# Computer Organization



## Computer organization & Architecture

↳ How a Computer works internally → Digital logic (prerequisite)

Circuits & Design

↳ CPU, RAM, I/O → Keyboard, Mouse, disk

Intel i3, i5, ...

4GB | 8GB | ...

IOT → Micro-controllers

Smart phone

Laptop

Smart watch

COA → Hardware

OS → Software

CD, DB

Micro-processors

**what's a Computer:** programmable Computer

CPU

Input data → COMPUTER → OUTPUT

↑ program


Neumann-Architecture

Stored program Computer

program: Sequence of Instructions

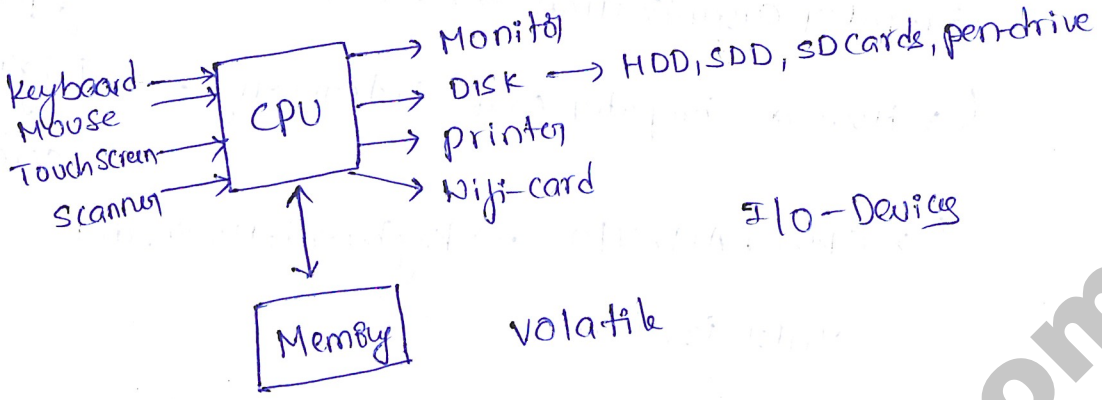
Compiler

C → MLL

Instruction: Binary Sequence / String

data : Binary - seq / String

## Block Diagram:




Memory ← Main → RAM, ROM
Secondary → DISKS

When the power is switched off all the data in main memory is removed.

RAM - Random-Access Memory

ROM - Read only Memory

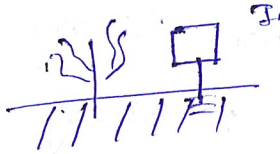
## Mother-board:

ROM

→ cpu slot (Micro processor)
→ PCI slot (can connect sound cards etc)
→ Slots for HDD, DVD-ROMS
→ RAM (Memory) Slots
→ Graphics card slot
→ Back panel with connections for pheripherals and USB Memory.

## Micro Controllers

ESP-32 chip
ARDUINO boards
Rasberry pi boards



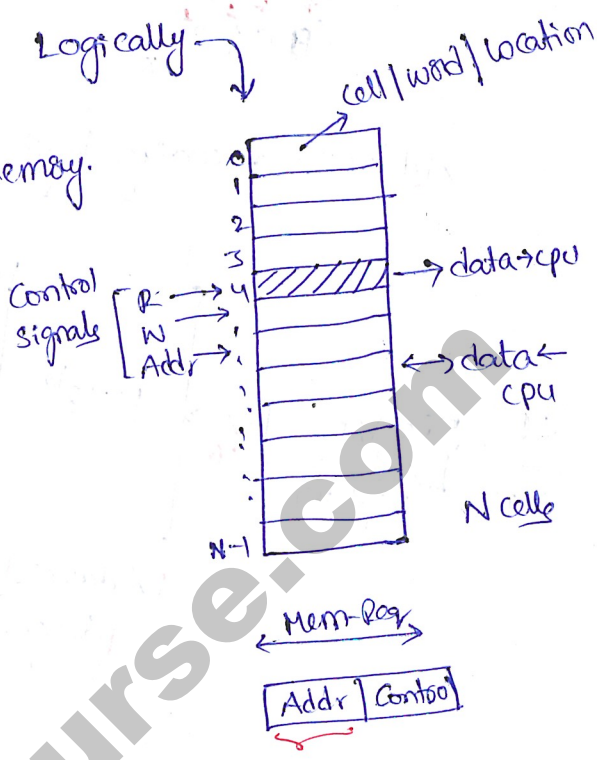
Every thing inside a chip (All Hardware of Computer) is a Micro controller chip.

## Basics of Memory organization:

→ All the programs are in Main Memory.

→ CPU performs memory-Requests

Logically →



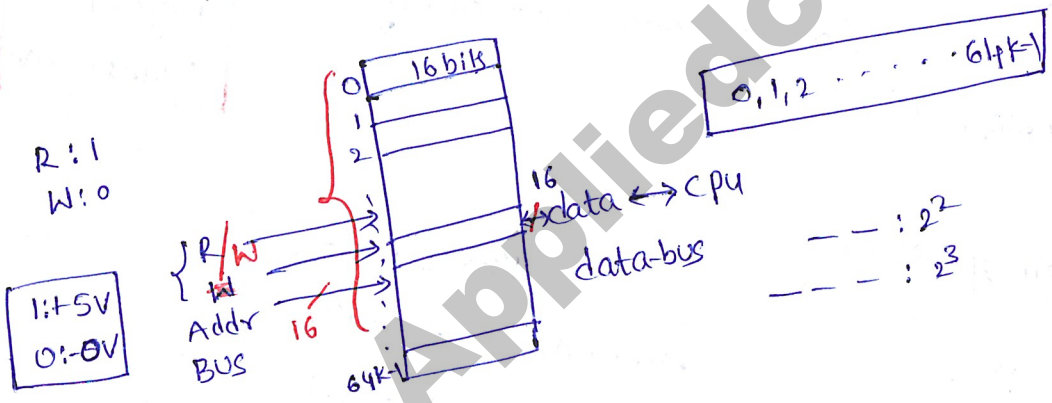

Eg: 6264 Memory chip

Datasheet 6264

64K X 16 → Size of each cell in bits

↓

N = # cells




$1K = 2^{10} = 1024 \approx 10^3$

$1M = 2^{20} \approx 10^6$

$1G = 2^{30} \approx 10^9$

$1T = 2^{40} \approx 10^{12}$

64K

$\Rightarrow 2^6 \times 2^{10} = 16$ bits

$2^{16}$ cells/Locations

K bits → $2^K$

16 bits → $2^{16}$ unique addresses

64K X 32

↓ Address bus ↓ databus = 32 bits

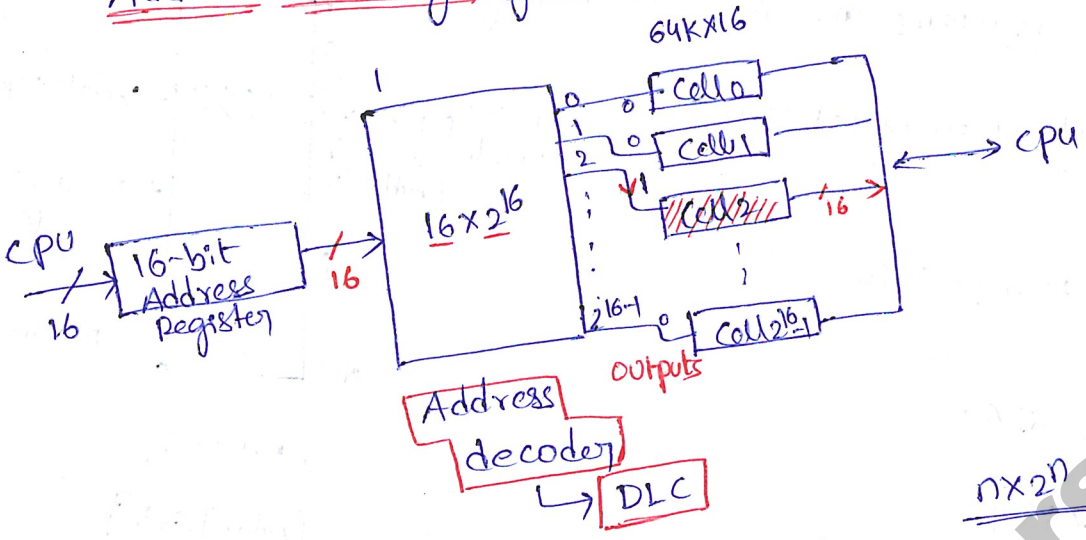
B C 1 C

$(1011\ 1100\ 0001\ 1100)_2$

$(B\ C\ 1\ C)_{16}$

Bin & Hex → C, DLC

## Address-decoding Logic:




## nx2^n : Decoder

At any time only one output is active/high

$(0000\ 0000\ 0000\ 0010)_2$

high: 1 5V
Low: 0 0V

16 bits → 0, 1, 2, ..., $2^{16}-1$

## Pin-diagram of 6264:

VCC: Power

$A_{12}$ - $A_0$: Address Bus

$D_7$ - $D_0$: Data Bus

GND: Ground 0V

R/W: 1 = R 0 = W

NC: Not Connected

OE: Enable output

CE1: chipselect1

} Control pins

13-bit Address

$A_0 - A_{12} \Rightarrow 2^{13}$

$\Rightarrow 2^3 \times 2^{10}$

$\Rightarrow$ 8K

8K x 8

Note: No cohere address will be stored. They are corresponding to the output line of address decoder.

H/W 8085, 8086

8-bit microprocessor

16-bit microprocessor

Most modern processor

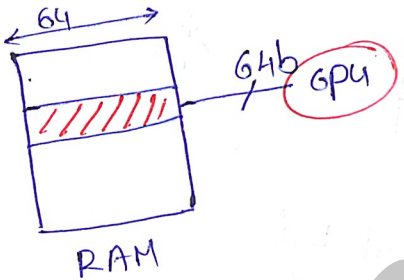
32-bit microprocessor

64-bit microprocessor

Win-64bit → 64 bit MP

Win 32-bit → 32 bit MP

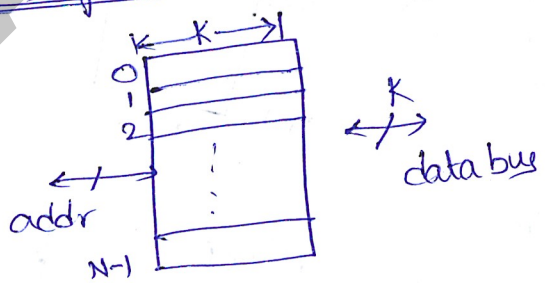
Word-size 64 bits




## Memory Addressability:

### Byte and word - Addressibility

### Physical Address Space:




$2^n$ cells / words

0 to $2^n - 1$

$n = \log_2 N$

**Byte Addressable:** one byte using

↳ 8 bits one Address if **N x 8**

Nibble: 4 bits

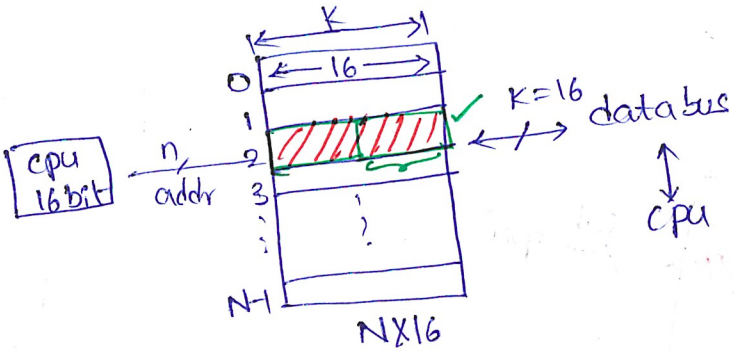
## Word Addressable:

1W = 1B (NX8)

1W = 2B (NX16) → One Address corresponds to 2B.

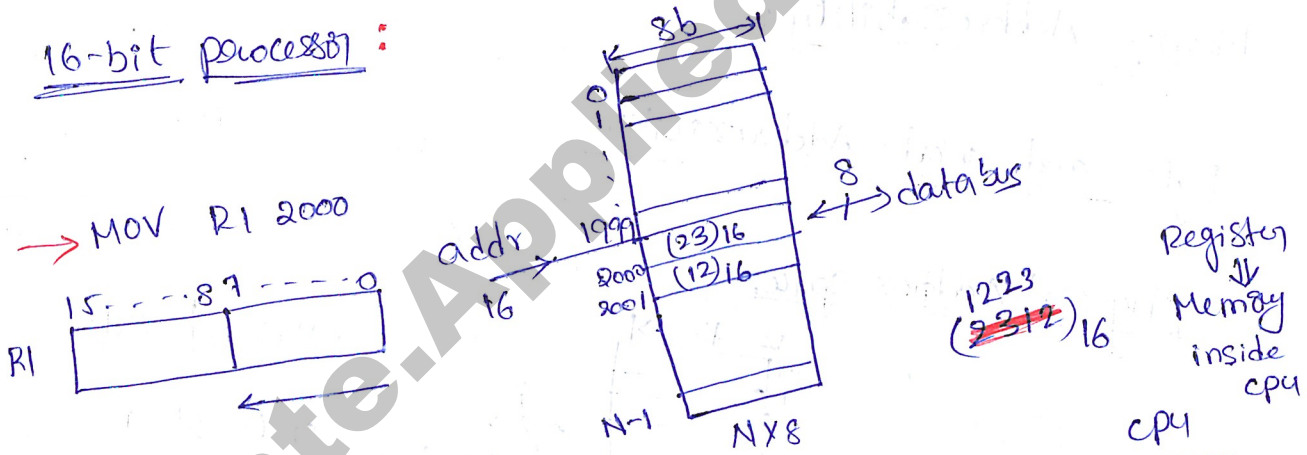
1W = 3B (NX24)

**MP:** Byte Addr




Ignore the first Byte

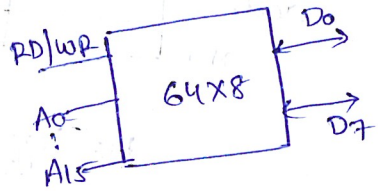
## 16-bit processor:

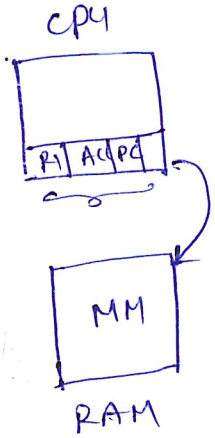



a = x;

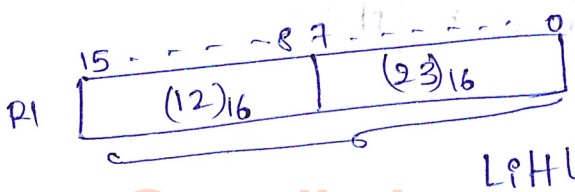
x = x + 1;  compiler →

2B [2000, 2001]



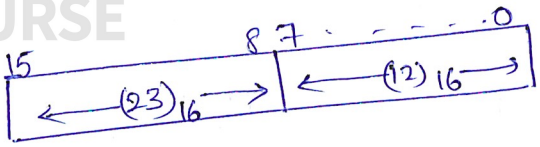




MOV R1 2000

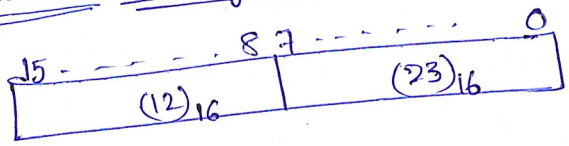



Little Endian

## Big-endian System:



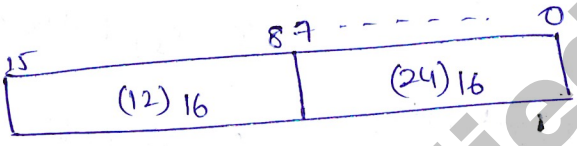

## Little-Endian System:



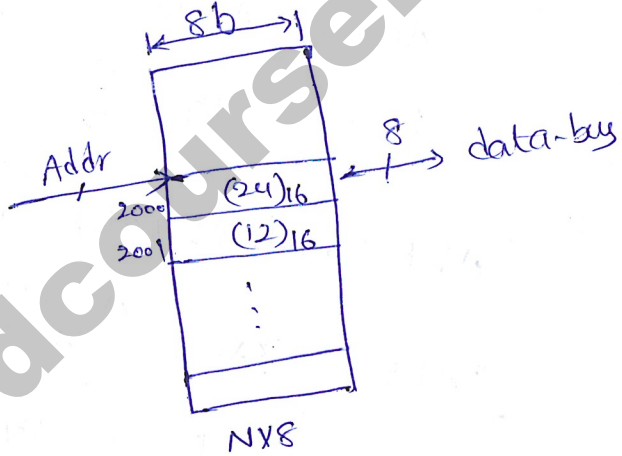

MOV R1 2000 → ALL → MLL

{10 111 010 ▨▨▨ . . . . . }

MOV R1 2000
INCR R1
MOV 2000 R1




(1224)16




## Cpu-Memory Interfacing:

### Pin-Structure:

8085 A
8-bit

8086
16bit

① Active-low → 0

② Active-High → 5V ↓ 1

$\overline{RD}$ RD'
$\overline{WR}$ WR'
$\overline{INTA}$
} Active when they are low

ALE

## (3) Time-multiplexed-Pin:

8085A: AD0 - - - AD7 — 8 bit

A8 - - - A15

1W = 1B

16-addr pins ⇒ $2^{16}$ cells

$= 2^{16}$ B

= 64KB size of MM

A0 - - - A7 A8 - - - - A15 ← addr

D0 - - - - D7 ← data

ALE : Address Latch Enable

Address Lines

ALE = 1 (A0 - - - A7)

ALE = 0 (D0 - - - D7) Data lines

cpu ⟷ RAM

① Send the address

② R & W data

8086 cpu: 16 bit MP

AD0 - - - AD15 A16 - - - A19

(0-19) 20 bit Address

Data ⇒ 16 bits

$2^{20}$ cells = 1MW

↓

2B

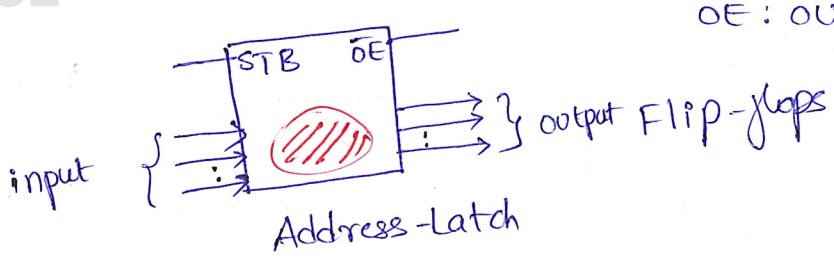
⇒ 2MB

MAX Mode ⟸ MN/$\overline{MX}$ ⟶ 0

MIN Mode ⟸ ↓ 1

## Address-Latch-chip:

STB: Strobe
OE: output Enable

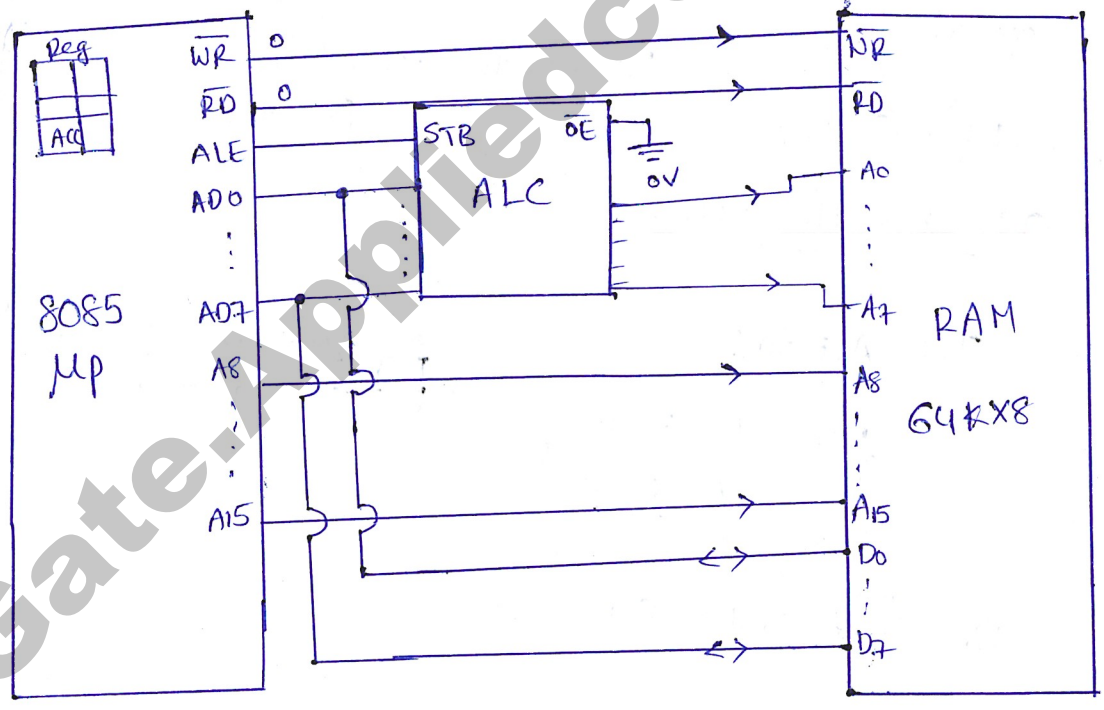



Address-Latch

STB $\begin{cases} 0 \Rightarrow \text{Input is disabled.} \\ 1 \Rightarrow \text{Input is Enabled} \end{cases}$

$\overline{OE}$ $\begin{cases} 0 \Rightarrow \text{output is Enabled.} \\ 1 \Rightarrow \text{output is disabled.} \end{cases}$

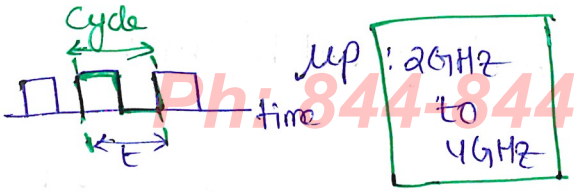
## Cpu-Memory Interfacing:




when ALE 0 Data pins are Connected to RAM

More the pins, the complexity of the Chip will increase.

## ① Clock-cycle:




Speed of µp

cycle-time $= \frac{1}{f}$

$= \frac{1}{\text{freq of operation of µp}}$

MOV, INCR → 4-clock-cycle

2GHz → 4 × 0.5 ns ⇒ 2 ns

4GHz → 4 × 0.25 ns ⇒ 1 ns

$2GHz = f$

$2 \times 2^{30} Hz = f$

$2 \times 10^9 = f$

$2 \times 10^9$ Hz

$2 \times 10^9$ cycle/sec

$t = \frac{1}{2 \times 10^9}$ sec

$= 0.5 \times 10^{-9}$

$= 0.5$ ns

$ms = 10^{-3}$

$\mu s = 10^{-6}$

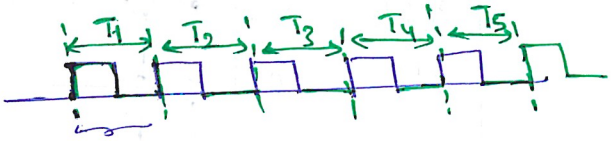
$ns = 10^{-9}$

## ② Machine (M/C) cycle:

# of Cycles required to complete an operation.

one instruction LDA $(2050)_{16}$ → Load into accumulator Register

### (a) Send the address




$T_1$: ALE to 1

$AD_7 - AD_0 : (80)_{16}$

$A_{15} - A_8 : (20)_{16}$

### (b) Send Control Signals:

$T_2$:
- ALE = 0
- $\overline{RD} = 0$
- $\overline{WR} = 1$

µp : 2GHz – 4GHz

RAM: 1444 MHz ≈ 1.4 GHz

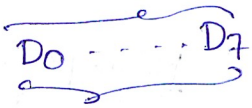
MM → 3GHz very Exp
- 8GB
- 16GB
- 32GB

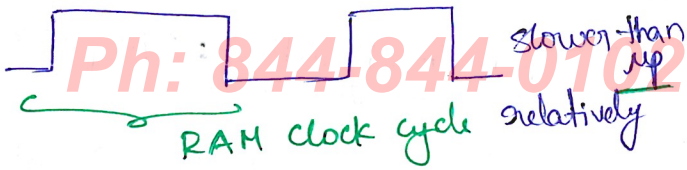
↓ Slower than µp

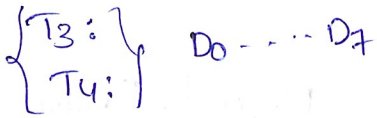
### (c) Read the data to ACC:

$T_3$:

$T_4$:

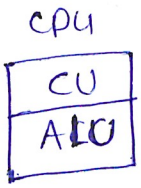



once the data is read from Memory to Data line, processor will store them inside Accumulator.

INCR ACC

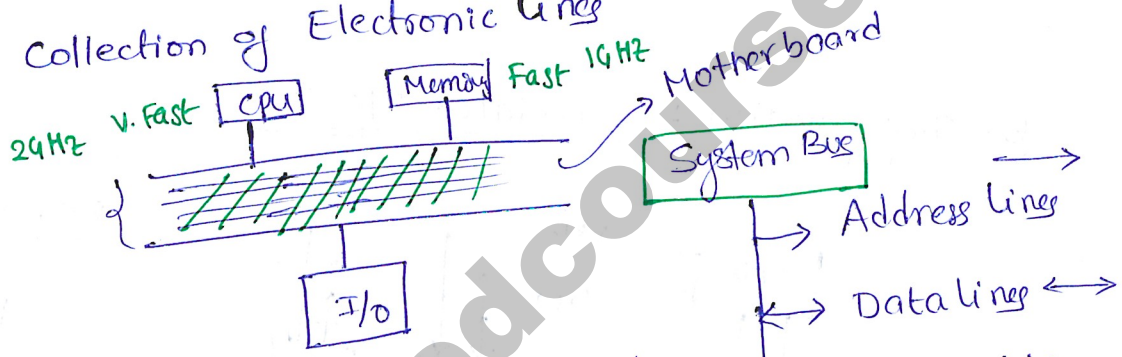



CU control unit

ALU Arithmetic logic unit

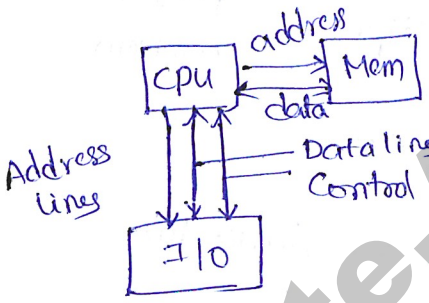
## System Bus Configuration:

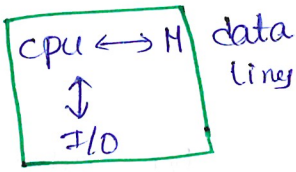
**BUS:** Collection of Electronic lines




Monitor, Keyboard, printer, Mouse, DISK

60Hz




**printer:** pages are over

↓

Need to send the interrupt signal to CPU to display a message to the user

CPU ↓ I/O  CPU ↑ I/O

CPU $\xrightarrow{1011}$ M

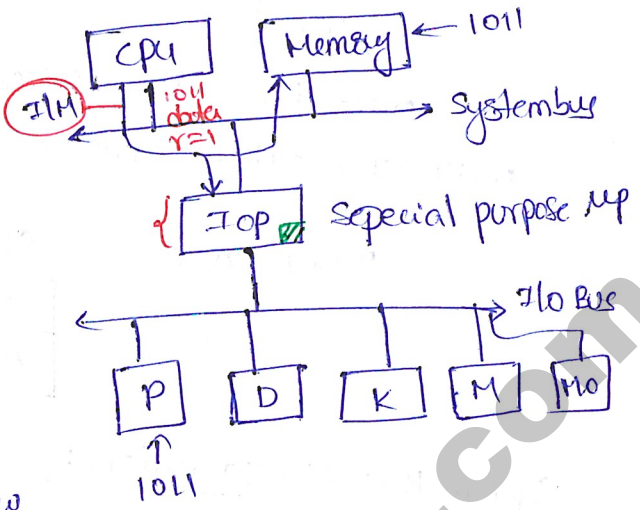
if keyboard address = 1011

## Ambiguity problem:

## Bus Configuration : I/O processor ①

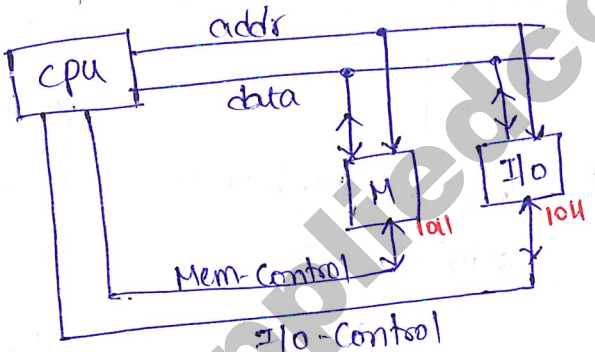


→ Common Control Signals
→ Common Address space
→ Different Buses
→ I/O – Memory direct access
→ Expensive




when I/O is active, M is low
M is active, I/O is low

## Bus Configuration : Isolated I/O ②




we can have multiple I/O devices

Eg: 8085A

Pin 34
IO/M
1 ⇒ IO
0 ⇒ Mem

Each I/O device have unique address

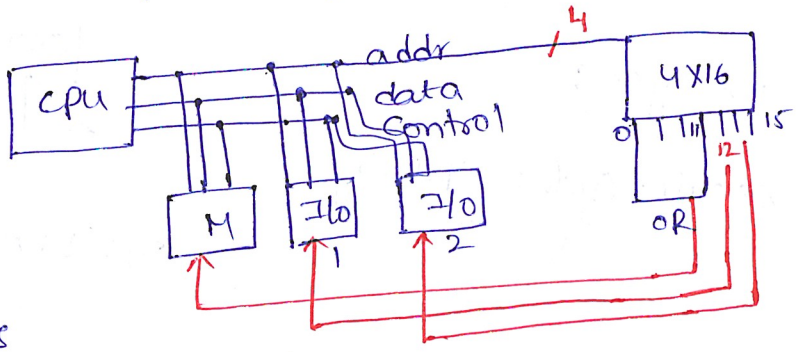
let addr = 1011, RD=1, IO/M Signal

| IO/M | RD | |
|---|---|---|
| 1 | 0 | I/O Read |
| 0 | 0 | Mem Read |

No requirement of IO processor.

## Bus Configuration : Memory mapped I/O (3)

→ Shared address space using decoding

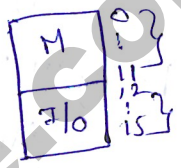


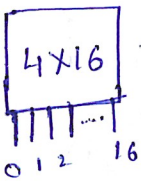

addr : 4 bits

1 1 – –

$2^4$ addr 00 – –
01 – –
10 – –
11 – –
→ Mem-addr

⇒ 11 – – → I/O device addr




(Q) µp that suppots 128MW memory

Memory-Mapped I/O

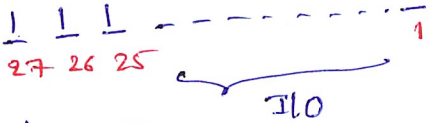
3MSB are 1 ⇒ assign to I/O ports/devices

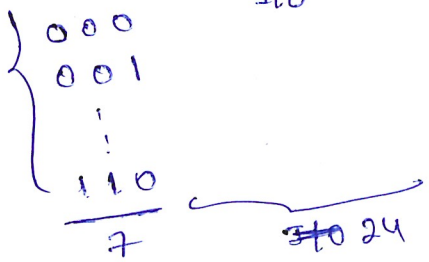
128MW
$2^7 \times 2^{20}$ W
connection point
$\Rightarrow 2^{27}$

Address : 27 bits

# I/O addresses : ?

# Memory addresses : ?



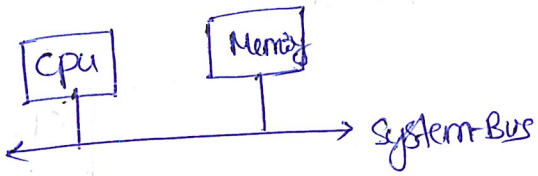




I/O Addresses : $2^{24}$

Memory Addresses :

$7 \times 2^{24}$

## Instruction cycle: Sequence of Execution Steps




① Fetch-Cycle:

② Execute-Cycle: cpu-org

$I_1$: LDA 2080 8085-Syntax

### Fetch-Cycle:

Load Accumulator

$(23)_H$ 0X23

Program:

opcode operand

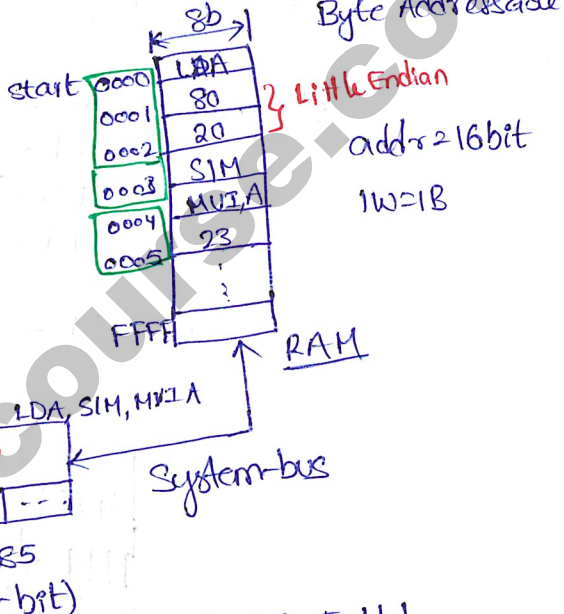
SIM Set Interrupt Mask

$I_1$ : LDA $(2080)_{16}$ → 3B

$I_2$ : SIM → 1B

$I_3$ : MVI A, 23H → 2B

Move Immediate




Program Counter : 0000

Step-Size : 3B

→ Stores the address of the next instruction to be executed.

