a=2
2+3 → instructions.
(optcode)

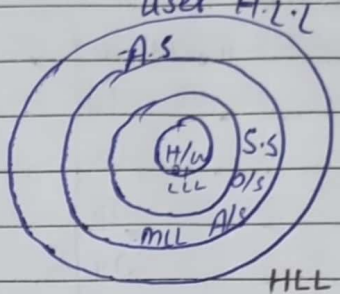
System Software → embedded Programming.
↓
Programming to run hardware

DATE ___/___/20___

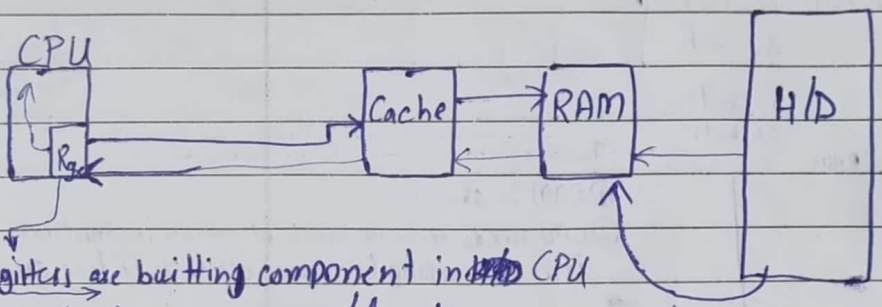
COAL

C.P.Language

H.L.L → A.L.L → M.L.L (AL) → L.L.L (01 Binary)



HLL
Sr. code
*.c
↓ compile
Assembler *.asm
↓ Assembler
*.obj
↓ Linker (*.obj + *.lib)
*.exe Machine Code
Hardware.



14
Registers are building component in CPU
Fastest storage area / location
Quickly accessible by CPU. (cuz they are built-in CPU)

Execute: Hard Disk ← *.exe Machine Code
RAM → Hardware.

Registers (General purpose)

- accumulator (a)
- ① accumulator extended (ax)
- ② base (bx)
- ③ counter/code (cx)
- ④ data (dx)

GP

Segment:
- code segment
- data "
- segment
- extended segment

Special:
- instruction Pointer ip
- stack Pointer sp

Index:
- Source Index (si)
- Destination Index (di)

Flag Register:
- f

Base Register:
- b

Efficiency of CPU → Accumulator: Input/Output, Operations.
Base: Holds address of Data.
Counter: Used in loop
Data: Holds data for output.

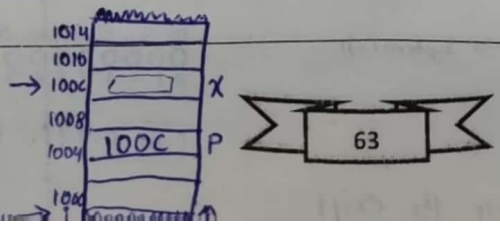
Source Index: Point the source Operand.
Destination " : Points the destination Operand.
Instruction Pointer: Holds the next instruction.

Code Segment: Holds addresses of code segments. Stack Pointer: Points the current top of the stack
Data Segment: Holds address of data segments
↳ used for regular data storage.
Stack Segment: Holds " of stack managment.
Extra segment: Holds address of data segment.
↳ Provides additional memory addressing capability beyond the limitation of the Data Segment in 16 bit real mode.

Flag Register: Holds current status of Program
Base Pointer: Base the top of stack

borrow
Carry
1
1 0 1
1 1 1
0

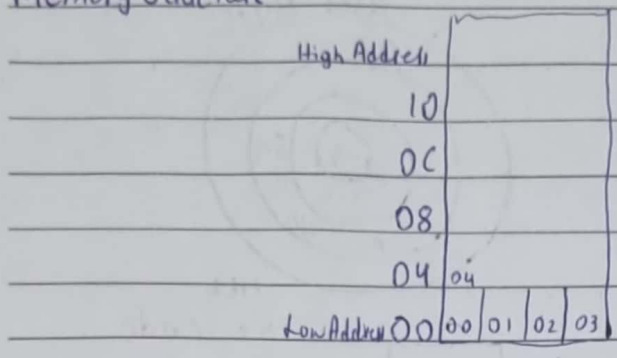

```
int x;
int *p;
x = 25;
p = &x;
```



storage high speed slow.

Pause
Pose
DATE __/__/20__

Memory Structure



scheme
Lower toward higher

0103 0A
0102 00
0101 05
0100 B8

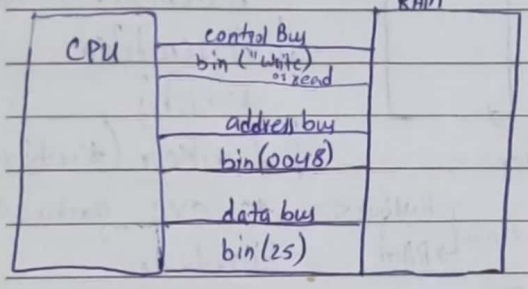
L to H: B80500
H to L: B80005

int c ← B80500

little endian

Big Endian

int x; //address 0048
x = 25;



Two Problems:
① RAM is slow!
② CPU needs to keep track of which instruction it is executing (plus some other stuff like errors)

[org 0x0100] First 100 chucks are reserved for DOS OS.
org → origin ↳ for cornel.

In general, all are based on:
Von-Neumann Architecture:
-) Code (instructions) & data both stored in memory
-) General Purpose
-) Fetch, decode, execute.

First Code → ax = 5
register [org 0x0100]
instruction ← mov ax, 5 → operand.
mov bx, 10
add ax, bx ∴ ax = ax + bx

mov bx, 13
add ax, bx

mov ax, 0x4c00 ; exit... (command)
int 0x21 ; ... is what the OS should do for me.
↳ interrupt

Request Pause the Programe.
then exit
Request Signal → System Call
Application (User Space) ↓ OS (Kernal Space)
↳ to Process the exit.

mount c c:/MP
c:
edit file.asm

nasm file.asm -o File.com
(CPU Kae brain me)
afd file.com

Press F1 for next step

nasm file.asm -l file.lst -o file.com
edit file.lst

B8 04 00 BB 03 00 01 D8
0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7

(mi 0100 To Bx in index.

