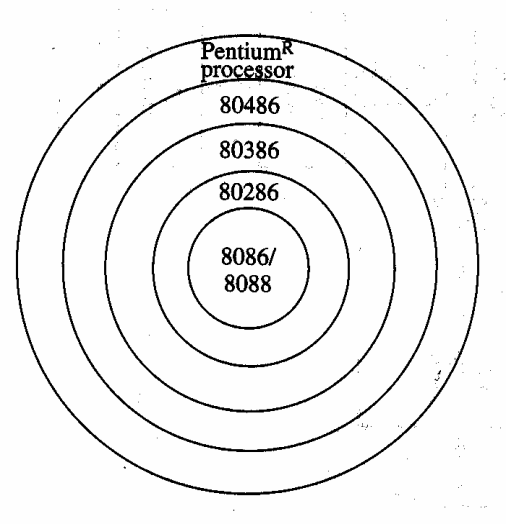
Real-Mode and Protected-Mode Memory Addressing

- Intel® 80386 all the way to **Intel® Pentium 4** processors support **three** modes of memory addressing: **real mode**, **protected mode**, and **virtual 8086 mode**.

REAL-MODE

- **Pentium 4** comes up in the **real-mode** after it is **reset**. It will remain in this mode unless it is switched to protected-mode by **software**.
- In real mode, the Pentium 4 operates as a very high performance 8086.
- Pentium 4 can be used to execute the **base instruction set** of the **8086** MPU (backward compatibility). In addition, a number of **new** instructions (called **extended instruction set**) have been added to enhance its performance and functionality (such new instructions can be run in the real-mode as well as the protected-mode).



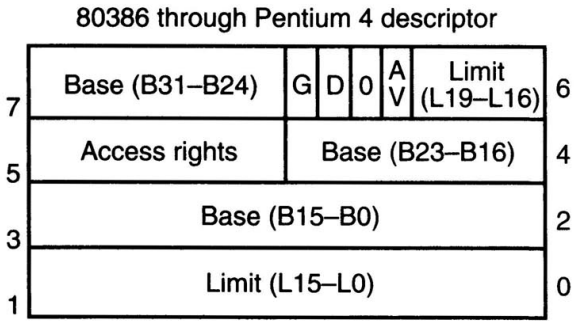
- In real-mode, only the first **1 M bytes** of memory can be addressed with the typical **segment:offset** logical address. Each **segment** is **64K bytes** long.
- Notice that the Pentium 4 microprocessor has **36 bit address bus**, which means it can support up to 2^{36} = **64G bytes** of total memory (which cannot be addressed in real-mode but can be addressed in protected mode).

PROTECTED-MODE

- In the protected-mode, memory **larger** than **1 MB** can be accessed. **Windows XP** operates in the **protected mode**.
- In addition, **segments** can be of **variable size** (below or above **64 KB**).
- Some **system control instructions** are *only* valid in the protected mode.
- In protected mode, the base:offset logical memory addressing scheme (which is used in real mode) is changed.
- The offset part of the logical memory address is still used. However, when in the protected mode, the processor can work either with 16-bit offsets (the 16-bit instruction mode) or with 32-bit offsets (the 32-bit instruction mode). A 32-bit offset allows segments of up to **4G bytes** in length. Notice that in real-mode the only available instruction mode is the 16-bit mode (during which accessing 32-bit registers requires the prefix 66h).
- However, the segment base address calculation is different in protected mode. Instead of appending a 0 at the end of the segment register contents to create a segment base address (which gives a 20-bit physical address), the segment register contains a **selector** that *selects* a **descriptor** from a descriptor table. The descriptor *describes* the memory segment's location, length, and access rights. This is similar to selecting one card from a deck of cards in one's pocket.
- Because the segment register and offset address still create a logical memory address, protected mode **instructions** are the same as real mode instructions. In fact, most programs written to function in the real mode will function without change in the protected mode.

