COMPUTER ORGANIZATION AND ARCHITETCURE

Dr. K. Geetha

Senior Assistant Professor, CSE, SOC

Outline

- Number systems
- Number systems conversion
- Representing numbers
  - Unsigned magnitude
  - Signed magnitude
  - 1’s complement
  - 2’s complement
  - Floating point
- Representing characters & symbols
  - ASCII
  - Unicode

Data Representation

Data Representation

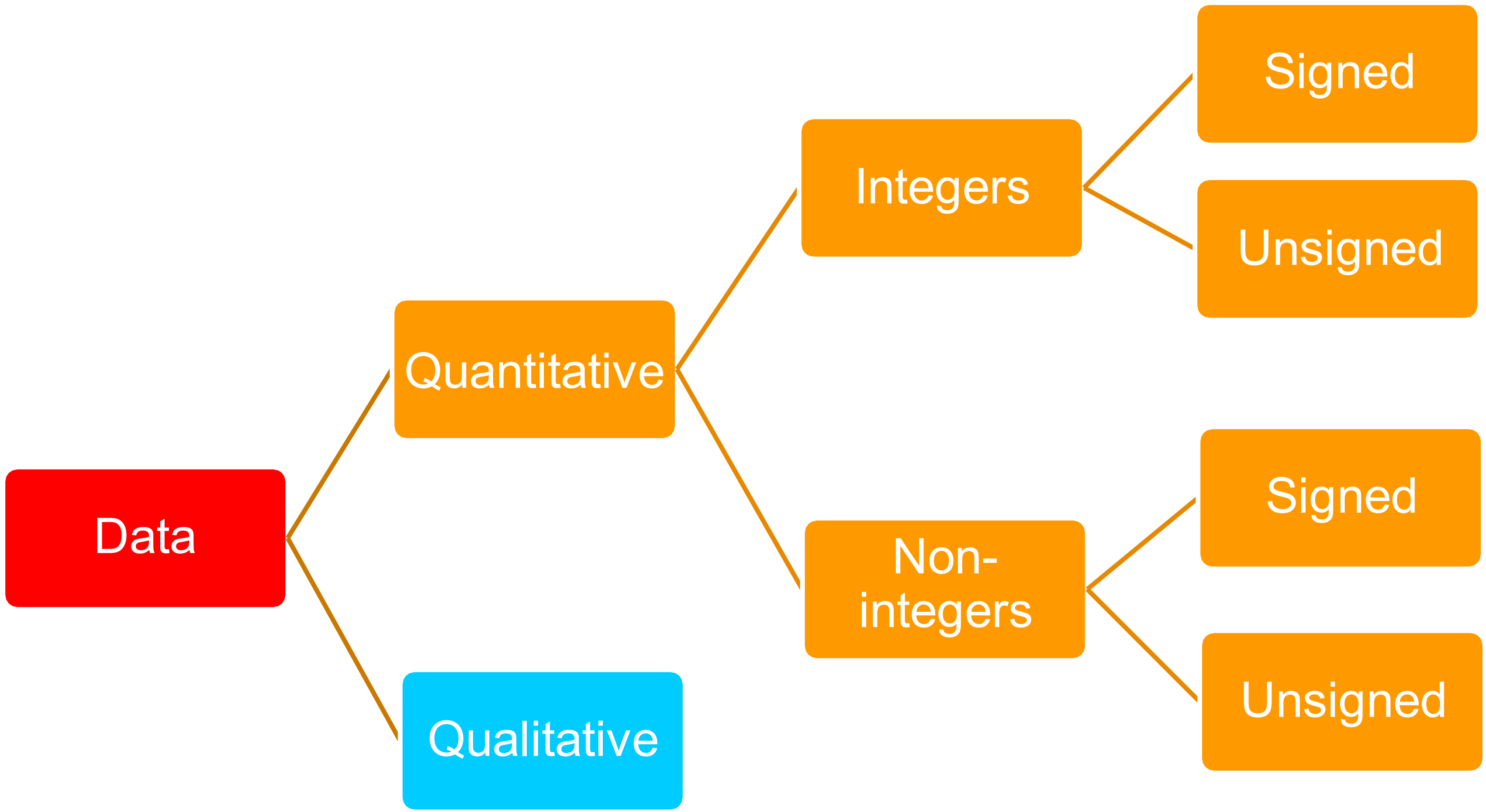
Qualitative

- Represents quality or characteristics
- Not proportional to a value
- Name, NIC no, index no, Address

Quantitative

- Quantifiable
- Proportional to value α
- No of students, marks ,CGPA

Data Representation (Contd.)



Data Representation in Computers

- Data are stored in Registers
- Registers are limited in number & size

n-1	n-2	…	…	…	2	1	0

n-bits

- With a *n*-bit register
  - Min value 0
  - Max value 2^n-1
  - MSB – n-1th bit = Sign

Number System

A number system of ***base***, or ***radix***, r is a system that uses r distinct symbols .

Numbers are represented by a string of digit .

A number N in base or radix b can be written as: **N = I . F**

$$(N)_b = d_{n-1}\, d_{n-2} \text{ — — — — } d_1\, d_0\, .\, d_{-1}\, d_{-2} \text{ — — — — } d_{-m}$$

In the above, d_{n-1} **to** d_0 **is integer** part referred as **I** , then follows a radix point, and then d_{-1} **to** d_{-m} **is fractional** part referred as **F** .

d_{n-1} = Most significant bit (MSB) , d_{-m} = Least significant bit (LSB)

Number Systems

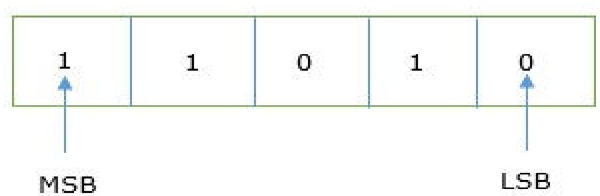
- <u>Decimal</u> number system   r = 10
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

10^5	10^4	10^3	10^2	10^1	10^0

- <u>Binary</u> number system r = 2
  - 0, 1

2^5	2^4	2^3	2^2	2^1	2^0

Each binary digit is also called a **bit.** Rightmost digit is **least significant bit (LSB)** leftmost digit is called **most significant bit (MSB)**.



