Chapter Two

Protected Mode Memory Addressing

2.5 Virtual Memory:

Another major feature of the 80386 is the ability to access virtual memory. A CPU with virtual memory is fooled into thinking that it has access to an unlimited amount of physical (DRAM) memory. DARM primary memory is also called main memory. In this scheme, every time the CPU looks for certain information, the operating system will first search for it in main DRAM memory and if it is not there, it will bring it into RAM from secondary memory (hard disk). If there is no room in RAM the job of the operating system to swap data out of RAM and make room for new data. Which data will be swapped out depends on how the operating system is designed. Some operating systems use the LRU (Last Recently Used) algorithm to swap data in and out of primary memory (DRAM). In the LRU method, the operating system keeps account of which data has been used the least number of times in a certain period, and when there is need for room it will swap out the least recently used data to hard disk to make room for the new data.

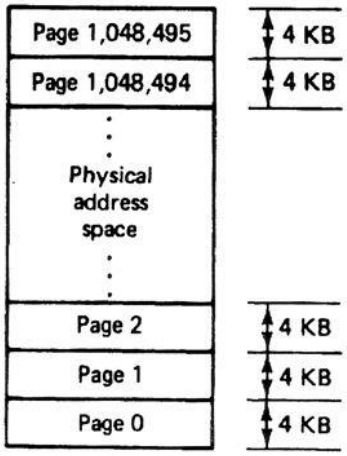


Figure (5) paged organization.

To implement virtual memory, two methods are used: segmentation and Paging. In segmentation, the size of the data swapped in and out can vary from 1byte to few megabytes (in 80386,80486 and Pentium, the upper limit can be as high as 4gigabytes). In paging shown Figure (5), the size is a multiple of one page of 4096 (4K) bytes. Paging is used widely since it prevents memory fragmentation, where available memory becomes fragmented into small sections of varied sizes. When this happens, the operating system must continuously move files around to make room for the new files, which could be any size. Paging makes the job of the operating system much easier since all the files will be multiple of 4K bytes. If the size of a file is not multiple of 4K bytes (which is the case most of the time), the operating system will leave the unused portion empty and the next file will be placed on a 4K boundary.

2.6 64 Terabytes of Virtual Memory:

The 14 bits of the selector (segment) register can have 16,834 (2^{14}) possible combinations. Each possible value can access a descriptor that can hold addresses of memory chunks as large as 4 gigabytes. Therefore, we have ($2^{14} \times 2^{32} = 64$) terabyes of virtual memory for the 80386 (recall that tera is defined as 2^{40}). To put it another way: the 80386 can access 64 terabytes of hard disk (virtual memory) as long as the virtual memory is broken down into 4gigabyte pieces, since it has only 32 address pins. While the segment limit in the 8086/286 is 64K bytes, the segment limit in the 386 was raised to 4G.

One of the drawbacks of 386 segmentations is its variable segment size, which leads to memory fragmentation. Another is absence of what is called a dirty bit in the access byte of the descriptor table. Assume that there is some memory that can be written into. The accessed (A) bit indicates if the data has been accessed but does not indicate any new data was written into it. Why should the

operating system care if the memory altered (written into)? If the data is altered, it is the job of the operating system to save it on the disk to make sure that the hard disk always has the latest the data. If the dirty bit is zero (D=0), it means that the data has not been altered and the operating system can abandon it when it needs room for new data (or code) since the original copy is on the hard disk. This will have time for the operating system. If the dirty bit is one (D=1), the operating system must save the data before it is lost or abandoned. Both problem of variable segment size and lack of a dirty bit in segmentation are fixed in the paging method of virtual memory.

2.7 The Memory Paging Mechanism:

The memory paging mechanism located within the 80386 and above allows any physical memory location to be assigned to any linear address. The linear address is defined as the address generated by a program. With the memory paging unit, the linear address is invisibly translated into any physical address. This allows an application written to function at a specific address to be relocated through the paging mechanism. It also allows to be placed into areas where no memory exists. Figure (6) show how the virtual to physical address translation.

