software languages. Over the same period of time, these higher-performance microprocessors have become more widely used in the design of new electronic equipment and computers. This book presents a detailed study of the software and hardware architectures of Intel Corporation's 8088 and 8086 microprocessors. An introduction to the 80286, 80386, 80486, and Pentium processors is also included.

In this chapter we begin our study with an introduction to microprocessors and microcomputers. The following topics are discussed:

1.1 General Architecture of a Microcomputer System
1.2 Evolution of the Intel Microprocessor Architecture

▲ 1.1 GENERAL ARCHITECTURE OF A MICROCOMPUTER SYSTEM

The hardware of a microcomputer system can be divided into four functional sections: the *input unit, microprocessing unit, memory unit,* and the *output unit*. The block diagram in Fig. 1–1 shows this general microcomputer architecture. Each of these units has a special function in terms of overall system operation. Next we will look at each of these sections in detail.

The heart of a microcomputer is its microprocessing unit (MPU). The MPU of a microcomputer is implemented with a VLSI device known as a *microprocessor*, or just *processor* for short. A microprocessor is a general-purpose processing unit built into a single *integrated circuit* (IC). The microprocessor used in the original IBM PC is Intel Corporation's 8088, shown in Fig. 1–2.

Earlier we indicated that the 8088 is a 16-bit microprocessor. To be more accurate, it is the 8-bit external bus version in Intel's 8086 family of 16-bit microprocessors. Even though the 8088 has an 8-bit external bus, its internal architecture is 16 bits in width and it can directly process 16-bit-wide data. For this reason, the 8088 is considered a 16-bit microprocessor.

The 8088 MPU is the part of the microcomputer that executes instructions of the program and processes data. It is responsible for performing all arithmetic operations and making the logical decisions initiated by the computer's program. In addition to arithmetic and logic functions, the MPU controls overall system operation.

The input and output units are the means by which the MPU communicates with the outside world. Input unit such as the *keyboard* on the IBM PC, allow the user to input information or commands to the MPU; for instance, a programmer could key in the lines of a BASIC program from the keyboard. Many other input devices are available for the PC; two examples include a *mouse,* for implementing a user-friendlier input interface, and a *scanner*, for reading in documents.

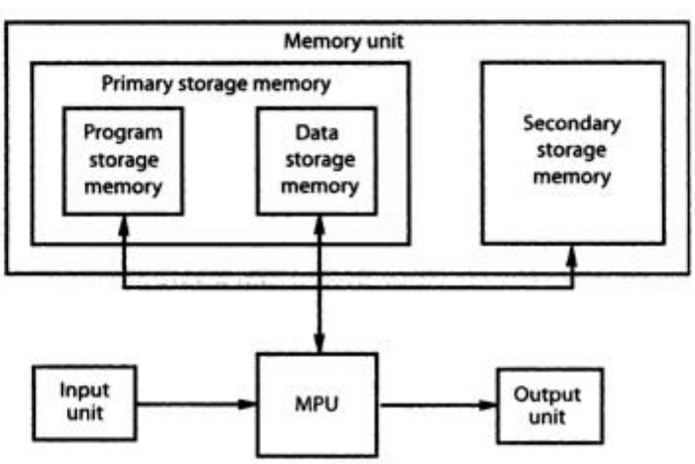

Figure 1–1 General architecture of a microcomputer system.

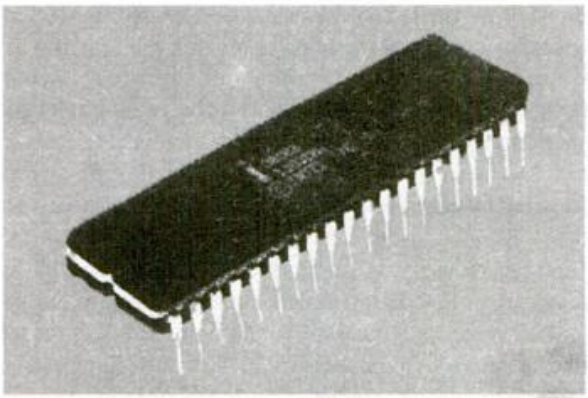

Figure 1–2 Intel Corporation's 8088 microprocessor. (Courtesy of Intel Corp.)

The most widely used output devices of a PC are the *monitor* and the *printer*. The output unit in a microcomputer is used to give the user feedback and to produce documented results. For example, key entries from the keyboard are echoed back to the monitor—that is, by looking at the information displayed on the monitor, the user can confirm that the correct entry was made. Moreover, the results produced by the MPU's processing can be either displayed or printed. For our earlier example of a BASIC program, once it is entered and corrected, a listing of the statements could be printed. Alternate output devices are also available for the microcomputer; for instance, many modern PCs are equipped with an advanced audio processing and speaker system.

The memory unit in a microcomputer is used to *store* information, such as number or character data. By "store" we mean that memory has the ability to hold this information for processing or for outputting at a later time. Programs that define how the computer is to operate and process data also reside in memory.

In the microcomputer system, memory can be divided into two different types: *primary storage memory* and *secondary storage memory*. Secondary storage memory is used for long-term storage of information that is not currently being used. For example, it can hold programs, files of data, and files of information. In the original IBM PC, the *floppy disk drives* represented the secondary storage memory subsystems. It had two $5^1/_4$-inch drives that used double-sided, double-density *floppy-diskette* storage media that could each store up to 360Kbytes (360,000 bytes) of data. This floppy diskette is an example of a removable media—that is, to use the diskette it is inserted into the drive and locked in place. If either the diskette is full or one with a different file or program is needed, the diskette is simply unlocked, removed, and another diskette installed.

The IBM PCXT also employed a second type of secondary storage device called a *hard disk drive*. Modern hard disk drive sizes are 5Gbytes (5000 million bytes), 10Gbytes, 20Gbytes, 40Gbytes, 60Gbytes, and 80Gbytes. The hard disk drive differs from the floppy disk drive in that the media is fixed, which means that the media cannot be removed. However, being fixed is not a problem because the storage capacity of the media is so much larger. Today, desktop PCs are equipped with a hard disk drive in the 20Gbyte to 80Gbyte range.

Both the floppy diskette and hard disk are examples of read/write media—that is, a file of data can be read in from or written out to the storage media in the drive. Another secondary storage device that is very popular in personal computers today is a CD drive. Here a removable *compact disk* (CD) is used as the storage media. This media has very large storage capacity, more than 600Kbytes, but is read-only. This means you cannot write information onto a CD for storage. For this reason, it is normally used to store large programs or files of data that are not to be changed. Recently, a recordable CD (CD-R) has become available, and CD-R drives allow the PC to both read from and write to this media.

Primary storage memory is normally smaller in size and is used for the temporary storage of active information, such as the operating system of the microcomputer, the program that is currently being run, and the data that it is processing. In Fig. 1–1 we see that primary storage memory is further subdivided into *program-storage memory* and *data-storage memory*. The program section of memory is used to store instructions of the operating system and application programs. The data section normally contains data that are to be processed by the programs as they are executed (e.g., text files for a word-processor program or a database for a database-management program). However, programs can also be loaded into data memory for execution.

