Group name: DREAMERS
Group members: MAIZATUL AFRINA SAFIAH BINTI SAIFUL AZWAN
MADIHAH BINTI CHE ZABRI
ONG HAN WAH
NAYLI NABIHAH BINTI JASNI
MADINA SURAYA BINTI ZHARIN
AHMAD AIMAN HAFIZI BIN MUHAMMAD

WEEK 1

Activity 1

- Computer does not understand text, images and videos. It only understands the 0 and 1 format. It is called binary language. Do you agree?
  Agree because all data inside the computer is transmitted by electrical signals which is on and off. Thus, they need to be converted into binary which is a series of 0s and 1s forms for the computer to process it.
- Explain how computers understand text, images and videos?
  Computers take the form of binary digits to understand and be able to process text, images and videos. For example, we can use code like ASCII code which is enough for english words and Unicode for other languages that need more characters to convert the text into binary form. Next, binary digits which made up pixels for representing digital images can be used, 0 for white or 1 for black but we can add the bits if we want to produce more colours. In order to understand the video, the computers process pixels in a series of video frames.

Activity 2

• What, in general terms, is the distinction between computer organization and computer architecture?

- **Computer organization refers to the operational units and their interconnections that realize the architectural specification like how do the computers work. For example, method of signaling, types of memory and control signal. Meanwhile, computer architecture is mainly for design of computers, it usually refers to those features of the system that are visible to programmers and the logical execution of a program will directly impact such as input and output mechanisms, data types, sets and format of instructions and others.**

• Differentiate a von Neumann model vs non von Neumann model

- The Von Neumann model is a stored program computer. This model consists of 4 main sub-system which includes ALU, memory, input/output and control unit. In this model, the programs are stored in the memory while the program execution process is conducted sequentially. These features in the von Neumann model have been improved starting from the addition of processor and parallel processing which marks the starting of a non

von Neumann model. By adding the processor, it can improve the performance of the computer itself, and the addition of parallel processing will increase the speed by allowing multitasking during its execution. As the time passed by, the non von Neumann model also includes the microprocessor which is a chip that consists of many elements which can be used to save spaces.

Activity 3

• See slide 4, do you understand all the specs?

• Which machine uses dual core? What does dual/quad core mean? Do all machines have L3 cache?

- **The first machine uses dual core processors**
- **A dual core means the machine has CPU with two processors residing on a single chip**
- **all the machines in slide 4 have L3 cache**

• How fast are the processors of the machines?

Each processor in slide 4 has a clock speed of 1.4GHz, 2.7GHz, 2.9GHz, 3,2GHz and 3.4GHz respectively. When handling heavy workloads, the processors can boost their clock speed up to 2.7GHz, 3.2GHz, 3.6GHz, 3.6GHz and 3.8GHz respectively. The higher the clock speed, the faster the CPU processes its tasks.

• How big and fast (RPM) are the memory for the machines?

- ➔ When storing data, a 7,200 rpm hard drive is obviously faster than a 5,400 rpm hard drive.
- ➔ Small outline Dual Input memory Module (SO-DIMM) in machine 4 and 5 are just the half version of the Dual In Line Memory Module (DIMM)
- ➔ Machine 4 and 5 have a larger RAM which are to allocate more memory cache than other machines.
- ➔ Machine 4 and 5 are also capable of sustaining much larger data storage while having a higher performance in storing huge chunks of data in a very short time.
- ➔ Machine 1 is the smallest and slowest in terms of data storage

WEEK 2

- **Activity 1**

  Everybody ok? Can do all exercise?
  Yes Dr, we can.

2.1 Represent the following decimal numbers in both binary sign/magnitude and two's complement using 16-bits binary system:

a) +512 $_{10}$

+512 $_{10}$ = 00000010 00000000 $_{2}$

b) -29 $_{10}$

-29 $_{10}$ = -(00000000 00011101 $_{2}$)
= 11111111 11100010 $_{2}$ + 1
= 11111111 11100011 $_{2}$

2.2 Represent the following two's complement values in decimal:

a) 1101011_2

MSB -> 1 (negative value)

1101011_2 => 0010100_2 (flipped the value) + 1

=> 0010101_2

=> 21

=> -21(negative value)

b) 0101101_2

MSB -> 0 (positive value)

