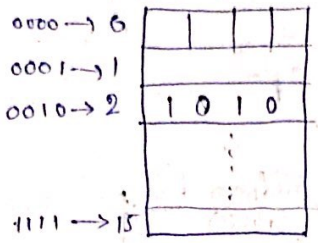
17/01/18

- **System bus** : (interconnection b/w CPU & memory)

1) Address bus
2) Data bus
3) Control bus



$N = 16$

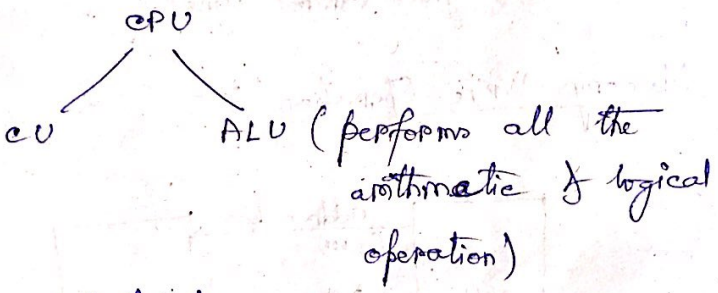
$\log_2 N$

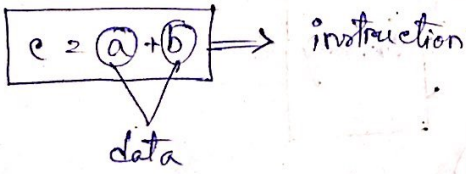
$= \log_2 16$

$= 4 \Rightarrow$ size of address bus

Size of data bus

$= 4 \Rightarrow$ because here each location requires 4 bits.



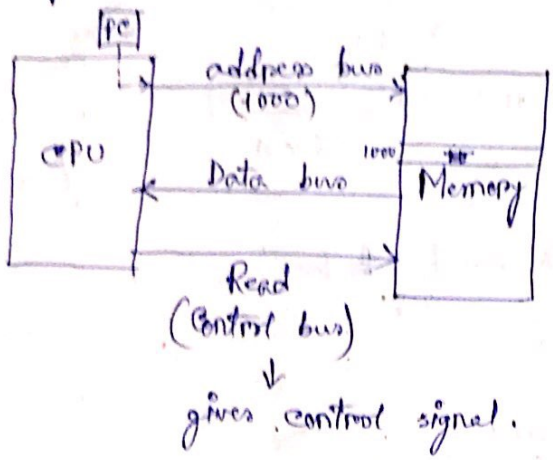


» Initially the instructions are stored in the memory in binary form.

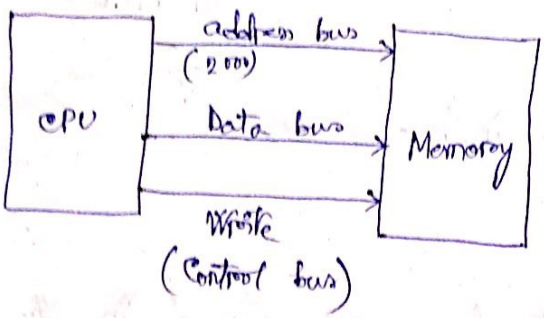
- **Steps for instruction execution cycle** :-

1) Fetch / Read the instructions from memory.
2) Decoding. (After decoding the instructions we know that this is 'add' instruction)
3) Fetch the operand. (access the content of a & b)
4) Execute the operation.
5) Storing the result.

- Program Counter → a special type of register which stores the starting address of a program.
- Memory Read :-

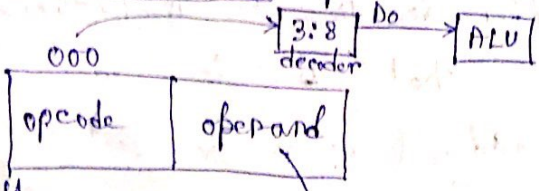


- Memory Write Operation :-



» Address bus & Control bus are unidirectional, but Data bus is bi-directional.

- Instructions has two parts —



000 Add
001 Sub
010 Mul
011 Div.

Address field or operand

(*) First, ALU gets the control signal from CU and then it performs all the arithmatic operation.

18/01/18

- **Von - Neumann Machine :-**

Stored Program Concept → is developed in Institute for Advanced Studies, (IAS) Princeton

I_1	Read a
I_2	Read b
I_3	c = a+b

⇒ There are 3 main concept for Von - Neumann Machine

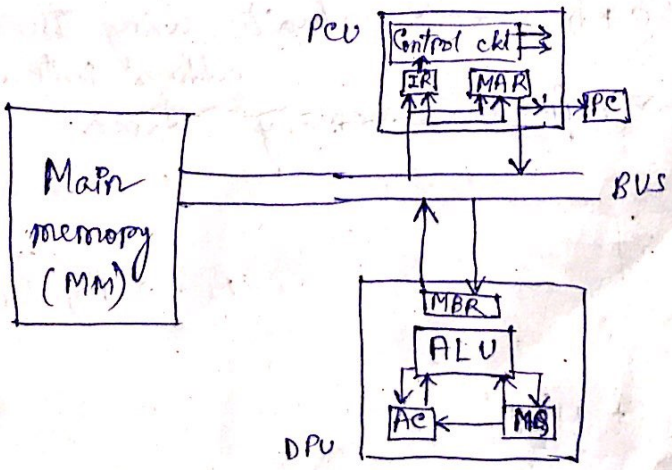
1) Data & Instruction will be stored in single memory.
2) Content of memory will be accessable by address, not by data-type.
3) Sequential execution.

This machine is also called IAS computer.

⇒ What is stored program concept ?

- **Structure of IAS computer :-**

consists of CPU, memory, register.



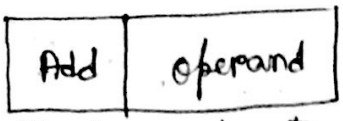
MBR → Memory Buffer Register / DR

AC → Accumulator

MQ register → multi or div.

IR → Instruction Register
MAR → Memory Address Register
PCU → Program Control Unit

DPU → Data Processing Unit



Opcode Instruction format

↓

IR

~~MBR/DR~~ AC → temporary register

PC

↓

MAR

↓

Address bus

↓

fetch the instruction from memory

↓

Opcode in IR

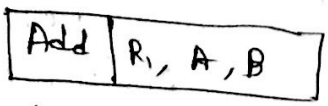
- Instruction format :-

The number of address field in the instruction format of a computer depends on the internal organisation. There are 4 types of instruction format →

1) Three Address Instruction
2) Two
3) One
4) Zero

⇒ $X = (A+B) * (C+D)$ → evaluate using three address instruction

A, B, C, D, X → represents memory address



opcode address

Ri → Register

$R_1 \leftarrow M[A] + M[B]$

↓

content of memory location A,

M[100]

Add	R_1, A, B	$R_1 \leftarrow M[A] + M[B]$
Add	R_2, C, D	$R_2 \leftarrow M[C] + M[D]$
MUL	X, R_1, R_2	$M[X] \leftarrow R_1 * R_2$

=> Two Address Instruction :—

Load (Reading) ↗ ↙ always write

MOV	R_1, A	$R_1 \leftarrow M[A]$
ADD	R_1, B	$R_1 \leftarrow R_1 + M[B]$
MOV	R_2, C	$R_2 \leftarrow M[C]$
Add	R_2, D	$R_2 \leftarrow R_2 + M[D]$
MUL	R_1, R_2	$R_1 \leftarrow R_1 * R_2$
MOV	X, R_1	$M[X] \leftarrow R_1$ => Store (writing)

=> One Address Instruction :—

In case of one address instruction Accumulator is used.

Load	A	$AC \leftarrow M[A]$
Add	B	$AC \leftarrow AC + M[B]$
Store	T_1	$M[T_1] \leftarrow AC$
Load	C	$AC \leftarrow M[C]$
Add	D	$AC \leftarrow AC + M[D]$
Store	T_2	$M[T_2] \leftarrow AC$
Load	T_1	$AC \leftarrow M[T_1]$
Mul	T_2	$AC \leftarrow AC * M[T_2]$
Store	T_3	$M[T_3] \leftarrow AC$
Store	X	$M[X] \leftarrow M[T_3]$

AC ← AC * M[T]

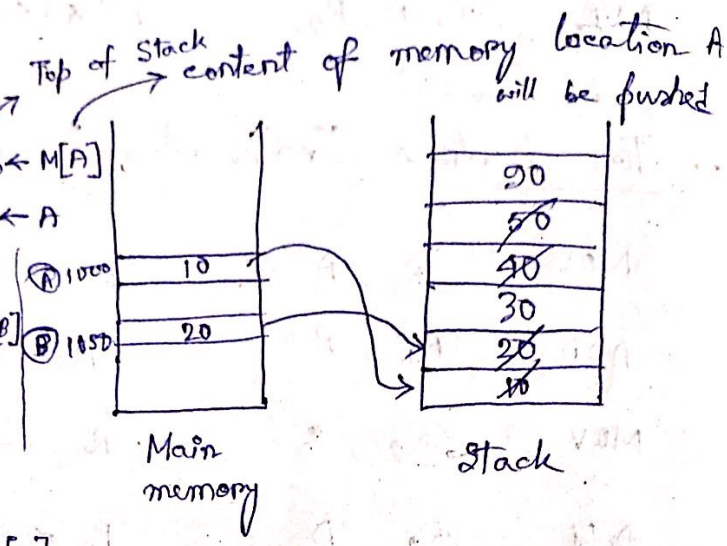
MUL	T
Store	X

M[X] ← AC

29/01/18

⇒ Zero Address Instruction :- / Stack organised instruction

Addition and multiplication instruction do not contain any address field.



PUSH C TOS ← M[C]

PUSH D TOS ← M[D]

ADD

MUL

POP X M[X] ← TOS

- **Types of Instruction :-**

1) **DATA TRANSFER :-**

MOV, LOAD, STORE, PUSH, POP

↓ ↓

Memory → Register

Register → Memory

2) **DATA PROCESSING :-**

Arithmetic, Logical

↓ ↓

ADD, MUL, DIV, SUB — XOR, XNOR, AND

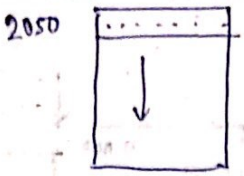
3) PROGRAM CONTROL :-

Branch instruction :

1001 : Add A

1002 : JMP 2050

1003 : JNC 2050



if (carry == 0)
    goto 2050;
else
    goto 1003;

→ ~~carry~~ Jump not carry [c == 0]

1004 : JC 2050 if (c == 1) [c == 1]

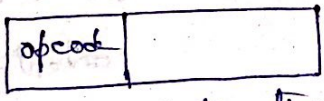
{ }

else

{ }

⇒ The way in which operands are accessed / chosen during program execution, is depended on the addressing mode.

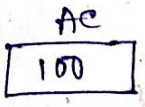
1) Immediate



→ faster, no need to address memory

Add 50

AC ← AC + 50



2) Direct

Effective address = address field → Here one ~~address~~ memory address is required.

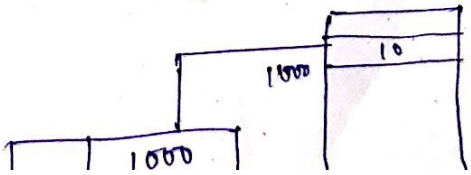
↓

Address of operand

Add 1000

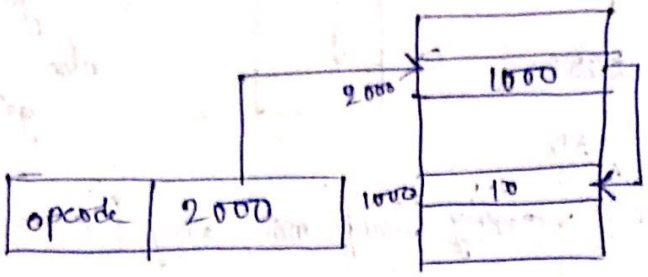
address

AC ← M[1000] + AC



3) Indirect

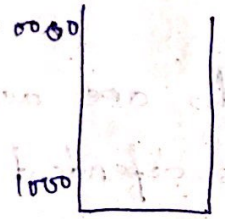
Effective address = content of (address field)
content of



Add (2000) ⟶ indirect addressing

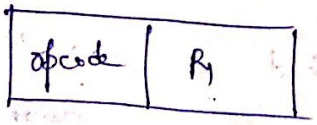
AC ← AC + M[M[2000]]

⇒ Advantage :-



Using direct addressing we can access the contents of 1000 memory locations But in case of indirect we can access the contents of more memory locations.

4) Register addressing :-



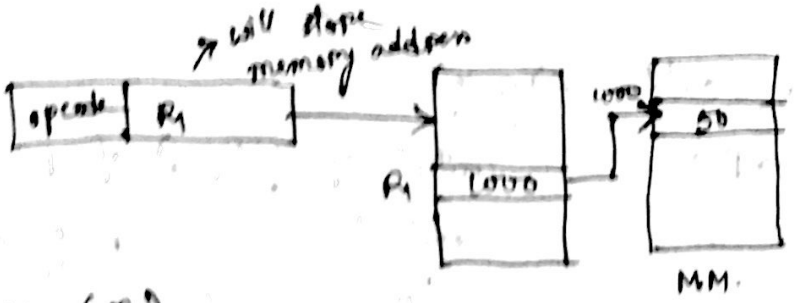
Add R1

AC ← AC + R1

Content of R1 and content of AC will be added

5) Register indirect :-

Effective address = (R1)



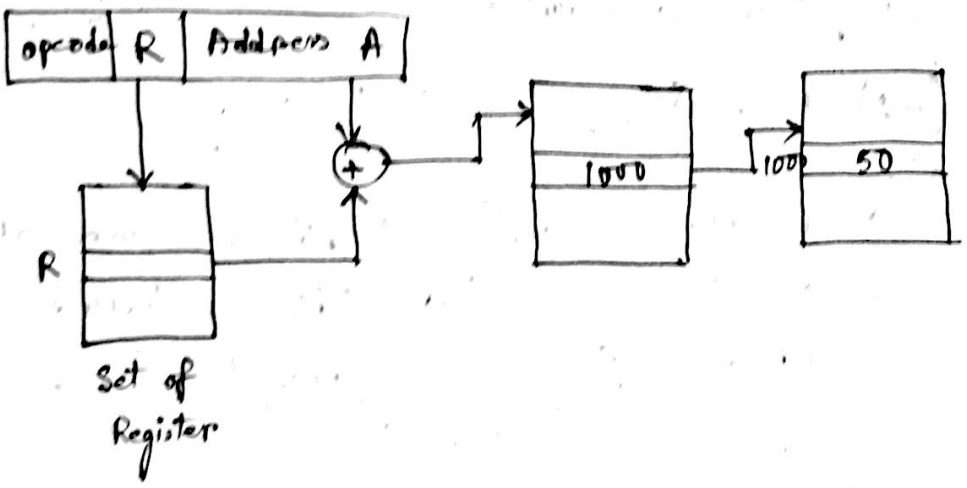
Add (R1)

$AC \leftarrow AC + M[R_1]$

⇒ Register indirect is faster than memory indirect.

