# Computer organization and Architecture.

**Computer Architecture:-** conceptual design & fundamental operational structure of a computer is called computer Architecture.

**Computer Organization:-** It deals with physical devices and their interconnection with a perspective of improving the performance.

## Things Included:

**Computer Architecture.**
- ↳ CPU design
- ↳ Instructions
- ↳ Addressing modes
- ↳ Data formats

**Computer Organization:**
- ↳ I/O organization.
- ↳ Memory organization
- ↳ Performance
  - ↳ Instruction activity.
    - ↳ Pipeline.

## Data formats.

- Numbers
  - Unsigned
  - Signed.
    - Fixed point
      - ↳ Sign magnitude.
      - ↳ 1's complement
      - ↳ 2's complement
    - Floating Point
- Characters.
  - Binary representation of characters, known as data formats.
    - ↳ ASCII
      American standard code for Information Interchange
    - ↳ EBCDIC
      Extended binary coded decimal

Computer's components:-

↳ CPU → control unit
↳ ALU (Arithmetic logic unit)

↳ Memory → Primary or main memory
↳ Secondary or auxillary memory

↳ I/O devices → Input devices
↳ Output devices

Other components.

→ System bus.

→ CPU register.

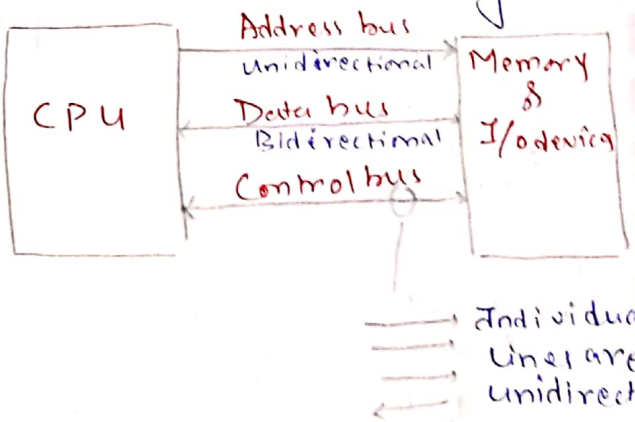
System bus:- ~~connection~~ collection of communication line within the computer system.

In the system we have three types of bus.

:- Address bus

→ Data bus

→ control bus.




CPU registers:- Small memories inside the CPU are called registers

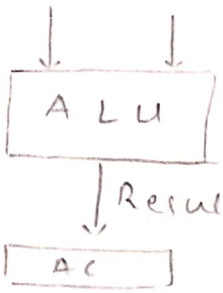
Types.

↳ General purpose registers (GPRs) (Stores general contents)

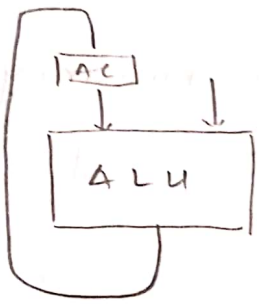
↳ Special purpose registers (Stores special contents)

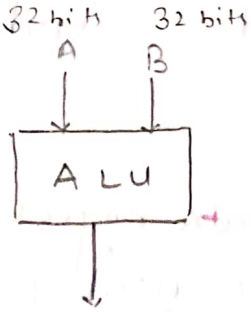
↳ Accumulator ↳ Status or flag register
↳ Program counter ↳ Address register
↳ Instruction Registers ↳ Data register
↳ Stack pointer

→ Accumulator :- It is used to store the result of ALU and sometimes one of the input operand for ALU also.




* If both the inputs are taken from Registers they are called "Register based architecture"
* If both the inputs should be taken from stack it is called "Stack based architecture
* If the input is coming from either Register or memory simultaneously it is called complex - Architecture
* If out of two inputs one is necessarily coming from accumulator it is called accumulator based architecture.



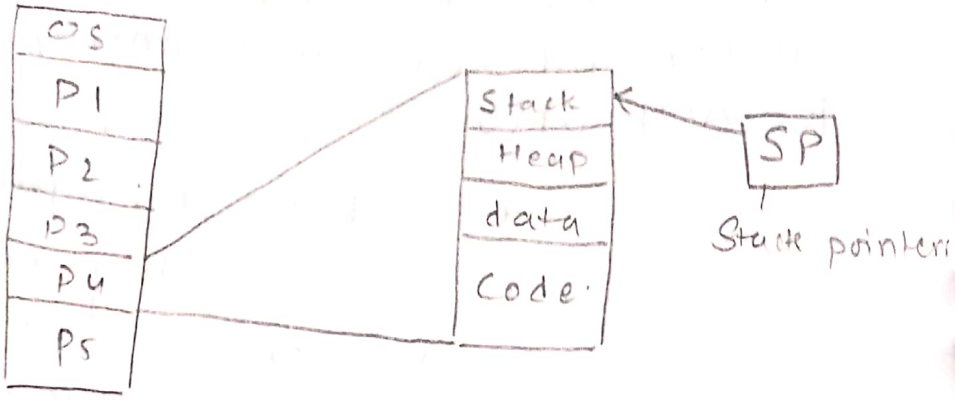




32 - bits architecture.
or
32 - bits CPU
↓
This will give 1word size.

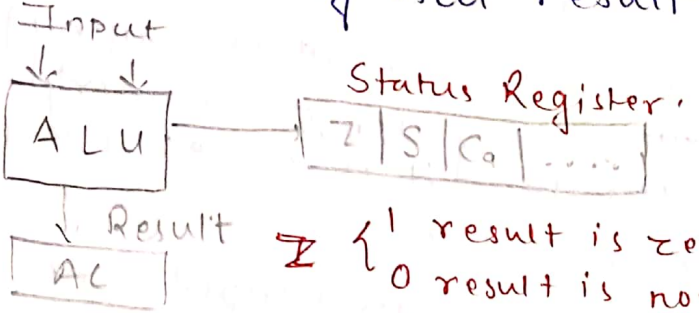
→ Program counter :- It stores the address of next instruction to be executed.

→ Instruction Register :- It is used to store the current Instruction which cpu executes; It is used to store the instruction before it goes for decode.

→ Stack pointer :- It is used to store the address of the top of the process stack.




→ Flag or Status register: It is used to store the status of ALU result



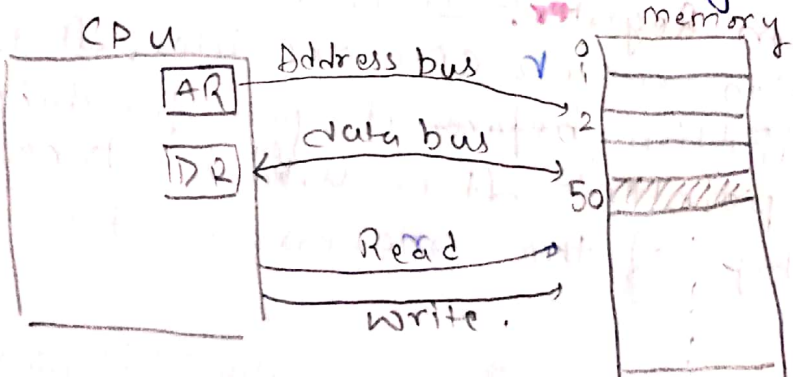

Z { 1 result is zero
  0 result is non-zero.

S { 0 +ive.
  1 -ive.

Ca { 0 no carry
  1 carry.

→ Memory Address Reg:- It is used to send address to memory (MAR)

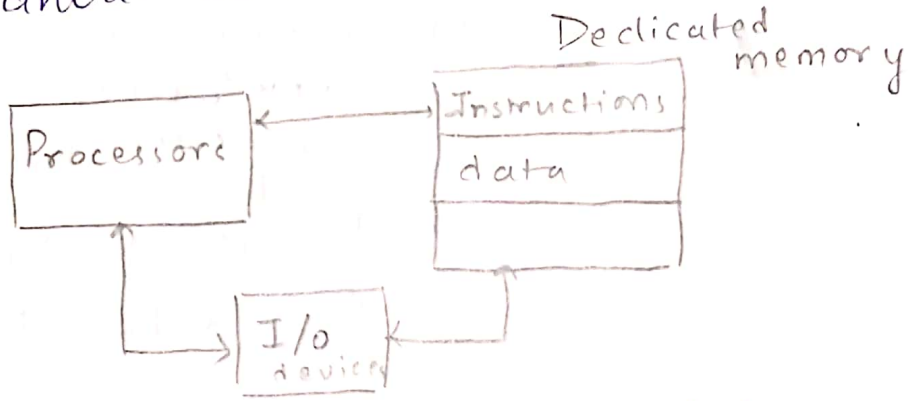
→ Data register or memory Data register or memory buffer register:- It is used to send data to memory (memory write) and to receive data from memory (memory read)



## * Von neumann architecture :-

There is a declicated memory connected to a processor in which Instructions and data are stored.

The processor can fetch the instruction & data from this memory, can execute the instructions using the data. & can copy the result in the memory without any manual work.

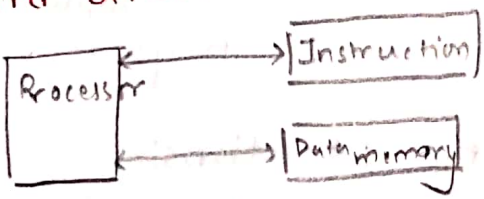



## Bottle neck of von neumann's model :-

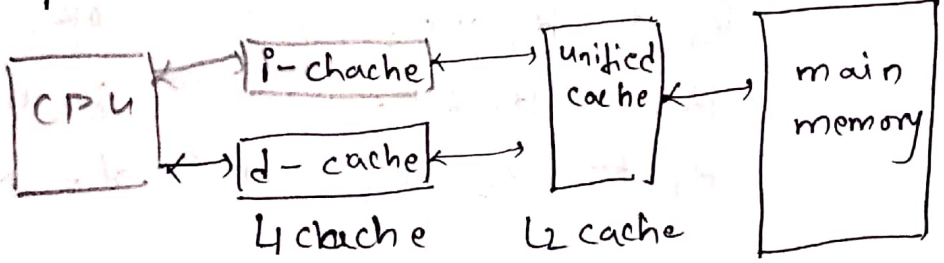
The instructions and data both cannot be fetched from memory simultaneously

Hence to solve this problem scientist gave another architecture, "Harvard Architecture"

## Harvard architecture.




## Improvised Harvard architecture




## Stored program architecture.

## ⇒ Micro-operation :-

The operations performed on the data stored in the registers

Types of microoperation.

↳ Register transfer.

↳ Register to register content transfer: $(R_1 \leftarrow R_2)$ or $(R_2 \rightarrow R_1)$

↳ Memory access.

↳ Read: Reg ← Memory

↳ Write | DR ← M[Address] (or) DR ← M[100H] | DR ← M[AR]

Memory ← Reg | (or) M[Address] ← DR

↳ Arithmetic micro-operation :-

$R_1 \leftarrow R_2 + R_3$ $\quad R_1 \leftarrow R_1 - 1$

$R_1 \leftarrow R_2 - R_3$ $\quad R_1 \leftarrow \overline{R_2}$ (1's complement)

$R_1 \leftarrow R_1 + 1$ $\quad R_1 \leftarrow \overline{R_2} + 1$ (2's complement)

$R_3 \leftarrow R_1 + \overline{R_2} + 1$ (Addition with 2's complement)

2's complement means its negative representation)

↳ Logical micro-operations:-

$R_1 \leftarrow R_2 \wedge R_3$ Logical AND

$R_1 \leftarrow R_2 \vee R_3$ " OR

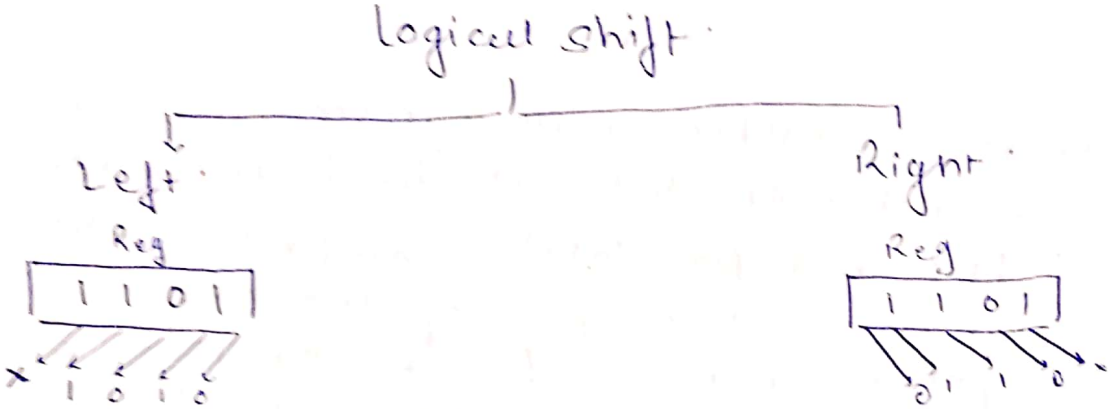
$R_1 \leftarrow R_2 \oplus R_3$ " X-OR

$R_1 \leftarrow \overline{R_2 \oplus R_3}$ " X-NOR.

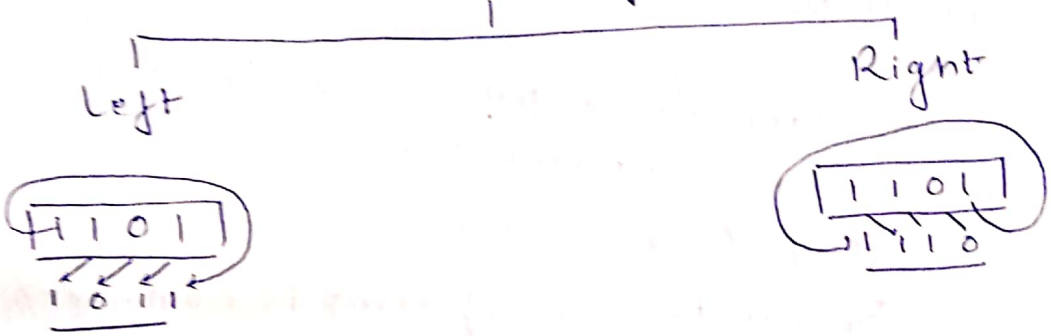
# Shift micro-operations

- Logical shift
- Circular shift (Rotation)
- Arithmetic

## Logical shift

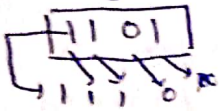



## Circular shift




Arithmetic shift: the shift operation is performed on arithmetic numbers (signed no.), after the shift operation sign of the no. should remain same.

## Arithmetic




Left :- It is same as logical left shift but it is performed only after CPU ensures that sign will not change after the operation.

eg:- 1 1 0 0 → 1 0 0 0

eg:- 1 0 1 0 → not-allowed → arithmatic-left shift overflow

to ensure the operation is allowed or not- cpu uses ⊕ xOR operation on the MSB

n-bits Reg

| $b_{n-1} \ldots b_1 b_0$ |
|---|

$b_{n-1} \oplus b_{n-2}$ → 0 allowed, 1 overflow

Question :- consider a new instruction i.e "branch-on-bit-set" (bbs), the instruction "bbs reg, pos, label" jumps to label if bit in position pos of the register operand reg is '1'. a register is 32 bits wide and bits are numbered 0 → 31, bit in position '0' been least significant. consider the following imulation of this instruction on a processor that does not have bbs implemented.

temp ← reg & mask

Branch to label if temp is non-zero

The variable temp is a temparary register. for correct imulation the variable "mask" must be generated by.

(a) mask ← 0x1 << pos

(b) mask ← 0xFFFFFFFF >> pos

(c) mask ← Pos

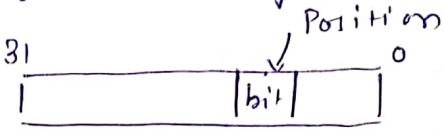
(D) mask ← 0xF

x<<y means shifing x in left direction y times.

bbs reg, pos, label.

reg → Register
pos → Position

temp ← reg & mask
Branch to label if temp is non zero




Temp : 0 0 0 . . 0 | 1 | . . 0

mask : 0 0 0 . . 0 | bit | . . 0

If mask ← 0 x F F F F F F F F >> Pos.

Pos = 3

Hence at mask ← 0 x 1 << Pos.

0 0 . . . . 0 0 1 | Pos = 0

0 0 . . . . 0 1 0 | Pos = 1

0 0 . . . . 1 0 0 | Pos = 2.

(0 3
P-42)

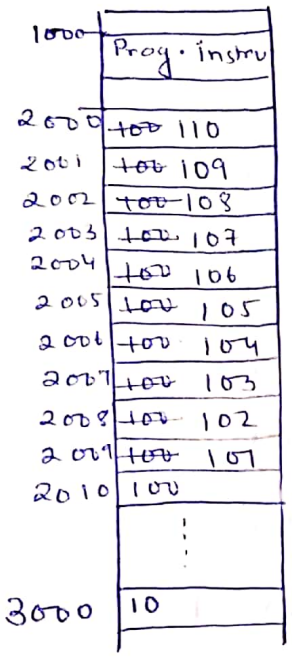
Question
Pg-42

loop: $R_1 \leftarrow M[3000]$
$R_2 \leftarrow M[R3]$
$R_2 \leftarrow R_1 + R_2$
$M[R_3] \leftarrow R_2$
$R_3 \leftarrow R_3 + 1$
$R_1 \leftarrow R_1 - 1$
Branch on nonzero
Halt

$R_1$ = 10, 9, 8, 7 ...

$R_2$ = 100+10, +9, 10 ...

$R_3$ = 2000, 2001, 2002 ...




Q3 memory references

$R_1 \leftarrow M[3000]$ — 1 times = 1 bit
$R_2 \leftarrow M[R_3]$ } — 10 times = 10×2 = 20 bit
$M[R_3] \leftarrow R_2$ .

20 + 1 bit = 21 bits.

Q4 content of memory location
2010 = "100"

Q5 If CPU receives interruption during the execution of any instruction then CPU completes the execution of current instruction, then CPU stores the address of next instruction onto the stack as return address and then CPU goes for interupt service.

1000 $R_1 \leftarrow M[3000]$
1007
1008 $R_2 \leftarrow M[R_3]$
1012 $R_2 \leftarrow R_1 + R_2$
1016 $M[R_3] \leftarrow R_2$
1020 $R_3 \leftarrow R_3 + 1$
1024 $R_1 \leftarrow R_1 - 1$
1028 Branch

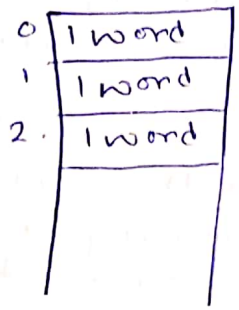
as byte addressable uses 4 blocks for 1 word.
hence the answer will be
1000 + 8 + 4 + 4 + 4 + 4
= 1024

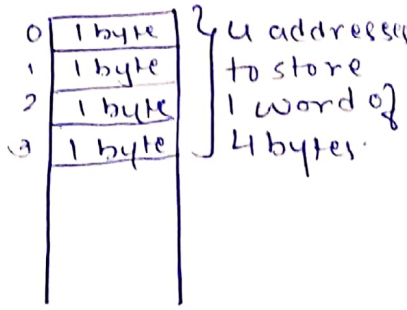
# Memory

Word addresable
(1 block consist of 1 word)

1 word = 32 bit
= 4 bytes

Byte addressable
(1 block consist of 1 byte)




---

Q1 P42.

1 word = 8 bit
= 1 byte

$A_0 \Rightarrow$ 00100011

| carry |
|---|
| $C_0$ |
| 1 |
| 1 |
| 0 |
| 0 |
| 0 |
| 1 |
| 0 |
| 0 |

$C_0$ 00010001
1 $C_0$ 0010000
11 $C_0$ 001000
011 $C_0$ 00100
0011 $C_0$ 0010
00011 $C_0$ 001
100011 $C_0$ 00
0100011 $C_0$ 0

B → 0xx3

C → 8x6xxxx0

Q1
(b) no. of 1 bits in $A_0$

Q2.
(a) RRCA, #1:
Right rotate A through carry by one bit.

Question 2007 GATE

In a simplified computer the instructions are:

OP Rj, Ri — Performs Rj OP Ri & stores the result in Register Ri
Rj OP Ri → Ri

OP m, Ri — Performs val OP Ri → Ri
[val denotes the content of memory location m]

MOV m, Ri — moves the content of memory location 'm' to register Ri

MOV Ri, m — moves the content of Ri to memory location m

The computer has only two registers and. 'OP' is either ~~add~~ ADD or SUB. Consider the following basic block.

$t_1 = a + b$
$t_2 = c + d$
$t_3 = e - t_2$
$t_4 = t_1 - t_3$

assume that all operands are initially in memory. The final value of computation should be in memory what is the minimum no. of MOV instruction in the CODE generated for the basic block.

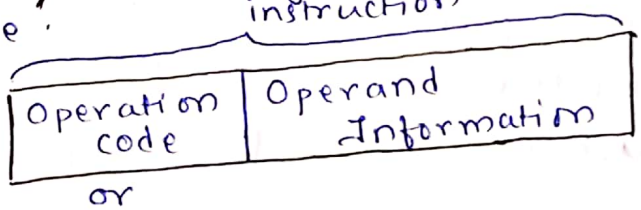
Answer :-

| | |
|---|---|
| MOV b, $R_1$ | $b \rightarrow R_1$ |
| ADD a, $R_1$ | $a + R_1 \rightarrow R_1$ |
| MOV D $R_2$ | $D \rightarrow R_2$ |
| ADD c, $R_2$ | $c + R_2 \rightarrow R_2$ |
| SUB e, $R_2$ | $e - R_2 \rightarrow R_2$ |
| SUB $R_1$, $R_2$ | $R_1 - R_2 \rightarrow R_2$ |
| MOV $R_2$ x | $R_2 \rightarrow x$ |

⟹ Instructions :-

definition: a group of bits which instructs computer to perform some operation.

Instructions are binary statements which is produced by compiler in a machine language or byte code or binary code.

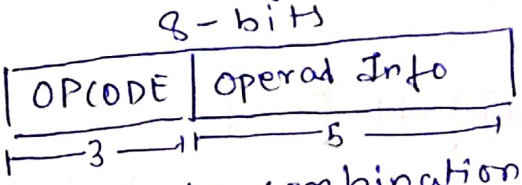



or

opcode.

eg:- For a system

8-bits




opcode combination = $2^3 = 8$

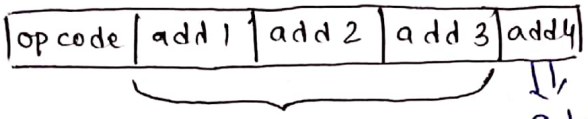
∴ Max no. of operations possible = 8

no. of instruction supported by the system (different - 2 type of instructions) = 8

If 12 opcode combination is required then bits of opcode will be = 4'

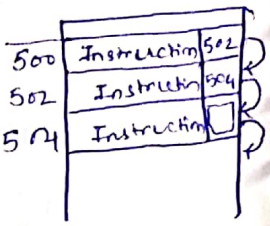
⇒ 4-address instruction:-

Within an instruction maximum 4 address can be specified




operands

address of next instruction.




This type of instructions were used in the type of CPU that doesnot have program counter register and hence the last address is used to point to next instruction. ~~These types~~ and the rest all 3 addresses could be memory or register operand.

-) Disadvantages

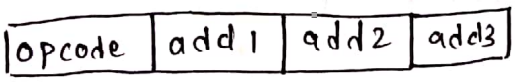
→ Large size instruction.

→ Due to this more memory is required to store the program.

→ If the program is relocated then the 4th address part of the instructions should be updated accordingly and updation of instruction is very costly operation.

⇒ 3-address Instruction.

maximum three addresses can be specified within an instruction.




Program counters used in such types of computers this stores the address of next instruction.

Hence the disadvantages of above cases are resolved and this terms into advantages of 3-address instructions.

- lesser size instruction
- no updation of addresses or relocation

ex:- 10101 10 11 01 $R_2 \leftarrow R_3 + R_1$

ADD $R_2$ $R_3$ $R_1$

* All ~~modern~~ modern days computer uses 3-address instruction.

-) 2-address instructions:

Maximum two addresses can be specified in an instruction

example :- 1010 | 10 11 $R_2 \leftarrow R_2 + R_3$

opcode $R_2$ $R_3$

one operand is used as source and destination both.

Disadvantages:

→ The value of common operand is updated by the result of the operation

→ More no: of instructions for a program as compaired to 3-address instruction format.

example :- $x = (A+B) * (C+D)$

3 address

$R_1 \leftarrow A + B$

$R_2 \leftarrow C + D$

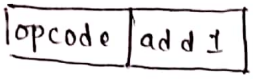
$x \leftarrow R_1 * R_2$.

2 address instruction:

- $R_1 \leftarrow A$
- $R_1 \leftarrow R_1 + B$
- $R_2 \leftarrow C$
- $R_2 \leftarrow R_2 + D$
- $R_1 \leftarrow R_1 * R_2$
- $x \leftarrow R1$

→ 1- address instruction

only one address can be specified in an instruction.




eg:- 10101 11 $AC \leftarrow AC + R_2$.

opcode $R_2$.

accumulator is used as 2nd operand implicitly (within the instruction we don't need to mention it)

disadvantages:

→ more no. of instructions for a program as compaired to two-address instruction format.

eg:- $x = (A+B) * (C+D)$

- $AC \leftarrow C$
- $AC \leftarrow AC + D$
- $R1 \leftarrow AC$
- $AC \leftarrow A$
- $AC \leftarrow AC + B$
- $AC \leftarrow AC * R1$
- $x \leftarrow AC$

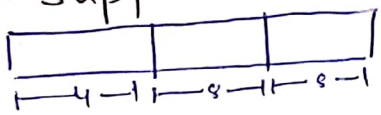
→ 0- address instruction

no any address is specified within an instruction

Such type of instruction are used in stack-based architecture.

eg:- 10101 2 operands are taken from stack

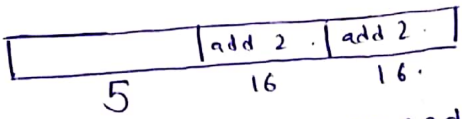
question a digital computer has 8-bit addresses and 20-bit instructions, minimum and maximum how many 2-address instructions the computer can support.

→ 


maximum instruction = $2^4 = 16$

minimum " = 1

question-a computer supports 25 2-address instructi if the address size is 16-bit then the size of an instruction in bits is?



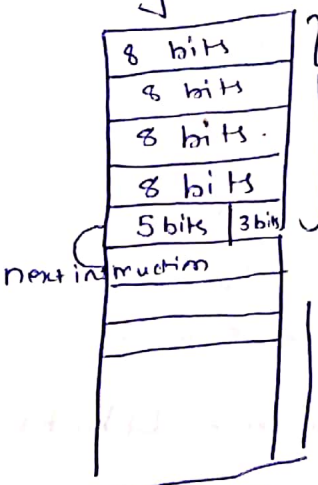

opcode bits = $\lceil \log_2 (\text{no. of instruction suported}) \rceil$

opcode bits = $\lceil \log_2 (25) \rceil = 5$

Size of instruction = $16 + 16 + 5 = 37$ bits.

→ In above question if the instructions are stored in the memory in byte align fashion then the amount of memory (in bytes) to store the program that consists 100 instructions is?

byte aligned fashion.




1 instruction will take 5 bytes.

⇒ 100 instruction will take 500 bytes of instruction

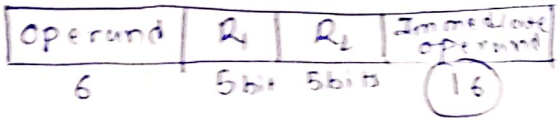
37 bits ≅ 5 byte

ISA :- Instruction set architecture collection of all the instructions supported by a processor (CPU) that collection is called Instruction set architecture

Question 17:- Page 44

no. of instructions = 40.
opcode - bits = 6-bits.

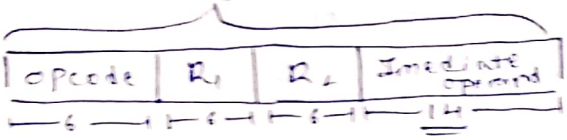
we have 24 registers hence to store these many registers we will require 5 bits ($R_0$ — $R_{23}$) i.e 5 bit numbering is required to store 24 Registers




Question 15 Page 44

32 bit architecture means ALU take 32 bit at once
and 1 word = 32 bits
1 instruction length = 1 words

no. of register bits (operand bits) = 64 = 6-bits
opcode bits = 6-bits.

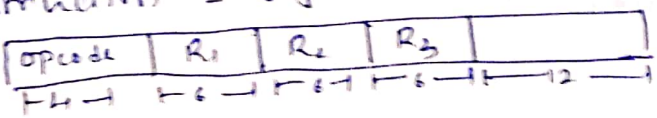



14 bits $\Rightarrow$ $2^{14}-1$ values = 16383 values

Question 18 Page 44

given:- 64 registers.
register reference = 6.
Instruction set = 12.
i.e 12 instructions are possible
$\therefore$ instruction bit = 4 bits

Instruction = 5 fields

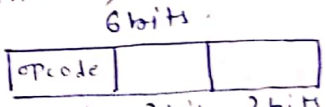



= 34 bits
= 5 bytes for an instruction

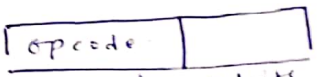
5 bytes for 1 instruction
500 bytes for 100 instruction

Question a digital computer has 6 bit instruction and 2 bit addresses, it supports 1 address & 2 address instructions both. If there are 3 : 2 address instructions then how many '1' address instructions can be formulated ✓(minimum & maximum)

→

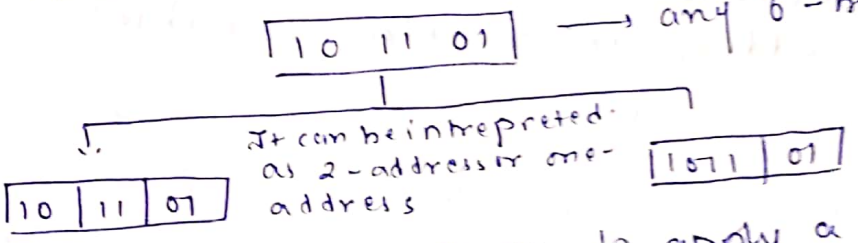



⇒ 4 instructions maximum




⇒ 16 instructions maximum
1 instruction minimum

Considering an instruction comes to CPU for execution




~~maxim~~ hence we have to apply a rule when the interpreter will go ~~for~~ one-address instruction or two address instruction. hence we have to apply some restriction on lower bit opcode.

total no. of instructions possible for (2 address) = 2bit = $2^2$ = 4 (00, 01, 10, 11)

total no. of used instructions ≥ 3 (00, 01, 10)

total unused instruction = 1 (11) ← assumed

Hence compiler will check for the first two bits and will go for one address instructions when it encounters '11'

Hence possible 1 address instruction = 1100, 1101, 1110, 1111 / 4

Hence possible 1 address instructions = (4) maximum

minimum = 1

| 2-address instructions | unused opcodes | 1-address instructions |
|---|---|---|
| 4 | 0 | $0 * 2^2 = 0$ |
| 3 | 1 | $1 * 2^2 = 4$ |
| 2 | 2 | $2 * 2^2 = 8$ |
| 1 | 3 | $3 * 2^2 = 12$. |
| 0 | 4 | $4 * 2^2 = 16$ |

⇒ only 2-address instruction supported.

⇒ only 1-address instruction suported.

question a computer has 32 bit instructions & 12 bit addresses if there are 254 2-address instructions then maximum how many 1-address instructions possible.

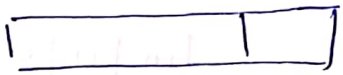
2 address


max opcodes = $2^8 = 256$.

used opcodes = : = 254

unused opcodes = 2.

32.




max = $2 * 2^{12} = 2^{13}$

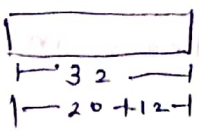
= 8192.

If ~~3 types of~~ there are 8000 1 address instructions are used then maximum how many 0-address are possible.

Max 1-address possible used = 8192.

= 8000

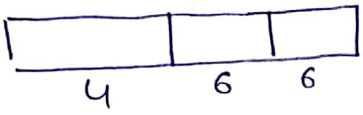
192.

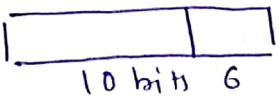



⇒ 192 $\times 2^{12}$ = maximum possible 0-address instructions

question 16 P 44. length of instructions = 16 bit.
address length = 6 bit.

Supports 2 address & 1 address




4bits 6bits.

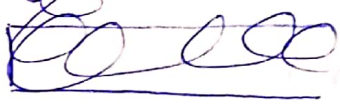
Unused

$(2^4 - n) \times 2^6$ bits

max opcodes = $2^4$.
used opcodes = n
unused = $2^4 - n$

gate A processor has 16 integer register. ($R_0, R_1 \ldots R_{15}$) and 64 floating point registers ($F_0, F_1, \ldots F_{63}$) it uses a two byte instruction format. There are 4 categaries of instruction: type 1, type 2, type 3 & type 4. Type 1 category consit of 4 instructions each with three integer register operands (3Rs). Type 2. category consist of 8 instructions each with 2 floating point register operands. (2Fs). Type 3 category consists of 14 instructions each with 1 integer register operand & 1 floating point register operand (1R + 1F). Type 4 category consist of 'N' no. of instructions, each with a floating point register operand.
The maximum value of N is?

