UNIT- I 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.

Introduction:

What is a Computer/Digital Computer?

It is a fast electronic computing device that

- accepts the input
- processes the accepted input(according to the internally stored instructions) and
- produces output

Type of Computers:

Computers are classified into many categories based on size, cost and performance.

1. Embedded computers:

These are integrated into a larger device in order to automatically monitor and control a physical process or environment. Ex: AC, Washing Machine, Microwave Oven etc.

2. Personal computers:

These are the most common type computer in homes, schools, offices etc,. It is the most common type of desk top computers with processing and storage units along with various input and output devices.

3. Portable/Notebook computers:

This type of computers provides the basic features of a personal computer in a smaller lightweight package. They can operate on batteries to provide mobility.

4. Workstation computers:

These are used in engineering applications of interactive design work, where the applications require a high computational capacity.

5. Enterprise computers:

These are used for business data processing in medium to large organizations that require much more computing power and storage capacity than work stations and also these computers are meant for sharing the data among more numbers of users via some form communication network.

6. Mainframe computers:

These are Large and expensive computers capable of simultaneously processing data for hundreds or thousands of users. Used to store, manage, and process large amounts of data that need to be reliable, secure, and centralized.

7. Super computers:

A computer considered to be the fastest in the world. It is used to execute tasks that would take lot of time for other computers. For Ex: Modeling weather systems.

Why do we need to study CO?

- Computers are became the most important parts of our daily life. Ex: Laptops, mobile phones, smart phones and some electronic gadgets etc.
- So it is **needed to understand the computer**.

1. How a computer works?
2. What parts are there in the computer?
3. How the system components are interconnected and how they work?

Computer Architecture Vs Computer Organization

- **Computer Architecture:**

The architecture (also called as Instruction Set Architecture (ISA)) of a computer is concerned with the structure and behavior of the computer by a user such as an assembly or machine language programmer.

- This is essential to understand (processor ISA, registers, instruction formats etc) computer by the machine or assembly language programmer who is likely to program a computer.

- **Computer Organization:**

Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system.

- **Benefits to the User**

I. Knowledge of computer organization can help one to understand the internal operations of a computer.

II. Computer organization deals with how the different components of a computer (processor, memory and peripherals) are interconnected and their roles playing during the program execution.

Basic Organization of a Computer:

The important parts a computer are the processor, main memory, hard disk(secondary memory), keyboard, monitor and peripheral devices. All these different parts of a computer are connected through a single bus called a Backplane Bus.

A single bus interconnecting all the components of a computer is usually called a "**Backplane Bus**". Because it is considered to be a backbone communication medium, to which various components of a computer are attached. Backplane bus was replaced by multiple buses in the modern computers for achieving higher performance.

- The basic organization of a computer is shown in the below figure (a).

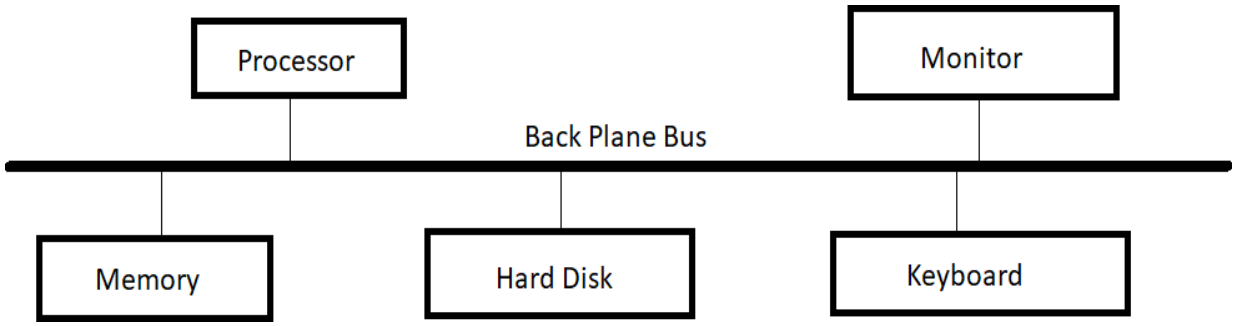


Figure (a): Basic organization of a digital computer

We briefly discuss the important parts of a Von Neumann computer (or) a digital computer.

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

Digital Computer operations can be described as follows:

- A set of instructions or a program will be stored in the memory.
- The CPU fetches those instructions sequentially from the memory, decodes them and performs the specified operation on associated data or operands in ALU.

- Processed data and results will be sent to an output unit, if required.
- All the activities like processing of any operation and data movement inside the computer are governed by control unit.

Main memory: In a computer, any memory (Main memory, Secondary memory) is used to store data and instructions. The main memory or primary memory is also called as *Random Access Memory* (RAM) because the CPU can access any location in main memory at random and either retrieve the data stored at that location, or can store some data at the location. The main memory is usually made from SRAM (Static RAM) or DRAM (Dynamic RAM).

Secondary memory or auxiliary memory is an additional memory to the main memory. Secondary memory is available in many forms such as CD, DVD, USB devices (like Pen drive, External Hard disk) .

Monitor: The monitor is an output device which is used to display the result/output from the computer. It is the primary output unit of the computer. Over the decades monitors have evolved from the pure text display devices to monitors capable of displaying high-quality graphics and animations. There are some other output devices such as printers and plotters (to get the output from computer in the form of printout), speakers (to get the output from in the form of sound) etc.

Keyboard: In the early computers, the card reader was the primary input device. However, a keyboard is the primary input device in the modern computers. Some other popular input devices are mouse, scanner, camera etc.

Peripheral Devices: Several peripheral devices can be attached to the backplane bus. These peripheral devices can be output devices (printers, plotters, loud speakers etc) or can be input devices (scanners, cameras etc.). Processor or CPU can provide the output to these output devices or can collect input from the input devices over backplane bus.

Backplane Bus: The backplane bus is a group of wires. These wires are partitioned into control, address and data wires. The control wires carry control signals to different units in the computer. The address wires carry the address of the specific data in memory and data wires are used to carry the data.

Historical Perspective:

- The computers that we use today have evolved over the last seven or eight decades.
- Evolution of computers is a continuous process, for the simplicity of our understanding we just discuss the major developments that took place in the last seven or eight decades.
- we can divide these developments into five generations based on the significant improvements in the technology or improvements to the other aspects that were incorporated into the computers.

1. First Generation Computers (1941-1956)
2. Second Generation Computers (1956-1963)
3. Third Generation Computers (1964-1971)
4. Fourth Generation Computers (1971- Present)
5. Fifth Generation Computers (Present and Beyond)

1. **First Generation Computers (1941-1956):**
   - → First generation computers were based on vacuum tubes which were glass (tubes) for performing computations and magnetic drums for storage of data.
   - → A lot of power consumption was done by the vacuum tubes.
   - → Low reliability: they were breaking down frequently.
   - → A lot of power consumption was done by vacuum tubes.
   - → An operator was needed to execute programs instruction by instruction.

Figure (b): vacuum tubes

Example: (First Generation Computers (1941-1956)

- → The ENIAC (Electrical Numerical Integrator and Calculator) machine was developed by John W.Mauchly and J. Presper Eckert at the University of Pennsylvania
- → It was developed for military needs.
- → 18,000 vacuum tubes were used approximately.
- → Punch-cards were used for input.
- → Weighed 30 tons and occupied a 30X50 foot space.

Figure (c): ENIAC Machine

2. **Second Generation Computers (1956-1963):**
   - → Vacuum tubes were replaced by Transistors. Transistor is a small device that transfers electronic signals through resistors.
   - → Transistors were made from semiconductor materials.
   - → Assembly language was used to programming.
   - → Peripherals such as printers, magnetic tape drives, and magnetic disk were started to appear in this generation.

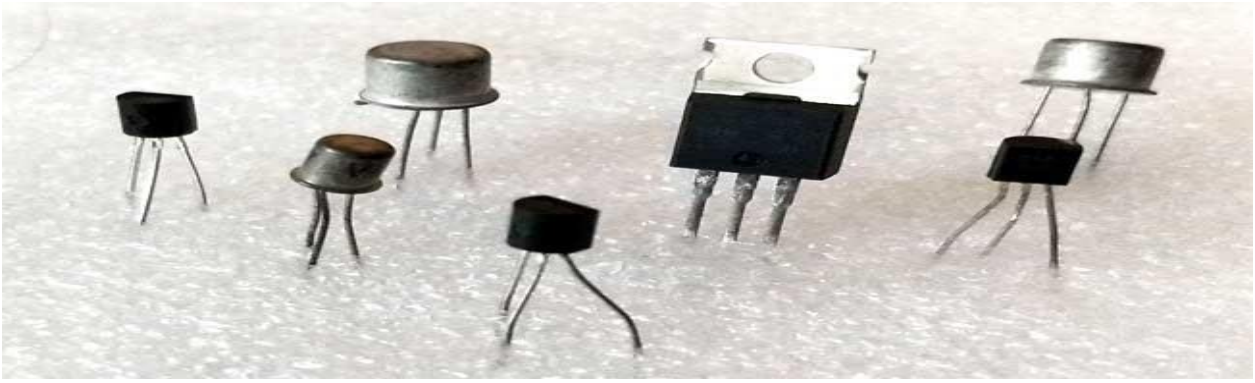


Figure (d): Transistors

Example: (Second Generation Computers (1956-1963))

→ The IBM 1400 Series were a major breakthrough by IBM.
→ The first computer in this series was IBM 1401.
→ This system contained many peripherals, which included a new high-speed printer.
→ This printer could print 600 lines per minute.