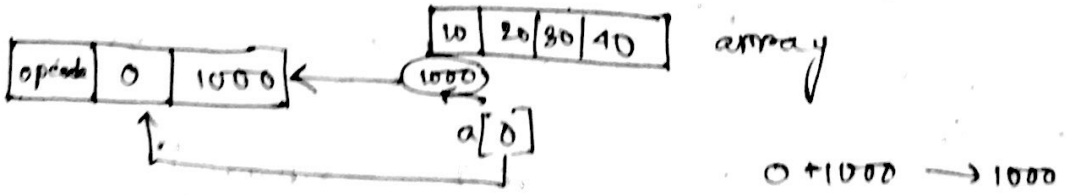
⇒ Memory direct " " " register ".

6) Displacement Addressing :-



This R may be be PC, index register, base register

(a) Index addressing :-



R → index register

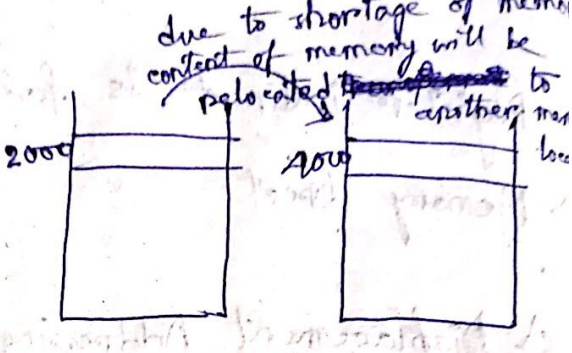
Address field → starting address

⑤ Relative addressing :-

R ⟶ PC content of (address) of PC and content of Register will be added.

⑥ Base addressing :-

R ⟵ base register



⇒ 2's compliment addition :-

(-5) + (+3)

+5 → 0101

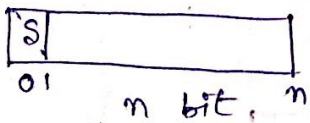
+3 → 0011

0 1000 (8)

using 4 bit we can not store (+8). This is called overflow.

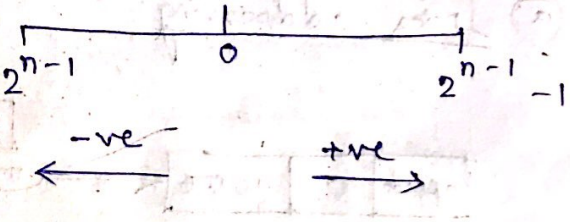
25/01/18

- Unsigned and signed number :-



S = 0 ⟶ +ve

S = 1 ⟶ -ve



n = 4 → range {-8 to +7}

+3 → 0 0 1 1

+5 → 0 1 0 1

S

0 1 0 0 0 (+8)

S

Using 4 bit we can calculate upto +7, so +8 is overflow

1 0 0 1

1 0 1 0

C_n C_{n-1}

C_4	C_3	C_2	C_1	C_0
0	1	1	1	0

$C_n \oplus C_{n-1} = 1$ Overflow occurred

⇒ (+3) + (-1)

C_4	C_3	C_2	C_1	
	1	1	0	
	0	0	1	1
	1	1	1	1
1	0	0	1	0

C_4 (+2)

$C_4 = 1$ $1 \oplus 1 = 0$

$C_3 = 1$

⇒ (-7) + (-2)

1 0 0 1

+ 1 1 1 0

1 0 1 1 0 1

$C_0 = 0$

$C_1 = 0$

$C_2 = 0$

$C_3 = 0$

$C_4 = 1$

$C_4 \oplus C_3$

$= 1 \oplus 0 = 1$ ⇒ Overflow will occur.

⇒ (+7) + (-3)

0 1 1 1

1 1 0 1

1 0 1 0 0

0 0 1 1

→ 1 1 0 0

$C_0 = 0$

$C_1 = 1$

$C_2 = 1$

$C_3 = 1$

$C_4 = 1$

$1 \oplus 1 \Rightarrow 0$

(No overflow)

⇒ (+7) + (+2)

```
  1 1 0 ◆
  0 1 1 1
  0 0 1 0
 ---------
0 1 0 0 1
```

$C_0 = 0$
$C_1 = 0$
$C_2 = 0$
$C_3 = 1$
$C_4 = 0$

↓ So, this is signed bit

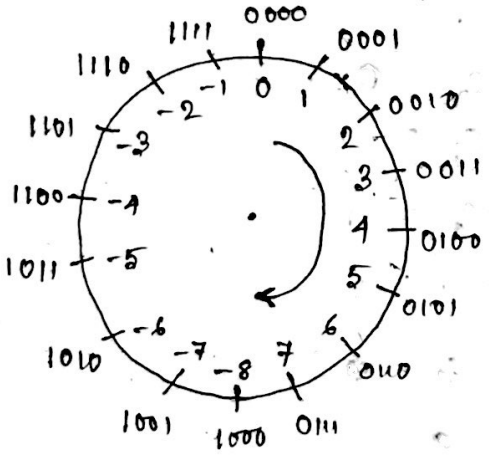
$C_4 \oplus C_3 = 0 \oplus 1$

$= 1 \Rightarrow$ Overflow

- **Addition - Modulo System :-**

N = 16 ⇒ Using 4-bit we can represent $2^4 = 16$ d

⇒ **Addition Modulo 16 :-**



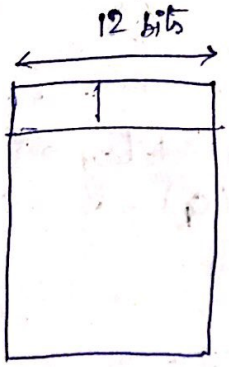
0 to 15

0 to (N-1)

Sign	
0	0 0 0 → 0
0	0 0 1 → 1
0	0 1 0
	:
0	1 1 1 → 7
1	0 0 0 → −8 (2's com
1	0 0 1 → −7
	:
1	1 1 1 → −1

+7 → Move upto +7
−3 then go upto 3
0100 places back (anticlockwise)

31/01/18



Address bus size = 16 = PC size

Data bus n = 12

PC = MAR = 16 bits

MDR = 12 bits ⟶ size of data

$2^{16} \times 12$ ⟹ Address space ⟶ how many location the processor can support.

» In case of IAS computer ⟶
IR only hold opcode.

» But in case of other computer ⟶
IR holds total instruction (opcode + operand)

∴ So, here total size of ~~inst~~ IR = 12 bits
= width of each location

Arithmetic & Logical Unit

C = A + B

Processor Performance

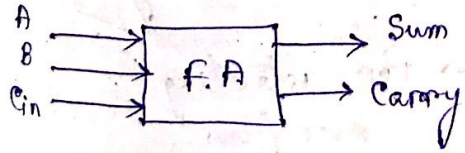
P_1 P_2

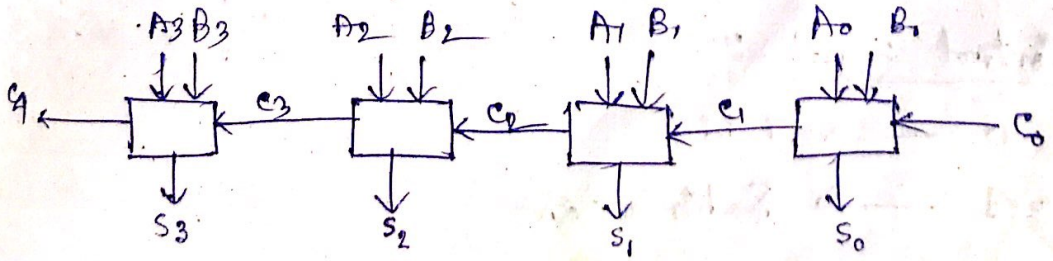
$t_1 > t_2$

- **Full Adder** ⟶

Sum = A ⊕ B ⊕ C_{in}

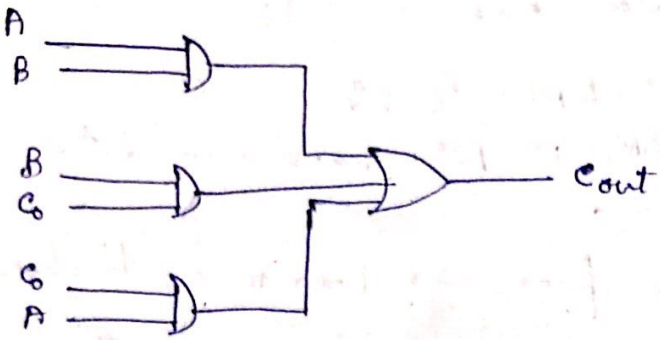
Carry = AB + BC_{in} + $C_{in}A$





C ⟶ Ripple carry Adder

⇒ • Gate delay :-



Gate delay = 2
(C_4)

Gate delay = 1
(S_0)

⇒ $C_2 = 2+2 = 4$ $\quad C_3 = 4+2 = 6$

$S_1 = 1+2 = 3$ $\quad S_2 = 4+1 = 5$

$C_4 = 6+2 = 8$

$S_3 = 6+1 = 7$

⇒ If we add 2 n bits number, then →

$C_{n-1} = 2(n-1)$ time unit.

$C_n = 2n$

$C_n \oplus C_{n-1} = 1$

$2n$ ↓ 1 G.D

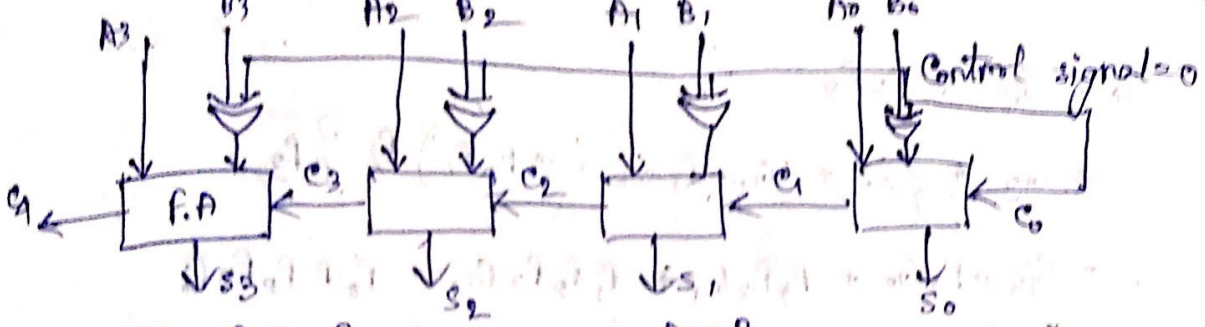
Total = $(2n+1)$ Gate Delay

↳ for detecting overflow.

⇒ Control signal :-

= 0 ⟶ Add

= 1 ⟶ Subtraction



$B \oplus 0 = B$

$B \oplus 1 = \overline{B}$

$A - B$

$= A + (2\text{'s comp of } B)$

Composite Adder-Subtractor circuit

$C_1 = 2+1 = 3$ $\quad$ $C_2 = 3+2 = 5$

$S_0 = 1+1 = 2$ $\quad$ $S_1 = 2+3 = 4$

$\overline{B}_0, \overline{B}_1, \overline{B}_2, \overline{B}_3$ will be available simultaneously for the 1st operation after one Gate delay.

$C_3 = 5+2 = 7$ $\quad$ $C_4 = 7+2 = 9$

$S_2 = 5+1 = 6$ $\quad$ $S_3 = 7+1 = 8$

$$\boxed{\begin{aligned} C_{n-1} &= 2(n-1)+1 \\ C_n &= 2n+1 \end{aligned}}$$

⇒ For overflow detection time delay = $\underline{2n+2}$

- **Carry Look Ahead Adder** :-

$$C_{i+1} = x_i y_i + y_i C_i + C_i x_i$$

$$= x_i y_i + C_i (x_i + y_i)$$

$$= G_i + C_i P_i$$

$$\boxed{\begin{aligned} P_i &= x_i + y_i \\ G_i &= x_i y_i \end{aligned}}$$

$C_1 = G_0 + C_0 P_0$

$C_2 = G_1 + C_1 P_1$

$= G_1 + P_1 (G_0 + C_0 P_0)$

$C_3 = G_2 + C_2 P_2$

$= G_2 + P_2 (G_1 + P_1 (G_0 + C_0 P_0))$

$= G_2 + (G_1 + P_1 G_0 + P_1 P_0 C_0) P_2$

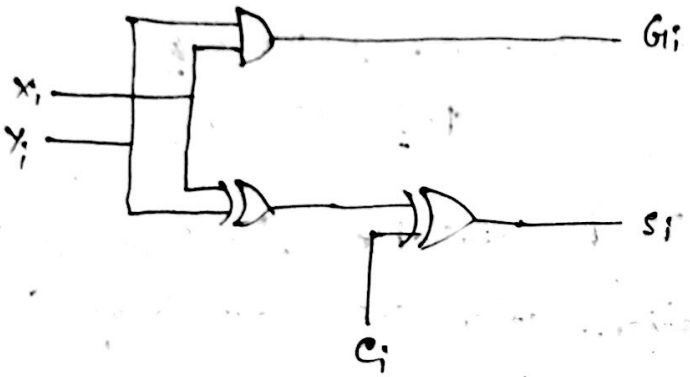
$$C_4 = G_3 + C_3 P_3$$

$$= G_3 + \{G_2 + (G_1 + P_1 G_0 + C_0 P_0 P_1) P_2\} P_3$$

$$= G_3 + P_3 G_2 + P_2 P_3 G_1 + P_1 P_2 P_3 G_0 + P_0 P_1 P_2 P_3 C_0$$

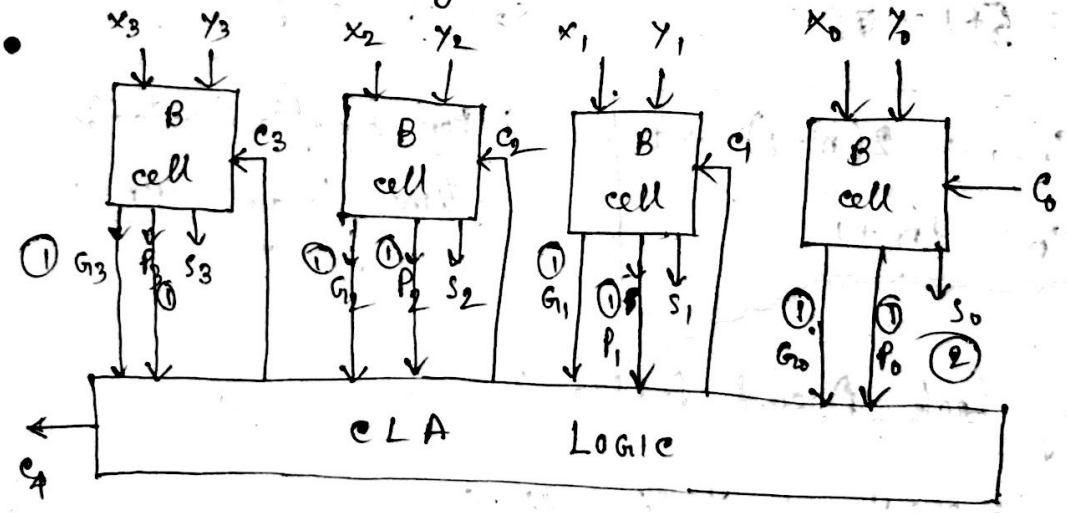
01/02/18

- Carry Look Ahead Adder :-



Bit Stage cell

- 4-b CLA



$$S_0 = P_0 \oplus C_0$$

$$= 1 + 1$$

$$= 2$$

$$C_1 = G_0 + P_0 C_0$$

$$= 1 + 1 + 1$$

$$= 3$$

$C_2, C_3, C_4 = 3$

$S_1, S_2, S_4 = 4$

$$C_2 = G_1 + P_1 (G_0 + C_0 P_0)$$

$$= G_1 + P_1 G_0 + P_0 P_1 C_0$$

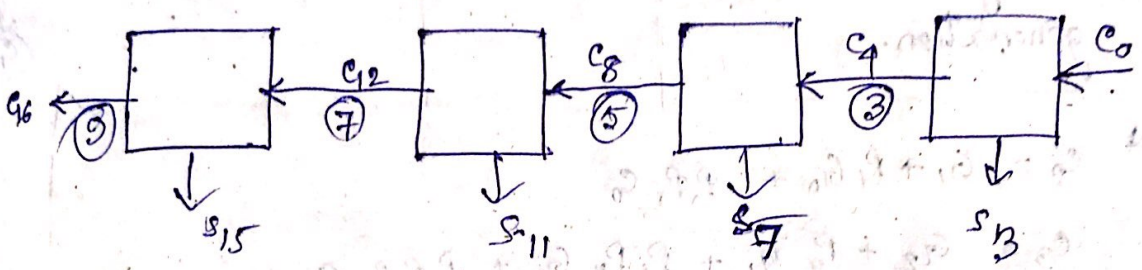
- **Fan in :-**

5 input gates are available.

