Sétif 1 University
Faculty of Sciences
Common Core Mathematics and Computer Science

1st semester (year 2025/26)
Machine Structures 1 course
Kara Abdelaziz professor

Problem set 3 solution
(Binary Encoding)

Exercise 1:

1).
1. Encoding on 4-bits the decimal: $(+6)_{10}$

- Convert **$(+6)_{10}$** to binary using SED by 2:

Division	Remainder
6 ÷ 2 = 3	0
3 ÷ 2 = 1	1
1 ÷ 2 = 3	1

$(+6)_{10} = (+110)_2$

2. Encoding on 6-bits the decimal: $(-12)_{10}$

- Convert **$(-12)_{10}$** to binary using SED by 2:

Division	Remainder
12 ÷ 2 = 6	0
6 ÷ 2 = 3	0
3 ÷ 2 = 1	1
1 ÷ 2 = 0	1

$(-12)_{10} = (-1100)_2$

3. Encoding on 8-bits the decimal: $(+63)_{10}$

- Convert **$(+63)_{10}$** to binary using SED by 2:

Division	Remainder
63 ÷ 2 = 31	1
31 ÷ 2 = 15	1
15 ÷ 2 = 7	1
7 ÷ 2 = 3	1
3 ÷ 2 = 1	1
1 ÷ 2 = 0	1

$(-63)_{10} = (-111111)_2$

4. Encoding on 10-bits the decimal: $(-1)_{10}$

- Convert **$(-1)_{10}$** to binary using SED by 2:

Division	Remainder
1 ÷ 2 = 0	1

$(-1)_{10} = (-1)_2$

Fixed-Width	Decimal values	Binary value	Unsigned Integer	Sign-Magnitude (MSB Sign bit)	One's Complement (Invert all bits)	Two's Complement (1C +1)
4-bits	+6	+110	$[0110]_{UI-4}$	$[0110]_{SM-4}$	$[0110]_{1C-4}$	$[0110]_{2C-4}$
6-bits	-12	-1100	N.A.	$[100110]_{SM-6}$	$[110011]_{1C-6}$	$[110100]_{2C-6}$
8-bits	+63	+111111	$[00111111]_{UI-8}$	$[00111111]_{SM-8}$	$[00111111]_{1C-8}$	$[00111111]_{2C-8}$
10-bits	-1	-1	N.A.	$[1000000001]_{SM-10}$	$[1111111110]_{1C-10}$	$[1111111111]_{2C-10}$

5. Decoding the 4-bits binary $[0110]_{4\text{-bits}}$:

- Decode **$[0110]_{UI-4}$** to Decimal is to convert $(0110)_2$ to Decimal using PF:

  $(0x2^3) + (1x2^2) + (1x2^1) + (0x2^0) = 0 + 4 + 2 + 0 =$ **$(6)_{10}$**

$[0110]_{UI-4} = (6)_{10}$

- Decode **$[0110]_{SM-4}$** to Decimal is to convert $(110)_2$ to Decimal using PF then adding sign:

  sign = 0 → positve number **(+)**
  $(1x2^2) + (1x2^1) + (0x2^0) = 4 + 2 + 0 =$ **$(6)_{10}$**

$[0110]_{UI-4} = (+6)_{10}$

- Decode **$[0110]_{1C-4}$** to Decimal, knowing sign is positive is to convert $(110)_2$ to Decimal using PF:

  sign = 0 → positve number **(+)**
  $(1x2^2) + (1x2^1) + (0x2^0) = 4 + 2 + 0 =$ **$(6)_{10}$**

$[0110]_{1C-4} = (+6)_{10}$

- Decode **$[0110]_{2C-4}$** to Decimal, knowing sign is positive is to convert $(110)_2$ to Decimal using PF:

  sign = 0 → positve number **(+)**
  $(1x2^2) + (1x2^1) + (0x2^0) = 4 + 2 + 0 =$ **$(6)_{10}$**

$[0110]_{2C-4} = (+6)_{10}$

6. Decoding the 6-bits binary $[110101]_{6\text{-bits}}$:

- Decode **$[110101]_{UI-6}$** to Decimal is to convert $(110101)_2$ to Decimal using PF:
  $(1x2^5) + (1x2^4) + (0x2^3) + (1x2^2) + (0x2^1) + (1x2^0) = 32 + 16 + 4 + 1 =$ **$(53)_{10}$**

$[110101]_{UI\text{-}6} = (53)_{10}$

- Decode **$[110101]_{SM\text{-}6}$** to Decimal is to convert the magnitude $(10101)_2$ to Decimal using PF then adding sign:

  sign = 1 → negative number **(-)**
  $(1x2^4) + (0x2^3) + (1x2^2) + (0x2^1) + (1x2^0) = 16 + 4 + 1 =$ **$(21)_{10}$**

$[110101]_{SM\text{-}6} = (-21)_{10}$

- Decode **$[110101]_{1C\text{-}6}$** to Decimal, knowing sign is negative is to invert all bits in $(110101)_2$ then convert it to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(110101)_2 \xrightarrow{invert} (001010)_2$
  $(1x2^3) + (0x2^2) + (1x2^1) + (0x2^0) = 8 + 2 =$ **$(10)_{10}$**

$[110101]_{1C\text{-}6} = (-10)_{10}$

- Decode **$[110101]_{2C\text{-}6}$** to Decimal, knowing sign is negative is to invert all bits in $(110101)_2$ to get 1C, then adding +1 to convert it after to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(110101)_2 \xrightarrow{1C} (001010)_2 \xrightarrow{+1} (001011)_2$
  $(1x2^3) + (0x2^2) + (1x2^1) + (1x2^0) = 8 + 2 =$ **$(11)_{10}$**

$[110101]_{2C\text{-}6} = (-11)_{10}$

7. Decoding the 8-bits binary $[10000000]_{8\text{-}bits}$:

- Decode **$[10000000]_{UI\text{-}8}$** to Decimal is to convert $(10000000)_2$ to Decimal using PF:

  $(1x2^7) + (0x2^6) + (0x2^5) + (0x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (0x2^0) =$ **$(128)_{10}$**

$[10000000]_{UI\text{-}8} = (128)_{10}$

- Decode **$[10000000]_{SM\text{-}8}$** to Decimal is to convert the magnitude $(0000000)_2$ to Decimal using PF then adding sign:

  sign = 1 → negative number **(-)**
  $(0x2^6) + (0x2^5) + (0x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (0x2^0) =$ **$(0)_{10}$**

$[10000000]_{SM\text{-}8} = (-0)_{10}$

- Decode **$[10000000]_{1C\text{-}8}$** to Decimal, knowing sign is negative is to invert all bits in $(10000000)_2$ then convert it to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(10000000)_2 \xrightarrow{invert} (01111111)_2$
  $(0x2^7) + (1x2^6) + (1x2^5) + (1x2^4) + (1x2^3) + (1x2^2) + (1x2^1) + (1x2^0) =$ **$(127)_{10}$**

$[10000000]_{1C\text{-}8} = (-127)_{10}$

- Decode **[10000000]$_{2C-8}$** to Decimal, knowing sign is negative is to invert all bits in $(10000000)_2$ to get 1C, then adding +1 to convert it after to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(10000000)_2 \xrightarrow{1C} (01111111)_2 \xrightarrow{+1} (10000000)_2$
  $(1x2^7) + (0x2^6) + (0x2^5) + (0x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (0x2^0)$ = **$(128)_{10}$**