DESCRIPTORS:

- The selector, located in the segment register, selects one of **8192 descriptors** from one of two tables of descriptors (stored in memory): the global and local descriptor tables. The descriptor describes the **location**, **length** and **access rights** of the memory segment.
- Each descriptor is **8 bytes long** and its format is shown below:



- The 8192 descriptor table requires 8 * 8192 = **64K bytes** of memory. The main parts of a descriptor are:
- **Base (B31 – B0):** indicates the starting location **(base address)** of the memory segment. This allows segments to begin at any location in the processor's 4G bytes of memory.
- **Limit (L19 – L0):** contains the **last offset address** found in a segment. Since this field is 20 bits, the segment size could be anywhere between 1 and 1M bytes. However, if the **G bit (granularity bit)** is set, the value of the limit is multiplied by **4K bytes** (i.e., appended with FFFH). In this case, the segment size could be anywhere between 4K and 4G bytes in steps of 4K bytes.
- Example,
- Base = Start = **10000000h**
- Limit = **001FFh** and G = **0**
- So, End = Base + Limit = 10000000h + 001FFh = 100001FFh
- Segment Size = **512 bytes**

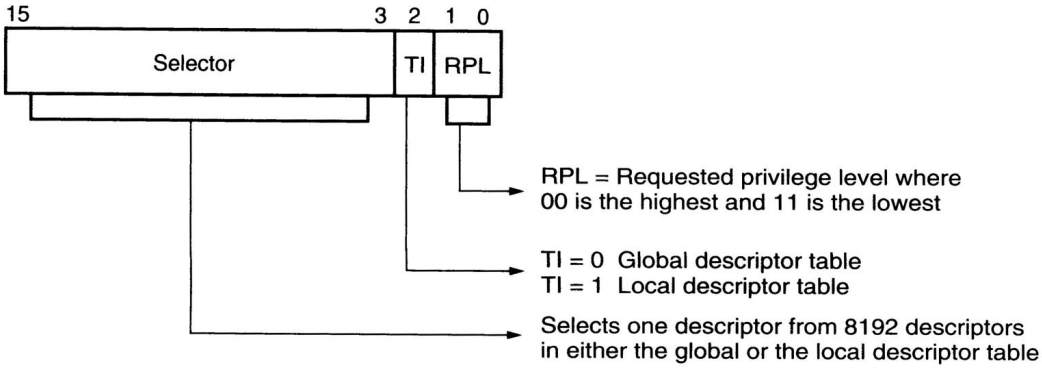
- Base = Start = **10000000h**
- Limit = **001FFh** and G = **1**
- So, End = Base + Limit * 4K = 10000000h + **001FFFFFh** = **101FFFFFh**
- Segment Size = **2M bytes**

- **AV bit:** is used by some operating systems to indicate that the segment is available (AV = 1) or not available (AV = 0).
- **D bit:** If D = 0, the instructions are 16-bit instructions, compatible with the 8086-80286 microprocessors. This means that the instructions use 16-bit offset addresses and 16-bit registers by default. This mode is the 16-bit instruction mode or DOS mode. If D = 1, the instructions are 32-bits by default (Windows XP works in this mode). By default, the 32-bit instruction mode assumes that all offset addresses and all registers are 32 bits. Note that the default for register size and offset address can be overridden in both the 16- and 32-bit instruction modes using the 66h and 67h prefixes. In 16-bit protected-mode, descriptors are still used but segments are supposed to be a maximum of 64K bytes.
- **Access rights byte:** allows complete control over the segment. If the segment is a data segment, the direction of growth is specified. If the segment grows beyond its limit, the microprocessor's operating system program is interrupted, indicating a **general protection fault**. You can specify whether a data segment can be written or is write-protected. The code segment can have reading inhibited to protect software. This is why it is called protected mode. This kind of protection is unavailable in real-mode.

