DUE : March 3, 2014

READ:
- Related sections of Chapter 2
- Related sections of Chapter 3
- Related sections of Appendix A
- Related sections of Appendix E

ASSIGNMENT: There are thirteen questions eleven 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 2.15 of Chapter II.

First, rewrite the assembly language instruction in the mnemonic notation. Assume that the instruction is in memory location 400000. Second, obtain the bits of the instruction, i.e. the corresponding machine language instruction in location 400000. Show the work during the conversions as done in class. Finally, use Hex coding to get the Hex digits as asked by the question.

Note that the “type” asked in the question is the format type of the instruction.

II) Solve Problems 2.18.1, 2.18.2 and 2.18.3.

Explain your decisions!

III) Solve Problem 2.19.2 of Chapter II.

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 left and AND operations 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 $t2 are R8 and R10, respectively.
IV) Solve Problem 2.20 of Chapter II.

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 and logic operations 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 and R9 has 12345678 as the initial value.
⇒ What you are asked to do is the following:

```
R8 :
   31  17  16
   15 bits Field 11  10  0
R9 :
   31  6 bits 26 25
   Field 26 bits 0
```

V) Solve Problem 2.23 of Chapter II.

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, register $t2 is R10.

VI) Solve Problems 2.25.1 and 2.25.2.

For 2.25.1: Explain your decision!

For 2.25.2: 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 “RPT” starts at 400200. What does “RPT” Stand for? Explain!
⇒ Assume that register R10 has 1 as the initial value.
VII) Solve Problem 2.38 of Chapter II.

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. Also, show the individual bits to describe the Load Byte Unsigned operation. Note the following:

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

VIII) Consider the following C language statement:

\[
a = 25 | b
\]

Write the corresponding minimal sequence of mnemonic machine language program. Assume that the value of “b” is in R9 and the value of “a” should be stored in R8.

In order to show clearly that the new code is correct, 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.
⇒ Assume that R9 has (6)\(_{10}\):
⇒ The operation performed is an OR operation : “|”.
⇒ Number 25 in the C-like statement in the question is in decimal. But, the hexadecimal notation is the default case in our mnemonic machine language programs.

IX) Solve Problem 2.39 of Chapter II.

The pseudoinstruction that stores the value in the assembly language notation is the following:

⇒ LI $t1, large => $t1 = large

“LI” stands for “Load Immediate”!

Implement the above pseudoinstruction in terms of (by using) actual EMY instructions. Write a mnemonic machine language program with comments that implements the architectural operations required by the pseudo instruction. Note the following:

⇒ Assume that your mnemonic machine language program starts at memory location 400400 and looks like as follows:
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.”

XI) Solve Problem 3.21 of Chapter III.

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.

XII) Solve Problem 3.22 of Chapter III.

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.
XIII) Solve Problem 3.23 of Chapter III.

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.

RELEVANT QUESTIONS AND ANSWERS

Q1) The EMY machine language instruction set does not have the following instruction which is shown in the mnemonic notation:

\[
400300 \quad \text{ADDMRM} \quad R8, (R9, R10) \quad \# M[R9] \leftarrow R8 + M[R9 + R10]
\]

i) Implement the instruction: ADDMRM R8, (R9, R10)

by using a few actual EMY instructions in the mnemonic notation. Your piece of code starts at 400300. 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 “ADDMRM” stand for?

A1) The given instruction is the following:

\[
400300 \quad \text{ADDMRM} \quad R8, (R9, R10) \quad \# M[R9] \leftarrow R8 + M[R9 + R10]
\]

i) We see that we add a register and a memory operand and store the result in another memory location. We can implement it by using four instructions:

\[
\begin{align*}
400300 & \quad \text{ADD} \quad R11, R9, R10 \quad \# \text{Calculate the effective address of the memory operand} \\
400304 & \quad \text{LW} \quad R12, 0(R11) \quad \# \text{Load the memory operand} \\
400308 & \quad \text{ADD} \quad R13, R8, R12 \quad \# \text{Add the register and the memory operand} \\
40030C & \quad \text{SW} \quad R13, 0(R9) \quad \# \text{Store the result to the other memory location}
\end{align*}
\]

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

The semantics of the new instruction: M[Rt] \leftarrow Rs + M[Rt + Rd]

The format is the R format:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Rs</th>
<th>Rt</th>
<th>Rd</th>
<th>shamt</th>
<th>2nd Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

shamt is not used

Three arguments are used by the instruction: There are two memory arguments and a register argument.

