O.S

S.No	Topics (SubTopics	Date	P. no.
I.	Introduction & Background → Definition → Goals of operating System. → Types of operating System 1. Batch operating System 2. Multiprogramming OS 3. Multitasking OS 4. Multiprocessor Systems 5. Real Time systems.	19/01/19	
II.	PROCESS MANAGEMENT → Definition → Attributes → states & State Diagram → CPU bound & I/O bound → Context Switching → Three schedulers (Long, medium, Short) → Different times (Arrival time, completion Time)		
	CPU SCHEDULING 1.) First come First Serve 2.) Shortest Job first 3.) Shortest remaining Job first 4.) Round Robin Scheduling 5.) Longest Job First 6.) Longest remaining Time first 7.) Priority based Scheduling 8.) Highest response ratio next 9.) Multilevel Queue scheduling 10.) Multilevel Feedback Queue Scheduling	19/1/19 20/1/19	

S.No	Topics (Subtopics)	Date	P.No
	SYNCHRONIZATION	20/1/19	
	→ The Producer - Consumer		
	→ Printer Spooler Daemon		
	→ Basic definition (Critical section, Race condition)		
	→ Conditions required to achieve synchronization.	20/01/19	
	→ Solutions.		
	→ Lock Variables	27/01/19	77
	→ Strict alteration (Decker's Algorithm)		79
	→ Petersons Algorithm		83
	→ TSL Instruction set (Hardware type)		87
	SEMAPHORES		93
	→ Counting Semaphore		94
	→ Binary Semaphore		98
	CLASSICAL PROBLEMS OF IPC		
	→ Producers - Consumers with Semaphore.		107
	→ Readers - writers		112

I. Introduction & Background

II. Process management
- → process concept
- → CPU scheduling
- → synchronization
- → concurrent programming
- → Deadlocks
- → Threads.

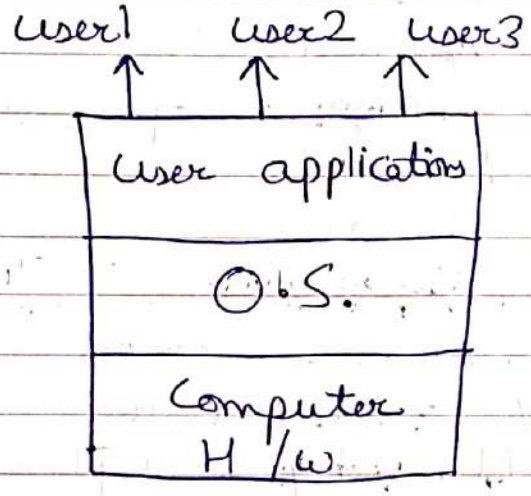
III. Memory management
- → RAM chip implementation
- → Loading, Linking and Address Binding.
- → Techniques
  - → paging
  - → multilevel paging
  - → inverted paging
  - → segmentation
  - → Segmentation paging
- → Virtual memory.

IV. File systems.

INTRODUCTION AND BACKGROUND

Q1.) What is an O.S.?

(1. Ans.) O.S is an interface between user and computer H/w.




```
main()
{
    int x;
    printf ("Hello");
}
```

printf internally calls write() system call in order to communicate with the monitor.

System Call :- System Call is request made by the user program to the operating system in order to get any kind of a service.

→ OS is also called as resource allocator because it is responsible for allocating resources of a computer.

Resources

Hardware Type
E.g: Devices, memory.

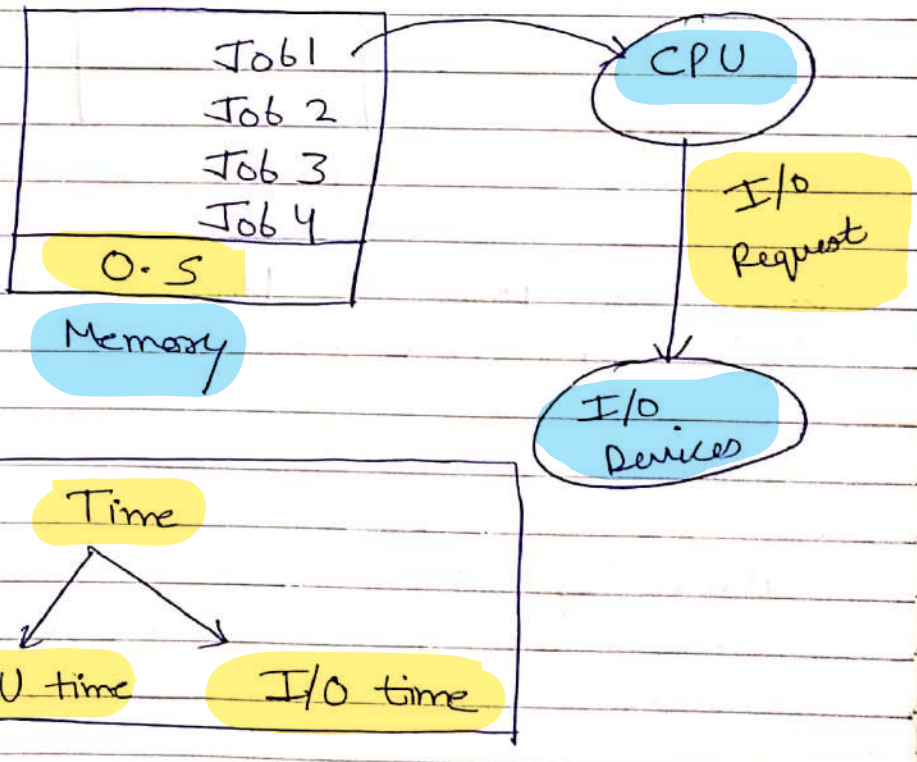
Software Type
Eg. files, directories.

GOALS OF OPERATING SYSTEM:-

1.) The primary goal is convenience.
2.) The secondary goal is efficiency.

TYPES OF O.S:-

1.) BATCH OPERATING O.S:-



→ If the job is completed completely, then only another job will be scheduled onto CPU.

Advantages of Batch O.S:-

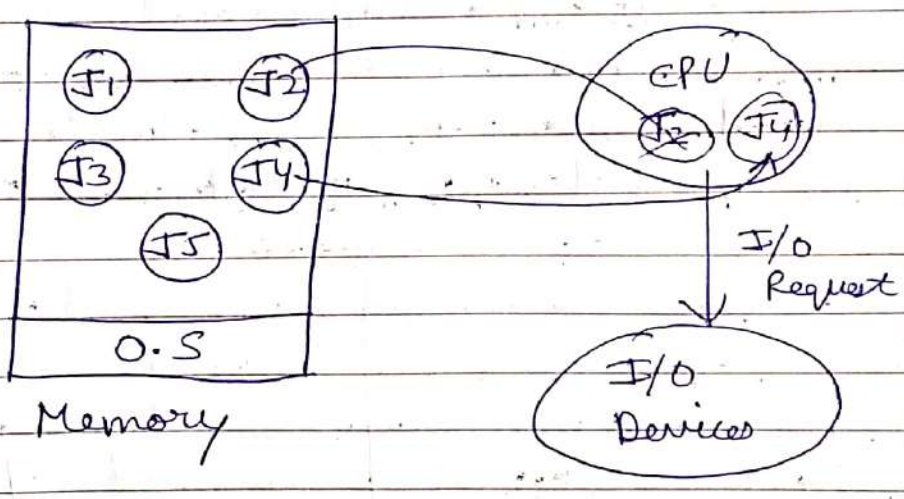
→ Increased CPU idleness.

→ Decreased throughput of the system.

Throughput - The No. of Jobs completed per unit time.

Eg: IBM OS/2.

2.) MULTIPROGRAMMING O.S:-



→ If the job is leaving the CPU to perform I/O operation, then another job, which is ready for execution will be scheduled onto the CPU.

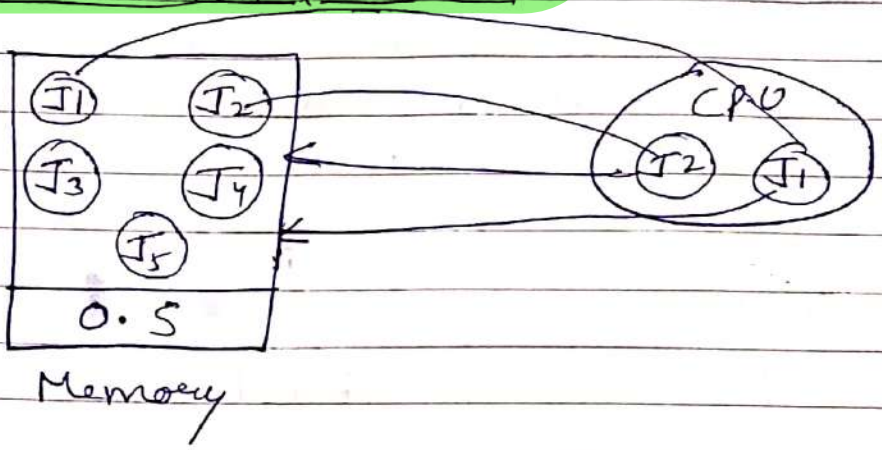
Advantages:-

→ Increased CPU utilization.

→ Increased throughput of the System.

Eg:- Windows, Unix, linux.

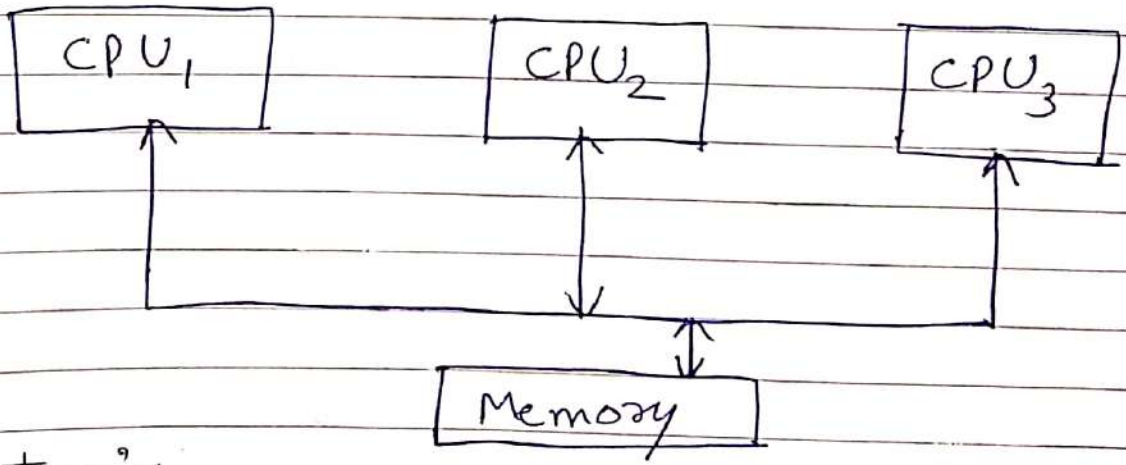
3.) MULTITASKING O.S:-



→ Multitasking is an extension of multiprogramming O.S.
→ The jobs will be executed in the time sharing mode.
→ At any point of time only one job can be done

Eg: Windows, Unix.

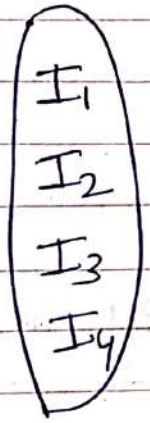
4.) Multiprocessor Systems:-



Advantages:-
→ Increased throughput of a system.
→ Reliability.
↳ fault tolerant systems.
→ Economical. Eg: UNIX.

5.) REAL TIME SYSTEMS:-

The Systems which are strictly deadly time bound are called as real time systems.



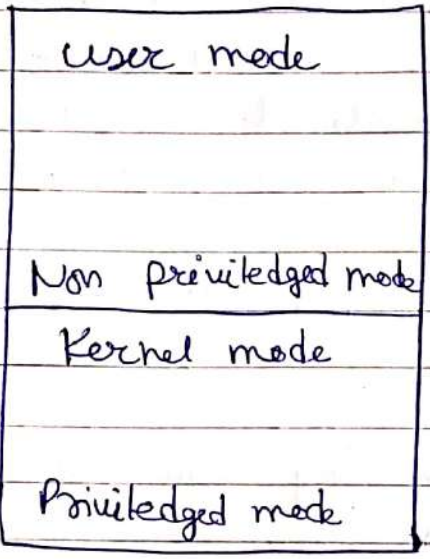
Hard real time	Soft real time
Eg: Satellite systems, missile systems,	Eg:- Banking sector.
Minor delay is not acceptable	Minor delay is acceptable.

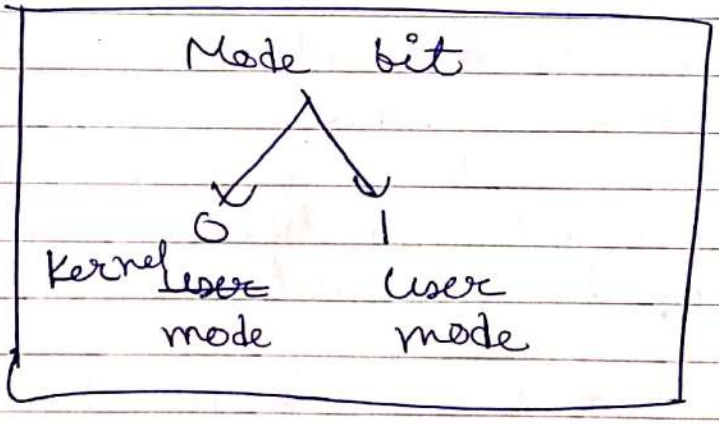
Eg:- S_x walks, V_x walks, RTOS.

"errant users" → unauthorised

OPERATION

DUAL MODE :-





→ In the hardware level, the instructions are executed by using dual mode operation.

1.) User mode / non priviledged mode
2.) Kernel mode / priviledged mode / System mode / monitor mode.

The dual mode operation is used in order to provide protection and security to the user programs and also to the operating system from errant users.

→ It is purely the decision of the O.S, in which particular mode the instruction has to be executed. Generally the priviliged instructions are executing in the kernel mode.

→ Non priviliged instructions are executing in the user mode.

→ In the booting time system always start in a kernel mode.

→ Operating System always runs only in a kernel mode.
The mode bit is used to identify in which particular mode, the current instruction is executing.

NOTE:- The mode switching takes very less time compare to process switching.

(d)

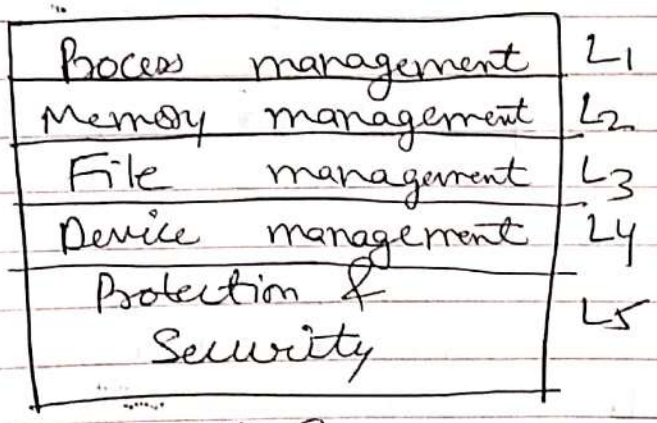
PRIVILEGED INSTRUCTIONS:-

1.) I/O operation
2.) Context switching
3.) Set the time of clock
4.) Disabling the interrupts.
5.) Changing the memory map.
6.) Clearing the memory map.

Non privileged instructions:-

1.) Reading the status of the processor.
2.) Reading the time of clock.

LAYERED APPROACH:-



→ The design and implementation of the O.S. will be done into multiple layers.
→ The main advantage of layered approach is debugging, abstraction and modularity.
→ Modularity means if any specific layer is updated, then it will not have any

(9.) (a) (c) (d)

impact on other layer.

Fork() System call Implementation:-

```
main()
{
int pid;
pid = fork();
if (pid<0)
  {
   printf("child process creation failed");
  }
else if (pid==0)
  {
   printf(" Child process");
  }
else
  {
   printf(" parent process");
  }
}
```

NOTE:- fork is a system call used to create the child process.

⇒ fork returns negative value if the child process creation is unsuccessful.

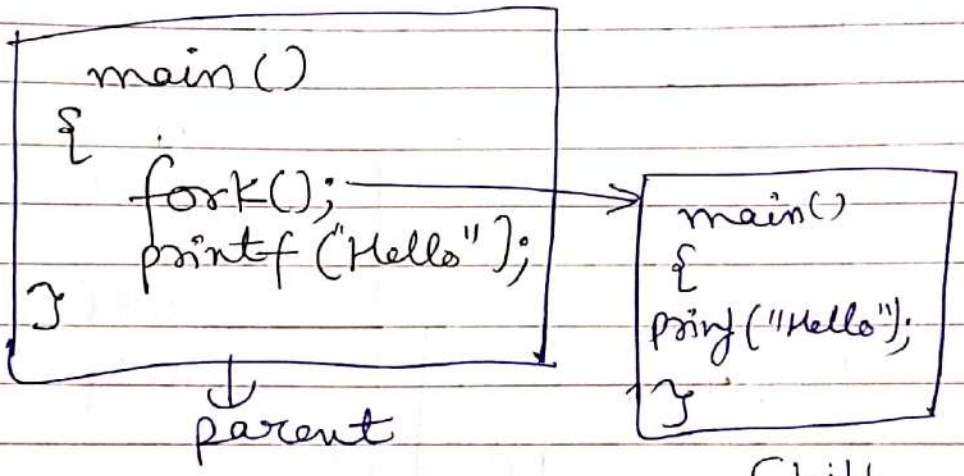
⇒ fork returns value 0 to the newly created child process.

⇒ "fork returns positive value" to the parent process.

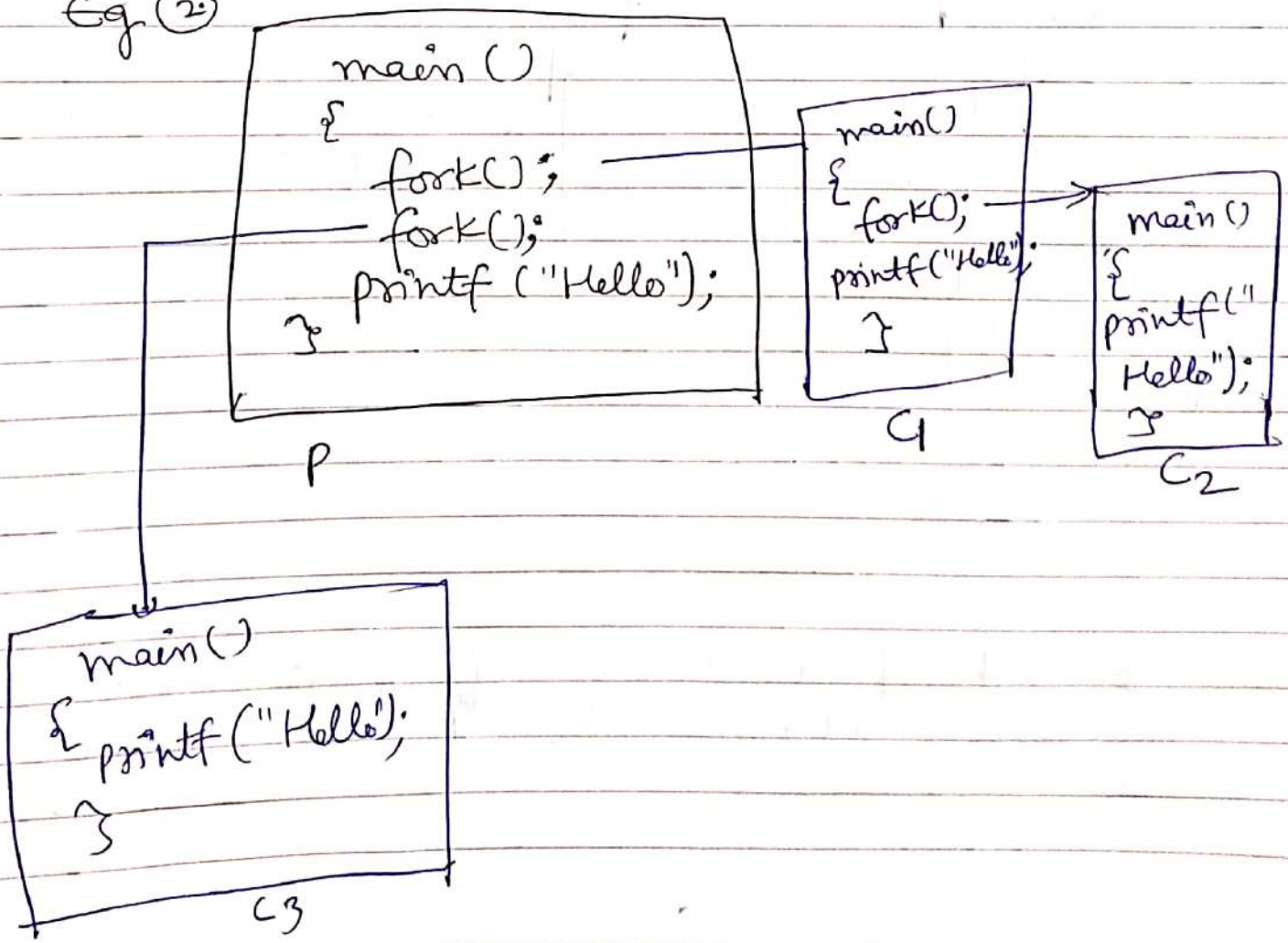
(Process id of a child process)

→ When the child process created, then the new memory location will be allocated to the child process and parent and the child process will have same virtual Address; but physical address will be different.

Eg① :-

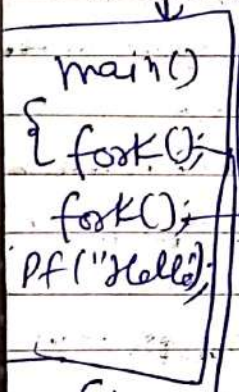


Eg ②

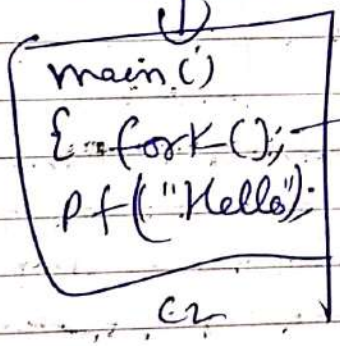


Eg:)

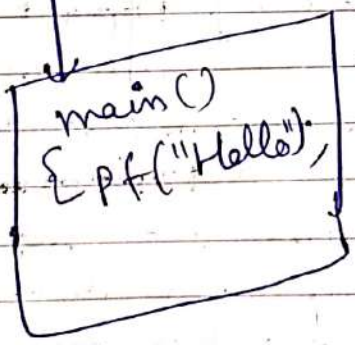
```
main()
{
fork();
fork();
fork();
printf("Hello");
}
```

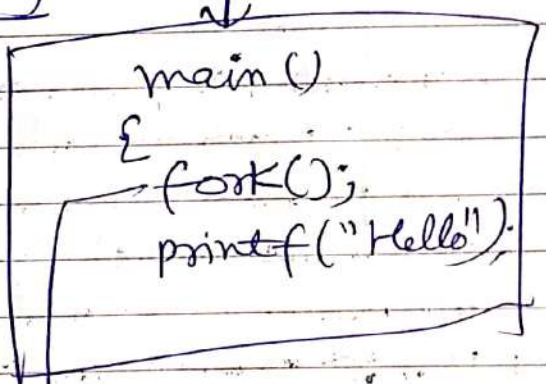


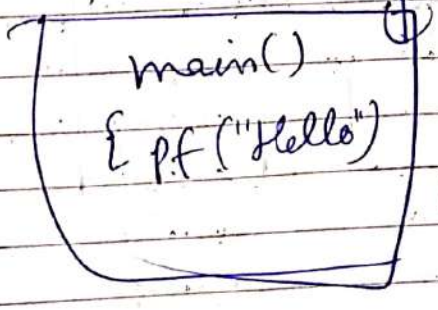
C1



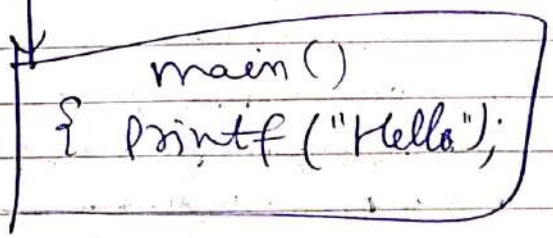
C2







C3



(7)

NOTE:- If the program have N fork calls then it will $2^N - 1$ child process.

PROCESS MANAGEMENT

Definition:- The program under execution is called as process.

- → It should reside in the main memory.
- → It occupied the CPU to execute the Instructions.
- → Active and Dynamic.
  - Active (load on memory)
  - Dynamic (taking Inputs and producing outputs).

Attributes:- Process will have various Attributes:-

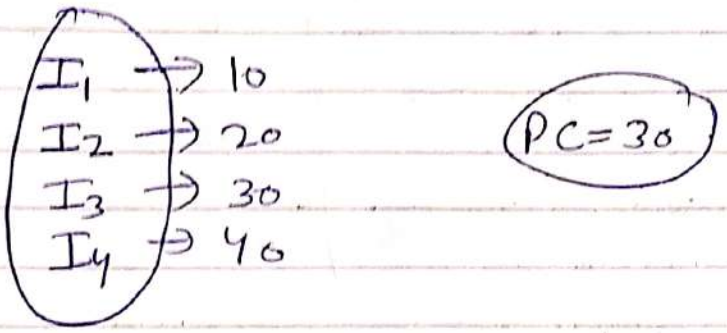
1.) Process id.
2.) Process state.
3.) Priority.
4.) Program counter.
5.) General purpose registers.
6.) List of open files.
7.) List of open devices.
8.) Protection information.

1.) Process id:- Process id is unique identification number, which is assigned by the operating system at the time of process creation.

2.) Process state:- The process state contains current state information of a process, where it is residing.

3.) Priority :- is a parameter assigned by the operation system at the time of Process creation

4.) Program counter :- Program counter contains address of the next instruction to be executed.



---

* All the Attributes of a process is called as context of a Processor.

* The context of a process will be stored in a P.C.B (progess control block).

P.id	P.S
priority	P.C
G.P.R	L.O.F
L.O.D	P.I

PCB.

→ Every process will have its own P.C.B.
→ PCB of the processes will be stored in the main memory.
→ PCB will be implemented by using double linked list.

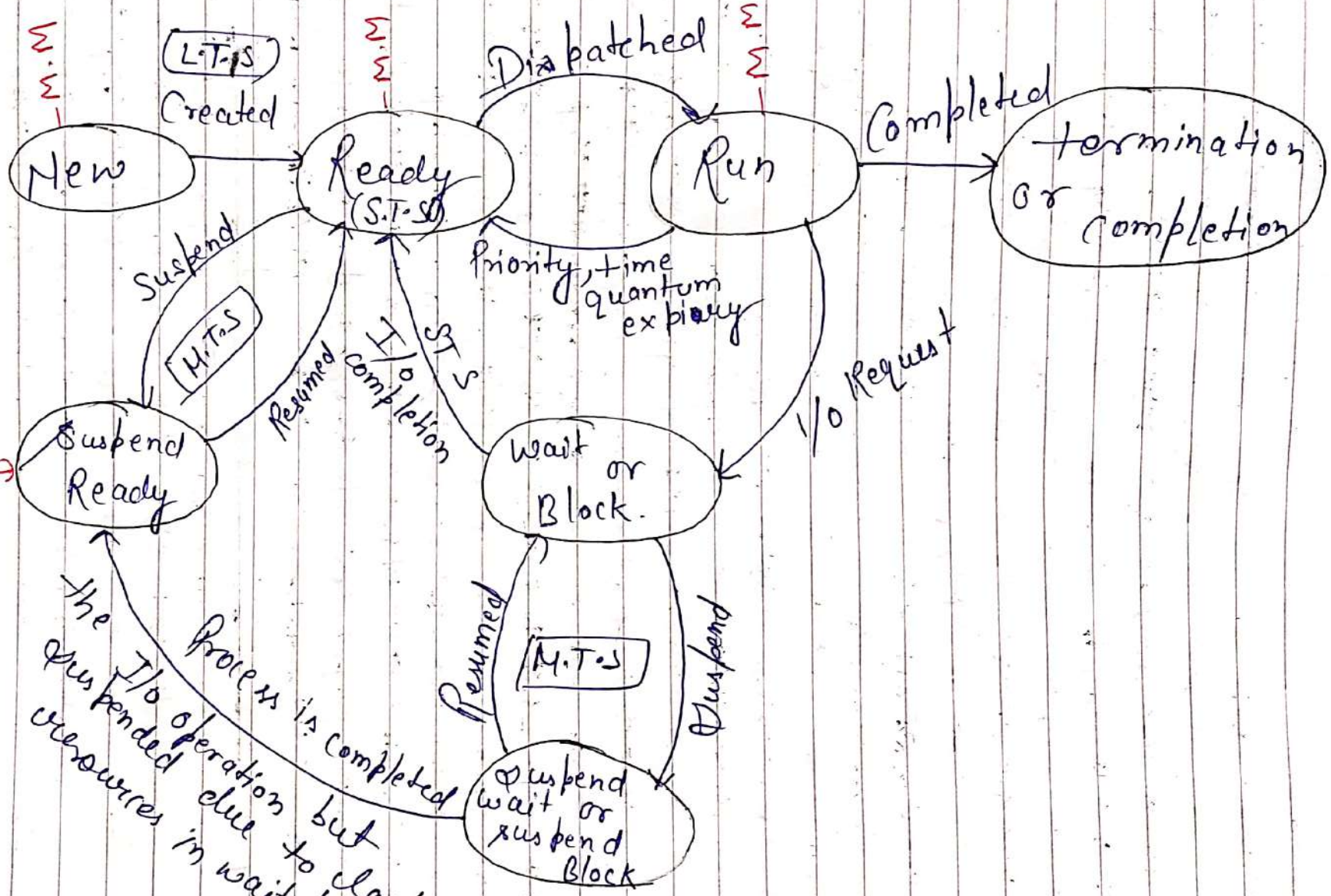
A Process will have various states :-

1.) New
2.) Ready
3.) running
4.) wait or block
5.) termination or completion.
6.) Suspend ready
7.) Suspend wait or suspend Block.

Various Operations performed on the processor :-

1.) Creating
2.) scheduling
3.) dispatching
4.) executing
5.) Killing or terminated
6.) Suspending.
7.) Resuming.

Process State Diagram :-



→ Initially process will be in the new state, it means process is under creation or process is being created.

→ Once the process is created, it will move on to ready state & in ready state there will be multiple no. of processes.

→ One of the process will be selected & that will be dispatched onto the running state.

→ When the process is the running state, it occupies the CPU & executing the instruction of the process & performing CPU time.

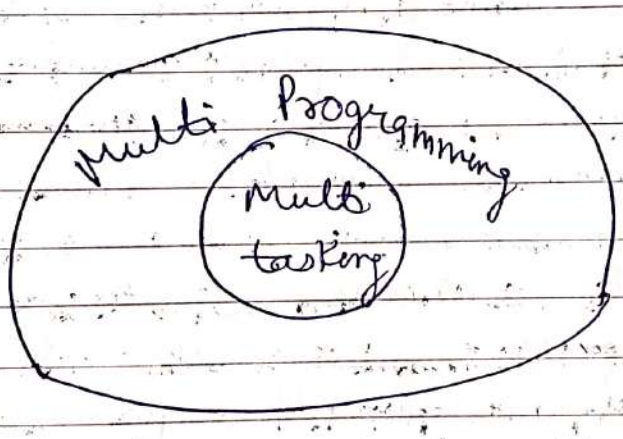
→ In the running state, there will be only one process at any point of time.

→ If the running process requires I/O operation, then it will be moved on to wait & block state & in waits & block state also their can be multiple process, it means multiple processess will perform I/O operation simultaneously.

Multi Programming :-

→ Non preemptive (Co-operative scheduling)

Preemptive (or) multitasking (or) time sharing (time quantum)

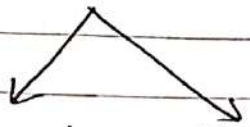


→ When the process is in the ready, running & the wait states then it is residing in the main memory.

→ If the more no of process are getting created, then the resources may not be sufficient to manage the processes in the ready state. In that case, some of the processes will be suspended & they will be moved onto suspend ready state.

→ When the process is in suspend ready state, then it is residing in the Banking stor (Secondary memory).

Processes



CPU bound I/O bound.

* CPU bound :- The processes which require more amounts of CPU time are called as CPU bound.

→ These process will spend more time in a running state.

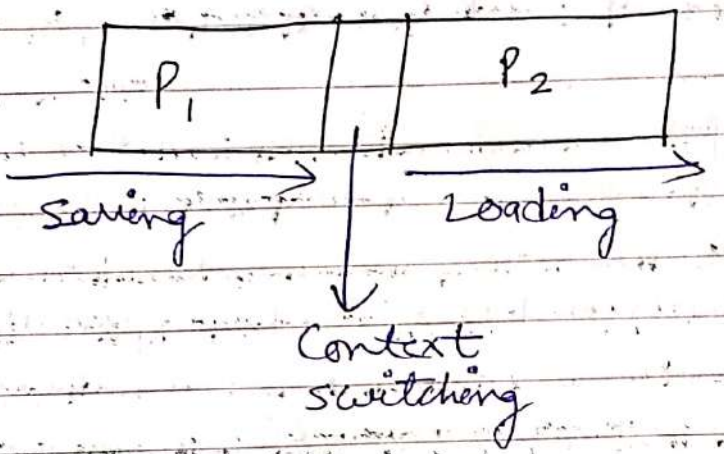
* I/O bound Process :- The processes which require more amount of I/O time are called as I/O bound press.

→ These process will spend more time in the wait or block state.

DEGREE OF MULTIPROGRAMMING :- The No. of processes present in the main memory at any point of time is called as degree of multiprogramming.

Each and Every time, when the process is moving from one state to other state the context of a process will change.

CONTEXT SWITCHING:-



⇒ saving the context of one process & loading the context of another process is called as context switching.

⇒ If the context of the process is more, context switching time will also increase, which is undesirable.

Context switching time is considered as overhead (burden) of the system.

NOTE:- In some special cases, if there is only one process, then still it is considered as a context switching.

E.g:- Round Robbin scheduling with 1 process.

3 - Schedulers :- [JOB SCHEDULER]

1.) LONG TERM SCHEDULER (L.T.S.) :- Long term scheduler is responsible for creating & bringing new processes into the system.

2.) SHORT TERM SCHEDULER (CPU SCHEDULER)
STS :- is responsible for selecting one of the process in the ready state for scheduling onto the CPU.

3.) MID TERM SCHEDULER (MTS) :-
(Medium term scheduler) is responsible for suspending & resuming the processes. (swapping)
→ The job done by MTS is called as a swapping :-

DISPATCHER :- is responsible for loading the selected job onto the CPU.

→ Dispatcher is also responsible of performing context switching.

→ Long term scheduler should select good combination of both the CPU bound & the I/O bound process in order to get a good throughput for the system.

→ LTS controls the degree of Multiprogramming.

Q1.) Consider a system which has n CPU processors, Then what is the minimum & maximum number of process that may present in the ready, running and wait state?

Ans.)

	Min	Max
Ready	0	Any
Run	0	n
wait	0	Any

Any depends on maximum degree of Multiprogramming in the system.

Process have various different Times :-

1.) Arrival Time :- The time when the process is arrived into ready state is called as arrival time of a process.

2.) Completion Time :- The time when the process is completing its total execution is called a completion time.

3.) Burst Time :- The time required by the process for its execution is called as a Burst time of a process.

4.) Turn about Time :- The time difference b/w completion time & arrival time is called as a Turn Around time of a Process.

$$TAT = CT - AT$$

$$TAT = WT + BT$$

5.) Waiting time :- The time difference b/w TAT & burst Time is called as a waiting time of a Process.

$$WT = TAT - B.T$$

6.) Response Time :- The time difference b/w First Response & Arrival time is called as a Response time of a Process.

CPU Scheduling.

CPU SCHEDULING:-

Who: Short term scheduler.
Where: Ready state.
When: 1. Rem → termination,
Run → wait,
Run → Ready,

2. Wait → Ready.
3. New → Ready.

GOAL OF THE CPU SCHEDULING:-

1.) To maximize the CPU utilization & throughput of the system.

2.) To minimize the Average TAT, Average WT & Average response time of the processes.

1.) FIRST COME FIRST SERVE (FCFS):-

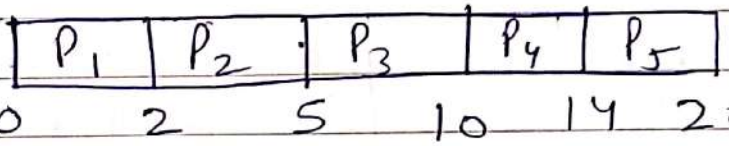
CRITERIA:- Arrival Time.
MODE:- Non Preemptive.

Margin of error – AT ko carefully Dekhe, if it is continuous or not. (CPU idleness).

1.)

P.No	AT	BT
1	0	2
2	1	3
3	2	5
4	3	4
5	4	6

P.No	AT	BT	CT	TAT	WT
1	0	2	2	2	0
2	1	3	5	4	1
3	2	5	10	8	3
4	3	4	14	11	7
5	4	6	20	16	10



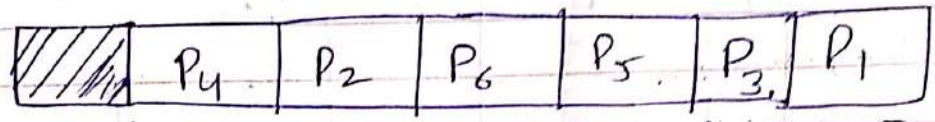
Average $TAT = \frac{41}{5} = 8.20$

Average $WT = \frac{21}{5} = 4.20$

NOTE :- first look at the A.T. carefully.

2.)

P. No	AT	BT	CT	TAT	WT
1	6	5	32	26	21
2	2	7	10	8	1
3	5	6	27	22	16
4	1	2	3	2	0
5	4	3	21	17	14
6	3	8	18	15	7

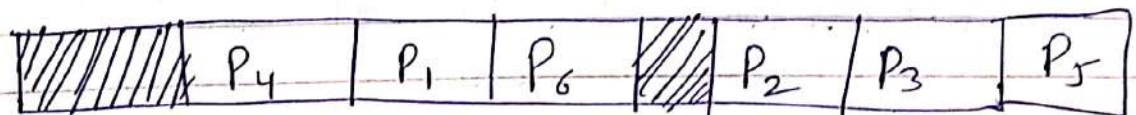


Average TAT = $\frac{90}{6} = 15.$

Average WT = $\frac{59}{6} = 9.83$

3.)

P. No	AT	BT	CT	TAT	WT
1	3	2	7	4	2
2	9	8	17	08	0
3	12	9	26	14	5
4	2	3	5	3	0
5	15	8	34	19	11
6	3	1	8	5	4



Average ~~Waiting~~ TAT = $\frac{53}{6} = 8.83$

Average WT = $\frac{22}{6} = 3.666$

Convoy effect:- In the FCFS, the first process is CPU bound process & followed by many I/O bound process, then it will have major affect on Average WT of the Processes. This effect is called Convoy effect.

4.)

P.No	AT	BT	CT	TAT	WT
1	0	20	20	20	0
2	1	1	21	20	19
3	2	1	22	20	19

Average TAT = 20

Average WT = $\frac{38}{3} = 12.66$.

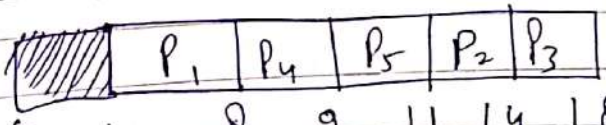
P.No	AT	BT	CT	TAT	WT
1	0	1	1	1	0
2	1	1	2	1	0
3	2	20	22	20	0

Average TAT = $\frac{22}{3} = 7.3$

2.) SHORTEST JOB FIRST (SJF):-

Criteria :- Burst time.
Mode :- non preemptive.

P No	AT	BT	CT	TAT	WT
1	1	7	8	7	0
2	2	3	14	12	9
3	3	4	18	15	11
4	4	1	9	5	4
5	5	2	11	6	4



Average TAT = $\frac{45}{5} = 9$

Average WT = $\frac{28}{5} = 5.60$

Pros: gives minimum average response time.
Cons:- long running jobs may starve if too many short jobs.

NOTE:- If the burst time of the process is matching then schedule the process, which has the lowest arrival time.

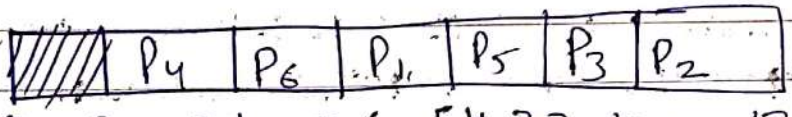
Q1.)

P.No	AT	BT	CT	TAT	WT
1	24	18	54	30	12
2	15	27	127	112	85
3	21	23	100	79	56
4	2	19	21	19	0
5	19	23	77	58	35
6	17	15	36	19	4

$= \frac{317}{6} = 52.83$

$\frac{192}{6} = 32$

317/6 = 54.66 = 38.83 $\frac{192}{6} = 32$



0 2 21 36 54 77 100 127

SHORTEST REMAINING JOB FIRST:-

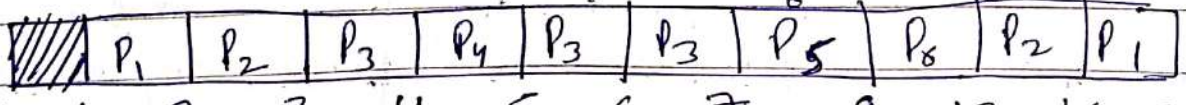
SHORTEST JOB FIRST (S.J.F.)