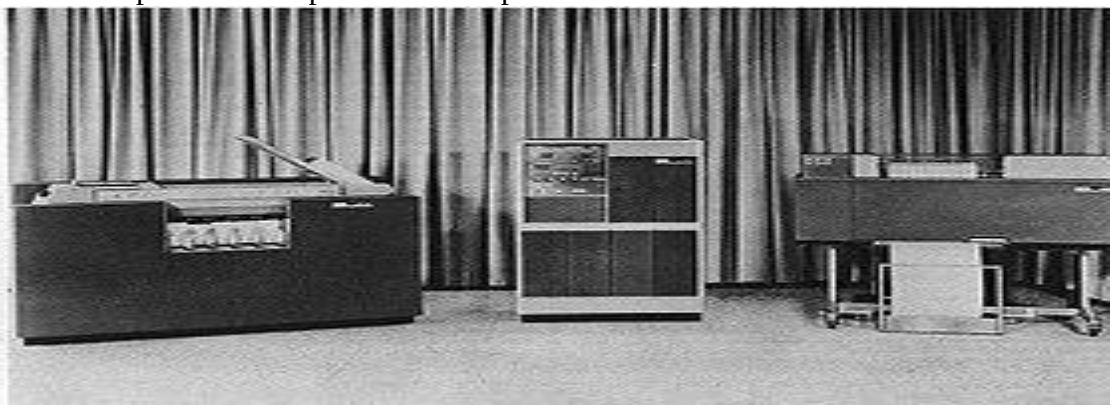


Figure (e): IBM 1401 system

3. Third Generation Computers (1964-1971):

→ Third generation computers were developed using ICs (Integrated Circuits).
→ Computers were compact in size, more energy efficient and reliable.
→ Peripherals such as monitors and keyboards were used.
→ Some operating systems (OS) were developed and used.
→ High level programming languages like COBOL, FORTRAN were used to write large programs.

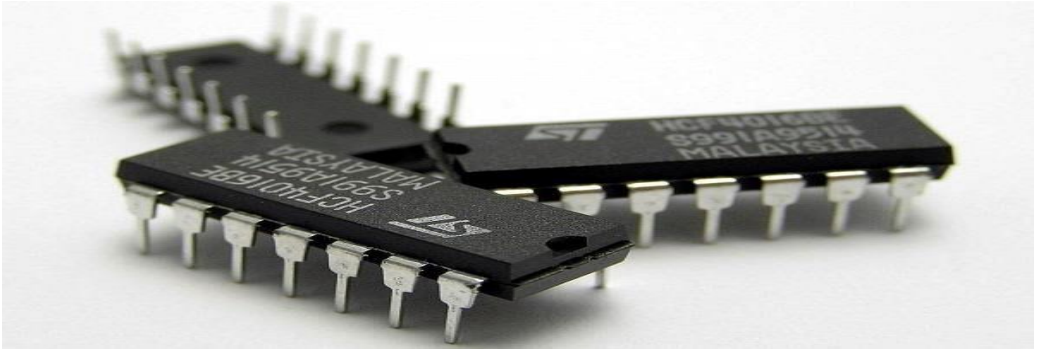


Figure (f): Integrated Circuits (ICs)

4. **Fourth Generation Computers (1971-Present):**
   → Fourth generation computers were developed using VLSI(Very Large Scale Integration), ULSI (VLSI(Ultra Large Scale Integration).
   → VLSI & ULSI made it possible to pack millions of components on small chip.
   → They reduced the size and price of computers and also at the same time increased the power, efficiency and reliability.
   → This generation starts with minicomputers, then desktop computers, then laptops and recently hand held computers.
   → Many high level programming languages and operating systems are available.
5. **Fifth Generation Computers (Present and Beyond):**
   → Fifth generation computers are not yet really but they are shaping in the research laboratories.
   → They are expected to heavily incorporate Artificial Intelligence (AI) techniques.
   → They would able to take audio and visual user commands and give response as expected.
   → They would use massive parallel processing and would have enormous computing powers.

Bus Structures:

A Bus is a group of wires or lines, connecting all major internal components to the CPU and memory to transfer the data from one part of a computer to another and also connecting other peripherals of a computer.

A system bus typically consists of about 50 to 100s of separate lines. Each line is assigned a particular meaning or function.

These lines are classified into **three** groups,

1. **Data lines:** provide a path for moving data among system modules. These lines collectively are called the **data bus**.
2. **Address lines:** These are used to identify the source or destination of the data on the data bus.
3. **Control lines:** These are used to control the access to and the use of the data and address lines. Because the data and address lines are shared by all components, there must be a means of **controlling their use**.

In addition, there may be power distribution lines that supply power to the attached components.
There are two types of Bus structures, they are:

1. Single Bus structure 2. Multiple Bus structure

1. Single Bus structure

- In single bus structure, all functional units are connected to a common bus called "system bus".
- Since single bus can be used for only one transfer at a time, only two units can use the bus and communicate with each other at a time.
- The bus control lines are used to manage multiple requests for use of the bus.

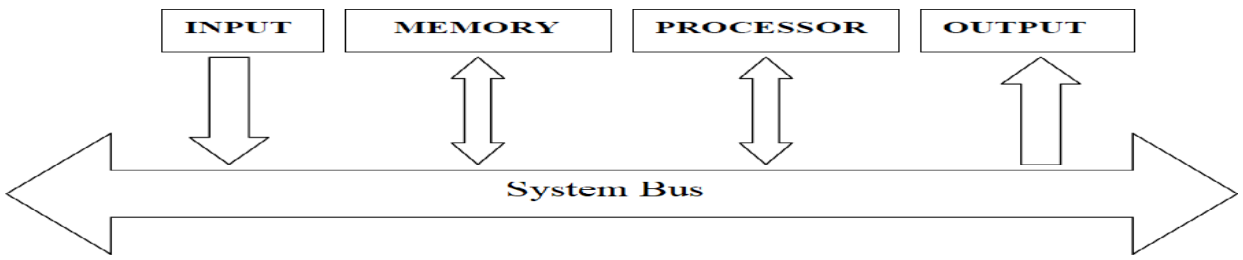


Figure (g): Single Bus structure

Advantage:

- low cost and flexibility for attaching peripheral devices.

Disadvantages: when large number of devices are connected to the common bus, computer system performance will be degraded. The reasons are:

1. When more devices are connected to the common bus, we need to share the bus among these devices. The sharing mechanism co-ordinates the use of bus. This co-ordination requires finite amount of time called "**propagation delay**". When control of bus passes from one device to another frequently, these propagation delays affect the performance of computer system.
2. When **aggregate data transfer demand approaches the capacity of the bus**, the bus may become a bottleneck. In such situations, we have to increase the data rate of the bus (high speed shared bus) or we have to use wider bus.

There are two approaches to get out these problems, they are

i. **Buffer Registers**
ii. **Multiple Bus**

Buffer Registers:

Devices connected to a bus have different speed of operation. Among the functional units, input and output devices are slower devices and memory and processor units are faster devices. Since all these functional units are connected over a bus, if the speed of bus is bounded by the slowest device connected to it, the efficiency will be very low.

A common approach to solve this is using "buffer registers". Buffer registers are used to hold the information during transfers. They prevent high speed processor from being locked to slower I/O devices during data transfers. This allows processor to switch rapidly from one device to another processing the data transfers involving several I/O devices.

Example: consider the transfer of data from processor to printer.

- The processor sends data to printer buffer over the bus.
- Once the buffer is loaded, printer starts printing without any intervention by the processor.
- Now, the bus and processor are no longer needed and can be released for other activity.
- Printer continues printing data in its buffer and is not available until this process is completed.

Thus, buffer registers smooth out the timing differences among processors, memories and I/O devices.

2. Multiple Bus structure

- In multiple bus structure, several devices are connected using more than one bus. These buses will have the hierarchical structure.

- There can be many transfers at a time, many units can use the bus and communicate with each other at a time.

- Thus achieving more concurrency in operations, which leads to better performance at an increased cost.

The structure of the multiple bus is shown in the below figure (h).

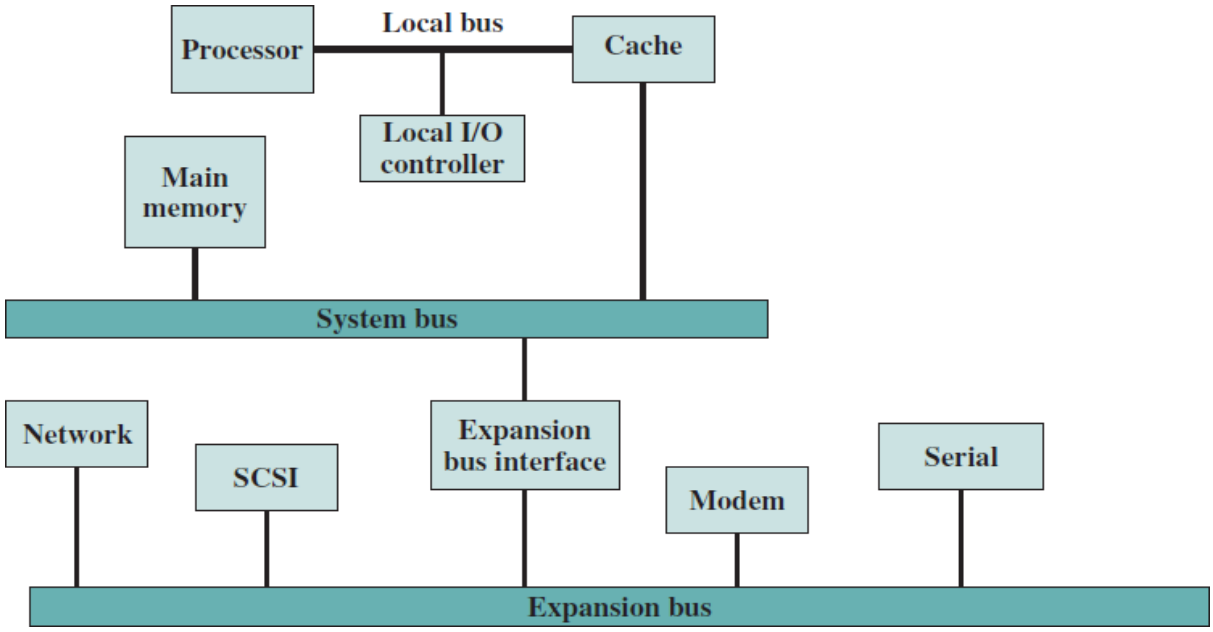


Figure (h): Multiple Bus structure

Data types:

Binary information in digital computers is stored in memory or processor registers. Registers contain either data or control information. Data are numbers and other binary coded information that are operated on to achieve required computational results.

