Chapter 13
Virtual Memory Technologies and Virtual Addressing

Comer, D. (2017). *Essentials of Computer Architecture* (2nd ed.). CRC Press.

Mano, M. M., Kim, C. R. & Martin, T. (2015). *Logic and Computer Design Fundamentals* (5th ed.). Pearson.

Contents

1. Definition of Virtual Memory

- *Virtual Memory* refers to a mechanism that hides the details of the underlying physical memory to provide a more convenient memory environment.
- A virtual memory system creates an illusion – an address space and a memory access scheme that overcome limitations of the physical memory and physical addressing scheme.

2. Memory Management Unit and Address Space

- *Memory Management Unit* (*MMU*) describes an memory controller. An MMU creates a *virtual address space* for the processor.
  - The addresses a processor uses are *virtual addresses* because the MMU translates each address into an underlying physical memory.
- An MMU that can map from byte addresses to underlying word addresses can be extended to create more complex memory organizations.

2. Memory Management Unit and Address Space

- For example, Intel designed a network processor that used two types of physical memory: SRAM and DRAM.
  - Recall that SRAM is faster than DRAM, but costs more, so the system had a smaller amount of SRAM (intended for items that were accessed frequently) and a large amount of DRAM (intended for items that were not accessed frequently).
- The SRAM physical memory was organized with four bytes per word and the DRAM physical memory was organized with eight bytes per word.
- Intel's network processor used an embedded RISC processor that could access both memories.
  - It integrated both physical memories into a single virtual address space.

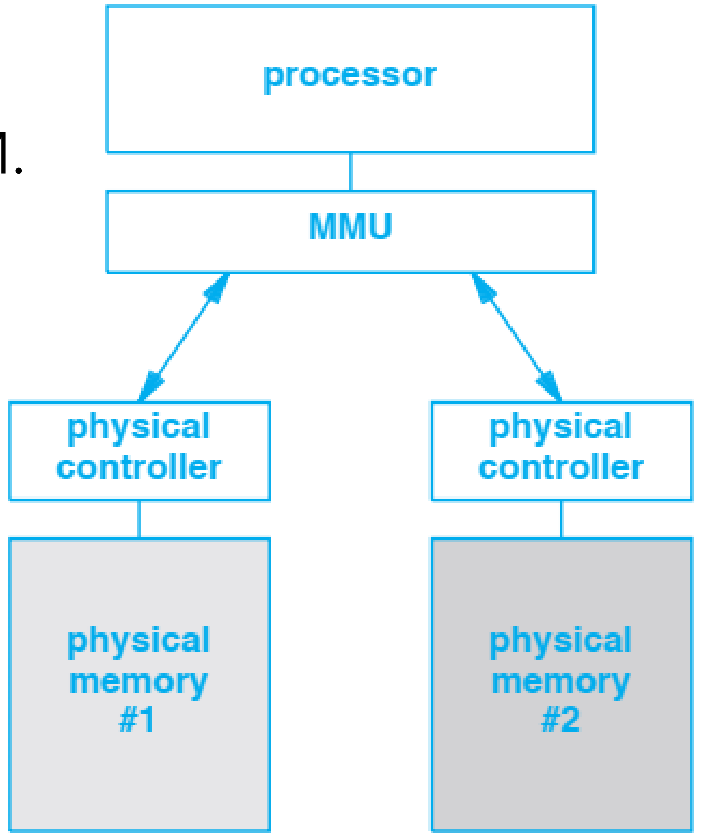


Figure 13.1 Illustration of an architecture in which two dissimilar memories connect to a processor. The processor can use either memory.

2. Memory Management Unit and Address Space

- The MMU divides the address space into two parts, which the MMU associates with physical memory 1 and physical memory 2.
  - For example, if each physical memory contains a gigabyte (0x40000000 bytes) of RAM, the MMU can create a virtual address space that maps addresses 0 through 0x3fffffff to the first memory and addresses 0x40000000 through 0x7fffffff to the second memory.