One of the operands is a register and so we use the register addressing mode. The second operand is a memory location whose address is the sum of two registers for which we use the Register Indexed Indirect addressing mode. This addressing mode is not covered in class. The destination argument is a memory location whose address is provided by a register for which we use the register indirect addressing mode. This is also not covered in class. EMY is not a
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:

\[
\text{ADDI} \quad $t0, $t1, \text{big}
\]

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:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>400420</td>
<td>40400000</td>
</tr>
<tr>
<td>400424</td>
<td>C5040420</td>
</tr>
</tbody>
</table>

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:

```
 1100 0101 0000 0100 0000 0100 0010 0000
  C     5     0     4     0     4    2     0
```

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

```
0100 0000 0 100 0000 0000 0000 0000 0000 0000 0000
4     0      4      0      0     0      0      0
```

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: \((+1.1\times2^1)\) and the FP number in decimal: \((+1.5\times2^1)\) = 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 DECM 2C(R8) # M[R8 + 2C+] → (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 DECM 2C(R8) # M[R8 + 2C+] ← M[R8 + 2C+] - 1
```

i) We see that we decrement a memory location by 1. We can implement it by using three instructions:
ii) The syntax of the new instruction: `DECM Disp(Rs)`

The semantics of the new instruction: $M[Rs + Disp] <--- M[Rs + Disp] - 1$

The format is the **I format**:  

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Rs</th>
<th>Rt</th>
<th>DOImm</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

Rt is **not** used

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:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>R2</th>
<th>R4</th>
<th>R5</th>
<th>R8</th>
<th>R9</th>
<th>R10</th>
<th>R31</th>
<th>M[10000000]</th>
<th>Memory Accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>405000</td>
<td>SLT</td>
<td>R8</td>
<td>R4</td>
<td>R0</td>
<td>2</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>405004</td>
<td>BNE</td>
<td>R8</td>
<td>R0</td>
<td>5</td>
<td>8400</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>10000000</td>
</tr>
<tr>
<td>405008</td>
<td>SRL</td>
<td>R4</td>
<td>R4</td>
<td>1</td>
<td>8000</td>
<td>-</td>
<td>402548</td>
<td>80000000</td>
<td>-</td>
</tr>
<tr>
<td>40500C</td>
<td>ADDI</td>
<td>R5</td>
<td>R5</td>
<td>(-1)_{10}</td>
<td>4</td>
<td>SRL</td>
<td>R4</td>
<td>R4</td>
<td>1</td>
</tr>
<tr>
<td>405010</td>
<td>BNE</td>
<td>R5</td>
<td>R0</td>
<td>(-3)_{10}</td>
<td>8400</td>
<td>?</td>
<td>?</td>
<td>10000000</td>
<td>402548</td>
</tr>
<tr>
<td>405014</td>
<td>ADD</td>
<td>R2</td>
<td>R4</td>
<td>R0</td>
<td>8000</td>
<td>-</td>
<td>402548</td>
<td>80000000</td>
<td>-</td>
</tr>
<tr>
<td>405018</td>
<td>JR</td>
<td>R31</td>
<td>8000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>80000000</td>
<td>Continue</td>
</tr>
<tr>
<td>40501C</td>
<td>LW</td>
<td>R9</td>
<td>0(R10)</td>
<td>8000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>80000000</td>
<td>-</td>
</tr>
<tr>
<td>405020</td>
<td>SRL</td>
<td>R4</td>
<td>R4</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>405024</td>
<td>OR</td>
<td>R4</td>
<td>R9</td>
<td>R4</td>
<td>8000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>405028</td>
<td>ADDI</td>
<td>R5</td>
<td>R5</td>
<td>(-1)_{10}</td>
<td>8400</td>
<td>?</td>
<td>?</td>
<td>10000000</td>
<td>402548</td>
</tr>
<tr>
<td>40502C</td>
<td>BNE</td>
<td>R5</td>
<td>R0</td>
<td>(-4)_{10}</td>
<td>4</td>
<td>SRL</td>
<td>R4</td>
<td>R4</td>
<td>1</td>
</tr>
<tr>
<td>405030</td>
<td>ADD</td>
<td>R2</td>
<td>R4</td>
<td>R0</td>
<td>8000</td>
<td>-</td>
<td>402548</td>
<td>80000000</td>
<td>-</td>
</tr>
<tr>
<td>405034</td>
<td>JR</td>
<td>R31</td>
<td>8000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>80000000</td>
<td>-</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>PC</th>
<th>Instruction</th>
<th>R2</th>
<th>R4</th>
<th>R5</th>
<th>R8</th>
<th>R9</th>
<th>R10</th>
<th>R31</th>
<th>M[10000000]</th>
<th>Memory Accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>405000</td>
<td>?</td>
<td>?</td>
<td>8400</td>
<td>2</td>
<td>?</td>
<td>?</td>
<td>10000000</td>
<td>402548</td>
<td>80000000</td>
<td>-</td>
</tr>
<tr>
<td>......</td>
<td>...</td>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
<td>Continue</td>
<td>......</td>
</tr>
</tbody>
</table>

Continue the table until you cannot anymore...

b) Explain what this piece of program does in a few sentences.
### A5) a)

<table>
<thead>
<tr>
<th>PC</th>
<th>Instruction</th>
<th>R2</th>
<th>R4</th>
<th>R5</th>
<th>R8</th>
<th>R9</th>
<th>R10</th>
<th>R31</th>
<th>M[10000000]</th>
<th>Memory Accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>405000</td>
<td>?</td>
<td>?</td>
<td>8400</td>
<td>2</td>
<td>?</td>
<td>?</td>
<td>10000000</td>
<td>402548</td>
<td>80000000</td>
<td>-</td>
</tr>
<tr>
<td>405004</td>
<td>SLT R8, R4, R0</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>0</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>Inst. Read</td>
</tr>
<tr>
<td>405008</td>
