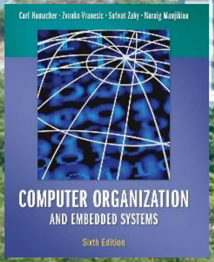
香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Lecture 02: Number and Character Representation

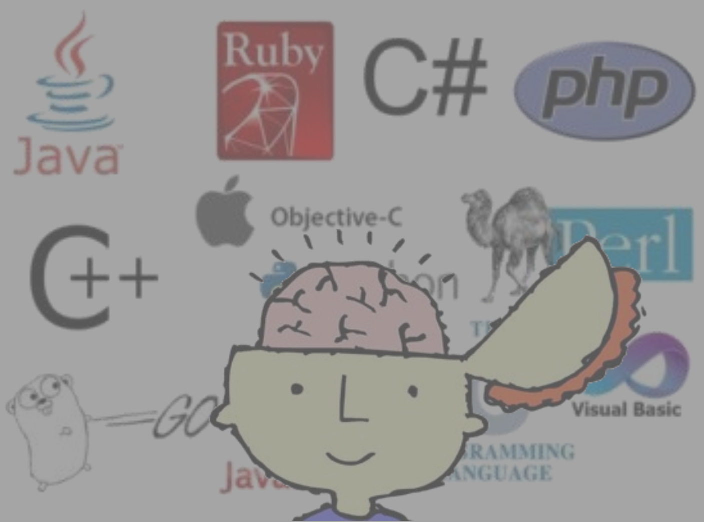
Ming-Chang YANG

mcyang@cse.cuhk.edu.hk



Reading: Chap. 1.4~1.5, 9.7~9.8

Recall: How to talk to the computer?



High-level Language

- Easy for programmer to understand
- Human understandable English words

→ **Language Translation** →

Machine Language

- The computer's own language
- Binary numbers (All 1s and 0s)

Outline

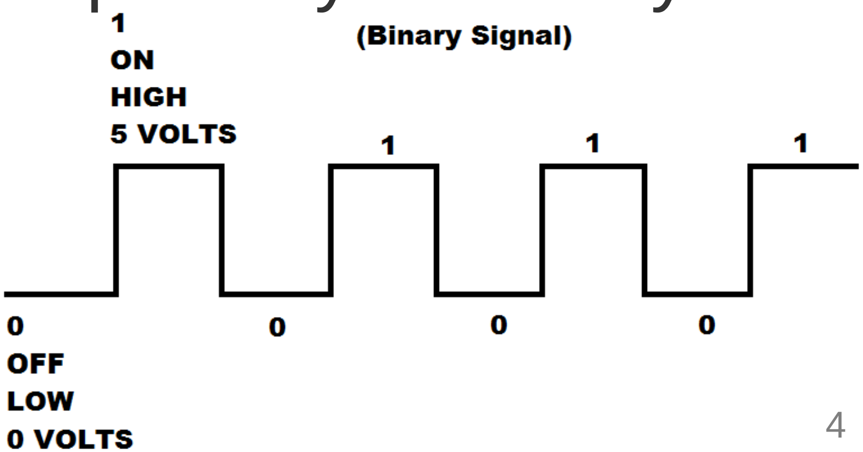
- Number Representation
  - Number Systems
  - Integers
    - Unsigned and Signed Integer
    - Arithmetic Operations
  - Floating-Point Numbers
    - Unsigned Binary Fraction
    - Floating-Point Number Representation
    - Arithmetic Operations
- Character Representation
  - ASCII

Number Systems

- Common number systems:
  - The *radix* or *base* of the number system denotes the number of digits used in the system.

