UNIT – III

Data Representation:

3.1 – Data Types:

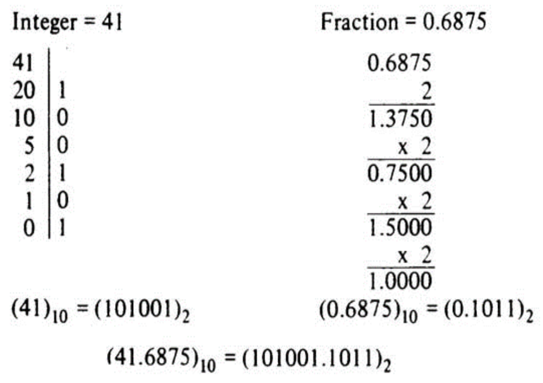
- Registers contain either data or control information
- Control information is a bit or group of bits used to specify the sequence of command signals needed for data manipulation
- Data are numbers and other binary-coded information that are operated on
- Possible data types in registers:
  - ➢ Numbers used in computations
  - ➢ Letters of the alphabet used in data processing
  - ➢ Other discrete symbols used for specific purposes
- All types of data, except binary numbers, are represented in binary-coded form
- 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
- The string of digits 724.5 represents the quantity 7 x 102 + 2 x 101 + 4 x 100 + 5 x10-1
- The string of digits 101101 in the binary number system represents the quantity

  1 x 25 + 0 x 24 + 1 x 23 + 1 x 22 + 0 x 21 + 1 x 20 = 45
- (101101)2 = (45)10
- We will also use the octal (radix 8) and hexidecimal (radix 16) number systems

  (736.4)8 = 7 x 82 + 3 x 81 + 6 x 80 + 4 x 8-1 = (478.5)10

  (F3)16 = F x 161 + 3 x 160 = (243)10
- Conversion from decimal to radix r system is carried out by separating the number into its integer and fraction parts and converting each part separately
- Divide the integer successively by r and accumulate the remainders
- Multiply the fraction successively by r until the fraction becomes zero

Figure 3-1 Conversion of decimal 41.6875 into binary.



```
Integer = 41              Fraction = 0.6875

41 |                         0.6875
20 | 1                            2
10 | 0                       1.3750
 5 | 0                          x 2
 2 | 1                       0.7500
 1 | 0                          x 2
 0 | 1                       1.5000
                                x 2
                             1.0000
```

$(41)_{10} = (101001)_2$ $(0.6875)_{10} = (0.1011)_2$

$(41.6875)_{10} = (101001.1011)_2$

• Each octal digit corresponds to three binary digits

• Each hexadecimal digit corresponds to four binary digits

• Rather than specifying numbers in binary form, refer to them in octal or hexadecimal and reduce the number of digits by 1/3 or ¼, respectively Computer

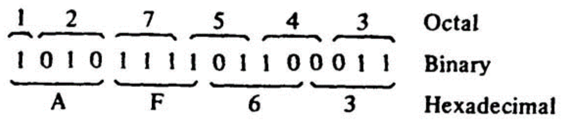
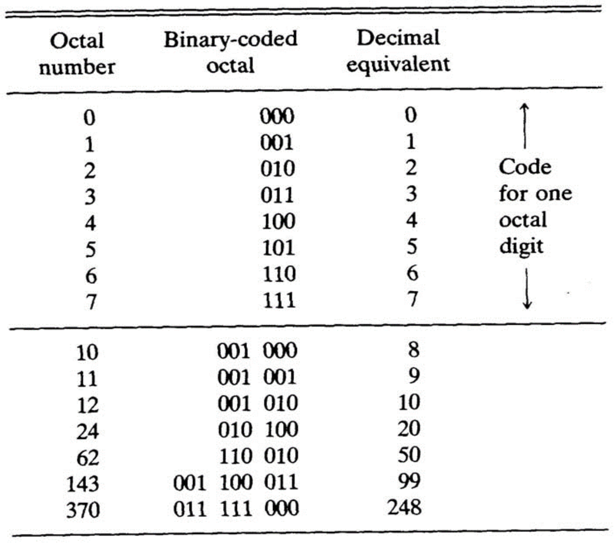


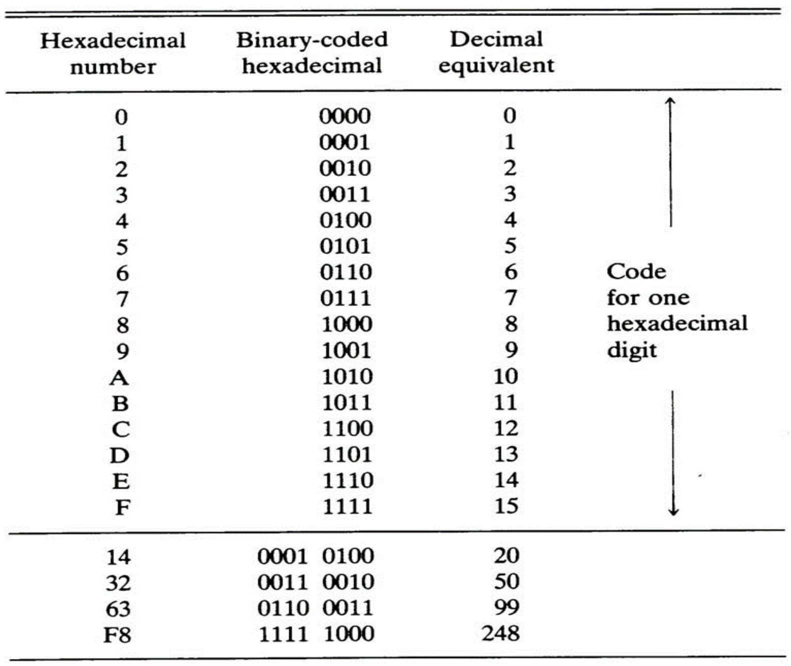
Figure 3-2 Binary, octal, and hexadecimal conversion.

TABLE 3-1 Binary-Coded Octal Numbers



Octal number	Binary-coded octal	Decimal equivalent	
0	000	0	↑
1	001	1	
2	010	2	Code
3	011	3	for one
4	100	4	octal
5	101	5	digit
6	110	6	
7	111	7	↓
10	001 000	8	
11	001 001	9	
12	001 010	10	
24	010 100	20	
62	110 010	50	
143	001 100 011	99	
370	011 111 000	248	

TABLE 3-2 Binary-Coded Hexadecimal Numbers



Hexadecimal number	Binary-coded hexadecimal	Decimal equivalent	
0	0000	0	↑
1	0001	1	
2	0010	2	
3	0011	3	
4	0100	4	
5	0101	5	
6	0110	6	Code
7	0111	7	for one
8	1000	8	hexadecimal
9	1001	9	digit
A	1010	10	
B	1011	11	
C	1100	12	
D	1101	13	
E	1110	14	
F	1111	15	↓
14	0001 0100	20	
32	0011 0010	50	
63	0110 0011	99	
F8	1111 1000	248	

• A binary code is a group of n bits that assume up to 2n distinct combinations

• A four bit code is necessary to represent the ten decimal digits – 6 are unused

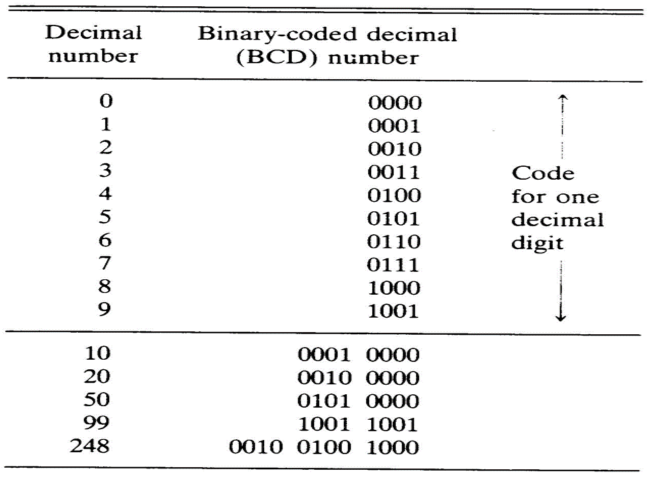
• The most popular decimal code is called binary-coded decimal (BCD)

