Module - II
Computer ORGANIZATION

Total - 50
(Module 1 + 2)
(38 + 12)

Written by : Pushpendra

ADDRESSING MODES -

- The way, the operand are chosing during Program execution on the addressing Mode of the instruction.

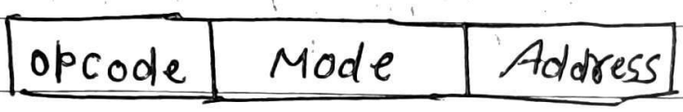
- Computer has a variety of addressing Mode -
  - to give Programing flexibility to the user
  - to use the bits in the address field of the instruction efficiently.

Advantage - It provide flexibility for writing programs that are more efficient with respect to number of instruction & execution time.

Program Counter -

There is one register in the computer called Program counter or PC. that keeps the tracks of the instruction in the Program stored in Memory.
It holds the address of the instruction to be executed next & is incremented each time an instruction is fetched from Memory.

ex- instruction format with a distinct addressing mode.



opcode → specifies the operation be performed
Mode → is used to locate the operands needed for operation.

Types -

Implicit / Implied mode -

In this mode, the operands are specified implicitly in the definition of the instructions.

complement Accumulator (CMA)
(operand is in the accumulator)

ii) Immediate Mode - In this mode, the operand is specified in the instruction itself.

- These are useful for initializing registers to constant value.

iii) Direct address Mode -

The instruction contains the exact Memory address where the operand (data) is located.

ex → if Memory location 1000, data is at then instruction will specify 1000

(iv) Indirect addressing Mode -

- The instruction gives the address of a location that holds another address where the actual data is stored.

ex - instruction says to go Memory location 2000, which contains 3000 & 3000 has actual data.

(v) Register Mode -

- the instruction refer to a register where the operand is stored.

ex - the instruction specifies that data is in Register 1.

(vi) Register Indirect Mode - The instruction refers to a register that holds the address of the data.

ex: MOV R_1, [R_2] R_2 says Memory address

[ ] → Means Memory

Register R1 contains the Memory address 4000, where the actual data is stored

(vii) Immidiate Mode - The instruction directly contains the value (data) to be used, not an address.

it fast since no Memory lookup is needed.

ex: MOV R_1, 200

(viii) Auto increment & Auto Decrement Mode

Auto Increment - like Register Indirect Mode, but the Register's Value increases after accessing the data.

ex: ADD [R1] + 1

Auto Decrement - Simil'ar, but register Value Decrease before accing the data.

ex: SUB - [R1]

(viii)

(ix) Relative addressing Modes -

- The instruction give an address relative to the program counter (PC), which is a Register tracking the current instruction location.

ex - if PC is at address 100, an offset might take you to 120.

EA = PC + offset
(Effective address)

x) Index Addressing -

- an index register is added to an address in the instruction to find the actual address of the data.

ex - instruction gives an address + XR (Index Register) to find the address.

EA = IR/XR + displacement

xi) Base Addressing - use a Base address stored in a register & an offset given in the instruction to get the final address.

EA = BAR + displacement
(Base Register)

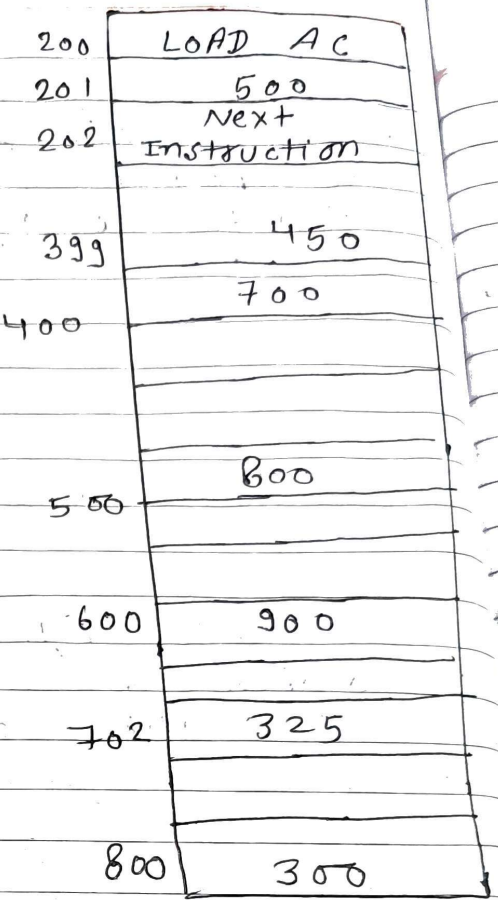
Q.

PC [200]

R1 [400]

XR [100]

[AC]



Address

- LOAD AC, 500

opcode operand

- Direct addressing Mode -

EA = 500, AC = 500

- Indirect addressing Mode -

EA = 800, AC = 300

- Register Direct -

EA = X, AC = 400

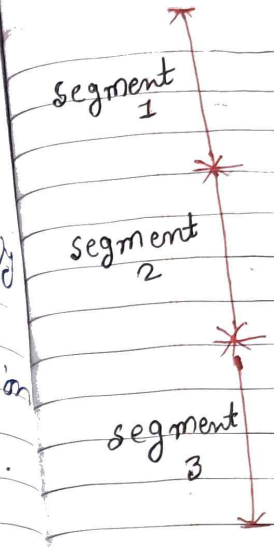
- Register Indirect -
  EA = 400 , AC = 700
- Implied addressing Mode -
  Not applicable.
  because the instructions in the given data have the accumulator by default.
- Indexed Addressing -
  EA = IR + displacement
  100 + 500 = 600
  AC = 900
- Auto Increment -
  EA = 400 , AC = 700
- Auto Decrement -
  EA = 399 , AC = 450
- Immidiate -
  EA = 201 , AC = 500
- Relative Addresing -
  EA = 202 + 500 = 702
  AC = 325

(EA = Next instruction + -]

→ PIPELINE –

- It is a technique of decomposing a sequential process into sub-operations, with each sub-process being executed in a segment that operates concurrently with all other segment.

- It can be visualized as a collection of processing segments through which binary information flows.



Fig

Instruction Pipeline –

- Fetch
- Decode
- Execute

Three stages of instruction in computer.

