# MODULE 1: BASIC STRUCTURE OF COMPUTERS

**BASIC CONCEPTS**

• **Computer Architecture (CA)** is concerned with the structure and behaviour of the computer.
• CA includes the information formats, the instruction set and techniques for addressing memory.
• In general covers, CA covers 3 aspects of computer-design namely: 1) Computer Hardware, 2) Instruction set Architecture and 3) Computer Organization.

**1. Computer Hardware**

➢ It consists of electronic circuits, displays, magnetic and optical storage media and communication facilities.

**2. Instruction Set Architecture**

➢ It is programmer visible machine interface such as instruction set, registers, memory organization and exception handling.
➢ Two main approaches are 1) CISC and 2) RISC.
(CISC→Complex Instruction Set Computer, RISC→Reduced Instruction Set Computer)

**3. Computer Organization**

➢ It includes the high level aspects of a design, such as
→ memory-system
→ bus-structure &
→ design of the internal CPU.

➢ It refers to the operational units and their interconnections that realize the architectural specifications.
➢ It describes the function of and design of the various units of digital computer that store and process information.

*****************************************************************

**Q. With a neat diagram describe the functional units of a computer. Give few examples for I/O devices (6 M)**
**Q.Explain the operation of computer with neat block diagram (10 M)**

**FUNCTIONAL UNITS**

• A computer consists of 5 functionally independent main parts:
1) Input
2) Memory
3) ALU
4) Output &
5) Control units.

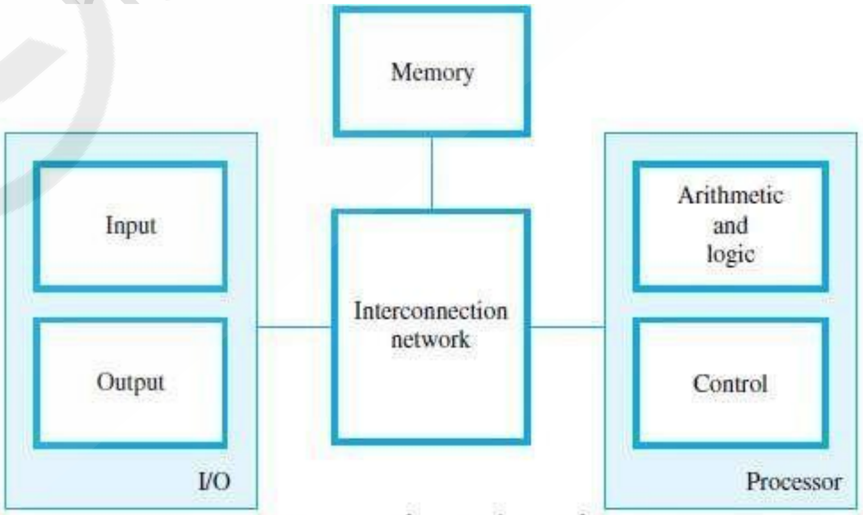


**Figure 1.1** Basic functional units of a computer.

**Input unit:-** Computers accept coded information through input units, which read the data. Whenever a key is pressed, corresponding character or number is translated into its equivalent binary code over a cable & fed either to memory or processor.
**Ex:** Keyboard, Joysticks, trackballs, mouse, scanners etc are other input devices.

**Output unit:-** Its basic function is to send the processed results to the outside world.
Examples:- Printer, speakers, monitor etc.

**Memory unit: -** Its function is to store programs and data. It is basically of two types
1. Primary memory 2. Secondary memory

1. **Primary memory: -** The programs which has to be executed must be stored in this memory. Each memory location stores data in it. Each memory location is identified by its address.
Examples:- RAM, ROM, CACHE etc..

2 **Secondary memory: -** It is used where large amounts of data & programs have to be stored, particularly information that is accessed infrequently.
Examples: - Magnetic disks & tapes, optical disks (ie CD-ROM's), floppies etc.,

**Arithmetic logic unit (ALU):-** This unit performs arithmetic operations like addition, subtraction, division, multiplication and logical operations like not, and, or, xor etc. on the operands(data) that are brought into the ALU from memory. Then according to the instructions the operation is performed in the required sequence.
**Control unit:-** The actual timing signals that govern the transfer of data between input unit, processor, memory and output unit are generated by the control unit.
*****************************************************************

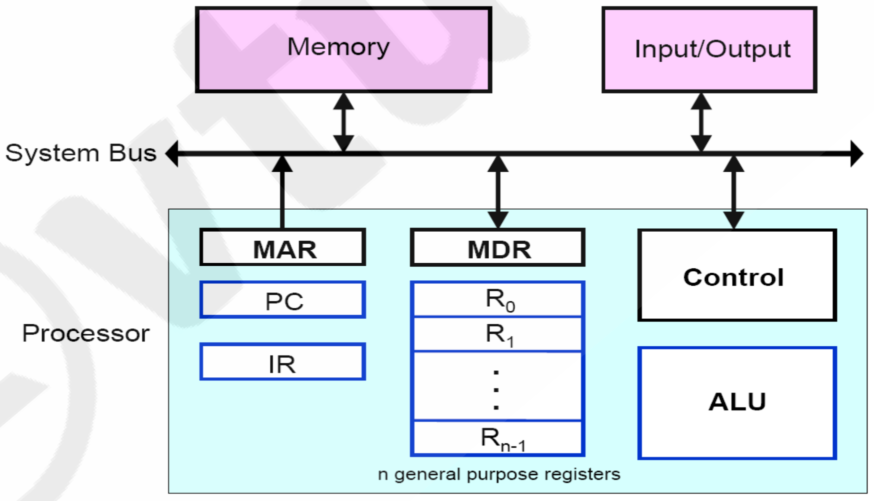
**BASIC OPERATIONAL CONCEPTS**
• An Instruction consists of 2 parts, 1) Operation code (Opcode) and 2) Operands.

| | |
|---|---|
| *OPCODE* | *OPERANDS* |

• The data/operands are stored in memory.
• The individual instruction are brought from the memory to the processor.
• Then, the processor performs the specified operation.
• Let us see a typical instruction
*ADD LOCA, R0*
• This instruction is an addition operation. The following are the steps to execute the instruction: Step 1: Fetch the instruction from main-memory into the 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 same instruction can be realized using 2 instructions as:
*Load LOCA, R1*
*Add R1, R0*
• The following are the steps to execute the instruction:
Step 1: Fetch the instruction from main-memory into the processor.
Step 2: Fetch the operand at location LOCA from main-memory into the register R1. Step 3: Add the content of Register R1 and the contents of register R0.
Step 4: Store the result (sum) in R0.
*****************************************************************
**Q. With a neat diagram, discuss the operational concepts in a computer highlight the role of PC, MAR, MDR and IR – ( 08 M)**

**Various Processor Registers**
- The **processor** contains ALU, control-circuitry and many registers.
- The processor has ‘n“ general-purpose registers $\mathbf{R_0}$ through $\mathbf{R_{n-1}}$.
- The **IR (Instruction Register)** holds the instruction that is currently being executed.
- The **control-unit** generates the timing-signals that determine when a given action is to take place.
- The **PC( Program Counter)** 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.
- The **MAR** (Memory Address Register) holds the address of the memory-location to be accessed.
- The **MDR**(Memory Data Register) contains the data to be written into or read out of the addressed location.

