PROBLEM 1. [15 points] We would like to make the following modifications that could be made to the MIPS instruction set architecture:

- 64 registers
- twice as many different instructions as we learned in the class

You will investigate the impact of these changes on the instruction format of the MIPS architecture. Note that we continue to use 32-bit wide instructions.

1.1 How this would this affect the size of each of the bit fields in the R-type instructions? Label all the fields with their name and bit length and explain the consequence (in terms of instruction sets). Assume that the amount of instructions affects both op and funct fields.

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 bits</td>
<td>6 bits</td>
<td>6 bits</td>
<td>6 bits</td>
<td>0 bits</td>
<td>7 bits</td>
</tr>
</tbody>
</table>

Recall that sll $rd, $rt, shift_amnt and srl $rd, $rt, shift_amnt. Since the shift field (shamt) is decreased to 0 bits, the shift operations like sll or srl will be removed from the instruction sets.

1.2 How this would this affect the size of each of the bit fields in the I-type instructions? Label all the fields with their name and bit length and explain the consequence (in terms of instruction sets).

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate / address</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 bits</td>
<td>6 bits</td>
<td>6 bits</td>
<td>13 bits</td>
</tr>
</tbody>
</table>

The range of immediate or address will be within $0 \sim 2^{13}$ for unsigned numbers or $-2^{12} \sim 2^{12} - 1$ for 2’s-complement signed numbers. This also means half-word operations (like lb, sb) of I-type cannot be supported any more.

PROBLEM 2. [10 points] You are given a memory address in binary: 0000 0001 0000 1001 1000 0000 0010 00102.

2.1 For this entry above, what is the value of the number in decimal? Write down how to do the conversion.

\[
\begin{align*}
0000 & \quad 0001 \quad 0000 \quad 1001 \quad 1000 \quad 0000 \quad 0010 \quad 0010_2 \\
= & \quad 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^{10} + 1 \times 2^8 + 1 \times 2^5 + 1 \times 2^1 \\
= & \quad 1739984_{10} \quad [1]
\end{align*}
\]
2.2 What instruction does it represent? And write down the MIPS code.

Since the first 6 bits are 0000 00\(_2\), it can be add, sub or slt. The last 6 bits are 10 0010\(_2\) = 34\(_{10}\).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>01000</td>
<td>01000</td>
<td>10000</td>
<td>00000</td>
<td>10010</td>
</tr>
</tbody>
</table>

So, this MIPS instruction is sub: sub $s0, $t0, $t1 [2].

PROBLEM 3.  [25 points] Consider the following MIPS code:

```
add $t1, $0, $0
add $s2, $0, $0
Loop: lw $s1, 0($s0)
add $s2, $s2, $s1
addi $s0, $s0, 4
addi $t1, $t1, 1
slt $t2, $t1, 100
bne $t2, $0, Loop
```

3.1 Explain what each line of the above MIPS code does, and figure out what this code does.

```
add $t1, $0, $0  # $t1 = 0 [1]
add $s2, $0, $0  # $s2 = 0 [1]
Loop: lw $s1, 0($s0)  # $s1 = Mem($s0) [2]
add $s2, $s2, $s1  # $s2 += $s1 [1]
addi $s0, $s0, 4  # $s0 += 4 [1]
addi $t1, $t1, 1  # $t1 += 1 [1]
slt $t2, $t1, 100  # if $t1 < 100, $t2 = 1; otherwise, $t2 = 0 [2]
bne $t2, $0, Loop  # if $t2 != 0, go to Loop [1]
```

It computes the sum of an input array, where this array is size of 100.

3.2 Translate the above loop into C. Assume that the C-level integer \( i \) is held in register \( \$t1 \), \( \$s2 \) holds the C-level 32-bit integer called \( r \), and \( \$s0 \) holds the base address of the 32-bit integer \( a[] \).

```
[5] r = 0;
for (i = 0; i < 100; ++i) { r += a[i]; }
```

3.3 How many MIPS instructions in total are executed?

\[ 2 + 6 \times 100 = 602 \text{ instructions in total.} \]

3.4 Rewrite the above MIPS loop to reduce the total amount of MIPS instructions executed. How many MIPS instructions in total are reduced?

```
[6] One possible solution is to remove the calculation regarding to the iterator \( \$t1 \) in the loop.
```
addi $t1, $s0, 400  # $t1 = $s0+400
add $s2, $0, $0   # $s2 = 0