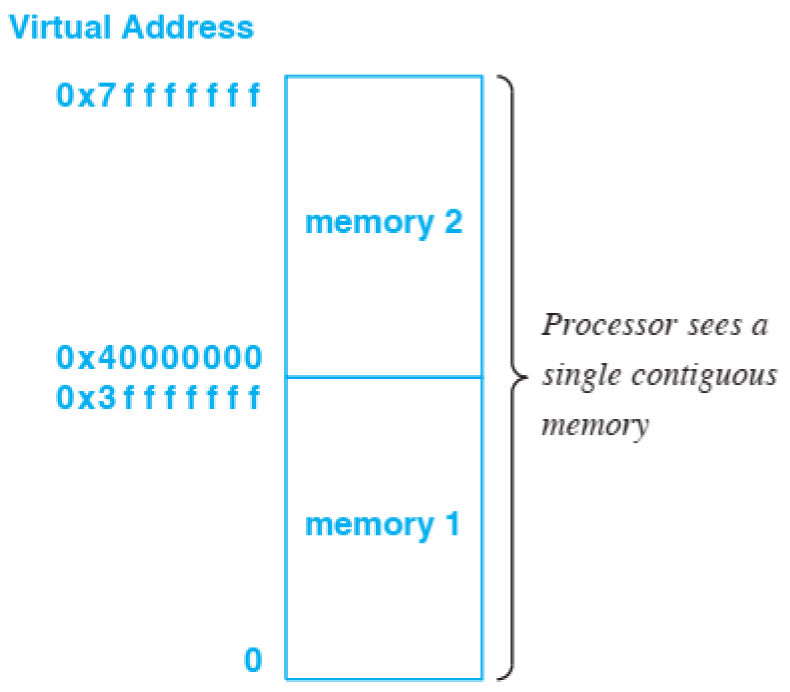


Figure 13.2 Illustration of a virtual memory system that divides an address space among two physical memories†. The MMU uses an address to decide which memory to access.

