

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

MODULE 1

Basic Structure of Computers: Computer Types, Functional Units, Basic Operational Concepts, Bus Structures, Software, Performance - Processor Clock, Basic Performance Equation.

Machine Instructions and Programs: Numbers, Arithmetic Operations and Characters, IEEE standard for Floating point Numbers, Memory Location and Addresses, Memory Operations, Instructions and Instruction Sequencing.

BASIC STRUCTURE OF COMPUTERS

COMPUTER TYPES

A Computer is a fast electronic calculating machine that accepts digitized input information, processes it according to a list of internally stored instructions, and produces the resulting output information. The list of instructions is called a computer *program*. The internal storage is called *computer memory*.

Many types of computers exist that differ widely in size, cost, computational power, and intended use. The different types of computers are:

1. Personal computer:

- Found wide use in homes, schools, & business offices.
- Most common form of desktop computers.
- Have processing and storage units, visual display and audio output units, and a keyboard that can all be located easily on a home or office desk.
- The storage media include hard disks, CD-ROMs, and diskettes.

2. Portable Notebook Computer:

- Compact version of the personal computer with all of these components packaged into a single unit the size of a thin briefcase.
- Compact and Lightweight personal computers designed for mobility.
- Commonly referred to as laptops.

3. Workstations:

- Computers with high-resolution graphics input / output capability.
- Have significantly more computational power than personal computers.
- Often used by professionals in engineering applications, especially for interactive design work (graphic design, animations etc.).

4. Enterprise systems or mainframes:

- Used for business data processing in medium to large corporations.

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

- ❑ Requires much more computing power and storage capacity than workstations.
- ❑ Designed to support and manage the core business operations and information flow of an organization.
- ❑ Eg: Enterprise Resource Planning (ERP) Systems.

5. Servers:

- ❑ Powerful computers or systems that provide resources, data, services, or programs to other computers, known as clients, over a network.
- ❑ Contain sizable database storage units and are capable of handling large volumes of requests to access the data.
- ❑ Widely accessible to the education, business, and personal user communities.
- ❑ The requests and responses are usually transported over Internet communication facilities.

6. Super Computers:

- ❑ Extremely powerful computing machines designed to perform complex and large-scale computations at exceptionally high speeds.
- ❑ Used for the large-scale numerical calculations required in applications such as weather forecasting, aircraft design and scientific simulation.

NOTE: In enterprise systems, servers, and supercomputers, the functional units, including multiple processors, may consist of a number of separate and large units.

FUNCTIONAL UNITS

A computer consists of 5 functionally independent main parts: input, memory, arithmetic & logic unit, output & control units, as shown in figure 1.1.

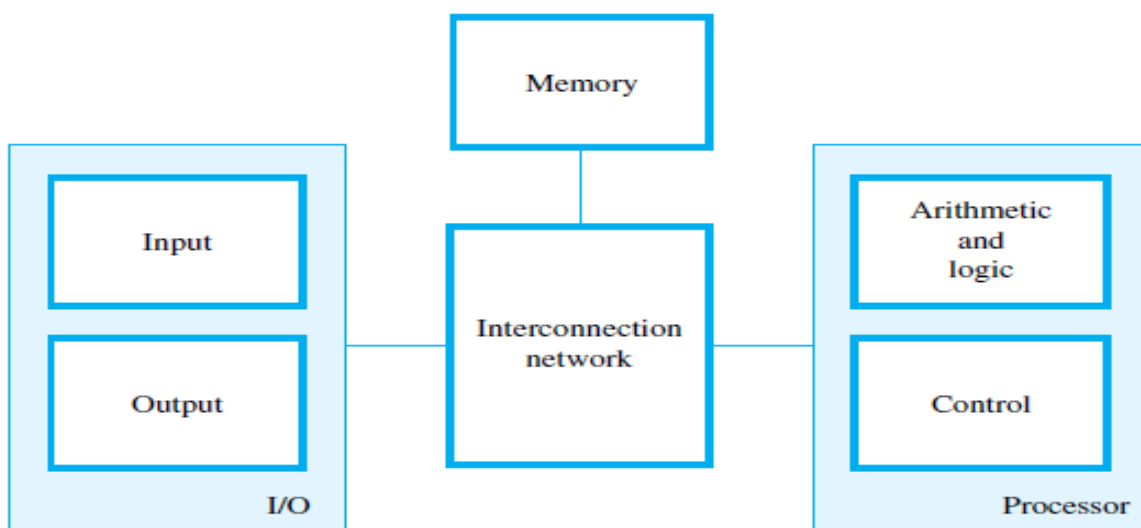


Figure 1.1 Basic functional units of a computer.

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

Input Unit: Computers accept coded information through input units, which read the data. Eg: keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Other examples: joysticks, trackballs, and mouse. Microphones can be used to capture audio input which is then sampled and converted into digital codes for storage and processing.

Memory Unit: The function of memory unit is to store programs and data. There are two classes of storage: primary and secondary.

Primary storage is a fast memory & operates at electronic speeds. Programs must be stored in the memory while they are being executed. Primary storage is expensive. Thus additional, cheaper, *secondary storage* is used when large amounts of data and many programs have to be stored. Example: Magnetic disks, tapes & optical disks (CD-ROMs), Hard disks etc.

Arithmetic and Logic Unit (ALU): Most computer operations are executed in ALU of the processor. Example: To add two numbers located in the memory, they are brought into the processor, and addition is carried out by the ALU. The sum may then be stored in the memory or used immediately.

Any other arithmetic or logic operation, for example, multiplication, division, or comparison of numbers, is initiated by bringing the required operands into the processor, where the operation is performed by the ALU. When operands are brought into the processor, they are stored in high-speed storage elements called *registers*. Each register can store one word of data.

Output Unit: The function of output unit is to send the processed results to the outside world. Eg: Monitor, Printer. Some units, such as graphic displays, provide both an output function and an input function.

Control Unit: The operation of various units must be coordinated & it is the task of the control unit. The control unit is effectively the nerve center that sends control signals to other units and senses their states.

I/O transfers, consisting of input and output operations, are controlled by the instructions of I/O programs that identify the devices involved and the information to be transferred. Data transfers between the processor and the memory are also controlled by the control unit through timing signals.

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

The operation of a computer can be summarized as:

- ❑ The computer accepts information in the form of programs and data through an input unit & stores it in the memory.
- ❑ Information stored in the memory is fetched, under program control, into an arithmetic and logic unit, where it is processed.
- ❑ Processed information leaves the computer through an output unit.
- ❑ All activities inside the machine are directed by the control unit.

BASIC OPERATIONAL CONCEPTS

To perform a given task, an appropriate program consisting of a list of instructions is stored in the memory. An Instruction consists of 2 parts: Operation code (Opcode) and Operands. The data/operands are stored in memory. The individual instructions are brought from the memory to the processor. Then, the processor performs the specified operation.

Consider a typical instruction example: **ADD LOCA, R0**. The steps to execute this instruction are:

Step 1: Fetch the instruction from main memory into processor.

Step 2: Fetch the operand at location LOCA from main-memory into the processor.

Step 3: Add the memory operand (i.e. fetched contents of LOCA) to the contents of register R0.

Step 4: Store the result (sum) in R0.

The above instruction can be realized using 2 instructions as:

Load LOCA, R1

Add R1, R0

The steps to execute the instruction are:

Step 1: Fetch instruction from main-memory into processor.

Step 2: Fetch the operand at location LOCA from main-memory into register R1.

Step 3: Add content of Register R1 & contents of register R0.

Step 4: Store the result (sum) in R0.