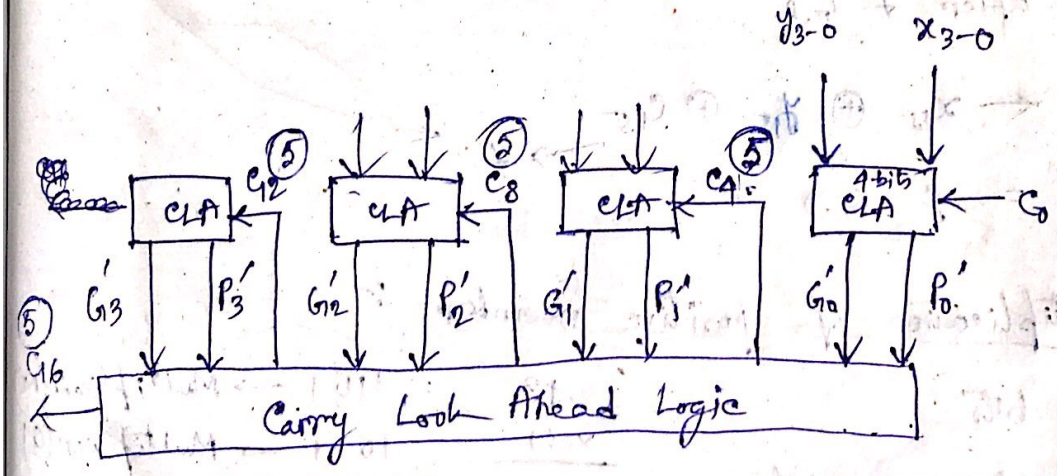
disadvantage :-

for carry look ahead adder for 16 bits it requires gate of 15 i/p AND gate which is not possible.

- **Cascading CLA :-**



- **Higher level Carry Look Ahead Adder :-**



$$C_4 = \underbrace{G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0}_{G_0'} + \underbrace{P_3P_2P_1P_0C_0}_{P_0'}$$

G_0', P_0' ⟶ 3 G.Delay

G_2', P_2' ⟶ 3 "

05/02/18

- **16-bit Adder :-**

	C_{16}	S_{15}
Ripple carry adder	$2n = 32$	$(2n-1) = 31$
Cascading CLA	9	10
$S_i = x_i \oplus y_i \oplus c_i$		
High level carry propagation & generation.	C_4, C_8, C_{12}, C_{16} = 5	8

$S_{15} = x_{15} \oplus y_{15} \oplus C_{15}$

C_{15} ← C_{12}

$C_2 = G_1 + P_1 G_0 + P_0 P_1 C_0$

$C_3 = G_2 + P_2 G_1 + P_1 P_2 G_0 + P_0 P_1 P_2 C_0$

$C_{15} = G_2^I P_2^I G_1^I + P_2^I P_1^I G_0^I + P_2^I P_1^I P_0^I C_{12}$

↳ after 7 G.D.

$S_{15} \leftarrow x_{15} \oplus y_{15} \oplus C_{15}$ → 7

= 8 G.D

- **Multiplication of positive number :-**

n-bits

product → 2n bits

13 × 11

1101 → Multiplicant (M)

1011 → Multiplier (Q)

1101 → Partial product (PP_0)

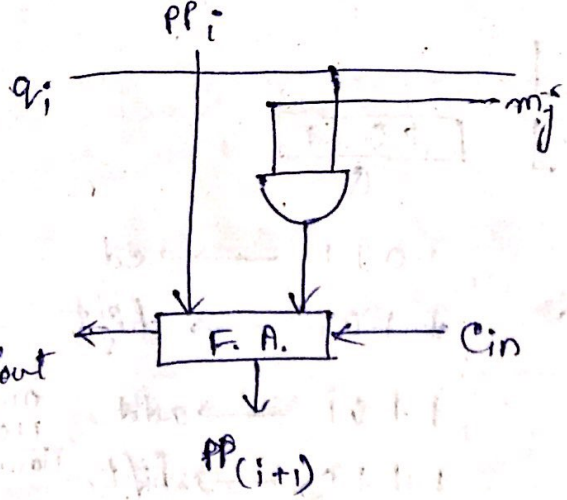
1101 → PP_1

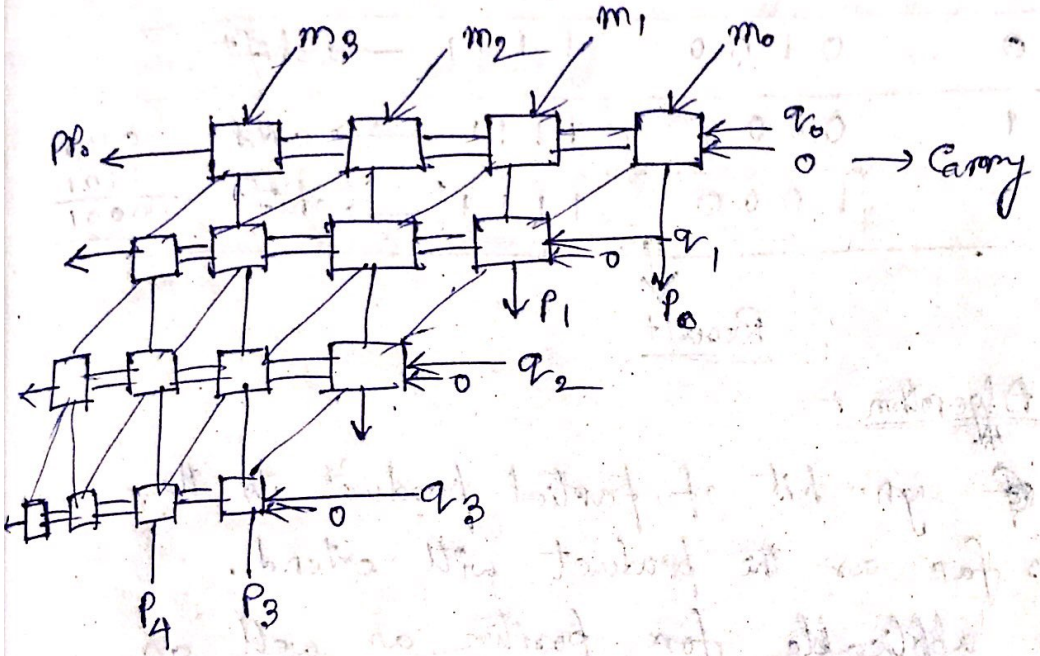
0000 → PP_2

1101 → PP_3

1 0 0 0 1 1 1 1

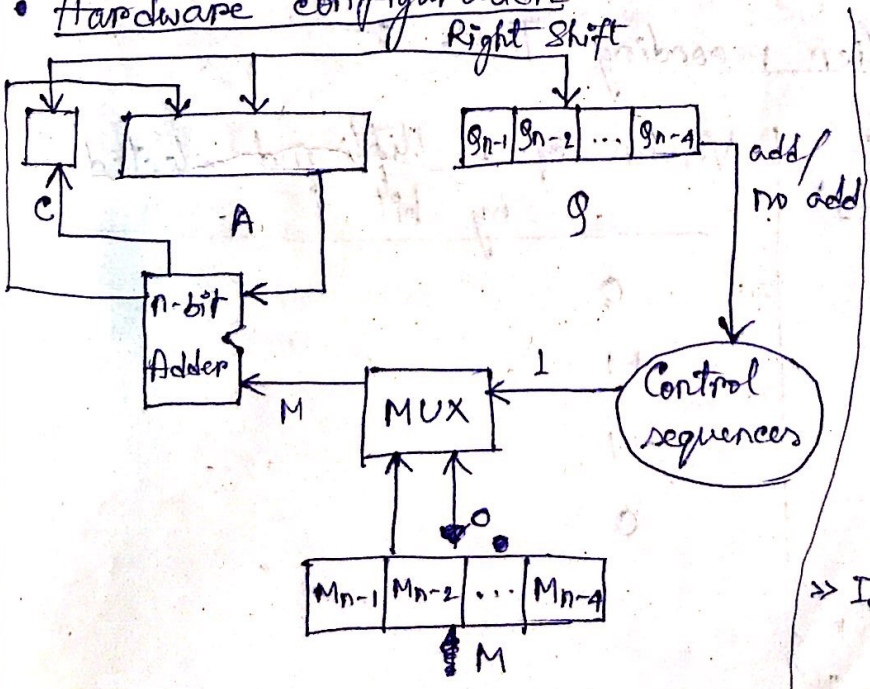
$P_7\ P_6\ P_5\ P_4\ P_3\ P_2\ P_1\ P_0$





07/02/18

- **Hardware configuration :-**



Initially, C, A ← 0.

Size of A, Q → n' bit.

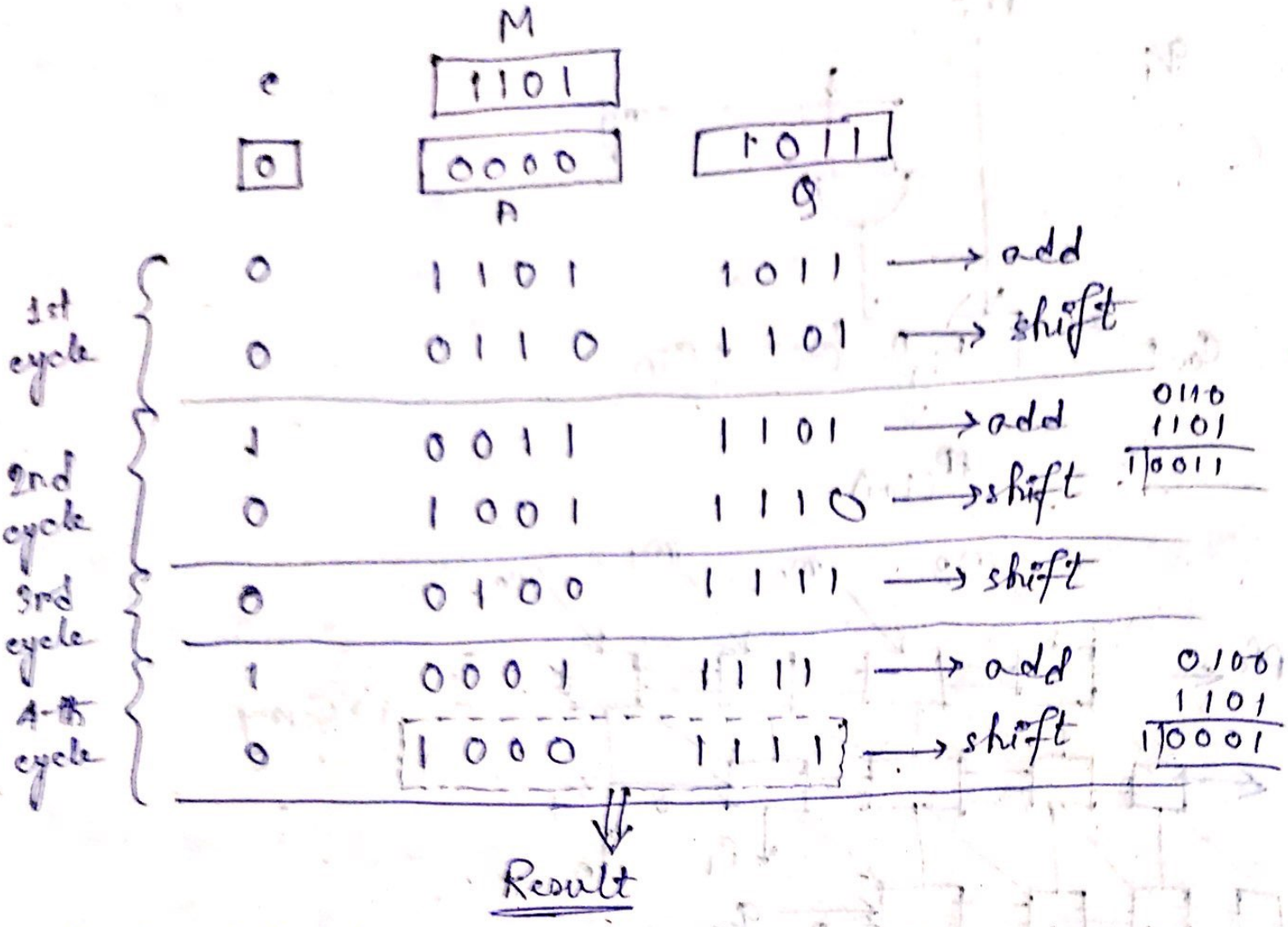
Q ← Multiplier

M ← Multiplicand

» If $q_i = 1$

→ add + shift

» If $q_i = 0$ → shift



- **Booth's Algorithm :-**

  Extend the sign-bit of partial product to the left as far as the product will extend. It is applicable for positive as well as negative product.

- **Booth Multiplier recording Table :-**

Multiplier Bit(i)	Bit(i-1)	Version of multiplicand selected by bit i

• Advantages of Booth's Algo :-

1) It is equally applicable for +ve as well as -ve multiplier.

2) If the ~~multiplier~~ multiplier consists of sequence of 1's. In case of normal multiplication no. of partial product will be total no. of 1's, but in case of Booth's Algorithm no. of PP will be reduced.

5 PP

0 1 1 1 1 0 0 → 8

+1 0 0 0 0 -1 0

↓

2 PP in Booth's algo

+13 = 0 1 1 0 1

-6 = 1 1 0 1 0

× 0 -1 +1 -1 0

0 0 0 0 0

1 1 1 1 0 0 1 1 0 ← 2's comp of (+13).