Number Systems contd…

- Octal number system r = 8
  - 0, 1, 2, 3, 4, 5, 6, 7

8^5	8^4	8^3	8^2	8^1	8^0

- Hexadecimal number system r = 16
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

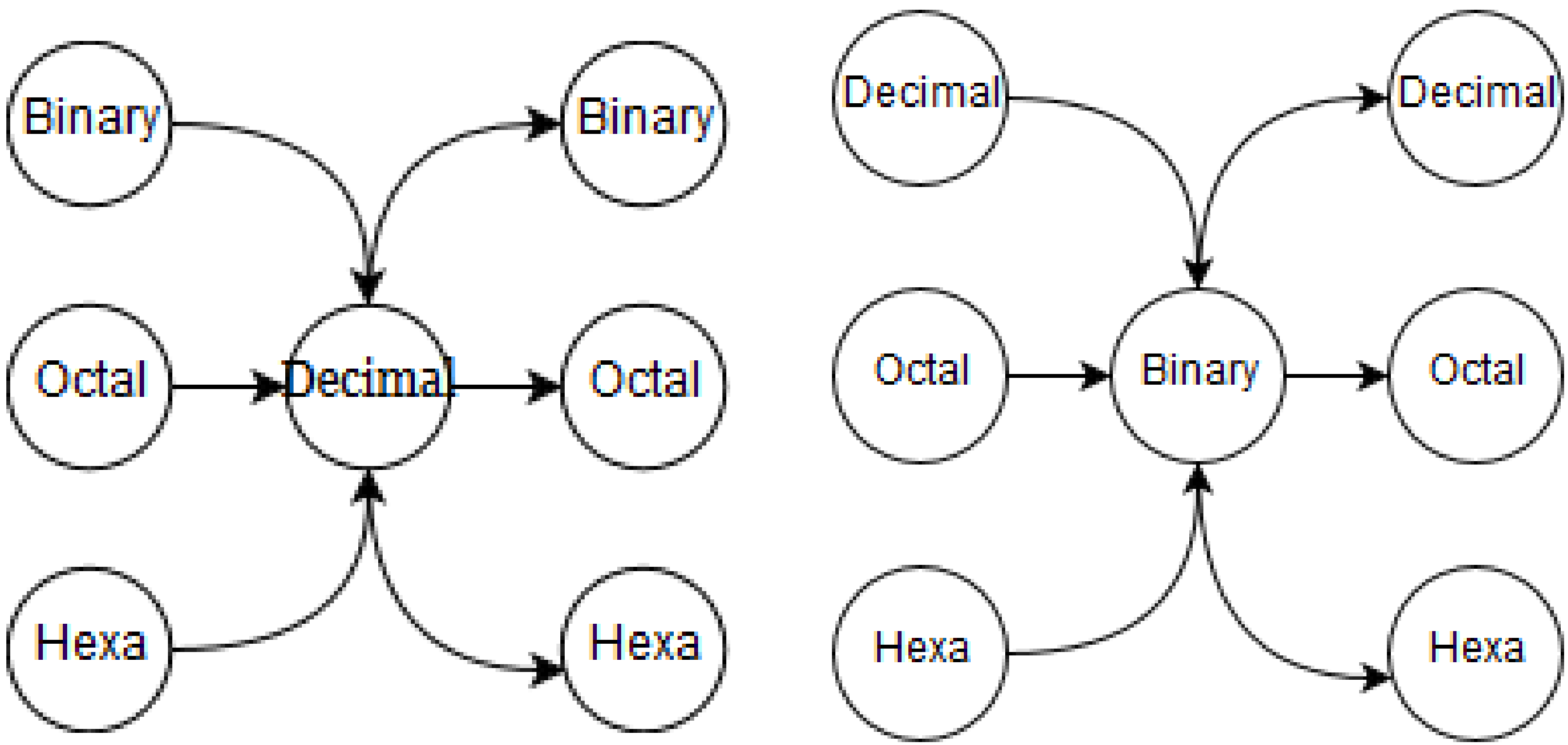
16^5	16^4	16^3	16^2	16^1	16^0

Number Systems relationship

HEXADECIMAL	DECIMAL	OCTAL	BINARY
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111

System	**Radix**	**Symbols**
Binary - B	2	0,1
Octal - O	8	0,1,2,3,4,5,6,7
Decimal- D	10	0,1,2,3,4,5,6,7,8,9
Hexa - H	16	0,1,2,3,4,5,6,7,8,9 A,B,C,D,E,F

Number System Conversion



Number System Conversion

I. Conversion from **decimal** to any **base r = 2,8,16 (Integer part)**

1. Divide **I** by **r** , collect the quotient **q** and remainders **rem**
2. Repeat step 1 with **I = q** until **q** becomes 0
3. Write the **rem** from **bottom to top** to provide the integer equivalent of the result.

162 / 2	= 81	rem 0
81 / 2	= 40	rem 1
40 / 2	= 20	rem 0
20 / 2	= 10	rem 0
10 / 2	= 5	rem 0
5 / 2	= 2	rem 1
2 / 2	= 1	rem 0
1 / 2	= 0	rem 1

Example: 162.375: So, $(162.375)_{10} = (10100010.011)_2$

Number System Conversion

I. Conversion from **decimal** to any **base r = 2,8,16 Fraction part**

Example: 162.375: So, $(162.375)_{10} = (10100010.011)_2$

1. Multiply **F by r** and find the product
2. Repeat step 1 with **F** part of the product until any of the following is satisfied
   1. F = 0
   2. F recurs again
   3. Repeat for **p** times where **P** refers to precision in terms of no. of digits
3. Write the **I** part of the product from **top to bottom** to provide the fraction equivalent of the result

$0.375 \times 2 = 0.750$
$0.750 \times 2 = 1.500$
$0.500 \times 2 = 1.000$

Number System Conversion

Decimal to Octal $(152.512)_{10} = (?)_8$

8	152	Remainder	
8	19	0	LSB
	2	3	
		2	MSB

$0.513 \times 8 = 4.104$ 4

$0.104 \times 8 = 0.832$ 0

$0.832 \times 8 = 6.656$ 6

$0.656 \times 8 = 5.248$ 5

$0.248 \times 8 = 1.984$ 1

$(0.513)_{10} = (0.40651\ldots)_8$

Complete answer is $(152.512)_{10} = (230.40651\ldots)_8$

Number System Conversion

Decimal to Hexa $(2607.565)_{10} = (?)_{16}$

16	2607	Remainder
16	162	15 LSB
	10	2
		10 MSB

$(2607)_{10} = (A2F)_{16}$

0.565 x 16 = 9.04	9
0.04 x 16 = 0.64	0
0.64 x 16 = 10.24	10 = A
0.24 x 16 = 3.84	3
0.84 x 16 = 13.44	13 = D
0.44 x 16 = 7.04	7
0.04 x 16 = 0.64	0

$(0.565)_{10} = (0.90A3D70...)_{16}$

Complete answer is $(2607.565)_{10} = (A2F.90A3D70...)_{16}$

Thank You

Binary Number System

- **Base = 2**
  - 2 digits { 0, 1 }, called *b*inary dig*its* or "*bits*"
- **Weights**
  - Weight = $(Base)^{Position}$

4	2	1		1/2	1/4
1	0	1	•	0	1
2	1	0		-1	-2

- **Magnitude**
  - Sum of "*Bit* x *Weight*"

$$1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2}$$

$$=(5.25)_{10}$$

- **Formal Notation** $(101.01)_2$
- **Groups of bits**
  - 4 bits = *Nibble* 1 0 1 1
  - 8 bits = *Byte* 1 1 0 0 0 1 0 1

Octal Number System

- **Base = 8**
  - 8 digits { 0, 1, 2, 3, 4, 5, 6, 7 }
- **Weights**
  - Weight = (*Base*) ^{Position}
- **Magnitude**
  - Sum of "*Digit* x *Weight*"
- **Formal Notation**