⇒ Type 1



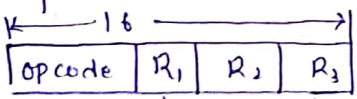
$(R_0, R_1 \ldots R_{15}) \Rightarrow$ Integer = 4 bits

$(F_0, F_1 \ldots F_{63}) \Rightarrow F = 6$-bits.

2 bytes instruction

| Type 1 | Type 2 | Type 3 | Type 4 |
|---|---|---|---|
| 4 | 8 | 14 | N1 |
| 3R | 2F | (1R + 1F) | (1F) |

Type 1

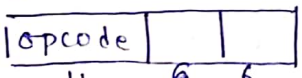



max = $2^4$ = 16

used = = 4

unused = 12

Type 2.

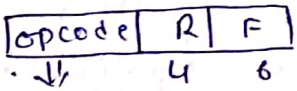



max = $12 \times 2^0 = 12$.

used = = 8.

unused = 4.

Type 3




6bits

4 + 2

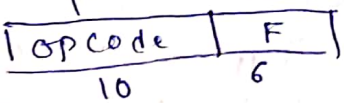
unused

max = $4 * 2^2 = 16$

used = 14

unused = 2.

Type 4.



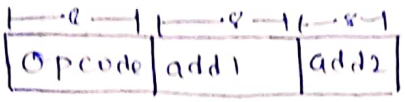

6 + 4

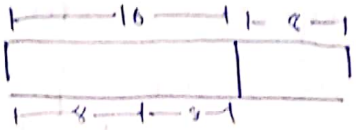
unused

$\rightarrow 2 * 2^4 = \underline{32} = N$

question a compwer supports 3 byte instructions and 8 bit addresses there are maximum 1024 '1' address instructions. then how many 2 address instructions are supported by the system?

→ Assume n 2-add instruction supported




max = $2^8$ = 256.

used = = n.

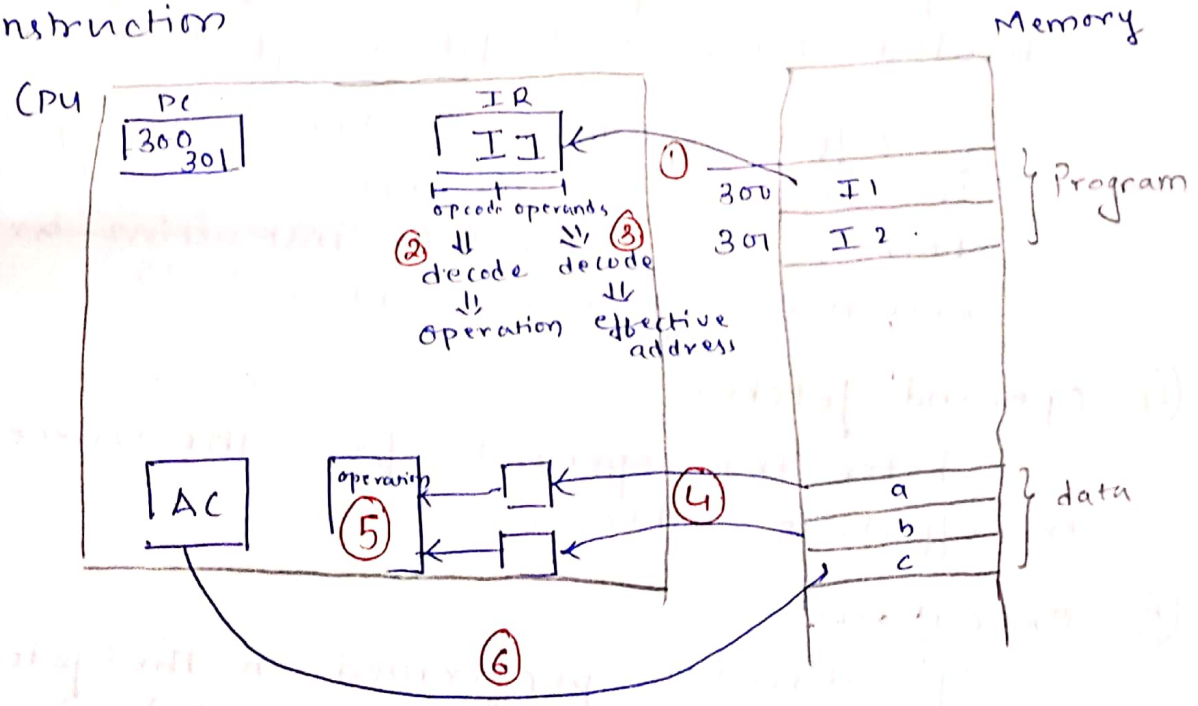
unused = 256 - n.

$(256-n) \times 2^8 = 1024$.

$n = 252$

## * Instruction cycle :-

System performs 6 phases to execute an instruction




1. Instruction Fetch
2. Instruction decode
3. Effective add calculation
4. Operand fetch

① Instruction fetch

Using 'PC' value next instruction is copied from memory to 'IR' Instruction register. during Instruction fetch only the value of 'PC' is incremented by the size of instruction.

② Instruction decode:

opcode part of instruction is decoded by CPU to obtain operation.

③ Effective address calculation

Operand Information part of instruction is decoded by CPU to obtain effective address

⇒ effective address :- Address of operand in computation type instruction & target address in branch type instruction are known as effective address.

④ Operand fetch.

Fetch the operands from the source using the effective addresses.

⑤ execution:

Operation is performed on the fetched operand and result is generated.

⑥ Write back

The generated result is copied back to the destination

Note: all type of instructions donot require all '6' phases

⇒ Addressing modes:-

It specifies how and from where the operand is obtained using the address field of instruction. They are used in (3rd) step of instruction cycle i.e effective address calculation phase.

I add instruction

| opcode | add I |
|---|---|

⇓

011 = 3

· add I could be anything.

· #3 (no. three)

R3 (Register no. 3)

M[3] (Memory address 3)

therefore to satisfy ambiguities the mode is introduced.

| opcode | mode I | add I |
|---|---|---|

00 ⇒ operand

01 → Reg.

10 → Mem. addres

2add instruction

| opcode | add 1 | mode 1 | add 2 |
|---|---|---|---|

| opcod | mod 1 | add 1 | mod 2 | add 2 |
|---|---|---|---|---|

→ Types of addressing modes.

→ Implied mode: The opcode definition itself specifies the operand hence no need to specify any explicit address for the operand.

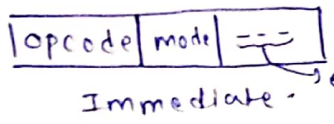
| opcode | mode | add |
|---|---|---|

INC A → INCA ⇒ Increament Accumulator

There is no need to specify the address of special registers explicitly hence the operation's opcode will ~~no~~ specify it

→ Immediate mode:- The address part of instruction specifies operand value

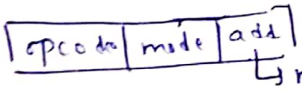
This mode is used to intialize register with constant value.




directly value is mentiond instead of address.

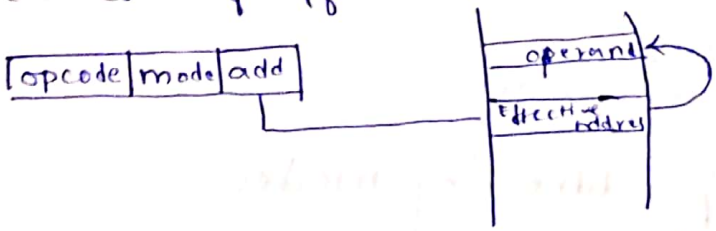
→ Direct mode: (absolute mode):

The address field of instruction specifies the effective address




→ Indirect mode:

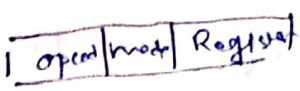
The address field of instruction specifies address of effective address




This mode is used to implement pointers

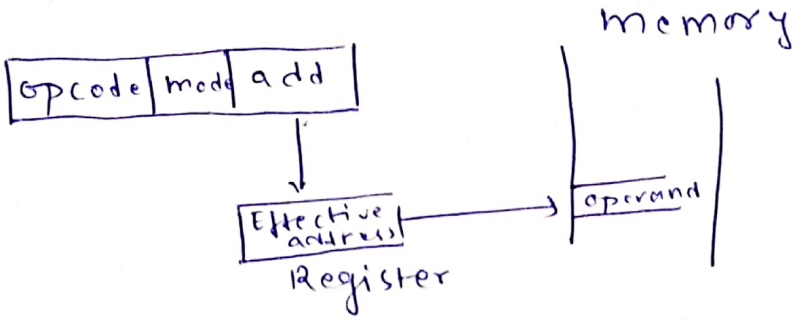
→ Register mode:

address field of instructions specifies a register which holds operand.



→ Register indirect mode :-

Address field of instruction specifies a register which holds effective address



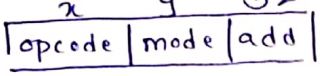

This mode is used to shorten the instruction length.

consider a system with 4 GB Ram & 64 Regs

Mem add = 32-bits

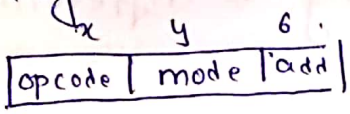
Reg. reference 6-bits

⇒ Direct mode

① 


length of instruction = x+y+32

② Register indirect




Length of instruction x+y+6.

time to get operand

① = 1 mem. access time.

② = 1 Reg access time + 1 mem access time

≃ 1 mem access time

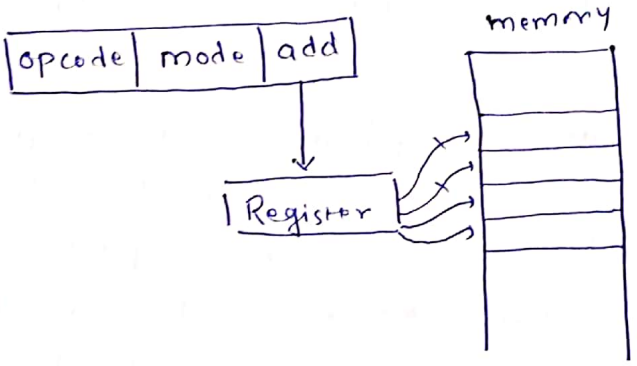
So within same amount of time Register Indirect mode gives better performance by reducing the length of instruction.

~~Auto~~/

→ Auto increment / Auto decrement mode:-

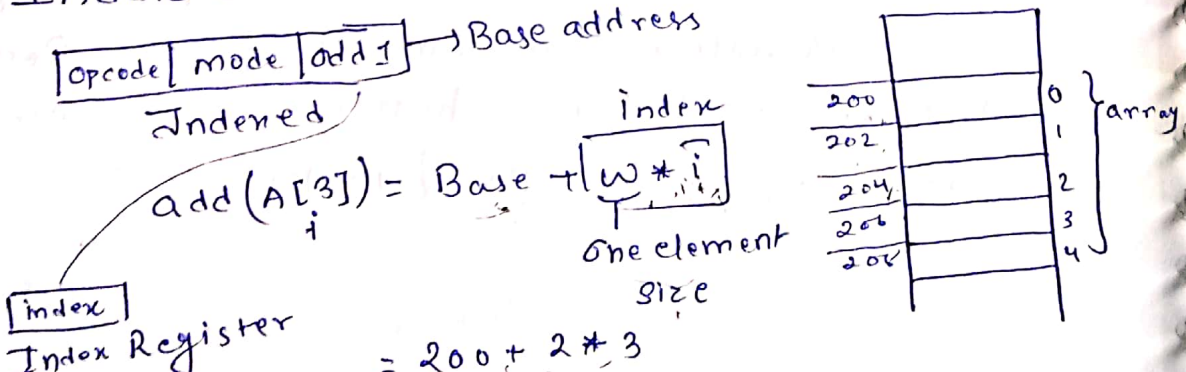
It is a varient of Register indirect mode in which the content of Register (Effective address) is automatically incremented or decremented. This mode is use to access a table of content (array) sequencially

Autoincreament → post increament
Auto decreament → pre decrement.




Here same instruction is used 'n' no. of times into a loop instead of creating n different instructions and storing it.

→ Indexed mode or Indexed Reg! mode:-




$= 200 + 2 * 3$

$= 206$.

effective address = add field of instruction (base) + index Reg value (index)

Address of element of array = Base + index

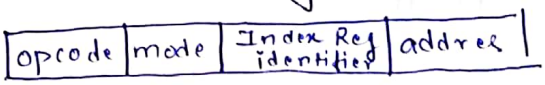
Instruction to calculate index is used before the instruction of access array element to calculate the index value for further usage

Index register is a special register which may or may not be available in CPU of different type.

Implementation

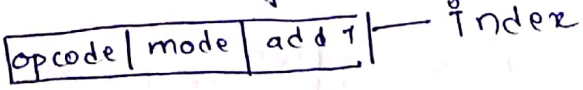
Index Register is Special purpose reg.
↓
index Register will be implicitly accessed

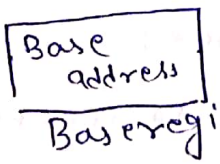
If index register is not available as special purpose register.
↓
any of the general purpose register will be used as Index register.




If the program data (array) is relocated then the new base address should be updated in the address part of the instruction but updation of instruction is a costly operation
hence to solve this we have another mode

→ Base register mode:




$$\text{effective address} = \frac{\text{add. field of instruction}}{\text{Index}} + \frac{\text{Base Reg value}}{\text{Base add.}}$$

Here the index is kept inside the instruction and the base address is stored in base register
Hence on relocation only value of register is updated that reduces the complexity of relocation

## Implementation

Base register as special purpose register.

⇓

Base Register is implicitly accessed.

Base register is not available as special purpose register.

⇓

any of the general purpose register will be used as Base register

| Opcode | mode | Base Reg. Identifier | add |
|---|---|---|---|

If the program data is relocated then the new base address should be updated in the base register hence no need to change the instruction or code.

Instruction to calculate the index value (w.r.t i) will be runned in compile time.

→ **Base + index register mode :-**

| opcode | mode | add. |
|---|---|---|

Base add → Base Register

Index → Index Reg.

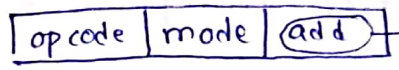
Base adderess and Index, both are stored into the registers. and effective address is passed to the instruction

Effective add = Base. Register value + index Reg. value

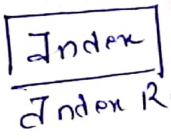
→ **Base + Index + displacement mode :-**

This mode is used to access array of records (structures)

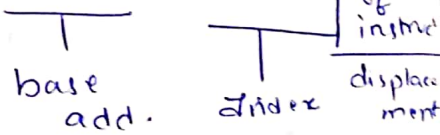
displacement field will be used to enter into specific element of structure.



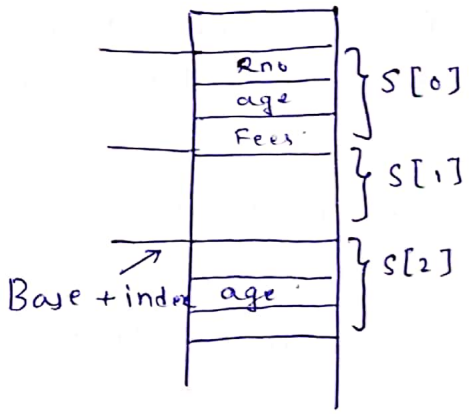





Effective add = Base Reg value + index Reg value + field value of instruction

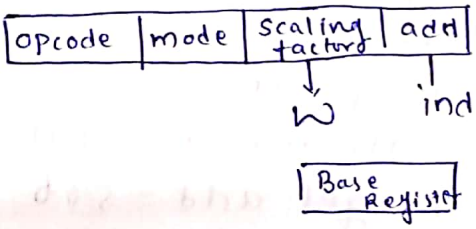



Structure
- Rno
- age
- fee.




Base + index will give address of s[2] but to access insid it displacement is going to use.

→ Scaled mode:




Here the instruction to calculate the index is not excluded externally but 'scaling factor' helps to get the value of 'w' and no. of instruction in Base or index mode is reduced.

This can be used with both Base & index register mode.

Effective addd = (Base Reg value) + (Scaling factor) * (add field value of inst?)

Base w i

## PC Relative mode (or position independent mode)

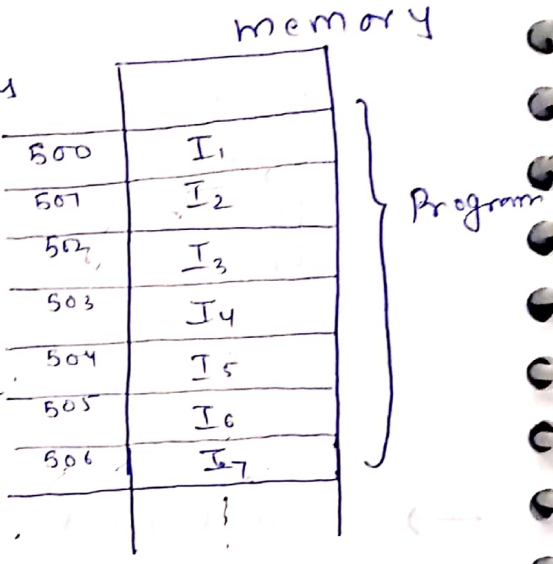
This mode is used for branch type of instructions

↳ Branch condition :-

Assume that CPU is currently executing instruction I2.

PC = 502.

$I_2$ is a branch instruction




condition = false

↓

Branch is not taken

↓

Next instruction => next in the sequence.

=> I3

No change in PC explicitly

condition = true

↓

Branch is taken.

↓

Next instruction => Target instruction (Assume it is I7)

=> Target add = 506

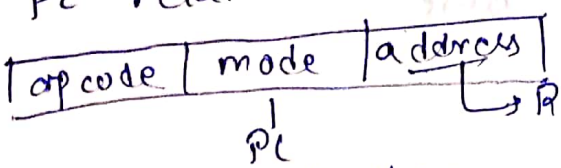
PC = Target add.

Target add := PC + Relative location to skip.

So to calculate the target address we use.

PC relative mode

(PC-value will give the value to jump in branch)




Effective add = PC value + add. field value of instruction.

When the branch instruction is encountered the PC-relative mode will be activated only when the condition is true and condition is checked in execution Phase

When the program is relocated the address part of instruction will not be affected as it will contain only the no. of skips. Hence it is also called (position independent instruction)

*Relative no. of skips is also known as 'offset'

→ Forward branching => offset => +ve value

→ Backward branching => offset => -ve value

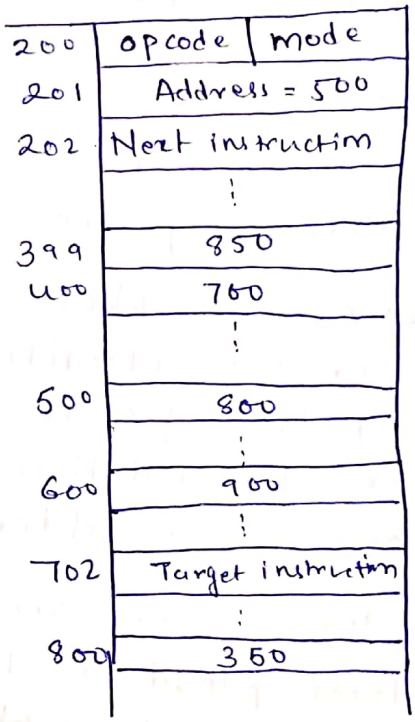
Example.

PC = ~~200~~ 202

Reg = ~~400~~ 399

XR = 100

XR = Index Reg.




memory.

PTO.

| mode. | Effective add. | Operand. |
|---|---|---|
| 1) Immediate mode | 201 | 500 |
| 2) Direct mode | 500 | 800 |
| 3) Indirect mode | 800 | 350 |
| 4) Reg. mode. | Register address is not available. | 400 |
| 5) Reg. Indirect mode. | 400 | 700 |
| 6) Auto decrement | 399 | 850 |
| 7) Index Reg mode | $500 + \frac{100}{\times 2} = 600$ | 900 |
| 8) PC - Relative | 202 + 500 = 700 | |

Register address is never taken.

For PC- Relative mode :-

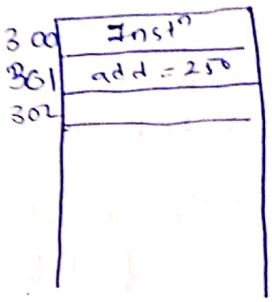
The target address is calculated in effective address phase.

and in execution phase the branch is checked for the condition and if condition is false 'pc' is given next instruction address but if condition is true then the 'pc' will be given address i.e. target address

question an instruction is stored at location 300 with its address field at location 301. the address field has the value 250. A processor register contains the no. 200. Evaluate the effective address if the addressing mode is

1) direct mode.
2) Immediate mode
3) Pc - relative
4) Register indirect

| mode | Effective add. | operand. |
|---|---|---|
| direct mode | 250 | |
| Immediate | 301 | |
| Pc - relative | 200 | |
| Reg-indirect | 302 + 250 = 552 | |

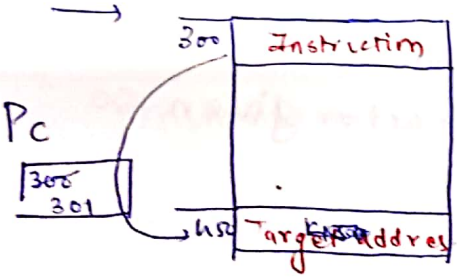



Question a relative branch mode type instruction is stored in memory at address 300. The branch is made to an address 450.

(91) What should be the value of relative address field of the instruction

2) Determine the value of program counter before instruction fetch, after instruction fetch & after execution of this instruction

→




1) Relative address Field.

Target add = PC + Relative add.

450 = 301 + Relative add.

Relative address = 149

2) Before instruction Fetch

PC = 300

After instruction Fetch

PC = 301

After Execution

PC = 450

# Instruction cycle.

- Fetch cycle.
  - Instruction fetch
- Execution cycle.
  - Instruction decode
  - Effective add. calculati
  - Operand Fetch
  - Execution
  - Write back.

Page 44.
Question 14

LW R1, 20(R2)

R1 ← Memory.

LW ⇒ Operation. (copy memory to $R_1$)

Effective add = 20 + R2.

LW → load word.

$R_2$ → General purpose register.

hence its not PC-relative.

→ There is no scaling factor given. so not scaling mode.

Hence 'D' is correct

Q-7 – Rom is Read only memory

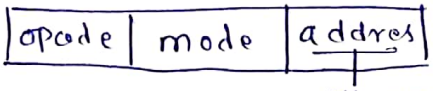
– PC points to next instruction

✓ Stack is LiFo.

– not all instruction affects the flag.

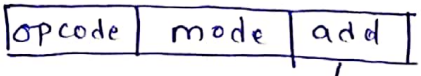
Q8 Pg 43. Implementation of Branch instruction

Position independent.




Target addr.

Position independent




relative location to skip.

This will give problem when there is a need to relocate program.

⇒ Relative mode is also known as Position independent mode.

Q9 In Pc relative mode relocation of Instruction takes place without any change in code

→ Base register mode relocation of Data takes place.

Hence Both are true here.

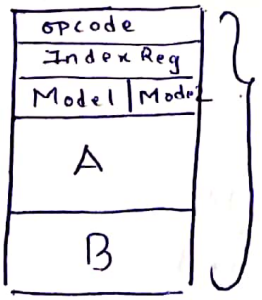
Q10

instruction ⇒ 3 words,

ADD A[R0] (@B) — Source Indirect. (1 write back, 2)

A[R0] → destination (after operation address will be stored here)

(@B) → Indexed.




3 word Instruction

Execution cycle does not include the instruction Fetch.

→ A & R0 are already in the CPU

So in A[R0] '1' memory cycle is accessed

@B is indirect mode then it will take '2' memory access

Total 1+3 = 4

Hence '3' memory access for effective address calculation and operation fetch & '1' memory access for write back.

Q12

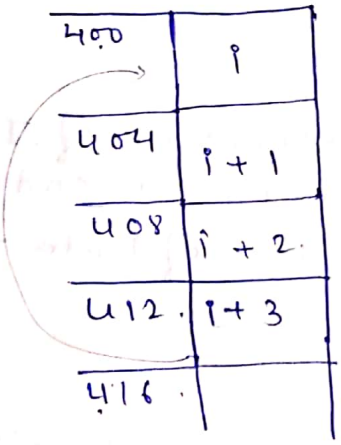
| opcode | add$_1$ | add$_2$ | add$_3$ |
|---|---|---|---|

An implied accumulator register is a Special register i.e we can use it explicitly through its address.

Q19. Instruction size = 4B

Offset = relative location. are stored in Bytes

out of all the instruction the instn. with offset is branch instruction.




CPU executing i+3.

hence PC = 416.

Target add = 400

PC + offset = 400

416 + offset = 400

offset = -16

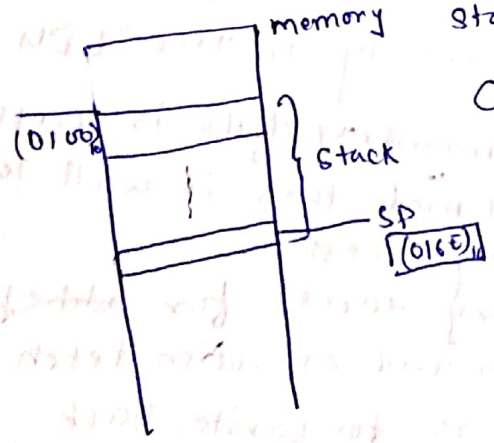
IF the target of the branch instruction is 'i'

that means target address = 400

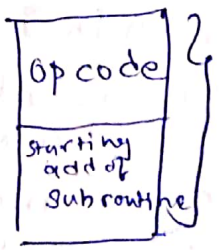
Q13. Byte addressable memory

PC, PSW. all register ar of 2 Bytes

memory stack grows upward.




CALL instruction → 2 words

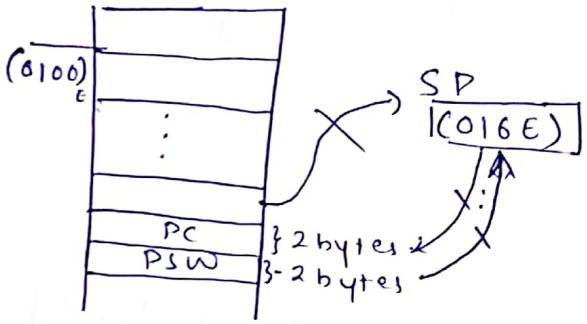



1 word = 2 byte

CALL
↳ store current PC value in stack
↳ " PSW value in stack
↳ load starting address of subroutine in PC

PC = $\boxed{(5FA0)_{16}}$ Before fetch of call instruction.

Hence $(5FA0)_{16}$ is address of call instruction




016E
+ 2
0170 After PC in stack
+ 2
0172 After PSW in stack.

SP points on the top of stack and call function Pushes the two value in stack.

question:- In the above question what will be the value of PC before fetch of call instruction, After Fetch of call instruction, and after the execution phase of call instruction.

Befor fetch ⇒ $(5FA0)_{16}$.

After Fetch ⇒ 5FA0 + 4 bytes = 5FA4

After Execution ⇒ Starting address of subroutine.

one instruction is 4 byte long.

---

## Branching

**Jump**

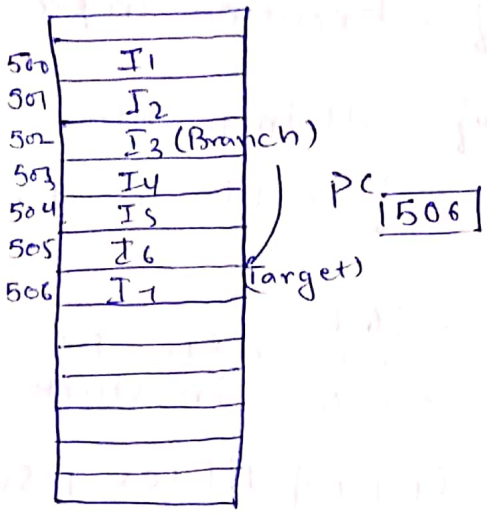
— PC value updated by Target

**Function call**

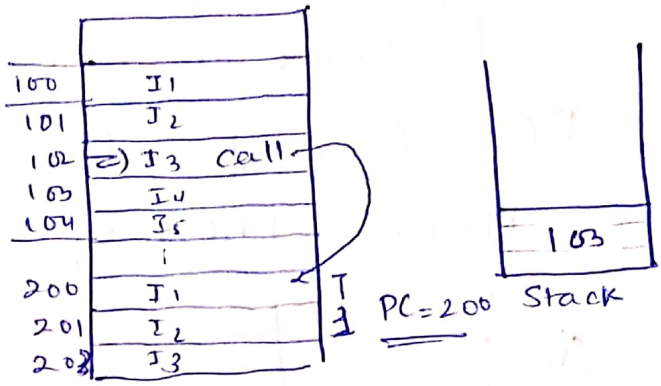
→ PC value + other regs are stored on stack

→ PC = target (starting add of function)

Jump.




Function call




Pag-43
Que 11

$R1 \longrightarrow M[100]$
$M[100] \longrightarrow R_2.$
$M[100] \longrightarrow R_3.$

$\Rightarrow$

$R_1 \longrightarrow M[100]$
$R_1 \longrightarrow R_2$
$R_1 \longrightarrow R_3.$

# ⟹ CPU Design ⟸

CPU cycle time:- The time in which cpu can perform one micro-operation.

CPI :- (cycles per instruction): No. of cycles required to execute an instruction

⇒ Execution time = no. of instruction * $CPI_{avg}$ * 1 CPU cycle tim

⇒ clock rate of CPU = 1/1 cycle time.

⇒ Execution time = $\frac{\text{no. of instruction} * CPI_{avg}}{\text{clock rate}}$.

| Instruction Type | No. of Instruction($n_i$) | CPI ($CPI_i$) | |
|---|---|---|---|
| Load & store | 30 | 6 | 180 |
| ALU | 40 | 5 | 200 |
| Branch | 10 | 4 | 40 |
| Other. | 20 | 5 | 100 |
| | 100 | | 520 |

$$CPI_{Avg} = \frac{520}{100} = 5.2$$

$$CPI_{Avg} = \frac{\sum_{i=1}^{n} CPI_i * n_i}{\sum_{i=1}^{n} n_i}$$

$CPI_i$ → CPI for instruction of type i

$n_i$ → no. of instruction of type i

Performance of CPU is given by MIPS (million instructions per second)

$$MIPS = \frac{\text{no. of instructions}}{\text{Execution time} \times 10^6}$$

$$MIPS = \frac{\text{Clock rate}}{CPI_{Avg} * 10^6}$$

question gate. Consider two processors $P_1$ & $P_2$ executing the same instruction set. Assume that under identical conditions, For the same input, a program running on $P_2$ takes 25% less time but incures 20% more CPI as compaired to program running on $P_1$. If the clock frequency of $P_1$ is 1GHz then clock frequency of $P_2$ (in GHz) is?

| | $P_1$ | $P_2$ |
|---|---|---|
| execution Time. | $t_1$ | $t_2 = 0.75 t_1$ |
| CPI | $C_1$ | $C_2 = 1.2 C_1$ |
| Freq. | 1GHz | ? |

no. of instruction $n_1 = n_2$.

$$\text{execution time} = \frac{\text{no. of inst.} * CPI}{\text{frequency}}$$

$$\text{no. of inst.} = \frac{\text{Ex time} \times \text{freq}}{CPI}$$

$$\frac{t_1 * F_1}{C_1} = \frac{t_2 * F_2}{C_2} \Rightarrow \frac{t_1 \times F_1}{C_1} = \frac{0.75 t_1 \times F_1}{1.2 C_1}$$

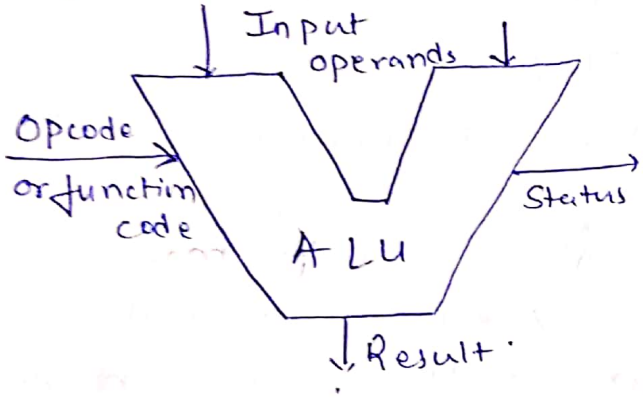
$$\Rightarrow F_1 = 1.6 \text{ GHz}.$$

Note:

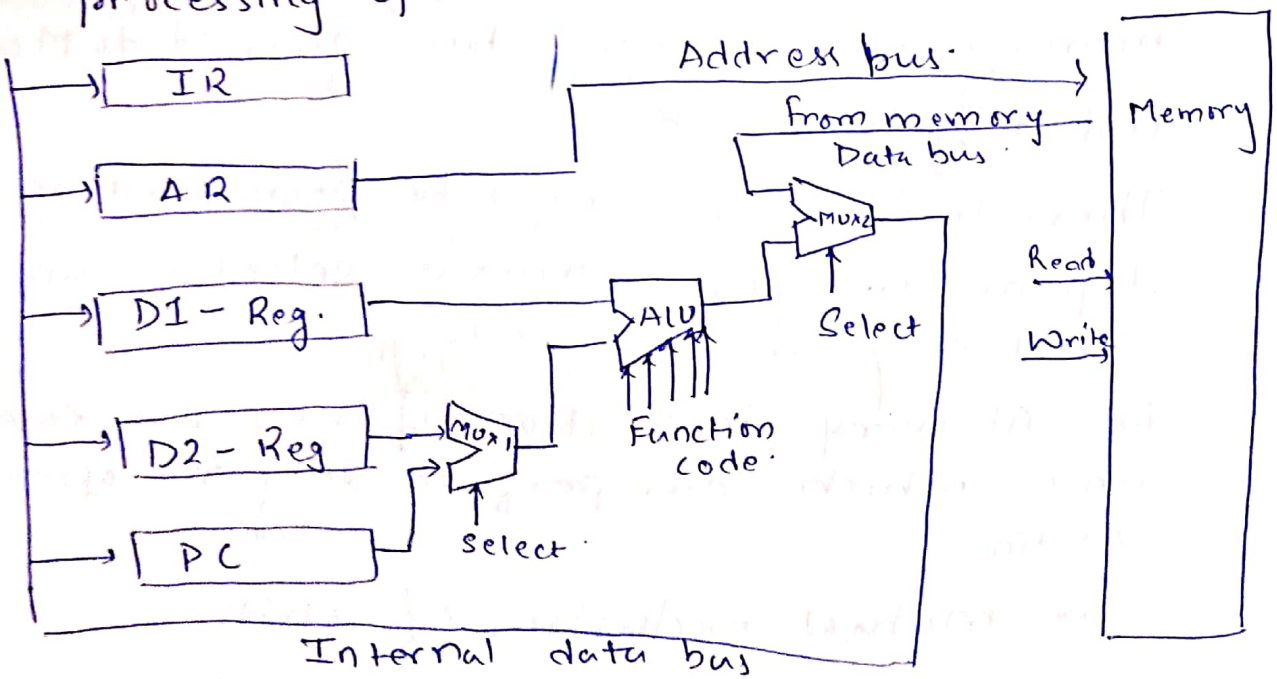
MIPS count doesnot provide always the correct performance of the CPU. because MIPS doesnot consider the complexity of instructin

* FLOPS — Floating point operations per second. It is another way to measure performance of CPU i.e performance measure matric.