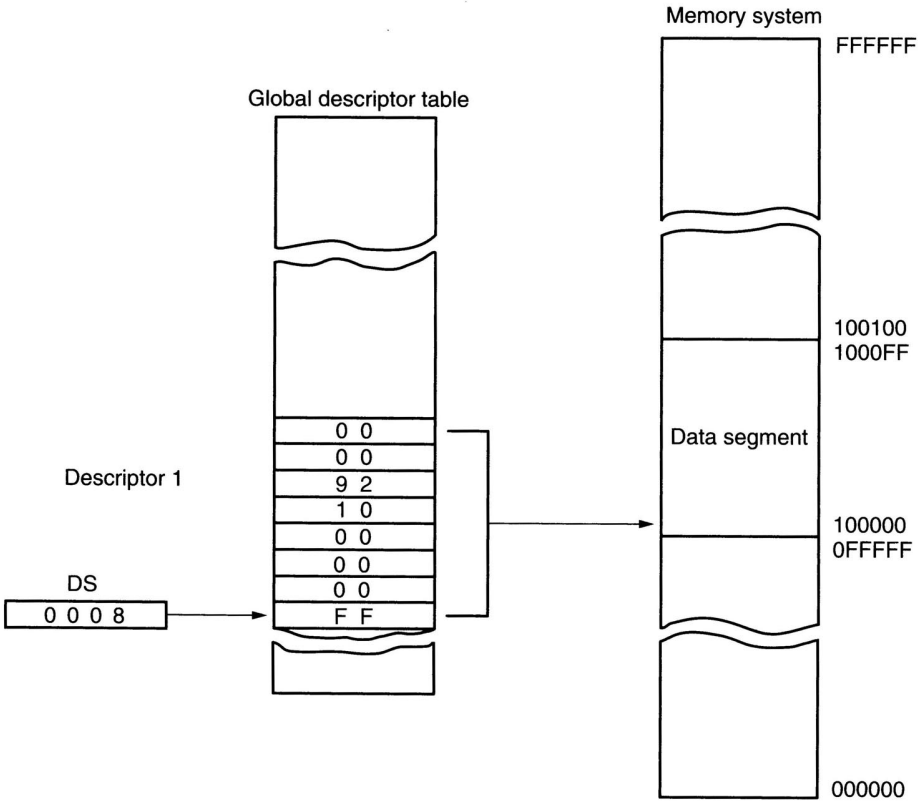
SELECTORS:

- Descriptors are chosen from the descriptor table by the segment register. There are two descriptor tables:
- **Global descriptors table:** contains segment definitions that apply to **all** programs (also called **system descriptors**).
- **Local descriptors table:** usually unique to an application (also called **application descriptors**).
- Each descriptor table contains 8192 descriptors, so a total of 16,384 descriptors are available to an application at any time. This allows up to 16,384 memory segments to be described for each application.
- The Figure below shows the segment register in the protected mode. It contains:

- **13-bit selector field:** chooses one of the 8192 descriptors from the descriptor table (2^{13} = 8192).
- **Table indicator (TI) bit:** selects either the global descriptor table (TI = 0) or the local descriptor table (TI = 1).
- **Requested privilege level (RPL) field:** requests the access privilege level of a memory segment. The highest privilege level is 00 and the lowest is 11. If the requested privilege level matches or is higher in priority than the privilege level set by the access rights byte, access is granted. Windows uses privilege level 00 (ring 0) for the kernel and driver programs and level 11 (ring 3) for applications. Windows does not use levels 01 or 10. If privilege levels are violated, the system normally indicates a **privilege level violation**.



- Example:
- **Real Mode: DS = 0008h**, then the data segment begins at location **00080h** and its length is **64K bytes**.
- **Protected Mode: DS = 0008h =** 0000 0000 0000 **1000**, then the selector selects **Descriptor 1** in the **global** descriptor table using a requested privilege level of **00**. The global descriptor table is stored in memory as shown below.



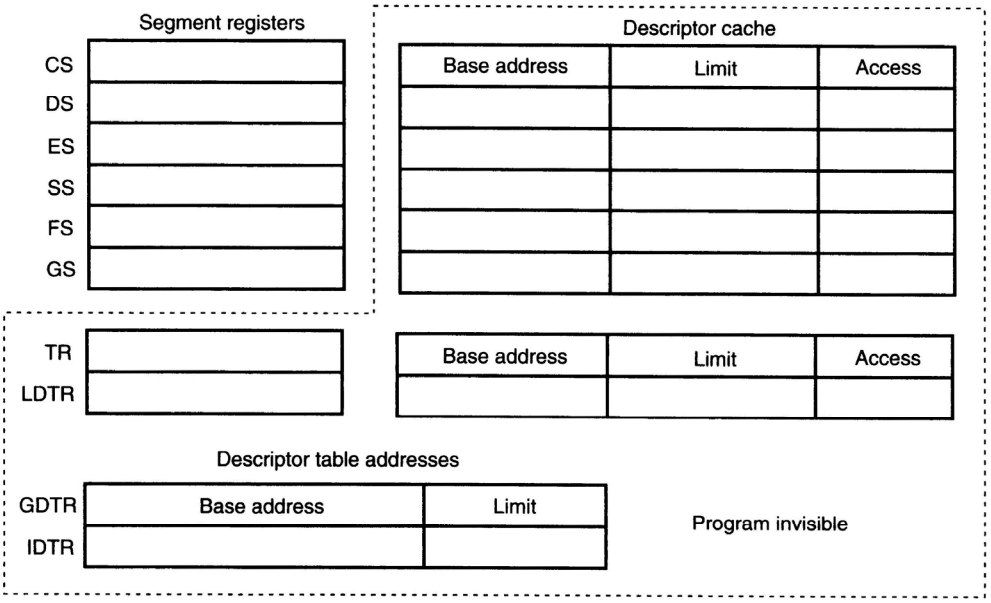
- Descriptor number 1 contains a descriptor that defines the base address as 00100000h with a segment limit of 000FFh. This refers to memory locations **00100000h – 001000FFh** for the data segment.

