From: Digital Fundamentals by: Floyd

14

INTRODUCTION TO MICRO-PROCESSORS AND COMPUTERS

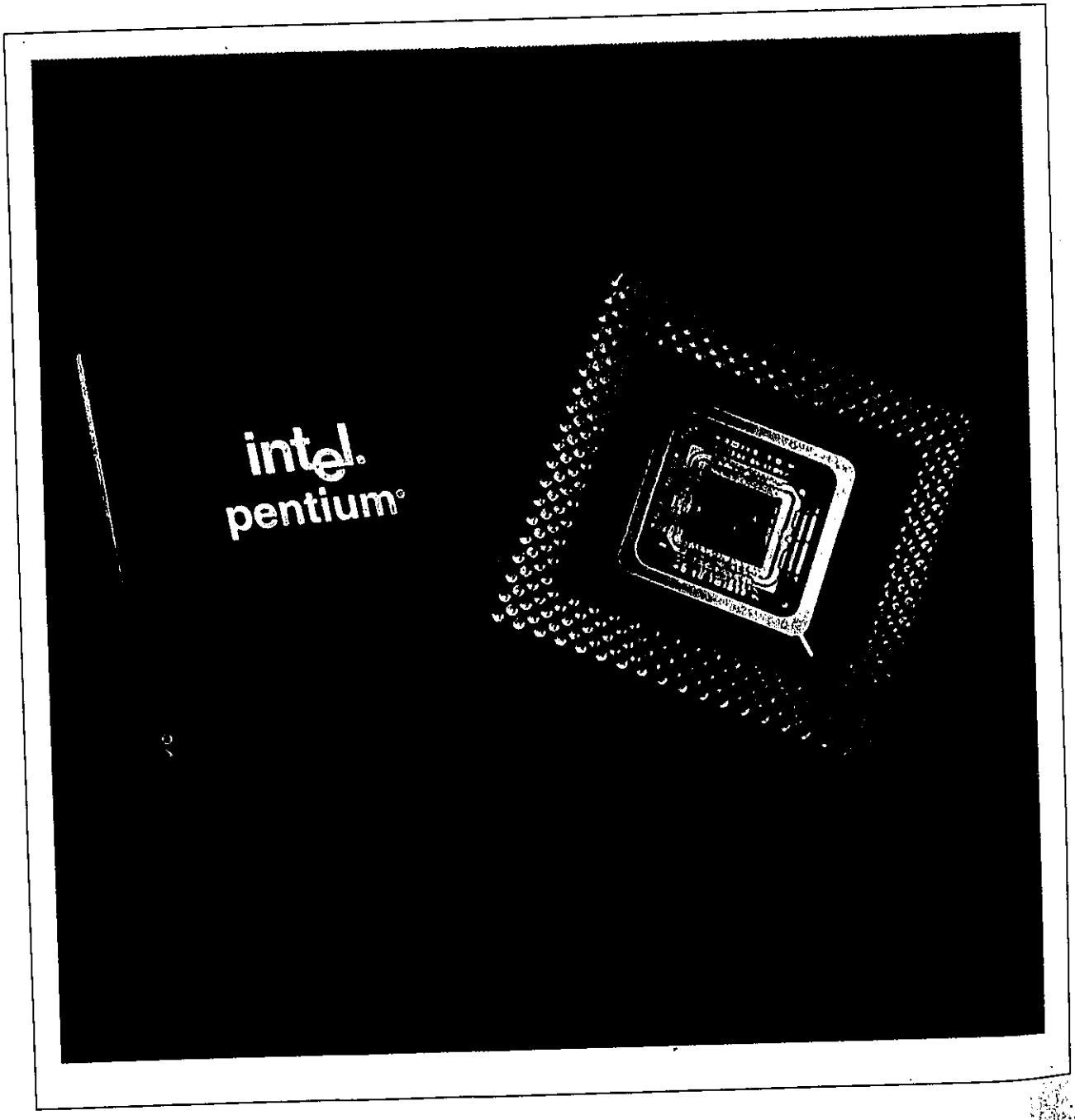
■ CHAPTER OBJECTIVES

- ❑ Name the basic elements of a microprocessor
- ❑ Name the basic units of a computer
- ❑ Discuss the characteristics and development of the Intel microprocessor family
- ❑ Discuss the characteristics and development of the Motorola microprocessor family
- ❑ Distinguish between assembly language and machine language
- ❑ Explain the basic operation of an 8086/8088 CPU
- ❑ Explain the basic architecture of the 8086/8088 microprocessors
- ❑ Explain the multiplexed bus operation of the 8086/8088 microprocessor
- ❑ Discuss the software model of the Pentium processors
- ❑ Describe a simple assembly language program
- ❑ Describe the seven instruction groups for the 80X86 and Pentium processors
- ❑ Describe the function of the bus controller in an 8086/8088 CPU
- ❑ Analyze a timing diagram for a memory-read cycle and a memory-write cycle
- ❑ Distinguish between a dedicated I/O port and a memory-mapped I/O port
- ❑ Compare polled I/O, interrupt-driven I/O, and software interrupts
- ❑ Describe the functions of PIC and PPI devices
- ❑ Define and explain the advantage of DMA

■ CHAPTER OVERVIEW

This chapter provides a brief introduction to microprocessors and computers. Naturally, a single chapter must be limited because one or more chapters could easily be devoted to each of the section topics. Keep in mind, however, that the purpose here is to give you a basic introduction.

Both the Intel and Motorola microprocessor families are briefly discussed. The Intel 8088 is used as a "model" to illustrate basic microprocessor concepts with recent enhancements described in further detail. This is a valid approach because the 8086/8088 is the first generation of the Intel 80X86 family. Although the newer generations—the 80286, 80386, 80486, and Pentium family—are more powerful and contain advanced features, they are related in architecture and basic functions such as the register structure.



14–1 ■ THE MICROPROCESSOR AND THE COMPUTER

The microprocessor is a digital integrated circuit device that can be programmed with a series of instructions to perform specified functions on data. When a microprocessor is connected to memory and provided with input and output devices, it becomes a computer.

After completing this section, you should be able to

■ Define *microprocessor* and discuss its basic elements ■ Explain the function of the ALU, the register array, and the control unit ■ Describe the address bus, the data bus, and the control bus ■ Determine the amount of memory that a microprocessor can access based on the size of its address bus ■ Define *assembly language* ■ Describe the basic elements in a computer ■ Discuss what each part of a computer does ■ Explain what a peripheral device is

The Computer Revolution

Over the last 25 years, the microprocessor has revolutionized the computer field. Prior to microprocessors, computers were universally very large, very expensive, and almost none were owned by individuals. That all changed with the introduction of the **microprocessor,** a large-scale integrated circuit that contained the entire central processing unit of a computer on a single integrated circuit. The first computers that used microprocessors were not much more than hobby computers and were called "home computers" or microcomputers. In 1981, IBM entered the small computer market with its personal computer (commonly referred to as a PC), which was based on the Intel 8088 microprocessor. Computers that are compatible with the original Intel architecture are still referred to as PCs, but they are much more than "personal" computers. Today, microprocessor-based computers have become the workhorse computers for business and industry as well as individuals.

Basic Elements of a Computer System

All computer systems consist of basic functional blocks that include a *central processing unit* (CPU), *memory,* and *input/output ports.* These functional blocks are connected together with three buses, as shown in the block diagram of Figure 14–1. The three buses are the data bus, the address bus, and the control bus. Input and output devices are connected through the input/output ports. A **port** is a physical interface on a computer through which data are passed to and from peripherals.

Instructions and data are stored in memory in specific locations determined by the **program,** a group of instructions designed to solve a specific problem. Each location has a unique **address** associated with it. Instructions are obtained by the CPU by placing an address on the address bus. Instructions are transferred via the data bus as they are requested by the CPU. The CPU executes the instructions sequentially; frequently, the instructions modify data stored in memory or obtained from an input device. Processed data may be stored back in memory or sent to an output device via the data bus. Signals on the control bus are generated by the CPU to coordinate all of these operations, which are described in more detail in this chapter.

Microprocessors

As previously defined, a microprocessor is a large-scale integrated circuit that contained the entire CPU of a computer on a single integrated circuit. As microprocessor-based computers evolved, they have become much more powerful and have widely replaced traditional mainframe computers in business and industry.

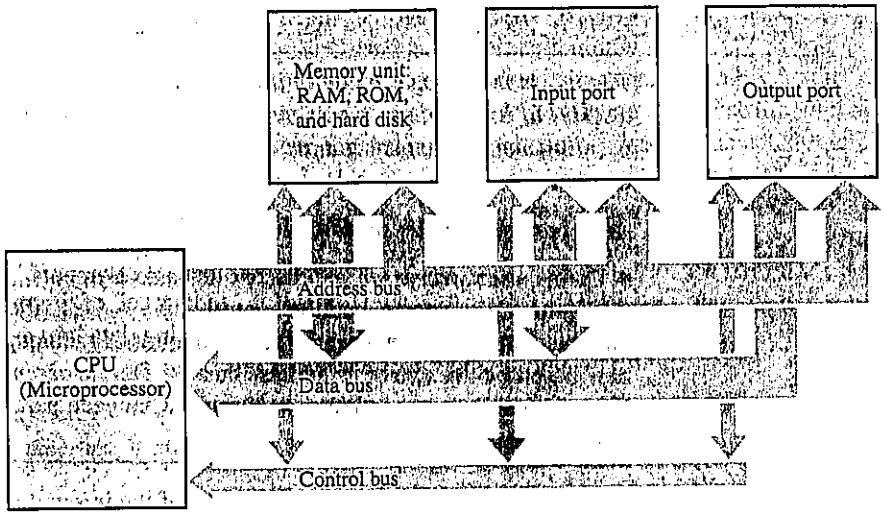
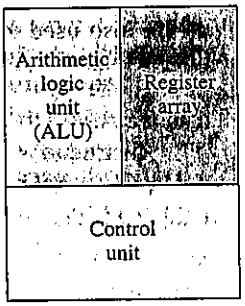


FIGURE 14–1
Basic computer block diagram.

Microprocessors are used in powerful workstations and servers as well as in peripheral devices such as printers and disk drives. In addition, microprocessors are used in many applications involving control and monitoring functions—including industrial and machine control, engine controls, and data collection to name a few. For many of these control applications, a cousin of the microprocessor known as a **microcontroller** has found widespread application.

Within a given microprocessor are several units, each designed for a specific job. The specific units, their design and organization, are called the microprocessor's **architecture.** The architecture determines the instruction set and the process for executing those instructions. Three of the most basic units, common to all microprocessors, are the *arithmetic logic unit* (ALU), the *register array,* and a *control unit,* as shown in Figure 14–2. Other units work in conjunction with these three basic units to form a specific microprocessor.

FIGURE 14–2
Three basic elements of a microprocessor.



One interesting form of microprocessor is called a **coprocessor.** A coprocessor is actually a microprocessor designed with a limited instruction set optimized to perform arithmetic operations very quickly. As microprocessors changed, one of the changes has been to incorporate the coprocessor inside what was originally just the microprocessor. Thus, the coprocessor for the Pentium is actually included inside the Pentium.

Arithmetic Logic Unit The ALU is the key processing element of the microprocessor. It is directed by the control unit to perform arithmetic operations such as addition and subtraction and logic operations such as NOT, AND, OR, and exclusive-OR. Data for the ALU are obtained from the register array.

Register Array The **register array** is the collection of registers that are contained within the microprocessor. During the execution of a program, data and addresses are temporarily stored in registers that make up this array. The ALU can access the registers very quickly, making the program run more efficiently. Some registers are classed as general-purpose registers, meaning they can be used for any purpose dictated by the program. Other registers have specific capabilities and functions and cannot be used as general-purpose registers. Still others are called *program invisible registers,* used only by the processor and not available to the programmer.

Control Unit The **control unit** is "in charge" of the processing of instructions. It provides the timing and control signals for getting data into and out of the microprocessor and for synchronizing the execution of instructions.

Microprocessor Buses

The three buses mentioned earlier are part of the internal and external connections for microprocessors to allow data, addresses, and control signals to be moved.

The Address Bus The **address bus** is a "one-way street" over which the microprocessor sends an address code to a memory or other external device. The size or width of the address bus is specified by the number of conductive paths. The first microprocessors used in computers had 16 address lines, which could address 65,536 (2^{16}) unique locations (this is called 64k). The more bits there are in the address bus, the more memory locations a given microprocessor can access. The number of address lines increased to 20, 24, and 32 bits as microprocessor technology advanced. The Pentium, with 32-bit address lines, can address over 4,295,000,000 (4G) memory locations!

The Data Bus The **data bus** is a "two-way street" on which data or instruction codes are transferred into the microprocessor or on which the result of an operation or computation is sent out from the microprocessor. The original microprocessors had data buses that were eight bits wide. Depending on the particular microprocessor, the data bus size is 8 bits, 16 bits, 32 bits, or 64 bits wide.

The Control Bus The **control bus** is used by the microprocessor to coordinate its operations and to communicate with external devices. The control bus has signals that enable either the memory or an input/output port at the proper time to read or write data. Control lines are also used to insert special wait states for slower devices and prevent bus contention, a condition that could occur if two or more devices try to communicate at the same time.

Microprocessor Programming

All microprocessors work with a basic instruction set, formulated by the designers of the processor. The Pentium, for example, has hundreds of variations of its instructions divided into seven basic groups:

- Data transfer
- Arithmetic
- Bit manipulation
- Loops and jumps
- Strings
- Subroutines and interrupts
- Processor control

Each instruction consists of a binary string (1s and 0s) that is decoded by the microprocessor before being executed. These instruction groups are described in Section 14–4. The binary code instructions are called *machine language* and are all that the microprocessor

recognizes. The first computers were programmed by actually writing the instructions in binary code, which was a tedious job and prone to error.

To simplify the task of writing instructions for a computer, assembly language was devised. **Assembly language** is classified as a low-level language because the English-like instructions contained in a given assembly language can be directly translated into binary instructions or data patterns. An **assembler,** which is a program, converts the English-like instructions, called **mnemonics,** in the assembly language into the machine language (binary patterns) used by the microprocessor, as illustrated in Figure 14–3. Other instructions used in assembly language describe data structures and other information needed by the assembler. These instructions are called **pseudo-operations** because they are not translated into actual instructions for the microprocessor.

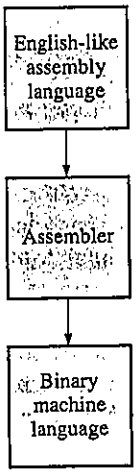


FIGURE 14–3
Block diagram of microprocessor programming.

Assembly language and the corresponding machine language is specific to the type of microprocessor or microprocessor family. For example, assembly instructions written for the Motorola 600 series processors will not work on the Intel family of processors. The advantage of assembly language is that the programmer has direct control of the process, and usually codes written in it execute faster. In addition, the programmer can manage machine processes, such as interrupts or I/O devices, better with assembly language than most other computer languages.

On the other hand, high-level programming languages such as BASIC, Pascal, C, or FORTRAN are independent of the type of microprocessor in a computer system. A program called a **compiler** or interpreter translates the high-level program statements into machine language. The advantage is that a general-purpose language can be designed for specific types of problems, such as business applications. For general problem solving, high-level languages are superior to assembly language. Programming is discussed further in Section 14–4.

A Computer System

For a computer to accomplish a given task, it must communicate with the "outside world" by interfacing with people, sensing devices, or devices to be controlled. A typical computer system is shown in Figure 14–4. Usually, there is a keyboard for data entry, a video monitor, and a printer. A mouse is also used on most computers. Other types of peripherals (such as an external disk drive, modem, scanner, graphics tablet, or voice input) are often part of a system.

Central Processing Unit The part of the computer in control of running a program is the CPU which, for microprocessor-based computers, is the microprocessor itself. Basically, the CPU addresses a memory location, obtains (fetches) a program instruction that is stored

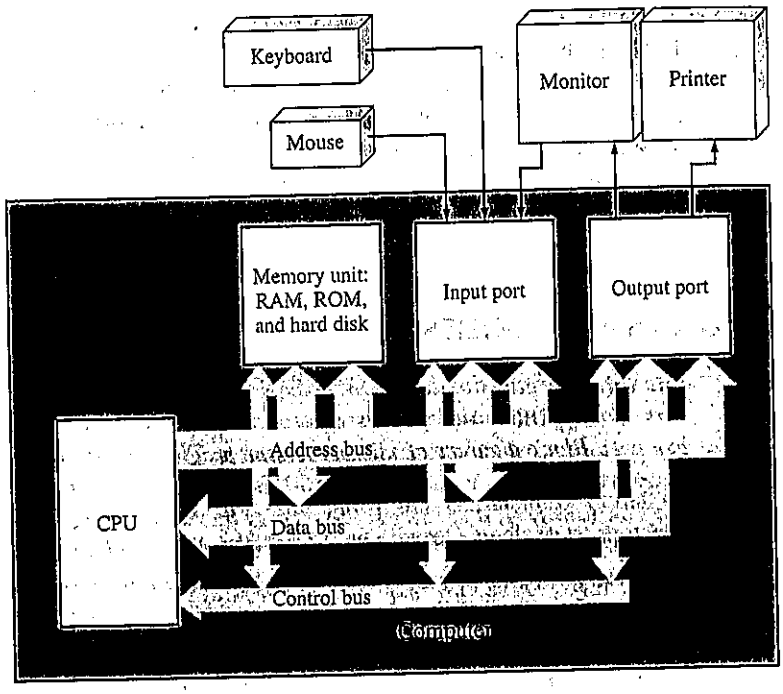


FIGURE 14–4
A typical computer system.

there, and carries out (executes) the instruction. After completing one instruction, the CPU moves on to the next one (although newer microprocessors can operate on more than one instruction simultaneously). This fetch and execute process is repeated until all of the instructions in a specific program have been executed. A simple example of an application program is a set of instructions stored in the memory as binary codes that directs the CPU to fetch a series of numbers also stored in the memory, add them, and store the sum back in the memory.

Memory Unit The memory unit typically consists of RAM, ROM, and hard disk memories. The RAM stores data and programs temporarily during processing. Data are numbers or other information in binary form, and programs are lists of instructions. Because the RAM is generally a volatile memory, everything must be backed up in nonvolatile disk storage.

The ROM stores system programs such as the **BIOS** (basic input/output system). The **BIOS** contains certain instructions telling the CPU what to do when power is first turned on and directs the power on self-test or POST and reports any errors. It also contains routines for handling video display graphics, the keyboard, asynchronous communications, as well as other peripherals. These programs can be accessed by other programs though a series of software interrupts.

Input Ports Generally, the computer receives information from an input device via an input port. (Some microprocessors do not use ports; they communicate with I/O devices through locations in memory specifically reserved for the I/O device as will be discussed in Section 14–7). The keyboard, mouse, and hard disk are examples of input devices that are accessed by the CPU typically via individual input ports.

