# Unit-4 (Part-A)
# COMPUTER-ARITHMATIC

## 4.1 COMPUTER ARITHMETIC

→ Data is manipulated by using the arithmetic instructions in digital computers. Data is manipulated to produce results necessary to give solution for the computation problems. **The Addition, subtraction, multiplication and division are the four basic arithmetic operations**. If we want then we can derive other operations by using these four operations

→ To execute arithmetic operations there is a separate section called arithmetic processing unit in central processing unit. The arithmetic instructions are performed generally on binary or decimal data. **Fixed-point numbers are used to represent integers or fractions**. We can have signed or unsigned negative numbers. Fixed-point addition is the simplest arithmetic operation.

→ If we want to solve a problem then **we use a sequence of well- defined steps**. These steps are collectively called **algorithm**. To solve various problems we give algorithms. In order to solve the computational problems, arithmetic instructions are used in digital computers that manipulate data. These instructions perform arithmetic calculations.

→ And these instructions perform a great activity in processing data in a digital computer. As we already stated that with the four basic arithmetic operations addition, subtraction, multiplication and division, it is possible to derive other arithmetic operations and solve scientific problems by means of numerical analysis methods.

→ A processor has an arithmetic processor(as a sub part of it) that executes arithmetic operations. The data type, assumed to reside in processor, registers during the execution of an arithmetic instruction.

→ Negative numbers may be in a signed magnitude or signed complement representation. **There are three ways of representing negative fixed point -** binary numbers signed magnitude, signed 1's complement or signed 2's complement. Most computers use the signed magnitude representation for the mantissa.

### 4.1.1 Addition and Subtraction with Signed –Magnitude Data

→ We designate the magnitude of the two numbers by A and B. Where the signed numbers are added or subtracted, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operation performed.

→ These conditions are listed in the first column of Table 4.1. The other columns in the table show the actual operation to be performed with the magnitude of the numbers.

→ The last column is needed to present a negative zero. In other words, when two equal numbers are subtracted, the result should be +0 not -0.

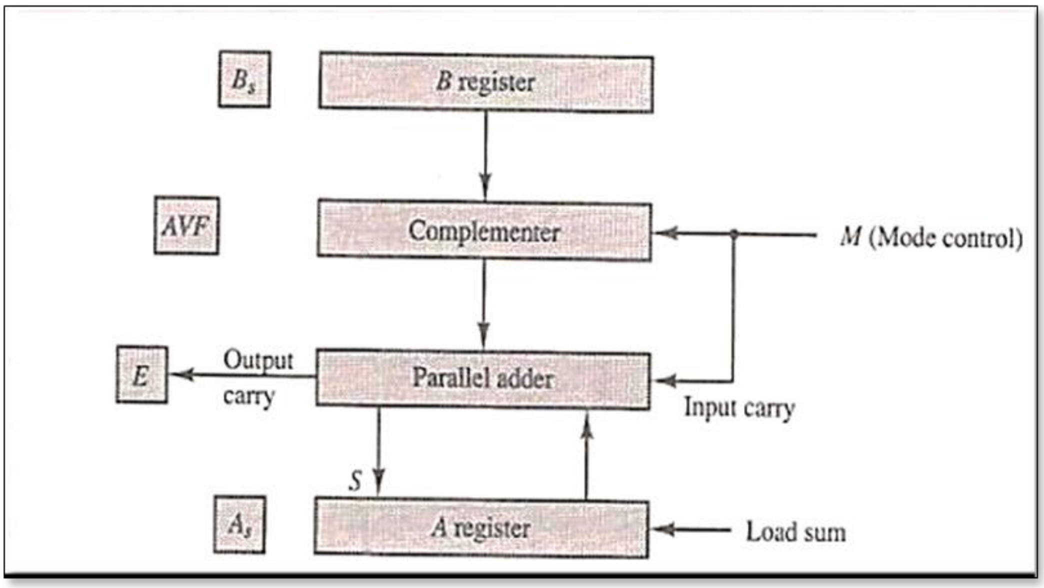
→ The algorithms for addition and subtraction are derived from the table and can be stated as follows (the words parentheses should be used for the subtraction algorithm).

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

→ **When the signs of A and B are same**, add the two magnitudes and attach the sign of result is that of A.

→ **When the signs of A and B are not same**, 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 will be positive.

## <u>4.1.2 Hardware Implementation for Signed-Magnitude Data:</u>



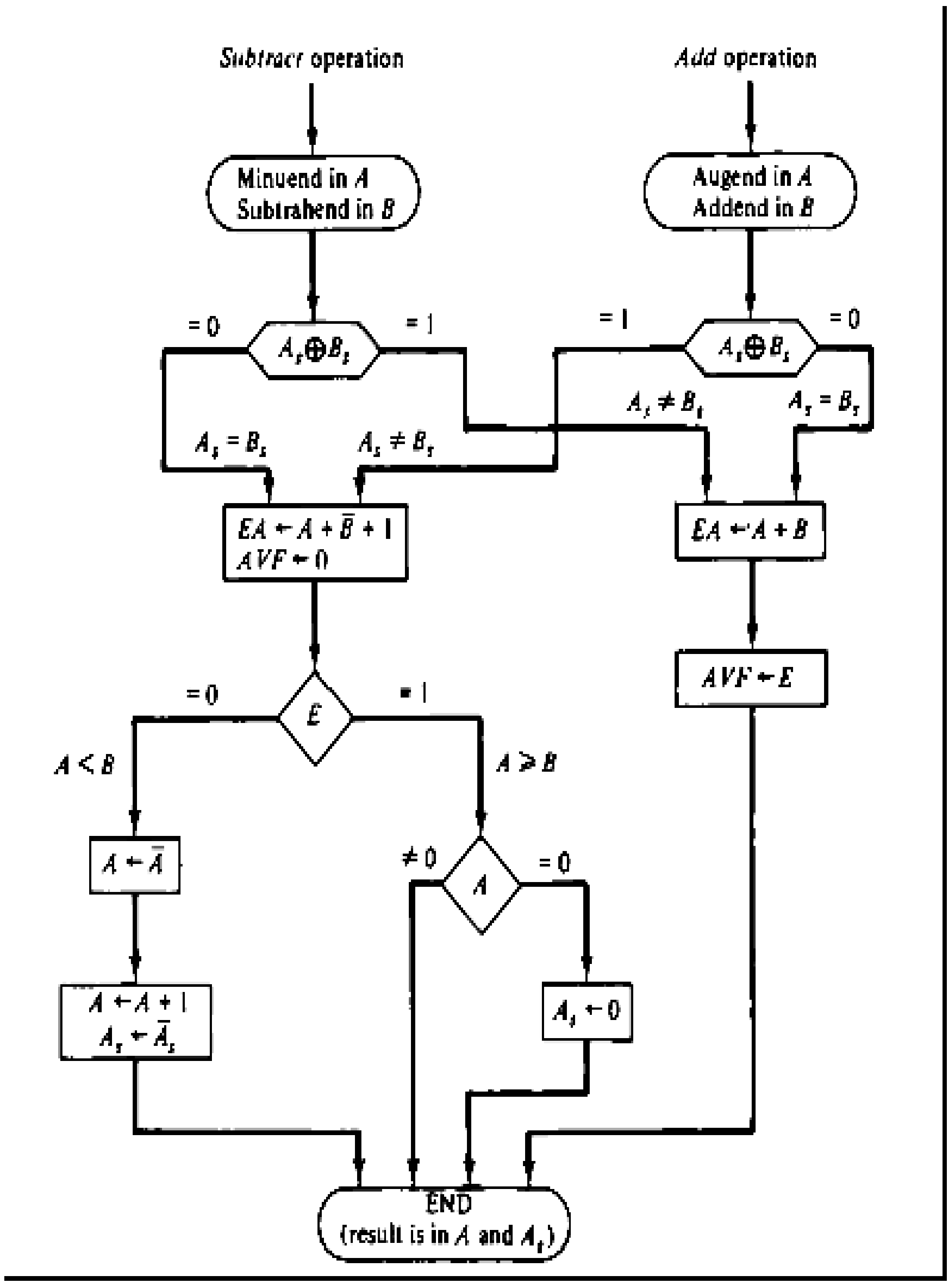
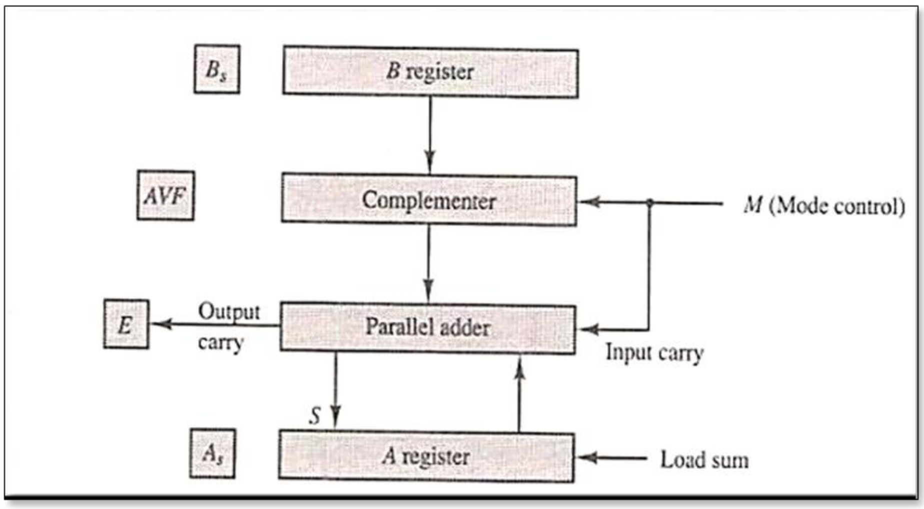
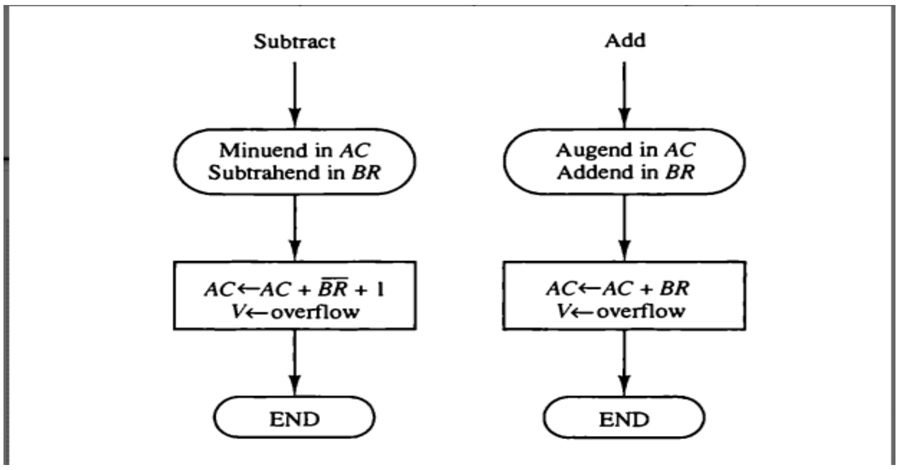


**Fig: Flowchart for addition and subtraction**

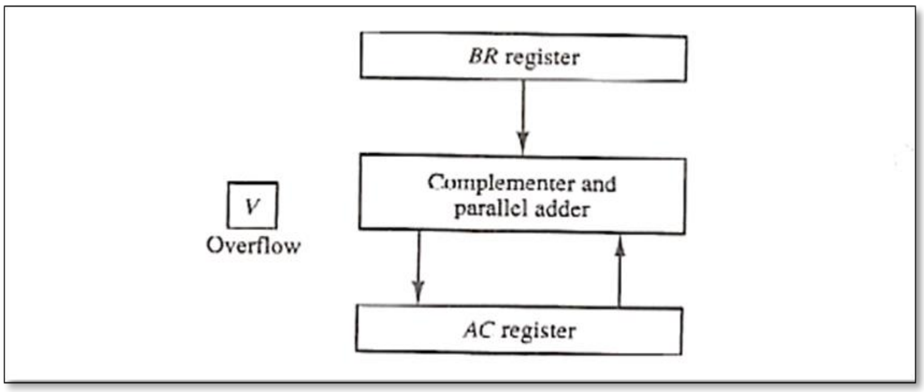
## **4.1.3 Hardware Implementation for Signed-Magnitude Data:**



4.1.4 Algorithm for adding and subtracting numbers in signed 2'scomplement



## **4.1.5 Addition and Subtraction with signed 2's Complement Data:**



## 4.2 MULTIPLICATION ALGORITHMS:

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

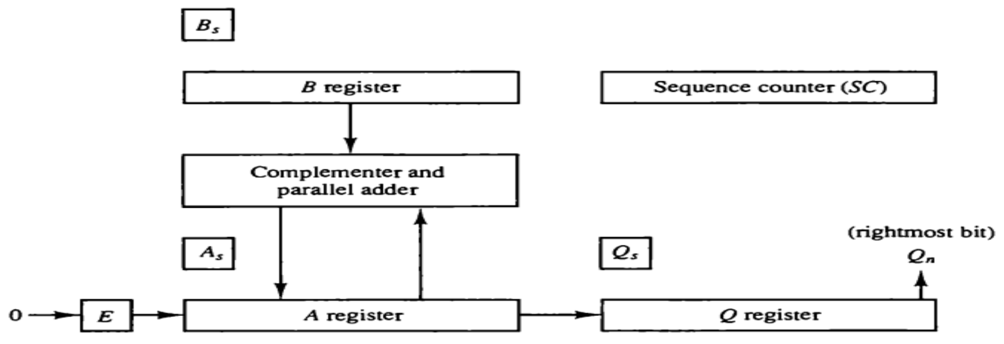
```
    23      10111 Multiplicand
    19  x   10011 Multiplier
   ----------------
            10111
           10111
          00000
         00000
        10111
-------------------
437     110110101 Product
```

This process looks at successive bits of the multiplier, least significant bit first. If the multiplier bit is 1, the multiplicand is copied as it is; otherwise, we copy zeros. Now we shift numbers copied down one position to the left from the previous numbers. Finally, the numbers are added and their sum produces the product.

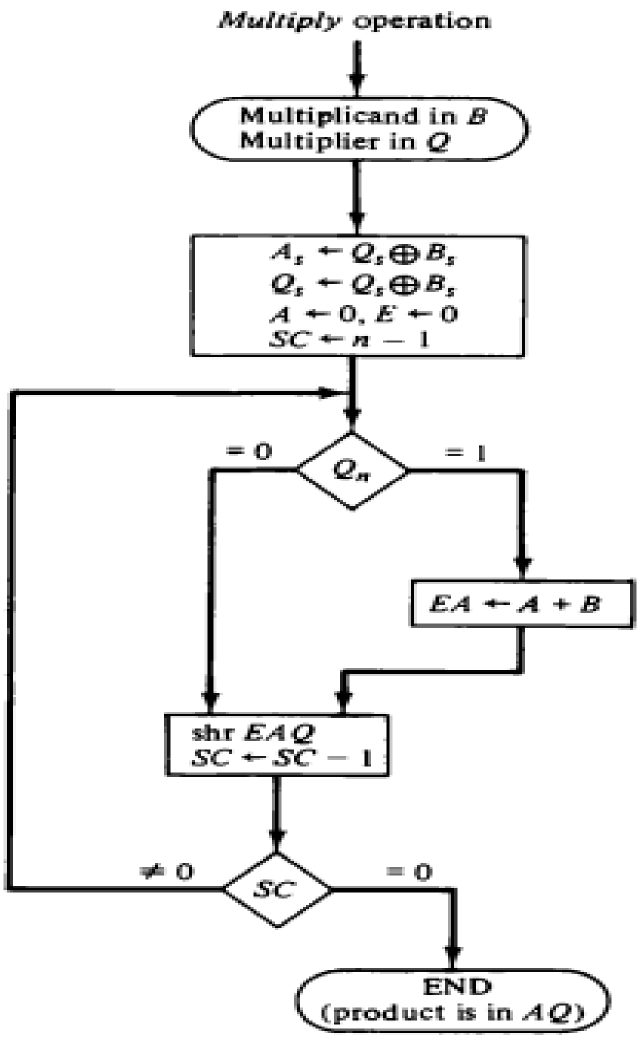
Hardware Implementation for Signed-Magnitude Data:

When multiplication is implemented in a digital computer, we change the process slightly. Here, instead of providing registers to store and add simultaneously as many binary numbers as there are bits in the multiplier, it is convenient to provide an adder for the summation of only two binary numbers, and successively accumulate the partial products in a register. Second, instead of shifting the multiplicand to left, the partial product is shifted to the right, which results in leaving the partial product and the multiplicand in the required relative positions. Now, when the corresponding bit of the multiplier is 0, there is no need to add all zeros to the partial product since it will not alter its value.

