UNIT – III

Data Representation: Data types, Complements, Fixed Point Representation, Floating Point Representation.

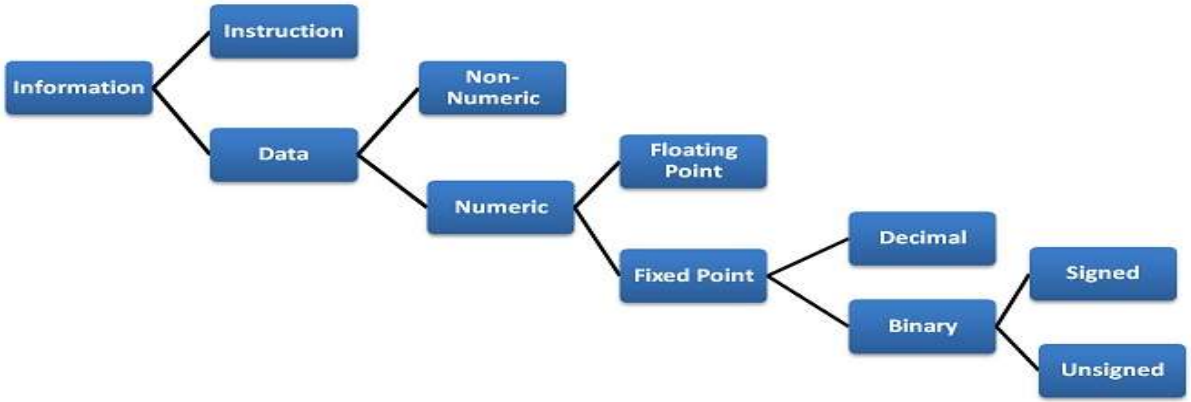
Data Representation

- Computer don’t understand human language.
- Any data(letters, symbols, pictures, audio, videos, etc.,) fed to computer should be converted to machine language first.
- Digital computers store and process information in binary form as digital logic has only two values "1" and "0”. This system is called Radix 2.
- There are many other representations like Octal (Radix 8), Hexadecimal (Radix 16), Binary coded decimal (BCD) etc.
- CPU width is measured in bits such as 8-bit CPU, 16-bit CPU, 32-bit CPU etc.
- Similarly, each memory location can store a fixed number of bits and is called memory width.
- Information handled by a computer(instruction and data) is internally represented in Binary.
- The programmer to handle the interpretation of the binary pattern and this interpretation is called *Data Representation*.

Data Representation

Choice of Data representation to be used in a computer is decided by

- The number types to be represented (integer, real, signed, unsigned, etc.)
- Range of values likely to be represented (maximum and minimum)
- The precision of the numbers i.e. maximum accuracy of representation (floating point single precision, double precision etc)
- If non-numeric(character), character representation standard to be chosen(ASCII, EBCDIC,..).
- The hardw



Data Types

- Binary information in digital computers is stored in memory or registers.
- The data types found in digital registers or memory is classified as one of the following types:
  - Numbers used in arithmetic computation
  - Letters of alphabets used in data processing
  - Other symbols used for specific purpose

Data Types

- A number system of base or radix (r) is a system that uses distinct symbols for r digits
  - Binary ->0,1
  - Octal->0,1,2,3,4,5,6,7
  - Decimal->0,1,2,3,4,5,6,7,8,9
  - Hexadecimal->0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Data Types

A number in radix r can be converted to the familiar decimal system by forming the sum of the weighted digits. For example, octal 736.4 is converted to decimal as follows:

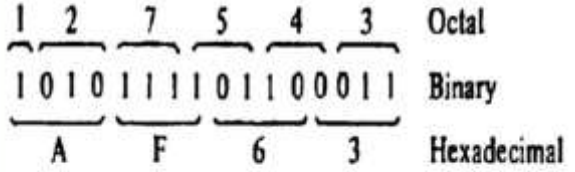
$$(736.4)_8 = 7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1}$$

$$= 7 \times 64 + 3 \times 8 + 6 \times 1 + 4/8 = (478.5)_{10}$$

The equivalent decimal number of hexadecimal F3 is obtained from the following calculation:

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

Data Types



Binary, octal, and hexadecimal conversion

Data Types

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

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

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

Data Types

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	

Data Types

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	

Data Types

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	

Data Types

- Many application of digital computers require the handling of data that consists not only of numbers but also letters of alphabets and certain special characters.
- The standard alphanumeric binary code is ASCII(American Standard code for Information Interchange)which uses 7 bits to code 128 characters.

