weightage : 9 - 12 marks

chapter - 1 (introduction to COA)

(i) Introduction
(ii) Components of computer
(iii) types of registers
(iv) instruction cycle
(v) memory concept
(vi) byte & word addressable
(vii) system bus
(viii) byte ordering

chapter - 2 (instruction format and addressing modes)

(i) Instruction concept
(ii) machine instruction
(iii) instruction format
(iv) expand opcode technique
(v) addressing modes concept

(vi) types of addressing modes
(vii) instruction set architecture

chapter - 3 (ALU, data path and control unit)

(i) Data path
(ii) micro instruction
(iii) micro program
(iv) control unit design

chapter - 4 (pipelining)

(i) Pipeline concept
(ii) Pipeline types
(iii) Perfomance evaluation
(iv) Dependencies in pipeline
- structural dependency
- data dependency
- control dependency

(v) pipeline hazards

chapter - 5 (cache memory)

(i) Memory concept
(ii) types of memory organization
(iii) cache memory
- cache organization
- mapping technique
- replacement algorithm
- updating technique and multi level cache

chapter - 6 (secondary memory and I/O interface)

(i) disk concept
(ii) disk structure
(iii) disk access time
(iv) I/O interface and its types

Introduction of COA

Topic : Computer generation

	1st	IInd	IIIrd	IVth and Vth
	1942-1955	1955-1964	1965-1974	1974-present
component	Vaccumetube	Transistor	Integrated chip	VLSI/ULSI very large scale integration
Language	M/C language	Assemly	H.L.L and OOP's	OOPS and RDBMS, AI

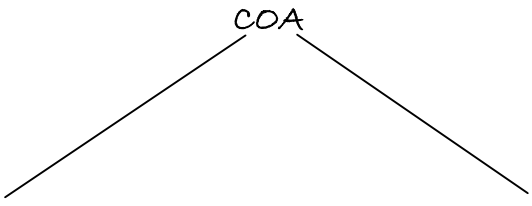
Note :

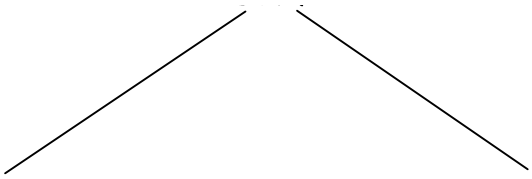
First digital computer was ENIAC in 1943
(Electronic numerical integrator and computer)

Topic : Computer organisation and architecture

why study COA?

to know how the computer works and design(built).





Computer architecture
attribute that is visible programmer
- represents internal design
- Instruction format
- Adressing mode
- ALU
- number of bit required to represent the data

Computer organisation
deals with how features will implement
- how various memory and I/O interact and implement or communicate with the system

Note :

intel x86 share the same architecture but ogranisation is different (how features are implemented that is different)

intel x86
80186
80286
80386
80486
80586

family

topic - 2 (components of computer)

Monday, July 8, 2024 4:46 AM

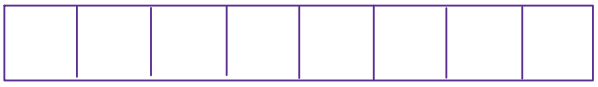
component of the computer

1. CPU
   - ALU (arithmetic logical unit)
   - Registers
   - CU (control unit)
2. memory
   - primary/main memory
   - secondary/auxillary memory
3. I/O
   - Input device
   - Output device

Register : stores the data (temporary storage) (register present inside the CPU) (fastest) (made with flip-flop; flip-flop is 1 bit storage device)

Size : SM>Main Memory>cache>register

Speed: register>cache>Main Memory>Secondary memory

8 bit register :  8 bit storage device / stores 8 bit data.

register types in CPU

1. Based on the task assigned to them
   - (A) General purpose register : not for any specific purpose
   - (B) Special purpose register : made for specific purpose

topic - 3 (types of registers)

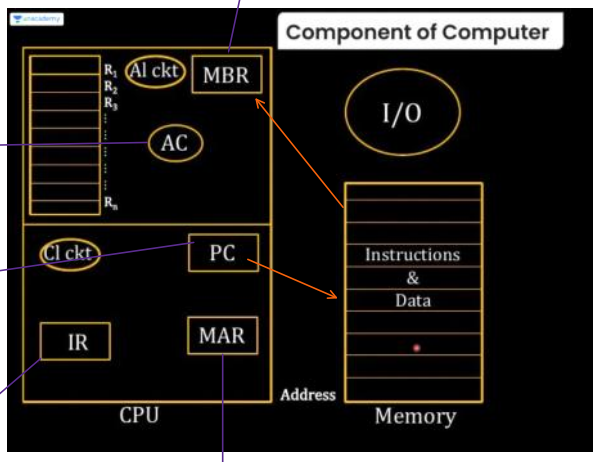
Monday, July 8, 2024 4:46 AM

types of registers

Special purpose register types :

memory buffer register : hold the instructions or data
why MBR/DR/MDR(data)?
connected to the data line of the system bus

Accumlator : contains the temporary result of the ALU operation (or) first operand of the ALU operation.
example : ADD[4000]
AC ← AC + M[4000]



general purpose register : used to process the data.
stack pointer (sp register) : contains the top of stack address. stack[LIFO:last in first out] memory insert and delete operation perfomed at same end (one end) called TOS and this TOS address is pointed/denoted by stack pointer register.
PSW (program status register)/flag regsiter : it stores the status of the ALU result.

program counter : contains starting address of the next instruction to be executed (fetch) (supplied by programmer) but dont know the address.

instruction register : contains the instruction which is currently executed by the CPU.
why only IR and not mbr?
because (Instruction format is pre-defined in IR)

memory address register : stores all the address of memory used for read/write operation.
why MAR/AR?
because it is connected to address line of the system bus. knows address and how to and where to go etc.

PC : next instruction address
IR : instruction of current execution

1. Memory address register (MAR)
2. I/O address register (I/O AR)
3. Program counter (PC)
4. Stack pointer register (SP)

→ stores the address

5. Memory buffer register (MBR/MDR/DR)
6. I/O data/buffer register (I/O BR)
7. Instruction register (IR)
8. Accumlator (AC)
9. Flag register / program status word (PSW)

→ stores the instruction (or) data

Program counter : When instruction is fetched (fetch cycle executed) then PC denotes the starting address of next instruction.

Program counter : When instruction is fetched (fetch cycle executed) then PC denotes the starting address of next instruction.

PC increment by 1 (or) PC increment by '4' (1word=4Byte) or '8'(1word=8Byte) or 'X' value.
gets Instruction address register / Instruction point register.

2. Based on the information/content they have

(A) Data register : stores the data

(B) Address register : store the addresses

memory access through MAR and MBR to

1. Memory read
   - Address : MAR/AR
   - Data : MBR
2. Memory write
   - Address : MAR/AR
   - Data : MBR

topic - 4 (instruction cycle)

Monday, July 8, 2024 4:46 AM

Instruction cycle

the process required for each instruction exectuion

(OR)

the phases that we need to go through in order to exectute the instructions

(OR)

instruction cycle describes the execution sequence of the instruction

Instruction cycle contains 2 sub cycle :

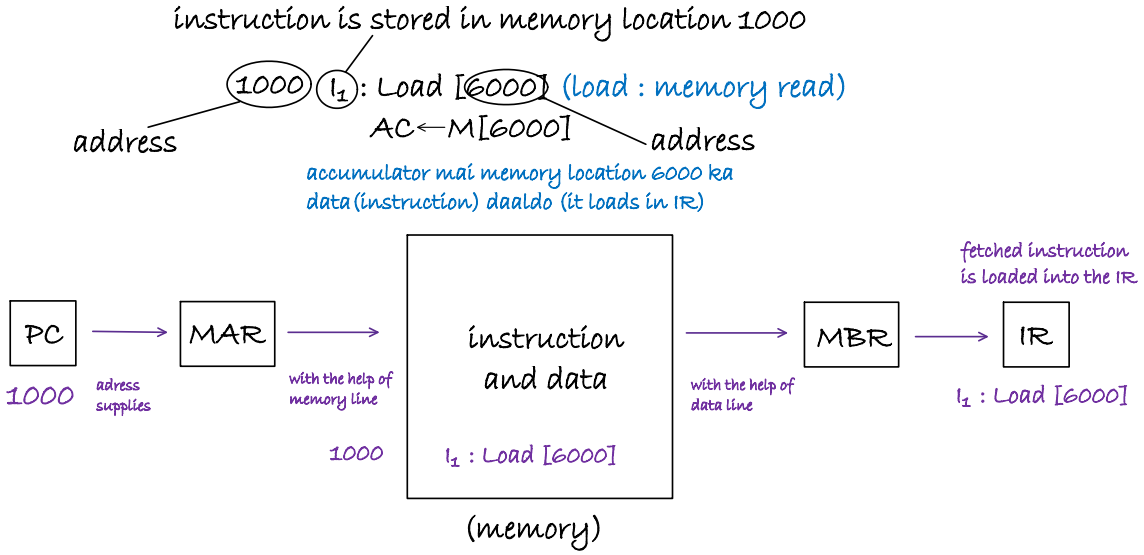
1. Fetch cycle
2. Exectute cycle ——— Decode
   Execute

1. Fetch cycle : to fetch (bring) the instruction from main memory to CPU. (doesn't care what is the instruction)
   (works as a postman)

   Memory ⟶ CPU(IR)

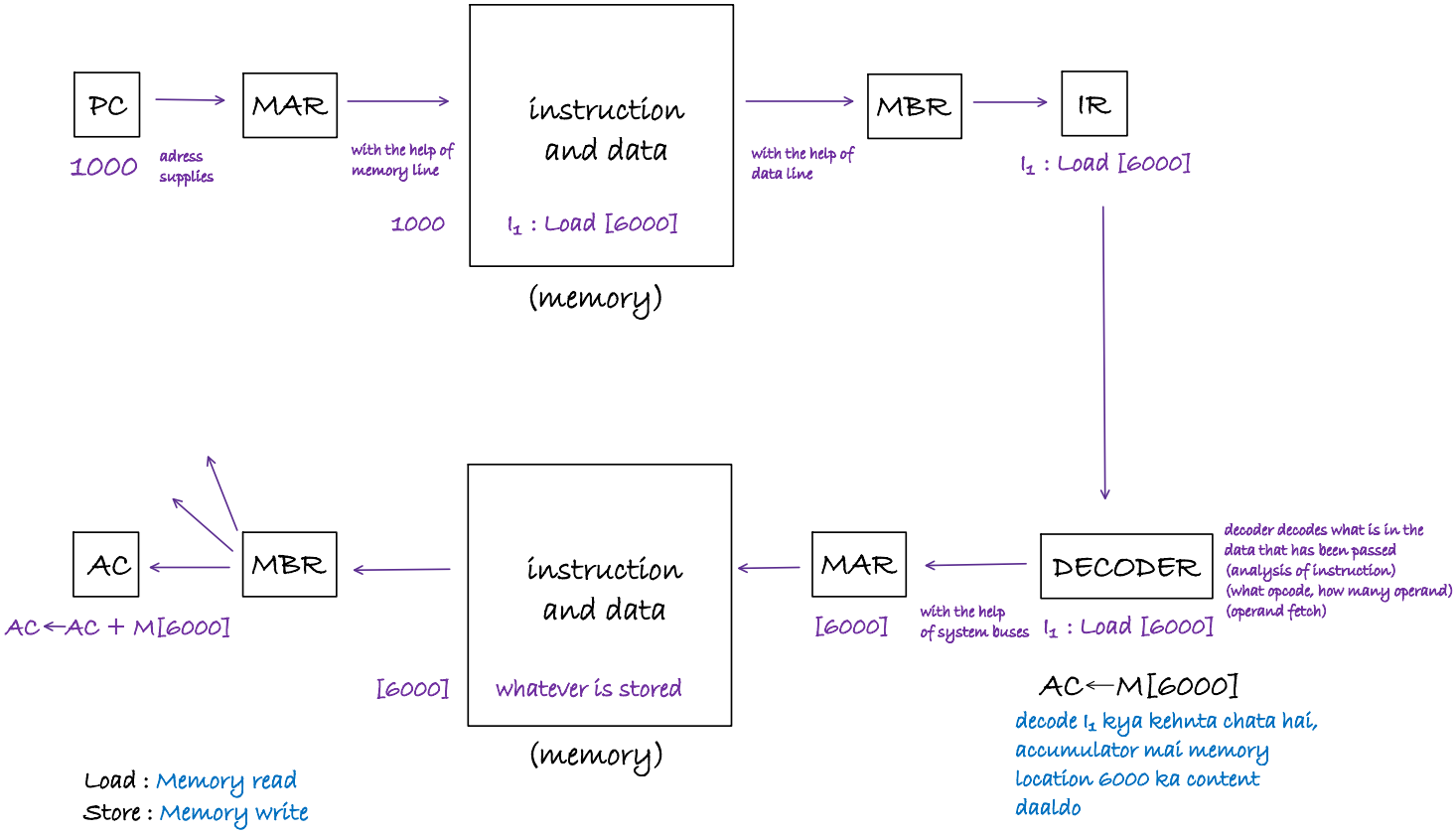
   and at the end of fetch cycle program counter is incremented.

instruction store hota hai memory mai aur execute hota hai CPU mein



2. Execute cycle : the objective of the execute cycle is to execute (to process) the fetch instruction. it decodes; does the analysis of the instruction. (what is OPCODE, how many

operand, operand address calculation, operand fetch, processing, result storage)



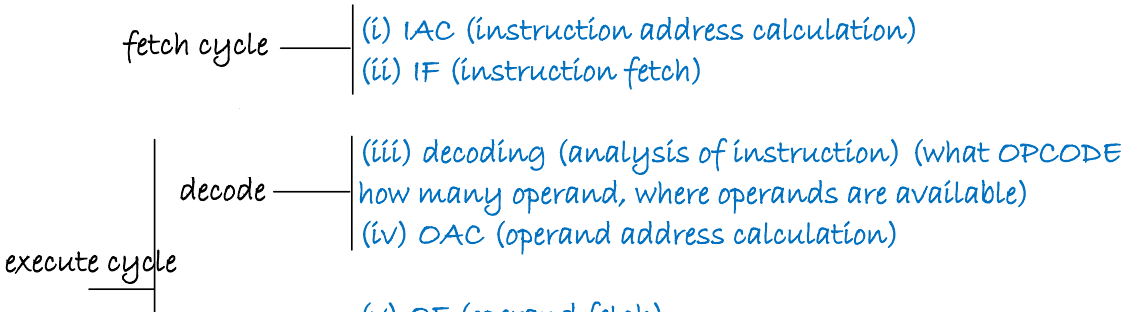
Step 1 : at the beginning of each instruction cycle the processor fetches an instruction from memory.

Step 2 : the program counter (PC) holds the address of next instruction to be fetched next.

Step 3 : The processor increments the PC after each instruction fetch so that it will fetch the next instruction in sequence.

Step 4 : The fetched instruction is loaded into the instruction register (IR)

Step 5 : The processor interpret the instruction and performs the action



execute cycle

(iv) OAC (operand address calculation)

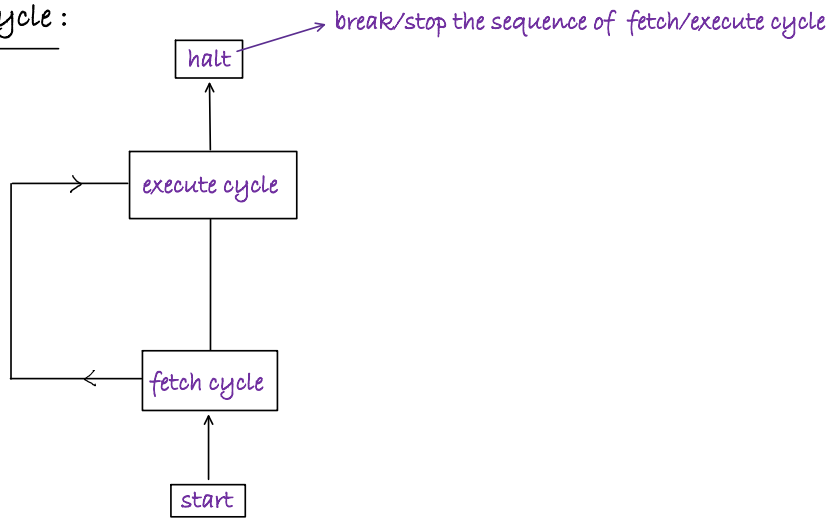
(v) OF (operand fetch)

execute

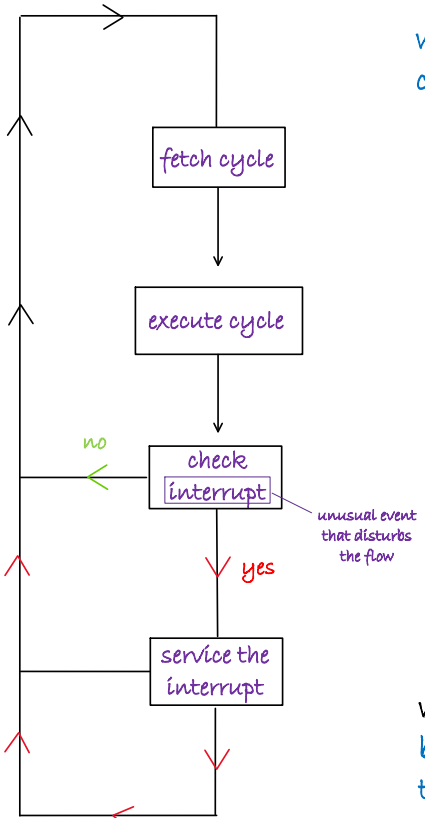
(vi) DP (data processing)

(vii) result storage

Instruction cycle :

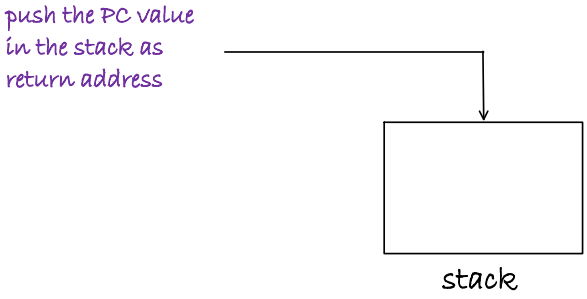


Instruction cycle with interrupt cycle :

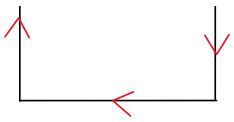


when CPU encounter the interrupt then after finishing (completion) of current instruction execution, interrupt will be serviced.

- when CPU encounter the interrupt then it push the PC (program counter) value into the stack as a return address and control transfer to ISR (interrupt subroutines)

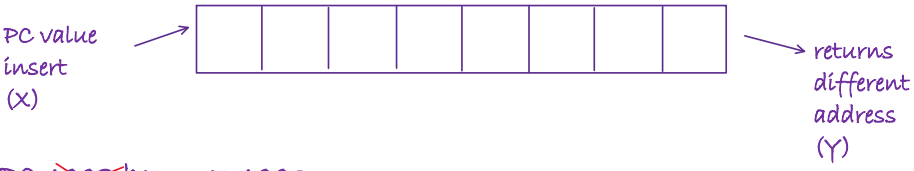


why PC value Push into stack?
because stack works in LIFO
the value you are leaving at last will be the first we get.



because stack works in LIFO
the value you are leaving at last will be the first we get.

What happens if return address stored in Queue?



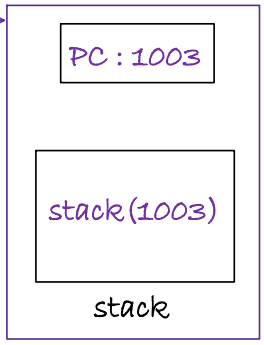
I_3 fetch PC:~~1002~~ becomes 1003
PC moves to next instruction

I_1 : 1000
I_2 : 1001
I_3 : 1002 (interrupt)
I_4 : 1003

after execution of I_3 we have to service interrupt (interrupt will have any address (call) where we have to service) so PC will save the address of interrupt and after servicing the interrupt we return to I_4

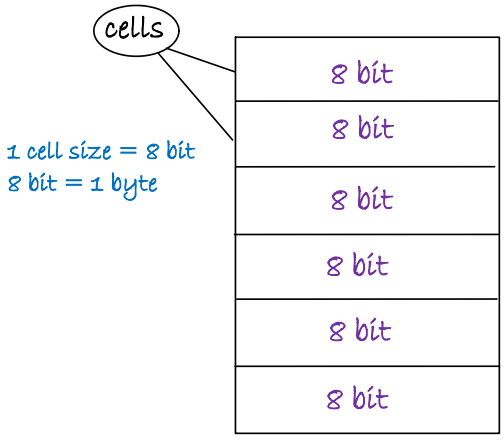
when we will go to service the interrupt we will be going before telling the stack in CPU about the next instruction so we will continue the next time from here

after servicing the interrupt i will POP the PC value from stack.

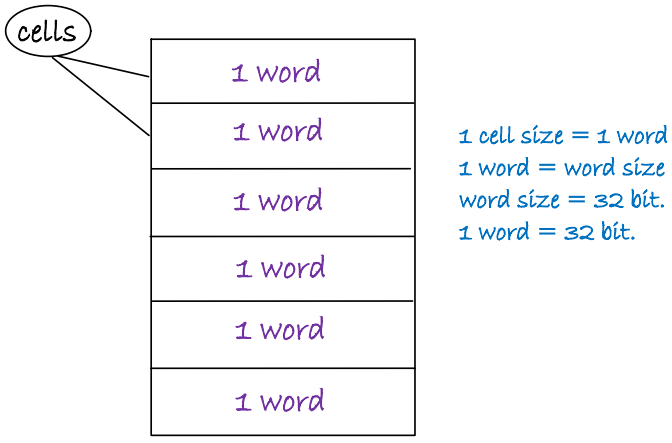


memory

byte addressable



word addressable



note:
word size is given in the question

word to byte and byte to word conversion :

1 word = 32 bit [4 byte]
2 word = 64 bit [8 byte]
4 word = 128 bit [16 byte]

4 byte = 1 word
8 byte = 2 word
16 byte = 4 word

Q. I_1 : 1 word
I_2 : 2 word

Program stored at starting address 2000, memory is
(i) word addressable
(ii) byte addressable
during the execution of I_2 what is the value of PC? (word size : 32bit)

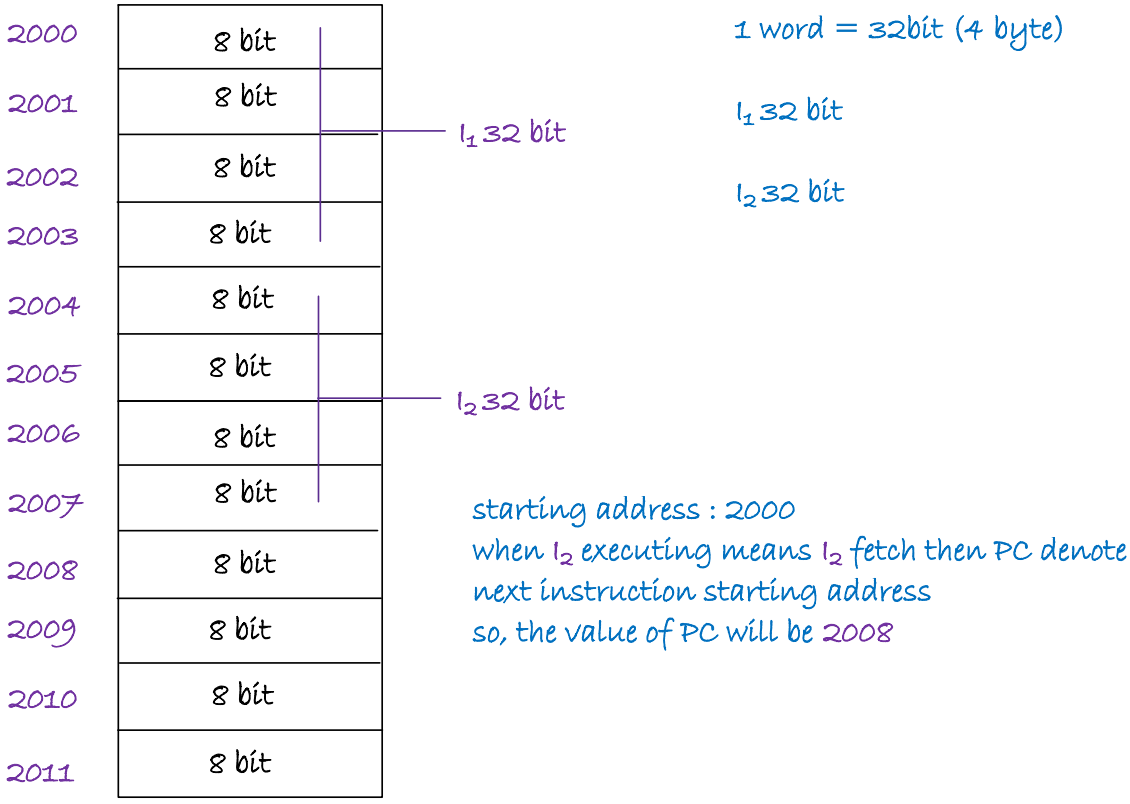
(i) word addressable :

I_1 32 bit	2000
I_2 32 bit	2001
next instruction	2002

starting address : 2000
when I_2 executing means I_2 fetch then PC denote next instruction starting address

so, the value of PC will be 2002

(ii) byte addressable :



1 word = 32bit (4 byte)

I_1 32 bit

I_2 32 bit

starting address : 2000
when I_2 executing means I_2 fetch then PC denote next instruction starting address
so, the value of PC will be 2008

Questions :

Q.1 Consider the following program segment execute on Hypothetical processor. [4 Marks]
Assume that program is stored in the memory address 1000(Decimal) onwards. During the execution of I_6 what could be value present in the Program counter. Assume that word size is 32 bit & memory is Byte Addressable?

Instruction	Size (in words)
I_1	2
I_2	1
I_3	1
I_4	3
I_5	1
I_6	2
I_7	1

Sol. Memory address: 1000. word size: 32 bits (4 byte)
Memory: Byte Addressable.
I_6 Execution, PC value - ?

Byte Addressable.

Instruction	Size (in words)
I_1	2
I_2	1

8 byte (1000 - 1007) (2 word: 64 bit (8 byte)
4 byte (1008 - 1011)

Instruction	Size (in words)
I_1	2
I_2	1
I_3	1
I_4	3
I_5	1
I_6	2
I_7	1

8 byte (1000 - 1007) (2 word : 64 bit (8 byte)
4 byte (1008 - 1011)
4 byte (1012 - 1015)
12 byte (1016 - 1019 - 1023 - 1027)
4 byte (1028 - 1031)
8 byte (1032 - 1035 - 1039)
4 byte (1040 1043)

1040

after I_6 executing means I_6 fetch then PC denote next instruction starting address

Q.2 Consider the following program segment execute on Hypothetical processor. [4 Marks]
Assume that word size is 32 bit & memory is word addressable. The program is stored in the memory at address 100(Decimal) onwards. During the execution of I_5. What could be value present in the program counter?

Instruction	Size (in words)
I_1	2
I_2	1
I_3	1
I_4	3
I_5	1
I_6	2
I_7	1

unacademy

Sol. word size = 32 bit. (4 byte)
Memory - word addressable.

Starting address - 1000
During I_5 - PC present value?

Instruction	Size (in words)
I_1	2
I_2	1
I_3	1
I_4	3
I_5	1
I_6	2
I_7	1

1000 - 1001
1002
1003
1004 - 1006
1007
1008 - 1009
1010

1008 Ans.

unacademy

Q. Consider the following program segment for a hypothetical CPU Having three users registers R1, R2 and R3. [GATE-2 Marks]

Instruction	Operation	Instruction size (in words)
MOV R1, 5000	R1 ← Memory[5000]	2 (1000 - 1001)
MOVR2, (R1)	R2 ← Memory[(R1)]	1 (1002)
ADD R2, R3	R2 ← R2 + R3	1 (1003)
MOV 6000, R2	Memory [6000] ← R2	2 (1004 - 1005)
HALT	Machine Halts	1 (1006)

interrupt

PC : 1004

stack (1004)

stack

interrupt encounter then PC values stores the next instruction starting address, push into the stack as a return address

Consider that the memory is word addressable with size 32 bits and the program has been loaded starting from memory location 1000 (decimal). If an interrupt occurs during the ADD instruction, what will be the return address pushed on to the stack

(a) 1007 (b) 1004 (c) 1005 (d) 1016

1000 (decimal). If an interrupt occurs during the ADD instruction, what will be the return address pushed on to the stack

(a) 1007 (b) 1004 (c) 1005 (d) 1016

Q. Consider the following program segment for a hypothetical CPU Having three users registers R1, R2 and R3. [GATE-2 Marks]

Instruction	Operation	Instruction size (in words)
MOV R1, 5000	R1 ← Memory[5000]	2 (1000 - 1007)
MOVR2, (R1)	R2 ← Memory[(R1)]	1 (1008 - 1011)
ADD R2, R3	R2 ← R2 + R3	1 (1012 - 1015)
MOV 6000, R2	Memory [6000] ← R2	2 (1016 - 1019 - 1023)
HALT	Machine Halts	1 (1024 - 1027)

Consider that the memory is with size 32 bits and the program has been loaded starting from memory location 1000 (decimal). If an interrupt occurs

During MOV 6000 R_2 Instruction then what PC value Pushed into the Stack ?

the return address (in decimal) saved in the stack will be

(a) 1007 (b) 1020 (c) 1024 (d) 1028

Q. Consider the following Program Segment for a hypothetical CN.

Instruction	Meaning	Instruction size (in words)	FeD + Execute
I_1 MOV r_0, 2000	r_0 ← M[2000]	3	3x3 + 4 = 13
I_2 MOV r_1, 3000	r_1 ← M[3000]	3	3x3 + 4 = 13
I_3 MUL r_0, r_1	$r_0 \leftarrow r_0 * r_1$	1	1x3 + 6 = 9
I_4 MOV 6000, r_0	M[6000] ← r_0	3	3x3 + 4 = 13
I_5 HALT	Machine Halt	1	1x3 + - = 3

Let the Clock Cycle required for various operation be as follows:

Instruction Fetch & Decode: 3 clock cycle per word

MUL with both operand & stored in register: 6 Clock Cycle.

Register to/from memory transfer: 4 clock cycle

The total number of clock cycle required to execute the program is___

51

depends on type of instruction

register to/from memory transfer : 4 clock cycle

MUL with both operand and stored in register : 6 clock cycle

instruction fetch and decode : 3 cycle per word

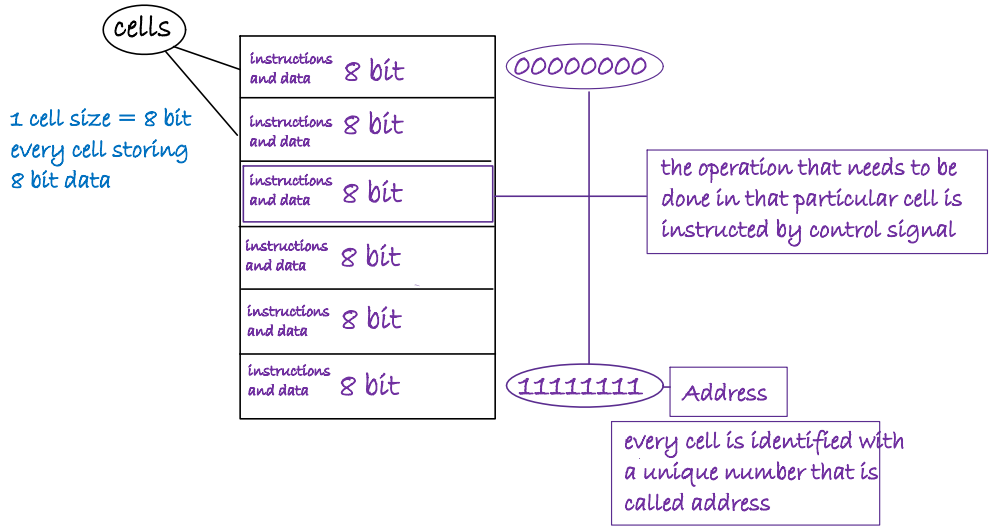
Mnemonic ≈ OPCODE

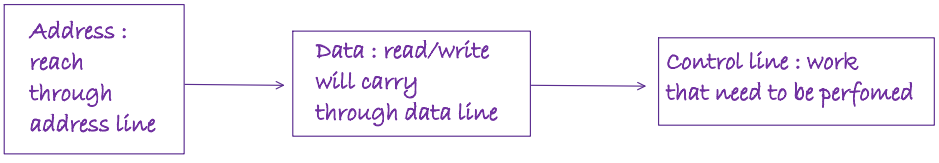
HALT : only fetch and not decode

byte and word addressable

- memory is a storage element in the computer which store instruction and data.
- memory is organised into equal parts, each parts is called cells.
- each cell is identified by a unique number called as address.

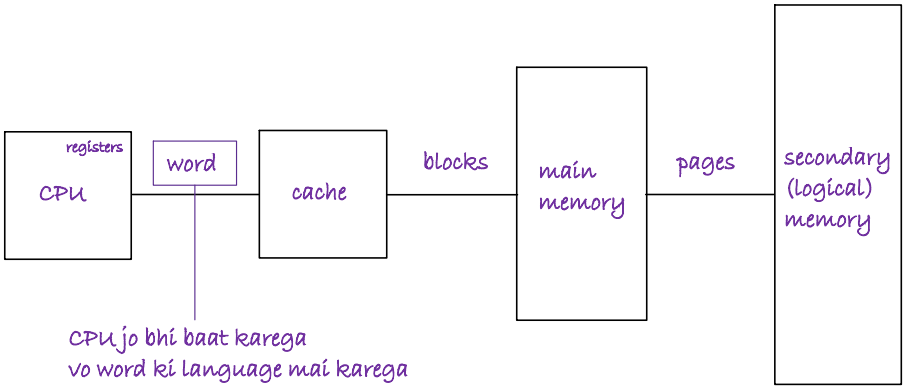
byte addressable





note :

word size = number of Data lines = data bus width
(data register size) (MBR/MDR/DR)
word (unit of data a/c to CPU) CPU talks in word unit.



note :

MAR/AR register size = number of address lines = address bus width

ex:

32bit processor

word size = 32bit

Data line = 32bit

Data bus = 32bit

MBR = 32bit

DR/MDR = 32bit

ALU = 32bit

AC = 32bit

Register = 32bit

ex.

24bit address

Address line = 24bit

Address Bus = 24bit

MAR = 24bit

AR = 24bit

PC = 24bit

$2^1 = 2$	$2^{30} = 1G$ (giga)
$2^2 = 4$	$2^{40} = 1T$ (tera)
$2^3 = 8$	$2^{50} = 1P$ (peta)
$2^4 = 16$	$2^{60} = 1E$ (exa)
$2^5 = 32$	$2^{70} = 1Z$ (zeeta)
$2^6 = 64$	$2^{80} = 1y$ (yotta)
$2^7 = 128$	
$2^8 = 256$	
$2^9 = 512$	1 byte = 8 bit
$2^{10} = 1024$	1 Nibble = 4 bit
$2^{10} = 1k$ (kilo)	
$2^{20} = 1m$ (mega)	

n bits =..... then N = ?

3 bit = $2^3 = 8$

13 bit = $2^{13} = 2^3 \times 2^{10} = 8K$ (8 x 1k)

22 bit = $2^{22} = 2^2 \times 2^{20} = 4M$ (4 x 1M)

35 bit = $2^{35} = 2^5 \times 2^{30} = 32G$ (32 x 1G)

N =..... then n = ? bits

(always search for closer and greater)

20 = 5bit

50 = 6bit

150 = 8bit ($2^8=256$ not $2^7=128$ because it is lesser than 150)

200 = 8bit

100 = 7bit

500 = 9bit

512 = 9bit

memory is represented in the form of 2^n x m

n : number of address line (a.l)

m : number of data line (d.l)

Address line : specify the capacity of the memory (0000 - 4bit) (0010 0011 - 8bit)

Data line : specify the capacity of data (cell size)

n-bit address line can represent 2^n memory cells (memory locations)

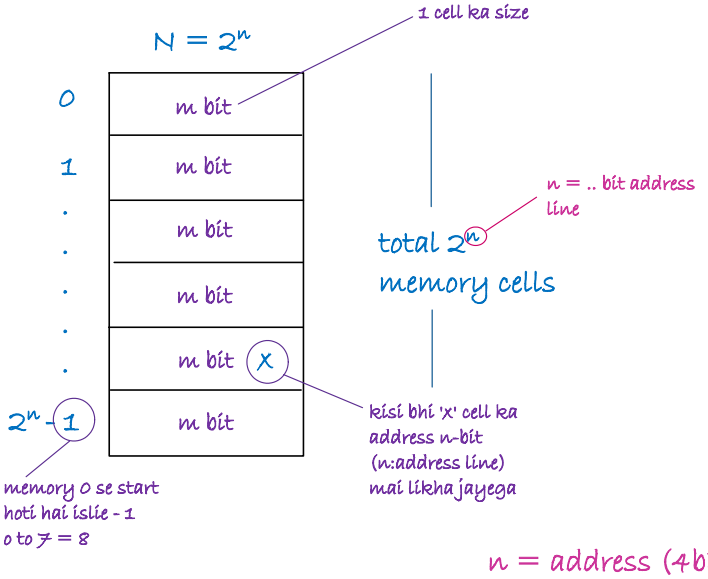
0 to $2^n - 1$ because memory starts from 0.

or in some places :

N x M

N : total number of memory cells (memory location)
M : each sell size

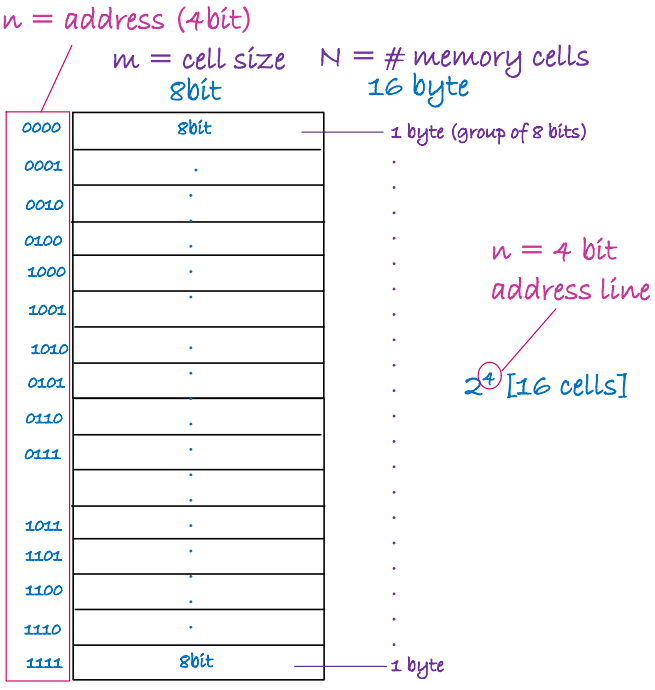
to represent any of the cells (among 2^n) n bit address are required.
(or)
n bit address can represent 2^n memory cells



for example : 16 byte meri memory hai
total 2^n mere cells hai
16 ke lie 4 bit lagega
aur 1 byte means 8 bit

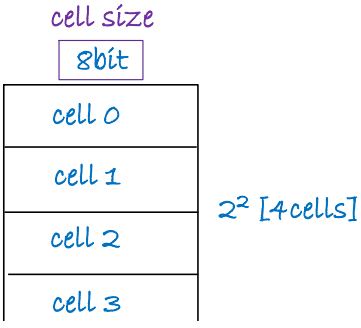
16 byte = 2^n x m
2^4 x 8bit
4 bit ki address line
8 bit ki data line

n = 4 bit (address line)
m = 8 bit (data line)
N = 2^4 =16 memory locations/cells

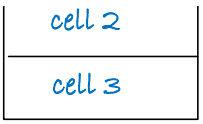


n = 2 (capacity of memory) address line.
m = 8 (cell size/cell capacity) data line.
N = 4 (total cells)

N = 2^n memory location/cells
N = 2^2 = 4 memory location/cells

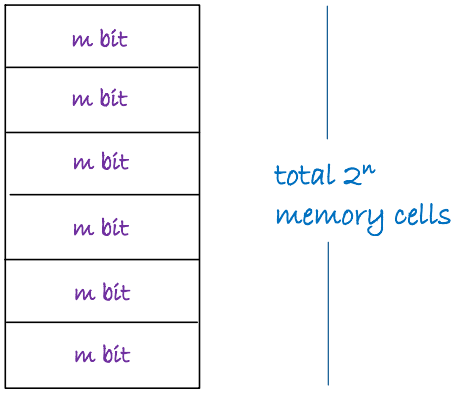


$N = 2^n$ memory location/cells
$N = 2^2 = 4$ memory location/cells



memory = 2^n x m
each cell size = m bit
total 2^n memory cells

to represent any of the cells among 2^n
n bit address are required.



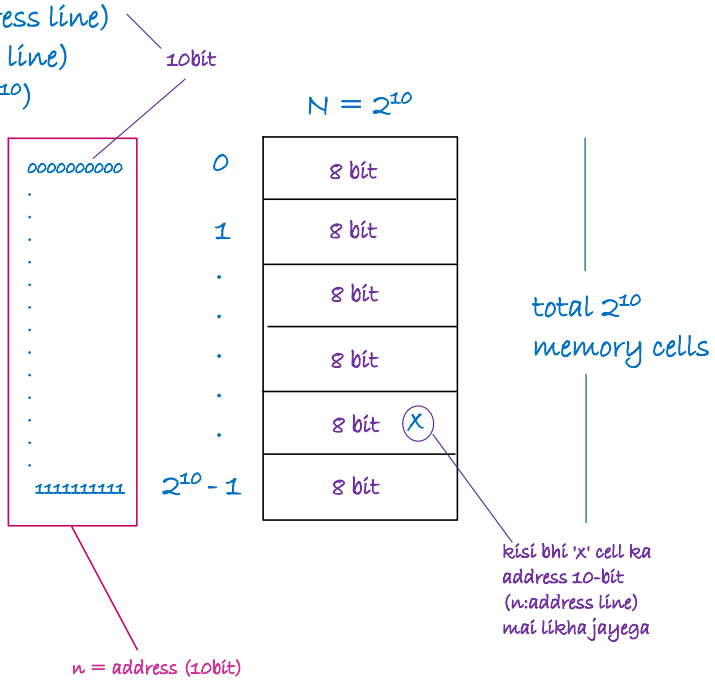
Questions :

Q1. Memory : 1k byte

$= 2^{10}$ x 8bit ($1k = 2^{10}$ and 1byte = 8bit)

10 bit address line (MAR)
8bit data line (MBR)
2^{10} memory cells / locations
10bit address is required to represent any cell (among 2^{10})

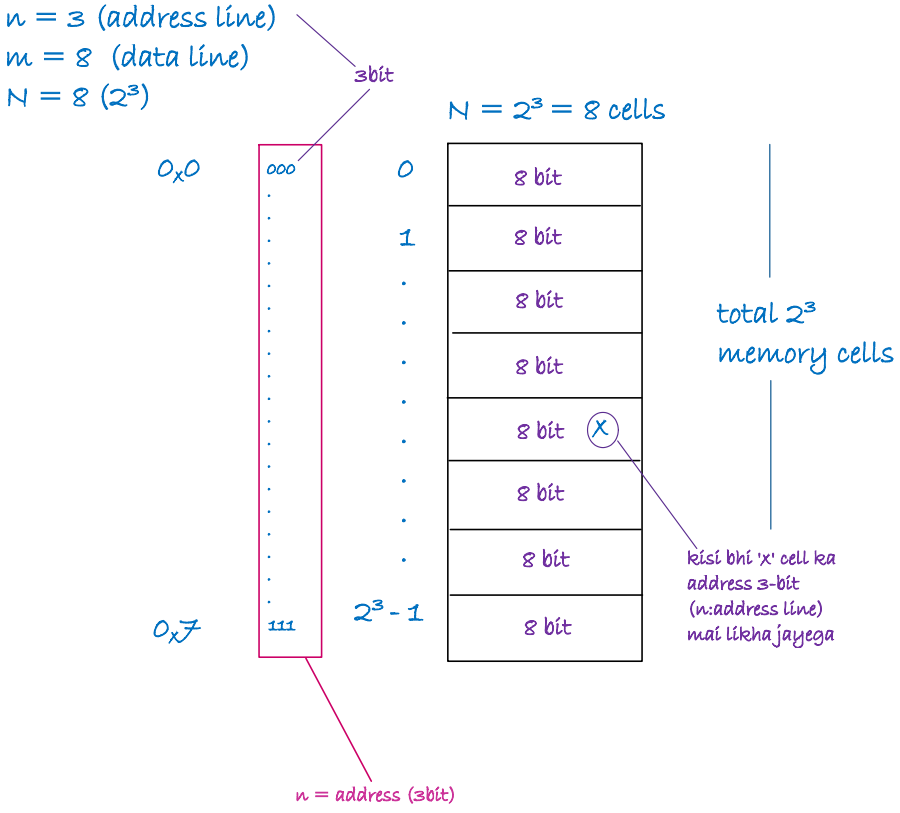
n = 10 (address line)
m = 8 (data line)
N = 1024 (2^{10})



Q2. Memory : 8 byte

$= 2^3 \times 8bit$ (2^3 and 1byte = 8bit)

3 bit address line (MAR)
8 bit data line (MBR)
2^3 memory cells / locations
3bit address is required to represent any cell (among 2^3)



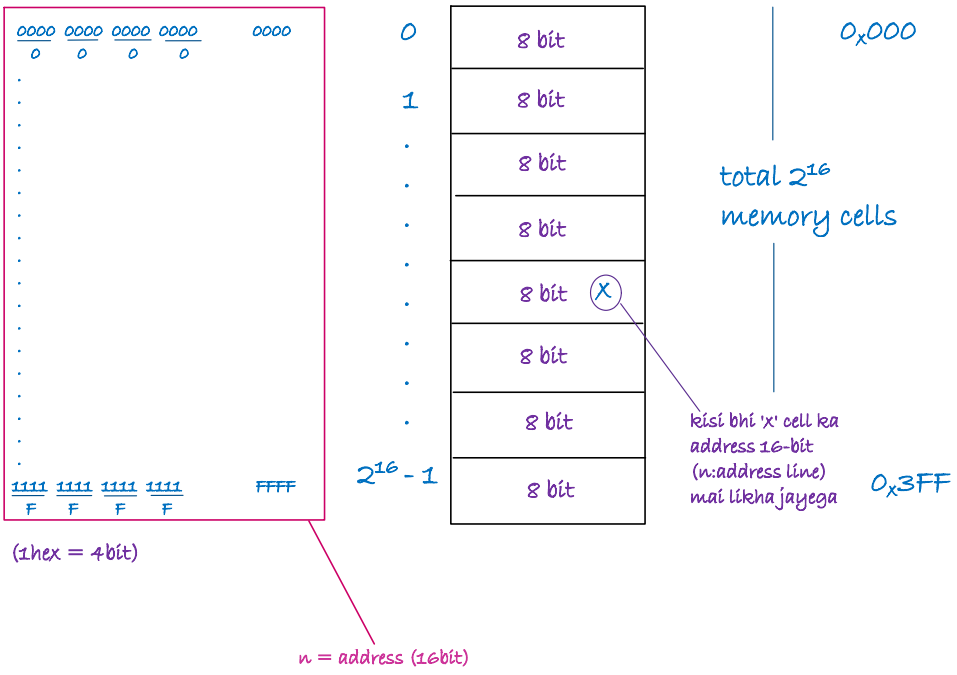
Q.3. Memory : 64kbyte

$= 2^6 \times 2^{10} \times 8bit$ (2^6 =64, 2^{10} =1k and 1byte = 8bit)
$= 2^{16} \times 8bit$

16 bit address line (MAR)
8 bit data line (MBR)
2^{16} memory cells / locations
16 bit address is required to represent any cell (among 2^{16})

n = 16 (address line)
m = 8 (data line)
N = 65536 (2^{16})

N = 2^{16} = 65536 cells

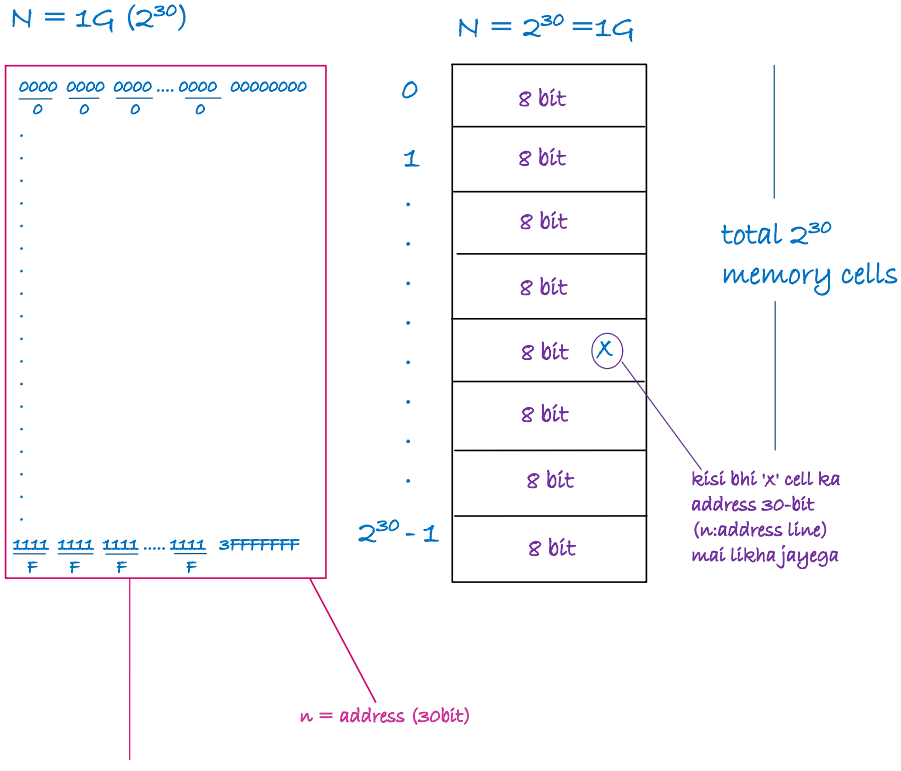


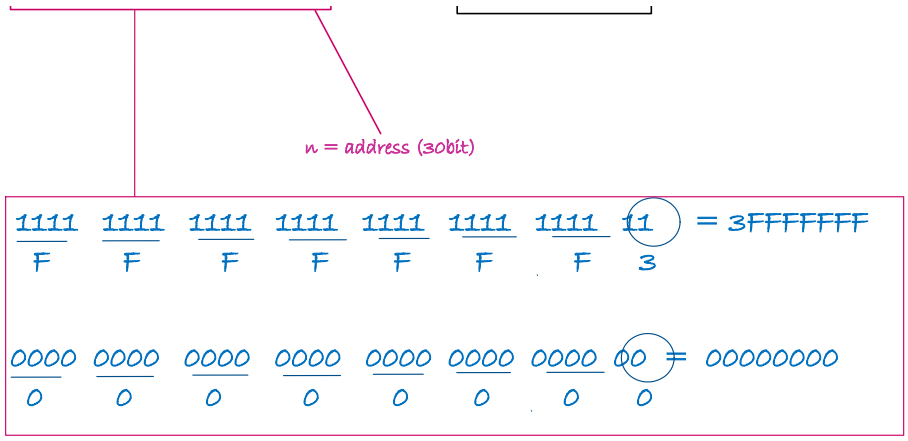
Q.3. Memory : 1Gbyte

$= 2^{30} \times 8bit$ ($2^{30} = 1G$ and $1byte = 8bit$)

30 bit address line (MAR)
8 bit data line (MBR)
2^{30} memory cells / locations
30 bit address is required to represent any cell (among 2^{30})

n = 30 (address line)
m = 8 (data line)
N = 1G (2^{30})





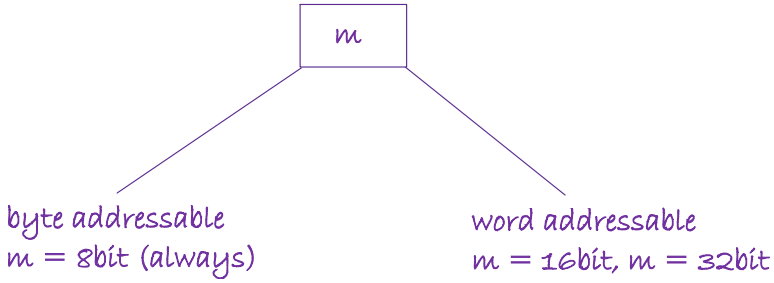
hexa decimal
symbol : 0_x or $()_{16}$

4bit pair :
0000 - 0
1111 - F

0000 - 0
1000 - 8
1001 - 9

A - 10
B - 11
C - 12
D - 13
E - 14
F - 15

4 binary bit : 1 Hex value



Terms and notations :

Digital computers : 0 and 1
Each is called bit : Binary digit
byte = 8bit

b : bits (bit : binary digit)
B : bytes (byte : group of 8bits)
W : words

Word : how many bits the CPU process at a time
term word is used to refer a group of bytes that is processed simuntaneously

32bit processor	64bit processor
word size = 32bit CPU perform operations on 32bit (4byte) data at a time	word size = 64bit CPU perform operations on 64bit (8byte) data at a time

the exact number of bytes that constitute a word depends on the system
for ex.
- pentium : 1 word = 4bytes = 32bits
- itanium : 1 word = 8bytes = 64bits

if 32bit processor that means 1 word = 32bits
if 64bit processor that means 1 word = 64bits

Sometimes we also use *doubleword* and *quadword*. A doubleword has twice the number of bits as the word and the quadword has four times the number of bits in a word.

based on the cell size, memory configuration is divided into types :

Byte addressable memory.

default

8 bit
8 bit
8 bit
8 bit
8 bit
8 bit
8 bit
8 bit

cell size = 8 bit
- byte addressable

Word addressable memory.

1 word
1 word
1 word

cell size = 1 word
size of word is given in computer
- word addressable

1. Byte addressable memory : when the cell size is 8 bit then corresponding address is byte addressable.

4x8=2bit address

16x8=4bit address

32x8=5bit address

64x8=6bit address

byte addressable because each cell size is 8 bit

256x8=8bit address

512x8=9bit address

1kx8=10bit address

1mx8=20bit address

2^nx8=nbit address

2. word addressable memory : when the cell size is given in the form of words (depends on word length) the corresponding address is word addressable.

4xw=2bit address

16xw=4bit address

32xw=5bit address

64xw=6bit address

256xw=8bit address

512xw=9bit address

1kxw=10bit address

1mxw=20bit address

2nxw=nbit address

cell size must be a word depends upon the word length of the processor

Microprocessor	memory Byte[B] (data line)	CPU word (data line) [1 word size = word length] (depends upon word length of the processor)	cells/ interface
8bit processor	8bit	8bit	1
16bit processor	8bit	16bit	2
32bit processor	8bit	32bit	4
64bit processor	8bit	64bit	8
.	.	.	.
.	.	.	.
.	.	.	.
x bit processor	8bit	x bit	x/8

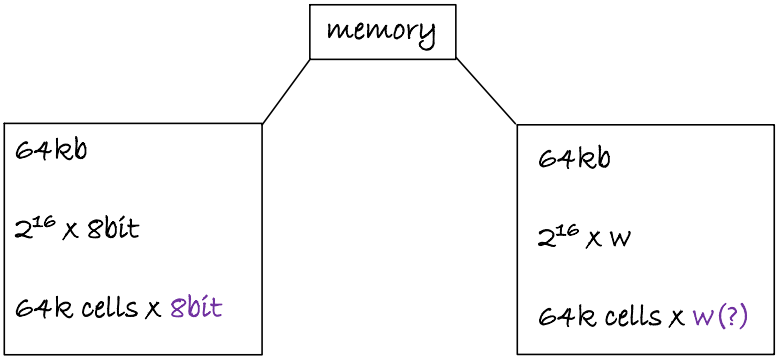
n bit processor

8bit

unique meaning
no ambiguity

nbit

no unique meaning
hence ambiguity



note:

- default memory configuration (data type in memory) is byte addressable so in the memory chip data is stored byte wise
- but default data type in the CPU is words. so the operation are performed on a CPU according to word format.
- to synchronise memory data type (byte) with CPU data type (word) memory interfacing will be adjusted by the designer
- according to the word length of the CPU, to access the data from memory to CPU (multiple byte access). kaunsa uthega : byte ordering

if my processor is working on 32 bits then i can say that my processor can perform operation on 32bit at a time but in memory chip it is stored in 8bit format so we have pick multiple memory cells.

ex. 16 bit proccessor

MOV A_x [6000] (instruction)

word length = 16bit = 2byte (multibyte) 2 consecutive bytes parallely

In CPU whatever operation performs will perform on 16bit data

M[6000] ——
M[6001] —— A_x

byte ordering [endian mechanism] :
memory chip is byte addressable
(little endian and big endian)

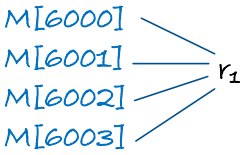
when we use multibyte data then byte ordering is used

ex. 32 bit proccessor

MOV r_1 [6000] (instruction:move to r_1)

word length = 32bit = 4byte (multibyte) 4 consecutive bytes parallely

In CPU whatever operation performs will perform on 16bit data



kaunsi byte uthani hai vo byte ordering (endian's mechanism) batata hai.

Questions :

Q. The Capacity of a memory unit is defined by the Number of word Multiplied by the number of bits/word. How many separate address and data lines are needed for a memory of 64K×16?

(a) 8 address, 8 data line (b) 16 address, 8 Data line

(c) 15 address, 16 Data line (d) 16 address, 16 Data line

GATE 2005

64K X16 ——— 2^{10} = 1K

2^{16} X 16 bit 2^{6} = 64

Q. Consider a system which has 1024 k words. Each word has the size of 32 bits then what is the capacity of memory in MB (Mega Byte) _____

1024 K word. — [2^{10} = 1K, 2^{10} = 1024]

$2^{10} \cdot 2^{10}$. 4 byte

$2^{20} \cdot 2^{2}$ Byte

2^{22} Byte ⇒ 4 MByte. [2^{20} = 1M] [2^{2} = 4]

1 Word = 32 Bit

1 word = 4 byte.

Q. A processor can support a maximum memory of 4GB, where the memory is word-addressable (a word consists of two bytes). The size of the address bus of the processor is at least 31 bits. [GATE-2016]

memory : 4Gb $2^{2} \cdot 2^{30}$ byte [1giga = 2^{30}] 2^{32} byte

generally, 1 word = 4byte
but in given que. 1 word = 2bytes

4Gbyte

2G x 2byte = $2^{30} \cdot 2^{1} = 2^{31}$

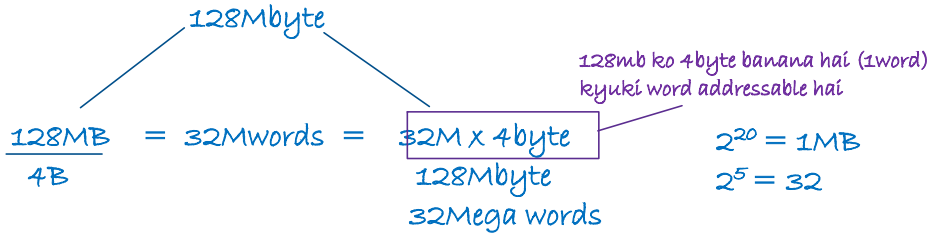
$\frac{4\text{Gbyte}}{2\text{byte}}$ = 2Gwords 2^{31} = 31 words

Q. consider a 32 bit hypothetical processor which support 128Mbyte memory. system is enhanced (new design) with a word addressable memory. then how many address line are required in the new system?

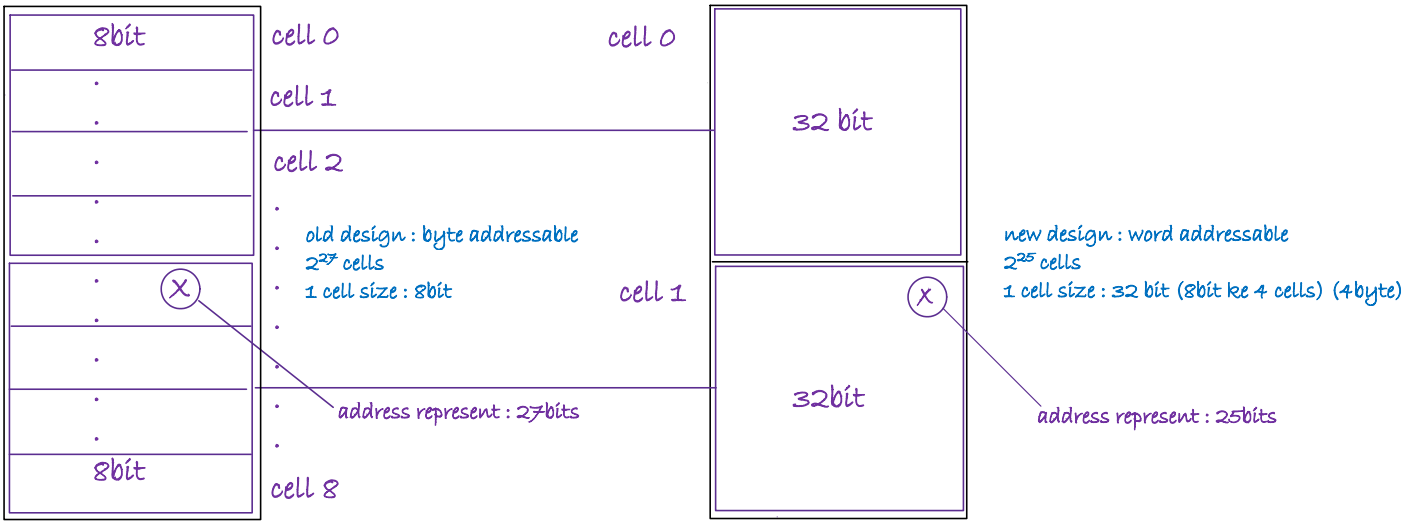
if only 128Mbyte

$2^8.2^{20}$ x 8bit
2^{27}byte
Address = 27bit

but new design : word addressable
1 word = 32bit
1word = 4byte



2^{25} word



note :
8 bit : byte addressable
multiple of 8 : word addressable

Q. consider a 64 bit hypothetical processor which support 512Gbyte memory. system is enhanced (new design) with a word addressable memory. then how many address line are required in the new system?

512Gbyte
= 2^{30} x 2^9 x 8 bit
= 2^{39}

1 word = 64bit

1word = 8byte

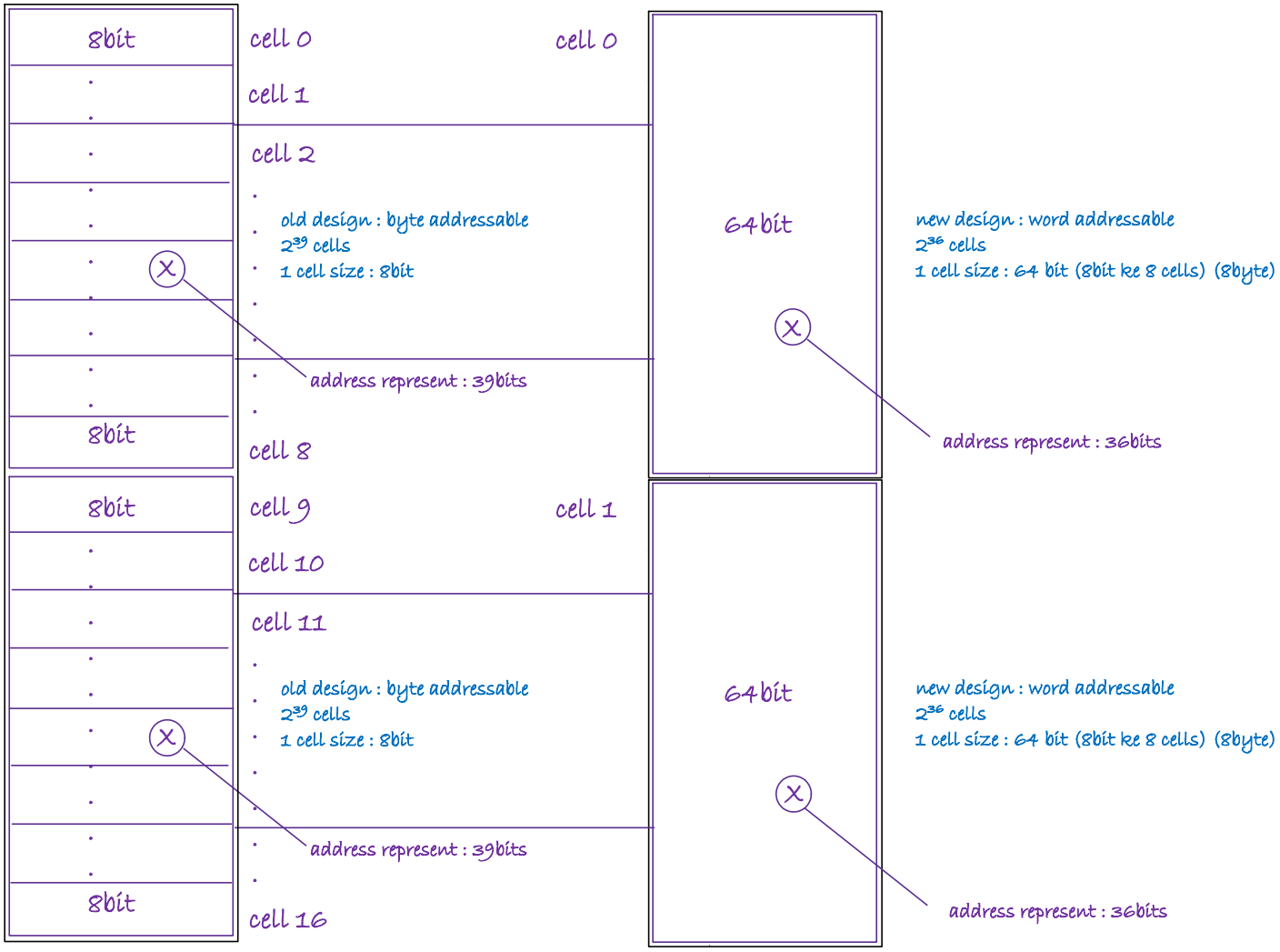
512Gbyte

512gb ko 8byte banana hai (1word) kyuki word addressable hai

$\frac{2^{30} \times 2^{9}}{2^{3}} = \frac{512GB}{8B} = 64\ Gwords = 64G \times 8byte$

512Gbyte
64Gwords

2^{36} word



Q. consider a 64 bit hypothetical processor which support 4Gwords memory. system is enhanced (new design) with a byte addressable memory. then how many address line are required in the new system?

4G words
$= 2^{30} \times 2^{2}$
$= 2^{32}$

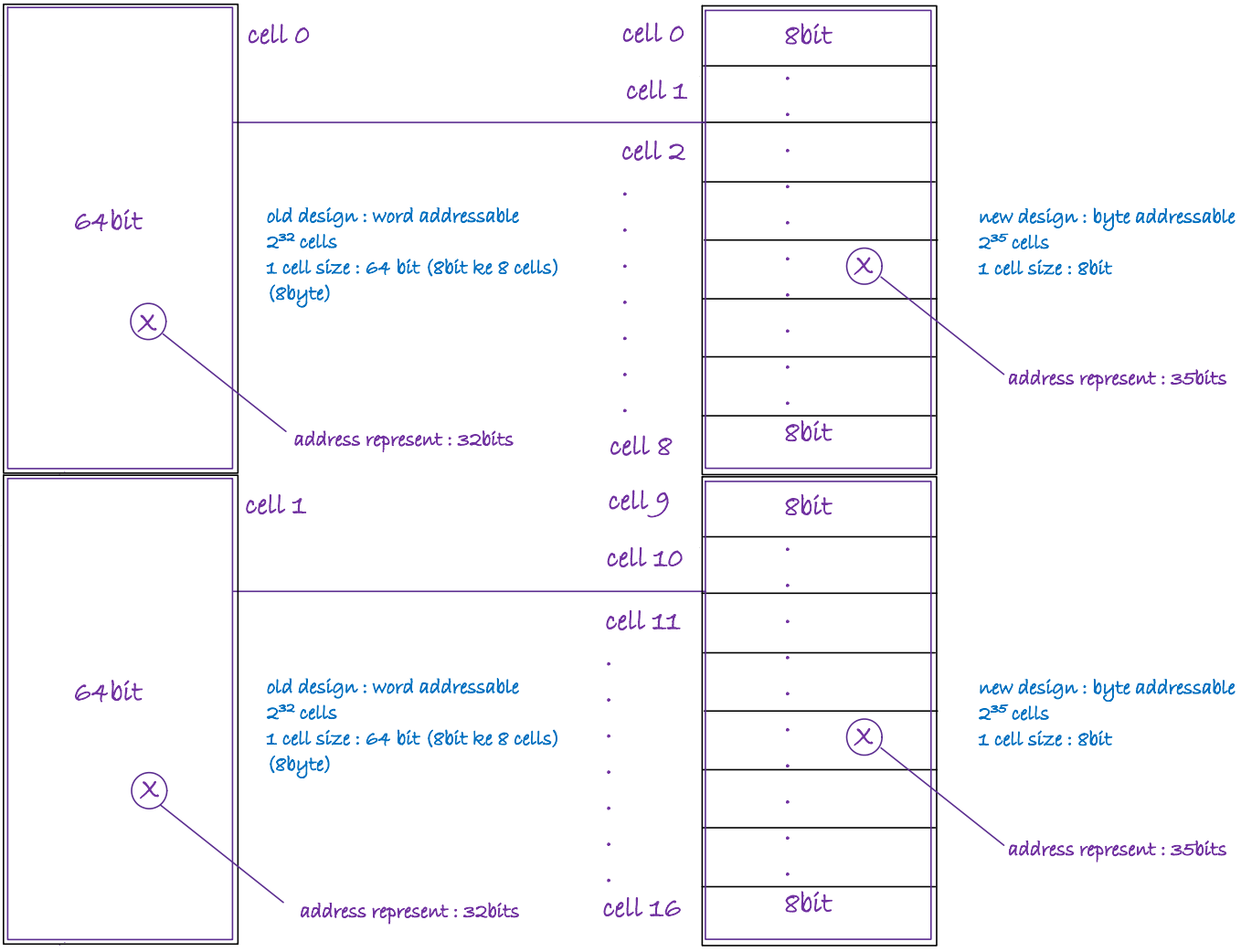
1 byte = 8 bit
1 word = 64bit
1 word = 8 bute

4Gwords
$= 2^{32} \times 8byte$
$= 2^{35}$

4G words
$= 2^{30} \times 2^{2}$
$= 2^{32}$

1 byte = 8 bit
1 word = 64 bit
1 word = 8 byte

4Gwords
$= 2^{32} \times 8$byte
$= 2^{35}$
$= 2^{30} \times 2^{5}$
$= 1G \times 32$
$= 32G$byte.



note :

in the processor design operation are always perfomed on word format so when the word length of the CPU/Processor is greater than 8bit then multiple byte (cells) accessing is required to process the data parallely.