0101101_2 => +45

2.3 **Convert the following 8 bit two's complement value to 16 bits binary system:**

a) 11001100_2 => $11111111\ 11001100_2$

b) 00101110_2 => $00000000\ 00101110_2$

2.4 Assume numbers are represented in 8-bits two's complement representation. Show the calculation of the following:

a) 6 >> 0000 0110 $0000\ 0110_2$ 6

13 >> 0000 1101 $+\ 0000\ 1101_2$ + 13

--- ---

$0001\ 0011_2$ 19

b) -6 >> 1111 1010 $1111\ 1010_2$ (-6)

13 >> 0000 1101 $+\ 0000\ 1101_2$ + 13

--- ---

$0000\ 0111_2$ 7

c) 6 - 13

```
                                  11
6 >> 0000 0110          0000 0110        6
-13 >> 0000 1101    +   1111 0011   +(-13)
         1111 0010      ----------    --------
       +        1      1111 1001      (-7)   >> check:  0000 0110
        ----------                                    +         1
         1111 0011                                     ----------
                                                       1111 0111 (7)
```

d) - 6 - 13

```
                      1 <  111    1
-6 >> 0000 0110        1111 1010       (-6)
                 1     + 1111 0011    + (-13)
        1111 1001       ----------     -------
       +        1      1110 1101      (-19) >> check : 0001 0010
        ----------                                    +         1
        1111 1010                                     ----------
                                                       0001 0011 (19)

-13 >> 0000 1101
         1111 0010
       +        1
         ----------
         1111 0011
```

2.5 **Find the following differences using twos complement arithmetic:**

a) 111000_2 -110011_2 --- 000101_2 ---	b) $1100\ 1100_2$ $-0010\ 1110_2$ --- $1001\ 1110_2$ ---	c) $1111\ 0000\ 1111_2$ $-1100\ 1111\ 0011_2$ --- $0010\ 0001\ 1100_2$ ---

- **ACTIVITY 2**

2.1 Perform the subtraction in two's complement representation for $37_{10} - 17_{10}$ in 8 bit and 16 bit binary system.

8 bit binary system

$37_{10} \Rightarrow 00100101_2$

$17_{10} \Rightarrow 00010001_2 \Rightarrow 11101110_2$ (flipped) + 1 = 11101111_2

$$\begin{array}{rl} & 00100101_2 \\ + & 11101111_2 \\ \hline \textbf{discarded} \Rightarrow & \mathbf{1}\ 00010100_2 \\ \hline \end{array}$$

16 bit binary system

$37_{10} \Rightarrow 00000000\ 00100101_2$

$17_{10} \Rightarrow 00000000\ 00010001_2$

$$\begin{array}{rl} & 00000000\ 00100101_2 \\ - & 00000000\ 00010001_2 \\ \hline & 00000000\ 00010100_2 \\ \hline \end{array}$$

2.2 In 4-bit binary arithmetic, find the multiplication of 5_{10} with 4_{10} using the 1st version of highly optimized multiplication hardware.

Iteration	Step	Multiplier(MP)	Multiplicand(MC)	Product(P)
0	Initial value	0101	0000 0100	0000 0000
1	1a: P=P+MC	0101		0000 0100
	2: Shift MC left		0000 1000	
	3: Shift MP right	0010		
2	1: No operation	0010		0000 0100
	2: Shift MC left		0001 0000	

	3: Shift MP right	0001		
3	1a: P=P+MC	0001		0001 0100 (Answer=20_{10})
	2: Shift MC left		0010 0000	
	3: Shift MP right	0000		
4	1: No operation	0000		0001 0100
	2: Shift MC left		0100 0000	
	3: Shift MP right	0000		

- **ACTIVITY 3**

2.3 In 6-bit binary arithmetic, find the multiplication of 21 $_{10}$ with 14 $_{10}$ using the 1st version of highly optimized multiplication hardware.

