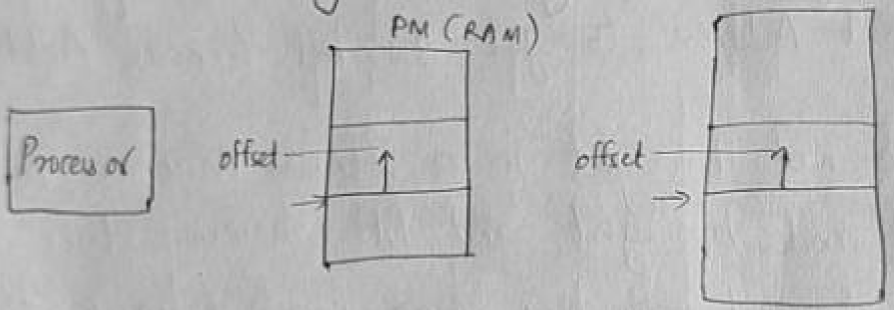
Subject: Microprocessor Semester: IV

80386 Address Translation – Paging

Paging is the second stage of Address Translation.
Paging converts a linear address to a physical address.
Linear address is of 32 bits. The physical address that is obtained after translation is also of 32 bits. Why are we converting a 32 bit no into another 32 bit no?



Originally, there was some file (segment) in virtual mly which started at an address and there was an offset which we wanted to access.
When a segment is copied from VM to PM, the offset is not changing. So you were assuming that your addresses are in line and hence the name Linear Address. But that's not how it is. A segment from VM is never brought to PM in one piece (in linear form). Because you will not have so much space continuously available in PM.

Subject: Microprocessor Semester: IV

The segment is brought in parts called pages and these pages are loaded in PM

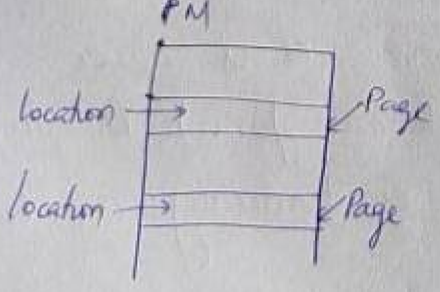
So now we realize that offset is basically some location in some page. Earlier we assumed that there was no paging and so we assumed that the entire segment was copied in physical memory. So we were adding Base Address and Offset Address to get 32 bit Linear Address

Pages are not brought in a linear manner. They are shuffled and brought and fit wherever slots are available. So the desired page has gone somewhere in the PM and that address is to be calculated.

So the linear address (32 bit) is converted into physical address (32 bit).

Linear address is divided into 2 parts.

Page No	Location



In PM, pages are shuffled but a location (say location 10) will remain location 10 in the page. So only base address of the page is changed.

So when we convert

Linear Address | desired page no | location |
to
Physical Address | final page no | location |

location remains same

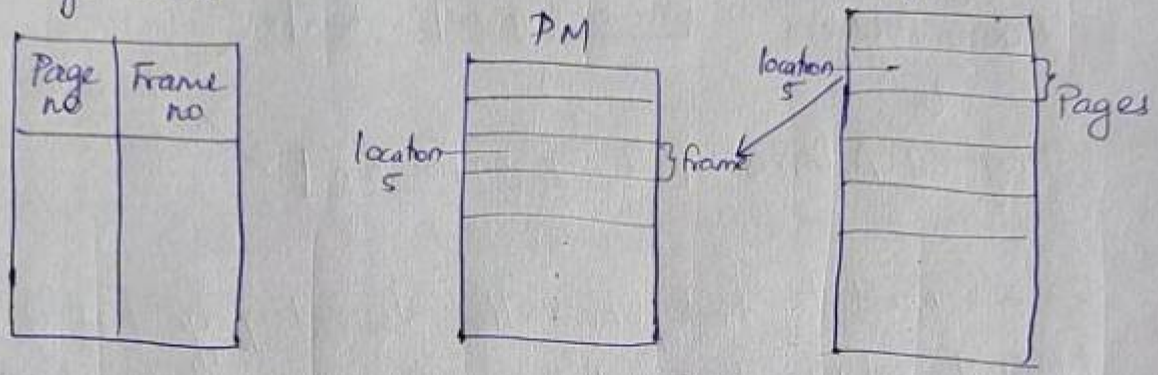
So we need to find where the desired page has gone in physical mly ie the relation b/w desired page no. and final page no. This is stored in a table called the <u>Page Table</u>.

