



02: Data Representation and Arithmetic

IT1206- Computer Systems

Level I - Semester 1

List of Sub Topics

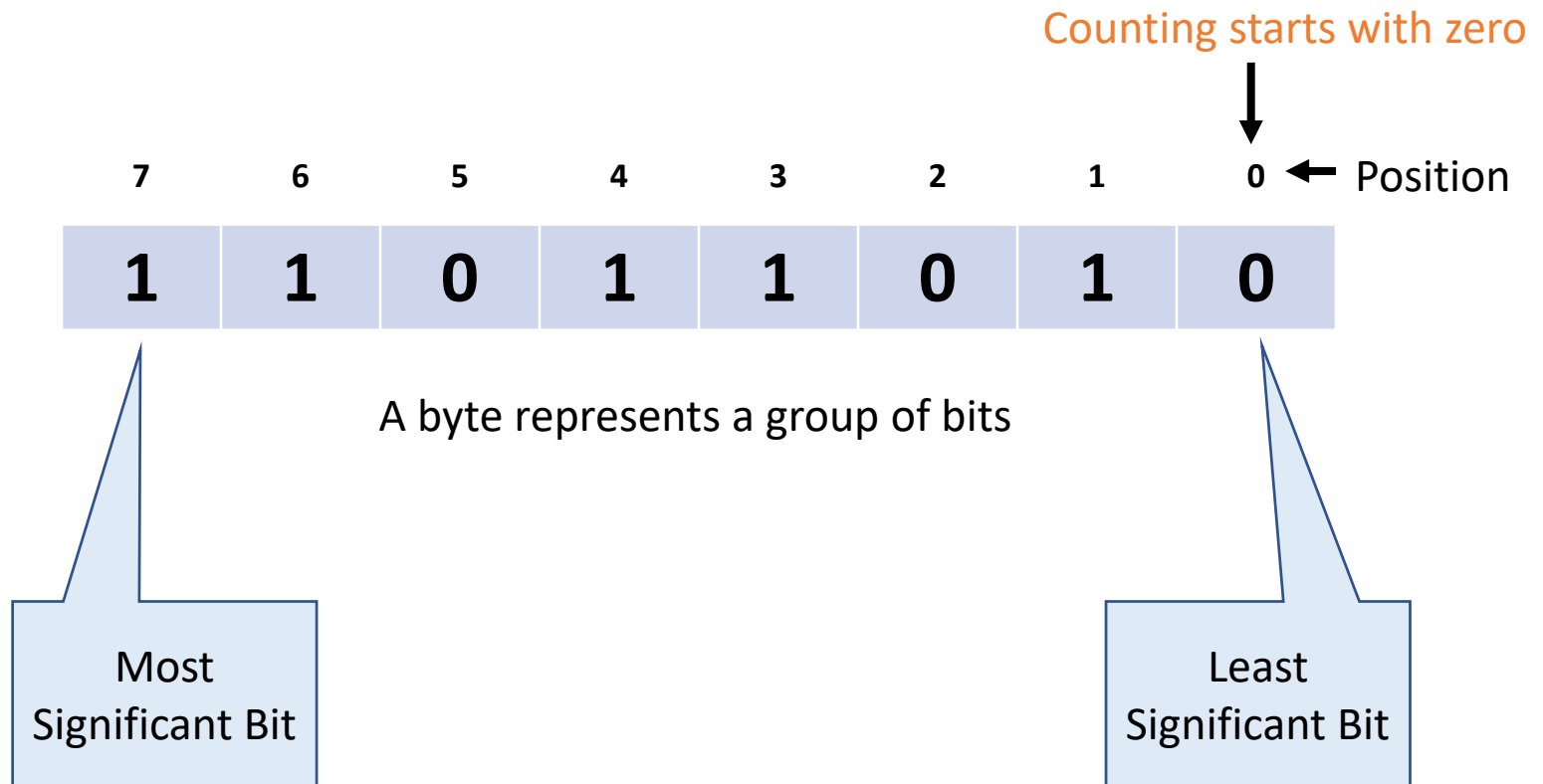
- Positinal Numbering System
- Decimal-Binary Conversion
 - Unsigned whole number conversion
 - Fraction conversion
 - Power of two Radices conversion
- Signed Integer Representation
 - Signed Magnitude
 - Two's complement
- Floating Point Representation
- Character Codes

What is a bit ?

- A bit is the smallest unit of data on a machine.
- A bit can hold only one of two values: 0 or 1
- In a computer, a bit represents one of possible two states.
- Because bits are so small, you rarely work with information one bit at a time.

What is a byte ?

- Conventionally, a group of 8 bits are named as a byte.



Word

- A word is a fixed-sized piece of data which is handled as a unit by the CPU of a computer.
- E.g. 32 bit, 64 bit, and so on.
- A word may consist with several bytes.

Units of Data Measurement

- 8 Bits = 1 Byte
- 1024 Bytes = 1 Kilobyte (KB)
- 1024 KB = 1 Megabyte (MB)
- 1024 MB = 1 Gigabyte (GB)
- 1024 GB = 1 Terabytes (TB)
- 1024 TB = 1 Petabytes (PB)
- 1024 PB = 1 Exabyte
- 1024 EB = 1 Zettabyte
- 1024 ZB = 1 Yottabyte

Positional Numbering System

- A positional numbering system is the representation of numbers by an ordered set of numerals symbols (digits) in which the value of a numeral symbol depends on its position.
- We all are familiar with base (radix) 10 number system.
- In based 10 number system, number 283.75 means;
 $2 \times 10^2 + 8 \times 10^1 + 3 \times 10^0 + 7 \times 10^{-1} + 5 \times 10^{-2}$

2	8	3	•	7	5
10^2	10^1	10^0		10^{-1}	10^{-2}

Number Systems


- There are many other number systems
 - Decimal [0,1,2,3,4,5,6,7,8,9]
 - Binary [0,1]
 - Hexa Decimal [0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F]
 - Octal [0,1,2,3,4,5,6,7]

Converting Decimal to Binary (whole numbers)

- To convert decimal whole number to binary, start with the integer part in the given number and divide it by 2
- Keep the resulting quotient and the remainder separately.
- Continue dividing the quotient by 2 until you get a quotient of zero.
- At the end just write out the remainders in the reverse order.

Converting Decimal to Binary (whole numbers)

123		1111011₂
$\div 2$		
<hr/> 61	→ remainder 1	
$\div 2$		
<hr/> 30	→ remainder 1	
$\div 2$		
<hr/> 15	→ remainder 0	
$\div 2$		
<hr/> 7	→ remainder 1	
$\div 2$		
<hr/> 3	→ remainder 1	
$\div 2$		
<hr/> 1	→ remainder 1	
$\div 2$		
<hr/> 0	→ remainder 1	



Converting Decimal to Binary (fractional numbers)

- To convert decimal fraction to binary, start with the fractional part in the given number and multiply it by 2.
- Keep the resulting integer and fractional parts separately.
- Continue multiplying by 2 until you get a resulting fractional part equal to zero.
- At the end just write out the integer parts from the results of each multiplication starting from the first multiplication.

$$\begin{array}{l} 0.375 \times 2 = \mathbf{0} + 0.75 \\ 0.75 \times 2 = \mathbf{1} + 0.5 \\ 0.5 \times 2 = \mathbf{1} + 0 \end{array} \quad \downarrow$$

$$\mathbf{0.375_{10} = 0.011_2}$$

Activities

Convert following numbers to binary

- 2273
- 12478
- 0.2487
- 0.99724
- 347.35
- 437.639

Converting Binary to Decimal

- If we want to convert 101.11_2 to decimal;

1	0	1	•	1	1
2^2	2^1	2^0		2^{-1}	2^{-2}

$$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$$

$$4 + 0 + 1 + 0.5 + 0.25$$

$$5.75$$

Converting Between Power-of-Two Radices

- Hexadecimal and Octal are two number systems where their radices are of power of two ($16 = 2^4$ and $8 = 2^3$).
- We can easily convert numbers between power of two radices number systems.

