3- The 8-bit register AR, BR, CR and DR initially have the following values-

→ AR = 10010011 BR = 10000111
CR = 11001101 DR = 11110011

① AR ← AR ∨ BR

```
  10010011
  10000111
AR ← 10010111
```

BR ← $\overline{BR}$
BR ← 01111000

CR ← CR - 1

```
  11001101
+ 11111111
  11001100
```

DR ← AR + $\overline{BR}$ + 1

```
  01111000
BR • 10000111
     +1
  10001000
DR ← 10001000
  10010011
 (1)00011011
DR ← 00011011
```

BR ← cir BR

```
11110000
```

Addressing Mode-

LDA → Load/Copy data Address.
mov A,B → Move

(Part of Microprocessor)

MVI A, 32 H
MOV A, B
LDA 2004
LDA a B

- The manner or ways in which the operand (data) is given in the instruction.

There are 11 addressing Mode -

1 → Immediate Addressing Mode → If the data is given in the instruction itself is called immediate addressing mode.

Eg - MVI A, 32 H

Q. What do you mean by addressing mode & Explain each addressing mode

2- Register Addressing Mode - If the operant is given in the instruction is called Register Addressing mode.

Eg → mov A, B

3- Direct Addressing Mode - In the direct addressing mode, if the address of the operant is given in the instruction is called direct address.

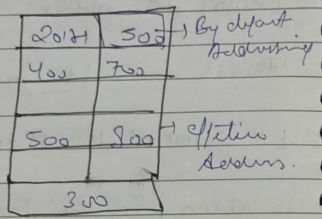
Eg → LDA 400.

A
B	=
0	8
1	9

4- Indirect Addressing Mode-

Address field-

In Indirect Addressing mode address field point outs effective address, this effective address have the operand. The data will store into Register.



5→ Implied Addressing Mode-

Implied Addressing Mode means the operand is implied or implicit that means the processor is operating with one register only and operand is in that register only.

6→ Register Indirect Addressing Mode-

In the register Indirect Addressing Mode the address of the operand is given in the register pair, this register pair address pointing out operand of memory location.

And this operand will store in accumulator.

(7) Auto Increment Addressing Mode-

It is same as register Indirect Addressing mode, in this Addressing mode, after the execution the instruction the address of the operand will be increment by 1.

(clock cycle -

(8) → Auto Decrement Addressing Mode-

(first decrement then execute)

It is (a type) or same as register indirect addressing mode in auto decrement addressing mode before the execution of the instruction, address in register will be decrement by 1.

~~Address~~

~~(8) Index Addressing~~

(9) Relative Addressing Mode-

$EA = PC + IR$ (Address field)

Effective address

PC
2000 H
(Programme counter)

* In relative addressing mode effective address will be equal to programme counter address plus address field.

* PC → Provides the address of the next instruction.

⑩ Index Addressing :-

EA = XR + IR
↓ Instruction
Index register
register (Address field)

➡ Index Addressing

E

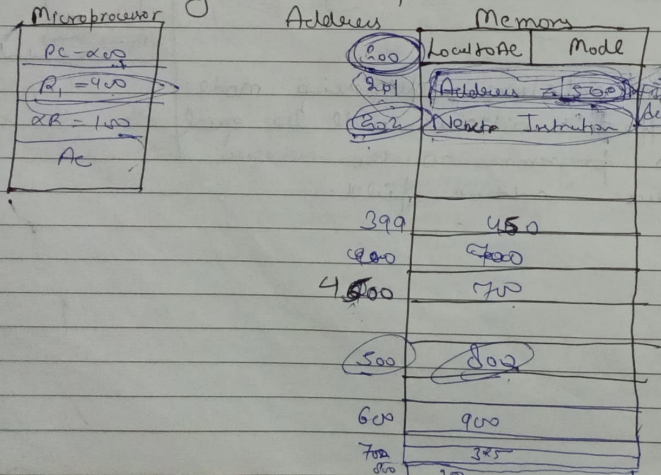
It means effective address can we calculate combinly through index register and address field. provides us new address that is effective address.

⑪ Base register addressing mode :-

It means effeline address can we calculate combinly through base register address and address field provides us new address called effeline address.

EA = BAR + IR

Addressing Mode Example-



Q- Addressing Modes	Effective address	Content of Accumulator
Direct Addressing	500 (A)	800 H
Immediate Addressing	201 H	500 H
Indirect Addressing	800 H	300 H
Relative	702 H	325 H
Indore	600 H	900 H
Register	X Micro (R1)	400 H
Rejistr Indirect	400 H	700 H
Register Auto Incre	400 H	700 H
Auto Decrement	399 H	450 H

→ By Default
→ Data is in Instruction
→
Base Register | 600 H | 900 H
Implied Arm

If BAR is not given then XR is BAR

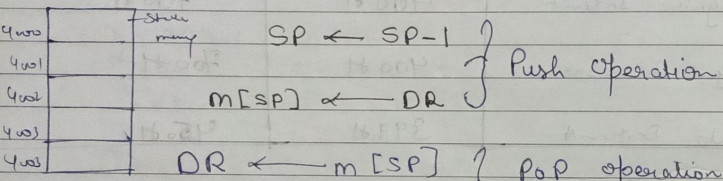
Implied cmA —

Stack organisation – (Reverse polish notation) –

LIFO → (Last In First Out)

- Stack works in LIFO manner, this type of memory are structured memory i.e Stack is a part of memory in which through one address we can acess whole memory (Stack)

→ Stack Pointer (SP) point out the top of Stack that means SP contains the top of stack address



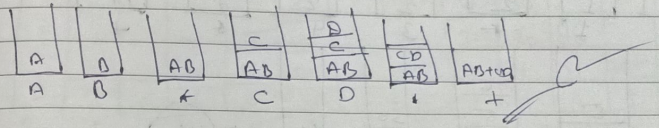
→ Reverse polish notation →

A+B → Infix
+AB → Prefix or Polish Notation.
AB+ → Postfix or Reverse Polish notation

- Infix form of eq^n we use in our daily life when the microprocessor operates the infix first it convert into Postfix or Reverse Polish Notation because the Reverse Polish for of eq^n can be solved with the help of Stack.

Here, Microprocessor operates any eq^n with the help of stack.

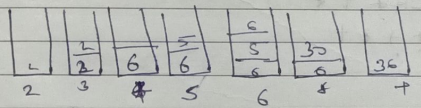
Q→ Convert infix eq^n into Postfix.
(A*B) + (C*D).
→ AB* + CD*
→ AB* CD* +



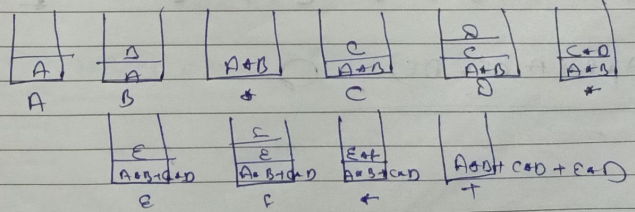
Q (2*3) + (5*6) using stack & Reverse notation.

