fetched, processed, and finally the results are produced. In other words, the execution of an instruction is completed in an instruction cycle.

Von Neumann computer organization was a revolutionary concept and held the center stage in computer design for the next several decades. That is, the organizations of all computers that were designed over the next few decades were essentially based on the Von Neumann style of architecture. However toward the end of 1980s, the quest for faster computations made designers to notice a few shortcomings of the Von Neumann computing. One of the shortcomings of the Von Neumann computers arises due to the fact that a single connection exists between the processor and memory (see Fig. 1.2). Consequently, at any time only one memory access can occur. That is, at a time either an instruction can be fetched or a data item can be accessed. This appeared as a problem in parallel execution of instructions (this issue is discussed in detail in Chapter 10). The term Von Neumann bottleneck is often used to indicate that both fetching an instruction and accessing (reading and writing) data over the same bus from memory by the processor at the same time in a Von Neumann computer is not possible, this is a bottleneck in achieving high-performance computations.

1.5 Basic Organization of a Computer

The important parts of a digital computer are the processor, main memory, hard disk (secondary memory), key board, monitor, and peripheral devices. In the simplest organization of a computer, all these different parts of a computer can be connected through a single bus called a backplane bus as shown in Fig. 1.3.

A single bus interconnecting all the components of a computer is usually called a backplane bus. This is so because the single bus can be considered to be a backbone communication medium, to which various components of the computer are attached. Although a backplane bus was used in the early computers, it was later replaced by multiple specialized buses in modern computers for achieving higher performance.

In the following, we briefly discuss the important parts of a Von Neumann computer.

Processor: The processor or the central processing unit (CPU) is responsible for fetching an instruction stored in the memory and executing it. It contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, and a control unit. The control unit generates the

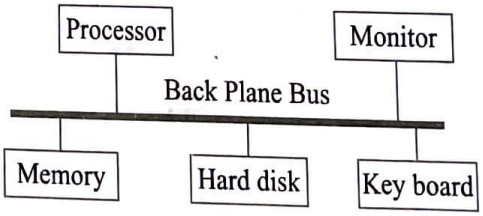

FIGURE 1.3 Basic organization of a digital computer.

necessary control signals for fetching and executing the instructions, and storing the results.

Main memory: The main memory is also called random access memory (RAM) because the CPU can access any location in memory at random and either retrieve the binary information stored at that location, or can store some binary information at the location. The main memory at present is usually made from dynamic RAM (DRAM) modules.

Monitor: The monitor is a visual display unit and serves as the primary output unit. The name monitor comes from the older computer systems, in which a primary video display unit was used by the operators to keep track of (monitor) the internal state of the computer. Over the years the monitors have evolved from pure text display devices to monitors capable of displaying high-quality graphics and animation.

Keyboard: In the early computers, the card reader was the primary input device. However, a keyboard is the primary input device in the modern computers.

Peripheral devices: Several peripheral devices can be attached to the backplane bus. These peripheral devices can be either output devices, such as printers and loud speakers, or can be input devices, such as scanners and cameras. The CPU can provide output to these output devices or collect input from the input devices over the backplane bus.

Backplane bus: The backplane bus is a group of wires. These wires are partitioned into control and address wires. The control wires carry control signals to different units. The address wires carry the address of the specific data in memory and the data wires are used to carry the data. This common bus system is further discussed in Sec. 7.2.

1.6 Historical Perspective

The computers that we use now have evolved over the last seven or eight decades. Although the evolution of computer has been a rather continuous process, for the simplicity of our understanding of the major developments that took place, we can divide these developments into five generations based on the significant improvements in implementation technology or improvements to the other aspects that were incorporated into the computers. We discuss these different generations of computers in the following. We also mark the time period over which these computer generations existed.

First Generation Computers (1941–1956)

The first generation computers are characterized by their use of vacuum tubes for performing computations and magnetic drums for storage of data.

It is necessary to point out that vacuum tubes consumed considerably large amounts of energy, and consequently were getting overly heated up and therefore were breaking down frequently. This resulted in low reliability of these computers and caused significant down time. The magnetic drums could store very small amounts of data. Machine language was the only programming language that was used. Initially an operator was required to execute programs instruction by instruction by performing necessary switch settings and using patch cords for setting up the data and control paths. Later on during the first generation, program execution could be fully automated through the development of the stored program concept. It may be noted that there were no standard or predominant architectures or organizations which could serve as models to the computers that were being designed. This may be because the computers were still in their nascent stage of development. The architectures and organizations varied greatly among the different computers that were developed, as there were no standard or predominant computer architectures or organizations. In addition, the available computers were not general purpose machines, as they were being developed keeping some special applications in mind. This restricted to the number of each computer type that could be sold to at most few dozens.