Binary (*base* 2)	0 1
Octal (*base* 8)	0 1 2 3 4 5 6 7
Decimal (*base* 10)	0 1 2 3 4 5 6 7 8 9
Hexadecimal (*base* 16)	0 1 2 3 4 5 6 7 8 9 A B C D E F

- The most natural way in a computer system is by binary numbers (0, 1).
  - (0, 1) can be represented as (off, on) electrical signals.



https://social.technet.microsoft.com/wiki/contents/articles/22118.declaring-numeric-data-types.aspx

Conversion of Number Systems

Decimal	Binary	Octal	Hexadecimal
0 0	0 0 0 0	0 0	0
0 1	0 0 0 1	0 1	1
0 2	0 0 1 0	0 2	2
0 3	0 0 1 1	0 3	3
0 4	0 1 0 0	0 4	4
0 5	0 1 0 1	0 5	5
0 6	0 1 1 0	0 6	6
0 7	0 1 1 1	0 7	7
0 8	1 0 0 0	1 0	8
0 9	1 0 0 1	1 1	9
1 0	1 0 1 0	1 2	A
1 1	1 0 1 1	1 3	B
1 2	1 1 0 0	1 4	C
1 3	1 1 0 1	1 5	D
1 4	1 1 1 0	1 6	E
1 5	1 1 1 1	1 7	F
1 6 ? ? ?	1 0 0 0 0	2 0	1 0

Outline

- Number Representation
  - Number Systems
  - Integers
    - Unsigned and Signed Integer
    - Arithmetic Operations
  - Floating-Point Numbers
    - Unsigned Binary Fraction
    - Floating-Point Number Representation
    - Arithmetic Operations
- Character Representation
  - ASCII

Unsigned Integer Representation

- Consider an *n*-bit vector

$$B = b_{n-1} \dots b_1 b_0,$$

where $b_i = 0\ or\ 1$ (binary number) for $0 \le i \le n-1$

  – Most Significant Bit (MSB): b_{n-1} (i.e., the leftmost bit)
  – Least Significant Bit (LSB): b_0 (i.e., the rightmost bit)

- This vector can represent the value for an unsigned integer $\mathrm{V}(B)$ in the range 0 to $2^n - 1$, where

$$\mathrm{V}(B) = b_{n-1} \times 2^{n-1} + \cdots + b_1 \times 2^1 + b_0 \times 2^0$$

- For example, if $B = 1001$ *(n=4)*

$$\mathrm{V}(B) = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 9$$

Signed Integer Representation (1/3)

- To represent both positive and negative numbers, we need different systems to representing signed integer.
- In *written* decimal system, a signed integer is usually represented by a “+” or “–” sign and followed by the magnitude.
  - E.g. –73, –215, +349
- In binary system, we have three common choices:
  - **Sign-and-magnitude**
  - **1’s-complement**
  - **2’s-complement**

Signed Integer Representation (2/3)

- Positive values: MSB decides the sign (0: "+", 1: "–"), and the remaining bits represent an unsigned integer.
  - Positive values have identical representations in all systems.
- Negative values have different representations:
  - **Sign-and-magnitude** (MSB: sign, other bits: magnitude)
    - Negative values: changing the MSB from 0 to 1.
      - E.g. –3 is represented by 1011.

```
ex: 0011
     ↓
    1011
```

  - **1's-complement**
    - Negative values: inverting each bit of the positive number.
      - E.g. –3 is obtained by flipping each bit in 0011 to yield 1100.

```
ex: 0011
    ↓↓↓↓
    1100
```

  - **2's-complement**
    - Negative values: subtracting the positive number from 2^n *or* adding 1 to 1's-complement of that negative number.
      - E.g. –3 is obtained by adding 1 to 1100 to yield 1101.

```
ex:              ex:
    10000            1100
-)  0011         +)  0001
-------          -------
    1101             1101
```

Signed Integer Representation (3/3)

B	Values Represented		
$b_3b_2b_1b_0$	Sign-and-magnitude	1's-complement	2's-complement
0 1 1 1	+ 7	+ 7	+ 7
0 1 1 0	+ 6	+ 6	+ 6
0 1 0 1	+ 5	+ 5	+ 5
0 1 0 0	+ 4	+ 4	+ 4
0 0 1 1	+ 3	+ 3	+ 3
0 0 1 0	+ 2	+ 2	+ 2
0 0 0 1	+ 1	+ 1	+ 1
0 0 0 0	+ 0	+ 0	+ 0
1 0 0 0	− 0	− 7	− 8
1 0 0 1	− 1	− 6	− 7
1 0 1 0	− 2	− 5	− 6
1 0 1 1	− 3	− 4	− 5
1 1 0 0	− 4	− 3	− 4
1 1 0 1	− 5	− 2	− 3
1 1 1 0	− 6	− 1	− 2
1 1 1 1	− 7	− 0	− 1