[10000000]$_{2C-8}$ = $(-128)_{10}$

8. Decoding the 10-bits binary $[1111111111]_{10\text{-bits}}$:

- Decode **[1111111111]$_{UI-10}$** to Decimal is to convert $(1111111111)_2$ to Decimal using PF:

  $(1x2^9) + (1x2^8) + (1x2^7) + (1x2^6) + (1x2^5) + (1x2^4) + (1x2^3) + (1x2^2) + (1x2^1) + (1x2^0)$
  $= 2^{10} - 1$ = **$(1023)_{10}$**

[1111111111]$_{UI-10}$ = $(1023)_{10}$

- Decode **[1111111111]$_{SM-10}$** to Decimal is to convert the magnitude $(111111111)_2$ to Decimal using PF then adding sign:

  sign = 1 → negative number **(-)**
  $(1x2^8) + (1x2^7) + (1x2^6) + (1x2^5) + (1x2^4) + (1x2^3) + (1x2^2) + (1x2^1) + (1x2^0)$
  $= 2^9 - 1$ = **$(511)_{10}$**

[1111111111]$_{SM-10}$ = $(-511)_{10}$

- Decode **[1111111111]$_{1C-10}$** to Decimal, knowing sign is negative is to invert all bits in $(1111111111)_2$ then convert it to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(1111111111)_2 \xrightarrow{invert} (0000000000)_2$
  $(0x2^9) + (0x2^8) + (0x2^7) + (0x2^6) + (0x2^5) + (0x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (0x2^0)$
  = **$(0)_{10}$**

[1111111111]$_{1C-10}$ = $(-0)_{10}$

- Decode **[1111111111]$_{2C-10}$** to Decimal, knowing sign is negative is to invert all bits in $(1111111111)_2$ to get 1C, then adding +1 to convert it after to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(1111111111)_2 \xrightarrow{1C} (0000000000)_2 \xrightarrow{+1} (0000000001)_2$
  $(0x2^9) + (0x2^8) + (0x2^7) + (0x2^6) + (0x2^5) + (0x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (1x2^0)$
  = **$(1)_{10}$**

[1111111111]$_{2C-10}$ = $(-1)_{10}$

2). To get the number of representable values and the ranges for 4-bits, 6-bits, 8-bits, 10-bits, we can use the table below summarizes all the formulas related to representable values and ranges for each integer scheme:

Formulas for N-bits	Unsigned Integer (UI)	Sign-Magnitude (SM)	One's Complement (1C)	Two's Complement (2C)
|values|=	2^N	2^N-1	2^N-1	2^N
Range=	[0, 2^N-1]	[-(2^{N-1}-1), +(2^{N-1}-1)]	[-(2^{N-1}-1), +(2^{N-1}-1)]	[-(2^{N-1}), +(2^{N-1}-1)]

- For 4-bits:

Formulas for 4-bits	Unsigned Integer (UI)	Sign-Magnitude (SM)	One's Complement (1C)	Two's Complement (2C)
|values|=	2^4=**16**	2^4-1=**15**	2^4-1=**15**	2^4=**16**
Range=	[0, 2^4-1] = **[0, 15]**	[-(2^{4-1}-1), +(2^{4-1}-1)] = **[-7, +7]**	[-(2^{4-1}-1), +(2^{4-1}-1)] = **[-7, +7]**	[-(2^{4-1}), +(2^{4-1}-1)] = **[-8, +7]**

- For 6-bits:

Formulas for 6-bits	Unsigned Integer (UI)	Sign-Magnitude (SM)	One's Complement (1C)	Two's Complement (2C)
|values|=	2^6=**64**	2^6-1=**63**	2^6-1=**63**	2^6=**64**
Range=	[0, 2^6-1] = **[0, 63]**	[-(2^{6-1}-1), +(2^{6-1}-1)] = **[-31, +31]**	[-(2^{6-1}-1), +(2^{6-1}-1)] = **[-31, +31]**	[-(2^{6-1}), +(2^{6-1}-1)] = **[-32, +31]**

- For 8-bits:

Formulas for 8-bits	Unsigned Integer (UI)	Sign-Magnitude (SM)	One's Complement (1C)	Two's Complement (2C)
|values|=	2^8=**256**	2^8-1=**255**	2^8-1=**255**	2^8=**256**
Range=	[0, 2^8-1] = **[0, 255]**	[-(2^{8-1}-1), +(2^{8-1}-1)] = **[-127, +127]**	[-(2^{8-1}-1), +(2^{8-1}-1)] = **[-127, +127]**	[-(2^{8-1}), +(2^{8-1}-1)] = **[-128, +127]**

- For 10-bits:

Formulas for 10-bits	Unsigned Integer (UI)	Sign-Magnitude (SM)	One's Complement (1C)	Two's Complement (2C)
|values|=	2^{10}=**1024**	2^{10}-1=**1023**	2^{10}-1=**1023**	2^{10}=**1024**
Range=	[0, 2^{10}-1] = **[0, 1023]**	[-(2^{10-1}-1), +(2^{10-1}-1)] = **[-511, +511]**	[-(2^{10-1}-1), +(2^{10-1}-1)] = **[-511, +511]**	[-(2^{10-1}), +(2^{10-1}-1)] = **[-512, +511]**

Exercise 2:

1). The minimum number of bits (N) required to represent +45 and +128 is calculated using the reverse formula of the power 2 function, which is the log_2 function.

We have:

$$2^n - 1 = max \Rightarrow 2^n = max + 1$$
$$\Rightarrow log_2(2^n) = log_2(max + 1)$$
$$\Rightarrow n = ln(max + 1)/ln(2)$$

Knowing that:

$$log_p(x) = ln(x)/ln(p)$$

That makes:

- $Log_2(45+1) = ln(46)/ln(2) = 5.5235 \approx$ **6 bits** (round up)

$(+45)_{10}$ needs **6-bits**.

And:

- $Log_2(128+1) = ln(129)/ln(2) = 7.0112 \approx$ **8 bits** (round up)

$(+128)_{10}$ needs **8-bits**.

<u>Remark 1</u>: We need to *round up* because we don't have a fractional number to count a bits number, we need a natural number that directly succeed the fractional number.

<u>Remark 2</u>: If the calculator is not accessible to calculate the logarithm. The easiest way is to convert the Decimal number to binary using SED, then count the bits. Or to look for the power of 2 directly greeter than the Decimal number, and conclude the bits number.

2). To determine the number of bits (N) required to represent +45 and -128 in all encoding schemes, we need to use the range formula for each scheme.

- Unsigned Integer:

  Comparing +45 to the range = **$[0,2^N-1]$**
  we get:
  $+45 \leq 2^N-1 \rightarrow 46 \leq 2^N \rightarrow N \geq 6$ (we take the smallest)
  +45 requires N = 6-bits in UI

  Comparing -128 to the range = **$[0,2^N-1]$**
  we observe:
  Unsigned Integer can't be negative.
  128 can't be represented in UI.