A, B, C, D, E, H, L

C → gcc → Loader MLL → Loader MM

μp Fetches LDA 2080 goes to Execute cycle

Program Counter : 0003

Step-Size : 1B

μp Fetches SIM

Next goes to Execute Cycle

Program Counter : 0004

Step-Size : 2B

If Instruction is Executed

Program Counter : 0006

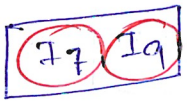
**Fixed-length Instruction cpus:** All instructions are fixed Length

↓

Step size is fixed

**Variable-length instruction cpu:** ↘

(8085, 8086 - - - -)




Stepsize = f (opCode)

Q) 32-bit µp ⇒ 1W = 4B

$I_1$ : 2W 234 to 238 starting memory addr = 0×234

$I_2$ : 1W 23C ⇒ value of pc when $I_6$ is being executed? ↓ $(258)_{16}$

$I_3$ : 1W 240 Memory is Byte addressable.

$I_4$ : 2W 244

$I_5$ : 1W 24C

$I_6$ : 2W 250

$I_7$ : 1W 258

25C

0, 1, 2 - - 9, A, B, C, D, E, F

Hexadecimal

234
8
23C

⇒ 23C
14
240

240
4
244
8
24C
8
250
8
258

Q) Memory is word addressable (same-as-above)

1W = 32b = 4B

program-starting-addr = $(2346)_{10}$

pc-value when $I_6$ is being executed.

$(2346)_{10}$

Start

$I_1$: 2346

$I_2$: 2348

$I_3$: 2349

$I_4$: 2350

$I_5$: 2352

$I_6$: 2353

PC → $I_7$: 2355 . $(2355)_{10}$

$I_8$: 2356

Q3 cpu with 24-bit instructions program:

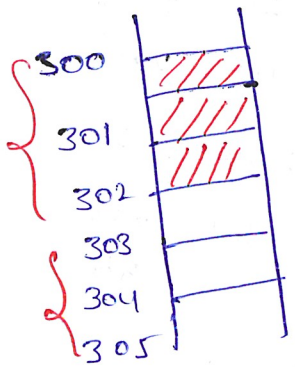
Fixed-len-instruction - cpu

program-load-address = $(300)_{10}$

which of these is a valid value of pc.

24bit = 3B

(a) 400 ✗ (b) 500 ✗ (c) 600 (d) 700 ✗




By default Memory is Byte addressable

pc : 300

pc : 303

pc : 306

: 309

300 + i*3

300 + 100*3

Execution-Cycle:

LDA 2080

opcode operand

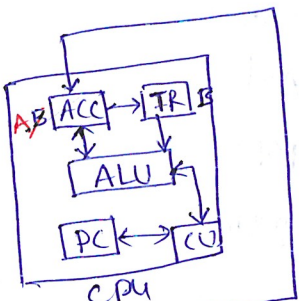
→ Instruction format → 1B → 2B → 3B...

→ cpu-organization

# CPU organization: Stack CPU

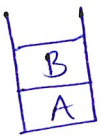
## Stack-CPUs: (Zero address Instructions)

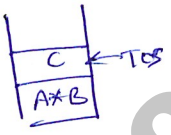
$I_1$ : MOV A, 2080
$I_2$ : MOV B, 2090
$I_3$ : MOV C, 2096
$I_4$ : PUSH A
$I_5$ : PUSH B
$I_6$ : MUL  opcode & No addr
$I_7$ : PUSH C
$I_8$ : ADD
$I_9$ : pop D → D = A*B+C

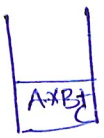



TR - Temporary Register


TOS : Top of the Stack




ALU: Arithmatic and logic unit

→ Zero-address Instruction Format ← ALU instr

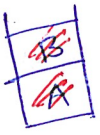
→ Non-All instructions need not be 0-address-instr

→ Instruction-cycle : MUL $I_6$

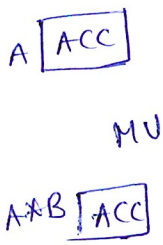
Fetch: Read MUL from memory to cpu based on PC by the control unit.

Execute: Interpret the opcode

pop ← 2nd op
pop ← 1st op
MUL ←
PUSH →

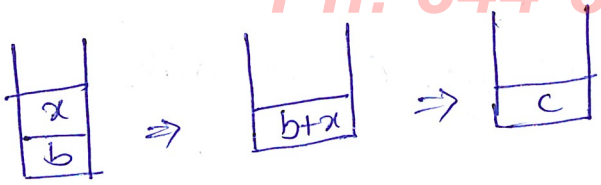




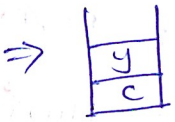


Q Stack-cpu:

I1: Push b
I2: push x
3: Add
4: pop c
5: push c
6: push y
7: ADD
8: push c
9: Sub
10: pop z




c = b + x






⇒ c+y-c ⇒ y


⇒ z = y

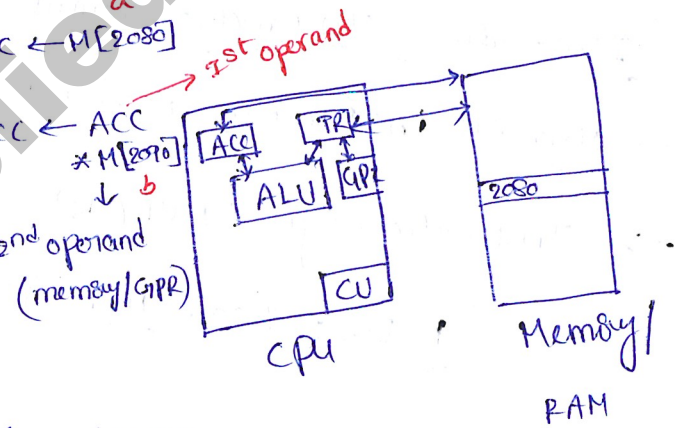
## cpu organization: Accumlator cpu

1-address Instruction

I1 : Load 2080; ACC ← M[2080]

I2 : opcode MUL 2090; ACC ← ACC * M[2090]

I3 : ADD 2096; ACC ← ACC + M[2096]




Instruction Cycle: I2

(a) Fetch: MUL 2090 read from Memory.

(b) Execute: Interpret

TR ← M[2090]

Multiply

ACC ← m[2080] + m[2090]

(a*b)+c

Example: ENIAC (1945) had 20 ACC

- IBM 701 (1952) — General register cpu
- IBM 650
- PDP 8

one-addr-Instructions : opcode Addr — Memory/Register

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

(P1)

1: Load A
2: Add B — ACC ← A+B
3: STORE T — T ← A+B
4: Load C
5: Add D — ACC ← C+D
6: Store Y — Y ← C+D
7: Load T — ACC ← (A+B)
8: DIV Y — ACC ← (A+B)/(C+D)
9: Store X — X

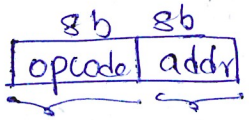
(P2)

1: ~~Load~~ C
load
2: ADD D — ACC C+D
3: ~~Load~~ STORE T — T ← C+D
4: Load A
5: ADD B — ACC ← A+B
6: DIV T — ACC ← (A+B)/(C+D)
7: STORE X

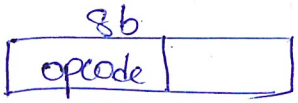
Expand opcode-technique:

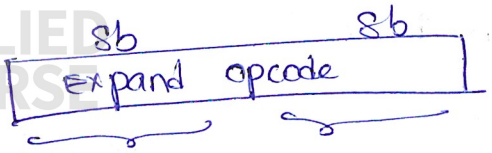
up with fixed-length instructions (16b)

① 1-addr instruction:




② 0-addr instruction:






0000 0000
1111 1111

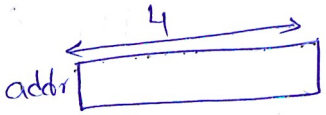
Q) up that Supports 0-addr-inst & 1-addr-instr

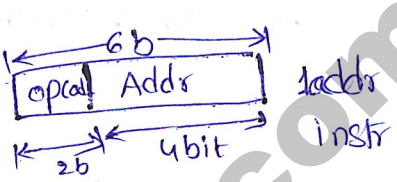
Memory size : 16W = 24 W

fixed-sized instr : 6bits = 1W

Two 1-addr instructions

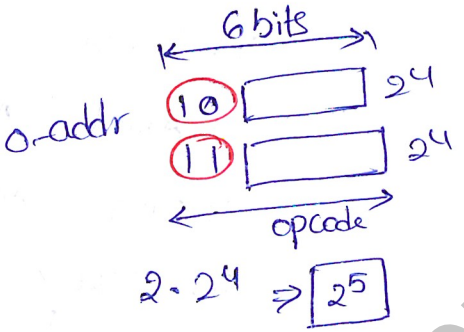
# possible zero-addr instructions = ?





0-addr




$2 \cdot 2^4 \Rightarrow \boxed{2^5}$

Q) 32-bit up:

256 MW memory (max-supported)

fixed-size addr-instr : 1W

Eight 1-addr instr

# possible 0-address instr = ?

256 MW $\Rightarrow 2^8 \times 2^{20}$ W

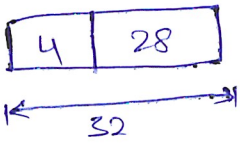
$\Rightarrow 2^{28}$ W

Memory

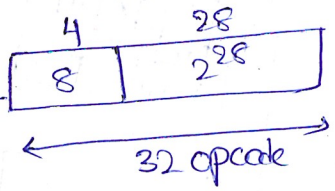
Addr: 28 bits.

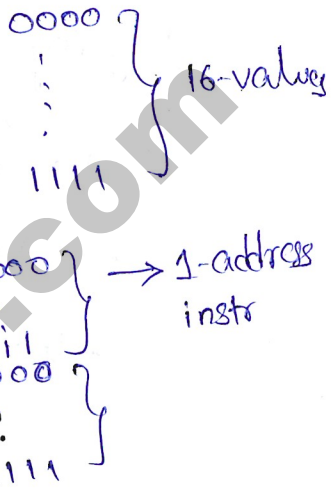
Memory

CPU:




## 0-address instructions:




$\Rightarrow 2^3 \times 2^{28}$

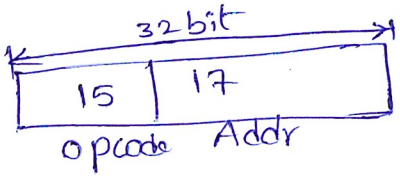
$\Rightarrow 2^{31}$ possible 0-address instructions.

(Q) up with 0-addr & 1-addr instruction

fixed-sized-instr : 32b

Memory : 128 KW $= 2^{17}$ W

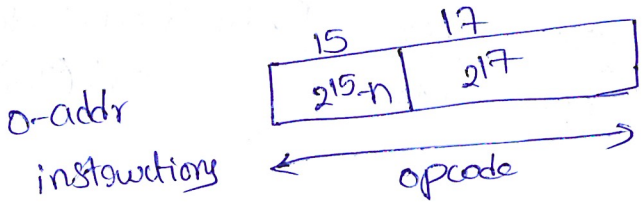
If n 1-addr instruction exist, # 0-address-instr = ?




$2^{15}$

$\Rightarrow 2^{15}$

n 1-addr instructions




$(2^{15}-n)2^{17}$

## CPU organization: General Register CPU

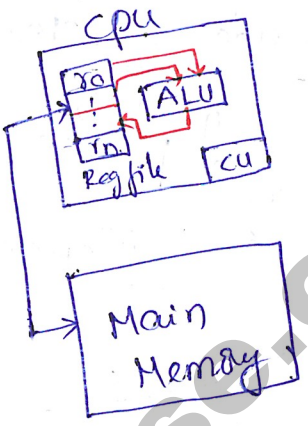
- Reg - Mem Reference architecture
- Reg - Reg Reference Architecture.

### Reg-Reg Reference CPU:

opcode dest op1 op2

$I_1$: ADD $r_0, r_1, r_2$ $\quad r_0 \leftarrow r_1 + r_2$

- 3- Address - instruction
- Load & Store : Reg $\leftrightarrow$ Mem




**Eg:** $X = (A+B) * (C+D)$ (3-reg)

```
Load  r0, A     (2-addr-instn)
Load  r1, B
add   r0, r0, r1
Load  r1, C
Load  r2, D
add   r1, r1, r2  (3-address-instruction)
MUL   r0, r0, r1
Store X, r0
```

$r_0$ □
$r_1$ □
$r_0 \leftarrow A+B$
$r_1 \leftarrow C+D$
$r_0 \leftarrow (A+B) * (C+D)$

### 4- Address instructions:

$I_1$: opcode dest-addr src1 (op1) src2 (op2) addr-next-instr (addr($I_{20}$))

$I_2$
:
$I_{20}$

→ Not used widely

PC → Address of the next instruction.

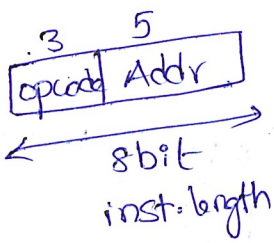


## Instruction - Execution Sequence:

### Accumulator-cpu:

↓

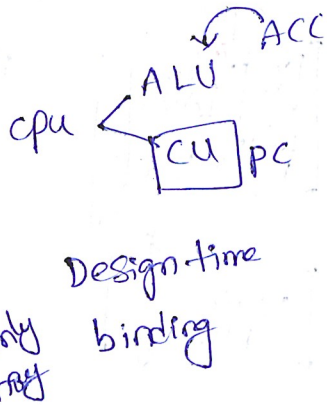
1-Address instruction



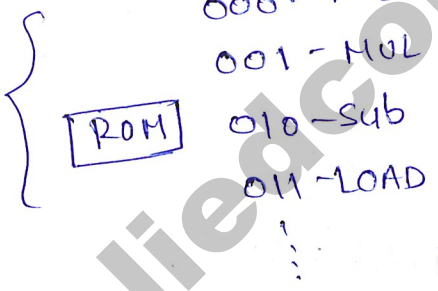

### Fetch:

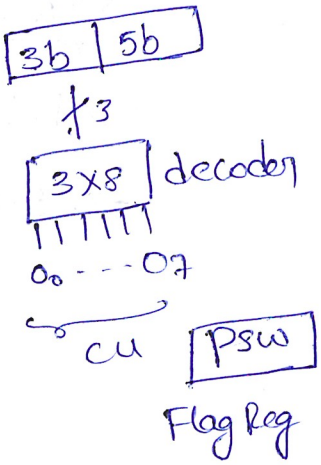
RD = 0 addr of instr

① PC → cpu generate mem-Request → Memory → Instruction Register



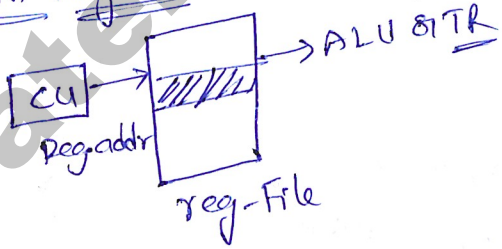

② Instruction decode



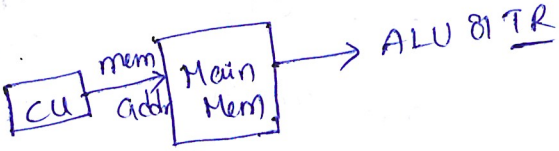




③ operand Fetch:

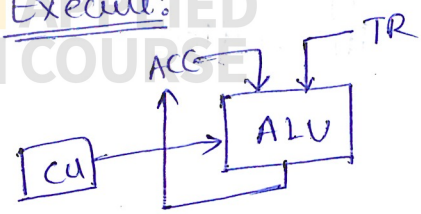
(a) From Register




(b) From Memory



(4) Execute:




## Reg-Mem Reference Architecture:

MUL r0, 2080
operand1 operand2 → Memory Location

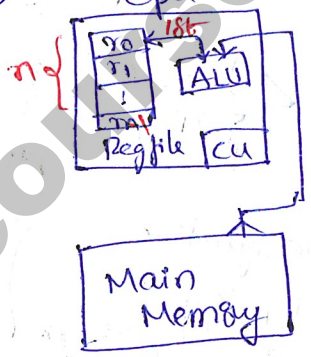
in Reg-Reg Reference
MUL r0, r1
Reg Reg

→ I1 : ADD r0, 2080 (r0 + m[2080])
r0

→ 2-addr-format

→ Fetch

→ Execute: Interpret
Fetch m[2080]
ADD r0 + m[2080]
r0 ← r0 + m[2080]




Reg-File
↓
Collection of Registers

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

1: MOV r0, A
2: Add r0, B
3: MOV r1, C
4: Add r1, D
5: MUL r0, r1
6: MOV X, r0

r0 ← A
r0 ← A+B
r1 ← C
r1 ← C+D
r0 ← r0 * r1

{ MOV T r1
MUL r0 T

(Q) CPU that supports 2-addr, 1-addr & 0-addr instructions.

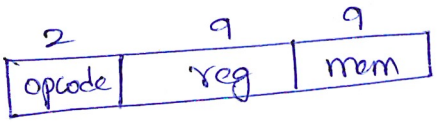
20-bit instructions

512W memory

Two 2-addr-instr

150 1-addr-instr

# possible 0-addr-instructions = ?



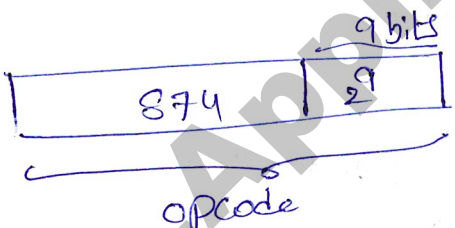

Memory
512W

2-addr { 00, 01, 10, 11 } → 1-address, 0-address


$2 \times 2^9 \Rightarrow 2^{10} \Rightarrow 1024$

- given 150 - 1 addr instructions

Zero-address Instructions:




$\Rightarrow 874 * 2^9$ zero-address instructions possible.

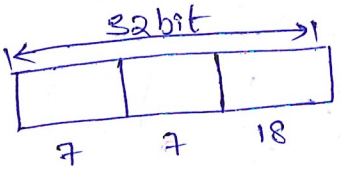
(Q) instruction-format <opcode, reg-addr, mem-addr>

# registers = 128 = $2^7$

256 KW Memory

32-bit instructions

256 ⇒ 28 KW ⇒ $2^{18}$ W




(a) n 2-addr instructions

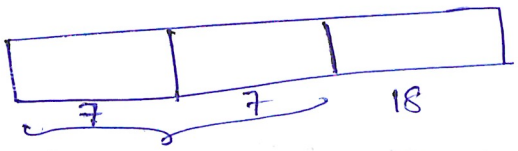
# 1-addr instr = ?

$2^7$ possible 2-address Instr

given n- 2Address instructions.

$2^7 - n$

1-address Instructions

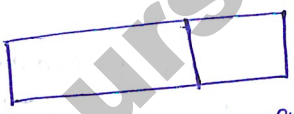



$\Rightarrow \boxed{(2^7-n)*2^7}$ $\Rightarrow$ 1-address instructions.

(b) n 2-addr instr

m 1-addr instr

# 0-addr instructions = ?




$\frac{(2^7-n)*2^7}{\text{possible 1-address Instructions}}$

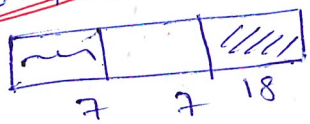
Given m - 1address Instructions

$(2^7-n)*2^7 - m$

$\Rightarrow$ # of possible zero address instructions.

$[(2^7-n)*2^7 - m]*2^{18}$ opcode Memory

opcode, reg




No memory

$(2^7-n)$ reg $2^{18}$ $[(2^7-n)2^{18} - m]*2^7$

Program status word: (psw) → Currently-executing-instr

→ Conditional Flags
→ Control-Flags

Conditional Flags:
- Carry (8086)
- Parity
- Auxillary carry
- Zero
- Sign
- overflow.

ALU

Control-Flags:
- TRAP
- Interrupt
- Direction

$i_3, i_5, i_7, i_9$

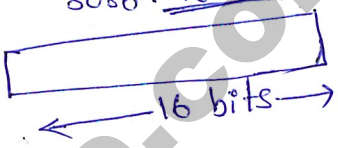
8085: 8 bit

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| S | Z | | AC | | P | | C |

1W = 1B Flag-Register

Flag: 1 (Set)
0 (Reset)

8086: 16-bit




① Carry-Flag: (CY)

ALU → 4 bit unsigned int ⇒ 0 to $2^{4}-1$
0 to 15

0000 0
⋮
0111 to
⋮
1111 15

→ 8+5

1000
0101
1101 = ACC, CY=0

No carry

⇒ 8+9

1 1000
1001
0001 = ACC CY=1 ⇒ 10001

② Parity-Flag:

(Eg) ACC = 1010 1010 → 10111010

PF=1 if # of ones is even

PF=0 if # of ones is odd.

ACC = 1011 1111

⇒ Error-correcting & Detection Codes

③ Auxillary-carry: set if ∃ a carry from LN to UN

```
  2A
+ 34
ACC = 5E
```

4 bits: Nibble

BCDEF

```
  2A
  2A
ACC = 54
```

AC = 0

AC = 1

10
10
(20)

Binary-coded-Decimal-Arithmetic: (BCD)

BCD → Unpacked
BCD → packed

$(26)_{10} = (11010)_2$

16+8+2

$(26)_{10}$ = 0000 0010 0000 0110

packed ↓ Use Nibble

ⓐ $(26)_{10}$

⇒ 26 = 0010 0110
27 = 0010 0111 +
53 = 0100 1101 = ACC  CY=0  AC=0

Rule: If LN > 9 then add 6 to Accumlator & AC=0

0100 1101
0000 0110
0101 0011

0 - - - 15 Hexadecimal
0 - - 9 Decimal

⑤

```
  28
  29
+ 57
```

0010 1000
0010 1001
0101 0001

0101 0001
0000 0110
0101 0111

Rule: If LN < 9 & AC = 1 then Add 6

© 88 + 89 = 177

1000 1000
1000 1001
1 0001 0001

AC=1
CY=1

Rule: AC=1 & LN<9 add 6 to LN

CY=1 & UN<9 add 6 to UN

⇒

1 0001 0001
0110 0110
1 0111 0111

177

(4) Zero-Flag: Acc=0 ZF=1

2000: LDA 2080
2003: JZ 4000
2004:
:
4000: ==

JZ: Jump if zero

pc: ~~2004~~ ↓ 4000

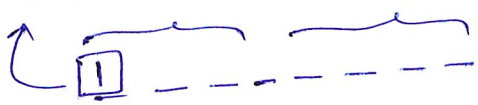
JNZ: Jump if Non-zero

if(x==0) { } else { }

for(i=10; i>0; i--) { == }

while
switch

(5) Sign-Flag:




Signed-Integer

(6) overflow : Signed - Airthmatic

unsigned ⟵ 0 - $(2^4-1)$

0 - 15

| 2's Complement - Binary | Equivalent Decimal |
|---|---|
| 0010 | +2 |
| 0111 | +7 |
| 0000 | +0 |
| 1000 | -8 |
| 1111 | -1 |
| 1101 | -3 |

Signed -8 ... -1 +0 ... +7

1000

1's Complement ⇓ 0111

2's Complement ⇓ 0111 + 1 = 1000 (-8)

-1 ⇒ 0001

1's ⇓ 1110

2's 1110 + 1 = 1111

-3 ⇒ 0011

1's ⇒ 1100

2's 1100 + 1 = 1101

n-bit unsigned Integers

$0 - 2^{n-1}$

Signed Integers

$-(2^{n-1})$ to $+(2^{n-1}-1)$

(a) +7 0111

+3 0011

+10 > +7 1010 ⇒ overflow = 1

(b) +7 0111
−3 1101
+4 < 7 1 1 1
(1) 0100

OV = 0

(c) −7 11
+3 1001
−4 0011
1100

⇒ 1100 = −4

No In carry
No OUT carry

OV = 0

| | Incarry | out-carry | overflow |
|---|---|---|---|
| (c) | 0 | 0 | 0 |
| (d) | 0 | 1 | 1 |
| (a) | 1 | 0 | 1 |
| (b) | 1 | 1 | 0 |

(d) −7 1 1001
−3 1 1101
−10 0110

−10 −8, ... −10 ... −7

OV = 1

MSB

Incarry XOR OUT carry

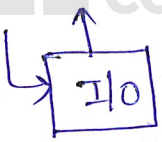
debugging: gdb

## Control Flags:

(1) TRAP → 0 (GIO)
→ 1 (TRCE) single-step (break)

→ $I_1$ : ⟷ Var & Reg
→ $I_2$
:
:

② Interrupt-Flag:

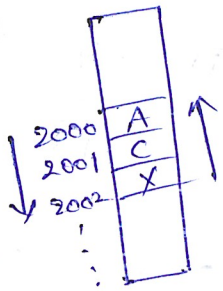



→ 0 (disable-INTR)

→ 1 (Enable-INTR)

③ Direction Flag: String-instructions → Seq of char

1B




0 : L → H ACX

1 : H → L XCA

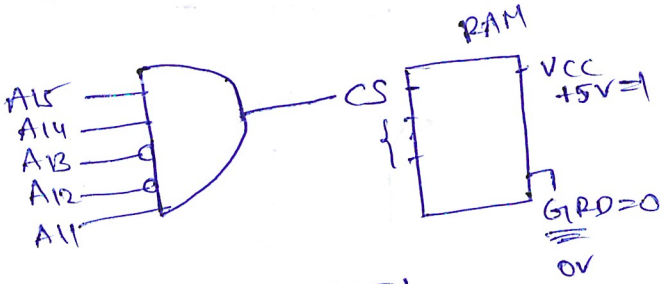
PSW

8086

8085

## More Solved problems:

① The Chip select logic for a certain DRAM chip in a single memory system design is shown below. Assume that the memory system has 16 address lines denoted by A15 to A0 what is the range of addresses (in hexa decimal) of the memory system that can get enabled by the chip select (CS) signal?




⇒ CS=1

(A) C800 to CFFF

(B) CA00 to CAFF

(C) C800 to C8FF

(D) DA00 to DFFF

$A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}\ A_{10} \cdots A_0$

COT OST DLD

1 1 0 0 1

1100 | 1 000 | 0-F | 0-F

C

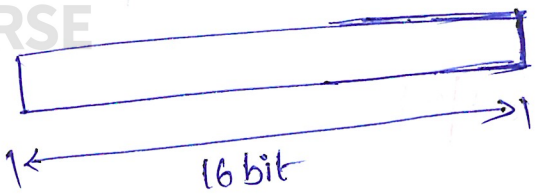
1111

C 8-F 0-F 0-F

⇒ C800 ⇓ CFFF

(2) A processor has 16-integer registers (R0, R1, -- R15) and 64 floating point registers (F0, F1, -- F63). It uses a "2-byte" instruction format. There are 4 categories of instructions: Type-1, Type-2, Type-3 and Type-4. Type-1 category consists of 4 instructions, each with 3 integer register operands (3R). Type-2 category consists of eight instructions, each with 2-floating point registersoperands (2FS). Type-3 category consists of fourteen instructions, each with one integer register operand and one floating point register operand (1R+1F). Type-4 category consists of N instructions, each with a floating point register operand (1F). The maximum value of N is _____

(1)




4 bits
int reg

float 6 bits

Type 1: [4] 4 4 4 4 ⇒ $2^{12} \times 4 = 2^{14}$

Type 2: [8] 4 6 6 ⇒ $2^{12} \times 8 = 2^{15}$

Type 3: [14] 6b 4 6 ⇒ $2^{10} \times 14$

Type 4: [N] 10 6b ⇒ $N \times 2^6 = 2048$

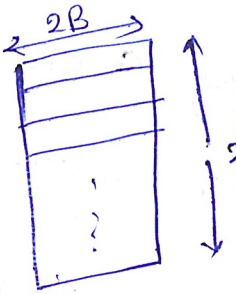
$2^{16} - 2^{14} - 2^{15} - 2^{10} \times 14$

$2048 = 211$

$\Rightarrow N = \frac{2^{11}}{2^6} \Rightarrow 2^5 = 32$

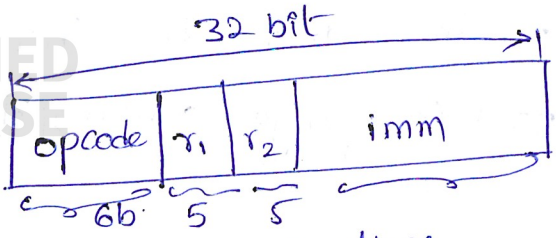
(Q3) A processor can support a maximum memory of 4GB, where memory is word-addressable (a word consists of two bytes). The size of the address bus of the processor is at _____ least bits.

(31)




$\Rightarrow \frac{2^{32}}{2} \Rightarrow 2^{31}$

(Q4) 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 _____




40 distinct instructions $\Rightarrow 32 - (6+5+5)$

$\Downarrow$

16 bits

$2^5 = 32$

$2^6 = 64$

$2^4 = 16$ } reg

$2^5 = 32$ }

(Q5) 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 a memory in a byte-aligned fashion. If a program has 100 instructions the amount of memory (in Bytes) consumed by the program text is _____

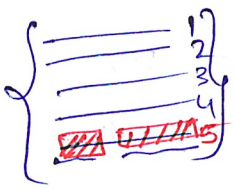
twelve $\Downarrow$ 4 bits

opcode, $sr_1$, $sr_2$, $d_1$, 12bit imm

$\Downarrow$ $\Downarrow$ $\Downarrow$ $\Downarrow$ $\Downarrow$

4b 6b 6b 6b 12

$\frac{34b}{8} = 4.25B \approx 5B$




$5 \times 100 = 500B$

(Q6) For computers based on 3-address instruction format, each address field can be used to specify which of the following

S1 : A memory operand

S2 : A processor register

X S3 : An implied accumulator register.

(A) Either S1 or S2

(B) Either S2 or S3

(C) Only S2 and S3

(D) All of S1, S2 and S3.

