

UNIT - 1

Syllabus: Basic Structure of Computers: Basic Organization of Computers, Historical Perspective, Bus Structures, Data Representation: Data types, Complements, Fixed Point Representation. Floating, Point Representation. Other Binary Codes, Error Detection Codes. Computer Arithmetic: Addition and Subtraction, Multiplication Algorithms, Division Algorithms.

Computer: It is an electronic device designed to accept inputs, process it at a high speed and display the result.

1. It describes the function and design of the various units of digital computer that store and process information.
2. It also deals with the units of computer that receive information from external sources and send computed results to external destinations.

Types of Languages: Just as humans use language to communicate, and different regions have different languages, computers also have their own languages that are specific to them. Different kinds of languages have been developed to perform different types of work on the computer. Basically, languages can be divided into two categories according to how the computer understands them.

➤ **Low-Level Languages:** A language that corresponds directly to a specific machine. Low-level computer languages are either machine codes or are very close to them. A computer cannot understand instructions given to it in high-level languages or in English. It can only understand and execute instructions given in the form of machine language i.e. binary. There are two types of low-level languages:

- **Machine Language:** a language that is directly interpreted into the hardware. Machine language is the lowest and most elementary level of programming language and was the first type of programming language to be developed. Machine language is basically the only language that a computer can understand and it is usually written in hex. It is represented inside the computer by a string of binary digits (bits) 0 and 1. The symbol 0 stands for the absence of an electric pulse and the 1 stands for the presence of an electric pulse. Since a computer is capable of recognizing electric signals, it understands machine language.

Advantages:

- Machine language makes fast and efficient use of the computer.
- It requires no translator to translate the code. It is directly understood by the computer.

Disadvantages:

- All operation codes have to be remembered
- All memory addresses have to be remembered.
- It is hard to amend or find errors in a program written in the machine language.
- **Assembly Language:** A slightly more user-friendly language that directly corresponds to machine language. Assembly language was developed to overcome some of the many inconveniences of machine language. This is another low-level but very important language in which operation codes and operands are given in the form of alphanumeric symbols instead of 0's and 1's.

These alphanumeric symbols are known as mnemonic codes and can combine in a maximum of five-letter combinations e.g. ADD for addition, SUB for subtraction, START, LABEL etc. Because of this feature, assembly language is also known as 'Symbolic Programming Language.'

Advantages:

- Assembly language is easier to understand and use as compared to machine language.
- It is easy to locate and correct errors.

- It is easily modified.

Disadvantages:

- Like machine language, it is also machine dependent/specific.
- Since it is machine dependent, the programmer also needs to understand the hardware.

➤ **High-Level Languages:** Any language that is independent of the machine. High-level computer languages use formats that are similar to English. The purpose of developing high-level languages was to enable people to write programs easily, in their own native language environment (English). High-level languages are basically symbolic languages that use English words and/or mathematical symbols rather than mnemonic codes. Each instruction in the high-level language is translated into many machine language instructions that the computer can understand.

Advantages:

- High-level languages are user-friendly.
- They are easier to learn.
- They are easier to maintain.
- A program written in a high-level language can be translated into many machine languages and can run on any computer.
- programs developed in a high-level language can be run on any computer text

Disadvantages: A high-level language has to be translated into the machine language by a translator, which takes up time

Computer Types: Basing capacity, technology used and performance of computer, they are classified into two types

- According to computational ability
- According to generation or Historical Perspective

According to computational ability (Based on Size, cost and performance):

There are mainly 4 types of computers. These include:

- a) Micro computers
- b) Mainframe computers
- c) Mini computers
- d) Super computer

a) Micro computers: -

Micro computers are the most common type of computers in existence today, whether at work in school or on the desk at home. These computers include:

- 1 Desktop computer
- 2 Personal digital assistants (more commonly known as PDA's)
- 3 Palmtop computers
- 4 Laptop and notebook computers

Micro computers were the smallest, least powerful and least expensive of the computers of the time. The first Micro computers could only perform one task at a time, while bigger computers ran multi-tasking operating systems, and served multiple users. Referred to as a personal computer or "desktop computer", Micro computers are generally meant to service one user (person) at a time. By the late 1990s, all personal computers run a multi-tasking operating system, but are still intended for a single user.

b) Mainframe Computers :-

The term Mainframe computer was created to distinguish the traditional, large, institutional computer intended to service multiple users from the smaller, single user machines. These computers are capable of handling and processing very large amounts of data easily and quickly. A mainframe speed is so fast that it is measured in millions of tasks per milliseconds (MTM). While other computers became smaller, Mainframe computers stayed large to maintain the ever growing memory capacity and speed.

Mainframe computers are used in large institutions such as government, banks and large corporations. These institutions were early adopters of computer use, long before personal computers were available to individuals. "Mainframe" often refers to computers compatible with the computer architectures established in the 1960's. Thus, the origin of the architecture also affects the classification, not just processing power.

c) Mini Computers / Workstation :-

Mini computers, or Workstations, were computers that are one step above the micro or personal computers and a step below mainframe computers. They are intended to serve one user, but contain special hardware enhancements not found on a personal computer. They run operating systems that are normally associated with mainframe computers, usually one of the variants of the UNIX operating system.

Type	Word length	Memory	Processing speed	Application
Super computer	64-96 bits	256MB	400-10000mips	Sophisticated Scientific problems, Weather forecasting, Aerodynamics, Atomic Research etc
Main Frame	48-64 bits	128mb	30-100mips	Large industries, banks, airlines, NGO's.

d) Super Computer:-

A Super computer is a specialized variation of the mainframe. Where a mainframe is intended to perform many tasks, a Super computer tends to focus on performing a single program of intense numerical calculations. Weather forecasting systems, Automobile design systems, extreme graphic generator for example, are usually based on super computers.

Mini	32bits	96mb	10-30mips	Interactive and multi user environment.
Micro	8-32 bits	64MB	1-5MIPS	General purpose calculations, Industrial Control, Office Automation, e.t.c

According to Generations of Computers or Historical Perspective:

The history of computer development is often referred to in reference to the different generations of computing devices. Each generation of computer is characterized by a major technological development that fundamentally changed the way computers operate, resulting in increasingly smaller, cheaper, more powerful and more efficient and reliable devices.

Read about each generation and the developments that led to the current devices that we use today.

a) First Generation (1940-1956): Vacuum Tubes

The first computers used vacuum tubes for circuitry and magnetic drums for memory, and were often enormous, taking up entire rooms. They were very expensive to operate and in addition to using a great deal of electricity, generated a lot of heat, which was often the cause of malfunctions.

First generation computers relied on machine language, the lowest-level programming language understood by computers, to perform operations, and they could only solve one problem at a time. Input was based on punched cards and paper tape, and output was displayed on printouts.

Example: The UNIVAC and ENIAC computers are examples of first-generation computing devices. The UNIVAC was the first commercial computer delivered to a business client, the U.S. Census Bureau in 1951.

b) Second Generation (1956-1963): Transistors:-

Transistors replaced vacuum tubes and ushered in the second generation of computers. The transistor was invented in 1947 but did not see widespread use in computers until the late 1950s. The transistor was far superior to the vacuum tube, allowing computers to become smaller, faster, cheaper, more energy-efficient and more reliable than their first-generation predecessors. Though the transistor still generated a great deal of heat that subjected the computer to damage, it was a vast improvement over the vacuum tube. Second-generation computers still relied on punched cards for input and printouts for output.

Second-generation computers moved from cryptic binary machine language to symbolic, or assembly, languages, which allowed programmers to specify instructions in words. High-level programming languages were also being developed at this time, such as early versions of COBOL and FORTRAN. These were also the first computers that stored their instructions in their memory, which moved from a magnetic drum to magnetic core technology.

The first computers of this generation were developed for the atomic energy industry.

c) Third Generation (1964-1971): Integrated Circuits

The development of the integrated circuit was the hallmark of the third generation of computers. Transistors were miniaturized and placed on silicon chips, called semiconductors, which drastically increased the speed and efficiency of computers.

Instead of punched cards and printouts, users interacted with third generation computers through keyboards and monitors and interfaced with an operating system, which allowed the device to run many different applications at one time with a central program that monitored the memory. Computers for the first time became accessible to a mass audience because they were smaller and cheaper than their predecessors.

d) Fourth Generation (1971-Present): Microprocessors

The microprocessor brought the fourth generation of computers, as thousands of integrated circuits were built onto a single silicon chip. What in the first generation filled an entire room could now fit in the palm of the hand. The Intel 4004 chip, developed in 1971, located all the components of the computer—from the central processing unit and memory to input/output controls—on a single chip.

In 1981 IBM introduced its first computer for the home user, and in 1984 Apple introduced the Macintosh. Microprocessors also moved out of the realm of desktop computers and into many areas of life as more and more everyday products began to use microprocessors.

As these small computers became more powerful, they could be linked together to form networks, which eventually led to the development of the Internet. Fourth generation computers also saw the development of GUIs, the mouse and handheld devices.

e) Fifth Generation (Present and Beyond): Artificial Intelligence

Fifth generation computing devices, based on artificial intelligence, are still in development, though there are some applications, such as voice recognition, that are being used today. The use of parallel processing and superconductors is helping to make artificial intelligence a reality. Quantum computation and molecular and nanotechnology will radically change the face of computers in years to come. The goal of fifth-generation computing is to develop devices that respond to natural language input and are capable of learning and self-organization.

Functional Units: Every Digital computer systems consist of five distinct functional units. These units are as follows:

- Input unit
- Central Processing unit
- Arithmetic logic unit
- Memory unit
- Out put unit

These units are interconnected by electrical cables to permit communication between them. A computer must receive both data and program statements to function properly and be able to solve problems. The method of feeding data and programs to a computer is accomplished by an input device. Computer input devices read data from a source, such as magnetic disks, and translate that data into electronic impulses for transfer into the CPU. Some typical input devices are a keyboard, a mouse, or a scanner. Central Processing Unit The brain of a computer system is the central processing unit (CPU). The CPU processes data transferred to it from one of the various input devices. It then transfers either an intermediate or final result of the CPU to one or more output devices. The CPU is the computing center of the system. It consists of a control section, an arithmetic-logic section, and an internal storage section (main memory). Each section within the CPU serves a specific function and has a particular relationship with the other sections within the CPU.

Input Unit: An input device is usually a keyboard or mouse, the input device is the conduit through which data and instructions enter a computer. The most common input device is the keyboard, which accepts letters, numbers, and commands from the user. Another important type of input device is the mouse, which lets you select options from on-screen menus. You use a mouse by moving it across a flat surface and pressing its buttons. A variety of other input devices work with personal computers, too: The trackball and touchpad are variations of the mouse and enable you to draw or point on the screen. The joystick is a swiveling lever mounted on a stationary base that is well suited for playing video games

Central processing unit (CPU):

The part of the computer that executes program instructions is known as the processor or central processing unit (CPU). In a microcomputer, the CPU is on a single electronic component, the microprocessor chip, within the system unit or system cabinet. The system unit also includes circuit boards, memory chips, ports and other components. A microcomputer system cabinet will also house disk drives, hard disks, etc., but these are considered separate from the CPU. This is principal part of any digital computer system, generally composed of control unit, and arithmetic-logic unit the ‘heart’ of the computer. It constitutes the physical heart of the entire computer system; to it is linked various peripheral equipment, including input/output devices and auxiliary storage units

Arithmetic-Logic Unit:

The arithmetic-logic section performs arithmetic operations, such as addition, subtraction, multiplication, and division. Arithmetic-Logic Unit usually called the ALU is a digital circuit that performs two types of operations— arithmetic and logical. Arithmetic operations are the fundamental mathematical operations consisting of addition, subtraction, multiplication and division. Logical operations consist of comparisons. That is, two pieces of data are compared to see whether one is equal to, less than, or greater than the other. The ALU is a fundamental building block of the central processing unit of a computer. .

