n =	1	2	3	4	5	>5
PF =	1.00	0.83	0.58	0.50	0.42	0.42

Time	1	2	3	4	5	6	7	8	9	10	11	12
Reference sequence	1	2	1	0	4	1	3	4	2	1	4	1
$S_t(1)$	1	2	1	0	4	1	3	4	2	1	4	1
$S_t(2)$		1	2	1	0	4	1	3	4	2	1	4
$S_t(3)$				2	1	0	4	1	3	4	2	2
$S_t(4)$					2	2	0	0	1	3	3	3
$S_t(5)$							2	2	0	0	0	0
n = 1												
n = 2			S									S
n = 3			S			S		S			S	S
n = 4			S			S		S		S	S	S
n = 5			S			S		S	S	S	S	S

Figure 4.60. Stack processing of a reference sequence using the LRU replacement policy.

From the inclusion property of stack replacement policies follows that the page fault frequency decreases with the increase of the available memory capacity n, or, equivalently, the hit ratio increases. If the next page address x is in set $P_t(n)$, it must also be in set $P_t(n + 1)$ because $P_t(n) \subseteq P_t(n + 1)$. Hence if a hit occurs with capacity n, a hit also occurs when the capacity is increased to $n + 1$. This inclusion property does not hold for all replacement policies. The example in Figure 4.59 shows that increasing n from 3 to 4 frames in a system with FIFO replacement policy increases the page fault frequency in this case from 0.75 to 0.83. However, this phenomenon is relatively rare, not occurring for most reference sequences.

4.8.7. Memory Management in the Intel Architecture

4.8.7.1. Memory Management Overview

The memory management system of the *Intel Architecture* processors (*Pentium Pro*, *Pentium* II, *Pentium* III, *Pentium* 4) is divided into two parts: *segmentation* and *paging*. Segmentation provides a mechanism of isolating individual code, data, and stack modules so that multiple programs (or tasks) can run on the same processor without interfering with one another. Paging provides a mechanism for implementing a demand-paged virtual memory system, where sections of a program's execution environment are mapped into physical memory as needed. When the processor operates in protected mode, some form of segmentation must be used. There is no mode bit to disable segmentation. The use of paging, however, is optional.

The *Intel Architecture* has three address spaces: *virtual*, *linear*, and *physical*. Figure 4.61 represents the relationship between these address spaces. The segmentation unit translates a virtual address into a linear address. When the paging is not used, the lin-

ear address corresponds to the physical address. When paging is used, the paging unit translates the linear address into a physical address.

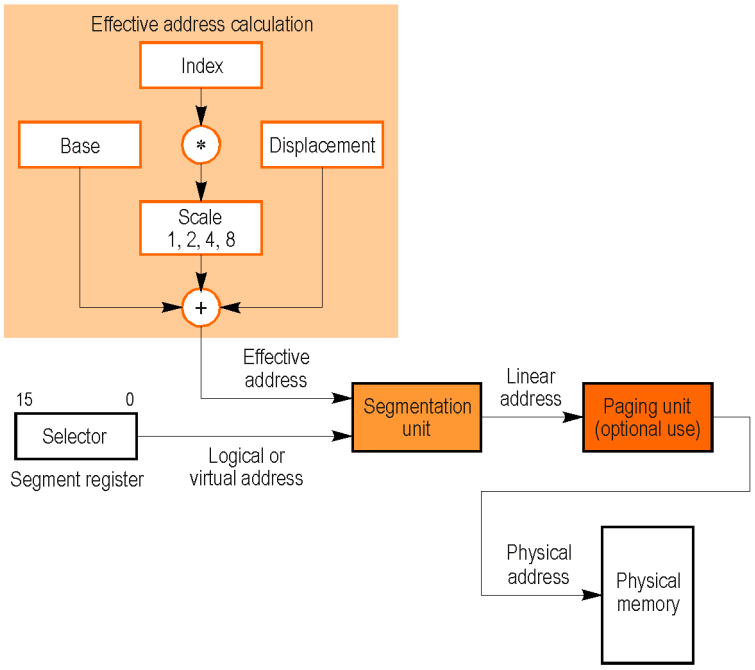

Figure 4.61. Address translation for the *Intel Architecture*.

