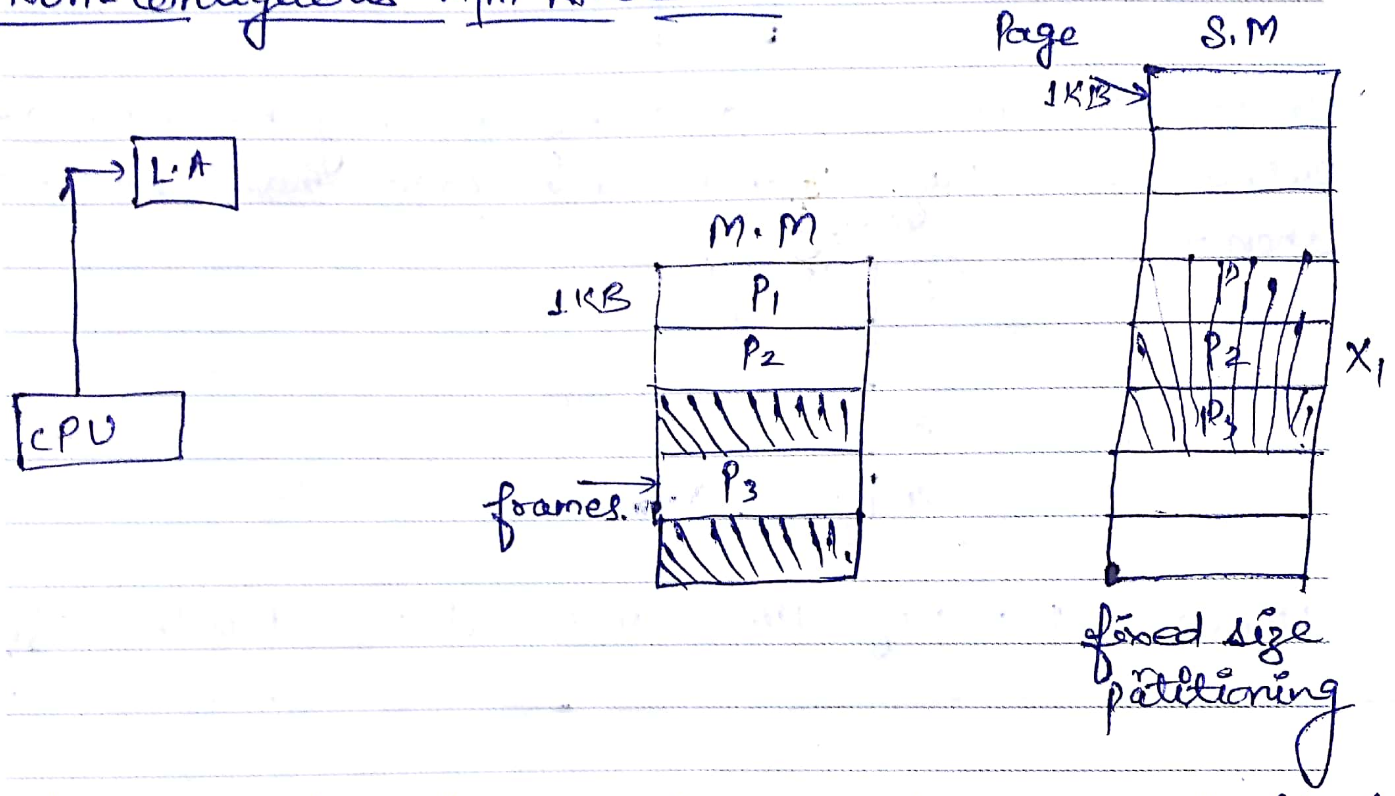
L.A for S.M but main m/m access by lesser time.