• BCD is different from converting a decimal number to binary

• For example 99, when converted to binary, is 1100011

• 99 when represented in BCD is 1001 1001

TABLE 3-3 Binary-Coded Decimal (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	

• The standard alphanumeric binary code is ASCII

• This uses seven bits to code 128 characters

• Binary codes are required since registers can hold binary information only Computer

3.2 – Complements

- Complements are used in digital computers for simplifying subtraction and logical manipulation
- Two types of complements for each base r system: r's complement and (r – 1)'s complement
- Given a number N in base r having n digits, the (r – 1)'s complement of N is defined as (rn – 1) – N
- For decimal, the 9's complement of N is (10n – 1) – N
- The 9's complement of 546700 is 999999 – 546700 = 453299
- The 9's complement of 453299 is 999999 – 453299 = 546700
- For binary, the 1's complement of N is (2n – 1) – N
- The 1's complement of 1011001 is 1111111 – 1011001 = 0100110
- The 1's complement is the true complement of the number – just toggle all bits
- The r's complement of an n-digit number N in base r is defined as rn – N
- This is the same as adding 1 to the (r – 1)'s complement
- The 10's complement of 2389 is 9999 – 2389(9's compliment+1) = 7610 + 1 = 7611
- The 2's complement of 101100 is 111111-1011100 (1's compliment+1) = 010011 + 1 = 010100

3.3 Fixed-Point Representation:

- Positive integers and zero can be represented by unsigned numbers
- Negative numbers must be represented by signed numbers since + and – signs are not available, only 1's and 0's are
- Signed numbers have MSB as 0 for positive and 1 for negative – MSB is the sign bit
- Two ways to designate binary point position in a register
  1. Fixed point position
  2. Floating-point representation
- Fixed point position usually uses one of the two following positions
  - ➢ A binary point in the extreme left of the register to make it a fraction
  - ➢ A binary point in the extreme right of the register to make it an integer
  - ➢ In both cases, a binary point is not actually present
- The floating-point representations uses a second register to designate the position of the binary point in the first register

• When an integer is positive, the msb, or sign bit, is 0 and the remaining bits represent the magnitude

• When an integer is negative, the msb, or sign bit, is 1, but the rest of the number can be represented in one of three ways

- ➢ Signed-magnitude representation
- ➢ Signed-1's complement representation
- ➢ Signed-2's complement representation

• Consider an 8-bit register and the number +14

* The only way to represent it is 00001110

• Consider an 8-bit register and the number –14

* Signed magnitude: 1 0001110

* Signed 1's complement: 1 1110001

* Signed 2's complement: 1 1110010

• Typically use signed 2's complement

• Addition of two signed-magnitude numbers follow the normal rules

* If same signs, add the two magnitudes and use the common sign

* Differing signs, subtract the smaller from the larger and use the sign of the larger magnitude

* Must compare the signs and magnitudes and then either add or subtract

• Addition of two signed 2's complement numbers does not require a comparison or subtraction – only addition and complementation

- ➢ Add the two numbers, including their sign bits
- ➢ Discard any carry out of the sign bit position
- ➢ All negative numbers must be in the 2's complement form
- ➢ If the sum obtained is negative, then it is in 2's complement for

+6 00000110 -6 11111010

+13 00001101 +13 00001101

+19 00010011 +7 00000111

+6 00000110 -6 11111010

-13 11110011 -13 11110011

-7 11111001 -19 11101101

- Subtraction of two signed 2's complement numbers is as follows
    - ➢ Take the 2's complement form of the subtrahend (including sign bit)
    - ➢ Add it to the minuend (including the sign bit)
    - ➢ A carry out of the sign bit position is discarded
- An overflow occurs when two numbers of n digits each are added and the sum occupies n + 1 digits
- Overflows are problems since the width of a register is finite
- Therefore, a flag is set if this occurs and can be checked by the user
- Detection of an overflow depends on if the numbers are signed or unsigned
- For unsigned numbers, an overflow is detected from the end carry out of the msb
- For addition of signed numbers, an overflow cannot occur if one is positive and one is negative – both have to have the same sign
- An overflow can be detected if the carry into the sign bit position and the carry out of the sign bit position are not equal

  +70 0 1000110 -70 1 0111010

  +80 0 1010000 -80 1 0110000

  +150 1 0010110 -150 0 1101010
- The representation of decimal numbers in registers is a function of the binary code used to represent a decimal digit
- A 4-bit decimal code requires four flip-flops for each decimal digit
- This takes much more space than the equivalent binary representation and the circuits required to perform decimal arithmetic are more complex
- Representation of signed decimal numbers in BCD is similar to the representation of signed numbers in binary
- Either signed magnitude or signed complement systems
- The sign of a number is represented with four bits

- 0000 for +
- 1001 for –

• To obtain the 10’s complement of a BCD number, first take the 9’s complement and then add one to the least significant digit

• Example: (+375) + (-240) = +135

0 375 (0000 0011 0111 1010) BCD

+9 760 (1001 0111 0110 0000) BCD

0 135 (0000 0001 0011 0101) BCD

3.4 – Floating-Point Representation:

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

• The first part represents a signed, fixed-point number – the mantissa

• The second part designates the position of the binary point – the exponent

• The mantissa may be a fraction or an integer

• Example: the decimal number +6132.789 is

- Fraction: +0.6123789
- Exponent: +04
- Equivalent to +0.6132789 x 10+4

• A floating-point number is always interpreted to represent m x re

• Example: the binary number +1001.11 (with 8-bit fraction and 6-bit exponent)

- Fraction: 01001110
- Exponent: 000100
- Equivalent to +(.1001110)2 x 2+4

• A floating-point number is said to be normalized if the most significant digit of the mantissa is nonzero

• The decimal number 350 is normalized, 00350 is not

• The 8-bit number 00011010 is not normalized

• Normalize it by fraction = 11010000 and exponent = -3

• Normalized numbers provide the maximum possible precision for the floating-point number

COMPUTER ARITHMETIC :

Introduction:

- 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.

Addition and Subtraction :

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.

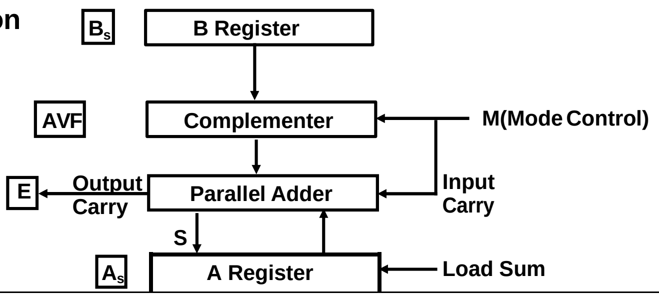
The algorithms for addition and subtraction are derived from the table and can be stated as follows (the words parentheses should be used for the subtraction algorithm)

SIGNED MAGNITUDEADDITION AND SUBTRACTION

Addition: A + B ; A: Augend; B: Addend
Subtraction: A - B: A: Minuend; B: Subtrahend

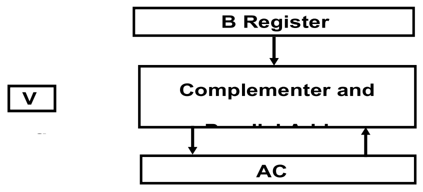
Operation	Add Magnitude	Subtract Magnitude		
		When A>B	When A<B	When A=B
(+A) + (+B)	+(A + B)			
(+A) + (- B)		+(A - B)	- (B - A)	+(A - B)
(- A) + (+B)		- (A - B)	+(B - A)	+(A - B)
(- A) + (- B)	- (A + B)			
(+A) - (+B)		+(A - B)	- (B - A)	+(A - B)
(+A) - (- B)	+(A + B)			
(- A) - (+B)	- (A + B)			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

Hardware Implementation

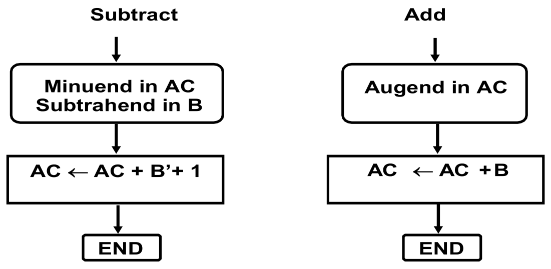


SIGNED 2'S COMPLEMENT ADDITION AND SUBTRACTION

Hardware



Algorithm



Algorithm:

- The flowchart is shown in Figure 7.1. The two signs A, and B, are compared by an exclusive-OR gate.