Hexadecimal

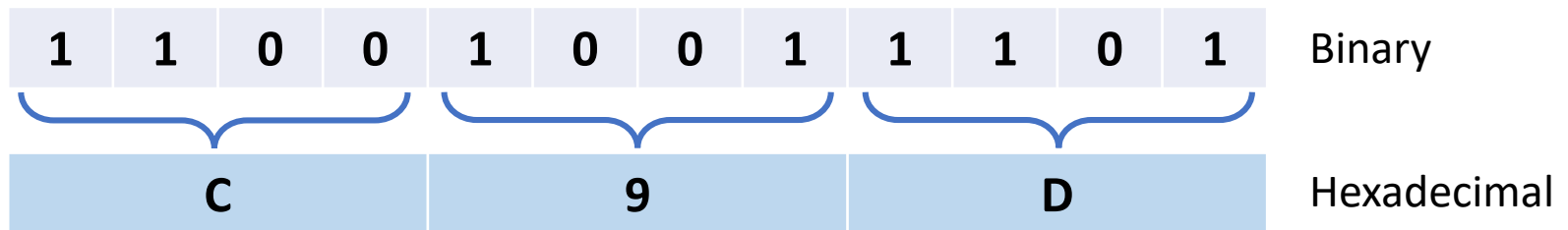
HEX	Bit Pattern	HEX	Bit Pattern
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Octal

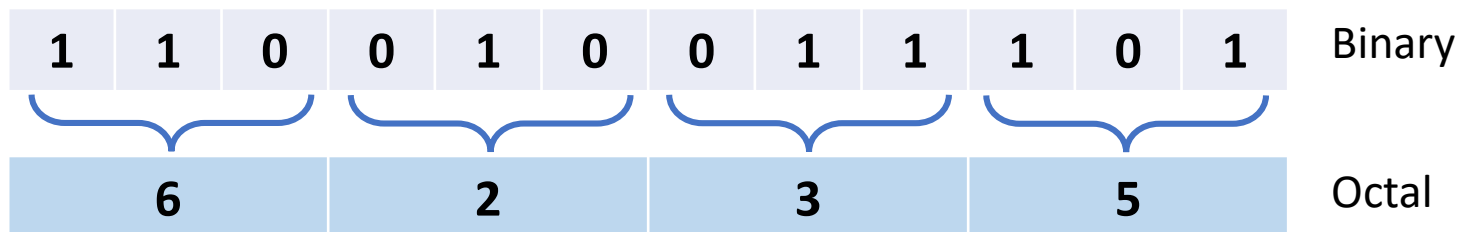
Octal	Bit Pattern
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Converting Between Power-of-Two Radices

- Ex. 110010011101_2
- Conversion to hexadecimal



- Conversion to octal



Binary Addition

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 10$ (carry: 1)

- E.g.

$$\begin{array}{r} 1\ 1\ 1\ 1\ 1\ (\text{carry}) \\ \ 0\ 1\ 1\ 0\ 1 \\ + \ 1\ 0\ 1\ 1\ 1 \\ \hline = \underline{\underline{1\ 0\ 0\ 1\ 0\ 0}} \end{array}$$

Binary Subtraction

- $0 - 0 = 0$
- $0 - 1 = 1$ (with borrow)
- $1 - 0 = 1$
- $1 - 1 = 0$

- E.g.

$$\begin{array}{rcccccc} & * & & * & * & & \text{(borrow)} \\ 1 & 0 & 1 & 1 & 0 & 1 & \\ - & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline = & 0 & 1 & 0 & 1 & 1 & 0 \\ \hline \hline \end{array}$$

Binary Multiplication

- E.g.

[illegible]

Binary Division

- E.g.

$$\begin{array}{r} 101 \leftarrow \text{Quotient} \\ 101 \overline{) 11011} \\ \underline{- 101} \\ 11 \\ \underline{- 00} \\ 111 \\ \underline{- 101} \\ 10 \leftarrow \text{Remainder} \end{array}$$

Representing Numbers

- Problems of number representation
 - Positive and negative
 - Radix point
 - Range of representation
- Different ways to represent numbers
 - Unsigned representation: non-negative integers
 - Signed representation: integers
 - Floating-point representation: fractions

Unsigned Binary Numbers

- Unsigned binary numbers
 - Have 0 and 1 to represent numbers
 - Only positive numbers stored in binary
 - The Smallest binary number that can be stored using 8 bits would be

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

 which equals to 0
 - The largest binary number would be

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

 which equals to $128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255 = 2^8 - 1$
- Therefore the range is 0 - 255 (256 numbers)

Signed Magnitude Representation

- Signed Magnitude (Signed binary) numbers
- Have 0 and 1 to represent numbers
- By convention, the leftmost bit is used as the sign bit
 - 0 for positive
 - 1 for negative

Sign bit



Signed Magnitude (Signed Binary) Numbers

- Signed Magnitude (Signed binary) numbers
- The Smallest positive binary number is

$$\begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} = 0$$

- The largest positive binary number is

$$\begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline \end{array} = 127$$

- Therefore the range for positive numbers is 0 - 127
(the range is 128 numbers)

Challenges with Negative Numbers in Signed Binary

Problems with simple signed representation

- Two representation of zero: + 0 and – 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

 and

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

- Need to consider both sign and magnitude in arithmetic
- E.g. $5 - 3$

$$= 5 + (-3)$$

$$= 00000101 + 100000011$$

$$= 10001000$$

$$= -8$$



Problems with Simple Signed Representation

- Need to consider both sign and magnitude in arithmetic

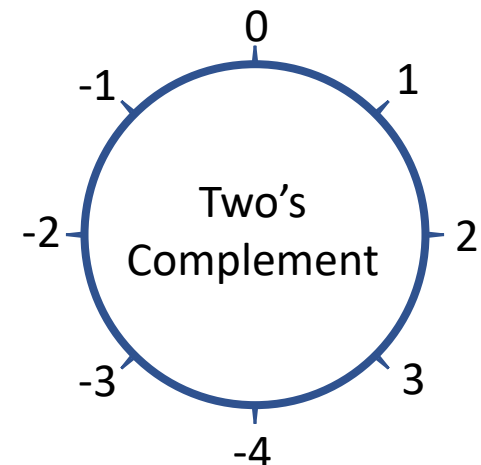
$$\begin{aligned}\text{E.g. } &= 18 + (-18) \\ &= 00010010 + 10010010 \\ &= 10100100 \\ &= -36 \quad \mathbf{X}\end{aligned}$$

- We have to change the way these arithmetic operations are performed based on the signs of two numbers.

Two's Complement Number System

- Two's complement is the most common method of representing signed integers on computers.
- In this scheme most significant bit indicates the sign.
- In order to represent the negative number, the positive value of the number will be two's complemented.

Bits	Unsigned Value	Two's Complement Value
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1



Two's Complement

- Start from the signed binary representation of its positive value
- Copy the bit pattern from right to left until a 1 has been copied
- Complement the remaining bits: all the 1's with 0's, and all the 0's with 1's
- Ex. -105_{10} in Two's complement ($105_{10} = 0110\ 1001_2$)
 $0110\ 1001 \Rightarrow 1001\ 0111$
 $-105_{10} = 1001\ 0111$
- An exception: $1\ 0\ 0\ 0\ 0\ 0\ 0\ 0 = -128_{10}$

Example: Conversion

- Ex. Convert decimal -104 to two's complement
- First signed magnitude representation of 104;
 $104_{10} = 0110\ 1000_2$
- Find the first digit 1 from right to left
 $0110\ \underline{1}000$
- Copy everything till first digit 1 from right to left
 $____ 1000$
- Then complement the remaining bits
10011000
- Therefore, two's complement of -104_{10} is 10011000

Your Turn...!

- What is the SMALLEST and LARGEST signed binary numbers that can be stored in 1 BYTE ?