3-address

opcode addr1, addr2

(Q7) Consider two processors $P_1$ and $P_2$ executing the same instruction set. Assume that under identical conditions for the same input, a program running on $P_2$ takes 25% less time but 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 1GHz, then the clock frequency of $P_2$ (in GHz) is _____

(a) 1.6

(b) 3.2

(c) 1.2

(d) 0.8

$P_1$ $P_2$

$f_2$ 1GHz $f = ?$

$\Rightarrow t = \frac{1}{f}$

$= \frac{1}{1 \times 10^9}$

$= 10^{-9} s = 1ns$

$P_1$

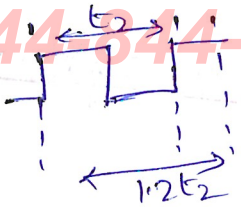
1 CPI

$t_1 = 1\,ns$

$P_2$

1.2 CPI

$t_2 = 0.75\,ns$



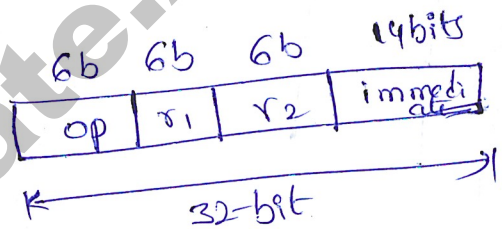

$\Rightarrow 0.75\,ns = 1.2t_2$

$\Rightarrow t_2 = \frac{0.75\,ns}{1.2}$

$f_2 = \frac{1.2}{0.75}\,GHz$

$= \boxed{1.6\,GHz}$

(Q8) A machine has 32-bit architecture, with 1-word long instructions. It has 64 ($2^6$) registers, each of which is 32 bits long. It needs to support 45 ($< 2^6$) instructions, which have an immediate operand in addition to two register operands. Assuming that the immediate operand is an unsigned integer, the maximum value of immediate operand is ——




14 bits

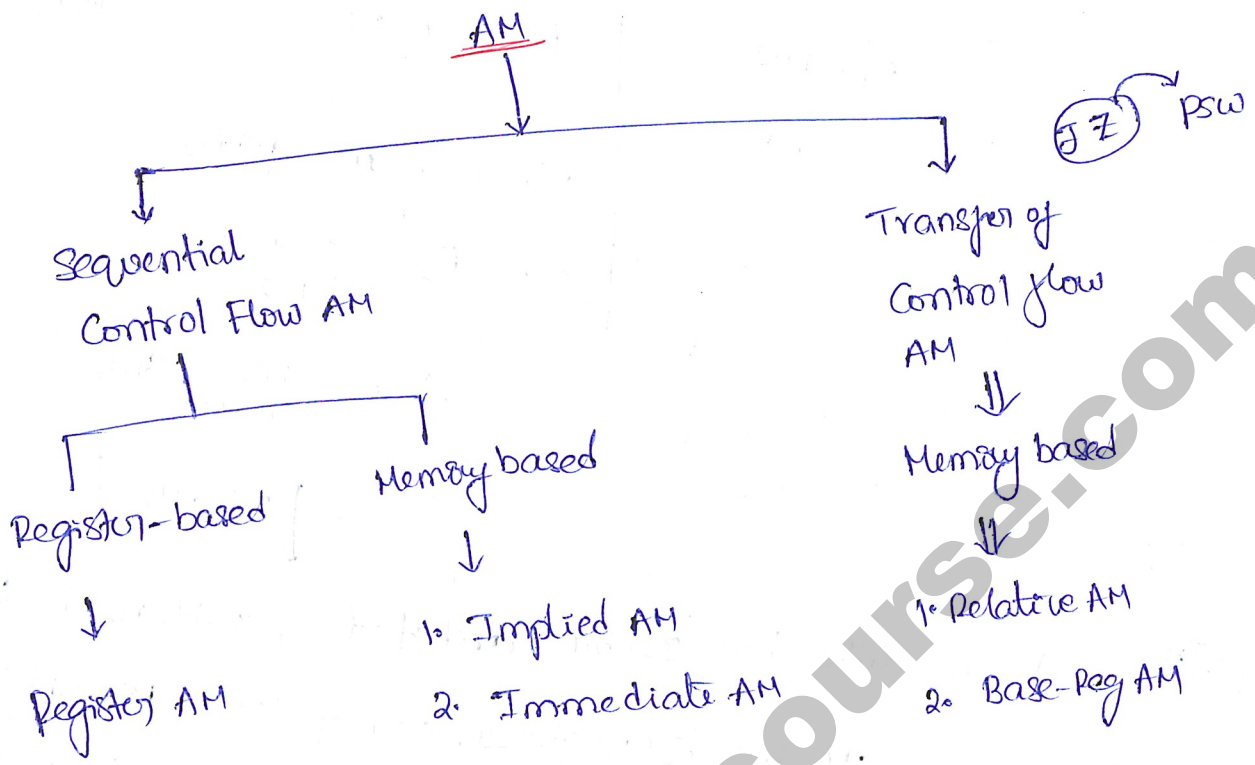
$0 \text{ to } (2^{14}-1)$

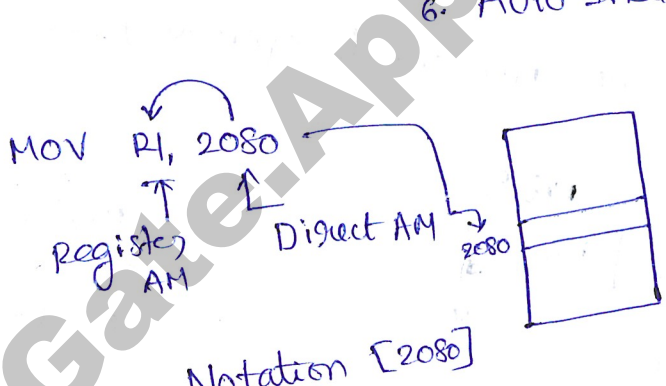
$\Rightarrow 16384-1$

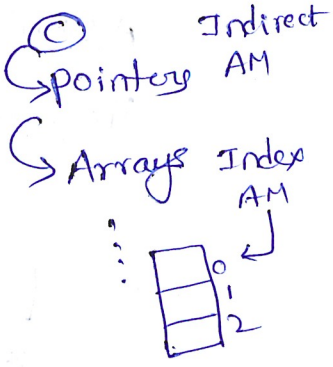
$\Rightarrow \boxed{16383}$

# Addressing Modes: Introduction



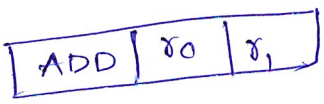


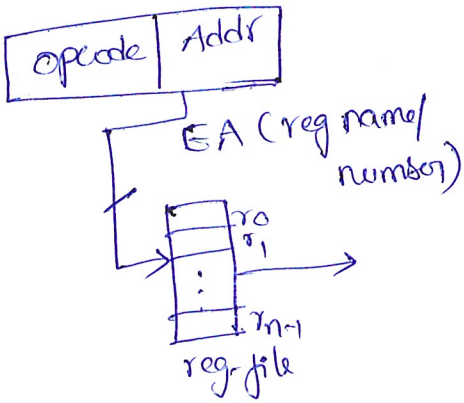

## Register-based AM:

Eg:




$r_0 \leftarrow r_0 + r_1$

Effective Address



## Instruction decoding Stage (ID)

PC → IR ← ////

Register | memory Reference (Reg)

**Execute stage:** 3-reg mem-reg

$r_0 \leftarrow r_0 + r_1$ (ALU)

W

mem-reference

**Implied / Implicit AM:**

1 reg mem reg

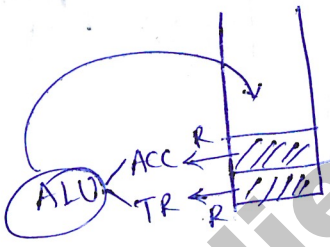
eg: STC set carry → PSW/flag-reg, CY ← 1

CLC clear carry CY ← 0

0-addr-instr

| opcode |
|---|

ADD (stack cpu)




zero-addressing instructions

**Immediate AM:**

(Eg) MOV $r_0$ $(23)_{16}$ → $r_0 \leftarrow 23$

source

Constant

| opcode | addr |
|---|---|

Range is limited

↑

data (limited bits)

int a = 56;

(Eg) MOV $(23)_{16}$ $r_0$

dest src

Meaningless

$(23)_{16} \leftarrow r_0$ ✗

int 56 = a;

(6) up with multiplexed $AD_0 - AD_{23}$

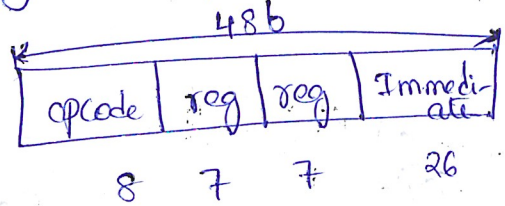
(8) 256 2-word instructions

(7) 128 processor-registers

address - 24 bits
Data - 24 bits.

Instr format: opcode, reg, reg, immediate

Largest unsigned number in immediate field?

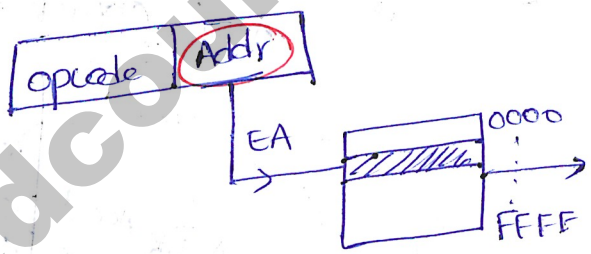



$= 2^{26} - 1$

## Direct/Absolute AM:

Eg: Add $r_0$, [2080]

$r_0 \leftarrow r_0 + m[2080]$




Usage: static-variables

```
int main() {
    static int i=0;
    :
}
```

Dir AM

Imm AM

Compiler-Design

Symbol Table

MOV [2080] $(00)_{16}$

Life time of static variable: program throughout the

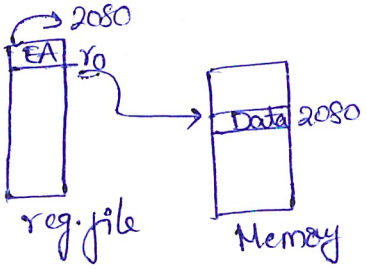
## Indirect Addressing Mode:- pointers

Eg: Add @$r_0$, $r_1$ (~~memory~~ Register Indirect)

reg.indir AM

reg.Am

[$r_0$]



$M[2080] \leftarrow M[2080] + r_1$

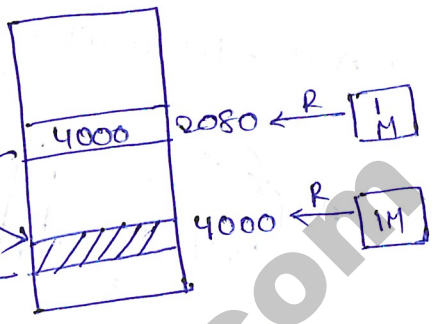
Eg: Add @2080, $r_1$ (Memory - Indirect AM)

mem-indir reg AM

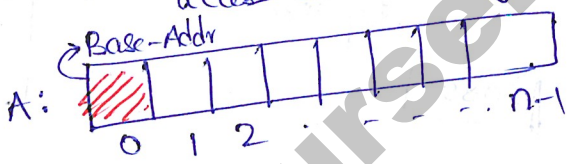
AM

IR

$m[4000] \leftarrow m[4000] + r_1$




Indexed AM: Arrays → Randomly access data @ any index

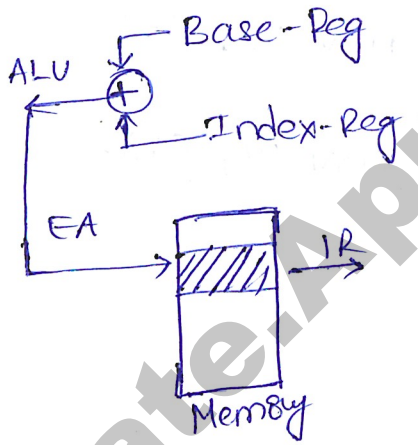


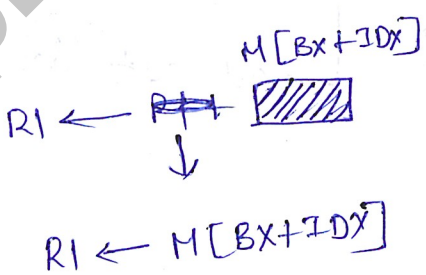

Eg: MOV R1, BX, IDX

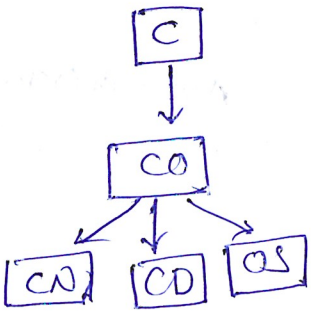
Reg AM

Base Addr → Base-addr-Reg

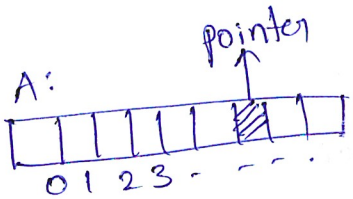
IDX: Index-Reg

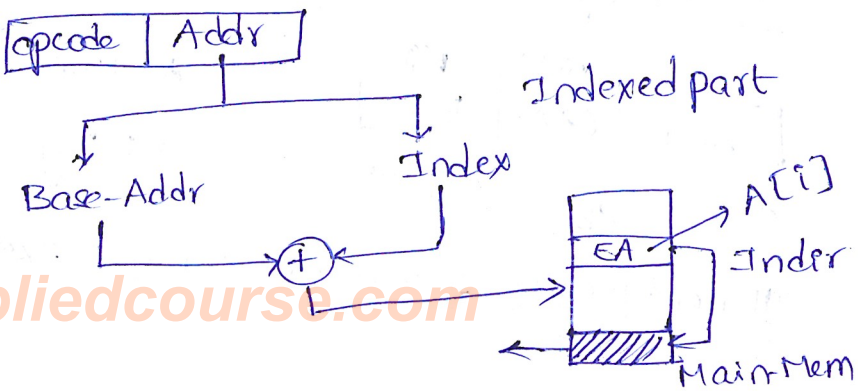



Indirect - Indexed AM:



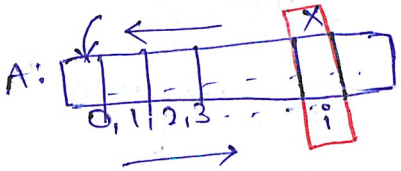



## Auto & Indexed AM:

Linear-array-elements + linear-access

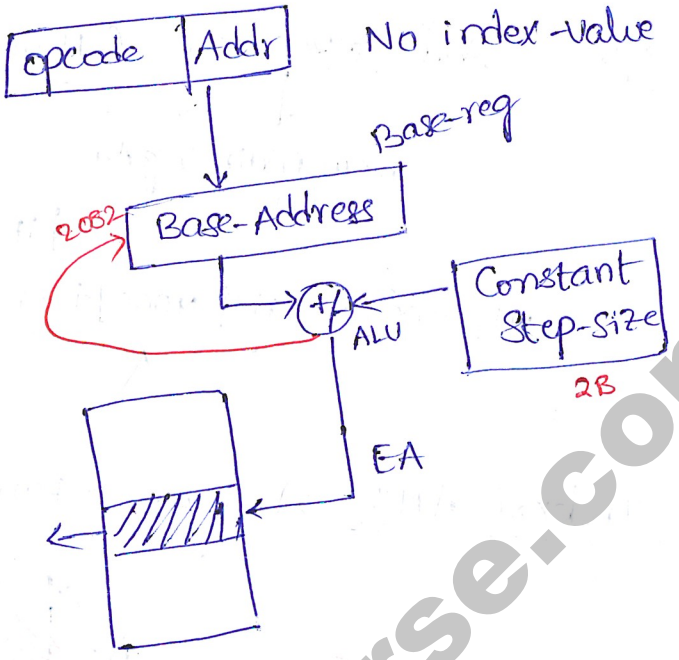
→ Auto-increment AM

→ Auto-Decrement AM




2080 + 2 Stepsize

⇒ 2082




Auto-increment AM
- Pre-auto incr — MOV r0, +(r1)
- (or)
- post auto incr — MOV r0, (r1)+

C: x++ vs ++x

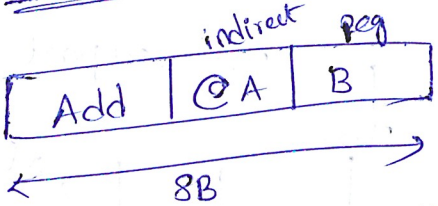
Auto-decrement AM
- Pre-auto decr — MOV r0, -(r1)
- post-auto decr — MOV r0, (r1)-

r1 : 2080

S.S : 2B

## More Solved problems:

(6) 32 bit up



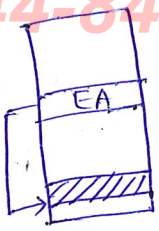

A: Mem-addr

B: reg-addr

# memory-cycles required during execution-cycle.

| Fetch | Execute |
|---|---|
| 2 Mem-R | ① 2 mem-R |
| | ② 1 reg-R |
| | ③ Arithmetic |
| | ④ 1 mem-R + 1 mem-W |

ID: 0 1-Reg R



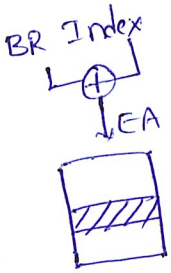

Q 3.2 GHz µP

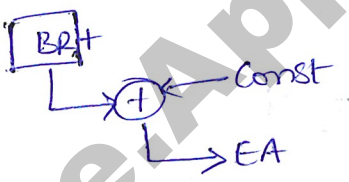
| AM | freq |
|---|---|
| Reg | 20% → 0 ⇒ 0 |
| Immediate | 18% → 0 ⇒ 0 |
| Direct | 22% → 8 ⇒ 0.22×8 |
| Mem-Indirect | 17% → 8+8 ⇒ 0.17×16 |
| Indexed | 11% → 4+8 ⇒ 0.11×12 |
| Auto-Indexed | 12% → 4+8 ⇒ 0.12×12 |

8-clock cycles: Mem-Reg
4: Arithmetic op
0: Register-operand

Avg-fetch-rate = ? (in W/S)






7.24 Cycles

7.24 × 0.3125

⇒ 2.26 ns

⇒ 1W

f 3.2 GHz

⇒ $t = \frac{1}{3.2} 10^{-9}$

= 0.3125 ns/cycle

Avg. fetch rate = ?

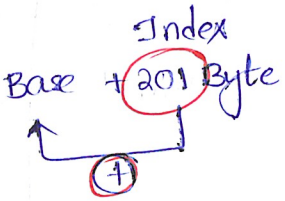
$= \frac{1}{2.26 \times 10^{-9}}$

= 442.4 MW/S

(03) Consider the c struct defined below:

```
struct data {
    int marks[100];     — 2B
    char grade;         → 1B
    int cnumber;        → 2B
};
struct data student;
```

Let




The base address of the student is available in register R1.

The field student.grade can be accessed efficiently using

✗ (A) post-increment addressing mode (R1)+

✗ (B) post-decrement addressing mode –(R1)

✗ (C) Register direct addressing mode R1

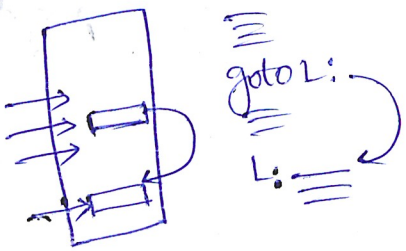
✓ (D) Index addressing mode, X(R1), where X is an offset represented in 2's Complement 16-bit representation.

Transfer of Control AM: Based & Relative

→ PC-relative AM

→ Based/ Base-reg AM

→ Jump/Branch: unconditional




→ Conditional Jump/Branch if ZF in psw=1

JZ addr

JNZ

for( . ; ▨ ; )
{
⋮
}

while(_)
{
≡
}
y

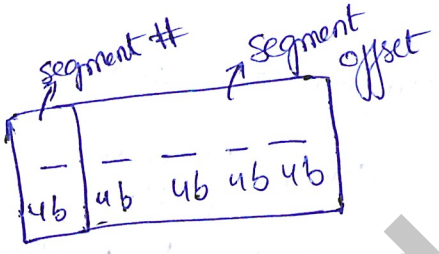
## Segmented-Memory:

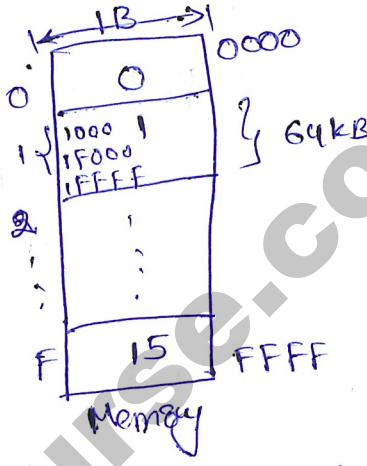
OS: Paging, Seg, VM

20 bit addr

$2^{20}$ calls = 1MB mem

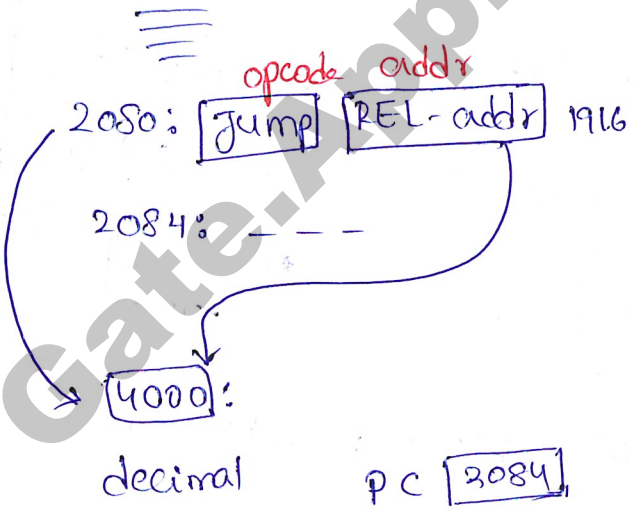
$\frac{2^{20}}{2^4} = 2^{16} = 64KB$




16-equal sized Segments

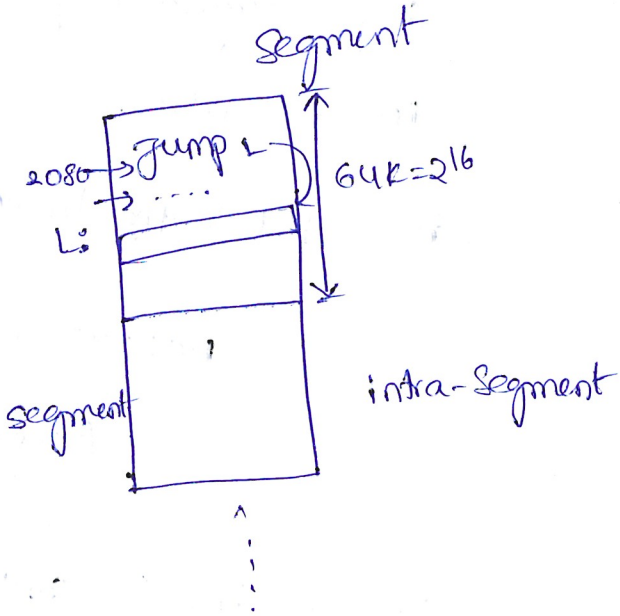
## PC-relative AM:




decimal PC 2084

4000-2084

= 1916



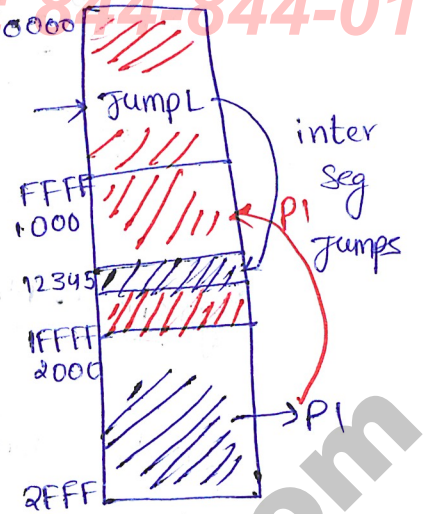
## Base-Addr Am: / Based-Am

EA = Base-Addr + addr-offset

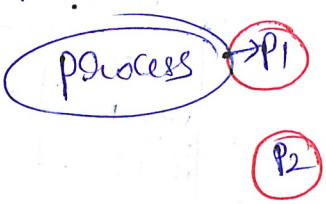
1000 2345

Jump Bx offset

jump R1 2345




## Dynamic program-Relocation

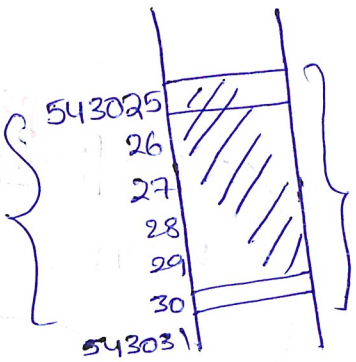



## Solved problems:

(Q) 6B long PC-relative jump-instr started at addr $(543025)_{10}$ onwards. JUMP Rel-addr (-48)

(-48) Signed displacement is present in the addr-field of the instruction.

(a) what is the branch-addr:




PC: 543031 + (-48)

= 542983

(b) If base-reg contains 683823; what's the branch address if base-reg addr is used.

jump R1 -48

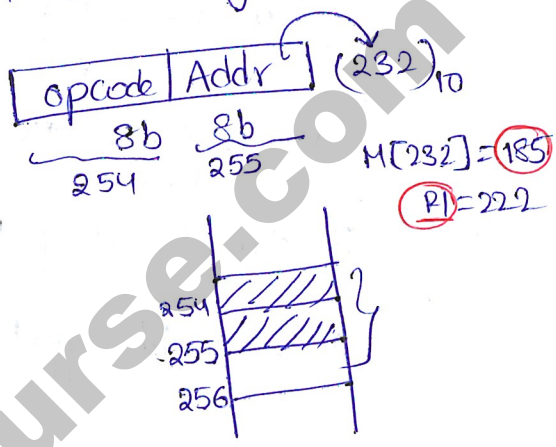
Base reg: 683823
+ -48

(c) based + Indexed Am is used with index-value = 32

branch-addr = ?

683823 + 32 - 48

(Q2) A 2B-instruction is present in the memory location starting at $(254)_{10}$.




(a) calc EA if immediate AM → 255

(b) Reg Am: Reg Address of R1

(c) Direct Am: 232

(d) Reg-indirect Am: 222

(e) Mem-indirect Am: 185

(f) Indexed-Am (R1: index-Reg): BA + RA
232 + 222 = 454

(g) PC-relative AM
Jump → PC: 256 + 232
= 488

(Q3) Consider a RISC machine where each machine 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 |
|---|---|
| 0-3 → i | add R2, R3, R4 |
| 4-7 → i+1 | sub R5, R6, R7 |
| 8-11 → i+2 | cmp R1, R9, R10 |
| 12-15 → i+3 | beq R1, offset |
| 16 → | |

beq: branch if equal

PC: 16 −16

If the target of the branch instruction is i, then the decimal value of the offset is −16

## Instruction Set: Introduction:

1. Data-Transfer : MOV, LOAD, STORE, PUSH, POP, IN, OUT etc.
2. Data Manipulation
3. Transfer of Control / program control
   - Jump
   - JZ
   - JNZ ...

### Data-Manipulation-instr: Data Sheet

1. Arithmatic instruction: ADD, MUL, SUB, DIV, INC, DEC etc

(a) INC $r_0$

$r_0$: 1111 1111 +1

0 ← CY: 0000 0000

(b) DEC $r_0$

$r_0$: 0000 0000 −1

CY: 1111 1111 CY: 1

CY bit does not change.

```
for(i=0; i<10; i++)
{
  .
  .
}
```

10000 0000
-1
01111 1111

② Logical Instructions : AND, OR, XOR, CMP etc.

x % y
MOD

ⓐ CMP $r_0$ $r_1$ $\quad r_0 - r_1 \rightarrow$ SUB (ALU)

$r_0 = r_1 \Rightarrow$ CY: 0 Z: 1

$r_0 > r_1 \Rightarrow$ CY: 0 Z: 0

$r_0 < r_1 \Rightarrow$ CY: 1 Z: 0

PSW

CY
1011 1011 $r_0$
1010 1010 $r_1$

CY 0 1010 ← $r_0$
−1011 ← $r_1$

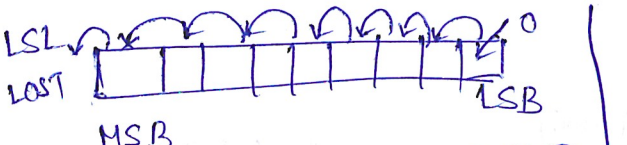
③ Shift-operations :-

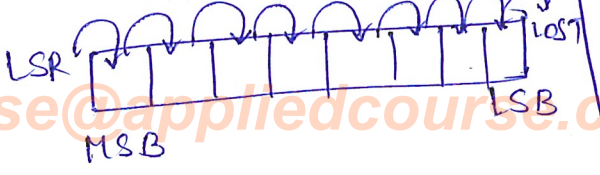
c

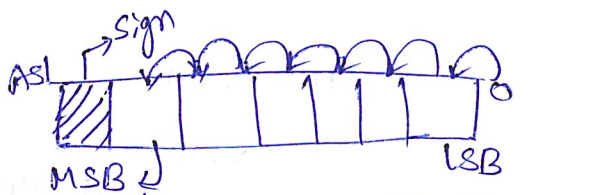
a<<2; a>>2;

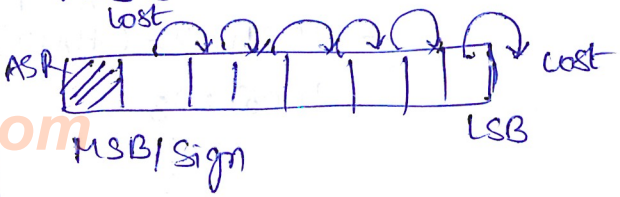
- Logical
  - LSL
  - LSR
- arithmetic
  - ASL
  - ASR

Signed: Numeri



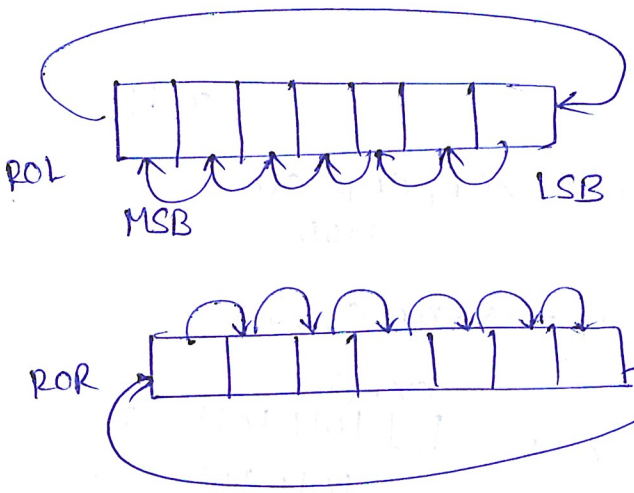



## (4) Rotate Instruction:

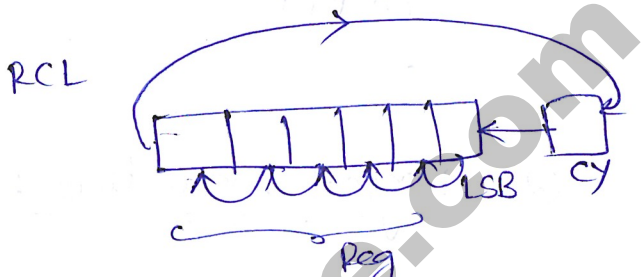
- Without carry
  - ROL
  - ROR
- With carry
  - RCL
  - RCR

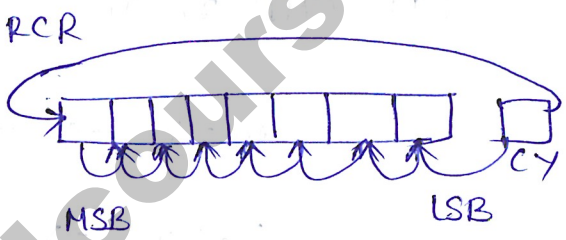
Rotate with Carry Right










## Transfer-of-Control / program-control Instructions:

- Unconditional TOC
  - Jump
  - HALT
- Conditional TOC
  - JNZ
  - JZ

{ sub-routines / sub-programs / functions

### (1) Jump / BRANCH

1000: $I_1$

1002: $I_2$

JUMP [2000]

→ 1004: ~~$I_3$ (HALT)~~ ~~PC: 1003~~ PC: 2000

1006: $I_3$

⋮

1006 : $I_3$

⋮

2000 : $I_4$

2002 : $I_5$

⋮ ⋮

② HALT ⟶ User view : Computer is Halted/Stopped RESTART

⟶ CPU-view ⟶ HALT ⇒ (PC = 1002)

HLT

1000 : $I_1$

1001 : $I_2$

→ 1002 : $I_3$ [CHALT] ← PC : 1003 X IR : [HALT]

PC : 1003 : $I_4$ ↓

⋮ PC : 1002

③ JZ & JNZ (Conditional Transfer of Control)

✓1000 : MOV $r_0$, $(03)_{16}$

✓ 1001 : $I_2$

✓ 1002 : DEC $r_0$

1004 : JNZ [100]

→1006 : $I_3$

1007 : $I_4$

⋮

$r_0$ : ~~2~~ ~~1~~ 0 ⟶ ACC & ALU

ZF : 1

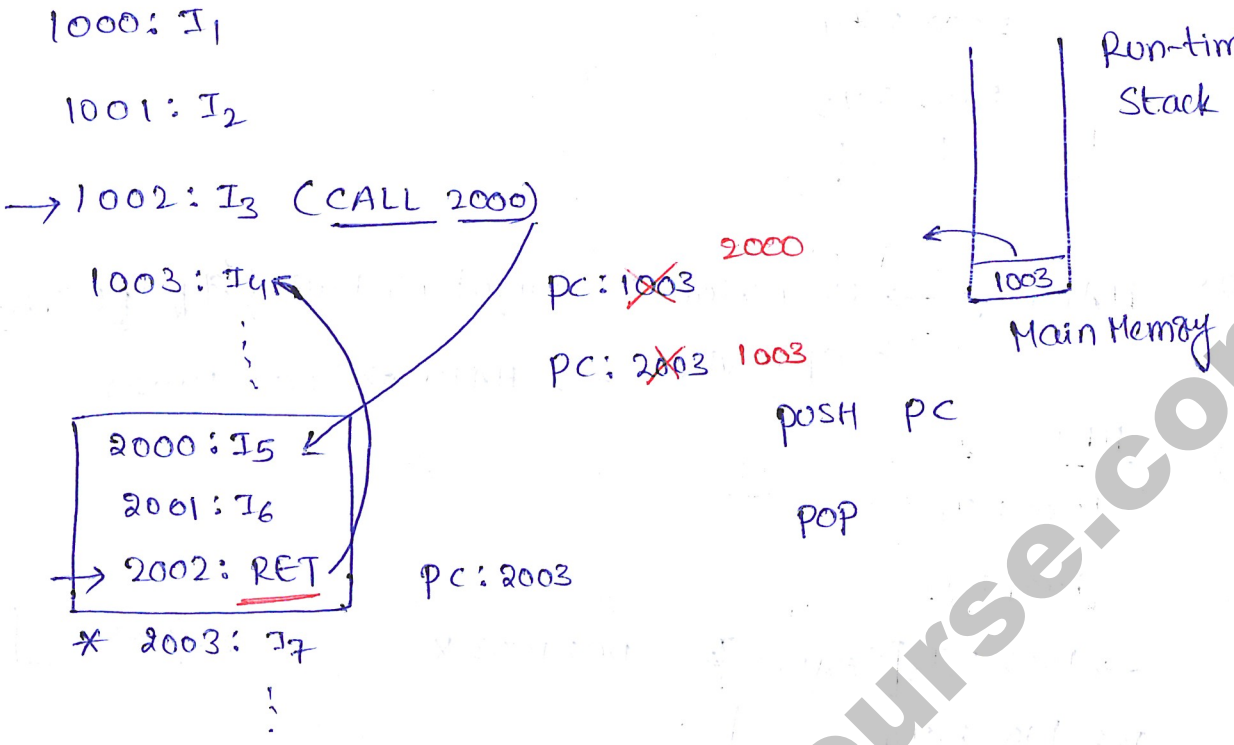
ZF : 1 if ACC = 0

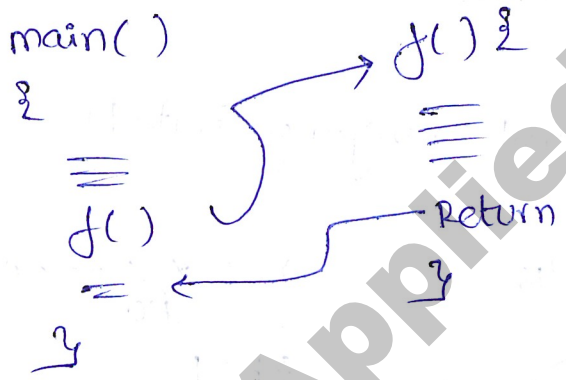
ZF : 1 if result of prev-instr is zero

# Sub-routines / Sub-programs using CALL & RET instr:



DS & Alg → Recursion




## Direct Vs Indirect Transfer of Control

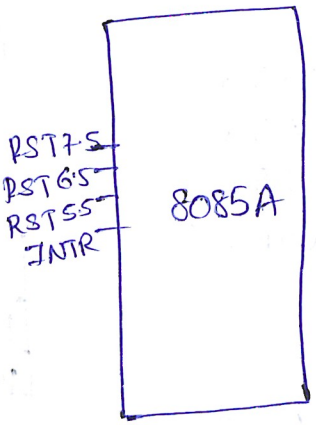
EA & Addressing Modes

- Unconditional
  - Jump addr
  - CALL addr
  - HALT
- Conditional
  - JNZ addr
  - JZ addr
  - CALLZ addr
  - CALLNZ addr

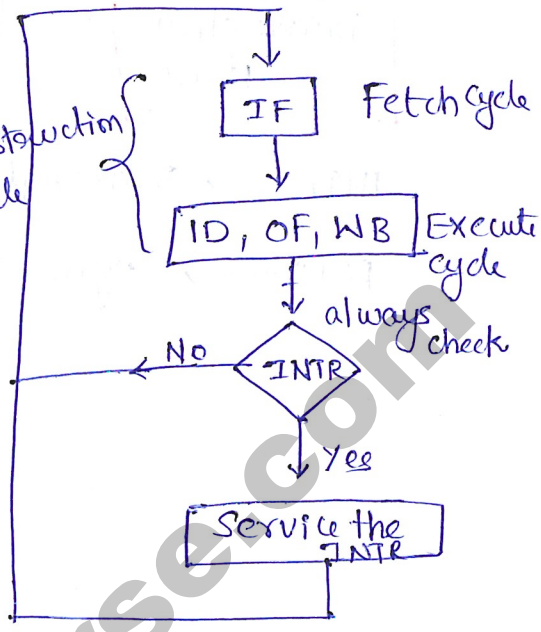
RET → unconditional

RET Z, RET NZ
RET C, RET NC
} Conditional RET

# Instruction-Cycle with interrupts + problems:




→ MOV $r_0$, [2080]
↕
→ ADD $r_0$, $r_1$




- RST 7.5
- RST 6.5
- RST 5.5
- INTR

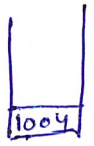
H/W interrupt pins

IOT

$\overline{INTA}$ - Interrupt Acknowledgement

Instruction Fetch
Instruction Decode
Operand Fetch
Write back

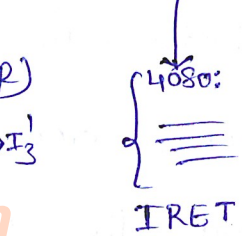
## Interrupt Cycle

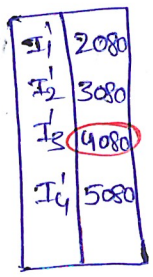



1. Check the INTR pins
2. Context-switch / Transfer of control using Interrupt vector table (IVT)
3. Interrupt sub-routine (ISR) ↳ $I_3'$
4. IRET / RETI
   Interrupt Return / Return Interrupt

1000: $I_1$
1002: $I_2$ ← INTR
1004: $I_3$ ← check INTR Pins
1005: $I_4$

$I_3'$



## Single-INTR vs Nested-INTR processing

Single-INTR → Disable other INTR.

