Number Systems

A number system of base, or radix r, is a system that uses distinct symbols for r digits. Numbers are represented by a string of digit symbols. To determine the quantity that the number represents, it is necessary to multiply each digit by an integer power of r and then form the sum of all weighted digits.

For example, the decimal number system in everyday use employs the radix 10 system. The 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The string of digits 724.5 is interpreted to represent the quantity

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

ie., 7 hundreds plus 2 tens plus 4 units plus 5 tenths.

Binary number system uses the radix 2. The two digit symbols used are 0 and 1. The string of digits 101101 is interpreted to represent the quantity

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

To distinguish between different radix numbers, the ~~num~~ digits will be enclosed in parantheses and the radix of the number inserted as a subscript. For example, to show the equality between decimal and binary forty five will be written as $(101101)_2 = (45)_{10}$.

Octal and Hexadecimal number system consists of radix 8 and 16 respectively. The eight symbols of the Octal system are 0, 1, 2, 3, 4, 5, 6, and 7. The 16 ~~num~~ symbols of the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. The symbols A, B, C, D, E, F correspond to the decimal numbers 10, 11, 12, 13, 14, 15 respectively.

Conversion

The conversion of a decimal integer into a base r representation is done by successive divisions by r and accumulation of the remainders. The conversion of a decimal fraction to radix r representation is accomplished by successive multiplications by r and accumulation of the integer digits so obtained.

Decimal to Binary

2 ways of converting from decimal to binary.

① Sum-of-Weights Method

Determines the set of binary weight values whose sum is equal to the decimal number

eg. $9 = 8 + 1 = 2^3 + 2^0$.

By placing a 1 in the appropriate weight positions, 2^3 and 2^0 and a 0 in the other positions to get the binary number.

$2^3 \quad 2^2 \quad 2^1 \quad 2^0$

$(1 \quad 0 \quad 0 \quad 1)_2$.

$12_{10} = 8 + 4 = 2^3 + 2^2 \rightarrow 1100_2$

$25_{10} = 16 + 8 + 1 = 2^4 + 2^3 + 2^0 \rightarrow 11001_2$

$58_{10} = 32 + 16 + 8 + 2 = 2^5 + 2^4 + 2^3 + 2^1 \rightarrow 111010_2$

$82_{10} = 64 + 16 + 2 = 2^6 + 2^4 + 2^1 \rightarrow 1010010_2$

② Repeated Division-by-2 Method

The number is divided by 2 and then dividing each resulting quotient by 2 until there is a 0 quotient. The remainders generated by each ~~res~~ division form the binary number. The first remainder to be produced is the least significant bit (LSB) in the binary number.

12		19		45	
2 | 12		2 | 19		2 | 45	
2 | 6	0 (LSB)	2 | 9	1	2 | 22	1
2 | 3	0	2 | 4	1	2 | 11	0
1	1 (MSB)	2 | 2	0	2 | 5	1
		1	0	2 | 2	1
				1	0
$(1100)_2$		$(10011)_2$		$(101101)_2$	

Converting Decimal Fractions to Binary

$$0.625_{10} = 0.5 + 0.125 = 2^{-1} + 2^{-3} = 0.101_2$$

Another method is repeated multiplication-by-2. Multiplying fractional number by 2 and then multiplying each resulting fractional part of the product by 2 until the fractional product is zero. The carries generated by each multiplication form the binary number.

Carry ↓ Fractional part

$0.3125 \times 2 = 0.625 \rightarrow 0$ (MSB)

$0.625 \times 2 = 1.25 \rightarrow 1$

$0.25 \times 2 = 0.50 \rightarrow 0$

$0.50 \times 2 = 1.00 \rightarrow 1$ (LSB)

0.0101_2

Qn. Convert the following decimal number to binary using sum of weights method and repeated division-by-2 method. (repeated multiplication-by-2 for fractions)
(a) 23 (b) 57 (c) 45.5 (d) 14 (e) 21 (f) 0.375

Octal to Decimal

$$2374_8 = 2\times8^3 + 3\times8^2 + 7\times8^1 + 4\times8^0$$
$$= 2\times512 + 3\times64 + 7\times8 + 4\times1$$
$$= 1024 + 192 + 56 + 4 = (1276)_{10}$$

Decimal to Octal

$$\begin{array}{r|l} 8 & 359 \\ \hline 8 & 44 - 7 \text{ (LSD)} \\ \hline & 5 - 4 \end{array}$$
(MSD)

$(547)_8$

$$(0.325)_8 = 3\times0.125 + 2\times0.015625 + 5\times0.001953$$
$$= 0.375 + 0.03125 + 0.009765$$
$$= (0.416015)_{10}$$

Octal to Binary

Each octal digit is represented by 3 bits as

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

To convert an octal number to a binary number, simply replace each octal digit by the appropriate three bits.

