

HOMEWORK II

DUE : February 22, 2012

READ :

- Related sections of Chapter 2
- Related sections of Chapter 3
- Related sections of Appendix B

ASSIGNMENT : There are eight questions seven of which are from chapters II and III of the textbook.

Solve all homework and exam problems as shown in class and past exam solutions.

I) Solve Problem 2.14.1 of Chapter II for only part (a).

First, write a mnemonic machine language program with comments. **Then**, show the table of execution of instructions with used register and memory location values until the code completes. **Finally**, show the individual bits to describe the shift to the right/left operation(s) to get new values. Note the following :

- The piece of mnemonic machine language program you will write starts at 400000.
- According to MIPS software conventions registers \$t0 and \$t1 are R8 and R9, respectively.
- Assume that register R8 has 45678F2A as the initial value.
- Change “31 - i bits” in the figure to “32 - i bits” since “31 - i bits” is incorrect

II) Solve Problem 2.16.1 of Chapter II for only part (b).

To solve the problem do the following : **First**, convert the assembly program to a mnemonic machine language program with comments. **Then**, show the table of execution of instructions with used register and memory location values until the code completes. Note the following :

- The piece of mnemonic machine language program you will write starts at 400000.
- According to MIPS software conventions, registers \$0, \$t0, \$t1 and \$t2 are R0, R8, R9 and R10, respectively.

III) Solve Problem 2.17.3 of Chapter II for only part (a).

You will do your programming in the mnemonic machine language with comments. In addition to implementing the instruction, also show the **table** that has the values of registers and memory locations “touched” and memory accesses made by your sequence of instructions.

- ➔ The piece of mnemonic machine language program to implement “SUBI” starts at 400200.
- ➔ Assume that register R11 has $(-3)_{10}$ as the initial value.

IV) Solve Problem 2.30.1 of Chapter II for only part (b).

The pseudoinstruction implements the following : If (\$t1 == large) then go to LOOP

Consider also, the following list of pseudoinstructions :

- ➔ BEQ \$t2, small, LOOP => If (\$t2 == small) then go to LOOP
- ➔ LI \$t1, small => \$t1 = small
- ➔ LI \$t1, large => \$t1 = large

Implement the above four (4) **pseudoinstructions** in terms of (by using) actual EMY instructions. For each pseudoinstruction, write a mnemonic machine language program with comments that implements the architectural operations required by the pseudo instruction. Note the following :

- ➔ Assume that each mnemonic machine language program you write starts at memory location 400400 and looks like as follows :

```
400400 mnemonic rd/rt/rs/Imm/... # Comments are helpful
400404 mnemonic rd/rt/rs/Imm/... # Place the second instruction if necessary
400408 ---      ---             # Place the third instruction if necessary
```

- È Assume that number “large” is C6A4F80E. Number “small” is A2. Label “LOOP” corresponds to memory location 400300.

V) Consider the following piece of mnemonic machine language code :

```
400000  ADD  R8, R0, R0
400004  BEQ  R5, R0, 3
400008  ADD  R8, R8, R4
40000C  ADDI R5, R5, (-1)10
400010  J    100001
400014  ADDI R8, R8, (100)10
400018  ADD  R2, R8, R0
```

Determine what this piece of code implements.

To solve the problem do the following :

First, add comments to the above mnemonic machine language program.

Then, show the table of execution of instructions with used register and memory location values until the code completes.

Finally, describe in one sentence what it does, i.e. the purpose of the code. Note the following :

→ Assume that initially register R4 has value 4 and represents variable “a” and register R5 have value 2 and represents variable “b.”

VI) Solve Problem 3.10.2 of Chapter III for only part (a).

The Instruction Register is an organizational register not visible to the machine language programmer. That is why we did not discuss it. The register keeps the current instruction executed. Thus, the question is asking you to interpret the bit sequence as an instruction.

Show the conversion steps, the work, as done in class and past exam solutions.

VII) Solve Problem 3.10.3 of Chapter III for only part (a).

Show the conversion steps, the work, as done in class and past exam solutions so that it is clear that a calculator is not used.

L/S architecture anymore, since ADDMRM accesses the **memory** for data. ADDMRM stands for **Add Memory and Register to Memory**.

We perform three memory accesses for the new instruction. One to fetch the instruction, one to read a data element and one to write a data element.

Q2) Consider the following pseudoinstruction :

ADDI \$t0, \$t1, big

The instruction adds \$t1 and a 32-bit number named “big” and stores the result in \$t0.

Implement the pseudoinstruction by writing a piece of EMY **mnemonic** machine language code that uses instructions covered **in class**.

Assume that your piece of code starts at 400400. Assume also that number “big” has the value 56789ABC. Use software conventions discussed in class. Add comments to your code.

If the code contains more than five (5) instructions, it will not earn points.

A2) The code is as follows :

```
400400    LUI    R10, 5678            # Leftmost 16 bits of R10 are initialized to “5678”
400404    ORI    R10, R10, 9ABC       # Rightmost 16 bits of R10 are initialized to “9ABC”
400408    ADD    R8, R10, R9         # R8 gets R9 + 56789ABC
```

Q3) The EMY memory has the following information in these two locations :

400420 40400000
400424 C5040420

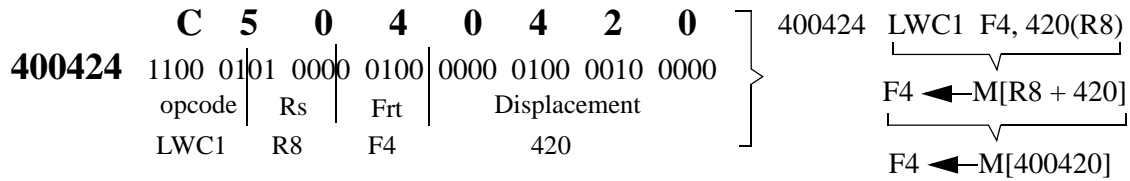
i) Can you identify what is stored in these two locations ? That is, can you tell if there are data or instructions there ? If yes, precisely describe them.

ii) If you are told that at the moment the memory has these bit patterns, the Program Counter (PC) is 400424 and the computer is about to start an instruction execution, how would you answer the questions in part (i) above ? **Assume** that **R8** has 400000 already.