Iteration	Step	Multiplier(MP)	Multiplicand(MC)	Product(P)
0	Initial value	001110	000000 010101	000000 000000
1	No operation	001110		000000 000000
	2: Shift MC left		000000 101010	
	3: Shift MP right	000111		
2	P=P+MC	000111		000000 101010
	2: Shift MC left		000001 010100	
	3: Shift MP right	000011		
3	P=P+MC	000011		000001 111110
	2: Shift MC left		000010 101000	
	3: Shift MP right	000001		
4	P=P+MC	000001		000100 100110
	2: Shift MC left		000101 010000	
	3: Shift MP right	000000		

5	No operation	000000		000100 100110
	2: Shift MC left		001010 100000	
	3: Shift MP right	000000		
6	No operation	000000		000100 100110 Answer= 294_{10}
	2: Shift MC left		010101 000000	
	3: Shift MP right	000000		

- **ACTIVITY 4**

2.4 In 6-bit binary arithmetic, find the multiplication of 21_{10} with (-14_{10}) by using:
a) the two's complement binary numbers. Proof that it yields incorrect results.

Iteration	Step	Multiplier(MP)	Multiplicand(MC)	Product(P)
0	Initial value	110010	0000 0001 0101	0000 0000 0000
1	No operation	110010		0000 0000 0000
	2: Shift MC left	110010	0000 0010 1010	
	3: Shift MP right	011001		
2	P = P + MC	011001		0000 0010 1010
	2: Shift MC left	011001	0000 0101 0100	
	3: Shift MP right	001100		
3	No Operation	001100		0000 0010 1010
	2: Shift MC left	001100	0000 1010 1000	
	3: Shift MP right	000110		
4	No Operation	000110		0000 0010 1010
	2: Shift MC left	000110	0001 0101 0000	

	3: Shift MP right	000011		

5	P = P + MC	000011		0001 0111 1010
	2: Shift MC left	000011	0010 1010 0000	
	3: Shift MP right	000001		
6	P = P + MC	000001		0100 0001 1010 (acquired answer = +1050) (supposed answer = -294)
	2: Shift MC left	000001	0101 0100 0000	
	3: Shift MP right	000000		

b) the 1st version of highly optimized multiplication hardware.

Iteration	Step	Multiplier(MP)	Multiplicand(MC)	Product(P)
0	Initial value	001110	1111 1110 1011	0000 0000 0000
1	No operation			
	2: Shift MC left		1111 1101 0110	
	3: Shift MP right	000111		
2	1a: P = P + MC	000111		1111 1101 0110
	2: Shift MC left		1111 1010 1100	
	3: Shift MP right	000011		
3	1a: P = P + MC	000011		1 1111 1000 0010
	2: Shift MC left		1111 0101 1000	

	3: Shift MP right	000001		
4	1a: P = P + MC	000001		1 1110 1101 1010 1110 1101 1010(2s) Flip => 0001 0010 0101 + 1 => 0001 0010 0110 => -294
	2: Shift MC left		1110 1011 0000	
	3: Shift MP right	000000		
5	No operation	000000		
	2: Shift MC left		1101 0110 0000	
	3: Shift MP right	000000		
6	No operation	000000		
	2: Shift MC left		1010 1100 0000	
	3: Shift MP right	000000		

- **ACTIVITY 5**

Using a 4-bit binary arithmetic, find the division of the following numbers with the 1st version of highly optimized division hardware.

a) 6_{10} by 3_{10}

6_{10} / 3_{10} = ----------

6_{10} >> 0110_2
3_{10} >> 0011_2

Iteration	Steps	Quotient (Q)	Divisor (D)	Remainder (R)
0	Initial value	0000	0011 0000	0000 0110
1	1: R = R - D			1101 0110
	2b: R < 0; R = R +D Q: Shift Left (+0)	0000		0000 0110
	3: D = Shift right		0001 1000	
2	1: R = R - D			1110 1110
	2b: R < 0; R = R +D Q: Shift Left (+0)	0000		0000 0110
	3: D = Shift right		0000 1100	
3	1: R = R - D			1111 1010
	2b: R < 0; R = R +D Q: Shift Left (+0)	0000		0000 0110
	3: D = Shift right		0000 0110	
4	1: R = R - D			**0000 0000**
	2a: No operation Q: Shift Left (+1)	0001		
	3: D = Shift right		0000 0011	
5	1: R = R - D			1111 1101
	2b: R < 0; R = R +D Q: Shift Left (+0)	**0010**		0000 0000
	3: D = Shift right		0000 0001	

Answer : 6/ 3 = 2 Remainder 0

b) 6_{10} by (-3_{10})

