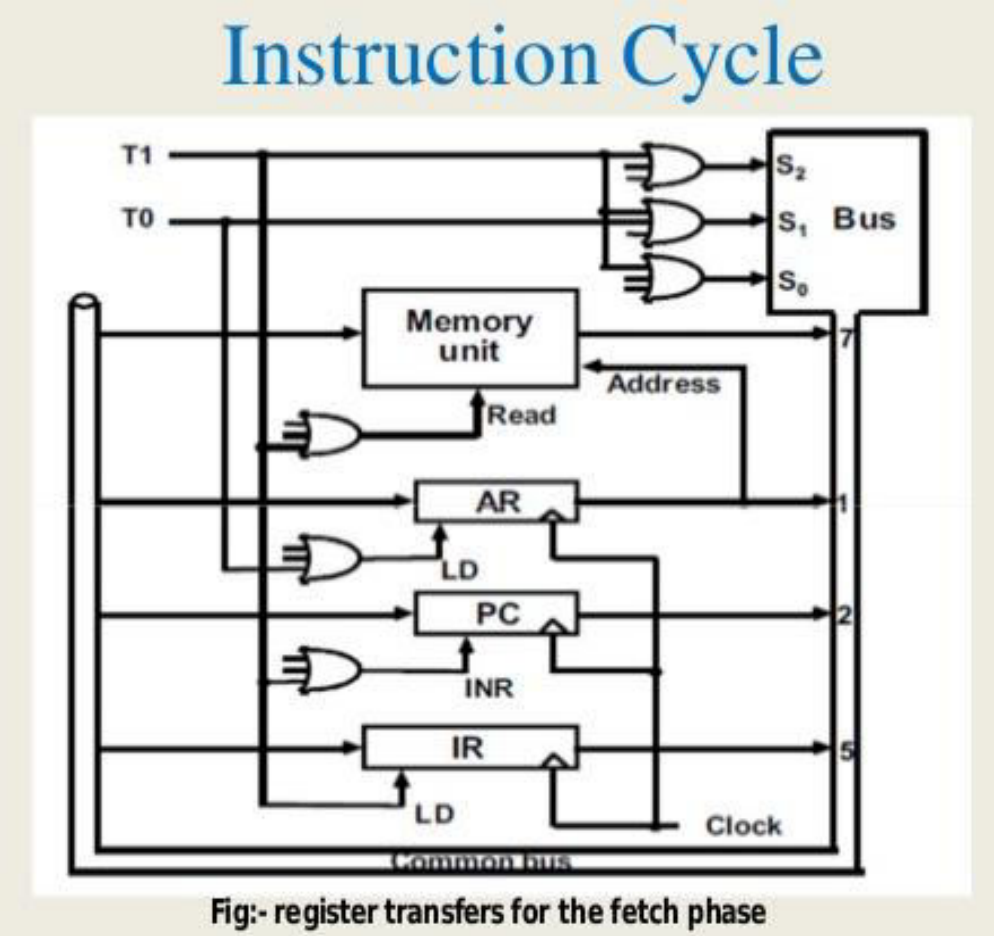


Fig:- register transfers for the fetch phase

- Diagram shows how the first two register transfer statements are implemented in the bus system.