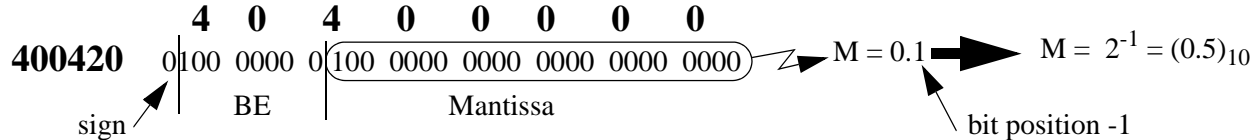
If you think there is (are) instruction(s) there, show the value of the destination register of the last instruction, after the run is complete.

A3) i) No ! We cannot tell if these locations contain instruction or data, since instructions and data can be stored anywhere in the memory. This is a fundamental property of stored-program computers. However, we have a software convention that instructions are stored starting at 400000, then we might have instructions in those two locations. Thus, we need to know if the software convention is strictly followed.

ii) Since PC now has 400424 and the computer is about to start an instruction execution, memory location 400424 has an instruction :



Since the LWC1 reads location 400420 to store in F4, location 400420 has a FP number, a data element :



The sign bit is 0, thus the FP number is positive.

The biased exponent is 10000000. Only bit seven (the leftmost bit) is 1. Then, $BE = 2^7 = 128$. Since BE is 128, the FP number is normalized. The real exponent is :

$$e = BE - 127 = 128 - 127 = 1$$

Finally, the FP number in binary power base : $(+1.1 * 2^1)$ and the FP number in decimal : $(+1.5 * 2^1) = 3$.

Note that, this code is **NOT** a good code because of two reasons : (i) The location before the LWC1 instruction has a FP number. In order to execute the LWC1 instruction, we must branch or jump to it to skip over the FP number. (ii) the MIPS software convention that static data be stored starting at 10000000 is not followed ! The reason why we have a separate space for static data starting at 10000000 is partly to avoid this kind of situations. Mixing instructions and data like this is risky and confusing !

Q4) The EMY machine language instruction set does not have the following instruction which is shown in the mnemonic notation :

$$400000 \quad \text{DECM} \quad 2C(R8) \quad \# M[R8 + 2C^+] \leftarrow ((M[R8 + 2C^+] - 1))$$

i) Assume that “2C” is a 16-bit signed displacement. **Implement** the instruction by using a few EMY instructions in the mnemonic notation. Add comments to your code. Your piece of code starts at 400000.

ii) Assume that this instruction is added to the EMY instruction set. **Describe** its syntax, semantics, format, etc. What does “DECM” stand for ?

A4) The given instruction is the following :

$$400000 \quad \text{DECM} \quad 2C(R8) \quad \# M[R8 + 2C^+] \leftarrow M[R8 + 2C^+] - 1$$

i) We see that we decrement a memory location by 1. We can implement it by using three instructions :

```

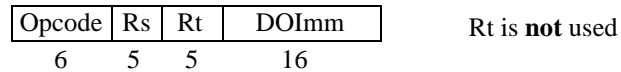
400000    LW      R9, 2C(R8)    # Read the memory location
400004    ADDI   R9, R9, (-1)10 # Decrement by 1
400008    SW      R9, 2C(R8)    # Store the new value back

```

ii) The **syntax** of the new instruction : `DECM Disp(Rs)`

The **semantics** of the new instruction : $M[Rs + Disp^+] \leftarrow M[Rs + Disp^+] - 1$

The format is the **I format** :



Three arguments are used by the instruction : The destination and the first source arguments are memory arguments. We use the **2-byte signed displacement** addressing mode for them. The other source argument is always $(-1)_{10}$. We use the **Implied** addressing mode for it. `DECM` stands for Decrement Memory.

We perform **three** memory accesses for the new instruction. One to fetch the instruction, one to read a data element and one to write a data element.

Q5) Consider the following EMY mnemonic machine language piece of program :

```

405000    SLT    R8, R4, R0
405004    BNE   R8, R0, 5
405008    SRL   R4, R4, 1
40500C    ADDI  R5, R5, (-1)10
405010    BNE   R5, R0, (-3)10
405014    ADD   R2, R4, R0
405018    JR    R31
40501C    LW    R9, 0(R10)
405020    SRL   R4, R4, 1
405024    OR    R4, R9, R4
405028    ADDI  R5, R5, (-1)10
40502C    BNE   R5, R0, (-4)10
405030    ADD   R2, R4, R0
405034    JR    R31
-----
10000000 80000000

```

a) The initial values of the registers and data memory locations used are shown below :

PC	Instruction	R2	R4	R5	R8	R9	R10	R31	M[10000000]	Memory Accesses
405000	?	?	8400	2	?	?	10000000	402548	80000000	-
.....	Continue

Continue the table until you cannot anymore...

b) Explain what this piece of program does in a few sentences.

A5) a)

PC	Instruction	R2	R4	R5	R8	R9	R10	R31	M[10000000]	Memory Accesses
405000	?	?	8400	2	?	?	10000000	402548	80000000	-
405004	SLT R8, R4, R0	NS	NS	NS	0	NS	NS	NS	NS	Inst. Read
405008	BNE R8, R0, 5	NS	NS	NS	NS	NS	NS	NS	NS	Inst. Read
40500C	SRL R4, R4, 1	NS	4200	NS	NS	NS	NS	NS	NS	Inst. Read
405010	ADDI R5, R5, (-1) ₁₀	NS	NS	1	NS	NS	NS	NS	NS	Inst. Read
405008	BNE R5, R0, (-3) ₁₀	NS	NS	NS	NS	NS	NS	NS	NS	Inst. Read
40500C	SRL R4, R4, 1	NS	2100	NS	NS	NS	NS	NS	NS	Inst. Read
405010	ADDI R5, R5, (-1) ₁₀	NS	NS	0	NS	NS	NS	NS	NS	Inst. Read
405014	BNE R5, R0, (-3) ₁₀	NS	NS	NS	NS	NS	NS	NS	NS	Inst. Read
405018	ADD R2, R4, R0	2100	NS	NS	NS	NS	NS	NS	NS	Inst. Read
402548	JR R31	NS	NS	NS	NS	NS	NS	NS	NS	Inst. Read

b) This piece of program, which is a subroutine, implements a complex instruction : **Arithmetic Shift Right (ASR)**. The syntax of it is :

ASR Rs, k

where “k” is the number of times we want to shift register Rs to the right. Obviously, “k” should satisfy : $0 < k < 32$.

The subroutine is passed the number to be shifted (the value of Rs) in R4. Number k is passed to the subroutine in R5. The result of the arithmetic shift is returned in R2.