Class Exercise 2.1

Student ID: ____________ Date:
Name: ________________ ________

- Question: Which representation system(s) uses distinct representations for $+0$ and -0?
- Answer: ______________________________

- Question: Which representation system(s) has only one representation for 0?
- Answer: ______________________________

- Question: Which representation system(s) is able to represent -8 for 4-bit numbers?
- Answer: ______________________________

Class Exercise 2.2

- Question: Consider the decimal number `-56`. Please use 8 bits to represent it in:
  - Sign-and-magnitude: ________________
  - 1's-complement: ________________
  - 2's-complement: ________________
- Question: Consider the 8-bit string `10110101`, what is its decimal value when interpreted as:
  - Sign-and-magnitude: ________________
  - 1's-complement: ________________
  - 2's-complement: ________________
- Question: Given n bits, what is the range of integers can be represented by the three representations?
- Answer: ____

Addition of Unsigned Integers

- Addition of 1-bit unsigned numbers:

```
    0        1        0        1
  + 0      + 0      + 1      + 1
  ---      ---      ---      ---
    0        1        1       10
```

carry-out sum

- To add multiple-bit numbers:
  - We add bit pairs starting <u>from the low-order (right) end</u>, propagating carries <u>toward the high-order (left) end</u>.
    - The carry-out from a bit pair becomes the carry-in to the next bit pair.
    - The carry-in must be added to a bit pair in generating the sum and carry-out at that position.
    - For example,

carry-in 1

```
    01111111
  + 00000001
  ----------
    10000000
```

high-order low-order

Arithmetic of Signed Integers

- The three signed integer representation systems differ only in the way of representing negative values.
- Their relative merits on performing arithmetic operations can be summarized as follows:
  - **Sign-and-magnitude**: the simplest representation, but it is also the most awkward for addition/subtraction operations.
  - **1's-complement**: somewhat better than the sign-and-magnitude system.
  - **2's-complement**: the most efficient method for performing addition and subtraction operations.
    - This is also why the 2's-complement system is the one most often used in modern computers.

Why 2's-complement Arithmetic?

- First consider adding $+7$ to -3:
  - What if we perform this addition by adding bit pairs from right to left (as what we did for n-bit unsigned numbers)?

$$
\begin{array}{rrrrr}
 & 0 & 1 & 1 & 1 \\
+ & 1 & 1 & 0 & 1 \\
\hline
1 & 0 & 1 & 0 & 0
\end{array}
$$

leftmost carry-out bit

  - If the leftmost carry-out bit is ignored, we get $(+4)_{10}$.
- Rules for *n*-bit signed number addition/subtraction:
  - $\boldsymbol{X+Y}$
    - Add their n-bit 2's-complement representations from right to left
    - Ignore the carry-out bit at the MSB position
  - $\boldsymbol{X-Y}$
    - Interpret as, and perform $X+(-Y)$
  - *Note: The sum should be in the range of* $-2^{n-1} \sim (2^{n-1}-1)$

Class Exercise 2.3

- Using 4-bit 2's-complement number to calculate:
  - **2+3**
  - **4+(-6)**
  - **(-5)+(-2)**
  - **2-4**
  - **(-7)-1**
  - **(-7)-(-5)**

Sign Extension for 2's-complement

- We often need to represent a value given in a certain number of bits by using a larger number of bits.

