Computer Architecture (CA)

1st → functional blocks & Digital Computer
- → stored programmed Concept
- → Accumulation Based on cpu
- → Instruction cycle

2nd → Instruction format
- → Different types of A/c processor (zero, one, two, three instruction format)
- → Addressing modes

3rd → memory organisation
- . need of memory Hierarchy
- . Associative memory
- . cache memory
- . memory Inter leaving

4th → I/O organisation
- . Need for I/O interface
- . Data Transfer method
  - i) programmed I/O
  - ii) Interrupt Driven I/O
  - iii) DMA
- → pipeline processor & parallel process

5th → Data Representation
- . Fixed point
- . Floating point

computer : It is a computing machine which process the data under control of application pgm which is introduced by the user

→ the functionality of the computer pgm execution

pgm : sequence of instructions along with data

Instruction : It is a set of Binary ~~operations~~ sequence inside the processor to perform some operations

ex : CPU-X support 8 different operations with opcode

∴ No. of bits required in opcode size - 3-bits

16 OP = 4bit
32 OP = 8-bit

opcode : operation to be performed. No. of bits require to represent the operation is called "opcode"

(Instruction)

→ operations are stored in ROM

→ consider to hiphical computer which have a bit opcode

→ to many instructions are passed.

opcode size = $\log_2$ (No. of instruction/operations)

$12 = \log_2(N)$ $\quad [2^{10} = 1024 \text{ KB}]$

$N = 2^{12}$, $2^2 \times 2^{10} = 4 \times 1024 = 4096$.

Data Encoding : In this process N signals are represents with $\log_2 N$ bits format. $\equiv \square \rightarrow \log_2 N$

Decoding : In this process N-bit decoder to produce 2^N

Data : the set of Binary sequence along with value

CA : It deals with system attributes which are visible to programmer & this attributes have direct impact of programmer

CO : It deals with operational units of OS & its After connection I/O org, memory org, performance of computer.

→ Based on the program storage structure divided into two types :

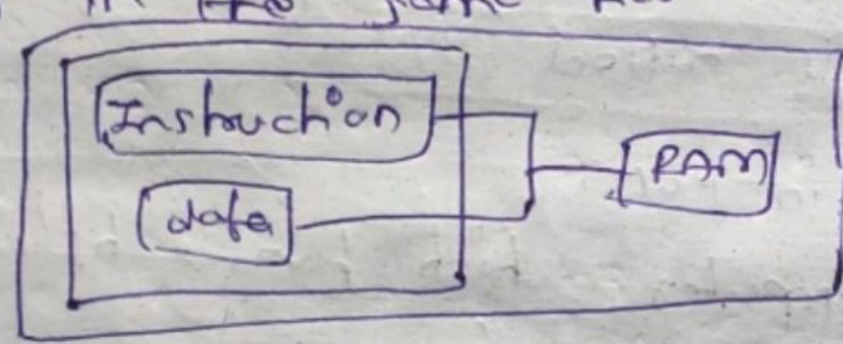
1) von-Neumann Architecture

→ In this architecture application pgms are stored in the main memory if it is require

→ Main memory is a RAM (volatile). It allows read & write operation so we can change the application operation in the main memory.

→ In this architecture CPU always executed the main memory at the part of the pgm.

→ In this architecture we can run different application operation in the same hardware



Ex : 8085, 8086 microprocessor

2) Harvard Architecture

→ pgm logic & operation data both are present in the separated memory is called as code memory (ROM) & data memory (RAM) respectively.

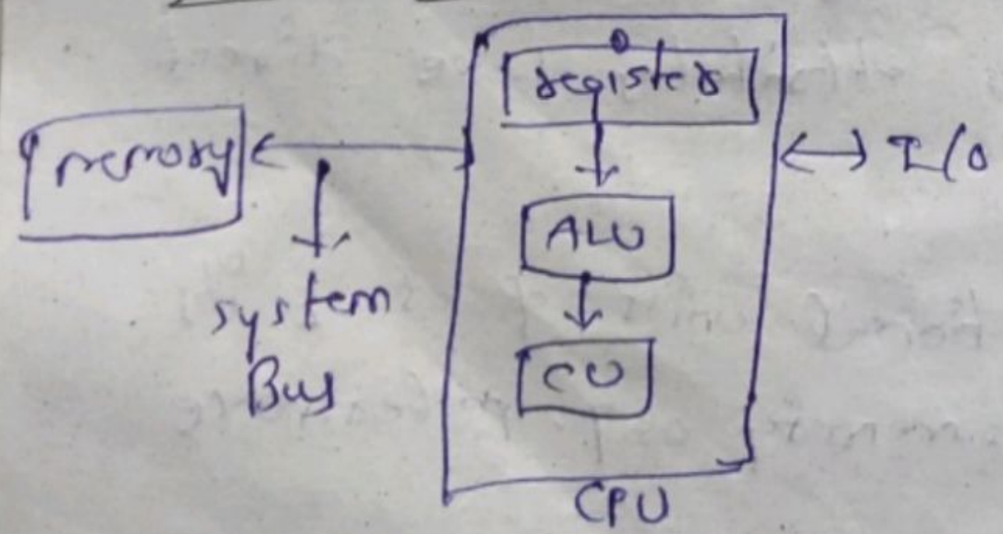
→ Application pgms stored in ROM & run in RAM

→ CPU is always execute the predefine pgm the code memory on different data elements

Ex : micro controller 8051 — 4KB - ROM, 128 B - RAM

Compounds of a computer

Digital computer



Compounds of a computer

1) CPU
2) memory — primary, Secondary
3) I/O Devices
4) system Bus

divided

d in the

s read &

n

the

pplication

8086

coprocessor

the

ROM)

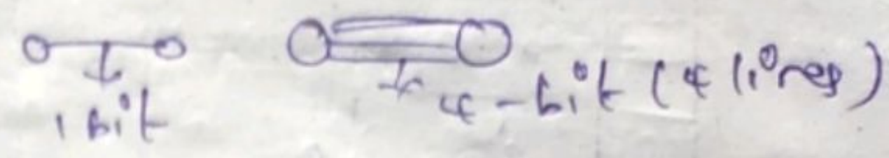
code

uter

mory

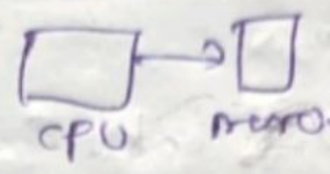
ordary

System Bus : connection lines b/w the two devices with in the computer system



→ system Bus divided into 3 types :
1) Address Bus 2) Data bus 3) Control bus

1) Address Bus : uni dimensional bus it carry the info. from cpu to memory.



→ Based on the address lines we determine the capacity of the memory. Ex : 8085, $AD_0, AD_1, AD_2 \cdots AD_7$ 8 lines

2) Data Bus : This Bus used to carry the binary sequence b/w the cpu & memory, I/O devices

→ Based on the no. of Data lines we can determine what length of the processor. It is a bidirectional bus

Ex : 8086 - 16 bit

3) Control Bus : This are used to carry the control signals timing signal b/w the cpu & memory of I/O devices

→ control signal is used to indicate the type of operation & timing signal are used to synchronization the memory & I/O operation with the processors block

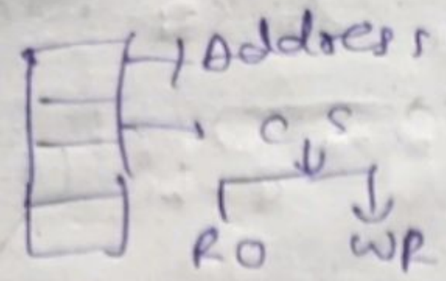
memory : the pgms are stored in the memory. every memory chip is divided into equal parts called cells.

→ Each cell is associated with unique number is called "address"

→ Each cell is able to resize control signal to indicated the type of operation read signal & write signal

→ when the cpu works to executed a pgm generate memory request to get the instruction from memory

→ thus, the cpu memory request contain address along with control signal

1) 

2) CPU memory

address	CS

→ where the instruction is present in the memory

Control signal :- It is used to indicate type of operation

Memory chip configuration :

Memory capacity : $\frac{\text{No. of blocks}}{\text{Address port}} \cdot \frac{\text{each block size}}{\text{Data port}}$

1) memory size is 64KB i) No. of address line ii) Address bits iii) Data size

64KB
$2^6 K$
$2^6 \times 2^{10} = 2^{16}$

8 bit

no. of address line $= 2^{16}$
address bits = 16 bits
data size = 8 bit

Total
1) Data space in a chip = 64 KB
2) No. of address bits required for cell = 16 bits
3) no. of address bits required for chip = 16 bits
a) Data bus size = 8 bits

2) memory size (or) pgm stored in memory – 64KB
↓
64K × 16

2 word = 16-bit
= 2 bytes

no. of address line $= 2^{16}$
Data size = word size = 16 bits

3) 128 K × 32
↓
$2^7 \times 2^{10} = 2^{17}$ – address lines

Address bits = 12
Data bus = 32 bits

W4K × 4
R4CA × 8
3) 256 M × 6

MOV AX, [2000] A = [AH: 40 | AL: 50]
MSB LSB

→ their is the no information available which address contain lower bit & higher bit
→ their is a possibility turbulity to avoid this their is read of memory address representations

→ two types endian:
→ the order of a low address certain lower bit high address contain higher bit

ENDIAN: the order of a data storage low address contain higher bit order lower bit

2/9/25

CPU	bit	nibble	Byte	word
8-bit	1-bit 8-cells	4-bit 2 cells	8-bits 1 cell	8-bits 1 cell
16-bit	1-bit 16 cells	4-bit 4 cells	8-bits 2 cells	16-bits 1 cell
32-bit	1-bit 32 cells	4-bit 8-cells	8-bits 4-cells	32-bits 1-cell

Byte addressable

→ ~~Byte addressable~~

8-bit	16-bit	32-bit
1 block	2 blocks	4 blocks

→ Byte addressables are reusable in all the cpu's by changing the interfacing. therefore system enhancement cost is low & performance is high

Word Addressable

→ Redesigning the memory with respect to the word length & the CPU (No reusable memory chip). therefore system cost is high & performance is also high

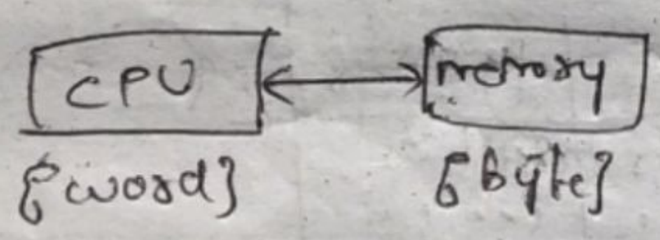
8-bit	16-bit	32-bit
1 block	1-block	1-block

Note :

→ Default memory configuration in the system Design is Byte Addressable. It means the data is stored in the memory byte wise sequence. So, the memory datatype is "byte"

→ CPU always performs the operations on a word formatted data. So, CPU datatype is "word"

→ CPU datatype is synchronized with memory datatype by implementing interfacing b/w them i.e., multiple memory cells are interfaced to CPU, with respect to word size of a CPU.

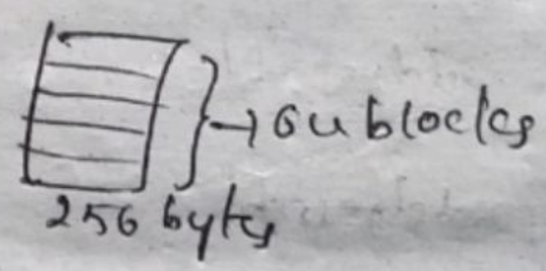


storage = byte
Accessing = word

Q) The memory size is 256 bytes word addressable memory 1 ~~address~~ word = 4 bytes, no. of cells in memory & address size is

Sol) memory size = 256 bytes (word addressable)

1 word = 4 bytes



→ no of cells = $\frac{256}{4} = 64$

→ address size = $\log_2(64)$

$= \log_2 2^6 = 6\log_2 2 = 6 \times 1 = 6$ bits

d) 2GB, word size = 8 bytes

address bits = 28 bits $\frac{2GB}{8B} = \frac{2^1 \times 2^{30} \times B}{8B} = \frac{2^{31} B}{8B} = 2^{28}$

e) 128Gb, word size = 16 bits, if the memory capacity is words, bytes, & bits & no. of bits required for addressing

memory size = 128Gbits

word size = 16 bits

i) bit addressable =

$\rightarrow 128Gb = 2^7 \times 2^{30} = 2^{37} = 37$ bits

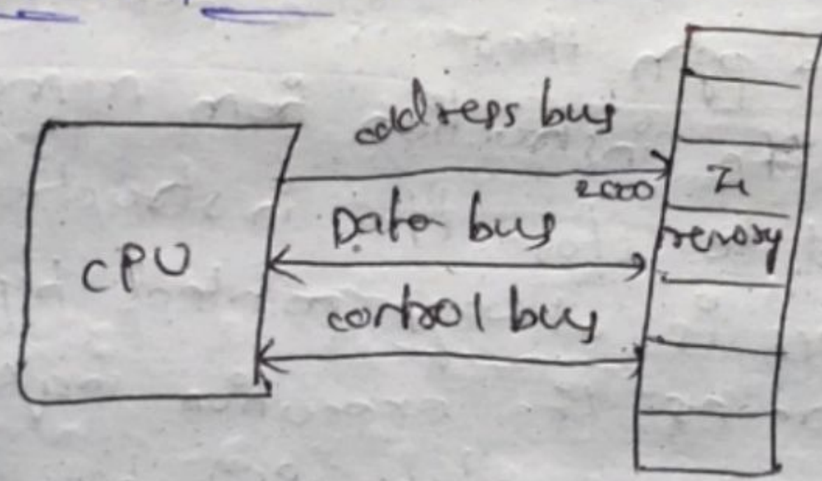
ii) Byte Addressable =

$\frac{128G\text{ bits}}{8\text{-bits}} = \frac{2^7 \times 2^{30}}{8b} = 2^4 \times 2^{30} = 16G = 34$ bits

iii) word Addressable =

$\rightarrow \frac{128G\text{ bits}}{16\text{ bits}} = \frac{2^7 \times 2^{30}}{2^4} = 2^3 \times 2^{30} = 2^{33} = 33$ bits.

Memory Read operation



→ CPU sends the address to memory through Address bus

→ CPU enables the read control signal to memory

→ memory performs read operation in given address & content of the data send to the CPU using data bus

Memory write operation

→ CPU sends the address to memory through Address bus

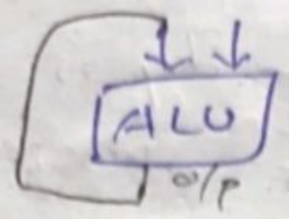
→ CPU sends the data to memory using Data bus

→ CPU enables the write signal to memory

→ memory perform the write operation on provided address

CPU Registers

1) Accumulator :- It is used to store the result of ALU & sometimes one of the i/p operand for the ALU



2) memory address register :- It is used to store the address to access memory

→ It holds the address of the location to be access from memory

3) memory data register :- It is used to send the data to the memory (memory write operation). & to recieve the data from the memory

4) IR (Instruction Register) : It is used to hold the current instruction when the CPU is executing.

5) program counter :-

→ when CPU executing a pgm there is a need of instruction address to get the instruction from memory. But these memory is used to store the instruction & data

→ the instruction address is always pointed by a program counter register. that means the execution sequence always depends on program counter (PC) bcoz CPU processing the instruction based on PC address/value

→ if the memory cell is enabled based on the PC

→ the pgm execution sequence depends on the PC, the PC functionality is pointed to the starting instruction address & immediate point to the next instruction address

→ the PC implies in two modes :

1) Implicity Mode :

$$PC = PC + \text{step size}$$

2) Explicity mode

$$PC = PC + \text{target address}$$

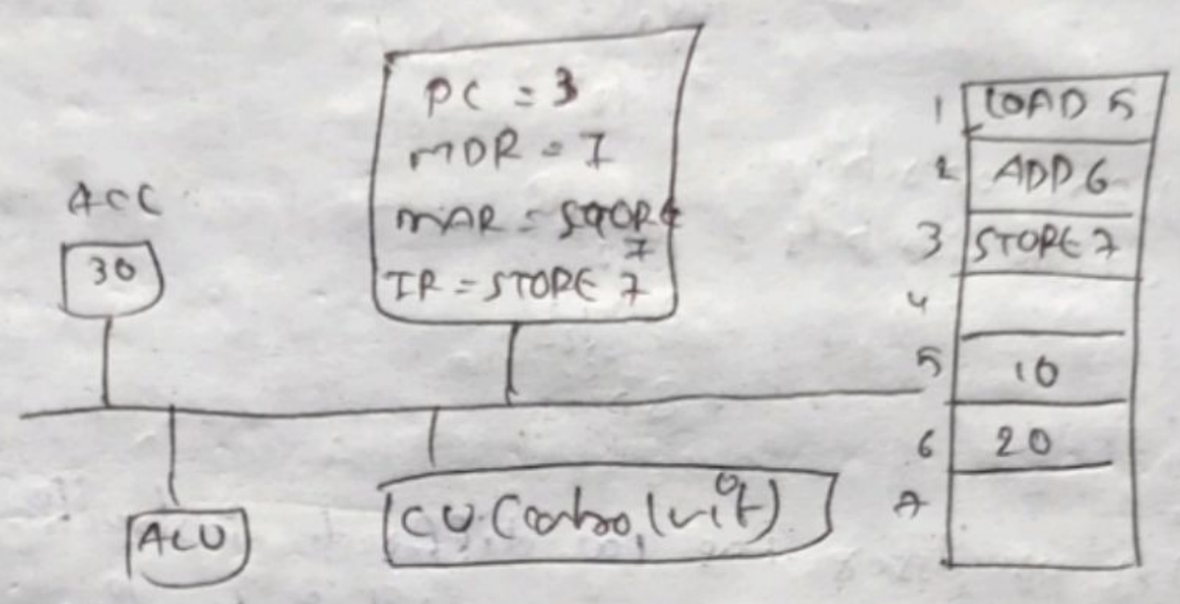
→ It stores the address of the next instruction to be executed.