The hardware for multiplication consists of the equipment given in Figure .The multiplier is stored in the register and its sign in Qs. The sequence counter SC is initially set bits in the multiplier. After forming each partial product the counter is decremented. When the content of the counter reaches zero, the product is complete and we stop the process.



| Multiplicand $B = 10111$ | $E$ | $A$ | $Q$ | $SC$ |
|---|---|---|---|---|
| Multiplier in $Q$ | 0 | 00000 | 10011 | 101 |
| $Q_n = 1$; add $B$ | | $\underline{10111}$ | | |
| First partial product | 0 | 10111 | | |
| Shift right $EAQ$ | 0 | 01011 | 11001 | 100 |
| $Q_n = 1$; add $B$ | | $\underline{10111}$ | | |
| 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 $B$ | | $\underline{10111}$ | | |
| Fifth partial product | 0 | 11011 | | |
| Shift right $EAQ$ | 0 | 01101 | 10101 | 000 |
| Final product in $AQ = 0110110101$ | | | | |



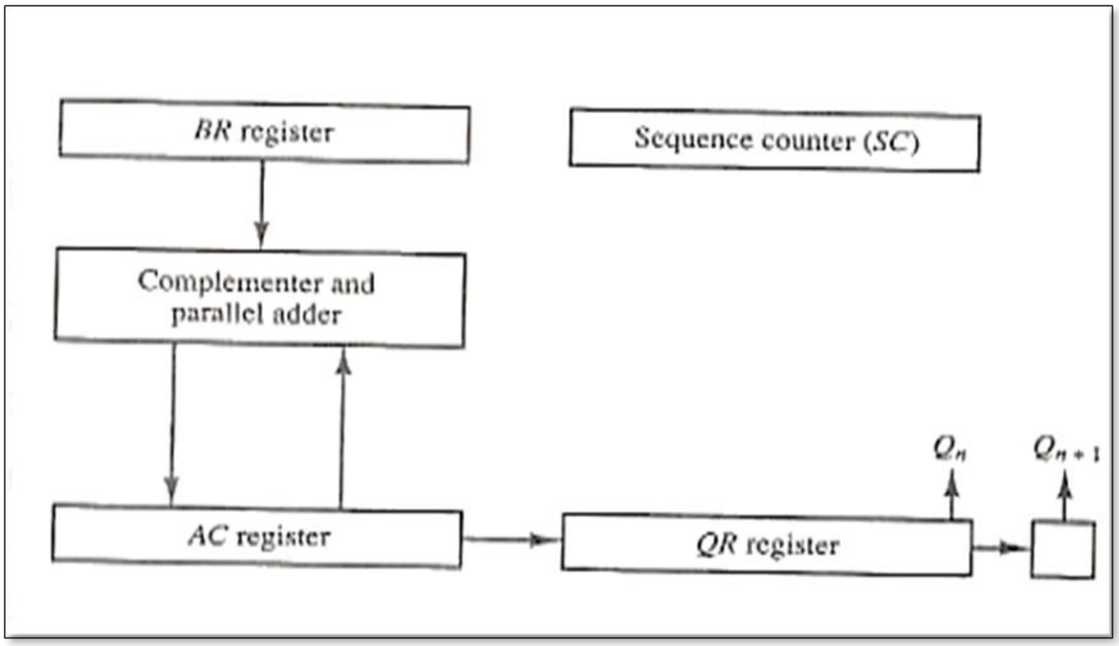
## 4.2.1 Booth Multiplication Algorithm:

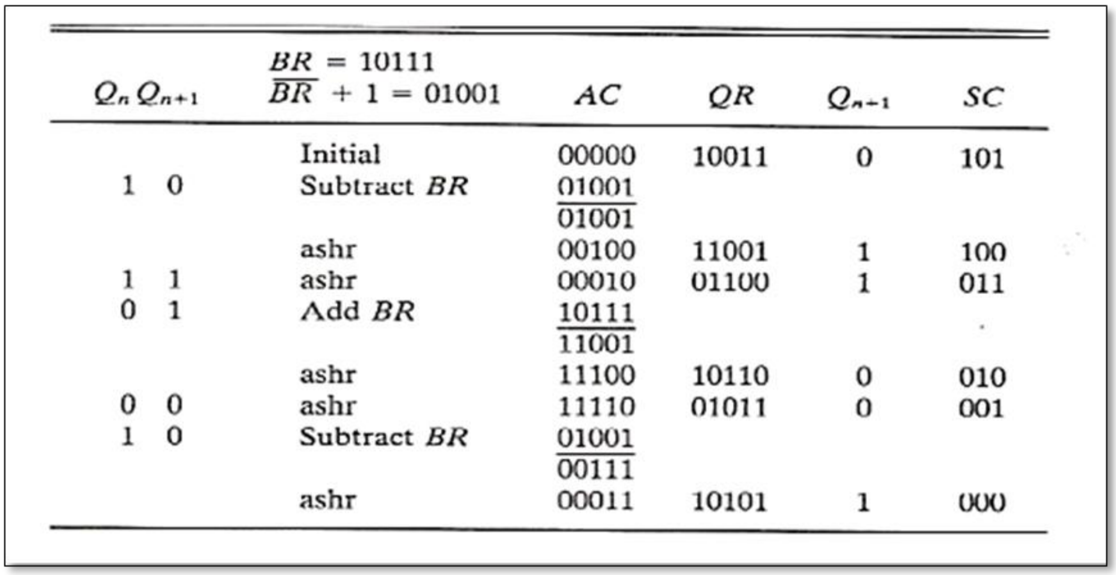
If the numbers are represented in signed 2's complement then we can multiply them by using Booth algorithm.

The hardware implementation of Booth algorithm requires the register configuration shown in Fig. Qn represents the least significant bit of the multiplier in register QR. An extra flip-flop Qn+1 is appended to QR to provide a double bit inspection of the multiplier.

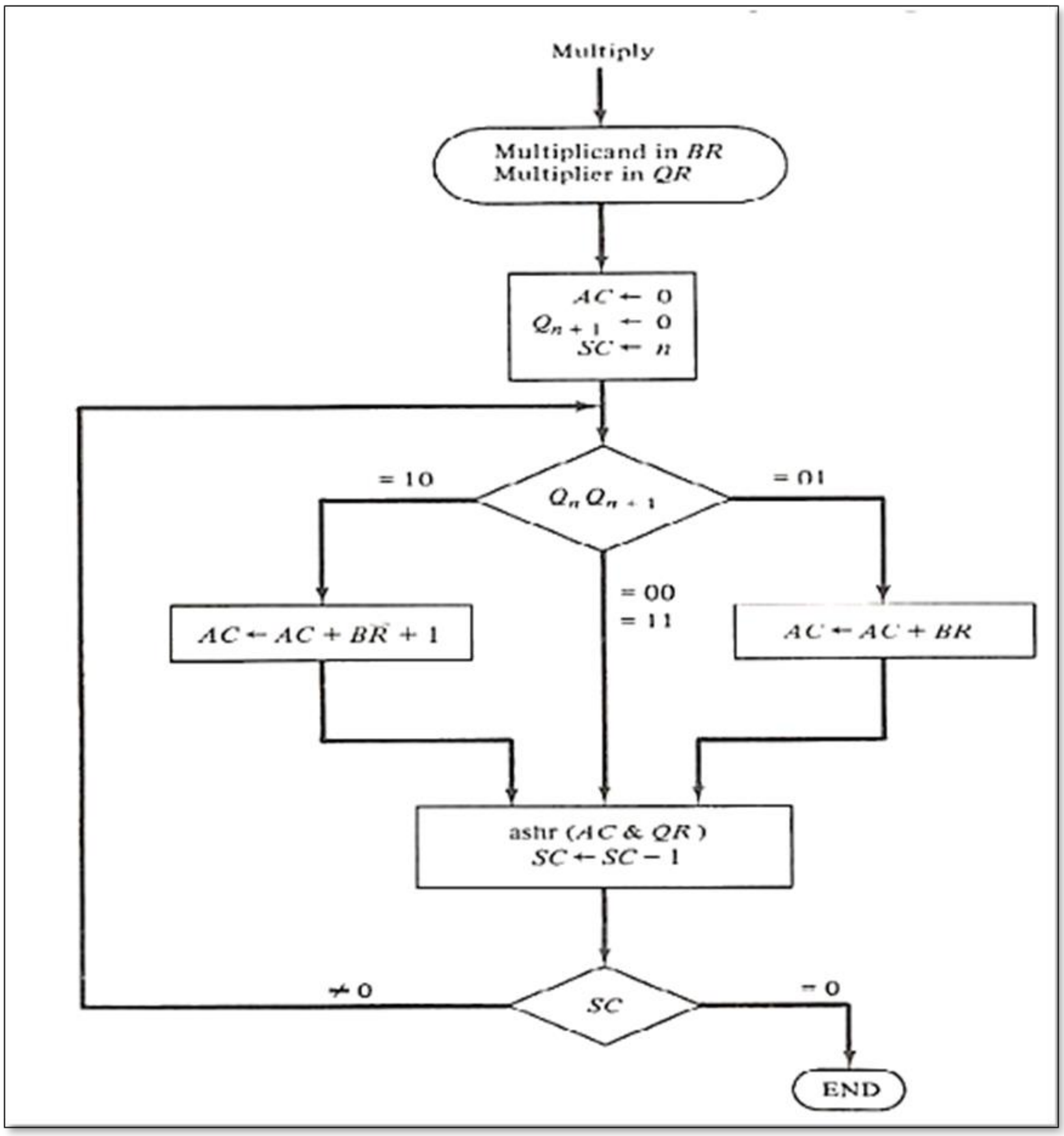
The flowchart for Booth algorithm is shown in Figure 4.7(b). AC and the appended bit Qn+1 are initially set to 0 and the sequence counter SC is set to a number n equal to the number of bits in the multiplier. The two bits of the multiplier in Qn and Qn+1 are inspected. If the two bits are 10, it means that the first 1 in a string of 1's has been encountered. This needs a subtraction of the multiplicand from the partial product in AC. If the two bits are equal to 01. It means that the first 0 in a string of 0's has been encountered. This needs the addition of the multiplicand to the partial product in AC. When the two bits are equal, the partial product does not change. An overflow cannot occur because the addition and subtraction of the multiplicand follow each other. Hence, the two numbers that are added always have opposite sign, a condition that excludes an overflow. Next step is to shift right the partial product and the multiplier (including bit Qn+1). This is an arithmetic shift right (ashr) operation which shifts AC and QR to the right and leaves the sign bit in AC same The sequence counter decrements and the computational loop is repeated n times.

A numerical example of Booth algorithm is given in Table 4.3 for n =5. It gives the multiplication of (-9) x (-13) = +117. Note that the multiplier in QR is negative and that the multiplicand in BR is also negative. The 10-bit product appears in AC. The final value of Qn+1 is the original sign bit of the multiplier and should not be taken as part of the product.



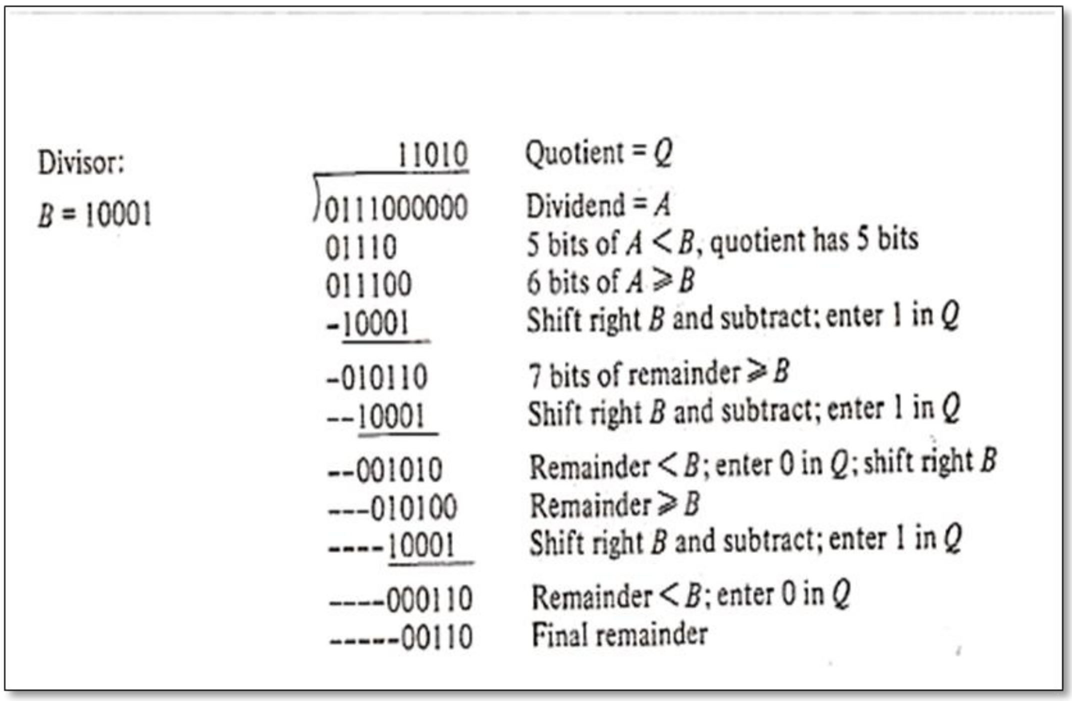


| $Q_n Q_{n+1}$ | $BR = 10111$<br>$\overline{BR} + 1 = 01001$ | $AC$ | $QR$ | $Q_{n+1}$ | $SC$ |
|---|---|---|---|---|---|
| | Initial | 00000 | 10011 | 0 | 101 |
| 1 0 | Subtract $BR$ | $\underline{01001}$ | | | |
| | | 01001 | | | |
| | ashr | 00100 | 11001 | 1 | 100 |
| 1 1 | ashr | 00010 | 01100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