GO GETTERS

DATE ___/___/20___

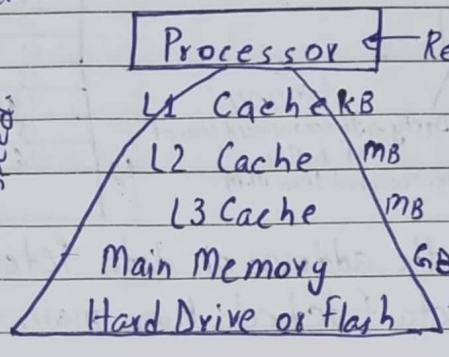
Memory Hierarchy Design

delay ↑ Sharing of memory by layering.

- Programmer want unlimited amount of memory with low latency.
- Fast memory technology is more expensive per bit than slower memory
- Solution: Organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller & faster memories, each containing a subset of the memory below it, proceed in steps up towards the processor.
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the illusion of a large, fast memory being presented to the Processor

With respect to time ↓ P ← [ ] ← HDD

With respect to space ↓ Latency/Size ↑ increase. Speed ↓



P ← (1) ← (L2) ← L3 ← RAM ← H

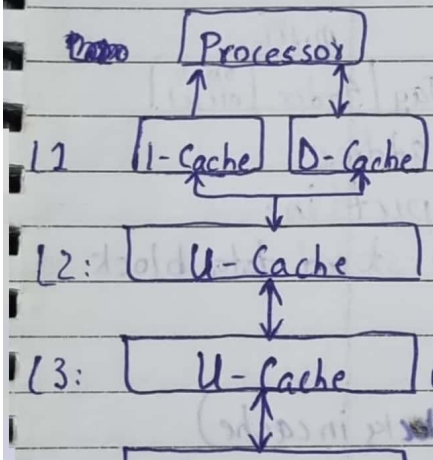
PC = PC+1

300 ← PC, IR, ACC

↓ Address

Holds the address of next inst to be executed.

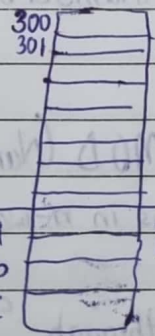
a+b ↓ Data



icache: Instruction Cache D-Cache: Data cache.

U-Cache: Unified Cache

Different functional units fetch information from I-cache and Dcache: decoder and scheduler operate with I-cache but integer execution unit and floating point communication with D-cache.



hit: when data recieved or task complete

Miss: If data not found

if CPU asked some data but can't find it.

When a word is not found in the cache, a miss occurs.

- Fetch word from low level in hierarchy requiring high latency reference
- Low level maybe cache or the main memory.

GO GETTERS

Miss Rate:

↳ Fraction of cache access that result is a miss

↳ Causes of miss (3C's)

- Compulsory
- Capacity
- Conflict.

Compulsory: First ref to a block - would happen even for infinite cache.

Capacity: Blocks discarded & later retrieved.

Conflict: Program make repeated references to multiple addresses from different blocks that map to the same location in the cache.

Placement Policies:

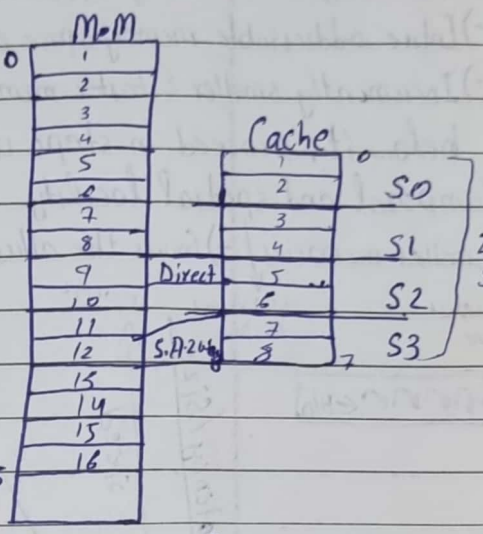
- Main memory has a much larger capacity than cache
- ★ Mapping between main and cache memories.
- ★ Where to put a block in cache.

DATE ___/___/20___

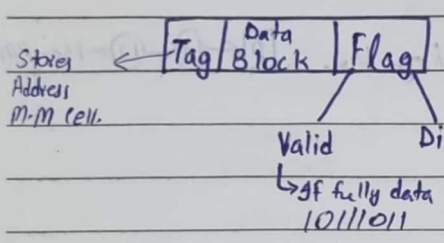
How cache can communicate with Main Memory

3-Approaches to match cache memory with main

① Direct Map
② Set Associative — 2 way, 4 way, 8 way.
③ Full Associative.
↳ Whole cache will go to one chunck of M.M.
↳ A block can be placed in any location in cache.

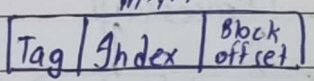


Cache Row:



Tag → Contains part of the address of data fetched from main memory.
Data bloc → contains data fetched from main memory.
Flag → Valid, dirty

Memory address is split (MSB to LSB) into 3



Tag: contains the most significant bits of the address.
index: gives the cache row the data has been put in.
block offset: gives the desired data within the stored data block within the cache row.

Direct Map Formula: (Block Address) MOD (Number of blocks in cache)

$12 \text{ MOD } 8 = 4$

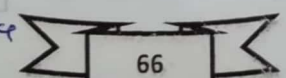
Set Associative Formula: (Block Address) MOD (Number of sets in cache)
↳ A block can be placed in one of n locations in n-way set associative cache.

Writing to cache: two strategies
- Write-through
  ↳ Immediatly update lowerlevel of hierarchy
  latency: cache & memory (update)
- Write-back
  ↳ Only update lowerlevels of hierarchy when an updated blocked is replaced.

- Both strategies use write buffer to make writes asynchronous.

Q) Why write buffer? So CPU doesn't stall.
Q) Why buffer, why not just one register. Burst of writes are common.

GO GETTERS



Q) Are read after write (RAW) hazard an issue for write buffer?
Yes! Drain buffer before next read or can read data from buffer.

DATE ___/___/20___

Q) You have 32 blocks of M.M and 16 blocks cache determine the cache block to map with 10 block of MM. using (2 way) S.A -

$10 \% 8 = 2$

Hit: data appears in some block in the upper level (example: Block X)
Hit Rate: the fraction of memory access found in the upper level.
Hit Time: Time to access the upper level which consist of RAM access time + Time to determine hit/miss

- Hit Time << Miss Penalty (500 instruction on 21264)

Miss: data needs to be retrieved from a block in the lower level (Block Y)

- Miss Rate = 1 - (Hit Rate)
- Miss Penalty = Time to replace a block in the Upperlevel + Time to deliver the block the processor.