Instruction in locations 405008 through 405014 rotate the value if it is positive. Instructions in locations 40501C through 405030 rotate the value if it is negative.

Q6) The EMY machine language instruction set does **not** have the following instruction which is shown in the mnemonic notation :

400000 SLTM R8, (R9), (R10) # If M[R9] < M[R10] then R8 ← 1 else R8 ← 0

i) Implement the instruction

SLTM R8, (R9), (R10)

by using a few EMY instructions in the mnemonic notation. Your piece of code starts at 400000.

Use software conventions discussed in class. Add comments to your code.

ii) Assume that this instruction is added to the EMY instruction set. **Describe** its syntax, semantics, format, etc. If there is a **new** addressing mode that is **not** discussed in class, indicate so. What does “SLTM” stand for ?

A6) The given instruction is the following :

400000 SLTM R8, (R9), (R10) # If M[R9] < M[R10] then R8 ← 1 else R8 ← 0

i) We see that we compare two memory locations and store the result in a register. We can implement it by using three instructions :

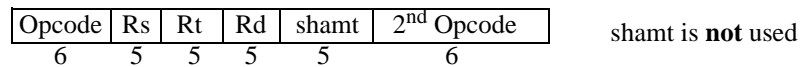
```

400000    LW    R11, 0(R9)    # Read the memory location pointed by R9
400004    LW    R12, 0(R10)   # Read the memory location pointed by R10
400008    SLT   R8, R11, R12  # Compare them and store the result in R8
  
```

ii) The **syntax** of the new instruction : **SLTM Rd, (Rs), (Rt)**

The **semantics** of the new instruction : If $M[Rs] < M[Rt]$ then $Rd \leftarrow 1$ else $Rd \leftarrow 0$

The format is the **R format** :



Five arguments are used by the instruction :

There are two **memory** arguments. We use the **Register Indirect** addressing mode for them. This addressing mode is **not** covered in class. A destination argument is a **register** for which we use the **register** addressing mode. Finally, two source arguments are constants : **1** and **0**. We use the **Implied** addressing mode for them. EMY is **not** a L/S architecture anymore, since SLTM accesses the **memory** for data.

We perform **three** memory accesses for the new instruction. One to fetch the instruction and two to read two data elements. SLTM stands for **Set on Less Than Memory**.

Q7) Consider the following piece of EMY **mnemonic** machine language program :

```

400F00    LW    R12, 0(R10)    # R10 initially has 10000400
400F04    SLT   R13, R12, R9   # R9 initially has 2A
400F08    SW    R13, 0(R11)    # R11 initially has 10000600
400F0C    ADDI  R10, R10, 4
400F10    ADDI  R11, R11, 4
400F14    ADDI  R8, R8, (-1)10 # R8 initially has 2
400F18    BNE  R8, R0, (-7)10
-----
10000400    7
10000404    1F
-----
10000600    ?
10000604    ?
  
```

i) **Obtain** the table that shows the **values** of registers and memory locations used by the above piece of EMY code as shown in class.

ii) **Determine** what this piece of code does. That is, what is its **purpose** in a few sentences ?

iii) **Determine** the total number of memory accesses made for the code.

iv) Assume that the EMY machine language instruction set is added a **new** instruction : **CASTM**. The syntax and semantics of the new instruction are shown below :

The **syntax** of the new instruction : CASTM (Rd), (Rs), Rt

The **semantics** of the new instruction : If $M[Rs] < Rt$ then $M[Rd] \leftarrow 1$ else $M[Rd] \leftarrow 0$

It turns out that this new instruction can be used for the program above. Rewrite the code by using this new EMY instruction in the mnemonic notation.

Your piece of code starts at 400F00. Use software conventions discussed in class. Add comments to your code. What does “CASTM” stand for ?

v) **Obtain** the table that shows the **values** of registers and memory locations used by the piece of EMY code **you** write in part (i) above as shown in class.

vi) **Determine** the total number of memory accesses made for **your** code in part (i) above.

A7) i) The table showing the values of registers and memory locations used by the code is as follows :

Instruction	PC	R8	R9	R10	R11	R12	R13	M[10000400]	M[10000404]	M[10000600]	M[10000604]
Initial	400F00	2	2A	10000400	10000600	?	?	7	1F	?	?
LW R12, 0(R10)	400F04	NS	NS	NS	NS	7	NS	NS	NS	NS	NS
SLT R13, R12, R9	400F08	NS	NS	NS	NS	NS	1	NS	NS	NS	NS
SW R13, 0(R11)	400F0C	NS	NS	NS	NS	NS	NS	NS	NS	1	NS
ADDI R10, R10, 4	400F10	NS	NS	10000404	NS	NS	NS	NS	NS	NS	NS
ADDI R11, R11, 4	400F14	NS	NS	NS	10000604	NS	NS	NS	NS	NS	NS
ADDI R8, R8, (-1) ₁₀	400F18	1	NS	NS	NS	NS	NS	NS	NS	NS	NS
BNE R8, R0, (-7) ₁₀	400F00	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS
LW R12, 0(R10)	400F04	NS	NS	NS	NS	1F	NS	NS	NS	NS	NS
SLT R13, R12, R9	400F08	NS	NS	NS	NS	NS	1	NS	NS	NS	NS
SW R13, 0(R11)	400F0C	NS	NS	NS	NS	NS	NS	NS	NS	NS	1
ADDI R10, R10, 4	400F10	NS	NS	10000408	NS	NS	NS	NS	NS	NS	NS
ADDI R11, R11, 4	400F14	NS	NS	NS	10000608	NS	NS	NS	NS	NS	NS
ADDI R8, R8, (-1) ₁₀	400F18	0	NS	NS	NS	NS	NS	NS	NS	NS	NS
BNE R8, R0, (-7) ₁₀	400F1C	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS

ii) This code works on two vectors which have the same length. It compares each element of the first vector with a constant which is in this case 2A. If the element is less than the constant, it stores a 1 to the corresponding element of the second vector. Otherwise, it stores a 0 to that element. In the given problem, there are two elements per vector.

iii) The code has two iterations. We execute the following instructions with their associated number of memory accesses : $2(LW + SLT + SW + ADDI + ADDI + ADDI + BNE) = 2(2 + 1 + 2 + 1 + 1 + 1 + 1) = 2 * 9 = 18$.