## 4.3 DIVISION ALGORITHMS

Division of two fixed-point binary numbers in signed magnitude representation is performed with paper and pencil by a process of successive compare, shift and subtract operations. Binary division is much simpler than decimal division because here the quotient digits are either 0 or 1 and there is no need to estimate how many times the dividend or partial remainder fits into the divisor. The division process is described in Figure 4.10. The divisor B has five bits and the dividend A has ten.



```
Divisor:              11010      Quotient = Q
B = 10001          )0111000000   Dividend = A
                    01110        5 bits of A < B, quotient has 5 bits
                    011100       6 bits of A ≥ B
                    -10001       Shift right B and subtract: enter 1 in Q
                    -010110      7 bits of remainder ≥ B
                    --10001      Shift right B and subtract; enter 1 in Q
                    --001010     Remainder < B; enter 0 in Q; shift right B
                    ---010100    Remainder ≥ B
                    ----10001    Shift right B and subtract; enter 1 in Q
                    ----000110   Remainder < B; enter 0 in Q
                    -----00110   Final remainder
```

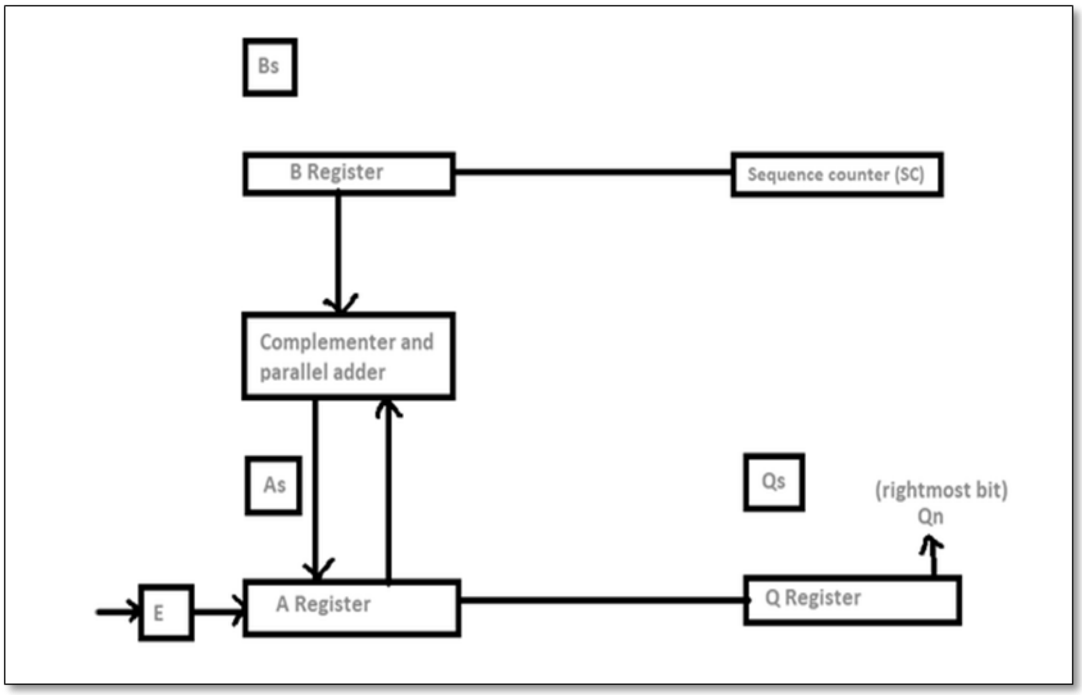
The devisor is compared with the five most significant bits of the dividend. Since the 5-bit number is smaller than B, we again repeat the same process. Now the 6-bit number is greater than B, so we place a 1 for the quotient bit in the sixth position above the dividend. Now we shift the divisor once to the right and subtract it from the dividend.

The difference is known as a partial remainder because the division could have stopped here to obtain a quotient of 1 and a remainder equal to the partial remainder. Comparing a partial remainder with the divisor continues the process.

If the partial remainder is greater than or equal to the divisor, the quotient bit is equal to 1. The divisor is then shifted right and subtracted from the partial remainder.

If the partial remainder is smaller than the divisor, the quotient bit is 0 and no subtraction is needed. The divisor is shifted once to the right in any case. Obviously the result gives both a quotient and a remainder.

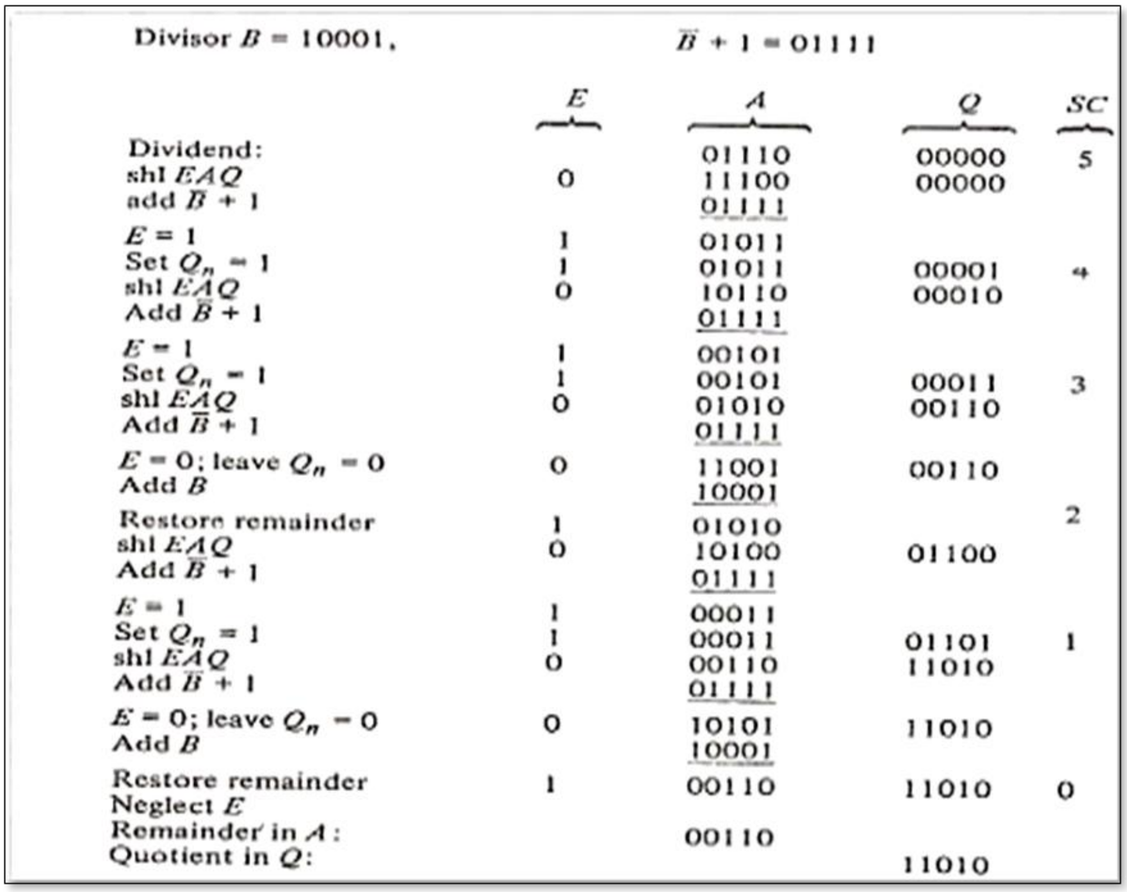
### 4.3.1 Hardware Implementation for Signed-Magnitude Data:



In hardware implementation for signed-magnitude data in a digital computer, it is convenient to change the process slightly. Instead of shifting the divisor to the right, two dividends, or partial remainders, are shifted to the left, thus leaving the two numbers in the required relative position. Subtraction is achieved by adding A to the 2's complement of B. End carry gives the information about the relative magnitudes.

The hardware required is identical to that of multiplication. Register EAQ is now shifted to the left with 0 inserted into Qn and the previous value of E is lost. The example is given in Figure 4.10 to clear the proposed division process. The divisor is stored in the B register and the double- length dividend is stored in registers A and Q. The dividend is shifted to the left and the divisor is subtracted by adding its 2's complement value.

The register E keeps the information about the relative magnitude. A quotient bit 1 is inserted into Qn and the partial remainder is shifted to the left to repeat the process when E = 1. If E =0, it signifies that A < B so the quotient in Qn remains a 0 (inserted during the shift). To restore the partial remainder in A the value of B is then added to its previous value. The partial remainder is shifted to the left and the process is repeated again until we get all five quotient-bits. Note that while the partial remainder is shifted left, the quotient bits are shifted also and after five shifts, the quotient is in Q and A has the final remainder. Before showing the algorithm in flowchart form, we have to consider the sign of the result and a possible overflow condition. The sign of the quotient is obtained from the signs of the dividend and the divisor. If the two signs are same, the sign of the quotient is plus. If they are not identical, the sign is minus. The sign of the remainder is the same as that of the dividend.



Divisor $B$ = 10001, $\overline{B} + 1$ = 01111

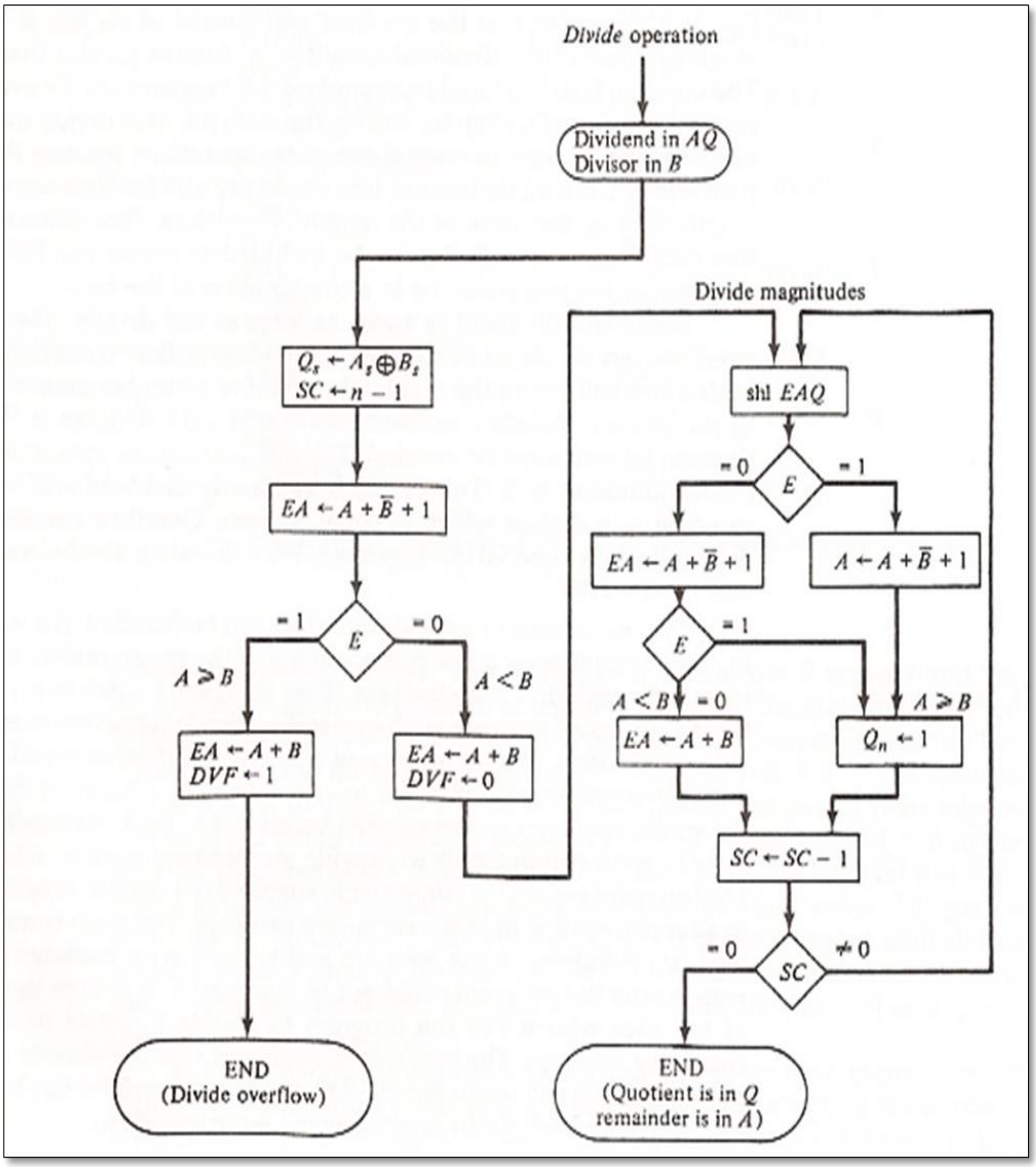
| | $E$ | $A$ | $Q$ | $SC$ |
|---|---|---|---|---|
| Dividend: | | 01110 | 00000 | 5 |
| shl $EAQ$ | 0 | 11100 | 00000 | |
| add $\overline{B} + 1$ | | 01111 | | |
| $E = 1$ | 1 | 01011 | | |
| Set $Q_n = 1$ | 1 | 01011 | 00001 | 4 |
| shl $EAQ$ | 0 | 10110 | 00010 | |
| Add $\overline{B} + 1$ | | 01111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

## 4.3.2 Hardware Algorithm:

Figure 4.6 is a flowchart of the hardware multiplication algorithm. In the beginning, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in Bs and Qs respectively. We compare the signs of both A and Q and set to corresponding 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 the number of bits of the multiplier. Since an operand must be stored with its sign, one bit of the word will be occupied by the sign and the magnitude will consist of n-1 bits.

Now, the low order bit of the multiplier in Qn is tested. If it is 1, the multiplicand (B) is added to present partial product (A), 0 otherwise. 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. When SC = 0 we stops the process.

The hardware divide algorithm is given in Figure A and Q contain the dividend and B has the divisor. The sign of the result is transferred into Q. A constant is set into the sequence counter SC to specify the number of bits in the quotient. As in multiplication, we assume that operands are transferred to registers from a memory unit that has words of n bits. Since an operand must be stored with its sign, one bit of the word will be occupied by the sign and the magnitude will have n-1 bits. We can check a divide-overflow condition by subtracting the divisor (B) from half of the bits of the dividend stored (A). If A<B, the divide-overflow occur and the operation is terminated. If A≥ B, no divide overflow occurs and so the value of the dividend is restored by adding B to A.



Divide operation
Dividend in AQ
Divisor in B
$Q_s \leftarrow A_s \oplus B_s$
$SC \leftarrow n - 1$
$EA \leftarrow A + \overline{B} + 1$
E
= 1
= 0
A ⩾ B
A < B
$EA \leftarrow A + B$
$DVF \leftarrow 1$
$EA \leftarrow A + B$
$DVF \leftarrow 0$
END
(Divide overflow)
Divide magnitudes
shl EAQ
E
= 0
= 1
$EA \leftarrow A + \overline{B} + 1$
$A \leftarrow A + \overline{B} + 1$
E
= 1
A < B
= 0
A ⩾ B
$EA \leftarrow A + B$
$Q_n \leftarrow 1$
$SC \leftarrow SC - 1$
SC
= 0
≠ 0
END
(Quotient is in Q
remainder is in A)

# Unit-4(Part-B)
# I/O Organization

## Computer Peripherals

→ The computer peripherals are devices. We connect the peripheral devices externally to the computer but actually not part of the computer system.

→ The best examples for peripherals devices are the printer, monitor, keyboard, touchpad, joystick, and many more. These are not the core elements in computer architecture.

→ The **central processing unit(CPU),** motherboard, power supply are comes under the core elements. Aside from these things, every element in the computer comes under the peripheral devices. The computer also needs more elements for the functioning of the elements in it like hard drive, ram, and Rom. We connect the peripheral devices to the computer to expand the operations and different functionality.