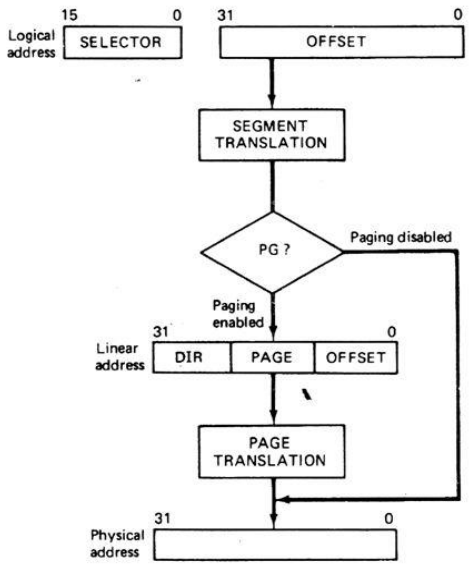


Figure (6) Virtual to physical address translation.

2.8 Paging Register

The paging unit is controlled by the contents of the microprocessors control register. Figure (7) show the contents of control register CR0 through CR3. Note that these registers are only available to the 80386 through the Pentium Pro microprocessors. Also note that the Pentium/Pentium Pro contain an additional control register labeled CR4 that controls extension provided in the Pentium/ Pentium Pro processor.

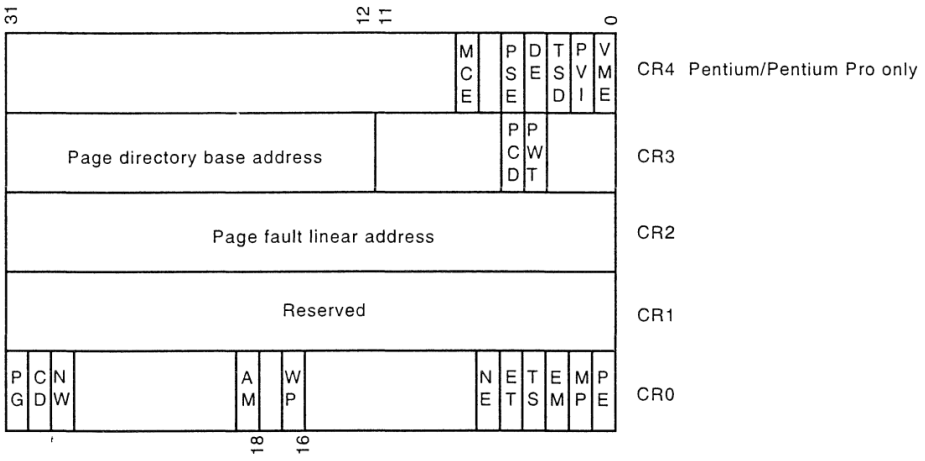


Figure (7) the control register structure of the microprocessor.

The linear address as shown in Figure (8), as it is generated by software, is broken into three sections that are used to access the Page Directory Entry, Page Table Entry and Page Offset Address. Notice how the leftmost 10bits address an entry in the page directory. Each page directory entry represents a 4M byte section of the memory system. The contents of the page directory select a page table that is indexed by the next 10bits of the linear address (bit position 12-21). This means that address 00000000H-00000FFH selects page directory entry 0 and page table entry 0. Notice that this is a 4k byte address range. The offset part of the linear address (bit position 0-11) next selects a byte in the 4K byte memory page. In Figure (9), if the page table 0 entry contains address 00100000H, then the physical address is 00100000H-00100FFFH for linear address 00000000H-00000FFFH. This means that when the program accesses a location between 00000000H-00000FFFH, the microprocessor physically addresses location 00100000H-00100FFFH.