Criteria :- Burst time

mode :- Preemptive

PNo	AT	BT	CT	TAT	WT
1	1	7	22	21	14
2	2	5	16	14	9
3	3	3	7	4	1
4	4	1	5	1	0
5	5	2	9	4	2
6	6	3	12	6	3

$= \frac{50}{6} = 8.33$ $= \frac{29}{6} = 4.83$

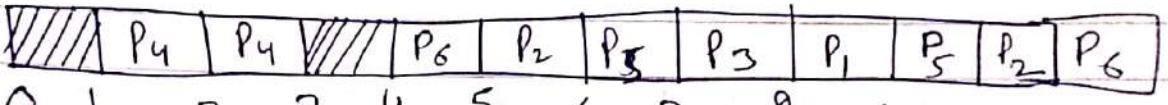


0 1 2 3 4 5 6 7 9 12 16 22

Q.)

PNo	AT	BT	CT	TAT	WT
1	8	2	16	2	0
2	5	7	19	14	7
3	7	1	8	1	0
4	1	2	3	2	0
5	6	4	13	7	3
6	4	9	27	23	14

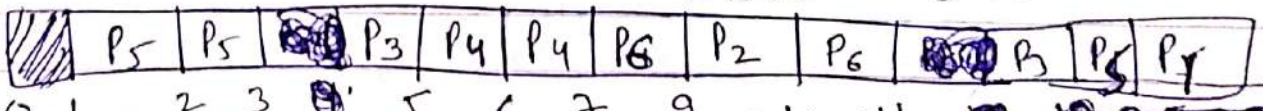
$\frac{49}{6} = 8.16$ $\frac{24}{6} = 4$



Q.)

PNo	AT	BT	CT	TAT	WT
1	11	8	23	12	4
2	9	1	10	1	0
3	3	6	15	12	6
4	5	2	7	2	0
5	1	10	31	30	20
6	7	3	11	04	1

61/6 = 10.16 31/6 = 5.16



Use Pencil for Doing calculation work.

NOTE:- Grant chart banane ke Baad, BT (updated) ko Pencil se Mita Do.

→ In case of SRTF, watch next AT carefully, (upto which you can run previous Process).

gate 2016

2 Marks

①

P.No	AT	BT	CT	TAT
1	0	10	20	20
2	3	6	10	7
3	7	1	8	1
4	8	3	13	5

33/4 = 8.25

Solve it by SRTF algorithm?

P_1	P_2	P_3	P_2	P_4	P_1

0 3 7 8 10 13 20

Q2.) Consider three processes A, B, C to be scheduled as per S.R.T.F. algorithm. The process A is known to be scheduled first, and when 'A' has been running for 7 units of time, the process C has arrived. The process C has run for 1 unit of time, the process B has arrived & completed running in 2 units of time. Then what could be the minimum burst times of the processes A & C?

a) 11 & 4
b) 11 & 3
c) 12 & 3
d) 12 & 4

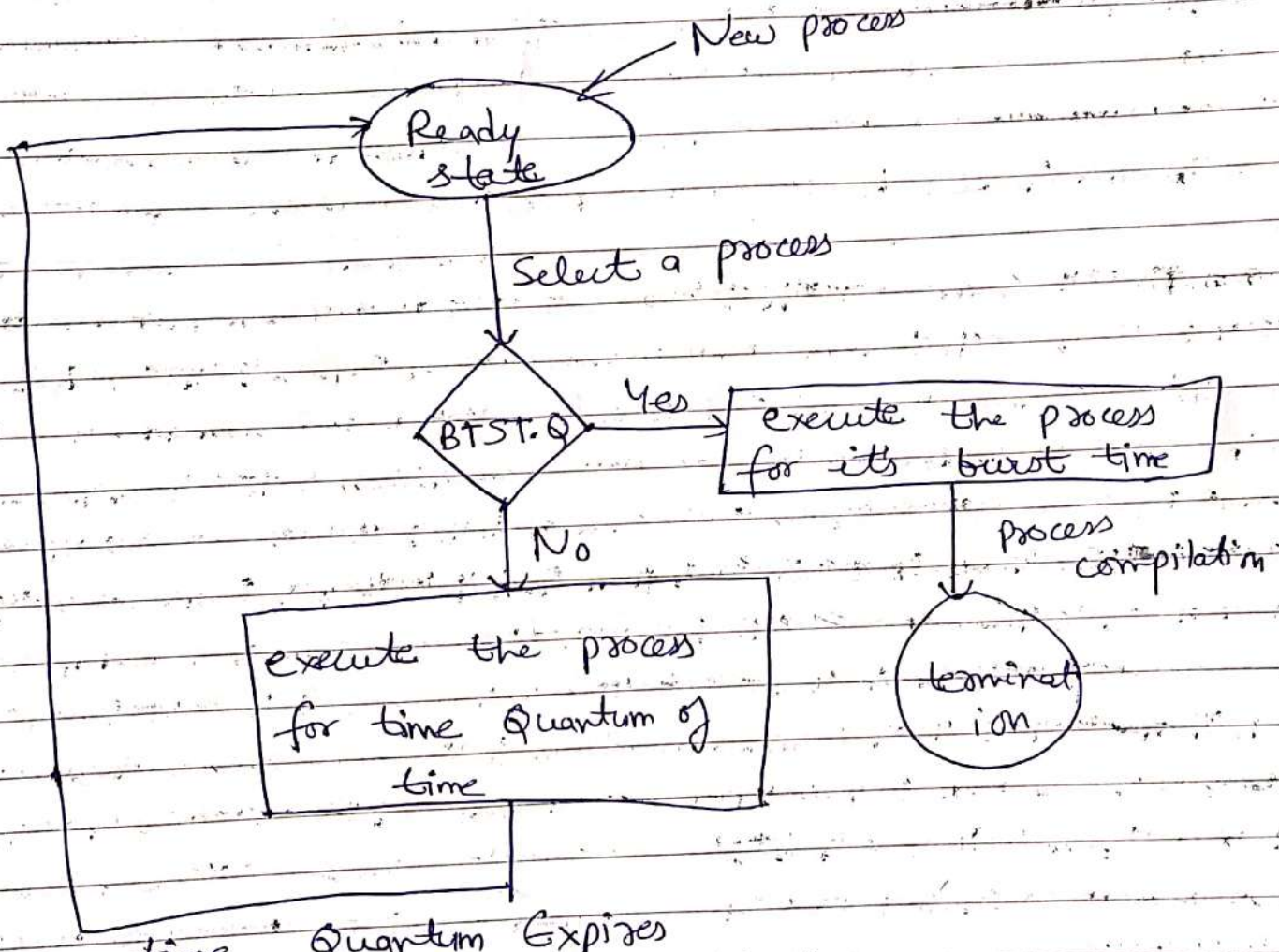
A remaining time is more than C.
& C remaining time is more than B.

30 A → 7 + 5 12
C → 1 + 3 = 4
B → 2 [illegible]

ROUND ROBIN SCHEDULING :-
(time slice)

Criteria :- Time quantum, Arrival Time

mode :- preemptive



C.S = Context switching

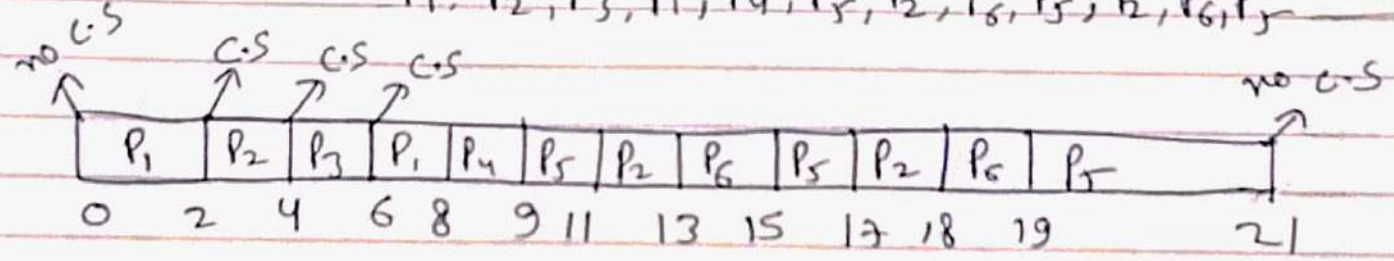
Q1.) T.Q = 2

P.No	A.T	B.T	C.T	T.A.T	WT
1	0	4	8	8	4
2	1	5	18	17	12
3	2	2	6	4	2
4	3	1	9	6	5
5	4	6	21	17	11
6	6	3	19	13	10

$\frac{65}{6} = 10.83$ $\frac{44}{6} = 7.33$

Ready queue :-

$P_1, P_2, P_3, P_1, P_4, P_5, P_2, P_6, P_5, P_2, P_6, P_5$



Average TAT = 10.83

Average WT = 7.33

first response - AT

same as first

2.)

R.T	PNo	AT	B.T
0	1	0	4
1	2	1	5
2	3	2	2
5	4	3	1
5	5	4	6
7	6	6	3

$\frac{20}{6} = 3.33$

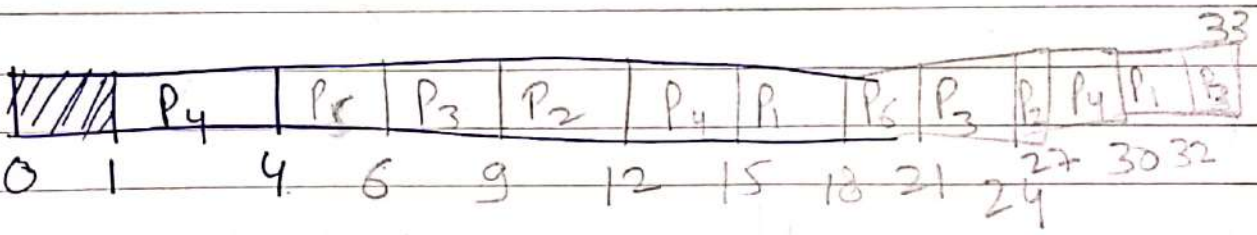
3.) T.Q = 3

P.No	AT	B.T	C.T	TAT	WT	RT
1	5	5	32	27	22	10
2	4	6	27	23	17	5
3	3	7	33	30	23	3
4	1	9	30	29	20	0
5	2	2	6	4	2	2
6	6	3	21	15	12	12

$\frac{128}{6} = 21.33$

Ready Queue :-

$P_4\ P_5\ P_3\ P_2\ P_4\ P_1\ P_6\ P_3\ P_2\ P_4\ P_1\ P_3$



NOTE:- for calculation of Response Time, check Gant chart from starting & putting the values in corresponding Process Id.

4.) T.Q = 2

P.No	AT	BT	CT	TAT	WT	RT
1	2	3	16	14	11	2
2	5	3	19	14	11	8
3	0	6	18	18	12	0
4	3	1	9	6	5	5
5	4	2	11	7	5	5
6	1	4	13	12	8	1
				71	52	21

R.Q = P_3 P_6 P_1 P_3 P_4 P_5 P_6 P_2 P_1 P_3 P_2

P_3	P_6	P_1	P_3	P_4	P_5	P_6	P_2	P_1	P_3	P_2
0	2	4	6	8	9	11	13	15	16	18	19

A

NOTE:- (i) If the round robbin, if the time quantum is less, no. of context switching will increase and ~~No. of~~ the response time will decrease.

(ii) If the time quantum is large, then no. of context switches will decrease, & response time will be more.

(iii) If the time quantum is greater than all the burst time, the round robbin similarly behaves like a FCFS algorithm.

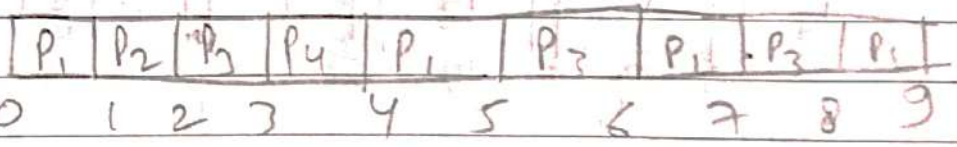
Q.) Consider a system, which has 4 processes, P_1, P_2, P_3, P_4 all arriving in the ready queue in the same order at time 0, The Burst time requirements of these processes are:-

B.T - 4, 1, 8, 1 respectively.
Then what is the completion time of Process P_1, assuming round scheduling with T.Q = 1?

P. No	AT	BT	CT	TAT	WT	RT
1	0	4	9			
2	0	1	2			
3	0	8	13			
4	0	1	4			

Ready Queue -

$P_1\ P_2\ P_3\ P_4\ P_1\ P_3\ P_1$



gate 1998

Q.) Consider n processes, sharing the CPU in round robbin algorithm. The context switching time is "S" units, then what must be the time quantum "q", such that the no. of context switches are reduced, but at the same time each process is guaranteed to get its turn/job at the CPU after every "t" seconds of time?

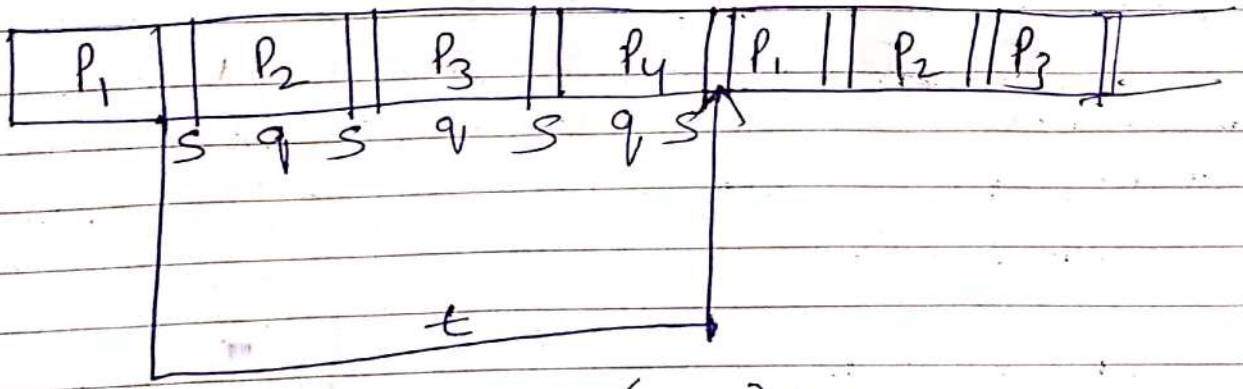
Assume AT = 0.

a.) $q = \frac{t-ns}{n+1}$

b.) $q = \frac{t+ns}{n-1}$

c.) $q = \frac{t-ns}{n-1}$

d.) $q = \frac{t+ns}{n+1}$



$t = n \times S + (n-1) * q$

$(n-1) * q = t - ns$

$$q = \frac{t-ns}{n-1}$$

$$q \leq \frac{t-ns}{n-1}$$

LONGEST JOB FIRST (L.J.F):-

Criteria :- Burst time.
Mode :- Non preemptive.

41/5 = 8.20

P.T	A.T	B.T	CT	TAT	WT
1	0	2	2	2	0
2	1	6	23	22	16
3	2	7	9	7	0
4	3	4	27	24	20
5	4	8	17	13	5
				68	41

68/5 = 13.60

P_1	P_3	P_5	P_2	P_4

0 2 9 17 23 27

Average TAT = 13.60

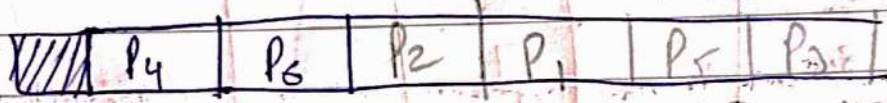
Average WT = 8.20

NOTE:- If the burst time of the processes are matching, schedule the process which have lowest arrival time.

Q1.)

P. No	AT	BT	CT	TAT	WT
1	24	19	89	65	46
2	17	23	70	53	30
3	21	16	121	100	84
4	2	18	20	18	0
5	19	16	105	86	70
6	13	27	47	34	7

2+

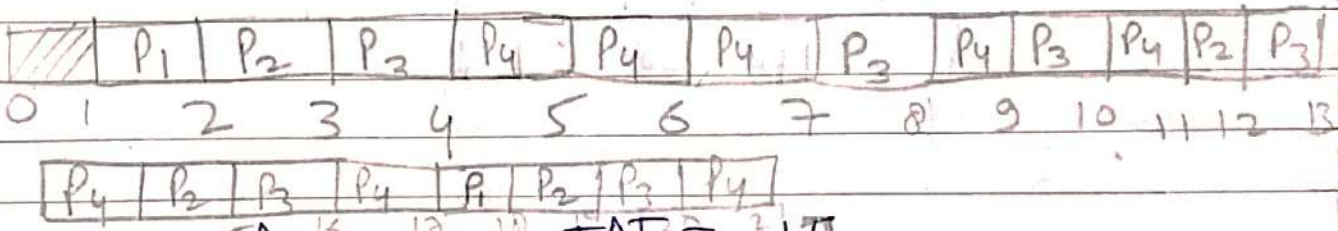


0 2 18 47 70 89 105 121

6.) LONGEST REMAINING TIME FIRST (L.R.T.F):-

CRITERIA:- Burst time.
MODE :- preemptive.

PNo	AT	BT	CT	TAT	WT
1	1	2	18		
2	2	4	19		
3	3	6	20		
4	4	8	21		



Average TAT = 15

Average WT = 10

GATE

	P.No	AT	BT	CT	TAT	WT
2 2	1	0	2	12	12	10
4 3	2	0	4	13	13	9
4 4	3	0	8	14	14	6
					39/3	25/3 = 8.33

What is average TAT using LRTF algo ?

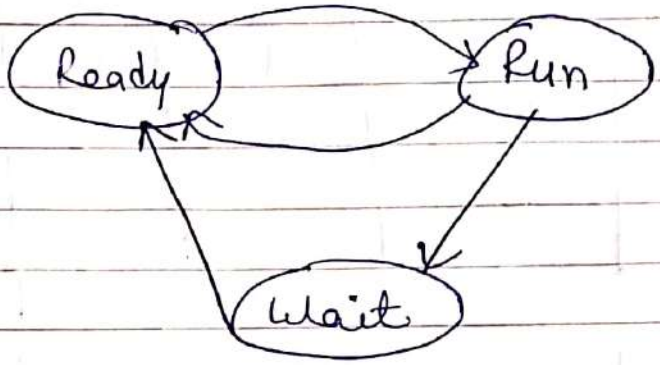
11.66

P3	P3	P3	P3	P2	P3	P2	P3	P1	P2	P3	P1	P2	P3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

A TAT = 13

A WT = 25/3 = 8.33

CPU TIME, I/O Time Concept:-



Life cycle:-

CPU time
I/O time
CPU time
I/O time
|
|
Complete

Cases:-

1.) CPU ✓
2.) I/O ✓
3.) CPU, I/O ✓
4.) CPU, I/O, I/O ✓
5.) I/O, I/O, I/O ✓
6.) CPU, CPU, I/O ✗

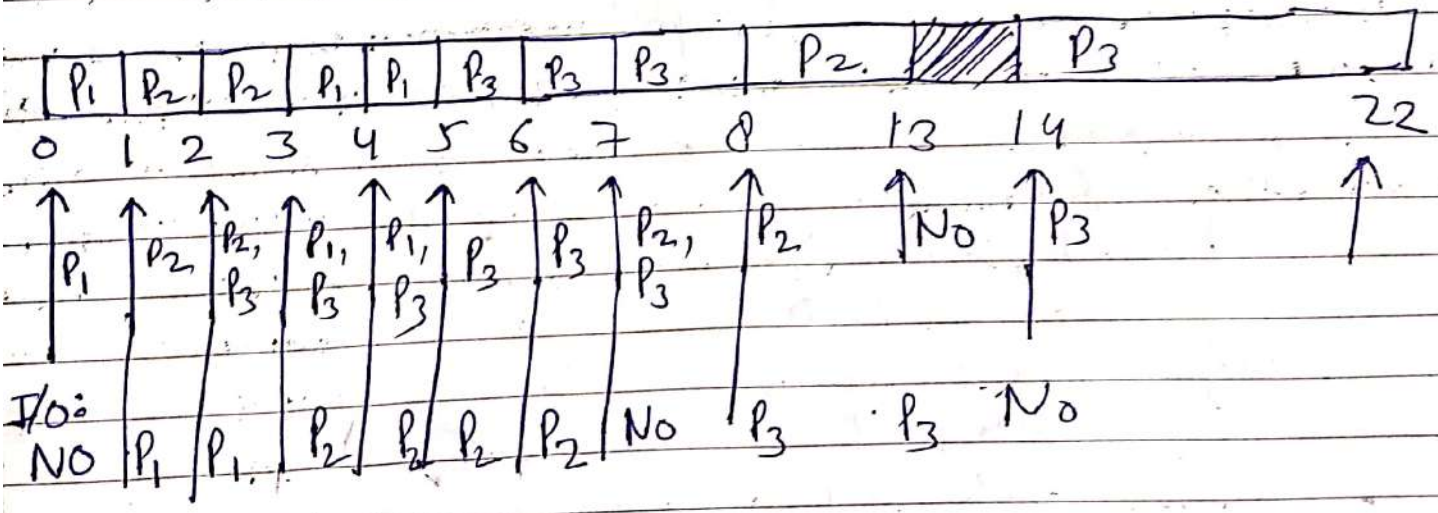
Q.)

A.T.	P. No	Execution Time			C.T
		CPU Time	I/O time	CPU time	
0	1	1 0	2	2	5
1	2	2 1 0	4	5	13
2	3	3 2 1 0	6	8	22

What is the completion Time of processes P_1, P_2, P_3 using S.R.T.F algorithm?

NOTE:- 1.) The process first spends CPU time followed by I/O time and followed by CPU time again.

2.) I/O of the processes can be overlapped as much as possible.



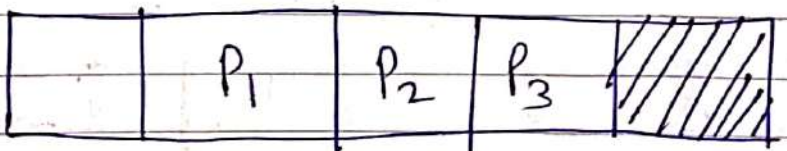
Q2.)

P.No	AT	Execution Time: I/O Time	Execution Time: CPU Time	Execution Time: I/O Time	Completion Time
1	0	40	140	240	20
2	0	840	28	4	50
3	0	1280	42	6	94

What is the completion time using SRTF ?

NOTE:- 1.) The process first spends I/O time followed by CPU time and followed by I/O time again.

2.) I/O of the processes can be overlapped as much as possible.



0 4 18 46 88 94

No P1 P2, P3 No

I/O: P1, P2, P3 — I/O: P2, P3 — I/O: P1 — P2 — P3

P-85
Q-28

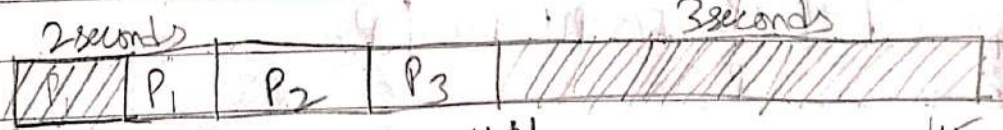
GATE 2006

Q1.) Consider three processes, All arriving at the time '0', with the total execution time of 10, 20 and 30 units respectively. Each process spends first 20% of execution time doing I/O, the next 70% of execution time doing computation, and last 10% of time doing I/O again. The O.S. uses S.R.T.F algorithm. All I/O operations are overlapped as much as possible. for what percentage of ~~the~~ does the CPU remain idle?

a.) 0%.
b.) 10.6%.
c.) 30.0%.
d.) 89.4%.

P.No	A.T	Execution Time			Completion time
		I/O	CPU	I/O	
1	0	2	7	1	10
2	0	4	14	2	20
3	0	6	21	3	30
			47		

S.R.T.F



0 2 9 23 45

0	2	9	23	45
No I/O	P_1	P_2, P_3	P_3	No I/O
P_1, P_2, P_3	I/O: P_1	I/O: P_2	I/O: P_3	
	P_2, P_3	P_1		

$= \frac{5}{47} \times 100$

$= 10.6\%$

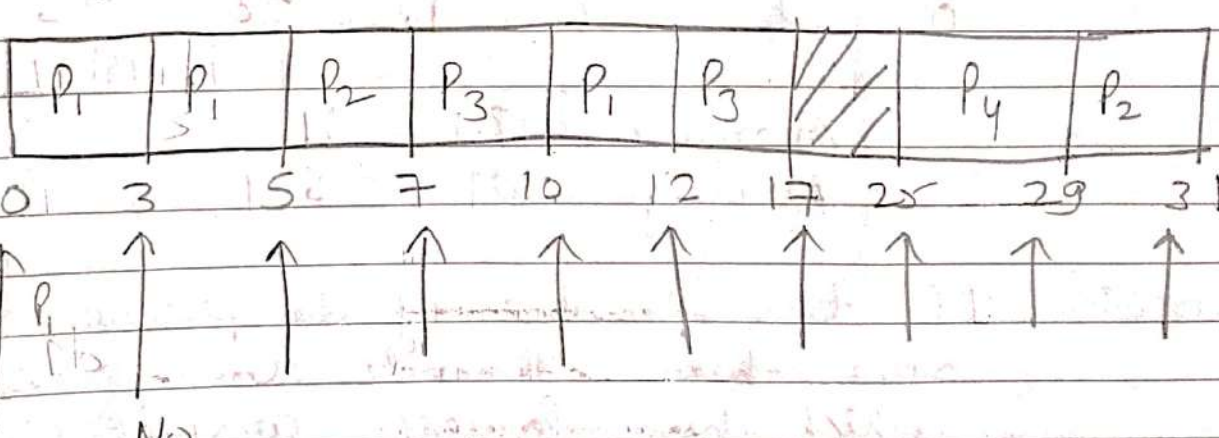
Q2.)

P.No	AT	CPU Time	I/O Time	CPU Time	C.T
1	0	5 2	5	2	12
2	3	2	22	2	31
3	7	8	0	0	17
4	25	9	2	1	39

What is the completion time of P_1, P_2, P_3, P_4 using S.R.T.F algorithm?

NOTE: 1) The process first spends CPU time followed by I/O Time followed by CPU time

2) I/O of the processes can be overlapped as much as possible



28/4/22

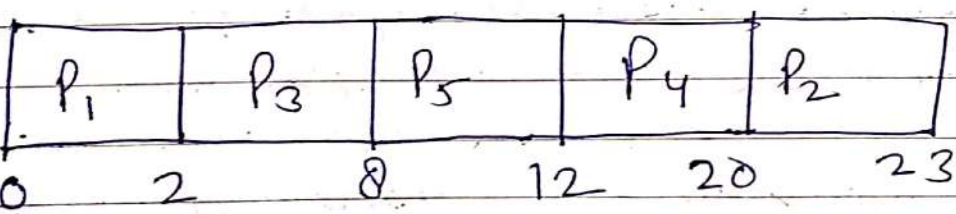
PRIORITY BASED SCHEDULING:-

CRITERIA:- PRIORITY
MODE:- NON PREEMPTIVE

PRIORITY	PNo	AT	B.T	TAT	WT	C.T
2.	1	0	2	2	0	2
4	2	1	3	22	19	23
6	3	2	6	6	0	8
5	4	3	8	17	9	20
high ← 7	5	4	4	8	4	12

55/5 32/5 = 6.4
=11

Assuming :- Highest Number have highest priority.



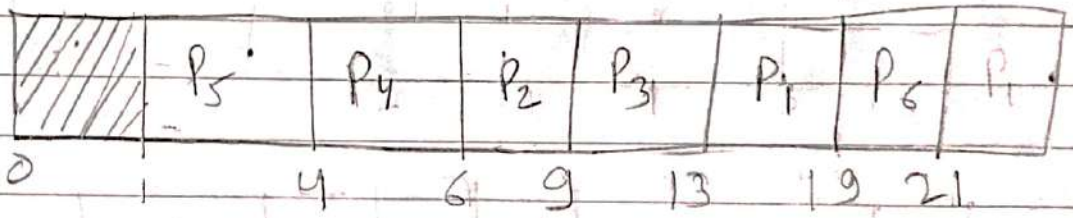
Average TAT = 11
Average WT = 6.4

NOTE:- If the priorities of the processes are same, then schedule the process which has lowest arrival time.

Q.)

Priority	P.No	AT	BT	CT	TAT	WT
4	1	4	6	19	15	9
7	2	6	3	9	3	0
6	3	3	4	13	10	6
6	4	2	2	6	4	2
1	5	1	3	4	3	0
3	6	2	2	21	19	17

$\frac{54}{6} = 9$ $\frac{34}{6}$ =



Average TAT = 9

Average WT = 5.66

Q1.)

Criteria :- priority.

mode :- preemptive.

priority	P. No	A.T	B.T	C.T	T.A.T
3	1	0	4	19	19
2	2	1	6	25	24
4	3	2	5	17	15
6	4	3	3	13	10
8	5	4	7	11	7

$= 15$

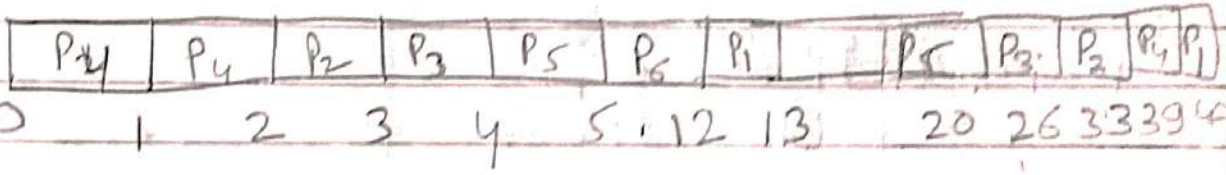
P_1	P_1	P_3	P_4	P_5	P_4	P_3	P_1	P_2

0 1 2 3 4 11 13 17 19 25

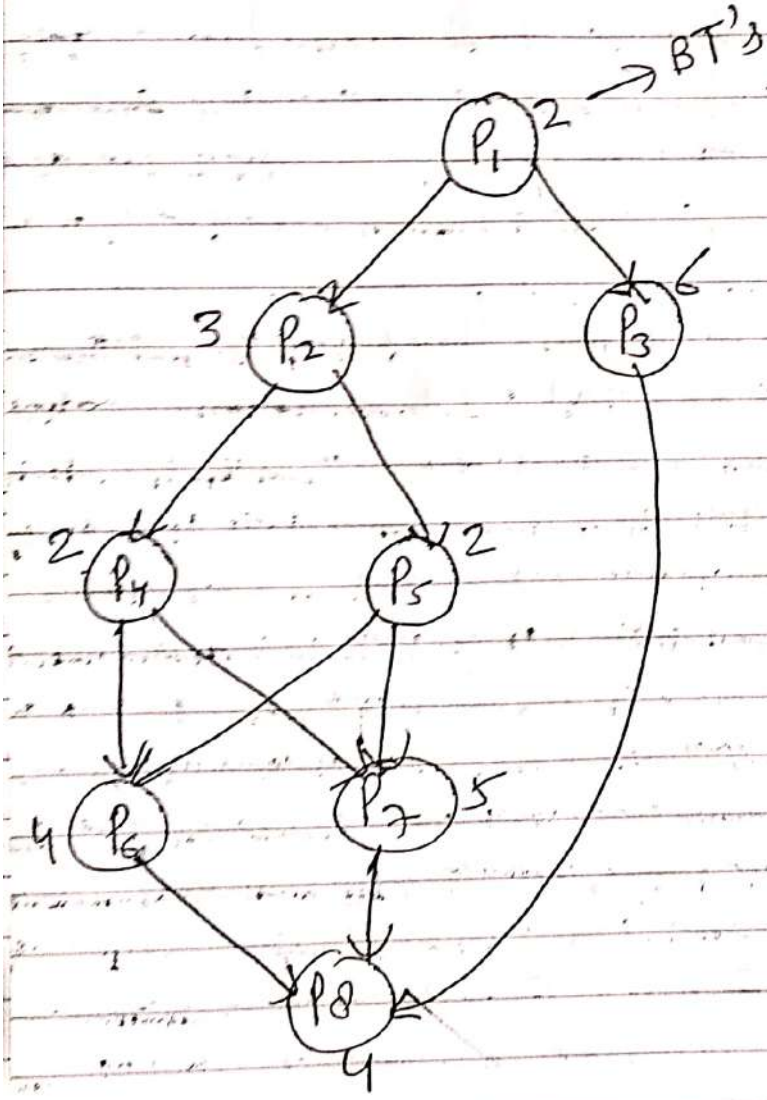
25.66 18.16

Q2.)

Priority	P.No	AT	BT	CT	TAT	WT
2 -	1	1	9			
3 -	2	2	8			
5	3	3	7			
2 -	4	0	6			
6	5	4	8			
High → 7	6	5	7			



a.) Consider the following dependency graph between the processes:-

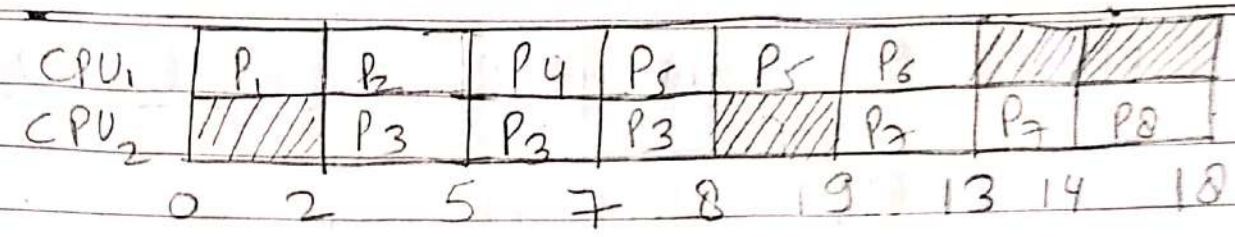


At what time, all the processes complete their execution by using 2 CPU processors?

a.) 17 b.) 18 c.) 19 d.) 20 e.) 21.

NOTE:- 1.) one process can not use '2 CPUs' at the same time.

2.) Use non preemptive mode


CPU1
CPU2
P1
P2
P4
P5
P5
P6
P3
P3
P3
P7
P7
P8
0
2
5
7
8
9
13
14
18

HIGHEST RESPONSE RATIO NEXT (HRRN):-

CRITERIA:- Response Ratio
MODE:- Non preemptive

$$\text{Response Ratio} = \frac{W + S}{S}$$

W = Waiting Time
S = Service Time or Burst time.

NOTE:- The above algorithms favours the shortest jobs and limits the waiting time of longer jobs.

Eg:-

P.No	AT	BT	C.T	TAT	WT
1	0	3	3	3	0
2	2	6	9	7	1
3	4	4	13	9	5
4	6	5	20	14	9
5	8	2	15	7	5

$\frac{40}{5} = 8$ $\frac{20}{5} = 4.$

P_1	P_2	P_3	P_5	P_4

0 3 9 13 15 20

at time 9:

$RP_3 = \frac{w+s}{s} = \frac{5+4}{4} = \frac{9}{4} = 2.25$

$RP_4 = \frac{w+s}{s} = \frac{3+5}{5} = \frac{8}{5} = 1.6$

$RP_5 = \frac{w+s}{s} = \frac{1+2}{2} = \frac{3}{2} = 1.5$

at time 13:

$RP_4 = \frac{7+5}{2} = \frac{12}{5} = 2.4$ — P_4

$RP_5 = \frac{15+2}{2} = \frac{7}{2} = 3.5$ — P_5

Average TAT $= \frac{40}{5} = 8$

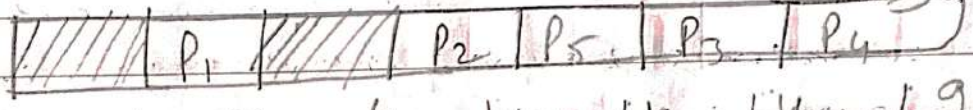
Average WT $= \frac{20}{5} = 4.$

Eg:

P.No	AT	BT	CT	TAT	WT
1	1	2	3	2	0
2	4	6	10	6	0
3	5	3	14	9	6
4	6	5	19	43	8
5	8	1	11	3	2

$\frac{16}{5} = 3.2$

$\frac{33}{5} = 6.60$



at time 10:

$RP_3 = \frac{5+3}{3} = \frac{8}{3} = 2$

$RP_4 = \frac{4+5}{5} = \frac{9}{5} = 2$

$RP_5 = \frac{2+1}{1} = \frac{3}{1} = 3$

at time 11:

$RP_3 = \frac{6+3}{3} = \frac{9}{3} = 3$

$RP_4 = \frac{5+5}{5} = \frac{10}{5} = 2$

Average TAT = 6.60

Average WT = 3.2

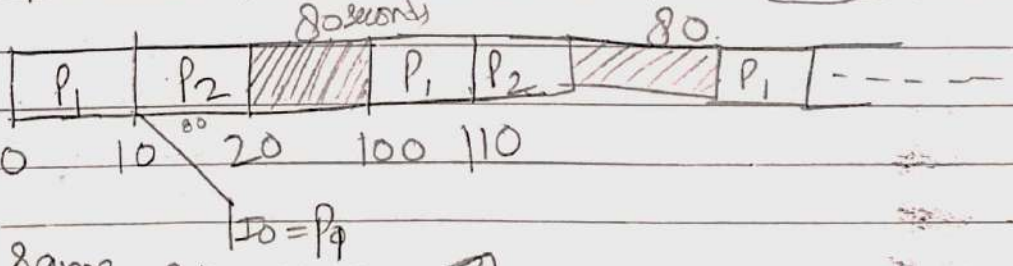
P-85, Q-29

GATE Consider a system has only two processes, both of which alternate 10ms of CPU burst with 90ms of I/O burst. Both the processes created nearly at the same time. The I/O of both the processes can proceed in parallel which of the following scheduling strategies will result in least CPU utilization over a long period of time?

a.) F.C.F.S.
b.) S.R.T.F.
c.) Priority
d.) Round Robin with time quantum of '5'ms.

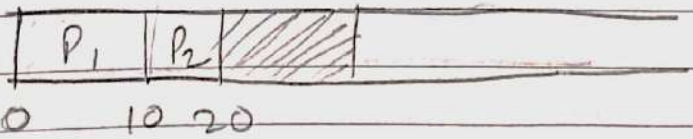
P.No	CPU	I/O	CPU	I/O	CPU
P_1	10	90	0	90	0
P_2	10	90	0	90	0

a.) F.C.F.S :-



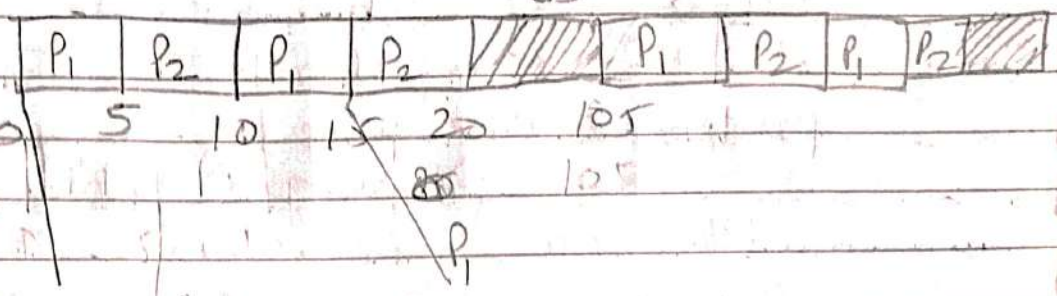
Same as FCFS

b.) S.R.T.F



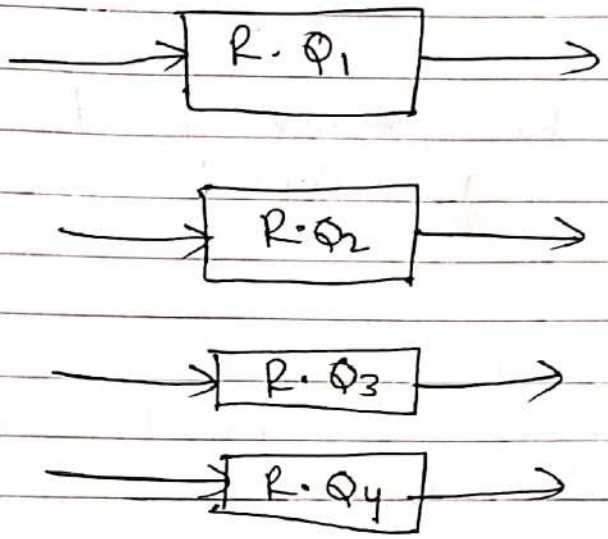
c) Priority :- Same as S.R.T.F.

d.) Round Robin with time quantum of 5 ms:



Answer is option d as the CPU idle time in Round Robin is 85 which in other scheduling algorithm have 80, so Round Robin will result in low CPU Utilization.

MULTILEVEL QUEUE SCHEDULING:-



→ Depending on the priority of a process, in which particular ready queue, the process has to be placed will be decided.

→ The high priority processes will be placed in top priority ready queue & the low priority processes will placed on a low priority ready queue.

→ only on the completion of processes from the top level ready queue, the further level ready queue process will be scheduled.

→ If this is strategy is followed then the processes which are placed in the bottom level ready queue will suffer from a starvation.

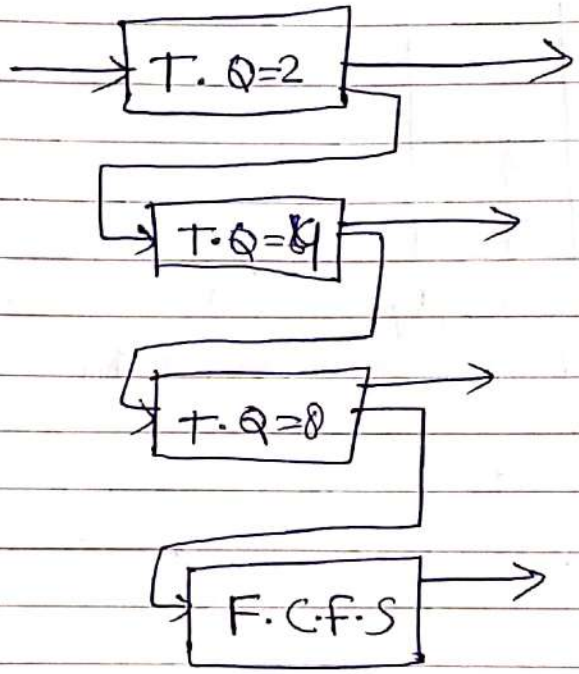
STARVATION: The Indefinite waiting of a process is called as a Starvation.

