- Assum 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.

→ 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.

⚠ NEED[i] = Max[i] - Allocation[i],

Scanned with CamScanner

- Available,

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

Algorithm (Bankers)

1- let W = available;

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

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

K
~~1~~ 0
~~1~~ 0
~~1~~ 0

Chapter #8

Memory Management.

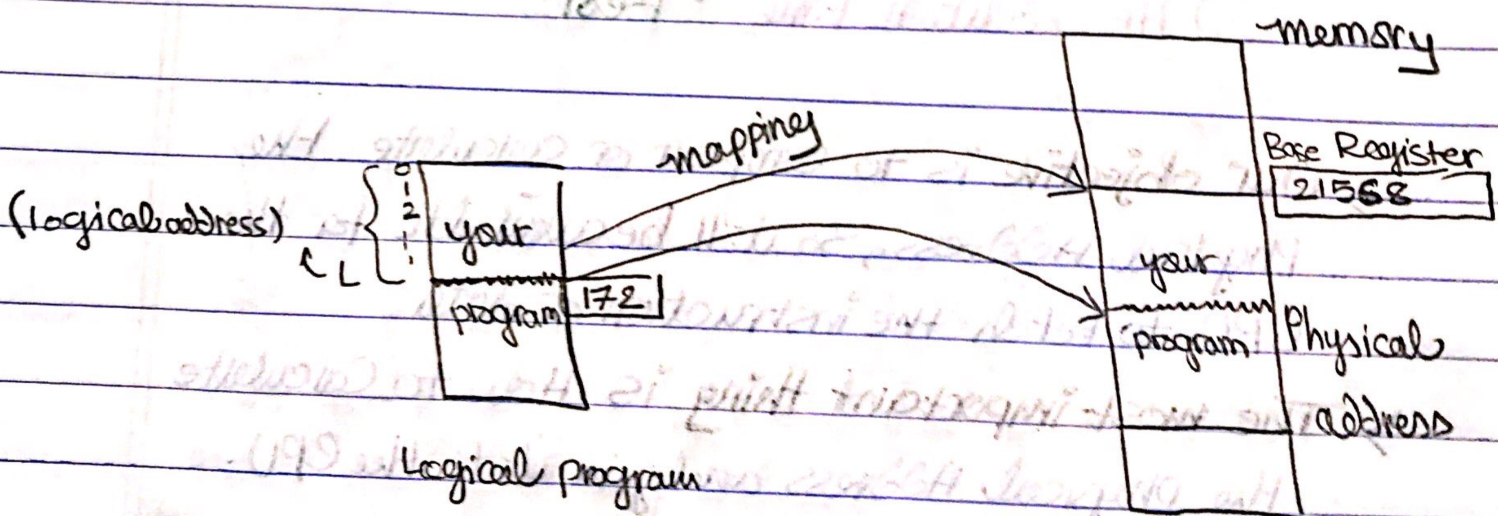
'Ordinary Memory Management'

↳ 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.



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

$PA = 172 + 21568$

$= 21740.$

Scanned with CamScanner

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 Computer or calculate the Physical Address, So it'll be available for the CPU to fetch the instruction or data.

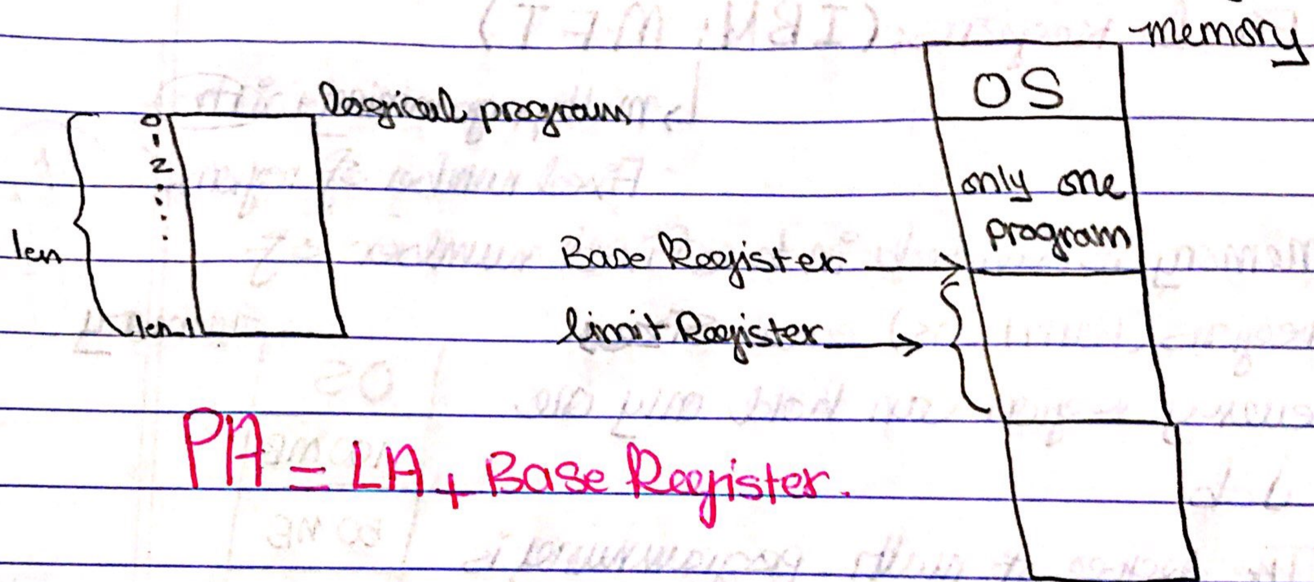
→ **The most important thing** is How to Calculate the physical Address and give it to the CPU. (in memory management)

تابع

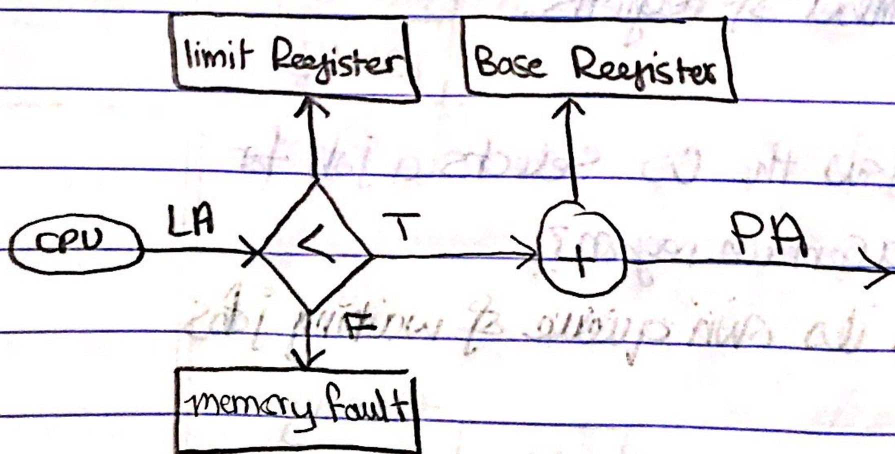
[1] Contiguous Allocation _ Multiple Partitions (Regions)

- 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 into it.
- Hardware support means: what data structure we need to compute the PA ?!