  If the output of the gate is 0 the signs are identical; If it is 1, the signs are different.

- For an add operation, identical signs dictate that the magnitudes be added.For a subtract operation, different signs dictate that the magnitudes be added.

- The magnitudes are added with a microoperation EA A + B, where EA is a register that combines E and A. The carry in E after the addition constitutes an overflow if it is equal to 1. The value of E is transferred into the add-overflow flip-flop AVF.

- The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation. The magnitudes are subtracted by adding A to the 2's complemented B. No overflow can occur if the numbers are subtracted so AVF is cleared to 0.

- 1 in E indicates that A >= B and the number in A is the correct result. If this numbs is zero, the sign A must be made positive to avoid a negative zero.

- 0 in E indicates that A < B. For this case it is necessary to take the 2's complement of the value in A. The operation can be done with one microoperation A A' +1.

- However, we assume that the A register has circuits for microoperations complement and increment, so the 2's complement is obtained from these two microoperations.

- In other paths of the flowchart, the sign of the result is the same as the sign of A. so no change in A is required. However, when A < B, the sign of the result is the complement of the original sign of A. It is then necessary to complement A, to obtain the correct sign.

- The final result is found in register A and its sign in As. The value in AVF provides an overflow indication. The final value of E is immaterial.

- Figure 7.2 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 As and Bs. 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 two numbers.
  - The add-overflow flip-flop *AVF* holds the overflow bit when A and B are added.
- The A register provides other microoperations that may be needed when we specify the sequence of steps in the algorithm.

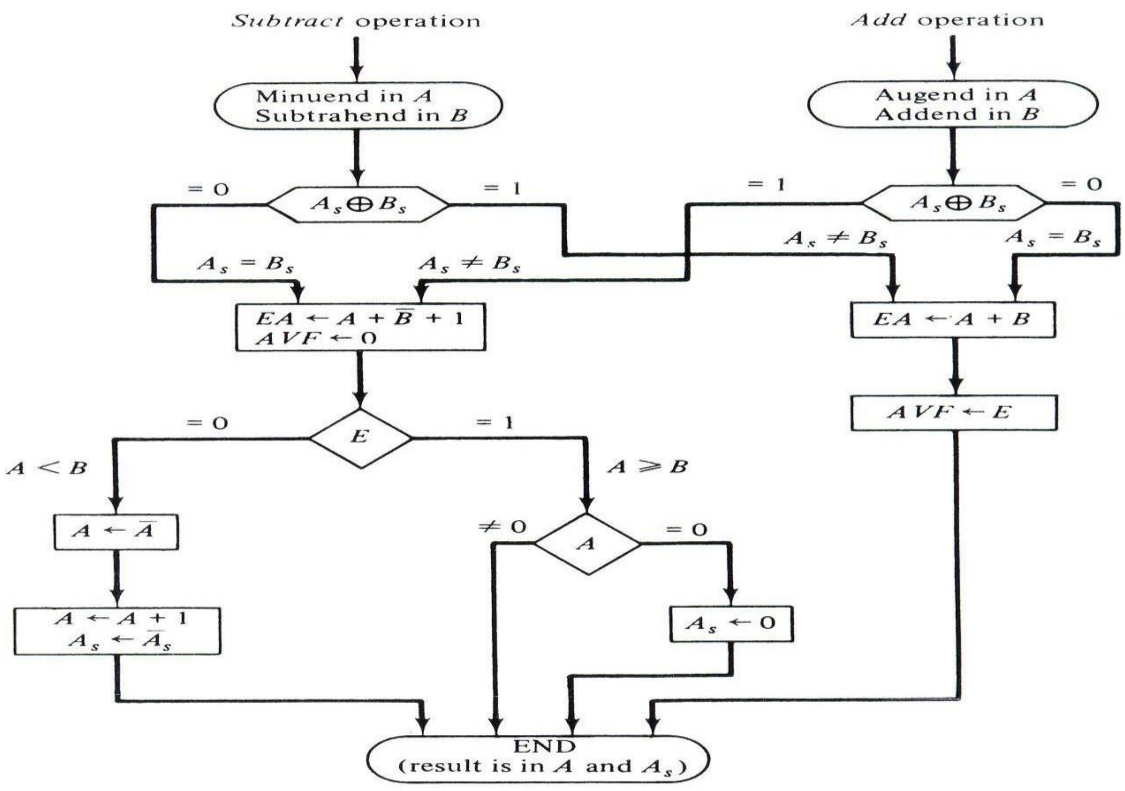
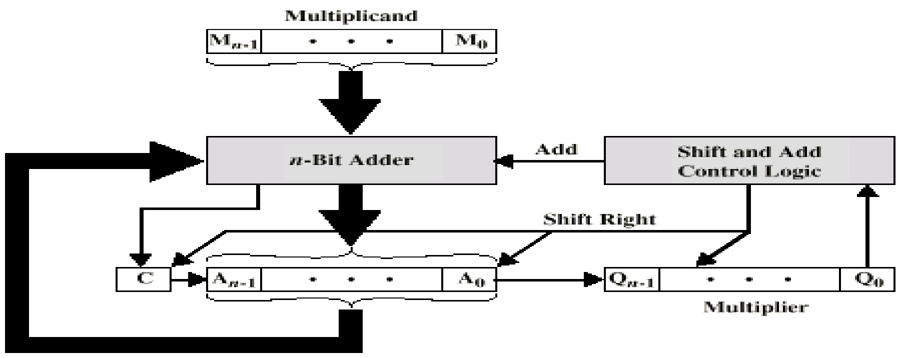


Figure 10-2 Flowchart for add and subtract operations.

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.



C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

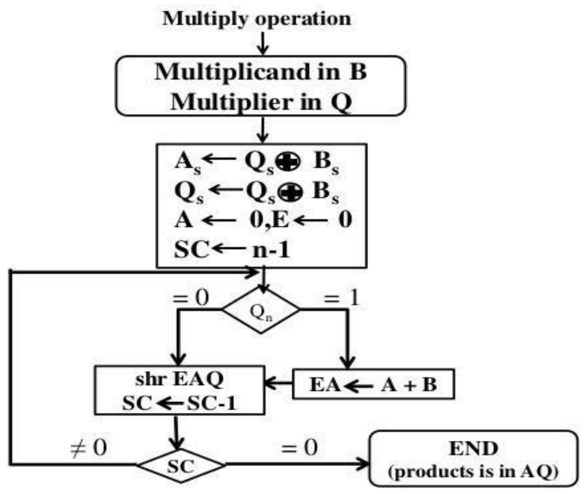


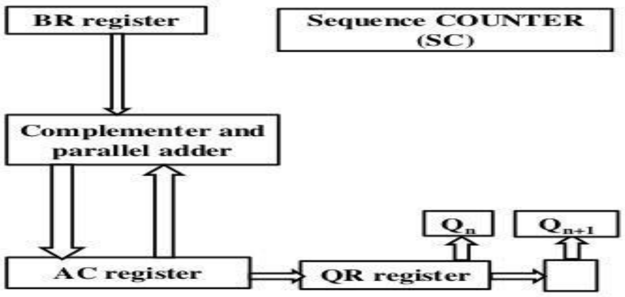
Figure: Flowchart for multiply operation.