6) Flag register : It is used to store the status of ALU result

7) stack pointer : It is used to store address of top of processor stack

Fetch cycle :

1) MAR ← PC
2) MDR ← [MAR] [1]
   PC = PC + 1
3) IR ← MAR



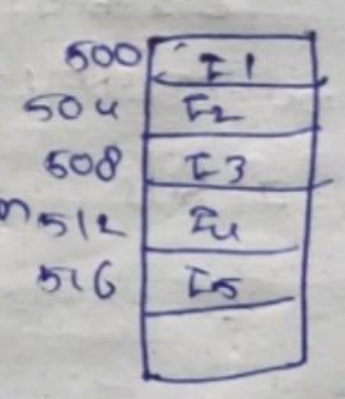
Execution cycle

1) MAR ← IR[address]
   LOAD 5
2) MDR ← MAR
3) ACC ← (ALU)(+)MDR

Q) If CPU has 4 byte instruction. A program (I_1 to I_{200}) starts of address 500 in decimal. Find the address of the instruction

a) I_1 = 500
b) I_5 = 516
c) I_{200} =
c) I_7 = 524

4-byte → size
500 (starting address



$I_5 = PC + (I_4)*4$ → Instruction no
$500 + 4*4$
$= 500 + 16$
$= 516$

$I_7 = PC + I_6 * 4$
$= 500 + 6*4$
$= 524$

$I_{200} = PC + I_{199} * 4$
$= 500 + 199*4$

Q) A CPU has 24-bit instruction. A program starts at address 300 which of the following is legal pgm counter

a) 400
b) 500
c) 600
d) 200

24-bit → 3 bytes

$\frac{400}{3}$ ✗

$\frac{600}{3} = 200$ ✓

Instruction format

opcode	address/operand	mode

mode → optional

1) 8 operations

opcode size bits = $\log_2$(no. of operations)

$= \log_2(2^3)$

$= 3\log_2 2 = 3$ bit

2) 256 operations

opcode size bits = $\log_2 2^8$

$= 8\log_2 2$

$= 8$ bits

Q) A processor has 100 different instructions & 32-general purpose registers. A 32-bit instruction word has an opcode, two registers & Immediate operand. The no. of bits available for immediate operand,

a) 14
b) 16
c) 12
d) 15

Sol) 100 operations
32 registers
32-bit

opcode	R_1	R_2	Imm. operand
7-bits	5	5	

$R_1 = \log_2 32 = 5$

100 operations $= 2^7 = 128$
7-bits

32 = opcode + R_1 + R_2 + Imm. operand

Imm operand = 32 − (opcode + R_1 + R_2)

$= 32 - (7 + 5 + 5)$

$= 15$ bit

Q) which of the following not involved in memory write operations

a) MAR
b) PC
c) MDR
d) Data bus

Q) What is the control unit function in CPU?

a) to transfer the data into primary storage
b) to store the program instruction
c) to perform the logical operation
d) to decode a program instruction

e) the sequence of events that has happen during a fetch operation

a) PC → MAR → MDR → IR

b) PC - memory → MDR → IR

c) - - - -

Types of Instructions

1) Data transfer Instructions ⟶ opcode D S

2) Data manipulation instructions → ALU (opcode), D, S_1, S_2

3) Transfer of control instructions → opcode Destination

ex: JUMP

→ stack organisation of CPU (zero Address Instruction format)

→ Accumulator orgⁿ of CPU (one Address Instruction format)

3) General Register orgⁿ of CPU

i) Register - Memory orgⁿ of CPU [Two - Address Instruction format]

ii) Register - Register orgⁿ of CPU (three - Address Instruction format]

stack orgⁿ of CPU

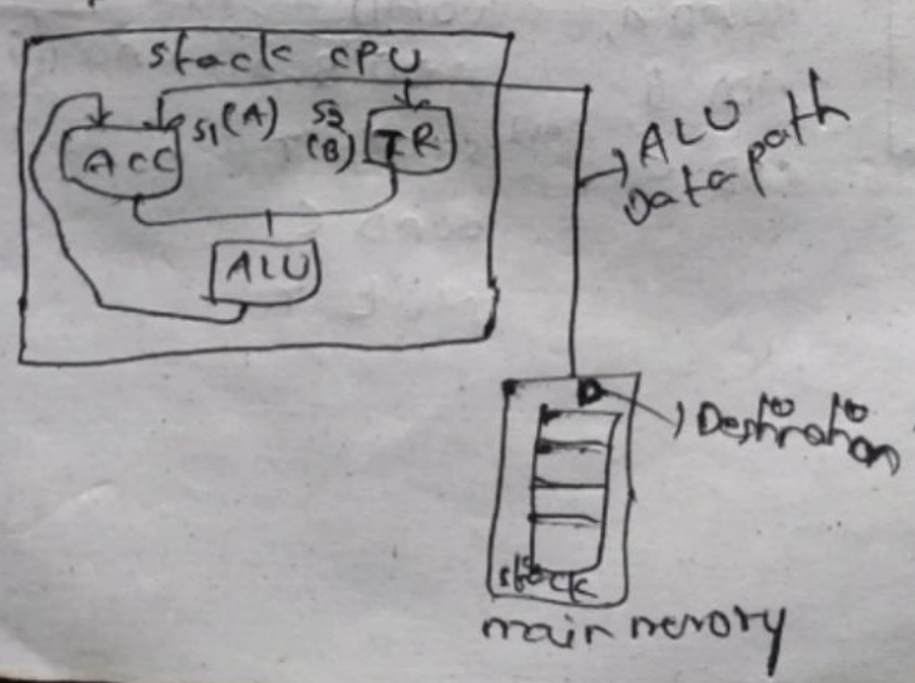
→ In this organisation ALU operations are performed only on the stack data

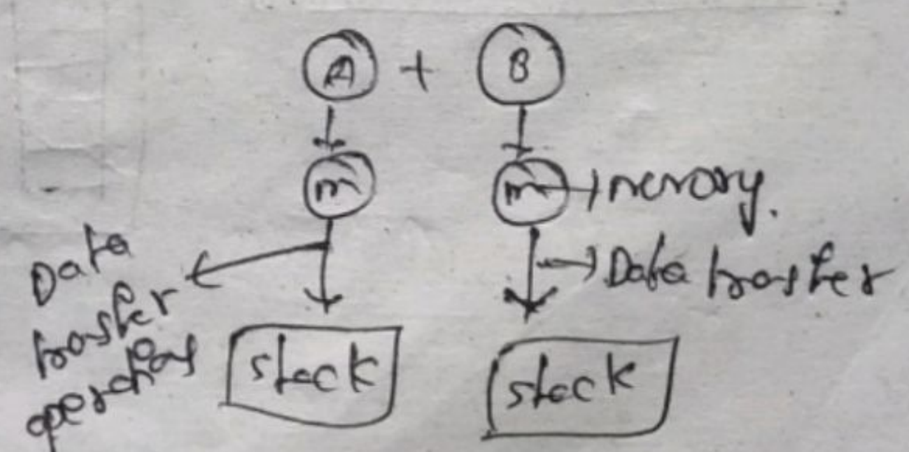
→ stack initialized in the memory using stack pointer

→ the compatable instruction format is only "opcode"

→ type of operation : ALU, D, S_1, S_2 ← top of the stack

→ In this orgⁿ data transfer operations are PUSH, POP





Accumulator orgⁿ of CPU

ALU D S1 S2

top

Acc Acc register/memory

X = A + B

ACC ← A LOAD A

ACC ← A+B ADD B

STORE X

→ In this orgⁿ the first operand is always required in Accumulator

→ 2nd operand required in register/memory

→ After data processing stored in Accumulator

→ In this CPU design only one Accumulator is default location. It address need not be required in the instruction.

→ | Opcode | One Address |

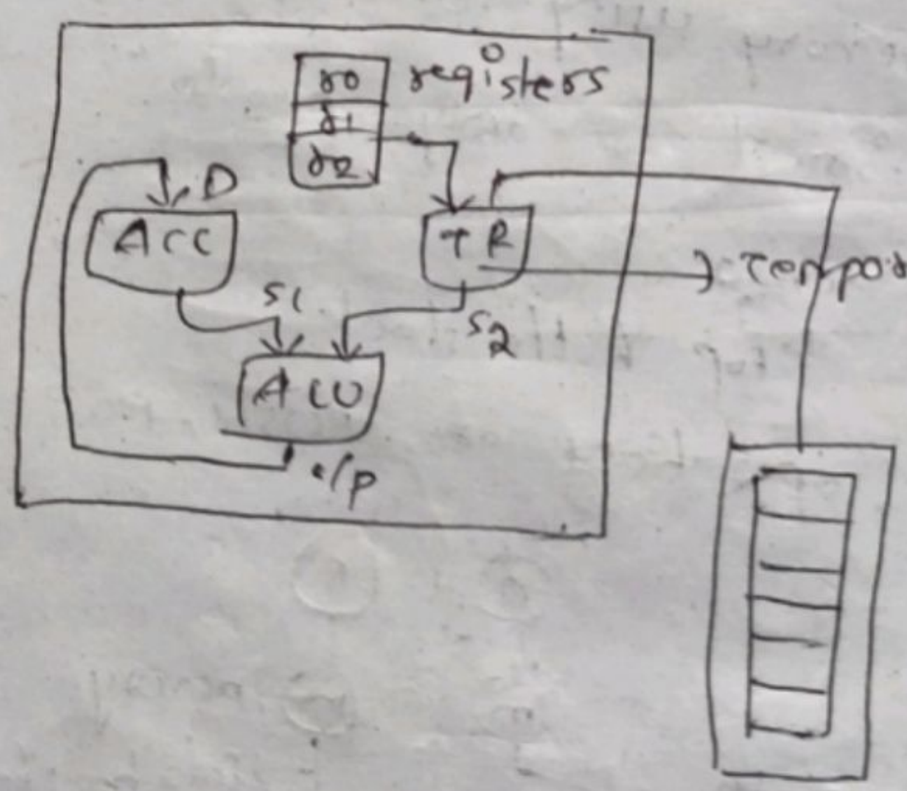
→ type of operation

→ Data transfer D S [LOAD, STORE]

→ Data Manipulation ALU D S1 S2

↓ ↓ ↓

Acc Acc register/memory



(A+B) * C

S1 S2 ↓

S1 S2

LOAD A, LOAD A, ACC ← A

ADD B, ADD B, ACC ← ACC + [B]

ADD C, (or) STORE T

LOAD C

MUL T.

Register - Memory orgⁿ of CPU

ALU D, S_1, S_2

D → S_1 ; S_1 → register ; S_2 → register/memory

$x = A \oplus B$

A → S_1 → register ; B → reg/mem

Instruction format:

opcode	Address 1	Address 2

→ Type of operation
- → Data transfer D, S, (mov)
- → Data manipulation AC D S_1 S_2 (D → reg, S_1 → reg, S_2 → reg/mem)

→ Data Transfer instructions (MOV)

MOV R0, A
ADD R0, B
MOV X, R0

→ In this orgⁿ ALU first operand in a register, 2nd operand in register/memory

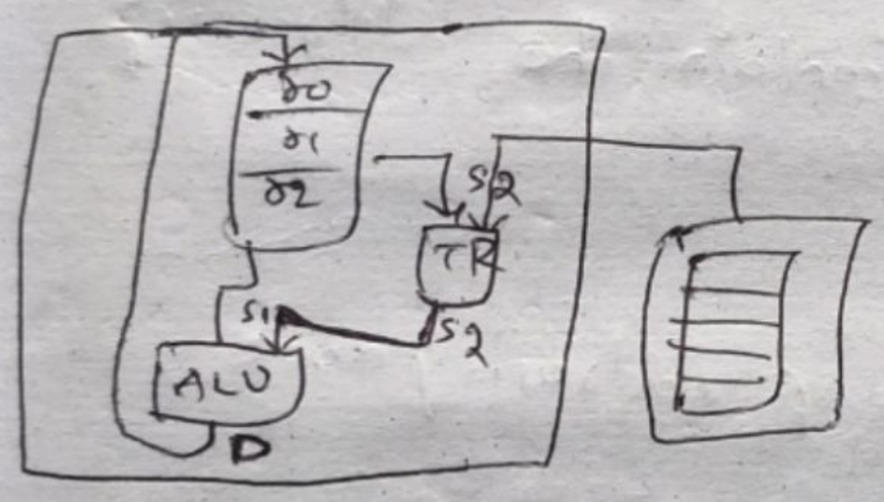
→ After Data processing the result is stored in S_1 location

→ S_1 is acting as a destination, the destination address is not required.

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

S_1 S_2 → 1st ; 2nd → 2nd ; 1st, 2nd → 3rd

→ ~~Instruction~~



MOV R0, A
ADD R0, B
MOV R1, C
ADD D
MUL R1, R2

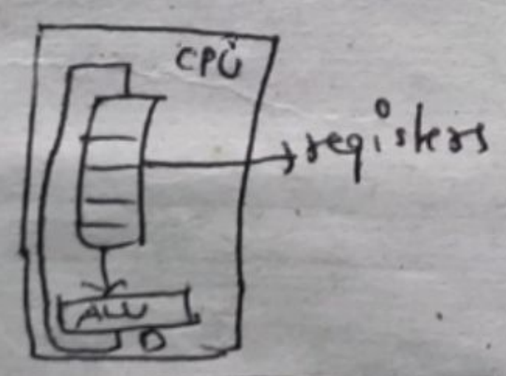
Register - Register orgⁿ of CPU

ALU D, S_1, S_2 → registers

A ⊕ B → Reg, Reg

LOAD
STORE

~~LOAD~~



LOAD R0, A
LOAR R1, B
ADD, R2, R0, R1

RISC → powerful processor

opcode	Ad1	Ad2	Ad3

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

ADD R0, A, B (D, S_1, S_2)

ADD, R1, C, D

MULX, R0, R2

Addressing Modes

→ To specify the location of data or location of the required object

→ In the computer instruction Design data transfer instructions, Data Manipulation Instruction, Data Transfer control instructions

→ Data transfer & manipulation instructions are sequential. This focus on the data

→ Transfer control Instructions focuses on the Instruction

→ The o/p of the addressing mode is a "effective address"

→ effective address is actual address of object.

object/data = [EA], content of [2000] = 20
effective address

→ Instruction Design modes

1) Implicit :- opcode itself contain the addressing mode.

~~Implicit add~~

opcode

↳ Type of operation, Type of Addressing mode

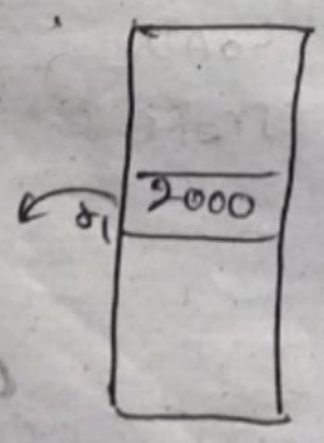
2) Explicit :-

opcode	Address	mode

↳ Type of Addressing mode

Ex :-

data = (EA)
= [EA]
= 2000



the
nsfer
sfer
quential.
structions
ective address
20
dressing
ode
ode

Addressing mode symbols

1) # → Immediate
2) [ ] → direct
3) ( ) → Indirect
4) R → register

A.M

- Sequential A.M
  - register based AM (EA = register)
  - memory based AM (EA = memory address)
    1) Implied
    2) Immediate
    3) direct
    4) Indirect
    5) base index
    6) Auto index
- Transfer control flow AM (focused on instruction)
  - i) Relative (PC Relative AM)
  - ii) Based Register AM

9/2/25

Implied Addressing mode :

opcode
- Type of operation
- Data Information

→ there is no address

ex: STC (set the carry flag), CLC (clear the carry flag)

stack operations (push, pop)

Immediate Addressing mode (fastest one)

opcode | Address → data information

EA = constant value

Data

→ this constant value

ex, 3bit 3bit

opcode | Address → 0 to 2^3-1 → 0 to 7 (unsigned)

$2^3 = 8$ operations

Note:
→ Range of the constants are restricted by the size of the address field.

Register Addressing mode

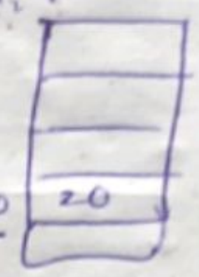
→ It is a local register value.

→ opcode | Address field → register name

EA = register name

Data (o/p of the AM) is content of the EA

Ex: register



EA = R0 = 20

Ex: mov R1, R0, source = mov R0, 20

Direct Addressing mode (static variable)

→ Another name is "Absolute Addressing mode".