NOTE:- To avoid the problem of starvation, the concept of Ageing will be used.

Ageing :- If the waiting time of a process increases then the priority of a process will be increases

If the age of a process increases by increasing the priority, the process will definitly get a chance to execute and starvation problem will be avoided.

MULTI LEVEL FEED BACK QUEUE SCHEDULING



NOTE:- In the above algorithm, every process will definitely get a chance to execute but still there is a possibility for the process to go into starvation.

P.No	AT	B.T
1	0	2
2	1	3
3	2	5
4	3	6
5	4	8
6	5	7

what is the throughput of the system using FCFS algorithm?

P_1	P_2	P_3	P_4	P_5	P_6

0 2 5 10 16 24 31

$$\text{throughput} = \frac{\text{No. of processes}}{\max(C.T) - \min(A.T)}$$

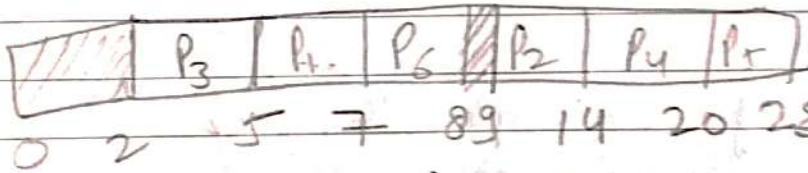
$$= \frac{6}{31-0} = \frac{6}{31} = \boxed{0.19}$$

$$= \boxed{19\%}$$

Schedule length = $\boxed{\max(C.T) - \min(A.T)}$

P.No	A.T	B.T
1	3	2
2	9	5
3	2	3
4	12	6
5	13	8
6	3	1

What is the throughput of the system using F.C.F.S algorithm?



$$= \frac{6}{28-3} = \frac{6}{25} = 0.23$$

$= 23\%$.

26

0.23
26) 60
52
80

*NP - Non preemptive

	Algorithm	Starvation
1.)	FCFS	No.
2.)	NP → SJF	Yes.
3.)	SRTF	Yes.
4.)	* RR *	No
5.)	NP → LJF	Yes.
6.)	LRTF	No.
7.)	NP → Priority	Yes
8.)	Pre- ~~LRTF~~ Priority	Yes
9.)	H.R.R.N	No.
10.)	M.L.Q	Yes
11.)	M.L.F.B.Q	Yes

NOTE:- Majority of the operating systems practicly implement Round Robin scheduling.

22/01/19

NOTE: In the shortest job first scheduling Algorithm, the burst times of the processes are expected by using the following formula

$$\tau_{n+1} = \alpha \cdot T_n + (1-\alpha) \cdot \tau_n$$

τ_{n+1} → next expected burst time
τ_n → previous expected burst time.
T_n → previous actual burst time

'α' is a parameter which controls, the relative weight of recent and past history.

$$0 \leq \alpha \leq 1$$

P-86

(32)

Process	Burst Time
P_1	5
P_2	8
P_3	3
P_4	5

(33)

$\alpha = 1$

$\tau_5 = \alpha \cdot T_4 + (1-\alpha)\tau_4$

$\tau_5 = 5 + 0$

$\tau_1 = 10$

$\alpha = 0.5$

$\tau_2 = 0.5 \times 5 + 0.5 \times 10$

$= 2.5 + 5$

$= 7.5$

$\tau_3 = 0.5 \times 8 + 0.5 \times 7.5$

$= 4 + 3.75 = 7.75$

$\tau_4 = 0.5 \times 3 + 0.5 \times 7.75$

$= 1.5 + 3.875 = 5.375$

$\tau_5 = 0.5 \times 5 + 0.5 \times 5.375$

$= 2.5 + 2.6875$

$= \boxed{5.1875}$

(9)

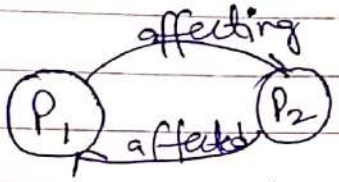
2.5
2.6875
5.1875

(C) 5.

M.O.S by D.S. Tenambanan

SYNCHRONIZATION:

Processes are categorized into two types



Processes

$P_1 \| P_2$

Cooperative independent

The execution of one process affects or affected by other process, then those processes are said to be cooperative Process, otherwise they are said to be independant Process.

Understanding × Synchronization :-

1.) The problems arises NOT having Synchronization between the processes.

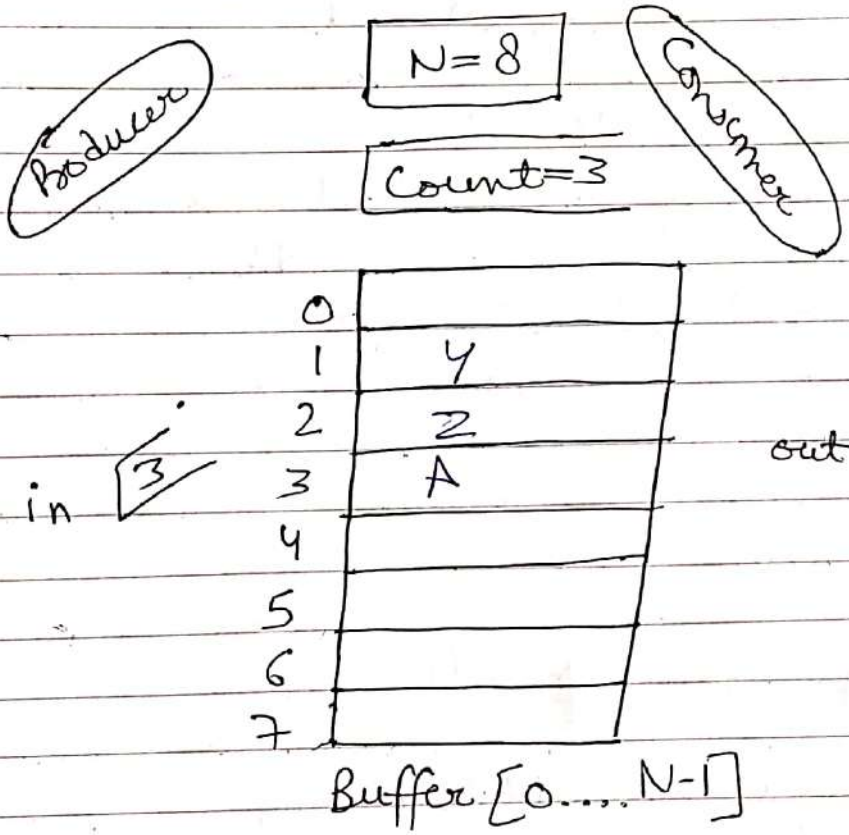
2.) the conditions to be followed by to achieve synchronization.

3.) The solutions (wrong and Right).

Synchronization

→ Problem x Arriving x During x Synchronization

The x PRODUCER- x CONSUMER:-



Analysis:-

Itemp = 'A'
ItemC = 'x'

P → I
P → II
C → I
C → II
C → III
P → III

Rp [3, 4]

Rc [3, 2]

1.) Inconsistency

```
int count = 0;
void producer(void)
{   int itemp;
    while(true)
    {
      produce_item(itemp);
      while(count == N);
      Buffer[in] = Itemp;
      in = (in+1) mod N;
      count = count+1;
    }
}
```

I. loads Rp, m[count]
II. INCR Rp
III. Store m[count], Rp

```
void consumer(void)
{
    int itemc;
    while(true)
    {
      while(count == 0);
      Itemc = Buffer[out];
      out = (out+1) mod N;
      count = count-1;
      process_item(itemc);
    }
}
```

I. load Rc, m[count]
II. DECR Rc
III. Stor m[count], Rc

→ In is a variable, used by the producer to identify the next empty slot in the buffer.

→ OUT is a variable, used by the consumer to identify, where it has to consume the item.

→ COUNT is a variable used by both producer and consumer to identify the number of items present in the buffer at any point of time.

Shared Resources:
1. Buffer
2. count variable.

Two conditions to be followed:-
1.) When the Buffer is full, the producer is not allowed to produce the item in buffer.

2.) When the Buffer is empty, the consumer is not allowed to consume the item from buffer.

UNIVERSAL ASSUMPTION :-

The running process can get preempted at any point of time after completion of current instruction.

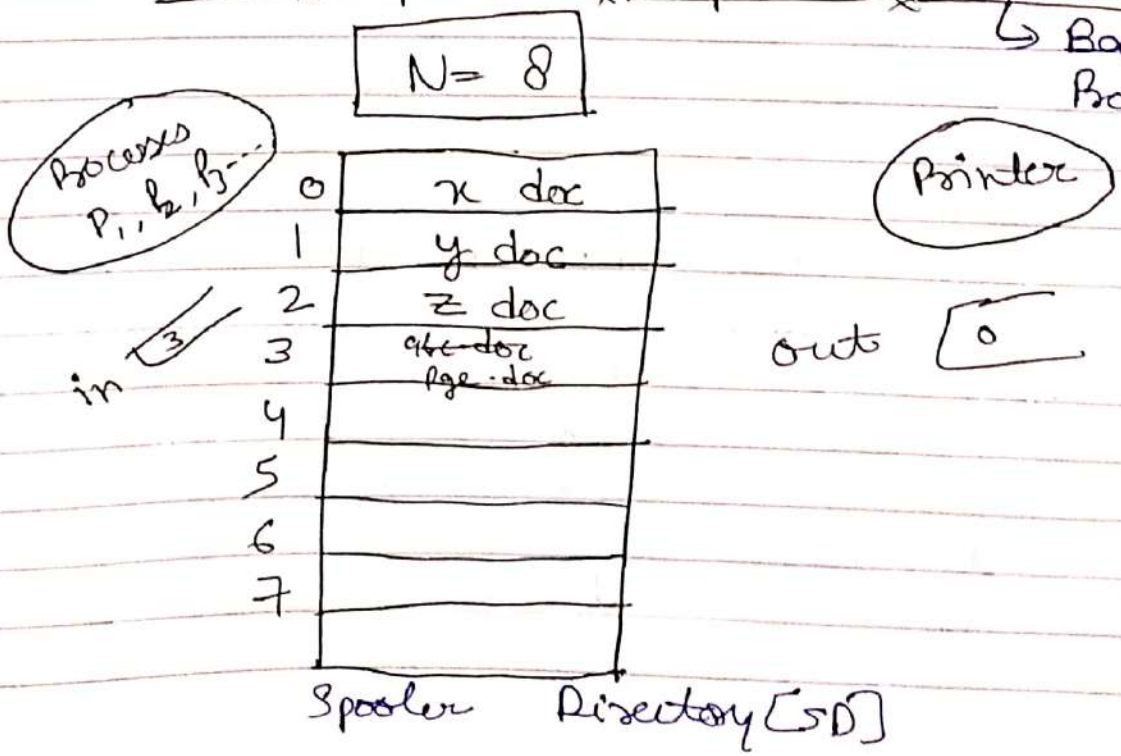
Analysis :-

Problem

1.) Inconsistency :- The producer and consumer are not properly synchronized while sharing the common variable, COUNT, hence it is leading to the problem of inconsistency.

The printer spooler Daemon :-

↳ Background Process



Enter - File

respective process Register

File name

I.) load Ri, m[in]
II.) Store SD[Ri], "F-N"
III.) INCR Ri
IV.) Store m[in], Ri.

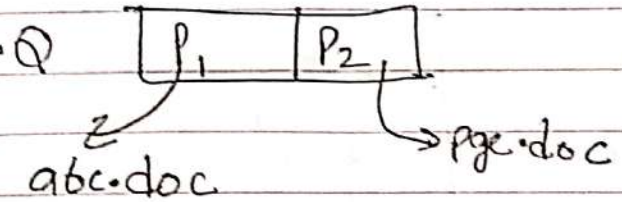
NOTE:- "In" is a variable used by all the processes to identify the next empty slot in the spooler directory.

"out" is a variable used by the printer to identify from where it has to print the document.

Shared resources :-

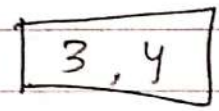
1.) "in" variable.
2.) Spooler directory.

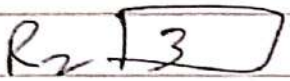
Analysis:



P → I
P → II
P → III

P2 → I
P2 → II





1.) Inconsistency :-

2.) Loss of data :- The processes are not properly synchronized while sharing the common variable "IN", hence it is leading to the Problem of loss of data.

3.) Deadlock : If the processes are not properly synchronized while sharing common variable or common resources then also possible for Deadlock.

Definition :-

1.) CRITICAL SECTION :- The portion of program text, where shared variables or shared resources will be placed.

Eg :- count = count+1;
count = count-1;

2.) NON CRITICAL SECTION :- The portion of program text, where the independent code of process will be placed.

Eg :- IN = (in+1) mod N;
out =

3.) RACE CONDITION :- The final value of any variable depends on execution sequence of the processes. These type of condition is called as race condition.

count = 4	count = 2
P → I	C → I
P → II	C → II
C → I	P → I
C → II	P → II
C → III	P → III
P → III	C → III

OR

NOTE:- To avoid the problem of race condition, only one process should be allowed to enter critical section at any point of time.

Conditions to be followed to achieve Synchronization:-

1.) Mutual Exclusion:-
- → No two process may be simultaeneously present inside the critical section at any point of time.
- → Only one process is allowed to enter into critical section at any point of time.

2.) Progress:- → No process running outside the critical section should lock the other interested process from entering into critical section when critical section is free.

→ If there is only one process trying to enter into Critical section, then it should be definitely allowed to enter into Critical section.

→ If two or more processes are trying to enter the critical section, then one process should be definitely allowed to enter into Critical section.

3.) Bounded waiting:

→ No process should have to wait forever to enter into critical section.

→ There should be a bound on getting chance to enter into critical section.

→ Some process is indefinitely waiting to enter into critical section, because critical section is always busy by some other processes, this situation should not arise.

→ If the bounded waiting is not satisfied, then it is possible for starvation.

4.) No assumptions related to hardware & the processor speed.

Solutions:-

I.) Software Type:-
- a.) Lock variables
- b.) strict alteration of Peters algo
- c.) petersons algorithms.

II.) Hardware Type:-
- a.) TSL instruction set
  - ↳ test and set lock

III.) O.S. Type:-
- a.) Counting semaphore.
- b.) Binary semaphore.

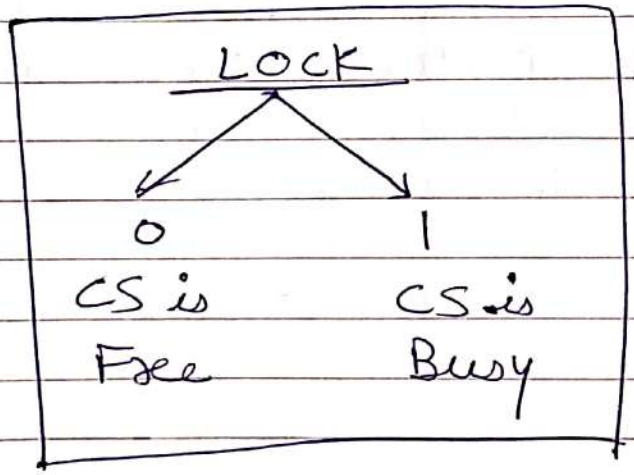
IV.) Programming language Compile support type:-
- a.) monitors.

Solution:

1.) LOCK VARIABLES:-

Entry Section:- → respective process register

I. Load R1, m[lock]
II. CMP R1, #0
III. JNZ to step (1)
IV. store m[lock], #1
V. C.S.
VI. Store M[lock], #0



Analysis:-

R.Q | P1 | P2

Lock = 0, 1, 1

C.S (P2) (P1)

P1 → I
P1 → II
P1 → III

R1 | 0

P2 → I
P2 → II
P2 → III

R2 | 0

$P_2 \rightarrow$ IV

$P_2 \rightarrow$ V

$P_1 \rightarrow$ IV

$P_1 \rightarrow$ V

Mutual Exclusion is not satisfied

NOTE:- We have proved that both the processes are enter into C.S. at the same time and Mutual exclusion is not satisfying, Hence solution is considered to be incorrect.

(5) STRICT ALTERATION OR DECKERS ALGORITHM:

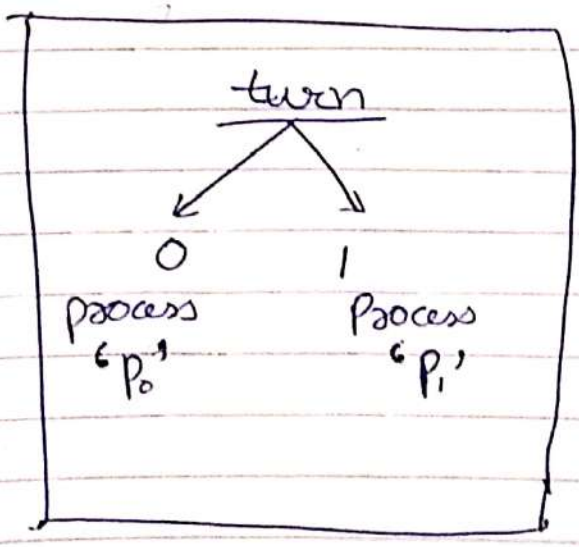
(process takes "turn" to enter into C.S.)

Process P_0 code

```
while (true)
{
  Non_CS ();
while (turn!=0);
  C.S.

  turn=1;
}
```

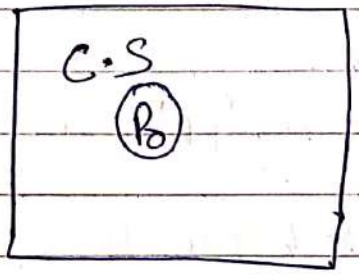
Process 'P1' code

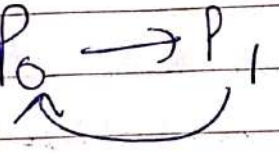
```
while (true)
{
  Non-CS();
while(turn!=1);
  C.S.

  turn=0;
}
```



Analysis :-

turn = 0





(1.) Mutual exclusion is satisfied.
(2.) Progress is not satisfying.

IMPORTANT POINTS:-

1.) The preemption is just a temporary stop. The process will come back and continue the remaining execution.

2.) If there is any possibility of solution becoming wrong by taking a preemption then consider the preemption.

3.) If any solution have deadlock, then the progress is not satisfied.

Q-15

doubt

Q.) Consider the two processes P_1 & P_2, accessing their critical section. The initial values of shared boolean variables $S1, S_2$ are randomly assigned.

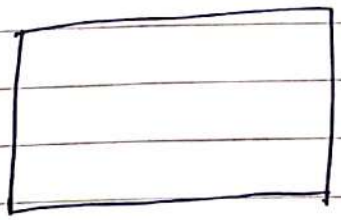
P_1	P_2
while ($S_1 == S_2$);	while ($S1 != S_2$);
C.S.	C.S.
$S_1 = S_2$;	$S_2 = not(S_1)$;

Which of the following is true?

a.) M.E is satisfied but NOT progress.
b.) progress is satisfied but NOT M.E.
c.) Neither M.E. NOR progress.
d.) Both are satisfied.

Analysis:

$S_1 = 1$
$S_2 = 0$



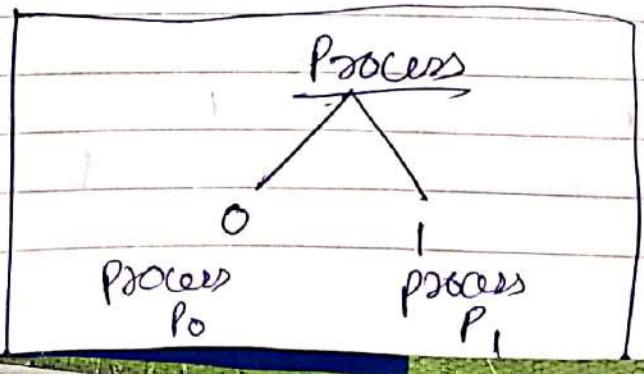
(a. Ans)

3) PETERSONS ALGORITHM :- (2 process solution)

```
#define   N 2
#define   TRUE 1
#define   FALSE 0
 int   turn;
 int   interested[N];

 void   enter_region(int process)
{  1  int other;
   2.  other = 1-process;                    → P0
   3.  interested[process] = TRUE;           → P1
   4.  turn = process;                       → P1 → P0
   5.  while(turn == process && interested
                [other] == TRUE);
}
```

C.S.

```
 void   leave_Region(int Process)
 {
       interested[process] = FALSE;
 }
 initially
 interested[0] = FALSE;
 interested[1] = FALSE;
```



(Round Robin)

"turn" is a shared variable used by both the process P0 & P1.

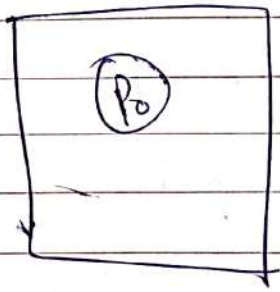
Analysis ①: Initially

interested[0] = FALSE; TRUE
interested[1] = FALSE;

turn = 0 or 1

Process 'P0'	Process 'P1'
other = 0 or 1	other = 0

C.S



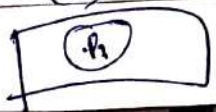
Analysis ②: initially false

interested[0] = FALSE;
interested[1] = FALSE; True

turn = 0 or 1

Process P0	Process P1
other = 1	other = 0

C.S



M.E. → Mutual Exclusion

1.) M.E. is satisfied.
2.) Progress is satisfied.
3.) Bounded waiting is satisfied.

→ It is a correct solution.

We have proved that all the three conditions are satisfied in the above solution, hence it is consider to be correct solution & it is properly providing synchronization to the processes, to access the common Critical section.

Doubt

Q1.) Assume that both the processes P_0 & P_1 are trying to enter into critical section at the same time, then which process will enter into Critical section first?

a.) The process which executes statement '2' first.

b.) The process which executes statement '3' first;

c.) The process which executes statement '4' first;

d.) We can not say.

(Solve:)

3. P_0 / P_1

4. P_1 / P_0

Turn = 0.

while (turn == process && i == ___);

0 1

F T

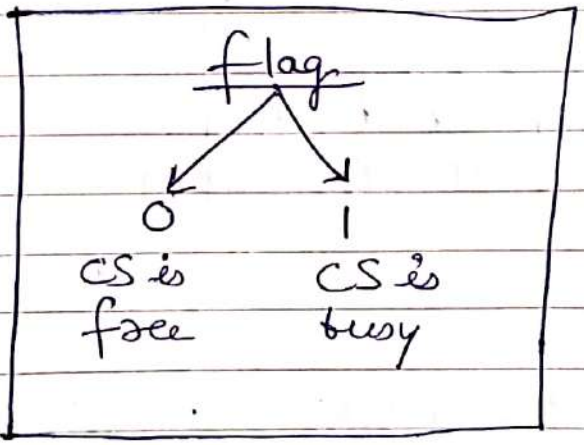
⇒ HARDWARE TYPE SOLUTION:-

1) TSL instruction set:-

TSL Register flag:- copies the current value of flag into register and stores the value of '1' into flag in a single atomic cycle without any preemption.

Entry section:-

I. TSL R_i, m[flag]
II. CMP R_i, #0
III. JNZ to step ①
IV. C.S.
V. Store M[flag], #0.



⇒ **Analysis:-** flag = ∅, ✗, 1

$P_1 \rightarrow$ I
$P_1 \rightarrow$ II
$P_1 \rightarrow$ III
$P_1 \rightarrow$ IV

R_1 [0]

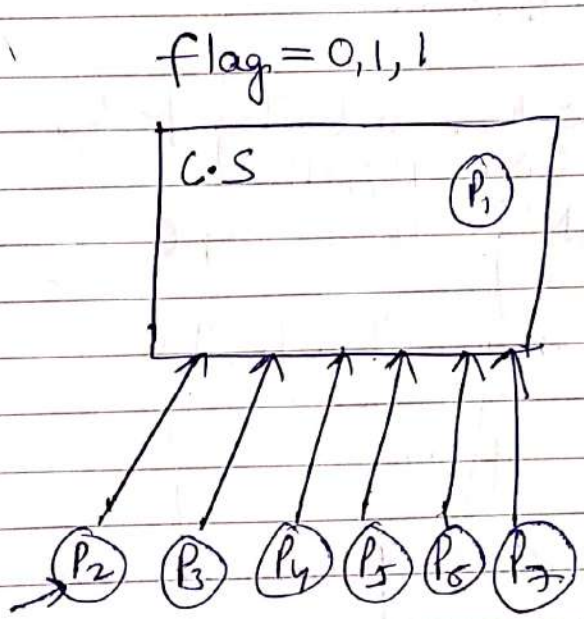
(P1) C.S.

R_2 [1]

$P_2 \rightarrow$ I
$P_2 \rightarrow$ II
$P_2 \rightarrow$ III

1.) Mutual Exclusion is satisfied.
2.) Progress is satisfied.
3.) Bounded waiting is not satisfied.

Analysis → (Bounded waiting's)



$P_1 \rightarrow$ I
$P_1 \rightarrow$ II
$P_1 \rightarrow$ III
$P_1 \rightarrow$ IV

$P_2 \rightarrow$ I
$P_2 \rightarrow$ II
$P_2 \rightarrow$ III.

R_1 [0]
R_2 [1]

Bounded waiting on Round Robin:- can be not satisfied.

Flag = 0, 1, 0

Ready Queue = P_1, P_2, P_3, P_1, P_4, P_5, P_2, P_6, P_7, P_3,

↓ ↑

can going to indefinite waiting (starvation)

NOTE:- (1.) If some process is in the C.S, then all the other processes, which are trying to enter into critical section will be repeatedly checking for I, II and III instructions.

(2.) These processes are busy in checking those instructions and waiting to enter into critical section, this is called as a "Busy waiting".

(3.) If the "Busy waiting", we are wasting the CPU cycles (time).

Busy waiting is also called as "spin lock".

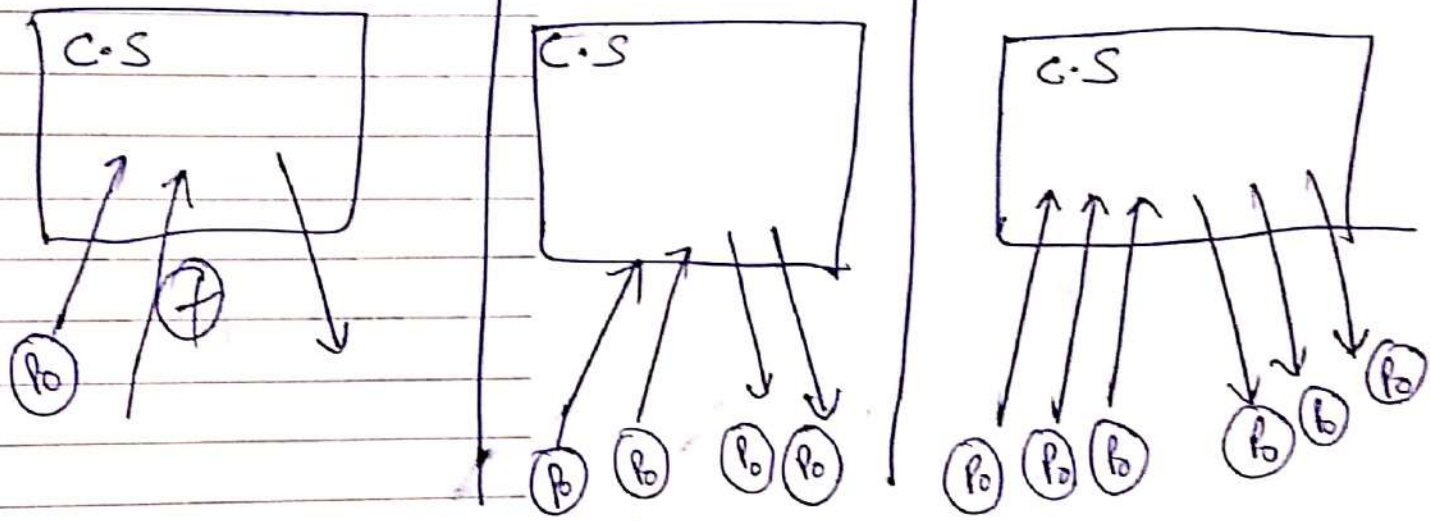
(3.) To avoid the problem of busy waiting, the concept of "semaphore" will be used.

Table

Solution	M.E	progress	bounded waiting
lock variable	✗	✓	✗
Strict alteration (or) Deckers algorithm	✓	✗	✓
Petersons algorithm	✓	✓	✓
TSL instruction set	✓	✓	✗

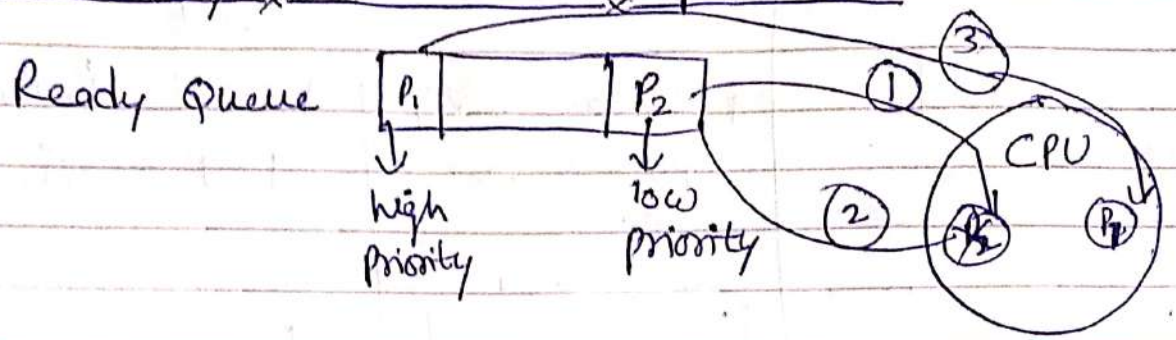
Progress

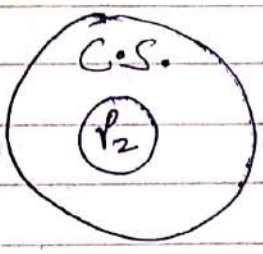
Strict alteration	Petersons algo	TSL instruction set
Process → P_0, P_1	process → P_0, P_1	Process → P_0, P_1, P_2, P_3 - -
progress ✗	progress ✓	progress ✓



first check deadlock then progress.

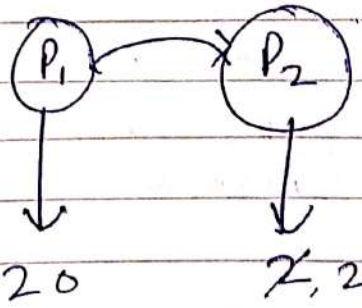
Priority Inversion problem:-





live lock
P_2 → Ready
P_1 → run

solution:- priority Inheritance

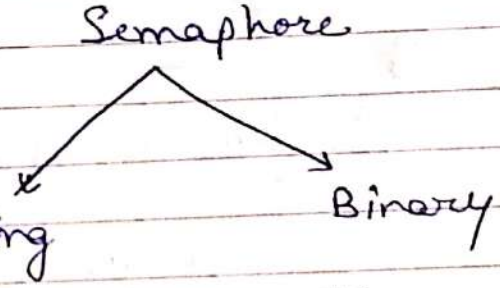


[changing the priority, from lowest to highest]

SEMAPHORE:- Semaphore is an integer variable used by the processes in a mutual exclusive manner to achieve synchronization.

→ The improper usage of semaphore will also give the wrong results.

→ Semaphore is categorized into two types
1.) ~~obligate~~ counting semaphore
2.) Binary semaphore



→ The two different operations performed on a semaphore variable.
1.) Down(); or wait(); or P();
2.) Up(); or signal(); or V() or Release();

COUNTING SEMAPHORE:-

```
Down (Semaphore S)
{
    S.value = S.value - 1;
    if (S.value <= 0)
    {
        Block the process and
        place its PCB in the
        suspended list();
    }
}

Up (semaphore S)
{
    S.value = S.value + 1;
    if (S.value <= 0)
    {
        Select a process from
        the suspended list and
        wakeup();
    }
}
```

NOTE:- → After performing down operation if the process is getting suspended, then it is called as a unsuccessful down operation.

→ If it is a unsuccessful down operation, then the process will not continue further execution in code.

$S \geq 1$

→ After performing down operation, if the process is not getting suspended, then it is called as a successful down operation.

→ If it is a successful down operation, then only the process will continue further execution in the code.

→ The initial positive value of the semaphore indicates No of successful down operations we can perform.

→ The down operation of counting semaphore will be successful only if when initial value of semaphore $\boxed{S \geq 1.}$

UP operation ⇒ The process performing up operation will definitely continue the further execution.

→ There is no unsuccessful UP operation and UP operation is always successful.

Q1.) Consider a system where initial value of the Counting semaphore $S = +17$, then various semaphore operations like 20P, 14V, 6P, 8V, 3P are performed. Then what is the final value of the semaphore.

Solve.) $S = +17 - 20 + 14 - 6 + 8 - 3$.

$S = 10$.

NOTE.) :- → The negative value of the counting semaphores indicates no of processes (blocked / suspended lists.)

Example:- $S = +1, 0, -1, -2$

P_1	P_2	P_3
P	P	P
✓	×	×

S.2 | P_2 P_3

Gate 2016) Numerical Type

(Q.) Consider the counting semaphore S. The various semaphore operations like 20 P, 12 V are performed. Then what is the largest initial value of semaphore is , so that 1 process will remain in a blocked lists.

Ans.)

$$S - 20 + 12 = -1$$

$$S = -1 + 20 - 12$$

$$\boxed{S = 7}$$

BINARY SEMAPHORE:-

```
Down ( Semaphore S)
{
    if (S.value == 1)
        S.value = 0;
    else
    {   Block the process and
        place its pcb in the
        suspended list();
    }
}

UP (Semaphore S)
{
    if (Suspended list is Empty)
        S.value = 1;
    else
    {   Select the process from
        suspended list and wakeup();
    }
}
```

NOTE :- 1.) After performing down operation if the process is getting suspended then it is called as a unsuccessful down operation. If it is a unsuccessful down operation, the process will not continue further execution of code.

→ After performing down operation, if the process is not suspended, then it is called as successful down operation.

→ If it is a successful operation then only process will continue further execution in the code.

Down operation → The down operation in the binary operation will be successfull, only when initial value of semaphor is '1'.

UP operation :- There is no unsuccessful up operation, up operation is always successfull.

→ The process performing up operation will definitely continue the execution.

$S = \phi, \phi, 1, 0, 1$

P_1	P_2	P_3	P_4	P_5	P_6	P_7
P	P	P	✓	✓	✓	✓
✓	×	×	✓	✓	✓	✓

S.L. (P2) (P3)

Q1.) Consider the binary semaphore variable S=1. the various semaphore operations like 9P, 14V, 6P, 8V, 3P, 2V are performed. Then what is the final value of the binary semaphore!

Ans.) S = 1, 0, 1, 0, 1, 0, 0
9P, 14V, 6P, 8V, 3P, 2V

S.L [8 0, 5 0 2 0]

final answer is 0.

NOTE:-

1.) Every semaphore variable will have its own suspended list.

No preemption before or after down()

2.) The down and Up operations are atomic.

3.) When more than one processes is in the suspended list, then every time when one process will wake up and that will be based on FIFO.

4.) If two or more processes are in the suspended list and there is no other process to wake up. these processes then those processes are said to be involved in the deadlock.

5.) When the process is in the suspended list, the process will reside in the ready state, but it will not participate in the scheduling.

p-down
v-up

(Q1.) Each process P_i, $i = 1$ to 9 executes the following code.

```
while(true)
{
  p(mutex);
  [ C.S. ]
  V(mutex);
}
```

The process 'P_{10}' executes the following code

```
while(true)
{
  V(mutex);
  [ C.S. ]
  V(mutex);
}
```

what is the max no. of processes, that may present inside the critical section at any point of time?

NOTE: The initial value of binary semaphore mutex = 1;

a.) 2.
(Options in case of P(mutex);) (ii) b.) 3.
c.) 9.
(i) d.) 10.

e.) I → (iii) ✓

a(iii) [P(mutex) V(mutex)] (e)

b(ii) [V(mutex); P(mutex);] (b)

Q.) Consider the two concurrent processes 'P' and 'Q' executing their respective codes.

process 'P' code	process 'Q' code

while (true)
{
W:
print ('0');
print ('0');
X:
}
	while (true)
	{
	Y:
	print ('1');
	print ('1');
	Z:
	}

(i.) What should be the binary semaphore operations on W, X, Y, Z respectively, and what must be the initial values of binary semaphore 'S' and 'T' in order to get print output always as 0011001100 ---- ---- ....

a.) W = p(S), X = V(S), Y = P(T), Z = V(T), S = T = 1;
b.) W = p(S), X = V(T), Y = P(T), Z = V(S), S = T = 1;
c.) W = p(T), X = V(S), Y = P(S), Z = V(T), S = 1, T = 0;
d.) W = p(S), X = V(T), Y = P(T), Z = V(S), S = 1, T = 0;
correct

010 or 0110
101 or 10001

Q.) Consider the → (same as above)

↑

(ii.) Which of the following will ensure that, the output string will never contain the substring of the form 01^n0 or 10^n1 where 'n' is odd?

a.) $x = V(S)$, $w = p(S)$, $y = P(T)$, $z = V(T)$, $S = T = 1$;
b.) $x = V(S)$, $w = p(T)$, $y = P(S)$, $z = V(T)$, $S = T = 1$;
c.) $x = V(S)$, $w = p(S)$, $y = P(S)$, $z = V(S)$, $S = 1$;
d.) $x = P(T)$, $w = V(S)$, $y = P(S)$, $z = V(T)$, $S = T = 1$;

2004

Q17.) (option-d)

A	B
L_1: P(S_x);	L_3: P(S_x)
L_2: P(S_y);	L_4: P(S_y)
C.S	C.S.
V(S_x);	V(S_y);
V(S_y);	V(S_x);
}	}

Analysis:-

$S_x = 1, 0, 1$

$S_y = 1, 0, 1$

$S_L = S_x$ (B)

Workbook 16

Q16.) Ans.)

Case 1:-

$S_0 = 1.0\,1\,0\,1\,0$

$S_1 = 0.1\,0\,1\,1$

$S_2 = 0.1\,1\,0\,1$

000

$P_0 \rightarrow P_1 \rightarrow P_0 \rightarrow P_2$

$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_0$

Case 2:-

$S_0 = 1\,0\,1\,1$

$S_1 = 0\,1\,0$

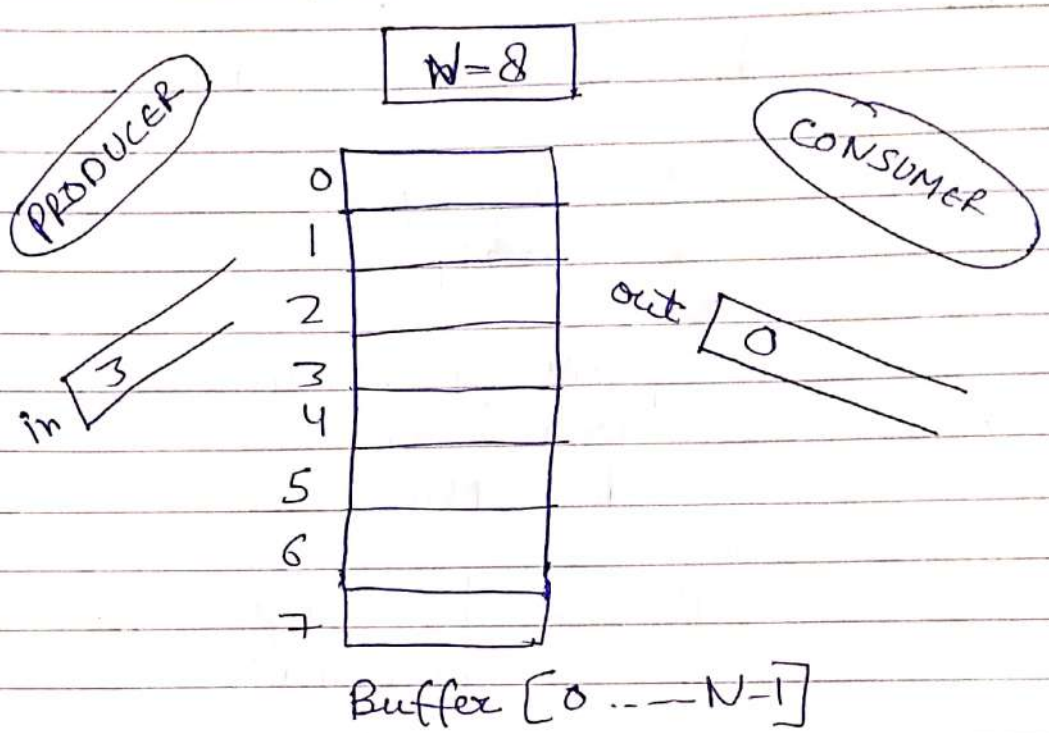
$S_2 = 0\,0\,1\,0$

00

(9. Ans.) At least twice

CLASSICAL PROBLEMS OF IPC:-

PRODUCERS- CONSUMERS with Semaphore:-



```
Semaphore mutex = 1;
Semaphore Empty = N;
Semaphore Full = 0;

void producer (void)
{
    int itemp;
    while (true)
    {
        produce-item (itemp);
        down (Empty);
        down (Mutex);

        Buffer[in] = itemp;
        in = (in+1) mod N;

        Up (Mutex);
        Up (Full);
    }
}

void consumer (void)
{
    int itemC;
    while (true)
    {
        down (Full);
        down (mutex);

        Itemc = Buffer [out];
        out = (out+1) mod N;
```

```
        Up(Mutex);
        Up(Empty);
        process_item (itemc);
        }
    }
```

→ Mutex is binary Semaphore used by producer and Consumer in a mutual exclusive manner to access the buffer.

→ Empty is a counting semaphore variable represents no. of slots empty in the buffer at any point of time.

→ fully is a counting semaphore variable represents no of slots full in the buffer at any point of time.

