

Number Systems and Number Representation

EE1001 (part A)

Tommy WS Chow

EE CityU

July 2020

Index: Recap your High School maths

$$\square (y^2)(y^5) = y^{2+5} = y^7$$

$$\square (x^{1/2}) = \sqrt{x}, \quad x^{-2} = \frac{1}{x^2}$$

$$\square (y^a)^3 = (y^a)(y^a)(y^a)^3 = y^{3a}$$

$$\square (y^{2a})^3 = (y^{2a})(y^{2a})(y^{2a}) = y^{6a}$$

$$\square (y^m)(y^k) = y^{(m+k)}$$

$$\square (y^m)(x^{-k}) = \frac{y^m}{x^k}$$

Exponents

$$4 = (4)(10^0)$$

$$4.0 = (4)(10^0)$$

$$40 = (4)(10^1)$$

$$0.4 = (4)(10^{-1})$$

$$400 = (4)(10^2)$$

$$0.04 = (4)(10^{-2})$$

$$4000 = (4)(10^3)$$

$$0.004 = (4)(10^{-3})$$

$$40000 = (4)(10^4)$$

$$0.0004 = (4)(10^{-4})$$

- We will use this fundamental concept for floating point representation of numbers.
- Floating point is an important concept for computer science and computer engineering.

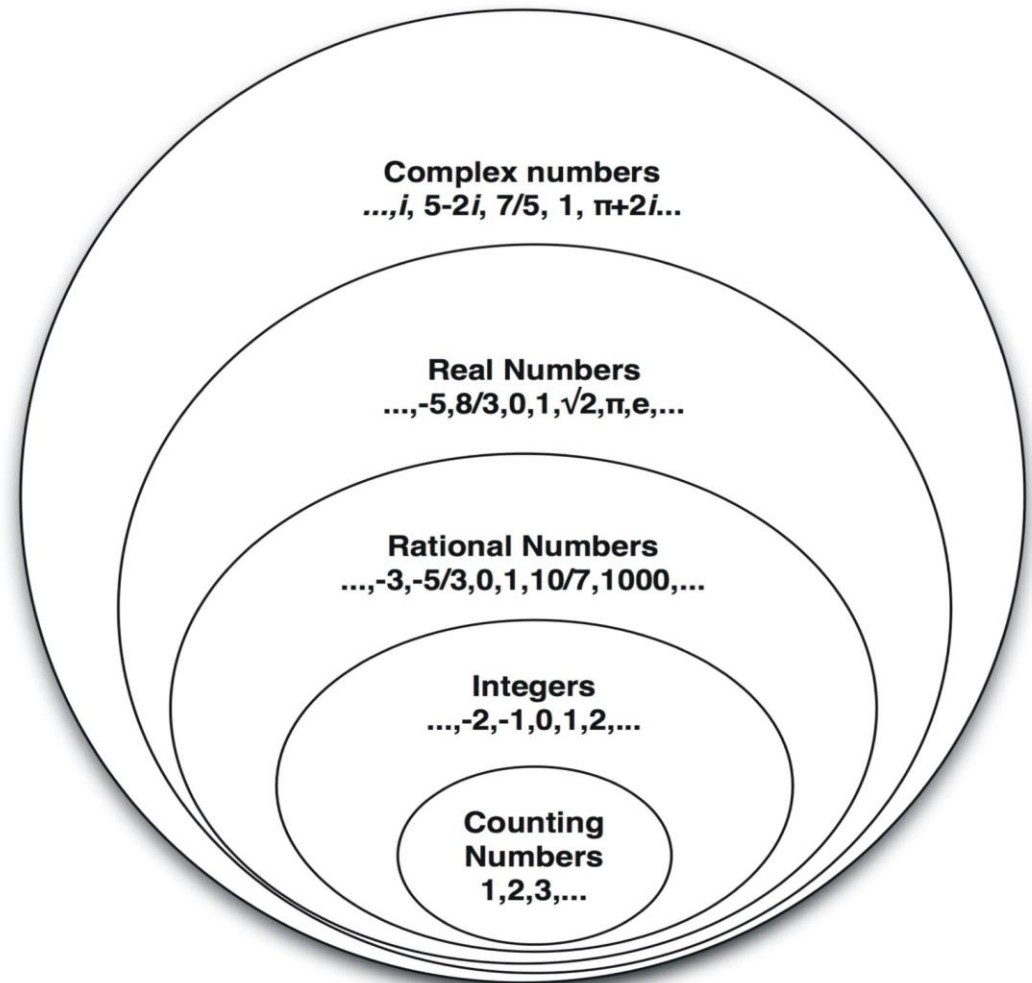
- ❑ 1.1 Number Systems
- ❑ 1.2 Signed and Unsigned Binary Numbers
- ❑ 1.3 1's Complement
- ❑ 1.4 2's Complement
- ❑ 1.5 Hexadecimal Number
- ❑ 1.6 Binary Arithmetic
- ❑ 1.7 Fixed-Point Representation
- ❑ 1.8 Floating-Point Representation
- ❑ 1.9 Floating-Point Arithmetic

Number Systems

Types of Numbers

- ❑ Natural Numbers
- ❑ Integers
- ❑ Rational Numbers
- ❑ Real Numbers
- ❑ Complex number

$$-1 = i^2$$



Can you give an example of an irrational number?

Natural Numbers

- ❑ The most basic set of numbers.
- ❑ The set of all positive integers,
 - 0 is also considered as a natural number by many.
- ❑ The numbers used to count objects.
- ❑ The earliest number set in existence.
- ❑ The capital letter N ,
 - The usual denotation of the set of natural numbers.
- ❑ Example of natural numbers:
 - $\{0, 1, 2, 3, \dots\}$

Integers

- ❑ The set of integers \mathbf{Z} ,
 - Take the set of natural numbers then add to it the negatives of all natural numbers.
- ❑ While $+/−$ and \times are defined on \mathbf{Z} , \div is not,
 - Dividing one integer by another will not yield a third integer as the result. (You are really performing the operation over the set of rational numbers or real numbers, rather than over integers.)
- ❑ Example of integers:
 - $\{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$

Rational Numbers

- ❑ The set of rational numbers \mathbf{Q} (stands for quotient),
 - Expand the set of integers by adding to it all the quotients (ratios) of integers.
 - The set of numbers of the form $\frac{a}{b}$, where \mathbf{a} and \mathbf{b} are integers (and $\mathbf{b} \neq \mathbf{0}$, of course).
- ❑ Division (\div) is defined on rational numbers
 - Any rational number divided by any nonzero rational yields another rational number.
- ❑ Example of rational number:
 - $\{-0.1\}$, as a fraction $\{-\frac{1}{10}\}$.