The data types found in the registers of digital computers may be classified as being one of the following categories:

(1) Numbers used in arithmetic computations
(2) Letters of the alphabet used in data processing and
(3) Other discrete symbols used for specific purposes.

The **binary number system** is the most natural system to use in a digital computer. But sometimes it is convenient to employ different number systems (**Decimal Number System, Octal Number System, Hexadecimal Number System**), especially the decimal number system, since it is used by people to perform arithmetic computations.

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.

- ✓ Base or Radix for Binary Number system is 2 (It has only 2 symbols 1 and 0 only).
- ✓ Base or Radix for Decimal Number system is 10 (It has 10 symbols from 0 to 9).
- ✓ Base or Radix for Octal Number system is 8 (It has 8 symbols from 0 to 7).
- ✓ Base or Radix for Hexadecimal Number system is 16(It has 16 symbols from 0 to 9 and also from A to F).

All types of data, except binary numbers, are represented in computer registers in binary-coded form, why because registers are made up of flip-flops and they can store only l's and 0's.

Data Conversion:

Converting the data from one number system to another is common in computers. Let us see how to convert data from one number system to another.

1. <u>Conversion from Decimal to Binary & Vice-Versa:</u>

1. *Divide the decimal number by the **base of the binary number system** and keep track of the quotient and remainder.*
2. *Repeatedly divide the successive quotients while keeping track of the **remainders** generated until the quotient is zero.*
3. *The remainders generated during the process, written in **reverse order** of generation from left to right, form the equivalent number in the **binary** system.*

<u>Ex 1:</u> Convert the decimal number 167 into its equivalent binary number

		Quotient	Remainder
167/2	=	83	1
83/2	=	41	1
41/2	=	20	1
20/2	=	10	0
10/2	=	5	0
5/2	=	2	1
2/2	=	1	0
1/2	=	0	1

The equivalent binary number for the decimal number 167 is : 10100111

i.e., $(167)_{10} = (10100111)_2$

<u>Ex 2:</u> Convert the decimal number 266 into its equivalent binary number.

2	266	0
2	133	1
2	66	0
2	33	1
2	16	0
2	8	0
2	4	0
2	2	0
2	1	

The equivalent binary number for the decimal number 266 is : 100001010

i.e., $(266)_{10} = (100001010)_2$

2. Conversion from Decimal to Octal:

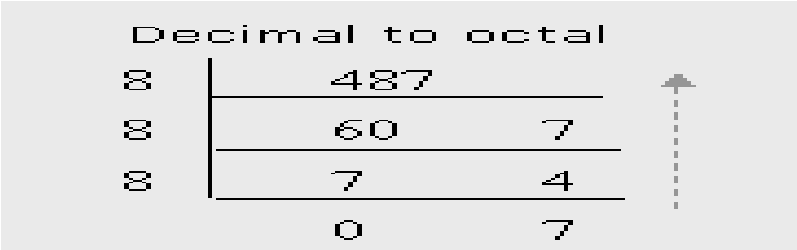
1. *Divide the decimal number by the base of the octal number system and keep track of the quotient and remainder.*
2. *Repeatedly divide the successive quotients while keeping track of the remainders generated until the quotient is zero.*
3. *The remainders generated during the process, written in reverse order of generation from left to right, form the equivalent number in the octal system.*

Ex 1: Convert the decimal number 167 into its equivalent octal number.

		Quotient	Remainder
167/8	=	20	7
20/8	=	2	4
2/8	=	0	2

The equivalent octal number is: 247 i,e., $(167)_{10} = (247)_8$

Ex 2: Convert the decimal number 487 into its equivalent octal number.



The equivalent octal number is: 747 i,e., $(487)_{10} = (747)_8$

3. Conversion from Decimal to Hexadecimal:

1. *Divide the decimal number by the **base of the Hexadecimal number system** and keep track of the quotient and remainder.*
2. *Repeatedly divide the successive quotients while keeping track of the **remainders** generated until the quotient is zero.*
3. *The remainders generated during the process, written in **reverse order** of generation from left to right, form the equivalent number in the **Hexadecimal** system.*

The below table shows decimal digits and its equivalent hexadecimal digits:

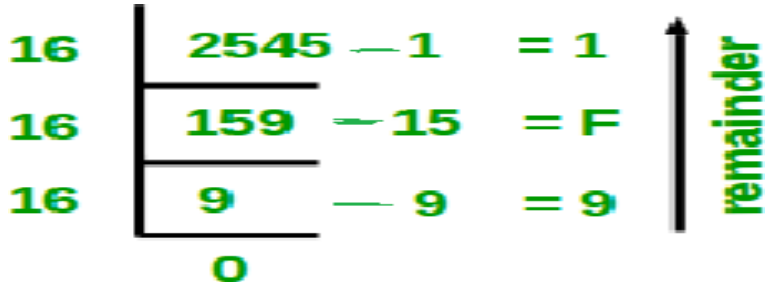
Decimal number	Hexadecimal number
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Ex 1: Convert the decimal number 167 into its equivalent hexadecimal number.

		Quotient	Remainder
167/16	=	10	7
10/16	=	0	A

The equivalent Hexadecimal is: A7 i,e., $(167)_{10} = (A7)_{16}$

Ex 2: Convert the decimal number 2545 into its equivalent hexadecimal number.



The equivalent Hexadecimal number is: 9F1 i,e., $(2545)_{10} = (9F1)_{16}$

4. Conversion from Binary to Decimal:

To convert a number expressed in the binary system to the decimal system, we perform the arithmetic calculations of Equation given below, that is, multiply each digit by its weight, and add the results.

$$d_0b^0 + d_1b^1 + \_ \_ \_ + d_{n-1}b^{n-1} + d_nb^n$$

where d_0 represents the least significant digit (LSD) and where d_n represents the most significant digit (MSD).

Here, b^0, b^1….. b^n are binary powers and $d_0, d_1 \ldots\ldots\ldots\ldots d_n$ are binary digits.

Ex 1: Convert the binary number 10100111 into its equivalent in the decimal system.

$(10100111)_2 = 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 + 1 \times 2^7$

$= 1+2+4+0+0+32+0+128$

$= (167)_{10}$

5. Conversion from Binary to Octal:

To convert a number in the binary system to the octal system,

1. *Form 3-bit groups starting from the right of given binary number.*
2. *Add extra 0s at the left-hand side of the binary number if the number of bits is not a multiple of 3.*
3. *Then replace each group of 3 bits by its equivalent octal digit.*

Why three bit groups? Simply because $2^3 = 8$
The below table shows the 3-bit binary and its equivalent octal digits,

3-bit binary	Octal digit
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Ex 1: Convert the binary number 10100111 into its equivalent in the octal system.

$(10100111)_2 =$ $\overbrace{010}^{2}\ \overbrace{100}^{4}\ \overbrace{111}^{7}$

$= (247)_8$

6. Conversion from Binary to Hexadecimal:

To convert a number in the binary system to the hexadecimal system,

1. *Form 4-bit groups starting from the right of given binary number.*
2. *Add extra 0s at the left-hand side of the binary number if the number of bits is not a multiple of 4.*
3. *Then replace each group of 4 bits by its equivalent hexadecimal digit.*

Why four bit groups? Simply because $2^4 = 16$
The below table shows the 4-bit binary and its equivalent hexadecimal digits,

4-bit binary	Hex digit	4-bit binary	Hex digit
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Ex 1: Convert the binary number 1101011111 into its equivalent hexadecimal number.

$(1101011111)_2 =$ $\overbrace{00}^{3}\overbrace{1101}^{5}\overbrace{0111}^{F}$

$= (35F)_{16}$

7. Conversion from Octal to Binary:

To convert a number in the octal system to the binary system,

- ***For each octal digit, write the 3-bit equivalent binary groups.***
- ***We should write exactly 3 bits for each octal digit even if there are leading '0's. For example, for octal digit 0, write the three bits 000.***

Ex 1: Convert $(247)_8$ into its equivalent Binary Number.

$(247)_8 =$ $\overbrace{010}^{2}\overbrace{100}^{4}\overbrace{111}^{7}$

$= (010100111)_2$

Ex 2: Convert $(105)_8$ into its equivalent Binary Number.

$(105)_8 =$ 1 0 5 / 001 000 101

$= (001000101)_2$

8. Conversion from Octal to Decimal:

To convert a number expressed in the octal system to the decimal system, we perform the arithmetic calculations of Equation given below, that is, multiply each digit by its weight, and add the results.

$$d_0b^0 + d_1b^1 + \_ \_ \_ + d_{n-1}b^{n-1} + d_nb^n$$

where d_0 represents the least significant digit (LSD) and where d_n represents the most significant digit (MSD).

Here, b^0, b^1….. b^n are octal powers, d_0, d_1 ..... d_n are octal digits.

Ex 1: Convert the octal number 247 into its equivalent decimal number.

$(247)_8 = 7.8^0 + 4.8^1 + 2.8^2$

$= 7+32+128$

$= 167$

$(247)_8 = (167)_{10}$

9. Conversion from Hexadecimal to Binary:

To convert a number in the hexadecimal system to the binary system,

- ***For each hexadecimal digit, write the 4-bit equivalent binary groups.***
- ***We should write exactly 4 bits for each hexadecimal digit even if there are leading '0's. For example, for decimal hexadecimal digit 0, write the 4 bits 0000.***

Ex 1: Convert $(247)_{16}$ into its equivalent Binary Number.

$(247)_{16} =$ 2 4 7

0010 0100 0111

$= (001001000111)_2$

10. Conversion from Hexadecimal to Decimal:

To convert a number expressed in the hexadecimal system to the decimal system, we perform the arithmetic calculations of Equation given below, that is, multiply each digit by its weight, and add the results.

$$d_0b^0 + d_1b^1 + \_ \_ \_ + d_{n-1}b^{n-1} + d_nb^n$$

where d_0 represents the least significant digit (LSD) and where d_n represents the most significant digit (MSD).