Analysis:

Mutex = 1 0 1 0 1
Empty = 5 4 5 4
Full = 3 4 3
8 8 8

Itemp = A, B.
Itemc = X, Y.

Q.) what happens if we interchange

```
down(Empty);
down(mutex);
```

in the producer code.

a.) No problem, the solution still works correct.

b.) both producer & consumer will access the buffer at the same time.

c.) Some of the items produced by the producers will be lost.

~~d.)~~ It is possible for deadlock.

Analysis:-

Mutex = ~~1~~, 0
Empty = ~~0~~, -1
Full = ~~0~~, 7

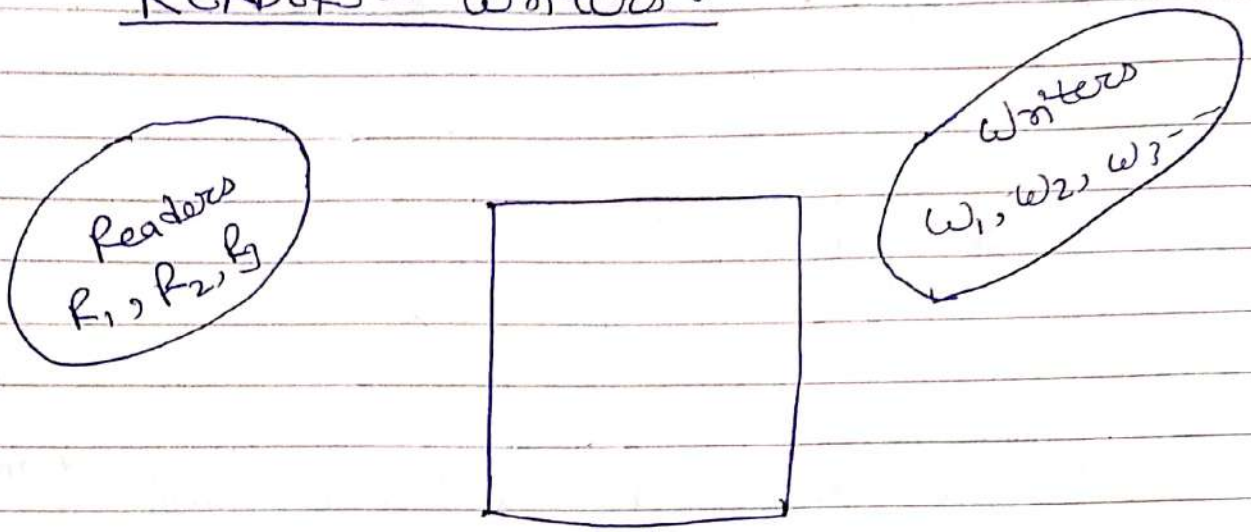
Take producer as full.

Q2.) What happens if we interchange

```
up(mutex);
up(full);
```

a.) No problem.
b.) both consumer & producer will access the buffer at same time.
c.) some of the items by the producers will be lost.
d.) It is possible for deadlock.

READERS - writers :-




```
int     rc = 0;
semaphore   mutex = 1;
semaphore   db = 1;
void    Reader (void)
{
    while (true)
    {
        down (mutex);
        rc = rc + 1;
        if (rc == 1) { down (db); }
        up (mutex);

        [ D.B. ]

        down (mutex);
        rc = rc - 1;
        if (rc == 0) up (db);
        up (mutex);
    }
}
```

```
void writer (void)
{
    while(true)
    {
        down(db);
        D.B.
        up(db);
    }
}
```

4 conditions to be followed:-

1.) R → W ✗
2.) R → R ✓
3.) W → R ✗
4.) W → W ✗

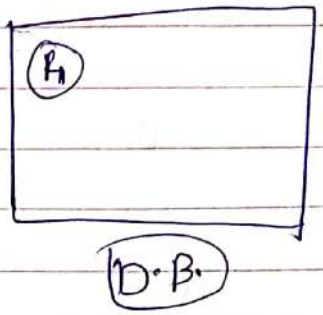
→ RC is an integer variable represents reader's count i.e. The No. of readers present in the database at any point of time.

→ Mutex is a binary semaphore used by the reader's in a mutual exclusive manner.

→ db is also a binary semaphore used by the reader's & writers in a mutual exclusive manner.

Analysis 1:-

RC = 0, 1
Mutex = 1 0 1
db = 1 0



[for first 2 conditions]

Analysis 2:-

RC = 0
Mutex = 1
db = 1 0

1Q.) What happens if we interchange

down (mutex);
~~down~~ rc = rc + 1;

in a reader's code?

a.) No problem, the solution still works correct.
b.) Multiple readers are not allowed into database.
c.) both reader and writer will enter into database at the same time.
d.) It is possible for deadlock.

2Q.) What happens if we interchange

preemption.
if (rc == 1) down(db);
up (mutex);

(Preemption before if)

in the reader's code.

a.) No problem, the solution will work correct.
b.) Multiple readers are not allowed into database.
c.) both reader and writer will enter into database at the same time.
d.) It is possible for deadlock.

3. Q) what happens if we interchange

```
down(mutex);
rc = rc - 1;
```

in the reader's code.

a.) No, The problem will still work properly.
b.) Multipk readers are not allowed into database.
c.) both reader and writex will enter into database at the same time.
d.) It is possible for deadlock.

Solve:)

2/02/19

P-95

19.) mutex=
R=0; 1
W=0;

Analysis :-

R = ~~0~~ 1
W = 0
mutex = ~~1~~ 0 1

D.B.

up(mutex); — 2

~~(b)~~ R ≥ 1 or W == 1 — 3

~~(a)~~
~~(c)~~ → R=0, W=0, mutex = 1
~~(d)~~ (d) Right answer

20.) (D) Ans.)

Q.) Consider the following code used by classical readers & writers.
which of the following is true?

a.) The solution is correct & it is properly providing the synchronization to classical readers & writers.

b.) Multiple writers are allowed to enter into database at the same time.

c.) Both Reader & writers are allowed to enter into database at the same time.

d.) It is possible for deadlock.

```
int rc=0;
int wc=0;
Semaphore mutex=1;
Semaphore r=1;
Semaphore db=1;

void Reader(void)
{
  while (TRUE)
  {
    down(r);
    down(mutex);
    up(r);
    rc = rc+1;
    if (rc==1) down(db);
    if (wc==1)
    {   down(r);
        up(db);
    }
```

```
up(mutex);
```

D.B.

```
down(mutex);
rc = rc - 1;
if (rc == 0) up(db);
} } up(mutex);

void writer(void)
{   while(TRUE)
    {
        down(r);
        wc = wc + 1;
        down(db);
```

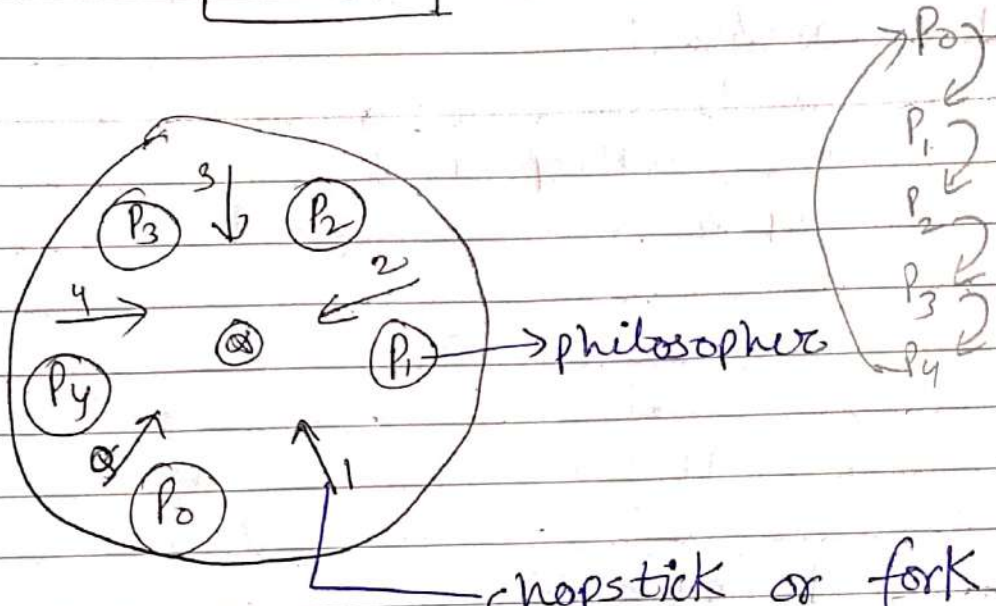
D.B.

```
        up(db);
        wc = wc - 1;
        up(r);
    }
}
```

(d Ans) deadlock.

Dining philosophers :-

N=5



int i → philosopher No.

```
void philosopher (~~void~~ int i)
{
    while(true)
    {
        thinking();
        take_fork(i); //take left fork
        take_fork((i+1) % N); //take right fork
        eat();
        put_fork(i); //put left fork back
        put_fork((i+1) % N); //put right fork
                              back
    }
}
```

If all the philosophers are hungry at the same time, then everybody will take their left fork first and when they tried to attempt to right fork, then nobody will get the right fork and all the philosophers will wait on each other for the Right fork and they will going to deadlock.

Solution with the help of Semaphore.

```
#define N 5
#define THINKING 0
#define HUNGRY 1
#define EATING 2
#define LEFT (i+N-1)%N      → left philosopher
#define RIGHT (i+1)%N       → Right philosopher
Semaphore mutex=1;
Semaphore S[N]; //all S[i]'s are initialized
                  to '0';

int state[N];
    //an array to keep track of
      every philosophers state.

void philosopher(int i)  → philosopher No
{
    while (true)
    {
        thinking();
        take_forks(i);
        eat();
        put_forks(i);
```

```
    }
}
take_forks( int i)
{
    down (mutex);
    state [i] = HUNGRY;
    test (i);
    up (mutex);
    down (s [i]);
}

put_folks (int i)
{
    down (mutex);
    state [i] = THINKING;
    test (LEFT);
    test (RIGHT);
    up (mutex);
}

void test (int i)
{
    if (state [i] == HUNGRY && state [LEFT]
        != EATING &&
        state [RIGHT] != EATING)
        {
            state [i] = EATING;
            Up(s[i]);
        }
}
```

→ Mutex is a binary semaphore, used by the philosopher in a mutual exclusive manner.

→ S[N] is an array of binary semaphore, initially, all are assigned to 0.

→ State[N] is an integer array used to keep track of every philosopher state. Initially all the philosophers will be in the thinking state.

- Analysis :-

Mutex = 1 ⊘ 1

P_0 = T	S[0] = 0
P_1 = T	S[1] = 0
⇒ P_2 = T ✗ E	S[2] = ⊘ ✗ 0
P_3 = T	S[3] = 0
P_4 = T	S[4] = 0

Q.) Assume that philosopher P_1 & P_3 are in eating state. Then philosopher P_2 is also trying to going to eating state. Then which statement in the above code is controlling the philosopher?

a.) down(mutex);
b.) If condition
c.) test function
d.) down(S[i]);

Q2.) What happens if we interchange

up(mutex);
down(s[i]); in the takefork()?

a.) No problem the solution still works correct.
b.) More than two philosophers can go into eating state at the same time.
c.) It is possible for deadlock.
d.) None of the above.

Now

Q.) Assume that P_1 and P_3 are in the eating state, then Philosopher P_2 is also trying to go to eating state, Then it will be suspended. Then what is the procedure and how philosopher P_2 will go into eating state?

GATE

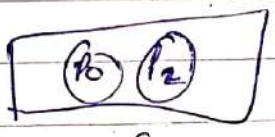
Q.) Let p[0] ..... p[4] be the processes and m[0] ..... m[4] be the binary semaphore mutexes all are initialized to 1. Each process 'P_i' executes the following code.

```
wait( m[i]);
wait( m[(i+1)%4]);

C.S.

Signal (m[i]);
Signal (m[(i+1)%4]);
```

Preempted

→ Consider the following statements
I. M.E. is Satisfied.
II. M.E is NOT Satisfied.
III. It is possible for deadlock.

Which of the above are TRUE?
a.) only I
b.) only II
c.) only I, III
d.) only II, III.



Analysis:-

$P_0 = m_0, m_1$
$P_1 = m_1, m_2$
$P_2 = m_2, m_3$
$P_3 = m_3, m_0$
$P_4 = m_4, m_1$

$m_0 = 1$
$m_1 = 1$
$m_2 = 1$
$m_3 = 1$
$m_4 = 1$

Q.) Consider the following code used by the processes to access the common critical section.

```
Shared boolean LOCK = FALSE;
    boolean Key;
    do
    {
        Key = TRUE;
        while (Key == TRUE)
        Swap (Key, lock);

        C.S.
        lock = FALSE;
    } while (TRUE);
```

NOTE: Swap() function atomically swaps two variables by using call by reference.

⇓

⇒ Consider the following statements

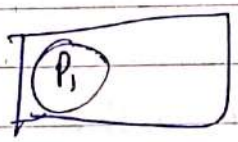
I. M.E is satisfied
II. M.E is not satisfied
III. It is possible for deadlock.

Which of the above are TRUE ?

a.) Only I ✓
b.) Only II
c.) Only I, III
d.) Only II, III

LOCK = ~~FALSE~~ TRUE

P_1	P_2	P_3
Key = ~~True~~ False	Key = True	



C.S.

MONITORS:-

→ Monitors is programming language, compiler support type of solution to achieve synchronization.

→ Monitors is collection of variables, condition variables and procedures combined together in a special kind of module or package.

→ The processes running outside the monitor cannot directly access, the internal variables of the monitor but however they can call procedures of the monitor.

→ Monitors has an important property that only one process can be active inside the monitor at any point of time.

Syntax:- monitor (Keyword) example (Name of the monitor)

```
{
    variables;
    condition variables;
    procedure P1
    {
        ≡
    }
    procedure P2
    {
        ≡
    }
}
```

CONDITION VARIABLES :-

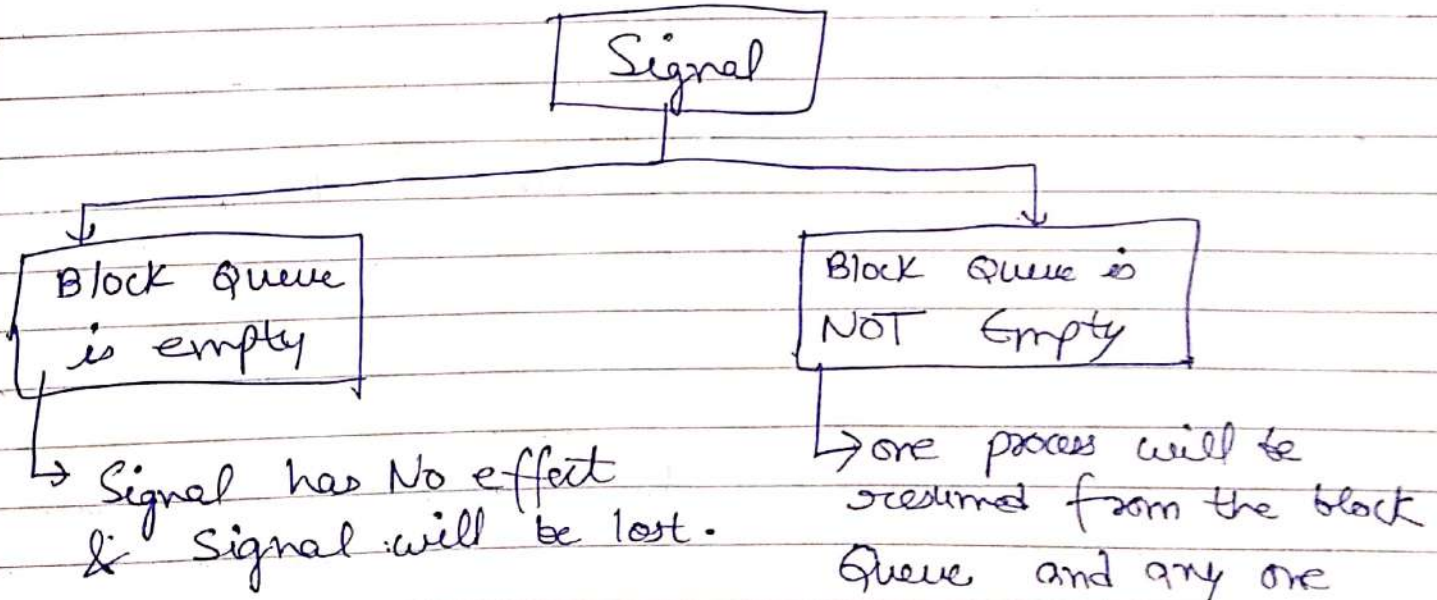
```
condition x, y;
```

The two different operations performed on the condition variables of the monitor.

1.) wait();
2.) signal();

1.) wait() :- The process performing wait operation on any condition variable will be suspended and suspended process will be placed in the "block Queue" of respective condition variable.

2.) Signal() :- The process performing signal operation will have two cases



PRODUCERS - CONSUMERS WITH MONITORS :-

N = 8

Count = 3

Buffer [0..... N-1]

```
monitor  producer_consumer
{
    int  count = 0;
    condition  Empty, Full;
    procedure  Enter_item
    {
        if (count == N) wait (Full);

        enter (item);

        count = count + 1;
        if (count == 1) Signal (Empty);
    }
```

```
procedure  Remove_item
{
    if (count == 0) wait (Empty);
    remove (item);
    count = count - 1;
    if (count == N-1) Signal (Full);
  }
}
```

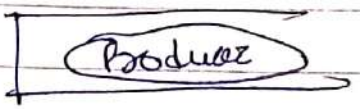
```
procedure  producer
{
    int  itemp;
    while (true)
    {
        produce_item (itemp);
        producer_consumer.enter_item;
    }
}
```

```
procedure  Consumer
{   int  itemc;
    while(true)
    {   producer_consumer.remove_item;
        process_item (itemc);
    }
}
```

Analysis:-

Buffer is full:-

Block Queue of full



When Buffer ~~is~~ Empty:-

Block Queue of Empty

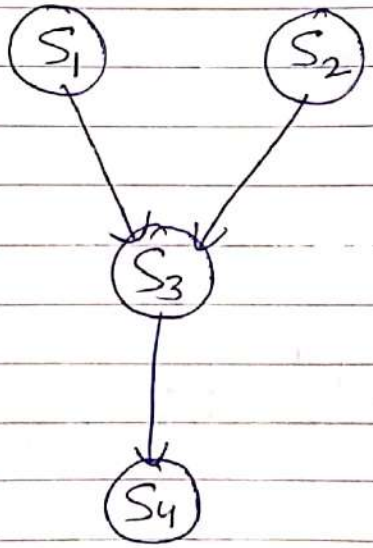
NOTE:- Even though we are using the count variable, there will ~~be~~ not be any inconsistency because inside the monitor only one process can be active at any point of time.

CONCURRENT PROGRAMMING:-

S_1 : a = b + c;
S_2 : d = e * f;
S_3 : g = a / d;
S_4 : h = g * i;

Read Set = { b, c, e, f, a, d, g, i };
Write Set = { a, d, g, h };

Precedance graph:-



NOTE:- Any two statements 'S_i' and 'S_j' can be executed concurrently or parallely if they are following the below conditions.

1.) $R(S_i) \cap W(S_j) = \phi$.
2.) $W(S_i) \cap R(S_j) = \phi$.
3.) $W(S_i) \cap W(S_j) = \phi$.

2.) → The real concurrent programming is possible only on the multiprocessor System.

3) Concurrent has the three different meanings

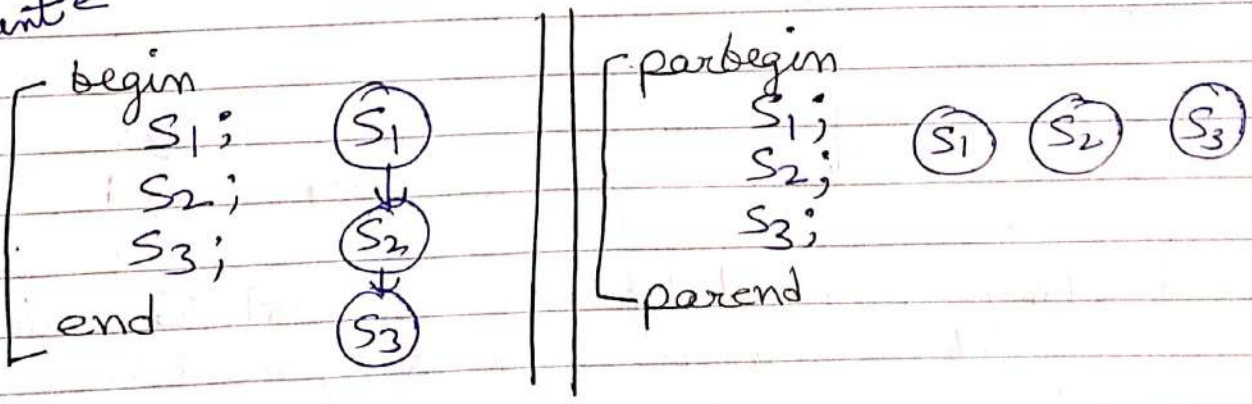
- They can execute concurrently or parallely.
- They dont have any dependency.
- Any one can start first.

4.) The concurrent program will be written by using

parallel

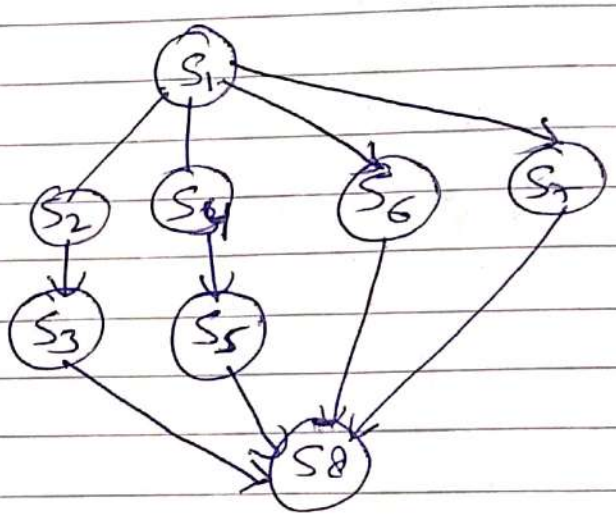
par begin – par end
(or)
cobegin – coend

concurrent

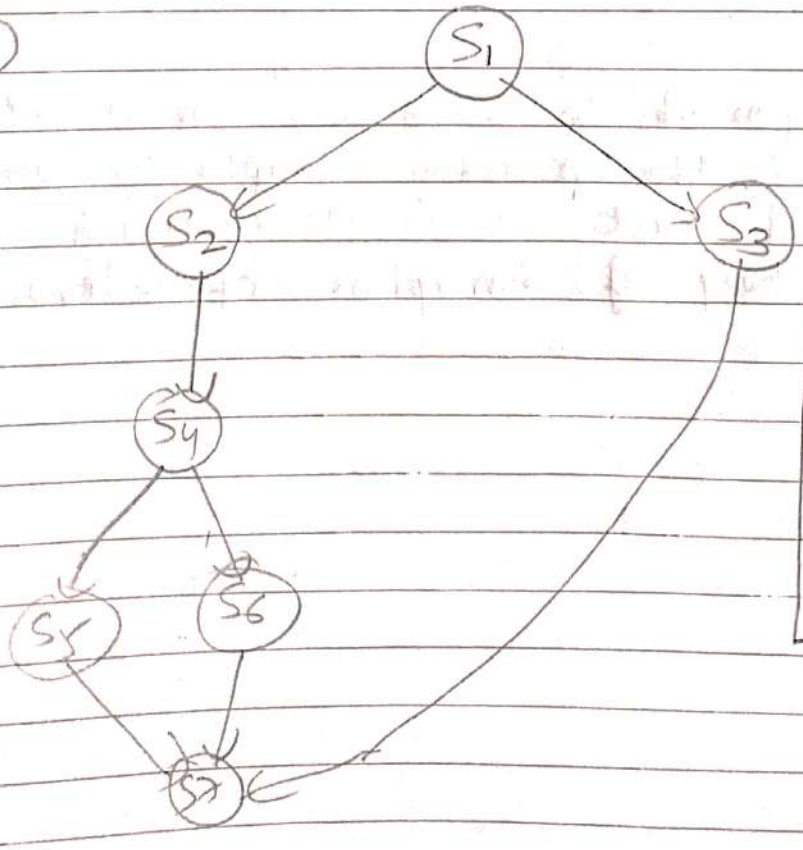


Eg①:

```
begin
  S1;
  parbegin
    begin
      S2;
      S3;
    end
    begin
      S4;
      S5;
    end
    S6;
    S7;
  parend
  S8;
end
```

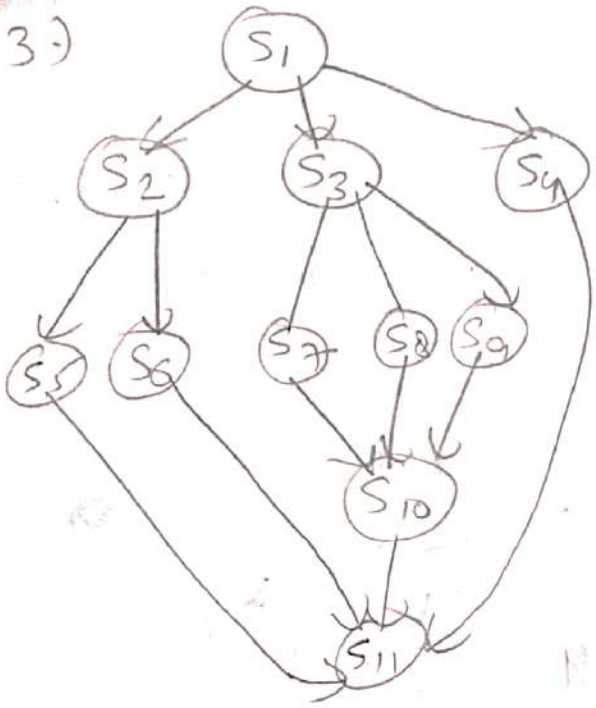


②




```
begin
  S1;
  parbegin
  begin S2;
    S4;
    parbegin
      S5;
      S6;
    parend
  end
  S3
  Parend
  S7;
end
```

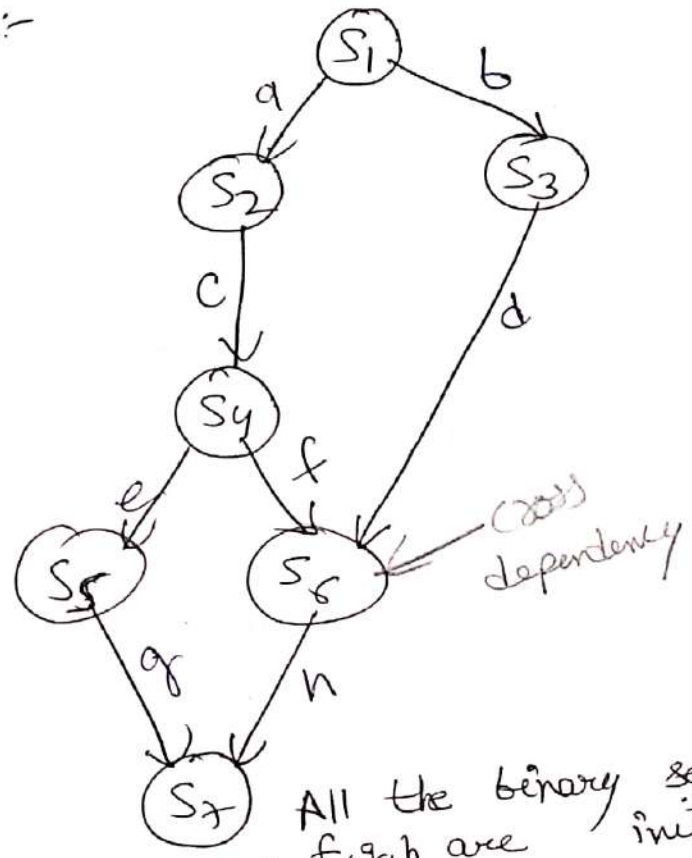
3.)




```
S1;
parbegin
  begin
    S2;
    parbegin
      S5;
      S6;
    parend;
  end
  begin
    S3;
    parbegin
      S7;
      S8;
      S9;
    parend
    S10;
  end
  S4;
parend
S11;
```

NOTE:- It is not possible to write the concurrent program for all the precedence graphs by using parbegin and parend but it is very much possible with the help of semaphore operations.

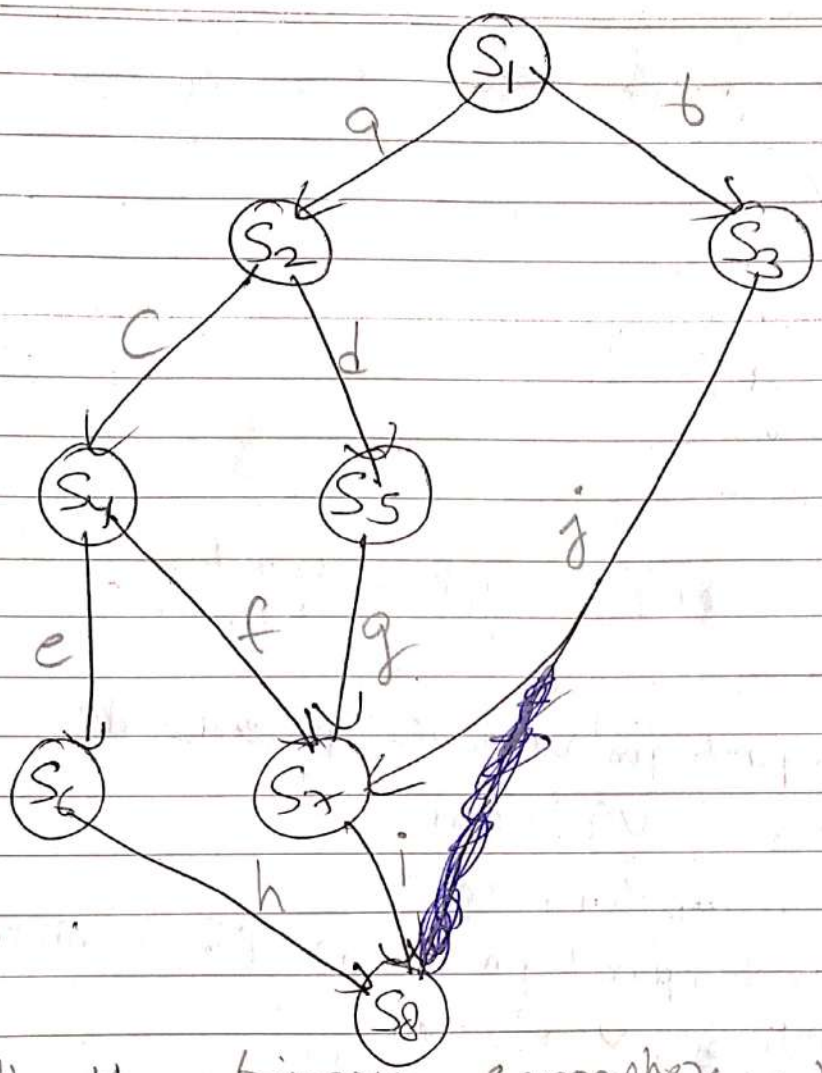
Eg:-



All the binary semaphore variables a, b, c, d, e, f, g, h are initialized to '0'.

```
parbegin
    begin S1, parbegin V(a), V(b), parend, end;
    begin P(a), S2, V(c), end;
    begin P(b), S3, V(d), end;
    begin P(c), S4, parbegin V(e), V(f), parend, end;
    begin P(e), S5, V(g), end;
    begin P(d), P(f), S6, V(h), end;
    begin P(g), P(h), S7, end;
parend
```

Q.)



All the binary semaphore variables a, b, c, d, e, f, g, h, i, j is initialized to 0.

```
parbegin
    begin S1, parbegin V(a), V(b), parend, end;
    begin P(a), S2, parbegin V(c), V(d),
                parend, end;
    begin P(c), S4, parbegin V(e), V(f),
                parend, end
    begin P(d), S5, f      j, V(g),
                end;
    begin P(e), S6, f      j, V(h),
                end;
    begin P(f), P(g), P(j), S7,
                V(i),       end;
```

begin P(b), S_3, V(j), end.
begin P(i). P(n), S_8, end.

9.) Consider the following concurrent program:-

```
int x = 0, y = 0;
parbegin
    begin
        x = 1;
        y = y + x;
    end
    begin
        y = 2;
        x = x + 3;
    end
parend
```

what can be the final values of x and y after completion of above concurrent programs?

I. x = 1, y = 2
II. x = 1, y = 3
III. x = 4, y = 6

which of the following are possible?

a.) only I, II
b.) only II, III
c.) only I, III
d.) All I, II, III

P-91

(Sol.) (b. Ans.) only I, II, III.

2013

Q25)

P-96)

option:)

a.) -2
b.) -1
c.) +1
d.) +2

P(S): w, x	y, z P(S):
I. load R_i, M[x]	I. load R_i, M[x]
II. INCR R_i	II. DECR R_i
III. Store M[x], R_i	III. Store M[x], R_i
V(S);	V(S);

W	X	Y	Z
P(S);	P(S);		
I. load R_w, M[x]	I. load R_x, M[x]	I. load R_y, M[y]	I. load R_z, M[z]
II. INCR R	II. INCR R_x	II. DECR R_y	II. DECR R_z
III. Store M[x], R_w	III. Store M[x], R_x	III. Store M[x], R_y	III. Store M[x], R_z
V(S);	V(S);	V(S);	V(S);
+1	+2	-2	-4

S=2
x = 0, -2, -4, -6, +2

w → I
w → II

4 → I, II, III

2 → I, II, III

w → III
x → I, II, III

P(w) | 0, 1

Q:) Consider the following concurrent program

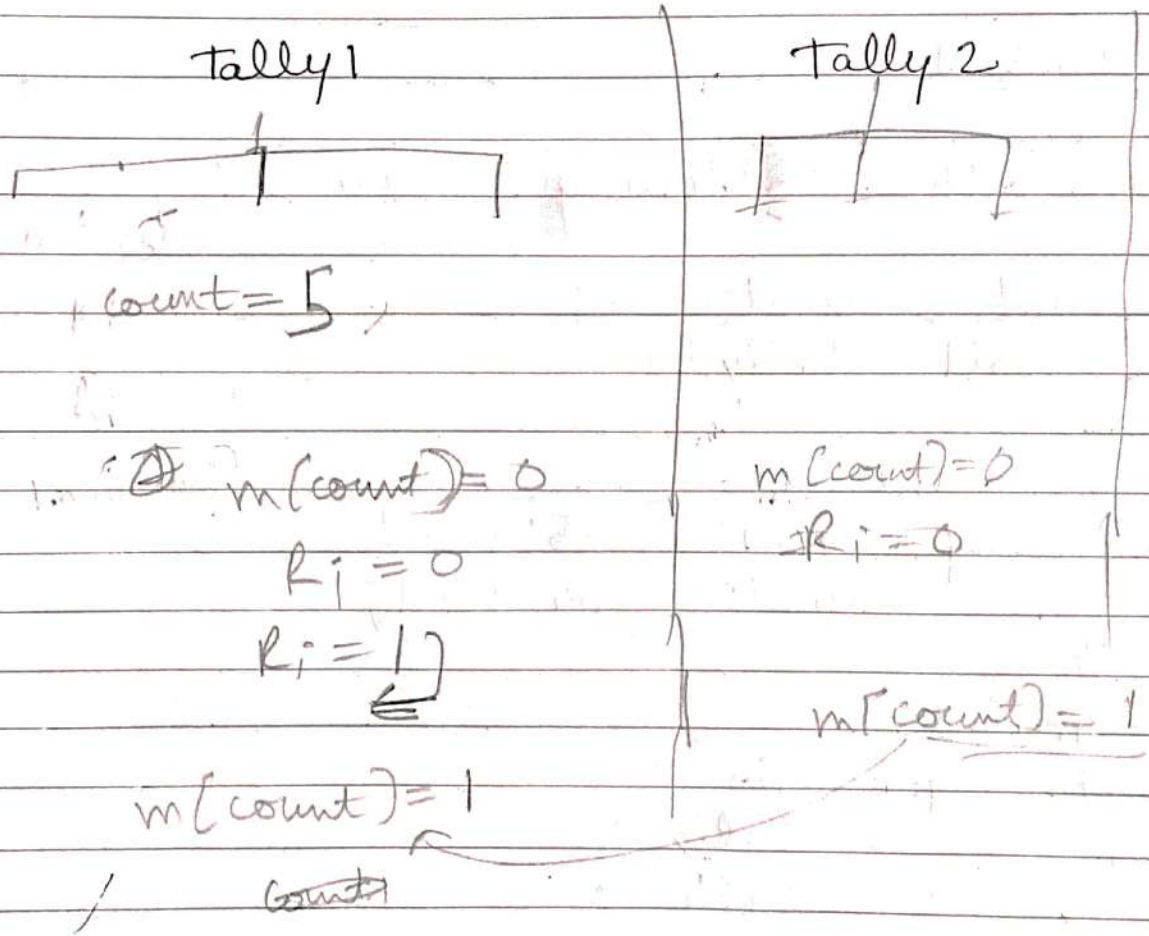
```
int count = 0;
void tally()
{
    int i;
    for(i=1; i<=5; i++)
        count = count + 1;
}
main()
{   parbegin
        tally();
        tally();
    parend
}
```

What can be the minimum final value of count after completion of both the functions of tally()

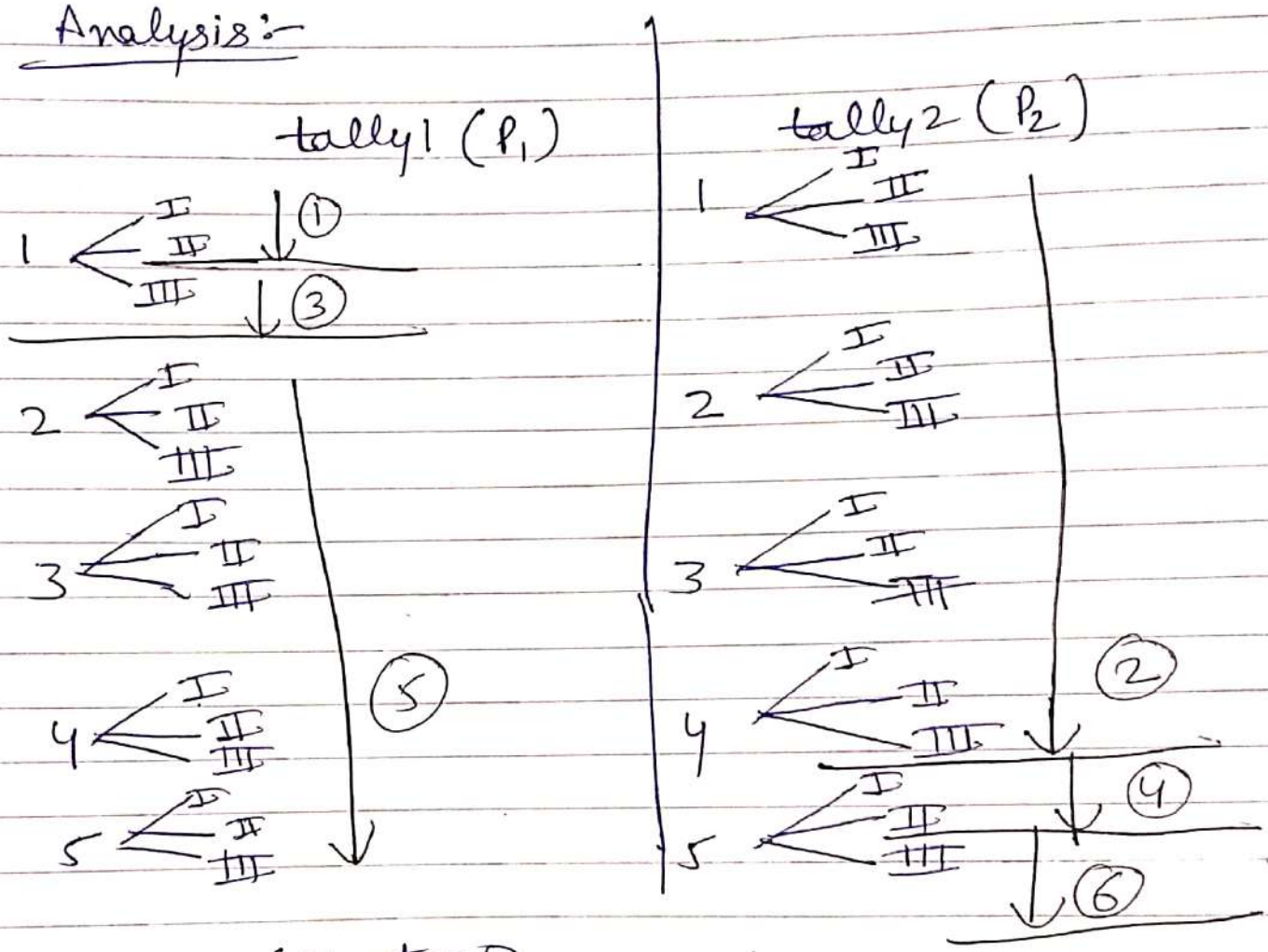
NOTE:- 'count = count + 1' will execute in 3 instructions like load, increment and store.

I. load R_i, m[count]
II. INCR R_i
III. store M[count], R_i

a.) 1 b) 2 c.) 3 d.) 4 e.) 5



Analysis:-



count = 0

P_1 : 1st iteration → I
P_1 : 1st iteration → II

R_1 [∅, 1]

P_2 : 1, 2, 3, 4 iterations

P_1 : 1st iteration → III

P_2 : 5th iteration → I
P_2 : 5th iteration → II

R_2 [Y, 2]

P_1 : 2, 3, 4, 5 iterations

P_2 : 5th iteration → III

Short URL

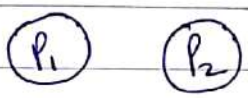
https://bit.ly/2TtKQno -Cbook

Deadlock :-

Def:- The two or more processes are waiting on some event to happen, which never happens then those processes are said to be involved in the deadlock.