$6_{10} / 3_{10}$ = ---------- (dividend and divisor positive and perform division)

$6_{10} >> 0110_2$
$3_{10} >> 0011_2$

Iteration	Steps	Quotient (Q)	Divisor (D)	Remainder (R)
0	Initial value	0000	0011 0000	0000 0110
1	1: R = R - D			1101 0110
	2b: R < 0; R = R +D Q: Shift Left (+0)	0000		0000 0110
	3: D = Shift right		0001 1000	
2	1: R = R - D			1110 1110
	2b: R < 0; R = R +D Q: Shift Left (+0)	0000		0000 0110
	3: D = Shift right		0000 1100	
3	1: R = R - D			1111 1010
	2b: R < 0; R = R +D Q: Shift Left (+0)	0000		0000 0110
	3: D = Shift right		0000 0110	
4	1: R = R - D			**0000 0000**
	2a: No operation Q: Shift Left (+1)	0001		
	3: D = Shift right		0000 0011	
5	1: R = R - D			1111 1101
	2b: R < 0; R = R +D Q: Shift Left (+0)	**0010**		0000 0000
	3: D = Shift right		0000 0001	

Answer : 6 / 3 = 2 Remainder 0

Based on rules: +6 / - 3 : Quotient = -2, Remainder = +0
Final answer: 6 / (-3) = -2 Remainder 0

c) (-12_{10}) by 5_{10}

$12_{10} >> 1100_2$

$5_{10} >> 0101_2$

Iteration	Steps	Quotient (Q)	Divisor (D)	Remainder (R)
0	Initial value	0000	0101 0000	0000 1100
1	1: R = R - D			1011 1100
	2b: R < 0; R = R +D Q: Shift Left (+0)	0000		0000 1100
	3: D = Shift right		0010 1000	
2	1: R = R - D			1110 0100
	2b: R < 0; R = R +D Q: Shift Left (+0)	0000		0000 1100
	3: D = Shift right		0001 0100	
3	1: R = R - D			1111 1000
	2b: R < 0; R = R +D Q: Shift Left (+0)	0000		0000 1100
	3: D = Shift right		0000 1010	
4	1: R = R - D			0000 0010
	2a: No operation Q: Shift Left (+1)	0001		
	3: D = Shift right		0000 0101	
5	1: R = R - D			1111 1101
	2b: R < 0; R = R +D Q: Shift Left (+0)	0010		0000 0010
	3: D = Shift right		0000 0010	

Taking the absolute value of 12 and 5 to perform division, the result is:
+12 / +5 = +2 remainder +2
For -12 / +5, remainder = -2, quotient = -2
Final answer: **-12 / 5 = -2 remainder -2**

WEEK 3

- **ACTIVITY 6**

Based on the following format, write the decimal number 65536 in the floating-point representation and store it in the floating-point format. Show your working.

65536 = 1.0×2^{16}

= 0.1×2^{17}

$17_{10}=10001_2$

0	1 0 0 0 1	1 0 0 0 0 0 0 0
Sign	exponent	Fraction

- **ACTIVITY 7**

Transform (-33.625_{10}) to floating point using the following format (radix 2). Show your working.

33/2 = 16 r = 1 0.625*2 = 1.25 >>1

16/2 = 8 r = 0 0.25*2 = 0.5 >>0

8/2 = 4 r = 0 0.5*2 = 1.0 >> 1

4/2 = 2 r = 0 0.0*2 = 0.0 >> 0

2/2 = 1 r = 0

$33_{10} = 100001_2$ **$33.625_{10} = 100001.1010_2$**

$-33.625_{10} = -100001.1010_2 \times 2^0$
$= -0.1000011010_2 \times 2^6$
Bias value, $b = 2^{5-1} = 16$
Biased exponent, $E_b = 16 + 6 = 22_{10} = 10110_2$

1	1 0 1 1 0	1 0 0 0 0 1 1 0
Sign	E_b	Fraction

Transform (-0.03125_{10}) to floating point using the following format (radix 2). Show your working.