Clock Pulse
1.
2.
3.
4.
5.
6.
7

ex. $A_i * B_i + C_i$ $\quad i = 1$ to 7

The suboperation performed in each segment of the pipeline are follown:

$R_1 \leftarrow A_i$, $R_2 \leftarrow B_i$ Input A_i & B_i

$R_3 \leftarrow R_1 * R_2$, $R_4 \leftarrow C$ Multiply & Input C_i

$R_5 \leftarrow R_3 + R_4$ Add C_i to Product

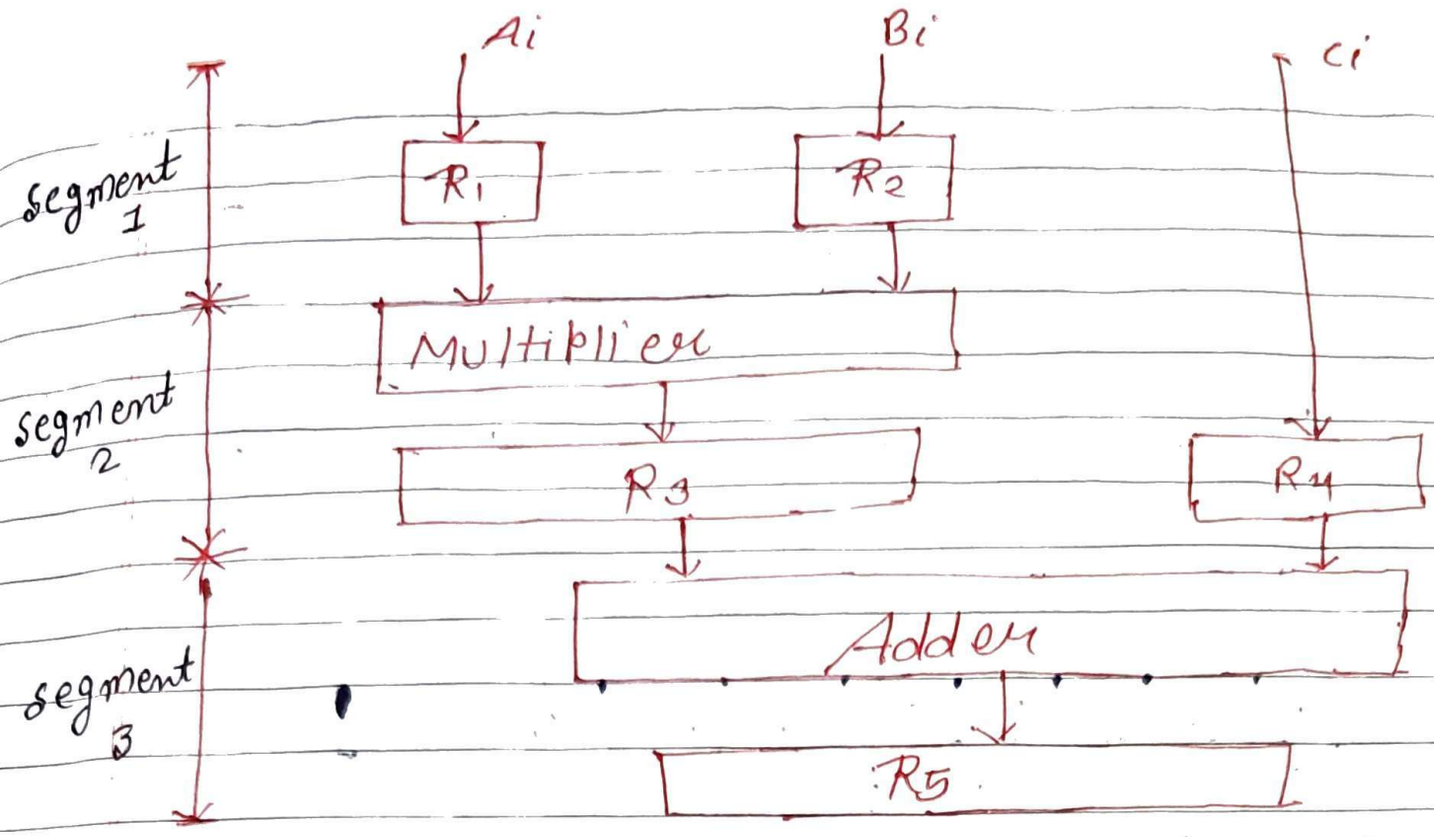


Fig :- Example of pipeline processing-

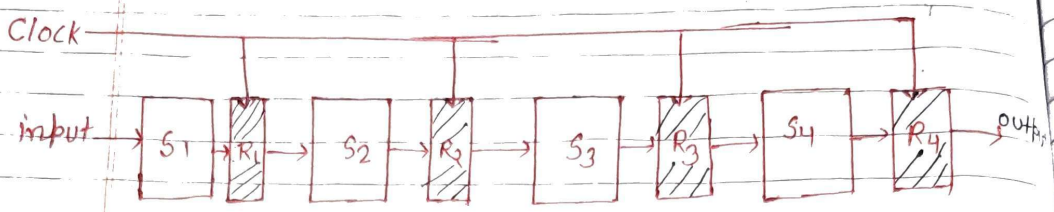
Clock Pulse	Segment 1		Segment 2		Segment 3
	R_1	R_2	R_3	R_4	R_5
1.	A_1	B_1	—	—	—
2.	A_2	B_2	$A_1 * B_1$	C_1	—
3.	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4.	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5.	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6.	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
			$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
					$A_7 * B_7 + C_7$

So in $A_i * B_i + C_i$

Where $i \Rightarrow 1-7$ (1 to 7)

$\Rightarrow$ 9 clock Pulse

☆. General Four - Segment Pipeline -



☆. Space time diagram -

	1	2	3	4	5	6	7	8	9	→ clock cycles
Segment 1	T_1	T_2	T_3	T_4	T_5	T_6				
2		T_1	T_2	T_3	T_4	T_5	T_6			
3			T_1	T_2	T_3	T_4	T_5	T_6		
4				T_1	T_2	T_3	T_4	T_5	T_6	

Q. Pipeline Question -

$(A_i + B_i) * (C_i + D_i)$ $\quad i = 1$ to 6

- $R_1 \leftarrow A_i$ $\quad R_3 \leftarrow C_i$
  $R_2 \leftarrow B_i$ $\quad R_4 \leftarrow D_i$ $\quad$ input A_i, B_i & C_i, D_i

- $R_5 \leftarrow R_1 + R_2$ $\quad R_6 \leftarrow C_i + D_i$
  ~~Multiply~~ Add R_6 to R_5