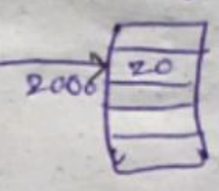
→ where the data stored in the memory

Ex: MOV R0, [2000] → EA

R0 = 20

opcode | Address → memory address

Data = [EA]
= [memory Address]
= [2000] = 20

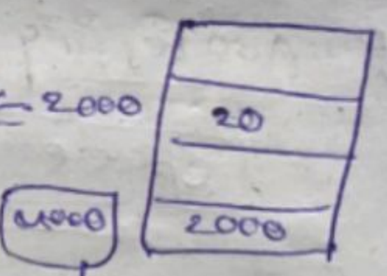


Indirect Addressing mode

→ It is like a pointers

→ There are 2 types of AM's

1) Memory Indirect AM



DEA ← 2000

IEA ← 4000

EA = [4000]

opcode | address → address of EA/variable

EA = Address of EA

Data = [EA]
= [[memory address]]
= [[4000]] = [2000] = 20

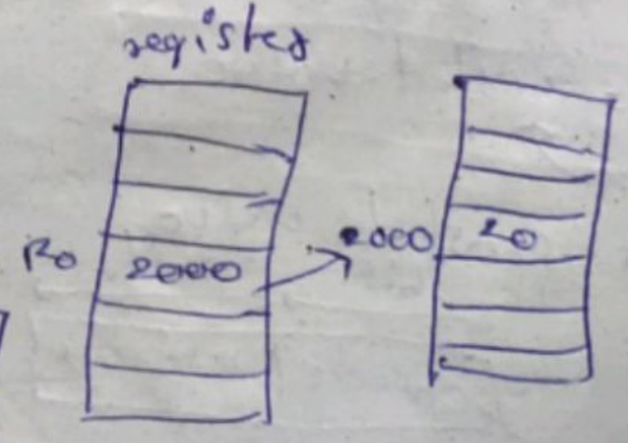
Ex: mov R0, @4000 (or)
mov R0, [[4000]]
MOV R0, (4000)

Register Indirect AM

opcode | Address field

Ex: EA = [Register Name]

Data = [EA]
= [[Register Name]]
= [[R0]] = [2000]
= 20



mov R1, #25
mov R1, R0
mov R0, [2000]
mov R0, [[2000]]
mov R0, [[R0]]

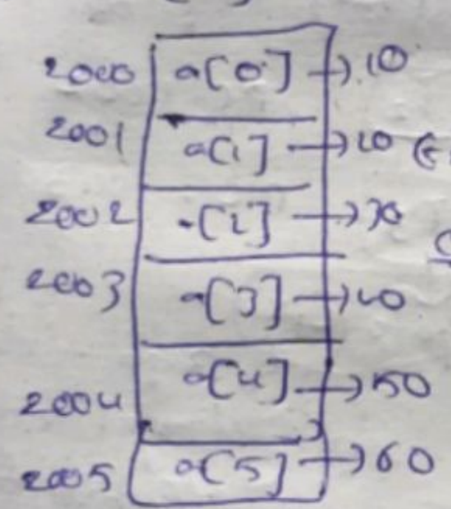
<u>Note</u>: Effective Address is computed by adding the constant to content of some register

EA = (const) + Register

1) Index AM
2) Auto increment / decrement
3) PC-relative AM
4) Base relative AM

1) <u>Index AM</u>

char a[10]



EA = Base address + index value × block size (or) step size = 1 byte

Base address → const; index value → stored in index register (R1)

Ex: a[4], const

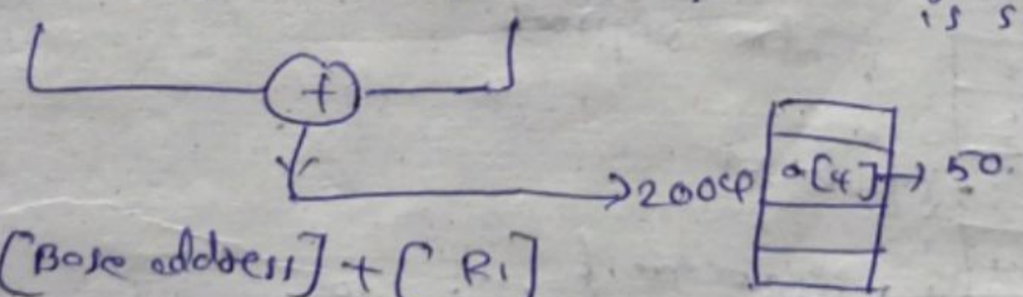
EA = 2001 + 4 × 1
= 2004

Data = [EA]
= [2004] = 50

[opcode | address]

address → starting address of the memory ← Base address

Index register (position of an array) is stored in index register



EA = address field (Base address) + [R1]
Data = [EA]

Ex: mov R1, 2000 (R0)
1st step = mov R0, #4

EA = 2000 + 4 = 2004
Data = [EA] = [2004] = 50

Ex: 8086
MOV BX, 2000
MOV SI, #4
ADD AX, [BX][SI]

Auto increment / decrement AM

EA = [ ] + [illegible]

opcode	address

address → Base address (BA)

EA = ++BA / --BA

BA++ / BA--

pre increment

mov R0, 2000
mov R1, ++(R0)

Transfer of control flow AM

→ location of the next instruction → sequential instructions

→ PC = PC + step size

execution

	memory	CPU	
PC = 1000	I_1	I_1	PC = PC+1 = 1001
PC = 1001	I_2	I_2	PC = 1002
PC = 1002	I_3	I_3	PC = 1003

1000	I_1
1001	I_2
1002	I_3 → goto L1
1003	I_4
L1 1004	I_5
1005	I_6
1006	I_7

BA
TA
EA

goto L1 → unconditional

→ Base address
→ target address
→ effective address

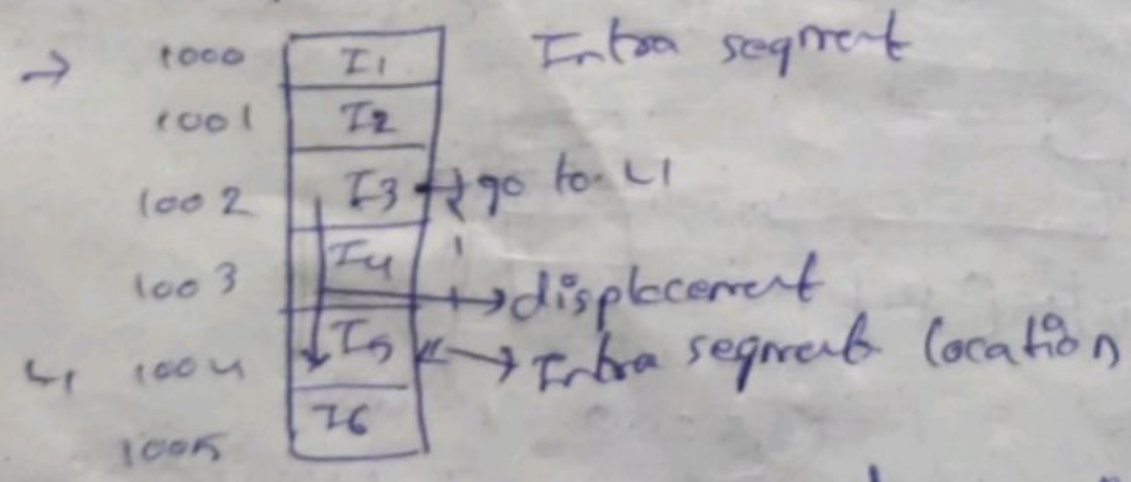
PC = ~~1003~~ 1004

1) PC-relative AM (Pc-location AM)

EA = PC + Address field value (Relative value)

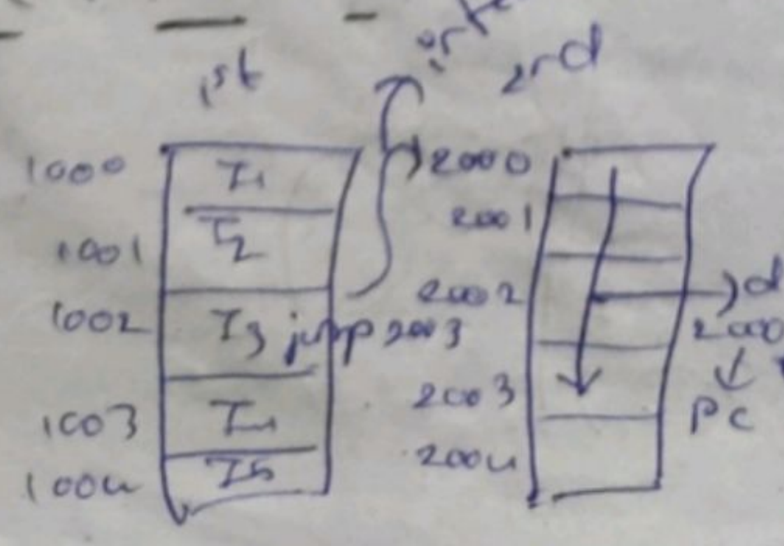
PC → current instruction

PC ← PC + Relative value



2) Base Relative AM

it is also called right position independent AM



EA = Relative value + content of the base register value

EA = [R0] + Address field

PC ← [R0] + Relative value

~~Right position independent Am~~

Q) the AM in which the location of the data is contained with inopcode is called Implied AM

Q) the AM that permits relocation without any change in the code

a) indirect b) Index c) base register d) PC-relative

a) Consider a 32-bit jump instruction placed in a memory with starting address of 1000.
a) -6 signed constant is present in the address field of Instruction that is that EA when the instruction is designed with

a) PC-relative b) base register c) (Base register = 600)
↳ EA = ? ↳ EA = ?

JUMP	-6

32-bit = 4 bytes

1) PC-relative
EA = PC + Relative
= 404 - 6
EA = 398

2) base register
EA = BR + Relative
= 600 - 6 = 594

2013
Q) which of the following AM is suitable for PGM re-location at the run time
a) Absolute AM b) Direct Am c) Relative Am d) Indirect

2014
Q) the AM's in which the operand is given explicitly in the instruction itself Immediate Am

2020
Q) move, cost, ADD are called opcodes

2021 (AP)
Q) the AM where you directly specify the operand value Immediate Am

Q) the following the register transfer indicates which AM
AC ← m[ADR + XR]
a) Direct b) relative c) indirect d) index

2021)
Q) LW (LOAD R1, 20(R1)) EA of the memory location of operand, is obtained by addition of constant 20

a) register Am b) Immediate Am c) Base index AM
d) registed indirect scale AM

Memory Orgn

performance $\propto \frac{1}{\text{Access time}}$

CPU Access time < Execution time

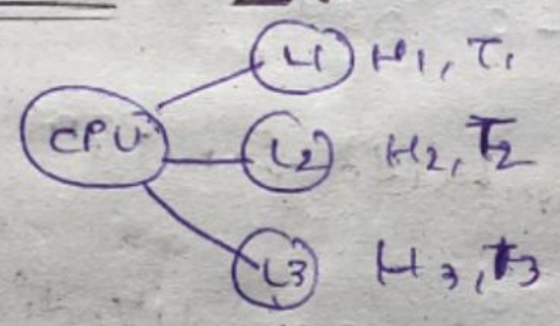
Total time = Hit/Miss + Access time + Transfer time
↳ based on the bandwith

→ the data available in memory is Hit
→ the data is not available in memory is miss
→ Hit & miss are neglegable time
→ Two types of Access.
  1) simultaneous Access
  2) Hierarchy Access
→ Hit Ratio = $\frac{\text{no. of hits}}{\text{Hits + miss}}$

Hit + miss = Total no. of CPU reference

Hit + miss = 100% → 1 success ratio
Hit = 1 − miss ; miss = 1 − Hit

1) simultaneous Access

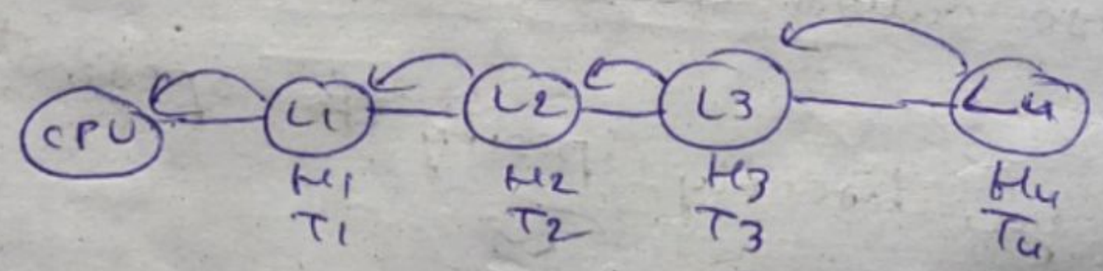


$(t_1 < t_2 < t_3 < t_4)$

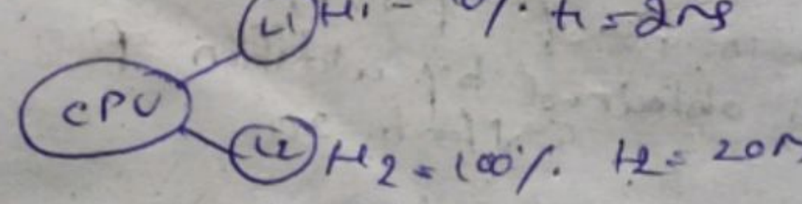
Total avg Access time = $H_1 \times t_1 + (1-H_1) H_2 \times t_2 + \underbrace{(1-H_1)}_{\text{1st miss}}\underbrace{(1-H_2)}_{\text{2nd miss}} H_3 \times t_3$

2) Hierarchy Access

→ the last level of the memory is always "Hit"



Total avg Access time → $H_1 T_1 + \underbrace{(1-H_1)}_{\text{miss 1}} H_2 (t_2 + t_1) (1-H_1)(1-H_2) H_3 (t_3 + t_2 + t_1)$
$(1-H_1)(1-H_2)(1-H_3) H_4 (t_4 + t_3 + t_2 + t_1)$



(1)

simultaneous = $H_1 t_1 + (1-H_1) T = 0.9 \times 2 + (0.1) 20$
$= 1.8 + 2 = 3.8$

Hierarchy = $H_1 t_1 + (1-H_1) H_2 (\tau_2 + t_1)$
$= 0.9 \times 2 + (1-0.9)(20+2$
$= 1.8 + (0.1)(2.2)$
$= 1.8 + 2.2 = 4.0$ ns

If code with reusability $\tau_1 = 2$ ns, $\tau_2 = 20$ ns

I_1 → data (Assume Data in L2 memory)
I_2
I_3 → data
I_4
I_5 → data
I_6
I_7 → data

} Reuse the I_1 data

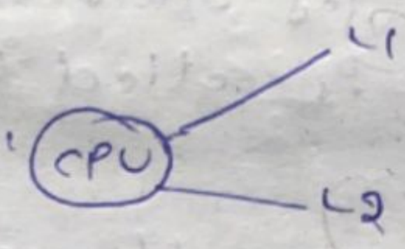
a) simultaneous

b) Hierarchy

23/2/25

$\tau_1 = 2$ ns
$\tau_2 = 20$ ns

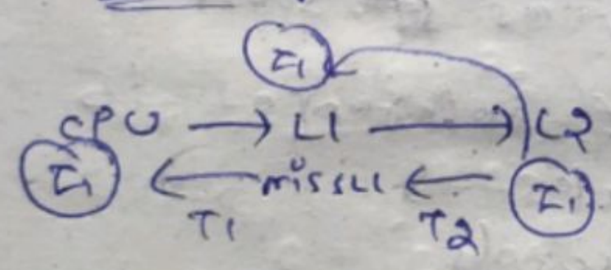
1) simultaneously :



I_1 : miss L1, Hit L2 (20ns)
I_3 : miss L1, Hit L2 (20ns)
I_5 : miss L1, Hit L2 (20ns)
I_7 : miss L1, Hit L2 (20ns)

Total time = 4 × 20 ns
= 80 ns

2) Hierarchy :



→ after the instruction/data moves from one level of memory to other it always copy & paste

I_1 : miss L1, Hit L2 ($\tau_2 + \tau_1$)
I_3 : Hit L1 (τ_1)
I_5 : Hit L1 (τ_1)
I_7 : Hit L1 (τ_1)

$= (\tau_2 + \tau_1) + (3\tau_1)$
$= 4\tau_1 + \tau_2$
$= 4 \times 2 + 20 = 28$ ns

•) $\tau_1 = 2$ ns
$\tau_2 = 20$ ns
I_1 : 1 word
I_3 : 1 word
I_5 : 1 word
I_7 : 1 word

} → L2 memory

(1) simultaneously :
I_1 → miss L1, Hit L2 (20ns)
I_3 → " , "
I_5 → " , "
I_7 → " , "
Total time = 4 × 20 = 80 ns → best case