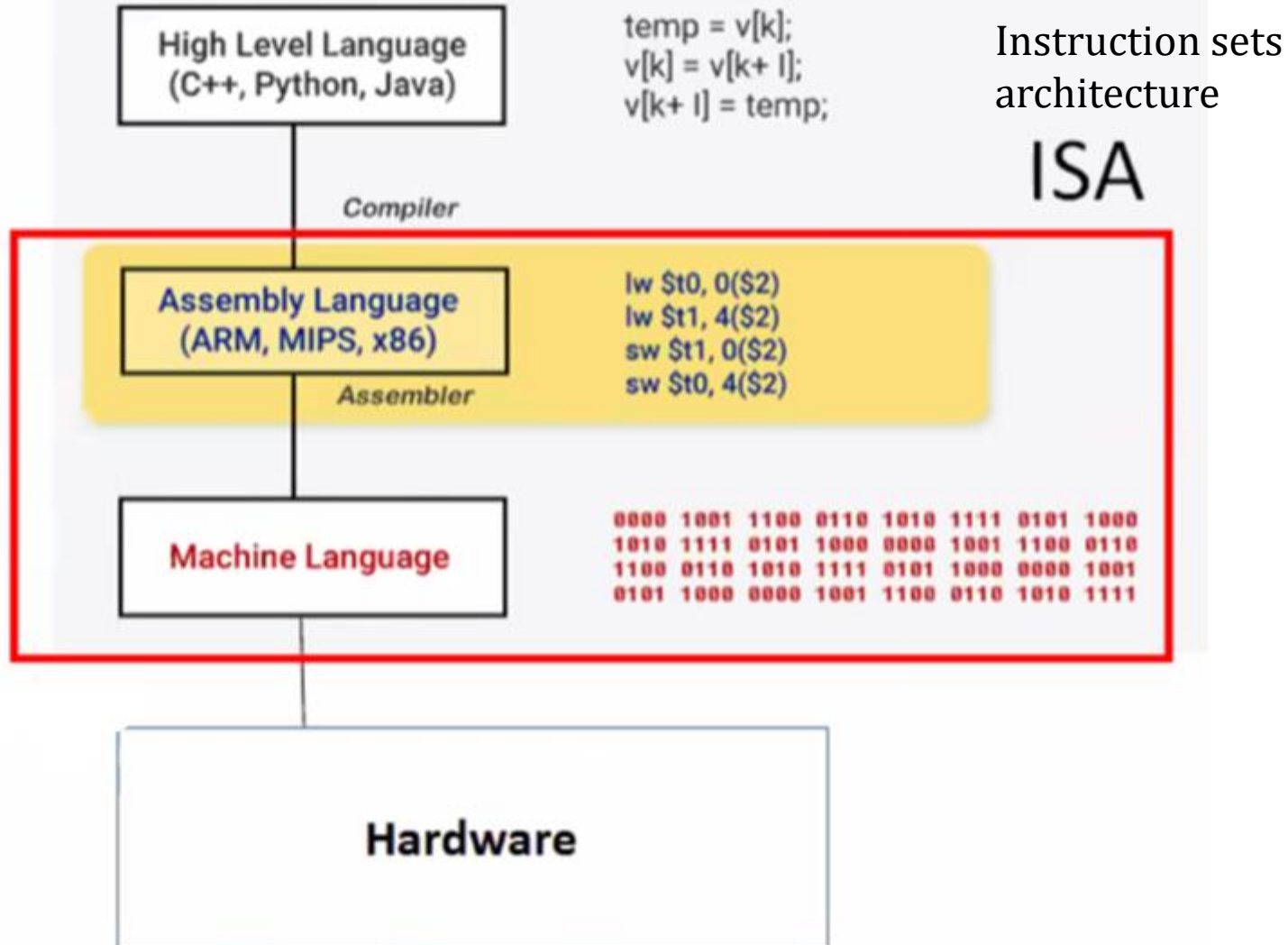
9/8/2020

Real Numbers

- ❑ A subset of the complex numbers.
- ❑ A real number r is just a complex number without the imaginary part, $r + 0i$.
- ❑ The set of all real numbers is usually denoted by the capital letter **R** .
- ❑ Example of real numbers :
 - $\{1, -10, -0.67, \frac{4}{5}, \pi, \sqrt{2}, 187\}$

Signed and Unsigned Binary Numbers

Levels of Program Code



Representing number

$$345.865 = 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0 + 8 \times 10^{-1} + 6 \times 10^{-2} + 5 \times 10^{-3}$$

- First separate the number into two integers: 345 (before decimal place) and 865 after decimal place.
- We will next divide 345 by 2 to obtain 172 with a remainder of 1 (172.5 is 172+1/2). This indicates that the least significant bit is one
- This process is repeated until the integer goes to zero.

Binary Number Representation

Successive Division by 2

$$\begin{array}{r}
 2 \overline{) 29} \\
 2 \overline{) 14} \\
 2 \overline{) 7} \\
 2 \overline{) 3} \\
 2 \overline{) 1} \\
 0
 \end{array}$$