2 3 * + 5 * 6 *
2 3 * 5 6 * +

6 + 30 = 36



Q A * B + C * D + E * F
AB* + CD* + E * F +
AB* CD* + E * F +
AB * CD * + EF * +

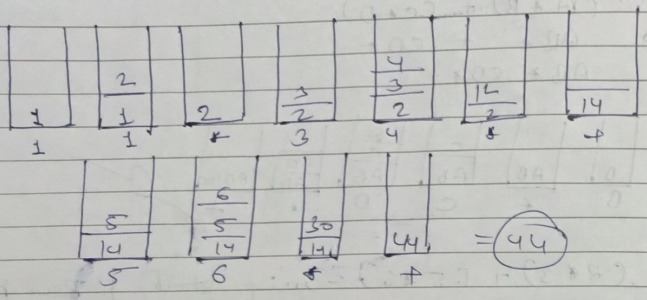


2 + 10 + 30 = 44

1 * 2 + 3 * 4 + 5 * 6

12 * + 34 * + 56 *

12 * 34 * + 56 * +

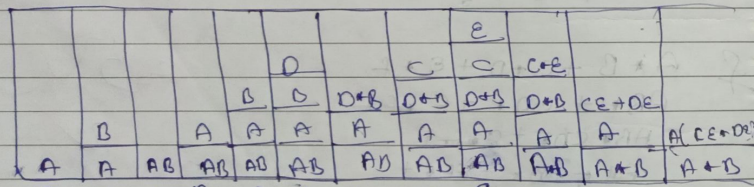


Q. A * B + A * (B * D + C * E)

A * B + A * BD * CE * +

A * B + ABD * CE * + *

AB * ABD * CE * + * +



A * B + A(CE + DE)

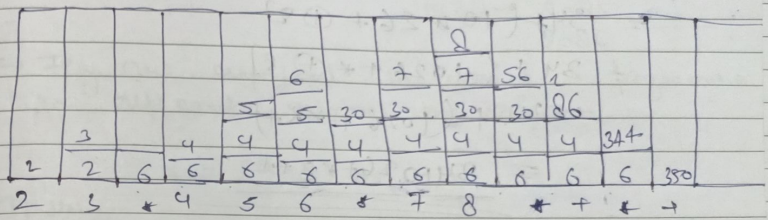
Q = 2 * 3 + 4 * (5 * 6 + 7 * 8)

2 * 3 + 4 * (56 + 78 *)

23 * + 4 * (56 + 78 * +)

23 * + 456 * 78 * + * +

23 * 456 * 78 * + * + *



→ Micro processor solve the eq using stack

⇒ A(B+C * (D+E)) / F * (G+H)

~~A(B + C)~~

A(B+C * DE+) / F * (GH+) ⇒ A(B+CDE+*) / F * (GH+*)

A(BCDE+*+) / FGH+* → ABCDE+*+* / FGH+*

ABCDE+*+*FGH+*/

→ Ex 1 →

→ (3+4)[10*(2+6)+8]

= 34+ [10* (26+)+8]

= 34+[10*26+ ⊕ 8]

= 34 + (1026+* ⊕ 8]

= 34 + (1026+*8+)

= 341026+*8++

Left one is destination syntax.

Note → If there is [ ] bracket means address is given in the question.
If there is no square bracket then means data is given.

→ 3 Address, 2 Address, 1 Address and 0 address Instruction -

Intel →
1974 → 4004
1978 → 8085 - Pentium
8086

→ They are required to write a programme in different manner or ways -

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

By using three address instruction mode -

ADD R_1, A, B	$R_1 \leftarrow m[A] + m[B]$
ADD R_2, C, D	$R_1 \leftarrow m[C] + m[D]$
~~MUL X, R1, R2~~	~~m[X] ← m[R1] * m[R2]~~
MUL X, R_1, R_2	$m[X] \leftarrow R_1 * R_2$

Advantage
- The three address instruction have less no of instruction.

Disadvantage
- The size of the instruction is large.

Two address - $X = (A+B) * (C+D)$

MOV R_1, A	$R_1 \leftarrow m[A]$
ADD R_1, B	$R_1 \leftarrow m[B] + R_1$
MOV R_2, C	$R_2 \leftarrow m[C]$
ADD R_2, D	$R_2 \leftarrow R_2 + m[D]$
MOV X, R_1	$X \leftarrow [R_1]$
MUL X, R_2	$R_1 \leftarrow R_1 * m[R_2]$

MUL R_1, R_2 → $R_1 \leftarrow [R_1] * R_2$

MOV a, R_1 → $[a] \leftarrow R_1$

- **One Address** – (Implied Addressing part) → P

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

$AC \leftarrow m[A]$

(Accumulator)

LOAD A	$AC \leftarrow m[A]$
ADD B	$AC \leftarrow AC + m[B]$
LOAD C	$AC \leftarrow m[C]$
ADD D	$AC \leftarrow AC + m[D]$
MUL T	$AC \leftarrow AC * m[T]$
MOV X	$m[X] \leftarrow AC$

- **Zero Address** – $X = (A+B) * (C+D)$

(Operands on stack base)

TOS → Top of Stack

PUSH A	TOS ← A
PUSH B	TOS ← B
ADD	TOS ← A+B
PUSH C	TOS ← C
PUSH D	TOS ← D
ADD	TOS ← C+D
MUL	TOS ← (A+B) * (C+D)
POP X	M[X] ← TOS

UP

PC = 200
R_1 = 400
XR = 100
AC

Address Limit

Memory

200	Load to AC	Mode
201	Address = 500	
202	Next Instruction	
399	450	
400	700	
500	800 ← EA	
600	900	
702	325	
800	300	

Addressing Mode	Effective Address	Content AC (value)	
Direct	500 H	800 H	(address + memory location given) (By default address)
Immediate	201 H	500 H	(In Instruction)
Indirect	800 H	300 H	It point effective address. (Address Limit)
Relative	702 H	325 H	
Index	600 H	900 H	
Register	——	400 H	→ given in register.
Register Indirect	400 H	700 H	→ through this register
Auto Increment	400 H	700 H	→ Register Indirect – after
Auto Decrement	399 H	450 H	→ Register Indirect – before

Relative → EA = PC + (IR) Address field. (Instruct Register)
(Instruction)

Index → EA = XR + IR (100 + 500)

$X = A \times B + A \times (B \times D + C \times D)$

3 Address —

Mul R_1, B, D	$R_1 \leftarrow m[B] \times m[D]$
Mul R_2, C, D	$R_2 \leftarrow m[C] \times m[D]$
ADD R_1, R_1, R_2	$R_1 \leftarrow R_1 + R_2$
MUL R_3, R_1, A	$R_3 \leftarrow R_1 * m[A]$
MUL R_4, A, B	$R_4 \leftarrow m[A] \times m[B]$
Add X, R_3, R_4	$m[X] \leftarrow R_3 + R_4$

0 Address —

AB× (+) A× (BD×CD×+)
AB× (+) ABD×CD×+×
AB×ABD×CD×+×+

PUSH A	Tos ↔ A
PUSH B	Tos ↔ B
MUL	Tos ↔ A×B
PUSH A	Tos ↔ A
PUSH B	Tos ↔ B
PUSH D	Tos ↔ D
MUL	Tos ↔ B×D
PUSH C	Tos ↔ C
PUSH D	Tos ↔ D
MUL	Tos ↔ C×D
ADD	Tos ↔ (B×D)+(C×D)
MUL	Tos ↔ A×((B×D)+(C×D))
Add	Tos ↔ (A×B)+A×((B×D)+(C×D))
POP	m[X] ↔ Tos

1 Address → $A \times B + A \times (B \times D + C \times D)$

LOAD A	ACC ← m[A]
MUL LOAD B	ACC ← ACC × m[B]
MOV T	m[T] ← ACC
LOAD B	AC ← m[C]
MUL D	AC ← AC × m[A]
MOV U	m[U] ← AC
LOAD C	AC ← m[C]
MUL D	AC ← AC × m[D]
ADD U	ACC ← AC + m[U]
MOV V	m[V] ← ACC
LOAD A	

- An instruction is stored at location 300 with its Address field at location 301. The Address field has value 400. The processor Register contains 200. Evaluate effective —

Direct → 400 H
Immediate → 301 H
Relative → PC+R → 502
Register Indirect → [200]
Index → 200 + 400 = 600

followed by the above instruction 300
Const. 301
What must the address field 301 of an Indexed addressing mode instruction be to make it the same register indirect mode instruction when the address field value is 0.