0.03125*2=0.0625
0.0625*2=0.0125
0.0125*2=0.25
0.25*2=0.5
0.5*2=1.0

$-0.03125_{10} = -0.00001_2 \times 2^0$
$= -0.1_2 \times 2^{-4}$
Biased value, $b = 2^{5-1} = 16$.
Biased exponent, $E_b = 16+(-4) = 12_{10} = 1100_2$

1	0 1 1 0 0	1 0 0 0 0 0 0 0
Sign	E_b	Fraction

Complete the table with the normalized binary number and its exponent respectively using single precision floating point.

	Binary Values	Normalized as	Exponent (e')
(a)	1101.101	**1.101101 x** 2^3	**3**
(b)	0.00101	**1.01 x** 2^{-3}	**-3**
(c)	1.0001	**1.0001 x** 2^0	**0**
(d)	10000011.0	**1.00000110 x** 2^7	**7**

- **ACTIVITY 8**

Complete the table with the biased exponent (E_B) and binary representation for each number using the type of floating-point respectively.

	Exponent (e')	Biased exponent (E_b)			
		Single precision		Double precision	
		dec	bin	dec	bin
a)	3	130	10000010	1026	10000000010
b)	-3	124	01111100	1020	01111111100
c)	0	127	01111111	1023	01111111111
d)	7	134	10000110	1030	10000000110

- **ACTIVITY 9**

Convert the following number to single precision floating-point.

(a) (– 33.625)

Step 1 Normalized the bit value

$33_{10} \Rightarrow 100001_2$ $33.625_{10} = 100001.101_2 \times 2^0$

0.625 x 2 = 1.25 $= 1.00001101_2 \times 2^{0+5}$

0.25 x 2 = 0.5 $= 1.00001101_2 \times 2^5$

0.5 x 2 = 1.0

Step 2 Determine sign bit (S)

S ⇒ 1

Step 3 Determine biased exponent, E_b

b = $2^{n-1} = 2^{8-1} - 1 = 127$

$E_b = e' + b = 5 + 127 = 132_{10} = 10000100_2$

Step 4 Determine fraction

1) Drop leading 1 in the fraction

$1.00001101_2 \times 2^{10000100} \Rightarrow 00001101$

2) Expand padding to 23 bits

0000 1101 0000 0000 0000 000 $= 1.101_2 \times 2^5$

3) Complete representation

1	1000 0100	0000 1101 0000 0000 0000 000

(b) 0.03125_{10}

Step 1 Normalized the bit value

0.03125 x 2 = 0.0625 >> 0 $0.03125_{10} = 0.0001_2$

0.0625 x 2 = 0.125 >> 0 $0.0001_2 = 1.0 \times 2^{-4}$

0.25 x 2 = 0.5 >> 0

0.5 x 2 = 1.0 >> 1

Step 2 Determine sign bit (S)

S ⇒ 0

Step 3 Determine biased exponent, E_b

b = $2^{n-1} = 2^{8-1} - 1 = 127$

$E_b = e' + b = -4 + 127 = 123_{10} = 0111\ 1011_2$

Step 4 Determine fraction

4) Drop leading 1 in the fraction

$1.0_2 \times 2^{0111\ 1011}$ ⇒ 00000000000000000000000

5) Expand padding to 23 bits

0000 0000 0000 0000 0000 000 $= 1.0_2 \times 2^{-4}$

6) Complete representation

0	0111 1011	0000 0000 0000 0000 0000 000

- **ACTIVITY 10**

Complete the table with all sign (S) S), exponent (e') and fraction (F) values if single precision floating point applied.

	Binary Values	(decimal) E_B	*S*	(binary) E_B	*Fraction*
(a)	-1.11	**0**	**1**	**0000 0000**	**1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**
(b)	+1101.101	**130**	**0**	**1000 0010**	**1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

(c)	-0.00101	**124**	**1**	**0111 1100**	**0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**
(d)	+100111.0	**132**	**0**	**1000 0100**	**0 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 0 0 0 0 0 0**
(e)	+0.0000001101011	**120**	**0**	**0111 1000**	**1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

- **ACTIVITY 11**

Convert the following number to double precision floating-point.