PROGRAM-INVISIBLE REGISTERS:

- The global and local descriptor tables are found in the memory system. In order to specify the address of these tables, Pentium 4 contains program-invisible registers LDTR and GDTR (these registers are not directly addressed by software).

- The **GDTR** (global descriptor table register), **LDTR** (local descriptor table register) and **IDTR** (interrupt descriptor table register) contain the **base address** of the descriptor table and its **limit**. The limit of these descriptor tables is 16 bits because the maximum table length is 64K bytes (but of course, the table could be smaller than 64K byte, hence the need for the limit).

- Before using the protected mode, the interrupt descriptor table, global descriptor table along with the corresponding registers IDTR and GDTR must be initialized. This is why the Pentium 4 boots in the real mode not protected mode, and why the maximum descriptor table size is 64K bytes.



Notes:
1. The 80286 does not contain FS and GS nor the program-invisible portions of these registers.
2. The 80286 contains a base address that is 24-bits and a limit that is 16-bits.
3. The 80386/80486/Pentium/Pentium Pro contain a base address that is 32-bits and a limit that is 20-bits.
4. The access rights are 8-bits in the 80286 and 12-bits in the 80386/80486/Pentium–Pentium 4.

- Each of the segment registers also contains a program-invisible portion used as a cache to store the corresponding 8 byte descriptor to avoid repeatedly accessing memory every time the segment register is referenced (hence the term cache).
- These program-invisible registers are loaded with the base address, limit, and access rights each time the number in the segment register is changed.
- The TR (task register) holds a selector, which accesses a descriptor that defines a task. A task is most often a procedure or application program. The descriptor for the procedure or application program is stored in the global descriptor table, so access can be controlled through the privilege levels. The task register allows a context or task switch in multitasking systems in about 17μs.
- **Notice:** The memory system for the Pentium 4 is 4G bytes in size, but access to the area between 4G and 64G is enabled with bit position 4 of the control register CR4 and is accessible only when 4M paging is enabled. When in this paging mode, address lines $A_{35} - A_{32}$ are enabled with a special new addressing mode, controlled by other bits in CR4.