Data Types

American Standard Code for Information Interchange (ASCII)

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

Complements

- Complements are used in digital computers for simplifying the subtraction operation.
- There are two types of complement for base r-system
  - (r-1)'s complement
  - r's complement

Complements

- Let's suppose a given number N in base r having n digits, the (r-1)'s complement of N is defined as (r^n-1)-N.
- (r-1)'s complement can be achieved as:
  - Subtracting each digit from 9(for decimal)
  - Subtracting each digit from 1(for binary)
  - Subtracting each digit from 7(for octal)
  - Subtracting each digit from F(15)(for hexadecimal)

Complements

- r's complement of n-digit number N in base r is defined as r^n-N
- r^n-N = (r^n-1)-N+1 = (r-1)'s complement + 1

Complements

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

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

Complements

Consider, for example, the subtraction 72532 – 13250 = 59282. The 10's complement of 13250 is 86750. Therefore:

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

Complements

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

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

There is no end carry

Answer is negative 59282 = 10's complement of 40718

Fixed Point Representation

- Sign bit placed in the leftmost position of the number determine if the number is positive or negative. It is 1 for negative and 0 for positive.
- In addition to sign bit a number may have decimal or binary point. There are two ways of specifying the position of the binary point in register.
  - By giving it fixed position
  - By employing floating point representation

Fixed Point Representation

- The fixed point assumes one of the following case:
  - Binary point in the extreme left to make stored number fraction .
  - Binary point in extreme right to make the stored number an integer
- Floating point representation uses the second register to store a number that designates the position of the binary point.

Fixed Point Representation

- When an integer binary number is positive, sign is represented by 0 and magnitude by positive binary number.
- When it is negative sign bit is represented by 1 and rest of the number can be represented in one of the following ways:
  - Signed-Magnitude Representation
  - Signed-1's Complement Representation
  - Signed-2's Complement Representation

Fixed Point Representation

- There is only one way to represent +14(0 0001110) in 8-bit register.
- There are 3 different ways to represent -14 with 8 bits:
  - Signed Magnitude Representation: 1 0001110
  - Signed 1's Complement Representation: 1 1110001
  - Signed 2's Complement Representation: 1 1110010

Fixed Point Representation

Decimal Representation	Sign-Magnitude Representation	Twos Complement Representation
+8	—	—
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
−0	1000	—
−1	1001	1111
−2	1010	1110
−3	1011	1101
−4	1100	1100
−5	1101	1011
−6	1110	1010
−7	1111	1001
−8	—	1000

Fixed Point Representation

- Sometime there is need to convert the data from one bit length to another(4 bit to 8 bit etc.)
- This conversion is very simple to apply in case of signed magnitude representation, just shifting of the signed bit is required.
- But this method won't be applicable in case of signed 2'complement.
- In the case of 2's complement representation, the process called sign extension is implemented.

Fixed Point Representation

+18	=	00010010	(twos complement, 8 bits)
+18	=	0000000000010010	(twos complement, 16 bits)
−18	=	11101110	(twos complement, 8 bits)
−32,658	=	1000000001101110	(twos complement, 16 bits)

Arithmetic Addition

- Adding the numbers in signed 2's complement system require only addition and complementation.
  - Add the two numbers, including their sign bits.
  - Discard any carry out of the sign(leftmost) bit position.

  Note:

  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.