- $R_7 \leftarrow R_5 * R_6$
  Multiply R_5 & R_6

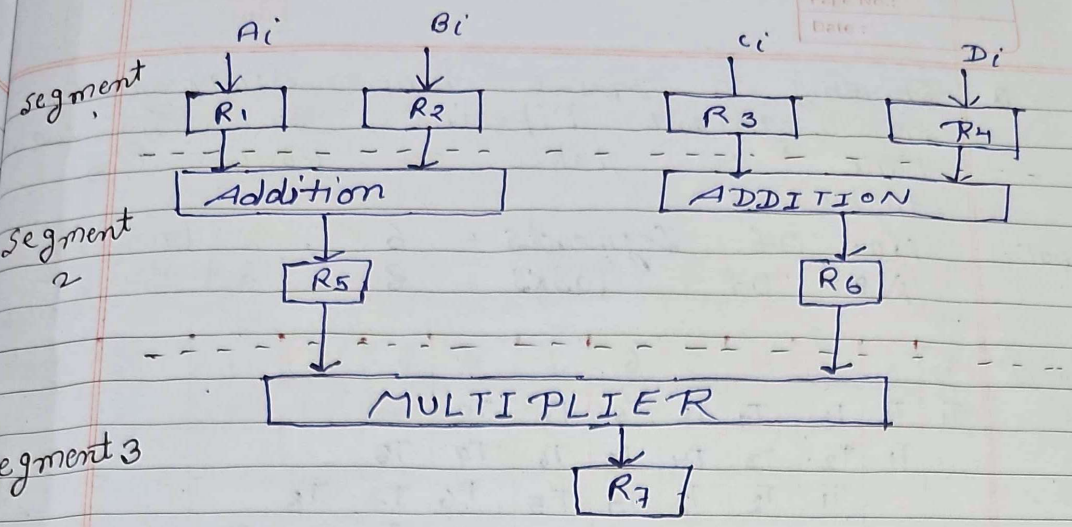


Fig: Example of Pipeline processing

	segment 1				segment 2		segment 3
Clock Pulse	R_1	R_2	R_3	R_4	R_5	R_6	R_7
1	A_1	B_1	C_1	D_1	—	—	—
2	A_2	B_2	C_2	D_2	(A_1+B_1)	(C_1+D_1)	—
3	A_3	B_3	C_3	D_3	(A_2+B_2)	(C_2+D_2)	$(A_1+B_1)*(C_1+D_1)$
4	A_4	B_4	C_4	D_4	(A_3+B_3)	(C_3+D_3)	$(A_2+B_2)*(C_2+D_2)$
5	A_5	B_5	C_5	D_5	(A_4+B_4)	(C_4+D_4)	$(A_3+B_3)*(C_3+D_3)$
6	A_6	B_6	C_6	D_6	(A_5+B_5)	(C_5+D_5)	$(A_4+B_4)*(C_4+D_4)$
					(A_6+B_6)	(C_6+D_6)	$(A_5+B_5)*(C_5+D_5)$
							$(A_6+B_6)*(C_6+D_6)$

Q. Showing a space time diagram for six segment pipeline, showing the time it takes to process 8 tasks.

Sol: No. Of Segments - 6, $K + (n-1)t_p$
No. Of Tasks - 8 $6 + 8 - 1 = 13$ cycles

Segment	1	2	3	4	5	6	7	8	9	10	11	12	13
1	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8					
2		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8				
3			T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8			
4				T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8		
5					T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	
6						T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8

★ Pipeline speed up-

- let n no. of tasks to be performed
- let K be the ~~time~~ number of segment pipeline with clock time tp.

Therefore, the first task T_1 require Time = $K * tp$

Now remaining $(n-1)$ task require $(n-1) * tp$

(The remaining (n-1) task emerge from the pipe at the rate of 1)

→ To complete n tasks with K segment pipeline =

$K + (n-1)$ clock cycles

for Non-Pipelined-
Time require to complete each task
$\Rightarrow t_n$

Time require to complete n task $\Rightarrow n * t_n$

Speed up $= \dfrac{n * t_n}{[k + (n-1)]t_p}$

$(\text{speed up})_{max} \Rightarrow \dfrac{t_n}{t_p}$

Q. A non-pipelined system takes 50 ns to process a task. the same task can be processed in a six-segment pipeline with a clock cycle of 10 ns. Determine the speed up ration of the pipeline for 100 tasks.

$t_n = 50$ ns
$k = 6$
$t_p = 10$ ns
$n = 100$

$S = \dfrac{n t_n}{(k + n - 1) t_p} \Rightarrow \dfrac{100 * 50}{(6 + 100 - 1) \cdot 10}$

$\Rightarrow 4.76$

$S_{max} = \dfrac{t_n}{t_p} = \dfrac{50}{10} = 5$

Instruction code-

→ computer Registers
→ computer instructions
→ Timing & control
→ Instruction cycle
→ Register - Reference instructions
→ Memory - Reference instructions
→ Input - output & interrupt

Program - A series of instruction that tells the computers what operation to perform which data to use, & in what order

Instruction - A code in binary (1s & 0s) that tells the computer to carry out a specific task.

Instruction cycle - The computer reads an instruction from memory & places it in a register. It then reads & decodes the instruction & perform the necessary steps to complete it.

Instruction code - A set of bits that guides the computers to perform a specific action.

Parts of Instruction code-

• opcode - specific the operation to Perform (like add, substract)

- Address (operand) – specifies where to find the data to use for the operation:

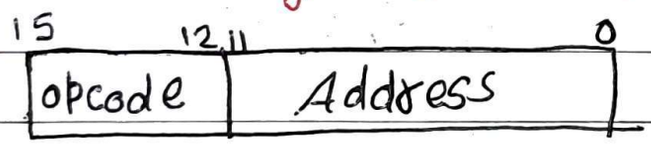
Instruction format

operation code (opcode) :-
- Group of bits that define the operation
- which type of operation will perform.

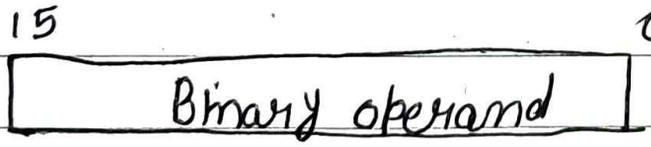
ex add, substract, multiply, shift, complement

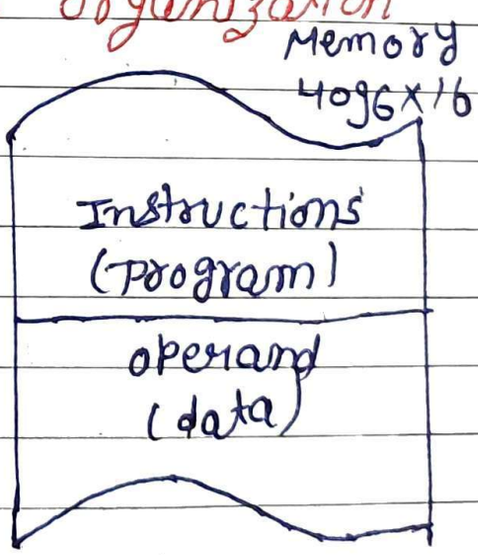
Address (operand):
- specifies the location of operand (register or Memory words)
- Memory words are specified by their address
- Register are specified by their K-bit binary code

Fig - Stored program organization



Instruction format





Processor register (accumulator or ac)

- The ability to store & execute instruction is the most Imp property of general purpose computer That type of Stored program is called Stored Program organization

- Accumulator (AC):-

→ computer that have a single-processor register usually assign to it the name accumulator & label it Ac.