Here, b^0, b^1….. b^n are hexadecimal powers, d_0, d_1 ..... d_n are hexadecimal digits.

Ex 1: convert the hexadecimal number A7 into its equivalent decimal number.

$(A7)_{16} = 7.16^0 + A.16^1$
$= 7 + 160$
$= 167$
$(A7)_{16} = (167)_{10}$

Conversion from Octal to Hexadecimal and Vice-Versa:

we don't normally require conversion between hex and octal numbers. If we need to do this, use binary as the intermediate form, as shown below:

Hex => Binary => Octal
Octal => Binary => Hex

11. Conversion from Octal to Hexadecimal:

Ex 1: Convert the octal number 247 into its equivalent hexadecimal number.

i. First we need to convert the given octal number 247 into binary,

$(247)_8 =$ 2 4 7 / 010 100 111

$= (010100111)_2$

ii. Next we need to convert the resultant binary number 010100111 into hexadecimal ,

$(010100111)_2 =$ 0000 1010 0111
0 A 7
$= (0A7)_{16}$

Here, we can ignore the leading hexadecimal zero's at MSD positions, So answer will be $(A7)_{16}$

12. Conversion from Hexadecimal to Octal :

Ex 1: Convert the hexadecimal number A7 into its equivalent octal number.

i. First we need to convert the given hexadecimal number A7 into binary,

$(A7)_{16} =$ A 7
1010 0111

$= (10100111)_2$

ii. Next we need to convert the resultant binary number 10100111 into octal ,

$(10100111)_2 =$ 010 100 111
2 4 7

$= (247)_8$

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:

1. r's complement
2. (r - 1)'s complement.

When the value of the **base r** is substituted in the name,

i. For binary numbers, **2's and 1's complement**
ii. For decimal numbers, **10's and 9's complement.**
iii. For octal numbers, **8's and 7's complement**
iv. For hexadecimal numbers, **16's and F's complement**

(r-1)'s Complement (9's complement):

Given a number N in base r having n digits,

- **(r - 1)'s complement of N = (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.

If n = 4, we have 10^4 = 10000 and 10^4 - 1 = 9999. Then the 9' s complement of a decimal number is obtained by subtracting each digit from 9.

Ex:

- 9's complement of 546700 is 999999 - 546700 = 453299
- 9's complement of I2389 is 99999 - 12389 = 87610.

(r-1)'s Complement (1's complement):

Given a number N in base r having n digits,

- **(r - 1)'s complement of N = (r^n - 1) - N.**
- For binary numbers r = 2 and r - 1 = 1, so the **1's complement** of N is (2^n - 1) - N. Now, 2^n represents a number that consists of a single 1 followed by n 0's. 2^n - 1 is a number represented by n 1's.

If n = 4, we have 2^4 = 10000 and 2^4 - 1 = 1111. Then 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 I's into 0's and 0's into 1's.

Therefore, the 1's complement of a binary number is formed by changing 1's into 0's and 0's into 1's.

Ex:

- 1's complement of 1011001 is 0100110.
- 1's complement of 0001111 is 1110000.

r's Complement:

The r's complement of an n-digit number N in base r is defined as r^n - N for N ≠ 0 and 0 for N = 0.

- Comparing with the (r - I)'s complement, the r's complement is obtained by adding 1 to the (r - 1)'s complement.
- so , **r^n- N = [(r^n- 1) - N] + 1**.

Ex:

- **10's complement** of 2389 is 7610 + 1 = 7611. It is obtained by adding 1 to the 9' s complement value of 2389.
- **2's complement** of binary 101100 is 010011 + 1 = 010100 and is obtained by adding 1 to the 1's complement value of 101100.

Subtraction of Unsigned Numbers:

The subtraction of two n-digit unsigned numbers M – N (N ≠ 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 ≥ 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.

Ex 1: Case 1(M ≥ N):The subtraction 72532 - 13250 = 59282,

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

Ex 2: Case 1(M < N): The subtraction 13250 – 72532 = 59282

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

- There is no end carry
- Answer is negative 59282 = 10's complement of 40718

- Subtraction with complements is done with binary numbers in a similar manner using the same procedure outlined above.

Ex 1: Two binary numbers X = 1010100 and Y = 1000011, we perform the subtraction X – Y and Y - X using 2's complements:

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

$$\begin{aligned} Y &= 1000011 \\ \text{2's complement of } X &= +0101100 \\ \text{Sum} &= 1101111 \end{aligned}$$

- There is no end carry
- Answer is negative 0010001 = 2's complement of 1101111

Fixed-Point Representation:

- Positive integers, including zero, can be represented as unsigned numbers. However, to represent negative integers, we need a notation for negative values.
- Computers must represent everything with 1's and 0's, including the sign of a number. the sign bit placed in **the leftmost position of the number**.
- The convention is to make the sign bit equal to **0 for positive** and to **1 for negative**.

binary point:

In addition to the sign, a number may have a binary (or decimal) point.

The representation of the binary point in a register is complicated by the fact that it is characterized by a position in the register.

There are two ways of specifying the position of the binary point in a register:

1) **By giving it a fixed position (**or)
2) **By employing a floating-point representation.**

Fixed-point representation: This method assumes that the binary point is always fixed in one position. The two positions most widely used are (1) a binary point in the extreme left of the register to make the stored number a fraction, and (2) a binary point in the extreme right of the register to make the stored number an integer.

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

Integer Representation (Signed numbers):

When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number. When the number is negative, the sign is represented by 1 but the rest of the number may be represented in one of three possible ways:

1. Signed-magnitude representation
2. Signed-1' s complement representation
3. Signed 2' s complement representation

- 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.

Example: Representation of -14 with 8-bit register in 3-ways.

Signed-magnitude representation: 1 0001110
Signed-1' s complement representation: 1 1110001
Signed 2' s complement representation: 1 1110010

- ✓ The signed-magnitude representation of - 14 is obtained from +14 by complementing only the sign bit.
- ✓ The signed-1's complement representation of –14 is obtained by complementing all the bits of +14, including the sign bit.
- ✓ The signed-2' s complement representation is obtained by taking the 2' s complement of +14, including its sign bit.

Arithmetic Addition(2's Complement Addition):

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

Ex:

+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 is 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.

- A subtraction operation can be changed to an addition operation if the sign of the subtrahend is changed. This is demonstrated by the following relationship:

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

Example:

Consider the subtraction of (-6) - (- 13) = +7.
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 . Removing the end carry, we obtain the correct answer 00000111 (+ 7).

Overflow

- When two numbers of n digits each are added and the sum occupies n + 1 digits, we say that an **overflow** occurred. When the addition is performed with paper and pencil, an overflow is not a problem since there is no limit to the width of the page to write down the sum.
- An overflow is a problem in computers because the width of registers is finite. A result that contains n + 1 bits cannot be accommodated in a register with a length of n bits.

An overflow cannot occur after an addition if one number is positive and the other is negative,

An overflow may occur if the two numbers added are both positive or both negative.

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

An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. 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.

Decimal Fixed-Point Representation

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.

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

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. Of course, the decimal numbers inside the computer must be in BCD, including the sign digits

The subtraction of decimal numbers either unsigned or in the signed-10's complement system is the same as in the binary case. Take the 10's complement of the subtrahend and add it to the minuend.

Floating-Point Representation:

The floating-point representation of a number has two parts.

1. The first part represents a signed & fixed-point number called the **mantissa.**

2. The second part designates the position of the decimal (or binary) point and is called the **exponent**.

Example:

The decimal number + 6132.789 is represented in floating-point with a fraction and an exponent as follows:

Fraction	*Exponent*
+0.6132789	+04

- This representation is equivalent to the scientific notation +0. 6132789 X 10^{+4}.
- Floating-point is always interpreted to represent a number in the following form:

$$\mathbf{m \times r^e}$$

m- mantissa, r-radix or base of the given number and e-Exponent.

A floating-point binary number is represented in a similar manner except that it uses base 2 for the exponent.

Ex: + 1001 . 11 is represented with a n 8-bit fraction and 6-bit exponent.

Fraction	*Exponent*
01001110	000100

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

Normalization: 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.

Other Binary codes:

A **binary code** is a group of n bits that assume up to 2" distinct combinations of 1's and 0's with each combination representing one element of the set that is being coded.

- For example, a set of 4 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 8 elements requires a 3-bit code,
- A set of 16 elements requires a 4-bit code, and so on.

A binary code will have some unassigned bit combinations if the number of elements in the set is not a multiple power of 2, one of such type of code is called ***binary-coded decimal*** *and is commonly referred to by its abbreviation* ***BCD****.*

The below table shows the BCD numbers

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

Alphanumeric Representation: Many applications of digital computers require the handling of data that consist not only of numbers, but also letters of the alphabet and certain special characters. An ***alphanumeric character*** set is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet and a number of special characters, such as $, + , and = .

The standard alphanumeric binary code is the **ASCII (American Standard Code for Information Interchange)**, which uses seven bits to code 128 characters. The binary code for the uppercase letters, the decimal digits, and a few special characters is listed in Table shown below.

Some of the ASCII codes are shown in the below table

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

Digital computers also employ other binary codes for special applications. A few additional binary codes encountered in digital computers are:

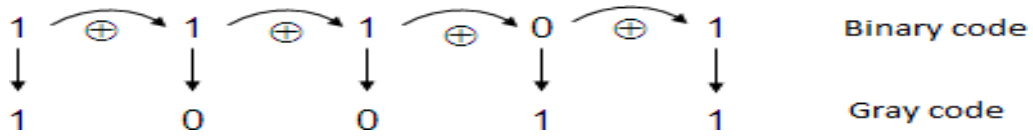
1. Gray code
2. Other decimal codes
3. Other alphanumeric codes

1. **Gray code:** It is a kind of binary number system in which every successive pair of numbers differs in only one bit.

