1  Big versus Little Endian Addressing [5 points]

1.  

<table>
<thead>
<tr>
<th>3a</th>
<th>2b</th>
<th>fe</th>
<th>ca</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

2.  

<table>
<thead>
<tr>
<th>ca</th>
<th>fe</th>
<th>2b</th>
<th>3a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

2  The MIPS ISA [40 points]

2.1  Warmup: Computing a Fibonacci Number [15 points]

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

```mips
fib:
    addi $sp, $sp, -16 // allocate stack space
    sw $16, 0($sp)  // save r16
    add $16, $4, $0 // r16 for arg n
    sw $17, 4($sp)  // save r17
    add $17, $0, $0 // r17 for a, init to 0
    sw $18, 8($sp)  // save r18
    addi $18, $0, 1 // r18 for b, init to 1
    sw $31, 12($sp) // save return address
    add $2, $17, $18 // c = a + b
    branch:
        slti $3, $16, 2 // use r3 as temp
        bne $3, $0, done
        add $2, $17, $18 // c = a + b
        add $17, $18, $0 // a = b
        add $18, $2, $0 // b = c
        addi $16, $16, -1 // n = n - 1
        j branch
    done:
        lw $31, 12($sp) // restore r31
        lw $18, 8($sp)  // restore r18
        lw $17, 4($sp)  // restore r17
        lw $16, 0($sp)  // restore r16
        addi $sp, $sp, 16 // restore stack pointer
        jr $31           // return to caller
```

2.2  MIPS Assembly for REP MOVSB [25 points]

Assume: $1 = ECX, $2 = ESI, $3 = EDI

(a)  

```mips
bez $1, $0, AfterLoop // If counter is zero, skip
```
CopyLoop:
lf $4, 0($2) // Load 1 byte
sb $4, 0($3) // Store 1 byte
addiu $2, $2, 1 // Increase source pointer by 1 byte
addiu $3, $3, 1 // Increase destination pointer by 1 byte
addiu $1, $1, -1 // Decrement counter
bne $1, $0, CopyLoop // If not zero, repeat

AfterLoop:
Following instructions

(b) The size of the MIPS assembly code is 4 bytes × 7 = 28 bytes, as compared to 2 bytes for x86 REP MOVSB.

(c) The count (value in ECX) is 0xfee1dead = 4276215469. Therefore, loop body is executed (4276215469 × 6) = 25657292814 times. Total instructions executed = 25657292814 + 1 (beq instruction outside of the loop) = 25657292815.

(d) The count (value in ECX) is 0x00000000 = 0. Therefore, loop body is executed 0 times. Total instructions executed = 1 (beq instruction outside of the loop).

3 Data Flow Programs [15 points]

4 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.

5 Performance Evaluation [9 points]

• ISA A: \( \frac{10 \text{ Instructions}}{\text{cycle}} \times 500,000,000 \text{ cycle/second} = 5000 \text{ MIPS} \)

• ISA B: \( \frac{2 \text{ Instructions}}{\text{cycle}} \times 600,000,000 \text{ cycle/second} = 1200 \text{ MIPS} \)

• Don’t know.
  The best compiled code for each processor may have a different number of instructions.

6 LC-3b Microcode [80 points]
7 Microarchitecture vs. ISA [15 points]

(a) The ISA level is the interface a machine exposes to the software. The microarchitecture is the actual underlying implementation of the machine. Therefore, the microarchitecture and changes to the microarchitecture are transparent to the compiler/programmer (except in terms of performance), while changes to the ISA affect the compiler/programmer.

The compiler does not need to know about the microarchitecture of the machine in order to compile the program correctly.

(b) (i) ISA
(ii) Microarchitecture
(iii) ISA
(iv) ISA
(v) Microarchitecture
(vi) ISA
(vii) Microarchitecture
(viii) Microarchitecture

8 Single-Cycle Processor Datapath [30 points]

9 Pipelining [30 points]

Given the following code:

MUL R3, R1, R2
ADD R5, R4, R3
ADD R6, R4, R1
MUL R7, R8, R9
ADD R4, R3, R7
MUL R10, R5, R6

Note: Each instruction is specified with the destination register first. Calculate the number of cycles it takes to execute the given code on the following models:

Note: For all machine models, use the basic instruction cycle as follows:

- Fetch (one clock cycle)
- Decode (one clock cycle)
- Execute (MUL takes 6, ADD takes 4 clock cycles). The multiplier and the adder are not pipelined.
- Write-back (one clock cycle)

Do not forget to list any assumptions you make about the pipeline structure (e.g., how is data forwarding done between pipeline stages).

(a) A non-pipelined machine

9 + 7 + 7 + 9 + 7 + 9 = 48 cycles

(b) A pipelined machine with scoreboarding and five adders and five multipliers without data forwarding

| Cycles | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
|--------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| MUL R3, R1, R2 | F | D | E | E | E | E | E | W |
| ADD R5, R4, R3 | F | D | E | E | E | E | E | E | W |
| ADD R6, R4, R1 | F | D | E | E | E | E | E | E | W |
| MUL R7, R8, R9 | F | D | E | E | E | E | E | E | W |
| ADD R4, R3, R7 | F | D | E | E | E | E | E | E | W |
| MUL R10, R5, R6 | F | D | E | E | E | E | E | E | W |

28 cycles (or 26 cycles with internal register file data forwarding)

(c) A pipelined machine with scoreboarding and five adders and five multipliers with data forwarding.