0 0 0 0 1 1 0 1 0 0

1 1 1 0 0 1 0 0 0

1 1 1 0 1 0 0 1 0 → 0 0 0 1 0 0 1 1 0 1

+1

0 0 0 1 0 0 1 1 1 0

carry (1)

↓

sign-bit

0 0 0 1 0 0 1 1 1 0 → (-) 78 (Ans)

- **Bit pair recording multiplication :-**

This technique reduces no. of PP by 2 times. for n-bit operand no. of PP will be $n/2$. And it is directly derived from Booth's Algo by grouping the multiplier bits in pair.

-1	0	-2
+1	0	+2
-1	+1	-1
+1	-1	+1
0	0	0
0	-1	-1
0	+1	+1

-2 ⟶ 2's comp and shift left

+2 ⟶ left shift

multiplication

right shift ⟶ division

-1 ⟶ 2's compliment

+13 = 0 1 1 0 1

-6 = 1 1 0 1 0

0 -1 +1 -1 0

1 ∨ ∨

0 -1 -2

1 1 1 1 1 0 0 1 1 0 ⟶ left shift result

1 1 1 1 1 0 0 1 1 x x ⟶ So we shift twice for pair.

1 1 1 0 1 1 0 0 1 0

⇒ -9 = 1 0 1 1 1

+3 = 0 0 0 1 1

0 0 +1 0 -1

1 ∨ ∨

0 +1 -1

0 1 0 0 0

+1

0 1 0 0 1

-9 = 1 0 1 1 1

+3 = 0 0 0 1 1

0 0 +1 0 -1

1 ∨ ∨

0 +1 -1

0 0 0 0 0 1 0 0 1

1 1 1 1 0 1 1 1 x x

1 1 1 1 0 0 1 0 1

0 0 0 0 0 1 1 0 1 0

+1

2's comp

$-7 = 1\ 1\ 0\ 0\ 1$

$-4 = 1\ 1\ 1\ 0\ 0$

$0\ 0\ -1\ 0\ 0$

$1 \quad \checkmark \quad \checkmark$

$0 \quad -1 \quad 0$

$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$

$0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ x\ x$

$+16\ +8\ +4\ +2\ +1$

$0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0 = (+)28$

8	4	2	1
0	1	1	ϕ
1	0	0	0

$+1$

$0\ 1\ 0\ 0$

$1\ 0\ 1\ 1$

$+1$

$1\ 1\ 0\ 0$

$0\ 0\ 1\ 1\ 0$

$+1$

$0\ 0\ 1\ 1\ 1$

08/02/18

- **Carry Save Adder :-**

$1\ 1\ 0\ 1$

$1\ 1\ 1\ 1$

$1\ 1\ 0\ 1$

$1\ 1\ 0\ 1$

$1\ 1\ 0\ 1$

$1\ 1\ 0\ 1$

RCA / CSA

$1\ 0\ 0\ 1\ 0\ 1\ 1 \rightarrow S$

$0\ 1\ 1\ 1\ 1\ 0\ 0 \rightarrow C$

$1\ 1\ 0\ 0\ 0\ 0\ 1\ 1$

$1\ 1\ 1\ 1$ A

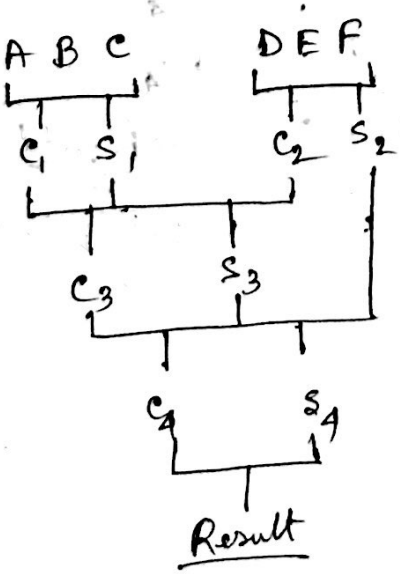
$1\ 1\ 1\ 1$ B

$1\ 1\ 1\ 1$ C

$1\ 1\ 1\ 1$ D

$1\ 0\ 1 \rightarrow S$

$1\ 1\ 0 \rightarrow C$



$1\ 1\ 1\ 1$ A

$1\ 1\ 1\ 1$ B

$1\ 1\ 1\ 1$ C

$1\ 0\ 1\ 1\ 0\ 1 \rightarrow S_1$

$0\ 1\ 1\ 1\ 1\ 0 \rightarrow C_1$

$1\ 1\ 1\ 1\ 0 \rightarrow D$

$1\ 1\ 0\ 1\ 0\ 0\ 1 \rightarrow S_2$

$0\ 1\ 1\ 1\ 1\ 0\ 0 \rightarrow C_2$

$1\ 1\ 1\ 0\ 0\ 0\ 0\ 1$ (Ans)

1 1 1 1 1 1
1 0 1 0 1 0

0 0 0 0 0 0 A
1 1 1 1 1 1 0 B
0 0 0 0 0 0 0 0 C
1 1 1 1 1 1 0 0 0 D
0 0 0 0 0 0 0 0 0 0 E
1 1 1 1 1 1 1 0 0 0 0 F

0 0 0 0 0 0 A
1 1 1 1 1 1 B
0 0 0 0 0 0 C
0 1 1 1 1 1 1 0 → S_1
0 0 0 0 0 0 0 0 0 → G
0 0 1 1 1 1 0 0 → C_2
0 0 1 0 0 0 1 1 1 0 → S_2
0 0 0 1 1 1 0 0 0 0 → C_3

1 1 1 1 1 1 D
0 0 0 0 0 0 E
1 1 1 1 1 1 F
1 1 0 0 0 0 1 1 → S_2
0 0 1 1 1 1 0 0 → C_2

0 0 1 0 0 0 1 1 1 0 → S_3
0 0 0 1 1 1 0 0 0 0 → C_3
1 1 0 0 0 0 1 1 → S_2
1 1 0 0 0 1 0 1 1 1 0 → S_4
0 0 0 0 1 0 1 0 0 0 0 0 → C_4
0 1 1 0 1 1 0 0 1 1 1 0 ⟹ Result

0 0 1 0 0 0

14/02/18

- **Division :-**

1000 | 11

1000 → Dividend
11 → Divisor

» **Longhand technique :-**

11 | 1000 | 0
11
-ve remainder

⇒

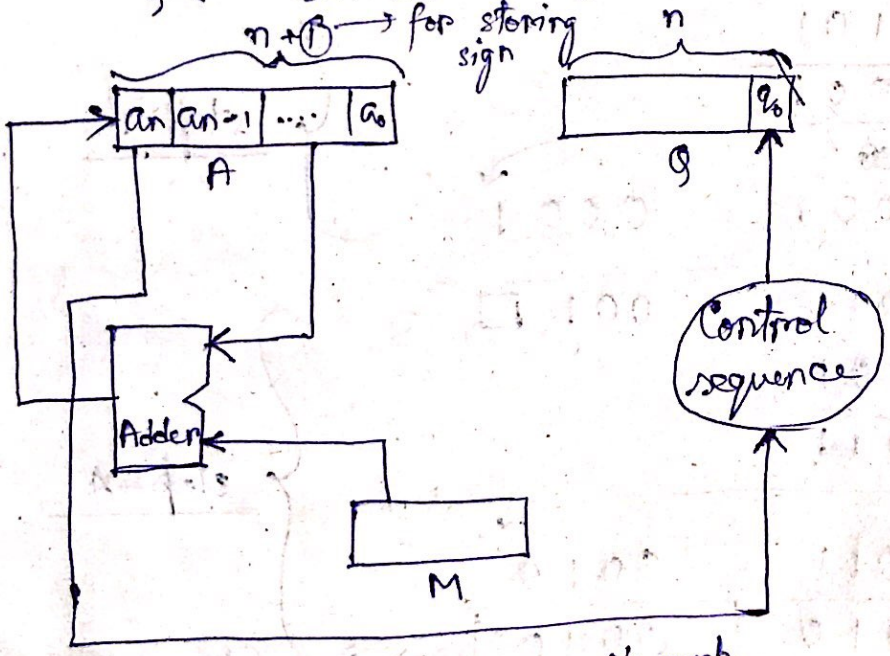
11 | 1000 | 01 → Quotient
11
10 → Remainder

» **Restoring :-**

Do the following for n times. [n → dividend's bits]

1) Shift A and Q left one bit position.

Q ← Dividend , M ← divisor , A ← 0.

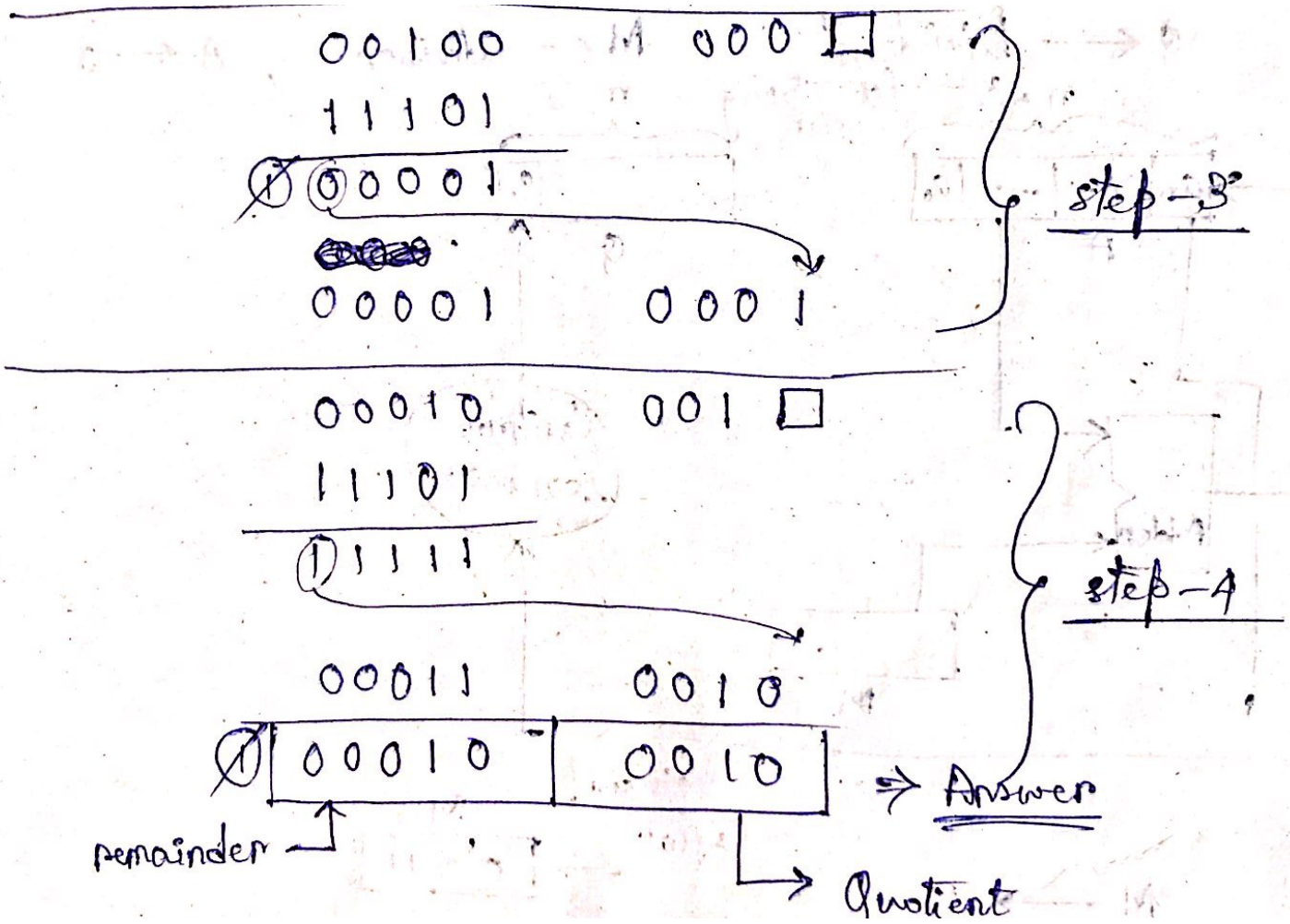


2's comp
1101 ← M
0011

M → 11
Q → 1000

⇒

	A	Q
	00000	1000
Shift	00001	000 □
Sub	+ 1.1101 → (2's comp of M)	
Step-1	(1)1110	



A	Q	
0 0 1 1	1 0 □	
1 1 0 0		
1 0 1 1 1		
0 0 1 0 0	1 0 0	Step - 2
0 0 1 1	0 1 □	
1 1 0 0		
1 1 1 1		
0 1 0 0	0 1 0	Step - 3
0 0 1 1	0 1 0	
0 1 1 0	1 0 □	
1 1 0 0		
0 0 1 0		
0 0 1 0	1 0 1	
0 0 1 0	1 0 1	

- Non - restoring :-

(S-1) 1) If the sign of A is 0, shift A and Q left one bit position.

$A \leftarrow A - M$

else

left shift A and Q

$A \leftarrow A + M$.

2) If sign of A is 0

$q_0 \leftarrow 1$

else

$q_0 \leftarrow 0$.

(S-2) If the sign of A is 1,

$A \leftarrow A + M$

1010
1011

M = 01101

	A	Q
	00000	1100
shift	00001	100□
	11011	
	11100	
	11100	1000
	11001	000□
	01010	
	00011	
	00011	0001
	00110	001□
	11011	
	00001	
	00001	0001
	00010	001□
	11011	
	11101	
	(1)11101	0010
+	01010	
	00111	0010

(1) (2) (3) (4)

15/02/18

- Floating point representation :—

fractional no.

(1) Exponent
(2) Mantissa
(3) Sign of exponent
n n mantissa

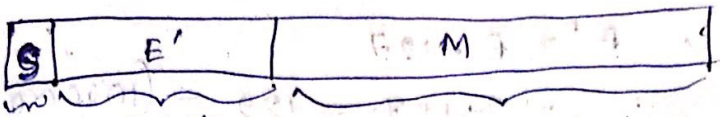
4.567 → Normalized form

0.75 × 2 = 1.50 — 1
0.50 × 2 = 1.0 — 0

$(0.75)_2 = (0.11)_2$

IEEE-754 format :-

1) Single precision → 32-bit
2) Double " → 64-bit



1-bit | 8-bits for exponent, bias = 127 | 23-bits mantissa

$bias = 2^{n-1} - 1$, $n = 8$

E' → biased exponent

$E \leftarrow E' - 127$

↓ actual exponent

① $E' = 0$, $M = 0$
$N = 0$

② $E' = 0$, $M \neq 0$
Denormalized form
0.0110

③ $E' = 255$, $M = 0$
$N = \infty$

④ $E' = 255$, $M \neq 0$
N = Not a Number (NaN)

Eg: 0/0, $\sqrt{-1}$

E'

$0 \leq E' \leq 2^n - 1$

$0 \leq E' \leq 2^8 - 1$

$0 \leq E' \leq 255$

$\boxed{1 \leq E' \leq 254}$

$E' = 0$, $E' = 255$ } Reserved

$\Rightarrow$ $1 - 127 \leq E \leq 254 - 127$

$-126 \leq E \leq 127$

⇒ 4.5 in IEEE-754 32 bit format.

