Academic Year: 2022-23 **Semester: IV**
Class/Branch: SE **Subject: Microprocessor**

80386 Address Translation:

Segment translation (First half of address translation).

Segment translation converts 48 bit Virtual Address into 32 bit Linear Address.

Note:- The total 48 bit address comprising of 16 bit segment address and 32 bit offset address is called Virtual address.
Virtual address is translated into a 32 bit physical address using 2 translations: Segment translation (compulsory)
Paging (optional)

If paging is not implemented, then the 32 bit linear address is the final 32 bit physical address.

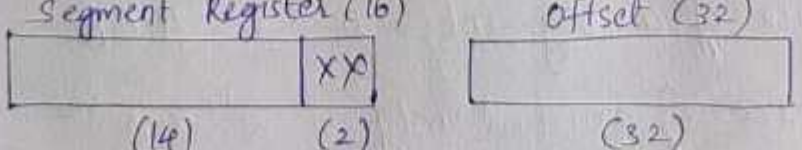
For eg.

In 8086, MOV CL, [2000]
↑ offset address.

In 8086, offset address is 16 bits.

In 80386, MOV CL, [00002000]

In 80386, offset address is 32 bits.

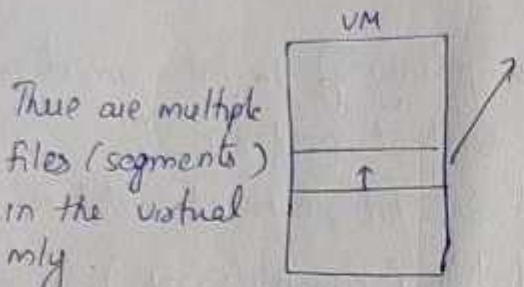


32+14 = 48 bit virtual address.

Academic Year: 2022-23 | **Semester: IV**
Class/Branch: SE | **Subject: Microprocessor**

Virtual memory means- the maximum accessible mly of a processor implemented using secondary storage devices like Hard disk.

Think of the files in hard disk as segments.



So the maximum no. you can put in the segment register is the maximum no. of segments you can have.

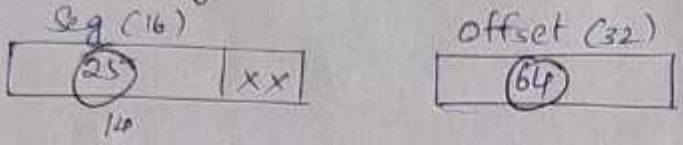
The maximum no you can put in the offset register is the maximum size of a segment.

Segment Register is 16 bit, but out of the 16 bits, 2 bits dont take part in address translation. 2 bits are used for specifying priviledge level (used in protection mechanism)
So there are 14 bits.

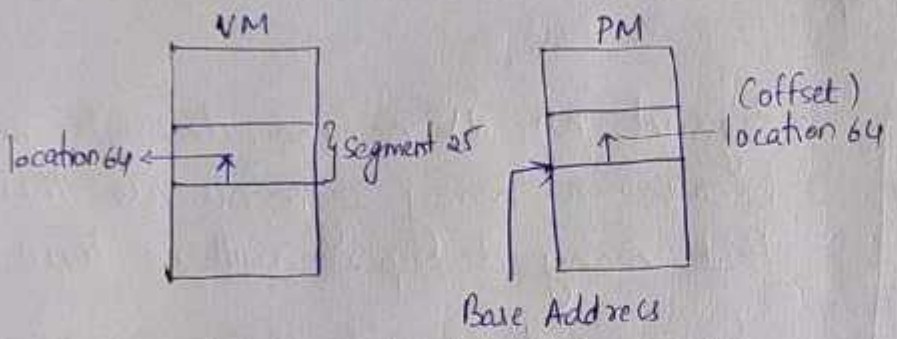
* So maximum no. of segments = 2^{14} = 16 K
* Maximum size of a segment = 2^{32} = 4 GB
* Maximum size of virtual mly = $2^{14} \times 2^{32} = 2^{46}$ = 64 TB

Academic Year: 2022-23 | **Semester: IV**
Class/Branch: SE | **Subject: Microprocessor**

When we access a file in hard disk, it gets copied into RAM and it is played / executed from the RAM.