$13_8 \rightarrow (001\,011)_2$ $\quad$ $25_8 \rightarrow (010\ 101)_2$ $\quad$ 47_8 $\quad$ 170_8

73_8

752_8 $\quad$ 5276_8 $\quad$ $37.12_8 \rightarrow (011\ 111.001010)_2$

125_8 $\quad$ 46_8 $\quad$ 723_8 $\quad$ 5624.37_8

Binary to Octal

Break the binary number into groups of three bits and convert each group into the appropriate octal digit.

$\underbrace{110}\ \underbrace{101} = (6\ 5)_8$ $\quad$ $\underbrace{101}\underbrace{111}\underbrace{001} = 5\ 7\ 1_8$ $\quad$ $\underbrace{001}\underbrace{011}\underbrace{100}\underbrace{110} = (1\ 3\ 4\ 6)_8$

Qn. Convert the following:

Octal to decimal - (a) 73_8 (b) 125_8

Decimal to Octal - (a) 98_{10} (b) 163_{10}

Binary to Octal - (a) 001001101101100_2

(b) 110101111_2 (c) 100110001.101111_2

(d) 10111111.0011_2

Hexadecimal

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Binary to Hexadecimal

Break the binary number into four bit groups starting at the binary point and replace each group with the equivalent hexadecimal symbol.

$\underbrace{1100}\underbrace{1010}\underbrace{0101}\underbrace{0111}$
$(C\quad A\quad 5\quad 7)_{16}$

$(\underbrace{0011}\ \underbrace{1111}\ \underbrace{0001}\ \underbrace{0110}\ \underbrace{1001})_2$
$(3\quad F\quad 1\quad 6\quad 9)_{16}$

Hexadecimal to Binary

Replace each hexadecimal symbol with the appropriate four bit binary code.

$10A4$
$(0001\ 0000\ 1010\ 0100)_2$

$(CF83)_{16}$
$(1100\ 1111\ 1000\ 0011)_2$

Qn Convert the following

Binary to hexadecimal - $(0011\ 1001\ 1000.1110)_2$

Hex to Binary - $(9742)_{16}$ $(D2E.8)_{16}$

Hexadecimal to Decimal.

First way is to convert the hexadecimal number to binary and then convert from binary to decimal.

$1C$
$(0001\ 1100)_2$

$00011100_2 = 2^4 + 2^3 + 2^2$
$= 16 + 8 + 4 = 28_{10}$

$(A85)_{16}$
$(1010\ 1000\ 0101)_2$

$101010000101_2 = 2^{11} + 2^9 + 2^7 + 2^2 + 2^0$
$= 2048 + 512 + 128 + 4 + 1$
$= (2693)_{10}$

Another way to convert ~~a~~ is by multiplying each hexadecimal digit by its weight and then taking the sum of these products.

$(E5)_{16} = E \times 16 + 5 \times 1 = 14 \times 16 + 5 \times 1 = 224 + 5 = 229_{10}$

$(B2F8)_{16} = B \times 4096 + 2 \times 256 + F \times 16 + 8 \times 1$
$= 11 \times 4096 + 2 \times 256 + 15 \times 16 + 8 \times 1 = (45816)_{10}$

Decimal to Hexadecimal

16 | 650
16 | 40 — A
      2 — 8

$(650)_{10} = 28A_{16}$

Representation of Negative 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. As an example, consider the signed number 14 stored in an 8 bit register. +14 is represented by a sign bit of 0 in the left most position followed by the binary equivalent of 14 : 00001110. Each of the eight bits of the register must have a value and therefore 0's must be inserted in the most significant positions following the sign bit. Although there is only one way to represent +14, there are 3 different ways to represent −14 with eight bits.