Figure 1.2 shows how the memory and the processor can be connected. In addition to ALU and control circuitry, the processor contains a number of registers used for different purposes.

Instruction Register (IR): IR holds the instruction that is currently being executed.

Its output is available to the control circuits, which generate the timing signal.

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

Program Counter (PC): PC contains the memory-address of the next-instruction to be fetched & executed. During the execution of an instruction, the contents of PC are updated to point to next instruction.

Memory Address Register: MAR holds the address of the memory location to be accessed.

Memory Data Register: MDR contains the data to be written into or read out of the addressed location.

MAR and MDR facilitates the communication with memory.

General Purpose Registers: Named R_0 through R_{n-1} .

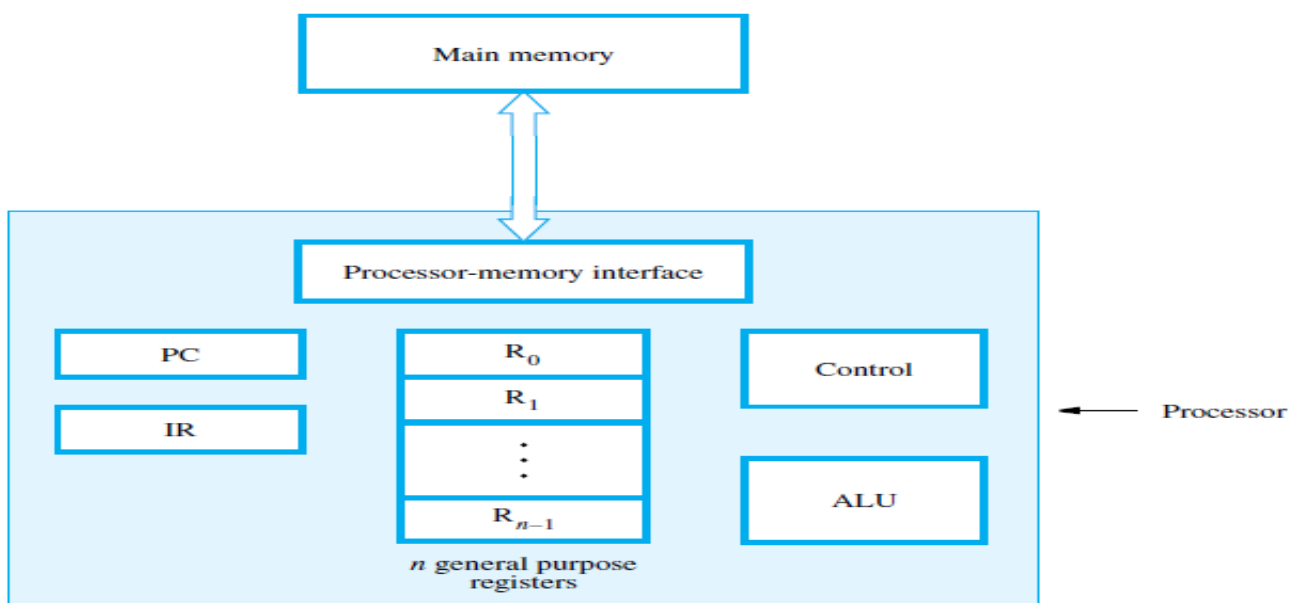


Figure 1.2 Connection between the processor and the main memory.

The steps to execute an instruction are:

- 1) The address of first instruction (to be executed) gets loaded into PC.
- 2) The contents of PC (i.e. address) are transferred to the MAR & control-unit issues Read signal to memory.
- 3) After certain amount of elapsed time, the first instruction is read out of memory and placed into MDR.
- 4) Next, the contents of MDR are transferred to IR. At this point, the instruction can be decoded & executed.
- 5) To fetch an operand, its address is placed into MAR & control-unit issues Read signal. As a result, the operand is transferred from memory into MDR, and then it is transferred from MDR to ALU.
- 6) Likewise required number of operands is fetched into processor.

- 7) Finally, ALU performs the desired operation.
- 8) If the result of this operation is to be stored in the memory, then the result is sent to the MDR.
- 9) The address of the location where the result is to be stored is sent to the MAR and a Write cycle is initiated.
- 10) At some point during execution, contents of PC are incremented to point to next instruction in the program.

BUS STRUCTURE

A bus is a group of lines that serves as a connecting path for several devices. A bus may be lines or wires. The lines carry data or address or control signal. There are 2 types of Bus structures: single bus structure and multiple bus structure.

Single Bus Structure: The simplest way to interconnect functional units is to use a single bus, as shown in figure 1.3. All units are connected to this bus. Because the bus can be used for only one transfer at a time, only 2 units can actively use the bus at any given time. Bus control lines are used to arbitrate multiple requests for use of the bus.

Advantages: Low cost & Flexibility for attaching peripheral devices.

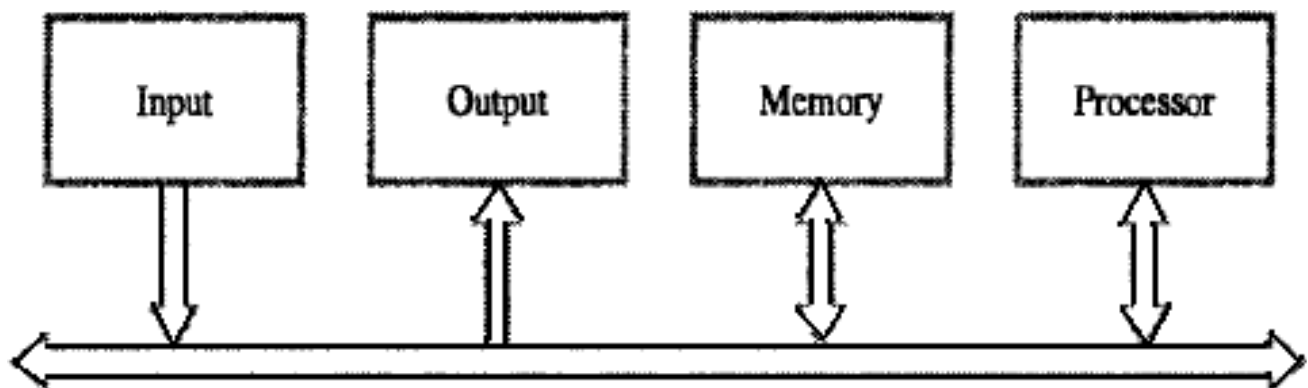


Figure 1.3 Single-bus structure.

Multiple Bus Structure: Systems that contain multiple buses achieve more concurrency in operations. 2 or more transfers can be carried out at the same time.

Advantage: Better performance.

Disadvantage: Increased cost.

The devices connected to a bus vary widely in their speed of operation. To synchronize their operational-speed, buffer-registers can be used. *Buffer Registers* are included with the devices to hold the information during transfers. They prevent a high-speed processor from being locked to a slow I/O device during data transfers.

SOFTWARE

System software is a collection of programs that are executed as needed to perform functions such as:

- Receiving and interpreting user commands
- Entering and editing application programs and storing them as files in secondary storage devices
- Managing the storage and retrieval of files in secondary storage devices
- Running standard application programs such as word processors, spreadsheets, or games, with data supplied by the user
- Controlling I/O units to receive input information and produce output results
- Translating programs from source form prepared by the user into object form consisting of machine instructions
- Linking and running user-written application programs with existing standard library routines, such as numerical computation packages

System software is thus responsible for the coordination of all activities in a computing system.