Output Ports The computer sends information to output devices via an output port (or through reserved memory locations). An example of a peripheral with which a computer communicates through an output port is the video monitor. Some peripherals function as both input and output devices; examples are external disk drives, and modems. For devices that require two-way communication, a separate input and output port is set up for the device.

SECTION 14–1 REVIEW

1. Name the basic elements of a computer.
2. Describe each of the three buses used by a microprocessor.
3. What are two types of information stored in memory?
4. What is assembly language? What two types of operations are in assembly language?

14–2 ■ MICROPROCESSOR FAMILIES

The Intel Corporation is a pioneer in the development and marketing of microprocessors. Today, Intel microprocessor products have a large share of the world market and are used in most PC-compatible computers. This section contains a brief history and description of the evolution of the 80X86 microprocessor family from the 8086/8088 to the Pentium I, II, and III. Motorola also has enjoyed significant success with its line of microprocessors. The first Apple computer was based on the Motorola 6800, and all Apple Macintosh computers today use Motorola microprocessors. As in the 80X86 Intel family, each Motorola microprocessor builds on the basic design established by the first device with added features and improved performance. Our emphasis is on main features and improvement from one generation of devices to the next. Keep in mind that microprocessor technology advances at a fast pace, and performance parameters change as current devices are improved or replaced with new versions.

After completing this section, you should be able to

■ Discuss the Intel 80X86 family of microprocessors ■ Discuss the Motorola 680X0 family of microprocessors ■ Describe a cache memory and state its purpose ■ Describe pipelining and state its purpose

The Early Intel Microprocessors

Intel's first microprocessor was the 4004, introduced in 1971. It was designed for the Busicom calculator and had only a 4-bit data bus. This device was followed by the 8008, which had an 8-bit data bus and was used by Don Lancaster, a computer hobbyist, to create an early computer. Two more 8-bit microprocessors, the 8080 and the 8085 were introduced in the mid-1970s. These two devices could address only $2^{16} = 65{,}536$ memory locations (64 kbytes of memory) with their 16-bit address buses. The 8080 microprocessor was used in the first computer available to the public, the Altair.

The 80X86 Family of Microprocessors

Since its introduction in 1978, the so-called 80X86 architecture has undergone a number of evolutionary stages. The first generation of the 80X86 family includes the 8086, the 8088, and the 80186. Next came the 80286, followed by the 80386, and then the 80486. The Pentium is the fifth Intel microprocessor; and now it is followed by the Pentium Pro, Pentium II, and Pentium III microprocessors, frequently referred to as simply processors. Another version of the Pentium microprocessor, dubbed the Celeron, was introduced in 1998. Each new microprocessor built upon the basic concepts of the 8086/8088 with additional features and improved performance.

There are three basic methods for improving the ability of a microprocessor to process data. Chip designers are always trying to increase the clock speed, but a major problem with just increasing the speed is that the circuit dissipates more power (increased heat) as the clock speed is increased. A second method is to increase the size of the

data bus, to bring more information to the processor. Finally, the internal architecture can be changed to execute instructions more efficiently. An internal architecture change of particular importance was the introduction of pipelining and duplicate execution units for parallel processing. As you review the history of microprocessors, you will see that each of the three methods mentioned has been instrumental in improving the performance as new microprocessors were introduced and are key parts of microprocessor specifications.

The 8086/8088 and 80186 Developed in 1978, the 8086 was the first of the 80X86 family and is the basis for all Intel microprocessors that followed. The 8086 was a 16-bit microprocessor (16-bit data bus) and represented a significant departure from the earlier 8-bit devices. The number of address lines in the 8086 increased to twenty, making it capable of addressing 2^{20} = 1,048,576 memory locations (1 Mbyte). The 8086 was a factor of ten improvement over the earlier 8080 processor. The various versions of the 8086 operated at clock frequencies of 5 MHz, 8 MHz, or 10 MHz.

The 8088 is essentially an 8086 with the 16-bit internal data bus multiplexed down to an 8-bit external data bus. It was intended to meet the demand for applications in simpler 8-bit systems and was used in the original IBM computer (PC). The success of the IBM computer led Fortune magazine to name the company "one of the business triumphs of the seventies."

The 80186 is an 8086 with several support functions such as clock generator, system controller, interrupt controller, and direct memory access (DMA) controller integrated on the chip. An increased clock frequency of 12.5 MHz was added, and the 5 MHz rate available in the 8086/8088 was dropped, resulting in a selection of 8 MHz, 10 MHz, or 12.5 MHz. The 80186 represented a relatively small increase in performance over the 8086; it was never used as the CPU of a commercial computer.

The 80286 (286) The 80286 was introduced in 1982, and the memory addressing capability was increased to 24 address lines. This provides for addressing 2^{24} = 16,777,216 memory locations (16 Mbytes). Even more importantly, an advanced mode of operation called *protected mode,* was first incorporated in the 80286 and used in all of the following Intel processors. This mode allows access to additional memory locations and advanced programming features such as **multitasking,** an operating system environment in which the computer seems to run multiple programs or tasks simultaneously. In addition, the 286 added a new functional addressing unit devoted to virtual addressing, a concept that allows a large program to execute in a smaller physical memory. Although the 80286 operates at the same clock frequencies as the 80186, it represented a major change in architecture and added a number of new instructions devoted to multitasking. Even with all of the improvements, the 286 could execute all of the code written for the earlier 8086.

The 80386 (386) In 1985, a major step was taken when the first Intel 32-bit microprocessor was introduced. The 80386 has both a 32-bit data bus and a 32-bit address bus. The DX version has a 32-bit external data bus while the SX version has a 16-bit external data bus. With 32 address bits, 2^{32} = 4,294,967,296 memory locations (4 Gbytes) can be accessed.

A new paging mechanism was added to the 80386 that enabled more efficient use of memory yet retained the upward compatibility for earlier processors that was necessary. The process of forming the physical address was broken into two layers of lookup tables where all of the information about the address was stored. This process enabled any program written for the 8086 to have its own 1 Mbyte memory area, as if it were operating alone in the original 8086 memory. Operating systems (like OS/2) could run several tasks on a rotating basis so that several users had concurrent access to programs.

The 386 is compatible with all earlier models, including the register array. Although the general-purpose registers are 16 bits wide on all earlier machines, they were extended to 32 bits on the 386 and given important new capabilities. In addition to some other features,

the 80386 was the first Intel microprocessor to enhance processing speed with the use of instruction pipelining. This technique allows the microprocessor to start working on a new instruction before it has completed the current one. Versions that operate at clock frequencies of 16 MHz, 20 MHz, 25 MHz, or 33 MHz are available in the 80386.

Other more economical versions of the 80386 include the 80386SX and the 80386SL in which the data bus is multiplexed down to 16 bits and the address bus down to 24 bits. All of the 80386 microprocessors can also operate with a math (numeric) coprocessor that can be used to enhance or support the functions of the microprocessor by performing floating-point operations.

The 80486 (486) The next step in the evolutionary process was the introduction of the 80486 with an internal 8-kbyte cache memory in 1989. As you learned in Chapter 12, cache memory reduces memory access time by storing recently used instructions or data in a very fast SRAM rather than having the information in the relatively slow main memory (DRAM). Also, an internal math coprocessor was incorporated, and the internal architecture was redesigned to speed certain instructions. Although there were relatively few changes to the programming model between the 386 and the 486, the introduction of cache memory and the other improvements resulted in a significantly better processor. The 80486 was Intel's first processor that had over 1 million transistors. Certain versions of the 80486 operate at clock frequencies up to 66 MHz.

The Pentium The Pentium, introduced in 1993, retains the 32-bit address bus of the 80486 but doubles the data bus to 64 bits. The Pentium retained the 32-bit registers of the 486 but made it possible to deal with the 64-bit data bus by adding a second execution unit. The Pentium also has two 8-kbyte cache memories, one for instructions and one for data. A dual pipeline method, known as *superscalar architecture,* increases the speed at which instructions can be processed above that of the single pipelining method introduced in the 80386. The Pentium has internal circuitry to recognize when both pipelines may be used; in those instances, the Pentium can execute two instructions at once. Another improvement was a much improved math coprocessor. The original Pentium operated at a clock frequency of 60 MHz or 66 MHz.

Pentium Pro The Pentium Pro, code-named the P6, was introduced in 1995 and followed the basic architecture of the original Pentium. Designed for high-end computers, work stations, and servers, the Pentium Pro contained three integer units, a floating-point unit, and both level one and level two cache memory. The Pentium Pro is packaged with a second, speed-enhancing cache memory chip. It operates at clock speeds up to 200 MHz.

Pentium II The Pentium II was introduced in 1997 and incorporates Intel MMX technology designed specifically to process video, audio, and graphics data efficiently. The MMX technology improves video compression/decompression, image manipulation, encryption, and I/O processing. The Pentium II has 57 new instructions designed to speed the repetitive sequences often found in multimedia operations. The Pentium II is compatible with all of the earlier processors from the 8086 on. Additional members of the Pentium II family were introduced in 1998, including the Celeron processor, designed for basic PC market. By 1999, the clock speed of the Pentium II was up to 450 MHz, and the clock speed of the Celeron processor was up to 400 MHz. Further speed enhancements are expected in the future.

Pentium III The Pentium III was introduced in 1999. It was designed for the consumer and business market desktop computers. The Pentium III included 70 new instructions designed to enhance 3D, imaging, and video applications. Initial computers with the Pentium III operated at 500 MHz, but the Pentium III has been tested (by Intel) at speeds of over 1 GHz using special cooling techniques.

The Motorola 6800 and Its Variations

The first microprocessors from Motorola were all 8-bit devices (8-bit data bus). The 6800 appeared in 1975 with a clock frequency of 2 MHz and was capable of addressing 64 kbytes of memory with a 16-bit address bus. The internal architecture included two accumulators, a program counter, index register, and a stack pointer but no general-purpose registers. (An **accumulator** is a register with special capability for arithmetic and logic operations.) For use as a "scratch-pad," the 6800 used RAM memory instead of general-purpose registers.

In the 6802, a 128-byte RAM was added for use in the place of some registers. The clock frequency was increased in the 6803 to 3.58 MHz and a UART (universal asynchronous receiver/transmitter) was added for serial communications. The last of the 8-bit microprocessors was the 6809, which offered an enhanced instruction set that included a multiply instruction. All of the 8-bit devices retained the 16-bit address bus.

The 680X0 Family of Microprocessors

The 68000 The 68000 was the first of Motorola's 16-bit microprocessors and was introduced in 1979. Although it is classified as a 16-bit processor, internally it had 32-bit registers. It also had 24 address lines that could access 16 Mbytes of memory, compared to the 1 Mbyte capability of the 8086/8088. Unlike the 8086, it also had independent address and data lines. With a clock frequency of 16 MHz, the 68000 could run faster than any of the Intel devices at that time. Also, the RAM-based register concept used in the 6800 was abandoned in favor of an internal register array that included eight 32-bit general-purpose registers, seven 32-bit address registers, two 32-bit stack pointers, as well as a 32-bit program counter and a 16-bit status register. The processors that followed were all upward compatible; that is, programs written for the 68000 can run on all of the 680X0 family.

The 68020 Motorola officially entered the world of 32-bit microprocessors in 1985 with the 68020, which had an increase in performance over 16-bit processors. It could address 4 Gbytes of memory (using a 32-bit address bus) and featured a 256-byte cache memory for instructions, an innovative feature at that time. The clock frequency was increased to 33 MHz, and a math coprocessor (the 68882) and other support chips were available.

The 68030 Introduced in 1991, the 68030 added another 256-byte cache for data, and the clock speed was increased to up to 50 MHz. The address bus was increased to 32 bits. With these and other improvements that produced increased efficiency, the 68030 could execute instructions at more than double the speed of its immediate predecessor.

The 68040 Introduced in 1995, the 68040 increased the data and instruction caches to 4 kbytes each, and an on-chip math coprocessor featuring floating-point support was added. The processor also had multiple independent pipelines to enable concurrent execution of instructions and an internal floating-point unit and on-chip memory management.

The 68060 The 68060 was also introduced in 1995 and featured a superscalar architecture characterized by multiple pipelining of instructions and two 8-kbyte caches (for data and for instructions). The power consumption was reduced by lowering the operating voltage to 3.3 V. The top clock frequency is 75 MHz.

The Power PC The MPC601 was introduced in 1992 as the first of a series 64-bit microprocessors with superscalar architecture that can effectively execute up to three instructions per clock cycle. It has 32 kbytes of cache memory and an internal math coprocessor. The Power PC is a RISC type of microprocessor. RISC means reduced instruction set computer, an approach that reduces the number of instructions available but optimizes the reduced instruction set and results in overall improved performance. The latest addition to the Power PC family is the 750, announced in 1999. It features a 32-kbyte instruction cache and a 32-kbyte data cache. The speed of the 750 is up to 400 MHz.

SECTION 14–2 REVIEW

1. What is meant by the term *32-bit microprocessor?*
2. What was the first 16-bit Intel microprocessor?
3. What is meant by RISC?
4. What is the purpose of a cache memory?
5. What is superscalar architecture?

14–3 ■ THE 8086/8088 MICROPROCESSOR AND SOFTWARE MODEL FOR THE PENTIUM PROCESSOR

As you saw in the last section, the 80X86 microprocessor family has undergone a tremendous change from the 8086/8088 to the Pentium family, both in speed and in complexity. However, the basic register set and other features of the 8086/8088 have been retained (and expanded) throughout the evolutionary process so that all of the newer Intel processors respond to the same instructions (as well as a number of new instructions) as the original 8086/8088. This section starts with a limited introduction of the 8086/8088 to introduce basic concepts of microprocessor architecture, operation, and programming. The section ends with a brief overview of the principal changes to the register structure that forms the software model of the newer processors. The approach is to show a basic processor (the 8086/8088) and discuss the enhancements to the Intel line as it has evolved.

After completing this section, you should be able to

- Discuss the basic microprocessor operation
- Describe the bus interface unit
- State the purpose of the segment registers
- State the purpose of the instruction pointer
- Describe the execution unit
- Describe the general set of registers
- State the purpose of the flag register

Basic Operation

A microprocessor executes a program (list of instructions) by repeatedly cycling through the following three steps:

1. Fetch an instruction from memory and place it in the CPU.
2. Decode the instruction; if other information is required by the instruction, fetch the other information. In the decode step, the program counter is updated to point to the next instruction.
3. Execute the instruction (do what the instruction says). Results are returned to registers and memory during this step.

Prior to the 8086/8088 microprocessors, these steps were performed sequentially; the 8085 processor could not fetch an instruction at the same time it was executing it. The architecture of the 8086/8088 was fundamentally different. It provided for two separate internal units: the execution unit (**EU**), which executes instructions, and the bus interface unit (**BIU**), which interfaces with the system buses and fetches instructions, reads operands, and writes results. These units are shown in Figure 14–5.

The BIU performs all the bus operations for the EU, such as data transfers from memory or I/O. While the EU is executing instructions, the BIU "looks ahead" and fetches more instructions from memory. This action is called **prefetching** or *pipelining*. The concept of prefetching is to allow the processor to execute instructions at the same time as the next instruction was being fetched, eliminating idle time. The prefetched instructions are stored in an internal high-speed memory called the instruction **queue** (pronounced "Q"). The queue allows the BIU to keep the EU supplied with instructions. The EU does not

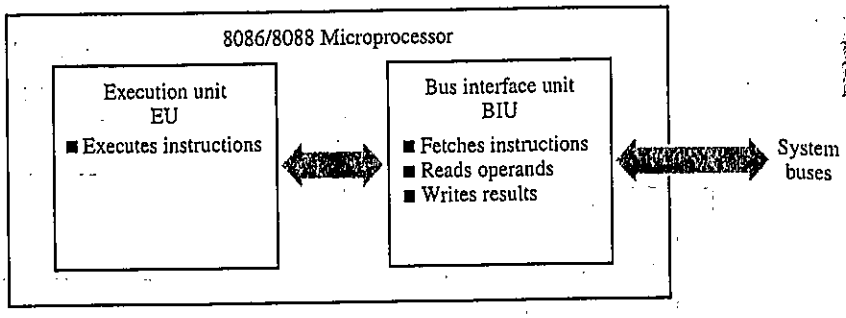


FIGURE 14–5
The 8086/8088 has two separate internal units, the EU and the BIU.

have to wait for the next instruction to be fetched from memory; but instead it retrieves the next instruction directly from the queue in much less time. In the Pentium, this process is taken a step further. Two complete execution units enable two instructions to execute at the same time provided they are independent. Certain compilers are designed to take advantage of the two execution units by a process known as **instruction pairing** to remove dependencies.

Basic 8086/8088 Architecture

Figure 14–6 is a block diagram of the architecture (internal organization) of an 8088 microprocessor. Externally, the 8088 has 20 address bits that can address 1 Mbyte (1,048,576 bytes) of memory and uses an 8-bit data bus. Internally, the 8088 has a 16-bit data bus and a 4-byte queue. The 8086 is identical except that it has an external 16-bit data bus and a 6-byte instruction queue.

The Bus Interface Unit (BIU)

The major parts of the BIU are the 4-byte instruction queue, the segment registers (CS, DS, SS, and ES), the instruction pointer (IP), and the address summing block (Σ). The 16-bit internal data buses and the Q bus interconnect the BIU and the EU.

Instruction Queue The instruction queue increases the average speed with which a program is executed (called the **throughput**) by storing up to four bytes (six in the 8086). As described earlier, this technique allows the 8088 essentially to do two things, fetch and execute, at one time. This feature has been expanded in subsequent processors to include much larger and faster queues.

Segment Registers The 8086/8088 processors have four segment registers (CS, DS, SS, and ES) that are all 16-bit registers used in the process of forming a 20-bit address. A **segment** is a 64-kbyte block of memory and can begin at any point in the 1 Mbyte (1,048,576 bytes) of memory space, provided it begins on a 16-byte boundary (evenly divisible by 16).

In designing the 8086/8088 and subsequent processors, Intel chose a unique method of generating the required 20-bit physical address using two 16-bit registers. One of the registers that forms the physical address (or actual) is always a segment register; the other register is a 16-bit general register containing address information. The principal advantage of the method selected was to allow codes to be easily relocatable. A **relocatable code** can be moved anywhere within the memory space without changing the basic code.

