1  **The SPIM Simulator** [5 points]

There isn’t a solution per se to this. The expectation is that you are familiar with SPIM/XSPIM, understand its strengths and limitations, and are using it to debug your labs and homeworks.

2  **Big versus Little Endian Addressing** [5 points]

1. 7d ea d3 21

2. 21 d3 ea 7d

3  **Instruction Set Architecture (ISA)** [25 points]

**Code size:**

Each instruction has an opcode and a set of operands

- The opcode is always 1 byte (8 bits).
- All register operands are 1 byte (8 bits).
- All memory addresses are 2 bytes (16 bits).
- All data operands are 4 bytes (32 bits).
- All instructions are an integral number of bytes in length.

**Memory Bandwidth:**

Memory bandwidth consumed = amount of code transferred (code size) + amount of data transferred

Amount of data transferred = number of data references * 4 bytes

We will call the amount of code transferred as I-bytes and the amount of data transferred as D-bytes.

(a), (b)

<table>
<thead>
<tr>
<th>Instruction Set Architecture</th>
<th>Opcode</th>
<th>Operands</th>
<th>I-bytes</th>
<th>D-bytes</th>
<th>Total Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero-address</td>
<td>PUSH</td>
<td>B</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PUSH</td>
<td>C</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADD</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>A</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PUSH</td>
<td>A</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PUSH</td>
<td>B</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADD</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>B</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PUSH</td>
<td>A</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PUSH</td>
<td>B</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADD</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP</td>
<td>D</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
<td>36</td>
<td>66</td>
</tr>
</tbody>
</table>
The three-address memory-memory machine is the most efficient as measured by code size - 21 bytes.

The three-address load-store machine is the most efficient as measured by total memory bandwidth consumption (amount of code transferred + amount of data transferred) - 52 bytes.

4 The ARM ISA [40 points]

4.1 Warmup: Computing a Fibonacci Number [15 points]

NOTE: More than one correct solution exists, this is just one potential solution.

```assembly
fib :
mov r1 , #0       // initialize the function
mov r2 , #1
add r3 , r1 , r2
branch :
cmp r0 , #0       // check if we are done
ble done
```
add r3, r1, r2 // fib(n) = fib(n-1)+fib(n-2)
mov r1, r2 // increment the indices
mov r2, r3
subs r0, r0, #1 // decrement the counter
b branch // loopback
done:
mov r0, r3
mov r15, r14 // return

4.2 ARM Assembly for REP MOVSB [25 points]
(a)  
subs r0, r3, #1, // check the condition
blt finish

copy:
ldrb r4, [r1] #1 // load 1 byte, move the source pointer to the next addr
strb r4, [r2] #1 // store (copy) 1 byte, move the destination pointer to the next addr
subs r3, r3, #1 // decrement the counter
bge copy

finish:
Following instructions

(b) The size of the ARM assembly code is 4 bytes × 6 = 24 bytes, as compared to 2 bytes for x86 REP MOVSB.

(c) The count (value in ECX) is 0xcafebeef = 3405692655. Therefore, loop body is executed (3405692655 * 4) = 13622770620 times. Total instructions executed = 13622770620 + 2 (instructions outside of the loop) = 13622770622.

(d) The count (value in ECX) is 0x00000000 = 0. In this case, the total instruction executed is 2 instructions (instructions outside the loop. Note that this can varies based on the assemble codes).
5 Data Flow Programs [15 points]

6 Performance Metrics [10 points]
- No, the lower frequency processor might have much higher IPC (instructions per cycle).
  More detail: A processor with a lower frequency might be able to execute multiple instructions per cycle while a processor with a higher frequency might only execute one instruction per cycle.
- No, because the former processor may execute many more instructions.
  More detail: The total number of instructions required to execute the full program could be different on different processors.

7 Performance Evaluation [15 points]
- ISA A: $\frac{6 \text{ instructions}}{\text{cycle}} \times 400,000,000 \text{ cycle/second} = 2400$ MIPS
- ISA B: $\frac{2 \text{ instructions}}{\text{cycle}} \times 800,000,000 \text{ cycle/second} = 1600$ MIPS
- Don’t know.
  The best compiled code for each processor may have a different number of instructions.
8 Fixed Length vs. Variable Length [15 points]

Code size:

We can use Huffman encoding to encode the opcode

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>numBits</th>
<th>Total Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB</td>
<td>0</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>ADD</td>
<td>10</td>
<td>11</td>
<td>55</td>
</tr>
<tr>
<td>BEQ</td>
<td>110</td>
<td>17 to 41 (Assuming immediate is 8 to 32 bits)</td>
<td>51 to 123</td>
</tr>
<tr>
<td>MULT</td>
<td>1110</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td></td>
<td>226 to 298</td>
</tr>
</tbody>
</table>

In a traditional MIPS, this will take 640 bits (20 instructions, 32 bits each) So, the saving is 64.7% to 46.6%.