→ Let us discuss some of the **functionalities of the peripheral device**. Let's take the printer, we connect the printer externally to print the documents or files that we stored on our computer devices. All the peripheral devices depend on the computer system. The Peripheral devices can be subdivided into three categories. They are Input devices, output devices, and the last category is storage devices.

→ The **input devices** are used to take the input from the user. The mouse keyboard is an example of the input devices. The output devices take the input from the input devices and result from the output. Monitor and printer are the best examples for the output devices.

→ The **storage device** stores the information that given by the user. flash drive, the hard disk is the example for the storage devices. But there are some devices that come under more than one of these categories. For example CD-ROM drive. The CD-ROM drive can be used to read the data by acting as the input devices and can write the data while acting as the output devices.

→ The peripheral device can be of the external and the internal types., printer, scanner, etc are external devices. We connect these devices externally. The optical disc drive comes under the internal devices. These devices are located inside the computer. The peripheral devices can easily connect and disconnect.

→ We can replace the devices when it is needed. We can only maintain essential devices and can discard them when it is not necessary. But some of the peripheral devices are must necessary like a monitor. If the monitor is removed then no functionality can be done.

→ The below figure 1.1 represents all the internal and external peripheral devices are connected to the computer. The central processing unit (CPU) referred with number 2. the motherboard referred with 8. The power supply in 5. expansion slots 4. hard disk drive 7, the optical disk drive 6. Scanner 1, monitor 10, keyboard 13, mouse 14, speakers 9, and printer 16. The numbers 11 and 12 comes under the software.

→ There are many more peripherals than recently been introduced like webcam, microphone. All the internal

→ peripheral devices we directly connect to the motherboard. The motherboard is the central processor where all the operations of the input can be done using many different types of slots on the motherboard.

The **external peripheral devices can be both wired and wireless devices**. Let us study the function of each peripheral device in detail. We can get or put information in the peripheral devices. Let us discuss the functioning of all the peripheral devices in detail.

## Printer

- The printer is the external peripheral device that we connect externally to the computer. The peripheral device is only used when it is necessary. We can remove or connect the printer as of our need. The printer has the functionality to make a copy of the document, photos, or papers. We have different types of printers.

- There are **Inkjets and laser printers** are the most commonly used. We also have other printers like thermal printers and dot-matrix printers. Normally, we use the inkjet printers, we use in our homes. The printer moves back and forth across the paper. The laser prints are mostly used for small business and large business purposes.

- The **laser printer** will provide high-quality print and can come in different sizes. The thermal printers can use a special type of paper called thermal paper and has a wax- based ink and turns black when the heat is applied. The last type of printer is called the thermal printer, the thermal printer is mostly not in use. The DOT printer is an old technology that can produce mediocre print quality.

## Keyboard

- The computer keyboard is one of the external peripheral devices. We will connect externally to the computer. It is one of the most essential parts of the peripheral device. Without the keyboard, we cannot give input to perform the operations on the computer.

- **ASCII** represents American Standard Code for Information Interchange. It is the standard binary code used to represent alphanumeric characters. Alphanumeric characters are used for the transfer of information to and from the I/O devices and the computer. This standard helps seven bits to code **128 characters**. However, there is an additional bit on the left that is always assigned 0. Therefore, there are 8 bits in total.

- The ASCII code consists of **34 nonprinting characters and 94 characters used for various control operations**. There are 26 uppercase letters A through Z, 26 lowercase letters a through z, numerals from 0 to 9, and 32 printable characters including %,*.

- The control characters are used to route the data and arrange the printed text into a prescribed format.

A list of control characters is shown in the table.

**Control Character and Description**

| Control Character | Description |
|---|---|
| **NUL** | **Null** |
| **SOH** | **Start of Heading** |
| **STX** | **Start of text** |
| **EOT** | **End of the transmission** |
| **ENQ** | **Enquiry** |
| **ACK** | **Acknowledge** |
| **DLE** | **Data Link Escape** |
| **ETB** | **End of the transmission block** |
| **EM** | **End of medium** |

## Types of Control Characters

There are three types of control characters that are as follows −

**Format Effectors** − It can control the design of printing. It contains familiar typewrite controls including Back Space (BS), Horizontal Tabulation (HT), and Carriage Return (CR).

**Information Separators** − It can separate the information into divisions including paragraphs and pages. It includes Record Separator (RS) and File Separator (FS).

**Communication Control** − It can be used during the transmission of text between remote terminals.

## Mouse

- The mouse comes under the external peripheral device. It is an input device where we give input to the computer. By moving the pointer on the monitor we select the items on the monitor screen.

- On the mouse, we normally have **two buttons**. In between, there presents a **scroll** will between them. With the help of the scroll will we can move the page. The mouse can do many functions on the computer system, we can point and select items on the screen, we can draw an paint on the computer screen, we can also play several games by using the mouse.

- We can move items on the monitor from one place to another by clicking the left button on the mouse. There are different types of mouse-like scroll mouse.

- The scroll mouse has two buttons and scrolls to move up and down. Another type of mouse is the optical mouse.

- The **optical mouse** consists of red light under it. The wireless mouse is another type that will not attach to the computer, it can communicate through signals.

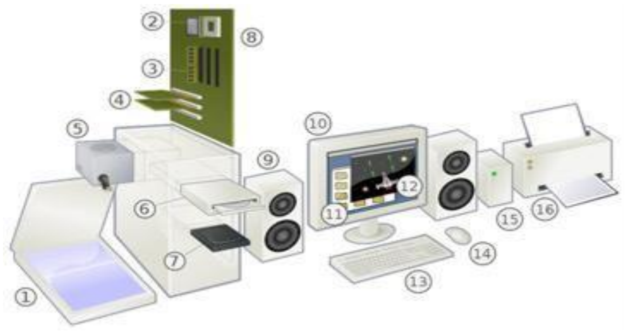


Fig. 1.1

**Hard disk**

- The Hard disk is the internal peripheral device to store the data in it. The hard disk uses magnetic storage to store the data and digital storage to retrieve the data from it. we use a magnetic material to retrieve the information more rapidly. the platters that are present on the magnetic heads where we can read and write the data on the platter surface.

- We can access the stored information by two types. Using random access memory(RAM) and read-only memory (ROM). We can access the data in any order in the random access memory. The hard disk is usually nonvolatile storage, we can retain the data after the system gets turned off.

- We have the peripheral ports. These peripheral ports are used to connect the different devices of the computer.

- We have **different types of ports in the peripheral devices like serial ports, parallel ports**, USB ports, and many more. The ports act as the connecting points to the various peripheral devices to the computer. Let us study different types of peripheral ports.

- **The serial port's** name itself tells that the data can be transferred in a serial manner. The serial ports can be of 9 pin connector or as a 24 pin connector. The parallel ports can send the data up to 8 bits. we can send and receive the data at a time. apart from the serial ports, the parallel ports transmit the data in a parallel manner.

- **The parallel ports** can have a 25-bit connector. The main function of the parallel ports is to send and receive data in the form of bytes. at the same time, we usually use parallel ports to connect many peripheral devices like printers, scanners, hard disk drives, and many more. we can transmit the data-parallel to each other.

- we have another type of port called USB ports, the USB ports stand for universal serial bus. We use the single standardized. The USB ports are easy to connect to any number of devices. we cannot connect printer, scanner, digital cameras telephone, etc similar to the Serial and the parallel ports.

- Another type of peripheral port is the ethernet port. The ethernet port opens a piece of network equipment, where we can plug ethernet cables into it. The main purpose of the Ethernet port is to connect the network hardware to the ethernet network. The ethernet cable only accepts the RJ45 connector to it.

- The last type of port is called HDMI peripheral port. The HDMI stands for High definition multimedia interface. The HDMI peripheral port is a digital connector, it can carry the high videos and the audio channels in a single cable.

- These peripheral devices can be of both internal and external devices that can provide input and output services for the functioning of the computer. We have must essential peripheral devices and nonessential peripheral devices. Keyboard, hard disk, monitor mouse are the essential peripheral devices. The printer, scanner is the nonessential peripheral devices.

- The external peripheral is used only when it is required. We can remove and attach the external device according to the user's need. Unlike core elements, all components in the device come under the peripheral devices.

- Motherboard, random access memory, central processing unit, graphical processing unit are the core components of the computer.

## Input-Output Interface

Input-output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral.

**The major differences are:**

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.

2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be need.

3. Data codes and formats in peripherals differ form the word format in the CPU and memory.

4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

Interface units are used because they interface between the processor bus and the peripheral device. In addition, each device may have its own controller that supervises the operations of the particular mechanism in the peripheral.

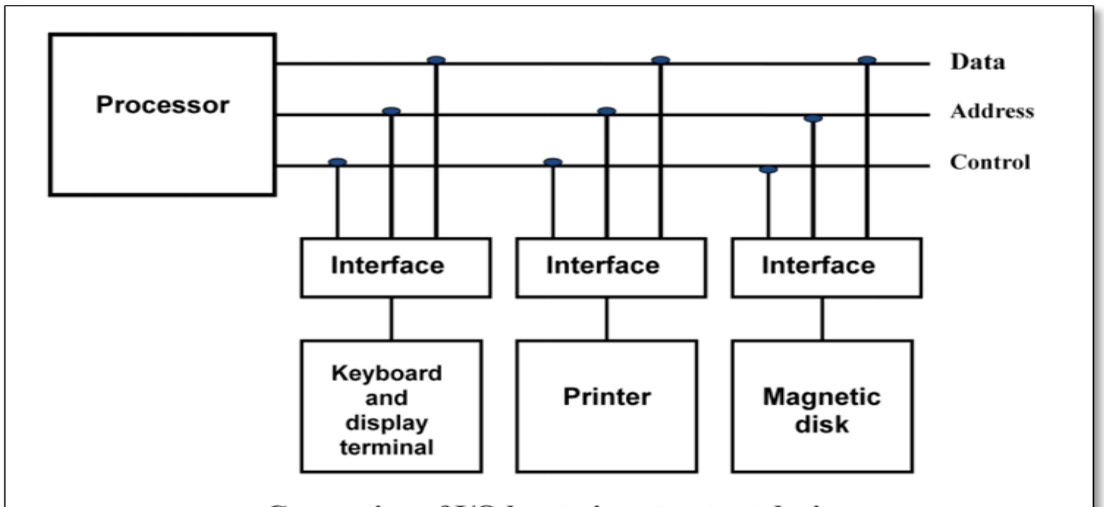
Two main types of interface are CPU interface that corresponds to the system bus and input-output interface that depends on the nature of input- output device.

## 4.7.1. I/O Bus and Interface Modules

- A typical communication link between the processor and several peripherals is shown in Fig below. I/O bus consists of data lines, address lines, and control lines. The magnetic disk, printer, and terminal are employed in practically any general-purpose computer. The magnetic tape is used in some computers for backup storage. Each peripheral device has associated with it an interface unit.

- Each interface decodes the address and control received from the I/O bus, interprets them for the peripheral, and provides signals for the peripheral controller. Each peripheral has its own controller that operates the particular electromechanical device.

- The I/O bus from the processor is attached to all peripheral interfaces. To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines.



Connection of I/O bus to input-output devices

All peripherals whose address does not correspond to the address in the bus are disabled their interface

- The interface selected responds to the function code and proceeds to execute it. The function code is referred to as an I/O command and is in essence an instruction that is executed in the interface and its attached peripheral unit.

- There are four types of commands that an interface may receive. They are classified as control, status, status, data output, and data input.

- **A control command** is issued to activate the peripheral and to inform it what to do. For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction. The particular control command issued depends on the peripheral, and each peripheral receives its own distinguished sequence of control commands, depending on its mode of operation.

- **A status command** is used to test various status conditions in the interface and the peripheral.

- **A data output command** causes the interface to respond by transferring data from the bus into one of its registers.

- **The data input command** is the opposite of the data output. In this case the interface receives an item of data from the peripheral and places it in its buffer register

### 4.7.2. I/O versus Memory Bus

In addition to communicating with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address, and read/write control lines. There are three ways that computer buses can be used to communicate with memory and I/O:

1. Use two separate buses, one for memory and the other for I/O.

2. Use one common bus for both memory and I/O but have separate control lines for each.

3. Use one common bus for memory and I/O with common control lines.

In the first method, the computer has independent sets of data, address, and control buses, one for accessing memory and the other for I/O. This is done in computers that provide a separate I/O processor (IOP) in addition to the central processing unit (CPU).

The purpose of the IOP is to provide an independent pathway for the transfer of information between external devices and internal memory.

### 4.7.3. Isolated Versus Memory-Mapped I/O

- Many computers use one common bus to transfer information between memory or I/O and the CPU. The distinction between a memory transfer and I/O transfer is made through separate read and write lines.
- The CPU specifies whether the address on the address lines is for a memory word or for an interface register by enabling one of two possible read or write lines. The I/O read and I/O write control lines are enabled during an I/O transfer.

- In the isolated I/O configuration, the CPU has distinct input and output instructions, and each of these instructions is associated with the address of an interface register.

- When the CPU fetches and decodes the operation code of an input or output instruction, it places the address associated with the instruction into the common address lines. At the same time, it enables the I/O read (for input) or I/O write (for output) control line.

- The isolated I/O method isolates memory word and not for an I/O addresses so that memory address values are not affected by interface address assignment since each has its own address space.

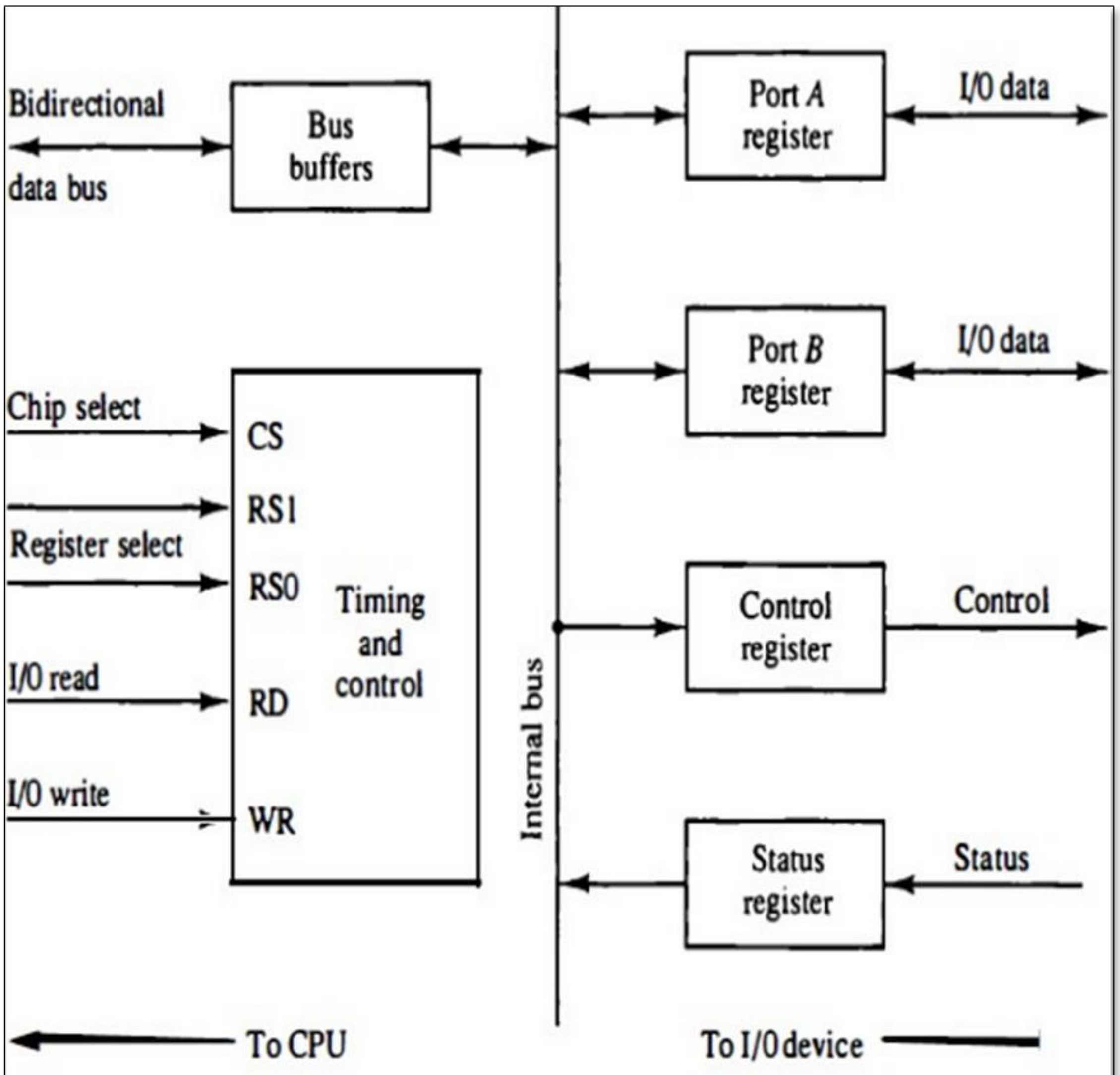
- The other alternative is to use the same address space for both memory and I/O. This is the case in computers that employ only one set of read and write signals and do not distinguish between memory and I/O addresses. This configuration is referred to as memory-mapped I/O.