Eight-bit signed integers

Bits ◆	Unsigned value ◆	Two's complement value ◆	
0000 0000	0	0	Positive
0000 0001	1	1	
0000 0010	2	2	
0111 1110	126	126	
0111 1111	127	127	
1000 0000	128	-128	Negative
1000 0001	129	-127	
1000 0010	130	-126	
1111 1110	254	-2	
1111 1111	255	-1	

Source: https://en.wikipedia.org/wiki/Two%27s_complement

Two's Complement - Benefits

- One representation of zero
- Arithmetic works easily
- Negating is fairly easy

Ranges of Integer Representation

- **8-bit unsigned binary representation**

- Largest number: $1\ 1\ 1\ 1\ 1\ 1\ 1\ 1_2 = 255_{10}$
- Smallest number: $0\ 0\ 0\ 0\ 0\ 0\ 0\ 0_2 = 0_{10}$

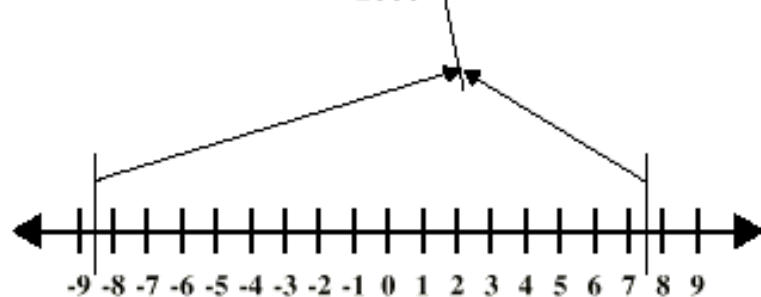
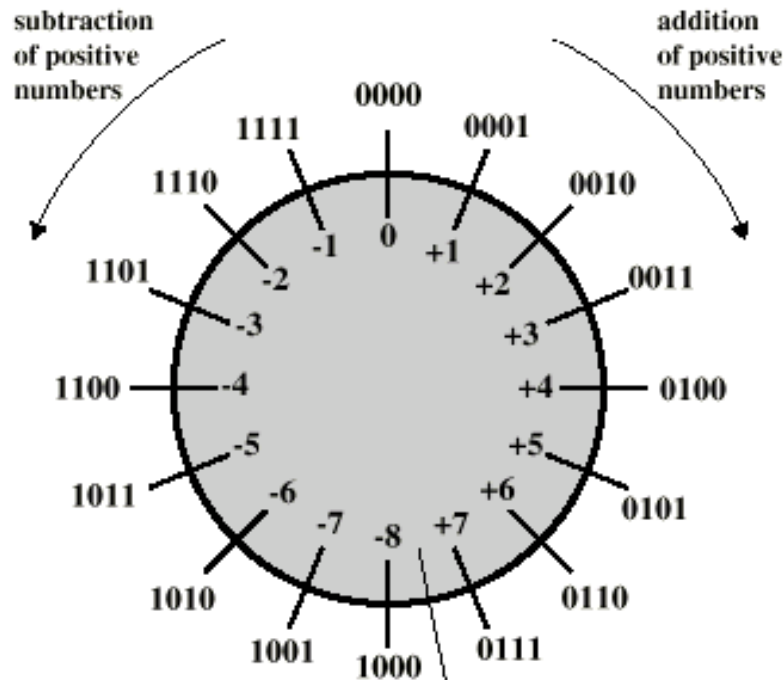
- **8-bit two's complement representation**

- Largest number: $0\ 1\ 1\ 1\ 1\ 1\ 1\ 1_2 = 127_{10}$
- Smallest number: $1\ 0\ 0\ 0\ 0\ 0\ 0\ 0_2 = -128_{10}$

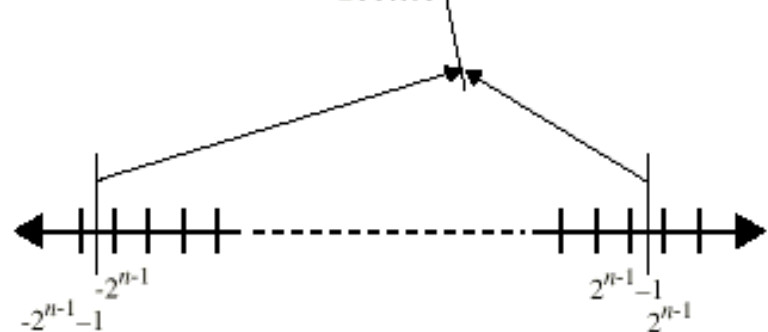
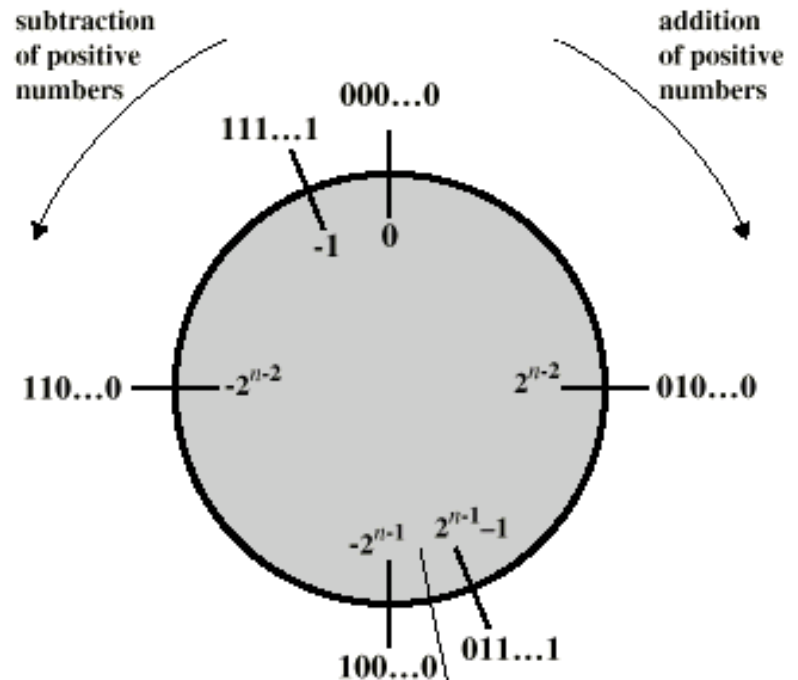
- The problem of overflow

- $130_{10} = 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0_2$
- $0\ 0\ 0\ 0\ 0\ 1\ 0_2$ in two's complement

Geometric Depiction of Two's Complement Integers



(a) 4-bit numbers



(b) n-bit numbers

Integer Data Types in C++

Type	Size in Bits	Range
unsigned int	16	0 – 65535
int	16	-32768 - 32767
unsigned long int	32	0 to 4,294,967,295
long int	32	-2,147,483,648 to 2,147,483,647

Activities

- Convert following two's complement numbers to Decimal representation.
 - 0010 1101
 - 1101 1010
 - 0101 0101
 - 1010 1100

Addition with Two's Complement

- Add 9_{10} to -23_{10}

9	0000 1001
-23	1110 1001
	<hr/>
?	<u>1111 0010</u>

- Check whether 1111 0010 represents expected -14_{10} in two's complement representation.

Addition with Two's Complement

- Add -9_{10} to 23_{10}

Carries	1	111	111
-9		1111	0111
23		0001	0111
?		<u>0000</u>	<u>1110</u>

- The carry going out (encircled) at the sign bit can be discarded as we are working only with 8 bits.
- Check whether 1111 0010 still represents the expected 14_{10} in two's complement representation.

Addition with Two's Complement

- Add -9_{10} to 23_{10}

Carries	1	111	111
-9		1111	0111
23		0001	0111
?		<u>0000</u>	<u>1110</u>

- We do not consider this as an overflow as it gives the correct answer.
- An overflow occurs if two positive numbers are added and the result is negative, or if two negative numbers are added and the result is positive.
- It is not possible to have overflow when using two's complement notation if a positive and a negative number are being added together.

Addition with Two's Complement

- Add -9_{10} to 23_{10}

Carries	1	1	1	1	1
-9		1	1	1	0
23		0	0	0	1
?		0	0	0	1

- Carry going into the sign bit (highlighted in green) is the same as the carry going out of the sign bit (highlighted in yellow).
- In such a scenario, no overflow occurs.
- When these carries are different, an overflow indicator is set in the arithmetic logic unit, indicating the result is incorrect.

Overflow in Addition

- Add 126_{10} to 23_{10}

Carries	0	1	1	1	1	1
126		0	1	1	1	1
23		0	0	0	1	0
?		1	0	0	1	0

- Carry going into the sign bit (highlighted in green) is 1 and the carry going out of the sign bit (highlighted in yellow) is 0.
- Addition of two positive numbers has resulted a negative number.
- Carries are different in here and an overflow has occurred.