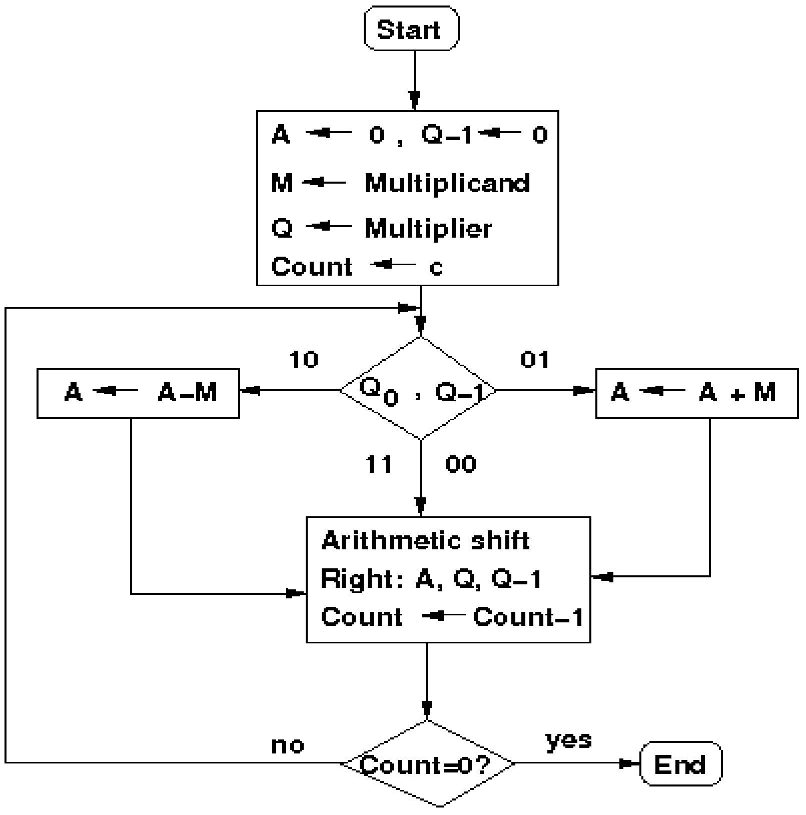
Booth's algorithm :

- Booth algorithm gives a procedure for multiplying binary integers in signed- 2's complement representation.
- It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as $2^{k+1} - 2^m$.
- For example, the binary number 001110 (+14) has a string 1's from 2^3 to 2^1 (k=3, m=1). The number can be represented as $2^{k+1} - 2^m$. $= 2^4 - 2^1 = 16 - 2 = 14$. Therefore, the multiplication M X 14, where M is the multiplicand and 14 the multiplier, can be done as M X 2^4 – M X 2^1.
- Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.

Hardware for Booth Algorithm

- Sign bits are not separated from the rest of the registers
- 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
- Flip-flop Qn+1 is appended to QR to facilitate a double bit inspection of the multiplier





- As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting of partial product.

- Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial, 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 in a string of 0’s in the multiplier.

3. The partial product does not change when multiplier bit is identical to the previous multiplier bit.

- The algorithm works for positive or negative multipliers in 2’s complement representation.

- This is because a negative multiplier ends with a string of 1’s and the last operation will be a subtraction of the appropriate weight.

- The two bits of the multiplier in Qn and Qn+1 are inspected.

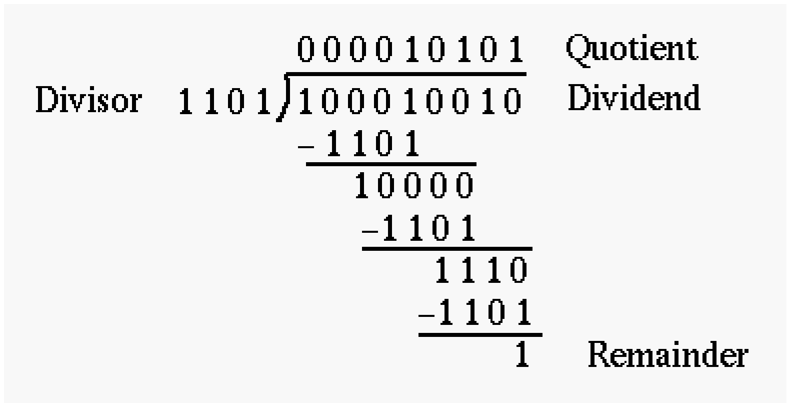
- If the two bits are equal to 10, it means that the first 1 in a string of 1 's has been encountered. This requires a subtraction of the multiplicand from the partial product in AC.

- 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.

- When the two bits are equal, the partial product does not change.

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

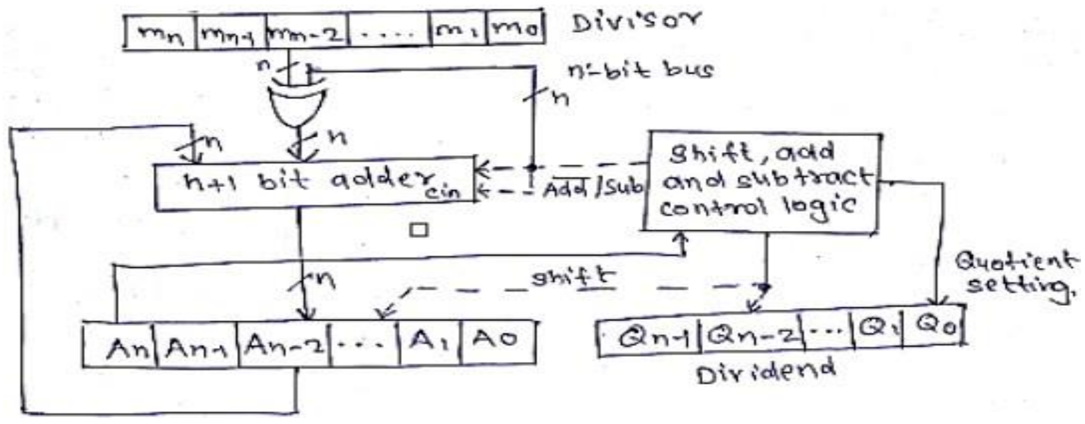


- The devisor is compared with the five most significant bits of the dividend.
- Since the 5-bit number is smaller than B, we again repeat the same process.
- Now the 6-bit number is greater than B, so we place a 1 for the quotient bit in the sixth position above the dividend.
- Now we shift the divisor 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 partialremainder.
- 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 to1.
- 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.

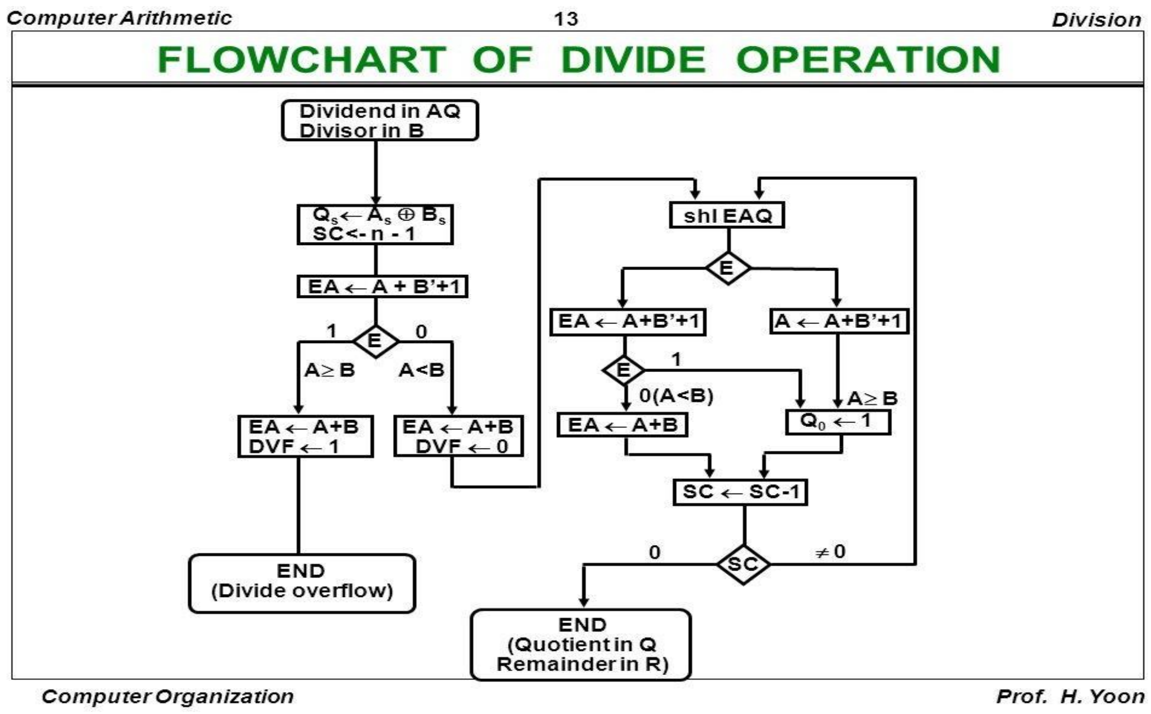
Hardware Implementation for Signed-Magnitude Data

- 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.