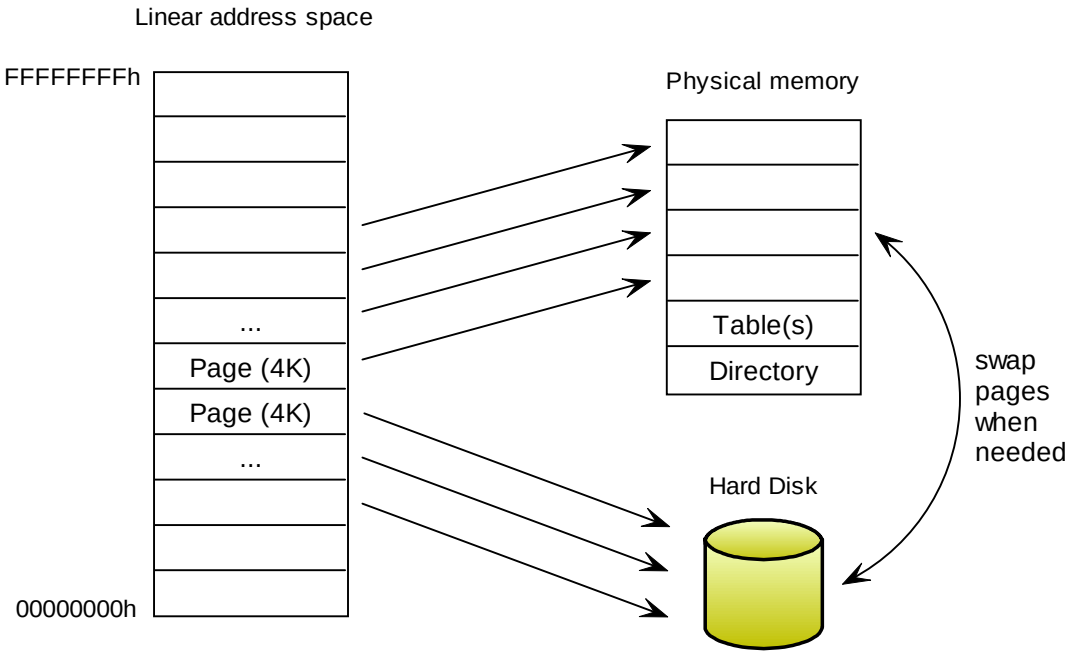
Memory Paging

- Paging is enabled when the PG bit in control register CR0 is set. The paging mechanism can function in both the real and protected modes.
- When paging is enabled, physical memory is divided into small blocks (typically 4K bytes or 4M bytes) in size, and each block is assigned a page number. The operating system keeps a list of free pages in its memory. When a program makes a request for memory, the OS allocates a number of pages to the program.
- A key advantage to memory paging is that memory allocated to a program does not have to be **contiguous**, and because of that, there is very little internal fragmentation - thus **little memory is wasted**.
- Example: Show memory page allocation for the following sequence:
- Program A requests 3 pages of memory.
- Program C requests 2 pages of memory.
- Program D requests 2 pages of memory.
- Program C terminates, leaving 2 empty pages.
- Program B requests 3 pages of memory, and it is allocated the 2 empty pages that program C left, plus an additional page after program D.

Page Number	Program Allocated to	Physical Memory Address
0	Program A.0	0000 - 0FFF
1	Program A.1	1000 – 1FFF
2	Program A.2	2000 – 2FFF
3	Program B.0	3000 – 3FFF
4	Program B.1	4000 – 4FFF
5	Program D.0	5000 – 5FFF
6	Program D.1	6000 – 6FFF
7	Program B.2	7000 – 7FFF

- Consequently, Program A's page tables would contain the following mapping (Program's Page #=>OS Page #): (0=>0, 1=>1, 2=>2); Program B's: (0=>3,1=>4,2=>7); and Program D's : (0=>5, 1=>6). And these programs operate as if they have contiguous memory space.

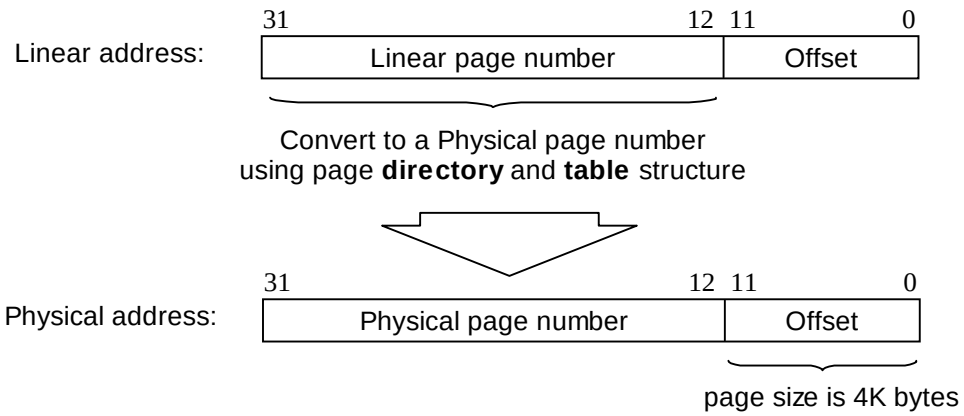
- Another advantage of paging is the **use of virtual memory**, where the OS keeps track of pages that have not been used for some time. Then, when the OS deems fit, the OS swaps out a page to disk, and brings another page into memory. In this way, you can use more memory than the computer physically has.