Loop: lw $s1, 0($s0)  # $s1 = Mem($s0)
add $s2, $s2, $s1  # $s2 += $s1
addi $s0, $s0, 4  # $s0 += 4
bne $t1, $s0, Loop  # if $s0 != $t1, go to Loop

Hence, the total instructions are reduced to $2 + 4 \times 100 = 402$ instructions. 200 instructions are reduced. [2]

**PROBLEM 4.** [30 points] Consider the following C code function to compute the product of a 32-bit integer array A[] with size of n. It uses a divide and conquer method:

```c
int prodArray (int A[], int n) {
    if (!n)
        return 1;
    else if (n == 1)
        return A[0];
    int mid = n / 2;
    int left = prodArray(A, mid);
    int right = prodArray(A + mid, n - mid);
    return left * right;
}
```

4.1 Translate the above C code to MIPS code using procedures.

```mips
prodArray:  # n == 0
    bne $a1, $zero, n1 [1]
    addi $v0, $zero, 1 [1]
    jr $ra

n1:  slti $t0, $a1, 2  # n == 1
    beq $t0, $zero, dnc
    lw $v0, 0($a0) [1]
    jr $ra

dnc:  # divide and conquer
    addi $sp, $sp, -16  # Adjust stack pointer [1]
    sw $a0, 0($sp)      # Save $a0 = &A[] [1]
    sw $a1, 4($sp)      # Save $a1 = n [1]
    sw $ra, 8($sp)      # Save $ra [1]
    srl $a1, $a1, 1     # n / 2 [1]
    jal prodArray [1]
    sw $v0, 12($sp)     # Save return result [1]
```
sll $t0, $a1, 2       [1]
add $a0, $a0, $t0    # Address of A[] [1]
lw $t0, 4($sp)       [1]
sub $a1, $t0, $a1    # n - n / 2 [1]
jal prodArray       [1]
lw $t0, 12($sp)      [1]
lw $ra, 8($sp)       [1]
lw $a1, 4($sp)       [1]
lw $a0, 0($sp)       [1]
addi $sp, $sp, 16    [1]
mul $v0, $t0, $v0
jr $ra                [1]

4.2 Write a complete MIPS code and run it in QtSpim to compute prodArray(16) with A[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 ,14, 15, 16}. Show the final values of registers you used.

Code runs [4] and computes correctly [5].

4.3 [Bonus: 10 points] Try to use a minimum number of MIPS instructions as possible. Use the same or less amount of MIPS instructions for the above C code, compared to the amount of instructions given by our solution.

PROBLEM 5. [20 points] Assume that X consists of 3 bits: x2 x1 x0 and that y is represented by 1 bit.

(a) y is true if and only if X contains only one 1.
(b) y is true if and only if X is larger than 4.

For each of the above statements, finish the following exercises.

5.1 Write the logic function of y with its truth table.

(a): $y = \overline{x2} \cdot \overline{x1} \cdot x0 + \overline{x2} \cdot x1 \cdot \overline{x0} + x2 \cdot \overline{x1} \cdot \overline{x0}$ [3]

(b): $y = x2 \cdot \overline{x1} \cdot x0 + x2 \cdot x1 \cdot \overline{x0} + x2 \cdot x1 \cdot x0$
$= x2 \cdot \overline{x1} \cdot x0 + x2 \cdot x1 = x2 \cdot (x1 + x0)$ [3]

Truth tables are shown in Table [1] [2] and Table [3] [2].

5.2 Using LOGISM, draw the gate diagram using AND, OR and NOT gates only.

See Figure [4] each [5]
Table 1: The true table: \( y \) is true if and only if \( X \) contains only one 1

\[
\begin{array}{ccc|c}
 x_0 & x_1 & x_2 & y \\
 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 1 \\
 0 & 1 & 0 & 0 \\
 0 & 1 & 1 & 0 \\
 1 & 0 & 0 & 1 \\
 1 & 0 & 1 & 1 \\
 1 & 1 & 0 & 0 \\
 1 & 1 & 1 & 1 \\
\end{array}
\]

Table 2: The true table: \( y \) is true if and only if \( X \) is larger than 4

\[
\begin{array}{ccc|c}
 x_0 & x_1 & x_2 & y \\
 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 \\
 0 & 1 & 0 & 0 \\
 0 & 1 & 1 & 0 \\
 1 & 0 & 0 & 0 \\
 1 & 0 & 1 & 1 \\
 1 & 1 & 0 & 0 \\
 1 & 1 & 1 & 1 \\
\end{array}
\]

Figure 1: Gates of (a) and (b)