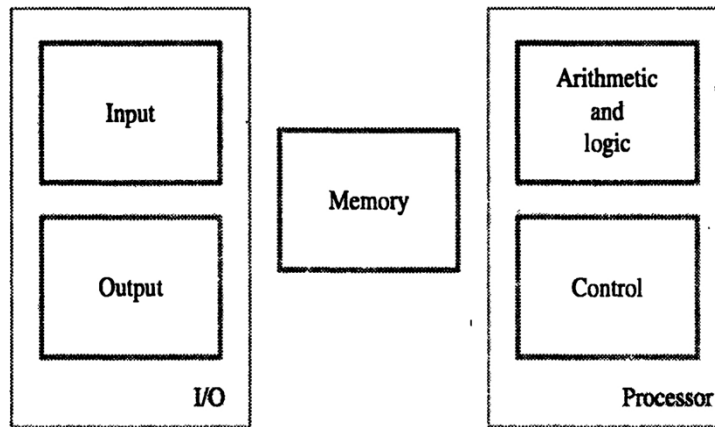


Figure 1.1 Basic functional units of a computer.

Memory unit: The function of the memory is to store programs and data. There are two classes of storage, called Primary and Secondary.

- **Primary storage:** It is a fast memory that operates at electronic speeds. Programs must stay in memory while they are being executed. The memory contains a large number of semiconductor storage cells, each capable of storing one bit of information.
- **Random access memory:** Memory in which any location can be reached in a short and fixed amount of time after specifying its address is called random access memory (RAM). The time required to access one word is called the memory **access time**.
- **Cache memory:** The small, fast, Ram units are called caches. They are tightly coupled with the processor and are often contained on the same integrated circuit chip to achieve high performance.
- **Main memory:** The largest and slowest units is referred to as the main memory.

Output Unit: An **output device** is any piece of computer hardware equipment used to communicate the results of data processing carried out by an information processing system (such as a computer) to the outside world. In computing, input/output, or I/O, refers to the communication between an information processing system (such as a computer), and the outside world. Inputs are the signals or data sent to the system, and outputs are the signals or data sent by the system to the outside.

Examples of output devices:

- Speaker
- Headphones
- Screen
- Printer

The Basic Operational Concepts of a Computer:

To perform a given task, an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which execute the specified operations. Data to be used as operands are also stored in the memory. The top level view of the computer is as follows.

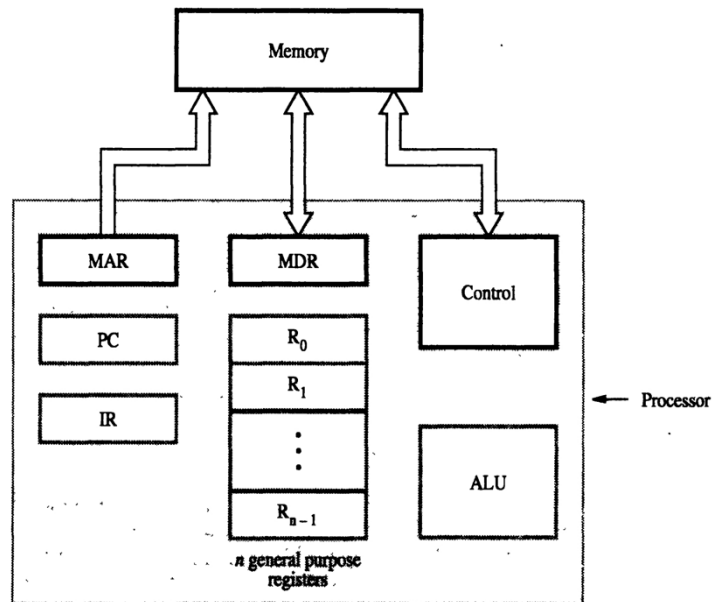


Figure 1.2 Connections between the processor and the memory.

Instruction register (IR):

1. The instruction register holds the instruction that is currently being executed.
2. Its output is available to the control circuits, which generate the timing signals that control the various processing elements involved in executing the instruction.

Program counter (PC):

1. The program counter is another specialized register.
2. It keeps track of the execution of a program.
3. It contains the memory address of the next instruction to be fetched and executed.
4. During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed.

Memory address register (MAR) & Memory data register(MDR):-

1. These two registers facilitate communication with the memory.
2. The MAR holds the address of the location to be accessed.
3. The MDR contains the data to be written into or read out of the addressed location.

Operating steps for Program execution:

1. Execution of the program (stored in memory) starts when the PC is set to point to the first instruction of the program.
2. The contents of the PC are transferred to the MAR and a Read control signal is sent to the memory.
3. After the time required to access the memory elapses, the addressed word is read out of the memory and loaded into the MDR. Next, the contents of the MDR are transferred to the IR. At this point, the instruction is ready to be decoded and executed.
4. If the instruction involves an operation to be performed by the ALU, it is necessary to obtain the required operands.
5. If an operand resides in memory (it could also be in a general purpose register in the processor), it has to be fetched by sending its address to the MAR and initiating a Read cycle.
6. When the operand has been read from the memory into the MDR, it is transferred from the MDR to ALU.
7. After one or more operands are fetched in this way, the ALU can perform the desired operation.
8. If the result of the 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 the execution of the current instruction, the contents of the PC are incremented so that the PC points to the next instruction to be executed.
11. Thus, as soon as the execution of the current instruction is completed, a new instruction fetch may be started.
12. In addition to transferring data between the memory and the processor, the computer accepts data from input devices and sends data to output devices. Thus, some machine instructions with the ability to handle I/O transfers are provided.

Bus Structures:

A bus, in computer terms, is simply a collection of wires over which information flows between two or more devices. The following are different types of busses:

1. Data Bus
2. Address Bus
3. Control Bus

The Data bus Carries data from one component (source) to other component (destination) connected to it. The data bus consists of 8, 16, 32 or more parallel signal lines. The data bus lines are bi-directional. This means that CPU can read data on these lines from memory or from a port, as well as send data out on these lines to a memory location.

The Address bus is the set of lines that carry information about where in memory the data is to be transferred to or from. It is an unidirectional bus. The address bus consists of 16, 20, 24 or more parallel signal lines. On these lines CPU sends out the address of the memory location.

The Control Bus carries the Control and timing information. Including these three the following are various types of busses. They are

System Bus: A System Bus is usually a combination of address bus, data bus, and control bus respectively.

Internal Bus: The bus that operates only with the internal circuitry of the CPU.

External Bus: Buses which connects computer to external devices is nothing but external bus.

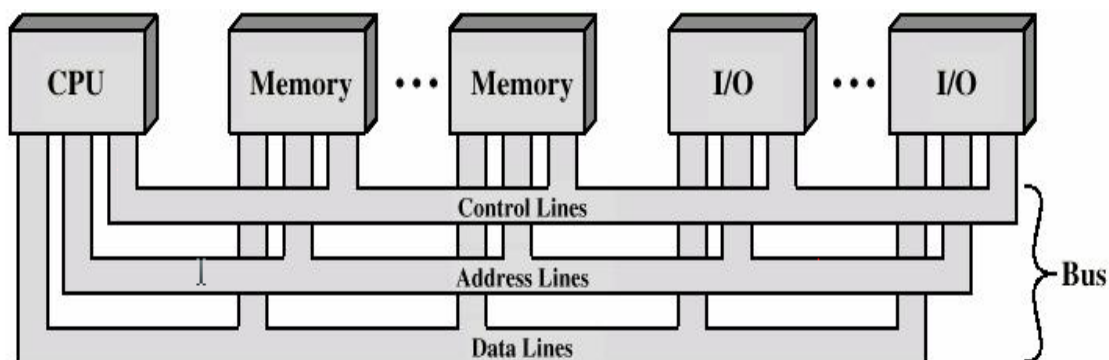
Back Plane: A Back Plane bus includes a row pf connectors into which system modules can be plugged in.

I/O Bus: The bus used by I/O devices to communicate with the CPU is usually reffered as I/O bus.

Synchronous Bus: While using Synchronous bus, data transmission between source and destination units takes place in a given timeslot which is already known to these units.

Asynchronous Bus: In this case the data transmission is governed by a special concept. That is handshaking control signals.

The Bus interconnection Scheme:-



Single bus structure :-

1. A group of lines(wires) that serves as a connecting path for several devices of a computer is called a bus.
2. In addition to the lines that carry the data, the bus must have lines for address and control purposes.
3. The simplest way to interconnect functional units is to use a single bus, as shown below.
4. All units are connected to this bus. Because the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time.
5. Bus control lines are used to arbitrate multiple requests for use of the bus.

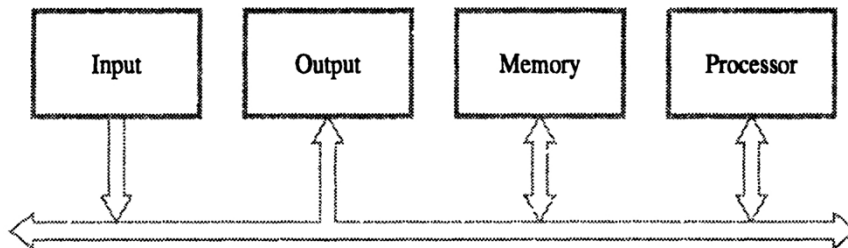
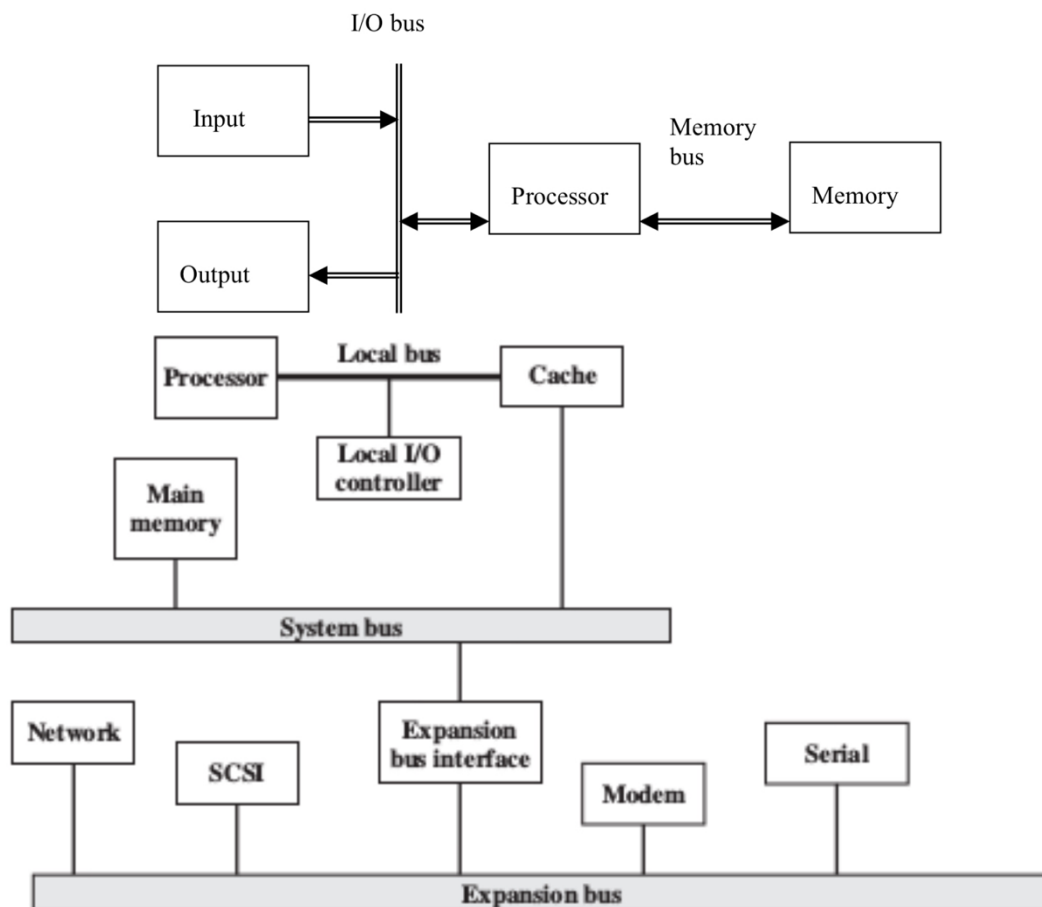


Figure 1.3 Single-bus structure.