Second Generation Computers (1956–1963)

The transistor was invented by Shockley, Brattain, and Bardeen in 1947. The use of discrete transistors, in place of vacuum tubes marked the start of the second generation computers. The transistors were made from semiconductor materials and consumed much less power than the vacuum tube devices and consequently were much more reliable. They also operated much faster. The second generation computers used assembly language compared with machine language that was used for programming the first generation computers. Consequently, much larger and more sophisticated programs could be written as compared with the programs that were being written for the first generation computers. Different types of peripheral devices such as printers, magnetic tape drives, magnetic disk storage, and punch cards for program input started to appear, and system programs such as rudimentary operating systems were developed.

Third Generation Computers (1964–1971)

The integrated circuit (IC) was invented in 1958 by Jack Kilby and it provided a big leap to the computers. The third generation computers were implemented using ICs. These computers were much more compact, more energy efficient and reliable than the second generation computers. Peripheral devices such as monitors and keyboards were used. The operating system was vastly improved and allowed a computer to run many different application programs using multiprogramming. High-level languages such as COBOL (Common Business-Oriented Language) and FORTRAN (Formula Translator) were used to write large programs.

Fourth Generation Computers (1971–Present)

Fourth Generation computers are the modern day computers. The size of the fourth generation computers progressively shrunk with improvements to the ICs design and manufacturing technologies, while their performance kept on improving. Very large scale integration (VLSI) and ultra large scale integration (ULSI) ICs made it possible to pack millions of components into a small chip. This reduced the size and price of the computers at the same time increased their power, efficiency, and reliability. The early fourth generation computers were the minicomputers. Starting with the minicomputers as the early fourth generation computers, the computer sizes kept shrinking in the subsequent years—from minicomputers, to desktops, to laptops, and more recently to hand held devices. Networking of computers allowed effortless sharing of data among computers that might be located at widely separated locations, and local area network (LAN) interfaces got built into every computer.

Fifth Generation Computers (Present and Beyond)

Fifth generation computers are not yet a reality but are taking shape in advanced research laboratories. These computers are expected to heavily incorporate artificial intelligence (AI) techniques and are expected to be able to take audio and visual user commands and carry out the instructions. These computers would use massive parallel processing and would have enormous computing powers at their disposal.

1.7 Performance Benchmarking

Before we study the organization and architecture issues in the subsequent chapters, we need to understand how one can determine whether one processor (or computer) works faster than another. Intuitively, a computer works faster than another, if it can execute programs faster than the other. But, this intuitive knowledge about the speed of a computer may be inadequate in many situations. As an example, suppose you are working for a certain company and the company wishes to select a computer that meets a certain budget allocated for this purpose, and would be used to host the company's web site. For this, the company management asks you to select a computer from among the ones available in the market that fits the budget and is likely to perform the best when used as the web server for your company.

Of course, you can float an enquiry to determine the computers that are available in the market and those that meet the budget. But, how do you determine the computer that would perform the best as a web server? If you could actually get all those computers that are available in the market that meet the budget, and then host your company's web server on each of them and measure their performance under various load conditions, then you can determine the one that would perform the best. But, this option is clearly be impractical, considering the huge effort, time, and cost involved.

Number Systems

radix

A number system of *base,* or *radix, r* is a system that uses distinct symbols for r digits. Numbers are represented by a string of digit symbols. To determine the quantity that the number represents, it is necessary to multiply each digit by an integer power of r and then form the sum of all weighted digits.

decimal

For example, the decimal number system in everyday use employs the radix 10 system. The 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The string of digits 724.5 is interpreted to represent the quantity

$$7 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1}$$

that is, 7 hundreds, plus 2 tens, plus 4 units, plus 5 tenths. Every decimal number can be similarly interpreted to find the quantity it represents.

binary

The *binary* number system uses the radix 2. The two digit symbols used are 0 and 1. The string of digits 101101 is interpreted to represent the quantity

$$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 45$$

To distinguish between different radix numbers, the digits will be enclosed in parentheses and the radix of the number inserted as a subscript. For example, to show the equality between decimal and binary forty-five we will write $(101101)_2 = (45)_{10}$.

octal
hexademical

Besides the decimal and binary number systems, the *octal* (radix 8) and *hexadecimal* (radix 16) are important in digital computer work. The eight symbols of the octal system are 0, 1, 2, 3, 4, 5, 6, and 7. The 16 symbols of the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The last six symbols are, unfortunately, identical to the letters of the alphabet and can cause confusion at times. However, this is the convention that has been adopted. When used to represent hexadecimal digits, the symbols A, B, C, D, E, F correspond to the decimal numbers 10, 11, 12, 13, 14, 15, respectively.