→ The operation is performed with the Memory operand & the content of AC.

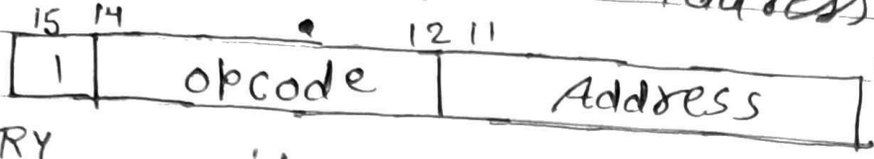
☆ Addressing of operand-

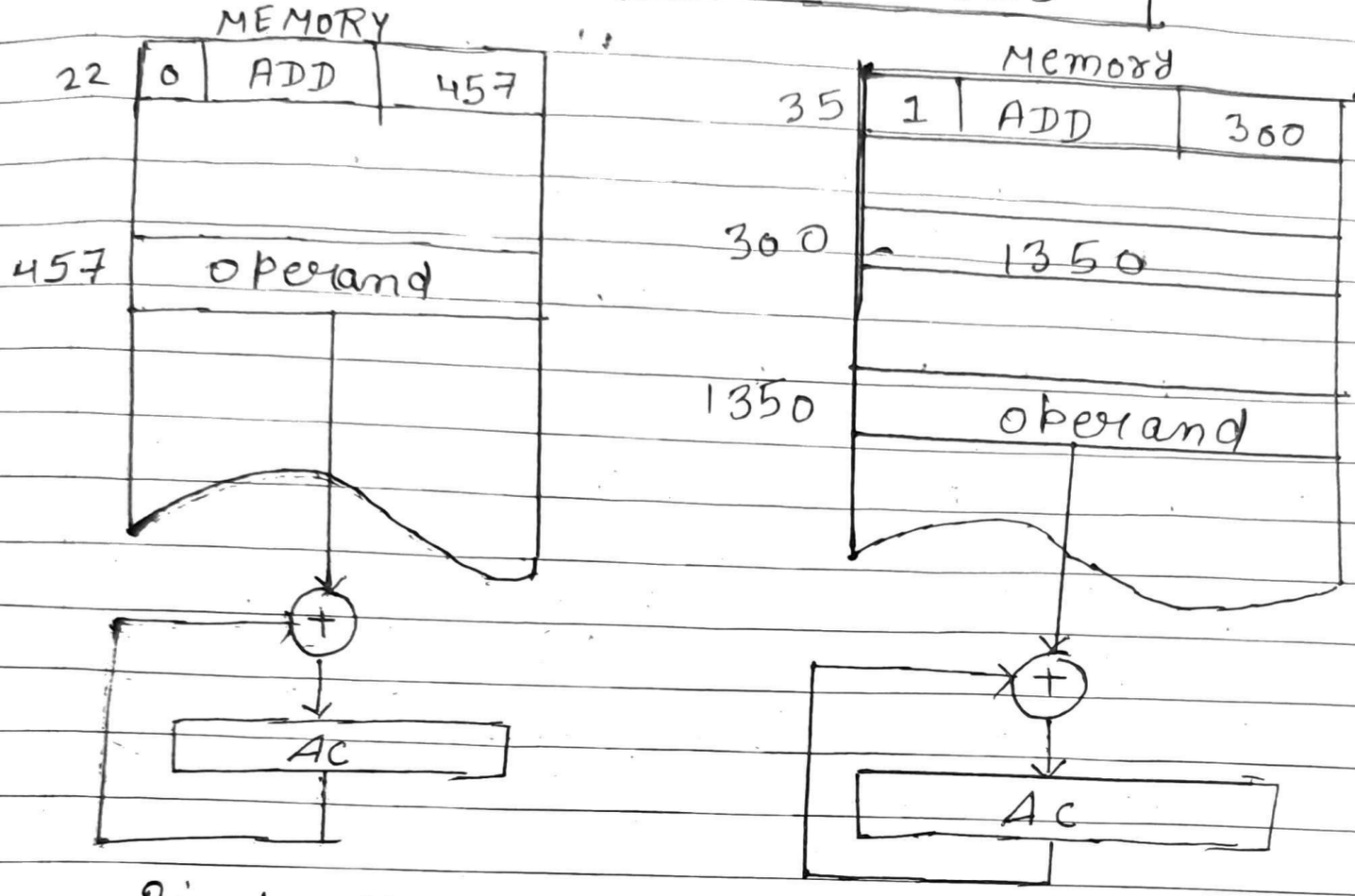
- Sometimes, instead of using bits in the instruction code as an address, it can be useful to use them as the actual data.

immediate operand - if the second part of an instruction contains the actual data, it is called an immidiate operand

Direct address - if the second part of an instruction contains the exact address where the data is stored, it is called direct address.

Indirect address - if the second part of an instruction contains the address of Memory location that stores the address of the data, is called indirect address.





Direct address

Indirect address

- Direct Address instruction-
- This instruction is stored in Memory address 22. Since the "I" bit is 0, it is recognized as a direct address instruction.

- The instruction tells the computer to perform an ADD operation. The address part of the instruction (457 in binary) point to the location here the data (operand) is stored.

The computers find the data at Memory address 457 & adds it to the content of the accumulator (AC).

Indirect address instruction-

This instruction is stored in Memory address 35. The "I" bit is 1, so it is recognized as an indirect address instruction.

The address part is 300 in binary. The computer goes to address 300 to find another address where the data (operand) is located.

The address is stored at location 300 is 1350. So, the computer then goes to address 1350 to get the data.

The data found at address 1350 is then added to the content of the Accumulator.