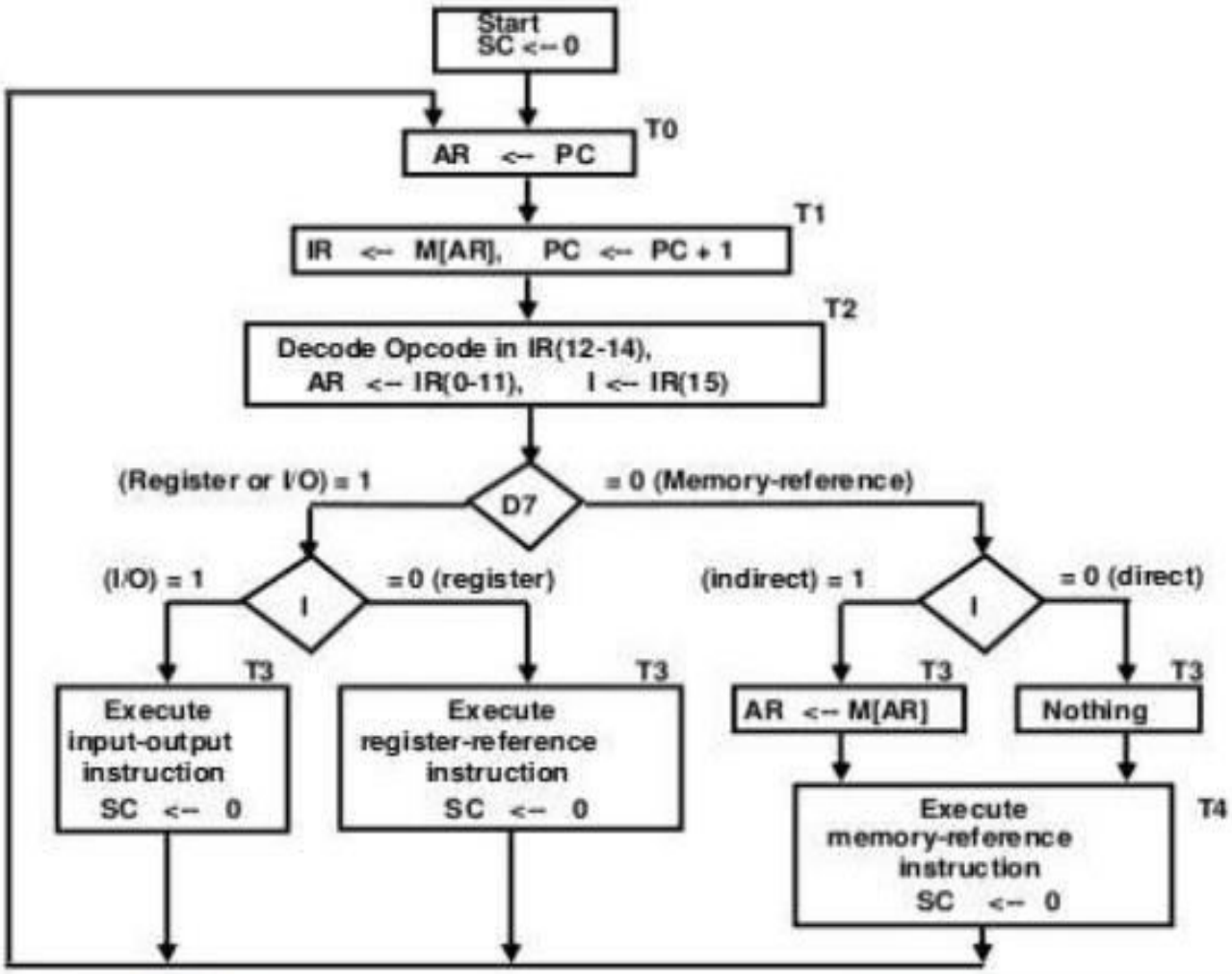
- To provide the data path for the transfer of PC to AR we must apply timing signal T_0 to achieve the following connection.:
  1. Place the content of PC onto the bus by making the bus selection inputs $S_2S_1S_0$ equal to 010.
  2. Transfer the content of the bus to AR y enabling the LD input of AR.

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

  T1: IR ← M[AR], PC ← PC + 1

- It is necessary to use timing signal T1 to provide the following connections in the bus system.
  1. Enable the read input of memory.
  2. Place the content of memory onto the bus by making $S_2S_1S_0 = 111$.
  3. Transfer the content of the bus to IR by enabling the LD input of IR.
  4. Increment PC by enabling the INR input of PC.

Determine The Type of Instruction



Register-Reference Instructions

- Register-reference instructions are recognized by the control when D7 = 1 and I = 0.

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 $IR(0–11)$ 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

Memory-Reference Instructions

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 AR and was placed there during timing signal T_2 when I = 0, or during timing signal T_3 when I = 1
- Memory cycle is assumed to be short enough to complete in a CPU cycle
- The execution of MR instruction starts with T_4

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.

- The micro operations that execute this instruction are :

D_0T_4 : DR ← M[AR]
D_0T_5: AC ← AC Λ DR, SC ← 0

ADD to AC

- The 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 micro operations needed to execute this instruction are

D_1T_4: DR ← M[AR]
D_1T_5: AC ← AC + DR, E ← Cout, SC ← 0

LDA : Load to AC

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

D_2T_4: DR ← M[AR]
D_2T_5: AC ← DR, SC ← 0

STA : Store AC

- This instruction stores the content of AC into the memory word specified by the effective address.
- we can execute this instruction with one micro operation:

  D_3T_4: M[AR] ← AC, SC ← 0

BUN : Branch Unconditionally

- 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 micro operation:

  D_4T_4: 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.