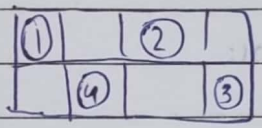
A[ ] = 5
↓ Memory

Age = 20;
↑ Address
Variable

dw : define words
↳ size is 16 bits/2 bytes.

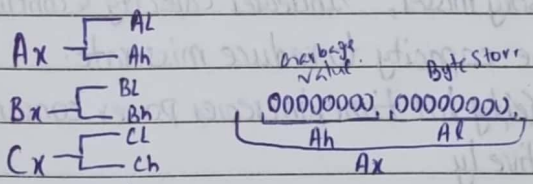
db: define bytes.
↳ to define space, specific space as per needed

[org 0x0100]



dw → 0005
db → 0A05

mov ax, [num1]
mov bx, [num1+1]
add ax, bx



first loader finds the global variable.

mov bx, [num1+2]
add ax, bx

Mov [num1], [num2] , illegal
we can't assign memory to memory
Aik wat mae aik hi address carry kr ga

mov [num1+3], ax

xor ax, ax
↳ to clear garbage value.

mov ax, 0x4c00
int 0x21

(ZF) Zero Register: tells either it's 0 or 1
↳ tell the status of program
→ code executed if ZF=0 (result is not zero)
→ code executed if ZF=1 (result is zero).

GO GETTERS

num1 : dw 5, 10, 8, 0
↓
Global variable: In AL, global variable are typically accessed using memory addressing - The memory address of global variable is known at compile time and the variable is accessed using its address

How global variable seen by assembly language

DATE ___/___/20___

Listing: ~~Practice~~

nasm work.asm -l work.lst -o work.com
edit work.lst

List
- to see real address of memory.

Hit rate: fraction found in that level.
So high that usually talk about MissRate.

Average memory-access time = HitTime + Miss rate x Miss Penalty (ns or clocks)

Miss Penalty: time to replace a block from lower level, including time to replace in CPU
access time: time to lower level = f(latency to lower level)
transfer time: time to transfer block = f(BW between upper & lower levels, block size)

Improve Performance by: 1) Reduce the miss rate.
2) Reduce the miss penalty, or
3) Reduce the time to hit in the cache.

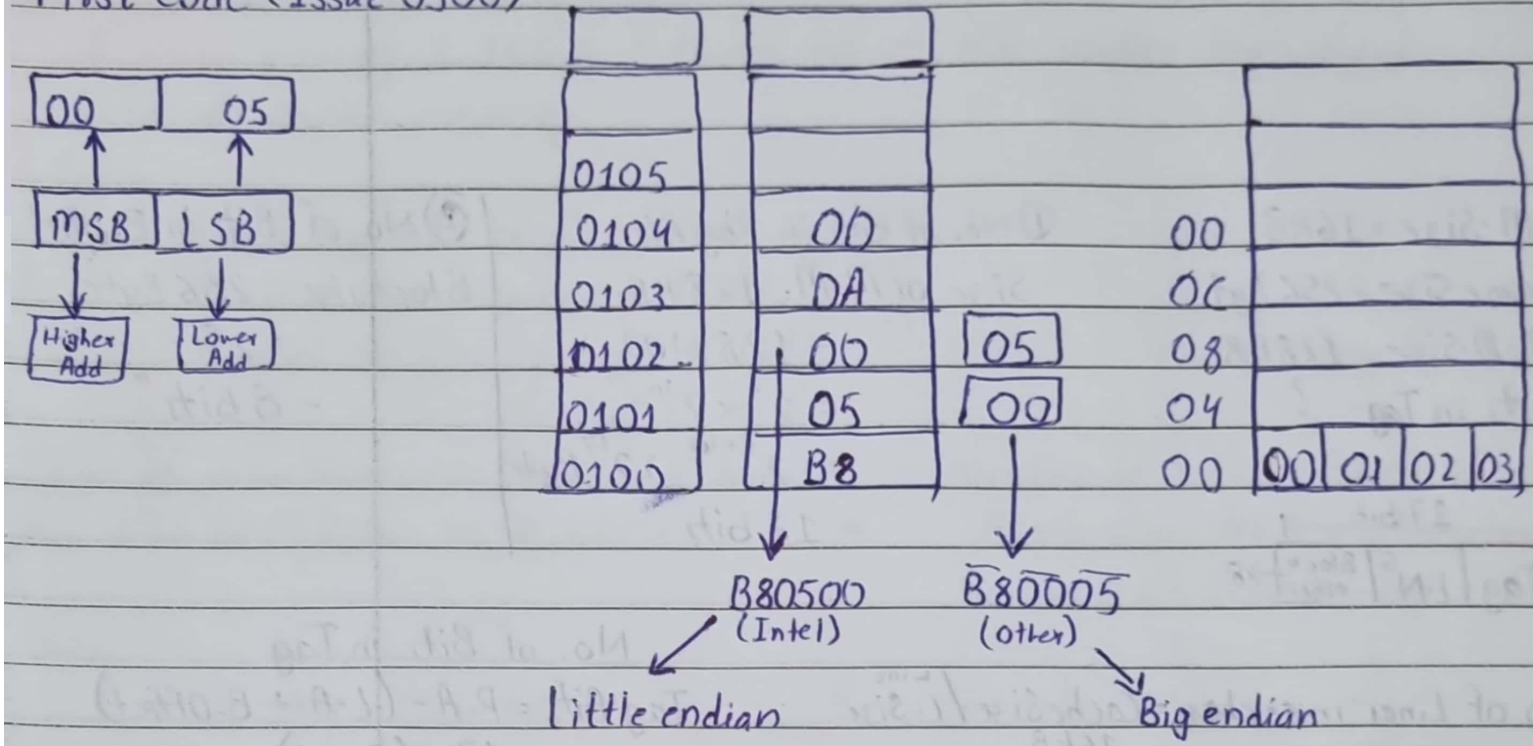
Six basic cache optimizations:

- Larger block size
  ↳ Reduce compulsory misses, Increases capacity & conflict misses, increases miss penalty.
- Larger total cache capacity to reduce miss rate.
  ↳ Increases hit time, increases power consumption
- Higher associatively
  ↳ Reduce conflict misses Increases hit time, increase power consumption.
- Higher number of cache levels.
  ↳ Reduces overall memory access time
- Giving priority to read misses overwrites.
  ↳ Reduce miss penalty.
- Avoiding address translation in cache indexing
  ↳ Reduce hit time.