Suppose I want to access location 64 of file 25



Note:
Location 64 will remain location 64 when the file is copied from VM to PM. This means during this translation offset address will not change.

So what is to be known is the base address of my segment in PM.

The base address of any segment is given by an object called Descriptor.

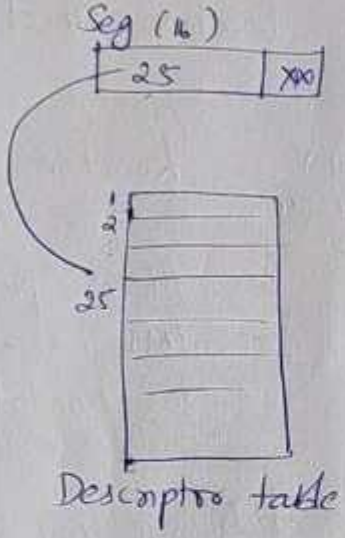
Every segment has a descriptor.

Every file has a descriptor. If there are 4000 files in your computer, there are 4000 descriptors present in your computer.

So in segment translation, descriptor gives the base address, to which offset address is added to get the linear address of the location.

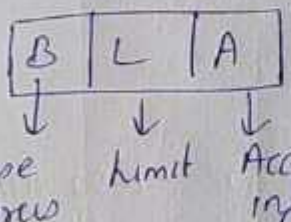
Academic Year: 2022-23 | **Semester: IV**
Class/Branch: SE | **Subject: Microprocessor**

As every segment has a descriptor, there are 16k descriptors possible. All those descriptors are stored in a table called Descriptor Table.



Segment no 25 means descriptor no 25 because as many segments are there, that many descriptors will be there.

So segment no acts as a SELECTOR or index. It selects the target segment descriptor, gets the base address from the descriptor, adds offset to it to get the target location.



Descriptor also gives LIMIT. Limit means the size of the segment. Limit is used to ensure that the offset address does not go beyond the limit. If it goes beyond limit it means that you are trying to access a location which does not exist in the segment.

Academic Year: 2022-23 **Semester: IV**
Class/Branch: SE **Subject: Microprocessor**

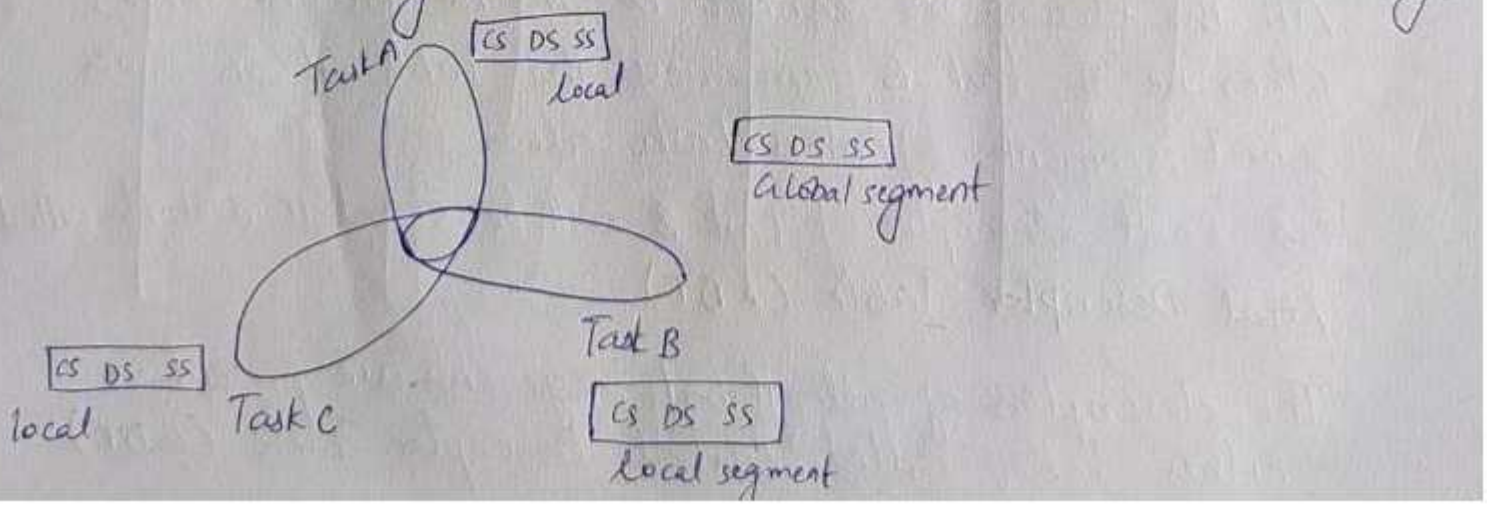
Note :- The maximum size of a segment is 4GB. Not all segments are of 4 GB, there are smaller segments also. So if the offset address is within the limit access will be given, otherwise access is denied.