3. Address Translation or Address Mapping

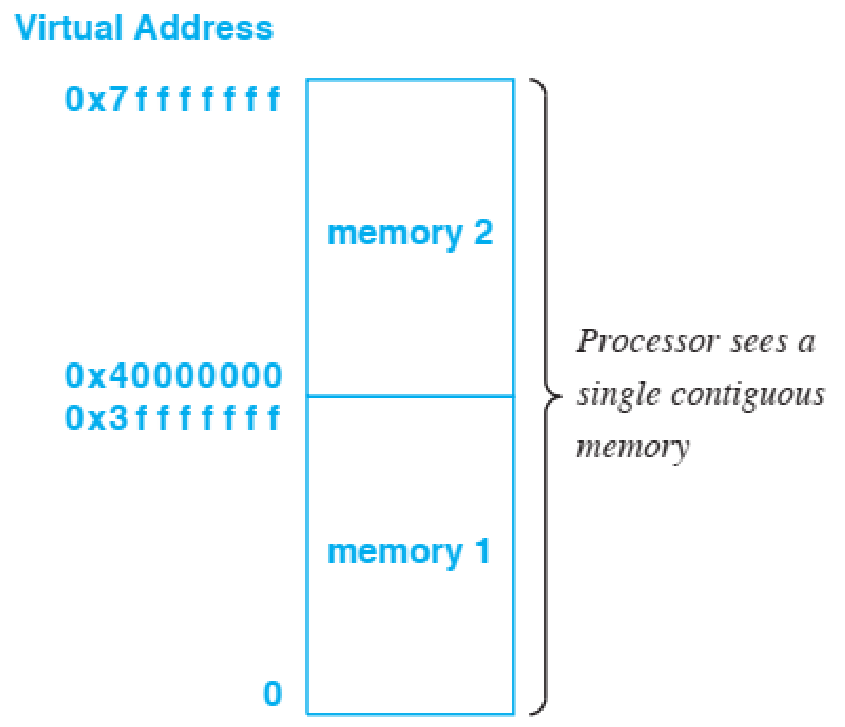


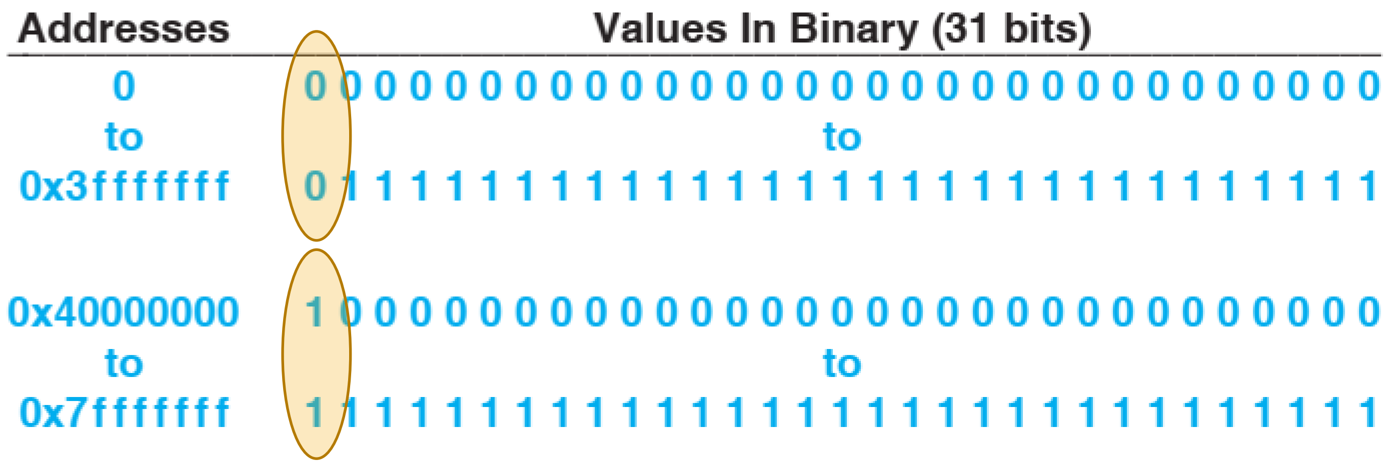
Figure 13.2 Illustration of a virtual memory system that divides an address space among two physical memories†. The MMU uses an address to decide which memory to access.

```
Receive a virtual memory request from processor;
Let V be the address in the request;
if (V < 0x40000000) {
        Pass the unmodified request (address V) to memory 1;
} else {     /* map the address for memory 2 */
        V2 = V – 0x40000000;
        Pass the modified request (address V2) to memory 2;
}
```

Figure 13.3 The sequence of steps used by a Memory Management Unit to create the virtual memory depicted in Figure 13.2. The MMU maps the virtual address space onto two physical memories.

3. Address Translation or Address Mapping

- In practice, an MMU does not use subtraction to implement address translation because subtraction requires substantial hardware (e.g., an ALU) and takes too much time to perform for each memory reference. **The solution consists of using powers of two to simplify the hardware.**



Addresses	Values In Binary (31 bits)
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
to	to
0x3fffffff	0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0x40000000	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
to	to
0x7fffffff	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Figure 13.4 The binary values for addresses in the range 0 through 2 gigabytes. Except for the high-order bit, values above 1 gigabyte are the same as those below.

4. Multiple Virtual Spaces and Multiprogramming

- The most common technology for multiprogramming uses virtual memory to establish a separate virtual address space for each program.
- The mechanism in the figure divides the physical memory into equal-size areas that are known as *partitions*.