Diff b/w arithmetic sim & logical sim —

Design full adder using Decoder — Decoder 3×8

Design 16×1 MUX using NAND Gate.

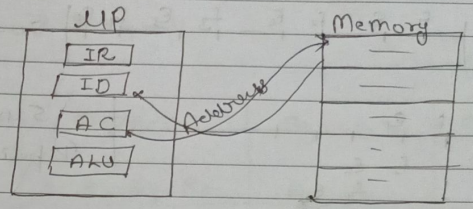
S	S_0	S_1	S_2	S_3	Y
1	0	0	0	0	I_0
1	0	0	0	1	I_1
1	0	0	1	0	I_2
1	0	0	1	1	I_3
1	0	1	0	0	I_4
1	0	1	0	1	I_5
1	0	1	1	0	I_6
1	0	1	1	1	I_7
1	1	0	0	0	I_8
1	1	0	0	1	I_9
1	1	0	1	0	I_{10}
1	1	0	1	1	I_{11}
1	1	1	0	0	I_{12}
1	1	1	0	1	I_{13}
1	1	1	1	0	I_{14}
1	1	1	1	1	I_{15}

$Y = S(\bar{S}_0\bar{S}_1\bar{S}_2\bar{S}_3 I_0 + S_0\bar{S}_1 S_2 S_3 I_1 + \bar{S}_0 S_1 \bar{S}_2 \bar{S}_3 I_2 + \bar{S}_0\bar{S}_1 S_2 S_3 I_3 +$

Enable

Select lines

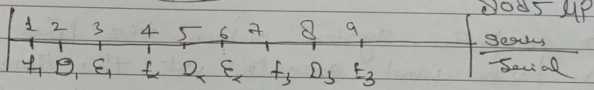
Pipelining –

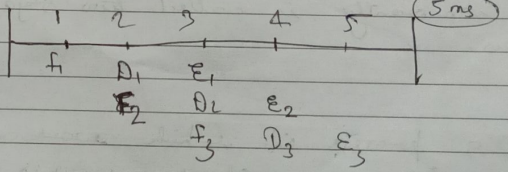


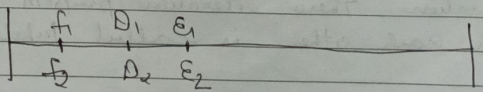
Steps –

Fetching (Buses)
Decoding (Decoder)
Execution (ALU)

9ms Time







M T W T F S S

Page No.:

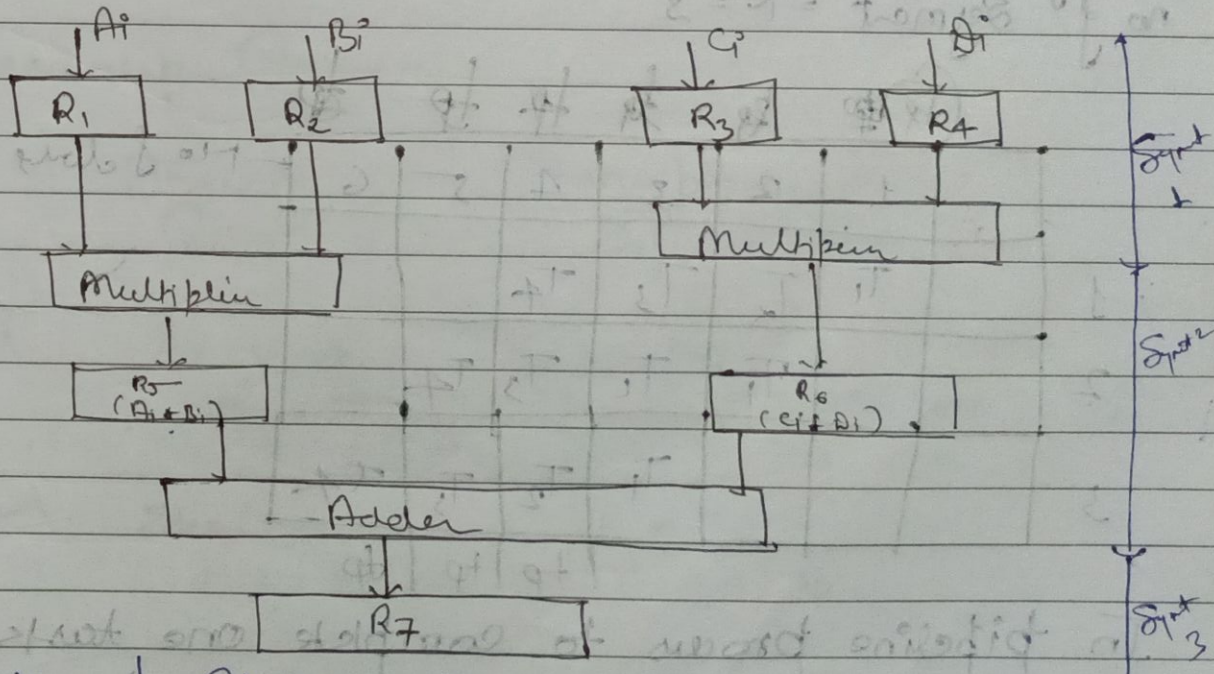
Date:

Q $Y = (A_i * B_i) + (C_i * D_i)$

$\Rightarrow i = (1, 2, 3 \ldots \ldots \ldots 8)$ = 8 iteration

$\Rightarrow$ 3 segments

No of clocks	Segments 1	Segments 2	Segment 3
	R_1 R_2 R_3 R_4	R_5 R_6	R_7
1	A_i B_i C_i D_i	—	—
2		$A_i * B_i$ $C_i * D_i$	—
3			$A_1 * B_1 + C_1 * D_1$
	A_8 D_8 C_8 D_8		



Serial: 3 × 8 = 24 clocks

Pipelining: 10 clocks

- K represents total no of Segments, n represents total no of iteration

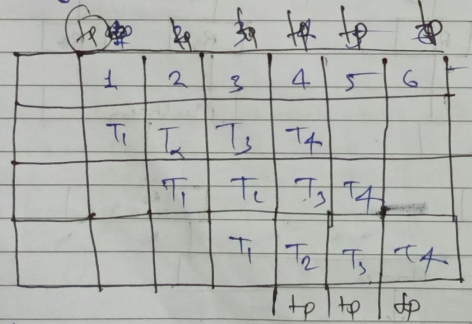
In seq = $K \times n$

In pipeline = $K + n - 1$

$$\text{Speedup} = \frac{K \cdot tn}{(K+n-1) \cdot tp}$$

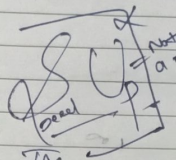
Space time Diagram -

No. of task = n = 4
no of Segment = K = 3



In pipeline process to complete one task K * tp time will consumed. where K are the no of Segments and n are the no of task.

- In pipeline process remaining n-1 task will perform in $(n-1) * tp$ time
- The total time will consumed in pipeline process will be



$= K * tp + (n-1)tp$
$= Ktp + (n-1)tp$
$= (K + n - 1)\, tp$

This much time consumed by whole pipeline process

In non-pipeline process two complete n task $n * tn$ time will consumed, when the task is not segmented.