**STEPS TO EXECUTE AN INSTRUCTION between processor and memory**
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, it's 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.



***********************************************************************

Q. Explain single bus and multiple bus structure in computer (06 M)

**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: 1) Single Bus Structure and 2) Multiple Bus Structure.

**1) Single Bus Structure**
- ➢ 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:**
  1) Low cost &
  2) Flexibility for attaching peripheral devices.

**2) Multiple Bus Structure**

- Systems that contain multiple buses achieve more concurrency in operations.
- Two or more transfers can be carried out at the same time.
- **Advantage:** Better performance.
- **Disadvantage:** Increased cost.

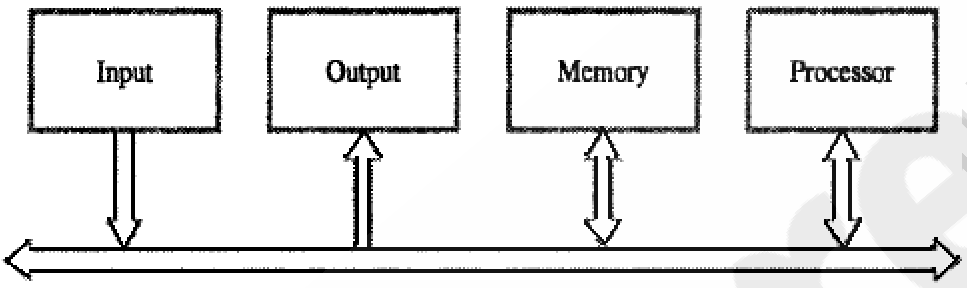


**Figure 1.3** Single-bus structure.

- 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.
  → prevent a high-speed processor from being locked to a slow I/O device during data transfers.

*******************************************************************

## Memory mapped I/O

It is an usual assumption that the addresses issued by the processor to access instructions and operands always refer to memory locations. Many computers use an arrangement called memory mapped I/O in which some memory address values are used to refer to peripheral device buffer registers, such as DATAIN and DATAOUT. Thus no special instructions are needed to access the contents of these registers; data can be transferred between these registers and the processor using instructions such as Move, Load or Store.
For Example, the contents of the keyboard character buffer DATAIN can be transferred to register R1 in the processor by the instruction
MoveByte    DATAIN,R1
Similarly, the contents of register R1 can be transferred to DATAOUT by the instruction
MoveByte  R1, DATAOUT
The status flags SIN and SOUT are automatically cleared when the buffer registers DATAIN and DATAOUT are referenced, respectively. The MoveByte operation code signifies that the operand size is a byte, to distinguish it from the operation code Move that has been used for word operands.

*******************************************************************

**Q. Explain system Software functions in computer(06M).**

- In order for a user to enter and run an application program, the computer must already contain some system software in its memory.
- System software is a collection of programs that are executed as needed to perform functions such as
- Receiving and interpreting user commands
- Running standard application programs such as word processors, etc, or games
- Managing the storage and retrieval of files in secondary storage devices
- 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

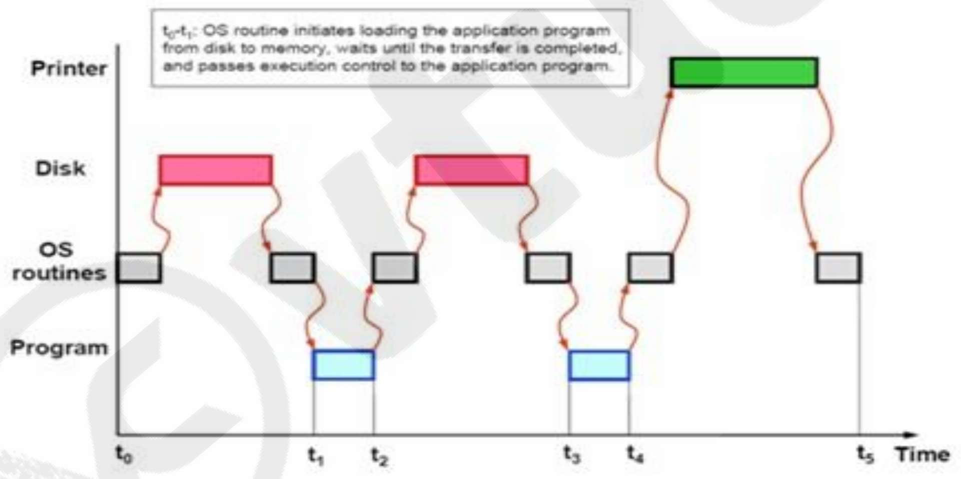
**Q. What is operating system? Explain user program and OS routine sharing the processor (08 M).**

- Operating system (OS)
  - This is a large collection of routines, that controls the sharing and interaction among various computer units as they perform application programs
- The OS routines perform the tasks required to assign computer resource 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
- In the diagram, a system with one processor, one disk, and one printer is given to explain the basics of OS
  - Assume that part of the program's task involves reading a data file from the disk into the memory, performing some computation on the data, and printing the results.

In the diagram shown, the above procedure as per to the time slots is explained.

i) In the time period to – t1, **Operating system loads the application program from disk to memory.**
ii) In the time period t1 – t2 **program is executed**
iii) t2-t3 **program accesses disk**
iv) t3 – t4 **program executes some more**
v) t4-t5 **program accesses printer**
vi) At t5 **program terminates**

**User Program and OS Routine Sharing**



$t_0$-$t_1$: OS routine initiates loading the application program from disk to memory, waits until the transfer is completed, and passes execution control to the application program.
Printer
Disk
OS routines
Program
$t_0$
$t_1$
$t_2$
$t_3$
$t_4$
$t_5$
Time

**************************************************************************

**PERFORMANCE**

- The most important measure of performance of a computer is how quickly it can execute programs.
- The speed of a computer is affected by the design of
  1) Instruction-set.
  2) Hardware & the technology in which the hardware is implemented.
  3) Software including the operating system.
- Because programs are usually written in a HLL(HLL→ High Level Language), performance is also affected by the compiler that translates programs into machine language.

• For best performance, it is necessary to design the compiler, machine instruction set and hardware in a co-ordinated way.

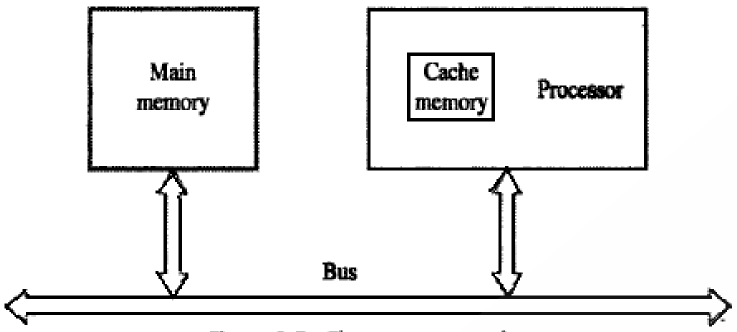


**Figure 1.5** The processor cache.

• Let us examine the flow of program instructions and data between the memory & the processor.
• At the start of execution, all program instructions are stored in the main-memory.
• As execution proceeds, instructions are fetched into the processor, and a copy is placed in the cache.
• Later, if the same instruction is needed a second time, it is read directly from the cache.