Remainders

1 LSB
0
1
1
1 MSB

Read the remainders
from the bottom up

$$29 = 11101$$

MSB, Most
significant bit

LSB, Least
significant bit

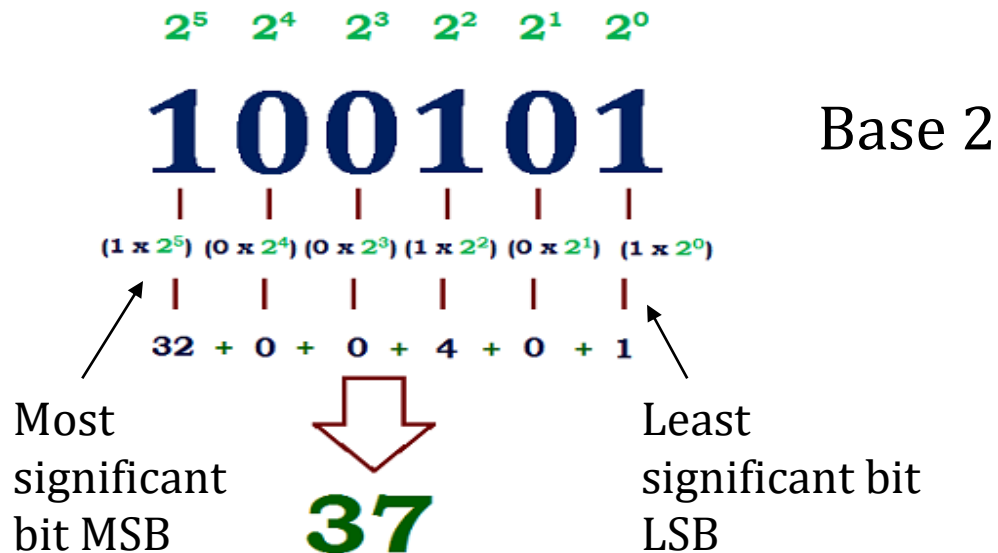
$$\begin{array}{cccccccc}
 & & & & & 0.5 & 0.25 & 0.125 & 0.0625 \\
 & & & & & 1/2 & 1/4 & 1/8 & 1/16 \\
 16 & 8 & 4 & 2 & 1 & & & & \\
 29.625 = & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0
 \end{array}$$

- We have difficulty to use binary to represent 29.779 in an exact way.
- We may need many more bits to approximate the “.779 “exactly or just accurately.
- .11000111011011001001₁₋₁₄

Binary Numbers Representation

Computer can understand only (0, 1) language.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	1	0	0	0	1	0	0
<div>←</div>							
128 + 64 + 0 + 0 + 0 + 4 + 0 + 0							



Binary Numbers Representation (cont)

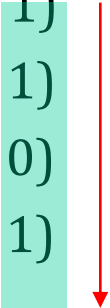
- ❑ First $345/2 = 172$ (remainder 1) — Least Significant Bit (LSB)
- ❑ Next $172/2 = 86$ (remainder 0)
- ❑ Then $86/2 = 43$ (remainder 0)
- ❑ Then $43/2 = 21$ (remainder 1)
- ❑ Then $21/2 = 10$ (remainder 1)
- ❑ Then $10/2 = 5$ (remainder 0)
- ❑ Then $5/2 = 2$ (remainder 1)
- ❑ Then $2/2 = 1$ (remainder 0)
- ❑ Then $1/2 = 0$ (remainder 1) — Most Significant Bit (MSB)
- ❑ End.

- ❑ This will lead to a binary number [101011001]
MSB..... .LSB

- ❑

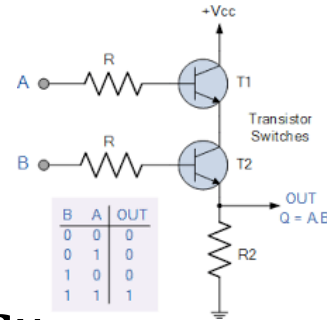
	256	128	64	32	16	8	4	2	1	
	1	0	1	0	1	1	0	0	1	
										= 1+8+16++64+256 = 345

Representing fractional number

- ❑ In the case of the portion of the number to the right of the decimal place we would perform a multiplication process with the most significant bit coming first.
 - ❑ First $0.865 \times 2 = 1.730$ (1st digit left of decimal is 1)
 - ❑ Next $0.730 \times 2 = 1.460$ (1st digit left of decimal is 1)
 - ❑ Then $0.460 \times 2 = 0.920$ (1st digit left of decimal is 0)
 - ❑ Then $0.920 \times 2 = 1.840$ (1st digit left of decimal is 1)
- 
- ❑ Note that if the term on the right of the decimal place does not easily divide into base 2, the term to the right of the decimal place could require a large number of bits. Typically the result is truncated to a fixed number of decimals.
 - ❑ The binary equivalent of $0.865 \cong .1101$ (approximate only)

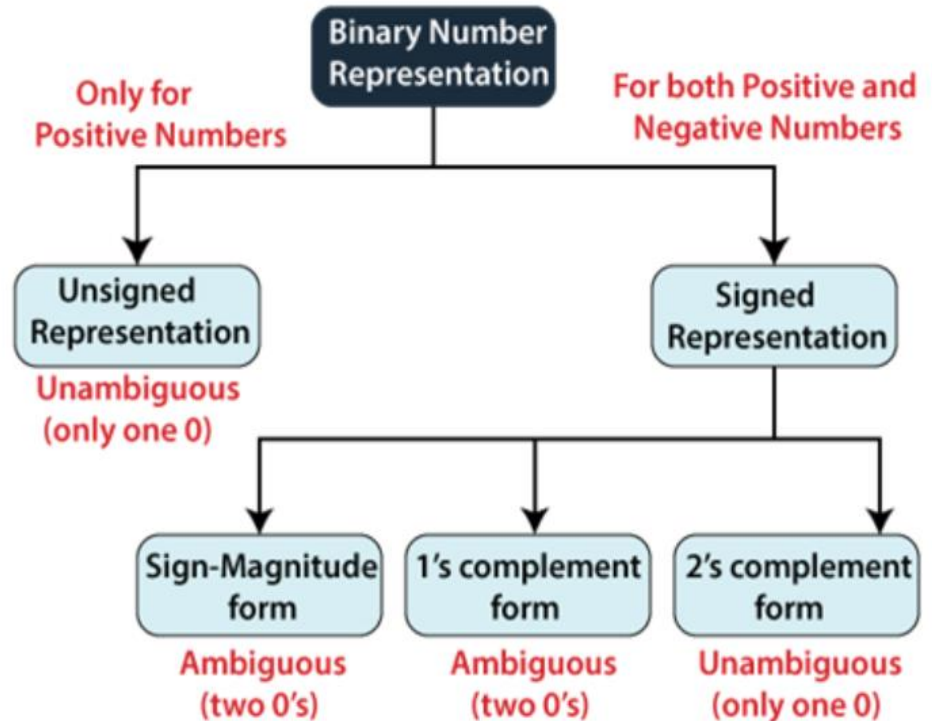
Binary Numbers Representation

- ❑ In number representation techniques, the binary number system is the most used representation technique in digital electronics.
- ❑ Complements are used for representing the negative decimal number in binary form.
- ❑ Different types of complement are possible of the binary number, but **2's complements** is the most widely used for binary numbers.



Binary Numbers Representation (cont.)