  Answer: we need Base & limit Registers.



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



Scanned with CamScanner

* There are two various 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.

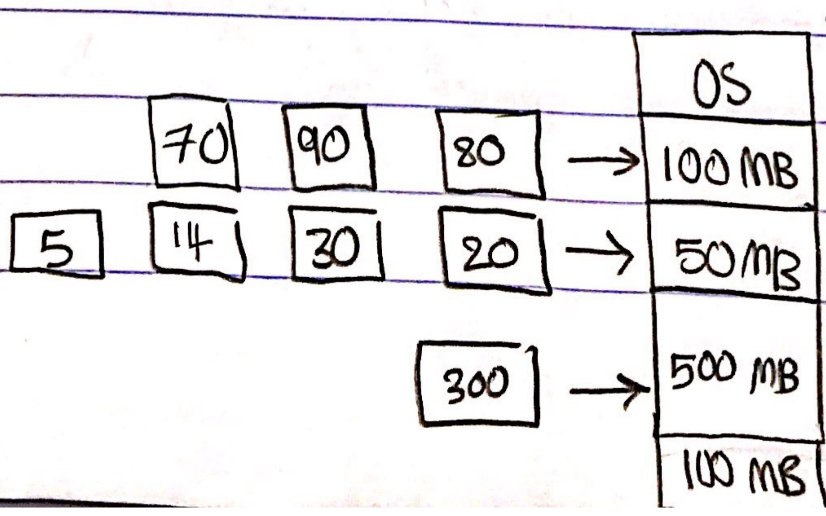
- Memory is divided into a fixed number of regions (Partitions) and Sizes.
- every region can hold only one Job
- The degree of multi-programming is bounded by the number of regions.

memory

OS
100 MB
50 MB
500 MB
100 MB

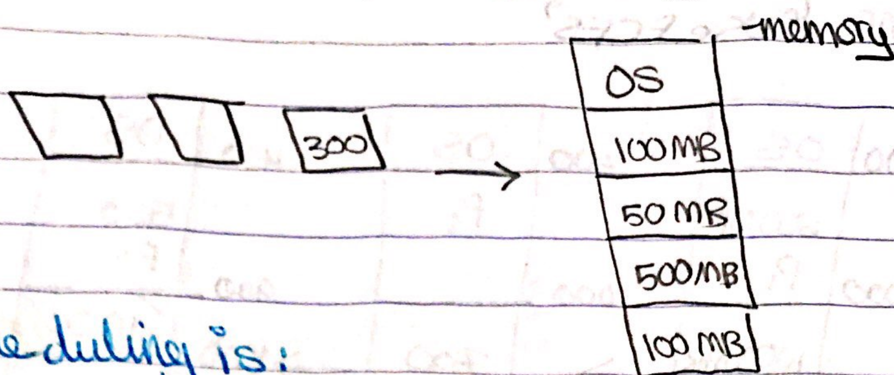
Job scheduling, How the OS selects a job for a certain region?!

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



Scanned with CamScanner

(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:

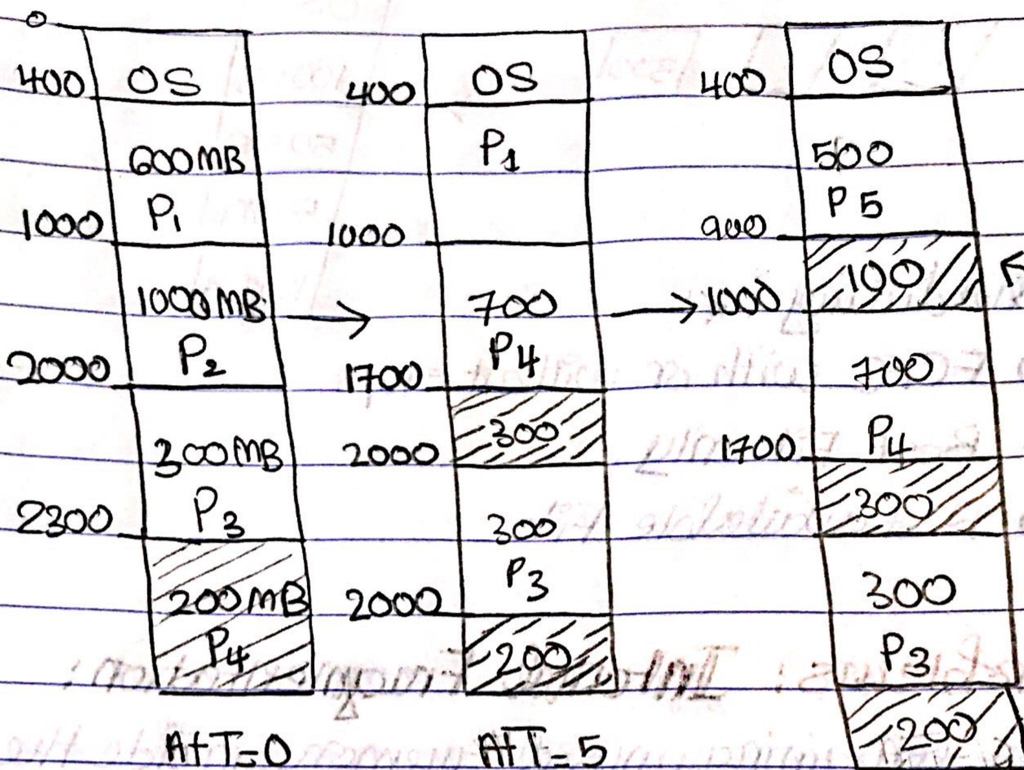
example:

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

Scanned with CamScanner

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



- after a while, memory will contain:
  - allocated regions.
  - Set of holes "Eternal Fragmentation"

* Job Scheduling: How we select a hole "variable region" for a process?!

(1) First Fit

(2) Best Fit.

(3) Worst Fit.

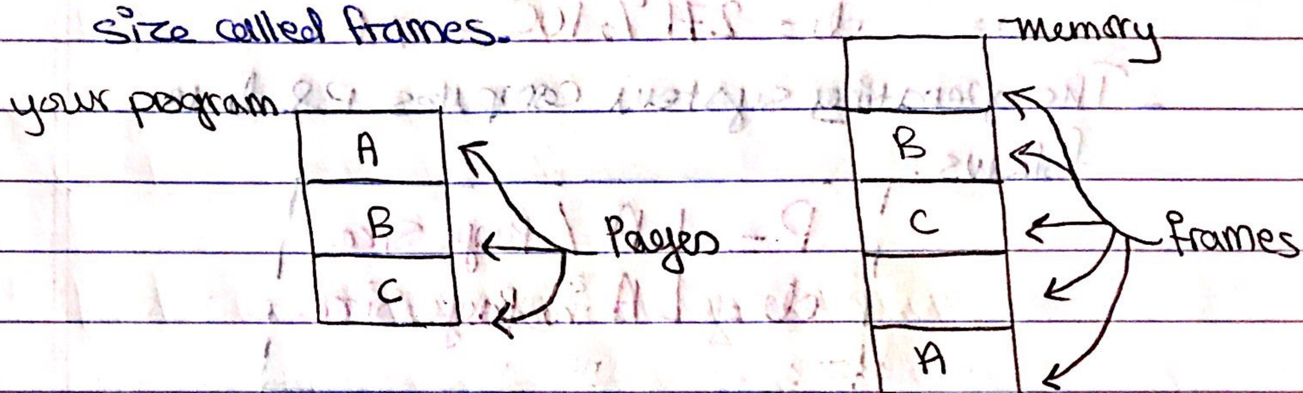
Scanned with CamScanner

⚠ Problem: external Fragmentation "holes"

∴ Solution: Compaction.

