

Assume process 4 requested (3, 3, 0), Is the system Safe?
available $0 \ 0 \ 2$ ∇ Not Safe.

Banker's Algorithm:-

- We will consider only one resource type for simplicity.
- Each process declares its maximum needs at the beginning.
- When a process request a resource it might has to wait.
- When the process gets all resources it must release them in a finite time.

$\Rightarrow$ Data structure used :

Array max ,

$max[i] = j$, means P_i will need a max of j units of the resource.

Allocation;

$Allocation[i] = j$, means P_i is currently allocated j units of the resource.

Array $NEED$,

$NEED[i] = j$, means P_i needs j units of the resource.

$$\nabla NEED[i] = max[i] - Allocation[i]$$

- Available,

Available = W , W is the available number of units available of the resource.

→ Algorithm (Banker's)

1 - Let W = available;

2 - Define an array $K[i] = 1 \forall i = 0, 1, \dots, n-1$

3 - Find an i such that:

$K[i] = 1$ & $Need[i] \leq W$

If no such i exists GoTo step(5)

4 - $W = W + Allocation[i]$

$K[i] = 0$

GoTo step(3)

5 - IF $K[i] = 0 \forall i$ then

System is SAFE

else

System is UNSAFE.

K	
1	
1	
1	
1	
1	
1	

example:

Process	max	Allocation	Needs
P_0	10	5	5
P_1	4	2	2
P_2	9	2	7

available ($W = 3$)

3 5 10
12

10
10
10
K

Chapter #8

Memory Management.

'Ordinary Memory Management'

↳ means: all programs must be admitted (allocated to memory before execution starts)

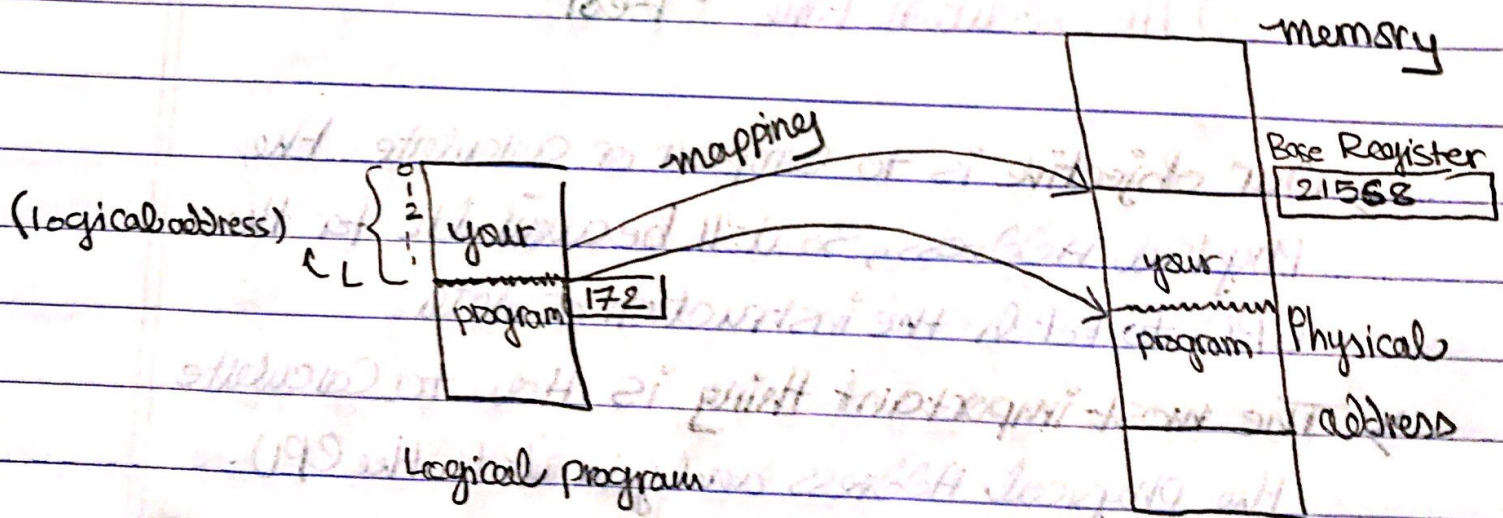
Logical Address VS. Physical Address:-

* Logical Address:

The address seen in your program, It's the offset of the address in the program.

* Physical Address:-

It's the actual address in memory.



$$PA = LA + \text{Base Register}$$

$$PA = 172 + 21568$$

$$= 21740.$$

☐ Binding Times:

When the OS determines the physical addresses?

(1) At Compilation time.

The PAs are assigned at the beginning, which means the program must be loaded into memory every time at the same location. Also notice that the program can't change its location during execution.

(2) At Loading time.

The PAs are decided when the program is loaded into memory.

⚠ Problem: the program can't be moved during execution.

(3) At execution time. 'Best'

→ Our objective is to compute or calculate the physical address, so it'll be available for the CPU to fetch the instruction or data.

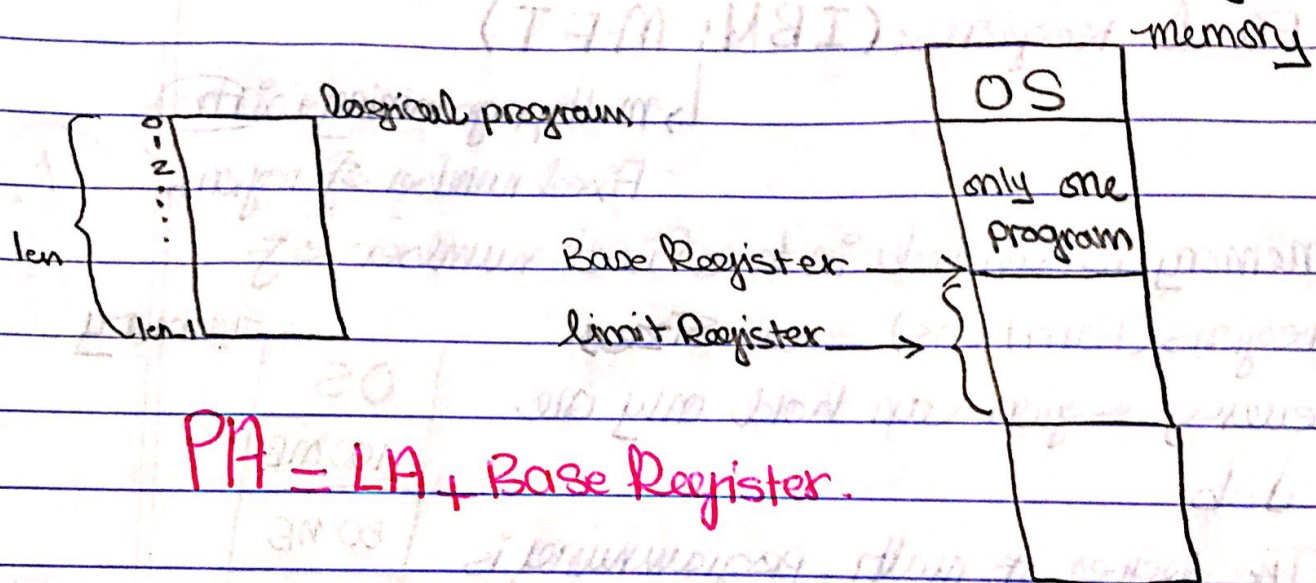
→ The most important thing is how to calculate the physical address and give it to the CPU.