- ❑ Unsigned numbers do not use sign bit to represent positive numbers.
- ❑ Signed numbers use sign bit to differentiate +ve and -ve numbers.

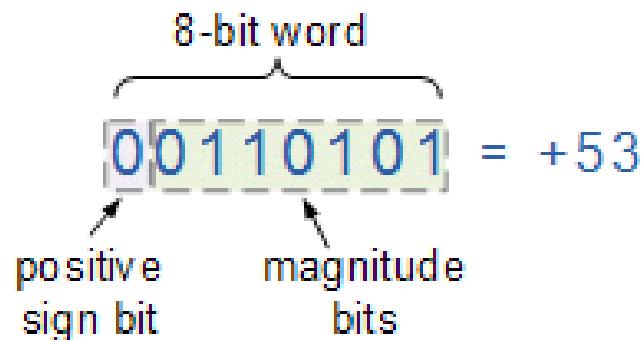


Signed Binary Numbers

- ❑ Representing negative binary number
- ❑ The signed numbers are represented in three ways,
 - Sign-Magnitude form
 - 1's Complement
 - 2's Complement

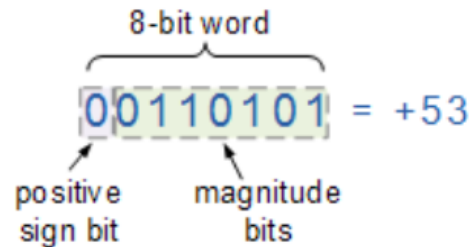
Sign-Magnitude Form

- ❑ A binary number has a bit for a sign (sign bit MSB).
- ❑ If this bit is set to 1, the number will be negative else the number will be positive.
- ❑ Apart from this sign-bit, the $n - 1$ bits represent the magnitude of the number
- ❑ Reduced range due to using MSB as sign bit.

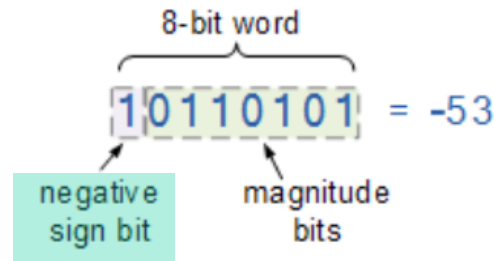


Sign-Magnitude Form (cont)

Positive Signed Binary Numbers



Negative Signed Binary Numbers



Sign bit

Signed binary numbers use the MSB as a sign bit to indicate the number is +ve or -ve number.

Major disadvantages:

- Two 0's
- Reduced range

Sign-Magnitude Form (cont)

Sign 4 2 1

1	1	1	1	-7
1	1	1	0	-6
1	1	0	1	-5
1	1	0	0	-4
1	0	1	1	-3
1	0	1	0	-2
1	0	0	1	-1
1	0	0	0	-0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

Two 0's problem

Weird result

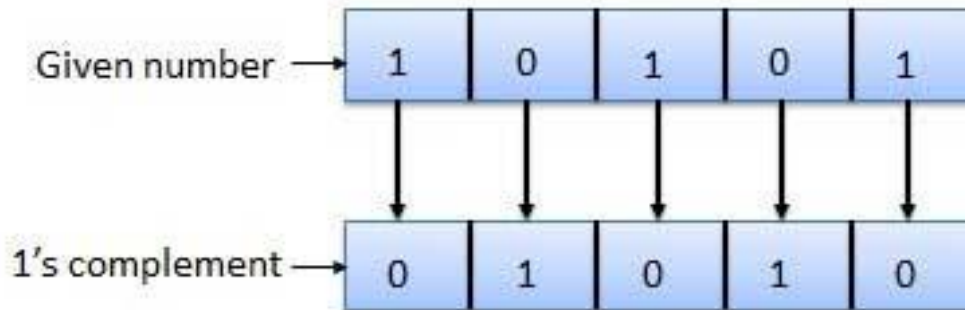
1101 -5

+0101 +5

1 0010 +2

1's Complement

- ❑ By inverting each bit of a number.
- ❑ The binary number also has an extra bit for sign representation as a sign-magnitude form.



1's Complement (cont)

4 bits case

8	4	2	1	
1	0	0	0	-7
1	0	0	1	-6
1	0	1	0	-5
1	0	1	1	-4
1	1	0	0	-3
1	1	0	1	-2
1	1	1	0	-1
1	1	1	1	-0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

Still have two 0's problem

Less weird

1010	-5	1011	-4
+0101	+5	+ 0100	+4
1111	-0	1111	-0

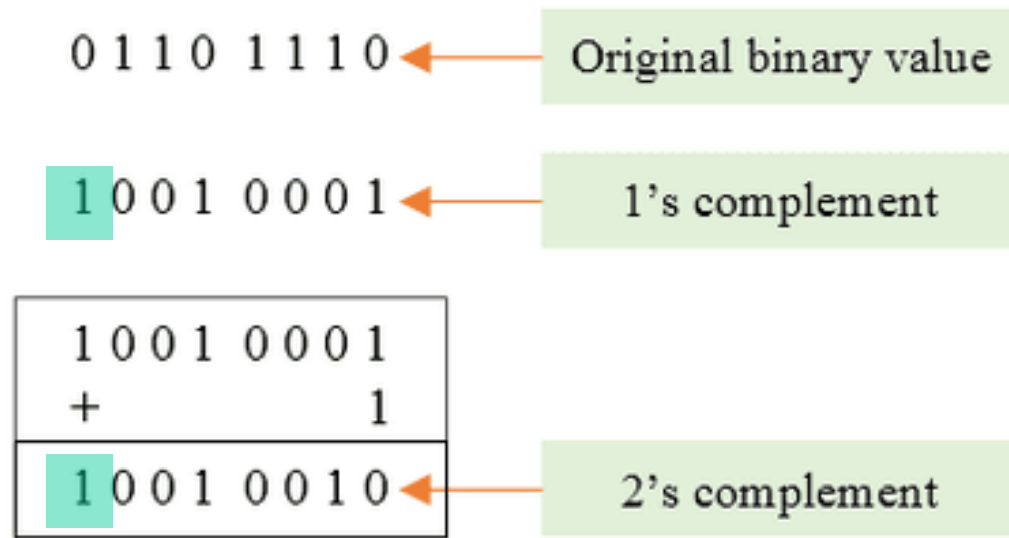
1's Complement Table

Binary Number	1's Complement
0000	1111
0001	1110
0010	1101
0011	1100
0100	1011
0101	1010
0110	1001
0111	1000
1000	0111
1001	0110
1010	0101
1011	0100
1100	0011
1101	0010
1110	0001
1111	0000

9/8/2020

2's Complement

- ❑ By inverting each bit of a number and adding **plus 1** to its **least significant bit**.
- ❑ The binary number also has an extra bit for sign representation as a sign-magnitude form.