(a) (– 0.75_{10})

0.75*2=1.5

0.5*2=1.0

$0.75_{10} = 0.11_2 \times 2^0$

$= 1.1_2 \times 2^{-1}$

Sign bit, S = 1

$E_b = -1+1023 = 1022_{10} = 1111111110_2$

1bit	11 bits	52 bits
1	0 1 1 1 1 1 1 1 1 1 0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0...0
Sign	E_b	Fraction

(b) 10.4_{10}

$10_{10} = 1010_2$

0.4*2=0.8

0.8*2=1.6

0.6*2=1.2

0.2*2=0.4

$10.4_{10} = 1010.0110_2 \times 2^0$

$= 1.0100110_2 \times 2^3$

Sign bit, S = 0

$E_b = 3+1023 = 1026_{10} = 10000000010_2$

Fraction = 0100110

1bit	11 bits	52 bits
0	1 0 0 0 0 0 0 0 0 1 0	0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0...0
Sign	E_b	Fraction

- **ACTIVITY 12**

Add these two binary floating-point numbers. 0.6015625 + 0.78125 = 1.0110_2

0.6015625 x 2 = 1.203125
0.203125 x 2 = 0.40625
0.40625 x 2 = 0.8125
0.8125 x 2 = 1.625
0.625 x 2 = 1.25
0.25 x 2 = 0.5
0.5 x 2 = 1.0

$(0.1001101 \times 2^{0}) \times 2^{-1} = 1.001101 \times 2^{-1}$

0.78125 x 2 = 1.5625
0.5625 x 2 = 1.125
0.125x 2 = 0.25
0.25 x 2 = 0.5
0.5 x 2 = 1.0

$(0.11001 \times 2^{0}) \times 2^{-1} = 1.1001 \times 2^{-1}$

$(1.001101 \times 2^{-1}) + (1.1001 \times 2^{-1})$

Step 1: Both have same exponent

Step 2: Add the fraction

$$
\begin{array}{rl}
 & 1.001101 \times 2^{-1} \\
+ & 1.1001 \quad \times 2^{-1} \\
 & \text{----------} \\
 & 10.110001 \times 2^{-1}
\end{array}
$$

Step 3: Normalize the sum

$10.110001 \times 2^{-1} \Rightarrow 1.0110001 \times 2^{0}$
Step 4: : Round the fraction (to 4 decimal digits for fraction).

1.0110×2^{0}

Step 5: Normalize
Since it had been normalized, then no need.

Answer = 1.0110 x2 ^{0}

Sum in decimal is : 1.0110 x2 ^{0} = $1.0110_2 = 1.011_2$

$= (1 \times 2^{0}) + (1 \times 2^{-2}) + (1 \times 2^{-3})$

$= 1.375_{10}$

- **ACTIVITY 13**

Given two numbers 0.5 and –0.4375.

(a) Multiply the numbers.

0.5 x 2 = 1.0
0.1 x 2 ^{0} = 1.0 x 2 $^{-1}$

0.4375 x 2 = 0.875
0.875 x 2 = 1.75
0.75 x 2 = 1.5
0.5 x 2 = 1.0
(0.0111 x 2 ^{0}) x 2 $^{-2}$ = 1.11 x 2 $^{-2}$

Step 1: Add the exponent of the 2 numbers.

(-1) + (-2) = -3

If bias considered (–3) + 127 = 124

Step 2: Multiply the fraction.

```
      1.1 1
x       1.0
----------------
      0 0 0
    1 1 1
```

1.1 1 0

Step 3: Normalize the product.

$1.110 \Rightarrow 1.110 \times 10^{-3}$

Step 4: Round the fraction.

1.1100×10^{-3}

Step 5: Normalize
Since it had been normalized, then no need

.

Step 6: Set the sign of the product.

-1.1100×10^{-3}

(b) Converting to decimal to check the results. Show your workings.