We take **N = 6-bits in UI**

- Sign-Magnitude:

  Comparing +45 to the range = **$[-(2^{N-1}-1), +(2^{N-1}-1)]$**
  we get:
  $+45 \leq 2^{N-1}-1 \rightarrow 46 \leq 2^{N-1} \rightarrow N-1 \geq 6 \rightarrow N \geq 7$ (we take the smallest)
  +45 requires N = 7-bits in SM

  Comparing -128 to the range = **$[-(2^{N-1}-1), +(2^{N-1}-1)]$**
  we get:
  $-128 \geq -(2^{N-1}-1) \rightarrow -128 \geq -2^{N-1}+1 \rightarrow -129 \geq -2^{N-1} \rightarrow +129 \leq +2^{N-1} \rightarrow N-1 \geq 8$
  $\rightarrow N \geq 9$ (we take the smallest)
  -128 requires N = 9-bits in SM

For both we take the bigger: **N = 9-bits in SM**

- One's Complement:

  Comparing +45 to the range = **$[-(2^{N-1}-1), +(2^{N-1}-1)]$**
  we get:
  $+45 \leq 2^{N-1}-1 \rightarrow 46 \leq 2^{N-1} \rightarrow N-1 \geq 6 \rightarrow N \geq 7$ (we take the smallest)
  +45 requires N = 7-bits in 1C

  Comparing -128 to the range = **$[-(2^{N-1}-1), +(2^{N-1}-1)]$**
  we get:
  $-128 \geq -(2^{N-1}-1) \rightarrow -128 \geq -2^{N-1}+1 \rightarrow -129 \geq -2^{N-1} \rightarrow +129 \leq +2^{N-1} \rightarrow N-1 \geq 8$
  $\rightarrow N \geq 9$ (we take the smallest)
  -128 requires N = 9-bits in 1C

For both we take the bigger: **N = 9-bits in 1C**

- Two's Complement:

  Comparing +45 to the range = **$[-(2^{N-1}), +(2^{N-1}-1)]$**
  we get:
  $+45 \leq 2^{N-1}-1 \rightarrow 46 \leq 2^{N-1} \rightarrow N-1 \geq 6 \rightarrow N \geq 7$ (we take the smallest)
  +45 requires N = 7-bits

  Comparing -128 to the range = **$[-(2^{N-1}), +(2^{N-1}-1)]$**
  we get:
  $-128 \geq -(2^{N-1}) \rightarrow -128 \geq -2^{N-1} \rightarrow +128 \leq +2^{N-1} \rightarrow N-1 \geq 7$
  $\rightarrow N \geq 8$ (we take the smallest)
  -128 requires N = 8-bits

For both we take the bigger: **N = 8-bits in 2C**

Exercise 3:

1). Additions and subtractions on 8-bits:

1. $(75)_{10} + (37)_{10}$ in 2C is $(+75)_{10} + (+37)_{10}$:

- Convert $(75)_{10}$ to 8-bits Two's Complement by first converting $(75)_{10}$ to binary using SED by 2. Knowing the number is positive, the binary don't change in Two's Complement:

Division	Remainder
75 ÷ 2 = 37	1
37 ÷ 2 = 18	1
18 ÷ 2 = 9	0
9 ÷ 2 = 4	1
4 ÷ 2 = 2	0
2 ÷ 2 = 1	0
1 ÷ 2 = 0	1

  positive number **(+)** → sign = 0
  $(+75)_{10} = (+1001011)_2 =$ **$[01001011]_{2C\text{-}8}$**

- Convert $(+37)_{10}$ to 8-bits Two's Complement by first converting $(37)_{10}$ to binary using SED by 2. Knowing the number is positive, the binary don't change in Two's Complement:

Division	Remainder
37 ÷ 2 = 18	1
18 ÷ 2 = 9	0
9 ÷ 2 = 4	1
4 ÷ 2 = 2	0
2 ÷ 2 = 1	0
1 ÷ 2 = 0	1

positive number **(+)** → sign = 0
$(+37)_{10} = (+100101)_2 =$ **$[00100101]_{2C\text{-}8}$**

- The addition $[01001011]_{2C\text{-}8} + [00100101]_{2C\text{-}8}$:

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

$[01001011]_{2C\text{-}8} + [00100101]_{2C\text{-}8} =$ **$[01110000]_{2C\text{-}8}$**

- Decode $[01110000]_{2C\text{-}8}$ to Decimal, knowing sign is positive is to convert $(1110000)_2$ to Decimal using PF:

sign = 0 → positve number **(+)**
$(1x2^6) + (1x2^5) + (1x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (0x2^0) =$ **$(112)_{10}$**

$[01110000]_{2C\text{-}8} =$ **$(+112)_{10}$**

- Knowing the range of 2C 8-bits is [-128, +127], the addition $(75)_{10} + (37)_{10} = (112)_{10}$, produces a **correct result**. The result belongs to the range, there is **no overflow**.

2. $(91)_{10} + (53)_{10}$ in 2C is $(+91)_{10} + (+53)_{10}$:

- Convert $(91)_{10}$ to 8-bits Two's Complement by first converting $(91)_{10}$ to binary using SED by 2. Knowing the number is positive, the binary don't change in Two's Complement:

Division	Remainder
91 ÷ 2 = 45	1
45 ÷ 2 = 22	1
22 ÷ 2 = 11	0
11 ÷ 2 = 5	1
5 ÷ 2 = 2	1
2 ÷ 2 = 1	0
1 ÷ 2 = 0	1

positive number **(+)** → sign = 0
$(+91)_{10} = (+1011011)_2 =$ **$[01011011]_{2C\text{-}8}$**

- Convert $(+53)_{10}$ to 8-bits Two's Complement by first converting $(53)_{10}$ to binary using SED by 2. Knowing the number is positive, the binary don't change in Two's Complement:

Division	Remainder
53 ÷ 2 = 26	1
26 ÷ 2 = 13	0
13 ÷ 2 = 6	1
6 ÷ 2 = 3	0
3 ÷ 2 = 1	1
1 ÷ 2 = 0	1

positive number **(+)** → sign = 0
$(+53)_{10} = (+110101)_2 =$ **$[00110101]_{2C\text{-}8}$**

- The addition $[01011011]_{2C\text{-}8} + [00110101]_{2C\text{-}8}$:

$$\begin{array}{rcccccccc} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & \\ & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ + & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline = & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}$$

$[01011011]_{2C\text{-}8} + [00110101]_{2C\text{-}8} =$ **$[10010000]_{2C\text{-}8}$**

- Decode $[10010000]_{2C\text{-}8}$ to Decimal, knowing sign is negative is to invert all bits in $(10010000)_2$ to get 1C, then adding +1 to convert it after to Decimal using PF:

sign = 1 → negative number **(-)**
$(10010000)_2 \xrightarrow{1C} (01101111)_2 \xrightarrow{+1} (01110000)_2$
$(0x2^7) + (1x2^6) + (1x2^5) + (1x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (0x2^0) =$ **$(112)_{10}$**

$[10010000]_{2C\text{-}8} =$ **$(-112)_{10}$**

- Knowing the range of 2C 8-bits is [-128, +127], the addition should be $(91)_{10} + (53)_{10} = (+144)_{10}$, but produces $(-112)_{10}$ which is an **incorrect result**. The result don't belongs to the range, this is an **overflow**.