2's Complement (cont.)

❑ Example 1: 0100 (representing +4 in a 4 bits case)

- Change all 0's to 1 and all 1's to 0. So the 1's complement of the number is 1011
- Add 1 to the LSB of this number
 - $(1011) + 1 = 1100$ (-4) $-8+4=-4$

❑ Example 2: 0110 (representing =6 in a 4 bits case)

- Change all 0's to 1 and all 1's to 0. So the 1's complement of the number is 1001
- Add 1 to the LSB of this number
 - $(1001) + 1 = 1010$ (-6) $-8+2=-6$

2's Complement (cont.)

8 4 2 1	
1 0 0 1	-7
1 0 1 0	-6
1 0 1 1	-5
1 1 0 0	-4
1 1 0 1	-3
1 1 1 0	-2
1 1 1 1	-1
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7

Normal result

1011	-5
+0101	+5
0000	0

No two 0's problem

1001	-7
-8 + 1 =	-7

The MSB use as both sign
bit and magnitude

2's Complement Table

Binary Number	1's Complement	2's complement
0000	1111	0000
0001	1110	1111
0010	1101	1110
0011	1100	1101
0100	1011	1100
0101	1010	1011
0110	1001	1010
0111	1000	1001
1000	0111	1000
1001	0110	0111
1010	0101	0110
1011	0100	0101
1100	0011	0100
1101	0010	0011
1110	0001	0010
1111	0000	0001

Hexadecimal Number (base 16)

Hexadecimal Number

- ❑ There are 16 digits in hexadecimal numbers represented
- ❑ Hex is used in microprocessor, as it is easy to understand than 0101001. It is a shortcut notation for binary numbers because it works out nicely for 8, 16, 32, and 64 bit processors. You can represent all binary numbers with each set of 4 bits as 1 hex character A=1010, B=1011, F = 1111, 0 = 0000, 7 = 0111, etc.
- ❑ From 0 to 9 same like decimals
 - after that, it starts with an alphabetical representation of preceding numbers such as A (10), B(11), C(12), D(13), E(14) and F(15).

Hexadecimal to Decimal

- The representation of a hexadecimal number into decimal form

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Decimal to Hexadecimal

- ❑ Firstly divide the number by 16.
- ❑ Take the quotient and divide again by 16,
- ❑ The remainder left will produce the hex value.
- ❑ Repeats the steps until the quotient has become 0.

- ❑ Example: Convert $(242)_{10}$ into hexadecimal.
 - Divide 242 by 16 and repeat the steps, till the quotient is left as 0.
 - Therefore, $(242)_{10} = (F2)_{16}$

$$\begin{array}{r|l} 16 & 242 \\ \hline 16 & 15 \quad 2 \rightarrow 2 \\ \hline & 0 \quad 15 \rightarrow F \end{array}$$

More Examples on Decimal to Hexadecimal

Example

Hexadecimal Number – $19FDE_{16}$

Calculating Decimal Equivalent –

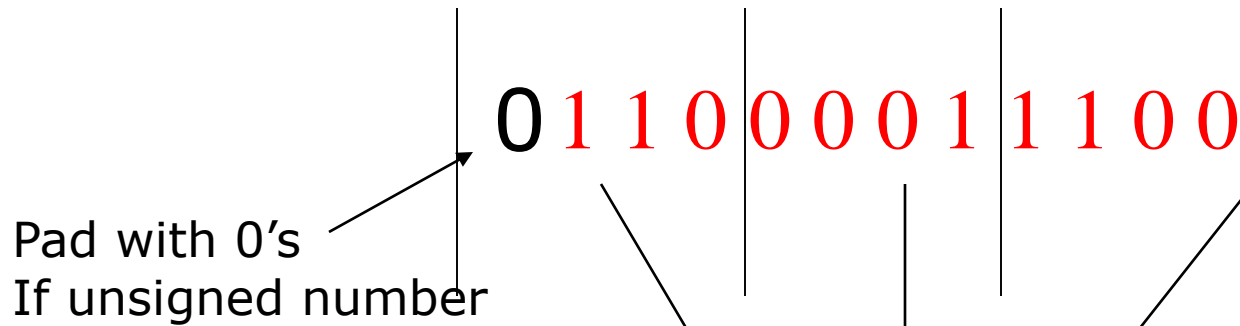
Step	Hexadecimal Number	Decimal Number
Step 1	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$
Step 2	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$
Step 3	$19FDE_{16}$	$(65536 + 36864 + 3840 + 208 + 14)_{10}$
Step 4	$19FDE_{16}$	106462_{10}

Binary to Hex Conversion

1. Divide binary number into 4-bit groups

Pad with 0's
If unsigned number

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



2. Substitute hex digit for each group

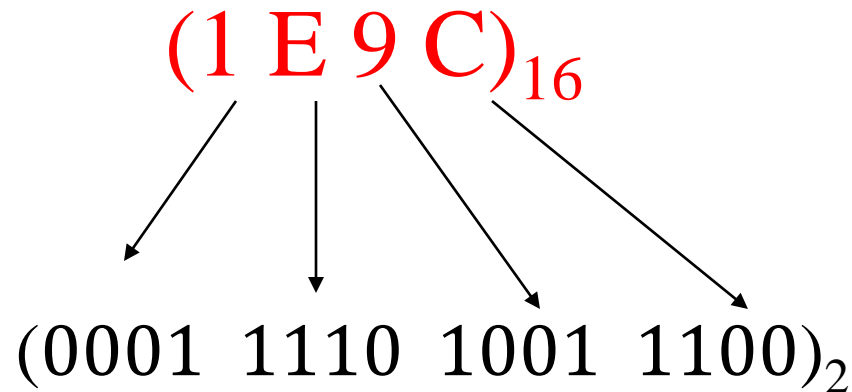
61C₁₆



Hexadecimal to Binary Conversion

Example

1. Convert each hex digit to equivalent binary



Binary Arithmetic

Addition – Unsigned Binary Numbers

- ❑ Adding unsigned numbers:
 - Recall that with 4 bit numbers we can represent numbers from 0 to 15.
 - Addition is done exactly like adding decimal numbers, except that you have only two digits (0 and 1).
- ❑ $0+0 = 0$, with no carry,
- ❑ $1+0 = 1$, with no carry,
- ❑ $0+1 = 1$, with no carry,
- ❑ $1+1 = 0$, and you carry a 1.