*************************************************************************

**Q. Explain in brief different types of key parameters that affect the processor performance (05 M)**

## Factors for improving or affecting the performance

### 1. 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.
• Let P = Length of one clock
cycle R = Clock rate.
• Relation between P and R is given by

$$R = \frac{1}{P}$$

• R is measured in cycles per second.
• Cycles per second is also called Hertz (Hz)

### 2. BASIC PERFORMANCE EQUATION (**Q. Explain computer basic performance equation -04 M)**

• Let T = Processor time required to executed a program.
• N = Actual number of instruction executions.
S = Average number of basic steps needed to execute one machine instruction.
R = Clock rate in cycles per second.
• The program execution time is given by

$$T = \frac{N \times S}{R} \quad \text{--------(1)}$$

• Equ1 is 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 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.
  - ➢ The value of R can be increased by using a higher frequency clock.
• Care has to be taken while modifying values since changes in one parameter may affect the other.

**3. CLOCK RATE**

• There are 2 possibilities for increasing the clock rate R:

1) Improving the IC technology makes logic-circuits faster.
2) Reducing the amount of processing done in one basic step also reduces the clock period P.

**4. Compiler:**

- A compiler translates a high-level language program into a sequence of machine instructions.
- To reduce N, we need a suitable machine instruction set and a compiler that makes good use of it.
- Goal is to reduce N×S

  N= is the actual number of instruction execution

  S= is the total number of basic steps needed to execute one machine cycle instruction.

**5. Pipelining and superscalar operation.**

– Pipelining: by overlapping the execution of successive instructions

– Superscalar: different instructions are concurrently executed with multiple instruction pipelines. This means that multiple functional units are needed. This means that multiple functional units are used creating parallel paths through which different instructions can be executed in parallel with such an arrangement, it becomes possible to start the execution of several instructions in every clock cycle. This mode of operation is called superscalar execution.

**6. CISC and RISC**

➢ It is much easier to implement efficient pipelining in processor with simple instruction sets

- Reduced Instruction Set Computers (RISC)
- Complex Instruction Set Computers (CISC)

**INSTRUCTION SET: CISC AND RISC**

| **RISC** | **CISC** |
|---|---|
| Simple instructions taking one cycle. | Complex instructions taking multiple cycle. |
| Instructions are executed by hardwired control unit. | Instructions are executed by microprogrammed control unit. |
| Few instructions. | Many instructions. |
| Fixed format instructions. | Variable format instructions. |
| Few addressing modes, and most instructions have register to register addressing mode. | Many addressing modes. |
| Multiple register set. | Single register set. |
| Highly pipelined. | No pipelined or less pipelined. |

*********************************************************************

**Problem 1:**

List the steps needed to execute the machine instruction:

*Add R2, LOCA*

in terms of transfers between the components of processor and some simple control commands. Assume that the address of the memory-location containing this instruction is initially in register PC.

**Solution:**

1. Transfer the contents of register PC (address of this instruction) to register MAR.
2. Issue a Read command to memory.

   And, then wait until the instruction is transferred into register MDR.

3. Transfer the instruction from MDR into IR(Instruction Register) and decode it.
4. Transfer the address LOCA from IR to MAR.
5. Issue a Read command and wait until MDR is loaded with data present at LOCA address.
6. Transfer contents of MDR to the ALU.
7. Transfer contents of R2 to the ALU.
8. Perform addition of the two operands in the ALU and transfer result at location LOCA.
9. Transfer contents of PC to ALU.
10. Add 1 to operand in ALU and transfer incremented address to PC.

*********************************************************************

Problem 2:

List the steps needed to execute the machine instruction:

*Add R4, R2, R3*

in terms of transfers between the components of processor and some simple control commands. Assume that the address of the memory-location containing this instruction is initially in register PC.

**Solution:**

1. Transfer the contents of register PC to register MAR.
2. Issue a Read command to memory.
   And, then wait until the instruction is transferred into register MDR.
3. Transfer the instruction from MDR into IR and decode it.
4. Transfer contents of R1 and R2 to the ALU.
5. Perform addition of two operands in the ALU and transfer answer into R3.
6. Transfer contents of PC to ALU.
7. Add 1 to operand in ALU and transfer incremented address to PC.

*********************************************************************

**Problem 3:**

(a) Given a short sequence of machine instructions for the task "Add the contents of memory-location A to those of location B, and place the answer in location C". Instructions:

Load LOC, Ri and
Store R*i*, LOC

are the only instructions available to transfer data between memory and the general purpose registers. Add instructions do not change contents of either location A or B.

(b) Suppose that Move and Add instructions are available with the formats:

Move Location1, Location2 and
Add Location1, Location2

These instructions move or add a copy of the operand at the second location to the first location, overwriting the original operand at the first location. Either or both of the operands can be in the memory or the general-purpose registers. Is it possible to use fewer instructions of these types to accomplish the task in part (a)? If yes, give the sequence.

**Solution:**

(a)

```
Load    A,    R0
Load    B,    R1
Add R0, R1
Store R1, C
```

(b) Yes;

```
Move B, C
Add A, C
```

*********************************************************************

**Problem 4:**

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*S)/R= (250*4+400*5+350*3)/1X$10^9$ =(1000+2000+1050)/1*$10^9$= 4.05 µs.
**********************************************************************

**Problem 5:**
For the following processor, obtain the performance.
Clock rate = 800 MHz
No. of instructions executed = 1000
Average no of steps needed / machine instruction = 20

**Solution:**

$T = \frac{N \times S}{R}$ = (1000*20)/800 * $10^6$=25 micro sec or 25*$10^{-6}$ sec

**********************************************************************

# MODULE 1 (CONT.): MACHINE INSTRUCTIONS & PROGRAMS

Computers are built using logic circuits that operate on information in the form of binary digit whose values are 0 and 1. The most natural way to represent a number in a computer system is by a string of bits, called a binary number. A text character can also be represented by a string of bits called a character code.

**Numeric Data Representation**

**Fixed Point**

| Decimal | Binary | Octal | Hexadecimal |
|---|---|---|---|
| 00 | 0000 | 00 | 0 |
| 01 | 0001 | 01 | 1 |
| 02 | 0010 | 02 | 2 |
| 03 | 0011 | 03 | 3 |
| 04 | 0100 | 04 | 4 |
| 05 | 0101 | 05 | 5 |
| 06 | 0110 | 06 | 6 |
| 07 | 0111 | 07 | 7 |
| 08 | 1000 | 10 | 8 |
| 09 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

**Representation:**
It's the representation for integers only where the decimal point is always fixed. i.e at the end of rightmost point. it can be again represented in two ways.

1. **Sign and Magnitude Representation:**
In this system, the most significant (leftmost) bit in the word as a sign bit. If the sign bit is 0, the number is positive; if the sign bit is 1, the number is negative.
The simplest form of representing sign bit is the sign magnitude representation.
One of the draw back for sign magnitude number is addition and subtraction need to consider both sign of the numbers and their relative magnitude.
Another drawback is there are two representation for 0(Zero) i.e +0 and -0.