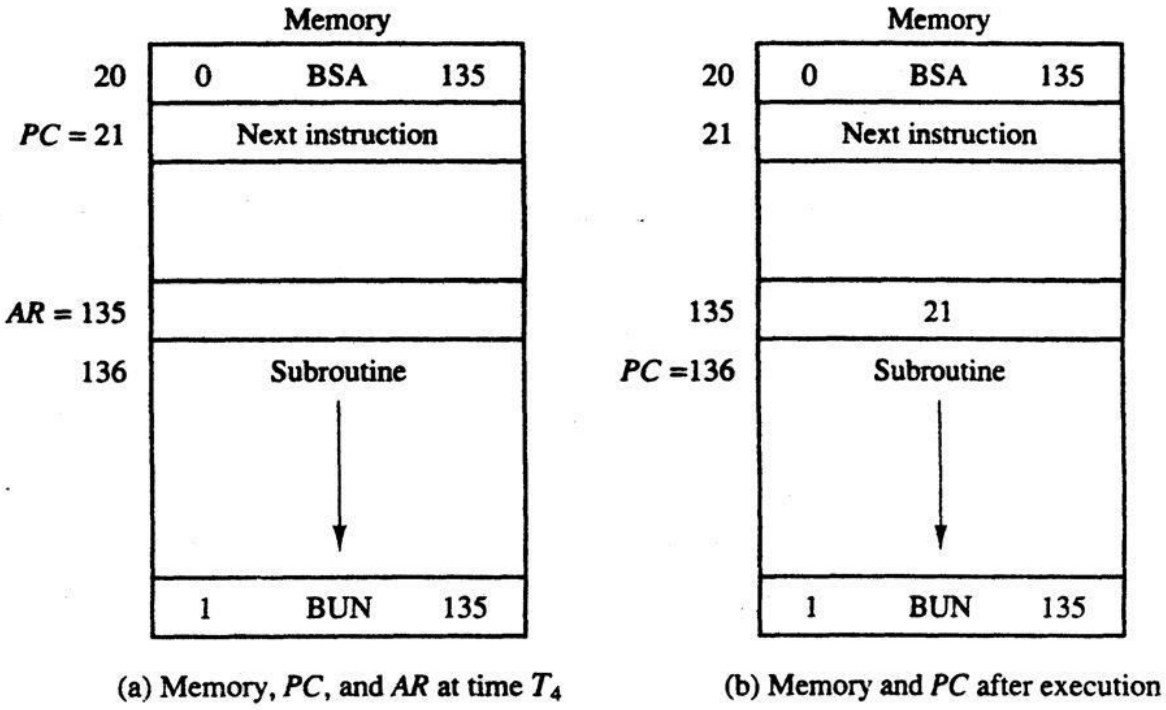
- When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address.

- The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subordinate.

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

- A numerical example that demonstrates how this instruction is used with a subordinate is shown in following diagram.

BSA: Branch and Save Return Address



(a) Memory, *PC*, and *AR* at time T_4 (b) Memory and *PC* after execution

- 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:

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

- 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 subordinate 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.

ISZ : Increment and Skip if Zero

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

Input-Output and Interrupt

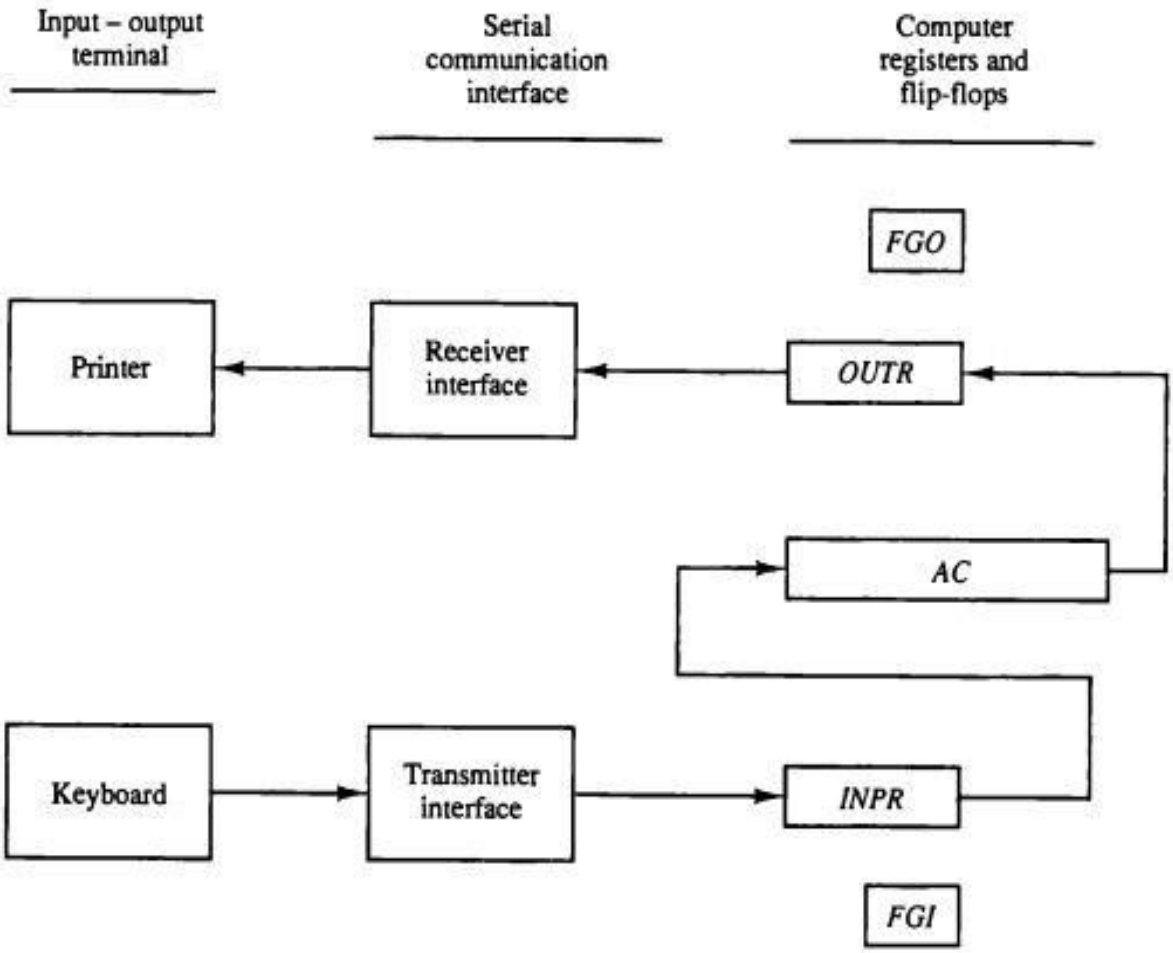
- Instructions and data stored in memory must come from some input device. Computational results must be transmitted to the user through some output device.