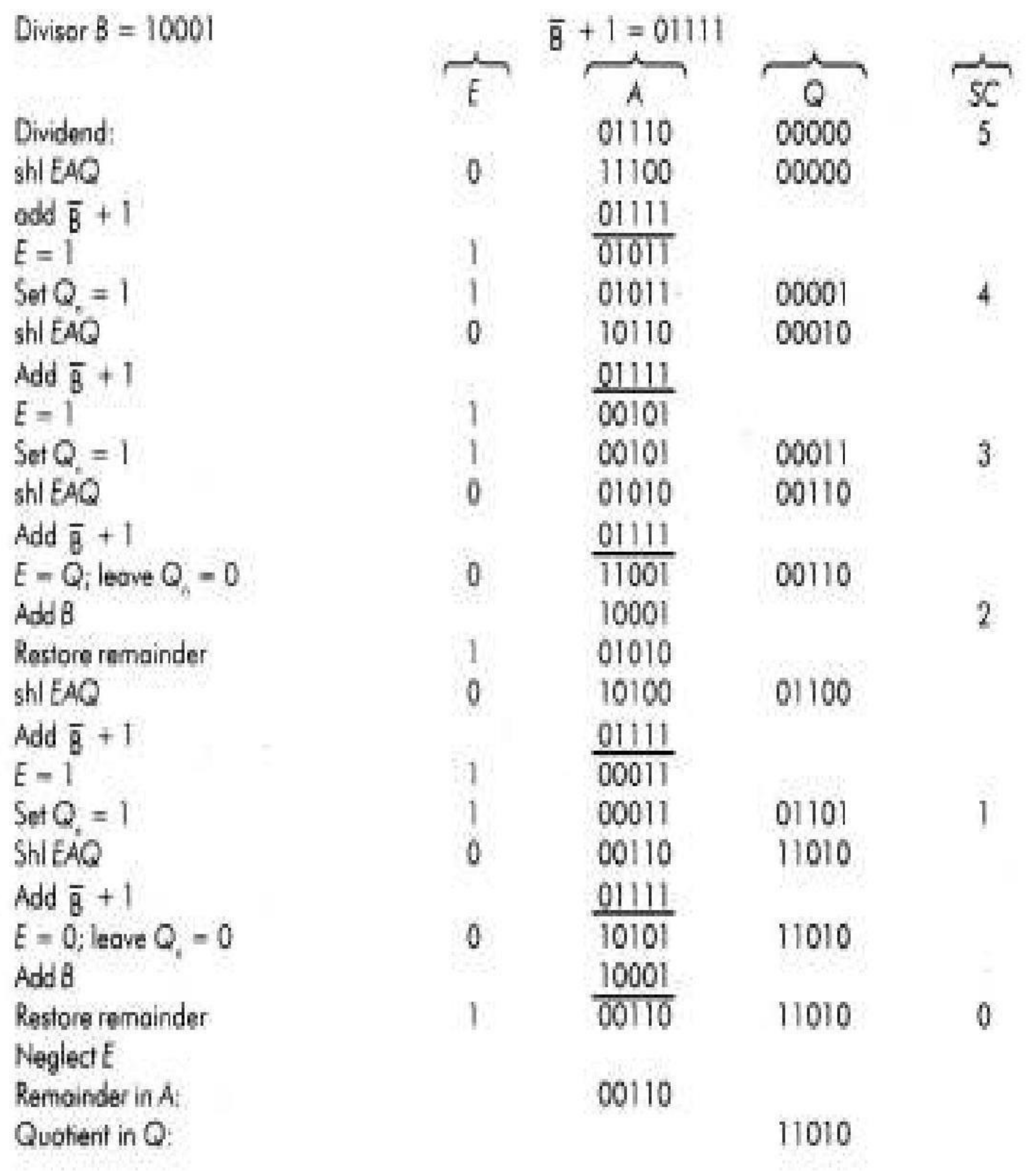
- The hardware required is identical to that of multiplication. Register EAQ is now shifted to the left with 0 inserted into Qn and the previous value of E is lost.
- The example is given in Figure 4.10 to clear the proposed division process.
- The divisor is stored in the B register and the double-length 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. E



Algorithm:



Example of Binary Division with Digital Hardware



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

	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		
$E = Q$; 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	

Floating -Point Arithmetic Operations :

- In many high-level programming languages we have a facility for specifying floating-point numbers.
- The most common way is by a real declaration statement. High level programming languages must have a provision for handling floating-point arithmetic operations.
- The operations are generally built in the internal hardware. If no hardware is available, the compiler must be designed with a package of floating-point software subroutine.
- Although the hardware method is more expensive, it is much more efficient than the software method.
- Therefore, floating- point hardware is included in most computers and is omitted only in very small ones.

Basic Considerations :

- There are two part of a floating-point number in a computer - a mantissa m and an exponent e.
- The two parts represent a number generated from multiplying m times a radix r raised to the value of e. Thus

$$m \text{ x } r^{e}$$

- The mantissa may be a fraction or an integer.
- The position of the radix point and the value of the radix r are not included in the registers.
- For example, assume a fraction representation and a radix10.
- The decimal number 537.25 is represented in a register with m = 53725 and e = 3 and is interpreted to represent the floating-point number

$$.53725 \text{ x } 10^{3}$$

- A floating-point number is said to be normalized if the most significant digit of the mantissa in nonzero.

- So the mantissa contains the maximum possible number of significant digits.

- We cannot normalize a zero because it does not have a nonzero digit.

- It is represented in floating-point by all 0's in the mantissa and exponent.

- Floating-point representation increases the range of numbers for a given register.

- Consider a computer with 48-bit words. Since one bit must be reserved for the sign, the range of fixed-point integer numbers will be + (24^{7} – 1), which is approximately + 101^{4}.

- The 48 bits can be used to represent a floating-point number with 36 bits for the mantissa and 12 bits for the exponent.

- Assuming fraction representation for the mantissa and taking the two sign bits into consideration, the range of numbers that can be represented is

$$+ (1 - 2^{-35}) \times 2^{2047}$$

- This number is derived from a fraction that contains 35 1's, an exponent of 11 bits (excluding its sign), and because $2^{11}-1 = 2047$.

- The largest number that can be accommodated is approximately 10^{615}.

- The mantissa that can accommodated is 35 bits (excluding the sign) and if considered as an integer it can store a number as large as (2^{35} –1).

- This is approximately equal to 10^{10}, which is equivalent to a decimal number of 10 digits.

- Computers with shorter word lengths use two or more words to represent a floating-point number.

- An 8-bit microcomputer uses four words to represent one floating-point number. One word of 8 bits are reserved for the exponent and the 24 bits of the other three words are used in the mantissa.
- Arithmetic operations with floating-point numbers are more complicated than with fixed-point numbers.
- Their execution also takes longer time and requires more complex hardware.
- Adding or subtracting two numbers requires first an alignment of the radix point since the exponent parts must be made equal before adding or subtracting the mantissas.
- We do this alignment by shifting one mantissa while its exponent is adjusted until it becomes equal to the other exponent.
- Consider the sum of the following floating-point numbers:

$$.5372400 \times 10^{2}$$

$$+ .1580000 \times 10^{-1}$$

- Floating-point multiplication and division need not do an alignment of the mantissas.
- Multiplying the two mantissas and adding the exponents can form the product.
- Dividing the mantissas and subtracting the exponents perform division.
- The operations done with the mantissas are the same as in fixed-point numbers, so the two can share the same registers and circuits.
- The operations performed with the exponents are compared and incremented (for aligning the mantissas), added and subtracted (for multiplication) and division), and decremented (to normalize the result).
- We can represent the exponent in any one of the three representations - signed-magnitude, signed 2's complement or signed 1's complement.
- Biased exponents have the advantage that they contain only positive numbers.
- Now it becomes simpler to compare their relative magnitude without bothering about their signs.

- Another advantage is that the smallest possible biased exponent contains all zeros.

- The floating-point representation of zero is then a zero mantissa and the smallest possible exponent.