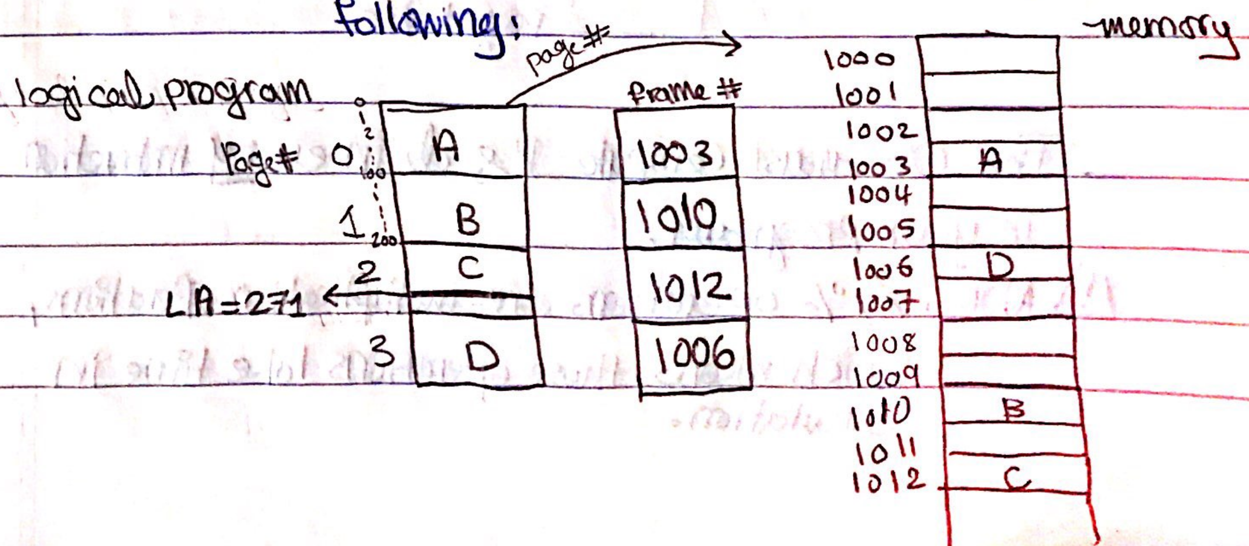
[2] Non-Contigious _ Paging:

- The logical program is divided into equal size partitions called Pages.
- Memory is divided in to partitions of the same size called frames.

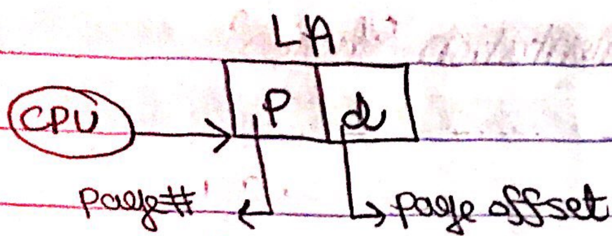


- Hardware support needed to Compute the PA: we need what's called "page table", which is a table that contains the "frame numbers" of the pages in the logical program.

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



Scanned with CamScanner



example: take the LA = 271

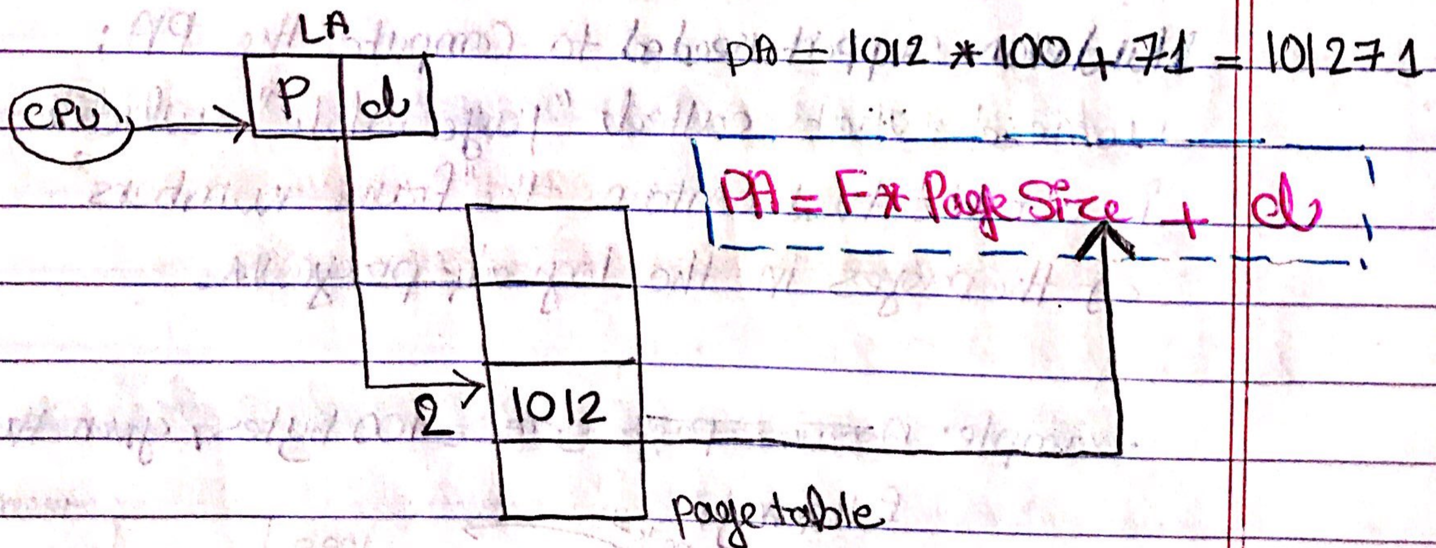
LA = 271 ⇒ { P = 2, d = 71,

$P = 271 / 100 = 2$

$d = 271 \% 100 = 71$

- The operating system computes P & d as follows:

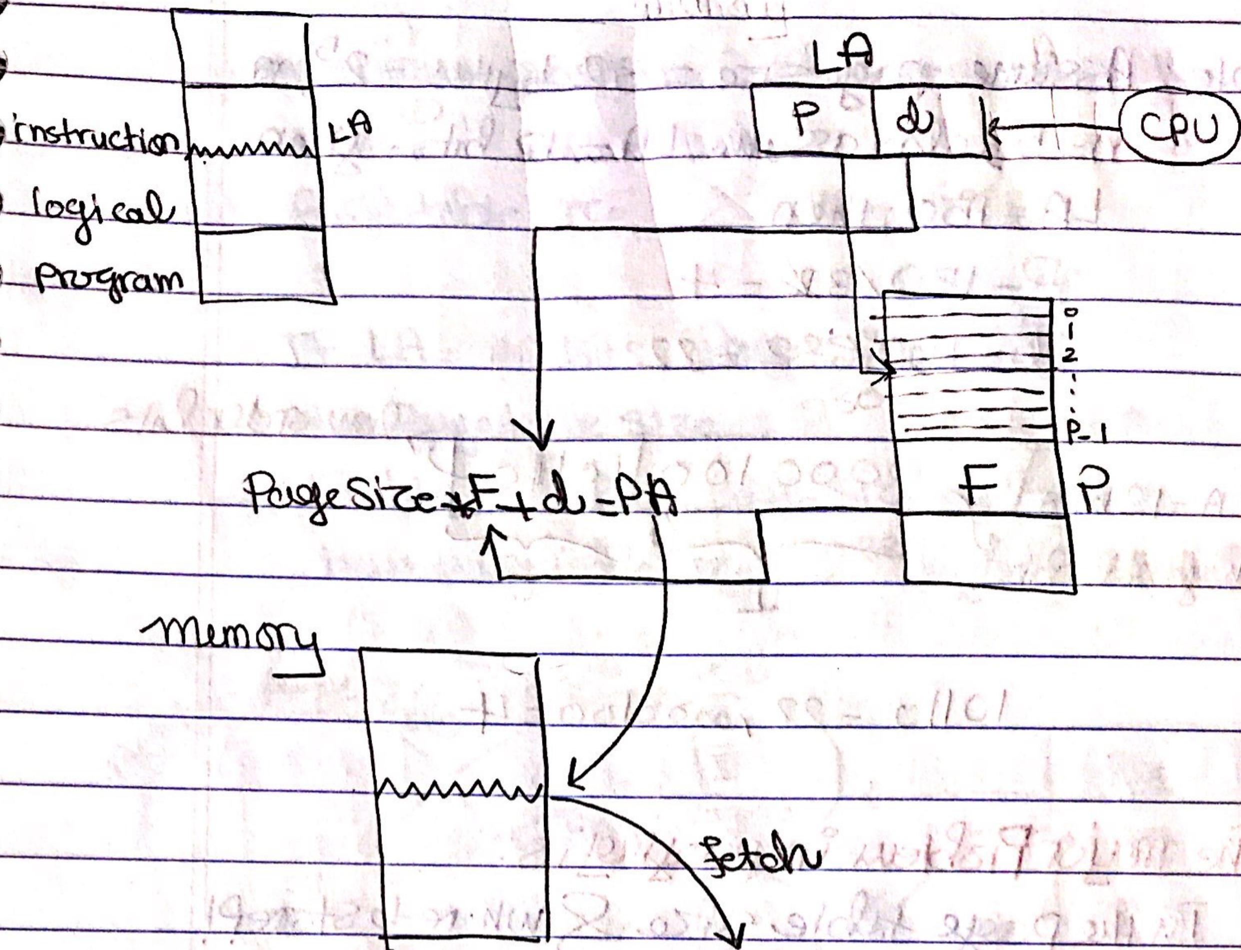
$$P = LA / \text{Page Size}$$
$$d = LA \% \text{Page Size}$$



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

⚠ Note: /, % operations are multiplication operations, which means, those operations take time in calculation.

Scanned with CamScanner



– Does the OS, in real life performs these two operations?

→ NO

⚠ Note: the page size in practice is always 2^n bytes.

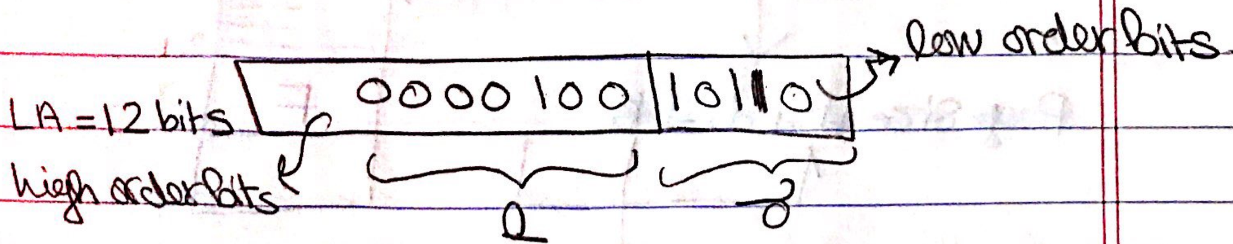
Generally, $1024 \leq$ page size ≤ 8192
2^{10} 2^{13}

most of the times, 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.

Scanned with CamScanner

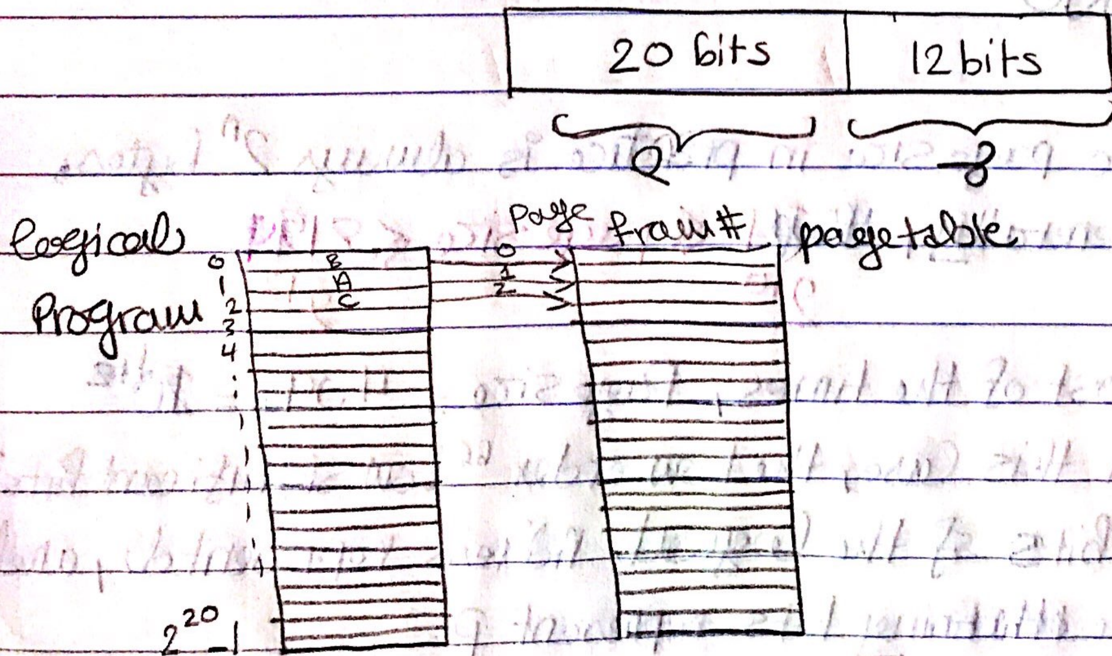
example: Assume pagesize = 32 bytes = 2^5,
n = 5, also assum LA = 12 bits, given
LA = 150, then:
P = 150/32 = 4
d = 150%32 = 22



10110 = 22, 0000100 = 4.