Typically, primary storage memory is implemented with both *read-only memory* (ROM) and *random-access read/write memory* (RAM) integrated circuits. The original IBM PC had 48Kbytes of ROM and could be configured with 256Kbytes of RAM without adding a memory-expansion board. Modern PC/ATs made with the Pentium IV processors are typically equipped with 128Mbytes of RAM.

Data, whether they represent numbers, characters, or instructions of a program, can be stored in either ROM or RAM. In the original IBM PC, a small part of the operating system and BASIC language are resident to the computer because they are supplied in ROM. By using ROM, this information is made *nonvolatile*—that is, the information is not lost if power is turned off. This type of memory can only be read from; it cannot be written into. On the other hand, data that are to be processed and information that frequently changes must be stored in a type of primary storage memory from which they can be read by the microprocessor, modified through processing, and written back for storage. This requires a type of memory that can be both read from and written into. For this reason, such data are stored in RAM instead of ROM.

Earlier we pointed out that the instructions of a program could also be stored in RAM. In fact, to run the PC *operating system* such as Windows 98®, it must be loaded into the RAM of the microcomputer. Normally the operating system, supplied on CDs, is first read from the CDs and written onto the hard disk. This is called *copying* of the operating system onto the hard disk. Once it is copied, the CD version of Windows 98 may not be used again. The PC is set up so that when it is turned on, Windows 98 is automatically read from the hard disk, written into the RAM, and then run.

RAM is an example of a volatile memory—that is, when power is turned off, the data that it holds are lost. This is why Windows 98 must be reloaded from the hard disk each time the PC is turned on.

▲ 1.2 EVOLUTION OF THE INTEL MICROPROCESSOR ARCHITECTURE

Generally, microprocessors and microcomputers are categorized in terms of the maximum number of binary bits in the data they process—that is, their word length. Over time, five standard data widths have evolved for microprocessors and microcomputers: *4-bit, 8-bit, 16-bit, 32-bit,* and *64-bit.*

Figure 1–3 illustrates the evolution of Intel's microprocessors since their introduction in 1972. The first microprocessor, the 4004, was designed to process data arranged as 4-bit words. This organization is also referred to as a *nibble* of data.

The 4004 implemented a very low performance microcomputer by today's standards. This low performance and limited system capability restricted its use to simpler, special-purpose applications. It was commonly used in electronic calculators.

Beginning in 1974, a second generation of microprocessors was introduced. These devices, the 8008, 8080, and 8085, were 8-bit microprocessors and were designed to process 8-bit (1-byte-wide) data instead of 4-bit data. The 8080, identified in Fig. 1–3, was introduced in 1975.

These newer 8-bit microprocessors were characterized by higher-performance operation, larger system capabilities, and greater ease of programming. They were able to provide the system requirements for many applications that could not be satisfied with the earlier 4-bit microprocessors. These extended

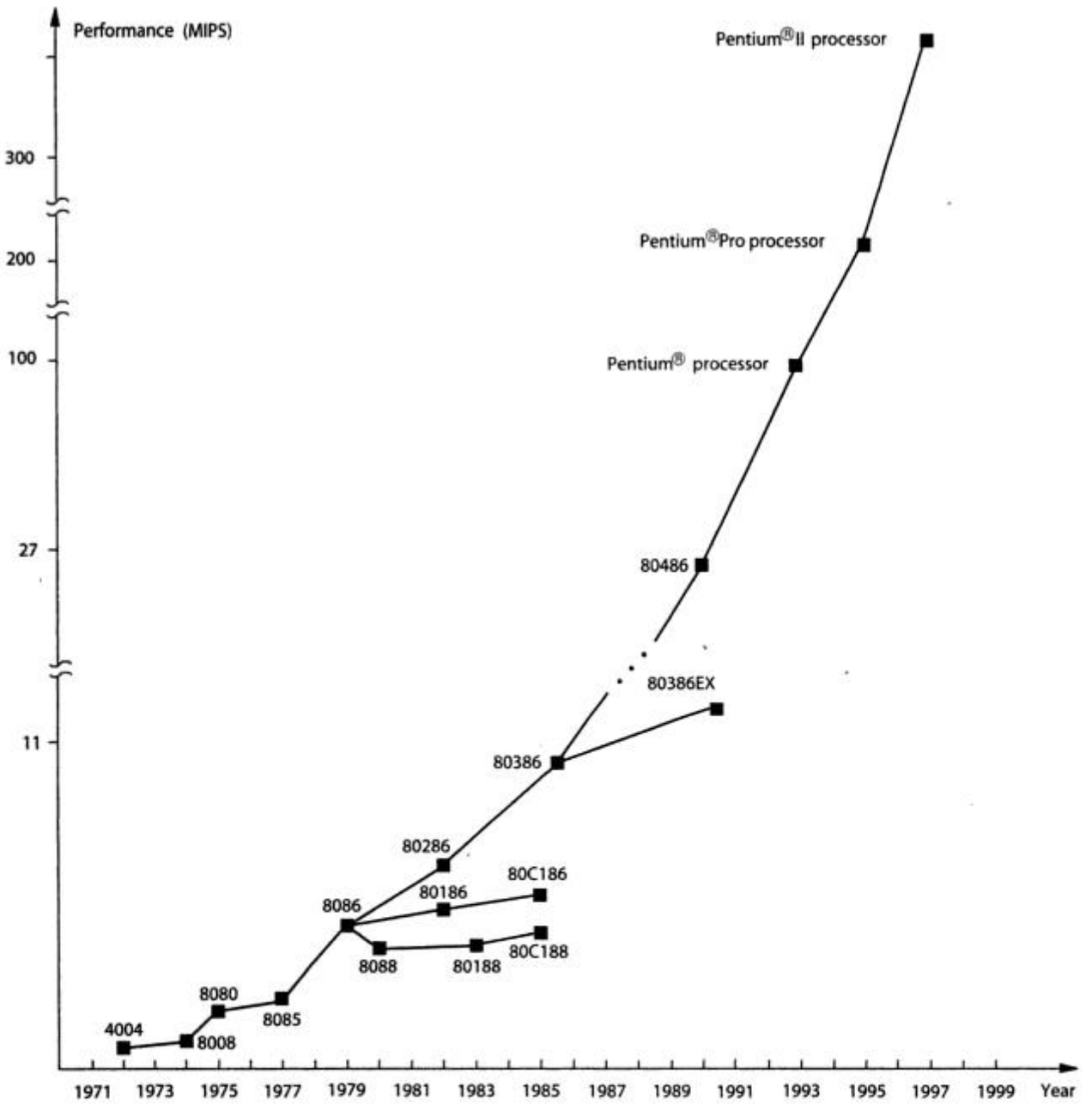

Figure 1–3 Evolution of the Intel microprocessor architecture.

capabilities led to widespread acceptance of 8-bit microcomputers for special-purpose system designs. Examples of these dedicated applications are electronic instruments, cash registers, and printers.