Arithmetic Addition

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

Floating Point Representation

- With a fixed point notation it is possible to represent a range of positive and negative integers centered on 0.
- This approach has limitation. Very large nor can very small fractions can be represented.
- In decimal numbers we use scientific notation to represent very large and very small numbers.
- Same approach can be used to represent binary numbers.

Floating Point Representation

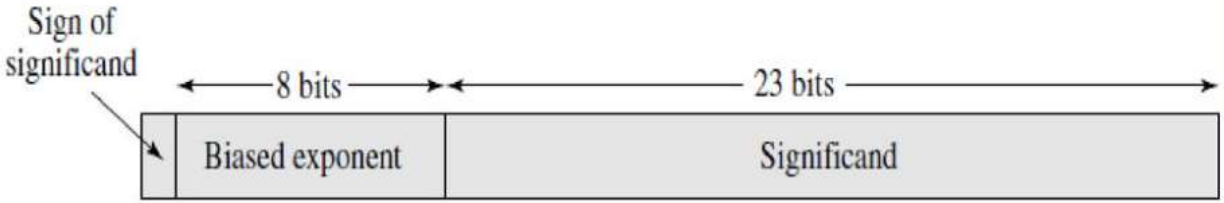
- We can represent the number in the form

$$\pm S \times B^{\pm E}$$

This number can be stored in a binary word with three fields:

- Sign: plus or minus
- Significand S
- Exponent E

Floating Point Representation



(a) Format

$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.6328125$
$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.6328125$
$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.6328125$
$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.6328125$

(b) Examples

Figure Typical 32-Bit Floating-Point Format

Floating Point Representation

- Left most bit stores the sign of the number
- The exponent value is stored in the next 8 bits using the biased representation.
- In this representation, to obtain the actual exponent a value called bias subtracted is subtracted from the field.
- Typical value for bias is 2^(k-1)-1 where k is the number of bits.
- In this case biased value is 127.

Floating Point Representation

- Final 23 bits portion is significand or mantissa.
- To simplified the operation, it is required that it is normalized. For our example a normalized nonzero number is one in the form.

$$\pm 1.bbb \ldots b \times 2^{\pm E}$$

Where b is either 0 or 1

Floating Point Representation

(a) Single format

(b) Double format

Figure IEEE 754 Formats

Floating Point Representation

- Floating point representation is defined in IEEE Standard 754.
- It defines both 32 bit single and 64 bit double format with 8-bit and 16 bit exponent.

Integer Representation

- Fixed-point numbers are also known as Integers or whole numbers.
- Computers use a fixed number of bits to represent an integer. The number of bits used to represent integer implies the maximum number that can be represented in the system hardware.
- The commonly-used bit-lengths for integers are 8-bit, 16-bit, 32-bit or 64-bit.
  An n-bit pattern can represent 2^n distinct integers and can represent integers from 0 to 2^n-1.
  Ex: If n=8, Range is 0-2^8-1, i.e. 0-255
- Besides bit-lengths, there are two representation schemes for integers:
  - Unsigned Integers: zero and positive integers.
  - Signed Integers: zero, positive and negative integers.

Unsigned Integers

- Unsigned integers can represent zero and positive integers, but not negative integers.
- The value of an unsigned integer is interpreted as " magnitude of its underlying binary pattern".

  Ex1: Suppose that n=8 and the binary pattern is 0100 0001B, the value of this unsigned integer is $1\times2^0 + 1\times2^6 = 65D$.

  Ex2: Suppose that n=16 and the binary pattern is 0001 0000 0000 1000B, the value of this unsigned integer is $1\times2^3 + 1\times2^{12} = 4104D$.

  Ex3: Suppose that n=16 and the binary pattern is 0000 0000 0000 0000B, the value of this unsigned integer is 0.