⚠ The major Problem in paging is:
In the page table size & where to store?!

example: Assum LA = 32 bits, pagesize = 4096 = 2^{12}

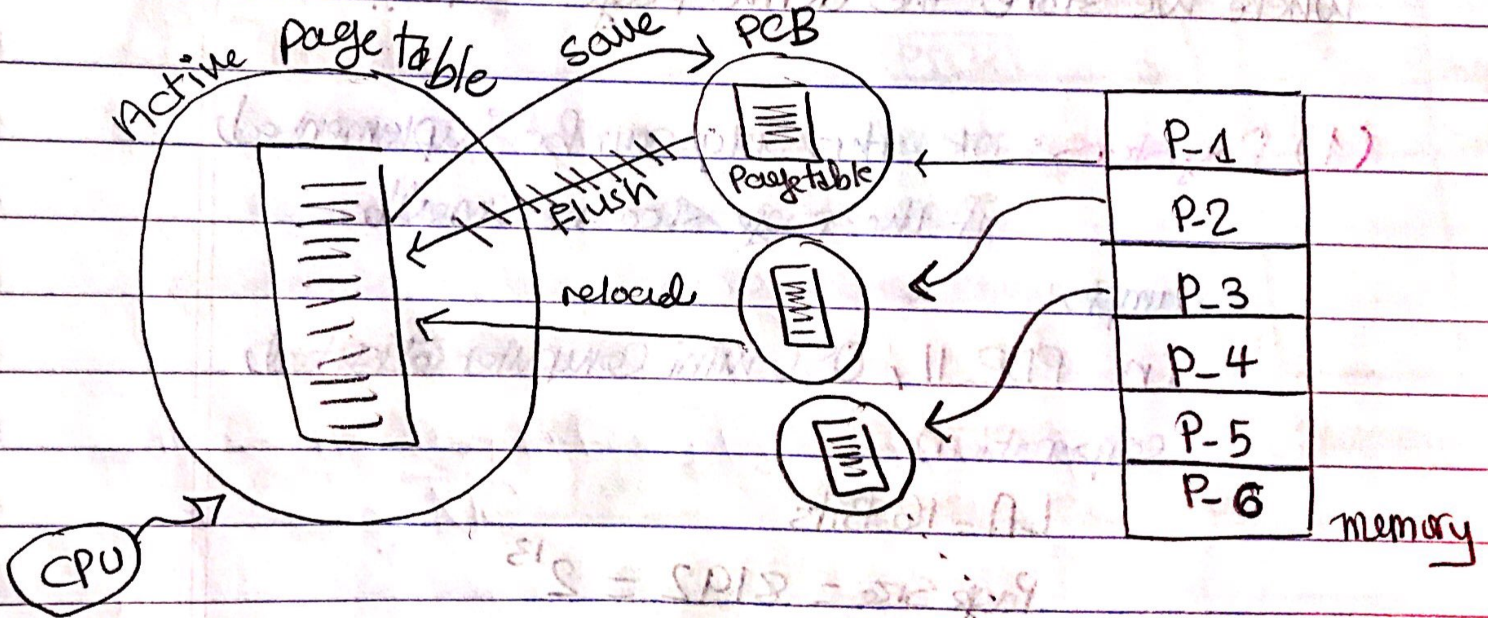


maximum program can be executed on this machine $= 2^{20}$

Page table size $= 2^{20} * 4 = 4MB$

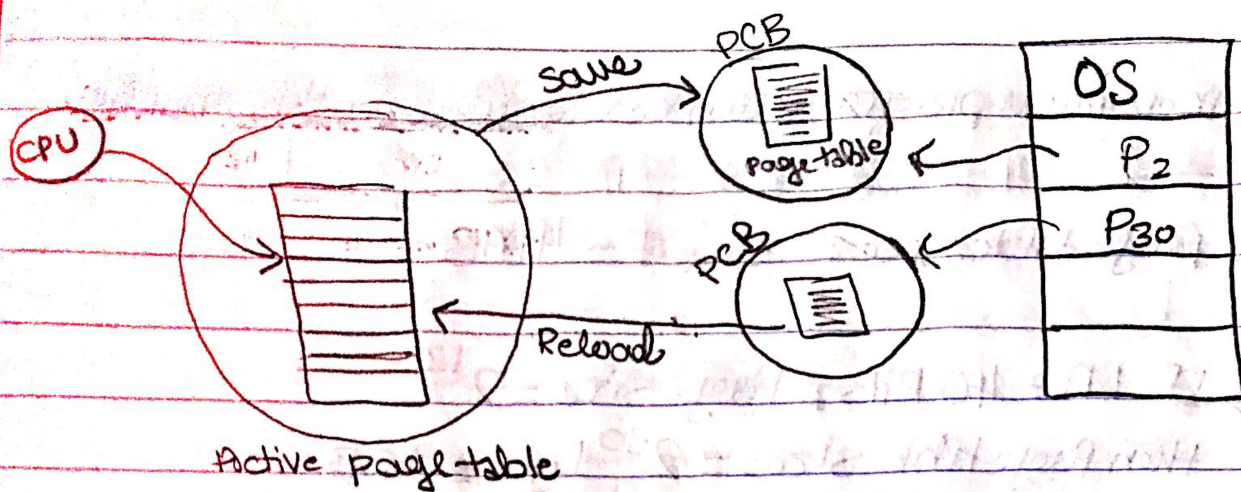
→ If LA = 40 Bits, Page size $= 2^{12}$
then Page table size $= 2^{30}$ byte = 1 GB

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



Where does the OS store the page table?!

the active page table is the page table executing



Active page-table

Implementation of the page table

"Where we store the active page table ?!

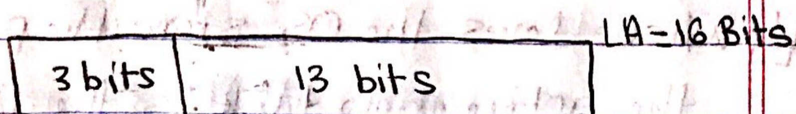
(1) Registers: OK, this only can be implemented if the page size is small.

example:

In PDP_11, OS (mini Computer digital corporation)