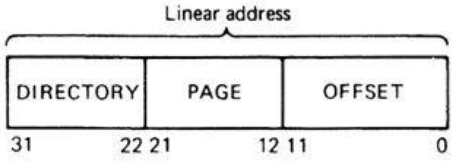


Figure (8) Linear address format

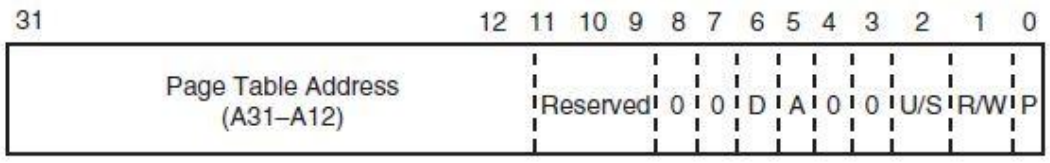


Figure (9) The page table directory entry.

Because the act of re-paging a 4K byte section of memory requires access to the page directory and page table, both located in memory, Intel has incorporated

a cache called TLB (Translation Look-aside Buffer). In the 80486 microprocessor, the cache holds 32 most recent page translation addresses. This means that the last 32 page table translation are stored in the TLB, so if the same area of memory is access to the page directory and page tables is not required. This speeds program execution. If a translation is not in TLB, then the page directory and page table must be accessed, which requires additional execution time. The Pentium and Pentium Pro both contain a separate TLB for each of their instruction and data caches.

2.9 The Page Directory

The page directory contains the location of up to 1024 page translation tables. Each page translation table translates a logic address into a physical address. The page directory is stored in the memory and accessed by the page descriptor address register (CR3) as shown in Figure (7). Control register CR3 holds the base address of the page directory, which starts at any 4K-byte boundary in the memory system.

The page directory contains up to 1024 entries, which are each four bytes long. The page directory itself occupies one 4K-byte memory page. Each entry in the page directory (see Figure 9) translates the leftmost 10 bits of the memory address. This 10-bit portion of the linear address is used to locate different page tables for different page table entries. The page table addresses (A32–A12), stored in a page directory entry, and accesses a 4 K-byte-long page translation table. To completely translate any linear address into any physical address requires 1024 page tables that are each 4K bytes long, plus the page table directory, which are also 4K bytes long. This translation scheme requires up to 4M plus 4K bytes of memory for a full address translation. Only the largest operating systems support this size address translation. Many commonly found operating systems translate

only the first 16M bytes of the memory system if paging is enabled. This includes programs such as Windows. This translation requires four entries in the page directory (16 bytes) and four complete page tables (16K bytes).

This seems like a very long and inefficient process, and it is. This is a reason for the TLB (Translation Look-aside Buffer). The TLB inside the 386 holds the list of the most recently (commonly) used physical addresses of the page frames. When the CPU wants to access a piece of information (data or code) by proving the linear address, it first compares the 20-bit upper address with the TLB to see if the table entry for the desired page is already inside the CPU. This result in two possibilities: (1) if it matches, it picks the 20-bit physical address of the page and combines it with the lower 12 bits of the linear address to a make a 32-bit physical address to put on the 32 address pins to fetch the data (or code); (2) if it does not match, the CPU must fetch into TLB the page table entry from memory.

The page table directory entry control bits, as illustrated in Figure (9), each perform the following functions:

- ***D (Dirty):*** *is undefined for page table directory entries by the 80386 microprocessor and is provided for use by the operating system.*
- ***A (Accessed):*** *is set to a logic 1 whenever the microprocessor accesses the page directory entry.*
- ***R/W (Read/write)*** *and* ***U/S (user/supervisor)*** *are both used in the protection scheme, as listed in Table (1). Both bits combine to develop paging priority level protection for level 3, the lowest user level.*
- ***P (Present)****: If a logic 1, indicates that the entry can be used in address translation. If P = 0, the entry cannot be used for translation. A not present entry can be used for other purposes, such as indicating that the page is*

currently stored on the disk. If P = 0, the remaining bits of the entry can be used to indicate the location of the page on the disk memory system.