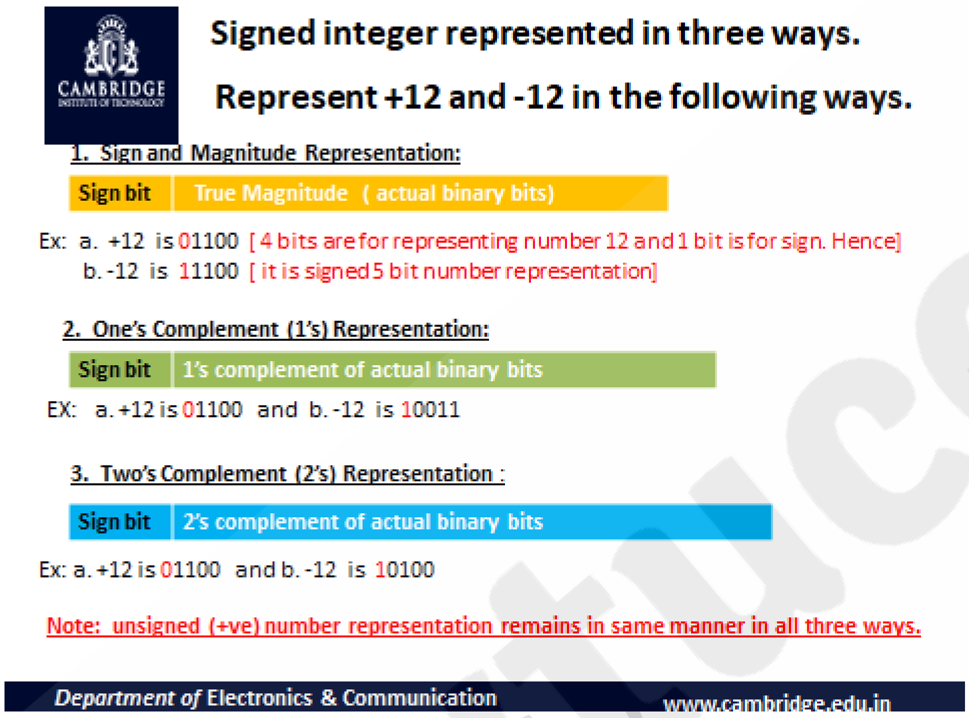
2. **One's Complement (1's) Representation:**
In this representation negative values are obtained by complementing each bit of the corresponding positive number.

For example 1s complement of 0101 is 1010 . The process of forming the 1s complement of a given number is equivalent to subtracting that number from 2n -1 i.e from 1111 for 4 bit number.

**Two's Complement (2's) Representation** Forming the 2's complement of a number is done by subtracting that number from 2n . So 2's complement of a number is obtained by adding 1 to 1s complement of that number.

Ex: 2's complement of 0101 is 1010 +1 = 1011

**NOTE:** In all systems, the leftmost bit is 0 for positive number and 1 for negative number.



## Signed integer represented in three ways.

## Represent +12 and -12 in the following ways.

1. Sign and Magnitude Representation:

| Sign bit | True Magnitude ( actual binary bits) |
|---|---|

Ex: a. +12 is 01100 [ 4 bits are for representing number 12 and 1 bit is for sign. Hence]
b. -12 is 11100 [ it is signed 5 bit number representation]

2. One's Complement (1's) Representation:

| Sign bit | 1's complement of actual binary bits |
|---|---|

EX: a. +12 is 01100 and b. -12 is 10011

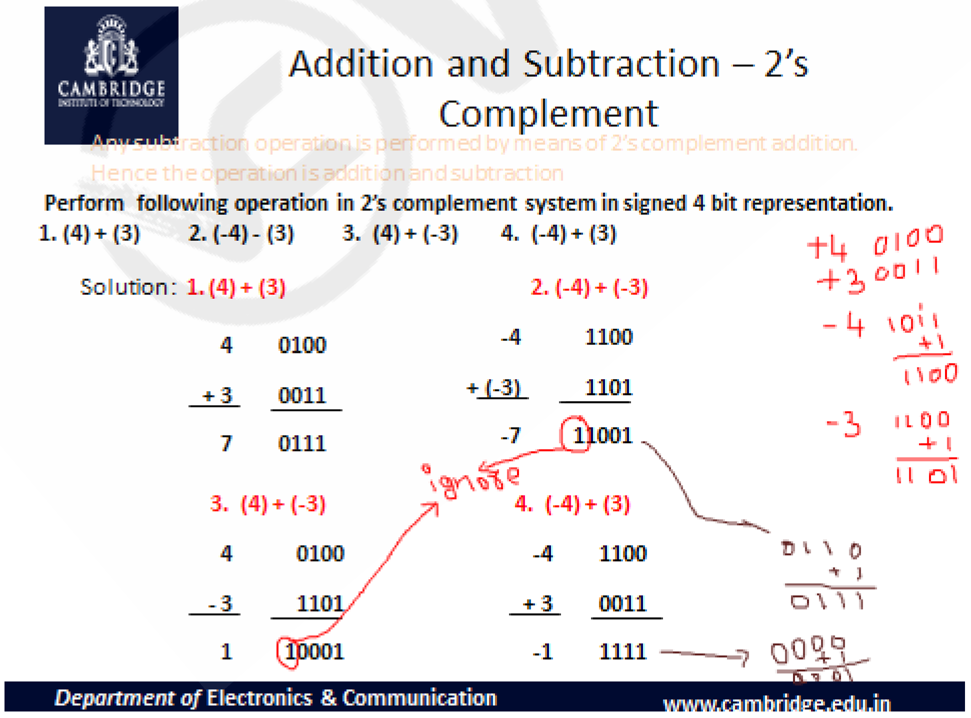
3. Two's Complement (2's) Representation :

| Sign bit | 2's complement of actual binary bits |
|---|---|

Ex: a. +12 is 01100 and b. -12 is 10100

Note: unsigned (+ve) number representation remains in same manner in all three ways.

*Department of* Electronics & Communication www.cambridge.edu.in



## Addition and Subtraction – 2's Complement

Any subtraction operation is performed by means of 2's complement addition.
Hence the operation is addition and subtraction

Perform following operation in 2's complement system in signed 4 bit representation.

1. (4) + (3)  2. (-4) - (3)  3. (4) + (-3)  4. (-4) + (3)

Solution: 1. (4) + (3)

```
   4    0100
 + 3    0011
   7    0111
```

2. (-4) + (-3)

```
  -4     1100
+(-3)    1101
  -7    11001
```

3. (4) + (-3)

```
   4     0100
  -3     1101
   1    10001
```

ignore

4. (-4) + (3)

```
  -4    1100
  +3    0011
  -1    1111
```

```
+4  0100
+3  0011
-4  1011
       +1
    1100
-3  1100
       +1
    1101
```

```
0110
  +1
0111
```

```
0000
  +1
0001
```

*Department of* Electronics & Communication www.cambridge.edu.in

**Floating-point representation:**

Floating-point numbers are so called as the decimal or binary point floats over the base depending on the exponent value. It consists two components.