In Signed-magnitude representation - 1 0001110
In Signed-1's Complement representation - 1 1110001
In 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 the positive number including its sign bit.

Representation of BCD numbers

Binary Coded Decimal (BCD) means that each decimal digit is represented by a binary code of four bits. To express any decimal number in BCD, simply replace each decimal digit by the appropriate four bit code.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

The six four bit combinations that are not used are 1010, 1011, 1100, 1110 and 1111

3 → 0011

9.2 → 1001.0010

34.8 → 0011 0100.1000

1000 0110 → 8 6

Qn. Convert the following

Decimal to BCD - (a) 18 (b) 65 (c) 92.

(d) 150 (e) 321 (f) 1472

BCD to Decimal - (a) 00110001 (b) 01010011

(c) 100101110100 (d) 000110000110000.0111

Character Representation

Many applications of digital computers require the handling of data that consist not only of numbers, but also of the letters of the alphabet and certain special characters. Such a set contains between 32 and 64 elements (if only uppercase letters are included) or between 64 and 128 (if both uppercase and lowercase letters are included). In the first case, the binary code will require six bits and in the second case, seven bits.

ASCII.

Americal Standard Code for Information Interchange (ASCII) is the most widely used code to represent characters. It is a seven bit code in which the decimal the decimal digits are represented by the BCD code preceded by 011.

Character	ASCII Code
A	100 0001
B	100 0010
C	100 0011
⋮	
P	101 0000
Q	101 0001
	⋮

Character	ASCII Code
0	011 0000
1	011 0001
2	011 0010
	⋮
9	011 1001
Space	010 0000
.	010 1110
(	010 1000
+	010 1011

EBCDIC (Extended Binary Coded Decimal Interchange Code

This is an eight bit code in which the decimal digits are represented by the BCD code preceded by 1111. Both lowercase and uppercase letters are represented in addition to numerous other symbols and commands. For example, uppercase A is 11000001, and lowercase a is 10000001.

ASCII and EBCDIC Codes are commonly used in data transfer and computer interface applications.

Tutorial - I

Representation of floating point numbers

Two parts to represent a floating point number.

(1) Mantissa – a signed, fixed point number

(2) Exponent – designates the position of the decimal (or binary) point

The fixed point mantissa may be a fraction or an integer. For eg. +6132.789 is represented as

Fraction	Exponent
+0.6132789	+04

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

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

Fraction	Exponent
01001110	000100

The fraction has a 0 in the leftmost position to denote positive.

A floating point number is said to be normalized if the most significant digit of the mantissa is nonzero. For eg. 350 is normalized but 00035 is not. 00011010 is not normalized and is normalized by shifting three positions to the left and discarding the leading 0's to obtain 11010000

Binary Arithmetic

Since digital systems do not process the decimal numbers, and they only process binary numbers, it is necessary to learn the binary arithmetic.

Addition.

A (Augend)	B (Addend)	S (Sum)	C (carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

① $(1010)_2 + (0101)_2$

$$\begin{array}{r} 1010 \\ 0101 \\ \hline 1111 \end{array} \qquad \begin{array}{l} = \text{decimal } 10 \\ = \text{decimal } 5 \\ \hline \text{decimal } 15 \end{array}$$

② Add $(10)_{10} + (11)_{10}$ in binary

$$\begin{array}{r} 1010\ + \\ 1011 \\ \hline (10101)_2 \end{array} = (21)_{10}$$

③ Add 26 and 13 in binary

$$\begin{array}{rl} 26 & = (11010)_2 \\ 13 & = (01101)_2 \\ \hline & (100111)_2 = (39)_{10} \end{array}$$

④ Add 1011.011 and 110.1

1011.011	11.375
110.1	6.5
10001.111	17.875

⑤ Add 101.11, 1101.01 and 10000.001

101.11	5.75
1101.01	13.25
10000.001	16.125
100011.001	35.125

Subtraction

Binary Subtraction can be carried out in either one of two different ways.

① Direct Subtraction

② Complement Subtraction.