GO GET

DATE ___ / ___ / 20___

First Code (Issue 0500)



#Assemble and run

```
nasm c.asm -o c.com
afd c.com
```

#Assemble with listing

```
nasm c.asm -l c.lst -o c.com
```

DATE __/__/20__

Numerical:

(Q1) Consider a direct map

CM Size = 16kB
Frame Size = 256 Byt
M.M Size = 128kB
Bits in Tag = ?

① No. of Bits in Phy Add
Size of M.M = 128 kB
= 128 × 100
= $2^7 \times 2^{10}$
= $2^{7+10} \Rightarrow 2^{17}$ Bytes
= 17 bits

② No. of Bits in B. offset
Block size = 256 Bytes
= 2^8
= 8 bits

17 bits

Tag	L.N^G	Block offset → 8

No of Lines in cache = Cache Size / L. Size (Line)
= 16kB / 256 Bytes
= $2^4 \times 2^{10} / 2^8 \Rightarrow 2^6$ Bytes
= 6 Bits

No. of Bits in Tag
Tag Bits = P.A − (L.A + B. Offset)
= 17 − (8 + 6)
= 3 Bits

(Q.2)

Number of Bits in Block offset
Block Size = 1kB = 2^{10} bytes
= 10 bits

Number of bits in Line Number
Line in cache = Cache Size / Line Size
= 512kB / 1kB
= 2^9 lines
= 9 bits.

P.A = 7 + 9 + 10 = 26 bit

M.M = $2^{26} \Rightarrow 2^{16} \times 2^{10}$
65000 × 1000
$2^{10} \times 2^{10} \times 2^6$
K × K × 2^6
64 kB × kB ⇒ 64MB

∴ kB × kB = MB
$10^3 \times 10^3 = 10^6$

GO GETTERS

7.3) Consider a direct mapped cache with block size 4kB - The size of main memory is 16GB & there are 10 bits in the tag. Find Size of cache memory.

Solution:

Block size = frame size = line size = 4kB

Size of main memory = 16GB

Number of bits in tag = 10 bit

$(10^9 = \text{Giga (GB)})$

$2^4 \times 2^{10} \times 2^{10} \times 2^{10}$

Size of main memory = 16GB = 2^{34} bytes

Num of bit in PhyAddress = 34 bits

Number of Bits in Block offset

Block size = 4kB = 2^{12} bytes $2^2 \times 2^{10}$

Number of bits in block offset = 12 bits.

34 - (10+12)

No of lines = 12 bits

Size of Cache Memory =

Tot num of lines in cache x Line Size

$2^{12} \times 4$kB

$2^{12} \times 2^{12} = 2^{10} \times 2^{10} \times 2^4 = 2^{24}$

= 16 MB

Problem # 06

Cache Memory Size = 8kB

Block Size = Frame Size = Line Size = 32 bytes

Bits in Physical Address = 32 bits

Bits in Blockoffset:

32 bytes = 2^5

5 bits

Bit in Line Number:

$\frac{\text{Cache memory size}}{\text{Line Size}} \Rightarrow \frac{8\text{kB}}{32\text{ bytes}} \Rightarrow \frac{2^{13}\text{ bytes}}{2^5\text{ bytes}}$

2^8 lines $\Rightarrow$ 8 bits

Bits in Tag:

32 - (8+5)

19 bits

Memory Size Needed At Cache Controller =

Number of lines in Cache x (1 Valid bit + 1 modified bit + 19 bits to identify bloc

$2^8 \times 21$ bits

5,376 bits

Option 4 is correct

DATE ___/___/20___

Set Associative:

Problem-01: Consider a 2-way set associative mapped cache size of 16kB with block size 256 bytes. The size of main memory is 128 kB. Find Number of bits in tag.

Sol: Given: SetSize = 2, Cache memory size = 16kB

Block size = Frame Size = Line Size = 256 bytes.

P.Ad = 17 | Tag | Line set (5) | B.off (8)

Bit of Phy Add:

MM Size = 128 kB = 128×2^{10}

$= 2^7 \times 2^{10} = 2^{17}$

Bit of Phy Add = 17 bit

Bits of Block offset:

Block Size = 256 B

$= 2^8$ B = 8 B

No. of line in cache:

Cache/line Size = $16kB/256B = 16 \times 2^{10}/256$

$= 2^4 \times 2^{10}/2^8 = 2^6$

= 64 lines.

Total No. of sets in cache:

Cache line/set No = $64/2 \Rightarrow 32$ sets

$= 2^5 \Rightarrow 5$ bit

Tag Bits = Phy Add − [(Line S) + (B.off)]

= 17 − (5+8) = 4 Bits

Problem: 07: Consider a direct mapped cache with 8 each block (0-7). If the memory block request are in the order.

3, 5, 2, 8, 0, 6, 3, 9, 16, 20, 17, 25, 18, 30, 24, 2, 63, 5, 82, 17, 24.

Which of the following memory blocks will not be in the cache at the end of the sequence? (1) 3 (2) 18 (3) 20 (4) 30

Also calculate the hit ratio & miss ratio.

	Cache Memory	
0	8, 0, 16, 24	
1	9, 17, 25, 17	3%8 = 3, M
2	2, 18, 2, 82	5%8 = 5, M
3	3	2%8 = 2, M
4	20	8%8 = 0, M
5	5	0%8 = 0, M
6	6, 30	6%8 = 6, M
7	63	3%8 = 3, H
		9%8 = 1, M
		16%8 = 0, M

H.R = 3/21 | M.R = 18/21

18 will not be in the cache.

GO GETTERS

LRU: Least Recent Use

(P# 10,11) → Assignment.

DATE ___/___/20___

Problem08:

In 5 Block of cache → 7 of M Block

0	4,45
1	8,22
2	25
3	8
4	19,3
5	6,7
6	16
7	35

H·R = 5/17 M·R = 12/17

Problem09: 16/4 ⇒ 4

0	0,48		0%4=0, M
1	4,32	0 set	255%4=3, M
2	8		1%4=1, M
3	216,92		4%4=0, M
4	1,		3%4=3, M
5	133	1 set	8%4=0, M
6	129		133%4=1, M
7	73		159%4=3, M
8			216%4=0, M
9		2 set	129%4=1, M
10			63%4=3, M
11			8%4=0, H
12	286,155		48%4=0, M
13	3	3 set	32%4=0, M
14	159		73%4=1, M
15	63		92%4=0, M
			155%4=3, M

H·R = 1/17 M·R = 16/17

216 is not present
So D is correct

GO GETTER

latency = delay Sequential = More time. Parallal = Less Time.