Table (1) User and Supervisor level access rights

U/S	R/W	User	Supervisor
0	0	None	Read/Write
0	1	None	Read/Write
1	0	Read only	Read/Write
1	1	Read/Write	Read/Write

2.10 The Page Table

The page table contains 1024 physical page addresses, accessed to translate linear address into a physical address. Each page table translates a 4M section of the linear memory into 4M of physical memory. The format for the page table entry is the same as for the page directory entry. The main difference is that the page directory entry contains the physical address of a page table, while the page table entry contains the physical address of a 4K-byte physical page of memory. The other difference is the D (dirty bit), which has no function in the page directory entry, but indicates that a page has been written to in a page table entry. Figure (10) illustrates the paging mechanism in the 80386 microprocessor.

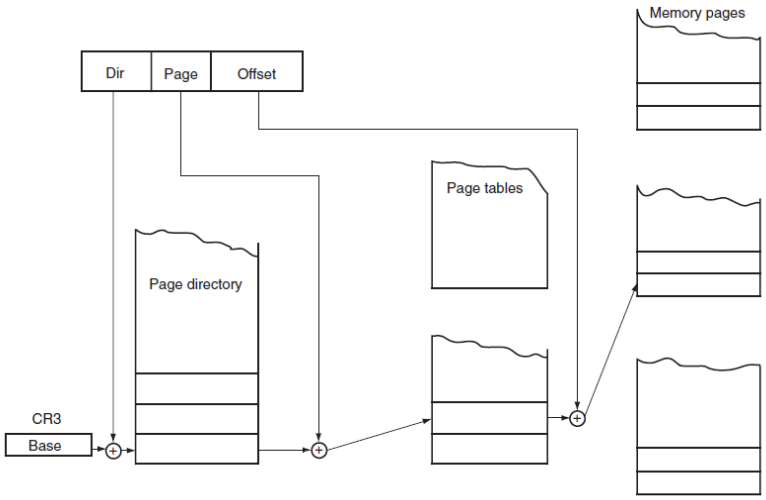
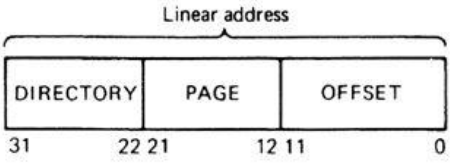


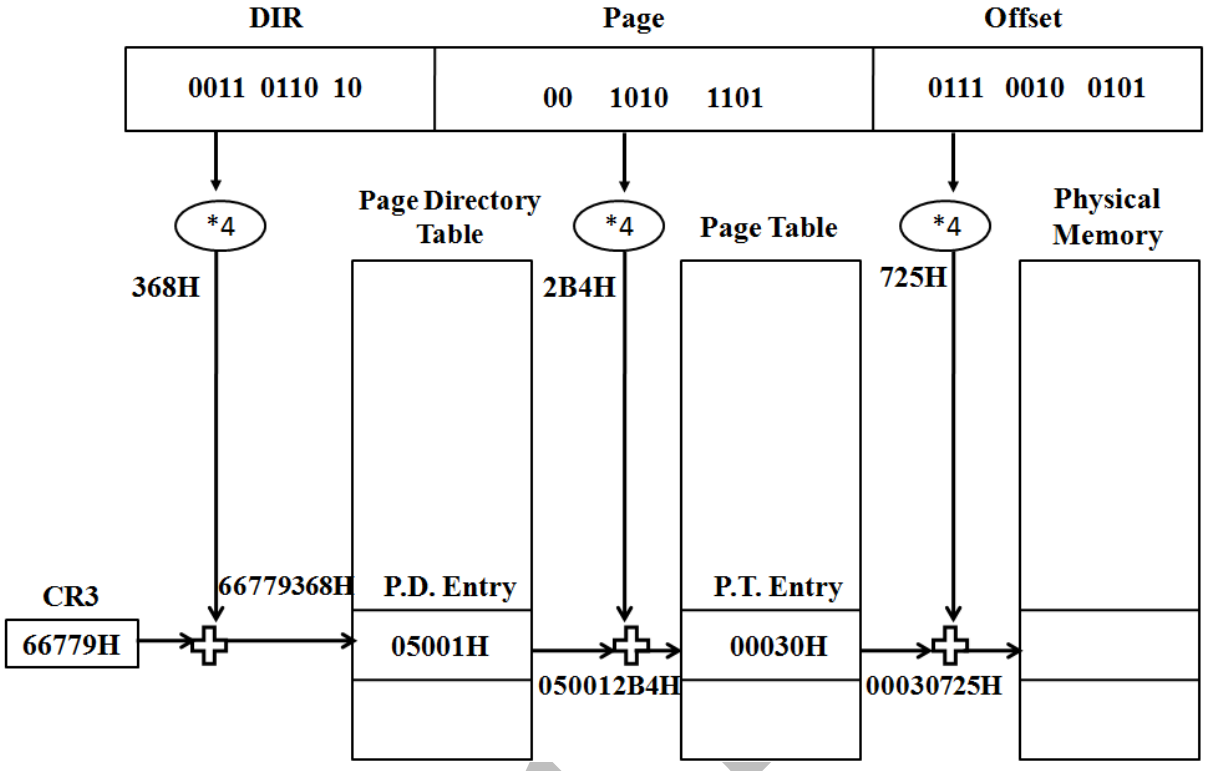
Figure (10) the paging mechanism in the 80386 microprocessor.