- *How to represent a signed number in 2's-complement form using a larger number of bits?*
- Sign Extension: Repeat the sign bit as many times as needed to the left.
  - Positive Number: Add 0's to the left-hand-side
    - E.g. 0111 → **0000**0111
  - Negative Number: Add 1's to the left-hand-side
    - E.g. 1010 → **1111**1010

Overflow in Integer Arithmetic

- In **Unsigned** Number Arithmetic:
  - A carry-out of 1 at MSB always indicates an overflow.
    - E.g. 1111 + 0001 = **1**0000
- In **2's-complement Signed** Number Arithmetic:
  - The value of the carry-out bit from the sign-bit position is NOT an indicator of overflow.
    - E.g. $(+7)_{10} + (+4)_{10} = (0111)_2 + (0100)_2 = (\mathbf{1}011)_2 = (-5)_{10}$
    - E.g. $(-4)_{10} + (-6)_{10} = (1100)_2 + (1010)_2 = (\mathbf{0}110)_2 = (+6)_{10}$
  - *How to detect the overflow in 2's-complement system?*
    - Addition of opposite sign numbers *never* causes overflow.
    - If the numbers are the same sign and the result is the opposite sign, an overflow has occurred.
      - E.g. $(+7)_{10} + (+4)_{10} = (\mathbf{0}111)_2 + (\mathbf{0}100)_2 = (\mathbf{1}011)_2 = (-5)_{10}$
      - E.g. $(-4)_{10} + (-6)_{10} = (\mathbf{1}100)_2 + (\mathbf{1}010)_2 = (\mathbf{0}110)_2 = (+6)_{10}$

Outline

- Number Representation
  - Number Systems
  - Integers
    - Unsigned and Signed Integer
    - Arithmetic Operations
  - Floating-Point Numbers
    - Unsigned Binary Fraction
    - Floating-Point Number Representation
    - Arithmetic Operations
- Character Representation
  - ASCII

Unsigned Binary Fraction

- Consider a *n*-bit unsigned binary fraction:

$$B = 0.b_{-1}b_{-2} \dots b_{-n}$$

where $b_{-i} = 0 \text{ or } 1$ (binary number) for $1 \leq i \leq n$

- This vector can represent the value for an unsigned binary fraction $\mathrm{F}(B)$, where

$$\mathrm{F}(B) = b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \cdots + b_{-n} \times 2^{-n}$$

- The range of $\mathrm{F}(B)$ is

Why? Geometric Series

$$S_n = \sum_{i=1}^{n} a_i r^{i-1} = a_1 \left(\frac{1 - r^n}{1 - r} \right)$$

$$0 \leq \mathrm{F}(B) \leq \underline{1 - 2^{-n}}$$

$$0 \leq \mathrm{F}(B) \approx +1.0, \textit{ for a large } n$$

Binary Fraction to Decimal Fraction

- What is the binary fraction 0.011010_2 in decimal ?

.	0	1	1	0	1
32	16	8	4	2	1

8+4+1=13

- Answer: 13 / 32 = **0.40625**

Decimal Fraction to Binary Fraction

- What is the decimal fraction 0.6875_{10} in binary ?

$0.6875 * 2 = \mathbf{1}.\underline{3750} \rightarrow 0.\mathbf{1}???_2$

$0.\underline{3750} * 2 = \mathbf{0}.\underline{7500} \rightarrow 0.1\mathbf{0}??_2$

$0.\underline{7500} * 2 = \mathbf{1}.\underline{5000} \rightarrow 0.10\mathbf{1}?_2$

$0.\underline{5000} * 2 = \mathbf{1}.\underline{0000} \rightarrow 0.101\mathbf{1}_2$

$0.\underline{0000} * 2 = 0 \rightarrow$ **End**

- Answer: $\mathbf{0.1011_2}$

$0.6875 * 10 = \mathbf{6}.\underline{875} \rightarrow 0.\mathbf{6}???_{10}$

$0.\underline{8750} * 10 = \mathbf{8}.\underline{7500} \rightarrow 0.6\mathbf{8}??_{10}$

...

Class Exercise 2.4

- What is the decimal fraction 0.1_{10} in binary ?
- Answer:

What did we learn so far?