64	8	1		1/8	1/64
5	1	2	•	7	4
2	1	0		-1	-2

$$5 * 8^2 + 1 * 8^1 + 2 * 8^0 + 7 * 8^{-1} + 4 * 8^{-2}$$

$$=(330.9375)_{10}$$

$$(512.74)_8$$

Decimal Number System

★ **Base (also called radix) = 10**

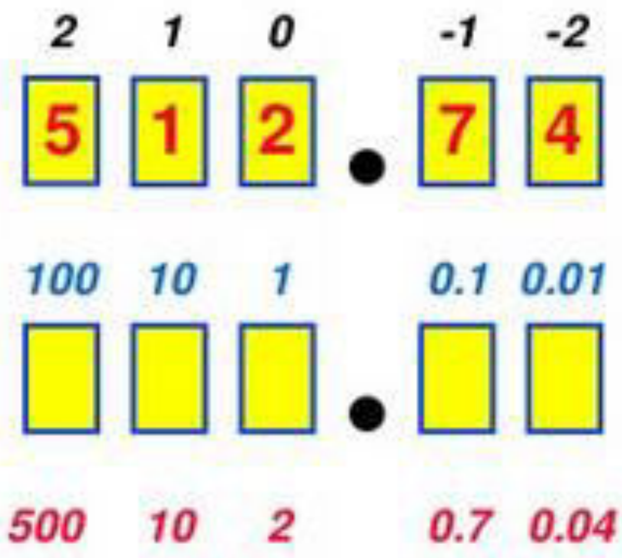
- **10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }**

★ **Digit Position**

- **Integer & fraction**

★ **Digit Weight**

- **Weight = (*Base*)** ^{Position}



★ **Magnitude**

- **Sum of "*Digit* x *Weight*"**

$$d_2 * B^2 + d_1 * B^1 + d_0 * B^0 + d_{-1} * B^{-1} + d_{-2} * B^{-2}$$

★ **Formal Notation**

$$(512.74)_{10}$$

Hexa Decimal Number System

- ★ **Base = 16**
  - 16 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }
- ★ **Weights**
  - Weight = $(Base)^{Position}$
- ★ **Magnitude**
  - Sum of "*Digit* x *Weight*"
- ★ **Formal Notation**

256	16	1		1/16	1/256
1	E	5	•	7	A
2	1	0		-1	-2

$$1*16^2+14*16^1+5*16^0+7*16^{-1}+10*16^{-2}$$

$$=(485.4765625)_{10}$$

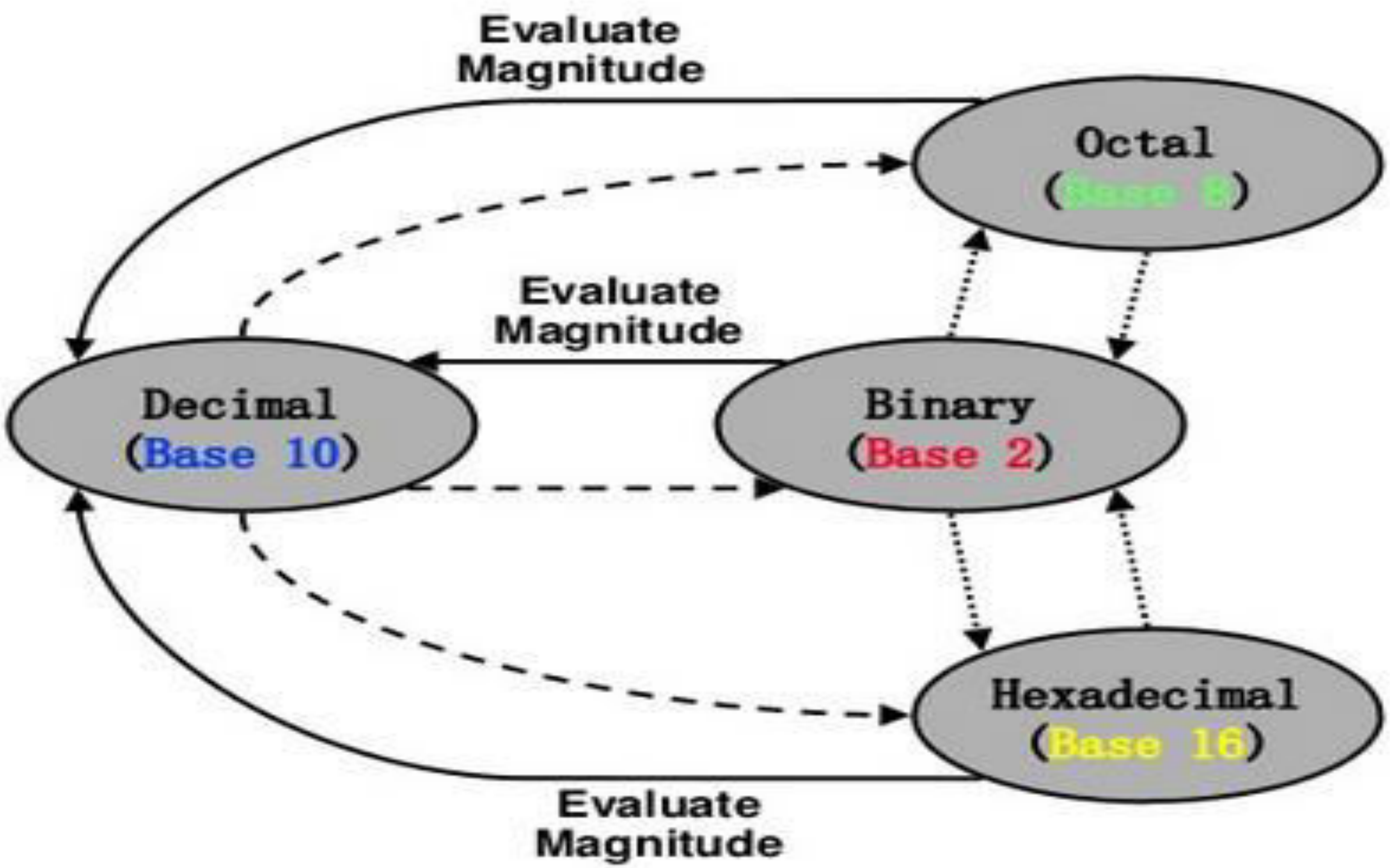
$$(1E5.7A)_{16}$$

Powers of 2

n	2^n
0	$2^0=1$
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
4	$2^4=16$
5	$2^5=32$
6	$2^6=64$
7	$2^7=128$

n	2^n	
8	$2^8=256$	
9	$2^9=512$	
10	$2^{10}=1024$	Kilo
11	$2^{11}=2048$	
12	$2^{12}=4096$	
20	$2^{20}=1M$	Mega
30	$2^{30}=1G$	Giga
40	$2^{40}=1T$	Tera

Number base Conversions



Number System Conversion

I. Conversion from any **base r = 2,8,16 to decimal**

A number N in base or radix b can be written as: **N = I . F**

$$(N)_b = d_{n-1}\, d_{n-2} \text{——————} d_1\, d_0 \,.\, d_{-1}\, d_{-2} \text{——————} d_{-m}$$

$$I = (d_{n-1} * r^{n-1}) + (d_{n-2} * r^{n-2}) + (d_{n-3} * r^{n-3}) + \ldots\ldots\ldots + (d_1 * r^1) + (d_0 * r^0)$$
$$F = (d_{-1} * r^{-1}) + (d_{-2} * r^{-2}) + \ldots\ldots (d_{-m} * r^{-m})$$

Number System Conversion examples….