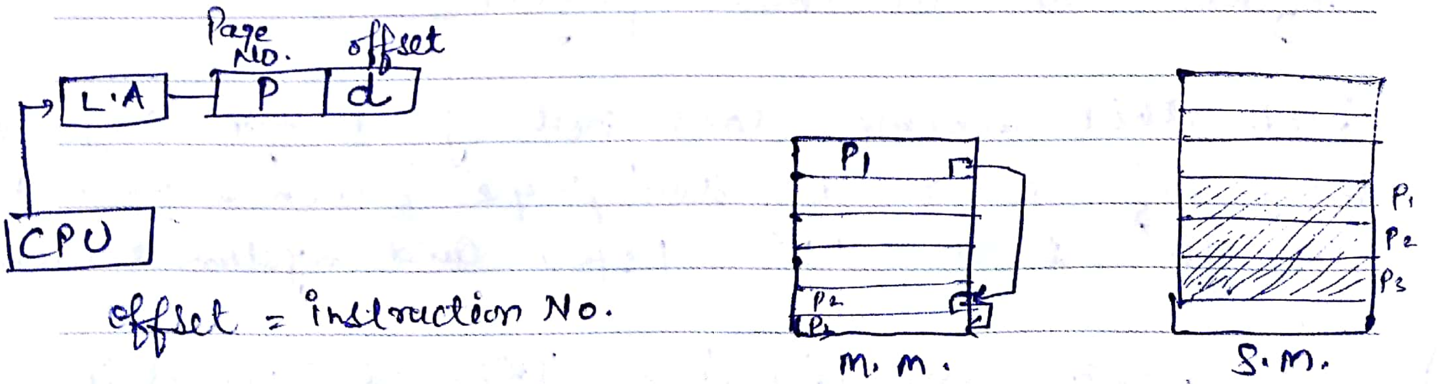
UNIT-4.

Virtual m/m :-

Non-Contiguous m/m Allocation :-



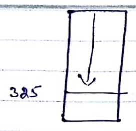
whereever the pages are empty we put that page into main m/m.



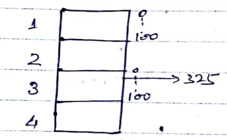
offset = instruction No.

for every process the base address of first page contains a pointer to next page.

like we say-



Instruction no. 325 and if we say we have distributed the pages and each page has 100 instruction then,-



Hence p ⇒ page No. and d = instruction offset.

So in non-contiguous the base address of first page will be stored and the pointer will define the another pages.

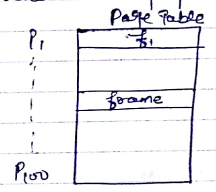
But this scheme does not good for paging. becz if we need 200th page, hence we need to go 1 to 199th page. and system become slow.

So next solution is - Indexed Pagetable.

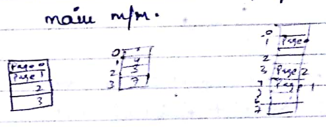
Paging → Suffers from Internal fragmentation
→ Instruction speed ↓ (Space Utilization↑, but Access Time↓)

A Page Table is a data structure not a H/W and it contains the no. of entries equal to the no. of pages, a process having in secondary memory.

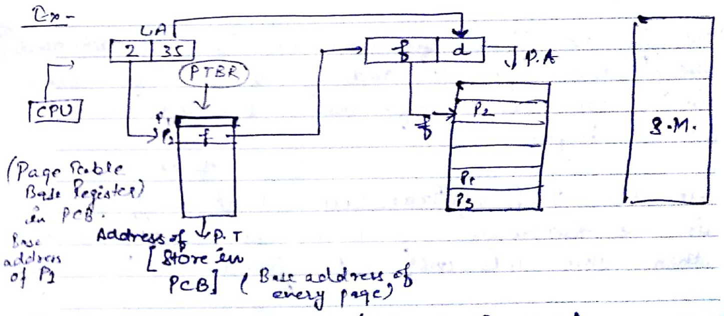
Every process has a independent page table.
Suppose 100 pages in S.M so we have 100 entries in page table -



that means each page has frame No. where that page is stored in main m/m.



Ex-

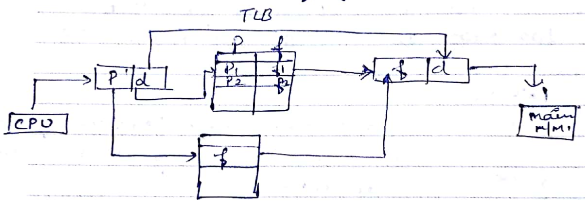

LA
2
35
CPU
PTBR
P1
P2
f
(Page Table Base Register) in PCB.
Base address of P1
Address of P.T [Store in PCB] (Base address of every page)
f
d
P.A
P2
P1
P3
S.M.

★ Access very fast (Non-Contiguous)

★. Independent data structure which take heavy penalty (metadata).