⇒ ALU (Arithmetic Logic unit)




⇒ Data path := collection of functional units such as ALU, multiplexer.etc to perform data processing operations.



Instruction Fetch: IR ← M[PC]

But this instruction cannot be executed directly there are lot of processes included to execute this instruction.

This instruction will be divided as:

AR ← PC.

IR ← M[AR], PC ← PC + 1

i.e First PC value is supposed to be added to the AR and the M[AR] to [IR]

this happen in many steps further.

Firstly PC value will be forwarded to MUX1 with select =1 and then it moves further to ALU, ALU uses some function code to move it further such that A + B → B and then it moves further to Mux2 with select 1 which further passes to Internal data bus and through that data bus it gets copied into AR and further AR moves address through address bus to memory and memory bus gives it to Mux2 and it gets to 'IR'

These instruction cannot be performed in 1 step as everytime mux's selector can select only one output to flow.

i.e At every instruction if they use common unit which can perform only one operation at time.

i.e muthu exclusion of units

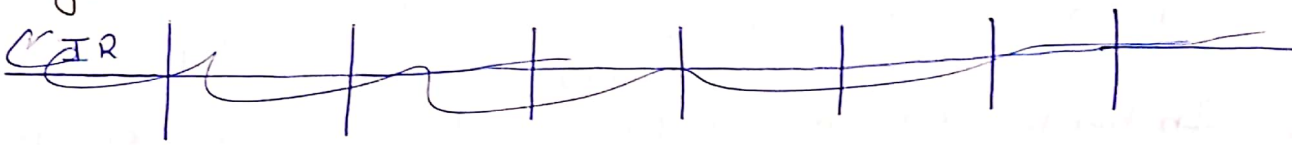
Cpu can perform only those instructions at a time which do not require common unit. Control unit will generate signals.

## * Control unit:

It generates control signals and sends those signals to the units of the system. and units perform their respective operation.

→ **Control variable:** The appropriate name of a control signal.

→ **Control word:** Set of all control signals generated by control unit at a time.




| IR | | AR | | D1 | | D2 | | PC | | | ALU | MUX1 | MUX2 | memory | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| In | out | In | out | In | out | In | out | In | out | Inc | | Select | Select | Read | write |

→ Control word For (AR ← PC)

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | xxxx | 1 | 1 | 0 | 0 |

Control word for (IR ← M[AR], PC ← PC + 1

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 00 | 0 | 0 | 1 | xxxxx | 0X | 0 | 1 | 0 | |

## → Control unit organization:-

→ Two types of control unit organization.

1) Hardwired control unit.
2) Microprogrammed control unit

**Hardwired control unit:** The control logic is implemented using logic gates, decoder & other digital circuits (Hardwired C.U.)

Advantages:

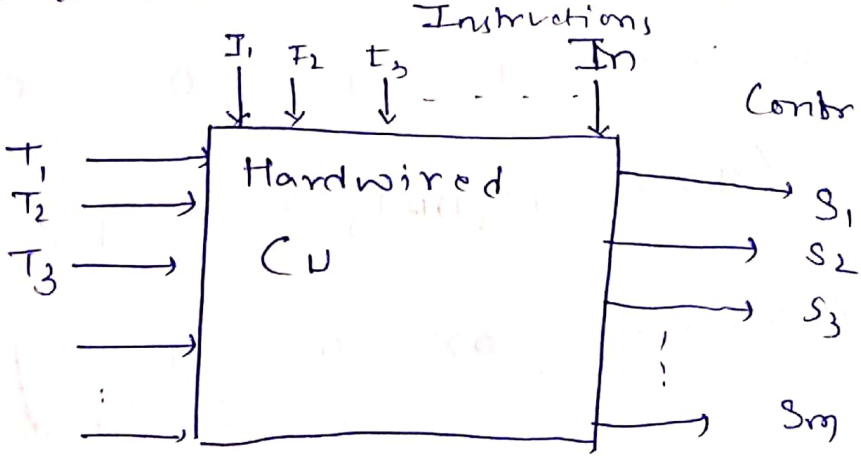
→ Faster mode of operation.

Disadvantage.

→ updation in control logic is difficult because rearranging wire among circuits is difficult.

In hardwired the logical gates are implemented in some circuit called control unit.

the instructions will come and for every instruction certain signals are supposed to be enabled or disabled at a particular set of time




So every Instruction cycle needs different set of signals this set of signals will lead to microoperations ∴ For every Instruction the time will start again from $t_1$

| | $I_1$ | $I_2$ | IB | In |
|---|---|---|---|---|
| $t_1$ | S1, S5, S9 | | | |
| $t_2$ | S2, S3, S4, S10 | | | |
| $t_3$ | - - - - - | | | |
| ⋮ | | | | |
| $t_n$ | - - - | | | |

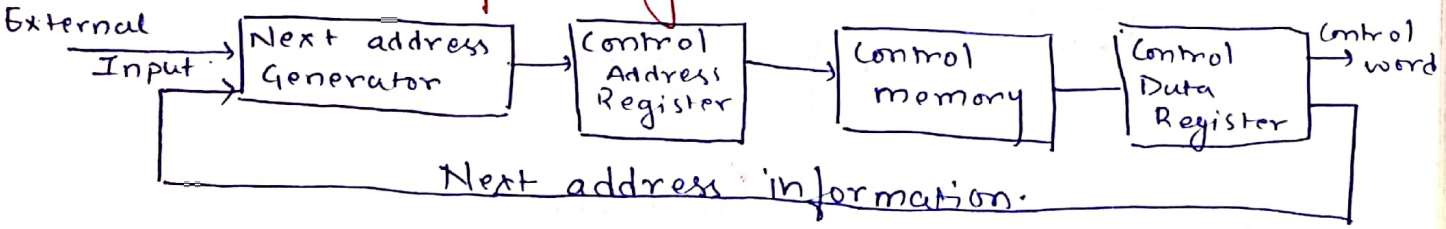
at one ~~cpu tt~~ microoperation time set of signals will generated. and for all instruction cycle time sequence and set of signals are formed. i.e for every Instruction again the time sequence is formed.

## Microprogrammed control unit :-

all the possible control words are stored in a memory. Based on the requirement. the appropriate control word is fetched and signals are sent to corresponding units.

Advantage: Updation in control logic is easy
Disadvantage: It is slower than hardwired CU.

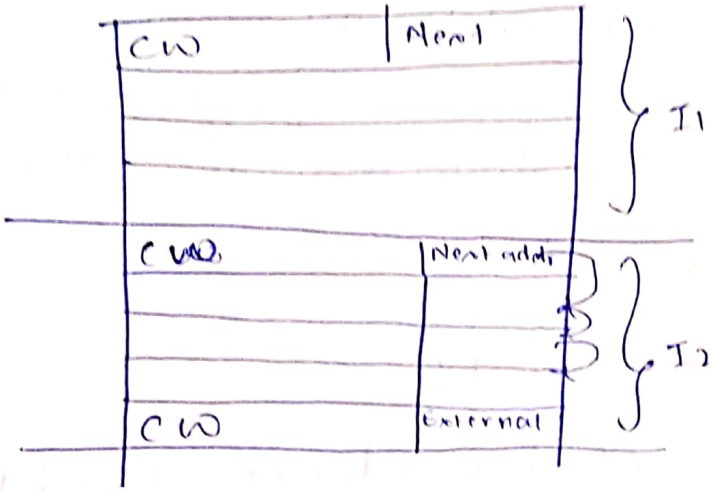
## Control word sequencing:-



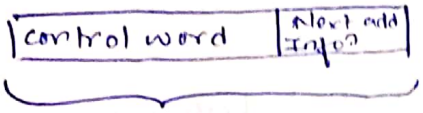

all the control word which are responsible to execute one type of instruction, are stored in control memory sequencially.

Inside control memory control word block is attached to another block which alerts whether to take the next address or to ~~take~~ stop the instruction cycle and weight for External input.

control memory.

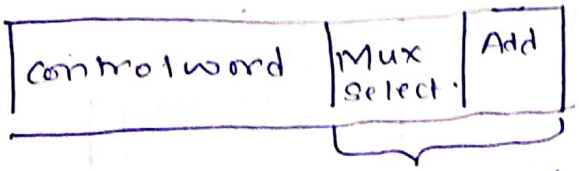



at every address in control memory the following information is stored



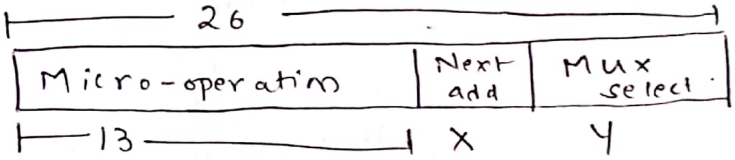

This is known as microinstructions.

Format of microinstruction is



Q4
P57

micro-instruction = 26 bit

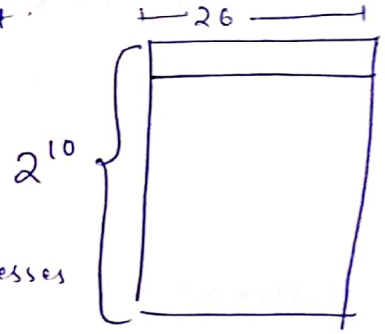



Mux needs 8 status bits that means Select requires 3 bits input.

i.e $Y = 3$

$13 + x + 3 = 26$.

$\therefore x = 10$ i.e $2^{10}$ addresses can be stored




how size of memory = $2^{10} * 26$ bits

= 26 k bits.

1 word = 26 bits.

$\therefore$ size of memory in words = 1024 words.

## Types of microprogrammed CU.

### ↳ Horizontal.

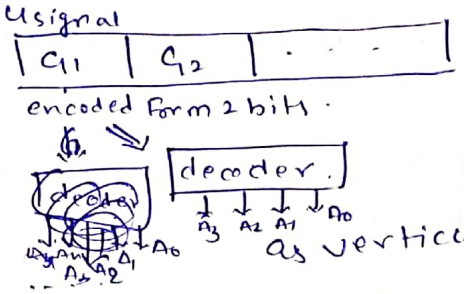
→ 1 Bit for each control signal in a control word

Hence larger sized control word.

### ↳ vertical.

The control signals are divided into multiple groups in such a way that within each group only one signal is enabled at a time.

each group information is stored in encoded form to decrease the size of control word. The control signals are generated using the decoders.



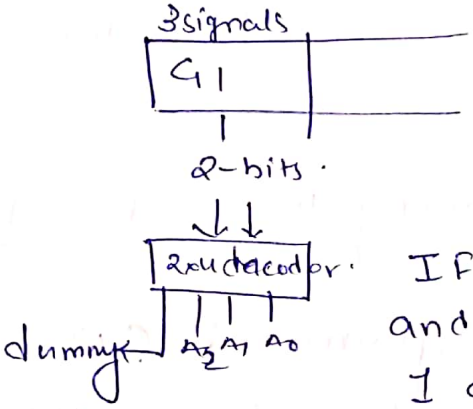

as vertical needs decoder.

Horizontal is faster than vertical.

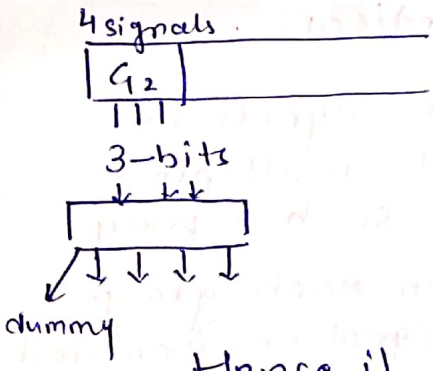
among all these CUs.

Hardwired is faster then Horizontal is faster than vertical.

→ atmost 1 signal enabled at a time within each group.




IF 3 signals are passed. and atmost we require 1 at a time.
then encoder will need. 2bits. and decoder will generate one extra signal for dummy

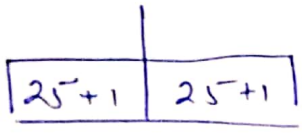



Hence if 4 signals are to be generated at a time the encoder will require $\lceil \log_2 5 \rceil$ bits i.e 3 bits to store 4 for 4 signals and 1 For dummy dummy says when to keep all the signal disabled

If any signal cannot be grouped then it is stored as horizontal way.

Qn A micro programmed controlled unit is required to generate a total of 25 control signals assume that during any micro Instruction atmost two control signals are active minimum no. of bits required in the control word to generate the required control signals will be ?

There are 25 signals without mutually exclusive 2 groups will be generated as to make atmost 2 pairs of signals to be active at a time.




5 bits + 5 bits = 10 bits.

Qn A CPU can support 16 distinct instructions. To execute each instruction a sequence of 64 micro-operations is required. the micro programmed control unit stores each micro-instructions in three fields 1) Horizontal control word to generate 128 signals 2) Next address field 3) MUX select field to select 1 of 8 MUX inputs.

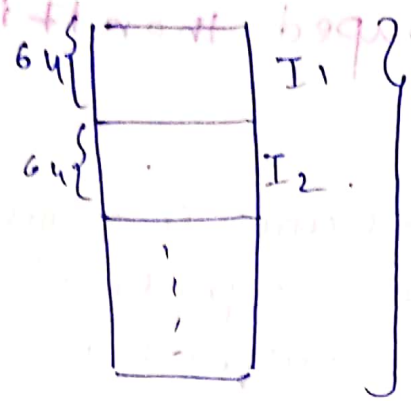
What is the size of control memory required for this control unit.

→ no. of inst? = 16

no. of micro-operation sequence per inst? = 64.

∴ total micro instructions stored in memory
$= 16 * 64 = 1024 = 2^{10}$

∴ control memory address size = 10 bit. to store 2^10 instructions.




$2^{10}$ micro-instructions required.

⇒ 10 bits to store them all

micro instruction:

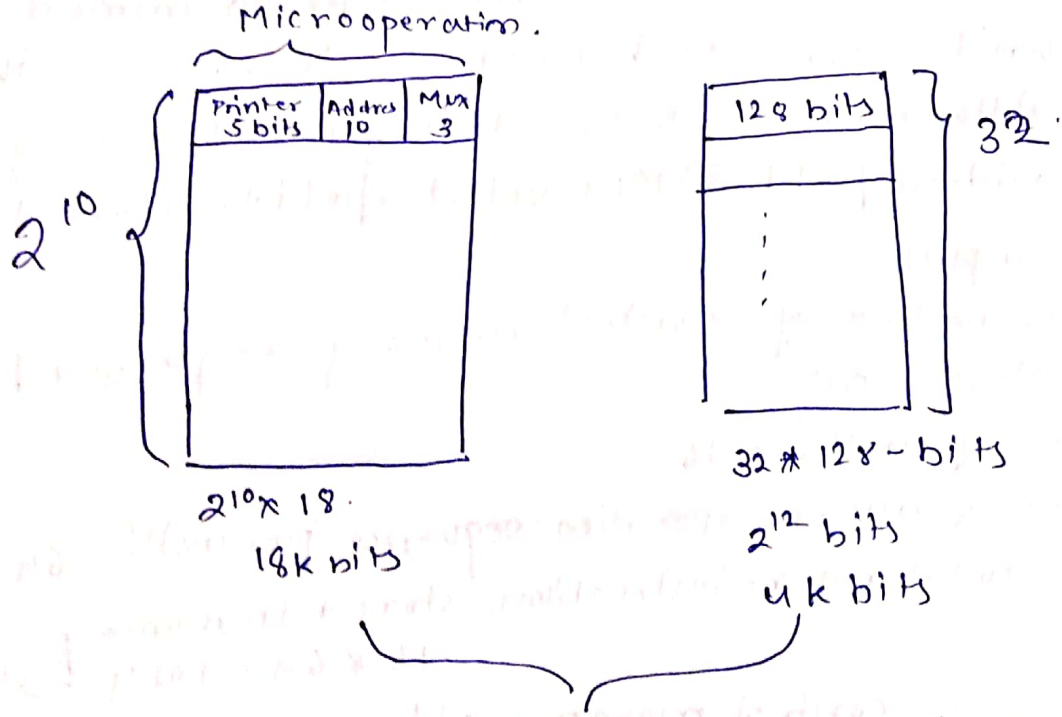
| control word | add | MUX |
|---|---|---|
| 128 | 10 | 3 |

= 128 + 10 + 3 = 141 bits.

∴ Size of control memory = $2^{10} * 141$ = 141k bits

If we consider nano programmed control unit.

Unique no. of microoperations (control words) = 32.




22 k bits will only be required

# RISC vs CISC

| RISC | CISC |
|---|---|
| → Reduced Inst? set | → Complex inst? set. |
| → Less no. of Inst? supported. | → more no. of inst? supported. |
| → Simple inst? | → Complex inst? |
| → less no. of addressing modes | → more no. of addressing modes. |
| → Fixed length inst? | → variable length Inst? |
| → Hard wired control unit | → micro programmed CU also possible. |
| → Reg-To-Reg (Input, Output) arithmetic operations only. | → Reg-to-memory & memory-to-Reg also possible. |
| → More no. of Registers | → Less no. of Registers. |

Q2 [S1]

9(2)
P(51) | all are correct.

98
P52 | (d)

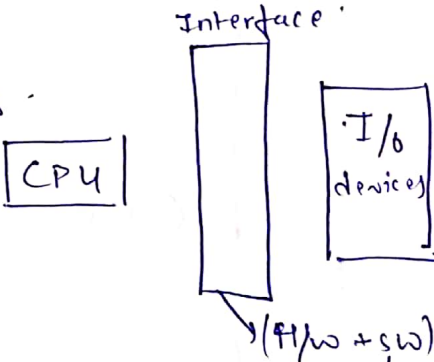
9|3
51 | (d) Initiation of interrupt service.

# I/O organization:

Peripheral device: a device which is connected to CPU externally apart from memory.

category of peripheral devices:

↳ Input device: } (aka) I/O devices.
↳ Output device.
↳ Storage device.




Need for interface:

1) Conversion of signals required.
2) Synchronization is required.
3) Data Format conversion is required.
4) All the i/o devices should be managed in such a way that each device should not disturb the operation of CPU & any other i/o device.

Primary memory is not included in I/O devices because there is no require of any interface to communicate with CPU i.e ~~why they are not incl~~ Peripheral devices

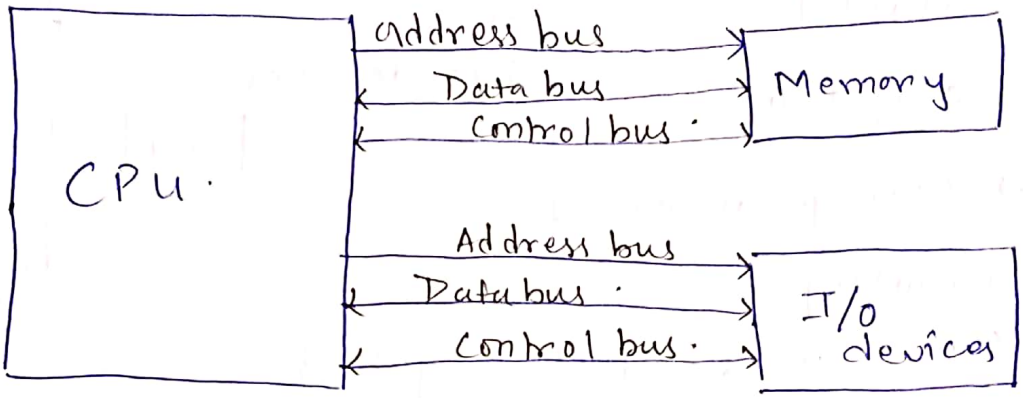
I/O processor = Interface + DMA controller + I/O instruction execution.

There are three ways ~~bet~~ in which CPU can communicate with memory & I/O

# * Communication of CPU with Memory & I/O

There are three ways in which CPU communicates with memory & I/O

1) Seperate Buses For both memory & IO

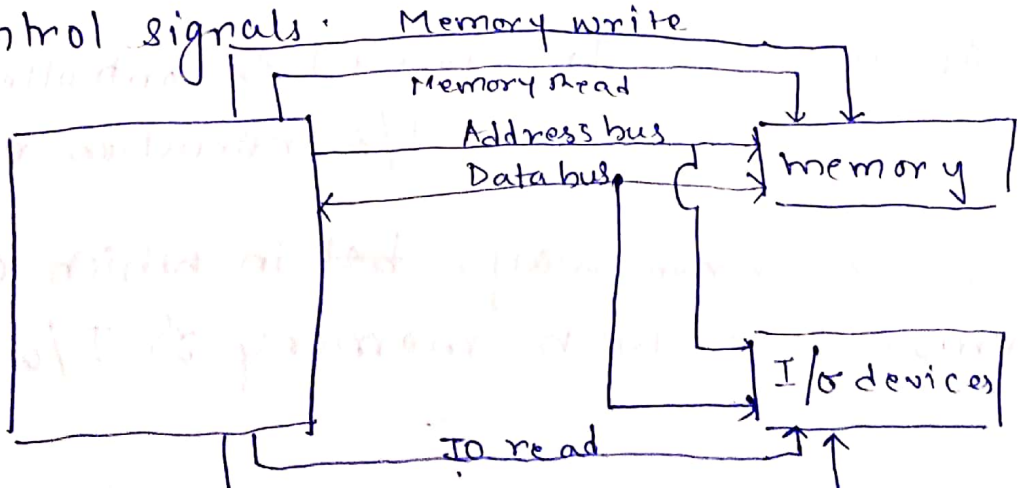



Address Bus in I/O is required to justify which I/O device is used at instance.

Advantages:
Easy to implement.

Disadvantage.
Costly.

2) Common address & Data buses with Separate control signals.



Here control signals (Memory Read/write, I/O Read or write) will suggest which address should be checked. and CPU will decide which operation to be done and control signal ~~which~~ which say what to access

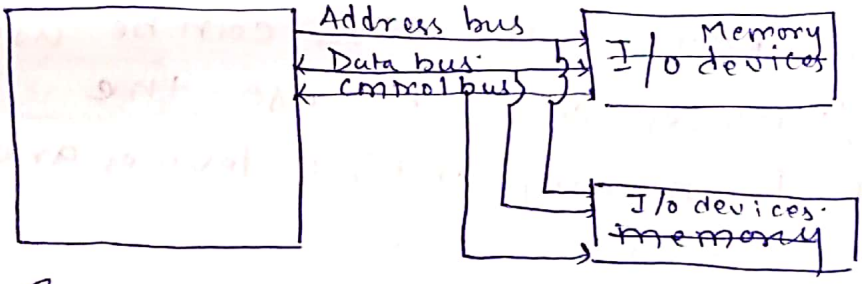
This type of communication is termed as (I/O mapped I/O) or (port mapped I/O)

Memory Read/write, I/O read or write is stored in a singal group in vertical micropmogramming i.e only one out of 4 will be active at time

Advantage:
No memory wastage

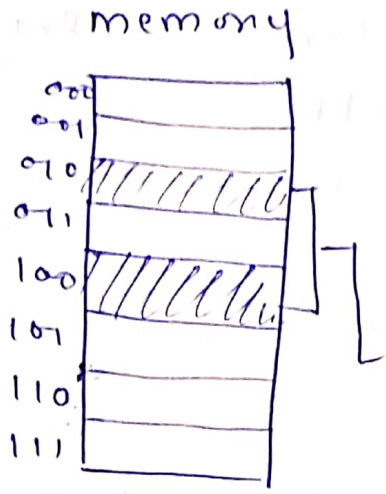
Disadvantage:
- less no. of Instructions to access I/O devices.
- less no of Addressing modes to access I/O devices.

3) Common buses For both memory & I/O.




From all available addresses Few are assigned to I/O devices, these addresses are stored in memory hence For every address which directs to I/O device memory is wasted.

eg: consider a system with 8B (Byte addressable memory and 2 I/O devices d1, d2.




assume

| device | address |
|---|---|
| $d_1$ | 010 |
| $d_2$ | 100 |

These two addresses are useless to memory which is memory wastage.

These address table will be stored in memory controler which decides the allocation of memory and assigns the control to memory or I/O device.

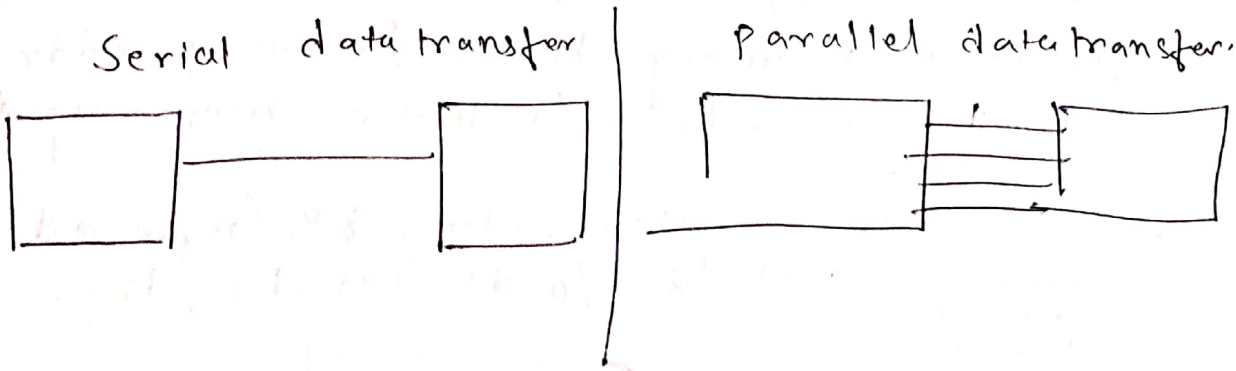
Port mapped I/O

→ Advantages:
- more no. of Instructions to access the I/O devices
- more no. of addressing mode to access the I/O device

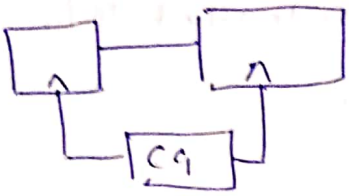
→ Disadvantage.
- a part of memory is wasted.

All the memory access instructions can be used to access I/O devices also because the control signals for memory and I/O devices are common.

* Serial asynchronous data transfer.

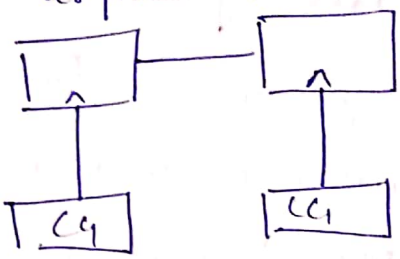


Synchronous



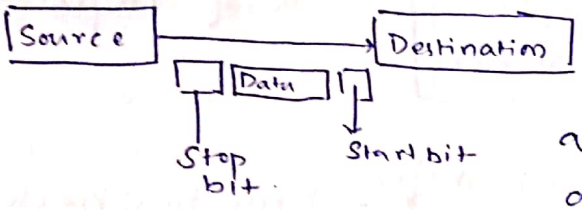

Here common clock generator is used & is tuned based on the slowest m/c which tends to lower performance

asynchronous.




Here externally Synchronisation is provided to reduce the data loss between m/c. but performance will be upgraded.

Synchronous Serial data transfer: In this a fixed amount of data is sent from source to destination which also takes '2' bits 1 for 'stop' & 1 for start bit this will synchronize the two machines both Sender & Destination is aware with the time gap of data transfer.




assume.
data size = 8 bit.
a total size of 8+1+1=10 bit. are transfered to transfer a char of 8-bits.

* Modes of Data transfer:

⟹ Modes of data transfer:-

1. Programmed I/O or Program controlled I/O
2. Interrupt initiated / driven I/O
3. DMA (Direct memory Access).

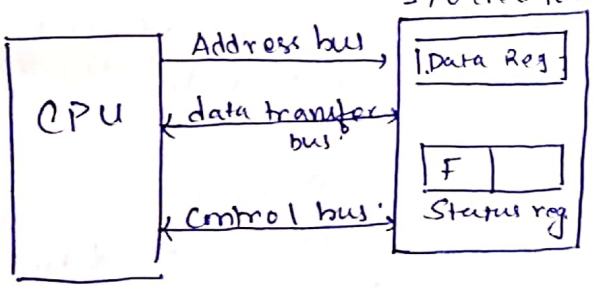
→ Programed I/O or program controlled I/O

There is no any provision (signal) through which I/O device can inform to CPU about data transfer.

→ In such case the device sets its own Status and ~~wait~~ waits.

→ CPU runs a program periodically and checks the status of every device one by one.

→ If any device has its status set then CPU performs data transfer for that device.




Status register by default is of "1 Byte"

Total time required to transfer data = time to check the Status of device
i.e Time required to transfer status register (1B) from I/O device to CPU
+
time to transfer actual data.

disadvantage here is the wastage of CPU time. everytime it has to check for the status of I/O device.

→ Interrupt initiated I/O:-

I/O devices have a provision (Interrupt signal) to inform CPU about data transfer.

—When CPU recieves Intrupt from any device.

(i) CPU completes the execution of current instruction

(ii) PC value + other status values CPU Stores onto the stack.

(iii) CPU takes a branch to service the Interrupt.

(iv) after the service using the stack content CPU resumes previous process.

Interrupt Service Routine (ISR): a function execution of which services the Interupt.

Vector address:- Starting address of ISR is known as vector address.

Interrupts are of two types based on vector address.

- vectred
- Non-vectored or scalar.

In vectored Interrupt cpu will generate an Interrupt and it will also give vector address of ISR to resolve the interrupt service and service is done.

The device sents starting address of ISR (vector address) along with the interrupt signal. In such case CPU directly jumps to the provided address, executes ISR & services the interrupt.

→ Non vectored Interrupt:
The device sends only interrupt signal to CPU. In such case CPU first executes 'default service routine' which helps the CPU to resolve the scenario and to obtain the actual ISR address.

→ Based on acceptability interrupts are of two types.

- Maskable
  CPU can accept or rejects
- non maskable
  always accepted
  eg: Trap

→ Based on source Interrupts are of two types.