iv) The new piece of code is as follows :

```

400F00      CASTM  (R11), (R10), R9      # If M[R10] < R9, M[R11] gets 1 else 0
400F04      ADDI  R10, R10, 4           # Update the first vector pointer
400F08      ADDI  R11, R11, 4           # Update the second vector pointer
400F0C      ADDI  R8, R8, (-1)10       # Decrement the loop-end counter
400F10      BNE   R8, R0, (-5)10       # If not the end, go back to 400F00

```

CASTM replaces three instructions of the original code. CASTM means **compare and set less than memory**.

v) The table showing the values of registers and memory locations used by the code is as follows :

Instruction	PC	R8	R9	R10	R11	M[10000400]	M[10000404]	M[10000600]	M[10000604]
Initial	400F00	2	2A	10000400	10000600	7	1F	?	?
CASTM (R11), (R10), R9	400F04	NS	NS	NS	NS	NS	NS	1	NS
ADDI R10, R10, 4	400F08	NS	NS	10000404	NS	NS	NS	NS	NS
ADDI R11, R11, 4	400F0C	NS	NS	NS	10000604	NS	NS	NS	NS
ADDI R8, R8, (-1) ₁₀	400F10	1	NS	NS	NS	NS	NS	NS	NS
BNE R8, R0, (-5) ₁₀	400F00	NS	NS	NS	NS	NS	NS	NS	NS
CASTM (R11), (R10), R9	400F04	NS	NS	NS	NS	NS	NS	NS	1
ADDI R10, R10, 4	400F08	NS	NS	10000408	NS	NS	NS	NS	NS
ADDI R11, R11, 4	400F0C	NS	NS	NS	10000608	NS	NS	NS	NS
ADDI R8, R8, (-1) ₁₀	400F10	0	NS	NS	0	NS	NS	NS	NS
BNE R8, R0, (-5) ₁₀	400F14	NS	NS	NS	NS	NS	NS	NS	NS

vi) The CASTM instruction makes three memory accesses. One to fetch the instruction, one to read a location and one to write to a location. We execute the following instructions with their associated number of memory accesses : $2(\text{CASTM} + \text{ADDI} + \text{ADDI} + \text{ADDI} + \text{BNE}) = 2(3 + 1 + 1 + 1 + 1) = 2 * 7 = 14$.

Q8) Consider the following pseudoinstruction whose syntax is shown below :

SWAPMR Rt, Disp(Rs)

The **semantics** of the pseudoinstruction : $Rt \leftarrow M[Rs + \text{Disp}^+]$

a) Assume that the pseudoinstruction is used as follows :

SWAPMR R8, 0(R9)

Implement the pseudoinstruction by using real EMY instructions in the mnemonic notation. Your piece of code starts at 40EC00. Use software conventions discussed in class. Add comments to your code.

b) Assume that the SWAPMR instruction is added to the EMY instruction set.

- i) **Indicate** the instruction format, arguments, addressing modes, memory accesses made, etc. of the new instruction.
 ii) Consider the following piece of EMY mnemonic machine language code :

```
400000 LW R8, 0(R9)
400004 SWAPMR R8, 0(R10)
400008 SW R8, 0(R9)
```

Obtain a table that shows the values of registers and memory locations used by the above piece of EMY code as shown in class. **Pick** appropriate values for registers and memory locations. Also **show** the number of memory accesses made for each instruction. Then, **indicate** in a few sentences what this piece of code does.

A8) a) The SWAPMR R8, 0(R9) instruction is implemented as follows :

```
40EC00 LW R10, 0(R9) # Load from M[R9]
40EC04 SW R8, 0(R9) # Store R8 in M[R9]
40EC08 ADD R8, R10, R0 # Move M[R9] to R8
```

b) i) The instruction format is the **I format**. There are **two** arguments. One argument is a register argument (Rt) which is using the **Register** addressing mode. The other argument is a memory location, using the **2-byte signed displacement** addressing mode. **Three** memory accesses are made for this instruction. One access is to fetch the instruction and two for data (one read and one write).

ii) We assume R9 has 10000000, R10 has 10000F00 and memory locations 10000000 and 10000F00 have A7 and 2CE respectively. The table showing the values of registers and memory locations used by the code is as follows :

Instruction	PC	R8	R9	R10	M[10000000]	M[10000F00]	Mem. Accesses
Initial	400000	?	10000000	10000F00	A7	2CE	-----
LW R8, 0(R9)	400004	A7	NS	NS	NS	NS	2
SWAPMR R8, 0(R10)	400008	2CE	NS	NS	NS	A7	3
SW R8, 0(R9)	40000C	NS	NS	NS	2CE	NS	2

The sequence of instructions swaps two memory locations.

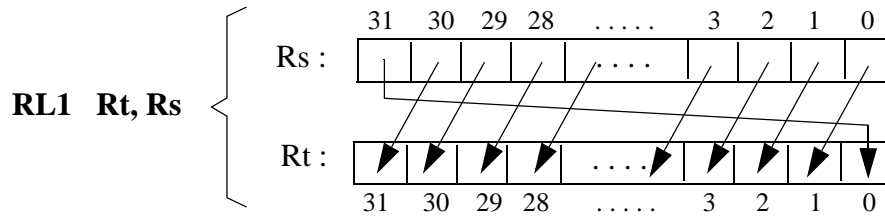
Q9) Consider the following piece of EMY mnemonic machine language program :

```
400000 SLT R9, R8, R0 # R8 initially has F0F0F0F0
400004 SLL R10, R8, 1
400008 ??? ???
40000C ORI R10, R10, 1
```

Assume that the above piece of code implements the following **pseudoinstruction** :

- The **syntax** of the pseudoinstruction : RL1 Rt, Rs
- The **semantics** of the pseudoinstruction : Rt ← Rotate Rs Left by 1

The pseudoinstruction rotates Rs to the left by 1 and then stores on Rt, hence the name “RL1.”



Above, the pseudoinstruction is used as follows : `RL1 R10, R8`

a) Determine the instruction in location 400008 in the code. **Explain** your decision. **Add** comments to each instruction of the code above.

b) Then, Obtain a table that shows the values of registers and memory locations used by the new piece of EMY code as shown in class. Also **show** the number of memory accesses made for each instruction.

c) Assume that the RL1 instruction is added to the EMY instruction set. **Indicate** the instruction format, arguments, addressing modes, memory accesses made, etc. of the new instruction. If there is a **new** addressing mode **not** discussed in class, indicate so.

d) Consider the following piece of EMY mnemonic machine language code that uses the **new** instruction :

```

400000  ADD      R9, R8, R0          # R8 has 4 initially
400004  RL1      R9, R9
400008  ADDI    R10, R10, (-1)10    # R10 has 3 initially
40000C  BNE    R10, R0, (-3)10

```

Obtain a table that shows the values of registers and memory locations used by this piece of EMY code as shown in class. Also **show** the number of memory accesses made for each instruction. **Determine** what this piece of code does ! That is, what is its **purpose** ?