Remark 3: There is another way much simpler to detect overflows. In the 2 cases, if a positive number is added to another positive number produces a negative number. Or the case if a negative number added to another negative number produces a positive number.

3. $(112)_{10} - (65)_{10}$ in 2C is $(+112)_{10} + (-65)_{10}$:

- Convert $(112)_{10}$ to 8-bits Two's Complement by first converting $(112)_{10}$ to binary using SED by 2. Knowing the number is positive, the binary don't change in Two's Complement:

Division	Remainder
112 ÷ 2 = 56	0
56 ÷ 2 = 28	0
28 ÷ 2 = 14	0
14 ÷ 2 = 7	0
7 ÷ 2 = 3	1
3 ÷ 2 = 1	1
1 ÷ 2 = 0	1

positive number **(+)** → sign = 0
$(+112)_{10} = (+1110000)_2 = \mathbf{[01110000]_{2C\text{-}8}}$

- Convert $(-65)_{10}$ to 8-bits Two's Complement by first converting $(+65)_{10}$ to binary using SED by 2. Knowing the number is negative, all the bits in the binary needs to be inverted to get 1C, then adding +1:

Division	Remainder
65 ÷ 2 = 32	1
32 ÷ 2 = 16	0
16 ÷ 2 = 8	0
8 ÷ 2 = 4	0
4 ÷ 2 = 2	0
2 ÷ 2 = 1	0
1 ÷ 2 = 0	1

$(+65)_{10} = (+1000001)_2$

negative number **(-)** → sign = 1
$(01000001)_2 \xrightarrow{1C} (10111110)_2 \xrightarrow{+1} (10111111)_2$
$(-65)_{10} = \mathbf{[10111111]_{2C\text{-}8}}$

- The addition $[01110000]_{2C\text{-}8} + [10111111]_{2C\text{-}8}$:

```
  1 1 1 1
    0 1 1 1 0 0 0 0
+   1 0 1 1 1 1 1 1
= X|0 0 1 0 1 1 1 1
```

$[01011011]_{2C\text{-}8} + [00110101]_{2C\text{-}8} = \mathbf{[00101111]_{2C\text{-}8}}$

Remark 4: Like we discussed in lecture, the case of the additional bit in the addition of Two's Complement. In this case, the 9th bit crossed in the result. It is always ignored, and doesn't systematically mean an overflow.

- Decode $[00101111]_{2C\text{-}8}$ to Decimal, knowing sign is positive is to convert $(00101111)_2$ to Decimal using PF:

  sign = 0 → positive number **(+)**
  $(0x2^7) + (0x2^6) + (1x2^5) + (0x2^4) + (1x2^3) + (1x2^2) + (1x2^1) + (1x2^0) = \mathbf{(47)_{10}}$

$[00101111]_{2C\text{-}8} = \mathbf{(+47)_{10}}$

- Knowing the range of 2C 8-bits is [-128, +127], the subtraction should be $(112)_{10} - (65)_{10} = (+47)_{10}$ produces a **correct result**. The result belongs to the range, there is **no overflow**.

4. $(-46)_{10} - (101)_{10}$ in 2C is $(-46)_{10} + (-101)_{10}$:

- Convert $(-46)_{10}$ to 8-bits Two's Complement by first converting $(+46)_{10}$ to binary using SED by 2. Knowing the number is negative, all the bits in the binary needs to be inverted to get 1C, then adding +1:

Division	Remainder
46 ÷ 2 = 23	0
23 ÷ 2 = 11	1
11 ÷ 2 = 5	1
5 ÷ 2 = 2	1
2 ÷ 2 = 1	0
1 ÷ 2 = 0	1

$(+46)_{10} = (+101110)_2$

negative number **(-)** → sign = 1
$(00101110)_2 \xrightarrow{1C} (11010001)_2 \xrightarrow{+1} (11010010)_2$
$(-46)_{10}$ = **$[11010010]_{2C\text{-}8}$**

- Convert $(-101)_{10}$ to 8-bits Two's Complement by first converting $(-101)_{10}$ to binary using SED by 2. Knowing the number is negative, all the bits in the binary needs to be inverted to get 1C, then adding +1:

Division	Remainder
101 ÷ 2 = 50	1
50 ÷ 2 = 25	0
25 ÷ 2 = 12	1
12 ÷ 2 = 6	0
6 ÷ 2 = 3	0
3 ÷ 2 = 1	1
1 ÷ 2 = 0	1

$(+101)_{10} = (+1100101)_2$

negative number **(-)** → sign = 1
$(01100101)_2 \xrightarrow{1C} (10011010)_2 \xrightarrow{+1} (10011011)_2$
$(-101)_{10}$ = **$[10011011]_{2C\text{-}8}$**

- The addition $[11010010]_{2C\text{-}8} + [10011011]_{2C\text{-}8}$:

```
    1   1   1
    1 1 0 1 0 0 1 0
+   1 0 0 1 1 0 1 1
-------------------
= X 0 1 1 0 1 1 0 1
```

$[11010010]_{2C\text{-}8} + [10011011]_{2C\text{-}8}$ = **$[01101101]_{2C\text{-}8}$**

- Decode $[01101101]_{2C\text{-}8}$ to Decimal, knowing sign is positive is to convert $(01101101)_2$ to Decimal using PF:

  sign = 0 → positive number **(+)**
  $(0x2^7) + (1x2^6) + (1x2^5) + (0x2^4) + (1x2^3) + (1x2^2) + (0x2^1) + (1x2^0)$ = **$(109)_{10}$**

$[01101101]_{2C\text{-}8}$ = **$(+109)_{10}$**

- Knowing the range of 2C 8-bits is [-128, +127], the subtraction should be $(-46)_{10} - (101)_{10} = (-147)_{10}$, but produces $(+109)_{10}$ which is an **incorrect result**. The result don't belongs to the range, this is an **overflow**.

2). Let perform the 4 binary additions on 8-bits:

$$\begin{array}{r} {\scriptstyle 1} \\ 0\,0\,0\,1\,0\,1\,0\,0 \\ +\ 0\,0\,1\,0\,0\,1\,1\,1 \\ \hline = \ 0\,0\,1\,1\,1\,0\,1\,1 \end{array} \qquad \begin{array}{r} {\scriptstyle 1\ 1\ 1} \\ 0\,0\,0\,1\,0\,1\,0\,1 \\ +\ 1\,1\,1\,0\,0\,1\,1\,1 \\ \hline = \ 1\,1\,1\,1\,1\,1\,0\,0 \end{array}$$

$$\begin{array}{r} {\scriptstyle } \\ 1\,0\,0\,0\,0\,0\,0\,1 \\ +\ 0\,0\,0\,1\,0\,1\,1\,0 \\ \hline = \ 1\,0\,0\,1\,0\,1\,1\,1 \end{array} \qquad \begin{array}{r} {\scriptstyle 1} \\ 1\,1\,1\,1\,1\,1\,1\,1 \\ +\ 1\,0\,0\,0\,0\,0\,0\,0 \\ \hline = \ \cancel{1}\,0\,1\,1\,1\,1\,1\,1\,1 \end{array}$$

1. We need to interpret the binary numbers as encoded on 2C and UI:

1.1. The addition $[00010100]_{8\text{-bits}} + [00100111]_{8\text{-bits}} = [00111011]_{8\text{-bits}}$ interpreted as 2C:

- Decoding $[00010100]_{2C\text{-}8}$ to Decimal, knowing sign is positive is to convert $(00010100)_2$ to Decimal using PF:

  sign = 0 → positive number **(+)**
  $(0x2^7) + (0x2^6) + (0x2^5) + (1x2^4) + (0x2^3) + (1x2^2) + (0x2^1) + (0x2^0) = \mathbf{(20)}_{10}$

$[00010100]_{2C\text{-}8} = \mathbf{(+20)}_{10}$

- Decoding $[00100111]_{2C\text{-}8}$ to Decimal, knowing sign is positive is to convert $(00100111)_2$ to Decimal using PF:

  sign = 0 → positive number **(+)**
  $(0x2^7) + (0x2^6) + (1x2^5) + (0x2^4) + (0x2^3) + (1x2^2) + (1x2^1) + (1x2^0) = \mathbf{(39)}_{10}$

$[00100111]_{2C\text{-}8} = \mathbf{(+39)}_{10}$

- Decoding $[00111011]_{2C\text{-}8}$ to Decimal, knowing sign is positive is to convert $(00111011)_2$ to Decimal using PF:

  sign = 0 → positive number **(+)**
  $(0x2^7) + (0x2^6) + (1x2^5) + (1x2^4) + (1x2^3) + (0x2^2) + (1x2^1) + (1x2^0) = \mathbf{(59)}_{10}$

$[00111011]_{2C\text{-}8} = \mathbf{(+59)}_{10}$

- Knowing the range of 2C 8-bits is [-128, +127], the addition should be $(+20)_{10} + (+39)_{10} = (+59)_{10}$ produces a **correct result**. The result belongs to the range, there is **no overflow**.

1.2. The addition $[00010101]_{8\text{-bits}} + [00100111]_{8\text{-bits}} = [00111011]_{8\text{-bits}}$ interpreted as UI, is the same as the addition because the 3 numbers are positives.

- $[00010100]_{2C\text{-}8} = \mathbf{(+20)}_{10}$
- $[00100111]_{2C\text{-}8} = \mathbf{(+39)}_{10}$
- $[00111011]_{2C\text{-}8} = \mathbf{(+59)}_{10}$

- Knowing the range of UI 8-bits is [0, +255], the addition should be $(+20)_{10}$ + $(+39)_{10}$ = $(+59)_{10}$ produces a **correct result**. The result belongs to the range, there is **no overflow**.

2.1. The addition $[00010101]_{8\text{-bits}}$ + $[11100111]_{8\text{-bits}}$ = $[11111100]_{8\text{-bits}}$ interpreted as 2C:

- Decoding $[00010101]_{2C\text{-}8}$ to Decimal, knowing sign is positive is to convert $(00010101)_2$ to Decimal using PF:

  sign = 0 → positive number **(+)**
  $(0x2^7) + (0x2^6) + (0x2^5) + (1x2^4) + (0x2^3) + (1x2^2) + (0x2^1) + (1x2^0)$ = **$(21)_{10}$**

$[00010101]_{2C\text{-}8}$ = **$(+21)_{10}$**

- Decoding $[11100111]_{2C\text{-}8}$ to Decimal, knowing sign is negative is to invert all bits in $(11100111)_2$ to get 1C, then adding +1 to convert it after to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(11100111)_2 \xrightarrow{1C} (00011000)_2 \xrightarrow{+1} (00011001)_2$
  $(0x2^7) + (0x2^6) + (0x2^5) + (1x2^4) + (1x2^3) + (0x2^2) + (0x2^1) + (1x2^0)$ = **$(25)_{10}$**

$[11100111]_{2C\text{-}8}$ = **$(-25)_{10}$**

- Decoding $[11111100]_{2C\text{-}8}$ to Decimal, knowing sign is negative is to invert all bits in $(11111100)_2$ to get 1C, then adding +1 to convert it after to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(11111100)_2 \xrightarrow{1C} (00000011)_2 \xrightarrow{+1} (00000100)_2$
  $(0x2^7) + (0x2^6) + (0x2^5) + (0x2^4) + (0x2^3) + (1x2^2) + (0x2^1) + (0x2^0)$ = **$(4)_{10}$**

$[11100111]_{2C\text{-}8}$ = **$(-4)_{10}$**

- Knowing the range of 2C 8-bits is [-128, +127], the addition should be $(+21)_{10}$ + $(-25)_{10}$ = $(-4)_{10}$ produces a **correct result**. The result belongs to the range, there is **no overflow**.

2.2. The addition $[00010101]_{8\text{-bits}}$ + $[11100111]_{8\text{-bits}}$ = $[11111100]_{8\text{-bits}}$ interpreted as UI:

- Decoding $[00010101]_{UI\text{-}8}$ to Decimal, knowing sign is positive is the same as $[00010101]_{2C\text{-}8}$:

$[00010101]_{UI\text{-}8}$ = **$(+21)_{10}$**

- Decoding $[11100111]_{UI\text{-}8}$ to Decimal, is to convert $(11100111)_2$ to Decimal using PF:

  $(1x2^7) + (1x2^6) + (1x2^5) + (0x2^4) + (0x2^3) + (1x2^2) + (1x2^1) + (1x2^0)$ = **$(231)_{10}$**

$[00010101]_{UI\text{-}8}$ = **$(+231)_{10}$**

- Decoding $[11111100]_{UI\text{-}8}$ to Decimal, is to convert $(11111100)_2$ to Decimal using PF:

  $(1x2^7) + (1x2^6) + (1x2^5) + (1x2^4) + (1x2^3) + (1x2^2) + (0x2^1) + (0x2^0)$ = **$(252)_{10}$**

$[00010101]_{UI\text{-}8}$ = **$(+252)_{10}$**

- Knowing the range of UI 8-bits is [0, +255], the addition should be $(21)_{10}$ + $(231)_{10}$ = $(252)_{10}$ produces a **correct result**. The result belongs to the range, there is **no overflow**.

Remark 5: Important point raised here confirming what we studied in lecture. Is that the same adder produces correct result for the same binary number interpreted in 2 different representations 2C and UI.

3.1. The addition $[10000001]_{8\text{-bits}}$ + $[00010110]_{8\text{-bits}}$ = $[10010111]_{8\text{-bits}}$ interpreted as 2C:

- Decoding $[10000001]_{2C\text{-}8}$ to Decimal, knowing sign is negative is to invert all bits in $(10000001)_2$ to get 1C, then adding +1 to convert it after to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(10000001)_2 \xrightarrow{1C} (01111110)_2 \xrightarrow{+1} (01111111)_2$
  $(0x2^7) + (1x2^6) + (1x2^5) + (1x2^4) + (1x2^3) + (1x2^2) + (1x2^1) + (1x2^0)$ = **$(127)_{10}$**

$[10000001]_{2C\text{-}8}$ = **$(-127)_{10}$**

- Decoding $[00010110]_{2C\text{-}8}$ to Decimal, knowing sign is positive is to convert $(00010110)_2$ to Decimal using PF:

  sign = 0 → positive number **(+)**
  $(0x2^7) + (0x2^6) + (0x2^5) + (1x2^4) + (0x2^3) + (1x2^2) + (1x2^1) + (0x2^0)$ = **$(22)_{10}$**

$[00010110]_{2C\text{-}8}$ = **$(+22)_{10}$**

- Decoding $[10010111]_{2C\text{-}8}$ to Decimal, knowing sign is negative is to invert all bits in $(10010111)_2$ to get 1C, then adding +1 to convert it after to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(10010111)_2 \xrightarrow{1C} (01101000)_2 \xrightarrow{+1} (01101001)_2$
  $(0x2^7) + (1x2^6) + (1x2^5) + (0x2^4) + (1x2^3) + (0x2^2) + (0x2^1) + (1x2^0)$ = **$(105)_{10}$**

$[10010111]_{2C\text{-}8}$ = **$(-105)_{10}$**

- Knowing the range of 2C 8-bits is [-128, +127], the addition should be $(-127)_{10}$ + $(+22)_{10}$ = $(-105)_{10}$ produces a **correct result**. The result belongs to the range, there is **no overflow**.