- $tn > tp$ (tn time is always greater than tp)

Speed-up Ratio -

$$\text{Speed-up Ratio} = \frac{\text{Non-pipeline time}}{\text{Pipeline time}}$$

non segmented

$$\text{Speed-up Ratio} = \frac{n \cdot tn}{(K+n-1)\, tp}$$

Segmented

$$\text{Speed-up Ratio} = \frac{K \cdot n \cdot tn}{(K+n-1)\, tp}$$

Q. A non-pipeline system takes 50 nsec to process a task. The same task can be processed in a 6 segment pipeline with a clock cycle of 10 nsec. Determine the speed up ratio for n = 100

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

Speed up ratio $= \frac{50 \times 100}{(6+100-1) \times 10} = \frac{5000}{1050} = \frac{100}{21} = 4.76$

Efficiency $= \frac{100}{21} = 4.76$

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

$K = 6$
$n = 8$

No of clock = 9

No of clock	1	2	3	4	5	6	7	8	9	10	11	12	13
	T_1	T_1	T_1	T_1	T_1	T_1	T_2	T_8					
		T_2	T_2	T_2	T_2	T_2	T_3	T_8					
			T_3	T_3	T_3	T_3	T_4	T_8					
				T_4	T_4	T_4	T_5	T_8					
					T_5	T_5	T_6	T_8					
						T_6	T_7	T_8					

1 o/p

n+(k-1) clock

Pipeline -

No of clock = (k+n-1)

t_p = 1 nsec (assume)

(14-1)

13

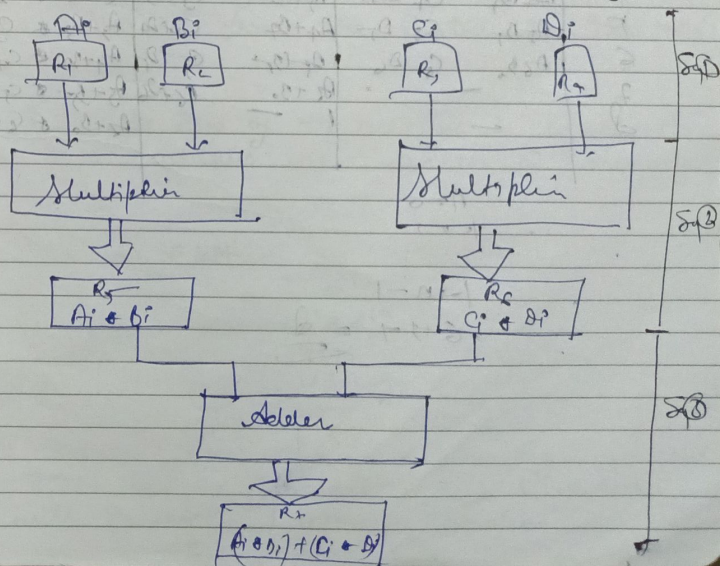
total time = 14 × 10

= 130 nsec.

Q- In certain, it is necessary to perform the arithmetic operation.

$(A_i * B_i) + (C_i * D_i)$

Specify the pipeline configration to carry out this task.



$(A_1 + B_2) * (C_1 + D_1)$

⇒ The no of segment will be decide by the data will be store into registers. If we store data three times into register that means three segment will be used.

⇒ List the contain of all the register in the pipeline for iteration i = 1 ——— 6

= No of data	Segment 1		Segment 2		Segment 3
	A_1 B_1	C_1 D_1			
	R_1 R_2	R_3 R_4	R_5	R_6	R_7
1	A_1 B_1	C_1 D_1	~~$R_5 = B_1 * C_1 + D_1$~~		—
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	$A_5 + B_5$	$C_5 + D_5$	$A_4 + B_4$ * $C_4 + D_4$
7	—		$A_6 + B_6$	$C_6 + D_6$	$A_5 + B_5$ * $C_5 + D_5$
8	—		—		$A_6 + B_6$ * $C_6 + D_6$

A = 6

1 in - 1

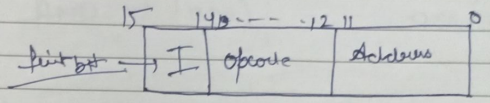
6 + 1 - 1 = 8

Computer Instruction (Memory Reference Instruction)

1st format :

Memory Reference format -

- It follow 16 bit instruction -



- If 'I' = 0 (Direct Addressing Mode)
  If I = 1 (Indirect " ")

- Opcode - "Operation code" (unique), it represent by three binary pattern (bits) these binary bits are used to store the instruction into the memory

- Here the size of the address is 12 bit, the 12 bit address can access 2^{12} memory

~~for~~ 2 bit -

2^{12}

$2^2 * 2^{10}$

4096

4 K Ans

To access Four thousand memory, we need 4000 unique address, where 1 address will point out 1 memory location.

1 Hexadecimal represents 4 binary

X → Don't care

12 bit address will represent by 3 don't care value in hexadecimal

- Memory Reference Instruction - Direct = 0 = I
  Indirect = 1 = I
- First Instruction - (AND)

Symbol	Hexadecimal	
	I = 0	I = 1
AND → (000)	0 x x x	8 x x x

- Accumulator contain logically and with the operand of given memory location and the result will be store in accumulator. The address in the instruction can be any value so it represent by three don't cares.
- ADD → (001)

	1 x x x	9 x x x

- In this instructor accumulator of operand ADD with the operand of given memory location and the result will store in accumulator.
- LDA (load in acc) (010)

	2 x x x	A x x x

- This instruction we means Operand of given memory location copy into accumulator.

- STA (Store in acc.)

"	"
3××××	B××××

- Operand of accumulator will be copy into memory location, its address will be given in the instruction.

- BUN (Branch unconditional)

"	"
4××××	C××××

- Its mean this instruction change the Squence of the programme through address will be given in the instruction.

(to escape the instruction)

- BSA (Branch Same Address)

"	"
5××××	D××××

Return

Main prog ko chor Subroutine programe me jana.

- When the programmer shift from main programme to Subroutine programm after execution of the Subroutine the programme shift from Subroutine to routine (main) programme. For returning programme need the return address, so the return address should be save at the safe place.

8bit = 1 byte.
4bit = Nibble.

ISZ →

	Direct	Indirect
(Increment the value skip in accumulator)	"	"
	6××××	E××××

(Increment by 1 in accumulator value)

Flag → status of current situation (result).

In basic process there are 5 flags —

SF	ZF	ACF	PF	CF

Sign flag

SF = 1 → -ve

SF = 0 → +ve

→ Sign flag — After arithmatic, if the result is -ve sign flag will be 1. if the result is +ve sign flag will be 0, after any arithmatic and logical operation.

→ Zero flag (ZF) → In zero flag if the result is zero zero flag will be 1, if the result is non-zero so the zero flag will be 0.

→ Auxillary carry flag (ACF) :- If the carry shift from 4bit to 5bit or 1 nibble to 2 nibble is called Auxillary carry flag. So the AUF will be 1. otherwise it is zero.

result is even 1

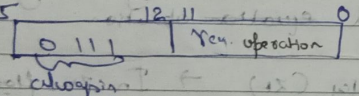
- Parity flag -
  If the result have even no of ones so the parity flag will be 1. Otherwise it will be zero.

- Carry flag -
  After the arithmatic operation if the carry generate ~~is called~~. it will be store in carry flag, so the carry flag will be 1. If the carry not generate so the carry flag will be zero.