<td>BNE R8, R0, 5</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>Inst. Read</td>
</tr>
<tr>
<td>40500C</td>
<td>SRL R4, R4, 1</td>
<td>NS</td>
<td>4200</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>Inst. Read</td>
</tr>
<tr>
<td>405010</td>
<td>ADDI R5,R5, (-1)$_{10}$</td>
<td>NS</td>
<td>NS</td>
<td>1</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>Inst. Read</td>
</tr>
<tr>
<td>405008</td>
<td>BNE R5, R0, (-3)$_{10}$</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>Inst. Read</td>
</tr>
<tr>
<td>40500C</td>
<td>SRL R4, R4, 1</td>
<td>NS</td>
<td>2100</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>Inst. Read</td>
</tr>
<tr>
<td>405010</td>
<td>ADDI R5, R5, (-1)$_{10}$</td>
<td>NS</td>
<td>NS</td>
<td>0</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>Inst. Read</td>
</tr>
<tr>
<td>405014</td>
<td>BNE R5, R0, (-3)$_{10}$</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>Inst. Read</td>
</tr>
<tr>
<td>405018</td>
<td>ADD R2, R4, R0</td>
<td>2100</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>Inst. Read</td>
</tr>
<tr>
<td>402548</td>
<td>JR R31</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>Inst. Read</td>
</tr>
</tbody>
</table>

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

\[ \text{ASR} \quad \text{Rs, } k \]

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:

\[ 400000 \quad \text{SLTM} \quad R8, (R9), (R10) \quad \# \text{ If } \text{M[R9]} < \text{M[R10]} \text{ then } R8 \quad \rightarrow 1 \text{ else } R8 \quad \rightarrow 0 \]

i) Implement the instruction

\[ \text{SLTM} \quad R8, (R9), (R10) \]

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:

\[
\begin{array}{c}
400000 & \text{SLTM} & R8, (R9), (R10) & \# \text{If } M[R9] < M[R10] \text{ then } R8 \leftarrow 1 \text{ else } R8 \leftarrow 0
\end{array}
\]

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

\[
\begin{array}{c}
400000 & \text{LW} & R11, 0(R9) & \# \text{Read the memory location pointed by } R9 \\
400004 & \text{LW} & R12, 0(R10) & \# \text{Read the memory location pointed by } R10 \\
400008 & \text{SLT} & R8, R11, R12 & \# \text{Compare them and store the result in } R8
\end{array}
\]

ii) The syntax of the new instruction: \text{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 \textbf{R} format:

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
\text{Opcode} & \text{Rs} & \text{Rt} & \text{Rd} & \text{shamt} & \text{2nd Opcode} \\
\hline
0 & 6 & 5 & 5 & 5 & 2nd Opcode \\
\hline
\end{tabular}
\end{center}

\text{shamt is not used}

Five arguments are used by the instruction:

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

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

Q7) Consider the following piece of \textbf{EMY mnemonic} machine language program:

\[
\begin{array}{c}
400F00 & \text{LW} & R12, 0(R10) & \# \text{R10 initially has 1000400} \\
400F04 & \text{SLT} & R13, R12, R9 & \# \text{R9 initially has 2A} \\
400F08 & \text{SW} & R13, 0(R11) & \# \text{R11 initially has 1000600} \\
400F0C & \text{ADDI} & R10, R10, 4 & \\
400F10 & \text{ADDI} & R11, R11, 4 & \\
400F14 & \text{ADDI} & R8, R8, (-1)_{10} & \# \text{R8 initially has 2} \\
400F18 & \text{BNE} & R8, R0, (-7)_{10} & \\
\hline
10000400 & 7 & \\
10000404 & 1F & \\
\hline
10000600 & ? & \\
10000604 & ? & \\
\hline
\end{array}
\]

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

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

iii) \textbf{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] ← 1 else M[Rd] ← 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:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>400F00</td>
<td>2</td>
<td>2A</td>
<td>10000400</td>
<td>10000600</td>
<td>?</td>
<td>?</td>
<td>7</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>LW</td>
<td>400F04</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>7</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>SLT</td>
<td>400F08</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>7</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>SW</td>
<td>400F0C</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>7</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI</td>
<td>400F10</td>
<td>NS</td>
<td>NS</td>
<td>10000404</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI</td>
<td>400F14</td>
<td>NS</td>
<td>NS</td>
<td>10000604</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI</td>
<td>400F18</td>
<td>1</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>BNE</td>
<td>400F00</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>LW</td>
<td>400F04</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1F</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>SLT</td>
<td>400F08</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>SW</td>
<td>400F0C</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI</td>
<td>400F10</td>
<td>NS</td>
<td>NS</td>
<td>10000408</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI</td>
<td>400F14</td>
<td>NS</td>
<td>NS</td>
<td>10000608</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI</td>
<td>400F18</td>
<td>0</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>BNE</td>
<td>400F1C</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
</tbody>
</table>

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:

