### **Tutorial 02:**

Addition/Subtraction using 2's Complement &

**Floating-point Numbers** 

**Computer Science Department** 

CS2208b: Fundamentals of Computer Organization and Architecture

**Winter 2018** 

Instructor: Mahmoud R. El-Sakka

Office: MC-419

Email: elsakka@csd.uwo.ca

**Phone:** 519-661-2111 x86996

### **Binary Arithmetic**

☐ These tables cover the fundamental arithmetic operations.

Addition	Subtraction	Multiplication						
0 + 0 = 0 (carry 0)	0 - 0 = 0 (borrow 0)	$0 \times 0 = 0$						
0 + 1 = 1 (carry 0)	0 - 1 = 1 (borrow 1)	$0 \times 1 = 0$						
1 + 0 = 1 (carry 0)	1 - 0 = 1 (borrow 0)	$1 \times 0 = 0$						
1 + 1 = 0 (carry 1)	1 - 1 = 0 (borrow 0)	$1 \times 1 = 1$						

#### **Addition (three bits)**

#### **Subtraction (three bits)**

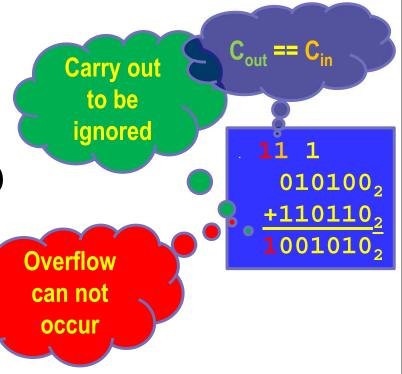
1 - 1 - 1 = 1 (borrow 1)

# Sign and Magnitude Addition/Subtraction

- The operations are carried out similar to normal math calculations
- The resultant sign is arranged separately
  - $\square$  The sign of A B depends on the values of A and B
  - $\square$  If B > A, the answer will be calculated as -(B A), O.W., it is (A B)
- The location of the radix points need to be aligned before performing the operation.
- If the provided number of bits are not enough to hold the result, it means an overflow occurred.

- A subtraction operation is converted to an addition operation (after performing the 2's complement to the operand appearing after the negative sign)
- When adding two positive numbers and finding the result is negative, this means an overflow occurred.
- When adding two negative numbers and finding the result is positive, this means an overflow occurred.
- Overflow will never occur when adding a positive number to a negative number, or vice versa.
- How about
  - □ subtracting a negative number from a positive number?
  - □ subtracting a positive number from a negative number?

- **■** *Example 1*:
  - Perform  $20_{10} 10_{10}$  using 2's complement 6-bit system
- $\bullet$  20<sub>10</sub>  $\rightarrow$  10100<sub>2</sub>
- $10_{10}$  →  $1010_2$
- $20_{10} 10_{10} \rightarrow 10100_2 1010_2$ 
  - $\rightarrow$  010100<sub>2</sub> 001010<sub>2</sub>
  - $\rightarrow$  010100<sub>2</sub> + (-001010<sub>2</sub>)
  - $\rightarrow 010100_2 + 110110_2$
  - **→** 001010<sub>2</sub>
  - **→** +10<sub>10</sub>



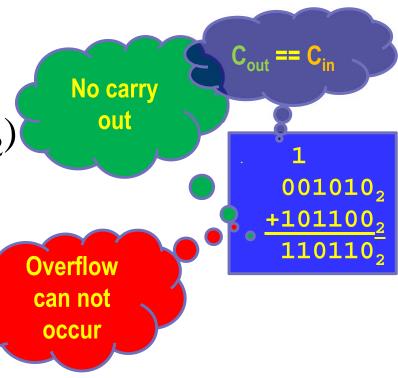
**■** *Example 2*:

Perform  $10_{10} - 20_{10}$  using 2's complement 6-bit system

- $\bullet$  10<sub>10</sub>  $\rightarrow$  1010<sub>2</sub>
- $\bullet$  20<sub>10</sub>  $\rightarrow$  10100<sub>2</sub>

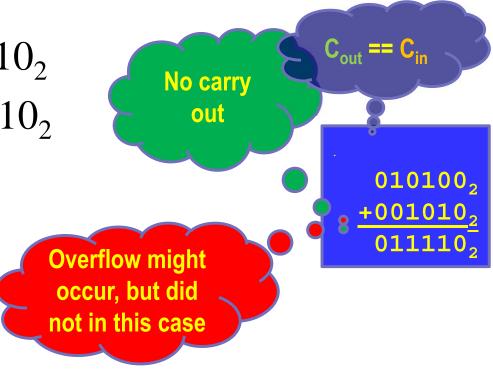
■ 
$$10_{10} - 20_{10}$$
 →  $1010_2 - 10100_2$ 