Application programs are usually written in a high-level programming language, such as C, C++, Java, or Fortran. A programmer using a high-level language need not know the details of machine program instructions. A system software program called a *compiler* translates the high-level language program into a suitable machine language program. *Text editor* is a system program that all programmers use for entering and editing application programs.

Operating System (OS) is a large program, or a collection of routines, that is used to control the sharing of and interaction among various computer units as they execute application programs. The OS routines perform the tasks required to assign computer resources to individual application programs. These tasks include assigning memory and magnetic disk space to program and data files, moving data between memory and disk units, and handling I/O operations.

Sharing of processor: Consider a system with one processor, one disk, and one printer. First the file is transferred into the memory and then execution of the program is started. When execution reaches the point where the data file is needed, the program requests the OS to transfer the data file from the disk to the memory. The OS performs this task and passes execution control back to the application

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

program, which then proceeds to perform the required computation. When the computation is completed and the results are ready to be printed, the application program again sends a request to the operating system. An OS routine is then executed to cause the printer to print the results.

The sharing of the processor execution time is illustrated by a time-line diagram, shown in Figure 1.4. During the time period t_0 to t_1 , an OS routine initiates loading the application program from disk to memory, waits until the transfer is completed, and then passes execution control to the application program. A similar pattern of activity occurs during period t_2 to t_3 and period t_4 to t_5 , when the OS transfers the data file from the disk and prints the results. At t_5 , the OS may load and execute another application program.

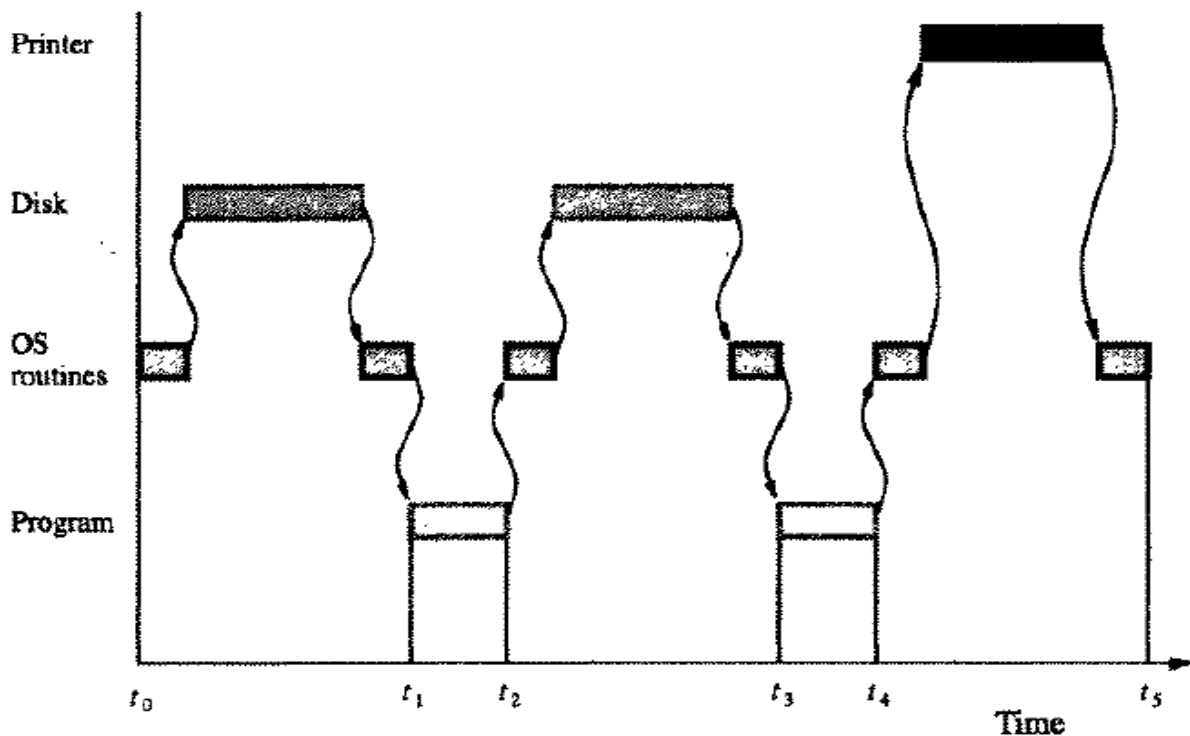


Figure 1.4 User program and OS routine sharing of the processor.

The disk and the processor are idle during most of the time period t_4 to t_5 , the OS can load the next program to be executed into the memory from the disk while the printer is operating. Similarly, during t_0 to t_1 , the OS can arrange to print the previous program's results while the current program is being loaded from the disk. Thus, OS manages the concurrent execution of several application programs to make the best possible use of computer resources. This pattern of concurrent execution is called *multiprogramming* or *multitasking*.

PERFORMANCE

The most important measure of the performance of a computer is how quickly it can execute programs. The speed with which a computer executes programs is affected by the design of its instruction set, its hardware and its software, including the operating system, and the technology in which the hardware is implemented. Because programs are usually written in a high-level language, performance is also affected by the compiler that translates programs into machine language.

The total time required to execute the program is called *elapsed time* and is a measure of the performance of the entire computer system. It is affected by the speed of the processor, the disk, and the printer. The performance of the processor is considered only for the periods during which the processor is active. The *processor time* depends on the hardware involved in the execution of individual machine instructions. The flow of program instructions and data between the memory and the processor are explained below:

- ❑ At the start of execution, all program instructions and the required data are stored in the main memory.
- ❑ As execution proceeds, instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache.
- ❑ When the execution of an instruction calls for data located in the main memory, the data are fetched and a copy is placed in the cache.
- ❑ Later, if the same instruction or data item is needed a second time, it is read directly from the cache.

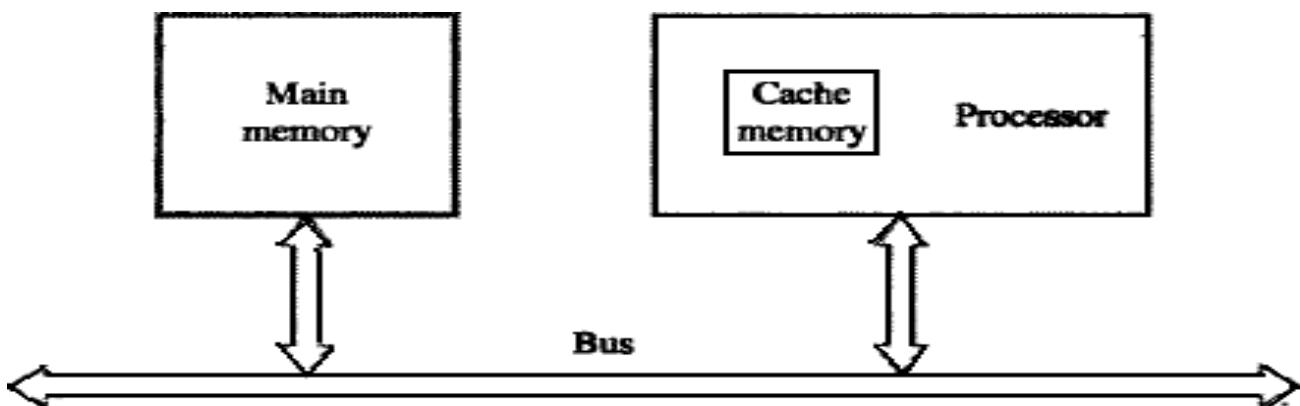


Figure 1.5 The processor cache.

A program will be executed faster if the movement of instructions and data between the main memory and the processor is minimized, which is achieved by using the cache.