Addition- Unsigned Numbers (cont.)

□ Adding unsigned numbers:

- E.g., add the numbers $(06)_{10} = (0110)_2$ and $(07)_{10} = (0111)_2$
- Answer: $(13)_{10} = (1101)_2$

Decimal	Unsigned Binary
$\begin{array}{r} 1 \text{ (carry)} \\ 06 \\ +07 \\ \hline 13 \end{array}$	$\begin{array}{r} 110 \text{ (carry)} \\ 0110 \\ +0111 \\ \hline 1101 \end{array}$

□ The only difficulty adding unsigned numbers occurs when you add numbers that are too large.

- Consider $13+5$.
- The result is a 5 bit number.
- The carry bit from adding the two most significant bits represents a results that *overflows*.

Decimal	Unsigned Binary
$\begin{array}{r} 0 \text{ (carry)} \\ 13 \\ +05 \\ \hline 18 \end{array}$	$\begin{array}{r} 1101 \text{ (carry)} \\ 1101 \\ +0101 \\ \hline 10010 \end{array}$

Addition – Signed Numbers

- ❑ Adding signed numbers:
 - Recall that signed 4 bit numbers (2's complement) can represent numbers between -8 and 7.
- ❑ To see how this addition works, consider three examples.
- ❑ The extra carry (overflow) from the most significant bits has no meaning.

Decimal	Signed Binary
$\begin{array}{r} -2 \\ +3 \\ \hline 1 \end{array}$	$\begin{array}{r} 1110 \text{ (carry)} \\ 1110 \\ +0011 \\ \hline 0001 \end{array}$
Decimal	Signed Binary
$\begin{array}{r} -5 \\ +3 \\ \hline -2 \end{array}$	$\begin{array}{r} 011 \text{ (carry)} \\ 1011 \\ +0011 \\ \hline 1110 \end{array}$
Decimal	Signed Binary
$\begin{array}{r} -4 \\ -3 \\ \hline -7 \end{array}$	$\begin{array}{r} 1100 \text{ (carry)} \\ 1100 \\ +1101 \\ \hline 1001 \end{array}$

Subtraction in 2's complement

13 - 9 = 4 (4 bits representation)

13 : 1 1 0 1

9 : 1 0 0 1

-9 in 2's complement: 0 1 1 0

+ 1

-9 = 0 1 1 1

23 + (-9)

1 1 0 1

+ 0 1 1 1

1 0 1 0 0 = 4

overflow



Fixed-Point Representation

Fixed-Point Representation

Unsigned fixed point

Integer	Fraction
---------	----------

Signed fixed point

Sign	Integer	Fraction
------	---------	----------

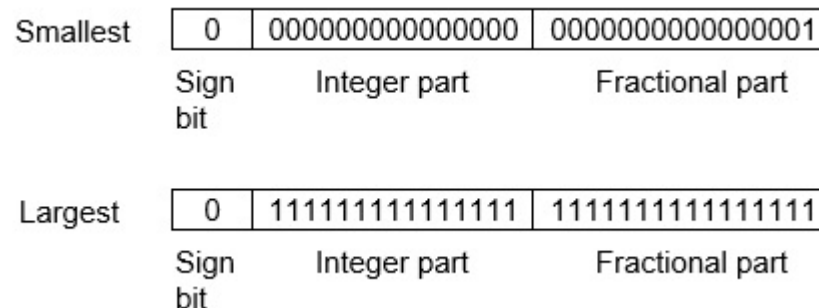
- ❑ We can represent these numbers using:
 - Signed representation,
 - 1's complement representation,
 - 2's complementation representation,
- ❑ 2's complementation representation is preferred in computer system because of
 - Unambiguous property,
 - Easier for arithmetic operations.

Fixed-Point Representation (cont.)

- ❑ Assume number is using 8-bit format which reserve 1 bit for the sign, 4 bits for the integer part and 3 bits for the fractional part.
 - $(10010.110)_2 = (-1) \times (1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-2})$
 $= (-1) \times (2 + 0.5 + 0.25)$
 $= -2.75$
- ❑ Disadvantage is relatively **limited range** of values that they can represent.
- ❑ It is usually inadequate for numerical analysis as it does not allow enough numbers and accuracy.
- ❑ A number whose representation exceeds 32 bits would have to be stored inexactly.

Fixed-Point Representation (cont.)

- ❑ The smallest positive number and largest positive number which can be store in **32-bit representation (1 sign bit, 15 bits integer, 16 bits fraction)** as given above format.



- ❑ The smallest positive number is $2^{-16} \approx 0.000015$ approximate.
- ❑ The largest positive number is $(2^{15} - 1) + (1 - 2^{-16}) = (32768 - 1) + (0.9999847412 \dots) = 32767.9999847\dots$
- ❑ The radix point can be moved to either left or right to increase accuracy or range.

Class work

In a 8 bits representation, assume we don't have a sign bit (just positive number).

Find the number in decimal

1. 0001 0.110

2. 000.1 0110

3. 0001 0110 (integer binary)

Pros and cons of fixed point representation

- ❑ Fixed point numbers are indeed very close to integer representation
- ❑ One may consider integer representation a “special case” of fixed point numbers, where the radix point is at position 0 (no fraction).
- ❑ Thus the benefit of fixed point arithmetic is that they are straightforward and efficient as integers arithmetic in computer.
- ❑ The major disadvantage of fixed point number is the loss of range and precision compared to floating point representation.
- ❑ Fixed point is simple but powerful. It is still being used in many game and DSP applications.

Floating-Point Representation

Floating-Point Representation

- ❑ Computers represent numbers with fractions as floating point
- ❑ The decimal point (or radix) is not set in a fixed position like we discussed in fixed point representation
- ❑ Computers have restriction. They cannot store or manipulate an infinite amount number of floating point values that are restricted by the size of RAM.
- ❑ Use “mantissa” and “exponent” to represent a number with the radix point being float around.