Input-Output Configuration

- 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 diagram.

- 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. The flag bit is set to 1 when new information is available in the input device and is cleared to 0.
- When the information is accepted by the computer. The 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.
- 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.

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.

Program Interrupt

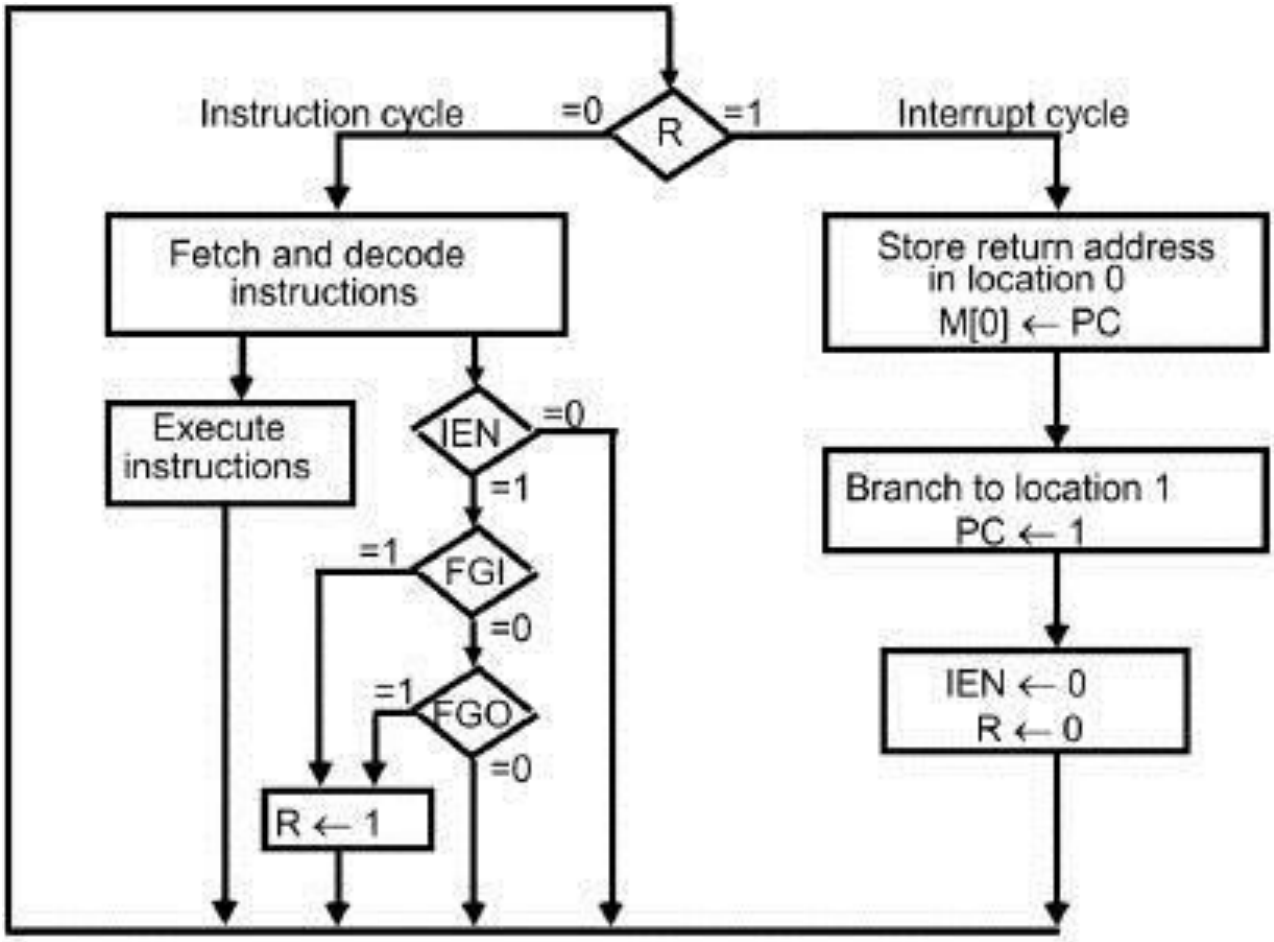
- The computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer.