A number in radix r can be converted to the familiar decimal system by forming the sum of the weighted digits. For example, octal 736.4 is converted to decimal as follows:

$$\begin{aligned}(736.4)_8 &= 7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1} \\ &= 7 \times 64 + 3 \times 8 + 6 \times 1 + 4/8 = (478.5)_{10}\end{aligned}$$

The equivalent decimal number of hexadecimal F3 is obtained from the following calculation:

$$(F3)_{16} = F \times 16 + 3 = 15 \times 16 + 3 = (243)_{10}$$

conversion

Conversion from decimal to its equivalent representation in the radix r system is carried out by separating the number into its *integer* and *fraction* parts and

converting each part separately. The conversion of a decimal integer into a base r representation is done by successive divisions by r and accumulation of the remainders. The conversion of a decimal fraction to radix r representation is accomplished by successive multiplications by r and accumulation of the integer digits so obtained. Figure 4.1 demonstrates these procedures.

The conversion of decimal 41.6875 into binary is done by first separating the number into its integer part 41 and fraction part .6875. The integer part is converted by dividing 41 by $r = 2$ to give an integer quotient of 20 and a remainder of 1. The quotient is again divided by 2 to give a new quotient and remainder. This process is repeated until the integer quotient becomes 0. The coefficients of the binary number are obtained from the remainders with the first remainder giving the low-order bit of the converted binary number.

The fraction part is converted by multiplying it by $r = 2$ to give an integer and a fraction. The new fraction (*without* the integer) is multiplied again by 2 to give a new integer and a new fraction. This process is repeated until the fraction part becomes zero or until the number of digits obtained gives the required accuracy. The coefficients of the binary fraction are obtained from the integer digits with the first integer computed being the digit to be placed next to the binary point. Finally, the two parts are combined to give the total required conversion.

Octal and Hexadecimal Numbers

The conversion from and to binary, octal, and hexadecimal representation plays an important part in digital computers. Since $2^3 = 8$ and $2^4 = 16$, each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits. The conversion from binary to octal is easily accomplished by partitioning the binary number into groups of three bits each. The corresponding octal digit is then assigned to each group of bits and the string of digits so obtained gives the octal equivalent of the binary number. Consider, for example, a 16-bit register. Physically, one may think of the register as composed of 16 binary storage cells, with each cell capable

Integer = 41

41	
20	1
10	0
5	0
2	1
1	0
0	1

$(41)_{10} = (101001)_2$

Fraction = 0.6875

0.6875
2
1.3750
× 2
0.7500
× 2
1.5000
× 2
1.0000

$(0.6875)_{10} = (0.1011)_2$

$(41.6875)_{10} = (101001.1011)_2$

FIGURE 4.1 Conversion of decimal 41.6875 into binary.

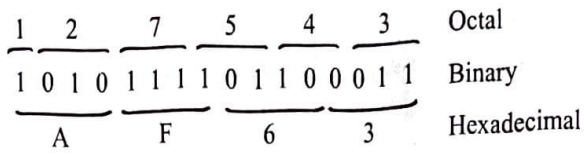

FIGURE 4.2 Binary, octal, and hexadecimal conversion.

of holding either a 1 or a 0. Suppose that the bit configuration stored in the register is as shown in Fig. 4.2. Since a binary number consists of a string of 1's and 0's, the 16-bit register can be used to store any binary number from 0 to $2^{16} - 1$. For the particular example shown, the binary number stored in the register is the equivalent of decimal 44899. Starting from the low-order bit, we partition the register into groups of three bits each (the sixteenth bit remains in a group by itself). Each group of three bits is assigned its octal equivalent and placed on top of the register. The string of octal digits so obtained represents the octal equivalent of the binary number.

Conversion from binary to hexadecimal is similar except that the bits are divided into groups of four. The corresponding hexadecimal digit for each group of four bits is written as shown below the register of Fig. 4.2. The string of hexadecimal digits so obtained represents the hexadecimal equivalent of the binary number. The corresponding octal digit for each group of three bits is easily remembered after studying the first eight entries listed in Table 4.1. The correspondence between a hexadecimal digit and its equivalent 4-bit code can be found in the first 16 entries of Table 4.2.