- External (Hardware)
  If any I/O device generates the interrupt.
- Internal (Software)
  During execution of any instruction if any unexpected error occurs then an internal interrupt is generated.
  Internal interrupt are always non maskable.

eg for interal interupt :

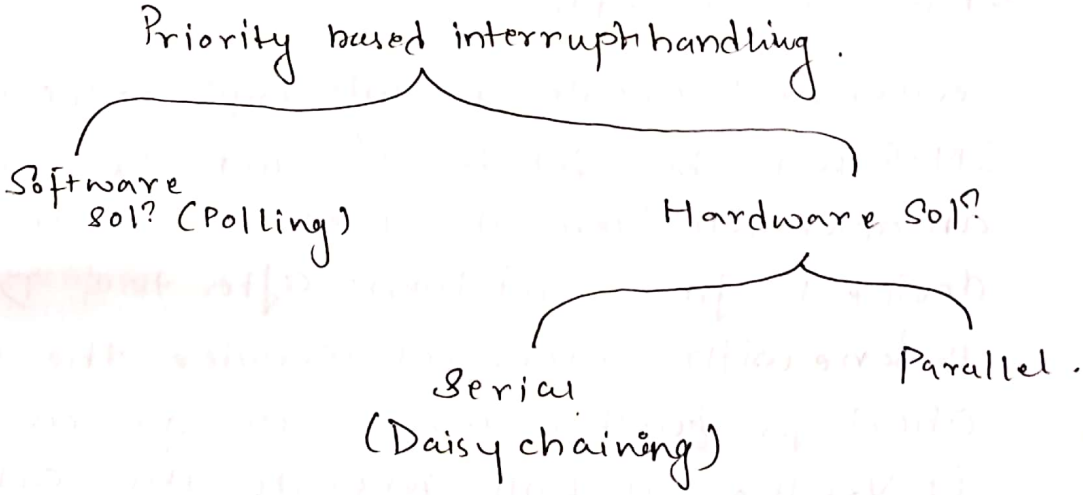
↳ $R_1 \leftarrow R_2 / R_3$ ; $R_3 = 0$

↳ Page fault.

↳ addressing error in segmentation.

↳ Exceptional handling in program.

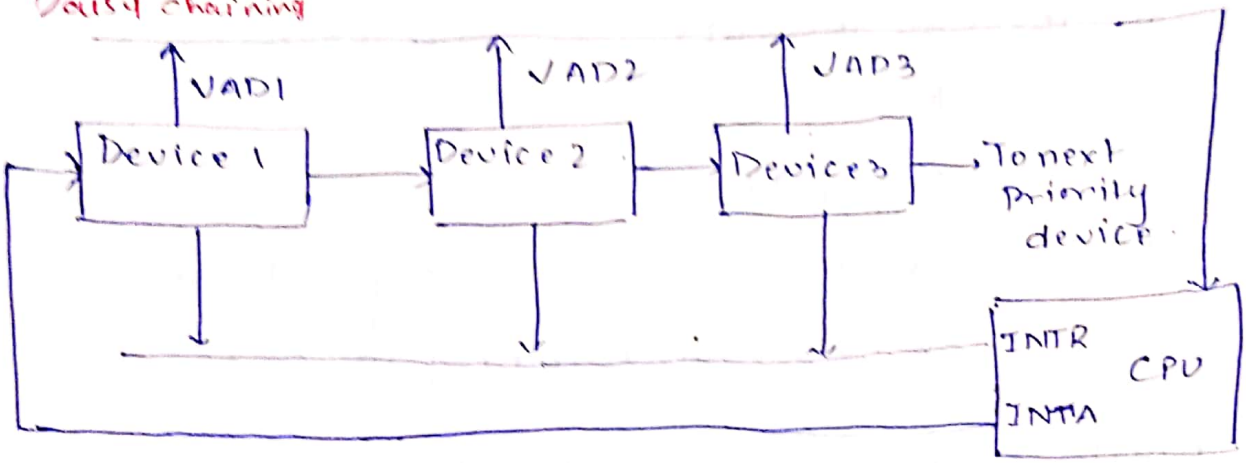
If multiple devices generate the interrupt simultaneously then CPU will service the interrup of highest priority device first.




Polling: whenever the interrupt occurs the program will check every devices one by one from which the interrupt is emitted then it will resolve the interrupt.

Polling will only be started when interrupt is encountered.

Daisy chaining




INTR: Interrupt Request.

INTA: Interrupt Acknowledgement.

VAD: Vector address.

Devices will create an interrupt requests and INTR will be set to '1' and CPU will generate an acknowledgement which will move to device 1 first and then after the Device 1 is done with interrupt service the CPU will check for further more interrupts available if Yes then it will generate the acknowledgement and will send it to Device 1, Device 1 will pass it to another Device 2 so on. If all the interrupts are handled then only the CPU will stop the process hence not all devices are scanned for interrupt. only the one with interrupt.

In daisy chaining the device which electrically nearest to CPU will be having highest priority.

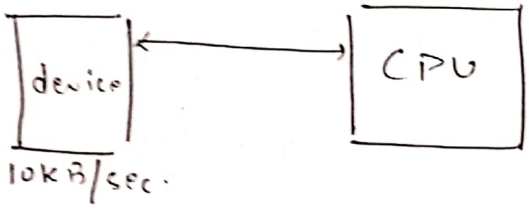
Else generally the devices which is most frequently accesed by CPU will have highest priority i.e HDD.

Time required to perform data transfer (or interrept service) = interrupt overhead (Time from Receiving Intr to starting Intr Service) + Service time (data transfer time)

Q6 / PG4

Device speed = 10kB/sec.
Data transferred bytewise.
Interrupt overhead = 4 μsec.

Device speed is internal processing speed of that device not the transfer or data transmission of from ~~data~~ device to CPU.




Here transmission time is neglible or data transfer time bet? Device to CPU or memory = 0.

Time required in programmed I/O.

= Time to check Status + data transfer time.

Status register = 1 Byte.

∴ 1 Byte transfer time from Device + 0 = 100 μs + 0
= 100 μs.

10 KB of data can be transferred in = 1 sec.
1B of ... ... in = $\frac{1}{10k}$ sec
= $= 10^{-4}$ sec
= 100 μ sec.

| | Time. | Otherwise bits, bytes etc |
|---|---|---|
| k | $10^3$ | $2^{10}$ |
| M | $10^6$ | $2^{20}$ |
| G | $10^9$ | $2^{30}$. |

Time required in Interrupt I/O = interrupt overhead time + data transfer time.

$= 4\mu s + 0$

$= 4\ \mu sec.$

Performance of Programmed I/O = $1/100$ M sec.

Performance of Interrupt I/O = $1/4$ M sec.

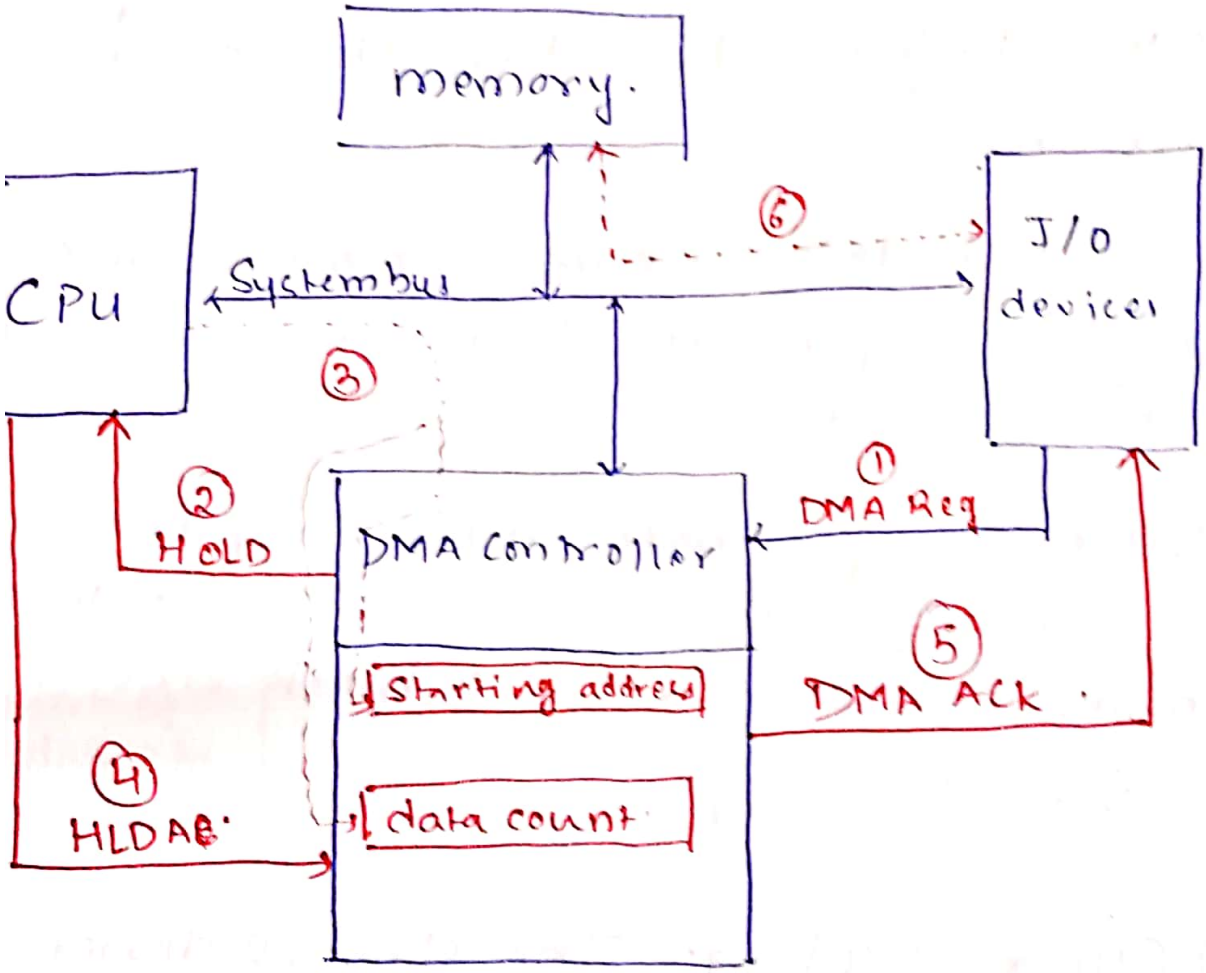
Performance gain = $\frac{1/4}{1/100} = \frac{100}{4} = 25$.

Performance gain = $\frac{\text{Slower technique time.}}{\text{faster technique time.}}$

## * DMA :-

Direct memory access : It is a technique which enables direct data transfer between memory and I/O device without and Interferon of CPU.

To implement this technique ; A specific hardware is used known as DMAC (DMA Controler)




arting address :- Memory address starting from where the data transfer should be done.

ata count :- No. of bytes or words to be transfered.

| | After 1B is transfered | After 2B | After 3B |
|---|---|---|---|

Note: During DMA Transfer CPU cannot perform those operation which require system buses. which means the CPU will be blocked mostly. System bus is used by DMA controller.

* Modes of DMA Transfer:-

ones the control of the buses is given to DMA controller how much data is transfered through DMA before CPU takes the control of the buses back.

(i) Burst mode:- The entire amount of data is transfered before the CPU takes the control of the buses back.

(ii) Block transfer mode:- A block of data is transfered before the CPU takes the control of the buses back.. usually a block size is bet? 512B to 1kB

(iii) Cycle Stealing:- The slow IO device takes some time to prepare the data. during that time CPU keeps the control of the buses. Ones the data is prepared CPU gives the control of the buses to DMA controler for one cycle. in that cycle the prepared data is transfered to memory and then CPU takes the controls of Buses back.

Time to prepare the data = $t_x$.

time to transfer data to memory = $t_y$.

% of time CPU blocked = $\frac{t_y}{t_x + t_y} * 100\%$ → (1)

When I/O can start preparation of next data during transfer to memory.

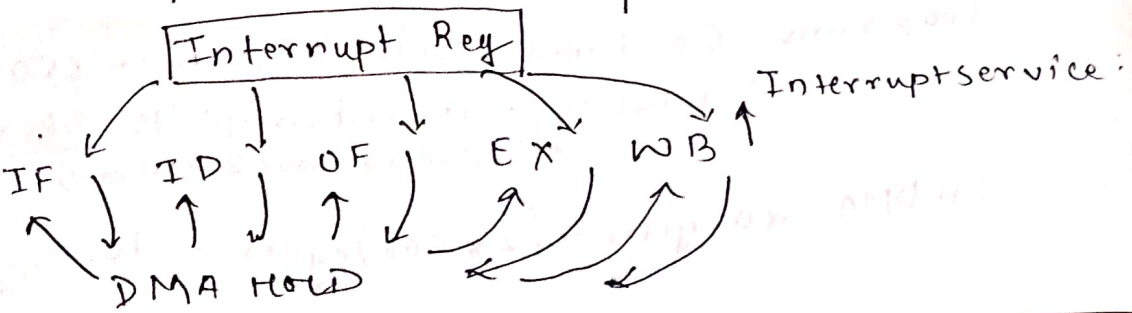
% of time CPU blocked = $\frac{t_y}{t_x} * 100\%$ → (2)

as $t_y$ is included in $t_x$ itself

If $t_x$ is Largen then only use the second formula.

If $t_x \geq 2t_y$ then only use second formula. else use the First one.

| | Burst mode | Cycle stealing. |
|---|---|---|
| 1) Amount of total data to be transfered. | Small | Large. |
| → Amount of data transfered in one time | More. | less. |
| → CPU waiting | More | Less. |
| → DMAC waiting | Less | more. |



DMA controller is a special purpose processor which is used for data transfer between memory and I/O.

Q8/64 Here 500B from I/O to memory using interupt I/O which uses CPU Hence it is not DMA. it is interrupt driven. program based i/o

---

1 cycle ← Initialize add. Reg.

1 cycle ← Initialize count = 500.

2 ← loop: load a byte from device.

2 ← Store in the mem at add given by add reg.

1 ← Inc add reg.

1 ← Dec the count.

4 ← if count != 0 goto loop.

---

DMA

500B from I/O to memory

20 cycles for initialization.

2 cycles/byte.

$$\text{Speed up} = \frac{\text{Slower time}}{\text{Faster time}}.$$

$$= \frac{\text{Interrupt I/O time}}{\text{DMA time}}.$$

Loop runs 500 times with 7 cycles = 3500

∴ Total cycle in Interrupt I/O = 3502 (Including above 2 statement)

In DMA 20 cycles + (2 * 500) cycles = 1020 cycles.

$\therefore$ speedup $= \frac{3502}{1020} = 3.4$ (approx)

Q2 / 63

Data count Reg = 16 bits

File 29,154 KB from disk to memory

Data count Reg says how many bits of data can be transfered at time.

Max value of data count register $= 2^{16} - 1$

$= 65535$

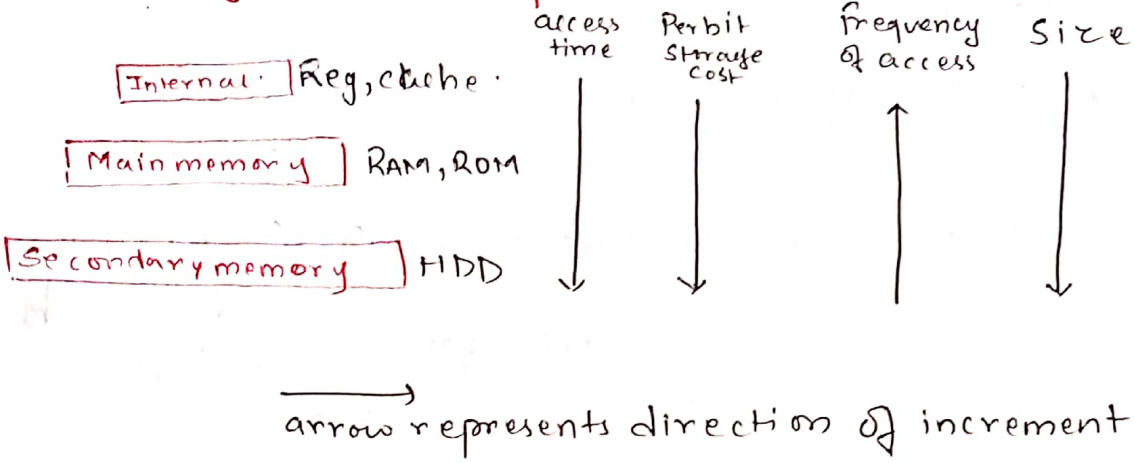
$\therefore$ minimum no. of time required $= \frac{29154 \text{ KB}}{65535 \text{ B}}$.

$= \frac{29154 \times 2^{10} \text{ B}}{65535 \text{ B}}$.

## * Memory organization:

a physical device which is use to store the contents in the system is known as memory.

## → Memory Heirarchy




→ arrow represents direction of increment.

Memory is divided among two functionality purpose

4 (i) Maximize access speed.

(ii) Minimize cost of bit per storage.

| memories | Size (in bits) | Per bit storage cost. | Cost of memory. |
|---|---|---|---|
| $M_1$ | $S_1$ | $C_1$ | $S_1 C_1$ |
| $M_2$ | $S_2$ | $C_2$ | $S_2 C_2$ |
| $M_3$ | $S_3$ | $C_3$ | $S_3 C_3$ |
| | $(S_1 + S_2 + S_3)$ bits | | $S_1C_1 + S_2C_2 + S_3C_3$ is Total cost of memory system |

∴ average per bit storage cost $= \frac{S_1C_1 + S_2C_2 + S_3C_3}{S_1 + S_2 + S_3}$

$$= \frac{\sum_{i=1}^{n} S_i C_i}{\sum_{i=1}^{n} S_i}$$

$S_i$ = size (in bits) of memory $M_i$

$C_i$ = per bit storage cost of memory $M_i$

**memory representation:**

The memory is represented by

= no. of cells * 1 cell capacity.

= no. of memory address * bits per address

| no. of cells | 4 | 8 | 16 | $2^n$ | x |
|---|---|---|---|---|---|

4 GB memory ⟶ 4 G × 1 B.
(4 G: no. of cells; 1 B: size of each cell)

$= 2^{32} \times 1B$.

∴ It will require 32 bit address size.

ie to have capacity of 4GB we require atleast 32 bit address size.

2 | 5

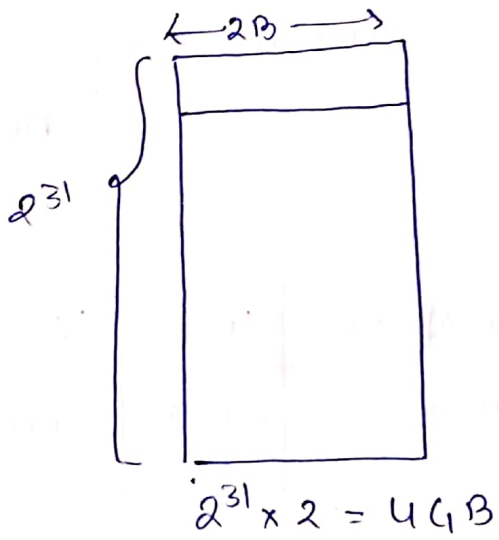
4GB
word addressable.
word size = 2B

no. of words = $\frac{4GB}{2}$.

$= 2GB$

$= 2^{31}$ Bytes

address = 31-bits




⟹ **Main memory.**

It is used to store current running processes (Instructions) & their data.

Two types of Main memories.

⟶ ROM ⟹ Non-volatile.

⟶ RAM ⟹ Volatile.

During Booting process CPU does the POST & then it takes OS from the HDD ~~ROM~~ and stores it to the RAM this Booting process instructions are also stored in ROM so ROM is also necessary in computer

Program responsible For Booting is Bootstrap program.

Bootstrap program is stored in ROM

OS is stored in HDD

But in modern days computer Bootstrap program is stored in HDD and Bootstrap Loader is stored in ROM so that you can get previlage to change Bootstrap programs.

This Bootstrap loader will Load the program from HDD to RAM.

Types of ROM :-

(1) EROM (Eraseable Rom) :- With the help lazer or UV rays the content can be erased

(2) PROM (Programable Rom) user can perform one time writing in such ROM.

eg: CD-Rom.

(3) EPROM (Erasable & Programable) :- user can perform erase operation using UV rays which may require certain Lab environment.

(4) EEPROM (Electrically erasable programable ROM) Electric current can erase the content.

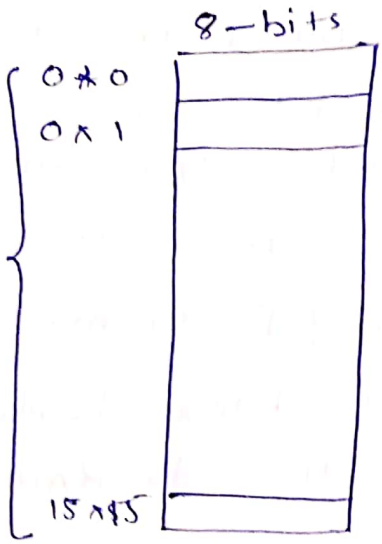
(5) EAPROM (Electrically alterable programmable ROM) Electric current can be use to alter the content.

GATE:- The amount of ROM needed to store the table for multiplication of two 4 bit unsigned integer is?

For multiplication of 2 4bits will require 8 bits.

| A | B |
|---|---|
| 4 bits | 4 bits |
| $2^4$ possible | $2^4$ possible. |

$2^4 \times 2^4 = 2^8$




∴ Size of ROM required = $2^8 \times 8$

$= 2^{11}$ bits

$= 2k$ bits.

| A | B | A*B | A+B | Multiplication table size. | Addition table size. |
|---|---|---|---|---|---|
| 4 bits | 4 bits | 8-bits | 5 bits | $2^8 \times 8$ bits | $2^8 \times 5$ bits |
| n-bits | n bits | 2n-bits | (n+1)-bits | $2^{2n} \times 2n$ bits | $2^{2n} \times (n+1)$ bits |

# Types of RAM

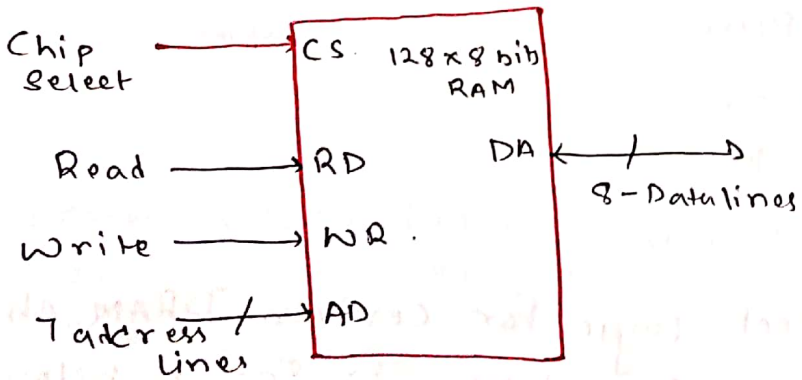
## SRAM (static RAM)

- Built using flip flops
- No Refresh or recharge
- Faster
- Expensive.
- Used for cache memory

## DRAM (Dynamic RAM)

- Build using capacitors. (In form of Electric charges & this charge tends to ~~discharge~~ discharge.
- Periodic refresh or recharge is required.
- Slower as refresh operation requires time.
- During refresh no read write is performed.
- Inexpensive.
- used for Main memory.

## RAM chip:



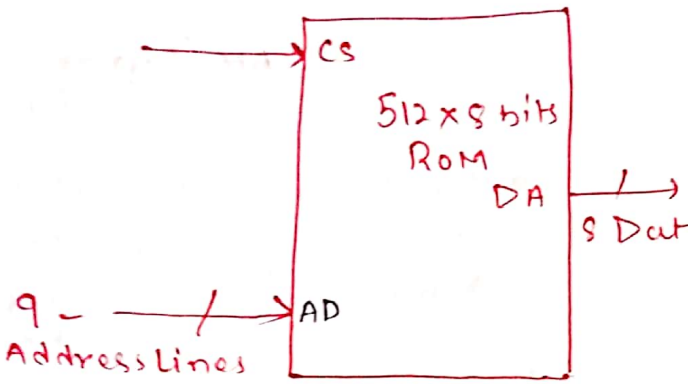

Cross on line says many lines are involved

CS – 1 pin
RD – 1 pin
WR – 1 pin
Add – 7 pins
DA – 8 pins

| CS | RD | WR | Operation. |
|---|---|---|---|
| 0 | x | x | no operation |
| 1 | 0 | 0 | no operation |
| 1 | 0 | 1 | write |
| 1 | 1 | x | Read |

x → means any input doesn't matter

## ROM chip:




CS – 1 pin
AD – 9 pins
DA – 8 pins.
18 pins.

| CS | operation |
|---|---|
| 0 | No operation |
| 1 | Read. |

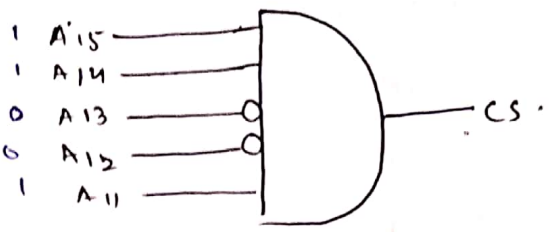
no need of Read/write column as only Read is performed.

question - A ROM chip of 1024 x 8 bits has 4 Select lines and operates from 5 volt power supply how many pins are needed For the IC package.

address = 10 pins.
data = 8 pins.
CS = 4 pins.
power = 1 pin
ground = 1 pin
24 pin.

size = 1024 x 8 bit
$= 2^{10} \times 8$.

gate: The chipselect logic for certain DRAM chip in a memory system design is shown below. assume that the memory system has 16 address lines denoted by A15 to A0. what is the range of addresses (In Hexadecimal) of the memory system that can be enabled by the chipselect (cs) signal




(A) C800 to CFFF
(B) C800 to C8FF
(C) DA00 to DFFF
(d) CA00 to CAFF.

to enable CS i.e CS = 1 all the given lines should be ANDed to give out 1 and remaining can be any thing.

∴ 1100 1000 0000 0000 i.e C800 to C8FF
C 8 0 0

1100 1111 1111 1111
C F F F

## * Organising multiple RAM chips.

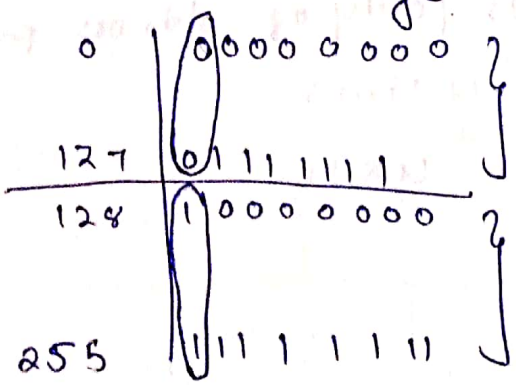
Total memory capacity = no. of chips * 1 chip capacity

Consider 2 Rams of size 128 x 8-bits each

∴ total capacity = 128 x 8 bit ⟶ 7 bit address
+ 128 x 8 bit ⟶ 7 bit address
256 x 8 bit ⟶ 8 bit address

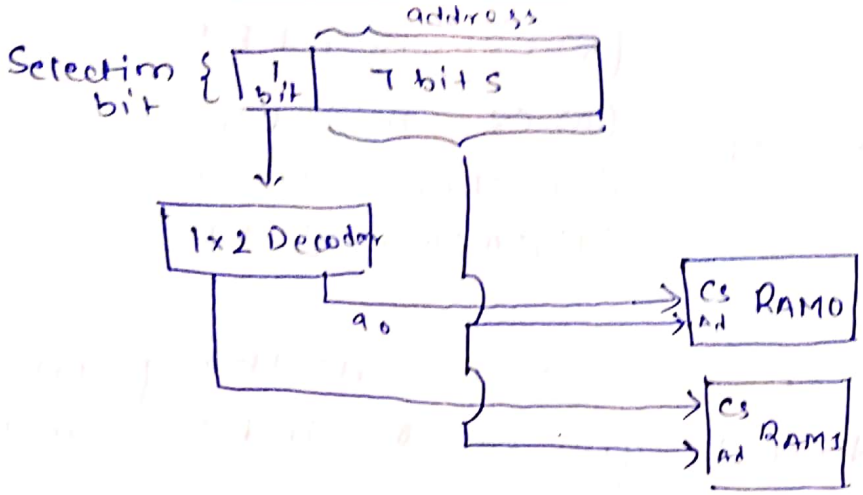
Previously only 7 bits were required to store addresses now 8 bit will be required.

Address Range.




RAM0 will have these addreses

RAM1 will have these addresses




Question (i) How many 128 x 8 bits RAM chips are needed to provide a memory capacity of 2048 bytes.

(ii) How many lines of the address bus must be used to access 2048 bytes of memory? How many of these lines will be common to all chips

(iii) How many lines must be decoded for chip select specify the size of decoder.

→ no. of chips = $\frac{\text{Total memory capacity}}{\text{1 chip capacity}}$.

$= \frac{2048\ B}{128\ B} = \frac{2^{11}}{2^7} = 2^4 = 16.$

address lines = 11 lines.

common = 7 lines (only of address part)

Lines going for CS = 4 lines.

decoder size = 4 x 16.

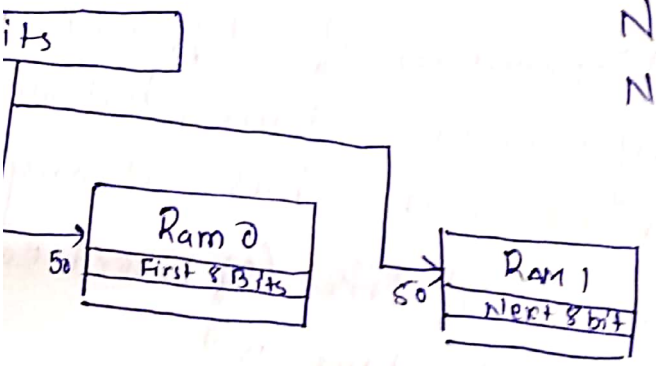
n How many 128 x 8 bit ram chips are
ded to provide the memory capacity of
128 x 16 bits

. of chips = $\frac{\text{Total memory capacity}}{\text{1 chip capacity.}}$

$= \frac{128 \times 16 - \text{bits}}{128 \times 8 - \text{bits.}}$

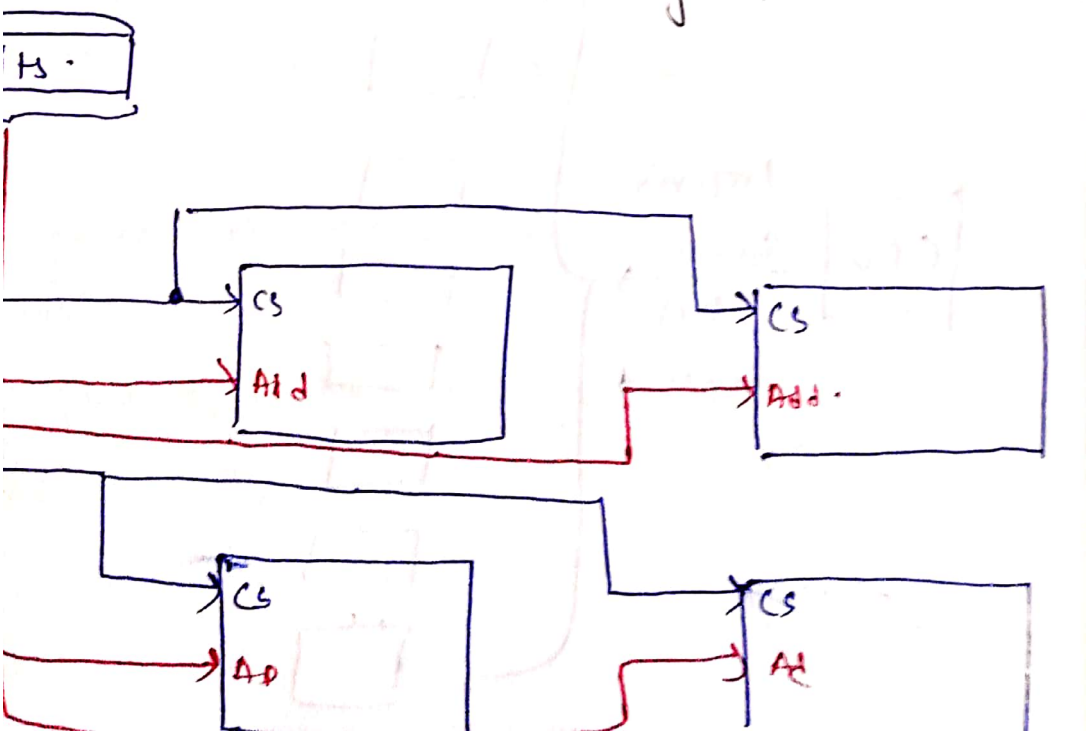
$= 2.$

~~address~~ no. of address is same but the stored per address is doubled here we horizontal arrangement where we utes no. of bits of Address among 2 chips natally




No. of addresses ⇒ same
No. of data bits per address ⇒ double

w many 128 x 8 bits ram chips are
l to provide a capacity of 256 x 16 bit



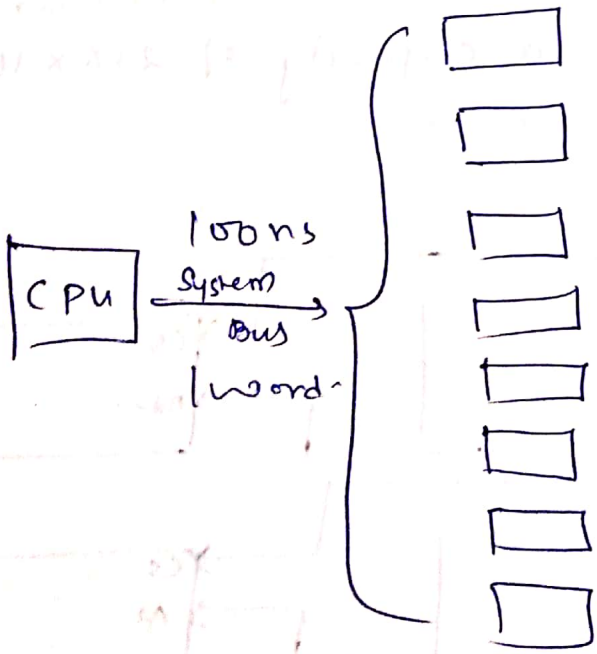
How many 32k x 1 ram are needed to provide memory capacity of 256k bytes.