- At the maximum rate, the computer will check frequently the flag between each transfer. The computer is wasting time while checking the flag instead of doing some other useful processing task.

- An alternative to this is to let the external device inform the computer when it is ready for the transfer. In the meantime the computer can be busy with other tasks. This type of transfer uses the **interrupt facility.**

- While the computer is running a program, it does not check the flags. However, when a flag is set, the computer is interrupted from proceeding with the current program and is informed of the fact that a flag has been set.

- The interrupt enable flip-flop **IEN** can be set and cleared with two instructions. When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer. When IEN is set to 1 (with the ION instruction), the computer can be interrupted.



- 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.

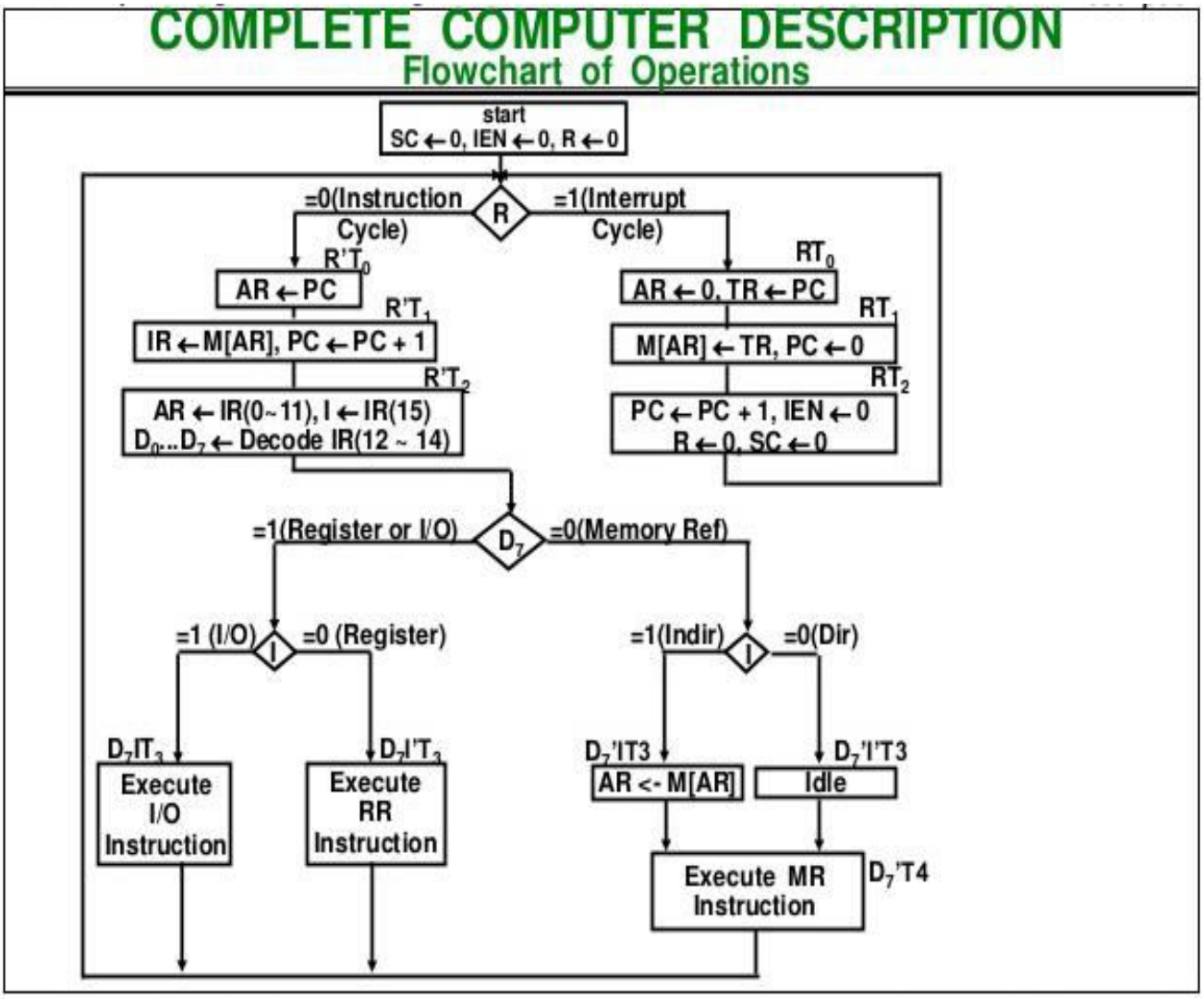
- If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information. In this case, control continues with the next instruction cycle.

