Introduction to Paging

* It is sometimes said that the operating system takes one of two approaches when solving most any space-management problem
* The first approach is to chop things up into variable-sized pieces, as we saw with segmentation in virtual memory
* In virtual memory, we call this idea paging, and it goes back to an early and important system, the Atlas
* Instead of splitting up a process’s address space into some number of variable-sized logical segments, we divide it into fixed-sized units, each of which we call a page
* To help make this approach more clear, let’s illustrate it with a simple example
* Figure presents an example of a tiny address space, only 64 bytes total in size, with four 16-byte pages
* Real address spaces are much bigger, of course, commonly 32 bits and thus 4-GB of address space, or even 64 bits; in the book, we’ll often use tiny examples to make them easier to digest
* Physical memory, as shown in Figure, also consists of a number of fixed-sized slots, in this case eight page frames making for a 128-byte physical memory, also ridiculously small
* As you can see in the diagram, the pages of the virtual address space have been placed at different locations throughout physical memory; the diagram also shows the OS using some of physical memory for itself
* Probably the most important improvement will be flexibility: with a fully-developed paging approach, the system will be able to support the abstraction of an address space effectively, regardless of how a process uses the address space; we won’t, for example, make assumptions about the direction the heap and stack grow and how they are used
* Let’s imagine the process with that tiny address space is performing a memory access
* Specifically, let’s pay attention to the explicit load of the data from address into the register eax