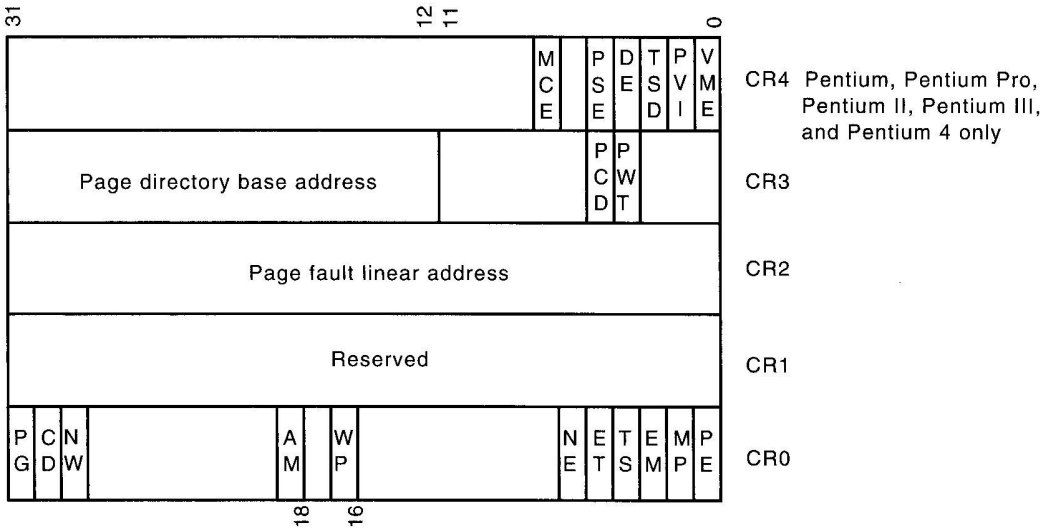
- When paging is enabled, a **logical address** is mapped into a **linear address** (anywhere between 0 and 4G bytes). Then this linear address is mapped to another **physical address** (depending on the amount of physical memory).
- The linear address is defined as the address generated by a program. The physical address is the actual memory location accessed by a program.
- Translation from the linear address to the appropriate physical address is done at the **hardware level**, and is handled by the **memory management unit** (MMU), and the process is **transparent** to the Assembly programmer.

THE PAGE DIRECTORY AND PAGE TABLE

- To convert a 32-bit linear address into a 32-bit physical address, we need to understand that the most significant **20 bits** of the linear address indicate the **linear page number**, while the least significant **12 bits** of the linear address indicate the **offset within this page**. The offset should remain the same but the linear page number has to be converted into a physical page number.



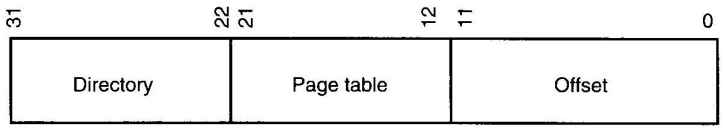
- To translate to a physical page number, you have to look up a **page directory**, which contains **page directory entries**. The page directory contains **1024** page directory entries, each of which is 4 bytes (32 bits). This means the page directory is **4 K bytes** long. There is only one page directory in memory and its base address is contained in CR3. Note that this address locates the page directory at any 4K boundary in the memory system because it is appended internally with 000h.



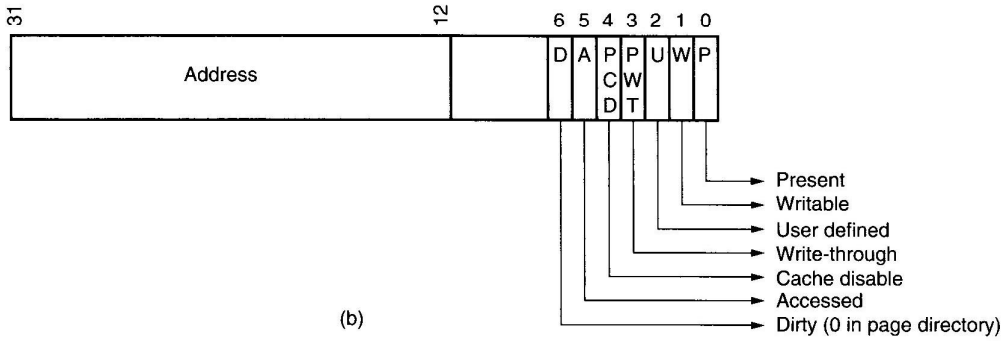
- Each page directory entry is a physical address pointing to a **page table**, which contains **page table entries**. Each page table contains **1024** page table entries, each of which is 4 bytes (32 bits). This means that each page table is **4 K bytes** long.
- Each page table entry points to the starting physical address of a page in memory (i.e., the **physical page number**).
- This means that if we have **one** page directory and **1024** page tables, then

