These slides are being provided with permission from the copyright for CS2208 use only. The slides must not be reproduced or provided to anyone outside of the class.

All download copies of the slides and/or lecture recordings are for personal use only. Students must destroy these copies within 30 days after receipt of final course evaluations.

# Tutorial 03: Addition/Subtraction using 2's Complement

Computer Science Department

CS2208: Introduction to Computer Organization and Architecture

Fall 2022-2023

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)**

#### 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 needs 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

- $20_{10} \rightarrow 10100_2$
- $10_{10}$  →  $1010_2$
- $20_{10} 10_{10} \rightarrow 10100_2 1010_2$ 
  - $\rightarrow$  010100<sub>2</sub> 001010<sub>2</sub>
  - $\rightarrow 010100_2 + (-001010_2)$
- This is the answer in 2's complement
- **→** 010100<sub>2</sub> +
- $^{\circ}$   $\rightarrow$  001010<sub>2</sub>
- This step is not needed. It is just for you to verify.  $\longrightarrow$  +10<sub>10</sub>

 $(-001010_2)$   $110110_2$ Overflow can not occur

**Carry out** 

to be

0101002

+110110

1001010,

CS 2208: Introduction to Computer Organization and Architecture

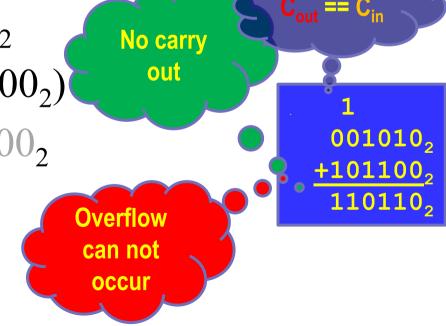
**■** *Example 2*:

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>
- $10_{10} 20_{10} \rightarrow 1010_2 10100_2$ 
  - $\rightarrow$  001010<sub>2</sub> 010100<sub>2</sub>
  - $\rightarrow$  001010<sub>2</sub> + (-010100<sub>2</sub>)
- This is the answer in 2's complement
- $\rightarrow$  001010<sub>2</sub> + 101100<sub>2</sub>
- **→** 110110<sub>2</sub>
- $-001010_2$

This step is not needed. It is just for you to verify.

**→**-10<sub>10</sub>



**■** *Example 3*:

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

- $20_{10} \rightarrow 10100_2$
- $10_{10}$  →  $1010_2$
- $\blacksquare 20_{10} + 10_{10} \rightarrow 10100_2 + 1010_2$

This is the answer in 2's complement

- $\rightarrow$  010100<sub>2</sub> + 001010<sub>2</sub>
- $^{\circ}$   $\rightarrow$  0111110<sub>2</sub>

This step is not needed. It is just for you to verify.

$$\longrightarrow +30_{10}$$



■ *Example 4*:

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

- $\bullet$  20<sub>10</sub>  $\rightarrow$  10100<sub>2</sub>
- $10_{10}$  →  $1010_2$

$$-20_{10} - 10_{10} \rightarrow -10100_2 - 1010_2$$

$$\rightarrow$$
 -010100<sub>2</sub> - 001010<sub>2</sub>

$$\rightarrow$$
 (-010100<sub>2</sub>)+ (-001010<sub>2</sub>

This is the answer in 2's complement

This step is not

needed. It is just\_ for vou to verify.

$$\rightarrow$$
 101100<sub>2</sub>+

$$-011110_2$$

$$\rightarrow$$
 -30<sub>10</sub>

Overflow might occur, but it did not in this case

110110<sub>2</sub>

Carry out to be ignored

1111 101100<sub>2</sub> +110110<sub>2</sub> 1100010<sub>3</sub>

© Mahmoud R. El-Sakka

CS 2208: Introduction to Computer Organization and Architecture

**■** *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

Overflow might 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>

Carry out to be ignored

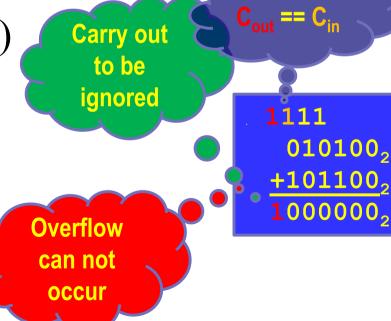
Overflow might occur, and indeed it did in this case

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

**■** *Example 7*:

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

- $\blacksquare 20_{10} \rightarrow 10100_2$
- $20_{10} 20_{10} \rightarrow 10100_2 10100_2$ 
  - $\rightarrow$  010100<sub>2</sub> 010100<sub>2</sub>
  - $\rightarrow$  010100<sub>2</sub> + (-010100<sub>2</sub>)
  - $\rightarrow 010100_2 + 101100_2$
  - $\rightarrow$  000000<sub>2</sub>
- needed. It is just for you to verify.



This is the answer

in 2's complement

This step is not

**■** *Example 8*:

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

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

■ 
$$31_{10} + 1_{10}$$
  $\Rightarrow$   $111111_2 + 1_2$  No carry out

Overflow might occur, and indeed it did in this case

Overflow might occur, and indeed it did in this case

**■** *Example 9*:

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

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

Carry out to be ignored

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

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

This is the answer in 2's complement

This step is not

needed. It is just,

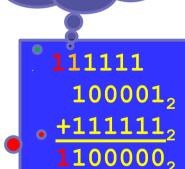
for vou to verify.

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

$$-100000_2$$

$$\rightarrow$$
 -32<sub>10</sub>

Overflow might occur, but it did not in this case



© Mahmoud R. El-Sakka

CS 2208: Introduction to Computer Organization and Architecture

**■** *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  $-3_{10}$   $-0.25_{10}$ 

- $-3_{10} 0.25_{10} \rightarrow -11_2 0.01_2$ 
  - $\rightarrow$   $(-000011_2) + (-0000.01_2)$

This is the answer in 2's complement

- $\rightarrow$  ( 111101<sub>2</sub>) + ( 1111.11<sub>2</sub>)
- **→** 111100.11<sub>2</sub>



1100.11<sub>2</sub>

 $-3.25_{10}$ 

Overflow might occur, but it did not in this case

111101.00<sub>2</sub>
.+111111.11<sub>2</sub>
1111100.11<sub>2</sub>

 $C_{out} == C_{in}$ 

111111

Binary points
MUST be
aligned

CS 2208: Intro- puter Organization and Architecture