Direct Subtraction

A (Minuend)	B (Subtrahend)	D (Difference)	B (Borrow)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

① Subtract 10101 from 11011

11011	27
− 10101	21
00110	6

② 10000 − 16
− 1010 − 10
00110 6

③ 10110 − 22
1010 − 10
01100 − 12

④ 1001.10 − 9.50
− 0111.01 − 7.25
0010.01 2.25

⑤ 1010.11 10.75 −
0110.01 6.25
0100.10 4.50

One's Complement Subtraction method

Case.(i) – Subtraction of Smaller number from larger number

Procedure

① Determine 1's complement of the smaller number (subtrahend)
② Add 1's complement to the larger number (minuend) and the result is a positive number.
③ Carry ~~will be~~ generated after the addition ~~which~~ is called "End-Around Carry" (EAC). Remove the EAC and add it to the result.

① Subtract 101011 from 111101 using 1's complement method.

111101
1's Compl. 010100
EAC (1)010001 +
1
010010 = $(18)_{10}$

Add EAC

② 28 − 8.

28 = 11100
8 = 01000

1's complement of 8 = 10111

11100
10111
(1)10011 +
1
10100 = $(20)_{10}$

③ 30 − 25

11110 11001

Minuend − 11110

1's Compl of Subtrahend − 00110

①00100 +

1 Add EAC

00101

④ 25.5 = $(11001.1)_2$

$(12.25)_{10}$ = $(01100.011)_2$

1's Compl of 12.25 = 10011.100

11001.10 +

10011.10

①01101.00 +

1 Add EAC

01101.01 = $(13.25)_{10}$.

⑤ 10.625 = 1010.101

8.75 = 1000.110

1's Compl of 8.75 = 0111.001

1010.101

0111.001

①0001.110 +

1

0001.111 = $(1.875)_2$.

Case (ii) − Subtraction of larger number from smaller number

Procedure

① Determine the 1's complement of the larger number (Subtrahend)

② Add 1's complement to the smaller number (minuend)

③ After addition, no carry will be generated but answer is in 1's complement form (negative number).

To get the answer in true form, take the 1's complement of it and assign negative sign to the answer.

① 43 − 57

Minuend	43	−	101011
1's complement of Subtrahend	57	−	000110
			110001

The result has no carry, so the answer is in 1's complement form.

1's complement of 110001 is $-(001110)$.

Answer in true form $= (-14)_{10}$.

② 8 − 10.

8 − 1000

10 − 1010

Minuend	−	1000
1's complement of subtrahend	−	0101
		1101

No carry, so the answer is in 1's complement form.

∴ True answer $= -(0010)_2$.

$= (-2)_{10}$.

③ $(8.75)_{10} = (1000.110)_2$

$(10.625)_{10} = (1010.101)_2$.

Minuend	−	1000.110
1's complement of Subtrahend	−	0101.010
		1110.000

④ $\frac{5}{8} - \frac{7}{8}$

$\frac{5}{8} = 0.101$

$\frac{7}{8} = 0.111$

1's Comp = 0.000

Minuend	−	0.101
1's complement of Subtrahend	−	0.000
		0.101

Answer is in 1's complement form.

∴ True answer is $-(0.010)_2 = (-2/8)_{10}$

⑤ $16.875 - 11.125$

$16.875 = 10000.111$

$11.125 = 01011.001$

Minuend	~~10000.00~~ 10000.111 +
1's Complement of Subtrahend	10100.110
	1 00101.101

Advantages

① Since 1's Complement Subtraction can be accomplished with a binary adder, this method is useful in arithmetic logic circuits.

② It is very easy to find the 1's Complement of a number.

Disadvantages

① Hardware implementation is difficult and it gives the concept of negative zero.

2's Complement Method

The 2's complement of any binary number is determined by adding 1 to 1's complement of that number. 2's complement form is used to represent negative numbers. 2's complement of 1 is 1 and zero is 10.

Case (i) : Subtraction of a smaller number from larger number

Procedure

① Determine the 2's complement of the small number (Subtrahend)

② Add 2's complement to the minuend.

③ Discard the carry generated.

① $(111001)_2 - (101011)_2$

Minuend	-	111001
2's complement of Subtrahend	-	010101
		① 001110

$= 57$
43
14

② $(100.5)_{10} - (50.75)_{10}$

Minuend	-	1100100.10
2's complement of Subtrahend	-	1001101.01
		① 0110001.11

$= (49.75)_{10}$

③ $(1111)_2 - (1010)_2$. ④ $(112)_{10} - (65)_{10}$

⑤ 22 - 7 ~~⑥ (100.5)~~

Case (ii) : Subtraction of a larger number from Smaller number

Procedure

① Determine the 2's complement of the subtrahend.