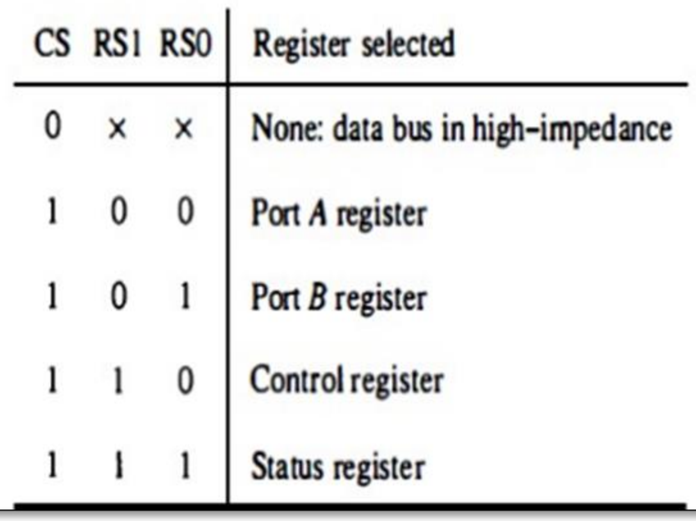
- In a memory-mapped I/O organization there are no specific input or output instructions. The CPU can manipulate I/O data residing in interface registers with the same instructions that are used to manipulate memory words.

- Each interface is organized as a set of registers that respond to read and write requests in the normal address space.

### 4.7.4. Example of I/O Interface

An example of an I/O interface unit is shown in block diagram form in Fig below.





| CS | RS1 | RS0 | Register selected |
|---|---|---|---|
| 0 | × | × | None: data bus in high-impedance |
| 1 | 0 | 0 | Port *A* register |
| 1 | 0 | 1 | Port *B* register |
| 1 | 1 | 0 | Control register |
| 1 | 1 | 1 | Status register |

- It consists of two data registers called ports, a control register, a status register, bus buffers, and timing and control circuits. The interface communicates with the CPU through the data bus. The chip select and register select inputs determine the address assigned to the interface.

- The I/O read and write are two control lines that specify an input or output, respectively. The four registers communicate directly with the I/O device attached to the interface.

- The I/O data to and from the device can be transferred into either port A or Port B. The interface may operate with an output device or with an input device, or with a device that requires both input and output. If the interface is connected to a printer, it will only output data, and if it services a character reader, it will only input data.

- A magnetic disk unit transfers data in both directions but not at the same time, so the interface can use bidirectional lines. A command is passed to the I/O device by sending a word to the appropriate interface register.

- In a system like this, the function code in the I/O bus is not needed because control is sent to the control register, status information is received from the status register, and data are transferred to and from ports A and B registers. Thus the transfer of data, control, and status information is always via the common data bus.

- The distinction between data, control, or status information is determined from the particular register with which the CPU communicates.

- The interface registers communicate with the CPU through the bidirectional data bus. The address bus selects the interface unit through the chip select and the two register select inputs.

- A circuit must be provided externally (usually, a decoder) to detect the address assigned to the interface registers. This circuit enables the chip select (CS) input when the interface is selected by the address bus.

- The two register select inputs RS1 and RS0 are usually connected to the two least significant lines of the lines address bus. These two inputs select one of the four registers in the interface as specified in the table accompanying the diagram.

- The content of the selected register is transfer into the CPU via the data bus when the I/O read signal is enabled. The CPU transfers binary information into the selected register via the data bus when the I/O write input is enabled.

## **4.8. Asynchronous Data Transfer**

- The internal operations in a digital system are synchronized by means of clock pulses supplied by a common pulse generator.
- Clock pulses are applied to all registers within a unit and all data transfers among internal registers occur simultaneously during the occurrence of a clock pulse. Two units, such as a CPU and an I/O interface, are designed independently of each other.
- If the registers in the interface share a common clock with the CPU registers, the transfer between the two units is said to be synchronous.
- In most cases, the internal timing in each unit is independent from the other in that each uses its own private clock for internal registers. In that case, the two units are said to be asynchronous to each other. This approach is widely used in most computer systems.
- Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted. One way of achieving this is by means of a strobe pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.
- Another method commonly used is to accompany each data item being transferred with a control signal that indicates the presence of data in the bus.
- The unit receiving the data item responds with another control signal to acknowledge receipt of the data. This type of agreement between two independent units is referred to as handshaking.
- The strobe pulse method and the handshaking method of asynchronous data transfer are not restricted to I/O transfers. In fact, they are used extensively on numerous occasions requiring the transfer of data between two independent units.
- In the general case we consider the transmitting unit as the source and the receiving unit as the destination.
- For example, the CPU is the source unit during an output or a write transfer and it is the destination unit during an input or a read transfer. It is customary to specify the asynchronous transfer between two independent units by means of a timing diagram that shows the timing relationship that must exist between the control signals and the data in buses.
- The sequence of control during an asynchronous transfer depends on whether the transfer is initiated by the source or by the destination unit.

**There are two types of asynchronous data transmission methods**

1. Strobe control
2. Handshaking

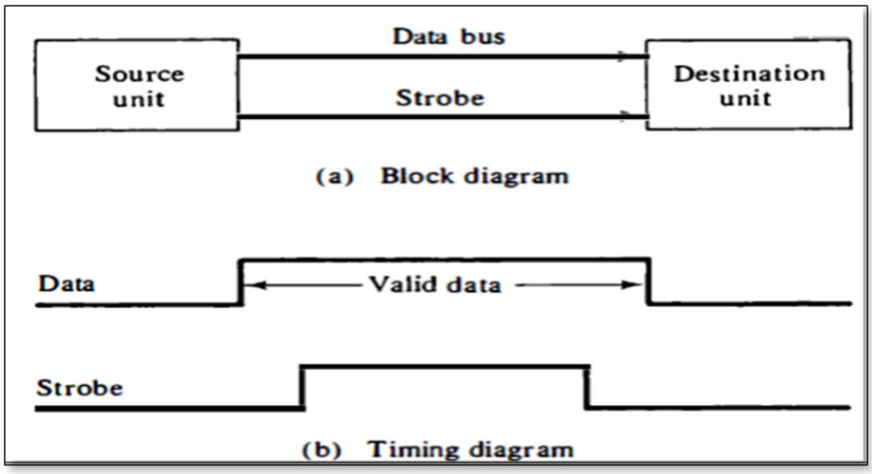
## 4.8.1. Strobe Control

The strobe control method of asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source or the destination unit.

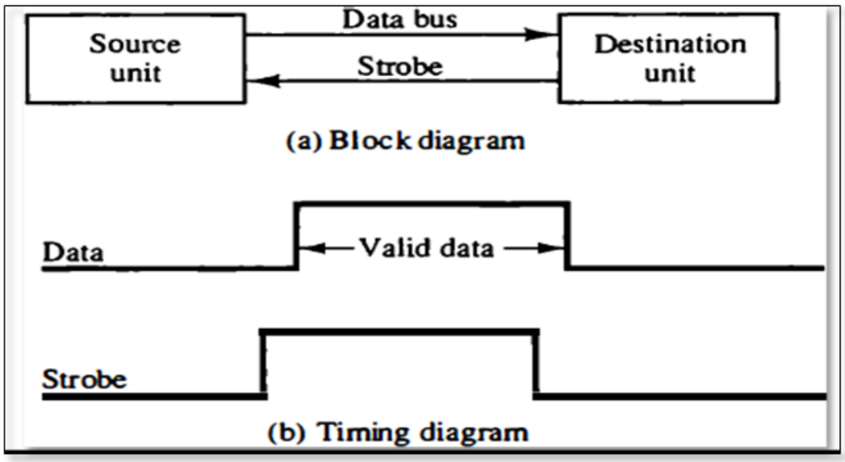
### i. source-initiated transfer.

- The data bus carries the binary information from source unit to the destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available in the bus.

- The strode signal is given after a brief delay, after placing the data on the data bus. A brief period after the strobe pulse is disabled the source stops sending the data.



### ii. Destination-initiated strobe for data transfer



- In this case the destination unit activates the strobe pulse, informing the source to provide the data. The source places the data on the data bus. The transmission id stopped briefly after the strobe pulse is removed.

- The **disadvantage of the strobe method** is that the source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was placed in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on the bus. This difficulty is solved by using hand shaking method of data transfer.

## 4.8.2. HANDSHAKING

- The disadvantage of the strobe method is that the source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was placed in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on the bus.

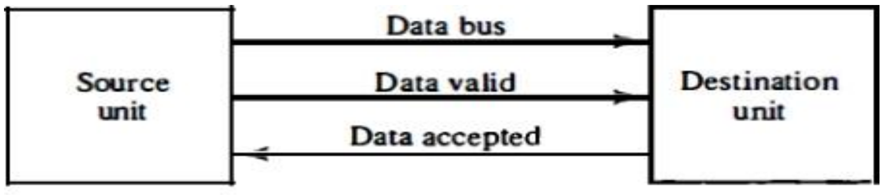
- The handshake method solves this problem by introducing a second control signal that provides a reply to the unit that initiates the transfer. The basic principle of the two-write handshaking method of data transfer is as follows.

- One control line is in the same direction as the data flow in the bus from the source to the destination. It is used by the source unit to inform the destination unit whether there are valued data in the bus.

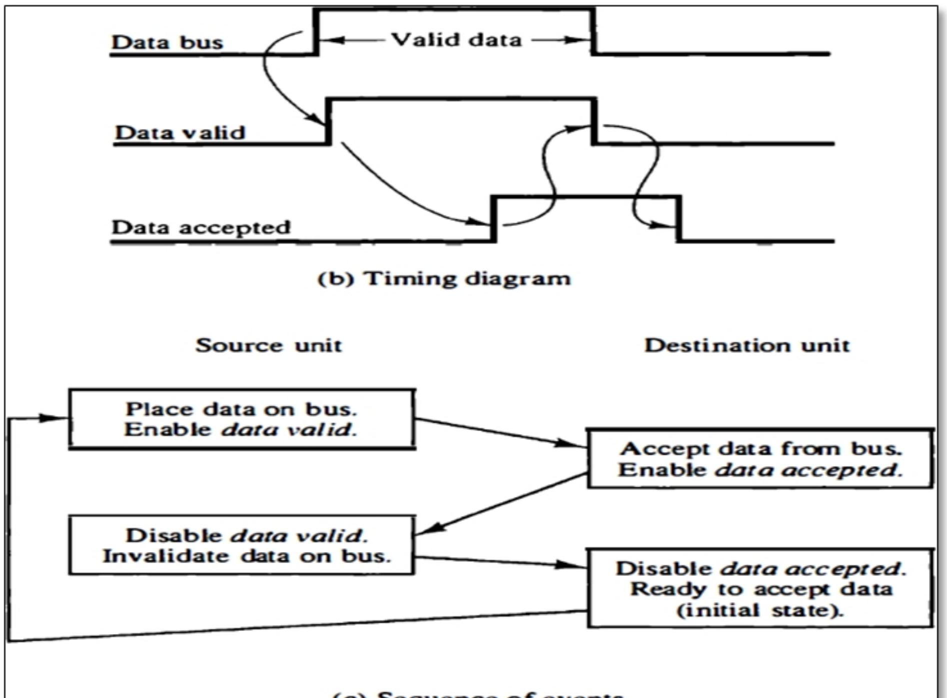
- The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept data. The sequence of control during the transfer depends on the unit that initiates the transfer.

**Data Transfer Procedure When Initiated By the Source**

The two handshaking lines are data valid, which is generated by the source unit, and data accepted, generated by the destination unit. The timing diagram shows the exchange of signals between the two units.



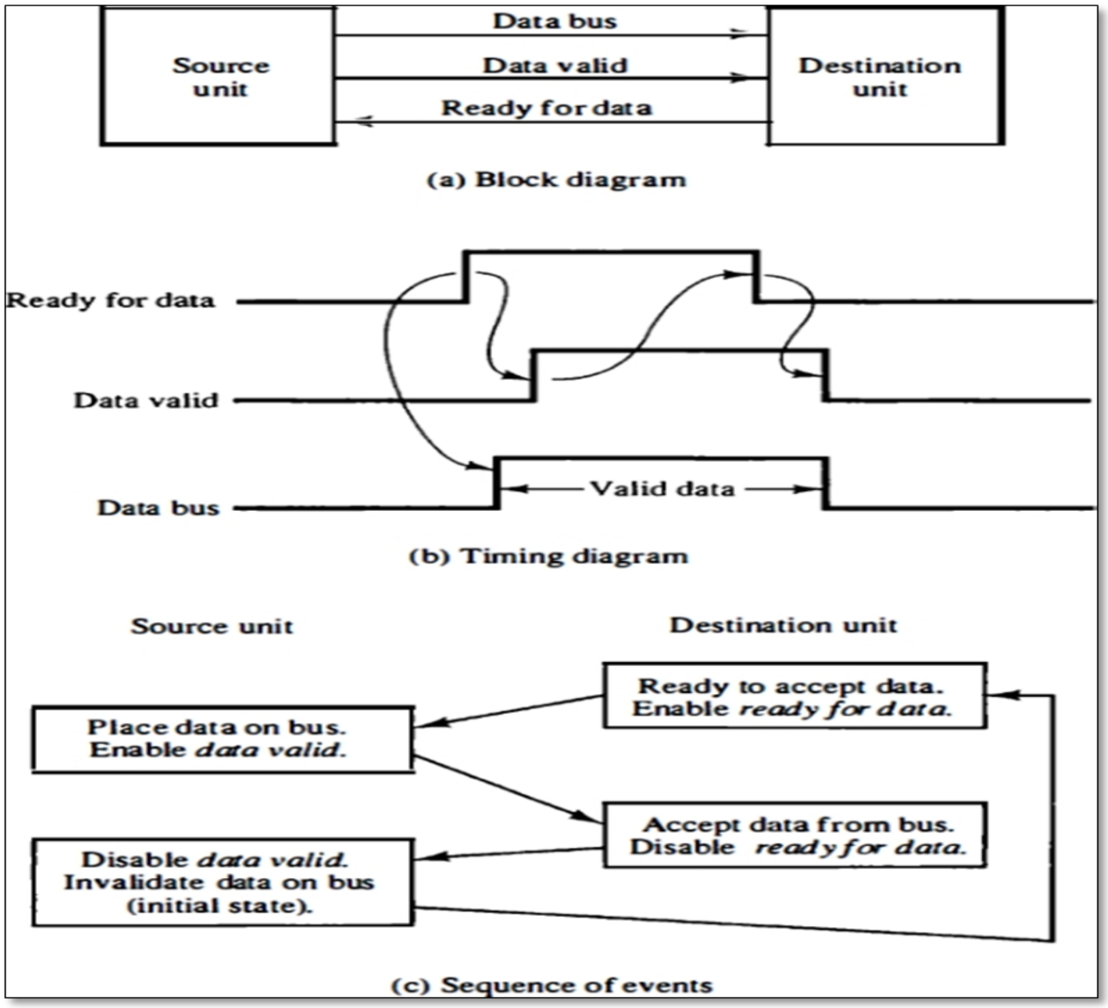
(a) Block diagram


Data bus
Valid data
Data valid
Data accepted
(b) Timing diagram
Source unit
Destination unit
Place data on bus. Enable *data valid*.
Accept data from bus. Enable *data accepted*.
Disable *data valid*. Invalidate data on bus.
Disable *data accepted*. Ready to accept data (initial state).