0.5×2 = 1

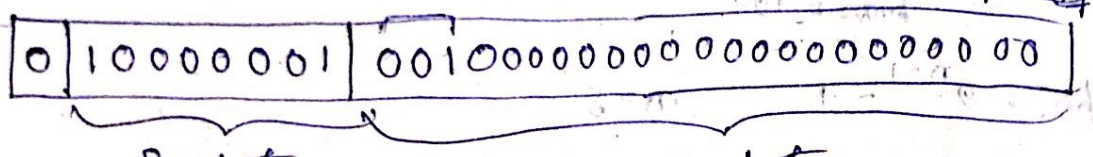
= 100.1

= 1.001×2^2 (Normalized form)

E = 2 , E' = E + 127

= 2 + 127 = 129 = $(10000001)_2$

2 | 129



⇒ 1.1×2^{-1}

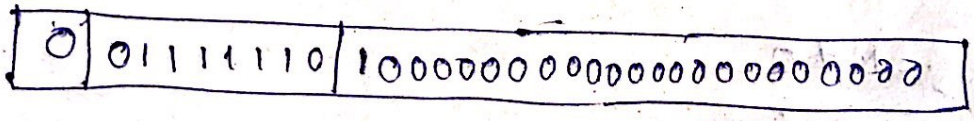
E = −1 E' = 126

= 111

= $(01111110)_2$

2	126
2	63
2	31
2	15
2	7
2	3
2	1
	0

0 1 1 1 1 1 1

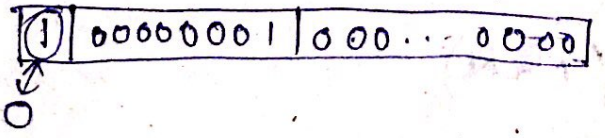


⇒ Range of Single Precision format :-

Min

$E' = 00000001$ $E' = 1$ $E = 1 - 127$

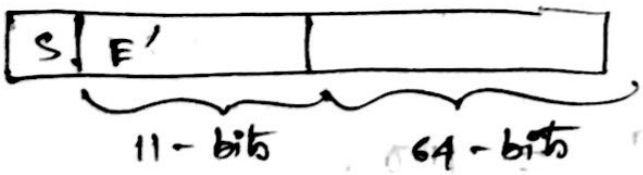
$M = 00000\ldots0$ $= -126$



Actual no = $\pm 1.0 \times 2^{-126}$

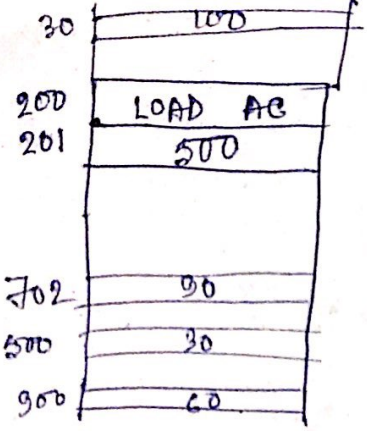
(-ve) overflow (-ve) underflow (+ve) underflow (+ve) overflow

- Double Precision :-



1) 15.375

2) 62.1234



1) Effective Address = 500

Content of AC = 30

2) Immediate :-

operand = 500.

EA = 201

Content of AC = 500.

3) Indirect :-

EA = M[500]

operand = M[30] = 100.

Content of AC = 100

4) Relative :-

EA = content of PC + address field

= 202 + 500

= 702

operand = M[702] = 90

content of AC = 90.

5) Index :-

EA = content of index reg + address field

= 400 + 500

= 900

operand M[900] = 60

content of AC = 60.

6) Register indirect (use R_1) :-

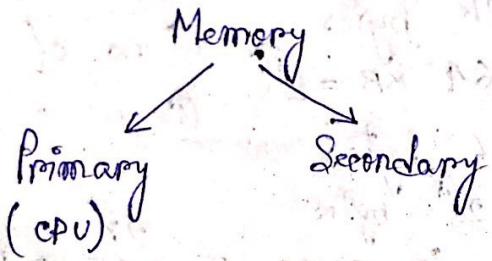
EA = (R_1)

= 400

operand = M[400] = 70

content of AC = 70.

21/02/18



Information required by CPU is stored in pri. memory. If more than one software is present in computer & we are using one software currently → the other softwares will be stored in secondary memory as backup.

- Boot Strap Loader :-

1) Loads OS from disk to primary memory
2) Prepare OS for general use
3) Transfer the control to OS.

Loaded in ROM, as initial program is required for starting.

08/08/18

1) Direct Mapping :-

$a = b \bmod M$

[a = cache block no.
b = Main memory (MM) n. no.
m = no. of blocks in cache]

2) Associative Mapping :-

Any block from MM can be mapped to any block in cache memory.

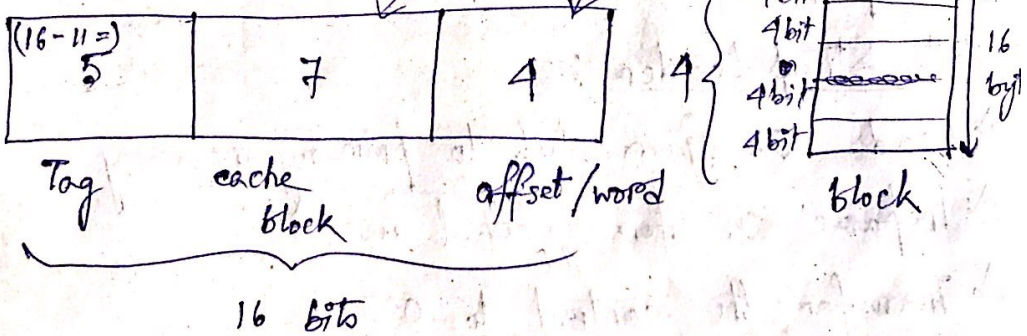
⇒ cache size = 2K bytes
MM n = 64 KB = 2^{16}
Block n = 16 bytes. = 2^4 bytes

(1) no. of block in CM ?
(2) n n n = MM ?

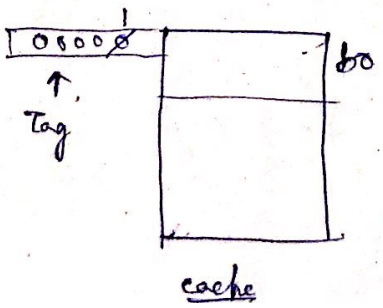
∴ no. of block in cache = $\frac{2K}{16} = 2^7 = 128$

size of MM address bus = 16 bits

⇒ Direct Mapping



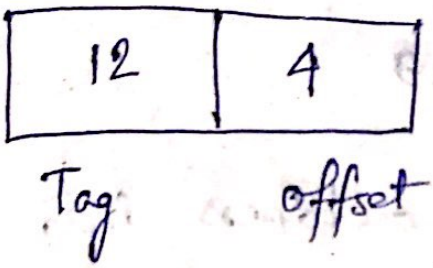
[In case of direct mapping, block 0, 128, 256 ... all will be mapped to cache block 0.]

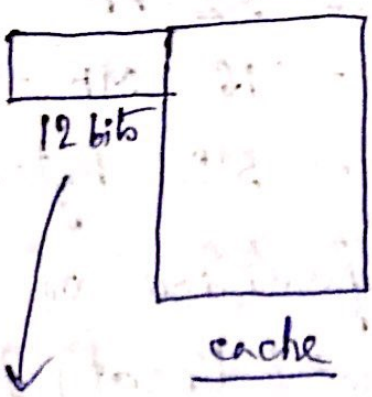


No. of comparison = 1

(*) Drawback :-
→ So, previous data is overwritten.

⇒ Associative Mapping :-





No. of comparisons = all the tags attached to cache memory × no. of blocks in cache

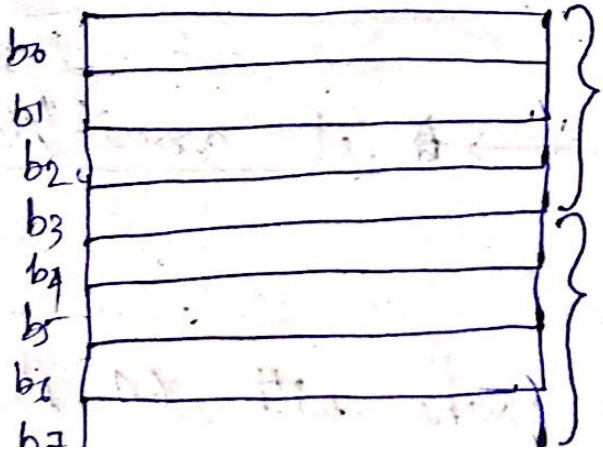
= 1 × 128 = 128

• Drawback :-

In terms of performance it is best, but cost will be increased.

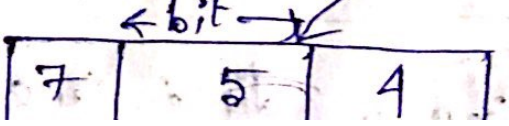
⇒ Set associative mapping :- (k way)

Each set contains k no. of blocks.



∴ no of sets

$= \frac{128}{4} = 32 = 2^5$



⇒ cache size = 32 KB

MM n = 16 MB = $2^4 \times 2^{20} = 2^{24}$

block n = 512 bytes = 2^9

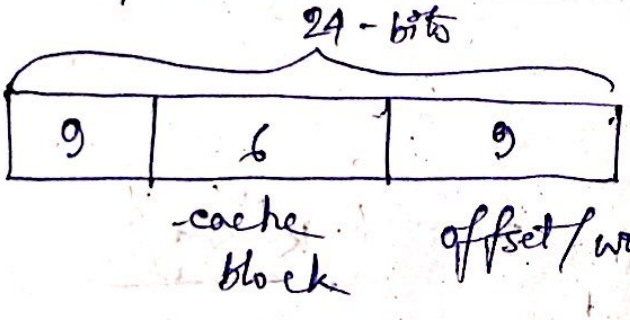
Show the partition of MM address into tag, index, offset (direct) & 4 way set associative

∴ no. of cache block = $\frac{32 \times 1024}{2^9}$

= $\frac{2^5 \times 2^{10}}{2^9} = 2^6$

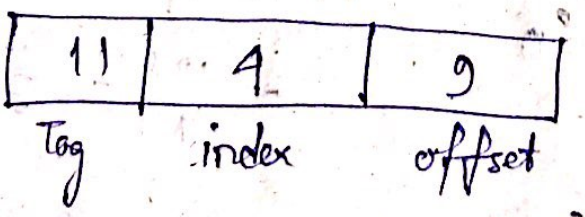
= 64 bits

∴ size of MM address bus = 24



→ Direct mapping

∴ no. of sets = $\frac{64}{4} = 16 = 2^4$



→ set associative

Word 1200 ⟶ MM block no. = $\frac{1200}{16}$ = 75

for direct mapping ⟶

$a = b \bmod m$

$= 75 \bmod 64$

$= 11$

∴ cache block no. = 11. (Ans.

⇒ Consider a 2-way set associative cache with 128 blocks, & block size = 16 bytes. find out the set which will contain MM address 1500?

Sol^n:- Each set contains 2 no. of blocks.

No. of sets = $\frac{128}{2}$ = 64

∴ MM block = $\frac{1500}{16}$ = 93

∴ set no = 93 mod 64

= (29) (Ans)

- Read Miss :-

1200 ⟶ address of MM

CPU will search data at address 1200.

While reading, If the data address is found in cache memory ⟶ then it is called Read Hit.

- Cache Write :- MM memory address = 16.

CPU always generate address for MM not for cache memory.

While writing - if the data address is found in cache memory ⟶ then it is called Write Hit.

If data address is not found while writing then it is called, Write Miss.

- Write through :-

» While writing operation, MM & cache memory both will be updated.

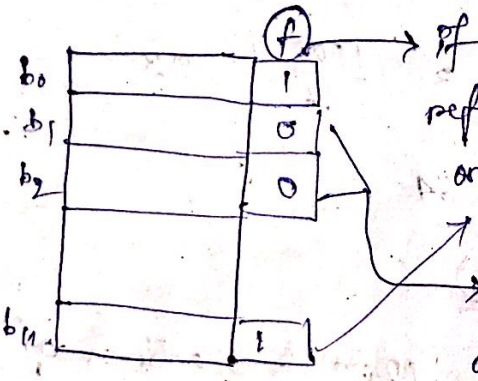
» Advantage :-

Here MM & cache memory both contains same data.

- Write back :- (faster) → advantage over write through.

Only cache memory will be updated.

- When we are updating any location in cache memory, the flag bit is set.



If flag bit is 1 while refreshing cache memory, then only MM will be updated.

MM will not be updated while refreshing.

Disadvantage :-

⇒ for write back, when one block is updated 1/multiple times in cache memory, then the data in MM & cache memory does not remain same.

- Write allocate :- from MM

Blocks may be brought into cache memory. After that it will be updated.

(in case of write miss)

- Write no-allocate :- (in case of write miss)
Blocks is updated in MM only.

⇒ Consider a fully associative write back policy.
Some sequence of memory operation is given →

write M[100]
write M[100]
Read M[200]
write M[200]
write M[100]

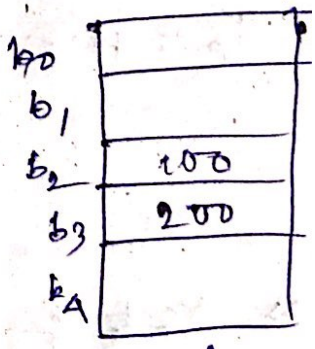
Find out no. of hits →
i) write allocate
ii) " no- "

(i) Write allocate :- (3 hit)

Miss, hit, miss, hit, hit

[initially cache memory does not contain any data] → [address 100 is allocated after this step]

[because CM contains only address 100] → [Now, address 200 is allocated]



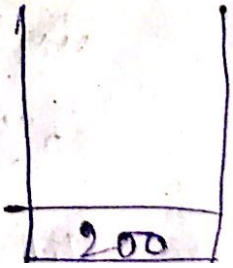
cache

(ii) Write no-allocate :- (1 hit)

miss, miss, miss, hit, miss

↓ empty

↓ (only MM is updated for read operation, address



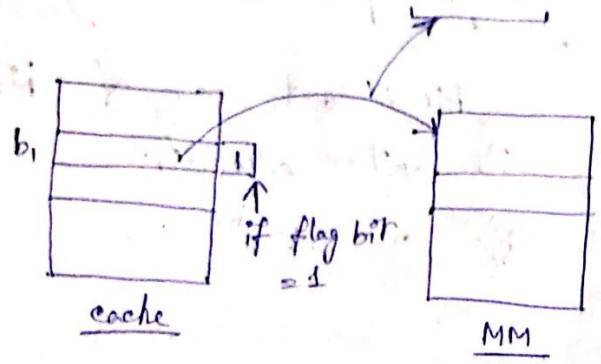
14/03/18

- Miss penalty :-

T_{CA} → Cache Access Time
T_{MS} → MM " "

Write back

» Read : $T_{CA} + T_{MS} + P_d T_{MS}$ [P_d = Prob. that flag is set]



Avg. memory access time for read →

$T_R = H \times T_{CA} + (1-H)\{T_{CA} + (1+P_d) T_{MS}\}$

Read Miss Penalty

» Write :- $2T_{CA} + T_{MS} + P_d T_{MS}$

Avg. memory access time for write →

$T_W = H \times T_{CA} + (1-H)\{2T_{CA} + (1+P_d) \cdot T_{MS}\}$

⇒ cache access time = 90 ns
MM " " = 500 ns , $P_d = 0.5$
80% memory read , 20% write
Hit ratio for write operation, $H_R = 0.9$
$H_W = 0.8$

1) Calculate memory access time for read operation for both write back & write through.