```assembly
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)       # 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:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>400F00</td>
<td>2</td>
<td>2A</td>
<td>10000400</td>
<td>10000600</td>
<td>7</td>
<td>IF</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>CASTM (R11), (R10), R9</td>
<td>400F04</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI R10, R10, 4</td>
<td>400F08</td>
<td>NS</td>
<td>NS</td>
<td>10000404</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI R11, R11, 4</td>
<td>400F0C</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>10000604</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI R8, R8, (-1)10</td>
<td>400F10</td>
<td>1</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>BNE R8, R0, (-5)10</td>
<td>400F00</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>CASTM (R11), (R10), R9</td>
<td>400F04</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI R10, R10, 4</td>
<td>400F08</td>
<td>NS</td>
<td>NS</td>
<td>10000408</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI R11, R11, 4</td>
<td>400F0C</td>
<td>NS</td>
<td>NS</td>
<td>10000608</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>ADDI R8, R8, (-1)10</td>
<td>400F10</td>
<td>0</td>
<td>NS</td>
<td>NS</td>
<td>0</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
<tr>
<td>BNE R8, R0, (-5)10</td>
<td>400F14</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
</tr>
</tbody>
</table>

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: 

\[2(CASTM + ADDI + ADDI + ADDI + BNE) = 2(3 + 1 + 1 + 1 + 1) = 2 \times 7 = 14.\]

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

```
SWAPMR    Rt, Disp(Rs)
```

The **semantics** of the pseudoinstruction: \(Rt \leftrightarrow M[Rs + Disp^+]\)

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 is 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:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>400000</td>
<td>?</td>
<td>10000000</td>
<td>1000F00</td>
<td>A7</td>
<td>2CE</td>
<td>-----</td>
</tr>
<tr>
<td>LW</td>
<td>400004</td>
<td>A7</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>2</td>
</tr>
<tr>
<td>SWAPMR</td>
<td>400008</td>
<td>2CE</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>A7</td>
<td>3</td>
</tr>
<tr>
<td>SW</td>
<td>40000C</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>2CE</td>
<td>NS</td>
<td>2</td>
</tr>
</tbody>
</table>

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.”