- Exponent
- Mantissa

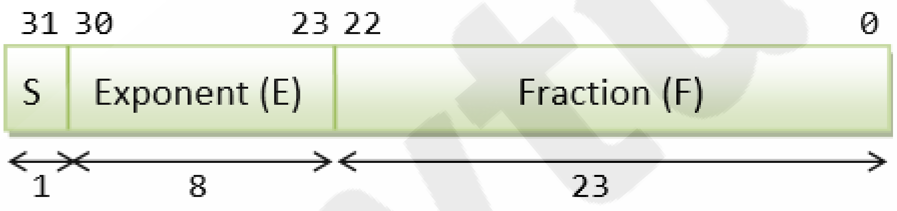
Table - 1 Precision Representation

| Precision | Base | Sign | Exponent | Significand |
|---|---|---|---|---|
| Single precision | 2 | 1 | 8 | 23+1 |
| Double precision | 2 | 1 | 11 | 52+1 |

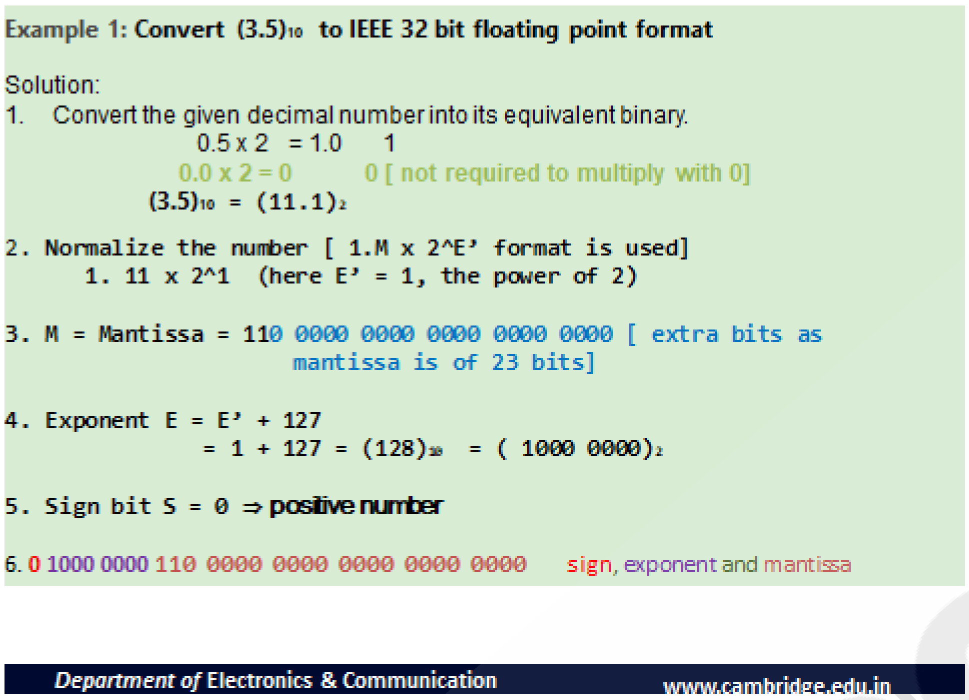
## *IEEE-754 32-bit Single-Precision Floating-Point Numbers*

In 32-bit single-precision floating-point representation:

- The most significant bit is the *sign bit* (S), with 0 for positive numbers and 1 for negative numbers.
- The following 8 bits represent *exponent* (E).
- The remaining 23 bits represents *fraction* (F).



32-bit Single-Precision Floating-point Number



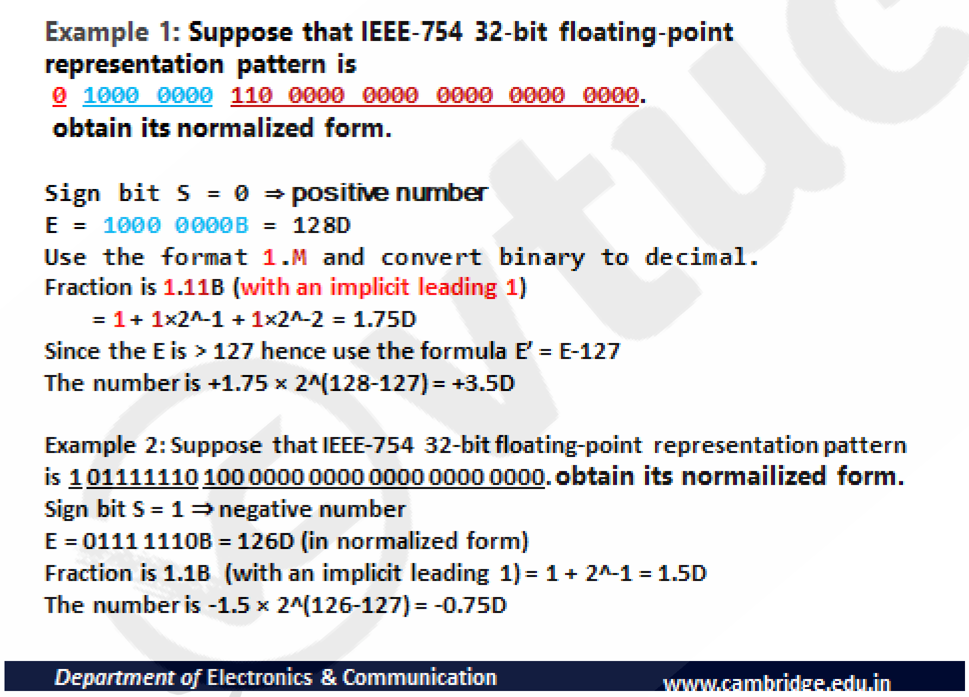
**Example 1: Convert $(3.5)_{10}$ to IEEE 32 bit floating point format**

Solution:

1. Convert the given decimal number into its equivalent binary.

```
            0.5 x 2  = 1.0      1
            0.0 x 2 = 0         0 [ not required to multiply with 0]
        (3.5)10 = (11.1)2
```

```
2. Normalize the number [ 1.M x 2^E' format is used]
      1. 11 x 2^1  (here E' = 1, the power of 2)

3. M = Mantissa = 110 0000 0000 0000 0000 0000 [ extra bits as
                  mantissa is of 23 bits]

4. Exponent  E = E' + 127
                = 1 + 127 = (128)10  = ( 1000 0000)2

5. Sign bit S = 0 ⇒ positive number

6. 0 1000 0000 110 0000 0000 0000 0000 0000     sign, exponent and mantissa
```



**Example 1: Suppose that IEEE-754 32-bit floating-point representation pattern is**
<u>0</u> <u>1000 0000</u> <u>110 0000 0000 0000 0000 0000</u>.
**obtain its normalized form.**

Sign bit S = 0 ⇒ positive number
E = 1000 0000B = 128D
Use the format 1.M and convert binary to decimal.
Fraction is 1.11B (with an implicit leading 1)
= 1 + 1×2^-1 + 1×2^-2 = 1.75D
Since the E is > 127 hence use the formula E' = E-127
The number is +1.75 × 2^(128-127) = +3.5D

Example 2: Suppose that IEEE-754 32-bit floating-point representation pattern is <u>1</u> <u>01111110</u> <u>100 0000 0000 0000 0000 0000</u>. **obtain its normailized form.**
Sign bit S = 1 ⇒ negative number
E = 0111 1110B = 126D (in normalized form)
Fraction is 1.1B (with an implicit leading 1) = 1 + 2^-1 = 1.5D
The number is -1.5 × 2^(126-127) = -0.75D

## Binary, Signed-Integer Representations

| *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 2.1. Binary, signed-integer representations.