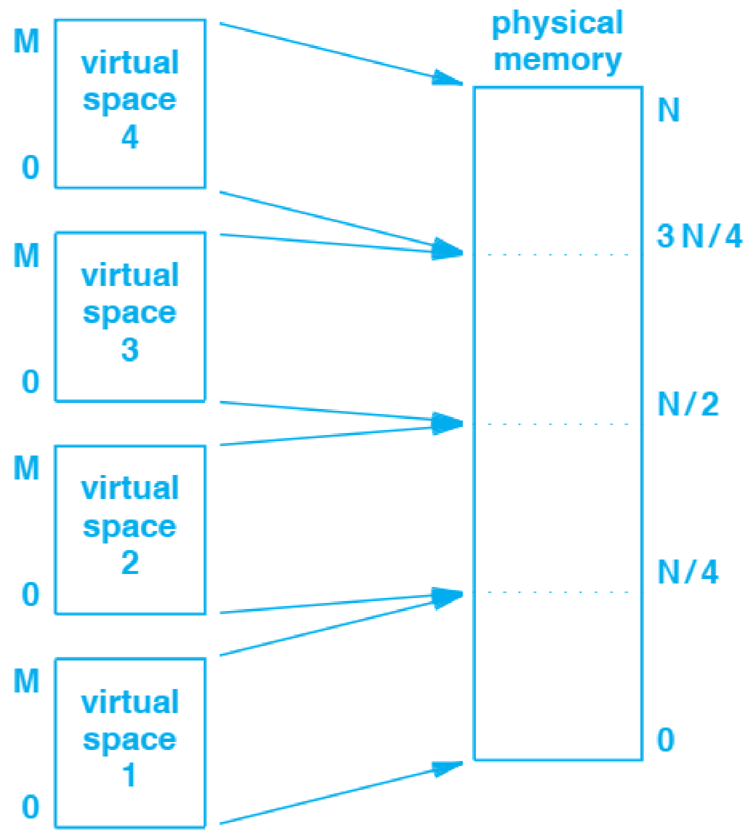


Figure 13.6 Illustration of four partitions mapped onto a single physical memory. Each virtual address space starts at address zero.

5. Creating Virtual Spaces Dynamically

- MMU in a general-purpose system can be changed dynamically at run time. That is, when the system boots, the processor tells the MMU exactly how to map the virtual address space onto the physical memory.
- In general, the address space to be used is part of the *processor mode*. **The processor begins running in *real mode*, which means that the processor passes all memory references directly to the physical memory without using the MMU.**
- While operating in real mode, the processor can interact with the MMU to establish a mapping.
- Once a mapping has been specified, the processor can execute an instruction that changes the mode, enables the MMU, and branches to a specified location.
- The MMU translates each memory reference according to the mapping that was configured.

Base-Bound Registers

- A mechanism known by the name *base-bound* is among the oldest and easiest dynamic virtual memory schemes to understand.
- In essence, the *base-bound* scheme creates a single virtual address space and maps the space onto a region of physical memory.
- The name refers to a pair of registers that are part of the MMU; both must be loaded before the MMU is enabled.
    - The *base* register holds an address in physical memory that specifies where to map the virtual address space
    - The *bound* register holds an integer that specifies the size of the address space.

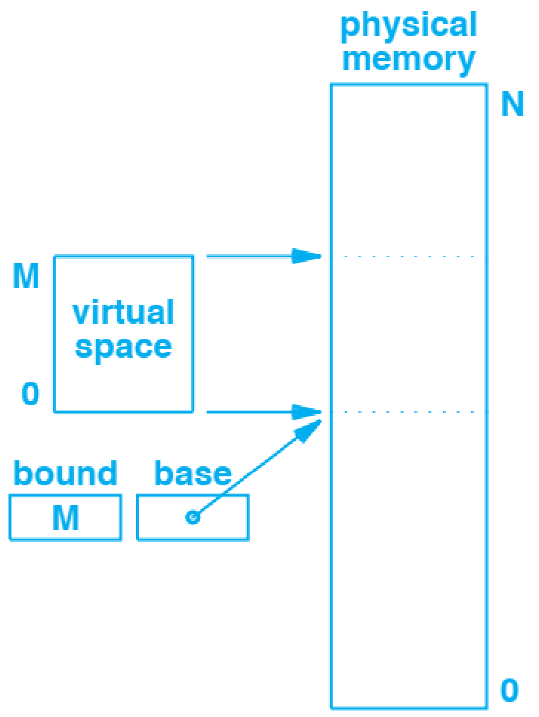


Figure 13.7 Illustration of a virtual memory that uses a base-bound mechanism. The base register specifies the location of the virtual address space, and the bound register specifies the size.

Base-Bound Registers

- Changing The Virtual Space
  - The base-bound mechanism is dynamic (i.e., easy to change). The idea is that an operating system can use the base-bound mechanism to move among multiple virtual address spaces.
  - For example, suppose the operating system has loaded two application programs at different locations in memory. The operating system, which runs in real mode, controls the MMU.
    - When an application, A, is ready to run, the operating system configures the MMU to point to A's section of the memory, enables the MMU mapping, and branches to the application.
    - Later, when control returns to the operating system, the operating system selects another application to run, B, configures the MMU to point to B's memory, enables the MMU, and branches to the code for B.
    - Each application's virtual address space starts at zero; the application remains unaware of its location in physical memory.