Total capacity = 256k x 8 (Default unit of capacity is bit)

1 chip capacity = 32k x 1

∴ Ram required = '64'

→ Consider a main memory system that consist of 8 memory modules attached to the system bus which is 1 word wide. When a write request is made, the bus is occupied for 100ns and ~~500ns~~ for 500ns there after, the addressed memory module executes 1 cycle accepting and storing the data. The (internal) operation of different memory modules may overlap in time, but only one request can be on the bus at any time. The maximum no. of stores (of 1 word each) that can be inisiated in 1 ms is?




∴ after every 500ns the modules are freed.

∴ no Request will be stopped.

∴ $\frac{1 \text{ m sec}}{100 \text{ ns}}$

$= \frac{10^9}{100 \times 10^3} = 10^4$

$= 10000$

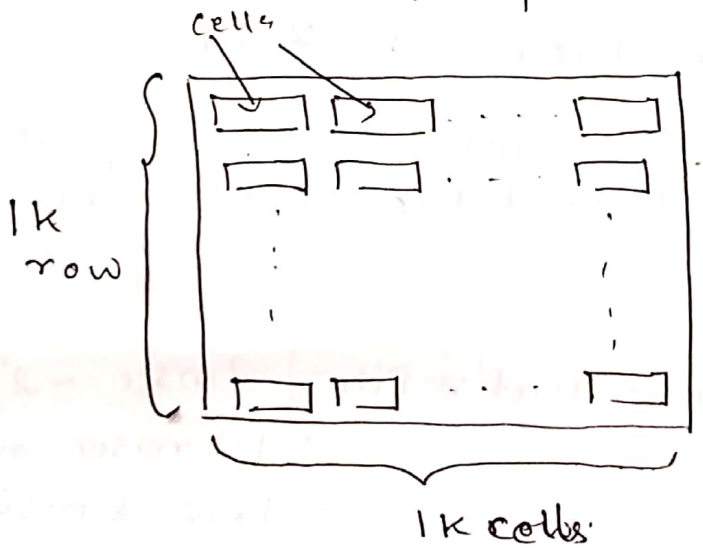
## Arrangement of cells in DRAM chip:-

⇒ In 1 refresh operation 1 row of cells is refreshed.

> 1 chip refresh time = no. of rows in chip * 1 refresh time.

memory system refresh time = 1 chip refresh time.
[all chips can be refreshed parallely]

1MB DRAM chip (Byte addressable)

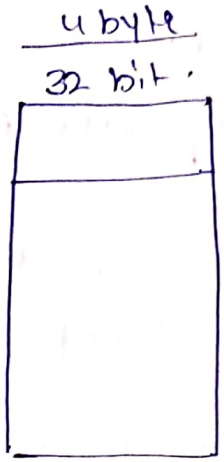



1 refresh operation time = 1 row refresh time
i.e whole row is refreshed at a time.

→ a main memory unit with a capacity of 4MB is build using 1M x 1 bit Dram chips. each DRAM chip has 1K rows of cells with 1K cells in each row. the time taken for a single refresh operation is 100 ns. The time required to perform 1 refresh operation on all the cells in the memory unit is?

(a) 100 ns (b) $100 * 2^{10}$ ns (c) $100 \times 2^{20}$ ns.
(D) $3200 * 2^{20}$ ns.

Q24
P23




Requirement
1 GB ⟶ 256M x 4 Byte.

Given.
256M x 4 Bits.
= 8 chips (Horizontal only)

no. of row = $2^{14}$

1 operation = $50 \times 10^{9}$

1 chip refresh time = $2^{14} * 50$ nsec

But after every 2 milisecond 1 refresh time starts at that time nothing read or write will occur.

Time remaining for read write = 2 msec. $- 2^{14} \times 50$ nsec
= 1.2 msec. or
= 1.1808 msec.

$2^{14} \times 50$ nsec.
$= 2^{4} \times 2^{10} \times 50$ nsec.

$2^{10}$ can be taken as 1000 or 1024

$= 16 \times 1k \times 50 \times$ nsec. or $16 * 50 \times 1024$
$= 800$ Usec $= 819200$ nsec
$= 0.8$ msec. $= 0.8192$ msec.

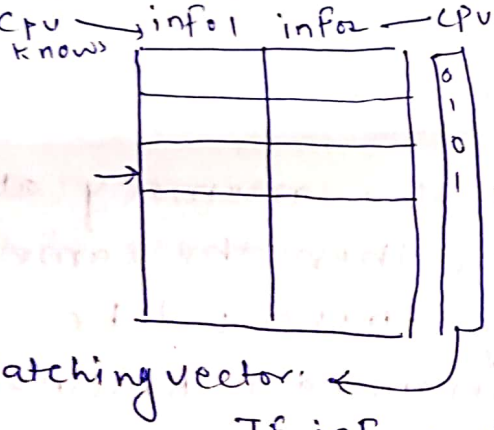
∴ ~~Time remaining~~
% time for read/write $= \frac{1.2}{2} * 100\%$.
$= 60\%$.
or
1.1808 ...

## * Associative Memory:-

No any address for the cells in this memory hence searching is performed based on the content stored in the cells. That is why this memory is known as content addressable memory (CAM)

Every cell of memory contains 2 info? which CPU knows & which CPU requires.

Every cell of memory contains a matching logic hardware to match the content in cells. Hence this memory becomes very faster but expensive also.



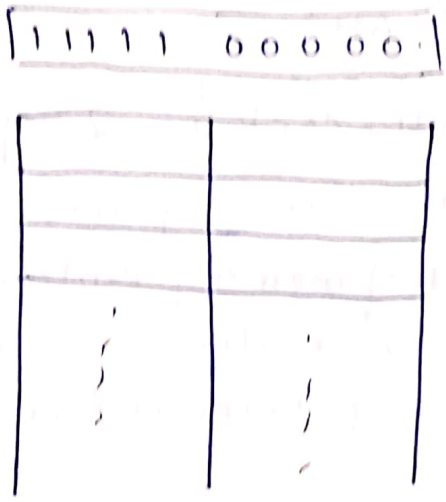

CPU generates some Information of the data and sends it for matching to the memory. this sending takes place parallely into all the cells. at a time.

If information matches with any cell its corresponding cell information is copied into CPU.

This memory is used to implement Cache & TLB

matching is done by looking at the matching vector. if multiple matching occurs then all the corresponding values are sent to CPU sequencially

Comparison or information blocks Info1 & info2 can be divided according to users choice. This variable or variation in information block is looked by "Mask register"




## Locality of reference.

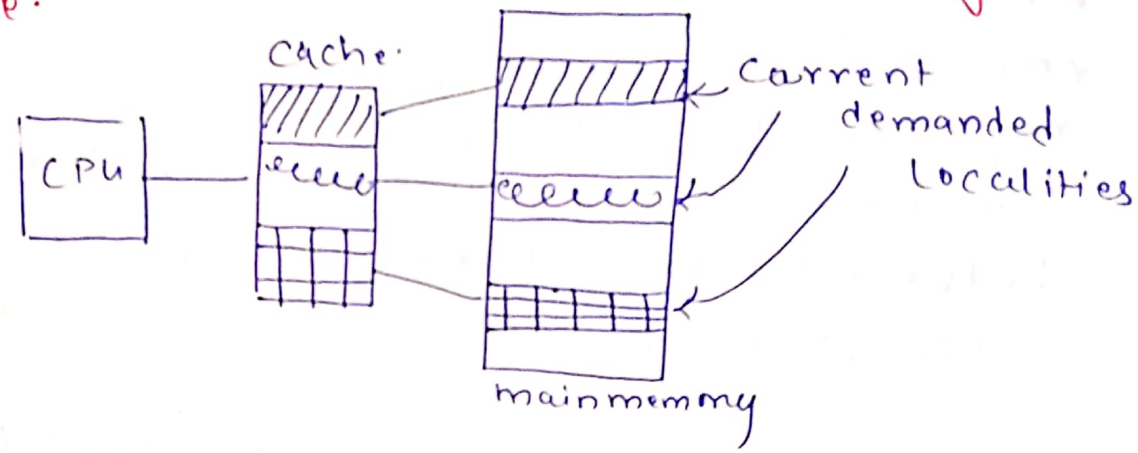
IF CPU reference the memory at a particular address then the same address or the nearby addresses will be refered soon. This phenomena is known as locality of Reference.

- Temporal (time)
  If same address is refered again
- Spatial (space)
  If nearby address are refferd.

## * cache memory.

Based on locality of reference concept the current demanded localities (Block) are kept into a smaller and faster memory known as cache memory.

Use of cache reduces over all memory access time.




### Working of cache:-

If CPU's demanded content is present in cache it is known as cache hit.

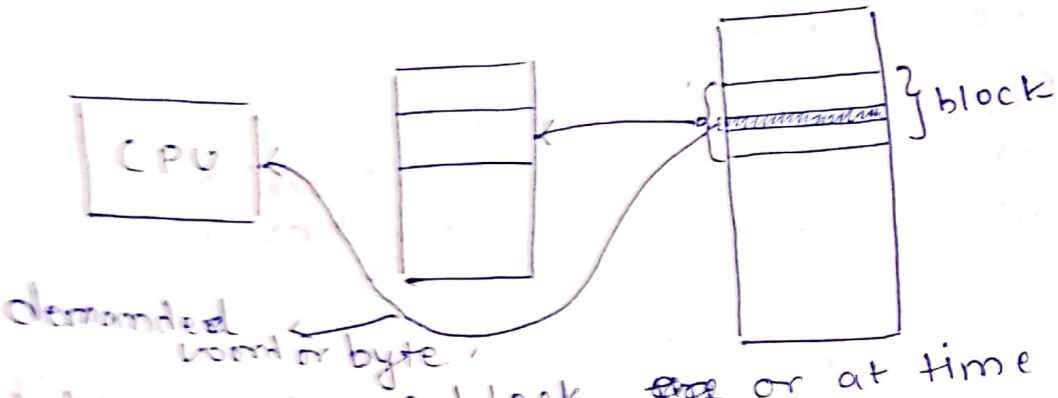
& If not present in cache its cache miss.

Performance of cache is given by cache Hit ratio

⟹ Cache hit ratio (h) = $\frac{\text{no. of hits in cache}}{\text{total memory references}}$

⇒ miss = 1 − h

In case of cache miss 1 block (containing the demanded content) is copied from main memory to cache. because of this if the other content of block CPU demands then there will be hit in cache.




while copying a block ~~free~~ or at time of miss ~~CPU~~ the block is transfered to cache and the required info is transfered to CPU at the same time so total time taken will only be the total time taken to copy the whole block to cache. as it is maximum

$$\therefore \text{average memory access time} = \text{Hit ratio} * \text{mem access time for hit}$$

$$\text{average memory access time} = \text{Hit ratio}(h) * (\text{memory access time for hit}) + (1-h) * (\text{mem access time for miss})$$

(1−h) = miss ratio

## Types of cache access :-

1) Simultaneous access.

request for both cache & main memory generated simultaneously.

When there is hit then only cache access time is required and RAM access request is discarded but in case of miss the RAM access is already present.

$$\therefore t_{avg} = H * t_{cm} + (1-H)\, t_{mm}$$

Where $t_{avg}$ = avg memory access time

H = hit ratio; $t_{cm}$ = cache memory access time.

$t_{mm}$ = main (RAM) memory access time

(ii) Hierarchical access :-

only the Faster memory is accessed First i.e cache. If hit then great or if missed then the miss phenomen will be initiated to access main memory

$$t_{avg} = H * t_{cm} + (1-H)(t_{cm} + t_{mm})$$

$$= Ht_{cm} + 1 t_{cm} + t_{mm} - Ht_{cm} - Ht_{mm}.$$

$$\boxed{t_{avg} = t_{cm} + (1-H)\, t_{mm}}$$

If in question Hierarchy or level given then use Hierarchy access else by default Simultaneous is used.

* If locality of reference should be used

(i) Simultaneously access :-

at time of miss the whole block is accesed

$t_{avg} = H * t_{cm} + (1-H)\, t_{block}$

$t_{block}$ = mm. access time For a block

= block size * $t_{mm}$.

(ii) Hierarchical access.

$t_{avg} = t_{cm} + (1-H)\, t_{block}$.

* If memory access and block transfer time should be needed then. Previously only reading was done now write or transfer

(i) Simultaneous access :-

$$t_{avg} = H * t_{cm} + (1-H)(t_{block} + t_{cm})$$

(ii) Hierarchical access :-

$$t_{avg} = t_{cm} + (1-H)(t_{block} + t_{cm})$$

in one access of cache memory entire block can be accessed

→ If tcm & tmm is not given ⟹ use general formula

→ If tcm & tmm given ⟹ case(i) If word hierarchy or level given then use hierachical.

else
use simultaneous

Q 18 P 22 | 0.8 * 5 + 0.2 * 50
= 14 nsec.

question: In the two level memory Hierarchy if the top level has an access time of 8 ns & the bottom level has an access time of 60 nsec. What is the hit rate required on a top level to give an ~~acess t~~ average access time of 10 ns.

top level ⟹ mem which is closest to CPU.

$t_{avg} = t_{cm} + (1-h) t_{mm}$

$10 = 8 + (1-H)60$

$H = 0.96$ i.e 96.6 % of Hit ratio

→ a computer system contains a cache, uncached memory access takes 8 times longer than to access cache if the cache has hit ratio of 0.9 then ratio of cached memory access time to uncached memory access time is?

→ ~~$t_{cm} = \frac{1}{8} t_{mm}$~~

~~$h = 0.9$~~

Uncached memory means the memory system without cache i.e only main memory.

Cached means memory system with cache (cache + mm)

$t_{cm} = \frac{1}{8} t_{mm}$.

or $t_{mm} = 8\, t_{cm}$

$h = 0.9$

$$\frac{\text{Cached access time}}{\text{uncached access time}} = \frac{(0.9)x + (0.1)8x}{8x} = \frac{1.7}{8} = 0.2125$$

## * Cache writing :-

If CPU perform write on a cache content. then corresponding main memory content should also be updated.

→ Write through.

→ Write back.

Write through : If ~~cache~~ CPU updates the cache simultaneously main memory should also be updated. i.e

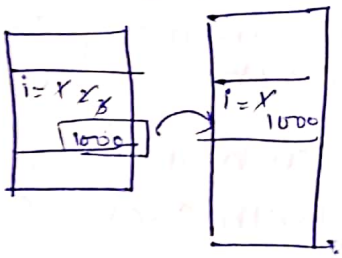
i.e as soon as the cache memory content is updated corresponding main memory content is also updated immediately.

☺ Same value for ~~cache~~ content in cache & main memory always

☹ time consuming.

→ Write back:-

If CPU updates the cache contents then let it happen the main memory content is not updated immediately. When the updated block is to be replaced then only it is copied back to main memory.

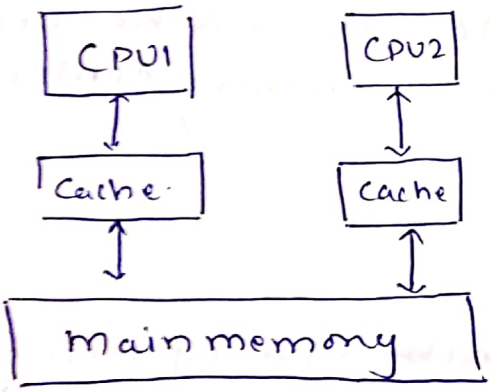



every operation gets completed on the cache then only the result is forwarded to RAM.

Cache memory will only be accessed by CPU.

Two different devices trying to access same content from two different memory will lead to problem of coherrence.

* Cache coherrence: In multiprocessor system.




In write trough:—

$t_{avg\ read} = t_{avg}$ formulas.

$t_{avg\ write} = max(t_{cm}, t_{mm})$

where always $t_{mm}$ will be higher

$= t_{mm}$.

$t_{avg(read+write)} = \%\ of\ read * t_{avg\ read} + \%\ of\ write * t_{avg\ write}$

Question

$t_{cm} = 100 nsec.$

$t_{mm} = 1000 nsec$

$H_{read} = 80\%$

Write through policy.

% of read = 70 %. % of write = 30%

$t_{avg\ read} = ?$

$t_{avg\ write} = ?$

$t_{avg\ read+write} = ?$

$t_{avg\ read} = (H)\, t_{cm} + (1-H)\, t_{mm}.$

$= (0.80)\ 100 ns + (0.20)\ 1000$

$= 280 ns.$

$t_{avg\ write} = 1000 nsec.$

$t_{avg\ read+write} = 0.7 * 280 + 0.3 * 1000 = 496 ns$

gate 2014 The memory access time is 1ns. for a read operation with a hit in cache; 5ns for a read operation with a miss in cache; 2ns for a write operation with a hit in cache & 10ns for a write operation with a miss in cache. execution of a sequence of instruction involves 100 instruct ion fetch operations 60 memory operand read operations & 40 memory operand write operatin the cache hit ratio is 0.9 the average memory access time (in nanoseconds) in executing the sequence of instructions is?

| | hit | miss |
|---|---|---|
| read | 1nsec | 5nsec |
| write | 2ns | 10nsec. |

H = 0.9.

$t_{avg\ read} = 0.9 * 1 + 0.1 * 5 = 1.4ns.$

$t_{avg\ write} = 0.9 * 2 + 0.1 * 10 = 2.8ns.$

execution of seq of inst?

↳ 100 instruction fetch ⇒ read = 100

↳ 60 operand read ⇒ 60 read.

↳ 40 memory write.

∴ 100 + 60 = 160 read operation.

40 write.

∴ 80% read & 20% write.

$\therefore t_{avg} = \frac{80}{100} \times 1.4nsec + 0.2 * 2.8ns.$

$= \underline{1.68}\ nsec.$

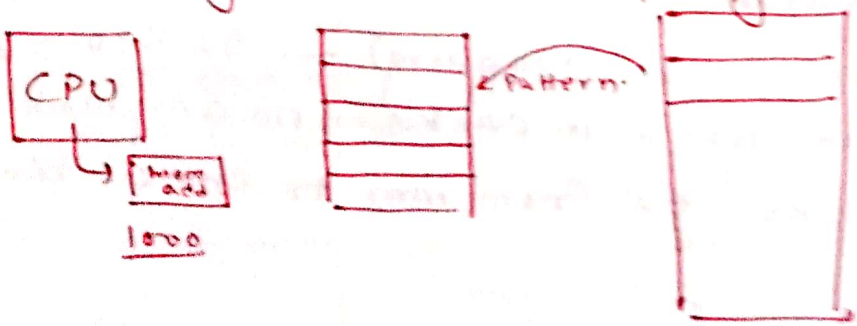
## * Cache mapping

CPU always generates main memory address the size of main memory is larger which can not be stored in cache memory fully There should be a pattern with the help of which the main memory content from a specific address is copied to a specific Location in cache memory this pattern is known as cache mapping


this pattern helps us to search in cache using the main memory address i.e if that pattern is present in cache then its a hit else its missed.

Transformation of main memory data into cache memory is known as cache mapping.

* there are three types of mapping.
  (i) Direct mapping
  (ii) Set associative mapping
  (iii) Fully associative mapping.

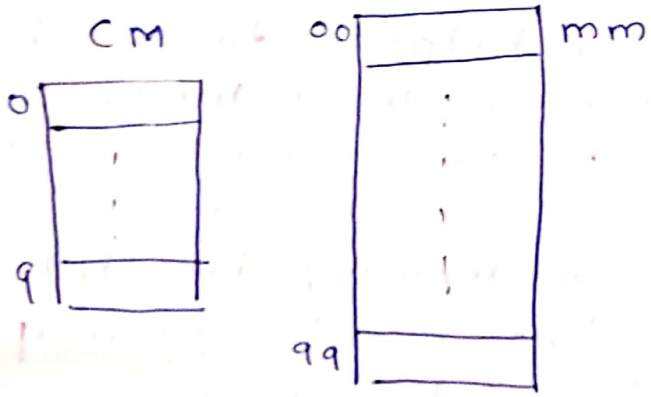


## * Direct mapping :-

No. of blocks in cm = 10 (0 - 9)

no. of blocks in mm = 100 (00 - 99)

| CM | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| MM | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 |
| | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| | : | | | | | | | | | |
| | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 |




Cm block no = mm block no. % no. of blocks in CM (mod)

CPU demands mm blocks — 52.

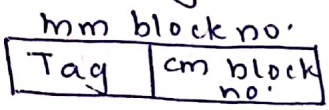
CM block no. (mapping) — 52 % 10 = 2.

check block no. 2 in cache ⇒ no any block ⇒ miss

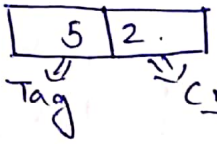
bring block no. 52 from mm to cm at block '2'

| CPU demands mm block | CM block no. (mapping) | Operation. |
|---|---|---|
| 52 | 52 % 10 = 2 | Check block no. 2 in cache ⇒ no any block is present ⇒ miss bring block 52 from mm to cm at block no. 2. |
| 96 | 96 % 10 = 6 | miss ⇒ bring block no. 96 from mm to cm at block no 6. |
| 32. | 32 % 10 = 2 | check block no. 2 in cache ⇒ Block 52 present ⇒ miss. bring block no. 32 from mm to cm at block 2 by replacing block 52. |

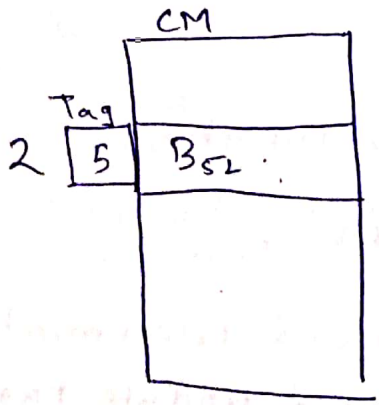
So 2, 12, 22, ... 92. all these blocks are competitors For being in cache memory at block no. 2.
therefore every block we have 10 competitors each
But cache memory cannot differentiate the competito
block ID present in Block as it is in Binary.
So with every Block ID some another field is attacked to distingwish between every competitors
That distingwisher is known as Tag.



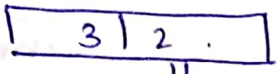

eg: mm block no ⇒ 52.



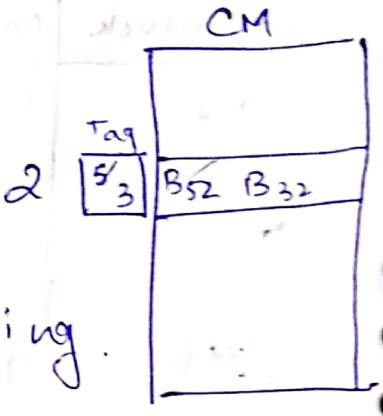

goto block no. 2.
no block present
miss
bring block 52 in cm block 2



mm block no. 32.




goto CM block no. 2.



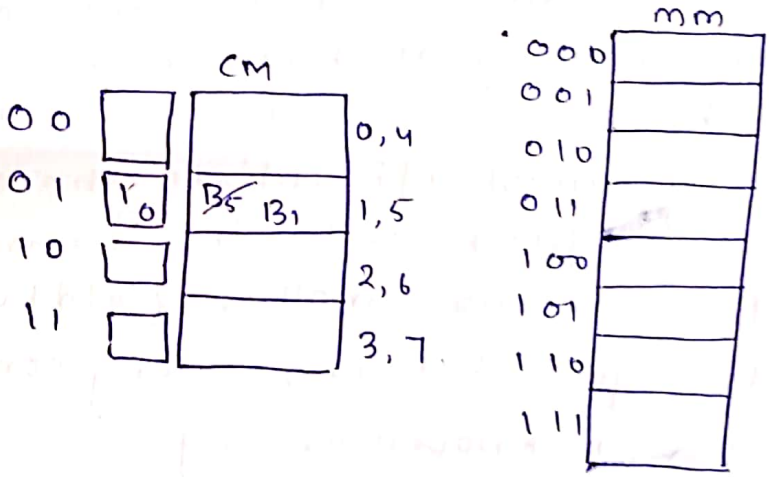

Tag present = 5
Tag in request = 3
} no matching.

miss

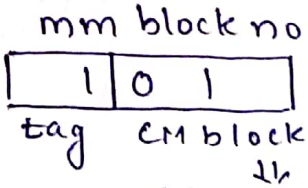
bring block 32 in cache & update tag to 3.

---

no. of blocks in CM = 4.
no. of blocks in mm = 8




CPU Req => MM block no. required = 101




goto block no. 01 in cache

no block => miss

Bring block no. 5 from mm to CM, in block no. 01 & update tag bcyl

CPU req => mm block no. = 1 => 001

mm block

| 0 | 0 1 |
|---|---|

tag ⇓ goto block no. 01 in cache

⇓

tag present = 1

tag req = 0 } no match

⇓

miss

⇓

Bring block 1 from mm &
replace block no. 5 & update
tag by 0.

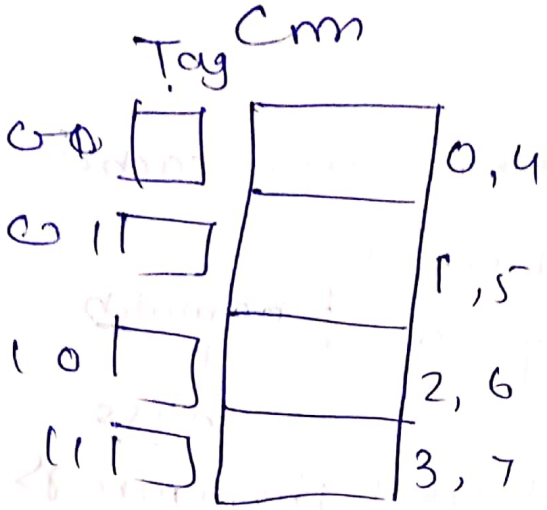
The tags are distributed and is stored inside the cache controller.

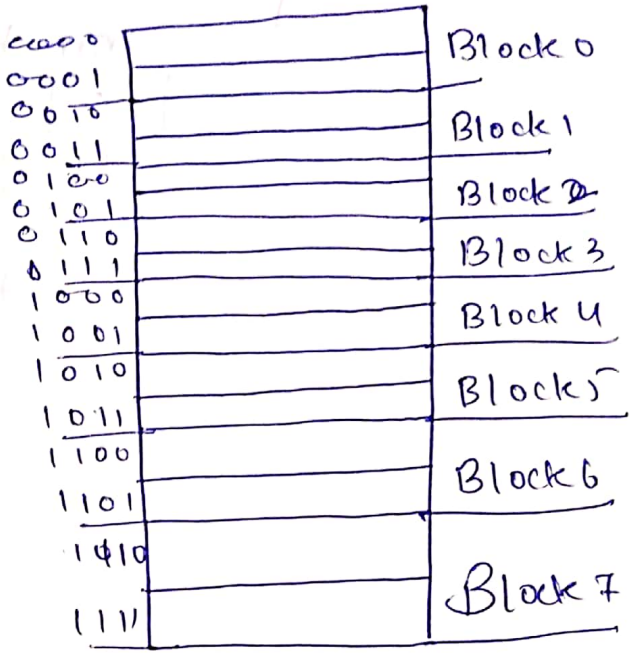
Cache controller maintains One tag information For each block in cache.

∴ Size of tag directory = no. of blocks in CM * Tag bits

or

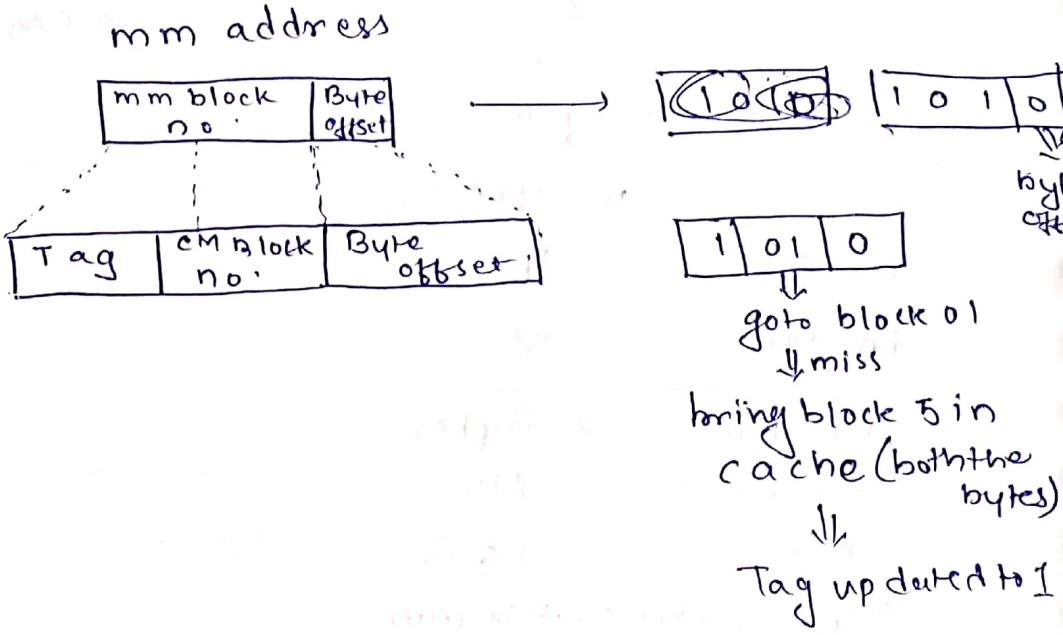
Size of meta-data.



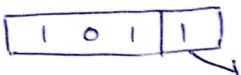




If CPU generates any address each block has 2 Byte which can be represented by 0, 1 in the LSB of Address. Byte no. ⇒ Byte offset.

∴ CPU generates address ⟹ 1010



CPU generates add 1011




~~block~~ byte offset = 1


goto cm block 01 ⇒ tag present = 1
tag required = 1

hit.

Send byte no. 11 to CPU
(nothing to replace)

In this case replacement will only take place when entire block is missmatched.

bits in byte offset = $\log_2$(Block Size

Tag directory size = no. of blocks in cache * Tag size

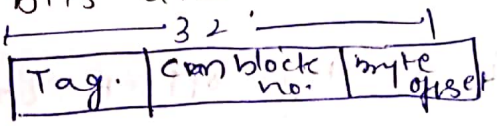
$= 2^{12} * 16\text{-bits}$

$= 64$ k bits

$= 8$ k bytes

Cache memory block no. is also known as (index)

question 2005 consider a direct mapped cache of size 32 kB with block size 32 bytes. CPU generates 32 bit addresses the no. of bits need for cache indexing & the no. of tag bits are?




no. of bits in byte offset = $\log_2(2^5) = 5$ bits

no. of bits in CM block no = $\log_2(2^{10}) = 10$ bits

no. of block = $\frac{\text{cache size}}{\text{block size}} = 2^{10}$

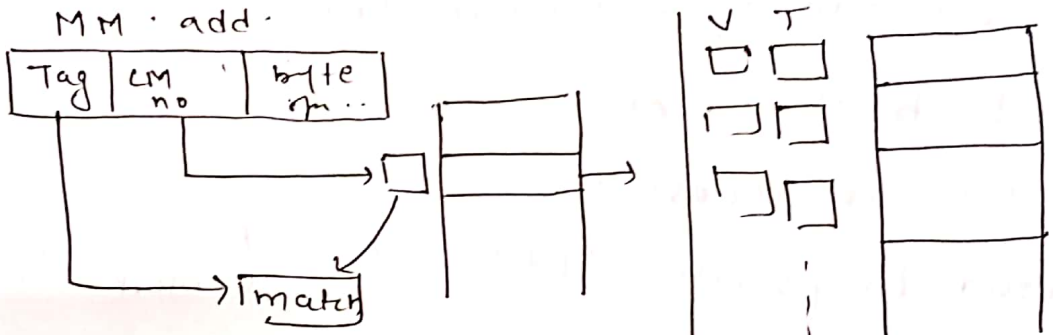
$\therefore$ 32 - 15 = 17 bits for tag.

→ cache initialization.

Initially when cpu is turned on the tag bit may have certain garbage value and also the cache memory.

hence after the CPU send some request the cache may generate 'hit' and CPU will get garbage value

∴ We introduce another bit (valid bit) ~~combi~~including 0' s/1



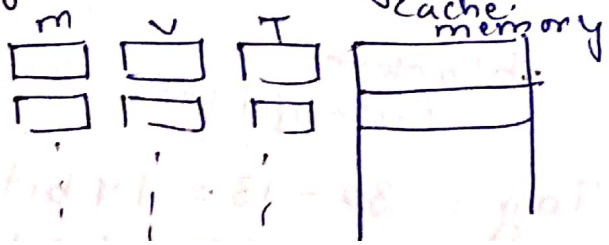

Cache is initialized from '0' on every power on which says it is invalid.

