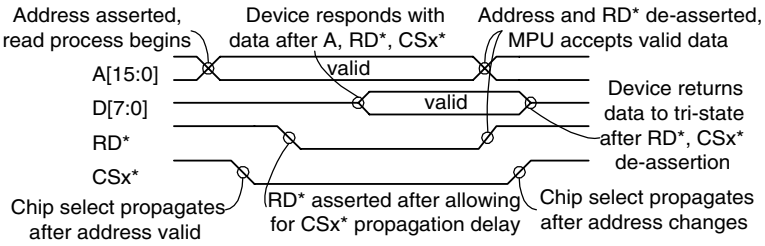

FIGURE 3.8 MPU read timing.

MPU and the devices with which it is communicating. For this interface to work properly, the MPU must allow enough time for the read to occur, regardless of the specific device with which it is communicating. In other words, it must operate according to the capabilities of the slowest device—the least common denominator.

Write timing is very similar, as seen in Fig. 3.9. Again, the MPU asserts the desired address onto A[15:0], and the appropriate chip select is decoded. At the same time, the write data is driven onto D[7:0]. Once the address and data have had time to stabilize, and after allowing time for the chip select to propagate, the WR* enable signal is asserted to actually trigger the write. The WR* signal is de-asserted while data, address, and chip select are still stable so that there is no possibility of writing to a different location and corrupting data. If the WR* signal is de-asserted at the same time as the others, a race condition could develop wherein a particular device may sense the address (or data or chip select) change just prior to WR* changing, resulting in a false write to another location or to the current location with wrong data. Being an asynchronous interface, the duration of all signal assertions must be sufficient for all devices to properly execute the write.

An MPU interrupt signal is asserted by the serial port controller to enable easier programming of the serial port communication routine. Rather than having software continually poll the serial port to see if data are waiting, the controller is configured to assert INTR* whenever a new byte arrives. The MPU is then able to invoke an ISR, which can transfer the data byte from the serial port to the RAM. The interrupt also helps when transmitting data, because the speed of the typical serial port (often 9,600 to 38,400 bps) is very slow as compared to the clock speed of even a slow MPU (1 to 10 MHz). When the software wants to send a set of bytes out the serial port, it must send one byte and then wait a relatively long time until the serial port is ready for the next byte. Instead of polling in a loop between bytes, the serial port controller asserts INTR* when it is time to send the next byte. The ISR can then respond with the next byte and return control to the main program that is run-

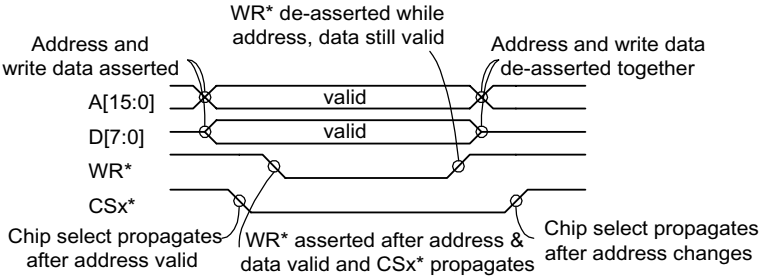

FIGURE 3.9 MPU write timing.

ning at the time. Each time INTR* is asserted and the ISR responds, the ISR must be sure to clear the interrupt condition in the serial port. Depending on the exact serial port device, a read or write to a specific register will clear the interrupt. If the interrupt is not cleared before the ISR issues a return-from-interrupt, the MPU may be falsely interrupted again for the same condition.

This computer contains two other functional elements: the clock and reset circuits. The 1-MHz clock must be supplied to the MPU continually for proper operation. In this example design, no other components in the computer require this clock. For fairly simple computers, this is a realistic scenario, because the buses and memory devices operate asynchronously. Many other computers, however, have synchronous buses, and the microprocessor clock must be distributed to other components in the system.

The reset circuit exists to start the MPU when the system is first turned on. Reset must be applied for a certain minimum duration after the power supply has stabilized. This is to ensure that the digital circuits properly settle to known states before they are released from reset and allowed to begin normal operation. As the computer is turned on, the reset circuit actively drives the RST* signal. Once power has stabilized, RST* is de-asserted and remains in this state indefinitely.

3.6 ADDRESS BANKING

A microprocessor's address space is normally limited by the width of its address bus, but supplemental logic can greatly expand address space, subject to certain limitations. Address banking is a technique that increases the amount of memory a microprocessor can address. If an application requires 1 MB of RAM for storing large data structures, and an 8-bit microprocessor is used with a 64-kB address space, address banking can enable the microprocessor to access the full 1 MB one small section at a time.

Address banking, also known as *paging*, takes a large quantity of memory, divides it into multiple smaller banks, and makes each bank available to the microprocessor one at a time. A *bank address register* is maintained by the microprocessor and determines which bank of memory is selected at any given time. The selected bank is accessed through a portion of the microprocessor's fixed address space, called a *window*, set aside for banked memory access. As shown in Fig. 3.10a, the upper 16 kB of address space provides direct access to one of many 16-kB pages in the larger banked memory structure. Figure 3.10b shows the logical implementation of this banked memory scheme. A

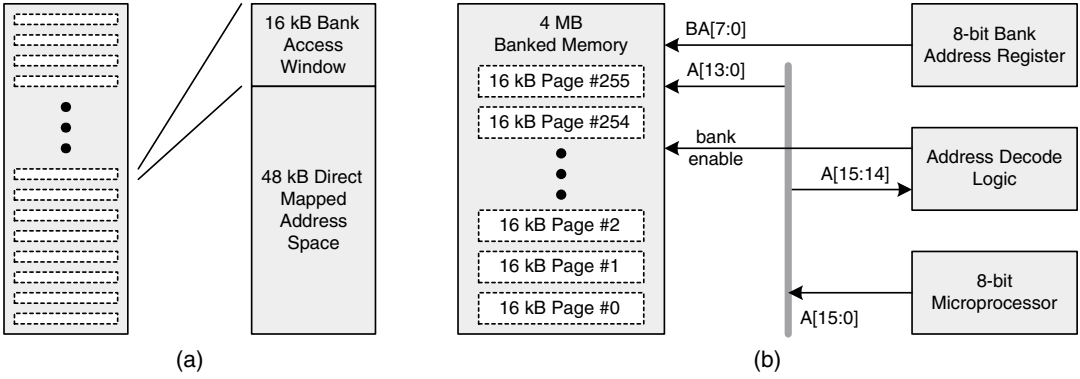

FIGURE 3.10 Address banking.