Base-Bound Registers

- Virtual Memory and Protection
  - The base-bound scheme uses the bound register to guarantee that a program will not exceed its allocated space.
  - Of course, to implement protection, the MMU must check each memory reference and raise an error if the program attempts to reference an address greater than *M*.
  - The protection offered by a base-bound mechanism provides an example of an important concept:
    - A virtual memory system that supports multiprogramming must also provide protection that prevents one application from reading or altering memory that has been allocated to another application.

Segmentation

- The memory mappings described above are intended to map a **complete address space** (i.e., all memory that is needed for an application to run, including the compiled program and the data the program uses). We say that a virtual memory technology that maps an entire address space is a *coarse granularity mapping*.
- The alternative, which consists of mapping **parts of an address space**, is known as a *fine granularity mapping*.

Segmentation

- To reduce the amount of memory needed, the architects proposed that each program be divided into **variable-size blocks**, and only the blocks of the program that are needed at any time be loaded in memory.
- That is, *pieces* of the program are kept on an external storage device, typically a disk, until one of them is needed. At that time, the operating system finds an unused region of memory that is large enough, and loads the *piece* into memory.
- The operating system then configures the MMU to establish the mapping between the virtual addresses that the piece uses and the physical addresses used to hold the piece.
- The variable-size piece scheme is known as *segmentation*, and the pieces of programs are known as *segments*.

Segmentation

- The central problem with segmentation arises after an operating system begins to move blocks in and out of memory. Because **segments are variable size**, the memory tends toward a situation in which the unused memory is divided into many small blocks. Computer scientists use the term *fragmentation* to describe the situation, and say that the memory becomes *fragmented*.

Demand Paging

- An alternative to segmentation was invented that has become extremely successful.
- *Demand paging* follows the same general scheme as segmentation: divide a program into pieces, keep the pieces on external storage until they are needed, and load an individual piece when the piece is referenced.
- The most significant difference between demand paging and segmentation lies in how the program is divided. Instead of variable-size segments that are large enough to hold a complete function, demand paging uses **fixed-size blocks called *pages***.
- Initially, when memories and application programs were much smaller, architects chose a page size of 512 bytes or 1 Kbyte; current architectures use larger page sizes (e.g., Intel processors use 4 Kbyte pages).

Demand Paging

- Paging Terminology and Data Structures
  - The term ***page*** refers to a block of a program's address space, and the term ***frame*** refers to a slot in physical memory that can hold a page. Thus, we say that software loads a page into a frame of memory.
  - When a page is in memory, we say that the page is *resident*, and the set of all pages from an address space that are currently in memory is called the *resident set*.

Demand Paging

- Paging Terminology and Data Structures
  - The primary data structure used for demand paging is known as a *page table*.
  - The easiest way to envision a page table is to imagine a one-dimensional array that is indexed by a page number. That is, entries in the table have index zero, one, and so on.
  - Each entry in the page table either contains a null value (if the page is not resident) or the address of the frame in physical memory that currently holds the page.

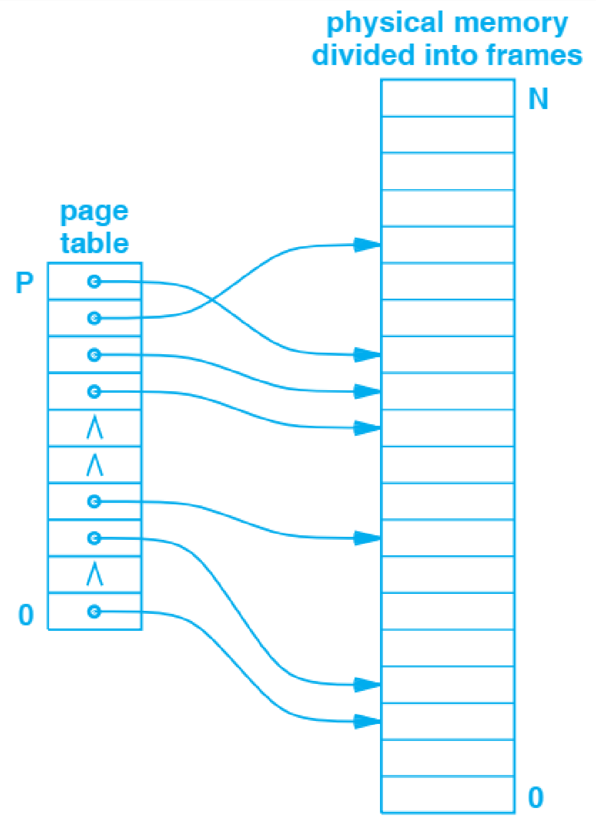


Figure 13.8 Illustration of an active page table with some entries pointing to frames in memory. A null pointer in a page table entry (denoted by Λ) means the page is not currently resident in memory.