Each of the four segments identify the starting address of a 64-kbyte (65,536-byte) block representing a "window" in the entire 1-Mbyte (20-bit) memory space. The starting

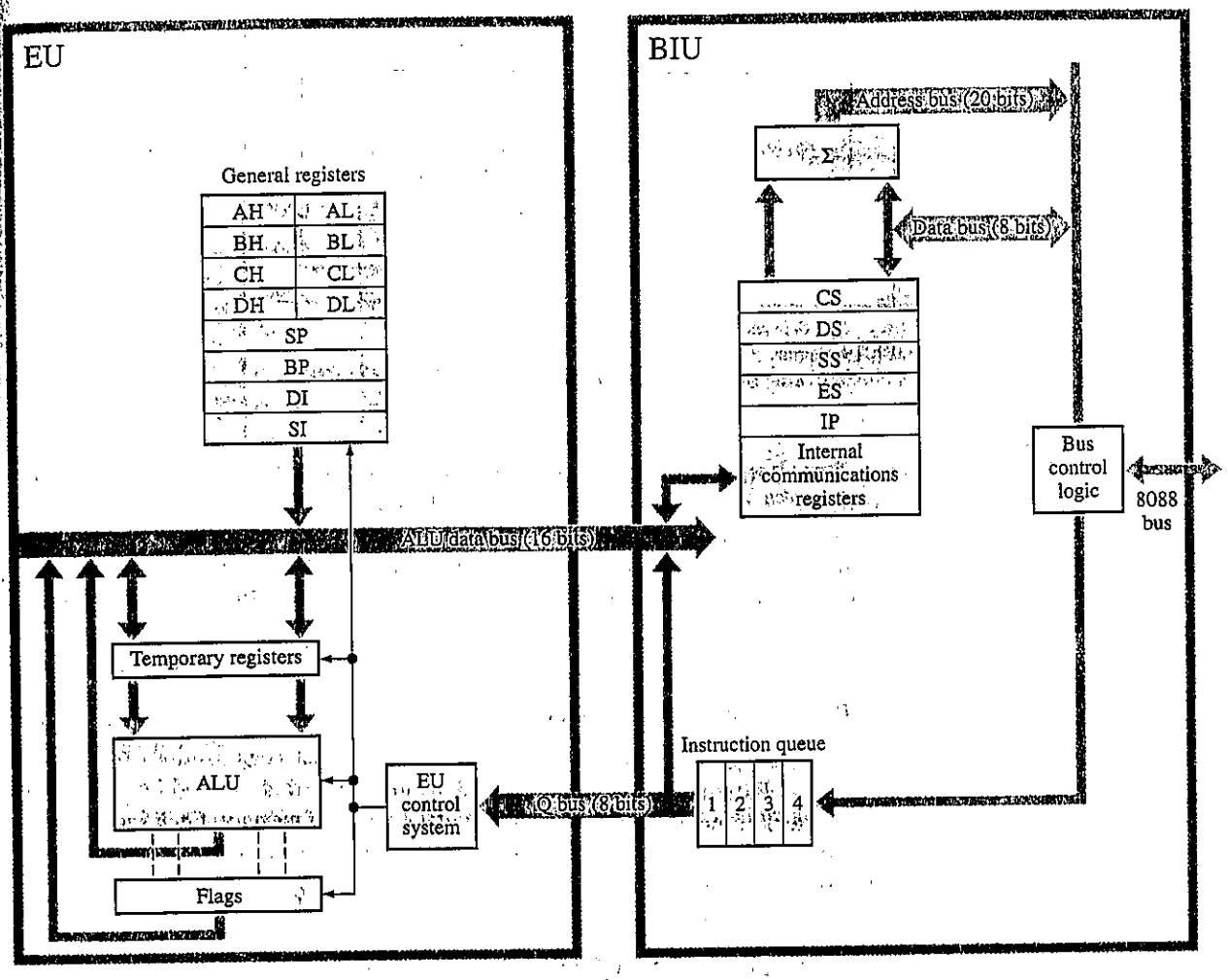


FIGURE 14–6
The internal organization of the 8088 microprocessor.

address of a segment is represented by the 16-bit number in the segment register plus an implied 4 bits appended to the right that are always assumed to be zero. In other words, the segment registers contain the most significant 16 bits that represent the physical starting address of the segment.

The four segment registers (CS, DS, SS, and ES) can be changed by the program to point to other 64-kbyte blocks if necessary. (For small codes, it is normally not necessary to change the segments.) The four segments can be separate locations within the memory space or can overlap, depending on the size and requirements of the particular code. They can even be defined as the same 64-kbyte block. In the 8086/8088, currently addressable memory segments are those defined by the segment address contained in the CS (code segment) register, the DS (data segment) register, the SS (stack segment) register, and the ES (extra segment) register. In later processors, other segment registers were added.

As mentioned, within each segment are 64 kbytes of memory. To find a given memory location, a segment address is combined with an offset address. The segment address represents the most significant sixteen bits (four hex digits) of the physical address which represent the beginning address of a segment. The offset address is sixteen additional bits that represent the distance from the start of the segment to the physical address within the segment. Figure 14–7 illustrates how the memory is be divided into segments and offsets and shows examples of nonoverlapping and overlapping segments.

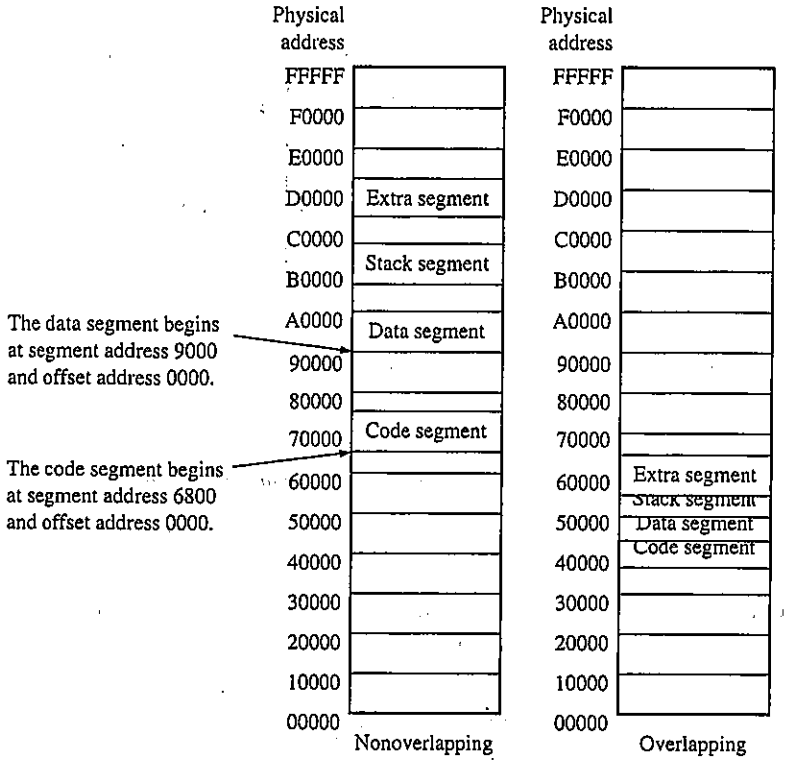


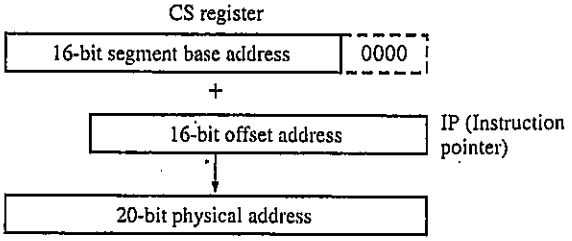
FIGURE 14–7
Nonoverlapping and overlapping segments in the first 1 Mbyte of memory. Each segment represents 64 kbytes.

Instruction Pointer (IP) and Address Summing Block The 16-bit IP (instruction pointer) points to the offset of the next instruction to be executed in memory. The IP always references the CS (code segment) register; thus, the physical address of the next instruction is formed by combining the code segment and the instruction pointer. The IP always contains the offset address of the next instruction, and the CS register always contains the segment address. This address is shown in assembly language as CS:IP.

To form the 20-bit physical address of the next instruction, the 16-bit offset address in the IP is added to the segment address contained in the CS register, which has been shifted four bits to the left, as indicated in Figure 14–8. As mentioned earlier, an assumed binary 0000 is the least significant position. The addition is then done by the address summing block.

Figure 14–9 illustrates the addressing of a location in memory by the segment: offset method. In this figure, $A000_{16}$ is in the segment register and $A0B0_{16}$ is in the IP. When the CS register is shifted and added to the IP, we get $A0000_{16} + A0B0_{16} = AA0B0_{16}$ for the physical address.

FIGURE 14–8
Formation of the 20-bit physical address from the segment base address and the offset address.



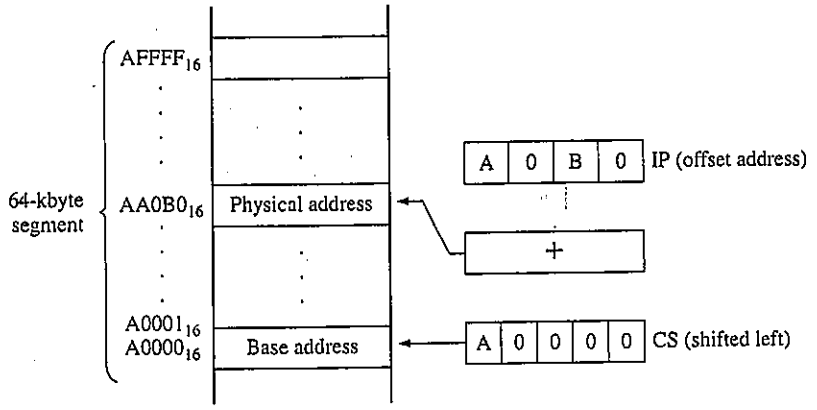


FIGURE 14–9
Illustration of the segmented addressing method.

EXAMPLE 14–1 The hexadecimal contents of the CS register and the IP are shown in Figure 14–10. Determine the physical address in memory of the next instruction.

FIGURE 14–10

$A034_{16}$ CS

$0FF2_{16}$ IP

Solution Shifting the CS base address left four bits (one hex digit) effectively places a 0_{16} in the LSD position, as shown in Figure 14–11. The shifted base address and the offset address are added to produce the 20-bit physical address.

FIGURE 14–11

A0340	+	0FF2	=	A1332
Base address (shifted left 4 bits)		Offset address		Physical address of next instruction

Related Problem Determine the physical address if the CS register contains $6B4D_{16}$.

It is important to understand how the segment:offset method is used to form the physical address; however, in programming work, it isn't usually necessary for the programmer to specify actual physical addresses. This job is done by the assembler program using labels supplied by the programmer. When a physical address is required, the programmer generally specifies it with the segment:offset method. Thus, the address for Example 14–1 would be given as simply A034:0FF2.

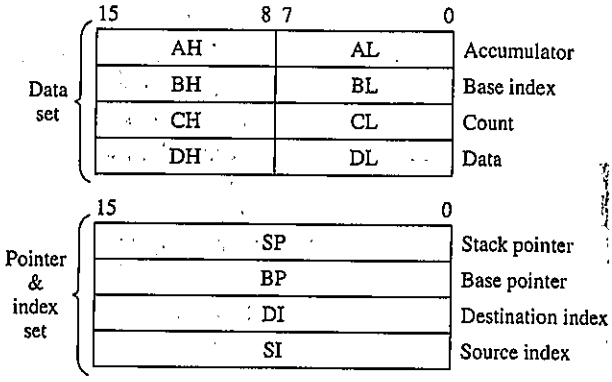
The Execution Unit (EU)

The EU decodes instructions fetched by the BIU, generates appropriate control signals, and executes the instructions. The main parts of the EU are the arithmetic logic unit (ALU), the general registers, and the flags.

The ALU This unit does all the arithmetic and logic operations, working with either 8-bit or 16-bit operands.

The General Registers This set of 16-bit registers is divided into two sets of four registers each, as shown in Figure 14–12. One set consists of the data registers, and the other set consists of the pointer and index registers. The **pointer** and index registers are generally used to keep offset addresses (as used here, a pointer refers to a specific memory location). In the case of the stack pointer (SP) and the base pointer (BP), the default reference to form a physical address is the stack segment (SS). The index pointers (SI and DI) and the base register (BX) generally default to the data segment (DS) register (an exception is made for certain instructions to this general rule).

FIGURE 14–12
The 8086/8088 general register set.

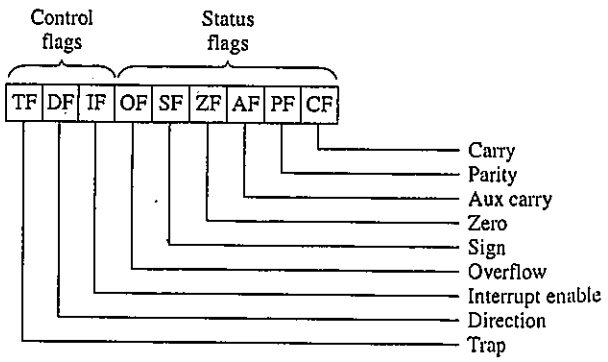


Each of the 16-bit data registers (AX, BX, CX, DX) has two separately accessible 8-bit sections. Depending on the program, they can be used either as a 16-bit register or as two 8-bit registers. The low-order bytes of the data registers are designated as *AL*, *BL*, *CL*, and *DL*. The high-order bytes are designated as *AH*, *BH*, *CH*, and *DH*. These registers can be used in most arithmetic and logic operations in any manner specified by the programmer for storing data prior to and after processing. Also, some of these registers are used specifically by certain program instructions.

The pointer and index registers are the stack pointer (SP), the base pointer (BP), the source index (SI), and the destination index (DI). These registers are used in various forms of memory addressing under control of the EU.

The Flags The flag register contains nine independent status and control bits (**flags**), as shown in Figure 14–13. A status flag is a one-bit indicator used to reflect a certain condition after an arithmetic or logic operation by the ALU, such as a carry (CF), a zero result (ZF), or the sign of a result (SF), among others. The control flags are used to alter processor operations in certain situations.

FIGURE 14–13
The 8088 status and control flags.



Software Model of the Pentium Family of Processors

As Intel introduced newer microprocessors, capabilities and speed increased dramatically. With the Pentium processor, the earlier pipeline concept introduced in the 8086/8088 was increased to two integer pipelines. The external coprocessor was incorporated within the microprocessor, and address and data buses were greatly expanded. Other improvements (such as clock speed, reduced instruction clock cycles, branch prediction capability, and an integral floating-point unit) made the Pentium a significantly better processor than its predecessors. In addition to processor improvements, many improvements to other parts of computers occurred (such as bus protocols and size, speed, memory size, and cost). Despite all of these changes, the designers of the newer processors maintained upward compatibility; that is, the newest Pentium could still run the software for any of the processors that preceded it. This was done by maintaining the basic software model (register structure) of the original 8086/8088.

The registers described previously for the 8086/8088 are a subset of the registers in the Pentium family of processors. Beginning with the 80386 processor, the register set was expanded to include 32-bit registers. The 32-bit registers kept the original names but an E (for Extended) was added as a prefix to the register names; thus, the 32-bit designation for the AX register is the EAX. In addition, two new segment registers were added. The extended registers are shown in Figure 14–14. The shaded areas represent the registers only available on the 386 and above.

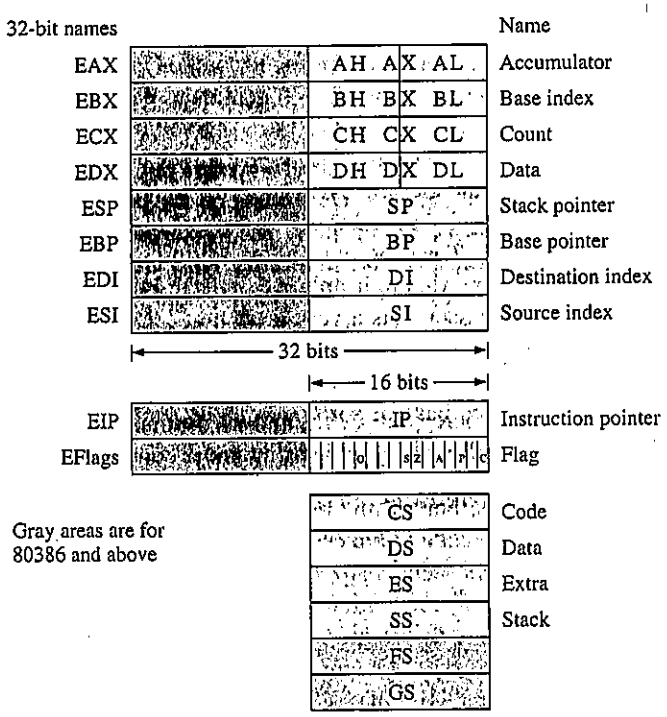


FIGURE 14–14
Registers for the Intel processors from 8086/8088 through Pentium.

In addition to the extended registers, the addressable space in memory was increased dramatically with the introduction of newer processors. To keep the upward compatibility, Intel reserved the first 1 Mbyte of memory for codes running in real mode. **Real mode** is any operation that allows the processor to only access the first 1 Mbyte of memory to simulate the 8086/8088. Code written for an earlier processor can run in real mode on a newer processor (although the reverse is not strictly true). Code written in real mode is generally compatible (with some exceptions) with all of the Intel processors from the 8086/8088 upward.

SECTION 14–3 REVIEW

1. Name the general-purpose registers in the 8086/8088 microprocessor.
2. What is the purpose of the BIU?
3. Does the EU interface with the system buses?
4. What is the function of the instruction queue?
5. What is the advantage of the segment:offset method of forming addresses?
6. What is instruction pairing?

14–4 ■ MICROPROCESSOR PROGRAMMING

All computers must be programmed to perform even the most elementary tasks. This section focuses on the basic concept of programming in assembly language, a low-level language that is a symbolic representation of the basic machine language used by microprocessors.

After completing this section, you should be able to

■ Explain the general purpose of a computer program ■ Describe a simple assembly language program ■ Discuss how a microprocessor executes a program ■ Describe the seven instruction groups for the 80X86 and Pentium processors

The circuitry and physical components (**hardware**) of a computer are useless without programs (**software**). As mentioned earlier, a program is a list of instructions arranged to achieve a specific result. To give you an idea of how a computer carries out a task under program control, we'll look at two simple assembly language programs and use them to illustrate the process that occurs in any computer. A complete description of assembly language is beyond the scope of this text.

The programs in this section should give you some feel for computer software operation. Although only a few instructions will be introduced, the 80X86 family of microprocessors has a very large number of instructions available; this forms the basis of their *C*omplex *I*nstruction *S*et *C*omputer (CISC). These processors, with their complex instruction sets, allow enormous flexibility and programming power. Because of the huge installed base of software that depends on this instruction set, it is not likely to be significantly changed in the future.

Another design philosophy emerged in the 1980s that concentrated on fewer instructions, but optimizing their execution. Processors following this architecture philosophy were called *R*educed *I*nstruction *S*et *C*omputers (RISC) and have formed the basis of computers such as Apple's Power PC line. To keep the discussion within reasonable bounds, we will concentrate only on the Intel processors based on the original 8086/8088.