Floating-Point Representation (cont)

Scientific Notation

2's complement

Mantissa and **exponent**

Tells where the radix point is placed

0.1010 [011]

3 bits exponent to indicate the exponent

+4 2 1

Exponent **011** – we know it is a **positive exponent**

Exponent = 3, which means 2^3

Mantissa

0.1010 so we move the radix point to become

0101.0

Binary number becomes $0101.0 = 5.0_{10}$

Floating-Point Representation (cont)

5 bits Mantissa, 3 bits exponent

Positive exponent

Example:

0.1010[010]

Exp = +2

0.1010 becomes 10.10
= 2.5_{10}

Negative exponent

Example:

^{-4 2 1}
0.1101 [110]

Exp = -4 + 2 = -2

It means to move to radix
point 2 places to the left

Mantissa

^{1/2 0.125 0.0625 0.015625}
0.1101 becomes 0.0 0 1 1 0 1

= 0.125 + 0.0625 + 0.015625

= 0.203125_{10}

Floating-Point Representation (cont)

Negative Number

-ve -4 2
1.0 11 [0 1 0]
Exp= +2
Exp+ -2 implies move
radix point to the right by
2 places
1.011 becomes

 0.25
-4 2 1 0.5
1 01. 10

$$= -4 + 1 + 0.5$$
$$= -2.5_{10}$$

Example 6 bits exponent
1.110100000 [000011]

Exp = +3 or 2^3
Mantissa:
1.110100000 becomes

-8 4 2 1 0.5
1 1 1 0. 1 00000

$$= -8 + 4 + 2 + 0.5$$
$$= -1.5_{10}$$

Home work

□ Floating point 8 bits representation, 3 bits are exponent

1. 0.1100 [011]

2. 1.1101 [110]

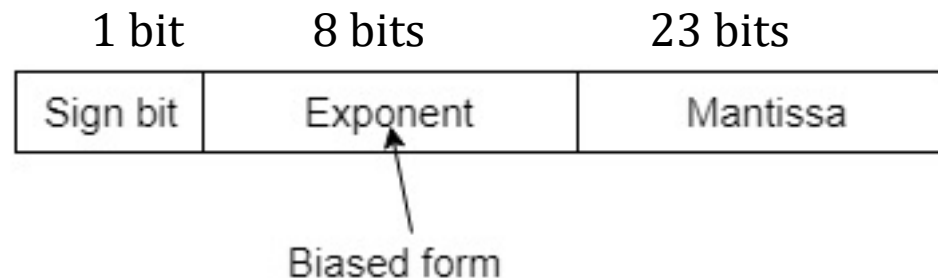
IEEE 754 floating point representation

IEEE 754 Floating-Point Representation

- ❑ A practical engineering approach for computer systems implementation
- ❑ How floating point numbers are stored in computer
- ❑ IEEE 754 was developed in 1985 after addressing many practical hardware implementation problems.
- ❑ IEEE 754 is the most common representation today for hardware, including Intel-based PC. Macs and most Unix platforms.
- ❑ Only the mantissa m and the exponent e are physically represented in the computer (including their sign).
- ❑ 32 bits is single precision, and 64 bits is double precision.

IEEE 754 Floating-Point Representation (cont.)

- Actual number is $(-1)^s (1 + m) 2^{(e - Bias)}$,
where s is the sign bit, m is the mantissa, e is the exponent
value, and $Bias = 127$ for 32 bits single precision.



- The Exp is in a range of 128 (255-127) to -127.
- The floating point representation is more flexible and can represent very large and very small number.

IEEE 754 Floating-Point Representation (cont.)

Example: 263.3_{10} to IEEE 754 32 bits single precision format

Step 1: 263 into binary 1 0000 0111

Step 2: 0.3 to binary: 01001 1001 1001 1001 ... can go as many as we like

Step 3: 263.3 in binary 1 0000 0111 . 01001 1001 1001 1001 ...

Step 4: move the radix point just behind the leading bit, count how many shift and that is the exponent. In this case move to left 8 times, exponent = 8

scientific representation $1.0000011101001\ 1001\ 1001\ 1001... \times 2^8$

Step 5: $8 = e - 127$, $e = 135$, 8 bit exponent is 1000 0111

Step 6: sign bit = "0" positive number

Step 8: combine the sign bit, the exponent 8 bits and the 23 bits fraction (-1 the leading bit)

Answer: 0 10000111 0000 0111 0100 1100 1100 110

IEEE 754 Floating-Point Representation (cont.)

Example: Find the decimal from 32 bit IEEE 754 double precision floating point number

0 1000111 00000111 0000000000000000

Step 1: Sign bit = “0” positive number

Step 2: Exponent 1000111 = 135, or exponent= 135-127 =8, 2^{+8}

Step 3: Mantissa 00000111 0000...0 (fraction part)

$$1+m = 1 + \frac{1}{64} + \frac{1}{128} + \frac{1}{256} = 1.02734375$$

Step 4: Answer

$$(-1^0)(1.02734375)(2^8) = +263_{10}$$

That is exactly the number that we used excluding the 0.3

IEEE 754 Floating-Point Representation (cont.)

32 bits single precision example

1 000 1000 1 000 1000 1000 0000 0000 0000

s	Exp (8 bits)	Mantissa (23 bits)
1	0001 0001	0001 0001 0000 0000 0000 000

Actual Exp: 0001 0001 - 127 (bias) = 16 + 1 - 127 = -110

Mantissa: .0001 0001 0000 0000 0000 000
 0.5 1/16 1/256

$$m = 1/16 + 1/256 = 0.06640625$$

Now back to the IEEE 754 definition

$$\begin{aligned} &= (-1)^1 (1 + 0.06640625) (2^{-110}) \\ &= -1.06640625 \times 2^{-110} \end{aligned}$$

IEEE 754 Floating-Point Representation (cont.)