Carry Vs. Overflow

- CPUs often have flags to indicate both carry and overflow.
- The overflow flag is used only with signed numbers.
- For unsigned numbers carry flag is sufficient to detect incorrect outcome after the arithmetic operation.
- However, carry flag is insufficient to detect incorrect outcome in signed number arithmetic operations.
- When carry going **into** the sign bit and the carry going **out** of the sign bit are different in signed number arithmetic, then that is an overflow.

Carry Vs. Overflow

Operation	Result	Carry ?	Overflow ?	Correct Result ?
0100 + 0010 ([+4] + [-2] in decimal)	0110 (+6 in decimal)	No	No	Yes
0100 + 0110 ([+4] + [+6] in decimal)	1010 (-6 in decimal)	No	Yes	No
1100 + 1110 ([-4] + [-2] in decimal)	1010 (-6 in decimal)	Yes	No	Yes
1100 + 1010 ([-4] + [-6] in decimal)	0110 (+6 in decimal)	Yes	Yes	No

Carry – indicates there is a carry out from the sign bit

Fractional Numbers in Decimal

- $16.357 =$ the SUM of ...

$$7 \times 10^{-3} = 7/1000$$

$$5 \times 10^{-2} = 5/100$$

$$3 \times 10^{-1} = 3/10$$

$$6 \times 10^0 = 6$$

$$1 \times 10^1 = 10$$

- $7/1000 + 5/100 + 3/10 + 6 + 10 = 16 \frac{357}{1000}$

Fractions in Binary

- $10.011 = \text{the SUM of ...}$

$$1 * 2^{-3} = 1/8$$

$$1 * 2^{-2} = 1/4$$

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

$$0 * 2^0 = 0$$

$$1 * 2^1 = 2$$

- $1/8 + 1/4 + 2 = 2\frac{3}{8}$
- $(10.011)_2 = 2\frac{3}{8}$ in Decimal (Base 10)

Activities

- What is Binary 011.0101 in Base 10?
- What is Binary 101.1101 in Base 10?
- What is Binary 111.111 in Base 10?
- What is Decimal $3\frac{5}{16}$ in Binary ?

Scientific Notation in Decimal

- Consider the following representation in decimal number ...

$$135.26 = .13526 \times 10^3$$

$$13526000 = .13526 \times 10^8$$

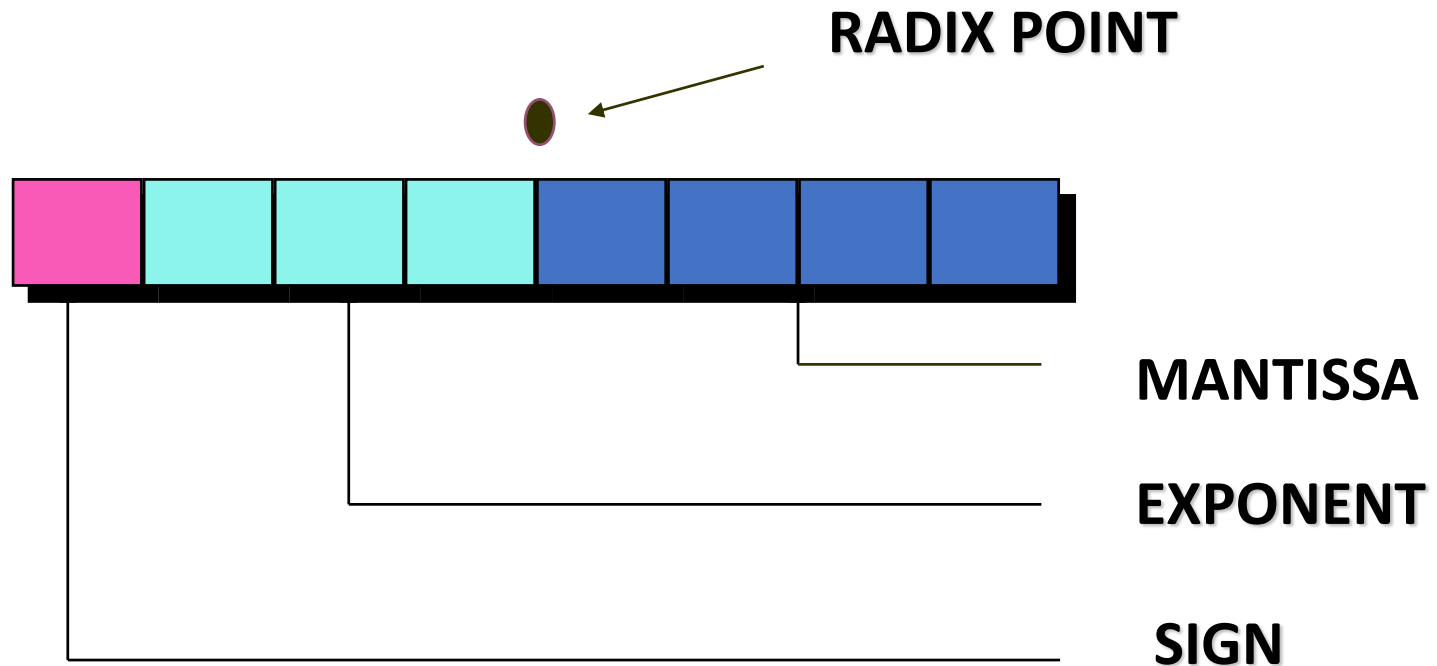
$$0.0000002452 = .2452 \times 10^{-6}$$

- $.13526 \times 10^3$ has the following components:
 - a Mantissa = .13526
 - an Exponent = 3
 - a Base = 10

Scientific Normalized Notation for Binary Numbers

- Scientific notation for binary. Examples ...
 - $11011.101 = 1.1011101 \times 2^4$
 - $-10110110000 = -1.011011 \times 2^{10}$
 - $0.00000010110111 = 1.0110111 \times 2^{-7}$

Floating Point Format in 1 Byte



SIGN = 0 (+ve) | 1 (-ve)

EXPONENT in EXCESS FOUR Notation

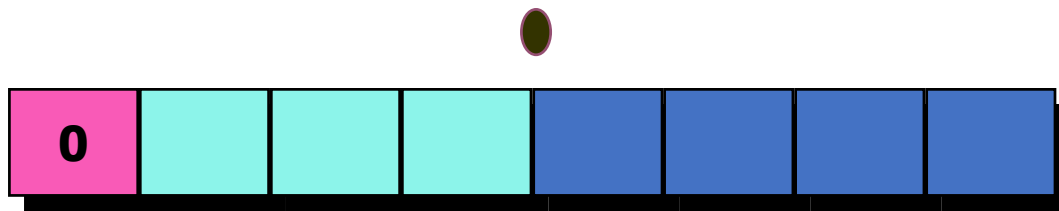
Floating Point Example

- To STORE the number ...

$$+1\frac{1}{8} = 1.001 \times 2^0$$

in FLOATING POINT NOTATION ...

1. STORE the SIGN BIT



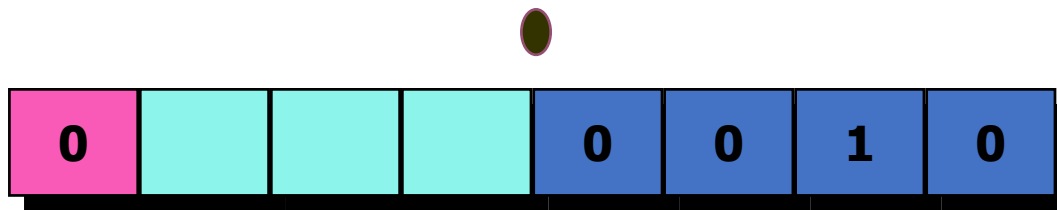
Floating Point Example

- To STORE the number ...

$$+1\frac{1}{8} = 1.001 \times 2^0$$

in FLOATING POINT NOTATION ...

