"When you wish to produce a result by means of an instrument, do not allow yourself to complicate it."

—Leonardo da Vinci

CHAPTER 4 MARIE: An Introduction to a Simple Computer

4.1 INTRODUCTION

Designing a computer nowadays is a job for a computer engineer with plenty of training. It is impossible in an introductory textbook such as this (and in an introductory course in computer organization and architecture) to present everything necessary to design and build a working computer such as those we can buy today. However, in this chapter, we first look at a very simple computer called MARIE: a **M**achine **A**rchitecture that is **R**eally **I**ntuitive and **E**asy. We then provide brief overviews of Intel and MIPs machines, two popular architectures reflecting the CISC and RISC design philosophies. The objective of this chapter is to give you an understanding of how a computer functions. We have, therefore, kept the architecture as uncomplicated as possible, following the advice in the opening quote by Leonardo da Vinci.

4.2 CPU BASICS AND ORGANIZATION

From our studies in Chapter 2 (data representation) we know that a computer must manipulate binary-coded data. We also know from Chapter 3 that memory is used to store both data and program instructions (also in binary). Somehow, the program must be executed and the data must be processed correctly. The **central processing unit** (**CPU**) is responsible for fetching program instructions, decoding each instruction that is fetched, and performing the indicated sequence of operations on the correct data. To understand how computers work, you must first become familiar with their various components and the interaction among these

components. To introduce the simple architecture in the next section, we first examine, in general, the microarchitecture that exists at the control level of modern computers.

All computers have a CPU that can be divided into two pieces. The first is the **datapath**, which is a network of storage units (registers) and arithmetic and logic units (for performing various operations on data) connected by buses (capable of moving data from place to place) where the timing is controlled by clocks. The second CPU component is the **control unit**, a module responsible for sequencing operations and making sure the correct data are where they need to be at the correct time. Together, these components perform the tasks of the CPU: fetching instructions, decoding them, and finally performing the indicated sequence of operations. The performance of a machine is directly affected by the design of the datapath and the control unit. Therefore, we cover these components of the CPU in detail in the following sections.

4.2.1 The Registers

Registers are used in computer systems as places to store a wide variety of data, such as addresses, program counters, or data necessary for program execution. Put simply, a **register** is a hardware device that stores binary data. Registers are located on the processor so information can be accessed very quickly. We saw in Chapter 3 that D flip-flops can be used to implement registers. One D flip-flop is equivalent to a 1-bit register, so a collection of D flip-flops is necessary to store multi-bit values. For example, to build a 16-bit register, we need to connect 16 D flip-flops together. We saw in our binary counter figure from Chapter 3 that these collections of flip-flops must be clocked to work in unison. At each pulse of the clock, input enters the register and cannot be changed (and thus is stored) until the clock pulses again.

Data processing on a computer is usually done on fixed-size binary words stored in registers. Therefore, most computers have registers of a certain size. Common sizes include 16, 32, and 64 bits. The number of registers in a machine varies from architecture to architecture, but is typically a power of 2, with 16 and 32 being most common. Registers contain data, addresses, or control information. Some registers are specified as "special purpose" and may contain only data, only addresses, or only control information. Other registers are more generic and may hold data, addresses, and control information at various times.

Information is written to registers, read from registers, and transferred from register to register. Registers are not addressed in the same way memory is addressed (recall that each memory word has a unique binary address beginning with location 0). Registers are addressed and manipulated by the control unit itself.

In modern computer systems, there are many types of specialized registers: registers to store information, registers to shift values, registers to compare values, and registers that count. There are "scratchpad" registers that store temporary values, index registers to control program looping, stack pointer registers to man-

age stacks of information for processes, status (or flag) registers to hold the status or mode of operation (such as overflow, carry, or zero conditions), and general purpose registers that are the registers available to the programmer. Most computers have register sets, and each set is used in a specific way. For example, the Pentium architecture has a data register set and an address register set. Certain architectures have very large sets of registers that can be used in quite novel ways to speed up execution of instructions. (We discuss this topic when we cover advanced architectures in Chapter 9.)

4.2.2 The ALU

The **arithmetic logic unit** (**ALU**) carries out the logic operations (such as comparisons) and arithmetic operations (such as add or multiply) required during the program execution. You saw an example of a simple ALU in Chapter 3. Generally an ALU has two data inputs and one data output. Operations performed in the ALU often affect bits in the **status register** (bits are set to indicate actions such as whether an overflow has occurred). The ALU knows which operations to perform because it is controlled by signals from the control unit.