↓
in memory management

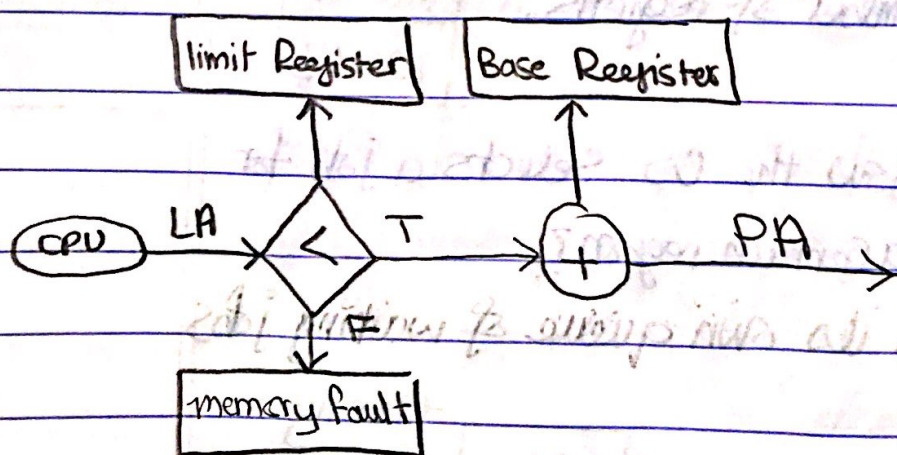
[1] ^{مقاله} Contiguous Allocation - Multiple Partitions (Regions)

- Memory is divided into Partitions or regions.
- Every region can hold only one process.
- When a region or partition becomes free, a new program is loaded in to it.
- **Hardware support means:** what data structure we need to compute the PA?

Answer: we need Base & limit Registers.



$$PA = LA + \text{Base Register}$$



There are two variants from this MM algorithm:-

- (1) Fixed Regions.
- (2) Dynamic (variable) Regions.

Multiple partitions:

- Base & Limit.

- $PA = Base + LA$.

} Revision.

(1) Fixed Regions: (IBM: MFT)

↳ multiprogramming with Fixed number of regions.

- Memory is divided into a fixed number of regions (Partitions) and sizes.

- every region can hold only one job

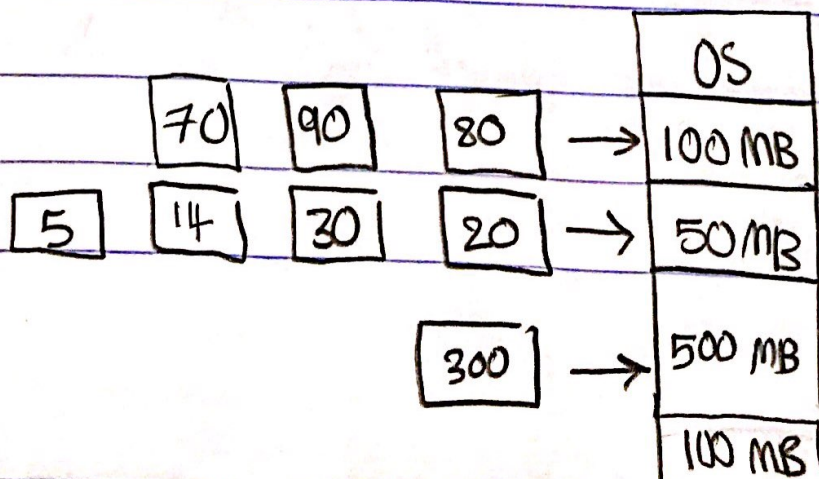
- The degree of multi-programming is bounded by the number of regions.

OS
100 MB
50 MB
500 MB
100 MB

memory

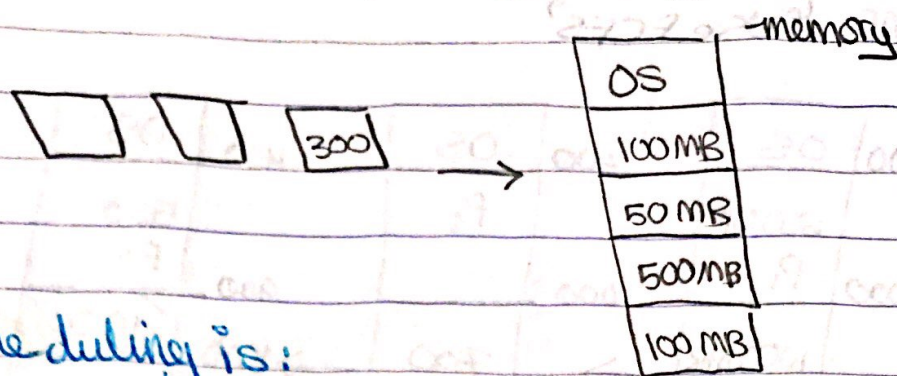
Job scheduling, How the OS selects a job for a certain region?

(a) Each region has its own queue of waiting jobs



memory

(b) There's only one queue of waiting jobs.



* Scheduling is:

- (1) FCFC with or without skip.
- (2) Best Fit only.
- (3) Best available Fit.

⚠ Problems: **Internal Fragmentation:**

The remaining unused memory inside the region.

⚠ External Fragmentation: The unused region which is small to fit any available job.

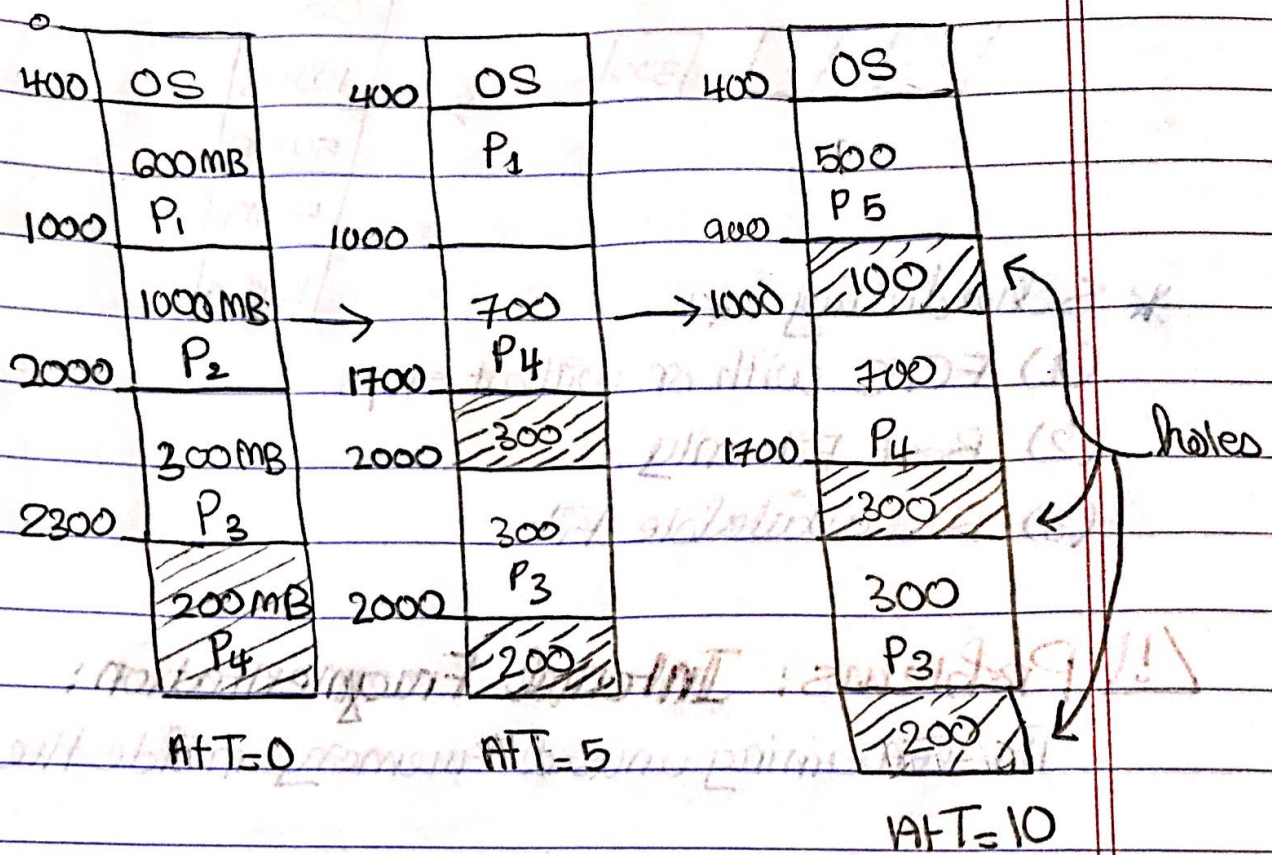
(2) **Dynamic (variable) regions:**

example:

assume we have the following queue of jobs:

Process	memory needed	time in memory
P ₁	600 MB	10
P ₂	1000 MB	5
P ₃	300 MB	20
P ₄	700 MB	8
P ₅	500 MB	15

Assume we have memory 2500 MB, OS is reserving 400 MB. Use FCFS?



after a while, memory will contain:

— allocated regions.