ISZ - Increment in the value of accumulator & skip increment if zero flag is one.

Types of Instruction -

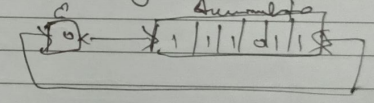
- Register reference Instruction :-



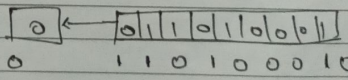
- 7 opd → Register reference Instruction.
- 0 to 6 opd → Memory ref. with direct
- 8 to E opd → " " " with Indirect.

- CLA → 7800
  (Clear operation)
  → Clear the value of accumulator (reset the accumulator)
- CLE → 7400
  → Clear the value of Extra flag (carry)
- CMA → 7200
  Complement the value of accumulator
- CME → 7100
  Complicit value of E-flag. (Carry flag)
- CIR → 7080
  (Circular right b/w accumulator & E-flag)

Ex-



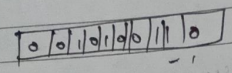
- CIL → 7040
  (Circular left b/w accumulator & E-flag).



INR
(Increment) in the accumulator

- INC → 7020

Increment the value in accumulator by 1.



0010100111

- SPA → 7010 (Skip the next instruction if the data of accumulator is +ve)

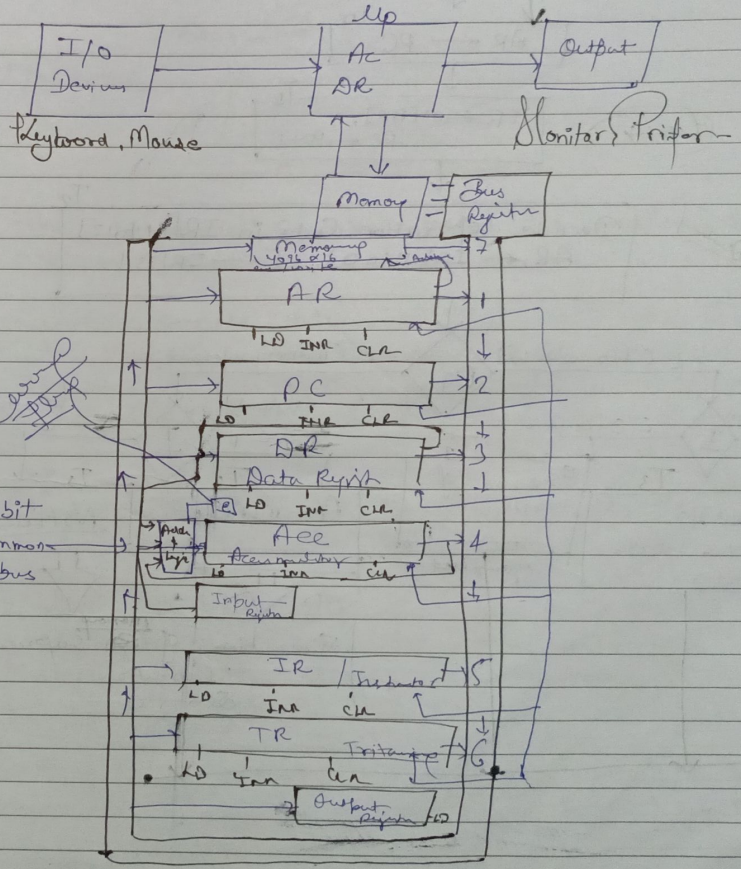
(Decision making instruction)

- SMA → 7008

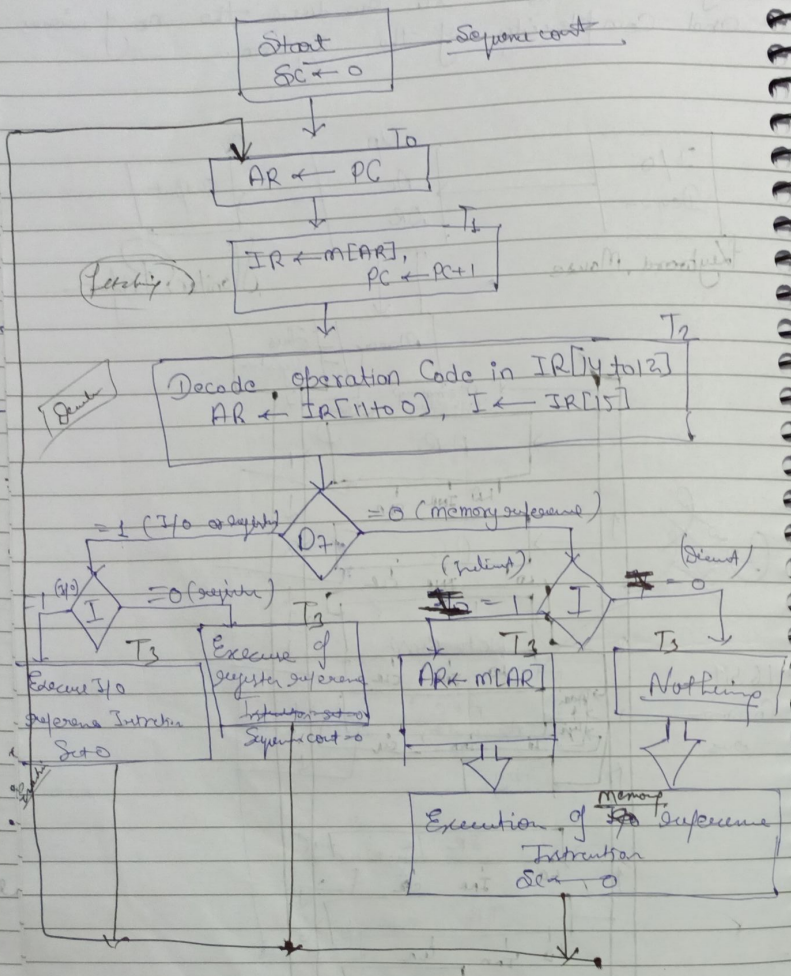
Skip the next instruction if the data of accumulator is -ve)

Common Bus System -

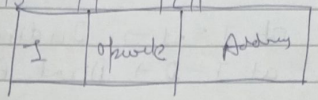
The common bus system is used to reduce the no of wires and complexity of the system.



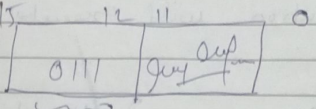
Flow chart of Instruction Cycle:



Memory ref

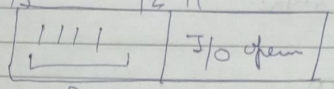


Register ref



Start with

I/O reference



- The programme counter held the address to address register.
- Instruction register receives the instruction from the memory location pointed by address register. Side by side PC will be increment by 1. (PC ← PC+1) (Nothing)
- Instruction register sent the instruction to decoder decoder decode the instruction.
- D7 bit identifies the category of instruction if D7 = 1 then it will be I/O or register reference register if D7 = 0 it will be memory reference register.
- Further I identifies the category or I/O or register if I = 1 → I/O, I = 0 → register.

- if D_7 is 0 at I=01 → Indirect Addressing
  if D_7 is 0 at I=0 → Direct Addressing memory ref.

After the instruction will execute & Sequence cnt become 0.

It means it represent the processor is ready to execute the next instruction by following the same steps.