| Cycles | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|--------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| MUL R3, R1, R2 | F | D | E | E | E | E | E | E | W |
| ADD R5, R4, R3 | F | D | E | E | E | E | E | E | W |
| ADD R6, R4, R1 | F | D | E | E | E | E | E | E | W |
| MUL R7, R8, R9 | F | D | E | E | E | E | E | E | W |
| ADD R4, R3, R7 | F | D | E | E | E | E | E | E | W |
| MUL R10, R5, R6 | F | D | E | E | E | E | E | E | W |

24 cycles

(d) A pipelined machine with scoreboarding and one adder and one multiplier without data forwarding

<table>
<thead>
<tr>
<th>Cycles</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL R3, R1, R2</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD R5, R4, R3</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD R6, R4, R1</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL R7, R8, R9</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5/9
ADD R4, R3, R7  
MUL R10, R5, R6

31 cycles (or 29 cycles with internal register file data forwarding)

(c) A pipelined machine with scoreboarding and one adder and one multiplier with data forwarding

Cycles  
MUL R3, R1, R2  
ADD R5, R4, R3  
ADD R6, R4, R1  
MUL R7, R8, R9  
ADD R4, R3, R7  
MUL R10, R5, R6

27 cycles

10 Hardware vs Software Interlocking [30 points]

(a) Calculate the number of cycles it takes to execute each of these two code segments on machines I and II.

Solution:

Code Segment A:

Machine I:
The machine stalls for three cycles after each instruction, as each instruction is dependent on the previous one.

F D E E E W  
F D E E E W  
F D E E E W  
F D E E E W

18 cycles

Machine II:
The compiler places nops between every two instructions as they are dependent and it can’t find other independent instructions.

ADD R5 <- R6, R7  
NOP  
NOP  
ADD R3 <- R5, R4  
NOP  
NOP  
ADD R6 <- R3, R8  
NOP  
NOP  
ADD R9 <- R6, R3
The execution time line of this modified code segment is as shown below. N indicates a NOP in a pipeline stage.

\[
\begin{array}{ccccccc}
F & D & E & E & E & W \\
N & N & N & N & N & N \\
N & N & N & N & N & N \\
N & N & N & N & N & N \\
F & D & E & E & E & W \\
N & N & N & N & N & N \\
N & N & N & N & N & N \\
N & N & N & N & N & N \\
F & D & E & E & E & W \\
N & N & N & N & N & N \\
N & N & N & N & N & N \\
N & N & N & N & N & N \\
F & D & E & E & E & W
\end{array}
\]

18 cycles

Code Segment B:

Machine I:

The execution timeline is as below:

\[
\begin{array}{ccccccc}
F & D & E & E & E & W \\
F & D & E & E & E & W \\
F & D & E & E & E & W \\
F & D & E & E & E & W \\
F & D & E & E & E & W \\
E & E & E & W \\
D & E & E & E & W
\end{array}
\]

13 cycles

Machine II:

For code segment B, the compiler can find independent instructions to place between dependent instructions. The compiler reorders instructions such that each pair of dependent instructions are separated by two independent instructions.

\[
\begin{align*}
\text{ADD} & \quad R4 \leftarrow R5, R6 \\
\text{ADD} & \quad R8 \leftarrow R9, R10 \\
\text{ADD} & \quad R3 \leftarrow R1, R2 \\
\text{ADD} & \quad R7 \leftarrow R1, R4 \\
\text{ADD} & \quad R12 \leftarrow R8, R2
\end{align*}
\]

The execution timeline for this reordered code is as below:

\[
\begin{array}{ccccccc}
F & D & E & E & E & W \\
F & D & E & E & E & W \\
F & D & E & E & E & W
\end{array}
\]

7/9
This is assuming that machine I does not implement any instruction reordering. Instructions could be reordered in hardware in machine I too producing similar results as machine II. However, this reordering logic could increase the critical path.

(b) Calculate the machine code size of each of these two code segments on machines I and II, assuming a fixed-length ISA, where each instruction is encoded as 4 bytes. **Solution:**

Code Segment A:
- Machine I - 16 bytes
- Machine II - 52 bytes (because of the additional NOPs)

Code Segment B:
- Machine I - 20 bytes
- Machine II - 20 bytes (no additional NOPs as compiler was able to find independent instructions)

(c) Which machine takes a smaller number of cycles to execute each code segment A and B?

**Solution:**
- Code Segment A: Both machines take the same number of cycles - 18 cycles
- Code Segment B: Machine II takes a smaller number of cycles

(d) Does the machine that takes a smaller number of cycles for code segment A also take a smaller number of cycles than the other machine for code segment B? Why or why not?

**Solution:**
- For code segment A, both machines take the same number of cycles as every instruction is dependent on the previous instruction. Therefore, the compiler cannot find independent instructions to place between dependent ones.

- For code segment B, machine II takes a smaller number of cycles as the compiler can find independent instructions to place between dependent instructions.

(e) Would you say that the machine that provides a smaller number of cycles as compared to the other machine has higher performance (taking into account all components of the Iron Law of Performance)?

**Solution:**
- Which machine that takes a smaller number of cycles depends on the code segment, as seen in the previous part of this question. For code segment A both machines take the same number of clock cycles. However, machine I’s cycle time could be longer due to interlocking hardware. Therefore, it could potentially incur a larger execution time.
- For code segment B, machine II has higher performance than machine I, as machine II takes a smaller number of cycles to execute and also has simpler hardware than machine I (could potentially have a smaller cycle time).

(f) Which machine incurs lower code size for each code segment A and B?
Solution:
For code segment A, machine I has lower code size than machine II.
For code segment B, both machines have the same code size.

(g) Does the same machine incur lower code sizes for both code segments A and B? Why or why not?

Solution:
No.
For code segment A, machine I has lower code size as the compiler inserts NOPs in machine II.
For code segment B, both machines incur the same code size as the compiler does not need to add additional NOPs.