- $\rightarrow$  001010<sub>2</sub> 010100<sub>2</sub>
- $\rightarrow$  001010<sub>2</sub> + (-010100<sub>2</sub>)
- $\rightarrow$  001010<sub>2</sub> + 101100<sub>2</sub>
- **→** 110110<sub>2</sub>
- **→**-001010<sub>2</sub>



- **■** *Example 3*:
  - Perform  $20_{10} + 10_{10}$  using 2's complement 6-bit system
- $\bullet$  20<sub>10</sub>  $\rightarrow$  10100<sub>2</sub>
- $\bullet$  10<sub>10</sub>  $\rightarrow$  1010<sub>2</sub>

■ 
$$20_{10} + 10_{10} \Rightarrow 10100_2 + 1010_2$$
  
⇒  $010100_2 + 001010_2$   
⇒  $0111110_2$   
⇒  $+30_{10}$ 



- **■** *Example 4*:
  - Perform  $-20_{10} 10_{10}$  using 2's complement 6-bit system
- $\bullet$  20<sub>10</sub>  $\rightarrow$  10100<sub>2</sub>
- $\bullet$  10<sub>10</sub>  $\rightarrow$  1010<sub>2</sub>
- $-20_{10} 10_{10} \rightarrow -10100_2 1010_2$ 
  - $\rightarrow$  -010100<sub>2</sub> 001010<sub>2</sub>
  - $\rightarrow$  (-010100<sub>2</sub>)+ (-001010<sub>2</sub>)
  - $\rightarrow$  101100<sub>2</sub> + 110110<sub>2</sub>
  - $\rightarrow$  100010<sub>2</sub>
  - **→** -011110<sub>2</sub>
  - $\rightarrow$  -30<sub>10</sub>



1111

101100,

Overflow might occur, but did not in this case

**■** *Example 5*:

Perform  $20_{10} + 20_{10}$  using 2's complement 6-bit system

 $\bullet$  20<sub>10</sub>  $\rightarrow$  10100<sub>2</sub>

■ 
$$20_{10} + 20_{10}$$
 →  $10100_2 + 10100_2$   
→  $010100_2 + 010100_2$   
No carry out  $010100_2$   
+  $010100_2$   
occur, and indeed it did in this case

**■** *Example 6*:

Perform  $-20_{10} - 20_{10}$  using 2's complement 6-bit system

- $\bullet$  20<sub>10</sub>  $\rightarrow$  10100<sub>2</sub>
- $-20_{10} 20_{10} \rightarrow -10100_2 10100_2$ 
  - $\rightarrow$  -010100<sub>2</sub> 010100<sub>2</sub>
  - $\rightarrow$  (-010100<sub>2</sub>)+ (-010100<sub>2</sub>) Carry out
  - $\rightarrow$  101100<sub>2</sub> + 101100<sub>2</sub>

 $10100_2$  Carry out to be ignored

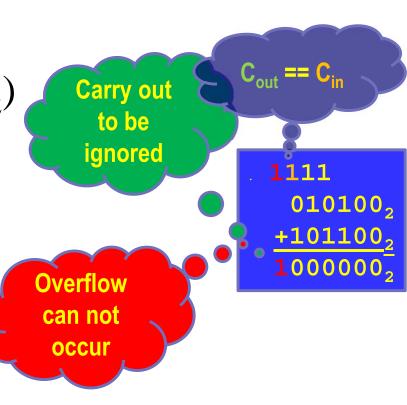
Overflow might occur, and indeed it did in this case

 $\begin{array}{c|c}
1 & 11 \\
 & 101100_{2} \\
 & +101100_{2} \\
\hline
 & 1011000_{2}
\end{array}$ 