Any program, or series of instructions, must eventually be given to the microprocessor as machine language or **machine code,** a series of binary codes that the processor understands. While it is possible to program directly in machine language, the process is extremely difficult, prone to error, and time consuming. Assembly language is the next level above machine code. In assembly language, machine instructions are replaced with a mnemonic, a short alphabetic code that is easier for humans to remember. A mnemonic is frequently referred to as an **op-code.** The op-code usually is accompanied by one or two operands. An **operand** is a variable, a register, a memory location, or a value. The translation from mnemonics and operands written by the programmer to machine code understood by the microprocessor is accomplished by the assembler program, discussed earlier. There is a one-to-one correspondence between the assembly language instruction and the machine code produced by the assembler.

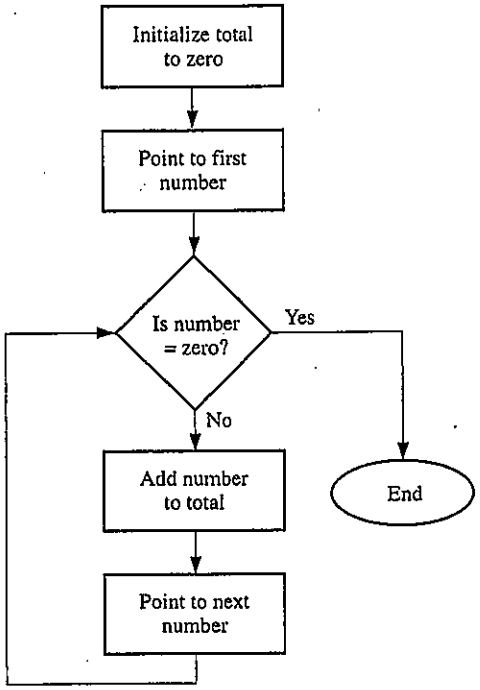
As an example of the programming process, let's assume we want the computer to add a list of unsigned numbers from memory and place the sum back in memory. The last

number in the list is a zero, which is used as a last data indicator. The steps in our example task are as follows:

1. Clear a register for the total.
2. Point to first number in memory.
3. Check to see if number is zero. If zero, end problem.
4. If number is not zero, add the number in memory to the total.
5. Point to next number.
6. Repeat steps 3, 4, and 5.

These steps are shown on the flowchart in Figure 14–15.

FIGURE 14–15
Flowchart for adding a list of unsigned numbers.



COMPUTER NOTE
Grace Hopper, a mathematician and pioneer programmer, developed considerable troubleshooting skills as a naval officer working with the Harvard Mark I computer in the 1940s. She found and documented in the Mark I's log the first real computer bug. It was a moth that had been trapped in one of the electromechanical relays inside the machine, causing the computer to malfunction. From then on, when asked if anything was being accomplished, those working on the computer would reply that they were "debugging" the system. The term stuck, and finding problems in a computer (or other electronic device), particularly the *software*, would always be known as debugging.

The following assembly language program implements the addition problem represented by the flowchart in Figure 14–15. Notice that the square brackets are used to indicate that a register is "pointing" to a memory location. Also, the program contains an assembler directive (can you find it?). An **assembler directive** (also called a pseudo-op) is an instruction to the assembler, not to the microprocessor. Two labels, "next" and "done" are shown in the program. Labels represent an address. Comments (not used by the processor) are preceded with a semicolon.

```
        mov ax,0                ; clear ax register for total
        mov bx,50H              ; point to a location in memory (50H)
                                ; where the data starts
next:   cmp word ptr [bx],0     ; check if the number pointed to is
                                ; zero
        jz done                 ; if zero, go to "done" as this is the
                                ; last data point
        add ax,[bx]             ; add memory location pointed to by bx
                                ; to the total in ax
        add bx,02               ; point to the next number (two bytes
                                ; per word)
        jmp next                ; repeat the process
done:   mov [bx],ax             ; replace the zero in memory with the
                                ; total in ax
        nop                     ; no operation—this is here to signal
                                ; the end
```

Depending on the assembler, most programs in assembly language will have a number of assembler directives that are used by the assembler to do a variety of tasks. These tasks include setting up segments, using the appropriate instruction set, describing data sizes, and performing many other "housekeeping" functions. To simplify the explanation, only one directive (required) was shown in the preceding program. The directive was **word ptr,** which is used to indicate the size of the data pointed to by the BX register.

The Debug Assembler

With a few small changes, you can run the preceding program, if you choose, by using a built-in assembler, present in DOS-based PCs. You will be able to observe it execute step by step. All DOS-based PCs have a program called **Debug** that includes a primitive assembler. To use Debug, go into DOS and type **Debug<cr>** at the DOS prompt. (**<cr>** stands for "carriage return," which means to press the ENTER key.) You should see a minus sign, which is the Debug prompt. Debug has a number of commands to observe or enter data or programs. The complete list of Debug commands will be shown if you type **?** at the Debug prompt. Before writing and executing an assembly program, you can enter some data by typing the information shown in bold:

```
-a 50
```

This tells Debug to start assembling instructions at the current data segment at an offset of 50H. Although these are data that are being entered, it is simpler to enter a 16-bit word this way. (Keep in mind that all data in Debug is entered in HEX.) Debug responds with a segment address, which will undoubtedly be different than that shown (20D8), but it doesn't matter. The offset address (50H) will be the same. Type the information shown in bold (remember each **<cr>** means press the ENTER key).

```
20D8:0050 dw 30 <cr>
20D8:0052 dw 15 <cr>
20D8:0054 dw a0 <cr>
20D8:0056 dw 0c <cr>
20D8:0058 dw 00 <cr>
<cr>
```

The **dw** is an assembler directive. It is not stored itself; it merely informs the assembler that each data point is two bytes long. In the data shown, only one byte is used, but the program will save each point in two locations with a zero in the high-order position.

You can now enter the program at location 100 as follows:

```
          -a 100 <cr>
20D8:0100 mov ax,0 <cr>
20D8:0103 mov bx,50 <cr>
20D8:0106 cmp word ptr [bx],0 <cr>
20D8:0109 jz 112 <cr>
20D8:010B add ax,[bx] <cr>
20D8:010D add bx,2 <cr>
20D8:0110 jmp 106 <cr>
20D8:0112 mov [bx],ax <cr>
20D8:0114 nop <cr>
20D8:0115 <cr>
```

To confirm that the program has been entered correctly, you can type **u 100 114** at the Debug prompt, and the code you typed will be shown on the screen. (It will be shown in capital letters). Now type **r** after the Debug prompt, and a list of the 16-bit registers and condition of the flags will appear. Notice that the IP should have 100, the starting address of the code. Underneath the list of registers, the first instruction (MOV AX, 0000) will be shown.

You can cause Debug to execute this instruction with the **t** (trace) command. This will bring onto the screen the latest condition of all of the registers and show the next instruction (MOV BX, 0050). Executing this with a **t** command will show that the number

0050 has been moved into the BX register. The steps up to this point are shown in Figure 14–16. Notice that the first data point is shown in the lower right.

Continuing in this way, you can execute the entire code and observe the changes to the registers as the microprocessor follows the instructions, as shown in Figure 14–17. This program has a common programming structure called a *loop*. A loop is a repetitive

```
-u 100 114
20D8:0100 B80000        MOV     AX,0000
20D8:0103 BB5000        MOV     BX,0050
20D8:0106 833F00        CMP     WORD PTR [BX],+00
20D8:0109 7407          JZ      0112
20D8:010B 0307          ADD     AX,[BX]
20D8:010D 83C302        ADD     BX,+02
20D8:0110 EBF4          JMP     0106
20D8:0112 8907          MOV     [BX],AX
20D8:0114 90            NOP
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=20D8  ES=20D8  SS=20D8  CS=20D8  IP=0100   NV UP EI PL ZR NA PE NC
20D8:0100 B80000        MOV     AX,0000
-t

AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=20D8  ES=20D8  SS=20D8  CS=20D8  IP=0103   NV UP EI PL ZR NA PE NC
20D8:0103 BB5000        MOV     BX,0050
-t

AX=0000  BX=0050  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=20D8  ES=20D8  SS=20D8  CS=20D8  IP=0106   NV UP EI PL ZR NA PE NC
20D8:0106 833F00        CMP     WORD PTR [BX],+00                DS:0050=0030
-
```

FIGURE 14–16
Steps in beginning to execute the addition program with Debug.

```
DS=20D8  ES=20D8  SS=20D8  CS=20D8  IP=0110   NV UP EI PL NZ NA PO NC
20D8:0110 EBF4          JMP     0106
-t

AX=00F1  BX=0058  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=20D8  ES=20D8  SS=20D8  CS=20D8  IP=0106   NV UP EI PL NZ NA PO NC
20D8:0106 833F00        CMP     WORD PTR [BX],+00                DS:0058=0000
-t

AX=00F1  BX=0058  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=20D8  ES=20D8  SS=20D8  CS=20D8  IP=0109   NV UP EI PL ZR NA PE NC
20D8:0109 7407          JZ      0112
-t

AX=00F1  BX=0058  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=20D8  ES=20D8  SS=20D8  CS=20D8  IP=0112   NV UP EI PL ZR NA PE NC
20D8:0112 8907          MOV     [BX],AX                          DS:0058=0000
-t

AX=00F1  BX=0058  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=20D8  ES=20D8  SS=20D8  CS=20D8  IP=0114   NV UP EI PL ZR NA PE NC
20D8:0114 90            NOP
-d 0050 005f
20D8:0050  30 00 15 00 A0 00 0C 00-F1 00 00 00 00 20 20 20  0.............
-
           Original data                    Sum
```

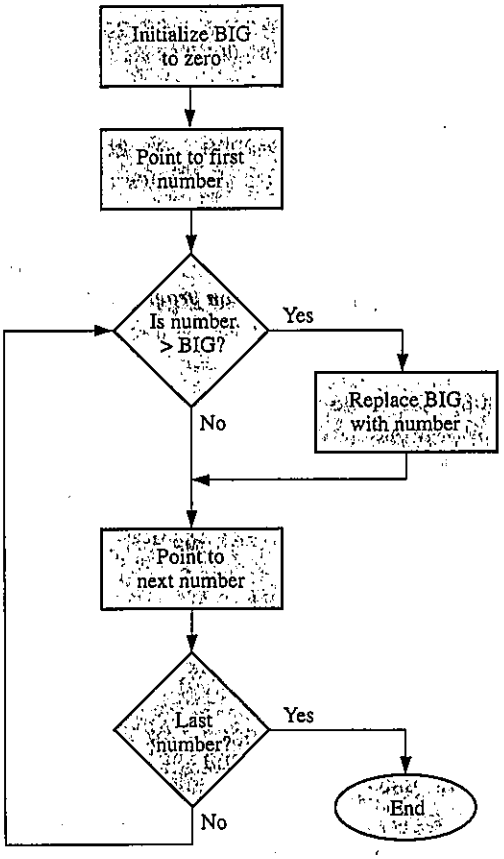
FIGURE 14–17
Last portion of tracing the addition program. The sum 00F1 is shown in blue with the low part (F1) given first.

group of instructions that are executed until some condition is met; in this case, the condition is finding a zero in the data. After the zero has been found, the last instruction will be executed and the sum will be stored (in this case, 00F1 is the hex sum) in place of the zero that indicated the last data point. You can observe this by pressing **d 0050 005F** (display between addresses 0050 and 005F) at the Debug prompt when you reach the last instruction (NOP), as shown in Figure 14–17. The result appears as the 9th and 10th bytes (the 5th word) on the line following the display instruction. Notice that the least significant part of the answer is shown first. When executed in "real time" by the microprocessor, this program actually uses only about 1 μs to do this entire process. If you choose to repeat the process, you will need to reload the zero at location 0058 because it has been replaced with the sum. (Recall that the program uses the zero as a "last data point" sensor.)

EXAMPLE 14–2 Write the instructions for an assembly language program that will find the largest unsigned number in the data and place it in the last position. Assume the last data point is signaled with a zero.

Solution The flowchart is shown in Figure 14–18.

FIGURE 14–18
Flowchart. The variable BIG represents the largest value.



The data is assumed to be the same as before. The program listing (with comments) is as follows:

```
        mov ax,0000          ; initial value of BIG is in the ax
                             ; register
        mov bx,0050          ; point to a location in memory
                             ; (50H) where the data starts
repeat: cmp [bx],ax          ; is data point larger than BIG?
        jbe check            ; if the data point is smaller, go
                             ; to "check"
```

```
          mov ax,[bx]             ; otherwise, put new largest data
                                  ; point in ax
check:    add bx,02               ; point to the next number in
                                  ; memory (two bytes per word)
          cmp word ptr [bx],0     ; test for last data point
          jnz repeat              ; continue if the data point is not
                                  ; a zero
          mov [bx],ax             ; save BIG in memory
          nop                     ; no operation
```

The Debug listing of this program is shown in Figure 14–19. Data is entered in the same way as before starting at location 0050. In this case the same data is used, but you may choose new data if you prefer. It is important that a zero be entered as the last data point because the program continues until it finds this point. The zero is replaced each time the program is run with the largest data point. The program is entered by starting the assembly at location 100 by entering the program after the a **100** command is issued.

FIGURE 14–19
Listing of Debug portion of program.

```
-a 100
20D8:0100 mov ax,0
20D8:0103 mov bx,50
20D8:0106 cmp [bx],ax
20D8:0108 jbe 10c
20D8:010A mov ax,[bx]
20D8:010C add bx,2
20D8:010F cmp word ptr [bx],0
20D8:0112 jnz 106
20D8:0114 mov [bx],ax
20D8:0116 nop
20D8:0117
```

The program can be traced to watch the execution, one step at a time. Alternatively, you can enter **g= 100 116** to "go" between address 100 and 116. Figure 14–20 shows the data before and after execution. Note that each data point is stored in two

```
-a 100
20D8:0100 mov ax,0
20D8:0103 mov bx,50
20D8:0106 cmp [bx],ax
20D8:0108 jbe 10c
20D8:010A mov ax,[bx]
20D8:010C add bx,2
20D8:010F cmp word ptr [bx],0
20D8:0112 jnz 106
20D8:0114 mov [bx],ax
20D8:0116 nop
20D8:0117
-d 50 5f
20D8:0050  30 00 15 00 A0 00 0C 00-00 00 00 00 00 20 20 20   0...........
-g= 100 116

AX=00A0  BX=0058  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=20D8  ES=20D8  SS=20D8  CS=20D8  IP=0116   NV UP EI PL ZR NA PE NC
20D8:0116 90            NOP
-d 50 5f
20D8:0050  30 00 15 00 A0 00 0C 00-A0 00 00 00 00 20 20 20   0...........
```

End of data signal

Data before executing program

Data are unchanged.

End of data now has largest data point.

FIGURE 14–20
Data before and after a run.

bytes (although the data is only one byte long) because the data was defined as words. Also, note that the low byte preceded the high byte in memory. After the program is run, the last data point (formerly zero) is seen to be equal to the largest value (A0 in this example). This value is seen to be in both the AX register and in memory.

Related Problem Explain how you could change the flowchart to find the smallest number in the list instead of the largest.

Types of Instructions

The programs in this section only show a few of the hundreds of variations of instructions available to programmers. To simplify learning the Intel instruction set, instructions are divided into seven categories. These categories are described here.

Data Transfer The most basic data transfer instruction MOV was introduced in the example programs. The MOV instruction, for example, can be used in several ways to copy a byte, a word (16 bits), or a double word (32 bits) between various sources and destinations such as registers, memory, and I/O ports. (A better mnemonic for MOV might have been "COPY" because this is what the instruction actually does.) Other data transfer instructions include IN (get data from a port), OUT (send data to a port), PUSH (copy data onto the stack, a separate area of memory), POP (copy data from the stack), and XCHG (exchange).

Arithmetic There are a number of instructions and variations of these instructions for addition, subtraction, multiplication, and division. The ADD instruction was used in both example programs. Other arithmetic instructions include INC (increment), DEC (decrement), CMP (compare), SUB (subtract), MUL (multiply), and DIV (divide). Variations of these instructions allow for carry operations and for signed or unsigned arithmetic. These instructions allow for specification of operands located in memory, registers, and I/O ports.

Bit Manipulation This group of instructions includes those used for three classes of operations: logical (Boolean) operations, shifts, and rotations. The logical instructions are NOT, AND, OR, XOR, and TEST. An example of a shift instruction is SAR (shift arithmetic right). An example of a rotate instruction is ROL (rotate left). When bits are shifted out of an operand, they are lost; but when bits are rotated out of an operand, they are looped back into the other end. These logical, shift, and rotate instructions can operate on bytes or words in registers or memory.

Loops and Jumps These instructions are designed to alter the normal (one after the other) sequence of instructions. Most of these instructions test the processor's flags to determine which instruction should be processed next. In Example 14–2, the instruction JBE and JNZ were used to alter the path. Other instructions in this group include JMP (unconditional jump), JA (jump above), JO (jump overflow), LOOP (decrement the CX register and repeat if not zero) and many others.

Strings A **string** is a **contiguous** (one after the other) sequence of bytes or words. Strings are common in computer programs. A simple example is a sentence that the programmer wishes to display on the screen. There are five basic string instructions that are designed to copy, load, store, compare, or scan a string—either as a byte at a time or a word at a time. Examples of string instructions are MOVSB (copy a string, one byte at a time) and MOVSW (copy a string, one word at a time).

Subroutine and Interrupts A **subroutine** is a miniprogram that can be used repeatedly but programmed only once. For example, if a programmer needs to convert ASCII numbers from a keyboard to a BCD format, a simple programming structure is to make the

required instructions a separate process and "call" the process whenever necessary. Instructions in this group include CALL (begin the subroutine) and RET (return to the main program).

Processor Control This is a small group of instructions that allow direct control of some of the processor's flags and other miscellaneous tasks. An example is the STC (set carry flag) instruction.

SECTION 14–4 REVIEW

1. Define *program*.
2. What is a complex instruction set?
3. What is an op-code?
4. What is an operand?
5. What is a string?

14–5 ■ THE CENTRAL PROCESSING UNIT (CPU)

A typical CPU (or MPU) consists of a microprocessor and various associated ICs called support circuits. First, we will look at the general operation of a CPU, and then we will examine a specific implementation using the 8088 microprocessor as a basic model and discuss some differences with it and newer processors.

After completing this section, you should be able to

- Explain the purpose of the CPU in a computer
- Discuss system buses
- Describe a basic CPU using an 8088 microprocessor
- Explain the purpose of a bus controller
- Discuss bus interfacing

As you saw in the last section, the instructions for the 80X86 family needed to be upward compatible to avoid the problem of making older programs immediately obsolete when a new processor was introduced. (Motorola did the same with its 68000 family.) Instructions for the 8086/8088 could run on the new processors (but not the reverse). Although the original 8086/8088 has long been obsolete for PCs, many controllers still use it and the related microcontroller ICs. In addition, many of the significant improvements that occurred impacted the speed of the processor (such as clock speed and pipelining), but these improvements had a relatively small effect on the basic programming structure. (An exception is in the addition of protected mode which is complicated.) For these reasons, a study of the 8086/8088 CPU and supporting ICs is a useful starting point for understanding the basic architecture of computers. We will introduce the 8086/8088 processors in this discussion, but highlight some key enhancements that have occurred with newer processors to give you a sense of important changes in technology.