LA = 16 Bits

Page Size = 8192 = 2^{13}



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

* Page table size = 2^3 * Frame number size $= 2^3 * 4 = 32$ bytes

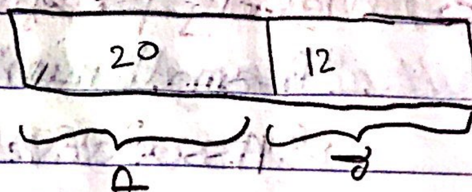
	Logical program
0	
1	
2	
⋮	
7	

Page table
F
F
F
⋮

Logical program   page table

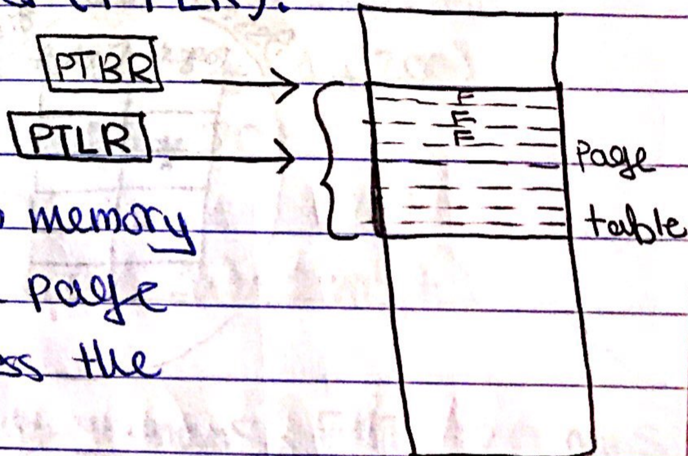
Scanned with CamScanner

But, If LA = 32 bits
page size = 4096 = 2^{12}

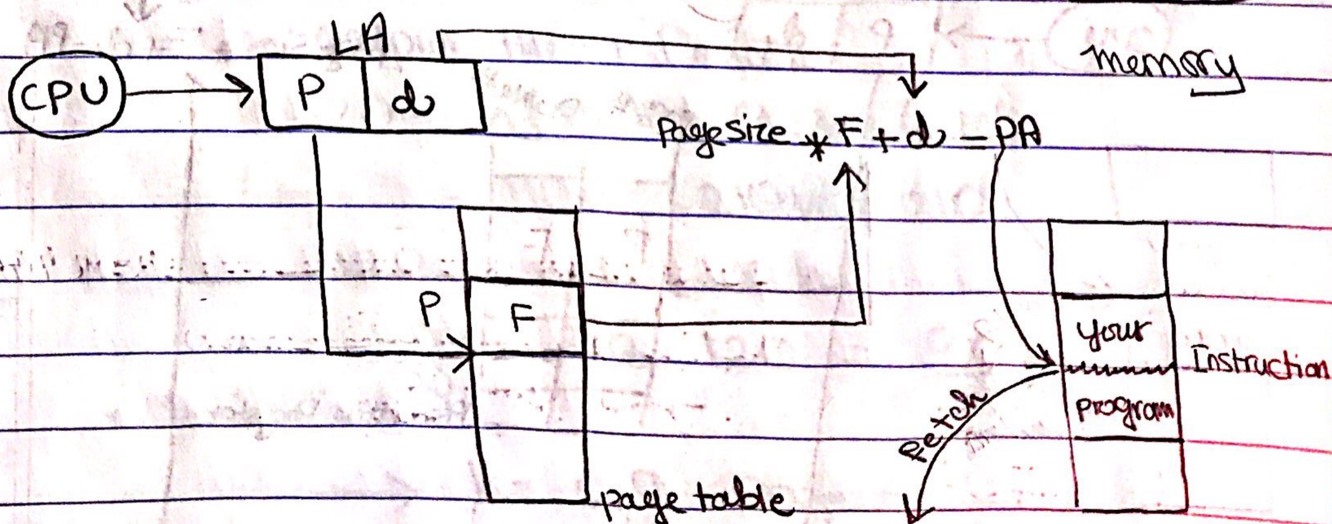


∴ size of page table = $2^{20} * 4 = 4MB$

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

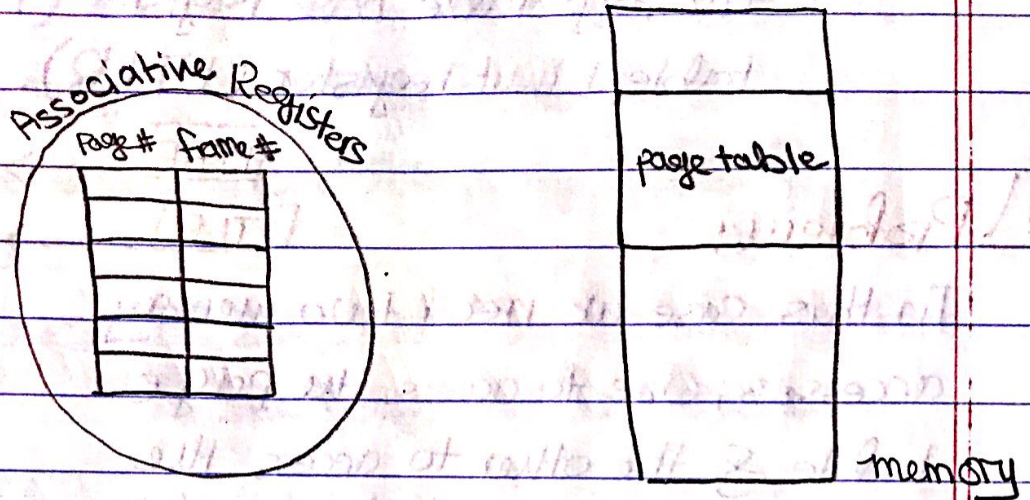


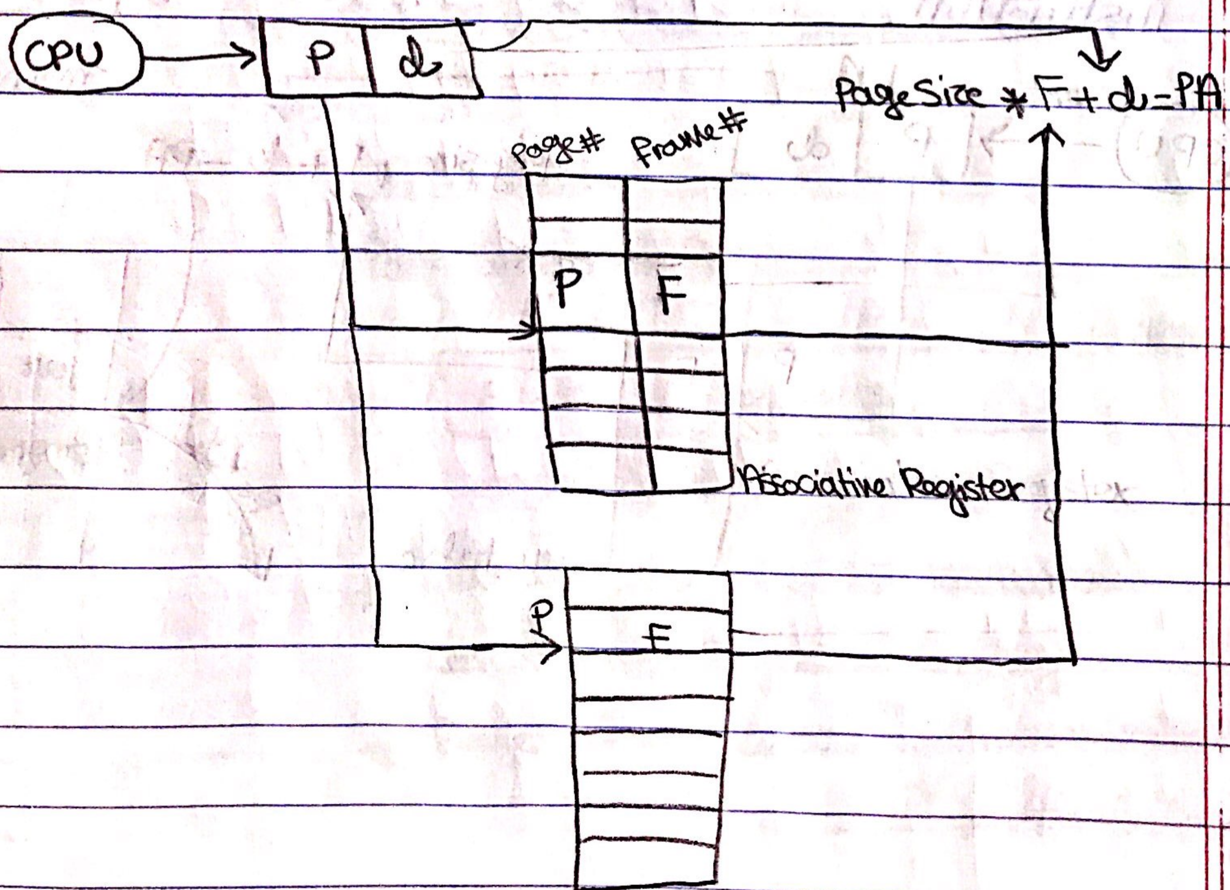
⚠ Problem:
In this case we need two memory accesses, One to access the page table & the other to access the instruction.