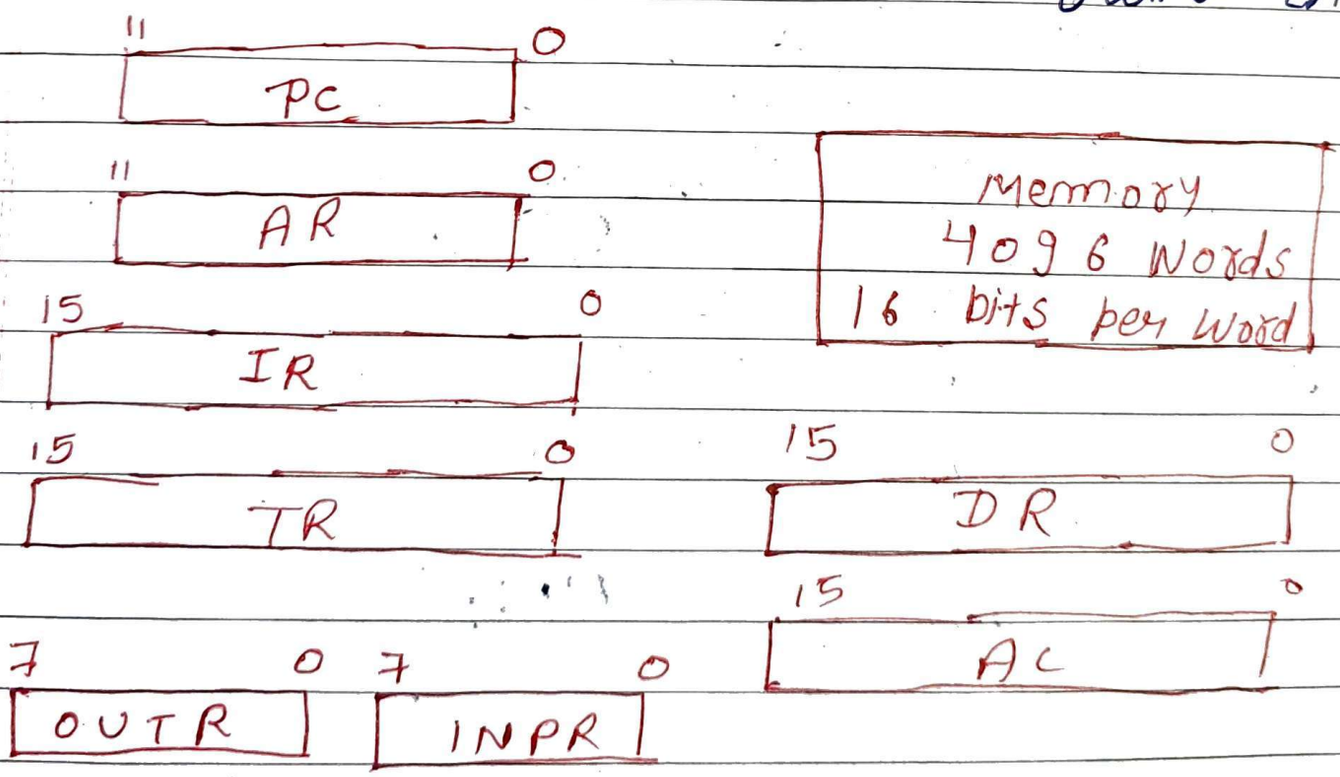
Effective address-

The effective address is the actual location in memory where the data (operand) is found in computation-type instruction or the target in branch-type instructions.

→ for the direct address instruction, E.A = 457

→ for the indirect address instruction, EA = 1350

Register Symbol	No. of bit's	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 charac-



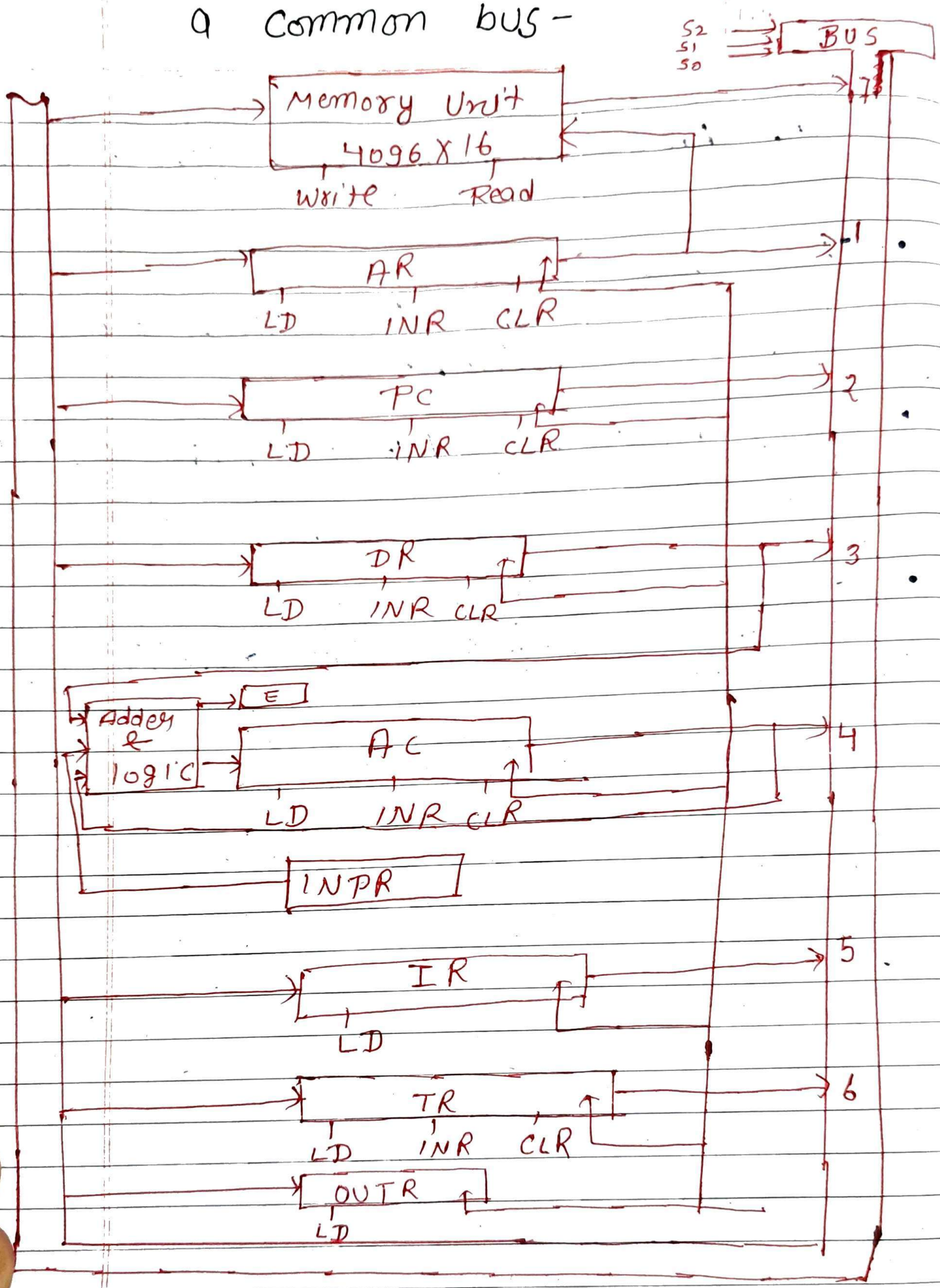
Main Points of Register-

- The data register (DR) holds the operand 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 bit since this is width of a Memory address.
- The program counter (Pc) also has 12 bit's & it holds the address for the next instruction to be read from Memory after the current instruction is executed.
- Two registers are Used for input & 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 basic computer has eight register & one control unit.
- There need to be ways to move data from one register to another & between memory & registers.
- Using a common bus is more efficient way to transfer data in a system with many register.
- Seven registers & Memory are connected to this common bus for transfering information.