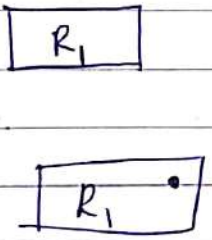
Basics of Deadlocks:-

Processes



Resources

Resources with instances:-

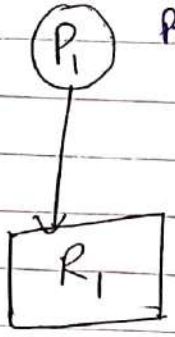


} Single instance of Resource 'R1'.

R_2 : '2' instances of Resource 'R_2'.

Requesting Edge:-

for one instance of Resource 'R_1'.



Process P_1 requesting Resource 'R_1'.

Deadlock

process 'P_2' is requesting for '2' instances of resource 'R_2'.

ALLOCATION EDGE:-

'1' instance of resource 'R_1' allocated to process 'P_1'.

'2' instances of resource 'R_2' are allocated to process 'P_2'.

⇒ The resource request and resource allocation will be represented by using resource allocation graph.

$$RAG = G(V, E)$$

V → Processes, Resources

E → Requesting, Allocation

The Resource Request / Release Life Cycle:-

1.) The process will make the request for the resource.

2.) The Operating system clearly validates the request of the process.

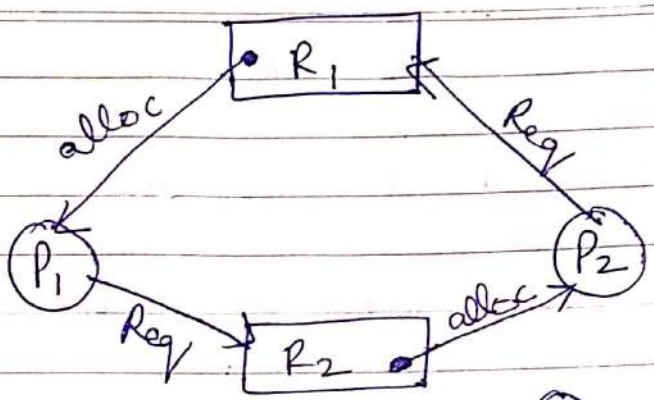
3.) If the request made by the process is valid, then the Operating system will check for the availaibility of the resource.

4.) If the resoure is freely available, then it will allocated to the process, otherwise process has to wait.

5.) If all the resources required further start of execution are allocated then the process will go into execution.

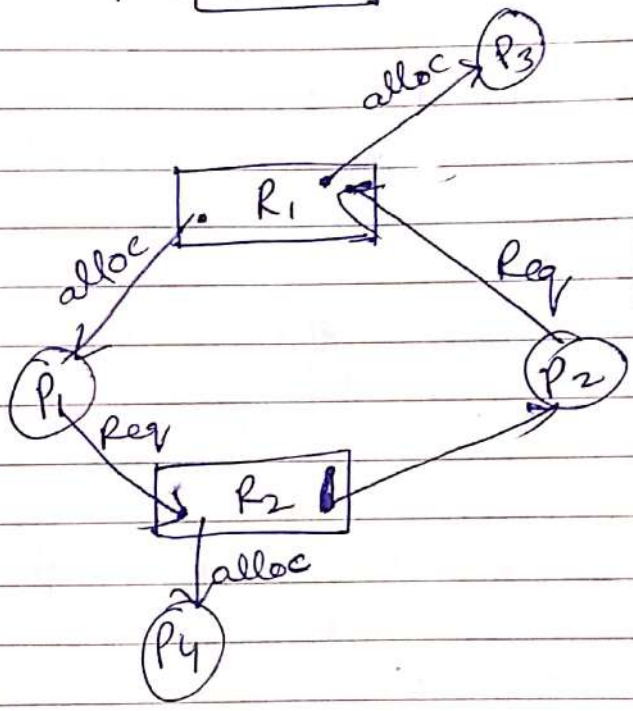
6.) Once the execution of the process is completed then it will release all the resources.

Eg1:



Deadlock exists in the RAG.

Eg2):-

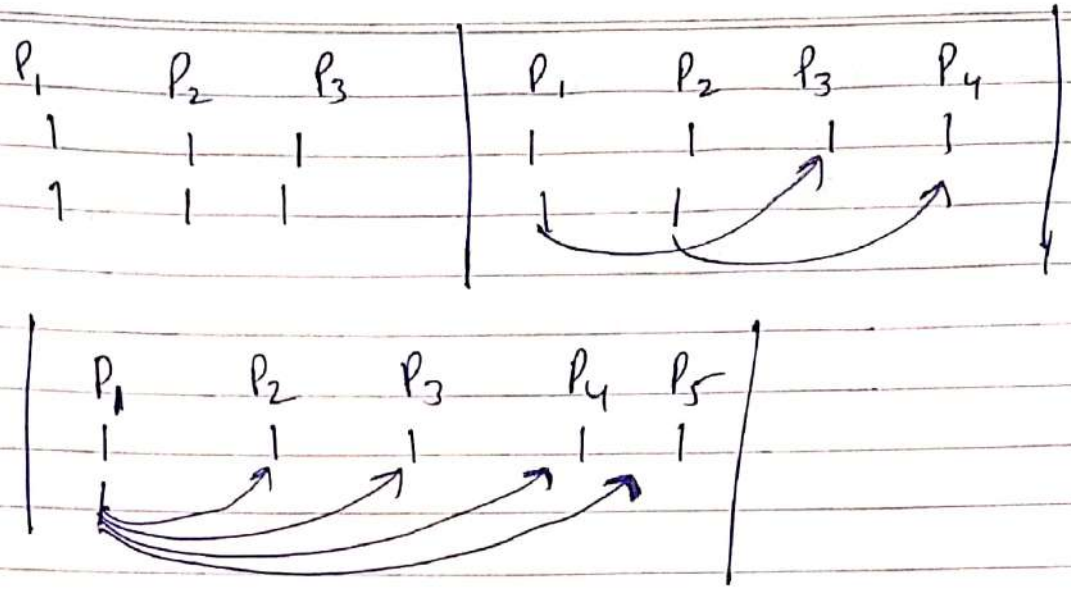


No deadlock exists in a RAG.

Q.) Consider a system which has n processes, 6 → Tape drives. Each process requires 2 tape drives to complete their execution. Then what is the maximum value of n, which ensures deadlock free execution?

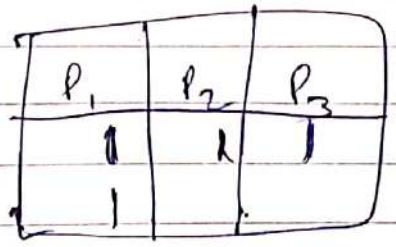
a.) 2
b.) 3
c.) 4
✓ d.) 5

-1



(Q.) Consider a system which has 3 processes & each process requires 2 resources to complete their execution. Then what is the minimum no. of resources are required to ensure deadlock free execution?

a.) 2
b.) 3
✓ c.) 4.
d.) 5

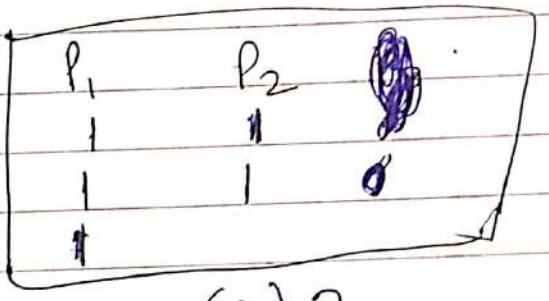


$n+m-1$
$= 3+2-1$
$= 4.$

3 + 3
4

Q.) Consider a system which has n processes and 6 resources. Each process requires 3 resources to complete there execution. What is the maximum value of process to ensure deadlock free execution?

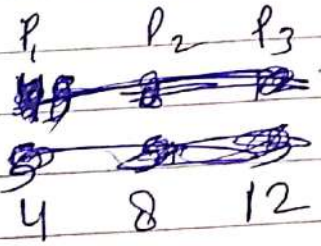
a.) 2
b.) 3
c.) 4
d.) 5



(a.) 2.

Q.) Consider a system which has 3 processes a P_1, P_2 and P_3. The peak demand of each process is 5, 9, 13 respectively for the resource 'R'. What is minimum no. of resources required to ensure deadlock free execution?

a.) 22
b.) 23
c.) 24
d.) 25



+1

25

Page-102

f-20

Q23.)

P_1	P_2	P_3	P_4	
4	3	7	x	9

$3+2+6+x = 20$

$11+x = 20$

$x = 20-11$

$x = 9$.

~~3+2+6+9+1~~

(20)

1.) Necessary Condition :- The deadlock may be possible or may not be possible.

2.) Sufficient Condition :- The deadlock never be possible.

→ (The least upper bound which satisfies the condition is called as sufficient condition).

Page-101)

Q13) $n \rightarrow$ Processes
$m \rightarrow$ Resources.
$P_i \rightarrow S_i$.

Individual

a.) $\forall i, S_i < m$ ✗

$m = 10$

$P_1 = 9$
$P_2 = 9$
$P_3 = 9$
⋮
$P_{100} = 9$

b.) $\forall i, S_i < n$ ✗

c.) $\sum_{i=1}^{n} S_i < m+n$ ✓

$\sum_{i=1}^{n} S_i - n + 1 = m$

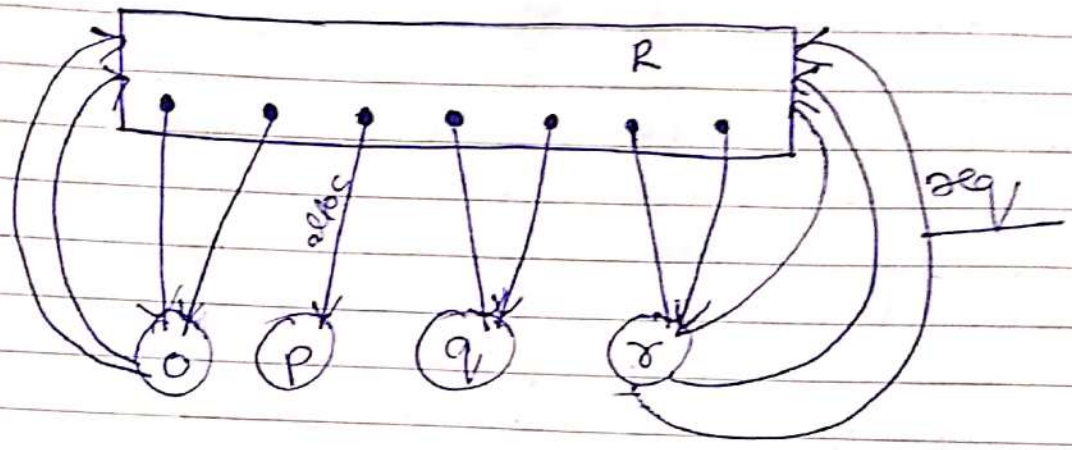
$\sum_{i=1}^{n} S_i = m+n-1$

$\sum_{i=1}^{n} S_i < m+n$

d.) $\sum_{i=1}^{n} S_i < (m \times n)$

Page-100

(Q.10.)



$n = 4 (O, P, Q, R)$

$x_O = 2$	$y_O = 2$
$x_P = 1$	$y_P = 0$
$x_Q = 2$	$y_Q = 0$
$x_R = 2$	$q_R = 3$

$y_P = y_R = 0$

(b. Ans.) $x_P + x_Q \geq \max y_k, \; k \neq P, Q.$

$x_P + x_Q$ → available resources

Concepts of Deadlocks :-

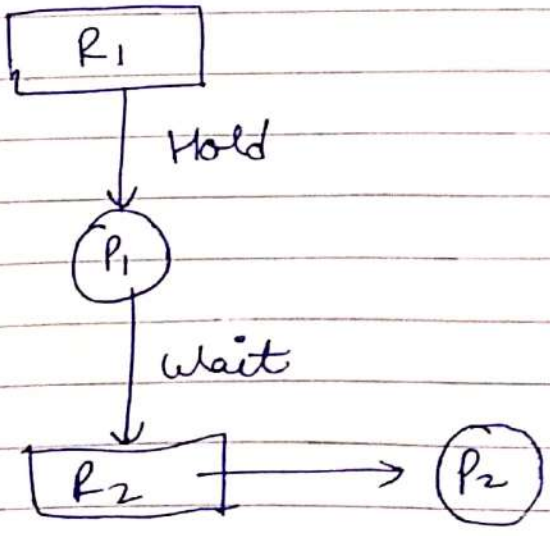
1.) Deadlock characteristics
2.) Deadlock Prevention
3.) Deadlock avoidance
4.) Deadlock Detection
5.) Deadlock Recovery

Deadlock characteristics :-

1.) Mutual Exclusion :-> The resource has to be allocated to only one process or it is freely available.

→ There should be a one-to-one relationship between the resource and the process.

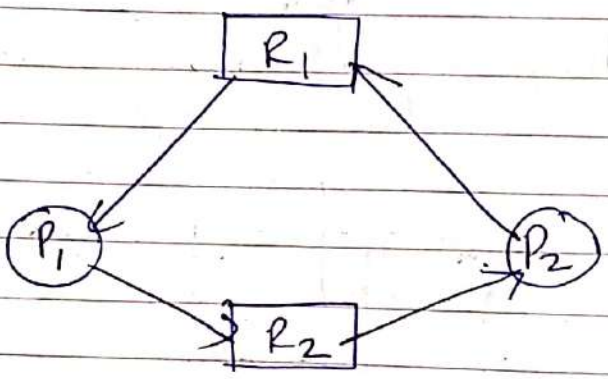
2.) Hold and wait :- The process is holding the resource and waiting on some other resource simultaneously.

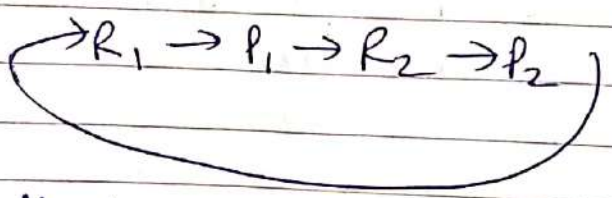


3.) No Preemption :- The resource has to be "voluntarily" released by the process after completion of the execution.

→ It is not allowed to forcefully preempt the resource from the process.

4.) Circular Wait :- The processes are circularly waiting on each other for the resources.

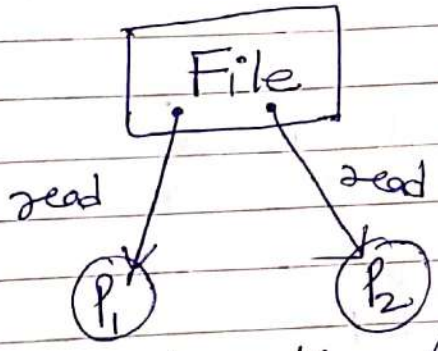




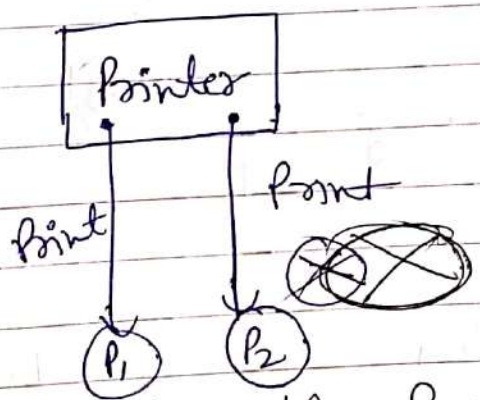
NOTE :- If all these above four conditions are existing simultaneously in the system, then definitely there exist a deadlock.

DEADLOCK PREVENTION:-

1.) Mutual Exclusion:- It is not possible to dissatisfy the mutual exclusion always because of sharable and non sharable resources.



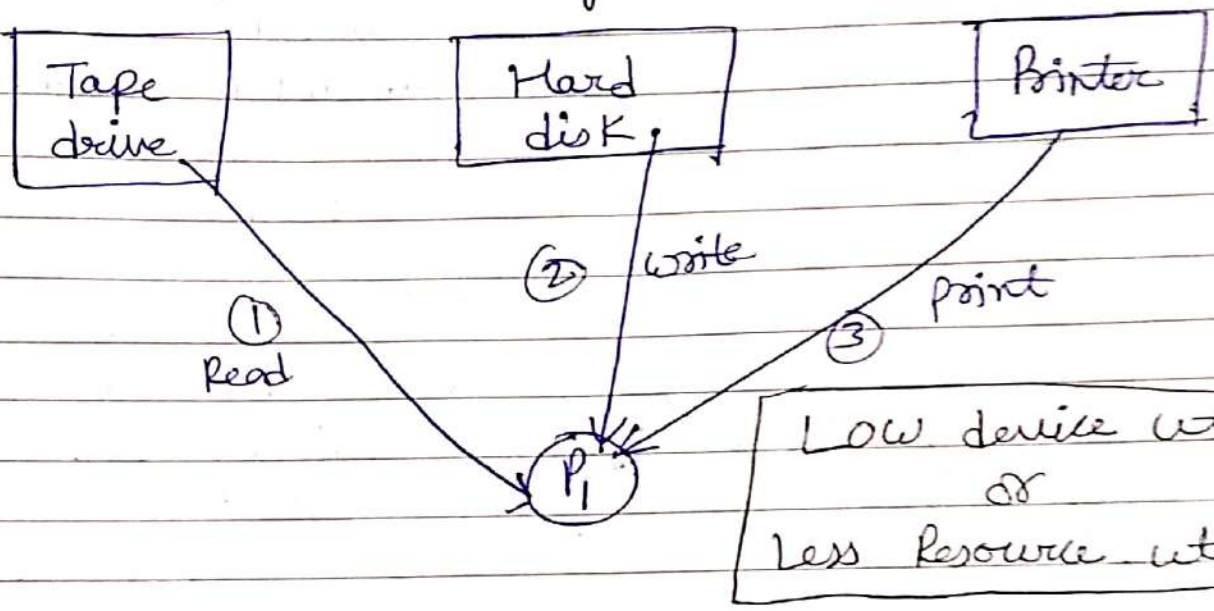
sharable Resource



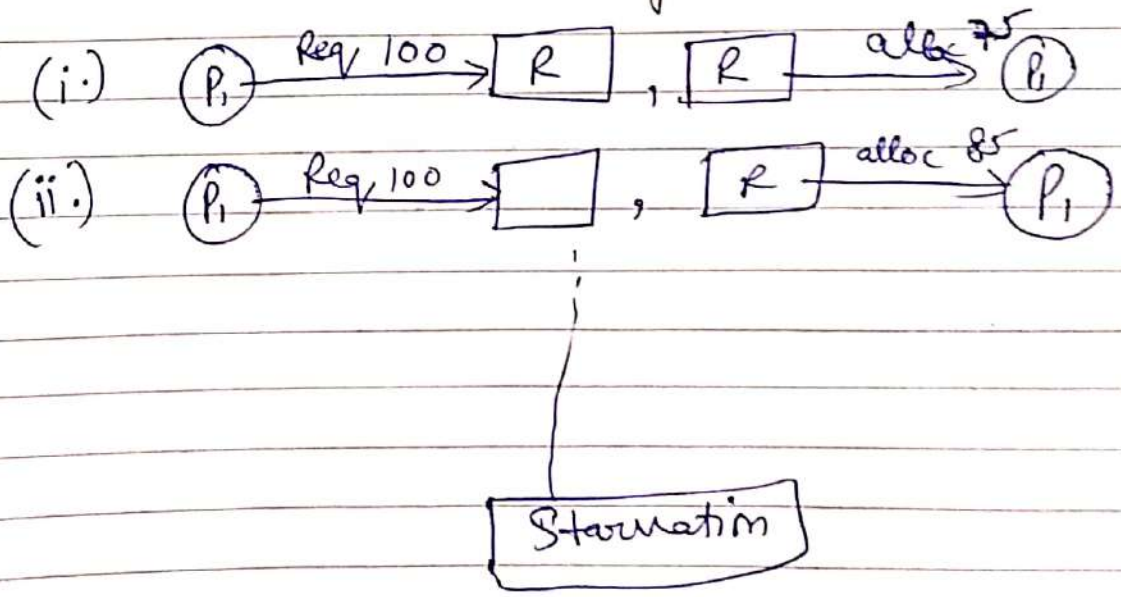
Non sharable Resource.

2.) HOLD AND WAIT:-

(i) Allocate all the Resources required by the process befor start of execution.



(ii.) The process should release all the existing resources, befor making the new Request.



NOTE:- If the above condition is followed to dissatisfied hold & wait, then it may be possible that the process will go into Starvation.

3.) NO ~~x~~ PREEMPTION :- The process P_1 is requesting for resource 'R_1'.

If the Resource R_1 is free then it will be allocated to the process 'P_1'.

The Resource R_1 is not free, ~~then~~ and it is allocated to some other process 'P_2'.

If the process 'P_2' is in the execution, then process 'P_1' has to wait.

The process P_2 is not in the execution and it is waiting on some other Resource 'R_2'.

↳ Preemption. Preempt the resource R_1 from process P_2 and allocate it to the Process 'P_1'.

4.) CIRCULAR WAIT :-

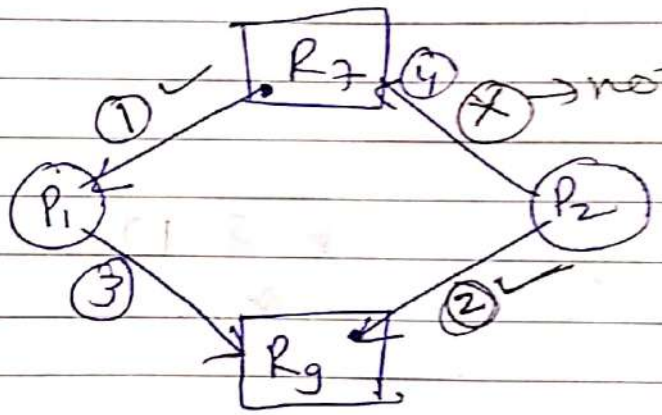
→ The resources will be assigned with a unique numerical numbers.

→ The process can request for the resource only in the increasing (or decreasing) order of enumeration (numbering).

increasing order

1.) $P_1 \rightarrow R_7$ ✓ | $P_1 \rightarrow R_5$ × | $P_1 \rightarrow R_9$ ✓

2.)



DEADLOCK AVOIDANCE :-

Deadlock avoidance is implemented by using Bankers algorithm :-

Bankers Algorithm :-

Processes → P_0, P_1, P_2, P_3, P_4
Resources → A, B, C.

	Max Need			Current Allocation			Current available			Remaining Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0 →	7	5	3	0	1	0	3	3	2	7	4	3 → P_0
P_1 →	3	2	2	2	0	0	5	3	2	1	2	2 → P_1
P_2 →	9	0	2	3	0	2	7	4	3	6	0	0 → P_2
P_3 →	2	2	2	2	1	1	7	5	3	0	1	1 → P_3
P_4 →	4	3	3	0	0	2	~~5~~	~~5~~	5	4	3	1 → P_4
				7	2	5	~~10~~	10	7			

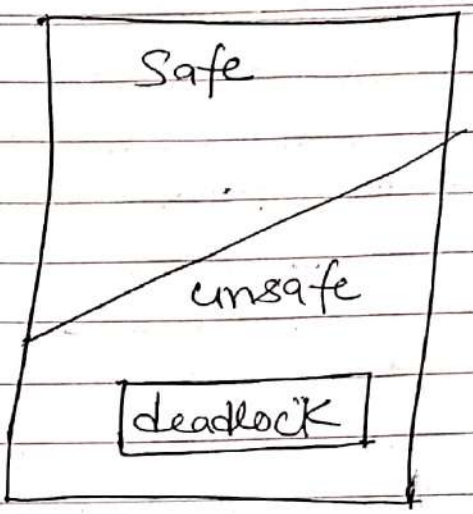
Total Available

A	B	C
↓	↓	↓
10	5	7

Remaining need = Max need − current allocation

Current available = total Available − total Allocation

Safe Sequence :- P_1, P_3, P_0, P_2, P_4



→ If we can satisfy the remaining need of all the processes with the current available processes then system is said to be in the safe state otherwise system is said to be in the unsafe state.

→ If the system is in the unsafe state, then it is possible for the deadlock.

→ The order in which we satisfy the remaining need of all the processes is called safe sequence.

→ Safe sequence may not be unique, there can be multiple safe sequence.

→ The unsafe state purely depends on the behaviour of the processes.

→ The deadlock avoidance is less restrictive than deadlock prevention.

→ Each and every time when the process is requesting

P-99

Gate

Q2.)

	Max Need	Current Allocation	Current Available	Remaining Need
	A B C	A B C	A B C	A B C
$P_0 \rightarrow$	6 5 4	0 3 4	4 3 1	6 2 0
$P_1 \rightarrow$	3 4 2	2 1 2	6 4 3	1 3 0
$P_2 \rightarrow$	1 0 4	0 0 2	6 7 7	1 0 2
$P_3 \rightarrow$	3 2 5	1 2 1		2 0 4

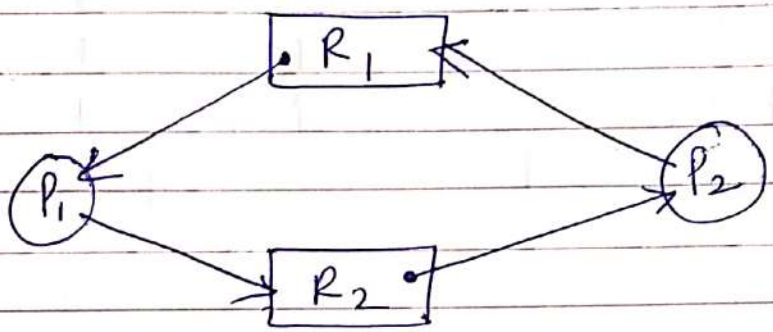
P: P_1, P_0, P_2, P_3.

Q: P_1, P_2, P_3, P_0

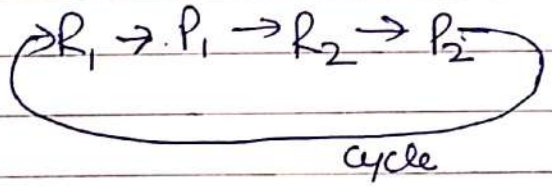
(a) P & Q only

DEADLOCK DETECTION :-

1.) The Resources are of single Instance Type :-



Rem : Cycle detection algorithm.



NOTE ⇒ If all the resources are of single instance type, then the cycle in the resource allocation graph is necessary and sufficient condition for occuring of deadlock.

⟶ If all the resource are of non single instance type then cycle in the resource allocation graph is just a necessary condition but not sufficient condition for occuring of a deadlock.

(ii) The resource are of Multiple instance Type :-

	Current Allocation A B C	Current available A B C	Remaining Need A B C
P_0	→ 0 1 0	0 0 0	0 0 0 → P_0
P_1	→ 2 0 0		2 0 2 → P_1
P_2	→ 3 0 3		0 0 0 → P_2
P_3	→ 2 1 1		1 0 0 → P_3
P_4	→ 0 0 2		0 0 2 → P_4

Safe sequence :- P_0,

What happens if the process P_2 is requesting for additional resources of $P_2 \Rightarrow (0, 0, 1)$.

$$\begin{array}{r} 000 \text{ (old)} \\ +\ 001 \text{ (New)} \\ \hline 001 \text{ (latest)} \end{array}$$

Current allocation	Current available	Remaining Need
A B C	A B C	A B C
0 1 0	0 0 0	0 0 0 → P_0
2 0 0	0 1 0	2 0 2 → P_1
3 0 3		0 0 1 → P_2
2 1 1		1 0 0 → P_3
0 0 2		0 0 2 → P_4

Safe Sequence :- P_0, ⇒ unsafe ⇒ deadlock

P_1 P_2 P_3 P_4

DEADLOCK RECOVERY:-

1) KILLING THE PROCESS:-

a) Kill all the processes, which are involved in the deadlock.

b) Kill one after the other.
→ Low priority
→ % of process completion

P_1 ↓ 95% P_2 ↓ 5%.

→ No of Resources the process is Holding

P_1 ↑ Hold 20R P_2 ↑ Hold 1R

→ No. of Resources the process Requesting.

P_1 ↓ Req 1R P_2 ↓ Req 20R

2.) Resource Preemption :-

→ Resources will be preempted from the processes which are in the deadlock and preempted resources will allocated to some other processes, so that there is a possibility of recovery, the system from the deadlock.

→ If the above condition is followed to recover the system from deadlock, then there may be a possibility of the process to into starvation.

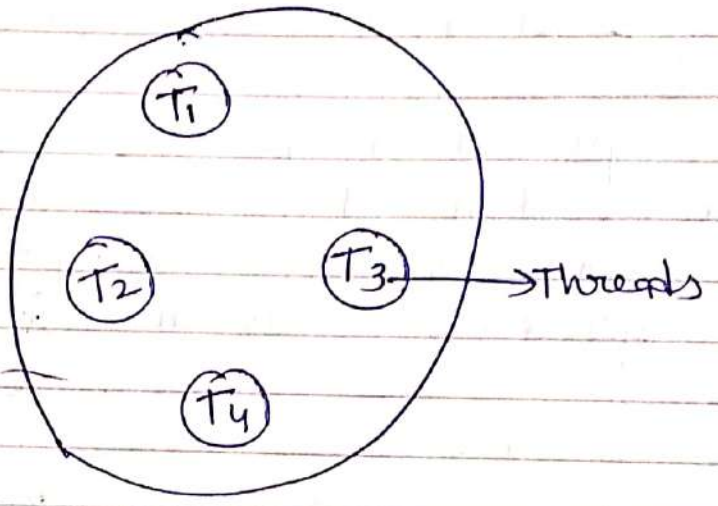
3.) Ostrich Algorithm :-

→ ignore the deadlock.

THREADS:-

Def:- The light weight process

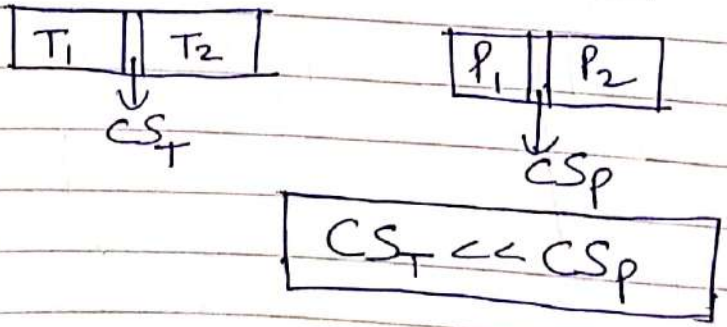
→ No. of instructions will be less.
→ context (Attributes) will be less.



Advantages of Threads:-

1.) **Responsiveness:-** If the process is divided into multiple threads, then if one thread is completed the execution immediately the output will be responded, this response will be faster compare to the response of the process.

2.) **Faster context Switches:-**

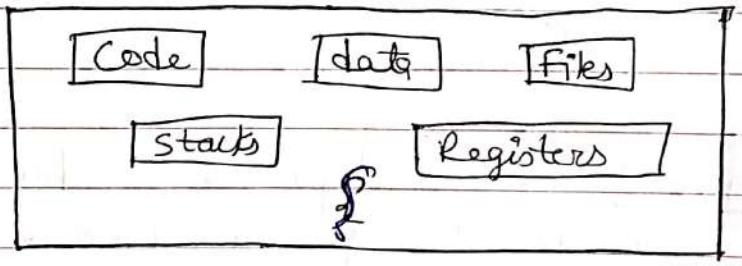


→ The context switching time between the threads will be very less compare to the context switching time b/w the processes because threads will have less context compare to the processes.

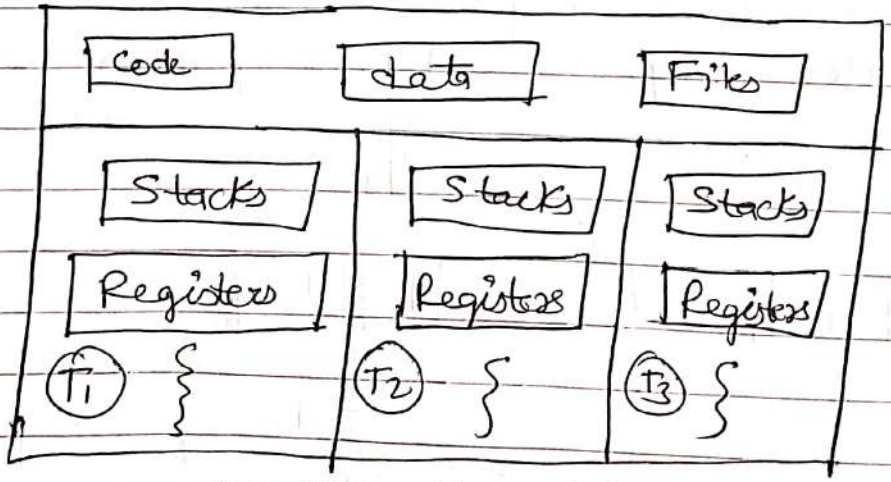
3.) Effective Utilization of Multi processor Systems:-

If the process is divided into multipk threads then different threads can be schedule on to different processes, so that process execution will be faster.

4.) Resource Sharing:



Singe threaded process



Multi threaded proces

→ The Resources like code, data, (Global variables), files, heap and memory will be shared among all the threads within the process, but stacks and registers cannot be shared and every thread will have its own stacks and Registers.

5.) Enhanced Throughput of the System:-

→ If the process is divided into multipk threads and if we consider as one thread as one job, then No. of jobs completed per unit time will increase and throughput of the system will be enhanced.

6.) ECONOMICAL:-

→ The implementation of the threads does not require any cost and there are various programming language API which support Implementation of the threads.

Eg:- JAVA API

The Thread are further categorized into two types :-

1.) User level Threads.
2.) Kernel level Threads.

User level Threads	Kernel level Threads
1.) User level Threads are implemented by user or programmer.	1.) Kernel level threads are implemented by OS.
2.) O.S. does not know about user level threads, it views User level thread as a process only. It means O.S. can not recognize user level threads.	2.) Kernel Level Threads are recognized by operating System.
3.) If one user level thread is performing blocking system call, then the entire process will be blocked.	3.) If one Kernel level Thread is performing blocking system call, then another thread will continue the execution.
4.) User Level Threads are designed as dependent Thread.	4.) Kernel Level Threads are designed as independent threads.

5.) Implementation of user level thread is easy or simple.	5.) Implementation of Kernel level Threads is complicated.
6.) User level Threads will have less context.	6.) Kernel level Threads will have more context.
7.) No Hardware support Required for the User level Threads.	7.) Scheduling of Kernel level Threads requires Hardware Support.

NOTE:- "GANG SCHEDULING" is used in the Threads.

Workbook

P-79

Q14.) (d. Ans.)

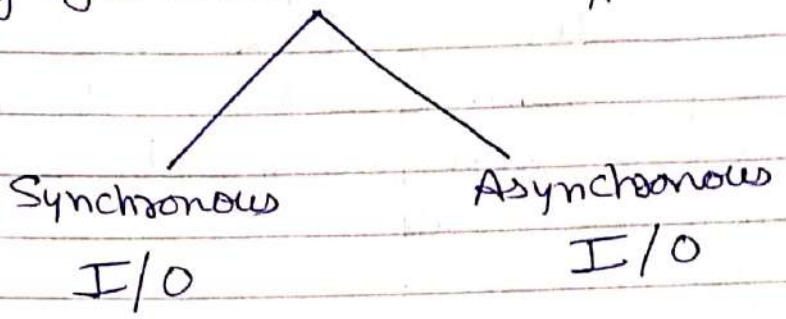
P-80

Q18.) (a) only i, ii & iii.

Q20.) (d) both heap & global variable.

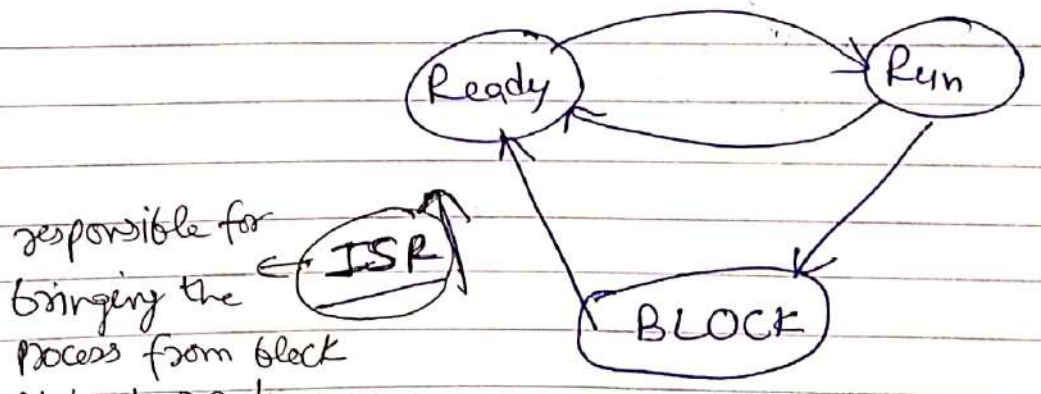
Q16.)

I/O Operation :- I/O of the process is categorized into two types



1.) SYNCHRONOUS I/O :- In Synchronous I/O, the process performing I/O operation will be placed in the blocked state till the I/O operation is completed.

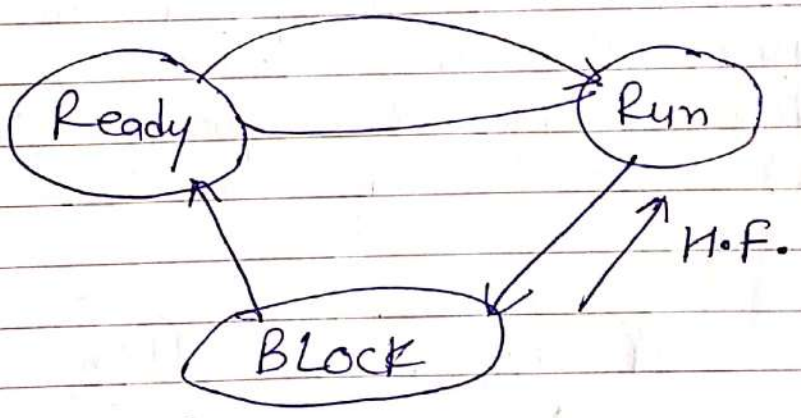
→ Once I/O is completed, an interrupt Service Routine will be initiated which brings the process from block state to ready state.



2.) ASYNCHRONOUS I/O :- In asynchronous I/O while initiating the I/O request, the "handler function" will be registered.

The process is Not placed in the blocked state and it continues to execute the remaining code after initiating the I/O request.

→ Once I/O is completed, the signal mechanism is used to notify the process that the data is available and "Register handler function" will be asynchronously invoked.



O.S. by
William Stallings

MEMORY MANAGEMENT:-

Main memory
physical memory
primary memory
RAM memory

FUNCTIONALITY:- Allocating and deallocating the memory to the process.

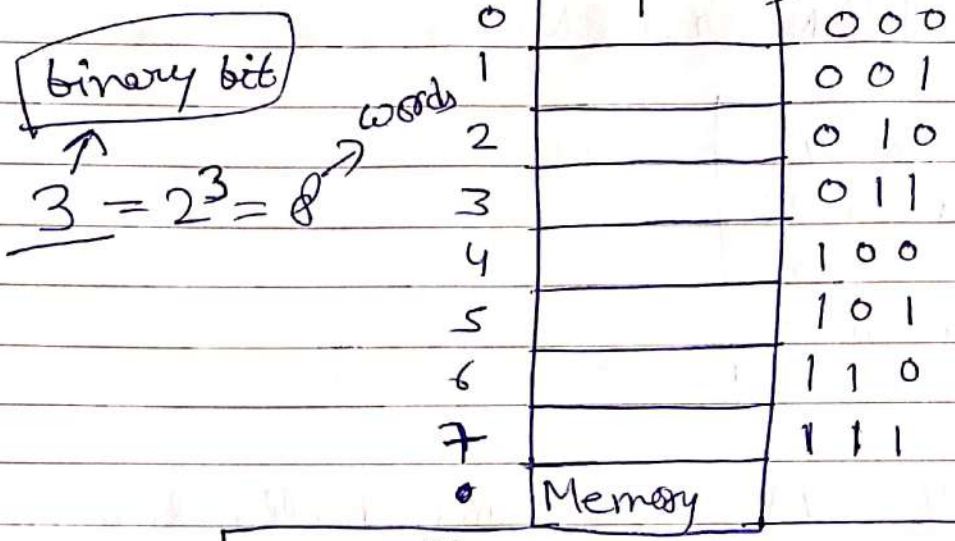
GOAL:- The efficient Utilization of memory by minimizing the internal and external fragmentation.

Memory →
- → Byte Addressable. Eg: 512 KB
- → Word Addressable. Eg: 512 KW

$2^2 = 4$ bytes.
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024 \cong 10^3 = 1K$
$2^{20} = 1M$
$2^{30} = 1G$

1 Byte = 8 bits
1 word = ?

$2^{40} = 1T.$



Capacity of the memory = No. of words in memory × word size

$= 8 \times 16$ bits
$= 8 \times 2$ bytes.
$= 16$ bytes.

Q.) Consider a system which has 256 Mega words and each word is having size of 128 bits then what is the capacity of memory in bytes

$= 128$ bits $\times 256$ Megawords
$= 16 \times 256$ Megawords
$= 2^8 \times 2^{20} \times 2^4$ B
$= 2^{32}$ Bytes $= 2^2 \times 2^{30}$ B
$= 4 \times 1$ GB
$= 4$ GB.

32
=

264
×6
384

Q.) Consider a system which have 64 Gigawords & each one having size of 48 bits then what is the capacity of Memory in bytes?

$\Rightarrow 2^{30} \times 2^{6} \times 6$

$= 384$ GB.

Q.) Consider a system where 32 binary bits are used to represent all the words of the memory. And each word is having the size of 32 bits. What is the capacity of memory in bytes?