The main virtue of the single-bus structure is its low cost and its flexibility for attaching peripheral devices.

Two Bus Structure: The bus is said to perform two distinct functions by connecting the I/O units with memory and processor unit with memory. The processor interacts with the memory through a memory bus and handles input/output functions over I/O bus. The I/O transfers are always under the direct control of the processor, which initiates transfer and monitors their progress until completion. The main advantage of this structure is good operating speed but on account of more cost



(a) Traditional bus architecture

Traditional/Multiple Bus Structure: There is a local bus that connects the processor to cache memory and that may support one or more local devices. There is also a cache memory controller that connects this cache not only to this local bus but also to the system bus.

On the system, the bus is attached to the main memory modules. In this way, I/O transfers to and from the main memory across the system bus do not interfere with the processor's activity. An expansion bus interface buffers data transfers between the system bus and the I/O controllers on the expansion bus.

Some typical I/O devices that might be attached to the expansion bus include: Network cards (LAN), SCSI(Small Computer System Interface), Modem, Serial Com etc..

Data Representation:

Data Types: Binary information in digital computers is stored in memory or processor registers. Registers contain either data or control information. Control information is a bit or a group of bits used to specify the sequence of command signals needed for manipulation of the data in other registers. Data are numbers and other binary coded information that are operated on to achieve required computational results.

The computer has to represent any type of data say numbers, characters, special symbols which are used in various programs. The data represented in the various registers must be categorized so as to convert them into data represented in the computer.

The data types found in digital computers can be classified as:-

1. Numbers used in arithmetic computations.
2. Letters of alphabet used in data processing.
3. discrete symbols.

Number System: There are mainly 4 types of Number systems, each has a specific Radix(r) or Base.

S.NO	System	Base	digits
1	Decimal	10	0,1,2...,9
2	Binary	2	0,1.
3	Octal	8	0,1,2..7.
4	Hexadecimal	16	0,1,2..9,A,B...,F.

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. For example, the decimal number system has a radix 10. 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

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

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

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

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. 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:

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

Convert Binary, Octal, and Hexadecimal to Decimal:

1. $(101101)_2 = 1 \times (2^5) + 0 \times (2^4) + 1 \times (2^3) + 1 \times (2^2) + 0 \times (2^1) + 1 \times (2^0) = (45)_{10}$
2. $(672.4)_8$
 $= 6 \times (8^2) + 7 \times (8^1) + 2 \times (8^0) + 4 \times (8^{-1})$
 $= 6 \times (64) + 7 \times (8) + 2 \times (1) + 4 \times (1/8) = (442.5)_{10}$
3. $(FADE)_{16} = 15 \times (16)^3 + 10 \times (16)^2 + 13 \times (16)^1 + 14 \times (16)^0$
 $= 15 \times (4096) + 10 \times (256) + 13 \times (16) + 14 = (64222)_{10}$
4. $(DEAF)_{16} = (57007)_{10}$

Convert Decimal to Binary, Octal, and Hexadecimal(Decimal integer to base r):

1. Repeatedly divide decimal integer by base r and collect reminders
2. First reminder a is $a \times r^0$
3. Second remainder b is $b \times r^1$ and so on
4. $(25)_{10} = (110101)_2$
 $25 / 2 = 12$ remainder 1
 $12 / 2 = 6$ remainder 0
 $6 / 2 = 3$ remainder 0
 $3 / 2 = 1$ remainder 1
 $1 / 2 = 0$ remainder 1
 $(25)_{10} = (11001)_2$

Decimal fraction to base r:

1. Repeatedly multiply decimal fraction by r and collect integers.
2. First integer a is $a \times r^{-1}$
3. Second integer b is $b \times r^{-2}$ and so on.

Ex: $(.375)_{10} = (?)_2$
 $(.375) \times 2 = 0.750$
 $(.750) \times 2 = 1.500$
 $(.500) \times 2 = 1.000$
 $(.375)_{10} = (.011)_2$

Binary to Octal and Hexadecimal: 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.

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.

1	2	7	5	4	3	Octal
1	0	1	0	1	1	Binary
A	F	6	3			Hexadecimal

Decimal Representation: In a computer, Decimal numbers are represented as binary-coded alphanumeric characters. These codes, 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. 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.

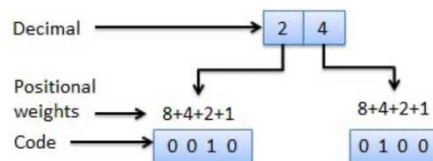
Advantages of Binary Code: Following is the list of advantages that binary code offers.

1. Binary codes are suitable for the computer applications.
2. Binary codes are suitable for the digital communications.
3. Binary codes make the analysis and designing of digital circuits if we use the binary codes.
4. Since only 0 & 1 are being used, implementation becomes easy.

Types of Binary Code: The Binary Codes are broadly categorized into the following four types

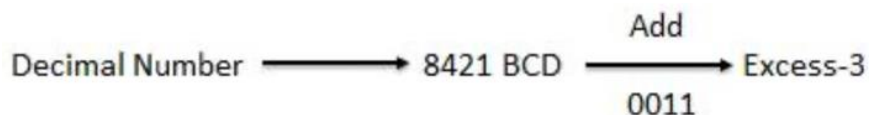
1. Weighted Codes
2. Non Weighted Codes
3. Binary Coded Decimal Code
4. Alphanumeric Codes

Weighted Codes: Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.



Non-Weighted Codes: In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

- Excess-3 code: The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding $(0011)_2$ or $(3)_{10}$ to each code word in 8421. The excess-3 codes are obtained as follows.



- Gray Code: It is the non-weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position. It has a very special feature that, only one bit will change each time the decimal number is incremented. As only one bit changes at a time, the gray code is called as a unit distance code. The gray code is a cyclic code. Gray code cannot be used for arithmetic operation.

TABLE 3-5 4-Bit Gray Code

Binary code	Decimal equivalent	Binary code	Decimal equivalent
0000	0	1100	8
0001	1	1101	9
0011	2	1111	10
0010	3	1110	11
0110	4	1010	12
0111	5	1011	13
0101	6	1001	14
0100	7	1000	15

Binary Coded Decimal (BCD) code: In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. In the BCD, with four bits we can represent sixteen numbers (0000 to 1111). But in BCD code only first ten of these are used (0000 to 1001). The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD. It is very similar to decimal system, but addition and subtraction have different rules and complicated.

Decimal digit	BCD 8421	2421	Excess-3
0	0000	0000	0011
1	0001	0001	0100
2	0010	0010	0101
3	0011	0011	0110
4	0100	0100	0111
5	0101	1011	1000
6	0110	1100	1001
7	0111	1101	1010
8	1000	1110	1011
9	1001	1111	1100

Alphanumeric Codes: The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items. The following two alphanumeric codes are very commonly used for the data representation.

- American Standard Code for Information Interchange (ASCII).
- Extended Binary Coded Decimal Interchange Code (EBCDIC).

ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

Complements: Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are two types of complements for each base r system: the r 's complement and the $(r - 1)$'s complement. When the value of the base r is substituted in the name, the two types are referred to as the 2's and 1's complement for binary numbers and the 10's and 9's complement for decimal numbers.

$(r - 1)$'s Complement:

For Decimal Numbers :

- Given a number N in base r having n digits, the $(r - 1)$'s complement of N is defined as $(r^n - 1) - N$.
- For decimal numbers $r = 10$ and $r - 1 = 9$, so the 9's complement of N is $(10^n - 1) - N$.
- Now, 10^n represents a number that consists of a single 1 followed by n 0's.
- $10^n - 1$ is a number represented by n 9's.
- For example, with $n = 4$ we have $10^4 = 10000$ and $10^4 - 1 = 9999$. It follows that the 9's complement of a decimal number is obtained by subtracting each digit from 9.
- For example, the 9's complement of 546700 is 999999 - 546700 = 453299 and the 9's complement of 12389 is 99999 - 12389 = 87610.

For Binary Numbers :

- For binary numbers, $r = 2$, $r - 1 = 1$, so the 1's complement of N is $(2^n - 1) - N$.
- Again, 2^n is represented by a binary number that consists of a 1 followed by n 0's.
- $2^n - 1$ is a binary number represented by n 1's.
- For example, consider $2^4 = 16 = (10000)_2$ and $2^4 - 1 = (1111)_2$. Thus the 1's complement, of a binary number is obtained by subtracting each digit from 1.
- However, the subtraction of a binary digit from 1 causes the bit to change from 0 to 1 or from 1 to 0.
- Therefore, the 1's complement of a binary number is formed by changing 1's into 0's and 0's into 1's.
- For example, the 1's complement of 1011001 is 0100110 and the 1's complement of 0001111 is 1110000.
- The $(r - 1)$'s complement of octal or hexadecimal numbers are obtained by subtracting each digit from 7 or F (decimal 15) respectively.

r 's Complement:

- The r 's complement of an n digit number N in base r is defined as $r^n - N$, for $N \neq 0$ and 0 for $N = 0$.
- Comparing with the $(r - 1)$'s complement, the r 's complement is obtained by adding 1 to the $(r - 1)$'s complement since $r^n - N = [(r^n - 1) - N] + 1$.
- Thus the 10's complement of the decimal 2389 is $7610 + 1 = 7611$ and is obtained by adding 1 to the 9's complement value. The 2's complement of binary 101100 is $010011 + 1 = 010100$ and is obtained by adding 1 to the 1's complement value.
- The 10's complement of N , can be formed by leaving all least significant 0's unchanged, subtracting the first nonzero least significant digit from 10, and then subtracting all higher significant digits from 9.
- The 10's complement of 246700 is 753300 and is obtained by leaving the two zeros unchanged, subtracting 7 from 10, and subtracting the other three digits from 9.

- Similarly, the 2's complement can be formed by leaving all least significant 0's and the first 1 unchanged, and then replacing 1's by 0's and 0's by 1's in all other higher, significant bits.
- The 2's complement of 1101100 is 0010100 and is obtained by leaving the two low-order 0's and the first 1 unchanged, and then replacing 1's by 0's and 0's by 1's in the other four most significant bit.