## 2's-Complement Add and Subtract Operations

```
(a)     0010   (+2)          (b)      0100   (+4)
      + 0011   (+3)                 + 1010   (- 6)
        ----                          ----
        0101   (+5)                   1110   (- 2)

(c)     1011   (- 5)         (d)      0111   (+7)
      + 1110   (- 2)                + 1101   (- 3)
        ----                          ----
        1001   (- 7)                  0100   (+4)

(e)     1101   (- 3)                  1101
      - 1001   (- 7)     ⇒          + 0111
                                      ----
                                      0100   (+4)

(f)     0010   (+2)                   0010
      - 0100   (+4)      ⇒          + 1100
                                      ----
                                      1110   (- 2)

(g)     0110   (+6)                   0110
      - 0011   (+3)      ⇒          + 1101
                                      ----
                                      0011   (+3)

(h)     1001   (- 7)                  1001
      - 1011   (- 5)     ⇒          + 0101
                                      ----
                                      1110   (- 2)

(i)     1001   (- 7)                  1001
      - 0001   (+1)      ⇒          + 1111
                                      ----
                                      1000   (- 8)

(j)     0010   (+2)                   0010
      - 1101   (- 3)     ⇒          + 0011
                                      ----
                                      0101   (+5)
```

Figure 2.4. 2's-complement Add and Subtract operations.

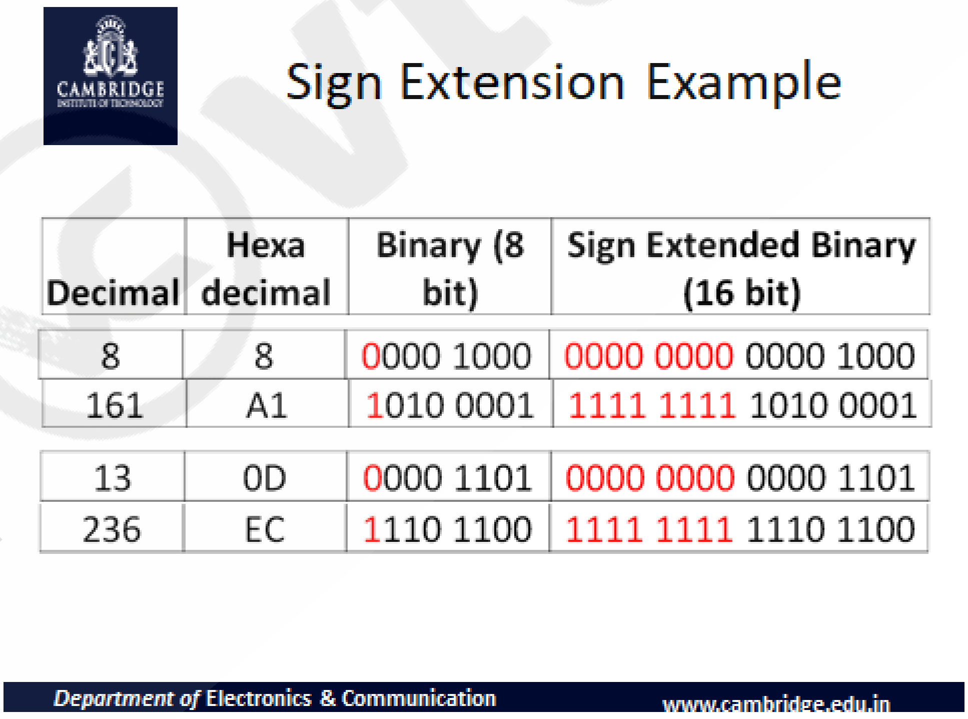
## Overflow Conditions

```
          0111                       1000
 5         0101             -7        1001
 3         0011             -2        1100
-8         1000              7       10111

Overflow                    Overflow

          0000                       1111
 5         0101             -3        1101
 2         0010             -5        1011
 7         0111             -8       11000

No overflow                 No overflow
```

**Overflow when carry-in to the high-order bit does not equal carry out**

Overflow occurs in following conditions:
1. 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.

A simple way to detect overflow is to examine the signs of the two summands X and Y and the sign of the result. When both operands X and Y have the same sign, an overflow occurs when the sign of S is not the same as the signs of X and Y.



## Sign Extension Example

| Decimal | Hexa decimal | Binary (8 bit) | Sign Extended Binary (16 bit) |
|---|---|---|---|
| 8 | 8 | 0000 1000 | 0000 0000 0000 1000 |
| 161 | A1 | 1010 0001 | 1111 1111 1010 0001 |
| 13 | 0D | 0000 1101 | 0000 0000 0000 1101 |
| 236 | EC | 1110 1100 | 1111 1111 1110 1100 |

*Department of* Electronics & Communication www.cambridge.edu.in

**MEMORY-LOCATIONS & ADDRESSES**

• **Memory** consists of many millions of storage cells (flip-flops).
• Each cell can store a bit of information i.e. 0 or 1 (Figure 2.1).
• Each group of n bits is referred to as a **word** of information, and n is called the **word length**.
• The word length can vary from 8 to 64 bits.
• A unit of 8 bits is called a **byte**.
• Accessing the memory to store or retrieve a single item of information (word/byte) requires distinct addresses for each item location. (It is customary to use numbers from 0 through $2^k$-1 as the addresses of successive-locations in the memory).
• If $2^k$ = no. of addressable locations;
then $2^k$ addresses constitute the address-space of the computer.
For example, a 24-bit address generates an address-space of $2^{24}$ locations (16 MB).