\[
\begin{array}{c|c|c|c|c|c|c}
Rs & 31 & 30 & 29 & 28 & \ldots & 3 & 2 & 1 & 0 \\
\hline
Rt & \text{\rightarrow} & \text{\rightarrow} & \text{\rightarrow} & \text{\rightarrow} & \ldots & \text{\rightarrow} & \text{\rightarrow} & \text{\rightarrow} & \downarrow \\
\end{array}
\]

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, 1 # Since R9 is not equal to R0, do not skip the next instruction
```

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:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>PC</th>
<th>R8</th>
<th>R9</th>
<th>R10</th>
<th>Mem. Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>00000</td>
<td>?</td>
<td>?</td>
<td></td>
<td>---------</td>
</tr>
<tr>
<td>SLT R9, R8, R0</td>
<td>00004</td>
<td>NS</td>
<td>1</td>
<td>NS</td>
<td>1: IF</td>
</tr>
<tr>
<td>SLL R10, R8, 1</td>
<td>00008</td>
<td>NS</td>
<td>NS</td>
<td>E1E1E1E0</td>
<td>1: IF</td>
</tr>
<tr>
<td>BEQ R9, R0, 1</td>
<td>0000C</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1: IF</td>
</tr>
<tr>
<td>ORI R10, R10, 1</td>
<td>00010</td>
<td>NS</td>
<td>NS</td>
<td>E1E1E1E1</td>
<td>1: IF</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>PC</th>
<th>R8</th>
<th>R9</th>
<th>R10</th>
<th>Mem. Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>400000</td>
<td>4</td>
<td>?</td>
<td>3</td>
<td>----</td>
</tr>
<tr>
<td>ADD</td>
<td>400004</td>
<td>NS</td>
<td>4</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>RL1</td>
<td>400008</td>
<td>NS</td>
<td>8</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>ADDI</td>
<td>40000C</td>
<td>NS</td>
<td>NS</td>
<td>2</td>
<td>1 : IF</td>
</tr>
<tr>
<td>BNE</td>
<td>400010</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>RL1</td>
<td>400008</td>
<td>NS</td>
<td>(16)10</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
</tbody>
</table>

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
```

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

```
BLP Rs, Offset  
1) Rs ← Rs - 4
2) If Rs ≠ FFFFFFFC then PC ← 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:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>400000</td>
<td>10000004</td>
<td>?</td>
<td>?</td>
<td>0</td>
<td>1000A000</td>
<td>1</td>
<td>NS</td>
<td>NS</td>
<td>2</td>
</tr>
<tr>
<td>LW R9, 0(R8)</td>
<td>400004</td>
<td>NS</td>
<td>2</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>2 : IF &amp; DR</td>
</tr>
<tr>
<td>SLL R9, R9, 1</td>
<td>400008</td>
<td>NS</td>
<td>4</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>SLT R10, R9, R0</td>
<td>40000C</td>
<td>NS</td>
<td>NS</td>
<td>0</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>BEQ R10, R0, 1</td>
<td>400014</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>SW R9, 0(R8)</td>
<td>400018</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>4</td>
<td>NS</td>
<td>2 : IF &amp; DW</td>
</tr>
<tr>
<td>BLP R8, (-7)10</td>
<td>400000</td>
<td>10000000</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>LW R9, 0(R8)</td>
<td>400004</td>
<td>NS</td>
<td>1</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>2 : IF &amp; DR</td>
</tr>
<tr>
<td>SLL R9, R9, 1</td>
<td>400008</td>
<td>NS</td>
<td>2</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>SLT R10, R9, R0</td>
<td>40000C</td>
<td>NS</td>
<td>NS</td>
<td>0</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>BEQ R10, R0, 1</td>
<td>400014</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>SW R9, 0(R8)</td>
<td>400018</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>2</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>2 : IF &amp; DW</td>
</tr>
<tr>
<td>BLP R8, (-7)10</td>
<td>40001C</td>
<td>FFFFFFC</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>SW R11, 0(R12)</td>
<td>400020</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>0</td>
<td>2</td>
<td>NS</td>
<td>2 : IF &amp; DW</td>
</tr>
</tbody>
</table>

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 FFFFFFC 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)
```

