5.3 Page Translation

A linear address is a 32-bit address into a uniform, unsegmented address space. This address space may be a large physical address space (i.e., an address space composed of 4 gigabytes of RAM), or paging can be used to simulate this address space using a small amount of RAM and some disk storage. When paging is used, a linear address is translated into its corresponding physical address, or an exception is generated. The exception gives the operating system a chance to read the page from disk (perhaps sending a different page out to disk in the process), then restart the instruction which generated the exception.

Paging is different from segmentation through its use of small, fixed-size pages. Unlike segments, which usually are the same size as the data structures they hold, on the i486 processor, pages are always 4K bytes. If segmentation is the only form of address translation which is used, a data structure which is present in physical memory will have all of its parts in memory. If paging is used, a data structure may be partly in memory and partly in disk storage.

The information which maps linear addresses into physical addresses and exceptions is held in data structures in memory, called *page tables*. As with segmentation, this information is cached in processor registers to minimize the number of bus cycles required for address translation. Unlike segmentation, these processor registers are completely invisible to application programs. (For testing purposes, these registers are visible to programs running with maximum privileges; see Chapter 10 for details.)

The paging mechanism treats the 32-bit linear address as having three parts, two 10-bit indexes into the page tables and a 12-bit offset into the page addressed by the page tables. Because both the virtual pages in the linear address space and the physical pages of memory are aligned to 4K-byte page boundaries, there is no need to modify the low 12 bits of the address. These 12 bits pass straight through the paging hardware, whether paging is enabled or not. Note that this is different from segmentation, because segments can start at any byte address.

The upper 20 bits of the address are used to index into the page tables. If every page in the linear address space were mapped by a single page table in RAM, 4 megabytes would be needed. This is not done. Instead, two levels of page tables are used. The top level page table is called the *page directory*. It maps the upper 10 bits of the linear address to the second level of page tables. The second level of page tables maps the middle 10 bits of the linear address to the base address of a page in physical memory (called a *page frame address*).

An exception may be generated based on the contents of the page table or the page directory. An exception gives the operating system a chance to bring in a page table from disk storage. By allowing the second-level page tables to be sent to disk, the paging mechanism can support mapping of the entire linear address space using only a few pages in memory.

The CR3 register holds the page frame address of the page directory. For this reason, it also is called the page directory base register or PDBR. The upper 10 bits of the linear address are scaled by four (the number of bytes in a page table entry) and added to the value in the PDBR register to get the physical address of an entry in the page directory. Because the page frame address is always clear in its lowest 12 bits, this addition is performed by concatenation (replacement of the low 12 bits with the scaled index).

When the entry in the page directory is accessed, a number of checks are performed. Exceptions may be generated if the page is protected or is not present in memory. If no exception is generated, the upper 20 bits of the page table entry are used as the page frame address of a second-level page table. The middle 10 bits of the linear address are scaled by four (again, the size of a page table entry) and concatenated with the page frame address to get the physical address of an entry in the second-level page table.

Again, access checks are performed, and exceptions may be generated. If no exception occurs, the upper 20 bits of the second-level page table entry are concatenated with the lowest 12 bits of the linear address to form the physical address of the operand (data) in memory.

Although this process may seem complex, it all takes place with very little overhead. The processor has a cache for page table entries called the translation lookaside buffer (TLB). The TLB satisfies most requests for reading the page tables. Extra bus cycles occur only when a new page is accessed. The page size (4K bytes) is large enough so that very few bus cycles are made to the page tables, compared to the number of bus cycles made to instructions and data. At the same time, the page size is small enough to make efficient use of memory. (No matter how small a data structure is, it occupies at least one page of memory.)

5.3.1 PG Bit Enables Paging

If paging is enabled, a second stage of address translation is used to generate the physical address from the linear address. If paging is not enabled, the linear address is used as the physical address.

Paging is enabled when bit 31 (the PG bit) of the CR0 register is set. This bit usually is set by the operating system during software initialization. The PG bit must be set if the operating system is running more than one program in virtual-8086 mode or if demand-paged virtual memory is used.

5.3.2 Linear Address

Figure 5-12 shows the format of a linear address.