- 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.

- **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.

Complete Computer Decription

- The final flowchart of the instruction cycle, including the interrupt cycle for the basic computer, is shown in following diagram



Design of Basic Computer

- The basic computer consists of the following hardware components
    1. A memory unit with 4096 words of 16bits
    2. Nine registers : AR, PC, DR, AC, IR, TR, OUTR, INPR, and SC
    3. Seven F/Fs : I, S, E, R, IEN, FGI, and FGO
    4. Two decoder in control unit : 3 x 8 operation decoder, 4 x 16 timing decoder
    5. A 16-bit common bus
    6. Control Logic Gates
    7. Adder and Logic circuit connected to the AC input

Control Logic Gates

1. Signals to control the inputs of the nine registers
2. Signals to control the read and write inputs of memory
3. Signals to set, clear, or complement the F/Fs
4. Signals for S2 S1 S0 to select a register for the bus
5. Signals to control the AC adder and logic circuit

Control of Registers and Memory

- The control inputs of the registers connected to the common bus :
  1. Load the register (LD).
  2. Increment the register (INR).
  3. Clear the register (CLR).
- Suppose that to drive the gate structure associated with control input of AR then

Address Register; AR

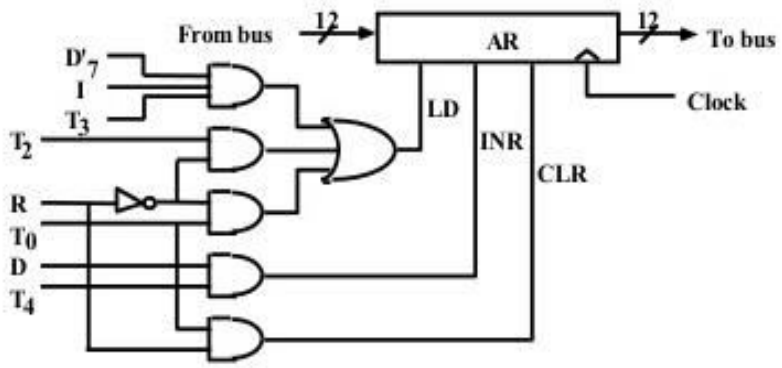
Scan all of the register transfer statements that change the content of AR:

$R'T_0$:	$AR \leftarrow PC$	$LD(AR)$
$R'T_2$:	$AR \leftarrow IR(0\text{-}11)$	$LD(AR)$
D'_7IT_3:	$AR \leftarrow M[AR]$	$LD(AR)$
RT_0:	$AR \leftarrow 0$	$CLR(AR)$
D_5T_4:	$AR \leftarrow AR + 1$	$INR(AR)$

$LD(AR) = R'T_0 + R'T_2 + D'_7IT_3$

$CLR(AR) = RT_0$

$INR(AR) = D_5T_4$



Control of Single Flip-Flops

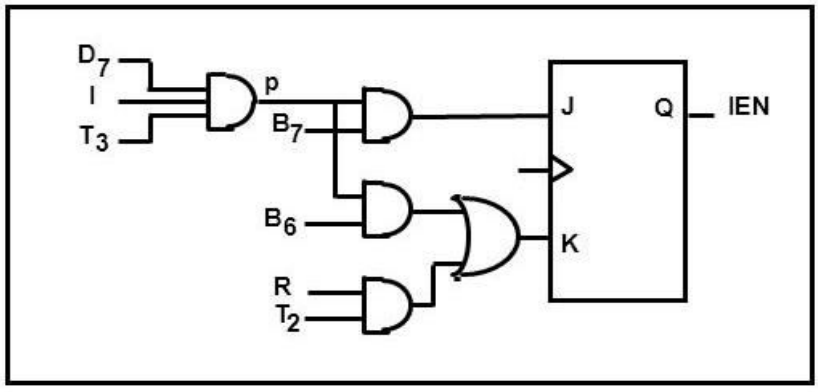
- The control gates for the seven flip-flops S, E, R, IEN, FGI, FGO, and I can be determined.
- It is shown that the statements that change the state of the Flip-flop IEN and JK flip-flop for the IEN, the complete control logic will be

CONTROL OF FLAGS

IEN: Interrupt Enable Flag

pB_7: IEN ← 1 (I/O Instruction)
pB_6: IEN ← 0 (I/O Instruction)
RT_2: IEN ← 0 (Interrupt)