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:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>400A00</td>
<td>?</td>
<td>?</td>
<td>10000000</td>
<td>10008000</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>6</td>
<td>2</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>LW</td>
<td>R12, 0(R10)</td>
<td>400A04</td>
<td>NS</td>
<td>1</td>
<td>NS</td>
<td>NS</td>
<td>6</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>2 : IF &amp; DR</td>
</tr>
<tr>
<td>LW</td>
<td>R13, 0(R11)</td>
<td>400A08</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>2</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>2 : IF &amp; DR</td>
</tr>
<tr>
<td>MULT</td>
<td>R12, R13</td>
<td>400A0C</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>0</td>
<td>C</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>MFHI</td>
<td>R8</td>
<td>400A10</td>
<td>0</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
<tr>
<td>MFLO</td>
<td>R9</td>
<td>400A14</td>
<td>NS</td>
<td>C</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>NS</td>
<td>1 : IF</td>
</tr>
</tbody>
</table>
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 \((10000...0)_2 = (-2^{31})_{10}\)

Represent this smallest negative integer number precisely in the IEEE-754 single-precision format. Note that the largest negative integer representable by EMY is \((11111..1)_2 = (-1)_{10}\). 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: \(1\ 10011110\ 00000000000000000000000\)

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
-----  -----
10002000  0100  0000  0110  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.

\[
\begin{array}{cccccc}
\text{Opcode} & \text{Rs} & \text{Rt} & \text{Displacement} & \text{Address}
\end{array}
\]

\[
\begin{array}{cccccc}
1100 & 0101 & 0001 & 0011 & 0000 & 0000 & 0000 & 0000 & 0000
\end{array}
\]

\[
400508 \rightarrow \text{LWC1 F3, 0(R8)} \rightarrow \text{F3} \leftarrow \text{M[R8 + 0]}^+[1]
\]

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:

\[
\begin{array}{c}
\text{Sign} \\
\text{Mantissa}
\end{array}
\]

\[
\begin{array}{c}
0 \\
00000000000000000000000000000000
\end{array}
\]

\[
\begin{array}{c}
\text{BE} \\
11000000000000000000000000000000
\end{array}
\]

\[
\begin{array}{c}
\text{Mantissa} \\
11000000000000000000000000000000
\end{array}
\]

\[
\begin{array}{c}
\text{BE} = 2^7 = 128
\end{array}
\]

\[
\begin{array}{c}
0 < \text{BE} < 255 \Rightarrow \text{the number is normalized}
\end{array}
\]

\[
\begin{array}{c}
e = \text{BE} - 127 = 128 - 127 = 1
\end{array}
\]

\[
\begin{array}{c}
\text{Mantissa} = 2^{-1} + 2^{-2} = 1/2 + 1/4 = 0.5 + 0.25 = 0.75
\end{array}
\]

\[
(0 \ 10000000 \ 11000000000000000000000000000000)_{\text{IEEE-754}} = (1.75 \times 2^1)_{10}
\]

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:

\[
\begin{array}{c}
\text{F8} = 11000001 \ 10000000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \\
\text{F10} = 01000001 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000
\end{array}
\]

After the instruction is executed, the value of F6 is:

\[
\text{F6} = \text{11000000000000000000000000000000}
\]

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:

\[
\text{F8} = 11000001 \ 10000000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000
\]

The leftmost bit is the sign bit = 1 => The sign of the number is **negative**.
The next eight bits indicate the biased exponent, \( BE = 1000 \ 0011 = 2^7 + 2^1 + 2^0 = 128 + 2 + 1 = 131 \)

➤ 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 : \( F8 = -1.0 \times 2^4 \)

\( F8 = 0100 \ 0001 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \)

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 : \( F10 = 1.0 \times 2^3 \)

\( F10 = 1100 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \)

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 : \( F6 = -1.0 \times 2^1 \)

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

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

\[ 40020A \ \text{DIV} \ \text{S} \ \text{F6, F8, F10} \]

Q15) 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 : \( 3.5 \times 2^{-7} \)

b) The EMY computer executes the following instruction : \( \text{DIV} \ \text{R9, R12} \)

Prior to the execution, R9 has \( \text{FFFFFDF} \) and R12 has 7. What is the effect of this instruction ?