Q. how to fill memory (starts from 1000 and 4 byte. 1 word = 4byte)

acceptable

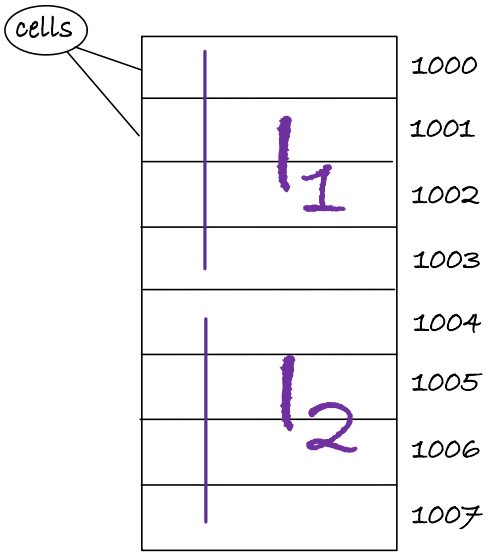
1000
1001
1002
1003

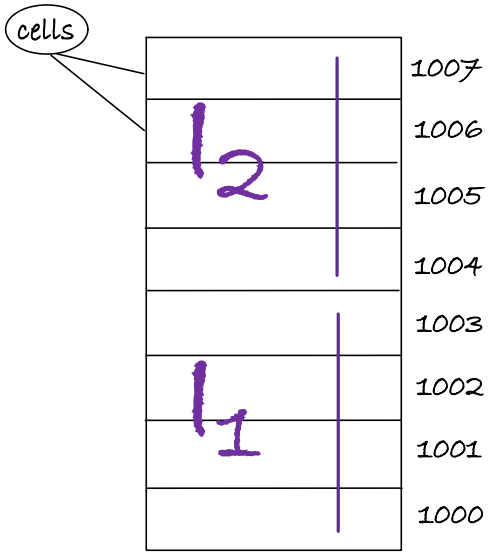
(OR)

1000
999
998
997

never acceptable

two ways to store: 1000 - 1003 : i_1 and 1004 - 1007 : i_2





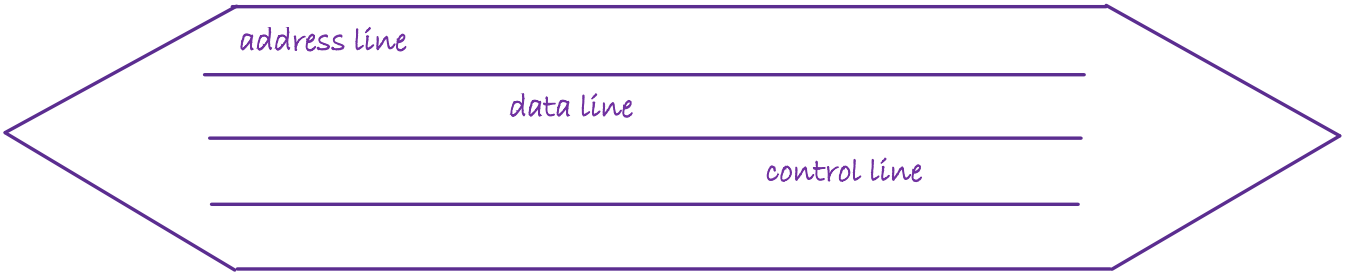
note :

memory always in increasing order.

system bus

system bus is collection of lines/wires which are used to provide the communication (transmission media) between the components of the computer [CPU, I/O, memory, etc].

collection of lines/wires/electrical lines (not wireless) that provides communication/transmission media between the components of the computer.



system bus

note :

one line carry 1 bit of data at any point of time.

system bus contain 3 types of buses
(i) Address bus (line)
(ii) Data bus (line)
(iii) Control bus (line)

(i) Address bus : Collection of address lines.
- Address line : Address line are used to carry the address towards memory and I/O

note :

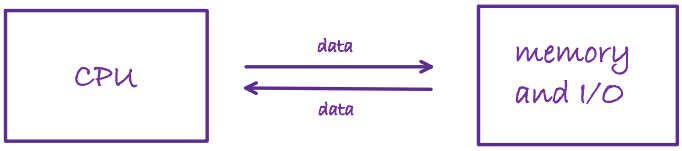
Address lines are uni-directional (from CPU to memory and I/O or components of system)

(ii) Data bus (line) : Collection of data lines

- Data lines : Data lines are used to carry the data (binary sequence)

note :

Data lines are bi-directional (from CPU to memory and I/O and vice-versa)

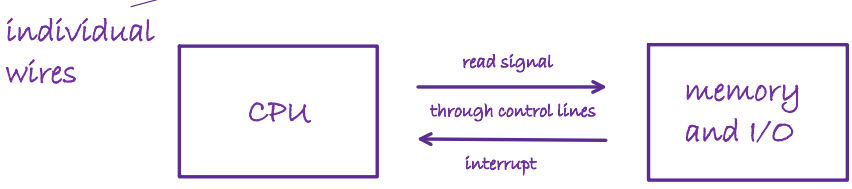


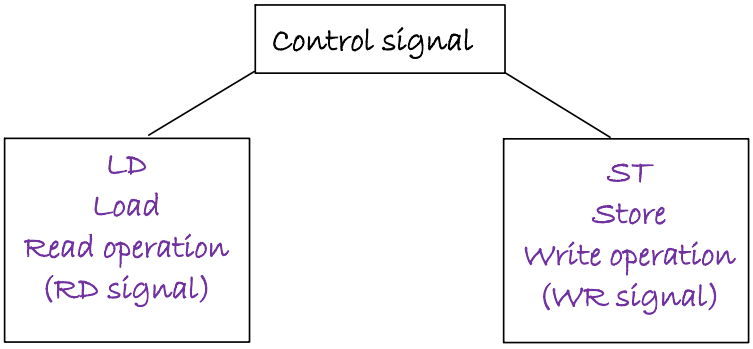
(iii) Control bus (line) : Collection of control lines

- Control lines : Control lines are used to carry the control signals

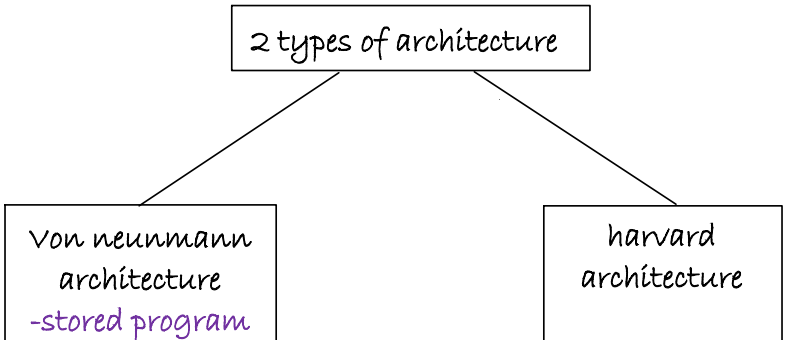
note :

Control lines are individual uni-directional (from CPU to memory and I/O and vice-versa)





control signal generated by control unit (CU).
control unit is supervisior of system that control every activity



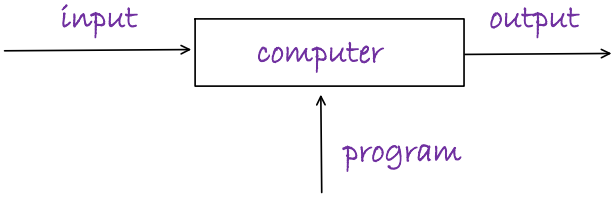
architecture
-stored program concept
-Computer works on stored program concept

architecture

Von neunmann architecture (stored program concept)

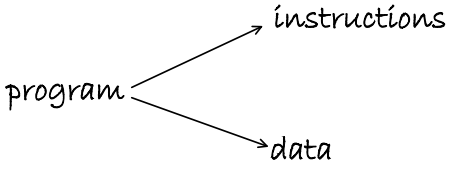
- main memory contain the instruction and data
- ALU operating on binary data
- control unit interpret the instruction from memory and executes it
- Input/output equipment operated by control unit with the help of control signals.

computer : it is a computational device used to process the data under the control of program.



function of computer is program execution.

program : it is sequence of instructions along with the data



instruction : its a binary sequence which is designed inside the processor to perform some task

computer binary ki language samzta hai, agar usko program assign krke toh vo 101010 ki format mai krega.

binary sequence - bind with - operation
[101010101010] (Instruction ka binary : processor knows)

data : its a binary sequence which is associated with a value [based on data format]

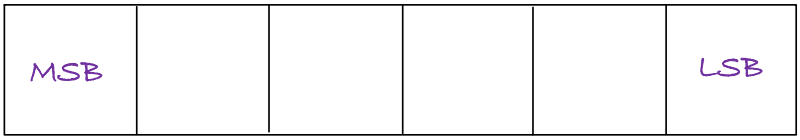
computer binary ki language samzta hai, agar usko program assign krke toh vo 101010 ki format mai krega.

binary sequence - bind with - values
[101010101010] (data ka binary : we know it)

topic 8 (byte ordering)

Monday, July 8, 2024 4:48 AM

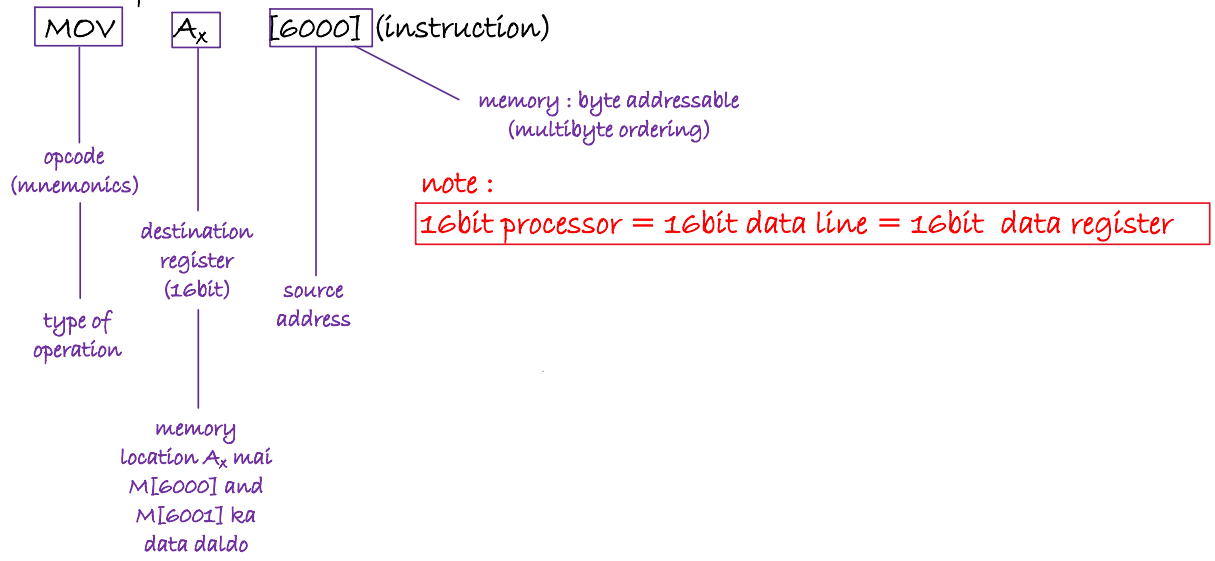
byte ordering



most significant portion of data. [big end of data]

least significant portion of data. [little end of data]

ex. 16 bit proccessor



note :

16bit processor = 16bit data line = 16bit data register

word length = 16bit = 2byte (multibyte)

In CPU whatever operation performs will perform on 16bit data

M[6000] —
M[6001] — Ax

Data in hexadecimal (16bit data)

Data = 0X 11 21

Data = $(11\ 21)_{16}$

(16bit binary)

Data : 0001 0001 0010 0001

Hexa decimal : 1 1 2 1

4 HEX digit
4 x 4 = 16 bit.

2 bytes of data stored in A_x

2 ways :

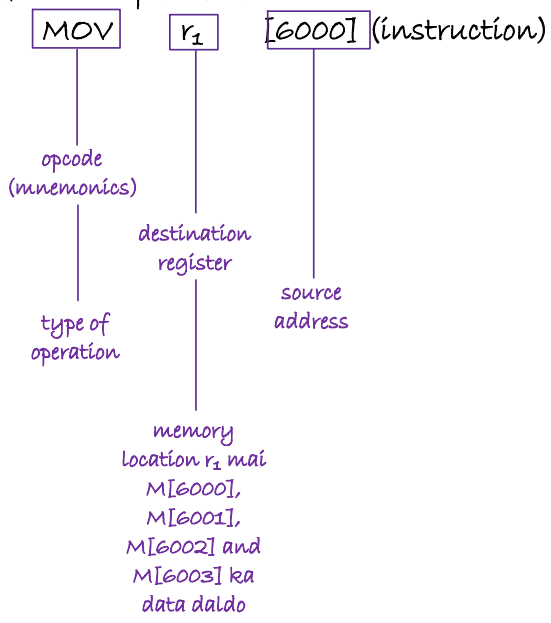
(i) A_x = | 11 | 21 |

(OR)

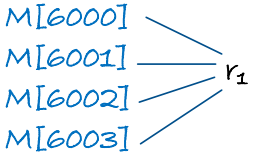
(ii) A_x = | 21 | 11 |

one instruction and 2 output
hence creates ambiguity

ex. 32 bit proccessor



word length = 32bit = 4byte (multibyte)
In CPU whatever operation performs will perform on 16bit data



Data in hexadecimal (32bit data)

Data = 0x 11 21 53 56

Data = $(11\ 21\ 51\ 56)_{16}$

(32bit binary)

Data : 0001 0001 0010 0001 0101 0001 0101 0110

Hexa decimal : 1 1 2 1 5 1 5 6

8 HEX digit

8 x 4 = 32 bit.

4 bytes of data stored in r_1

2 ways :

(i) r_1 = | 11 | 21 | 51 | 56 |

(OR)

(ii) r_1 = | 56 | 51 | 21 | 11 |

one instruction and 2 output hence creates ambiguity

- why endian mechanism required?

to solve the problem of ambiguity, endian (byte ordering) mechanism is required.

note :

- memory chip is byte addressable, so in the memory chip data is stored byte wise.
- in the processor operation are performed on word format. when word size is greater than 8bit [1byte] (i.e

Q. memory size = 512GByte

then the number of bits in address if -

(i) byte addressable

(ii) word addressable with 1 word = 16bit

(iii) word addressable with 1 word = 64bit

- 512GB

$= 2^{30} \times 2^{9} \times 8$ bit

$= 2^{39}$

- 512GB (word size = 16bit) (2byte)

$= \frac{512GB}{2byte}$

= 256Gword
= $2^{30} \times 2^{8}$
Address = 38bit

- 512GB (word size = 32bit) (4byte)
= 512GB
4byte

= 128Gword
= $2^{30} \times 2^{7}$
Address = 37bit

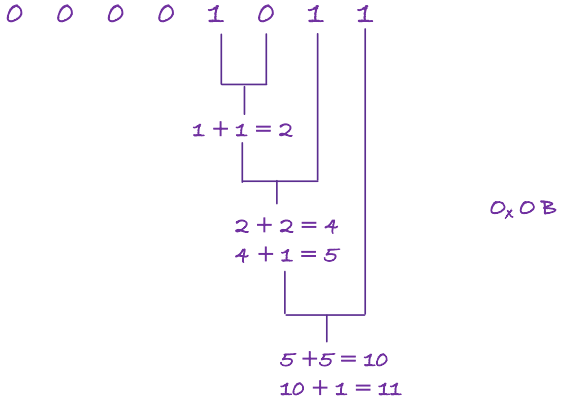
Q. consider a 32bit processor with memory = 128Gword then no. of address bits if memory is byte addressable

1 word = 32bit = 4byte
128Gword = 128G x 4byte = 512Gbyte = $2^{30} \times 2^{9}$
= $2^{37} \times 2^{2}$
= 2^{39}

Conversions :

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

write 11 in 8 bit :



$0_x 0B$

BINARY HEX

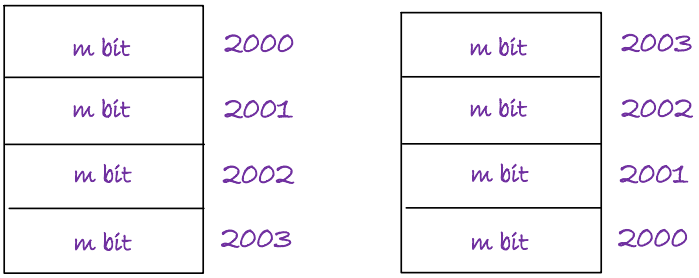
0000 : 0 1010 = 10 = A

BINARY		HEX
0000 : 0		
0001 : 1		
0010 : 2	1010 = 10 =	A
0011 : 3		
0100 : 4	1011 = 11 =	B
0101 : 5		
0110 : 6	1100 = 12 =	C
0111 : 7		
1000 : 8	1101 = 13 =	D
1001 : 9		
	1110 = 14 =	E
	1111 = 15 =	F

why 4 digit?
because in hexadecimal
1hex value = 4bit
hex = 0-9 and A to F

Binary	Hexa decimal
8bit : 0100 1001	$(0_x 49)$ or $(49)_H$
16bit : 0001 0010 0011 0100	$(0_x 1234)$ or $(1234)_{16}$
32bit : 0001 1010 0010 1011 1 10 2 11 A B	$(0_x 1A\ 2B\ 3C\ 4D)$ $(0_x 1A\ 2B\ 3C\ 4D)_{16}$
0011 1100 0100 1100 3 12 4 13 C D	
32bit : 0101 1011 1001 1110 5 B 9 E	$(0_x 5B\ 9E\ 21\ B3)$ $(0_x 5B\ 9E\ 21\ B3)_{16}$
0010 0001 1011 0011 2 1 B 3	
32bit : 00000001 00000010 01 02	$(0_x 01\ 02\ 03\ 04)$ $(0_x 01\ 02\ 03\ 04)_{16}$
00000011 00000100 03 04	

- memory starting from 2000:
32bit [4byte]



Q. consider 32bit data 0_x (1A 2B 3C 4D) stored at memory location 2000 onwards then result is ?

solution :

Data : 0_x (1A 2B 3C 4D)

two ways of storing data : 32bit data

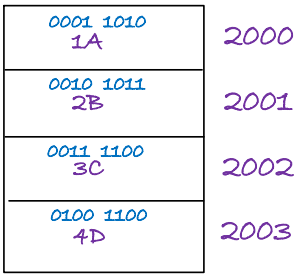
1A	2B	3C	4D
0001 1010	0010 1011	0011 1100	0100 1101
8bit	8bit	8bit	8bit
(4)	(3)	(2)	(1)

(or)

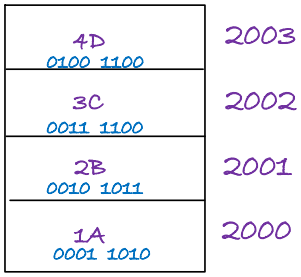
(1)	(2)	(3)	(4)

one instruction create 2 output that is creating ambiguity, to solve the problem of ambiguity there is a endian's mechanism

byte ordering : for multibyte and memory is byte addressable
1 word = 32bit = 4byte



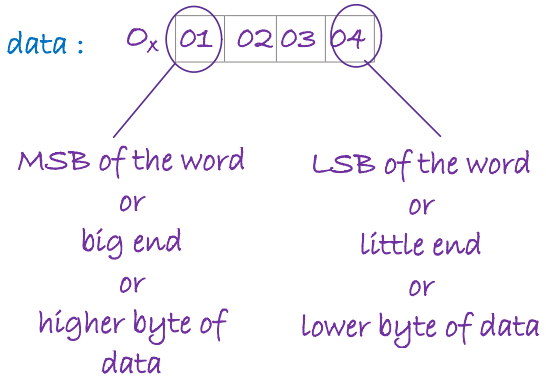
byte addresable hai 8 bit rakhega



byte addresable hai 8 bit rakhega

ex.1. 32bit data : O_x (01 02 03 04)

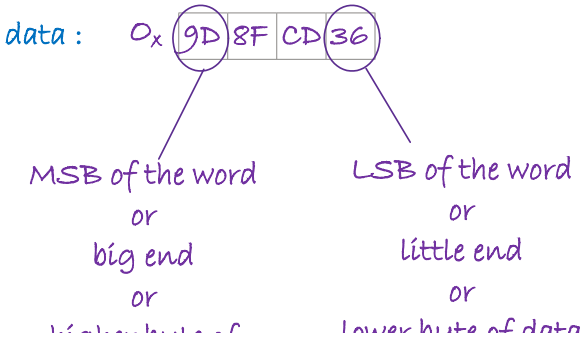
0000 0001 0000 0010 0000 0011 0000 0100



ex.2. 32bit data : O_x(9D 8F CD 36)

1001 1101 1000 1111 1100 1101 0011 0110

9 D 8 F C D 3 6



big end
or
higher byte of
data

little end
or
lower byte of data

Topic - byte ordering : when data is multibyte we use byte ordering

types :

1. Little endian : lower address contain lower bytes and higher address contain higher bytes.

(OR)

right to left

(OR)

little endian starts with little end and store at lower memory address

(OR)

little end of the word stored at lowest memory address

Q. 32bit data : (9D 8F CD 36) stored at memory location starting from 1000 then store the data in little endian format

1001 1101 1000 1111 1100 1101 0011 0110

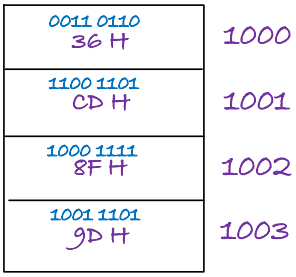
9 D 8 F C D 3 6

data : 0_x | 9D | 8F | CD | 36 |

MSB of the word
or
big end

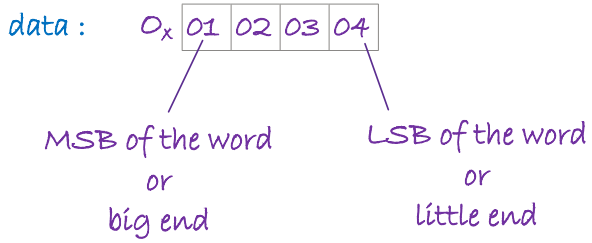
LSB of the word
or
little end

little end : starts from little end and little end stored at lowest memory address

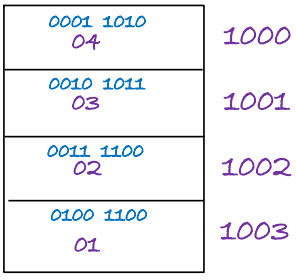


byte addresable hai 8 bit rakhega

Q. 32bit data : 0_x(01 02 03 04) stored at memory location starting from 1000 then store the data in little endian format



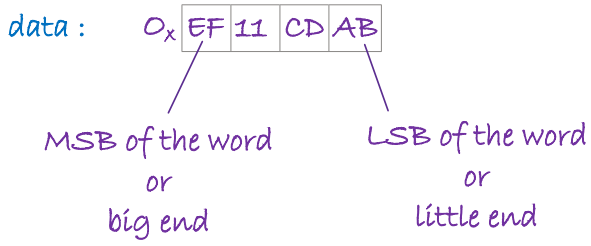
little end : starts from little end and little end stored at lowest memory address



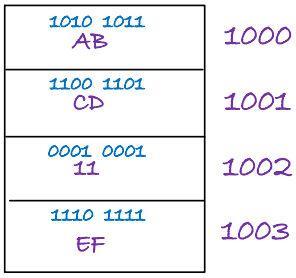
byte addresable hai 8 bit rakhega

Q. 32bit data : 0_x(AB CD 11 EF) stored at memory location starting from 1000 then store the data in little endian format

endian format

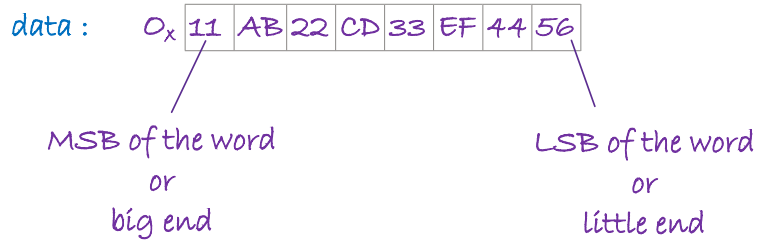


little end : starts from little end and little end stored at lowest memory address

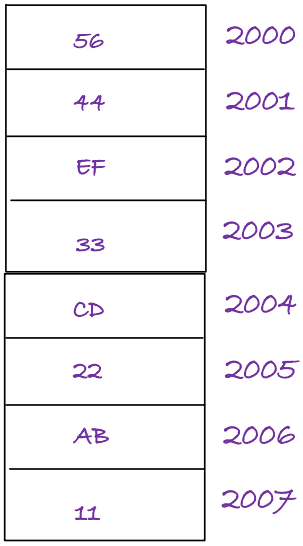


byte addresable hai 8 bit rakhega

Q. 64bit data : O_x(11 AB 22 CD 33 EF 44 56) stored at memory location starting from 2000 then store the data in little endian format.



little end : starts from little end and little end stored at lowest memory address



byte addresable hai 8 bit rakhega

2. Big endian : lower memory address contain higher byte and higher memory address contain lower byte.

(OR)

left to right

(OR)

start from the MSB of word stored at lower memory address.

(OR)

big end of word stored at lowest memory address.

Q. 32bit data : O_x(9D 8F CD 36) stored at memory location starting from 1000 then store the data in big endian format

1001 1101 1000 1111 1100 1101 0011 0110

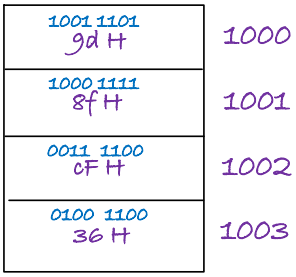
9 D 8 F C D 3 6

data : O_x | 9D | 8F | CD | 36 |

MSB of the word or big end

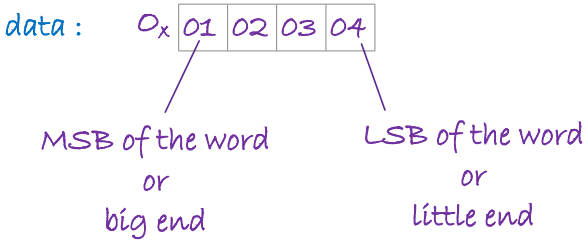
LSB of the word or little end

big end : starts from big end and big end stored at lowest memory address

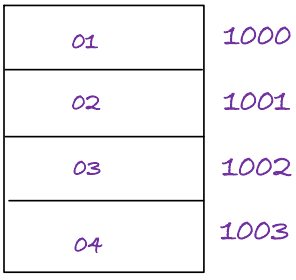


byte addresable hai 8 bit rakhega

Q. 32bit data : O_x(01 02 03 04) stored at memory location starting from 1000 then store the data in big endian format



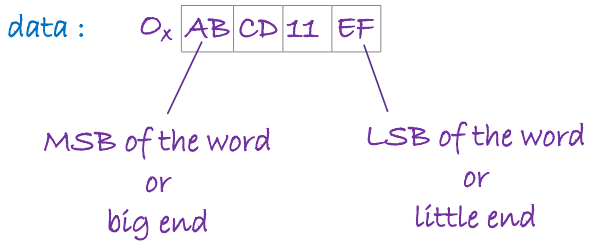
big end : starts from big end and big end stored at lowest memory address



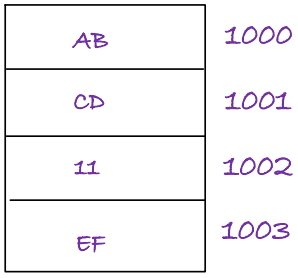
byte addresable hai 8 bit rakhega

Q. 32bit data : O_x(AB CD 11 EF) stored at memory location starting from 1000 then store the data in Big endian format

endian format

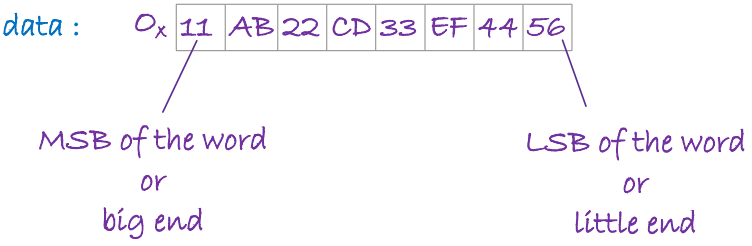


big end : starts from big end and big end stored at lowest memory address

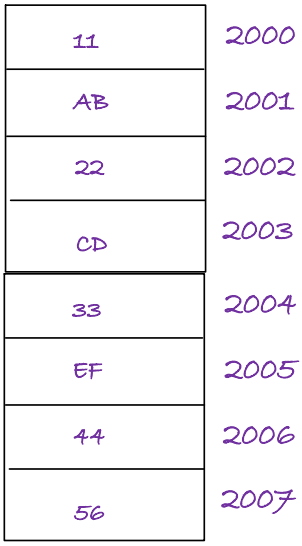


byte addresable hai 8 bit rakhega

Q. 64bit data : O_x(11 AB 22 CD 33 EF 44 56) stored at memory location starting from 2000 then store the data in big endian format.



big end : starts from big end and big end stored at lowest memory address



byte addresable hai 8 bit rakhega

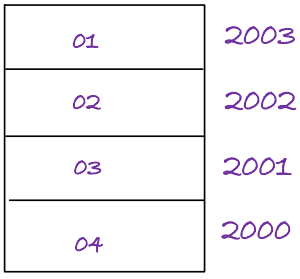
TYPE : 2

when data is already stored in main memory and we have to write in little endian and big endian format?

little endian : starts from little end and little end stored at lower address.

big endian : starting from big end and big end stored at lower address

Q. write in little and big endian format.



step one : write down the data

data : 0_x | 04 | 03 | 02 | 01 |

because address started in descending order

MSB of the word
or
big end

LSB of the word
or
little end

step two : write down in little and big endian

Little endian :

0_x	01	02	03	04

big endian :

0_x	04	03	02	01

Q. write in little and big endian format.

BA	2003
09	2002
24	2001
AB	2000

step one : write down the data

data : 0_x | AB | 24 | 09 | BA |

because address started in descending order

MSB of the word
or
big end

LSB of the word
or
little end

step two : write down in little and big endian

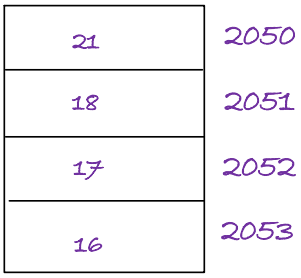
Little endian :

0_x	BA	09	24	AB

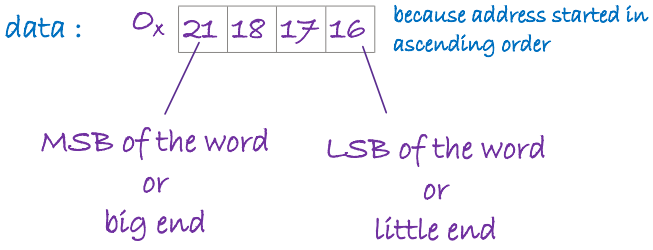
big endian :

0_x	AB	24	09	BA

Q. write in little and big endian format.

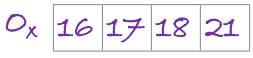


step one : write down the data

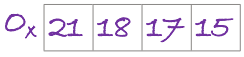


step two : write down in little and big endian

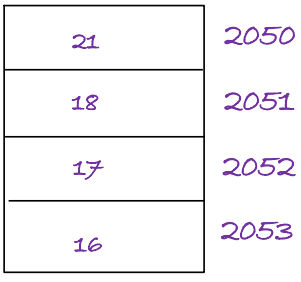
Little endian :



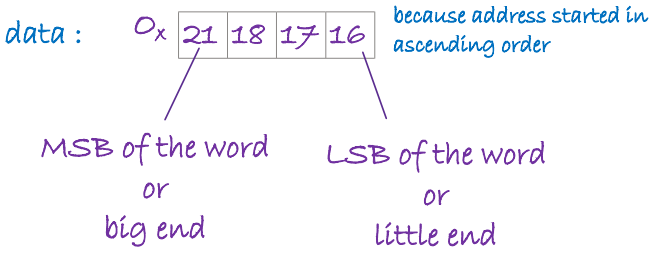
big endian :



Q. Identify the data format.



step one : write down the data



step two : write down in little and big endian

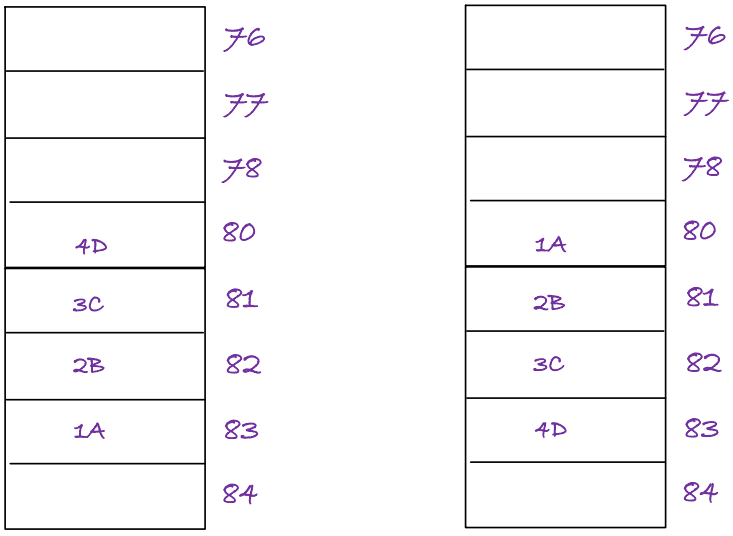
Little endian :

0x	16	17	18	21

big endian :

0x	21	18	17	16

Q. consider a CPU having a memory address 0 to 200. 32bit data stored at memory location 80 onward.



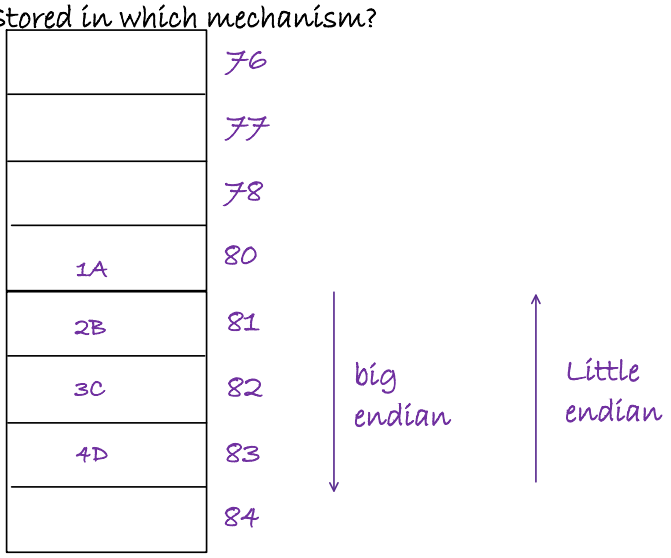
TYPE : 3

By watching memory layout how to write in little endian and big endian.

step by step !

q. Stored in which mechanism?

q. Stored in which mechanism?



step one : write down the data

data : O_x | 1A | 2B | 3C | 4D |

because address started in ascending order

MSB of the word or big end

LSB of the word or little end

step two : write down in little and big endian

Little endian :

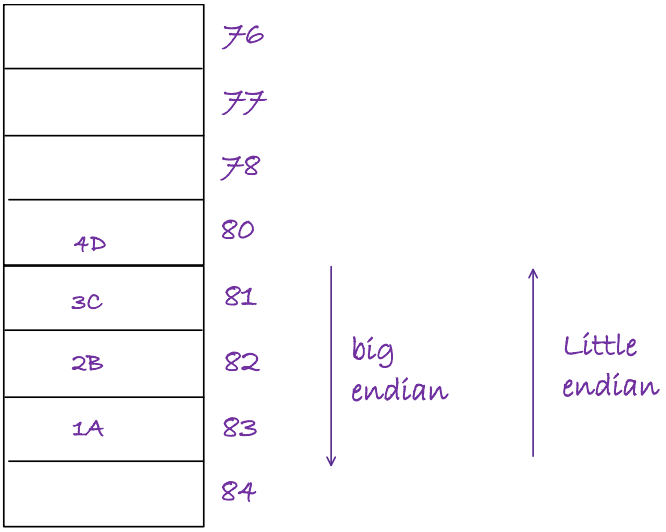
O_x | 4D | 3C | 2B | 1A |

big endian :

O_x | 1A | 2B | 3C | 4D |

Conclusion : this data is stored in big endian.

q. Stored in which mechanism?



step one : write down the data

data : 0_x | 4D | 3C | 2B | 1A | because address started in ascending order

MSB of the word or big end

LSB of the word or little end

step two : write down in little and big endian

Little endian :

0_x | 1A | 2B | 3C | 4D |

big endian :

0_x | 4D | 3C | 2B | 1A |

Conclusion : this data is stored in big endian.

Practise sheet :

Question 1 01:10

Suppose an integer is stored as 5 bytes, then a variable 'x' with value 0 × 0123456789 will be stored as:

Select one or more answers

Question 1

01:10

Suppose an integer is stored as 5 bytes, then a variable 'x' with value 0 × 0123456789 will be stored as:

Select one or more answers

A

0 × 100	0 × 101	0 × 102	0 × 103	0 × 104
0 1	2 3	4 5	6 7	8 9

in Little Endian

B

0 × 100	0 × 101	0 × 102	0 × 103	0 × 104
8 9	6 7	4 5	2 3	0 1

in Little Endian

C

0 × 100	0 × 101	0 × 102	0 × 103	0 × 104
0 1	2 3	4 5	6 7	8 9

in Big Endian

D

0 × 100	0 × 101	0 × 102	0 × 103	0 × 104
8 9	6 7	4 5	2 3	0 1

in Big Endian

Solution

In **little endian** representation, lower address of the memory contains lower byte of data and higher address of the memory contains higher byte of data .

0 × 100	0 × 101	0 × 102	0 × 103	0 × 104
8 9	6 7	4 5	2 3	0 1

In **Big endian** representation, lower address of the memory contain higher byte and higher address contain lower byte of data.

0 × 100	0 × 101	0 × 102	0 × 103	0 × 104
0 1	2 3	4 5	6 7	8 9

Ans. (2), (3)

Question 2

Suppose we are given 2 bytes of data as $_{0000000100101100}$. How it will be stored in Little and Big endian mechanism?

Select one or more answers

✓

0 × 100	0 × 101
00000001	00101100

in Big Endian

✓

0 × 100	0 × 101
00101100	00000001

in Little Endian

C

0 × 100	0 × 101
00101100	00000001

in Big Endian

D

0 × 100	0 × 101
00000001	00101100

in Little Endian

Result

Your answer was correct

Difficulty level: Easy - 28% learners got this correct

1m 38s

Others solved it in **53m 23s** avg

Your answer was correct
Difficulty level: Easy - 28% learners got this correct

1m 38s
Others solved it in **53m 23s** avg

Solution

In Big endian, first byte of binary representation is stored first.

0 × 100	0 × 101
00000001	00101100

in Big Endian

In Little endian, last byte of binary representation is stored first.

0 × 100	0 × 101
00101100	00000001

in Little Endian

Ans. (1), (2)

Question 3

Consider a hexadecimal number 5C4BAD9D. It is stored in little endian format from memory address 1000 H. Then what values will be stored in memory locations 1000 H, 1001 H, 1002 H, and 1003 H respectively.

Select your answer

A C5, B4, DA, D9

B D9, DA, B4, C5

C 5C, 4B, AD, 9D,

✓ 9D, AD, 4B, 5C

Result

Your answer was correct
Difficulty level: Easy - 26% learners got this correct

28s
Others solved it in **11m 7s** avg

Solution

In little Endian 5C4BAD9D will be stored as:

1000 H	1001 H	1002 H	1003 H
9D	AD	4B	5C

In Little Endian mechanism lower byte of data is stored in lower memory address location.

Question 4

Question 4

Consider a hypothetical processor that supports 48-bit words and 64 KB main memory. The content of main memory is_____.

	⋮
A028	GA
A029	TE
A030	UN
A031	AC
A032	AD
A033	EM
A034	20
A035	22
	⋮

If the data is stored in the main memory with a starting address of $(A030)_H$ and storing mechanism follows Big-Endian, what is the data word fetched from the main memory?

Select your answer

A GATEUNACADEM

✓ UNACADEM2022

C MEDACANUETAG

D 2202MEDACANU

Result

Your answer was correct

Difficulty level: Easy - 46% learners got this correct

37s

Others solved it in 6m 46s avg

Solution

Big Endian

⇓

Higher address contains least significant byte

Since data is stored from starting address $(A030)_H$ and processor support 48-bit words

⇓

6 subsequent blocks including $(A030)_H$ needs to be fetched.

UN	AC	AD	EM	20	22
[A030]	[A031]	[A032]	[A033]	[A034]	[A035]

Question 5

Consider a hexadecimal number F86CDA7B. It is stored in little endian format from memory address 1000 H. Then what values will be stored in memory locations 1000 H, 1001 H, 1002 H, and 1003 H respectively.

Select your answer

A B7, AD, C6, 8F

B DA, 7B, C6, 8F

C 6C, F8, 7B, DA,

✓ 7B, DA, 6C, F8

Result

Your answer was correct
Difficulty level: Easy - 48% learners got this correct

17s
Others solved it in 55m 49s avg

Solution

In little Endian F86CDA7B will be stored as:

1000 H	1001 H	1002 H	1003 H
7B	DA	6C	F8

In Little Endian mechanism lower byte of data is stored in lower memory address location.
Ans. (4)

note :
In memory data stored from starting address and in increasing order.
[2000], [2001], [2002].....

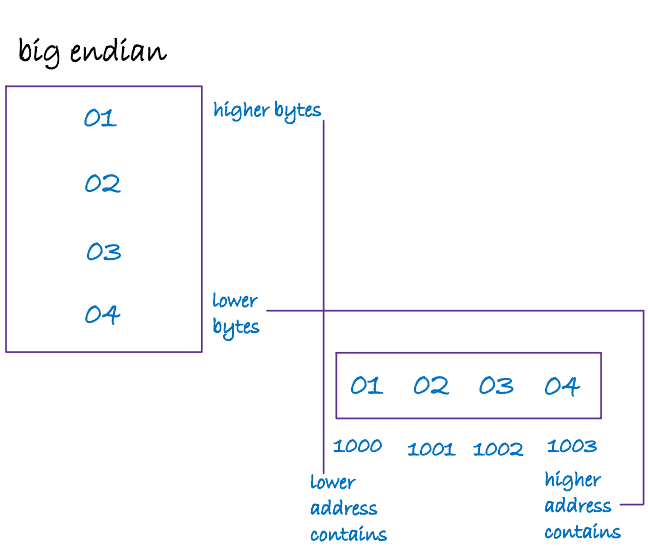
Little endian	Big endian
starts from the little end and little end stored at lower address	starts from the big end and big end stored at lower address

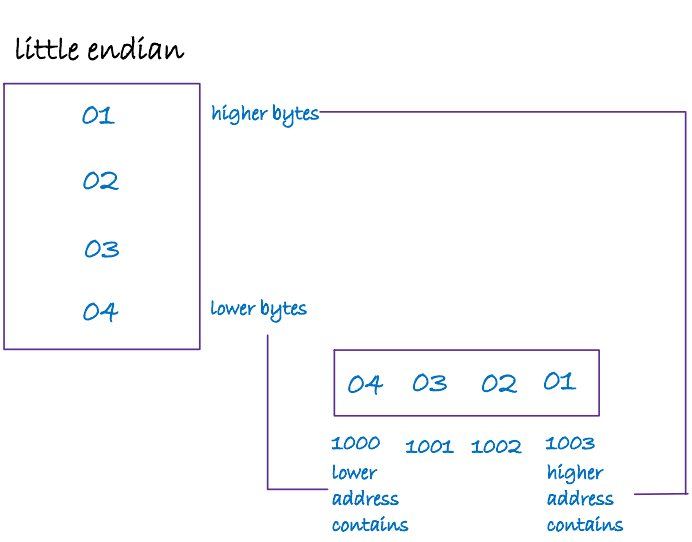
Points about byte ordering concept :

1.Endian ness (little endian and big endian) is used in processor design time. it means endian ness is the property of the CPU (processor), not the property of main memory.

2. Endianness does effect the ordering of data item (does apply) on multibyte data value, individual data items.

3. Endianess does not effect the ordering of data item (does not apply on) structure like strings, arrays and struct for every individual data item BUT (if multibyte data) then endianness (byte ordering) concept is applied.
- If data structure : 2 byte (multibyte) then each with endianess
- If data structure : 1 byte then not with endianess





type : 4
if data is given in little endian then how to write in big endian.

Q. O_x 0001
O_x 6665
O_x 4243
O_x 0100
2byte unsigned integer, stored in little endian format then write in big endian.

little endian big endian data

little endian	big endian	data
1000 1001 66 65	1000 1001 65 66	0_x 66 65
4000 4001 00 01	4000 4001 01 00	0_x 00 01
9000 9001 42 43	9000 9001 43 42	0_x 42 43
11000 11001 01 00	11000 11001 00 01	0_x 01 00

Processors that use

1. little endian
   - Intel x86
   - VAX
   - alpha
2. Big endian
   - IBM370
   - IBM 390
   - Motorola 680 x0
   - sun sparc

```
struct{
   int      a;     //0x1112_1314                    word
   int      pad;   //
   double   b;     //0x2122_2324_2526_2728          doubleword
   char*    c;     //0x3132_3334                    word
   char     d[7];  //'A','B','C','D','E','F','G'    byte array
   short    e;     //0x5152                         halfword
   int      f;     //0x6162_6364                    word
} s;
```

Big-endian address mapping

Byte address								
	11	12	13	14				
00	00	01	02	03	04	05	06	07
	21	22	23	24	25	26	27	28
08	08	09	0A	0B	0C	0D	0E	0F
	31	32	33	34	'A'	'B'	'C'	'D'
10	10	11	12	13	14	15	16	17
	'E'	'F'	'G'		51	52		
18	18	19	1A	1B	1C	1D	1E	1F
	61	62	63	64				
20	20	21	22	23				

Little-endian address mapping

								Byte address
					11	12	13	14	
07	06	05	04	03	02	01	00	00
21	22	23	24	25	26	27	28	
0F	0E	0D	0C	0B	0A	09	08	08
'D'	'C'	'B'	'A'	31	32	33	34	
17	16	15	14	13	12	11	10	10
		51	52		'G'	'F'	'E'	
1F	1E	1D	1C	1B	1A	19	18	18
				61	62	63	64	
				23	22	21	20	20

Figure 12.13 Example C Data Structure and Its Endian Maps

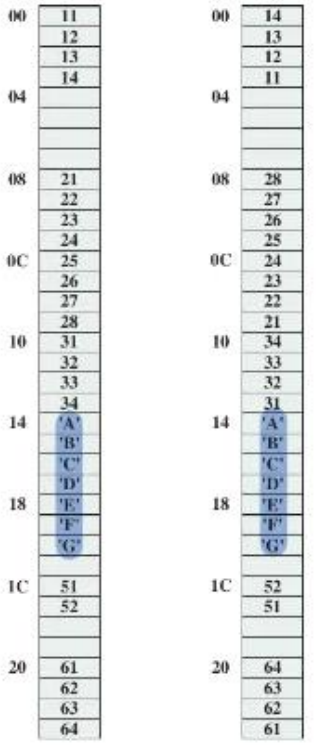
lower left results from compilation of that structure for a big-endian machine, and that in the lower right for a little-endian machine. In each case, memory is depicted as a series of 64-bit rows. For the big-endian case, memory typically is viewed left to right, top to bottom, whereas for the little-endian case, memory typically is viewed as right to left, top to bottom. Note that these layouts are arbitrary. Either scheme could use either left to right or right to left within a row; this is a matter of depiction, not memory assignment. In fact, in looking at programmer manuals for a variety of machines, a bewildering collection of depictions is to be found, even within the same manual.

- Endianness does not affect the ordering of data items within a structure. Thus, the four-character word c exhibits byte reversal, but the seven-character byte array d does not. Hence, the address of each individual element of d is the same in both structures.

The effect of endianness is perhaps more clearly demonstrated when we view memory as a vertical array of bytes, as shown in Figure 12.14.

There is no general consensus as to which is the superior style of endianness.[4] The following points favor the big-endian style:

- **Character-string sorting:** A big-endian processor is faster in comparing integer-aligned character strings; the integer ALU can compare multiple bytes in parallel.

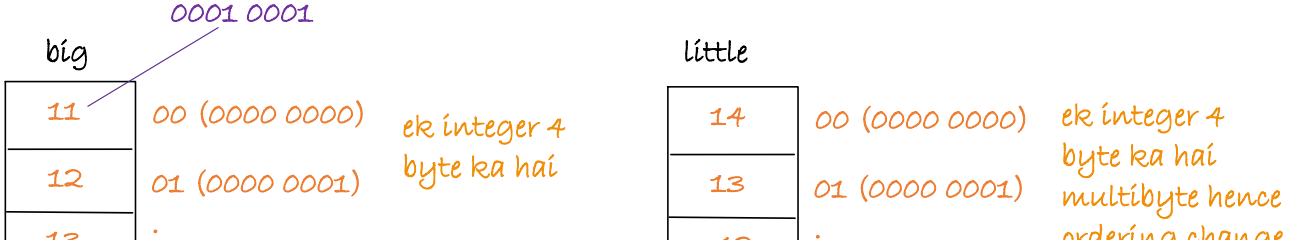


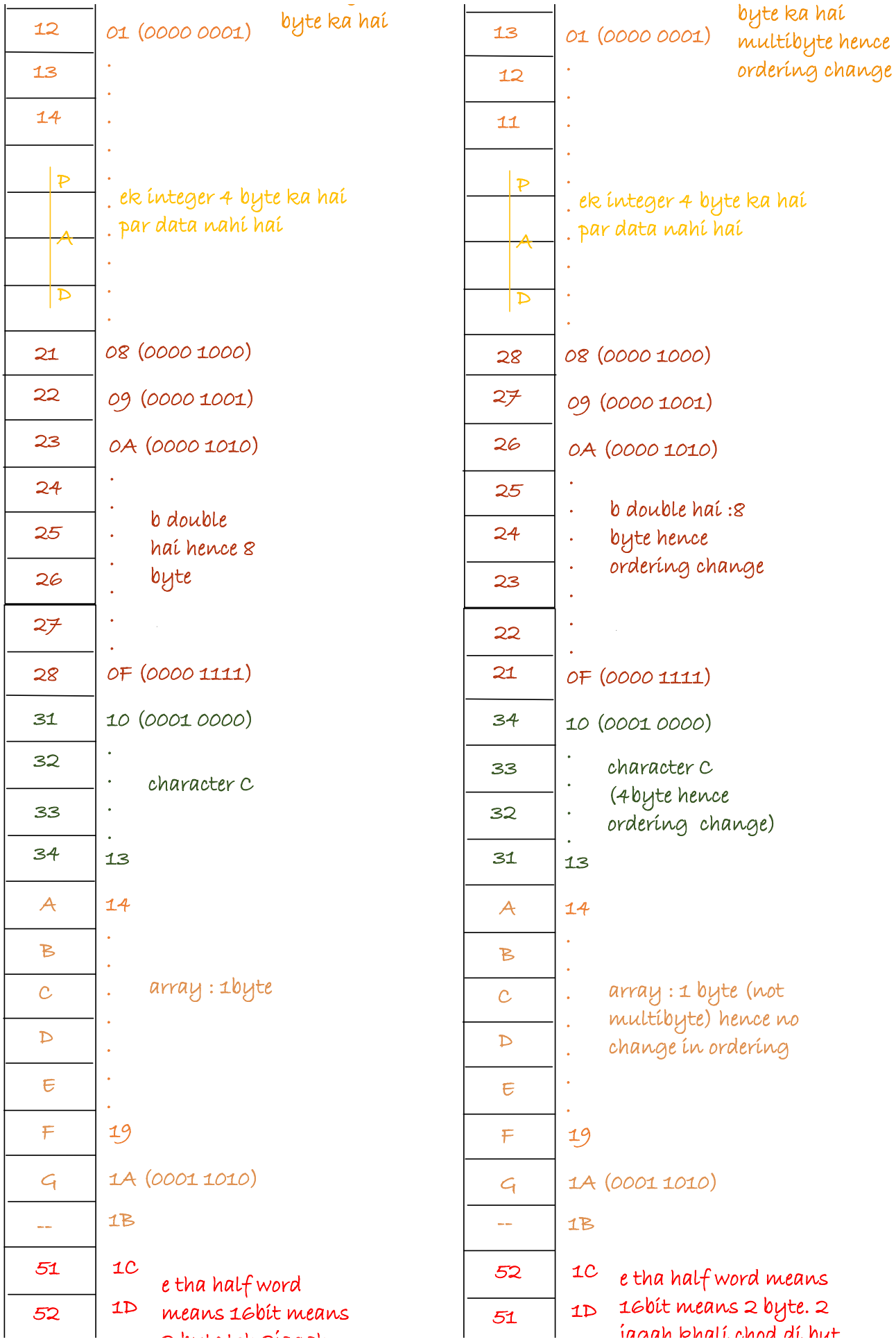
(a) Big endian (b) Little endian

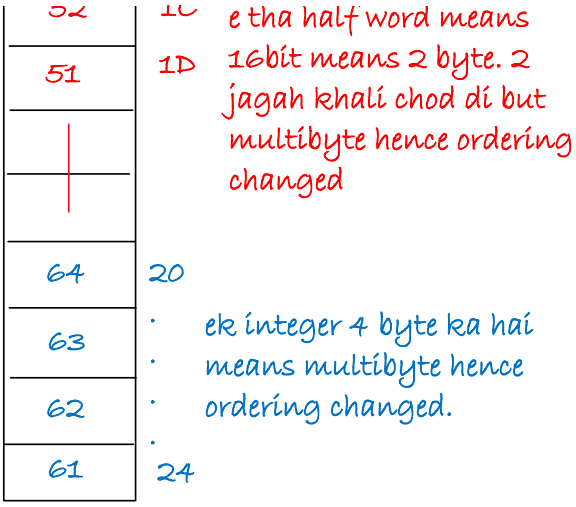
Figure 12.14 Another View of Figure 12.13

[4]The prophet revered by both groups in the Endian Wars of *Gulliver's Travels* had this to say. "All true Believers shall break their Eggs at the convenient End." Not much help!

```
struct{
   int      a;     //0x1112_1314                    word
   int      pad;   //
   double   b;     //0x2122_2324_2526_2728          doubleword
   char*    c;     //0x3132_3334                    word
   char     d[7];  //'A','B','C','D','E','F','G'    byte array
   short    e;     //0x5152                         halfword
   int      f;     //0x6162_6364                    word
} s;
```

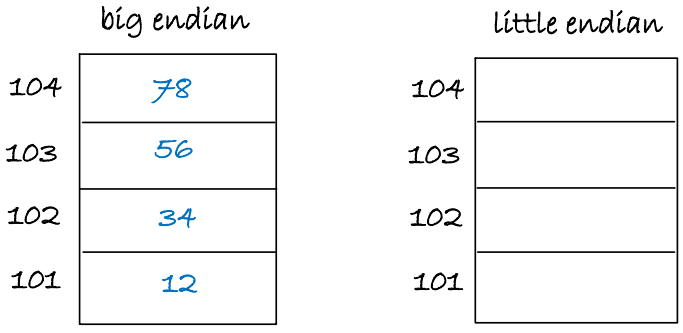



12
01 (0000 0001)
byte ka hai
13
14
P
A
D
ek integer 4 byte ka hai
par data nahi hai
21
08 (0000 1000)
22
09 (0000 1001)
23
0A (0000 1010)
24
25
26
b double
hai hence 8
byte
27
28
0F (0000 1111)
31
10 (0001 0000)
32
character C
33
34
13
A
14
B
C
array : 1byte
D
E
F
19
G
1A (0001 1010)
--
1B
51
1C
52
1D
e tha half word
means 16bit means
13
01 (0000 0001)
byte ka hai
multibyte hence
ordering change
12
11
P
A
D
ek integer 4 byte ka hai
par data nahi hai
28
08 (0000 1000)
27
09 (0000 1001)
26
0A (0000 1010)
25
24
23
b double hai :8
byte hence
ordering change
22
21
0F (0000 1111)
34
10 (0001 0000)
33
32
31
character C
(4byte hence
ordering change)
13
A
14
B
C
D
E
F
19
G
1A (0001 1010)
--
1B
52
1C
51
1D
e tha half word means
16bit means 2 byte. 2



Q. an array of 2 two byte integers is stored in big endian machine in byte address as shown below what will be its storage pattern in little endian machine?

Address	Data
0 x 104	78
0 x 103	56
0 x 102	34
0 x 101	12



it is a 2-byte integer arrays
visualization :

A	B
34	78
12	56

it is multibyte but they are in 2 byte pair so the ordering will occur in A and B individually

A	B
12	56
34	78

little endian

104	56
103	78
102	12
101	34

Topic - Pins : system contain hardware pin to perform the operation

(i) active low pin
(ii) active high pin
(iii) time multiplex pin

(i) active low pin : this pin is enabled when input is 0 or in the low state

denoted as : pinname

ex. RD, WR ke upar bar

(ii) active high pin : this pin is enabled when input is 1 or in the high state
ex. INTR, HLDA, ALE etc.

(iii) time multiplex pin : this pin carries the multiple meaning but one only at a time
- address pins are time multiplexed with data pins to carry the address and data.

but how to know if it is carrying the address or data?
- with the help of ALE

ALE (address latch enable)
- (1) then time multiplexer pins carries the address
- (0) then time multiplexer pins carries the data

advantage : number of hardware pins will be reduced.

1. Address line : address line are used to carry the address (CPU to memory) it is unidirectional.

note :

based on address line we can determine the capacity of memory [number of Main memory location/cells]

ex. 8085 processor [40pin IC]
AD_0-AD_7, A_8-A_{15}
16bit address
2^{16} cells/location = 64kb cells

Bonus : 16 pins ka kaam 8 mai ho gaya

AD_0-AD_7 , A_8-A_{15}
A : address : 16bits
D: data : 8 bits and 8 bits

ex. 8086processor
AD_0-AD_{15}, A_{16}-A_{19}
20bit processor
2^{20} cells/location = 1M cells

Bonus : 32 pins ka kaam 16 mai ho gaya

AD_0-AD_{15}, A_{16}-A_{19}
A : address : 20bits
D: data : 16 bits

2. Data line : data lines are used to carry the data (CPU to memory) it is bi-directional

note :

- based on the data line we can determine the word length of the processor or CPU
- the performance of the processor is measured by word length of CPU.

ex. 8085processor
AD_0-AD_7 , A_8-A_{15}
word length = 8bit
operation performed on 8bit data format

AD_0-AD_7 , A_8-A_{15}
A : address : 16bits
D: data : 8 bits and 8 bits

ex. 8086processor
AD_0-AD_{15}, A_{16}-A_{19}
word length = 16bit
operation perfomed on 16bit data format

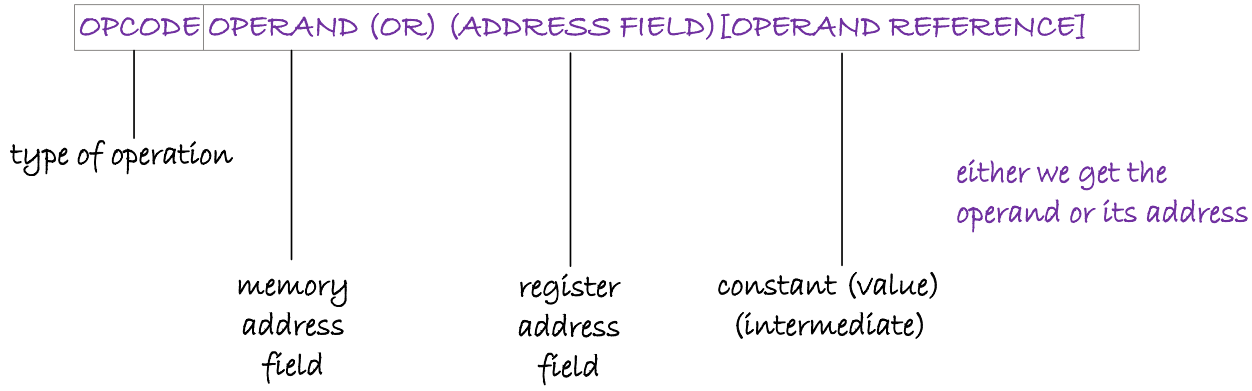
AD_0-AD_{15}, A_{16}-A_{19}
A : address : 20bits
D: data : 16 bits

machine instruction and addressing modes

topic - Instruction : instruction is a binary code (binary sequence) which is designed inside the processor to perform some operations.

binary sequence - bind with - operations

instruction made up of two things



1. OPCODE : operation / operational code tells us about type of operation

- if opcode is 2bit then $2^2 = 4$ operation perfomed

assume 00 ADD [Addition]
01 SUB [subtract]
02 MUL [multiply]
03 LOAD [load data from memory]

- if opcode is 3bit then $2^3 = 8$ operation perfomed

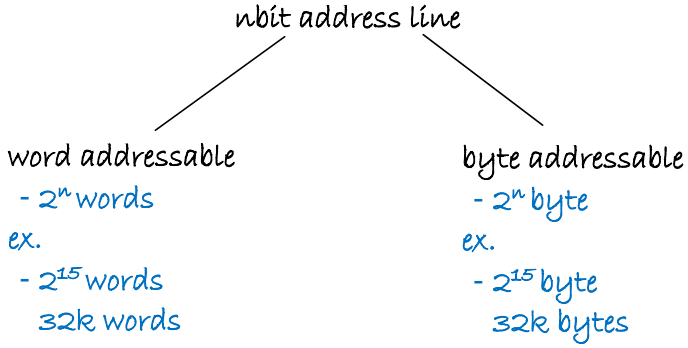
assume 000 OR
001 AND
010 NAND
011 XOR
100 ADD
101 LOAD
110 STORE
111 MUL

note :

nbit opcode performs 2^n operations

2. OPERAND : data
(OR)
OPERAND : operand reference (address field) address mil jata hai operand ka

Address field : n bit address field can specify 2^n memory cells (loactions)
1kb memory then address : $[\log_2 1kb] = [\log 2^{10}] = 10bit$



2^{10} =1k (kilo)
2^{20} =1m (mega)
2^{30} =1G (giga)
2^{40} =1T (tera)
2^{50} =1P (peta)
2^{60} =1E (exa)
2^{70} =1Z (zeeta)
2^{80} =1y (yotta)
1 byte = 8 bit
1 Nibble = 4 bit

1. if OPCODE field is given then number of operations performed?

OPCODE field	operations performed
1bit	= 2^1 = 2
2bit	= 2^2 = 4
3bit	= 2^3 = 8
4bit	= 2^4 = 16
5bit	= 2^5 = 32
6bit	= 2^6 = 64
7bit	= 2^7 = 128
8bit	= 2^8 = 256
9bit	= 2^9 = 512
10bit	= 2^{10} = 1024

2. if OPCODE is 6 bit then number of operations performed?

6bit = 2^6 = 64

2. if OPCODE is 6 bit then number of operations performed?

6bit $= 2^6 = 64$

3. if 100 operations performed then number of bits in OPCODE field?

n = 7bits
because , $2^6 = 64$
$2^7 = 128$
hence, 7bits.

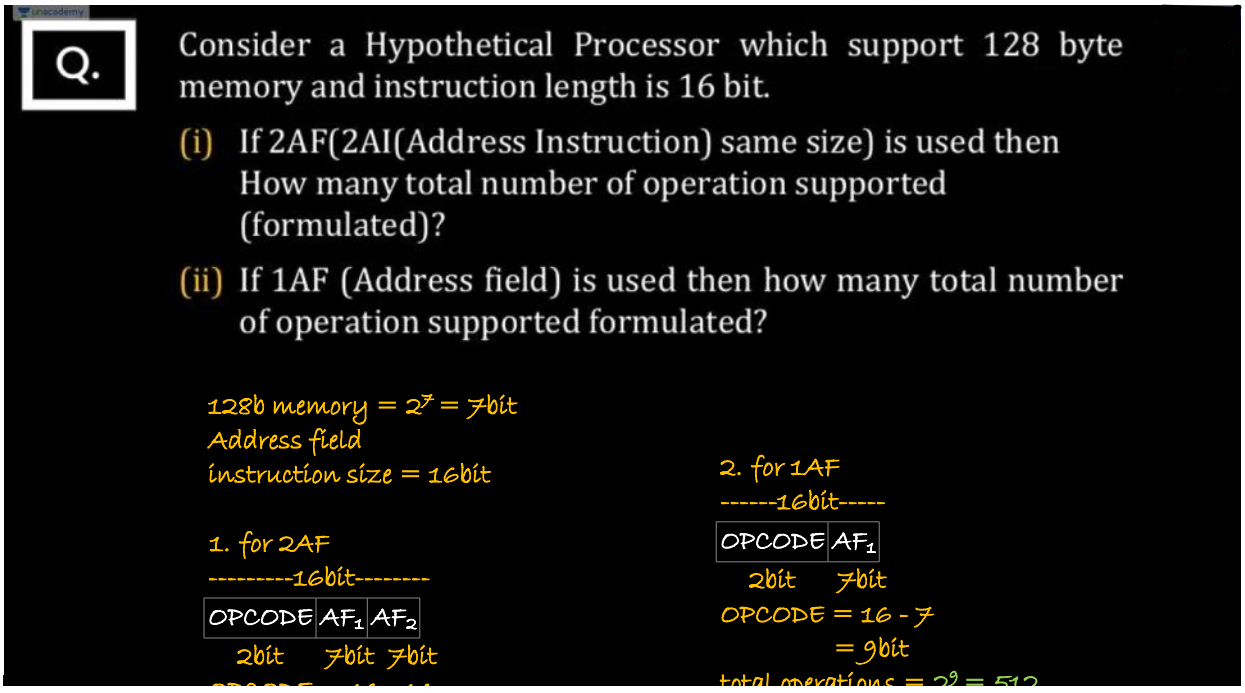
4. if memory is 256k byte then address field?

256KB = $2^8 . 2^{10}$ Byte = 2^{18}
Address field : 18bit

5. if processor has 30 registers then number of bits required to represent register

30 = $2^n = 2^5$
register address field = 5bits

total number of operation / instruction	number of bits in OPCODE field
100	7bit
55	6bit
210	8bit
30	5bit
50	6bit
90	7bit
25 register	5bit
50 register	6bit
110 operation	7bit
13bit address field	2^{13} = 8kb/8kword
8bit OPCODE	2^{8} = 256 operation
200 operation	OPCODE : 8bit
8 operations / instructions	OPCODE : 3bit



OPCODE	AF_1	AF_2
2bit	7bit	7bit

OPCODE = 16 - 14
= 2bit
total operations - 2^2 = 4 operations

OPCODE = 16 - 7
= 9bit
total operations = 2^9 = 512 operations

Q. A Hypothetical Processor support 100 different operation and 3 address memory field (same size). Instruction is stored in 1 MB memory. Then what is the length of the instruction?

100 operation = OPCODE = 7bit
memory = 1mb = 2^{20} byte = 20bits address field

OPCODE	AF_1	AF_2	AF_3
7bit	20	20	20

instruction length = 7 + 20 + 20 + 20
=67

Q. Consider a Hypothetical CPU which supports 110 instruction, 50 registers and 512KB memory space. Instruction contain 2 register operands, Memory operands and 13 bit Immediate constant fields. Program contain 300 instruction. Memory storage space required in Bytes to store the program is _____.

110 operationinstruction = OPCODE = 7bit
50 register = register address field = 6bit
memory = 512kb = 2^{19} byte = 19bits address field
immediate field = 13bit

OPCODE	reg AF	ref AF	memory AF	immediate field
7bit	6bit	6bit	19bit	12bit

instruction length = 7 + 6 + 6 + 19 + 13
=51bits = 51/8 = 6.375 (7 : ceiling) bytes
divided by 8 for bytes.
1 instruction size : 7 byte
program contains 300 instruction
program size = 300 x 7 byte = 2100 byte

Q. A processor has 40 distinct instructions and 24 general purpose registers. A 32-bit instruction word has an opcode, two register operands and an immediate operand. The number of bits available for the immediate operand field is 16bits.

Q. A processor has 40 distinct instructions and 24 general purpose registers. A 32-bit instruction word has an opcode, two register operands and an immediate operand. The number of bits available for the immediate operand field is 16bits.

[GATE-2016 (Set-2)]

40 operationinstruction = OPCODE = 6bit
24 register = register address field = 5bit

OPCODE	reg AF	ref AF	immediate field
6bit	5bit	5bit	

immediate = 32 - (6+5+5)
=16bits

Topic : Instruction format/Instruction set architecture

instruction made up of two things

OPCODE	OPERAND (OR) (ADDRESS FIELD)	[OPERAND REFERENCE]

- OPERAND (OR) (ADDRESS FIELD) → memory address field
- OPERAND (OR) (ADDRESS FIELD) → register address field (no of registers register file size)
- [OPERAND REFERENCE] → constant (value) (intermediate)

example :

4bits	6bits	6bits
OPCODE	OPERAND REFERENCE	OPERAND REFERENCE

16bits

total operation : 2^4 = 16 operations/instructions

the instruction set architecture is the part of the processor that is visible to the programmer or compiler writer. the ISA serves as a boundary between software and hardware. software is

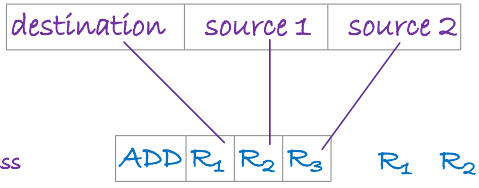
converted to machine instructions using software (compiler). then the instructions are executed using hardware.

An ISA contains :
- the functional definition of storage locations (registers, memory) and operations (add, multiply,branch,load,store,etc)
- precise description of how to invoke and access them

AN ISA does not contains :
- how operations are implemented
- which operations are fast and which are slow
- which operations take more power and which take less.

ADD: | destination | source 1 | source 2 |

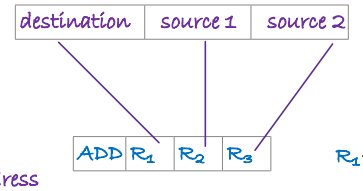
1. Number of explicit operand : how many operands are available



3 AI / 3 AF	OPCODE AF_1 AF_2 AF_3	OPCODE ke alawa 3 address	ADD R_1 R_2 R_3	R_1 $R_2 + R_3$
2 AI / 2 AF	OPCODE AF_1 AF_2	OPCODE ke alawa 2 address	ADD R_1 R_2	R_1 $R_1 + R_2$
1 AI / 1 AF	OPCODE AF_1	OPCODE ke alawa 1 address	ADD R_1	AC AC + R_1
0 A1 / 0 AF	OPCODE	OPCODE ke alawa no address	ADD	ADD

2. Location of the operands : data mera kaha hoga? registers, accumulator or memory?

based on the availability of ALU operand

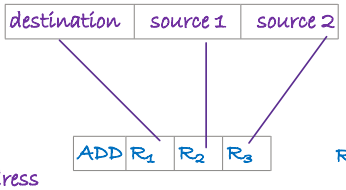


general register organisation	3 AI / 3 AF	OPCODE AF_1 AF_2 AF_3	OPCODE ke alawa 3 address	ADD R_1 R_2 R_3	$R_1 \leftarrow R_2 + R_3$
general register	2 AI / 2 AF	OPCODE AF_1 AF_2	OPCODE ke alawa 2 address	ADD R_1 R_2	$R_1 \leftarrow R_1 + R_2$

general register organisation	2 AI / 2 AF	OPCODE AF_1 AF_2	OPCODE ke alawa 2 address	ADD R_1 R_2	$R_1 \leftarrow R_1 + R_2$
accumalator based organisation	1 AI / 1 AF	OPCODE	OPCODE ke alawa 1 address	ADD R_1	$AC \leftarrow AC + R_1$
stack based	0 A1 / 0 AF	OPCODE	OPCODE ke alawa no address	ADD	ADD

3. specification of the operands : specific location operand ki

based on the availability of ALU operand



general register organisation	3 AI / 3 AF	OPCODE AF_1 AF_2 AF_3	OPCODE ke alawa 3 address	ADD R_1 R_2 R_3	$R_1 \leftarrow R_2 + R_3$
general register organisation	2 AI / 2 AF	OPCODE AF_1 AF_2	OPCODE ke alawa 2 address	ADD R_1 R_2	$R_1 \leftarrow R_1 + R_2$
accumalator based organisation	1 AI / 1 AF	OPCODE	OPCODE ke alawa 1 address	ADD R_1	$AC \leftarrow AC + R_1$
stack based	0 A1 / 0 AF	OPCODE	OPCODE ke alawa no address	ADD	ADD

4. sizes of operand supported : sizes the operand support

- byte (8-bits)
- half-word (16bits)
- word (32-bits)
- double (64-bits)

5. supported operations :

- ADD
- SUB
- MUL
- AND
- OR
- CMP

- MOVE

(i) stack based organisation : in the stack based organisation ALU operations are perfomed on stack data. ALU operand stack mai milega aur result bhi.

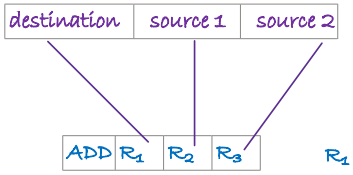
so both the operand (data) must be required (present) in the stack and after the processing result is also stored in the stack.

in stack CPU "0" (zero) explicit operand (or) all operand are implicit (alag se address pass karne ke zarurt nahi hai, dono operand stack mai hi mil jayenge) that means ALU operand (data) before the ALU operation must be present in stack. (sidhe ADD likh dete hai)

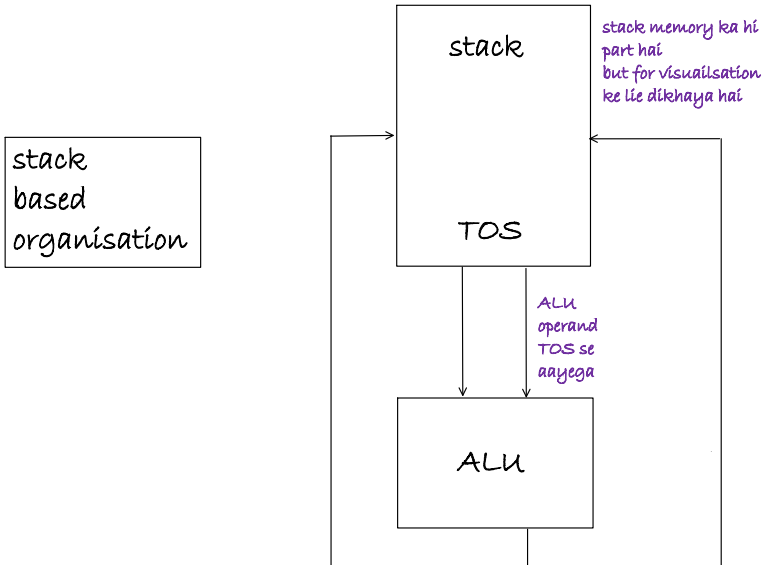
- what is stack?
stack is a block (part) of a memory in RAM but to control, CPU keeps a stack pointer register.

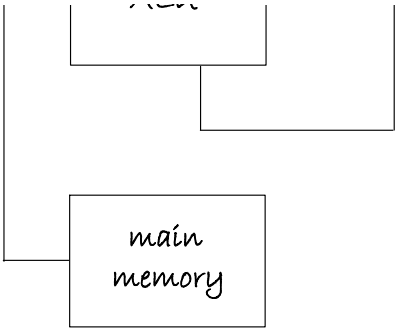
- what stack pointer (SP) register does?
stack pointer register points to the top of the stack (TOS) address.

- what is top of the stack?
in the stack, insert and delete operation are perfomed at the one end (same end) called top of the stack and this is default address in stack pointer register.