PROCESSOR CLOCK

Processor circuits are controlled by a timing signal called a *clock*. The clock defines regular time intervals, called *clock cycles*. To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps, such that each step can be completed in one clock cycle. If P is the length of one clock cycle then its inverse is the clock rate, $R = 1/P$, which is measured in *cycles per second*. Cycles per second is called hertz (Hz).

BASIC PERFORMANCE EQUATION

Let 'T' be the processor time required to execute a program and 'N' be the actual number of instruction executions. The average number of basic steps needed to execute one machine instruction is S, where each basic step is completed in one clock cycle. If the clock rate is R cycles per second, the program execution time is

$$T = (N \times S) / R$$

This equation is often referred to as the basic performance equation.

To achieve high performance, the computer designer must reduce the value of T, which means reducing N and S, and increasing R. The value of N is reduced if the source program is compiled into fewer machine instructions. The value of S is reduced if instructions have a smaller number of basic steps to perform or if the execution of instructions is overlapped. Using a higher-frequency clock increases the value of R, which means that the time required to complete a basic execution step is reduced.

Problem: A program contains 1000 instructions. Out of that 25% instructions requires 4 clock cycles, 40% instructions requires 5 clock cycles and remaining require 3 clock cycles for execution. Find the total time required to execute the program running in a 1 GHz machine.

Solution:

$$N = 1000$$

25% of N= 250 instructions require 4 clock cycles.

40% of N =400 instructions require 5 clock cycles.

35% of N=350 instructions require 3 clock cycles.

$$T = (N \times S) / R = (250 \times 4 + 400 \times 5 + 350 \times 3) / 1 \times 10^9$$

$$T = (1000 + 2000 + 1050) / 1 \times 10^9$$

$$T = 4.05 \mu s.$$

MACHINE INSTRUCTIONS AND PROGRAMS

NUMBERS, ARITHMETIC OPERATIONS AND CHARACTERS

Number Representation: Three systems used for representing binary numbers are: Sign-and-magnitude, 1's complement and 2's complement. In all three systems, the leftmost bit is 0 for positive numbers (MSB=0) and 1 for negative numbers (MSB=1). Figure 1.6 illustrates all three representations using 4-bit numbers.

B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	-0	-7	-8
1 0 0 1	-1	-6	-7
1 0 1 0	-2	-5	-6
1 0 1 1	-3	-4	-5
1 1 0 0	-4	-3	-4
1 1 0 1	-5	-2	-3
1 1 1 0	-6	-1	-2
1 1 1 1	-7	-0	-1

Figure 1.6: Binary, signed-integer representations.

In *sign-and-magnitude system*, negative value is obtained by changing the MSB from 0 to 1 of the corresponding positive value. For example, +5 is represented by 0101 & -5 is represented by 1101.


In *1's complement system*, negative values are obtained by complementing each bit of the corresponding positive number. For example, -5 is obtained by complementing each bit in 0101 to yield 1010. The operation of forming the 1's complement of a given number is equivalent to subtracting that number from $2^n - 1$.

In *2's complement system*, forming the 2's complement of a number is done by subtracting that number from 2^n . For example, -5 is obtained by complementing each bit in 0101 & then adding 1 to yield 1011. The 2's complement of a number is obtained by adding 1 to the 1's complement of that number.

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

Addition of unsigned numbers: Consider adding two 1-bit numbers. The sum of 1 & 1 requires the 2-bit vector 10 to represent the value 2. We say that sum is 0 and the carry out is 1 as illustrated in figure 1.7.

$$\begin{array}{r}
 0 \\
 + 0 \\
 \hline
 0
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 + 0 \\
 \hline
 1
 \end{array}
 \qquad
 \begin{array}{r}
 0 \\
 + 1 \\
 \hline
 1
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 + 1 \\
 \hline
 10
 \end{array}$$



 Carry-out

Figure 1.7: Addition of 1-bit numbers.

Addition and subtraction of signed numbers: The 2's-complement system is the most efficient method for performing addition and subtraction operations. The rules governing addition and subtraction of n -bit signed numbers using the 2's-complement representation system may be stated as follows:

- ❑ To *add* two numbers, add their n -bit representations, ignoring the carry-out bit from the most significant bit (MSB) position. The sum will be the algebraically correct value in 2's-complement representation if the actual result is in the range -2^{n-1} through $+2^{n-1} - 1$.
- ❑ To *subtract* two numbers X and Y , i.e., to perform $X - Y$, form the 2's-complement of Y , then add it to X using the *add* rule. Again, the result will be the algebraically correct value in 2's-complement representation if the actual result is in the range -2^{n-1} through $+2^{n-1} - 1$.

Figure 1.8 shows some examples of addition and subtraction in the 2's-complement system.

In 1's complement representation, the result obtained after an addition operation is not always correct. The carry-out (C_0) cannot be ignored. If $C_0=0$, the result obtained is correct. If $C_0=1$, then a 1 must be added to the result to make it correct.

Overflow in Integer Arithmetic: When result of an arithmetic operation is outside the representable-range, an arithmetic overflow is said to occur. For example, if we add two numbers +7 and +4, then the output sum S is 1011, which is the code for -5, an incorrect result. An overflow occurs in following 2 cases:

- 1) Overflow can occur only when adding two numbers that have the same sign.
- 2) The carry-out signal from the sign-bit position is not a sufficient indicator of overflow when adding signed numbers.

**COMPUTER ORGANIZATION &
ARCHITECTURE – BEC306C**

<p>(a) $\begin{array}{r} 0010 \\ + 0011 \\ \hline 0101 \end{array}$</p> <p>(c) $\begin{array}{r} 1011 \\ + 1110 \\ \hline 1001 \end{array}$</p> <p>(e) $\begin{array}{r} 1101 \\ - 1001 \\ \hline \end{array}$</p> <p>(f) $\begin{array}{r} 0010 \\ - 0100 \\ \hline \end{array}$</p> <p>(g) $\begin{array}{r} 0110 \\ - 0011 \\ \hline \end{array}$</p> <p>(h) $\begin{array}{r} 1001 \\ - 1011 \\ \hline \end{array}$</p> <p>(i) $\begin{array}{r} 1001 \\ - 0001 \\ \hline \end{array}$</p> <p>(j) $\begin{array}{r} 0010 \\ - 1101 \\ \hline \end{array}$</p>	<p>(+2) (+3) (+5)</p> <p>(-5) (-2) (-7)</p> <p>(-3) (-7)</p> <p>(+2) (+4)</p> <p>(+6) (+3)</p> <p>(-7) (-5)</p> <p>(-7) (+1)</p> <p>(+2) (-3)</p>	<p>⇒</p> <p>⇒</p> <p>⇒</p> <p>⇒</p> <p>⇒</p> <p>⇒</p> <p>⇒</p>	<p>(b) $\begin{array}{r} 0100 \\ + 1010 \\ \hline 1110 \end{array}$</p> <p>(d) $\begin{array}{r} 0111 \\ + 1101 \\ \hline 0100 \end{array}$</p> <p>$\begin{array}{r} 1101 \\ + 0111 \\ \hline 0100 \end{array}$</p> <p>$\begin{array}{r} 0010 \\ + 1100 \\ \hline 1110 \end{array}$</p> <p>$\begin{array}{r} 0110 \\ + 1101 \\ \hline 0011 \end{array}$</p> <p>$\begin{array}{r} 1001 \\ + 0101 \\ \hline 1110 \end{array}$</p> <p>$\begin{array}{r} 1001 \\ + 1111 \\ \hline 1000 \end{array}$</p> <p>$\begin{array}{r} 0010 \\ + 0011 \\ \hline 0101 \end{array}$</p>	<p>(+4) (-6) (-2)</p> <p>(+7) (-3) (+4)</p> <p>(+4)</p> <p>(-2)</p> <p>(-2)</p> <p>(+3)</p> <p>(-2)</p> <p>(-8)</p> <p>(+5)</p>
---	---	--	---	---