Example (10): *Explain using diagram the translation of the linear address (368AD725H) to a physical address, if the value of PDBA of CR3 is (66799H). The entries of the tables are: P.D. Entry=05001H and P.T. Entry=00030H.*

Solution:

Linear address =368AD725H

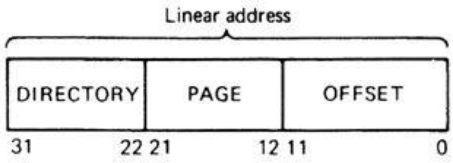


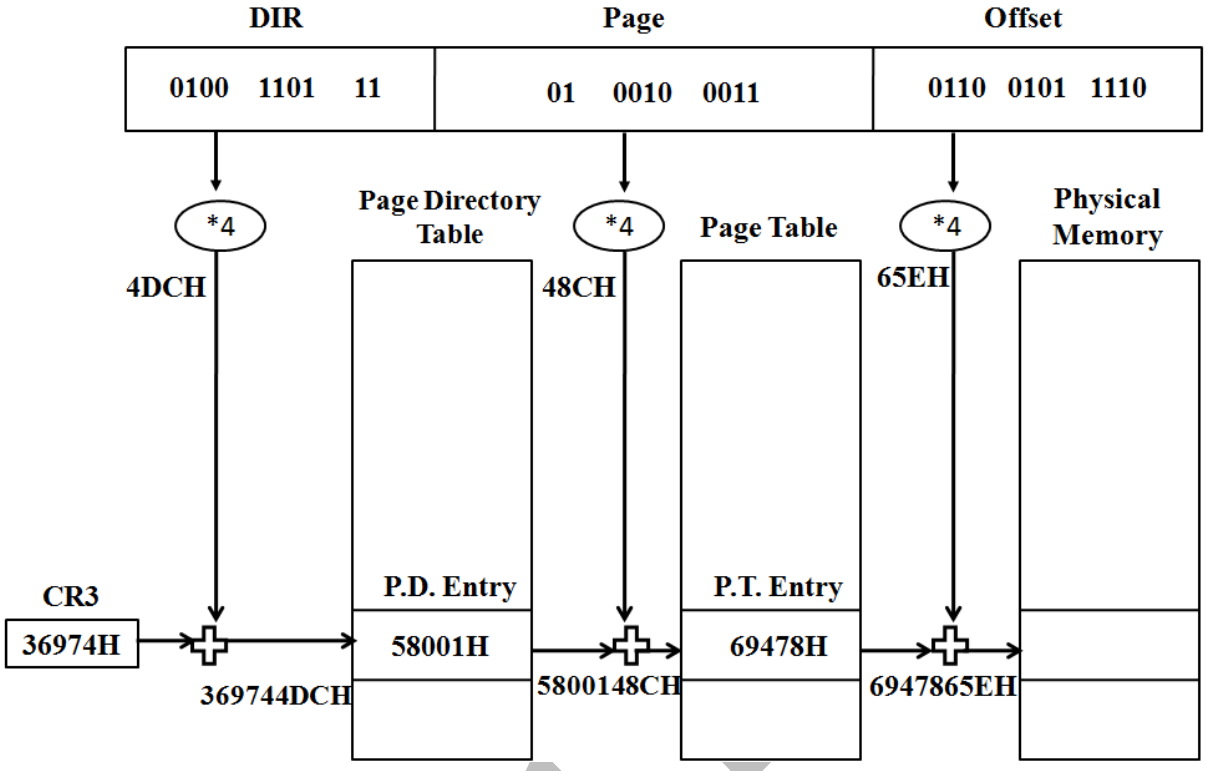


Example (11): *Explain using diagram the translation of the linear address (45D2365EH) to a physical address, if the value of PDBA of CR3 is (36974H). The entries of the tables are: P.D. Entry=58001H and P.T. Entry=69478H.*

Solution:

Linear address =(45D2365EH)