★ Every process has page table. and address of page table is stored in PCB.

★. No. case external fragmentation.



Becz of paging concept we have two-times access of main m/m (Page table + Actual Instruction).

Now for the first ~~f~~ time we will go each and every time for P_1 and frames no. for same in Page Table. ∵ for the instruction no. in a same page.

We use TLB (Translation look Aside Buffer) is a hardware. first it is empty and then this TLB will get filled. But costly.

Every process doesn't have TLB. hence if context switch, all the entries will be deleted.

Numerical ⇒ MM = 64 MB , LA = 32 bytes PS = 4KB.

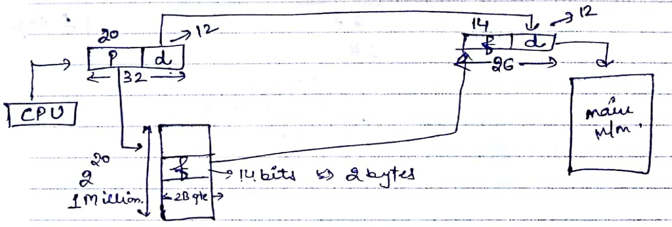
Page Size

Total space wastage (size) in maintaing a page table?
System is byte addressable.

$\frac{64\,MB}{\cancel{1\ byte}}$ ⇒ No. of locations.

$64 \times M = 2^6 \times 2^{20} \Rightarrow 2^{26}$ locations.

So no. of bits ⇒ 26 for addresses.



Page size ⇒ 4KB each page has entry of 1 ~~K~~B.

$\frac{4\,KB}{1\,\cancel{K}B} \Rightarrow 4K = 2^2 \times 2^{10} = 2^{12}$

12 bits required for offset.

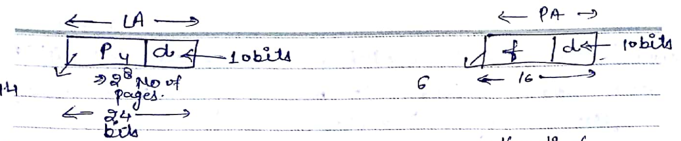
Hence

$2^{20} \Rightarrow 1\,M$

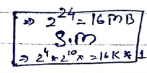
2 Byte ⇒ 2 B

⇒ 1 M × 2B = 2 MB wastage.

LA = 24 b   PA = 16 b   PS = 1KB.



ex-



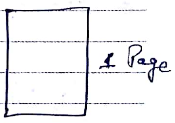
$\Rightarrow 2^{24} = 16MB$
S.M
$\Rightarrow 2^4 * 2^{10} * = 16K * 1 KB = 16MB$

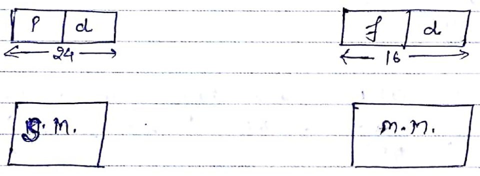
size of S.M = 16 MB.

$2^{16} \Rightarrow 2^{10} \cdot 2^6$
$\Rightarrow 32K$
M.M.

Size $\Rightarrow$ 32K * 1B
= 32KB.

Page size $\Rightarrow \frac{1KB}{1B}$
$= 1K = 2^{10}$
(10bits)
for locate any Page No.
$\Rightarrow$



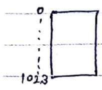


Assignment $\Rightarrow$ ①:- Disadvantages of TLB.
②- Difference b/w paging and segmentation.

$2^{10} = 1K$ $2^{40} = 1$ Terra.
$2^{20} = 1M$
$2^{30} = 1G$

Address to Space Translation –   n bits = 2^n combinations.

I have a address if it $\Rightarrow$ 10 bits $\Rightarrow 2^{10}$ ~~bits~~ = 1024 address locations



locations can be addressed.

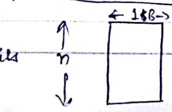
And generally, it is said to be size of each page is 1 B = 8 bits. (If not given).

Let suppose a address is of 'n' bits then what is the size of m/m? - Depending on no. of location * size of each page (location).
$\Rightarrow 2^n * 1B \Rightarrow 1024 * 1B$
$\Rightarrow 1K * 1B$
$\Rightarrow$ 1KB of m/m.