In the segment register, 2 bits are for priviledge level. They give the priviledge level of the program that you are writing ie the Requester Priviledge Level.

In the descriptor, there will be the priviledge level of the segment.

Segment priviledge level is compared with the requester priviledge level. If the requestor has equal or greater priviledge level, then access will be allowed, otherwise not. All these enforce protection in 80386.

Suppose there are 3 tasks running simultaneously in a multitasking environment and these tasks are switching



Academic Year: 2022-23 | **Semester: IV**
Class/Branch: SE | **Subject: Microprocessor**

Task A → I am going to whatsapp you { Receive message }
Task B → You are going through your photos.
Task C → You are listening songs.

When I am going to whatsapp you, should I know, which song you are playing? NO

Mean the data of one task should never be available to the other task.

So every task has its own local segments.

Also there are some programs and data which are available to all task eg: current system time, date etc. All these data is stored in global segment.

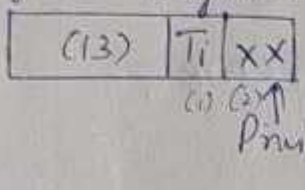
Lets say when Task B is running it has access to
→ global segment
→ its own local segment.

All the descriptors cannot be present in one table, otherwise if task B can access the table, it can access local segment of other tasks also.
So local descriptor of task B will be stored in a table called Local Descriptor Table (LDT).

The descriptors of all global segments are present in another table called Global Descriptor Table (GDT)

So when you switch from task A to task B, GDT will remain constant but LDT keeps changing.
Every task has its own personal LDT.

Segment Register (16)



We have 2 types of descriptor tables - LDT & GDT.
Whether you want to access LDT or GDT will be given by a bit called Ti present in the segment register.

Ti (Table Identifier)

Ti = 0 → GDT
= 1 → LDT.

So only 13 bits act as an index/selector in the descriptor table

Both GDT and LDT have 8K descriptors.

There are 16K possible segments. Segments can either be global or local segments.
So 16K is divided exactly into half. So there are 8K global segments and 8K local segments.
8K global segments are available to all tasks.
8K local segments are divided amongst various tasks (not equally)

Academic Year: 2022-23 **Semester: IV**
Class/Branch: SE **Subject: Microprocessor**

Say Task B may need only 1 local segment
Task C may need no local segment
Task A may need 1000 local segments

So now 13 bits are working as SELECTOR.