TABLE 4.1 Binary-Coded Octal Numbers

Octal number	Binary-coded octal	Decimal equivalent	
0	000	0	↑
1	001	1	
2	010	2	Code
3	011	3	for one
4	100	4	octal
5	101	5	digit
6	110	6	
7	111	7	↓
10	001 000	8	
11	001 001	9	
12	001 010	10	
24	010 100	20	
62	110 010	50	
143	001 100 011	99	
370	011 111 000	248	

Table 4.1 lists a few octal numbers and their representation in registers in binary-coded form. The binary code is obtained by the procedure explained above. Each octal digit is assigned a 3-bit code as specified by the entries of the first eight digits in the table. Similarly, Table 4.2 lists a few hexadecimal numbers and their representation in registers in binary-coded form. Here the binary code is obtained by assigning to each hexadecimal digit the 4-bit code listed in the first 16 entries of the table.

Comparing the binary-coded octal and hexadecimal numbers with their binary number equivalent we find that the bit combination in all three representations is exactly the same. For example, decimal 99, when converted to binary, becomes 1100011. The binary-coded octal equivalent of decimal 99 is 001 100 011 and the binary-coded hexadecimal of decimal 99 is 0110 0011. If we neglect the leading zeros in these three binary representations, we find that their bit combination is identical. This should be so because of the straightforward conversion that exists between binary numbers and octal or hexadecimal. The point of all this is that a string of 1's and 0's stored in a register could represent a binary number, but this same string of bits may be

TABLE 4.2 Binary-Coded Hexadecimal Numbers

Hexadecimal number	Binary-coded hexadecimal	Decimal equivalent
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15
14	0001 0100	20
32	0011 0010	50
63	0110 0011	99
F8	1111 1000	248

Code for one hexadecimal digit

interpreted as holding an octal number in binary-coded form (if we divide the bits in groups of three) or as holding a hexadecimal number in binary-coded form (if we divide the bits in groups of four).

The registers in a digital computer contain many bits. Specifying the content of registers by their binary values will require a long string of binary digits. It is more convenient to specify content of registers by their octal or hexadecimal equivalent. The number of digits is reduced by one-third in the octal designation and by one-fourth in the hexadecimal designation. For example, the binary number 1111 1111 1111 has 12 digits. It can be expressed in octals as 7777 (four digits) or in hexadecimal as FFF (three digits). Computer manuals invariably choose either the octal or the hexadecimal designation for specifying contents of registers.

Decimal Representation

The binary number system is the most natural system for a computer, but people are accustomed to the decimal system. One way to solve this conflict is to convert all input decimal numbers into binary numbers, let the computer perform all arithmetic operations in binary and then convert the binary results back to decimal for the human user to understand. However, it is also possible for the computer to perform arithmetic operations directly with decimal numbers provided they are placed in registers in a coded form. Decimal numbers enter the computer usually as binary-coded alphanumeric characters. These codes, introduced later, may contain from six to eight bits for each decimal digit. When decimal numbers are used for internal arithmetic computations, they are converted to a binary code with four bits per digit.

binary code

A binary code is a group of n bits that assume up to 2^n distinct combinations of 1's and 0's with each combination representing one element of the set that is being coded. For example, a set of four elements can be coded by a 2-bit code with each element assigned one of the following bit combinations; 00, 01, 10, or 11. A set of eight elements requires a 3-bit code, a set of 16 elements requires a 4-bit code, and so on. A binary code will have some unassigned bit combinations if the number of elements in the set is not a multiple power of 2. The 10 decimal digits form such a set. A binary code that distinguishes among 10 elements must contain at least four bits, but six combinations will remain unassigned. Numerous different codes can be obtained by arranging four bits in 10 distinct combinations. The bit assignment most commonly used for the decimal digits is the straight binary assignment listed in the first 10 entries of Table 4.3. This particular code is called *binary-coded decimal* and is commonly referred to by its abbreviation BCD. Other decimal codes are sometimes used and a few of them are given in Sec. 4.5.

BCD

It is very important to understand the difference between the *conversion* of decimal numbers into binary and the *binary coding* of decimal numbers. For example, when *converted* to a binary number, the decimal number 99 is represented by the string of bits 1100011, but when represented in BCD, it becomes 1001 1001. The *only* difference between a decimal number repre-

TABLE 4.3 Binary-Coded Decimal (BCD) Numbers

Decimal number	Binary-coded decimal (BCD) number	
0	0000	↑ Code for one decimal digit
1	0001	
2	0010	
3	0011	
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	1001	↓
10	0001 0000	
20	0010 0000	
50	0101 0000	
99	1001 1001	
248	0010 0100 1000	

sented by the familiar digit symbols 0, 1, 2, ... , 9 and the BCD symbols 0001, 0010, ... , 1001 is in the symbols used to represent the digits—the number itself is exactly the same. A few decimal numbers and their representation in BCD are listed in Table 4.3.