$= 4 \times 2^{32}$

$= 4 \times 2^{32}$

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

$= 16 \times GB$

$2^6 \times 2^4 = 64$

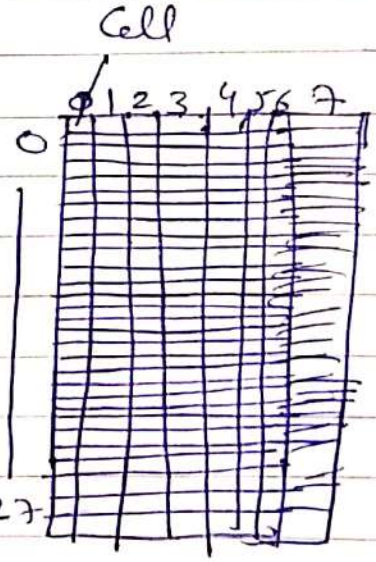
RAM CHIP IMPLEMENTATION :-

RAM CHIP SIZE = 128 Byte x 16 byte

organize the main memory capacity of 16 KB.

1.) No. of RAM chips required?
2.) Draw the memory organization? (byte)
3.) what is the size of Decoder required?
4.) Draw the memory organization map with the word size of 16 bits.

RAM CHIP SIZE = 128 B

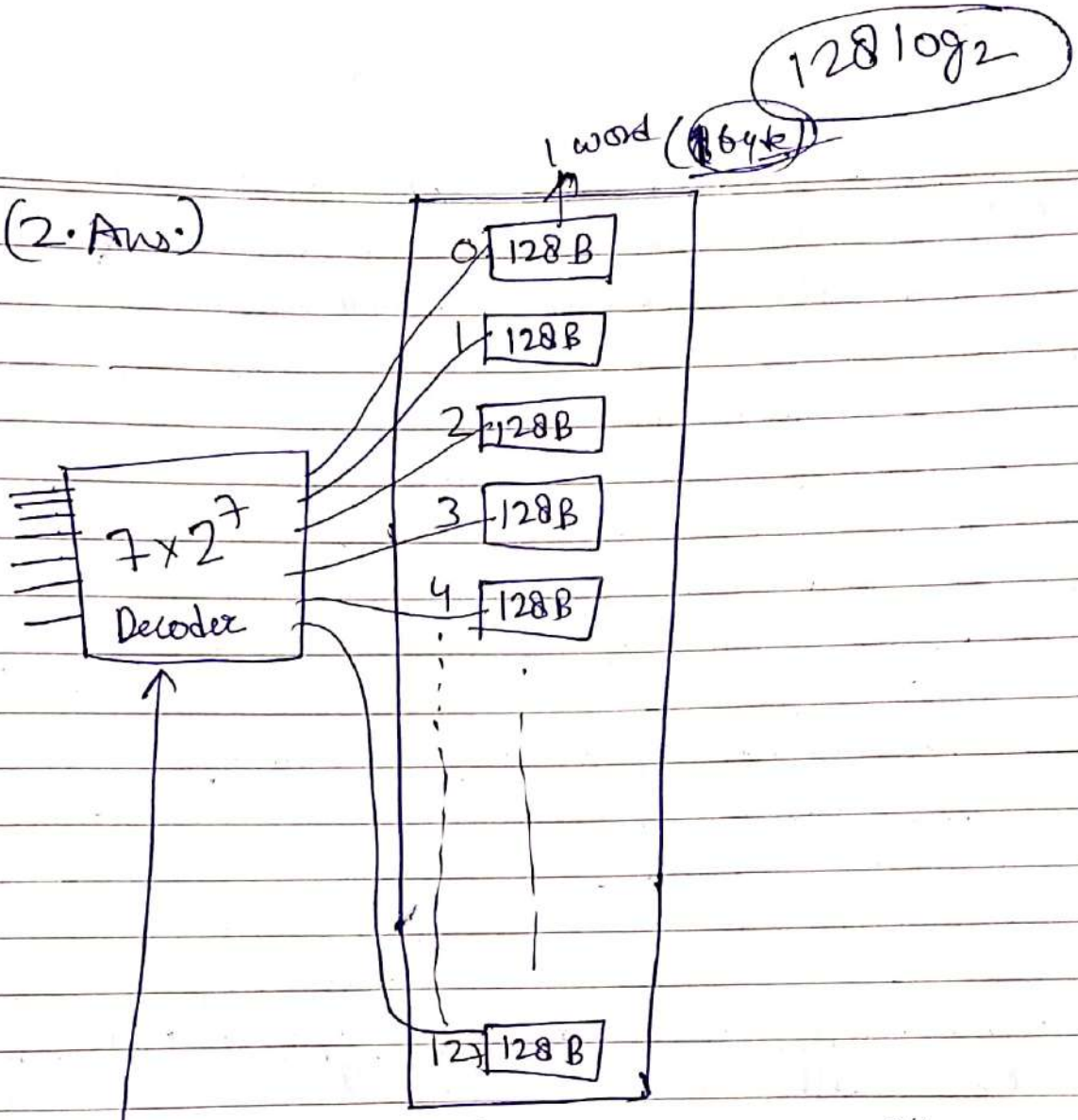


Q1.) No of RAM chips required = ?

$$= \frac{16\,KB}{128\,B} = \frac{2^4 \times 2^{10}}{2^7} = 2^7 = 128.$$

128 log 2

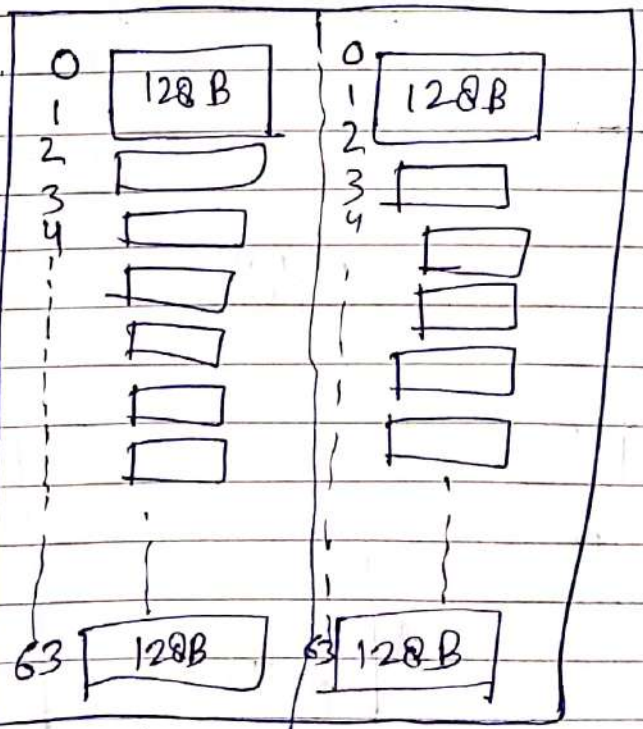
(2. Ans.)



16 KB of memory with word size as 8 bits (1 byte).

(3. Ans.)

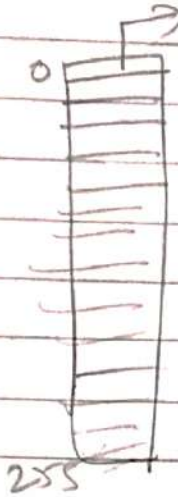
(4. Ans.)



16 KB of memory with word size as 16 bits

$\frac{2^{20}}{2^5} = 2^{15} =$ ~~2048~~ $\frac{32\text{byte}}{2^5}$ ~~256~~ = 32

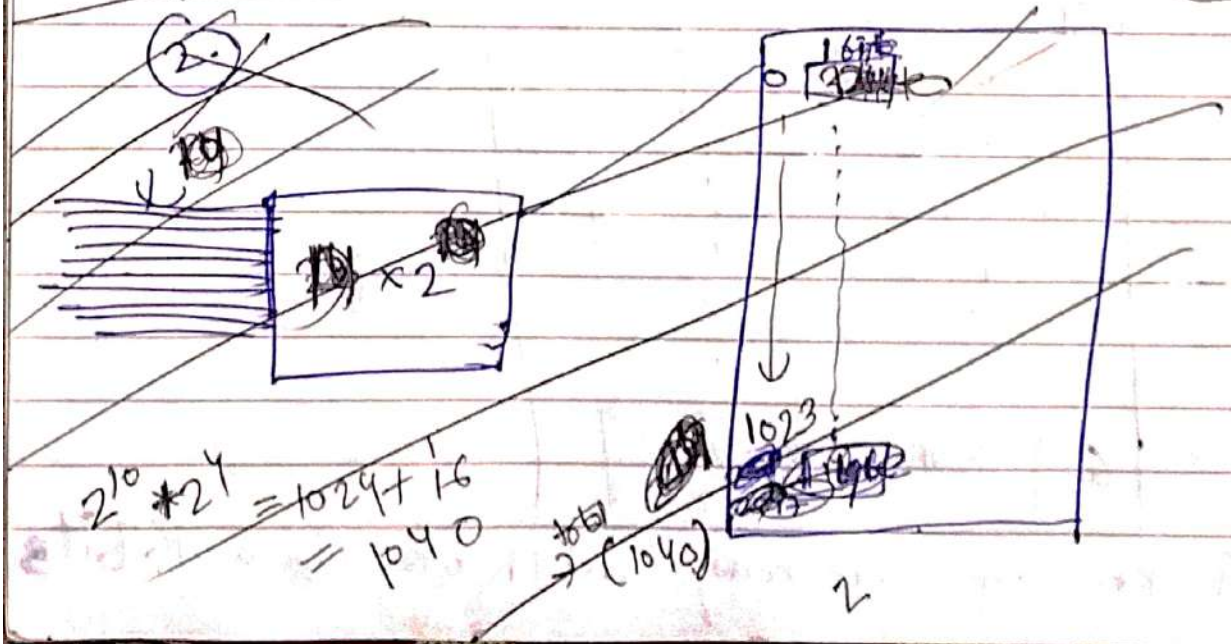
Q2.) RAM chip size = 256 × 1-bit
Orgenize the main memory capacity of 32 MB.



① RAM CHIP SIZE = $\frac{32\text{ MB}}{32\text{ B}}$

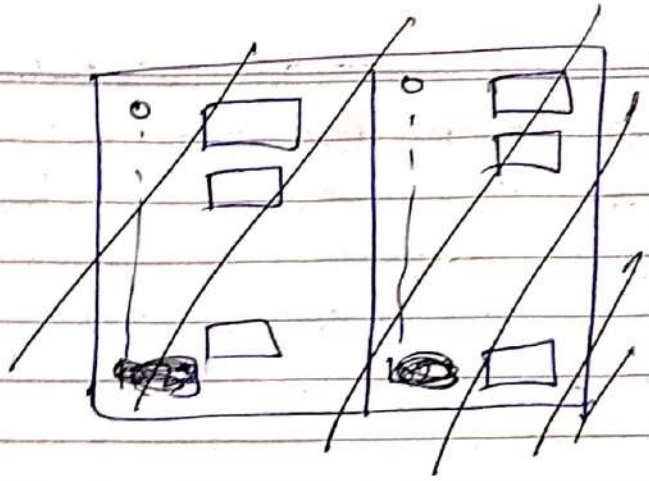
$= \frac{2^{20} \times 32}{32}$

$2^{20} =$ 1024

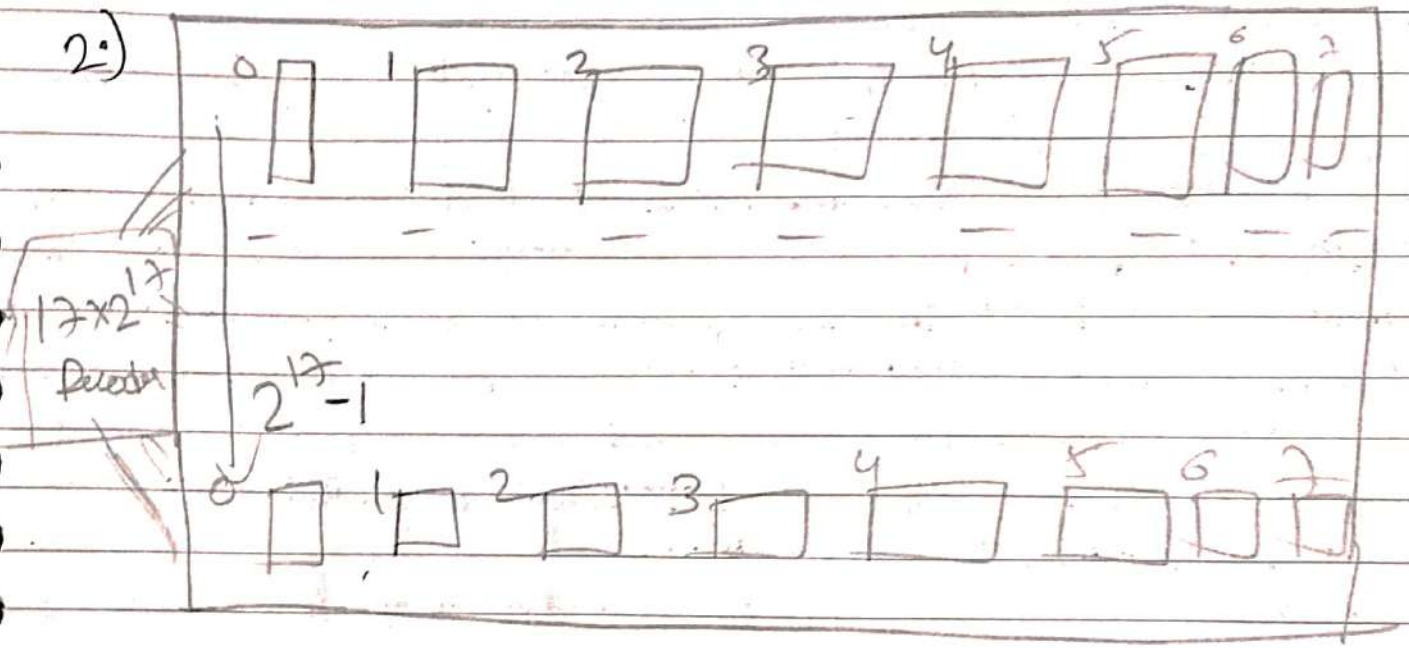


$2^{10} * 2^4 = 1024 + 16$
$= 1040$

4.)



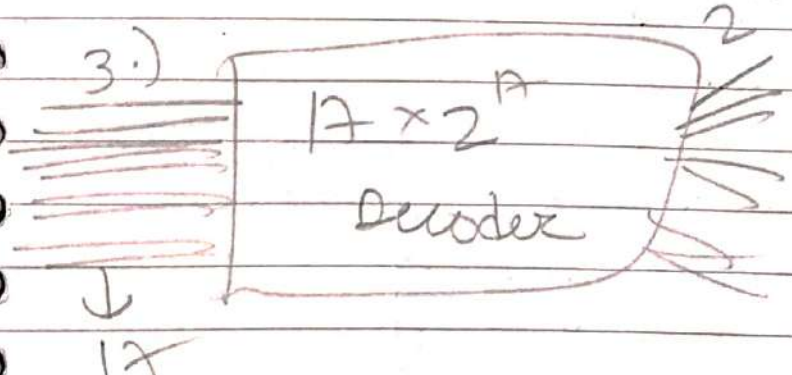
2.)



$$= \frac{2^{20}}{8} = \frac{2^{20}}{2^3} = 2^{17} =$$

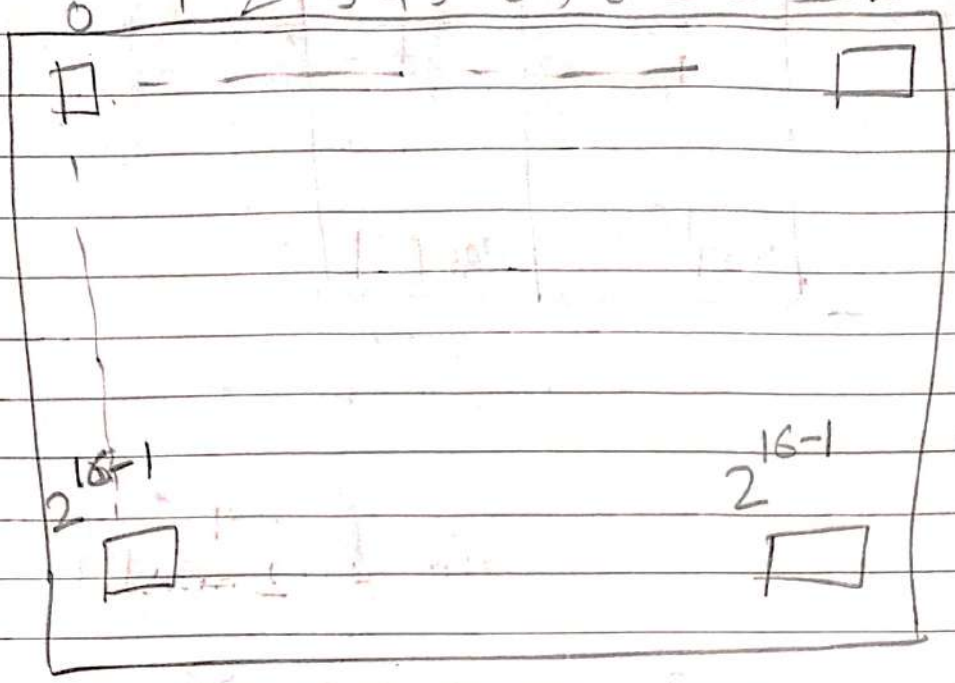
32 MB of memory with word size 8 bits.

3.)



$\frac{2^{20}}{8}$ 22

4.) 0 1 2 3 4 5 6 7 8 - - - 15



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

32 MB of memory with word size of 16 bits.

$\frac{1024}{8} = 128$

64

Q.) RAM chip Size = 512 × 2-bits
organize the main memory capacity of 16 MB.

①

$$= \frac{16\,MB}{512 \times 2\,bits}$$

$$= \frac{16M \times 8\,bits}{512 \times 2\,bits}$$

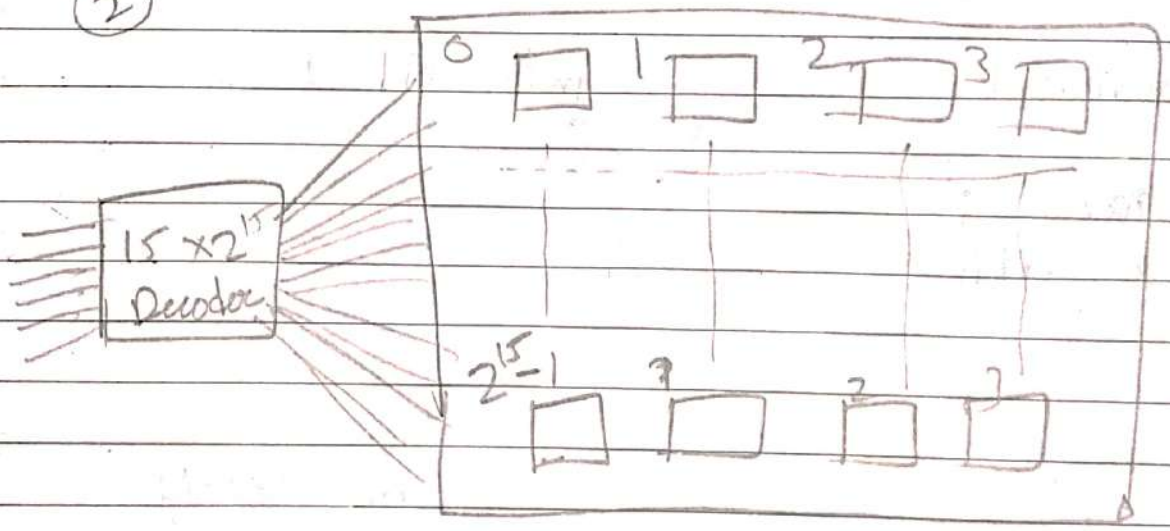
$$= \frac{2^4 \times 2^{20} \times 2^3}{2^9 \times 2^1}$$

$$= \frac{2^{27}}{2^{10}}$$

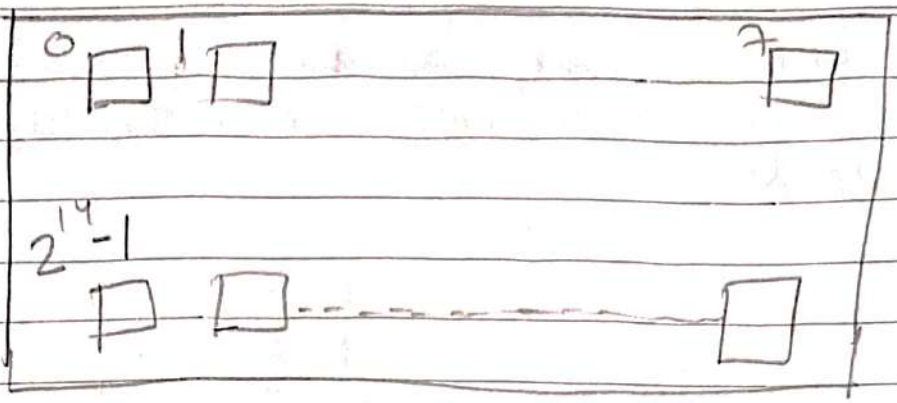
$$= 2^{17}$$

$$= \boxed{128}$$

②



$\frac{2^{17}}{2^3}$



16 MB, 16 bits

LOADING, LINKING AND ADDRESS BINDING :-

Loading :- Bringing the program from secondary memory to main memory is called loading

LINKING :- Establishing the linking between all the modules of all the functions of a program in order to continue the program execution is called as linking.

```
main()
{
  int x;
  x = add();
  .
  .
  .
}
```

linking

```
int add()
{
  ----
  ----
  ----
  ----
  ----
  return total;
}
```

Loading and Linking is further categorized into two types:-

1.) Static
2.) Dynamic

1.) STATIC:- Loading the entire program into memory before start of program execution.

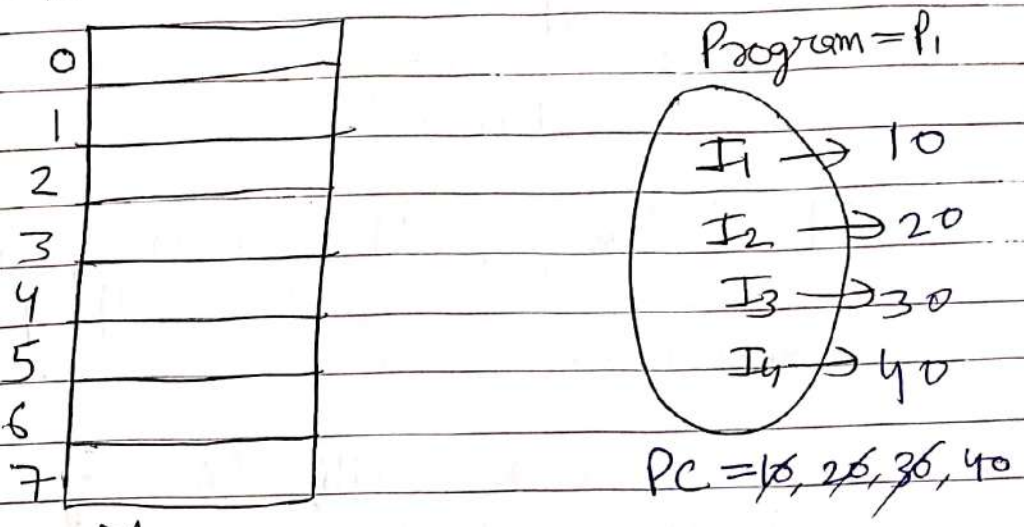
→ Inefficient utilization of memory, because whether it is required or not required, the entire program will be load into main memory.

→ Program execution will be faster.
→ If the static loading is used, then accordingly static linking will be applied.

2.) DYNAMIC:- Loading the program into memory on demand is called as Dynamic loading.

→ Efficient Utilization of a memory.
→ Program execution will be slower.
→ If the dynamic loading is used accordingly, dynamic linking will be applied.

ADDRESS BINDING :-



Association of program instructions and data to the actual physical memory locations is called as address binding.

→ Address Binding is categorized into 3 types

- → Compile Time AB (Compiler)
- → Load Time AB (Loader)
- → Execution Time A.B or Dynamic AB. (processor)

1.) Compile Time AB :- If the compiler is responsible of performing address binding, then it is called as compile time addressing Binding.

→ This type of Address Binding will be done before loading the program into memory.

→ The compiler requires to interacts with the O.S. memory manager to perform

compile Time Addressing binding.

2.) LOAD TIME A.B :-

→ This type of Address binding will be done after loading the program into memory.

→ Load Time AB will be done by O.S. memory manager ("Loader")

3.) EXECUTION TIME AB OR DYNAMIC AB :-

→ The Address binding will be postponed even after loading the program into memory.

→ The program will keep on changing the locations in the memory till the time of program execution.

→ This type of Address Binding will be done by the processor at a time of program execution.

NOTE→ Many of the ~~process~~ operating system practically implement dynamic loading, dynamic linking and dynamic Address binding.

Ex:- Windows, Unix, Linux.

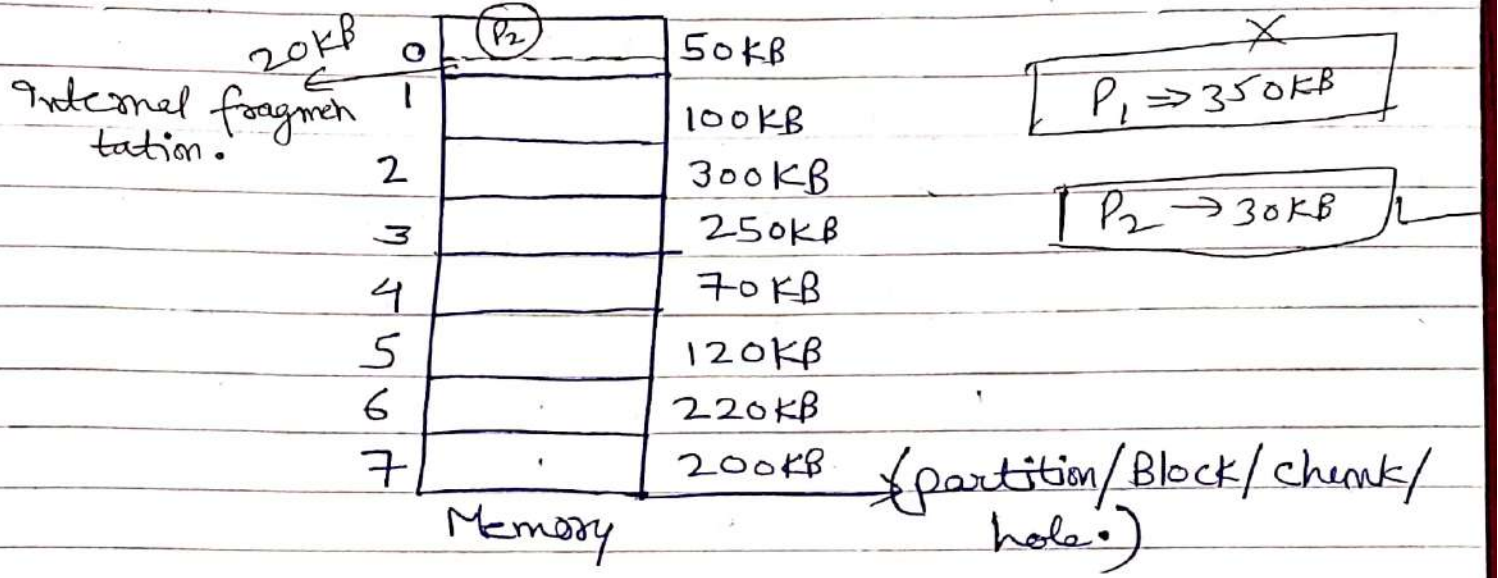
MEMORY MANAGEMENT TECHNIQUES:-

- Contigous
  - fixed partition scheme
  - Variable partition scheme
- Non Contigous
  - paging
  - Multilevel Paging
  - Inverted paging
  - Segmentation
  - Segmented paging

Memory Management Techniques

CONTIGUOUS MEMORY MANAGEMENT:-

1) FIXED PARTITION SCHEME:-

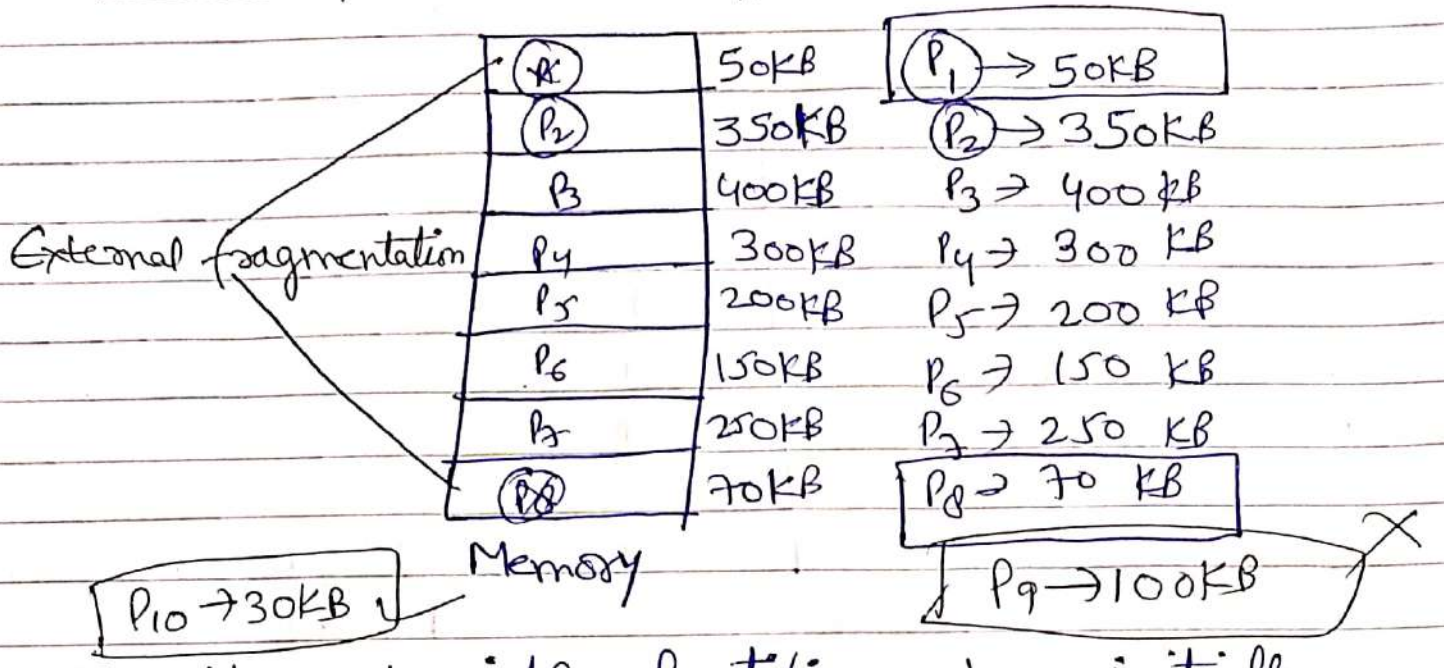


→ In the fixed partition scheme, memory will be divided into fixed number of partitions.
→ Fixed means no. of partitions are fixed in a memory.
→ In every partition only one process will be accomodated.
→ The degree of multiprogramming is restricted by no. of partitions in the Memory.
→ Maximum size of a process is restricted by maximum size of a partition.
→ Every partition is associated with the limit registers:-

Limit Registers

→ Lower Limit:- starting address of the partition
→ Upper Limit:- Ending Address of the position.

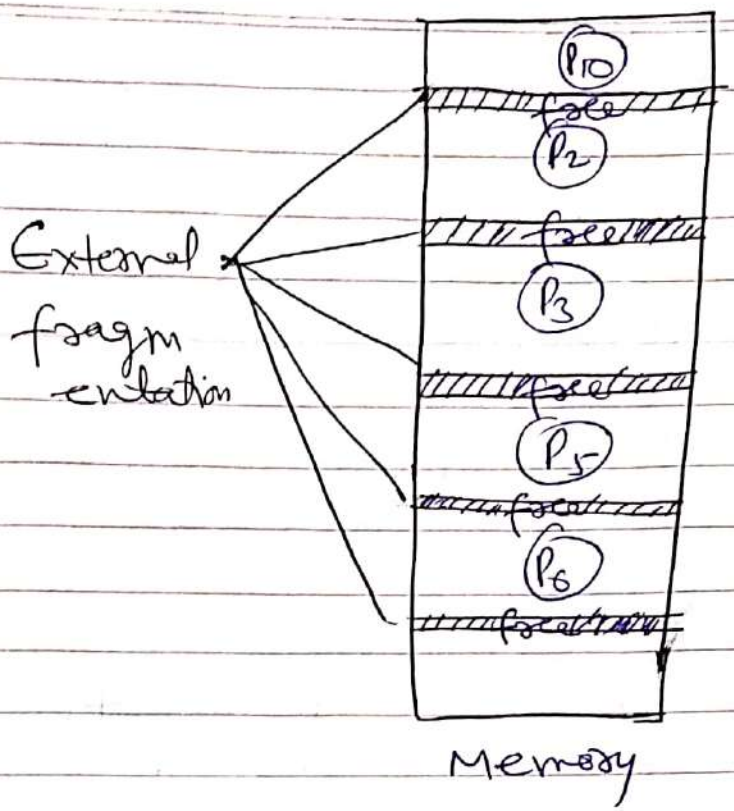
VARIABLE PARTITION SCHEME:-



→ In the Variable Partition scheme, initially memory will be single continuous free block.

→ Whenever the request by the process arrives accordingly partition will be made in the memory.

→ If the smaller processes keep on coming, then the larger partition will ~~remain~~ made into smaller partitions.



To avoid the problem of external fragmentation the following techniques are used.

1.) Compaction:- Moving all the processes towards the top or towards the bottom to make the free available memory in a single continuous place is called as compaction.
→ Compaction is undesirable to implement because it interrupts all the running processes in the memory.

2.) Implementing Non-contiguous memory management techniques.

PARTITION ALLOCATION METHODS :-

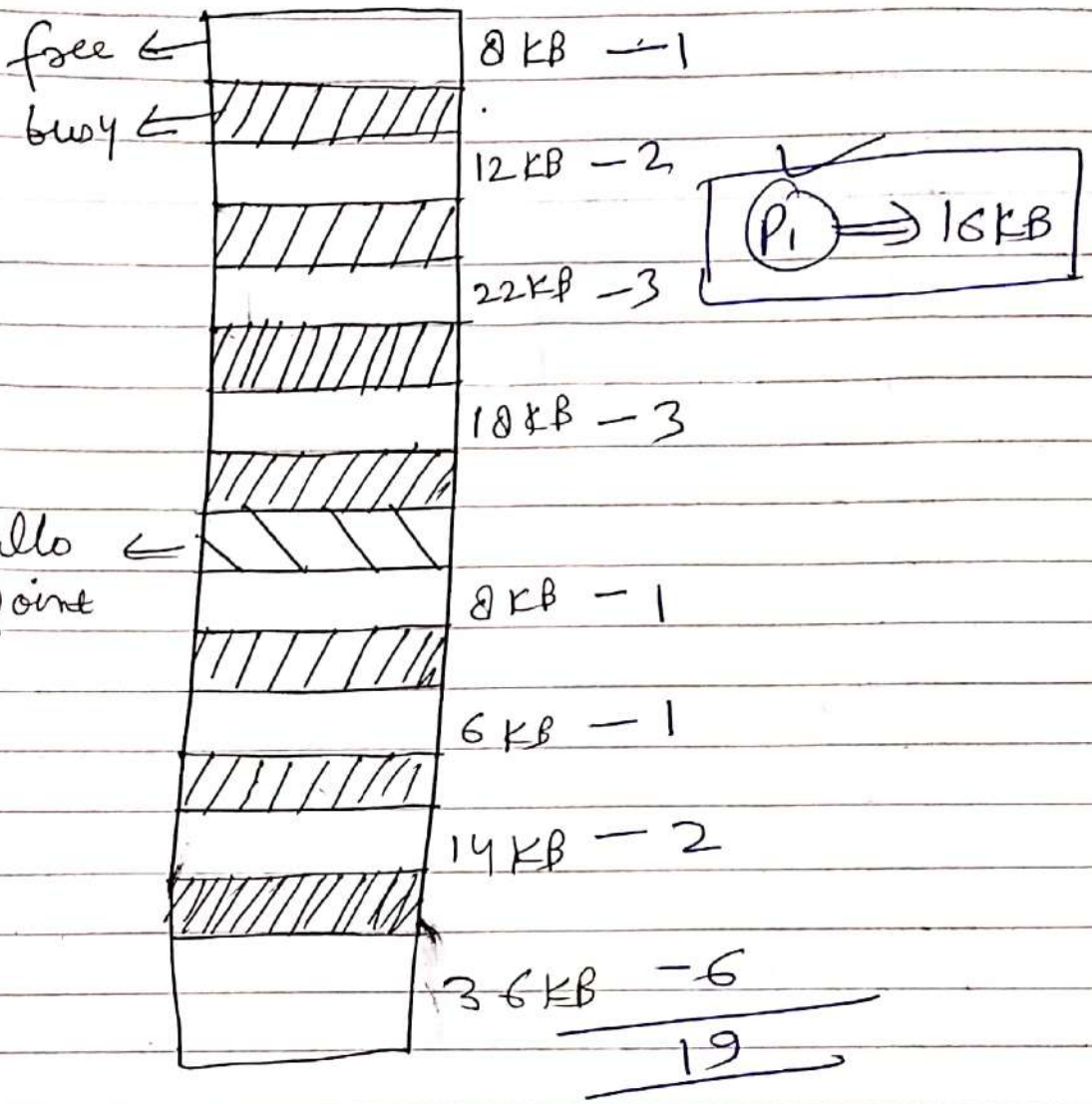
1.) **FIRST FIT :-** Allocate the process in a partition which is first sufficient partition from the top of the memory.

2.) **BEST FIT :-** Allocate the process in a partition which is smallest sufficient among the free available partitions.
→ to find out the smallest sufficient, it requires to search all the free partitions in the memory.

3.) **WORST FIT :-** Allocate the process in a partition, which is largest sufficient among the free available partitions.
→ to find out the largest sufficient, it requires to search all the free partition in the memory.

4.) **Next FIT :-** The next fit also works like first fit, but it will search for the first sufficient partition from "last allocation point".

Q.) Consider the following memory map,

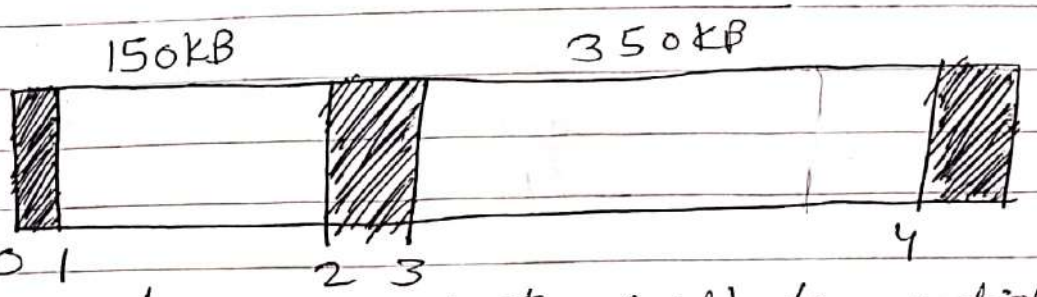


1.) First Fit: 22KB,
2.) Best Fit: 18KB,
3.) Worst Fit: 36KB,
4.) Next Fit: 36KB,

Q2.) How many successive Requests of 6KB will be satisfied by using First Fit assuming variable partition scheme?

a.) 17
b.) 18
✓ c.) 19 ✓
d.) 20
e.) 21

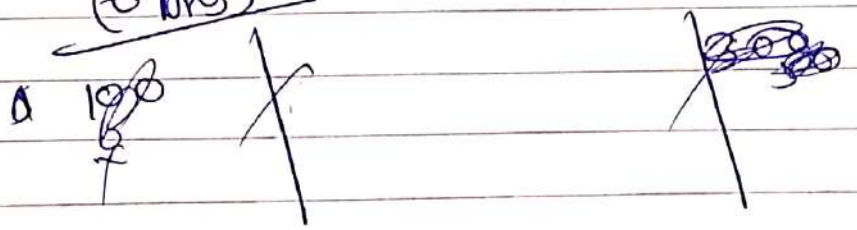
Consider the following memory map requests from the processes are 300KB, 25KB, 125KB 50KB respectively.



The above requests could be satisfied with ? (Assume variable partition scheme.)

a.) Best fit but Not first fit
b.) First fit but Not Best fit
c.) Both first fit and Best fit
d.) None of the above.

(b. Any.)



PAGING

NON CONTIGUOS MEMORY MANAGEMENT TECHNIQUES:-

PAGING:- It is using the terminology:-

1.) Logical Address space (L.A.S) or virtual address space (V.A.S.)
⇒ Represented in the form of words & Bytes
e.g., 512KW or 512KB.

2.) Logical Address (L.A.) or Virtual Address (V.A.)
⇒ Represented in the form of bits.
e.g. 19 bits.

3.) Physical Address space (P.A.S.)
⇒ Represented in the form of words or Byts.
Eg. 32MW or 32MB.

4.) Physical Addres (P.A.)
⇒ Represented in the form of bits.
Eg:- 25 bits.

Eg①: L.A = 39 bits

$L.AS = 2^{39} W$
$= 2^9 \times 2^{30} W$
$= \underline{512\ GW}.$

Eg.② LAS = 256MW

$L.A = 2^8 \times 2^{20}$
$= 28$ bits.