A15) a) \( 3.5 \times 2^{-7} = (\ ?)_{2} \Rightarrow \)
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:

<table>
<thead>
<tr>
<th>The integer part</th>
<th>The fraction part</th>
<th>3.5 * 2^{-7} = 11.1 * 2^{-7} is NOT normalized.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 / 2 = 1 &amp; 1</td>
<td>0.5 * 2 = 1.0</td>
<td>= 1.11 * 2^{-6} after normalization.</td>
</tr>
<tr>
<td>1 / 2 = 0 &amp; 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>=&gt; (3)_{10} = (11)_2 Unsigned</td>
<td>=&gt; (0.5)_{10} = (0.1)_2 Unsigned</td>
<td></td>
</tr>
</tbody>
</table>

The exponent is calculated as follows:

\[ e = -6 \Rightarrow BE = e + 127 = -6 + 127 = 121 \]

Biased Exponent (BE) :
\[ (121)_{10} = (1111001)_2 \text{ Unsigned} \]

By using 8 bits :
\[ 01111001 = BE \]

The FP number in the IEEE-754 format :
\[ 0 01111001 1100...0 \]

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 = FFFFFDF  R12 = 7

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}

Hi register gets the remainder. It has -5 ⇒ (+5)_{10} = (00...0101)_{2} ⇒ (11...1011)_{2} => FFFFFFFFB

Lo register gets the quotient. It has -4 ⇒ (+4)_{10} = (00...0100)_{2} ⇒ (11...11100)_{2} ⇒ FFFFFFFC

Q16) Convert the following single-precision IEEE-754-format FP number to a decimal number as shown in class such that all calculations are manual :

\[ (0001 0001 0100 0000 0000 0000 0000 0000)_{\text{IEEE-754}} = (?)_{10} \]
A16) We will show the single-precision IEEE-754 format in decimal:

\[(0001 0001 0100 0000 0000 0000 0000 0000)_{10} = (\)?)_{Decimal}\]

\[0 \quad 001 \quad 0001 \quad 0 \quad 100 \quad 0000 \quad 0000 \quad 0000 \quad 0000 \quad 0000 \]

\[\text{Sign} \quad \text{BE} \quad \text{Mantissa}\]

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

\[\text{BE} \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 7 \quad 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1 \quad 0 \quad \]

\[\text{0 < BE < 255 => the number is normalized}\]

\[e = \text{BE} - 127 = 34 - 127 = -93\]

\[\text{Mantissa} \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad\]

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

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

Q17) Convert the following decimal number to a single-precision IEEE-754-format FP number as shown in class such that all calculations are manual: \((29)_{10} = (?)_{\text{IEEE-754}}\)

A17) \((29)_{10} = (?)_{\text{IEEE-754}}\)

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:

<table>
<thead>
<tr>
<th>The integer part :</th>
</tr>
</thead>
<tbody>
<tr>
<td>29 / 2 = 14 &amp; 1</td>
</tr>
<tr>
<td>14 / 2 = 7 &amp; 0</td>
</tr>
<tr>
<td>7 / 2 = 3 &amp; 1</td>
</tr>
<tr>
<td>3 / 2 = 1 &amp; 1</td>
</tr>
<tr>
<td>1 &amp; 2 = 0 &amp; 1 msb</td>
</tr>
<tr>
<td>\Rightarrow (29)<em>{10} = (11101)</em>{2} Unsigned</td>
</tr>
</tbody>
</table>

\[29 = 11101 \cdot 2^0 \quad \text{which is NOT normalized}\]

\[= 1.1101 \cdot 2^4 \quad \text{after normalization}\]

<table>
<thead>
<tr>
<th>BE :</th>
</tr>
</thead>
<tbody>
<tr>
<td>131 / 2 = 65 &amp; 1</td>
</tr>
<tr>
<td>65 / 2 = 32 &amp; 1</td>
</tr>
<tr>
<td>32 / 2 = 16 &amp; 0</td>
</tr>
<tr>
<td>16 / 2 = 8 &amp; 0</td>
</tr>
<tr>
<td>8 / 2 = 4 &amp; 0</td>
</tr>
<tr>
<td>4 / 2 = 2 &amp; 0</td>
</tr>
<tr>
<td>2 / 2 = 1 &amp; 0</td>
</tr>
<tr>
<td>1 &amp; 2 = 0 &amp; 1 msb</td>
</tr>
<tr>
<td>\Rightarrow (10000011)_{2} Unsigned</td>
</tr>
</tbody>
</table>

| Fraction is 1101000...000 |
| e = 4 |
| \Rightarrow BE = e + 127 = 4 + 127 = 131 |

The FP number in the IEEE-754 format:

\[\begin{array}{c}
0 \\
\text{Sign} \\
10000011 \\
\text{BE} \\
1101000...000 \\
\text{M (23 bits)}
\end{array}\]