**■** *Example 7*:

Perform  $20_{10} - 20_{10}$  using 2's complement 6-bit system

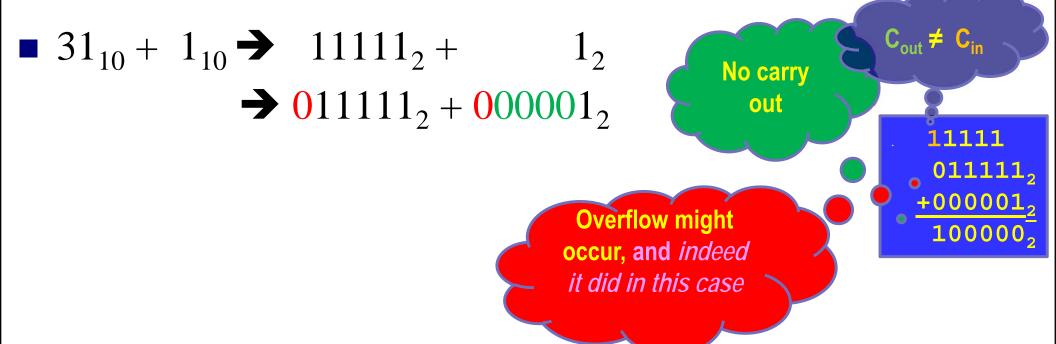
- $\blacksquare 20_{10} \rightarrow 10100_2$
- $\blacksquare 20_{10} 20_{10} \rightarrow 10100_2 10100_2$ 
  - $\rightarrow$  010100<sub>2</sub> 010100<sub>2</sub>
  - $\rightarrow 010100_2 + (-010100_2)$
  - $\rightarrow 010100_2 + 101100_2$
  - **→** 0000000<sub>2</sub>
  - $\rightarrow 0_{10}$



**■** *Example 8*:

Perform  $31_{10} + 1_{10}$  using 2's complement 6-bit system

- $\blacksquare$  31<sub>10</sub>  $\rightarrow$  11111<sub>2</sub>



- **■** *Example 9*:
  - Perform  $-31_{10}$   $1_{10}$  using 2's complement 6-bit system
- $\blacksquare$  31<sub>10</sub>  $\rightarrow$  111111<sub>2</sub>

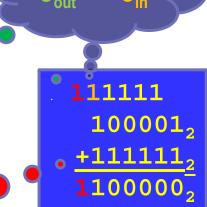
Carry out to be ignored

$$-31_{10} - 1_{10} \rightarrow -111111_2 - 1_2$$

$$\rightarrow$$
 (-0111111<sub>2</sub>) + (-000001<sub>2</sub>)

$$\rightarrow$$
 ( 100001<sub>2</sub>) + ( 111111<sub>2</sub>)

Overflow might occur, but did not in this case



**■** *Example 10*:

Encode –3.25<sub>10</sub> using 2's complement 6-bit system

- $\blacksquare$  3.25<sub>10</sub>  $\Rightarrow$  11.01<sub>2</sub>
- $-3.25_{10} \rightarrow -0011.01_2$ 
  - **→** 1100.11<sub>2</sub>

Carry out to be ignored

You can also look at it as if it is  $-3_{10}$   $-0.25_{10}$ 

$$-3_{10} - 0.25_{10} \rightarrow -11_2 - 0.01_2$$

$$\rightarrow$$
  $(-000011_2) + (-0000.01_2)$ 

$$\rightarrow$$
 ( 111101<sub>2</sub>) + ( 1111.11<sub>2</sub>)

Overflow might occur, but did not in this case

CS 2208: Introduced puter Organization and Architecture



Binary points
MUST be
aligned

### **Example of Decimal to IEEE-754 Floating-point Conversion**

**■** *Example 11*:

Convert 16777216.75<sub>10</sub> into a 32-bit single-precision IEEE-754 FP value.

- Convert 16777216.75<sub>10</sub> into a fixed-point binary
  - $16777216_{10} = 1\,0000\,0000\,0000\,0000\,0000\,0000_2$  and
  - $0.75_{10} = 0.11_2.$
  - Therefore,  $16777216.75_{10} = 100000000000000000000000011_2$ .
- Normalize 1 0000 0000 0000 0000 0000 0000.11<sub>2</sub> to
   1.0000 0000 0000 0000 0000 0000 11<sub>2</sub> x 2<sup>24</sup>.
- The sign bit, S, is 0 because the number is positive
- The *biased exponent* is the *true exponent* plus 127; that is,  $24 + 127 = 151_{10} = 1001 \ 0111_2$
- o The significand is 000 0000 0000 0000 0000 011
  - the leading 1 is stripped and
  - the significand to be rounded to 23 bits (rounded to nearest FP number).

What is the effect of using rounding toward +∞, rounding toward -∞, or rounding using truncation?

### **Example of Decimal to IEEE-754 Floating-point Conversion**

**■** Example 12:

Convert 16777219<sub>10</sub> into a 32-bit single-precision IEEE-754 FP value.