$2^{13} = 8K$

So 13 bits are enough to select a descriptor out of 8K descriptors

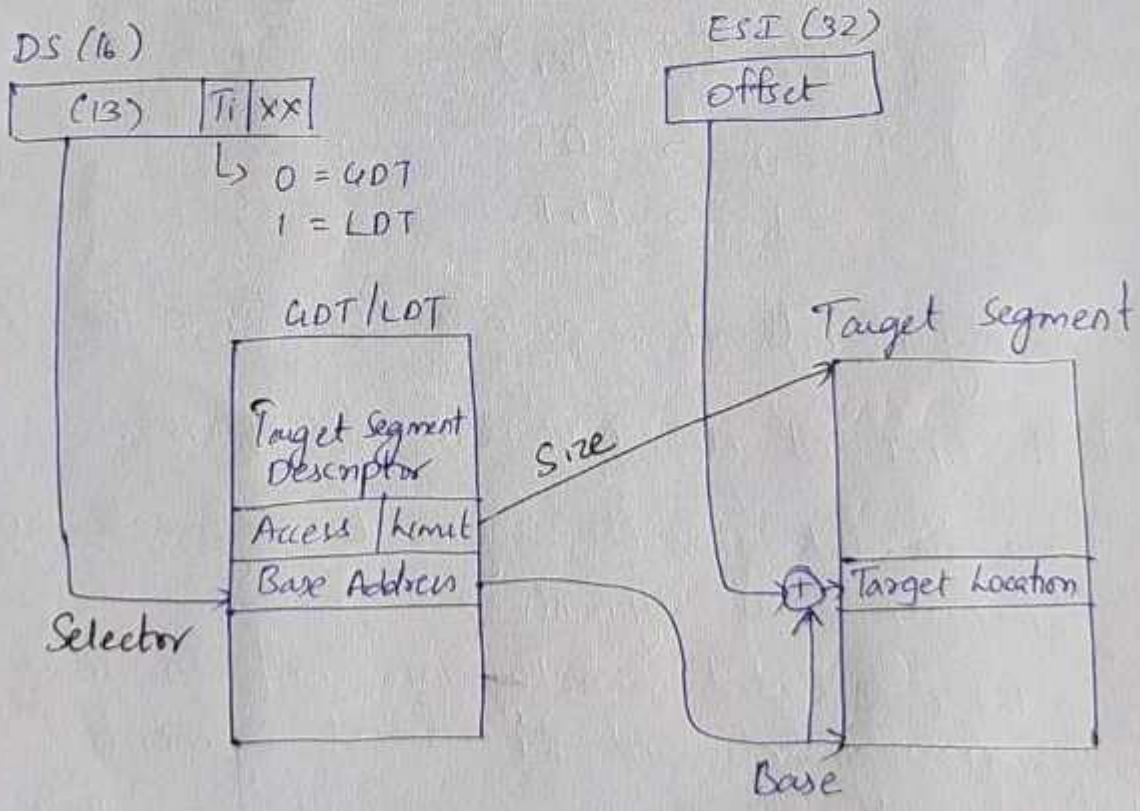
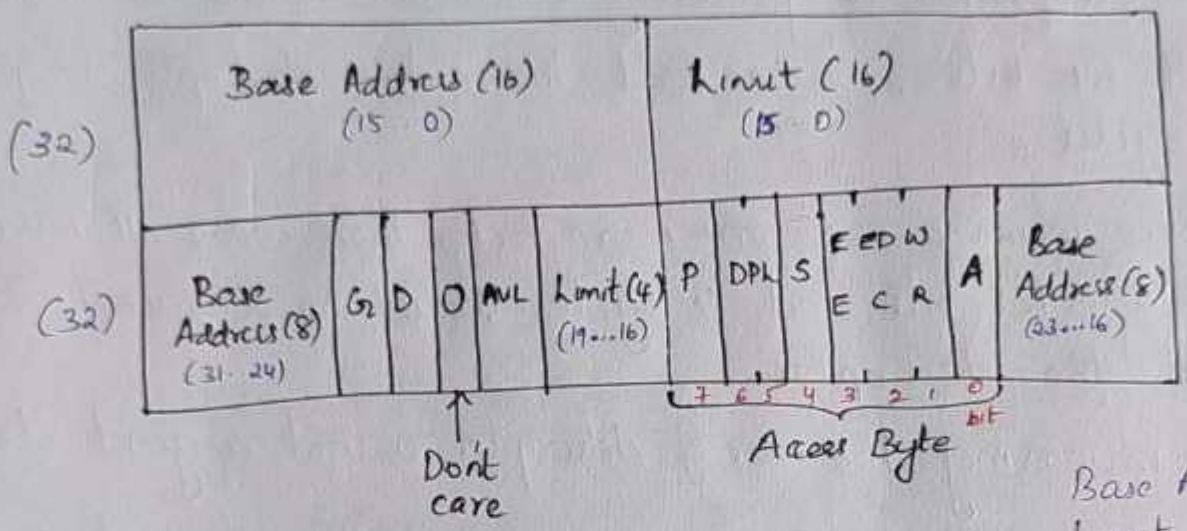


Figure:- Segment Translation

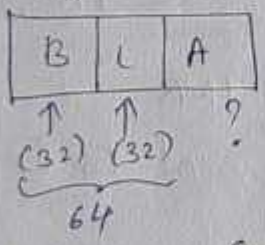
Academic Year: 2022-23 | **Semester: IV**
Class/Branch: SE | **Subject: Microprocessor**

Segment Descriptor Format : (64 Bit)



Base Address - 32 bit
Limit - 20 bit

A segment descriptor is of 64 bits



Base Address → Gives you the starting address of the segment in physical mdy. It has to be of 32 bits as it is in physical mdy.