An 8088-Based CPU

A typical CPU (or MPU) is implemented with a microprocessor and some additional support circuits. Figure 14–21 shows an 8088-based CPU.

The 8086/8088 Microprocessor As you know, the 8088 is basically the same as the 8086 microprocessor except that the 8088 has an 8-bit external data bus rather than a 16-bit data bus. The 8088 with its 8-bit data bus makes it compatible with many 8-bit peripheral devices. Internally, however, the 8088 can handle data in 16-bit words or in bytes, depending on how it is programmed. Also, the 8088, like the 8086, has a 20-bit address bus, which allows up to 1 Mbyte of memory to be addressed.

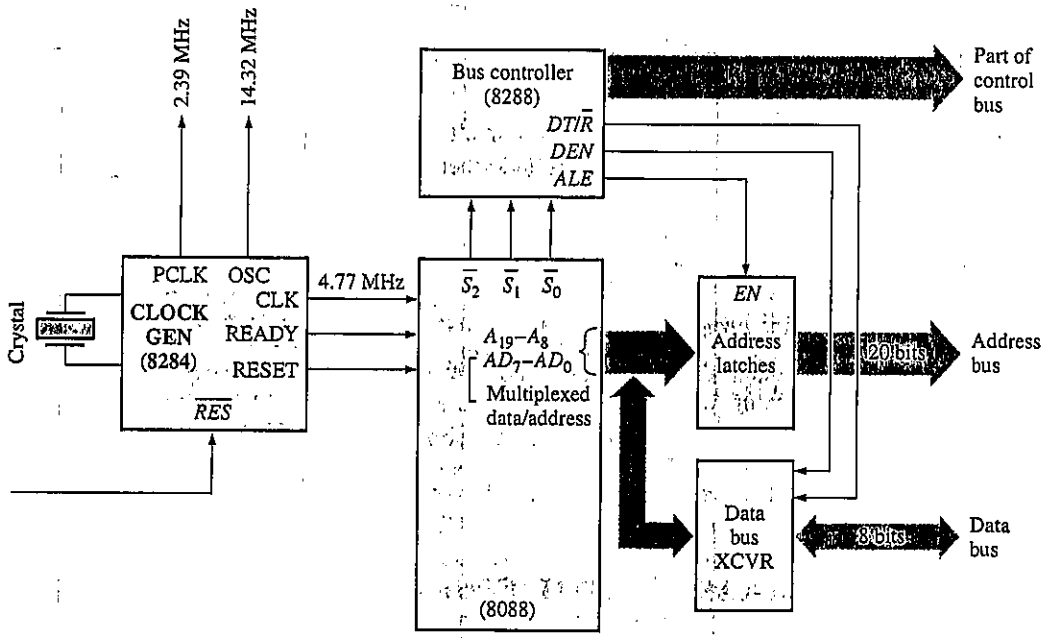
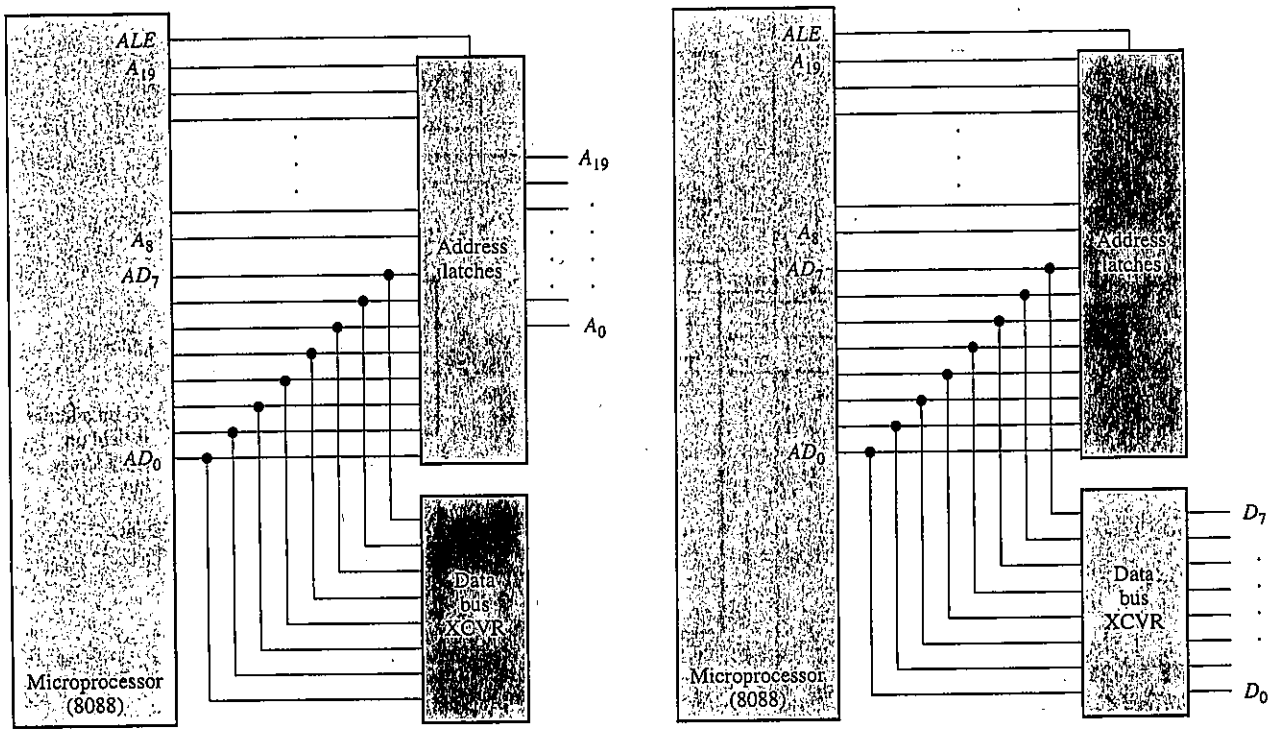


FIGURE 14–21
A simplified 8088-based CPU.

In the 8086/8088 processors, the data bus is combined with the lower address lines. There are a total of twenty lines used for addressing; in the 8088 these lines are AD_0–AD_7 and A_8–A_{19}. The eight lines designated AD_0–AD_7 serve as both data and address lines, using a multiplexed operation; A_8–A_{19} serve strictly for addressing. In the 8086, the lower 16 lines (AD_0–AD_{15}) are multiplexed; A_{16}–A_{19} serve strictly for addressing. The 80186/80188 also use multiplexed data and address lines, but these are the only processors in the 80X86 series that use multiplexed data and address lines. The principal advantage of this method was to eliminate pins, but these processors require extra circuitry to demultiplex these lines.

When a 20-bit address is sent out by the 8088 microprocessor, all twenty lines are used as the address bus, as illustrated in Figure 14–22(a). When the address is sent, a pulse called *ALE* (address latch enable) is also sent to signal external circuitry that a stable address is on the *AD* lines and can be latched. When data are sent or received by the microprocessor, the lower lines, (AD_0–AD_7 on the 8088) are used as a bidirectional data bus, as shown in Figure 14–22(b). The bus multiplexing is controlled by the bus controller. The bus controller is one of the support circuits. Other microprocessor input and output lines will be described in association with the appropriate support circuits.

The Clock Generator A clock generator circuit, such as the 8284, provides the basic timing signals to the 8088 and to other parts of the system including peripherals. It also provides a RESET signal to the 8088 to initialize the internal circuitry and a READY signal to synchronize the microprocessor with the rest of the system. An external resonant crystal is used to establish the frequency of oscillation (14.31818 MHz in this case), as shown in Figure 14–23. This frequency is available on the OSC output. A divide-by-three counter inside the clock generator produces a 4.772727 MHz clock with a duty cycle of 33% to run the microprocessor. This signal is available on the CLK output. Also, a 2.386363 MHz signal is derived from the CLK frequency and is available on the PCLK output of the clock generator. The 8086/8088 was the only processor that used the 8284 clock generator; however, the 80286 used a similar IC named the 82284. Later processors had many of the clock generator functions located internally to reduce the components required for the system. The Pentium requires an external oscillator running at the same frequency (currently several hundred MHz) as the system to control the basic system speed.



(a) Address latches enabled

(b) Data bus transceiver (XCVR) enabled

FIGURE 14–22
Multiplexed bus operation of the 8088.

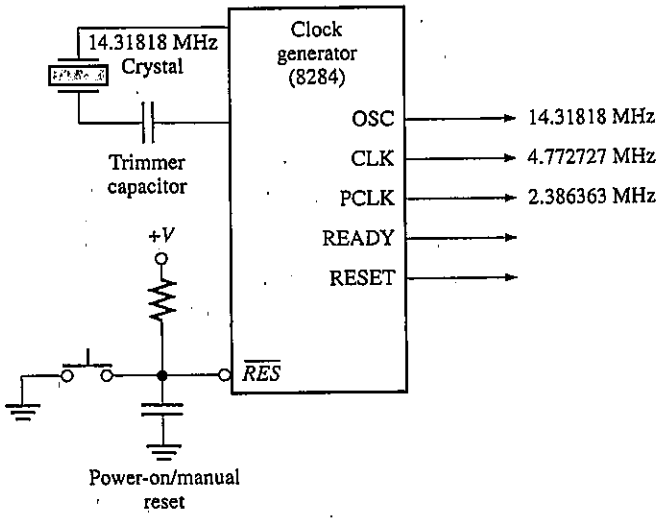


FIGURE 14–23
The 8284 clock generator.

The Bus Controller The 8288 bus controller is used to relieve the 8088 microprocessor of most bus control functions in order to increase the processing efficiency. One function of the bus controller in the CPU is to provide the signals to multiplex the AD_0–AD_7 bus lines coming from the microprocessor. The bus controller generates these control signals based on the codes on the status lines ($\overline{S}_0$, $\overline{S}_1$, $\overline{S}_2$) from the microprocessor.

When the microprocessor needs to communicate with memory or I/O, it sends a code on the status lines to the bus controller. The bus controller then generates an *ALE* signal, which latches the address placed on AD_0–AD_7 and A_8–A_{19} by the microprocessor into the

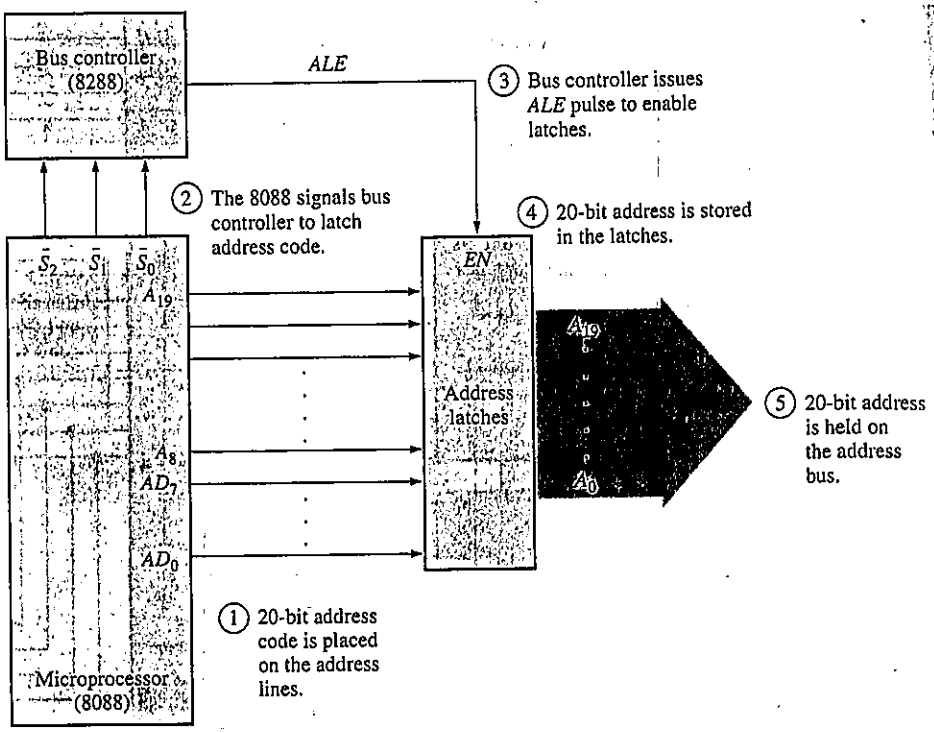


FIGURE 14–24
Steps in placing a valid address on the bus.

address latches. There are twenty latches (A_0–A_{19}), whose tristate outputs form the system address bus. Figure 14–24 illustrates the steps in this operation. During a memory read/write cycle, a valid address is available on the address bus; therefore, the lower eight bits (AD_0–AD_7) of the microprocessor's combined bus are freed up to send or receive data.

After the address has been latched, the microprocessor signals the bus controller via the status lines ($\overline{S}_0$, $\overline{S}_1$, and $\overline{S}_2$) that it wants to read data or write data to the memory location or I/O that is specified by the address that is held on the address bus. The status line codes for the four possible read/write operations are given in Table 14–1.

TABLE 14–1
The 8088 read/write status codes.

Status Code			Operation	Active Bus Controller Output Command
$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$		
0	0	1	Read I/O port	$\overline{IORC}$
0	1	0	Write I/O port	$\overline{IOWC}$
1	0	1	Read memory	$\overline{MRDC}$
1	1	0	Write to memory	$\overline{MWTC}$

When the bus controller receives one of the status codes, it sends a read command ($\overline{IORC}$ or $\overline{MRDC}$) or a write command ($\overline{IOWC}$ or $\overline{MWTC}$) to the appropriate device, and it also sends two signals to the tristate data bus transceiver (*XCVR*). These two signals are $DT/\overline{R}$ (data transmit/$\overline{\text{receive}}$) and *DEN* (data enable). The *DEN* signal enables the data bus transceiver, and the $DT/\overline{R}$ signal selects the direction of data on the data bus (D_0–D_7). Figure 14–25 illustrates an 8088 memory-read operation, and Figure 14–26 on page 770 illustrates an 8088 memory-write operation. Timing diagrams for the memory read and write cycles are given in Section 14–6.

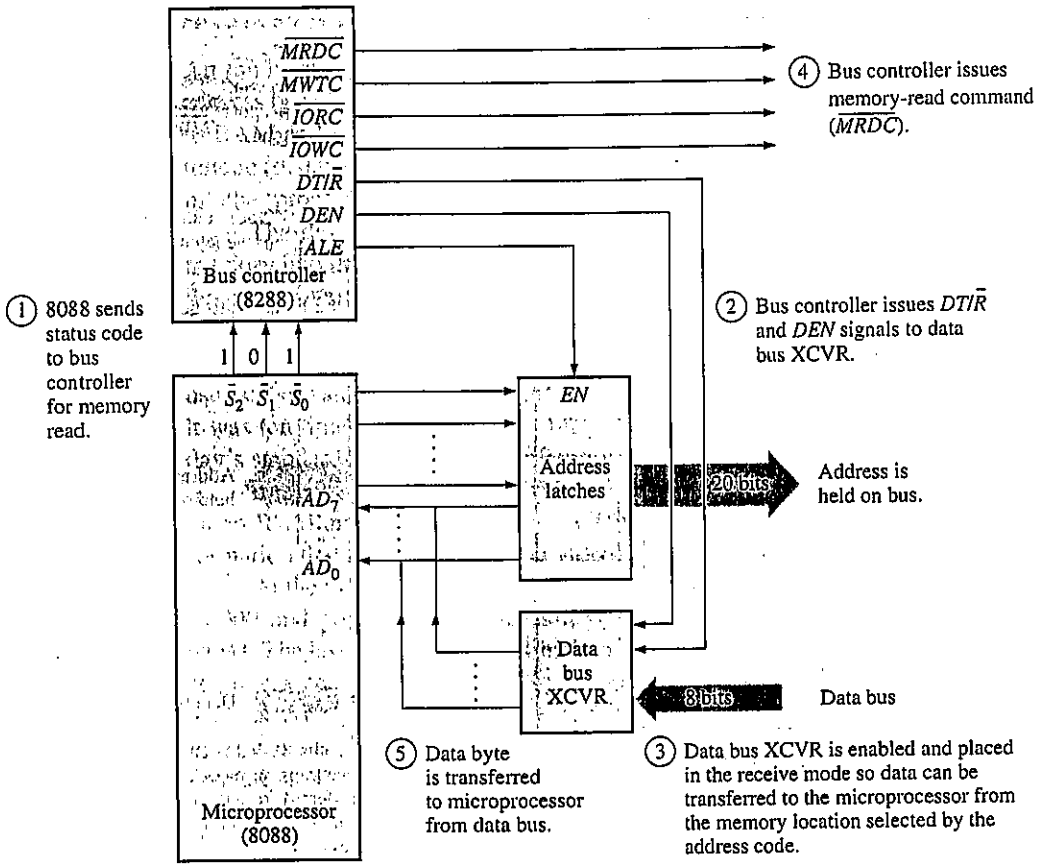


FIGURE 14–25
Example of an 8088 memory-read operation.

While many signals have been added to newer microprocessors for various tasks (cache control for example), other interface signals have remained the same. The type of bus cycle and the interrupt signals (with one addition) remain essentially the same as in the 8086/8088.

Processors starting with the 80286 did not multiplex the address or data and therefore did not require address latch circuits, but the number of pins increased and required new packaging. The 8086/8088 was originally packaged in a 40-pin dual-in-line package. Newer processors may have several hundred pins; the Pentium Pro, for example, has 387 pins. Package technology has improved dramatically in the last decade, including increased numbers of pins, increased thermal performance, and smaller sizes with significantly greater density of parts.

CPU Enhancements

A major change in the CPU since the 8086/8088 is the use of "pipelined" architecture. Pipelining speeds up the processing of instructions by fetching and executing instructions in many stages, each working in parallel. Some instructions are able to be executed in a single clock cycle in the Pentium that formerly took as much as 16 or more cycles on the 8086/8088.

A second important enhancement is the addition and expansion of cache memory. Cache memory can be either internal or external to the processor. An internal cache memory is designed to be very fast. It can store instructions or data and increases the processor speed by eliminating delays required to access external memory. The original Pentium was the first CPU to feature separate data and instruction caches.