we have a total of 1M table entries or 1 M pages. Since each page is 4K bytes long, this will cover a total of 4G bytes of maximum physical memory.

- The figure below Part (a) shows the linear address (generated by the software) and how it selects one of the 1024 page directory entries from the page directory (using the left most 10 bits) and then selects one of the 1024 page table entries (using the next 10 bits). Part (b) of the figure shows the page table entry, which contains the physical page number that must be associated with the offset.



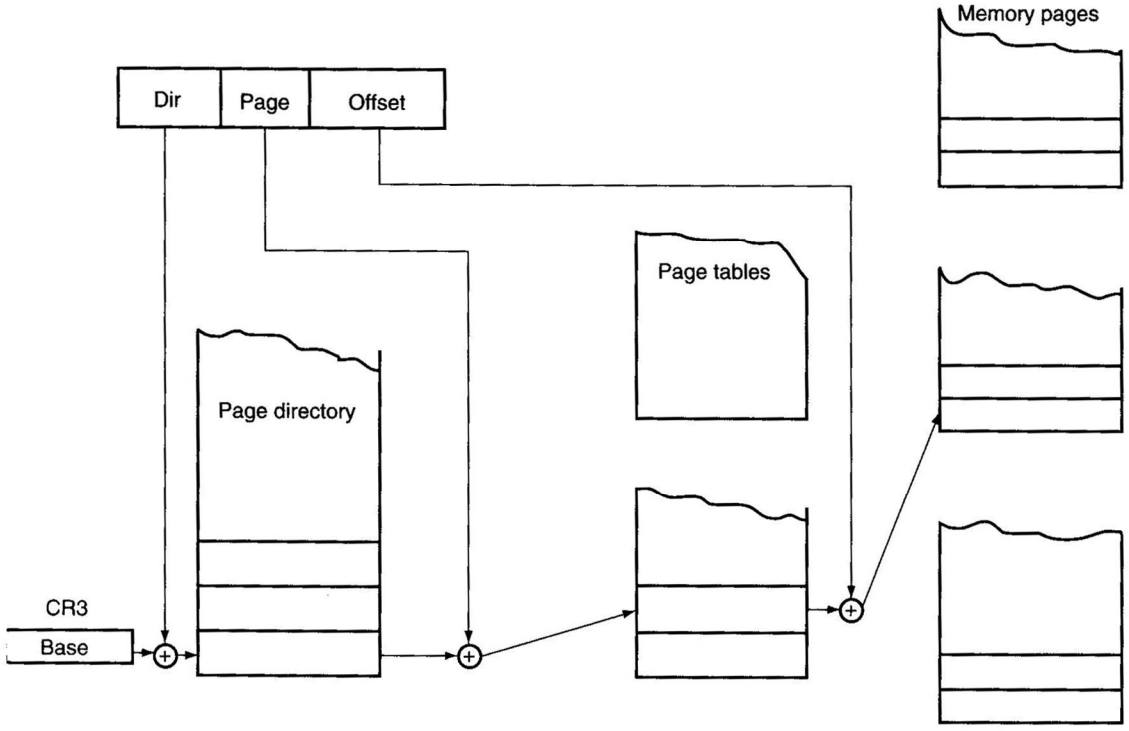
(a)