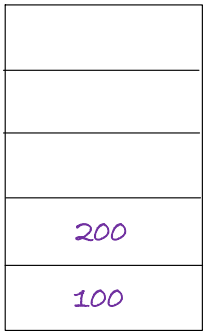


teeno data stack mai vo bhi top of stack me





Push 100
Push 200
Add

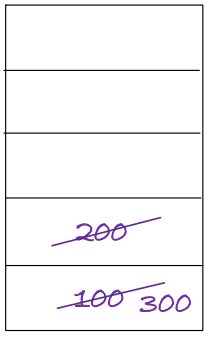


stack memory

TOS (POP) + TOS (POP) = 300

200 100

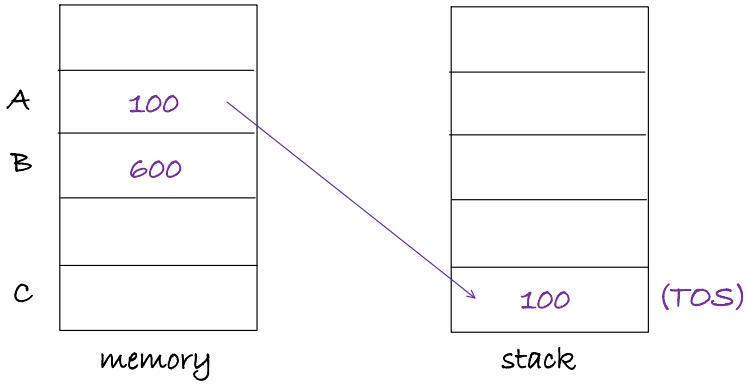
pop karlenge stack se



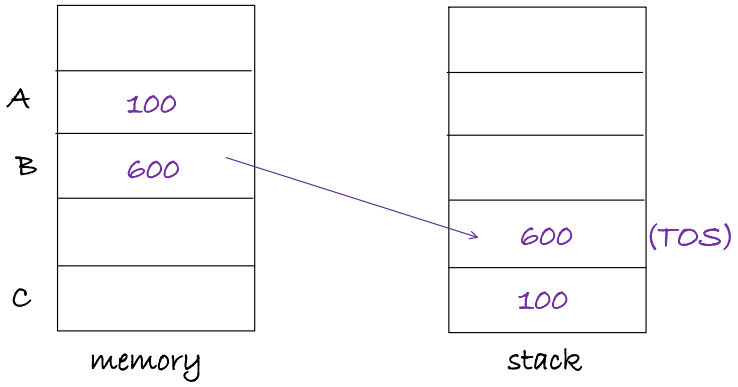
stack memory

In depth explanation :

I_1 : PUSH A : push whatever value present at location A to the top of the stack (which present in location A)
TOS ← M[A] (TOS mai memory loaction A ka data daal dena)

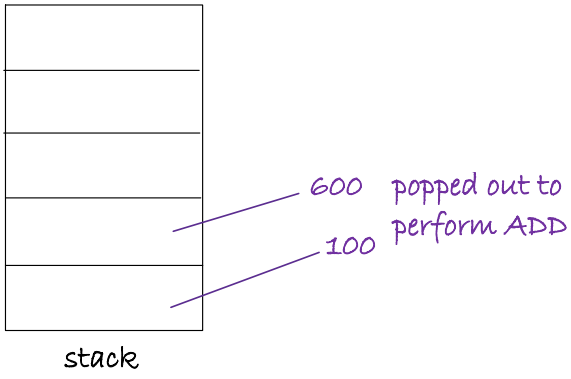


I_2: PUSH B : push whatever value present at location B to the top of the stack (which is present in location B)
TOS ← M[B] (TOS mai memory loaction B ka data daal dena)



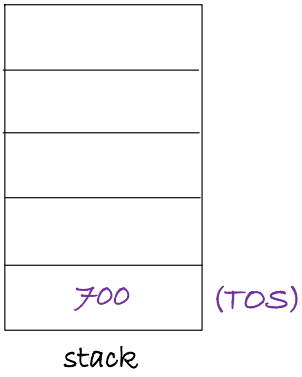
I_3: ADD : add the top two (2) elements of the stack (pop 2 TOP element) from the stack and perform addition and save the result back to the TOP of the stack.

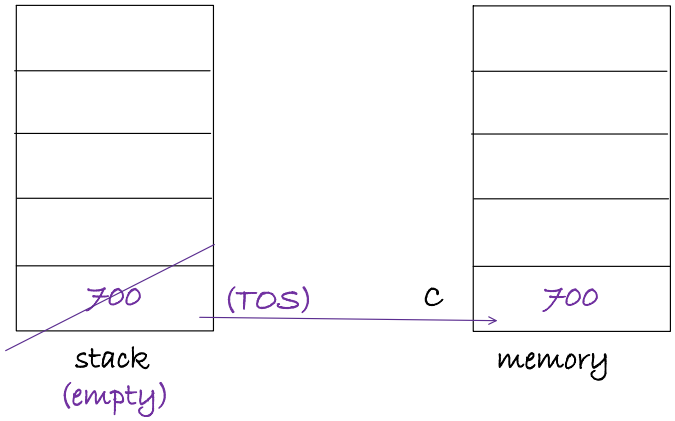
ADD : 100 + 600 = 700 (TOS)



I_4: POP C : store the value(result) (which is present in TOS) at the top of the stack to memory loaction C

M[C] ←TOS





note:

- in stack CPU ALU operation are 0 address field
- In the STACK-CPU data transfer operation (PUSH, POP) are not 0 address field, its 1 address field.

but we write directly!

- PUSH A :

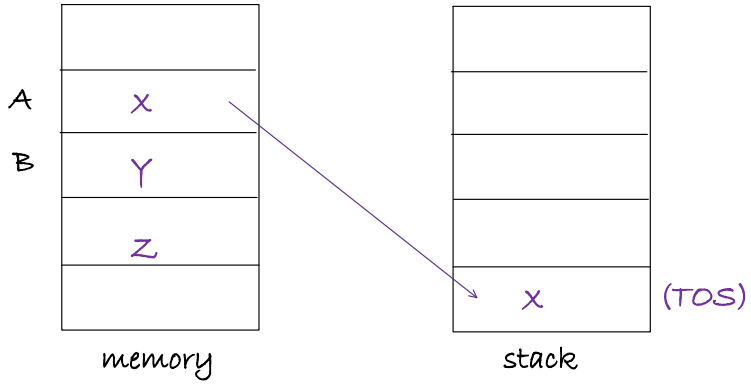
- PUSH B :

- ADD :

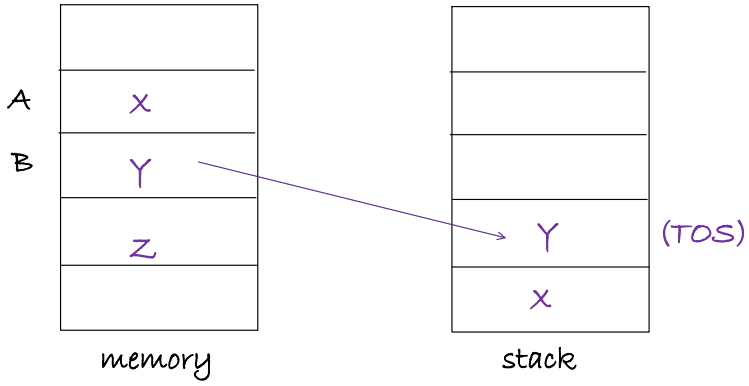
- POP C :

Q. (X * Y) + Z : how many machine instruction required to execute using stack based organisation?

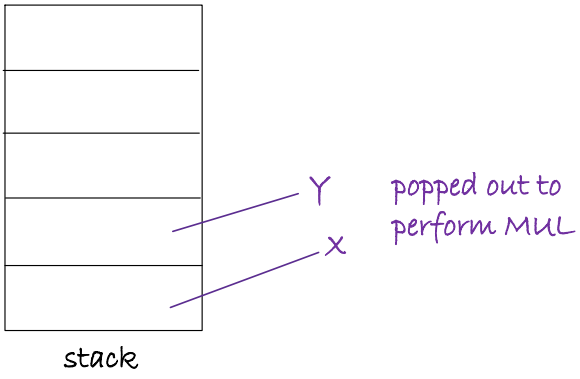
sol. PUSH X : X pushed into TOS



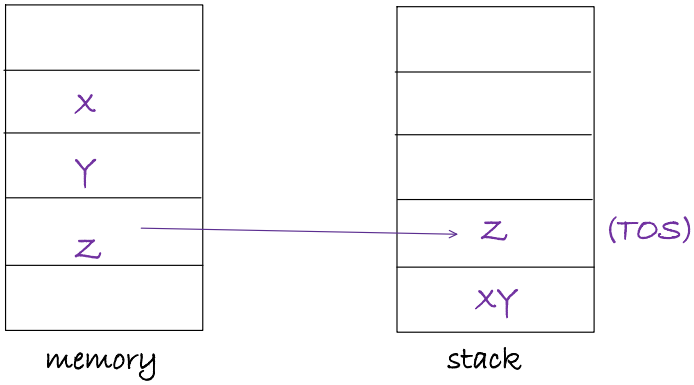
PUSH Y : Y pushed into TOS



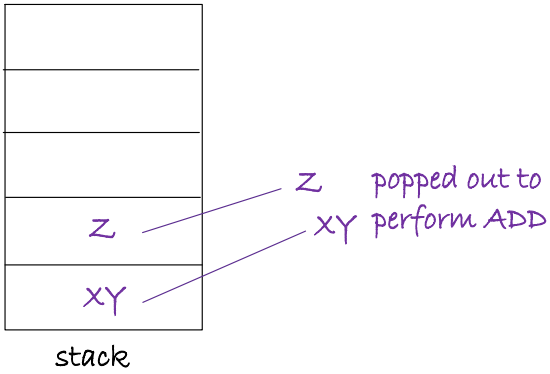
MUL : X and Y Popped out and performed the multiply operation



PUSH Z : Z pushed into TOS



ADD : XY and Z Popped out and perfomed Addition



Q. Consider a 32 bit Hypothetical Processor which use STACK-CPU. Which support 1 Word opcode and 24 bit address following statement is executed on a STACK-CPU (Stack is Initially Empty)

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

Q. How many Machine Instruction are required using Stack-CPU?

I_1: PUSH A

I_2: PUSH B

I_3: ADD

I_4: PUSH C

I_5: PUSH D

I_6: ADD

I_7: MUL

I_8: POP X

Q.3 What is the status of the Stack at the end of program execution?

EMPTY
because we popped the result from stack

Q. How much Memory space is required for the program in Byte?

address = 24 bit = 3byte (8+8+8)
OPCODE = 1word = 4byte (32bit)

I_1: PUSH A : 4B + 3B = 7B

I_2: PUSH B : 4B + 3B = 7B

I_3: ADD : 4B = 4B

I_4: PUSH C : 4B + 3B = 7B

I_5: PUSH D : 4B + 3B = 7B

I_6: ADD : 4B = 4B

I_7: MUL : 4B = 4B

I_8: POP X : 4B + 3B = 7B

operation or instruction

address

only operation

I_7: MUL : 4B = 4B

I_8: POP X : 4B + 3B = 7B

Q. Consider a processor which contain the following pin structure

$AD_0 - AD_{23}$, $A_{24} - A_{39}$

Processor contain 250 register instruction is designed with 4 fields i.e OPCODE, register address, memory address and 16 bit immediate field.

Processor support 180 instruction (operation). A program contain 400 instruction then how much space is required for the program

(i) In Byte?

(ii) Words?

OPCODE	REF AF	MEM AF	IMMEDIATE FIELD
8bit	8bit	40bit	16bit

250 register = reg address field = 8bit

180 operation hence OPCODE = $2^n = 2^8$ = 8bit

memory address field = 40bit
(word size) data = 24

hence, length of instruction = 8+8+40+16 = 72bits = 9byte.
one instruction size = 9byte
program contains 400 instruction : program size
in bytes = 400 x 9 = 3,600 bytes
in word (3byte) = 3600/3 = 1200 words

Q. A machine has a 32-bit architecture, with 1-word long instructions. It has 64 registers, each of which is 32 bits long. It needs to support 45 instruction, which have an immediate operand in addition to two register operands. Assuming that the immediate operand is an unsigned integer the maximum value of the immediate operand is _____. **[GATE-2014 (Set-1)]**

----------32bit----------

OPCODE	REG1	REG2	IMMEDIATE
6bit	6bit	6bit	14bit

1 word long instruction hence instruction size = 32bits
it have 64 registers = reg AF = 6bits
45 instructions / operations = OPCODE = 6bit

immediate field = 32 - (6+6+6)
= 14bits

immediate unsigned = 14bits
unsigned range = 0 to $2^n - 1$
= 0 to $2^{14} - 1$

immediate unsigned = 14 bits
unsigned range = 0 to $2^n - 1$
= 0 to $2^{14} - 1$
= 0 to 16383

Instruction format

OPCODE	OPERAND (OR) OPERAND REFERENCE

single accumulator ogranisation

- in the accumulator based organisation first ALU operand are always present in the accumulator register and second ALU operand present in the memory.
- after the processing result is also stored in the accumulator (OR) accumulator is used as destination to store the result of ALU operation.
- accumulator is resgister in the CPU which is associated with the ALU.
- the operand in the accumulator is loaded from the memory using the LOAD command or instruction and the result is stored in the memory from the accumulator using the STORE command.

note :
here 1 operand is implicit (present in accumlator) and 1 operand is explicit (present in the memory address)

OPCODE	OPERAND (OR) OPERAND REFERENCE

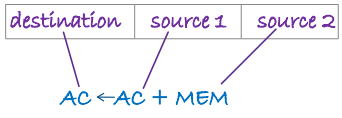
1 AI / 1 AF

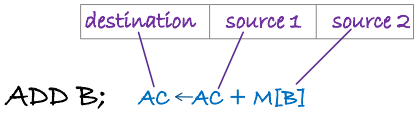
OPCODE	ADDRESS FIELD

ADD	R_1

type of operation

(i) ALU operation :

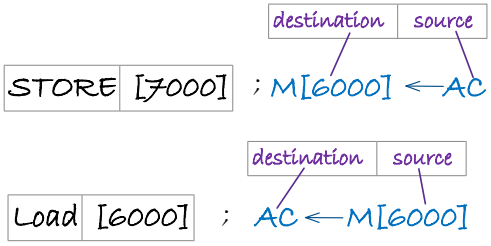


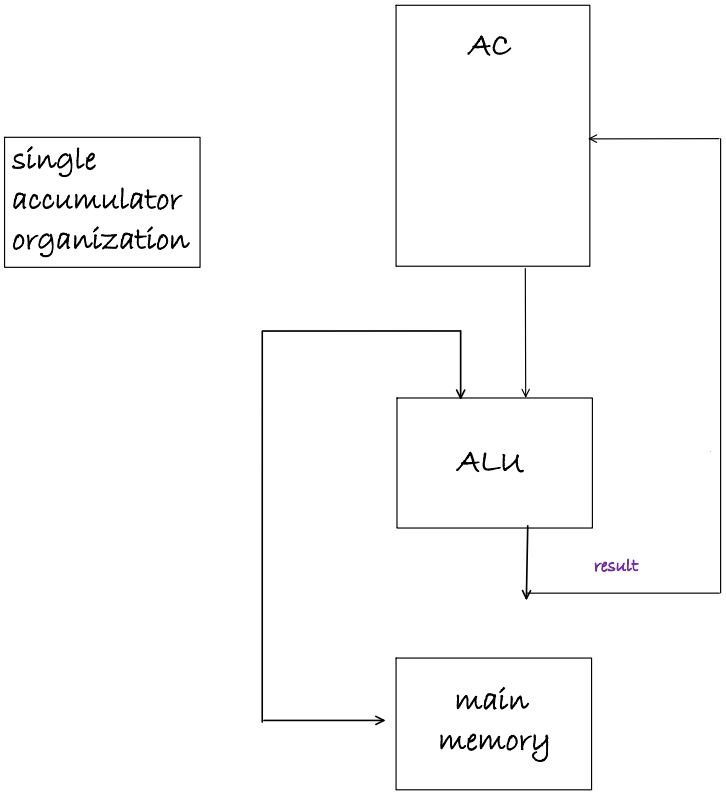


(ii) data transfer : Memory AC [STORE]

accumulator mem [LOAD]

(ii) data transfer : Memory AC [STORE]
accumulator mem [LOAD]





Accumulator based org.

ex.1 A+B using AC. CPU

I_1 : LOAD A; AC ← M[A]

I_2 : ADD B; AC ← AC + M[B]

2 instruction using accumulator CPU

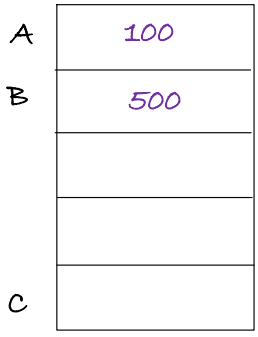
ex. C=A+B using AC. CPU WHERE A, B, C are the memory address

I_1: LOAD A; AC ← M[A]; AC ← 100

I_2: ADD B; AC ← AC + M[B]; AC ← 500

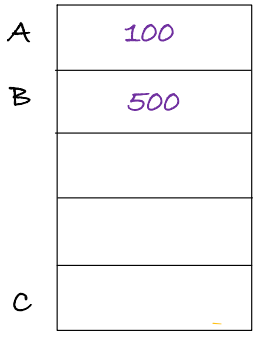
I_3: STORE C; M[C] ← AC; M[C] ← 600

I_1: LOAD A; AC ← M[A]; AC ← 100
100



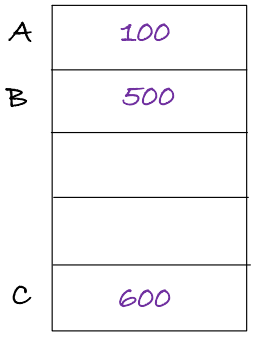
load the content (data) of memory location/address (A) to the accumulator

I_2: ADD B; AC ← AC + M[B]; AC ← 500
100 500



fetch the content (data) from memory location (address) B and add that contain with accumulator (AC) data or value present in the accumulator and save the result back to the Accumulator

I_3: STORE C; M[C] ← AC; M[C] ← 600



store the value (data) from the accumulator (AC) to the memory location C.

Defintions

I_1 : LOAD X; AC ← M[A] ; load the content (data) of memory location/address (A) to the accumulator

I_2: ADD B; AC ← AC + M[B]; fetch the content (data) from memory location (address) B and add that contain with accumulator (AC) data or value present in the accumulator and save the result back to the Accumulator

I_3: STORE C; M[C] ← AC; store the value (data) from the accumulator (AC) to the memory location C.

ex.3 Z = (X * Y)

I_1 : LOAD X; AC ← M[X]

I_2 : MUL Y; AC ← AC + M[Y]

I_3 : STORE Z; M[Z] ← AC

3 instruction using accumulator CPU

ex.4 (X * Y) + Z ; how many machine instructions are required using AC-CPU

I_1 : LOAD X; AC ← M[X]

I_2 : MUL Y; AC ← AC * M[Y]

I_3 : ADD Z; AC ← AC + M[Z]

I_2 : MUL Y; AC ← AC *M[Y]

I_3 : ADD Z; AC ← AC+M[Z]

3 instruction using accumulator CPU

ex.4 (X * Y) + Z ; how many machine instructions are required using AC-CPU

I_1 : LOAD X; AC ← M[X]

I_2 : MUL Y; AC ← AC *M[Y]

I_3 : ADD Z; AC ← AC+M[Z]

I_4 : STORE A; M[A] ← AC

4 instruction using accumulator CPU

#memory spills : to store the intermediate result in the memory

ex.5 X = [(A+B) * (C+D)] ; A,B,C,D and X are the variable in the memory.

I_1 : LOAD A;

I_2 : ADD B; AC ← AC + M[B]

I_3 : STORE Temp; m(temp.) ← AC — memory spills : to store the intermediate result in the memory

save your operation before loading another variable
spills : 1

I_4 : LOAD C; AC ← M[C]

I_5 : ADD D; AC ← AC + M[D]

to save in temporary memory
we use temp.

I_5 : MUL Temp; AC ← AC + M[temp]

I_6 : STORE X; M[C] ← AC

save your operation
but spills : 0 because i didnt want to save but the execution is completed and the question asked to save.

7 instruction using accumulator CPU

memory spills : 1

stack based	accumulator based
alu operand : implicit (stack)	1 operand alu implicit (accumulator)

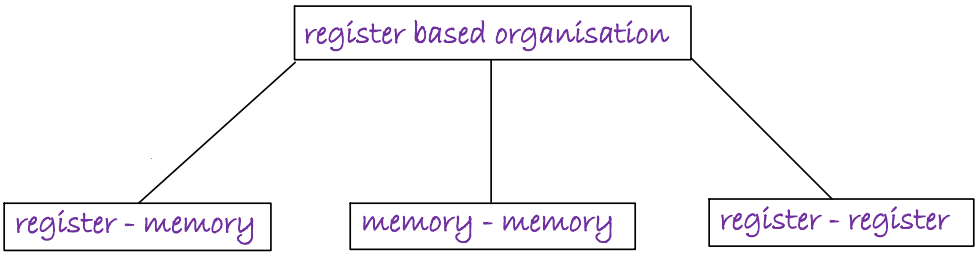
stack based

alu operand : implicit (stack)
alu operation : 0 AI
data transfer : 1 AI

accumulator based

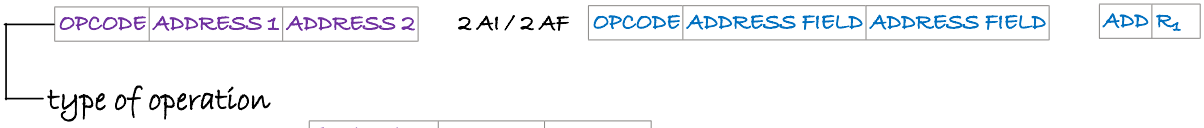
1 operand alu implicit (accumulator)
2nd operand memory and result stored in AC.

General register organisation

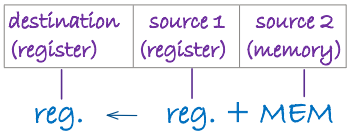


(i) register - memory : in this organisation first ALU operand present in the registers (general purpose register) and second ALU operand present in memory and after processing result stored in register.

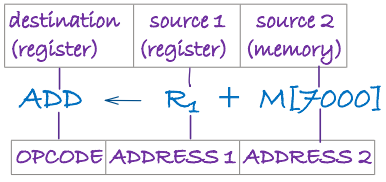
this architecture is different from the accumulator architecture in a way that it has multiple register used to store the operand and result.



(i) ALU operation :

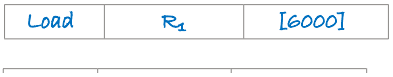


ADD R_1 7000;



(ii) data transfer : Register memory [LOAD]
memory register [STORE]

- Load R_1 [6000] ; $R_1 \leftarrow M[6000]$



OPCODE	ADDRESS 1	ADDRESS 2

- STORE R_1 [6000] ; M[6000] $\leftarrow R_2$

STORE	[9000]	R_2

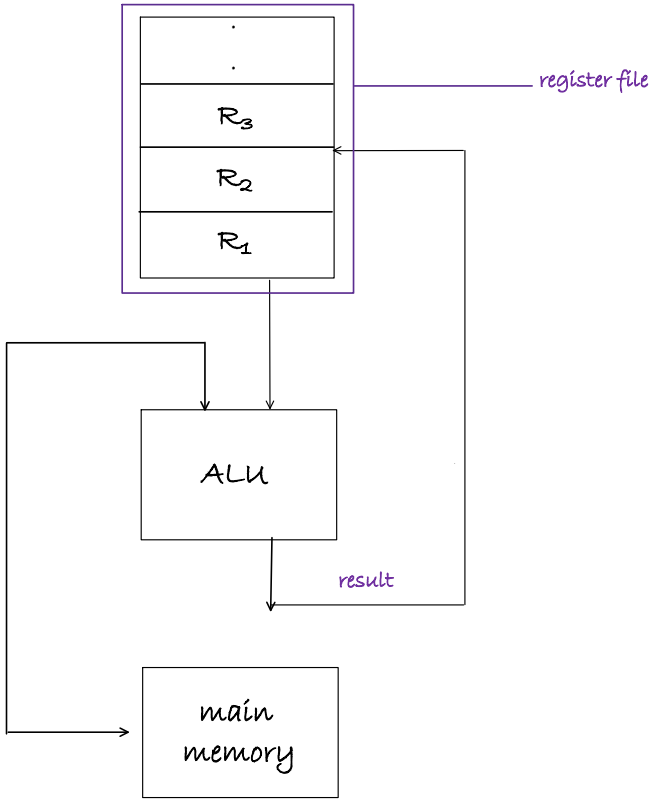
OPCODE	ADDRESS 1	ADDRESS 2

Ex. ADD $\leftarrow R_1$ [4000]

$R_1 \leftarrow R_1$ + M[4000] (R_1 mai R_1 + memory location 4000 ka data)

destination (register)	source 1 (register)	source 2 (memory)

#register file : the number of register supported by the processor



ex.1 Z = (X + Y) using register memory CPU

I_1 : LOAD R_1 X; $R_1 \leftarrow$ M[X]

I_2 : ADD R_1 Y; $R_1 \leftarrow R_1$ +M[Y]

I_3 : STORE Z R_1; M[Z] $\leftarrow R_1$

3 machine using register-memory CPU

ex.2 (A + B) using register memory CPU

I_1 : LOAD R_1 A; $R_1 \leftarrow M[A]$

I_2 : ADD R_1 B; $R_1 \leftarrow R_1 + M[B]$

2 machine using register-memory CPU

ex.3 (X * Y) + Z ; X, Y and Z are variable in the memory (memory address)

I_1 : LOAD R_1 X; $R_1 \leftarrow M[X]$ // I_1 : MOV R_1 X;

I_2 : MUL R_1 Y; $R_1 \leftarrow R_1 * M[Y]$

I_3 : ADD R_1 Z; $R_1 \leftarrow R_1 + M[Z]$

3 machine using register-memory CPU

(ii) memory - memory : all the operand must be required in the memory

(we dont study it)

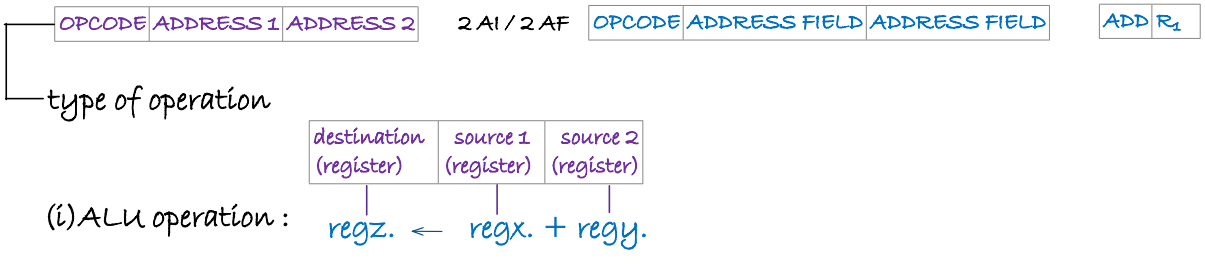
ADD A, B, C

m[A] ←m[B] + m[C]

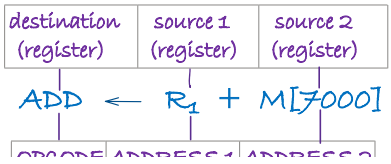
(iii) register - register : in this architecture all the ALU operand required/must be present in the register.

(RISC)fastest

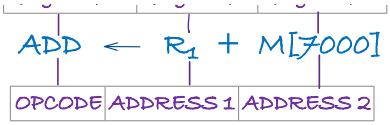
- this architecture supports more number of registers.



ADD R_1 7000;



ADD R_1 7000;



destination	source

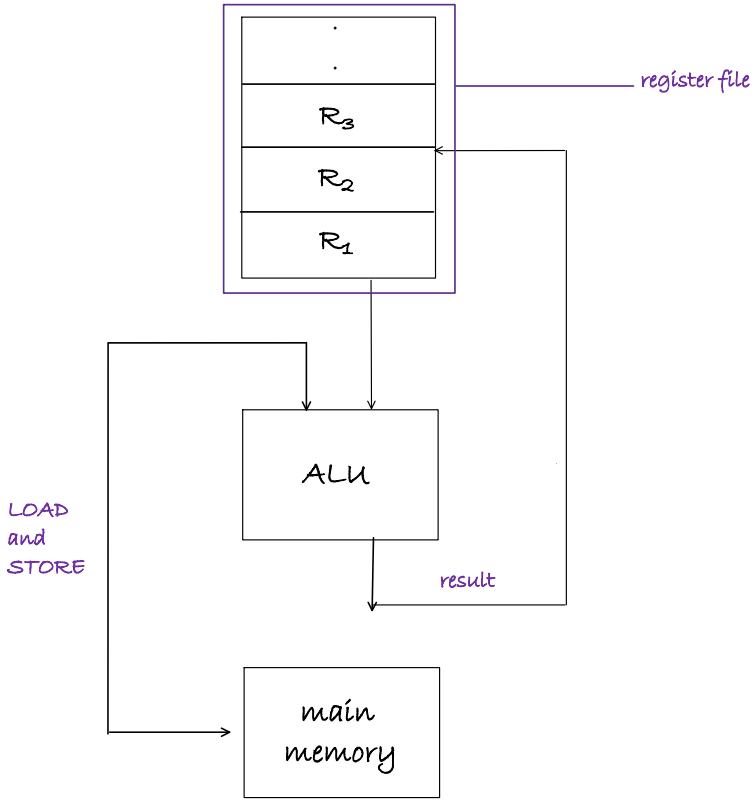
(ii) data transfer : Register memory [LOAD]
memory register [STORE]

- Load R_1 [6000] ; $R_1 \leftarrow M[6000]$

Load	R_1	[6000]

OPCODE	ADDRESS 1	ADDRESS 2

note :
arithmatic operations does not access the memory only load and store instructions are used to access the memory



ex.1 Z = (X + Y) using register-register CPU

I_1 : LOAD R_1 X; $R_1 \leftarrow M[X]$

I_2 : LOAD R_2 Y; $R_2 \leftarrow M[Y]$

I_2 : LOAD R_2 Y; $R_2 \leftarrow M[Y]$

I_3 : ADD R_3 R_1 R_2; $R_3 \leftarrow R_1 + R_2$

I_4 : STORE Z R_3 ; $M[Z] \leftarrow R_3$

4 machine using register-register CPU

ex.2 C=A+B using AC. CPU WHERE A, B, C are the memory address

I_1 : LOAD R_1 A

I_2 : LOAD R_2 R_3

I_3 : ADD R_3 R_1 R_2 ; $R_3 \leftarrow R_1 + R_2$

I_4 : STORE C R_3 ; $M[C] \leftarrow R_3$

ex.3 (X * Y) + Z ; register-register CPU

I_1 : LOAD R_1 X

I_2 : LOAD R_2 Y

I_3 : MUL R_3 R_1 R_2; $R_3 \leftarrow R_1 * R_2$

I_4 : LOAD R_4 Z

I_5 : ADD R_5 R_3 R_4; $R_5 \leftarrow R_3 + R_4$

5 machine using register-register CPU

ex.3 (A + B) * (C + D) ; register-register CPU

I_1 : LOAD R_1 A

I_2 : LOAD R_2 B

I_3 : ADD R_3 R_1 R_2;

I_4 : LOAD R_4 C

I_5 : LOAD R_5 D

I_6 : ADD R_6 R_4 R_5

I_7 : MUL R_7 R_3 R_6

I_8 : STORE X R_7

how many minimum no of registers are required?

I_1 : LOAD R_1 A

I_2 : LOAD R_2 B

I_3 : ADD R_1 R_2 R_3; $R_1 \leftarrow A + B$

I_4 : LOAD R_2 C

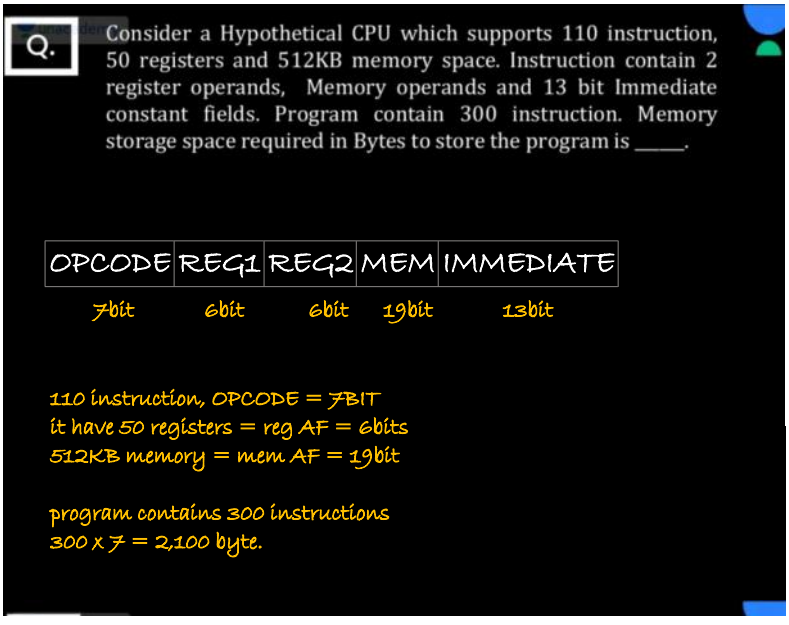
I_5 : LOAD R_3 D

I_6 : ADD R_2 R_2 R_3; $R_2 \leftarrow C + D$

I_7 : MUL R_3 R_1 R_2; $R_3 \leftarrow R_1 * R_2$

I_8 : STORE X R_3; $M[X] \leftarrow R_3$

3 Registers



300 x 7 = 2,100 byte.

Q. Consider a processor with 64 registers and an instruction set of size twelve. Each instruction has five distinct fields, namely, opcode, two source register identifiers, one destination register identifier, and a twelve-bit immediate value. Each instruction must be stored in memory in a byte-aligned fashion. If a program has 100 instructions, the amount of memory (in bytes) consumed by the program text is _____.

[GATE-2016 (Set-2): 2Marks]

OPCODE	REG1	REG2	REG2	IMMEDIATE
4bit	6bit	6bit	6bit	12bit

12 instruction, OPCODE = 4bit
it have 64 registers = reg AF = 6bits
instruction size = 4+6+6+6+12 = 34 bits = 5byte.

program contains 100 instructions
porgram size : 100 x 5 = 500 = 500byte.

2AF:

16bit (instruction)

OPCODE	AF_1	AF_2
2	7	7

2AF = total no of operations = 2^2 = 4 operation

1AF:

16bit (instruction)

OPCODE	AF_1
9	7

1AF = total no of operations = 2^9 = 512 operation

expand opcode technique

expand opcode length is required in the fixed length instruction supported CPU design to implement the various instruction with different formats (alag alag format ke operation support karne ke lie)

variable length instruction/
fixed length opcode

OPCODE = 8Bbit
AF = 8bit

16bit(instruction)

1AF | OPCODE (8) | AF (8) | = 16bit

0AF | OPCODE (8) | = 8bit

instruction ki length variable hai

fixed length
instruction/variable length
opcode

OPCODE = 8Bbit
AF = 8bit

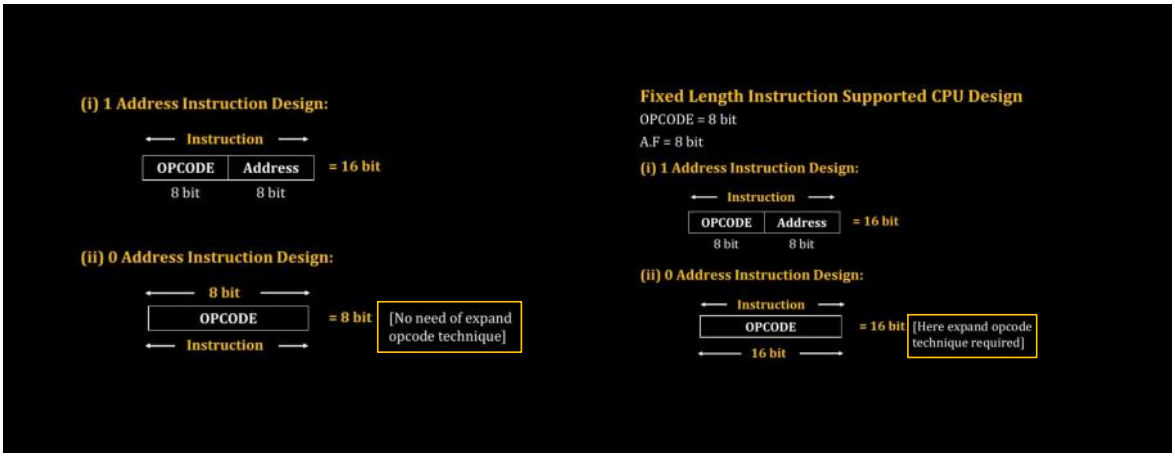
16bit(instruction)

1AF | OPCODE (8) | AF (8) | = 16bit

0AF | OPCODE (16) | = 16bit

instruction ki length fix hai toh puri 16bit opcode ko dedenge

variable length : OPCODE fixed
fixed length : variable size OPCODE



expand opcode technique

in expand opcode technique we start from primitive instruction (smallest bit in OPCODE)

expand opcode length is required in the fixed length instruction supported CPU design to implement the various instruction

Assume category :

(i) Primitive instruction : lowest/smallest bits in OPCODE.

(ii) Derived instruction : higher(more) bit in OPCODE.

(iii) more (further) derived instruction : highest (more and more) bit in OPCODE.

jis OPCODE ka bit sabse kam hai waha se start karenge
OPCODE bit in increasing order (smallest...more.higher....highest)

OPCODE bits

execution 1 : smallest bit in OPCODE

execution 2 : more higher bit in OPCODE

execution 3 : highest bit in OPCODE

Assume primitive instruction means smallest opcode instruction

Step 1 : identify the primitive instruction (lowest OPCODE bit) in the CPU according to the CPU

step 2 : calculate total number of possible operation

step 3 : identify the free opcode after allocating existed instruction

step 4 : calculate the number of derived instruction possible by multiply

$$\text{free opcode} \times 2^{\text{increment bit in opcode}}$$

Q. Consider a processor which support 6 bit instruction and 4 bit address field. If there exist 2 one address instruction then how many 0 address instruction can be formulated?

--------6bit---------

OPCODE	AF
2bit	4bit

lowest OPCODE bit
(primitive instruction)

-----6bit------

OPCODE
6bit

(derived instruction)
(type 2)

given 1 address instruction (AI) : 2

step 1 : find primitive instruction (type 1) : sabse pehle format likhlo jaise 1AF aur 0 AF diya hai. phir OPCODE but dekho jisme kam vo primitive.

--------6bit---------

OPCODE	AF
2bit	4bit

lowest OPCODE bit
(primitive instruction)

-----6bit------

OPCODE
6bit

(derived instruction)
(type 2)

step 2 : Total number of possible operation : OPCODE $= 2^2 = 4$

given 1 address instruction (AI) : 2

step 3 : idenitfy number of free OPCODE after allocating 1 address instruction $4-2 = 2$

step 4 : calculate the number of derived instruction/operation

free OPCODE x $2^{\text{increment bit in OPCODE}}$

increment bit in OPCODE : pehle 2 bit ka tha abhi 6 bit ka

increment bit in OPCODE $= 6-2 = 4$

$2 \times 2^{6-2} = 2 \times 2^4 = 32$

--------6bit---------

OPCODE	AF
2bit	4bit

lowest OPCODE bit
(primitive instruction)

-----6bit------

OPCODE
6bit

(derived instruction)
(type 2)

feeling / working of the concept

1AF

---------6bit---------

OPCODE	AF
2bit	4bit

lowest OPCODE bit
(primitive instruction)

0AF

-----6bit------

_ _	_ _ _ _
2bit	4bit

(derived instruction)
(type 2)

(primitive instruction) (type 2)

total operation = $2^2 = 4$

2bit ka OPCODE already tha ab AF wala 4 bit idhar OPCODE me club hokar kaam karega

00	USED (given : 2)
01	
10	FREE
11	

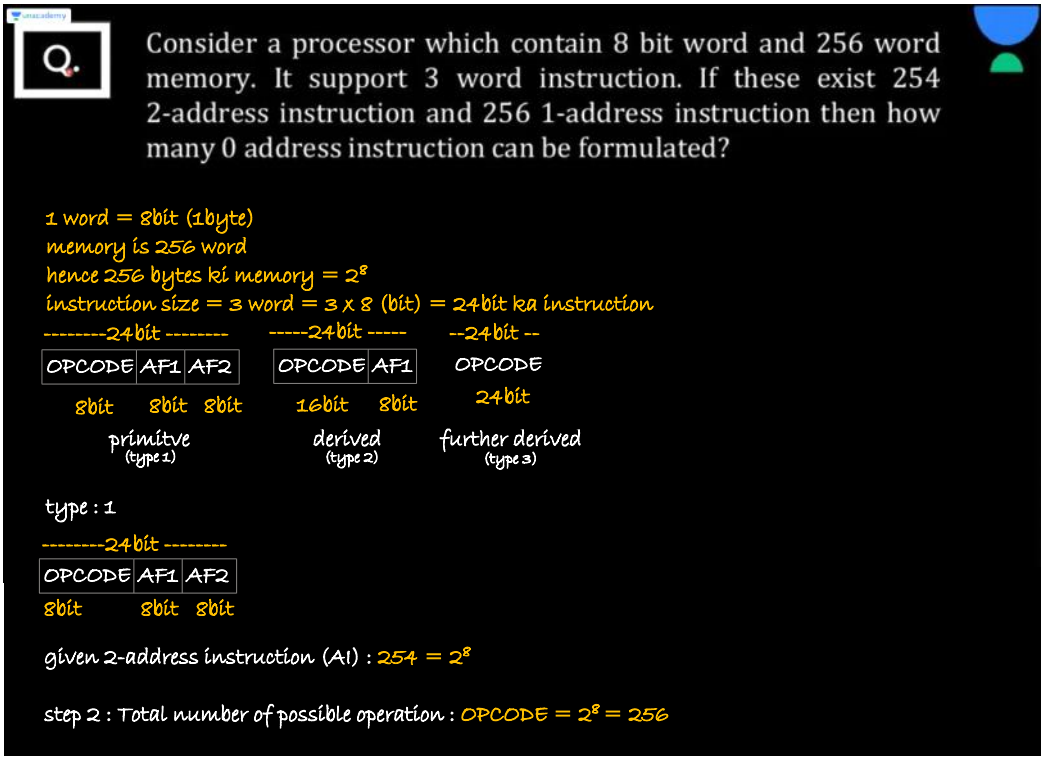
yeh 2 bache hue kitno ke sath combination banayege?

2 ki power 4 combination islie kiya :

__	____
10	0000
	0001
.	
.	
.	
.	
.	
.	
10	1110
	1111
	16

__	____
11	0000
	0001
.	
.	
.	
.	
.	
.	
11	1110
	1111
	16

agar expand OPCODE nahi hota toh 4 x 2^4 hota lekin idhar 2 already use hogye islie 2 x 2^4



given 2 address instruction (AI) : 254 = 2

step 2 : Total number of possible operation : OPCODE = $2^8 = 256$

step 3 : idenitfy number of free OPCODE after allocating 2 address instruction 256-254 = 2

type : 2

-----24bit -----

OPCODE	AF_1
16bit	8bit

given 2-address instruction (AI) : $256 = 2^8$

calculate the number of derived instruction/operation

= free OPCODE x 2 (increment bit in OPCODE)

= 2 x 2^{16-8}

= 2 x 2^8

= 512

idenitfy number of free OPCODE after allocating 1 address instruction 512 - 256 = 256

type : 3

--24bit --

OPCODE
24bit

calculate the number of derived instruction/operation

= free OPCODE x 2 (increment bit in OPCODE)

= 256 x 2^{24-16}

= 256 x 2^8

= 2^{16}

= 65536.

type : 1

--------24bit --------

OPCODE	AF1	AF2
8bit	8bit	8bit

given 2-address instruction (AI) : 254

step 2 : Total number of possible operation : OPCODE = $2^8 = 256$

step 3 : idenitfy number of free OPCODE after allocating 2 address instruction 256-254 = 2

type : 2

-----24bit -----

OPCODE	AF_1
16bit	8bit

given 2-address instruction (AI) : 256

calculate the number of derived instruction/operation

= free OPCODE x 2 (increment bit in OPCODE)

= 2 x 2^{16-8}

= 2 x 2^8

= 512

idenitfy number of free OPCODE after allocating 2 address instruction 512 - 256 = 256

type : 3

--24bit --

OPCODE
24bit

calculate the number of derived instruction/operation

= free OPCODE x 2 (increment bit in OPCODE)

= 256 x 2^{24-16}

= 256 x 2^8

= 2^{16}

= 65536.

Q. Consider a processor with 16 bit instruction. Processor has 15 registers and support 2 address instruction and 1 address instruction. If processor support 256 1-address instruction then number of 2-address instruction are

A) 128

B) 192

C) 240 ✓

D) 248

A 128 B 192

C 240 D 248

16bit instruction
Registers : 15 then address field : 2^4=4bit

--------16bit --------			-----16bit -----		--16bit --
OPCODE	AF1	AF2	OPCODE	AF1	OPCODE
8bit	4bit	4bit	12bit	4bit	16bit
primitve (type 1)			derived (type 2)		further derived (type 3)

type : 1

--------16bit --------		
OPCODE	AF1	AF2
8bit	4bit	4bit

given 2-address instruction (AI) : X

step 2 : Total number of possible operation : OPCODE = 2^8 = 256

step 3 : idenitfy number of free OPCODE after allocating 2 address instruction 256-x

type : 2

-----16bit -----	
OPCODE	AF_1
12bit	4bit

calculate the number of derived instruction/operation

= free OPCODE x $2^{\text{increment bit in OPCODE}}$
= 256-x x 2^{12-8}
= 256-x x 2^4
= 256-x x 16
16 = 256-x
x = 256 - 16
x = 240.

GATE 1996

Instn size = 11 bit, Address field = 4bit Expand Opcode Technique

'5' 2 Address Instn & 32 1AI then Asking OAI ?

11bit instruction
address field : 4bit

--------11bit --------			-----11bit -----		--11bit --
OPCODE	AF1	AF2	OPCODE	AF1	OPCODE
3bit	4bit	4bit	7bit	4bit	11bit
primitve (type 1)			derived (type 2)		further derived (type 3)

type : 1

--------11bit --------		
OPCODE	AF1	AF2
3bit	4bit	4bit

given 2-address instruction (AI) : 5

step 2 : Total number of possible operation : OPCODE = 2^3 = 8

given 2-address instruction (AI) : 5

step 2 : Total number of possible operation : OPCODE = $2^3 = 8$

step 3 : idenitfy number of free OPCODE after allocating 2 address instruction : 8-5 = 3

type : 2

-----11bit -----

OPCODE	AF_1
7bit	4bit

calculate the number of derived instruction/operation

= free OPCODE x $2^{\text{increment bit in OPCODE}}$

= 3 x 2^{7-3}

= 3 x 2^4

= 3 x 16

= 48

idenitfy number of free OPCODE after allocating 1 address instruction : 48 - 32 = 16

type : 3

--11bit --

OPCODE
11bit

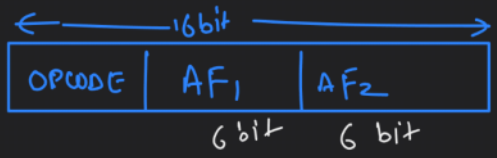
calculate the number of derived instruction/operation

= free OPCODE x $2^{\text{increment bit in OPCODE}}$

= 16 x 2^{11-7}

= 16 x 2^4

= 256.



14 : 2AI

127 : 1AI

60 : 0AI

Now free

16bit instruction

address field : 6bit

--------16bit --------			-----16bit -----		--16bit --
OPCODE	AF1	AF2	OPCODE	AF1	OPCODE
4bit	6bit	6bit	10bit	6bit	16bit
primitve (type 1)			derived (type 2)		further derived (type 3)

type : 1

--------16bit --------

OPCODE	AF1	AF2
4bit	6bit	6bit

given 2-address instruction (AI) : 14

step 2 : Total number of possible operation : OPCODE = $2^4 = 16$

step 3 : idenitfy number of free OPCODE after allocating 2 address instruction : 16 - 14 = 2

OPCODE	AF1	AF2
4bit	6bit	6bit

primitve (type 1)

OPCODE	AF1
10bit	6bit

derived (type 2)

OPCODE
16bit

further derived (type 3)

type : 1

--------16bit --------

OPCODE	AF1	AF2
4bit	6bit	6bit

given 2-address instruction (AI) : 14

step 2 : Total number of possible operation : OPCODE = $2^4 = 16$

step 3 : idenitfy number of free OPCODE after allocating 2 address instruction : 16 - 14 = 2

type : 2

-----16bit -----

OPCODE	AF_1
10bit	6bit

calculate the number of derived instruction/operation

= free OPCODE x $2^{\text{inorement bit in OPCODE}}$

= 2 x 2^{10-4}

= 2 x 2^6

= 2 x 64

= 128

idenitfy number of free OPCODE after allocating 1 address instruction : 128 - 127 = 1

type : 3

--16bit --

OPCODE
16bit

calculate the number of derived instruction/operation

= free OPCODE x $2^{\text{inorement bit in OPCODE}}$

= 1 x 2^{16-10}

= 1 x 2^6

= 64

idenitfy number of free OPCODE after allocating 0 address instruction : 64 - 60 = 4

4 OPCODE can be used in future for new operations.

Q Consider a 16 bit Hypothetical Processor which support 1 word long Instruction. CPU Contain 30 Register, 4KB Memory Size If there exists 11 2 Address Register Reference Instn & 10 1 Address Memory Reference Instn then How Many 0 Address Instn Can be Supported ?

16bit processor : word length = 16bit

instruction : 16bit

Register = 30 then address field : 5bit

memory : 4kb, memory AF : 12bits

--------16bit --------

OPCODE	REG1	REG2
6bit	5bit	5bit

-----16bit -----

OPCODE	MEM
4bit	12bit

--16bit --

OPCODE
16bit

topic : Addressing modes

- addressing mode tells us where the operand is present (location of the operand)

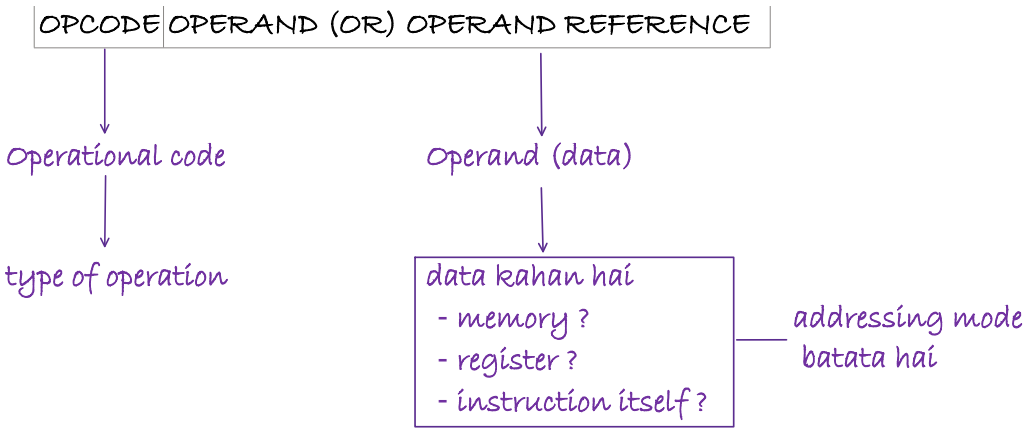
- addressing mode is a technique used to calculate the effective address and operand.

- addressing mode shows the way where the operand is present.

- addressing mode shows the way how to get the operand.

operand kaha aur kaise milega vo batata hai addressing mode.

instruction



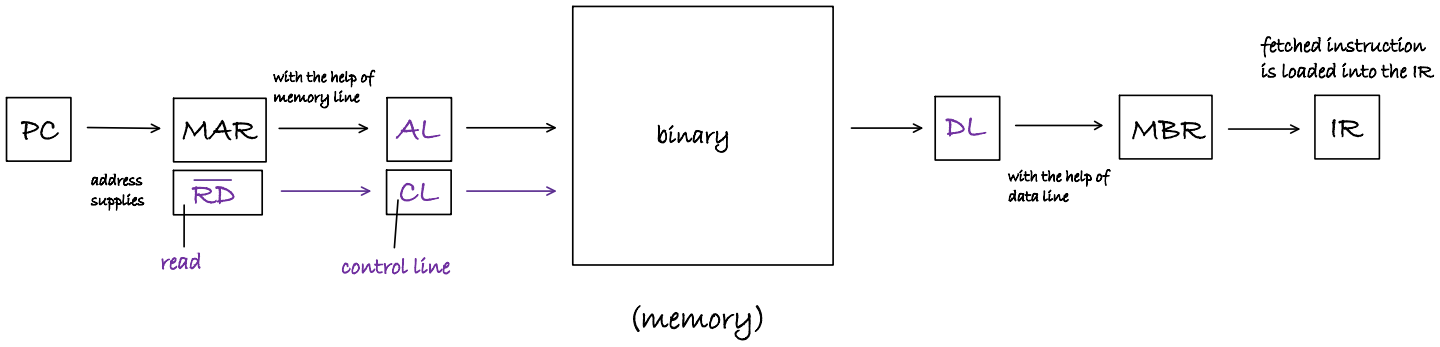
Effective address

Effective address is the actual address of the operand.

Addressing mode

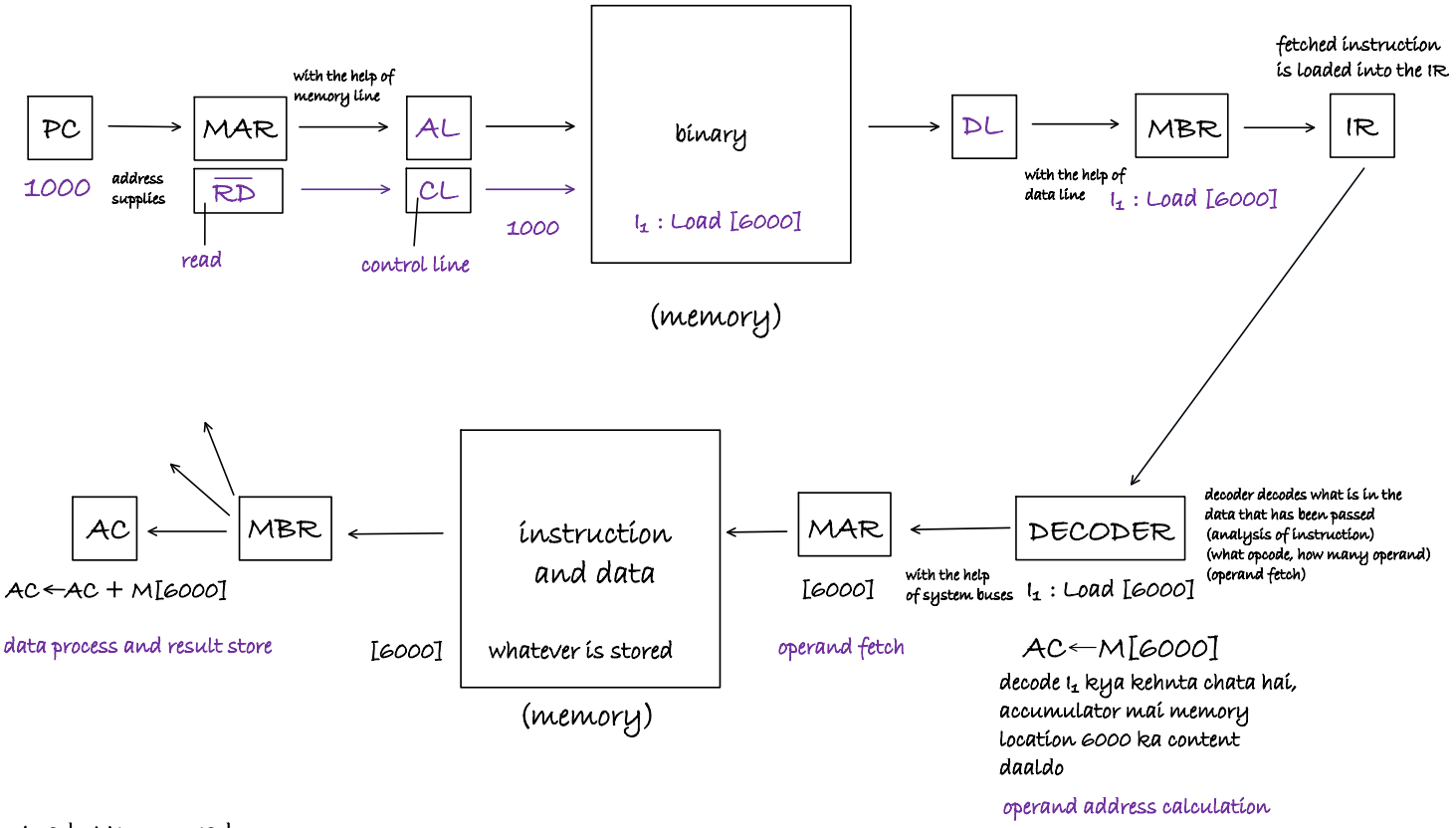
Different ways in which the location of the operand is specified in an instructions are referred as addressing mode. Operand kaha present hai aur usko kaise lena hai)

1. Fetch cycle : to fetch (bring) the instruction from main memory to CPU. (doesn't care what is the instruction)
   (works as a postman)



IR : | OPCODE | MODE | OPERAND/ADDRESS FIELD |

2. Execute cycle : the objective of the execute cycle is to execute (to process) the fetch instruction. it decodes; does the analysis of the instruction. (what is OPCODE, how many operand, operand address calculation, operand fetch, processing, result storage)



Load : Memory read
Store : Memory write

Why addressing mode?

reason 1 :

- to get the location of the operand
- to get to know where the required operand is present
- how to access (deal) with the address field.

reason 2 :

whenever we write program in high level language (C, C++) then we use different structures (machine language kaise deal karega)

(i) constant

(ii) variable(global, local, static)

(iii)pointers

(iv) arrays

H.L.L converted to →

assembly language

all these features are implement by addressing mode in assembly language

machine language

reason 3 : instruction hota hai in binary (OPCODE pata hai humein lekin 5 kya hai)

OPCODE	ADDRESS

can be

- value(constant)
- register (direct, indirect)
- memory (direct, indirect)

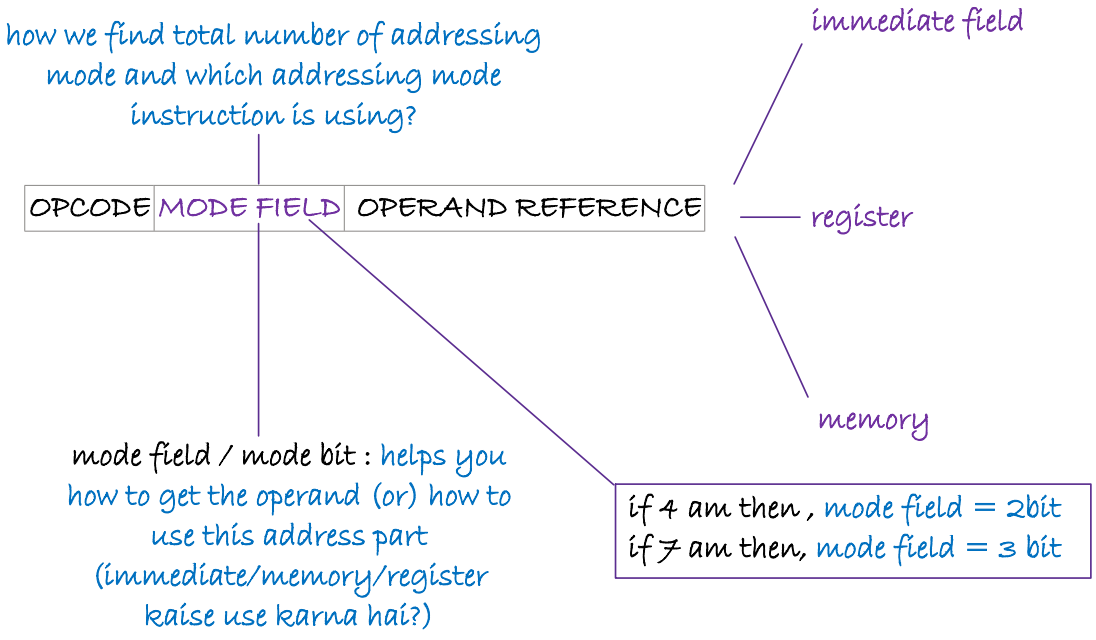
example :

OPCODE	ADDRESS
4bit	3bit
1101	101
opcode	5

what is 5?
- value (constant)?
- register (direct, indirect)?
- memory (direct, indirect)?

to remove this confusion we need addressing mode.

mode field (mode bit) :



so in the computer operand (data) are present in the register (or) memory (or) instruction itself

based on these there are various types of addressing modes.

types of addressing modes

(i) immediate addressing modes (#) or (I)

(ii) direct / absolute addressing mode [ ]

(iii) memory indirect addressing mode (@) or ( )

(iv) register direct addressing mode : reg. name

(V) register indirect addressing mode : index reg. name

(VI) PC relative addressing mode

(VI) PC relative addressing mode

(Vii) index register addressing mode

(Viii) based register addressing mode

(ix) auto decrement addressing mode

(x) auto increment addressing mode

(xi) implied / implicit addressing mode

displacement addressing mode

when operand (data) is present in either immediate field (or) register (or) memory then why various addressing modes are used?

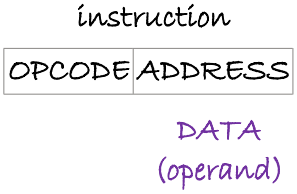
because by 3 only, we cannot implement variable, pointers, arrays, loops etc. so we need different - different addressing modes.

H.L.L ⟶ assembly ⟶ machine language

(i) immediate addressing mode : in this addressing mode operand is present (placed) in the instruction itself

(OR)

operand are present in the address field of the instruction.



note : immediate addressing mode are used to access the constant (or) initialize the register (or) variable with value.

but immediate addressing mode have some limitation (disadvantage)

- addressing mode cannot be used as a destination address because constant does not have any storage

MOV 100 ← R_1

100 mai R_1 nahi daal sakte

- range of constant is limited by the size of address field

n bit unsigned range = (0 to 2^n-1)
n bit signed range = - (2^{n-1} to 2^{n-1}-1)

example :

data available hai instruction mai hi usko access karne ke lie memory ke pass nahi jana pad raha

(i)

MOV R_1 #1000:

(OR)

MVI R_1 1000:

data

$R_1 \leftarrow 1000$

R_1 mai 1000 daal do

(ii)

ADD R_1 #5000

(OR)

ADDI R_1 5000

data

$R_1 \leftarrow R_1 + 5000$

add immediate into R_1

move immediate

MOV R_1 #4000

(OR)

MVI R_1 #4000

$R_1 \leftarrow 4000$

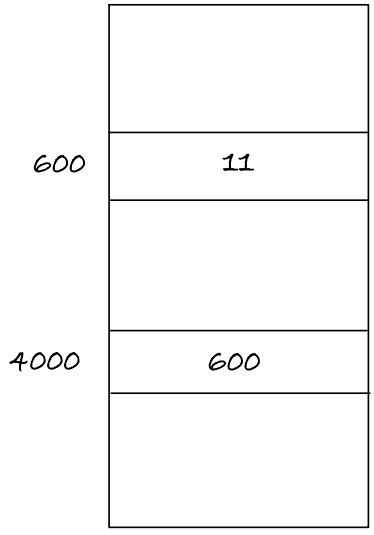
source and destination as well

ADDI R_0 4000

(OR)

ADD R_0 #4000

$R_0 \leftarrow R_0 + 4000$



memory

MOV R_5 #600

$R_5 \leftarrow 600$

ADD R_7 #600

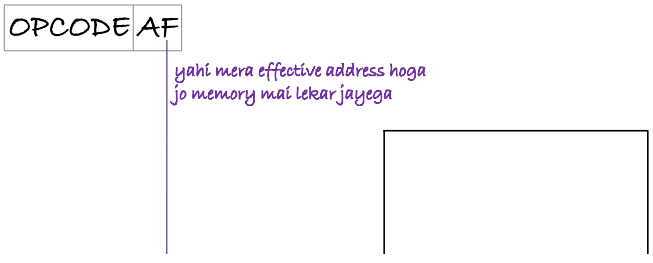
$R_7 \leftarrow R_7 + 600$

memory mai hum gaye hi nahin

(ii) memory direct / absolute addressing mode : in this addressing mode operand is present (placed) in the memory and instruction contains the effective address

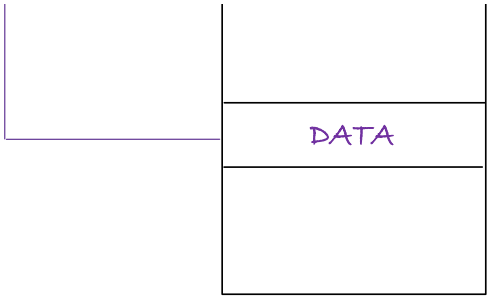
(OR)

address field of the instruction specify the effective address



yahi mera effective address hoga jo memory mai lekar jayega

operand hoga memory mai aur



operand hoga memory mai aur instruction ka address field uska address rakhegi.

note :
- this addressing mode are used to access the variabes.

- 1 (one) memory reference is required to access (read (OR) write) the data.

example :

(i)

ADD R_1 [1000]

$R_1 \leftarrow R_1 + M[1000]$

R_1 mai R_1 aur memory location 1000 ka data add hokar daal do.

(ii)

MOV R [6000]

$R_3 \leftarrow M[6000]$

R_3 mai memory location 6000 ka data daal do.

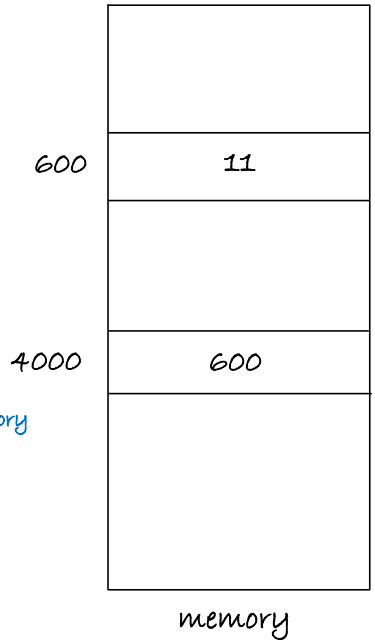
MOV R_1 [4000]

$R_1 \leftarrow M[4000]$

$R_1 \leftarrow 600$

R_1 mai memory location 4000 ka data daal do

R_1 mai 600 (memory location 4000 ka data) daal diya



ADD R_1 [4000]

$R_1 \leftarrow R_1 + M[4000]$

$R_1 \leftarrow R_1 + 600$

R_1 mai R_1 + memory location 4000 ka data daal do

R_1 mai 600 (memory location 4000 ka data) + R_1 daal diya

ADD R_4 [600]

$R_4 \leftarrow R_4 + M[600]$

$R_4 \leftarrow R_4 + 11$

R_4 mai R_4 + memory location 600 ka data daal do

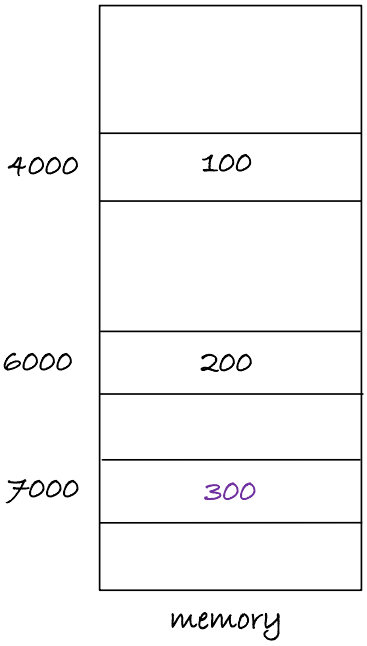
R_4 mai 11 (memory location 600 ka data) + R_4 daal diya

OPCODE	destination	source 1	source 2
ADD	[7000]	[4000]	[6000]

M[7000] ⟵ M[4000] + M[6000]

M[7000] ⟵ 100 + 200

M[7000] ⟵ 300



M[7000] ⟵ M[4000] + M[6000]

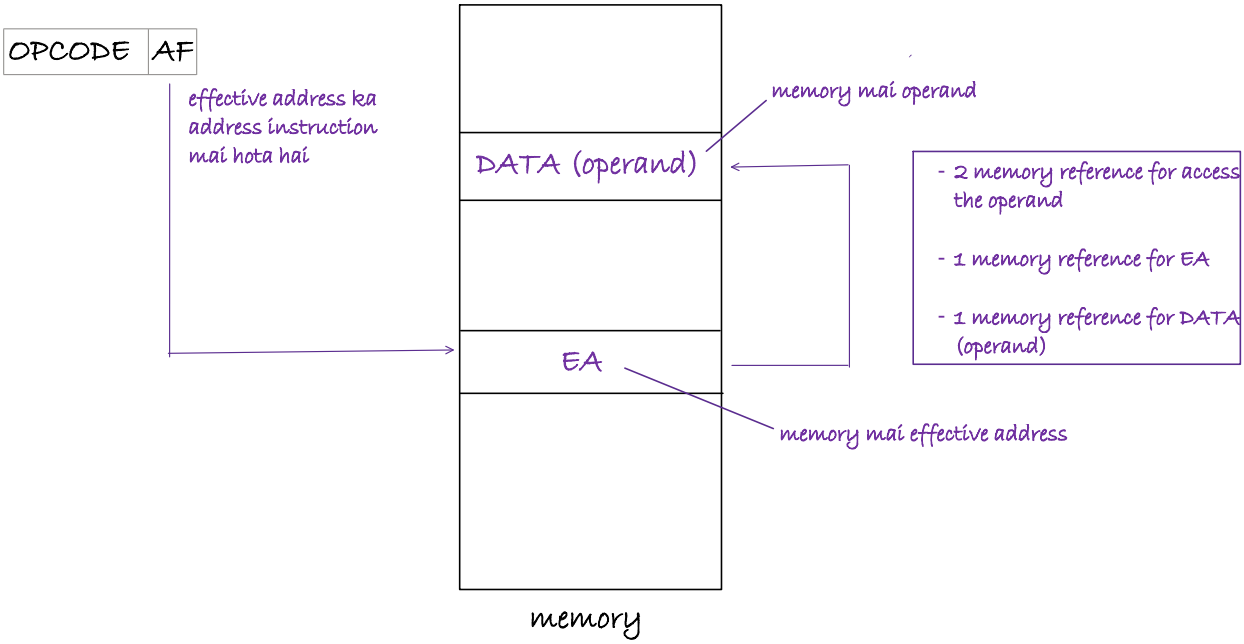
1 memory reference for write

1 memory reference for read

1 memory reference for read

memory reference : kitni baar memory ko access karna pada

(iii) memory indirect addressing mode : in this addressing mode operand is present (placed) in the memory and effecive address (address of operand) is also present in the memory, instruction contains the address of effective address.



note :
- indirect address is used to access the pointers.

- 2 memory reference are required to access (read/write) DATA (operand)

example :

(i)

ADD R_1 [1000]

(OR)

ADD R_1 @1000

$R_1 \leftarrow R_1 + M[1000]$

R_1 mai R_1 aur memory location 1000 ka data add hokar daal do.

(ii)

MOV R_5 @7000

$R_5 \leftarrow M[7000]$

R_5 mai memory location 7000 par jaakar uska data daal R_1 mai daaldo.

(address of effective address)

ADD R_1 @4000

$R_1 \leftarrow R_1 + M[4000]$

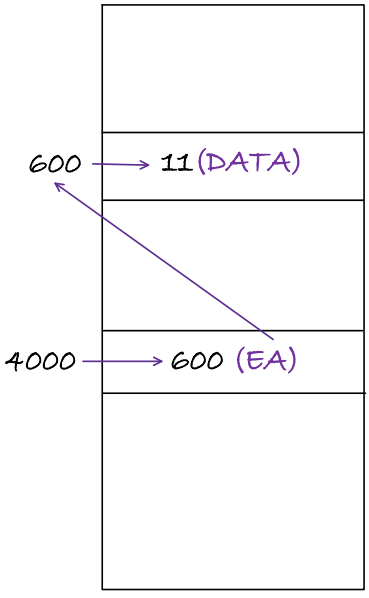
R_1 mai memory location 4000 par jaakar uska data daal R_1 mai add karke R_1 mai daaldo.

$R_1 \leftarrow R_1 + M[600]$

R_1 mai memory location 600 (M[4000]) par jaakar uska data daal R_1 mai add karke R_1 mai daaldo.

$R_1 \leftarrow R_1 + 11$

R_1 mai 11 (M[600]) ko R_1 mai add karke R_1 mai daaldo.



MOV R_3 @4000

$R_3 \leftarrow M[4000]$

R_3 mai memory location 4000 par jaakar uska data daal R_3 mai daaldo

$R_3 \leftarrow M[600]$

R_1 mai memory location 600 (M[4000]) par jaakar uska data daal R_1 mai daaldo.

$R_1 \leftarrow 11$

R_1 mai 11 (M[600]) ko daaldo.