2. STORE the MANTISSA BITS



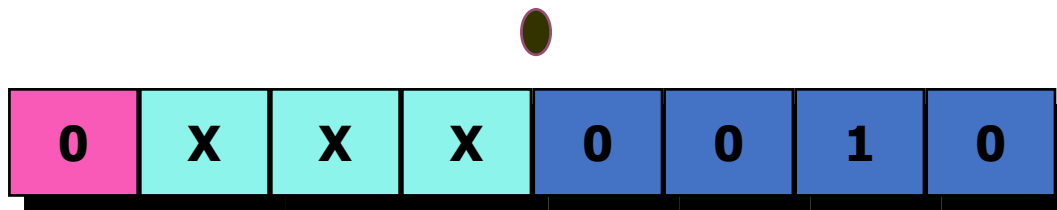
Floating Point Example

- To STORE the number ...

$$+1\frac{1}{8} = 1.001 \times 2^0$$

in FLOATING POINT NOTATION ...

3. STORE the EXPONENT BITS



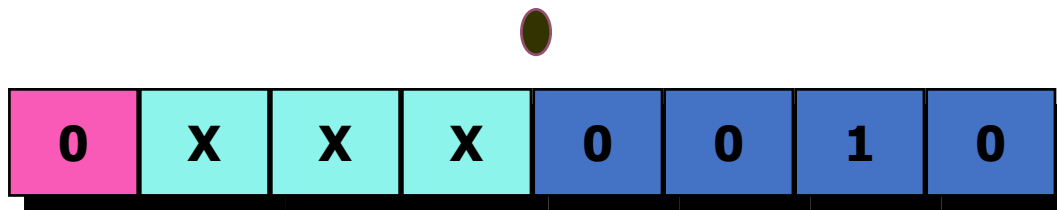
Floating Point Example

- To STORE the number ...

$$+1\frac{1}{8} = 1.001 \times 2^0$$

in FLOATING POINT NOTATION ...

3. STORE the EXPONENT BITS



Exponent is represented in **Biased** representation

Biased (Excess-k) Representation

- **EXCESS THREE NOTATION**

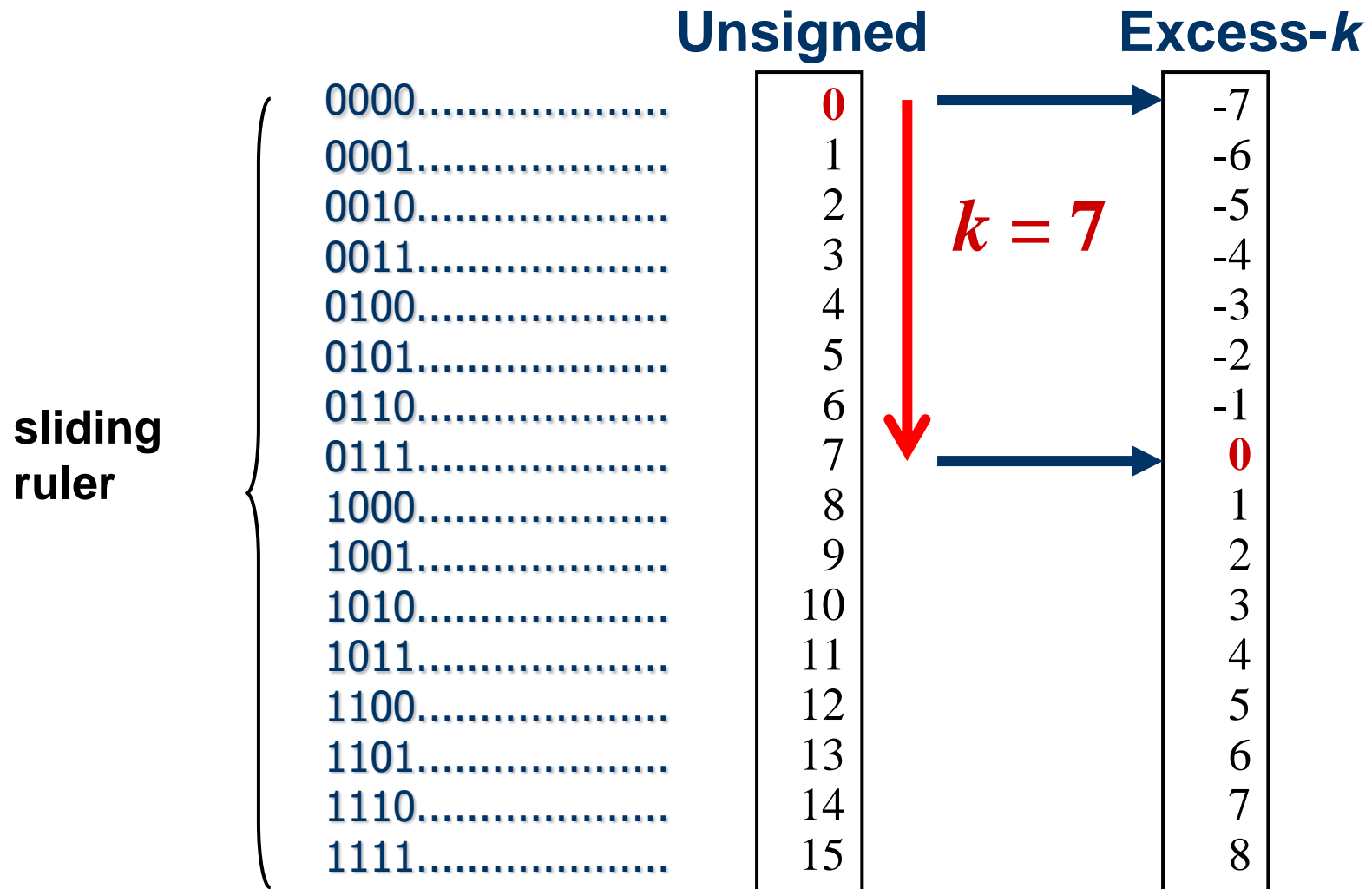
An excess notation system using bit pattern of length three

Bit Pattern	Value Representation
111	4
110	3
101	2
100	1
011	0
010	-1
001	-2
000	-3

Biased (Excess-k) Representation

- Usually, for **N** bit numbers, **k** is $(2^{N-1}-1)$
 - E.g. for 4-bit integers, **k** is 7
for 3-bit integers, **k** is 3
for 8-bit exponent, **k** is 127
- The actual value of each bit string is its unsigned value minus **k**
- To represent a number in **excess-k**, add **k**

Biased (Excess-k) Representation with 4 Bits



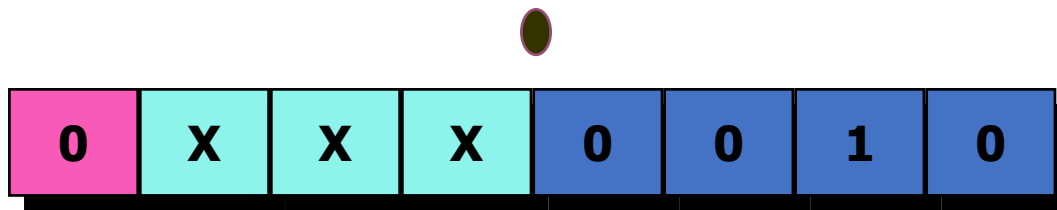
Floating Point Example - Exponent

- To STORE the number ...

$$+1\frac{1}{8} = 1.001 \times 2^0$$

in FLOATING POINT NOTATION ...

3. STORE the EXPONENT BITS



Let's bring Excess-3 representation for the Exponent

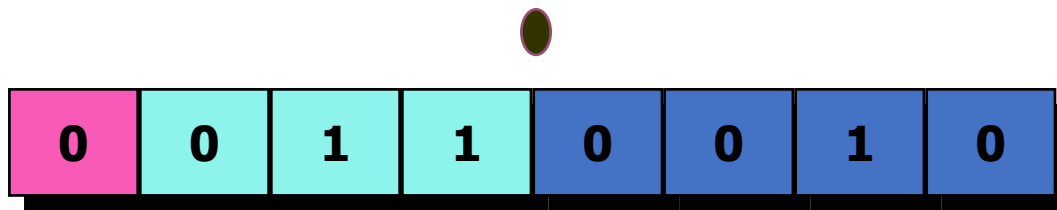
Floating Point Example - Exponent

- To STORE the number ...

$$+1\frac{1}{8} = 1.001 \times 2^0$$

in FLOATING POINT NOTATION ...

3. STORE the EXPONENT BITS



Unsigned 0 = 011 in Excess-3

Floating Point Example 2

- To STORE the number ...

$$-3\frac{1}{4} = -11.01$$

in FLOATING POINT NOTATION ...