→ Performance improvement of write back cache.

There is block present in cache where CPU has only performed Read hence there is no need to replace the corresponding main memory Location

So only those Locations are supposed to be modified which uses write operation

hence to keep track of this we add modifying bit or dirty bit on the address



If modifying bit is 1 then there is write operation and if modifying bit is 0 there is only read

∴ Tag directory size = No. of blocks in cache * [Tag bits + Valid bits + modifying bit + any extra bit]

~~only those~~ among all the blocks accessed only those blocks are copied back to Main memory which has a dirty block or modification or has a write operation on them.

⇒ For write back cache:

Simultaneous access:

$t_{avg\,read} = t_{avg\,write} = H * t_{cm} + (1-H)(t_{block} + t_{write\,back})$

$t_{write\,back} = x * t_{block}$

$x$ = Fraction of dirty blocks

Hierachical :-

$t_{avg\,read} = t_{avg\,write} = H * t_{cm} + (1-H)(t_{cm} + t_{block} + t_{write\,back})$

q6 P20

Cache size = 8kB

block size = 32B

MM add = 32-bits.

No. of blocks in cache = $\frac{8 \times 2^{10}}{2^5} = 2^8$

∴ 8 bits to store it

~~Block~~ Byte offset = $32 = 2^5 \Rightarrow$ 5 bits

Tag = 32 - 13 = 19 bits

=) Tag director size = $2^8 * (19 + 1 + 1)$

= 5376 - bits.

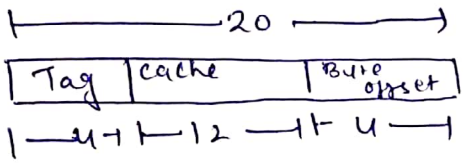
**Q19 P22**

main memory size = $2^{20}$ Bytes of 1 Byte each

⇓ 20 Bits to store this address.

no. of blocks in cache = $2^{12}$.

block size = 16 B = $2^4$ =) 4 bits for byte offset.




mm add =) $(E201F)_{16}$ = convert in binary divide these bits according to block

| E | 201 | F |
|---|---|---|

**Q15 21)**

Cache size = 64 words.

mm size = 256 words =) address in mm = 8-bit.

block size = 16 words = $2^4$ =) word offset = 4-bits

| Tag | cm block no. | word offset |
|---|---|---|
| 2 | 2 | 4 |

no. of blocks in cache = $\frac{64}{16}$ = 4 = $2^2$.

∴ cm block no. size = 2

| block | tag |
|---|---|
| 0 | 10 |
| 1 | 10 |
| 2 | 00 |
| 3 | 01 |

word no. =) 50 =) ~~01000010~~ 00110010

| 00 | 11 | 00 | 10 |
|---|---|---|---|

⇓

block no. 3 in cache.

⇓ tag there = 01, tag required = 00

miss

⇓

add 50 in 3rd block.

132 = 10000100

| 1 | 0 | 00 | 0100 |
|---|---|---|---|

Block no 00
⇓
Tag there = 10
Tag request = 10 ] hit.
⇓
132 present.

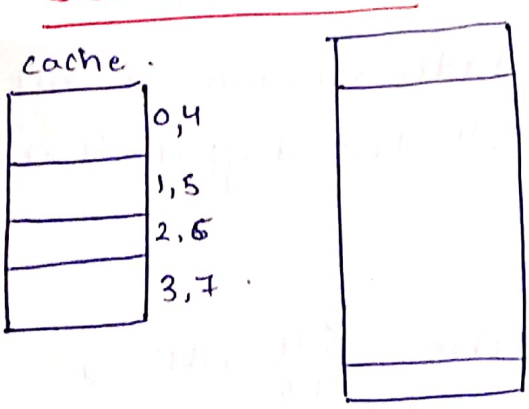
150 = 10010110

| 10 | 01 | 0110 |
|---|---|---|

⇓
goto block 1
⇓
tag there = 10
Tag request = 10 ] hit.
⇓
150 present.

∴ 150 & 132 is present.

## * Set associative:




In previous arrangement if we get successive request of CPU toward a same block will increase the no. of misses in cache.

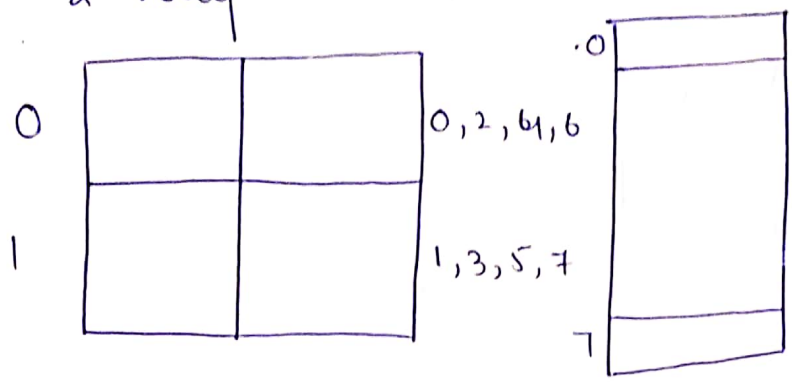
eg: 1, 5, 1, 5, 1, 5 . . .
↓ | | . . . . .
mis | miss | miss
replace | replace

main memory block no. = 1, 5, 1, 5, 1, 5

Cache memory block no = 1 always.

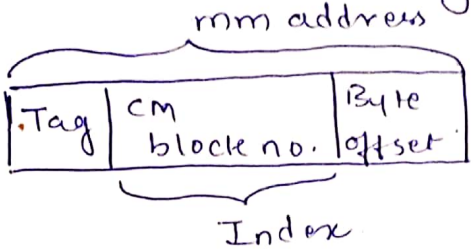
In set-associativity the cache blocks are divided into sets of Blocks.

2-way set associative.

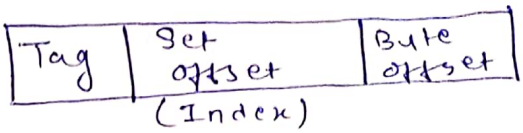



CPU Request:
m m block no:-
1, 5, 1, 5, 1, 5 ...
miss miss hits.

In direct mapping.




In set associative:



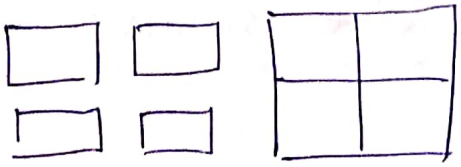

$$\text{no. of sets in cache} = \frac{\text{no. of blocks in cache}}{\text{associativity}}$$

$$\text{bits in set offset} = \log_2(\text{no. of sets in cache})$$

associativity ⟶ k-associative (Generally 2, 4, 8...)

⟶ Tags in set asso... is also stored in same way every block in a set has its own tag.

∴ Tag directory = no. of blocks in cache * [Tag + extra bits]

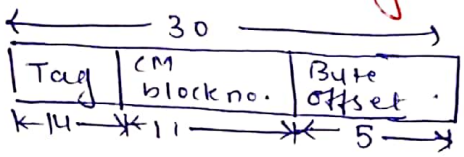


But in set-associativity the no. of competitors per block increases. hence will the no. of bits in tag. i.e ~~no~~. size of tag block in set associative is more as compared to direct ~~to~~ mapping

Question.

mm. add = 30-bits
Cache size = 64kB
block size = 32 Bytes.

In direct mapping:.

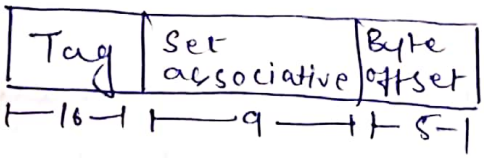



no. of blocks in cache = $\frac{64kB}{32B}$.

$= 2k$

$= 2^{11}$

In 4-way set associativity.




no. of set in Cache = $\frac{\text{no. of blocks}}{\text{associative}} = \frac{2^{11}}{2^2} = 2^9$

Set offset = 9-bits.

Size of tag directory
= no. of blocks in cache * Tag bits

$= 2^{11} * 16$-bits.

Q7 ~~mm - add = 256 kf~~

Cache size = 256 kB

Block size = 32 B

mm - address size = 32 bits.

tag directory contains

address tag + 2 valid bits + 1 modified bit + 1 replacement bit.

⟶ no. of blocks in cache = $\frac{256\ kB}{32\ B}$

$= \frac{2^8 \times 2^{10}}{2^5}$

Q.13 block size = 16 words.

no. of blocks in cache = 256 blocks.

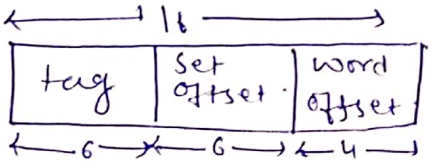
mm blocks = 4096 blocks.

mm size = 4096 * 16 words

= $2^{16}$ word.

=) mm address = 16-bits.

word offset = $\log(2^4) = 4$.




no. of set in cache

$= \frac{\text{no. of blocks}}{\text{associativity}}$

$= \frac{256}{4} = 2^6$

Set offset = 6 bits.

∴ Tag bits = 16 − 10 = 6 bits

Q.9 Cache size = 16kB.

mm-size = $4 \times 2^{30} = 2^{32}$.

address size = 32 bit.

block size = 8 word = 8 x 4 = 32 Byte.

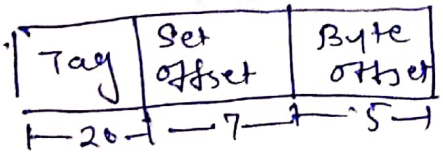
1 word = 32 bit = 4 byte.

∴ blocks in cache = $\frac{2^4 \times 2^{10}}{2^5} = 2^9$.

∴ no. of set in cache = $\frac{\text{no. of blocks}}{\text{associativity}}$

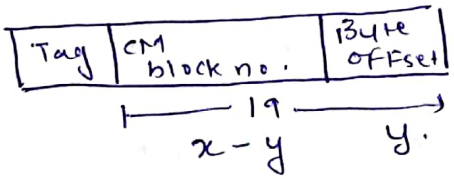
$= \frac{2^9}{2^2} = 2^7$

Set offset = 7 bits



q 22 cache size = $2^{19}$ B.

In Direct mapping

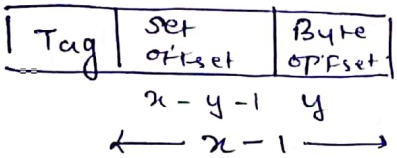



cache size $x = \log_2(\text{cache size})$

CM block no + Byte offset = $\log_2(\text{cache size})$

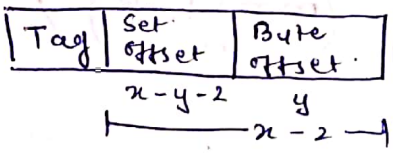
always in direct mapping

2-way:



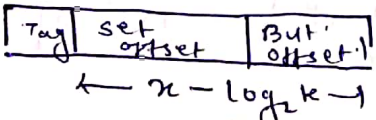

$$\text{no. of sets} = \frac{2^{x-y}}{2} = 2^{x-y-1}$$

4 — way.




$$\text{no. of sets} = \frac{2^{x-y}}{2^2} = 2^{x-y-2}.$$

k — way




$$\therefore \text{Tag} = \text{mm-add} - [(\log_2(\text{cache size}) - \log_2 k]$$

here

cache size = 512 kB = $2^{19}$ B

k = 8 mm-add = 40-bits

$\therefore$ Tag = $40 - [\log_2(512\text{kB}) - \log_2 8]$

= 24 bits.

Q 25 ~~mm address = P bit~~

Size of mm = $2^P$ ~~$\times 2^{10}$ bits~~

$\therefore$ mm address = P bits.

1 Word = $2^w$ bytes. $k = k$.

Cache size = ~~$2^M \times 2^w$ bytes~~ $\cdot 2^N$

Block size = $2^{w+}$

$$\text{Tag} = \text{mm add} - [\log_2(\text{cache size}) - \log_2 k]$$

$$= P - [\log_2(2^N) - \log_2 k]$$

$$= P - N + \log_2 k.$$

---

In direct mapping

*** CM block no. will be mapping = (mm block no.) % no. of block in cache

In set associativity

CM set no. = (mm block no.) % no. of set.

---

Here blocks per set = 2 . i.e 2-way associativity

no. of CM blocks = 2c.

$\therefore$ no. of set = $\frac{2c}{2} = c$

no. of blocks in mm = 2cm

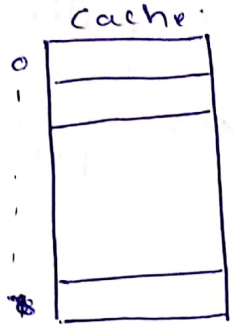
CM set no. = (mm block no.) % no. of sets in cache.

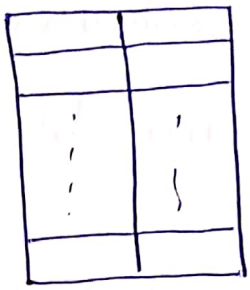
$$= \boxed{k \% c}$$

# * Fully associative mapping

Direct: | Tag | CM block | byte offset |

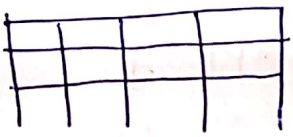
2way | Tag | CM block | Set offset | Byte offset |



If all the ~~sets m~~ blocks of main memory mapped to a single set in cache then it will be fully associativity.

4way | Tag | Set offset | Byte offset |



∴ For k way k sets will be there.

∴ Set offset = CM block / associati

But at the end when k be equal to no. of blocks in cache then it is fully associative.

and Set offset will need 0 bit.

i.e there is no set offset part in cache.

| Tag | Byte offset |

Set offset = CM block / associativity = 1

= '0' bit

⟶ A fully associative cache contain $2^{12}$ blocks If main memory contains $2^{16}$ blocks then the size of memory (In kb) required at cache controller to store the tag directory

Tag = mm block number.

| MM block no | Byte offset |
|---|---|

no. of blocks in mm = $2^{16}$

mm block number = 16-bits.

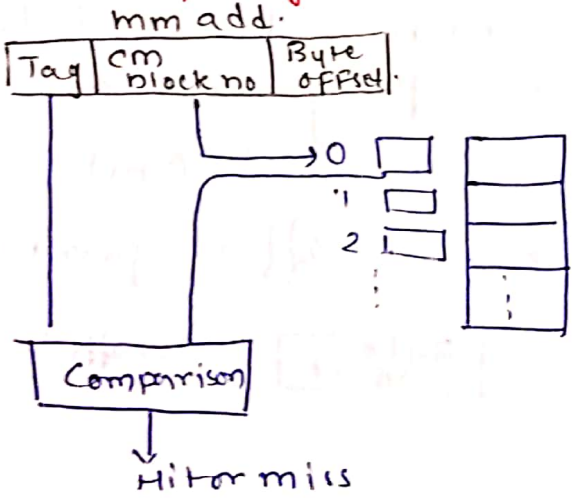
Tag directory size = $2^{12} * 16$-bits.

= $2^{16}$-bits

= 64K bits.

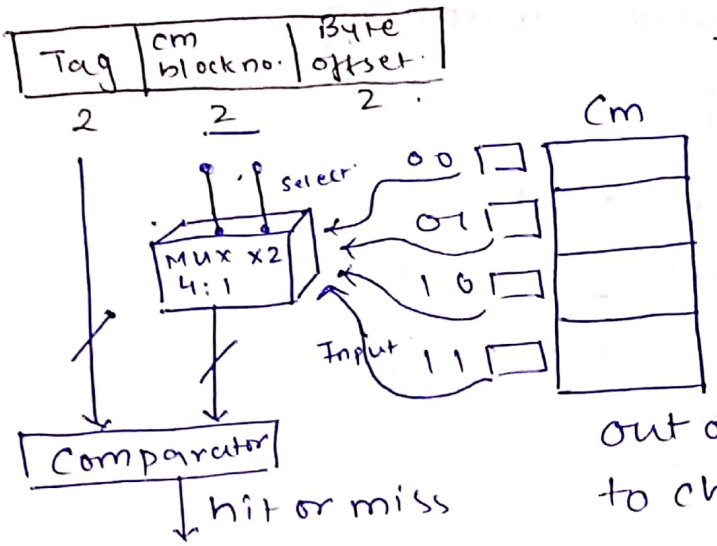
* Hardware implementation of Mapping.

→ Direct mapping:-




Assume; 6 bit

| Tag | cm block | Byte offset |
|---|---|---|
| 2 | 2 | 2 |




There will be 2 similar mux required as mux takes only one bit at a time.

out of 4 tags we need to choose 1

∴ In general:

A mux can forward only 1 bit of tag.

No. of comparators = 1

~~No. of~~ Size of comparator := Size of tag in address

(or ~~tag~~ bits in address)

No. of Mux = Size of tag in address.

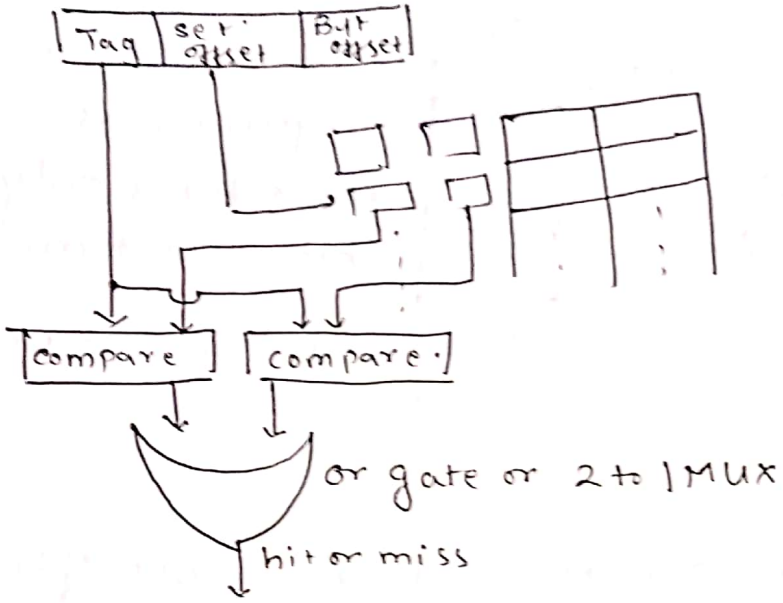
= Size of comparator.

Size of Mux = (No. of blocks in cache : 1)

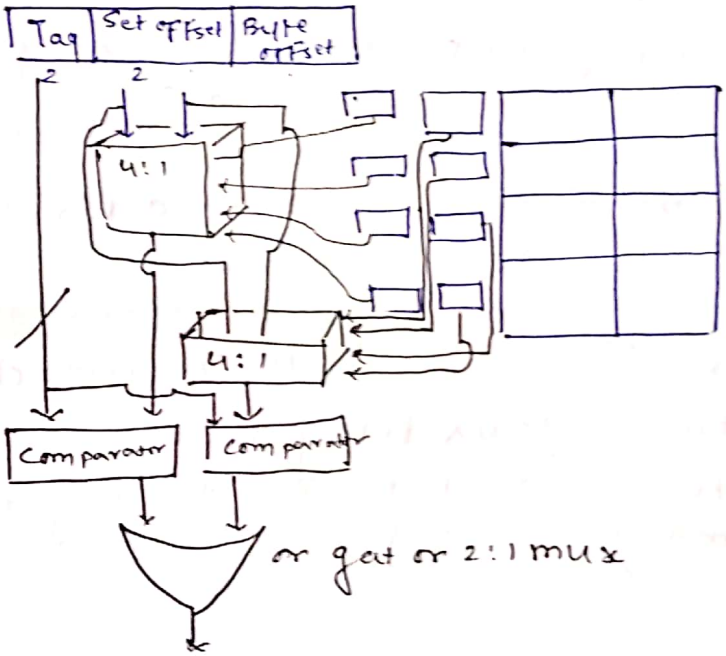
Hit latency time = MUX delay + Comparator delay

(Time required to declare hit or miss) (for selecting tag) (for tag comparison)

→ In set associative mapping:




eg:




For k-way set associative mapping

No. of comparators = k

Size of comparators = tag-bits

No. of MUX = k * tag-bits

(For tag selection)

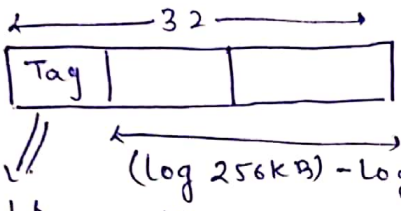
Size of MUX = No. of sets in cache : 1

Size of MUX = k : 1

(For hit or miss)

hit latency time = Mux delay (For selecting tag) + Comparator delay + Mux-or gate delay (hit or miss)

(Q1)




32 - 16 = 16-bits = 18 - 2 = 16.

∴ tag bits = 16.

∴ 4 comparators of size 16 bits each will be needed.

k = 4

no. of compatator = k = 4

bits of comparator = tag bits = 16.

Q11

Cache size = 32kB.

2 way

block size = 32B

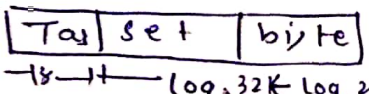
mm add = 32-bits.

$h_1$

2 to 1 Mux ⇒ 0.6 ns.

k-bit comparator = k/10 nsec.

We need tag size first.




15 - 1 = 14.

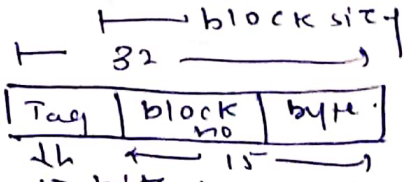
Cache size = 32kB

direct mapped.

block size = 32B

mm add = 32.

$h_2$.




no. of blocks in cache = $\frac{32\,kB}{32} = 2^{10}$

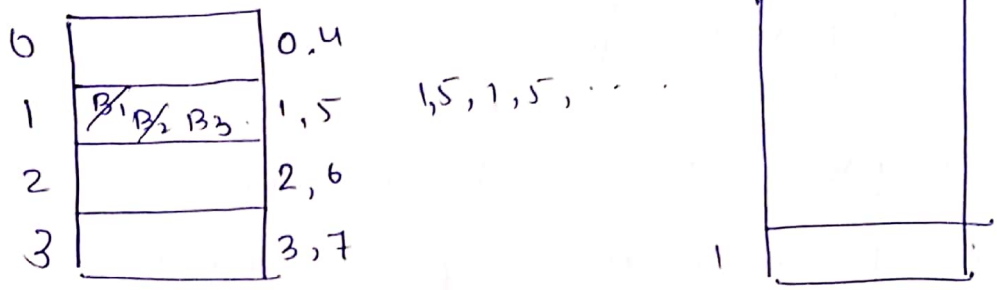
no. of sets = $\frac{2^{10}}{2} = 2^9$

$h_1 = 0 + \frac{18}{10} + 0.6 = 2.4$ ns.

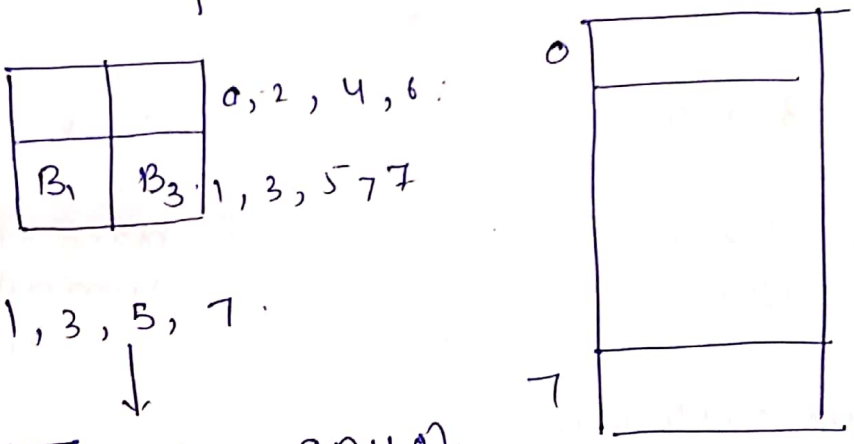
$h_2 = 0 + \frac{17}{10} + = 1.7$ ns.

## * Cache block replacement:

In direct mapping if there occurs any replacement theres only one choice to make. Hence no replacement policy needed




in two-way associativite.




1, 3, 5, 7.

↓

To replace any of $B_1$ or $B_3$

Replacement policy is used.

Policies:

(i) FIFO

(ii) Optimal

(iii) LRU

consider a direct mapped cache with 8 cache blocks (0 - - - 7). If the memory block request are in the following order.

3, 5, 2, 8, 0, 63, 9, 16, 20, 17, 25, 18, 30, 24, 2, 63, 5, 82, 17, 24.

Which of the following memory blocks will not be at the end of of sequence in cache

a) 3 , b) 18 c) 20 d) 30

| | |
|---|---|
| 0 | ~~8~~ ~~0~~ 16 24 |
| 1 | ~~9~~ ~~17~~ ~~25~~ 17 |
| 2 | ~~2~~ ~~18~~ ~~2~~ 82 |
| 3 | 3 |
| 4 | 20 |
| 5 | 5 |
| 6 | 30 |
| 7 | 63 |

Q14
21)

| | | | | |
|---|---|---|---|---|
| 0 | ~~0~~ 48 | ~~4~~ 32 | 8 | ~~216~~ 72. |
| 1 | 1 | 133 | 129 | 73 |
| 2 | | | | |
| 3 | ~~255~~ 155 | 3 | 159 | 63 |

0, 4, 8, 216, 8, 48, 32.

∴ 216 is not there.

Should be moded by k in k associative.

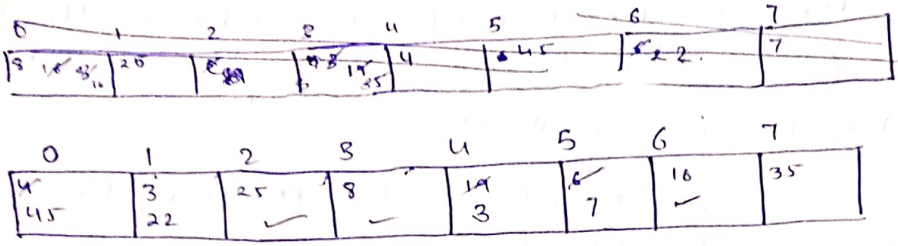
Q17)

| | | |
|---|---|---|
| 0 | ~~8~~ ~~0~~ 8 | 12 |
| 1 | | |

8/1, ~~12~~, ~~0~~, 12, 8

↑ main

Q16



Q20

Byte addressable main memory

mm size = $2^{16}$B ⇒ mm add = 16 bit.

direct mapped cache.

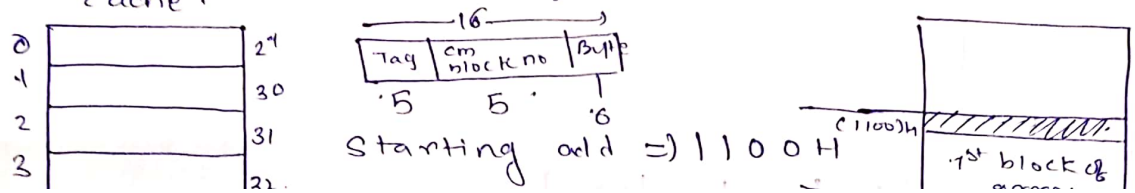
no. of blocks in cache = 32.

block size = 64B.

50×50 2-d array ⇒ array size = 2500 B

Base address of array = (1100H)

cache.



∴ in 2nd access 8 misses will occur (1 → 8) and upto 32 there will be hits again from (33 → 40) misses will occur

∴ total misses in 2nd access

= 8 + 8 = 16,

∴ total 40 + 16 = 56 misses.

and at everytime line 4 to line 11 will be replaced

GATE 2006. A CPU has 32KB direct mapped cache with Q. 128 byte block size suppose A is 2 dimensional array of size 512 × 512 with elements that occup 8 Bytes each consider the following two C code segments.

```
P1. For(i=0; i<512; i++)
    {
      for (j=0; j<512; j++)
        {
          x += A[i][j];
        }
    }
```

```
P2. For(i=0; i<512; i++
    {
      for(j=0; j<512; j++)
        {
          x+ = A[j][i];
        }
    }
```

P1 & P2 are executed independently with same initial state ~~mai~~ mainly 'A' is not in cache & i, j, x are in registers let the no. of cache misses experienc by P1 be $m_1$ & that for P2 be $m_2$.

(i) the value of $m_1$ is ?

(ii) the value of $m_1/m_2$ is?

→ Direct mapping.

Cache size = 32 kB ⇒ blocksize = 128 B.

A : 512 × 512 (each of 8 B)

no. of blocks in ~~array~~ cache = $\frac{32 kB}{128 B} = 2^k = 256$

Array size = $(2^9 * 2^9) * 2^3 B = 2^{21} B$.

$P_1$

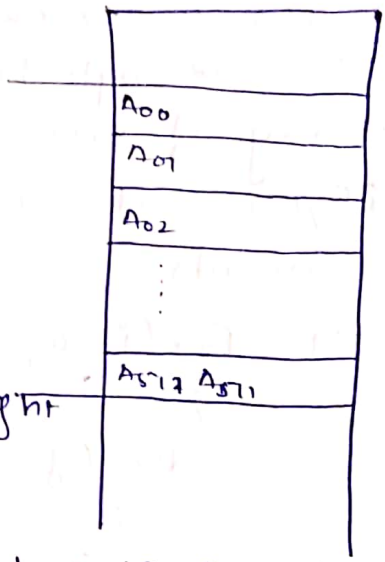
$A_{00}$ $A_{01}$ $A_{02}$ . . . $A_{0511}$

.

.

.

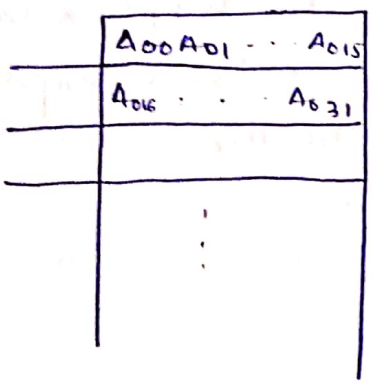
$A_{5110}$ . . . $A_{511511}$




In one block 128 B can be brought at every miss

∴ no. of array elements in 1 block = $\frac{128 B}{8 B} = 2^4 = 16$

∴ at $A_{00}$ 1 miss and 15 elements will be no miss




When CPU access $A_{00}$ a block of 16 element $A_{00}$ - $A_{015}$ is copied in cache ie 1 miss for 16 elements.

∴ for Array $2^9 \times 2^9 \Rightarrow 2^{18}$ elements

no. of misses = $\frac{2^{18}}{2^4} = 2^{14} = 16384$

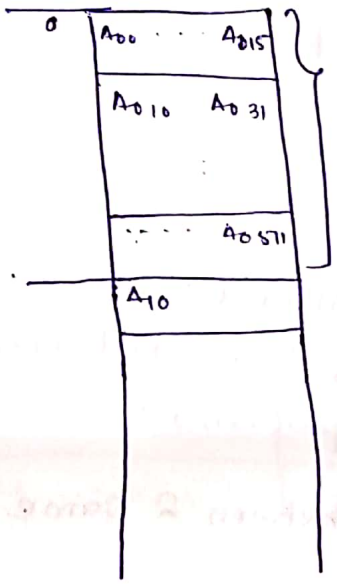
no. of elements in 1 row = 512.

Size of elements = $512 \times 2^3 = 2^{12}$ B.

No. of blocks required for 1 row = $\frac{2^{12}\,B}{2^7} = 2^5 = 32$ Blocks

∴ single row will take 32 blocks in cache.

after 32 blocks another row will start.




∴ no. of rows can be stored in cache $\Rightarrow \frac{256}{32} = 2^3 = 8$

∴ $A_{80} - A_{815}$ will replace $A_{00} - A_{015}$ on access.

∴ For each element of array.

$\Rightarrow$ 1 miss

$M_2 = 2^{18}$ misses.

∴ $M_1/M_2 = \frac{2^{14}}{2^{18}} = 1/16.$

2014 Gate → an access sequence of cache block addresses is of length N and contains 'n' unique block addresses the no. of unique block addresses between two consicutive access to same block is bounded above by k. What is the miss ratio if the access sequence is passed through a cache of associativity $A \geq k$. excersing LRU replacement policy.

(A) n/N (B) 1/N (C) 1/A (D) k/n

Sequence of access = N

i.e 'N' times CPU is accessing cache.
and out of them 'n' blocks are only unique remaining are repeated.

unique blocks

1 | 3 4 2 3 4 2 3 4 2 | 1.

no. of unique blocks bet? two consiqutive same block is < k.

∴ unique block between two consequetive same block should be maximum k-1

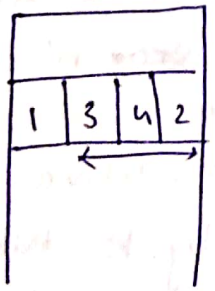
associativity A ≥ k

assume k=4 & A=4

no. of misses that can be encountered
= n + extra misses of repeated accesse because of replacement.

as A ≥ k and unique symbols between 2 same symbols < K

then in A associative blocks.
there will always be hits among the replacement of repeated accesses.
∴ extra misses = 0




Hence total miss encountered will be n.

miss ratio = $\frac{n}{N}$

only unique no. will be having misses & everytime the repeated access is there. it will be hit.