#indirect : direct na jaakr address pass karte hue address par jaa rahe hai

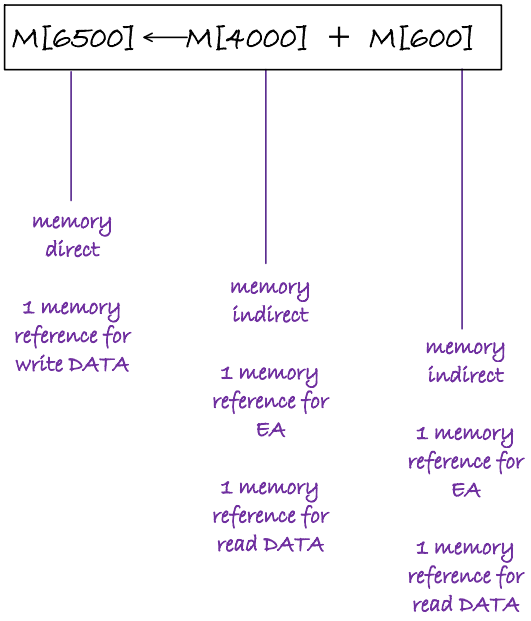
OPCODE	destination	source 1	source 2
ADD	[6500]	@4000	@600

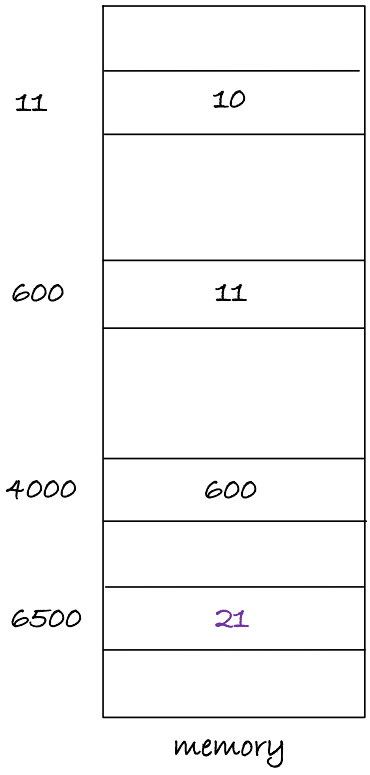
M[6500] ⟵ M[4000] + M[600]

M[6500] ⟵ M[600] + M[11]

M[6500] ⟵ 11 + 10

M[6500] ⟵ 21

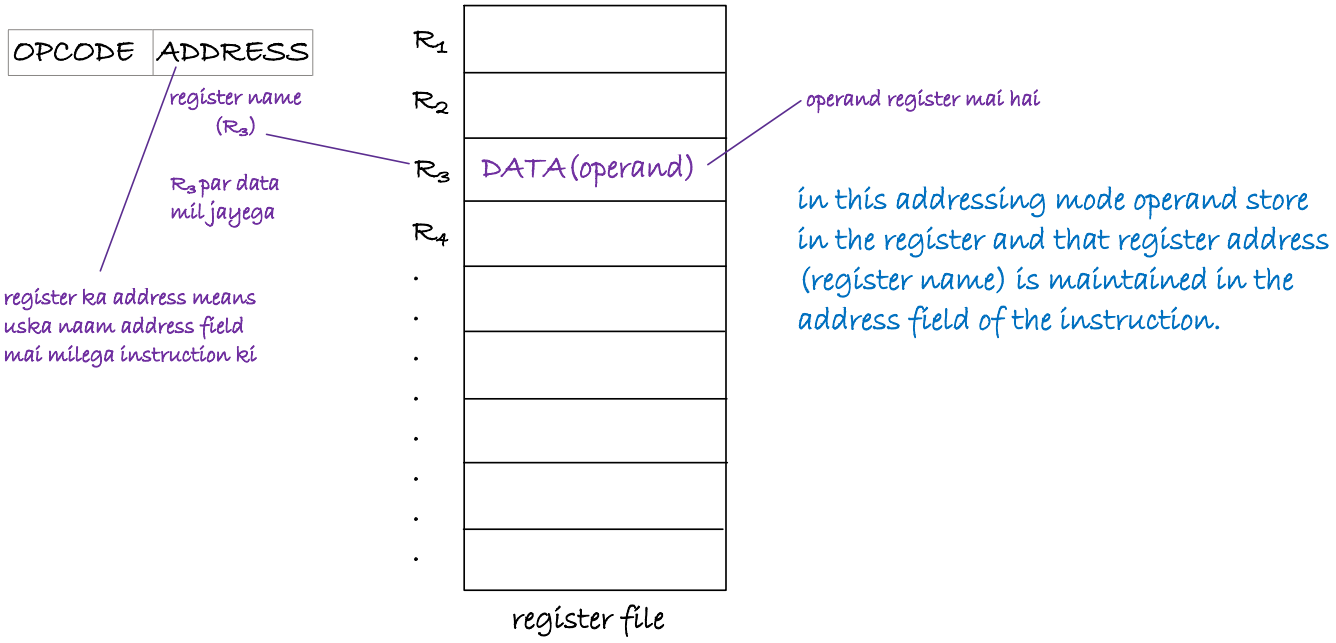




2 addressing modes are used here
- memory direct
- memory indirect

note : we can have different tyoe of addressing modes in one instruction

(iv) register addressing mode : this addressing mode is same as memory direct addressing mode but the difference is here the operand are present in the register instead of memory.



in this addressing mode operand store in the register and that register address (register name) is maintained in the address field of the instruction.

example :

(i)

ADD R_1 [6000] R_1 @7000

$M[6000] \leftarrow R_1 + @7000$

memory direct

1 memory reference

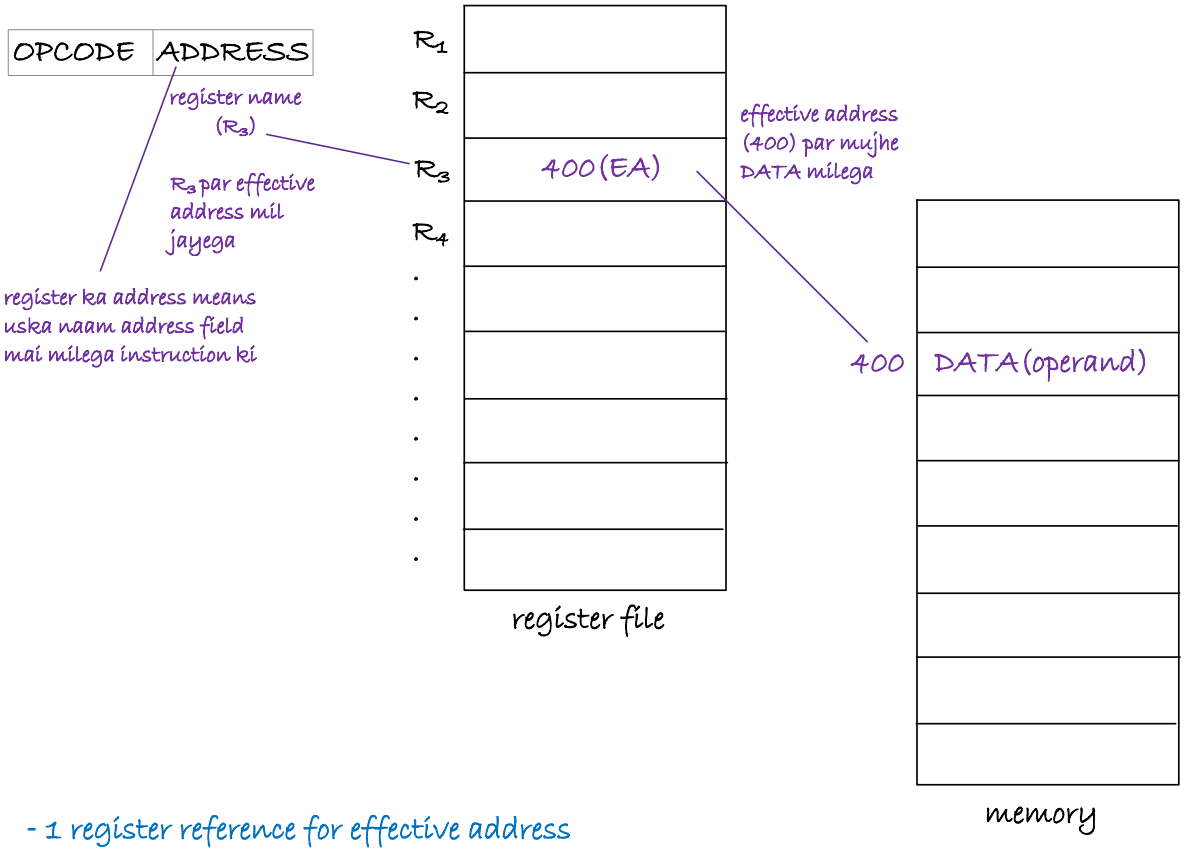
register direct

1 register reference

memory indirect

2 memory reference

(v) register indirect addressing mode : in this addressing mode operand are present or placed in the memory and effective address present in the register.



- 1 register reference for effective address
- 1 memory reference for read/write DATA

example :

(i)

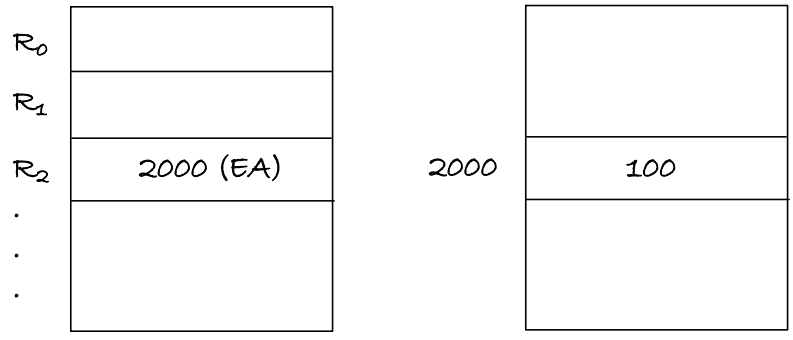
OPCODE	destination	source 1
LOAD	R_1	@R_2

$R_1 \leftarrow @R_2$

$R_1 \leftarrow M[R_2]$

$R_1 \leftarrow M[2000]$

$R_1 \leftarrow 100$



LOAD	R_1	@R_2

register direct

1 register reference

register indirect

1 register reference

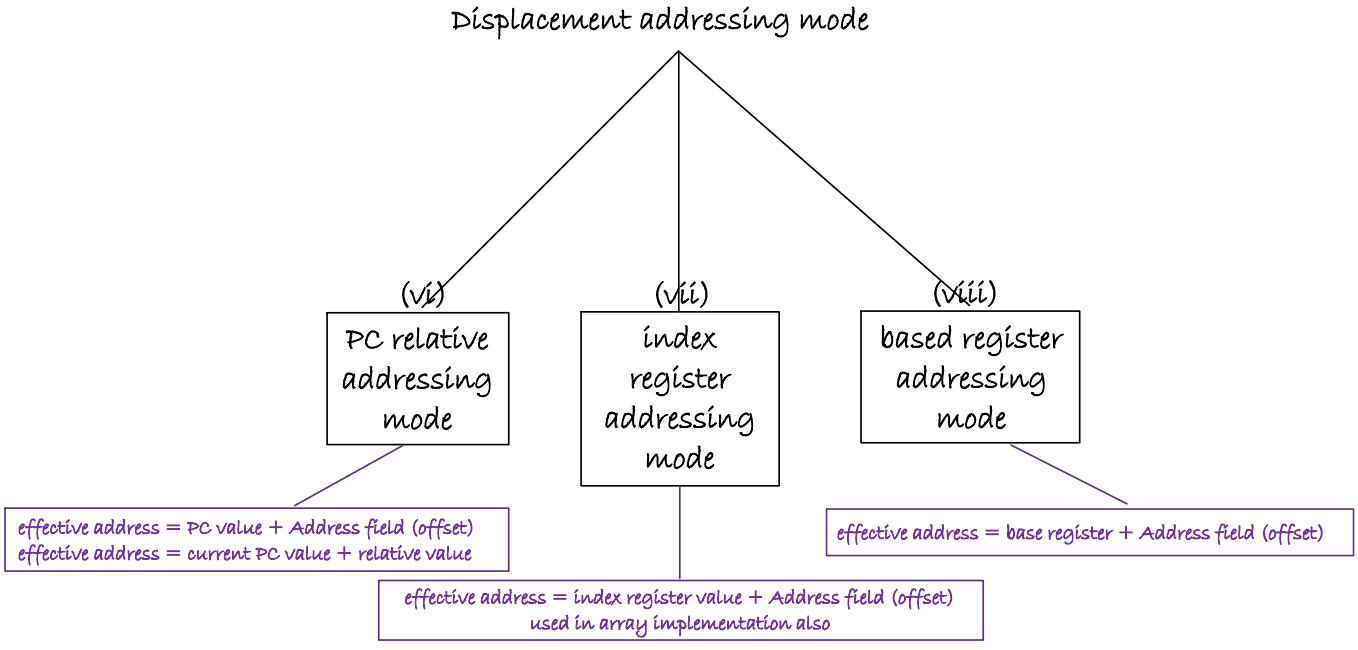
1 memory reference

when we have memory indirect then why register indirect is used?

(i) to shorten the instruction length.

Example : if memory size is 16GB then 34 bits memory address but in the processor (CPU) we have 16, 32 or 64 registers so it requires (4, 5 or 6bits) fewer bits to access the registers.

(ii) accessing the register very fast compared to memory.

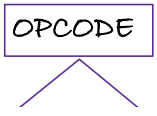


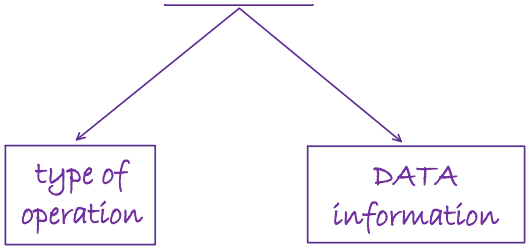
(ix) auto decrement and increment addressing mode : it is similar to register indirect addressing mode in which register value decrement (or) increment

Decrement : Pre-decrement

Increment : Post-increment

(ix) implied / implicit addressing mode : in this addressing mode operand (data info) are present in the OPCODE itself





example :
STC (set carry)
cy = 1

CLC (clear carry)
cy = 0

stack based

ADD

this will also tell us the operations that need to be performed.

TOS (POP)
TOS (POP)
ADD
PUSH (TOS)

important points

- constant : immediate addressing mode
- variable : direct / absolute addressing mode
- pointer : indirect addressing mode
- array : index register addressing mode
- reallocation : base register addressing mode

- ❑ Immediate operand is placed in the instruction
- ❑ Direct operand is placed in the memory and instruction contains the effective address
- ❑ Indirect operand is placed in the memory and effective address (address of operand) is also present in the memory
- ❑ Register same as memory direct am but operand is present in the register instead of memory
- ❑ Register Indirect operand is present or placed in the memory and effective address is present in the register
- ❑ Displacement
- ❑ Stack / implied / implicit operand (DATA info) is present in the OPCODE itself

Q Consider a 16 bit Instruction LOAD to AC With Address 500.

Starting from Memory Location 200 Onwards. Memory is Byte Addressable Opcode is 8 bit

--------16bit--------

LOAD to AC	500
8bit OPCODE	8bitAF

AC ← M[500]

Using various AM Tell EA & Content of AC(Operand)

Q

operand immediate mai

Addressing Mode	Effective Address	Content Of AC
Direct address	500	800
Immediate Operand	201	500
Indirect Address	800	300
Relative address	702	325
Indexed address	600	900
Register	R_1	400
Register Indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

PC = 200

R1 = 400

XR = 100

AC

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

- direct mai effective address instruction mai hai so we go to 500, 500 is address of effective address 800
- indirect mai effective address memory mai hi hai toh 500 mera address

effective address = PC value + Address field (offset)
= 202 + 500
= 702

effective address = index register value + Address field (offset)
= 100 + 500
= 600

Q. In which of the following addressing modes, operand is NOT A part of instruction? [MSQ]

A Immediate

B Direct

C Indirect

D Register

Q. Consider a 16 bit hypothetical processor which support 1 Address Instruction Design with various addressing mode. It contain 8 bit OPCODE. It supports 1 word Instruction stored in the Memory with a starting address of $(745)_{10}$(Decimal) onwards. Address field value of the instruction is 222. Register r_1 contain 111 memory content of [222] is 155. Which of the

contain 8 bit OPCODE. It supports 1 word instruction stored in the Memory with a starting address of $(745)_{10}$(Decimal) onwards. Address field value of the instruction is 222. Register r_1 contain 111 memory content of [222] is 155. Which of the following is/are correct about effective address of various Addressing Mode (AM)? (all values are in decimal)

(I) In the Immediate AM Effective Address is 745.
(II) In the Immediate AM Effective Address is 746.
(III) In the Memory Indirect AM Effective Address is 155.
(IV) In the Index AM effective Address is 333 (r_1 as index register)

--------16bit--------

OPCODE	ADDRESS
	222

$r_1 = 111$

745	OPCODE
746	ADDRESS
222	155

(I) In the Immediate AM Effective Address

effective address = 746
operand (DATA) = 222

(ii) in memory direct

effective address = 222
operand (DATA) = 155

(III) In the Memory Indirect AM Effective Address is

effective address = 155
operand (DATA) = do not know m[155]

(IV) In the Index AM effective Address is (r_1 as index register)

effective address = 111 + 222 = 333

Consider the following program segment, Here R1, R2 and R3 are the general purpose register.

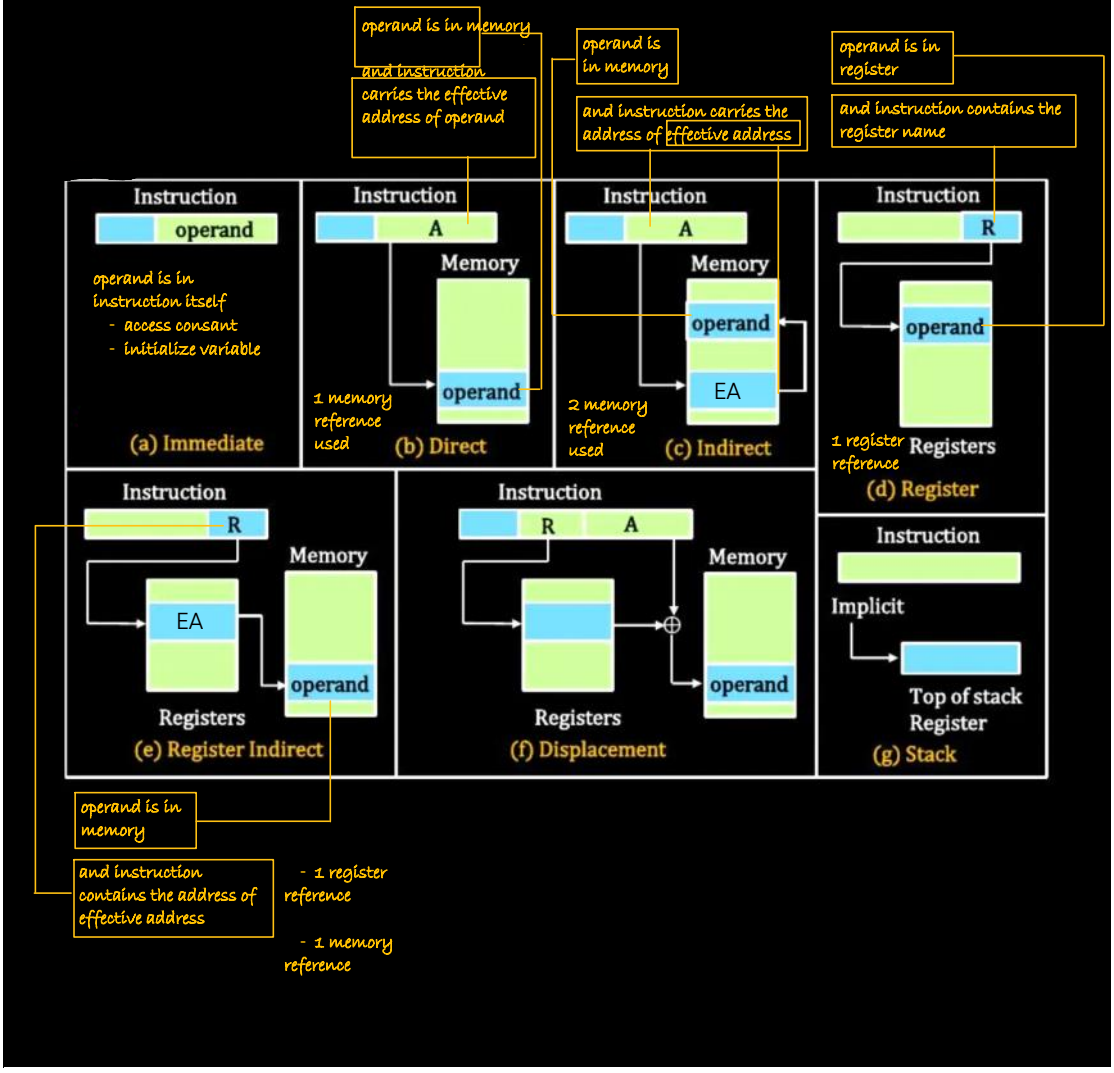
Instruction	Operation	Instruction size (no. of words)
MOV R1, (3000)	R1←M[3000]	2
LOOP		
MOV R2, M[R3]	R2←M[R3]	1
ADD R2, R1	R2←R1 + R2	1
MOV (R3), R2	M[R3] ←R2	1
INC R3	R3←R3 + 1	1
DEC R1	R1←R1 - 1	1
BNZ LOOP	Branch on not zero	2
HALT		Stop

Assume that the content of memory location 3000 is 10 and the content of the Register R3 is 2000. The content of each of the memory locations from 2000 to 2010 is 100. The program is loaded from the memory location 1000. All the Numbers are in decimal.

loop : result of previous instruction par lagta hai means

Numbers are in decimal.

loop : result of previous instruction par lagta hai means MOV R_1, (3000) ke behalf par baat karega



effective address par jaakar jo address milega usme operand

effective address is register name in instruction waha par content milega

operand mai hi

effective address par operand

13.1 / ADDRESSING MODES 459

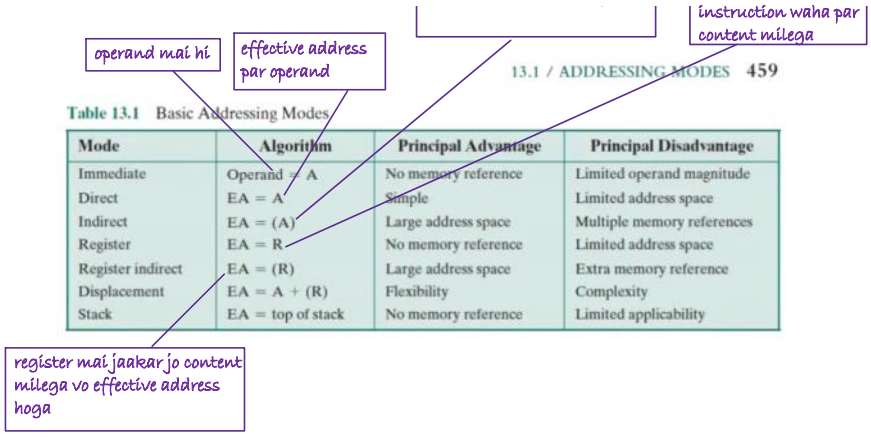


Table 13.1 Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

register name (or) register address are different?

Processor generally having 32 registers (or) 64 registers (GPR)

if 32 register then register AF = 5bit

assume,

if i write R_1 then computer writes 00001

how i write (register name)

how computer writes (address)

Problems

13.1 Given the following memory values and a one-address machine with an accumulator, what values do the following instructions load into the accumulator?

- Word 20 contains 40.
- Word 30 contains 50.
- Word 40 contains 60.
- Word 50 contains 70.

1. Load immediate 20 : 20 (instruction mai hi operand)

AC ← 20

2. Load direct 20 : 40 (operand)

load [20]
AC ← M[20]
AC ← 40

3. Load indirect 20 : 60 (address of immediate address)

AC ⟵40

3. Load indirect 20 : 60 (address of immediate address)

load (20)
or
load @20
AC ⟵[M[20]]
AC ⟵M[40]
AC ⟵60

20	40
30	50
40	60
50	70

4. Load immediate 30 : 30 (instruction mai hi operand)

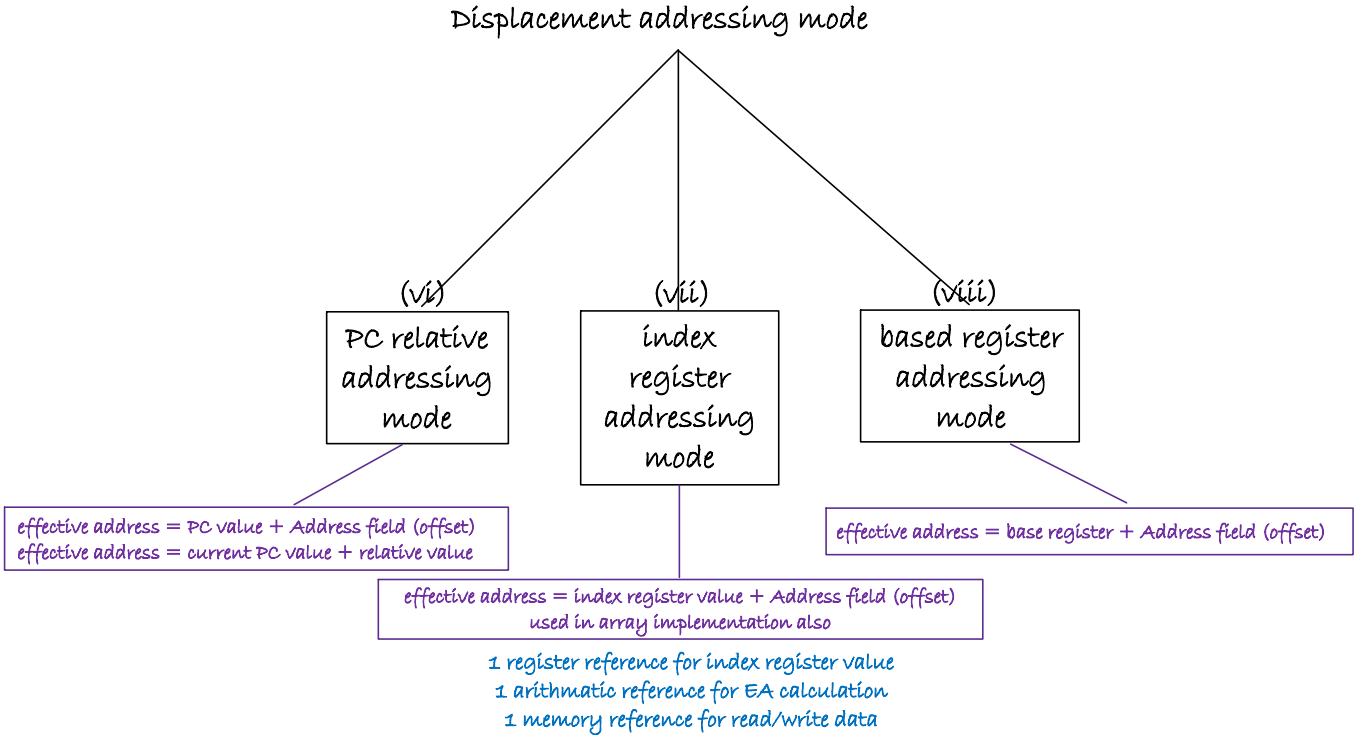
AC ⟵30

5. load direct 30 : 50 (operand)

load [30]
AC ⟵M[30]
AC ⟵50

6. Load indirect 30: 70 (address of immediate address)

load (30)
or
load @30
AC ⟵[M[30]]
AC ⟵M[50]
AC ⟵70



topic : displacement addressing mode

A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing. It is known by a variety of names depending on the context of its use, but the basic mechanism is the same. We will refer to this as **displacement addressing**:

$$EA = A + (R)$$

Displacement addressing requires that the instruction have two address fields, at least one of which is explicit. The value contained in one address field (value = A) is used directly. The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address.

We will describe three of the most common uses of displacement addressing:

- Relative addressing
- Base-register addressing
- Indexing

mentioned hogi

because in the CPU only one program counter

format :

OPCODE	REGISTER	AF(A)

If Register is :

- Program counter then it will become relative addressing mode / PC relative
- index register then it will become indexed addressing mode
- base register then it will become based register mode

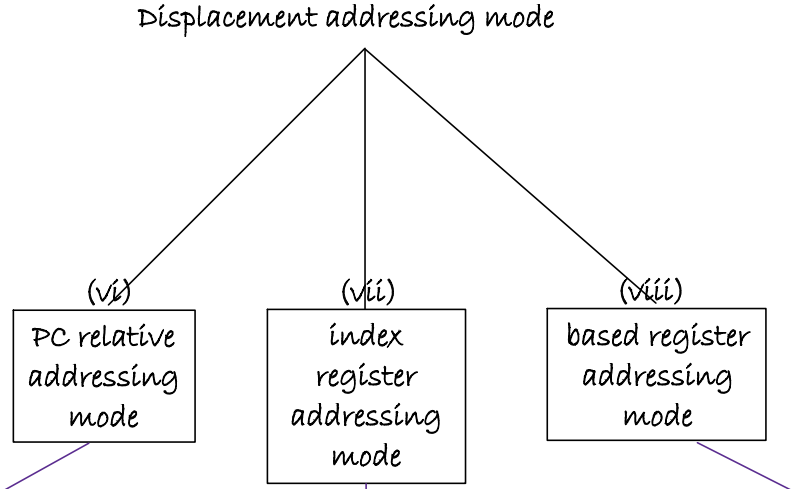
effective address = R + A (AF) offset

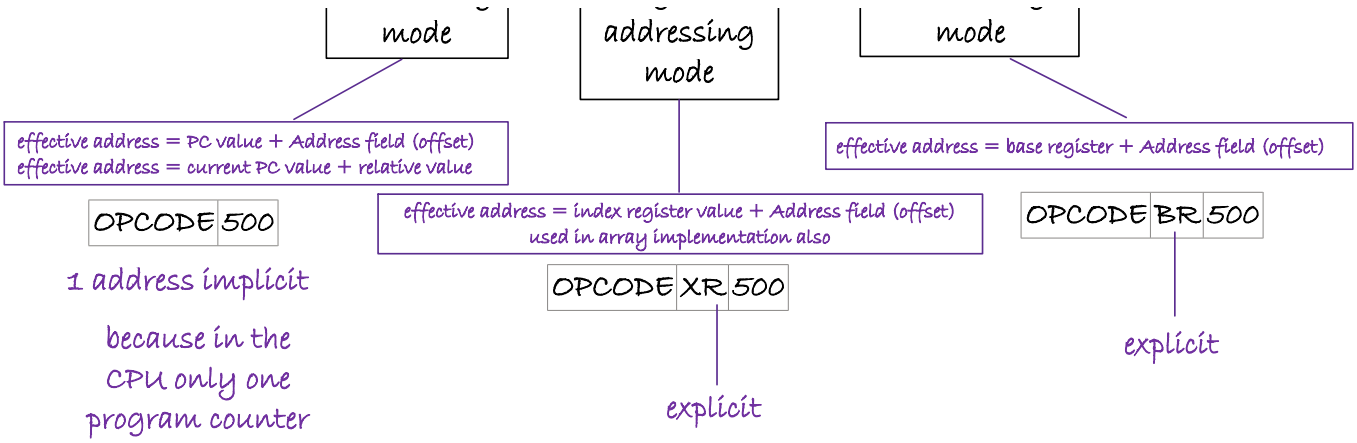
here, R can be
- current value
- index register
- base register

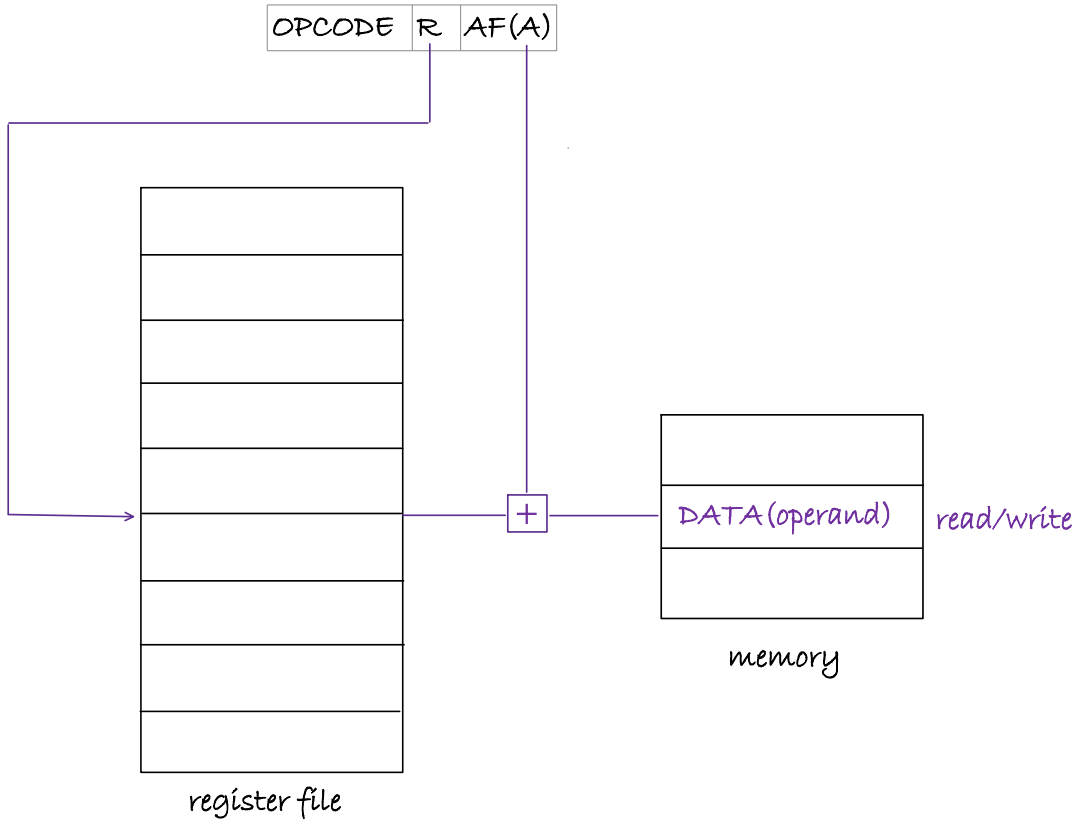
may be implicit

explicit

because in the CPU only one program counter







(i) index register addressing mode : index addressing mode are used to implement the array

to implement the array we require two things

- starting / base address of array

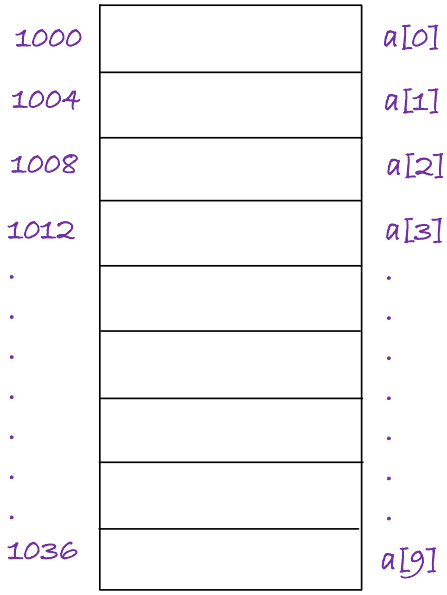
- index value

starting / base address of array : present in the address field of the instruction (offset)

index value : stored in index register

effective address = index register value + Address field (offset)

assume
int = 4byte



register file

in H.L.L

a[2] = 1008

here compiler will convert a[2] into address 1008

a[n] = starting address of array + n x size of data type

in assembly language

effective address = 8 + 1000

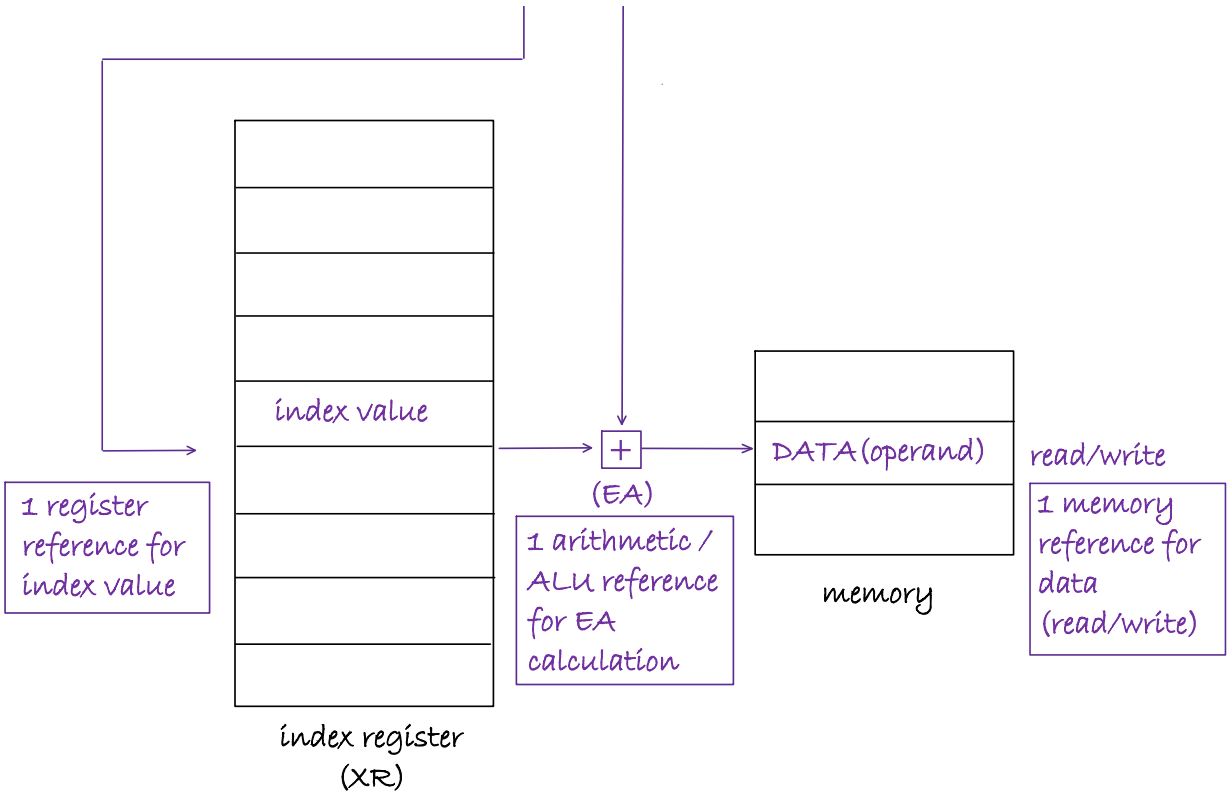
index value (stored in index register) — starting address

here we have to tell the index value.

a[2] = 1000 + 2 x 4

= 1000 + 8 (index value (stored in index register))

= 1008



load to AC | 500

XR = 100

1 register reference for index value

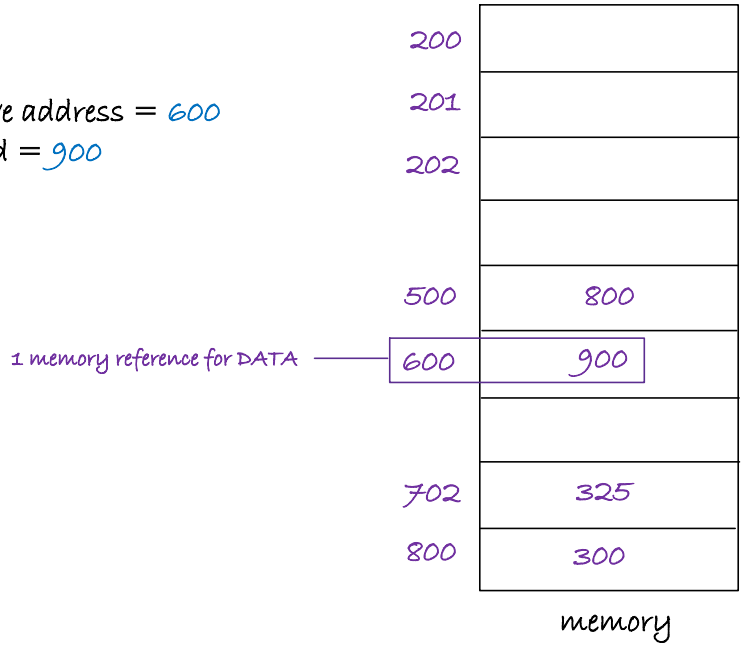
effective address = XR value + Address field

= 100 + 500 — 1 ALU reference for EA calculation

= 600

effective address = 600

operand = 900



consider a processor have 32 registers

index register (XR)

- at the CPU design time, one special purpose register is made as index register then this special purpose register implicitly access the index value from index register. (32 mai se ek special purpose means index register bana diya, it will be used for only this purpose)
  disadvantage : one register wastage because of not heavy utilisation (kabhi kabhi hi kaam aayega index register)
- in most of the cases, any one general purpose register is designated as index register and that register name will be mentioned. (kisi general purpose register ko as a index register use karlenge whenever needed so we will have 32 registers)
  example : r_1 as a index register
  advantage : we can use this register in future as general purpose register (index register ko phir hum wapas se general purpose jaise use kar sakte)

Q.3 If we use internal data forwarding to speed up the performance of a CPU (R1, R2 and R3 are registers and M[100] is a memory reference), then the sequence of operations.

R1 → M[100]
M[100] → R2
M[100] → R3 can be replaced by

[GATE - 2004: 2 Mark]

A. R1 → R3
R2 → M[100]

B. M[100] → R2
R1 → R2
R1 → R3

C. R1 → M[100]
R2 → R3

D. R1 → R2
R1 → R3
R1 → M[100]

Q.4 Match List-I with List-II and select the correct answer using the codes given below the lists:

[GATE - 2005: 2 Mark]

List-I	List-II
A. A[I] = B[J];	1. Indirect addressing
B. while[*A++];	2. Indexed addressing
C. int temp = *X;	3. Auto increment

Codes:

	A	B	C
A	3	2	1
B	1	3	2
C	2	3	1
D	1	2	3

Q.5 The memory locations 1000, 1001 and 1020 have data values 18, 1 and 16 respectively before the following program is executed. [GATE - 2006: 2 Mark]

MOVI	Rs, 1	Move immediate
LOAD	Rd, 1000(Rs)	Load from memory
ADD I	Rd, 1000	Add immediate
STOREI	0(Rd), 20	Store immediate

Which of the statements below is TRUE after the program is executed?

A. Memory location 1000 has value 20

B. Memory location 1020 has value 20

C. Memory location 1021 has value 20

D. Memory location 1001 has value 20 ✓

MOVI | Rs, 1 | Move immediate

$RS = 1$

LOAD | Rd, 1000(Rs) | Load from memory

OPCODE	A	R_S

contains index value

effective address $= A + R$

$= m[1000 + R_s]$

$= m[1000 + 1]$

$= m[1001]$

index addressing mode used here

$RD \leftarrow m[1001]$

ADD I | Rd, 1000 | Add immediate

$RD \leftarrow RD + 1000$

$RD \leftarrow 1 + 1000$

$RD \leftarrow 1001$

STOREI | 0(Rd), 20 | Store immediate

$m[0+RD]$

$m[0 + 1001]$

$m[1001] \leftarrow 20$

(i) immediate am used here

(ii) index am used here

Q.7 The absolute addressing mode [GATE - 2002: 1 Mark]

A. The operand is inside the instruction

B. The address of the operand is inside the instruction ✓

A The operand is inside the instruction

B The address of the operand is inside the instruction

C The register containing the address of the operand is specified inside the instruction.

D The location of the operand is implicit.

Q. 8 A CPU has 24-bit instructions. A program starts at address 300 (in decimal). Which one of the following is a legal program counter (all values in decimal)? [GATE-1 Marks]

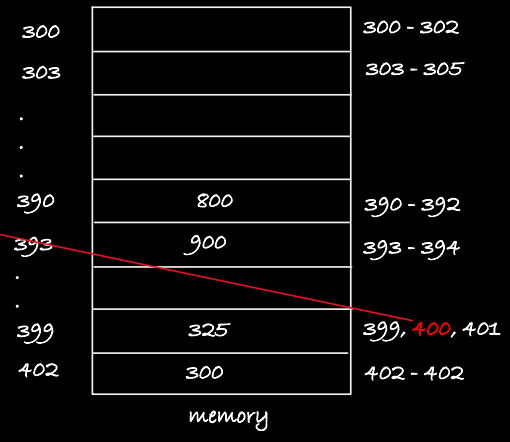
(a) 400 (b) 500 (c) 600 (d) 700

24 bit = 3byte
address = 300

assume 400 is starting address
300 + 3x = 400
3x = 100
x = 100/3 = 33.3333
no, it is not a starting address

24 bit = 3byte
address = 300

assume 600 is starting address
300 + 3x = 600
3x = 300
x = 300/3 = 100
yes, it is a starting address



PC denotes the starting address of the instruction

why auto decrement and auto increment am is used?

when we want to access / perform same operation in multiple location (assume 100 location) then different different instructions are required

but with the help of auto decrement and auto increment am it is possible by using only one instruction we perform same operation or access the multiple location

here we use loops.

example :
for (i = 1; i<=100, i++)

-----	address 1

-----	address 2

.

.

-----	address 100

Consider the following program segment, Here R1, R2 and R3 are the general purpose register.

Instruction	Operation	Instruction size (no. of words)
MOV R1, (3000)	R1←M[3000]	2
LOOP;		
MOV R2, M[R3]	R2←M[R3]	1
ADD R2, R1	R2←R1 + R2	1
MOV (R3), R2	M[R3] ←R2	1
INC R3	R3←R3+1	1
DEC R1	R1←R1-1	1
BNZ LOOP	Branch on not zero	2
HALT		Stop

register indirect am

R_1 pe memory location 3000 ka data daal do hence, $R_1 = 10$

loop or branch condition previous instruction ke result par depend karegi

m[R3], m[R3] ka value hai 2000 aur m[2000] par hai 100 hence R2 = 100

R1 = 10 and R2 = 100 hence R1 + R2 = 110

m[R3], means m[2000] mai 110 daal do

R3 hai m[2000] usme + 1 kardo hence, m[2001]

R1 hai 10 usme 1 decrement kardo hence 10 - 1 = 9

R1 mai 0 nahi hai toh dobara loop mai bejh do

Assume that the content of memory location 3000 is 10 and the content of the Register R3 is 2000. The content of each of the memory locations from 2000 to 2010 is 100. The program is loaded from the memory location 1000. All the Numbers are in decimal.

auto increment

auto decrement

2000	~~100~~	110
2001	~~100~~	109
2002	~~100~~	108
2003	~~100~~	107
2004	~~100~~	106
2005	~~100~~	105
2006	~~100~~	104
2007	~~100~~	103
2008	~~100~~	102
2009	~~100~~	101
2010	~~100~~	100

memory

out of loop

OPCODE AF

$r_3 = 2000$

M[3000] = 10

note : by using auto decrement / auto increment am we are accessing the multiple locations (or) we are performing same operation in multiple locations.

LOOP;		
MOV R2, M[R3]	R2←M[R3]	1 R2 mai m[R3] means m[2001]=100 hai vo daaldo
ADD R2, R1	R2←R1 + R2	1 R2 mai R1 (9) + R2 (100) = 109 daaldo
MOV (R3), R2	M[R3] ←R2	1 M[R3] means m[2001] mai R2 (109) daal do
INC R3	R3←R3+1	1 R3 (m[2001]) mai R3 (m[2001]) + 1 kardo = m[2002]
DEC R1	R1←R1-1	1 R1 mai R1 (9) - 1 = 8 daal do
BNZ LOOP	Branch on not zero	2 R1 mai 0 nahi hai toh dobara loop mai bejh do
HALT		Stop

so, the loop will continue to execute until R1 becomes 0 and in memory the decrement will continue

so, the loop will continue to execute until R1 becomes 0 and in memory the decrement will continue

Q.10 Assume that the memory is word addressable. After the execution of this program, the content of memory location 2010 is [2 marks]

(a) 100 (b) 101 (c) 102 (d) 110

Q.9 Assume that the memory is word addressable. The number of memory references for accessing the data in executing the program completely is [2 marks]

(a) 10 (b) 11 (c) 20 (d) 21

in 1 loop / iteration
2 times we access the memory

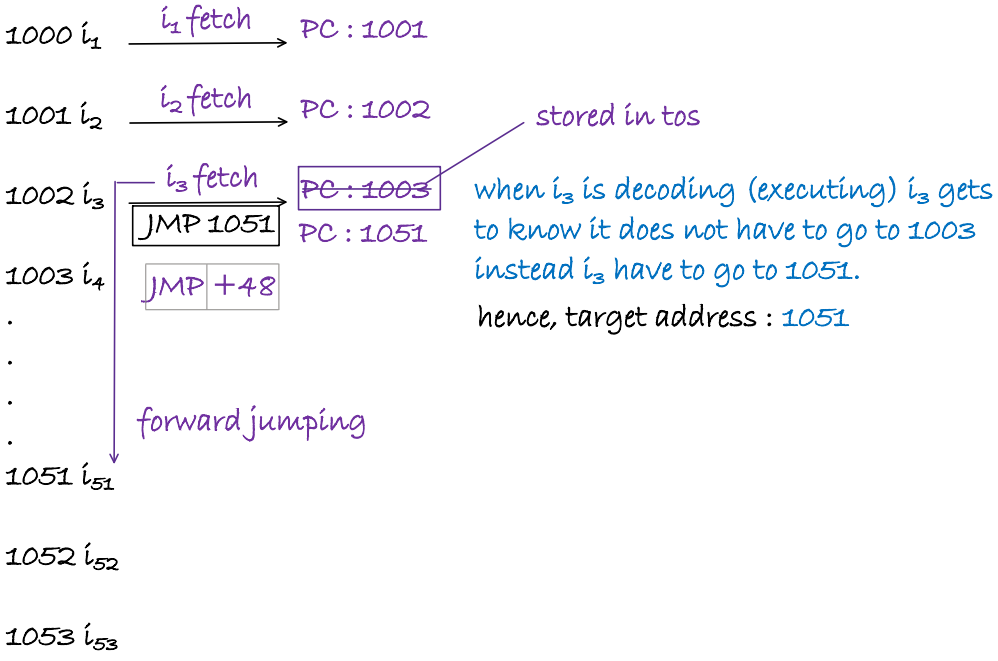
loop = total = 10 times
10 x 2 = 20

total memory reference = 1 (bahar wala) + 20 (loop wale) = 21

concept : how to find target address?

(i) in PC relative addressing mode

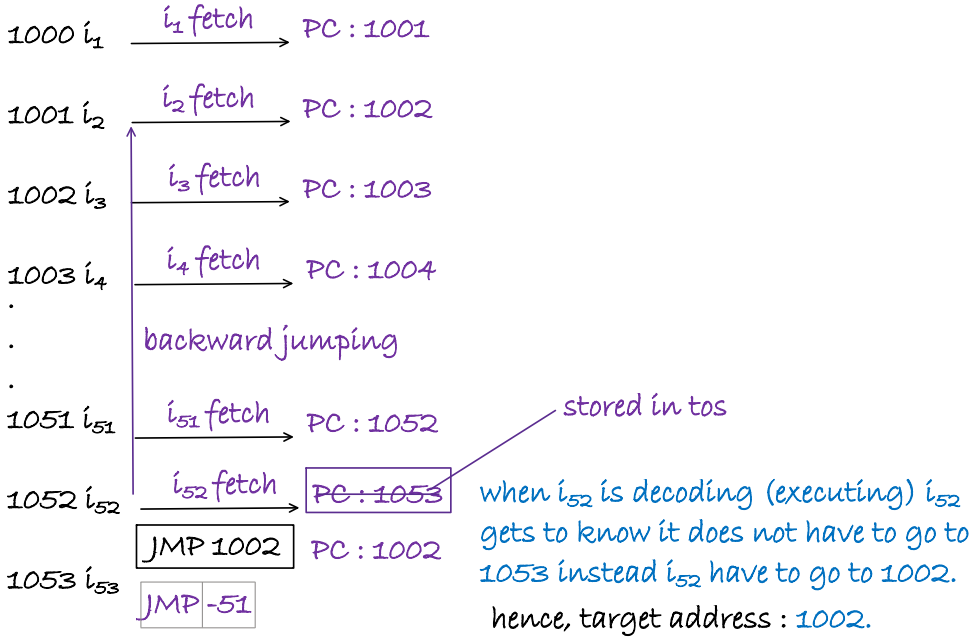
case (i)



what is the displacement / relative value?
target address = current PC value + displacement / offset / relative value

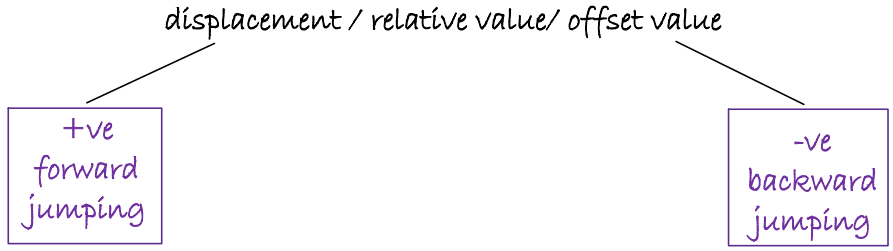
1051 = 1003 + displacement / offset / relative value
relative value = 1051 - 1003
relative value = +48

case (ii)

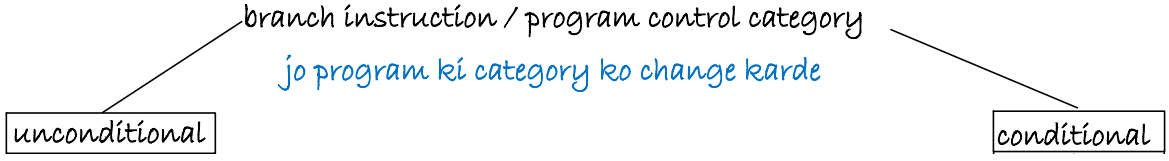


what is the displacement / relative value?
target address = current PC value + displacement / offset / relative value

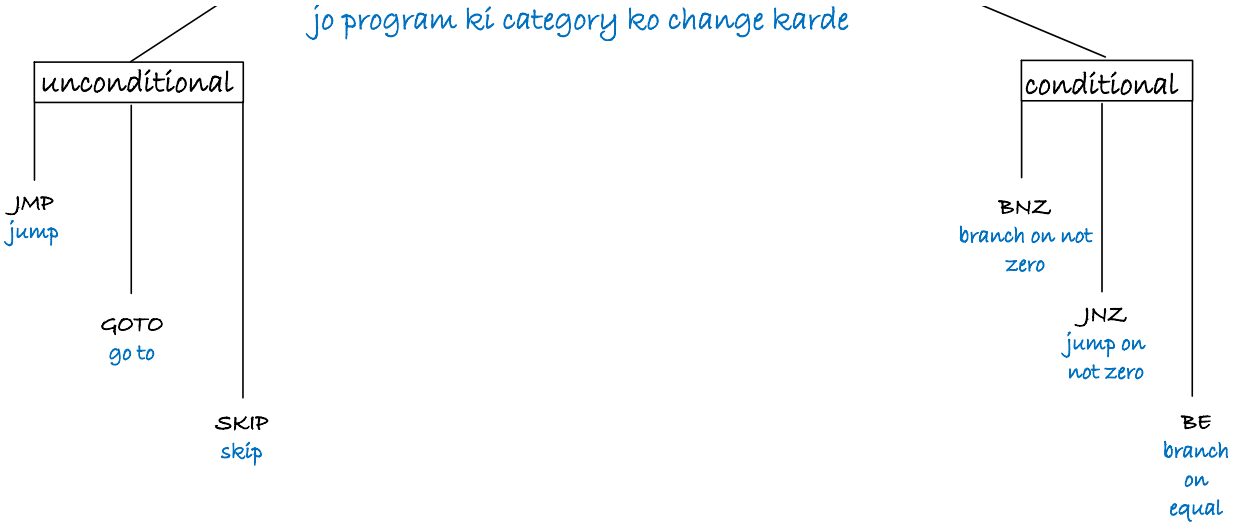
1002 = 1053 + displacement / offset / relative value
relative value = 1002 - 1053
relative value = -51

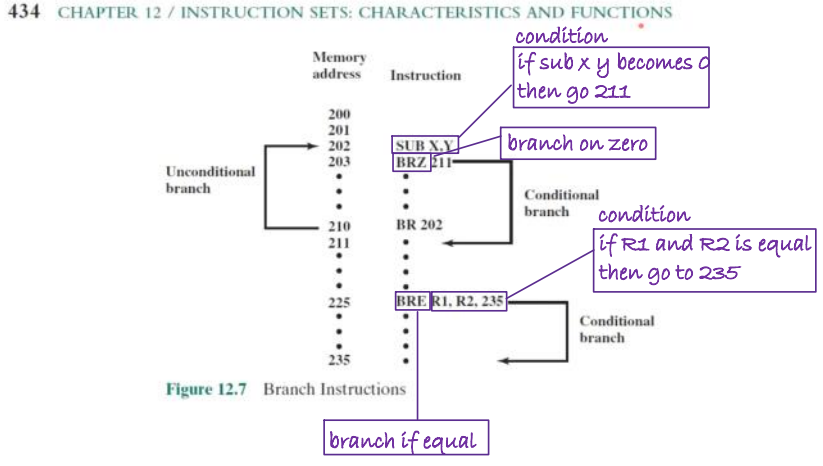


note : relative AM and base AM register are used to write re-allocatable code.



jo program ki category ko change karde





relative value or offset : displacement between current location of PC to target location.

case (i)

1000 i_1 —(i_1 fetch)→ PC : 1001

1001 i_2 —(i_2 fetch)→ PC : 1002

1002 i_3 —(i_3 fetch)→ PC : 1003 JMP +48 — jo kaam tha i_{51} par le jaane ka vo isse ho gaya

1003 i_4

.

.

.

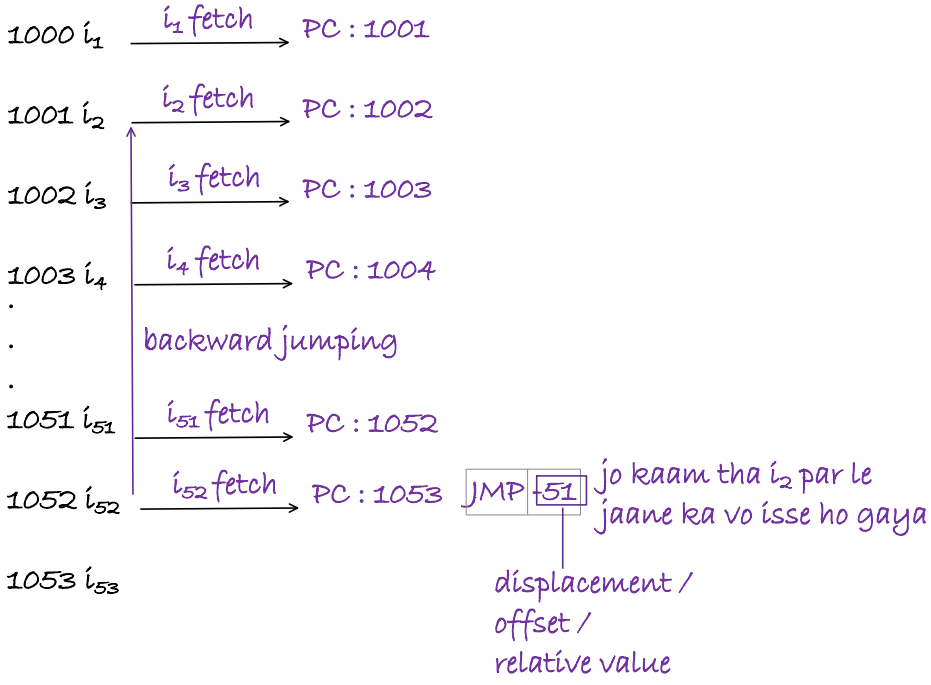
forward jumping

displacement / offset / relative value

target address = current PC value + displacement / offset / relative value

1051 = 1003 + displacement / offset / relative value
relative value = 1051 - 1003
relative value = +48

case (ii)



target address = current PC value + displacement / offset / relative value

1002 = 1053 + displacement / offset / relative value
relative value = 1002 - 1053
relative value = -51

benefit : relative AM and base AM register are used to write re-allocatable code.

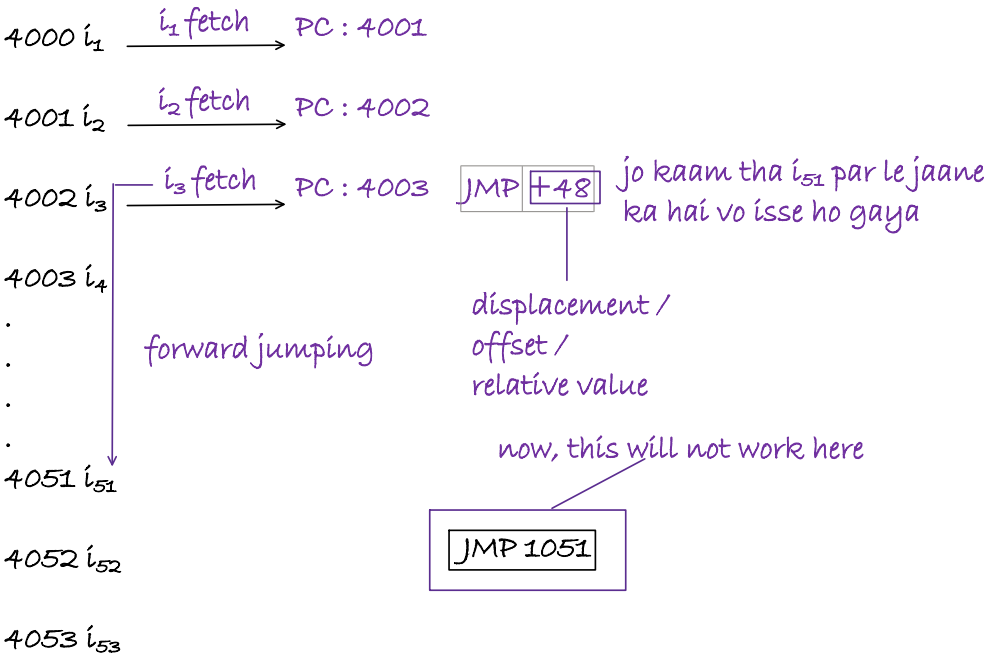
concept : re-allocatable code

if operation system changes or re-allocate the memory address, suppose we start from 4000 instead of 1000 then the procedure we followed before will not be applicable here but with the help of relative addressing mode this will work perfectly.

but why OS change or re-allocate the address?
suppose high priority process comes in CPU then OS will do the swapping means this program will be swapped out from memory to backing store.

after some time same program will reload but at different address

case (i) address changed from 1000 to 4000



target address = current PC value + displacement / offset / relative value
target address = 4003 + 48
target address =4051

benefit : agar mai memory location (re-allocate) change bhi kardu toh displacement / offset / relative idhar bhi kaam karega but with previous method , hume address 1000 par operation perform krna tha jisme JMP 1051 tha, toh hum sirf 1000 par hi operate kar paate but with displacement we can work on any address because it does not stick to any particular address (i.e 1000, 2000, etc) instead it is a operation like JMP +48 that can be perfomed on any address that will be perfoming forward jumping.

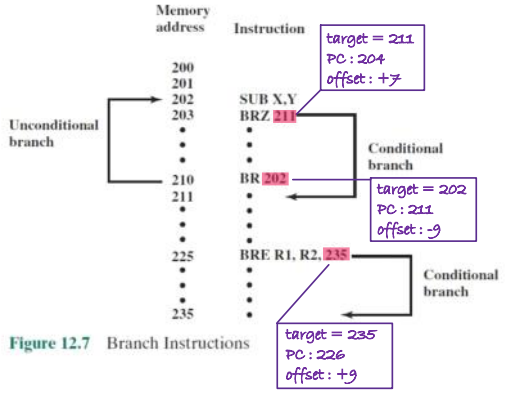
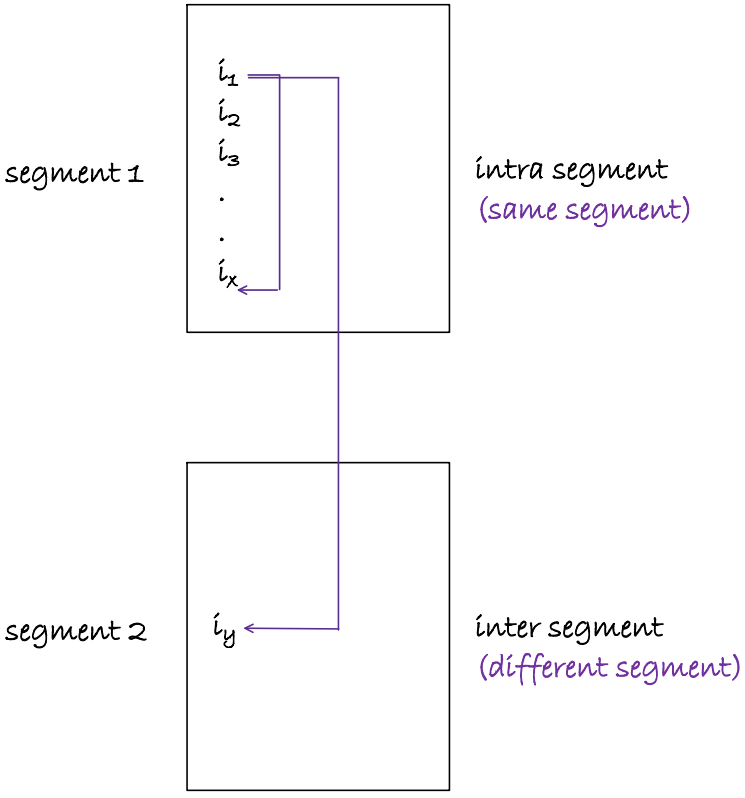


Figure 12.7 Branch Instructions

PC relative addressing mode : intra segment transfer of control (branching) when target address is present in same segment then during program execution control will be transferred with in the segment called intra segment branching.



8086 processor has 1mb memory is divided into 16 segment each 64kb

16 x 64kb
$2^4 \times 2^{16} = 2^{20}$ B
1mb

in PC relative addressing mode
target address = current PC value + displacement / offset / relative value

in base addressing mode
target address = base register value + offset / displacement

(ii) base register addressing mode : the only difference is that in based register we have to put the starting address in base register (due to re-allocation)

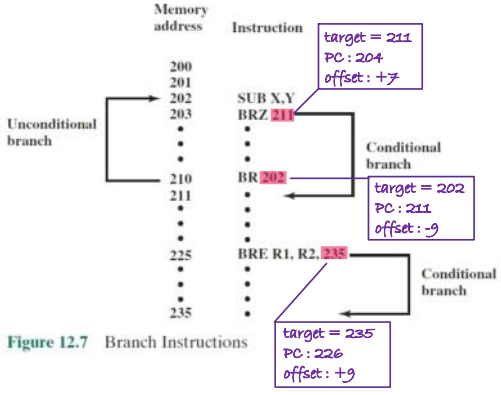


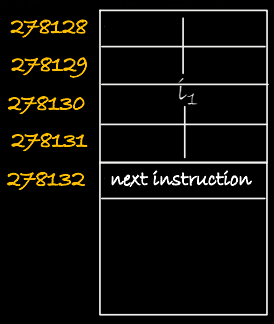
Figure 12.7 Branch Instructions

the only difference is that in based register we have to put the starting address in Base register

ex. BR = 200

#Q. Consider a 4 Byte long pc relative instruction is located in the memory with the starting address of 278128(Decimal). A -8 sign Displacement is present in the address field of the instruction . What is the branch address(Target address)?

starting address = 278128
instruction size = 4byte.
relative value = -8
target address = current pc value + displacement
target address = 27132 + (-8)
target address = 27124



#Q. Consider a 4 Byte long Jump instruction stored in the word addressable memory with a word size of 16bits. The starting address of instruction is 900(Decimal). A address field contain ---32. content of base register is 500. What is the branch address(Target address) when the instruction is designed with

(i) PC Relative Addressing Modes

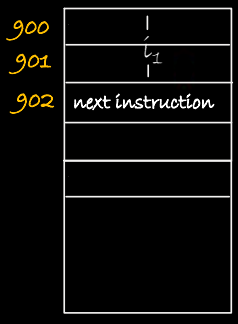
(i) Base Register Addressing Modes

(i) PC Relative Addressing Modes

word size = 16 bits = 2 byte
instruction size = 4byte = 2 words
PC value = 902
displacement value = -32

jmp	-32

target address = current pc value + displacement
target address = 902 + (-32)
target address = 870



(i) Base Register Addressing Modes

target address = base register value + displacement
target address = 500 + (-32)
target address = 468

#Q. Consider a 16bits which support 1 word long instruction, stored in the memory with a starting address of 900(Decimal). Instruction format contain 8bit Opcode & Address field. Instruction is designed with PC Relative JMP Operation.

During its execution control(Branch) will be transferred to an address 614(Decimal) then What is

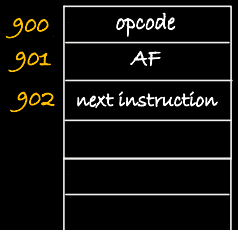
(i) What is the Relative Value in Address field of the instruction?

(ii) What is the PC Value before instruction fetch, after instruction fetch and after Execution phase?

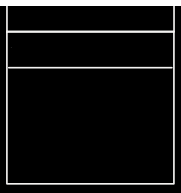
1 word size = 16 bit
instruction size = 16bit

opcode	af
8bit	8bit

PC value = 902
target address = 614



8bit 8bit



PC value = 902
target address = 614

(i) What is the Relative Value in Address field of the instruction?

target address = current PC value + displacement
614 = 902 + displacement
displacement = 614 - 902 = -288
backward jumping.

(ii) What is the PC Value before instruction fetch, after instruction fetch and after Execution phase?

PC value
(i) before instruction fetch :
900

(ii) after instruction fetch
902

(iii) after instruction fetch
614

assume starts from 1000

#Q. Consider a RISC machine where each instruction is exactly 4 bytes long. Conditional and unconditional branch instructions use PC-relative addressing mode with Offset specified in bytes to the target location of the branch instruction. Further the Offset is always with respect to the address of the next instruction in the program sequence. Consider the following instruction sequence

	Instr. No	Instruction	
1000 - 1003	i	add R2, R3, R4	
1004 - 1007	i +1	sub R5, R6, R7	
1008 - 1011	i + 2	cmp R1, R9, R10	compare
1012 - 1015	i + 3	beq R1, Offset	branch if equal
1016 - 1019			

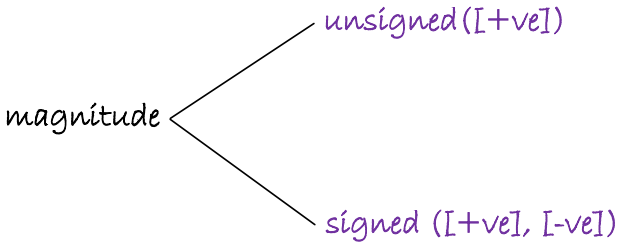
If the target of the branch instruction is i, then the decimal value of the Offset is ____.

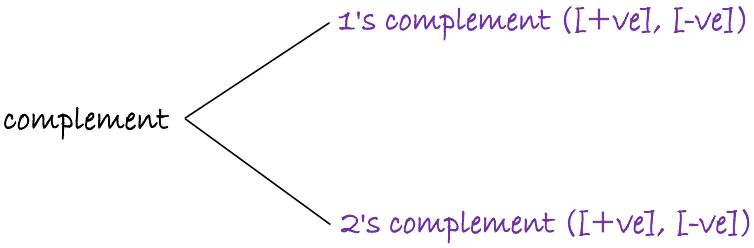
when i + 3 only fetch then
PC value becomes 1016
target address = 1000

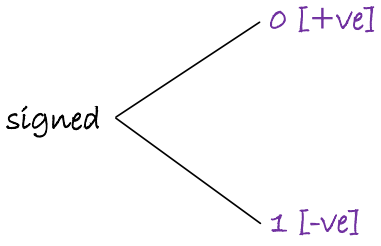
target address = current PC value + displacement
1000 = 1016 + displacement
displacement = 1000 - 1016 = -16
backward jumping.

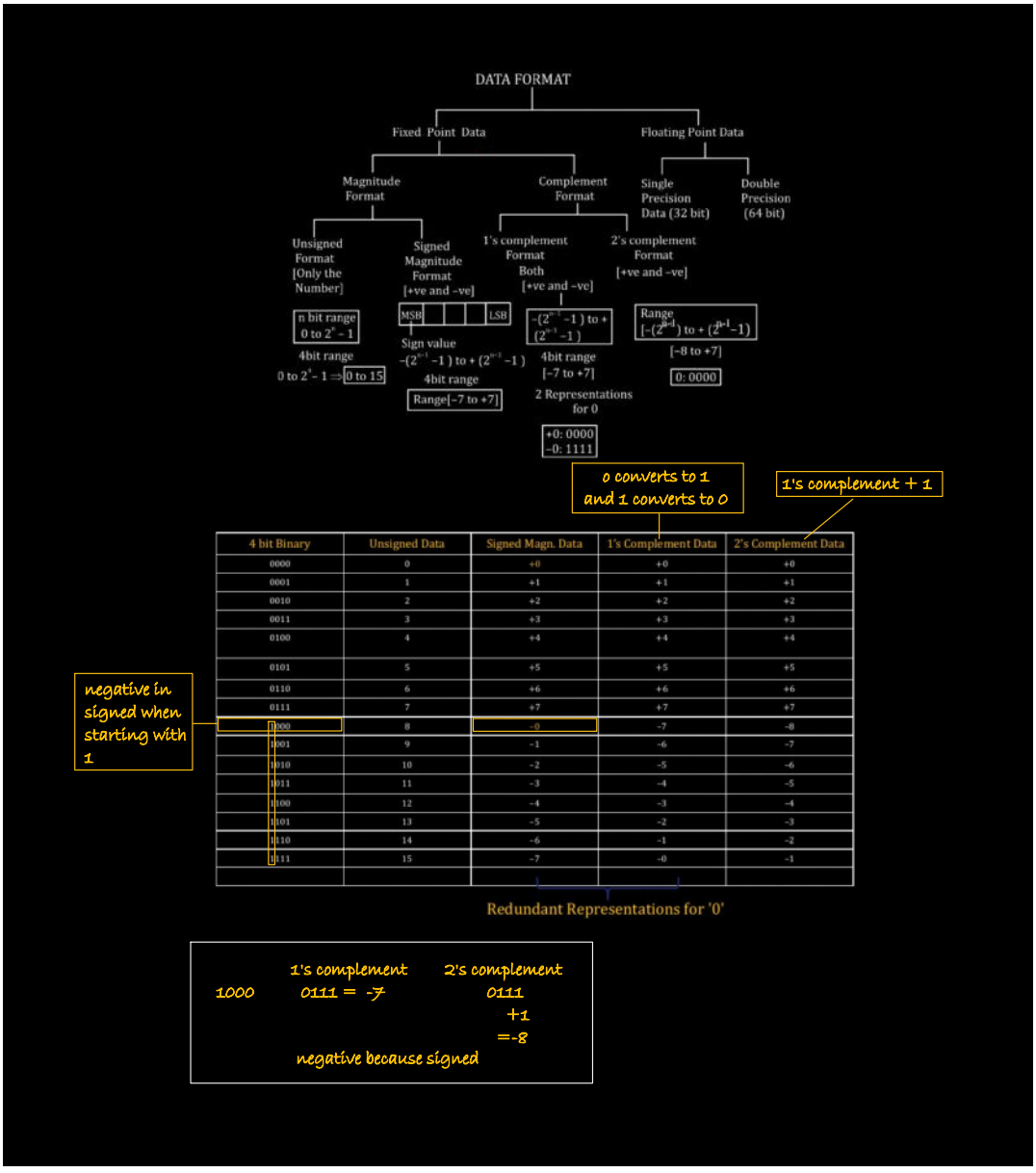
floating point representation

topic : number system









why 2's complement used in computer system?

signed	+0 (0000)	-0 (1000)
1's complement	+0 (0000)	-0 (1111)

2's complement mai 0 ke redundant representation nahi hote hai

redundant reresentation for '0'.
we use 2's complement to represent -ve numbers.

nbit 2's complement range = $-(2^{n-1})$ to $+(2^{n-1}-1)$

2^6	2^5	2^4	2^3	2^2	2^1	2^0 .	2^{-1}	2^{-2}	2^{-3}	2^{-4}
64	32	16	8	4	2	1	1/2	1/4	1/8	1/16
64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625

Q. how to write 2.5 in binary?

10.1

2^1 2^0 . 2^{-1}(1/2)
1 0 . 1

Q. how to write 3.25 in binary?

11.01

2^1 2^1 . (1/2) (1/4)
1 1 . 0 1

Q. how to write 11.25 in binary?

1011.01

Q. how to write 13.75 in binary?

1101.11

Q. how to write 6 in binary?

110

4 2 1
1 1 0

4bit : 0110
5bit : 00110
6bit : 000110

topic : floating point reprsentation

why we use floating point representation?