Subtraction of Unsigned Numbers:

Decimal Numbers: The subtraction of two n-digit unsigned numbers $M - N$ ($N \neq 0$) in base r can be done as follows:

1. Add the minuend M to the r 's complement of the subtrahend N . This performs $M + (r^n - N) = M - N + r^n$.
2. If $M \geq N$, the sum will produce an end carry r^n which is discarded, and what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r 's complement of $(N - M)$. To obtain the answer in a familiar form, take the r 's complement of the sum and place a negative sign in front.

Consider, for example, the subtraction 72532 complement of 13250 is 86750. Therefore:

$$\begin{array}{r}
 M = 72532 \\
 10\text{'s complement of } N = +86750 \\
 \text{Sum} = 159282 \\
 \text{Discard end carry } 10^5 = -100000 \\
 \text{Answer} = 59282
 \end{array}$$

Now consider an example with $M < N$. The subtraction 13250 - 72532 produces negative 59282. Using the procedure with complements, we get

$$\begin{array}{r}
 M = 13250 \\
 10\text{'s complement of } N = +27468 \\
 \text{Sum} = 40718
 \end{array}$$

There is no end carry and Answer is negative 59282 which is 10's complement of 40718 (i.e $99999 - 40718 = 59281 + 1$).

Binary Numbers: Subtraction with complements is done with binary numbers in a similar manner using the same procedure which is used for decimal numbers. Using the two binary numbers $X = 1010100$ and $Y = 1000011$, we perform the subtraction $X - Y$ and $Y - X$ using 2's complements as shown below:

$$\begin{array}{r}
 X = 1010100 \\
 2\text{'s complement of } Y = +0111101 \\
 \text{Sum} = 10010001 \\
 \text{Discard end carry } 2^7 = -10000000 \\
 \text{Answer: } X - Y = 0010001 \\
 \\
 Y = +000011 \\
 2\text{'s complement of } X = +0101100 \\
 \text{Sum} = 1101111
 \end{array}$$

There is no end carry, So answer is negative 0010001 = 2's complement of 1101111

Fixed Point Representation: In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. In addition to the sign, a number may have a binary (or decimal) point. The position of the binary point is needed to represent fractions, integers, or mixed integer-fraction numbers. There are two ways of specifying the position of the binary point in a register: by giving it a **fixed position** or by employing a **floating-point representation**.

“+” rep. by ‘0’

“-“ rep. by ‘1’

The fixed-point method assumes that the binary point is always fixed in one position. The two positions most widely used are:

- a binary point in the extreme left of the register to make the stored number a fraction
- a binary point in the extreme right of the register to make the stored number an integer.

The floating-point representation uses a second register to store a number that designates the position of the decimal point in the first register.

Integer Fixed Point Representation: When a number is positive and binary, the sign is represented by ‘0’ and the magnitude by a binary number. When a number is negative the sign is represented by 1 but the rest of the number may be represented in 3 popular Ways.

1. **Signed magnitude representation:** Signed Magnitude Representation is obtained by complementing sign bit only.
2. **Signed - 1’s complement representation:** Signed – 1’s complement is obtained by applying 1’s complement to all the bits including sign bit.
3. **Signed 2’s complement representation:** Signed – 2’s complement is obtained by applying 2’s complement to all the bits including sign bit.

The signed-magnitude representation of a negative number consists of the magnitude and a negative sign. In the other two representations, the negative number is represented in either the 1’s or 2’s complement of its positive value.

As an example, consider the signed number 12 stored in an 8-bit register. 12 is represented by a sign bit of 0 in the leftmost position followed by the binary equivalent of 12 : 00001100. Note that each of the eight bits of the register must have a value and therefore 0’s must be inserted in the most significant positions following the sign bit. Although there is only one way to represent 12, there are three different ways to represent 12 with eight bits.

	+12	-12
Signed – magnitude	0 0001100	1 0001100
Signed – 1’s complement	0 0001100	1 1110011
Signed – 2’s complement	0 0001100	1 1110100

Arithmetic Addition: The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. If the signs are the same, add the two magnitudes and give the sum the common sign. If the signs are different, subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude. For example, $(+25) + (-37) = -(37 - 25) = -12$.

2’s Complement Addition: The rule for adding numbers in the signed-2’s complement system does not require a comparison or subtraction, only addition and complementation. The procedure is Add the two

numbers, including their sign bits, and discard any carry out of the sign (leftmost) bit position. Numerical examples for addition are shown below. Note that negative numbers must initially be in 2's complement and that if the sum obtained after the addition is negative, it is in 2's complement form.

+ 6	00000110	-6	11111010
+13	00001101	+13	00001101
+19	00010011	+7	00000111
+ 6	00000110	-6	11111010
-13	11110011	-13	11110011
-7	11111001	-19	11101101

Arithmetic Subtraction (2's complement subtraction): Subtraction of two signed binary numbers when negative numbers are in 2's complement form can be stated as follows: Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit). A carry out of the sign bit position is discarded. This is demonstrated by the following relationship:

$$(\pm A) - (+B) = (\pm A) + (-B)$$

$$(\pm A) - (-B) = (\pm A) + (+B)$$

Consider the subtraction of (-6) - (-13). In binary with eight bits this is written as 11111010 - 11110011. The subtraction is changed to addition by taking the 2's complement of the subtrahend (-13) to give (+13). In binary this is 11111010 + 00001101 = 100000111. By removing the end carry, we will obtain the correct answer 00000111 (7).

Overflow: When two numbers of n digits each are added and the sum occupies n + 1 digits, then we say that an overflow occurred. An overflow is a problem in digital computers because the width of registers is finite. A result that contains n + 1 bits cannot be accommodated in a register with a standard length of n bits. Many computers detect the occurrence of an overflow, and when it occurs, a corresponding flip-flop is set which can then be checked by the user.

The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned. When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position. When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.

An overflow cannot occur after an addition if one number is positive and the other is negative, since adding a positive number to a negative number produces a result that is smaller than the larger of the two original numbers. An overflow may occur if the two numbers added are both positive or both negative.

For example, Two signed binary numbers, +70 and +80, are stored in two 8-bit registers. The range of numbers that each register can accommodate is from binary -128 to binary +127. Since the sum of the two numbers is +150, it exceeds the capacity of the 8-bit register. This is true if the numbers are both positive or both negative. The two additions in binary are shown below together with the last two carries.

carries: 0 1	+70	0 1000110	carries: 1 0	-70	1 0111010
	+80	0 1010000		-80	1 0110000
	+150	1 0010110		-150	0 1101010

Overflow Detection:-

1. An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position.
2. If these two carries are not equal, an overflow condition is produced.

3. If the two carries are applied to an exclusive – OR gate, an overflow will be detected when the output of the gate is equal to 1.
4. Carry in = Carry out in high-order digit
 - No overflow
5. Carry in $\neq$ carry out in high –order digit
 - Overflow

Decimal Fixed Point Representation: The representation of decimal numbers in registers is a function of the binary code used to represent a decimal digit. A 4-bit decimal code requires four flip-flops for each decimal digit.

Ex:- 3458 can be represented as 0011 0100 0101 1000 with 16 flip-flops..

The representation of signed decimal numbers in BCD is similar to the representation of signed numbers in binary. We can either use the familiar signed-magnitude system or the signed-complement system. The sign of a decimal number is usually represented with four bits to conform with the 4-bit code of the decimal digits. It is customary to designate a plus with four 0's and a minus with the BCD equivalent of 9, which is 1001.

The signed-complement system can be either the 9's or the 10's complement, but the 10's complement is the one most often used. To obtain the 10's complement of a BCD number, first take the 9's complement and then add one to the least significant digit. The 9's complement is calculated from the subtraction of each digit from 9.

The procedures developed for the signed-2's complement system apply also to the signed-10's complement system for decimal numbers. Addition is done by adding all digits, including the sign digit, and discarding the end carry. Obviously, this assumes that all negative numbers are in 10's complement form. Consider the addition $(+375) + (-240) = +135$ done in the signed 10's complement system.

$$\begin{array}{rcl}
 0\ 375 & (0000\ 0011\ 0111\ 0101)_{BCD} \\
 +\ 9\ 760 & (1001\ 0111\ 0110\ 0000)_{BCD} \\
 \hline
 0\ 135 & (0000\ 0001\ 0011\ 0101)_{BCD}
 \end{array}$$

The 9 in the leftmost position of the second number indicates that the number is negative. 9760 is the 10's complement of 0240. The two numbers are added and the end carry is discarded to obtain 135.

Floating Point Representation: The floating point representation of a number has two parts. The first part represents a signed, fixed-point number called the mantissa. The second part designates the position of the decimal (or binary) point and is called the exponent. The fixed point mantissa may be a fraction or an integer. For example, the decimal number 6132.789 is represented in floating-point with a fraction and an exponent as follows:

<i>Fraction</i>	<i>Exponent</i>
+0.6132789	+04

The value of the exponent indicates that the actual position of the decimal point is four positions to the right of the indicated decimal point in the fraction. This representation is equivalent to the scientific notation $0.6132789 * 10^{+4}$.

Floating-point is always interpreted to represent a number in the following form: $m * r^e$. Only the mantissa m and the exponent e are physically represented in the register (including their signs).

A floating-point binary number is represented in a similar manner except that it uses base 2 for the exponent. For example, the binary number 1001.11 is represented with an 8-bit fraction and 6-bit exponent as follows:

<i>Fraction</i>	<i>Exponent</i>
01001110	000100

The fraction has a 0 in the leftmost position to denote positive. The binary point of the fraction follows the sign bit but is not shown in the register. The exponent has the equivalent binary number 4. The floating-point number is equivalent to

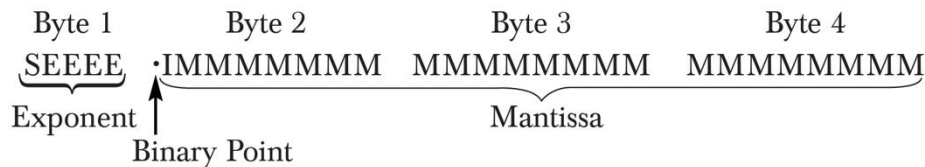
$$m \times 2^e = +(.1001110)_2 \times 2^{+4}$$

A floating-point number is said to be normalized if the most significant digit of the mantissa is nonzero. For example, the decimal number 350 is normalized but 00035 is not. For example, the 8-bit binary number 00011010 is not normalized because of the three leading 0's. The number can be normalized by shifting it three positions to the left and discarding the leading 0's to obtain 11010000. The three shifts multiply the number by $2^3 = 8$. To keep the same value for the float-ing-point number, the exponent must be subtracted by 3. Normalized numbers provide the maximum possible precision for the floating-point number.

Two main standard forms of floating-point numbers are from the following organizations

1. ANSI (American National Standards Institute)
2. IEEE (Institute of Electrical and Electronic Engineers).
1. **ANSI:** ANSI form represents floating point number in byte format. The following syntax represents the byte format.