Largest +ve number:

$$(0\ 11111110\ 111111111111111111111111)_2; (2 - 2^{-23}) \times 2^{127} \approx 3.4028234664 \times 10^{38}$$

Smallest +ve number:

$$(0\ 00000001\ 000000000000000000000000)_2; 2^{-126} \approx 1.1754943508 \times 10^{-38}$$

Largest -ve number:

$$(1\ 00000001\ 000000000000000000000000)_2; (-1)^1 \times 2^{-126} \approx -1.1754943508 \times 10^{-38}$$

Smallest -ve number:

$$(1\ 11111110\ 111111111111111111111111)_2; (-1)^1 \times (2 - 2^{-23}) \times 2^{127} \approx -3.4028234664 \times 10^{38}$$

$$+0: (0\ 00000000\ 000000000000000000000000)_2$$

$$-0: (1\ 00000000\ 000000000000000000000000)_2$$

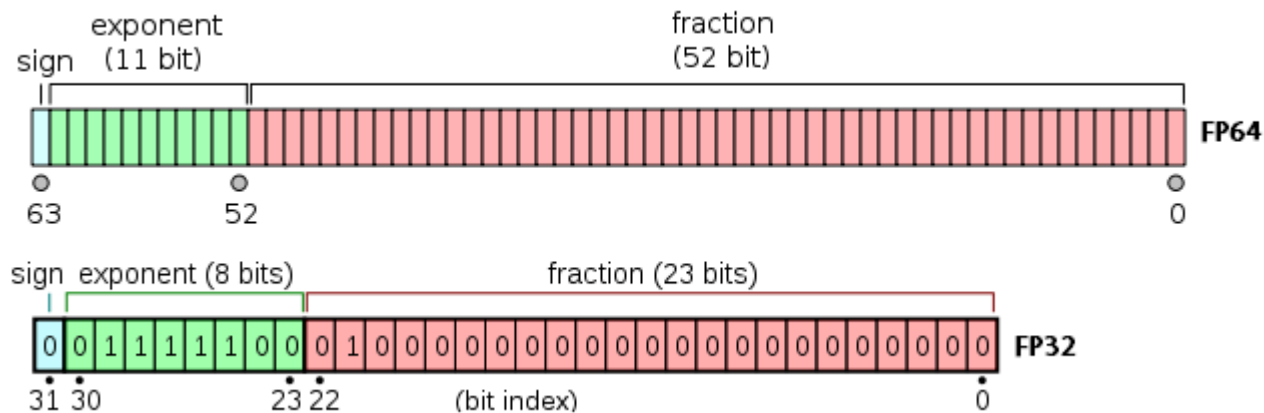
$$\text{infinity: } (0\ 11111111\ 000000000000000000000000)_2$$

$$-\text{infinity: } (1\ 11111111\ 000000000000000000000000)_2$$

IEEE 754 Floating-Point Representation (cont.)

64 bits double precision:

1 011 1111 1111 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
s | Exp (11 bits) | Mantissa (52 bits)



Desktop
Intel x86-64
Processor



Mobile
Qualcomm
ARM 64-bit
Processor



9/8/2020

Home exercise

Work out the 32 bit IEEE 754 single precision floating point representation for 147.625

Answer:

0 1000 0110 0010 0111 0100 0000 0000 000

Floating-Point Arithmetic

IEEE 754 Floating-Point Addition/ Subtraction

- Make sure the 2 exponents are of the same value
- Move the lower exponent to the same value of the higher exponent.
- Just work on the fraction or mantissa part
- Normalize it back after addition
- Subtraction use 2's complement and carry the same process as addition

IEEE 754 Floating-Point Addition Subtraction

Example X+Y (same exponent value)

X: 0 1101 0111 111 0011 1010 0000 1100 0011

Y: 0 1101 0111 000 1110 0101 1111 0001 1100

Step 1: Important step, are the exponent the same? Yes in this case

Step 2 Both exponents are 1101 0111, which is 215 (215-127=88), but we don't need to bother the -127 as we are not finding the absolute value. so we can add the mantissa direct.

Step 3: we can add the mantissa direct; recall it is 1+m

X: 1.111 0011 1010 0000 1100 0011 $\times 2^{215-127}$

Y: 1.000 1110 0101 1111 0001 1100

X+Y **11**.000 0001 1111 1111 1101 1111 $\times 2^{215-127}$

Step 4: Because "**11**", move the radix point to the one place to the right to align with the format

X+Y 1. 100 0000 1111 1111 1110 1111 **1** $\times 2^{**216**-127}$

Computer lost this last bit, lack of memory space

exponent becomes 1101 0111 +1 =1101 1000

Step 5: X+Y= [0] **[1101 1000]** [100 0000 1111 1111 1110 1111]

IEEE 754 Floating-Point Addition/ Subtraction

Example addition with different exponent values, find X+Y

X: 0 1101 0111 111 0011 1010 0000 1100 0011

Y: 0 1101 0001 000 1110 0101 1111 0001 1100

Exponent of X: 1101 0111 = 215

Exponent of Y: 1101 0001 = 209

Step 1: Thus shift the radix pt of Y to the right by 6 places

add 5 "0" 1.000 1110 0101 1111 0001 1100 becomes
0.000 0010 0011 1001 0111 1100 011100 (lost these 6 bits in the computer, no space)

Step 2: Add X to Y's mantissa (1+m)

1.111 0011 1010 0000 1100 0011
+ 0.000 0010 0011 1001 0111 1100 $\times(2^{215-127})$
1.111 0101 1101 1010 0011 1111 $\times(2^{215-127})$

Step 3: The form the computer will save is

[0] [1101 0111] [111 0101 1101 1010 0011 1111]

IEEE 754 Floating-Point Subtraction

Example $16 - 7 = 9$, $X - Y$

(16) X: [0] [1000 0011] [000 0000 0000 0000 0000 0000] exp=4 2^4

(7) Y: [0] [1000 0001] [110 0000 0000 0000 0000 0000] exp=2 2^2

Y has a lower exponent than X

Step 1: move to the same exponent, means shift Y's exponent by 2 places

Y: 1.110 0000 0000 0000 0000 0000 $\times(2^2)$ move the radix point to the right by 2 places

Y: 0.011 1000 0000 0000 0000 0000 $\times(2^4)$

Step 2: use 2's complement to get $-Y$

0.011 1000 0000 0000 0000 0000

1.100 0111 1111 1111 1111 1111

+ 1

-Y 1.100 1000 0000 0000 0000 0000

+X 1.000 0000 0000 0000 0000 0000 (1+m)

X-Y= 1.001 0000 0000 0000 0000 0000 "1" is the overflow bit, not count $\times(2^4)$

shift radix pt to the right by 1 place 1.001 0000 0000 0000 0000 0000 $\times(2^3)$

IEEE 754 format: [0] [1000 0010] [001 0000 0000 0000 0000 0000]

Check back in decimal: $X - Y = (1 + 0.125)(2^{130-127}) = 9$

End