② Add the 2's complement to minuend.

③ Answer is in 2's complement form. Take the 2's complement and assign negative sign to the answer.

No carry will be generated.

① 7 - 22

00111 - 10110

Minuend	-	00111
2's compl of Subtrahend	-	01010
		10001

Take 2's complement of 1001 = 1111

$= (-15)_{10}$

(2) 16.5 − 24.75

10000.10 − 11000.11

Minuend	= 10000.10
2's complement of Subtrahend	= 00111.01
	10111.11

2's complement of result $= -(01000.01)_2 = -(8.25)_{10}$.

Binary Multiplication

Multiplicand A	Multiplier B	Product P
0	0	0
0	1	0
1	0	0
1	1	1

(1) 7 × 5

111 × 101

```
      111  x
      101
     ----
      111
     000
    111
   ------
   100011   = 35.
```

(2) 4.75 × 3.625

100.110 × 11.101

```
       100.110  x
        11.101
      --------
       100110
      000000
     100110
    100110
   100110
  -------------
  10001.001110
```

$=(17.21875)_{10}$.

(3) 22 × 6   (4) ~~22~~ 27 × 21   (5) $\frac{3}{8} \times \frac{1}{4}$

Binary Division

$0 \div 1 = 0$

$1 \div 1 = 1$.

① $50 \div 5$

$110010 \div 101$

$= (1010)_2 = (10)_{10}$.

```
        1010
101 ) 110010
      101
      0101
       101
         0
```

②

```
       110.01
100 ) 11001
      100
      100
      100
       0100
        100
          0.
```

$25 \div 4 = (6.25)_{10}$.

16 8 4 2 1
11001

```
   6
4 )25
   24
```

③

```
           10
101.01 ) 1010.10
         101
         00.
```

④

```
      110.00
10 ) 1100.00
     10
     10
     10
      00
      00
       00
       00
```

Floating Point Arithmetic Operations

A floating point number in Computer registers consists of two parts:

① a mantissa m

② an exponent e

The two parts represent a number obtained from multiplying m times a radix r raised to the value of e. thus

$$m \times r^e.$$

The mantissa may be a fraction or an integer.
eg. 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

$$.53725 \times 10^3.$$

A floating point number is normalized if the most significant digit of the mantissa is nonzero.
A zero cannot be normalized because it does not have a nonzero digit. It is represented in floating point by all 0's in the mantissa and exponent.
Adding or Subtracting two floating point numbers requires first an alignment of the radix point since the exponent parts must be made equal before adding or subtracting the mantissas.

$$.5372400 \times 10^2 \quad +$$

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

- Shift the first number three positions to the left, Or shift the second number three positions to the right.
- When the mantissas are stored in registers, shifting to the left causes a loss of most significant digits and shifting to the right causes a loss of least significant digits. So second one is preferable.