Byte Format:

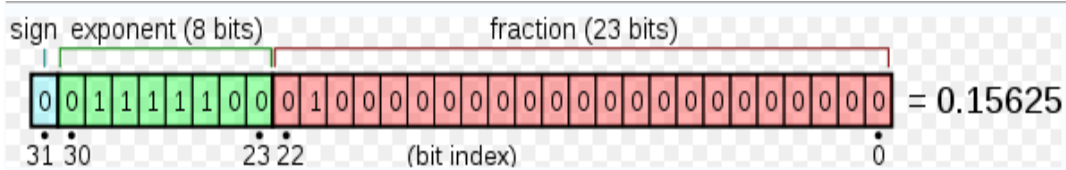


S = Sign of Mantissa, E = Exponent Bits in 2's complement, M = Mantissa Bits

$$\begin{aligned}
 13 &= 1101 = 0.1101 \times 2^4 \\
 &= 00000100 \ 11010000 \ 00000000 \ 00000000 \\
 -17 &= -10001 = -0.10001 \times 2^5 \\
 &= 10000101 \ 10001000 \ 00000000 \ 00000000 \\
 -0.125 &= -0.001 = -.1 \times 2^{-2} \\
 &= 11111110 \ 10000000 \ 00000000 \ 00000000
 \end{aligned}$$

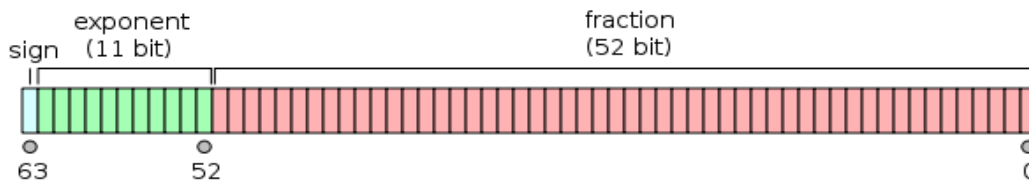
2. **IEEE 754:** IEEE standard represents floating point number in 32-bits and 64-bits format. The 32-bit standard representation is called as **single-precision** representation and 64-bit standard representation is called as **double precision** representation. The single precision representation occupies a single 32-bit word. These 32-bits are divided into the three fields as shown below:

Field 1	Sign	→	1-bit
Field 2	Exponent	→	8-bits
Field 3	Mantissa	→	23-bits



The sign of the number is given in the first bit, followed by a representation for the exponent. Instead of the signed exponent E , the value actually stored in the exponent field is $E^1 = E + 127$, followed by mantissa. The double precision representation occupies a single 64-bit word. These 64-bits are divided into the three fields as shown below:

Field 1	Sign	→	1-bit
Field 2	Exponent	→	11-bits
Field 3	Mantissa	→	52-bits



The sign of the number is given in the first bit, followed by a representation for the exponent. Instead of the signed exponent E , the value actually stored in the exponent field is $E^1 = E + 1023$, followed by mantissa.

Example: Represent 32.75 and 18.125 in single precision IEEE 754 representation.

Ans: IEEE 754 single precision standard representation has the following format.

Sign(1-bit)	Exponent(8-bits)	Mantissa(23-bits)
-------------	------------------	-------------------

1. 32.75: Since the given number is positive, hence the sign of the number is '0'. Initially convert the above number into its binary equivalent. That is, $(32.75)_{10} = (100000.11)_2$

Normalize the above binary value. That is $1.0000011 * 2^5$

The exponent value E is 5. The actual value stored in exponent is $E^1 = E + \text{bias}$ (127 for single precision). Hence, $E^1 = 5 + 127 = (132)_{10} = (10000100)_2$

Mantissa is 0000011

Finally, represent all the values in the diagrammatic format specified above that is

0	10000100	00000110000000000000000
---	----------	-------------------------

2. 18.125: Since the given number is positive, hence the sign of the number is '0'. Initially convert the above number into its binary equivalent. That is, $(18.125)_{10} = (10010.001)_2$

Normalize the above binary value. That is $1.0010001 * 2^4$

The exponent value E is 4. The actual value stored in exponent is $E^1 = E + \text{bias}$ (127 for single precision). Hence, $E^1 = 4 + 127 = (131)_{10} = (10000011)_2$

Mantissa is 0000011

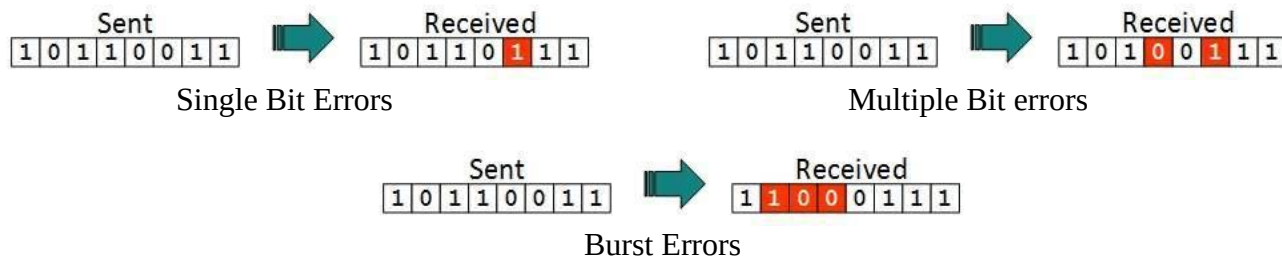
Finally, represent all the values in the diagrammatic format specified above that is

0	10000011	001000100000000000000000
---	----------	--------------------------

Other Binary Codes: Gray Code, BCD Code, excess – 3 code, Alphanumeric codes.

Error Detection Codes: When the digital information in the binary form is transmitted from one system to another system an error may occur due to presence of noise. Two types of codes that are used for single error detection and correction are error detection codes and error correction codes. An error detection code is a binary code that detects digital errors during transmission. The detected errors cannot be corrected but their presence is indicated. An error-correction code is a binary code that detects and corrects single bit error occurred during transmission. The most popular error correcting technique is Hamming Code.

The different types of errors are: Single Bit errors, Multiple Bit errors, and Burst Errors.



The most popular error detection technique is Parity Bit.

Parity Bit: A Parity Bit is used for the purpose of detecting errors during the transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The parity bit can be of two types. That is odd parity and even parity. If the total number of 1's in the data including the parity bit corresponds to an odd number of 1's then the parity is referred as odd parity. If the total number of 1's in the data including the parity bit corresponds to an even number of 1's then the parity is referred as even parity. Consider the binary representation of numbers along with their respective parity bits are shown below.

TABLE 3-7 Parity Bit Generation

Message <i>xyz</i>	<i>P</i> (odd)	<i>P</i> (even)
000	1	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	0	1

→*P*(odd) is chosen in such a that the sum of 1's is odd.

→*P*(even) is chosen in such a way that the sum is even.

During transfer of information from one location to another, the parity bit is handled as follows. At the sending end, the message (in this case three bits) is applied to a **parity generator**, where the required parity bit is generated. The message, including the parity bit, is transmitted to its destination. At the receiving end, all the incoming bits (in this case, four) are applied to a **parity checker** that checks the proper parity adopted (odd or even). An error is detected if the checked parity does not conform to the adopted parity. The parity

method detects the presence of one, three, or any odd number of errors. An even number of errors is not detected.

An error is detected if the checked parity does not correspond with the transmitted one. The circuit that generates the parity bit in the transmitter is called a **parity generator** and the circuit that checks the parity in the receiver is called a **Parity Checker**.

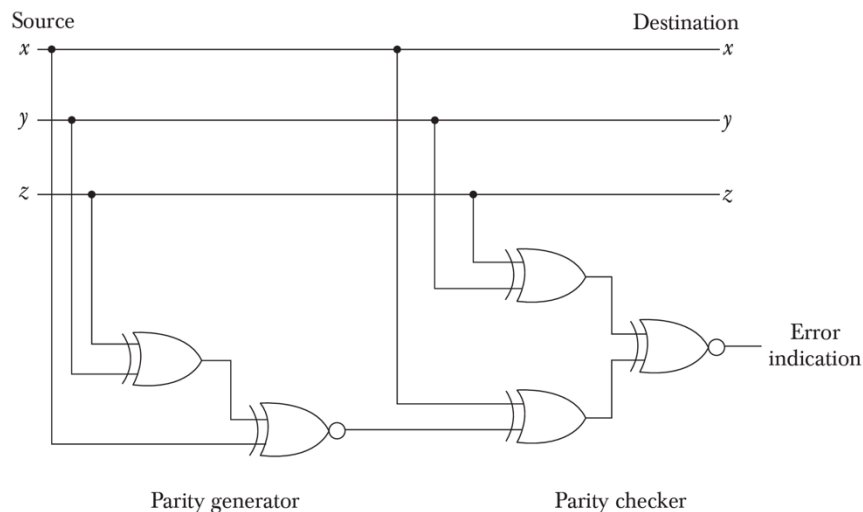
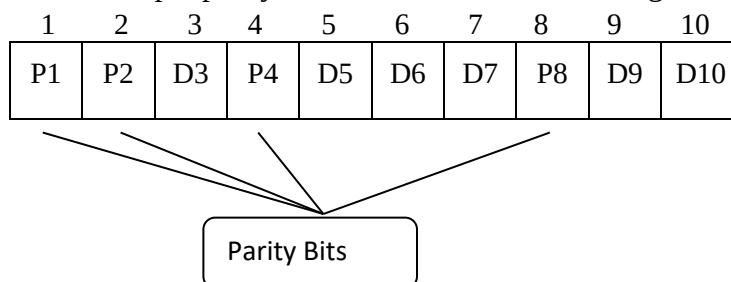


Figure 3-3 Error detection with odd parity bit.

The output of the parity checker would be 1 when an error occurs, otherwise 0. The even-parity generators and checkers can be implemented with exclusive OR functions. Odd-parity networks need an exclusive NOR at the output to complement the function.

Hamming code: A Hamming code is a linear error correcting code named after its inventor, Richard Hamming. Hamming codes can detect and correct single-bit errors, and can detect (but not correct) double-bit errors. The hamming code uses multiple parity bits. The format for hamming code is shown in the following figure.



Hamming code will be generated as shown below:

1. Mark all bit positions that are powers of two as parity bits. (positions 1, 2, 4, 8, 16, 32, 64, etc.)
2. All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.)
3. Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.
 Position 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1,3,5,7,9,11,13,15,...)
 Position 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc. (2,3,6,7,10,11,14,15,...)
 Position 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc. (4,5,6,7,12,13,14,15,20,21,22,23,...)
 Position 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc. (8-15,24-31,40-47,...)
 Position 16: check 16 bits, skip 16 bits, check 16 bits, skip 16 bits, etc. (16-31,48-63,80-95,...)
 Position 32: check 32 bits, skip 32 bits, check 32 bits, skip 32 bits, etc. (32-63,96-127,160-191,...)etc.