INTR → Interrupt cycle

RST 5.5 → Ignore this

Priority based

INTR → IC → INTR

RST 5.5 → I.C 5.5 → INTR

- RST 7.5 ✓
- 6.5 ✓
- 5.5 ✓
- INTR ✓

IOT

Need to place the highest priority interrupt First

Even

PC INTR

PC original

(Q1)

| | Instruction | Meaning | Size (W) |
|---|---|---|---|
| $I_1$: | MOV r0, @3000 | $r_0 \leftarrow M[3000]$ | 2 |
| $I_2$: | MOV r1, @2000 | $r_1 \leftarrow M[2000]$ | 2 |
| $I_3$: | Add r0, r1 | $r_0 \leftarrow r_0 + r_1$ | 1 |
| $I_4$: | MUL r0, r1 | $r_0 \leftarrow r_0 * r_1$ | 1 |
| $I_5$: | SUB r0, [4000] | $r_0 \leftarrow r_0 - m[4000]$ | 2 |
| $I_6$: | MOV 3(r0), r1 | $M[30 + [r_0]] \leftarrow r_1$ | 2 |
| INT → $I_7$: | Halt | Stop | 1 |

1W = 4B

| Addr | IF & ID | OF & WB |
|---|---|---|
| 5230 | 16 | 16 |
| 5238 | 16 | 16 |
| 5246 | 8 | 2 |
| 5250 | 08 | 2 |
| 5254 | 16 | 2+8 |
| 5262 | 16 | 2+8 |
| 5270 | 8 | — |
| 5274 | 88 | 56 |

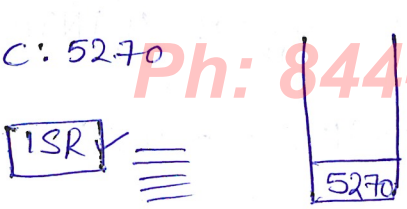
(a) Byte-address-Memory 32 bit MP

Starting-addr = $(5230)_{10}$

INTR occurs during execution of Halt

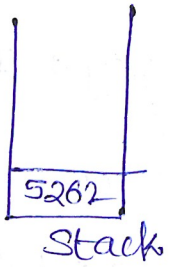
What's the return address pushed to the Stack?

HALT completes : PC: 5270




(b) INT occurs during execution of SUB, Return-addr pushed into Stack = ?

while executing IS
PC 5262
Contains




(c) µP : 2.5 GHz

IF & ID: 8 cycles / word

Cycle time = $\frac{1}{2.5\text{GHz}}$ = 0.4 ns

Arithmetic op?: 2 cycles

Memory : 8 cycles

Reg-reg : 0 cycles

execution time : = ?

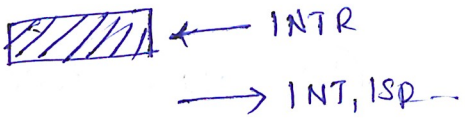
Total cycles = 144

⇒ 144 * 0.4 = 57.6 ns

(a) up that supports 4 INTR : $I_1, I_2, I_3, I_4$

ISR's service time : 20ns, 40ns, 16ns & 10ns

Response time of INTR : 5ns




priority : $I_1 > I_2 > I_3 > I_4$

Max & Min time required to service $I_4$ given nested-INT-processing.

If only $I_4$ Interrupt occurs then

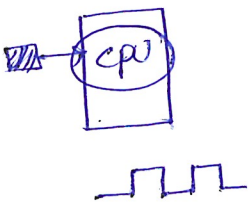
time to resolve $I_4$ : 5+10

Min Time = 15ns

Max time to resolve $I_4$ = 25+45+21+15 = 106ns

If high priority interrupts are coming frequently then serving of $I_4$ may lead to starvation.

## Types of Interrupts:




(1) Hardware : Pins

(a) External : I/O devices & Sensors

(b) Internal : clock-error, cpu-temp, invalid-opcode, div-by-zero, stack-overflow.

## ② Software Interrupts:

8085 RST 0
RST 1
RST 2
⋮
RST 7

IVT

| INTR | Addr |
|---|---|
| RST 0 | 00H |
| RST 1 | 08H |
| RST 2 | 10H |
| RST 3 | 18H |
| RST 4 | 20H |
| RST 5 | 28H |
| RST 6 | 30H |
| RST 7 | 38H |

Non-Maskable
Maskable
8085

## ③ Maskable & Non-Maskable INTR

Maskable (RST 7.5 6.5 5.5) — Ignore
Non-Maskable (TRAP) — cannot be ignored by the Micro processor

Eg: SIM : Set Interrupt Mask (Instruction)
RIM : Read Instruction Mask.

Zero-addr-instruction.

ACC

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| SCO | SDE | X | R7.5 | MSE | M7.5 | M6.5 | M5.5 |
| | | Don't care | | RST | | | |

8085

## ④ Vectored Vs Non-vectored Interrupt

8085 : INTR → INTA & wait for ISR-addr

7.5 0
6.5 1
5.5 2

| addr | |
|---|---|
| 0 | 2080 → ISR |
| 1 | 3080 |
| 2 | 4080 |
| 3 | 5080 |

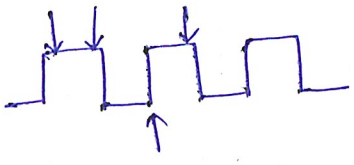
INTR# IVT

I/O → INTR
INTA
addr of ISR

(5) Level-Triggered

Elec-Engg


(6) Edge-Triggered



RISC (Reduced Instruction set Computer) (UCB, ...)

CISC (Complex Instruction set Computer)

CISC — 4 cycles

MUL [2080] [3080]

{ MUL r0, r1
, MUL r0, [2080]

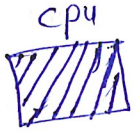
→ More # of instr ⇒ More Circuitary
→ fewer registers
→ Variable # cycle Instr
→ Pentium, i3, i5, i7, ... (Win)

RISC

1 LOAD R0, [2080]
1 LOAD R1, [2080]
1 MUL R0, R1
1 STORE [2080], R0

→ fewer # of instr ⇒ Simpler Circuitary
→ More # of reg → Pipelining
→ 1 cycle/instr
→ ARM Mobily, ipads, ...
→ Laptops

cpu → pipelining



Pipelining

Register-Organization in RISC:

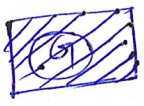
Total # of reg = $g + wxL + \frac{w(2*C)}{2}$

g: # of Global-reg

L: # of local reg / window

C: Common reg / window

w: # windows



Global Registers file


reg window

C: Common
L: Local


over Lapped reg windows

(Q1) Consider the following processor design characteristics.

I. Register-to-Register arithmetic operations only.

II. Fixed-length instruction format — 1 cycle/Instr

III. Hard wired control unit
↳ DLC

IV. which of the characteristic above are used in the design of a RISC processor?

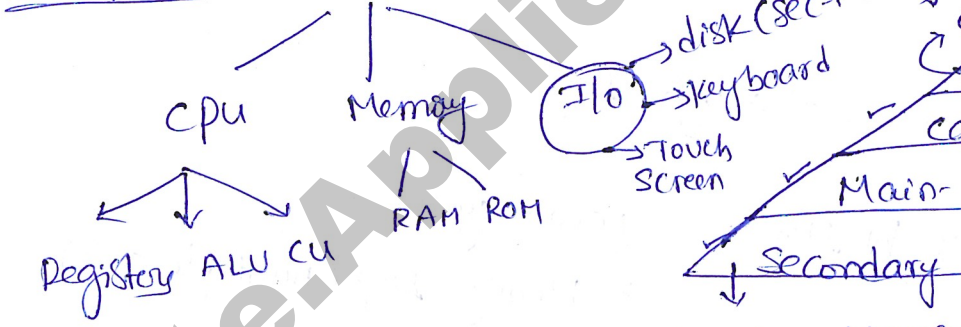
(A) I and II only

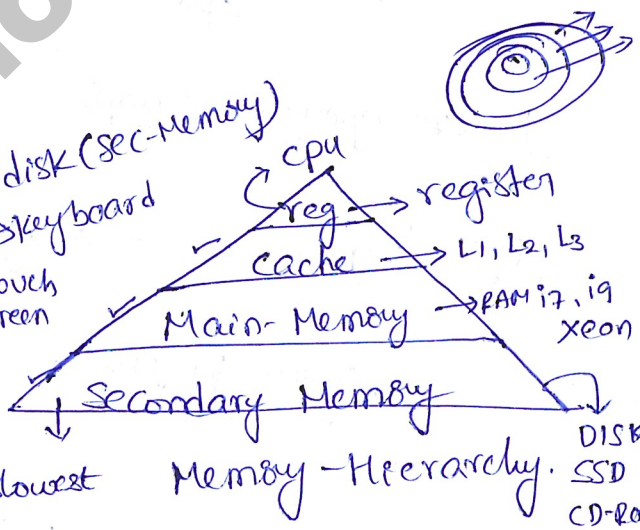
(B) II and III only

(C) I and III only

(D) I, II and III

## Components of a Computer:




```
for i=0 to 1000
{
:
:
}
```

CPU - 2.4 GHz
RAM: 144 MHz

① ↑ performance $\propto \frac{1}{\text{execution Time}}$ ↓

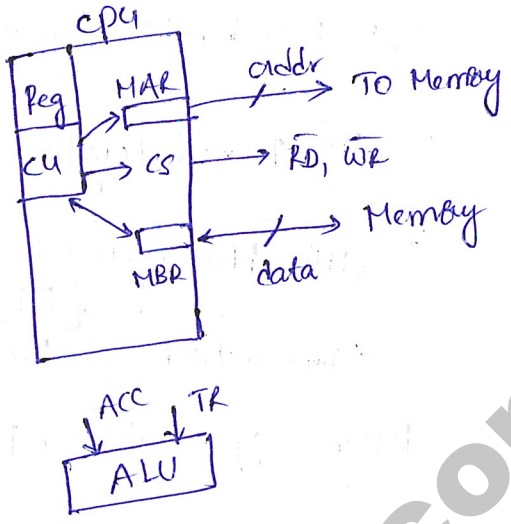
② ↓ execution time $\propto$ access-time ↓ LDA @2080 | MOV R0, R1

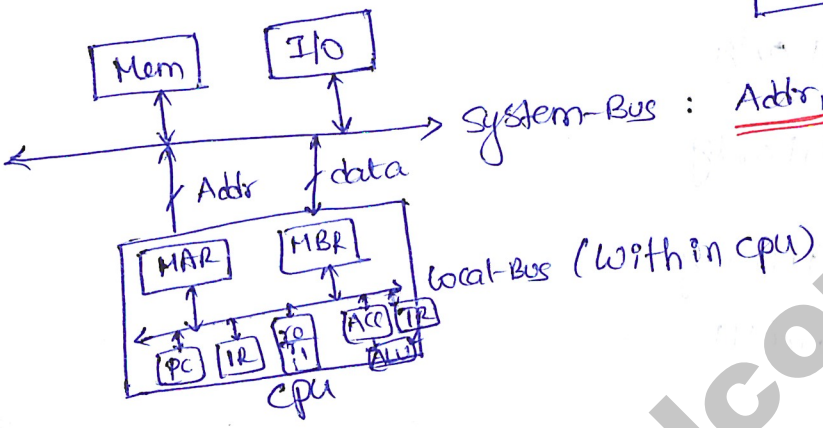
$\Rightarrow$ ↑ performance $\propto \frac{1}{\text{access-time}}$ ↓

**Registers:** PC, IR, ACC, TR, MAR, MBR/MDR, PSW, GPR

Memory-addr-Reg (MAR) : Stores the address temporarily

Memory Buffer/data Reg (MBR) ⇓ Stores the data temporarily.




**Micro-operation (μop)** — μp-instruction → μop1, μop2, μp3, ...

Eg: MOV R0, R1 → Single μop

→ cycles for μop → 1 clock cycle → Memory μop (multiple-cycles)

→ Control Signals : seq of CS

→ Control-Signals : ↑ CU

**Micro-program (μ-program) :** sequence of μ-operations

## ① Fetch Cycle:

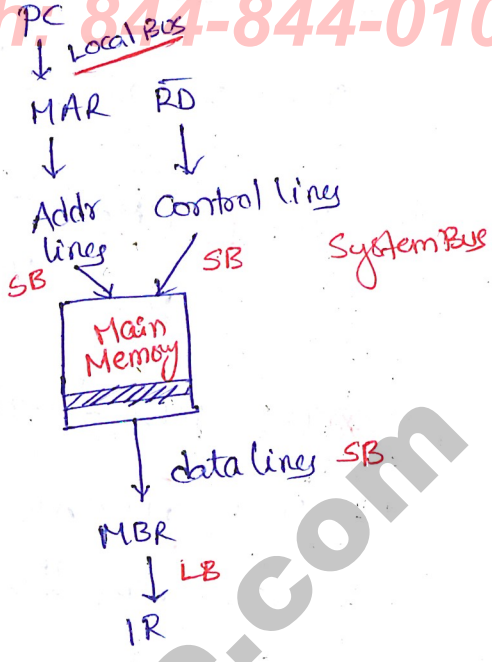
M-program:

$T_1$: PC → MAR (PC$_{OUT}$ & MAR$_{IN}$) → MOP → 1-cycle

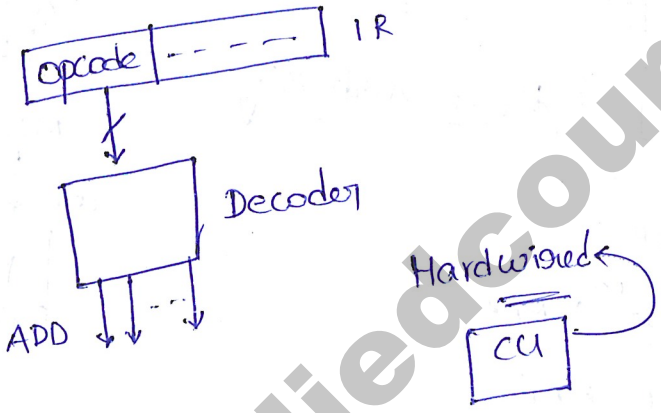
$T_2$: M[MAR] $\xrightarrow{SB}$ MBR → Multiple clock cycles

PC + I $\xrightarrow[ALU]{LB}$ PC (Local-Bus)

$T_3$: MBR $\xrightarrow{LB}$ IR → 1 cycle




## ② Execution-Cycle: Instruction-Decode




## ③ Execution-cycle: Operand Fetch (OF)

### ⑨ Direct Mode:

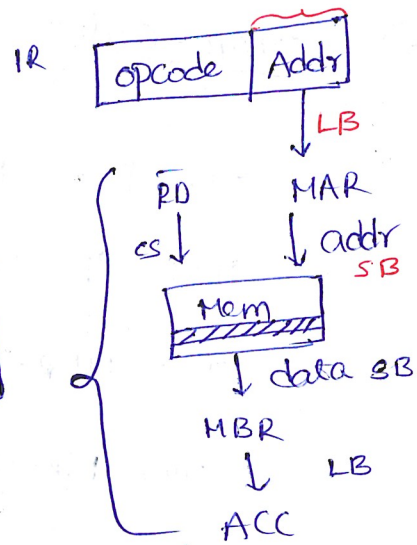
e.g) LDA [2080]

$T_1$: IR[addr] $\xrightarrow{LB}$ MAR

$T_2$: M[MAR] $\xrightarrow{SB}$ MBR (Multiple clock cycle)

PC+I $\xrightarrow{LB}$ PC

$T_3$: MBR $\xrightarrow{LB}$ ACC

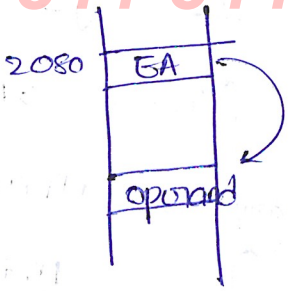


(b) Indirect-Mode: LDA @2080

T1: IR[addr] $\xrightarrow{LB}$ MAR

T2: M[MAR] $\xrightarrow{SB}$ MBR [2080]

PC+I $\xrightarrow{LB}$ PC



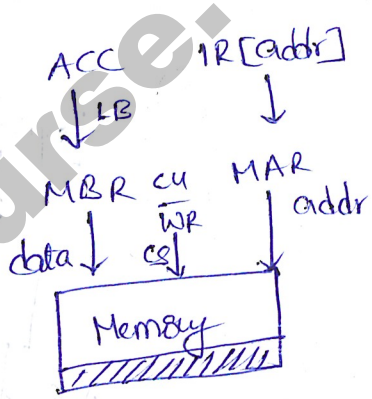

(*) T3: MBR $\xrightarrow{LB}$ MAR

T4: M[MAR] $\xrightarrow{SB}$ MBR operand

T5: MBR $\xrightarrow{LB}$ ACC

(4) Execution-stage: Write-back

Eg: STA [2000] direct-Addressing



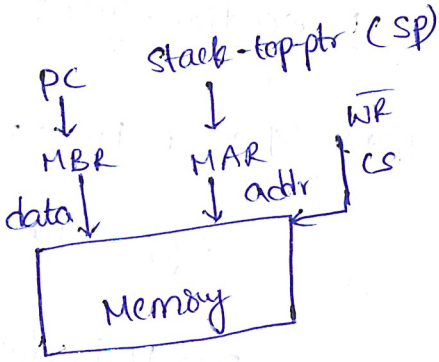

T1: ACC $\xrightarrow{LB}$ MBR

T2: IR[addr] $\xrightarrow{LB}$ MAR

T3: MBR $\xrightarrow{SB}$ M[MAR]

PC+I $\xrightarrow{LB}$ PC

(5) INTR-Cycle:




T1: PC → MBR

T2: SP → MAR

T3: MBR → M[MAR]

T4: ISR-addr → PC

ISR addr → PC

ADD $r_0$ $r_1$

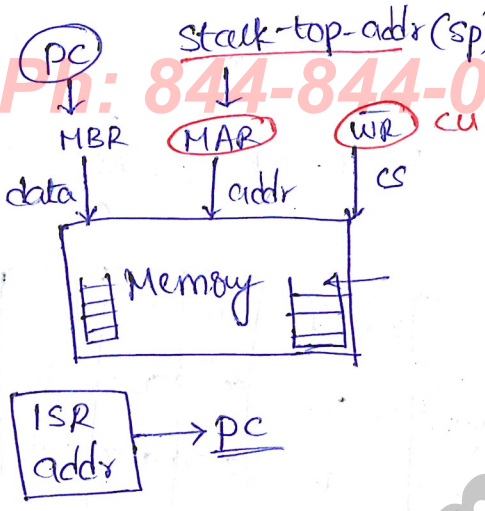
$r_0$ $\xrightarrow{LB}$ ACC

$r_1$ $\xrightarrow{LB}$ TR

ACC + TR → ACC

ACC $\xrightarrow{LB}$ $R_0$

⑤ INTR-Cycle:




✓ $T_1$: PC $\xleftarrow{LB}$ MBR

✓ $T_2$: SP $\xrightarrow{LB}$ MAR

$T_3$: MBR → M[MAR]

✓ $T_4$: ISR-addr → PC → IVT
MBR → PC

Q. which of the following operations is satisfied by the μ-prog

$T_1$: PC → MBR

$T_2$: X → MAR

$T_3$: MBR → memory

$T_4$: Y → PC

PC → Memory (Stack)

ⓐ IF ⓑ OF ⓒ Conditional Jump ⓓ INTR subprogram initialization.

× × ×

[CALL ... ret]

Control unit: (CU) ← Hardwired / Microprogrammed

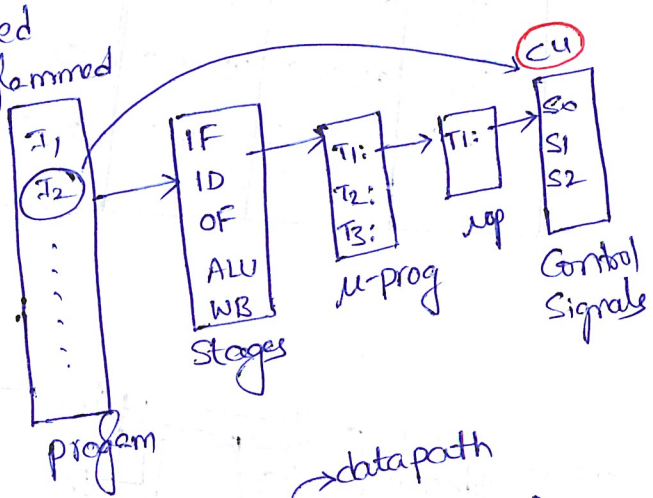
→ Control-signal-sequence

→ # Control signals

# instructions

Instr → μops

μop → control signals




(eg) PC $\xrightarrow{LB}$ MAR (μop) ← datapath

↳ S0: copy value from PC to LB

↳ S1: copy from LB to MAR

Eg) CS = {S0, S1, - - - - S7}

Instr-set = { ADD, SUB, MUL }
$I_1$ $I_2$ $I_3$
00 01 10

CU

| Mops | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|
| $T_1$ | S0, S3 | S1, S7 | S6 |
| $T_2$ | S1, S4 | S0, S5 | S1, S2 |
| $T_3$ | S0, S6 | S1, S2, S3 | S6, S7, S3 |

Mop

Control Signal - DB

$I_1 \rightarrow T_1$ :

$T_2$ :

$T_3$ :

Designing CU

## Hardwired Approach:

→ DLC using sum-of-products - expressions (OR + AND)

→ fast & real-time-applications

→ Inflexible

→ RISC

| Mop | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|
| $T_1$ | S0, S2, S3 | S1, S0, S3 | S2, S3, S1 |
| $T_2$ | S1, S0, S2 | S2, S1, S3 | S2, S1 |
| $T_3$ | S0, S2, S3 | S2, S1 | S0, S2, S3 |

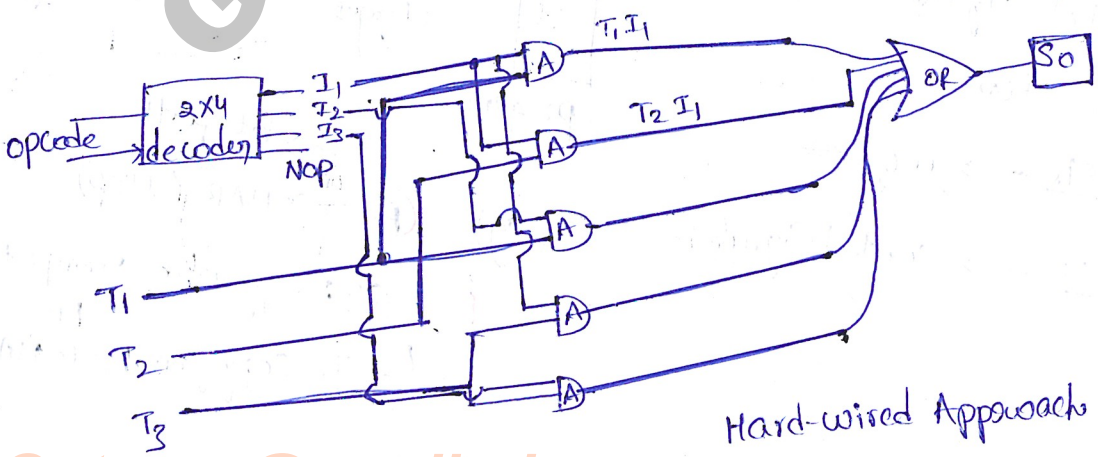
$S_0 = T_1(I_1 + I_2) + T_2(I_1) + T_3(I_1 + I_3)$

SOP

$S_1 = \ldots = T_1 I_1 + T_1 I_2 + T_2 I_1 + T_3 I_1 + T_3 I_3$

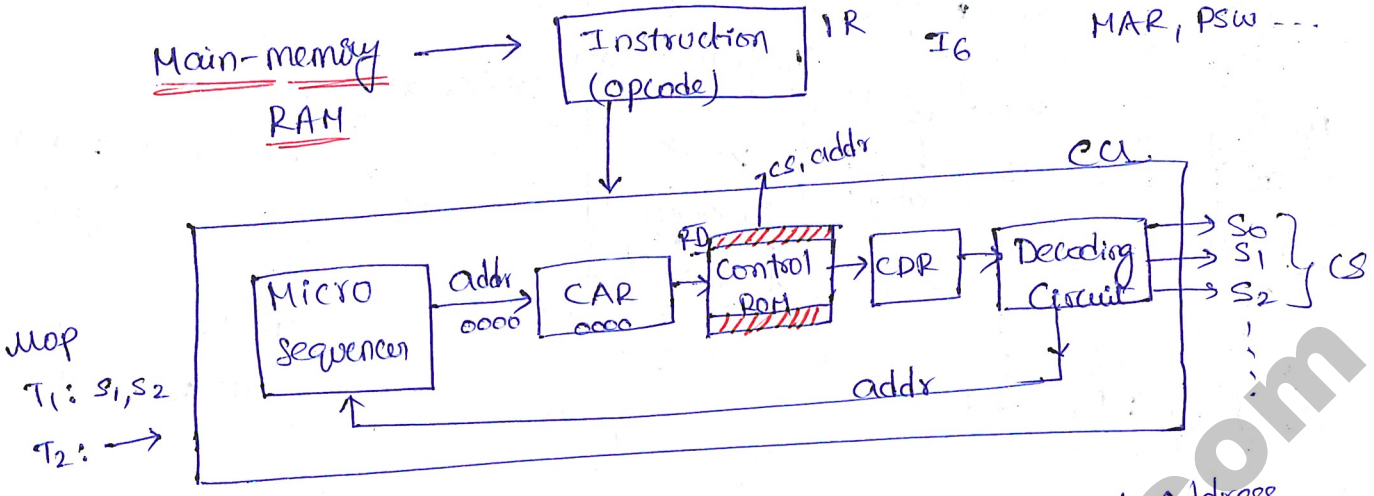
$S_2 = \ldots$

$S_3 = \ldots$




Hard-wired Approach

# Micro-programmed Control unit: μ-op, μ-instruction




Micro instructions / Control word

CAR: Control Address Register

ROM: Read only Memory

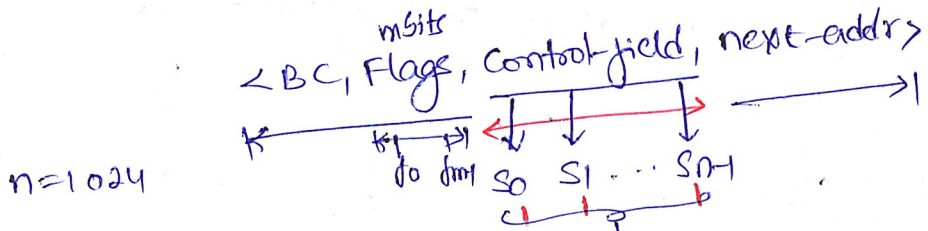
CDR: Control Data Register

## Control word / μ-instruction format:

| | (U, I, N) Branch condition | carry, sign, Z Flags | Control field $s_0 s_1 s_2 \cdots s_{n-1}$ | next-Control mem-addr (Control ROM) |
|---|---|---|---|---|
| 0000 | U | 101 | 1 0 1 ··· 1 | 0006 |
| 0001 | N | 010 | 0 1 0 0 ··· 1 | 0002 |
| 0002 | Z | 101 | 1 0 1 1 ··· 1 | 0010 |

Control-word / Instruction

## Horizontal-μinstruction:




n = 1024

CROM size will become huge

CDR → decoding-Circuit → $s_0, s_1, \cdots s_{n-1}$

n-CS

n-bits

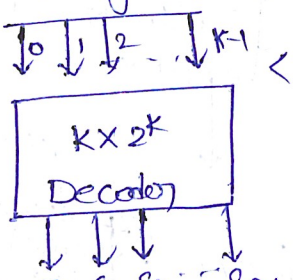
(decoded-form)

## Vertical Instruction:

<BC, Flags, controlfield, next-addr>

m flags ⇓ $\log_2 m$

$\log_2 m$ → decoder




<encoded-forms>

$K = \log_2 n$

n = # Control Signals

$n = 1024 = 2^{10}$

$m = 100 < 2^7$

0 1 0

n signals ↓

→ Flags : Encoded

| Horizontal | Vertical |
|---|---|
| → Longer CW ⇒ longer CROM | shorter CW = small CROM |
| → No decoder ⇒ Faster | Decoder ⇒ Slower |

CROM → CROM + decoder

Programmable ROM

PROM → PROM + decoder

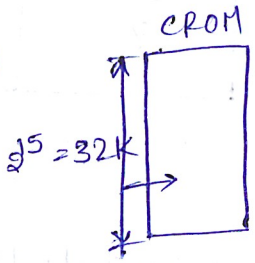
Speed : Hardwired > Horizontal > Vertical

Flexibility : Hardwired < Horizontal < vertical

Eg 1: CU with 32K control-word ROM

64 control-signals

8 flags




Horizontal CW: BC, flags, control field, next-addr in ROM

$1 \times 8 \times 64 \times 15 = 82$ bits

Virtual CW: $1 \times 3 \times 6 \times 15 = 24$ bits

## More Solved problems:

(Q1) 6-groups of mutually exclosive cs.

G1: 1
G2: 8
G3: 15
G4: 23
G5: 14
G6: 30

3X8

CF

(a) # bits in Control field (Horizontal)

1+8+15+23+14+30 = 91 bits.