Limit → gives the size of your segment. Limit is compared with offset address and offset address is of 32 bit. Since the max. size of a segment is 4GB, the limit also should be 32 bit.

So Base Address - 32 } 64 What about access info.?
Limit - 32

If you say increase the size of the descriptor, then the next compatible to powers of 2 number would be 128.

With 64 bits, we can read the entire descriptor in just 2 cycles.

If we increase even one more bit, then we will have to use 128 bits and 4 cycles would be required to read the descriptor.

So increasing the size of descriptor is not a good idea.

Somehow, space is to be made for storing access information.

So what they have done is that limit is only 20 bits.

With 20 bits → Max limit is $2^{20} = 1M$.

So maximum size of segment = 1MB.

But we want max. size of segment to be = 4 GB

Solution:- Multiply the limit by 4KB.

But it is not always multiplied by 4 KB. What if a segment is not a multiple of 4?

So there is a bit called G (Granularity)

G = 0 → limit is byte granular

G = 1 → limit is page granular

If G = 0, then whatever the limit is, is the actual size of the segment and so there is no multiplication factor.

If bigger segments are required, which cannot be accessed by using 20 bits, then keep G = 1. If G = 1, then whatever is specified in limit, will be multiplied by 4 KB.

D (Default Operand Size)

If D = 0 → operands are 16 bit operands (80286 segment)
D = 1 → operand are 32 bit operands (80386 segment)

AVL (Available to the programmer or OS.)

This means some more features can be added to the descriptor apart from the already available ones.

Access Byte (Bit 0 . . 7)

P (Present) (Bit 7)

If P = 1 → segment is present in physical mly.
P = 0 → segment is not present in physical mly.

DPL (Descriptor Priviledge Level) (Bit 5,6)

DPL - Priviledge level of the segment, used in priviledge tests.

A (Access) Bit 0:

If A = 1 → this segment has been recently accessed.
A = 0 → this segment has not been accessed for a long time.

If P=1 and A = 0, it means this segment is present in physical memory but it is not being used, that means it is wasting space. Such segments will be swapped out when there is shortage of space.

S (Segment Descriptor) (Bit 4)

S tells whether it is a user type (S = 1)
or it is a system segment descriptor (S = 0)

There is a descriptor for every segment.
Code, Stack and Data Segments are called user segments.
But there are descriptors for other types of segments.
For instance for interrupt, there are Interrupt Descriptor (IDT), there are task state descriptors (TSS) etc.
These are system segment descriptors.

Bit 3 E — E or C
Bit 2 ED — C
Bit 1 W — W

These 3 bits can take 2 different values
E ED W or E C R
↓ If data segment (S=1, E=0)
↓ If code segment (S=1, E=1)

Academic Year: 2022-23 **Semester: IV**
Class/Branch: SE **Subject: Microprocessor**

E ED W

It depends upon the first bit E. E stands for executable segment.

If you want to access a data segment, then S should be 1, indicating that it is a user segment.

User segment can be code (CS), data or stack segment (SS).

Data segment can be DS, ES, FS or GS

Code segment means executable segment

If E = 1 → Executable
E = 0 → Non-executable.

So if E = 0 it is definitely not a code segment. It would be data segment or stack segment

ED stands for Expansion Direction.

ED = 0 (Expand Up) → Data segment (DS, ES, FS or GS)
ED = 1 (Expand down) → Stack Segment

So if you wanted to access a data segment

S = 1
user segment

E = 0
non-executable

ED = 0
Data segment

If any of this check fails, access will be denied.
This is called type check.

W (Writable)

W = 0 Data Segment may not be written to
W = 1 Data Segment may be written to

E C R

E stands for executable segment.

C stands for Confirming

C = 1 ⇒ Code segment may only be executed when CPL ≥ DPL.

We know, there are 4 privilege levels PL0, PL1, PL2 & PL3.
But PL3 cannot access higher privilege levels.

But sometimes, if you want to allow a lower privilege level to call a s/m function (only code segment) then make C = 1 ⇒ then this segment can be accessed by any privilege level.

R (Readable)

R = 0 Code Segment may not be read
R = 1 Code Segment may be read