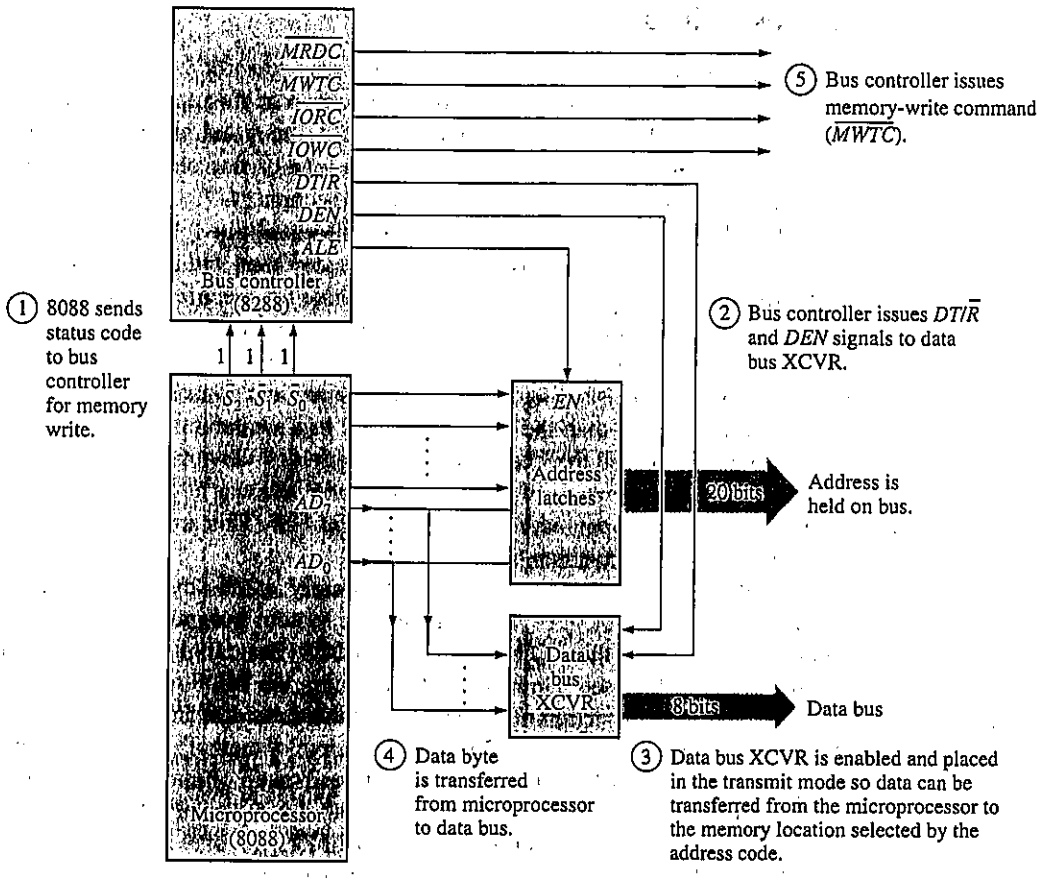


FIGURE 14–26
Example of an 8088 memory-write operation.

A third enhancement is the inclusion of the internal math coprocessor. The internal coprocessor performs math functions much faster and more efficiently than the coprocessors from the 80486 and earlier. In addition, internal dual integer processors and branch prediction logic are added in the Pentium. These and a number of other enhancements changed the requirements for the bus control logic which is now implemented with PLDs.

SECTION 14–5 REVIEW

1. What is the purpose of the clock generator in the CPU?
2. List two functions of the bus controller.
3. Explain why bus multiplexing is necessary in an 8088-based CPU.
4. What are three enhancements to the Pentium that were unavailable in the 8086/8088?

14–6 ■ THE MEMORY

This section covers the memory portion of a typical computer and discusses how it is used by the CPU.

After completing this section, you should be able to

■ Explain address allocation in computer memories ■ Discuss the read and write operations ■ Describe a memory-read cycle ■ Describe a memory-write cycle

Address Allocation

As you have seen, the 8086/8088 CPU can address up to 1,048,576 locations with a 20-bit address bus (A_0–A_{19}). In a typical computer, a portion of the total address space is allocated to RAM and ROM and a separate portion to I/O. Input and output ports have their own unique (dedicated) memory address space in the Intel processors but use memory mapping in other processors. This is discussed further in Section 14–7.

The designers of the original IBM PC allocated the 1 Mbyte of RAM and ROM memory into three general areas: a user area, a video area, and a system area. Within these three areas, certain reserved locations were set aside for specific purposes. The lower RAM area (called conventional memory) was a 640-kbyte block that started at the bottom of memory (location 0) and was principally available for users (a small portion is set aside for use by the system). This seemed like a very large amount of user memory at the time (since it was ten times more than the 8-bit processors had available), but it is very small by today's standards. (Recall that the Pentium can address 4 Gbytes!).

Immediately above the 640-kbyte block of RAM was a reserved 128-kbyte block of video RAM area between address A0000 and BFFFF. This area was set aside for the information that is displayed on the video screen.

At the top of the available 1 Mbyte of memory, a 256-kbyte block starting at address C0000 and going to FFFFF was assigned for system programs and implemented with ROM. The last part of this ROM included the BIOS (Basic Input Output System). Upon reset, the system addressed memory locations within the BIOS to "boot" the system. BIOS also reserved a small section of RAM (between address 400 and 4FF) as a work area, and the system used other parts of RAM near the bottom of memory to hold certain required information. Most of this structure has been retained on all of the Intel microprocessors.

Figure 14–27 shows a basic memory map for a typical 8086/8088 computer. The RAM is allocated addresses 00000_{16} through $BFFFF_{16}$ (a total of 786,431 locations) which includes the RAM and video areas mentioned. The ROM is allocated addresses $C0000_{16}$ through $FFFFF_{16}$ (a total of 262,143 locations).

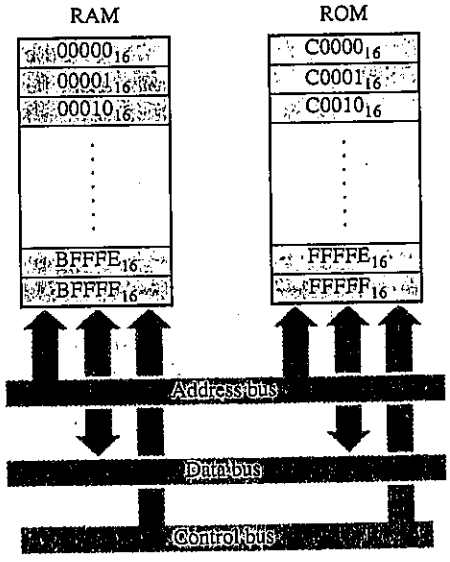


FIGURE 14–27
Example of a typical memory address allocation within an 8086/8088 computer.

As the number of address lines increased in succeeding processors, much of the earlier memory organization was retained; however, certain significant changes were introduced. With the 80386, a new *paging* mechanism was added to the processor for accessing memory. This unit was part of a memory management unit (MMU) that had its roots in the 80286 microprocessor. Paging allowed programs to be located anywhere in the available space and still operate properly by translating the addresses. This means that a program that required access to a certain location could function even if the location was

occupied by another program. In order to keep compatibility with earlier processors, the memory above FFFFF is referred to as extended memory. From 00000 to FFFFF, the memory is referred to as the real memory area. Programs operating in real mode operate in this area.

CPU-Memory Operation

The CPU handles two types of memory transfers, *read* and *write*. Basically, during the read operation, the CPU fetches either a program instruction, another address (that appears as data), or data itself. During the write operation, the CPU sends data resulting from a computation or some other operation back to RAM. A diagram of the 8088 CPU and memory portions of a computer is shown in Figure 14–28.

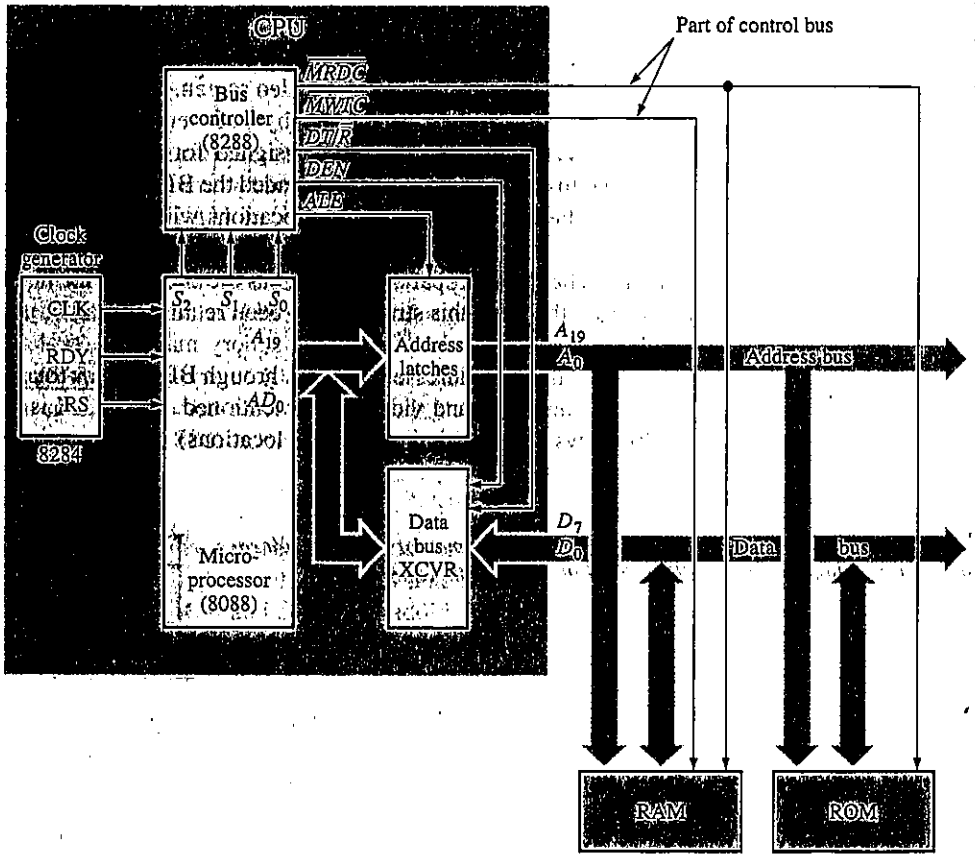
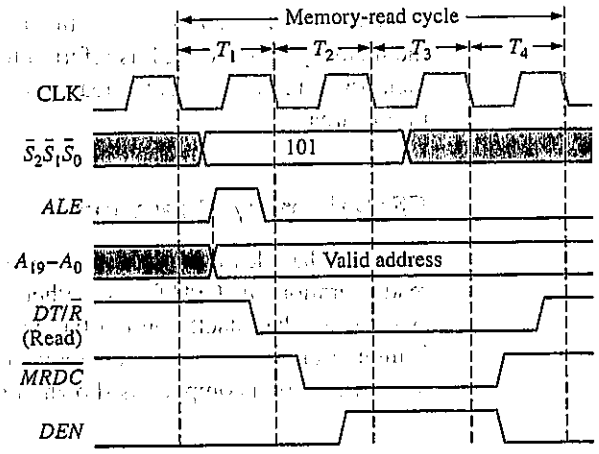


FIGURE 14–28
8088 CPU memory organization.

Memory-Read Cycle The 8086/8088 microprocessor initiates a memory-read cycle by sending a status code of $\overline{S}_2\,\overline{S}_1\,\overline{S}_0 = 101$ to the bus controller and placing a binary address on the AD_0–AD_7 and A_8–A_{19} bus lines. The bus controller then issues an *ALE* signal, which latches the address onto the address bus (A_0–A_{19}), freeing up the AD_0–AD_7 lines for data transfer. Also, the bus controller issues a low $DT/\overline{R}$ signal to the bus transceiver to set it up for receiving data from the memory and passing it on to the microprocessor.

Next, the bus controller issues a memory read command ($\overline{MRDC}$) to the memory. This command instructs the memory to place the contents of the selected address on the data bus. The *DEN* signal from the bus controller then enables the bus transceiver, and a

FIGURE 14–29
Basic timing diagram for an 8088 memory-read cycle.

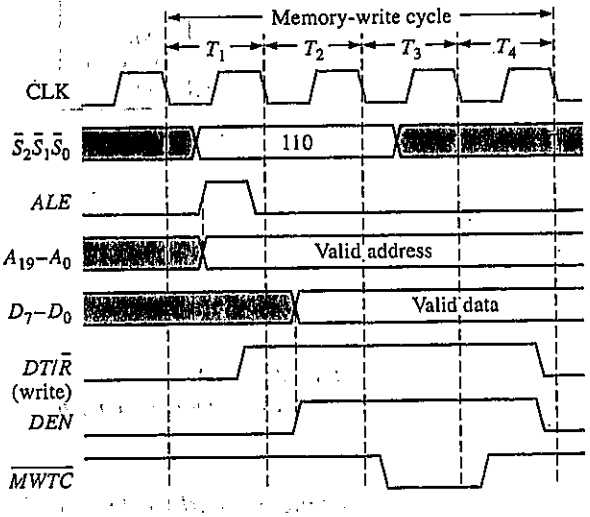


byte of data flows into the microprocessor. Figure 14–29 shows the basic timing diagram for an 8088 memory-read cycle.

Memory-Write Cycle The 8088 microprocessor initiates a memory-write cycle by sending status code $\overline{S}_2\ \overline{S}_1\ \overline{S}_0 = 110$ to the bus controller and placing a binary address on the bus lines. The bus controller then issues an *ALE* signal to latch the address onto the address bus (A_0–A_{19}) and a high $DT/\overline{R}$ signal to set up the bus transceiver for sending data to the memory. The microprocessor then places the data byte on the AD_0–AD_7 lines. The *DEN* signal from the bus controller then enables the bus transceiver to place the data byte on the data bus (D_0–D_7).

Next, the bus controller issues a memory-write command ($\overline{MWTC}$) to the memory. This command instructs the RAM to store the data byte in the selected address. Figure 14–30 shows a basic timing diagram for an 8088 memory-write cycle.

FIGURE 14–30
Basic timing diagram for an 8088-based memory-write cycle.



SECTION 14–6 REVIEW

1. Name the two cycles used by the 8088 CPU to transfer data to and from the memory.
2. How many clock cycles does it take to read data from the memory in an 8088-based system?
3. When does a valid address appear on the address bus (A_0–A_{19}) during a write cycle?

14–7 ■ THE INPUT/OUTPUT (I/O) PORT

In this section, you will see generally how the CPU communicates with input and output devices via I/O ports. An I/O port can be thought of as a "window" between the computer and the outside world through which information is passed.

After completing this section, you should be able to

■ Explain what an I/O port is ■ Discuss the basic types of ports ■ Explain the difference between dedicated and memory-mapped I/O ports ■ State the purpose of a programmable peripheral interface

A block diagram of a basic computer system, including I/O ports and typical peripherals, is shown in Figure 14–31. Notice that a port can be strictly for input, strictly for output, or bidirectional for both input and output. Although only four I/O ports are shown in the figure, the CPU is capable of handling many more than that. All of the Intel 80X86 and Pentium processors can address 64,000 ports!

As mentioned, an **I/O port** provides an interface between the computer and the "outside world" of peripheral equipment. Depending on the system design, the I/O ports can be

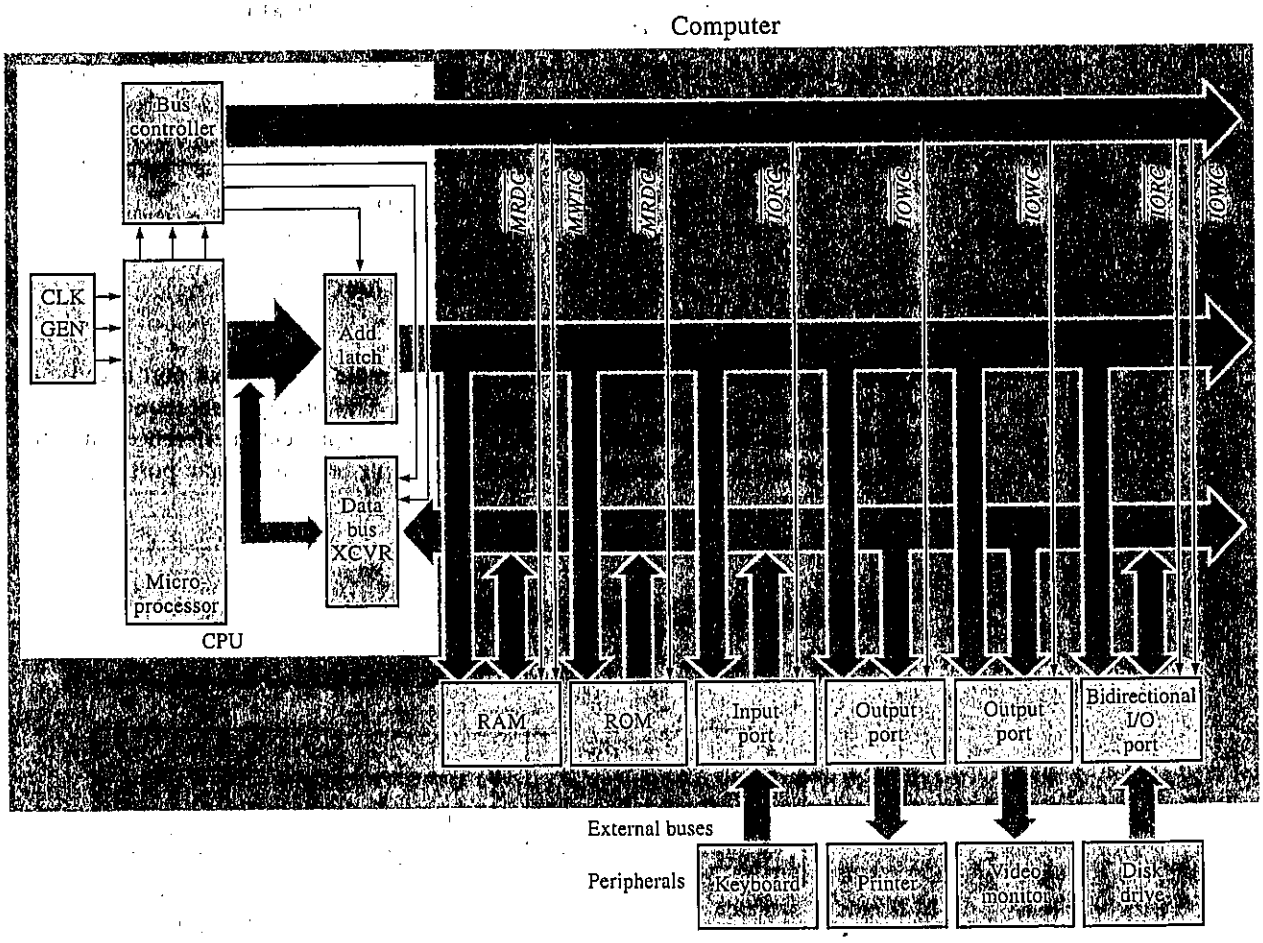


FIGURE 14–31
Basic 8088 computer system with I/O ports and peripherals.

either *dedicated* or *memory mapped.* These terms describe the ways in which the ports are accessed by the CPU.