$-1.1100 \times 10^{-3} = -0.0011100 = -0.00111$

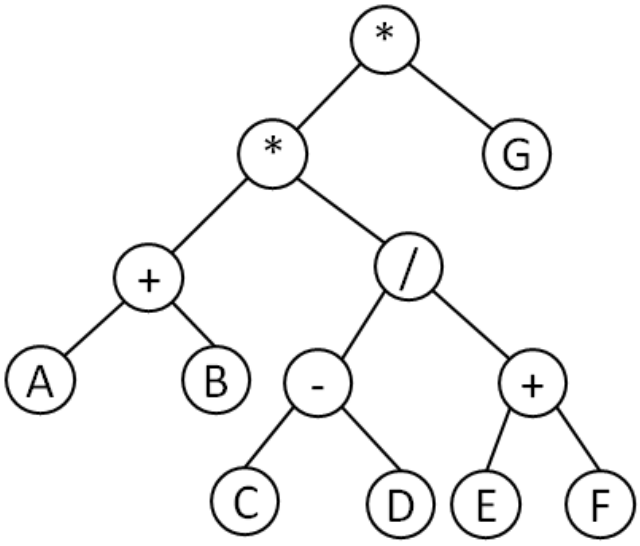
$= -(1 \times 2^{-3}) + (1 \times 2^{-4}) + (1 \times 2^{-5})$

$= -\ 0.21875_{10}$

WEEK 5

ACTIVITY 1

Given an expression as
(A+B) * ((C-D)/(E+F))*G
(a) Construct the expression tree.



(b) Convert into PRN postfix evaluation.
AB+CD-EF+/*G*

ACTIVITY 2

Get the infix expression for the following postfix:.

(a) AB + C -

Input	Output/Infix	Stack	Reason
AB + C -	empty	empty	A is operand, push A
B + C -	empty	A	B is operand, push B
+ C -	empty	A B	+ is operator, pop two operand, form infix operation
C -	A + B	empty	Push infix into stack
C -	empty	(A + B)	C is operand, puch C
-	empty	(A + B) C	- is operator, pop two operand, form infix expression
empty	(A + B) - C	empty	Push infix into stack
empty	empty	((A + B) - C)	Input empty, pop all
	((A + B) - C)		

(b) AB + CD -*

Input	Output/ Infix	Stack	Reason
AB + CD -*	empty	empty	A is an operand, push A
B + CD -*	empty	A	B is an operand, push B
+CD -*	empty	AB	+ is an operator, pop two operand, form infix expression
CD -*	A+B	empty	Push infix to stack
CD -*	empty	(A+B)	C is an operand, push C
D -*	empty	(A+B) C	D is an operand, push D
-*	empty	(A+B) CD	- is an operator, pop two operand, form infix expression
*	(A+B) (C-D)	empty	Push infix to stack
*	empty	(A+B) (C-D)	* is an operator, pop two operand, form infix expression
empty	(A+B) * (C-D)	empty	Push infix to stack
empty	empty	((A+B) * (C-D))	Input empty, pop all
	((A+B) * (C-D))		

(c) AB^C * D - EF / GH +/+

Input	output/ infix	Stack	Reason
AB^C * D - EF / GH +/+	empty	empty	A is operand, push A
B^C * D - EF / GH +/+	empty	A	B is operand, push B
^C * D - EF / GH +/+	empty	A B	^ is operator, pop two operand, form infix

			expression
C * D - EF / GH +/+	A ^ B	empty	Push infix into stack
C * D - EF / GH +/+	empty	(A^B)	C is operand, push C
* D - EF / GH +/+	empty	(A^B) C	* is operator, pop two operand, form infix expression
D - EF / GH +/+	(A ^ B) * C	empty	Push infix into stack
D - EF / GH +/+	empty	((A^B) * C)	D is operand, push D
- EF / GH +/+	empty	((A^B) * C) D	- is operator,pop two operand, form infix expression
EF / GH +/+	((A^B) * C) - D	empty	Push infix into stack
EF / GH +/+	empty	(((A^B) * C) - D)	E is operand, push E
F / GH +/+	empty	(((A^B) * C) - D) E	F is operand, push F
/ GH +/+	empty	(((A^B) * C) - D) E F	/ is operator,pop two operand, form infix expression
GH +/+	(((A^B) * C) - D) (E/ F)	empty	Push infix to stack
GH +/+	empty	(((A^B) * C) - D) (E/ F)	G is operand, push G
H +/+	empty	(((A^B) * C) - D) (E/ F) G	H is operand, push H
+/+	empty	(((A^B) * C) - D) (E/ F) G H	+ is operator,pop two operand, form infix expression
/ +	(((A^B) * C) - D) (E/ F) (G + H)	empty	Push infix to stack
/ +	empty	(((A^B) * C) - D) (E/ F) (G + H)	/ is operator,pop two operand, form infix expression
+	(((A^B) * C) - D) (E/ F) / (G + H)	empty	Push infix to stack