(c) Sequence of events

The sequence of events listed in part (c) shows the four possible states that the system can be at any given time. The source unit initiates the transfer by placing the data on the bus and enabling its data valid signal. The data accepted signal is activated by the destination unit after it accepts the data from the bus. The source unit then disables its data valid signal, which invalidates the data on the bus.

**Destination -Initiated Transfer Using Handshaking**


Source unit
Data bus
Data valid
Ready for data
Destination unit
(a) Block diagram
Ready for data
Data valid
Data bus
Valid data
(b) Timing diagram
Source unit
Destination unit
Ready to accept data. Enable *ready for data.*
Place data on bus. Enable *data valid.*
Accept data from bus. Disable *ready for data.*
Disable *data valid.* Invalidate data on bus (initial state).
(c) Sequence of events

The destination unit then disables its data accepted signal and the system goes into its initial state. The source dies not send the next data item until after the destination unit shows its readiness to accept new data by disabling its data accepted signal. This scheme allows arbitrary delays from one state to the next and permits each unit to respond at its own data transfer rate. The rate of transfer is determined by the slowest unit.

If one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a timeout mechanism, which produces an alarm if the data transfer is not completed within a predetermined time.

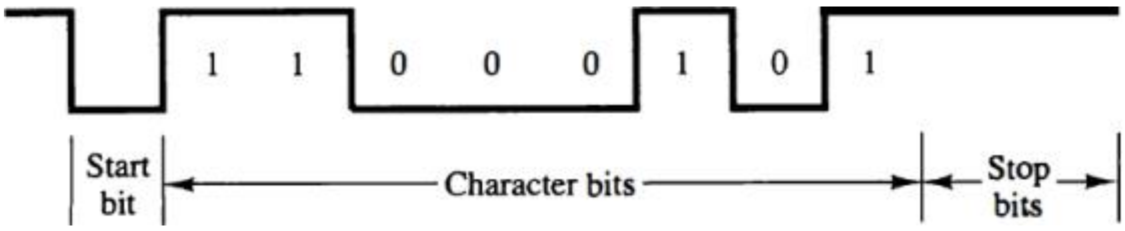
## 4.8.3. Asynchronous Serial Transfer

- The transfer of data between two units may be done in parallel or serial. In parallel data transmission, each bit of the message has its own path and the total message is transmitted at the same time.

- Serial transmission can be can be **synchronous or asynchronous**. In synchronous transmission, the two units share a common clock frequency and bits are transmitted continuously at the rate dictated by the clock pulses. In long-distant serial transmission, each unit is driven by a separate clock of the same frequency. Synchronization signals are transmitted periodically between the two units to keep their clocks in step with each other.

- In asynchronous transmission, binary information is sent only when it is available and the line remains idle when there is no information to be transmitted. This is in contrast to synchronous transmission, where bits must be transmitted continuously to deep the clock frequency in both units synchronized with each other.

- The convention is that the transmitter rests at the 1-state when no characters are transmitted. The first bit, called the start bit, is always a 0 and is used to indicate the beginning of a character. The last bit called the stop bit is always a 1. An example of this format is shown in Fig.



A transmitted character can be detected by the receiver from knowledge of the transmission rules:

1. When a character is not being sent, the line is kept in the 1-state.
2. The initiation of a character transmission is detected from the start bit, which is always 0.
3. The character bits always follow the start bit.
4. After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1-state for at least one bit time.

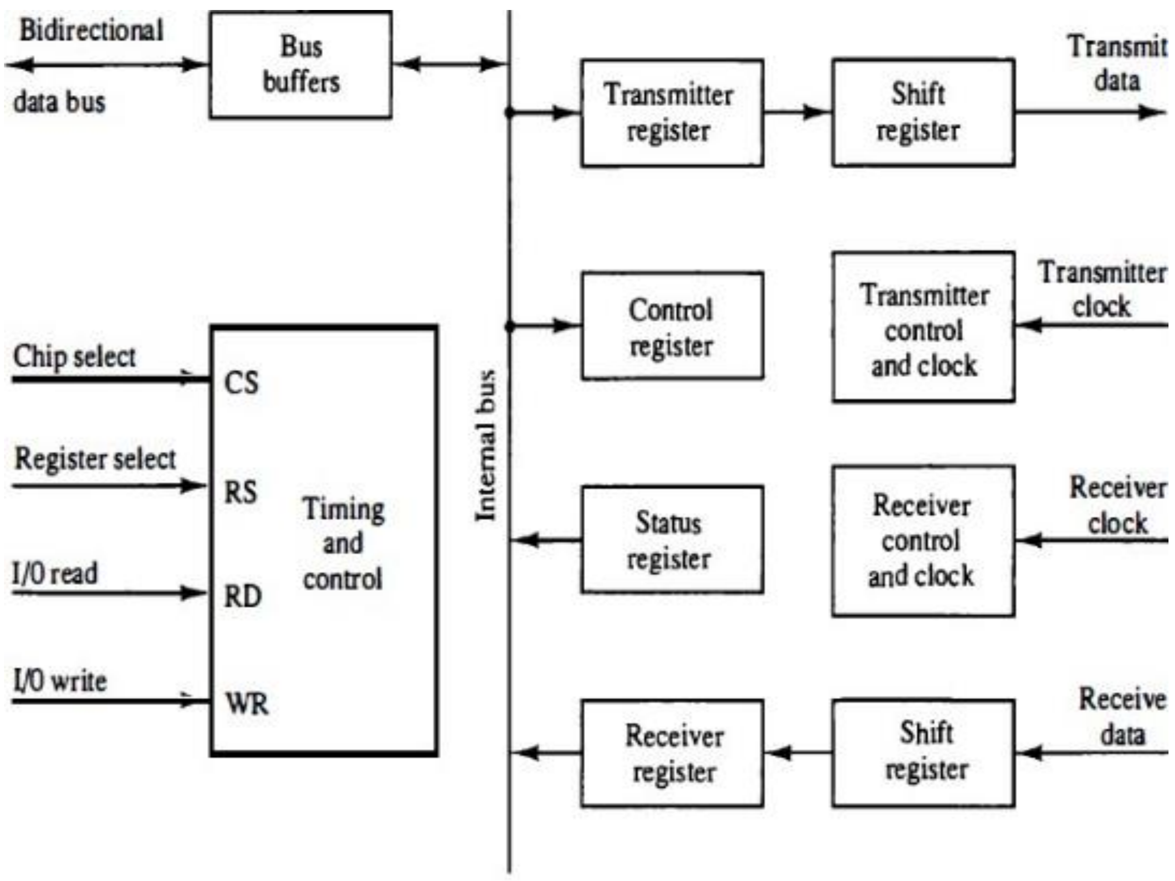
- At the end of the character the line is held at the 1-state for a period of at least one or two bit times so that both the transmitter and receiver can resynchronize. The length of time that the line stays in this state depends on the amount of time required for the equipment to resynchronize.

- Some older electromechanical terminals use two stop bits, but newer terminals use one stop bit. The line remains in the 1-state until another character is transmitted. The stop time ensures that a new character will not follow for one or two bit times.

- The baud rate is defined as the rate at which serial information is transmitted and is equivalent to the data transfer in bits per second. Ten characters per second with an 11-bit format has a transfer rate of 110 baud.

### **4.8.4. Asynchronous Communication Interface**

The block diagram of an asynchronous communication interface is shown in below figure. It functions as both a transmitter and a receiver. The interface is initialized for a particular mode of transfer by means of a control byte that is loaded into its control register.



| CS | RS | Operation | Register selected |
|---|---|---|---|
| 0 | × | × | None: data bus in high-impedance |
| 1 | 0 | WR | Transmitter register |
| 1 | 1 | WR | Control register |
| 1 | 0 | RD | Receiver register |
| 1 | 1 | RD | Status register |

## 4.9. Modes of Transfer

Data transfer to and from peripherals may be handled in one of three possible modes:

1. Programmed I/O
2. Interrupt-initiated I/O
3. Direct memory access (DMA)

### Programmed I/O

- **This** operations are the result of I/O instructions written in the computer program.

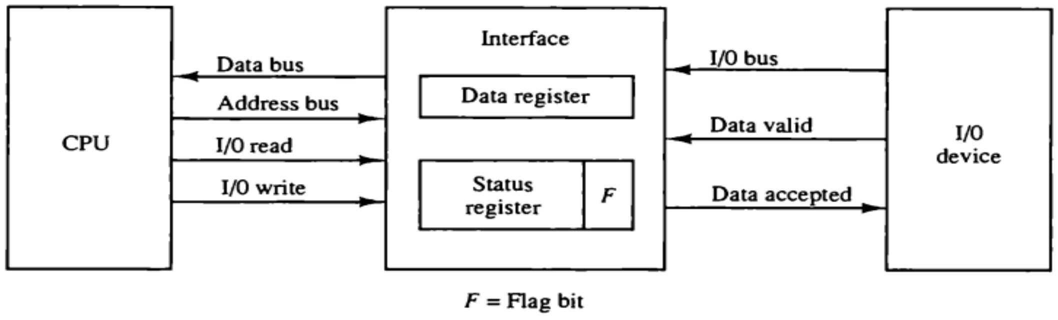
- When the interface determines that the device is ready for data transfer, it generates an interrupt request to the computer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to a service program to process the I/O transfer, and then returns to the task it was originally performing.

- Transfer of data under programmed I/O is between CPU and peripheral. In direct memory access (DMA), the interface transfers data into and out of the memory unit through the memory bus. The CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks.

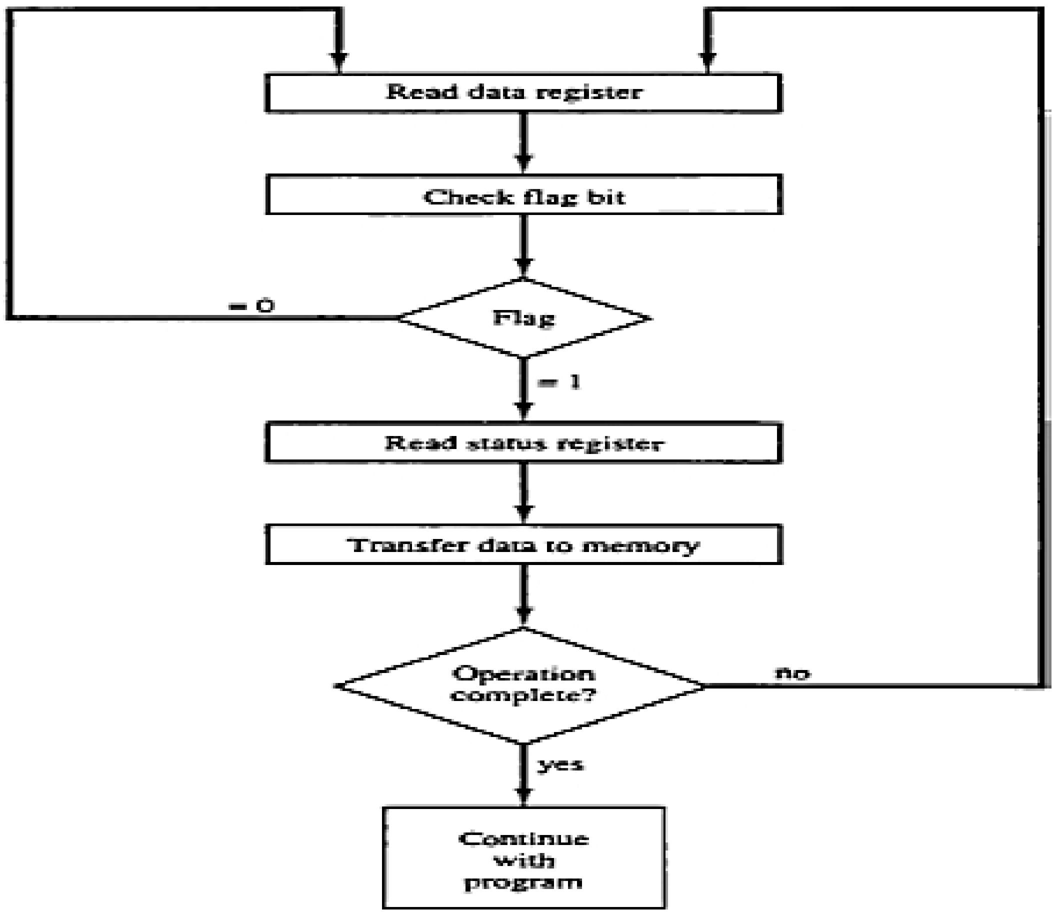
- Many computers combine the interface logic with the requirements for direct memory access into one unit and call it an I/O processor (IOP). The IOP can handle many peripherals through a DMA and interrupt facility. In such a system, the computer is divided into three separate modules: the memory unit, the CPU, and the IOP.

### 4.9.1. Example of Programmed I/O

In the programmed I/O method, the I/O device dies not have direct access to memory. A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU, and a store instruction to transfer the data from the CPU to memory.An example of data transfer from an I/O device through an interface into the CPU is shown in Fig.



The device transfers bytes of data one at a time as they are available. When a byte of data is available, the device places it in the I/O bus and enables its data valid line. The interface accepts the byte into its data register and enables the data accepted line. The interface sets a it in the status register that we will refer to as an F or “flag” bit. The device can now disable the data valid line, but it will not transfer another byte until the data accepted line is disabled by the interface. A flowchart of the program that must be written for the CPU is shown in Fig.



It is assumed that the device is sending a sequence of bytes that must be stored in memory. The transfer of each byte requires three instructions:

1. Read the status register.
2. Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
3. Read the data register.

Each byte is read into a CPU register and then transferred to memory with a store instruction. A common I/O programming task is to transfer a block of words form an I/O device and store them in a memory buffer. A program that stores input characters in a memory buffer.