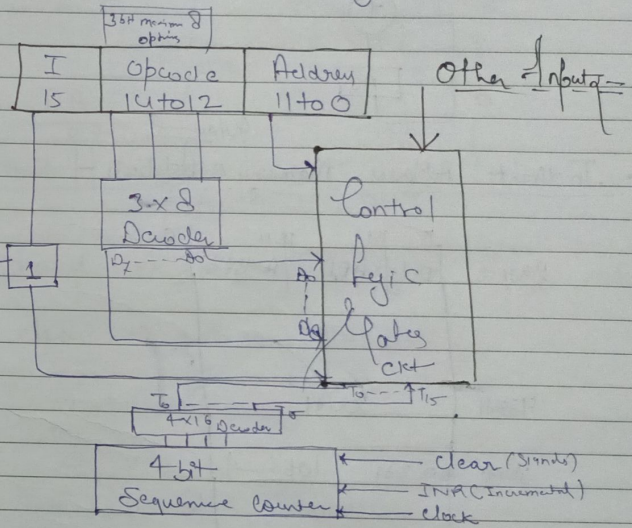
Memory reference

opCode	Command	Direct	Indirect
D_0	AND	0xxx	8xxx
D_1	ADD	1xxx	9xxx
D_2	LDA	2xxx	10xxx
D_3	STA	3xxx	11xxx
D_4	BUN	4xxx	12xxx
D_5	BSA	5xxx	13xxx
D_6	ISZ	6xxx	14xxx
D_7		7xxx	15xxx

Ref d/w Common Bus & Instruction cycle

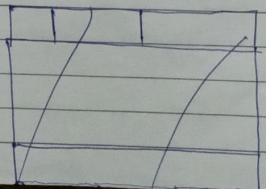
- Control unit (Time can control)

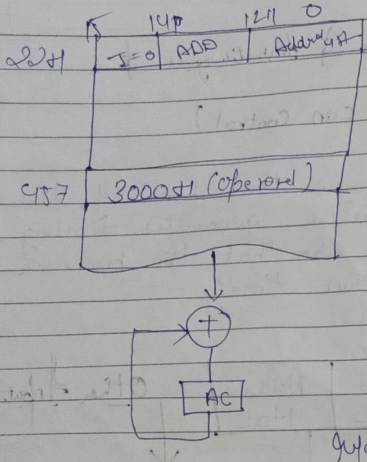
The timing & CU is the brain of the Computer System, which generates timing and control signals, so that the processor operates the instruction timely.



Computer register

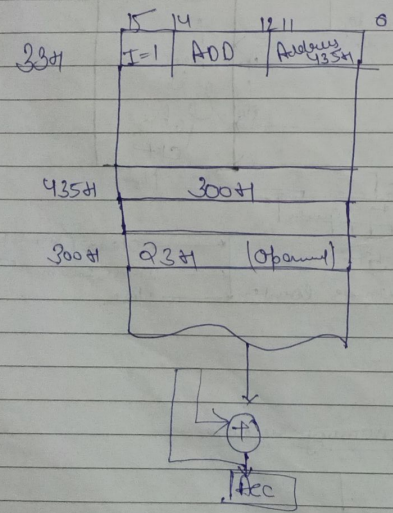
- Direct Addressing memory reference -



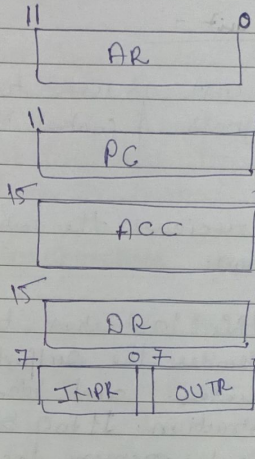


Operand

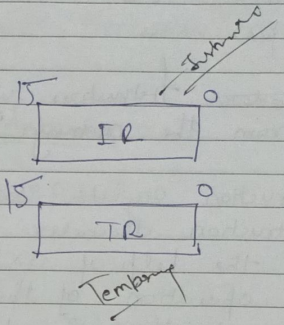
- Indirect Address Memory ^ Address -



- Register and memory size -



Memory
4096 x 16
4096 words



LD → Load → To enable register to get data.

CLR → clear → To reset the data.

INR → Incremental → by 1 bit

16 bit = 1 word
8 bit = 1 byte
4 bit = 1 nibble

Micro operations of memory reference instruction

• Working of timing of Control unit -

The timing and Control unit is used to provide the timing signals & control signals to processors.

Decoder Instruction register recieve the instruction from the memory location.

Instruction register send this instruction to instruction decoder, it decodes the output with the help of 3×8 Decoder and identifies the operation of the instruction. 11 to 0 bits point out the source operand memory location.

The four bits Sequence counter have three inputs

→ clock is used to triggred the input of

→ ~~clear~~ it decide which micro operation will perform in which clock. it also used to synchronize all the states of the processor.

→ Increment - It is used to increment the count value of the clock or counter

When the instruction is performed so the Sequence count will be zero and Sequence counter will be reset or clear by clear signal.

Sequence counter signal decodes through 4×16 decoder and sign to control logic ckt.

Control logic ckt recieves the output of sequence count also the decoded opcode of the instruction.

After that Control logic circuit sent the information to ALU or Registers.

• Micro operations of the instruction -

→ Case-I- AND to ACCUMULATOR -

- Sequence counter is zero.
- T_0 : $AR \leftarrow PC$
- T_1 : $IR \leftarrow M[AR]$, $PC \leftarrow PC+1$
- T_2 : Decode operation code [14 to 12.]
  $AR \leftarrow IR[11 \text{ to } 0]$, $I \leftarrow IR[15]$
- T_3 : Nothing. / Index T_3 : $AR \leftarrow M[AR]$

This means this is to and operation

- T_4 : $DR \leftarrow M[AR]$
- T_5 : $AC \leftarrow AC \wedge DR$

• Perform AND ~~operation~~ microoperation.

or

Perform AND to Acc microop.

$D_7 \dots D_0 \leftarrow IR[14\text{to}12]$

- Perform ADD to Ac. ADD 1xxx

T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC+1$
T_2: Decode operation Code IR[14to12], $AR \leftarrow IR[11to0]$
$I \leftarrow IR[15]$ memory.

T_3: Nothing

$D_7 I_4$: $D_0 \leftarrow m[AR]$
$D_7 I_5$: $AC \leftarrow AC + DR$.

- Perform LDA to Ac.

T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC+1$
T_2: Decode operation Code $\leftarrow IR[14\text{to}12]$, $AR \leftarrow IR[11to0]$
$I \leftarrow IR[15]$ memory.

T_3: Nothing / $AR \leftarrow m[AR]$

$D_2 T_4$: $DR \leftarrow M[AR]$
$D_2 T_5$: $AC \leftarrow DR$, $SC \leftarrow 0$

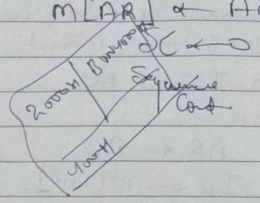
- Perform STA to Acc

T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC+1$
T_2: Decode operation Code IR[14to12], $AR \leftarrow IR[11to0]$
$I \leftarrow IR[15]$ memory.
T_3: Nothing / $AR \leftarrow m[AR]$.

PC is overwritten for type 1 instruction.

$D_3 T_4$: $DR \leftarrow M[AR]$
$D_3 T_5$: $AC \leftarrow m[DR]$

$M[AR] \leftarrow AC$
$SC \leftarrow 0$



- Perform BUN to Acc.

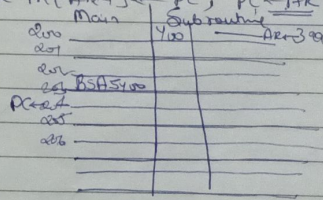
T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC+1$
T_2: Decode operation Code IR[14to12], $AR \leftarrow IR[11to0]$
$I \leftarrow IR[15]$ memory.