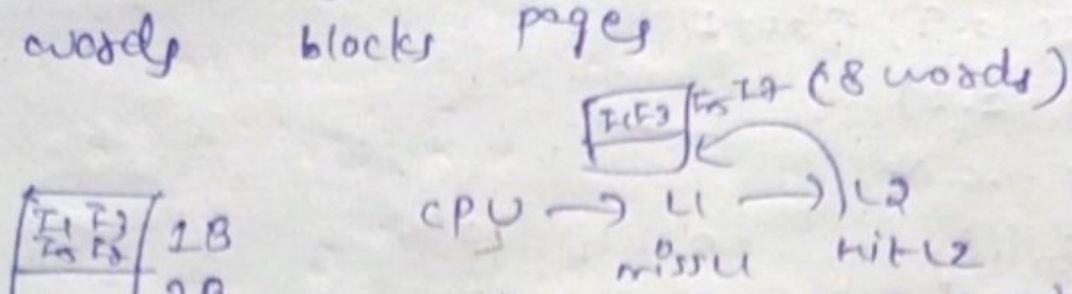
(2) Hierarchy.
I_1 → miss L1, Hit L2 ($\tau_2 + \tau_1$)
I_3 → " "
I_5 → " "
I_7 → " "

$4(\tau_2 + \tau_1) = 4(22)$
$= 88$ ns

CPU → c.m → m.m → s.m 1 block = 8 words

words blocks pages

Q)



$I_1 \rightarrow$ miss L1, Hit L2 $(\tau_2 + \tau_1)$
$I_3 \rightarrow$ Hit
$I_4 \rightarrow$ Hit
$I_2 \rightarrow$ Hit

$= (\tau_2 + \tau_1) + (3\tau_1)$
$= 28$ ns

Locality of Reference

→ the CPU has requested one address for memory access that particular address / near by address will be accessed soon is called locality of reference

→ there are 2 types

1) temporal locality (reuse the same data accessed frequently)

2) spatial locality → (According to the space)
→ near by address are references soon

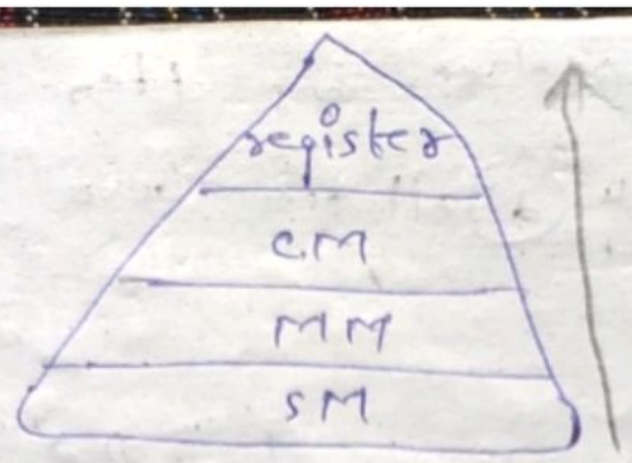
→ to satisfy the above locality of references to organise the data in memory blocks

→ A block can contain multiple words

→ Block size is decided by designer of the system

→ Hierarchy orgⁿ used in the system design

	register	cache memory	main memory	secondary memory
size	<1KB	<16MB	<16GB	>100 GB
Access time	0.25 - 0.5 ns	0.5 - 25 ns	80 - 250 ns	5,00,000 ns
Block size (Bandwith)	2000 - 100000	5000 - 10000	1000 - 500	20 - 150
managed by	compiler	Hardware	OS	OS
Backed by	cache memory	main memory	secondary memory	C.D



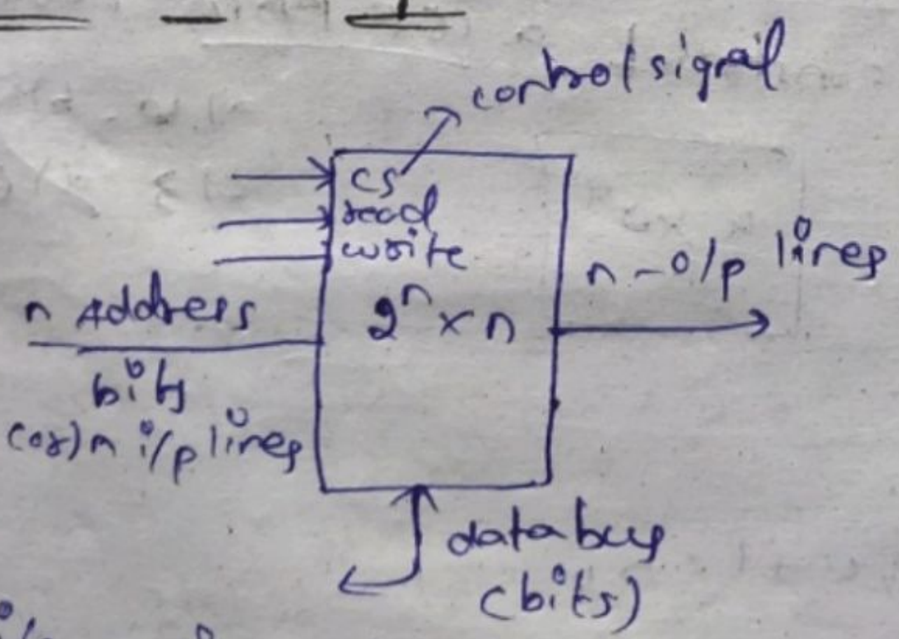
Bottom-top Approach of memory hierarchy states

1) capacity ↓
2) Access time ↓
3) performance ↑
4) cost / bit ↑

Difference b/w static & Dynamic RAM

static RAM	Dynamic RAM
→ It is designed with flip flops	→ It is designed with capacitors
→ no periodically refreshing	→ periodically refreshing required
→ It is faster	→ It is slower
→ It is costly	→ It is less expensive
→ Ex: cache memory	→ Ex: main memory
→ Idle (power consumption is very less)	→ Idle (power consumption is more)
→ execution power consumption is high	→ execution power consumption is low

RAM block Diagram

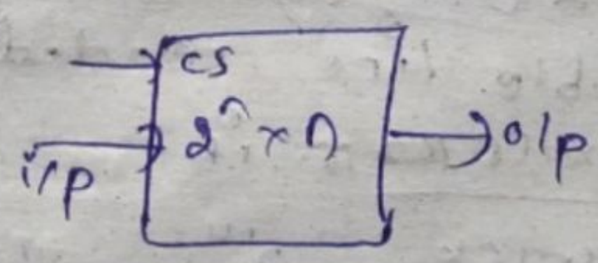


a) i/p = 7 lines o/p = 8 lines
what is RAM structure

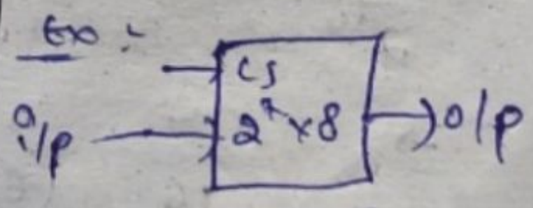
$\boxed{2^7 \times 8}$ Total no. of pins =

Total = 18 pins ← { i/p = 7 pins, o/p = 8 pins, CS, read, write = 3 pins }

ROM block diagram



Ex:



Total pins:
i/p = 8
o/p = 7
CS = 1
16 pins

Q) memory has a 14-bit address bus then how many memory locations

a) 16K b) 16384 c) 2^{14} (d) all the above

14-bit address b/w

↓

i/p lines = 2^{14}

1st → $2^{14} = 2^4 \times 2^{10}$

$= 2^4 \times 1K = 16K$

2nd → $2^{14} = 2^4 \times 2^{10}$

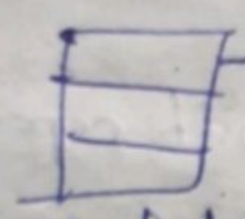
$= 2^4 \times 1K$

$= 2^4 \times 1024$

$= 16 \times 1024$

3 → 2^{14}

Q) 4096 bytes memory the block size is 2 bytes memory cycle time is 20ns then to find out the time to read the 1024 bytes



→ 2 bytes

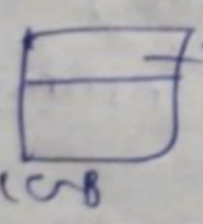
1024 bytes

blocks = $\frac{1024}{2}$ = 512

Total time = $512 \times 20\,ns = 10240\,ns$

Q) A processor can support the max memory size 4GB, where the memory is word addressable (1W = 2 Bytes). The size of the address bus of the processor is ____ bits

$\frac{4GB}{1W} = \frac{4GB}{2B} = \frac{2^2 \times 2^{30}\,B}{2B} =$



→ 1 word

1GB

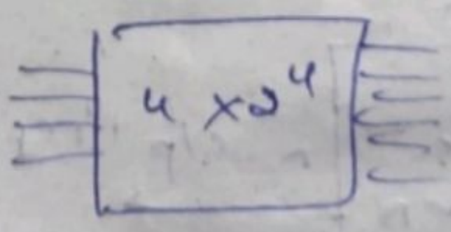
$1GB = 2^{30}$ bytes. $= 2^1 \times 2^{30} = 2^{31}$

Q) A RAM chip has capacity of 1024 words of 16 bits each (1K×16) the i/p 2×4 decoders with enable line needed to construct a 16K×16 RAM from 1K×16 RAM is

a) 4 b) 5

c) 3 d) 6

$\frac{16K \times 16K}{1K \times 16} = \frac{2^{14}}{2^{10}} = 2^4 = 16$



2×4 → 4

by using $16 \times 16 \rightarrow \frac{16}{4} + \frac{16}{4} + 1 = 5$

Q) consider a memory size of 2K x 8 bits what is the size of decoder needed to access the cell of the memory uniquely

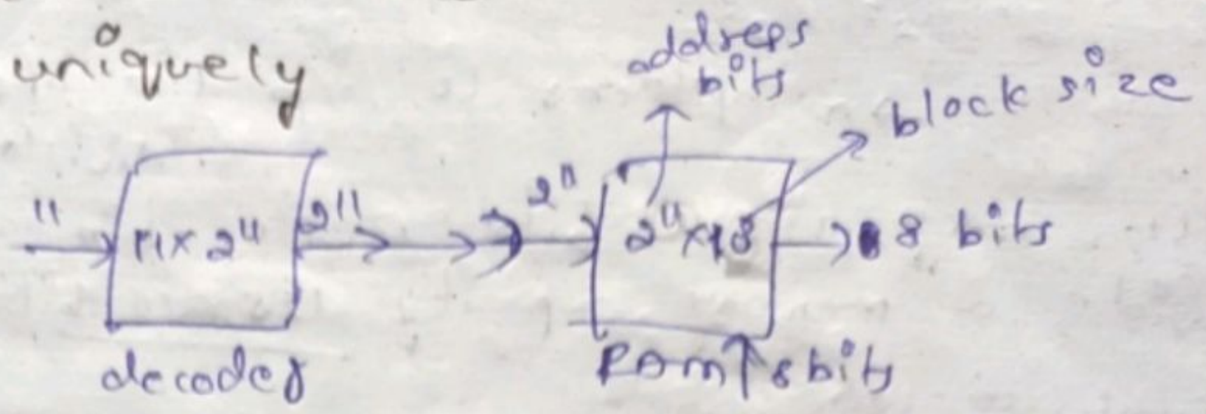
memory size $= (2K \times 8)$ bits

$1K = 2^{10}$

$2^1 \times 2^{10} \times 8$

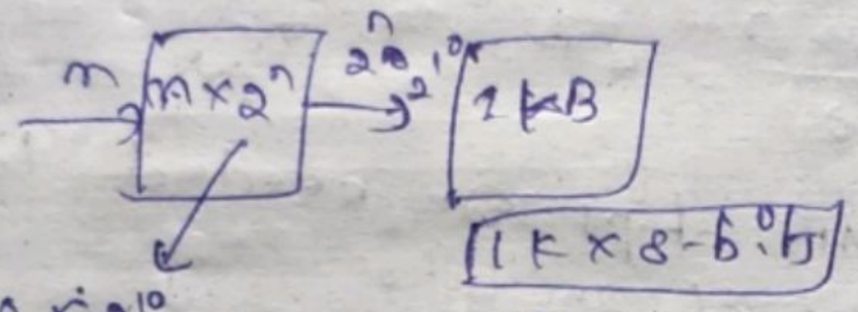
$2^{11} \times 8$

address bits



size of decoder = 11×2^{11}

Q) if there are m i/p lines, n o/p lines for decoder, that is used uniquely address a Byte addressable 1KB then the min value of $m \times n$:



10×2^{10} $\quad \therefore 1K = 2^{10}$

$m = 10,\ n = 2^{10}$

$m + n \rightarrow 10 + 1024 \rightarrow 1034$

Q) the amount of ROM need to implement a 4-bit multiplier is

a) 2K bits
b) 4K bits
c) 8K bits
d) 2K bits

4-bit = 2x4 = 8 bits

$2^8 \times 8 \rightarrow 2^4 \times 8 \rightarrow 2^4$

$= 2KB$

Q) Consider a 4-bits unsigned values A & B what will be the max size of result for

addition of A & B

ex: 4-bit-A : - - - - 15
4-bit-B : - - - - 15

A+B 11110 30

multiplication of A & B

4-bit A → n bit
4-bit B → n bit
} $n \times n = 2n$

A × B = n 15 → 4-bit
n 15 → 4-bit

225 $\quad 2 \times 4 = 8$ bits

$= 255 = 11111111$

8 bits

a) 8×1 multiplexer using 2×1

$\frac{8}{2^1} + \frac{4}{2^2} + \frac{2}{2^3} = 4+2+1 = 7$ 2×1 multiplexers are required

→ multiplexer to i/p line bfo divide chayali

→ a) How many 128×8 bits ROM chip are needed to provide a memory capacity 128×16 bits

Total size = 128×16 bits

using 128×8

so $\frac{128 \times 16}{128 \times 8} = 2$ (no. of ROM chips)

a) How many 128×8 bits RAM chip are need a memory 256×16

Total = 256×16 using 128×8

$\frac{256 \times 16}{128 \times 8} = 4$

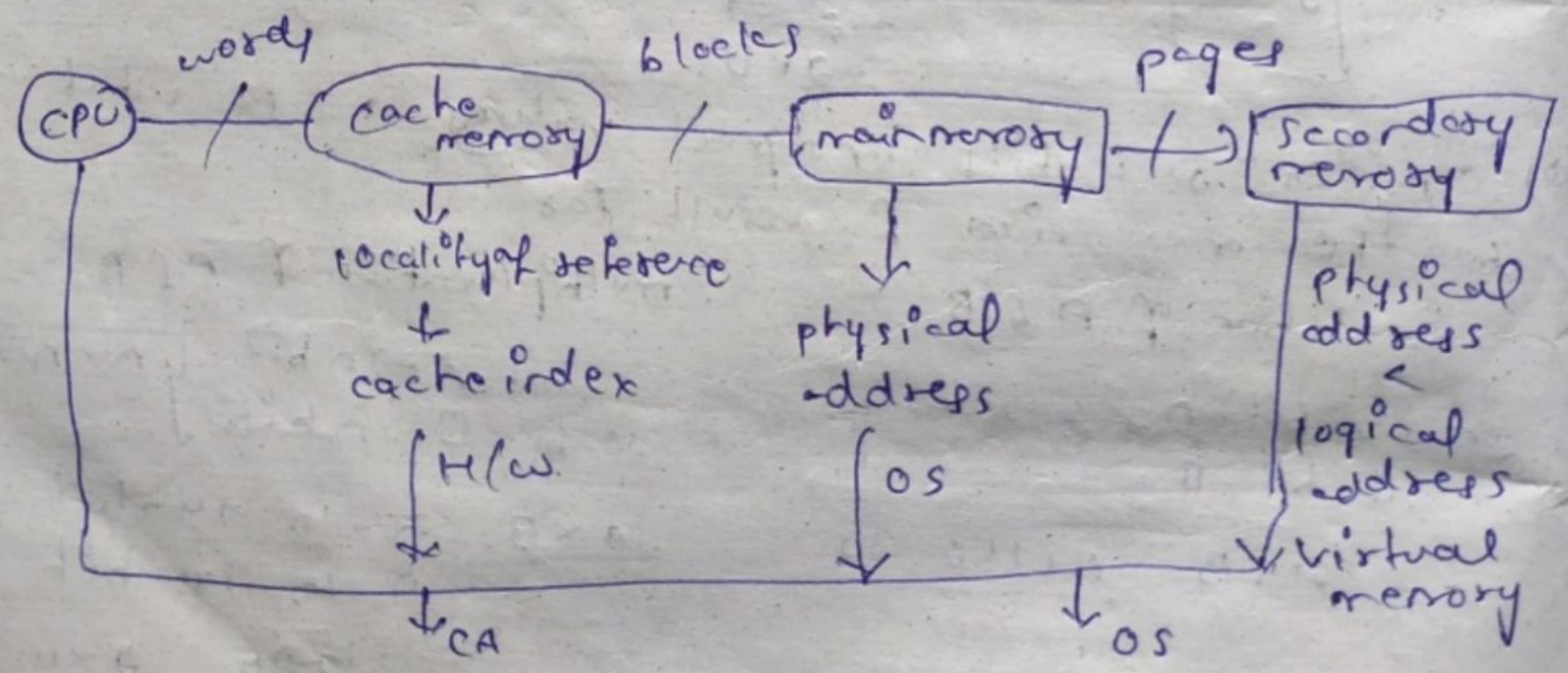
a) How many 32K × 1 bit RAM chips are needed to provide a memory capacity of 256 bytes

→ 256B → 256K × 8 bits

no. of RAM chips = $\frac{\text{total size}}{32K \times 1 \text{ bit}} = \frac{256K \times 8\text{-bits}}{32K \times 1 \text{ bit}}$