DATE ___/___/20___

Pipelining:

Sequentially it will take more time, while pipelined take less time / reduce time.

Throughput: No. of unit produced in a unit time.

I = instruction

Stage
- S_1 = IF (instruction fetch) — Fetching instruction from memory.
- S_2 = ID (instruction decode) — it decodes, read source regx & gen- generate control signals.
- S_3 = Ex (Execution) — compute an R-type outcome. Eg: Performing A+B
- S_4 = MEM (Memory) — Read & Write the memory.
- S_5 = WB (write Back) → in HD — Write back to hard drive store a result in destination reg.

Q)

	IF	ID	EX	WB
I_1	2	1	2	3 = 8
I_2	1	2	3	2 = 8
I_3	1	2	2	1 = 6
I_4	3	1	3	2 = 9

$InP_T = 31$ 0.C
↳ sequential

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
IF	I_2	I_1	I_2	I_3	I_4	I_4	I_4								
ID			I_1	I_2	I_2	I_3	I_3	I_4							
EX				I_1	I_1	I_2	I_2	I_2	I_3	I_3	I_4	I_4	I_4		
WB						I_1	I_1	I_1	I_2	I_2	I_3	-	-	I_4	I_4

$P_T = 15$
↳ After Pipeline.

Criteria for checking the performance of pipeline.

Speed-up: how much faster the pipeline execution is as compared to non-pipelin execution.

$$\text{Speedup} = \frac{\text{Non-pipeline execution time}}{\text{Pipeline exe time}} \qquad S.P = \frac{\uparrow P_t}{P_t}$$

efficiency: Accurate result in less time (kam time mae kitna acha output aata hai)

$$\text{efficiency}(n) = \frac{\text{Speed up}}{\text{no. of storage in pipline Archtuve}} \Rightarrow \frac{\text{Speed up}}{\text{Total no. of Phase.}}$$

Throughput: how many task you achieve in a unit time.

$$\text{Throughput} = \frac{\text{No. of inst executed}}{\text{Total time taken.}}$$

Non-Pipeline	Pipeline (k stages)
$T = n * t_n$	$T_k = (k+n-1) * t_p$

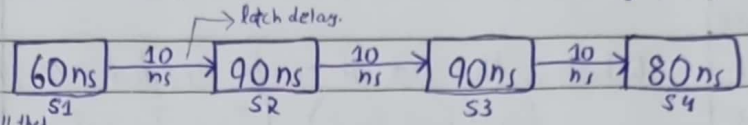
t_p: Clock cycle (time to complete each suboperation)

GO GETTERS

Frequency of the clock (f) = $\frac{1}{\text{Cycle Time}}$

DATE ___/___/20___

Q.) Consider a pipeline having 4 phases with duration 60, 50, 90, 80 ns - Give latch delay is 10ns - Calculate (1) Pipeline cycle time (2) Non-Pipeline exc time (3) Speedup ratio (4) Pipeline time for 1000 tasks (5) Sequential time 1000 task (6) through put.



→ we take max cuz till that every other would be completed.

(1) P_t = Max delay due to any stage + delay due to i5 regx

max [60, 50, 90, 80] + 10

$P_t = 100ns$

(2) $nP_t = \Sigma(\text{all stage})$

$= \Sigma(60+50+90+80)$

$nP_t = 280ns$

(3) Speed-up ratio = $S.P = \frac{npt}{P_t} \Rightarrow \frac{280}{100}$

$S.P = 2.8ns$

(4) Pipeline time 1000 task

P_t of 1 task + Time taken for 999 task

1 × 4 clock cycle + 999 × 1 c.c

$4 + 999 \Rightarrow 4 \times 100 + 999 \times 100$

$400 + 99900 \Rightarrow 100300n$

(5) Sequential Time (Non-Pipline)

1000 × nP_t for one task

1000 × Time taken for one task

1000 × 280 ns

280000ns

(6) Through put : T.p = $\frac{\text{no. of inst executed}}{\text{Total time}}$

No. of inst per unit time executed.

Pipeline = $\frac{1000}{100300}$

Non Pipeline = $\frac{1000}{280000}$

Q) Consider an inst pipeline with four stages (S1, S2, S3 & S4) each with combinational circ only. The pipeline registers are required b/w each stages & at the end of the last stage. Delay for the stage & for the pipeline registers are as given in the figure.

$nP_t = 30ns$

$P_t = \text{Max}[5, 6, 11, 8] + 1$

$= 11 + 1$

$P_t = 12n$

Speed-up = $\frac{nP}{P} \Rightarrow \frac{30}{12}$

$= 2.5$

Pipe exe time = 1 clock cycle

= 12 ns.

GO GETTERS

1.25×10^{-3} — 100
1.67×10^{-3} — x
x = 133.6
33.6 → increase

DATE ___/___/20___

(Q.) Consider a 4 stages pipeline processor. The number of cycle needed by the four instructions J1, J2, J3 & J4 in stage S1, S2, S3 & S4 is shown below.

	IF	ID	EX	WB
I_1	2	1	1	1
I_2	1	3	2	1
I_3	2	1	2	3
I_4	1	2	2	2

What is the no. of cycle needed exc the following loop?
for(i=1 to 2) [I1 ; I2 ; I3 ; I4]

Loop 1 , Loop 2 ,

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}
S1 I	I_1	I_1	I_2	I_3	I_3	I_4	I_1	I_1	I_2	I_3	I_4												
S2 D			I_1	I_2	I_2	I_2	I_3	I_4	I_4	I_1	I_2	I_2	I_2	I_3	I_4	I_4							
S3 E				I_1			I_3	I_2	I_3	I_3	I_4	I_4	I_1	I_2	I_2	I_3	I_3	I_4	I_4				
S4					I_1				I_2		I_3	I_3	I_3	I_4	I_4	I_1	I_2	I_3	I_3	I_5	I_4	I_4	

Problem: 4

1st stage : P_t = Max{800, 500, 400, 300} + 0
= 800 picoseconds
throughput = 1/800

2nd Stage : P_t = Max{600, 300} + 0
= 600 Picosec
throughput = 1/600

Throughput increase
[(Final throughput - initial throughput)/initial throughput] x 100
[(1/600 - 1/800)/(1/800)] x 100
33.33%

Problem: 9

IF, ID & WB stage take one clock cycly. EX depends on instruction, ADD & SUB need 1 clock cycle & MUL need 3 clockcycle in EX stage

	1	2	3	4	5	6	7	8
IF	A	M	S					
ID		A	M	S				
EX		·	A	M	M	M	S	
WB				A			M	S

8 clock cycle.