T_3: Nothing / $AR \leftarrow m[AR]$
$D_1 T_4$: $PC \leftarrow AR$ $SC \leftarrow 0$

- Perform BSA to Acc.

T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC+1$
T_2: Decode operation Code IR[14to12],
$AR \leftarrow IR[11to0]$, $I \leftarrow IR[15]$ memory.

T_3: Nothing / $AR \leftarrow m[AR]$
$D_5 T_4$: $M[AR] \leftarrow PC$, $PC \leftarrow AR+1$
OR $M[AR+1] \leftarrow PC$; $PC \leftarrow AR$.
$D_5 T_5$:



Interrupt cycle

$D_5 T_4$: $M[SP] \leftarrow PC$
$D_5 T_5$: $PC \leftarrow AR$

Increment the value of accumulator if
Skip the instruction followed if AC = 1

Perform ISZ to AC

T_0 : AR ← PC
T_1 : IR ← M[AR], PC ← PC+1
T_2 : D_7 — D_0 ← IR[14 to 12], AR ← IR[11 to 00]
I ← IR[15]
T_3 : Nothing
$D_7 T_4$: DR ← M[AR]
$D_7 T_5$: DR ← DR+1, if result=0 then PC ← PC+1 (it is not performed by ALU)
$D_7 T_6$: M[AR] ← DR

I/O Instruction –

- INP → f800 → Accumulator get data from I/p device
- OUT → f400 → Result show or send data to o/p register
- SKI → f200 → Skip if the I/p flag is 1.
- SKO → f100 → Skip if the o/p flag is 0.
- ION → f080 → I/p are Enable IEN = 1
- IOF → f040 → I/p are Disable IEN ← 0

(Interrupt flag)

- INP - f800 → AC ← INPR
- AC_7 — AC_0 ← Data received I/o system is 8 bit but 16 bit.
- AC_{15} — AC_8 ← Not data receive

T_3 ⇔ $D_7 . I . T_3$ ⇒ PB_{11} (11 bit is ON)

AC_7 to AC_0 ← INPR

- OUT - f400

$D_7 I T_3$: OUTR ← AC_7 to AC_0
PB_{10} : OUTR ← AC_7 to AC_0

- SKI - f200

AC_7 to AC_0 ← INPR ; FGI = 0
(Skip if FGI will be 0)

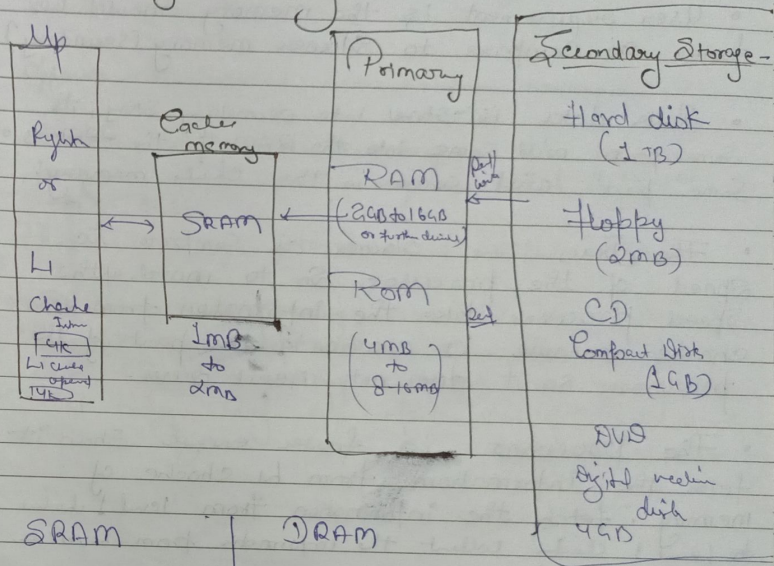
- SKO - f100

AC_7 to AC_0 → OUTR FGO = 1
Skip if FGI = 1

Memory Hierarchy

Memory Hierarchy

- Memory Hierarchy :-



SRAM	DRAM
Static Random Access memory	Dynamic Random Access memory
(Flip-flop)	(Capacitor charges & discharge)
Faster	Slower

- Secondary are slow than Primary.
(Depends on Practical Use) → because as compare to electric & chip operated memory.

- Secondary storage are cheap as compare to primary.
- Cost per bit is less.
- User requirement is the memory should be faster as compare to slower memory (secondary)
- Bulk data is stored into secondary storage, its some part will come into the RAM & the RAM its some part will come in the chache memory.
- The Buses are slower as compare to speed of the processor so to match the speed processor take the information from L1 chache because L1 chache is the part of the processor, so it does not need buses.
- The processor is faster enough than it fetch the information from L1 chache of memory, fetch the information from level 1 chache to level 1 chache collect the information from level 2 chache fetch from RAM.
RAM fetch the information from secondary storage. So this the need of hierarchy.
- Rom fetch the OS & store the programme of OS through which processor fetch the OS from secondary storage, after installing the OS the Computer System is ready to operate.

- Those types of memory depends on power supply like L2 chache & RAM so the data will be lost when the supply will be off are called volatile memory.

Opposite are called non-volatile memory.

- Rom is non-volatile memory so the primary programme of Computer system will store into the Rom.
This primary programme is called BIOS.
- SRAM (chache memory) is faster in a that why the processor operate with the chache.
But the secondary storage are very slow that why they are not enough fast enough.

Connection of memory to CPU

- Let say we need 512 bytes of RAM but the processor have 128 bytes of RAM.
- 2×4 Decoder will be used to select particular RAM

A09	A08	
0	0	→ RAM1
0	1	→ RAM2
1	0	→ RAM3
1	1	→ RAM4

- $\overline{CS2}$ pin is used to select either RAM or ROM if D_{10} bit will be 0, so all RAM's are selected if D_{10} bit is 1 ROM is selected
- CS1 → which Ram.

0–8

Q1 ① $\frac{2048}{128}$ = 16 chips.

② 2^{11} : (11) Address lines

$128 = 2^7 = 7$ lines

11 − 7 = 4 chip for decoder for selecting which to select the chip.

- 11 address line are required to access 2048 bytes of memory.
- 7 address line are required to access 128 bytes of memory.
- 4 × 16 Decoder. (16)

Q2 $\frac{4096}{128}$ = 32 RAM. $\frac{4096}{512}$ = 8 ROM chip

$4096 = 2^{12}$ = 12 common address line + line to select RAM + ROM

= 12 address lines are required here.

16 → 13 | 12 11 10 9 8 7 6 5 4 3 2 1

RAM — 5×32 Decoder → x x x x x x x

ROM — 0

write zero

2^{11} x x

RAM	
	0000 – 07ff
	1000 – 17ff

15th 8 to a chip either

• If n are the no of address line of the RAM & n+1 address line will represent either the RAM is selecting or Rom is selecting.