2.11 The Bigger the TLB, The Better:

*Since the TLB in the 386 keeps the list of address for the 32 most recently used pages, it allows the CPU to have access to 128K bytes (32*4=128) of code and data at any time without going through the time-consuming process of converting the linear address to a physical address (two-stage table translation) as show in Figure (11). Therefore, one way to enhance the processor is to increase the number of pages held by the TLB. This is what the Pentium has done. Table (2) compares paging and segmentation.*

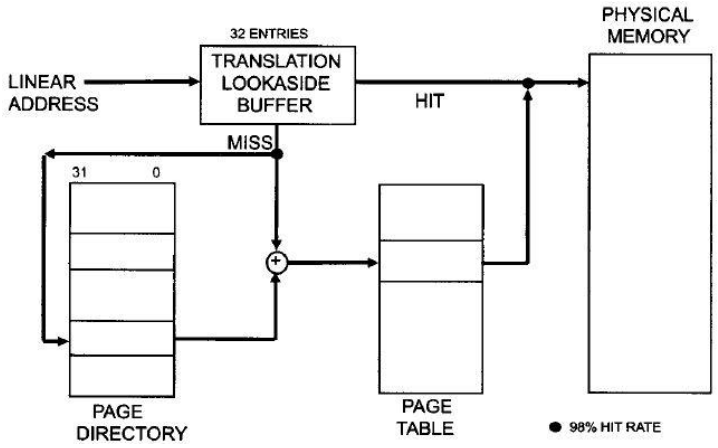
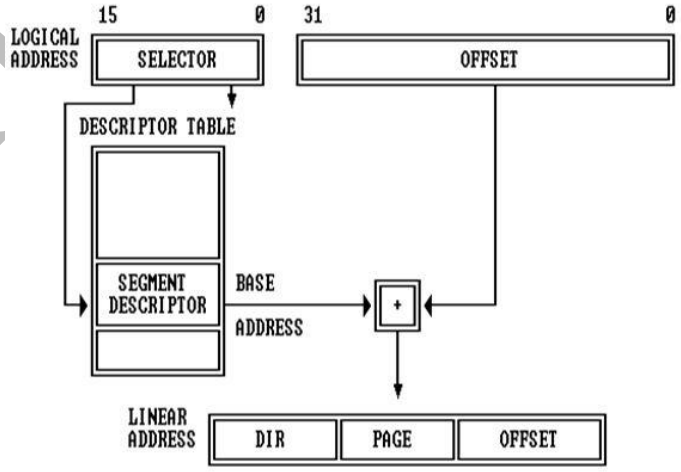