Dedicated I/O Ports

A dedicated I/O port is assigned a unique address within the I/O address space of the computer, and dedicated I/O commands are used for communication. This address space is completely separate from memory addresses. Dedicated I/O ports are accessed by the I/O read and write commands, $\overline{IORC}$ and $\overline{IOWC}$ in an 8086/8088-based system.

Output Port The basic operation of the output port is as follows: The port occupies one location within the I/O address space of the system; that is, it has a unique address different from the addresses of other ports. The lower 16 bits of the address bus are used for port addresses because the I/O space is mapped to 64k (2^{16} = 64k).

During a CPU write cycle, the port address is placed on the address bus, and the CPU issues a low $\overline{IOWC}$ (I/O write command) signal to enable the address decoder. The decoder then produces a signal to latch the data byte onto the port output lines. Figure 14–32 shows how a dedicated output port might be implemented.

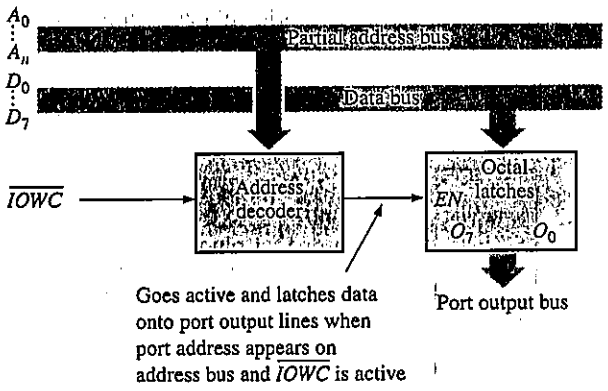


FIGURE 14–32
A basic dedicated output port.

Input Port A dedicated input port is not quite as simple as an output port because the port output lines are connected to the system data bus. This arrangement requires that the port output lines be disabled when the port is not in use to prevent interference with other activity on the data bus. Figure 14–33 shows an example of a basic input port implementation.

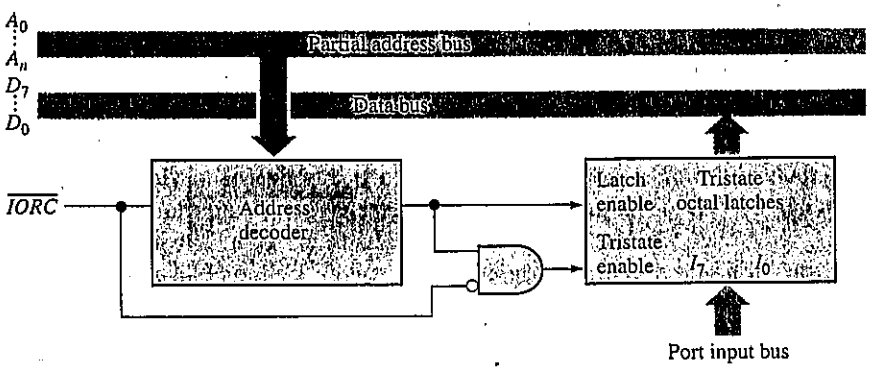


FIGURE 14–33
A basic dedicated input port.

Bidirectional Port The third type of dedicated I/O port is the bidirectional port, which is basically a combination of the input and output ports with additional enabling controls. A bidirectional port allows data to flow to or from a peripheral device such as a disk drive.

Memory-Mapped I/O Ports

The second approach to accessing I/O ports is the **memory-mapped port.** The port implementation is essentially the same as for the dedicated ports in Figure 14–32 and 14–33 except for two things:

1. The memory-mapped ports are assigned addresses within the computer memory address space and are actually viewed as memory locations by the CPU. All twenty address lines are used for memory-mapped I/O in the 8086/8088.
2. The CPU accesses a memory-mapped I/O port with the $\overline{MRDC}$ and $\overline{MWTC}$ commands rather than the $\overline{IORC}$ and $\overline{IOWC}$ commands. In other words, the memory-mapped I/O ports are treated exactly as memory locations. Instead of using the IN and OUT instruction in assembly language, memory mapped I/O uses the MOV instruction.

Although memory-mapped I/O design is simple, it has the disadvantage of using memory space, with the potential for fragmentation of memory. In addition, decoding the address requires the entire address bus rather than only part of the address bus.

Programmable Peripheral Interface

In many systems, special ICs are used to implement the I/O ports. One such device is the 8255A programmable peripheral interface **(PPI),** shown in Figure 14–34. This PPI chip continues to be very popular and has been used in many I/O applications and on add-on cards that plug into the computer's bus. The address decoder is a separate device for enabling the PPI when a port address is decoded. This particular PPI provides three I/O ports. Ports A and B have 8 lines each, and port C is split into two 4-line groups. The ports can be programmed for various combinations of input ports, output ports, and bidirectional ports and for control and handshaking lines by writing a word to a special control register. This means that the port can be dynamically configured; that is, software can change the size and type of port and can add or delete handshaking. The PPI can be used for either dedicated or memory-mapped ports.

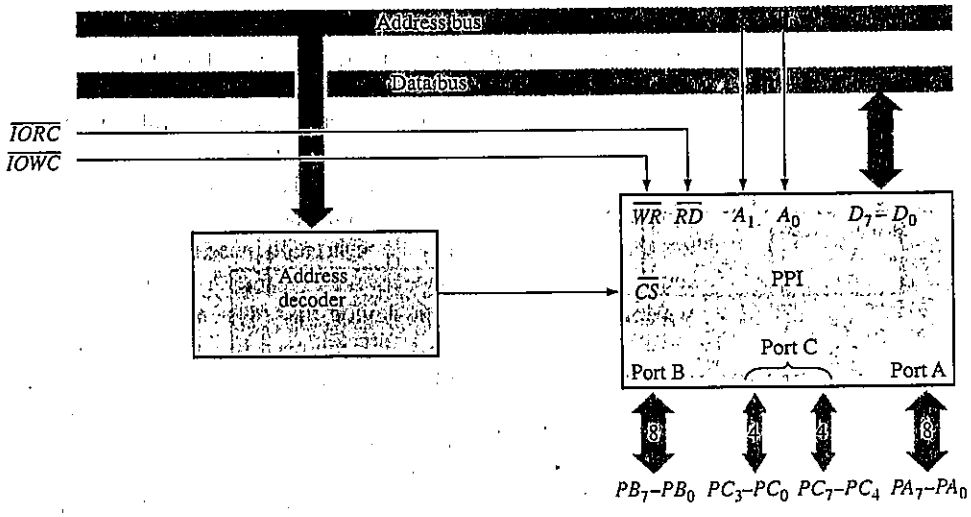


FIGURE 14–34
The 8255A programmable peripheral interface (PPI).

SECTION 14–7 REVIEW

1. What is an I/O port?
2. Explain the difference between a dedicated I/O port and a memory-mapped I/O port.
3. What is the purpose of a PPI?

14–8 ■ INTERRUPTS

The concept of an I/O port was presented in Section 14–7, and both dedicated and memory-mapped ports were introduced. In this section, the establishment of communications between a peripheral and the CPU is presented. Three methods are discussed: polled I/O, interrupt-driven I/O, and software interrupts.

After completing this section, you should be able to

■ Discuss the need for interrupts in a computer system ■ Describe the basic concept of a polled I/O ■ Describe the basic concept of an interrupt-driven I/O ■ Discuss a software interrupt

In microprocessor-based systems such as the personal computer, peripheral devices require periodic service from the CPU. The term *service* generally means sending data to or taking data from the device or performing some updating process. There are three ways that a service routine can be started: *polled I/O, interrupt driven I/O,* or *software interrupts.* An **interrupt** is a signal or instruction that causes the current process to be temporarily stopped while a service routine is run. The three methods for service are described further in this section.

In general, peripheral devices are very slow compared with the CPU. For example, the 8086/8088 CPU can perform about 1,000,000 read or write operations in one second (based on a 4.77 MHz clock frequency). A printer may average only a few characters per second (one character is represented by eight bits), depending on the type of material being printed and the type of printer. A keyboard input rate may be one or two characters per second, depending on the speed of the operator. So, in between the times that the CPU is required to service a peripheral, it can do a lot of processing. In most systems, this processing time must be maximized by using an efficient method of servicing the peripherals.

Polled I/O

One method of servicing the peripherals is called **polling.** In this method, the CPU must test each peripheral device in sequence at certain intervals to see if it needs or is ready for servicing. Figure 14–35 illustrates the basic polled I/O method.

The CPU sequentially selects each peripheral device via the multiplexer to see if it needs service by checking the state of its ready line. Certain peripherals may need service at irregular and unpredictable intervals, that is, more frequently on some occasions than on others. Nevertheless, the CPU must poll the device at the highest rate. For example, let's say that a certain peripheral occasionally needs service every 1000 μs but most of the time requires service only once every 100 ms. As you can see, precious processing time is wasted if the CPU polls the device, as it must, at its maximum rate (every 1000 μs) because most of the time the device will not need service when it is polled.

Each time the CPU polls a device, it must stop the program that it is currently processing, go through the polling sequence, provide service if needed, and then return to the point where it left off in its current program.

Another problem with the sequentially polled I/O approach is that if two or more devices need service at the same time, the first one polled will be serviced first; the other devices will have to wait although they may need servicing much more urgently than the first device polled. As you can see, polling is suitable only for devices that can be serviced at

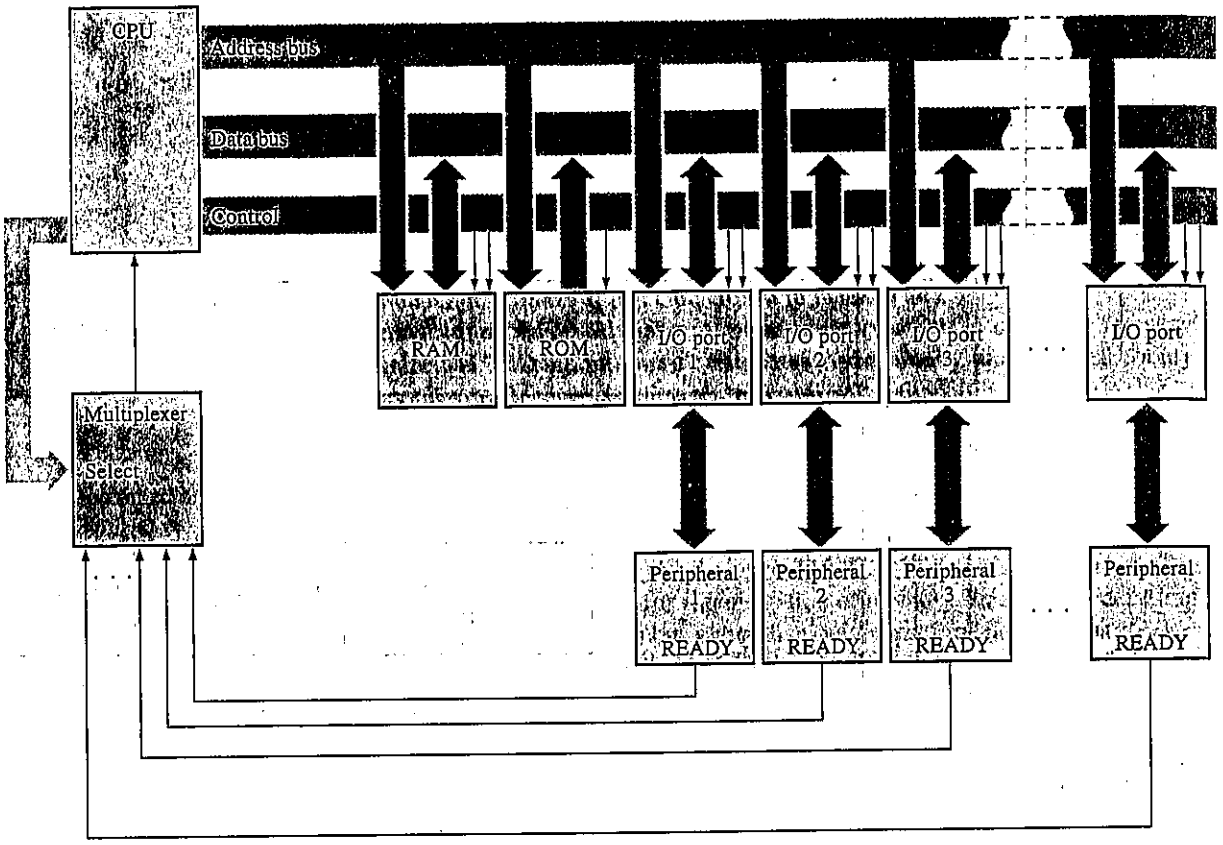


FIGURE 14–35
The basic polled I/O configuration.

regular and predictable intervals and only in situations in which there are no priority considerations.

Interrupt-Driven I/O

This approach overcomes the disadvantages of the polling method. In the interrupt-driven method, the CPU responds to a need for service only when service is requested by a peripheral device. Thus, the CPU can concentrate on running the current program without having to break away unnecessarily to see if a device needs service.

When the CPU receives an I/O interrupt signal, it temporarily stops its current program, acknowledges the interrupt, and fetches a special program (service routine) from memory for the particular device that has issued the interrupt. When the service routine is complete, the CPU returns to where it left off.

A device called a programmable interrupt controller (**PIC**) handles the interrupts on a priority basis. It accepts service requests from the peripherals. If two or more devices request service at the same time, the one assigned the highest priority is serviced first, then one with the next highest priority, and so on. After issuing an interrupt (*INTR*) signal to the CPU, the PIC provides the CPU with information that "points" the CPU to the beginning memory address of the appropriate service routine. This process is called *vectoring*. Figure 14–36 shows a basic interrupt-driven I/O configuration.

Software Interrupts

Another type of interrupt is called a **software interrupt.** Software interrupts are program instructions that can invoke the same service routines described previously. The difference is they are invoked from software rather than from external hardware. When invoked, the

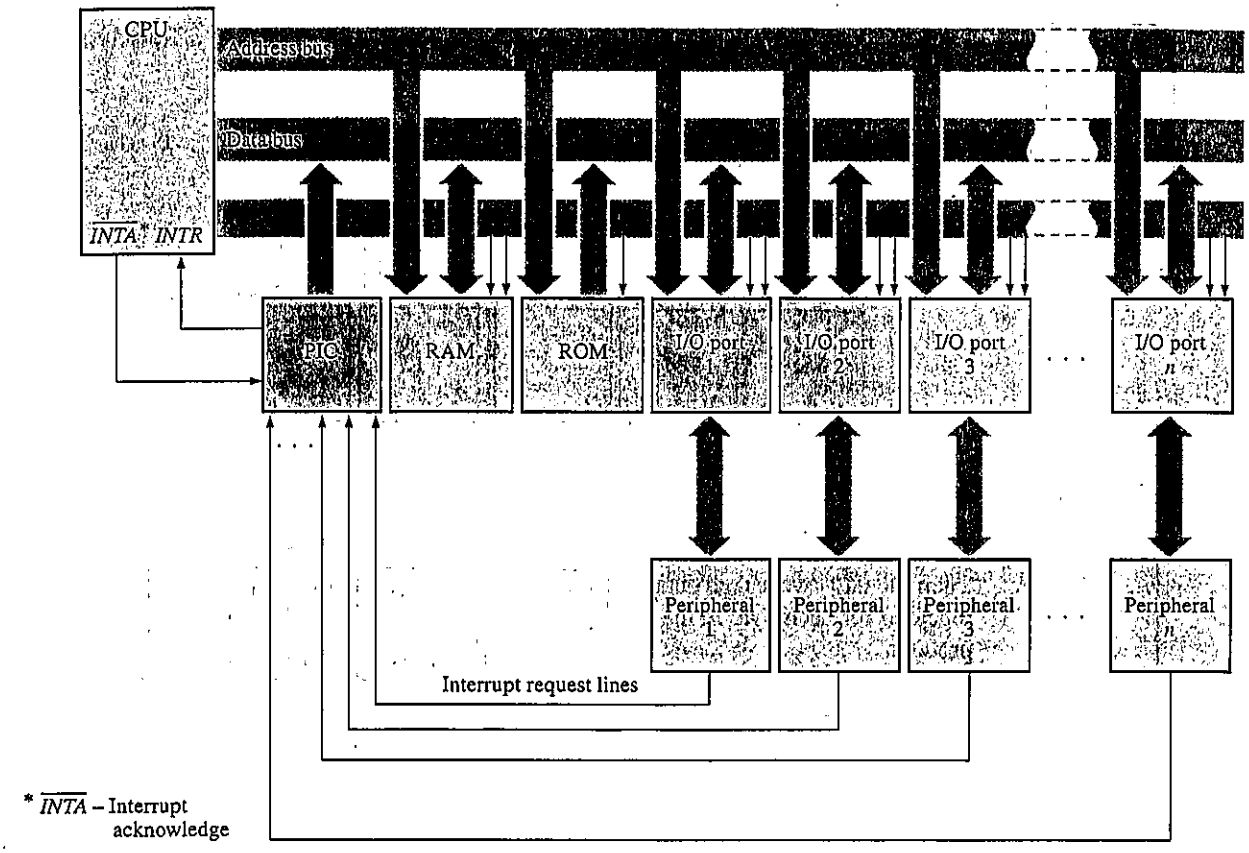


FIGURE 14–36
A basic interrupt-driven I/O configuration.

interrupt service routine executes exactly as if a hardware interrupt had occurred. The first five interrupts are defined by Intel. Others are defined by the BIOS and by DOS to perform many of the I/O operations, such as reading and writing data to the disk, writing data to the display, and reading data from the keyboard.

SECTION 14–8 REVIEW

1. How does an interrupt-driven I/O differ from a polled I/O?
2. What is the main advantage of an interrupt-driven I/O?
3. What is a software interrupt?

14–9 ■ DIRECT MEMORY ACCESS (DMA)

In this short section, we define the technique of data transfer called direct memory access (DMA). A comparison of a CPU-handled transfer and a DMA transfer is presented.

After completing this section, you should be able to

- Define the term *DMA*
- Compare a memory I/O data transfer handled by the CPU to a DMA transfer

All I/O data transfers discussed so far have passed through the CPU. For example, when data are to be transferred from RAM to a peripheral device, the CPU reads the first data byte from the memory and loads it into an internal register within the microprocessor.