(b) vertical

| 1 | 8 | 15 | 23 | 14 | 30 |
|---|---|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| 1 | 3 | 4 | 5 | 4 | 5 |

= 22 bits

(c) bits-saved 91-22 = 69

(Q) Two groups of cs: G1 & G2

G1 → 256 (one@a time) → vertical

G2 → (Multiple-Signals) → @time → horizontal

# bits in Control Field = 8 + 6 = 146

(Q) 256 instructions ⇒ 256X16 ⇒ $2^8 \times 2^4 = 2^{12}$

$T_1, T_2 \ldots T_{16}$ 16 cycle/Instruction → uop

Horizontal-CW

240-CS

16-flags

16-BC

Size of CAR & CDR = ?

12 bits 284 bits.

$2^{12}$ W

| BC | Flags | CS addr |
|---|---|---|
| 16 | 16 | 240 \| 12 |

16+16+240+12

= 284

32
12
240
284

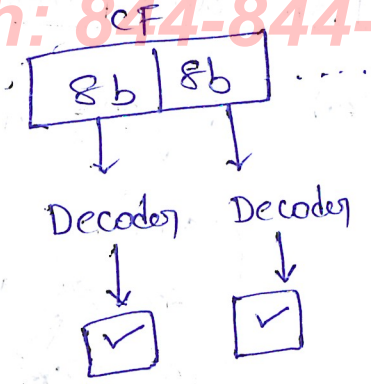
Q) default: Vertical

256-CS → CF

Each-Mop has 2 CS-active

8 bits # bits needed for CF: ?

16 bit




## Performance Evaluation: Pipelining

Performance (P) → Execution time (ET) $P \propto \frac{1}{ET}$

→ Throughput → OS $P \propto$ Throughput

Throughput = #jobs/unit-time (multi-programmed)

X → 2ns

Y → 6ns

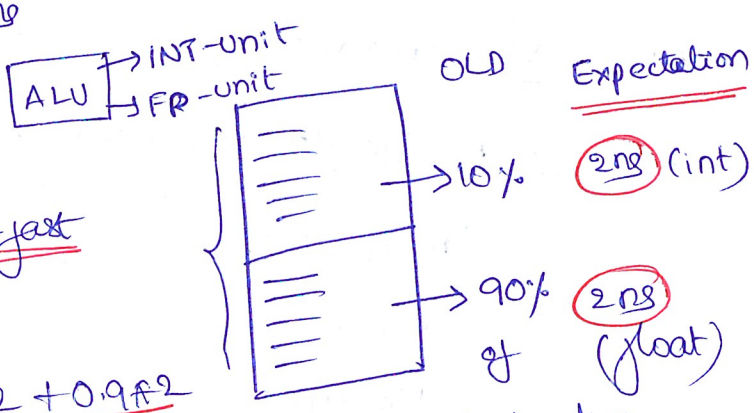
### Speed-up-factor (S):

$$S = \frac{P_x}{P_y} = \frac{1/ET_x}{1/ET_y} = \frac{ET_y}{ET_x} = \frac{6}{2} = 3$$

How fast is X (2ns) as compared to Y (6ns)?

### Amdhal's Law:

↳ Common-case-fast

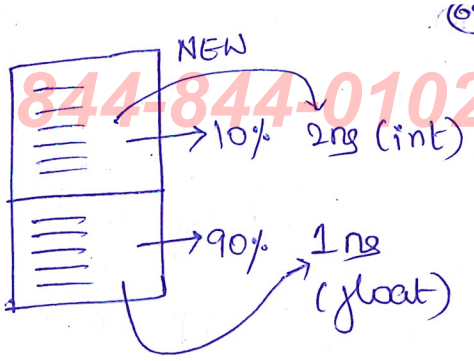



$$E(ET_{old}) = 0.1 * 2 + 0.9 * 2$$

$$= 2ns$$

$E(ET_{new}) = 0.1 * 2 + 0.9 * 1$

$= 1.1$ ns




$S_{overall} = \frac{2}{1.1}$

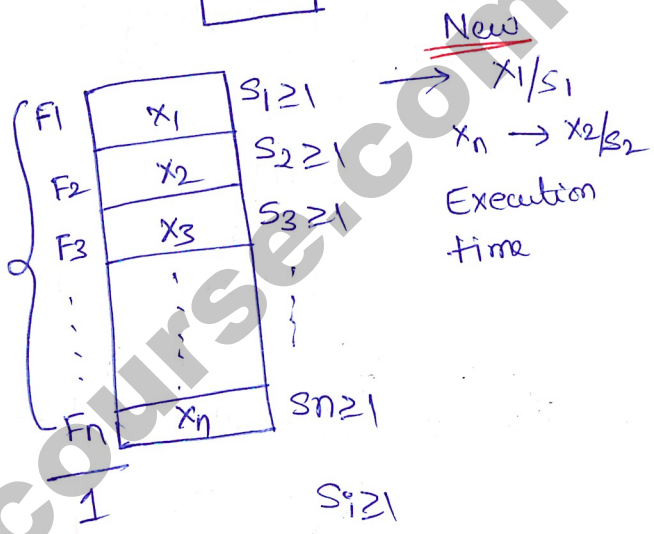
## Generalization:

$$E(ET_{old}) = \sum_{i=1}^{n} X_i F_i$$

$$E(ET_{new}) = \sum_{i=1}^{n} F_i (X_i / S_i)$$

$$S_{overall} = \frac{ET_{old}}{ET_{new}}$$

$S \geq 1$



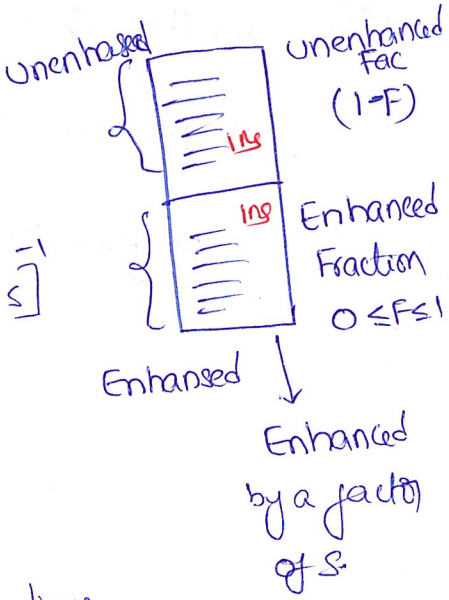

## Special-Case:

$$S_{overall} = \frac{E(ET_{old})}{E(ET_{new})} = \frac{1}{(1-F) + F/S}$$

$$= [(1-F) + F/S]^{-1}$$

$(1-F) * 1$

$+ \frac{1}{S} * F$




{ Assumption: Each instruction takes same time }

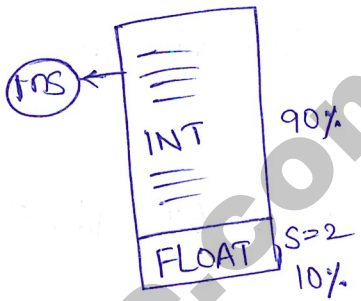
(Q) up for scientific-applications

Integer and floating point instructions.

FP-unit is enhanced & It runs 2X faster

10% of instructions in program are FP-operations

performance-gain = Soverall = ?




ETold = 1 ns

$ET_{new} = 0.9 + \frac{0.1}{2}$

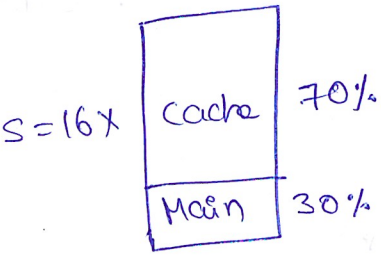
$S_{overall} = \frac{1}{0.9+0.05}$

$= \boxed{1.05}$

(Q) Cache is 16X faster than main-memory

cpu-refers to cache 70% of times

performance-gain = Soverall = ?



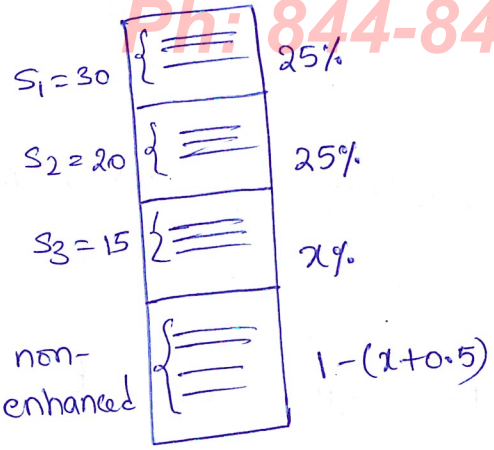

$= \frac{1}{(0.3*1 + 0.7 \times \frac{1}{16})}$

$= 2.9$

Q 3-enhancements:

Soverall = 10

Find x%




Eold = 1 ns

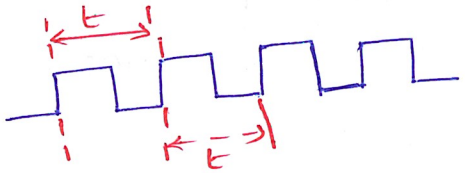
$$E_{new} = 0.25 \times \frac{1}{30} + \frac{0.25}{20} + \frac{x}{15} + (0.5 - x)$$

Soverall = 10

$$\Rightarrow \frac{1}{\frac{0.25}{30} + \frac{0.25}{20} + \frac{x}{15} + (0.5 - x)} = 10$$

$$\Rightarrow \boxed{x = 0.45}$$

## Calculation of cpu-time:




→ processor-clock

→ cycle-Time (t) = $\frac{1}{f}$

→ Constant frequency (f) = $\frac{1}{t}$

cpu 3.2 GHz

1 cycle → t sec

$\frac{1}{t}$ Hz ← 1 sec

cpu-time: IC * CPI * t

IC: Instruction-Count

CPI: Cycles-per-instruction

# instr * $\frac{\text{Cycles}}{\text{instr}}$ * $\frac{\text{Time}}{\text{cycles}}$

Different Instruction types: data-transfer, data-manipulation, Transfer-of-control ....

$$\text{CPU-time} = \sum_{i=1}^{k} IC_i * CPI_i * t$$

(a) $f = 3.2\text{GHz}$ μp $t = \frac{1}{3.2 \times 10^9} = 0.31\text{ns}$

| Instr-Type | IC | CPI |
|---|---|---|
| Load | 200 | 12 |
| Store | 300 | 10 |
| ARITH | 200 | 8 |
| Logical | 100 | 6 |
| shift | 150 | 4 |
| Branch | 50 | 2 |
| | 1000 | |

(a) Avg-instr-EI $= 8.3 \times 0.31\text{ns} = 2.57\text{ns}$

$0.2 \times 12 + 0.3 \times 10 + 0.2 * 8 + 0.1 \times 6 + 0.15 \times 4 + 0.05 \times 2$

$= 8.3$ cycle/instr

avg cycles per instruction

(b) MIPS-rate: avg 1 instr $\longrightarrow$ 2.57ns

$\leftarrow$ (15)

MIPS Millions of Instructions per Second

$0.3861 \times 10^9 \Leftarrow \frac{1}{2.57 \times 10^{-9}}$

$\Rightarrow 386.1 \times 10^6$

$\Rightarrow$ 386 MIPS

(c) program ET

⇒ 1000 * 2.57 ns

⇒ $2.57 * 10^{-6}$ s

(a)

cpu-1

t = 4ns (Cycle Time)

Load - 8 cycles
STORE - 8 } - 40%

ALU - 6 → 40%

Branch - 4 → 20%

avg. instr ETold = (0.4*8 + 0.4*6 + 0.2*4) *4

= 25.6 ns

Speed up = ?

cpu-2

t = 4.8 ns

CPI = 1

avg-Instr ETnew = [0.4*1 + 0.4*1 + 0.2*1] *4.8

= 4.8 ns

$\frac{25.6}{4.8}$ = 5.33

Flynn's cpu classification :- performance → ET, Throughput

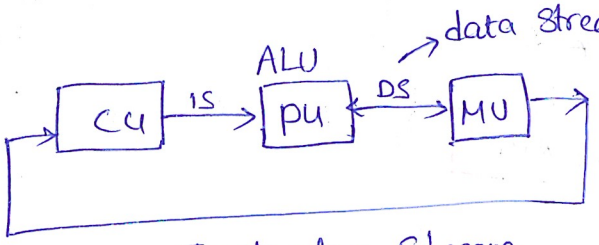
RISC vs CISC

Concurrency: 2 or more events @ same-time.

Event: program / sub-program / instruction / stage of an instruction.

SISD, SIMD, MISD, MIMD

## ① SISD: Single Instruction stream & Single data stream.



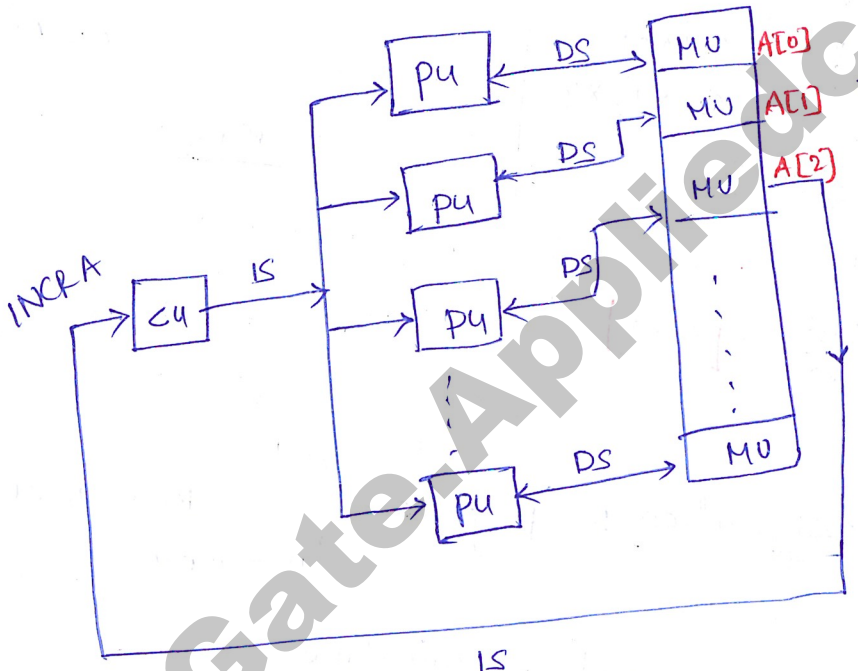

ALU: PU
MU: Reg / Cache / Main-mem

$X = X + 5;$

Uni-processor System

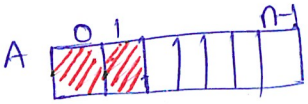

CU: Control unit
PU: processor unit
MU: Memory unit

## ② SIMD: Single Instruction multiple data.




→ Scientific-Applications

Eg: Add 1 to all elements of an array

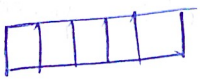



→ vector-processor

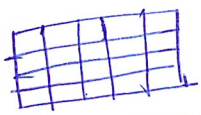
→ GPU (Graphical processing unit)

AI & ML (DL)

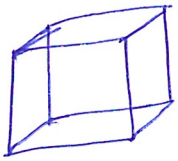
Ardeno 90's
Nvidia
AMD



Array



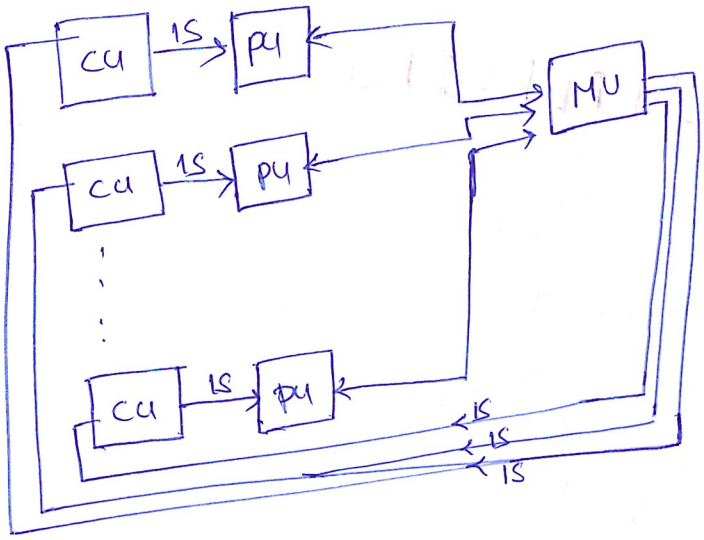
Matrix



Tensor (Tensorflow)

(3) MISD: Multiple Instructions single data.

Not very popular.




uniprocessor
↑
SISD
SIMD
MISD
MIMD
↓
Multiprocessor

x: Variable

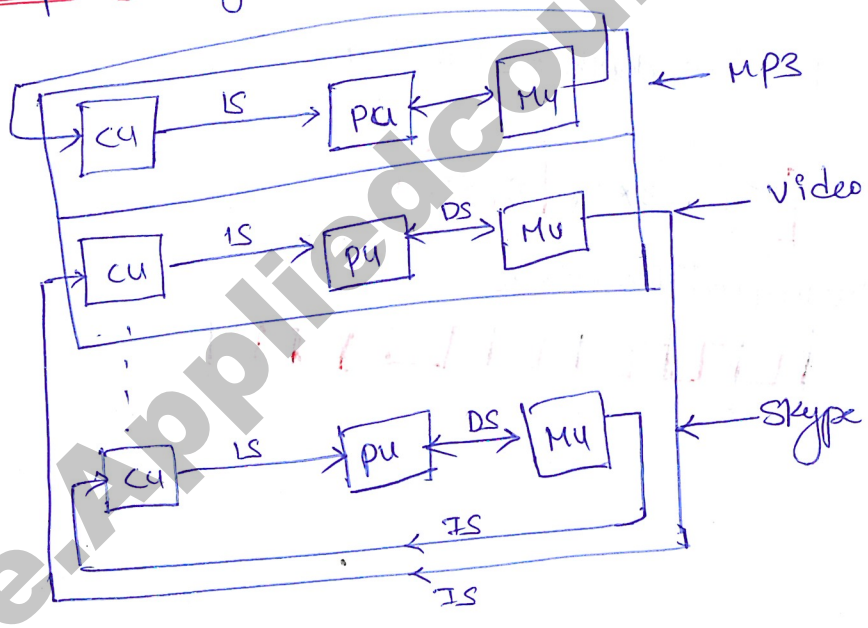
$y = x^2$

$z = \sqrt{x}$

$k = x^{1/3}$

$m = x/\sqrt{2}$

(4) MIMD: Multi-processor-System



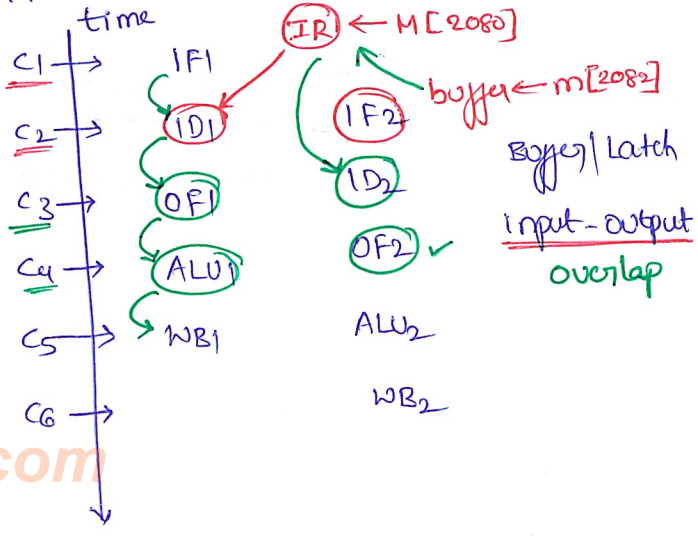

General purpose

i7, i5, Qualcomm Snapdragon

Pipelining: A Concurrency Approach → 2 or more Events @ same time

$I_1$: [2080] IF1 $S_1$, ID1 $S_2$, OF1 $S_3$, ALU1 $S_4$, WB1 $S_5$

$I_2$: [2082] $IF_2$, $ID_2$, $OF_2$, $ALU_2$, $WB_2$



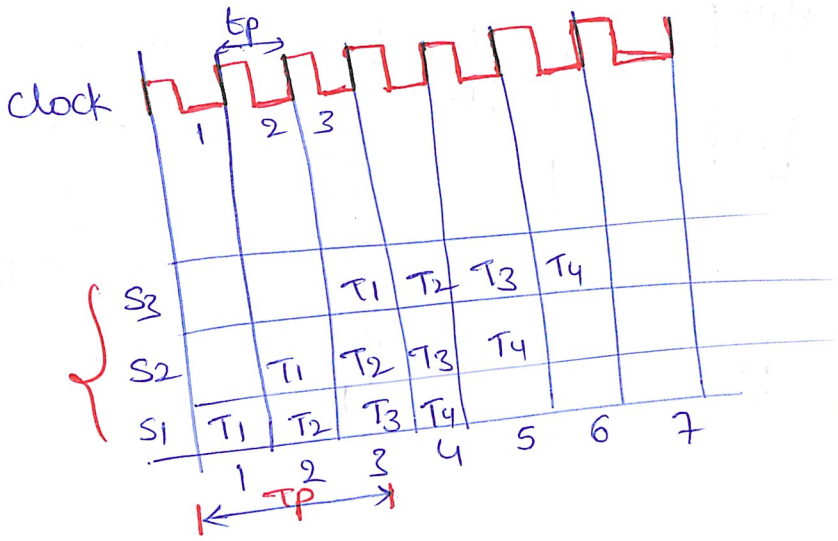

Buffer/Latch

input-output overlap

IR ← m[2080]



## Pipelined - Overlapped Execution:

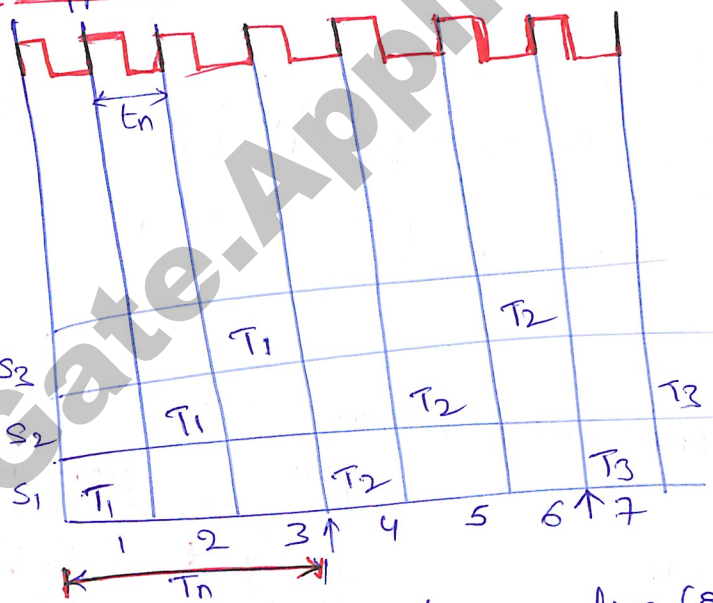



$T_i$ : Task$_i$ / instruction$_i$

pipeline-depth = 3 = #Stages.

$T_p = 3 * t_p \Rightarrow (Kt_p)$

## Non-Overlapped Execution:




Non-overlapped execution (or) non-pipelined

## Performance of a pipeline:

# Segments = K

Clock-cycle-time = $t_p, t_n$

# Tasks (or) instructions = n

→ Each Segment takes one-cycle

$ET_{pipe} = (K + (n-1))t_p$

$ET_{non\text{-}pipe} = n(t_n * k)$

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

$S = \boxed{\frac{n \cdot T_n}{T_p + (n-1)t_p}}$

Case: $n \to \infty$, $S = \frac{nT_n}{nt_p - t_p + Kt_p}$

$= \frac{nT_n}{nt_p + (k-1)t_p}$

K-Small << n is very Large

$\Rightarrow \boxed{S = T_n/t_p = \frac{kt_n}{t_p}} \approx K$, if $(t_n \approx t_p)$

## Throughput of a pipeline

$= \frac{\#\text{ tasks - completed}}{\text{time - taken}}$

n tasks $\longrightarrow (K+n-1)t_p$ sec

$\boxed{\frac{n}{(K+n-1)t_p} \leftarrow 1 \text{sec}}$

## Efficiency of appeline: $\frac{S}{K} = \eta$

(Q) Instruction-pipeline with S=12 and operating at 70% efficiency # stages =?

$$\eta = 70\%$$

$$= \frac{S}{K}$$

$$= \frac{12}{K}$$

$$= 0.7K = 12$$

$$\Rightarrow K = \frac{12}{0.7} = 17.1 \Rightarrow \boxed{18}$$

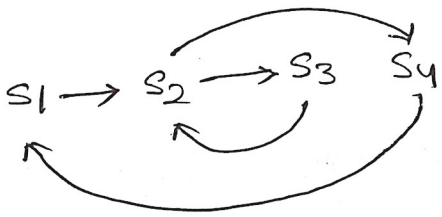
## Types of Pipelining:

### (1) Instruction/Linear pipeline

S1 → S2 → S3 → S4 → S5
IF ID OF ALU WB

forward-direction

### (2) Non-Linear pipeline




Tasks: Segments/Stages

### (3) Synchronous:

pipe-line operation is controlled by the clock-signal

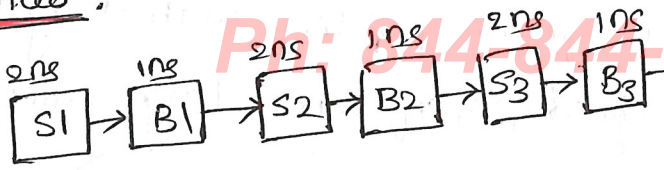
↳ All linear pipelining are Synchronous

### (4) Asynchronous pipeline:

operations are controlled based on Handshaking signals.

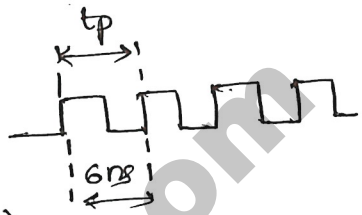
↳ All Non-linear pipelining are asynchronous.

⑤ uniform-delay / balanced :



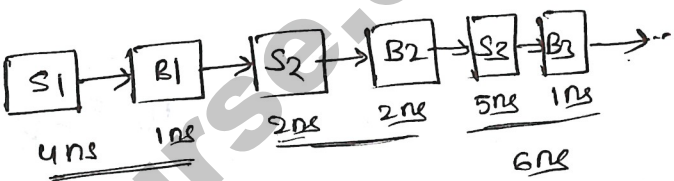

$t_p$ = ( Stage-delay + Buffer-delay + overhead)
$t_{IS}$

⑥ Non-uniform delay / Imbalanced :




$t_p = \max(SD_i + BD_i + O_i)$

clock time




⑦ Pipeline-A

$K_1 = 8$

uniform-delay = 2ns

Pipeline-B

$K_2 = 6$

Stage-wise delays : 2, 6, 4, 1, 3, 2 ns

Time saved when 100-tasks are pipelined through A instead of B.

ETA = (8+99)2 = 214ns

= (K+n-1)tp

(6+99)*6 = 630ns

⇒ 630 − 214 = 416ns

(Q2) 4-segment pipeline with delays: 20 ns, 30 ns, 40 ns & 15 ns
Speedup & $\eta$ when Large-number of tasks are Executed.

$$S = \frac{T_n}{t_p} = \frac{\text{Total time for non-pipelined execution}}{\text{clock time for pipelined Execution}}$$

$$= \frac{105}{40} = 2.625$$

$$\eta = \frac{S}{K} = \frac{2.625}{4} * 100$$

$$= \boxed{65.6\%}$$

(Q3) 4-stage pipeline with stage-delays: 30ns, 40ns, 20ns & 10 ns. Interface-register between the stages has a delay of 5ns. performance-gain of the pipeline?

Pipelined →

MAX (30, 40, 20, 10)
MAX (35, 45, 25, 15)

let n = Large.

$$S = \frac{T_n}{t_p} = \frac{100}{45}$$

$$= 2.22$$

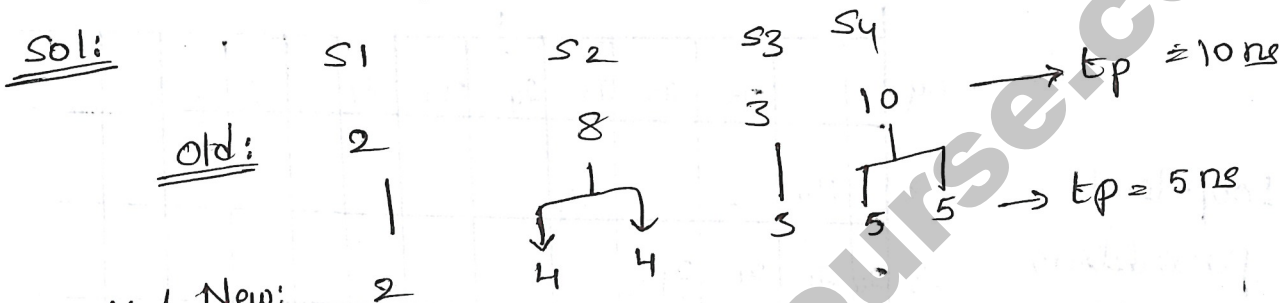
$$\eta = \frac{S}{K} = \frac{2.22}{K=4} * 100$$

$$= \boxed{55.5\%}$$

(Q4) 4-segment pipeline with stage-delays: 2ns, 8ns, 5ns, 10 ns. An Alternative pipeline contains same stages with same delays but stage 2 is divided into 2 sub-stages of equal-delays & stage 4 is divided into 2 sub-stages of same delay. Speedup of alternative pipeline over the other one.

let $\boxed{B = 0ns}$

Sol:



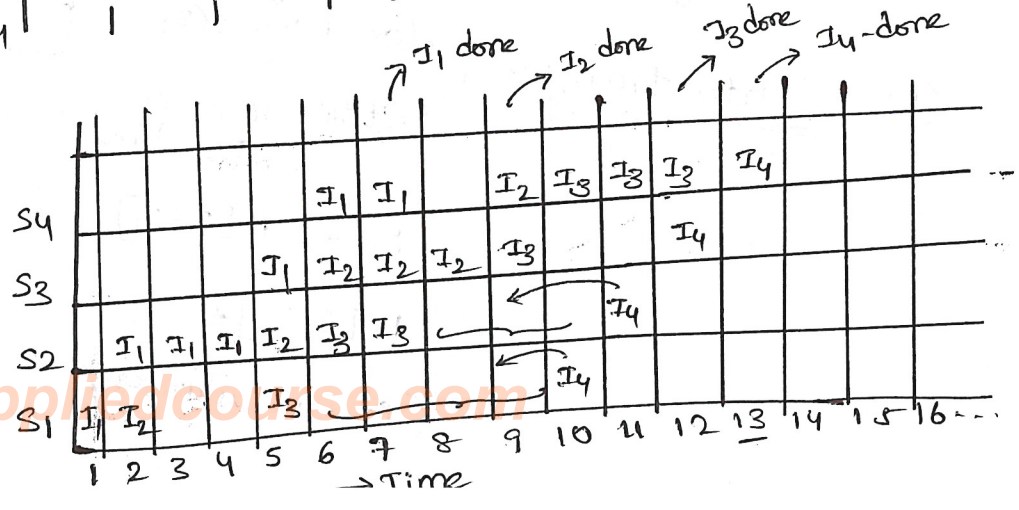

$$\frac{10}{5} = \boxed{2}$$

(Q5) 4-stage pipeline where different instructions spent different amount of time @ each-stage

| real world | | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|---|
| 7 | $I_1$ | 1 | 3 | 1 | 2 |
| 6 | $I_2$ | 1 | 1 | 3 | 1 |
| 7 | $I_3$ | 1 | 2 | 1 | 3 |
| 4 | $I_4$ | 1 | 1 | 1 | 1 |
| 24 | | | | | |

① # cycles to Complete the above program.

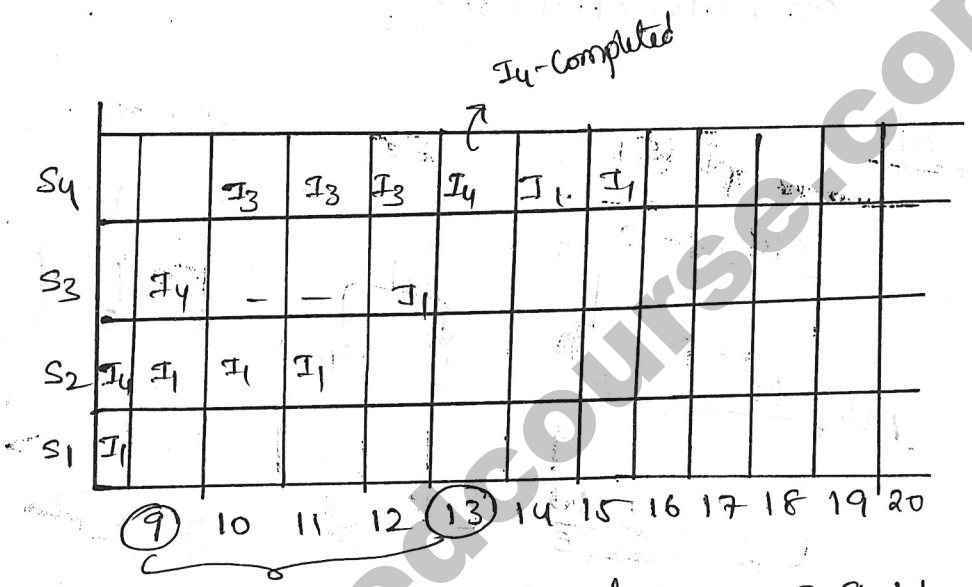
⇒ 13 Cycles



(b) for(i=1; i<=1000; i++)

{
$I_1$;
$I_2$;
$I_3$;
$I_4$;
}

Loop-level parallelism




# cycles-overlapped between 2-iterations = 13-9 +1

= (5)

i=1 $I_4$

i=2 $I_1$

**RISC Pipelining:** 5-Stage Instruction-pipeline

**Recap:** 3 types of instruction

smaller Instruction set

① **Data-transfer**

load r0 3(r1) (reg ← mem)

store 3(r0) r1 (mem ← reg)

$r_0 \leftarrow M[3+r_1]$

$M[3+r_0] \leftarrow r_1$

(index-addressing)

→ S1, S2, ...

