This exam is open book and open notes. You have 50 minutes. Credit for problems requiring calculation will be given only if you show your work.

1. Performance (Part 1) 10 Points

Measurements on a processor implementation have shown that data transfer, immediate, and branch instructions (e.g., lw, addi, lui, sw, beq, etc.) require more clock cycles to execute than other instructions. The table below shows the number of clock cycles required for each instruction type:

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Transfer / Immediate / Branch</td>
<td>2</td>
</tr>
<tr>
<td>All Other</td>
<td>1</td>
</tr>
</tbody>
</table>

When we run benchmark program P it executes 10 Million instructions. Measurements indicate that 30% of these instructions are data transfer, immediate, and branch instructions, while 70% are all other types of instructions.

(a) How many clock cycles will it take for program P to execute?

\[
\text{Clock Cycles} = 0.7 \times 10^6 \text{ instr} \times 1 \text{ cycle/instr} \\
+ 0.3 \times 10^6 \text{ instr} \times 2 \text{ cycle/instr} \\
= 13 \times 10^6 \text{ clock cycles}
\]

(b) For a 500MHz clock, what is the execution time of program P?

\[
\text{Exec. Time} = 13 \times 10^6 \text{ cycles} / 500 \times 10^6 \text{ cycles/sec} \\
= 0.026 \text{ sec}
\]

2. MIPS Instructions 10 Points

The Committee for Prevention of Complex Instruction Sets has discovered that the “load upper immediate” instruction:

\[\text{lui \ $t0, \ immediate}\]

can be replaced by one immediate instruction and one shift instruction. Write MIPS assembly code for the two instructions which can perform this task:

\[\text{addi \ $t0, \ $zero \ immediate} \]
\[\text{sll \ $t0, \ $t0, \ 16}\]
3. Performance (Part 2)  

Measurements performed by the Committee for Prevention of Complex Instruction Sets show that 6% of all of the instructions in program P are “load upper immediate” (lui) instructions. Assume that lui instructions in program P are eliminated and replaced with the combination in the last problem, creating a program P’.

(a) How many clock cycles will it take the modified program P’ to execute?

\[
\text{Clock Cycles} = 0.7 \times 10^6 \text{ instr} \times 1 \text{ cycle/instr} \\
+ 0.24 \times 10^6 \text{ instr} \times 2 \text{ cycle/instr} \\
+ 0.06 \times 10^6 \text{ instr} \times 3 \text{ cycle/instr} \\
= 13.6 \times 10^6 \text{ clock cycles}
\]

(b) For a 500MHz clock, what is the execution time of the modified program P’?

\[
\text{Exec. Time} = 13.6 \times 10^6 \text{ cycles} / 500 \times 10^6 \text{ cycles/sec} \\
= 0.0272 \text{ sec}
\]

(c) Using the method of performance comparison discussed in the book, how much faster (or slower) will the original program P be compared to P’?

\[\text{P is } 0.0272 / 0.026 = 1.046 \text{ times faster than P’}\]

4. MIPS Instructions  

Show the binary equivalent of the following MIPS instructions by filling in the space provided below (all registers are specified using their register number $0$-$31$). Also mark the boundaries between the fields (e.g., opcode, etc.) and show the names of the fields.

and $5, 6, 7$

```
Opcode (0)  rs (6)  rt (7)  rd (5)  shamt (0)  funct (36)
0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0
```

addi $8, $9, -3

```
Opcode (8)  rs (9)  rt (8)  immed (-3)
0 0 1 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Page 2 of 5
5. Instructions – Assembly Language 40 Points

The assembly language program shown below reads words from an array of unsigned 32-bit integers and computes a result that is placed in the $v0 return value register. This program includes at least one assembler “pseudoinstruction” which assembles into more than one MIPS instruction.

```assembly
main:    la   $t0, 0x40000      # t0 = base address of array
lw   $v0, 0($t0)       # initialize v0
loop:   lw   $t1, 0($t0)
        beq  $t1, $zero, done
        slt  $t2, $t1, $v0
        beq  $t2, $zero, endl
        add  $v0, $t1, $zero
endl:   addi $t0, $t0, 4
        j loop
done:    # end of program
```

The base address of the array is 0x40000 hexadecimal. The diagram below shows the contents of memory in and around the array:

<table>
<thead>
<tr>
<th>Memory Address (hexadecimal)</th>
<th>Memory Contents (decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0003FFFC</td>
<td>3</td>
</tr>
<tr>
<td>0x00040000</td>
<td>10</td>
</tr>
<tr>
<td>0x00040004</td>
<td>4</td>
</tr>
<tr>
<td>0x00040008</td>
<td>-1</td>
</tr>
<tr>
<td>0x0004000C</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

a) How many 32-bit memory words will this program occupy after it is assembled? (Be careful!)

2 instructions for `la` + 8 instructions = **10 words**

b) How many times will the loop in this program execute?

The program enters the loop 4 times, but exits the last iteration immediately after the “lw” at loop:
c) Fill in the values of the registers $t0$-$t2$ and $v0$ in decimal or hexadecimal after each iteration of the loop the space provided below. Cross out any unneeded space.

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>$t0$</th>
<th>0x40004</th>
<th>$t2$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t1$</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$v0$</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 2</th>
<th>$t0$</th>
<th>0x40008</th>
<th>$t2$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t1$</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$v0$</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 3</th>
<th>$t0$</th>
<th>0x4000C</th>
<th>$t2$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t1$</td>
<td>-1</td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>$v0$</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 4</th>
<th>$t0$</th>
<th>0x4000C</th>
<th>$t2$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t1$</td>
<td>0</td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>$v0$</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

d) What function is computed by this program and placed in register $v0$?

The **minimum** of the elements in the array up to the “0”.

5. Instruction Set Design  

As discussed in the book, the four principles of instruction set design are:

1. Simplicity favors regularity.
2. Smaller is faster.
3. Good design demands good compromises.
4. Make the common case fast.

A number of design decisions in the MIPS Architecture are listed below. Fill in the blank with the design principles which were applied in making each decision.

(a) The opcode field is in the same position in all instructions       1

(b) All instructions are exactly 32 bits long                      1

(c) Immediate instructions allow the use of small constants       4, (3)

(d) The lui instruction allows the use of larger constants         4, (3)

(e) The rs and rt fields are in the same position in R-type and I-type instructions 1

(f) A linkage register is used to store the return address for “leaf procedures” (instead of the stack) 4

(g) Only 32 general purpose registers are used for R-type instructions 2
6. Multiplication Hardware 10 Points

The two unsigned binary numbers shown below are to be multiplied using a standard “shift and add” sequential multiplier and a multiplier that uses Booth’s Algorithm:

\[
\begin{array}{c}
10011010 \\
\times \\
01110111
\end{array}
\]

Answer the following questions:

(a) How many bits will be needed to store the product of these two numbers?

16 bits (15 for this specific case, since the multiplier’s MSB is 0)

(b) How many additions will be performed by the standard shift-and-add multiplier?

The adder adds constantly, but only 6 sums are actually stored in the product register - the rest are ignored.

(c) How many additions will be performed by the Booth’s Algorithm multiplier?

2 (1 at the end of each run of 1’s)

(d) How many subtractions will be performed by the Booth’s Algorithm multiplier?

2 (1 at the beginning of each run of 1’s - note that the LSB must be counted as the beginning of a run of 1’s)