Basic Computer Register connected to a common bus -

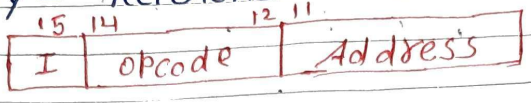


- The numbers next to each output show the decimal values of required binary code.
- for ex → the number next to DR's output is 3. the 16 bit output of DR is placed on the bus when $S_2S_1S_0 = 011$.
- Five registers have three control actions, LD (Load), INR (increment) & CLR (clear)
- When two numbers are added, the result goes to AC, & any overflow (carry-out) is sent to a flip-flop called E (extended-bit).
- An additional input of 8 bit comes from the input register INPR.
- The content of any register can be sent to the bus, the operation like addition can happen in the adder at the same time during one clock cycle.

Computer Instructions -

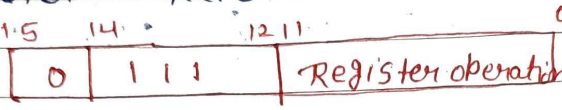
The basic computer has the instruction code formats. & each format has 16 bits.

a) Memory - Reference Instruction -



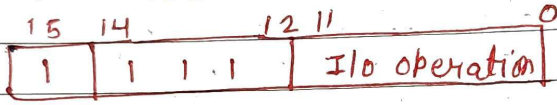
(opcode = 000 - 110)

b) Register - Reference Instruction -



(opcode = 111, I=0)

c) Input - output Instruction -



(opcode = 111, I=1)

☆ Timing & control Unit-

- The function of control unit is to generate relevant timing & control signals to all operation in the computer
- It controls the flow of data between the process & memory & peripherals.
- The control unit must communicate with both arithematic logic Unit (ALU) & main memory.
- The control unit instructs the arithmetic logic unit that which logical or arithematic operation is to be performed.
- Control unit generates control signal using one of the two organizations-

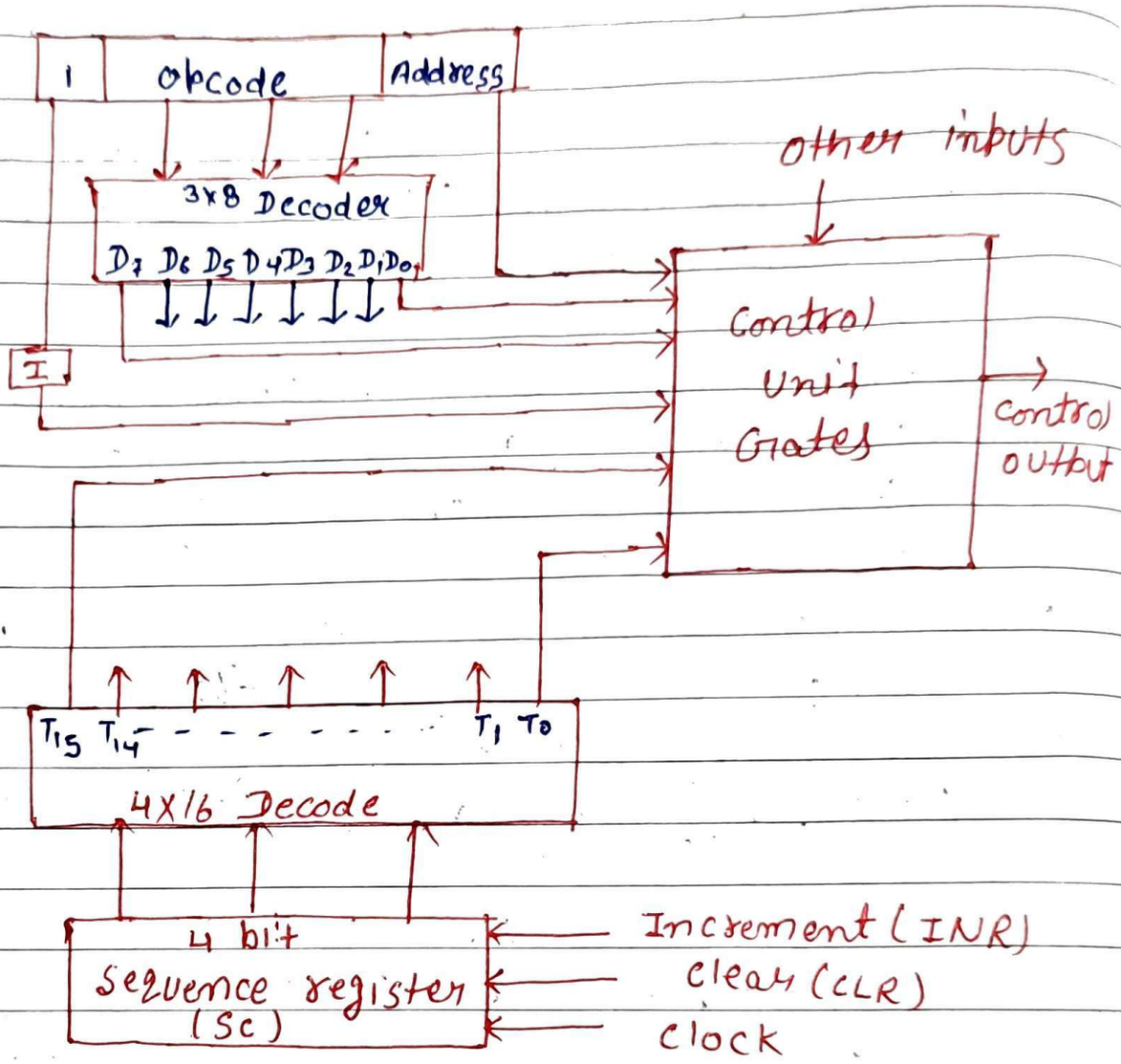
→ Hardwired control unit.
→ Micro-programmed control unit

1. Hardwired control unit -

→ It is implemented as logic circuits (gates, flip-flops, decodes etc.) in the hardware.

→ This technique is very complecated if we have a large control unit.

- In this technique, if the design has to be Modified or changed, require changes in the wiring among various components. the Modification of all combination circuit may be very difficult.