→ Cache miss panelty :-

The amount of time CPU spends to service the cache miss is known as cache miss panelty.

Which means bringing the missed block from main memory to cache.

assume,

No. of cycles needed to send address to main memory = 1

No. of cycles needed to access main memory = 10

No. of cycles needed to ~~send~~ transfer 1 cell data to CM = 1

| Block size | mm cell size | miss panelty |
|---|---|---|

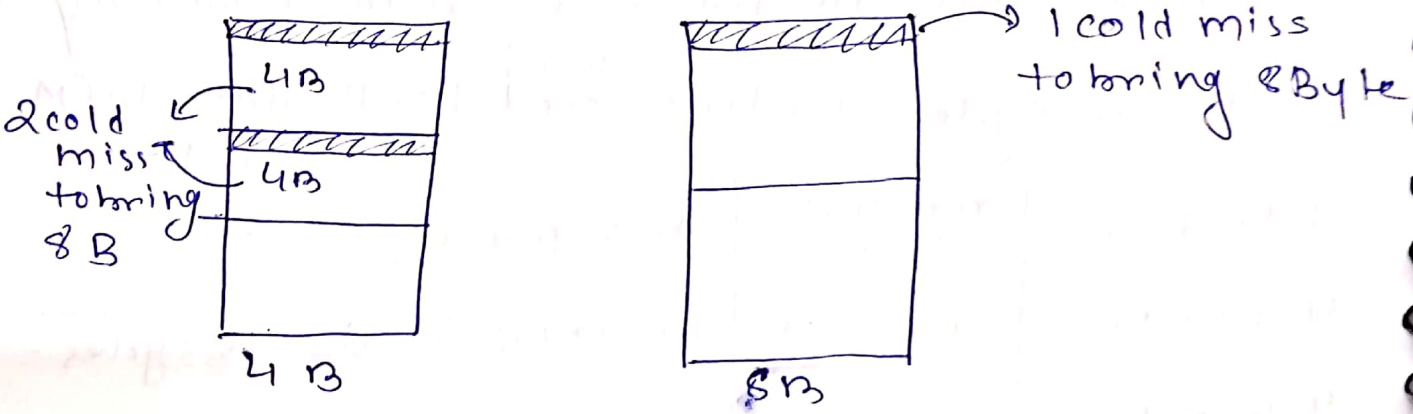
# Types of cache miss

→ Cold or compulsory misses.

If a cache miss occur due to first time access of any block.

to Reduce compulsory misses. increase the block size.

the more no. of block you can bring at '1' miss the lesser will be the compulsory miss.




---

→ Capacity miss.

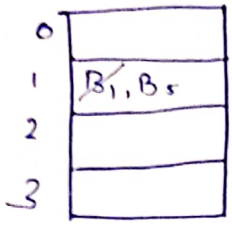
→ If the cache miss occurs because of the capacity of the cache which means when the cache memory is full

→ It should not be a cold or compulsory miss

To reduce capacity misses increase the cache size.

→ conflict miss.

If the cache miss occurs because of the mapping constraint (Tag missmatch).

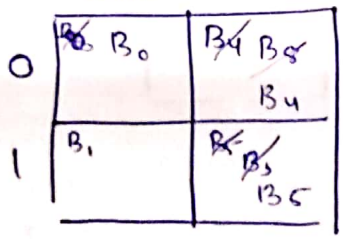



1, 5, 1
cold cold conflict.

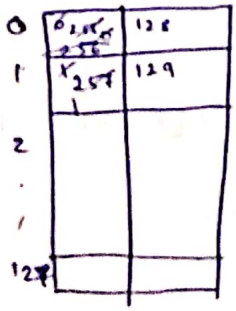
Ex: no. of blocks in cache = 4.
2way set associative
no. of set in cache = 4/2 = 2.
With LRU replacement.
CPU requests for mm block

0, 4, 0, 8, 0, 4, 1, 5, 1, 3, 1, 5
cold cold Hit cold Hit conflict cold cold Hit cold Hit conflict Capacity

Cm.




Q23 no of sets = 256/2 = 128.




| | 1st time | 2nd time |
|---|---|---|
| 0 | Cold | conflict |
| 128 | Cold | Hit |
| 256 | cold | conflict |
| 128 | Hit | Hit |
| 0 | Conflict | conflict |
| 128 | Hit | Hit |
| 256 | conflict | conflict |
| 128 | Hit | Hit |

| | 1st time | 2nd time |
|---|---|---|
| 1 | cold | conflict |
| 129 | cold | Hit |
| 257 | cold | conflict |
| 129 | Hit | Hit |
| 1 | conflict | conflict |
| 129 | Hit | Hit |
| 257 | conflict | conflict |
| 129 | Hit | Hit |

1st time — 4 conflict miss
2nd — 10th time → 8 conflict misses.
∴ Total = 4 + (9 × 8) = 4 + 72 = 76

→ Goal of using cache memory:

1. minimize access time
2. Maximize hit rate.
3. Minimize miss panelty

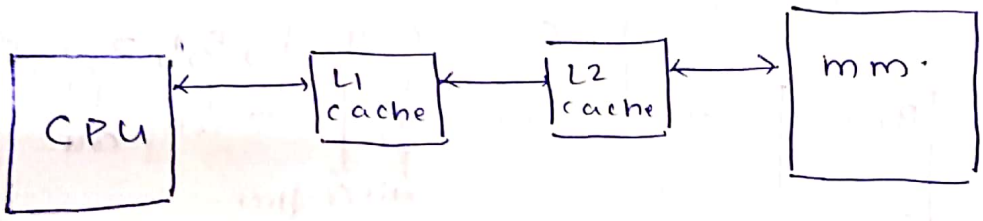
But to have minimum access time we use smaller cache.

And to have maximize hit rate. we need larger cache.

Hence the solution for this conflicts is to use multilevel cache.

to minimize miss panelty use non-block cache, multiport, multibank memories.

→ Multi-level cache:-




Simultaneous access:

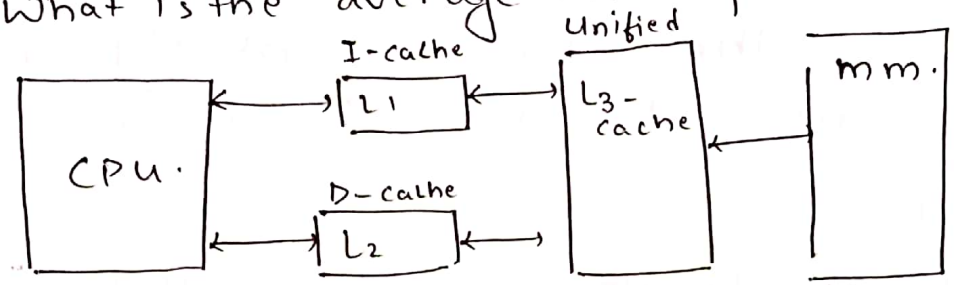
$$t_{avg} = H_1 t_1 + (1-H_1)\left[H_2 * t_2 + (1-H_2) * t_{mm}\right]$$

Hierarchical.

$$t_{avg} = H_1 * t_1 + (1-H_1)\left[H_2 * (t_2+t_1) + (1-H_2)(t_1+t_2+t_{mm})\right]$$

$$= t_1 + (1-H_1)\left[t_2 + (1-H_2)\, t_{mm}\right]$$

question: Consider a multilevel memory Hierarchi as shown below. the hit ratio of $L_1, L_2, L_3$ and main memory are 0.8, 0.85, 0.9 & 1 Respectively the access times of respective memories are. 10 ns, 10 ns; 50 ns & 500 ns. among total memory refrences 60% are for data access. What is the average memory access time.




$t_1 = 10$ ns.
$t_2 = 10$ ns
$t_3 = 50$ ns.
$t_{mm} = 500$ ns

$h_1 = 0.8$
$h_2 = 0.85$
$h_3 = 0.9$
$h_{mm} = 1$

$t_{avg\ inst^n} = t_1 + (1-H_1) * [t_3 + (1-H_3) * t_{mm}]$

$= 10 + 0.2 * [50 + 0.1 * 500]$

$= 30$ ns.

$t_{avg\ data} = t_2 + (1-H_2) * [t_3 + (1-H_3) * t_{mm}]$

$= 10 + 0.15 * [50 + 0.1 * 500]$

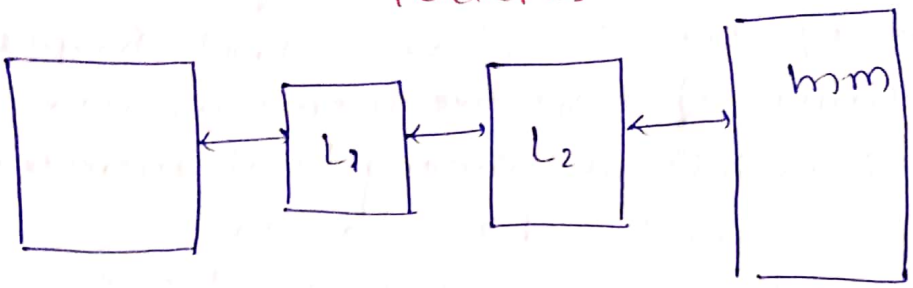
$= 25$ ns.

$\therefore t_{avg} = (0.6 * 25) + 0.4 * 30$ ns

$= 15 + 12.$

$= 27$ nsec.

→ Inclusion vs. Exclusion Policies.




~~The conte~~ The entire content of $L_1$ should be present in $L_2$ also this is inclusion policy.

→ That means if there's a miss in L1 & hit in L2 then bring the block from $L_2$ to $L_1$

→ Similarly If there's a miss in L1 & $L_2$ both then the block will be taken from mm to $L_2$ & then from $L_2$ to $L_1$

Exclusion policy :—

The entire content of $L_1$ should not be present in $L_2$ necessarily this is exclusion policy.

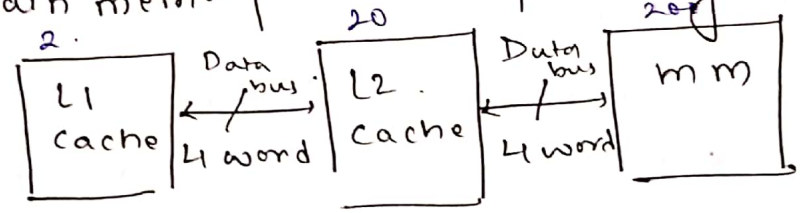
→ IF miss in $L_1$ and hit in $L_2$ bring blocks from $L_1$ to $L_2$.

→ IF miss in $L_1$ & also in $L_2$ then bring the block from main memory to $L_1$ & ~~send th~~ copy replaced block of $L_1$ into $L_2$.

The replaced block is called victim block and ~~vict~~ $L_2$ cache is known as victim cache. as $L_2$ gets populated by $L_1$'s victim blocks.

There might be difference in block size of both $L_1$ & $L_2$ then the amount of data copied from $L_2$ to $L_1$ will be according to $L_1$ block size.

a computer system has an $L_1$ cache & an $L_2$ cache and a main memory unit connected as shown. the block size in $L_1$ is 4 words the block size in $L_2$ is 16 words the memory access times are 2ns; 20ns; 200ns. for $L_1$ cache, $L_2$ cache & main memory unit respectively.




qui) When ther is a miss in $L_1$ cache & Hit in $L_2$ cache a block is transfered from $L_2$ to $L_1$ cache What is the time taken for this transfer.

ii) When there is a miss in both $L_1$ & $L_2$ cache First a block is transfered from main memory to $L_2$ and then a block is transfered from $L_2$ to $L_1$ cache. What is the total time taken for those transfers.

Databus transfers 4 words at time.

1) Here only transfer time is asked.

4 word in $L_1$
16 in $L_2$.

∴ 4 words block is required Will be directly transfered to $L_2$.

reading from $L_2$ = 20 ns
writing to $L_1$ = 2 ns.
22.

(ii) any no. of words can be accessed from mm.

∴ ~~from~~ block from mm to L2 =) 4 * [200 + 20]

= 880 nsec.

from $L_2$ to $L_1$ =) 22 ns.

∴ 880 + 22 ns = 902 ns.

## → Secondary Memory :-

### ↳ Magnetic Disk :-

Here storage is known as recording. and every platers has & both serface as recording serface

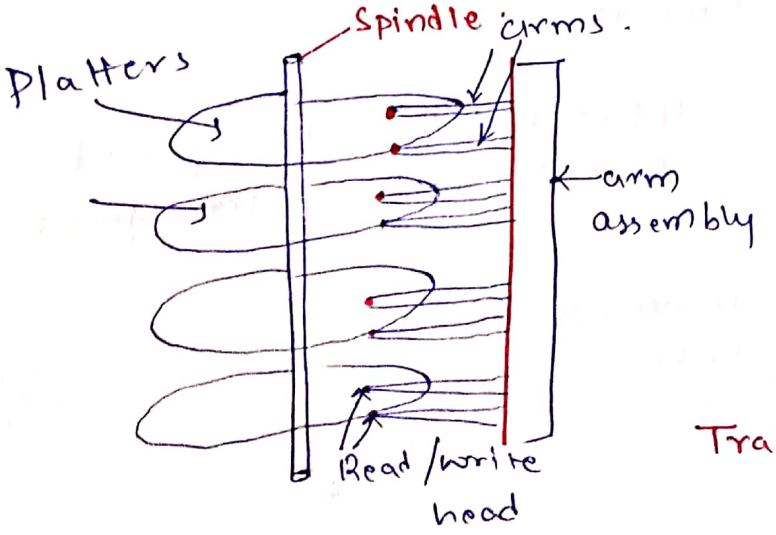
For every surface will require read/write heads which points at the surfaces.

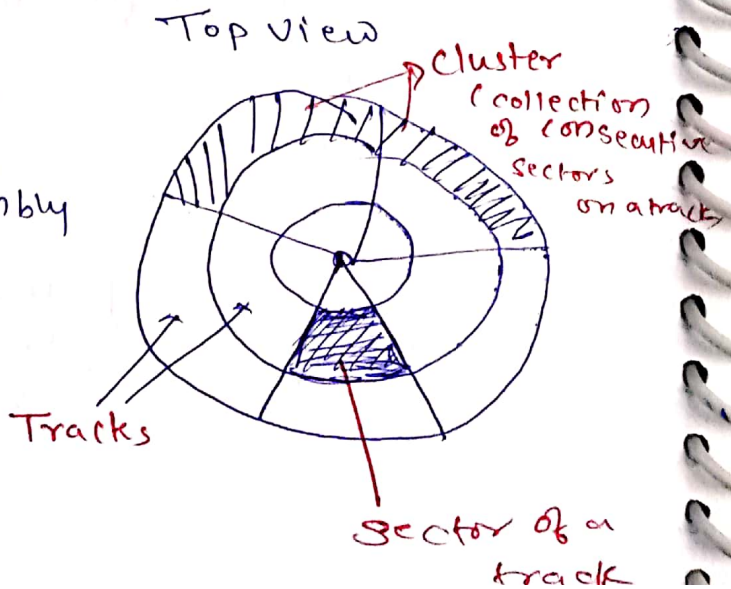
They are attached through arm assembly

every disk will be divided into concentric tracks and these tracks will be divided into sectors

Here sector of a track is Sector within particular track.

Collection of consicutive sectors is cluster.



# Storage.

Used in mordern days.

- Constant sector capacity or variable storage density
- variable sector capacity. or constant storage density.

Disk capacity = 2 * no. of plater in disk * no. of tracks per surface * no. of sector per track * 1 sector capacity.

For constant sector capacity.

Here smallest unit of disk is sector. which can be read or written at once.

hence address will be generated for sectors i.e every sector has one address

→ **Disk access time:**

In one access time is to access 1 sector.

∴ access time = Seek time + rotational latency + 1 sector transfer time + extra time.

- Seek time: (header should reach track on which sector is present)
- rotational latency: (to reach to the sector i.e the r/w header reaches to sector)

Seek time: Time required to move the arm over desired track.

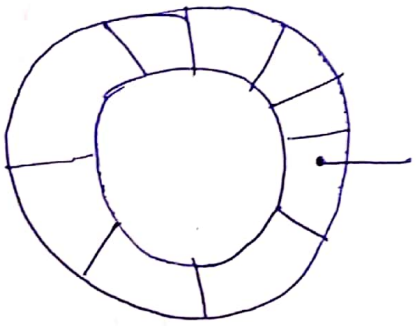
rotational latency = Time required to rotate the desired sector under read/write head.

1 sector transfer time = Time required to read or write one sector

average rotational latency = $\frac{1}{2}$ disk rotational time

Assume:- no. of sectors per track = 10

1 rotational time = 100 msec.



IF specific position of r/w head given calculate specific rotational latency otherwise use average rotational latency.

In one rotation 1 track can be transfered.

$$\therefore \text{1 sectors transfer time} = \frac{\text{1 rotation time}}{\text{no. of sectors per track}}$$

question consider a disk with 16 platter · each with 2 recording surfaces; 1k tracks per surface 2k sectors per track and 2048B per sector. disk rotates with 3000 rpm, seek time of disk is 10 ms

i) Find capacity of disk in GB

2) Find the no. of bits required for disk addressing

3) Find the disk access time.

4) what is disk transfer rate.

1) $16 \times 2 \times 2^{10} \times 2^{10} \times 2^{2} \times 2^{10}$

$2^{4} \times 2^{1} \times 2^{30} \times 1$

$2^{37} \times 2^{30} = 128\,GB$.

(i) disk capacity = $2 * 16 * 1K * 2K * 2048B$

$= 2^{37} B = 128 GB$

(ii) no. of sectors $= 2 * 16 * 1K * 2K$

$= 2^{26}$

address = 26 bits.

disk access time = seek time + (average) rotational latency + 1 sector transfer time.

$= 10 ms + \frac{20}{2} + \frac{20 ms}{2K}$

rotational spped = 3000 rotation

for 3000 rotation $\longrightarrow$ 60 sec

1 rotation $= \frac{60 sec}{3000}$

$= 20 m sec$

$\therefore$ disk access time $= 10 + 10 + 0.01 = 20.01 msec$

now 0.01 msec required to transfer $\rightarrow$ 2048 Bytes.

then in 1 ms $\longrightarrow$ 204800 Bytes.

$\rightarrow$ 200 kB/msec.

= ~~200 kBytes~~.

then in 1 sec $\longrightarrow \frac{2048}{0.01 \times 10^{-3} sec} = 200 MB/sec.$

If n no. of sectors to be transfered

i) If all the sectors are stored sequencially

Total disk access time = seek time + rotational latency + (n * 1 sector transfer time)

2) If sectors are stored on random locations

Total disk access time = $n * [$seek time + rotational latency + 1 sector transfer time$]$

Q5

⇒ 6000 rotation ⟶ 60 sec.

∴ rotation latency = 10 msec

$= 100 * \left[10\text{msec} + \frac{10}{2} + 0\right]$

$= 100 \times 15$ msec.

$= 1500$ msec

$= 1.5$ sec.

Q7

$\text{1 rotation time} = \frac{60 \times 1000\text{ ms}}{10000} = 6\text{ms}$

$\text{total file transfer time} = 2000 * \left[4 + \frac{6}{2} + \frac{6}{600}\right]$

$= 2000 * [4 + 3 + 0.01]$

$= 2000 * 7.01$ ms.

$= 14020$ msec

Q8

$\text{1 rotation time} = \frac{60 \times 1000}{15000} * = 4$ ms.

Avg rotation latency = 2 ms.

seek time = 4ms. = 2 × 2 ms.

To transfer $50 \times 10^6$ B ⟶ 1 sec.

512 B ⟶ $\frac{1}{50 \times 10^6} * 512$.

1 sector transfer time = 0.01 msec.

controler transfer time = $10 \times 0.01$

$= 0.1$ ms

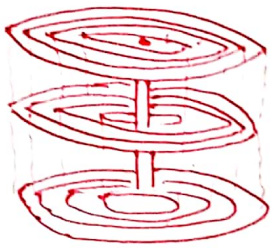
∴ 1 disk accesstime = seek + rotation latency + 1 sector transfer time + controller.

$= 4 + 2 + 0.01 + 0.1 = 6.11$ m sec.

→ Cylinder

Collection of the tracks of the same radius from all the surfaces forms a cylinder.

The data storage in the disk is performed cylinder vise to save the seek time.



no. of cylinders in disk = no. of tracks per surface

Q3 no of sectors per track = 63.

no of plater = 10 (each with 2 surfaces)

" " cylinders = 1000

add of sector ⇒ <c, h, s>

c → Cylinder, h → Surface, s → sector no.

0th sector = <0, 0, 0>

1st sector = <0, 0, 1>

<400, 16, 29> = ?

0th cylinder — 20 surfaces (0, 19)

0th sector — <0, 0, 0>

62nd Sector = <0, 0, 62>

63rd sector = <0, 1, 0>

<0, 1, 62>

∴ no. of sector per cylinder = 1260

Q25 9th ——→ <0, 19, 62>

1260th <1, 0, 0>

⋮

<1, 19, 62>

no. of sectors in 400 cylinders (0-399) = 400 * 1260
= 504000

no. of sector in 16 surface = 16 * 63 = 1008.

+29.

=) 504000
+ 1008
+ 29
---
5050 37

no. of sectors per cylinder = $n_c$.

no. of sector per track = $n_t$.

For address =) (c, h, s)

Sector no. = ((c * $n_c$) + (h * $n_t$) + s.

C = (Sector no) // nc.

h = (Sector no % nc) / nt.

s = (Sector no % nc) % nt.

∴ 1039 nc = 1260 nt = 63

1039 / 1260 =) 0

(1039 % 1260) / 63 =) 16

(1039 % 1260) % 63 =) 31

Q6 no. of surfaces = 16 (0-15)
no. of cylinders = 16384 (0-16383)
no. of sectors per track = 64.

Sector capacity = 512B
File 42797 KB
Starting add = <1200, 9, 40>

Sector no. of 1st sector = (1200 × 1024) + 9 * 64 + 40
no. of sector per cylinders = 16 * 64 = 1024

Sector no. of 1st sector = 1229416.

no. of sector required to store the file.

$$= \frac{42797\,KB}{512} = 85594 \text{ sectors.}$$

Sector no. of last sector of file = 1229416 + 85594 − 1 = 1315009

C = 1315009 / 1024 = 1284.
H = (1315009 / 1024) % 64 = 3.
S = (1315009 / 1024) % 64 = 1.

∴ add => <1284, 3, 1>

→ Parallel Processing

technique for simultaneous data processing

types.

- Vector processing
- Array processing
- Pipeline "

→ Flynn's classification of computers

1) SISD :- Single Instruction stream
Single Data stream

at a time single Instruction is fetched
at a time single Instruction is Executed.

2) SIMD: Single Instruction stream Multiple data stream

at a time single Instruction is fetched but multiple Instruction is Executed.

eg:- Pipeline.

3) ~~MISD~~ MISD: Multiple Instruction stream
Single data stream.

at a time multiple Instruction is fetched but single Instruction is executed.

4) MIMD: Multiple Instruction stream multiple data stream

multiple Instructions are fetched & multiple Instructions are executed.

eg: Super scalar System. which

ILP — Instruction level parallelism.

Superscalar systems uses multiple pipelines

## ⇒ Pipelining :-

→ The technique to divide an operation into sub-operations so that the suboperations can be performed in parallel over ~~m~~ different inputs.

→ Pipelining can be implemented only when the same processing should be applied over multiple inputs.

→ In pipelining entire processing is divided into multiple suboperations

→ Each suboperation is performed in a separate unit known as segment or pipeline stage.

→ all the segments can perform their respective suboperations in parallel on different different inputs.

→ entire processing performed on an input is known as a task.

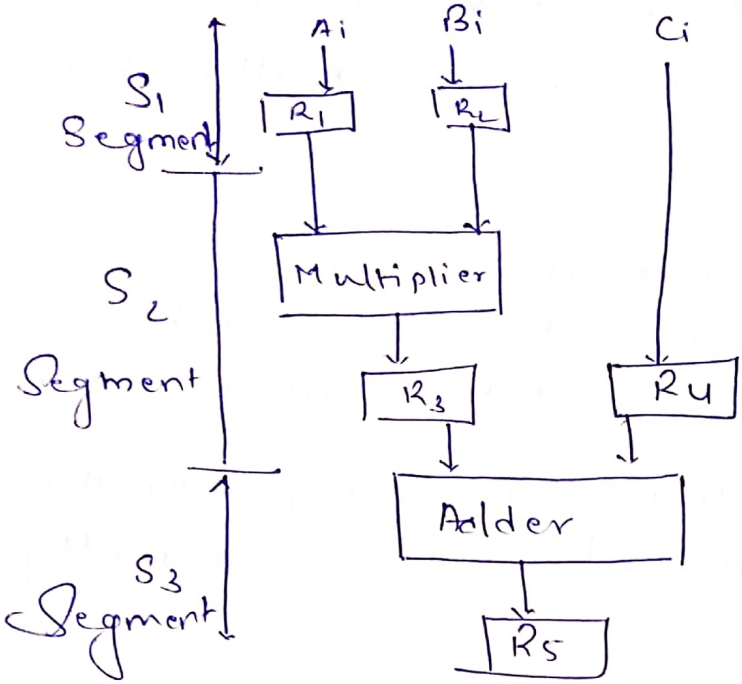
Consider an example.

$A_i * B_i + C_i$ for $I = 1$ to $5$

Seg 1: $R_1 \leftarrow A_i$ ; $R_2 \leftarrow B_i$

Seg 2: $R_3 \leftarrow R_1 * R_2$ , $R_4 \leftarrow C_i$

Seg 3: $R_5 \leftarrow R_3 + R_4$




| Segments / Clock cycle | Seg 1 | | Seg 2 | | Seg 3 |
|---|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ |
| 1 | $A_1$ | $B_1$ | | | |
| 2 | $A_2$ | $B_2$ | $A_1 * B_1$ | $C_1$ | |
| 3 | $A_3$ | $B_3$ | $A_2 * B_2$ | $C_2$ | $A_1 * B_1 + C_1$ |
| 4 | $A_4$ | $B_4$ | $A_3 * B_3$ | $C_3$ | $A_2 * B_2 + C_2$ |
| 5 | $A_5$ | $B_5$ | $A_4 * B_4$ | $C_4$ | $A_3 * B_3 + C_3$ |
| 6 | - | - | $A_5 * B_5$ | $C_5$ | $A_4 * B_4 + C_4$ |
| 7 | - | - | | | $A_5 * B_5 + C_5$ |

| inputs | no. of cycles in non-pipeline | no. of cycles in pipeline |
|---|---|---|
| $n = 5$ | $3 * 5 = 15$ | 7 |
| $n = 6$ | 18 | 8 |
| $n = 7$ | 21 | 9 |

# Pipeline cycle time.

The amount of time in which all the segment can perform their respective suboperations.

## General consideration about pipeline

Consider a k-segment pipeline with pipeline cycle time $t_p$ To perform 'n' no. of tasks.

Time required to perform first task equal to

$= k * t_p$

Time required to perform remaining (n-1) tasks =

$= (n-1) t_p$.

∴ Time required to perform n tasks $= (k+n-1) * t_p$

Total no. of cycles — cycle tim

→ Consider a non pipeline System which takes $t_n$ time to perform 1 task

∴ Time required to perform n task in non pipeline System $= n * t_n$.

Performace of pipeline is given by speed up ratio is $\frac{\text{Slower technique}}{\text{faster technique}}$

$$\text{Speed up } (S) = \frac{\text{Non-pipeline time}}{\text{Pipeline time}}$$

$$S = \frac{n * t_n}{(k+n-1) t_p}.$$

as the no. of task increases such that $n >> k$.

∴ Time required to perform n tasks = "$n * t_p$" as $k-1$ can be ignored

∴ $S_{max} = \frac{t_n}{t_p}$ Smax is also known as Sideal.

In ideal condition the initial $k-1$ task or $k-1$ cycles are ignored as this much time is negligible at higher rate.

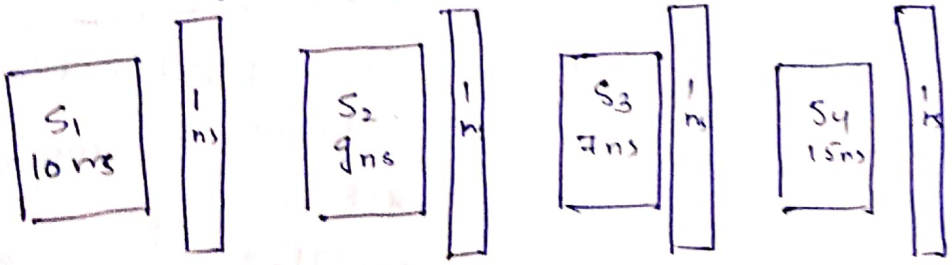
If non pipeline & pipeline system take same time to perform 1 task

∴ $t_n = k * t_p$

$t_n = k * t_p$

$\boxed{S_{max} = k}$

In pipeline system we use intermediat Buffer register to store output of each segment and to synchronous.




~~$t_p = max(segmentdelay)$~~

$t_p = max(segment\ delay) + Reg.\ delay$

$= max(10, 9, 7, 15) + 1$

$= 15 + 1 = 16\ ns$

Intermediate Reg or buffer used in pipeline systems only.

To get the better performance, design the pipelin in such a way that all the segment should take almost equal time.

Q.1 P30

k = 4 segments
n = 1000
$t_p$ = max(seg delays) + Reg. delay
= 160 + 5 = 165 ns.

∴ cycle time = $(k+n-1)\,t_p$ = (4 + 1000 − 1)165 ns.
= 165495 ns.
= 165.5 μs

Note:

Speed up

If value of n given

$$S = \frac{n * t_n}{(k+n-1)\,t_p}$$

If value of n not given

$$S = \frac{t_n}{t_p}$$

Q 20 $t_n$ = 12 cycles
k = 6
$t_p$ = 6

$S = t_n / t_p$
= 12/6 = 2.

19
P84

$D_1$

$k = 5$
3, 2, 4, 2, 3
$n = 100$
$t_p = \max(3, 2, 4, 2, 3)$
$= 4$ ns.
time $= (k+n-1)\, t_p$
$= (5+100-1)\, 4 = 416$ ns.

$D_2$

$k = 8$
2, 2, 2, 2, 2, 2, 2, 2
$n = 100$
$t_p = 2$ ns.
time $= (8+100-1)\, 2$ ns
$= 214$ ns.

$\therefore$ time Saved $= 416 - 214 = 206$ ns.

Q7 In steady stage.

$S_{max} = \frac{t_n}{t_p}$

$t_p = \max(5, 6, 11, 8) + 1$
$= 12$ ns
$t_n = 5 + 6 + 11 + 8$
$= 30$ ns.

$\therefore S_{max} = \frac{12}{30} = 2.5$

21

IF — 5
ID — 4
OF — 20
EX — 10
WB — 3

req = 2ns.
$n = 20$

~~Ep = 200~~.

NP.
$k = 5$
$n = 20$
$t_p = 20 + 2$
$= 22$ ns.

EP
IF = 5
ID - 4
OF1 = 12ns
OF2 = 8 ns
Ex = 10
WB = 3

req = 2ns.
$k = 6$
$n = 20$
$t_p = 12 + 2$
$= 14$ ns

$\therefore$ time in NP $= (5+20-1)\, 22$ ns.
$= 528$ ns
time in ep $= (6+20-1)\, 14$ ns
$= 350$

Speed up $= \frac{528}{350} = 1.51$

Q17 $P_1$ $k = 4$.

$t_p = \max(1, 2, 2, 1)$

$= 2$ ns.

$P_2$:

$t_p = \max(1, 1.5, 1.5, 1.5)$

$= 1.5$

$P_3$ : $t_p = 1$ ns.

$P_4$ : $t_p = 1.1$ ns.

$\therefore$ $P_3$ has lowest clock time $\Rightarrow$ highest frequency

Q13 non pipelined.

Clock rate = 2.5 GHz.

cycle time = 0.4 ns.

no. of cycles per inst? = 4.

$t_n = 4 \times 0.4 = 1.6$ ns.

Pipeline.

Clockrate = 2 GHz

Cycletime = 0.5 ns.

$t_p = 0.5$ ns.

$\therefore$ Speedup = $S_{max} = \frac{t_n}{t_p} = \frac{1.6 \text{ ns}}{0.5 \text{ ns}} = 3.2$.

In $t_p$ whole one cycle is performed. hence nothing is required to store 'n' or 'k'

→ **Instruction pipeline:**

If the pipeline processing is implemented on instruction cycle then that pipeline is known as instruction pipeline.

Assume;

5-stage inst? pipeline.