Signed Integers

- Signed integers can represent zero, positive as well as negative integers.
- Three representation schemes are available for signed integers:
  - Sign-Magnitude representation
  - 1's Complement representation
  - 2's Complement representation
- In these schemes, Most Significant Bit (MSB) is called the sign bit and is reserved for indicating the direction of the magnitude.
- If n bits are used for representation, n-1 bits indicate the absolute value of the number.
- The sign bit is used to represent the sign of the integer.
  - 0 - positive integers and
  - 1 - negative integers.
- The magnitude of the integer, however, is interpreted differently in different schemes.

Sign Integers in Sign-Magnitude Representation

In sign-magnitude representation:

- MSB is the sign bit, 0 - positive integer and 1 - negative integer.
- The remaining n-1 bits represents the magnitude (absolute value) of the integer.

Ex1: Suppose that n=8 and the binary representation is 0 100 0001B.
MSB=0 ⇒ positive, Absolute value is 100 0001B = 65D.
Hence, the integer is +65D

Ex2: Suppose that n=8 and the binary representation is 1 000 0001B.
MSB=1 ⇒ negative, Absolute value is 000 0001B = 1D.
Hence, the integer is -1D

Ex3: Suppose that n=8 and the binary representation is 0 000 0000B.
MSB=0 ⇒ positive, Absolute value is 000 0000B = 0D.
Hence, the integer is +0D

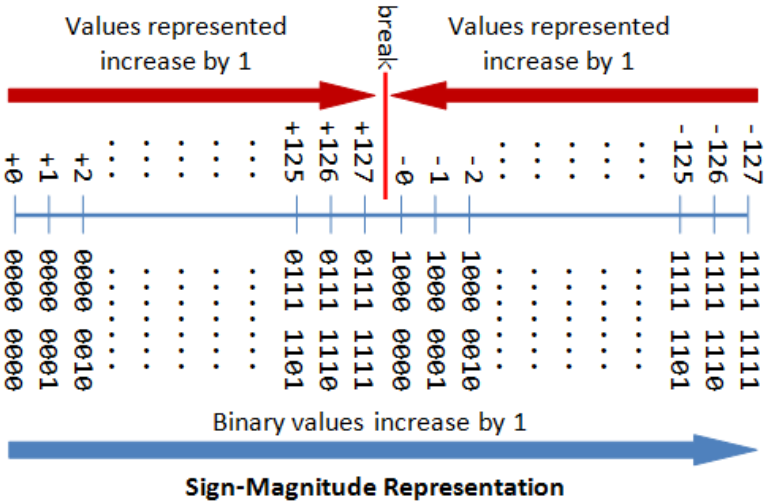
Ex4: Suppose that n=8 and the binary representation is 1 000 0000B.
MSB=1 ⇒ negative, Absolute value is 000 0000B = 0D.
Hence, the integer is -0D

Sign Integers in Sign-Magnitude Representation

The drawbacks of sign-magnitude representation are:

1. There are two representations (0000 0000B and 1000 0000B) for 0, which could lead to inefficiency and confusion.

2. Positive and negative integers need to be processed separately.



Sign Integers in 1's Complement Representation

In 1's complement representation:

- Again, MSB is the sign bit, with 0 representing positive integers and 1 representing negative integers.
- The remaining n-1 bits represents the magnitude of the integer, as follows:
  - for positive integers, absolute value of the integer is equal to "magnitude of (n-1)-bit binary pattern".
  - for negative integers, absolute value of the integer is equal to "magnitude of the complement (inverse) of the (n-1)-bit binary pattern" (hence called 1's complement).

Sign Integers in 1's Complement Representation

Ex 1: Suppose that n=8 and the binary representation 0 100 0001B.

Sign bit is 0 ⇒ positive, Absolute value is 100 0001B = 65D

Hence, the integer is +65D

Ex 2: Suppose that n=8 and the binary representation 1 000 0001B.

Sign bit is 1 ⇒ negative

Absolute value is the complement of **000 0001**, i.e., 111 1110B = 126D

Hence, the integer is **-126D**

Ex 3: Suppose that n=8 and the binary representation 0 000 0000B.