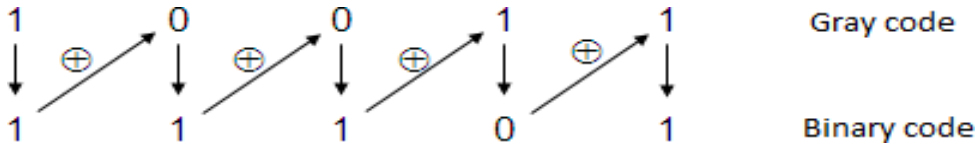
The advantage of the Gray code over straight binary numbers is that the Gray code changes by only one bit as it sequences from one number to the next.

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 code to Gray code:



Gray code to Binary code:



2 Other decimal codes:

i. **Weighted code:** The 2421 is an example of a weighted code. In a weighted code, the bits are multiplied by the weights indicated and the sum of the weighted bits gives the decimal digit.

EX: The bit combination 1101, when weighted by the respective digits 2421, gives the decimal equivalent of 2 x 1 + 4 x 1 + 2 x 0 + 1 + 1 = 7. The BCD code can be assigned the weights 8421 and for this reason it is sometimes called the 8421 code .

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

iii. **Self-complementing code:** The 9' s complement is easily obtained with the 2421 and the excess-3 codes are self-complementing.

A self-complementing property means that *the 9' s complement of a decimal number, when represented in one of these codes, is easily obtained by changing 1's to 0's and 0's to 1's.*

Four different binary codes for the decimal digit are shown in the below table

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

3. Other alphanumeric codes:

The ASCII code is the standard code commonly used for the transmission of binary information. Each character is represented by a 7-bit code and usually an eighth bit is inserted for parity. The code consists of 128 characters.

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

Error Detection Codes:

- Binary information transmitted through some form of communication medium is subject to external noise that could change bits from 1 to 0, and vice versa.
- 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.
- The most common error detection code used is the **parity bit** . A parity bit is an extra bit included with a binary message to make the total number of 1's either odd or even.

A message of 3 bits and 2 possible parity bits is shown in the below Table.

- ✓ The P(odd) bit is chosen to make the sum of 1's (in all four bits) odd .
- ✓ The P(even) bit is chosen to make the sum of all 1's even. In either case, the sum is taken over the message and the P bit.

Message xyz	P(odd)	P(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

- ❖ **Disadvantage (even-parity):** It is having a bit combination of all 0's, while in the odd parity there is always one bit (of the four bits that constitute the message and P) that is 1.

During transfer of information from one location to another, the parity bit is handled as follows.

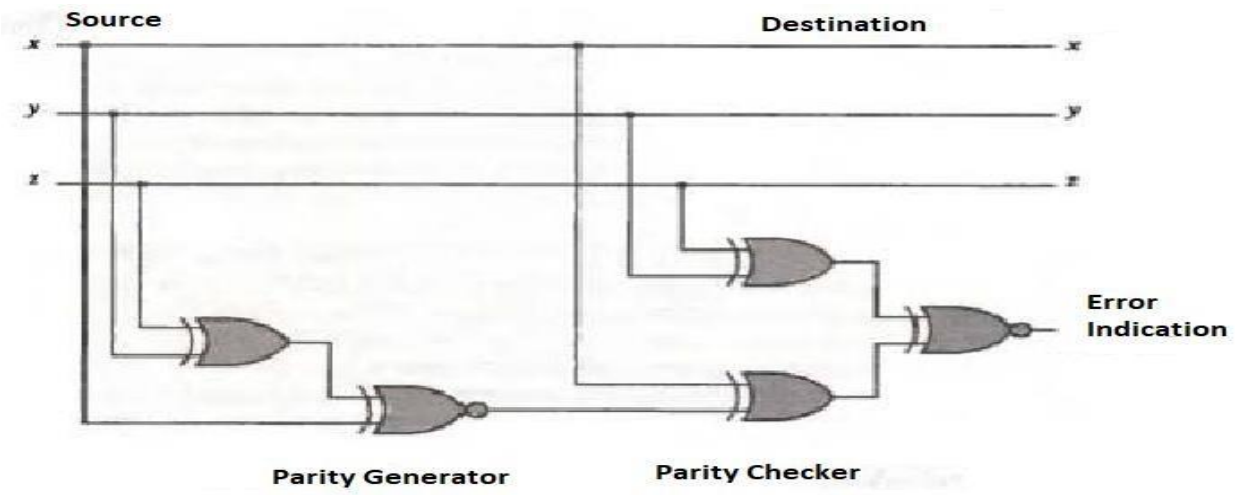
Parity Generator:

- At the sending end, the message (in our case 3-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.

Parity Checker:

- At the receiving end, all the incoming bits (in our case, 4-bits) 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.

Parity generator and checker networks are logic circuits constructed with exclusive-OR functions.



- ❖ The exclusive- OR function of three or more variables is by definition an odd function.
- ❖ An odd function is a logic function whose value is binary 1 if and only if, an odd number of variables are equal to 1.

EX: consider a 3-bit message to be transmitted with an odd parity bit. At the sending end, the odd parity bit is generated by a parity generator circuit.

- ❑ The message and the odd-parity bit are transmitted to their destination where they are applied to a parity checker. An error has occurred during transmission if the parity of the four bits received is even, since the binary information transmitted was originally odd.

Computer Aritnmetic:

Introduction:

- ➢ Arithmetic instructions in digital computers manipulate data to produce results necessary for the solution of computational problems.
- ➢ These instructions perform arithmetic calculations and are responsible for the bulk of activity involved in processing data in a computer.

The four basic arithmetic operations are **addition, subtraction, multiplication and division.** From these four bulk operations, it is possible to formulate other arithmetic functions and solve scientific problems by means of numerical analysis methods.

- ➢ An arithmetic processor is the part of a processor unit that executes arithmetic operations. The data type assumed to reside in **processor registers** during the execution of an arithmetic instruction is specified in the definition of the instruction. A:n arithmetic instruction may specify binary or decimal data, and in each case the data may be in fixed-point or floating-point form.
- ➢ We must be thoroughly familiar with the sequence of steps to be followed in order to carry out the operation and achieve a correct result. The solution to any problem that is stated by a finite number of well-defined procedural steps is called an **algorithm**.
- ➢ Usually, an algorithm will contain a number of procedural steps which are dependent on results of previous steps. A convenient method for presenting algorithms is a **flowchart**.

Addition and Subtraction:

- As we have discussed, 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-2's complement representation when performing arithmetic operations with integers.

i. Addition and Subtraction with Signed-Magnitude Data:

When 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 shown below.

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

Algorithm: (Addition with Signed-Magnitude Data)

i. When the signs of A and B are identical ,add the two magnitudes and attach the sign of A to the result.

ii. When the signs of A and B are different, 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.

iii. If the two magnitudes are equal, subtract B from A and make the sign of the result positive.

Algorithm: (Subtraction with Signed-Magnitude Data)

i. When the signs of A and B are different, add the two magnitudes and attach the sign of A to the result.

ii. When the signs of A and B are 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.

iii. If the two magnitudes are equal, subtract B from A and make the sign of the result positive.

The two algorithms are similar except for the sign comparison. The procedure to be followed for identical signs in the addition algorithm is the same as for different signs in the subtraction algorithm, and vice versa.

Hardware Implementation:

To implement the two arithmetic operations with hardware, it is first necessary that the two numbers be stored in registers.

i. Let A and B be two registers that hold the magnitudes of the numbers, and A_S and B_S be two flip-flops that hold the corresponding signs.

ii. The result of the operation may be transferred to a third register: however, a saving is achieved if the result is transferred into A and A_S. Thus A and A_S together form an accumulator register.

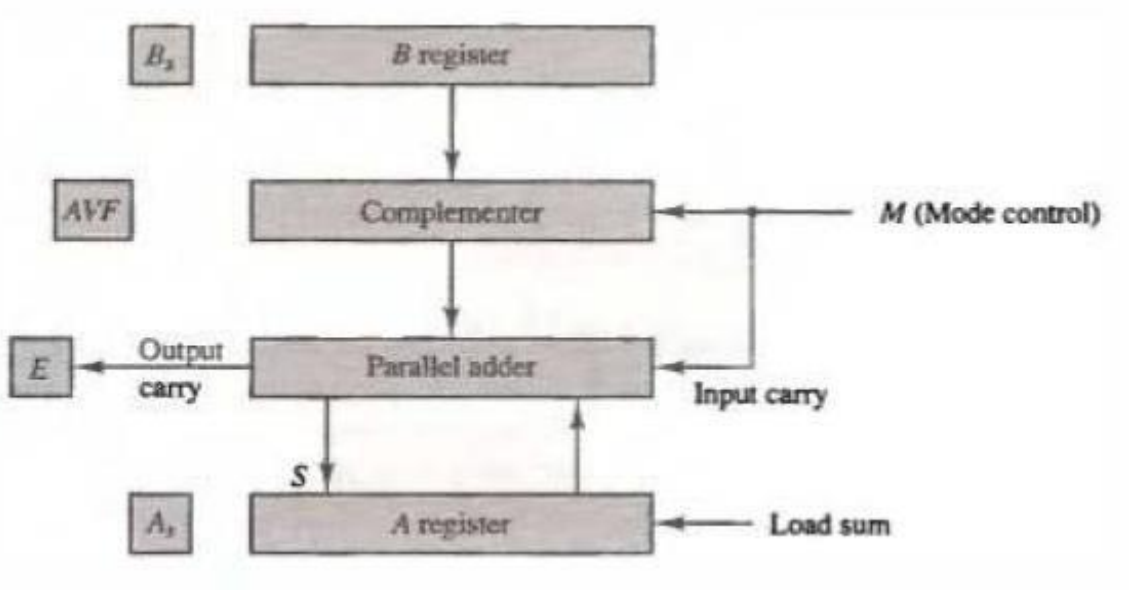
Consider now the hardware implementation of the algorithms above.

- First, **a parallel-adder** is needed to perform the microoperation A + B.
- Second, **a comparator circuit** is needed to establish if A > B, A = B, or A < B.
- Third, **two parallel-subtractor circuits** are needed to perform the microoperations A - B and B - A. The sign relationship can be determined from an exclusive-OR gate with A_S and B_S as inputs.