(3) Memory + Registers

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





Scanned with CamScanner

- Performance of Associative Register depends on Hit Ratio (h)
- Hit Ratio: probability that the desired page in the associative Register.

example:

① - memory access = 100 nanS → m
- Search time in associative register = 1 nanS → t
- Hit Ratio (h) = 0.95

→ Effective Access Time (EAT)

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

$= 0.95 * 101 + 0.05 * 201$

$= 106$ nanS.

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

② $m = 100$ nanS, $t = 10$ nanS, $EAT = 120$ nanS

compute the hit Ratio:

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

$= 110h - 210h + 210$

$120 = 210 - 100h$

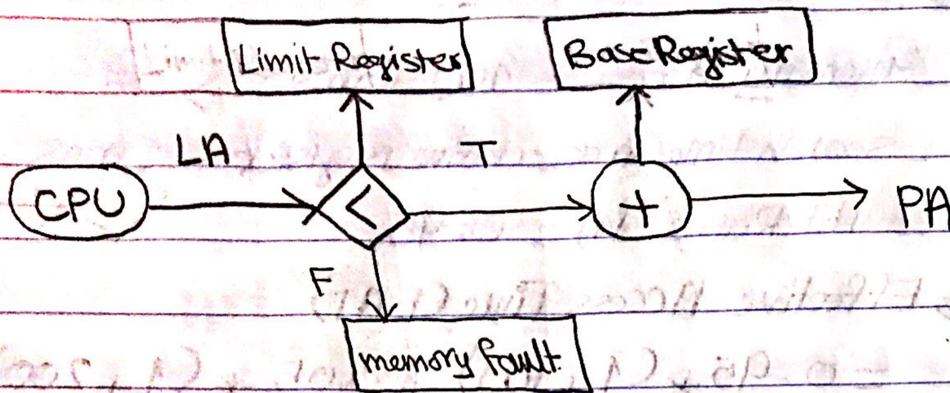
$\frac{100h}{100} = 210 - 120 = \frac{90}{100}$

→ $h = 0.9$ perfect.

Scanned with CamScanner

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

logical program
A
B
C
D

page table	L/IL	R/W
12510	1	1
7415	1	1
8001	1	1
120	0	0
	0	0
	0	0

Scanned with CamScanner

Paging Advantages:

"advantages of sharing pages"

U_1-word

W_1
W_2
D_1

page table

110
102
106

U_2-word

W_1
W_2
D_2

page table

110
102
112

Frame	Contents
100	
101	
102	W2
103	
104	
105	
106	D1
107	
108	
109	
110	W1
111	
112	D2
⋮	

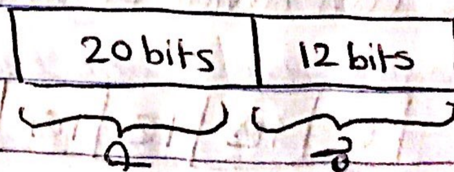
⚠ Disadvantage:

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

Multi-Level page table:

① Assume LA = 32 bits.

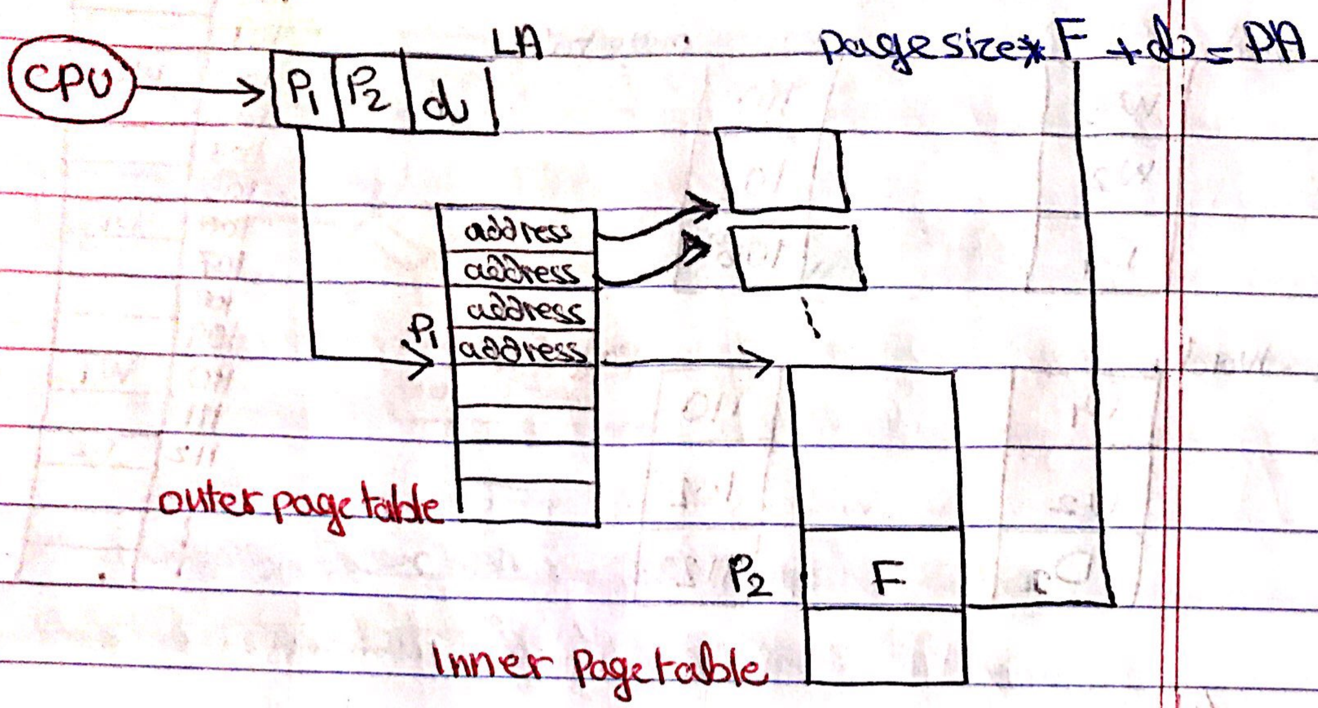
Page size = $4096 = 2^{12}$



∴ Page table size $= 2^{20} * 4 = 4$ MB

→ This is big to store contiguously in memory.