4.2.3 The Control Unit

The **control unit** is the "policeman" or "traffic manager" of the CPU. It monitors the execution of all instructions and the transfer of all information. The control unit extracts instructions from memory, decodes these instructions, making sure data are in the right place at the right time, tells the ALU which registers to use, services interrupts, and turns on the correct circuitry in the ALU for the execution of the desired operation. The control unit uses a **program counter** register to find the next instruction for execution and a status register to keep track of overflows, carries, borrows, and the like. Section 4.13 covers the control unit in more detail.

4.3 THE BUS

The CPU communicates with the other components via a bus. A **bus** is a set of wires that acts as a shared but common datapath to connect multiple subsystems within the system. It consists of multiple lines, allowing the parallel movement of bits. Buses are low cost but very versatile, and they make it easy to connect new devices to each other and to the system. At any one time, only one device (be it a register, the ALU, memory, or some other component) may use the bus. However, this sharing often results in a communications bottleneck. The speed of the bus is affected by its length as well as by the number of devices sharing it. Quite often, devices are divided into **master** and **slave** categories; a master device is one that initiates actions and a slave is one that responds to requests by a master.

A bus can be **point-to-point**, connecting two specific components (as seen in Figure 4.1a) or it can be a **common pathway** that connects a number of devices,

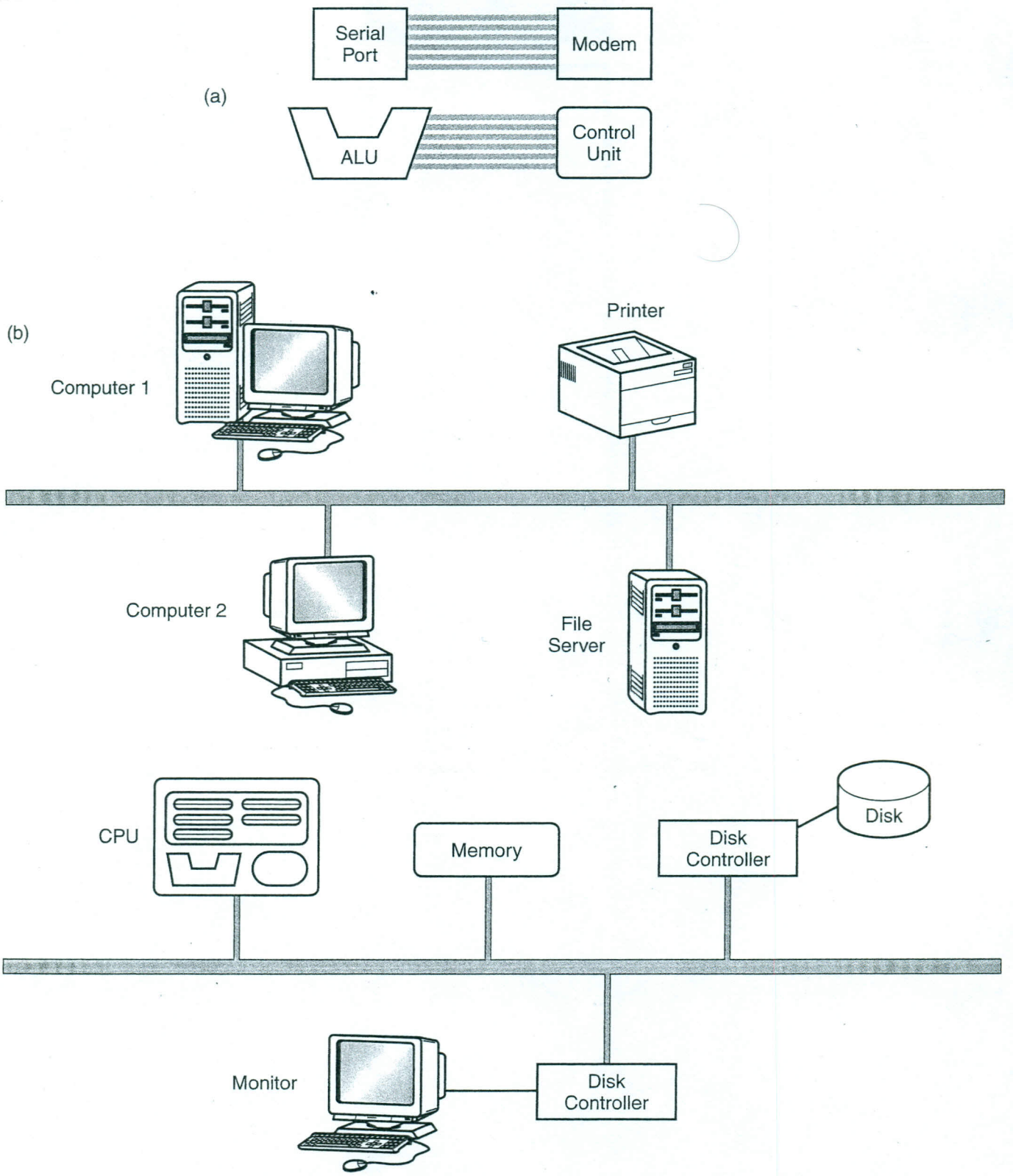

FIGURE 4.1 a) Point-to-Point Buses
b) Multipoint Buses

requiring these devices to share the bus (referred to as a **multipoint** bus and shown in Figure 4.1b).