Register Configuration

- The register configuration for floating-point operations is shown in figure 4.13. As a rule, the same registers and adder used for fixed-point arithmetic are used for processing the mantissas.
- The difference lies in the way the exponents are handled.

- The register organization for floating-point operations is shown in Fig. 4.13. Three registers are there, BR, AC, and QR. Each register is subdivided into two parts.

- The mantissa part has the same uppercase letter symbols as in fixed-point representation.

- The exponent part may use corresponding lower-case letter symbol.

FLOATING POINT ARITHMETIC OPERATIONS

$F = m \times r^e$

where m: Mantissa

r: Radix

Registers for Floating Point Arithmetic

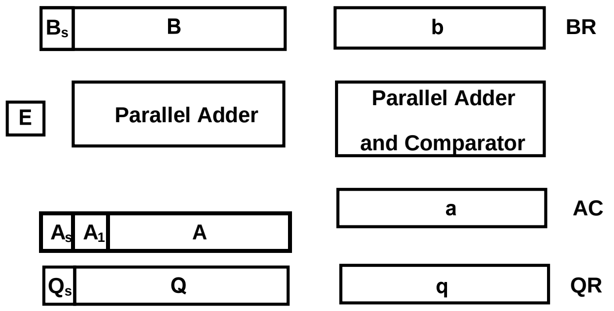


Figure 4.13: Registers for Floating Point arithmetic operations

- Assuming that each floating-point number has a mantissa in signed-magnitude representation and a biased exponent.
- Thus the AC has a mantissa whose sign is in As, and a magnitude that is in A.
- The diagram shows the most significant bit of A, labeled by A1.
- The bit in his position must be a 1 to normalize the number. Note that the symbol AC represents the entire register, that is, the concatenation of As, A and a.

- In the similar way, register BR is subdivided into Bs, B, and b and QR into Qs, Q and q.

- A parallel-adder adds the two mantissas and loads the sum into A and the carry into E.

- A separate parallel adder can be used for the exponents. The exponents do not have a district sign bit because they are biased but are represented as a biased positive quantity.

- It is assumed that the floating- point number are so large that the chance of an exponent overflow is very remote and so the exponent overflow will be neglected.
- The exponents are also connected to a magnitude comparator that provides three binary outputs to indicate their relative magnitude.
- The number in the mantissa will be taken as a fraction, so they binary point is assumed to reside to the left of the magnitude part.
- Integer representation for floating point causes certain scaling problems during multiplication and division.
- To avoid these problems, we adopt a fraction representation.
- The numbers in the registers should initially be normalized. After each arithmetic operation, the result will be normalized.
- Thus all floating-point operands are always normalize

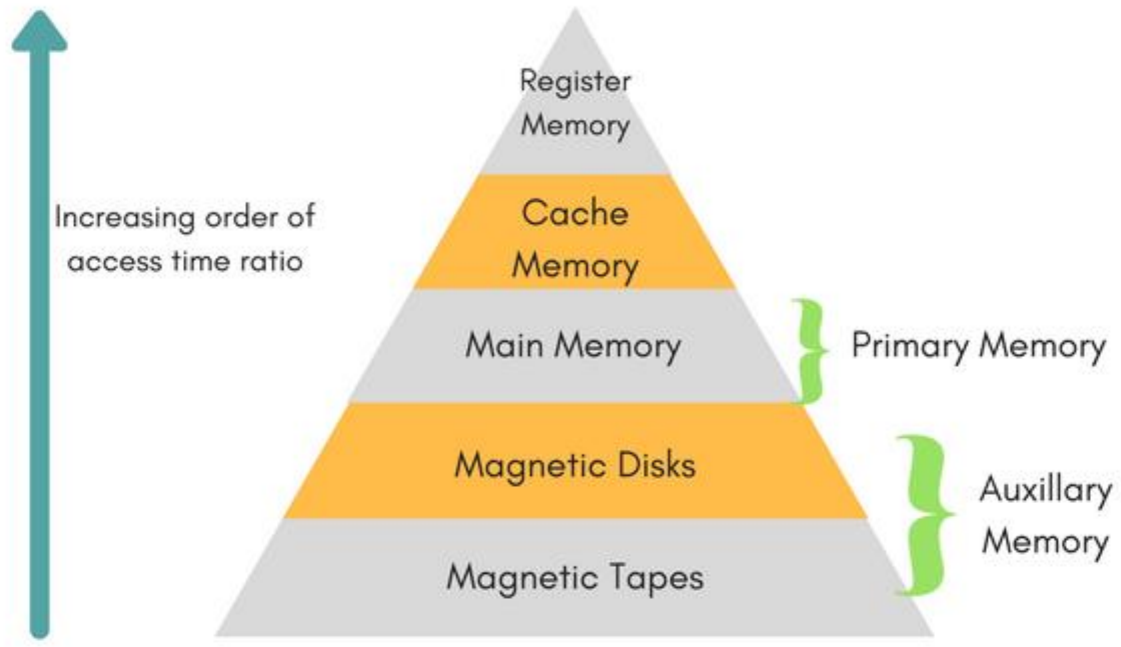
UNIT – IV

Memory Organization :

A memory unit is the collection of storage units or devices together. The memory unit stores the binary information in the form of bits. Generally, memory/storage is classified into 2 categories:

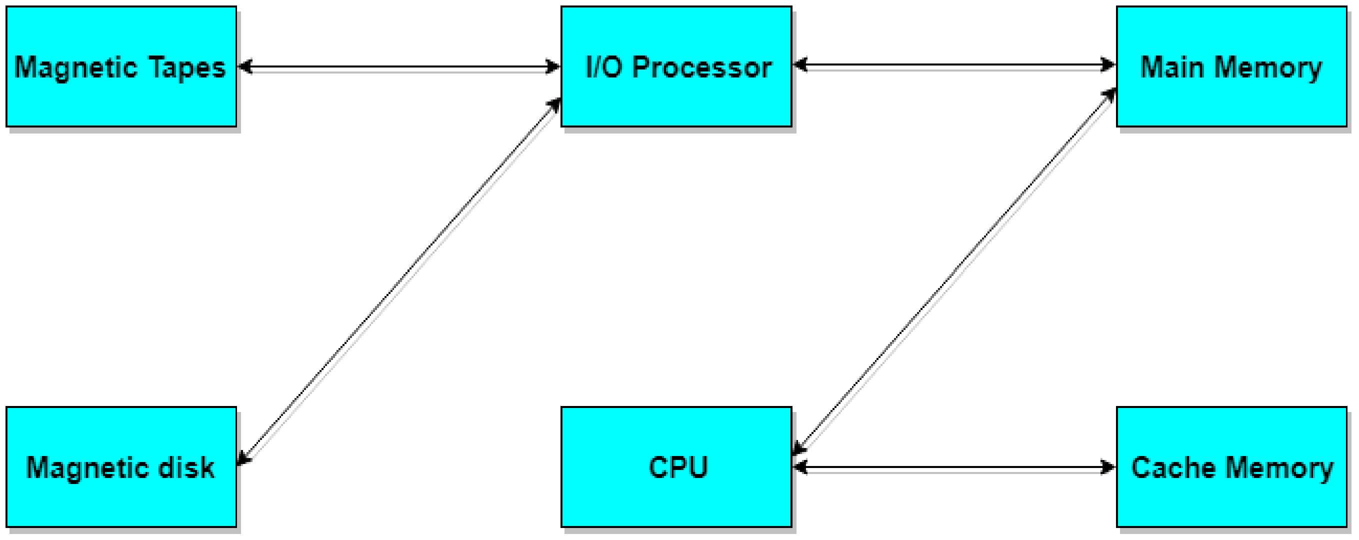
- **Volatile Memory**: This loses its data, when power is switched off.
- **Non-Volatile Memory**: This is a permanent storage and does not lose any data when power is switched off.

Memory Hierarchy :



- The total memory capacity of a computer can be visualized by hierarchy of components.
- The memory hierarchy system consists of all storage devices contained in a computer system from the slow Auxiliary Memory to fast Main Memory and to smaller Cache memory.
- **Auxillary memory** access time is generally **1000 times** that of the main memory, hence it is at the bottom of the hierarchy.
- The **main memory** occupies the central position because it is equipped to communicate directly with the CPU and with auxiliary memory devices through Input/output processor (I/O).
- When the program not residing in main memory is needed by the CPU, they are brought in from auxiliary memory.