- Some decimal fractions (e.g. 0.1_{10}) will produce infinite binary fraction expansions.
- The position of the binary point in a floating-point number varies (that's way called floating point!).

$$
\begin{aligned}
0.232 * 10^4 &= 2.320000 * 10^3 \\
&= 23.20000 * 10^2 \\
&\ldots
\end{aligned}
$$

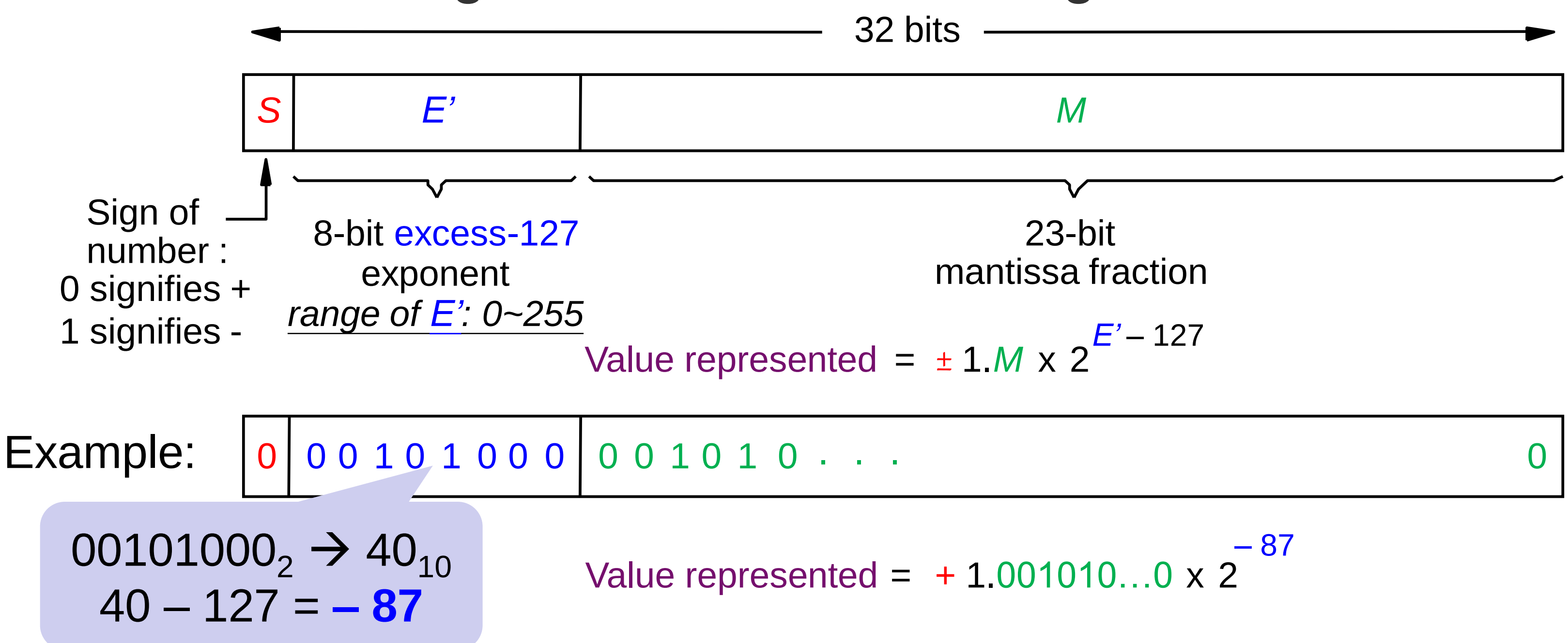
- A 32-bit signed integer in 2's-complement form can only represent values in the range of $-2^{31} \sim 2^{31} - 1$.
- We need a unique representation that can
  - Explicitly indicate the position of the binary point.
  - Represent both very large integers and very small fractions.

Floating Point Number Representation

- In decimal scientific notation, numbers are written as :
  $+6.0247 \times 10^{23}, +3.7291 \times 10^{-27}, -7.3000 \times 10^{-14}, \ldots$
- The same approach can be used to represent binary floating-point numbers (using 2 as the base) by:
  - Sign: A sign for the number
  - Mantissa: Some significant bits
  - Exponent: A signed scale factor (implied base of 2)
- To have a normalized representation for floating-point numbers, we should normalize Mantissa in the range $[1 \ldots B)$, where B is the base.
  - Binary System: $[1 \ldots 2)$
    - $(1.b_{-1}b_{-2} \ldots b_{-n})_2$ must in the range of $[1 \ldots 2)$.