Sign bit is 0 ⇒ positive, Absolute value is 000 0000B = 0D

Hence, the integer is +0D

Ex 4: Suppose that n=8 and the binary representation 1 111 1111B.

Sign bit is 1 ⇒ negative

Absolute value is the complement of **111 1111**, i.e., 000 0000B = 0D

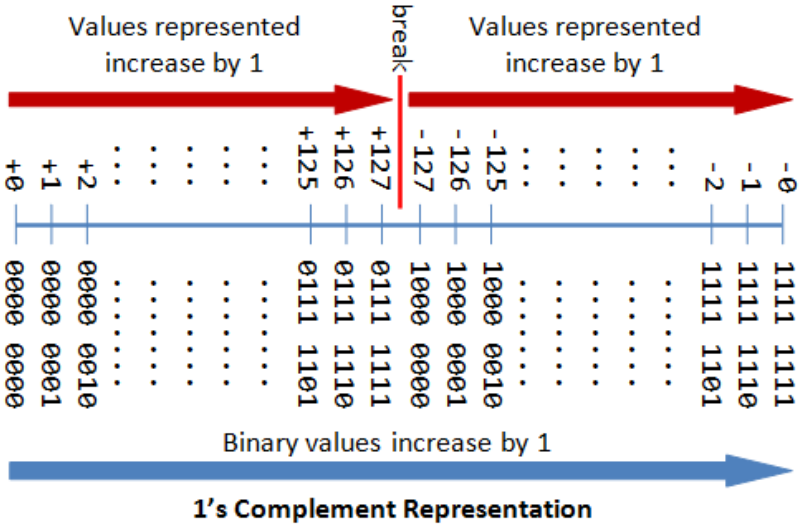
Hence, the integer is **-0D**

Sign Integers in 1's Complement Representation

The drawbacks are:

1.There are two representations (0000 0000B and 1111 1111B) for zero.

2.The positive integers and negative integers need to be processed separately.



Sign Integers in 2's Complement Representation

In 2's complement representation:

- Again, MSB is the sign bit, with 0 representing positive integers and 1 representing negative integers.
- The remaining n-1 bits represents the magnitude of the integer, as follows:
  - for positive integers, the absolute value of the integer is equal to "magnitude of the (n-1)-bit binary pattern".
  - for negative integers, the absolute value of the integer is equal to "magnitude of the complement of the (n-1)-bit binary pattern plus one" (hence called 2's complement).

Sign Integers in 2's Complement Representation

Ex 1: Suppose that *n*=8 and the binary representation 0 100 0001B.

Sign bit is 0 ⇒ positive, Absolute value is 100 0001B = 65D

Hence, the integer is +65D

Ex 2: Suppose that *n*=8 and the binary representation 1 000 0001B.

Sign bit is 1 ⇒ negative

Absolute value is the complement of **000 0001** plus 1, i.e., 111 1110 + 1 = 127D

Hence, the integer is -127D

Ex 3: Suppose that *n*=8 and the binary representation 0 000 0000B.

Sign bit is 0 ⇒ positive, Absolute value is 000 0000B = 0D

Hence, the integer is +0D

Ex 4: Suppose that *n*=8 and the binary representation 1 111 1111B.