— Set of holes 'External Fragmentation'

* Job Scheduling: How we select a hole 'Variable region' for a process?

(1) First Fit

(2) Best Fit.

(3) Worst Fit.

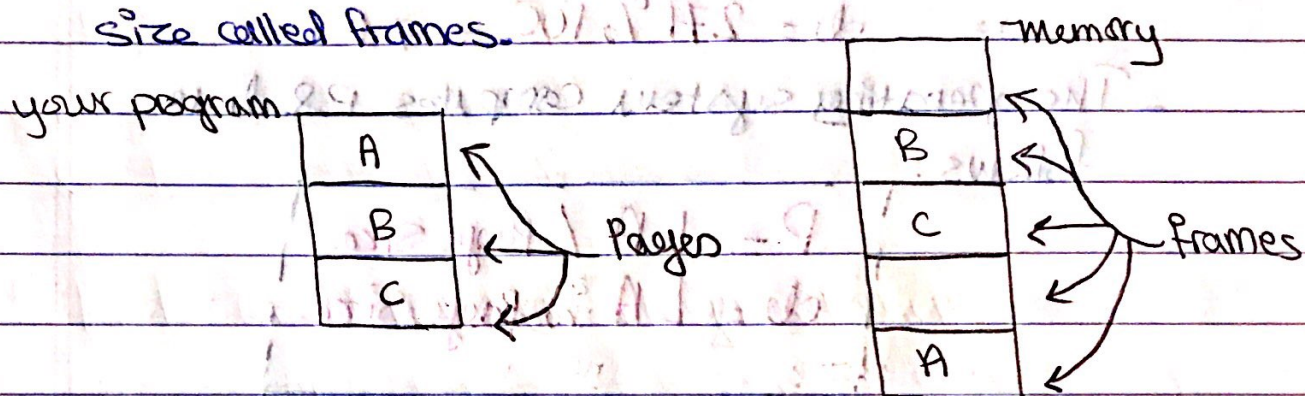
!! Problem: external Fragmentation. 'holes'

∴ Solution: Compaction.

[2] Non-Contiguous - Paging

- The logical program is divided into equal size partitions called pages.

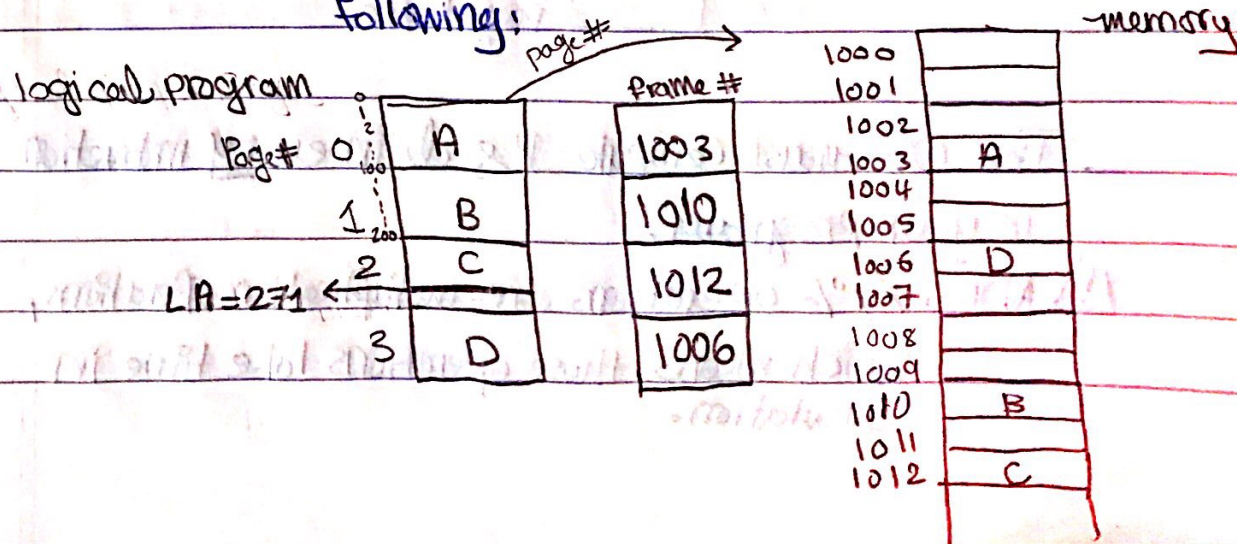
- Memory is divided into partitions of the same size called frames.

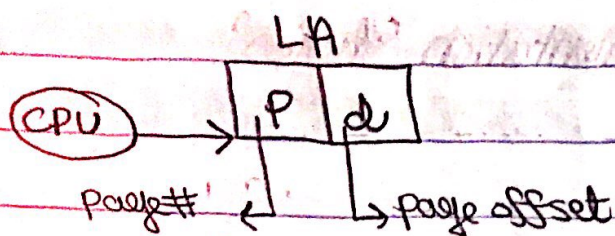


Hardware support needed to compute the PA:

we need what's called "page table", which is a table that contains the "frame numbers" of the pages in the logical program.

example: Assume page size = 100 bytes, given the following:





example: take the $LA = 271$

$$LA = 271 \Rightarrow \begin{cases} P = 2, \\ d = 71, \end{cases}$$

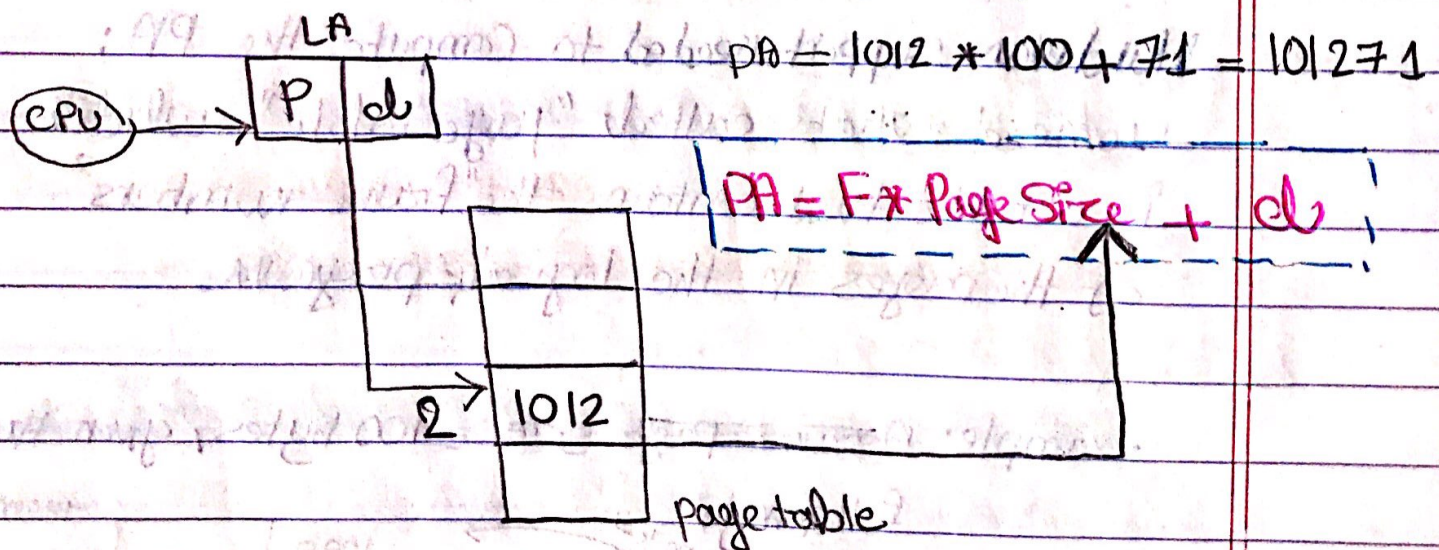
$$P = 271 / 100 = 2$$

$$d = 271 \% 100 = 71$$

The operating system computes P & d as follows:

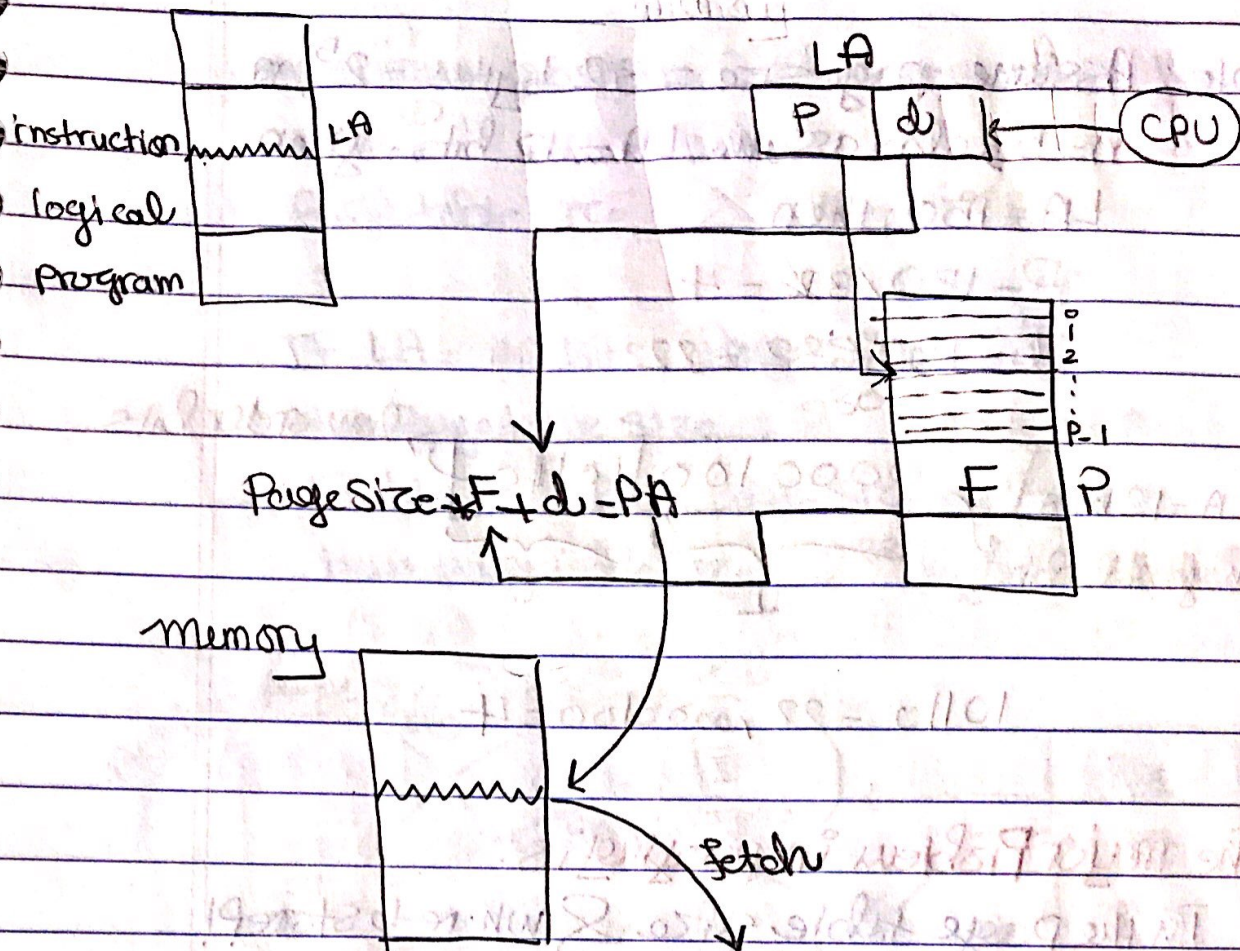
$$P = LA / \text{Page Size}$$

$$d = LA \% \text{Page Size}$$



The OS must compute P & d in every instruction in your program.

Note: $/, \%$ operations are multiplication operations, which means, these operations take time in calculation.



Does the OS, in real life performs these two operations?
 → NO

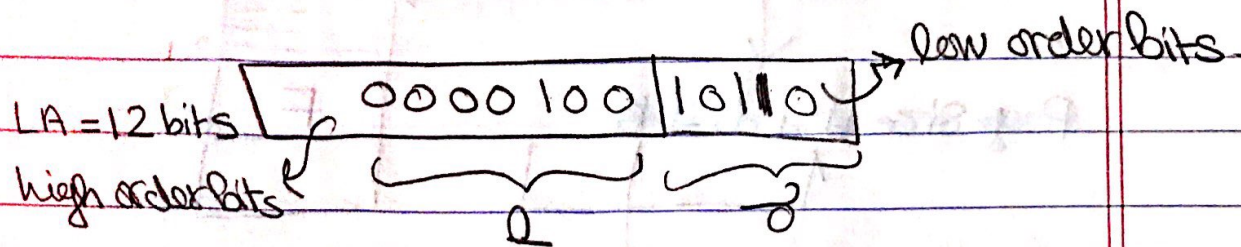
Note: the page size in practice is always 2^n bytes.
 Generally, $1024 \leq \text{page size} \leq 8192$
 $2^{10} \qquad \qquad \qquad 2^{13}$

most of the times, $\text{Page size} = 4096 = 2^{12}$
 In this case, the Low order "Low significant Bits" n bits of the logical Address represent d, and the remaining bits represent P.