A9) a) The missing instruction should be a BEQ instruction to **skip** the ORI if the rightmost bit should stay 0 :

```

400000  SLT     R9, R8, R0          # R8 is less than R0, store 1 on R9, since F0F0F0F0 < 0
400004  SLL    R10, R8, 1          # Shift left R8 by 1 and place a 0 on the rightmost bit
400008  BEQ    R9, R0, 1          # Since R9 is not equal to R0, do not skip the next instruction
40000C  ORI    R10, R10, 1        # Place a one on the rightmost bit of R10 since R8 is negative

```

b) The execution table is as follows :

Instruction	PC	R8	R9	R10	Mem. Acc.
Initial	400000	F0F0F0F0	?	?	-----
SLT R9, R8, R0	400004	NS	1	NS	1 : IF
SLL R10, R8, 1	400008	NS	NS	E1E1E1E0	1 : IF
BEQ R9, R0, 1	40000C	NS	NS	NS	1 : IF
ORI R10, R10, 1	400010	NS	NS	E1E1E1E1	1 : IF

c) Format, etc. :

- It uses the **R** format since we do **not** need a Disp, Imm or Offset. Also, this is an A/L instruction. Rd and Shamt are not used.
- It has **three** arguments.
 - Rt is a destination register explicitly specified by the instruction : Register addressing mode
 - Rs is a source register explicitly specified by the instruction : Register addressing mode
 - Number “1” is an implied data element : Implied addressing mode
- We make **one** memory access for the new instruction :
 - One to fetch the instruction

d) The table showing the values of registers and memory locations used by the code is as follows :

Instruction	PC	R8	R9	R10	Mem. Acc.	Instruction	PC	R8	R9	R10	Mem. Acc.
Initial	400000	4	?	3	-----	ADDI R10, R10, (-1) ₁₀	40000C	NS	NS	1	1 : IF
ADD R9, R8, R0	400004	NS	4	NS	1 : IF	BNE R10, R0, (-3) ₁₀	400004	NS	NW	NS	1 : IF
RL1 R9, R9	400008	NS	8	NS	1 : IF	RL1 R9, R9	400008	NS	(32) ₁₀	NS	1 : IF
ADDI R10, R10, (-1) ₁₀	40000C	NS	NS	2	1 : IF	ADDI R10, R10, (-1) ₁₀	40000C	NS	NS	0	1 : IF
BNE R10, R0, (-3) ₁₀	400004	NS	NS	NS	1 : IF	BNE R10, R0, (-3) ₁₀	400010	NS	NS	NS	1 : IF
RL1 R9, R9	400008	NS	(16) ₁₀	NS	1 : IF						

This code rotates a register by as many times as another register indicates. In this example, it rotates register R8 by R10 times and stores in R9.

Q10) Consider the following EMY mnemonic machine language program containing a **pseudoinstruction** :

```

400000    LW      R9, 0(R8)           # R8 initially has 10000004
400004    SLL    R9, R9, 1
400008    SLT    R10, R9, R0
40000C    BEQ    R10, R0, 1
400010    ADD    R11, R11, R9        # R11 initially has 0
400014    SW     R9, 0(R8)
400018    BLP    R8, (-7)10         # If not the end of the loop, go back to 400000
40001C    SW     R11, 0(R12)        # R12 has 1000A000
----
10000000    1
10000004    2
----
1000A000    ?

```

The **BLP** pseudoinstruction is a “Branch Loop” instruction that has the following syntax and semantics :

BLP Rs, Offset \longrightarrow 1) Rs \longleftarrow Rs - 4
 2) If Rs \neq FFFFFFFC then PC \longleftarrow PC + (Offset⁺ * 4)

i) **Obtain** a table that shows the values of registers and memory locations used by the above piece of EMY code as shown in class. Also **show** the number of memory accesses made for each instruction.

ii) What is the advantage of the BLP instruction ? **Rewrite** the above code with comments and with **only** actual EMY instructions that also implement the BLP instruction.

A10) i) The table showing the values of registers and memory locations used by the code is as follows :

Instruction	PC	R8	R9	R10	R11	R12	M[10000000]	M[10000004]	M[1000A000]	Mem. Acc.
Initial	400000	10000004	?	?	0	1000A000	1	2	?	-----
LW R9, 0(R8)	400004	NS	2	NS	NS	NS	NS	NS	NS	2 : IF & DR
SLL R9, R9, 1	400008	NS	4	NS	NS	NS	NS	NS	NS	1 : IF
SLT R10, R9, R0	40000C	NS	NS	0	NS	NS	NS	NS	NS	1 : IF
BEQ R10, R0, 1	400014	NS	NS	NS	NS	NS	NS	NS	NS	1 : IF
SW R9, 0(R8)	400018	NS	NS	NS	NS	NS	NS	4	NS	2 : IF & DW
BLP R8, (-7) ₁₀	400000	10000000	NS	NS	NS	NS	NS	NS	NS	1 : IF
LW R9, 0(R8)	400004	NS	1	NS	NS	NS	NS	NS	NS	2 : IF & DR
SLL R9, R9, 1	400008	NS	2	NS	NS	NS	NS	NS	NS	1 : IF
SLT R10, R9, R0	40000C	NS	NS	0	NS	NS	NS	NS	NS	1 : IF
BEQ R10, R0, 1	400014	NS	NS	NS	NS	NS	NS	NS	NS	1 : IF
SW R9, 0(R8)	400018	NS		NS	NS	NS	2	NS	NS	2 : IF & DW
BLP R8, (-7) ₁₀	40001C	FFFFFFFC	NS	NS	NS	NS	NS	NS	NS	1 : IF
SW R11, 0(R12)	400020	NS	NS	NS	NS	NS	NS	NS	0	2 : IF & DW

ii) The BLP instruction helps if a vector is stored starting at 10000000 and an application can work on its elements from the end towards the beginning. Since we execute scientific applications with a lot of loops, the instruction can be very helpful. As it is seen in part **(iii)** below, it combines **four** instructions into one.