- Programs not currently needed in main memory are transferred into auxiliary memory to provide space in main memory for other programs that are currently in use.
- The **cache memory** is used to store program data which is currently being executed in the CPU. Approximate access time ratio between cache memory and main memory is about **1 to 7~10**



Memory Access Methods

- Each memory type, is a collection of numerous memory locations.
- To access data from any memory, first it must be located and then the data is read from the memory location.
- Following are the methods to access information from memory locations:

1. **Random Access**: Main memories are random access memories, in which each memory location has a unique address. Using this unique address any memory location can be reached in the same amount of time in any order.
2. **Sequential Access**: This methods allows memory access in a sequence or in order.
3. **Direct Access**: In this mode, information is stored in tracks, with each track having a separate read/write head.

Main Memory :

- The memory unit that communicates directly within the CPU, Auxillary memory and Cache memory, is called main memory.
- It is the central storage unit of the computer system. It is a large and fast memory used to store data during computer operations.
- Main memory is made up of **RAM** and **ROM**, with RAM integrated circuit chips holing the major share.

RAM: Random Access Memory

- **DRAM**: Dynamic RAM, is made of capacitors and transistors, and must be refreshed every 10~100 ms. It is slower and cheaper than SRAM.
- **SRAM**: Static RAM, has a six transistor circuit in each cell and retains data, until powered off.
- **NVRAM**: Non-Volatile RAM, retains its data, even when turned off. Example: Flash memory.

ROM:

- Read Only Memory, is non-volatile and is more like a permanent storage for information.
- It also stores the **bootstrap loader** program, to load and start the operating system when computer is turned on.
- **PROM** (Programmable ROM), **EPROM** (Erasable PROM) and **EEPROM** (Electrically Erasable PROM) are some commonly used ROMs.

Auxiliary Memory :

Magnetic Tape:

- Magnetic tapes are used for large computers like mainframe computers where large volume of data is stored for a longer time.
- In PC also you can use tapes in the form of cassettes.
- The cost of storing data in tapes is inexpensive. Tapes consist of magnetic materials that store data permanently.
- It can be 12.5 mm to 25 mm wide plastic film-type and 500 meter to 1200 meter long which is coated with magnetic material.
- The deck is connected to the central processor and information is fed into or read from the tape through the processor.
- It’s similar to cassette tape recorder.
- Magnetic tape is an information storage medium consisting of a magnetisable coating on a thin plastic strip.

- Nearly all recording tape is of this type, whether used for video with a video cassette recorder, audio storage (reel-to-reel tape, compact audio cassette, digital audio tape (DAT), digital linear tape (DLT) and other formats including 8-track cartridges) or general purpose digital data storage using a computer (specialized tape formats, as well as the above- mentioned compact audio cassette, used with home computers of the 1980s, and DAT, used for backup in workstation installations of the 1990s).

- Magneto-optical and optical tape storage products have been developed using many of the same concepts as magnetic storage, but have achieved little commercial success.

Magnetic Disk:

- You might have seen the gramophone record, which is circular like a disk and coated with magnetic material.
- Magnetic disks used in computer are made on the same principle.
- It rotates with very high speed inside the computer drive.
- Data is stored on both the surface of the disk. Magnetic disks are most popular for direct access storage device.
- Each disk consists of a number of invisible concentric circles called tracks.
- Information is recorded on tracks of a disk surface in the form of tiny magnetic spots.
- The presence of a magnetic spot represents one bit and its absence represents zero bit.
- The information stored in a disk can be read many times without affecting the stored data.
- So the reading operation is non-destructive. But if you want to write a new data, then the existing data is erased from the disk and new data is recorded. For Example- Floppy Disk.
- The primary computer storage device. Like tape, it is magnetically recorded and can be re-recorded over and over.

- Disks are rotating platters with a mechanical arm that moves a read/write head between the outer and inner edges of the platter's surface.

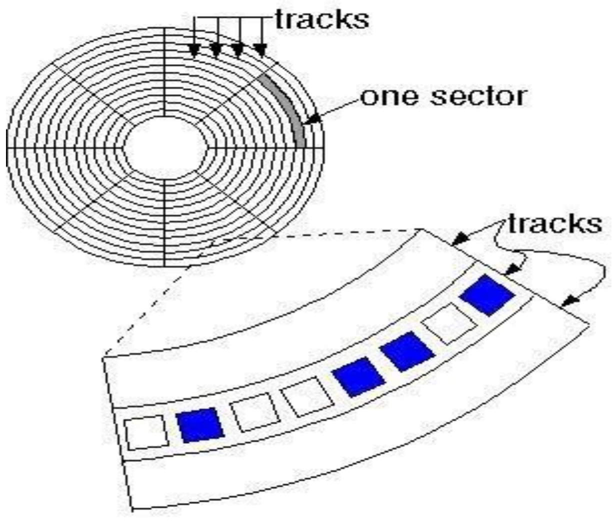
- It can take as long as one second to find a location on a floppy disk to as little as a couple of milliseconds on a fast hard disk. See hard disk for more details.

- The disk surface is divided into concentric tracks (circles within circles). The thinner the tracks, the more storage.

- The data bits are recorded as tiny magnetic spots on the tracks. The smaller the spot, the more bits per inch and the greater the storage.

Sectors :

- Tracks are further divided into sectors, which hold a block of data that is read or written at one time; for example, READ SECTOR 782, WRITE SECTOR 5448.
- In order to update the disk, one or more sectors are read into the computer, changed and written back to disk.
- The operating system figures out how to fit data into these fixed spaces.
- Modern disks have more sectors in the outer tracks than the inner ones because the outer radius of the platter is greater than the inner radius



Block diagram of Magnetic Disk

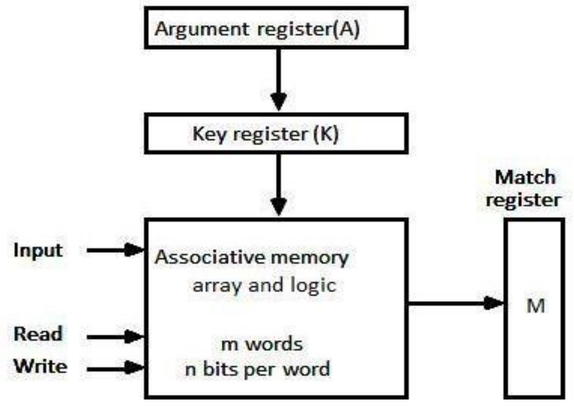
Optical Disk:

- With every new application and software there is greater demand for memory capacity.
- It is the necessity to store large volume of data that has led to the development of optical disk storage medium.
- Optical disks can be divided into the following categories:

1. **Compact Disk/ Read Only Memory (CD-ROM**
2. **Write Once, Read Many (WORM)**
3. **Erasable Optical Disk**

Associative Memory: Content Addressable Memory (CAM).

- The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address.
- A memory unit accessed by content is called an associative memory or content addressable memory (CAM).
- This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location.
- The block diagram of an associative memory is shown in figure 9.3.



- It consists of a memory array and logic form words with n bits per word.
- The argument register A and key register K each have n bits, one for
- each bit of a word. The match register M has m bits, one for each
- memory word.

Each word in memory is compared in parallel with the content of the argument register.

- The words that match the bits of the argument register set a corresponding bit in the match register.
- After the matching process, those bits in the match register that have been set indicate the fact that their corresponding words have been matched.
- Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.

Cache Memory:

- The data or contents of the main memory that are used again and again by CPU, are stored in the cache memory so that we can easily access that data in shorter time.
- Whenever the CPU needs to access memory, it first checks the cache memory.
- If the data is not found in cache memory then the CPU moves onto the main memory.
- It also transfers block of recent data into the cache and keeps on deleting the old data in cache to accomodate the new one.

Hit Ratio:

- The performance of cache memory is measured in terms of a quantity called **hit ratio**.
- When the CPU refers to memory and finds the word in cache it is said to produce a **hit**.
- If the word is not found in cache, it is in main memory then it counts as a **miss**.
- The ratio of the number of hits to the total CPU references to memory is called hit ratio.

```
Hit Ratio = Hit/(Hit + Miss)
```

Input/Output Organisation :

In this tutorial we will learn about how Input and Output is handled in a computer system.