## ② Data-Manipulation:

Add $r_0, r_1, r_2$ $\quad r_0 \xleftarrow{ALU} r_1 + r_2$

all operands in Registers (RISC)

## ③ Transfer of Control:

Unconditional | Conditional → complex Conditional Transfer of Control

Jump 2080

PC ← 2080

DJNZ r0, 2080

→ Dec r0

Compare r0 == 0

T F

PC ← 2080

JZ
JNZ

5 Stages → IF, ID, ALU-Execute, Mem-Access, WB

① LOAD $r_0$, 3($r_1$); IWord, Index Addr

$r_0 \leftarrow M[3+r_1]$

S1 IF: IR ← LOAD r0, 3(r1) (Mem)

PC ← PC + step-size (ALU)

S2 ID: S: M[3, r1] 3+r1 : EA

D: reg (r0)

mem-read

S3 EX/ALU: 3 + r1 (ALU) → ACC

TR, ACC, EA

S4 MEM-ACC: RD, Mem, data, ACC → Addr

S5 WB: data → reg-file

② STORE 3(r0), r1 ; M[3+r0] ← r1

S1 IF — IR ← Instruction(Mem)
PC ← pc + Step-size (ALU)

S2 ID — S: reg(r1)
D: M[3+r0] → EA

S3 EXE — Mem-write
3+r0 → ALU → ACC
TR, ACC, EA

S4 MEM-ACC — WR = 0, ACC → addr, r1 → data, main mem

S5 WB — Nothing

③ ADD r0, r1, r2 ;

1 IF — IR ← instruction
PC ← PC + Step-size

2 ID — S: r1, r2 op: Addition
D: r0 ALU

3 EXE — r1 + r2 → ALU → ACC

4 MEM-ACC — NOTHING

5 WB — ACC → r0, WR → reg-file

④ Jump 2080

① IF IR ← instruction
PC ← PC + Step-size

② ID opcode
PC ← 2080

③ Exe — X

④ Mem-ACC — X

⑤ WB PC ← $\overline{WR}$ 2080

⑤ DJNZ r0, 2080

① IF IR ← instruction
PC ← PC + stepsize

② ID ALU-operation on r0

③ Exe r0 ← r0-1
if r0 != 0
ACC=1
else ACC=0

④ Mem-ACC

⑤ WB PC ← 2080 if ACC=1
— if ACC=0

iphone & ipad A13 ARM

ACC $\xrightarrow{WR}$ PC
2080

Pipeline Dependencies: → Structural → Fix / Solution.
→ Data
→ Control

→ Major problems / issues

## Structural Dependancy:

⇓

→ Resource Conflict

→ Waiting / Stall

| | $S_1$ C1 | $S_2$ C2 | $S_3$ C3 | $S_4$ C4 |
|---|---|---|---|---|
| $I_1$ | IF | ID | ALU | WB → Mem-WB |
| $I_2$ | | IF | ID | ALU |
| $I_3$ | | | IF | ID |
| $I_4$ | | | | IF → Mem-RD |

WR RD

IF: Memory-RD → Mem

WB: Memory-WR

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| $I_1$ | IF | ID | ALU | WB | |
| $I_2$ | | IF | ID | ALU | |
| $I_3$ | | | IF | ID | |
| $I_4$ | | | | — | IF |

↓ wait / stall

Eg 2:

5-Stage RISC

WB: reg-write

| | C1 | C2 | C3 | C4 | C5 | C6 | C7 |
|---|---|---|---|---|---|---|---|
| $I_1$ | (Mem) | ID | ALU | Mem | $WB_R$ | | |
| $I_2$ | | (Mem) | (ID) | ALU | Mem | $WB_R$ | |
| $I_3$ | | | (Mem) | ID | ALU | Mem | $WB_R$ |
| → $I_4$ | | | | X | X | X | Mem |

3-cycle-Stalls

## Solution to structural-dep: Renaming

→ Code-Memory (CM)

→ Data-Memory (DM)

2-independant modules

RD → CM Instruction

WR → DM Data/op

| | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 |
|---|---|---|---|---|---|---|---|---|
| I1 | CM | ID | ALU | DM | WB | | | |
| I2 | | CM | ID | ALU | DM | WB | | |
| I3 | | | CM | ID | ALU | DM | WB | |
| I4 | | | | CM | ID | ALU | DM | WB |

Zero-Stalls

## (2) Data Dependency:

I1: Add r0, r1, r2

I2: Sub r3, r0, r4

(r0)old

r0new = r1 + r2

RISC

| | C1 | C2 | C3 | C4 |
|---|---|---|---|---|
| I1 | IF | ID | Exe | WB |
| I2 | | IF | ID | |

src1 : r0
src2 : r4
dest : r3

ID-Stage:

| | op | DEST | Indep src1 | Indep src2 | dep src1 | dep src2 |
|---|---|---|---|---|---|---|
| I1 | ADD | r0 | r1 | r2 | — | — |
| I2 | SUB | r3 | — | r4 | r0 (I1) | — |

TOMASULO - Algorithm

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| I1 | IF | ID | Exe | WB | |
| I2 | | IF | — | — | ID → r0new |

2-Stalls

Eg: 5-Stage RISC

I1: Add r0, r1, r2

I2: Sub r3, r0, r4

| | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ |
|---|---|---|---|---|---|---|
| $I_1$ | IF | ID | EX | MA | WB | |
| $I_2$ | | IF | ID | – | – | EX ... |

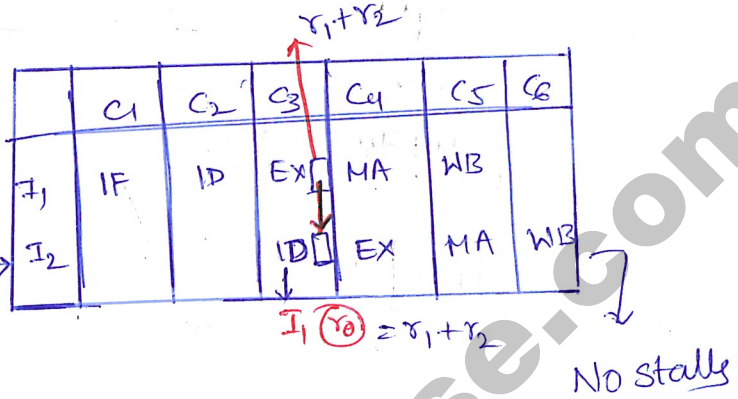
2-Stalls

Buffer / Latch

**Solution:**

(Operand forwarding)

↳ default in RISC →




No Stalls

**Eg:**

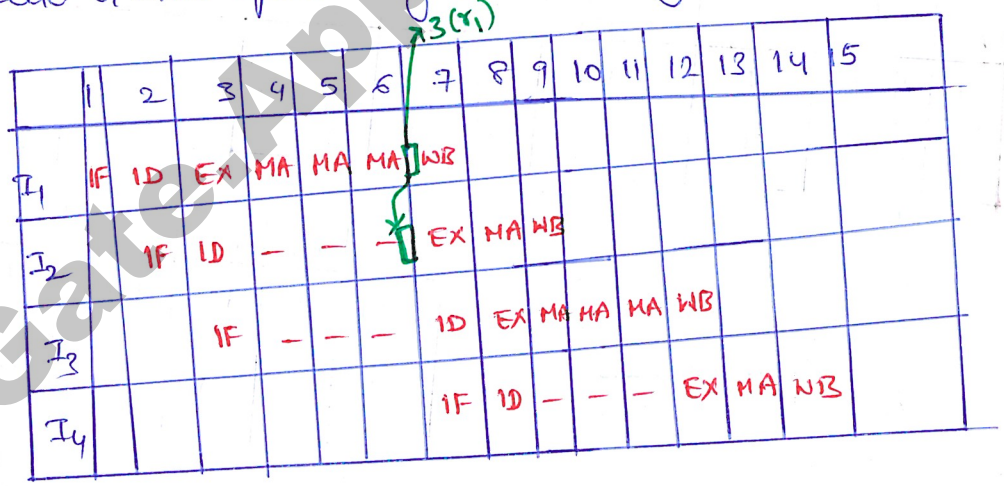
$I_1$ : Load $r_0, 3(r_1)$

$I_2$ : Add $r_3, r_0, r_2$

$I_3$ : Load $r_4, 4(r_2)$

$I_4$ : Sub $r_3, r_5, r_4$

→ Load & Store spend 3-cycles in MA-stage.




| | IF | ID | EX | MA | WB |
|---|---|---|---|---|---|
| $I_1$ | 1 | 1 | 1 | (3) | 1 |
| $I_2$ | 1 | 1 | 1 | 1 | 1 |
| $I_3$ | 1 | 1 | 1 | (3) | 1 |
| $I_4$ | 1 | 1 | 1 | 1 | 1 |

# Cycles to complete $I_1$–$I_4$ : 14

Eg: $I_1$: Add $r_0, r_1, r_2$

$I_2$: MUL $r_3, r_0, r_4$

$I_3$: DIV $r_5, r_6, r_7$

$I_4$: SUB $r_8, r_5, r_7$

→ EX takes 3 cycles for MUL & 6 cycles for DIV

→ RISC ⇒ operand forwarding

MA ← ALU

| | C1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $I_1$ | IF | ID | EX | WB | | | | | | | | | | | |
| $I_2$ | | IF | ID | EX | EX | EX | ~~WB~~ | ~~WB~~ | | | | | | | |
| $I_3$ | | | IF | ID | – | – | EX | EX | EX | ~~EX~~ | EX | EX | WB | | |
| $I_4$ | | | | IF | ID | – | – | – | – | – | ~~EX~~ | ~~WB~~ | – | EX | WB |

($r_0+r_2$; $r_6+r_7$)

# clock cycles to complete $I_1$-$I_4$ : 14

## ③ Control - Dependency:

RISC

| Jump | 2080 |
|---|---|

{IF: IR ← instruction
PC ← PC+SS

{ID: PC ← 2080

✓ 1000: $I_1$

✓ 1001: $I_2$

✓ 1002: $I_3$ (Jump 2080)

× 1003: $I_4$

⋮

✓ 2080: $BI_1$

✓ 2081: $BI_2$

⋮

| | C1 | C2 | C3 | C4 | C5 | C6 | C7 |
|---|---|---|---|---|---|---|---|
| $I_1$ | IF | ID | EX | MA | WB | | |
| $I_2$ | | IF | ID | EX | MA | WB | |
| $I_3$ | | | IF | ID | | | |
| $I_4$ | | | | IF | ID | EX | |
| $BI_1$ | | | | | IF | ID | EX ... |
| $BI_2$ | | | | | | IF | ID ... |

PC ← 2080

PC ← PC+SS

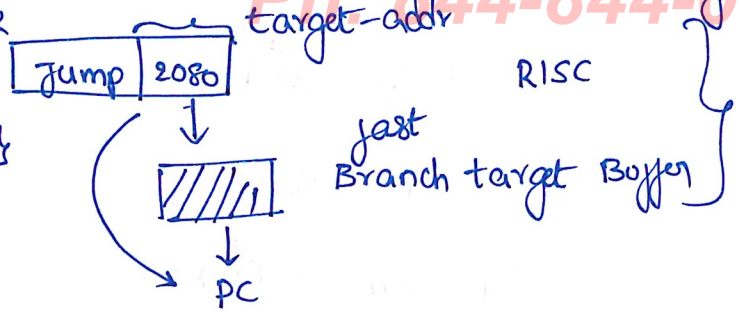
Unwanted instruction ⇓ {FLUSH/DELAY

## Hardware-Solution: Branch-prediction → CISC

→ Branch prediction :

→ {additional HW}

No stalls




→ jump @ 2080

↳ Not the target-addr ⇒ Branch prediction is not possible.

Branch-penalty = # stalls due to branch operation.

## Software-Solutions: Delayed-Branch (Compiler)

→ Rearrange (1) (or) add NOP instruction (2)

⇓

User-program:

1000: $I_1$

1001: $I_2$

1002: $I_3$ (Jump 2000)

1003: $I_4$

:

2000 : $BI_1$

2001 : $BI_2$

| | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ |
|---|---|---|---|---|---|---|---|
| $I_1$ | IF | ID | EX | MA | WB | | PC → 2080 |
| $I_3$ | | IF | ID | | | | |
| $I_2$ | | | IF | ID | EX | MA | WB |
| $BI_1$ | | | | IF | ID | EX | MA ... |
| $BI_2$ | | | | | IF | ID | EX ... |

$I_2$ & $I_3$ are independant

→ NOP-instruction

User program

1000: $I_1$

1001: $I_2$

1002: $I_3$ (Jump 2000)

1003: NOP

✗ 1004: $I_4$

:

2000: $BI_1$

2001 : $BI_2$

IF ID

IF ID

IF ...

# Instruction Scheduling (to avoid stalls) → DBMS

$I_1$ : Add $r_0, r_1, r_2$

$I_2$ : Sub $r_3, r_0, r_4$

$I_3$ : MUL $r_4, r_5, r_6$

$I_4$ : DIV $r_3, r_7, r_8$

data dep

out-of-order:

$I_1 \rightarrow I_3 \rightarrow I_4 \rightarrow I_2$

order: $I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow I_4$

(Stalls)

## Error Anti-data dependancy:

$I_1$ : Add $r_0, r_1, r_2$

$I_2$ : Sub $r_3, r_0, r_4$

$I_3$ : MUL $r_4, r_5, r_6$

$I_4$ : DIV $r_3, r_7, r_8$

$r_4 = 10$

$r_5 = 10$

$r_6 = 10$

order: $I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow I_4$

$r_4 = 10$

out of order

$I_1 \rightarrow I_3 \rightarrow I_4 \rightarrow I_2$

$r_4 = 100$ (at $I_3$)

$r_4 = 100$ (at $I_2$)

$I_3 - I_2$

$r_4$

write-before-read

## Error: o/p Dependency:

$I_1$ : Add $r_0, r_1, r_2$

$I_2$ : Sub $r_3, r_0, r_4$

$I_3$ : MUL $r_4, r_5, r_6$

$I_4$ : DIV $r_3, r_7, r_8$

out-of-order:

$I_1 \rightarrow I_3 \rightarrow I_4 \rightarrow I_2$

$I_4 - I_2$

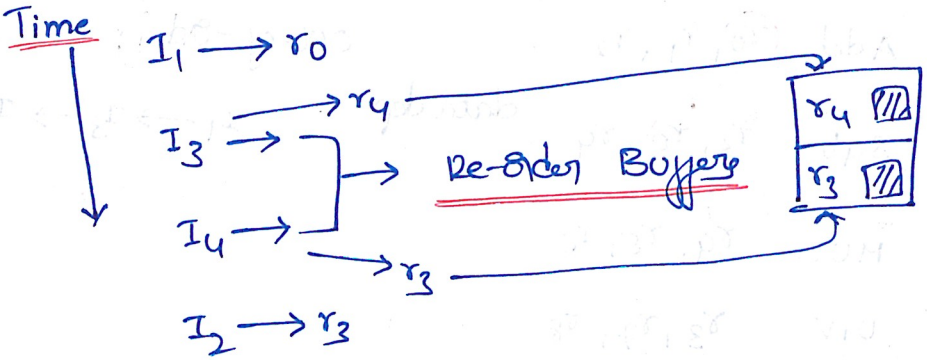
$r_3 = I_2$

write-before-write

order: $I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow I_4$

$r_3$ → $I_4$

## Hardware solution: Reorder Buffers




Re-Order Buffers to $r_4$ & $r_3$ from $I_3$ & $I_4$ resp

## More Solved problems -1

① Consider the sequence of machine instructions

| | |
|---|---|
| MUL R5, R0, R1 | 3 |
| DIV R6, R2, R3 | 5 |
| ADD R7, R5, R6 | 1 |
| SUB R8, R7, R4 | 1 |

In the above sequence R0 to R8 are general purpose registers. In the instructions shown, the first register stores the result of the operation performed, on the second and third registers. This sequence of instructions is to be executed in a pipelined instruction processor with the following 4 stages: (1) Instruction fetch and decode (IF) (2) operand Fetch (OF) (3) perform operation (PO) and (4) write back the result (WB). The IF, OF and WB stages take 1 clock cycle each for any instruction. The PO stage takes 1 clock cycle for ADD & SUB

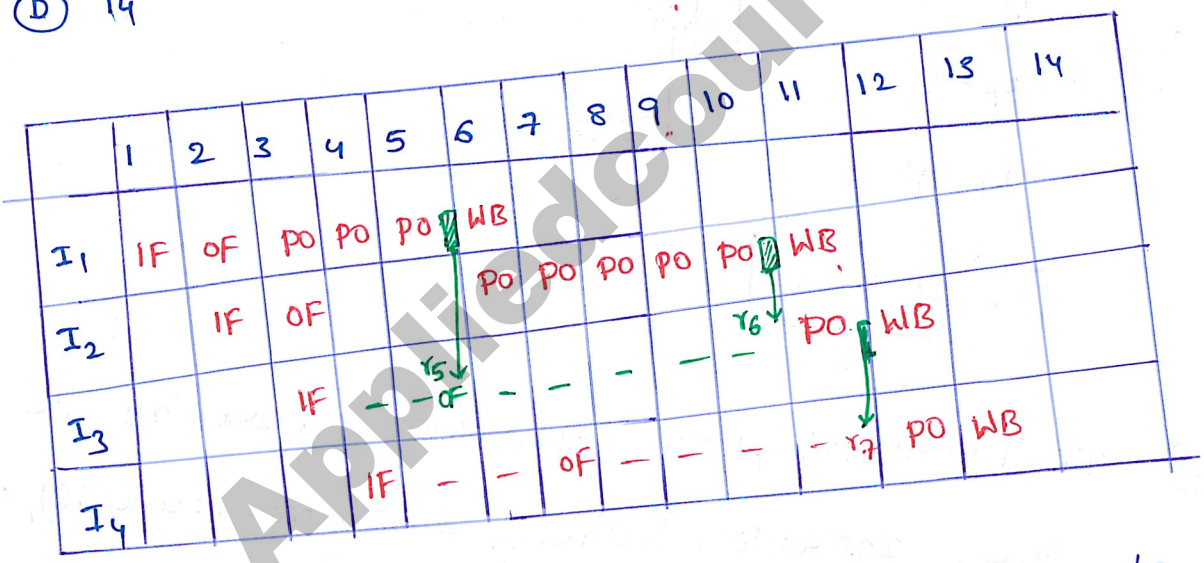
instruction, 3 clock cycles for MUL instruction and 5 clock cycles for DIV instruction. The pipelined processor uses "operand forwarding" from the PO stage to the OF stage. The number of clock cycles taken for the execution of the above sequence of instructions is _____

(A) 11

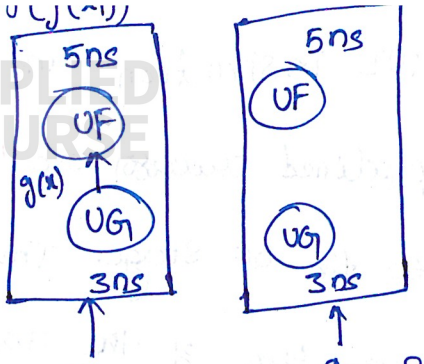
(B) 12

(C) 13

(D) 14




(2) Suppose the functions F and G can be computed in 5 and 3 nano seconds by functional units UF and UG, respectively. Given two instances of UF and two instances of UG, it is required to implement the computation F(G(Xi)) for $1 <= i <= 10$. Ignoring all other delays, the minimum time required to complete this computation is _____ nanoseconds.




Pipelining

$f(g(x_1))$ ←

$f(g(x_2))$ ←

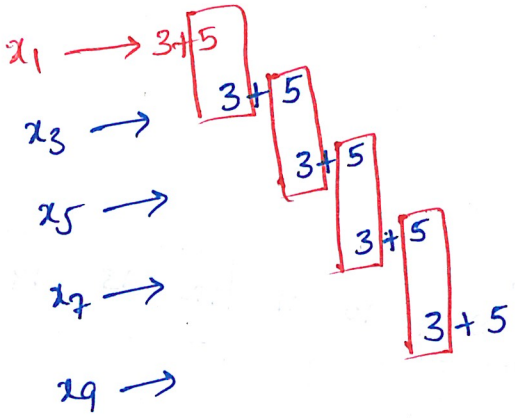
⋮

$f(g(x_{10}))$

(5) Odd { $x_1$, $x_3$, $x_5$ }

{ $x_2$ } Even

(5)




Bothe Even & odd cases runs parallely.

3+5×5 = 28

(3) The instruction pipeline of a RISC processor has the following stages. Instruction fetch (IF), Instruction Decode (ID), operand Fetch (OF), perform operation (PO) and write back (WB). The IF, ID, OF and WB stages take 1 clock cycle each for every instruction. Consider a sequence of 100 instructions. In the PO stage, 40 instructions take 3 clock cycles each, 35 instructions take 2 clock cycles each, and the remaining 25 instructions take 1 clock cycle each. Assume that there are no data Hazards and no Control Hazards. The number of clock cycles

required for Completion of execution of the sequence of instruction is 219

IF, ID, OF, WB → 1 cycle

n = 100 instructions → 40 → 3 cycles (PO)
35 → 2 cycles (PO)
25 → 1 cycle (PO)

K = 5

$(K+n-1) + (40 \times 2) + (35 \times 1) + (25 \times 0)$

$= 104 + 80 + 35$

$= \boxed{219}$

(4) Instruction Execution in a processor is divided into 5 stages. Instruction Fetch (IF), Instruction Decode (ID), operand Fetch (OF), Execute (EX) and write Back (WB). These stages take 5, 4, 20, 10 and 3 nano seconds respectively. A pipelined implementation of the processor requires buffering between each pair of consecutive stages with a delay of 2ns. Two pipelined implementations of the processor are contemplated.

(i) a Naive pipeline implementation (NP) with 5 stages and

(ii) An Efficient pipeline (EP) where the OF stage is divided into stages OF1 and OF2 with execution time of 12ns and 8ns respectively.

The Speedup (Correct to two decimal places) achieved by EP over NP in Executing 20 independant instructions with no hazard is _____

K+n-1

5, 4, 20, 10 and 3

5, 4, 12, 8, 10, 3

**NP** (K = 5)

$t_p$ = 22ns

n = 20

K = 5

$\Rightarrow (5+20-1) * 22$ ns

$\Rightarrow 24 * 22$ ns

$\Rightarrow$ **528 ns**

**EP** (K = 6)

$t_p'$ = 14ns

n = 20

K = 6

$(20+6-1) * 14$ ns

25 x 14

**350ns**

$\Rightarrow$ Speedup = $\frac{528}{350} = 1.508 =$ **1.51**

(5) The Stage delays in a 4-Stage pipeline are 800, 500, 400 and 300 pico seconds. The First stage (with delay 800 pico seconds) is replaced with a functionality equivalent design involving two stages with respective delays 600 and 350 pico seconds. The Throughput increase of the pipeline is _____ percent.

| 4 = K | K = 5 |
|---|---|
| 800, 500, 400, 300 | 600, 350, 500, 400, 300 |

Nano sec - $10^{-9}$ sec

Pico sec - $10^{-12}$ sec

$t_p = 800$ ns

If n is very Large

$\frac{n-k+1}{n} \approx 1$ instruction → cycle

$Thr_1 = \frac{1}{800}$

$t_p = 600$

$Thr_2 = \frac{1}{600}$

$\Rightarrow \frac{Thr_2 - Thr_1}{Thr_1} * 100$

$= \boxed{33.33\%}$

## More Solved problems-11

① Consider a 3 GHz processor with a three stage pipeline and stage latencies $v_1, v_2$ and $v_3$ such that $v_1 = 3v_2/4 = 2v_3$. If the Longest pipeline stage is split into two pipeline stages of equal latency, the new frequency is (4) GHz, ignoring delays in the pipeline registers.

(A) 2

(B) 4 ✓

(C) 8

(D) 16

$k=3$

$v_1 = \frac{3}{4}v_2 = 2v_3$

let $v_1 = 6x$ ns

$v_2 = \frac{4}{3} \times 6x = 8x$ ns

$v_3 = 3x$ ns

$t_p = 8x$ ns

$f = \frac{1}{8x} = 3$ GHz (Given)

$\frac{1}{x} = 24$ GHz

$6x \quad 8x \quad 3x$

$6x \quad 4x \quad 4x \quad 3x$

$t_p = 6x$

$f = \frac{1}{6x} = 4$ GHz

(2) Consider a non-pipelined processor with a clock rate of 2.5 GHz 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 GHz. Assume that there are no stalls in the pipeline. The speedup achieved in this pipelined processor is ——

(A) 3.2
(B) 3.0
(C) 2.2
(D) 2.0

$f = 2.5\,GHz \Rightarrow T = \frac{1}{2.5}$

$CPI = 4$ $\quad ET_{old} \Rightarrow 4 * \frac{1}{2.5}\,ns$

$= 1.6\,ns$

pipeline processor $K = 5$

$f_p = 2\,GHz \quad t = \frac{1}{2\,GHz}$

$= 0.5\,ns$

$CPI = 1$

$ET_{new} = 1 * 0.5\,ns$

$= 0.5\,ns$

$Speedup = \frac{1.6}{0.5}$

$= 3.2$

(3) Consider the following code sequence having five instructions I1 to I5. Each of these instructions has the following format.

op Ri, Rj, Rk

where operation op is performed on the contents of

registers Rj and Rk and the result is stored in Register Ri.



I1: ADD R1, R2, R3

I2: MUL R7, R1, R3

I3: SUB R4, R1, R5

I4: ADD R3, R2, R4

I5: MUL R7, R8, R9

Consider the following three statements.

X S1: There is an anti-dependance (write-before-read) between instructions I2 and I5

✓ S2: There is an anti-dependance between instructions I2 and I4.

X S3: Within an instruction pipeline an anti-dependance always creates one or more stalls.

Which one of the above statements is/are correct?

(A) Only S1 is true

(B) Only S2 is true

(C) Only S1 and S2 are true.

(D) Only S2 and S3 are true.

(4) Consider the following processors (ns stands for nano-seconds). Assume that the pipeline registers have zero latency.

P1: Four-stage pipeline with stage latencies 1ns, 2ns, 2ns, 1ns ⇒ 2

P2: Four-stage pipeline with stage latencies 1ns, 1.5ns, 1.5ns, 1.5ns ⇒ 1.5

P3: Five-stage pipeline with stage latencies 0.5ns, 1ns, 1ns, 0.6ns, 1ns ⇒ 1

P4: Five-stage pipeline with stage latencies 0.5ns, 0.5ns, 1ns, 1ns, 1.1ns ⇒ 1.1

Which processor has the highest peak clock frequency?

$\uparrow f = \frac{1}{t}\downarrow$

(A) $P_1$

(B) $P_2$

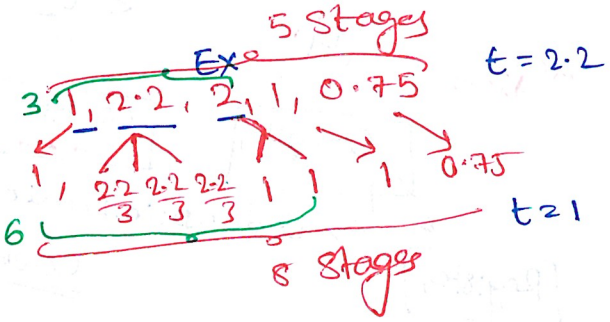
(C) $P_3$

(D) $P_4$

(5) An instruction pipeline has five stages, namely instruction fetch (IF), instruction decode and register fetch (ID/RF), instruction Execution (EX), memory access (MEM), and register write back (WB) with stage latencies 1ns, 2.2ns, 2ns, 1ns, and 0.75ns respectively. (ns stands for nano seconds).

To gain in terms of frequency, the designers have decided to split the ID/RF stage into three stages (ID, RF1, RF2) each of Latency 2.2/3 ns. Also EX Stage is split into two stages (EX1, EX2) each of latency 1ns. The new design has a total of 8 pipeline stages. A program has 20% branch instructions which execute in the EX stage and produce the next instruction pointer at the end of the EX stage in the old design and at the end of the ~~[and produce]~~ EX2 stage in the new design. The IF Stage stalls after fetching a branch instruction until the next instruction pointer is computed. All instructions other than the branch instruction have an average CPI of one in both the designs. The execution times of this program on the old and the new design are P and Q nanoseconds, respectively. The value of P/Q is 1.54

(A) 1.5
(B) 1.4
(C) 1.8
(D) 2.5



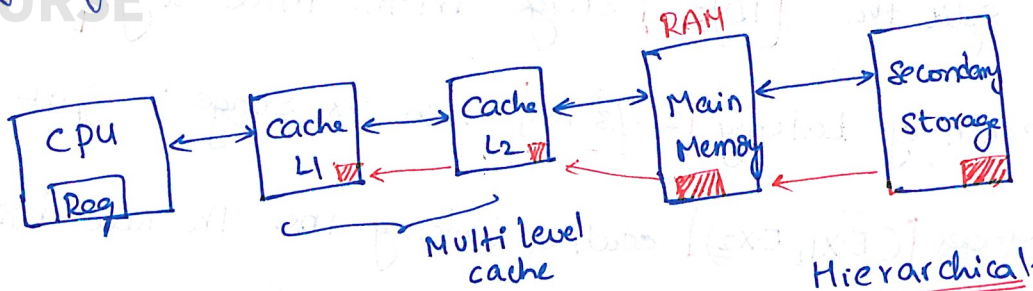

| | I st case | II nd case |
|---|---|---|
| 20% of branch instructions | $= (0.80 \times 1 + 0.2 \times 3) 2.2$ | $= (0.8 \times 1 + 0.2 \times 6) 1$ ns |
| | $P = 3.08$ | $Q = 2$ ns |

$P/Q = 3.08/2 = 1.54$

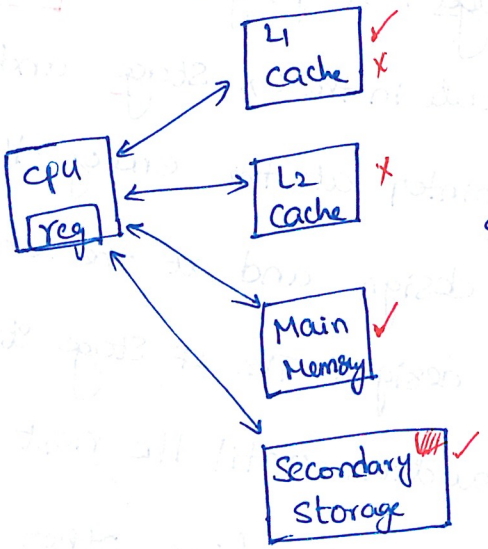
# Memory Organization:

Reg, Cache, RAM, Disks

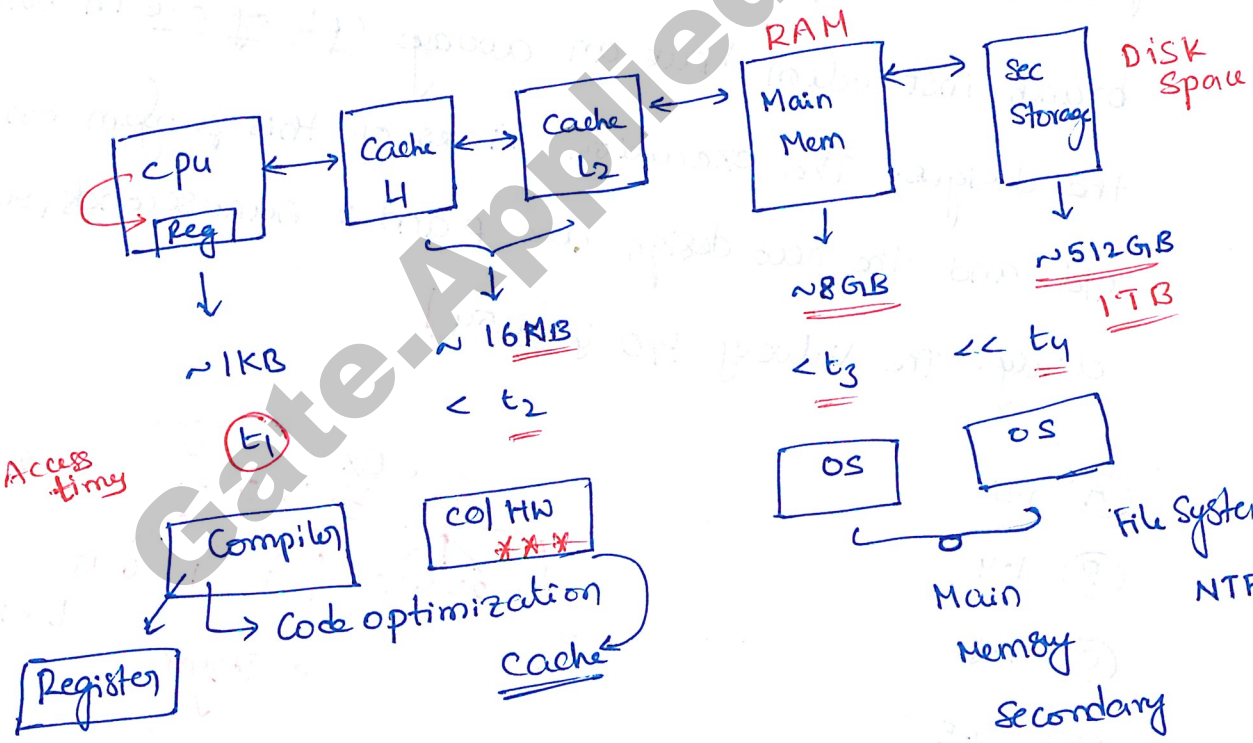





Hierarchical-access-memory-hierarchy (default)



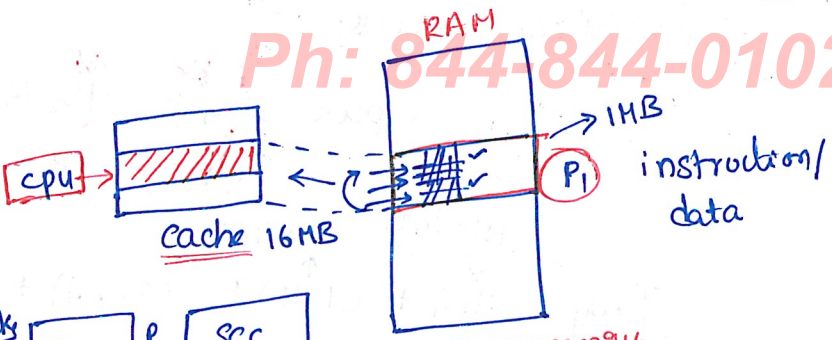

Simultanous-access-Memory-Hierarchy.