3.2. The addition $[10000001]_{8\text{-bits}}$ + $[00010110]_{8\text{-bits}}$ = $[10010111]_{8\text{-bits}}$ interpreted as UI:

- Decoding $[10000001]_{UI\text{-}8}$ to Decimal, is to convert $(10000001)_2$ to Decimal using PF:

  $(1x2^7) + (0x2^6) + (0x2^5) + (0x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (1x2^0)$ = **$(129)_{10}$**

$[00010101]_{UI\text{-}8}$ = **$(+129)_{10}$**

- Decoding $[00010110]_{UI\text{-}8}$ to Decimal, knowing sign is positive is the same as $[00010110]_{2C\text{-}8}$:

$[00010110]_{UI-8}$ = **$(+22)_{10}$**

- Decoding $[10010111]_{UI-8}$ to Decimal, is to convert $(10010111)_2$ to Decimal using PF:

  $(1x2^7) + (0x2^6) + (0x2^5) + (1x2^4) + (0x2^3) + (1x2^2) + (1x2^1) + (1x2^0)$ = **$(252)_{10}$**

$[10010111]_{UI-8}$ = **$(+151)_{10}$**

- Knowing the range of UI 8-bits is [0, +255], the addition should be $(129)_{10} + (22)_{10} = (151)_{10}$ produces a **correct result**. The result belongs to the range, there is **no overflow**.

4.1. The addition $[11111111]_{8\text{-bits}} + [10000000]_{8\text{-bits}} = [01111111]_{8\text{-bits}}$ interpreted as 2C:

- Decoding $[11111111]_{2C-8}$ to Decimal, knowing sign is negative is to invert all bits in $(11111111)_2$ to get 1C, then adding +1 to convert it after to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(11111111)_2 \xrightarrow{1C} (00000000)_2 \xrightarrow{+1} (00000001)_2$
  $(0x2^7) + (0x2^6) + (0x2^5) + (0x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (1x2^0)$ = **$(127)_{10}$**

$[11111111]_{2C-8}$ = **$(-1)_{10}$**

- Decoding $[10000000]_{2C-8}$ to Decimal, knowing sign is negative is to invert all bits in $(10000000)_2$ to get 1C, then adding +1 to convert it after to Decimal using PF:

  sign = 1 → negative number **(-)**
  $(10000000)_2 \xrightarrow{1C} (01111111)_2 \xrightarrow{+1} (10000000)_2$
  $(1x2^7) + (0x2^6) + (0x2^5) + (0x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (0x2^0)$ = **$(128)_{10}$**

$[10000000]_{2C-8}$ = **$(-128)_{10}$**

- Decoding $[01111111]_{2C-8}$ to Decimal, knowing sign is positive is to convert $(01111111)_2$ to Decimal using PF:

  sign = 0 → positive number **(+)**
  $(0x2^7) + (1x2^6) + (1x2^5) + (1x2^4) + (1x2^3) + (1x2^2) + (1x2^1) + (1x2^0)$ = **$(127)_{10}$**

$[01111111]_{2C-8}$ = **$(+127)_{10}$**

- Knowing the range of 2C 8-bits is [-128, +127], the addition should be $(-1)_{10} + (-128)_{10} = (-129)_{10}$, but produces $(+127)_{10}$ which is an **incorrect result**. The result don't belongs to the range, this is an **overflow**.

4.2. The addition $[11111111]_{8\text{-bits}} + [10000000]_{8\text{-bits}} = [01111111]_{8\text{-bits}}$ interpreted as UI:

- Decoding $[11111111]_{UI-8}$ to Decimal, is to convert $(11111111)_2$ to Decimal using PF:

  $(1x2^7) + (1x2^6) + (1x2^5) + (1x2^4) + (1x2^3) + (1x2^2) + (1x2^1) + (1x2^0)$ = **$(129)_{10}$**

$[11111111]_{UI-8}$ = **$(+255)_{10}$**

- Decoding $[10000000]_{UI-8}$ to Decimal, is to convert $(10000000)_2$ to Decimal using PF:

$$(1x2^7) + (0x2^6) + (0x2^5) + (0x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (0x2^0) = \mathbf{(128)_{10}}$$

$[10000000]_{UI-8} = \mathbf{(+128)_{10}}$

- Decoding $[01111111]_{UI-8}$ to Decimal, knowing sign is positive is the same as $[01111111]_{2C-8}$:

$[01111111]_{UI-8} = \mathbf{(+127)_{10}}$

- Knowing the range of UI 8-bits is [0, +255], the addition should be $(255)_{10} + (128)_{10} = (383)_{10}$, but produces $(127)_{10}$ which is an **incorrect result**. The result don't belongs to the range, this is an **overflow**.

Exercise 4:

1. Binary Single Precision decoding to Decimal:

1.1. Decoding the 32-bit binary number to decimal: $\mathbf{[41300000]_{FP-32}}$

- Convert the Single Precision from Hexadecimal to Binary:

$$[41300000]_{FP-32} = [0|100,0001,0|011,0000,0000,0000,0000,0000]_{FP-32}$$

- Extract Fields:
  S (Sign bit) = 0 → Positive number **(+)**
  E_{biased} = 10000010 → $E_b = 2^7 + 2^1 = \mathbf{(130)_{10}}$
  M = 011,0000,0000,0000,0000,0000 → **M = 011** (zeros after the last 1 don't count)

- Calculating the Real Exponent:
  Bias = $(127)_{10}$ (constant value)
  $E_{real} = E_{biased}$ - Bias = 130 - 127 = 3 → $\mathbf{E_r = 3}$

- Determine Significand:
  - Since the number is normalized, the significand is 1.M
  - Significand = $\mathbf{(1.011)_2}$

- Calculating the final value:
  The formula: $FP = (-1)^S \times 1.M \times 2^{Er}$
  $FP = +1.011 \times 2^3 = (1011)_2$
  Using FP: $(1011)_2 = (1x2^3) + (0x2^2) + (1x2^1) + (1x2^0) = \mathbf{(11)_{10}}$