example: Assume page size = 32 bytes = 2^5 , $n=5$, also assume LA = 12 bits, given LA = 150, then:

$$P = 150 / 32 = 4$$

$$el = 150 \% 32 = 22$$

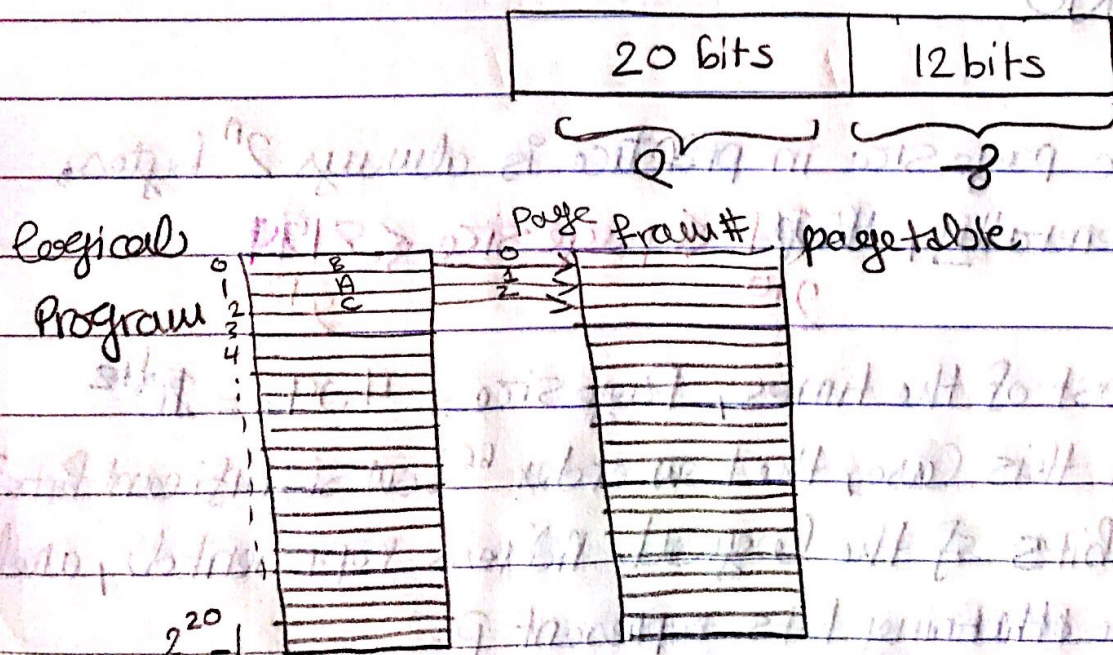


$$10110 = 22, 0000100 = 4.$$

!! The major Problem in paging is:

In the page table size & where to store it.

example: Assume LA = 32 bits, page size = 4096 = 2^{12}

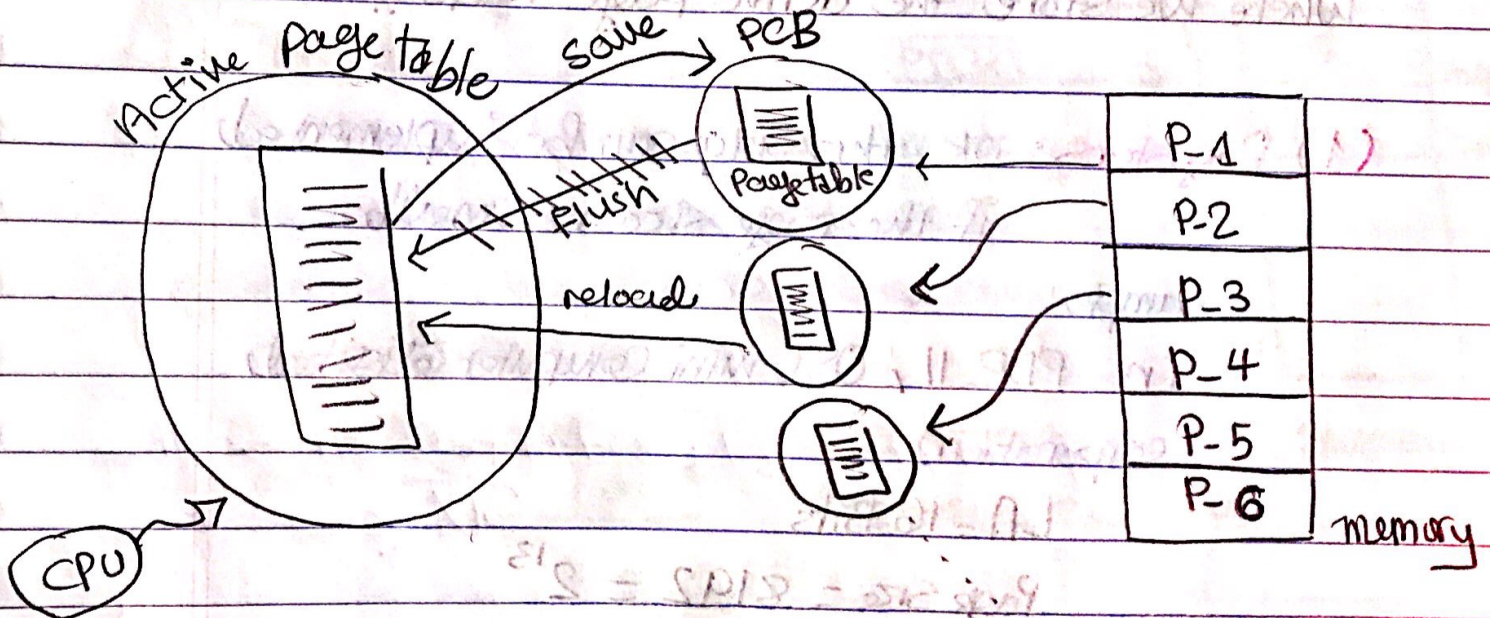


maximum program can be executed on this machine
 $= 2^{20}$

Page table size $= 2^{20} \times 4 = 4\text{MB}$

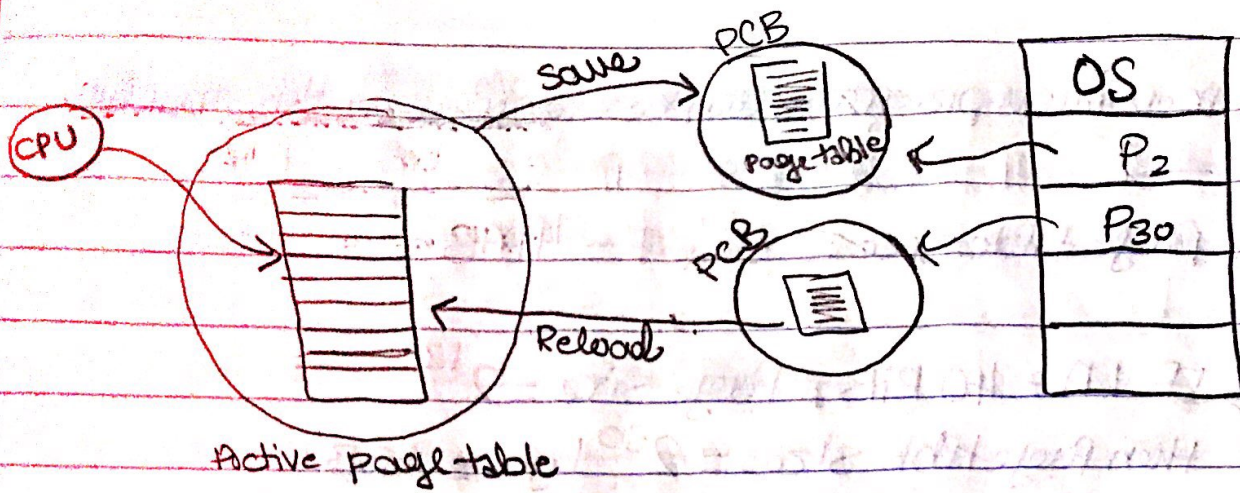
→ IF LA = 40 Bits, Page Size $= 2^{12}$
Then page table size $= 2^{30}$ byte $= 1\text{GB}$

→ IF LA = 48 Bits, Page Size $= 2^{12}$
Then page table size $= 2^{38}$ bytes $= 256\text{GB}$



Where does the OS store the page table?

The active page table is the page table executing



Implementation of the page table

Where we store the active page table?!