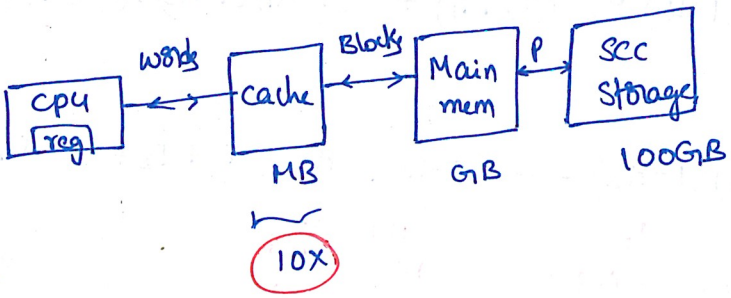


## Locality of Reference:

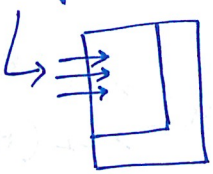
Spatial

Temporal




### Spatial



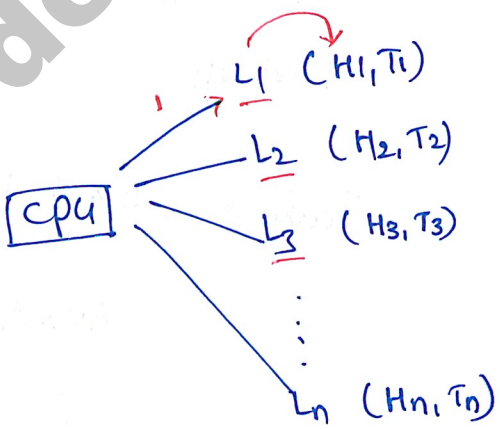
Keep Instructions & data in Cache is known as Spatial LOR

### Temporal

i, j, t, y variables


Repeatedly accessing the variables kept in cache known as Temporal LOR.

## Simultaneous - Access:




$T_1 \le T_2 \le T_3 \le \cdots$

Expected time to access

$$T_{avg} = H_1 T_1 + (1-H_1) H_2 T_2 + (1-H_1)(1-H_2) H_3 T_3 + \cdots + (1-H_1)(1-H_2)\cdots(1-H_{n-1}) H_n T_n$$

Hit: $H_1 T_1$ — Miss: $(1-H_1) H_2 T_2$

$(1-H_1)$: Miss $L_1$; $(1-H_2)$: Miss $L_2$

Expected Time to Access

$$\eta_{mem} = \frac{1}{T_{avg}} \text{ words/sec}$$

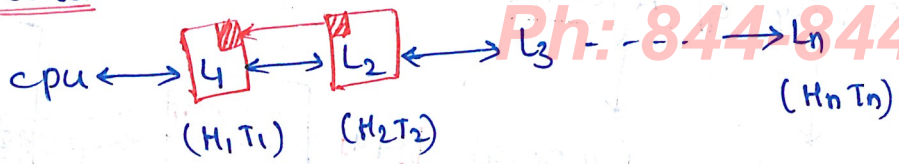
Avg Access Speed.

Hit-ratio/rate $= \frac{\#\text{hits}}{\#\text{accesses}}$, $0 \le Hi \le 1$

$= Hi$ [Miss]

$T_i$: time to access stage i

## Hierarchical - Access:




Hit in L1 ↓ Miss in L1 ↑

$$T_{avg} = H_1 T_1 + (1-H_1) H_2 (T_2+T_1) + (1-H_1)(1-H_2) H_3 (T_1+T_2+T_3)$$

$$+ \cdots$$

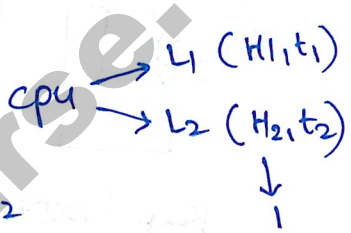
$$+ (1-H_1)(1-H_2) \cdots (1-H_{n-1}) H_n (T_1+T_2+\cdots T_n)$$

$L_1$ Miss, $L_2$ Miss, $L_{n-1}$ Miss, $H_n$ ↓ Hit in $L_n$

$\Rightarrow \boxed{H_n = 1}$

(1Q) 2-level memory-heirarchy (Sequential)

$L_1$ is 8 times faster than $L_2$




$T_1 = 20ns$

$T_1 = T_{avg} - 40ns$

Find $H_1$

$8t_1 = t_2$

$8 \times 20 = t_2$

$\Rightarrow \boxed{t_2 = 160}$

$$T_{avg} = H_1 * T_1 + (1-H_1) H_2 T_2 = T_1 + 40ns$$

$$20 \times H_1 + (1-H_1) \times 1 \times 160 = T_1 + 40$$

$$= 20 + 40$$

$$20H_1 + 160 - 160H_1$$

$$\Rightarrow \boxed{H_1 = 0.714} \quad (71.4\%)$$

# Cache - Memory organization:

Hierarchical Access


(Locality of Reference)

→ Memory - organization

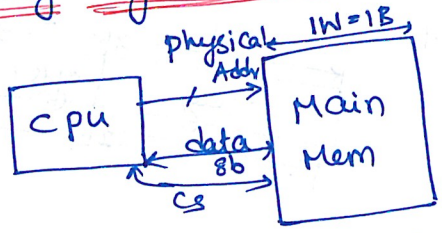
→ Mapping - techniques

→ Replacement - Methods

→ updations

→ Multi-level-Cache → $L_1$ $L_2$ $L_3$

## Memory organization:

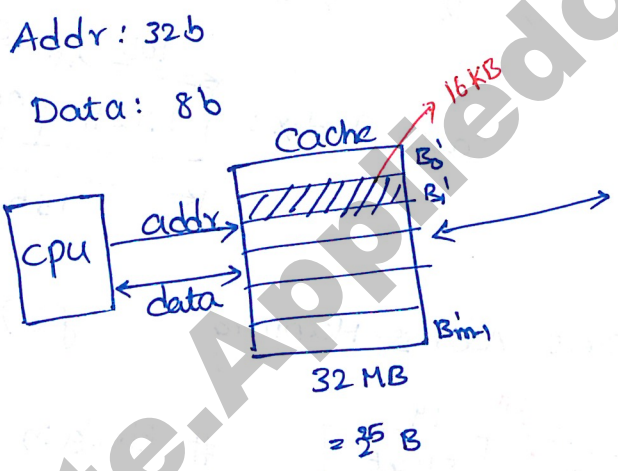


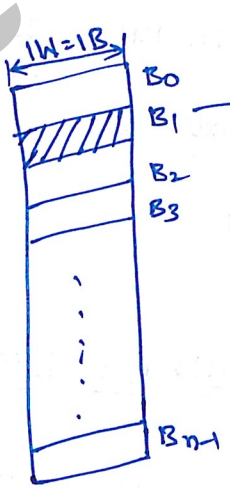

n = # blocks in MM

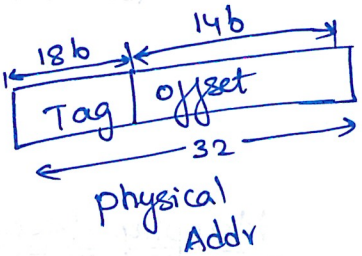
$4\,GB = 2^{32}B = 2^{32}W$

Addr: 32b

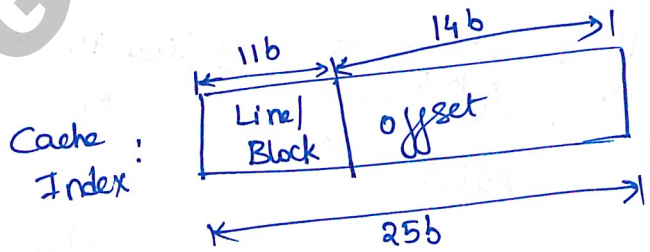
Data: 8b



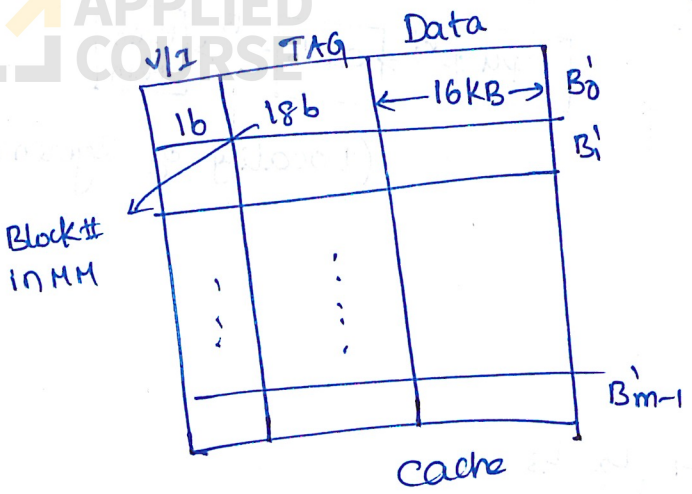

$4\,GB = 2^{32}B = 2^{32}W$




$$\#\text{ Blocks in MM} = \frac{2^{32}}{2^{14}} = 2^{18} = n$$

$$\#\text{ Blocks in CM} = \frac{2^{25}}{2^{14}} = 2^{11} = m$$

Total size of CM = m*(1+18+16k x 8) bits




MM

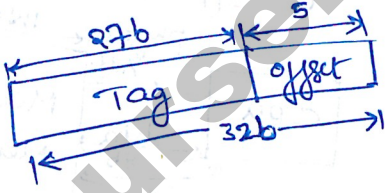
$\Rightarrow$ $B_0'$ ← $B_3$

CM MM

(Q1) 32KB cache $= 2^{15}$ B

32B-blocks $= 2^5$ B $\Rightarrow$ 1B = 1W

cpu: 32b physical Addr

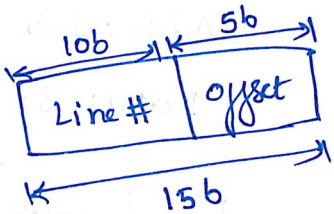



# lines in CM = $\frac{2^{15}}{2^5} = 2^{10} = m$

# blocks in MM = $\frac{2^{32}}{2^5} \Rightarrow 2^{32-5} \Rightarrow 2^{27}$ n

Addr-Format =

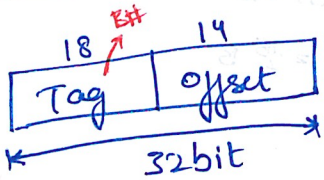
Cache Index:




## Mapping Techniques: Direct-Address-Mapping

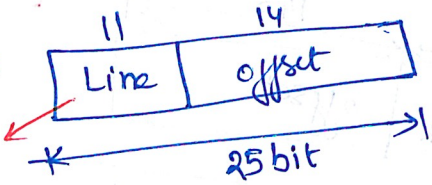
f( ): mod

physical-addr:




# blocks in MM $= 2^{18} = n$

cache-Index:

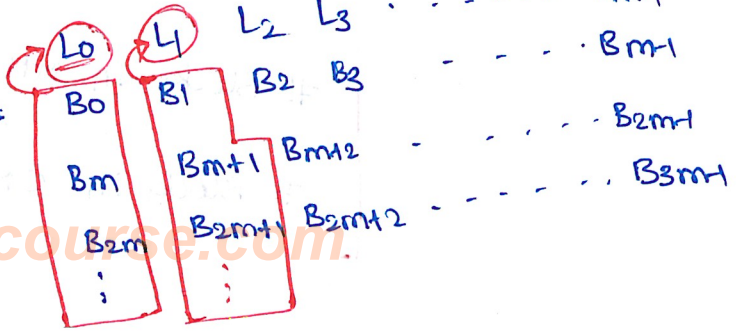



# lines in CM $= 2^{11} = m$

$pa_1 = B_2$
$pa_2 = B_{m+2}$

f: tag mod m = line#

B# C $2^{11}$



## Computing - Mod:

Base 10: $123 \bmod \underset{10^1}{10} = 3$

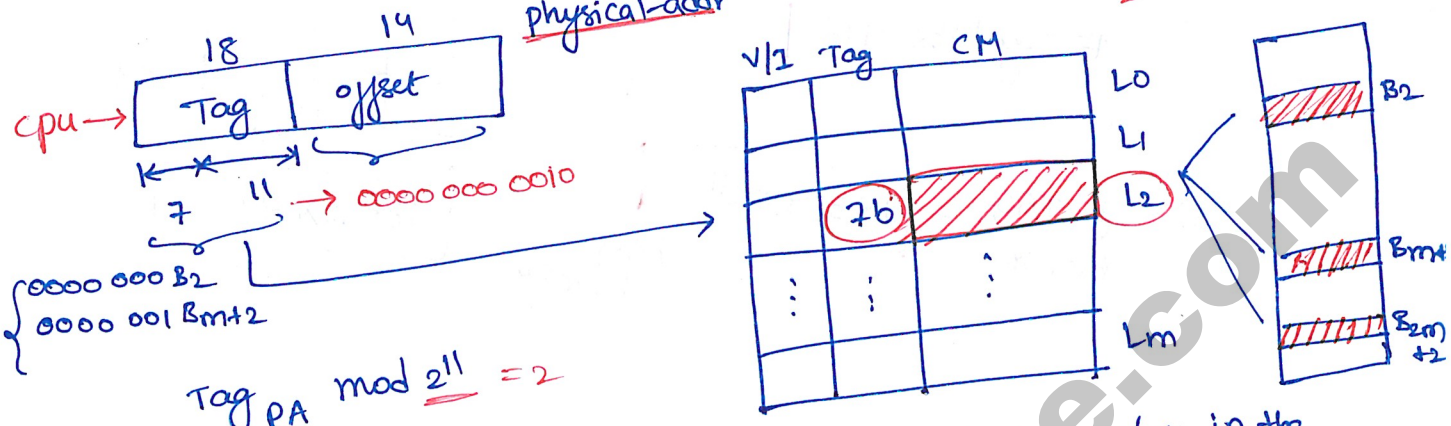
$1678 \bmod \underset{10^2}{100} = 78$

Base 2:

$1000 \bmod 100 = 00$

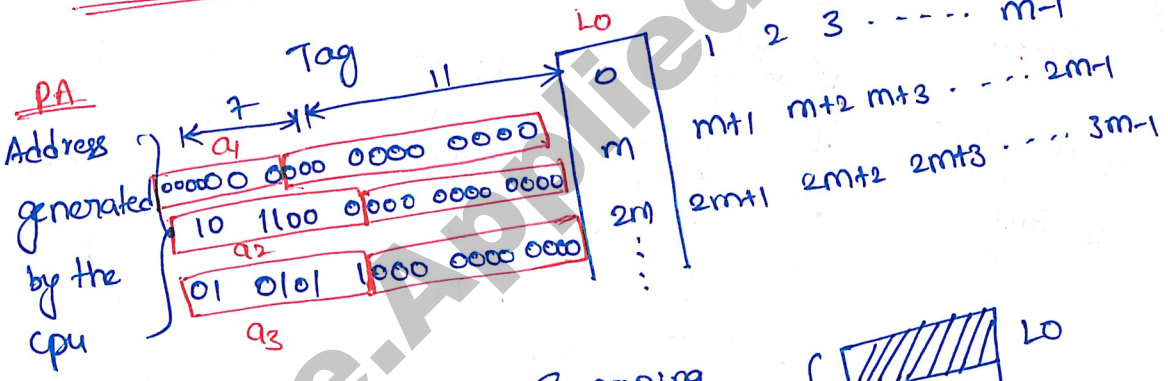
$1001 \bmod 100 = 01$

$1010 \bmod \underset{2^2}{100} = 10$

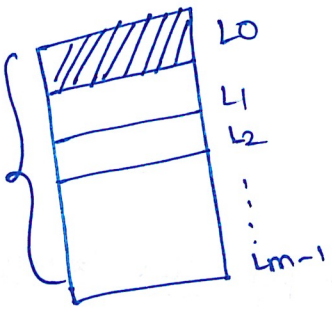



In case of cache MISS, we need to replace in the cache.

## Limitation of Direct Mapping:




→ Trashing: Too much Swapping

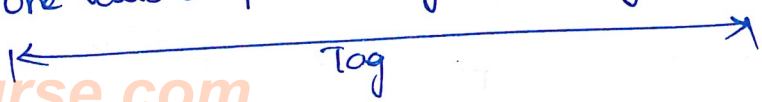



Eg: 64KB direct-mapped cache

$2^{16}$B 32B-blocks/line $\Rightarrow 2^5$

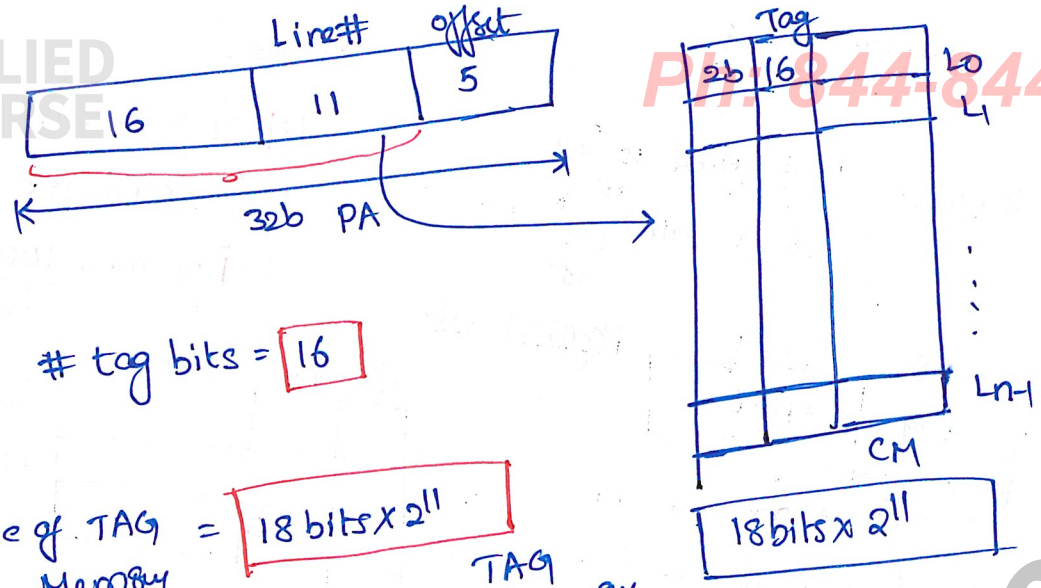
cpu: 32b-addr

cache-Controller: one valid bit, one modified bit, tag-bits.




# tag-bits

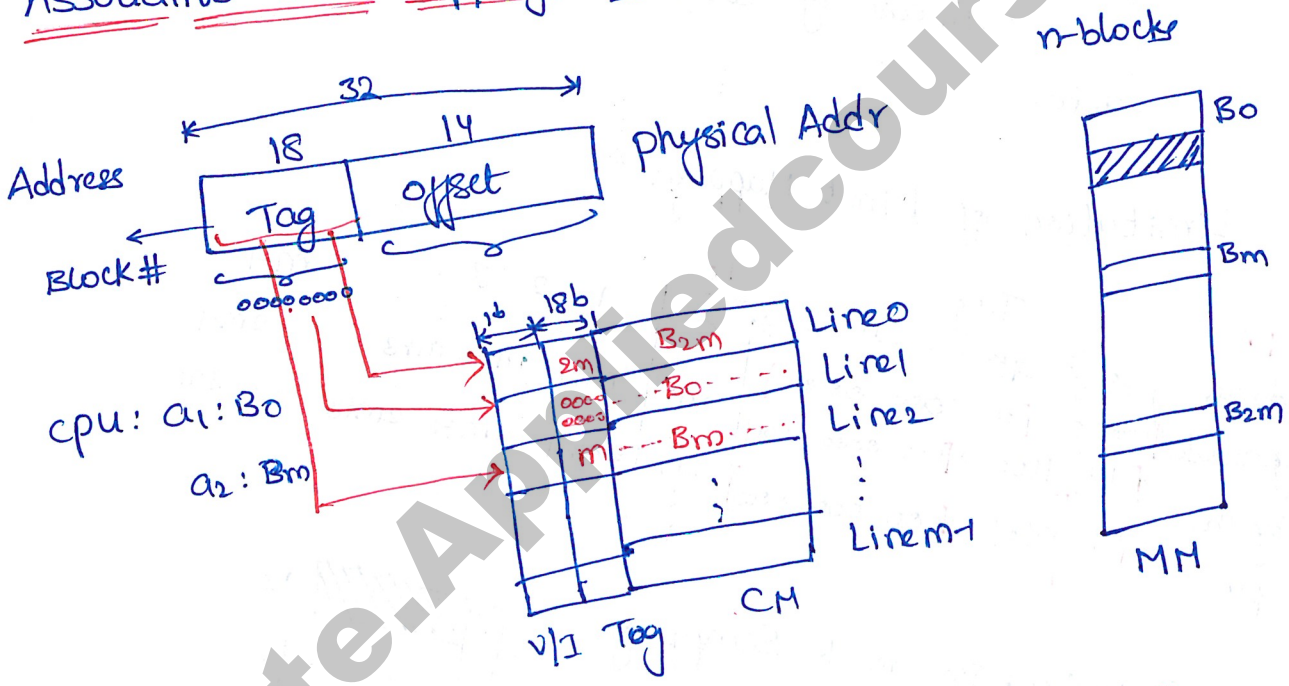
# size of tag-memory:




# tag bits = 16

Size of TAG Memory = 18 bits x $2^{11}$

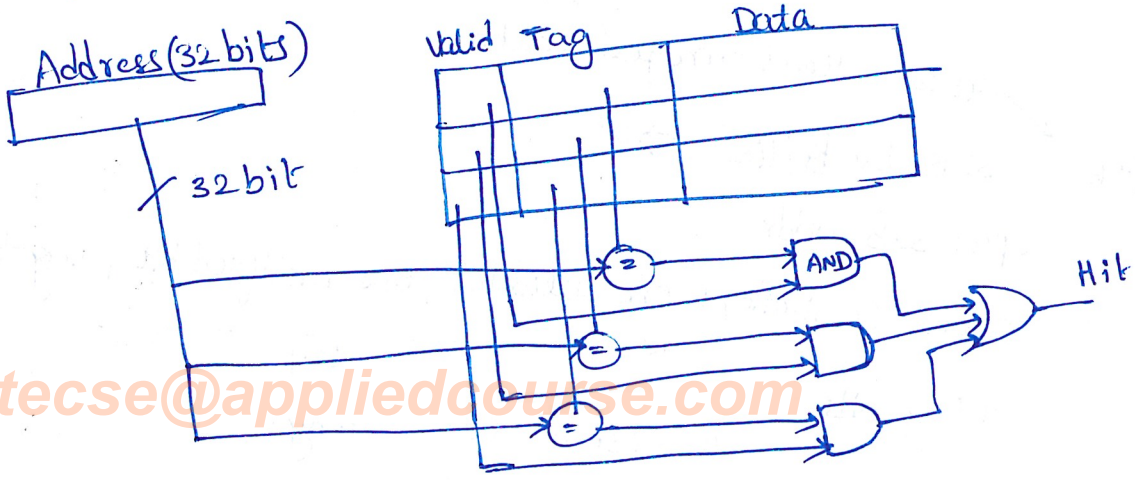
## Associative-addr-Mapping: (Fully)




## m-parallel-Comparisions:

m - $2^{11}$

Circuit Complex



(a) 32-KB fully-Associative Cache Memory

$2^{15}$

Line size = 32B $\Rightarrow 2^5$ B

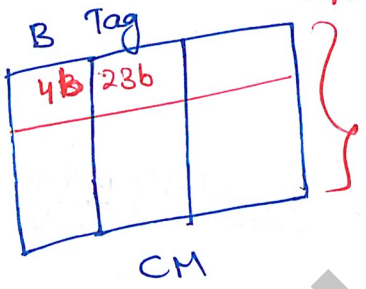
Main-mem-size = $2^{28}$ B

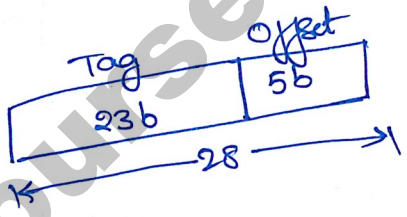
1 valid bit, 1 modified bit, 2 replacement bits.

(a) # tag-bits : 23

(b) Tag-Memory size: $(4+23)*2^{10}$ $\Rightarrow 27*2^{10}$ bits

# blocks = $n = \frac{2^{28}}{2^5} = 2^{23}$



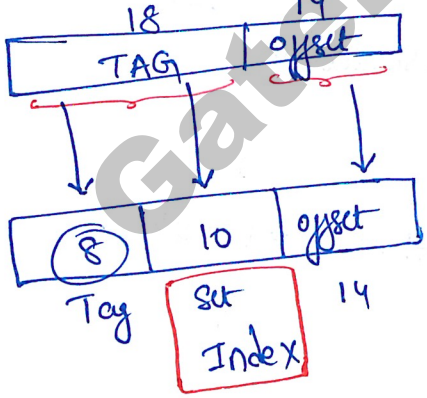



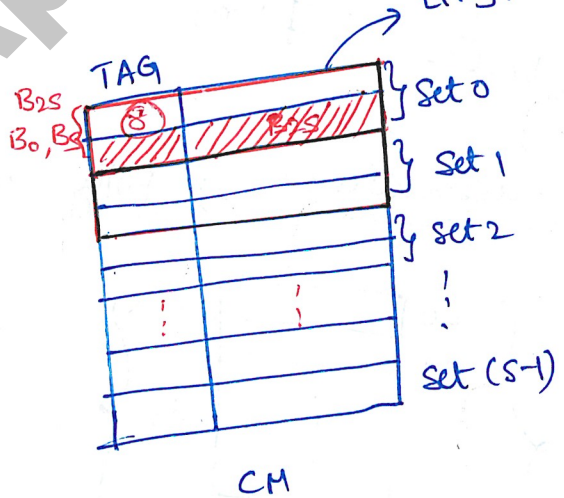

$\Rightarrow n = \frac{2^{15}}{2^5} = 2^{10}$

## Set-Associative Mapping / Cache Memory:

Flexible than DM | AM

Line/Blocks $m = 2^{11}$




2 way $\Downarrow$ 2 Blocks/set

each Set : 2 line

$S = \frac{m}{2} = \frac{2^{11}}{2} = 2^{10}$

Set-Index = TagpA mod $2^{10}$

a : B0

2-Comparisions ← 2-way

4-Comparisions ← 4-way

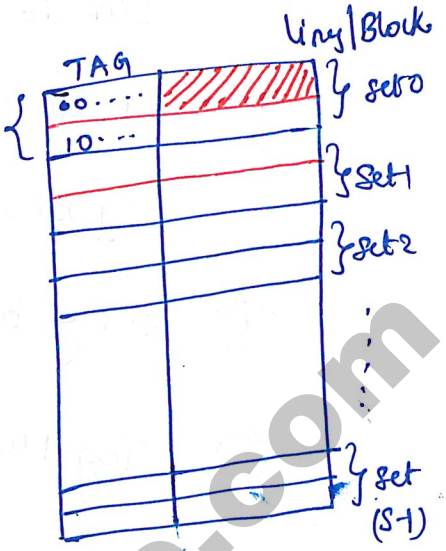
PA: 8b | 10 | 14b offset

$a_1$: 00 0000 0000 0000 0000

$a_2$: 10 1100 0000 0000 0000

$a_3$: 01 0101 1000 0000 0000

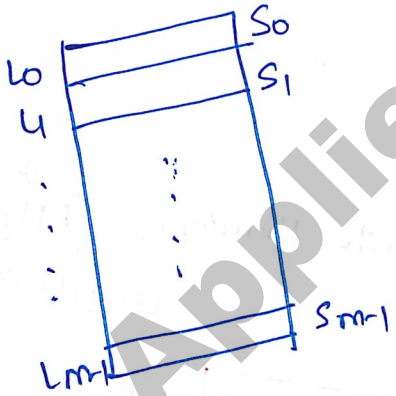
Cache: tag | set-index




# Comparisions = 2 in 2-way SA-Mem

= K in K-way SA-mem

## 1 way-set-Associative ⇒ Direct-Mapping



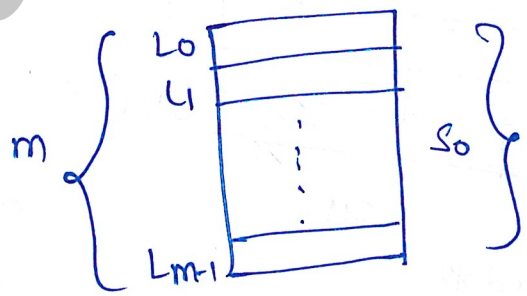

# sets = # of Lines

= $2^{11}$

In practice: 2-way, 4-way

## M-way set-associative: ⇒ Fully-Associative

# Line in CM = m




# sets = 1 = $2^0$

(a) CM-size = 32KB $= 2^{15}$ B

Block-size = 32B $= 2^5$ B

MM-size $= 2^{28}$ B

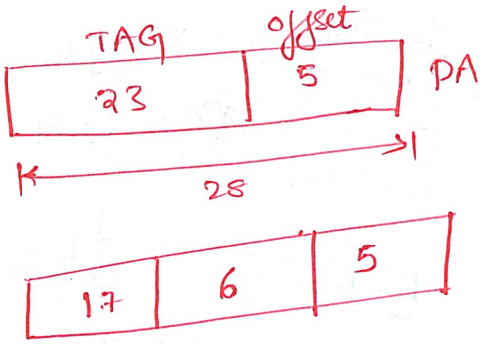
$\Rightarrow \frac{2^{15}}{2^5} = 2^{10}$ Lines

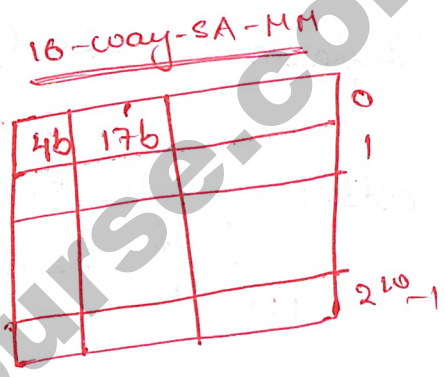
$S = \frac{2^{10}}{2^4} = 2^6$

$2^4$ = 16-way-SA-MM

(a) addr-format

(b) Size of Tag-Mem with 4-additional bits/block.




(46+176) * $2^{10}$

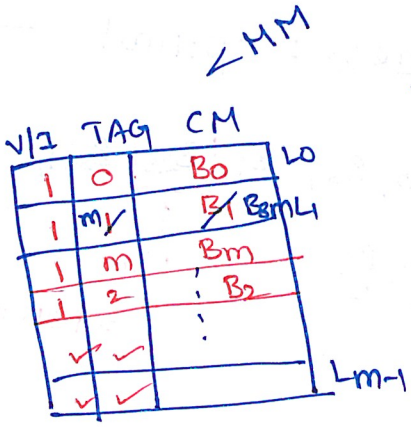
21Kb

## Replacement - Strategies:


Direct Addressing

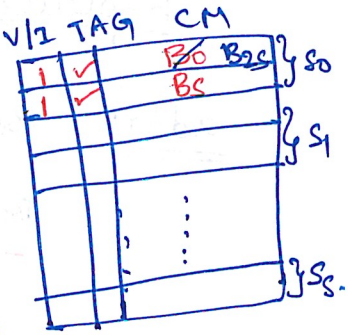
addr: Bo, B1, Bm, B2 ...

m mod m ⇒ 0 ⇒ L0




Associative Mapping

$B_{3m}$




2-way Set-Associative Mapping

addr: Bo, Bs, B2s, B1 ....