Binary to Decimal conversion

$(1101.01)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{(-1)} + 1 \times 2^{(-2)} = (13.25)_{10}$

Octal to Decimal conversion

$(431.2)_8 = 4 \times 8^2 + 3 \times 8^1 + 1 \times 8^0 + 2 \times 8^{(-1)} = (281.25)_{10}$

Hexadecimal to Decimal conversion

$(6E9.D8)_{16} = 6 \times 16^2 + 14 \times 16^1 + 9 \times 16^0 + 13 \times 16^{(-1)} + 8 \times 16^{(-2)} =$

$(1769.84375)_{10}$

Number System Conversion

Binary to Octal Conversion (2^1 ---> 2^3)

step 1a: Split the Integer part of given binary number into groups of 3 bits from right (LSB).

Step 1 b: Split the fraction part of given binary number into groups of 3 bits from left (MSB)

step 2: Add 0s to the left side in Integer part and , add 0s to the right side in the fraction for lack of 3 bits.

step 3: Find the Octal equivalent for each group in both integer and fraction portion

step 4: Form the each group Octal number together in the same order.

Number System Conversion

Solved Example:

Binary – Octal Conversion

- ★ $8 = 2^3$
- ★ Each group of 3 bits represents an octal digit

Example:

Assume Zeros

$(1 0 1 1 0 . 0 1)_2$

$(2 \quad 6 \; . \; 2)_8$

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

Works both ways (*Binary* to *Octal* & *Octal* to *Binary*)

Number System Conversion

Binary to Hexa Conversion (2^1 ---> 2^4)

step 1a: Split the Integer part of given binary number into groups of 4 bits from right (LSB).

Step 1 b: Split the fraction part of given binary number into groups of 4 bits from left (MSB)

step 2: Add 0s to the left side in Integer part and , add 0s to the right side in the fraction for lack of 4 bits.

step 3: Find the Hexa equivalent for each group in both integer and fraction portion

step 4: Form the each group Hexa number together in the same order.

Number System Conversion

Solved Example:

Binary – Hexadecimal Conversion

- ★ $16 = 2^4$
- ★ Each group of 4 bits represents a hexadecimal digit

Hex	Binary
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Example:

Assume Zeros

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

$(1 \quad 6 \quad . \quad 4)_{16}$

Works both ways (*Binary* to *Hex* & *Hex* to *Binary*)

Number System Chart

Decimal	Binary	Octal	Hex
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1’s Complement

★ **1’s Complement (*Diminished Radix* Complement)**

- **All ‘0’s become ‘1’s**
- **All ‘1’s become ‘0’s**

Example $(10110000)_2$

⇨ $(01001111)_2$

If you add a number and its 1’s complement …

$$
\begin{array}{r}
1\,0\,1\,1\,0\,0\,0\,0 \\
+\ 0\,1\,0\,0\,1\,1\,1\,1 \\
\hline
1\,1\,1\,1\,1\,1\,1\,1
\end{array}
$$

2's Complement

★ **2's Complement (*Radix* Complement)**

- **Take 1's complement then add 1**

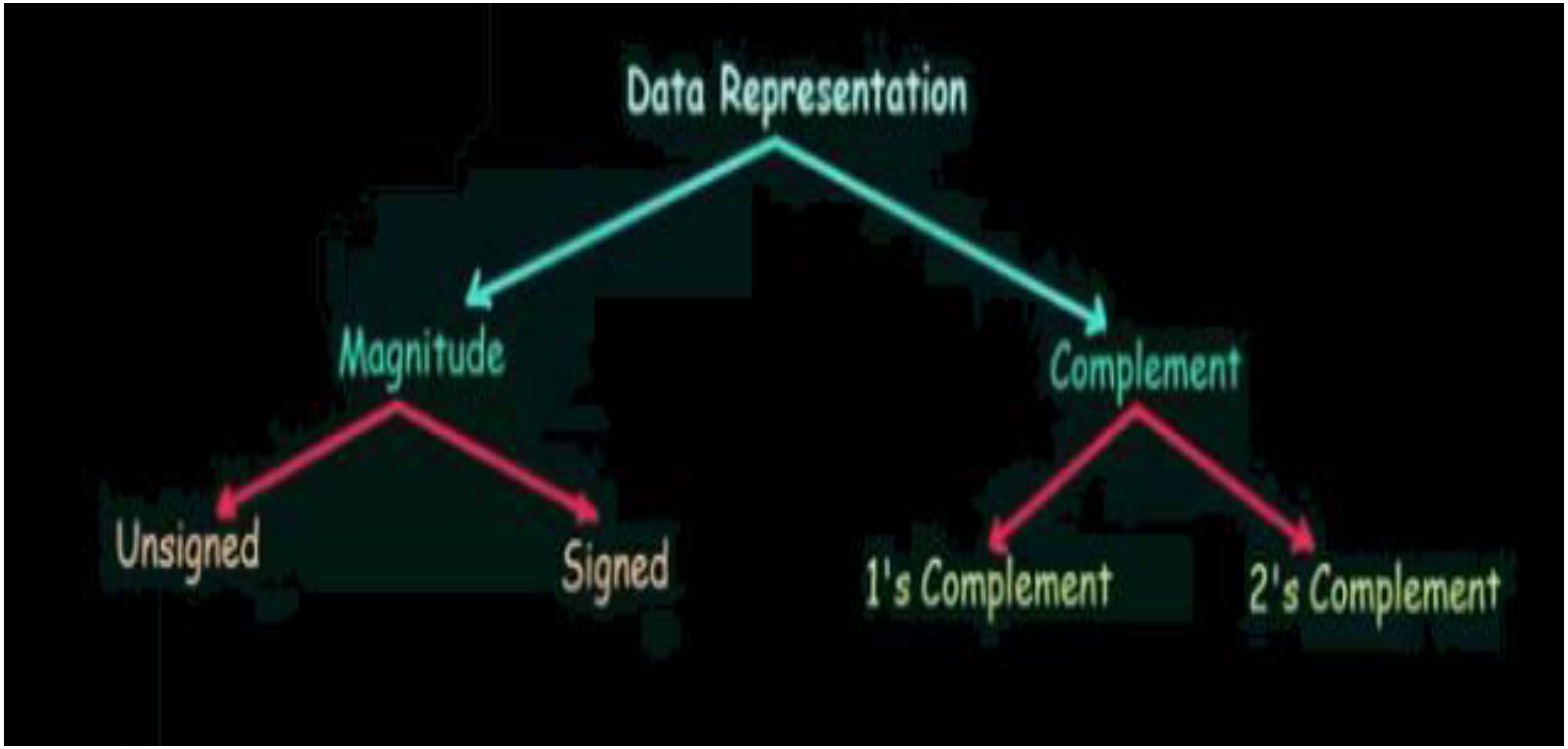
OR

- **Toggle all bits to the left of the first '1' from the right**

Example:

Number:	1 0 1 1 0 0 0 0	1 0 1 1 0 0 0 0
1's Comp.:	0 1 0 0 1 1 1 1	
+	1	
	0 1 0 1 0 0 0 0	0 1 0 1 0 0 0 0

Types of representation



Unsigned – magnitude rep.