- Convert 16777219<sub>10</sub> into a fixed-point binary
  - $16777219_{10} = 1\,0000\,0000\,0000\,0000\,0000\,0011_2$  and
- o Normalize 1 0000 0000 0000 0000 0000 0011<sub>2</sub> to  $1.0000\ 0000\ 0000\ 0000\ 0000\ 0011_2 \times 2^{24}$ .
- The sign bit, S, is 0 because the number is positive
- The *biased exponent* is the *true exponent* plus 127; that is,  $24 + 127 = 151_{10} = 1001 \ 0111_2$
- The significand is 000 0000 0000 0000 0000 0001
  - the leading 1 is stripped and
  - the significand to be rounded to 23 bits (rounded to nearest FP number).

What is the effect of using rounding toward +∞, rounding toward -∞, or rounding using truncation?

Mid-way → round to even significand



### **Example of Decimal to IEEE-754 Floating-point Conversion**

- □ Example 13: Convert 3.6<sub>10</sub> into a 32-bit single-precision IEEE-754 FP value.
  - Convert 3.6<sub>10</sub> into a fixed-point binary

$$\mathbf{3}_{10} = 11_2$$
 and

$$\bullet$$
 0.6<sub>10</sub> = 0.1001 1001 ... <sub>2</sub>.

- Therefore,  $3.6_{10} = 11.1001 \ 1001 \ \dots \ _2$
- o Normalize 11.1001 1001 ...  $_2$  to 1.11001 1001 ...  $_2 \times 2^1$ .

 $0.6 \times 2 = 1.2$   $0.2 \times 2 = 0.4$   $0.4 \times 2 = 0.8$   $0.8 \times 2 = 1.6$   $0.6 \times 2 = 1.2$ ...

- The sign bit, S, is 0 because the number is positive
- The *biased exponent* is the *true exponent* plus 127; that is,  $1 + 127 = 128_{10} = 1000 \ 0000_2$
- The significand is 110 0110 0110 0110 0110 0110 0110 ...
  - the leading 1 is stripped and
  - the significand to be rounded to 23 bits (rounded to nearest FP number).
- o The final number is 0100 0000 0110 0110 0110 0110 0110, or  $40666666_{16}$ . →  $3.5999999046325684_{10}$

#### Tutorial 02: Addition/Subtraction using 2's Complement &Floating-point Numbers

#### **Example of Decimal to IEEE-754 Floating-point Conversion**

□ Example 14: Convert 5.877472<sub>10</sub>×10<sup>-39</sup> into a 32-bit single-precision IEEE-754 FP value.  $\frac{\text{Log}_2(10) = 1/\log_{10}(2)}{\text{Log}_2(10)} = \frac{1}{\log_{10}(2)}$ 

$$10^{-39} = 2^{z} \implies \log_{2}(10^{-39}) = z \implies -39 \times \log_{2}(10) = z \implies z = -129.5551957$$

$$10^{-39} = 2^{-129.5551957} = 2^{-129} \times 2^{-0.5551957} = = 2^{-129} \times 0.680564734_{10}$$

$$5.877472_{10} \times 10^{-39} = 5.877472_{10} \times 0.680564734_{10} \times 2^{-129}$$

$$= 4_{10} \times 2^{-129} = 1_{10} \times 2^{-127}$$

- Convert 1<sub>10</sub> into a fixed-point binary
  - $1_{10} = 1.0_2$  (already normalized)
- o True exponent is less than -126 → underflow case
  - The exponent needs to be -126: -127<sub>10</sub> = -126 -1
  - Hence, the significant needs to be adjusted to compensate the -1
  - After moving the radix point backward by 1 position  $\rightarrow$  0.1<sub>2</sub> i.e.,  $2^{-127} = 0.1 \times 2^{-126}$
  - After Taking 23 bits → 0. 100 0000 0000 0000 0000 0000<sub>2</sub>
- The sign bit, S, is 0 because the number is positive

#### Tutorial 02: Addition/Subtraction using 2's Complement & Floating-point Numbers

#### **Example of Decimal to IEEE-754 Floating-point Conversion**

Example 15: Convert  $9.0_{10} \times 10^{-44}$  into a 32-bit single-precision IEEE-754 FP value.