Because of this sharing, the **bus protocol** (set of usage rules) is very important. Figure 4.2 shows a typical bus consisting of data lines, address lines, control lines, and power lines. Often the lines of a bus dedicated to moving data are called the **data bus**. These data lines contain the actual information that must be moved from one location to another. **Control lines** indicate which device has permission to use the bus and for what purpose (reading or writing from memory or from an input/output [I/O] device, for example). Control lines also transfer acknowledgments for bus requests, interrupts, and clock synchronization signals. **Address lines** indicate the location (e.g., in memory) that the data should be either read from or written to. The **power lines** provide the electrical power necessary. Typical bus transactions include sending an address (for a read or write), transferring data from memory to a register (a memory read), and transferring data to the memory from a register (a memory write). In addition, buses are used for I/O reads and writes from peripheral devices. Each type of transfer occurs within a **bus cycle**, the time between two ticks of the bus clock.

Because of the different types of information buses transport and the various devices that use the buses, buses themselves have been divided into different types. **Processor-memory buses** are short, high-speed buses that are closely matched to the memory system on the machine to maximize the bandwidth (transfer of data) and are usually design specific. **I/O buses** are typically longer than processor-memory buses and allow for many types of devices with varying bandwidths. These buses are compatible with many different architectures. A **backplane bus** (Figure 4.3) is actually built into the chassis of the machine and