(i) to represent very very large number (e.g. 9876556527....)

(ii) to represent very very small number (e.g. 0.00000000000000001)

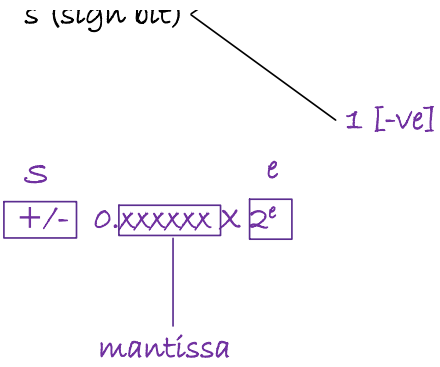
eg.
16bit number = $-(2^{16-1})$ to $+(2^{16-1})$
= (-32k to +32k -1)

but if we want to store 51,000 then it is not possible with 16bit data because range is between (-32k to +32k -1), hence this is not possible with fixed point.

syntax :

1bit	2bit	4bit
S	E	M

s (sign bit) → 0 [+ve]



E/BE= exponent / biased exponent

E= e + bias

(or)

BE= AE + bias

M= mantissa

e=exponent / actual exponent AE

Q. 1 +6.5 write in SEM format.

1bit	2bit	4bit
S	E	M

note : hume jo bhi floating point mai represent karna hota hai vo hamesha isi format mai karna hota hai

+/- 0.xxxxxx X 2^e

110.1

$0.1101 \times 2^{+3}$

(i) method to cross check

Q. is e=+3 correct or not?

$[0.1101] \times 2^{+3}$
$= [2^{-1}+2^{-2}+2^{-4}] \times 2^{+3}$
$= [2^{-1+3} + 2^{-2+3}+2^{-4+3}]$
$= 2^2 + 2^1 + 2^{-1}$
$= 4 + 2 + 0.5$
$= 6.5$

(ii) second technique
right alignment = 2^+ (power will be in positive)
left alignment = 2^- (power will be in negative)

110.1

point udhar hi rahega

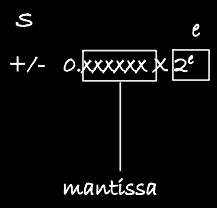
$0.1101 \times 2^{+3}$
three bits right mai sarka di means right alignment

point karna hi padega

$0.1101 \times 2^{+3}$

three bits right mai sarka di means right alignment
hence power is in positive.

$0.1101 \times 2^{+3}$



s(sign) = 0 (positive hai)
e = +3 [11]
M = 1101 (mantissa)

s(1bit)	e(2bit)	m(4bit)
0	11	1101

right alignment = 2^{+} (power will be in positive/increment in exponent)
left alignment = 2^{-} (power will be in negative/decrement in exponent)

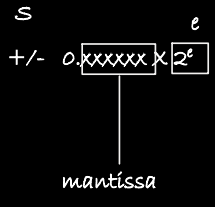
Q +(4.5)

1bit	2bit	4bit
S	E	M

100.1

$0.1001 \times 2^{+3}$

$0.1001 \times 2^{+3}$



s(sign) = 0 (positive hai)
e = +3 [11]
M = 1001 (mantissa)

s(1bit)	e(2bit)	m(4bit)
0	11	1001

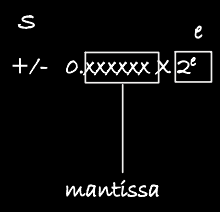
Q +(4.75)

1bit	2bit	5bit
S	E	M

100.11

$0.10011 \times 2^{+4}$

$0.10011 \times 2^{+3}$



s(sign) = 0 (positive hai)
e = +3 [11]
M = 10011 (mantissa)

s(1bit)	e(2bit)	m(5bit)
0	11	10011

if M was 4 bit then

note : format diya rahega question mai

1bit	2bit	4bit
S	E	M

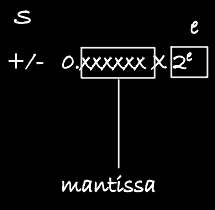
s(1bit)	e(2bit)	m(4bit)
0	11	1001

Q 0.00101

1bit	4bit	5bit
S	E	M

0.101×2^{-2}

0.101×2^{-2}



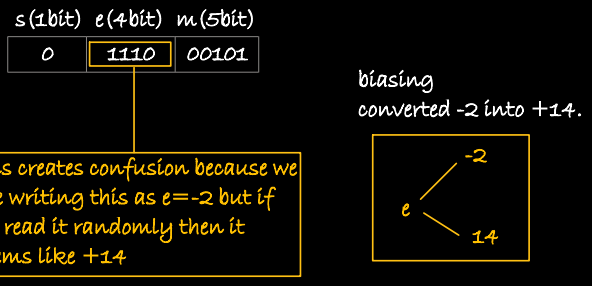
s(sign) = 0 (positive hai)

e = -2 [10] (no provision to represent (tell) exponent is negative)

s(sign) = 0 (positive hai)

e = -2 [10] (no provision to represent (tell) exponent is negative)
exponent is negative but sign bit is 0 because number is positive so how to deal with negative exponent?
hence, we have to take 2's complement.
e = 0010 (in 4bits)
1's complement : 1101
2' complement : 1101 + 1 = 14 (1110)

M = 101 (mantissa)



q.(i) what is bias exponent?

E

q.(ii) why biasing (or) bias exponent [E/BE] is used?

to convert negative/positive number into positive or '0'.
when e is -ve then we take 2's complement
nbit 2's complement range = $-(2^{n-1})$ to $+(2^{n-1}-1)$
5bit 2's complement range = $-(2^{5-1})$ to $+(2^{5-1}-1)$ = -16 to +15

q.(iii) how bias value is decided/selected?

if your exponent = k-bit
then bias value = 2^{k-1}

if exponent is kbit then 2's complement range = -2^{k-1} to $2^{k-1}-1$
if exponent is 4bit then 2's complement range = -2^{4-1} to $2^{4-1}-1$ = -8 to +7

in order to convert all numbers into positive numbers, take the most (highest) negative number and add as a bias

if exponent is kbit then bias = -2^{k-1}
if exponent is kbit then bias = $-2^{4-1} = 2^3 = 8$
if exponent is 4bit then 2's complement range = -2^{4-1} to $2^{4-1}-1$ = -8 to +7

e + bias = E
-8 + 8 = 0
-7 + 8 = 1
-6 + 8 = 2
-5 + 8 = 3
-4 + 8 = 4
-3 + 8 = 5
-2 + 8 = 6
-1 + 8 = 7
0
.
.
.
7 + 8 = 15

number kuch bhi ho we use bias exponent toh E ke lie kuch nahi sochna sab 0 hoga ya positive hoga

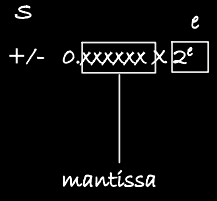
note : sometimes bias is given in this form
bias = 8
exponent = 4bit

Excess 8 code
$2^{K-1} = 8$
$2^{K-1} = 23$
K - 1 = 3
K = 4
E = 4 bit

Q. 0.00101

0.101×2^{-2}

0.101×2^{-2}



s(sign) = 0 (positive hai)

e = -2 [10] (no provision to represent (tell) exponent is negative)
exponent is negative but sign bit is 0 because number is positive so how to deal with negative exponent?
hence, we have to take 2's complement.
e = 0010 (in 4bits)
1's complement : 1101
2' complement : 1101 + 1 = 14 (1110)

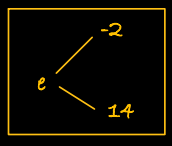
M = 101 (mantissa)

M = 101 (mantissa)

s(1bit)	e(5bit)	m(4bit)
0	01110	1010

this creates confusion because we are writing this as e=-2 but if we read it randomly then it seems like +14

biasing
converted -2 into +14.



continuation....

1bit	5bit	4bit
S	E	M

s(sign) = 0 (positive hai)
e = -2 [10]
M = 101 (mantissa)
E= e + bias

exponent : 5bit
hence , bias = 2^{5-1}
bias = $2^{5-1} = 2^4 = 16$

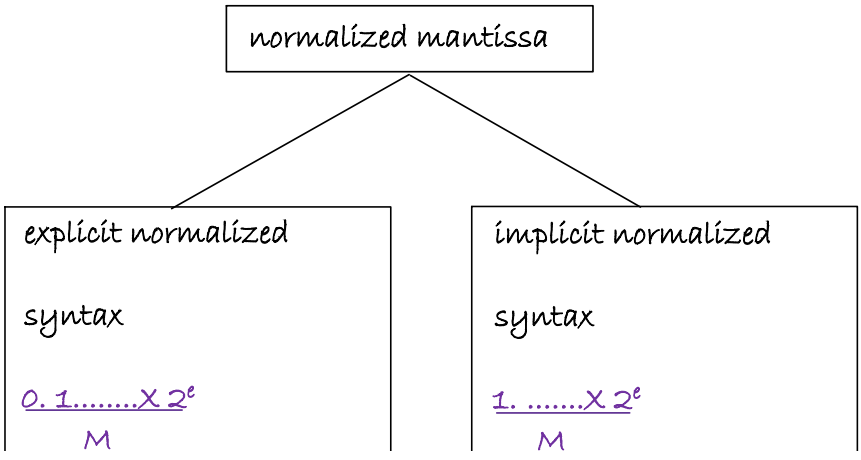
E= e + bias
E= -2 + 16 = 14

in 5 bits : 01110

1bit	5bit	4bit
S	01110	1010

q.(iv) how to write mantissa

we have to normalize the mantissa



$0.1\ldots\ldots \times 2^{e}$

M

point ke baad immediate bit 1 honi chaiye

formula to get number
[value formula]

$(-1)^{s} \times 0.M \times 2^{e}$

$(-1)^{s} \times 0.M \times 2^{E\text{-bias}}$

example :
(101.11)

$0.10111 \times 2^{+3}$

M=10111
e=3
E = e+bias
e = E - bias

$1.\ldots\ldots \times 2^{e}$

M

point ke pehle 1.
1.something

formula to get number
[value formula]

$(-1)^{s} \times 1.M \times 2^{e}$

$(-1)^{s} \times 1.M \times 2^{E\text{-bias}}$

example :
(101.11)

1.0111×2^{2}

M=0111
e=2
E = e+bias
e = E - bias

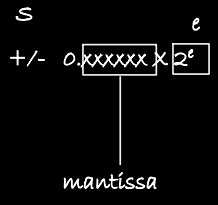
Q +6.75

1bit	4bit	5bit
S	E	M

110.11

exponent : 4bit
hence, bias = 2^{4-1}
bias = $2^{4-1} = 2^{3} = 8$

E= e + bias
E= e + 8



explicit	implicit
$+110.11 \times 2^{0}$	$+110.11 \times 2^{0}$
0.11011×2^{3}	1.1011×2^{2}
s(sign) = 0 (positive hai)	s(sign) = 0 (positive hai)
e = 3	e = 2
M = 11011	M = 1011
E= e + bias	E= e + bias
E = 3 + 8 = 11 (1011)	E = 2 + 8 = 10 (1010)

1bit	4bit	5bit
0	1011	11011

1bit	4bit	5bit
0	1010	10110

1bit	4bit	5bit
0	1011	11011

1bit	4bit	5bit
0	1010	10110

5 bits ke lie maine mantissa mai 0 lagaya but last mai zero kyu lagaya?

explicit	implicit
1bit 4bit 5bit 0 1011 11011 E = 1011 = 11 bias = 8 formula to get number [value formula] $(-1)^s \times 0.M \times 2^e$ $(-1)^s \times 0.M \times 2^{E-bias}$ $(-1)^0 \times 0.11011 \times 2^{11-8}$ $(-1)^0 \times 0.11011 \times 2^3$ 0.11011×2^3 110.11×2^0 6.75	1bit 4bit 5bit 0 1010 10110 E = 1010 = 10 bias = 8 formula to get number [value formula] $(-1)^s \times 1.M \times 2^e$ $(-1)^s \times 1.M \times 2^{E-bias}$ $(-1)^0 \times 1.10110 \times 2^{10-8}$ $(-1)^0 \times 1.10110 \times 2^2$ 1.10110×2^2 110.110×2^0 6.75 0 lagane se farq nahi pada last mein, result same aya.

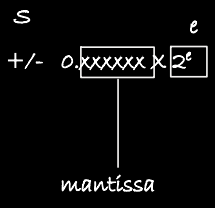
Q. +5.5

1bit	4bit	5bit
S	E	M

101.1

exponent : 4bit
hence , bias = 2^{4-1}
bias = $2^{4-1} = 2^3 = 8$

E = e + bias
E = e + 8



explicit	implicit
$+101.1 \times 2^0$ 0.1011×2^3 s(sign) = 0 (positive hai) e = 3 M = 1011 E= e + bias E = 3 + 8 = 11 (1011) 1bit 4bit 5bit 0 | 1011 | 10110 hexa : (1 B 6)$_{16}$	$+101.1 \times 2^0$ 1.011×2^2 s(sign) = 0 (positive hai) e = 2 M = 011 E= e + bias E = 2 + 8 = 10 (1010) 1bit 4bit 5bit 0 | 1010 | 01100 (1 4 C)$_{16}$

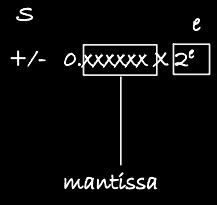
Q +4.875

1bit	4bit	5bit
S	E	M

100.111

exponent : 4bit
hence , bias = 2^{4-1}
bias = $2^{4-1} = 2^3 = 8$

E= e + bias
E= e + 8



explicit	implicit
$+100.111 \times 2^0$ 0.100111×2^3 s(sign) = 0 (positive hai) e = 3 M = 100111 E= e + bias E = 3 + 8 = 11 (1011) 1bit 4bit 5bit 0 | 1011 | 10011	$+100.111 \times 2^0$ 1.00111×2^2 s(sign) = 0 (positive hai) e = 2 M = 00111 E= e + bias E = 2 + 8 = 10 (1010) 1bit 4bit 5bit 0 | 1010 | 00111

5 bits ke lie maine mantissa mai se ek 1 hata diya

explicit	implicit
1bit 4bit 5bit 0 | 1011 | 10011 E = 1011 = 11 bias = 8	1bit 4bit 5bit 0 | 1010 | 00111 E = 1010 = 10 bias = 8

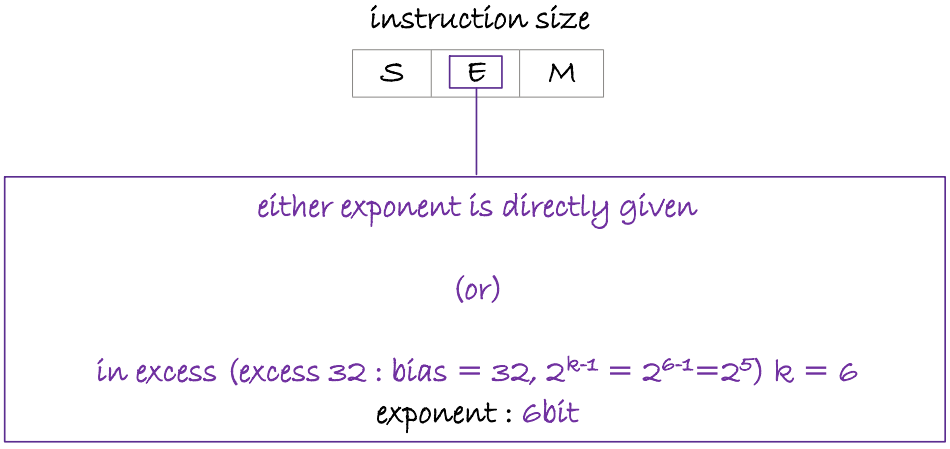
E = 1011 = 11 bias = 8 formula to get number [value formula] $(-1)^s \times 0.M \times 2^e$ $(-1)^s \times 0.M \times 2^{E-bias}$ $(-1)^0 \times 0.10011 \times 2^{11-8}$ $(-1)^0 \times 0.10011 \times 2^3$ 0.10011×2^3 100.11×2^0 4.75	E = 1010 = 10 bias = 8 formula to get number [value formula] $(-1)^s \times 1.M \times 2^e$ $(-1)^s \times 1.M \times 2^{E-bias}$ $(-1)^0 \times 1.00111 \times 2^{10-8}$ $(-1)^0 \times 1.00111 \times 2^2$ 1.00111×2^2 100.111×2^0 4.875

1 hatane se result inaccurate aya hai

either increase the bits in mantissa
(or) use implicit normalization.

mantissa : giving precision (accuracy) (large number / more and more bits) (in mantissa getting accurate for very small number)

exponent : giving range (power) (more bits in exponent means large-large number)

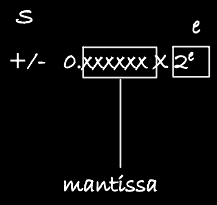


Q. Consider a 16 bit register used to store floating point number. Mantissa is Explicit normalized signed fraction number. Exponent is in Excess-32 form then what is 16-bit for $-(29.75)_{10}$ in the register?

1bit	6bit	9 bit
S	E	M

$-(29.75)_{10}$

11101.11×2^0



excess : 32
bias : $2^{k-1} = 2^{6-1} = 2^5 = 32$
k= 6bit
exponent : 5

E= e + bias
E= e + 32
explicit normalized :

11101.11×2^0
0.1110111×2^5

e = 5
E= 5 + 32
E= 37 (100101)
M= 1110111

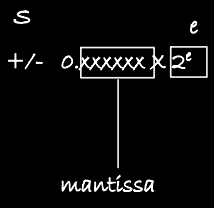
1bit	6bit	9 bit
1	100101	111011100
	4 B	D C

Q 21.75

1bit	7bit	8 bit
S	E	M

21.75

10101.11×2^0



bias : $2^{7-1} = 2^{7-1} = 2^6 = 64$
k= 7 bit

E= e + bias
E= e + 64

explicit	implicit
$+10101.11 \times 2^0$ 0.1010111×2^5 s(sign) = 0 (positive hai)	$+10101.11 \times 2^0$ 1.010111×2^4 s(sign) = 0 (positive hai)

0.1010111×2^5	1.010111×2^4
s(sign) = 0 (positive hai) e = 5 M = 1010111 E= e + bias E = 5 + 64 = 69 (1000101)	s(sign) = 0 (positive hai) e = 4 M = 010111 E= e + bias E = 4 + 64 = 68 (1000100)

1bit	7bit	8bit
0	1000101	1010111

1bit	7bit	8bit
0	1000100	01011100

#Q. Consider a 16 bit register used to store floating point number. Mantissa is normalized signed fraction number. Exponent is in Excess-32 form then what is 16-bit for $+(13.5)_{10}$ in the register?(Using Explicit & Implicit)

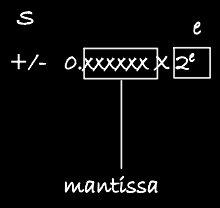
1bit	6bit	9 bit
S	E	M

13.5

1101.1×2^0

excess : 32
bias : $2^{6-1} = 2^{6-1} = 2^5 = 32$
k= 6 bit

E= e + bias
E= e + 32



explicit	implicit
$+1101.1 \times 2^0$ 0.11011×2^4 s(sign) = 0 (positive hai) e = 4 M = 1101 E= e + bias E = 4 + 32 = 36 (100100)	$+1101.1 \times 2^0$ 1.1011×2^3 s(sign) = 0 (positive hai) e = 3 M = 1011 E= e + bias E = 3 + 32 = 35 (100011)

1bit	6bit	9bit
0	100100	110110000

4 9 B 0

1bit	6bit	9bit
0	100011	101100000

4 7 6 0

explicit	implicit
1bit 6bit 9bit 0 100100 110110000 E = 100100 = 36 bias = 32	1bit 6bit 9bit 0 100011 101100000 E = 100011 = 35 bias = 32

E = 100100 = 36
bias = 32

formula to get number
[value formula]
$(-1)^s \times 0.M \times 2^e$
$(-1)^s \times 0.M \times 2^{E-bias}$

$(-1)^0 \times 0.110110000 \times 2^{36-32}$
$(-1)^0 \times 0.110110000 \times 2^4$
0.110110000×2^4
1101.10000×2^0
13.5

E = 100011 = 35
bias = 32

formula to get number
[value formula]
$(-1)^s \times 1.M \times 2^e$
$(-1)^s \times 1.M \times 2^{E-bias}$

$(-1)^0 \times 1.101100000 \times 2^{35-32}$
$(-1)^0 \times 1.101100000 \times 2^3$
1.101100000×2^3
1101.100000×2^0
13.5

#Q. Consider a 16 bit register used to store floating point number. Mantissa is Implicit normalized signed fraction number. Exponent is in Excess-64 form then
(i) what is the First Smallest Positive number?
(ii) what is the Second Smallest Positive number?
(iii) what is the Difference between First Smallest & Second Smallest Positive number?

1bit	7bit	8 bit
S	E	M

excess : 64
bias : $2^{k-1} = 2^{7-1} = 2^6 = 64$
k= 7 bit

E= e + bias
E= e + 64

implicit
(i) smallest mantissa : 0000 0000
(ii) largest / highest mantissa : 1111 1111
(iii) smallest exponent : 0000 0000
(minimum value in exponent)
(iv) large / highest exponent : 1111 111
(maximum value in exponent)

explicit
(i) smallest mantissa : 1000 0000 (.M point ke baad mantissa)
(ii) largest / highest mantissa : 1111 1111
(iii) smallest exponent : 0000 0000
(minimum value in exponent)
(iv) large / highest exponent : 1111 111
(maximum value in exponent)

note : in the implicit we cannot represent '0' in either
(i) implicit : 1.something (1.xxxxx) [(1.0) karne par bhi its not zero]

note : in the implicit we cannot represent '0' in either
(i) implicit : 1.something (1.xxxxx) [(1.0) karne par bhi its not zero]
(ii) explicit : 0.1 (0.1xxxxxx) [(0.1) karne par bhi its not zero]
actually hum 0 nahi likhte.

note :
(i) in the explicit represent we cannot represent 0 because 0.1 something is not 0
(ii) in the implicit representation we cannot represent '0' because 1.something is not 0
but here for first smallest we put all 0's in E and mantissa but actually value is not zero.

so for that '0' we use IEEE 754 single precision and double precision

(i) first smallest positive number :

	E (7bit)	M (8bit)
0	0000000	00000000

bias : $2^{7-1} = 2^6 = 64$

implicit :

$(-1)^S 1.00000000 \times 2^{E-Bias}$

$(-1)^0 1.00000000 \times 2^{0-64}$

$+1.00000000 \times 2^{-64}$

first smallest positive number : $+1.00000000 \times 2^{-64}$

(ii) second smallest positive number :

	E (7bit)	M (8bit)
0	0000000	00000001

second smallest positive mai idhar se 1 aajaygea

bias : $2^{7-1} = 2^6 = 64$

implicit :

$(-1)^S\ 1.00000001 \times 2^{E-Bias}$

$(-1)^0\ 1.00000001 \times 2^{0-64}$

$+1.00000001 \times 2^{-64}$

second smallest positive number : $+1.00000001 \times 2^{-64}$

(iii) difference between 1st smallest and 2nd smallest

$+1.00000001 \times 2^{-64} \quad - \quad +1.00000000 \times 2^{-64}$

$[1.00000001 - 1.00000000] \times 2^{-64}$

$[0.00000001] \times 2^{-64}$

$2^{-8} \times 2^{-64}$

2^{-72}

	E(7bit)	M(8bit)
0	0000000	00000000

#Q. Consider a 16 bit register used to store floating point number. Mantissa is Implicit normalized signed fraction number. Exponent is in Excess-64 form then
(i) what is the First Highest Positive number?
(ii) what is the Second Highest Positive number?
(iii) what is the Difference between First Highest & Second Highest Positive number?

(i) first highest positive number :

	E(7bit)	M(8bit)
0	1111 111	1111 1111

bias : $2^{7-1} = 2^6 = 64$

E : 127

implicit :

$(-1)^s 1.11111111 \times 2^{E-Bias}$

$(-1)^0 1.11111111 \times 2^{127-64}$

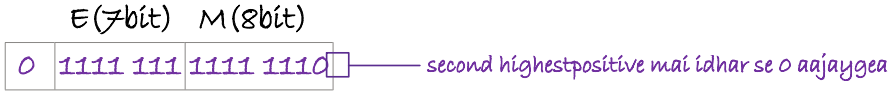
$+1.11111111 \times 2^{63}$

$[111111111] \times 2^{-8} \times 2^{63}$ (8bits shifted to left)

$2^9-1 \times 2^{-8} \times 2^{63}$

first highest positive number : $+1.11111111 \times 2^{63}$

(ii) second highest positive number :



bias : $2^{7-1} = 2^6 = 64$

implicit :

$(-1)^s 1.1111111 \times 2^{E-Bias}$

$(-1)^0 1.11111110 \times 2^{127-64}$

$+1.11111110 \times 2^{63}$

second highest positive number : $+1.11111110 \times 2^{63}$

(iii) difference between 1st highest and 2nd highest

$+1.11111110 \times 2^{63} \quad - \quad +1.11111111 \times 2^{63}$

$[1.111111110 - 1.11111111] \times 2^{63}$

$[111111111] \times 2^{-8} \times 2^{63}$

$2^{9}-1 \times 2^{-8} \times 2^{63}$

2^{72}

how to deal with bits?

(i) 111 ⟶ $2^{3}-1$

(ii) 1111 ⟶ $2^{4}-1$

(iii) 11111 ⟶ $2^{5}-1$

(iv) 111111 ⟶ $2^{6}-1$

(i) 0.111 ⟶ $1- 1/2^{3}$ (or) $1-2^{-3}$

(ii) 0.1111 ⟶ $1- 2/2^{4}$ (or) $1-2^{-4}$

(iii) 0.11111 ⟶ $1- 1/2^{5}$ (or) $1-2^{-5}$

(iv) 0.111111 ⟶ $1- 1/2^{6}$ (or) $1-2^{-6}$

proof :

(i) 0.111 ⟶ $1- 1/2^{3}$ (or) $1-2^{-3}$

proof : 0.111×2^{0}

111×2^{-3} (left alignment)

$(2^{3}-1) \times 2^{-3}$

$2^{3-3} - 1 \times 2^{-3}$

$1-2^{-3}$

maximum +ve value using explicit :

1bit	6bit	9bit
S	E	M

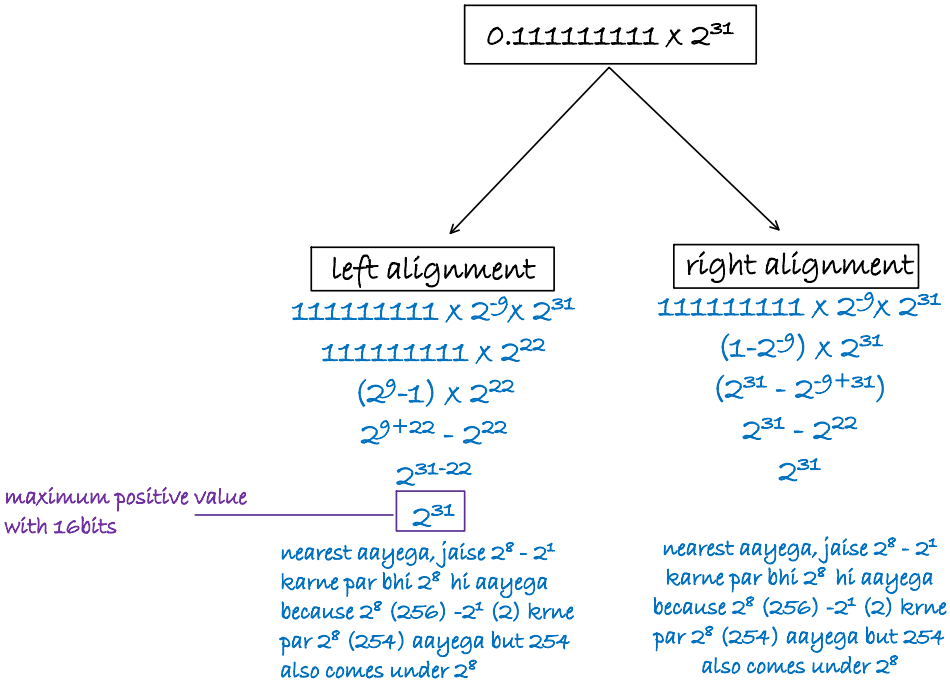
0	111111	111111111

bias : $2^{6-1}=32$

explicit : $(-1)^s\ 0.M \times 2^{e-bias}$

$(-1)^s\ 0.111111111 \times 2^{63-32}$

$0.111111111 \times 2^{31}$



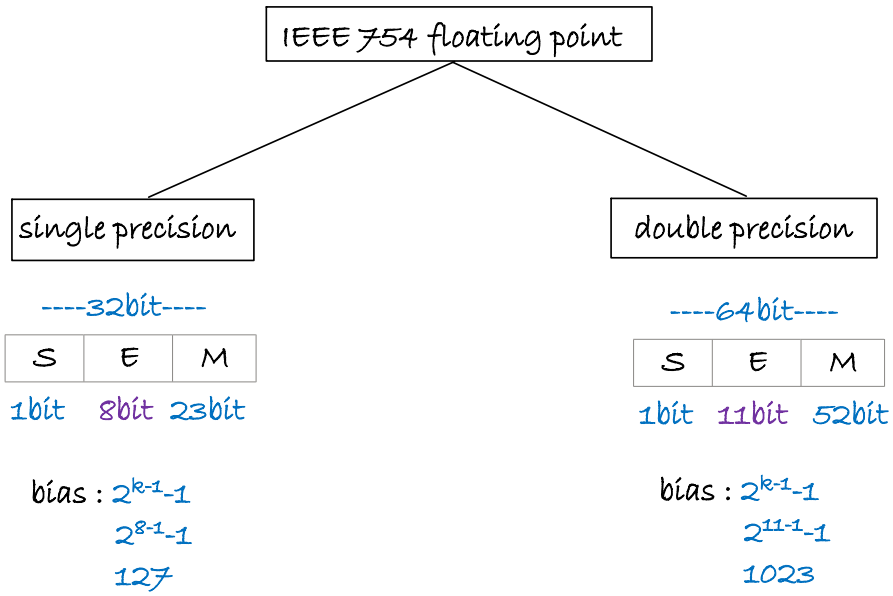
Disadvantage of convential floating point representation :

(i) it cannot represent '0'

(ii) it cannot represent infinity '∞'

(iii) it cannot represent the number which is not normalized.

that is why we use IEEE 754 floating point.



note : default we use implicit in IEEE because computer mai 1.M ki form mai likha jata hai aur accuracy jada aati hai

#Q. How to represent +(119) into IEEE 754 single precision & Double precision floating Point Representation?

single precision

default : implicit

119 : 1110111.0×2^0

1.110111×2^6 (right shift)

S : 0

M : 11011

e: 6

bias : 127

E : 6 + 127 = 133

E : 10000101

1bit	E(8bit)	mantissa (23bit)
0	10000101	11011100000000000000000

hexa : 42EE0000

implicit

E = 133
bias = 127

$(-1)^s 1.M \times 2^{E-bias}$

$(-1)^0 1.11011100000000000000000 \times 2^{133-127}$

$1.11011100000000000000000 \times 2^6$

+1110111.00000000000000000

+119

double precision

+119

double precision

default : implicit
119 : 1110111.0 x 2^0

1.110111 x 2^6 (right shift)
S : 0
M : 11011
e: 6
bias : $2^{11}-1$: 1023
E : 6 + 1023 = 1029
E : 10000101

1bit	E(11bit)	mantissa (52bit)
0	10000000101	1101110000...00 (0-46times)

hexa : 405DC00000000000

implicit

$(-1)^s$ 1.M x 2^{E-bias}

$(-1)^0$ 1.1101110...46times x $2^{1029-1023}$
1.1101110...46timesx 2^6
+1110111.00000000000000000
+119

note : how to write 1029 in binary

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1024	512	256	128	64	32	16	8	4	2	1
1								1		1
	0	0	0	0	0	0	0		0	

1000000101

#Q. Consider a 32 bit register which stores floating numbers in IEEE single precision format(Implicit). The value of the number, f 32 bit are given below is _____

Sign(1bit)	Exponent(8bit)	Mantissa(23bit)
0	10000100	11000000000000000000000

A 48
B 56 ✓
C 64
D 60

single precision

default : implicit

S : 0
M : 11000000000000000000000
e: 5
bias : 127
E : 5 + 127 = 132 (10000100)
E : 10000100

1bit	E(8bit)	mantissa (23bit)
0	10000100	11000000000000000000000

hexa : 42600000

implicit

$(-1)^S 1.M \times 2^{E-bias}$

$(-1)^0 1.11000000000000000000000 \times 2^{132-127}$
$1.11000000000000000000000 \times 2^5$
+111000.00000000000000000
+56

#Q. The value of float type variable is represented using the single precision 32 bit floating point IEEE 754 standard use 1 bit sign, 8bit for E and 23 bit mantissa. A float type variable X is assigned the decimal value (–14.25). Then representation of X in Hexa decimal notation is?

A 416C0000H
B 41640000H
C C16C0000H
D C1640000H

single precision

default : implicit
-14.25: -1110.01 x 2^0

-1.11001x 2^3 (right shift)
S : 0
M : 11001
e: 3
bias : 127
E : + 127 = 130
E : 10000010

1bit	E(8bit)	mantissa (23bit)
1	10000010	11001000000000000000000

hexa : C1640000

implicit

$(-1)^S 1.M \times 2^{E-bias}$

implicit

$(-1)^s 1.M \times 2^{E-bias}$

$(-1)^1 1.11001000000000000000000 \times 2^{130-127}$
$-.11001000000000000000000 \times 2^3$
-1110.0100000000000000000
-14.25

1bit	2bit	4bit
S	E	M

in convential representation
bias = 2^{k-1}

but in IEEE 754 floating point representation
bias = $2^{k-1}-1$

single precision = $2^{8-1}-1$
bias = 127

double precision = 2^{11-1}
bias = 1023

why bias = $2^{k-1}-1$?

in single precision

1bit	8bit	23bit
S	E	M

E = 00000000 M = 00000000000000000000000

E is reserved in IEEE for all 0 and all 1

S	E	M
1 bit	8 bit	23 bit

Sign(1 bit)	E(1 bit)	M(23 bit)	Value
0 or 1	00000000 E = 0	00000000000000000000000 M = 0	±0
0 or 1	11111111 E = 255	00000000000000000000000 M = 0	±∞
0 or 1	1 ≤ E ≤ 254	M = ...	Implicit Normalized form $(-1)^S \times 1.M \times 2^e$ $(-1)^S \times 1.M \times 2^{E-127 \text{ bias}}$
0 or 1	E = 0	M ≠ 0	Denormalized number/Fractional form $(-1)^S \times 0.M \times 2^{E-127 \text{ bias}}$
0 or 1	E = 255	M ≠ 0	Not a Number (NAN)

dono ko represent kar sakte hai aur denormalized ko bhi

NAN ko bhi represent kar sakte hai

somethings are fixed in IEEE :

when E = 0
E = 00000000
- M = 0 (+/- 0)
- M ≠ 0 (fraction / denormalized)

when E = 255
E = 11111111
- M = 0 (infinte)
- M ≠ 0 (NAN, not a number)

in double precision

1bit	11bit	52bit

S	E	M

E = 00000000000 M = 0000000000
0000000000

E is reserved in IEEE for all 0 and all 1

Sign (1 bit)	E(11 bit)	M(52 bit)	Value
0 or 1	0000 0000 000 E = 0	00000000000.. M = 0	± 0
0 or 1	1111 1111 111 E = 2047	0000000000.. M = 0	± ∞
0 or 1	1 ≤ E ≤ 2046	M = _____	Implicit Normalization $(-1)^S \cdot M \times 2^e$ $(-1)^S \times 1 \cdot M \times 2^{E-1023}$
0 or 1	E = 0	M ≠ 0	Denormalized number/ Fractional Form $(-1)^S 0 \cdot M \times 2^{E-1023}$
	E = 2047	M ≠ 0	Not a number

dono ko represent kar sakte hai aur denormalized ko bhi

NAN ko bhi represent kar sakte hai

when E = 0
E = 00000
000000

M = 0 (+- 0)

M ≠ 0 (fraction / denormalized)

when E = 2047
E = 111111
11111

M = 0 (infinte)

E = 111111
11111

M ≠ 0 (NAN, not a number)

why bias = $2^{k-1}-1$? in IEEE 754 floating point?

in single precision
if we take bias = $2^{k-1} = 2^{8-1} = 2^{7-1} = 128$ (or) excess 128 then there is a chance of getting E = 255

because 8bit 2's complement range = -2^{8-1} to $-2^{8-1}-1$ = -128 to 127

assume if e = 127
bias = 128

E = 127 + 128
E = 255

agar aap only 2^{k-1} loge toh 128 aayega
aur e =127 hai
then 255 aa jayega aur 255 infinite aur NAN ke lie bana hai then problem will arrive.

Note: Features IEEE 754 are special symbols to represent unusual events....

For example instead of interrupting on a divide by 0,softwarecan set the result to a bit pattern representing + ∞ ,- ∞ ; The largest exponent is reserved for these special symbols.

IEEE 754 has a symbols for the result of invalid operations, such 0/0, or subtract infinity from infinity . This symbols is NaN, for Not a Number.

The purpose of NaN is to allow programmer to postpone some test and decisions to a later time in this program. (when it is convenient)

#Q. What will ne maximum positive vale(+ve) in IEEE 754 single precision floating Point Representation?

1st highest(largest number) pucha hai

S	E(8bit)	M(23bit)
0	1111 1110	1111 1111 1111 1111 1111 111

Saare E hone par
255 aajyega
islie 0 lagaya

255 ke lie NAN aur inf ho jata hai islie hum ek kam karte hai

S=0
E = 254

255 ke lie NAN aur inf ho jata hai islie hum ek kam karte hai

S=0
E = 254
1<E<254 (for single precision)

$(-1)^s 1.M \times 2^{e\ (E\text{-}bias)}$
$(-1)^0 1.M \times 2^{254-127}$
$1.11111111111111111111111 \times 2^{127}$
$111111111111 \times 2^{-23} \times 2^{127}$
$(2^{24}-1) \times 2^{104}$
$2^{128}-2^{104}=2^{128}$

32 bit se 2^{128} tak represent kar paa raha hai

1st highest(largest number) without IEEE 754

S	E(6bit)	M(7bit)
0	111111	1111111

2st highest number without IEEE 754

S	E(6bit)	M(7bit)
0	111111	1111110

topic : denormalized number concept

single precision E = e + bias bias = 127 (in single precision)

IEEE E = 1 to 254 1≤E≤254

254 aaye aur overflow (255) na ho jaye E islie mene bias ko $2^{K-1}-1$ kar diya hai.

minimum E ka value 1 ho sakta
for E=1 laane ke lie 'e' will be?
E = e + bias
1 = e + 127
e = -126
1 ko laane ke lie e=-126 aayega, that means in worst case e =-126, if value 'e' is smaller than -126 (-127, -128..)then number is not able to normalize and we store as denormalize number.

eg. 1.1001×2^{-129}
e = -129
E = e + bias

-129 + 127 = -2
E = -2
E negative aa raha hai lekin humne E islie use kia tha ki vo hamesha positive aaye toh isse accha hume ise as a denormalize number store kar denge.

note : single precision for a normalized number in worst case will be : $1.M \times 2^{-126}$ then its normalized otherwise denormalized number.

- smallest positive single precision normalized number :

1.0000 0000 0000 0000 0000 000 x 2^{-126}
$1.M \times 2^{-126}$

- smallest single precision denormalized number :

0.0000 0000 0000 0000 0000 001 x 2^{-126}
(23 bits shifted, left alignment)
(converts into implicit)
$1.0 \times 2^{-23} \times 2^{-126}$
1.0×2^{-149}

- double precision range :

1.0×2^{-1074}

double precision

1bit	11bit	20bit
S	E	M

E = 1 bias = 1023

E = E + bias
1 = e + 1023
e = -1022

$+- 1.0 \times 2^{-1022}$
if e = -1023, 1024...then store as denormalized number

topic : floating point addition, subtraction, divison and multipication

(i) Add/subtraction rule :

lets say

(i)
$A = ...x2^{2}$ power same honi chaiye

(i)

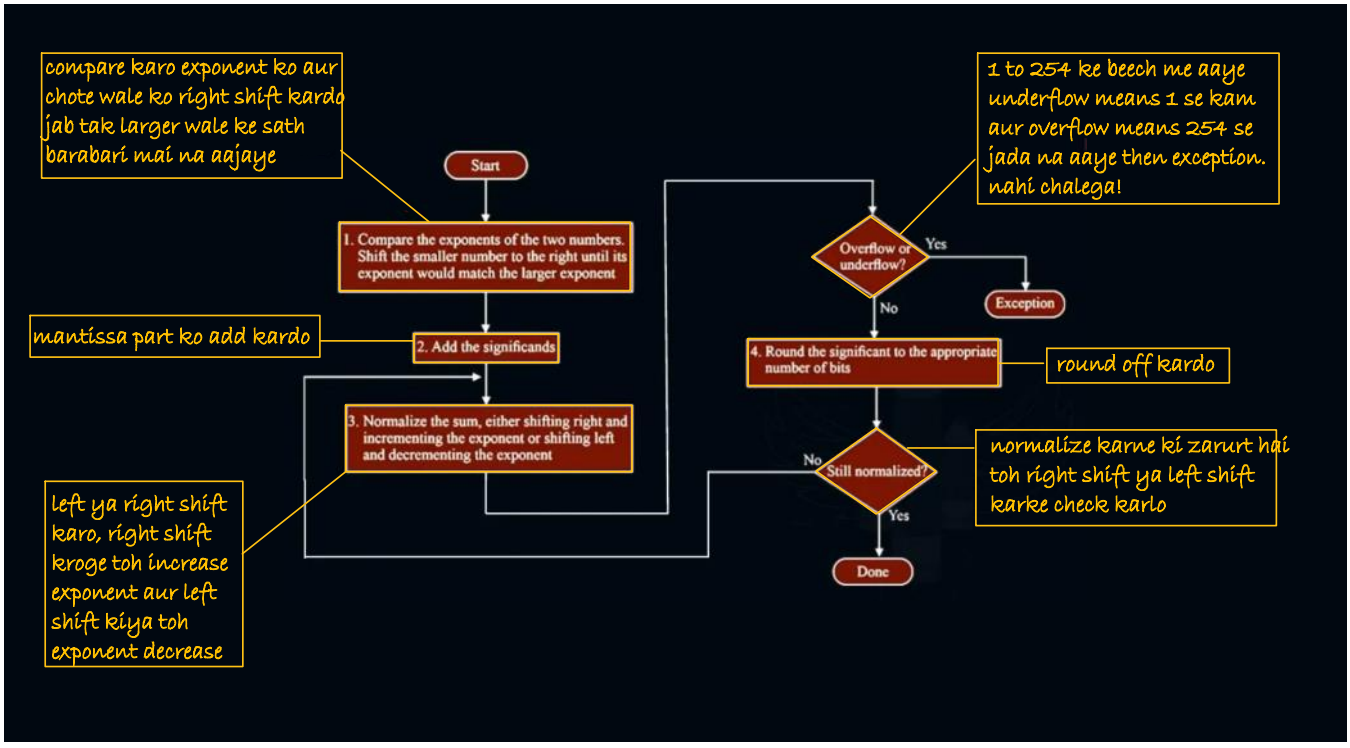
A = ...x2^2

and B = ...x2^3

power same honi chaiye

(i) choose the number with the smaller exponent and shift its mantissa right a number of steps equal to the difference in exponents.
(ii) set the exponent of the result equal to the larger exponent.
(iii) perform addition / subtraction on the mantissas and determine the sign of result.

normalize the resulting value, if neccesary
1.something x 2^e



example:
show the IEEE 754 binary representation of the number -0.75_{10} in single and double precision

-0.11×2^0
bias = 127
e = -1
S = 1
E = -1 + 127 = 126

1bit	8bit	bit
S	E	M

Example:

Show the IEEE 754 binary representation of the number – 0.75_{ten} in single and double precision.

-0.75 : -0.11×2^0

single precision

1bit	8bit	23bit
S	E	M

to make it implicit we have to left shift
-1.1×2^{-1}

S=-1
bias = $2^8-1 = 128-1 = 127$
E = E + bias = -1 + 127 = 126
E = 126 (001111110)

1bit	8bit	23bit
1	001111110	10000000000000000000

addition of two number

A : 9.999×10^1
B : 1.610×10^{-1}

$1.610 \times 10^{-1} = 0.1610 \times 10^0$ (right shift)
$= 0.01610 \times 10^1$ (right shift)

now they are equal.

9.999×10^1
0.016×10^1
10.015

10.015×10^1

normalized nahi hai toh normalized kar diya
1.0015×10^2

ab round off krdiya
1.002×10^2

(ii) multipication rule

(i) add the exponents and subtract 127

(ii) multiply the mantissas and determine the sign of the result.
(iii) normalize the resulting value if neccesary.

example :

$(1.110 \times 10^{10}) \times (9.200 \times 10^{-5})$

(i) add the exponents and subtract 127

10 + (-5) = 5
10 + 127 = 137
and -5 + 127 = 122
new exponent : 259

subtract 127 :
259 - 127 = 132
E : range is 1 to 254 hence 132 allowed

(ii) multiply the mantissas and determine the sign of the result.

1.110 (3points ke baad)
9.200 (3points ke baad)

10.212000 (total 6 points ke baad)
10.212×10^{5}

normalized : 1.0212^{+6}

(iii) divide rule
(i) subtract the exponents and add 127
(ii) divide the mantissas and determine the sign of the result.
(iii) normalize the resulting value if neccesary.

the addition or subtraction of 127 in the multiply and divide rules results from using excess -127 notation for exponents.

ALU and control unit

index :

(i) Component of computer

(ii) Register, ALU, timing and signal control

(iii) working of register

(iv) working of MUX, common bus

(v) micro operation

(vi) micro program

(vii) ALU data path

(viii) control unit

what is micro operation : how smallest operation is perfomed

instruction cycle

sub cycle -
(i) fetch cycle : to fetch the instruction from memory to CPU
PC MAR, MAR Memory, Memory MBR, MBR IR

(ii) execute cycle : to execute (to process) the fetch instruction.
it decodes; does the analysis of the instruction.

objective of micro operation : how execute, data path, register, MUX, common bus, system bus will work here and how control signal will be implemented? how, what, when and why?

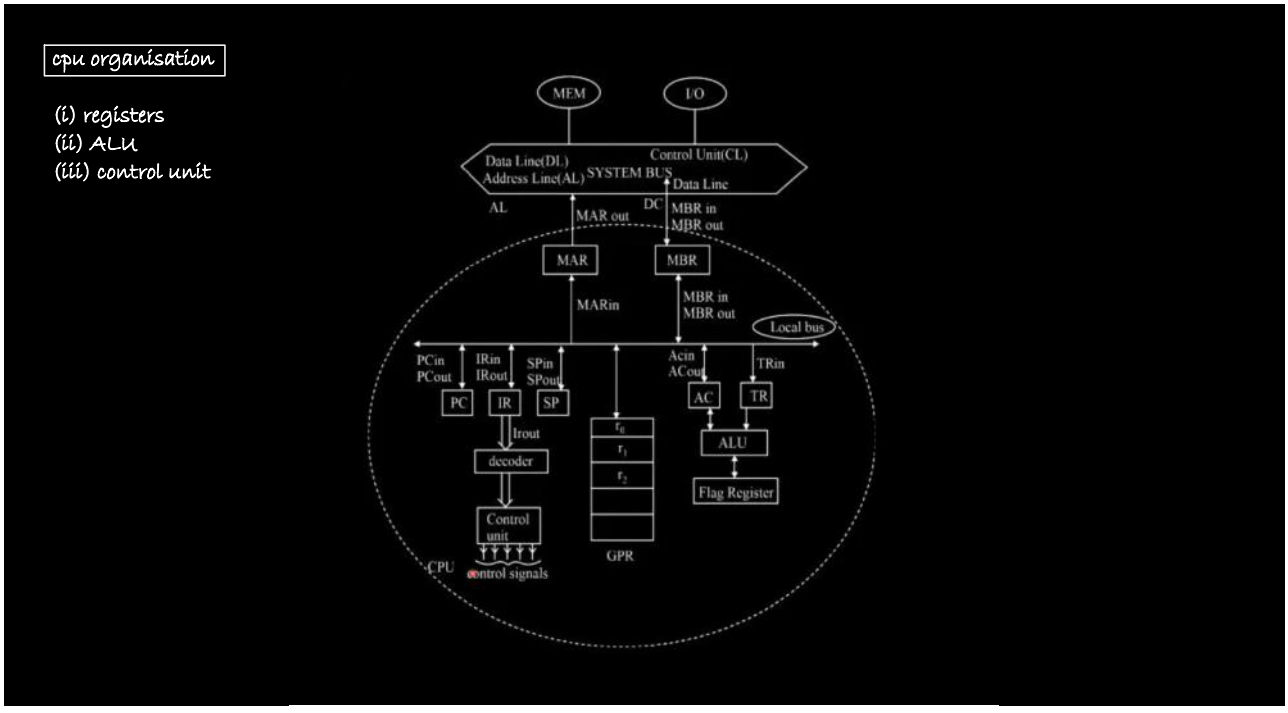
topic : structure of computer

component of the computer :

- ALU (arithmetic logical unit)

component of the computer :

1. CPU
   - ALU (arithmetic logical unit)
   - Registers
   - CU(control unit)
2. Memory
   - Primary/main memory
   - secondary/auxillary memory
3. I/O
   - Input device
   - Output device



#memory address register : stores all the address of memory used for read/write operation.

why MAR/AR?
because it is connected to address line of the system bus. knows address and how to and where to go etc.

#memory buffer register : hold the instructions or data

why MBR/DR/MDR(data)?
connected to the data line of the system bus

R_{in} : if the content of the bus loaded into register (andar aa raha hai)

R_{out} : if the content from ther register will be placed on bus (bahar jaa raha hai)

Register : collection of bits / sequence of bits, each bit stored in flip/flop
(fastest) - stores the data (temporary storage)
- register present inside the CPU
- made with flip-flop; flip-flop is 1 bit storage device)

8 bit register :  8 bit storage device / stores 8 bit data.

based on the information they have register types :
(i) data register : stores the data
(ii) address register : stores the address

based on the task/purpose assigned to them they have register types
(i) general purpose register : use for any purpose
(ii) special purpose register : use for specific purpose / pre-determined functionality

special purpose registers :

1. Memory address register (MAR)
2. I/O address register (I/O AR)
3. Program counter (PC)
4. Stack pointer register (SP)

→ stores the address

5. Memory buffer register (MBR/MDR/DR)
6. I/O data/buffer register (I/O BR)
7. Instruction register (IR)
8. Accumulator (AC)
9. Flag register / program status word (PSW)

→ stores the instruction (or) data

Program counter : When instruction is fetched (fetch cycle executed) then PC denotes the starting address of next instruction.

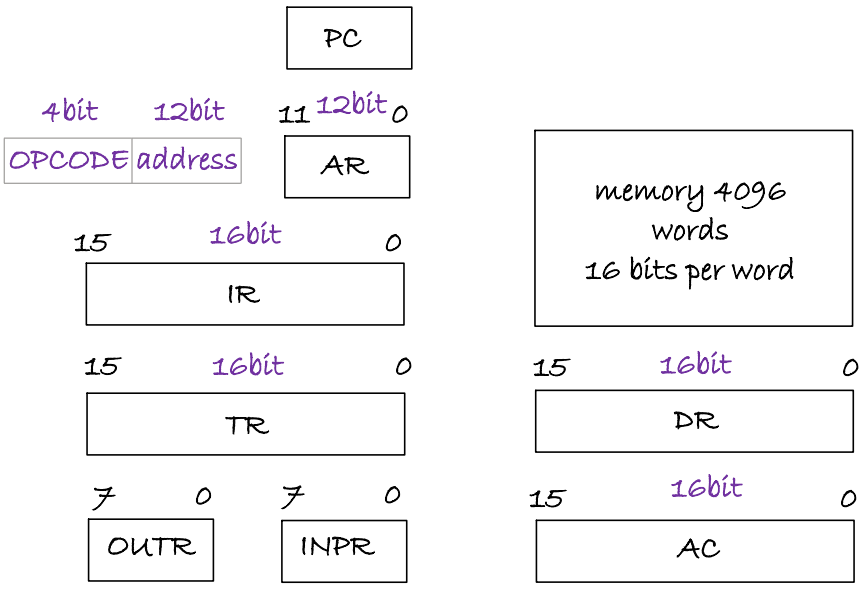
→ PC increment by 1 (or) PC increment by '4' (1word=4Byte) or '8'(1word=8Byte) or 'x' value.
gets Instruction address register / Instruction point register.

q. what is the size of each and every register?

memory : 4096 x 16bit
2^{12} x 16bit

address = 12bit
data = 16bit

12bit
11 0



basic computer registers and memory

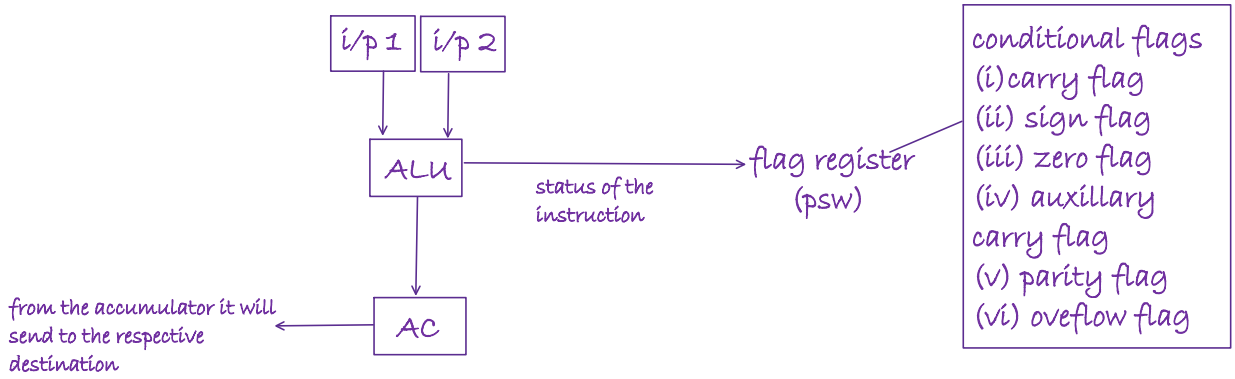
(i) MAR / AR : 12bit
(ii) PC / stack pointer register : 12bit

address register
stores the address

(iii) MBR / DR : 16bit
(iv) IR : 16bit
(v) AC : 16bit
(vi) TR/ flag register / program status word : 16bit

data register
stores the data

ALU (arithmetic logical unit) : ALU is a hardware that performs arithmetic, logical operations and condition checking, etc.
(or)
it performs multiple operation



CU (control unit) : timing signal and control signal. how and what to do each and everything.

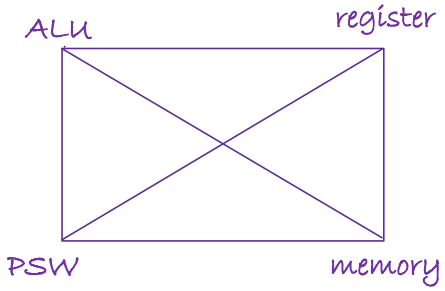
- timing signal : to execute the instructions in proper sequence
  Ex. fetch → decode → execute

in fetch cycle :
T_1: PC→MAR
T_2: MAR→MBR
T_3: MBR→IR

non technical example :
T_1: enrollment
T_2: admit card
T_3: exam writing
T_4: result

q.consider if we have(total 4 components) then total connection required?

$^4C_2$ = 6 connections. charo ko connect karne ke lie 6 connections chaiye.



q.consider if we have 16 registers, 1 memory, 1 ALU , 1PSW and 1 other component (total 20 components) then total connection required?

$^{20}C_2$ = 190 connections.
so the solution is, instead of using 190 connections, connect all components to a common bus(internal
bus). at a time which part (components) will communicate? for that control signals are required.

why common bus is used?

topic : working of registers and multiplexer

(i) the number of multiplexer required = size of register (#of bits in register)
(ii) size of multiplexer required = number of register

q.4 register : A, B, C and D, each register is of 4bits
(i) the number of multiplexer required = 4 (size of register (#of bits in register))
(ii) size of multiplexer required = 4x1 (number of register)

q.if we have m registers and each register size is n bits then what is the number of mux and size of mux?

(i) the number of multiplexer required = n (size of register (#of bits in register))

(ii) size of multiplexer required = m x 1 (number of register)

q.if we have 32 registers and each register size is n 8bits then what is the number of mux and size of mux?

(i) the number of multiplexer required = 8 (size of register (#of bits in register))

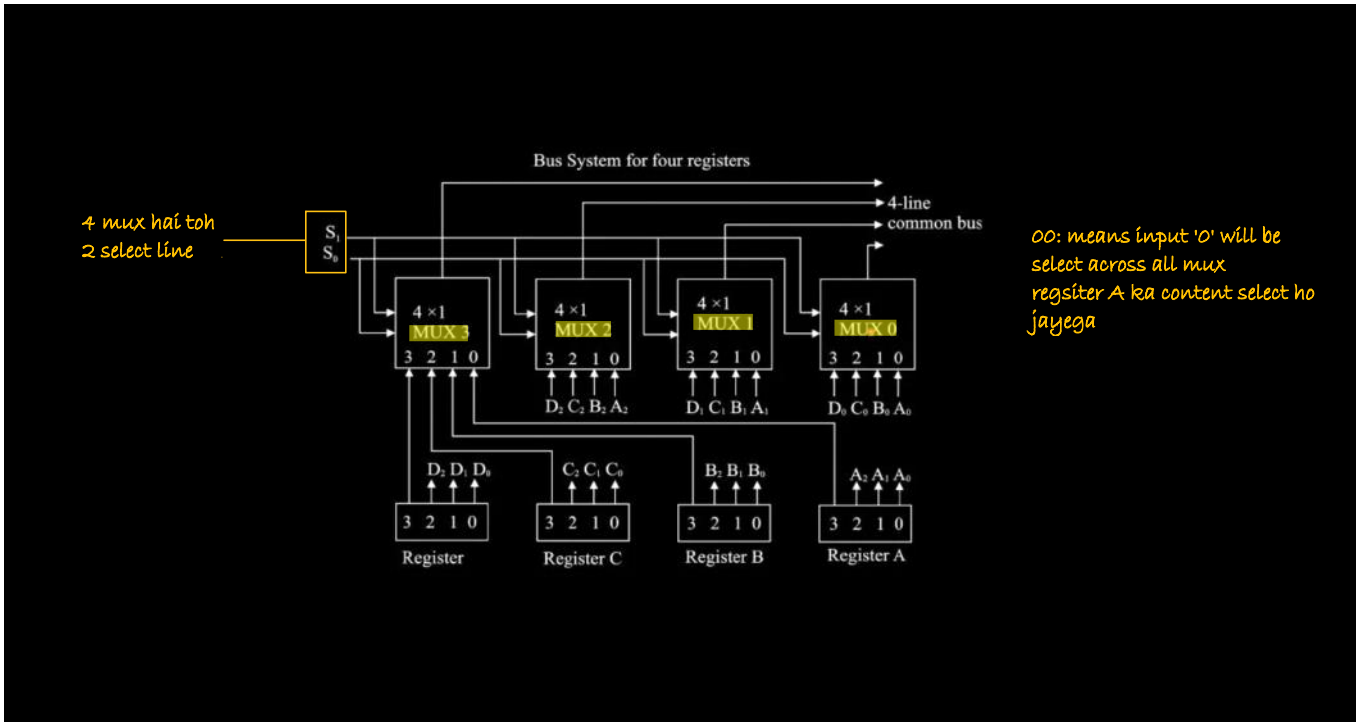
(ii) size of multiplexer required = 32 x 1 (number of register)

working of register and multiplexer

q.4 register : A, B, C and D, each register is of 4bits

(i) the number of multiplexer required = 4 (size of register (#of bits in register))

(ii) size of multiplexer required = 4x1 (number of register)



S_1	S_0	register selected	
0	0	A	00: means input '0' will be select across all mux register A ka content select ho jayega aur iska content phir common bus mai jayega.
0	1	B	01: means input '1' will be select across all mux regsiter B ka content select ho jayega

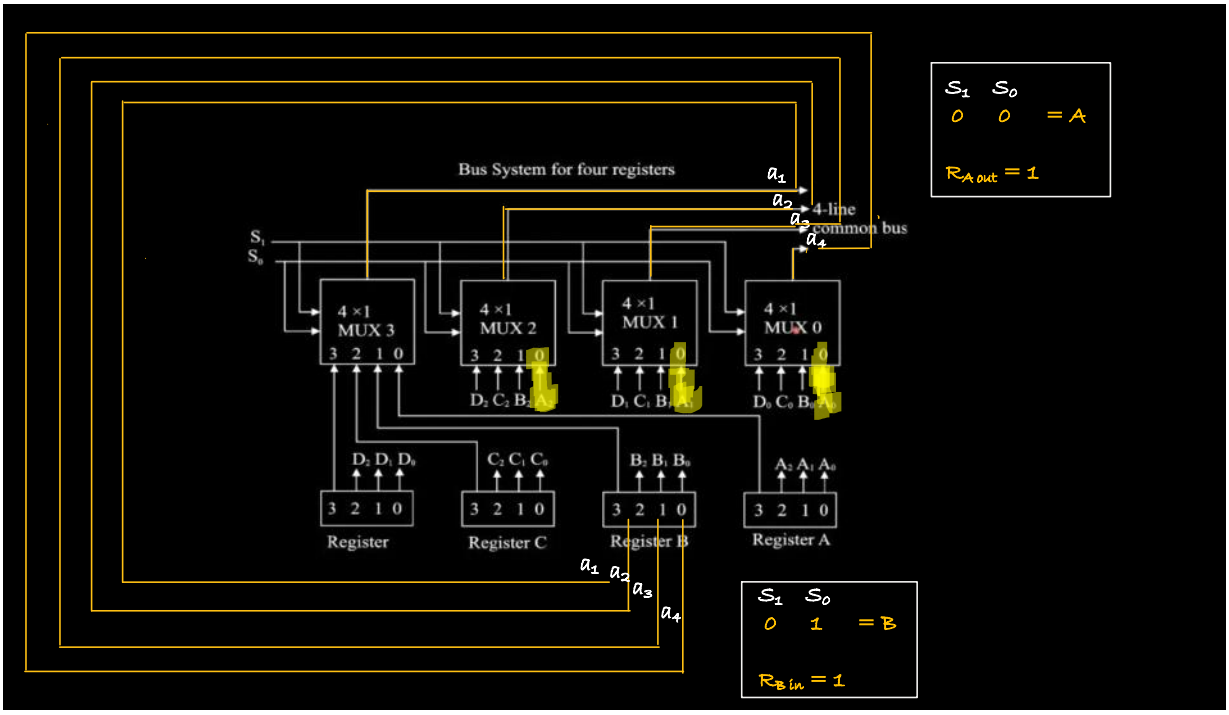
0	1	B	01: means input '1' will be select across all mux regsiter B ka content select ho jayega aur iska content phir common bus mai jayega.
1	0	C	10: means input '2' will be select across all mux register ka content select ho jayega aur iska content phir common bus mai jayega.
1	1	D	11: means input '3' will be select across all mux regsiter D ka content select ho jayega aur iska content phir common bus mai jayega.

how data is transferred

register A to register B : $R_A \rightarrow R_B$

Process : Register A content (data) will be given to MUX then that data will be transferred from MUX to common and then that data will be loaded into register B from common bus

step (i) : R_A MUX
select line 00 A will be selected/enabled and loaded into common bus



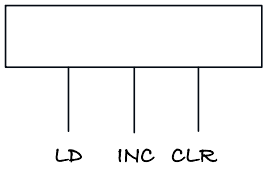
Why R_{out} and R_{in} is used?
register (RA) output connected to MUX;

then from the MUX to common bus;
then common bus to load into respective register (RB)

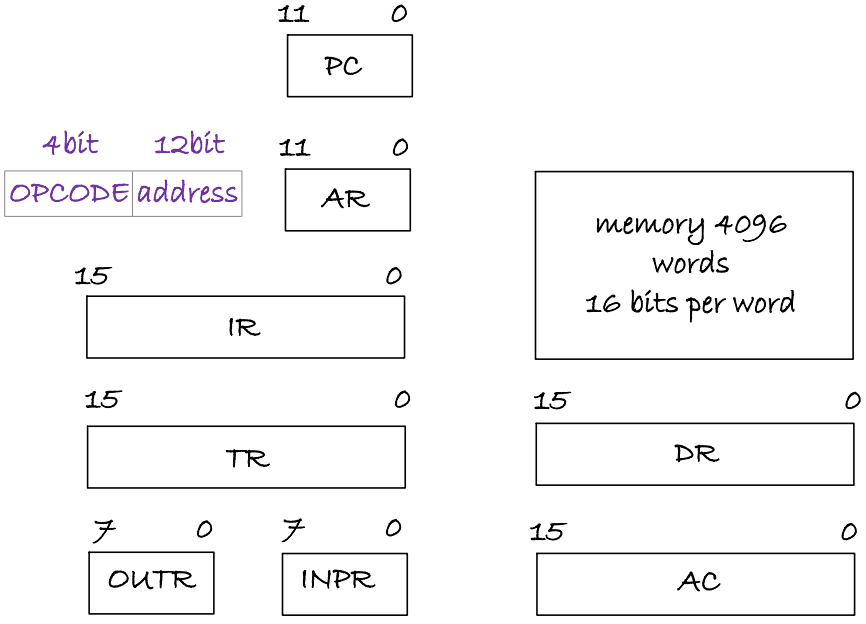
when R_{out} is set to 1 then respective register data is loaded into the MUX then MUX to common bus, common bus is connected to all registers, the register which have R_{in} is set to 1 in that respective register bus data is loaded.