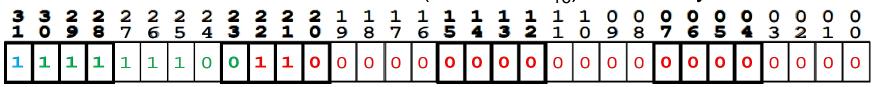
$$10^{-44} = 2^z$$
  $\rightarrow \log_2(10^{-44}) = z$   $\rightarrow -44 \times \log_2(10) = z$   $\rightarrow z = -146.164836175$   
 $10^{-44} = 2^{-146.164836175} = 2^{-146} \times 2^{-0.164836175} = = 2^{-146} \times 0.892029808_{10}$   
 $9.0_{10} \times 10^{-44} = 9.0_{10} \times 0.892029808_{10} \times 2^{-146} = 8.028268272_{10} \times 2^{-146}$ 

- o Convert 8.028268272<sub>10</sub> into a fixed-point binary
  - $\bullet$  8<sub>10</sub> = 1000<sub>2</sub> and
  - $0.028268272_{10} = 0.00000111001111001001..._2$ .
  - Therefore, 8.028268272<sub>10</sub> =  $1000.00000111001111001001..._2$ .
- o Normalization:  $9.0_{10} \times 10^{-44} = 8.028268272_{10} \times 2^{-146} = 1000.00000111001111001001..._2 \times 2^{-146} = 1.00000000111001111001001..._2 \times 2^{-143}$
- o True exponent is less than -126 → underflow case
  - The exponent needs to be -126: -143<sub>10</sub> = -126 -17
  - Hence, the significant needs to be adjusted to compensate the -17

  - After Taking only 23 bits → 0. 000 0000 0000 0000 0100 0000 0011...<sub>2</sub>
- The sign bit, S, is 0 because the number is positive
- The final number is 0000 0000 0000 0000 0000 0100 0000 or 00000040<sub>16</sub>

### **Example of IEEE-754 Floating-point to Decimal Conversion**

- □ Example 16: Convert FE600000<sub>16</sub> from 32-bit single-precision IEEE-754 FP value into a decimal value.
  - Convert the hexadecimal number (FE600000<sub>16</sub>) into binary form



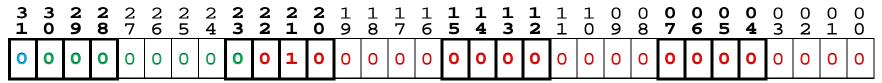
- Unpack the number into sign bit, biased exponent, and fractional significand.
  - S = 1
  - E = 1111 1100
  - F = 110 0000 0000 0000 0000 0000
- As the sign bit is 1, the number is negative.
- o We subtract 127 from the *biased exponent* 1111 1100<sub>2</sub> to get the *true exponent*  $\rightarrow$  1111 1100<sub>2</sub> 0111 1111<sub>2</sub> = 0111 1101<sub>2</sub> = 125<sub>10</sub>.
- The fractional significand is
   .110 0000 0000 0000 0000 0000<sub>2</sub>.
- Reinserting the leading one gives 1.110 0000 0000 0000 0000 00002.
- o The number is  $-1.11_2 \times 2^{125} = -1.75 \times 2^{125}$

$$2^{125} = 10^z$$
  $\rightarrow$   $\log_{10}(2^{125}) = z$   $\rightarrow$   $z = 37.62874946$   
 $2^{125} = 10^{37.62874946} = 10^{37} \times 10^{0.62874946} = 10^{37} \times 4.253529587$   
 $-1.75 \times 2^{125} = -1.75 \times 4.253529587 \times 10^{37} = -7.443676776 \times 10^{37}$ 

#### Tutorial 02: Addition/Subtraction using 2's Complement &Floating-point Numbers

### **Example of IEEE-754 Floating-point to Decimal Conversion**

- Example 17: Convert 00200000<sub>16</sub> from 32-bit single-precision IEEE-754 FP value into a decimal value.
  - Convert the hexadecimal number (00200000<sub>16</sub>) into binary form



- Unpack the number into sign bit, biased exponent, and fractional significand.
  - S = 0
  - $\blacksquare$  **E** = 0000 0000
  - F =010 0000 0000 0000 0000 0000

We are subtracting 126, not 127, from the biased exponent, because the biased exponent = 0.

As the sign bit is 0, the number is positive.
 We subtract 126 from the biased exponent 0, to get

o We subtract 126 from the *biased exponent*  $0_2$  to get the *true exponent*  $0_2$  0 0111 1110 $0_2$  = -126 $0_1$ 0.