In the mid-1970s, many of the leading semiconductor manufacturers announced plans for development of third-generation 16-bit microprocessors. Looking at Fig. 1–3, we see that Intel's first 16-bit microprocessor, the 8086, became available in 1979 and was followed the next year by its 8-bit bus version, the 8088. This was the birth of Intel's 8086 family architecture. Other family members such as the 80286, 80186, and 80188 were introduced in the years that followed.

These 16-bit microprocessors provided higher performance and had the ability to satisfy a broad scope of special-purpose and general-purpose microcomputer applications. They all have the ability to

handle 8-bit, 16-bit, and special-purpose data types, and their powerful instruction sets are more in line with those provided by a minicomputer.

In 1985, Intel Corporation introduced its first 32-bit microprocessor, the 80386DX, which brought true minicomputer-level performance to the microcomputer system. This device was followed by a 16-bit external bus version, the 80386SX, in 1988. Intel's second generation of 32-bit microprocessors, called the 80486DX and 80486SX, became available in 1989 and 1990, respectively. They were followed by a yet higher-performance family, the Pentium processors, in 1993. Today, its fourth generation member—the Pentium® IV processor, represents this family.

Microprocessor Performance: MIPS and iCOMP

Figure 1–3 illustrates the 8086 microprocessor families relative to their performance. Here performance is measured in what are called *MIPS*—that is, how many million instructions they can execute per second. Today, the number of MIPS provided by a microprocessor is the standard most frequently used to compare performance. Notice that performance has vastly increased with each new generation of microprocessor. For instance, the performance identified for the 80386 corresponds to an 80386DX device operating at 33 MHz and equals approximately 11 MIPS. With the introduction of the 80486, the level of performance capability of the architecture was raised to approximately 27 MIPS. This shows that performance of the 8086 architecture was more than doubled with the introduction of the 33-MHz 80486DX microprocessor.

The MIPS used in this chart are known as Drystone V1.1 MIPS—that is, they are measured by running a test program called the *Drystone program*, and the resulting performance measurements are normalized to those of a VAX 1.1 computer (VAX 1.1 was a minicomputer manufactured by Digital Equipment Corporation). Therefore, we say that the 80486DX is capable of delivering up to 27 VAX MIPS of performance.

Intel Corporation provides another method, the iCOMP index, for comparison of the performance of its 32-bit microprocessors in a personal computer environment. In the iCOMP index chart shown in Fig. 1–4, a bar is used to represent a measure of the performance for each of Intel's MPUs. Instead of being related to the performance of a test program, such as the Dystone program, the iCOMP rating of an MPU is based on a variety of 16-bit and 32-bit MPU performance components important to the personal computer—that is, the iCOMP rating encompasses performance components that represent integer mathematics, floating-point mathematics, graphics, and video. The contribution by each of these categories is also weighted based on an estimate of their normal occurrence in widely used software applications. For this reason, iCOMP is a more broad-based rating of MPU performance for the personal computer applications.

The higher the iCOMP rating, the higher the performance offered by the MPU. Notice that the members of the 80386 family offer low performance when compared to the newer 80486 and Pentium processor families. In fact, the slowest 80386SX MPU shown in Fig. 1–4, the –20, has a performance rating of 32, whereas the fastest 80386DX, the –33, is rated at 68. Therefore, a wide range of system-performance levels can be achieved by selecting among the various members of the 80386, 80486, and Pentium processor families.

Transistor Density

The evolution of microprocessors was made possible by advances in semiconductor process technology. Semiconductor-device geometry decreased from about 5 microns in the early 1970s to

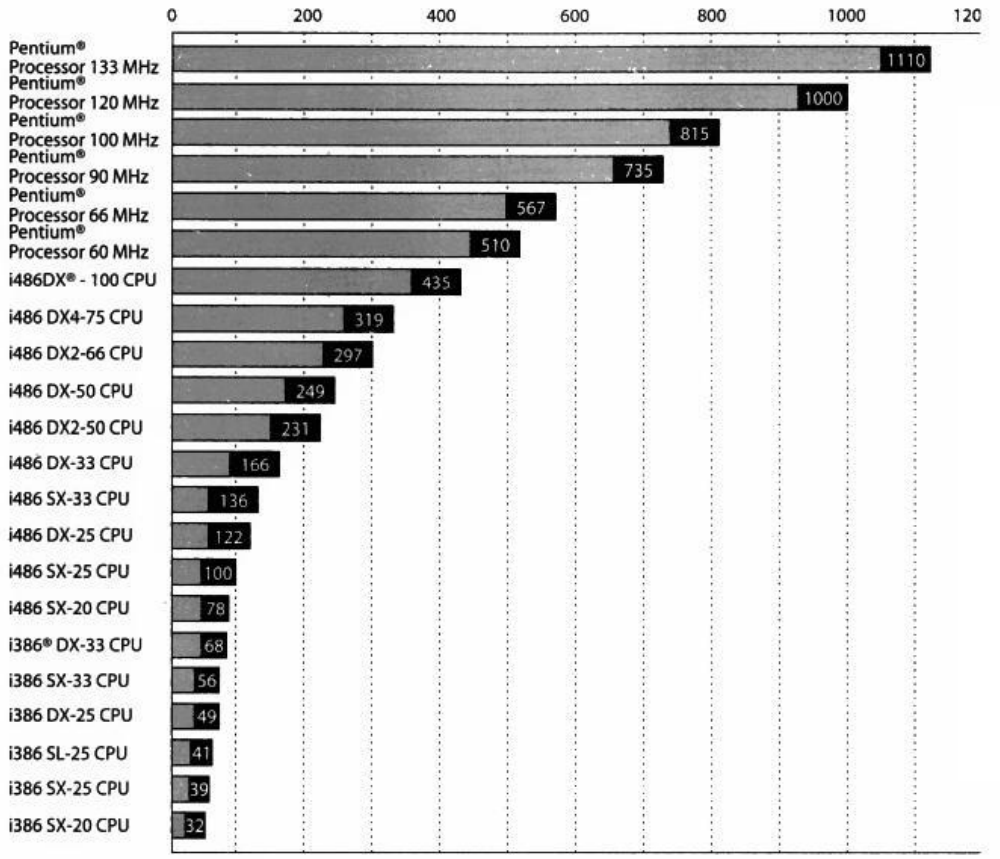

Figure 13–1 iCOMP™ Index rating chart. (Reprinted by permission of Intel Corporation. Copyright/Intel Corp. 1993)

execution unit, the *segment unit,* the *page unit,* the *bus unit,* the *prefetch unit,* and the *decode unit.* Let us now look more closely at each of the processing units of the 80386DX.

The bus unit is the 80386DX's interface to the outside world. By interface, we mean the path by which it connects to external devices. The bus interface provides a 32-bit data bus, a 32-bit address bus, and the signals needed to control transfers over the bus. In fact, 8-bit, 16-bit, and 32-bit data transfers are supported. These buses are demultiplexed like those of the 80286. That is, the 80386DX has separate pins for its address and data bus lines. This demultiplexing of address and data results in higher performance and easier hardware design. Expanding the data bus width to 32 bits further improves the performance of the 80386DX's hardware architecture as compared to that of either the 8086 or 80286.