R_A to R_B : $R_{A\ out}$ $R_{B\ in}$
00[A] $R_{A\ out} = 1$ then R_A data load to MUX then common; then we get $R_B = 1$ then from common bus content is loaded into register.

working of computer



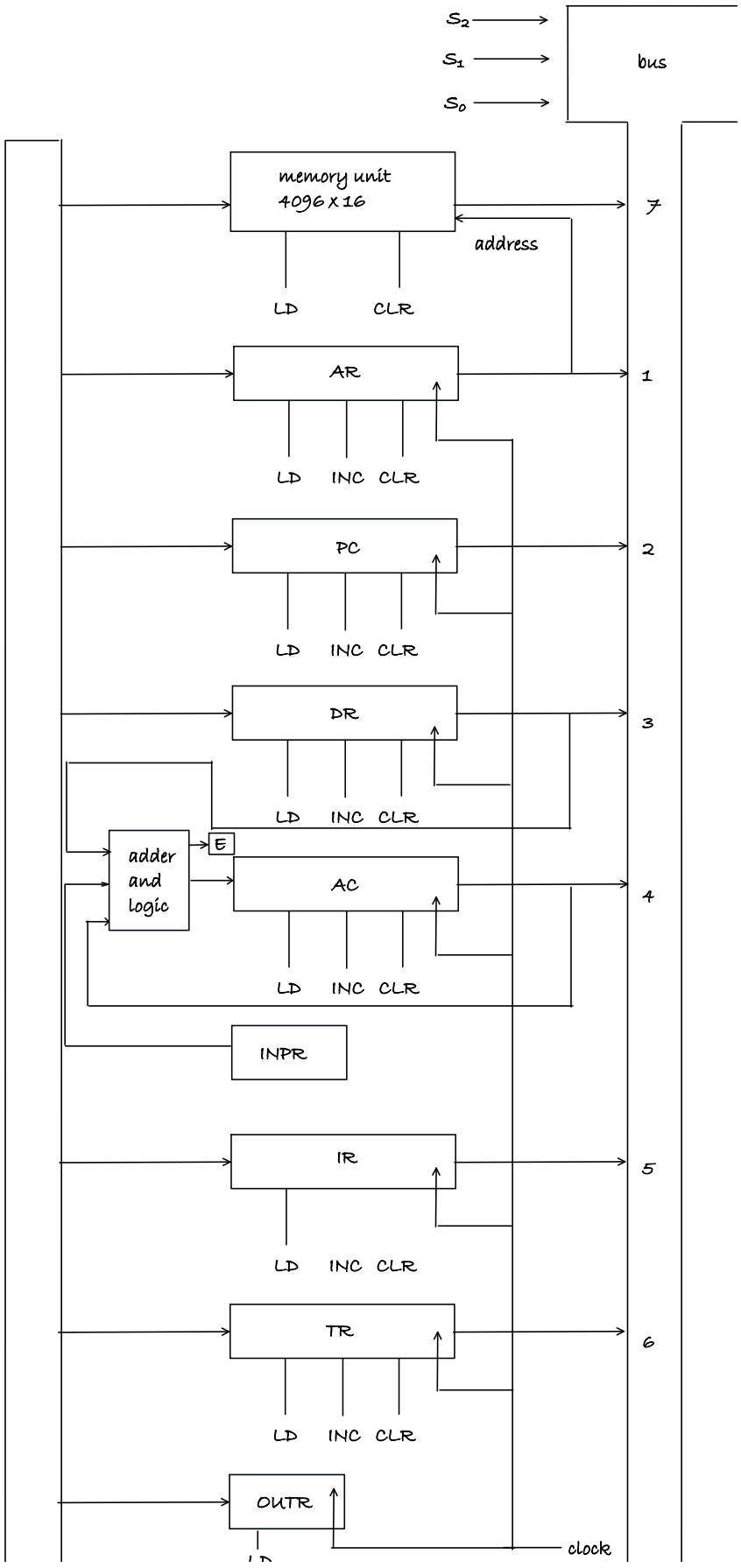
register me 3 input rehte hai
LD : load
INC : increment
CLR : clear

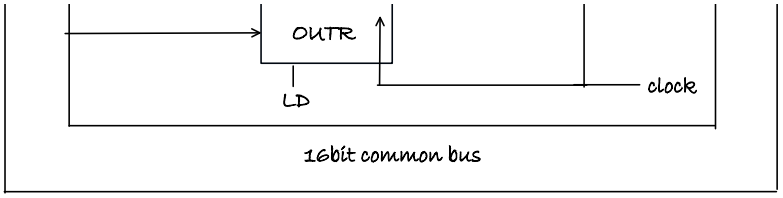


basic computer registers and memory

4096 x 16
2^{12} x 16

address line= 12bit
data line = 16bit





basic computer registers connected to a common bus

total components : 7

memory : 7

AR : 1

PC : 2

AC : 4

DR : 3

TR : 6

IR : 5

then 3 select lines are required

S_2	S_2	S_2	enables	
1	1	1	7(memory)	read control signal for load from memory
0	1	0	2(PC)	
1	0	1	5(IR)	

read : Load : memory read
write : store : memory write

in memory we have 'n' locations so, which memory address contains (data) loaded into the bus? it is given by AR(MAR) aur MAR mai PC se aayega

111 → 7 (memory)
010 → 2 (PC), PC will be enabled and content of PC will be loaded into common bus and AR

register(MAR) [Load(IN) is set to 1(active) so AR(MAR) get the memory address
fetch cycle here
memory to CPU (IR)
101→3(IR)

- memory
- PC
- IR

but in what sequence (timing) which operation performs?
it is the responsibility of timing signal and control signals, yeh timing signal batayega

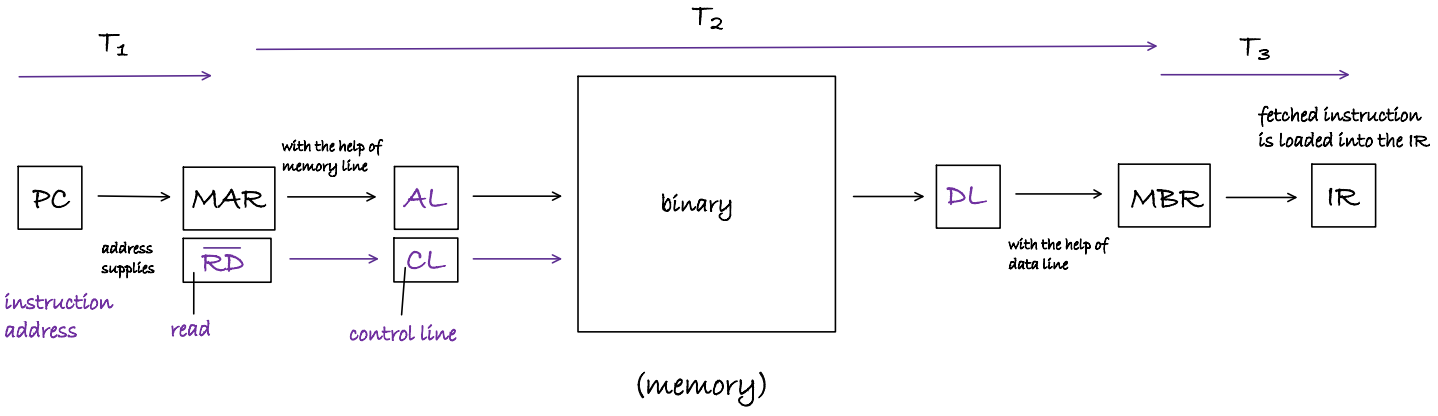
fetch cycle : instruction is fetch from memory to CPU (IR)

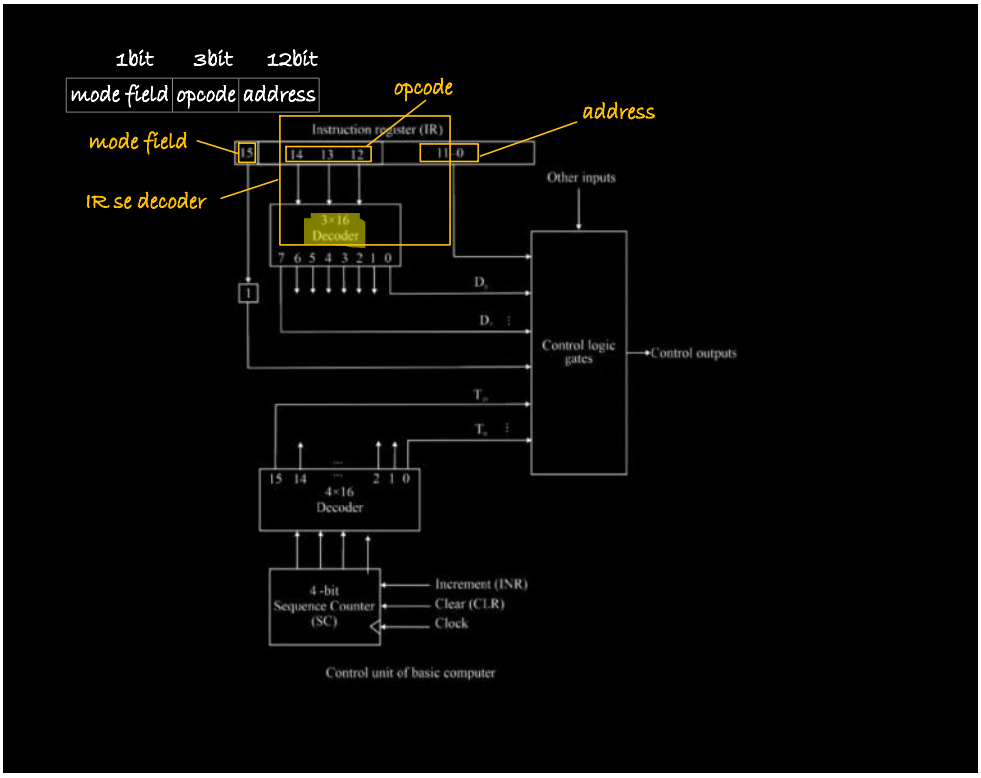
T_1: PC→M(MAR)

T_2 : MAR →Memory
T_2 : Memory →MBR [MAR→MBR]

T_3 : MBR→IR

PC se AR mai address waha se memory mai gaye aur memory se MBR aur MBR se hum IR mai aye aur ise hum ALU data path kehte hai





Control unit of basic computer

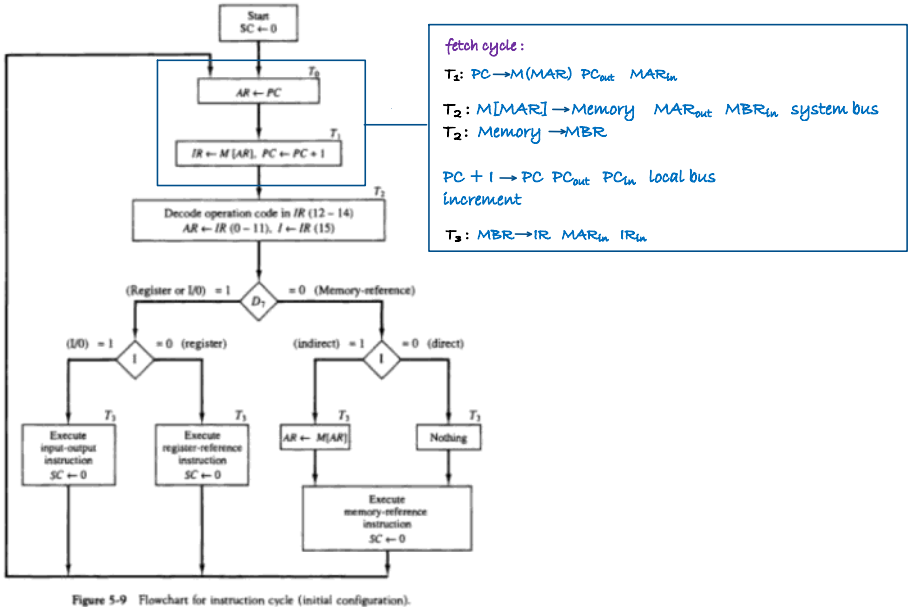


Figure 5-9 Flowchart for instruction cycle (initial configuration).

fetch cycle :

T_1: PC → M(MAR) PC_{out} MAR_{in}

T_2: M[MAR] → Memory MAR_{out} MBR_{in} system bus
T_2: Memory → MBR

PC + 1 → PC PC_{out} PC_{in} local bus
increment

T_3: MBR → IR MAR_{in} IR_{in}

ALU, data path and control unit

(i) micro instruction/operation

(ii) micro program

(iii) control unit design

what is data path?
MUX, ALU, buses, register ; how they process or path of their processing

every register have two switch

R_{in} : if the content of the bus loaded into register (andar aa raha hai)

R_{out} : if the content from ther register will be placed on bus (bahar jaa raha hai)

(i) micro instruction/operation : in fetch cycle, execute cycle, interrupt cycle we have small operations [x operations]

computer
↓
programs
↓
instruction/data
↓
instruction cycle
(subcycles)
fetch
execute
interrupt

chote chote operations hote is cycle mai aur use hum micro operations kehte hai aur execute karayega control signal aur control aayege control unit ki taraf se

micro operation example :
register to register transfer

micro operation consume 1 cycle to complete the execution

the functional, or atomic, operations of a processor

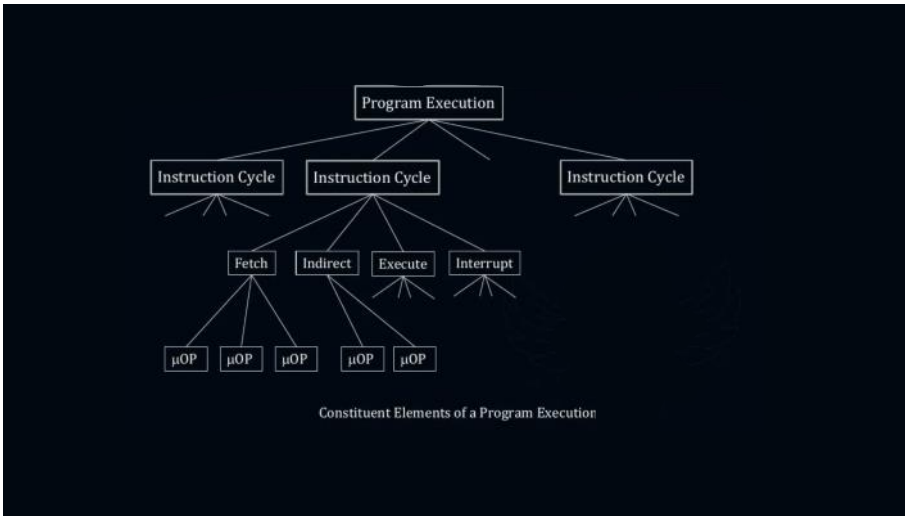
(i) series of steps, each of which involves the processor registers.

(ii) micro refers to the fact that each step is very simple and accomplishes very little

(iii) the execution of program consists of the sequential execution of instructions.

each instruction is executed during an instruction cycle made up of shorter sub cycles (fetch, indirect, execute, interrupt)

the execution of each sub cycle involves one or more shorter operations (micro-operations)



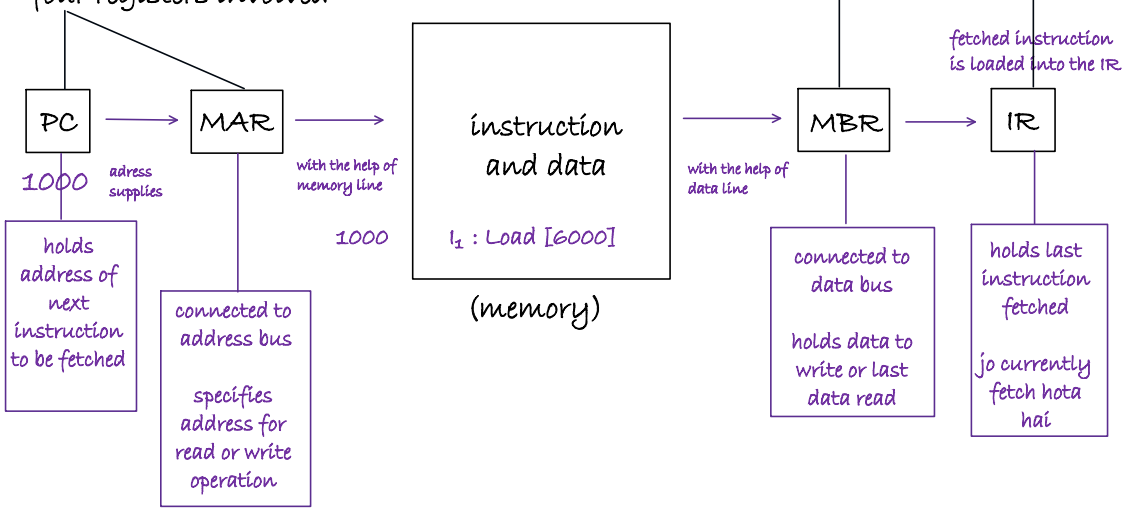
(ii) micro program : collection (in sequence) of micro operation. kisi task ko krne mai 3 sequence lage uska collection is micro program.

hardware level par jo kaam hota hai vo perform karta hai micro program.

instruction cycle

(i) fetch cycle : instruction is fetch from memory to CPU (IR)

four registers involved



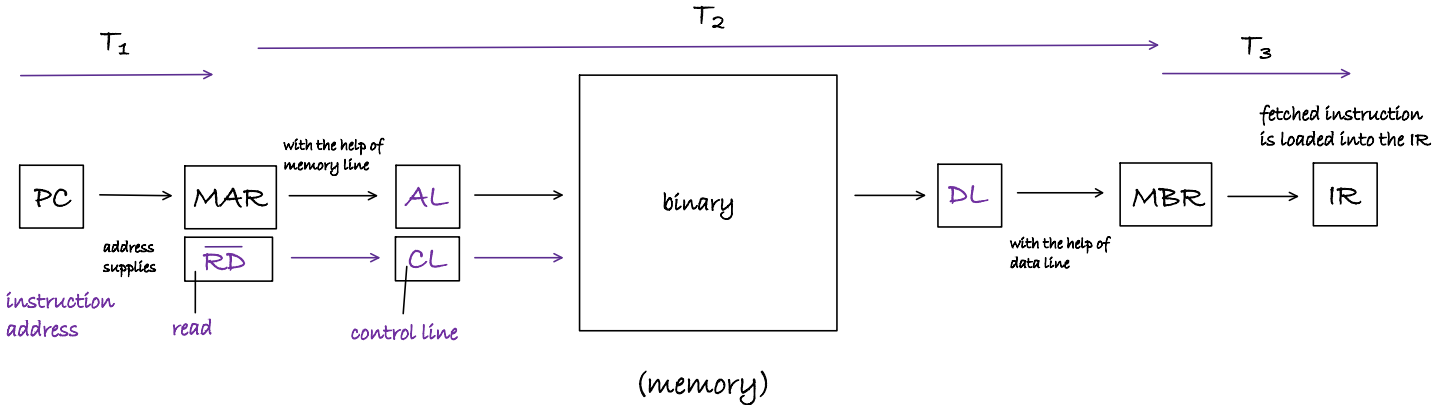
T_1: PC→M(MAR)

T_2 : MAR→Memory

T_2: Memory →MBR

T_3 : MBR→IR

micro operations in fetch cycle



T_1: PC → M(MAR) PC_{out} MAR_{in}

T_2: M[MAR] → Memory MAR_{out} MBR_{in} system bus
T_2: Memory → MBR

PC + 1 → PC PC_{out} PC_{in} local bus
increment

T_3: MBR → IR MAR_{in} IR_{in}

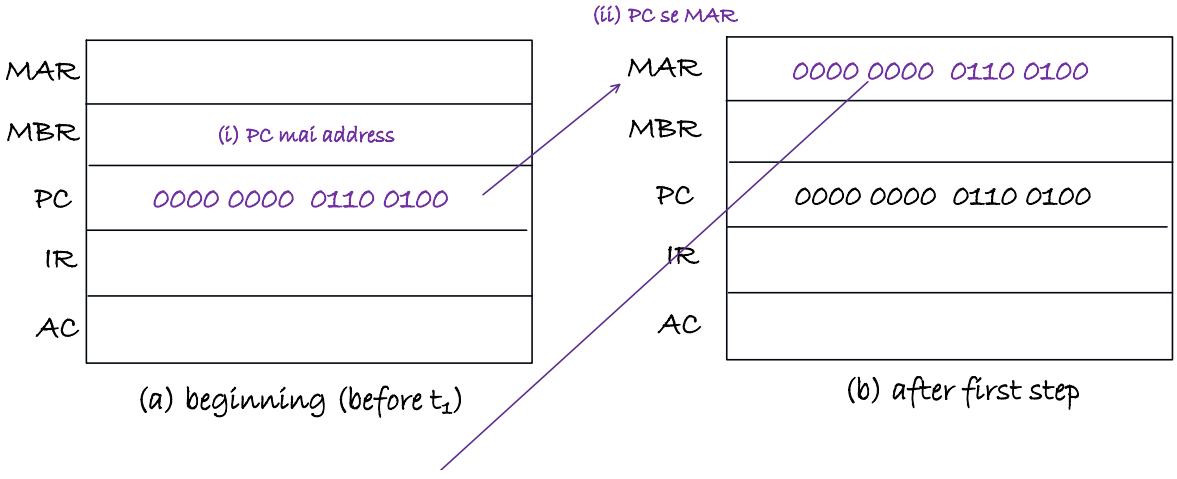
proper sequence must be followed

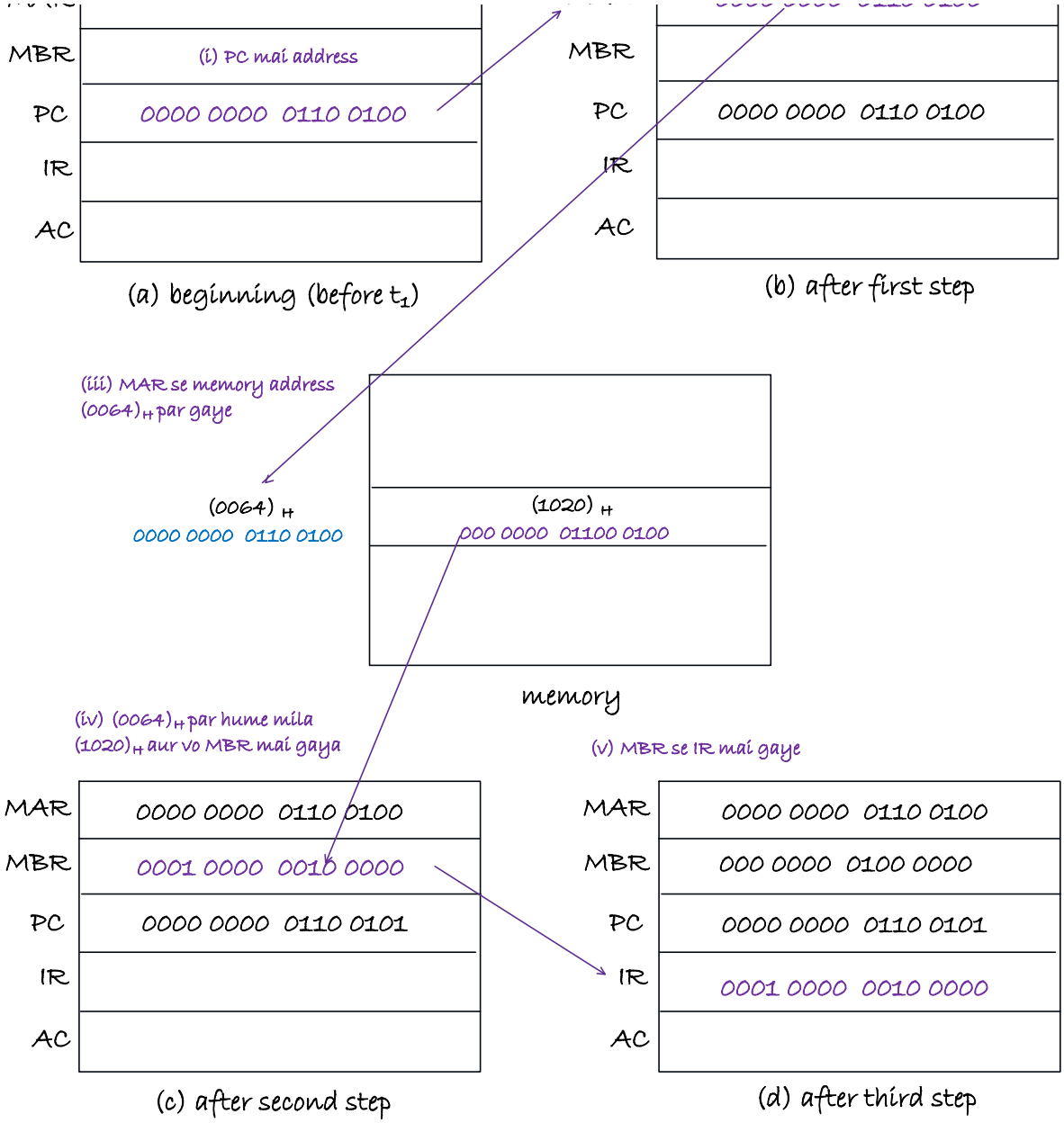
(i) MAR ← (PC) must precede MBR ← (memory)
conflicts must be avoided

(ii) must not read and write same register at same time

(iii) MBR ← (memory) and IR ← (MBR) must not be in a same cycle.

if different component are there then perform in a single cycle (no conflict)





at the end of fetch cycle instruction is fetched from memory to CPU (IR) register

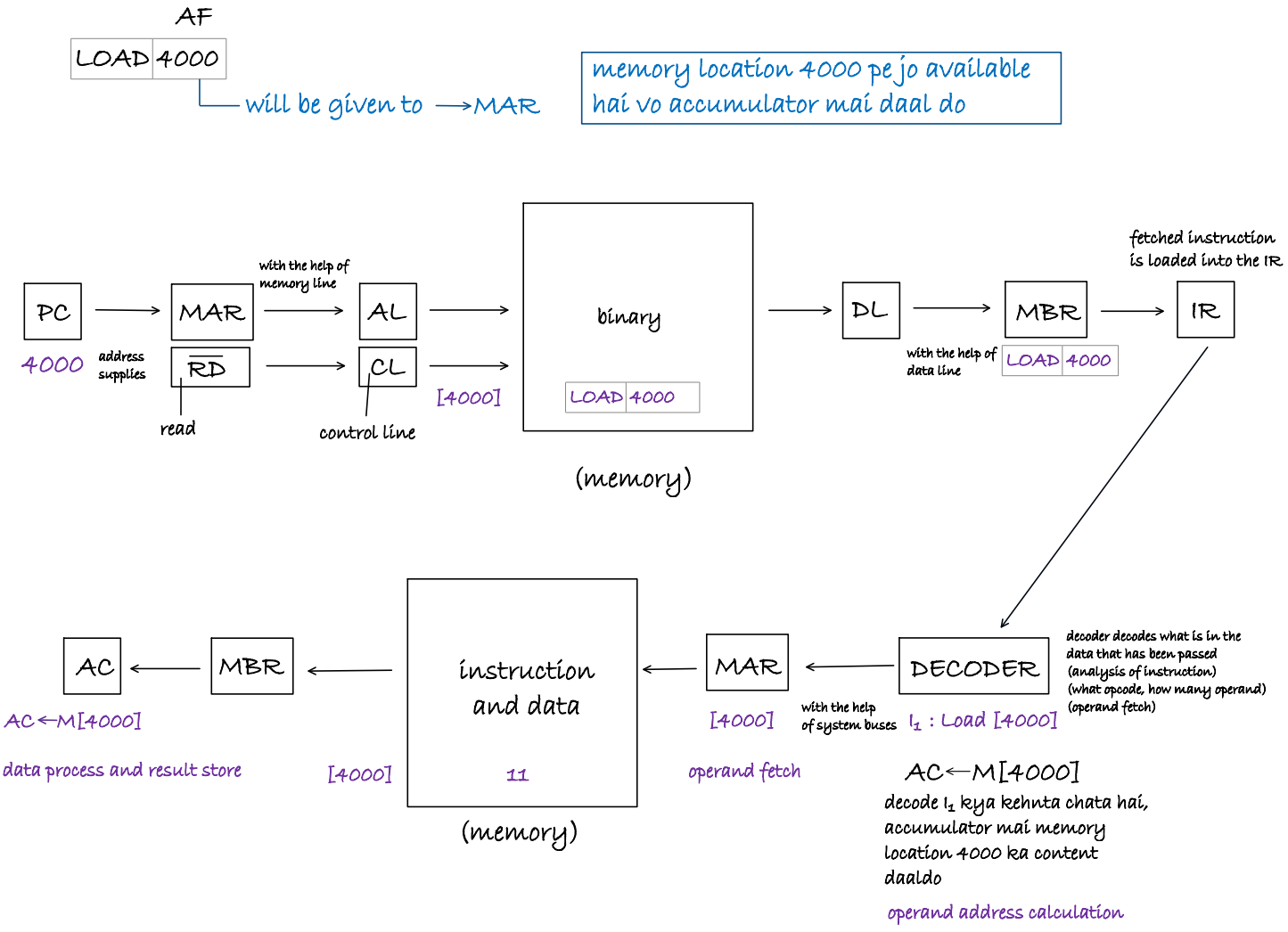
IR will have :

OPCODE	OPERAND REFERENCE

(ii) execute cycle :

- decode the instruction (analysis of instruction) [what opcode, how many opcode, kehna kya chate ho]
- operand address calculation
- OPERAND fetch (data)

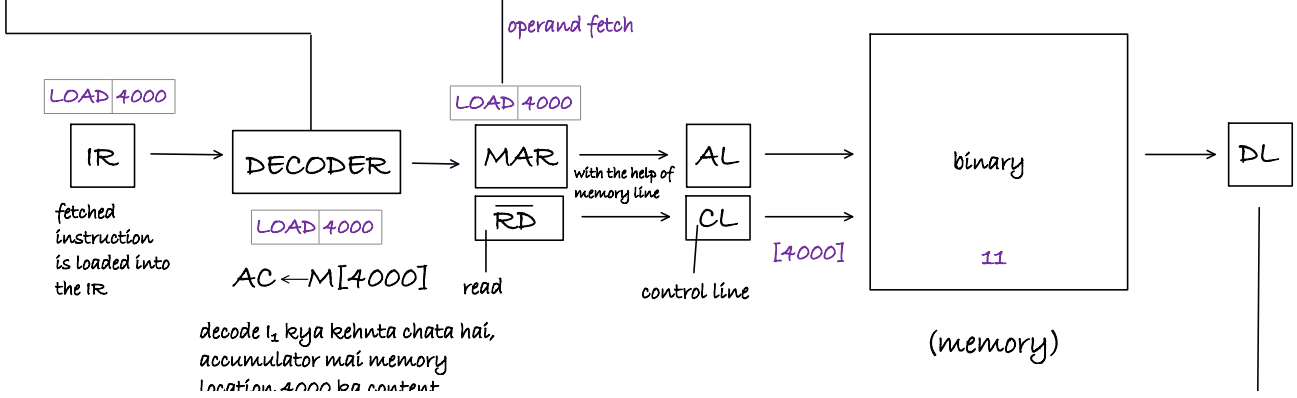
then data process karke result kara denge (write back)



(a) ID stage : enable the hardware to perform the operation (instruction decode / analysis)

(b) OF stage : AM's (addressing mode) are required to access (operand fetch)

example : execute cycle of direct addressing mode

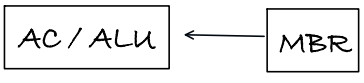


decode I_1 kya kehnta chata hai,
accumulator mai memory
location 4000 ka content
daaldo

operand address calculation

decoder decodes what is in the
data that has been passed
(analysis of instruction)
(what opcode, how many operand)
(operand fetch)

(memory)



AC ← M[4000]

11

data process
and
result store

execute cycle of direct addressing mode

T_1 : IR [address] → MAR : IR_{out} MAR_{in}

T_2 : M [MAR] → MBR(EA) : MAR_{out} MR_{in}

T_3 : MBR → AC/ALU : MBR_{out} AC/ALU_{in}

why ALU ? because if we have this kind of data type then we need ALU to perform these operations.

i_2 : ADD R_1 X : $R_1 \leftarrow R_1 + M[X]$

direct AM
(EA)

x location sabse pehle
MAR ko deni padegi

ADD R_1 X : content of location X ko add kardenge register R_1 mai

step (i) IR ki address field se MAR mai jayega

step (ii) jo reference memory location hai use read karlenge

step (iii) R_1 aur MBR ka content ALU ke through add karke
accumulator mai daal denge

soometimes additional micro-operations lag sakta hai (may) register ke content ko laane ke liye

additonal micro-operations may be required to extract the register reference from the IR and perhaps to stage the ALU inputs or outputs in some intermediate registers.

(i) direct addressing mode load[4000] memory read (done)

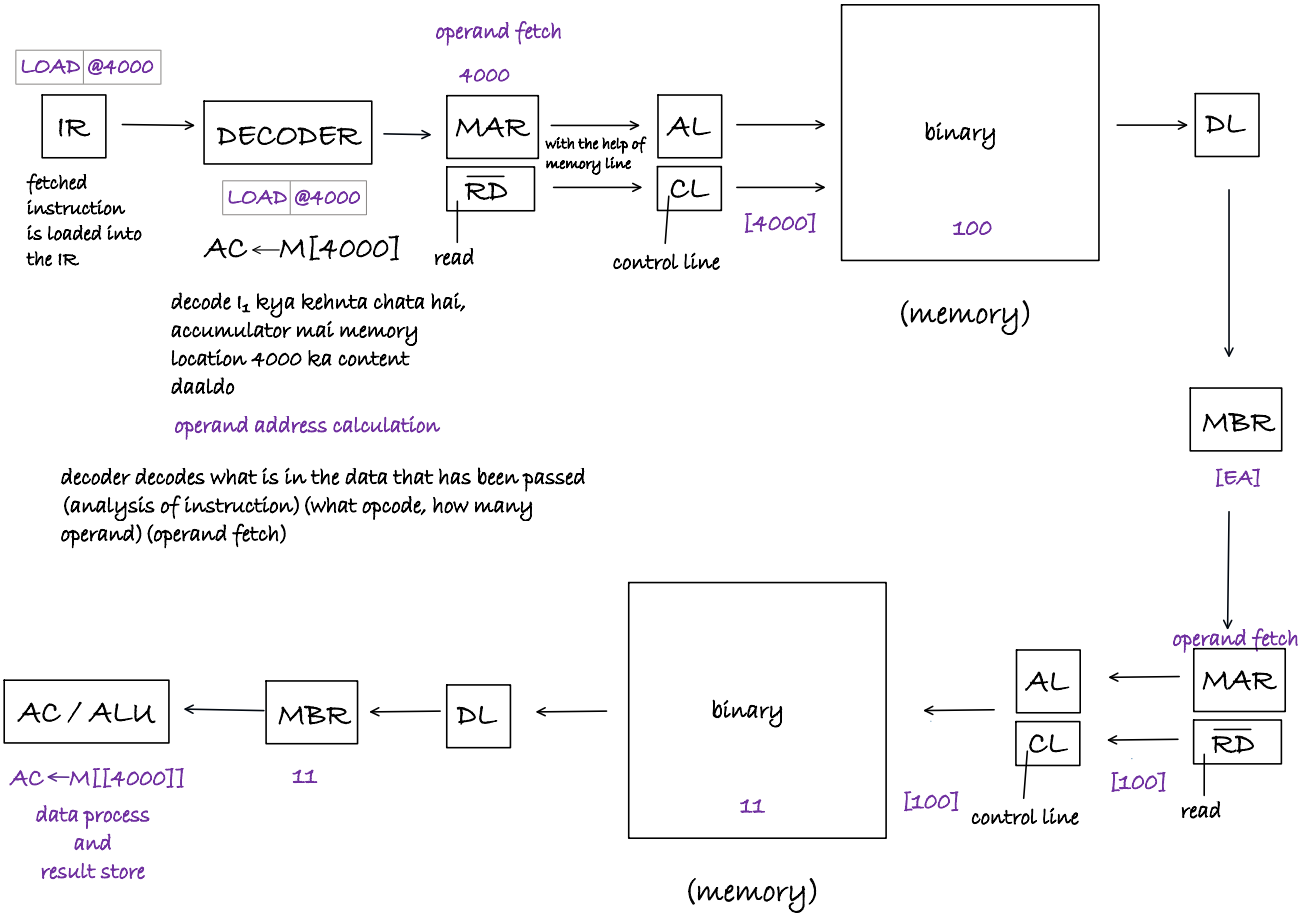
example : execute cycle of indirect addressing mode

load@4000 memory read

address of effective address

AC←M[M[4000]]

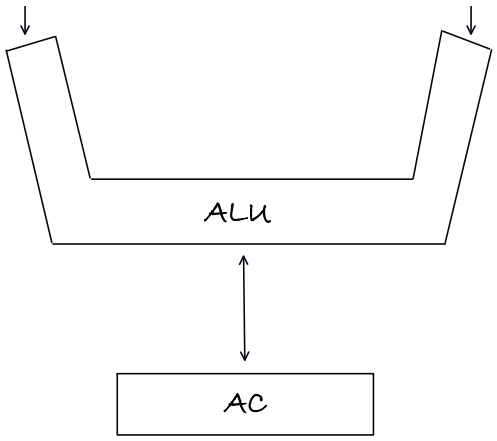
AC←M[EA]



q. $R_0 \leftarrow R_1 + R_2$

$T_1 : T_1 \leftarrow R_1 ; R_{1\,out}\ T_{1\,in}$

$T_2 : T_2 \leftarrow R_2 ; R_{2\,out}\ T_{2\,in}$



T_2 : $T_2 \leftarrow R_2$; $R_{2\,out}$ $T_{2\,in}$

T_3 : $AC \leftarrow T_1 + T_2$; $T_{1\,out}$ $T_{2\,out}$ ALU_{add} AC_{in}

T_4 : $R_0 \leftarrow AC$; AC_{out} $R_{0\,in}$

[MCQ]

#Q. Consider the following data path diagram [GATE-2020-CS:]

Consider an instruction: R0←R1+R2. The following steps are used to execute it over the given data path. Assume that PC is incremented appropriately. The subscripts r and w indicate read and write operations, respectively.

1. $R2_r$, $TEMP1_W$, ALU_{add}, $TEMP2_W$
2. $R1_r$, $TEMP1_W$,
3. PC_r, MAR_W, MEM_r
4. $TEMP2_R$,RO_W
5. MDR_r, IR_W

Which one of the following is the correct order of execution of the above steps?

A 3, 5, 1, 2, 4

B 2, 1, 4, 5, 3

C 3, 5, 2, 1, 4

D 1, 2, 4, 3, 5

PC to MAR : PC_{out} MAR_{in}
MAR to MBR : MAR_{out} MBR_{in}
MBR to IR : MBR_{out} IR_{in}

instruction given : $R_0 \leftarrow R_1 + R_2$

fetch cycle :
step (i) 3. PC_r, MAR_w, MEM_r : PC →MAR and memory read
step (ii) 5. MDR_r, IR_w : MDR →IR

execute cycle :
step (iii) 2. $R_{1\,r}$, $temp_{1\,w}$: $R_1 \leftarrow temp_1$
step (iv) 1. $R_{2\,r}$, $temp_{1\,r}$, ALU_{add}, $temp_{2\,w}$: $temp_2 \leftarrow R_2 + R_1$
step (v) 4. $temp_{2\,r}$, RO_w : $R_0 \leftarrow temp_2$

memory read

load [4000]

AC ← M[4000]

$\overline{RD}$ done

#memory store concept :

memory store

store [4000]

M[6000] ←AC

$\overline{WR}$

memory write

store [6000]

M[6000] ←AC

micro program

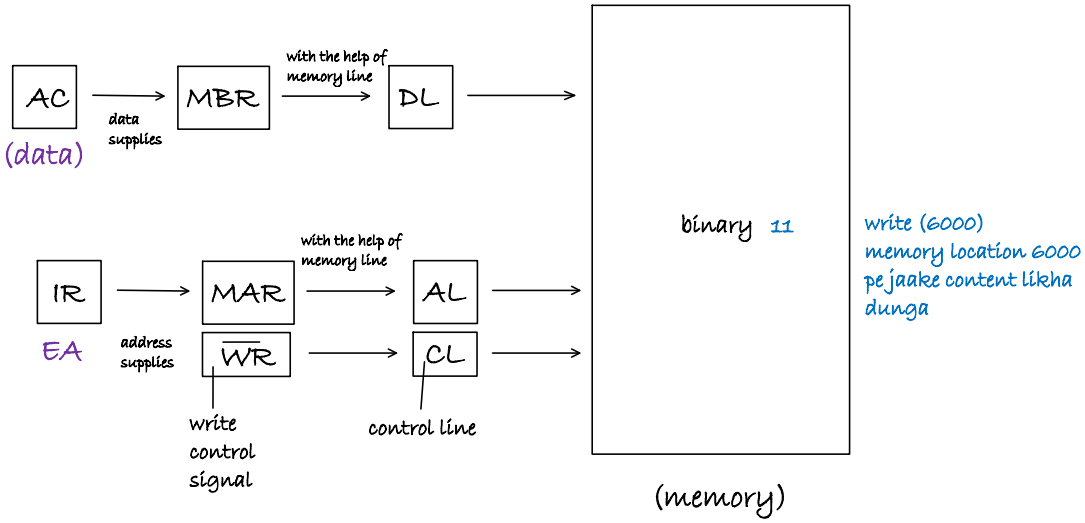
T_1 : AC →MBR ; AC_{out} MBR_{in}

T_2 : IR(address) →MBR ; IR_{out} MAR_{in}

T_3 : MBR →M[MAR] ; MBR_{out} MAR_{in}

data : MBR deals with data

address : MAR deals with address

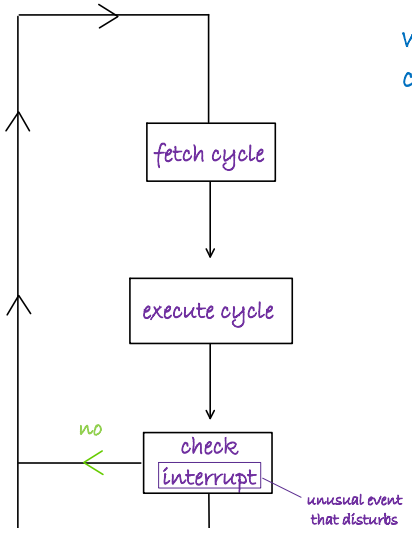


M[6000] ←AC

M[6000] ←data (11)

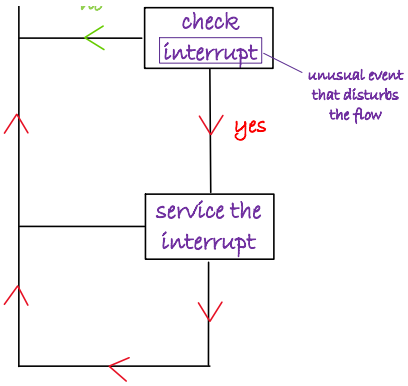
M[6000] ←content of accumulator

Instruction cycle with interrupt cycle :



when CPU encounter the interrupt then after finishing (completion) of current instruction execution, interrupt will be serviced.

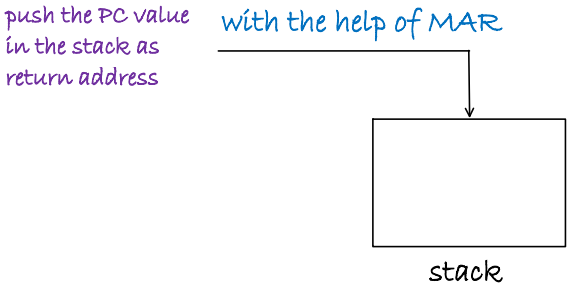
- when CPU encounter the interrupt then it push the PC (program counter) value into the stack as a return address and control transfer to ISR (interrupt subroutines)



if interrupt cycle occurs :

the nature of this cycle varies greatly from one machine to another

step (i) content of PC are transferred into MBR so that they can be saved for return from the interrupt.

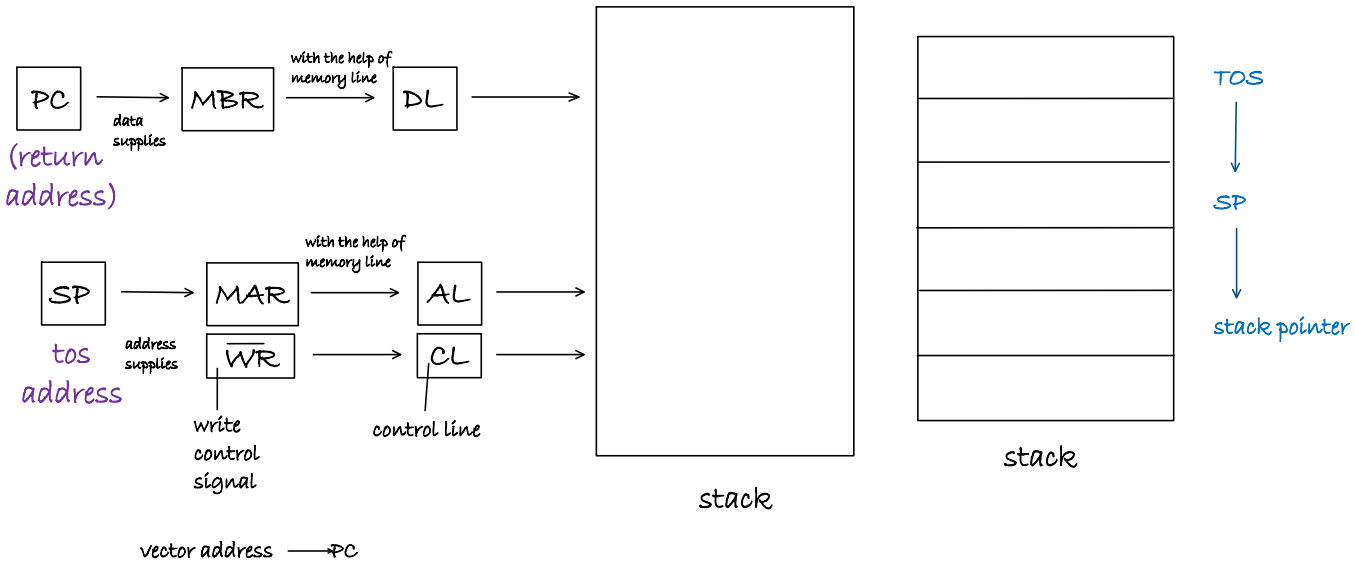


step (ii) then the MAR is loaded with the address at which the contents of PC are to be saved, and the PC is loaded with the address of the start of the interrupt processing routine.

- these two ations may each be single micro-operation(varies, depend on processor)
- because most processor provide multipe types or level of interrupts, it may take one or more additional micro operations to obtain the save address and the routine address before they can be transferred to the MAR and PC respectively

step (iii) once this is done, the final step is to store the MBR, which contains the old value of the PC, into memory

step (iv) the processor is now ready to begin the next instruction cycle.



note : stack memory ka address stack pointer rakhta hai

interrupt vector table : a data structure that associates a list of interrupt handlers with a list of interrupt requests in a table of interrupt vectors

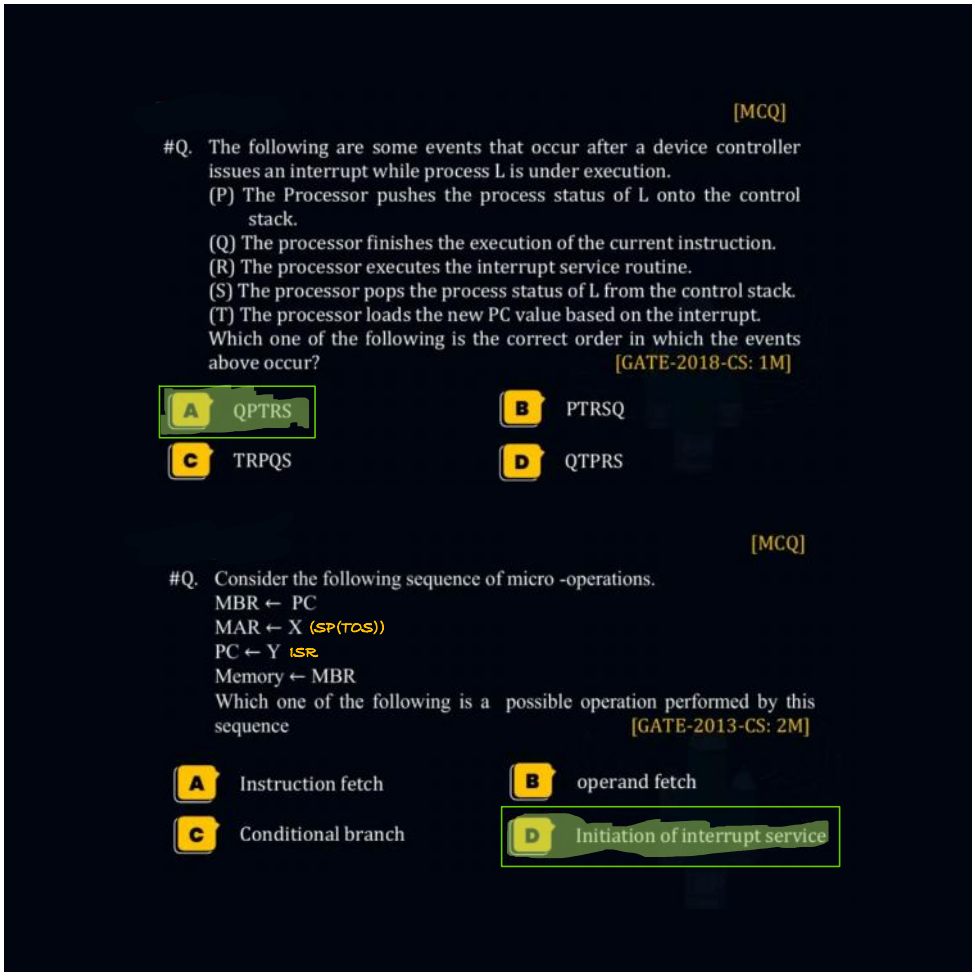
micro program

T_1 : PC → MBR ; PC_{out} MBR_{in}

T_2 : SP → MBR ; SP_{out} MBR_{in}

T_3 : MBR → M[MAR] ; MBR_{out} MAR_{in}
TOS

ISR address will be given to PC for interrupt service and once the service is completed RETI/IRET interrupt will return; then we will POP the PC value from the stack



[MCQ]

#Q. The following are some events that occur after a device controller issues an interrupt while process L is under execution.
(P) The Processor pushes the process status of L onto the control stack.
(Q) The processor finishes the execution of the current instruction.
(R) The processor executes the interrupt service routine.
(S) The processor pops the process status of L from the control stack.
(T) The processor loads the new PC value based on the interrupt.
Which one of the following is the correct order in which the events above occur? [GATE-2018-CS: 1M]

A QPTRS
B PTRSQ
C TRPQS
D QTPRS

[MCQ]

#Q. Consider the following sequence of micro -operations.
MBR ← PC
MAR ← X (SP(TOS))
PC ← Y ISR
Memory ← MBR
Which one of the following is a possible operation performed by this sequence [GATE-2013-CS: 2M]

A Instruction fetch
B operand fetch
C Conditional branch
D Initiation of interrupt service

topic : control unit functional requirements

(i) by reducing the operation of the processor to its most fundamental level we are able to define exactly what it is that the control unit must cause to happen
(kisi bhi processor ke operation ko reduce kara uske fundamental level par usi ke karan we are able to define exactly ki control unit kislie bani hai).

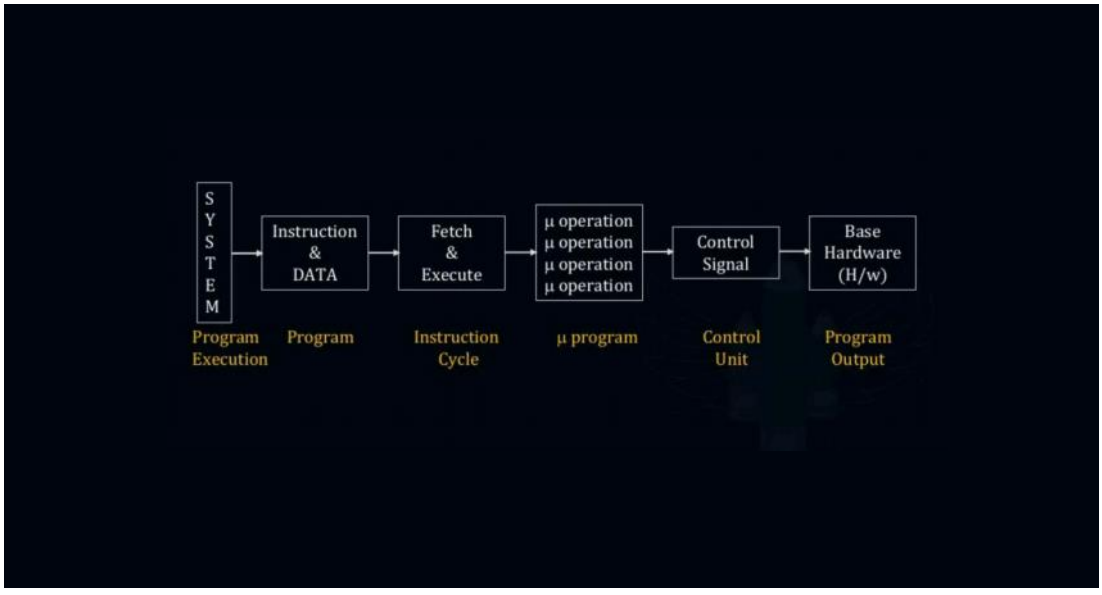
(ii) three step process leads to a characterization of the control unit :
- define basic elements of processor
- describe micro-operatons processor perfoms (kaise perform karna hai)
- determine the functions that the control unit must perform to cause the micro-operations to be perfomed. (uske lie kya function banega)

(iii) the control unit performs two basic tasks
- sequencing T_1 : PC $\rightarrow$ MAR (yeh cheez batayegi control unti)
- execution PC_{out} MAR_{in}

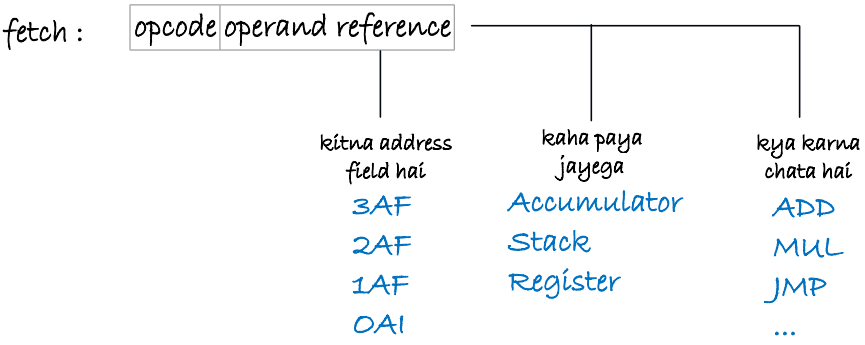
control unit :
control unit is the supervisior in the system that control each and every activity.

control unit takes several input but produce control signals and these control signals are required to perform the operations

(i) control signals are implemented in control unit. (control unit control signal ko generate karti hai)
(ii) control signals are required to execute the micro operation.
(iii) micro operation is the elementary operation in the hardware.
(iv) control unit generated the sequence of control signal.
(v) control signal are directly executed on a base hardware (H/W)
so, hardware generate the desired response.
computer system functionally is program execution.



IR mai aya because usme format pre-defined hota hai

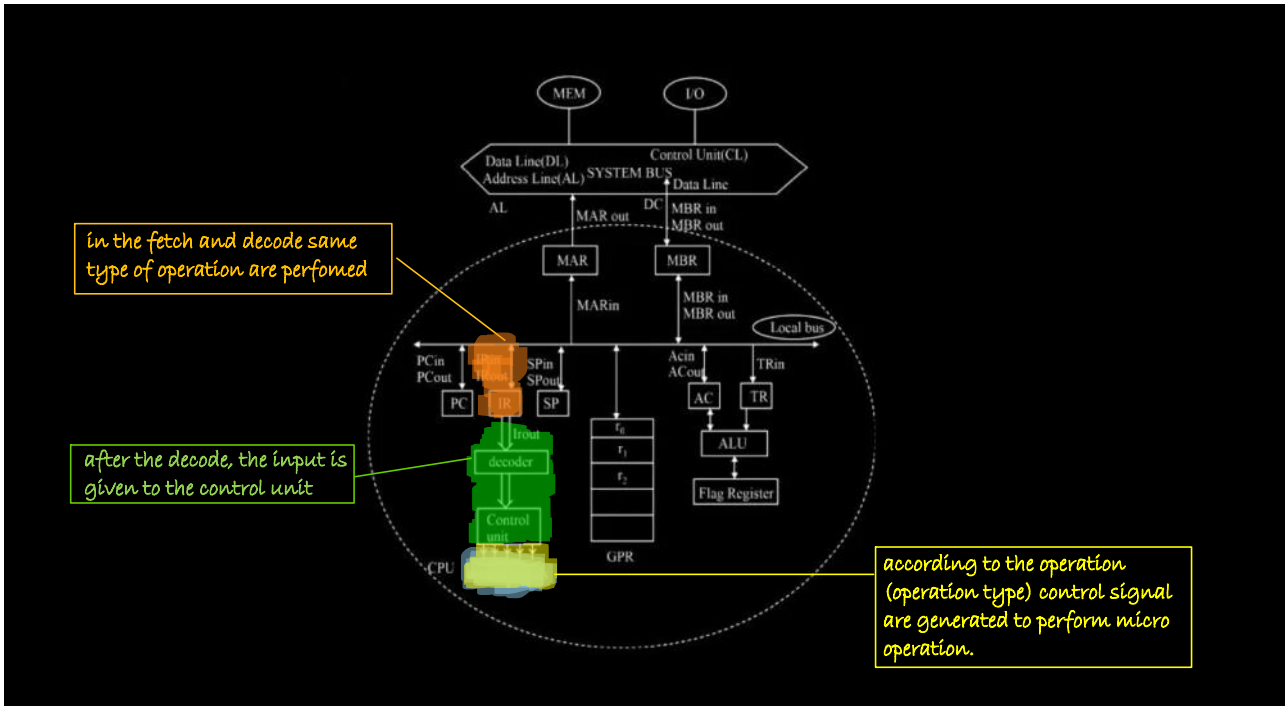


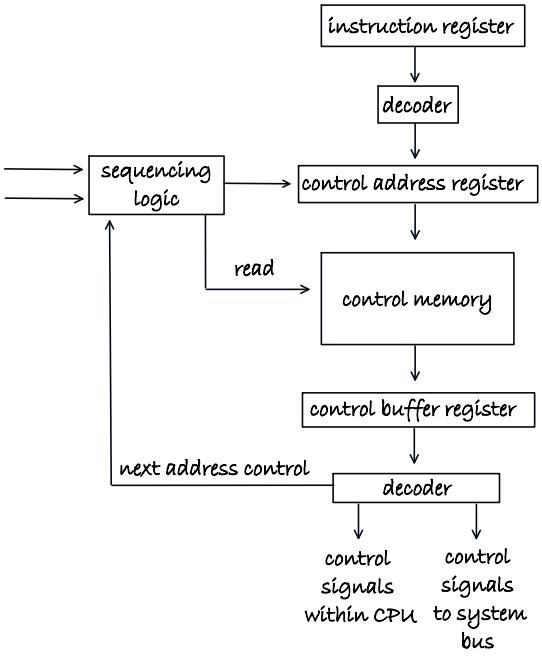
memory mai hai operand and then we studied addressing mode, there are 11 types of addressing modes and then data laane ka working humne working of register aur MUX padha (ALU data path) for syntax and format we studied micro operation and for fetch cycle we studied micro program and the working of execution will be done by control unit through control signals that generate control words.

note : in the fetch and decode same type of operation are perfomed. after the decode, the input is given to the control unit, according to the operation (operation type) control signal are generated to perform micro operation.

fetch cycle ⟶memory ⟶IR

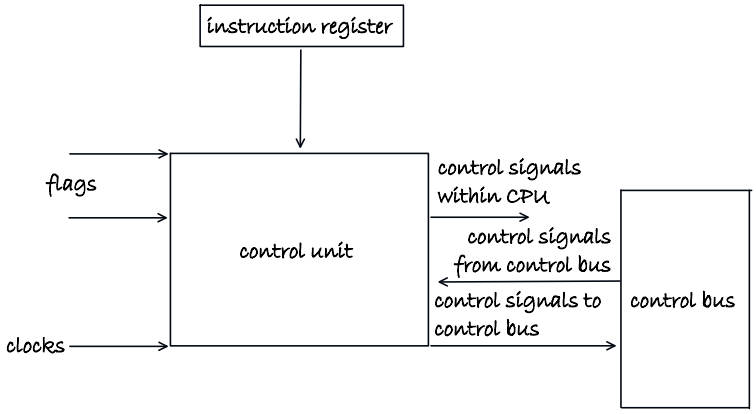
then IR ⟶ decoder ⟶control unit





bus

functioning of microporgrammed control unit



block diagram of control unit

topic : micro operations and control signal

	micro-operations	active control signals
fetch	T_1 : MAR ← PC (or) PC → MAR	C_2
	T_2 : MBR ← memory (or) PC ← (PC) + 1	C_5, C_R
	T_3 : IR ← (MBR)	C_4
indirect	T_1 : MAR ← IR (address)	C_8
	T_2 : MBR ← memory	C_5, C_R
	T_3 : IR(address) ← MBR(address)	C_4
interrupt	T_1 : MAR ← PC	C_1
	T_2 : MAR ← save address PC ← routine address	
	T_3 : memory ← MBR	C_{12}, C_W

whatever component we have (register, ALU, MUX, bus, etc) we use control word for it.

control unit : PC → MAR ; PC_{out} MAR_{in}

PC_{in}	PC_{out}	MAR_{in}	MAR_{out}	IR_{in}	IR_{out}	MBR_{in}	MBR_{out}	AC_{in}	AC_{out}	MUX	ALU	GPR	SP_{in}	SP_{out}
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

C_2 : one micro operation

control unit : memory → MBR ; $memory_{out}$ MBR_{in}

control unit : memory → MBR ; $memory_{out}$ MBR_{in}

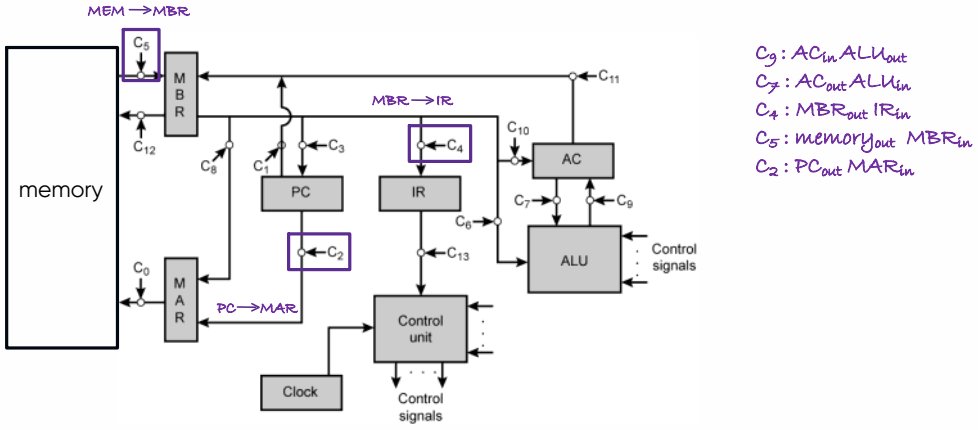
PC_{in}	PC_{out}	MAR_{in}	MAR_{out}	IR_{in}	IR_{out}	MBR_{in}	MBR_{out}	AC_{in}	AC_{out}	MUX	ALU	GPR	SP_{in}	SP_{out}
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

C_5 : one micro operation

control unit : MBR → IR ; MBR_{out} IR_{in}

PC_{in}	PC_{out}	MAR_{in}	MAR_{out}	IR_{in}	IR_{out}	MBR_{in}	MBR_{out}	AC_{in}	AC_{out}	MUX	ALU	GPR	SP_{in}	SP_{out}
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0

C_4 : one micro operation



C_9 : AC_{in} ALU_{out}
C_7 : AC_{out} ALU_{in}
C_4 : MBR_{out} IR_{in}
C_5 : $memory_{out}$ MBR_{in}
C_2 : PC_{out} MAR_{in}

how many control lines are present in hardware[S_0, S_1, S_2...]

how many instruction are implemented in hardware[I_1, I_2, I_3...]

how many micro operation are required for each instruction [T_1, T_2, T_3...]

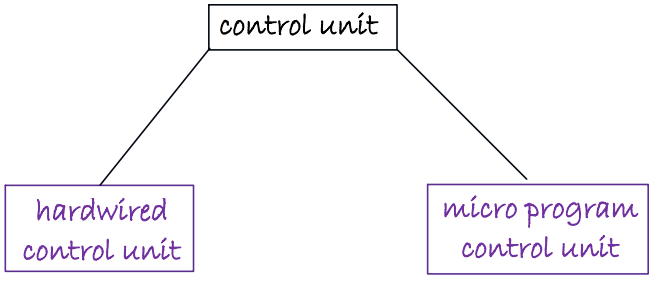
what the control signal required for each micro operation for each instruction

fetch	execute	indirect AM
T_1 PC → MBR	T_1 IR(AF) → MAR	T_1 IR(AF) → MAR
T_2 M(MAR) → MBR	T_2 M(MAR) → MBR	T_2 M(MAR) → MBR
T_3 MBR → IR	T_3 MBR → AC/ALU	T_3 MBR → M(MAR)
		T_3 M(MAR) → MBR
		T_3 MBR → AC/ALU

working : control unit generate the control signals and at the control unit design time, designer decides which control signal are generated in which cycle (T_1, T_2, T_3..) of different-different instructions and that will be stored in a table.

after that we make a boolean (logic) function for implementation.

control signal will be implemented into the control unit by using following approach :

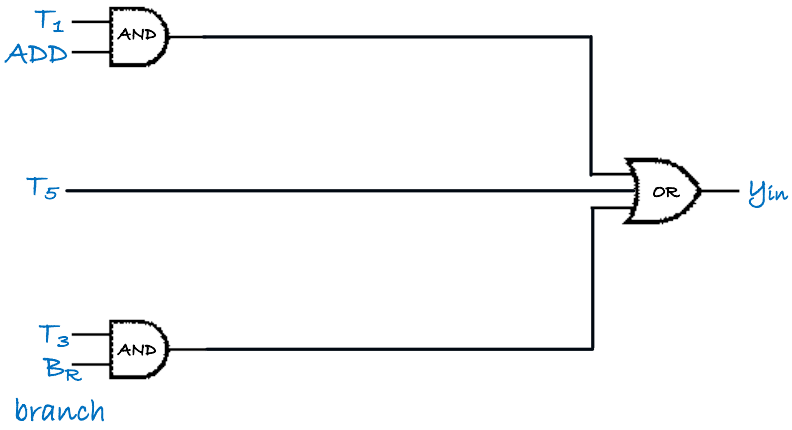


(i) hardwired control unit : in hardwired control unit, control signal are expressed in the S.O.P (sum of product) form and they are directly realized on the hardware.

- in the hardwired control unit they use fixed logic circuit to interpret the instruction then generate the control signal.

advantage : hardwired control unit is the fastest control unit
RISC is the hardwired control unit.

disadvantage : - it is not flexible,
- minor modification require designing and rewiring.
- it does not support new operations (once designed).



- here y_{in} is enabled
- during T_1 for add instruction
- during T_3 for branch instruction
- during T_5 for all instruction

$y_{in} = T_1.ADD + T_3.Branch + T_5$

#Q. Consider the following hypothetical CPU which uses 3 data register A, B & C and supports 3 instruction I1, I2 & I3 . Obtain the logic function that will generate the hardwired control for the signal Ain & Bout with the following data.

	I_1	I_2	I_3
T_1	Ain, Bout	Ain, Cin, Bout	Bin, Bout
T_2	Bin, Cin, Aout	Ain, Aout	Ain, Bin, Cout
T_3	Bin, Bout	Bin, Bout	Bin, Bout
T_4	Cin, Aout	Bin, Aout	Ain, Aout
T_5	End	End	End

Step 1 : Search where the control signals Ain & Bout are present.

	I_1	I_2	I_3
T_1	Ain, Bout	Ain, Cin, Bout	Bin, Bout
T_2	Bin, Cin, Aout	Ain, Aout	Ain, Bin, Cout
T_3	Bin, Bout	Bin, Bout	Bin, Bout
T_4	Cin, Aout	Bin, Aout	Ain, Aout
T_5	End	End	End

Step 2 : Options are in I.T format or T.I format.

Step 3 : For any particular time interval. Is the control signal presents for all the instructions?

$$A_{in} = T_1 I_1 + T_1 I_2 + T_2 I_2 + T_2 I_3 + T_4 I_3$$
$$A_{in} = T_1 (I_1 + I_2) + T_2 (I_2 + I_3) + T_4 I_3$$

$$B_{out} = T_1 I_1 + T_1 I_2 + T1 I_3 + T_3 I_1 + T_3 I_2 + T_3 I_3$$
$$B_{out} = T_1 (I_1 + I_2 + I_3) + T_3 (I_1 + I_2 + I_3)$$
$$B_{out} = T_1 + T_3$$

$$(I_1 + I_2 + I_3) = 1$$
$$(I_1 + I_2 + I_3) = 1$$

T_1 mai sab kaam kar raha so 1

#Q. A hardwired CPU use 10 control signals SI to S10 in various time steps T1 to T5 implement 4 instructions I1 to I4 as shown below.

	T1	T2	T3	T4	T5
I1	S1, S3, S5	S2, S4, S6	S1, S7	S10	S3, S8
I2	S1, S3, S5	S8, S9, S10	S5, S6, S7	S6	S10
I3	S1, S3, S5	S7, S8, S10	S2, S6, S9	S10	S1, S3
I4	S1, S3, S5	S2, S6, S7	S5, S10	S6, S9	S10

Which of the following pairs of expressions represent the circuit for generating control signals S5 and S10 respectively [(IJ + Ik) Tn indicates that the control signal should be generated in time step Tn if the instruction being executed is [IJ to IK]?

A. S5 = Tl + I2.T3 and S10 = (I1 + I3).T4 + (I2 + I4).T5

B. S5 = Tl + (I2 + I4).T3 and S10 = (I1 + I3).T4 + (I2 + I4).T5

C. S5 = T1 + (I2 + I4).T3 and S10 = (I2 + I3 + I4).T2 + (I1 + I3).T4 + (I2 + I4).T5

D. S5 = T1 + (I2 + I4).T3 and S10 = (I2 + I3).T2 + I4.T3 + (I1+I3).T4 + (I2+I4).T5

	T1	T2	T3	T4	T5
I1	S1, S3, S5	S2, S4, S6	S1, S7	S10	S3, S8
I2	S1, S3, S5	S8, S9, S10	S5, S6, S7	S6	S10
I3	S1, S3, S5	S7, S8, S10	S2, S6, S9	S10	S1, S3
I4	S1, S3, S5	S2, S6, S7	S5, S10	S6, S9	S10

Step 2 : Options are in I.T format or T.I format.

Step 3 : For any particular time interval. Is the control signal presents for all the instructions?

$$S5 = T_1 I_1 + T_1 I_2 + T_1 I_3 + T_1 I_4 + T_3 I_2 + T_3 I_4$$
$$S5 = T_1 (I_1 + I_2 + I_3 + I_4) + T_3 (I_2 + I_4)$$

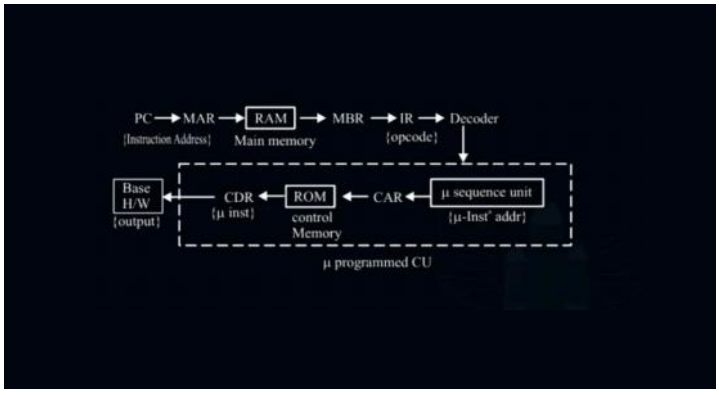
Step 3 : For any particular time interval. Is the control signal presents for all the instructions?

$S5 = T_1 I_1 + T_1 I_2 + T_1 I_3 + T_1 I_4 + T_3 I_2 + T_3 I_4$
$S5 = T_1 (I_1 + I_2 + I_3 + I_4) + T_3 (I_2 + I_4)$
$S5 = T_1 + T_3 (I_2 + I_4)$

$S10 = T_2 I_2 + T_2 I_3 + T_3 I_4 + T_4 I_1 + T_4 I_3 + T_5 I_2 + T_5 I_4$
$S10 = T_2 (I_2 + I_3) + T_3 I_4 + T_4 (I_1 + I_3) + T_5 (I_2 + I_4)$

$(I_1 + I_2 + I_3 + I_4) = 1$

(i) microprogrammed control unit : in microprogrammed control unit, control words are stored in the control memory
then according to the type of operations control signals are generated.
control memory is associated with CAR (control memory address) and CDR (control data register) to contain the control memory address and data respectively.



format of micro instruction:

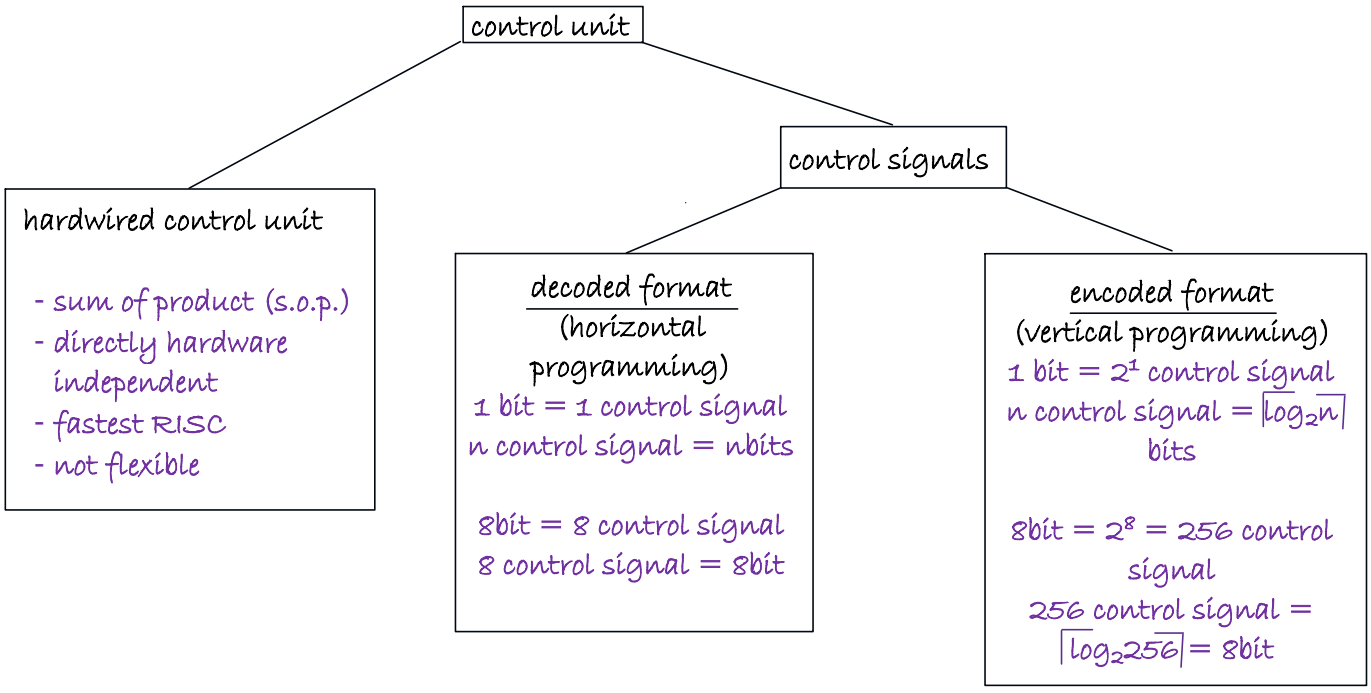
micro instruction / control field/ CDR

branch conditions (BC)	flags	control field	address field/NIA/CAR/CMA

how to find bits : $\lceil \log_2 BC \rceil$ $\lceil \log_2 flag \rceil$ $\lceil \log_2 Cm\ size \rceil$

NIA : next instruction address
CAR : control address register
CMA : control memory address

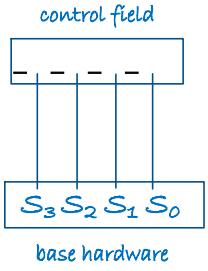
note : control fields depends upon control signals



(i) horizontal programming :

(1) number of control signals in the hardware : $S_0\, S_1\, S_2\, S_3$

(2) decoded format of control signal :



0 : disable
1 : enable

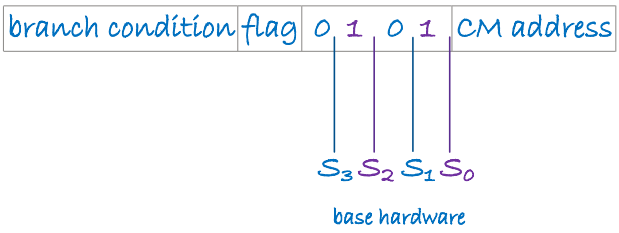
4 control signal on hardware then control field will be 4 bits

(3) design a horizontal micro instruction for control signal (cs) $[S_0\, S_2]$

control field

BC	flag	0 1 0 1	CM address

(4) operational state



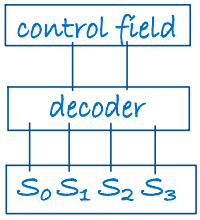
base hardware par S_0 and S_2 active ho jayega

(ii) vertical programming :

(1) number of control signals in the hardware : $[S_0\ S_1\ S_2\ S_3]$

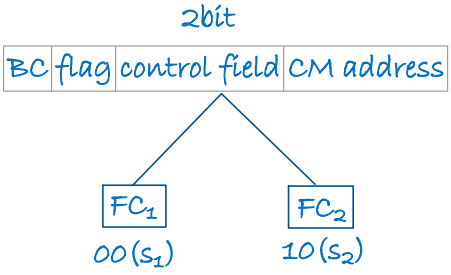
(2) encoded format of control signal = $\lceil \log_2 4 \rceil$ = 2bit control field

00 : S_0
01 : S_1
10 : S_2
11 : S_3



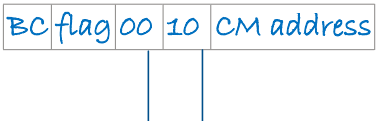
in vertical micro programing external decoder is required

(iii) design a vertical micro insruction for control signal : $[S_0\ S_2]$

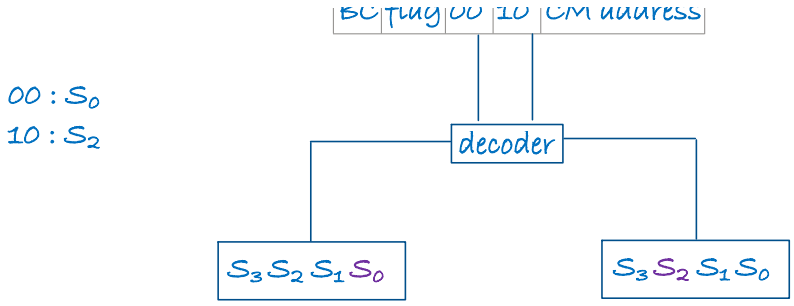


function code (FC) is generated by the control unit to give signal to CPU to perform operation

(iv) operational state :



00 : S_0

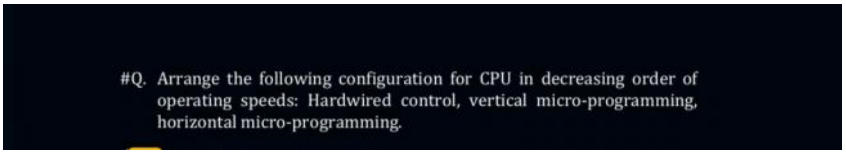


horizontal programming	vertical programming
(i) in this control signal are expressed in decoded format.	(i) in this control signal are expressed in encoded format.
(ii) n bit = n control signal n control signal = n bit 200 control signal = 200bits	(ii) n bit = 2^n control signal n control signal = $\log_2 n$ bits 200 control signal = 8bits
(iii) larger control word	(iii) smaller control word
(iv) no need of external decoder required to generate control signal	(iv) external decoder is required to generate the control signal.
(v) it is more flexibe compared to hardwired.	(v) it is more flexible compared to horizontal.
(vi) it support high degree of parallerlism (none/more than one operation)	(vi) it support low degree of parallelism (none/one operation)

note: default microprogram control unit is vertical micro program control unit used in CISC

speed : hardwired > horizontal > vertical

flexibility : vertical > horizontal > hardwired



#Q. Arrange the following configuration for CPU in decreasing order of operating speeds: Hardwired control, vertical micro-programming, horizontal micro-programming.