The *virtual* or *logical* address (which represents a far pointer) consists of a 16-bit *segment selector* and a 32-bit *effective address* (or *offset*). The segment selector is a unique identifier for a segment. Among other information, the selector provides an offset into a segment descriptor table (such as the global descriptor table, GDT, or the local descriptor table, LDT), which contains data structures called *segment descriptors*. Each segment has a segment descriptor, specifying the size of the segment, the access rights and privilege level for the segment, the segment type, and the address of the first byte of the segment in the linear address space (called the base address of the segment).

The effective address is computed by adding some combination of the addressing components. There are three addressing components: *displacement*, *base*, and *index*. The displacement is an 8-bit or 32-bit immediate value following the instruction code. The base is the contents of any general-purpose register and often points to the beginning of the local variable area. The index is the contents of any general-purpose register and often is used to address the elements of an array or the characters of a string. The index may be multiplied by a scale factor (1, 2, 4, or 8) to facilitate certain addressing, such as addressing arrays or structures. As indicated in Figure 4.61, the effective address is computed as:

$$\textit{Effective Address} = \textit{base} + (\textit{index} * \textit{scale}) + \textit{displacement}$$

As shown in Figure 4.62, segmentation provides a mechanism for dividing the processor's address space (called the *linear address* space) into smaller protected address spaces called *segments*. Segments can be used to hold the code, data, and stack for a program or to hold system data structures. If more than one program is running on a processor, each program can be assigned its own set of segments. The processor then establishes the boundaries between these segments and insures that one program does not interfere with the execution of another program by writing into the other program's segments. The operations that may be performed on a particular type of segment can be restricted.

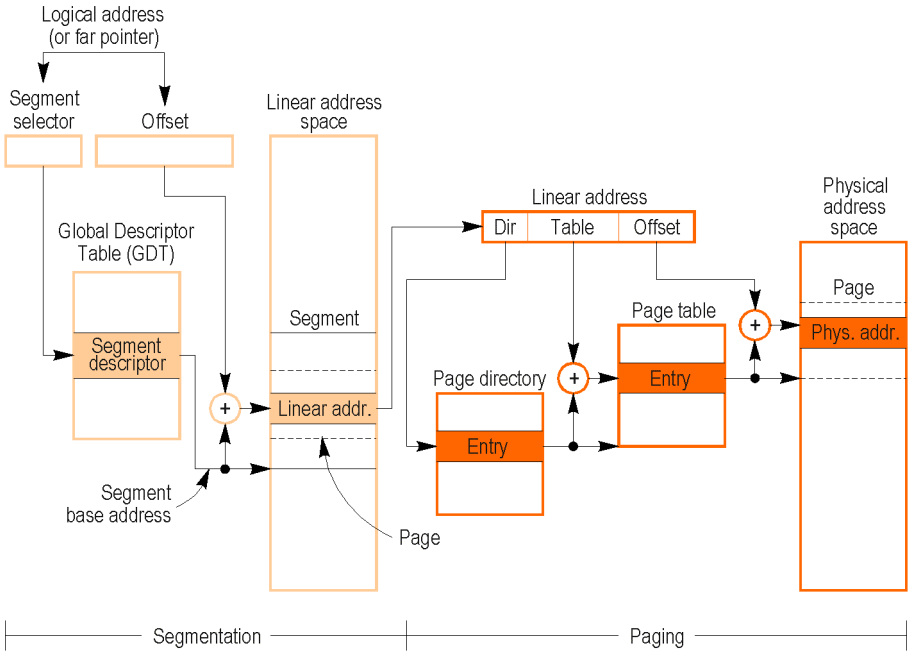

Figure 4.62. Segmentation and paging for the *Intel Architecture*.

All of the segments within a system are contained in the processor's linear address space. The size of a segment can vary from 1 byte to the maximum size of the main memory, 4 GB (2^{32} bytes). To locate a byte in a particular segment, a logical address must be provided. The segmentation unit adds the base address of the segment to the offset part of the logical address (i.e., the effective address) to form a 32-bit linear address in the processor's linear address space.

If paging is not used, the linear address space of the processor is mapped directly into the *physical address* space of processor. The physical address space is defined as the range of addresses that the processor can generate on its address bus.

When using paging, each segment is divided into pages (ordinarily 4 KB each in size), which are stored either in physical memory or on the disk. The operating system maintains a *page directory* and a set of *page tables* to keep track of the pages.