The bus unit is responsible for performing all external bus operations. This processing unit contains the latches and drivers for the address bus, transceivers for the data bus, and control logic for signaling whether a memory, input/output, or interrupt-acknowledge bus cycle is being performed. Looking at Fig. 13–2, we find that for data accesses, the address of the storage location to be accessed is input from the paging unit, and for code accesses, the prefetch unit provides the address.

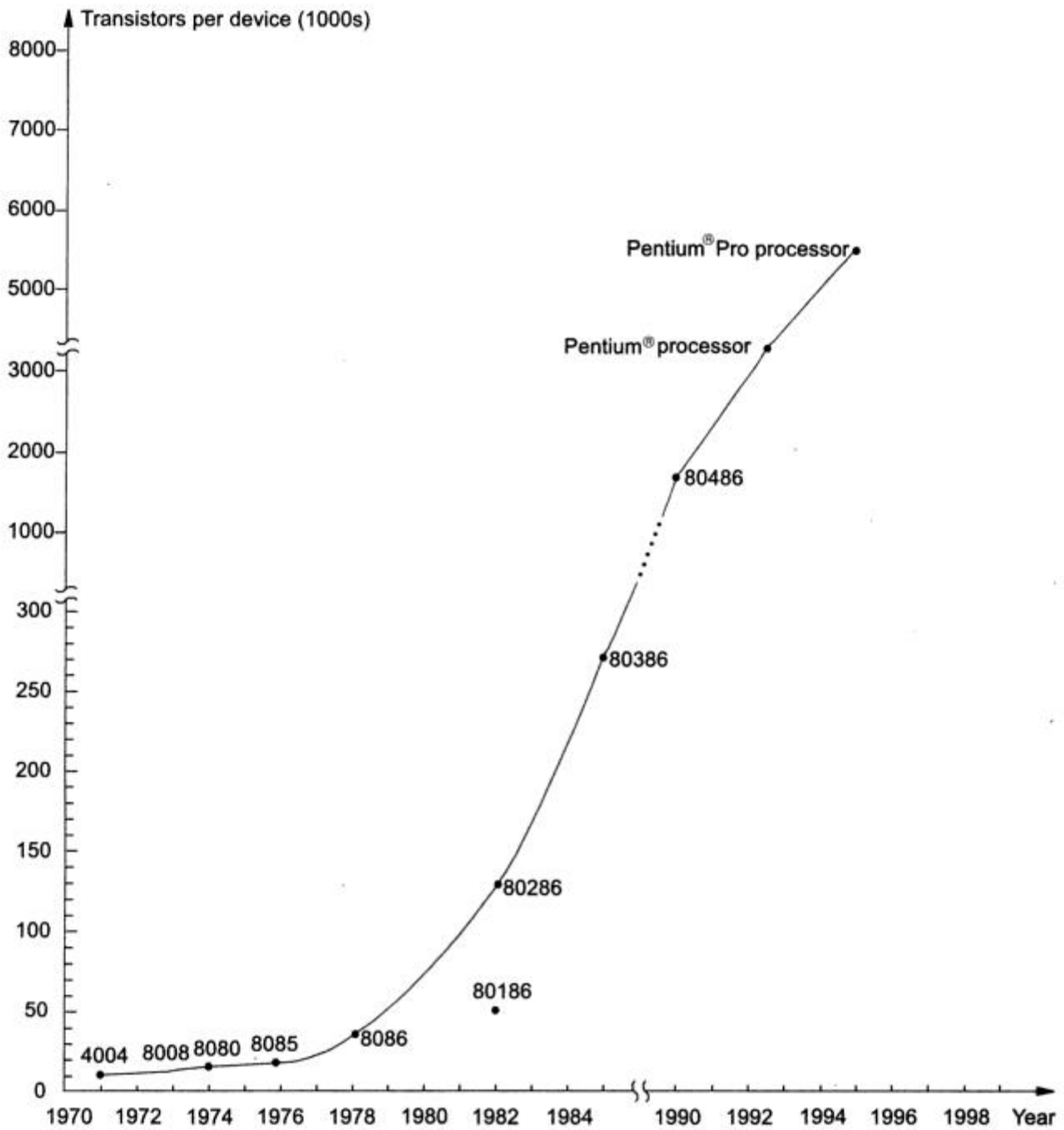

Figure 1–5 Device complexity.

as the 8080 were most widely used as *special-purpose microcomputers*—a system that has been tailored to meet the needs of a specific application. These special-purpose microcomputers were used in *embedded control applications*—applications in which the microcomputer performs a dedicated control function.

Embedded control applications are further divided into those that involve primarily *event control* and those that require *data control*. An example of an embedded control application that is primarily event control is a microcomputer used for industrial process control. Here the program of the microprocessor is used to initiate a timed sequence of events. On the other hand, an application that focuses

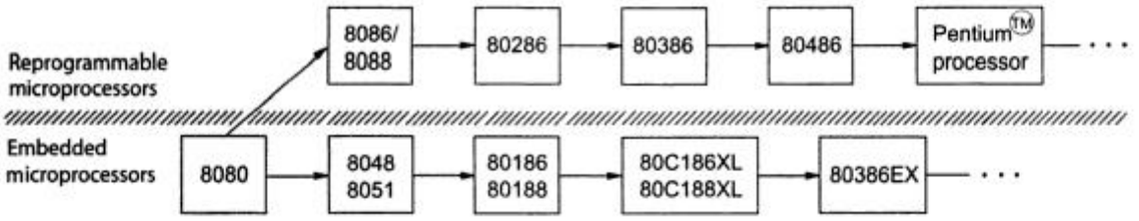

Figure 1–6 Processors for embedded control and reprogrammable applications.

more on data control than event control is a hard disk controller interface. In this case, a block of data that is to be processed—for example, a file of data—must be quickly transferred from secondary storage memory to primary storage memory.

The spectrum of embedded control applications requires a variety of system features and performance levels. Devices developed specifically for the needs of this marketplace have stressed low cost and high integration. In Fig. 1–6, we see that highly integrated 8-bit, single-chip microcomputer devices such as the 8048 and 8051 initially replaced the earlier multichip 8080 solutions. These devices were tailored to work best as event controllers. For instance, the 8051 offers one-order-of-magnitude-higher performance than the 8080, a more powerful instruction set, and special on-chip functions such as ROM, RAM, an interval/event timer, a universal asynchronous receiver/transmitter (UART), and programmable parallel input/output ports. Today, this type of embedded control device is called a *microcontroller*.

Later, devices such as the 80C186XL, 80C188XL, and 80386EX were designed to meet the needs of data-control applications. They are highly integrated and have additional features such as string instructions and direct-memory access channels, which handle the movement of data better. They are known as *embedded microprocessors*.