- **First write the number in standard form**



Floating Point Example 2

- To STORE the number ...

$$-3\frac{1}{4} = -11.01 = -\mathbf{1.101} \times 2^1$$

in FLOATING POINT NOTATION ...

1. STORE the SIGN BIT



Next, Mantissa...

Floating Point Example 2

- To STORE the number ...

$$-3\frac{1}{4} = -11.01 = -\mathbf{1.101} \times 2^1$$

in FLOATING POINT NOTATION ...

2. STORE the MANTISSA BITS



Next, Exponent...

Floating Point Example 2

- To STORE the number ...

$$-3\frac{1}{4} = -11.01 = -\mathbf{1.101} \times 2^1$$

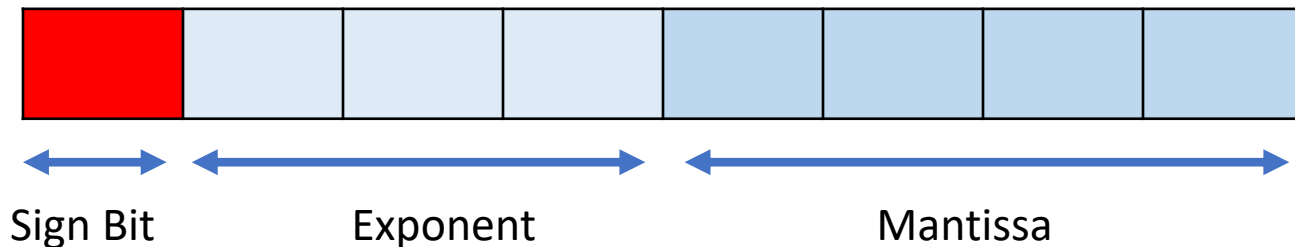
in FLOATING POINT NOTATION ...

2. STORE the EXPONENT BITS

1	1	0	0	1	0	1	0
---	---	---	---	---	---	---	---

Activities

- Try to represent following decimal floating-point numbers in binary representation given below.
 - 14.5
 - $11/64$
 - -14.5
 - $-1\frac{5}{8}$



Floating Point Binary to Decimal Conversion

1. Convert EXPONENT (EXCESS 4)
2. Apply EXPONENT to MANTISSA
3. Convert BINARY Fraction
4. Apply SIGN

Floating Point Binary to Decimal Conversion

- **Example** – 10111010 [Assume Exponent = 3 bits, Mantissa = 4 bits and Exponent in Excess-3]
- Sign = 1, Exponent = 011, Mantissa = 1010
- **STEP 1:** Convert Exponent
 - Excess-3 Value = 3
 - Real Value = 3 - 3 = 0
- **STEP 2:** Apply Exponent to Mantissa
 - 1.1010×2^0
- **STEP 3:** Convert Binary Fraction
 - $1 + \frac{1}{2} + \frac{1}{8} = 1\frac{5}{8}$
- **STEP 4:** Apply sign
 - $-1\frac{5}{8}$

IEEE Floating Point Standard

- IEEE has introduced standards for FP representations
- The standard is formally known as IEEE-754
- IEEE-754 Single Precision Floating Point

1 bit	8 bits	23 bits
Sign	Exponent (Bias 127)	Mantissa

- IEEE-754 Double Precision Floating Point

1 bit	11 bits	52 bits
Sign	Exponent (Bias 1023)	Mantissa

Round Off Error

- Try STORE $+2\frac{5}{16}$ in 8-bit Floating Point Representation

$$+2\frac{5}{16} = 10.0101 \text{ In Binary}$$



Round Off Error

- Try STORE $+2\frac{5}{16}$

$$+2\frac{5}{16} = 10.0101 = \mathbf{1.00101} \times 2^1$$

1. STORE the SIGN BIT

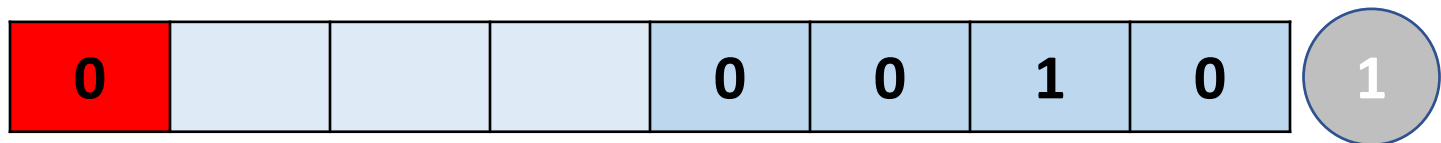


Round Off Error

- Try STORE $+2\frac{5}{16}$

$$+2\frac{5}{16} = 10.0101 = \mathbf{1.00101} \times 2^1$$

2. STORE the MANTISSA

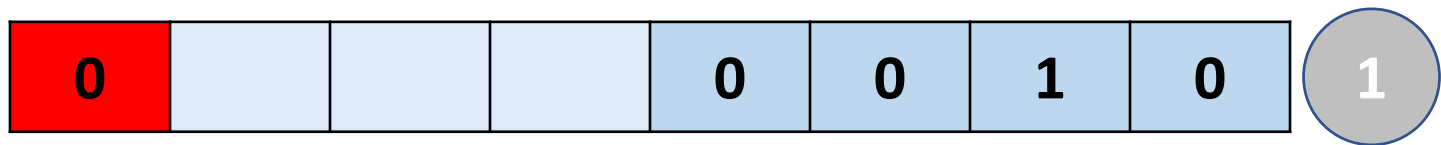


Round Off Error

- Try STORE $+2\frac{5}{16}$

$$+2\frac{5}{16} = 10.0101 = 1.00101 \times 2^1$$

2. STORE the MANTISSA



The last bit of Mantissa cannot be stored. DISCARD that bit.

Round Off Error

- Try STORE $+2\frac{5}{16}$

$$+2\frac{5}{16} = 10.0101 = \mathbf{1.00101} \times 2^1$$

3. STORE the EXPONENT

0	1	0	0	0	0	1	0
----------	---	---	---	---	---	---	---

Round Off Error

- Try STORE $+2\frac{5}{16}$

$$+2\frac{5}{16} = 10.0101 = 1.00101 \times 2^1$$

3. STORE the EXPONENT

0	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---

Conversion this number back to decimal will not give the original number ($+2\frac{5}{16}$)

Round Off Error

- Original Value = $2\frac{5}{16}$
- Stored Value = $01000010 = \mathbf{1.0010} \times 2^1$
 $= 10.01$
 $= 2\frac{1}{4}$
- Therefore, the Rounding Off Error in this case is
 $= 2\frac{5}{16} - 2\frac{1}{4}$
 $= \frac{1}{16} //$

Range of Floating Point Representations

In a floating point representation model,

- What is the Largest Positive number ?
- What is the Smallest Positive number ?
- What is the Largest Negative number ?
- What is the Smallest Negative number ?

Range: The Largest Positive Number

- What is the largest positive number ?
 - Sign bit should be 0
 - Mantissa should be all 1s
 - Exponent should be maximum (excess-k)

0	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

$$= 1.1111 \times 2^4$$

$$= 31$$

Range: The Smallest Positive Number

- What is the largest positive number ?
 - Sign bit should be 0
 - Mantissa should be all 0s
 - Exponent should be minimum (excess-k)



$$= 1.0000 \times 2^{-3} = 1/8$$

$$= \mathbf{0.125}$$

Range: The Smallest Negative Number

- What is the largest positive number ?
 - Sign bit should be 1
 - Mantissa should be all 0s
 - Exponent should be minimum (excess-k)

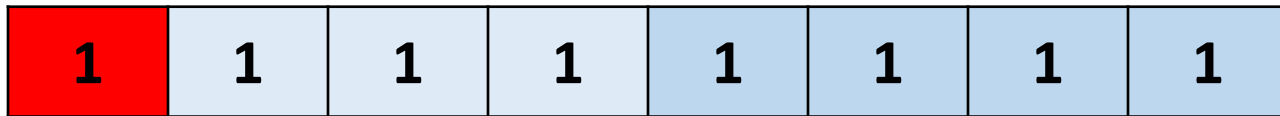
1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

$$= -1.0000 \times 2^{-3} = -1/8$$

$$= - \mathbf{0.125}$$

Range: The Largest Negative Number

- What is the largest positive number ?
 - Sign bit should be 1
 - Mantissa should be all 1s
 - Exponent should be maximum (excess-k)