An n-bit pattern can represent 2^n distinct integers.

Range 0 to $(2^n)-1$, as tabulated below

Can represent only +ve nos.

n	Minimum	Maximum
8	0	(2^8)-1 (=255)
16	0	(2^16)-1 (=65,535)
32	0	(2^32)-1 (=4,294,967,295) (9+ digits)
64	0	(2^64)-1 (=18,446,744,073,709,551,615) (19+ digits)

Negative numbers

- ★ **Computers Represent Information in '0's and '1's**
  - '+' and '–' signs have to be represented in '0's and '1's
- ★ **3 Systems**
  - Signed Magnitude
  - 1's Complement
  - 2's Complement

All three use the *left-most bit* to represent the sign:

- ♦ '0' ⇨ positive
- ♦ '1' ⇨ negative

Signed magnitude representation

★ **Magnitude is magnitude, *does not change with sign***

S	Magnitude (Binary)

$(+3)_{10} \Rightarrow (0\,0\,1\,1)_2$

$(-3)_{10} \Rightarrow (1\,0\,1\,1)_2$

Sign Magnitude

★ **Can't include the *sign bit* in 'Addition'**

$$
\begin{array}{rl}
0\,0\,1\,1 & \Rightarrow (+3)_{10} \\
+\;\; 1\,0\,1\,1 & \Rightarrow (-3)_{10} \\
\hline
1\,1\,1\,0 & \Rightarrow (-6)_{10}
\end{array}
$$

1’s Complement representation

★ **Positive numbers are represented in “Binary”**

0	Magnitude (Binary)

★ **Negative numbers are represented in “1’s Comp.”**

1	Code (1’s Comp.)

$(+3)_{10} \Rightarrow (0\ 011)_2$

$(-3)_{10} \Rightarrow (1\ 100)_2$

★ **There are 2 representations for ‘0’**

$(+0)_{10} \Rightarrow (0\ 000)_2$

$(-0)_{10} \Rightarrow (1\ 111)_2$

1's Complement range

★ **4-Bit Representation**

$2^4 = 16$ Combinations

$-7 \le \text{Number} \le +7$

$-2^3+1 \le \text{Number} \le +2^3-1$

★ **n-Bit Representation**

$-2^{n-1}+1 \le \text{Number} \le +2^{n-1}-1$

Decimal	1's Comp.
+7	0 1 1 1
+6	0 1 1 0
+5	0 1 0 1
+4	0 1 0 0
+3	0 0 1 1
+2	0 0 1 0
+1	0 0 0 1
+0	0 0 0 0
−0	1 1 1 1
−1	1 1 1 0
−2	1 1 0 1
−3	1 1 0 0
−4	1 0 1 1
−5	1 0 1 0
−6	1 0 0 1
−7	1 0 0 0

2's Complement representation

★ Positive numbers are represented in "Binary"

0	Magnitude (Binary)

★ Negative numbers are represented in "2's Comp."

1	Code (2's Comp.)

$(+3)_{10} \Rightarrow (0\ 011)_2$

$(-3)_{10} \Rightarrow (1\ 101)_2$

★ There is 1 representation for '0'

$(+0)_{10} \Rightarrow (0\ 000)_2$

$(-0)_{10} \Rightarrow (0\ 000)_2$

1's Comp. 1 1 1 1

$$\begin{array}{r} 1\,1\,1\,1 \\ +\quad\quad 1 \\ \hline 1\ \boxed{0\,0\,0\,0} \end{array}$$

2's Complement range

★ **4-Bit Representation**

$$2^4 = 16 \text{ Combinations}$$

$$-8 \le \text{Number} \le +7$$

$$-2^3 \le \text{Number} \le +2^3 - 1$$

★ **n-Bit Representation**

$$-2^{n-1} \le \text{Number} \le +2^{n-1} - 1$$

Decimal	2's Comp.
+7	0 1 1 1
+6	0 1 1 0
+5	0 1 0 1
+4	0 1 0 0
+3	0 0 1 1
+2	0 0 1 0
+1	0 0 0 1
+0	0 0 0 0
−1	1 1 1 1
−2	1 1 1 0
−3	1 1 0 1
−4	1 1 0 0
−5	1 0 1 1
−6	1 0 1 0
−7	1 0 0 1
−8	1 0 0 0

All types of representation

★ **4-Bit Example**

	Unsigned Binary	Signed Magnitude	1's Comp.	2's Comp.
Range	$0 \le N \le 15$	$-7 \le N \le +7$	$-7 \le N \le +7$	$-8 \le N \le +7$
Positive	Binary	0 □ □ □ Binary	0 □ □ □ Binary	0 □ □ □ Binary
Negative	X	1 □ □ □ Binary	1 □ □ □ 1's Comp.	1 □ □ □ 2's Comp.

Binary arithmetic

Binary addition:-

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Binary Multiplication:-

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Binary subtraction:-

A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Binary Division:-

A	B	Output
0	1	0
1	1	1
Division by zero is meaning less		

1's complement addition

With one's complement addition, the carry bit is "carried around" and added to the sum.

– Example: Using one's complement binary arithmetic, find the sum of 48 and - 19

```
 1   1 1
     00110000
     11101100
     --------
     00011100
          + 1
     --------
     00011101
```

We note that 19 in binary is 00010011,
so -19 in one's complement is: 11101100.

2's complement addition

48 = 00110000 19 = 00010011
-19 = 1 1101101 2'scomp(19)

48 = 00110000
-19 = 1 1101101
----
1 00011101

Discard the end around carry 1
result is 00011101 = + 29

2's complement addition contd..

Case 1: Two positive numbers

+29 ---- 0 001 1101 (Augend)

+19 ---- 0 001 0011 (Addend)

0 011 0000 (Sum = +48)

Case 2: Positive augend & negative addend

+39 ---- 0 010 0111 (Augend)

\- 22 ---- 1 110 1010 (Addend)-2's comp.

1 0 001 0001 (Sum = +17)

Discarded

2's complement addition contd..