Sign bit is 1 ⇒ negative

Absolute value is the complement of **111 1111** plus 1, i.e., 000 0000 + 1 = 1D

Hence, the integer is -1D

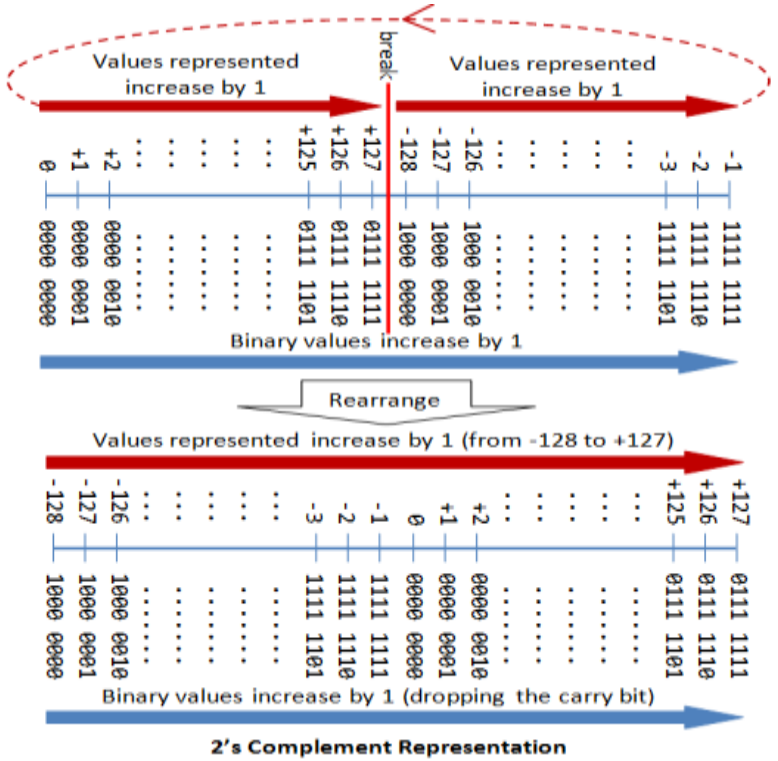
Sign Integers in 2's Complement Representation

Computers use 2's complement in representing signed integers. This is because:

1. There is only one representation for the number zero in 2's complement, instead of two representations in sign-magnitude and 1's complement.

2. Positive and negative integers can be treated together in addition and subtraction. Subtraction can be carried out using the "addition logic".

The range of *n*-bit 2's complement signed integer is $[-2^{(n-1)}, +2^{(n-1)}-1]$



Floating-Point Number Representation

- The maximum number at best represented as a whole number is 2^n.
- To increase the range of number representation, Floating Point representation with exponent and mantissa is used.(Mxr^E)
- The terms Mantissa, Significand and fraction are used synonymously.
- Floating Point number uses Sign and Magnitude representation for both mantissa and exponent.
- In computers, floating-point numbers are represented in scientific notation of fraction (F) as Mantissa and exponent (E) with a radix of 2 ($F \times 2^E$). Both E and F can be positive as well as negative.

$+ \quad 1.25 \times 10^{-1}$

Sign Mantissa Exponent

- Representation of floating point number is not unique.
- In computer, the representation is binary and the binary point is not fixed.

Ex: 23.345 can be written as 2.3345×10^1 or 0.23345×10^2 or 2334.5×10^{-2}.

- The fractional part can be normalized. In the normalized form, there is only a single non-zero digit before the radix point.

  Ex: 123.4567 can be normalized as 1.234567×10^{2}

  $1010.1011B$ can be normalized as 1.0101011×2^{3}
- Floating-point numbers suffer from loss of precision when represented with a fixed number of bits (e.g., 32-bit or 64-bit). The nearest approximation will be used which results in loss of accuracy.
- Modern computers adopt IEEE 754 standard for representing floating-point numbers.
- There are two representation schemes: 32-bit single-precision and 64-bit double-precision.

- In both the cases, MSB is sign bit for the mantissa part, followed by Exponent and Mantissa.
- The exponent part has its sign bit.
- In Single Precision, exponent is 8-bits and is represented in range -127 to +127.
- The Double Precision format has 11 bits for exponent meaning a number as large as -