The category of reprogrammable microprocessors represents the class of applications in which a microprocessor is used to implement a *general-purpose microcomputer*. Unlike a special-purpose microcomputer, a general-purpose microcomputer is intended to run a variety of software applications—that is, while it is in use it can be easily reprogrammed to run a different application. Two examples of reprogrammable microcomputers are the personal computer and file server. Figure 1–6 shows that the

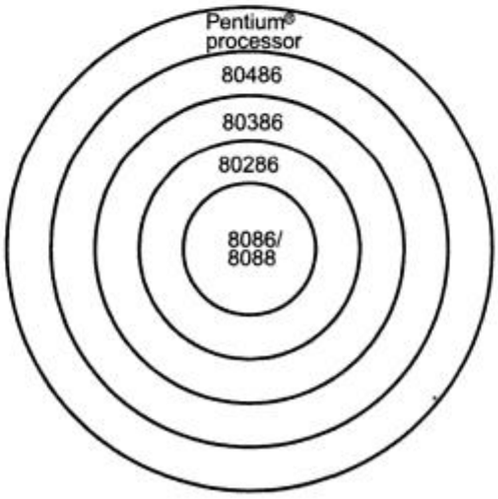

Figure 1–7 Code and system-level compatibility.

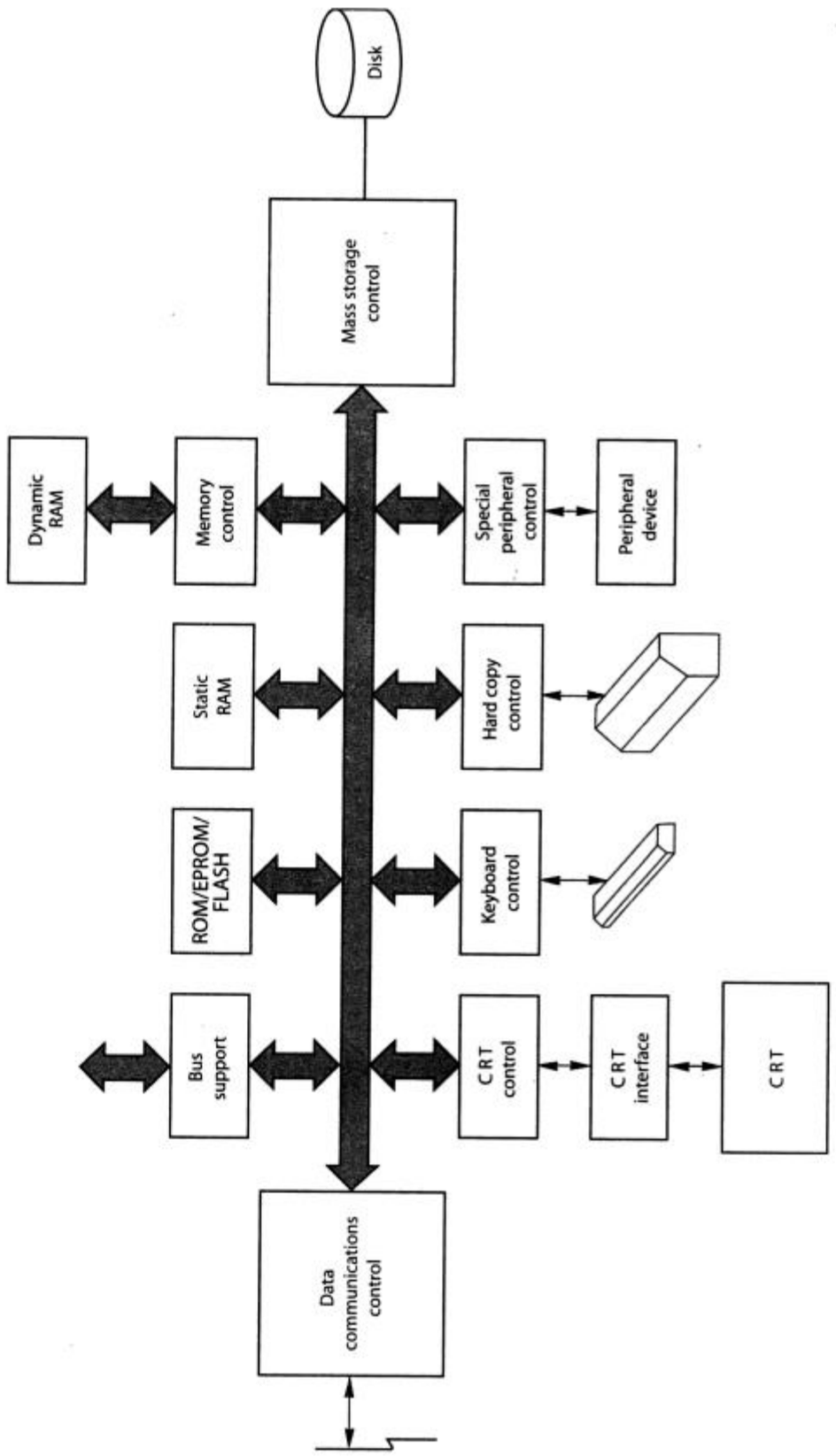

Figure 1–8 Peripheral support for the MPU.

8086, 8088, 80286, 80386, 80486, and Pentium processor are the Intel microprocessors intended for use in this type of application.

Architectural compatibility is a critical need of microprocessors developed for use in reprogrammable applications. As shown in Fig. 1–7, each new member of the 8086/8088 family provides a superset of the earlier device's architecture—that is, the features offered by the 80386 microprocessor are a superset of the 80286 architecture, and those of the 80286 are a superset of the original 8086/8088 architecture.

Actually, the 80286, 80386, 80486, and Pentium processors can operate in either of two modes: the *real-address mode* or *protected-address mode*. When in real mode, they operate like a high-performance 8086/8088. They can execute what is called the *base instruction set*, which is object code compatible with the 8086/8088. For this reason, operating systems and application programs written for the 8086 and 8088 run on the 80286, 80386, 80486, and Pentium processor architectures without modification. Further, a number of new instructions have been added in the instruction sets of the 80286, 80386, 80486, and Pentium processors to enhance their performance and functionality. We say that object code is *upward compatible* within the 8086 architecture. This means that 8086/8088 code will run on the 80286, 80386, 80486, and Pentium processors, but the reverse is not true if any of the new instructions are in use.

Microprocessors designed for implementing general-purpose microcomputers must offer more advanced system features than those of a microcontroller; for example, they need to support and manage a large memory subsystem. The 80286 is capable of managing a 1Gbyte (gigabytes) address space, and the 80386 supports 64Tbytes (64 terabytes) of memory. Moreover, a reprogrammable microcomputer, such as a personal computer, normally runs an operating system. The architectures of the 80286, 80386, 80486, and Pentium processors have been enhanced with on-chip support for operating system functions such as *memory management, protection,* and *multitasking*. These new features become active only when the MPU is operated in the protected mode. The 80386, 80486, and Pentium processors also have a special mode of operation known as *virtual 8086 mode* that permits 8086/8088 code to be run in the protected mode.