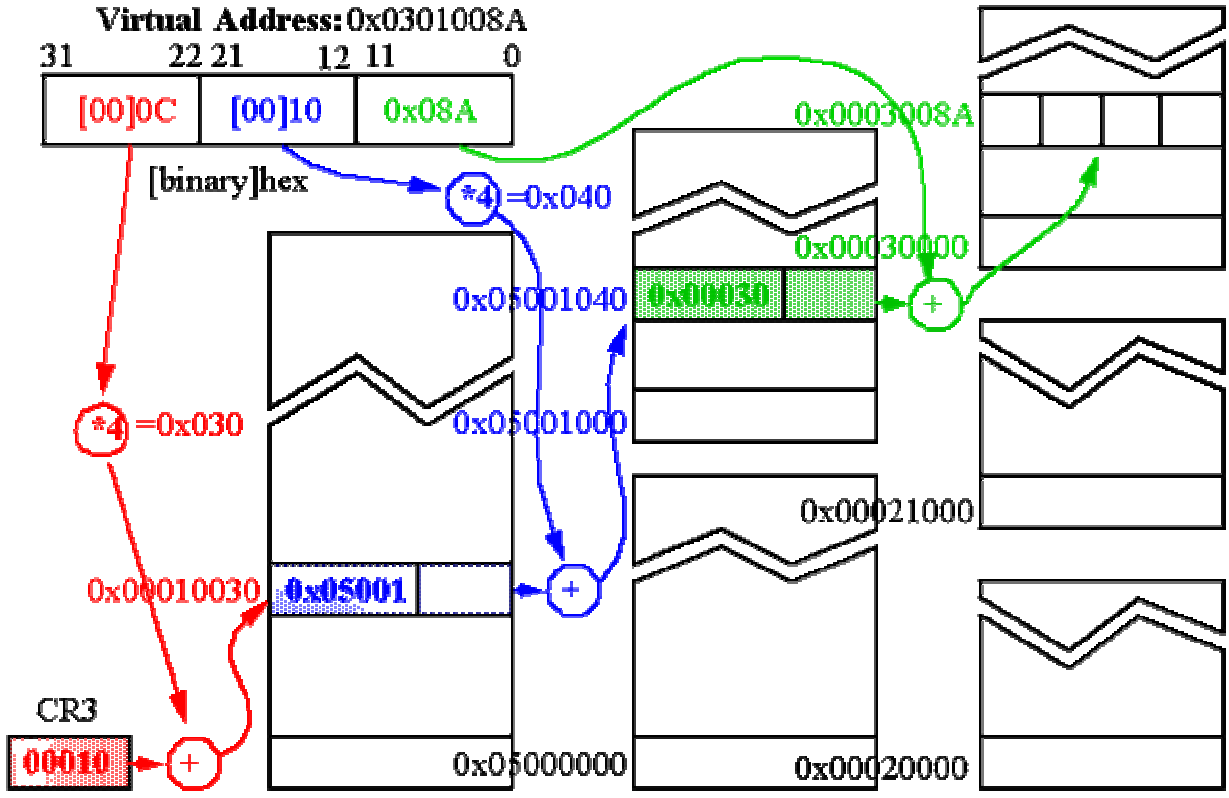
(b)

- For example, the linear addresses 00000000h-00000FFFh access the first page directory entry, and the first page table entry. Notice that one page is a 4K-byte address range. So, if that page table entry contains 00100000h, then the physical address of this page is 00100000h-00100FFFh for linear address 00000000h-00000FFFh. This means that when the program accesses a location between 00100000h and 00100FFFh, the microprocessor physically addresses location 00100000h-00100FFFh.
-
-
-
-

- The procedure for converting linear addresses into physical addresses:



- A numerical example is shown below:



- Because the act of re-paging a 4K-byte section of memory requires access to the page directory and a page table, which are both located in memory, Intel has incorporated a special type of cache called the **TLB (translation look-aside buffer)**. This cache holds the most recent page translation addresses, so if the same area of memory is accessed, the address is already present in the TLB, and access to the page directory and page tables is not required.
- If the entire 4G byte of memory is paged, the system must allocate 4K bytes of memory for the page directory, and 4K times 1024 or 4M bytes for the 1024 page tables. This represents a con considerable investment in memory resources. On the Pentium 4 microprocessors, pages can be either 4K bytes in length or 4M bytes in length. Although no software currently supports the 4M-byte pages, as the Pentium 4 and more advanced versions pervade the personal computer arena, operating systems of the future will undoubtedly begin to support 4M-byte memory pages.