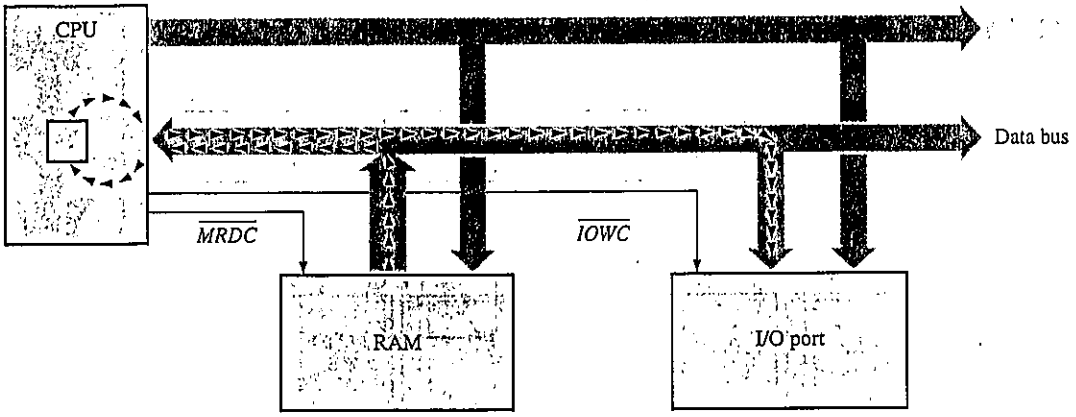


FIGURE 14–37
A memory I/O transfer handled by the CPU.

Then the CPU writes the data byte to the appropriate I/O port. This read/write operation is repeated for each byte in the group of data to be transferred. Figure 14–37 illustrates this process.

For large blocks of data, intermediate stops by the microprocessor consume a lot of time. For this reason, many systems use a technique called direct memory access (**DMA**) to speed up data transfers between RAM and certain peripheral devices. Basically, DMA bypasses the CPU for certain types of data transfers, thus eliminating the time consumed by the normal fetch and execute cycles required for each read or write operation.

For direct memory transfers, a device called the DMA controller takes control of the system buses and allows data to flow directly between RAM and the peripheral device, as indicated in Figure 14–38. Transfers between the disk drive and RAM are particularly suited for DMA because of the large amounts of data involved and the serial nature of the transfers. The DMA controller can handle data transfers several times faster than the CPU.

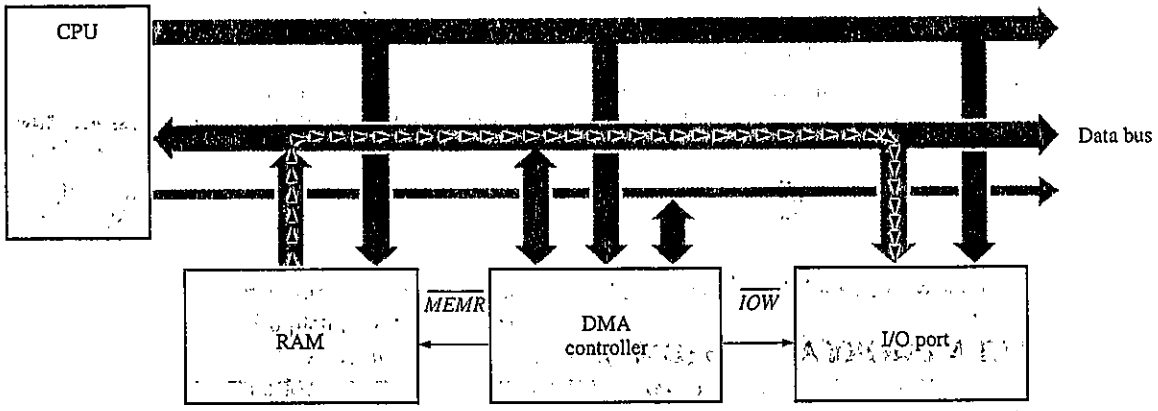


FIGURE 14–38
A DMA transfer.

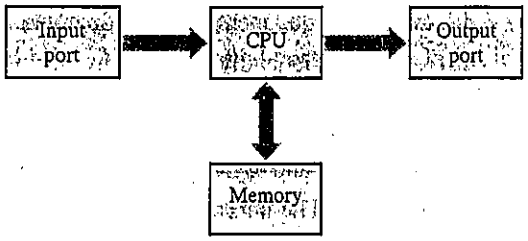
SECTION 14–9 REVIEW

1. What does DMA stand for?
2. Discuss the advantage of DMA, and give an example of a type of transfer for which it is often used.

■ SUMMARY

- Basic units of a computer are shown in Figure 14–39.

FIGURE 14–39



- The three basic computer buses are the *address bus, data bus,* and the *control bus.* The size of any bus is specified by the number of separate conductive paths.
- Typical peripheral devices include the keyboard, external disk drives, mouse, printer, modem, and scanner.
- The number of address lines increased from 20 for the 8086/8088 to 32 for the Pentium family. The data bus was originally 16 for the 8086 and is 64 for the Pentium family.
- General registers for the 8086/8088 are a subset of those in all of the Intel 80X86 and Pentium processors. They include

  Accumulator (AX, which includes AH and AL)
  Base (BX, which includes BH and BL)
  Count (CX, which includes CH and CL)
  Data (DX, which includes DH and DL)
  Stack pointer (SP)
  Base pointer (BP)
  Source index (SI)
  Destination index (DI)

  Beginning with the 80386, this basic set was expanded to the extended register set.
- The basic segment registers for the 8086/8088 are a subset of those in all of the Intel 80X86 and Pentium processors. The segment registers are

  Code segment (CS)
  Data segment (DS)
  Stack segment (SS)
  Extra segment (ES)

  Beginning with the 80386, two new segment registers were added.
- The flag registers for the 8086/8088 are a subset of those in all of the Intel processors. They include

  Trap (TF)
  Direction (DF)
  Interrupt enable (IF)
  Overflow (OF)
  Carry (CF)
  Sign (SF)
  Zero (ZF)
  Auxiliary carry (AF)
  Parity (PF)

- The basic "language" of a computer is called machine code in which instructions are given as a series of binary codes.
- In assembly language, machine instructions are replaced with a short alphabetic English mnemonic that has a one-to-one correspondence to machine code. Assembly language also uses directives to allow the programmer to specify other parameters that are not translated directly into machine code.
- Ports are an interface to external devices. They can be set up as input, output, or a combination of both. They can be accessed as dedicated or as memory-mapped and can be serviced by polling, interrupt-driven, or software.

■ KEY TERMS

These terms are also in the end-of-book glossary.

Accumulator A general-purpose register used for arithmetic and logic operations.

Address A unique memory location containing one byte.

Address bus Generally, a one-way group of conductors from the microprocessor to memory, containing the address information.

ALU Arithmetic logic unit; the key processing element of a microprocessor that performs arithmetic and logic operations.

Architecture The internal functional arrangement of the elements that give a device its particular operating characteristics.

Assembler A program that converts English-like mnemonics into machine code.

Assembler directive An instruction to the assembler; also called a pseudo-op.

Assembly language A form of computer program language in which statements expressed in mnemonics represent the instructions.

BIOS Basic input/output system; a set of programs in ROM that interfaces the I/O devices in a computer system.

BIU Bus interface unit; the portion of the CPU that interfaces with the system buses and fetches instructions, reads operands, and writes results.

Compiler A program that translates high-level program statements, such as BASIC, Pascal, or Fortran, into machine language.

Contiguous Joined together.

Control bus A one-way set of conductors that connects the CPU to other parts of the computer to coordinate its operations and to communicate with external devices.

Control unit The portion within the microprocessor that provides the timing and control signals for getting data into and out of the microprocessor and for synchronizing the execution of instructions.

Coprocessor A microprocessor designed with a limited instruction set optimized to perform arithmetic operations very quickly; it generally works in conjunction with a general-purpose microprocessor.

CPU Central processing unit (also called the MPU); the main part of a computer responsible for control and processing of data.

Data bus A bidirectional set of conductive paths on which data or instruction codes are transferred into the microprocessor or on which the result of an operation or computation is sent out from the microprocessor.

Debug A code within DOS that allows various operations on files. It includes a primitive assembler.

DMA Direct memory access; a method to directly interface a peripheral device to memory without using the CPU for control.

EU Execution unit; the portion of a CPU that executes instructions; it contains the arithmetic logic unit (ALU), the general registers, and the flags.

Flag A bit that indicates the result of an arithmetic or logic operation or is used to alter an operation.

Hardware The circuitry and physical components of a computer system (as opposed to the directions called software).

Instruction pairing The process of combining certain independent instructions so that they can be executed simultaneously by two separate execution units.

Interrupt A signal or instruction that causes the current process to be temporarily stopped while a service routine is run.

I/O port Input/output port; the interface between an internal bus and a peripheral.

IP Instruction pointer; a special register within the CPU that holds the offset address of the next instruction to be executed.

Machine code The basic binary instructions understood by the processor.

Memory-mapped I/O A method of addressing ports by assigning addresses within the computer memory address space to the port. The CPU views the I/O ports as memory locations.

Microcontroller A specialized microprocessor designed for control functions.

Microprocessor A digital integrated circuit device that can be programmed with a series of instructions to perform specified functions on data.

Mnemonic An English-like instruction that is converted by an assembler into a machine code for use by a processor.

Multitasking An operating system environment in which the computer seems to run multiple programs or tasks simultaneously.

Op-code The code for an instruction; a mnenomic.

Operand A variable, a register, a memory location, or a value used in an assembly language program as part of the instruction.

PIC Programmable interrupt controller; handles the interrupts on a priority basis.

Pointer The contents of a register (or registers) that contain an address.

Polling The process of checking a series of peripheral devices to determine if any require service from the CPU.

Port A physical interface on a computer through which data are passed to or from peripherals.

PPI Programmable peripheral interface; a special IC used to implement I/O ports.

Prefetching The process of executing instructions at the same time as other instructions are "fetched," eliminating idle time; also called *pipelining*.

Program A group of instructions designed to solve a specific problem.

Pseudo-operation An instruction to the assembler (as opposed to a processor).

Queue A high-speed memory that stores instructions or data.

Real mode Operation of an Intel processor in a manner to emulate the 8086's 1 Mbyte of memory.

Register array A set of temporary storage locations within the microprocessor for keeping data and addresses that need to be accessed quickly by the program.

Relocatable code A program that can be moved anywhere within the memory space without changing the basic code.

Segment A 64k block of memory.

Software Programs and instructions.

Software interrupt An instruction that invokes an interrupt service routine.

String A contiguous sequence of bytes or words.

Subroutine A series of instructions that can be assembled together and used repeatedly by a program but programmed only once.

Throughput The average speed with which a program is executed.

■ SELF-TEST

1. A basic computer does not include
   (a) an arithmetic logic unit (b) a control unit
   (c) peripheral units (d) a memory unit

2. A 20-bit address bus supports
   (a) 100,000 memory addresses (b) 1,048,576 memory addresses
   (c) 2,097,152 memory addresses (d) 20,000 memory addresses
3. The size of the data bus on the Pentium processors is
   (a) 16 (b) 24 (c) 32 (d) 64
4. A bus that is used to transfer information both to and from the microprocessor is the
   (a) address bus (b) data bus
   (c) both of the above (d) none of the above
5. An example of a peripheral unit is
   (a) the address register (b) the MPU
   (c) the video monitor (d) the interface adapter
6. Two types of memory transfers handled by the CPU are
   (a) direct and interrupt (b) read and write
   (c) bussed and multiplexed (d) input and output
7. Two types of I/O ports are
   (a) dedicated and memory-mapped (b) DMA and PPI
   (c) tristate and octal (d) read and write
8. In the Intel 80X86 family, the maximum number of 8-bit I/O devices is
   (a) 64 (b) 1000 (c) 64,000 (d) 1 million (e) unlimited
9. Polling is a method used for
   (a) determining the state of the microprocessor
   (b) establishing communication between the CPU and a peripheral
   (c) establishing a priority for communication with several peripherals
   (d) determining the next instruction
10. Of the following, which is a 8-bit register?
    (a) AH (b) BX (c) SS (d) IP
11. Essentially, a mnemonic is a(n)
    (a) flowchart (b) operand
    (c) machine code (d) instruction
12. DMA stands for
    (a) digital microprocessor address (b) direct memory access
    (c) data multiplexed access (d) direct memory addressing
13. A computer program is a list of
    (a) memory addresses that contain data to be used in an operation
    (b) addresses that contain instructions to be used in an operation
    (c) instructions arranged to achieve a specific result
14. A type of assembly language that alters the course of the program is called a
    (a) loop (b) jump
    (c) both of the above (d) none of the above
15. A type of interrupt that is invoked from within a program is called a
    (a) software interrupt (b) polled interrupt
    (c) direct interrupt (d) I/O interrupt

■ QUESTIONS

SECTION 14–1 The Microprocessor and the Computer

1. Name the three basic elements of a microprocessor.
2. (a) What is a coprocessor?
   (b) Where is the coprocessor for the Pentium?
3. Name the three basic buses in a computer and explain what each is used for.
4. What is a port?

SECTION 14–2 Microprocessor Families

5. (a) Name the first Intel microprocessor that used a 32-bit address bus.
   (b) How many locations could it address?
6. How many data lines are used in the Pentium?
7. Name the first Motorola microprocessor that used a 32-bit address bus.
8. What was the major difference between the Motorola MPC601 and the 68000 series of processors?

SECTION 14–3 The 8086/8088 Microprocessor and Software Model for the Pentium Processor

9. What are the three basic steps a processor repeatedly cycles through?
10. What is meant by "pipelining"?
11. Name the six segment registers of the 80386 and above.
12. Assume the code segment register contains the hex number 0F05 and the instruction pointer register contains the number 0100. What is the physical address of the next instruction to be executed?
13. Explain the difference between the AH, the AL, the AX, and the EAX registers.
14. (a) What is a flag?
    (b) What two purposes are flags used for?
15. (a) Explain the advantage of instruction pairing in the Pentium processor.
    (b) Why can't the 80486 use instruction pairing?

SECTION 14–4 Microprocessor Programming

16. What is an assembler?
17. Draw a flowchart for a program that adds the numbers from one to 10 and saves the result in a memory location named TOTAL.
18. Draw a flowchart showing how you could count the number of bytes in a string and place the count in a location in memory called COUNT. Assume the string starts at a location named START and has a 20H (ASCII for a space) to signal the end. You should not count the space character.
19. What does the assembler directive **word ptr** do?
20. Explain what happens when the instruction **mov ax,[bx]** is executed.
21. Describe the seven basic types of instructions for the Intel processors.

SECTION 14–5 The Central Processing Unit (CPU)

22. What is the difference between the $\overline{IORC}$ and $\overline{MRDC}$ signals?
23. Which address lines are used for I/O read and write operations in the 8086/8088?
24. What is the purpose of the *ALE* signal from the bus controller of the 8086/8088.
25. What is the function of the Status bits from the 8086/8088 CPU?
26. Name at least three enhancements to the Pentium processor that were not present in the original 8086/8088.

SECTION 14–6 The Memory

27. What locations in memory are reserved for the BIOS work area?
28. What is meant by the term *extended memory?*
29. When the bus controller sends the status code $\overline{S}_2\overline{S}_1\overline{S}_0 = 110$, what type of operation is indicated?

SECTION 14–7 The Input/Output (I/O) Port

30. Explain the difference between dedicated and memory-mapped I/O ports. Which type cannot use the IN and OUT instruction?
31. What is the maximum number of dedicated ports that can be connected to a Pentium processor?
32. Explain the function of an 8255A integrated circuit.

SECTION 14–8 Interrupts

33. Compare polled I/O to interrupt-driven I/O.
34. What is meant by the term *vectoring?*
35. What is meant by a software interrupt?

SECTION 14–9 Direct Memory Access (DMA)

36. Explain what happens in a DMA operation.

■ ANSWERS TO SECTION REVIEWS

SECTION 14–1

1. A computer includes a central processing unit (CPU), memory, and input/output ports.
2. The three buses are (1) the data bus which transfers data to and from the CPU, (2) the address bus which carries either a memory or port address, and (3) the control bus which carries timing and control signals.
3. Data and instructions
4. Assembly language is a low-level language that can be directly translated into binary instructions or data patterns. Two types of operations are instructions and pseudo-operations (directives).

SECTION 14–2

1. An 32-bit microprocessor has an 32-bit data bus.
2. The 8086 was the first 16-bit Intel microprocessor.
3. RISC means reduced instruction set computer.
4. Cache memory stores recently used instructions or data to speed up operations.
5. Superscalar refers to a type of architecture that allows more than one instruction to be executed during one clock cycle.

SECTION 14–3

1. The general-purpose registers in the 8086/8088 are

Accumulator (AX: AH, AL)	Stack pointer (SP)
Base index (BX: BH, BL)	Base pointer (BP)
Count (CX: CH, CL)	Destination index (DI)
Data (DX: DH, DL)	Source index (SI)

2. The BIU provides addressing and data interface.
3. No, the EU does not interface with the buses.
4. The instruction queue stores prefetched instructions for the EU to increase throughput.
5. Codes can be easily relocated within memory.
6. Instruction pairing is the process of combining independent instructions so that they can be executed simultaneously by the two execution units in the Pentium.

SECTION 14–4

1. A program is a list of computer instructions arranged to achieve a specific result.
2. A complex instruction set has a full repertoire of instructions available, giving the programmer flexibility.
3. An op-code is the code for an instruction.
4. An operand is the quantity or element operated on by an instruction.
5. A string is a contiguous sequence of bytes or words.

SECTION 14–5

1. The clock generator provides basic timing signals for the microcomputer.
2. The bus controller issues memory read and write commands, issues I/O read and write commands, and enables the address latches and the data bus transceiver.

3. The twenty bus lines available must be used for both the 20-bit address bus and the 8-bit data bus in the 8088.
4. Pipelining which enabled some instructions to execute in a single clock cycle, addition and expansion of cache memory, inclusion of the internal math coprocessor, internal dual integer processors and branch prediction logic

SECTION 14–6

1. The two CPU cycles are write and read.
2. Four clock cycles
3. A valid address appears on the bus after the leading edge of the *ALE* pulse.

SECTION 14–7

1. An I/O port is the interface between the internal data bus and a peripheral.
2. A dedicated I/O port has an address within the I/O address space; a memory-mapped I/O port has an address within the memory address space.
3. A PPI provides I/O ports.

SECTION 14–8

1. For an interrupt-driven I/O, the CPU provides service to a peripheral only when requested to do so by the peripheral; for a polled I/O, the CPU periodically checks a peripheral to see if it needs service.
2. Interrupt-driven I/Os save CPU time.
3. A software interrupt is an instruction that invokes an interrupt service routine.

SECTION 14–9

1. DMA is direct memory access.
2. A DMA transfer of data from memory to I/O or vice versa saves CPU time. Direct memory access is often used in transferring data between RAM and a disk drive.

■ ANSWERS TO RELATED PROBLEMS FOR EXAMPLES

14–1 $6C4C2_{16}$

14–2 Change first block (initialization block) to "BIG = FFFF"; this is the largest possible unsigned number. Change first question to "Is number < BIG?"

■ ANSWERS TO SELF-TEST

1. (c)	2. (b)	3. (d)	4. (b)	5. (c)	6. (b)	7. (a)	8. (c)
9. (b)	10. (a)	11. (d)	12. (b)	13. (c)	14. (c)	15. (a)	