4. Set a parity bit to 1 if the total number of ones in the positions it checks is odd. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

For example, if the data to be transmitted is 1101, then hamming code is given as follows

1	2	3	4	5	6	7
P1	P2	1	P4	1	0	1

P1 checks positions 1,3,5,7,9,11,13,15 etc....hence P1, D3 (1), D5 (1), D7 (1). Since total number of 1's is odd then P1 becomes 1. Then

0	P2	1	P4	1	0	1
---	----	---	----	---	---	---

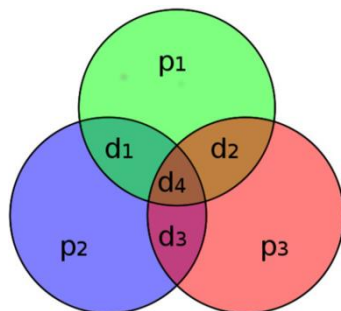
P2 checks positions 2, 3, 6, 7, 10, 11 etc that is P2, D3 (1), D6 (0), D7 (1). Since total number of 1's is even then P2 becomes 0. Then

0	0	1	P4	1	0	1
---	---	---	----	---	---	---

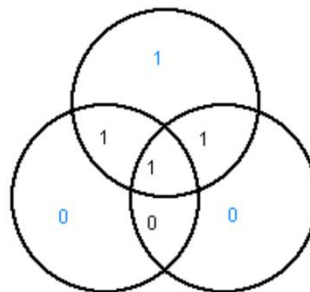
P4 checks positions 4, 5, 6, 7, 12, 13, 14, 15 etc..that is P4, D5 (1), D6 (0), D7 (1). Since total number of 1's is even then P4 becomes 0. Then

0	0	1	0	1	0	1
---	---	---	---	---	---	---

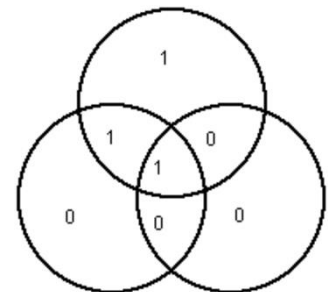
Hamming code uses Venn diagrams to detect and correct single bit errors. The following illustrates the use of Venn diagrams on 4-bit words. There are three intersecting circles with 7 compartments. Four inner compartments are used to store data bits and remaining compartments are filled with parity bits.



Bit Positions

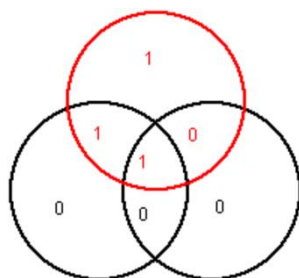


Date Sent

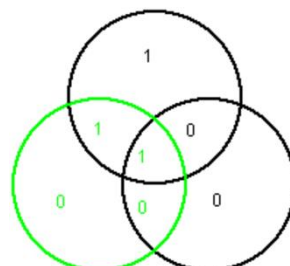


Data Received

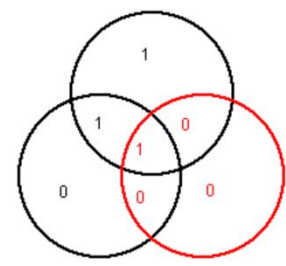
Imagine, data sent and received is 1101100 (left most 4 bits indicates data and right most three bits indicates parity bits.), 1001100. The receiver discovers that an error occurred by checking the parity of the three circles. Moreover, the receiver can even determine where the error occurred and recover the four original message bits.



$$1 + 0 + 1 + 1 = 1 \text{ NOT OK!}$$



$$0 + 1 + 0 + 1 = 0 \text{ OK!}$$



$$0 + 1 + 0 + 0 = 1 \text{ NOT OK!}$$

Since the parity check for top circle and right circle failed, but left circle was ok. There is only bit that could be responsible for this is d2 and make its complement to correct it. If the Center bit d4, is corrupted then all three parity check will fail. If parity bit itself corrupted, then only one parity check will fail.

Computer Arithmetic: The Addition, subtraction, multiplication and division are the four basic arithmetic operations. These operations addition, subtraction, multiplication, and division can be performed on the following types of data:

1. Fixed-point binary data (Signed Magnitude and Signed 2's complement representation)
2. Floating-point binary data
3. Binary-coded decimal (BCD) data

Addition and Subtraction: There are three ways of representing negative fixed-point binary numbers:

1. Signed-magnitude
2. Signed one's complement
3. Signed two's complement.

Most computers use the signed-2's complement representation when performing arithmetic operations with integers. For floating-point operations, most computers use the signed-magnitude representation for the Mantissa.

Addition and Subtraction with signed-magnitude Data: Consider the magnitude of the two numbers by A and B. When the signed numbers are added or subtracted, there are eight different conditions to consider, depending on the sign of the numbers and the operation performed. These conditions are listed in the following table.

TABLE 10-1 Addition and Subtraction of Signed-Magnitude Numbers

Operation	Add Magnitudes	Subtract Magnitudes		
		When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+(A + B)$			
$(+A) + (-B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(-A) + (+B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$
$(-A) + (-B)$	$-(A + B)$			
$(+A) - (+B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(+A) - (-B)$	$+(A + B)$			
$(-A) - (+B)$	$-(A + B)$			
$(-A) - (-B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$

Addition (Subtraction) Algorithm: When the signs of A and B are identical (different), add the two magnitudes and attach the sign of A to the result. When the signs of A and B are different (identical), compare the magnitudes and subtract the smaller number from the larger. Choose the sign of the result to be the same as A if $A > B$ or the complement of the sign of A if $A < B$. If the two magnitudes are equal/ subtract B from A and make the sign of the result positive.

Hardware Implementation: Addition and Subtraction with Signed-Magnitude Data Hardware Design shows a block diagram of the hardware for implementing the addition and subtraction operations. It consists of registers A and B and sign flip-flops A_s and B_s . Subtraction is done by adding A to the 2's complement of B. The output carry is transferred to flip-flop E, where it can be checked to determine the relative magnitudes of the two numbers. The add overflow flip-flop AVF holds the overflow bit when A and B are added.

The addition of A plus B is done through the parallel adder. The complemener provides an output of B or the complement of B depending on the state of the mode control M. When $M = 0$, the output of B is

transferred to the adder, the input carry is 0, and the output of the adder is equal to the sum $A + B$. When $M = 1$, the 1's complement of B is applied to the adder, the input carry is 1, and output $S = A + \underline{B} + 1$. This is equal to A plus the 2's complement of B , which is equivalent to the subtraction, $A - B$. The S (sum) output of the adder is applied to the input of the A register.

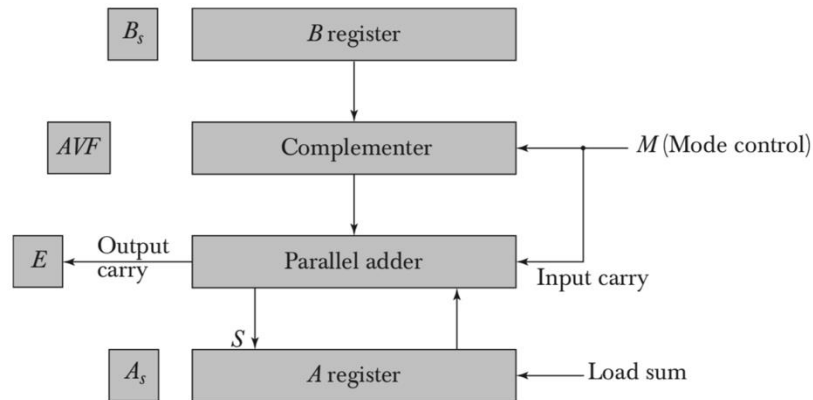


Figure 10-1 Hardware for signed-magnitude addition and subtraction.

Hardware Algorithm:

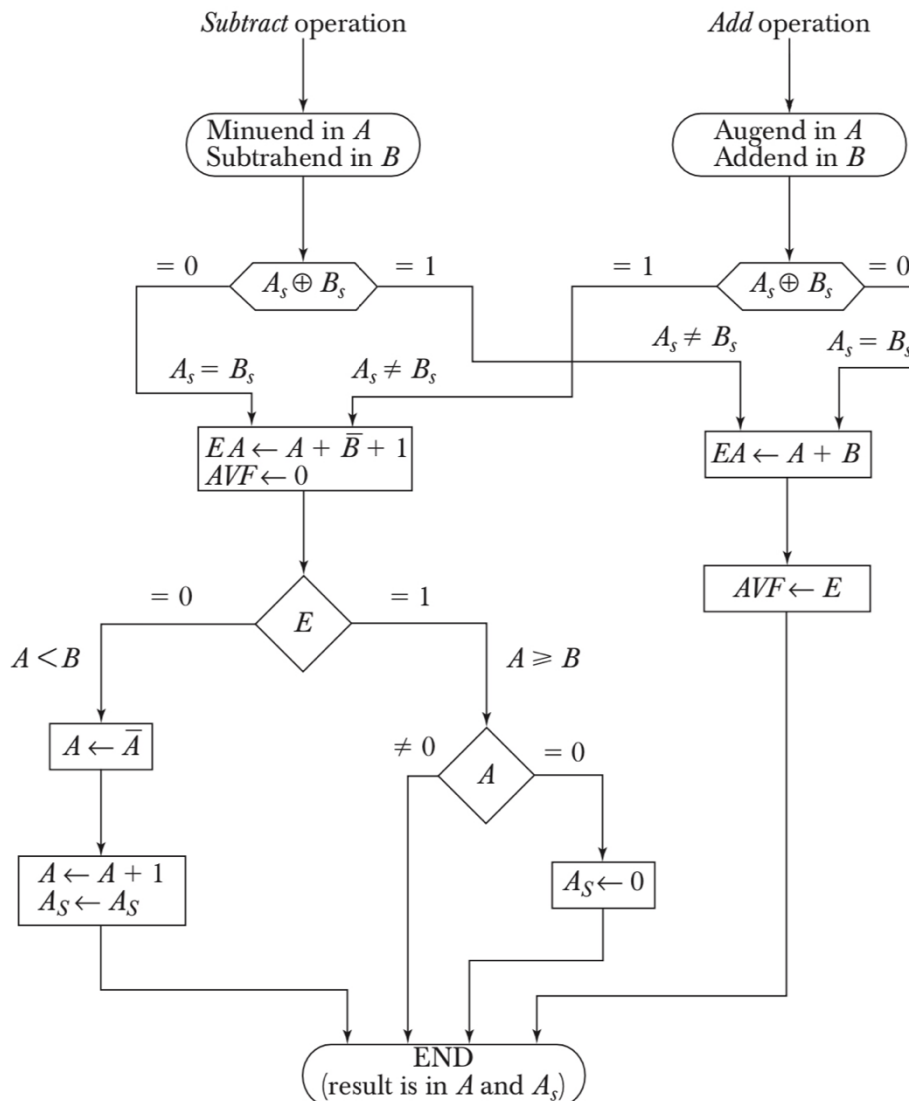


Figure 10-2 Flowchart for add and subtract operations.

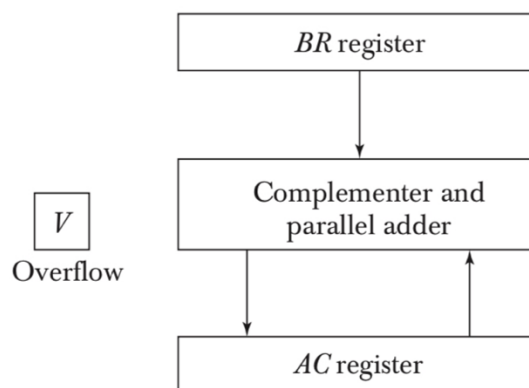
The flowchart for the addition and subtraction is presented in above figure.

- The two signs **As** and **Bs** are compared by an exclusive-OR gate.
- If the output of the gate is 0, the signs are identical; if it is 1, the signs are different.
- For an *add* operation, identical signs indicate that the magnitudes be added.
- For a *subtract* operation, different signs indicate that the magnitudes be subtracted.
- The magnitudes are added with a micro-operation $EA \leftarrow A+B$, where EA is a register that combines E and A.
- The carry in E after the addition indicates an overflow if it is equal to 1.
- If $E = 1$, then $A \geq B$.
 - However, if $A = 0$, then the sign is made positive.
- If $E = 0$, then $A < B$ and 2's complement of A is stored back into A and sign for A is complemented.