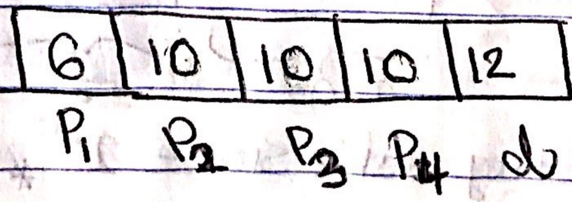
→ Let's divide the page table in to two Levels.



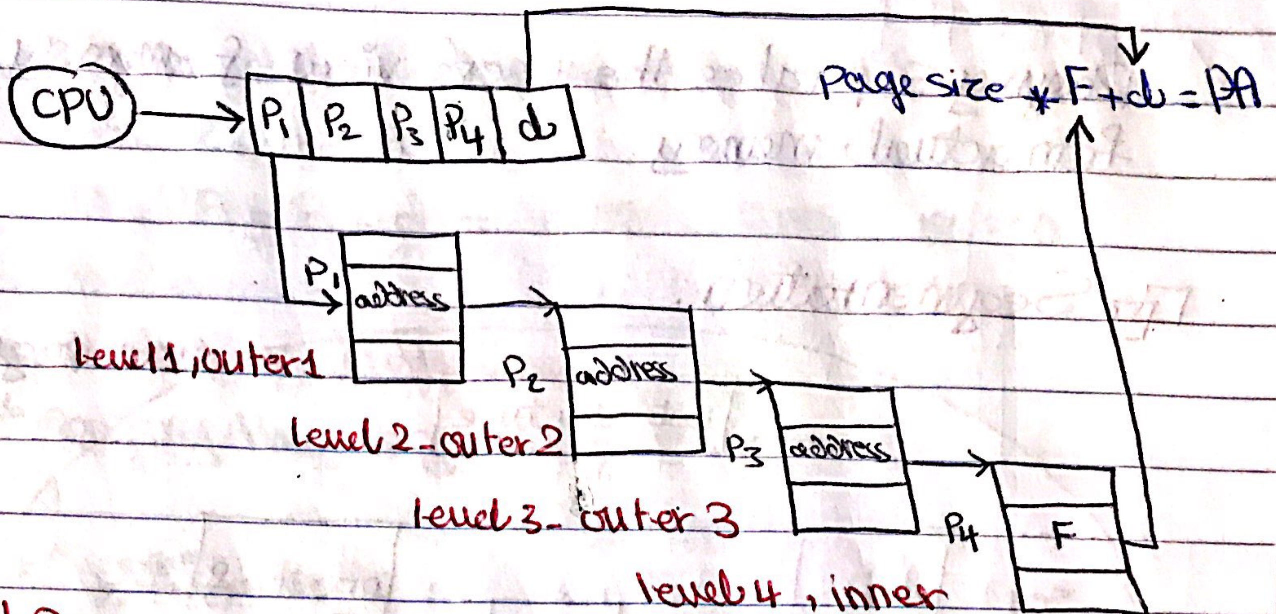
- In our example, assume $P_1 = 8$ bits, $P_2 = 12$ bits
- outer page table size $= 2^8 * 4 = 2^{10} = 1KB$
- Inner page table size $= 2^{12} * 4 = 2^{14} = 16KB$

(2) LA _ 48 bits, Page Size $= 2^{12}$ bytes

OS can divide LA in to 4 levels



Scanned with CamScanner



⚠ 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.

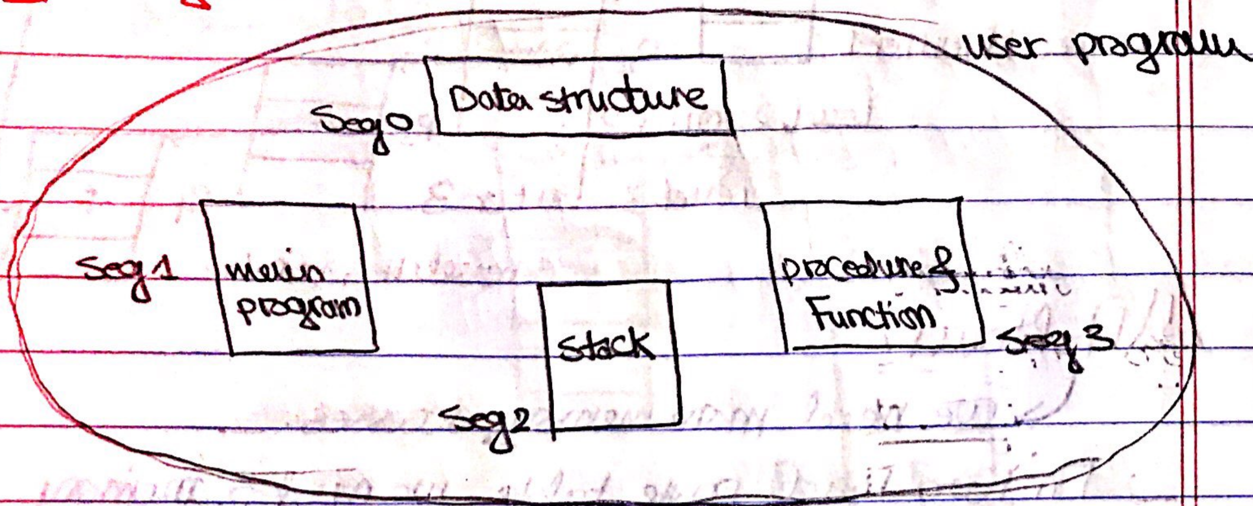
example: assume,
- memory access time = 100 nans
- associative register search time = 1 nans
- assume 4 levels of page table with hit ratio (h) = 0.98

$$EAT = 0.98(100+1) + 0.02(500+1)$$
$$= 0.98 * 101 + 0.02 * 501 = 109$$

Scanned with CamScanner

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

Segmentation:



	Segment Base Register	segment Length
0	27,000	7,000
1	1,000	5,000
2	18,000	3,000
3	21,000	2,000

Segment table

memory

memory	
	1,000
Seg 1	5,000
	18,000
Seg 2	3,000
	21,000
Seg 3	2,000
	27,000
Seg 0	7,000

S: Segment #

d: LA offset.