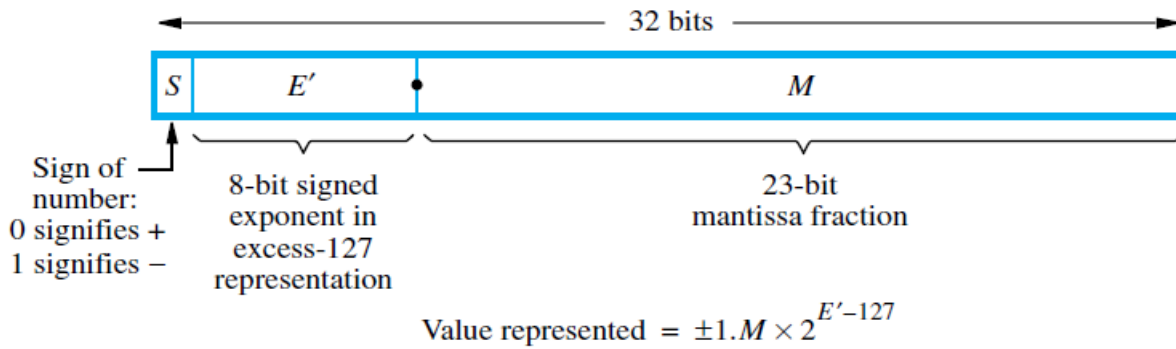
Figure 1.9: 2's-complement Add and Subtract operations.

Characters: The most common encoding scheme for characters is ASCII (American Standard Code for Information Interchange). Alphanumeric characters, operators, punctuation symbols, and control characters are represented by 7-bit codes. It is convenient to use an 8-bit *byte* to represent and store a character. The code occupies the low-order seven bits. The high-order bit is usually set to 0.

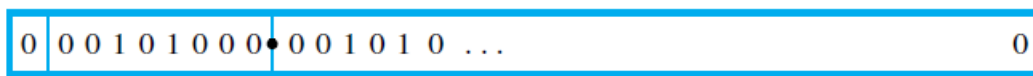
IEEE STANDARD FOR FLOATING POINT NUMBERS

The standard for representing floating point numbers in 32-bit is developed by IEEE (Institute of Electrical & Electronics Engineers). Basic IEEE format representation is shown in Figure 1.9a. First bit represents the sign of the number. E is signed exponent and E is in the range $-126 \leq E \leq 127$. E' is unsigned exponent and $E' = E + 127$. This is called excess-127 format. Thus E' is in the range $0 \leq E' \leq 255$. The last 23 bits represent the mantissa. Since binary normalization is used, the MSB of the mantissa is always equal to 1. (M represents fractional-part).

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

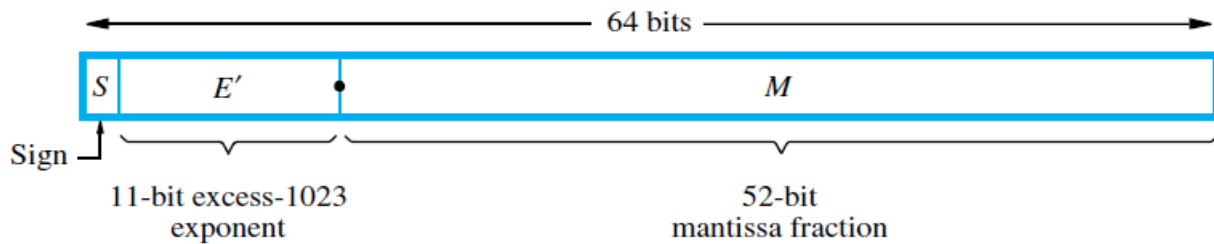


(a) Single precision



Value represented = $1.001010 \dots 0 \times 2^{-87}$

(b) Example of a single-precision number



Value represented = $\pm 1.M \times 2^{E'-1023}$

(c) Double precision

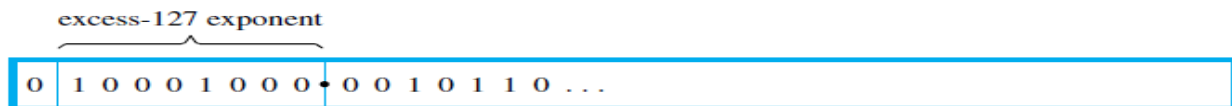
Figure 1.10: IEEE standard floating point formats

Single precision representation (fig. 1.10a) occupies a single 32-bit word. The 32 bit word is divided into 3 fields: sign (1 bit), exponent (8 bits) and mantissa (23 bits). The scale factor has a range of 2^{-126} to 2^{+127} , which is approximately equal to $10^{\pm 38}$. The 24-bit mantissa provides a precision equivalent to about 7 decimal-digits. An example of a single-precision floating-point number is shown in Figure 1.10b.

Double-precision format (fig. 1.10c) occupies a single 64-bit word, has increased exponent and mantissa ranges. The 11-bit excess-1023 exponent E' has the range $1 \leq E \leq 2046$ for normal values. The actual E is in the range $-1022 \leq E \leq 1023$, providing scale factors of 2^{-1022} to 2^{1023} (approximately $10^{\pm 308}$). The 53-bit mantissa provides a precision equivalent to about 16 decimal digits.

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

Normalization: When the decimal point is placed to the right of the first (non-zero) significant digit, the number is said to be normalized. If a number is not normalized, it can be put in normalized form by shifting the binary point and adjusting the exponent. Figure 1.11 shows an unnormalized value, $0.0010110... \times 2^9$, and its normalized version, $1.0110... \times 2^6$. As computations proceed, a number that does not fall in the representable range of normal numbers might be generated.



(There is no implicit 1 to the left of the binary point.)

$$\text{Value represented} = +0.0010110... \times 2^9$$

(a) Unnormalized value



$$\text{Value represented} = +1.0110... \times 2^6$$

(b) Normalized version

Figure 1.11: Floating-point normalization in IEEE single-precision format.

Special Values: The end values 0 and 255 of the excess-127 exponent E' are used to represent special values. When $E'=0$ and the mantissa fraction M is zero, the value exact 0 is represented. When $E'=255$ and $M=0$, the value ∞ is represented, where ∞ is the result of dividing a normal number by zero.

When $E'=0$ and $M \neq 0$, *denormal* numbers are represented. Their value is $\pm 0.M=2^{-126}$. When $E'=255$ and $M \neq 0$, the value represented is called not a number (NaN). A NaN is the result of performing an invalid operation such as $0/0$ or $\sqrt{-1}$.

MEMORY LOCATIONS AND ADDRESSES

The memory consists of millions of storage *cells*, each of which can store a *bit* of information i.e. 0 or 1. The memory is organized into a group of n bits and each group of n bits is referred to as a *word* of information, and n is called the *word length*. The memory of a computer can be schematically represented as a collection of words, as shown in Figure 1.12.

The word lengths typically range from 16 to 64 bits. If the word length of a computer is 32 bits, a single word can store a 32-bit signed number or four ASCII-encoded characters, each occupying 8 bits, as shown in Figure 1.13. A unit of 8 bits is called a *byte*.