Case 3: Positive addend & negative augend

- 47 ---- 1 101 0001 (Augend)

+29 ---- 0 001 1101 (Addend)

1 110 1110 (Sum = -18)-2's comp

Case 4: Two negative numbers

-32 ---- 1 110 0000 (Augend)

-44 ---- 1 101 0100 (Addend)

1 1 011 0100 (Sum = -76)-2's comp

discarded

Binary Subtraction – 1's complement

- Change "*Subtraction*" to "*Addition*"
- If "*Carry*" = 1 then add it to the LSB, and the result is positive (in *Binary*)
- If "*Carry*" = 0 then the result is negative (in *1's Comp.*)

$(5)_{10} - (1)_{10}$

$(+5)_{10} + (-1)_{10}$

$$\begin{array}{r} 0101 \\ +\ 1110 \\ \hline \boxed{1}\ 0011 \\ +\ \ \ \ \ 1 \\ \hline \underbrace{0100}_{+4} \end{array}$$

$(5)_{10} - (6)_{10}$

$(+5)_{10} + (-6)_{10}$

$$\begin{array}{r} 0101 \\ +\ 1001 \\ \hline \boxed{\cancel{0}}\ 1110 \\ \hline \underbrace{1110}_{-1} \end{array}$$

Binary Subtraction – 2’s complement

- ★ Change “*Subtraction*” to “*Addition*”
- ★ If “*Carry*” = 1 ignore it, and the result is positive (in *Binary*)
- ★ If “*Carry*” = 0 then the result is negative (in *2’s Comp.*)

$$(5)_{10} - (1)_{10}$$

$$(+5)_{10} + (-1)_{10}$$

$$\begin{array}{r} 0\,1\,0\,1 \\ +\ 1\,1\,1\,1 \\ \hline \cancel{\boxed{1}}\ \underbrace{0\,1\,0\,0}_{+4} \end{array}$$

$$(5)_{10} - (6)_{10}$$

$$(+5)_{10} + (-6)_{10}$$

$$\begin{array}{r} 0\,1\,0\,1 \\ +\ 1\,0\,1\,0 \\ \hline \cancel{\boxed{0}}\ \underbrace{1\,1\,1\,1}_{-1} \end{array}$$

2’s complement Subtraction

Case 1: Two positive numbers

+28 ---- 0 001 1100 (Minuend)

+19 ---- 1 110 1101 (Subtrahend)-2’s comp

1 000 1001 (Sum = +9)

discarded

Case 2: Positive no. & smaller Negative no.

+39 ---- 0 010 0111 (Minuend)

-21 ---- 0 001 0101 (Subtrahend)-2’s comp

0 011 1100 (Sum = +60)

2’s complement subtraction contd..

Case 3: Positive No. & larger Negative No.

+19 ---- 0 001 0011 (Minuend)

-43 ---- 0 010 1011 (Subtrahend)-2’s comp

0 011 1110 (Sum = +62)

Case 4: Two negative numbers

-57 ---- 1 100 0111 (Minuend)

-33 ---- 0 010 0001 (Subtrahend)-2’s comp

1 110 1000 (Sum = -24)

Thank You

Binary Arithmetic Problems

Add the following binary numbers:

1. $(1001)_2$ and $(0101)_2$
2. $(101.01)_2$ and $(1101.10)_2$

Subtract the following binary numbers:

1. $(0110)_2$ from $(1010)_2$
2. $(01011)_2$ from $(11011)_2$

Binary Arithmetic Problems

Solve the following binary multiplication

1. $(101)_2$ and $(11)_2$
2. $(1011)_2$ and $(1001)_2$

1. 5 * 3 = 15 = 1111
2. 2. 11*9 = 99 = **01100011**

Binary Arithmetic Problems

Solve the following division

1.(11001) by (101)

2. (110000) by (100)

1. 25 / 5 = 5 , 101   2. 48 / 4 = 12 , **01100**

Binary Codes

★ **Group of *n* bits**

- **Up to 2^n combinations**
- **Each *combination* represents *an element* of information**

★ **Binary Coded Decimal (BCD)**

- **Each Decimal Digit is represented by 4 bits**
- **(0 – 9) ⇨ Valid combinations**
- **(10 – 15) ⇨ Invalid combinations**

Decimal	BCD
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

BCD addition

★ **One decimal digit + one decimal digit**

- **If the result is 1 decimal digit (≤ 9), then it is a simple binary addition**

***Example*:**

$$\begin{array}{r} 5 \\ +\ 3 \\ \hline \boxed{8} \end{array} \Longleftrightarrow \begin{array}{r} 0101 \\ +\ 0011 \\ \hline \boxed{1000} \end{array}$$

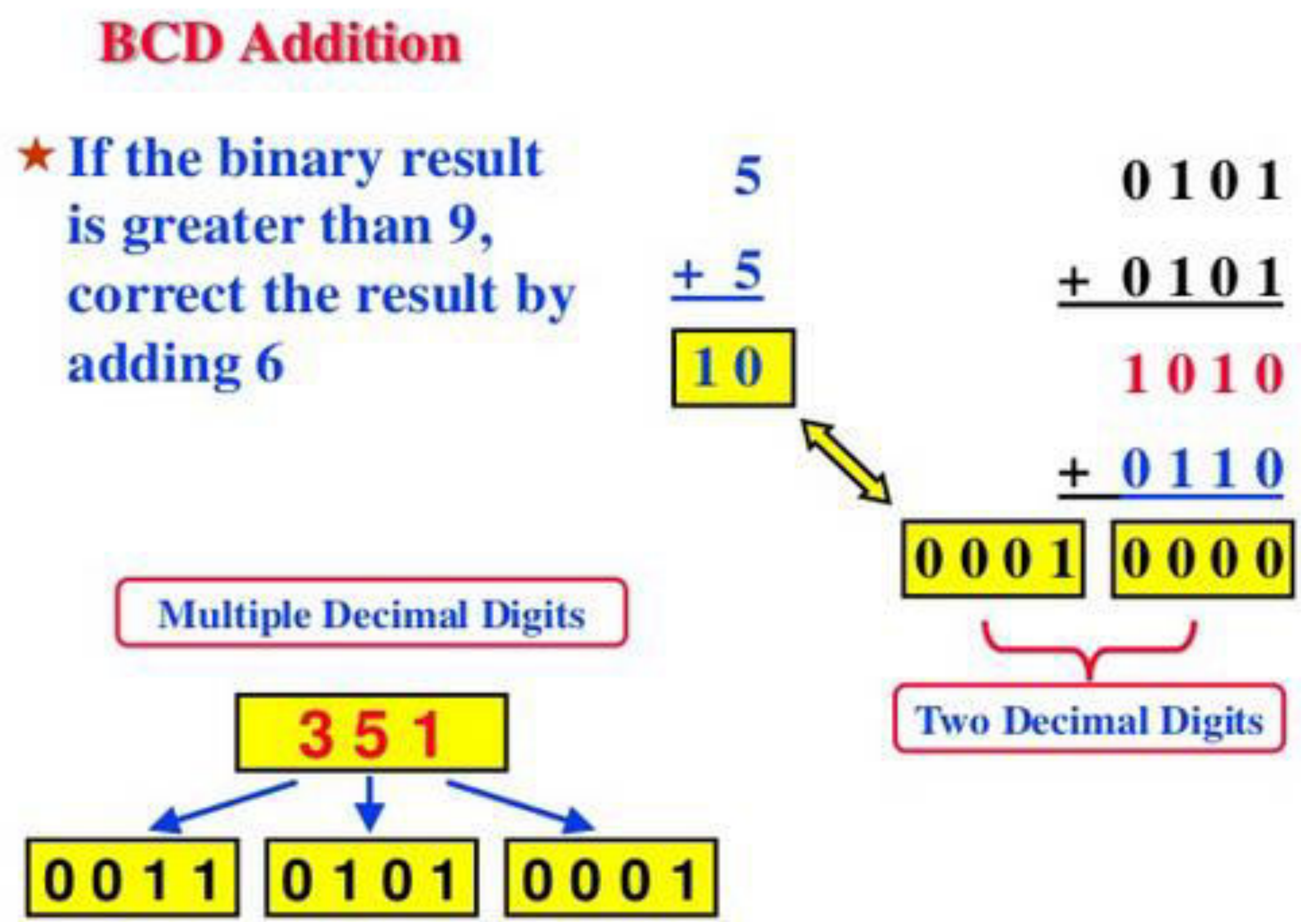
- **If the result is two decimal digits (≥ 10), then binary addition gives invalid combinations**