$\Rightarrow \frac{2^8 \times 2^{10} \times 8\text{-bit}}{2^5 \times 2^{10} \times 1\text{-bit}} = \frac{2^8}{2^5} \times \frac{8}{1} \rightarrow 2^3 \times 8 = 64$

Cache Memory



→ speed is imbalance b/w the CPU & main memory. To overcome this problem the cache memory is introduced

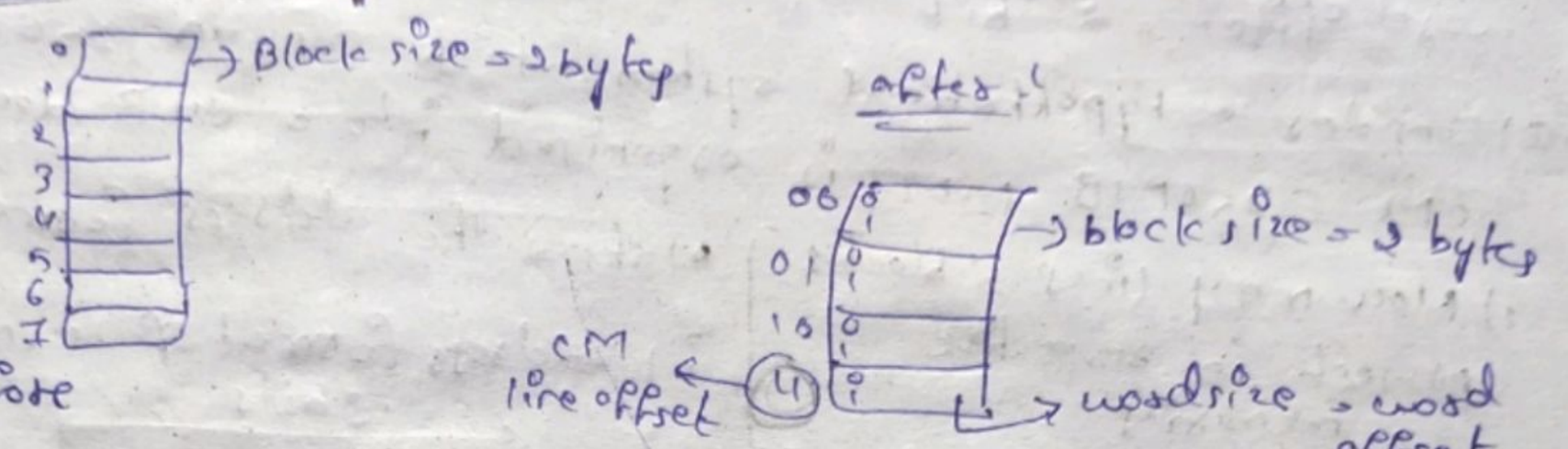
→ program size is larger than the main memory. To overcome this problem, by using virtual memory

Memory org

According to the memory hierarchy design data will be transfered from main memory to cache memory in a form of blocks. so, both of the memories organised into blocks. based on the block size

eg cache memory size = 8B main memory = 16B block size = 2B

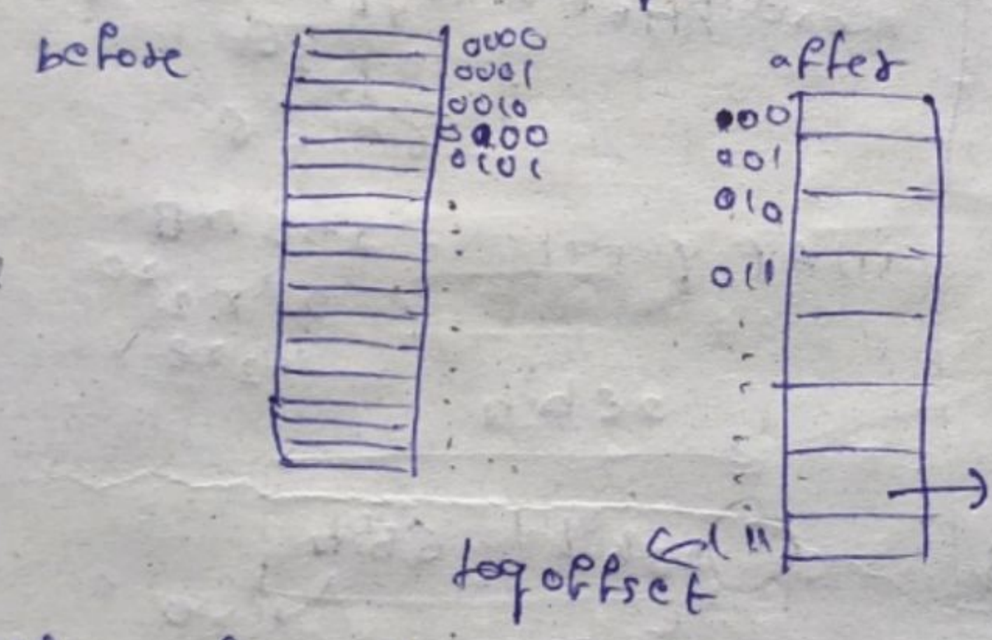
cache memory orgn



no. of cache memory blocks = $\frac{\text{total c.m. size}}{\text{block size}} = \frac{8}{2} = 4$ blocks

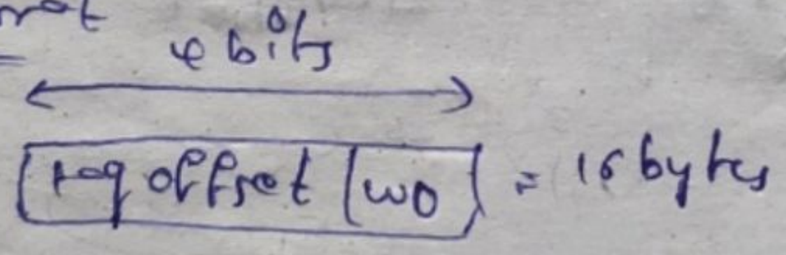
Main Memory orgn

M.M size is : 16 byte



no. of M.M blocks [no. of tags] = $\frac{\text{Total size of MM}}{\text{block size}} = \frac{16B}{2B} = 8$

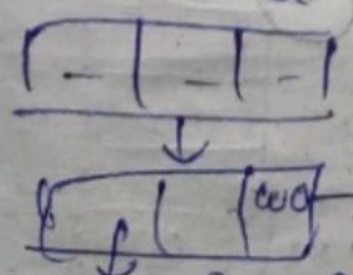
Address format



tag offset = $\log_2$(no. of tags)
$= \log_2(8)$
$= \log_2 2^3$
$= 3\log_2 2$
$= 3 \times 1 = 3$-bit

Address format of c.m

c.m line offset = $\log_2(4)$
$= \log_2 2^2$
$= 2$-bit



word offset = $\log_2$(Block size)
$= \log_2(2)$
$= 1$-bit

Q) CM = 16B, MM = 32B & block size = 4B

no. of CM blocks = $\frac{16B}{2B}$ = 4

line offset = $\log_2 4$
= $\log_2 2^2$
= $2\log_2 2 = 2\times1$
= 2 bits

word offset = 2-bit

no. of MM blocks = $\frac{32}{4}$ = 8

tag offset = $\log_2 8 = \log_2 2^3 = 3\times1$
= 3 - bits

word offset = $\log_2(4)$
= $\log_2 2^2 = 2\times1 = 2$ bit

Q) Consider a hypothetical system which contain 256KB CM & 256MB MM is organised into a 64B blocks

i) How many lines & blocks present in respective memories

CM size = 256KB
MM size = 256MB
Block size = 64B

→ no. of CM blocks = $\frac{256KB}{64B}$
= $\frac{2^8\times2^{10}B}{2^6\times B}$
= $2^2\times2^{10}$
= 2^{12}

→ no. of MM blocks = $\frac{256MB}{64B}$
= $\frac{2^8\times2^{20}B}{2^6\times B}$
= 2^{22}

a) show the address format

i) Address format of C.M = 256KB
= $2^8\times2^{10}$
= 2^{18}

18 bits

12 bits	WO 6 bits

line offset, word offset

block size = 64
word offset = $\log_2(64)$
= $\log_2 2^6$
= $6\times1 = 6$

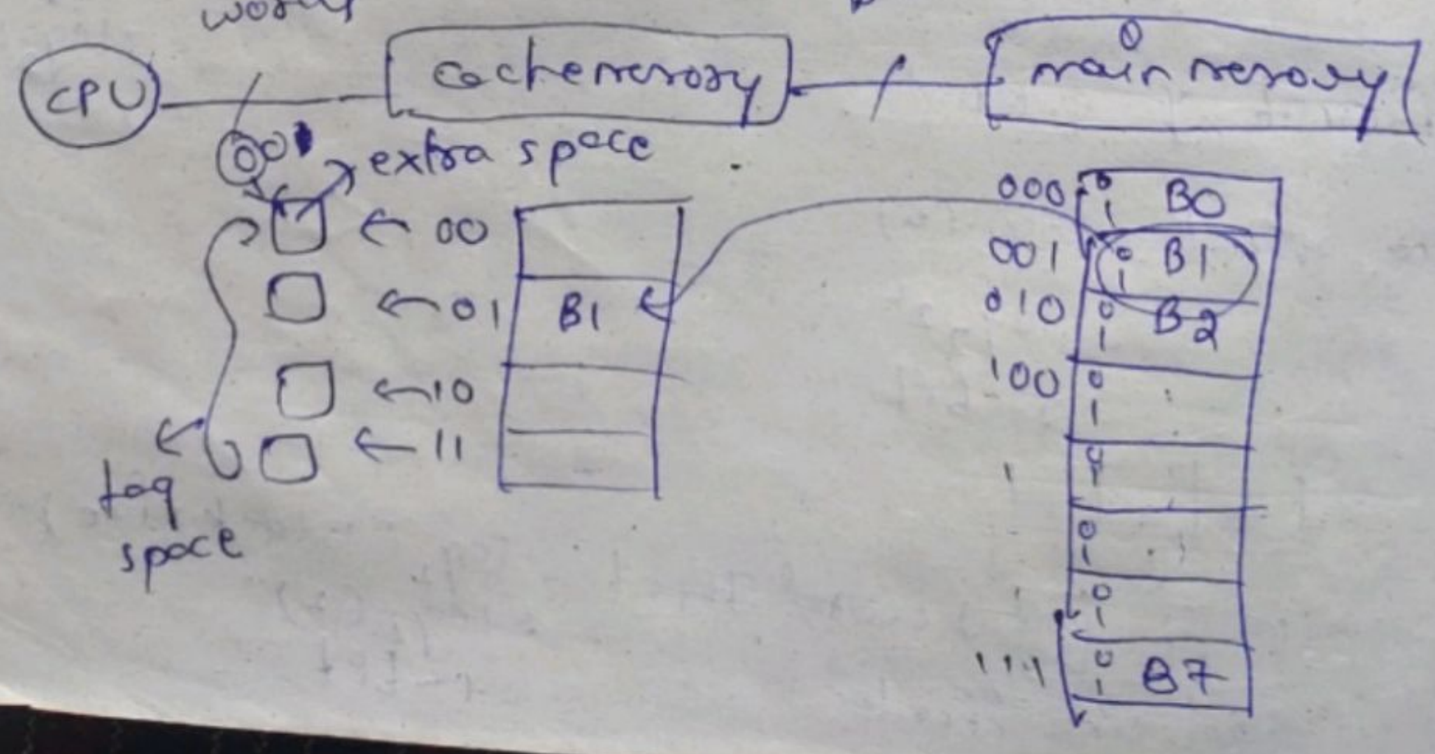
ii) Af of MM = 256 MB
= $2^8\times2^{20}$
= 2^{28}

28 bits

22 bit	6 bits

tag offset, WO

Mapping Techniques



→ The process of transferring the data from MM to CM is called mapping

→ During the mapping process data plan is transferred to CM along with physical address

→ In this process some extra space is maintained in the cache line used to accomodate the physical address info is called tag space / tag directory

Types of mapping technique

1) Direct mapping
2) Associative (full) Associative mapping
3) Set - Associative mapping

Direct Mapping

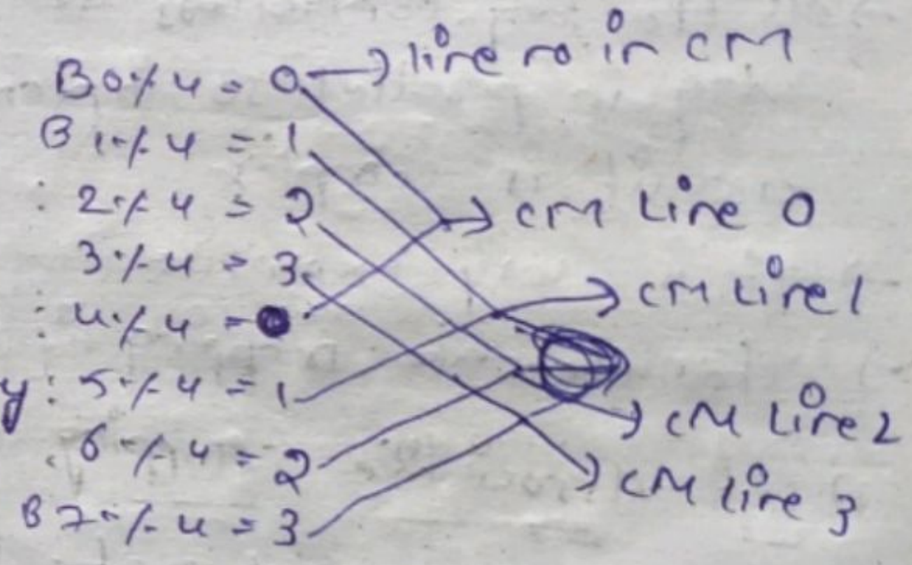
→ M MOD N = i

$\boxed{M \% N = i}$

M → main memory block no

N → no. of cache lines in CM

i = line number in cache memory



MM Address format

16 bit — 16 bytes

3 bit	4 bit
tag offset	WO

B3 = 011 → tag no (MM block no)

CM Address format

1-bit	2-bit	1 bit
tag offset	CM line offset	WO

B3%4 = 3 = 011 → tag offset, line offset

tag space

limitations

→ The direct mapping cache line is capable to holding only one block at a time. so no. of conflict miss (page fault) is increasing

→ When the CPU frequently refer the multiple blocks which are mapped into same cache lines then the cache blocks are continuously mapping so hit ratio will be following down. This phenomena is known as thrashing

2) Associative / Full Associative Mapping

→ A block in the MM is mapped to cache memory anywhere in the cache memory

→ If the CM is full then use the FIFO, LRU algorithms

→ By using B2 is replaced by B0 or LRU B2 is replaced by B3

MM Address format

4-bit	1 bit
3-bit	w0

CM Address format

B3 = 011

← 3 bit →	← 1 bit →
tag offset	w0

tag

Advantages

Fastest cache mapping

Disadvantages

→ All the tag controllers are compared with CPU generated tag in parallel so implementation of the comparision circuit is complex

→ It is very expensive

→ The alternate mapping is a set-Associative mapping

3) Set-Associative Mapping

no. of sets = $\frac{\text{no. of cache lines}}{2^1 2^1 2^2 2^3 \ldots}$ ← P-way set

S_0	00	01
S_1	10	11

CM = 8B, MM = 16B, Bsize = 2B

no. of cache lines = $\frac{8}{2} = 4$

let assume 2 way set

no. of sets = $\frac{4}{2} = 2$

4-way set

no. of set = $\frac{4}{4} = 1$

B a % s = i

M = MM Block no

S = no. of sets in CM

P = CM set no

Address format of MM

3 bit	1 bit
tag offset	w0

B0 % 2 = 0
1 % 2 = 1
2 % 2 = 0
3 % 2 = 1
4 % 2 = 0
5 % 2 = 1
6 % 2 = 0
7 % 2 = 1

→ set 0
→ set 1

For CM8 Address format

2 bit	1 bit	1 bit
tag offset		w0

↓ set offset

set offset = $\log_2$ (no. of sets)
= $\log_2(2)$
= 1 bit

Q) A computer has 128 kB, 8 way set Associative write back data cache with block size 32 byte. The process sends 32-bit Address to the cache controller, the no. of bits in tag field of an address is a) 18 b) 20 c) 14 d) 16

CM = 128 kB, MM = 2^{32} = 4 GB, Block size = 32 B

no. of cache lines → $\frac{128\,kB}{32\,B} = \frac{2^7 \times 2^{10}\,B}{2^5\,B} = 2^{12-5} = 2^{12}$

no. of sets = $\frac{2^{12}}{2^3} = 2^{12-3} = 2^9$

set offset = $\log_2(2^9) \rightarrow$ 9-bits

word offset = $\log_2(32) = \log_2(2^5) \rightarrow$ 5-bits

32-bit

18 bit	9 bit	5 bit
tag offset	set offset	wo

tag offset = 18

[32 − (9+5)]

26/2/25

Q) Consider 64 kB full Associative cache organised into a 32 byte block of MM 16 MB blocks. What is the size of tag MM & CM controller

CM size = 64 kB ⟶ CM = 64 kB → $2^6 \times 2^{10} = 2^{16}$ → 16-bits

block size = 32 B ⟶ 32 B → $\log_2(32)$ → 5-bits

MM size = 16 MB ⟶ 16 MB → $2^4 \times 2^{20} \rightarrow 2^{24}$ → 24 bits

size of tag in MM & CM = ?