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

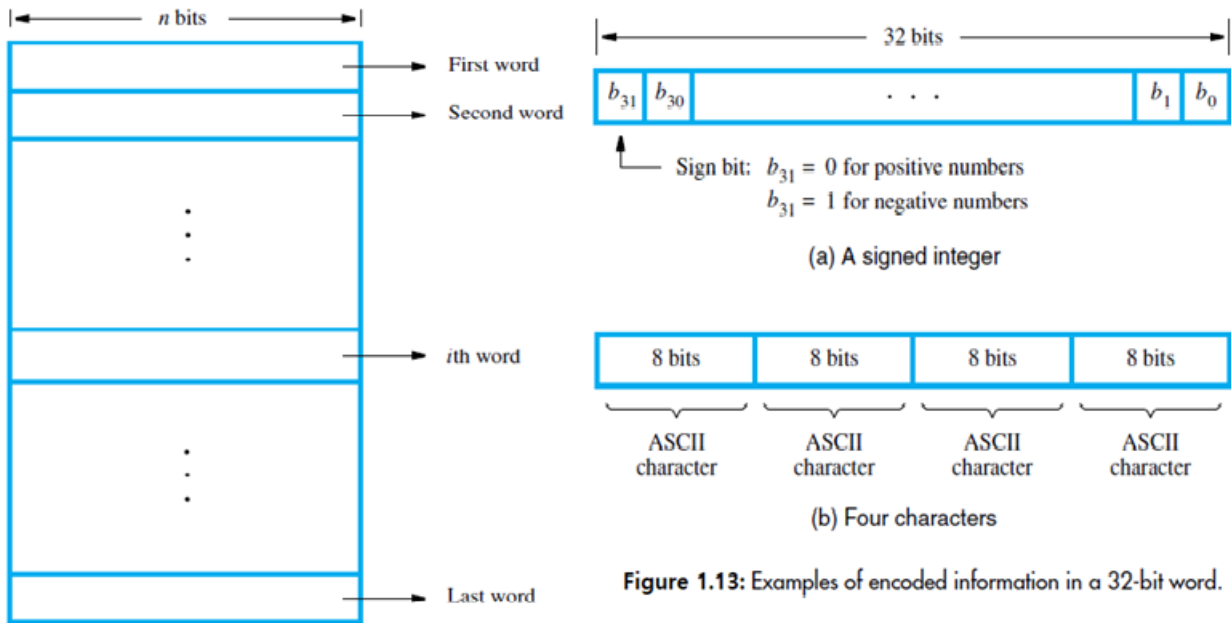


Figure 1.12: Memory words.

Figure 1.13: Examples of encoded information in a 32-bit word.

Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires distinct names or *addresses* for each location. It is customary to use numbers from 0 to $2k - 1$, for some suitable value of k , as the addresses of successive locations in the memory. Thus, the memory can have up to $2k$ addressable locations. The $2k$ addresses constitute the *address space* of the computer. For example, a 24-bit address generates an address space of 2^{24} (16,777,216) locations. This number is usually written as 16M, where 1M is the number 2^{20} (1,048,576).

Byte Addressability: A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits. It is impractical to assign distinct addresses to individual bit locations in the memory. The most practical assignment is to have successive addresses refer to successive byte locations in the memory. The term *byte-addressable memory* is used for this assignment. Byte locations have addresses 0, 1, 2, Thus, if the word length of the machine is 32 bits, successive words are located at addresses 0, 4, 8,, with each word consisting of four bytes.

Big-Endian and Little-Endian Assignments: There are two ways that byte addresses can be assigned across words, as shown in Figure 1.14. The name *big-endian* is used when lower byte addresses are used for the more significant bytes (the leftmost bytes) of the word. The name *little-endian* is used when the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word.

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

In both cases, byte addresses 0, 4, 8, . . . , are taken as the addresses of successive words in the memory of a computer with a 32-bit word length.

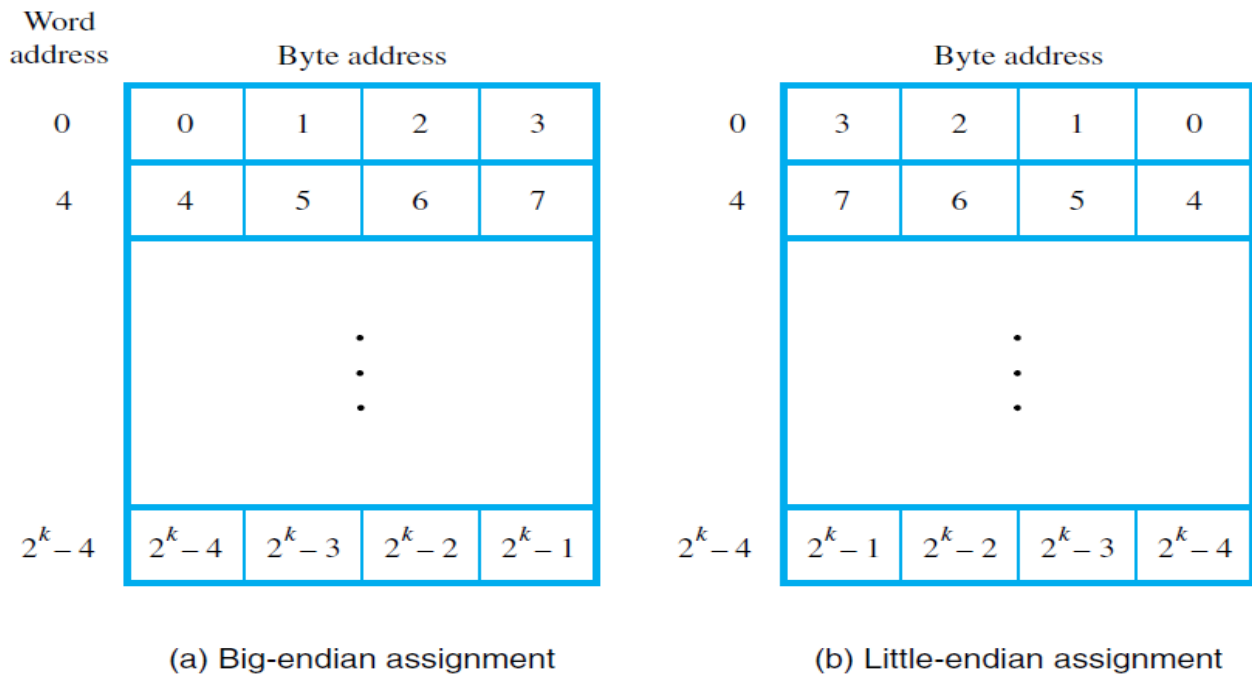


Figure 1.1.4: Byte and word addressing.

Word Alignment: The word locations is said to have *aligned* addresses if they begin at a byte address that is a multiple of the number of bytes in a word. For example, if the word length is 16 (2 bytes), aligned words begin at byte addresses 0, 2, 4, . . . , and for a word length of 64 (2^3 bytes), aligned words begin at byte addresses 0, 8, 16, Words are said to have *Unaligned Addresses*, if they begin at an arbitrary byte-address.

Accessing Numbers, Characters & Characters Strings: A number usually occupies one word. It can be accessed in the memory by specifying its word address. Similarly, individual characters can be accessed by their byte-address. To handle the strings of variable length, the length of the string is indicated in two ways. A special control character with the meaning "end of string" can be used as the last character in the string or a separate memory word location or register can contain a number indicating the length of the string in bytes.

MEMORY OPERATIONS

The two basic operations are: *Load (Read or fetch)* and *Store (Write)*.