Figure (11) Translation Look-aside Buffer9

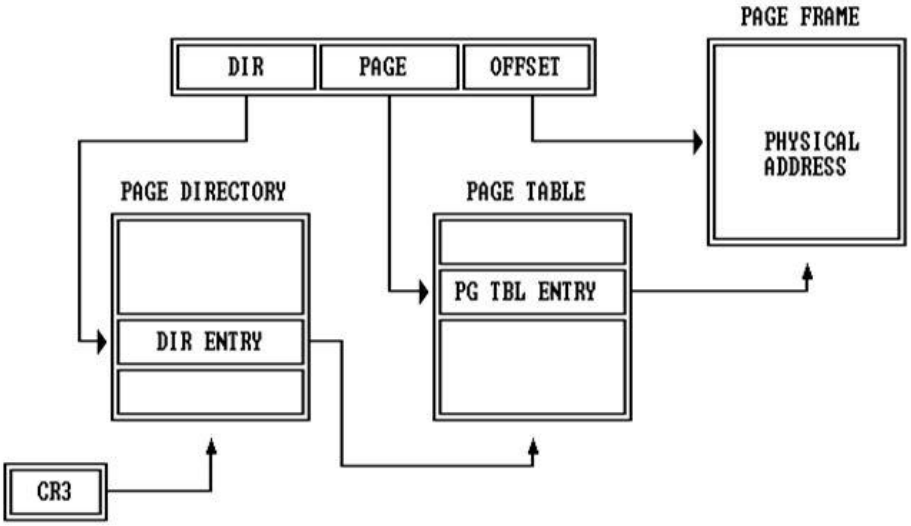
Table (2) Paging and Segmentation Comparison

Feature	*Paging*	*Segmentation*
size	*4K bytes*	*Any size*
Levels of privilege	*2*	*4*
Bass address	*4K bytes aligned*	*Any address*
Dirty bit	*Yes*	*No*
Access bit	*Yes*	*Yes*
Present bit	*Yes*	*Yes*
Read/ Write protection	*Yes*	*Yes*

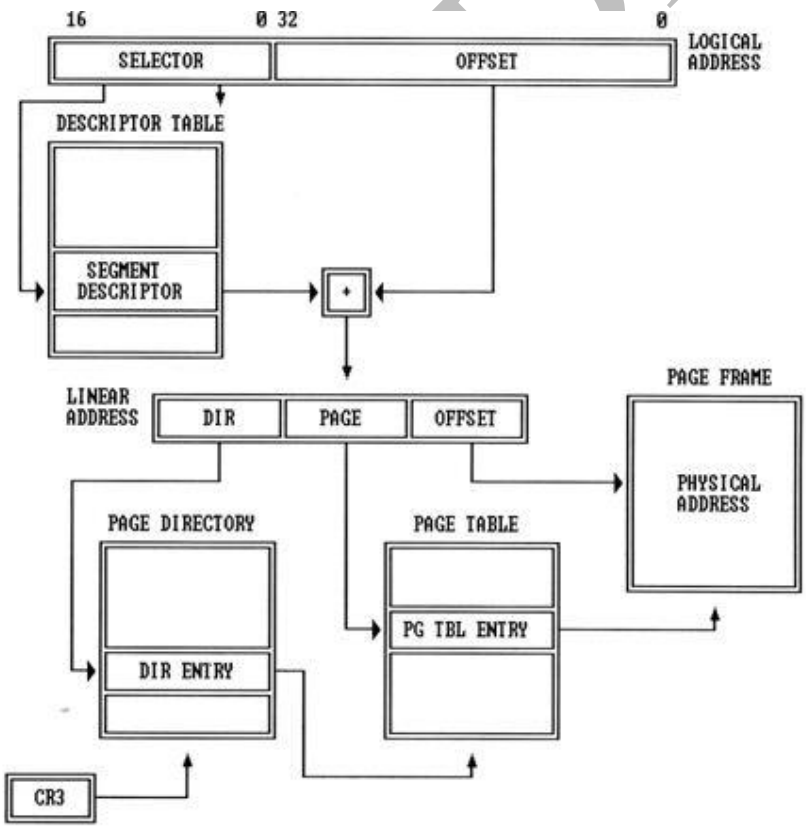
Compare between physical address in segment translation, page translation and 80386 addressing mechanism respectively as shown in Figure (11).



(a) segment translation



(b) page translation



(c) 80386 addressing mechanism

Figure (11) (a) segment translation, (b) page translation and (c) 80386 addressing mechanism respectively