As the true exponent is -126, then the F is not normalized

- $\circ$  The fractional significand is .010 0000 0000 0000 0000 0000<sub>2</sub>.
- o The number is  $.01_2 \times 2^{-126} = 2^{-2} \times 2^{-126} = 2^{-128}$

$$2^{-128} = 10^z$$
  $\rightarrow$   $\log_{10}(2^{-128}) = z$   $\rightarrow$   $z = -38.53183944
 $2^{-128} = 10^{-38.53183944} = 10^{-38} \times 10^{-0.53183944} = 10^{-38} \times 0.293873587$ 
 $2^{-128} = 0.293873587 \times 10^{-38} = 2.9387358 \times 10^{-39}$$ 

#### Tutorial 02: Addition/Subtraction using 2's Complement &Floating-point Numbers

### **Example of IEEE-754 FP to Decimal to IEEE-754 FP Conversion**

☐ Example 18:

Convert 4B800002<sub>16</sub> from the 32-bit single-precision IEEE-754 FP representation into decimal representation. <u>Then</u> add 1.0<sub>10</sub> to the result. And <u>finally</u> convert it back to the 32-bit single-precision IEEE-754 FP representation.

o Convert the hexadecimal number (4B800002<sub>16</sub>) into binary form

1	3	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	ĺ	Ŏ
0	1	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

- Unpack the number into sign bit, biased exponent, and fractional significand.
  - S = 0
  - E = 1001 0111
  - F =000 0000 0000 0000 0000 0010
- As the sign bit is 0, the number is positive.
- o We subtract 127 from the *biased exponent* 1001 0111<sub>2</sub> to get the *true exponent*  $\rightarrow$  1001 0111<sub>2</sub> 0111 1111<sub>2</sub> = 0001 1000<sub>2</sub> = 24<sub>10</sub>.
- The fractional significand is
   .000 0000 0000 0000 0000 0010<sub>2</sub>.
- Reinserting the leading one gives 1.000 0000 0000 0000 0000 0010<sub>2</sub>.
- o The number is  $+(1 + 2^{-22}) \times 2^{24} = 2^{24} + 2^2 = 1024_{10} \times 1024_{10} \times 16_{10} + 4_{10} = 16777220_{10}$

#### **Example of IEEE-754 FP to Decimal to IEEE-754 FP Conversion**

- ☐ Example 18 (continution):
  - Adding  $1.0_{10}$  to the result  $\rightarrow$  16777220<sub>10</sub> +  $1.0_{10}$  = 16777221<sub>10</sub>

Converting the result back to the 32-bit single-precision IEEE-754 FP format

- Convert 16777221<sub>10</sub> into a fixed-point binary
  - $16777221_{10} = 1\,0000\,0000\,0000\,0000\,0000\,0101_2$  and
- Normalize 1 0000 0000 0000 0000 0000 0101<sub>2</sub> to
   1.0000 0000 0000 0000 0000 0101<sub>2</sub> x 2<sup>24</sup>.
- The sign bit, S, is 0 because the number is positive
- o The *biased exponent* is the *true exponent* plus 127; that is,  $24 + 127 = 151_{10} = 1001 \ 0111_2$
- o The significand is 000 0000 0000 0000 0000 0010 1 ●●
  - the leading 1 is stripped and
  - the significand to be rounded to 23 bits (rounded to nearest FP number).

 $16777220_{10} + 1.0_{10} = 16777220_{10}!!!$ (This is due to the rounding error)

Mid-way →

round to even significand

#### **Example of IEEE-754 FP to Decimal to IEEE-754 FP Conversion**

□ Example 18 (continution):

#### Note that:

- $\circ$  16777220<sub>10</sub> = 1.0000 0000 0000 0000 0000 0101<sub>2</sub> × 2<sup>24</sup>
- $0 1.0_{10} = 1_2 \times 2^0$
- The absolute difference between the exponents of the two FP normalized numbers = 24
- The significand is expressed in 23 bits
- As the absolute difference between the exponents of the two FP normalized numbers is ≥ the number of significand bits + 1 → the result is the larger number of the two, which is 16777220<sub>10</sub>
- Run the following program to verify Example 18:

```
#include <stdio.h>
int main()
{
  float f = 16777220, ff;
  ff = f + 1;
  printf("%f %f \n", f, ff);
}
```

Change the "float" to "int" and the "%f" to "%d" and repeat executing the program again.



### **Final Word!!**

- **□** How can I verify my results?
- □ There are many online converters between IEEE FP format to float.
  - For example, <a href="https://www.h-schmidt.net/FloatConverter/IEEE754.html">https://www.h-schmidt.net/FloatConverter/IEEE754.html</a>