Addition and Subtraction with Signed-2's Complement Algorithm: The addition of two numbers in signed 2's complement form consists of adding the numbers with signed 2's complement data. The subtraction is performed by adding 2's complement of the subtrahend to the minuend.

When two numbers of n digits are added and sum occupies $n+1$ digits, we say that an overflow is occurred. An overflow can be detected by observing the carry into the sign bit position and carry out of the sign bit position. When the two carriers are applied to exclusive-OR gates, the overflow is detected when the output of the gate is equal to 1. The register configuration for the hardware implementation is as shown below.

Hardware for signed-2's complement addition and subtraction.



In this case, unlike signed magnitude data sign bits are not separated from the rest of the registers. The left most bit in AC and BR represent the sign bits of the numbers. The two sign bits are added or subtracted together with the other bits in the complementer and parallel adder. The overflow flip-flop V is set to 1 if there is an overflow. The output carry in this case is discarded.

The algorithm (flowchart) for adding and subtracting two binary numbers in signed 2's complement representation is as shown below.

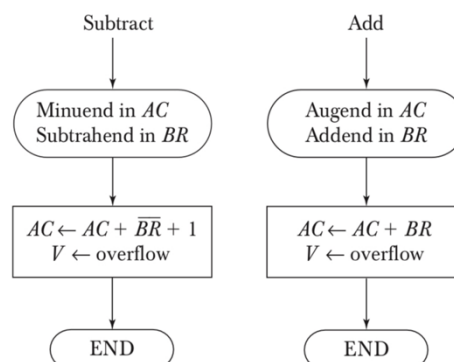


Figure 10-4 Algorithm for adding and subtracting numbers in signed-2's complement representation.

The sum is obtained by adding the contents of AC and BR (including their sign bits). The overflow bit is set to 1 if there is an overflow, else it is set to 0. The subtraction is obtained by adding the content of AC to the 2's complement of BR.

Multiplication: Multiplication of two fixed-point binary numbers in signed magnitude representation is done with successive shift and adds operations. This process is best illustrated with a numerical example:

23	10111	Multiplicand
19	× 10011	Multiplier
	10111	
	10111	
	00000	+
	00000	
	10111	
437	110110101	Product

This process looks at successive bits of the multiplier, least significant bit first. If the multiplier bit is 1, the multiplicand is copied down; otherwise, zeros are copied down. The numbers copied down in successive lines are shifted one position to the left from the previous numbers. Finally, the numbers are added and their sum produces the product.

The sign of the product is determined from the signs of the multiplicand and multiplier. If they are same, the sign of the product is positive. If they are different, the sign of the product is negative.

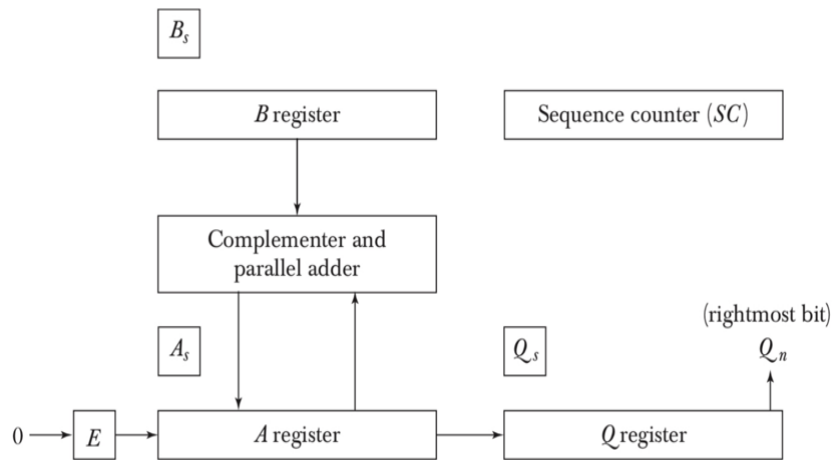
Multiplication is performed as shown below:

1. Multiplication involves the generation of partial products, one for each digit in the multiplier. These partial products are then summed to produce the final product.
2. The partial products are easily defined. When the multiplier bit is 0, the partial product is 0. When the multiplier is 1, then the partial product is the multiplicand.
3. The total product is produced by summing the partial products. For this operation, each successive partial product is shifted one position to the left relative to the preceding partial product.
4. The multiplication of two n-bit binary integers results in a product of up to 2n bits in length.

Hardware implementation for Multiplication with Signed Magnitude Data: Initially the multiplier is stored in Q register and its sign is Q_s . The sequence counter is initially set to a number which is equal to number of bits in the multiplier. The counter is decremented by 1 after forming each partial product. When the contents of the counter reaches zero, the product is formed and the process stops.

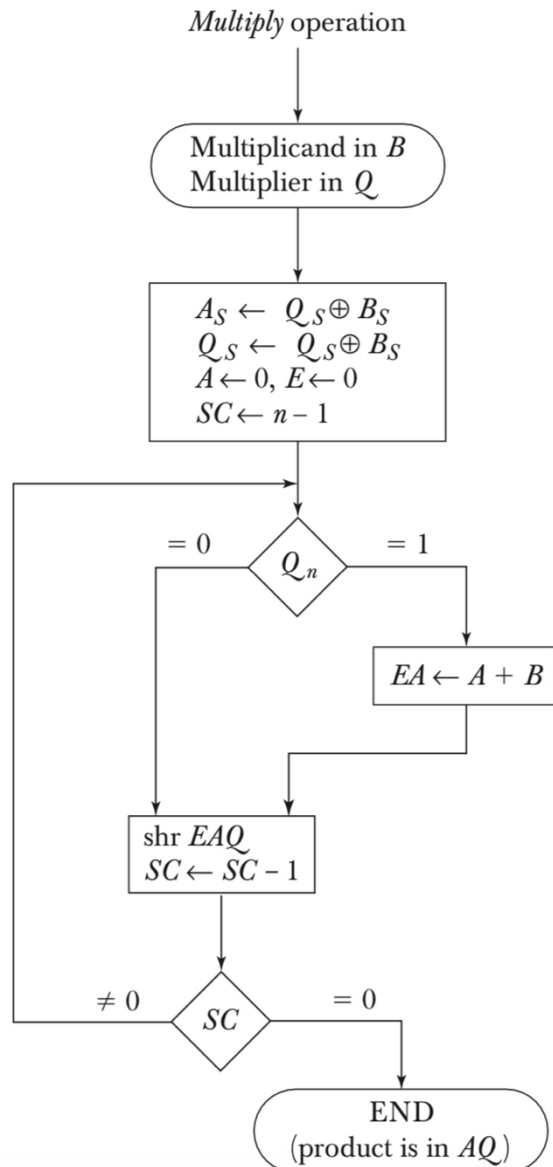
Initially, the multiplicand is in register B and the multiplier is in Q. The sum of A and B forms a partial product which is transferred to the EA register. Both partial product and multiplier are shifted to right. This shift will be denoted by shr EAQ. The least significant bit of A is shifted into the most significant position of Q, the bit from E is shifted into the most significant position of A, and 0 is shifted into E. After the shift, one bit of the partial product is shifted into Q, pursuing the multiplier bits one position to the right. In this manner, the rightmost flip-flop in register Q, designated by Q_n , will hold the bit of the multiplier, which must be inspected next.

Figure 10-5 Hardware for multiply operation.



Hardware Algorithm: The following flowchart represents hardware multiply algorithm.

Figure 10-6 Flowchart for multiply operation.



Initially, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in B_s and Q_s, respectively. The signs are compared, and both A and Q are set to correspond to the sign of the product since a double-length product will be stored in registers A and Q. Registers A and E are cleared and the sequence counter SC is set to a number equal to the number of bits of the multiplier.

After the initialization, the low-order bit of the multiplier in Q_n is tested. If it is a 1, the multiplicand in B is added to the present partial product in A. If it is a 0, nothing is done. Register EAQ is then shifted once to the right to form the new partial product. The sequence counter is decremented by 1 and its new value checked. If it is not equal to zero, the process is repeated and a new partial product is formed. The process stops when SC = 0. Note that the partial product formed in A is shifted into Q one bit at a time and eventually replaces the multiplier. The final product is available in both A and Q, with A holding the most significant bits and Q holding the least significant bits.

Example: $23 * 19 = 437$

TABLE 10-2 Numerical Example for Binary Multiplier

Multiplicand $B = 10111$	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
$Q_n = 1$; add B		<u>10111</u>		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
$Q_n = 1$; add B		<u>10111</u>		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
$Q_n = 0$; shift right EAQ	0	01000	10110	010
$Q_n = 0$; shift right EAQ	0	00100	01011	001
$Q_n = 1$; add B		<u>10111</u>		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in $AQ = 0110110101$				

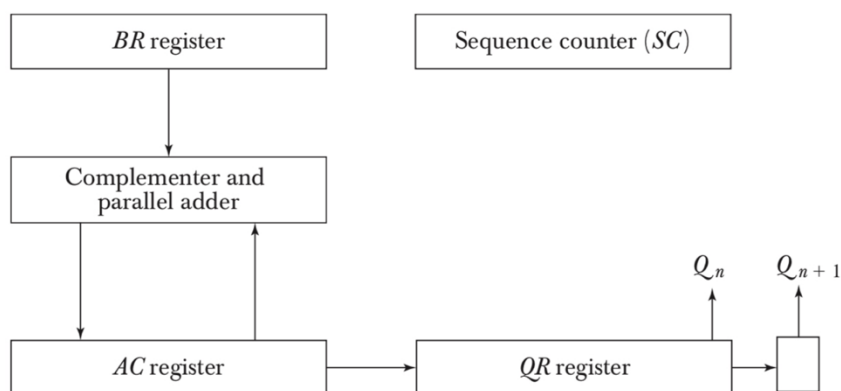
Multiplication with Signed-2's Complement data (Booth's algorithm): Booth algorithm multiplies binary integers in signed 2's complement representation. If the numbers are represented in signed 2's complement then we can multiply them by using Booth algorithm.

Booth algorithm needs examination of the multiplier bits and shifting of the partial product. Prior to the shifting, the multiplicand added to the partial product, subtracted from the partial product, or left unchanged by the following rules:

1. The multiplicand is subtracted from the partial product when we get the first least significant 1 in a string of 1's in the multiplier.(when $Q_n Q_{n+1} = 10$)
2. The multiplicand is added to the partial product when we get the first 0 (provided that there was a previous 1) in a string of 0's in the multiplier.(when $Q_n Q_{n+1} = 01$)
3. The partial product does not change when the multiplier bit is the same as the previous multiplier bit.(when $Q_n Q_{n+1} = 11$ or 00)

Hardware for Booth Algorithm: Booth Algorithm use registers AC, BR, and QR respectively. Q_n designates the least significant bit of the multiplier in register QR. An extra flip-flop Q_{n+1} is appended to QR to facilitate a double bit inspection of the multiplier.

Figure 10-7 Hardware for Booth algorithm.