MM :

19 bit	5 bit
tag offset	wo

tag offset = 24 − 5 = 19 bits

CM =

11 bit	5 bit
tag offset	wo

tag offset = 11 bits

Q) Assume direct mapped cache having 8 cache lines; each cache line consist of 2 words & each word is 1 Byte the address bus consist of 7-bit then the tag field of cache line?

→ 2 words → 2 bytes

1 word = 1 byte

7-bit → Address bus

3 bit	3 bit	1 bit
tag offset	line offset	wo

→ CM

wo = $\log_2(2) \rightarrow$ 1 bit

line offset = $\log_2(8) \rightarrow$ 3-bit

tag offset = 7 − (1+3)

= 7 − 4 = 3-bit

Q) Assume a 32-bit addressable physical memory & 2MB cache block of size 8KB each then how many bits are required in the tag field of the cache address, if the cache is 4-way set associative

main memory size $= 2^{32} = 2^2 \times 2^{30} \Rightarrow 4GB$

~~[illegible]~~ cm block size $= 8K$

size of cm $= 2MB$

no. of cm lines $= \frac{\text{cm size}}{\text{block size}} = \frac{2MB}{8KB} = \frac{2^1 \times 2^{20}}{2^3 \times 2^{10}} = \frac{2^{21}}{2^{13}} = 2^{21-13} = 2^8$

no. of sets $= \frac{\text{cm lines}}{\text{4-way set}} = \frac{2^8}{2^2} = 2^{8-2} = 2^6$

set offset $= \log_2(\text{no. of sets}) = \log_2(2^6) = 6$-bits

word offset $= \log_2(\text{block size}) = \log_2(\overset{8KB}{\ } \Rightarrow 2^3 \times 2^{10})$

$= \log_2(2^{13}) \Rightarrow 13$-bits

tag offset $= 32 - (6+13)$

$= 32 - 19 \Rightarrow 13$-bits

32-bit

tag	set	set offset	wo
13bit		6bits	13bit

Q) suppose a cache has 70% hit ratio or access time of 50 ns or a ~~cache~~ MM hit & an access time of 120 ns on cache miss then what is the effective access time?

Hit $= 70\% = \frac{70}{100} = 0.7\ (H_1)$ Cache Access time $= 50ns\ (T_1)$

miss $= 1 - \text{hit} = 1 - 0.7 = 0.3$ Main memory Access $= 120ns\ (T_2)$

$H_2 = 100\% = 1$

i) Hierarchical Access

Avg Access time $= H_1T_1 + (1-H_1)(T_1+T_2)H_2$

$= 0.7 \times 50 + (0.3) \times (50 + 120) \times 1$

$= 7 \times 5 + 0.3(170)$

$= \cancel{35 + [illegible]}$

$= 35 + 3 \times 17 =$

$= 35 + 51 = 86\ ns$

ii) simultaneous Access

$= H_1T_1 + (1-H_1)\ T_2 \cdot H_2$

$= 0.7 \times 50 + (0.3)\ 120 \times 1$

$= 7 \times 5 + 3 \times 12 \times 1$

$= 35 + 36 = 71\ ns$

Associative Memory

→ It is also known as content Addressable memory

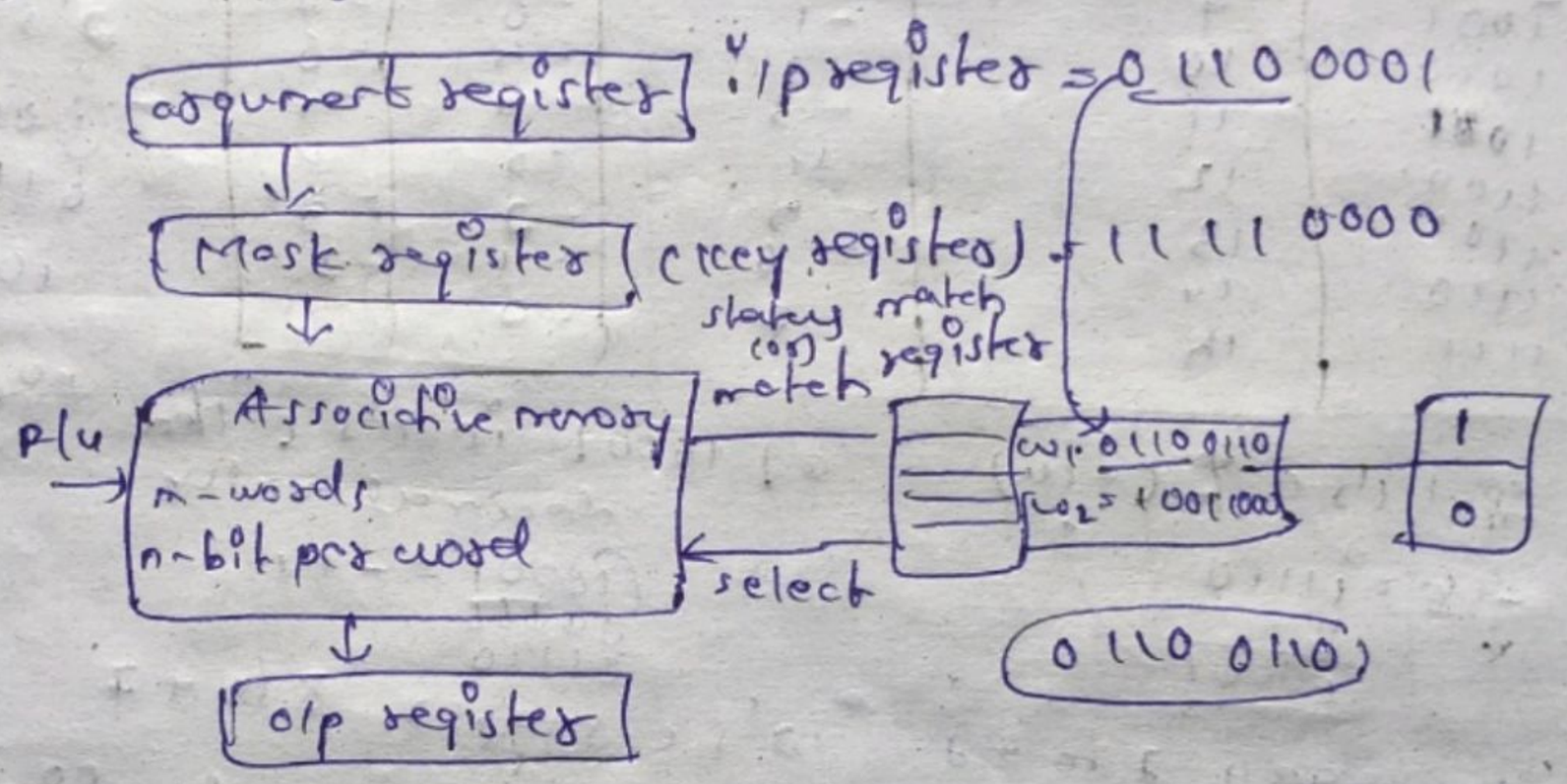
→ Differe~~te~~ from other memories such RAM, ROM, Disk etc

→ In the nature of the accessing the memory & reading the content

→ The address is given as an i/p from reading the content ~~[illegible]~~ of the location

→ In the case of Associative memory usually a part of the content / entire content is used for accessing the content. Given an argument as a i/p, the cache memory can search for whether it has this argument in any location, & if so post it. The associative memory searches all the locations simultaneously in parallel. It is useful as a cache memory where in search of the location is needed to match with a given argument (TAG)

Ex: argument register : 01100001
key register :- 11110000



Data Representation

- Data Representation
  - Fixed point Representation
    - Unsigned data Rep^n
      - → n = 4
      - range = 0 - $2^n - 1$
      - Ex: n = 4
      - $2^4 - 1 = 15$
      - i.e., range = 0 - 15
    - signed data Repres^n
      - signed magnitude
      - 1's complement
      - range for n-bit = $[-(2^{n-1}-1)$ to $2^{n-1}-1]$
      - For 2's complement = $[-(2^{n-1})$ to $2^{n-1}-1]$
  - Floating point Representation
    - single precision (32-bit)
      - Sign (1) | Exponent (8) | Mantissa (23)
    - Double precision (64-bit)
      - S (1) | E (11) | M (52)

Note: MSB = 0, +ve value, there is no need to do 1's & 2's c.
MSB = 1, -ve value, there is need to do 1's & 2's c

4-bit A B C D	unsigned	signed	1's	2's
0000	0	+0	+0	+0
0001	1	+1	+1	+1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	8	-0	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1

1's c
1000 = -7
0111

1001 = -6
0110

2's c
1000 = -8
0111
1
1000

Q) find 1's c of (-14)

-14 = 11110
↓
10001

Q) 10001 in 1's c then find out decimal value

10001 = -14
↓
01110

• 1's c range → -7 to +7 • 2's c range → -8 to +7

Q) find 2's c of (-14)

-14 = 11110
↓
10001
+1
10010

Q) 10001 in 2's then find decimal value

-14 = 10001
↓
01110
+1
01111 → 15

-15

Advantages of 2's complement
1) Each digit is a weighted digit
2) Addition & subtraction adder & subtractor one circuit is enough

Ex: -14 → 10001 → 1's c

$= -1\times2^4 + 0\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0$
$= -16 + 1 = -15$

-14 = 10001 → 2's c

$= 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
$= -16 + 0 + 0 + 0 + 1$
$= -16 + 1$
$= -15 + 1$
$= -14$

Sign extension method

1) To find 2's c of +8 in 16-bit

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

2) Find 2's c of (-7) in 8-bit

1 1 1 1 1 0 0 1 = -7 ⇒ $-2^7 = -128+64 = -64+32 = -32+16$

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ $= -16+8 = -8+1 = -7$

3) 1 1 1 1 1 1 1 1 1 1 1 1 1

2^{12}

$= \boxed{-2^{12}+2^4} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -1$

$-2^{12}+2^4$

-2^8+2^{10}

$-2^{10} = -1024$

$-1024+512$

$-512+2^8$

$-256+2^7$

$-128+2^6$

$-64+32$

$-32+16$

$-16+8$

$-8+4$

$-4+2$

$-2+1 = -1$

4) (1111 1010) in 2's

-2^8

$-8+2 = -6$

5) to find 2's c.r in 7 bits

1 1 0 1 0 1 1

- Unsigned Addition

4-bit → 15+15

15 = 1111

15 = 1111

30 1110 + 14 = [0 to 15]

(4-bit) + (4-bit) = 4-bit

↓

[0 to 2^4-1]

[0-15] = [0 to 15]

= 5 bit

= [0 to 2^5-1]

= [0 to 31]

= [16/17/18/19/20-31]

overflow → carry flag

0 [2^n-1]

carry flag = "is there an extra bit out of MSB"

(or)

"is borrow required into the MSB"

n+n = [n+1] bit

↓

extra bit

→ 2n flip flop

→ carry flag

→ PSW (program status word) (or) conditional flag register

PSW
- ACC — it stores the result of ALU
- flag register (it holds the system supported flags)

Ex: 5 = 0101
−10 = 1010
−5 = 1011

PSW: 1011, CF = 1

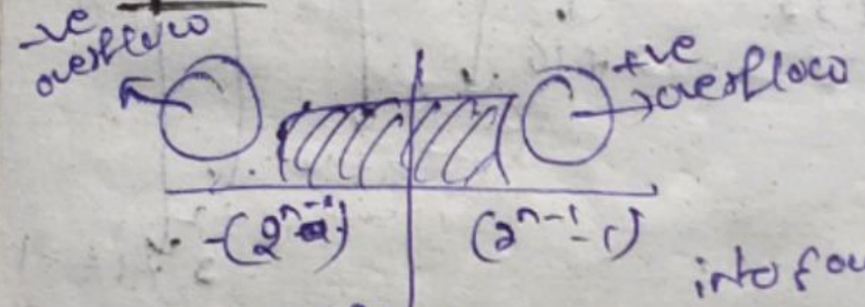
Ans = 11011

Ex: 6 → 0110
+7 → 0111
13 → 1101

PSW: 1101, flag register (CF = 0)

Ans = 01101 → 8+4+1 = 13

signed data



overflow

Ex) +7 = 0111
+7 = 0111
+14 = 1110

Ex) −6 = 1010
+2 = 0010
−4 = 1100

into & out of MSB = false

Ex) +6 = 0110
−2 = 1110
4 = 0100

out of MSB = T
into MSB = F

→ nsu
2's range = $-(2^{n-1})$ to $(2^{n-1}-1)$
= −8 to +7

−8 = 1000
−8 = 1000 overflow
−16 = 0000

16/3/25

+1.0
−1.0 } → IEEE notation, 754 (32 bit)

$\pm(1.0)_2 \times 2^{e} \rightarrow AE$

sign bit, mantissa

Actual exponent (AE) = 0
Base exponent = AE + Bias
= 0 + 127
BE = 127

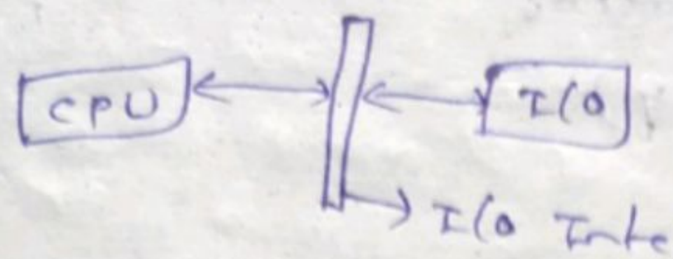
+1.0 | 0 | 01111111 | 0000 ... 23 bits

−1.0 | 1 | 01111111 | 0000 ... 23 bits

| 0/1 | 11111111 | M |
not a number

I/O orgⁿ

1) Can cpu directly connected to I/O devices ? No



Need for the I/O Interface

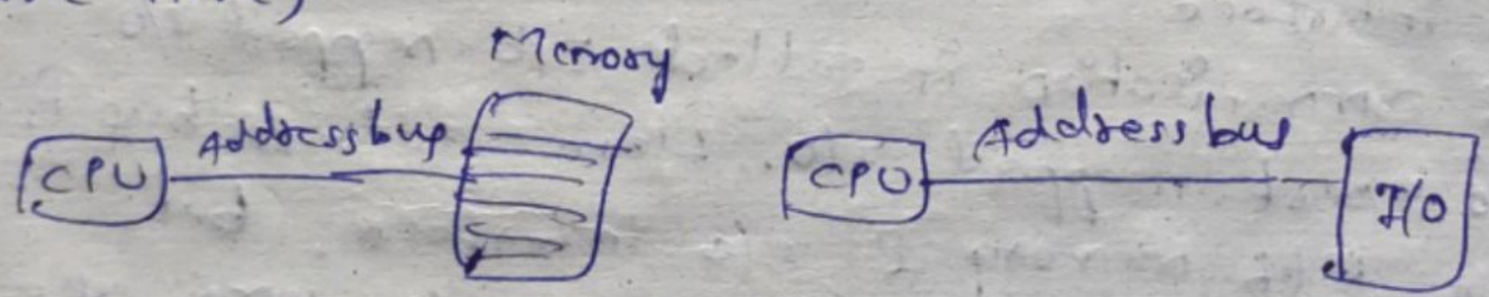
→ cpu is a electronic device where as I/O devices are electromagnetic devices

→ nature of the operation is different from the operation of cpu & memory. therefore conversions of signal values may be required

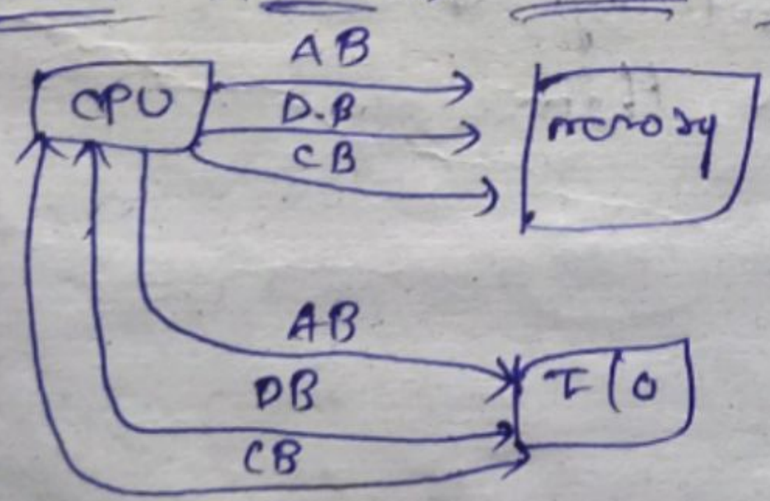
→ synchronize mechanism may be needed (Data transfer rate of peripherals lower than the cpu)

→ Data format conversion is required bcoz data course of I/O devices are different the word format in the cpu

→ operations of I/O devices shouldn't be overlapped (two or more devices can't perform operation at the same time)



(i) separate Address, control, Data bus



→ each I/O devices has an address. cpu select one I/O device by sending its I/O address through address bus

→ separate buses for the memory & I/O

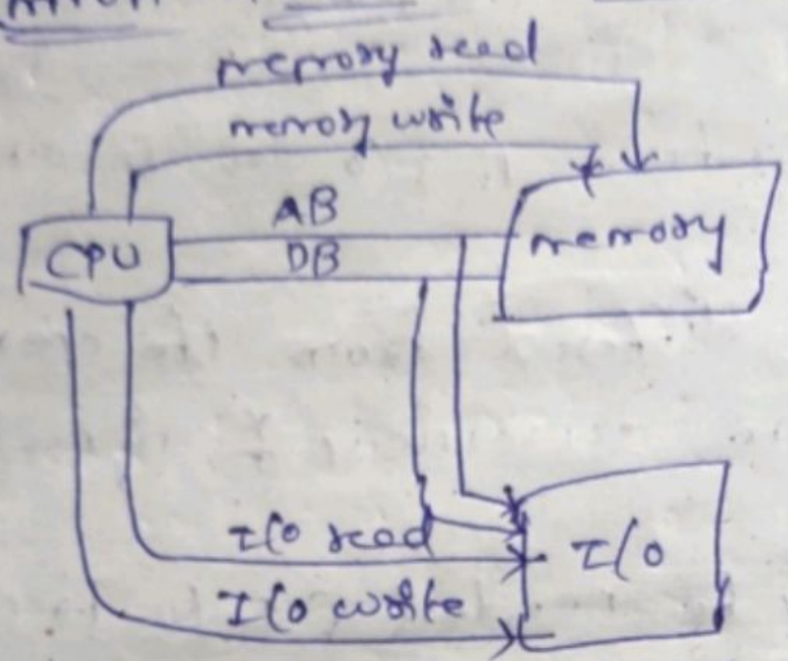
→ If the cpu want to access memory then the cpu access memory only not I/O.