2) Avg memory access for both read & write operation for both write back & write through.

Soln: $P_d = 0.5$. (Given)

1) $T_R = (0.9 \times 90) + 0.1\{90 + 1.5 \times 500\}$ ⇒ Write back

Write through :-

$T_R = (H_R \times T_{CA}) + (1 - H_R)(T_{CA} + T_{MS})$

$= (0.9 \times 90) + 0.1(90 + 500) = 140$ ns

Write back :-

$T_W = (0.8 \times 90) + (1 - 0.8)\{2 \times 90 + 1.5 \times 500\}$

$= 258$ ns

$T_{avg} = P_R \times T_R + P_W T_W$

$= 0.8 \times 165 + 0.2 \times 258$

$= 183.6$ ns.

Write through :-

$T_W = T_{MS} = 500$

$T_{avg} = 0.8 \times 140 + 0.2 \times 500$

$= 212$

⇒ cache access time = 50 ns

MM " " = 500 ns

$P_R = 0.8$, $H_R = 0.9$

write through

1) Avg access time for only Memory read operation

2) " " " " both read & write "

- **Cache Replacement Algorithm :-**

For Direct Mapping, no such algorithm is req

b_0	4
b_1	5
b_2	14
b_3	11
b_4	12
b_5	13

cache

1) FIFO → replace the block which is accessed for longest t
2) LRU → Least Recently Used
3) Random → Select one block for replacement

Eg :- 5, 4, 2, 3, 5, 1, 5, 10, 3 → MM blocks.

1) Calculate no. of hits & no. of miss.

~~5~~	1
~~4~~	5
~~2~~	10
	3

First time when cache memory is empty then always miss occur. It is called compulsory miss (for 5, compulsory miss)

i) FIFO →

M, M, M, M, H, M, M, M, H H = 2

ii) LRU →

M M M M H M H M H H = 3

5
~~4~~ 1
~~2~~ 10
3

⇒ Diff. b/w FIFO & LRU.

- Types of cache miss :-

1) Compulsory
2) Capacity
3) Conflict → direct mapping, set associative

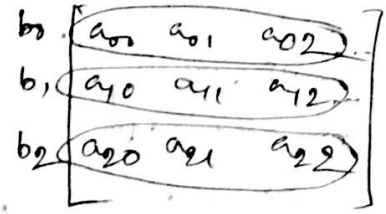
→ 0, 16, 32 all maps to the same block.

- Techniques to reduce cache misses :-

1) Increasing cache size
2) n block n
3) Higher associativity
4) Compiler optimization

(i) Loop interchange → column wise

```
for( j=0 ; j<1000 ; j++)
{   for( i=0 ; i<1000; i++)
    {
        a[i][j] = 2 * a[i][j] ;
    }
}
```



1000 blocks

(ii) Loop fusion : Combines 2 or more loops in one.

```
for( i=0; i<100; i++)
{   for(j=0; j<100; j++)
    {
        a[i][j] = b[j][j] + c[i][j];
    }
    a[i][j] = 2 * a[i][j];
}
```

loop fusion →

(iii) Merging arrays :- combines 2 separate arrays. (that might conflict for a block in cache memory).
into one seperate array.

```
int val[10];  ⟶ b0
int key[10];  ⟶ b16
struct merge
{  int val;
   int key;
}[10];
```

15/03/18

- Memory Expansion :-

1) Vertical expansion :
increasing no. of memory location or no. of memory word.
⇒ increase the size of address bus.

2) Horizontal expansion :-
increasing word length.
" size of data bus.

⇒ How many 256 × 8 RAM chips are needed to provide memory capacity of 1K × 16 ?

no. of chips ⟶

$= \frac{1K \times 16}{256 \times 8} = (4) \times (2)$

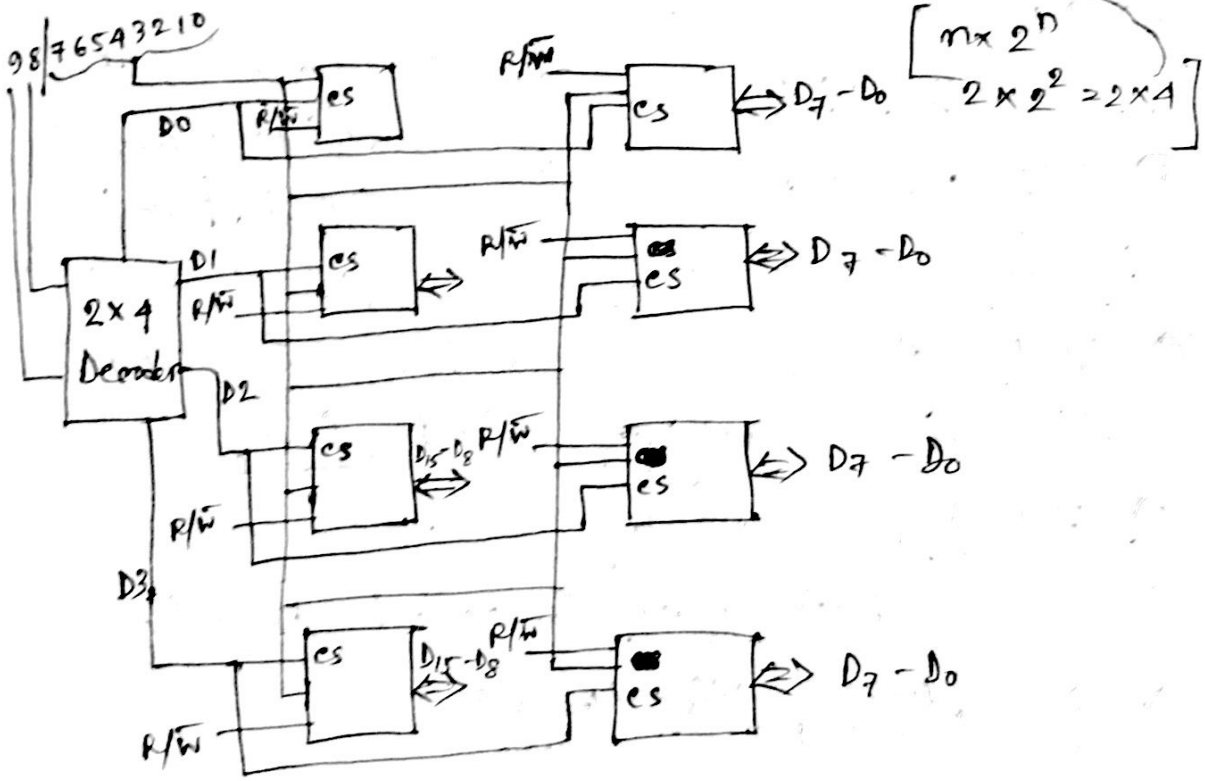
(4) → vertical, (2) → horizontal

2) How many lines of address bus must be used to access 1K × 16 memory? ⇒ $\log_2 1024 = 10$

3) How many of these lines will be common for all chips? ⇒ $\log_2 256 = 8$ $\quad (10 - 8) = 2 = n$

⇒ Specify the size of decoder. (2 × 4) decoder

$[m \times 2^n \quad 2 \times 2^2 = 2 \times 4]$

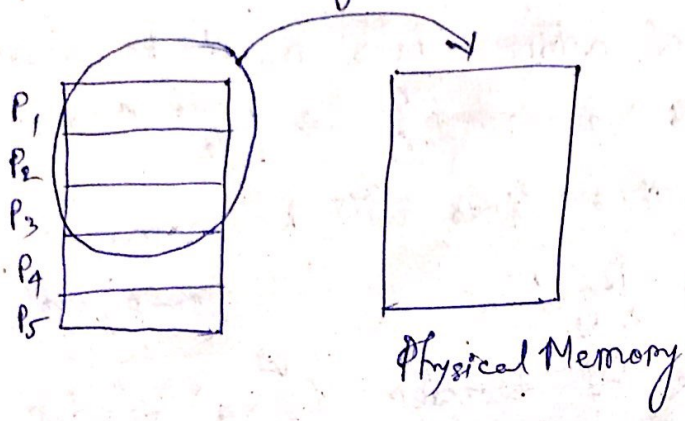


9) How many 512 K × 8 RAM chips are needed to provide a memory capacity of 8M × 32?

no. of chips = $\frac{8 \times 2^{20} \times 2^5}{512 \times 2^{10} \times 2^3} = 16 \times 2^2$

- Virtual Memory :-

It allows a execution of the program that may not be completely in MM (Physical Memory).



$P_1 P_2 P_3$ is currently utilizing by cpu. So it is loaded to Physical Memory (MM).

P_4, P_5 is not utilized. So, these 2 are in 2ndary/ Virtual Memory.

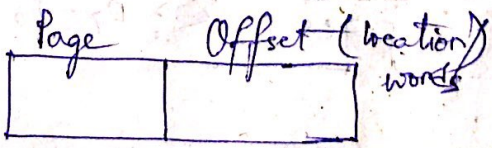
(**) Address of virtual memory — Virtual Address.

(**) Address of MM - — Physical Address.

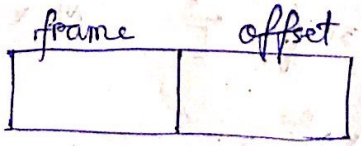
(**) Total virtual memory is divided into some blocks. Each block is called Page.

(**) Total Physical memory ~ . Each block is called Frame.

⇒ (**)



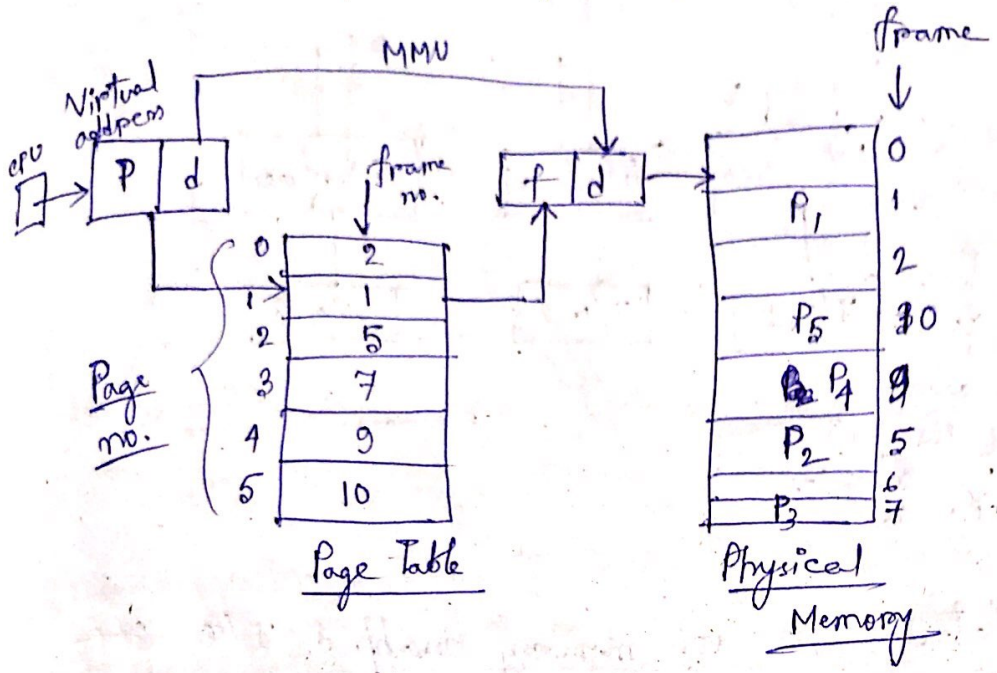
virtual address



Physical address

• Memory Management Unit :- (MMU)

It maps virtual memory to physical memory.

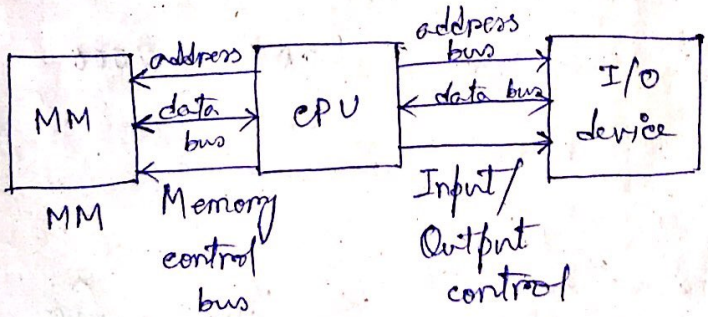


⇒ Virtual add = 48 bits
Phy. " = 36 "
Page size = 2 KB

How many virtual pg & phy. frames exist in the system?

9/03/18

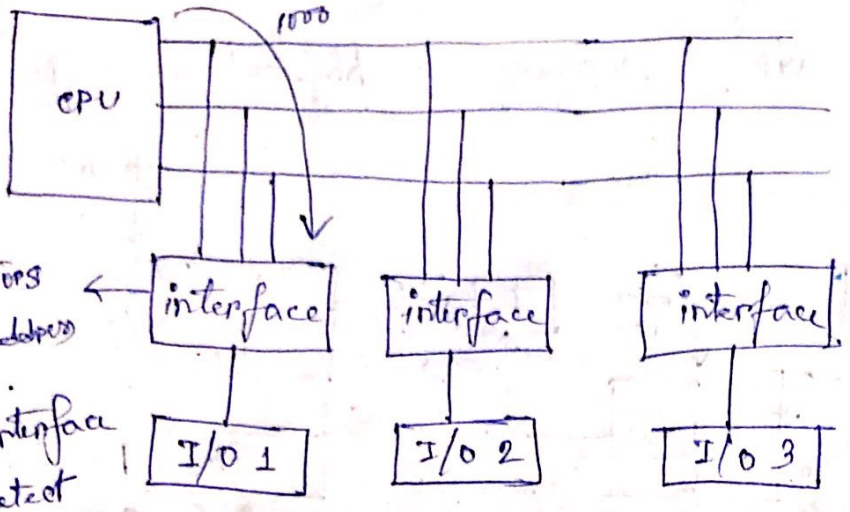
• Input and Output Device :-



I/O device is called Peripheral device.