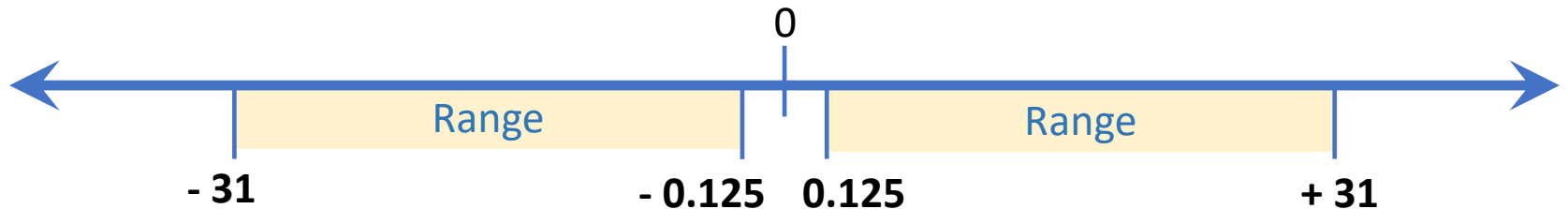


$$= -1.1111 \times 2^4$$

$$= -31$$

Range

- The range of 8 bit floating point representation that we used can be illustrated as below.



Challenge..!

- Try to find the ranges for IEEE-754 single and double precision representations. (Fill the tables)
- IEEE-754 Single Precision

Negative Min	Negative Max	Positive Min	Positive Max

- IEEE-754 Double Precision

Negative Min	Negative Max	Positive Min	Positive Max

Special Cases: Zero

- Zero cannot be represented with this format of FP as we assume there is always 1 immediate left of the radix point.
- Therefore, IEEE-754 defines all 0s in mantissa and exponent denote zero.
- Both cases are regarded as standard for 0

0	0000 0000	000 0000 0000 0000 0000 0000
1	0000 0000	000 0000 0000 0000 0000 0000

Special Cases: Infinity and NaN

- Infinity is denoted by all 1s in exponent with all 0s in mantissa.

+ Infinity	0	1111 1111	000 0000 0000 0000 0000 0000
- Infinity	1	1111 1111	000 0000 0000 0000 0000 0000

- Not a Number (NaN) is denoted by all 1s in exponent with any non-zero value in mantissa. Sign bit has no meaning in this situation.

1	1111 1111	Any Non-zero Value
---	-----------	--------------------

Special Cases: Denormalized Numbers

- IEEE-754 assumes that all FP numbers are normalized (there is always single digit at the immediate left to radix point and that digit is 1. Ex. 1.0101×2^{-3})
- However, IEEE-754 allows denormalized representation
- When exponent is all 0s and mantissa is non-zero, it is assumed the number is in denormalized form.
- Ex.

1	0000 0000	Any Non-zero Value
---	-----------	--------------------

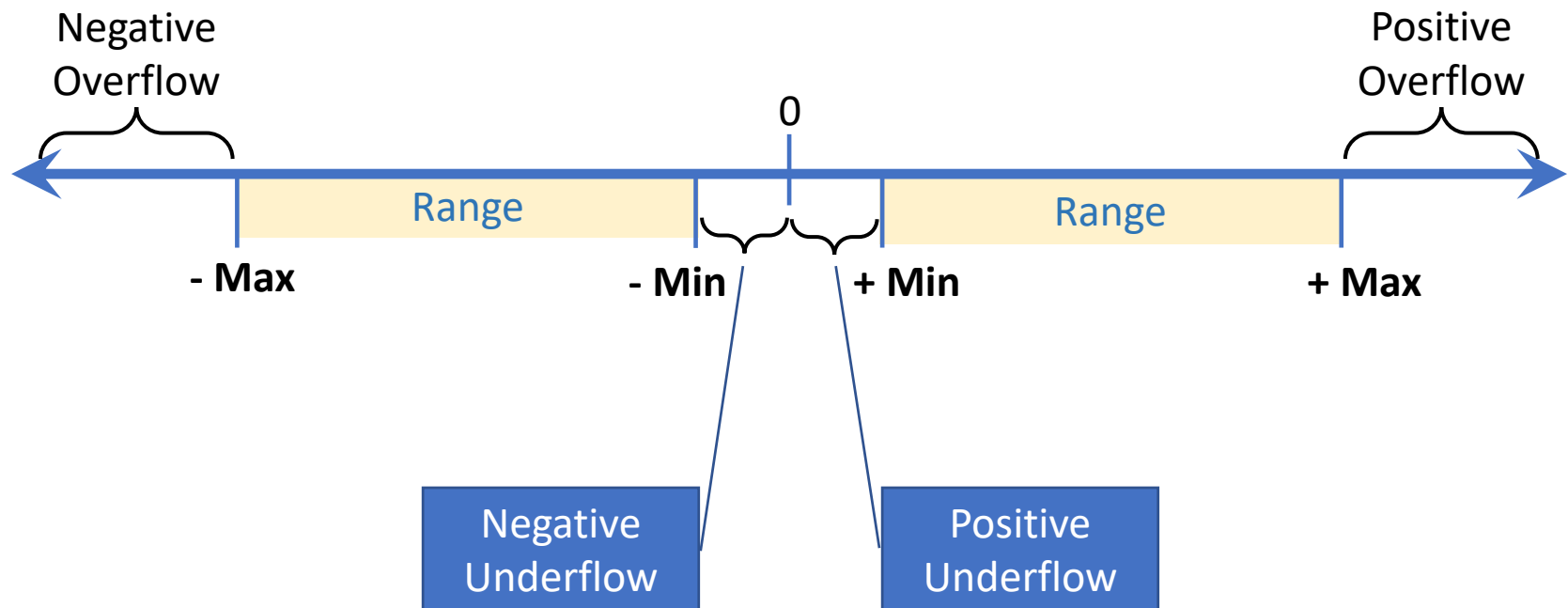
Here, exponent is in excess-127 form. Hence the true value of exponent is -127

Range for IEEE-754

- **Max:** $1.111\ 1111\ 1111\ 1111\ 1111\ 1111 \times 2^{127}$
 - Even though 2^{128} can be represented, it will make all 1s in exponent, which is the NaN
- **Min:** $0.000\ 0000\ 0000\ 0000\ 0001 \times 2^{-127}$
 - Since the exponent is -127 which is converted to all 0s with Excess-127 notation, the value represented will be regarded as denormalized number. Hence, there is no 1 left to the radix point.

Range for IEEE-754

- With the sign(+ or -), these **Min** and **Max** help to define the **-Min**, **-Max** and **+Min**, **+Max** respectively.



Problems with Floating Point Numbers

- Out of range: Overflow and Underflow
- Within the range: Rounding off error
- Floating point arithmetic is not always associative
 - Ex. There can be instances where three FP numbers a , b and c , that
 - $(a + b) + c \neq a + (b + c)$
- There can be instance where the FP arithmetic is not distributive.
 - Ex. $a \times (b + c) \neq ab + ac$

Character Codes

- Character encoding is the process of assigning numbers to graphical characters that gives the meaning to its representation.
- Character codes have evolved from basic coding systems such Binary Coded Decimal to Unicode
- Some of the Coding Systems
 - EBCDIC
 - ASCII
 - Unicode

EBCDIC

- EBCDIC stands for **Extended Binary Coded Decimal Interchange Code**.
- 8-bit Binary Coded Decimal representation.
- Characters are represented by appending digit bits to zone bits (see the figure in the next slide).
 - Character A = 1100 0001
 - Character a = 1000 0001