Demand Paging

- Address Translation in a Paging System
  - Imagine an address space divided into fixed-size pages.
  - Each page contains *K* bytes, bytes on page zero have addresses zero through *K*–1, bytes on page 1 have addresses *K* through 2*K*–1, and so on.

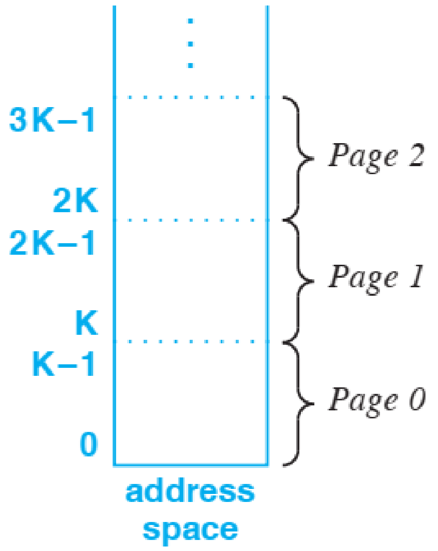


Figure 13.9 Illustration of a virtual address space divided into pages of K bytes per page.

Demand Paging

- Translation of a virtual address *V* to physical address *P*:
  - $page\ number = N = \lfloor V/K \rfloor$
  - $offset = O = V\ modulo\ K$
  - $physical\ address = P = pagenumber[N] + O$
- The division and modulo operations specified in the mathematical equations can be replaced by extracting bits if using a power of 2.

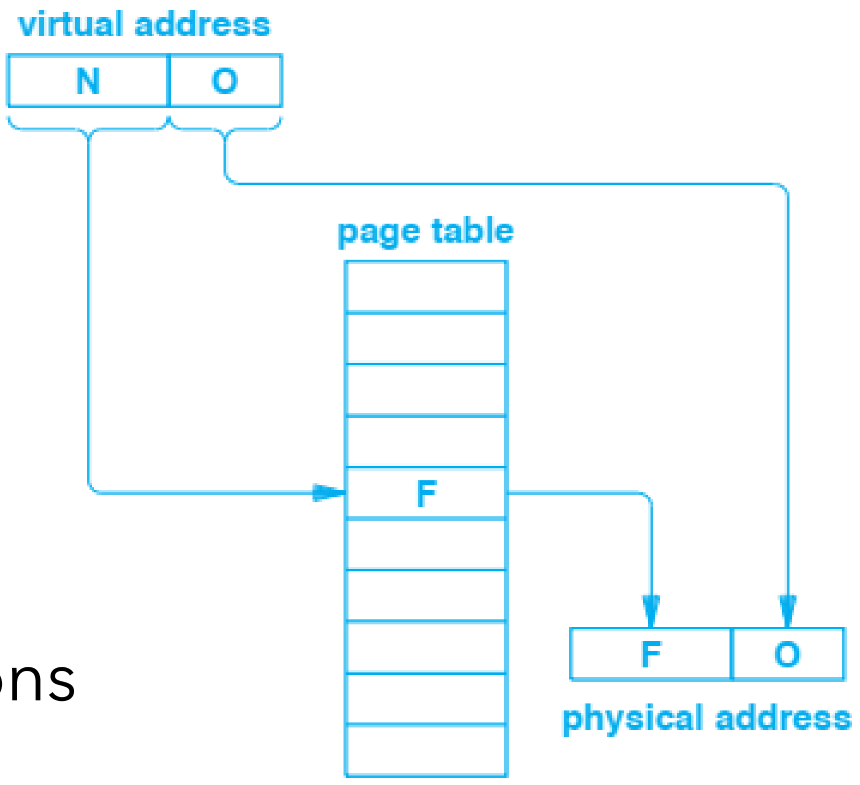


Figure 13.10 Illustration of how an MMU performs address translation on a paging system. Making the page size a power of two eliminates the need for division and remainder computation.