1.2. Decoding the 32-bit binary number to decimal: $\mathbf{[BE500000]_{FP-32}}$

- Convert the Single Precision from Hexadecimal to Binary:

$$[BE500000]_{FP-32} = [1|011,1110,0|101,0000,0000,0000,0000,0000]_{FP-32}$$

- Extract Fields:
  S (Sign bit) = 1 → Negative number **(-)**

E_{biased} = 01111100 → $E_b = 2^6 + 2^5 + 2^4 + 2^3 + 2^2$ = **$(124)_{10}$**
M = 101,0000,0000,0000,0000,0000 → **M = 101** (zeros after the last 1 don't count)

- Calculating the Real Exponent:
  Bias = $(127)_{10}$ (constant value)
  $E_{real} = E_{biased}$ - Bias = 124 - 127 = -3 → **$E_r = -3$**

- Determine Significand:
  - Since the number is normalized, the significand is 1.M
  - Significand = **$(1.101)_2$**

- Calculating the final value:
  The formula: $FP = (-1)^S \times 1.M \times 2^{Er}$
  $FP = -1.101 \times 2^{-3} = (0.001101)_2$
  Using FP.FPF: $(1.101)_2 \times 2^{-3} = ((1x2^0) + (1x2^{-1}) + (0x2^{-2}) + (1x2^{-3})) \times 2^{-3}$
  = **$(0.203125)_{10}$**

2. Decimal to Binary Single Precision encoding:

2.1. Encoding the decimal **$(+15.25)_{10}$** to Single Precision 32-bit binary number:

- Converting the Number to Binary:

  First, convert the integer part $(15)_{10}$ to binary using SED:

Division	Remainder
15 ÷ 2 = 7	1
7 ÷ 2 = 3	1
3 ÷ 2 = 1	1
1 ÷ 2 = 0	1

  $(15)_{10} = (1111)_2$
  Next, convert the fractional part $(0.125)_{10}$ to a binary fraction using SM:

Multiplication	Integer part
0.25 x 2 = 0.5	0
0.5 x 2 = 1.0	1

  $(0.25)_{10} = (0.01)_2$
  Combine the integer and fractional parts: **$(+15.25)_{10} = (+1111.01)_2$**

- Determining the Sign bit (S):

  The number +1111.01 is positive → **S=0**

- Normalizing the Binary Number:

  The binary number must be converted to the format $1.M \times 2^{Ereal}$.
  $(1111.01)_2 = 1.11101 \times 2^3$
  Real Exponent: **$E_r = 3$**
  The Mantissa: **M = 11101**

- Calculating the Biased Exponent (E_b):

  We have: **Bias = 127**.
  The Exponent formula:
  $E_r = E_b$ - Bias → $E_b = E_r$ + Bias

$E_b = 3 + 127 =$ **$(130)_{10}$**

Now, we convert E_b to its 8-bit binary representation using SED:

Division	Remainder
130 ÷ 2 = 65	0
65 ÷ 2 = 32	1
32 ÷ 2 = 16	0
16 ÷ 2 = 8	0
8 ÷ 2 = 4	0
4 ÷ 2 = 2	0
2 ÷ 2 = 1	0
1 ÷ 2 = 0	1

$(130)_{10} = (10000010)_2$

$E_b = [10000010]_{8\text{-bits}}$

- Assembling the Final 32-bit Representation:

We have the 3 fields in a 32-bits FP representation organized as follows:

Sing bit (1-bit) | Biased Exponent (8-bits) | Mantissa (23-bits)

Binary representation: **$[0|100,0001,0|111,0100,0000,0000,0000,0000]_{FP\text{-}32}$**

In hexadecimal: **$[41740000]_{FP\text{-}32}$**

Remark 6: It is impracticable and difficult to work with 32 digits in the binary representation. This why it become very convenient to use Hexadecimal. That confirms what we learned in chapter 2 about the advantage of using of Octal and Hexadecimal.

2.2. Encoding the decimal **$(-0.0625)_{10}$** to Single Precision 32-bit binary number:

- Converting the Number to Binary:

Convert the fractional part $(0.0625)_{10}$ to a binary fraction using SM:

Multiplication	Integer part
0.0625 x 2 = 0.125	0
0.125 x 2 = 0.25	0
0.25 x 2 = 0.5	0
0.5 x 2 = 1.0	1

$(-0.0625)_{10} = (-0.001)_2$

- Determining the Sign bit (S):

The number -0.001 is negative → **S=1**

- Normalizing the Binary Number:

The binary number must be converted to the format $1.M\times2^{Ereal}$.

$(-0.0001)_2 = -1\times2^{-4}$

Real Exponent: **$E_r = -4$**

The Mantissa: **M = 0**

- Calculating the Biased Exponent (E_b):

We have: **Bias = 127**.

The Exponent formula:

$E_r = E_b - Bias \rightarrow E_b = E_r + Bias$

$E_b = -4 + 127 =$ **$(123)_{10}$**

Now, we convert E_b to its 8-bit binary representation using SED:

Division	Remainder
123 ÷ 2 = 61	1
61 ÷ 2 = 30	1
30 ÷ 2 = 15	0
15 ÷ 2 = 7	1
7 ÷ 2 = 3	1
3 ÷ 2 = 1	1
1 ÷ 2 = 0	1

$(124)_{10} = (1111011)_2$

E_b = [01111011]$_{8\text{-bits}}$

- Assembling the Final 32-bit Representation:

  We have the 3 fields in a 32-bits FP representation organized as follows:
  Sing bit (1-bit) | Biased Exponent (8-bits) | Mantissa (23-bits)
  Binary representation: **[1|011,1101,1|000,0000,0000,0000,0000,0000]$_{FP\text{-}32}$**
  In hexadecimal: **[BD800000]$_{FP\text{-}32}$**

3. Binary Double Precision decoding to Decimal of the 64-bits: **[4035000000000000]$_{FP64}$**

Knowing that **N_e = 11-bits** and **N_m = 52-bits**, and the **Bias = 1023**.

- Convert the Double Precision from Hexadecimal to Binary:

  $[4035000000000000]_{FP\text{-}64}$ = [0|100,0000,0011|0101,0000,0000,0000,0000,
  0000,0000,0000,00000000,0000,0000,0000]$_{FP\text{-}64}$

- Extract Fields:
  S (Sign bit) = 0 → Positive number **(+)**
  E_{biased} = 100,0000,0011 → $E_b = 2^{10} + 2^1 + 2^0$ = **$(1027)_{10}$**
  M = 0101,0000,0000,·····,0000,0000 → **M = 0101**(zeros after the last 1 don't count)

- Calculating the Real Exponent:
  Bias = $(1023)_{10}$ (constant value)
  $E_{real} = E_{biased}$ - Bias = 1027 - 1023 = 4 → **E_r = 4**

- Determine Significand:
  - Since the number is normalized, the significand is 1.M
  - Significand = **$(1.0101)_2$**

- Calculating the final value:
  The formula: $FP = (-1)^S \times 1.M \times 2^{Er}$
  $FP = +1.0101 \times 2^4 = (10101)_2$
  Using FP: $(10101)_2 = (1x2^4) + (0x2^3) + (1x2^2) + (0x2^1) + (1x2^0)$ = **$(21)_{10}$**

4. Decimal encoding to Binary Double Precision 64-bits the decimal $\mathbf{(-4.75)_{10}}$:

- Converting the Number to Binary:

  First, convert the integer part $(4)_{10}$ to binary using SED:

Division	Remainder
4 ÷ 2 = 2	0
2 ÷ 2 = 1	0
1 ÷ 2 = 0	1

  $\mathbf{(4)_{10} = (100)_2}$

  Next, convert the fractional part $(0.75)_{10}$ to a binary fraction using SM:

Multiplication	Integer part
0.75 x 2 = 1.5	1
0.5 x 2 = 1.0	1

  $\mathbf{(0.75)_{10} = (0.11)_2}$

  Combine the integer and fractional parts: $\mathbf{(-4.75)_{10} = (-100.11)_2}$

- Determining the Sign bit (S):

  The number -100.11 is negative → **S=1**

- Normalizing the Binary Number:

  The binary number must be converted to the format $1.M\times2^{Ereal}$.