EBCDIC Code

Zone	Digit															
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL	SOH	STX	ETX	PF	HT	LC	DEL		RLF	SMM	VT	FF	CR	SR	SI
0001	DLE	DC1	DC2	TM	RES	NL	BS	IL	CAN	EM	CC	CU1	IFS	IGS	IRS	IUS
0010	DS	SOS	FS		BYP	LF	ETB	ESC			SM	CU2		ENQ	ACK	BEL
0011			SYN		PN	RS	UC	EOT				CU3	DC4	NAK		SUB
0100	SP										[.	<	(+	!
0101	&]	\$	*)	;	^
0110	-	/										,	%	_	>	?
0111										'	:	#	@	'	=	"
1000		a	b	c	d	e	f	g	h	i						
1001		j	k	l	m	n	o	p	q	r						
1010		~	s	t	u	v	w	x	y	z						
1011																
1100	{	A	B	C	D	E	F	G	H	I						
1101	}	J	K	L	M	N	O	P	Q	R						
1110	\		S	T	U	V	W	X	Y	Z						
1111	0	1	2	3	4	5	6	7	8	9						

EBCDIC Code Abbreviations

Abbreviations:

NUL	Null	TM	Tape mark	ETB	End of transmission block
SOH	Start of heading	RES	Restore	ESC	Escape
STX	Start of text	NL	New line	SM	Set mode
ETX	End of text	BS	Backspace	CU2	Customer use 2
PF	Punch off	IL	Idle	ENQ	Enquiry
HT	Horizontal tab	CAN	Cancel	ACK	Acknowledge
LC	Lowercase	EM	End of medium	BEL	Ring the bell (beep)
DEL	Delete	CC	Cursor Control	SYN	Synchronous idle
RLF	Reverse linefeed	CU1	Customer use 1	PN	Punch on
SMM	Start manual message	IFS	Interchange file separator	RS	Record separator
VT	Vertical tab	IGS	Interchange group separator	UC	Uppercase
FF	Form Feed	IRS	Interchange record separator	EOT	End of transmission
CR	Carriage return	IUS	Interchange unit separator	CU3	Customer use 3
SO	Shift out	DS	Digit select	DC4	Device control 4
SI	Shift in	SOS	Start of significance	NAK	Negative acknowledgement
DLE	Data link escape	FS	Field separator	SUB	Substitute
DC1	Device control 1	BYP	Bypass	SP	Space
DC2	Device control 2	LF	Line feed		

ASCII

- **American Standard Code for Information Interchange (ASCII)**
- Use bit patterns of length seven to represent
 - Letters of English alphabet: a - z and A - Z
 - Digits: 0 – 9
 - Punctuation symbols: (,), [,], {, }, ', ", !, /, \
 - Arithmetic Operation symbols: +, -, *, <, >, =
 - Special symbols: (space), %, \$, #, &, @, ^
- $2^7 = 128$ characters can be represented by ASCII

Character Representation: ASCII Table

Symbol	ASCII	Symbol	ASCII	Symbol	ASCII	Symbol	ASCII
(space)	00100000	A	01000001	a	01100001	0	00110000
!	00100001	B	01000010	b	01100010	1	00110001
“	00100010	C	01000011	c	01100011	2	00110010
#	00100011	D	01000100	d	01100100	3	00110011
\$	00100100	E	01000101	e	01100101	4	00110100
%	00100101	F	01000110	f	01100110	5	00110101
&	00100110	G	01000111	g	01100111	6	00110110
.....		

Character Representation: ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com

Character Representation: Unicode


- EBCDIC and ASCII are built around the Latin alphabet
 - Are restricted in their ability for representing non-Latin alphabet
 - Countries developed their own codes for native languages
- Unicode: 16-bit system that can encode the characters of most languages
- 16 bits = 2^{16} = 65,536 characters

Character Representation: Unicode

- The Java programming language and some operating systems now use Unicode as their default character code
- Unicode code space is divided into six parts
 - The first part is for Western alphabet codes, including English, Greek, and Russian
- Downward compatible with ASCII and Latin-1 character sets

Character Representation: Example

- English section of Unicode Table
 - ASCII equivalent of A is 41_{16}
 - Unicode is equivalent of A: $00\ 41_{16}$



	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074

Full chart list:

<http://www.unicode.org/charts/>

Sinhala Unicode

	0D8	0D9	0DA	0DB	0DC	0DD	0DE	0DF
0		භ 0D90	ච 0DA0	ධ 0DB0	ඳ 0DC0	ඹ 0DD0		
1	ඹ 0D81	ඵ 0D91	භ 0DA1	න 0DB1	ශ 0DC1	ඹ 0DD1		
2	ං 0D82	ඵ 0D92	භ 0DA2		භ 0DC2	ඹ 0DD2	ා 0DF2	
3	ං 0D83	ඵ 0D93	භ 0DA3	ඳ 0DB3	ස 0DC3	ඹ 0DD3	ා 0DF3	
4		ඹ 0D94	ඳ 0DA4	ප 0DB4	හ 0DC4	ඵ 0DD4		ඹ 0DF4
5	ඵ 0D85	ඹ 0D95	ඳ 0DA5	ඵ 0DB5	ඳ 0DC5			
6	ඵ 0D86	ඹ 0D96	ඳ 0DA6	ඵ 0DB6	ඳ 0DC6	ඵ 0DD6	ඵ 0DE6	
7	ඵ 0D87		ඵ 0DA7	හ 0DB7			ඵ 0DE7	

	0D8	0D9	0DA	0DB	0DC	0DD	0DE	0DF
8	ඵ 0D88		ඵ 0DA8	ඵ 0DB8		ඵ 0DD8	ඵ 0DE8	
9	ඵ 0D89		ඵ 0DA9	ඵ 0DB9		ඵ 0DD9	ඵ 0DE9	
A	ඵ 0D8A	ක 0D9A	ඵ 0DAa	ය 0DBA	ඵ 0DCA	ඵ 0DDA	ඵ 0DEA	
B	ඵ 0D8B	ඵ 0D9B	ඵ 0DAB	ර 0DBB		ඵ 0DDB	ඵ 0DEB	
C	ඵ 0D8C	ග 0D9C	ඵ 0DAC			ඵ 0DDC	ඵ 0DEC	
D	සා 0D8D	ස 0D9D	ක 0DAD	ඵ 0DBD		ඵ 0DDD	ඵ 0DED	
E	සා 0D8E	ඵ 0D9E	ඵ 0DAE			ඵ 0DDE	ඵ 0DEE	
F	ඵ 0D8F	ඵ 0D9F	ඵ 0DAF		ඵ 0DCF	ඵ 0DDF	ඵ 0DEF	

E.g. Unicode for Sinhala language's character ඵ is 0D85

Sinhala Unicode Chart: <https://unicode.org/charts/PDF/U0D80.pdf>

Tamil Unicode

	0B8	0B9	0BA	0BB	0BC	0BD	0BE	0BF
0		ஐ 0B90		ர 0BB0	ீ 0BC0	ஓ 0BD0		ய 0BF0
1				ற 0BB1	ு 0BC1			ள 0BF1
2	ஃ 0B82	ஒ 0B92		ல 0BB2	ூ 0BC2			த 0BF2
3	ஃ 0B83	ஒ 0B93	ண 0BA3	ள 0BB3				உ 0BF3
4		ஒள 0B94	த 0BA4	ழ 0BB4				ம் 0BF4
5	அ 0B85	க 0B95		வ 0BB5				ஸ் 0BF5
6	ஆ 0B86			ய 0BB6	ெ 0BC6		ஃ 0BE6	பு 0BF6
7	இ 0B87			ஷ 0BB7	ே 0BC7	ள 0BD7	க 0BE7	ஊ 0BF7

	0B8	0B9	0BA	0BB	0BC	0BD	0BE	0BF
8	ஈ 0B88		ந 0BA8	ஸ 0BB8	ை 0BC8		உ 0BE8	ஷ 0BF8
9	உ 0B89	ங 0B99	ன 0BA9	ஹ 0BB9			ங 0BE9	நு 0BF9
A	ஊ 0B8A	ச 0B9A	ப 0BAA		ொ 0BCA		சு 0BEA	நீ 0BFA
B					ோ 0BCB		நு 0BEB	
C		ஐ 0B9C			ெள 0BCC		சு 0BEC	
D					ஃ 0BCD		எ 0BED	
E	எ 0B8E	ஞ 0B9E	ம 0BAE	ா 0BBE			அ 0BEE	
F	ஏ 0B8F	ட 0B9F	ய 0BAF	ி 0BBF			கூ 0BEF	

E.g. Unicode for Tamil language's character அ is 0B85

Sinhala Unicode Chart: <https://unicode.org/charts/PDF/U0B80.pdf>

Thank you..!