Reprogrammable microcomputers, such as those based on the 8086 family, require a variety of input/output resources. Figure 1–8 shows the kinds of interfaces that are typically implemented in a personal computer or a microcomputer system. A large family of VLSI peripheral ICs (examples are floppy disk controllers, hard disk controllers, local area network controllers, and communication controllers) is needed to support a reprogrammable microprocessor such as the 8086, 80286, 80386, 80486, and Pentium processor. For this reason, these processors are designed to implement a multi-chip microcomputer system, which can easily be configured with the appropriate set of input/output interfaces.

REVIEW PROBLEMS

Section 1.1

1. What are the four building blocks of a microcomputer system?
2. What is the heart of the microcomputer system called?
3. Is the 8088 an 8-bit or a 16-bit microprocessor? Explain.
4. What is the primary input unit of the PC? Give two other examples of input units available for the PC.
5. What are the primary output devices of the PC?
6. Into what two sections is the memory of a PC partitioned?
7. What is the storage capacity of the standard $5^1/_4$-inch floppy diskette of the original PC? What is the storage capacity of the standard hard disk drive of the original PCXT?

8. What do ROM and RAM stand for?
9. How much ROM was provided in the original PC's processor board? What was the maximum amount of RAM that could be implemented on this processor board?
10. Why must Windows 98 be reloaded from the hard disk each time power is turned on?

Section 1.2

11. What are the standard data word lengths for which microprocessors have been developed?
12. What was the first 4-bit microprocessor introduced by Intel Corporation? Also name 8-bit,16-bit and 32-bit microprocessors introduced.
13. Name five 16-bit members of the 8086 family architecture.
14. What does MIPS stand for?
15. Approximately how many MIPS are delivered by the 33 MHz 80486DX?
16. What is the name of the program that is used to run the MIPS measurement test for the data in Fig. 1–3?
17. What is the iCOMP rating of an 80386SX-25 MPU? An 80386DX-25 MPU?
18. Approximately how many transistors are used to implement the 8088 microprocessor, the 80286 microprocessor, the 80386DX microprocessor, the 80486DX microprocessor and the Pentium processor?
19. What is an embedded microcontroller?
20. Name the two groups into which embedded processors are categorized based on applications.
21. What is the difference between a multichip microcomputer and a single-chip microcomputer?
22. Name six 8086 family microprocessors intended for use in reprogrammable microcomputer applications.
23. List the two modes of operation for 80386.
24. What is meant by upward software compatibility relative to 8086 architecture microprocessors?
25. List three advanced architectural features provided by the 80386DX microprocessor.
26. Give three types of VLSI peripheral support devices needed in a reprogrammable microcomputer system.

Software Architecture of the 8088 and 8086 Microprocessors

▲ INTRODUCTION

In this chapter we shall study in detail the 8088 and 8086 microprocessors and their assembly language programming. To program either the 8088 or 8086 using assembly language, we must understand how the microprocessor and its memory and input/output subsystems operate from a software point of view. For this reason, in this chapter, we will examine the *software architecture* of the 8088 and 8086 microprocessors. The description that follows frequently refers only to the 8088 microprocessor, but everything that is described for the 8088 also applies to the 8086. This is because the software architecture of the 8086 is identical to that of the 8088. The following topics are covered here.

2.1 Microarchitecture of the 8088/8086 Microprocessor
2.2 Software Model of the 8088/8086 Microprocessor
2.3 Memory Address Space and Data Organization
2.4 Data Types
2.5 Segment Registers and Memory Segmentation
2.6 Dedicated, Reserved, and General-Use Memory
2.7 Instruction Pointer
2.8 Data Registers
2.9 Pointer and Index Registers
2.10 Status Register

2.11 Generating a Memory Address
2.12 The Stack
2.13 Input/Output Address Space

▲ 2.1 MICROARCHITECTURE OF THE 8088/8086 MICROPROCESSOR

The microarchitecture of a processor is its internal architecture—that is, the circuit building blocks that implement the software and hardware architectures of the 8088/8086 microprocessors. Due to the requirement of additional features and higher performance, the microarchitecture of a microprocessor family evolves over time. In fact, a new microarchitecture is introduced for Intel's 8086 family every few years. Each new generation of processors (the 8088/8086, 80286, 80386, 80846, and Pentium processors) represents significant changes in the microarchitecture of the 8086.

The microarchitectures of the 8088 and 8086 microprocessors are similar. They both employ *parallel processing*—that is, they are implemented with several simultaneously operating processing units. Figure 2–1(a) illustrates the internal architecture of the 8088 and 8086 microprocessors. They contain two processing units: the *bus interface unit* (BIU) and the *execution unit* (EU). Each unit has dedicated functions and both operate at the same time. In essence, this parallel processing effectively makes the fetch and execution of instructions independent operations. This results in efficient use of the system bus and higher performance for 8088/8086 microcomputer systems.

The bus interface unit is the 8088/8086's connection to the outside world. By interface, we mean the path by which it connects to external devices. The BIU is responsible for performing all external bus operations, such as instruction fetching, reading and writing of data operands for memory, and inputting or outputting data for input/output peripherals. These information transfers take place over the system bus. This bus includes an 8-bit bidirectional data bus for the 8088 (16 bits for the 8086), a 20-bit address bus, and the signals needed to control transfers over the bus. The BIU is not only responsible for performing bus operations, it also performs other functions related to instruction and data acquisition. For instance, it is responsible for instruction queuing and address generation.

To implement these functions, the BIU contains the segment registers, the instruction pointer, the address generation adder, bus control logic, and an instruction queue. Figure 2–1(b) shows the bus interface unit of the 8088/8086 in detail. The BIU uses a mechanism known as an *instruction queue* to implement a pipelined architecture. This queue permits the 8088 to prefetch up to 4 bytes (6 bytes for the 8086) of instruction code. Whenever the queue is not full—that is, it has room for atleast 2 more bytes, and, at the same time, the execution unit is not asking it to read or write data from memory—the BIU is free to look ahead in the program by prefetching the next sequential instructions. Prefetched instructions are held in the first-in first-out (FIFO) queue. Whenever a byte is loaded at the input end of the queue, it is automatically shifted up through the FIFO to the empty location nearest the output. Here the code is held until the execution unit is ready to accept it. Since instructions are normally waiting in the queue, the time needed to fetch many instructions of the microcomputer's program is eliminated. If the queue is full and the EU is not requesting access to data in memory, the BIU does not need to perform any bus operations. These intervals of no bus activity, which occur between bus operations are known as *idle states.*

The execution unit is responsible for decoding and executing instructions. Notice in Fig. 2–1(b) that it consists of the *arithmetic logic unit* (ALU), status and control flags, general-purpose registers, and temporary-operand registers. The EU accesses instructions from the output end of the instruction queue and data from the general-purpose registers or memory. It reads one instruction byte after the other from the output of the queue, decodes them, generates data addresses if necessary, passes them to the BIU and requests it to perform the read or write operations to memory or I/O, and performs the operation