09/02/19

L.A > P.A

Eg 3.) P.A = 45 bits
↳ P.A.S = 2^{45} bits
$= 2^5 \times 2^{40}$ B
P.A.S = 32TB

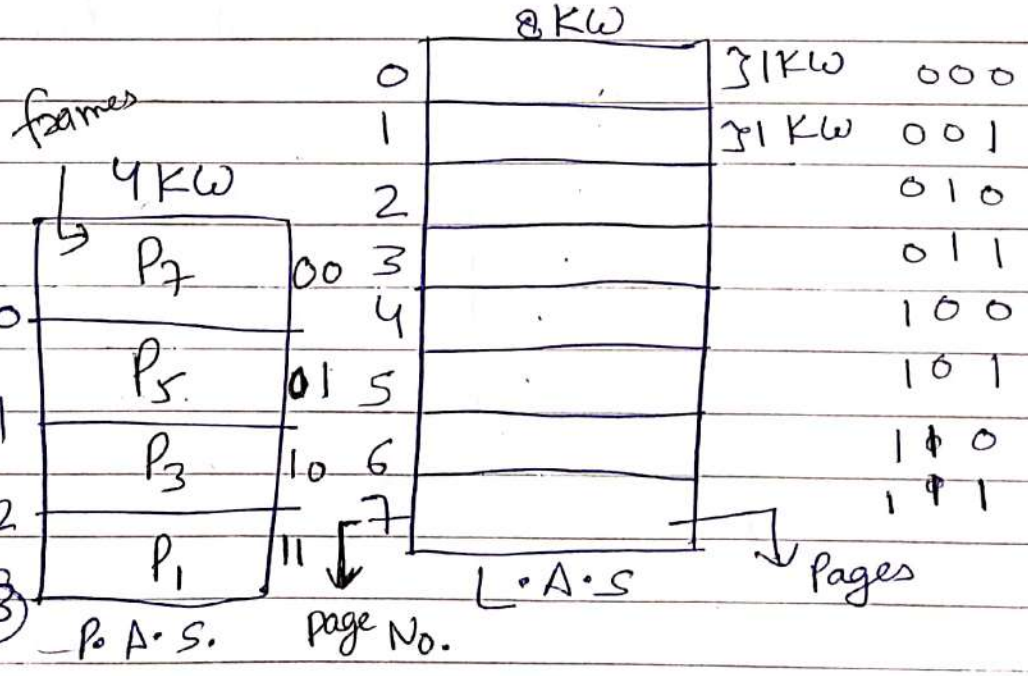
Eg 4.) ↳ P.A.S = 64KB
P.A = $2^6 \times 2^{10}$
P.A. = 16 bits.

→ The technique of mapping CPU generated logical address to physical address is called as Paging.

→ Paging is implemented in the hardware level.

Let us assuming
L.A. = 13 bits
P.A. = 12 bits

L.A.S = 2^{13}W = 8KW.
P.A.S = 2^{12}W = 4KW.



Assuming Page Size = 1KW

→ The logical address space will be divided into equal size pages.

$$\text{No. of pages} = \frac{\text{L.A.S.}}{\text{Page size}}$$

$$= \frac{8\text{KW}}{1\text{KW}} = 8.$$

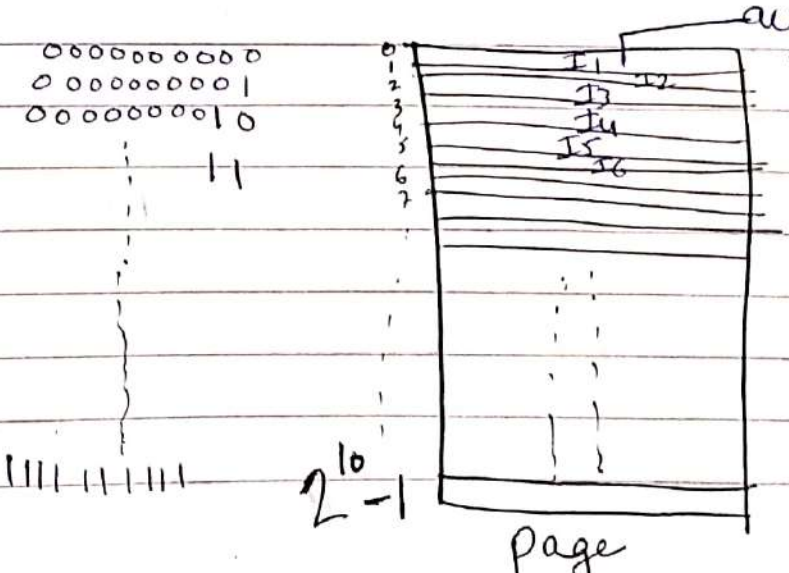
⇒ frames :- The physical address space is divided into equal size frames.

$$\text{Page Size} = \text{frame size}$$

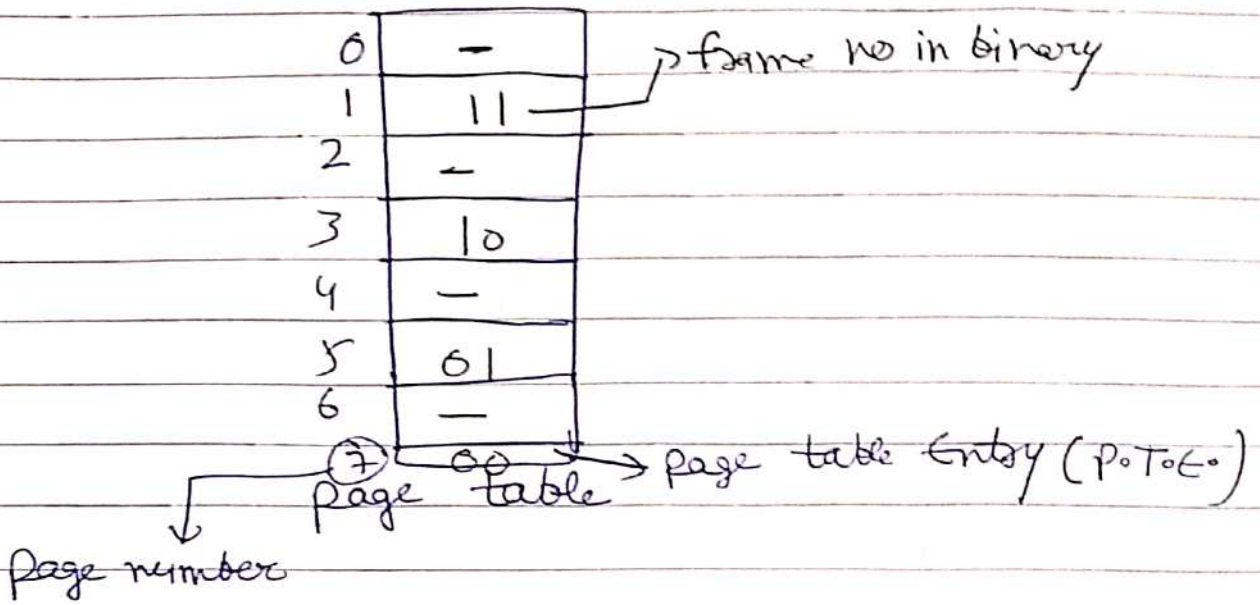
$$\text{No. of frames} = \frac{\text{P.A.S.}}{\text{frame size}} \quad = \frac{4\text{KW}}{1\text{KW}} = 4.$$

→ Some of the pages of L.A.S. will be brought into P.A.S.

$$\text{page size} = 1\text{KW}$$



→ Whenever the paging is applied, the page table has to be maintained.

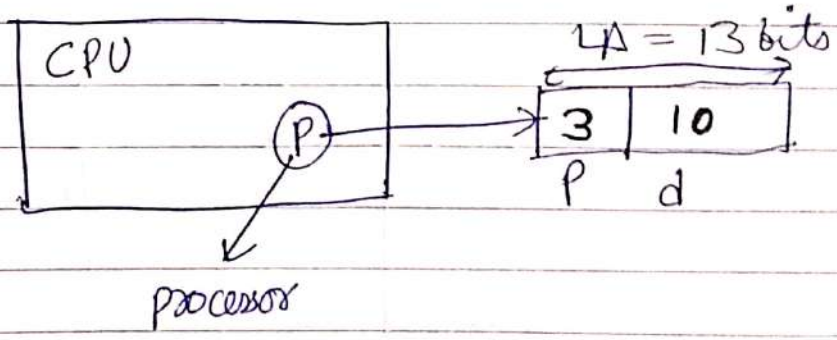


→ No. of entries in the page table is same as no of pages in L.A.S..

→ Page table entry definitely contains frame number.

→ Page table is also called as address Translation table.

→ frame number = Translation bits.



p = No. of bits required to represent the pages of LAS

or page number.

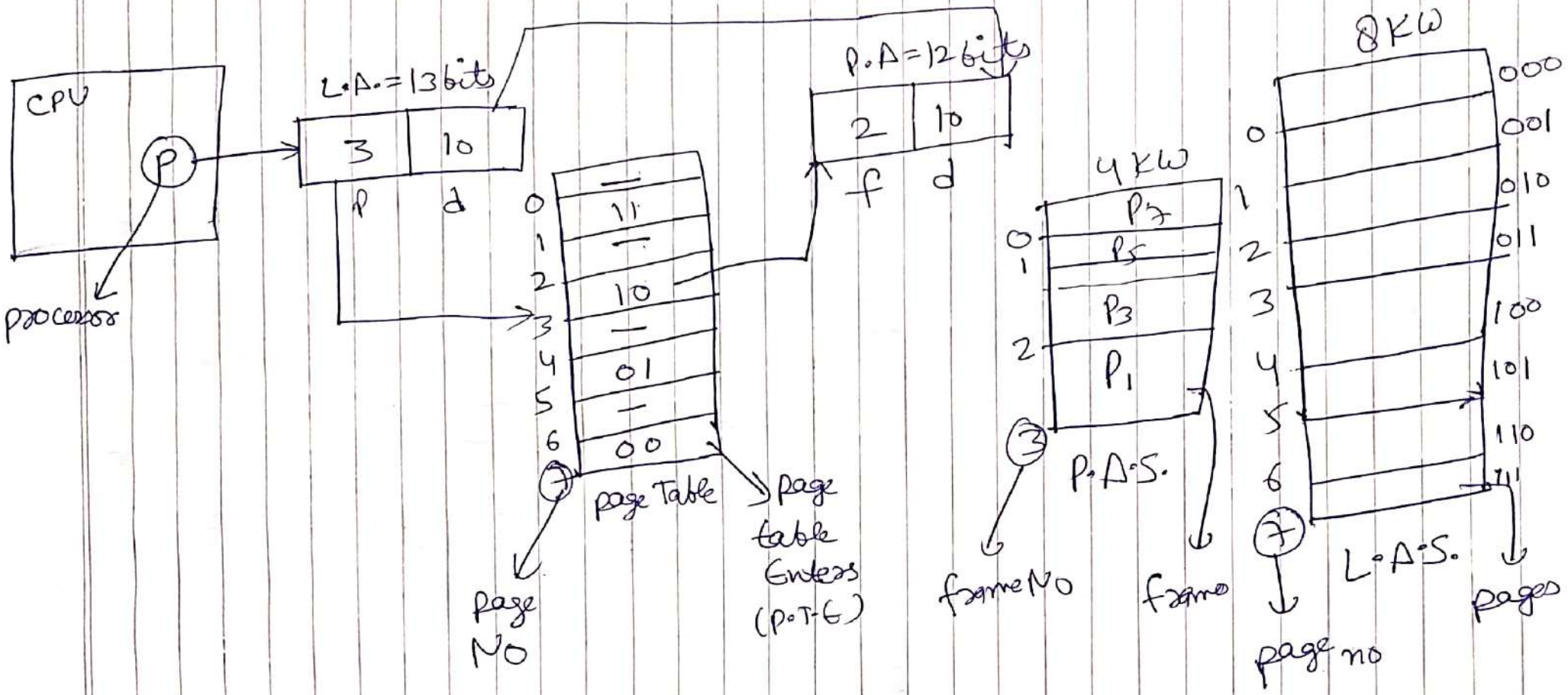
→ d = No. of bits required to represent word of a page. or
It is also called as page offset.

Physical address we assumed as a 12 bits.

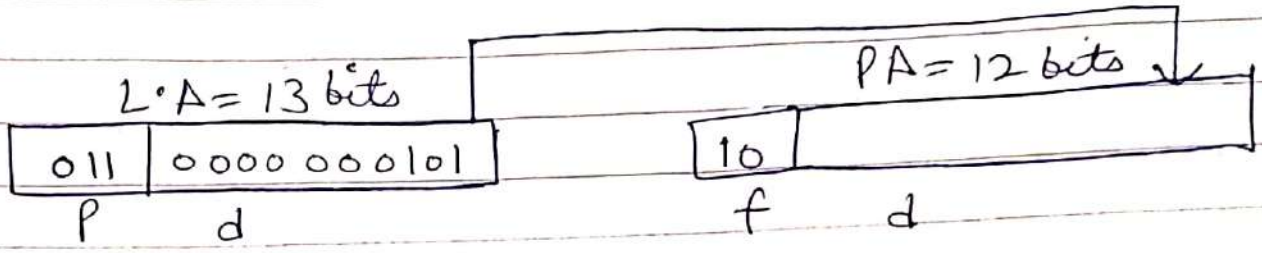
P.A = 12 bits

f = No. of bits required to represent frames of PAS
OR
frame No.

d = No. of bits required to represent word No. of a frame or frame offset.


CPU
processor
L.A.=13bits
3
10
p
d
P.A=12bits
2
10
f
d
page Table
page No
page table Entery (P.T.E)
4KW
P7
P5
P3
P1
P.A.S.
frame No
frame
8KW
000
001
010
011
100
101
110
111
L.A.S.
page no
page

Ex:-



Important points :-

1.) Whenever the process is created, paging will be applied on the process and page table will be created. And The base address of the page table will be stored in P.C.B.. (process Control Block).

2.) Paging is with respect to every process and every process will have its own page table.

3.) Page tables of the processes will be stored in the main memory.

4.) There is no external fragmentation in the paging.

5.) The internal fragmentation exists in the last page, and internal fragmentation in the paging is considered as a "$\frac{p}{2}$", where p is the page size.

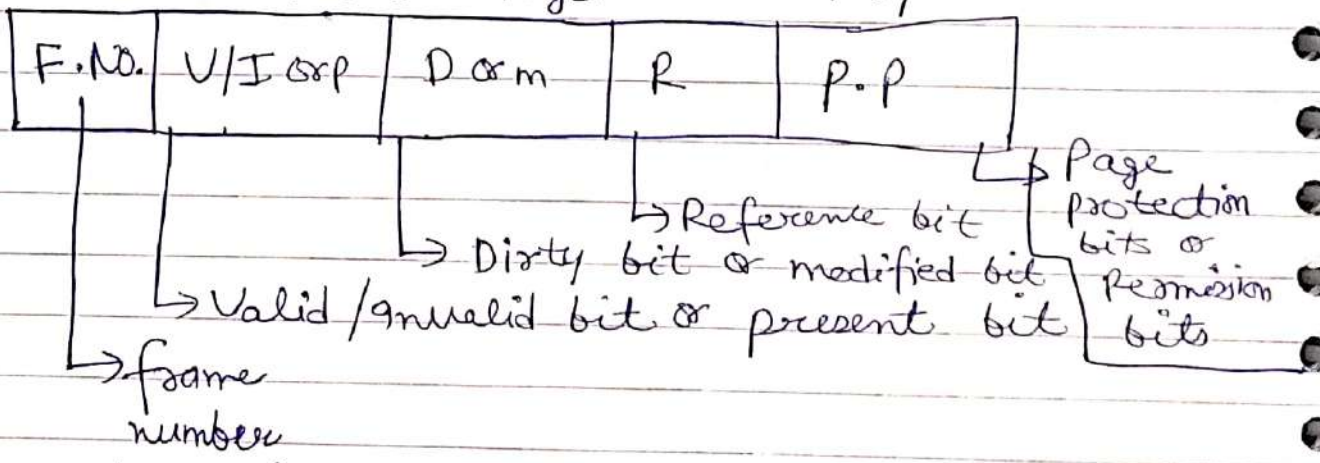
(5) Process size = 16 KW
Page size = 3 KW
No of pages = $\frac{16 KW}{3 KW} = 5.33 = 6$ (Internal fragmentation)

6.) Maintaining the page table is considered as "overhead" for the system.

NOTE:- Windows practically follows page size of 4 KB.

NOTE:- The Page table entry definitely contains frame Number, but sometimes along with the frame No, it may also contain additional bits like
1.) Valid / invalid bit or present bit
2.) Dirty bit / modified bit.
3.) Reference bit.
4.) Page protection bits or permission bits.

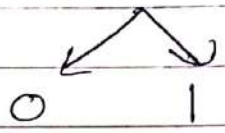
P.T.E = Page table Entry



frame number (or) is also called translation bits.

Valid/Invalid or present bit:-

These bits are used to identify whether the page is currently available or not available in the main memory.

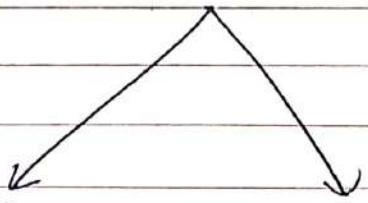


Not present in the main memory

present in the M.M.

Dirty Bit or modified bit:-

This bit is used to identify whether the page is modified or not modified by the processor while accessing the page as part of the execution.



Not modified

Modified.

Reference Bit:-

This bit is used to identify, how many times the page is referred by the processor while execution.

PAGE protection or permission bits :-

These bits are used for the protection and security from unauthorized access.

1MB = 1024 $2^{27} = 2^7 *$

91.) Consider a system which has logical address of 27 bits. And Physical address of 21 bits and page size is 4KW. Memory is word Addressable, then calculate No. of pages & No. of frames

Logical A.S. $= 2^{27}$

$= 2^{20} \times 2^7$

$= 128$ MB

Physical A.S $= 2^{21}$

$= 2^{20} \times 2^1$

$= 2$ MB

No. of pages $= \frac{128 \text{ MB}}{4\text{KW}}$

$= \frac{128 \times 2^{10}}{4\text{KW}}$

$= \frac{128 \times 2^{10}}{4}$

$= 32 \times 2^{10}$

$= \boxed{2^{15}} = 32\text{K}$

No of frames $= \frac{2 \times 2^{10}}{2^2}$

$= 2^{11-2}$

$= \boxed{2^9} = 512$

Q.) Consider a system which have No of pages as 8K, Page size is 16 KB. Memory is byte Addressable. Physical Address is 22 bits, then calculate logical address & No. of frames?

Ans.)

Physical A.S $= 2^{22}$

Page size $= 2^4 \times 2^{10}$

Logical Address Space = page size × No of pages

$= 16\,KB \times 8K$

$= 2^{13} \times 2^{14}B = 2^{27}B.$

LA = 27 bits.

No. of frames $= \frac{2^{22}\ byte}{2^{14}\ byte} = 2^8$

$= 256$

q.) Consider a system which has

LAS = 128 MW,
PA = 24 bits

Memory is word Addressable. The PAS is divided into 8K frames. Then what is the page size & how many pages in the LAS?

PAS = 2^{24} words

=

$\frac{\text{No. of frames}}{2^{24}}$ = frame size

$\Rightarrow \frac{2^{24}}{2^{13}}$ = frame size

$= \frac{2KW \times 2^{3} \times 1000}{2^{24}}$ = frame size.

$= 2^{11}$

=

No. of pages = $\frac{2^{7} \times 2^{20}}{2^{11}} = 2^{16}$ ✓

$= 64K.$

Worktext P-10 (14)

32 16 8 4 2 1
1 1 1 1 (29)

Q.) Consider a system which logical address of 32 bits and PAS = 64MB. Memory is byte addressable & Page size is 4 KB. Then what is the approximate size of page table in bytes.

a.) 1 MB
b.) 2 MB
c.) 4 MB
d.) 8 MB

~~LAS = 32 bits~~
~~= 5 bits.~~
~~PAS = 64MB~~
~~= 2^{26} bytes.~~

No. of frames
$= \frac{PAS}{\text{frame Size}}$
$= \frac{64MB}{4KB}$
$= \frac{2^{26}}{2^{12}} = 2^{14}$

14 bits are required

Page table size = No. of Entries in the PT * Page table Entry size.

$= \left(\frac{2^{32}}{2^{12}}\right) \times$ F.NO

$= 2^{20} \times 14$ bits
$= 2^{20} \times 2$ bytes
$\cong$ 2MB.

29 = 4 24

Q.) Consider a system which have page table with 4 K entries & logical address is 29 bits then what is the physical address, if the system has 512 frames.

4K entries = $2^2 \times 2^{10}$

No of pages = 2^{12}.

~~Page size = LAS / No of pages~~

$= \frac{2^{29}}{2^{12}}$

L.A = 29 bits

12	17
P	d

P.A = 26 bits

$512 = 2^9$

$= 9$ bits

9	17
f	d

Workbook

P-107

Q40)

L.A.S = 2^{16} bytes.

P.A.S = 2^{16} bytes.

Page size = 2^9 bytes.

Page T.E. size = 2 bytes. = 16 bits

Page Table size = ? (+ ~~buffer~~ + dirty + valid)

No. of frames = $\left(\frac{2^{16}}{2^9}\right)$ ~~× 2 bytes~~

= 2^7 ~~× 2~~

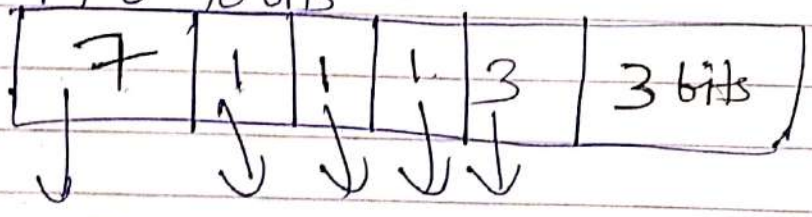
No of frames ~~Page table size~~ = ~~2^8 bytes~~ 2^7

= ~~$2^8 \times 2^3$ bits~~

= ~~2^{11} bits~~

= 2KB.

P.T.E = 16 bits



F.NO.

3bits

GATE-2016 (2 Marks)

(P.T.E.S = LAS / PS)

2015

Q.) Consider a system
L.A = 40 bits.
Page size = 16 Kb.
Memory is byte addressable. The page table entry requires 48 bits. Then what is the page table size ——— in Megabytes.

solve:) P.T.E. 48 bits.

Page Table size = No of frames × Page table Entry size.

$$= 48 \text{ bits} \times \frac{2^{40} \text{ bytes}}{16 \text{ Kb}}$$

$$= 6 \text{ bytes} \times \frac{2^{40}}{16 \times 2^{10} \times 8}$$

$$= \frac{2^{40} \times 6}{2^{10} \times 2^{4} \times 8}$$

$$= \frac{2^{40-14} \times 6}{}$$

$$= 6 \times 2^{26}.$$

$$= 6 \times 64 \times MB$$
$$= 384 \text{ MB}.$$

Q.) Consider a system which has

LAS = PAS = 'S' bytes.

& Page size is 'P' bytes.

Page T.E. Size = 'e' bytes.

then what is the optimal value of page size by minimising overhead of maintaining Page table size & internal fragmentation in the paging?

a.) $P = \sqrt{2Se^2}$

b.) $P = \sqrt{2S^2e}$

c.) $P = \sqrt{2Se}$

d.) $P = \sqrt{2(Se)^2}$

~~page size = PAS / No of frames~~

$= \frac{S \text{ bytes}}{S}$

~~No of frames = PAS / Page size = S / P~~

~~P = S / No of frames~~

Memory overhead = P.T. size + I.f. in paging.

$$= \text{No. of entries} \times \text{entry size} +$$

$$= \left(\frac{s}{p}\right) \times e + \frac{p}{2}$$

$$= \frac{d}{dp}\left(\left(\frac{s}{p}\right) \times e + \frac{p}{2}\right) = 0$$

$$\Rightarrow \left(\frac{-s}{p^2} \times e + \frac{1}{2}\right) = 0$$

$$\Rightarrow p^2 = 2se$$

$$\boxed{p = \sqrt{2se}}$$

PERFORMANCE OF PAGING:-

The Main memory access time = 'M'
The page tables are stored in the main memory.
Then the formula for effective memory access time

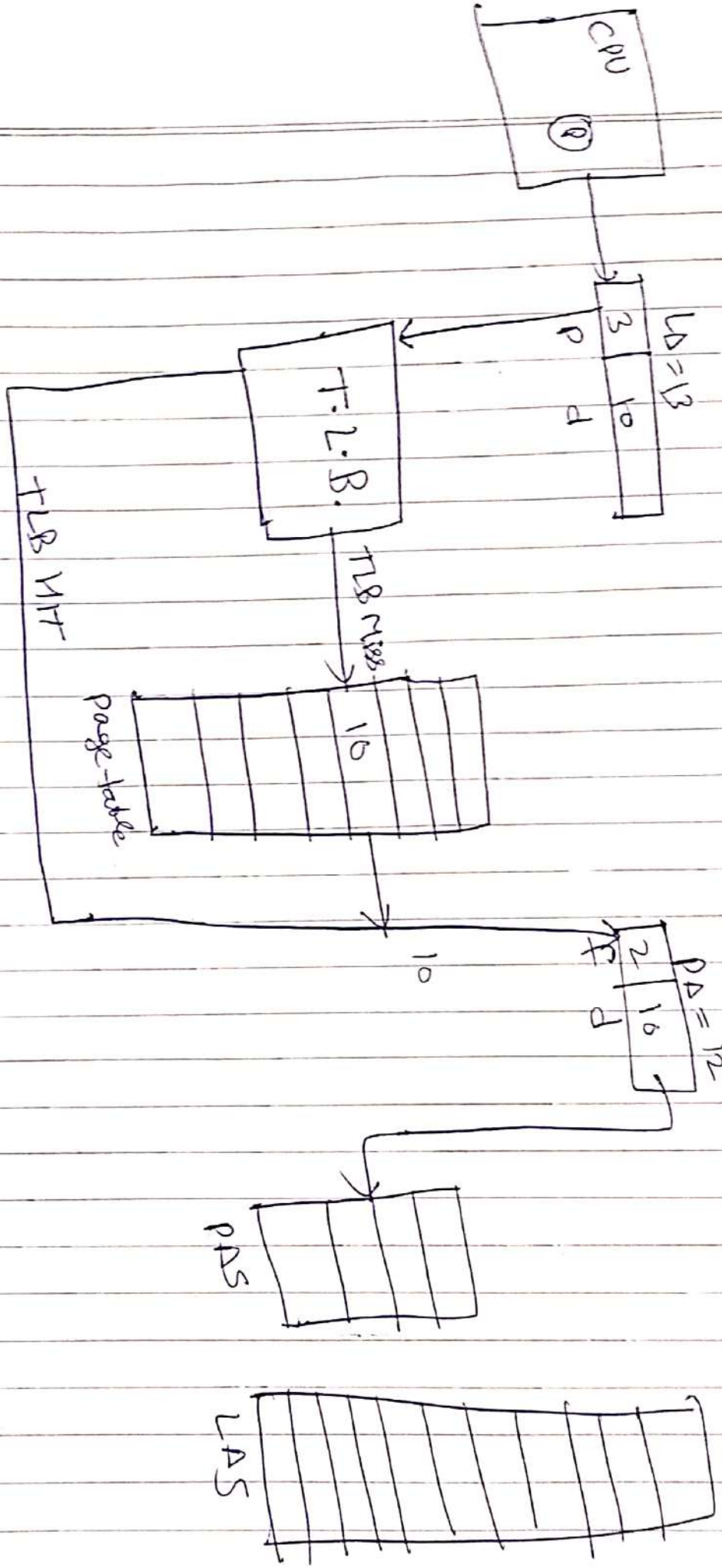
$$M \cdot M \cdot A \cdot T = 'M'$$

$$\boxed{E \cdot M \cdot A \cdot T = 2M}$$

1 M for
1 M for Actual time.

The T.L.B. (translation look aside buffer) is added to improve the performance of Paging.

→ TLB is a hardware device implemented using associative registers.
→ TLB access time will be very less compare to main memory access time.
→ TLB contains frequently refered page nos & the corresponding frame numbers.


CPU
LA=13
3
P
10
d
T.L.B.
TLB Miss
TLB Hit
page-table
10
10
PA=12
2
10
d
PAS
LAS

TLB access time = 'c'.
TLB hit ratio = 'x'

then the formula effective memory access time.

$$E.M.A.T = x(c+m) + (1-x)(c+2m)$$

Q.) Consider a system which has m.m.a.t of 100ns. TLB access time is 20ns, TLB hit ratio is 95%. Then what is effective memory a.t with TLB & without TLB?

Ans:)

mmat = 100ns.
TLB access time c = 20ns.
x = 95% = 0.95

with TLB

$= 0.95(20+100) + (1-0.95)(20+200)$
$= 0.95(120) + (0.05)(220)$
$= 125\ ns.$

without TLB

$EMAT = 2M$
$= 2 \times 100ns$
$= 200ns.$

Q:) How much hit ratio is required to reduce effective M.A.T from 300ns without TLB to 250 ns with TLB? The TLB access time is 60 ns.

Ans.)

$$EMAT = 2M$$

$$2M = 300$$

$$M = 150$$

$$E.M.A.T = x(C+m) + (1-x)(C+2m)$$

$$250 = x(60+150) + (1-x)(60+2\times150)$$

$$250 = x(210) + (1-x)(60+300)$$

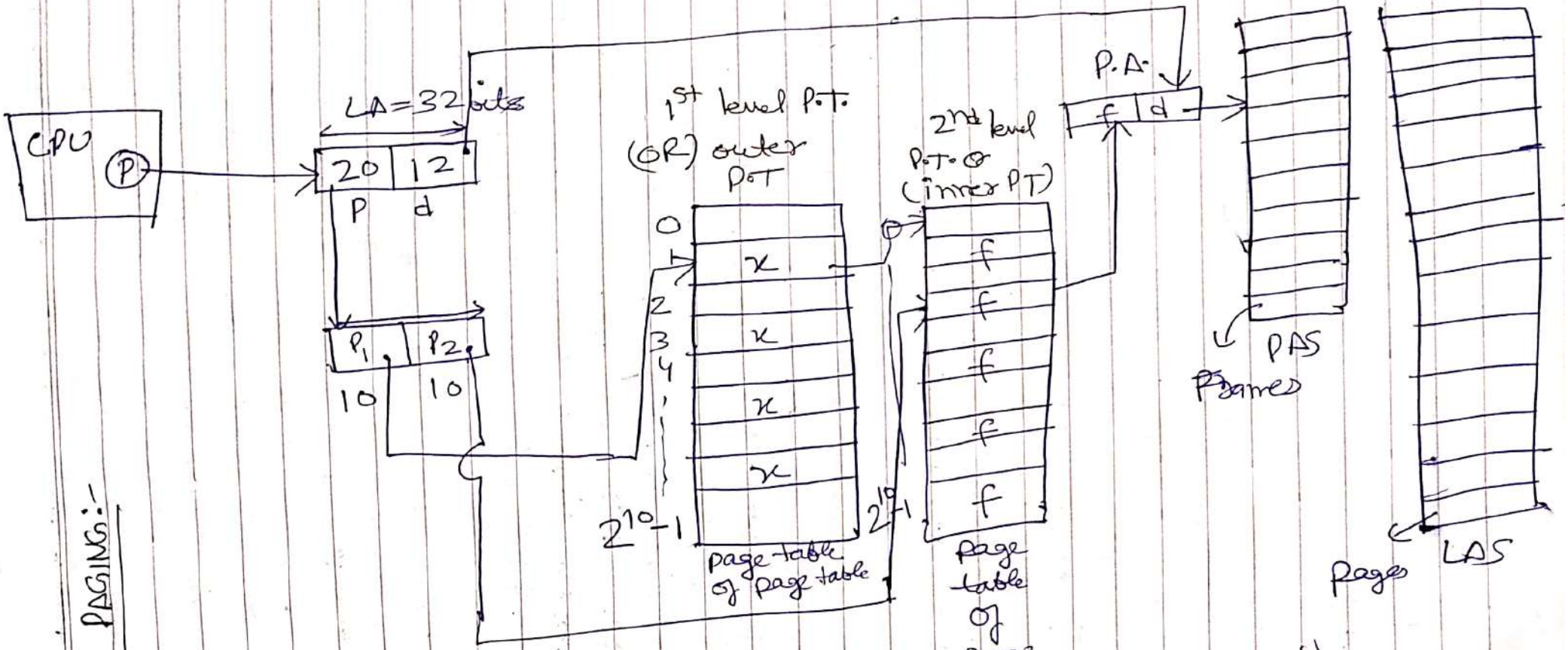
$$x = 0.7333$$

Q:) Consider a (1MB) system which has logical address of 32 bits and Page size of LAS = 4KW and memory is word addressable and page table entry size is one word. Then what is the Page table size?

Solve:)

x = Base Address of Page of P.T.

Page size of P.T = 1KW



MULTI LEVEL PAGING:-

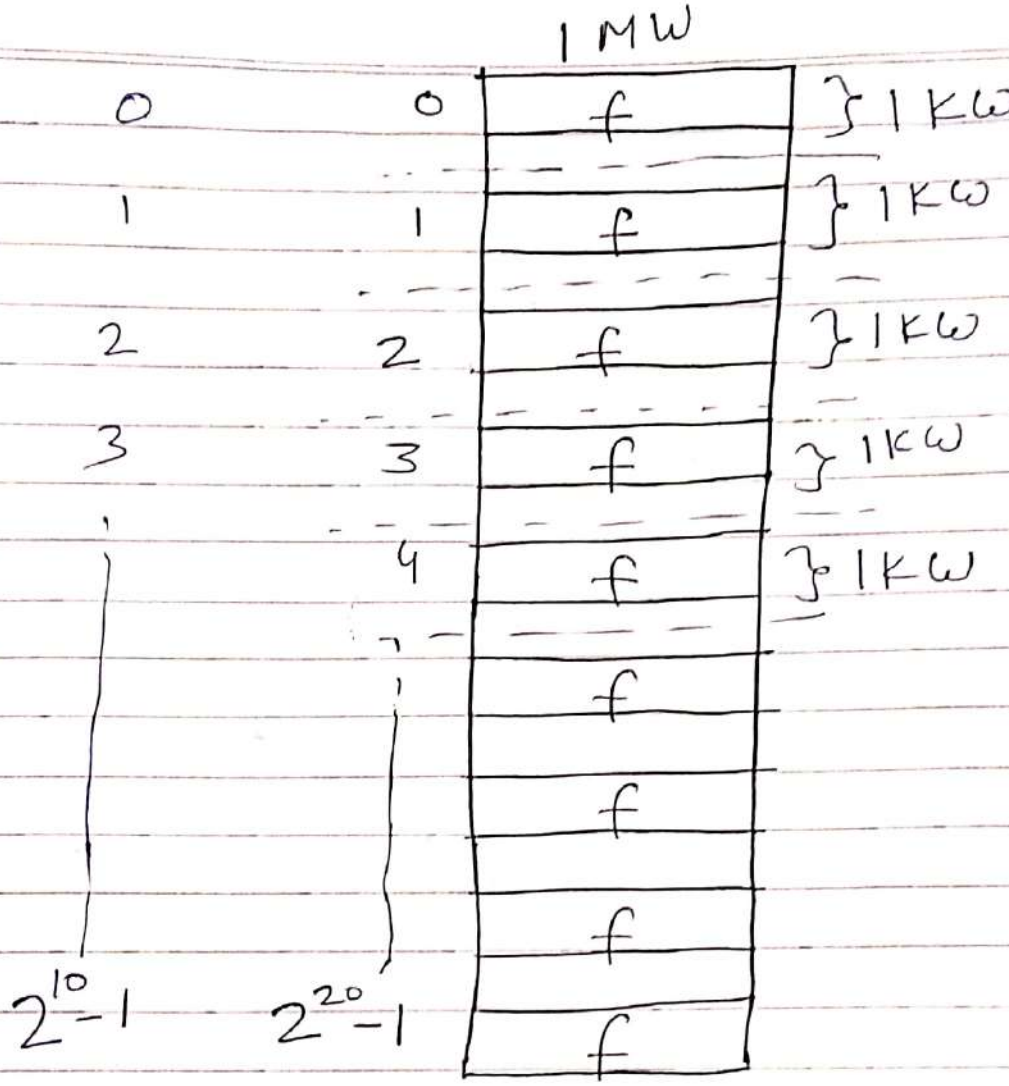
L.A = 32 bits

Page size of LAS = 4KW

PTE Size = 1W

PT Size = $\frac{2^{32}}{2^{12}} \times 1W = 2^{20}W = 1MW$

No of bits in x = No of bits in y = No of bits in f



→ To avoid the overhead of bringing large size page table into memory, the concept of multilevel paging will be implemented.

→ In the multilevel paging, paging will be applied on the page table and instead of bringing the entire page table into memory the pages of page table will be brought into memory.

→ P_1 is no of bits required to represent pages of page table or page no. of page table.

No. of pages on P.T. $= \frac{2^{20}W}{2^{10}W} = 2^{10} = 1K.$

→ P_2 = No. of bits required to represent word no. of page of page table. or (page offset of page table).

Q.) Consider a system which has 2 level paging applicable. The Page table has divided into 2K pages and each page have 4K entries. The memory is word addressable.
The PAS = 64 MW, which is divided into 8 K frames. Page table entry size in both the levels is 2 words. Then Calculate

i.) Length of logical address
ii.) Length of physicall address
iii.) Ist level page table size
iv.) IInd level Page table size (Page of page table).

P.T. size =
No of pages = 2K

1st level pages = 4K entries.
no of pages

Page table Entry size = 2W

Frame size = PAS = $\frac{2^6 \times 2^{20} W}{2^3 \times 2^{10}} = \frac{2^3 \times 2^{10}}{2^{13} W} = 8KW.$

(i.) LAS = 2K
= 2^{14}/w
LA = 14

(ii.) PAS = 64MW
= 2^{26} W
PA = 26

(iii.) = 4K × 2W = 2K ×

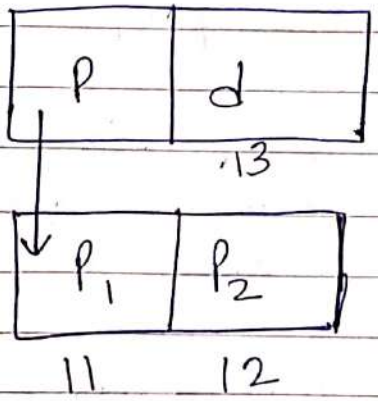
= [illegible]

= [illegible]

(iv) = 4K × 2W

= 2^{13}

(i) LA = 11+12+13 = 36



(ii) PAS = 64MW

= 2^{26} W

PA = 26W.

(iii.) = [illegible] × [illegible]

1st level P.T. Size = 2^{P_1} × PTE Size

= 2^{11} × 2W

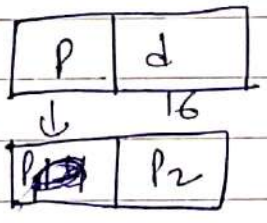
= 2^{12}W

= 4KW

4.) 2^{nd} level P.T. size = $2^{P_2} \times$ P.T.E. size
(Page of P.T)
$= 2^{12} \times 2w$
$= 2^{13} w = 8Kw$.

Q.) Consider a system which has 2 level paging applicable. The page table has divided into 8K pages. And each page having 16K entries. Memory is byte addressable. The PAS is 256 MB which is divided into 64KB frames. The page table entry size in both the levels is 32 bits then calculate.

1.) Length of logical address
2.) Length of physical address
3.) I^{st} level page table size
4.) II^{nd} level page table size
(page of page table).



2^{14}

frame size = 64KB
$= 2^{16}$

LA = 13+14+16
= 43

2.) PAS = 256 MB
$PAS = 2^{28}$
PA = 28.

3.) 1st level Page Size = 2^{P_1} x PTESize

$= 2^{13} \times 32$ bits

$= 2^{13} \times 2^2$

$= 2^{15}$ ω

= 32KB

4.) IInd level Page Size = 2^{P_2} x PTESize

$= 2^{14} \times 2^2$

$= 2^{16}$ ω

= 64KB.

PERFORMANCE OF TWO LEVEL PAGING :-

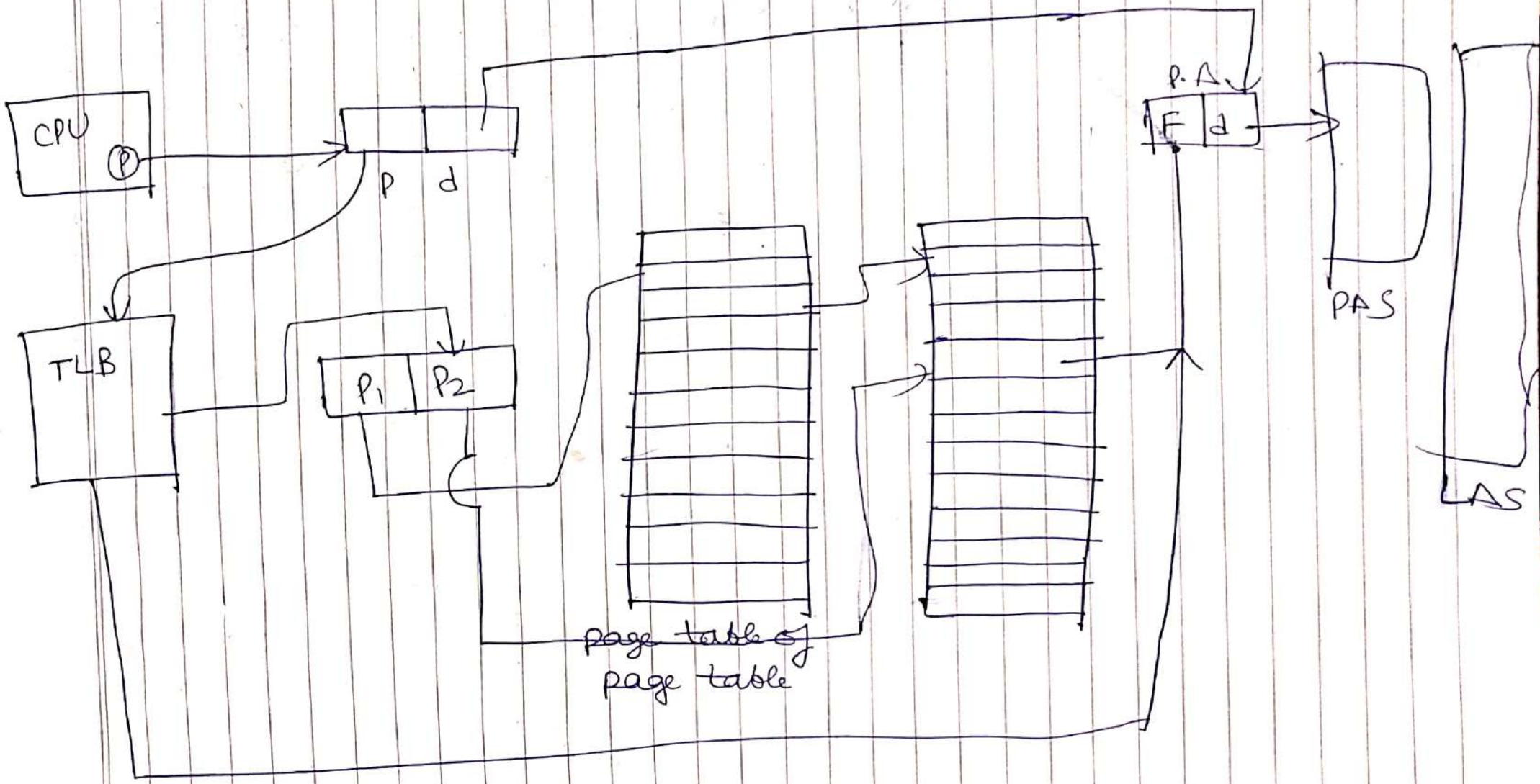
MMAT = 'M'

The page tables are stored in the main memory, then the formula for

$$E.M.A.T = 3M$$

→ The TLB is added to improve the performance of Paging.