### 4.9.2. Interrupt-Initiated I/O

The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer. The way that the processor chooses the branch address of the service routine varies from tone unit to another. In principle, there are two methods for accomplishing this. One is called vectored interrupt and the other, no vectored interrupt. In a non vectored interrupt, the branch address is assigned to a fixed location in memory. In a vectored interrupt, the source that interrupts supplies the branch information to the computer. This information is called the **interrupt vector.**

### 4.9.3. Software Considerations

A computer must also have software routines for controlling peripherals and for transfer of data between the processor and peripherals. I/O routines must issue control commands to activate the peripheral and to check the device status to determine when it is ready for data transfer. Error hacking and other useful steps often accompany the transfers. In interrupt-controlled transfers, the I/O software must issue commands to the peripheral to interrupt when ready and to service the interrupt when it occurs. In DMA transfer, the I/O software must initiate the DMA channel to start its operation.

## 4.10. Priority Interrupt

- Data transfer between the CPU and an I/O device is initiated by the CPU. However, the CPU cannot start the transfer unless the device is ready to communicate with the CPU.

- A priority interrupts is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or more request arrive simultaneously. The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced.

- Higher-priority interrupt levels are assigned to request which, if delayed of interrupted, could have serious consequences. Devices with high- speed transfers such as keyboards receive low priority. When two devices interrupt the computer at the same time, the computer services the devices interrupt the computer at the same time, the computer services the device, with the higher priority first.

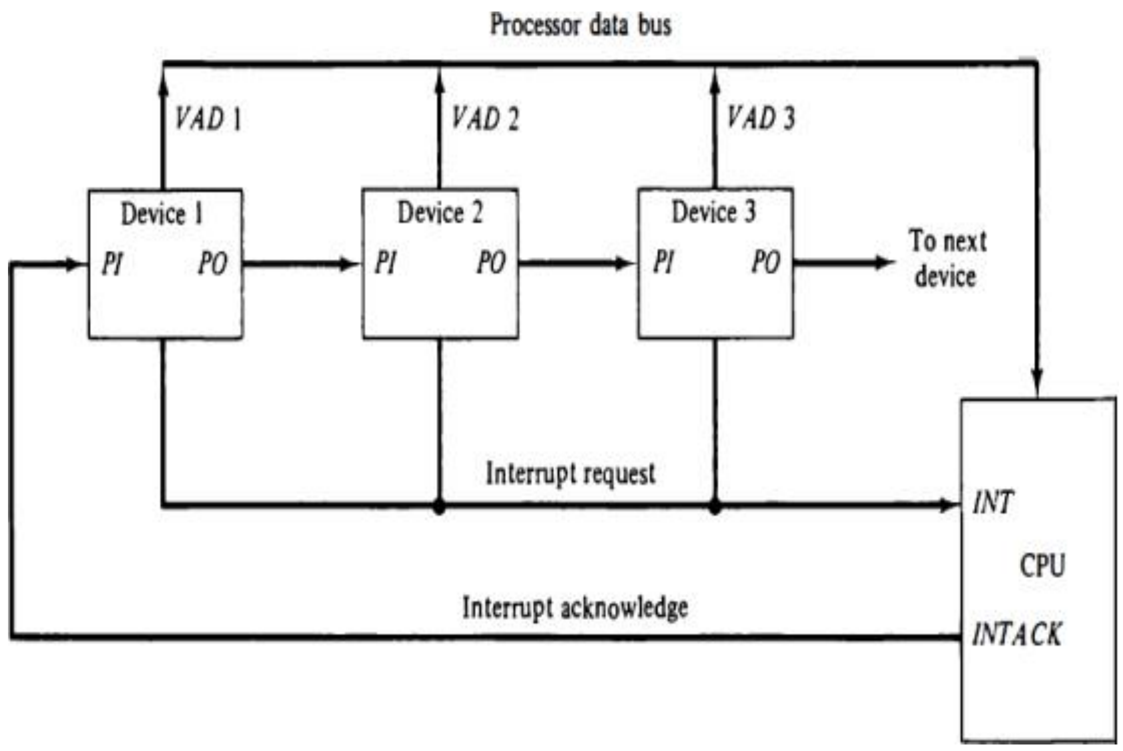
- The disadvantage of the software method is that if there are many interrupts, the time required to poll them can exceed the time available to service the I/O device. In this situation a hardware priority-interrupt unit can be used to speed up the operation.

- A hardware priority-interrupt unit functions as an overall manager in an interrupt system environment. It accepts interrupt requests from many sources, determines which of the incoming requests has the highest priority, and issues an interrupt request to the computer based on this determination.

- To speed up the operation, each interrupt source has its own interrupt vector to access its own service routine directly. Thus no polling is required because all the decisions are established by the hardware priority-interrupt unit. The hardware priority function can be established by either a serial or a parallel connection of interrupt lines. The serial connection is also known as the **daisy chaining method.**

**4.10.1.Daisy-Chaining Priority**

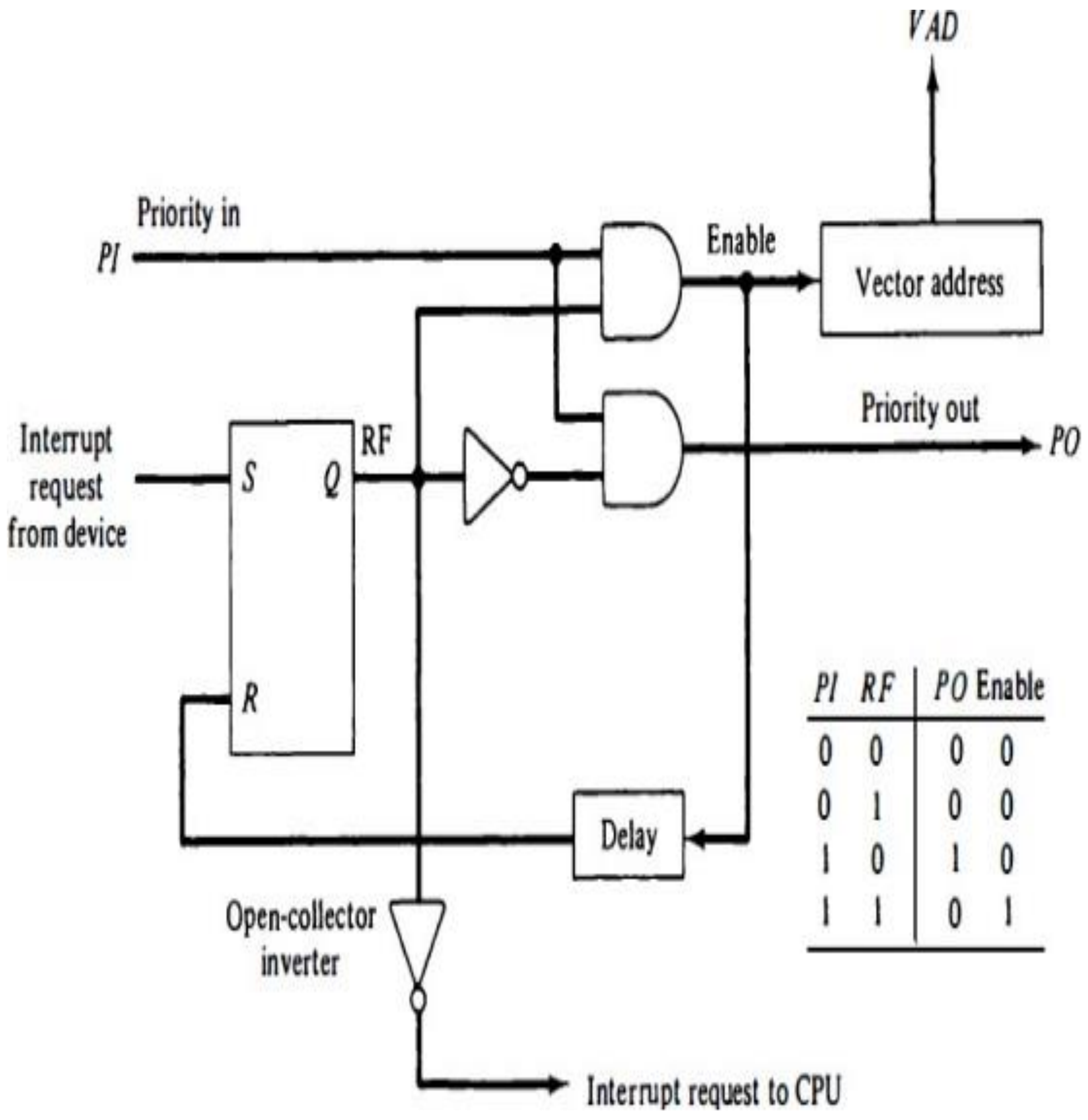
- The daisy-chaining method of establishing priority consists of a serial connection of all devices that request an interrupt.
- The device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the chain. This method of connection between three devices and the CPU is shown in Fig.



- The interrupt request line is common to all devices and forms a wired logic connection. If any device has its interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU.
- When no interrupts are pending, the interrupt line stays in the high-level state and no interrupts are recognized by the CPU.
- This signal is received by device 1 at its PI (priority in) input. The acknowledge signal passes on to the next device through the PO (priority out) output only if device 1 is not requesting an interrupt.
- If device 1 has a pending interrupt, it blocks the acknowledge signal from the next device by placing a 0 in the PO output. It then proceeds to insert its own interrupt vector address (VAD) into the data bus for the CPU to use during the interrupt cycle.
- A device with a 0 in its PI input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked.
- A device that is requesting an interrupt and has a 1 in its PI input will intercept the acknowledge signal by placing a 0 in its PO output. If the device does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output. Thus the device with PI = 1 and PO = 0 is the one with the highest priority that is requesting an interrupt, and this device places

its VAD on the data bus.

- The daisy chain arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU. The farther the device is from the first position; the lower is its priority. Below figure shows the internal logic that must be included with in each device when connected in the daisy-chaining scheme.



| PI | RF | PO | Enable |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

- The device sets its RF flip-flop when it wants to interrupt the CPU. The output of the RF flip-flop goes through an open-collector inverter, a circuit that provides the wired logic for the common interrupt line.

- If PI = 0, both PO and the enable line to VAD are equal to 0, irrespective of the value of RF. If PI = 1 and RF = 0, then PO = 1 and the vector address is disabled. This condition passes the acknowledge signal to the next device through PO. The device is active when PI = 1 and RF = 1.

- This condition places a 0 in PO and enables the vector address for the data bus. It is assumed that each device has its own distinct vector address. The RF flip-flop is reset after a sufficient delay to ensure that the CPU has received the vector address.

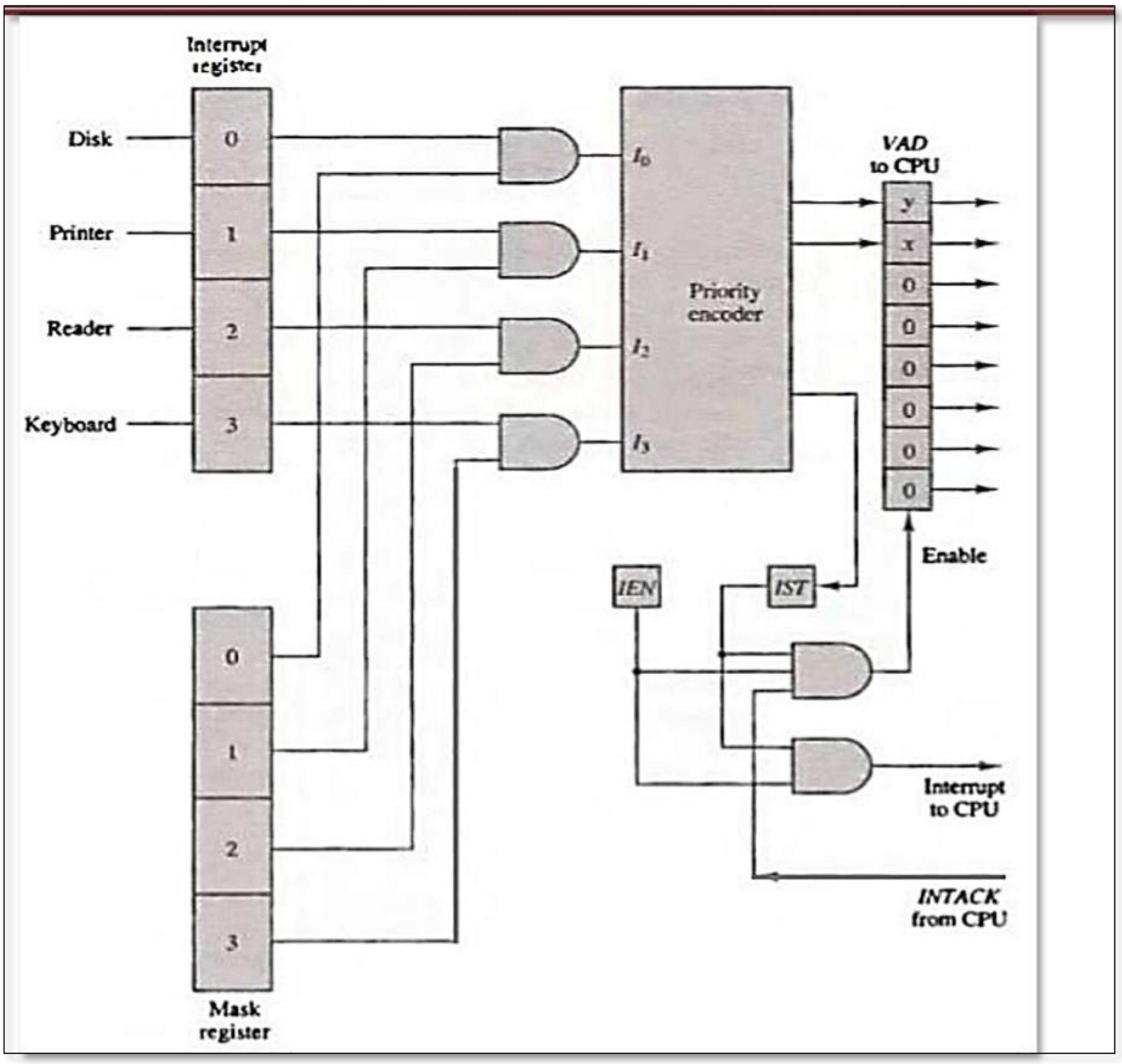
### 4.10.2. Parallel Priority Interrupt

- The parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device. Priority is established according to the position of the bits in the register. In addition to the interrupt register the circuit may include a mask register whose purpose is to control the status of each interrupt request.

- The mask register can be programmed to disable lower-priority interrupts while a higher-priority device is being serviced. It can also provide a facility that allows a high-priority device to interrupt the CPU while a lower-priority device is being serviced.

- The priority logic for a system of four interrupt sources is shown in Fig.It consists of an interrupt register whose individual bits are set by external conditions and cleared by program instructions. The magnetic disk, being a high-speed device, is given the highest priority.

- The printer has the next priority, followed by a character reader and a keyboard. The mask register has the same number of bits as the interrupt register. By means of program instructions, it is possible to set or reset any bit in the mask register.



## 4.11. Direct Memory Access (DMA)

- The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer.

- This transfer technique is called **direct memory access (DMA).** During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

- The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessors is to disable the buses through special control signals. Following figure shows two control signals in the CPU that facilitate the DMA transfer.

- The **bus request** (BR) input is used by the DMA controller to request the CPU to relinquish control of the buses. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, the data bus, and the read and write lines into a high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have a logic significance.

- The CPU activates the **Bus grant (BG)** output to inform the external DMA that the buses are in the high -impedance state.