2) ~~Common Address, Data, control Bus~~

→ separate connection for both memory & I/O. so, it is costly

→ CPU can perform operations either with memory or I/O but not at a time (parallely)

2) common address, Data Bus



→ CPU can communicate with memory & I/O by generating the address

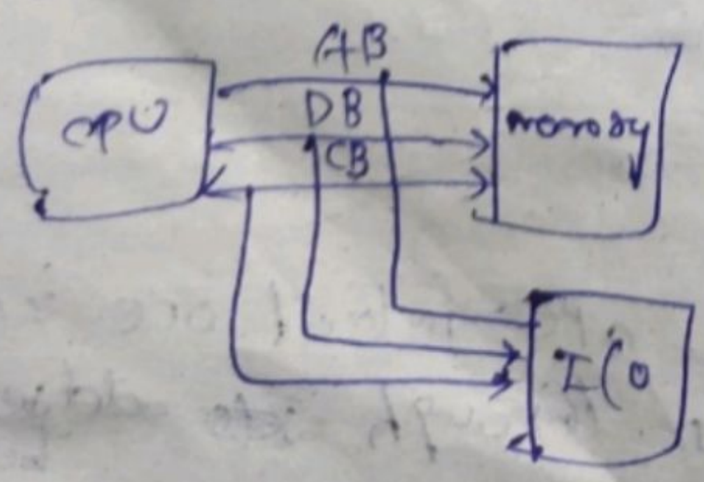
→ If the CPU want to access the memory, then the control signal of I/O will be disabled then the CPU can perform read & write operations in memory

→ CPU can distinguish b/w memory & I/O by control signals

→ CPU can either read/write operation at a time in memory/I/O interface

→ this type of communication is called I/O mapped I/O Isolated I/O, port mapped I/O. Here Address bus is common for both memory & I/O interface, but control signals are different, Address can be same as we have common bus & control signals are the one to make the difference b/w them.

3) All Common buses



→ Here, CPU uses common buses for both memory & IO interface

→ CPU can communicate with memory by generating the address, some addresses of memory are assigned for I/O devices

→ I/O devices doesn't know their own addresses, they use memory address.

→ this type of communications is called memory mapped IO

→ this is difficult to implement

→ this is cheap compared to mapped IO

difference b/w Isolated IO & memory mapped IO

memory mapped IO	Isolated IO
→ IO device doesn't have their own address space	→ IO device does have their own address space.
→ some memory wastage	→ no memory wastage
→ All the memory access instructions & addressing modes are used for I/O devices also	→ separate instruction for the IO devices & memory
→ more instruction for the I/O access	→ less instruction for the I/O access
→ more addressing for the I/O access	→ less addressing for the I/O access
→ ~~less~~ more no. of I/O devices	→ ~~more~~ less no. of I/O devices

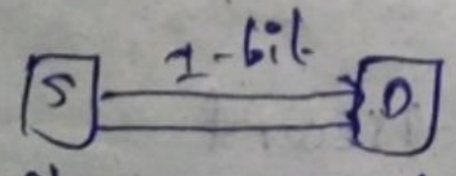
Data transfer modes

1) programmed I/O
2) Interrupt Driven I/O } I/O → CPU → memory
3) DMA → I/O → memory

asynchrous Data transfer → same clk pulse is applied

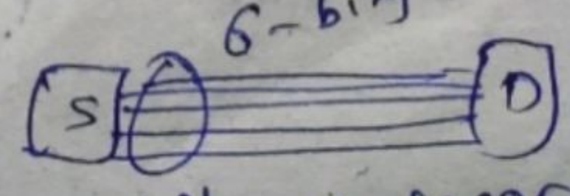
Asynchrous Data transfer { serial, parallel

serial

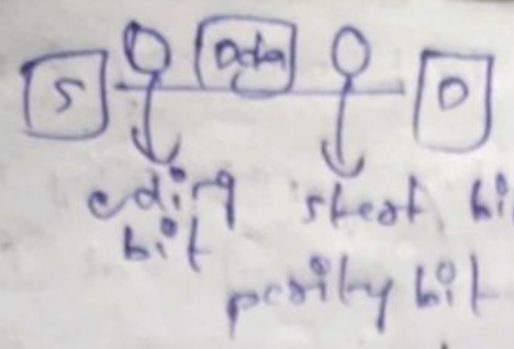


n-bit → n seconds

parallel

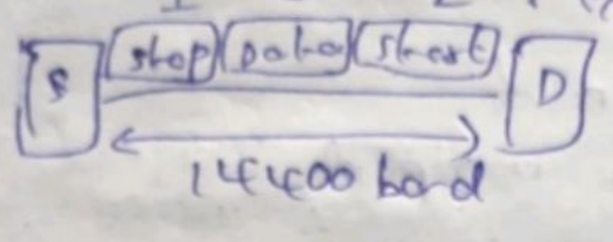


n-bit = 1 second



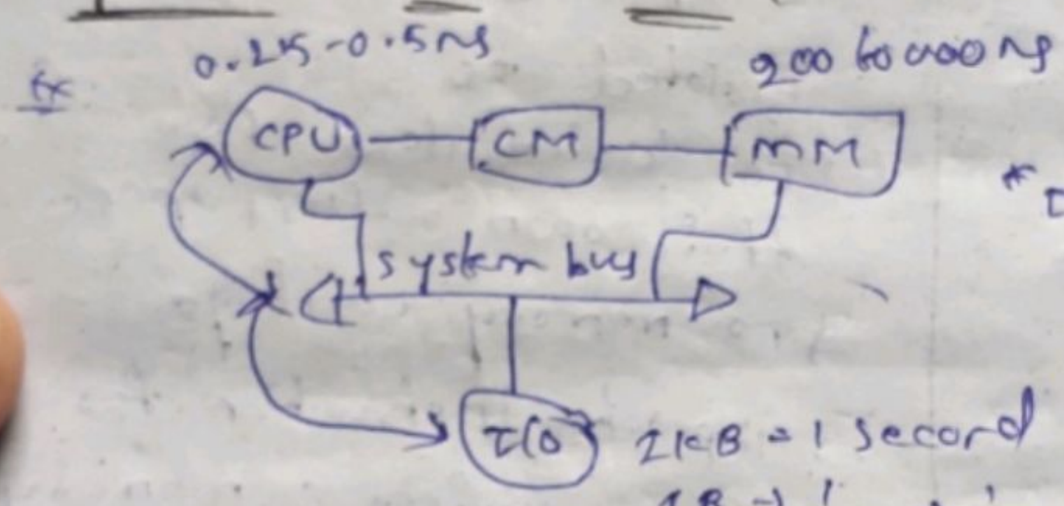
a) How many 8-bit characters can be transmitted per second over 14400 baud serial communication links using parity synchronous modes of transmission with 2 start bits, 8 data bits, 1 - stop bit, 1 parity bit

1 8 2 & extra 1 bit (parity bit)



$\frac{14400}{1+8+2+1} \rightarrow \frac{14400}{12} = 1200$ characters per second

Programmed I/O method



* I/O time ↑ CPU waiting time ↑

1KB = 1 second

$1B \rightarrow \frac{1}{1KB} \rightarrow \frac{1}{10^4} \rightarrow 10^{-4} \rightarrow 1ms$

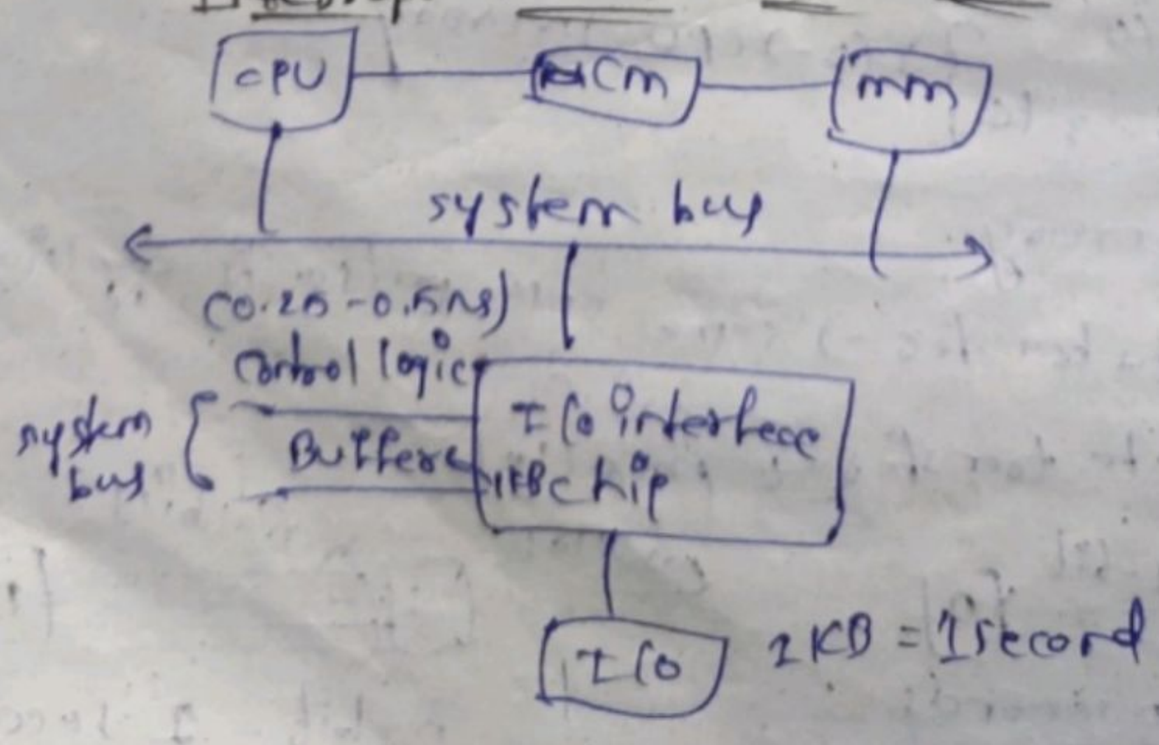
operations

→ In this mode, I/O ~~devices~~ are programmed ~~with~~ in the CPU. So, CPU will be responsible to control the I/O operations.

→ In this mode, the CPU utilization is very poor bcoz CPU will be undergoes waiting until the I/O operations are completed

→ In this mode, the execution time (CPU time) is depends on the speed of I/O device & amount of data to be transfered.

Interrupt Driven I/O method



Accessing sequence

1) CPU initialize the I/O interface chip with command later busy with other task

2) I/O interface control logic interpret the I/O command & enables the I/O operations in the I/O device.

3) Based on the speed of the devices consumes the time to prepare the data later transfer the data to interface buffer

4) when the data is present in the buffer then I/O interface chip sends the interrupt signal to CPU & waiting for interrupt acknowledgment signal

5) After receiving the interrupt acknowledgement signal, buffer content will be transferred to CPU. In this process CPU will be accessing the I/O data from the interface buffer.

→ Therefore, speed gap is synchronized b/w CPU & I/O

6) Different interface chips used in the system design is

8255, 8251, 8259A, 82371
↓ PPI, ↓ USART, ↓ INT controller, ↓ DMA

when the interrupt is occured?

→ First CPU will complete the current instruction
→ store the status of current process in the stack
→ CPU will go to the another process execution accordingly (stack)
→ after the execution is done, resume the previous process by taking out the values from the stack

→ the occurence of an interrupt in CPU h/w an in s/w
→ ISR (Interrupt service Routine)

Types of Interrupts

1) Vector, Non-vector
2) Maskable, Non-Maskable
3) Internal, External (s/w, h/w)

Vector Address :- vector gives location of ISR or it gives address of ISR or reference of ISR

ISR : A routine execution of which service the interrupt

Vector interrupt : Device send the interrupt signal & vector to the CPU

→ CPU reach the ISR by using vector & execute to provide the service

Non-vector interrupt

→ Device sends only the interrupt signal to the CPU

Maskable Interrupt

→ either the CPU can reject or accept the interrupts.

→ Accepted interrupts sometimes kept in waiting (pending) according to priority.

→ Non-Maskable Interrupt

→ CPU will always accept the interrupts because these are high priority interrupts

Internal / software Interrupt

→ CPU generate the errors itself during the instruction execution

→ CPU solve this errors first, then only CPU can execute the current instruction.

Ex :- page fault, segmentation fault

External / Hardware Interrupt

→ If any devices sends an any Interrupt to the CPU, this interrupts also known as Hardware interrupts.

simultaneous interrupts

→ CPU service the interrupt of the highest priority device first

→ To handle the priority, there are 2 solutions.

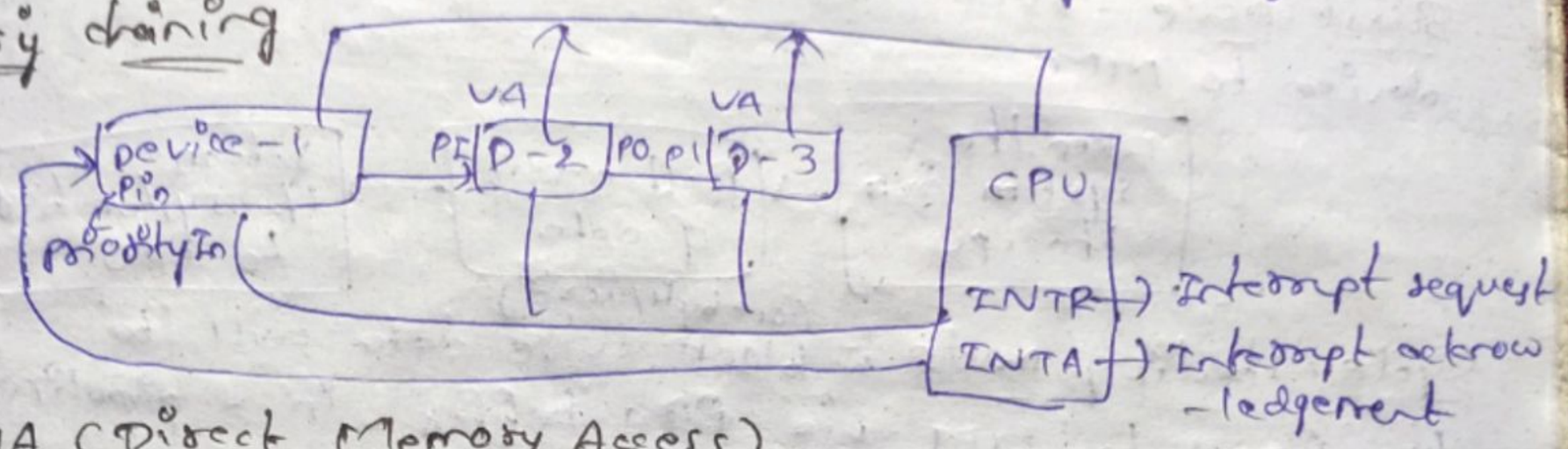
1) software sol^n

2) Hardware sol^n.

pooling sol^n / software sol^n : the cpu will run a program first based on the priority, the cpu resolve the interrupt