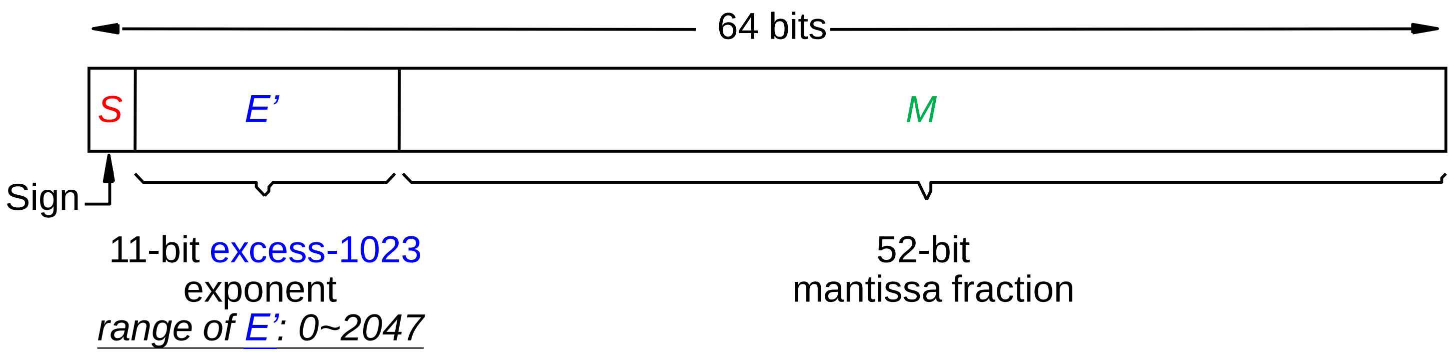
IEEE Standard 754 Single Precision

- The **single** precision format is a 32-bit representation.
  - The leftmost bit represents the sign, S, for the number
  - The next 8 bits, E’, represent the **unsigned integer** for the *excess-127 exponent* (with base of 2)
    - The actual **signed** exponent E is E’–127
  - The remaining 23 bits, M, are the significant bits



IEEE Standard 754 Double Precision

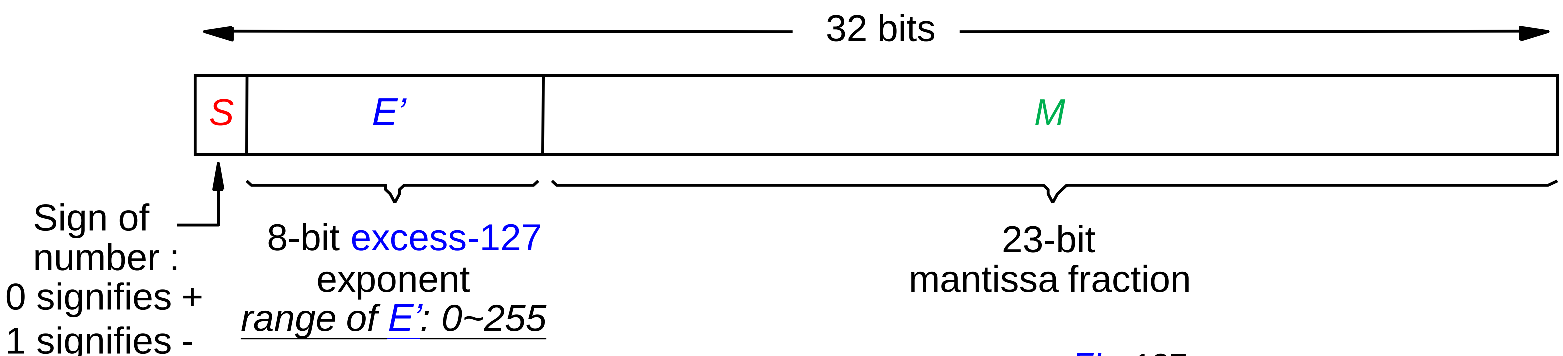
- The **double** precision format is a 64-bit representation.
  - The leftmost bit represents the sign, S, for the number
  - The next 11 bits, E', represent the **unsigned integer** for the *excess-1023 exponent* (with base of 2)
    - The actual **signed** exponent E is E'–1023
  - The remaining 52 bits, M, are the significant bits