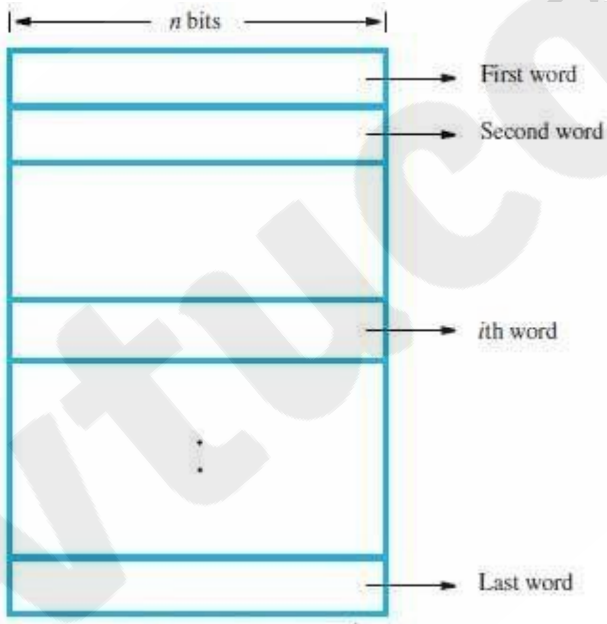


**Figure 2.1** Memory words.

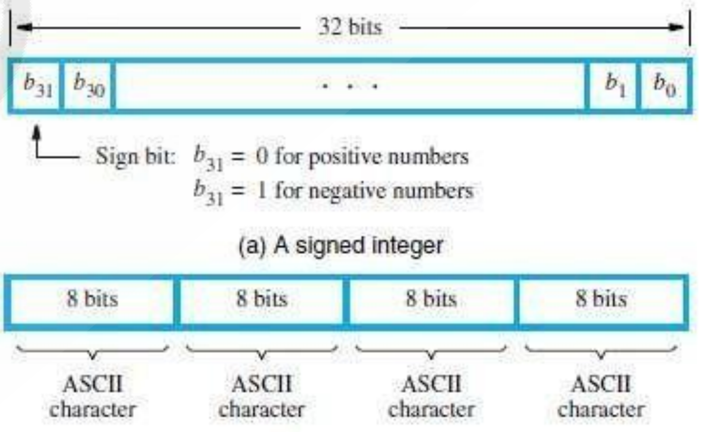


**Figure 2.2** Examples of encoded information in a 32-bit word.

## BYTE-ADDRESSABILITY

• In byte-addressable memory, successive addresses refer to successive byte locations in the memory.
• Byte locations have addresses 0, 1, 2. . . . .
• If the word-length is 32 bits, successive words are located at addresses 0, 4, 8. . with each word having 4 bytes.

## BIG-ENDIAN & LITTLE-ENDIAN ASSIGNMENTS

• There are two ways in which byte-addresses are arranged (Figure 2.3).
  1) **Big-Endian:** Lower byte-addresses are used for the more significant bytes of the word.
  2) **Little-Endian:** Lower byte-addresses are used for the less significant bytes of the word
• In both cases, byte-addresses 0, 4, 8. .........are taken as the addresses of successive words in the memory.

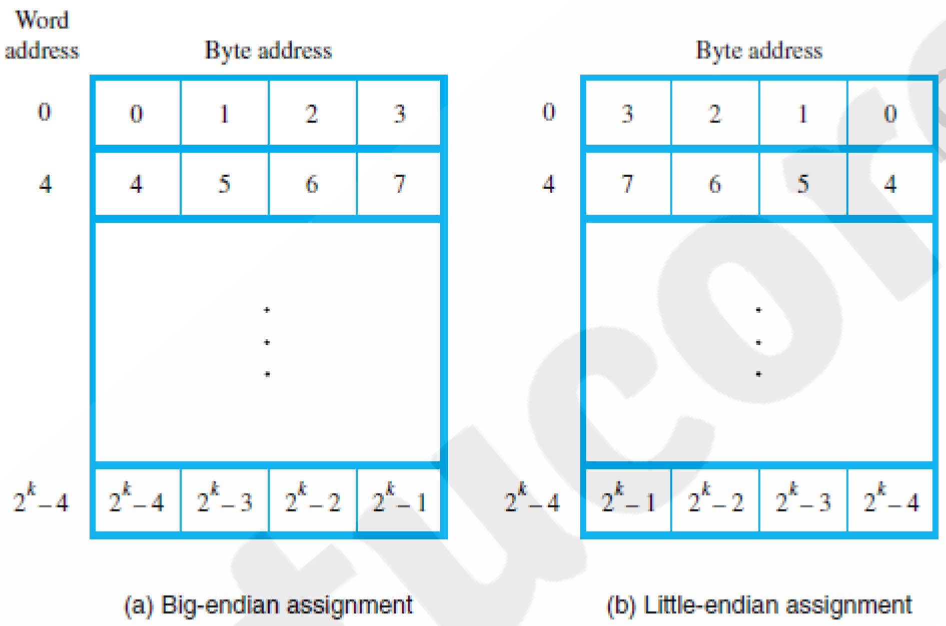

Word address
Byte address
0
4
$2^k-4$
0 1 2 3
4 5 6 7
$2^k-4$ $2^k-3$ $2^k-2$ $2^k-1$
(a) Big-endian assignment
Byte address
0
4
$2^k-4$
3 2 1 0
7 6 5 4
$2^k-1$ $2^k-2$ $2^k-3$ $2^k-4$
(b) Little-endian assignment

**Figure 2.3** Byte and word addressing.

• Consider a 32-bit integer (in hex): 0**x**12345678 which consists of 4 bytes: 12, 34, 56, and 78.
  ➢ Hence this integer will occupy 4 bytes in memory.
  ➢ Assume, we store it at memory address starting 1000.
  ➢ On little-endian, memory will look like

| Address | Value |
|---|---|
| 1000 | 78 |
| 1001 | 56 |
| 1002 | 34 |
| 1003 | 12 |

  ➢ On big-endian, memory will look like

| Address | Value |
|---|---|
| 1000 | 12 |
| 1001 | 34 |
| 1002 | 56 |
| 1003 | 78 |

## WORD ALIGNMENT

• Words are said to be **Aligned** in memory 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 . . . . .
  ➢ If the word length is 64(2 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.

• There are two ways to indicate the length of the string:

1) A special control character with the meaning "end of string" can be used as the last character in the string.
2) A separate memory word location or register can contain a number indicating the length of the string in bytes.

**MEMORY OPERATIONS**

• Two memory operations are:

1) Load (Read/Fetch) &
2) Store (Write).

• The **Load** operation transfers a copy of the contents of a specific memory-location to the processor. The memory contents remain unchanged.

• Steps for Load operation:

1) Processor sends the address of the desired location to the memory.
2) Processor issues „read" signal to memory to fetch the data.
3) Memory reads the data stored at that address.
4) Memory sends the read data to the processor.

• The **Store** operation transfers the information from the register to the specified memory-location. This will destroy the original contents of that memory-location.

• Steps for Store operation are:

1) Processor sends the address of the memory-location where it wants to store data.
2) Processor issues „write" signal to memory to store the data.
3) Content of register(MDR) is written into the specified memory-location.

**INSTRUCTIONS & INSTRUCTION SEQUENCING**

• A computer must have instructions capable of performing 4 types of operations:

1) Data transfers between the memory and the registers (MOV, PUSH, POP, XCHG).
2) Arithmetic and logic operations on data (ADD, SUB, MUL, DIV, AND, OR, NOT).
3) Program sequencing and control (CALL.RET, LOOP, INT).
4) I/0 transfers (IN, OUT).

## REGISTER TRANSFER NOTATION (RTN)

• The possible locations in which transfer of information occurs are: 1) Memory-location 2) Processor register & 3) Registers in I/O device.

| Location | Hardware Binary Address | Example | Description |
|---|---|---|---|
| Memory | LOC, PLACE, NUM | R1 ← [LOC] | Contents of memory-location LOC are transferred into register R1. |
| Processor | R0, R1 ,R2 | [R3] ← [R1]+[R2] | Add the contents of register R1 &R2 and places their sum into R3. |
| I/O Registers | DATAIN, DATAOUT | R1 ← DATAIN | Contents of I/O register DATAIN are transferred into register R1. |

## ASSEMBLY LANGUAGE NOTATION

• To represent machine instructions and programs, assembly language format is used.

| Assembly Language Format | Description |
|---|---|
| Move LOC, R1 | Transfer data from memory-location LOC to register R1. The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten. |
| Add R1, R2, R3 | Add the contents of registers R1 and R2, and places their sum into register R3. |

## BASIC INSTRUCTION TYPES

| Instruction Type | Syntax | Example | Description | Instructions for Operation C<-[A]+[B] |
|---|---|---|---|---|
| Three Address | Opcode Source1,Source2,Destination | Add A,B,C | Add the contents of memory-locations A & B. Then, place the result into location C. | |
| Two Address | Opcode Source, Destination | 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. | Move B, C<br>Add A, C |
| One Address | Opcode Source/Destination | Load A | Copy contents of memory-location A into accumulator. | Load A<br>Add B<br>Store C |
| | | Add B | Add contents of memory-location B to contents of accumulator register & place sum back into accumulator. | |
| | | Store C | Copy the contents of the accumulator into location C. | |
| Zero Address | Opcode [no Source/Destination] | Push | Locations of all operands are defined implicitly. The operands are stored in a pushdown stack. | Not possible |

• Access to data in the registers is much faster than to data stored in memory-locations.