→ The TLB contains frequently referred page numbers and corresponding frame numbers.


CPU
P
P d
P.A
F d
TLB
P1 P2
page table of page table
PAS
LAS

→ TLB ACCESS TIME = 'C'
→ TLB HIT RATIO = 'x'.

Then the formula for effective M.A.T

$$EMAT = x(C+m) + (1-x)(C+3m)$$

n level paging

$$EMAT = x(C+m) + (1-x)(C+(n-1)*m)$$

IMPORTANT POINTS:-

→ In the multilevel paging, when the paging applied on the page tables, then the last page table, which we get is called as first level page table (I^st Level).

→ In a multilevel paging, when the paging applied on the page tables, then I^st level page table entry contains base address of II^nd level page table, II^nd level page table entry contains base address of 3^rd level page table and so on. The final Page table entry contains frame no of a actual page.

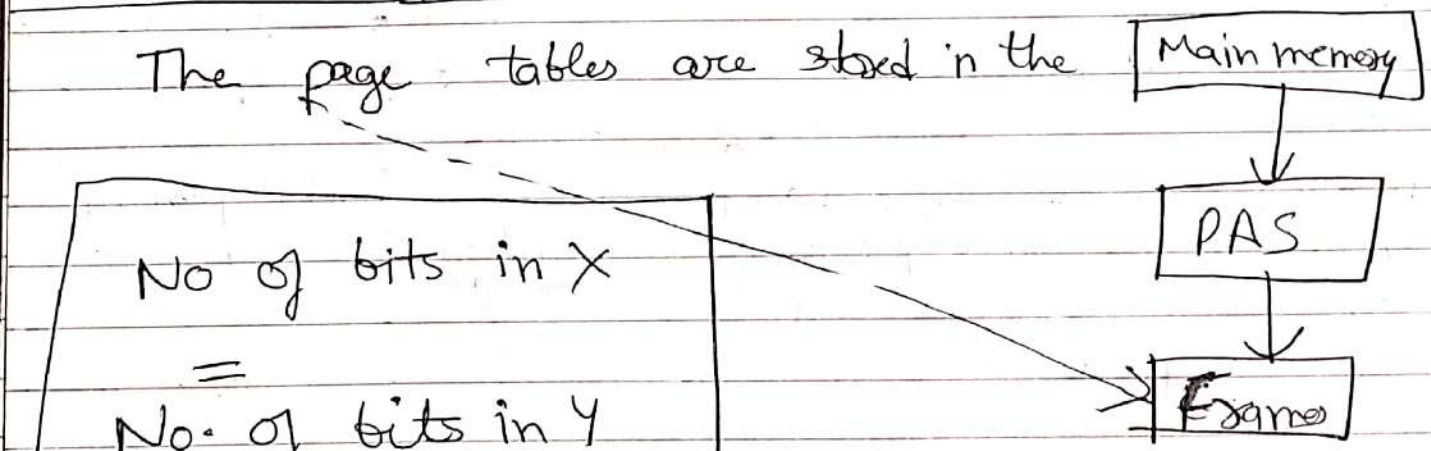
No of bits in X = No of bits in Y = No of bits in F

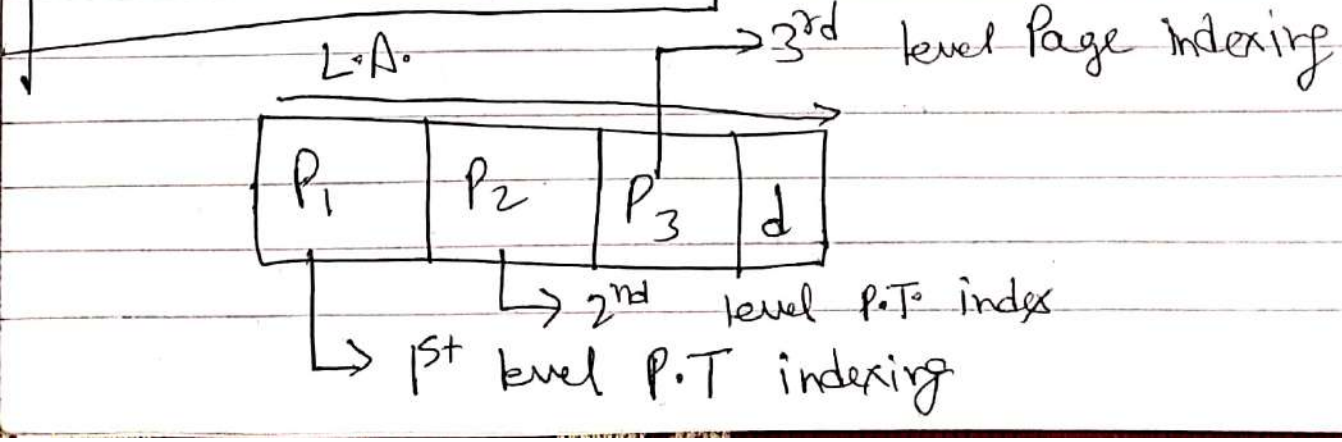
→ In the multilevel paging, when the paging applied on the page table, then whatever may be the levels of paging, all the page tables "Page of Page table" will be stored in a main memory.

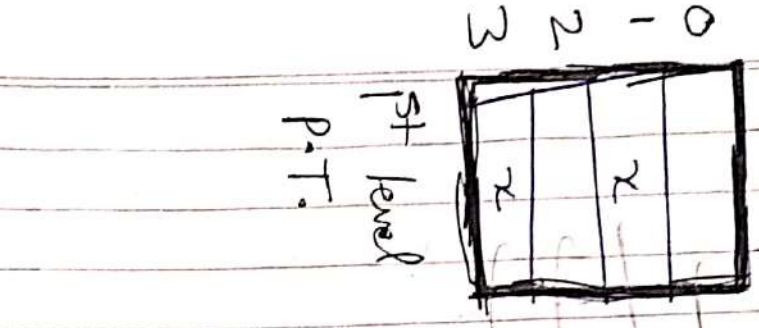
→ In the multilevel paging, when the paging applied on the page tables, then whatever may be the levels of paging. All the page table entries contains frame no.

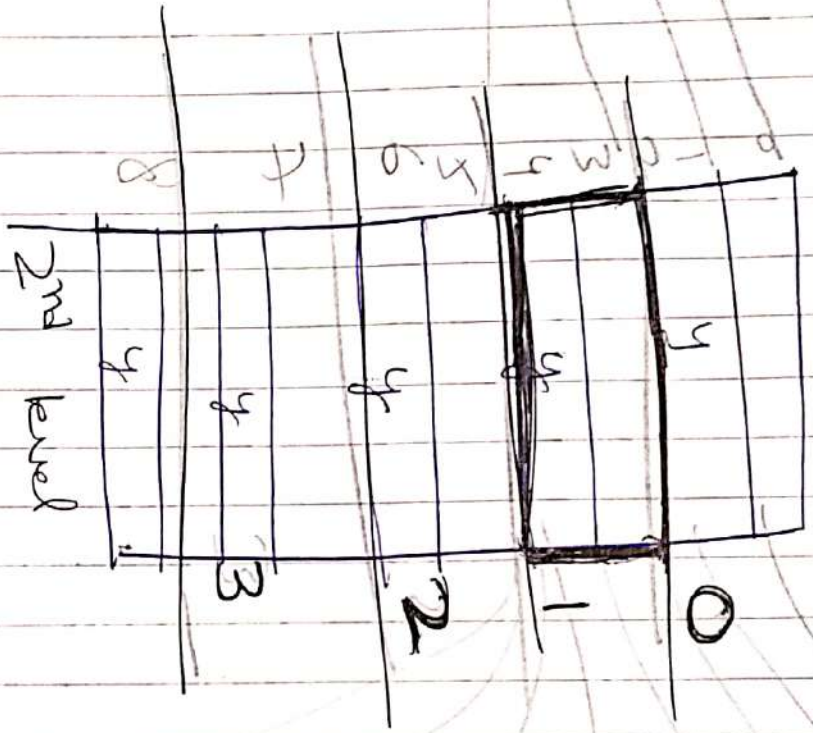
→ If the page size is not mentioned in the problem then generally page size will be same in all the places (levels).

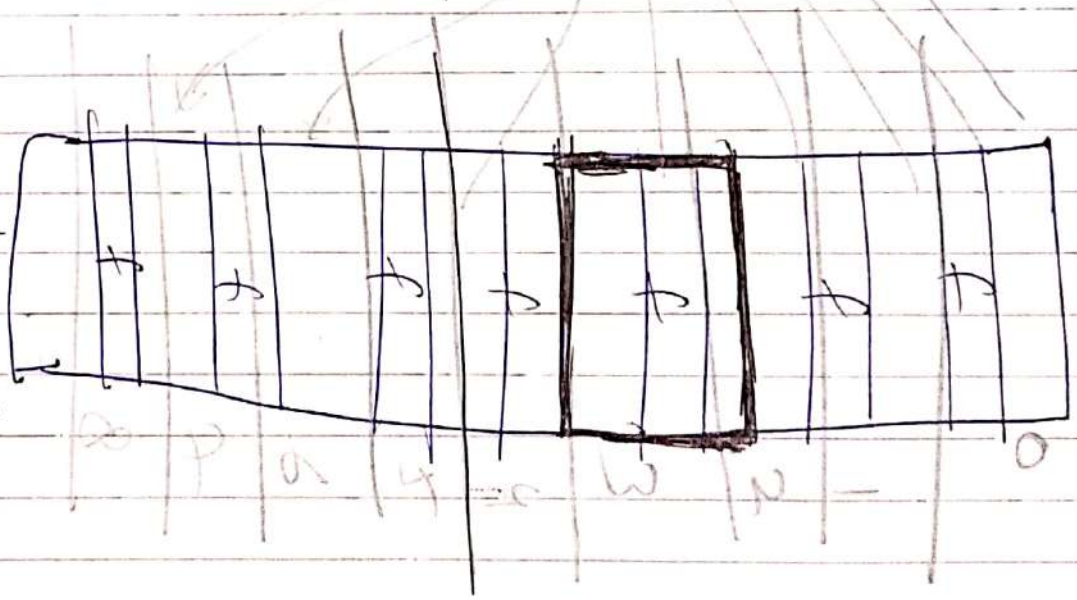
The page tables are stored in the Main memory






3
2
1
0
x
x
1st level
P.T.


y
y
y
y
y
2nd level
P.T.
3
2
1
0


f
f
f
f
f
f
f
3rd level
P.T.
0

P-106
Workbook

(22)

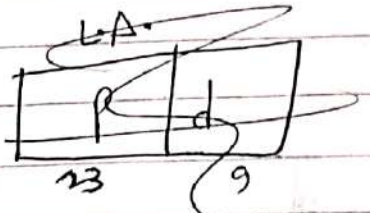
$32 - 9 = 23$

$2^{10} = 1024$

Q25.) (a) Logical address = 32 bit
= ~~4 bytes~~.

~~No. of~~ pages size = 1024 bytes. 2^9

Page table entry = 4 bytes.



~~1024 = 4 bytes × No. of Entries~~

~~$\frac{1024}{4}$ = No of Entries~~

~~256 = No. of Entries.~~

(b)=

P.T. size ≤ Page size

1st time paging :-

$$\text{P.T size} = \left(\frac{2^{32}}{2^{10}}\right) \times 4B$$

$$= 2^{22} \times 2^2 B$$

$$= 2^{24} B$$

$$= 16 MB$$

II^nd time paging :-

$$\text{P.T. size} = \left(\frac{16MB}{1KB}\right) \times 4B$$

$$= \frac{2^{24}}{2^{10}} \times 2^2 B$$

$$= 2^{16} B$$

$$= 64KB.$$

III^rd time paging :-

$$\text{P.T. size} = \frac{64\,\cancel{KB}}{1\,\cancel{KB}} \times 4B$$

$$= 64 \times 4B$$

$$= 2^8 B$$

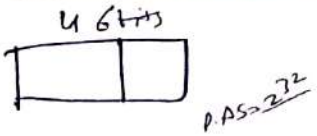
$$= 256 \text{ Bytes.}$$ (9^th level P.T. size)

$16MB \leq 1KB$	✗
$64KB \leq 1KB$	✗
$256B \leq 1KB$	✓

(C.) is right answer.

2 marks

Gate 2013



Q1.) Consider a system which has logical address of 46 bits, P.A = 32 bits.
Memory is byte addressable.
The Page table entry size is 32 bits.
The OS uses 3 level paging for logical to physical address translation and 1st level page table size is exactly same as page size. What is the page size?

a.) 2 Kb.
b.) 4 Kb.
c.) 8 Kb. ✓
d.) 16 Kb.

Solve:)

L.a = 46 bits

Page table entry size = 32 bits.

No of entries in the PT =

1st time paging :- Assuming page size x

$$P.T.\ size = \frac{2^{46}B}{x\ Bytes} \times 32\ bits$$

$$= \frac{2^{46} \times 4\ bytes}{x\ bytes} = \frac{2^{48}}{x} B.$$

2nd time paging :-

$$= \frac{2^{48}/x}{x\ bytes} \times 4$$

$$= \frac{2^{48}}{x^2} \times 2^2 = \frac{2^{50}}{x^2}$$

3rd time paging :-

$$\Rightarrow \frac{\frac{2^{50}}{x^2}}{x} \times 4 \text{ bytes}$$

$$\boxed{\text{P.T. size} = \frac{2^{52}}{x^3}} \leftarrow \text{9st level Page Table size.}$$

1st level page table size = Page size

$$\frac{2^{52}}{x^3} B = x B$$

$$x^4 = 252$$

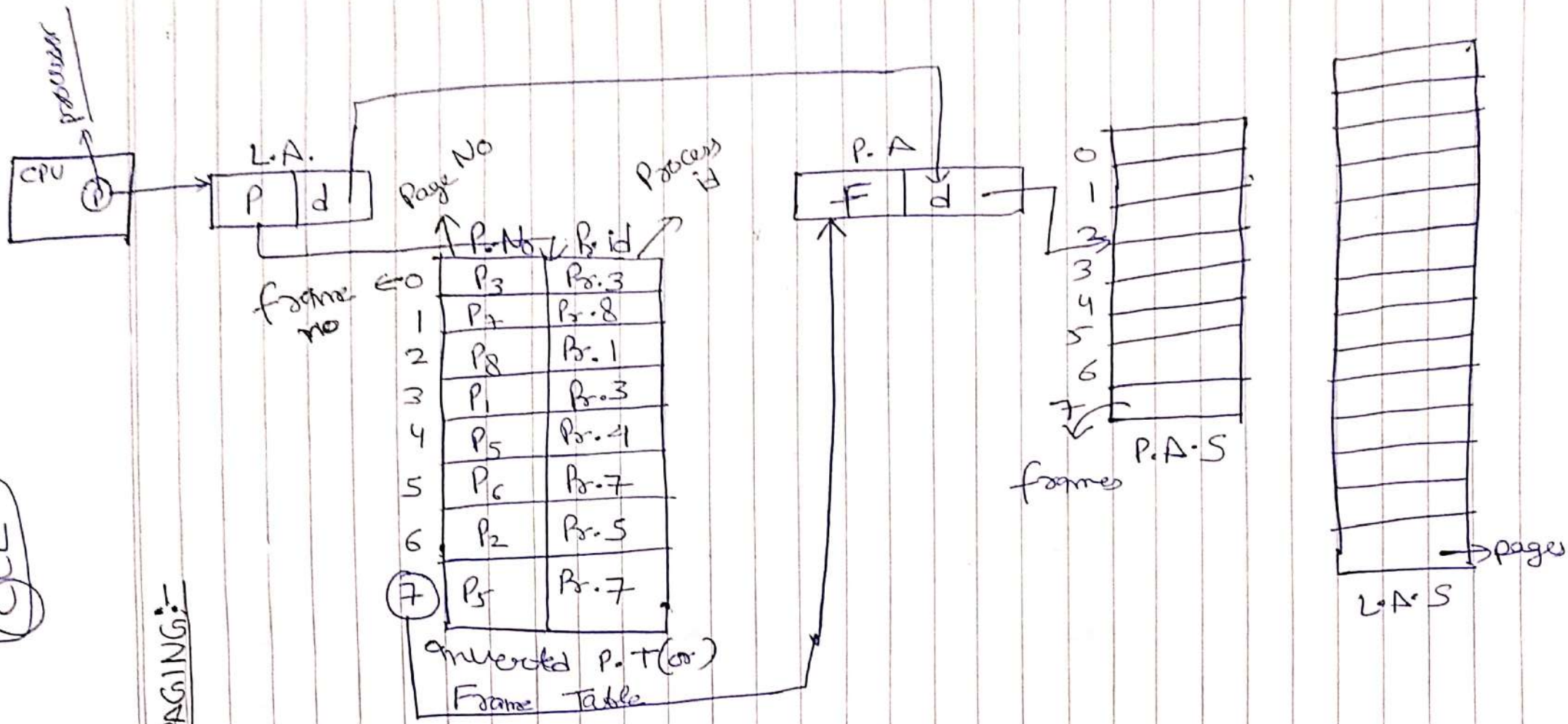
$$x^4 = (2^{13})^4$$

$$x = 2^{13} B$$

$$\boxed{x = 8 KB.}$$

option (C.)

INVERTED PAGING:-



Prateek Agarwal
89/100
B.Tech (Mechanical)

→ To avoid, the overhead of maintaining page table for every process, the concept of inverted paging will be implemented.

→ In the inverted paging, one one page table will be maintained for all the processes.

→ No. of entries in the inverted page table is same as the no. of frames in P.A.S.

→ The memory required to maintain page table for the process will be less, but the searching time for a corresponding page of a process will be more.

Eg:-

<P.No, Pr.id>
<P5, Pr7>

Q.) Consider a system which has logical address of 34 bits,
P.A = 29 bits
Page size = 16 KB
Page table entry size = 8 B.
Memory is byte addressable.
Calculate
i.) Conventional Page table size.
ii.) Inverted Page table size.

(Solve) $= 8 \times \frac{2^{34}}{2^{14}}$

$= 8 \times 2^{20}$ 8 MB.

ii.) $\Rightarrow 8 \times$ No. of frames

$\Rightarrow 8B \times \left(\frac{2^{29}}{2^{14}}\right)$

$\Rightarrow 2^{15} \times 8$

$= 2^{18} B$

$= 256 KB.$

P-106)

Workbook

□ + 12 bits

Q23.) L.A = 32 bits =

page size = 4KB.

(P.A.S) RAM = 128KB. 128^{2}

Page table E.S = 4B

Solve.) $\phi = \frac{4 \times 2^{32}}{2^{12}}$

$\frac{4 \times \frac{2^{17}}{2^{12}}}{2^{12}}$

2^{32-17}

(a) = $\boxed{2^{15} : 1}$

P-104) Workbook)

Q5.) ~~Logical A = 4KB~~

LAB - 32 bit

PAB = 30 bit

Page size = 4KB.

P.T.E.S = x+12

Page No = x.

No of pages = $\frac{2^{32}}{2^{12}}$

$= 2^{20}$

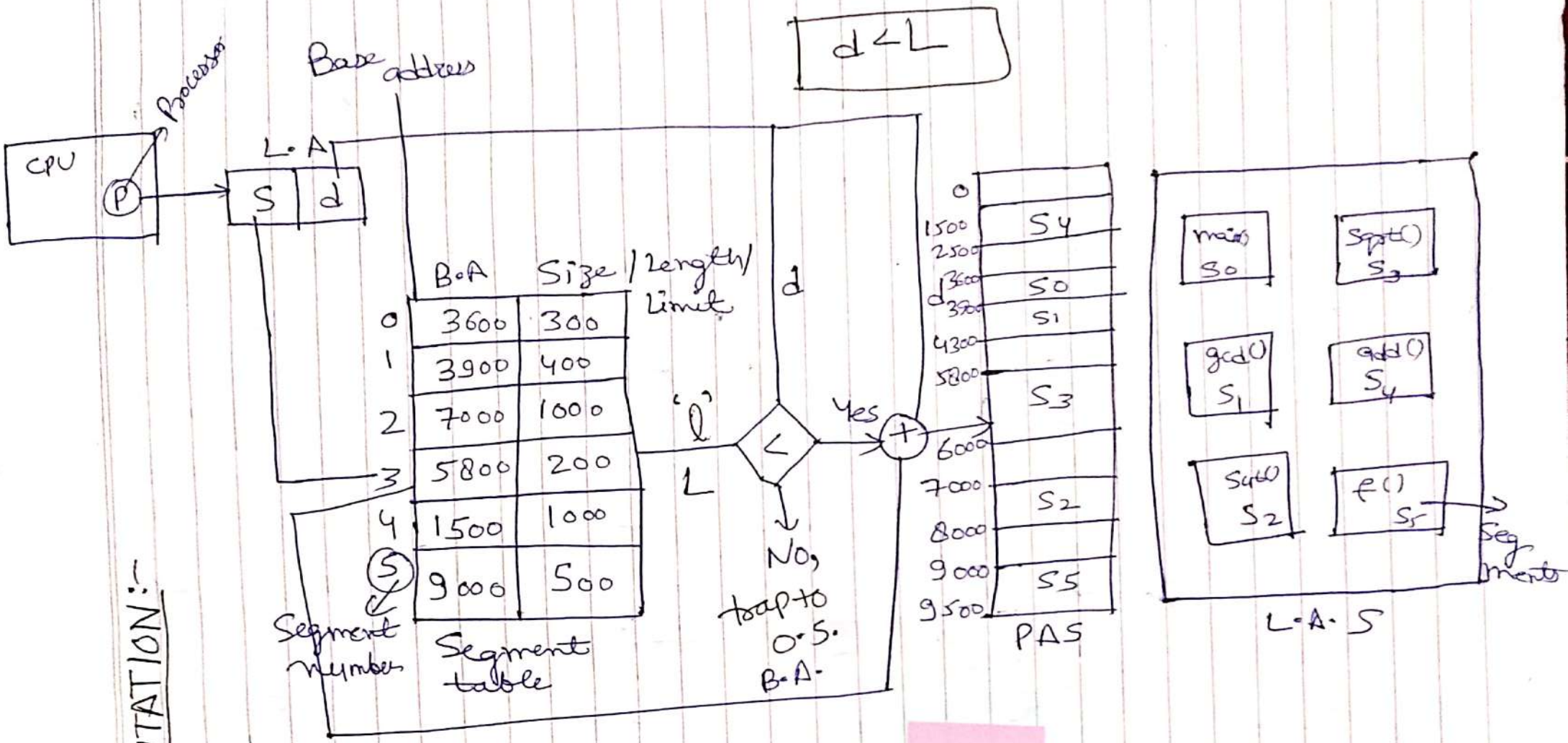
~~No. of frames = $\frac{2^{30}}{2^{12}} = 2^{28}$~~

= No. of entries × No. of frames

$= \left(\frac{2^{30}}{2^{12}}\right) \times \left(\text{P.No} + 12 \text{ bits}\right)$

$= \left(\frac{2^{30}}{2^{12}}\right) \times \left(\begin{array}{l} 20 \text{ bits} + 12 \text{ bits} \\ = 2^{18} \times 2^{2} B = \boxed{2^{20} B} \end{array}\right)$

SEGMENTATION:-



→ The paging does not follow users views of memory allocation.

→ To achieve, users view of memory allocation, the concept of segmentation will be implemented.

→ In a segmentation, L.A.S will be divided into various segments.

→ The segments of LAS will vary in the size.

→ Segments of LAS will be brought into P.A.S.

→ S = No of bits required to represents segments of L.A.S or segment number

→ d = No of bits required to represent word number of the segmator segment offset

→ No. of entries in the segment table is same as No of segments in LAS.

Eg :- ① d=210, L=200

d<L 210<200 ✗

⇒ Trap to OS.

Eg ② :- d = 190, L = 200

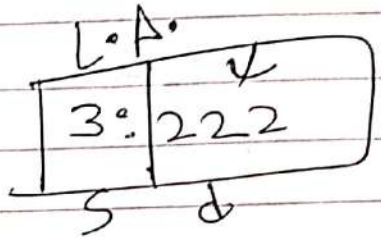
d < L

190 < 200 ✓

B.A + d ⇒ P.A.S.
5800 + 190 ⇒ 5990.

→ The variable size segments are brought from LAS to PAS, so it is similarly behaving like variable partition scheme. Hence, segmentation still suffers from External fragmentation.

Workbook
Page-106
Q24.)



498
222
720

⑨ ✓ 720

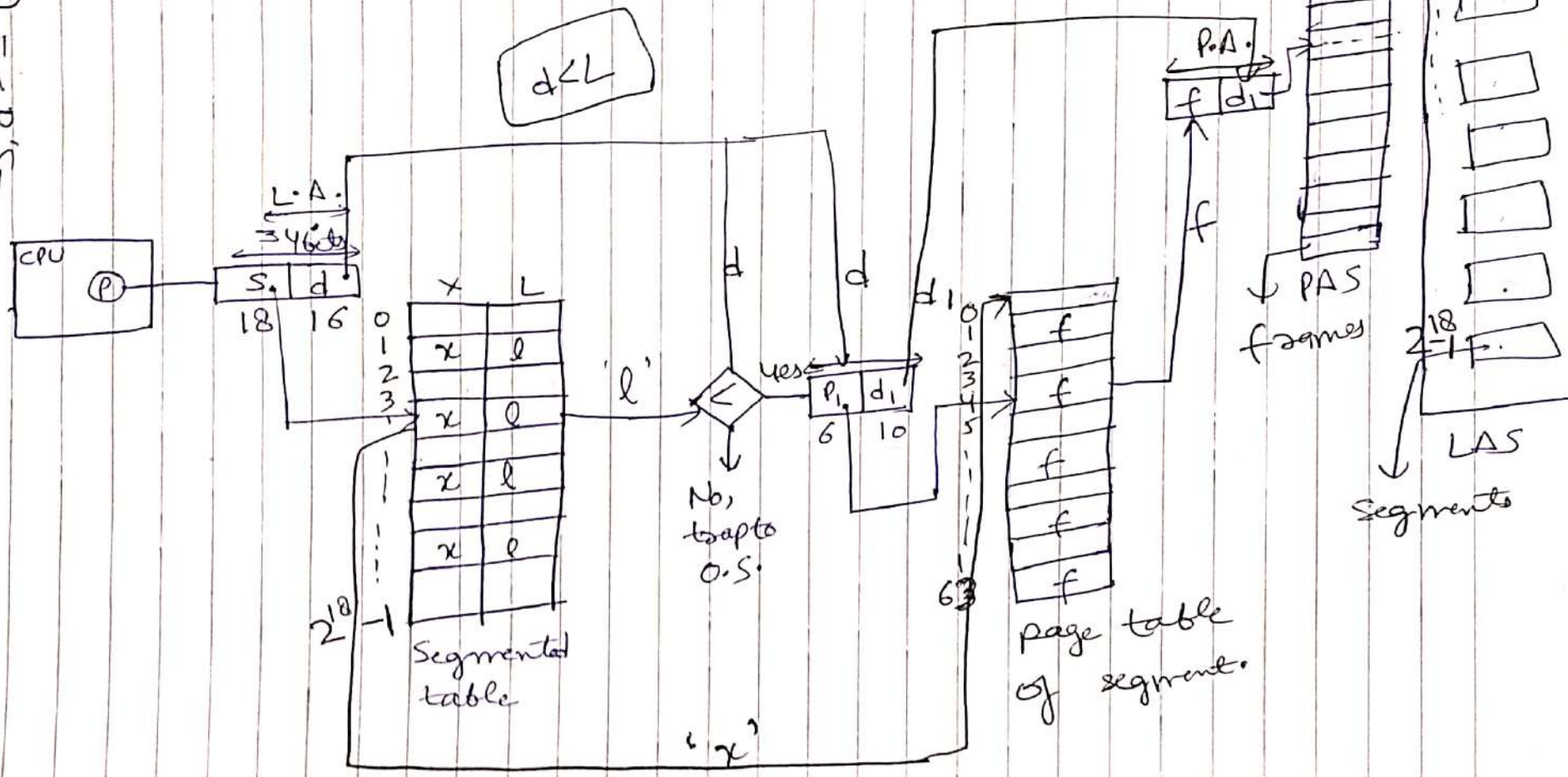
245

Paging on segment, when segment size increases

x = Base address of P.T. of respective Segment.
page size of segment = 1 KW.

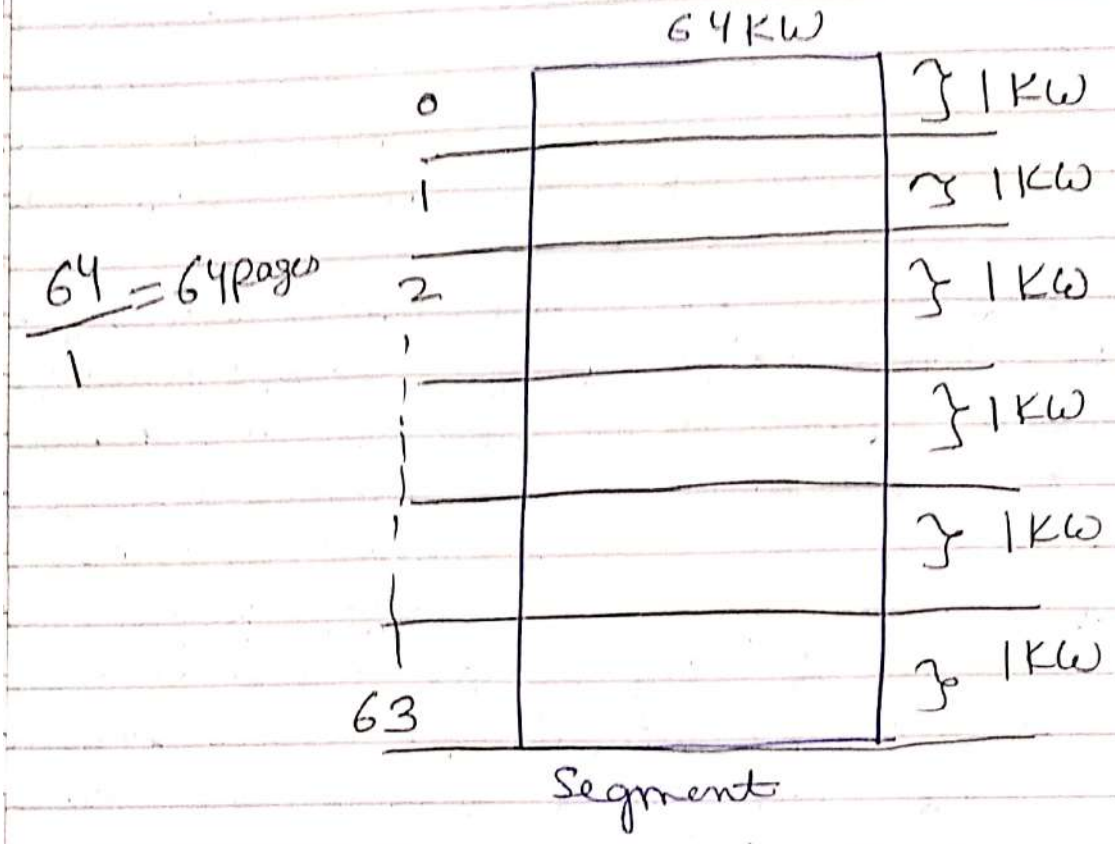
SEGMENTED PAGING :-

LA = 34 bits
<S, d> = (18, 16)



L.A = 34 bits
<s, d> = (18, 16)

Segment size = 2^{16}W = 64KW.



No of Pages of segment = Segment Size / size of a page in a segment.

→ To avoid the overhead of bringing large size segment into main memory, the concept of segmented paging will be implemented.

→ In the segmented paging, paging will be applied on the segment and instead of bringing the entire segment of memory, the pages of segment will be brought into memory.

No of entries in the page table of a segment is same as No. of pages on segment.

$P_1 \Rightarrow$ No of bits required to represent pages of segment or page no. of a segment.

$d_1 \Rightarrow$ No of bits required to represent word number of page of segment or page offset of segment.

Page size of segment is same as frame size of PAS.

CCL

Q.) Consider the system using Segmented paging architecture. The segment is divided into 1K pages. Each page having 512 entries. The memory is word addressable. The segment number requires 17 bits to represent all the segments of LAS and frame number requires 13 bit to represent all the frames of PAS. The page table entry size is 2 words. Then Calculate

(i.) Length of logical Address
(ii.) " " Physical Address
(iii.) Page table size of a segment.

(Ans.) (i.) 1K pages
$= 2^{10}$

S = 10 bits

PAS

PA = 13bit
PAS = 2^{13}

No. of pages in a segment = 512
$= 2^9$

L.A.S = 17bits

| 10 | 7 |

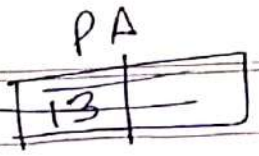
S = 10 bits
D = 7 bits

LAS = 2^{17} = 128KB

∴ LA = 17

~~Ph~~ = ~~48007~~ ~~LAS~~

(ii.) ~~P.A = 13~~

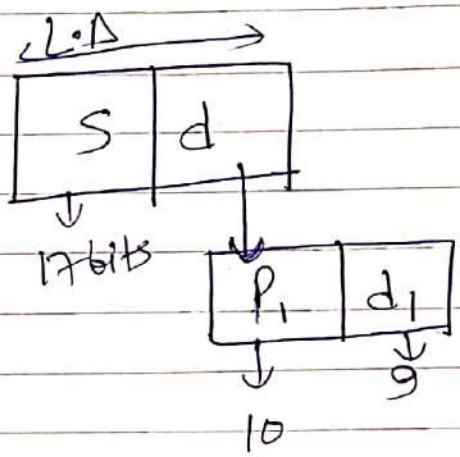


~~(iii.) Page table size = No. of entries in LAS × Page size~~

$= 2^{\cdot\cdot} \times 2^{1\cdot}$

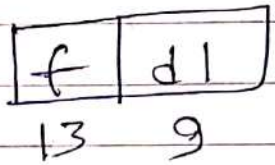
$= 2^{\cdot\cdot}$.

(1)



17 + 10 + 9 = 36 bits

(2.) P.A.



= 22 bits.

(3.) Page table size of a segment

= No. of entries × entry size

= No. of pages on segment × entry size

= 1K × 2W

= 2KW

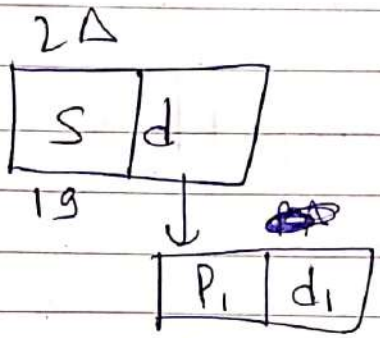
2^{10} $2^3 \times 2^{10}$

9.) Consider a system using segmented paging architecture. The segment is divided into 8K pages. and each page have 2K entries. The memory is byte addressable. The segment No requires 19 bits to represent all the segments of LAS. The PAS is 512Kb. And Page table entry size is 8 bits.

Calculate

(i.) Length of L.A.
(ii.) Length of P.A.
(iii.) Page table size of segment.
(iv.) No. of frames in PAS.

Solve:) (i.)

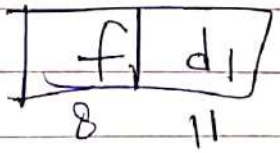


$19+13+11$
$= 43$ bits.

$d_1 = 2^{10} \times 2$
$= 2^{11}$

PAS $= 2^9 \times 2^{10}$
$= 2^{19}$

(ii.) P.A



PAS = 19 bit.

iii) Page table size = No of entries × P.T.E.S.

= $2^{13} \times 1$

= 2^{13}.

= 8KB.

iv)

No of frames in PAS = $\frac{\text{PAS}}{\text{frame size}}$

= 2^8. = (using PAS).

= 256.

Workbook
Page 106)

52 - 119 B

Q 28.) L.A.S = 2^{16} B
P.A.S = 2^{16} B

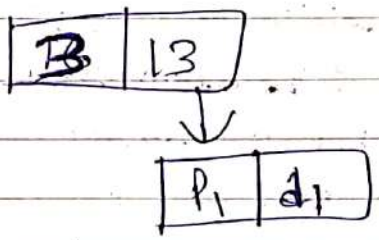
No of segments (pages on) = Equal size segments

= 3 bit.

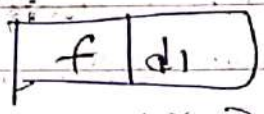
P.T.E.S = 2 B,

2^{-}

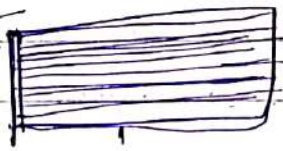
L.A. = 16 Bit



P.A.



2^3



8 segments

Segment size = 2^{13} Bytes.

Page size of segment = 2^{x} Bytes.

Page size of segment = Page table size

Page table size of a segment = B * No of Entries in PT of segment × P.T.E size

$$= \text{No of pages on segment} \times \text{PT. E. Size}$$

$$= \frac{2^{13}\,B}{2^{x}\,B} \times 2B$$

$$= 2^{14-x}\,B.$$

Page T. size of a segment = frame size of PAS

$$2^{14-x}\,B = 2^{x}\,B$$

$$2^{14} = 2^{2x}$$

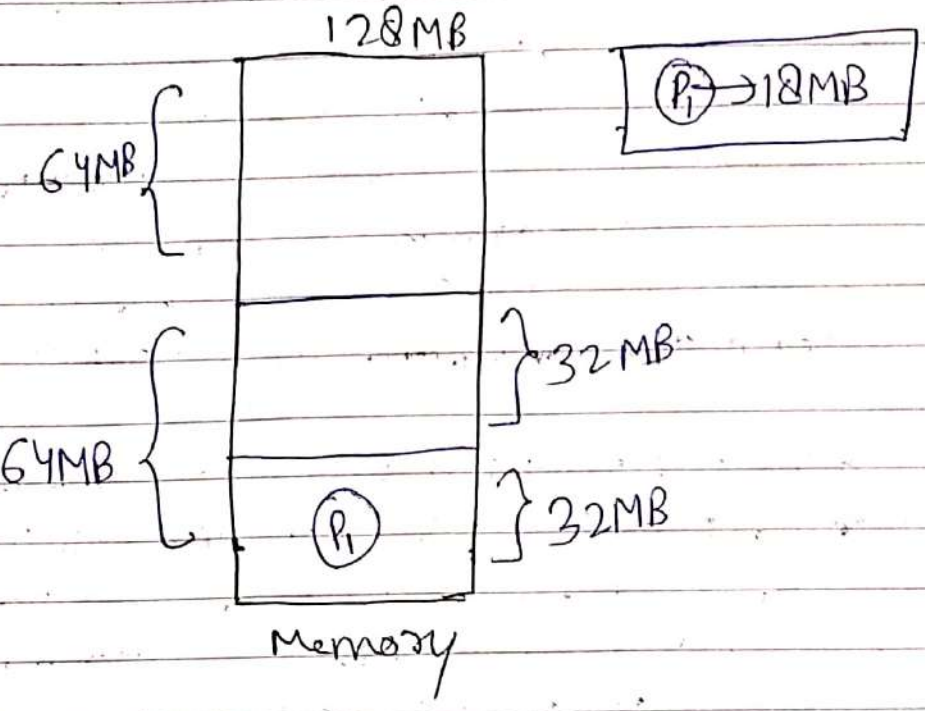
$$2x = 14$$

$$\boxed{x = 7}$$

page size of segment $= 2^{x}B = 2^{7}B$

$= 128\,B.$

Buddy System:-



→ In the buddy system, initially the memory will be single continuous free block.

→ Whenever the request by the process arrives memory will be divided into "two half blocks".

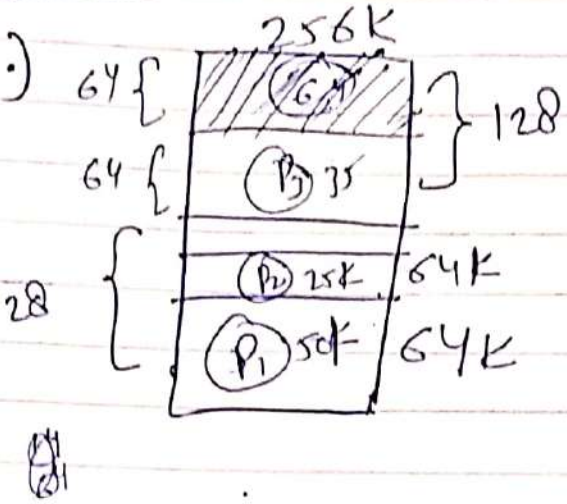
→ If the request by the process is too small, then the lower block of the memory is further divided into "two half blocks" again.

→ In the buddy system, memory will be allocated from lower blocks to higher blocks.

Workbook

P-105

Q 4.)



$\frac{256}{2} = \frac{128}{2} = 64$

(9) two block (32K, 64K)

3.) (9) 120K.

Q5.) (6.)