Q. Consider the case where SC is incremented to provide timing signals T0, T1, T2, T3 & T4 in Sequence.
At Timing T4, SC is cleared to 0 (zero) if decoder output D3 is active. This is expressed symbolically by the

Hardwired Control unit-

Consist of a :

- Instruction register
- Number of control logic gates.
- Two - Decoders
- 4-bit Sequence Counter

⇒ An instruction read from Memory is placed in the instruction register (IR).

→ The instruction register is divided into three parts :- the I bit, operation code & address part.

→ First 12 bits (0-11) to specify an address, next 3-bits specify the operation code [opcode] field of the instruction & last most bit specify the addressing Mode I.

I = 0, for direct address
I = 1, for Indirect address

→ First 12 bit's (0-11) are applied to the control logic gates.

→ The operation code bits (12-14) are decode with a 3x8 decoder.

→ The eight output (D0 through D7) from a Decoder goes to the control logic

gates to perform specific operation.

- last bit 15 is transfered to a I flip-flop designated by symbol I.
- The 4 bit sequence counter SC can count in binary from 0 through 15.
- The counter output is decoded in to 16 timing pulses T0 through T15.
- The sequence counter can be incremented by INR input or clear by CLR input.

2. Micro-Programmed Control unit -

→ Micro program consisting of Micro-instructions is stored in the control memory of the control unit.

- execution of a ~~Microprogram~~ Micro-instruction is responsible for generation of a set of control signals.

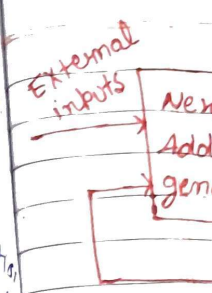
- A Micro ~~ins~~ instruction consist of -
- one or more micro-operation to be executed.
- Address of next micro instruction to be executed.

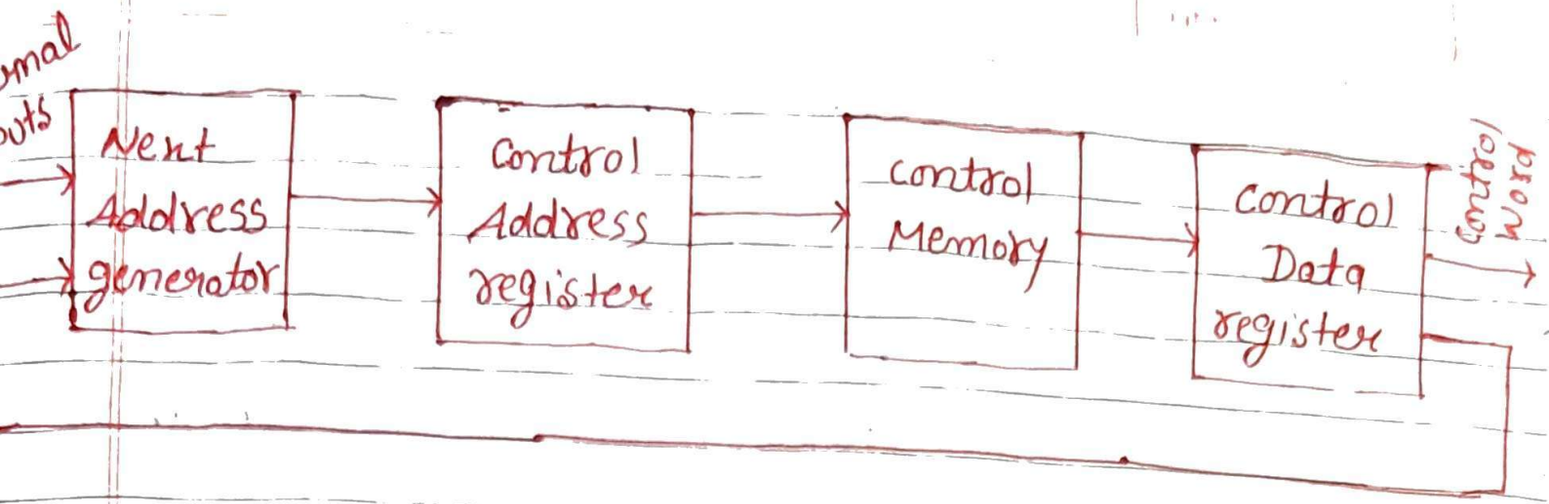
⇒ Micro-programs: Microprogramming is the concept for generating control signals using programs.
These programs are called Micro-program.

⇒ Micro-instruction: The instructions that make micro-program are called Micro-instructions.

⇒ Micro-Code: Micro-program is a group of Micro-instructions. The Micro-program can also be termed as Micro-code.

⇒ Control-Memory: Micro-programs are stored in the read only memory (ROM). That memory is called Control Memory.





- The address of Micro-instruction that is to be executed is stored in the control address register (CAR).
- Micro-instruction corresponding to the address stored in CAR is fetched from control memory & is stored in the control data register (CDR).
- The Micro-instruction contains control word to execute one or more micro-operation.
- After the execution of all micro-operations of micro-instructon, the address of next Micro instruction is located.

Diff. between Hardwired & Micro-Programmed control unit.

	Hardwired	Micro-programmed
Speed	Fast	slow
cost of implementation	More	cheaper
Flexibility	Not flexible, difficult to Modify for new instruction	flexible, new instruction can be easily added
Ability to handle Complex instruction	Difficult	Easier
Decoding	Complex	Easy
Application	RISC Microprocessor	CISC Microprocessor
Instruction set size	SMALL	LARGE
Control Memory	ABSENT	PRESENT
CHIP Area required	LESS	MORE

Instruction cycle-

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

→ In the basic computer, each instruction cycle consist of the following phases-

i. Fetch an instruction from Memory
ii. Decode the instruction
iii. Read the effective address from Memory.
iv. Execute the instruction.