When a program (or task) attempts to access a location in the linear address space, the processor uses the page directory and page tables to translate the linear address into a physical address and then performs the requested operation (read or write) on the memory location. If the page being accessed is not currently in physical memory, the processor interrupts execution of the program, reads the page into physical memory from the disk, and then continues executing the program.

4.8.7.2. Segmentation

The segmentation mechanism provided in the *Intel Architecture* can be used to implement a wide variety of memory systems. These systems range from flat models that make only minimal use of segmentation to protect programs, to multi-segmented models that employ segmentation to create a robust operating environment in which multiple programs and tasks can be executed reliably.

The simplest memory model for a system is the "*flat model*," in which the operating system and application programs have access to a continuous, unsegmented address space. To implement a flat model with the *Intel Architecture*, at least two segment descriptors must be created, one for code references and one for data references. Both of these segments, however, are mapped to the entire linear address space: that is, both segment descriptors have the same base address of 0 and the same segment limit of 4 GB. By setting the segment limit to 4 GB, the segmentation mechanism is kept from generating exceptions for out of limit memory references, even if no physical memory exists at a particular address. ROM (EPROM) is generally located at the top of the physical address space, because the processor begins execution of the program at the address FFFF FFF0h. RAM (DRAM) is placed at the bottom of the address space, because the initial base address for the data segment after processor reset is 0.

The *protected flat model* is like the flat model, except the segment limits are set to include only the range of addresses for which physical memory actually exists. A general-protection exception is then generated on any attempt to access non-existent memory. This model provides a minimum level of hardware protection against some kinds of program errors.

The *multi-segment model* uses the full capabilities of the segmentation mechanism to provide hardware protection of code and data. In this case, each program is given its own table of segment descriptors and its own segments. The segments can be private to their assigned programs or shared among programs. Access to all segments and to the execution environments of individual programs running on the system is controlled by hardware.

Access checks can be used to protect not only against referencing an address outside the limit of a segment, but also against performing illegal operations in certain segments. For example, if code segments are designated as read-only segments, write operations into code segments can be prevented by hardware. The access rights information created for segments can also be used to set up protection rings or levels.

Protection levels can be used to protect operating system procedures from unauthorized access by application programs.

4.8.7.3. Paging

When operating in protected mode, the *Intel Architecture* can map the linear address space directly into a large physical memory (for example, 4 GB of RAM), or indirectly (using paging) into a smaller physical memory and disk storage. The latter method of mapping the linear addresses space is referred to as demand-paged virtual memory.

When paging is used, the processor divides the linear address space into fixed-size pages (generally 4 KB in length) that can be mapped into physical memory and/or disk storage. When a program references a logical address in memory, the processor translates the address into a linear address and then uses its paging mechanism to translate the linear address into the corresponding physical address. If the page containing the linear address is not currently in physical memory, the processor generates a *page-fault exception*. This exception directs the operating system to load the page from disk storage into physical memory (perhaps writing different page from physical memory to the disk), then restart the instruction that generated the exception.

Paging is different from segmentation through its use of fixed-size pages. Unlike segments, which usually are the same size as the code or data structures they hold, pages have a fixed size. 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 can be partly in memory and partly on disk.

To minimize the number of bus cycles required for address translation, the processor holds the most recently accessed page-directory and page-table entries in cache memories called translation look-aside buffers (TLBs). The TLBs satisfy most requests for reading the current page directory and page tables without requiring a bus cycle. Extra bus cycles occur only when the TLBs do not contain a page-table entry, which typically happens when a page has not been accessed for a long time.

The information that the processor uses to translate linear addresses into physical addresses is contained in four data structures:

- *Page directory*: A table of 32-bit entries contained in a 4-KB page. Up to 1024 page-directory entries (PDEs) can be held in a page directory.
- *Page table*: A table of 32-bit entries contained in a 4-KB page. Up to 1024 page-table entries (PTEs) can be held in a page table. Page tables are not used for 2-MB and 4-MB pages. These pages are mapped directly from one or more page directories.
- *Page*: A 4-KB, 2-MB, or 4-MB flat address space.
- *Page Directory Pointer Table*: A table of four 64-bit entries, each of which containing a pointer to a page directory. This data structure is only used when

the physical address extension (PAE) is enabled. The PAE flag, located in bit 5 of control register CR4, enables an extension of physical addresses in the *Intel Architecture* from 32 bits to 36 bits. The processor provides 4 additional pins for the additional address bits. This option can only be used when paging is enabled.