$$.5372400 \times 10^2 \quad +$$

$$.0001580 \times 10^2$$

$$\overline{.5373980 \times 10^2.}$$

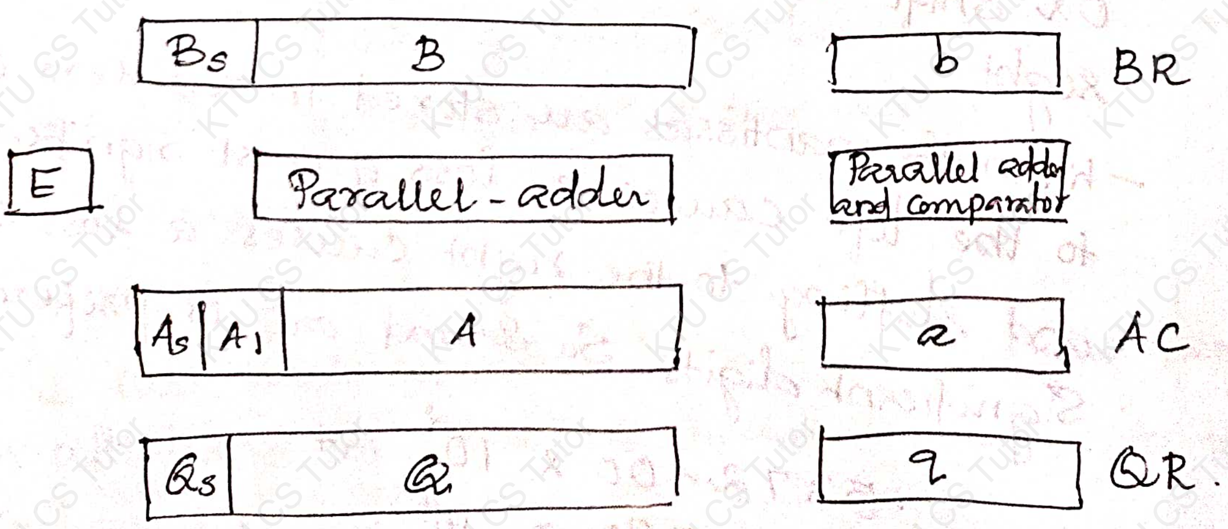
- When two normalized mantissas are added, the sum may contain an overflow digit.
- Overflow can be corrected by shifting the sum once to the right and incrementing the exponent.

$$
\begin{array}{r}
.56780 \times 10^5 \\
-\;.56430 \times 10^5 \\
\hline
.00350 \times 10^5
\end{array}
$$

To normalize, shift the mantissa to the left and decrement the exponent until a nonzero digit appears in the first position. $(.3500 \times 10^3)$

- For multiplication and division, alignment of mantissas are not required.
- Product can be formed by multiplying the two mantissas and adding the exponents.
- Division is accomplished by dividing the mantissas and subtracting the exponents.

Register Configeration



- 3 registers BR, AC and QR.
each register is subdivided into two parts.

① Mantissa part - upper case letter symbol

② exponent part - lower case letter symbol.

A_s, B_s, Q_s represents the sign bit.
A_1 is the most significant bit of A. The bit in this position must be a 1 for the number to be normalized.
Parallel adder adds the two mantissas and transfers the sum into A and the carry to E. Separate parallel adder is used for the exponents.
The numbers in the registers are assumed to be initially normalized. After each arithmetic operation, the result will be normalized.

Addition and Subtraction

- the two floating point operands are in AC and BR.
- Sum or difference formed in the AC.
- algorithm can be divided into 4 consecutive parts.