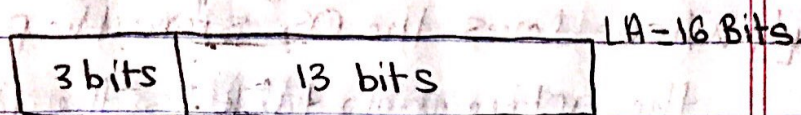
(1) **Registers**: OK, this only can be implemented if the page size is small.

example:

In PDP 11, OS (mini Computer digital corporation)

LA = 16 Bits

Page Size = 8192 = 2^{13}



0		F
1		F
2		F
...		...
7		

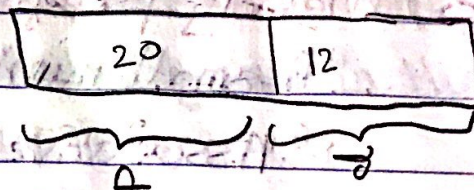
Logical Program page table

* max program can be executed
Containing only $2^3 = 8$ pages

* Page table size = $2^3 \times \text{Frame number size}$
= $2^3 \times 4 = 32$ bytes

But, If $LA = 32$ bits

$$\text{Page Size} = 4096 = 2^{12}$$

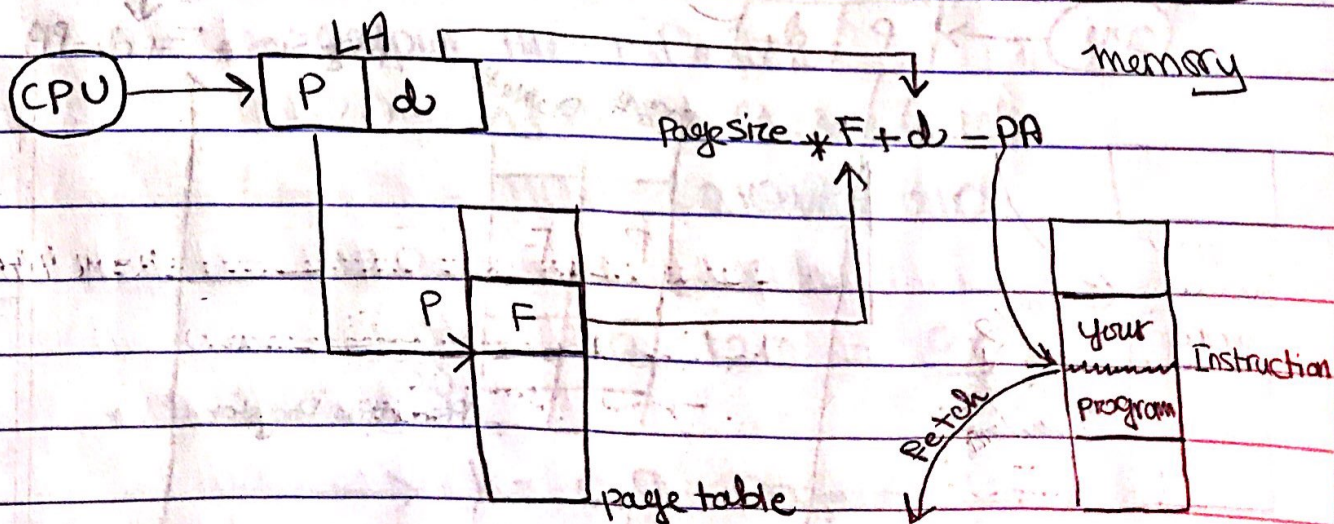
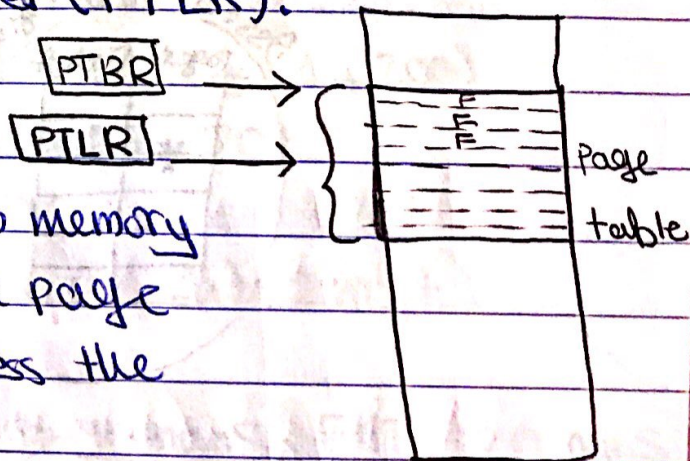


$$\therefore \text{Size of page table} = 2^{20} * 4 = 4 \text{ MB}$$

(2) Memory: Keep the page table in memory identified by the Page table base register (PTBR) & Page table Limit register (PTLR).

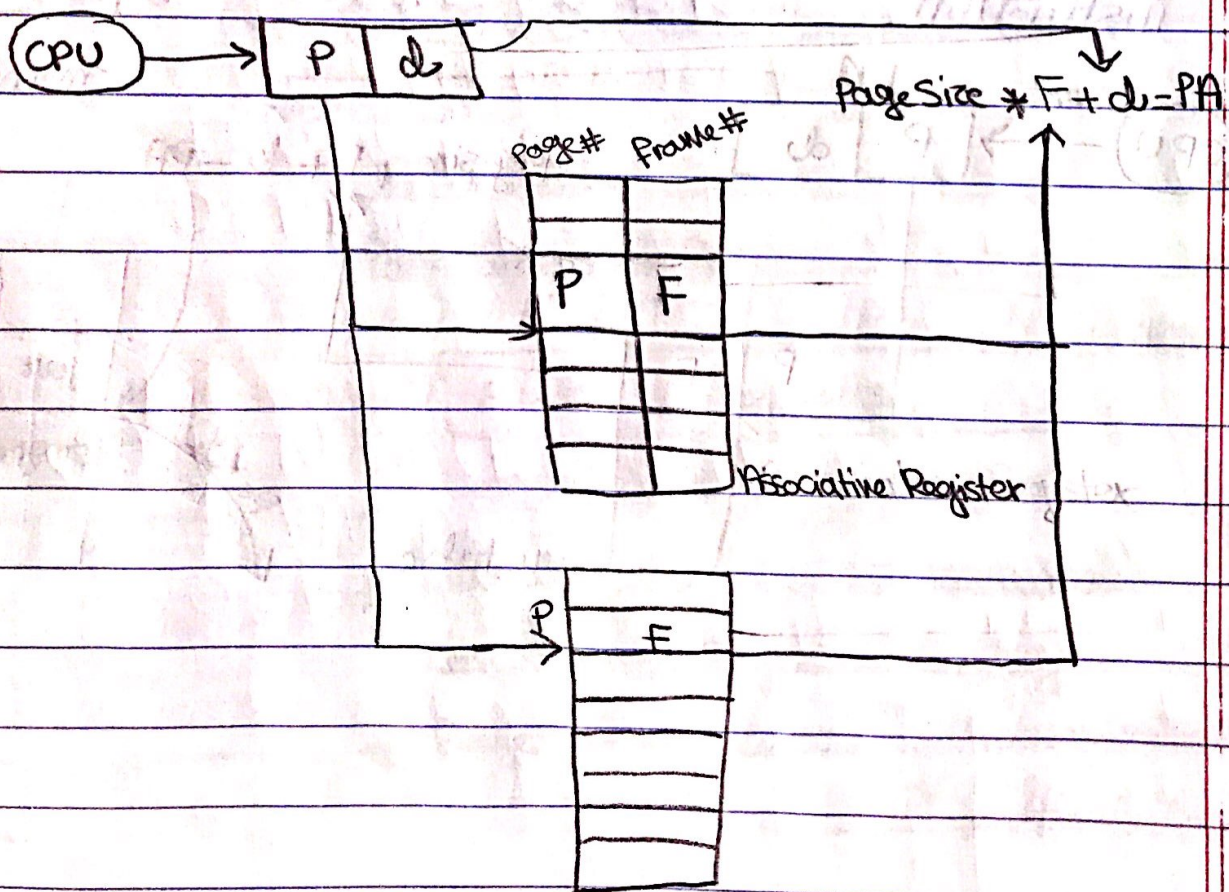
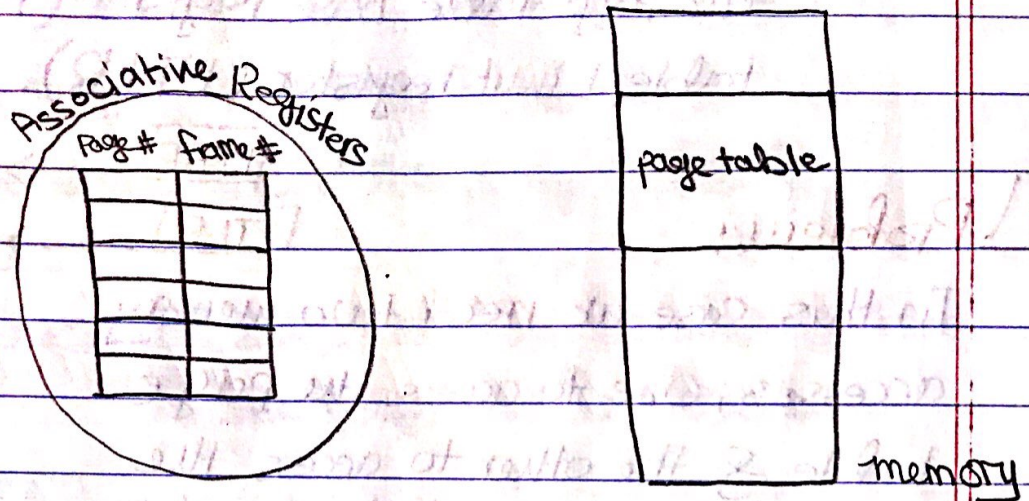
!! Problem:

In this case we need two memory accesses, One to access the page table & the other to access the instruction.



(3) Memory + Registers

- A small number of registers called ~~associative~~ 'Associative Registers' or 'translation Look aside buffers' are assigned or dedicated for a small page table whose entry contains (Page #, Frame #)



— Performance of Associative Register depends on Hit Ratio (h)

— Hit Ratio: Probability that the desired page ~~is~~ in the associative Register.

example:

① — memory access = 100 nans $\rightarrow m$

— search time in associative register = 1 nans $\rightarrow t$

— Hit Ratio (h) = 0.95

$\rightarrow$ Effective Access Time (EAT)

$$= 0.95 * (1 + 100) + 0.05 * (1 + 200)$$

$$= 0.95 * 101 + 0.05 * 201$$

$$= 108 \text{ nans}$$

$$\rightarrow EAT = h * (m + t) + (1 - h) * (2m + t)$$

② — $m = 100 \text{ nans}$, $t = 10 \text{ nans}$, $EAT = 120 \text{ nans}$

compute the hit Ratio:

$$120 = h(110) + (1 - h)(210)$$

$$= 110h - 210h + 210$$

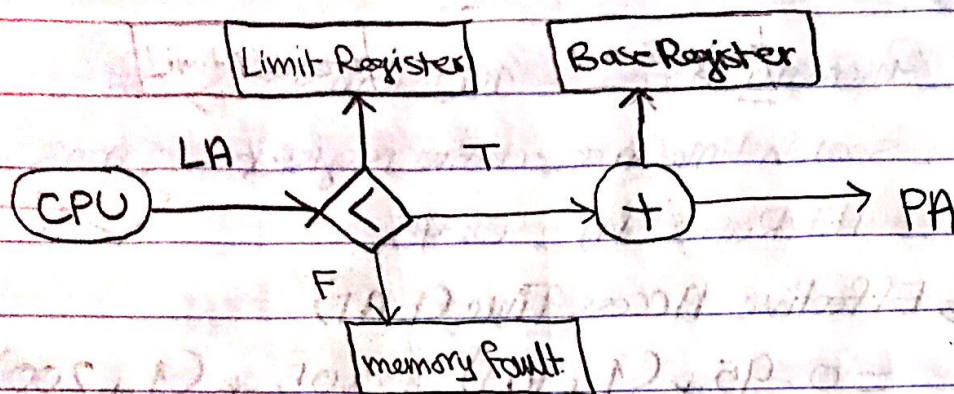
$$120 = 210 - 100h$$

$$\frac{100h}{100} = \frac{210 - 120}{100} = \frac{90}{100}$$

$$\rightarrow h = 0.9 \text{ perfect.}$$

Memory Protection:

In multiple partitions, memory protection is performed using base & Limit Registers.



In Paging, protection is performed using additional bits in the page table.

(1) Legal/Illegal Bits:

assume my program is 4 pages.

1: legal page

0: Illegal page

(2) R/W Bit (Read only Bit)

1: R/W page

0: Read only page

		L1/L	R/W
A	12510	1	1
B	7415	1	1
C	8001	1	1
D	120	0	0
		0	0
		0	0

Logical program ; page table

page table

☐ Paging Advantages:

Advantages of sharing pages?

U₁ - word

W ₁
W ₂
D ₁

page table

110
102
106

100	
101	
102	W ₂
103	
104	
105	
106	D ₁
107	
108	
109	
110	W ₁
111	
112	D ₂
...	
...	

U₂ - word

W ₁
W ₂
D ₂

page table

110
102
112

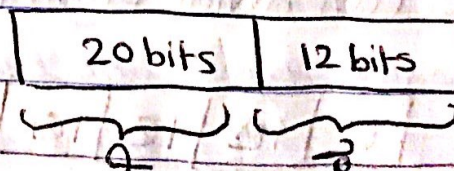
⚠ Disadvantage:

Some people have reservation that the program is divided into too many pieces in memory.

☐ Multi-Level page table:

① Assume LA = 32 bits.

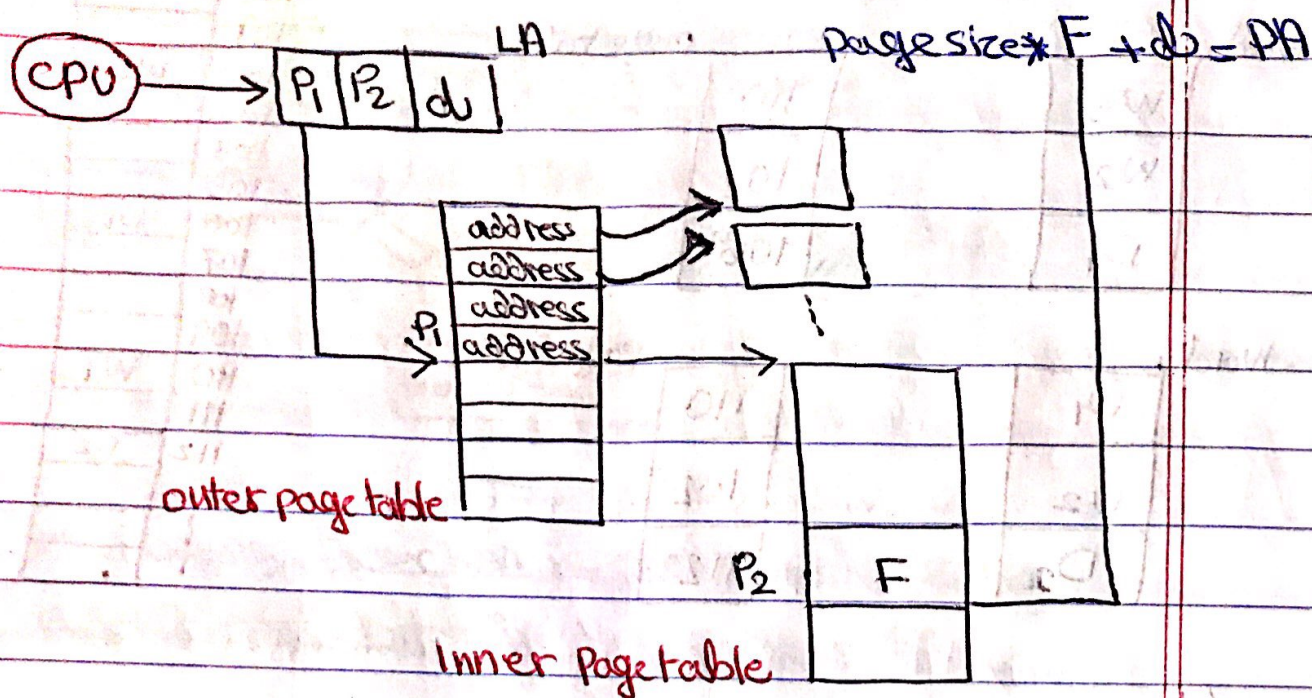
Page size = 4096 = 2¹²



∴ Page table Size = 2²⁰ * 4 = 4 MB

→ This is big to store contiguously in memory.