0 mod s

2s mod s  s mod s ⇒ 0

OS

## Replacement: FIFO Queue

addr: $B_0, B_S, B_1, B_{2S}, B_{S+1}, B_{2S+1} \cdots$

1 mods ⇒ 1

2S mods ⇒ 0

2S+1 mods ⇒ 1


2-way set associative Mapping

## Replacement: LRU (Least Recently used) → used in practical

addr: $B_0, B_S, B_1, B_{2S}, B_{S+1}, B_{2S+1} \cdots$

↑ LRU

↓ MRU

LRU

$2S \bmod S = 0$

$B_{2S+1} \bmod S \Rightarrow 1$


2-way set associative Mapping

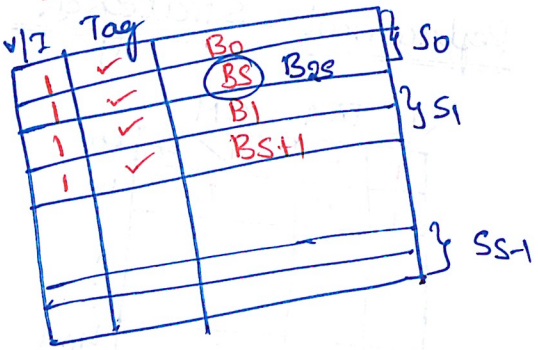
## Replacement: optimal → Not practical

addr: $B_0, B_S, B_1, B_{2S}, B_{S+1}, \cdots$

$B_0, B_S$

time

→ Future



Eg 1: 4-block Cache (Initially Empty)

Order of main-memory block reference

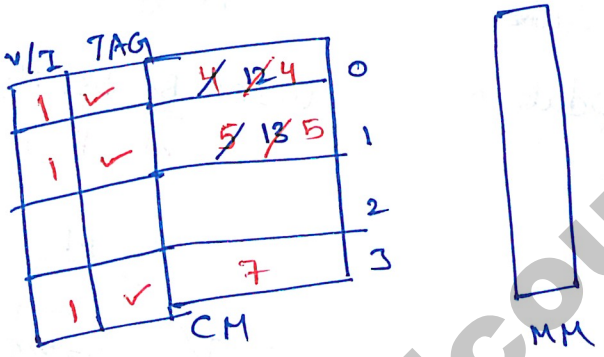
4, 5, 7, 12, 4, 5, 13, 4, 5, 7

Calculate hit-ratio if

① DAM

② 2-way-set-associative with LRU

①




m=4

⇒ 4 mod 4 = 0

5 mod 4 = 1

7 mod 4 = 1

12 mod 4 = 0

4 mod 4 = 0

5 mod 4 = 1 ⇒ HIT ✓

13 mod 4 ⇒ 1 ⇒

4 HIT ✓

5 HIT

7 HIT ✓

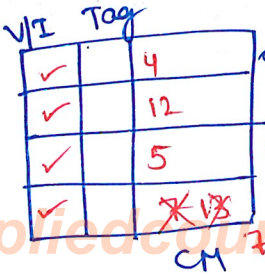
Hit Rate = $\frac{3}{10}$ = 0.3

30%

M M M M H H M H H M

4, 5, 7, 12, 4, 5, 13, 4, 5, 7

②




⇒ 4 mod 2 = 0 Set 0

⇒ 5 mod 2 = 1 Set 1

⇒ 7 mod 2 = 1 Set 1

⇒ 12 mod 2 = 0 Set 0

13 mod 2 = 1

⇒ $\frac{4}{10}$ = 40% Hit Ratio

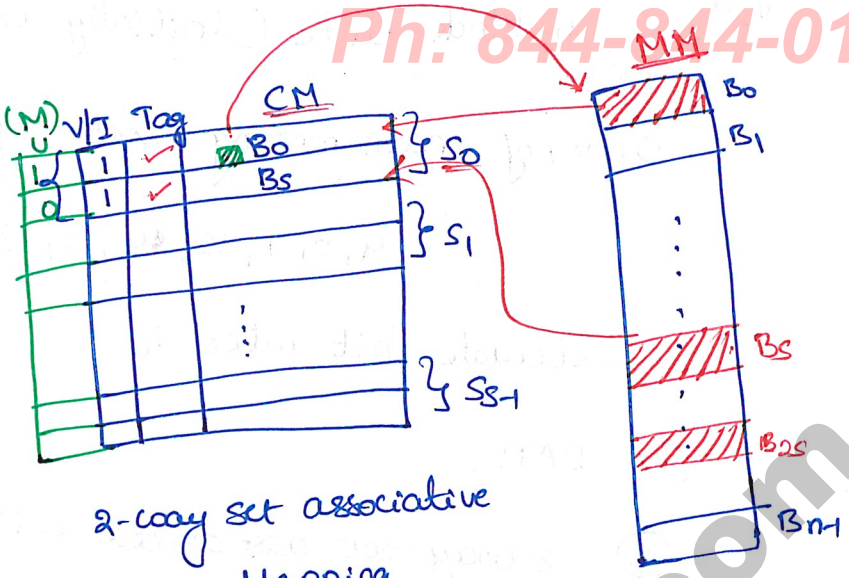
## Updation Techniques:

→ update / Modify bit

### Cache - coherance:

→ Write - Through

→ Write-back

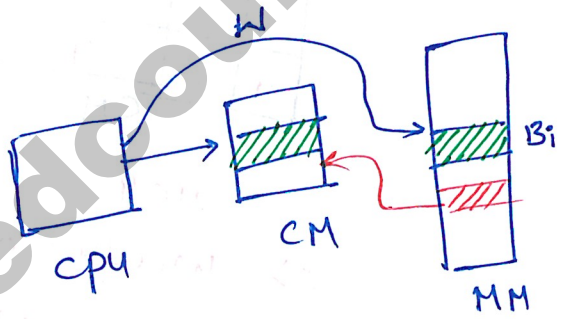



2-way set associative Mapping

Any Edit / updations in Cache memory need to update in Main memory.

addr: Bo, Bs, B2s, Bi
W R R

### Write - Through



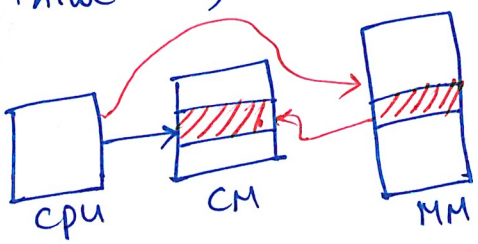

$T_W = \max(CM_W, MM_W)$

↳ Simultaneous Write

Expected time

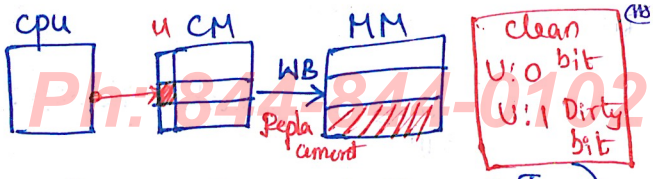
$T_{avg-r} = H_r T_r + (1-H_r)(T_{cr} + T_{mr} + T_{alloc})$ → which block is to be Replaced

$T_{avg-W} = \underbrace{H_W T_W}_{\text{Hit}} + \underbrace{(1-H_r)}_{\text{Miss}} (T_{Alloc} + T_W) + T_{mw}$




$$T_{avg-WT} = f_r * T_{avg-r} + f_w * T_{avg-W}$$

## Write-back :






$$T_{avg-r} = H_r T_{cr} + (1-H_r)\left[ P_d * (T_{WB} + T_{alloc} + T_{mr} + T_{cr}) + P_c * (T_{alloc} + T_{mr} + T_{cr}) \right]$$

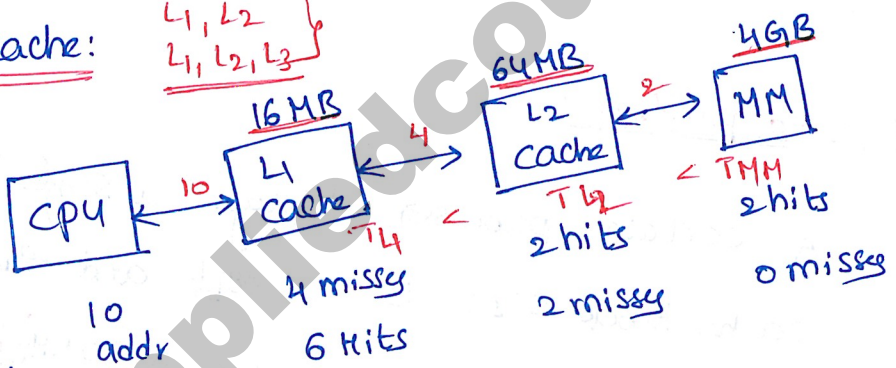
c : clean bit
d : dirty bit

$$T_{avg-w} = H_w T_{cw} + (1-H_w)\left[ P_d (T_{WB} + T_{alloc} + T_{mr} + T_{cw}) + P_c (T_{alloc} + T_{mr} + T_{cw}) \right]$$

$$T_{avg-WB} = f_r T_{avg-r} + f_w T_{avg-w}$$

## Multi-Level Cache:

$L_1, L_2$
$L_1, L_2, L_3$




## Miss-penalty :

Time taken to service a miss

## Global-miss-rate :

$\rightarrow$ (# addr gen) $\frac{4}{10}$ $\quad \frac{2}{10}$ $\quad \frac{0}{10}$

## Local-miss-rate $\rightarrow$ $\frac{4}{10}$ $\quad \frac{2}{4}$ $\quad \frac{0}{2}$

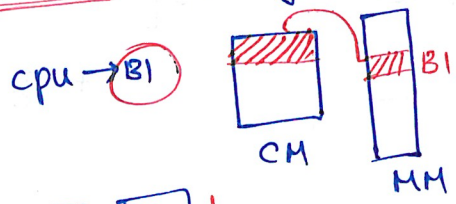
$$T_{avg} = (H_{L_1} * T_{L_1}) + \text{Miss rate}_{L_1} * \text{Miss-penalty}_{L_1}$$

$$\text{Miss-penalty}_{L_1} = (H_{L_2} * T_{L_2}) + \text{Miss rate}_{L_2} * \text{Miss-penalty}_{L_2}$$

$$\text{Miss-penalty}_{L_2} = T_{MM}$$

## Types of cache-Misses:

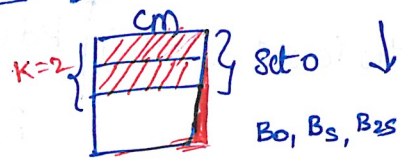
① Compulsory / Cold-start / First-ref miss ↓ — size of block ↑




② Capacity - Miss — size of cache ↑

CM ⇐ MM ↓ — K-way SA ↑

③ Conflict-Miss




Initially cache is Empty. we need to bring the data from MM and store in CM ⇒ Compulsory Miss
cold-start

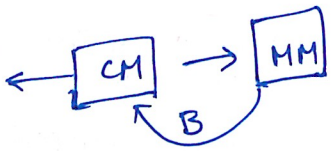
## More Solved problems-1

① 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 system when the program running on the processor issues a series of read operations $= 160 \times 10^6$ bytes/sec.

⇒ 160

$f = 60\,MHz \rightarrow 10^6 = M$

$t = \frac{1}{60M}$





best-case

$8W = 1 + 3 + 8 \times 1$

$= 12$ Cycles

$4B \times 8N \rightarrow 12$

$32B \rightarrow 12 \times \frac{1}{60M}$ sec

$= 160$ MB/s

(2) A certain processor uses a fully associative cache of size 16 KB, The cache block size is 16 bytes. Assume that the main memory is byte addressable and uses a 32-bit address. How many bits are required for the tag and the index fields respectively in the address generated by the processor?

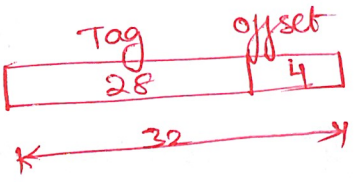
(A) 24 bits and 0 bits

(B) 28 bits and 4 bits

(C) 24 bits and 4 bits

(D) 28 bits and 0 bits.

16KB 16B




Cache Index

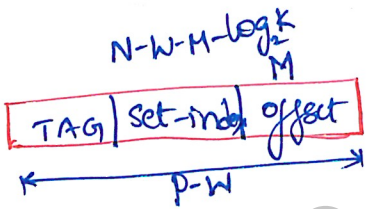
(3) The size of the physical address space of a processor is $2^P$ bytes. The word length is $2^W$ bytes. The capacity of cache memory is $2^N$ bytes. The size of the each cache block is $2^M$ words. For a K-way set-associative cache memory, the length (in number of bits) of the tag field is

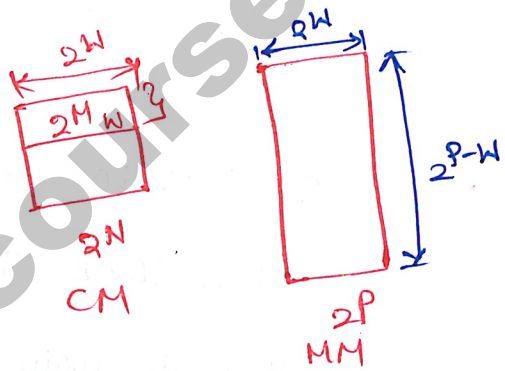
(A) $P - N - \log_2 K$

(B) $P - N + \log_2 K$

(C) $P - N - M - W - \log_2 K$

(D) $P - N - M - W + \log_2 K$




$CM_N = 2^{N-W}$

$2^{N-N-M}$ = # Blocks in CM

$$\text{\# sets} = \left(\frac{2^{N-W-M}}{K}\right)$$

$\Rightarrow N - W - M - \log_2 K$

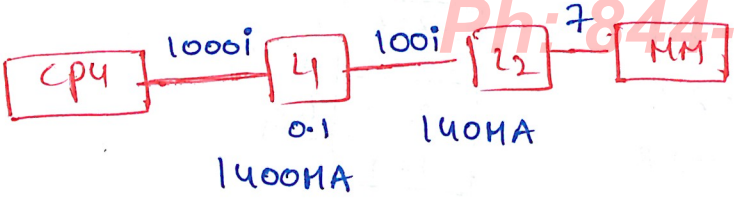
$\Rightarrow P - W - N + W + M + \log_2 K - M$

$P - N + \log_2 K$

(4) Consider a two-level cache hierarchy L1 and L2 caches. An Application incurs 1.4 memory accesses per instruction on average. For this application, the miss rate of L1 cache 0.1, the L2 cache Experience on average 7 misses per 1000 instructions. The miss rate of L2 expressed correct to two decimal places is 0.05

1.4 MA → 1 instruction





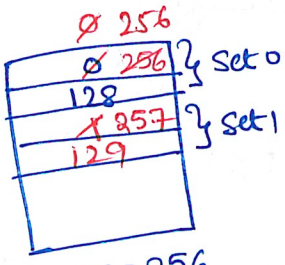

Global Miss-rate

$$LMR = \frac{7}{140}$$

$$= \frac{1}{20}$$

$$= 0.05 = 5\%$$

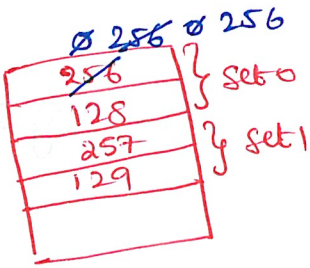
(5) Consider a two-way set associative cache with 256 blocks and uses LRU replacement, Initially the cache is Empty. Conflict misses are those misses which occur due to the contention of multiple blocks for the same cache set. Compulsory misses occur due to first time access to the block. The following sequence of accesses to memory blocks: (0, 128, 256, 128, 0, 128, 256, 128, 1, 129, 257, 129, 257, 129) is repeated 10 times. The number of Conflict misses experienced by the cache is ———




m = 256

S = 128

256 both Compulsory & conflict miss




1: 0, 256, → 2

2: 0, 256, 0, 256, → 4

1, 129, 1, 129 → 4 } 8

3rd Iteration

1 → 4
2 → 8
3 → 8
⋮
10 → 8 ⇒ 9×8+4 = 72+4 = 76 Conflict misses

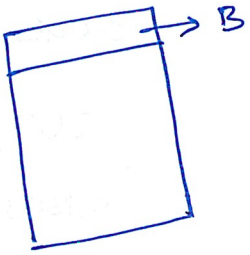
| | |
|---|---|
| 256 | $S_0$ |
| 128 | |
| 257 | $S_1$ |
| 129 | |

(6) A cache memory unit with a capacity of N words and block size of B words is to be designed. If it is designed as a direct mapped cache, the length of the TAG field is 10 bits. If the cache unit is now designed as a 16-way set associative cache, the length of the TAG field is _____ bits.

DAM:-

PA

| TAG | Line# | offset |
|---|---|---|
| 10 | $\log_2(\frac{N}{B})$ | $\log_2 B$ |




Number of Lines = $\frac{N}{B}$

S.A:

| TAG | Set No | offset |
|---|---|---|
| x | $\log_2 \frac{N}{B \times 16}$ | $\log_2 B$ |

$$10 + \log \frac{N}{B} + \log B = x + \log \frac{N}{16B} + \log B$$

$$10 + \log \frac{N}{B} = x + \log \frac{N}{B} - \log 16$$

$$10 + 4 = x$$

$$\Rightarrow x = 14$$

## More Solved problems-II

① In a two-level cache system, the access time of L1 and L2 are 1 and 8 clock cycles, respectively. The miss penalty from the L2 cache to main memory is 18 clock cycles. The miss rate of L1 cache is twice that of L2. The average memory access time (AMAT) of this cache system is 2 cycles. The miss rates of L1 and L2 respectively are

A. 0.111 and 0.056
B. 0.056 and 0.111
C. 0.0892 and 0.1784
D. 0.1784 and 0.0892

1C 8C
CPU — L1 → L2 — MM
x x/2 Misp: 18C

$$2 = 1 + x(8) + x \cdot \frac{x}{2} \cdot 18$$

$$\Rightarrow 9x^2 + 8x + 1 = 2$$

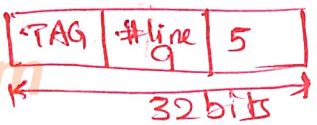
$$x = \frac{2}{18} = 0.111$$

② Consider a machine with byte addressable memory $2^{32}$ bytes divided into blocks of size 32 bytes. Assume a direct mapped cache having 512 cache lines are used with this machine. The size of the tag field in bits ____

(A) 12 (B) 16 (C) 18 (D) 24

32 − 14 = 18

DAM



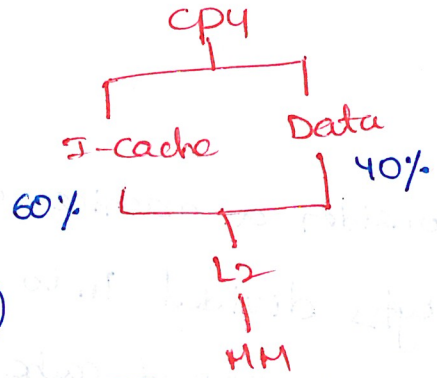
(3) The read access times and the hit ratios for different caches in a memory hierarchy are as given below.

Accounts for Hit rate

Hit rate * read time

| Cache | Read access time (in Nanoseconds) | Hit ration |
|---|---|---|
| I-cache (Instruction) | 2 | 0.8 |
| D-Cache (Data) | 2 | 0.9 |
| $L_2$-Cache | 8 | 0.9 |

The read-access time of main memory is 90 nano seconds. Assume that the caches use the referred-word-first read policy and the write back policy. Assume that all the caches are direct mapped address. Assume that the dirty bit is always 0 for all the blocks in the caches. In execution of a program, 60% of memory reads are for instruction fetch and 40% are for memory operand fetch. The average read access time in nano seconds (upto 2 decimal places) is _____




Instr: $2 + 0.2(8) + 0.2 \times 0.1(90)$

$= 2 + 1.6 + 1.8 = 5.4$ ns

Data: $= 2 + 0.1 \times 8 + 0.1 \times 0.1 \times 90$

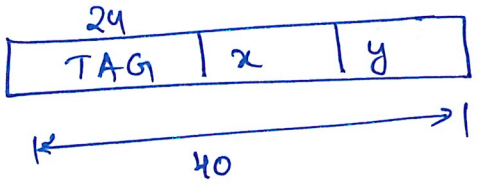
$= 3.7$ ns

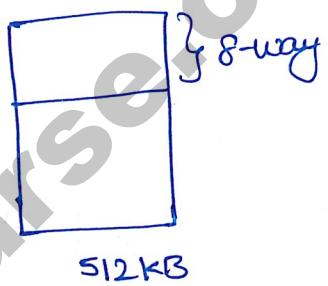
$= 0.6 \times 5.4 + 0.4 \times 3.7$

$= \boxed{4.72 \text{ sec}}$

(4) The width of the physical address on a machine is 40 bits. The width of the TAG field in a 512KB 8 way set-associative cache is _____

(A) 24 (B) 20 (C) 30 (D) 40




$\Rightarrow 2^{19}$

$\Rightarrow 2^{19} = 2^x \times 2^3 \times 2^y$

$\Rightarrow 19 = x + y + 3$

$\Rightarrow x + y = 16$

$\Rightarrow \boxed{40 - 16 = 24}$

# sets $= 2^x$

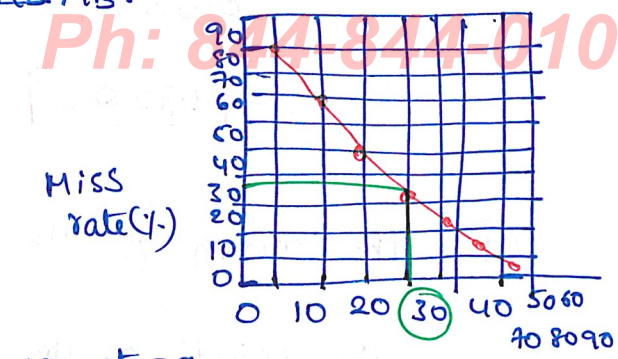
Block size $= 2^y$

(5) A file system uses an in-memory cache to cache disk blocks. The miss rate of the cache is shown in the figure. The latency to read a block from the cache is 1 ms and to read a block from the disk is 10 ms. Assume that the cost of checking whether a block exists in the cache is negligible. Available cache sizes are in multiply of 10MB. The smallest cache size required to ensure an average

(124)

read Lateney less than 6ms is ___30MB MB.

| Cache | DISK |
|---|---|
| 1MS | 10ms |




miss-rate = x

Hit-rate = 1-x

$\Rightarrow (1-x)1 + x(10)$

$9x + 1 < 6ms$

$x < 0.5556$

$\Rightarrow 55.56\%$

$\Rightarrow$ 30MB

(6) Assume that for a certain processor, a read request takes 50 nanoseconds on a cache miss and 5 nano seconds on a cache hit. Suppose while running a program, it was observed that 80% of the processor read requests result in a cache hit. The average read access time in nano seconds is ____

(A) 10

(B) 12

(C) 13

(D) 14

$.80 \times 5 + 0.2(50)$

$= 4 + 10$

$= 14$

(7) In designing a Computer cache System, the cache block (or cache Line) size is an important parameter. which one of the following statements is correct in this context?

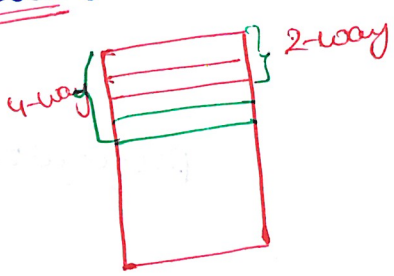
✗(A) A smaller block size implies better Spatial locality.

✗(B) A smaller block size implies a smaller cache tag and hence lower cache tag overhead.

✗(C) A smaller block size implies a Larger cache tag and hence lower cache hit time.

✓(D) A smaller block size incurs a lower cache miss penalty
↳ MM
CM ↲

(8) If the set associativity of a processor cache is doubled while keeping the capacity and block size unchanged, which one of the following is guaranteed to be NOT affected?

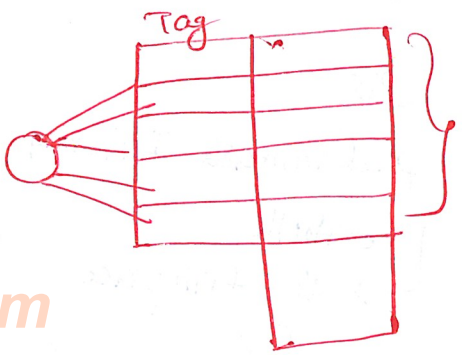



✗(A) width of TAG Comparator

✗(B) Width of set index decoder

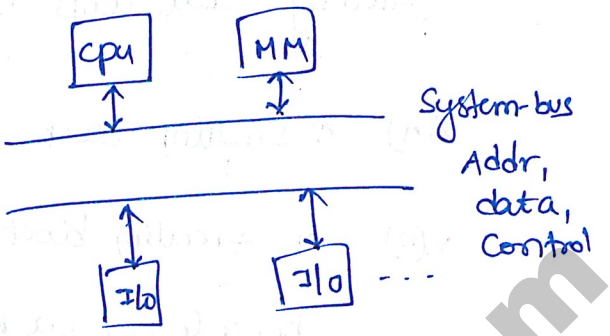
✗(C) Width of way selection multiplexers.

✓(D) Width of processor to main memory data bus.
word-size



# I/O Introduction:

## Input-output-organization




## I/O Transfer-modes

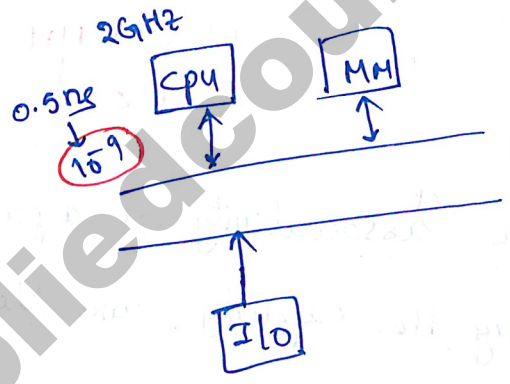
1. Programmed I/O
2. INT- Driven I/O
3. DMA

Keyboard
Mouse
printer

LAN/WIFI
DISK
⋮

## Programmed I/O:

$I_1$ : ..
$I_2$ : LDA I/O-addr
$I_3$ : ...
$I_4$ : ...



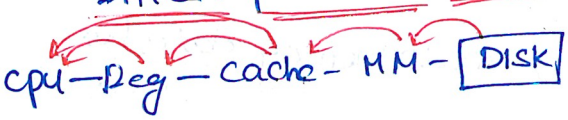

1W = 1D
1KB → 1Sec
1B = $\frac{1}{k}$ Sec
= 1ms

$ET_{prog-I/O}$ = 1ms

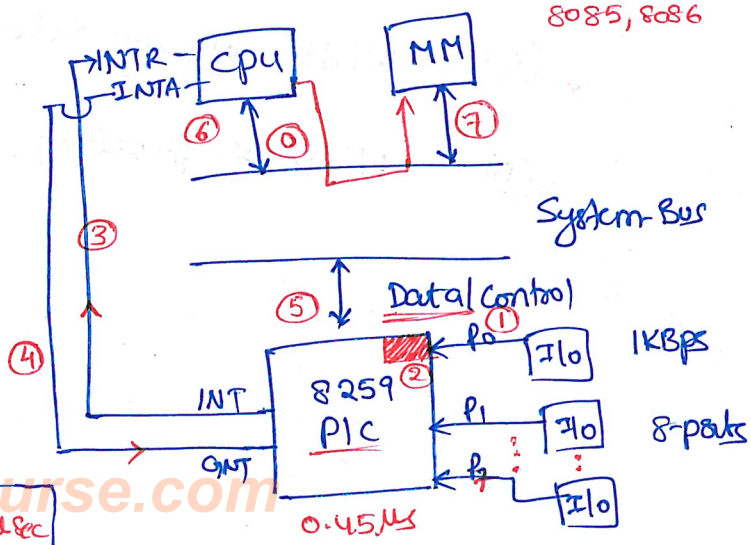
$10^3$B

## CPU-wait-time

## Interrupt-driven I/O:




8085, 8086

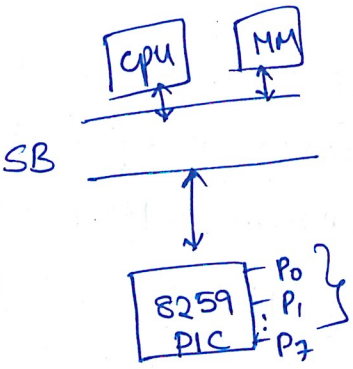


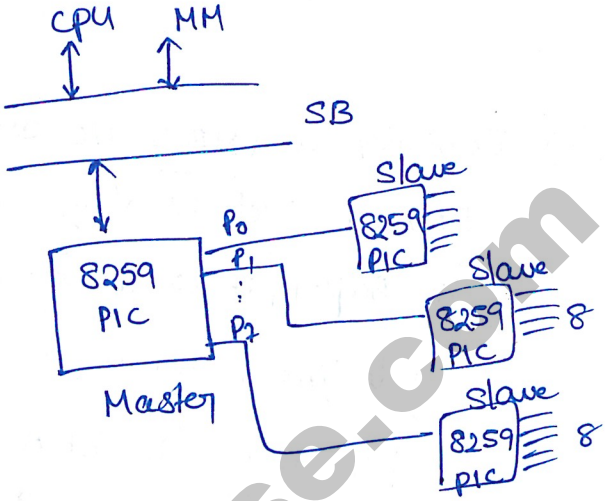

**PIC**
Programmable Interrupt Controller
→ I/O Interface

$ET_{INT-I_O}$ = 0.45μsec

Problem: HDD ⟶ MM

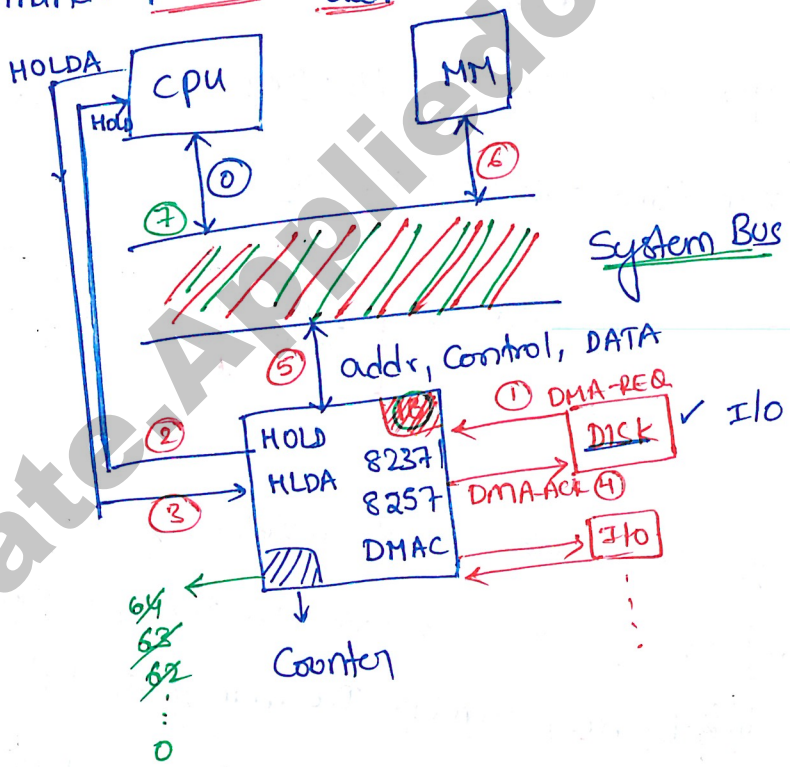
## Single-buffer vs Cascading Modes




## DMA (Direct Memory Access):

I/O Command: port #, mem addr (64), count, control-signals



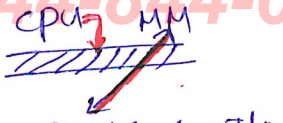

## DMA-Modes:

Cycle-stealing: Byte by Byte word by word

Burst-Mode: Batch Mode / Bulk mode

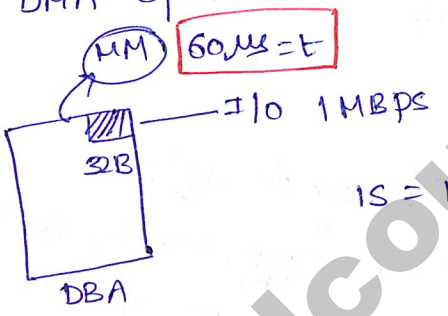
Timings:

* Preparation-time (cpu-busy-state)
* Transfer time (cpu-Blocked-state)




(1) Consider a 1MBPS I/O device interfacing with a cpu using DMA in cycle-Stealing mode. whenever 32B of data present in the Buffer, it is transferred to the main memory. Main memory cycle time is 60 micro-sec. percentage of time cpu is in Blocked state during the DMA operation is _____

Sol:




1S = 1MB ⇒ 1B = $\frac{1}{1M}$ S

= 1 μs

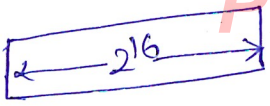
preparation time = 32 μs - Busy ⇒ 32B = 32 μs

Trans time = 60 μs ⇒ Blocked

$= \frac{60}{92} = \frac{60}{(60+32)}$

(2) The size of the data count register of a DMA Controller is 16 bits. The processor needs to transfer a file of 29,154 KB from the disk to main memory. The memory is byte addressable. The minimum number of times the DMA Controller needs to get Control of the System bus from the processor to transfer the file from the disk to main memory is 456

$2^{16}$




64KB

1 → 64 KB

? ← 29154KB

$\left\lceil \frac{29154}{64} \right\rceil$

$\Rightarrow \lceil 455.53 \rceil$

$\Rightarrow \boxed{456}$