The Load operation transfers a copy of the contents of a specific memory location to the processor. The memory contents remain unchanged. To start a Read operation,

the processor sends the address of the desired location to the memory and requests that its contents be read. The memory reads the data stored at that address and sends them to the processor.

The Store operation transfers an item of information from the processor to a specific memory location, overwriting the former contents of that location. To initiate a Write operation, the processor sends the address of the desired location to the memory, together with the data to be written into that location. The memory then uses the address and data to perform the write.

INSTRUCTIONS AND INSTRUCTION SEQUENCING

A computer must have instructions capable of performing four types of operations:

- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

Register Transfer Notation: Possible locations that may be involved in the transfer of information from one location in a computer to another are memory locations, processor registers, or registers in the I/O subsystem. The names that represent the addresses of memory locations may be LOC, PLACE, A, or VAR2. Predefined names for the processor registers may be R0 or R5. Registers in the I/O subsystem may be identified by names such as DATAIN or OUTSTATUS. To describe the transfer of information, the contents of any location are denoted by placing square brackets around its name.

For example, $R2 \leftarrow [LOC]$ means the contents of memory location LOC are transferred into processor register R2. $R4 \leftarrow [R2] + [R3]$ means the contents of registers R2 and R3 are added and transferred into register R4. This type of notation is known as *Register Transfer Notation* (RTN).

Assembly-Language Notation: To represent machine instructions and programs, *assembly language* is used. For example, the transfer of contents from memory location LOC to processor register R2, is specified by the statement

Move LOC, R2

Similarly, two numbers contained in processor registers R2 and R3 and placing their sum in R4 can be specified by the assembly-language statement as

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

Add R4, R2, R3

Basic instruction types: Consider the high level language statement

$$C \leftarrow [A] + [B] \dots\dots\dots (1)$$

The above action can be accomplished by different machines instructions as follows:

Three address instruction: In three address instruction, above equation (1) can be represented as:

Add A, B, C

Add the contents of memory-locations A & B. Then, place the result into location C. Operands A and B are called *source operands*, C is called *destination operand*, and Add is the operation to be performed on operands. The syntax of three address instruction is:

Operation Source1, Source2, Destination

Two address instruction: The syntax of two address instruction is:

Operation Source, Destination

The above addition operation of equation (1) can be represented as:

Move B, C

Add A, C

In 1st instruction, the contents of B is moved to C and in the 2nd instruction the contents of A is added with contents of C and then the sum is stored in C.

Another example for two address instruction is Add A, B. Add the contents of memory-locations A & B. Then, place the result into location B, replacing the original contents of this location. Operand B is both a source and a destination.

One address instruction: The syntax of one address instruction is:

Operation Source/Destination

Example: Add A ; Add contents of memory location A to contents of accumulator register A & place sum back into accumulator.

Load A ; Copy contents of memory location A into accumulator.

Store C ; Copy the contents of the accumulator into location C.

The above addition operation of equation (1) can be represented as:

Load A

Add B

Store C

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

Access to data in the registers is much faster than to data stored in memory-locations. Use of registers allows faster processing & results in shorter instructions, registers are used to store data temporarily in the processor during the processing.

Let R_i represent a general-purpose register. The instructions: *Load A, R_i* & *Store R_i, A* & *Add A, R_i* are generalizations of the Load, Store and Add Instructions for the single-accumulator case, in which register R_i performs the function of the accumulator.

In processors, where arithmetic operations are allowed only on operands that are in registers, the task $C \leftarrow [A] + [B]$ can be performed by the instruction sequence:

Move A, R_i

Move B, R_j

Add R_i, R_j

Move R_j, C

Instruction Execution and Straight-Line Sequencing: Consider the task $C \leftarrow [A] + [B]$. Figure 1.15 shows a possible program segment for this task as it appears in the memory of a computer. The three instructions of the program are in successive word locations, starting at location i . Since each instruction is 4 bytes long, the second and third instructions are at addresses $i + 4$ and $i + 8$.

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

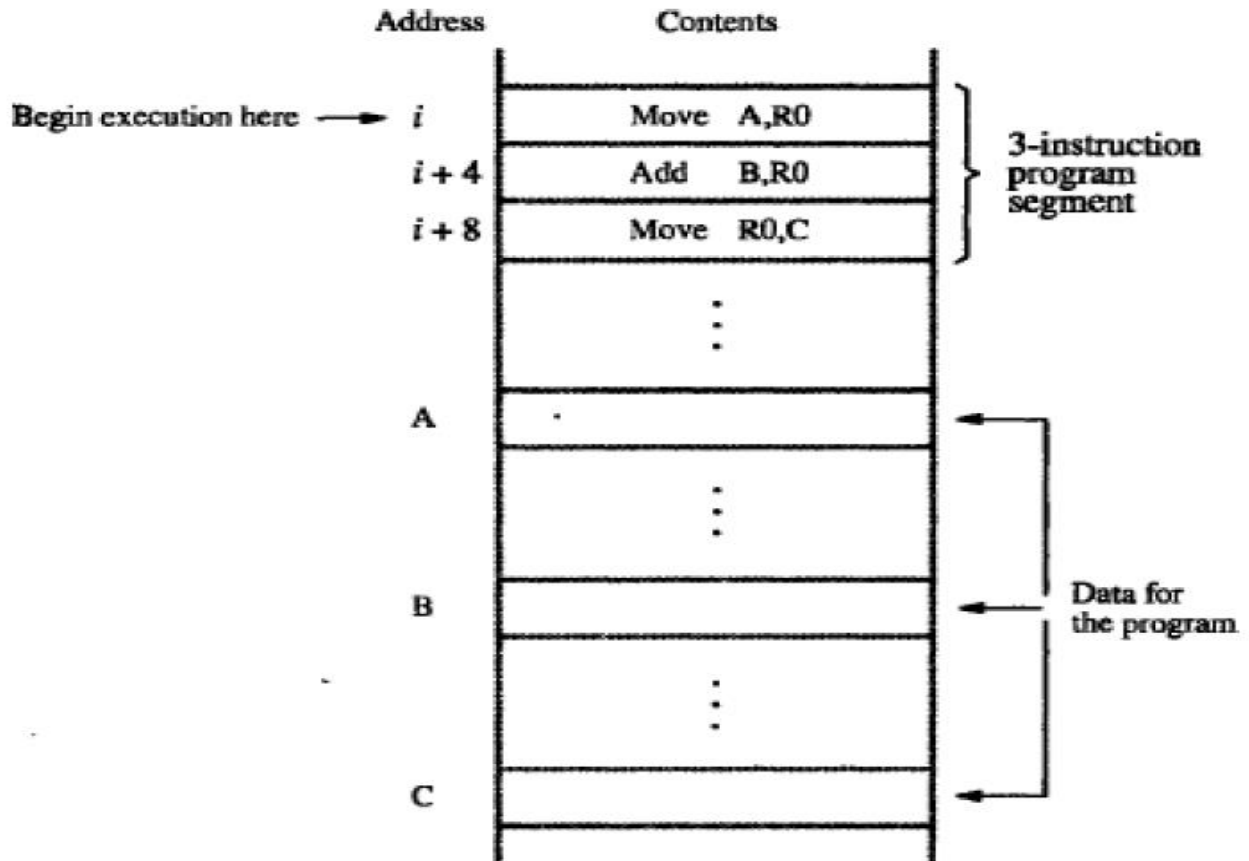


Figure 1.15: A program for $C \leftarrow [A] + [B]$.