The original code is rewritten without the BLP below :

```

400000    LUI    R13, FFF          # Initialize R13 to FFF0000
400004    ORI    R13, R13, FFFC   # R13 now has FFFFFFFC to be compared each iteration
400008    LW     R9, 0(R8)        # R8 initially has 10000004 and points at the vector to work on
40000C    SLL   R9, R9, 1        # Shift left R9 by 1
400010    SLT   R10, R9, R0      # Is R9 less than 0 ?
400014    BEQ   R10, R0, 1       # If no, skip the next instruction
400018    ADD   R11, R11, R9     # If yes, add R11 and R9. R11 initially has 0
40001C    SW    R9, 0(R8)       # Store R9 in the same memory location
400020    ADDI  R8, R8, (-4)10   # The vector pointer register (R8) is updated
400024    BNE   R8, R13, (-8)10  # If not the end of the loop, go back to 400000
400028    SW    R11, 0(R12)     # Store R11 in memory pointed by R12. R12 has 1000A000

```

Q11) Consider the following pseudoinstruction in the mnemonic machine language notation :

MULTM Rd, (Rs), (Rt) ==> (Rd, Rd + 1) ← (M[Rs]) * (M[Rt])

Register Rd is stored the most significant bits of the multiplication result. For example :

MULTM R8, (R10), (R11) ==> (R8, R9) ← (M[R10]) * (M[R11])

a) **Implement** the instruction

MULTM R8, (R10), (R11)

by using a few actual EMY instructions in the mnemonic notation. Your piece of code starts at 400A00. Use software conventions discussed in class. **Add** comments to your code.

b) Then, **obtain** a table that shows the values of registers and memory locations used by the piece of EMY code in part (a) as shown in class. Also **show** the number of memory accesses made for each instruction. Assume that R10 and R11 contain 10000000 and 10008000, respectively. Finally, assume that the numbers multiplied are 6 and 2.

c) Assume that this instruction is added to the EMY instruction set. **Indicate** its format, arguments, addressing modes, memory accesses made, etc. If there is a **new** addressing mode that is **not** discussed in class, indicate so.

d) Consider the following piece of EMY mnemonic machine language code that uses the **new** instruction :

```

400F00  MULTM    R8, (R10), (R11)
400F04  SW       R8, 0(R12)
400F08  ???
400F0C  ADDI      R10, R10, 4
400F10  ???
400F14  ADDI      R12, R12, 8
400F18  ADDI      R13, R13, (-1)10      # R13 has 1 initially
400F1C  BNE      R13, R0, (-7)10
    
```

Assume that R10, R11 and R12 contain 10000000, 10008000 and 1000A000, respectively. Also assume that the numbers multiplied are 6 and 2.

Determine the instructions in locations 400F08 and 400F10 in the code. **Explain** your decision for each instruction. **Determine** what this piece of code does ! That is, what is its **purpose** ?

A11) a) The implementation of the “**MULTM R8, (R10), (R11)**” is as follows :

```

400A00  LW       R12, 0(R10)      # Read the first number from the memory
400A04  LW       R13, 0(R11)     # Read the second number from the memory
400A08  MULT    R12, R13        # Multiply the two numbers
400A0C  MFHI    R8                # Move the most significant bits of the result to R8
400A10  MFLO    R9                # Move the least significant bits of the result to R9
    
```

b) The execution table is as follows :

Instruction	PC	R8	R9	R10	R11	R12	R13	Hi	Lo	M[10000000]	M[10008000]	Mem. Acc.
Initial	400A00	?	?	10000000	10008000	?	?	?	?	6	2	-----
LW R12, 0(R10)	400A04	NS	1	NS	NS	6	NS	NS	NS	NS	NS	2 : IF & DR
LW R13, 0(R11)	400A08	NS	NS	NS	NS	NS	2	NS	NS	NS	NS	2 : IF & DR
MULT R12, R13	40A00C	NS	NS	NS	NS	NS	NS	0	C	NS	NS	1 : IF
MFHI R8	400A10	0	NS	NS	NS	NS	NS	NS	NS	NS	NS	1 : IF
MFLO R9	400A14	NS	C	NS	NS	NS	NS	NS	NS	NS	NS	1 : IF

c) Format, etc. :

- It uses the **R** format since we need Rd.
- It has **three** arguments.
 - Rd and “Rd+1” are destination registers where Rd is explicitly specified by the instruction : Register addressing mode. The other is implied : The Implied addressing mode.
 - The source arguments are two memory locations whose addresses are contained by two registers explicitly specified. This is the register indirect addressing mode, **not** discussed in class.
- We make **three** memory access for the new instruction :
 - One to fetch the instruction
 - Two to read two memory locations for data

d) This piece of program multiplies two vectors stored in the memory and pointed by R10 and R11. It stores the results in the memory pointed by R12. The result is stored in two memory locations : The most significant bits are stored first and in the next location the least significant bits are stored. Then, the missing instructions are as follows :

400F08 SW R9, 4(R12) # Because the least significant bits are kept by R9 and stored in the memory pointed by R12. But R12 needs to be added “4” so that it is the next location

400F10 ADDI R11, R11, 4 # Because, R11 points at the second vector multiplied. This pointer needs to be updated to be ready for the next iteration

Q12) The smallest negative integer number represented by an EMY GPR register is $(10000\dots0)_2 = -(2^{31})_{10}$

Represent this smallest negative integer number precisely in the IEEE-754 single-precision format. Note that the largest negative integer representable by EMY is $(11111\dots1)_2 = (-1)_{10}$. Your answer must clearly show that you did not use a calculator to convert numbers from one system to another.

A12) First of all, the number is a negative number, we note it by assigning a 1 to the sign bit of the FP number.

Then, we concentrate on the positive number : $2^{31} = 1.0 * 2^{31}$

We see that the mantissa is all zeros and the real exponent is 31. The biased exponent, BE, is

$$BE = e + 127 = 31 + 127 = 158 = 2^7 + 2^4 + 2^3 + 2^2 + 2^1 = (10011110)_2$$

Finally, the FP number in the IEEE-754 format : 1 10011110 000000000000000000000000

In hexadecimal, the number is CF000000.