CPU	I/O device
i) electronic	i) electro-mechanical
ii) rate of data transfer — higher	ii) lower.
iii) data format.	iii)

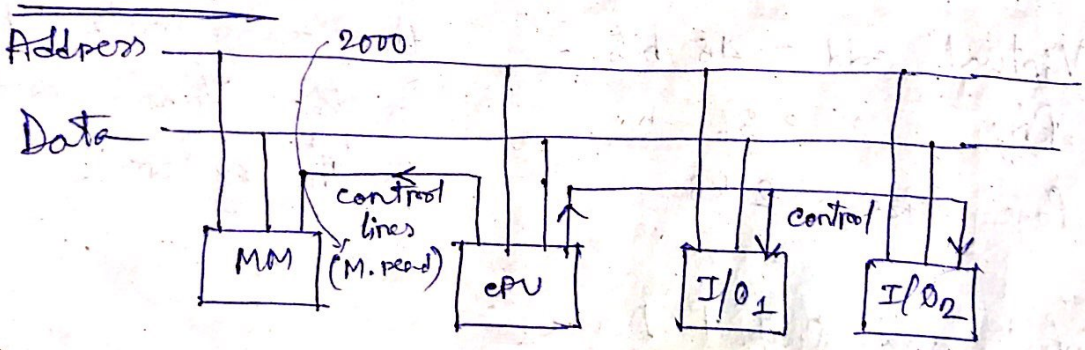
↓
I/O device enters the input as Ascii format.



monitors the address bus. This interface does detect its own address & enable the path b/w CPU & I/O 1.

- Isolated I/O vs. memory mapped I/O :-

» Isolated I/O :-

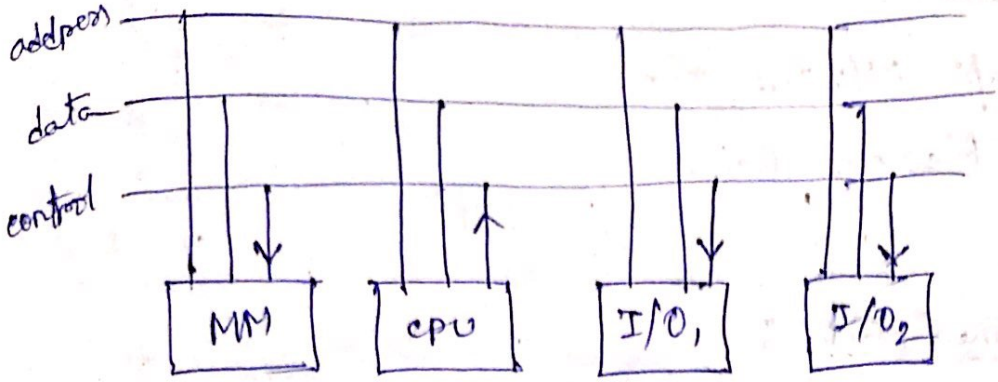


2 - control lines →

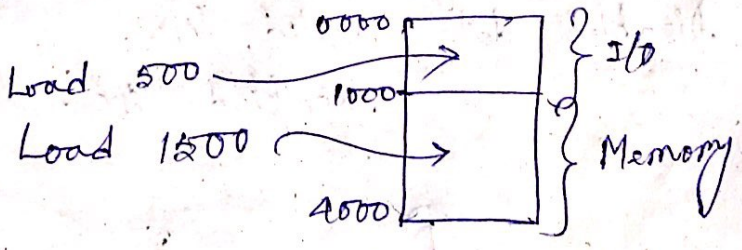
1) Memory Read — LOAD 2000
2) " Write

» Separate instruction will be there for memory operation & I/O operation.

» Memory Mapped I/O :-



Isolated I/O	Memory mapped I/O
1) different control line.	1) same control line.
2) ~~diff~~ instructions will be used for MM & I/O device.	2) same instruction.
3) diff address space.	3) common address space.

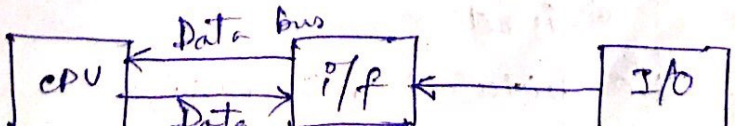


(*) In the control line, CPU specifies the I/O command

1) control ⟶ (memory read/write)

2) Status ⟶ checks if I/O device is ready or not. (for data transfer)

3) data input ⟶ if I/O device is ready CPU activate the data-input line.



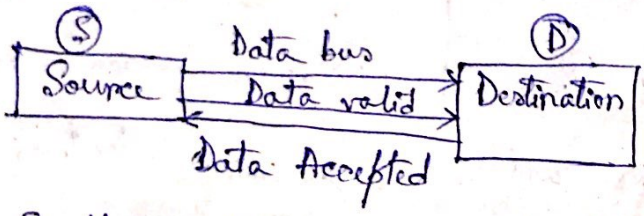
21/03/18

- **Modes of Data Transfer :-**

1) Programmed I/O
2) Interrupt initiated I/O
3) Direct Memory Access (DMA)

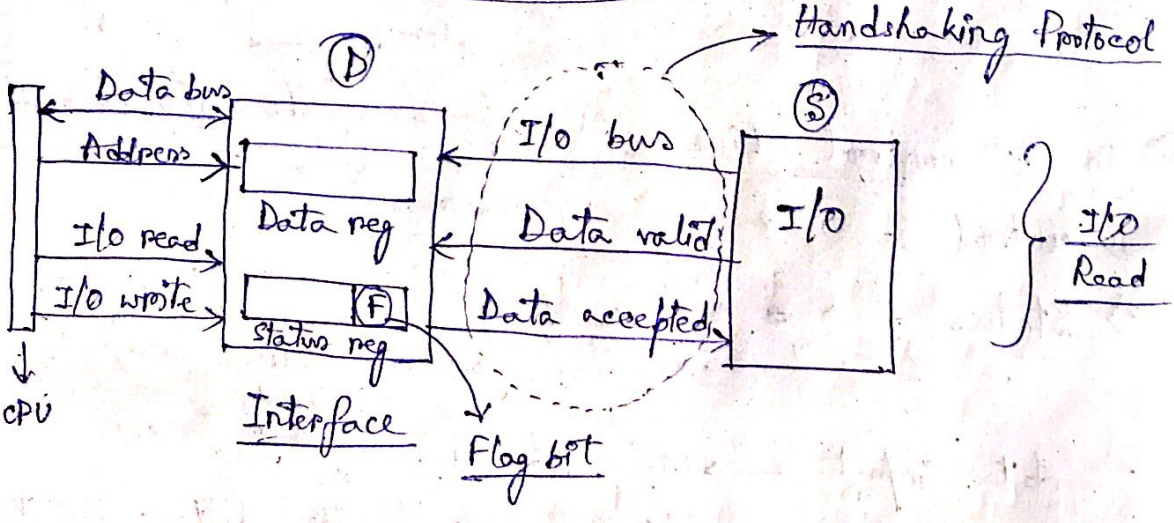
① Programmed I/O :-

» **Handshaking Protocol :**



S place data in data bus. It enables data valid line. When destination accepts the data valid line it ~~acknowledges~~ enables data accepted line (by D). Disables data valid line.

" " accepted line.



» f = 1 ⟶ Data present in Data register

» f = 0 ⟶ " is not " " " "
I/O device is not ready.

if (F == 1)

CPU accepts data from data reg.

Then data accepted line is disabled by interface.

» Drawback :-

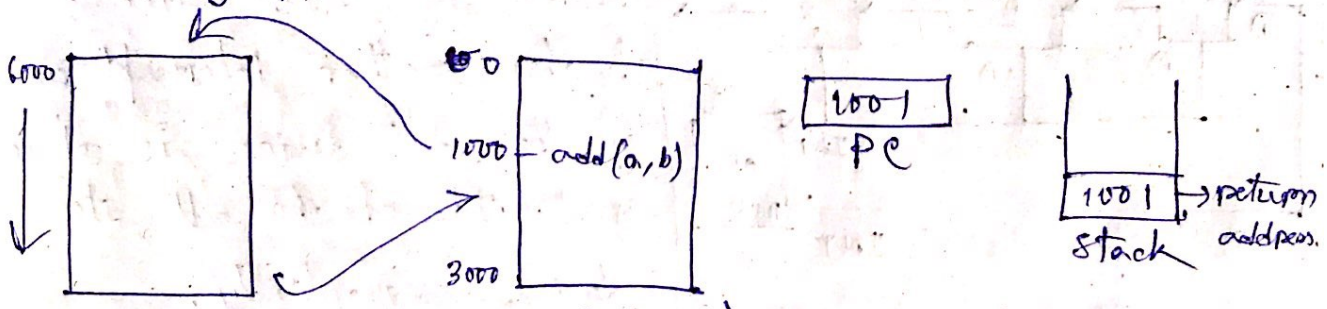
CPU has to continuously check flag bit whether F=0 or F=1. So, normal execution of CPU operation will be halted.

② Interrupt initiated I/O :-

To overcome the drawback of Programmed I/O.

i) CPU initiates I/O operation by placing address of I/O device in address bus.

A program is a set of instructions. Each instruction access memory address. Suppose at memory location 1000 → add(a,b) is a instruction → interruption occurs (means it gives signal to CPU that I/O device is ready for data transfer).



ISR (Interrupt Service Routine)

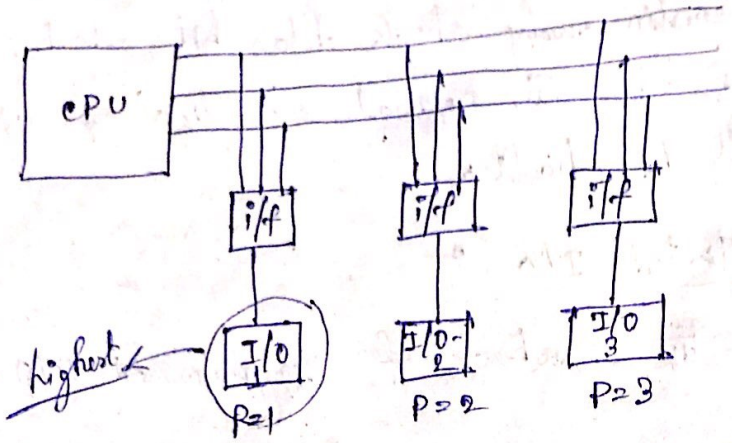
↓

This program contains what to do when interruption occurs.

» How CPU gets interrupt service routine :-

» Priority interrupt :-

Suppose at the same time interruption occurs of two or more I/O device.

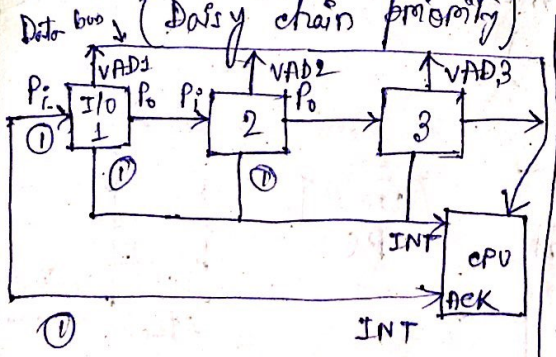


Here, I/O device with the highest priority will be serviced first.

» Identify highest priority → 2 ways →

1) using programming → Polling → software Technique

2) " hardware technique ↓



it is a program. It checks all I/O device & checks whether interrupt signal of the I/O device is set or not. It checks all I/O devices sequentially.

VAD → Vector Address

CPU accepts the interrupt by enabling Interrupt acknowledgement line.

Pi = 1
Po = 0 } block the signal to next device.

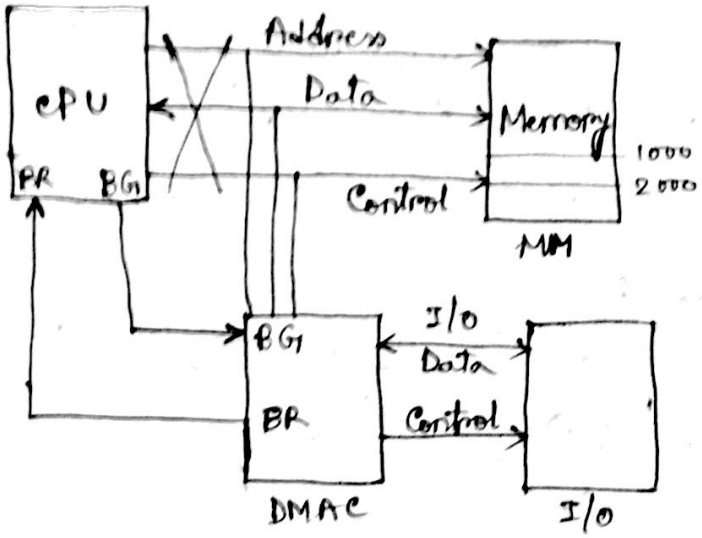
↳ Po = 1 → means I/O-1 passes the signal to I/O-2.

Prog. I/O and ~~DMA~~ interrupt initiated I/O requires CPU interaction.

But in case of DMA direct transfer of data from I/O device & memory.

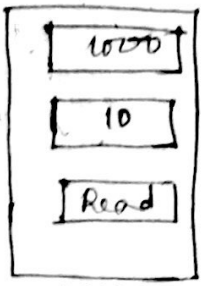
~~29/08/18~~

- DMA controller :



CPU place.

1) Starting Address of block of data to be read or written in memory.



Address register (present in DMAC)

2) no. of words to be transferred (eg: 10 words → 10 memory addresses)

3) control to specify read/write operation.

4) " " start the DMA operation.

BR ⟶ Bus request

BG ⟶ " Grant

Advantage : No CPU interaction req^d, DMAC can ~~control~~ transfer data directly b/w I/O & MM.

» 2 types of Transfer :-

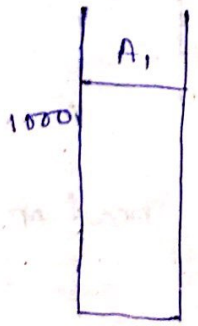
1) Burst Transfer :- (faster).

Transfer block of data at a time.
(consists of memory location).

2) Cycle Stealing :-

One word is Transferred at a time.

- Pipeline :-



A1
A2
A3

Answer scripts

Examiner

E1 E2 E3

time	E1	E2	E3
1	A1	—	—
2	A2	A1	—
3	A3	A2	A1 ⟶ 3
4	A4	A3	A2 ⟶ 4

(faster)

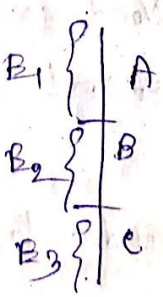
But for only E1

A1 ⟶ 3 time

A2 ⟶ 3+3 = 6 time

A3 ⟶ 9

A4 ⟶ 12 ... A10 ⟶ 30



A1 ⟶ 3
A2 ⟶ 4
A3 ⟶ 5
A4 ⟶ 6
A5 ⟶ 7
A10 ⟶ 12

For pipeline, total time unit is reduced to check to answer script

" " " " " with pipeline.

$= \frac{30}{12}$

no. of tasks = n (10)

n n segments = k (3) → examiner

clock period = t_p (1)

Time reqd without pipeline = $nk.t_p$

$\therefore$ speed up = $\frac{nkt_p}{kt_p + (n-1)t_p}$

$= \boxed{\frac{nk}{k+n-1}} \simeq \frac{nk}{n} = \boxed{k}$ [when $n \to \infty$, $n \gg k-1$].