← 14 →
(1B) $\Rightarrow 2^{14} * 1B = 2^{10} * 2^4 * 1B \Rightarrow 16KB.$

← 22 →
(2B) $\Rightarrow 2^{22} * 2B = 2^{10} * 2^{10} * 2^2 * 2B \Rightarrow 8MB.$

Space to Address Translation – Given m/m size = 64KB



$= 2^{16}$

MS = n * 1B   ~~$\Rightarrow 64 * 2^{10}$~~
64KB = n * 1B
$n = \frac{64KB}{1B}$ $= 2^6 * 2^{10}$

Advantage of TLB. Access Time

MM = 400 μs , TLB = 50 μs. h = 90%

If NO TLB, then time ⇒

⇒ 2 * 400 μs = 800 μs.

If TLB exists ⇒ $\overset{\text{If hit}}{0.9\,[50+400]} + 0.1\,[\overset{\text{TLB miss}}{50} + \overset{\text{P.T}}{400} + 400]$

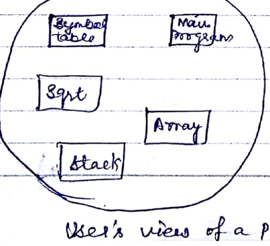
⇒ 450 × 0.9 + 850 × 0.1

= 405 + 85

= $\underline{\underline{490\ \mu s}}$

Segmentation :- Users prefer to view m/m as a collection of variable-sized segments with no necessary ordering among segments.

In a program you use functions, methods, array and these data elements is referred to by name. Users don't bother about that symbol table is stored at what address or before 'sqrt function'.



User's view of a program.

Each segment's element is identified by their offset from beginning of segment.

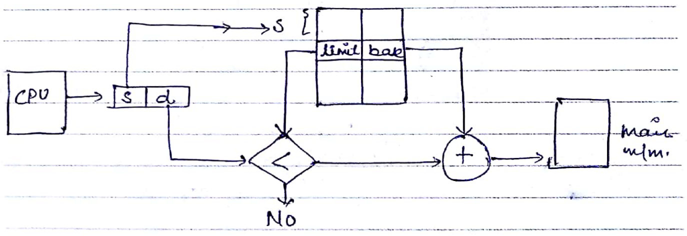
It is a scheme of m/m mgmt. that supports this user view of m/m.

A LA space is a collection of segments. Each segme
has a name or no. and a length.

A logical address consist two tuples –

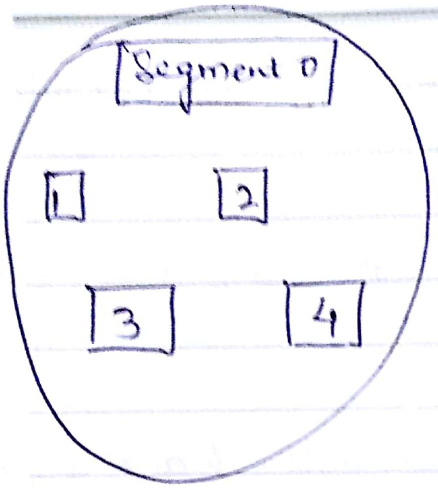
< segment-number , offset >

Compiler automatically structs segments reflecting the I/P programs.

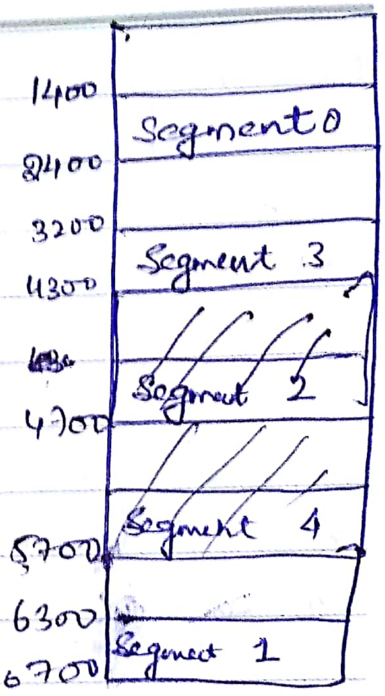


Segmentation h/w.

Scanned by CamScanner



	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700


1400
Segment 0
2400
3200
Segment 3
4300
Segment 2
4700
Segment 4
5700
6300
Segment 1
6700