+	empty	(((A^B) * C) - D) ((E/ F) / (G + H))	+ is operator,pop two operand, form infix expression
empty	(((A^B) * C) - D) + ((E/ F) / (G + H))	empty	Push infix to stack
empty	empty	((((A^B) * C) - D) + ((E/ F) / (G + H)))	Input empty, pop all
	((((A^B) * C) - D) + ((E/ F) / (G + H)))		

(d) AB + C * DE - - FG +^

Input	Output / infix	Stack	Reason
AB + C * DE - - FG +^	empty	empty	A is operand, push A
B + C * DE - - FG +^	empty	A	B is operand, push B
+C*DE- - FG +^	empty	A B	+ is operator, pop two operand, form infix expression
C*DE- - FG +^	A + B	empty	Push infix into stack
C*DE- - FG +^	empty	(A + B)	C is operand, push C
*DE - - FG +^	empty	(A + B) C	*is operator, pop two operand, form infix expression
DE - - FG +^	(A + B) * C	empty	Push infix into stack
DE - - FG +^	empty	((A + B) * C)	D is operand, push D
E - - FG +^	empty	((A + B) * C)D	E is operand, push E
- - FG +^	empty	((A + B) * C)DE	- is operator, pop two operand, form infix expression
- FG +^	((A + B) * C) (D-E)	empty	Push infix into stack
- FG +^	empty	((A + B) * C) (D-E)	- is operator, pop two operand, form infix

			expression
FG + ^	((A + B) * C) - (D-E)	empty	Push infix into stack
FG + ^	empty	(((A + B) * C) - (D-E))	F is operand, push F
G +^	empty	(((A + B) * C) - (D-E)) F	G is operand, push G
+^	empty	(((A + B) * C) - (D-E)) FG	+ is operator, pop two operand, form infix expression
^	(((A + B) * C) - (D-E)) F + G	empty	Push infix into stack
^	empty	(((A + B) * C) - (D-E)) F + G	^ is operator, pop two operand, form infix expression
empty	(((A + B) * C) - (D-E)) ^ F + G	empty	Push infix into stack
empty	empty	(((A + B) * C) - (D-E)) ^ (F + G)	Input empty, pop all
	(((A + B) * C) - (D-E)) ^ (F + G)		

(e) ABCDE ^*/-

Input	output/ infix	Stack	Reason
ABCDE^*/-	empty	empty	A is operand, push A
BCDE^*/-	empty	A	B is operand, push B
CDE^*/-	empty	A B	C is operand, push C
DE^*/-	empty	A B C	D is operand, push D
E^*/-	empty	A B C D	E is operand, push E
^*/-	empty	A B C D E	^ is operator, pop two operand, form infix expression
*/-	D^E	A B C	Push infix into stack

*/-	empty	A B C (D^E)	* is operator, pop two operand, form infix expression
/-	C*(D^E)	A B	Push infix into stack
/-	empty	A B (C*(D^E))	/ is operator, pop two operand, form infix expression
-	B/C*(D^E)	A	Push infix into stack
-	empty	A (B/(C*(D^E)))	- is operator, pop two operand, form infix expression
empty	A-(B/(C*(D^E)))	empty	Push infix into stack
empty	empty	(A-(B/(C*(D^E))))	Input empty, pop all
	(A-(B/(C*(D^E))))		

ACTIVITY 1

Given three arrays with multiple initializer in assembly language program. Complete the following diagram by writing:
(a) the offset for all memory. Assume the first offset is 00404004.
(b) the value of the data stored using Little Endian Order.

Offset:	Value:
00404004:	10
00404005:	11
00404006:	12
00404007:	13
00404008:	23
00404009:	01
00404010:	34
00404011:	02

Offset:	Value:
00404014:	56
00404015:	04
00404016:	56
00404017:	34
00404018:	21
00404019:	00
00404020:	89
00404021:	67

00404012:	45	00404022:	45
00404013:	03	00404023:	23

ACTIVITY 2

Given that A = 29, B = 4 and C = 31, complete the following table with value of each operand given the addressing mode

Operand	Addressing Mode	Value
C	Direct	7
R4	Register	16
R2	Register Indirect	18
34	Immediate	34
A	Indirect	5
R1, B	Displacement	19