Q13) Consider the following two memory locations with their sequences of bits :

```

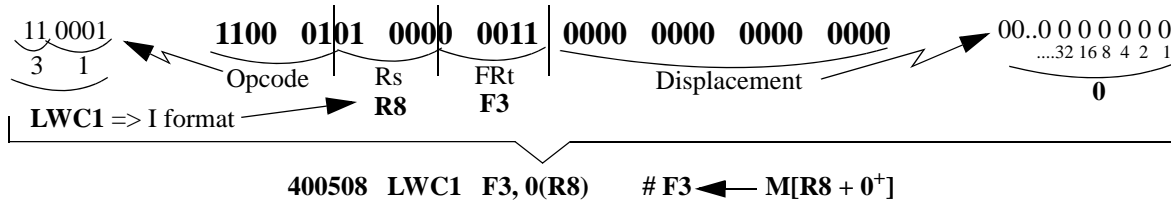
400508    1100 0101 0000 0011 0000 0000 0000 0000
-----
10002000  0100 0000 0110 0000 0000 0000 0000 0000

```

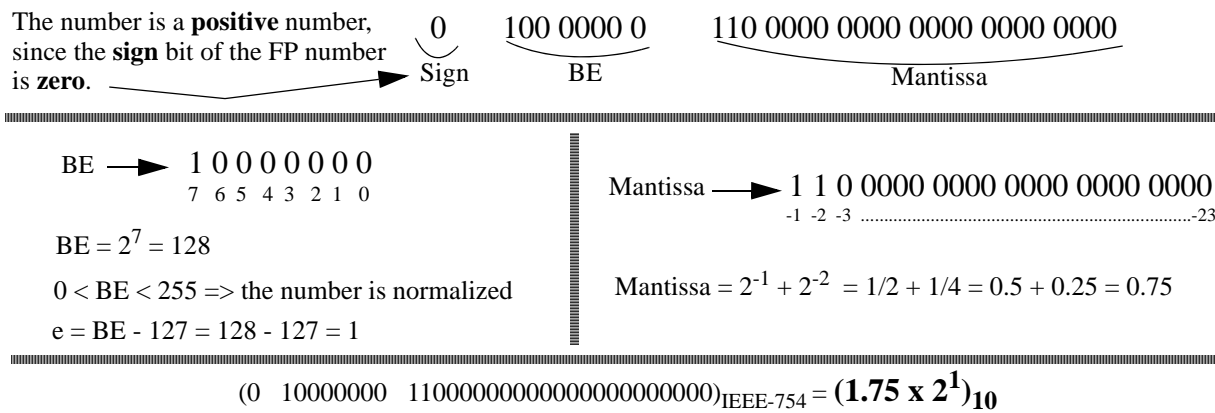
i) **Determine** what these sequences of bits **exactly** represent. If there is any **Rs** register used, its value is 10002000. Note that these memory locations are used for an application for which the MIPS software conventions are followed.

ii) Describe what happens after these memory locations are processed.

A13) i) The first bit sequence is in 400508. It must be an instruction according to the MIPS software conventions.



Since R8 is Rs and the question indicates the Rs register has 10002000, then F3 is loaded the content of memory location 10002000. Therefore, memory location 10002000 must have a **single-precision** floating-point number :



ii) After processing the two memory locations, floating-point register F3 is loaded with value $(1.75 \times 2^1)_{10}$.

Q14) There is an EMY instruction in location 40020A. It works on F8 and F10 and stores the result in F6. The initial values of F8 and F10 are as follows :

F8 = 1100 0001 1000 0000 0000 0000 0000 0000
 F10 = 0100 0001 0000 0000 0000 0000 0000 0000

After the instruction is executed, the value of F6 is : F6 = 1100 0000 0000 0000 0000 0000 0000 0000

Write down the executed mnemonic machine language instruction precisely. Your answer must clearly show that you did not use a calculator to convert numbers from one system to another.

A14) Since the register names start with an “F,” we have floating-point numbers and a floating-point instruction :

F8 = 1100 0001 1000 0000 0000 0000 0000 0000

The leftmost bit is the sign bit = 1 => The sign of the number is **negative**.

The next eight bits indicate the biased exponent, $BE = 1000\ 0011 = 2^7 + 2^1 + 2^0 = 128 + 2 + 1 = 131$

‰ Since $BE = 131$, the FP number is normalized.

‰ The real exponent is, $e = 131 - BE = 131 - 127 = 4$

The remaining 23 bits are for the fraction and they are all zero : **F8 = -1.0 x 2⁴**

F10 = 0100 0001 0000 0000 0000 0000 0000 0000

The leftmost bit is the sign bit = 0 => The sign of the number is **positive**.

The next eight bits indicate the biased exponent, $BE = 1000\ 0010 = 2^7 + 2^1 = 128 + 2 = 130$

‰ Since $BE = 130$, the FP number is normalized.

‰ The real exponent is, $e = 130 - 127 = 3$

The remaining 23 bits are for the fraction and they are all zero : **F10 = 1.0 x 2³**

F6 = 1100 0000 0000 0000 0000 0000 0000 0000

The leftmost bit is the sign bit = 1 => The sign of the number is **negative**.

The next eight bits indicate the biased exponent, $BE = 1000\ 0000 = 2^7 = 128$

‰ Since $BE = 128$, the FP number is normalized.

‰ The real exponent is, $e = 128 - 127 = 1$

The remaining 23 bits are for the fraction and they are all zero : **F6 = -1.0 x 2¹**

The arguments of the operation are as follows : $-1.0 \times 2^1 = -1.0 \times 2^4$ operation 1.0×2^3

In the EMY FP instruction set in Chapter 4, the only instruction that applies is the single-precision divide instruction :

40020A DIV.S F6, F8, F10

Q15) Consider the following IEEE-754 single-precision number : **0100 0100 0100 0000 0000 0000 0000 0000**

Convert the above number to a **decimal** number of the form : **k.m * 10^z**

“k” is a single decimal digit. “m” and “z” contain one or more decimal digits. Your answer must clearly show that you did not use a calculator to convert numbers from one system to another.

A15) 0100 0100 0100 0000 0000 0000 0000 0000

The leftmost bit is the sign bit = 0 => The sign of the floating-point number is **positive**.

The next eight bits indicate the biased exponent, $BE = 1000\ 1000 = 2^7 + 2^3 = 128 + 8 = 136$

‰ Since $BE = 136$, the FP number is normalized.

‰ The real exponent is, $e = 136 - 127 = 9$

The remaining 23 bits are for the fraction = 1000...0

‰ fraction = $.100...0 = 2^{-1} = .5$

The number is then $1.5 \times 2^9 = 1.5 \times 512 = 768$

The answer is **7.68×10^2** .

Q16) For both parts below, your answers must clearly show that you did not use a calculator.

a) Convert the following number to an IEEE-754 single-precision number : **3.5×2^{-7}**

b) The EMY computer executes the following instruction : DIV R9, R12

Prior to the execution, R9 has FFFFFFFD and R12 has 7. What is the effect of this instruction ?