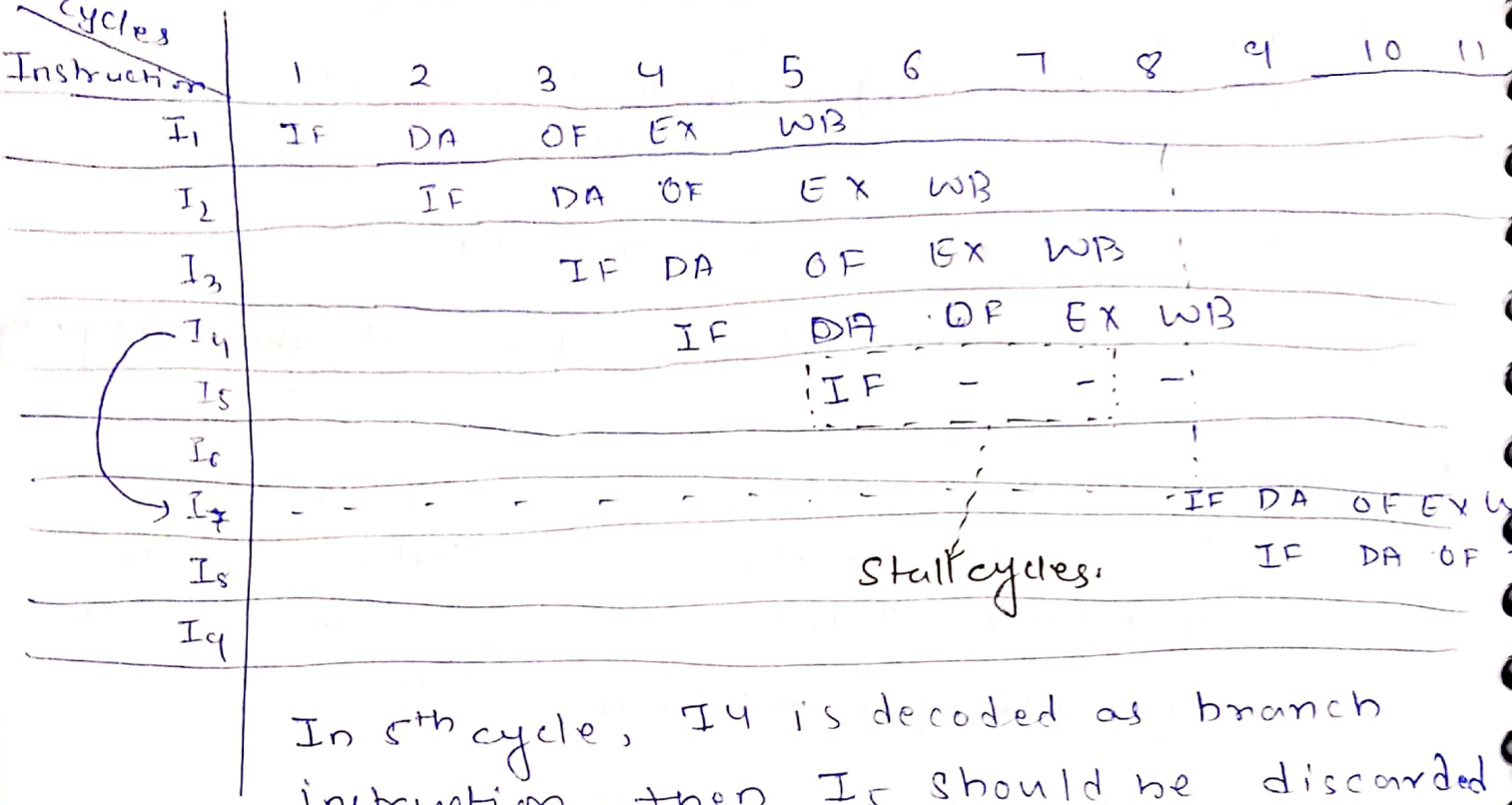
IF - Instruction fetch

DA - Decode & Add. calculation

OF - Operand fetch

EX - Execution

WB - Write Back.




In 5th cycle, $I_4$ is decoded as branch instruction, then $I_5$ should be discarded from pipeline & CPU will not take any new inst? till Ex phase of $I_4$.

In Ex phase of branch instruction ($I_4$) the condition is evaluated. Based on that in next cycle next instruction is fetched.

Total no. of instruction actualy executed = 6.

For n = 6 instructions:

no. of cycles required without any problem = 5 + 6 - 1 = 10

extra cycles because of branch problem = 3 extra cycles because of ($I_5$)

∴ total cyles will be = 10 + 3 = 13 cycles

This extra cyles is known as 'stall cycles' which is supposed to be discarded

the branch result is available after
th stage then no. of stall cycles because
ranch instruction is equal to "I - 1"

= 5
I = 7
= 10
I = 8
D = 6.

g ⇒ 1 ns.

$t_p = 10 + 1$
$= 11$ ns

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | FI | DI | FO | EI | WO | | | | | | | | | |
| | | FI | DI | FO | EI | WB | | | | | | | | |
| | | | FI | DI | FO | EI | WD | | | | | | | |
| | | | | FI | DI | FO | IE | WD | | | | | | |

→ Pipeline hazards :-

The reason which causes disturbance in smooth conduct of pipeline

i) Resource conflict: (collision or structural dependency)

If the same resource is required by two instructions simultaneously

→ IF ID (OF) EX WB

IF ID OF EX WB

(IF) ID OF EX WB

Here, the 'IF' operation of Instruction $I_3$ wants to access memory at same time $I_1$ is using memory to access 'OF'

(ii) Data Dependency:

If the result of an Instruction is used as an input in next instruction

i : $R_1 \leftarrow R_2 + R_3$ ~~IF ID OF EX WB~~

i+1 : $R_5 \leftarrow R_1 * R_4$.

i → IF ID OF EX WB.

i+1 → IF ID - - OF EX WB

a general pipeline hardware cannot detect data dependency. In this case compiler generates the Instructions which solves the data dependency problem.

This solution is known as delayed load

In above example IF hardware cannot detect problem the compiler will send set of instruction in such a way that i & i+1 donot come one after another

i : $R_1 \leftarrow R_2 + R_3$ IF ID OF EX WB

i+1 : No operation Instn IF ID OF EX WB

i+2 : No operation Instn IF ID OF EX WB

i+3 : $R_5 \leftarrow R_1 * R_4$ IF ID OF EX WB

Delayed Load is also known as software sol? As compiler provides it.

If pipeline hardware is intelligent enough to detect the data dependency. They might be among

→ Hardware interlock.

→ Operand Forwarding (bypassing)

→ Hardware interlock.

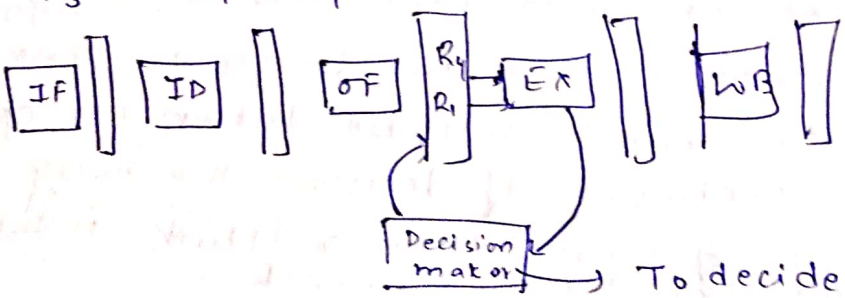
Hardware will provide stall cycles to delay operand Fetch as in last example.

→ Operand Forwarding (bypassing)

Avoiding the delay ie extra delay. If one instruction is dependent on another then on execution time of $I_1$ the result is forwarded to 'WB' phase and also to the Execution phase of next immediate instruction as these two executions are not happening simultaneously.

$R_1 \leftarrow R_2 + R_3$ IF ID OF EX → WB

$R_5 \leftarrow R_1 * R_4$ IF ID OF → EX WB




→ To decide if dependency then only Forward operand.

If operand Forwarding Sofn is used then no stall cycles because of data dependency

→ Branch difficulty. (control dependency)
Because of Branch inst?. if Hazard occurs

**Normal h/w**

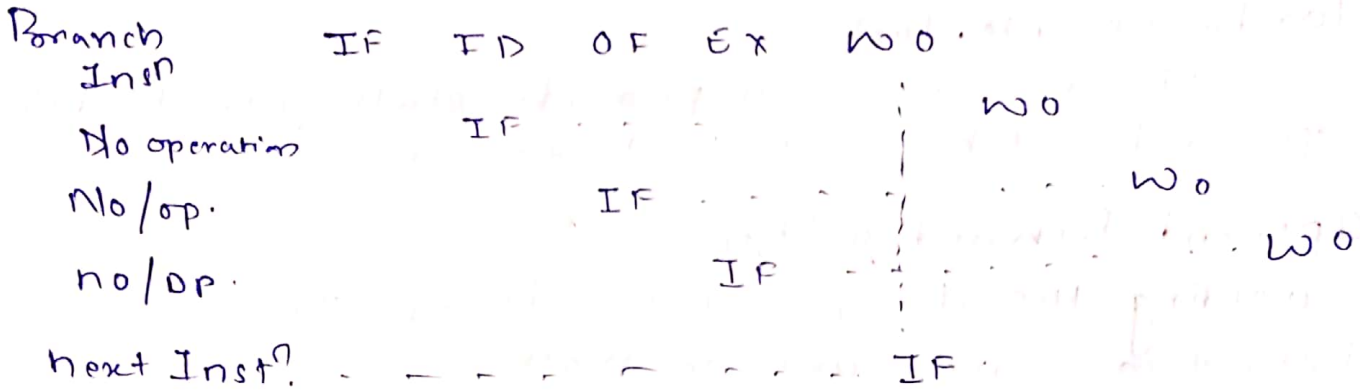
Cannot detect branch problem

Software sol? ⇒ delayed branch (by compiler)

**Intelligent h/w**

can detect branch problem.

H/w sol?




H/w sol?s ⇒ Branch prediction.

If any Branch instruction is encountered the h/w will take prediction whether to take a branch or to execute the next immediate instruction. If suppose 80% time the prediction is true then the branch will be taken an operation is continued if decision is wrong ie. 20% of time then rollback is taken, and executed the branch

⇒ Prefetch target instruction:

If any branch instruction is encountered then inside pipeline run 2 pipeline at a time 1 says branch to be taken other where branch is not taken after some cycles the result arrives which says which pipeline to adopt

and which to discard.

## Branch target buffer:-

| Branch Inst? | Target Inst? |
|---|---|
| $I_4$ | $I_9$ |
| $I_{12}$ | $I_{25}$ |

This buffer is used to store the condition of instruction if the inst? condition arrives again then we can refer to this buffer.

## Loop buffer:-

Used for small loops.

Store the loop instruction inside the Buffer which is added to pipeline and for every loop condition take those instructions from buffer

## ⟹ Data hazard classification

Consider two instructions i & j and i' executes before 'j'

(i) RAW (Read after write)

'j' is trying to read source before 'i' writes it. hence 'j' incorrecty gets old value.

i:- $R_1 \leftarrow R_2 * R_3$

j: $R_6 \leftarrow R_1 + R_5$

Data dependency

Sol? operand Forwarding.

(2) WAW (Write after write) (False dependency)

j tries to write a destination before i writes it

i : $R_1 \leftarrow R_2 * R_3$

ii : $R_1 \leftarrow R_6 + R_5$

---

Write Dependency.

Sol? Reg. Renaming

i : $R_1 \leftarrow R_4 * R_3$

ii : $R_8 \leftarrow R_6 + R_5$

(3) WAR (write after read.) (False dependency)

j tries to write a destination before i reads it hence i incorrectly gets new value.

i : $R_1 \leftarrow R_2 * R_3$

j : $R_2 \leftarrow R_6 + R_5$

---

Anti-dependency.

Sol? Reg. Renaming

(i) $R_1 \leftarrow R_2 * R_3$

(ii) $R_4 \leftarrow R_6 * R_5$

Q2. 1) Data dependency Q3 Register Renaming

2) Branch conflict

3) Resource conflict

$S_1$ ⟶ False.

$S_2$ ⟶ Yes

$S_3$ ⟶ No stalls created in Anti dependency

IF — 1 cycle  Given Operand forwarding

ID — 1 cycle.

OF — 1 cycle.

PO ——————————

WO — 1 cycle.

| ADD or SUB | MUL | DIV |
|---|---|---|
| 1 | 3 | 6 |

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IF | ID | OF | PO | Po | Po | Po | WD. | | | | | | | |

Q14 no. of inst? = n = 4

k = 4.

Without hazard no. of cycles = k + n - 1
= 4 + 4 - 1
= 7

extra = 6.

Total 13.

Q22. no. of instruction = n = 100

k = 4.

IF — 1.
ID — 1.
OF — 1
PO — 40 35 25
3 2 1

Extra (40 × 2) + (35 × 1) + (25 × 0) = 80 + 35 = 115

WB - 1.

∴ without hazard = k + n - 1 = 100 + 4 - 1.
= 103.

with extra stall cycles = 115.

∴ total = 103 + 115 = 219.

---

If k segment pipeline.
n inst^n. to execute

no. of cycles to execute n inst^n. = k + n - 1

CPI = $\frac{k+n-1}{n}$.

(In normal case)

If in case of hazard, total stall cycles = x.

$$CPI = \frac{(k+n-1)+x}{n}$$

In ideal conditions:

no. of cycles to execute n instⁿ = n.

$CPI = 1 = n/n$.

(In normal condition)

In case of hazard

$$CPI = \frac{n+x}{n}$$

Average instruction execution time = CPI * tp

**q.16** k = 6. all stage takes 'x' time, no register delay

tp = x. ; tn = 6x.

In ideal condition CPI = 1

$CPI_{avg} = 0.75 * 1 + 0.25(1+2)$

$= 1.5$

avg instⁿ execution time = 1.5x.

$$\text{Speedup} = \frac{\text{Time taken in Non pipeline}}{\text{Time taken in pipelined}}$$

$$= \frac{6x}{1.5x} = \boxed{4}$$

**q.4** 1 GHz = $\frac{1}{10^{-10}}$ = 1 nsec. 1 clock time

k = 5.

no. of stalls cycle because of branch = 3 − 1 = 2.

$CPI_{avg} = 0.8 * 1 + 0.2 * (1+2)$

$= 1.4$

1 instⁿ execution time = 1.4 * 1 ns

$10^9$ instructions execution time.

$= 10^9 \times 10^{-9} \times 1.4$

$= 1.4$ sec.

**Q9** $k = 5$

$t_p = 2$ ns.

$CPI = 1$

next inst? is Fetched only after completion of branch inst? => After 5th stage of branch inst?

no. of stall cycles $= 5 - 1 = 4$.

$CPI_{avg} = 0.8 * 1 + 0.2 (1 + 4)$

$= 1.8$.

1 inst execution time $= 1.8 \times t_p$.

$= 3.6$ ns.

**Q10** $CPI_{avg} = 0.8 * 1 + 0.2 * [0.2 * [1+4] + 0.8 [50 * [1+4] + 0.5 * 1]]$

0.8 non branch, 0.2 branch, 0.2 unconditional, 0.8 conditional, 50 branch taken, 0.5 branch not taken

$0.8 + 0.2 [0.2(5] + 0.8 (0.5 * 5 + 0.5)]$

$0.8 + 0.2 [1 + 0.8 (2.5 + 0.5)]$

$0.8 + 0.2 (1 + 0.8 (3])$

$0.8 + 0.2 (3.4) = 1.48$ cycles.

$\therefore$ 1 inst? execution time $= 1.48 * 2$ ns.

$= 2.96$ ns.

12 $k = 4$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $I_1$ | $S_1$ | $S_2$ | $S_2$ | $S_3$ | $S_4$ | $S_4$ | | | | | | | | |
| $I_2$ | | $S_1$ | $S_1$ | $S_2$ | $S_3$ | $S_3$ | $S_4$ | | | | | | | |
| $I_3$ | | | - | $S_1$ | $S_2$ | - | $S_3$ | $S_3$ | $S_4$ | | | | | |
| $I_4$ | | | | - | $S_1$ | $S_1$ | $S_2$ | - | $S_3$ | $S_3$ | $S_4$ | | | |
| $I_1$ | | | | | - | - | $S_1$ | $S_2$ | $S_2$ | - | $S_3$ | $S_4$ | $S_4$ | |

∴ $I_1$ on second time $S_4$ will be after 13 ns.

or

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $I_1$ | 1 | 3 | 4 | 6 | max of (3,3)+1 = 7 | | | | |
| $I_2$ | | 3 | 3+1=4 | max of 4+2=6 | max of (6,6)+1 = 7 | | | | |
| $I_3$ | | | 4 | 4+1=5 | max(3,6)+2 = 8 | max(7,7)+1 = 9 | | | |
| $I_4$ | | | | 4+2=6 | max(5,7)+1 = 7 | max(8,7)+2 = 10 | max(10,11)+1 = 11 | | |
| $I_1$ | | | | | 6+1 = 7 | max(7,7)+2 = 9 | max(10,9)+1 = 10+1 = 11 | max(11,11)+2 = 11+2 = 13. | |

Q11 if 100% efficiency ⇒ $S = S_{max}$.

88% efficiency ⇒ Speed up = 88% $S_{max}$

100 MHz.

cycle time = $\frac{1}{100\,MHz}$ = 10 nsec.

$t_p$ = 10 ns.

k – segments

$t_n = k * 10$ ns.

$S_{max} = k$

∴ $S$ = 8.8% of $S_{max}$

6.6 = 88% of $k$

$k = \frac{6.6}{0.88} = 7.5 \approx \boxed{8}$

In ideal conditions 1 results in 1 cycle,
hence in tp time
no. of results per unit time = $1/t_p$
= throughput of pipeline

throughput = no. of operations performed in unit time.

Q5 k = 4
800, 500, 400, 300
tp = 800 ps
throughput = $\frac{1}{800}$

k = 5
600, 350, 500, 400, 300
tp = 600 ps
= $\frac{1}{600}$

% of throughput increase = $\frac{\frac{1}{600} - \frac{1}{800}}{1/800} * 100\%$

$= (4/3 - 1) * 100\% = 33.3\%$

---

⟹ Non-Linear Pipeline :-

Linear

| | | | | |
|---|---|---|---|---|
| $S_1$ | x | | | |
| $S_2$ | | x | | |
| $S_3$ | | | x | |
| $S_4$ | | | | x |

non-Linear

| | | | | | |
|---|---|---|---|---|---|
| $S_1$ | x | | x | | |
| $S_2$ | | x | | | x |
| $S_3$ | | | | x | |
| $S_4$ | | | | | |

Initiation: Providing a new input in pipeline is known as initiation

Collision: resource conflict.

Latency: difference of no. of cycles b/w 2 initiation

Forbidden latency: latency which causes collision

Permissible latency: latency which does not cause collision.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| $S_1$ | $S_2$ | $S_3$ | $S_3$ | $S_2$ | | | |
| | $S_1$ | $S_2$ | $S_1$ | $S_3$ | $S_2$ | | |

} — latency = 1 (Permissible)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $S_1$ | $S_2$ | $S_1$ | $S_3$ | $S_2$ | | | |
| | | | $S_1$ | collision | | | |

] — latency = 3 (Forbidden)

Problem in non-linear pipeline:-

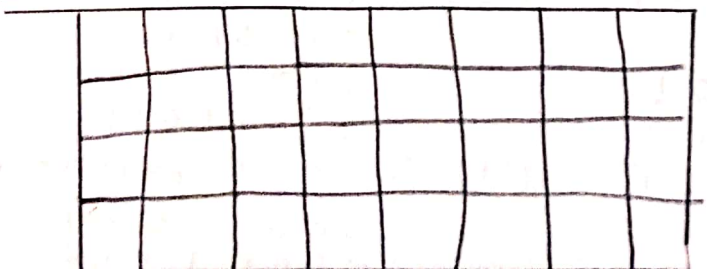
To find a minimum latency cycle which does not cause any collision.

$S_1$ $S_2$ $S_1$ $S_3$ $S_2$ :

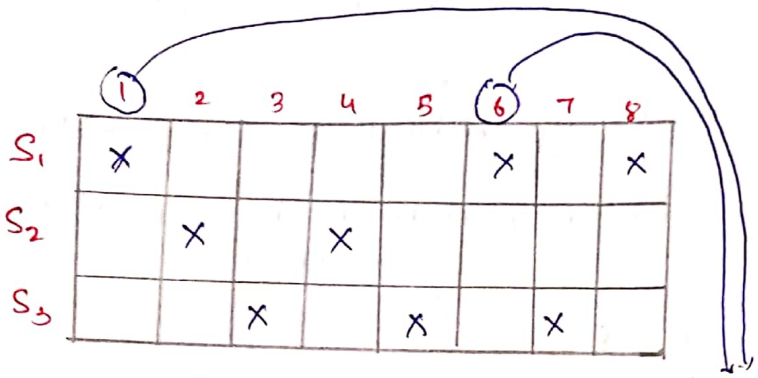
$S_1$ $S_2$ $S_1$ $S_3$ $S_2$

If we see above situation next initiation is occuring after whole instruction is executed. this situation is Sequencial set not pipeline hence to improve it we need minimum latency cycle which provide better outcome in lesser time. without any collision.

Eg: Reservation table.



eg:




Forbidden latency = (6-1), (8-1), (8-6), (4-2)
(5-3), (7-3), (7-5)

= 5, 7, 2, 2, 2, 4, 2

= 7, 5, 4, 2.

Permisible latency = 8, 6, 3, 1

* Collision vector → 0 — Permisible
1 — Forbidden

$C_8\ C_7\ C_6\ C_5\ C_4\ C_3\ C_2\ C_1$
0 1 0 1 1 0 1 0 → collission vector.

## → State transition Diagram

Take original collision vector as a state. For each 0 at position i in collision vector, right shift it 'i' position and take OR with original collision vector; which can give you new state.

* For state 01011010

(i) For 0 at position 1

00101101
(or) 01011010
01111111

(ii) For 0 at position 3:

00001011
01011010
01011011

(iii) For 0 at position 6:

00000001
01011010
01011011

(iv) For 0 at position 8 :-

```
  00000000
  01011010
  --------
  01011010
```

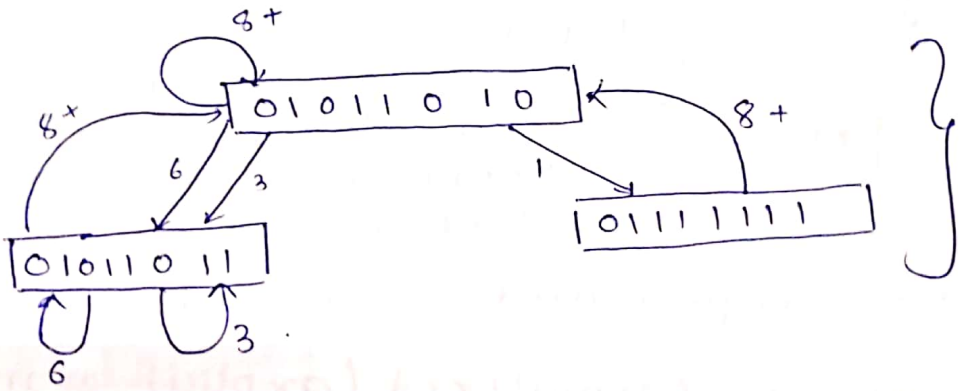
For state 01011011

(i) For 0 at position 3

```
  000 01011
  01011 010
  ---------
  01011 011
  ---------
```

(ii) For 0 at position 6

```
  000000 01
  01 0110 10
  ----------
  01 0110 11
  ----------
```




State diagram

→ Simple cycle :- A latency cycle in which every ~~Stage~~ State appears exactly once.

{8}, {1,8}, {3,8}, {6,8}, {3}, {6},

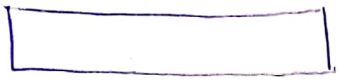
→ Greedy cycle :- A simple cycle in which only minimum latency from each state is considered

{1,8}, {3}

average latency ⇒ $\frac{1+8}{2} = 4.5, 3$

MAL (minimum average latency) = 3.

# * Floating point representation *



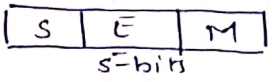
**Motivation:-** It can represent a larger range of numbers as compaired to the fixed point representation using same no. of bits.

In Floating point representation every no. is represented as follows.


S → sign ⇒ S { 0 → +ive, 1 → -ive.
E → Exponent.
M → Mentisa.

- Exponent is represented in biased form.
- Mentisa is a normalized (explicit or implicit) fraction number
- 

Bias form → Storing only unsigned value of exponent or storing exponent in unsigned value.

eg. 


5-bits ⇒ Range ⇒ (-16 to +15) ⇒ 0 to 31

biasing (excess -16)

| original exponent (e) | Stored Exponent (E) |
|---|---|
| -16 | 0 |
| -15 | 1 |
| -14 | 2. |
| ⋮ | ⋮ |
| 0 | 16 |
| ⋮ | ⋮ |
| 15 | 31 |

$E = e + (16)$ → bias.

if exponent is represented by k-bits
bias = $2^{k-1}$

Normalization → Explicit (convert no. into 0.1....)
→ Implicit (convert no. into 1.0....)

100.01 → Explicit normalization ⇒ $0.10001 * 2^3$ (M; compulsory '1' after '.')
→ Implicit normalization ⇒ $1.0001 * 2^2$ (M; compulsory '1' before '.')

In explicit → M = 10001, e = 3

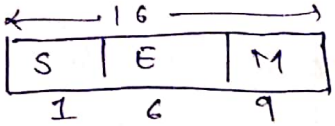
In implicit → M = 0001, e = 2.

| S | E | M |
|---|---|---|

Value (Explicit) = $(-1)^S * 0.M * 2^{E-bias}$.

Value (Implicit) = $(-1)^S * 1.M * 2^{E-bias}$.

→ Consider a 16-Bit register which is use to represent floating point numbers. Exponent is represented in Excess-32 form. Mentisa is a normalized number What is the 16-bit pattern for the value $-(13.25)_{10}$




bias = $32 = 2^{k-1}$

$k = 6$.

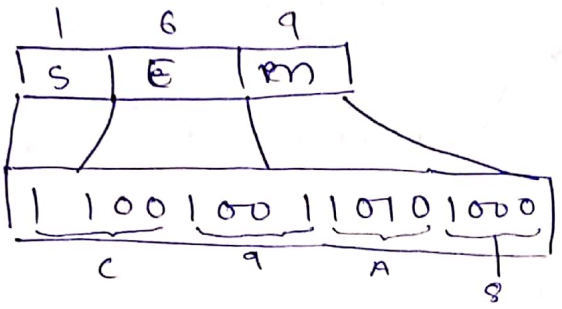
$-(13.25)_{10}$ ⇒ 13.25 = 1101.01 ⇒ Explicit normalization (Default)

$0.110101 \times 2^4$

M = 110101 ⇒ 11010100

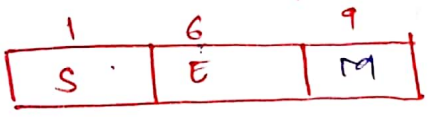
e = 4

E = 4 + 32 = 36 ⇒ 100100




C9A8

→ What is the maximum positive value which can be represented in this register.




⟶ S = 0

M = 111111111

E = 111111 = 63

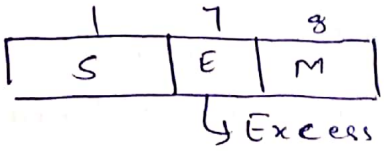
∴ value = $(-1)^0 * 0.111111111 * 2^{63-32}$

$= 111111111 * 2^{-9} * 2^{31}$

$= 111111111 * 2^{22}$.

$= (2^9 - 1) * 2^{22}$.

Q 4




Pure fraction — no normalization

0.239.

~~$0.239 \times 2^{-1} = 0.25 \times 2^{0}$~~

$0.239 \times 2 = 0.478 \longrightarrow 0$

$0.478 \times 2 = 0.956 \longrightarrow 0$

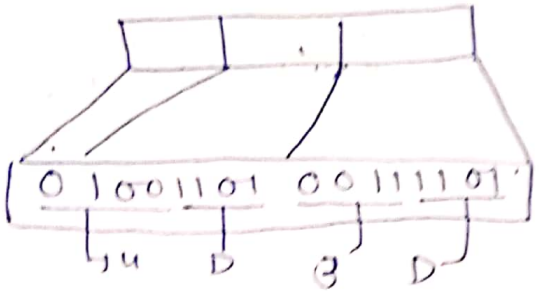
$0.958 \times 2 = 1.912 \longrightarrow 1$

$0.239 * 2^{13} \Rightarrow 0.00111101 * 2^{13}$

M = 00111101

e = 13

E = 13 + 64 = 77 = 1001101



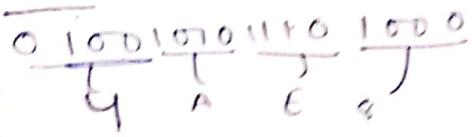

Q.5 $0.00111101 \Rightarrow$ Implicit = $1.1110100 * 2^{-3} * 2^{13}$

$1.11101000 * 2^{10}$

M = 11101000

e = 10

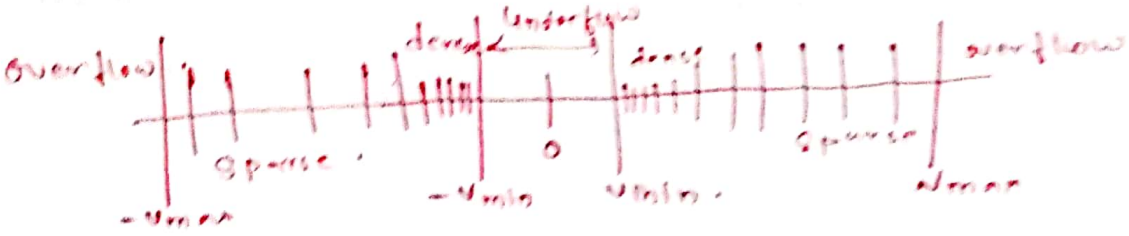
E = 10 + 64 = 74 = 1001010




Note: If more no. of bits in mantisa means more precision.

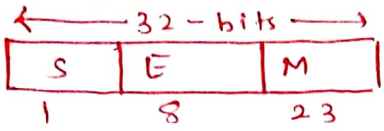
If more no. of bits in exponent means Larger range of numbers

This conversional floating point represent cannot store '0'



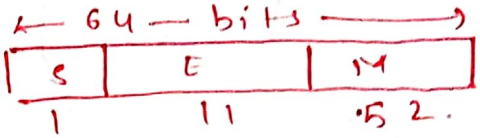
→ IEEE – 754 Floating point representation

Single precision




bias = 127 = $2^{8-1}-1 = 2^7-1$

Double precision




bias = $2^{10}-1 = 1023$

→ Some combinations of exponent are kept reserved to represent special numbers by keeping the bias value 127 or 1023.

→ Exponent combination all zeros or all ones represent special numbers

| S | E | M | Notation. |
|---|---|---|---|
| 0 | 000...0 | 0...0 | +0 |
| 1 | 000...0 | 0...0 | −0 |
| 0 | 11...1 | 0...0 | +∞ |
| 1 | 11...1 | 0...0 | −∞ |
| 0/1 | 11...1 | M ≠ 0 | Not any number (NAN) |
| 0/1 | 000...0 | M ≠ 0 | Fraction or denormalized no. |
| 0/1 | E ≠ 00...0 and E ≠ 11...1 | M = xx...x | Implicit normalized. |

* Fraction or Denormalized no:-

A very very small number which cannot be implicitly normalized.

min allowed exponent = -126.

Upto this if we are getting normalized numbe well and good if not the store the mentiga as it is

and keep Exponent bits = all '0's

eg:

→ 0.000 . . . 0101

normalized form

$1.01 * 2^{-130}$ → not allowed

e = -130

E = -130 + 127 = -3 cant be possible.

$0.000101 * 2^{-126}$

M = 000101

E = 0000 . . .

This could be Representation of Fraction no

→ 0 0 0 0 . . . 0101

Implicit normalized

$1.01 * 2^{-127}$

e = -127

E = -127 + 127 = 0 ⇒ not allowed for implicit normal form.

Value (implicit) = $(-1)^S * 1.M * 2^{E-bias}$

Value (denormalized) = $(-1)^S * 0.M * 2^{-126}$ or -1022.

Q7 $-14.25 = 1110.01 \Rightarrow$ Implicit normalisation

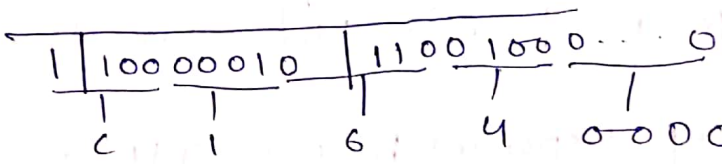
$\downarrow\downarrow$

$1.11001 \times 2^3$

$M = 11001000$

$e = 3$

$E = 3 + 127 = 130 = 10000010$



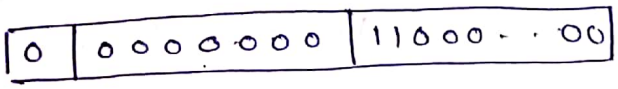

→ Consider the following 32-bit register that denotes single precision floating point number in IEEE 754 format. What is its decimal value representation.


→ Exp → 131

$Value = (-1)^0 * 1.11 * 2^{131-127}$

$= 1.11 * 2^4$

$= 11100 = +28$




Exp → all '0' · Man ≠ 0

∴ Denormalized no.

$Value = (-1)^0 * 0.11 * 2^{-126}$

$11.0 * 2^{-2} * 2^{-126}$

$3 * 2^{-128}$