  $(-100.11)_2 = -1.0011\times2^2$

  Real Exponent: $\mathbf{E_r = 2}$

  The Mantissa: **M = 0011**

- Calculating the Biased Exponent (E_b):

  We have: **Bias = 1023**.

  The Exponent formula:

  $E_r = E_b$ - Bias → $E_b = E_r$ + Bias

  $E_b = 2 + 1023 = (1025)_{10}$

  Now, we convert E_b to its 11-bit binary representation using SED:

Division	Remainder
1025 ÷ 2 = 512	1
512 ÷ 2 = 256	0
256 ÷ 2 = 128	0
128 ÷ 2 = 64	0
64 ÷ 2 = 32	0
32 ÷ 2 = 16	0
16 ÷ 2 = 8	0
8 ÷ 2 = 4	0
4 ÷ 2 = 2	0
2 ÷ 2 = 1	0
1 ÷ 2 = 0	1

  $(1025)_{10} = (100,0000,0001)_2$

  $\mathbf{E_b = [100,0000,0001]_{11\text{-}bits}}$

- Assembling the Final 64-bit Representation:

  We have the 3 fields in a 32-bits FP representation organized as follows:

  Sing bit (1-bit) | Biased Exponent (11-bits) | Mantissa (52-bits)

  Binary representation: $\mathbf{[1|100,0000,0001|0011,0000,\cdots\cdots,0000,0000]_{FP\text{-}64}}$

  In hexadecimal: $\mathbf{[C013,0000,0000,0000]_{FP\text{-}64}}$

5. Binary Simple Precision decoding to Decimal of the 32-bits: **$[00700000]_{FP32}$**

- Convert the Double Precision from Hexadecimal to Binary:

  $[00700000]_{FP-32} = [0|000,0000,0|111,0000,0000,0000,0000,0000]_{FP-32}$

- Extract Fields:
  S (Sign bit) = 0 → Positive number **(+)**
  E_{biased} = 00000000 → 0 = **$(0)_{10}$**
- M = 111,0000,0000,0000,0000,0000 → **M = 111** (zeros after the last 1 don't count)

It is important to notice **$E_b = 0$** and **$M \neq 0$**. This means the number is **Denormalized**.

  - $E_{min} = E_b$-Bias = 1-Bias

- Real minimum exponent:
  Bias = $(127)_{10}$ (constant value)
  E_{min} = 1 - Bias = 1 - 127 = -126 → **$E_{min} = -126$** (constant value too)

- Determine Significand:
  - Since the number is Denormalized, the significand is 0.M
  - Significand = **$(0.111)_2$**

- Calculating the final value using the Denormalized formula:
  The formula: $FP = (-1)^S \times 0.M \times 2^{min}$
  $FP = +0.111 \times 2^{-126} = (1.11)_2 \times 2^{-127}$
  Using FP: $(1.11)_2 = ((1x2^0) + (0x2^{-1}) + (1x2^{-2})) \times 2^{-127} = (1.75)_{10} \times 5.87747 \times 10^{-39}$
  $FP = 1.02855 \times 10^{-38}$

Remark 7: In this course, it is established to take 5 digits after the fractional dot for maximum precision.

6. Decimal encoding to Binary Double Precision 64-bits the decimal **$(2.938736 \times 10^{-39})_{10}$**:

- This number seems too small, it is preferable to check if it is smaller than 2^{Emin}:
  $E_{min} = -126$ (It is a constant)
  $2^{-126} = 1.17549 \times 10^{-38} > 2.938736 \times 10^{-39}$ → use of **Denormalized** scheme.

Remark 8: You can observe that it become very easy to compare fractional numbers represented in Scientific Notation directly by comparing their Exponents. This why IEEE-754 uses biased Exponent to represent negative numbers instead of MS, 1C, or 2C.

- Determining the Sign bit (S):

  The number $+2.938736 \times 10^{-39}$ is positive → **S=0**

- Getting the value of the Significand:

  We have: $FP = (-1)^S \times 0.M \times 2^{Emin} \rightarrow +2.938736 \times 10^{-39} = 0.M \times 2^{-126}$
  $\rightarrow 0.M = 2.938736 \times 10^{-39} / 2^{-126}$
  $\rightarrow 0.M = (0.25)_{10}$

- Converting the Number to Binary:
  Convert the fractional part $(0.25)_{10}$ to a binary fraction using SM:

Multiplication	Integer part
0.25 x 2 = 0.5	0
0.5 x 2 = 1.0	1

  $(0.25)_{10}$ = $(0.01)_2$

- Getting the value of M:
  $0.M = (0.25)_{10} = (0.01)_2 \rightarrow$ **M = 01**

- Assembling the Final 32-bit Denormalized Representation (knowing that $E_b = 0$):

  We have the 3 fields in a 32-bits FP representation organized as follows:
  Sing bit (1-bit) | Biased Exponent (8-bits) = 0 | Mantissa (23-bits)
  Binary representation: **[0|000,0000,0|010,0000,0000,0000,0000,0000]$_{FP-32}$**
  In hexadecimal: **[00200000]$_{FP-32}$**

Exercise 5:

1). Encoding the sentence "Salamou Alaykoum" into its 8-bit ASCII hexadecimal representation, we need to use the Extended ASCII table. We have to find the 8-bit ASCII hexadecimal value trough the table for each character:

"Salamou "							
S: 0x53	**a:** 0x61	**l:** 0x6C	**a:** 0x61	**m:** 0x6D	**o:** 0x6F	**u:** 0x75	**" ":** 0x20
"Alaykoum"							
A: 0x41	**l:** 0x6C	**a:** 0x61	**y:** 0x79	**k:** 0x6B	**o:** 0x6F	**u:** 0x75	**m:** 0x6D

Encoded Sentence in 8-bit ASCII Hexadecimal:

[53 61 6C 61 6D 6F 75 20 41 6C 61 79 6B 6F 75 6D]$_{ASCII}$

2). Decoding the following sequence to reveal the original sentence, is practically the opposite operation. Using the ASCII table to find the character behind each 2 hexadecimal digits (8-bits):

[59 65 61 72 20 32 30 32 35]$_{ASCII}$

59: **Y**	65: **e**	61: **a**	72: **r**	20: **" "**
32: **2**	30: **0**	32: **2**	35: **5**	

Decoded sentence: **"Year 2025"**

Remark 9: In this exercise we were forced to use Hexadecimal for a small sentences of ASCII characters, because using binary would be very tedious. For example the sentence "Salamou Alaykoum" would need 8-bits by 16 characters, 128-bits to write by hand. This is why Hexadecimal is preferable than binary for humans use.