1. (40 pts) Consider the following piece of SPIM code. The text segment starts at 0x00400000 and the data segment starts at 0x10010000. Assume all registers and memory locations are initialized to 0x00000000. The code is assembled in the order shown (i.e., no independent compilation of functions)

```
.data
label: .word 8,16,32,64
.byte 64, 32
.text
.globl main
main:
    la $4, label
    li $5, 16
    jal func
    li $v0 10
    syscall

func:
    add $2, $4, $0
    add $3, $5, $0
    add $9, $0, $0

loop:
    lw $22, 0($2)
    add $9, $9, $22
    addi $2, $2, 4
    slt $8, $2, $3
    bne $8, $0, loop
    add $2, $9, $0
    jr $31
```

a. (5 pts) How many bytes does the entire program occupy?
   Instructions = 60 bytes (15 instructions)
   Data = 18 bytes

   Add 4 bytes if you assume that the la instruction is translated into two instructions. In this case it is translated into 1 instruction since the lower address bits of the address of label are 0x0000.

b. (5 pts) During assembly, what are the values of the following labels?
   i. func  ____0x00400014____
   ii. label  ____0x10010000____

   Add 0x4 to the first address if you assume that the la instruction is translated into two instructions.
c. (10 pts) What would be the contents of the following byte locations assuming big endian and little endian format

<table>
<thead>
<tr>
<th>Byte Address</th>
<th>Byte Value (Little Endian)</th>
<th>Byte Value (Big Endian)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1001000b</td>
<td>0x00</td>
<td>0x20</td>
</tr>
<tr>
<td>0x10010004</td>
<td>0x10</td>
<td>0x00</td>
</tr>
</tbody>
</table>

d. (10 pts) The following is the binary representation of assembled SPIM code. Disassemble the encoding producing the original SPIM instructions. Use the opcode map at the end of this exam.

<table>
<thead>
<tr>
<th>Assembled Binary</th>
<th>MIPS Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xac220200</td>
<td>sw $2, 512($1)</td>
</tr>
<tr>
<td>0x0c10000e</td>
<td>jal 0x004000038</td>
</tr>
</tbody>
</table>

e. (10 pts) What is the value of 0x2a3c0000 assuming that it is a

a) Floating point number

1.01111 \times 2^{-43}

b) Character string

“* < null null”
2. **(10 pts)** Consider the voltage-frequency table below for a commercial processor that shows the full range of power states—these are the voltage-frequency pairs at which the processor can operate. Ignore the first two columns. Consider the execution of a single application. Compute an estimate of the ratio of the maximum to minimum dynamic power dissipation that can occur when this application is executed on this processor. You can leave your answer in expression form.

<table>
<thead>
<tr>
<th>HW Only (Boost)</th>
<th>CPU P-state</th>
<th>Voltage (V)</th>
<th>Freq (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pb0</td>
<td>1</td>
<td>2400</td>
<td></td>
</tr>
<tr>
<td>Pb1</td>
<td>0.875</td>
<td>1800</td>
<td></td>
</tr>
<tr>
<td>SW-Visible</td>
<td>P0</td>
<td>0.825</td>
<td>1600</td>
</tr>
<tr>
<td></td>
<td>P1</td>
<td>0.812</td>
<td>1400</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>0.787</td>
<td>1300</td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>0.762</td>
<td>1100</td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>0.75</td>
<td>900</td>
</tr>
</tbody>
</table>

The expression for dynamic power dissipation is

\[ P_{\text{dynamic}} = \alpha \left( \frac{C_L}{2} \right) \cdot V_{dd} \cdot V_{dd} \cdot F \]

Now instantiating values from the table we get the following ratio and given we are executing the same application we can obtain the following.

\[
\frac{1.0^2 \times 2.4}{0.75^2 \times 0.9}
\]
3. **(10 pts)** Consider the following sequence of procedure calls. The address of each individual jal instruction is shown next to the instruction.