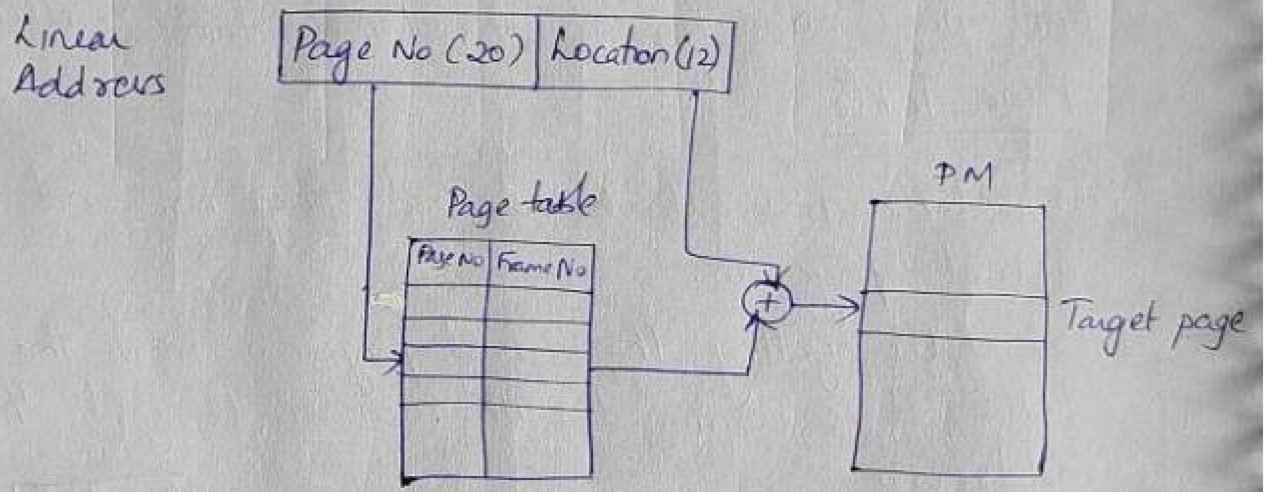
Page table has all the desired page no. and their actual page no.
So after getting the actual page no. from the page table, it is combined with location to get the physical address.

Desired page no.	Actual Page no

Virtual mly is divided into <u>pages</u> of <u>equal size</u>.
Physical mly is divided into <u>page frames</u>.
So page no. becomes the frame no, but location remains the same. So to find out which page no. has gone to which frame no, the page table is to be looked up. In the page table, there is an entry for every page.

Page table





In the page table, there are entries for every page.

How many pages are there?

The max. size of PM = 4GB $\left(\begin{array}{c} \text{32 bit address bus} \\ 2^{32} = 4\text{GB} \end{array} \right)$

A page is of size 4 KB

So total no of pages $= \frac{4\text{GB}}{4\text{KB}} = 2^{20} = 1$ Million pages.

So there are 1 Million entries in the page table.

Subject: Microprocessor Semester: IV

A page is of 4 KB

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

So location is of 12 bits

	(20)	(12)
LA	Page No	Location

20 bits gives the page no.

2^{20} = 1 Million pages.

Since the page table is having 1 million pages, to find 1 entry the entire 1 million entries will have to be scanned which takes a long time.

So the page table is organized such that there are multiple page tables and each page table has 1K pages.

How many page tables?

Total 1 million pages

Each table has 1K pages.