***Example*:**

$$\begin{array}{r} 5 \\ +\ 5 \\ \hline \end{array} \qquad \begin{array}{r} 0101 \\ +\ 0101 \\ \hline 1010 \end{array}$$

$$\boxed{0001}\ \boxed{0000} \Longleftrightarrow \boxed{10}$$

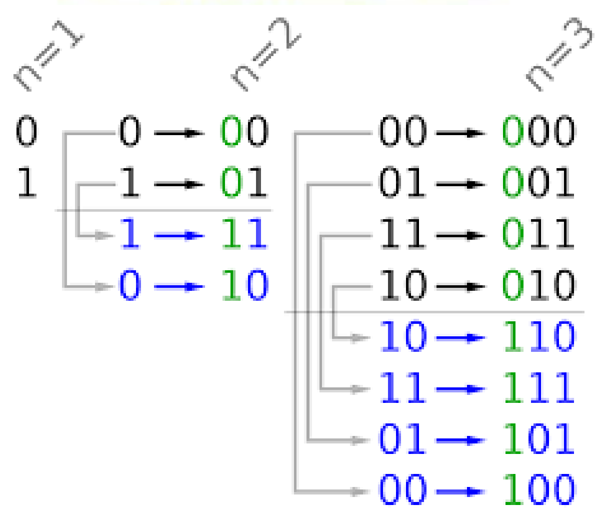
Binary Coded addition



Reflected code / Unweighted code

Gray Code

- One bit changes from one code to the next code
- Different than Binary



Decimal	Gray
00	0000
01	0001
02	0011
03	0010
04	0110
05	0111
06	0101
07	0100
08	1100
09	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

Binary
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

Floating point numbers

- Programming languages support numbers with fraction
  - Called floating-point numbers
  - Examples:

    3.14159265… (π)

    2.71828… (e)

    0.000000001 or 1.0×10^{-9} (seconds in a nanosecond)

    86,400,000,000,000 or 8.64×10^{13} (nanoseconds in a day)

    last number is a large integer that cannot fit in a 32-bit integer
- We use a scientific notation to represent
  - Very small numbers (e.g. 1.0×10^{-9})
  - Very large numbers (e.g. 8.64×10^{13})
  - Scientific notation: $\pm\, d\,.\, f_1 f_2 f_3 f_4 \ldots \times 10^{\pm e_1 e_2 e_3}$

Floating point numbers Contd..

- Examples of floating-point numbers in base 10 …
  - 5.341×10^{3} , 0.05341×10^{5} , -2.013×10^{-1} , -201.3×10^{-3}

    decimal point
- Examples of floating-point numbers in base 2 …
  - 1.00101×2^{23} , 0.0100101×2^{25} , -1.101101×2^{-3} , -1101.101×2^{-6}

    binary point
  - Exponents are kept in decimal for clarity
  - The binary number $(1101.101)_2 = 2^3+2^2+2^0+2^{-1}+2^{-3} = 13.625$
- Floating-point numbers should be normalized
  - Exactly one non-zero digit should appear before the point
    - In a decimal number, this digit can be from 1 to 9
    - In a binary number, this digit should be 1
  - Normalized FP Numbers: 5.341×10^{3} and -1.101101×2^{-3}
  - NOT Normalized: 0.05341×10^{5} and -1101.101×2^{-6}

Floating point Representation

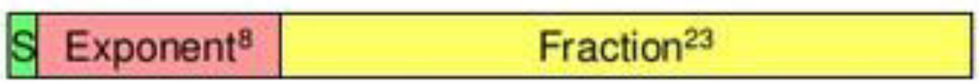
- A floating-point number is represented by the triple
  - *S* is the Sign bit (0 is positive and 1 is negative)
    - Representation is called sign and magnitude
  - *E* is the Exponent field (signed)
    - Very large numbers have large positive exponents
    - Very small close-to-zero numbers have negative exponents
    - More bits in exponent field increases range of values
  - *F* is the Fraction field (fraction after binary point)
    - More bits in fraction field improves the precision of FP numbers

S	Exponent	Fraction

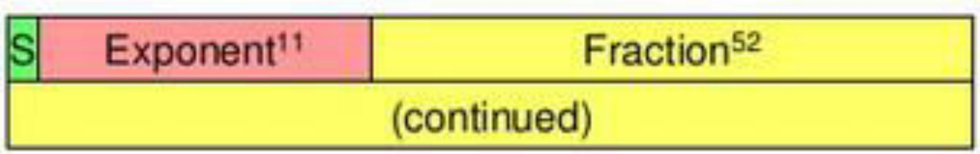
$$\boxed{\text{Value of a floating-point number} = (-1)^{S} \times \text{val}(F) \times 2^{\text{val}(E)}}$$

Floating point Standard

- Found in virtually every computer invented since 1980
  - Simplified porting of floating-point numbers
  - Unified the development of floating-point algorithms
  - Increased the accuracy of floating-point numbers
- Single Precision Floating Point Numbers (32 bits)
  - 1-bit sign + 8-bit exponent + 23-bit fraction



- Double Precision Floating Point Numbers (64 bits)
  - 1-bit sign + 11-bit exponent + 52-bit fraction



Floating point Normalization

- For a normalized floating point number (S, E, F)

- Significand is equal to $(1.F)_2 = (1.f_1 f_2 f_3 f_4 \ldots)_2$
  - IEEE 754 assumes hidden 1. (not stored) for normalized numbers
  - Significand is 1 bit longer than fraction
- Value of a Normalized Floating Point Number is

$$(-1)^S \times (1.F)_2 \times 2^{\text{val}(E)}$$
$$(-1)^S \times (1.f_1 f_2 f_3 f_4 \ldots)_2 \times 2^{\text{val}(E)}$$
$$(-1)^S \times (1 + f_1 \times 2^{-1} + f_2 \times 2^{-2} + f_3 \times 2^{-3} + f_4 \times 2^{-4} \ldots)_2 \times 2^{\text{val}(E)}$$

$(-1)^S$ is 1 when S is 0 (positive), and -1 when S is 1 (negative)

Biased Exponent representation

- How to represent a signed exponent? Choices are …
  - Sign + magnitude representation for the exponent
  - Two's complement representation
  - Biased representation
- IEEE 754 uses biased representation for the exponent
  - Value of exponent = val(E) = E – Bias (Bias is a constant)
- Recall that exponent field is 8 bits for single precision
  - E can be in the range 0 to 255
  - E = 0 and E = 255 are reserved for special use (discussed later)
  - E = 1 to 254 are used for normalized floating point numbers
  - Bias = 127 (half of 254), val(E) = E – 127
  - val(E=1) = –126, val(E=127) = 0, val(E=254) = 127

Biased exponent Contd..