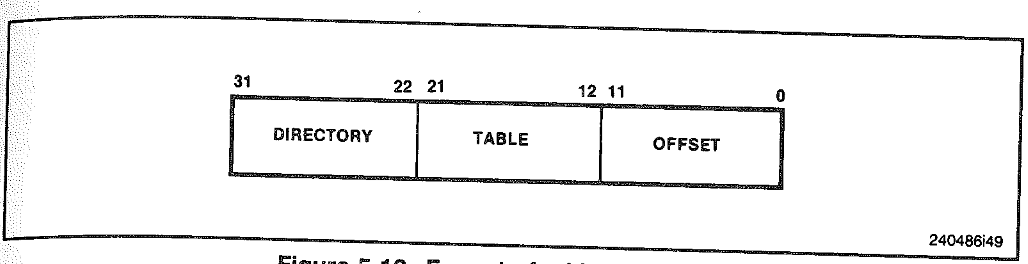


Figure 5-12. Format of a Linear Address

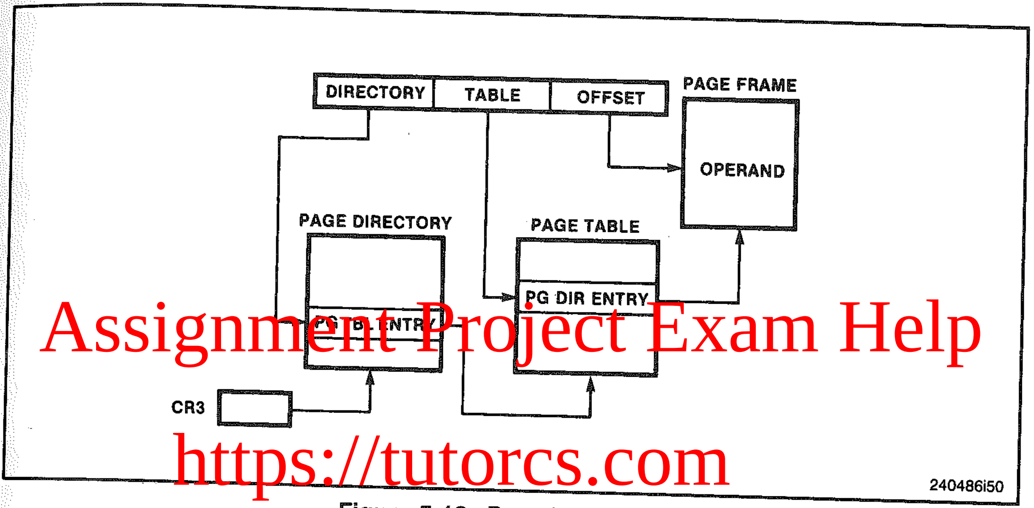


Figure 5-13. Page Translation

Figure 5-13 shows how the processor translates the DIRECTORY, TABLE, and OFFSET fields of a linear address into the physical address using two levels of page tables. The paging mechanism uses the DIRECTORY field as an index into a page directory, the TABLE field as an index into the page table determined by the page directory, and the OFFSET field to address an operand within the page specified by the page table.

5.3.3 Page Tables

A page table is an array of 32-bit entries. A page table is itself a page, and contains 4096 bytes of memory or, at most, 1K 32-bit entries. All pages, including page directories and page tables, are aligned to 4K-byte boundaries.

Two levels of tables are used to address a page of memory. The top level is called the page directory. It addresses up to 1K page tables in the second level. A page table in the second level addresses up to 1K pages in physical memory. All the tables addressed by one page directory, therefore, can address 1M or 2^{20} pages. Because each page contains 4K or 2^{12} bytes, the tables of one page directory can span the entire linear address space of the i486 processor ($2^{20} \times 2^{12} = 2^{32}$).

The physical address of the current page directory is stored in the CR3 register, also called the page directory base register (PDBR). Memory management software has the option of using one page directory for all tasks, one page directory for each task, or some combination of the two. See Chapter 10 for information on initialization of the CR3 register. See Chapter 7 for how the contents of the CR3 register can change for each task.

5.3.4 Page-Table Entries

Entries in either level of page tables have the same format. Figure 5-14 illustrates this format.

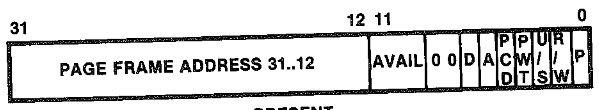
5.3.4.1 PAGE FRAME ADDRESS

The page frame address is the base address of a page. In a page table entry, the upper 20 bits are used to specify a page frame address, and the lowest 12 bits specify control and status bits for the page. In a page directory, the page frame address is the address of a page table. In a second-level page table, the page frame address is the address of a page containing instructions or data.

5.3.4.2 PRESENT BIT

The Present bit indicates whether the page frame address in a page table entry maps to a page in physical memory. When set, the page is in memory.

When the Present bit is clear, the page is not in memory, and the rest of the page table entry is available for the operating system, for example, to store information regarding the whereabouts of the missing page. Figure 5-15 illustrates the format of a page table entry when the Present bit is clear.