The below figure 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.

Figure (i): Hardware for addition and subtraction with Signed-Magnitude Data



The complementer 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 l's complement of B is applied to the adder, the input carry is 1, and output $S = A + \overline{B} + 1$. This is equal to A plus the 2's complement of B, which is equivalent to the subtraction A - B.

Hardware Algorithm

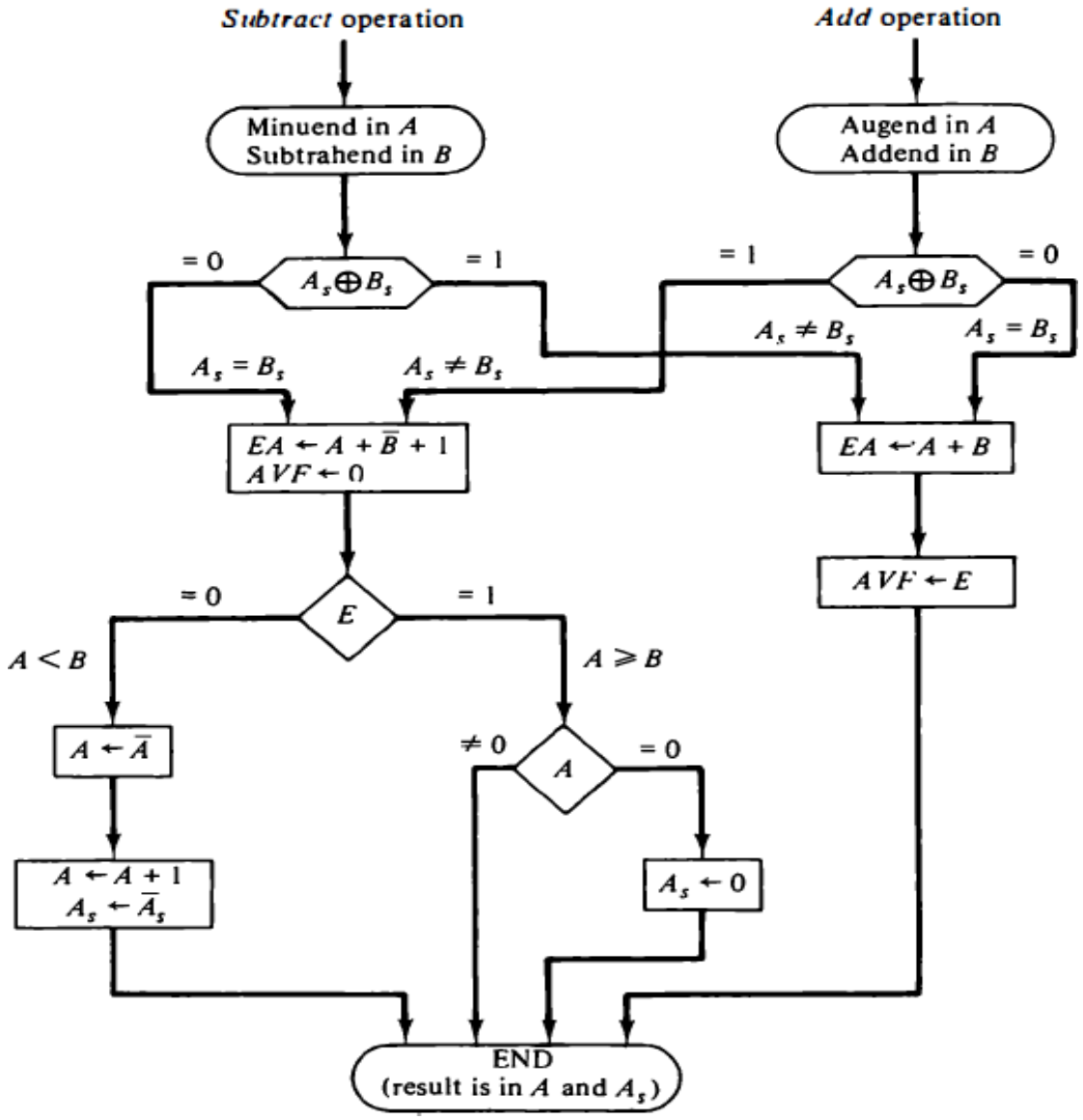
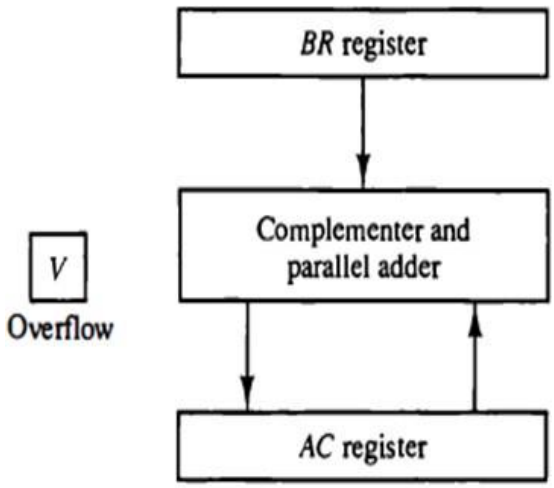


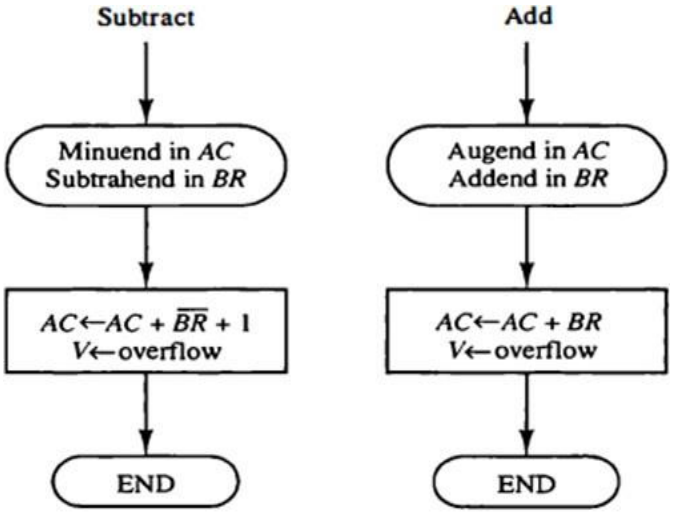
Figure (j): Flowchart for add and subtract operations

ii. Addition and Subtraction with Signed-2's Complement Data

- The register configuration for the hardware implementation is shown in the below Figure(a). We name the A register AC (accumulator) and the B register BR. The leftmost 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 for adding and subtracting two binary numbers in signed-2' s complement representation is shown in the flowchart of Figure(b). The sum is obtained by adding the contents of AC and BR (including their sign bits). The overflow bit V is set to 1 if the exclusive-OR of the last two carries is 1, and it is cleared to 0 otherwise. The subtraction operation is accomplished by adding the content of AC to the 2's complement of BR.
- Comparing this algorithm with its signed-magnitude counterpart, we note that it is much simpler to add and subtract numbers if negative numbers are maintained in signed-2' s complement representation.



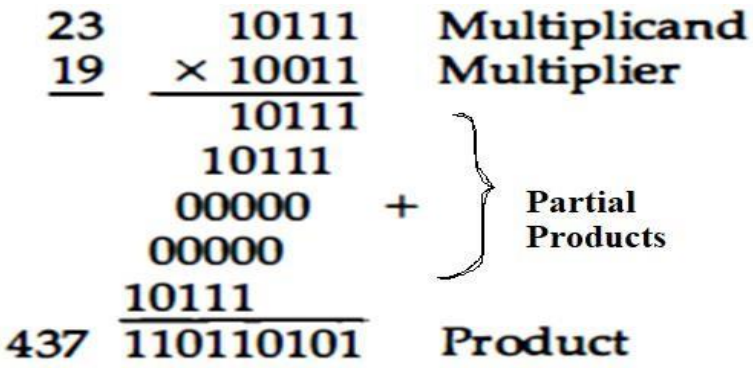
Figure(a): Hardware for addition & subtraction of 2's complement numbers



Figure(b): Algorithm for adding & subtracting of 2's complement numbers

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 **adds** operations. This process is best illustrated with a numerical example.



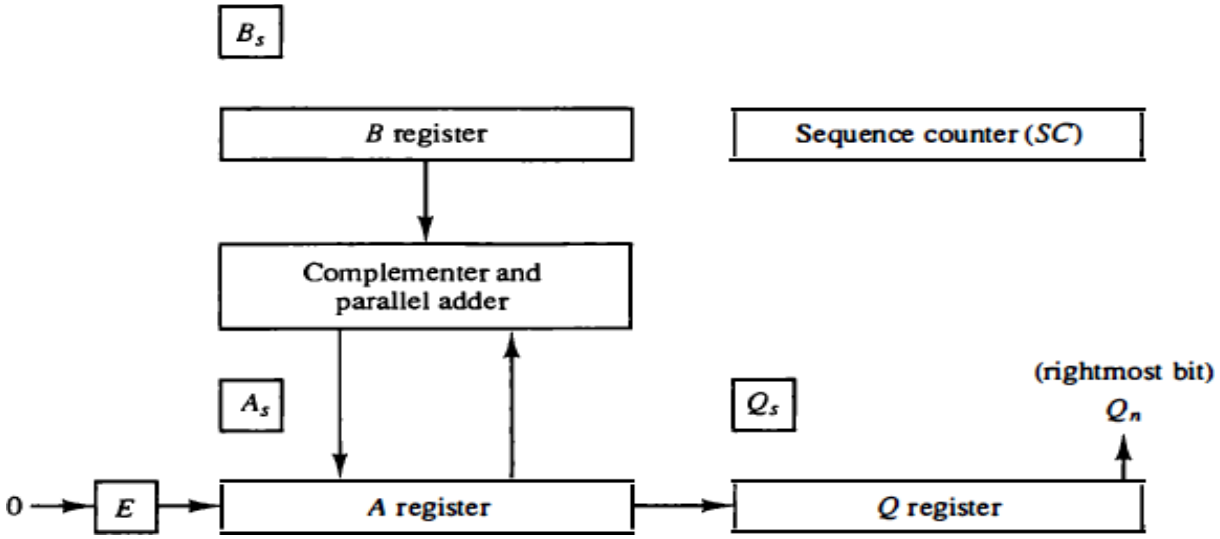
The process of multiplication:

- It consists of looking at successive bits of the multiplier, least significant bit first.
- If the multiplier bit is a 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 number.
- Finally, the numbers are added and their sum forms the product.