DATE __/__/20__

Interrupt:

- Hardware
  - Maskable → Non-Priority based.
  - Non-maskable → Priority Based Interrupt
- Software (exceptions)

•) An unscheduled event that disrupts program execution is called exception

•) An exception that come outside of computer is called interrupt.

In many computers exception are called interrupts but more precisely.

-) Exception is a signal from a program that requires OS to decide what to do next

-) While interrupt is a signal from a ~~signal from a~~ device attached to computer.

Why Interrupts?

Device & program needs CPU but when? We can't predict

Each device or program can give a signal to CPU in the form of interrupt.

CPU can determine which program or device needs it's attention.

Hardware Interrupts

↳ A signal to CPU through external device or hardware. Eg: key stroke, Mouse Movement

Maskable: This type of interrupt can be delayed by the processor.

Non-Maskable: This type of interrupt can not be delayed by the processor.

Software Interrupt (Exception)

It is caused by a special instruction or exceptional condition

E.g: Divide a number by zero, access to unavailable memory.

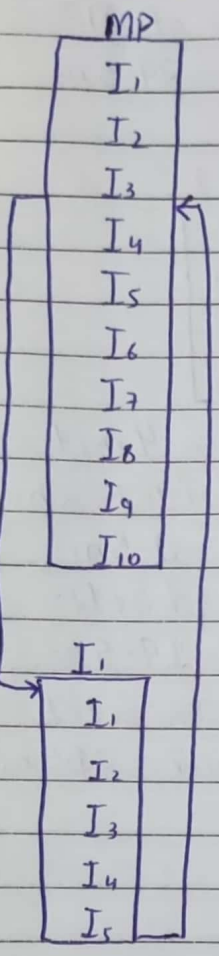
Interrupt Handler: A piece of code that is run as a result of an exception or interr

Multiple Interrupts: An interrupt event when processor is already busy in handling other interrupt.

GO GETTER

DATE ___/___/20___

Q) Consider a main processor consist on 10 instructions & required 20ms mili sec for execution. Find out the size of processor, time required ~~by~~ for each instruction to executed, waiting time for 1 inst, 3 inst & 6 inst.



Size of M.P = 10 inst
T. MP = 20ms
T/inst = $\frac{20}{10}$ = 2ms
Waiting of I_2 = 0ms
" " I_3 = $I_1 + I_2$ = 2+2 = 4ms
" " I_6 = 5×2 = 10ms

Q) Using the above numerical, an interrupt occurs during the execution of third instruction of main process, the size of interrupt is 5 instruction. Each instruction of interrupt takes 1ms.

Size. Inst = 5 inst
T/Inst = 1ms
Total. T = 5ms
Waiting time I_{intup} = 0ms

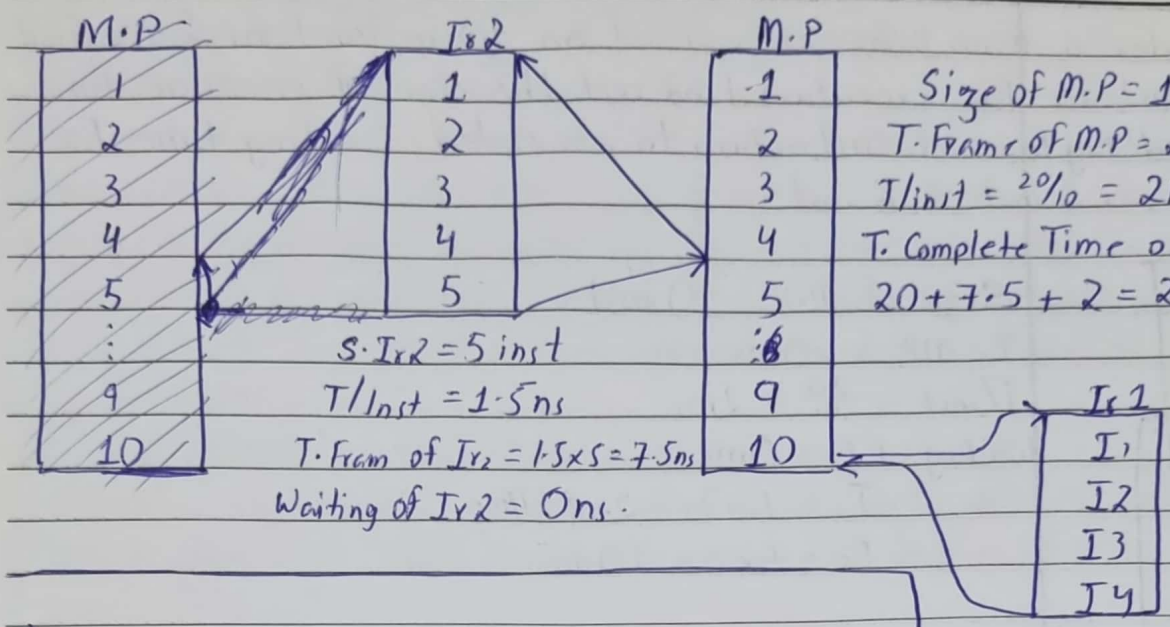
W.T of I_6 = (5×2)+5 = 15ms
" of I_4 = (2+2+2)+5 = 11ms
T. Comp. T. M.P = 20+5 = 25 m.s

Q) Consider a process of 10 instruction in time frame require in 20ns At the execution of 4th instruction 2 interrupts occured - Interrupt number 2 is non-maskable having 5 instruction, each inst takes 1.5ns for execution - Interrupt 1 consist of 4 insts with time frame 2ns - Do the necessary calculation.

GO GETTERS

maskable → after MP
non-maskable → on priority.