$p = D_7IT_3$ (Input/Output Instruction)

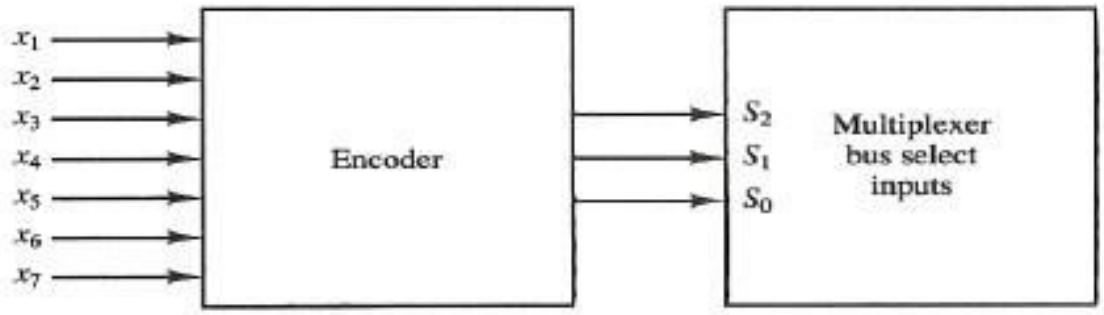


Control of Common Bus

- The 16-bit common bus is controlled by the three selection inputs S2, S1, and S0 the decimal number shown with each bus input must be applied to the selection inputs S2, S1, and S0 in order to select the corresponding register or memory.

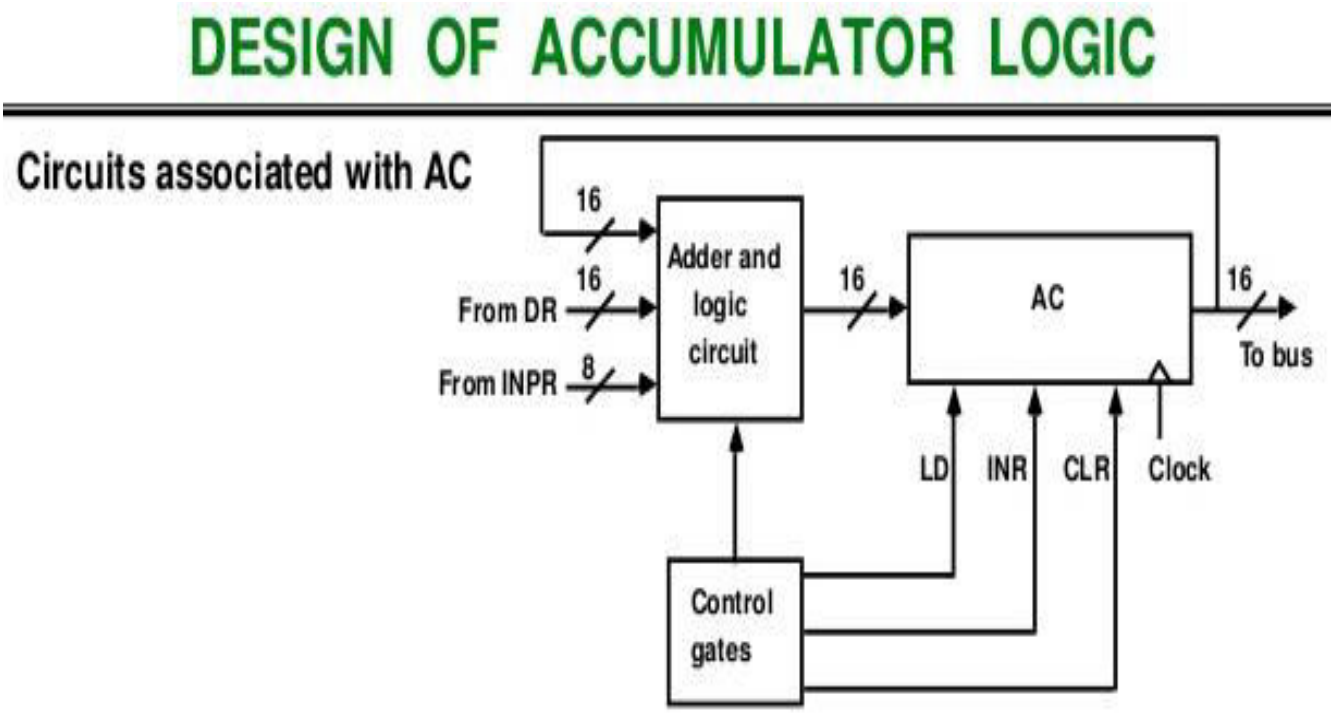
Inputs							Outputs			Register selected for bus
x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	0	0	0	1	*AR*
0	1	0	0	0	0	0	0	1	0	*PC*
0	0	1	0	0	0	0	0	1	1	*DR*
0	0	0	1	0	0	0	1	0	0	*AC*
0	0	0	0	1	0	0	1	0	1	*IR*
0	0	0	0	0	1	0	1	1	0	*TR*
0	0	0	0	0	0	1	1	1	1	Memory

- The placement of encoder at the inputs of bus selection logic is shown if following diagram.