Hardware sol^n : there are 2 types : 1) serial (or) Daisy chaining 2) parallel

Daisy chaining



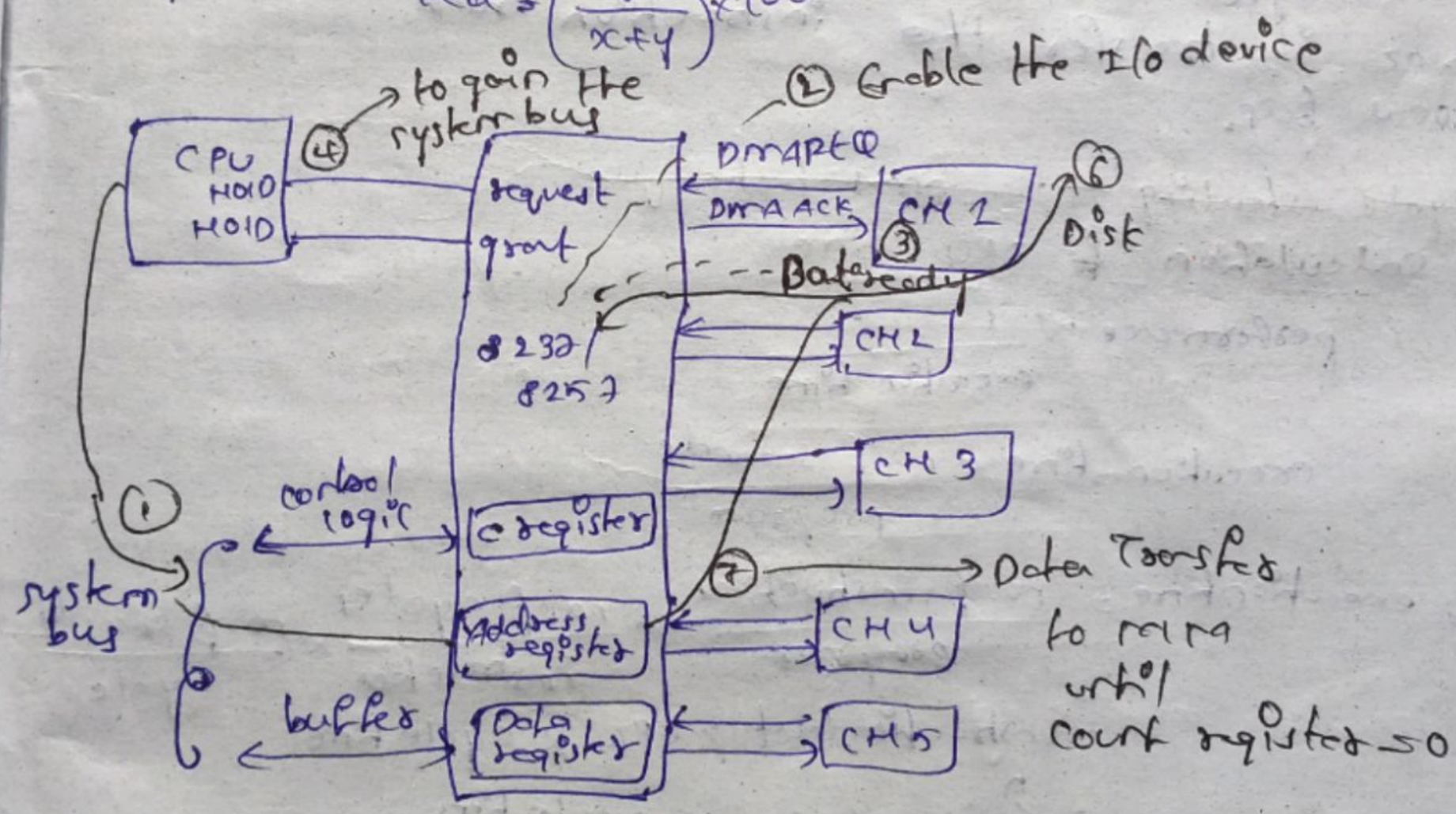
DMA (Direct Memory Access)

% of CPU busy = $\left(\frac{x}{x+y}\right) \times 100$

% of CPU blocked = $\left(\frac{y}{x+y}\right) \times 100$

preparation time = x

transfer time = y



→ the DMA operation CPU present in 2 states :

1) busy state 2) blocked state (hold state)

1) CPU is in busy state until the I/O device prepare the data

2) cpu is in blocked state until the data transfer to Main memory

x is a preparation time

y is a transfer time

→ It has 3 modes:
1) Burst mode
2) cycle stealing mode
3) Interleaving mode

Burst mode : Bulk amount of data transferred from I/O device to MM

Taken the system from the CPU → Bulk amount of data (or byte/word) → Return the system bus to CPU

Cycle-stealing mode : block wise data will be transferred to MM via DMA buffer in a pipelined manner

Interleaved mode : the data transferred in the form of word/byte but with neglegable of CPU waiting time bcoz it transfer the instruction from local bus to local bus.

c) ~~cycle stealing is a DMA technique~~

a) Calculation of CPU time

$$\text{performance} \propto \frac{1}{\text{execution time}}$$

$$\text{execution time} = \frac{\text{no. of seconds}}{\text{program}}$$

$$\text{CPU execution time} = \frac{\text{no. of instructions}}{\text{program}} \times \frac{\text{no. of cycles}}{\text{instruction}} \times \frac{\text{no. of seconds}}{\text{cycle}}$$

$$= \text{Instruction count} \times \text{CPI} \times \text{cycle time}$$

$$= \sum_{i=1}^{n} (IC_i \times CPI_i) \times \text{cycle time}$$

Amdahl's LAW

→ the first parameter common case fast

→ frequently used components modification

→ concentrate on performance gain of the system after enhance-ment part of the system functionality

$$\text{the speed of overall} = \frac{\text{performance of the system enhancement}}{\text{performance of the system without enhancement}}$$

~~(new) key~~

Speed up factor : it is used to measure the performance scale system "x" is compared with system "y", $s = \frac{\text{performance}(x)}{\text{performance}(y)} = \frac{\frac{1}{ETx}}{\frac{1}{ETy}} = \frac{ETy}{ETx}$

$$\frac{\frac{1}{ET(new)}}{\frac{1}{ET(old)}} = \frac{ET(old)}{ET(new)}$$

→ In the enhancement process only the part of the system is modified, so, new system contain two portions:

1) Unenhanced portion
2) Enhanced portion

execution time of new = ET of unenhanced portion + ET of enhanced portion

$$\text{Speed overall} = \frac{1}{\left[(1-f) + \frac{f}{s}\right]}$$

(or) → $S = \frac{1}{1-p}$

serial

$S = \frac{1}{1-(1-p)} = \frac{1}{p}$

f = enhanced fraction

s = speedup factor

p = parallel

Ex $p = 85\% \rightarrow \frac{1}{1-\frac{85}{100}} \rightarrow 20/3$

Pipelining

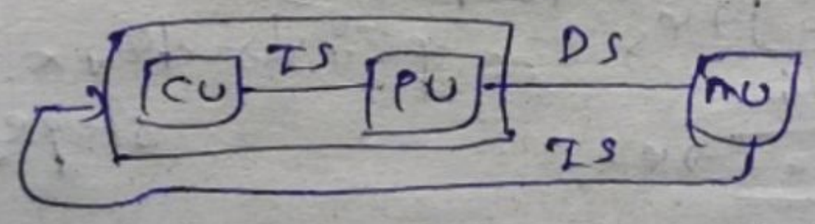
Flyn's classification

IS → Instruction string
DS → Data string

CU = control unit
MU = memory unit
PU = process unit

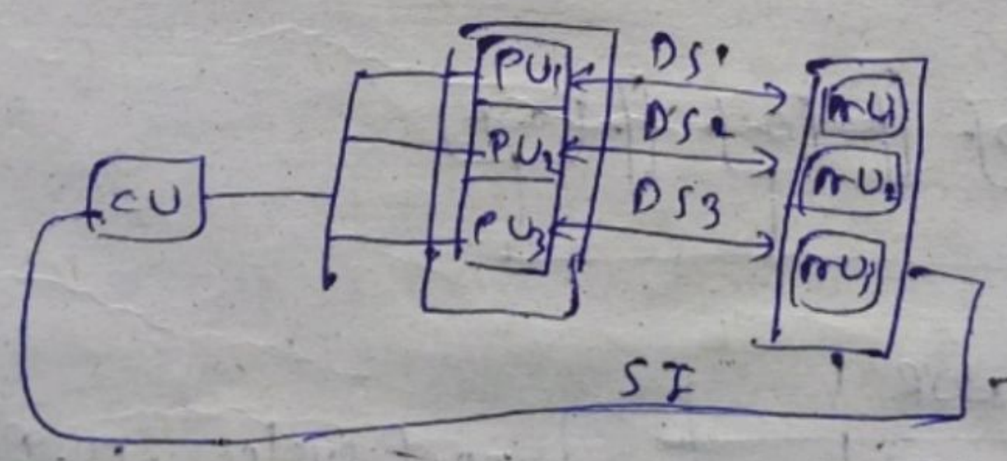
1) SISD (stored pgm concept / von neumann architecture)

SISD (single Instruction single Data)



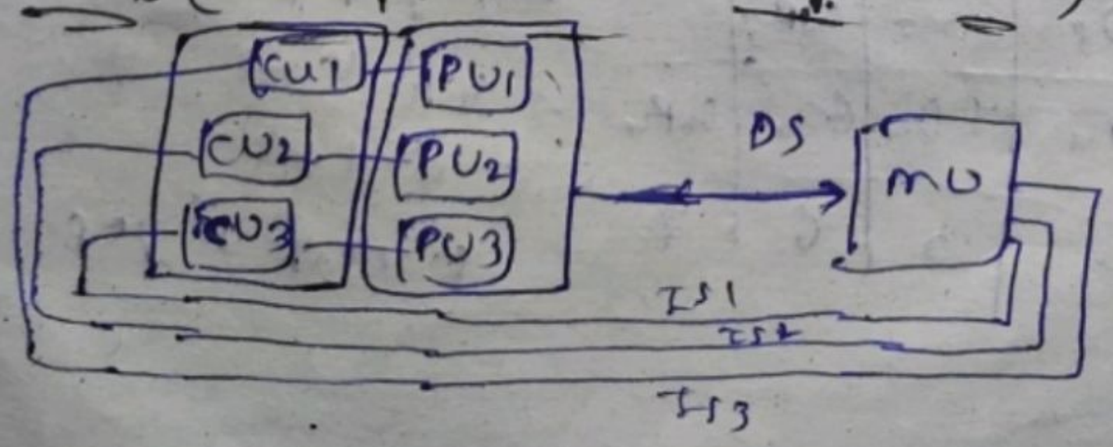
Ex: 8085/8086

2) SIMD (single Instr^n multiple Data)



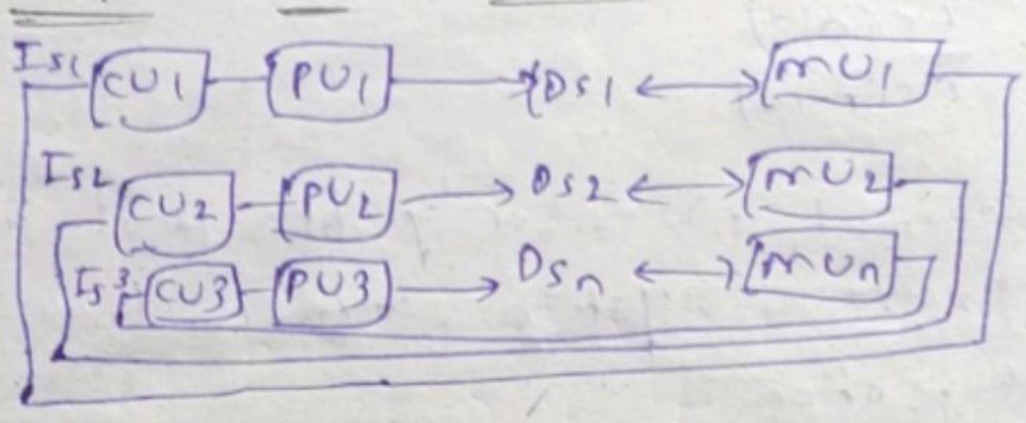
Ex: array ex a[10]
AI & ML
GPU
AMD

3) MISD (multiple Instr^n single Data)



practically it doesn't exist

4) MIMD (Multiple Instr^n Multiple Data String)



Ex: i7, i9
Qualcomm snapdragon

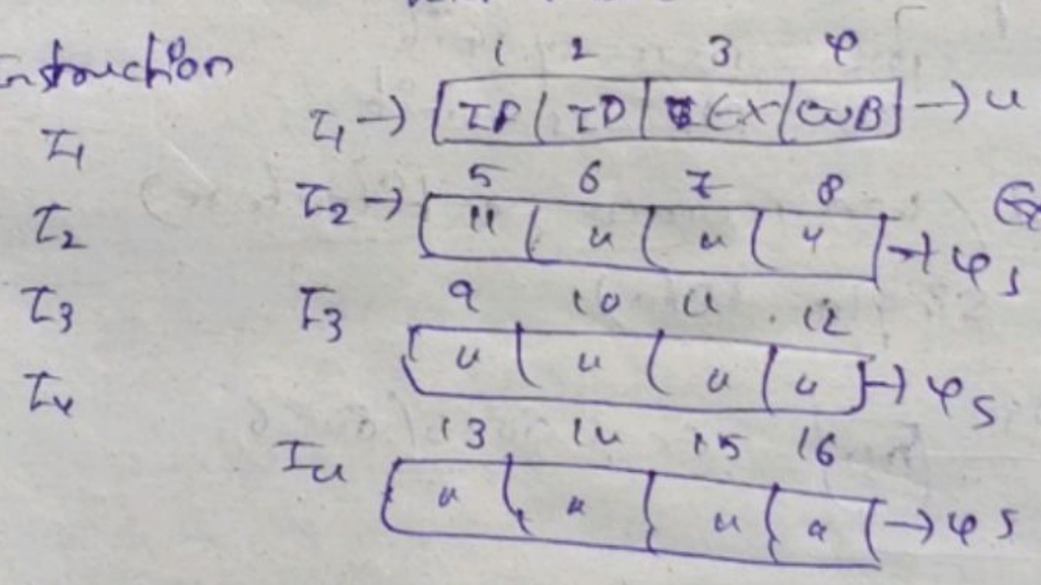
pipelining

→ pipelining uses the decomposition technique
→ the decomposition means → the problem start is divided into independent sub problems & assigned to a independent h/w later connected the h/w in pipelining sequence i.e., h/w is unit o/p is connected as an i/p to the second h/w unit.
→ accepting the new i/p at one end before the previously accepted i/p is appears on accepted the other end

SISD without pipelining

execution states - IF, ID, EX → ALU, WB (memory) (4 states) (or) cycles
fetch, Decode, write back

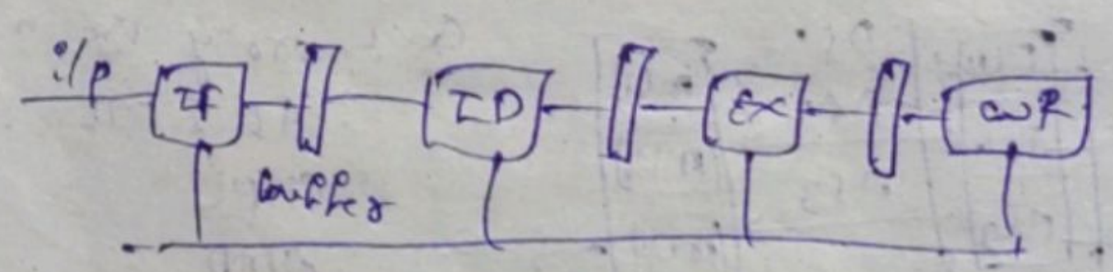
Instruction



Execution time = 4+4+4+4
= 16 seconds

no. of cycles = no. of instructions × no. of stages

SISD with pipelining



I1	IF	ID1	EX1	WR1			
I2		IF2	ID2	EX2	WR2		
I3			IF3	ID3	EX3	WR3	
I4				IF4	ID4	EX4	WR4
	1	2	3	4	5	6	7

no. of cycles = no. of instructions − chang. + k − 1
= 4 + 4 − 1
= 8 − 1 → 7
k = no. of stages

cycle time = stage delay
= maximum (stage delay)

Performance Analysis of pipelining

$ET(non) = n \times t_n \times k$

$ET(pipe) = (n + k - 1)\, t_p$

speed of factor $= \dfrac{ETnon}{ETpipe} = \dfrac{k n t_n}{(k+n-1)\, t_p}$

RISC & SISC

→ RISC
→ the no. of instr^n is less as compared to SISC

→ SISC
→ no. of instr^n are more

→ RISC
→ addressing modes are less

→ SISC
→ addressing modes are more

→ RISC
→ fixed length instr^n format
SISC → variable length instruction format

→ RISC → It consumes low power
SISC → It consumes more power

→ RISC processor highly pipelined
→ SISC processor lessly pipelined

* → RISC → memory chip → program must take less time to executed
(Register Register set Architecture)
SISC → memory is expensive → program must be short
(memory memory instruction set Architecture)

→ RISC → decrease the execution time (faster)
SISC → increase the execution time (slow)

* → RISC → LOAD & STORE are independently instructions
SISC → It supports LOAD & STORE but doesnt work directly

Ex :

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

ADD R_1, A, B
ADD R_2, C, D
MUL x, R_1, R_2

a) suppose, the largest n-bit number requires d-digit in decimal representation. which of the following relationship is correct

a) $n = 2^d$ c) $d < n\log_{10} 2$ d) $d \geq n\log_{10} 2$ b) $d = 2^n$

a) decimal no is 64 digit the no. of bits required for its equivalent binary representation.

$d = 64$

$10^{64} = 2^n$

$10^{64} = 2^n$

$\log_{10} 10^{64} = \log 2^n$

$\frac{\log 10^{64}}{\log 2} = n$

$\Rightarrow \boxed{n = 206.19}$

$d \geq n\log_{10} 2$

$\boxed{ans = 206}$

CA sir no = 9030585404