A. Hardwired control, vertical micro programming, horizontal micro programming.

B. Hardwired control, horizontal micro programming, vertical microprogramming

C. Horizontal micro programming, vertical micro programming. Hardwired control

D. Vertical micro programming, horizontal micro programming, hardwired control

#Q. Horizontal microprogramming.

A. Does not require use of signal decoders

B. Results in larger sized microinstructions than vertical microprogramming

C. Use one bit each control signal

D. All of the above

#Q. Consider a Hypothetical CU it has 1024 control words memory. Is support 48 control signals & 16 Flags. What is the size of the control words in bits & control memory in byte using

(i) Horizonal Programming?

(ii) Vertical Programming?

control memory : 1024 control word
1024 control word : 2^{10} control words
AF/NIA/CAR : 10bit
16 flag : 4bit

horizontal microprogramming	vertical microprogramming
48 control signal = 48bit control word = 4 + 48 + 10 = 62 bit flag | CF | CAR/NIA 4bit 48bit 10bit 1 control word = 62bit control memory = 1024 cw = 1024 x 62bit = 1024 x 62bit / 8 = 8k byte.	48 control signal = $\lceil \log_2 48 \rceil$ = 6bit control word = 4 + 6 + 10 = 20 bit flag | CF | CAR/NIA 4bit 6bit 10bit 1 control word = 20bit control memory = 1024 cw = 1024 x 20bit = 1024 x 20bit / 8 = 3k byte.

#Q. An instruction set of a processor has 125 signals which can be divided into 5 groups of mutually exclusive signals as follows:

Group 1 : 20 signals. Group 2 : 70 signals, Groups 3 : 2 signals. Groups 4 : 10 signals, Groups 5 : 23 signals.

How many bits of the control words can be saved by using vertical microprogramming over horizontal microprogramming?

A. 0

B. 103

C. 22

D. 55

	horizontal	vertical
G_{11} : 20 cs	20 bit	5 bit
G_{12} : 70 cs	70 bit	7 bit
G_{13} : 2 cs	2 bit	1 bit
G_{14} : 10 cs	10 bit	4 bit
G_{15} : 23 cs	23 bit	5 bit
	125 bit	22 bit

125 - 22 = 103

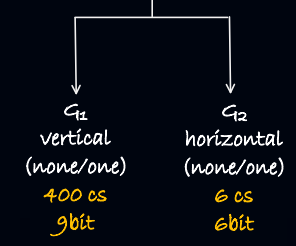
#Q. Control field of a micro Instruction support 2 groups of control signal in which Group 1 Indicate None/One of 400 control signal & Group 2 (Horizontal) Indicate 6 signals. Hardwire contain 16 Flags & 32 Branch condition.

If CAR Register size is 20 bit then what is CDR in bits & control memory in bits?

32 branch : 5 bit
AF/NIA/CAR : 20bit
16 flag : 4bit

micro instruction / control field/ CDR

branch conditions (BC)	flags	control field	address field/NIA/CAR/CMA
5bit	4bit		20bit



control field : 15 bit

micro instruction / control field/ CDR

branch conditions (BC)	flags	control field	address field/NIA/CAR/CMA
5bit	4bit	15bit	20bit

control word : 5 + 4 + 15 + 20 = 44 bit

control memory : 2^{20} control words
: 2^{20} x 44 bits

#Q. Consider a CPU where all the instructions require 7 clock cycles to complete execution. There are 140 instructions in the instruction set. It is found that 125 control signals are needed to be generated by the control unit. While designing the horizontal micro-programming control unit, single address field format is used for branch control logic. What is the minimum size of the control word and control address register?

[GATE IT 2008]

A 125, 7
B 125, 10
C 135, 7
D 135, 10

125 control words : 125 bits

total number of instructions : 140
number of cycles per instruction : 7 cycle ($T_1, T_2 ... T_7$)
total number of micro operations / instructions : 140 x 7 = 980 cw

control memory: 980 control words

CAR/AF : $\log_2 980$ = 10bit

horizontal

control field	address field/NIA/CAR/CMA
125bit	10bit

control word : 125 + 10 = 135 control words

CAR/AF : $\lceil \log_2 980 \rceil$ = 10bit

RISC Reduced Instruction set computer	CISC Complex Instruction set computer
1. It support less number of addressing Mode (AM)	1. It support more number of AM.
2. It support smaller Instruction set	2. It support larger Instruction set.
3. It support more number of Register	3. It support less number of Register
4. It support fixed length Instruction	4. It support variable length Instruction
5. It support 1 Instruction per cycle (CPI=1) (Cycle per Instruction =1)	5. It support number 1 Instruction Per cycle (CPI + 1)
6. It support pipeline successfully	6. It support unsuccessful Pipeline
7. It is the expensive processor used in Real Time application	7. It is the low expensive processor
8. It is a super computer	8. General Purpose computer
9. It uses hardwired control unit. (Motorola processor, power processer, ARM processor)	9. It uses microprogrammed (vertical) control unit (Pentium processer)

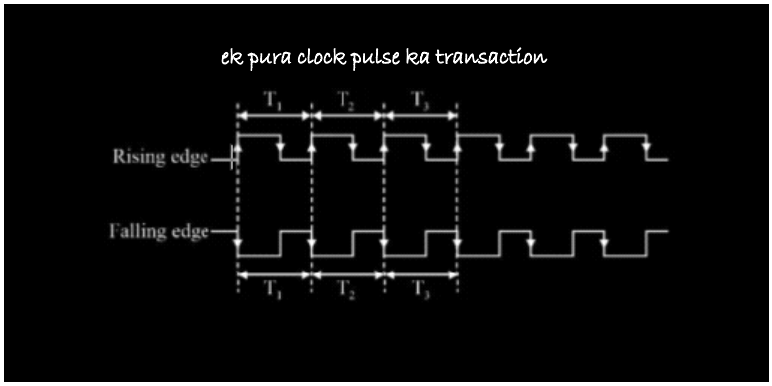
pre-requisites for pipelining

topic : CPU time calculation

(i) CPU time calculation program execution time
(ii) program execution time is calculated based on clock.
(iii) processor contain clock pins and these clock pin is externally connected with the clock generator
(iv) so in the computer system all the operation are controlled by the clock, so CPU contain pints which is externally connected with clock generator.
(v) clock generator is operating with a constant frequency to generate the clock pulse (clock signal)

Program ET (execution time) is calculated based on 2 factor :

(i) cycle ($T_1, T_2 \ldots T_n$) : cycle is defined as clock pulse transition either from rising edge to rising edge or falling edge to falling edge.



(ii) cycle time : the time required to transfer the pulse either form rising edge to rising edge or falling edge to falling edge is called cycle time

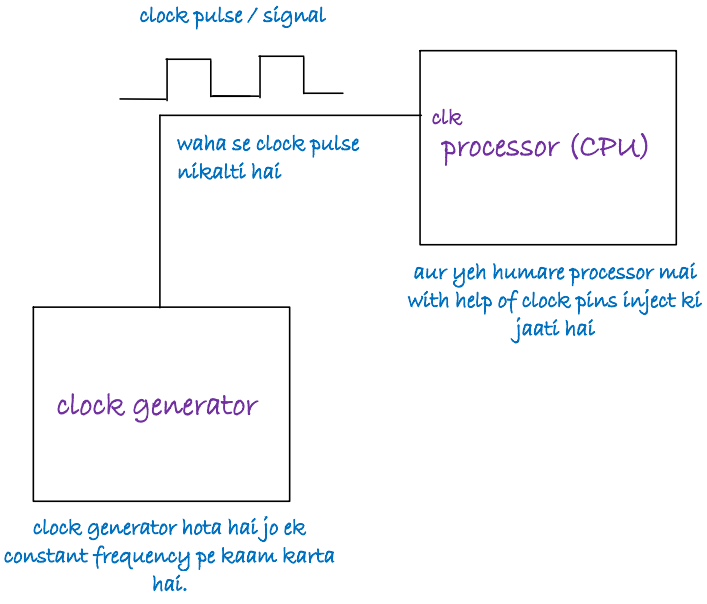
cycle time depend on clock frequency

$$\text{cycle time} \propto \frac{1}{\text{clock frequency}}$$

my computer properties :
64bit processor (word length = 64bit) (operation perfomed on 64bits)
8GB RAM (AL)
ITB hardisk (2^{40} byte hardisk)

what is 2GHZ processor ?

clock rate/frequence 2GHZ ki hai



1GHZ ↑ constant frequency

cycle time - $\frac{1}{\text{clock frequency}}$

- 1/1GHZ

- example :

1GHZ processor
clock frequency : 1GHZ
cycle time : 1/1GHZ
: $1/10^9$
: 10^{-9} second
cycle time : 1 nano second.

- example :

2GHZ processor
clock frequency : 2GHZ
cycle time : 1/2GHZ
: $1/2 \times 10^9$
: 0.5×10^{-9} second
cycle time : 0.5 nano second.

some conversions :

$2^{10} = 1k = 10^3$ ≡ 1mili second = 10^{-3} second
$2^{20} = 1m = 10^6$ ≡ 1micro second = 10^{-6} second
$2^{30} = 1g = 10^9$ ≡ 1nano second = 10^{-9} second

$2^{40} = 1t = 10^{12}$
$2^{50} = 1p$
$2^{60} = 1e$

q. CPU has 1 GHZ processor and program P_1 having 100 instruction and each instruction takes 5 cycle then what is the program execution time?
(i) in cycle
(ii) in time

1GHz processor
cycle time : 1 nano second

program P_1 : 100 instructions (number of instructions : instruction count (IC))
each instruction : 5 cycles (CPI (cycle per instruction))
program execution time (in cycle) : 100 x 5 = 500 cycle.
program execution time (in time) : 500 cycle = 500 x 1 nsec = 500nsec.

q. CPU has 2 GHZ processor and program P_1 having 100 instruction and each instruction takes 5 cycle then what is the program execution time?
(i) in cycle
(ii) in time

2GHz processor
cycle time : 0.5 nano second

program P_1 : 100 instructions (number of instructions : instruction count (IC))
each instruction : 5 cycles (CPI (cycle per instruction))
program execution time (in cycle) : 100 x 5 = 500 cycle.
program execution time (in time) : 500 cycle = 500 x 0.5 nsec = 250nsec.

q. CPU has 4GHZ processor and program P_1 having 100 instruction and each instruction takes 5 cycle then what is the program execution time?
(i) in cycle
(ii) in time

4GHz processor
cycle time : 0.25 nano second

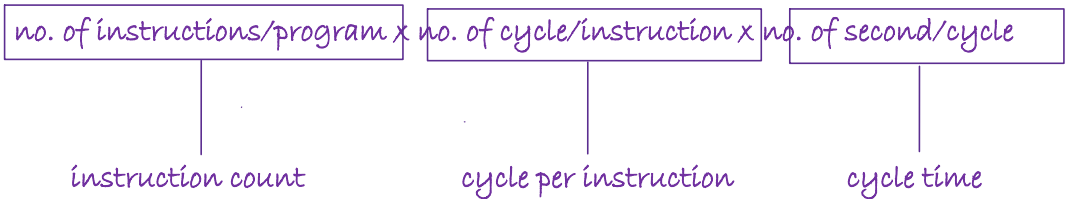
program P_1 : 100 instructions (number of instructions : instruction count (IC))
each instruction : 5 cycles (CPI (cycle per instruction))
program execution time (in cycle) : 100 x 5 = 500 cycle.
program execution time (in time) : 500 cycle = 500 x 0.25 nsec = 125nsec

note :
- number of instructions : instruction count (IC)
- CPI : cycle per instruction

concept : CPU time calculation

CPU time means program execution time

Program execution time = number of seconds / program



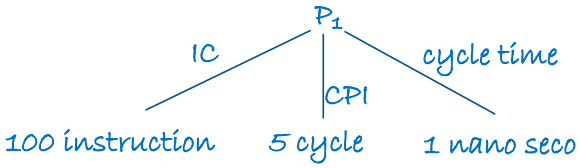
program execution time (CPU time) = IC x CPI x cycle time

q. CPU has 1 GHZ processor and program P_1 having 100 instruction and each instruction takes 5 cycle then what is the program execution time?
(i) in cycle

(ii) in time

1GHz processor
cycle time : 1 nano second

program P_1 : 100 instructions (number of instructions : instruction count (IC))
each instruction : 5 cycles (CPI (cycle per instruction))
program execution time (in cycle) : 100 x 5 = 500 cycle.
program execution time (in time) : 500 cycle = 500 x 1 nsec = 500nsec.



program execution time : IC x CPI x cycle time

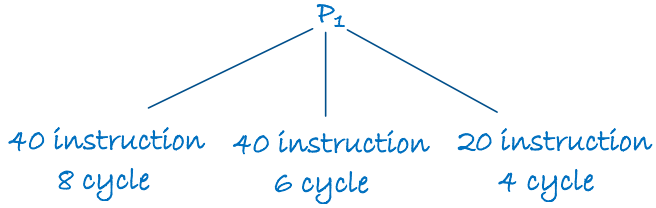
program execution time : 100 x 5 x 1 = 500 nano second

CPU time calculation
Program is a combination of data transfer, data manipulation and transfer of control instruction. different instruction takes (consume) different cycle to complete the execution. so,

program execution time : $(\sum(IC_i \times CP_i) \times$ cycle time)

i : type of instructions

modification : cycle time = 1nano sec



program execution time : $(\sum(IC \times CPI) \times$ cycle time)

= ((40 x 8) + (40 x 6) + 20 x 4) x 1 nsec
= (320 + 240 + 80) x 1nsec
= 640 x 1 nsec
= 600 nsec

q. consider a CPU with clock frequency (rate) of 400MHZ and if the CPU has average CPI 6

then average execution time is?

sol. cycle time : $\frac{1}{400 \times 10^6} = \frac{1}{4} \times 10^{-8} = \frac{1}{4} \times \frac{10}{10} \times 10^{-8}$

$= 2.5 \times 10^{-9}$
$= 2.5$ nsec

average instruction execution time : CPI x cycle time
: 6 x 2.5 nsec
: 15 nsec

q.consider a CPU operate at (run at) 800MHZ clock rate and iexecuting a program consiting 4000 instruction. if each instruction taking CPI 5 then total program execution time (CPU time)?

sol. cycle time : $\frac{1}{800 \times 10^6} = \frac{1}{8} \times 10^{-8} = \frac{1}{8} \times \frac{10}{10} \times 10^{-8}$

$= 1.25 \times 10^{-9}$ sec
$= 1.25$ nsec

Program ET : IC x CPI x cycle time
: 4000 x 5 x 1.25nsec
: 25000 nsec

#Q. Consider a 1.5 GHz clock frequency processor used to execute the following program segment

Instruction type	Instruction count [IC]	CPI
Load	300	11
Store	200	9
Arithmetic	250	7
Shift	150	6
Branch	50	4
Total	950	

q. what is average instruction execution time?
q. what is the MIPs rate?
q. what is total program execution time?

a. what is average instruction execution time?
(300 x 11) + (200 x 9) + (250 x 7) + (150 x 6) + (50 x 4) = 7,950
7950 / 950 = 8.3684

cycle time : 1/1.5 GHZ = (1/1.5) x 10^{-9} = 0.66 nano second

average instruction ET : 8.36 x 0.66 = 5.5176 nano second

b. what is the MIPs rate?
1 instruction ET : 5.51 x 10^{-9} second

average instruction ET : 8.36 x 0.66 = 5.5176 nano second

b. what is the MIPs rate?
1 instruction ET : 5.51×10^{-9} second
in 1 second, how many total number of instruction executed?
in 1 second : $1 / 5.51 \times 10^{-9}$ instruction or $1/5.51 \times 10^{9}$ instruction /sec
: 0.1814×10^{9} instruction / second
: 181.4×10^{6} instruction / second
: 181.4 MIPS

c. what is total program execution time?
total program ET : number of instruction in program x avg. instruction ET
: 950 x 5.51
: 5234.5×10^{-9}
: 5.234 microsec.

#Q. 1 nsec. clock cycle processor consume 4 cycle for load and store operation and 6 cycle for ALU operation and 2 cycle for branch operation. The relative frequency of these operation are 40%, 40% and 20% respectively.
(i) What is the average instruction ET?
(ii) What is the performance in term of MIPS?
(iii) If program contain 10^6 instruction them what is total program ET?

a. what is average instruction execution time?
(.40 x 4) + (.40 x 6) + (.20 x 2) cycle
= 1.6 + 2.4 + 0.4 = 4.4 cyce.

cycle time : 1 nano second

average instruction ET : 4.4 x 1 = 4.4 nano second

b. what is the MIPs rate?
1 instruction ET : 4.4×10^{-9} second
in 1 second, how many total number of instruction executed?
in 1 second : $1/4.4 \times 10^{-9}$s instruction / second
: $(1/4.4) \times 10^{9}$ instruction / second
: $(1000/4.4) \times 10^{6}$ instruction / second
: 227.2×10^{6} instruction / second
: 227.2 MIPS

c. if program contains 10^6 instruction then what is total program ET?
total program ET : number of instruction in program x avg. instruction ET
: $10^{6} \times 4.4 \times 10^{-9}$ second
: 4.4×10^{-3} second
: 4.4 mili second

note :
super computer : floaps
floating point operation per second

concept : perfomance gain (speed up factor)

perfomance gain (speed up factor) = perfomance of new / perfomance of old

perfomance gain (speed up factor) = ET_{old} / ET_{new} [or] $(1/ET_{new}) / (1/ET_{old})$

same work is done by

- aryan in 10 hours
- vipul in 4 hours

so vipul perfomance is fast.

perfomance is inversely propotional 1/ET

#Q. Consider a 2.3ns clock cycle processor which consume 9 cycle for load and store instruction and 7 cycle for ALU instruction and 3 cycle for branch instruction. Relative frequency of their instruction are 40% , 40% and 20% respectively. Processor is enhanced with an average CPI of 1. During the enhancement, cycle time is increased by 40%, them what is performance GAIN [speed up factor] of new and OLD Design?

old design :

program execution time : $(\sum(IC_i \times CP_i) \times$ cycle time)

average instruction ET : [.40 x 9 + .40 x 7 +.20 x 3] 2.3 nsec

: [3.6 + 2.8 + 0.6] x 2.3

average ET_{old} : 16.1 nsec

new design : CPI = 1

cycle time increased by 40%

new cycle time : 2.3 + 40% of 2.3

: 2.3 + 0.92 = 3.22 nsec

program execution time : $(\sum(IC_i \times CP_i) \times$ cycle time)

average instruction ET : [.40 x 1 + .4 x 1 +.2 x 1] 3.22 nsec

: [0.4 + 0.4 + 0.2] x 2.3 = 2.33 nsec

average ET_{new} : 3.22 nsec

perfomance gain (speed up factor) = $\frac{\text{perfomance of new}}{\text{perfomance of old}}$ $\frac{ET_{old}}{ET_{new}}$

$\frac{16.1 \text{ nsec}}{3.22 \text{ nsec}}$

s = 5

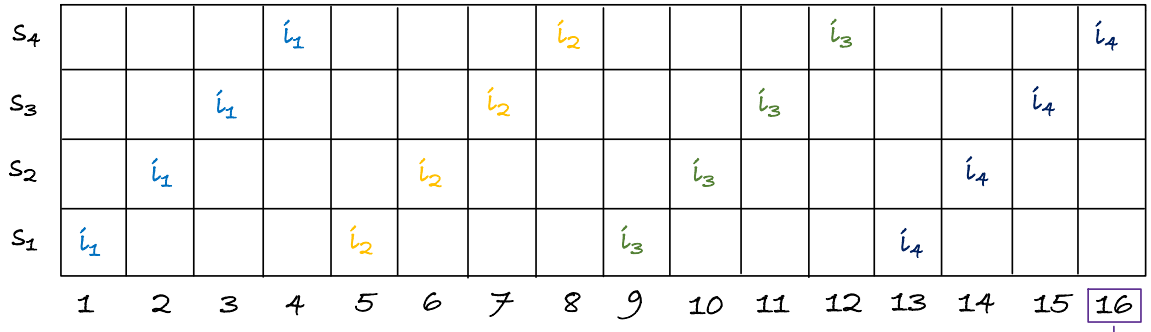
chapter - 5 (pipelining)

Monday, June 24, 2024 5:24 PM

instruction pipelining

non - pipelining : new input only accepted after completion of old input (or) accepting new input after previously accepted input appearas a output at the other end, in non pipelining non overlapping execution.

output end



input end

cycle

sabhi cheeze common clock pulse se control ho rahi hai

total 16 cycles

s : stage

- clock 1 par i_1 stage 1 mai execute ho raha hai
- clock 2 par i_1 stage 2 mai execute ho raha hai
- clock 3 par i_1 stage 3 mai execute ho raha hai
- clock 4 par i_1 stage 4 mai execute ho raha hai

.

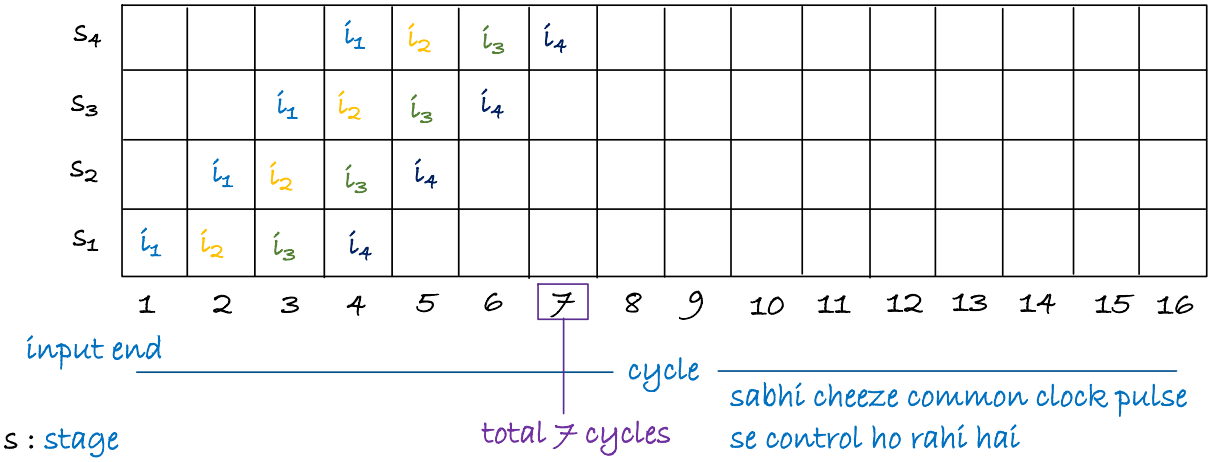
.

- clock 15 par i_4 stage 3 mai execute ho raha hai
- clock 16 par i_4 stage 4 mai execute ho raha hai

pipelining : pipelining is a mechanism which is used to improve the perfomance of the system in which task (instruction) are executed in overlapping (parallel) manner.

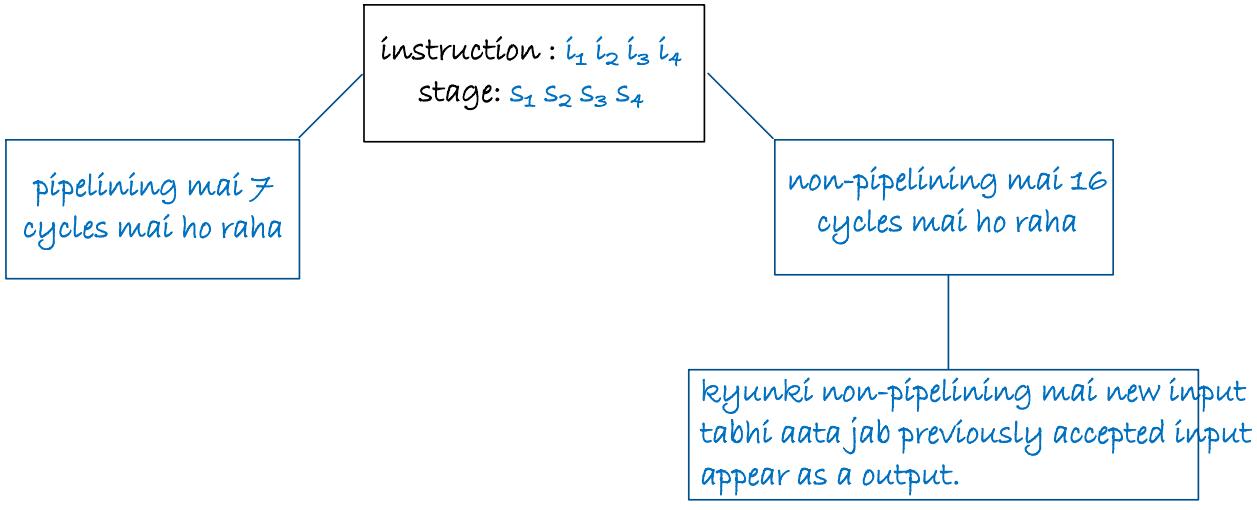
- pipelining is decomposition technique that means problem is divided into sub problem as assign the sub problem to the pipes then operate the pipe under the same clock

output end i_1 ends at 4

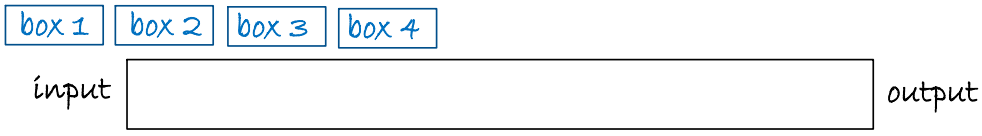


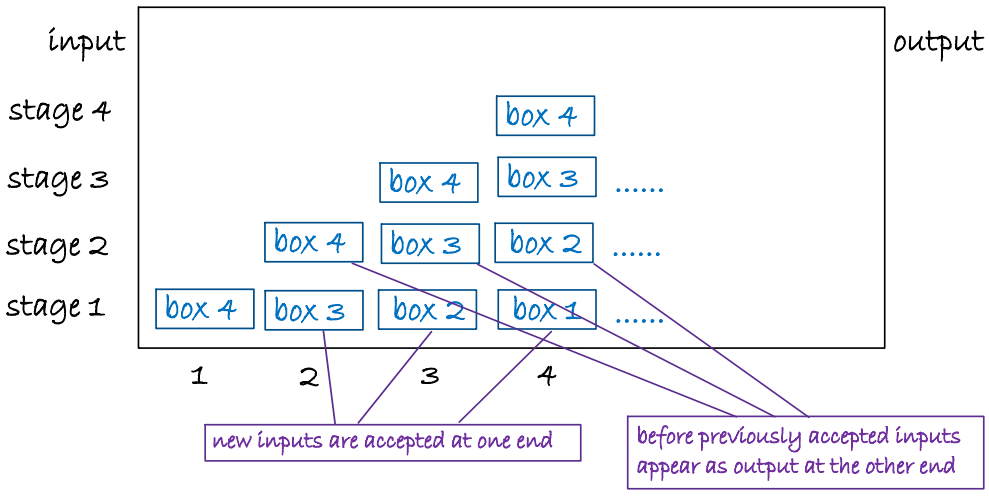
- clock 1 par i_1 stage 1 mai execute ho raha hai
- clock 2 par i_1 stage 2 mai aur i_2 stage 1 mai execute ho raha hai
- clock 3 par i_1 stage 3 mai, i_2 stage 2 mai aur i_3 stage 1 mai execute ho raha hai
- clock 4 par i_1 stage 4 mai, i_2 stage 3 mai, i_3 stage 2 mai aur i_4 stage 4 mai execute ho raha hai
- clock 5 par i_2 stage 4 mai, i_3 stage 3 mai aur i_4 stage 2 mai execute ho raha hai
- clock 6 par i_3 stage 4 mai aur i_4 stage 3 mai execute ho raha hai
- clock 7 par i_4 stage 4 mai execute ho raha hai

pipelining means accepting new input at one end before previously accepted input appears as a output at other end. this means new input are executed along with old input in overlapping manner (in a pipeline)

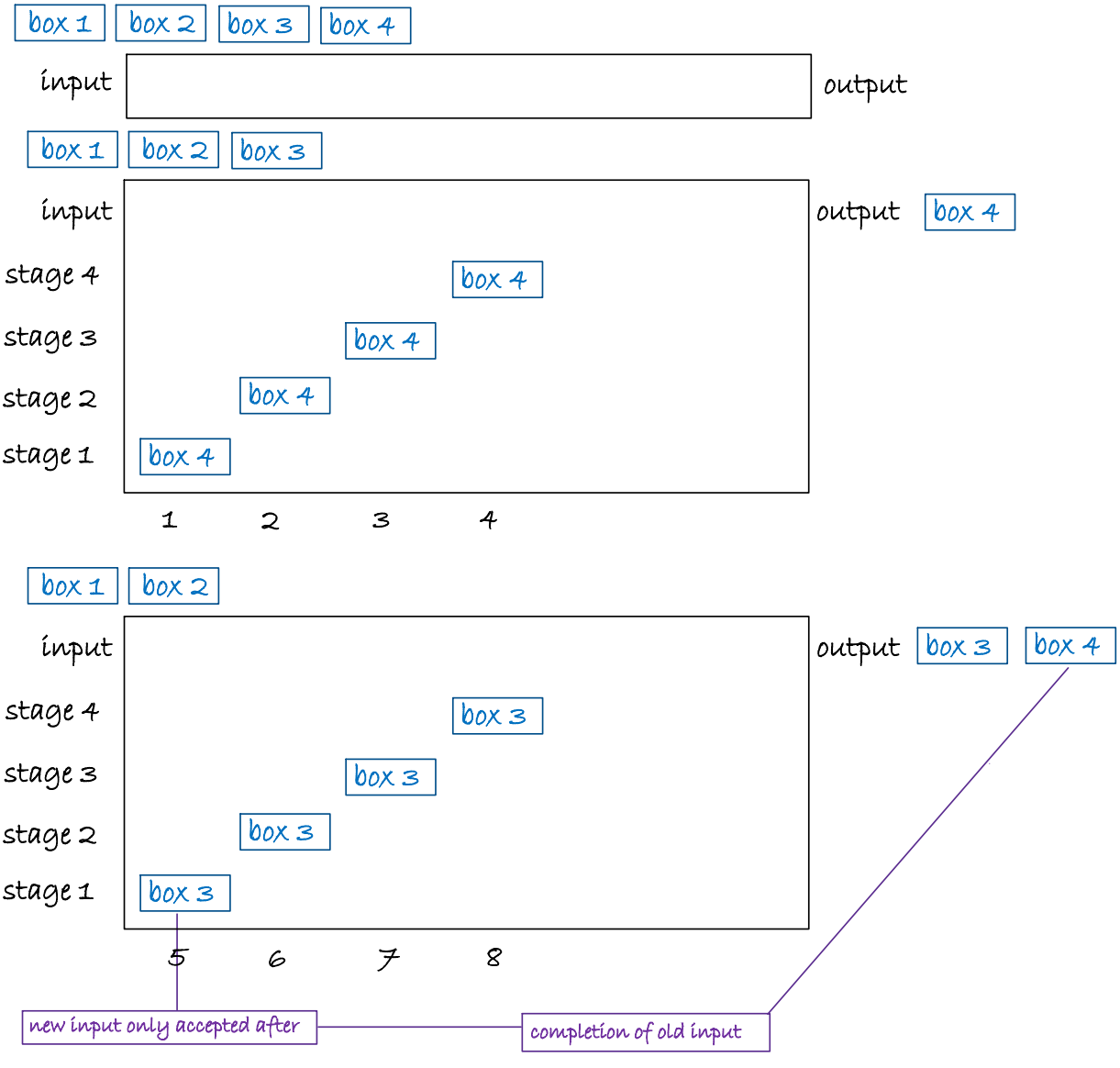


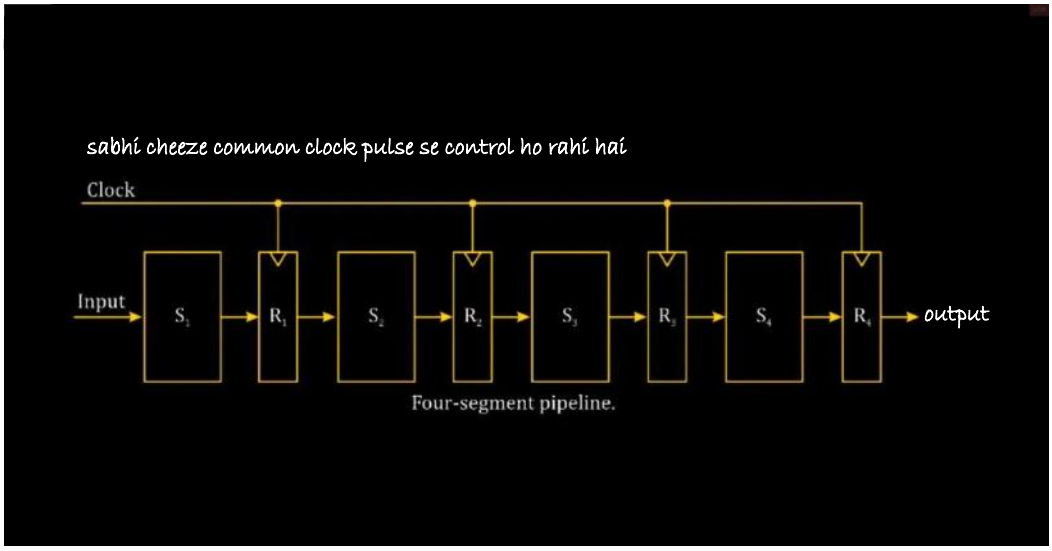
example : pipeline





example : non-pipeline





stage = segment

kisi bhi pipelining ke 2 end hote hai ek hota hai input end aur dusra output end and between these end multiple pipes are interconnected to functioning of pipelining
- these pipes are called stage (or) segment
- between the stages 'buffer' are used to store the intermediate results.
- these buffer is called as pipeline register (or) interface register (or) buffer (or) latch
- all the stages along with the buffer are controlled (or) connected by common clock.

S_1 ka data S_2 mai daal rahe but agar S_2 khali nahi hua then? islie buffer hota hai stages ke beech mai, s_1 ka data s_2 mai daal rahe hai islie s_1 se free karke buffer mai daal dete hai jisse s_1 free ho jaye jisse usme I_2 aajaye.

Execution time calculation / perfomance evaluation

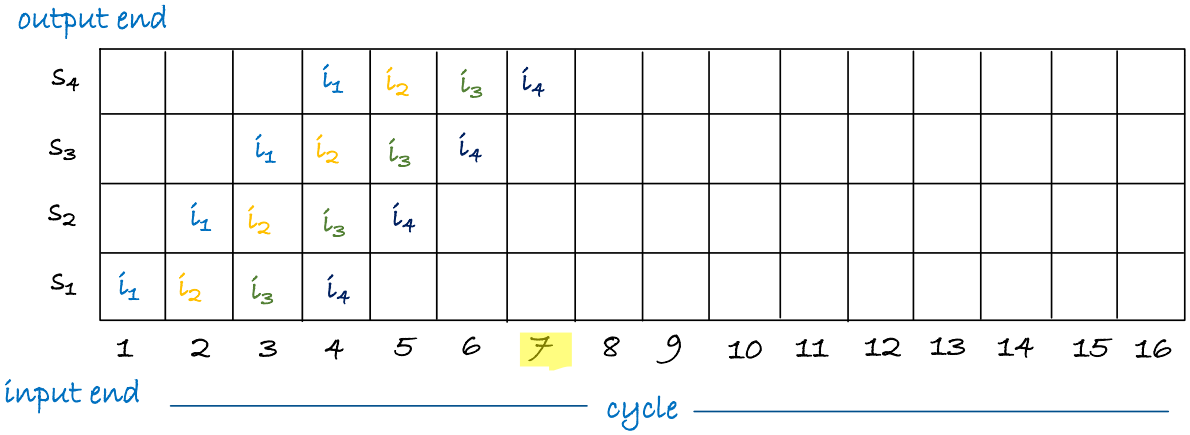
(i) execution time calculation in pipelining :

$ET_{pipelining} = k \times tp + (n-1)\, tp$

$ET_{pipelining} = [k+(n-1)]\, tp$

k : no of stages (segments)
n : no of instructions

tp : each stage delay in pipeline.



k : no of stages (segments) = 4
n : no of instructions = 4
tp : each stage delay in pipeline (assume) = 1 cycle

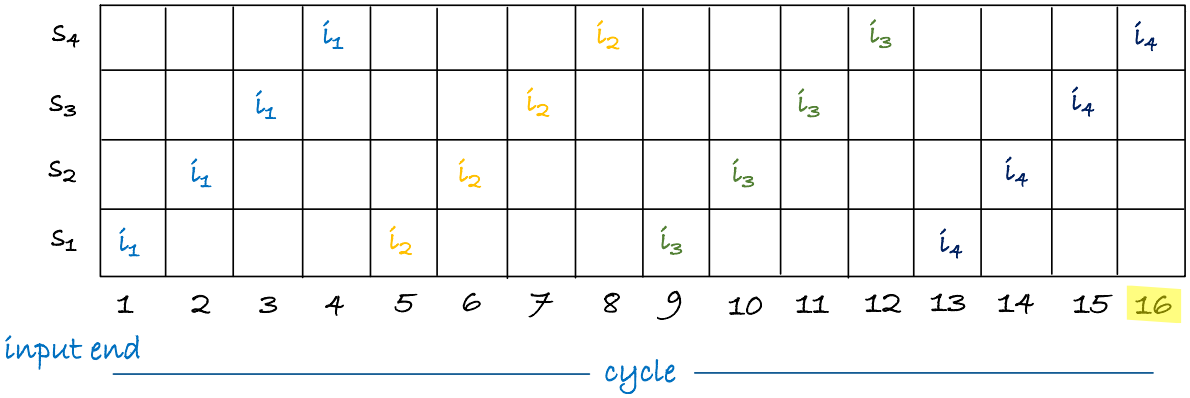
$ET_{pipelining} = [4 + (4-1)]1$
$ET_{pipelining} = [4 + 3]1$
$ET_{pipelining} = 7$ cycle.

(ii) execution time in non-pipelining :

in a non-pipelining if 1 instruction takes t_n time to execute then total time taken to execute n instruction in non pipeline is

$ET_{nonpipeline} = n \times tn$

n : no of instruction
tn : each instruction execution time in non-pipeline



cycle

n : no of instruction = 4
tn : each instruction execution time in non-pipeline = 4

$ET_{nonpipeline} = 4 \times 4$
$ET_{nonpipeline} = 16$ cycle.

perfomance gain of pipeline over non-pipeline :

$$\text{perfomance gain (speed up factor) (s)} = \frac{\text{perfomance of pipeline}}{\text{perfomance of non-pipeline}}$$

$$\text{perfomance} \propto \frac{1}{ET}$$

$$= \frac{1/ET_{pipeline}}{1/ET_{non\text{-}pipeline}}$$

$$\text{perfomance gain (speed up factor) (s)} = \frac{ET_{non\text{-}pipeline}}{ET_{pipeline}}$$

$$\text{perfomance gain (speed up factor) (s)} = \frac{n \times t_n}{[k + (n-1)]\, t_p}$$

when large number of instruction executed (or) n value is not given (or) ideal case

$$\text{perfomance gain (speed up factor) (s)} = \frac{t_n}{t_p}$$

tn : each instruction execution time in non-pipeline
tp : each stage delay in pipeline.

example :

$k = 4, n = 4$ $k = 4, n = 100$ $k = 4, n = 10000$

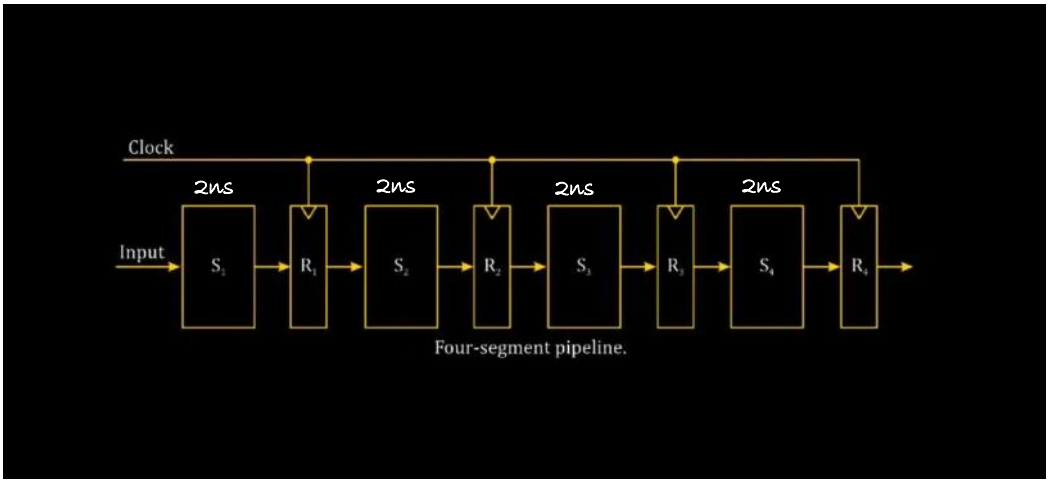
$$(s) = \frac{n \times t_n}{[k + (n-1)]\, t_p} \qquad (s) = \frac{n \times t_n}{[k + (n-1)]\, t_p} \qquad (s) = \frac{n \times t_n}{[k + (n-1)]\, t_p}$$

$$(s) = \frac{4 \times t_n}{[4 + (4-1)]\, t_p} \qquad (s) = \frac{100 \times t_n}{[4 + (100-1)]\, t_p} \qquad (s) = \frac{10000 \times t_n}{[4 + (10000-1)]\, t_p}$$

$$(s) = \frac{4t_n}{7t_p} \qquad (s) = \frac{100t_n}{103t_p} \qquad (s) = \frac{10000t_n}{10003\, t_p}$$

execution time for one instruction :

(i) $ET_{pipeline}$ for 1 instruction
(ii) $ET_{non\text{-}pipeline}$ for 1 instruction



Four-segment pipeline.

perfectly balance (uniform delay) pipeline (same delay lagega)

(i) $ET_{pipeline}$ for 1 instruction

$ET_{pipeline} = [k+(n-1)]\ tp$
$ET_{pipeline} = [4+(1-1)]\ 2$
$ET_{pipeline} = 8nsec$

tp : each stage delay in pipeline
tp : 2nsec

(ii) $ET_{non\text{-}pipeline}$ for 1 instruction

$ET_{non\text{-}pipeline} = n \times tn$
$ET_{non\text{-}pipeline} = 1 \times 8$
$ET_{non\text{-}pipeline} = 8nsec$

tn : each instruction execution time

tn : 2+2+2+2 = 8 nsec

$ET_{non\text{-}pipeline} = k \times tp$
tn : k x tp (only when perfectly balanced)

ideal case (or) when perfectly balanced

when each stage (all stage) are perfectly balanced (or) uniform delay then 1 task ET in no-pipelining is

$ET_{non\text{-}pipeline} = k \times tp$ (only when perfectly balanced)

$k = 4$
$tp = 2$

$ET_{non\text{-}pipelining} = 4 \times 2 = 8$ nsec

$$\text{perfomance gain (speed up factor) (s)} = \frac{t_n}{t_p}$$

$$\text{perfomance gain (speed up factor) (s)} = \frac{k \times t_p}{t_p}$$

$$\text{perfomance gain (speed up factor) (s)} = \text{no. of stages (K)}$$

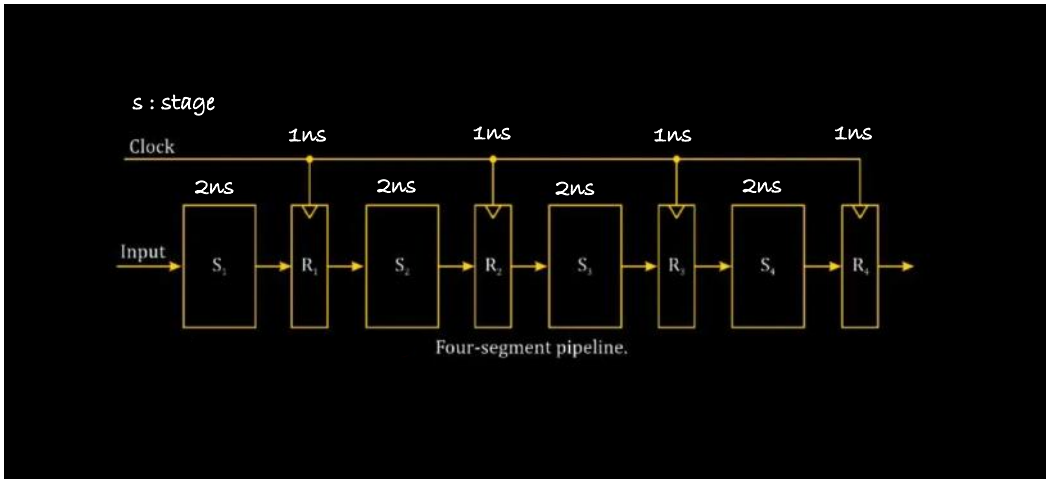
in ideal case when pipeline are perfectly balanced then maximum speed up factor is equal to number of stages in pipeline

types of pipeline :

(i) uniform delay pipeline

(ii) non-uniform delay pipeline

(i) uniform delay pipeline (perfectly balanced) : in uniform delay pipeline each stage taking the same amount of time (delay) to complete the assigned task.



Four-segment pipeline.

(i) $ET_{pipeline}$ for 1 instruction

$ET_{pipeline} = [k+(n-1)]\ tp$

k = 4
n = 1
tp : each stage delay in pipeline
tp : 2nsec
$ET_{pipeline} = [4+(1-1)]2$
$ET_{pipeline} = 8nsec$

(ii) $ET_{non\text{-}pipeline}$ for 1 instruction

$ET_{non\text{-}pipeline} = n \times tn$

n : 1
tn : each instruction execution time
tn : 2+2+2+2 = 8 nsec
$ET_{non\text{-}pipeline} = 1 \times 8$
$ET_{non\text{-}pipeline} = 8nsec$

in uniform delay :
1 task ET in pipeline = 1 task ET in non pipeline

if buffer delay is given or included then :

tp : stage delay + buffer delay

tp : 2 + 1 = 3 nsec

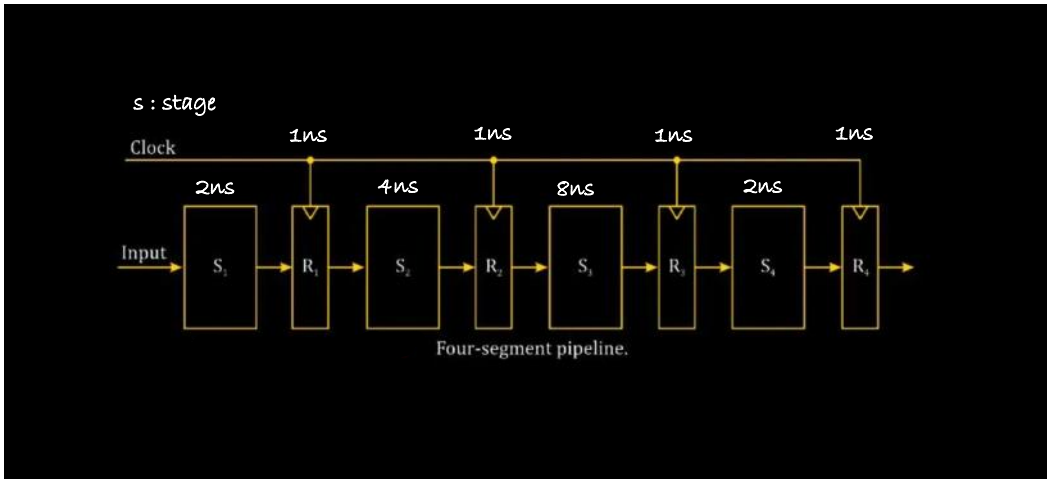
$ET_{non\text{-}pipeline} = k \times tp$
tn : k x tp (only when perfectly balanced)

(ii) non-uniform delay pipelining : in non-uniform delay pipeline each stage taking (maintain) different amount of time to complete the assigned task

tp : max (stage delay)

and if buffer delay is given then :

tp:max(stage delay + buffer delay)



Four-segment pipeline.

(i) $ET_{pipeline}$ for 1 instruction

$ET_{pipeline} = [k+(n-1)]\ tp$

k = 4
n = 1
tp : max (stage delay)
tp : max (2,4,8,2)
tp : 8sec
$ET_{pipeline} = [4+(1-1)]8$
$ET_{pipeline} = 32nsec$

(ii) $ET_{non\text{-}pipeline}$ for 1 instruction

$ET_{non\text{-}pipeline} = n \times tn$

n : 1
tn : each instruction execution time
tn : 2+8+4+2 = 16 nsec
$ET_{non\text{-}pipeline} = 1 \times 16$
$ET_{non\text{-}pipeline} = 16nsec$

if buffer delay is given or included then

if buffer delay is given or included then

tp : stage delay + buffer delay

tp : 2 + 1 = 3 nsec

q. if n = 1000 then ET in non pipeline and pipeline for previous data

(i) $ET_{pipeline}$ for 1 instruction

$ET_{pipeline} = [k+(n-1)]\ tp$

k = 4
n = 1000
tp : max (stage delay)
tp : max (2,4,8,2)
tp : 8sec
$ET_{pipeline} = [4+(1000-1)]8$
$ET_{pipeline} = 8024$nsec

(ii) $ET_{non\text{-}pipeline}$ for 1 instruction

$ET_{non\text{-}pipeline} = n \times tn$

n : 1000
tn : each instruction execution time
tn : 2+8+4+2 = 16 nsec
$ET_{non\text{-}pipeline} = 1000 \times 16$
$ET_{non\text{-}pipeline} = 16000$ nsec

if buffer delay is given or included then

tp : max (stage delay + buffer delay)

tp : max (2+1, 4+1, 8+1, 2+1)

tp : 9nsec

jab perfectly balanced hai toh same time lag raha lekin jab non uniform hai toh different time lag raha hai

important points:

(i) when pipeline are perfectly balanced (uniform delay) then 1 task ET in pipelining is same as 1 task ET in non-pipelining.

(ii) when pipeline are not perfectly balanced (non-uniform delay) then 1 task ET in pipeline is greater than 1 task ET in non-pipeline.

$T_1 \geq T_2$
T_1 : 1 task execution time in pipeline
T_2 : 1 task execution time in non-pipeline

(iii) but in non-uniform delay when number of task (instruction) increase then pipeline perfomance is best in non-uniform delay (or) uniform delay pipeline.

(iv) buffer delay is included only in pipeline and in non-pipeline we are not storing the intermediate result so in non pipeline buffer delay is not included.

Q. 2 4 segment pipeline have the respective stage delay of 10ns, 15ns, 20ns and 30ns. What is the efficiency of the pipeline when very large number of task are executed?

$k = 4$

tp : max (stage delay)
$t_p = max(10,15,20,30) = 30nsec$

tn : each instruction execution time
$t_n = 10 + 15 + 20 + 30 = 75$ nsec

$$s = \frac{t_n}{tp}$$

$$s = \frac{75}{30} = 2.5$$

$$n = \frac{s}{k} \quad \text{(neta : efficiency)}$$

$$n = \frac{2.5}{4} = 0.625 = 62.5\%$$

Q. 3 A 4 segment instruction pipeline has the respective stage delay of 8ns, 11ns 20ns, 2ns respectively. The interface register are used between the stages have a delay of 2ns. What is the approx. speed up factor when very large number of task are pipelined?

$k = 4$

tp : max (stage delay + buffer delay)
$t_p = max(8+2, 11+2, 20+2, 2+2) = 22nsec$

tn : each instruction execution time
$t_n = 8 + 11 + 20 + 2 = 41$ nsec

$$s = \frac{t_n}{tp}$$

$$s = \frac{41}{22} = 1.86$$

Q. 1 A 4-stage pipeline has the stage delays as 150, 120, 160 and 140 nanoseconds respectively. Registers that are used between the stages have a delay of 5 nanoseconds each. Assuming constant clocking rate, the total time taken to process 1000 data items on this pipeline will be **[GATE-2004: 2 Marks]**

nanoseconds each. Assuming constant clocking rate, the total time taken to process 1000 data items on this pipeline will be [GATE-2004: 2 Marks]

(A) 120.4 microseconds
(B) 160.5 microseconds
(C) 165.5 microseconds
(D) 590.0 microseconds

$k = 4$

tp : max (stage delay + buffer delay)
$t_p = max(150+5, 120+5, 160+5, 140+5) = 165nsec$

n : 1000

$ET_{pipeline} = [k+(n-1)]\ tp$
$ET_{pipeline} = [4+(1000-1)]165$
$ET_{pipeline} = 1003 \times 165 \times 10^{-9}$
$ET_{pipeline} = 165.5 \times 10^{-6}$

NAT Q. 5

A five-stage pipeline has stage delays of 150,120,150,160 and 140 nanoseconds. The registers that are used between the pipeline stages have a delay of 5 nanoseconds each.
The total time to execute 100 independent instructions on this pipeline, assuming there are no pipeline stalls, is _____ nanoseconds.
[GATE-2021(Set-1)-CS: 2M]

$k = 5$

tp : max (stage delay + buffer delay)
$t_p = max(150+5, 120+5, 160+5, 140+5) = 165nsec$

n : 100

$ET_{pipeline} = [k+(n-1)]\ tp$
$ET_{pipeline} = [5+(1000-1)]165$
$ET_{pipeline} = 1003 \times 165 \times 10^{-9}$
$ET_{pipeline} = 104 \times 165nsec$
$ET_{pipeline} = 17160\ nsec$

NAT Q. 6

Consider a 3-stage pipelined processor having a delay of 10 ns (nanoseconds), 20 ns, and 14 ns, for the first, second, and the third stages, respectively. Assume that there is no other delay and the processor does not suffer from any pipeline hazards. Also assume that one instruction is fetched every cycle.
The total execution time for executing 100 instructions on this processor is _____ns. [GATE-2023-CS: 1M]

$k = 3$

tp : max (stage delay)
$t_p = max(10,20,14) = 20nsec$

n : 100

$ET_{pipeline} = [k+(n-1)]\ tp$
$ET_{pipeline} = [3+(100-1)]20$
$ET_{pipeline} = 102 \times 20$
$ET_{pipeline} = 2040\ nsec$

concept - throughput :

in operating system : number of processes completed in per time unit

in computer network : throughput = efficiency x bandwith

in general : throughput means rate of output (kis rate se aapko output aa raha hai)

throughput : number of tasks are completed per unit of time

for n instruction :

$ET_{pipeline} = [k+(n-1)]$ tp time

means in $[k+(n-1)]$ tp time, n instruction (task) executed.

so, in 1 unit of time per unit $= \frac{n}{[k+(n-1)]}$

throughput : $\frac{n}{[k+(n-1)]}$

when numer of instructions are large (or) not given (or) in ideal case:

throughput : $\frac{1}{t_p}$

summary:

$$ET_{pipelining} = [k + (n-1)]\, tp$$

k : no of stages (segments)
n : no of instructions
tp : each stage delay in pipeline.

$$ET_{nonpipeline} = n \times tn$$

n : no of instruction
tn : each instruction execution time in non-pipeline

$$\text{perfomance gain (speed up factor) (s)} = \frac{\text{perfomance of pipe}}{\text{perfomance of non-pipe}}$$

$$\text{perfomance gain (speed up factor) (s)} = \frac{ET_{non\text{-}pipeline}}{ET_{pipeline}}$$

$$\text{perfomance gain (speed up factor) (s)} = \frac{n \times t_n}{[k + (n-1)]\, t_p}$$

k : no of stages (segments)
n : no of instructions
tp : each stage delay in pipeline.

when numer of instructions are large (or) not given (or) in ideal case:

$$\text{perfomance gain (speed up factor) (s)} = \frac{t_n}{t_p}$$

tn : each instruction execution time in non-pipeline
tp : each stage delay in pipeline.

$$\text{throughput} : \frac{n}{[k + (n-1)]}$$

when numer of instructions are large (or) not given (or) in ideal case:

$$\text{throughput} : \frac{1}{t_p}$$

$$n = \frac{s}{k}$$ n (neta)

n : efficiency
s : speed up factor
k : number of stages

in non-uniform

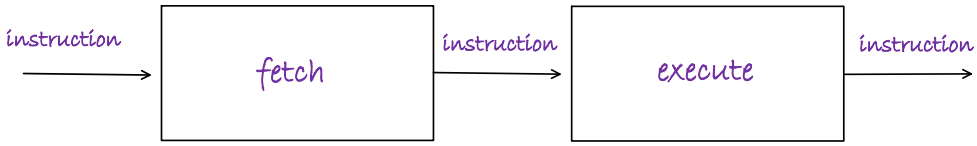
t_p : max (stage delay + buffer delay)

when perfectly balanced / ideal case:

$$t_n = k * tp$$

$$t_p = \frac{t_n}{k}$$

$$s = k$$



two stage instruction pipeline

Q. 1 Consider on instruction pipeline which has speed up factor 20 while operate with 80% efficiency. What could be the number of stages in the pipeline?

$s = 20$

$n = \frac{s}{k}$ (neta : efficiency)

$80\% = \frac{20}{k}$

$k = \frac{20}{80} \times 100 = 25$

Q. 2 4 segment pipeline have the respective stage delay of 10ns, 15ns, 20ns and 30ns. What is the efficiency of the pipeline when very large number of task are executed?

$k = 4$

tp : max (stage delay)
$t_p = max(10, 15, 20, 30) = 30nsec$

tn : each instruction execution time
$t_n = 10 + 15 + 20 + 30 = 75\ nsec$

$s = \frac{t_n}{tp}$

$s = \frac{75}{30} = 2.5$

$n = \frac{s}{k}$ (neta : efficiency)

$n = \frac{2.5}{4} = 0.625 = 62.5\%$

Q. 3 A 4 segment instruction pipeline has the respective stage delay of 8ns, 11ns 20ns, 2ns respectively. The interface register are used between the stages have a delay of 2ns. What is the approx. speed up factor when very large number of task are pipelined?

$k = 4$

tp : max (stage delay + buffer delay)
$t_p = max(8+2, 11+2, 20+2, 2+2) = 22nsec$

tn : each instruction execution time
$t_n = 8 + 11 + 20 + 2 = 41\ nsec$

t_p = max(8 + 2, 11 + 2, 20 + 2, 2 + 2) = 22nsec

tn : each instruction execution time

$t_n = 8 + 11 + 20 + 2 = 41$ nsec

$$s = \frac{t_n}{tp}$$

$$s = \frac{41}{22} = 1.86$$

Q. 4 Consider 4 stage pipeline with the respective stage delay of 20ns, 80ns, 50ns and 10ns. In the enhancement process of the pipeline largest stage is split into 2 equal stage delay.

(i) What is speed up between new and old design?

(ii) What is the clock frequency of new pipeline?

k = 4

tp : max (stage delay + buffer delay)

t_p = max(20, 80, 50, 10) = 80nsec

tn : each instruction execution time

$t_n = 20 + 80 + 50 + 10 = 160$ nsec

in new design :

largest stage is split into 2 equal stage delay

tp = max(20, 40, 40, 50, 10) = 50nsec

(i) What is speed up between new and old design?

$$s = \frac{\text{perfomance of new}}{\text{perfomance of old}}$$

$$s = \frac{ET_{old}}{ET_{new}}$$

$$s = \frac{80}{50} = 1.6$$

(ii) What is the clock frequency of new pipeline?

t_p = 50nsec

$$\text{frequency} = \frac{1}{\text{time}}$$

$$\text{frequency} = \frac{1}{50 \times 10^{-9}}$$

$$\text{frequency} = \frac{1}{50} \times 10^{9}$$

$$\text{frequency} = \frac{1000}{50} \times 10^{6} = 20\text{MHZ}$$

Q. 5 The stage delays in 5 stage pipeline are 900, 600, 550, 450 and 400 nanoseconds. The largest stage delay is replaced with a functionally equivalent design involving two stages with respective stage delay 440 and 460 nanoseconds. What is the throughput increase of the pipeline?

(a) 25% (b) 33.3%

(c) 50% (d) Does not change

$k = 5$

old design

tp : max (stage delay)

$t_p = max(900, 600, 550, 450, 400) = 900nsec$

1 instruction takes : 900×10^{-9} sec

in one second : $1/900 \times 10^9$ instruction

throughput : 1/900 instruction

new design

tp : max (stage delay)

$t_p = max(440, 460, 600, 550, 450, 400) = 600nsec$

throughput : 1/600 instruction

$$\% \text{ of increment in throughput} : \frac{\text{new - old}}{\text{old}}$$

$$\% \text{ of increment in throughput} : \frac{1/600 - 1/900}{1/900}$$

$$\% \text{ of increment in throughput} : \frac{1/6 - 1/9}{1/9}$$

$$\% \text{ of increment in throughput} : \frac{9-6}{54} \times \frac{9}{1} = 50\%$$

Q. 2 We have two designs D1 and D2 for a synchronous pipeline processor. D1 has 5 pipeline stages with execution times of 3 nses, 2 usec, 4 nsec, 2 nsec and 3 nsec while the design D2 has 8 pipeline stages each with 2 nsec execution time.
How much time can be saved using design D2 over design D1 for executing 100 instructions? **[GATE-2005: 2 Marks]**

A 214 nseconds

B 202 nseconds

C 86 nseconds

D -200 nsecond

design D1

$k = 5$

tp : max (stage delay)

$t_p = max(3, 2, 4, 2, 3) = 4nsec$

$ET_{pipeline} = [k + (n-1)]\ tp$

design D2

$k = 8$

tp : 2nsec

$ET_{pipeline} = [k + (n-1)]\ tp$

$ET_{pipeline} = [8 + (100-1)]2$

tp : max (stage delay)

t_p = max(3, 2,4,2,3) = 4nsec

$ET_{pipeline}$ = [k+(n-1)] tp

$ET_{pipeline}$ = [5+(100-1)]4

$ET_{pipeline}$ = 104 x 4

$ET_{pipeline}$ = 416 nsec

tp : 2nsec

$ET_{pipeline}$ = [k+(n-1)] tp

$ET_{pipeline}$ = [8+(100-1)]2

$ET_{pipeline}$ = 107 x 2

$ET_{pipeline}$ = 214 nsec

time saved = 416 - 214 = 202nsec

Q. 3 A nonpipelined single cycle processor operating at 100 MHz is converted into a synchronous pipelined processor with five stages requiring 2.5 nsec, 1.5 nsec, 2 nsec, 1.5 nsec and 2.5 nsec, respectively. The delay of the latches is 0.5 nsec. The speedup of the pipeline processor for a large number of instructions is [GATE-2008: 2 Marks]

(A) 4.5

(B) 4.0

(C) 3.33

(D) 3.0

- nonpipelined processor
- operating at 100MHz

so cycle time : $\frac{1}{100 \times 10^6} = \frac{1}{10^8} = \frac{10^{-8} \times 10}{10}$

$= 10 \times 10^{-9}$ sec

cycle time : 10nsec

k = 5

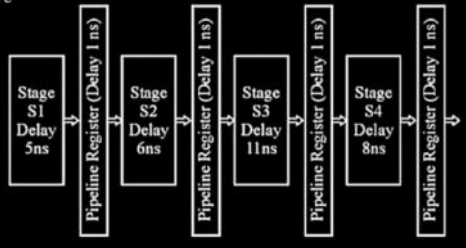
tp : max (stage delay + buffer delay)

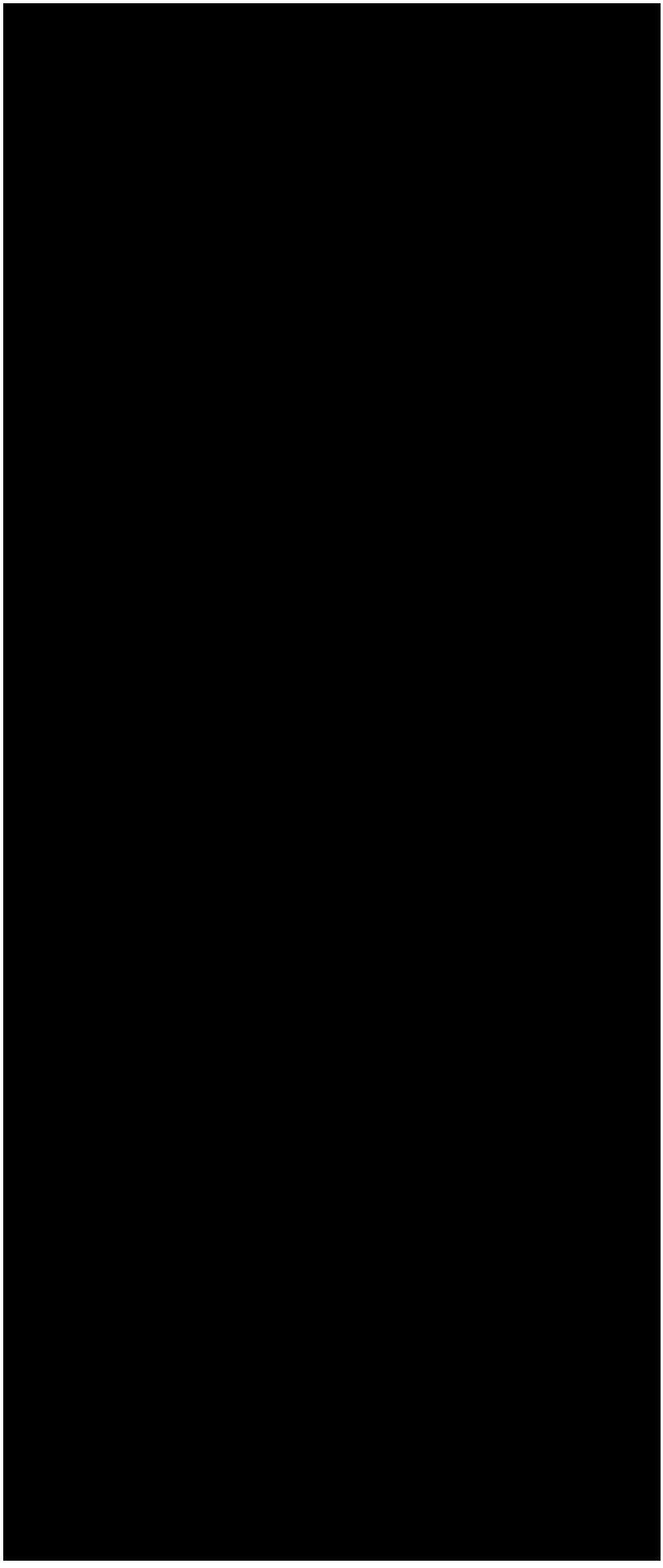
t_p = max(2+0.5, 1.5+0.5, 2+0.5, 1.5+0.5, 2.5+0.5) = 3nsec

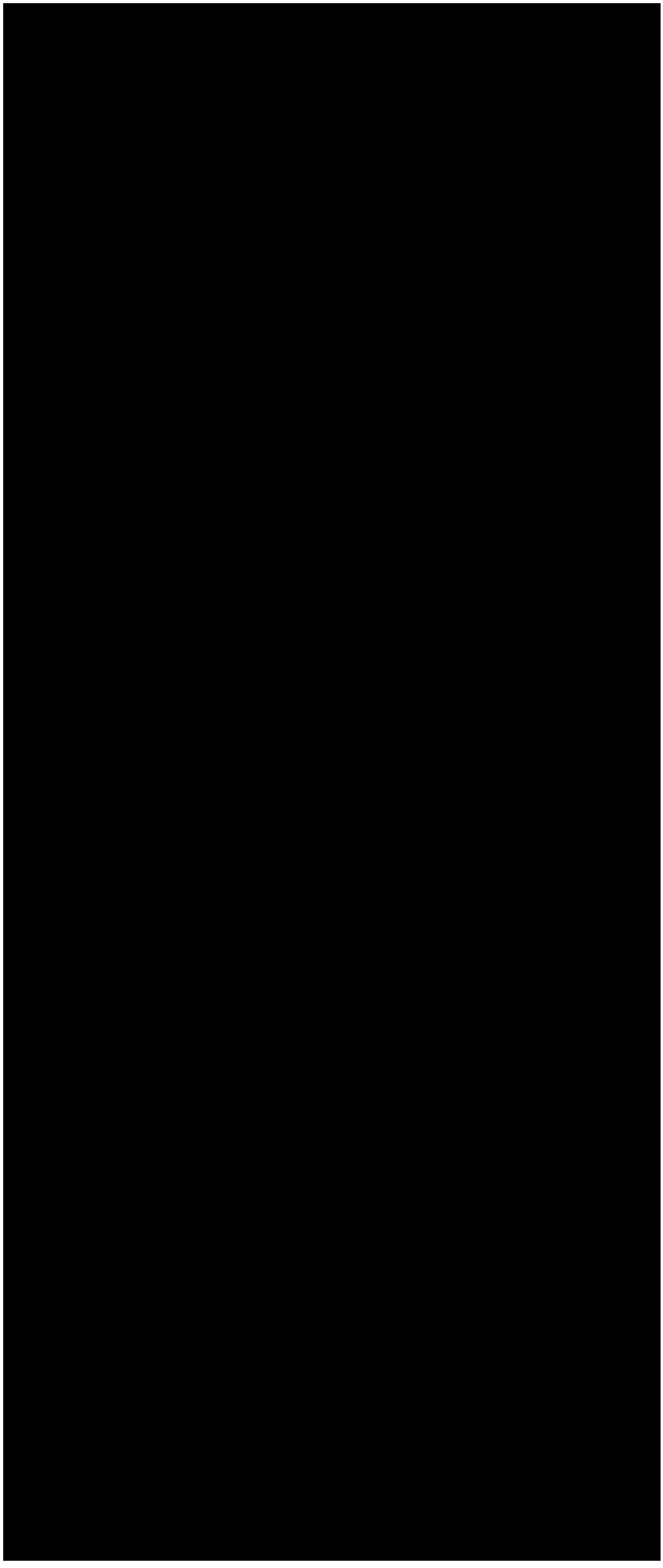
$s = \frac{t_n}{tp}$

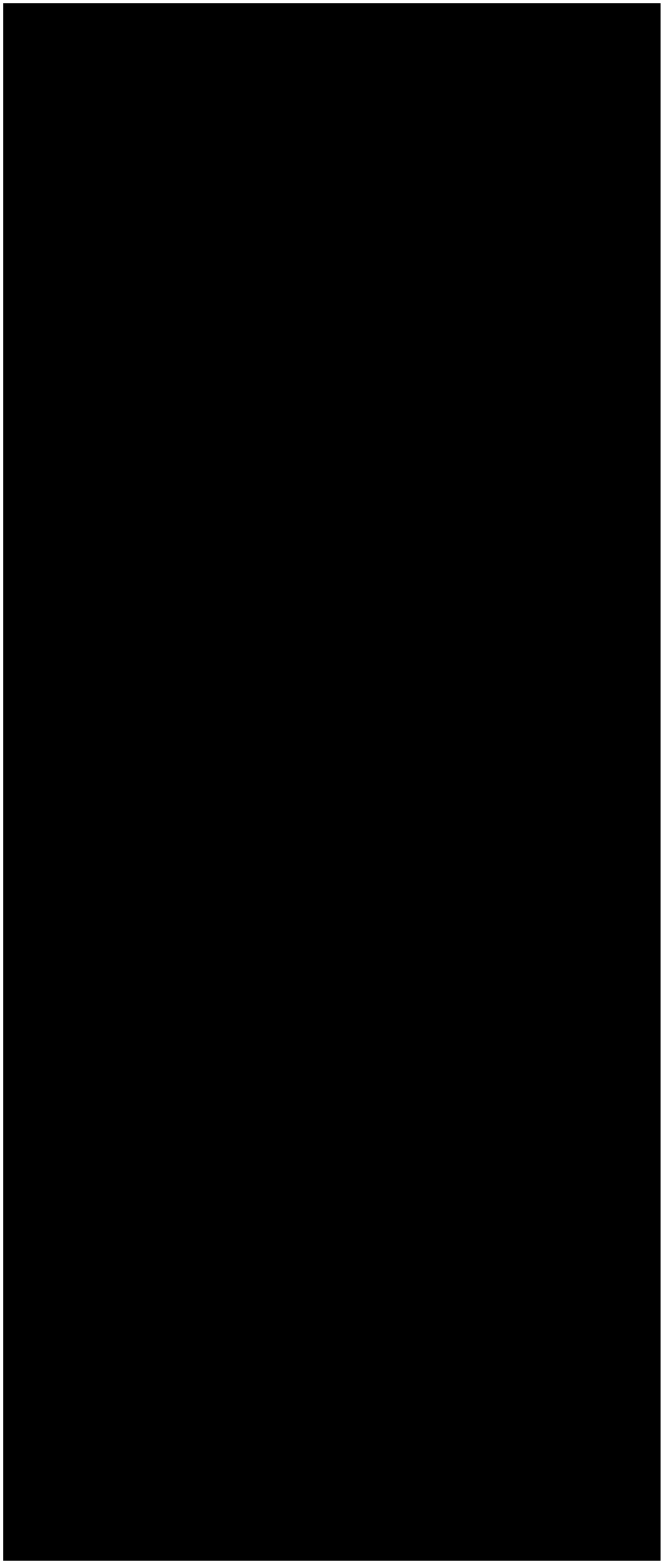
$s = \frac{10}{3} = 3.33$

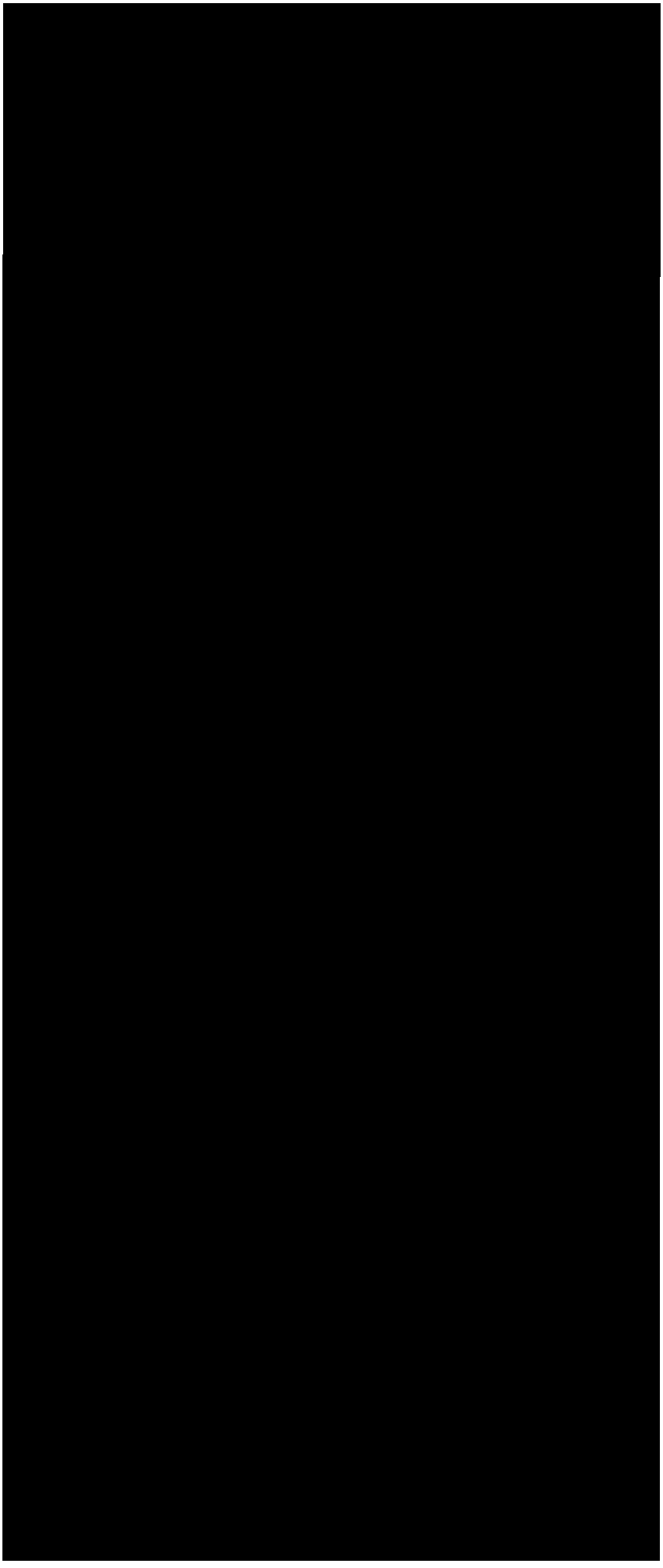
Q. 4 Consider an instruction pipeline with four stages (S1, S2, S3 and S4) each with combinational circuit only. The pipeline registers are required between each stage and at the end of the last stage. Delays for the stages and for the pipeline registers are as given in the figure