Main Program

- 0x00400020 jal funcA
- 0x00400040 jal funcB
- 0x00400080 jal funcC

The address of each individual jal instruction is shown next to the instruction.

a. **(5 pts)** Which of the following, if any, are valid contents of $31 when procedure C is executing? You must explain why to receive credit.

i. 0x00404024
   
   Invalid: This is not 0x04004024 which would be valid as the instruction following the jal funcC in funcA.

ii. 0x00408024
   
   Not Valid: this is the instruction following the jump to function A.

iii. 0x00400080
   
   Not Valid: This is the address of the jal funcC call. The return address should be pointing to the following instruction.
b. (5 pts) You are the developer for procedure C using the MIPS calling convention. What registers can you use in your code without having to save and restore them?

According to the MIPS calling convention you can use the following. If these are required by the caller, the caller is required to save and restore them.

$\texttt{t0-}\texttt{t9}$

$\texttt{a0-}\texttt{a3}$

$\texttt{v0-}\texttt{v1}$
4. **(15 pts)** We have a program, \( A \), that refers to a data item, \( L \) (this is a label), in an independently compiled code module \( B \). The instruction in \( A \) that refers to data item \( L \) is \( \text{lw} \ $t0, 0($gp) \) where the offset must be adjusted by the linker to generate correct address. The register \( $gp \) (MIPS global pointer) is initialized to 0x10018000. After programs \( A \) and \( B \) are placed in memory by the linker, the address of \( L \) is 0x10010020.

   a. **(5 pts)** What is the offset the static linker will use for the \( \text{lw} \) instruction.

      The offset must be negative since we must have
      \[
      0\times10018000 + \text{offset} = 0\times10010020
      \]
      \[
      \text{offset} = 0\times10010020 - 10018000
      \]

      0x8020: Note that this is a negative number. See example problem on page 127 for a detailed explanation.

   b. **(10 pts)** Describe how execution would take place via dynamic linking. To receive credit your answer should be clear on how code would be assembled and modified at runtime.

      You answer should follow Figure 2.22 or similar (i.e. the use of indirection through compiled stub code).
5. **(25 pts)** Consider the single cycle SPIM data path shown overleaf. All registers are initialized to their number, i.e., $12$ is initialized to 0x000000c.

   a. **(15 pts)** Fill in the table for the following instructions from Q1 for the first time through the loop

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Write Data Input To memory</th>
<th>AluSrc</th>
<th>MemToReg</th>
<th>RegWrite</th>
<th>Branch Address</th>
<th>AluOp</th>
</tr>
</thead>
<tbody>
<tr>
<td>bne $8, $0, loop</td>
<td>0x00000000</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0x00400020</td>
<td>01</td>
</tr>
<tr>
<td>add $9, $9, $22</td>
<td>0x00000008</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0x00404844</td>
<td>10</td>
</tr>
</tbody>
</table>

   b. **(10 pts)** On the figure, show the modifications, as necessary, to the data path required to implement the addi instruction. For credit you must clearly and legibly show any modifications to the data path. Describe below the changes required for control modifying the table as necessary.

<table>
<thead>
<tr>
<th>Control Signals</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RegDst</td>
<td>0</td>
</tr>
<tr>
<td>Branch</td>
<td>0</td>
</tr>
<tr>
<td>MemRead</td>
<td>X</td>
</tr>
<tr>
<td>MemToReg</td>
<td>0</td>
</tr>
<tr>
<td>ALUOp</td>
<td>00</td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>1</td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
</tr>
</tbody>
</table>

   addi $8, $9, 4
   No hardware additions are necessary. All of the necessary data transfer paths are available. The data path can be controlled to select the immediate operand as the second operand and setting ALUOp to 00 to force an addition operation. This also requires setting ALUSrc to select the immediate operand. The destination register must now be rt rather than rd (set RegDst). The controller must be updated to recognize the addi opcode and generate (e.g., via the PLA) the control signals above.