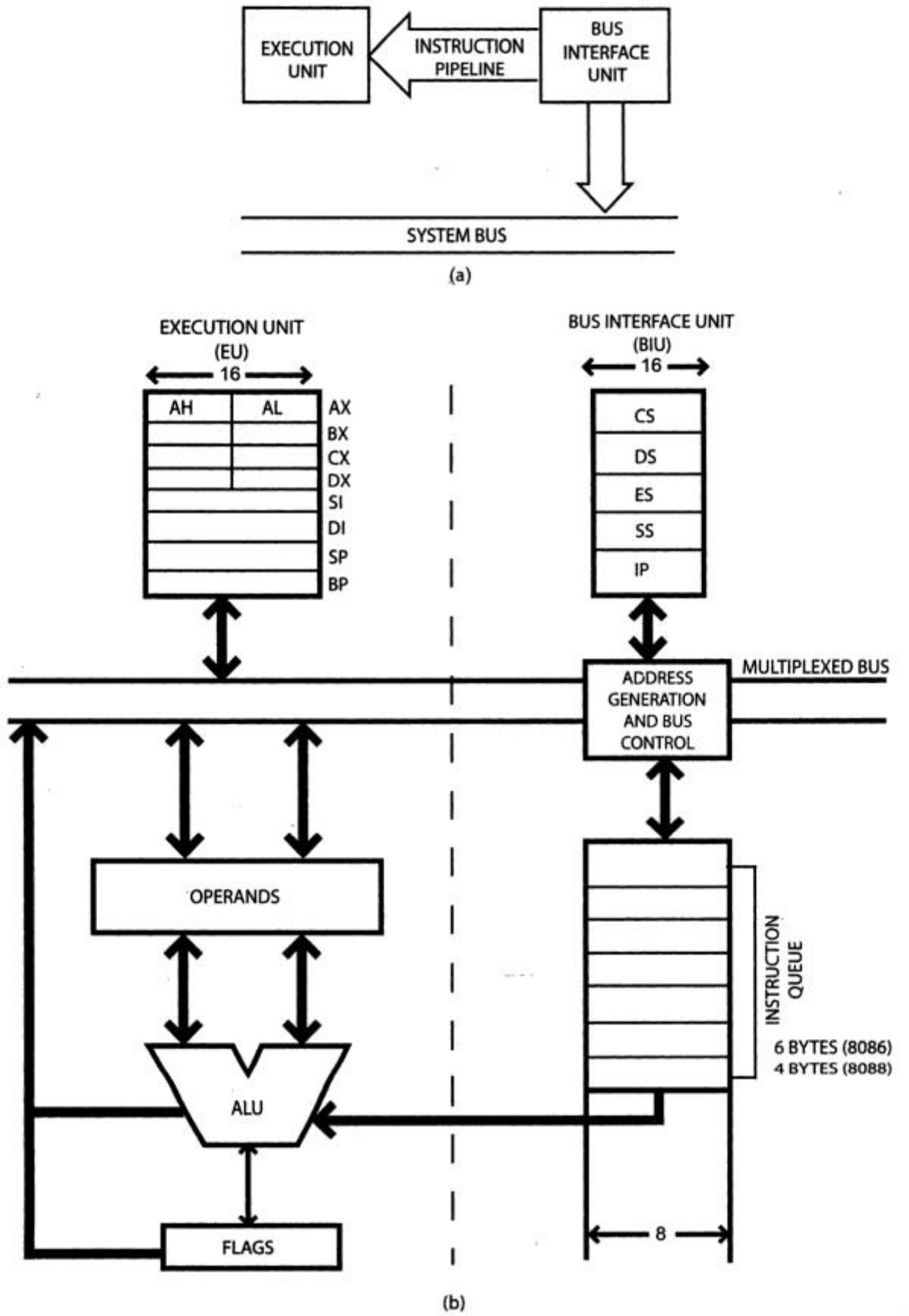

Figure 2–1 (a) Pipelined architecture of the 8088/8086 microprocessors. (Reprinted with permission of Intel Corporation, Copyright/Intel Corp. 1981) (b) Execution and bus interface units. (Reprinted with permission of Intel Corp., Copyright/Intel Corp. 1981)

specified by the instruction. The ALU performs the arithmetic, logic, and shift operations required by an instruction. During execution of the instruction, the EU may test the status and control flags, and updates these flags based on the results of executing the instruction. If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to the top of the queue.

▲ 2.2 SOFTWARE MODEL OF THE 8088/8086 MICROPROCESSOR

The purpose of developing a *software model* is to aid the programmer in understanding the operation of the microcomputer system from a software point of view. To be able to program a microprocessor, one does not need to know all of its hardware architectural features. For instance, we do not necessarily need to know the function of the signals at its various pins, their electrical connections, or their electrical switching characteristics. Also the function, interconnection, and operation of the internal circuits of the microprocessor need not be considered. What is important to the programmer is to know the various registers within the device and to understand their purpose, functions, operating capabilities, and limitations. Furthermore, it is essential that the programmer know show external memory and input/output peripherals are organized, how information is arranged in registers, memory, and input/output, and how memory and I/O are addressed to obtain instructions and data. This information represents the software architecture of the processor. Unlike the microarchitecture, the software architecture changes only slightly from generation to generation of processor.

The software model in Fig. 2–2 illustrates the software architecture of the 8088 microprocessor. Looking at this diagram, we see that it includes 13 16-bit internal registers: the *instruction pointer* (IP), four *data registers* (AX, BX, CX, and DX), *two pointer registers* (BP and SP), *two index registers* (SI and DI), and four *segment registers* (CS, DS, SS, and ES). In addition, there is another register called the *status register* (SR), with nine of its bits implemented as status and control flags.

Figure 2–2 shows that the 8088 architecture implements independent memory and input/output address spaces. Notice that the memory address space is 1,048,576 bytes (1 Mbyte) in length and the I/O address space is 65,536 bytes (64 Kbytes) in length. Our concern here is what can be done with this software architecture and how to do it through software. For this purpose, we will now begin a detailed study of the elements of the model and their relationship to software.

▲ 2.3 MEMORY ADDRESS SPACE AND DATA ORGANIZATION

Now that we have introduced the idea of a software model, let us look at how information such as numbers, characters, and instructions are stored in memory. As shown in Fig. 2–3, the 8088 microcomputer supports 1 Mbyte of external memory. This memory space is organized from a software point of view as individual bytes of data stored at consecutive addresses over the address range 00000_{16} to $FFFFF_{16}$. Therefore, memory in an 8088-based microcomputer is actually organized as 8-bit bytes, not as 16-bit words.

The 8088 can access any two consecutive bytes as a *word* of data. In this case, the lower-addressed byte is the least significant byte of the word, and the higher-addressed byte is its most significant byte. Figure 2–4(a) shows how a word of data is stored in memory. Notice that the storage location at the lower address, 00724_{16}, contains the value $00000010_2 = 02_{16}$.The contents of the next-higher-addressed storage location, 00725_{16}, are $01010101_2 = 55_{16}$. These two bytes represent the word $0101010100000010_2 = 5502_{16}$.