→ Let's divide the page table into two levels.



In our example, assume $P_1 = 8$ bits

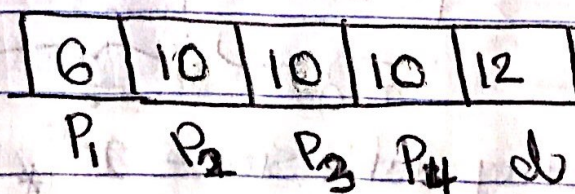
$P_2 = 12$ bits

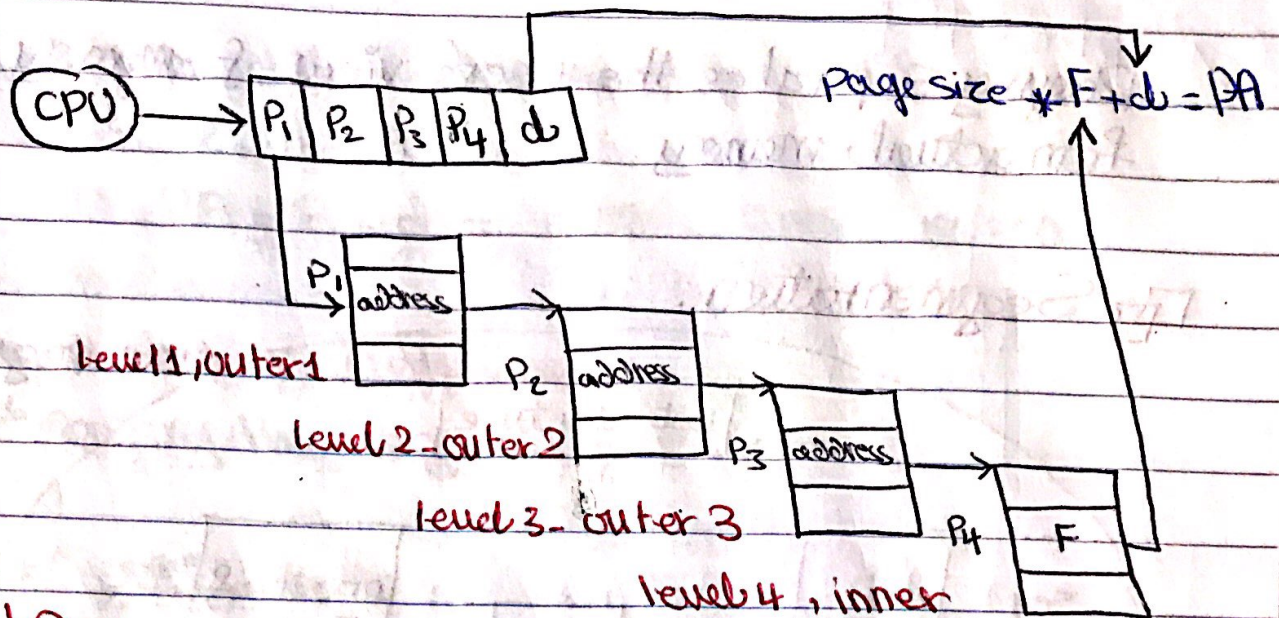
outer page table size = $2^8 * 4 = 2^{10} = 1\text{KB}$

Inner page table size = $2^{12} * 4 = 2^{14} = 16\text{KB}$

② $LA = 48$ bits, Page size = 2^{12} bytes

OS can divide LA into 4 levels





!! Problem:

→ we need more memory accesses.

— In two level page table, we need 3 memory accesses.

— one to get address (P_1).

— one to get F.

— One to fetch instruction.

— In 4-Level page table, we need 5 memory accesses. But, with good associative Register Algorithm, the performance will be as well good.

example: assume, — memory access time = 100 nans

— associative register search time = 1 nans

— assume 4 levels of Page table with

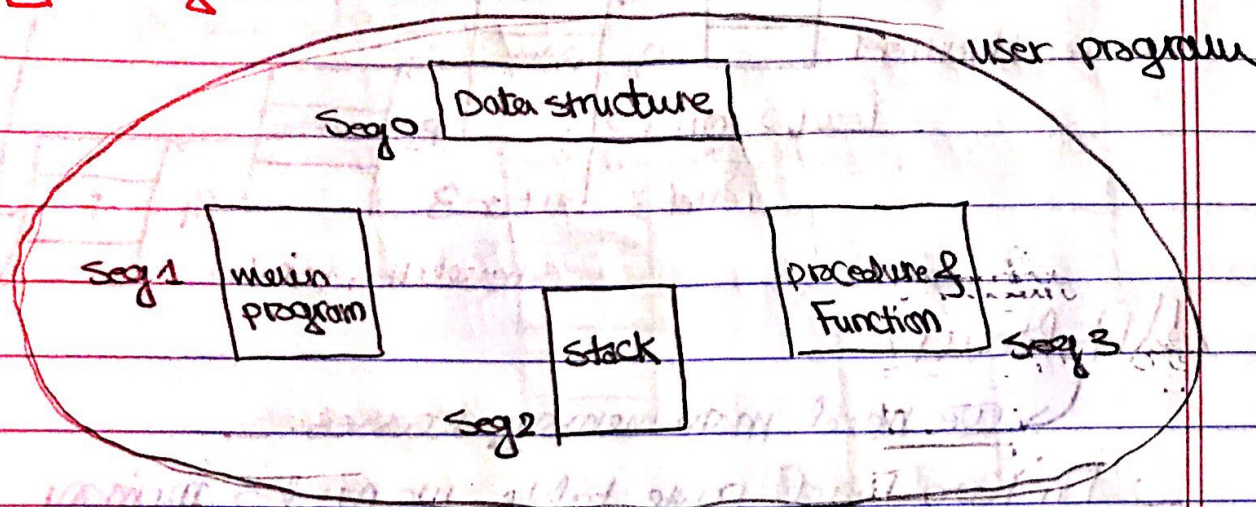
hit ratio (h) = 0.98

$$EAT = 0.98(100+1) + 0.02(500+1)$$

$$= 0.98 * 101 + 0.02 * 501 = 109$$

- Paging Separates the user's view of memory from actual memory.

Segmentation:



	Segment Base Register	segment Length	memory
0	27,000	7,000	Seg1
1	1,000	5,000	
2	18,000	3,000	Seg2
3	21,000	2,000	Seg3
			Seg0

Diagram illustrating memory layout and segment lengths:

- Segment 1 (Seg1) starts at 1,000 and ends at 18,000 (length 5,000).
- Segment 2 (Seg2) starts at 18,000 and ends at 21,000 (length 3,000).
- Segment 3 (Seg3) starts at 21,000 and ends at 27,000 (length 2,000).
- Segment 0 (Seg0) starts at 27,000 and ends at 34,000 (length 7,000).

S: Segment #

d: LA offset