- For double precision, exponent field is 11 bits
  - E can be in the range 0 to 2047
  - $E = 0$ and $E = 2047$ are reserved for special use
  - $E = 1$ to 2046 are used for normalized floating point numbers
  - Bias = 1023 (half of 2046), val(E) = $E - 1023$
  - val(E=1) = –1022, val(E=1023) = 0, val(E=2046) = 1023
- Value of a Normalized Floating Point Number is

$$(-1)^S \times (1.F)_2 \times 2^{E - \text{Bias}}$$
$$(-1)^S \times (1.f_1 f_2 f_3 f_4 \ldots)_2 \times 2^{E - \text{Bias}}$$
$$(-1)^S \times (1 + f_1 \times 2^{-1} + f_2 \times 2^{-2} + f_3 \times 2^{-3} + f_4 \times 2^{-4} \ldots)_2 \times 2^{E - \text{Bias}}$$

Single precision - example

- What is the decimal value of this Single Precision float?

1	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Solution:
  - Sign = 1 is negative
  - Exponent = $(01111100)_2 = 124$, $E -$ bias $= 124 - 127 = -3$
  - Significand = $(1.0100 \ldots 0)_2 = 1 + 2^{-2} = 1.25$ (1. is implicit)
  - Value in decimal = $-1.25 \times 2^{-3} = -0.15625$
- What is the decimal value of?

0	1	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Solution:

  implicit

  - Value in decimal = $+(1.01001100 \ldots 0)_2 \times 2^{130-127} =$
    $(1.01001100 \ldots 0)_2 \times 2^3 = (1010.01100 \ldots 0)_2 = 10.375$

Double precision - example

- What is the decimal value of this Double Precision float ?

0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Solution:
  - Value of exponent = $(10000000101)_2$ – Bias = 1029 – 1023 = 6
  - Value of double float = $(1.00101010 \ldots 0)_2 \times 2^6$ (1. is implicit) = $(1001010.10 \ldots 0)_2 = 74.5$

- What is the decimal value of ?

1	0	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Do it yourself! (answer should be $-1.5 \times 2^{-7} = -0.01171875$)

FP decimal to Binary

- Convert –0.8125 to binary in single and double precision
- Solution:
  - Fraction bits can be obtained using multiplication by 2
    - $0.8125 \times 2 = 1.625$
    - $0.625 \times 2 = 1.25$
    - $0.25 \times 2 = 0.5$
    - $0.5 \times 2 = 1.0$
    - Stop when fractional part is 0

    $0.8125 = (0.1101)_2 = \frac{1}{2} + \frac{1}{4} + 1/16 = 13/16$
  - Fraction = $(0.1101)_2 = (1.101)_2 \times 2^{-1}$ (Normalized)
  - Exponent = -1 + Bias = 126 (single precision) and 1022 (double)

Single Precision

1	0	1	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Double Precision

1	0	1	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Largest Normalized Float

- What is the Largest normalized float?
- Solution for Single Precision:

0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

  - Exponent – bias = 254 – 127 = 127 (largest exponent for SP)
  - Significand = $(1.111 \ldots 1)_2$ = almost 2
  - Value in decimal $\approx 2 \times 2^{127} \approx 2^{128} \approx 3.4028 \ldots \times 10^{38}$
- Solution for Double Precision:

0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

  - Value in decimal $\approx 2 \times 2^{1023} \approx 2^{1024} \approx 1.79769 \ldots \times 10^{308}$
- Overflow: exponent is too large to fit in the exponent field

Smallest Normalized float

- What is the smallest (in absolute value) normalized float?
- Solution for Single Precision:

0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

  - Exponent – bias = 1 – 127 = –126 (smallest exponent for SP)
  - Significand = $(1.000 \ldots 0)_2 = 1$
  - Value in decimal = $1 \times 2^{-126} = 1.17549 \ldots \times 10^{-38}$
- Solution for Double Precision:

0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

  - Value in decimal = $1 \times 2^{-1022} = 2.22507 \ldots \times 10^{-308}$
- Underflow: exponent is too small to fit in exponent field

Character Representation (Cont.)

- With a single byte (8-bits) 256 characters can be represented
- Standards
  - ASCII – American Standard Code for Information Interchange
  - EBCDIC – Extended Binary-Coded Decimal Interchange Code
  - Unicode

ASCII Code

- De facto world-wide standard
- Used to represent
  - Upper & lower-case Latin letters
  - Numbers
  - Punctuations
  - Control characters
- There are 128 standard ASCII codes
  - Can be represented by a 7 digit binary number
    - 000 0000 through 111 1111
  - Plus parity bit

ASCII code

American Standard Code for Information Interchange

Info	7-bit Code
A	1000001
B	1000010
⋮	⋮
Z	1011010
a	1100001
b	1100010
⋮	⋮
z	1111010
@	1000000
?	0111111
+	0101011

ASCII Table

ASCII	Hex	Symbol
0	0	NUL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOT
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS
9	9	TAB
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI

ASCII	Hex	Symbol
32	20	(space)
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/

ASCII	Hex	Symbol
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

ASCII Table (Cont.)

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
64	40	@	80	50	P	96	60	`
65	41	A	81	51	Q	97	61	a
66	42	B	82	52	R	98	62	b
67	43	C	83	53	S	99	63	c
68	44	D	84	54	T	100	64	d
69	45	E	85	55	U	101	65	e
70	46	F	86	56	V	102	66	f
71	47	G	87	57	W	103	67	g
72	48	H	88	58	X	104	68	h
73	49	I	89	59	Y	105	69	i
74	4A	J	90	5A	Z	106	6A	j
75	4B	K	91	5B	[	107	6B	k
76	4C	L	92	5C	\	108	6C	l
77	4D	M	93	5D	]	109	6D	m
78	4E	N	94	5E	^	110	6E	n
79	4F	O	95	5F	_	111	6F	o

Unicode

- Designed to overcome limitation of number of characters
- Assigns unique character codes to characters in a wide range of languages
- 65,536 (2^{16}) distinct Unicode characters

Unicode provides a unique number for every character,
no matter what the platform,
no matter what the program,
no matter what the language

Unicode Goals

- Universal – Should be the only character set ever needed
- Semantics – All characters must have well defined semantics
- Unicode Transformation Format (UTF) is available as 8,16,32 and are referred as
- UTF – 8, UTF – 16, UTF – 32

Error detecting codes

★ **Parity**

One bit added to a group of bits to make the total number of '1's (including the parity bit) *even* or *odd*

	Parity	4-bit Example	Parity	7-bit Example
• **Even**	1	0 1 1 1	0	1 0 0 0 0 0 1
• **Odd**	0	0 1 1 1	1	1 0 0 0 0 0 1

★ **Good for checking single-bit errors**