To permit efficient use of memory, words of data can be stored at what are called even- or odd-addressed word boundaries. The least significant bit of the address determines the type of word boundary. If this bit is 0, the word is at an *even-address boundary*—that is, a word at an even-address boundary corresponds to two consecutive bytes, with the least significant byte located at an even

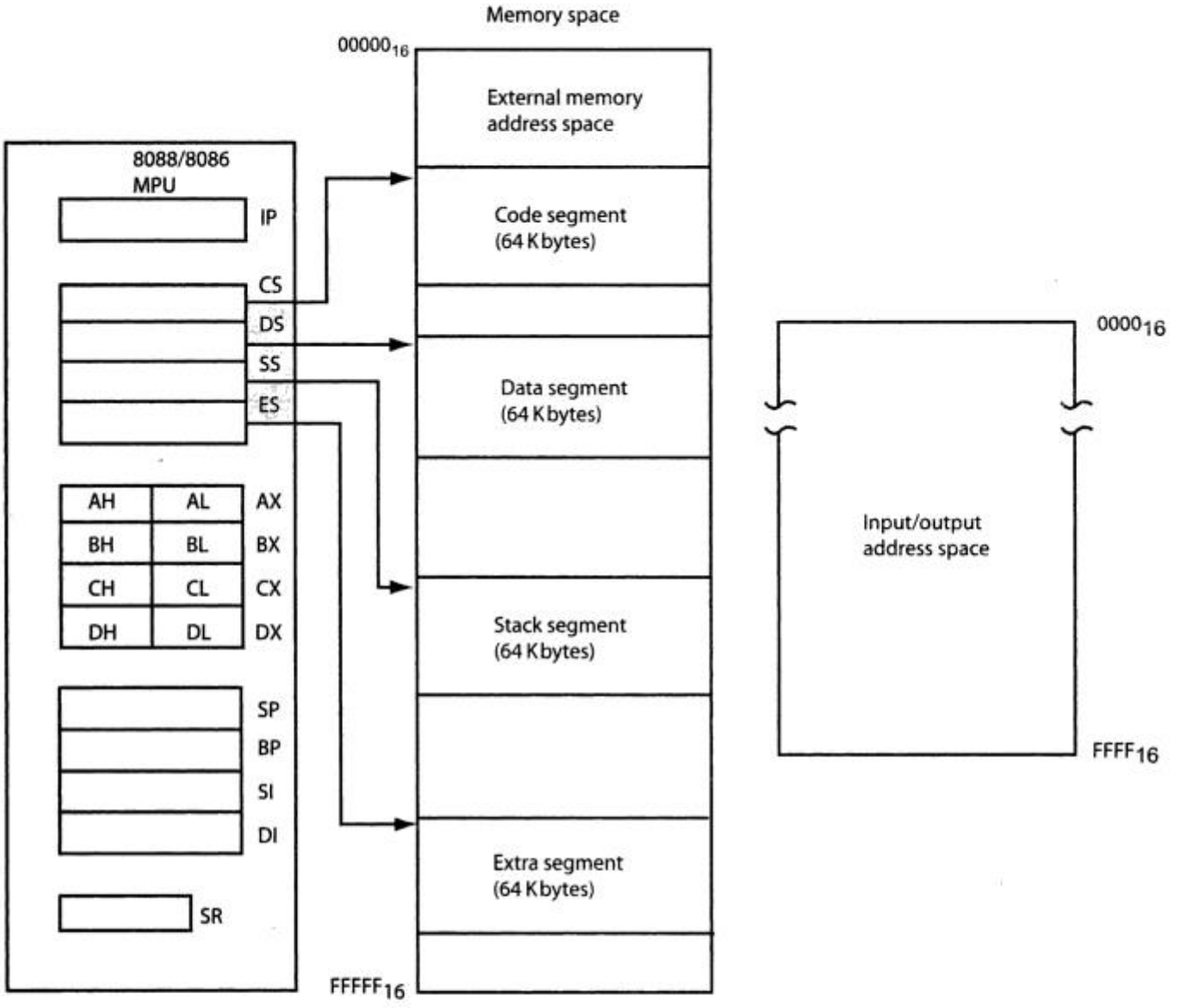

Figure 2–2 Software model of the 8088/8086 microprocessor.

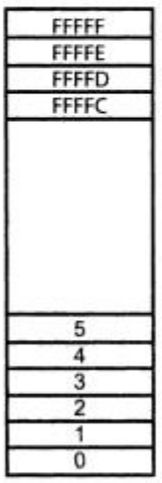

Figure 2–3 Memory address space of the 8088/8086 microprocessor.

address. For example, the word in Fig. 2–4(a) has its least significant byte at address 00724_{16}. Therefore, it is stored at an even-address boundary.

A word of data stored at an even-address boundary, such as 00000_{16}, 00002_{16}, 00004_{16}, and so on, is said to be an *aligned word*—that is, all aligned words are located at an address that is a multiple of 2. On the other hand, a word of data stored at an odd-address boundary, such as 00001_{16}, 00003_{16}, or 00005_{16} and so on, is called a *misaligned word*. Figure 2–5 shows some aligned and misaligned words of data. Here words 0, 2, 4, and 6 are examples of aligned-data words, while words 1 and 5 are misaligned words. Notice that misaligned word 1 consists of byte 1 from aligned word 0 and byte 2 from aligned word 2.

When expressing addresses and data in hexadecimal form, it is common to use the letter H to specify the base. For instance, the number $00AB_{16}$ can also be written as 00ABH.

EXAMPLE 2.1

What is the data word shown in Fig. 2–4(b)? Express the result in hexadecimal form. Is it stored at an even- or odd-addressed word boundary? Is it an aligned or misaligned word of data?

Solution

The most significant byte of the word is stored at address $0072C_{16}$ and equals

$$11111101_2 = FD_{16} = FDH$$

Its least significant byte is stored at address $0072B_{16}$ and is

$$10101010_2 = AA_{16} = AAH$$

Together the two bytes give the word

$$1111110110101010_2 = FDAA_{16} = FDAAH$$

Expressing the address of the least significant byte in binary form gives

$$0072BH = 0072B_{16} - 00000000011100101011_2$$

Because the rightmost bit (LSB) is logic 1, the word is stored at an odd-address boundary in memory; therefore, it is a misaligned word of data.

The *double word* is another data form that can be processed by the 8088 microcomputer. A double word corresponds to four consecutive bytes of data stored in memory; an example of double-word data is a *pointer*. A pointer is a two-word address element that is used to access data or code in memory. The

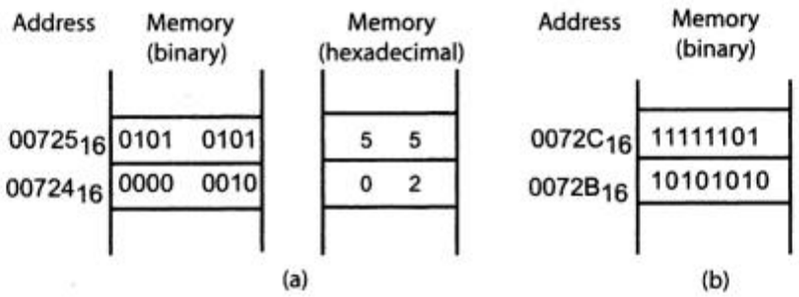

Figure 2–4 (a) Storing a word of data in memory. (b) An example.