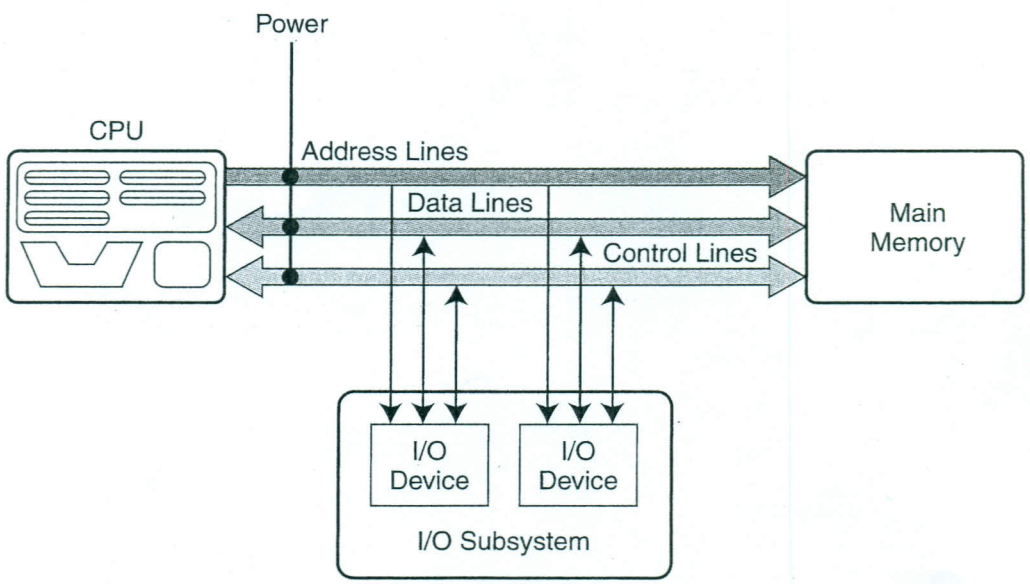

FIGURE 4.2 The Components of a Typical Bus

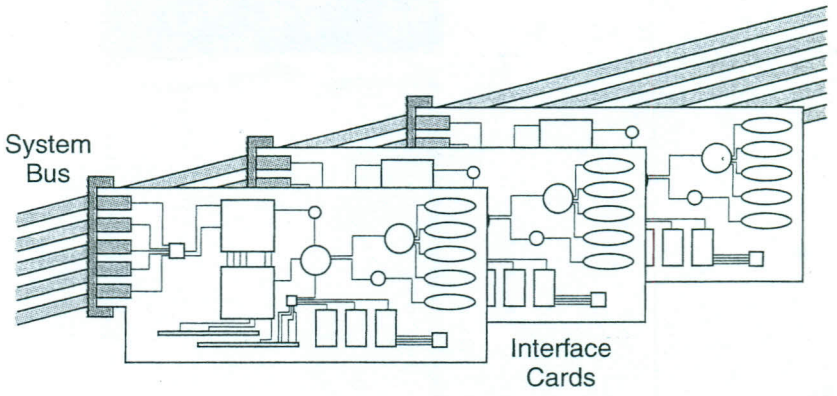

FIGURE 4.3 Backplane Bus

connects the processor, the I/O devices, and the memory (so all devices share one bus). Many computers have a hierarchy of buses, so it is not uncommon to have two buses (e.g., a processor-memory bus and an I/O bus) or more in the same system. High-performance systems often use all three types of buses.

Personal computers have their own terminology when it comes to buses. They have an internal bus (called the **system bus**) that connects the CPU, memory, and all other internal components. External buses (sometimes referred to as **expansion buses**) connect external devices, peripherals, expansion slots, and I/O ports to the rest of the computer. Most PCs also have **local buses**, data buses that connect a peripheral device directly to the CPU. These high-speed buses can be used to connect only a limited number of similar devices. Expansion buses are slower but allow for more generic connectivity. Chapter 7 deals with these topics in great detail.

Buses are physically little more than bunches of wires, but they have specific standards for connectors, timing, and signaling specifications and exact protocols for use. **Synchronous buses** are clocked, and things happen only at the clock ticks (a sequence of events is controlled by the clock). Every device is synchronized by the rate at which the clock ticks, or the **clock rate**. The bus cycle time mentioned is the reciprocal of the bus clock rate. For example, if the bus clock rate is 133 MHz, then the length of the bus cycle is 1/133,000,000 or 7.52 nanoseconds (ns). Because the clock controls the transactions, any **clock skew** (drift in the clock) has the potential to cause problems, implying that the bus must be kept as short as possible so the clock drift cannot get overly large. In addition, the bus cycle time must not be shorter than the length of time it takes information to traverse the bus. The length of the bus, therefore, imposes restrictions on both the bus clock rate and the bus cycle time.

With **asynchronous buses**, control lines coordinate the operations, and a complex **handshaking protocol** must be used to enforce timing. To read a word of data from memory, for example, the protocol would require steps similar to the following:

1. `ReqREAD`: This bus control line is activated and the data memory address is put on the appropriate bus lines at the same time.

2. `ReadyDATA`: This control line is asserted when the memory system has put the required data on the data lines for the bus.
3. `ACK`: This control line is used to indicate that the `ReqREAD` or the `ReadyDATA` has been acknowledged.

Using a protocol instead of the clock to coordinate transactions means that asynchronous buses scale better with technology and can support a wider variety of devices.

To use a bus, a device must reserve it, because only one device can use the bus at a time. As mentioned, bus masters are devices that are allowed to initiate transfer of information (control bus), and bus slaves are modules that are activated by a master and respond to requests to read and write data (so only masters can reserve the bus). Both follow a communications protocol to use the bus, working within very specific timing requirements. In a very simple system (such as the one we present in the next section), the processor is the only device allowed to become a bus master. This is good in terms of avoiding chaos, but bad because the processor now is involved in every transaction that uses the bus.

In systems with more than one master device, **bus arbitration** is required. Bus arbitration schemes must provide priority to certain master devices and, at the same time, make sure lower priority devices are not starved out. Bus arbitration schemes fall into four categories:

1. **Daisy chain arbitration:** This scheme uses a "grant bus" control line that is passed down the bus from the highest priority device to the lowest priority device. (Fairness is not ensured, and it is possible that low-priority devices are "starved out" and never allowed to use the bus.) This scheme is simple but not fair.
2. **Centralized parallel arbitration:** Each device has a request control line to the bus, and a centralized arbiter selects who gets the bus. Bottlenecks can result using this type of arbitration.
3. **Distributed arbitration using self-selection:** This scheme is similar to centralized arbitration but instead of a central authority selecting who gets the bus, the devices themselves determine who has highest priority and who should get the bus.
4. **Distributed arbitration using collision detection:** Each device is allowed to make a request for the bus. If the bus detects any collisions (multiple simultaneous requests), the device must make another request. (Ethernet uses this type of arbitration.)

Chapter 7 contains more detailed information on buses and their protocols.

4.4 CLOCKS

Every computer contains an internal clock that regulates how quickly instructions can be executed. The clock also synchronizes all of the components in the system. As the clock ticks, it sets the pace for everything that happens in the system, much like a metronome or a symphony conductor. The CPU uses this clock to