• Let Ri represent a general-purpose register. The instructions: *Load A,Ri*
*Store Ri,A*
*Add A,Ri*

are generalizations of the Load, Store and Add Instructions for the single-accumulator case, in which register Ri performs the function of the accumulator.

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

*Move A,Ri*
*Move B,Rj*
*Add Ri,Rj*
*Move Rj,C*

**INSTRUCTION EXECUTION & STRAIGHT LINE SEQUENCING**

• The program is executed as follows:

1) Initially, the address of the first instruction is loaded into PC (Figure 2.8).

**2)** 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*.

3) During the execution of each instruction, PC is incremented by 4 to point to next instruction.

• There are 2 phases for Instruction Execution:

**1) Fetch Phase:** The instruction is fetched from the memory-location and placed in the IR.

**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.

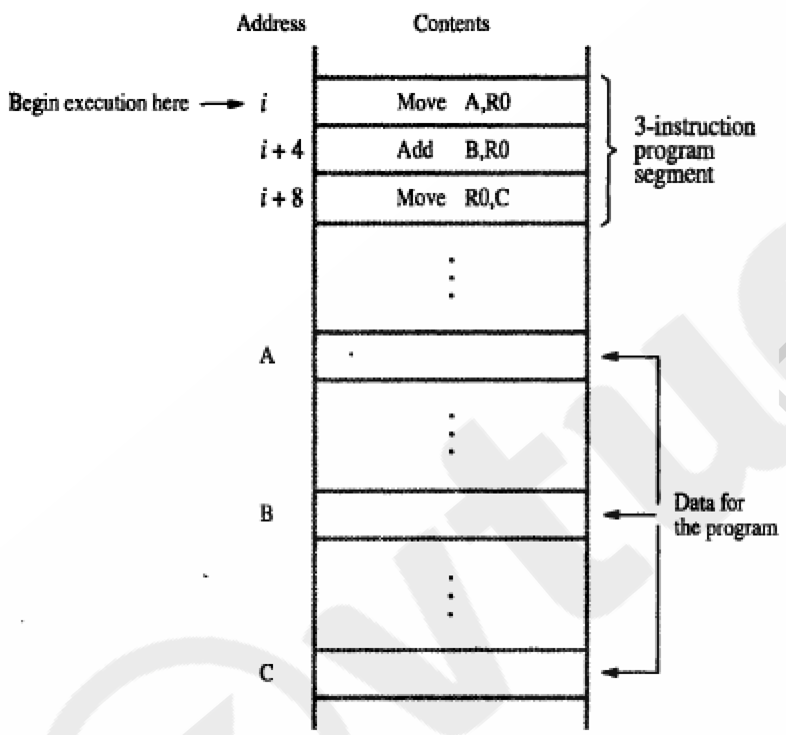


Figure 2.8 A program for C ← [A] + [B].

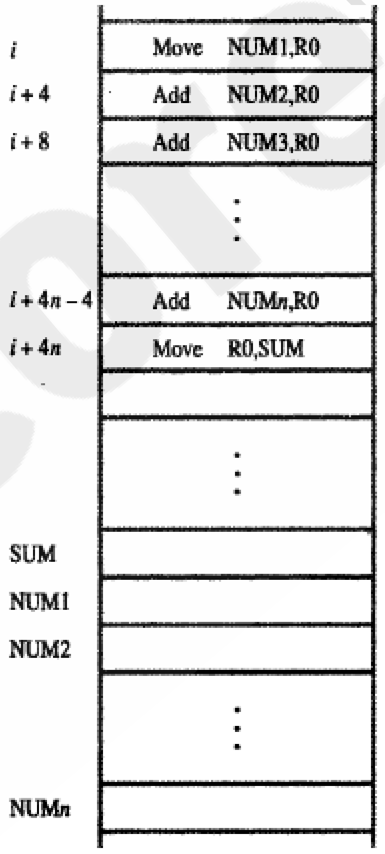


Figure 2.9 A straight-line program for adding n numbers.

**Program Explanation**

• Consider the program for adding a list of n numbers (Figure 2.9).

• The Address of the memory-locations containing the n numbers are symbolically given as NUM1, NUM2…..NUMn.

• Separate Add instruction 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**
• Consider the task of adding a list of „n" numbers (Figure 2.10).
• Number of entries in the list „n" is stored in memory-location **N**.
• Register **R1** is used as a counter to determine the number of times the loop is executed.
• Content-location N is loaded into register R1 at the beginning of the program.
• The **Loop** is a straight line sequence of instructions executed as many times as needed. The loop starts at location LOOP and ends at the instruction Branch>0.
• During each pass,
→ address of the next list entry is determined and
→ that entry is fetched and added to R0.
• The instruction *Decrement R1* reduces the contents of R1 by 1 each time through the loop.
• 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**.
• 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.

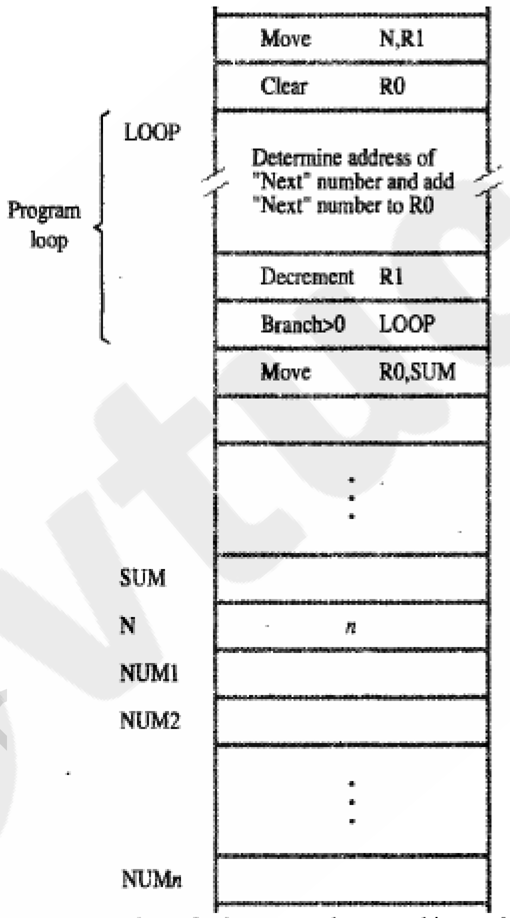


Figure 2.10 Using a loop to add *n* numbers.

**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 statue register).
• Four commonly used flags are:
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.