The sign of the product is determined from the signs of the multiplicand and multiplier. If they are alike, the sign of the product is **positive**. If they are unlike, the sign of the product is **negative**.

Hardware Implementation for Signed-Magnitude Data

→ The registers A, B and other equipment are shown in Figure (a). The multiplier is stored in the Q register and its sign in Qs. The sequence counter SC is initially set to a number equal to the number of bits in the multiplier. The counter is decremented by 1 after forming each partial product. When the content of the counter reaches zero, the product is formed and the process stops.



Figure(k): Hardware for multiply operation.

- ➔ Initially, the **multiplicand** is in register B and the **multiplier** in Q, Their corresponding signs are in Bs and Qs, respectively
- ➔ 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 the right. This shift will be denoted by the statement shr EAQ to designate the right shift.
- ➔ 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, pushing 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.

Hardware Algorithm:

→Initially, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in Bs and Qs, 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.

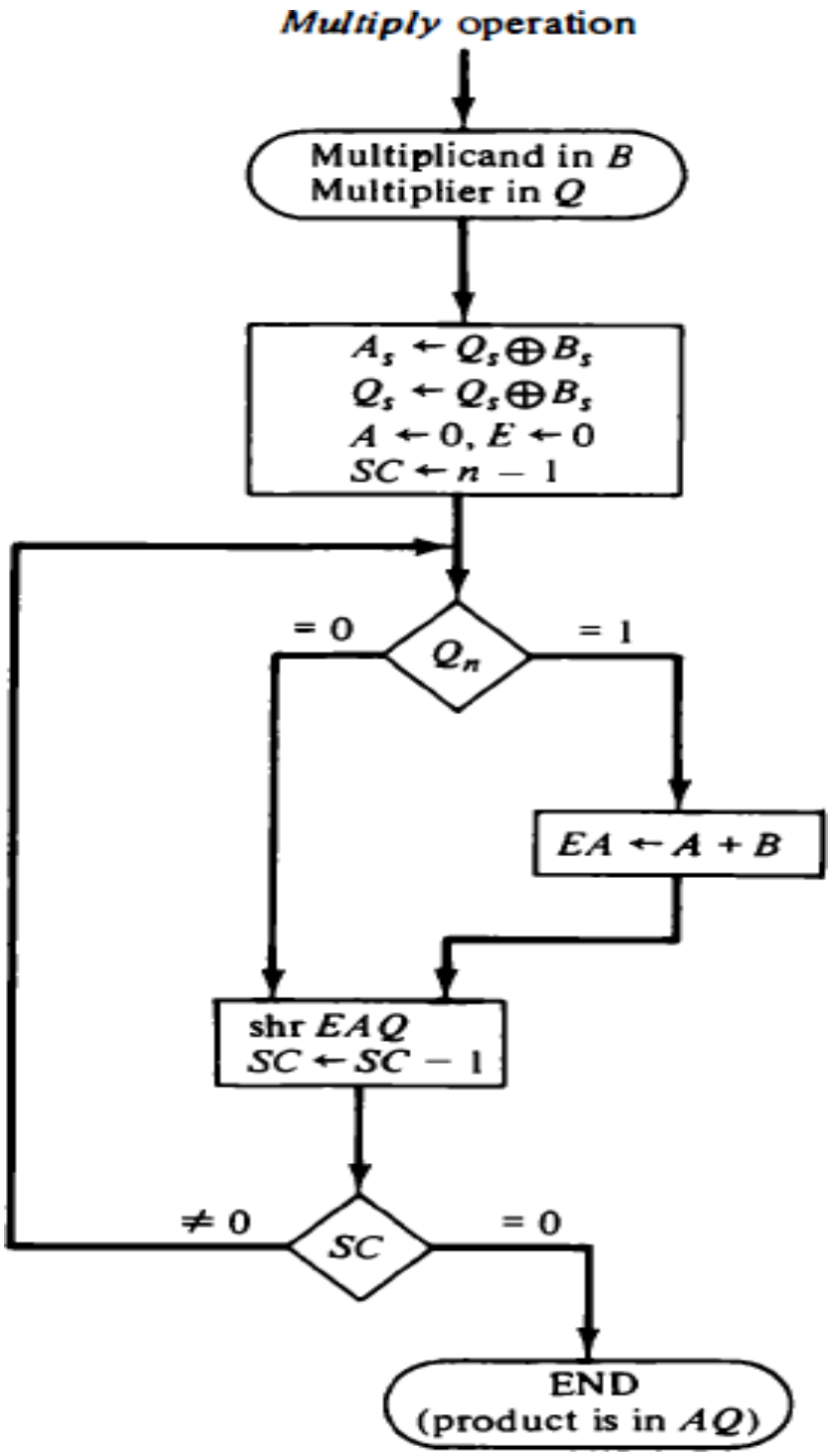
→After the initialization, the low-order bit of the multiplier in Qn is tested.

i. If it is 1, the multiplicand in B is added to the present partial product in A .
ii. If it is 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.

→The final product is available in both A and Q, with A holding the most significant bits and Q holding the least significant bits.

A flowchart of the hardware multiply algorithm is shown in the below figure (l).



Figure(l): Flowchart for multiply operation.

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

Figure (m): Numerical Example of multiplication

Booth Multiplication Algorithm:(multiplication of 2's complement data):

→Booth algorithm gives a procedure for multiplying binary integers in signed-2's complement representation.

→Booth algorithm requires examination of the multiplier bits and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to the **following rules**:

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier.
2. The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous 1) in a string of O's in the multiplier.
3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

Hardware implementation of Booth algorithm Multiplication:

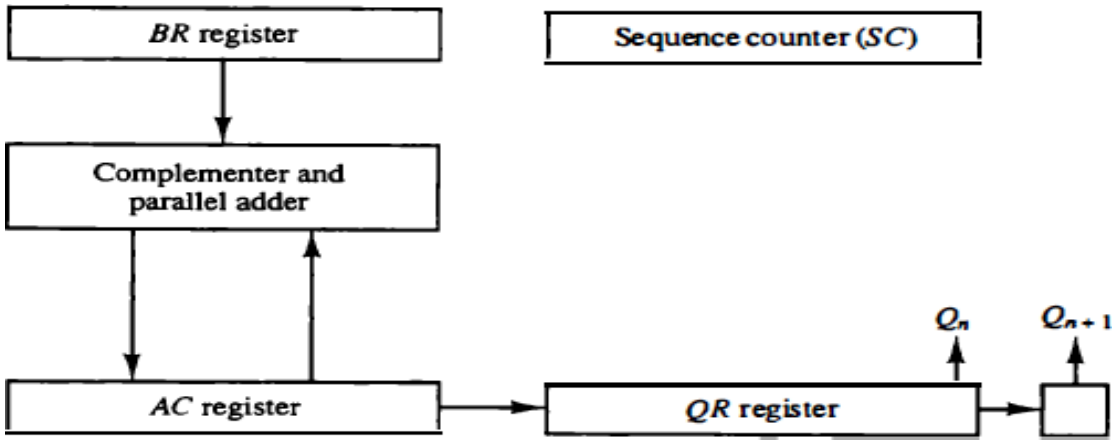


Figure (n): Hardware for Booth Algorithm

The hardware implementation of Booth algorithm requires the register configuration shown in figure (n). This is similar addition and subtraction hardware except that the sign bits are not separated from the rest of the registers. To show this difference, we rename registers A, B, and Q, as 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. The flowchart for Booth algorithm is shown in Figure (o).

Hardware Algorithm for Booth Multiplication:

→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.

i. If the two bits are equal to 10, it means that the first 1 in a string of 1's has been encountered. This requires a subtraction of the multiplicand from the partial product in AC.
ii. If the two bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product in AC.
iii. When the two bits are equal, the partial product does not change.
iv. 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.