Flow Chart for Booth Algorithm:

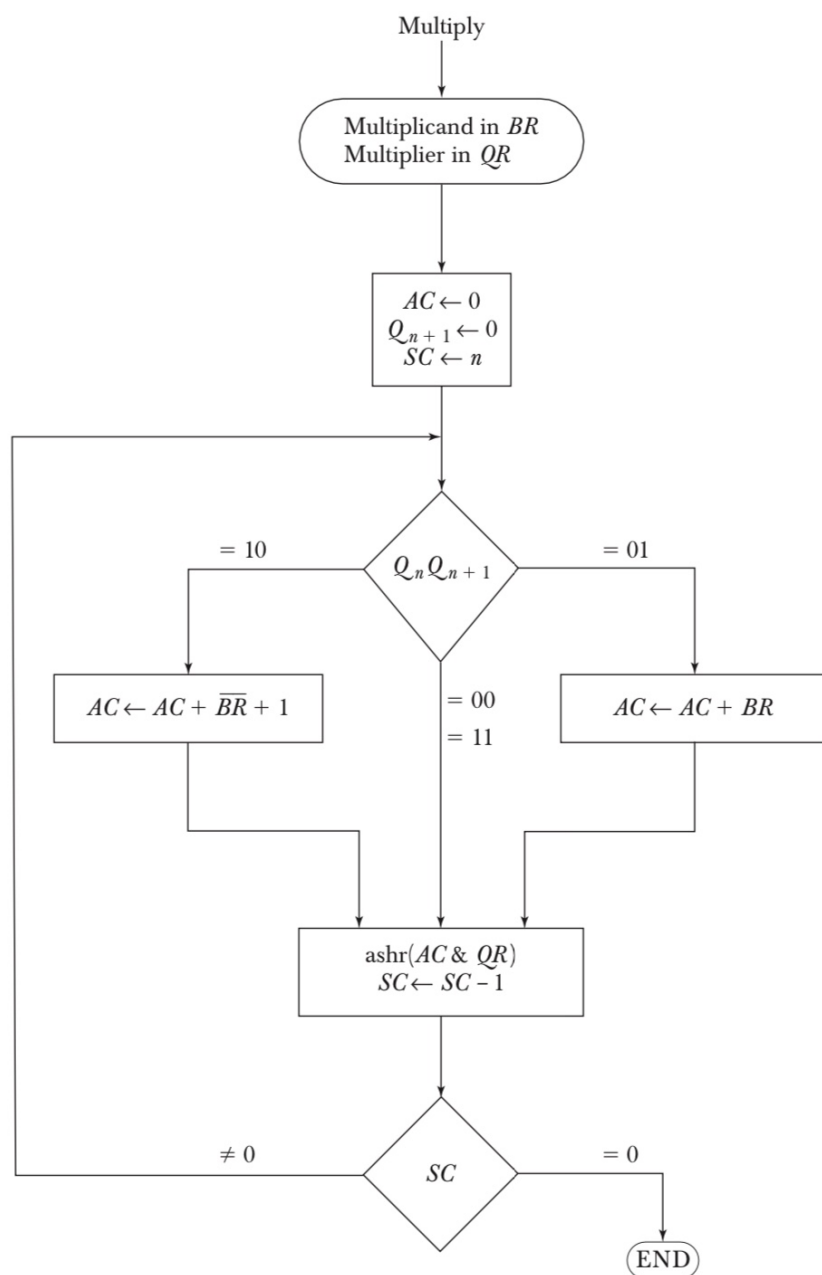


Figure 10-8 Booth algorithm for multiplication of signed-2's complement numbers.

Initially, multiplicand is in BR and multiplier is in QR. AC and the appended bit Q_{n+1} are initially cleared to 0 and the sequence counter SC is set to a number n equal to the number of bits in the multiplier. The two bits of the multiplier in Q_n and Q_{n+1} are inspected. If the two bits are equal to 10, it performs a subtraction of the multiplicand from the partial product in AC. If the two bits are equal to 01, it performs addition of the multiplicand to the partial product in AC. When the two bits are equal, the partial product does not change. The next step is to shift right the partial product and the multiplier (including bit Q_{n+1}). This is an arithmetic shift right (ashr) operation which shifts AC and QR to the right and leaves the sign bit in AC unchanged. The sequence counter is decremented and the computational loop is repeated n times.

Example: $(-9) * (-13) = +117$

TABLE 10-3 Example of Multiplication with Booth Algorithm

$Q_n Q_{n+1}$	$BR = 10111$ $\overline{BR} + 1 = 01001$	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract BR	<u>01001</u> 01001			
	ashr	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add BR	<u>10111</u> 11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract BR	<u>01001</u> 00111			
	ashr	00011	10101	1	000

Division: The divisor is stored in the B register and the double length dividend is stored in registers A and Q. In division algorithms there is a possibility of occurrence of divide overflow. The divide overflow occurs if the high order half bits of the dividend is greater than or equal to the divisor. If there is no overflow, the dividend is shifted left and the divisor is subtracted by adding its 2's complement value. The relative magnitude is available in E. If $E=1$, a quotient bit 1 is inserted in Q_n , and the partial remainder is shifted left, and repeat the process. If $E=0$, the quotient Q_n remains 0. The value of B is then added to restore the partial remainder in A. the partial remainder is shifted to the left and the process is repeated again. Finally the quotient is in Q and remainder is in A.

Hardware implementation for division operation is similar to multiplication operation.

Hardware Algorithm (Restore Method): The dividend is in A and Q and the divisor in B. The sign of the result is transferred into Q_s to be part of the quotient. A constant is set into the sequence counter SC to specify the number of bits in the quotient.

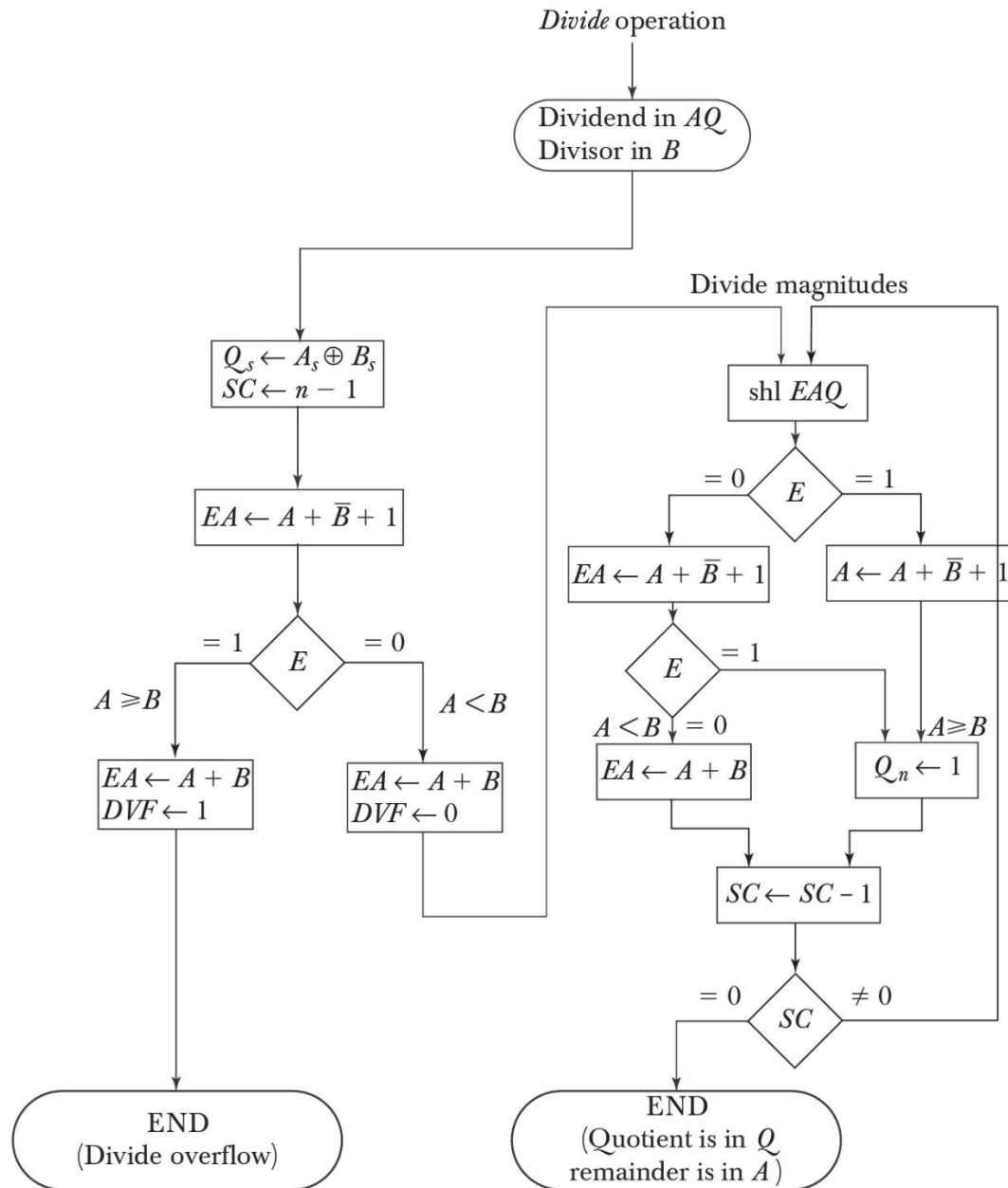
A divide-overflow condition is tested by subtracting the divisor in B from half of the bits of the dividend stored in A. If $A \geq B$, the divide-overflow flip-flop DVF is set and the operation is terminated prematurely. If $A < B$, no divide overflow occurs so the value of the dividend is restored by adding B to A.

The division of the magnitudes starts by shifting the dividend in AQ to the left with the high-order bit shifted into E. If the bit shifted into E is 1, B must be subtracted from A and 1 inserted into Q_n for the quotient bit.

If the shift-left operation inserts a 0 into E, the divisor is subtracted by adding its 2's complement value and the carry is transferred into E. If $E = 1$, it signifies that $A \geq B$; therefore, Q_n is set to 1. If $E = 0$, it signifies that $A < B$ and the original number is restored by adding B to A .

This process is repeated again with register A holding the partial remainder. After $n-1$ times, the quotient magnitude is formed in register Q and the remainder is found in register A . The quotient sign is in Q_s and the sign of the remainder in A_s is the same as the original sign of the dividend.

Figure 10-13 Flowchart for divide operation.



Comparison and Non-Restoring Method: In the non-restoring method, B is not added if the difference is negative, instead the negative difference is shifted.

In Comparison method, A and B are compared prior to the subtraction operation. Then if $A \geq B$, B is subtracted from A . If $A < B$ nothing is done.

Example: $448/17 = \text{Quotient } 26, \text{ Remainder } 6$

Divisor $B = 10001$,

$\bar{B} + 1 = 01111$

	<u>E</u>	<u>A</u>	<u>Q</u>	<u>SC</u>
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\bar{B} + 1$		<u>01111</u>		
$E = 1$	1	01011		
Set $Q_n = 1$	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\bar{B} + 1$		<u>01111</u>		
$E = 1$	1	00101		
Set $Q_n = 1$	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\bar{B} + 1$		<u>01111</u>		
$E = 0$; leave $Q_n = 0$	0	11001	00110	
Add B		<u>10001</u>		2
Restore remainder	1	01010		
shl EAQ	0	10100	01100	
Add $\bar{B} + 1$		<u>01111</u>		
$E = 1$	1	00011		
Set $Q_n = 1$	1	00011	01101	1
shl EAQ	0	00110	11010	
Add $\bar{B} + 1$		<u>01111</u>		
$E = 0$; leave $Q_n = 0$	0	10101	11010	
Add B		<u>10001</u>		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A :		00110		
Quotient in Q :			11010	

Figure 10-12 Example of binary division with digital hardware.