① Check for zeros

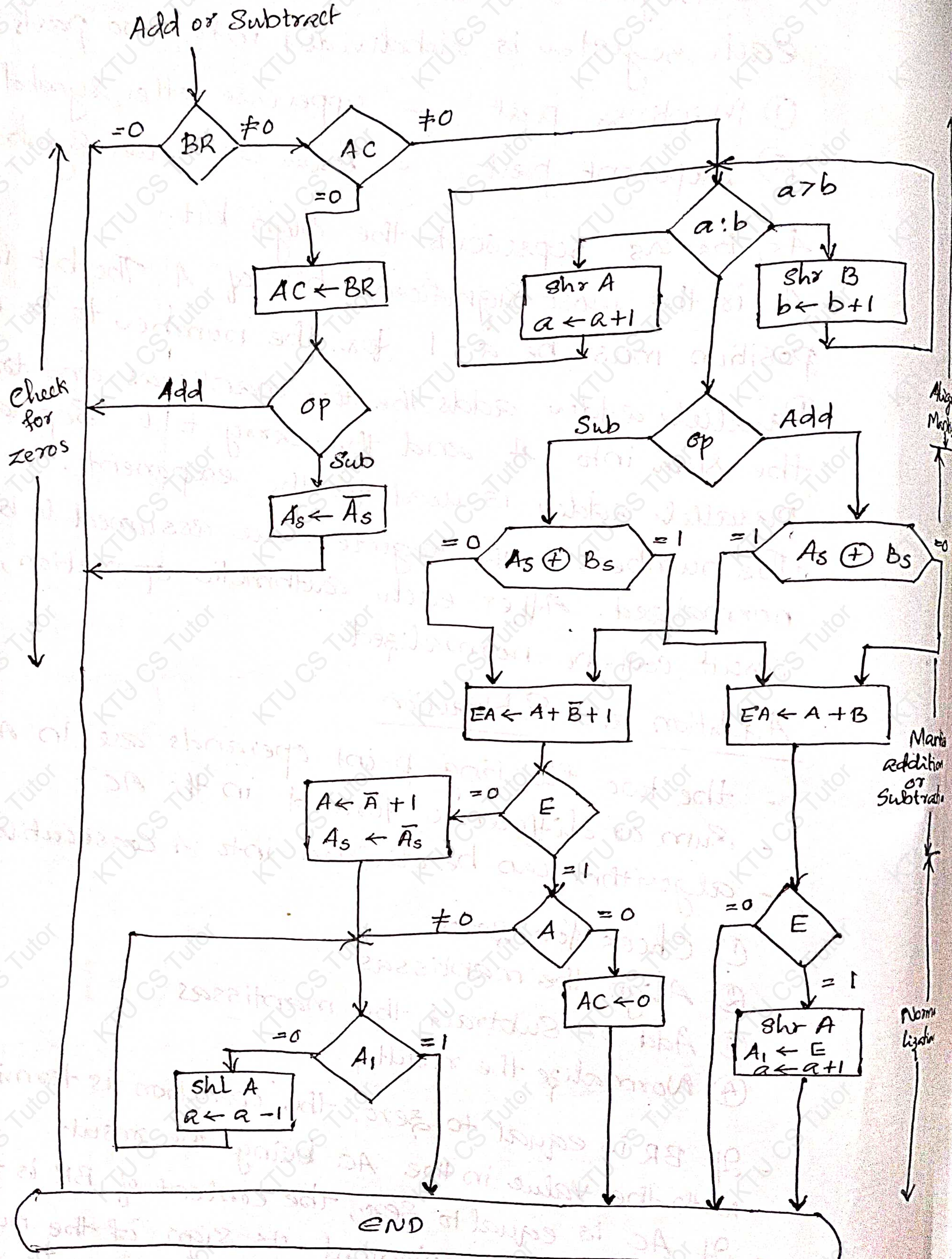
② Align the mantissas

③ Add or Subtract the mantissas

④ Normalize the result.

- If BR is equal to zero, the operation is terminated with the value in the AC being the result.
- If AC is equal to zero, the content of BR is transferred into AC and complement its sign if the numbers are to be subtracted.

If neither number is equal to zero, proceed to align the mantissas.