Fig. (a) Single Precision Floating Point representation

Fig. (b) Double Precision Floating Point representation

Character Representation

- In computer memory, characters are "encoded" (or "represented") using a chosen "character encoding schemes".
- The most commonly-used character encoding schemes are: 7-bit ASCII, 8-bit Latin-x, and Unicode.
- It is important to note that the representation scheme must be known before a binary pattern can be interpreted.
- The 7-bit ASCII(American Standard Code for Information Interchange) can represent 128 characters and symbols.
- ASCII is originally a 7-bit code. It has been extended to 8-bit to better utilize the 8-bit computer memory organization. (8^{th} - bit - parity check in early computers).
- Code numbers 32D (20H) to 126D (7EH) are printable (displayable) characters as tabulated (arranged in hexadecimal and decimal) as follows:

Code 32D (20H) -
blank or space character.
'0' to '9' : 30H-39H
'A' to 'Z'. : 41H-5AH
'a' to 'z'. : 61H-7AH

Code numbers 0D (00H) - 31D (1FH), and 127D (7FH) - special control characters, which are non-printable (non-displayable), as tabulated.

Many of these characters were used in the early days for transmission control (e.g., STX, ETX) and printer control which are now obsolete.

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Dec	0	1	2	3	4	5	6	7	8	9
3			SP	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~			

Screenshot

Floating-Point Arithmetic Operations

- The floating-point representation of a number has **two parts**:
  - **Mantissa:** represents a **signed, fixed-point number**
  - **Exponent:** designates the **position of the decimal** (or **binary**) **point**
- The **fixed-point Mantissa** may be a **fraction** or **an integer**.
  - **For example:** the decimal number **+ 6132.789** is represented in floating-point with a **fraction** and an **exponent** as follows:

Fraction	*Exponent*
+0.6132789	+04

  - This representation is equivalent to the scientific notation **$+0.6132789 \times 10^{+4}$ (= + 6132.789)**
  - 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**.

- **Floating-point** is always **interpreted** to represent a number in the following form:

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

  - Only the **mantissa 'm'** and the **exponent 'e'** are physically **represented** in the **register** (including their **signs**)
  - The **radix 'r'** and the radix-point position of the mantissa are **always assumed**
- A **floating-point binary number** is represented in a similar manner except that it uses **base 2** for the exponent.

- **Normalization:**
  - A floating-point number is said to be **normalized** if the **most significant digit of the mantissa is NONZERO**.
- **Normalized numbers** provide the **maximum possible precision** for the floating-point number.
- A **zero cannot be normalized** because it **does not have a NON-ZERO digit**.
  - It is usually represented in floating-point by **all 0's** in the **mantissa** and **exponent**.

- **Adding** or **subtracting** two numbers requires first an **alignment of the radix point**
- **exponent parts** must be made **equal** before **adding** or **subtracting the mantissas**.
- The alignment is done by shifting
  - **one mantissa** while its **exponent is adjusted until it is equal** to the **other exponent**
  - Consider the sum of the following floating-point numbers:

$$\begin{aligned} & .5372400 \times 10^{2} \\ + & .1580000 \times 10^{-1} \end{aligned}$$

- **shifting** to the **left** causes a **loss of most significant digits**
  - **This method** may cause an **error**
- **Shifting** to the **right** causes a **loss of least significant digits**
  - **This method** is preferable because **it only reduces the accuracy**
- After this is done, the mantissas can be added:

$$\begin{aligned} & .5372400 \times 10^{2} \\ + & .0001580 \times 10^{2} \\ \hline & .5373980 \times 10^{2} \end{aligned}$$

- When two normalized mantissas are **added**, the sum may contain an **OVERFLOW digit**.
  - An overflow can be **corrected** easily by **shifting** the **sum once to the RIGHT** and **INCREMENTING the exponent**.
- When two numbers are **subtracted**, the result may contain **most significant zeros** as shown in the following example:

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

- A **floating-point number** that has a '**0' in the most significant position** of the **mantissa** is said to have an **UNDERFLOW**.
- To **normalize** a number that contains an underflow,
  - it is necessary to **shift the mantissa** to the **LEFT** and **DECREMENT** the **exponent until a nonzero digit appears in the first position**.
- In the example above, it is necessary to **shift left twice** to obtain **.35000 X 10^3**

IEE754 Floating Point Formats