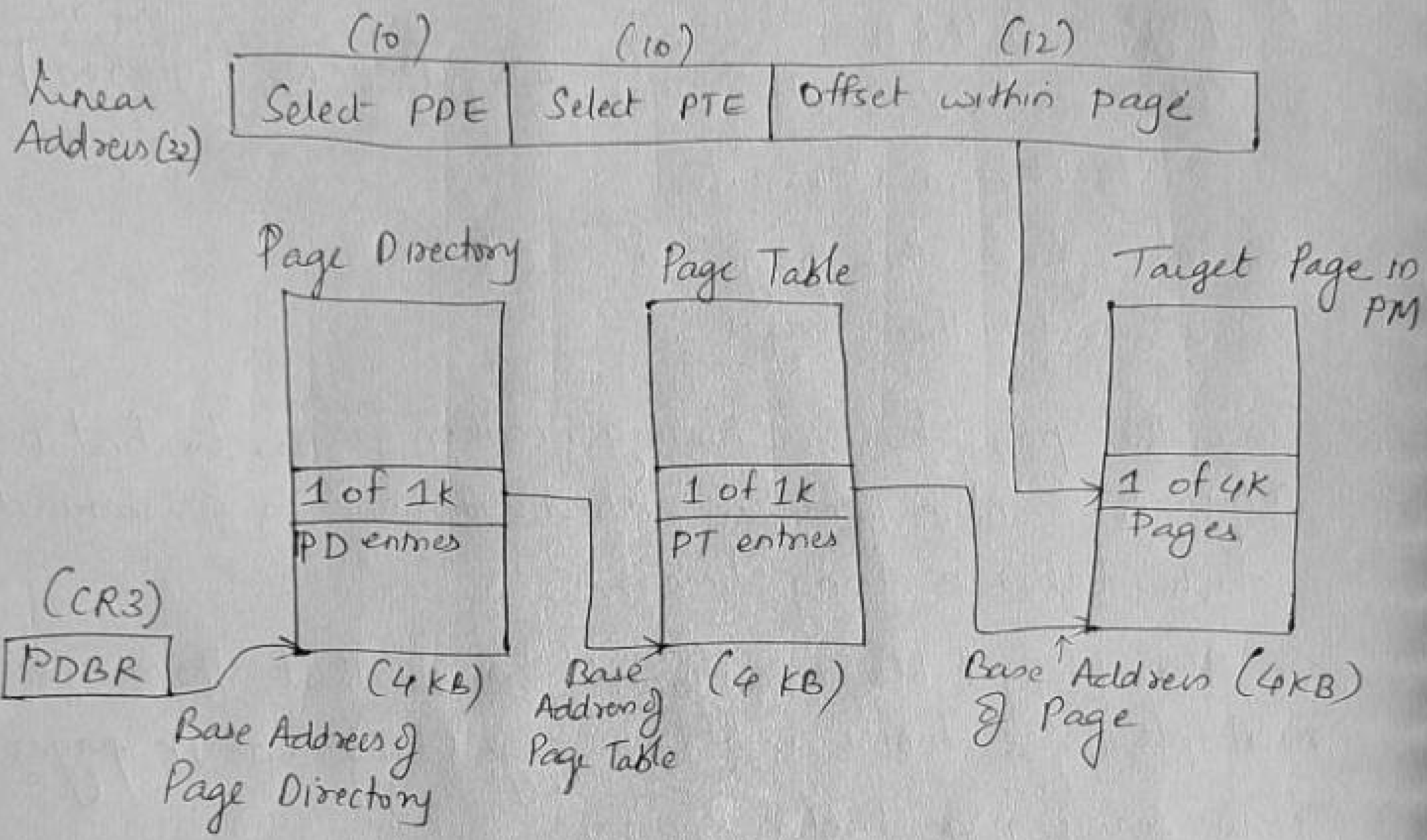
So $\frac{1M}{1K} = \frac{2^{20}}{2^{10}} = 2^{10} =$ 1K Page Tables

each having 1K page entries

So

	$\overbrace{\quad\quad\quad\quad}^{32}$		
LA	PDE(10)	PTE(10)	Location (12)
	Identifies page table within page directory	Identifies page within page table	Identifies location within page

Subject: Microprocessor Semester: IV



There are 1 million pages. Entries for the first 1k pages (Page Table Entry) are in the first page table, next 1K page is in the 2nd page table and so on.

All these 1K page tables are also scattered in mly. So base addresses of these page tables are given by the Page Directory (Page Directory Entry)

The base address of the Page Directory is given by Control Register 3 (CR3)

Page table → gives base address (32 bits) of the page
$\downarrow$ 4 bytes

Page table has 1K entries.
So size of page table = 1K × 4 bytes = <u>4 KB</u>

Page Directory ⇒ gives base address (32 bits) of the page table.
$\downarrow$ 4 bytes.

Page Directory has 1K entries for 1K page tables.
So size of page directory = 1K × 4 bytes = 4 KB.

Page 4KB, Page Table 4KB, Page Directory 4KB } All same size. Why?

Ans:- 1) Page Table & Page Directory has to be 4 KB. If it is more than 4 KB then page table will also be split into pages and then there should be some entity to hold that info.