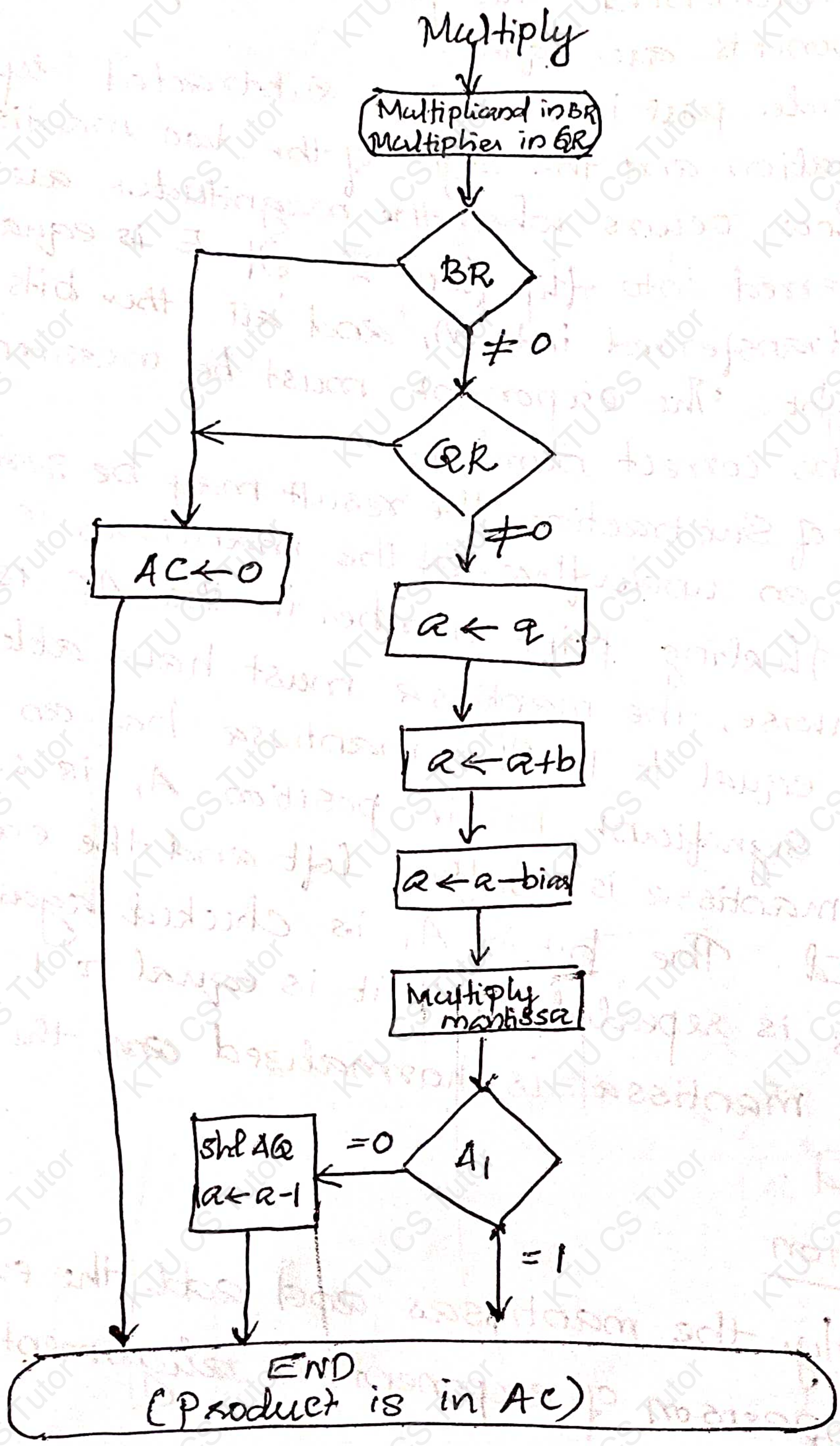
Add or Subtract
BR
=0
≠0
AC
≠0
=0
AC ← BR
op
Add
Sub
$A_s \leftarrow \overline{A_s}$
Check for zeros
a:b
a>b
shr A
a ← a+1
shr B
b ← b+1
Align Mantissa
op
Sub
Add
=0
$A_s \oplus B_s$
=1
=1
$A_s \oplus B_s$
=0
$EA \leftarrow A+\overline{B}+1$
EA ← A+B
Mantissa addition or Subtraction
E
=0
$A \leftarrow \overline{A}+1$
$A_s \leftarrow \overline{A_s}$
=1
A
≠0
=0
AC ← 0
E
=0
=1
shr A
$A_1 \leftarrow E$
a ← a+1
Normalization
A_1
=0
=1
shl A
a ← a − 1
END

If the two exponents are not equal, the mantissa having smaller exponent is shifted to the right and its exponent incremented. This process is repeated until the two exponents are equal.
The magnitude part is added or subtracted depending on the operation and the signs of the two mantissas.
If an overflow occurs when the magnitudes are added, it is transferred into flip flop E. If E is equal to 1, the bit is transferred into A_1 and all other bits of A are shifted right. The exponent must be incremented to maintain the correct number.
In the case of subtraction, the result may be zero or may have an underflow. If the mantissa is zero, the entire floating point number in the AC is made zero. Otherwise, the mantissa must have atleast one bit that is equal to 1. The mantissa has an underflow if the most significant bit in position A_1 is 0. In that case, the mantissa is shifted left and the exponent decremented. The bit in A_1 is checked again and the process is repeated until it is equal to 1. When $A_1 = 1$, the mantissa is normalized and the operation is completed.

Multiplication

- Multiply the mantissas and add the exponents.
- no comparison of exponents or alignment of mantissa is necessary.
- 4 parts.
  1. Check for zeros
  2. Add the exponents
  3. Multiply the mantissas
  4. Normalize the product

Steps 2 and 3 can be done simultaneously if separate adders are available for the mantissas and exponents.



The two operands are checked to determine if they contain a zero. If either operand is equal to zero, the product in the AC is set to zero and the operation is terminated.

If neither of the operands is equal to zero, the process continues with the exponent addition. The exponent of the multiplier is in q and the adder is between exponents a and b. The exponent from q should be transferred to a and add the two exponents.