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.

```
.data
start: .word 0x21, 22,
str: .ascii “CmpE”
.align 3
.word 24, 0x77

.text
.globl main
main:
  li $t3, 4
  lui $t0, 0x1001
  lui $t1, 0x1002
move:
  lw $t5, 0($t0)
  sw $t5, 0($t1)
  addi $t0, $t0, 4
  addi $t1, $t1, 4
  addi $t3, $t3, -1
end:
  bne $t3, $zero, move
done:
```

a. 

b. (5 pts) How many total bytes of storage are taken up by this program. If you need to make any assumptions about program assembly, state them explicitly.

Number of Bytes = ___60___

The .align 3 moves the loading pointer to 0x10010010. Therefore the data segment takes 24 bytes (6 words). Each instruction can be encoded in one instruction and therefore the 9 instructions take 36 bytes.

c. (10 pts) Show the addresses and corresponding contents of the first four word locations in the data segment that are loaded by the above SPIM code.

<table>
<thead>
<tr>
<th>Word Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10010000</td>
<td>0x00000021</td>
</tr>
<tr>
<td>0x10010004</td>
<td>0x00000016</td>
</tr>
<tr>
<td>0x10010008</td>
<td>0x45706d43</td>
</tr>
<tr>
<td>0x1001000c</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
d. **(4 pts)** What would the contents of register $t0$ (in hexadecimal notation), after the execution of the following instruction, assuming Big Endian storage format?

\[
\text{lw } t0, \text{ str}(0) \quad \text{___0x436d7045____}
\]

The byte value at the word boundary is the most significant byte of the 32-bit word.

e. **(10 pts)** Show the hexadecimal encoding of the following instruction

i. \text{bne } $3, $zero, move \quad \text{___0x1460fff8 (offset from PC)}

ii. \text{lw } t5, 0(t0) \quad \text{___0x8d0d0000}

d. **(6 pts)** The following is the binary representation of a block of assembled SPIM code. Disassemble the program 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>0xafbe003c</td>
<td>\text{sw } $30, 60($29)</td>
</tr>
<tr>
<td>0x00082821</td>
<td>\text{addu } $5, $0, $8</td>
</tr>
</tbody>
</table>

e. **(5 pts)** Consider a program that references independently compiled procedures. What is the difference between static and dynamic linking? Be specific.

Static Linking: All program modules are linked at link/load time. All unresolved references are resolved and all non-relocatable instructions are patched. The size of the executable is large since all references must be resolved, i.e., all potential callers must be linked even of they will not be called during execution.

Dynamic Linking: Unresolved references are resolved at runtime. At compile/assembly time stub code is inserted that effects an indirect jump to system software that loads the requested module and patches the indirect jump to the entry address of the module. The next time this label is referenced, the indirect jump will directly take place to the named module.
2. **(10 pts)** Consider the 32-bit ALU design discussed in class and shown below. Now suppose that we wished to add hardware support for a new instruction - \textbf{bre $t0$, $t1$} – the \textit{bit reverse} instruction. The contents of register \$t0 is obtained by reversing the order of bits in register \$t1. For example, the 4-bit number 1101 will become 1011 and the number 1100 would become 0011. \textbf{Clearly} i) describe the necessary changes to the ALU hardware showing the changes to the \(i\)th ALU bit position by modifying the figure below, and ii) list the corresponding values of the ALU control signals for this instruction. Provide a brief explanation.

- Add an additional input to each multiplexor.
- The new input of the multiplexor at bit position \(i\) is connected to the \((31-i)\)th bit of the input corresponding to \$t1, e.g., input a.
- When ALU control is set to 100, the output will be the bit reversal of the input.
- The value of binvert does not matter.
- As for the implementation, one option is to use the I-format with the destination register being rt.
- There are other solutions.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Binvert</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>bre</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>
3. **(15 pts)** The following questions address aspects of power/energy consumption in combinational logic.

a. **(5 pts)** Write an expression for the transition probability of a $0 \rightarrow 1$ transition for a 2-input OR gate. Recall this is used in the context of understanding the dynamic power consumption of a gate. You must also clearly explain your expression or it must be clear how you obtained this expression.

This is given by (probability that the output is 0) x (probability that at least one of the inputs is 1). If $p_A (p_B)$ represents the probability that input $A(B)$ is 1, then we have the transition probability as the following.

$$(1 - p_A)(1 - p_B)(1 - (1 - p_A)(1 - p_B))$$

b. **(5 pts)** Why does the energy dissipation of a 32-bit integer ALU depend in the input values?

The values of the inputs determine their bit patterns. The bit patterns determine the number of signal transitions produced by the combinational logic that comprise the ALU, and consequently determines the dynamic energy dissipated for that set of inputs. For example, consider the patterns where the carry signal ripples from the LSB position to the MSB position. This is not the case for all bit patterns.

c. **(5 pts)** Distinguish between energy dissipation and power dissipation.

In our usage, (dynamic) energy is the capacity to switch states ($0 \rightarrow 1 \rightarrow 0$). Energy can be dissipated over a period of time and is measured in joules. Power dissipation is a rate of expenditure of energy and is denoted in energy dissipated per sec, i.e., joules/sec.
4. **(35 pts)** Consider the single cycle SPIM data path shown overleaf. All registers are initialized to their number, i.e., \$12\ is initialized to 0x0000000c

a. **(15 pts)** Fill in the values of the signals below for each of the instructions.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>WriteData Input to Memory</th>
<th>AluSrc</th>
<th>MemToReg</th>
<th>RegWrite</th>
<th>Instruction (28:26)</th>
<th>AluOp</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $9, 24($0)</td>
<td>0x00000009</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>011</td>
<td>00</td>
</tr>
<tr>
<td>add $12, $13, $14</td>
<td>0x0000000e</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>000</td>
<td>10</td>
</tr>
</tbody>
</table>

b. **(10 pts)** Complete the controller hardware below for the \textbf{RegDst} and \textbf{MemWrite} control signals

![Controller Hardware Diagram]

RegDst

MemWrite

```
Inputs
Op5
Op4
Op3
Op2
Op1
Op0
```

```
R-format
lw
sw
beq
```

c. **(10 pts)** Show the physical modifications required to datapath to implement the jump instruction. Show all of the hardware modifications that will appear in the datapath including any signals that may have to be produced by the controller (but you do not have to state how they are set).