Operation for Fetch & Decode-

Common to all micro-problem operation

T_0 : $AR \leftarrow PC$

T_1 : $IR \leftarrow M[AR]$, $PC \leftarrow PC+1$

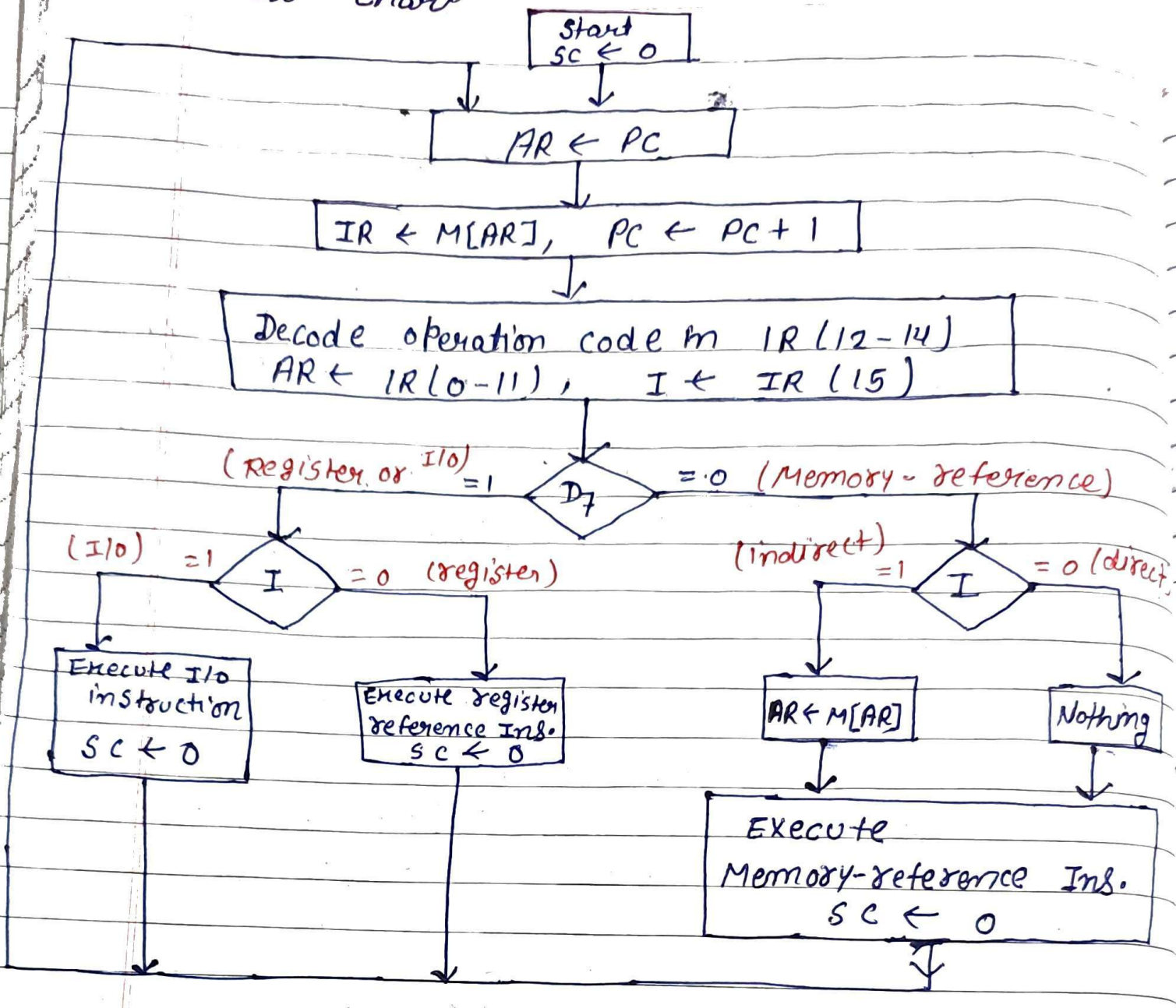
T_2 : $D_0 \dots D_7 \leftarrow$ Decode IR (12-14)

$AR \leftarrow IR$ (0-11)

$I \leftarrow IR$ (15)

- Initially program counter (PC) is loaded with the address of the first instruction in the Program.
- Place the content of PC into AR.
- The instruction is extracted in IR (Instruction register) from AR (Address register)
- last decode the instruction register.

Flow chart -



★. Register - Reference Instruction -

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

- These instructions use bits 0 through 11 of the instruction code to specify one of 12 instruction.
- These 12 bits are available in IR (0-11)

Memory-Reference Instructions-

symbol	operation-decoder	Symbolic description
• AND	D_0	$AC \leftarrow AC \wedge M[AR]$
• ADD	D_1	$AC \leftarrow AC + M[AR]$, $E \leftarrow Cout$
• LDA	D_2	$AC \leftarrow M[AR]$
• STA	D_3	$M[AR] \leftarrow AC$
• BUN	D_4	$PC \leftarrow AR$
• BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR+1$
• ISZ	D_6	$M[AR] \leftarrow M[AR]+1$ if $M[AR]+1 = 0$ then $PC \leftarrow PC+1$

- The decoded output D_i for $i = 0, 1, 2, 3, 4, 5, 6$ comes from the operation decoder & is listed in the table for each instruction.

- The effective address of the instruction is stored in the address register (AR). It's put there during timing signal T2 when $I = 0$ or during timing signal T3 when $I = 1$.

- The execution of memory-reference instructions begins with timing signal T4.

AND to Ac :

- This instruction perform an AND operation between each pair of bits in the accumulator (Ac) & the memory word at the effective address.
- The result of this operation stored in Ac.
- The Micro-operation for this instruction are :

D_0T_4 : $DR \leftarrow M[AR]$

D_0T_5 : $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$

ADD to AC :

→ This instruction adds the memory word at the effective address to the value in AC.

→ The sum is stored in Ac, & the carry-out (C_{out}) is moved to the E (Extended accumlation) flip-flop.

- The micro-operation for this instruction are :-

D_1T_4 : $DR \leftarrow M[AR]$

D_1T_5 : $AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$

★ LDA (LOAD TO AC) -

- This instruction moves the data from a memory location (specified by the effective address) into the AC (Accumulator).

steps :

- T_4 : copy data from memory to DR (Data Register)
- T_5 : Add data from DR to AC, adjust carry, & reset sequence counter (SC).

$D_1 T_4$: $DR \leftarrow M[AR]$

$D_1 T_5$: $AC \leftarrow AC + DR$, $E \leftarrow Cout$, $SC \leftarrow 0$

★ STA (Store AC) -

- This instruction saves the content of the AC into the memory location specified by the effective address.

Execution steps -

→ T_4 : store the content of AC into memory, then reset SC.

$D_3 T_4$: $M[AR] \leftarrow AC$, $SC \leftarrow 0$

☆ BUN (Branch unconditionally) -

- This instruction changes the program flow to a new address specified the effective address.

execution steps :-

T_4 : Transfer the effective address to PC & reset SC.

$D_4 T_4$: $PC \leftarrow AR$, $SC \leftarrow 0$

☆ BSA (Branch & save Return Address) -

- This instruction is ~~called~~ useful for branching to a portion of the program it's called a subroutine or procedure.

What it does -

- saves the address of next instruction (from PC) into the memory at the effective address.
- updates PC to point to the first instruction of the subroutine (EA +1).

steps -

- Save PC into Memory
- increment the effective address & updates PC.

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

Memory - Organization

- Memory Hierarchy -

i) Main Memory - memory unit that communicates directly with CPU (RAM)

ii) Auxiliary Memory - device that provide backup storage (Disk Drives)

iii) Cache Memory - Special very high-speed memory to increase the processing speed (Cache RAM)