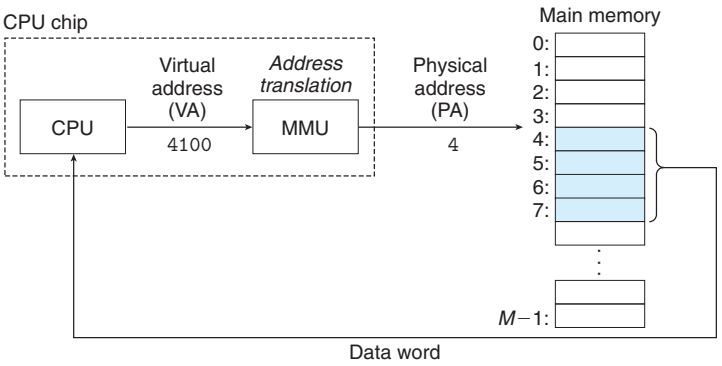

Figure 9.2
A system that uses virtual addressing.

With virtual addressing, the CPU accesses main memory by generating a *virtual address (VA)*, which is converted to the appropriate physical address before being sent to main memory. The task of converting a virtual address to a physical one is known as *address translation*. Like exception handling, address translation requires close cooperation between the CPU hardware and the operating system. Dedicated hardware on the CPU chip called the *memory management unit (MMU)* translates virtual addresses on the fly, using a lookup table stored in main memory whose contents are managed by the operating system.

9.2 Address Spaces

An *address space* is an ordered set of nonnegative integer addresses

$$\{0, 1, 2, \ldots\}$$

If the integers in the address space are consecutive, then we say that it is a *linear address space*. To simplify our discussion, we will always assume linear address spaces. In a system with virtual memory, the CPU generates virtual addresses from an address space of $N = 2^n$ addresses called the *virtual address space*:

$$\{0, 1, 2, \ldots, N - 1\}$$

The size of an address space is characterized by the number of bits that are needed to represent the largest address. For example, a virtual address space with $N = 2^n$ addresses is called an n-bit address space. Modern systems typically support either 32-bit or 64-bit virtual address spaces.

A system also has a *physical address space* that corresponds to the M bytes of physical memory in the system:

$$\{0, 1, 2, \ldots, M - 1\}$$

M is not required to be a power of 2, but to simplify the discussion, we will assume that $M = 2^m$.

The concept of an address space is important because it makes a clean distinction between data objects (bytes) and their attributes (addresses). Once we recognize this distinction, then we can generalize and allow each data object to have multiple independent addresses, each chosen from a different address space. This is the basic idea of virtual memory. Each byte of main memory has a virtual address chosen from the virtual address space, and a physical address chosen from the physical address space.

Practice Problem 9.1 **(solution page 916)**

Complete the following table, filling in the missing entries and replacing each question mark with the appropriate integer. Use the following units: K $= 2^{10}$ (kilo), M $= 2^{20}$ (mega), G $= 2^{30}$ (giga), T $= 2^{40}$ (tera), P $= 2^{50}$ (peta), or E $= 2^{60}$ (exa).

Number of virtual address bits (n)	Number of virtual addresses (N)	Largest possible virtual address
4		
	$2^? = 16$ K	
		$2^{24} - 1 = ?$ M $- 1$
	$2^? = 64$ T	
54		

9.3 VM as a Tool for Caching

Conceptually, a virtual memory is organized as an array of N contiguous byte-size cells stored on disk. Each byte has a unique virtual address that serves as an index into the array. The contents of the array on disk are cached in main memory. As with any other cache in the memory hierarchy, the data on disk (the lower level) is partitioned into blocks that serve as the transfer units between the disk and the main memory (the upper level). VM systems handle this by partitioning the virtual memory into fixed-size blocks called *virtual pages (VPs)*. Each virtual page is $P = 2^p$ bytes in size. Similarly, physical memory is partitioned into *physical pages (PPs)*, also P bytes in size. (Physical pages are also referred to as *page frames*.)

At any point in time, the set of virtual pages is partitioned into three disjoint subsets:

Unallocated. Pages that have not yet been allocated (or created) by the VM system. Unallocated blocks do not have any data associated with them, and thus do not occupy any space on disk.

Cached. Allocated pages that are currently cached in physical memory.

Uncached. Allocated pages that are not cached in physical memory.

The example in Figure 9.3 shows a small virtual memory with eight virtual pages. Virtual pages 0 and 3 have not been allocated yet, and thus do not yet exist