Value represented $= \pm 1.M \times 2^{E' - 1023}$

Example of IEEE Single Precision

- What is the IEEE single precision number `40C0 0000`$_{16}$ in decimal?
- Answer:



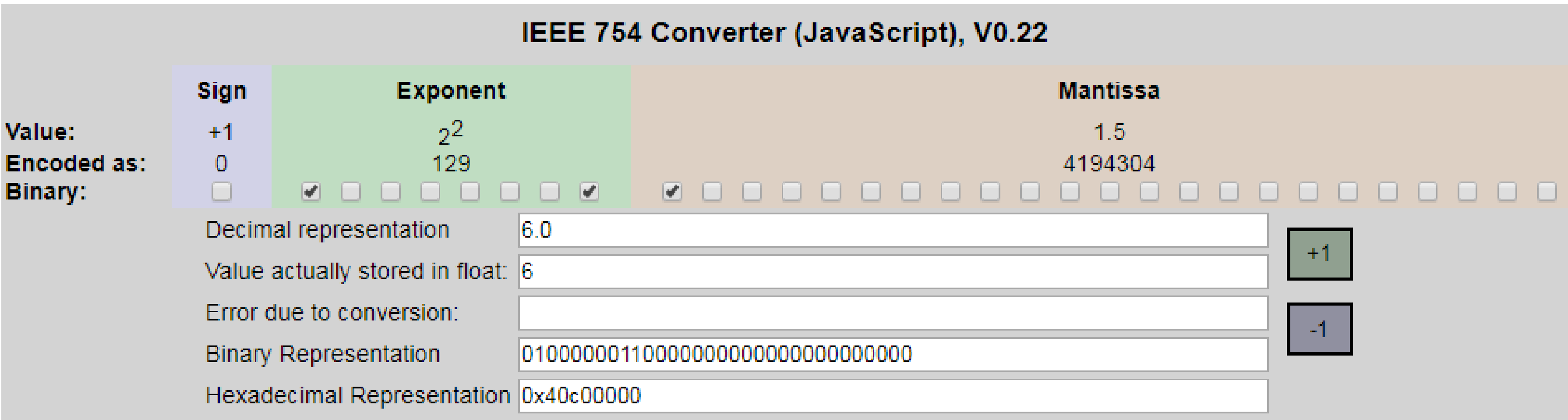
Value represented $= \pm 1.M \times 2^{E' - 127}$

– Format: 0100 0000 1100 0000 0000 0000 0000 0000
  - Sign: +
  - Exponent: 129 – 127 = +2
  - Mantissa: $100\ 0000\ldots_2$

– Decimal Value: $+1.100\ 0000\ldots_2 \times 2^{+2} = 1.5_{10} \times 2^{+2} = +6.0_{10}$

Useful Tool

- IEEE-754 Floating Point Converter
  – https://www.h-schmidt.net/FloatConverter/IEEE754.html