Input/Output Subsystem :

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It handles all the input-output operations of the computer system.

Peripheral Devices :

Input or output devices that are connected to computer are called **peripheral devices**. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be the part of computer system. These devices are also called **peripherals**.

For example: *Keyboards*, *display units* and *printers* are common peripheral devices.

There are three types of peripherals:

1. **Input peripherals** : Allows user input, from the outside world to the computer. Example: Keyboard, Mouse etc.
2. **Output peripherals**: Allows information output, from the computer to the outside world. Example: Printer, Monitor etc
3. **Input-Output peripherals**: Allows both input(from outised world to computer) as well as, output(from computer to the outside world). Example: Touch screen etc.

Interfaces :

- Interface is a shared boundary btween two separate components of the computer system which can be used to attach two or more components to the system for communication purposes.
- There are two types of interface:

1. CPU Inteface
2. I/O Interface

Let's understand the I/O Interface in details,

Input-Output Interface:

- Peripherals connected to a computer need special communication links for interfacing with CPU.
- In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers.
- These components are called **input-output interface units** because they provide communication links between processor bus and peripherals.
- They provide a method for transferring information between internal system and input-output devices.

Modes of I/O Data Transfer :

Data transfer between the central unit and I/O devices can be handled in generally three types of modes which are given below:

1. Programmed I/O
2. Interrupt Initiated I/O
3. Direct Memory Access

Programmed I/O

Programmed I/O instructions are the result of I/O instructions written in computer program. Each data item transfer is initiated by the instruction in the program.

Usually the program controls data transfer to and from CPU and peripheral. Transferring data under programmed I/O requires constant monitoring of the peripherals by the CPU.

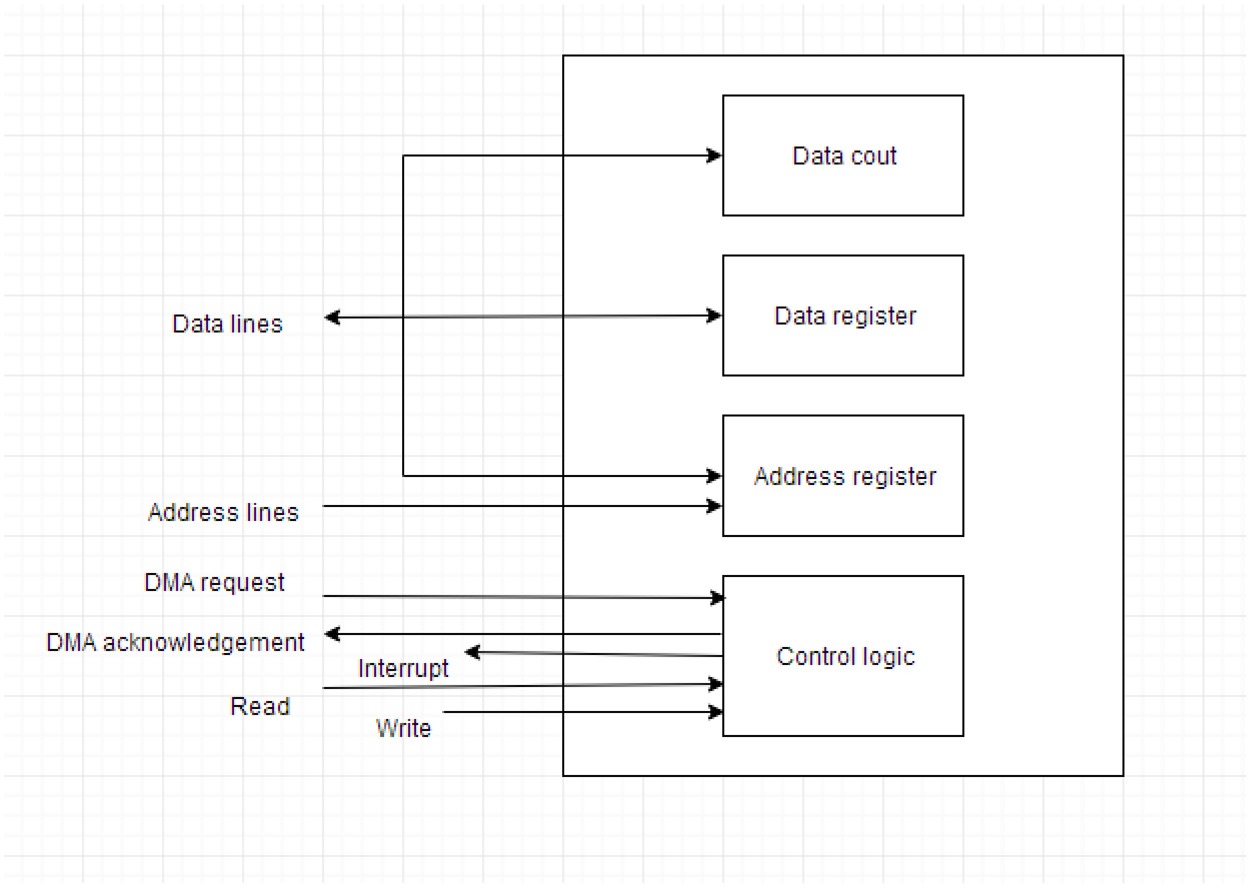
Interrupt Initiated I/O

- In the programmed I/O method the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer.
- This is time consuming process because it keeps the processor busy needlessly.
- This problem can be overcome by using **interrupt initiated I/O**. In this when the interface determines that the peripheral is ready for data transfer, it generates an interrupt.
- After receiving the interrupt signal, the CPU stops the task which it is processing and service the I/O transfer and then returns back to its previous processing task.

Direct Memory Access

- Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer.
- This technique is known as **DMA**.
- In this, the interface transfer data to and from the memory through memory bus.
- A DMA controller manages to transfer data between peripherals and memory unit.
- Many hardware systems use DMA such as disk drive controllers, graphic cards, network cards and sound cards etc.
- It is also used for intra chip data transfer in multicore processors.

- In DMA, CPU would initiate the transfer, do other operations while the transfer is in progress and receive an interrupt from the DMA controller when the transfer has been completed.



Above figure shows block diagram of DMA

Interrupts

- Data transfer between the CPU and the peripherals is initiated by the CPU.
- But the CPU cannot start the transfer unless the peripheral is ready to communicate with the CPU.
- When a device is ready to communicate with the CPU, it generates an interrupt signal.
- A number of input-output devices are attached to the computer and each device is able to generate an interrupt request.
- The main job of the interrupt system is to identify the source of the interrupt.
- There is also a possibility that several devices will request simultaneously for CPU communication. Then, the interrupt system has to decide which device is to be serviced first.

Priority Interrupt :

- A priority interrupt is a system which decides the priority at which various devices, which generates the interrupt signal at the same time, will be serviced by the CPU.
- The system has authority to decide which conditions are allowed to interrupt the CPU, while some other interrupt is being serviced.
- Generally, devices with high speed transfer such as *magnetic disks* are given high priority and slow devices such as *keyboards* are given low priority.
- When two or more devices interrupt the computer simultaneously, the computer services the device with the higher priority first.

Types of Interrupts:

Following are some different types of interrupts:

1.Hardware Interrupts

2.Software Interrupts

Hardware Interrupts :

- When the signal for the processor is from an external device or hardware then this interrupts is known as **hardware interrupt**.

Let us consider an example: when we press any key on our keyboard to do some action, then this pressing of the key will generate an interrupt signal for the processor to perform certain action. Such an interrupt can be of two types:

- **Maskable Interrupt**

  The hardware interrupts which can be delayed when a much high priority interrupt has occurred at the same time.

- **Non Maskable Interrupt**

  The hardware interrupts which cannot be delayed and should be processed by the processor immediately.

Software Interrupts:

The interrupt that is caused by any internal system of the computer system is known as a **software interrupt**. It can also be of two types:

- **Normal Interrupt**

  The interrupts that are caused by software instructions are called **normal software interrupts**.

- **Exception**

  Unplanned interrupts which are produced during the execution of some program are called **exceptions**, such as division by zero.