DATE ___/___/20___

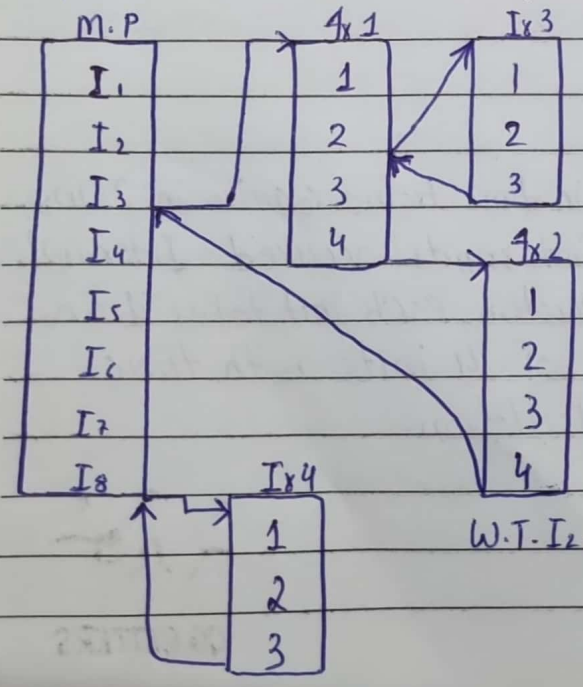


S.Ix2 = 5 inst
T/Inst = 1.5ns
T.Fram of Ix2 = 1.5×5 = 7.5ns
Waiting of Ix2 = 0ns.

Size of M.P = 10 inst
T.Frame of M.P = 20ns
T/inst = 20/10 = 2ns.
T. Complete Time of MP:
20 + 7.5 + 2 = 29.5ns

S.Ix1 = 4 inst
T.Frame of Ix1 = 2ns
T/Inst = 0.5ns
W.T.Ix1 = 7.5+12 =
= 19.5ns.
Total Complete Time of Ix1
19.5ns+2 = 21.5ns

Q) Consider a M.P of 8 inst each inst requires 1.5ms during the execution of inst 3 two intrupts occur both intruppt are non-maskable both are of some size having 4 inst with TimeFram 4ms During the execution of 7th inst of M.P another intruppt occur with size 3 inst having T.F 6ms In the execution of intruppt 1, at second inst a mini intrupt (non maskable) occur with size 3 inst & each inst required 1.5ms dr



W.T Ix1 = 4.5ms

W.T.MP = 4+4.5+4+6
= 18.5ms

T.Complete T.MP = 18.5+12
= 30.5ms

W.T.I2 = 4+4.5 = 8.5ms

W.Time Ix4 = 1.5ms

GO GETTER

DATE ___/___/20___

Pipeline Hazards: is a situation in which next instruction cannot execute in the following clock cycle.

There are 3 types of hazards.

(1) Structural Hazard: Cannot execute an instruction combination due to unavailabily of a resoaure. If MIPS had only one memory - The same memory was used to: •) fetch instruction •) Load/store data.

1	IF	ID	Exe	Mem	Wb	
2		IF	ID	Exe	Mem	Wb
3			IF	ID	Exe	Mem Wb
4				IF	ID	Exe Mem Wb

We have had a structural hazard with 4 inst
-) First inst wants to get data from memory
-) Fourth inst want to fectch inst from mem.
Without two memories, we will experiance a structural Hazard.

Structural dependency occurs due to resource conflict, when different instruction collide while trying to access the same resource in the same segment of the pipeline, then it cause structural hazard.

S. Hazard are minized using hardware technique is used called renaming, which splits the memory into two independent sub-modules to store instruction and data separation.

↱ RAW

(2) Data Hazard: An occurrence in which a planned instruction cannot execute in proper clock cycle cuz data that is needed to execute the instruction in not yet available.

Data Hazard occurs due to data dependency, on instruction which is still in pipeline. Example: Sum = a + b

Sub = Sum - c] we cannot take sum as input until the sum operation is not compictly performed. It's a dependency sub will need to wait

1	2	3	4	5
IF	ID	EXE	MEM	WB
	IF	ID	EXE	MEM WB

Sum insts writes value of sum in clock 5
However sub insts reads it in Clock 3.

Solution: Insert nops, or bubbles, No inst fetched during nop
Three nop inserted, Not a good solution as waste of time

GO GETTERS

1	2	3	4	5
IF	ID	EXD	MEM	WB
	Nop	Nop	Nop	Nop

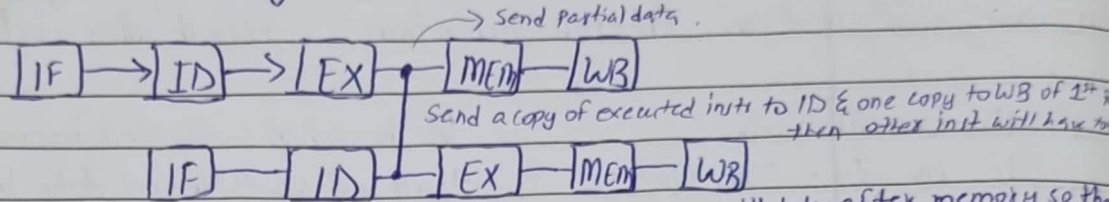
IF ID EXE MEM WB

DATE ___/___/20___

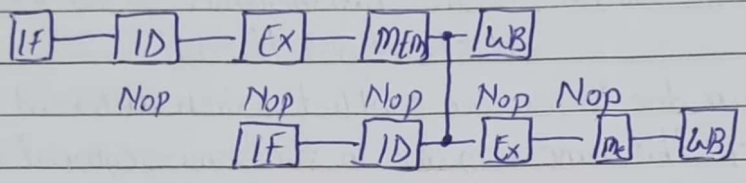
Data Hazard Sol

② Rely on optimizing compilers, can be used but not reliable
(as dependency can be data dependency as well)

Sol③ Implement forwarding or bypassing - It is a method to resolve data hazard by retrieving the missing data element from internal buffer rather than waiting for it to arrive from programmer-visible regi



Forwarding can't solve all stalls, like this one



lw = Load Word Sw = Store Word.

•) Reordering code to avoid Pipeline stalls.

Control Hazard: Operation can't execute in proper clock cycle because th instruction fetched is not the instruction to be executed

Sol①: Update ID phase in such manner that perform that perform following task in ID phase

•) Test register •) Calculate branch address •) Update PC

Even with this we need one stall.

Sol②: Predict about the branch, when prediction is correct, continue as it when prediction is wrong, insert stall.

Sol③ Delay branch decision.

Assembly lang: use memonics to create instruction.

DATE ___/___/20___

Compiler: Convert code as whole batch in A.L
Assembler: Convert that file in asm file
batch Processing/Packet Switching: compiling as whole.
Real time/circuit switching: On spot processing.