A16) a) $3.5 * 2^{-7} = (?)_2 \Rightarrow$

The number is a positive number, therefore the sign bit of the FP number will be **zero**.

The FP number with normalization is calculated as follows :

<p>The integer part :</p> $\begin{aligned} 3 / 2 &= 1 \ \& \ 1 \\ 1 / 2 &= 0 \ \& \ 1 \\ \Rightarrow (3)_{10} &= (11)_2 \text{ Unsigned} \end{aligned}$	<p>The fraction part :</p> $\begin{aligned} 0.5 * 2 &= 1.0 \\ \Rightarrow (0.5)_{10} &= (0.1)_2 \text{ Unsigned} \end{aligned}$	$3.5 * 2^{-7} = 11.1 * 2^{-7} \text{ is NOT normalized.}$ $= \mathbf{1.11} * 2^{-6} \text{ after normalization.}$
---	---	---

The exponent is calculated as follows :

$e = -6 \Rightarrow BE = e + 127 = -6 + 127 = 121$	$\left. \begin{array}{l} 121 / 2 = 60 \ \& \ 1 \\ 60 / 2 = 30 \ \& \ 0 \\ 30 / 2 = 15 \ \& \ 0 \\ 15 / 2 = 7 \ \& \ 1 \\ 7 / 2 = 3 \ \& \ 1 \\ 3 / 2 = 1 \ \& \ 1 \\ 1 / 2 = 0 \ \& \ 1 \end{array} \right\}$	<p>Biased Exponent (BE) :</p> $(121)_{10} = (1111001)_2 \text{ Unsigned}$ <p>By using 8 bits :</p> <p>01111001 = BE</p>
--	---	--

The FP number in the IEEE-754 format : $0, \underbrace{01111001}_{\text{BE}} \underbrace{1100...0}_{\text{M (23 bits)}}$

s BE M (23 bits)

b) The instruction is a DIV instruction. Thus, it is an **integer** division. It is a signed integer division

R9 = FFFFFFFDF R12 = 7

R9 = 1111 1111 1111 1111 1111 1111 1101 1111

R9 has a negative number. If we take it's 2's complement :

=> 0000 0000 0000 0000 0000 0000 0010 0001

= $2^5 + 2^0 = 32 + 1 = (+33)_{10}$

Therefore, R9 has $(-33)_{10}$

$$\frac{R9}{R12} = \frac{-33}{7} = -4 \text{ \& } -5$$

↑
↑
 Quotient Remainder

Hi register gets the remainder. It has -5 => $(+5)_{10} = (00...0101)_2 \Rightarrow (11...1011)_2 \Rightarrow \text{FFFFFFFB}$

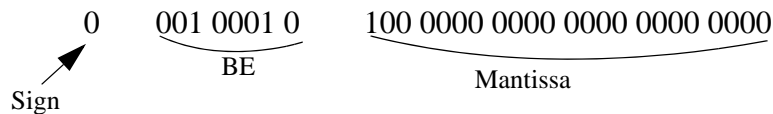
Lo register gets the quotient. It has -4 => $(+4)_{10} = (00...0100)_2 \Rightarrow (11...1100)_2 \Rightarrow \text{FFFFFFFC}$

Q17) Convert the following **single-precision IEEE-754-format** FP number to a **decimal** number as shown in class such that all calculations are **manual** :

$$(0001\ 0001\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000)_{\text{IEEE-754}} = (?)_{10}$$

A17) We will show the **single-precision IEEE-754** format in decimal :

$$(0001\ 0001\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000)_{10} = (?)_{\text{Decimal}}$$



The number is a **positive** number, since the **sign** bit of the FP number is **zero**.

BE	→	0 0 1 0 0 0 1 0	}	$0 < \text{BE} < 255 \Rightarrow$ the number is normalized $e = \text{BE} - 127 = 34 - 127 = -93$
		7 6 5 4 3 2 1 0		
		BE = $2^5 + 2^1 = 32 + 2 = 34$		

Mantissa	→	1 0 0 0000 0000 0000 0000 0000
		-1 -2 -3-23

$$\text{Mantissa} = 2^{-1} = 1/2 = 0.5$$

$$(0\ 00100010\ 100000000000000000000000)_{\text{IEEE-754}} = (1.5 \times 2^{-93})_{10}$$

Q18) Convert the following decimal number to a **single-precision** IEEE-754-format FP number as shown **in class** such that all calculations are manual : $(29)_{10} = (?)_{\text{IEEE-754}}$

A18) $(29)_{10} = (?)_{\text{IEEE-754}}$

The number is a positive number, therefore the **sign** bit of the FP number will be **zero**.

The decimal number does not have a fraction part. The FP number with normalization is calculated as follows :

<p>The integer part :</p> <p>$29 / 2 = 14 \text{ \& } 1$ $14 / 2 = 7 \text{ \& } 0$ $7 / 2 = 3 \text{ \& } 1$ $3 / 2 = 1 \text{ \& } 1$ $1 \text{ \& } 2 = 0 \text{ \& } 1$ msb $\implies (29)_{10} = (11101)_2$ Unsigned</p>	<p>$29 = 11101 * 2^0$ which is NOT normalized</p> <p>$= 1.1101 * 2^4$ after normalization</p> <div style="border: 1px solid black; padding: 2px; margin: 5px auto; width: 80%;"> <p>Fraction is 1101000...000</p> </div> <div style="border: 1px solid black; padding: 2px; margin: 5px auto; width: 80%;"> <p>$e = 4$ $\implies \text{BE} = e + 127 = 4 + 127 = 131$</p> </div>	<p>BE :</p> <p>$131 / 2 = 65 \text{ \& } 1$ $65 / 2 = 32 \text{ \& } 1$ $32 / 2 = 16 \text{ \& } 0$ $16 / 2 = 8 \text{ \& } 0$ $8 / 2 = 4 \text{ \& } 0$ $4 / 2 = 2 \text{ \& } 0$ $2 / 2 = 1 \text{ \& } 0$ $1 \text{ \& } 2 = 0 \text{ \& } 1$ msb $\implies (10000011)_2$ Unsigned</p>
---	---	---

The FP number in the IEEE-754 format : $\underbrace{0}_{\text{Sign}} \underbrace{10000011}_{\text{BE}} \underbrace{1101000\dots000}_{\text{M (23 bits)}}$