The program is executed as follows:

- ❑ The address of the first instruction (i) is placed into the Program Counter (PC).
- ❑ Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called *straight-line sequencing*.
- ❑ During the execution of each instruction, the PC is incremented by 4 to point to the next instruction.

There are 2 phases for Instruction Execution:

- 1) Fetch Phase: the instruction is fetched from the memory location whose address is in the PC. This instruction is placed in Instruction Register (IR) in the processor.
- 2) Execute Phase: The contents of IR is examined to determine which operation is to be performed. The specified-operation is then performed by the processor.

This involves a small number of steps such as fetching operands from the memory or from processor registers, performing an arithmetic or logic operation, and storing the result in the destination location. At some point during this two-

phase procedure, the contents of the PC are advanced to point to the next instruction. When the execute phase of an instruction is completed, the PC contains the address of the next instruction, and a new instruction fetch phase can begin.

Consider the task of adding a list of n numbers shown in Figure 1.16. The addresses of the memory locations containing the n numbers are symbolically given as NUM1, NUM2, . . . NUM n , and separate Add instructions is used to add each number to the contents of register R0. After all the numbers have been added, the result is placed in memory location SUM.

Branching: Instead of using a long list of Add instructions, as in Figure 1.16, it is possible to implement a program loop in which the instructions read the next number in the list and add it to the current sum. To add all numbers, the loop has to be executed as many times as there are numbers in the list. Figure 1.17 shows the structure of the desired program. The body of the loop is a straight-line sequence of instructions executed repeatedly. It starts at location LOOP and ends at the instruction Branch>0. During each pass through this loop, the address of the next list entry is determined, and that entry is fetched and added to R0.

Assume that the number of entries in the list, n , is stored in memory location N, as shown. Register R1 is used as a counter to determine the number of times the loop is executed. Hence, the contents of location N are loaded into register R1 at the beginning of the program. Then, within the body of the loop, the instruction “Decrement R1” reduces the contents of R1 by 1 each time through the loop. Execution of the loop is repeated as long as the result of decrement operation is greater than zero. Then *Branch Instruction* loads a new value into the program counter. As a result, the processor fetches and executes the instruction at this new address called the *Branch Target*.

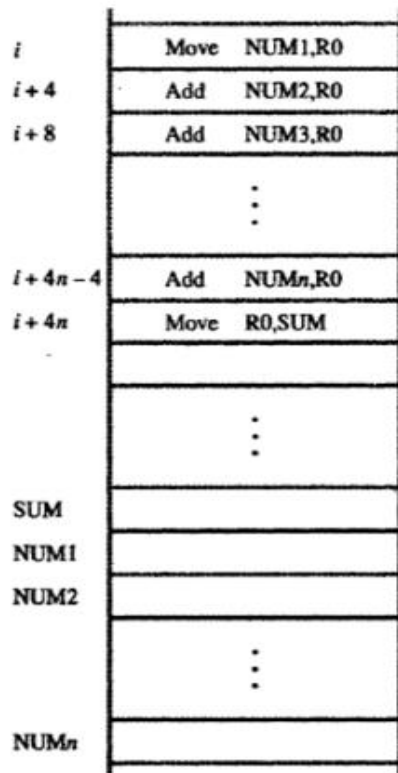


Figure 1.16: A straight-line program for adding n numbers.

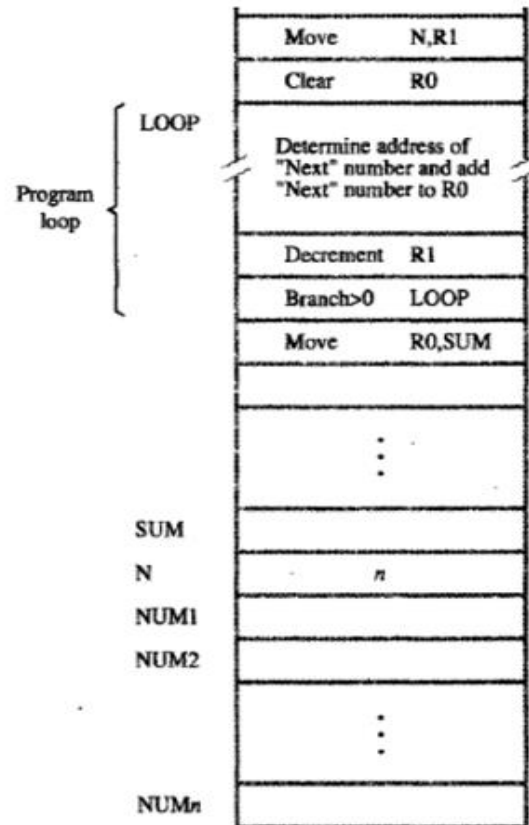


Figure 1.17: Using a loop to add n numbers.

A *conditional branch instruction* causes a branch only if a specified condition is satisfied. If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed. In Figure 1.17, the instruction “Branch>0 LOOP” is a conditional branch instruction that causes a branch to location LOOP if the content of register R1 is greater than zero. This means that the loop is repeated as long as there are entries in the list that are yet to be added to R0. At the end of the n th pass through the loop, the Decrement instruction produces a value of zero in R2, and, hence, branching does not occur. Instead, Move instruction is fetched and executed. It moves the final result from R0 into memory location SUM.

Condition Codes: The processor keeps track of information about the results of various operations. This is accomplished by recording the required information in individual bits, called *Condition Code Flags*. These flags are grouped together in a special processor-register called the condition code register (or status register). Individual condition code flags are set to 1 or cleared to 0, depending on the outcome of the operation performed.

Four commonly used flags are:

COMPUTER ORGANIZATION & ARCHITECTURE – BEC306C

- 1) N (negative) set to 1 if the result is negative; otherwise cleared to 0.
- 2) Z (zero) set to 1 if the result is 0; otherwise, cleared to 0.
- 3) V (overflow) set to 1 if arithmetic overflow occurs; otherwise, cleared to 0.
- 4) C (carry) set to 1 if a carry-out results from the operation; otherwise cleared to 0.

The N and Z flags indicate whether the result of an arithmetic or logic operation is negative or zero. The V flag indicates whether overflow has taken place. The processor sets the V flag to allow the programmer to test whether overflow has occurred and branch to an appropriate routine that corrects the problem.

The C flag is set to 1 if a carry occurs from the MSB position during an arithmetic operation. This flag makes it possible to perform arithmetic operations on operands that are longer than the word length of the processor.

ASSIGNMENT QUESTIONS

1. Write a note on Types of Computers.
2. Describe the functional units of a computer with neat block diagram.
3. With a neat diagram, explain the basic operational concepts of a computer highlighting the role of PC, IE, MAR & MDR.
4. Illustrate single bus structure of a computer.
5. List the functions of system software in computer.

**COMPUTER ORGANIZATION &
ARCHITECTURE – BEC306C**

6. What is Operating system? Explain user program and OS routine sharing the processor.
7. How to measure the performance of a computer? Explain.
8. Explain computer basic performance equation. / Explain the methods to improve the performance of Computer.
9. Write a note on Processor clock.
10. List out and explain the three systems used for representing signed numbers.
11. Discuss IEEE standard for single precision and double precision floating point numbers with standard notations.
12. Explain Little-endian and Big-endian byte address assignment
13. Illustrate Instruction and Instruction sequencing with an example.
14. Explain the 3-address, 2-address and 1-address instruction with an example.
15. With diagram, explain the concept of branching in program execution.
16. Write a short note on condition code flags.
17. The number of instructions to be executed in a program is 1000 and average number of steps needed / machine instruction is 20. Calculate the performance of the system assuming clock rate = 800 MHz.