If the MSB of (n+1) address will be 0 So the RAM will selected if 1 Rom is selecting

Connected by CS2

Q-3

RAM = 256 × 8 = 2^8 = 8
Rom = 1024 × 8 = 2^{10} = 10

2048 → RAM. $\frac{2048}{256}$ = 8 chip = 2^{11}

4096 → Rom = $\frac{4096}{1024}$ = 4 chip = 2^{12}

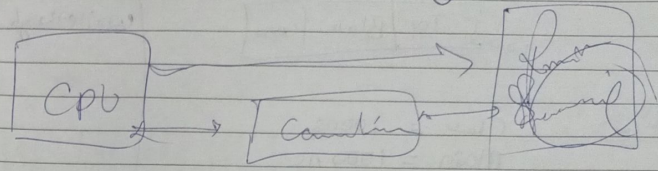
Interface = 4 × 4 = 16 register = 16 = 2^4

	16	15
RAM	0	0
Rom	9	1
Inp	1	0

CPU Connected with Primary memory -

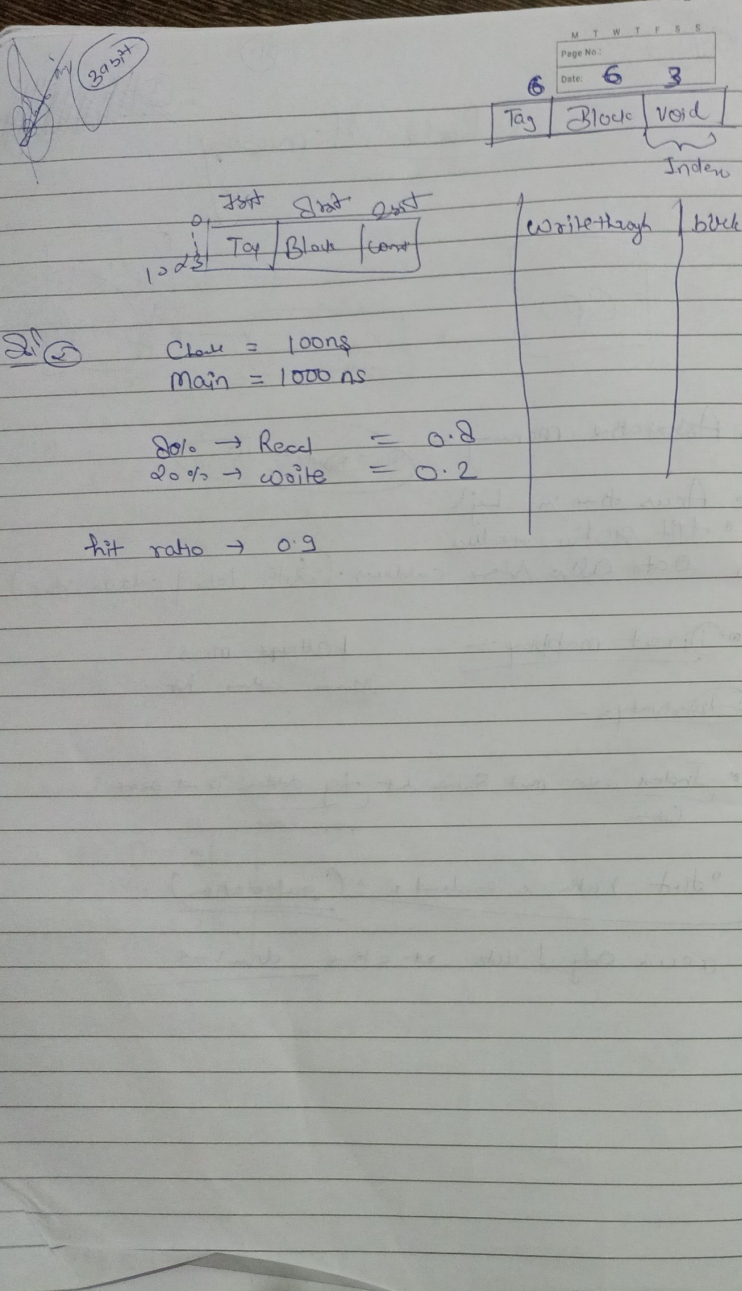
510

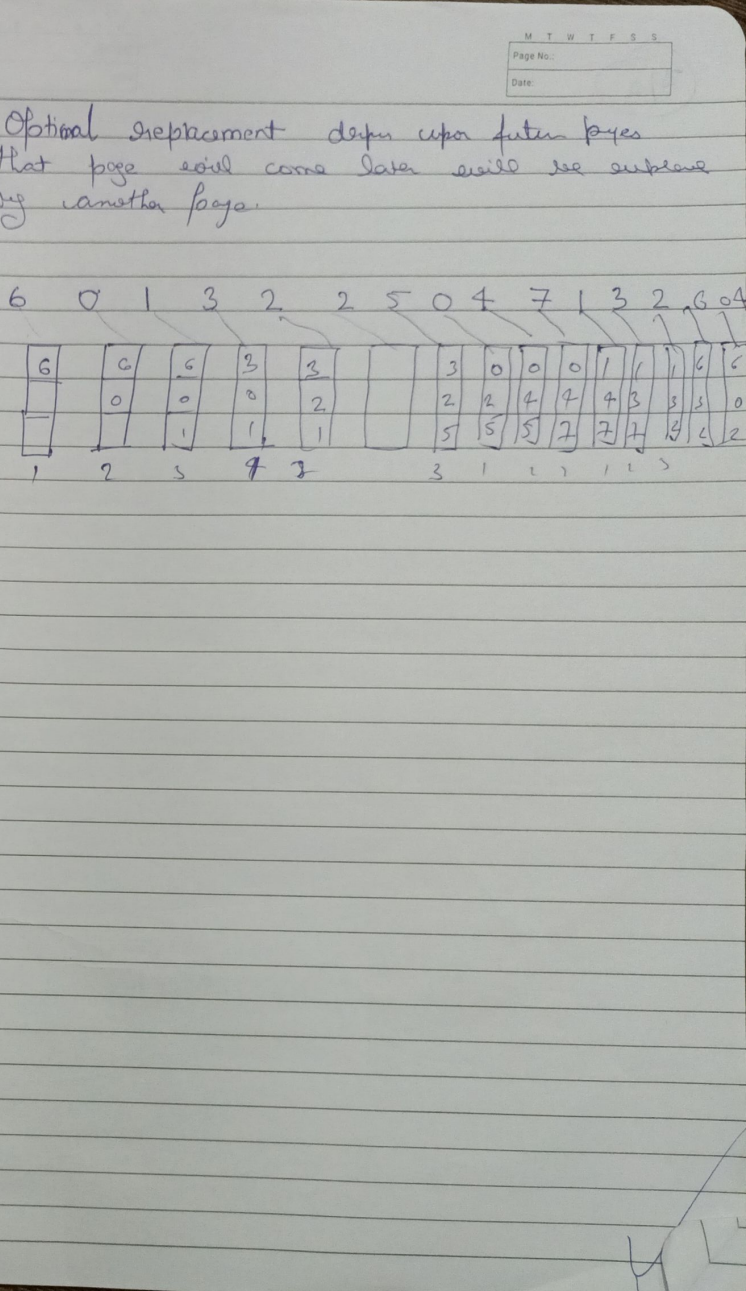
Cache Memory



- Associative memory - (1 Disadvantage)
- Access time is high
- Hit ratio increase
- Data also have address. (both data & address store)
- Direct mapping - 1 attempt miss Remain other hit.
- Disadvantage -
- Index address must same but tag address not same. Comm ram size
- Hit rate is reduced. : (Explanation).

access only 1 data at a time during


Page No.:
Date: 6 3
6
Tag
Block
Void
Index
Tag
Block
word
Write through
block
Cache = 100ns
Main = 1000 ns
80% → Read = 0.8
20% → write = 0.2
hit ratio → 0.9


Page No.:
Date:
• Optimal replacement depends upon future pages that page would come later would be replaced by another page.
6 0 1 3 2 2 5 0 4 7 1 3 2 6 0 4
1 2 3 4 7 3 1