Alphanumeric Representation

Many applications of digital computers require the handling of data that consist not only of numbers, but also of the letters of the alphabet and certain special characters. An *alphanumeric character set* is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet and a number of special characters, such as $, +, and =. Such a set contains between 32 and 64 elements (if only uppercase letters are included) or between 64 and 128 (if both uppercase and lowercase letters are included). In the first case, the binary code will require six bits and in the second case, seven bits. The standard alphanumeric binary code is the ASCII (American Standard Code for Information Interchange), which uses seven bits to code 128 characters. The binary code for the uppercase letters, the decimal digits, and a few special characters is listed in Table 4.4. Note that the decimal digits in ASCII can be converted to BCD by removing the three high-order bits, 011. A complete list of ASCII characters is provided in Table 11.1.

character

ASCII

Binary codes play an important part in digital computer operations. The codes must be in binary because registers can only hold binary information. One must realize that binary codes merely change the symbols, not the meaning of the discrete elements they represent. The operations specified for digital computers must take into consideration the meaning of the bits stored in registers so that operations are performed on operands of the same type. In inspecting the bits of a computer register at random, one is likely to find that it represents some type of coded information rather than a binary number.

TABLE 4.4 American Standard Code for Information Interchange (ASCII)

Character	Binary code	Character	Binary code
A	100 0001	0	011 0000
B	100 0010	1	011 0001
C	100 0011	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011		
L	100 1100		
M	100 1101	space	010 0000
N	100 1110	.	010 1110
0	100 1111	(	010 1000
P	101 0000	+	010 1011
Q	101 0001	$	010 0100
R	101 0010	*	010 1010
S	101 0011	)	010 1001
T	101 0100	–	010 1101
U	101 0101	/	010 1111
V	101 0110	,	010 1100
W	101 0111	=	011 1101
X	101 1000		
Y	101 1001		
Z	101 1010		

bits gives the decimal digit. For example, the bit combination 1101, when weighted by the respective digits 2421, gives the decimal equivalent of $2 \times 1 + 4 \times 1 + 2 \times 0 + 1 + 1 = 7$. The BCD code can be assigned the weights 8421 and for this reason it is sometimes called the 8421 code.

The excess-3 code is a decimal code that has been used in older computers. This is an unweighted code. Its binary code assignment is obtained from the corresponding BCD equivalent binary number after the addition of binary 3 (0011). *excess-3 code*

From Table 4.5 we note that the Gray code is not suited for a decimal code if we were to choose the first 10 entries in the table. This is because the transition from 9 back to 0 involves a change of three bits (from 1101 to 0000). To overcome this difficulty, we choose the 10 numbers starting from the third entry 0010 up to the twelfth entry 1010. Now the transition from 1010 to 0010 involves a change of only one bit. Since the code has been shifted up three numbers, it is called the excess-3 Gray. This code is listed with the other decimal codes in Table 4.6.

Other Alphanumeric Codes

The ASCII code (Table 4.3) is the standard code commonly used for the transmission of binary information. Each character is represented by a 7-bit code and usually an eighth bit is inserted for parity (see Sec. 4.6). The code consists of 128 characters. Ninety-five characters represent *graphic symbols* that include upper-and lowercase letters, numerals zero to nine, punctuation marks, and special symbols. Twenty-three characters represent *format effectors,* which are functional characters for controlling the layout of printing or display devices such as carriage return, line feed, horizontal tabulation, and back space. The other 10 characters are used to direct the data communication flow and report its status.

Another alphanumeric (sometimes called *alphameric*) code used in IBM equipment is the EBCDIC (Extended BCD Interchange Code). It uses eight bits for each character (and a ninth bit for parity). EBCDIC has the same character symbols as ASCII but the bit assignment to characters is different. *EBCDIC*

When alphanumeric characters are used internally in a computer for data processing (not for transmission purposes) it is more convenient to use a 6-bit code to represent 64 characters. A 6-bit code can specify the 26 uppercase letters of the alphabet, numerals zero to nine, and up to 28 special characters. This set of characters is usually sufficient for data-processing purposes. Using fewer bits to code characters has the advantage of reducing the memory space needed to store large quantities of alphanumeric data.

4.7 Error Detection Codes

Binary information transmitted through some form of communication medium is subject to external noise that could change bits from 1 to 0, and