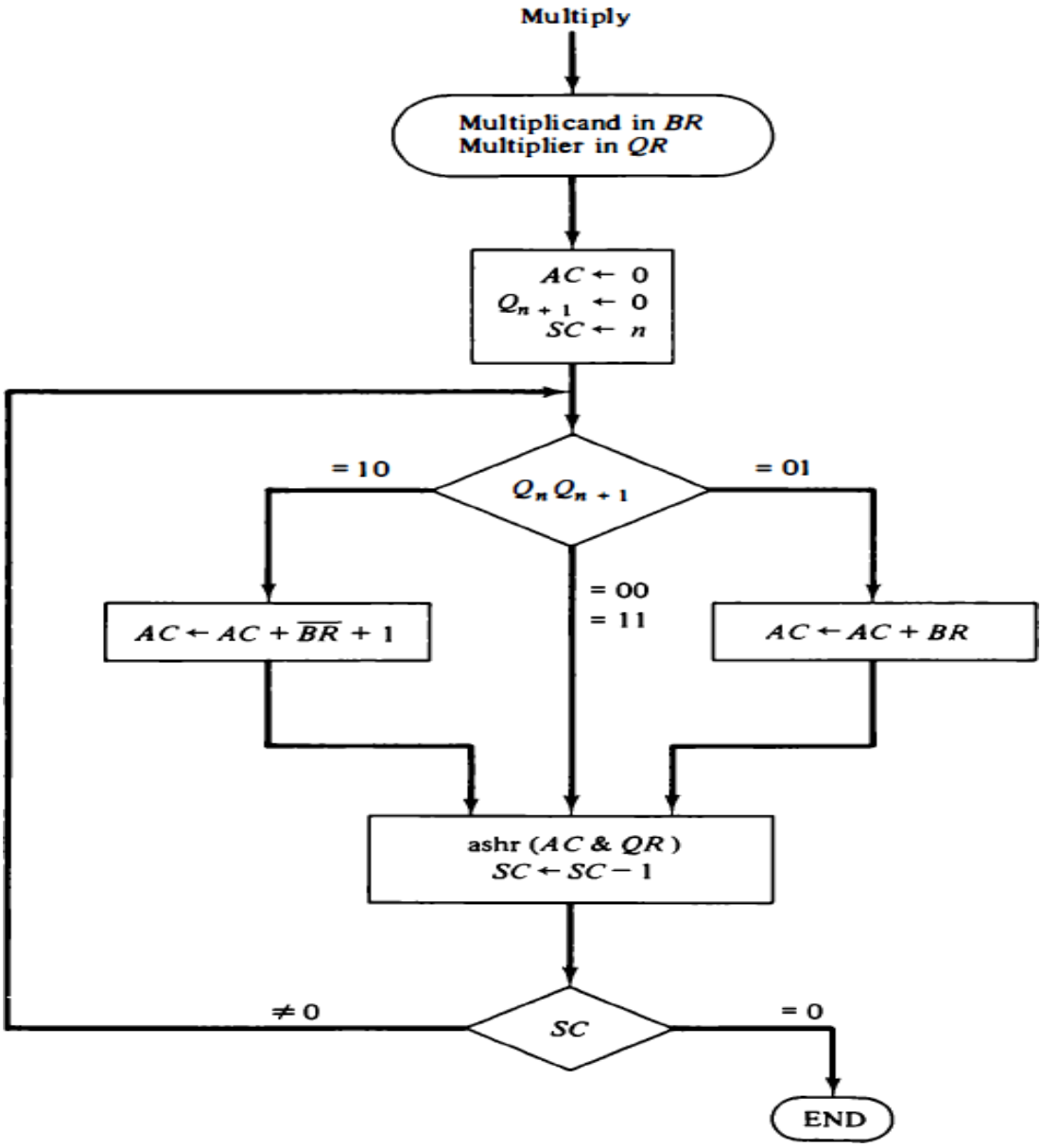


Figure (o): Booth Algorithm for multiplication of 2's complement numbers

Example: multiplication of (- 9) x (- 13) = + 117 is shown below. Note that the multiplier in QR is negative and that the multiplicand in BR is also negative. The 10-bit product appears in AC and QR and is positive.

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

Figure (p): Example of Multiplication with Booth Algorithm.

Division Algorithms:

- Division of two fixed-point binary numbers in signed-magnitude representation is done with paper and pencil by a process of successive compare, shift, and subtract operations.

The division process is illustrated by a numerical example in the below figure (q).

- The divisor B consists of five bits and the dividend A consists of ten bits. The five most significant bits of the dividend are **compared** with the divisor. Since the 5-bit number is smaller than B, we try again by taking the sixth most significant bits of A and compare this number with B. The 6-bit number is greater than B, so we place a 1 for the quotient bit. The divisor is then shifted once to the right and subtracted from the dividend.

- The difference is called a **partial remainder** because the division could have stopped here to obtain a quotient of 1 and a remainder equal to the partial remainder. The process is continued by comparing a partial remainder with the divisor.

- 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. Note that the result gives both a quotient and a remainder.

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

Figure (g): Example of Binary Division

Hardware Implementation for Signed-Magnitude Data:

The hardware for implementing the division operation is identical to that required for multiplication.

- ✓ 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 information about the relative magnitude is available in E.
- ✓ If E = 1, it signifies that A≥B. A quotient bit 1 is inserted into Q, and the partial remainder is shifted to the left to repeat the process.
- ✓ If E = 0, it signifies that A < B so the quotient in Qn remains a 0. The value of B is then added to restore the partial remainder in A to its previous value. The partial remainder is shifted to the left and the process is repeated again until all five quotient bits are formed.
- ✓ 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 the final remainder is in A.

The sign of the quotient is determined from the signs of the dividend and the divisor. If the two signs are alike, the sign o f the quotient is **plus**. If they are unalike, the sign is **minus**. The sign of the remainder is the same as the sign of the dividend.

Divide Overflow

- ❑ The division operation may result in a quotient with an overflow. This is not a problem when working with paper and pencil but is critical when the operation is implemented with hardware. This is because the length of registers is finite and will not hold a number that exceeds the standard length.
- ❑ To see this, consider a system that has 5-bit registers. We use one register to hold the divisor and two registers to hold the dividend. From the example shown in the above, we note that the quotient will consist of six bits if the five most significant bits of the dividend constitute a number greater than the divisor. The quotient is to be stored in a standard 5-bit register, so the overflow bit will require one more flip-flop for storing the sixth bit.
- ❑ This divide-overflow condition must be avoided in normal computer operations because the entire quotient will be too long for transfer into a memory unit that has words of standard length, that is, the same as the length of registers.
- ❑ This condition detection must be included in either the hardware or the software of the computer, or in a combination of the two.

When the dividend is twice as long as the divisor,

i. A divide-overflow condition occurs if the high-order half bits of the dividend constitute a number greater than or equal to the divisor.
ii. A division by zero must be avoided. This occurs because any dividend will be greater than or equal to a divisor which is equal to zero. Overflow condition is usually detected when a special flip-flop is set. We will call it a divide-overflow flip-flop and label it **DVF**.

Hardware Algorithm:

1. The dividend is in A and Q and the divisor in B . The sign of the result is transferred into Qs to be part of the quotient. A constant is set into the sequence counter SC to specify the number of bits in the quotient.

2. 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 ≥ 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.

3. 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, we know that EA > B because EA consists of a 1 followed by n-1 bits while B consists of only n -1 bits. In this case, B must be subtracted from EA and 1 inserted into Qn for the quotient bit.

4. 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 ≥ B; therefore, Qn is set to 1 . If E = 0, it signifies that A < B and the original number is restored by adding B to A . In the latter case we leave a 0 in Qn.

This process is repeated again with registers EAQ. After n times, the quotient is formed in register Q and the remainder is found in register A

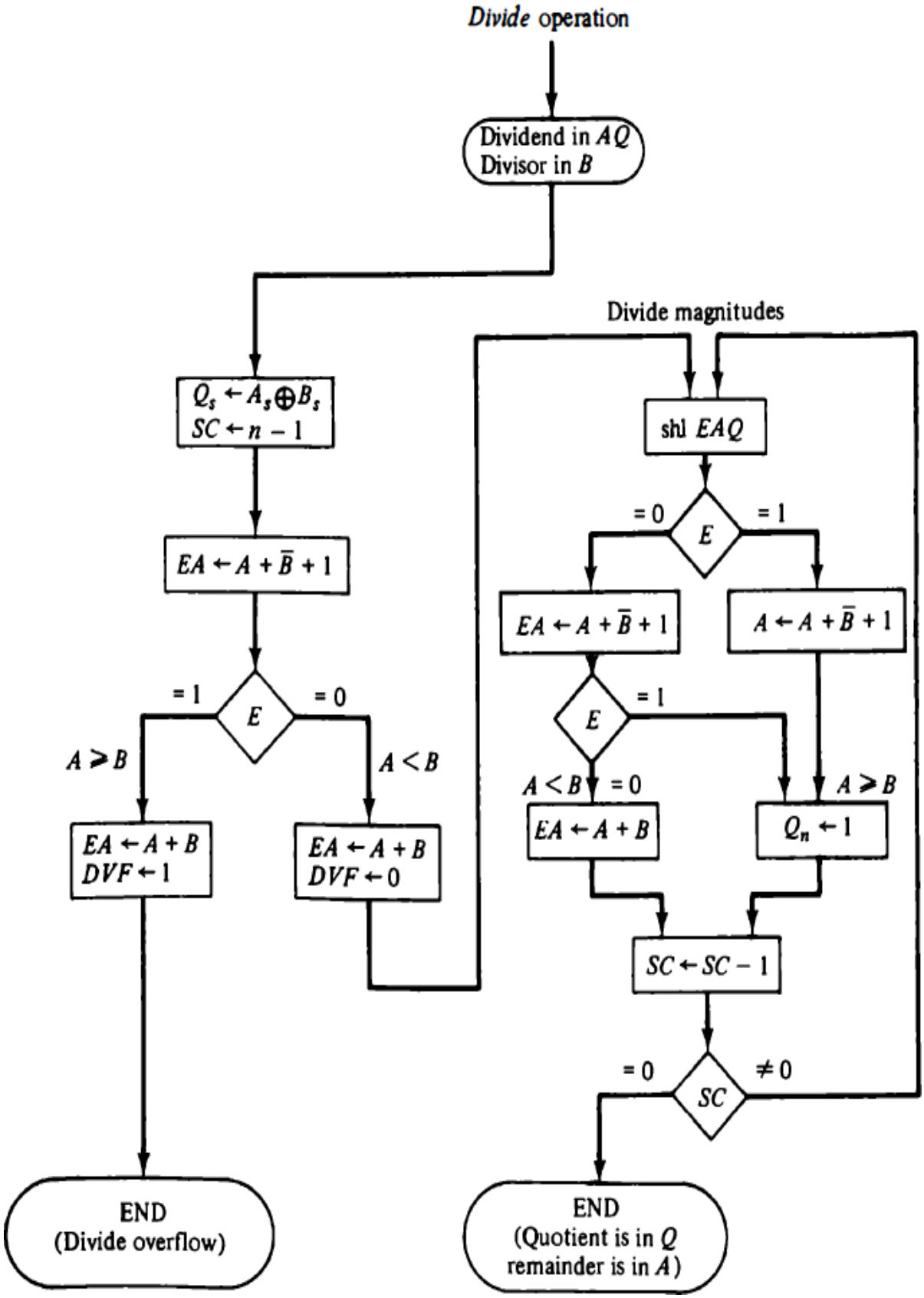


Figure (r): Flowchart for Divide operation

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		2
Restore remainder	1	01010		
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	

Figure (s): Example of Binary Division