Design of Accumulator Logic

- The circuits associated with the AC register shown in the diagram.



- 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.
- Moreover, the 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.

- Also, 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.

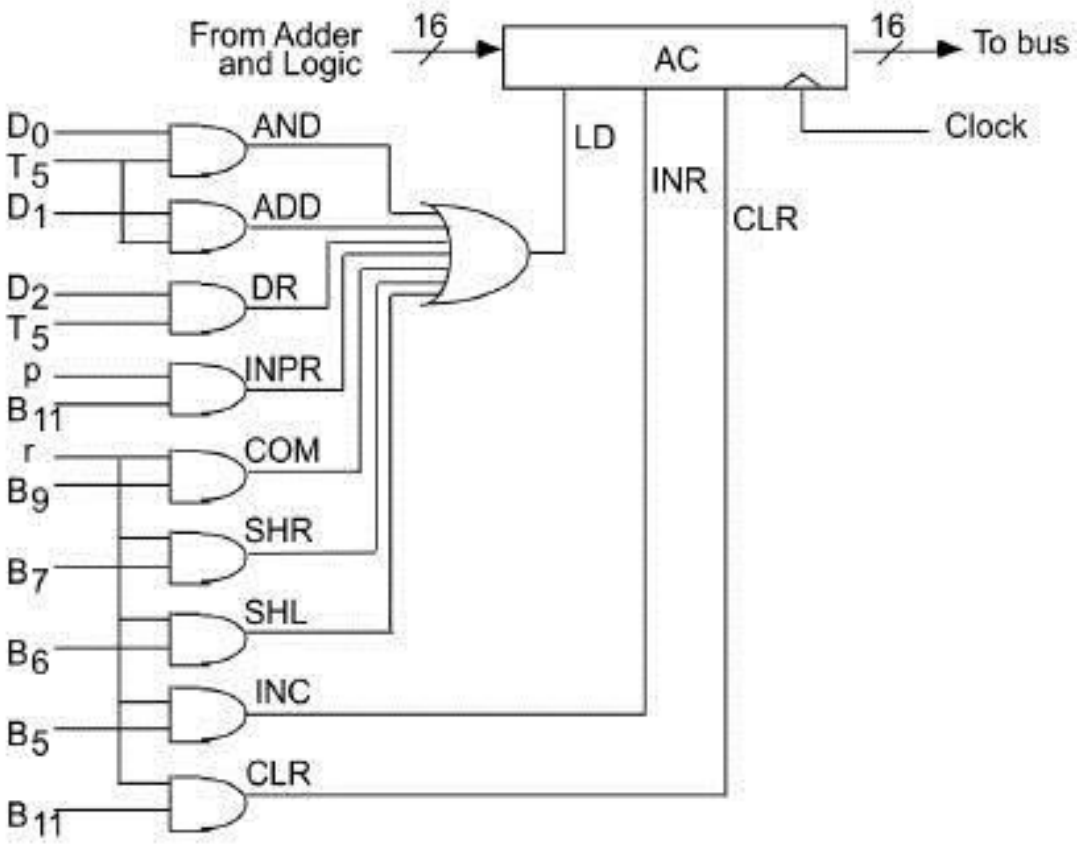
➢ In order to design the logic associated with AC, it is necessary to extract all the statements that change the content of AC.

All the statements that change the content of AC

D_0T_5:	AC ← AC ∧ DR	AND with DR
D_1T_5:	AC ← AC + DR	Add with DR
D_2T_5:	AC ← DR	Transfer from DR
pB_{11}:	AC(0-7) ← INPR	Transfer from INPR
rB_9:	AC ← AC′	Complement
rB_7 :	AC ← shr AC, AC(15) ← E	Shift right
rB_6 :	AC ← shl AC, AC(0) ← E	Shift left
rB_{11} :	AC ← 0	Clear
rB_5 :	AC ← AC + 1	Increment

Control of AC Register

➢ The gate structure that controls the LD, INR, and CLR inputs of AC shown in the diagram.



- The control function for the clear microoperation is rB_{11}, where $r = D7I'T_3$ and $B_{11} = IR(11)$.
- The output of the AND gate that generates this control function connected to the CLR input of the register.
- Similarly, the output of the gate that implements the increment micro operation connected to the INR input of the register.
- The other seven micro operations generated in the adder and logic circuit and loaded into AC at the proper time.