These tables provide access to either 4-KB or 4-MB pages when normal 32-bit physical addressing is being used and to either 4-KB or 2-MB pages when extended 36-bit physical addressing is being used.

Figure 4.63 shows the page directory and page table hierarchy when mapping linear addresses to 4-KB pages. The entries in the page directory point to page tables, and the entries in a page table point to pages in physical memory. This paging method can be used to address up to 2^{20} pages, which spans a linear address space of 2^{32} bytes (4 GB).

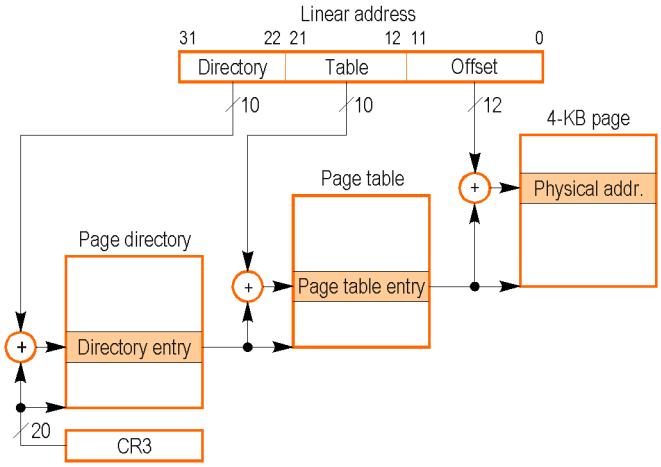

Figure 4.63. Linear address translation for the *Intel Architecture* (4-KB pages).

To select the various table entries, the linear address is divided into three sections:

- *Directory*: Bits 22 through 31 of the linear address contain an offset to an entry in the page directory. The selected entry provides the base physical address of a page table.
- *Table*: Bits 12 through 21 contain an offset to an entry in the selected page table. This entry provides the base physical address of a page in physical memory.
- *Offset*: Bits 0 through 11 contain an offset in the page.

Memory management software has the possibility to use one page directory for all tasks, one page directory for each task, or a combination of the two.

Figure 4.64 shows how a page directory can be used to map linear addresses to 4-MB pages. The entries in the page directory point to page tables, and the entries in a page table point to 4-MB pages in physical memory. This paging method can be used to map up to 1024 pages into a 4 GB linear address space. In this case, the linear address is divided into two sections:

- *Directory*: Bits 22 through 31 contain an offset to an entry in the page directory. The selected entry provides the base physical address of a 4-MB page.
- *Offset*: Bits 0 through 21 contain an offset in the page.

It is possible to access both page tables for 4-KB pages and 4-MB pages from the same page directory. A typical example of mixing 4-KB and 4-MB pages is to place the operating system or executive's kernel in a large page to reduce TLB misses and thus improve overall system performance. The processor maintains 4KB-page entries and 4-MB page entries in separate TLBs. So, placing often used code, such as the operating system's kernel, in a large page, frees up 4 KB-page TLB entries for application programs and tasks and for less frequently used utilities.

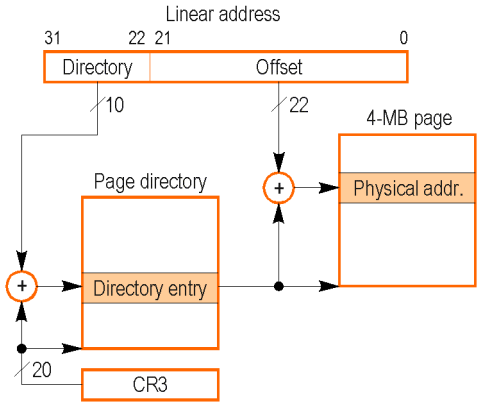

Figure 4.64. Linear address translation for the *Intel Architecture* (4-MB pages).

4.9. Problems

4.9.1. Using memory units of 4×2 bits of the type shown in Figure 4.65, design a 16×4 random-access memory unit.

4.9.2. Consider the generic 1D RAM organization illustrated in Figure 4.6. Assume the storage array is implemented by the DRAM cell of Figure 4.4(b). Describe three ways in which the RAM can be modified to double its transfer rate.

4.9.3. Using the 64-Mbit DRAM (8E1) presented in Section 4.4.4 as the basic component, design a 256M $\times$ 32-bit DRAM. Draw a diagram of the memory in the style of Figures 4.9 and 4.10.