If it is of 4 KB
the 1st page will begin at location 0.
next page will begin at location 4K.
next page will begin at location 8K
12K

Microprocessor Semester: IV

means every page will begin at a location which is a multiple of 4K.

If it is a multiple of 4K, then the last 12 bits are 0.

Anything multiple of 2 $(2^1) \rightarrow 10 \rightarrow$ last bit 0
Anything multiple of 4 $(2^2) \rightarrow 100 \rightarrow$ last 2 bit 0
8 $(2^3) \rightarrow 1000 \rightarrow$ last 3 bit 0

4K $(2^{12}) \rightarrow$ last 12 bit 0.

So a page table entry (PTE) need not give a 32 bit address as the last 12 bits are 0. It needs to give a 20 bit address (32-12) and the 12 bits can be used for other information like P bit, D bit, privilege info etc.

PTE

Base Address of Page	. . .	D	A	. .	U/S	R/W	P

D → Dirty bit – tells whether a page has been modified or not.

A → Access bit – tells whether the page has been accessed for long or not.

U/S → User or Supervisor
(PL3) (above PL3)

R/W → Read or Read & Write

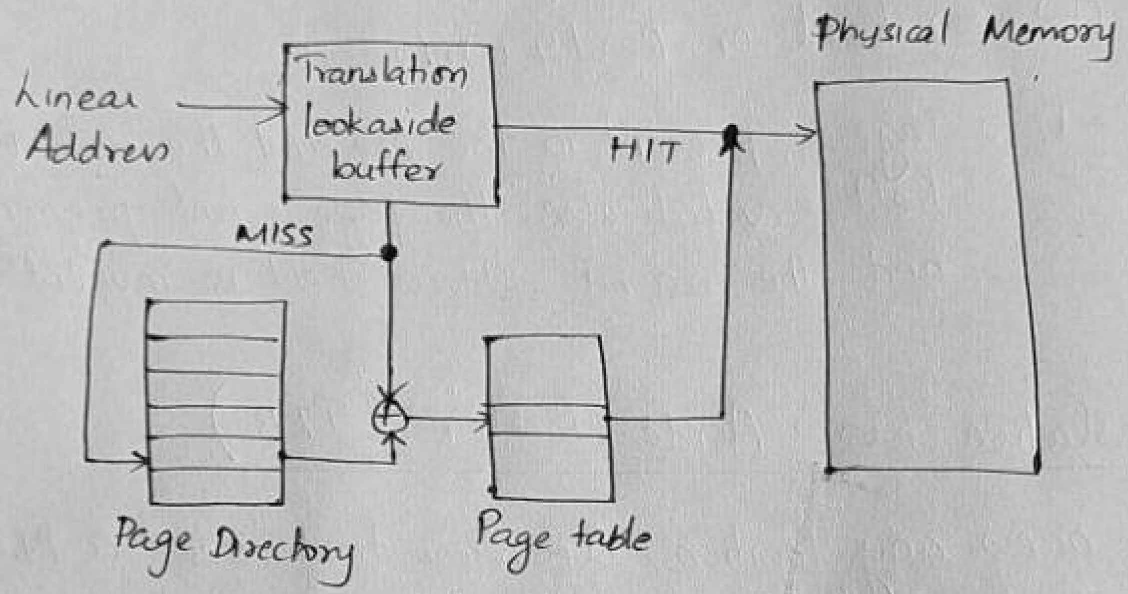
P = 1 → Page is present in the PM and the 20 bit address field is valid else the page is not present in PM and the 20 bit address field is invalid.

Translation Look Aside Buffer (TLB)

To access any location, µP must first access a PDE in the page directory, then a PTE in the page table, then access the page. This can make the process very slow. To speed up the process, an on-chip cache called "Translation Look-aside Buffer (TLB) is used.

TLB is an on-chip cache which stores 32 most recently used PTEs and PDEs. This makes subsequent access to these pages (whose information is cached in the TLB) much faster as there is no need to access the page directory and the page table. µP can directly obtain the starting address of the page frame from the TLB and hence directly access the page.

Due to principle of "Locality of Reference" most systems get a hit ratio of > 98% on the TLB, making the operations very fast.


Physical Memory
Linear Address
Translation lookaside buffer
HIT
MISS
Page Directory
Page table