$\Rightarrow t_p = 20$ ns, $k = 4$, $n = 100$ tasks.

speed up = ?

speed up = $\frac{100 \times 4}{4 + 100 - 1} = \frac{400}{103} = 3.88 \simeq 4$

26/03/18

- **Data Pipeline :-**

$A_i * B_i + C_i$ $\quad i = 1, 2, \ldots, 7$

$i=1$, $A_1 * B_1 + C_1$

$i=2$ $A_2 * B_2 + C_2$

:

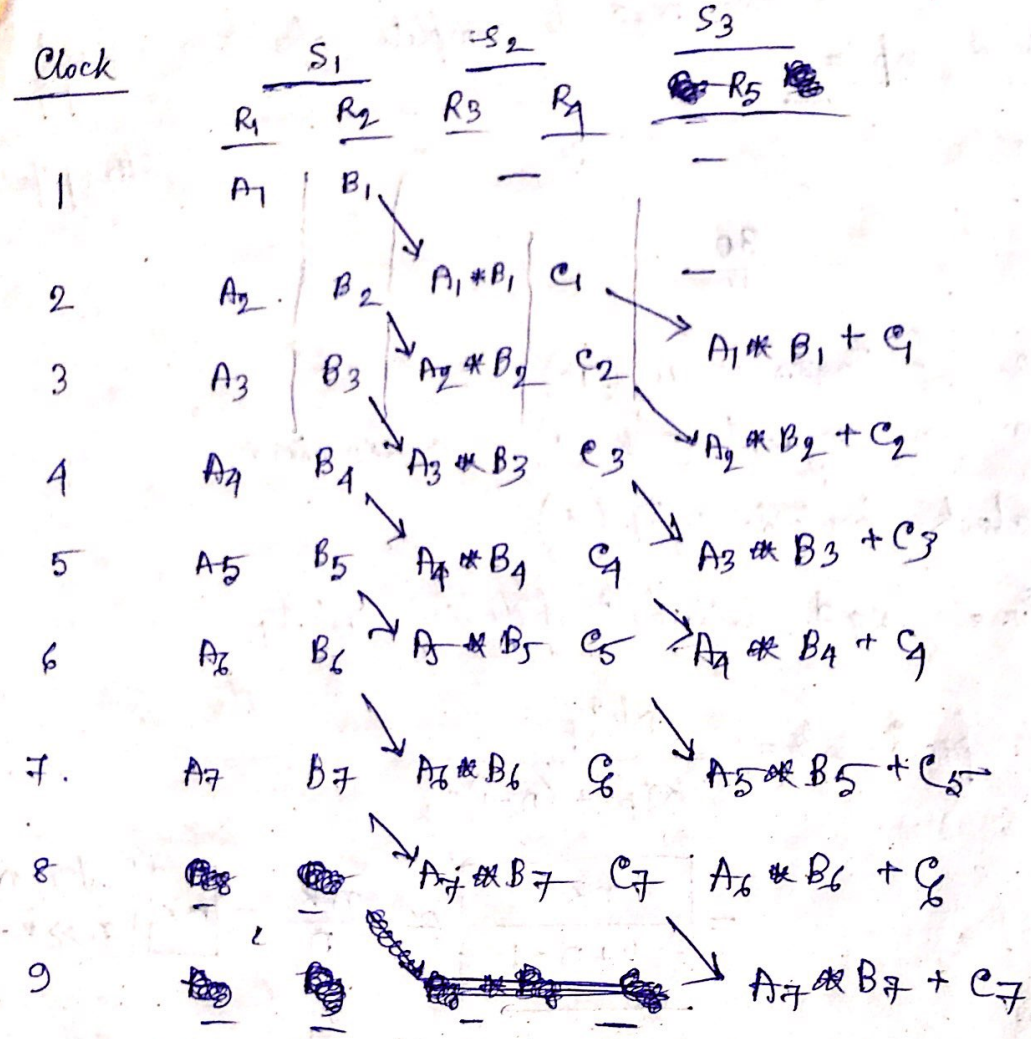
$i=7$, $A_7 * B_7 + C_7$

Segment - 1, S1 → 1st sub-operation

S_1 : $R_1 \leftarrow A_i$, $R_2 \leftarrow B_i$

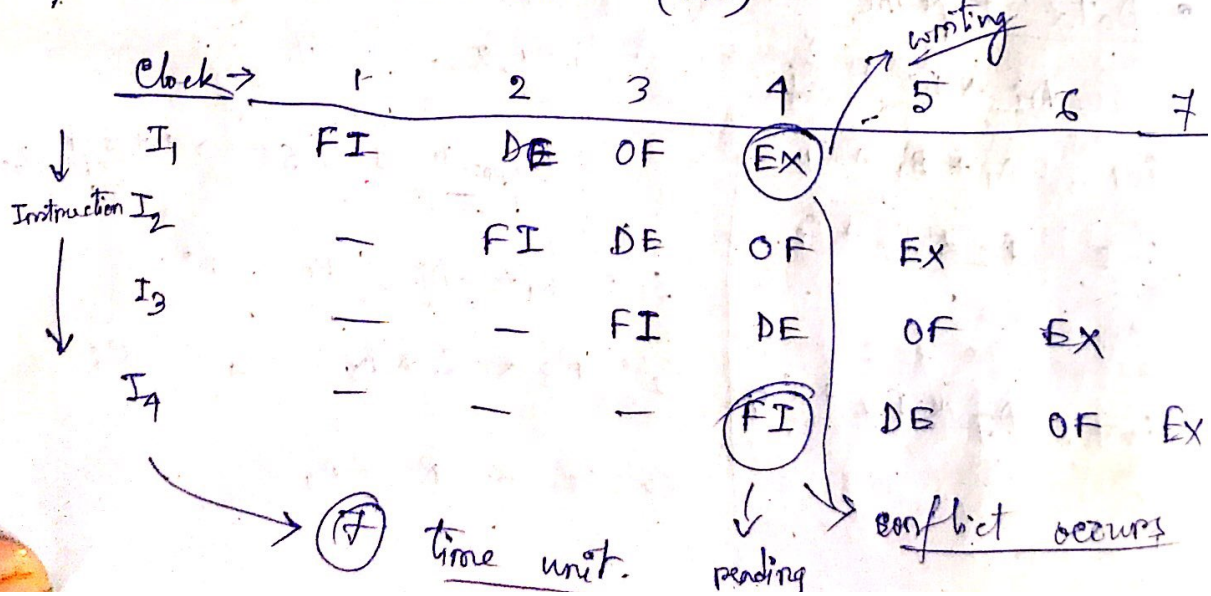
S_2 : $R_3 \leftarrow R_1 * R_2$, $R_4 \leftarrow C_i$

S_3 : $R_5 \leftarrow R_3 + R_4$



- **Instruction Execution Cycle :-** (Instruction Pipeline)

1) Fetch the instruction (FI)
2) Decode " " (DE)
3) Fetch " operands (OF)
4) Execute " instruction (EX)



$I_1 \rightarrow 4$, $I_2 \rightarrow 4+4$, $I_3 \rightarrow 4+4+4$, $I_4 \rightarrow 4+4+4+4$
$= 16$

(without pipeline)

Speed up $= \frac{16}{7} = 2.25$

⇒ No. of instruction, $n = 50$,
$k = 4$.

Sol^n :- speed up $= \frac{nk}{n+k-1} = \frac{200}{53} \simeq 4$.

Max^m speed up $= k = 3$.

- **Pipeline hazards** :-

3 types →

1) Structural hazards (Resource conflict)
2) Data
3) Control

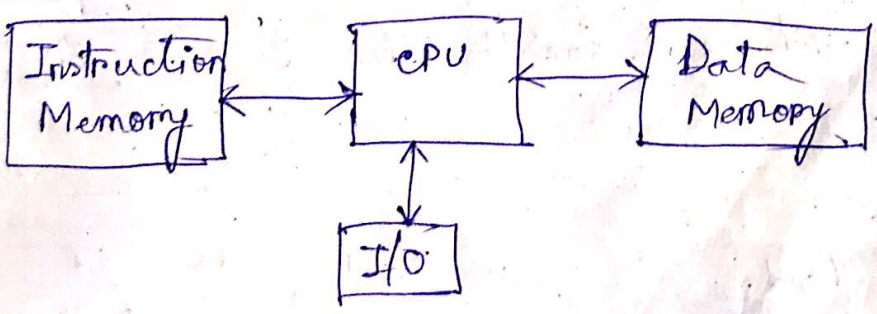
(when more than 1 instruction use the same resource at same time.)

→ To resolve → (use diff block)

Data memory

Instruction memory

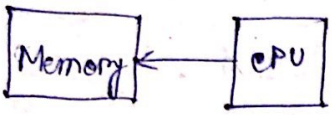
- **Harvard Architecture** :- (to resolve str. hazard)



28/03/18

Von - Neumann

1) Only one memory.
2) one bus. (1 set of bus)
3) slower.



Harvard

1) Instruction Memory, Data Memory
2) 2 buses. (2 set of buses)
3) faster.

- **Data Hazard :-**

Instructions

I :
J :

due to some haza
If J instruction is executed before completion of I, then data hazard occurs.

» 3 types of data hazard :-

(i) **Read After Write Hazard (RAW) :-**

I : $R_1 \leftarrow R_2^{10} + R_3^{5}$
J : $R_4 \leftarrow R_1 * 4$

Old value of R1 = 200

	1	2	3	4
I	FI	DE	OF	EX
			$R_2=10$	$R_1=15$
			$R_3=5$	
J		FI	DE	$R_1=200$ EX

$R_1 = 800$ → Wrong answer

first I writes some variables then J reads the variables.

⇒ **Solution :-**

1) Delayed Load. (right answer will come)

	1	2	3	4
I	FI	DE	OF	EX
			$R_2=10$	$R_1=15$
			$R_3=5$	
	NOP → increase the clock per			
J			FI	DE R

2) Compiler Optimization :- Compiler detects data hazard, after that it rearrange the instruction sequence.

I : $R_1 \leftarrow R_2 + R_3$
J : $R_4 \leftarrow R_1 * A$ } RAW
(K) : $R_5 \leftarrow R_6 + R_3$

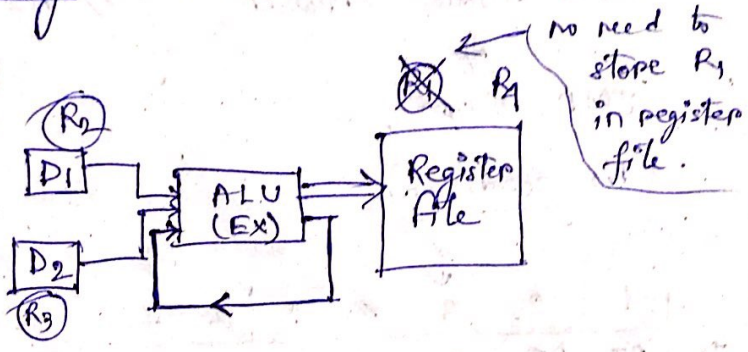
↓
Independent instruction

$\Rightarrow$

I : $R_1 \leftarrow R_2 + R_3$
K : $R_5 \leftarrow R_6 + R_3$
J : $R_4 \leftarrow R_1 * A$

3) Operand forwarding :- hardware detect the conflict.

I: $R_1 \leftarrow R_2 + R_3$
$R_4 \leftarrow R_1 * A$



(ii) Write After Read Hazard (WAR) :-

I : $R_1 \leftarrow (R_2) + R_3$
J : $(R_2) \leftarrow R_5 * 3$

[1st Read then write → no problem (no hazard)
But if write occurs before read then hazard occurs.]

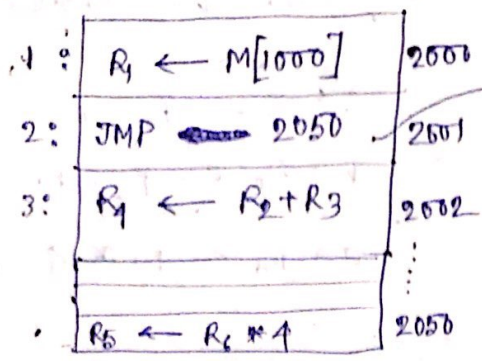
(iii) Write After Write Hazard (WAW) :-

I : $R_1 \leftarrow R_2 + R_3$
J : $R_1 \leftarrow R_4 - R_5$

[First I updates R_1 then J updates R_1.
But if J updates R_1 first before I updates R_1 then hazard occurs.]

(iv) No data hazard for RAR

• Control hazard : occurs in case of branch instruction



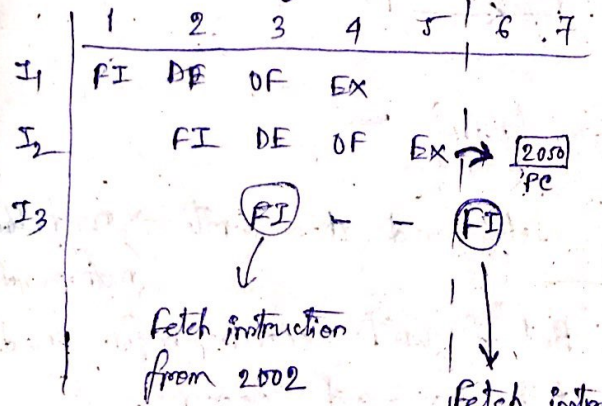
2 types of branch instructions →

1) Unconditional branch

JMP 2050
(No condition)

2) Conditional branch.

JNC 2050
If (C==0)
goto 2050
else
goto 2002.



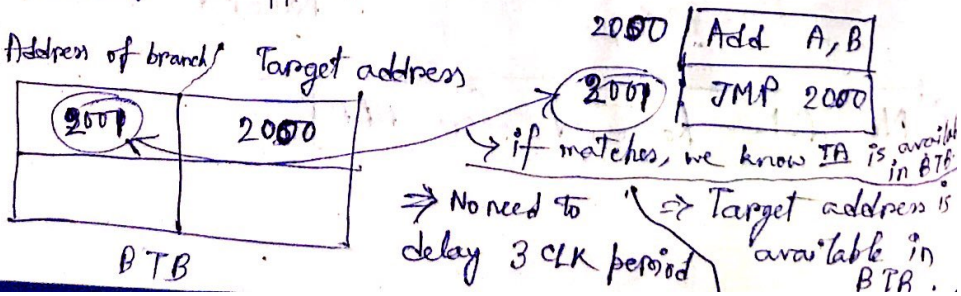
→ Only after knowing that 2nd instruction is branch instruction decoding will be halted after 3rd clk pulse.

(After knowing the PC address, it will fetch instruction from 2050)

» Solution

1) Delayed branch (6 - 3 = 3 CLK pulse delay)

2) Branch Target Buffer



→ if matches, we know IA is available in BTB

⇒ No need to delay 3 CLK period

⇒ Target address is available in BTB.

3) Branch Prediction :-

JNC 3000 <P>

P=1 → goto 3000

P=0 → next instruction

```
for (i=0; i<100; i++)
{   c ← A+B;
}
```

⇒ Always branch prediction is correct except (i=100).

⇓

miss prediction occurs.