- When the DMA terminates the transfer, it disables the bus request line. The CPU disables the bus grant, takes control of the buses, and returns to its normal operation.

- This mode of transfer is needed for fast devices such as magnetic disks, where data transmission cannot be stopped or slowed down until an entire block is transferred.

- An alternative technique **called cycle stealing** allows the DMA controller to transfer one data word at a time after which it must return control of the buses to the CPU.

### 4.11.1. DMA Controller

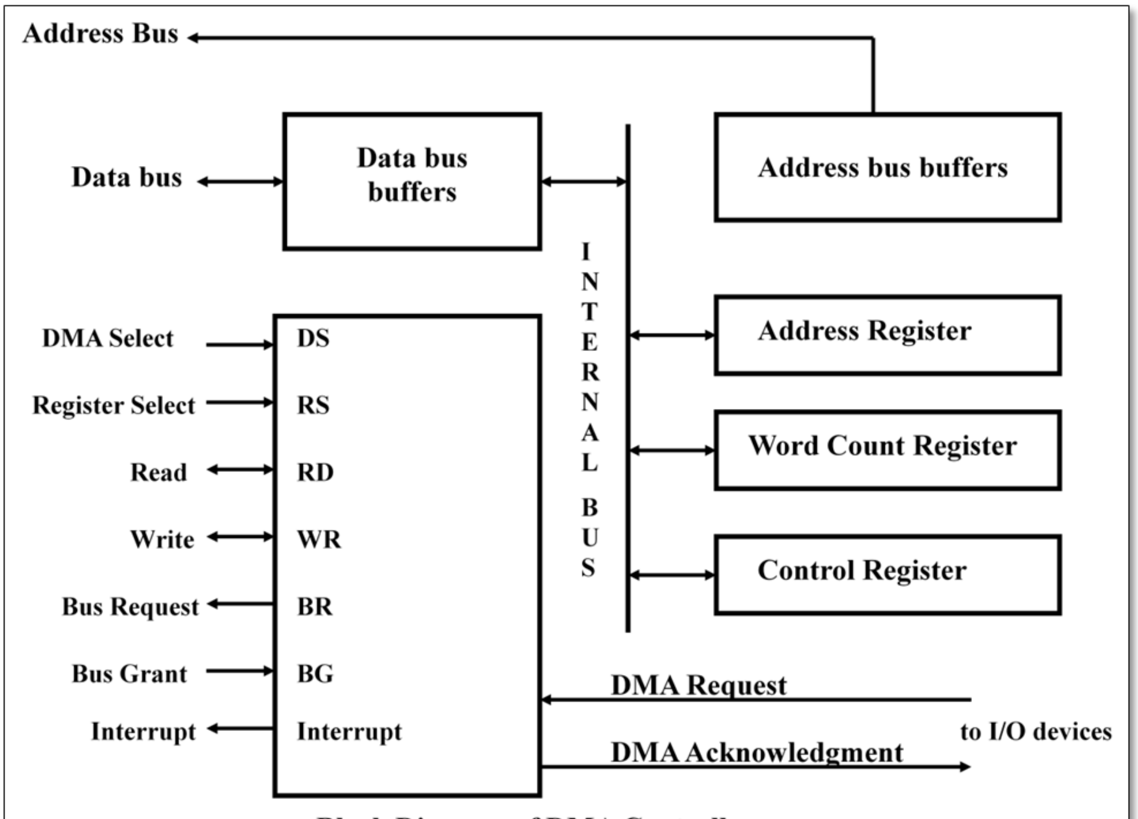
- The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. In addition, it needs an address register, a word count register, and a set of address lines

- The word count register specifies the number of words that must be transferred. The data transfer may be done directly between the device and memory under control of the DMA.

- The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (register select) inputs. The RD (read) and WR (write) inputs are bidirectional.

- When the BG (bus grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG = 1, the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control. ; the DMA communicates with the external peripheral through the request and acknowledge lines by using a prescribed handshaking procedure.

Following figure shows the block diagram of a typical DMA controller.



**Block Diagram of DMA Controller**

- The DMA controller has three registers: an address register, a word count register, and a control register. The **address register** contains an address to specify the desired location in memory. The address bits go through bus buffers into the address bus. The address register is incremented after each word that is transferred to memory.

- The **word count register** is incremented after each word that is transferred to memory. The word count register holds the number of words to be transferred. This register is decremented by one after each word transfer and internally tested for zero.

- The **control register** specifies the mode of transfer. All registers in the DMA appear to the CPU as I/O interface registers. Thus the CPU can read from or write into the DMA registers under program control via the data bus.

The CPU initializes the DMA by sending the following information through the data bus:

1. The starting address of the memory block where data are available (for read) or where data are to be stored (for write)
2. The word count, which is the number of words in the memory block
3. Control to specify the mode of transfer such as read or write
4. A control to start the DMA transfer

## 4.11.2. DMA Transfer

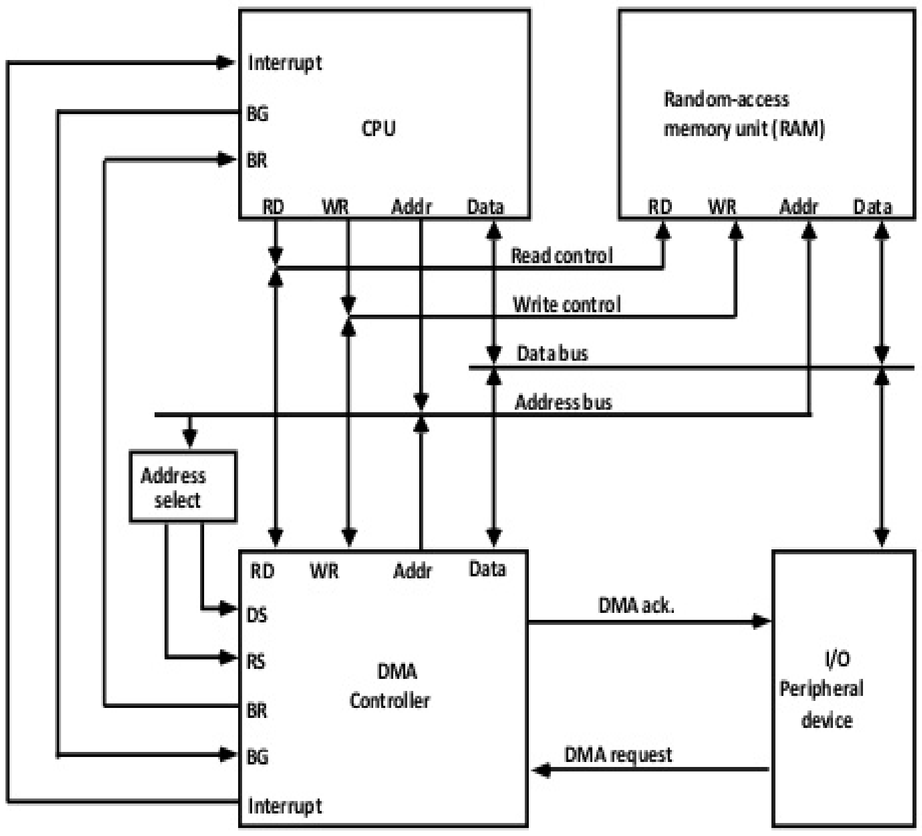
- The position of the DMA controller among the other components in a computer system is illustrated in Fig. The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines.

- The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory.

- When the peripheral device sends a DMA request, the DMA controller activates the BR line, informing the CPU to relinquish the buses. The CPU responds with its BG line, informing the DMA that its buses are disabled.

- The DMA then puts the current value of its address register into the address bus, initiates the RD or WR signal, and sends a DMA acknowledge to the peripheral device. Note that the RD and WR lines in the DMA controller are bidirectional. The direction of transfer depends on the status of the BG line.

- When BG line. When BG = 0, the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG = 1, the RD and WR and output lines from the DMA controller to the random-access memory to specify the read or write operation for the data.
- When the peripheral device receives a DMA acknowledge, it puts a word in the data us (for write) or receives a word from the data bus (for read). Thus the DMA controls the read or write operations and supplies the address for the memory. The peripheral unit can then communicate with memory through the data bus for direct transfer between the two units while the CPU is momentarily disabled.



**Input-Output Processor:**

- It is a processor with direct memory access capability that communicates with IO devices.
- IOP is similar to CPU except that it is designed to handle the details of IO operation.
- Unlike DMA which is initialized by CPU, IOP can fetch and execute its own instructions.
- IOP instruction are specially designed to handle IO operation.

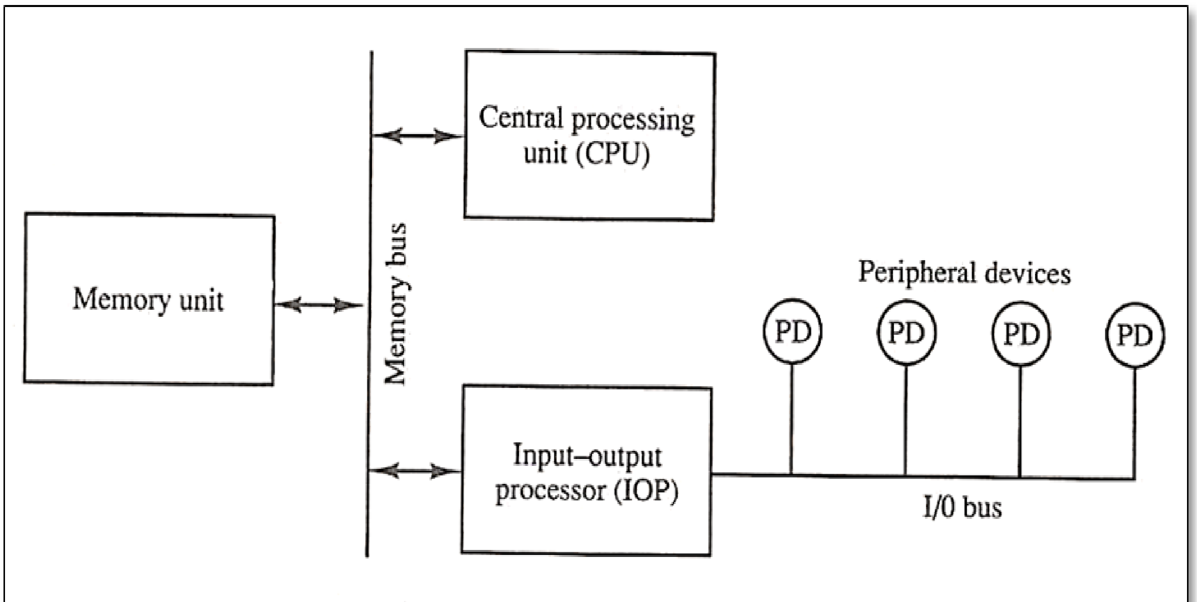


Figure Block diagram of a computer with I/O processor.

→ Memory occupies the central position and can communicate with each processor by DMA.

→ CPU is responsible for processing data.

→ IOP provides the path for transfer of data between various peripheral devices and memory.

→ Data formats of peripherals differ from CPU and memory. IOP maintain such problems.

→ Data are transfer from IOP to memory by stealing one memory cycle.

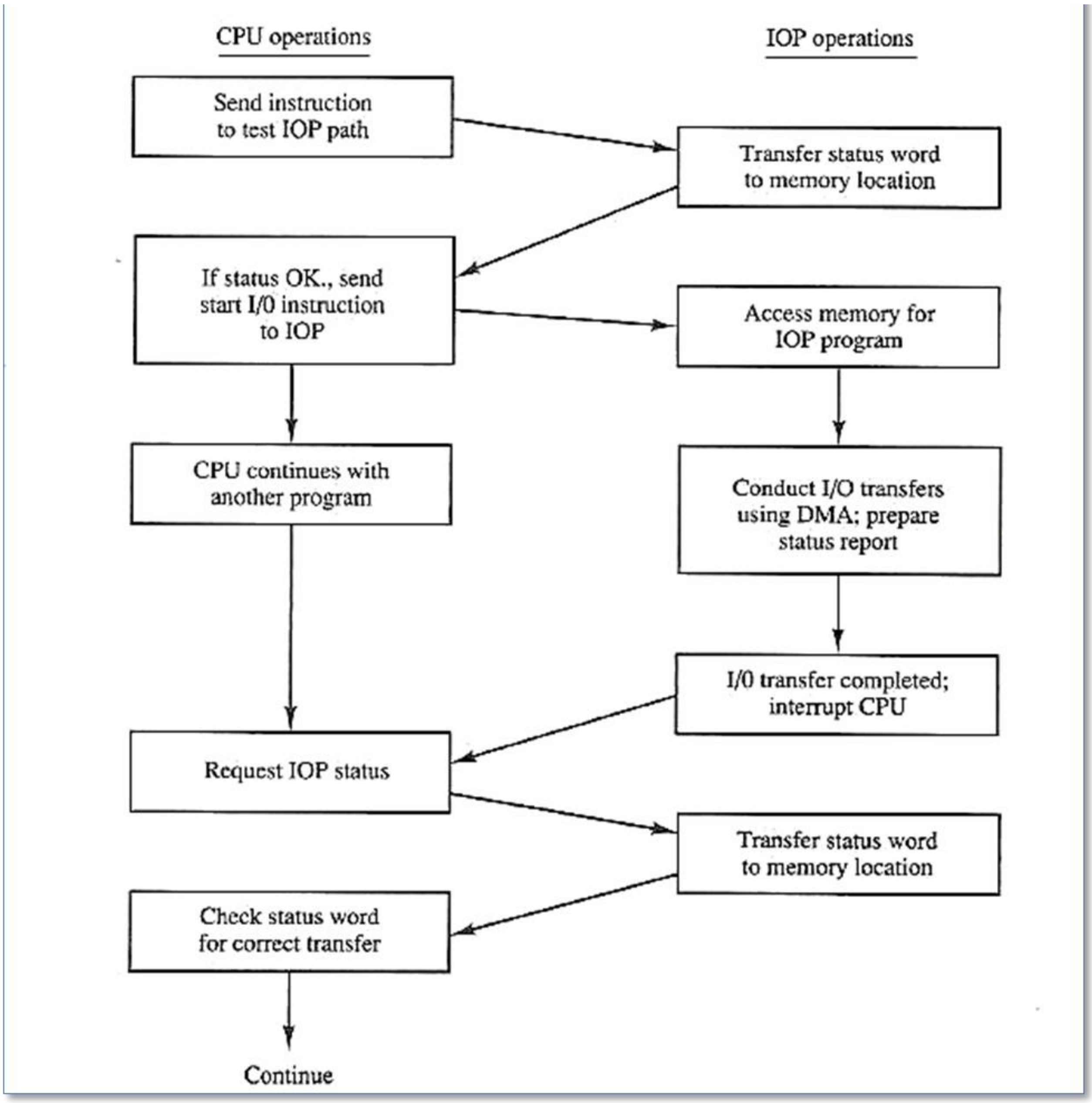
→ Instructions that are read from memory by IOP are called commands to distinguish them from instructions that are read by the CPU.

**Instruction *that are read from memory by an IOP***

→ Distinguish from instructions that are read by the CPU

→ Commands are prepared by experienced programmers and are stored in memory

→ Command word = IOP program


CPU operations
IOP operations
Send instruction to test IOP path
Transfer status word to memory location
If status OK., send start I/0 instruction to IOP
Access memory for IOP program
CPU continues with another program
Conduct I/O transfers using DMA; prepare status report
I/0 transfer completed; interrupt CPU
Request IOP status
Transfer status word to memory location
Check status word for correct transfer
Continue