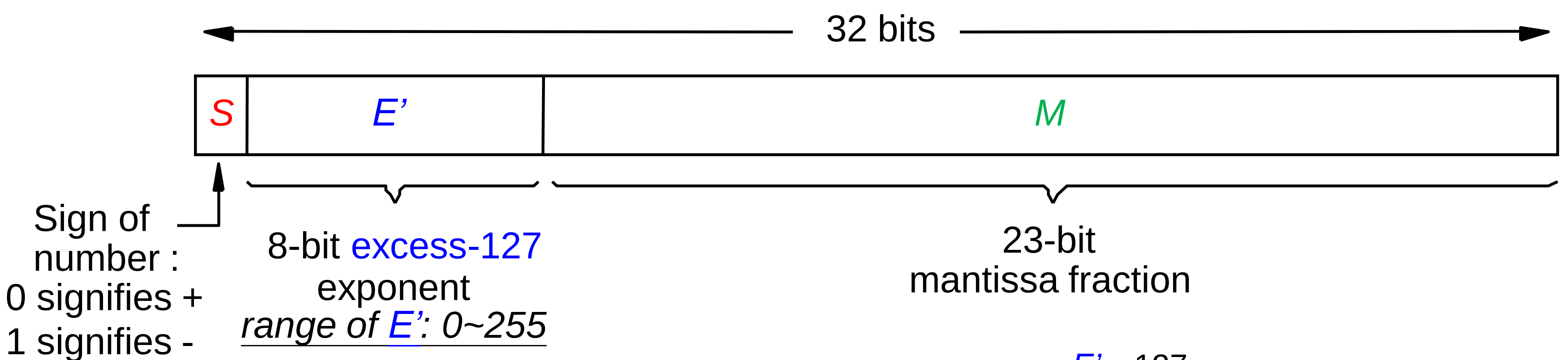


Class Exercise 2.5

Student ID: ____________ Date:
Name: __________________ ________

- What is -0.5_{10} in the IEEE single precision binary floating point format?
- Answer:

Special Values



Value represented = $\pm 1.M \times 2^{E' - 127}$

- When exponent $E' = 0$ (all 0's) and mantissa $M = 0$:
  - The value 0 is represented.
- When exponent $E' = 0$ (all 0's) and mantissa $M \neq 0$:
  - $\mathit{Denormal\ values}$ (i.e. very small values) are represented.
- When exponent $E' = 255$ (all 1's) and mantissa $M = 0$:
  - The value ∞ is presented.
- When exponent $E' = 255$ (all 1's) and mantissa $M \neq 0$:
  - $\mathit{Not\ a\ Number\ (NaN)}$ (e.g. $0/0$ or $\sqrt{-1}$) is presented.

Arithmetic on Floating-Point Number (1/2)

- When adding/subtracting floating-point numbers, their mantissas must be shifted with respect to each other.
  - E.g. adding $2.9400_{10} \times 10^2$ to $4.3100_{10} \times 10^4$
    - We rewrite 2.9400×10^2 as 0.0294×10^4
    - Then perform addition of the mantissas to get 4.3394×10^4.
- Add/Subtract Rule
  1) Choose the number with the smaller exponent and shift its mantissa right a number of steps equal to the difference in exponents.
  2) Set the exponent of the result equal to the larger exponent.
  3) Perform addition/subtraction on the mantissas and determine the sign of the result.
  4) Normalize the resulting value, if necessary.

Arithmetic on Floating-Point Number (2/2)

- Multiplication and division are somewhat easier than addition and subtraction.
  - No alignment of mantissas is needed.
- Multiply Rule
  1) Add the exponents and subtract 127 to maintain the excess-127 representation.
  2) Multiply the mantissas and determine the sign of the result.
  3) Normalize the resulting value, if necessary.
- Divide Rule
  1) Subtract the exponents and add 127 to maintain the excess-127 representation.
  2) Divide the mantissas and determine the sign of the result.
  3) Normalize the resulting value, if necessary.

Outline

- Number Representation
  - Number Systems
  - Integers
    - Unsigned and Signed Integer
    - Arithmetic Operations
  - Floating-Point Numbers
    - Unsigned Binary Fraction
    - Floating-Point Number Representation
    - Arithmetic Operations
- Character Representation
  - ASCII

Character Representation

- The most common encoding scheme for characters is ASCII (American Standard Code for Information Interchange).

- In ASCII encoding scheme, alphanumeric characters, operators, punctuation symbols, and control characters can be represented by 7-bit codes.
  – It is convenient to use an 8-bit *byte* to represent a character.
    - The code occupies the low-order 7 bits with the high-order bit as 0.

ASCII Table

Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(	72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29	)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[	123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	|
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D	]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]

Class Exercise 2.6

- Represent “`Hello, CSCI2510`” using ASCII code:

	Decimal	Binary
`H`		
`e`		
`l`		
`l`		
`o`		
`,`		
`C`		
`S`		
`C`		
`I`		
`2`		
`5`		
`1`		
`0`		

Summary

- Number Representation
  - Number Systems
  - Integers
    - Unsigned and Signed Integer
    - Arithmetic Operations
  - Floating-Point Numbers
    - Unsigned Binary Fraction
    - Floating-Point Number Representation
    - Arithmetic Operations
- Character Representation
  - ASCII