P	— PRESENT
R/W	— READ/WRITE
U/S	— USER/SUPERVISOR
PWT	— PAGE WRITE TRANSPARENT
PCD	— PAGE CACHE DISABLE
A	— ACCESSED
D	— DIRTY
AVAIL.	— AVAILABLE FOR SYSTEMS PROGRAMMER USE

NOTE: 0 INDICATES INTEL RESERVED. DO NOT DEFINE.

240486i51

Figure 5-14. Format of a Page Table Entry

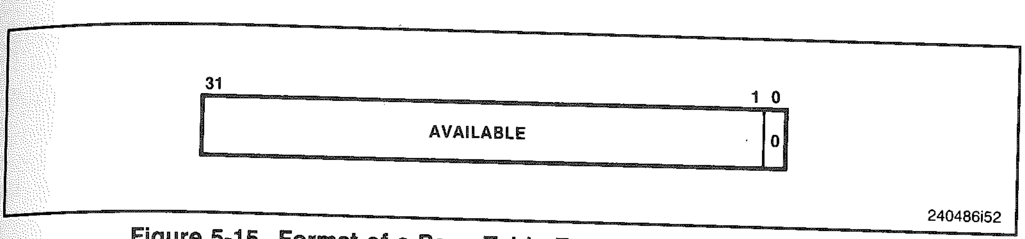


Figure 5-15. Format of a Page Table Entry for a Not-Present Page

If the Present bit is clear in either level of page tables when an attempt is made to use a page table entry for address translation, a page-fault exception is generated. In systems which support demand-paged virtual memory, the following sequence of events then occurs:

1. The operating system copies the page from disk storage into physical memory.
2. The operating system loads the page frame address into the page table entry and sets its Present bit. Other bits, such as the R/W bit, may be set, too.
3. Because a copy of the old page table entry may still exist in the translation lookaside buffer (TLB), the operating system empties it. See Section 5.3.5 for a discussion of the TLB and how to empty it.
4. The program which caused the exception is then restarted.

Since there is no Present bit in CR3 to indicate when the page directory is not resident in memory, the page directory pointed to by CR3 should always be present in physical memory.

5.3.4.3 ACCESSED AND DIRTY BITS

These bits provide data about page usage in both levels of page tables. The Accessed bit is used to report read or write access to a page or second-level page table. The Dirty bit is used to report write access to a page.

With the exception of the Dirty bit in a page directory entry, these bits are set by the hardware; however, the processor does not clear either of these bits. The processor sets the Accessed bits in both levels of page tables before a read or write operation to a page. The processor sets the Dirty bit in the second-level page table before a write operation to an address mapped by that page table entry. The Dirty bit in directory entries is undefined.

The operating system may use the Accessed bit when it needs to create some free memory by sending a page or second-level page table to disk storage. By periodically clearing the Accessed bits in the page tables, it can see which pages have been used recently. Pages which have not been used are candidates for sending out to disk.

The operating system may use the Dirty bit when a page is sent back to disk. By clearing the Dirty bit when the page is brought into memory, the operating system can see if it has received any write access. If there is a copy of the page on disk and the copy in memory has not received any writes, there is no need to update disk from memory.

See Chapter 13 for how the i486 processor updates the Accessed and Dirty bits in multiprocessor systems.

5.3.4.4 READ/WRITE AND USER/SUPERVISOR BITS

The Read/Write and User/Supervisor bits are used for protection checks applied to pages, which the processor performs at the same time as address translation. See Chapter 6 for more information on protection.

5.3.4.5 PAGE-LEVEL CACHE CONTROL BITS

The PCD and PWT bits are used for page-level cache management. Software can control the caching of individual pages or second-level page tables using these bits. See Chapter 12 for more information on caching.

5.3.5 Translation Lookaside Buffer

The processor stores the most recently used page table entries in an on-chip cache called the translation lookaside buffer or TLB. Most paging is performed using the contents of the TLB. Bus cycles to the page tables are performed only when a new page is used.

The TLB is invisible to application programs, but not to operating systems. Operating-system programmers must flush the TLB (dispose of its page table entries) when entries in the page tables are changed. If this is not done, old data which has not received the changes might get used for address translation. A change to an entry for a page which is not present in memory does not require flushing the TLB, because entries for not-present pages are not cached.

The TLB is flushed when the CR3 register is loaded. The CR3 register can be loaded in either of two ways:

1. Explicit loading using MOV instructions, such as:

   ```
   MV CR3, EAX
   ```

2. Implicit loading by a task switch which changes the contents of the CR3 register. (See Chapter 7 for more information on task switching.)

An individual entry in the TLB can be flushed using an INVLPG instruction. This is useful when the mapping of an individual page is changed.