In most cases, a user program goes through several steps some of which may be optional before being executed (Figure 7.3). Addresses may be represented in different ways during these steps. Addresses in the source program are generally symbolic (such as the variable count). A compiler typically **binds** these symbolic addresses to relocatable addresses (such as "14 bytes from the beginning of this module"). The linkage editor or loader in turn binds the relocatable addresses to absolute addresses (such as 74014). Each binding is a mapping from one address space to another. Classically, the binding of instructions and data to memory addresses can be done at any step along the way:

• **Compile time**. If you know at compile time where the process will reside in memory, then **absolute code** can be generated. For example, if you know that a user process will reside starting at location *R*, then the generated compiler code will start at that location and extend up from there. If, at some later time, the starting location changes, then it will be necessary to recompile this code. The MS-DOS .COM-format programs are bound at compile time.

• **Load time**. If it is not known at compile time where the process will reside in memory, then the compiler must generate **relocatable code**. In this case, final binding is delayed until load time. If the starting address changes, we need only reload the user code to incorporate this changed value.

• **Execution time**. If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time. Special hardware must be available for this scheme to work. Most general-purpose operating systems use this method.

A major portion of this chapter is devoted to showing how these various bindings can be implemented effectively in a computer system and to discussing appropriate hardware support.

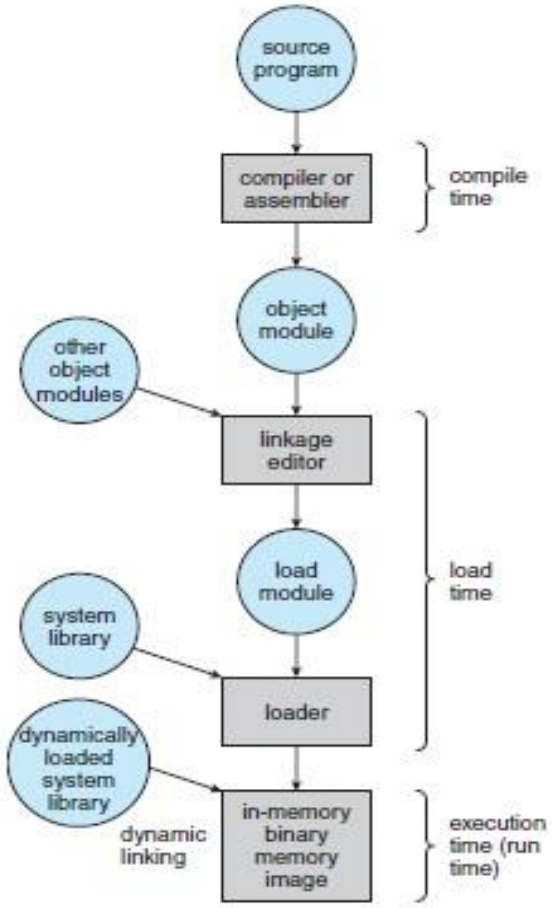

Figure 7-3 Multistep processing of a user program

7.1.3. Logical Versus Physical Address Space

An address generated by the CPU is commonly referred to as a **logical address**, whereas an address seen by the memory unit that is, the one loaded into the **memory address register** of the memory is commonly referred to as a **physical address**. The compile-time and load-time address-binding methods generate identical logical and physical addresses. However, the execution-time address binding scheme results in differing logical and physical addresses. In this case, we usually refer to the logical address as a **virtual address**. We use ***logical address*** and ***virtual address*** interchangeably in this text. The set of all logical addresses generated by a program is a **logical address space**. The set of all physical addresses corresponding to these logical addresses is a **physical address space**. Thus, in the execution-time address-binding scheme, the logical and physical address spaces differ.

The run-time mapping from virtual to physical addresses is done by a hardware device called the **memory-management unit (MMU)**. The base register is now called a **relocation register**. The value in the relocation register is added to every address generated by a user process at the time the address is sent to memory (see Figure 7.4). For example, if the base is at 14000, then an attempt by the user to address location 0 is dynamically relocated to location14000; an access to location346 is mapped to location 14346.

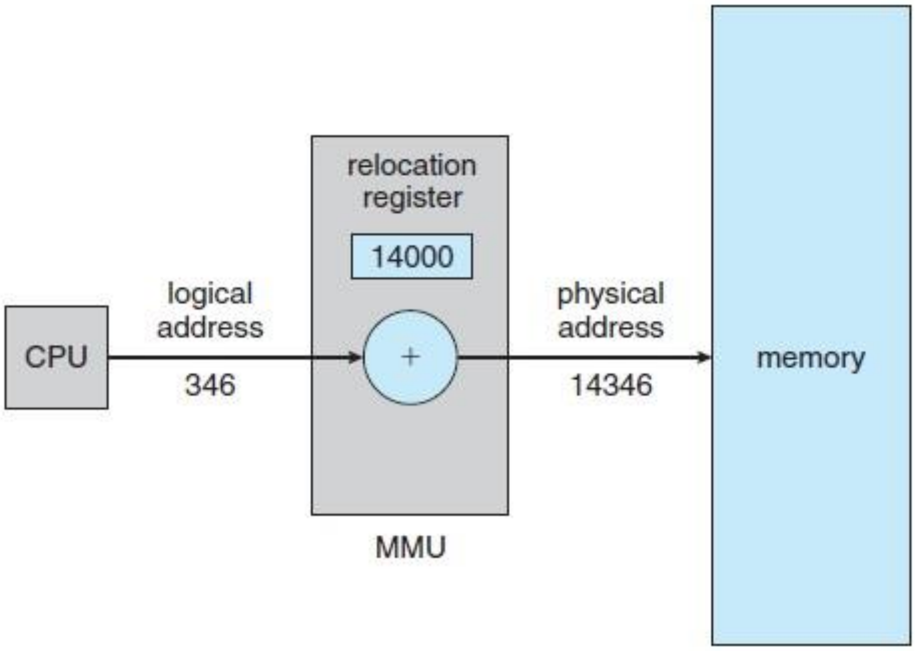

Figure 7-4 Dynamic relocation using a relocation register

The user program never sees the real physical addresses. The program can create a pointer to location 346, store it in memory, manipulate it, and compare it with other addresses all as the number 346. Only when it is used as a memory address (in an indirect load or store, perhaps) is it relocated relative to the base register. The user program deals with logical addresses. The memory-mapping hardware converts logical addresses into physical addresses. This form of execution-time binding was discussed in Section 8.1.2. The final location of a referenced memory address is not determined until the reference is made.

We now have two different types of addresses: logical addresses (in the range 0 to *max*) and physical addresses (in the range $R + 0$ to $R + max$ for a base value R). The user program generates only logical addresses and thinks that the process runs in locations 0 to *max*. However, these logical addresses must be mapped to physical addresses before they are used. The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.

7.2. Swapping

A process must be in memory to be executed. A process, however, can be **swapped** temporarily out of memory to a **backing store** and then brought back into memory for continued execution (Figure 8.5). Swapping makes it possible for the total physical address space of all processes to exceed the real physical memory of the system, thus increasing the degree of multiprogramming in a system.

7.2.1. Standard Swapping

Standard swapping involves moving processes between main memory and a backing store. The backing store is commonly a fast disk. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images. The system maintains a **ready queue** consisting of all processes whose memory images are on the backing store or in memory and are ready to run. Whenever the CPU scheduler decides to execute a process, it calls the dispatcher. The dispatcher checks to see whether the next process in the queue is in memory. If it is not, and if there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers and transfers control to the selected process.