Word → 16 bit size
doubleword → 32 bi
quad → 4
d.quad → double of quad. 8
QuadWord → 64
D.QuadWord → 128

Hit & Miss

Occurance of Hit - in a level when the address that an operation reference is found in that level of the memory hierarchy. Otherwise a Miss

Neither the hit rate nor the miss rate count the references that are handled by higher levels in the hierarchy.
For example: request that hit in the cache of our example memory hierarchy donot count in the hit rate or miss rate of the main memory.

Average Access time at a level

$T_{avg} = (T_{hit} \times P_{hit}) + (T_{miss} \times P_{miss})$ T=Time taken P=Probablity.

Hit rate of the lowest level in the hierarchy = 100% (all requests that reach the bottom level are handled by the bottom level)
Starts at the bottom level & work up to compute the avg access time of each level in the hierarchy.

Cache heirachy level → 3 level

(1) Hit rate = 75%, T_{hit} = 12ns, T_{miss} = 100ns

$T_{avg} = (12 \times 0.75) + (100 \times 0.25)$
$= 34ns$

Bottom to top

CPU
Cache
M.M → Virtual Memory
HD

1ms = 1000000ns

DATE ___/___/20___

(1) A memory system contains the cache, main memory, & virtual memory.
the access time of the cache = 5ns , Cache hit rate = 80%.
the access time of the main memory = 100ns , Main Memory hit rate = 99.5%.
The access time of the virtual memory = 10ms

Avg Access Time of M.M = (100ns x 0.995) + (10000000ns x 0.05)
= 50,099.5 ns

Request to reach cache = (5ns x 0.8) + (50,099.5 x 0.2)
= 10.024ns.

A - A ÷ B × B

↑ A mod B

DATE __/__/__

Direct Map ← 17

P1) | Tag (3) | Line No (6) | Block offset (8) |

B.O = 256B ⇒ 2^8

= 8 bits

MM = 128 ⇒ $2^7 \times 2^{10}$

= 17 bits

Line = $2^{14}/2^8 \Rightarrow 2^6$

= 6 bit

Tag = 17 - (6+8)

= 3 bits

P4) S.A

(i) 8-way set associative mapped cache, size of cache memory 512B, 10 bits in tag. Find size of main memory

10	x	y
tag	Line	B. off

Size of M.M = 10 + x + y

= 10 + 16

= 26

$2^6 \times 2^{10} \times 2^{10}$

64MB

Cache Size = B.off × Line × E

$512 = x \times y \times 2^3$

$2^{10} \times 2^9 = 2^x \times 2^y \times 2^3$

$19 = x + y + 3$

$x + y = 16$

P5)

(i) 4-way SA cache. Size of M.M is 64MB

10 bits in tag, Find size of cache memory.

10	x	y
Tag	Line	B.off

2^{26} = 26 bits

$26 = 10 + x + y$

$x + y = 16$

Cache Memory = $2^x * 2^y * 4$

$= x + y + 2$

$= 16 + 2 = 18$

$= 2^{18}$

Cache Mem = 256kB

P07)

0	8,16,24	•
1	1,9,17	17
2	2,18,282	
3	3	•
4	20	
5	5	•
6	6,30	
7	63	

3/21

18/21

P8)

	45	1	H = 5/17
	22	1	M = 12/17
••	25	11	
••	8	11	
	3	1	
	7		
•	16	1	
	35		

P10)

0	808	S0	H = 1/5
• 1	12		M = 4/5
2		S1	
3			

DATE ___/___/20___

SA

(P9)

0	48	S_0
1	432	
2	8	
3	92	
4	1	S_1
5	133	
6	129	
7	73	
8		S_2
9		
10		
11		
12	255 155	S_3
13	3	
14	159	
15	63	

(P11)

FIFO

	42	1	42	1
•	61	1	• 19	11
•••	13	11	••• 13	11
••	19	11	• 43	11
	43		61	1111
•	61		•• 19	111
•••	19		••• 13	1111
•	13		• 43	111

Hit FIFO: 5/15 Hit LRU = 6/15

0	12		0 12	S_0
• 1	43	61	1 42	
2	2		2 5 19	S_1
3	13		3 61	

Hit Direct Map: 1/15 Hit 2-Way SA = 5/15

M. Anas

Quiz CS221004 Date:

Q.) Suppose you have a direct map cache with 8 blocks, if you recieve following memory block request

3, 5, 2, 8, 0, 6, 3, 9, 16, 20, 17, 25, 18, 30, 24, 2, 63, 5, 82, 17

Which of the following block will remain in cache at the end.

(A) 18 (B) 30 (C) 3 (D) 20.

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

A

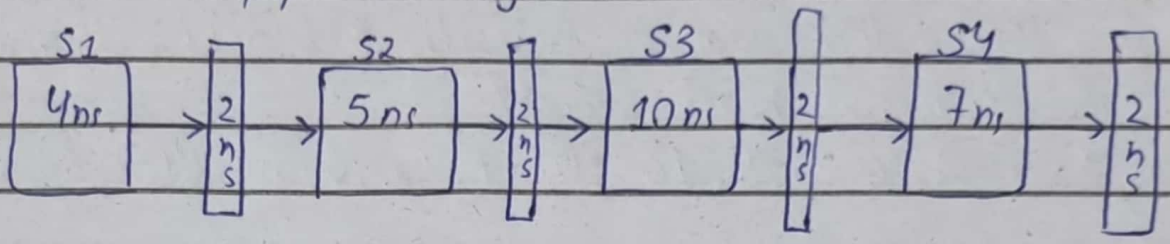
Option A will not remain in cache

Option B, C, D will remain in cache

3%8 = 3,
5%8 = 5
2%8 = 2
8%8 = 0
0%8 = 0
6%8 = 6
3%8 = 3, H
9%8 = 1
16%8 = 0
20%8 = 4
17%8 = 1
25%8 = 1
18%8 = 2
30%8 = 6
24%8 = 0
2%8 = 2
63%8 = 7
5%8 = 5 H
82%8 = 2
17%8 = 1
24%8 = 0, H

Date:

(Q.2) Consider a pipeline system



What will be the approx speed up.

Pipeline

Max Σ(4+5+10+7+) + time

10 + 2

12 ns

Non-Pipeline

Σ(all stage)

Σ(4+5+10+7)

26 ns

Speed up = $\frac{\text{Non-pipeline}}{\text{Pipelined}} = \frac{26}{12}$

$\frac{13}{6}$

= 2.166ns