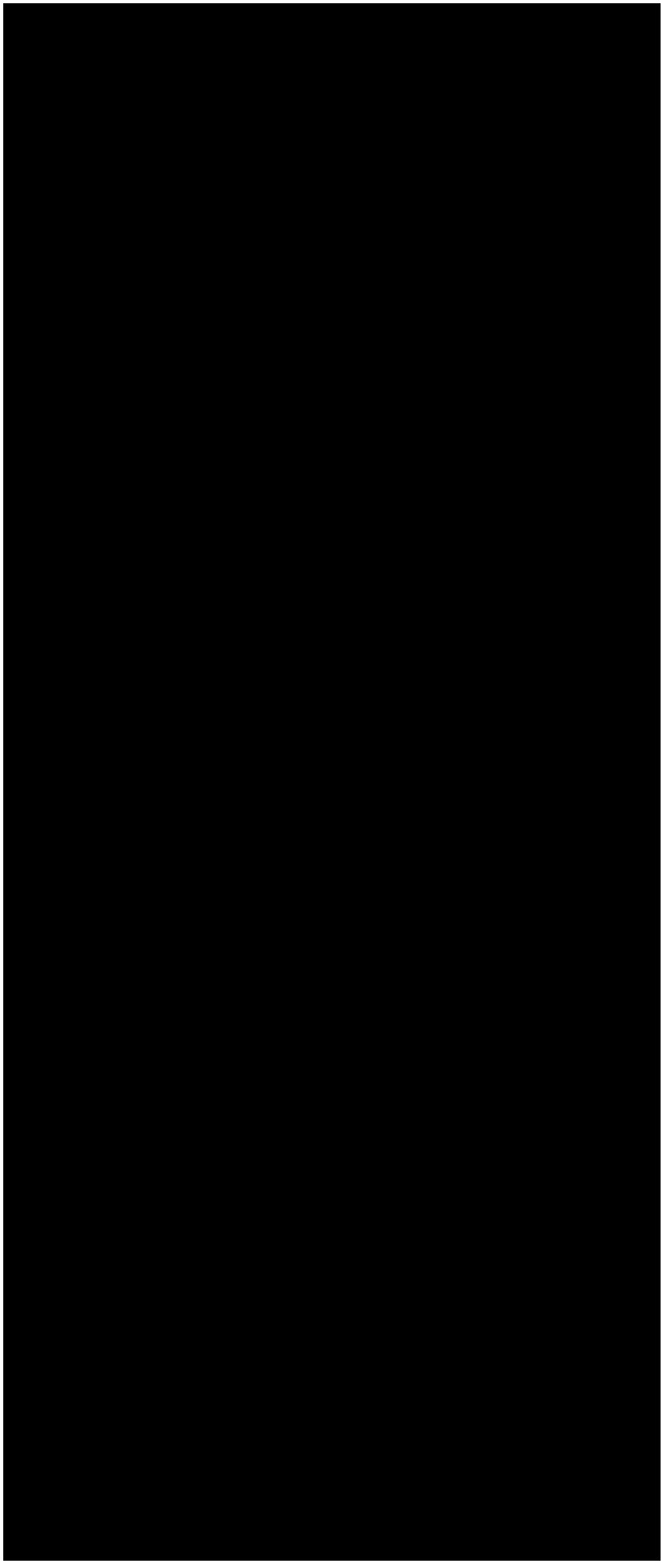


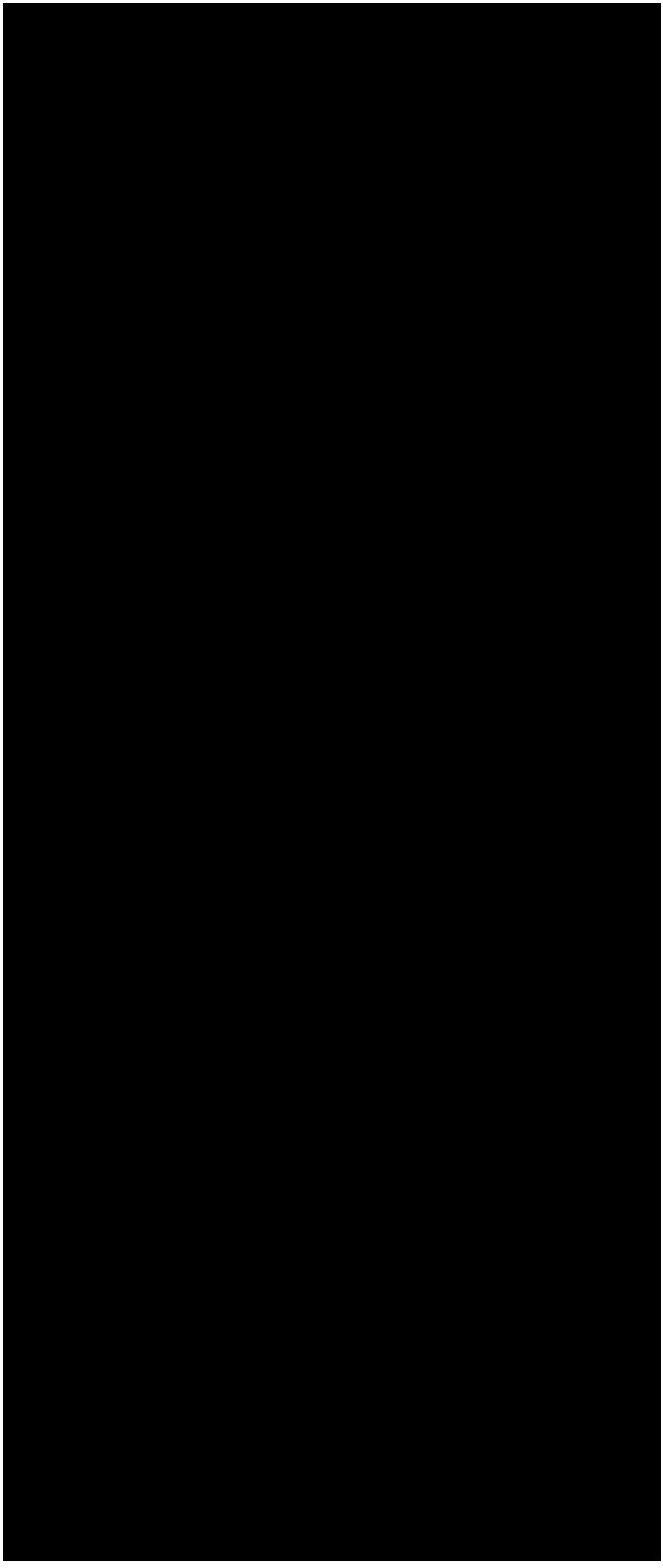












q.(i) why cycle per instruction (CPI) = 1 in pipeline.

q.(i) what is the meaning of cycle per instruction (CPI) = 1?

q.(ii) how design (construct) the pipeline?

q.(iii) why clock pulse is required?

q.(iv) how to set this CPI in uniform delay? and how to set this CPI in non-uniform delay?

q.(i) why cycle per instruction (CPI) = 1 in pipeline.

in the pipeline average CPI = 1

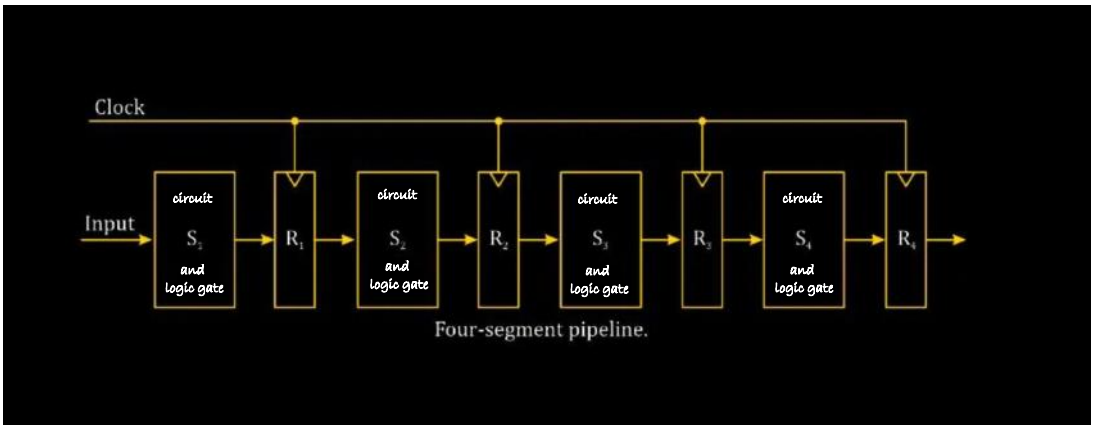
$ET_{pipeline}$ for n instruction = [K+(n-1)] cycle

1 average instruction = $\frac{[K+(n-1)]}{n}$ cycle

(K-1) negligible (1000 instruction ke comparison mai bahut kam k(3-4-5) ka value)

Cycle Per Instruction = 1

q.(ii) how design (construct) the pipeline?

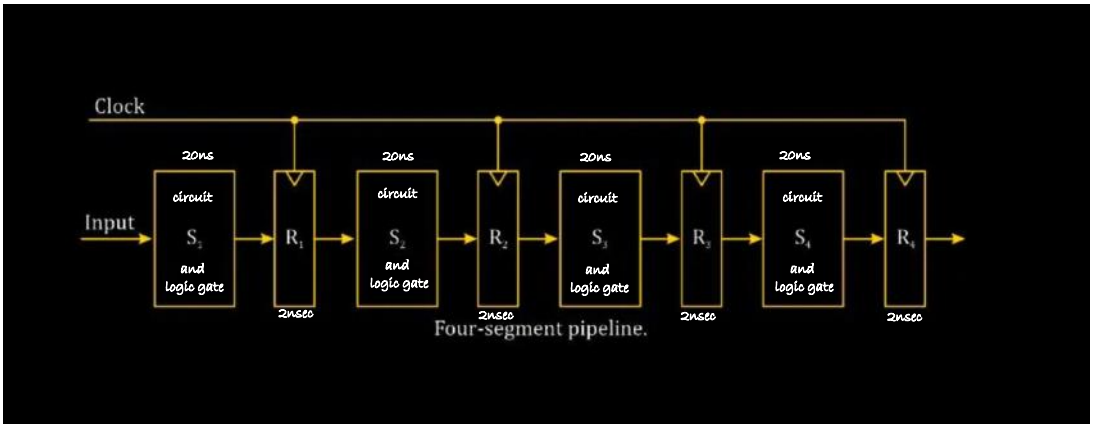


RISC : 5stages

if we want to construct N stage pipeline then entire CPU is divided into 'N' functional unit (independent functional unit) which is independent from each other.

independent functional unit : it means when one functional unit perform the task, in the same time other functional unit perform the other task (operation).
(functional unit : adder, subtractor,logic GATE, hardware etc)

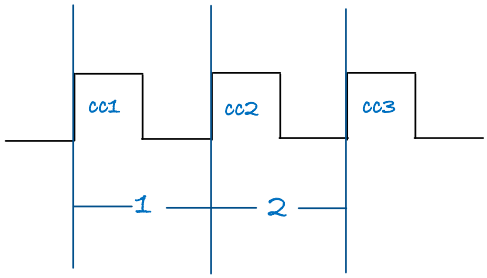
agar hum computer mai N stage ki pipeline banana chate hai toh hume entire CPU ko independent N functional unit mai construct krna padega



ex. instruction pipeline :
i1 is in decoding stage and i2 is in fetch stage in the same clock cycle number 2.

WB				
EX				
ID		i_1		
IF	i_1	i_2		
	1	2	3	4

note : characterstics of the pipeline is, in every new cycle new input must be inserted into the pipeline.



clock cycle / pulse / signal

q. (i) why clock pulse is required?

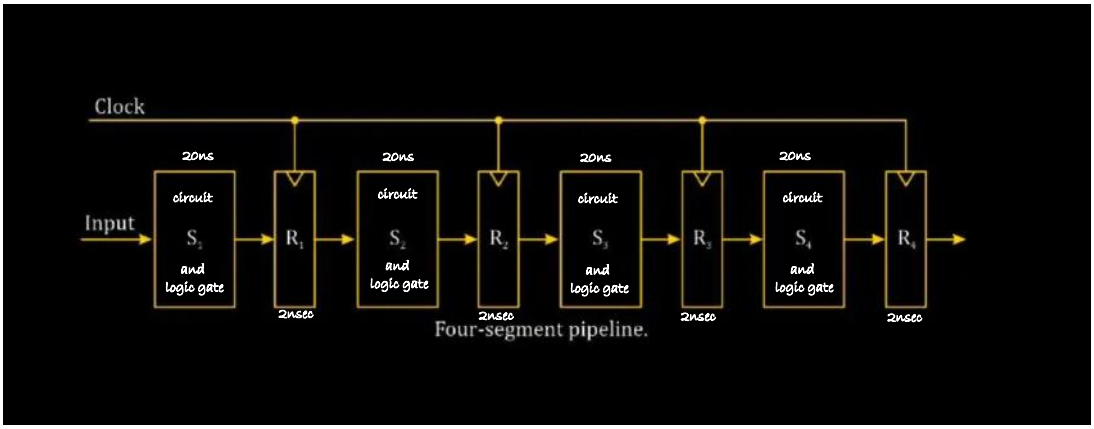
- because when we will enable the clock then the operation will perfom (data will move from one register to another register (or) any task)

- to provide the synchronization between the stages.

jab clock pulse enable karoge tabhi operation perform hoga.

q. (iv) how to set this CPI in uniform delay and in non-uniform delay?

(i) uniform delay :

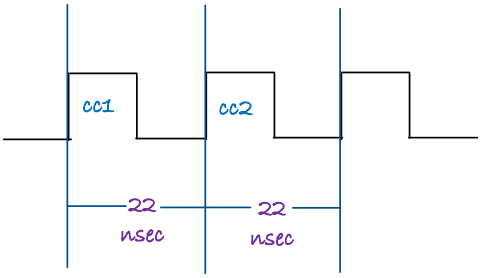


time : 22 nsec

frequency : $\frac{1}{\text{time}}$ hz $= \frac{1}{22\text{nsec}}$ hz

cycle time : 22nsec

every 22 nsec new input insert in the pipeline



clock cycle / pulse / signal

in uniform delay :

WB				i_1	i_2	i_3
EX			i_1	i_2	i_3	
ID		i_1	i_2	i_3		
IF	i_1	i_2	i_3			
	1	2	3	4	5	6

at
clock cycle (CC) 4 : i_1 out
clock cycle (CC) 5 : i_2 out
clock cycle (CC) 6 : i_3 out

first instruction takes : 22nsec = 4x 22
remaining instruction takes : (n-1)22

because after every 1 cycle we are getting output

$ET_{pipeline} = [k + (n-1)]$ cycle

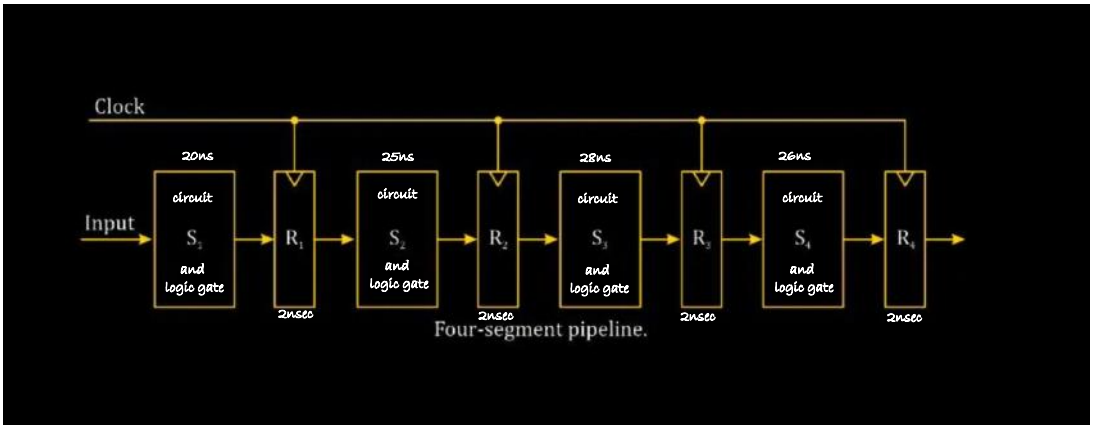
actually : k x cycle + (n-1) cycle

for the very i_{st} instruction — for remaining (n-1) instruction

4x 22 + (n-1)22

pehle instruction ke lie 4 cycle laga uske baad remaining (n-1) instructions ke lie har cycle par we are getting one output

(ii) non-uniform delay :



t_p : max (stage delay + buffer delay)

t_p : 30nsec CPI : 30nsec

q. what if we take minimum means (20+2) = 22 nsec?

time : 22 nsec

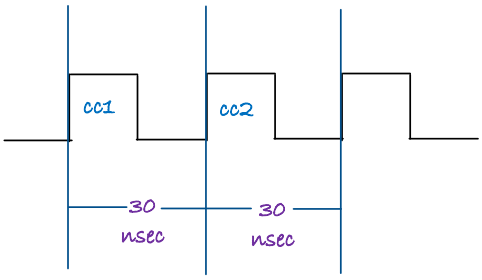
frequency : $\frac{1}{\text{time}}$ hz = $\frac{1}{22\text{nsec}}$ hz

cycle time : 22nsec

Stage 1 will complete its work in 22 nanoseconds, and data movement from stage 1 to stage 2 will occur within this period. However, for the remaining stages, tasks will not be completed within this same 22 nanosecond clock cycle. Therefore, the maximum stage delay is considered for the proper functioning of the pipeline.

maximum :

t_p : 30nsec



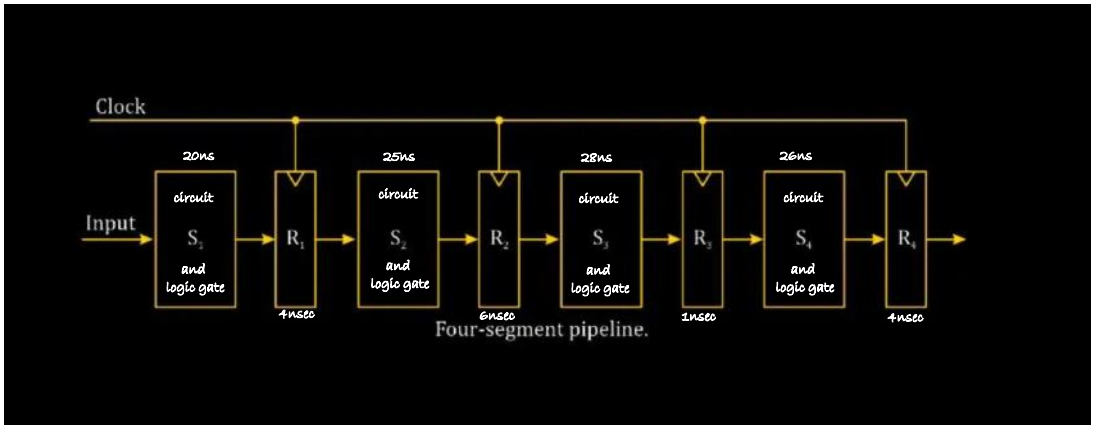
clock cycle / pulse / signal

q. if we take maximum : 30nsec then what does synchronization means?

In stage 1, the task will finish in 22 nanoseconds, but the clock cycle is set to 30 nanoseconds. This means that every 30 nanoseconds, the output from each stage will become available. Therefore, after one complete cycle of 30 nanoseconds, the output from stage 1 will be passed on to stage 2, and this sequence will continue for each subsequent stage.

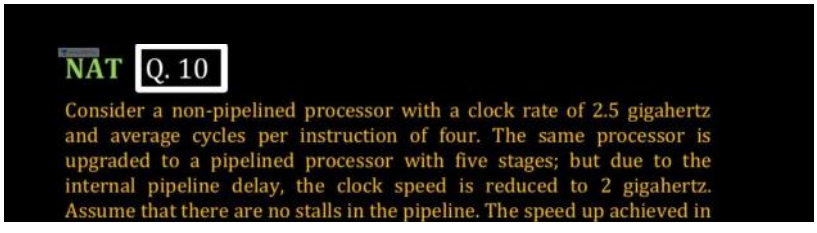
so synchronization!

q. what if register delay is also different?



t_p : max (stage delay + buffer delay)

t_p : 31nsec CPI : 31nsec



and average cycles per instruction of four. The same processor is upgraded to a pipelined processor with five stages; but due to the internal pipeline delay, the clock speed is reduced to 2 gigahertz. Assume that there are no stalls in the pipeline. The speed up achieved in this pipelined processor is_____.

[GATE-2015(Set-1)-CS: 2M]

nonpipelined processor

- operating at 2.5 GHz

so cycle time : $\frac{1}{2.5 \times 10^9} = \frac{10^9 \times 1}{2.5}$

$= 0.4$ nsec

cycle time : 0.4 nsec

$k = 4$

$ET_{non\text{-}pipeline} = n \times tn$

$ET_{non\text{-}pipeline} = 4 \times 0.4 = 1.6$ nsec

$s = \frac{t_n}{tp}$

$s = \frac{10}{3} = 3.33$

pipelined processor

- operating at 2 GHz

so cycle time : $\frac{1}{2 \times 10^9} = \frac{10^9 \times 1}{2}$

$= 0.5$ nsec

cycle time : 0.5 nsec

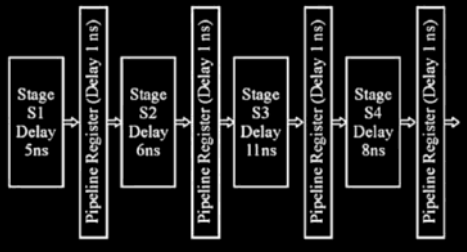
$k = 5$

$ET_{pipeline} = 0.5$ nsec

$s = \frac{t_n}{tp}$

$s = \frac{1.6}{0.5} = 3.2$nsec

q. relation between CPI and t_p



t_p : max (stage delay + buffer delay)

t_p : max (5+1, 6+1, 11+1, 8+1)

t_p : 12nsec

and CPI = 1 cycle

so we will set CPI = 12nsec

jo t_p aaya vo apan cycle time set kardete hai.

NAT Q. 14

Consider two processors P_1 and P_2 executing the same instructions set. Assume that under identical conditions, for the same input, a program running on P_2 takes 25% less time but incurs 20% more CPI (clock cycles per instruction) as compared to the program running on P_1. If the clock frequency of P_1 is $1GH_Z$, then the clock frequency of P_2 (in GH_Z) is _____.

[GATE-2014(Set-1)-CS: 2M]

$ET_{pipeline}$ = no of instruction x cycle time x CPI

same number of instruction in both processor P_1 and P_2

ETp_1 = cycle time x CPI

20%more CPI in P_2

ETp_2 = cycle time x 1.2 CPI

and 25%less time

$ETp_2 = 0.75 \times ETp_1$

clock frequency : 1GHZ

$cycle\ time_1$: 1nsec

$0.75 \times ETp_1 = 1.2\ CPI \times cycle\ time_2$

$0.75 \times CPI \times 1nsec = 1.2CPI \times cycle\ time_2$

$0.75\ CPI = 1.2\ CPI \times cycle\ time_2$

$cycle\ time_2 = \frac{0.75}{1.2} = 0.625\ nsec$

$$frequency = \frac{1}{time_2}$$

$$frequency = \frac{1}{0.625 \times 10^{-9}}$$

frequency = 1.6 GHZ

MCQ Q. 15

Consider the following processor design characteristics:

I. Register-to-register arithmetic operations only.
II. Fixed-length instruction format.
III. Hardwired control unit.

Which of the characteristics above are used in the design of a RISC processor?

[GATE-2018-CS: 1M]

(A) I and II only
(B) II and III only
(C) I and III only
(D) I, II and III

MCQ Q. 11

Consider a 4-stage pipeline processor. We want to execute a loop: For(i=1;i<=1000;i++){ I1, I2, I3, I4} where the time taken (in ns) by instruction I1 to I4 for stages S1, S2, S3, S4 is shown below:

	S1	S2	S3	S4
I1	1	2	1	2
I2	2	1	2	1
I3	1	1	2	1
I4	2	1	2	1

The Output of I1 for i=2 will be available after ?

[GATE-2004-CS: 2M]

A) 11ns

B) 12ns

C) 13ns

D) 28ns

		S_1	S_2	S_3	S_4
for i = 1	i_1	1	3	4	6
	i_2	3	4	6	7
	i_3	4	5	8	9
	i_4	6	7	10	11
for i = 2	i_1	7	9	11	13
	i_2	9	10	13	14
	i_3	10	11	15	16
	i_4	12	13	17	18

MCQ Q. 12

Consider a 4-stage pipeline processor. The number of cycles needed by the four instructions I1, I2, I3, I4 in stages S1, S2, S3, S4 is shown below:

	S1	S2	S3	S4
I1	2	1	1	1
I2	1	3	2	2
I3	2	1	1	3
I4	1	2	2	2

What is the number of cycles needed to execute the following loop?
for (i = 1 to 2) {I1; I2; I3; I4;}

[GATE-2009-CS: 2M]

A) 16

B) 23

C) 28

D) 30

		S_1	S_2	S_3	S_4
for i = 1	i_1	2	3	4	5
	i_2	3	6	8	10
	i_3	5	7	9	13
	i_4	6	9	11	15
for i = 2	i_1	8	10	12	16
	i_2	9	13	15	18
	i_3	11	14	16	21
	i_4	14	16	18	23

each instruction taking different amount of delay in different stages

$ET_{pipeline}$ = 11 (i=1) one round
$ET_{non\text{-}pipeline}$ = 23

speed up factor = $\frac{ET_{non\text{-}pipeline}}{ET_{pipeline}} = \frac{23}{11} = 2.09$

[weightage : 3-4marks]

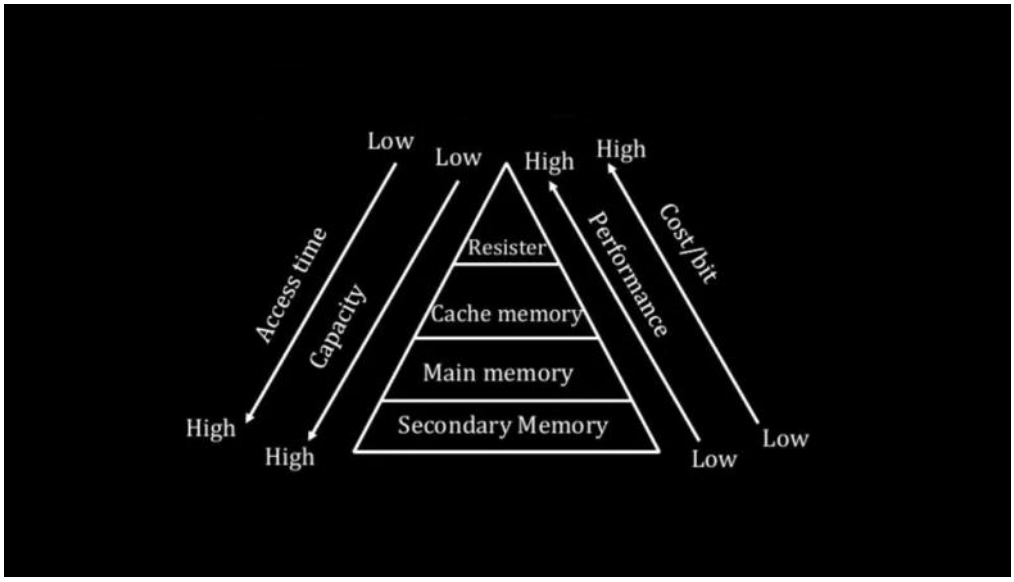
cache memory

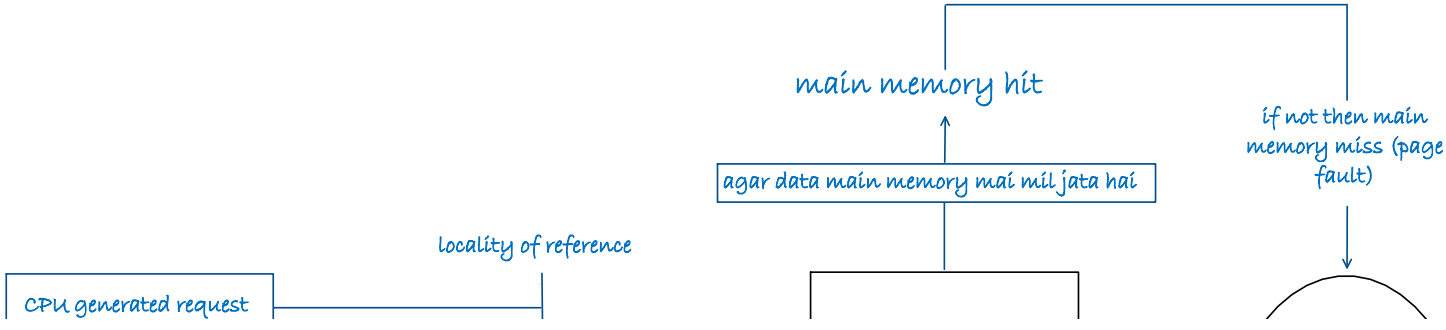
register	cache memory	main memory	secondary memory
fastest	fast	slow	slowest
flip-flop	SRAM/flip-flop	DRAM/capacitor	

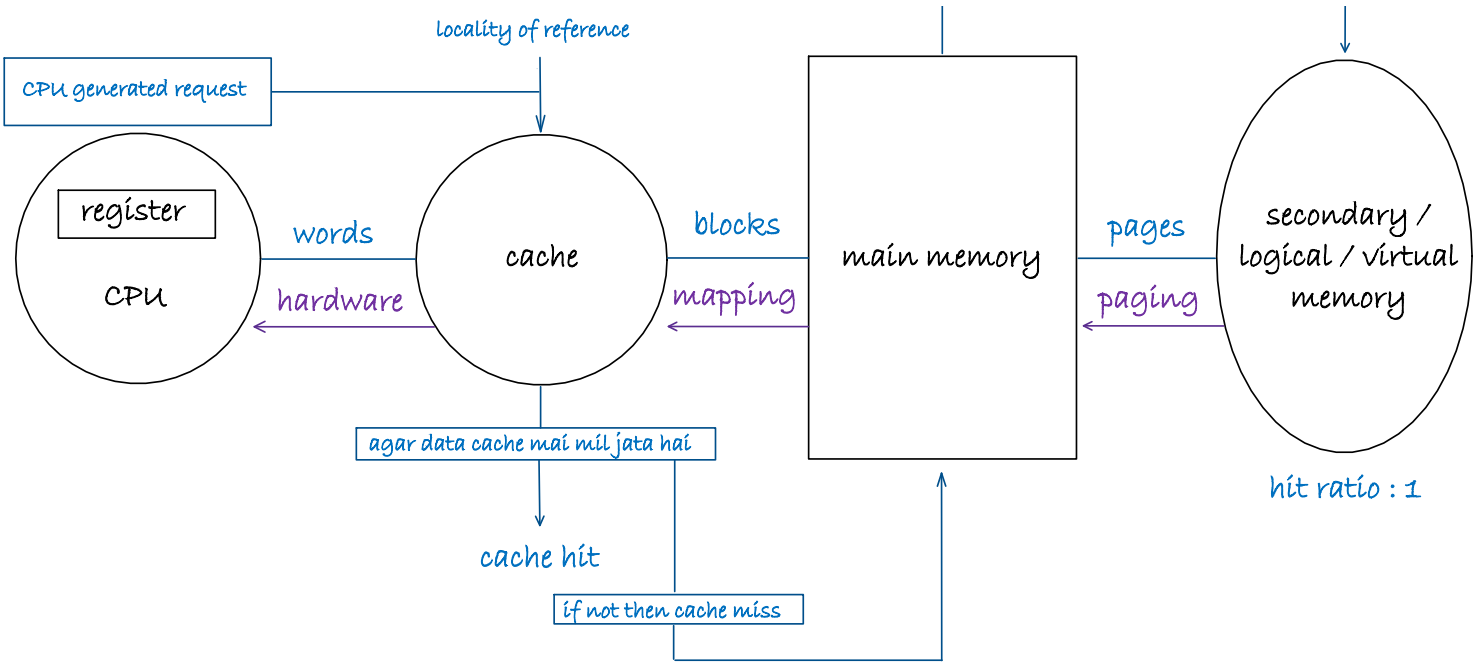
q. why cache memory?
- fastest compared to memory

register is also fastest but register size is small.

memory hierarchy : hierarchy design organize the system supported memory into 4 levels to minimize the accessing times.







q. what is hit ratio?

$$\text{hit ratio} = \frac{\text{number of hit}}{\text{total number of access}}$$

if cache hit ratio is 80% that means 80% reference found in cache.

topic : memory

(i) CPU generated request initially refer to the cache

(ii) if the reference (respective data) find in the cache then that is called cache hit (operation is called hit) then respective data is given cache to CPU in the form of words.

(iii) if the reference is not found in the cache then its called cache miss, then it is forwarded to main memory.

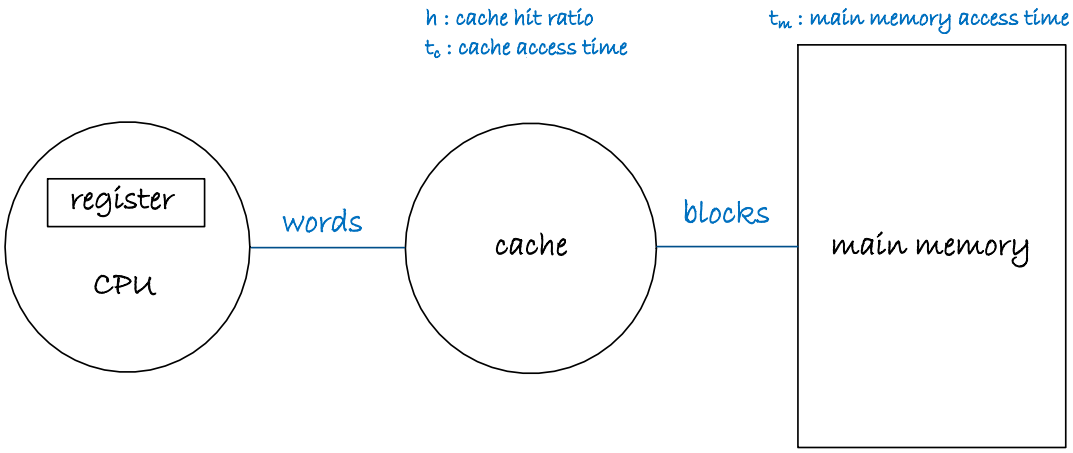
(iv) if the reference is found in the main memory then it is called main memory hit (page hit) then respective data is given main memory to cache in the form of blocks and cache to CPU in the form of words.

(v) if the reference is not found in the main memory then it is called main memory miss (page fault) then the reference forward to secondary memory.

(vi) secondary memory is the last level of memory in which hit ratio is always '1' so respective data is transferred from secondary memory to main memory in the form of pages, main memory to cache memory in the form of blocks, then cache to CPU in the

form of words.

mapping : the process of transferring the data from main memory to cache memory is called mapping.



access time : time needed by processor to read/write memory in ROM/RAM

average memory access time : [T_{avg}]

T_{avg} = hit x time taken by memory when there is a hit + (1-H) time taken by memory when there is a miss.

hit + miss = 1

$$\text{hit ratio} = \frac{\text{number of hit}}{\text{total number of access}}$$

Q.1 Calculate the average Access time , when the CPU request for the memory 100 times, out of 100 times, 90 times hit & 10 Time miss. If time taken when there is a hit(Each hit) is 20ns & time taken when there is a Miss(Each Miss) is 150ns. ?

total CPU request : 100
hit : 90 times
time taken : 20ns
hit miss : 10 times
time taken : 150ns

hit ratio : $\frac{\text{number of hit}}{\text{total number of access}} = \frac{90}{100} = 0.9$

miss ratio : (1-H) = 1-0.9 = 0.1

T_{avg} = hit x time taken by memory when there is a hit + (1-H) time taken by memory when there is a miss.
$T_{avg} = 0.9 \times 20 + 0.1 \times 150$
$T_{avg} = 18 + 15$
$T_{avg} = 33ns$

$T_{avg} = 0.9 \times 20 + 0.1 \times 150$
$T_{avg} = 18 + 15$
$T_{avg} = 33ns$

(or)

total time : $90 \times 20 + 10 \times 150$
total time : $1800 + 1500$
total time : $3300nsec$

$$T_{avg} = \frac{3300}{100} = 33nsec$$

Q.2 Calculate the average Access time , when the CPU request for the memory 400 times, out of 400 times, 300 times hit & 100 Time miss. If time taken when there is a hit(Each hit) is 20ns & time taken when there is a Miss(Each Miss) is 150ns. ?

total CPU request : 400
hit : 300 times
time taken : 20ns
hit miss : 100 times
time taken : 150ns

$$\text{hit ratio : } \frac{\text{number of hit}}{\text{total number of access}} = \frac{300}{400} = 0.75$$

miss ratio : (1-H) = $1-0.75 = 0.25$

T_{avg} = hit x time taken by memory when there is a hit + (1-H) time taken by memory when there is a miss.
$T_{avg} = 0.75 \times 20 + 0.25 \times 150$
$T_{avg} = 15 + 37.5$
$T_{avg} = 52.5ns$

(or)

total time : $300 \times 20 + 100 \times 150$
total time : $6000 + 15000$
total time : $21000nsec$

$$T_{avg} = \frac{21000}{400} = 52.5nsec$$

Q. Assume that for a certain processor, a read request takes 50 nanoseconds on a cache miss and 5 nanoseconds on a cache hit. Suppose while running a program, it was observed that 80% of the processor's read requests result in a cache hit. The average read access time in nanoseconds is _____. [GATE – 2015]

hit : 80%
time taken : 5nsec
hit miss : 20%
time taken : 50nsec

$$\text{hit ratio : } \frac{\text{number of hit}}{\text{total number of access}} = \frac{80\%}{100\%} = .8$$

miss ratio : (1-H) = $1-0.8 = 0.2$

T_{avg} = hit x time taken by memory when there is a hit + (1-H) time taken by memory when there is a miss.
$T_{avg} = .8 \times 5 + .2 \times 50$
$T_{avg} = 4 + 10$

$T_{avg} = 14ns$

types of memory organisation :

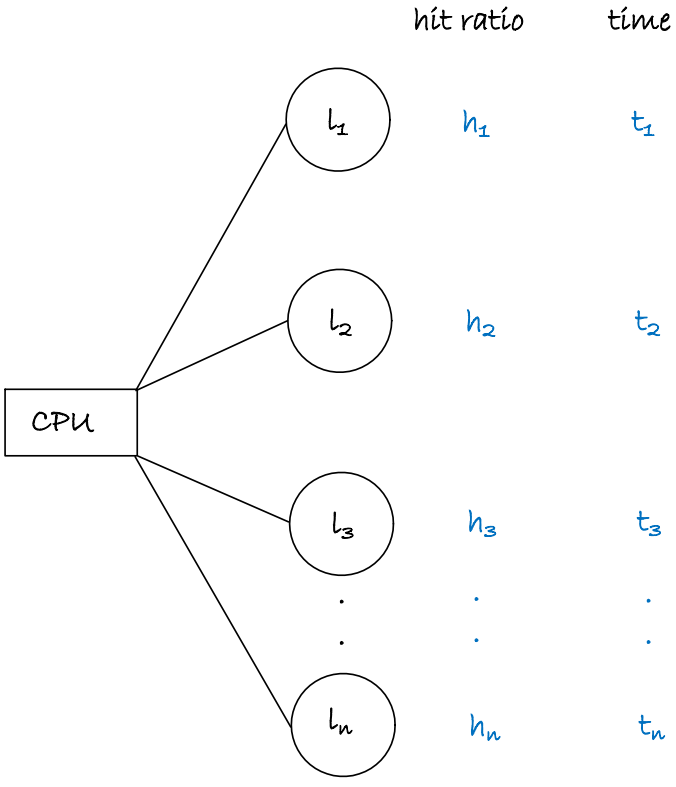
(i) simultaneous access memory organisation

(ii) hierarchical access memory organisation

(i) simultaneous access memory organisation : all the levels of memory directly connected to CPU (or) CPU is communication with all the level of memory directly but access (follow) in sequence

- when there is a miss in level 1 (l_1) then hit in level 2 (l_2) then directly data is given from level 2 (l_2) memory to CPU without copying into level 1 (l_1) memory

- when there is a miss in level 2 (l_2) then hit in level 3 (l_3) then directly data is given from level 3 (l_3) memory to CPU without copying into level 2 (l_2) and level 1 (l_1) memory



time : access time of the respective level memory

1 word access time : T_{avg}

number of words / sec (data transfer rate) (bandwidth) $= \frac{1}{T_{avg}}$

time required to access (read/write) 1 word from memory is called T_{avg}

$$T_{avg} = H_1t_1 + (1-H)H_2t_2 + (1-H_1)(1-H_2)H_3t_3 + \ldots\ldots (1-H_1)(1-H_2)\ldots\ldots(1-H_{n-1})H_nt_n$$

$T_{avg} = H_1t_1 + (1-H)H_2t_2 + (1-H_1)(1-H_2)H_3t_3 + \ldots\ldots (1-H_1)(1-H_2)\ldots\ldots(1-H_{n-1})H_nt_n$

pehle level mai mila

pehle level mai nahi mila

dusre level mai mila

teesre level mai mila

pehle aur dusre level mai nahi mila

pehle, dusre aur.. level mai nahi mila

h_n : always 1 (last level hit ratio is always 1)

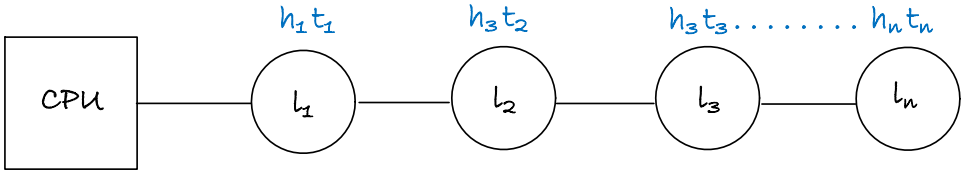
(ii) hierarchical access memory organisation : all in the hierarchical access CPU is communicating with only level 1 (l_1) memory

- when there is miss in level 1 (l_1) and hit in level 2 (l_2), firstly that data is copied from level 2 (l_2) to level 1 (l_1) then level 1 (l_1) to CPU

- when there is miss in level 1 (l_1) and level 2 (l_2) but hit in level 3 (l_3), firstly that data is copied from level 3 (l_3) to level 2 (l_2) then level 2 (l_2) to level 1 (l_1) then level 1 (l_1) to CPU

in the locality of reference is present (cache works on locality of reference).

n level

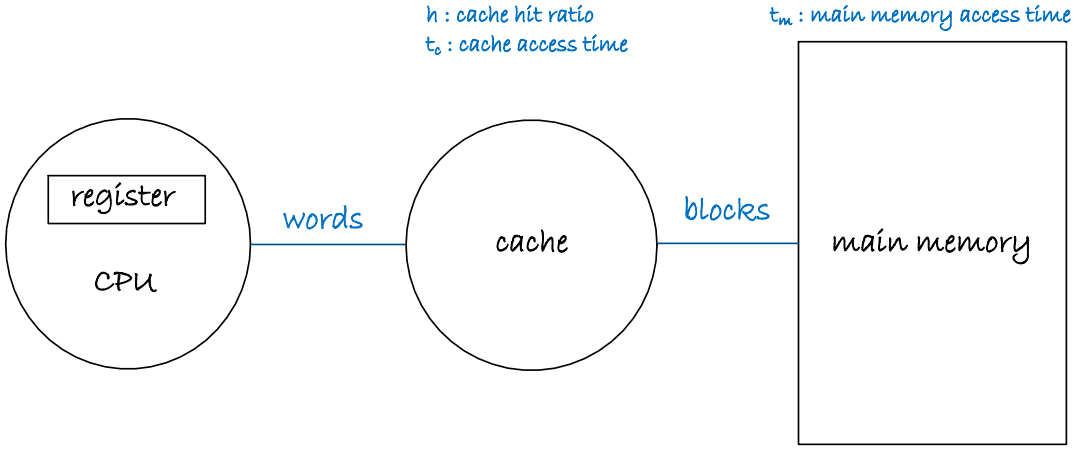


$T_{avg} = H_1t_1 + (1-H_1)H_2(t_2+t_1) + (1-H_1)(1-H_2)H_3(t_3+t_2+t_1) + \ldots\ldots (1-H_1)(1-H_2)\ldots\ldots(1-H_{n-1})H_n(t_n+t_{n-1}+\ldots.+t_3+t_2+t_1)$

pehle level mai mila

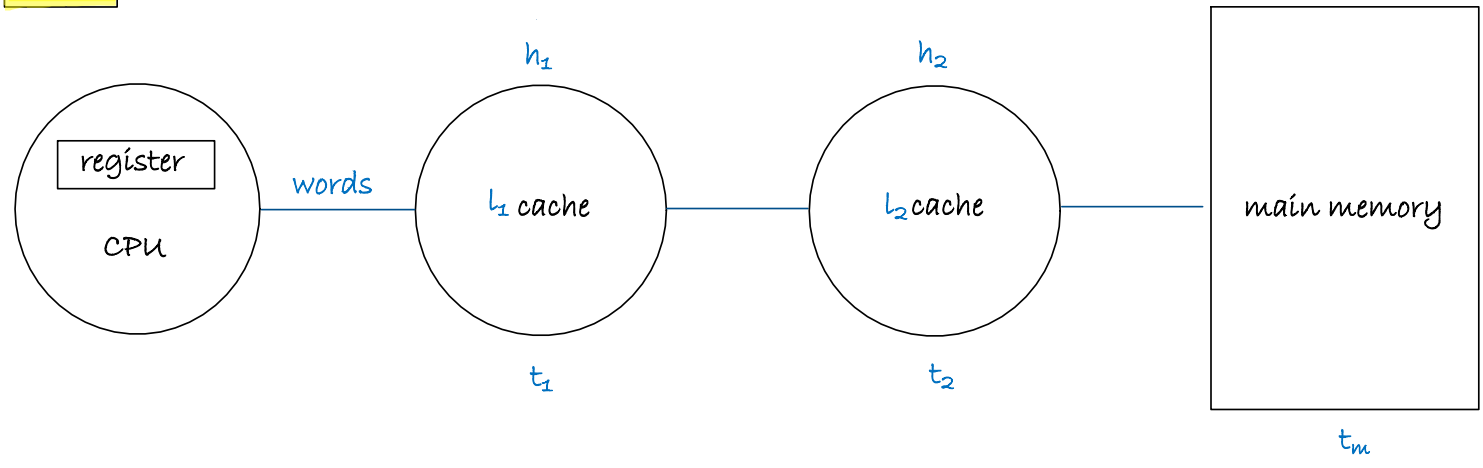
pehle level mai nahi mila

pehle aur dusre level mai nahi mila



$T_{avg} = h \times tc + (1-h)\,[t_m + t_c]$ (or) $T_{avg} = tc + (1-h)tm$

3 level



$T_{avg} = h_1 t_1 + (1-h_1)\, h_2\, (t_2+t_1) + (1-h_1)(1-h_2)(t_m+t_2+t_1)$

h_3 is missing because last level hit ratio always 1

case (i) : if all data are available in level 1

$T_{avg} = h \times tc + (1-h)\ [t_m + t_c]$
$T_{avg} = 1 \times tc + 0$
$T_{avg} = tc$

level 1 mai hi sab mil gaya, main memory mai jaane ki zarurat nahin.

q. hit ratio : 80%
tc : 2nsec
tm : 100nsec
by using hierarchical access.

$T_{avg} = h \times tc + (1-h)\ (tm + tc)$
$T_{avg} = 0.8 \times 2 + 0.2\ (102)$
$T_{avg} = 1.6 + 20.4$
$T_{avg} = 22nsec$

(or)

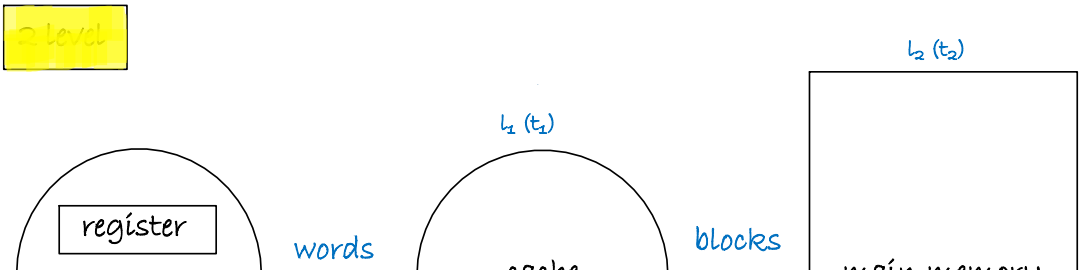
$T_{avg} = tc + (1-h)tm$
$T_{avg} = 2 + 0.2\ (100)$
$T_{avg} = 2 + 20$
$T_{avg} = 22nsec$

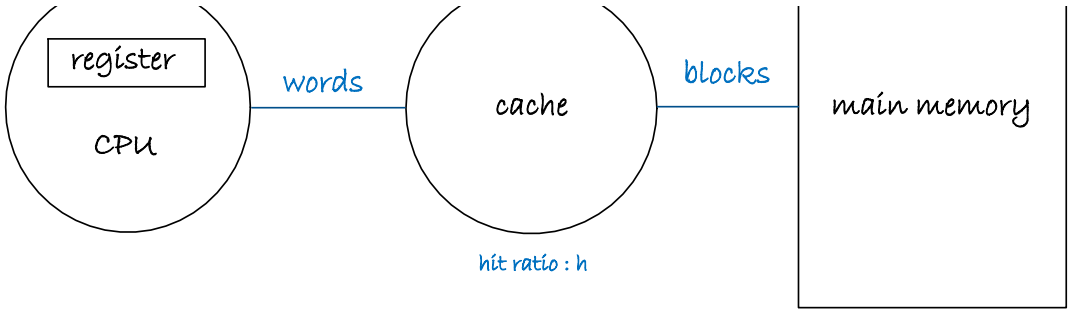
case (ii) : if 80% found in level 1 and remaining in level 2.

$T_{avg} = h \times tc + (1-h)\ (tm + tc)$
$T_{avg} = 0.8 \times tc + 0.2\ (tm + tc)$

(or)

$T_{avg} = tc + (1-h)tm$
$T_{avg} = tc + 0.2\ (tm)$





$T_{avg} = ht_1 + (1-h)(t_2+t_1)$ (or) $T_{avg} = t_1 + (1-h)\,t_2$

Q. Calculate the average Access time with the cache access time 1ns, and main memory access time 100ns, Hit ratio 90%? Using Hierarchical Access?

$T_{avg} = h \times tc + (1-h)[t_m + t_c]$
$T_{avg} = 0.9 \times 1 + (1-0.9)[100+1]$
$T_{avg} = 0.9+0.1x101$
$T_{avg} = 0.9+10.1$
$T_{avg} = 11nsec$

(or)

$T_{avg} = tc + (1-h)tm$
$T_{avg} = 1 + 0.1 \times 100$
$T_{avg} = 1 + 10$
$T_{avg} = 11nsec$

Q. Calculate the average Access time with the cache access time 1ns, and main memory access time 100ns, Hit ratio 80%? Using Hierarchical Access?

$T_{avg} = h \times tc + (1-h)[t_m + t_c]$
$T_{avg} = 0.8 \times 1 + (1-0.8)[100+1]$
$T_{avg} = 0.8+0.2x101$
$T_{avg} = 0.8+20.2$
$T_{avg} = 21nsec$

(or)

$T_{avg} = tc + (1-h)tm$
$T_{avg} = 1 + (1-0.8)100$
$T_{avg} = 1 + 0.2 \times 100$
$T_{avg} = 1 + 20$
$T_{avg} = 21nsec$

(Q.3) If Hit Ratio = 60% of Cache, tc = 1ns, tm = 100nsec, Hierarchical then T_{avg}.

$T_{avg} = h \times tc + (1-h)[t_m + t_c]$
$T_{avg} = 0.6 \times 1 + (1-0.6)[100+1]$
$T_{avg} = 0.6+0.4x101$
$T_{avg} = 0.6+40.4$
$T_{avg} = 41nsec$

$T_{avg} = 0.6+0.4X101$
$T_{avg} = 0.6+40.4$
$T_{avg} = 41nsec$

(or)

$T_{avg} = tc + (1-h)tm$
$T_{avg} = 1 + (1-0.6)100$
$T_{avg} = 1 + 0.4 \times 100$
$T_{avg} = 1 + 40$
$T_{avg} = 41nsec$

Q. In a 2 level memory, level 1 memory is 5 times faster than level 2. and its access time is 10ns < Average Access Time. Let level 1 Access time is 20ns, What is the hit ratio? Using simultaneous Access org?

$l_1 = t_1$
$l_2 = t_2$

$$5 = \frac{\text{perfomance of } l_1}{\text{perfomance of } l_2} = \frac{1/t_1}{1/t_2} = \frac{t_2}{t_1} \quad t_2 = 5t_1$$

$t_1 = T_{avg} - 10$
$T_{avg} = T_1 + 10$

$T_1 = 20nsec$
$T_2 = 5 \times 20 = 100ns$

$T_{avg} = 20 + 10 = 30$

$T_{avg} = h \times t1 + (1-h)t_2$
$30 = h \times 20 + (1-h)\ 100$
$30 = 20h + 100 - 100h$
$70 = 80h$
$h = 70/80 = 0.875$

note : cache is fast memory
- when hit ratio of cache is high that means it will take less time to access data
- when hit ratio of cache is low that means very less number of time data is found in cache, so in the case access time increases.

Q. Consider a system with 2 levels. Level 1 Access time is 20ns Level 2 Access time = 150ns T_{avg} = 30 using simultaneous Access.

(i) What is the Hit Ratio?

(ii) If the Hit Ratio is mode to 100% then what is the Access time of L_1 & L_2 Memory?

(i) What is the Hit Ratio?

$t_1 = 20nsec$
$t_2 = 150nsec$
$T_{avg} = 20nsec$

$T_{avg} = h \times t_1 + (1-h)t_2$
$30 = h \times 20 + (1-h)\ 100$
$30 = 20h + 100 - 100h$
$70 = 80h$
$h = 70/80 = 0.875$

(ii) If the Hit Ratio is mode to 100% then what is the Access time of L_1 & L_2 Memory?

$T_{avg} = h \times t_1 + (1-h)t_2$
$T_{avg} = 1 \times 20 + 0 \times 150$
$T_{avg} = 20\ nsec$

Q. If the above Question if T_{avg} is increased by 10% then what is % of change in Hit Ratio?

$T_{avg} = 30 + 10\%\ of\ 30$
$T_{avg(new)} = 30 + 3$
$T_{avg(new)} = 33$

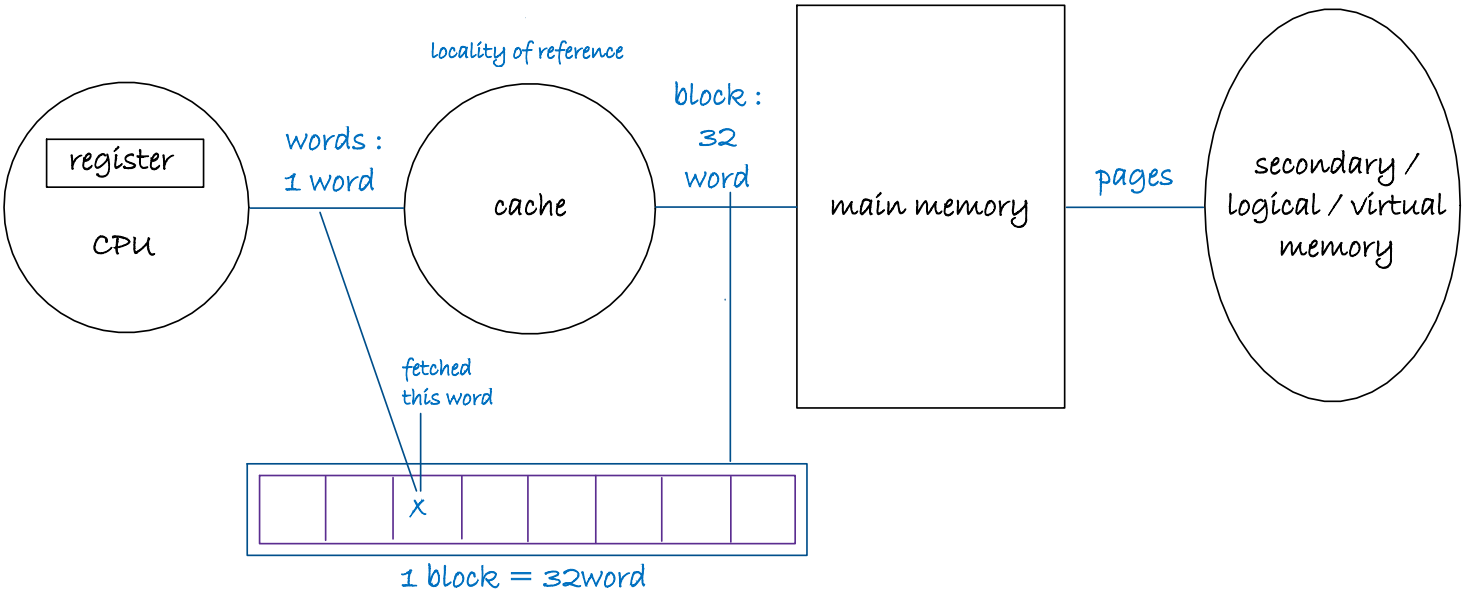
$T_{avg(new)} = h_{new} \times tc + (1-h_{new})\ tm$
$33 = h \times 20 + (1-h)\ 150$
$33 = 20h + 150 - 150h$
$130h = 117$
$h = 117/130 = 0.9 = 90\%$

new hit ratio : 90%
old hit ratio : 92.33%
percentage of change : 3.22%

T_{avg} increases so hit ratio decreases.

$$\text{hit ratio} \propto \frac{1}{T_{avg}}$$

topic : locality of reference



main memory to cache memory = 1 complete block
but cache memory to CPU 1 word requested/demanded

assume block size = 32word then 1 complete block (32word) will be transferred from main memory to cache memory

but only 1 word that is requested/demanded by the CPU given from cache memory to CPU.

blocks size >>>>>> word size

locality of reference : accessing the higher level of memory data from level 1 memory is called locality of reference. (data kahin bhi ho hum cache memory se lenge)

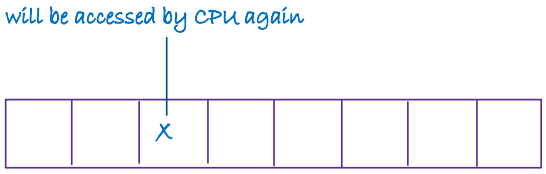
types :
(i) temporary LOR
(ii) spatial LOR

(i) temporary LOR : means the same word in the same block is reference by the CPU in near future (frequently)

(or)

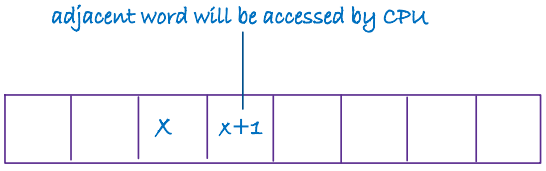
same data which access again and again then type of data stored in temporary LOR

will be accessed by CPU again



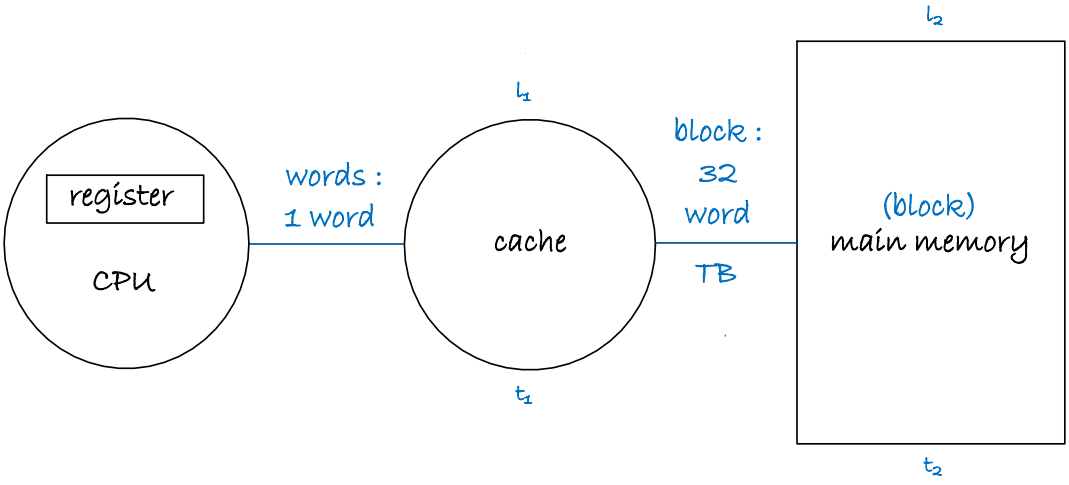
1 block = 32word

(ii) spatial LOR : means the adjacent word in the same block is reference by the CPU in a sequence.



1 block = 32word

CPU always access the data from the cache (faster/level one) memory. if there is a miss in level one (cache) memory and hit in main memory (level 2 memory) then one complete block is transferred from l_2 memory to l_1 memory and addressed word (with request/demanded) by the CPU given from level 1 (cache) to CPU.



TB : block transfer time from l_2 memory to l_1 memory

case (i) : if block size is 1 word

$TB = T_2$

case (ii) : if block size is n words

$TB = n \times T_2$

2 level

$T_{avg} = ht_1 + (1-h)\ (t_2+t_1)$ (or) $T_{avg} = t_1 + (1-h)\ t_2$

if locality of reference considered

$T_{avg} = ht_1 + (1-h)\ [tb + 1]$ (or) $T_{avg} = t_1 + (1-h)\ tb$

3 level

$T_{avg} = h_1\ t_1 + (1-h_1)\ h_2\ (t_2+t_1) + (1-h_1)\ (1-h_2)\ (t_m+t_2+t_1)$

if locality of reference considered

$T_{avg} = h_1\ t_1 + (1-h_1)\ h_2\ (tb_1+t_1) + (1-h_1)\ (1-h_2)\ (tb_2+tb_1+t_1)$

(or)

$T_{avg} = h_1\ t_1 + m_1\ h_2\ (tb_1+t_1) + m_1 m_2 (tb_2+tb_1+t_1)$

m_1 : miss in level 1
m_2 : miss in level 2

Q. In a 3 level memory, level 1 memory Access time is T1 , level 2 memory Access time is T2(TB1) and level 3 memory Access time is T3(TB2). Hit ratio of level 1 is h1 and Hit ratio of level 2 is h2 . What is the average Access time Using Hierarchical Access?
(i) If there is a hit in level1(h1=100%).
(ii) If there is a miss in level1 & hit in level2(h2=100%) .
(iii) If there is a miss in level1 and Level 2 & hit in level3 .

level 1 memory access time : T1
level 1 hit ratio : h1

level 2 memory access time : T2 (TB1)
level 2 hit ratio : h2

level 3 memory access time : T3 (TB2)

$T_{avg} = h_1t_1 + (1-h_1)h_2(t2+t1) + (1-h_1)(1-h_2)(t3+t2+t1)$
$T_{avg} = h_1t_1 + (1-h_1)h_2(TB1+t1) + (1-h_1)(1-h_2)(TB2+TB1+t1)$

(i) If there is a hit in level1(h1=100%).
$T_{avg} = t1$

(ii) If there is a miss in level1 & hit in level2(h2=100%)

$h_2 = 100\% = 1$
$h_1 = 0$
$T_{avg} = h_1t_1 + (1-h_1)h_2(t2+t1)$
$T_{avg} = 0xt_1 + (1-0)1(t2+t1)$
$T_{avg} = 1(t2+t1)$
$T_{avg} = t2+t1$

(iii) If there is a miss in level1 and Level 2 & hit in level3 .
$h_3 = 1$
$h_2 = 0$
$h_1 = 0$
$T_{avg} = h_1t_1 + (1-h_1)h_2(t2+t1) + (1-h_1)(1-h_2)(t3+t2+t1)$
$T_{avg} = 0xt_1 + (1-0)0(t2+t1) + (1-0)(1-0)(t3+t2+t1)$
$T_{avg} = 1x0(t2+t1) + (1)(1)(t3+t2+t1)$
$T_{avg} = t3+t2+t1$

with locality of reference :

level 1 memory access time : T1
level 1 hit ratio : h1

level 2 memory access time : T2 (TB1)
level 2 hit ratio : h2

level 3 memory access time : T3 (TB2)

$T_{avg} = h_1t_1 + (1-h_1)h_2(TB1+t1) + (1-h_1)(1-h_2)(TB2+TB1+t1)$

(i) If there is a hit in level1(h1=100%).
$T_{avg} = t1$

(ii) If there is a miss in level1 & hit in level2(h2=100%)

$h_2 = 100\% = 1$
$h_1 = 0$
$T_{avg} = h_1t_1 + (1-h_1)h_2(TB1+t1)$
$T_{avg} = 0xt_1 + (1-0)1(TB1+t1)$
$T_{avg} = 1(t2+t1)$
$T_{avg} = TB1+t1$

(iii) If there is a miss in level1 and Level 2 & hit in level3 .
$h_3 = 1$
$h_2 = 0$
$h_1 = 0$
$T_{avg} = h_1t_1 + (1-h_1)h_2(TB1+t1) + (1-h_1)(1-h_2)(TB2+TB1+t1)$

$h_3 = 1$
$h_2 = 0$
$h_1 = 0$
$T_{avg} = h_1t_1 + (1-h_1)h_2(TB1+t1) + (1-h_1)(1-h_2)(TB2+TB1+t1)$
$T_{avg} = 0xt_1 + (1-0)0(TB1+t1) + (1-0)(1-0)(TB2+TB1+t1)$
$T_{avg} = 1x0(TB1+t1) + (1)(1)(TB2+TB1+t1)$
$T_{avg} = TB2+TB1+t1$

Q. In a 2 level memory, level 1 memory Access time is 30ns and level 2 memory Access time is 250ns/word. Hit ratio of level 1 is 90% . If there is a miss in level1 then 4word block must be transferred(moved) from level 2 into level1 and then addressed word is given to CPU. What is the average Access time ?

level 1 memory access time : $T1 = 30nsec$
level 1 hit ratio : $h1 = 90\%$

level 2 memory access time : $T2 = 250nsec/word$

miss in level 1 then 4 word block must be transferred from level 2 to level 1

(hum T_1, T_2 aur T_{avg} ek word ka nikalte hai lekin yahan jo box aayega vo n word ka hoga, T_1 mai sirf one word aa raha lekin T2 mai 4 words)
if block size is n words : $TB = n \times T2$
$TB = 4 \times 250 = 1{,}000$ nsec

$T_{avg} = hT_1 + (1-h_1)(TB+T1)$
$T_{avg} = 0.9 \times 30 + (1-0.9)(1000+30)$
$T_{avg} = 27 + (0.1)(1030)$
$T_{avg} = 27 + 103$
$T_{avg} = 130nsec.$

(or)

$T_{avg} = T_1 + (1-h)TB$
$T_{avg} = 30 + (1-0.9)1000$
$T_{avg} = 30 + 0.1 \times 1000$
$T_{avg} = 30 + 100$
$T_{avg} = 130nsec.$

Q. Assume that for a certain processor, a read request takes 50 nanoseconds on a cache miss and 5 nanoseconds on a cache hit. Suppose while running a program, it was observed that 80% of the processor's read requests result in a cache hit. The average read access time in nanoseconds is _____.
(i) What is Data Transfer rate (performance) of this memory system (in words/sec)?
(ii) What is Bandwidth required of this memory system if word size is 8bit?

cache miss: 50nsec
cache miss: 20%

cache hit: 5nsec
cache hit: 80%

$T_{avg} = h \times t_1 + (1-h)t_2$
$T_{avg} = 0.8 \times 5 + (1-0.8)(50)$
$T_{avg} = 4 + (0.2)(50)$
$T_{avg} = 4 + 10$
$T_{avg} = 14nsec.$

(i) What is Data Transfer rate (performance) of this memory system (in words/sec)?

Data transfer rate : $\frac{1}{T}$

T_{avg}

Data transfer rate : $\frac{1}{14 \times 10^{-9}}$ words/sec

$= 1/14 \times 10^{9}$ words/sec

$= 1000/14 \times 10^{6}$ words/sec

$= 71.4 \times 10^{6}$ words/sec

= 72million words per second

(ii) What is Bandwidth required of this memory system if word size is 8bit?

word size : 8bit = 1byte
$= 72 \times 10^{6}$ words/sec
$= 72 \times 10^{6}$ byte/sec
= 72 mbps

Q. A direct mapped cache memory of 1 MB has a block size of 256 bytes. The cache has an access time of 3 ns and a hit rate of 94%. During a cache miss, it takes 20 ns to bring the first word of a block from the main memory, while each subsequent word takes 5 ns. The word size is 64 bits. The average memory access time in ns (round off to 1 decimal place) is ____. [GATE-2020]

cache memory : 1mb
block size : 256bytes

access time : 3ns
hir rate : 94%

cache miss : 20ns for first word and 5ns for remaining words
word size : 64bits = 8bytes

words in a block : $\frac{\text{block size}}{\text{word size}} = \frac{256B}{8B} = 32$ words

if block size is n words : TB = n x T2

first word : 20n = 1 x 20 = 20
remaining words : 5ns = 31 x 5 = 155

$T_{avg} = hT_1 + (1-h_1)(TB+T1)$
$T_{avg} = 0.94 \times 3 + (1-0.94)[20+(31 \times 5)+3]$
$T_{avg} = 2.82 + (0.06)[20+(155)+3]$
$T_{avg} = 2.82 + (0.06)[178]$
$T_{avg} = 2.82 + 10.68$
$T_{avg} = 13.5$

(B) If the cycle time of Memory is 500nsec then what is the Bandwidth (maximum Rate at which Cpu Access ___?? Byte/sec)

cycle time : 500nsec
500nsec (cycle time) access = 1byte

in one second : $\frac{1}{500 \times 10^{-9}} = \frac{1000 \times 10^{6}}{500}$ byte/sec

2mbps

Q. A certain processor deploys a single-level cache. The cache block size is 8 words and the word size is 4 bytes. The memory system uses a 60-MHz clock. To service a cache miss, the memory controller first takes 1 cycle to accept the starting address of the block, it then takes 3 cycles to fetch all the eight words of the block, and finally transmits the words of the requested block at the rate of 1 word per cycle. The maximum

Q. A certain processor deploys a single-level cache. The cache block size is 8 words and the word size is 4 bytes. The memory system uses a 60-MHz clock. To service a cache miss, the memory controller first takes 1 cycle to accept the starting address of the block, it then takes 3 cycles to fetch all the eight words of the block, and finally transmits the words of the requested block at the rate of 1 word per cycle. The maximum bandwidth for the memory requested block at the rate of 1 word per cycle. The maximum bandwidth for the memory system when the program running on the processor issues a series of read operations is _____$\times 10^6$ bytes/sec. [GATE-2019-CS: 2M]

cache block size : 8 words
1 word size : 4 byte
cache block size : 8 x 4 = 32 byte
memory system uses : 60MHz clock

cycle time = $\frac{1}{60 \times 10^6}$ sec

total time taken to transfer block : 1 cycle (accept the address) + 3 cycle (to fetch the complete block) + 8X1 (1 word per cycle) = 12 cycle

transferring 32 byte in 12 cycle

$12 \times \frac{1}{60 \times 10^6}$ sec

in one sec = $\frac{32 \times 60 \times 10^6}{12}$

$= 160 \times 10^6$ byte/sec