1 Problem 1

Chapter 2: Exercise 2.4. Part (b) only (i.e., 2.4.1b-2.4.6b):

Parts 2.4.1-3 deal with translating from C to MIPS. Assume that the variables \( f \), \( g \), \( h \), \( i \), and \( j \) are assigned to registers \( \$s0 \), \( \$s1 \), \( \$s2 \), \( \$s3 \), and \( \$s4 \), respectively. Assume that the base address of the arrays \( A \) and \( B \) are in registers \( \$s6 \) and \( \$s7 \), respectively.

\[
f = g - A[B[4]];
\]

2.4.1

For the C statement above, what is the corresponding MIPS assembly code?

\[
\begin{align*}
\text{lw} & \quad \$s0, 16(\$s7) \quad // f = B[4] \\
\text{sll} & \quad \$s0, \$s0, 2 \quad // f = f \cdot 4 = B[4] \cdot 4 \\
\text{add} & \quad \$s0, \$s0, \$s6 \quad // f = &A[f] = &A[B[4]] \\
\text{lw} & \quad \$s0, \text{0}(\$s0) \quad // f = A[f] = A[B[4]] \\
\text{sub} & \quad \$s0, \$s1, \$s0 \quad // f = g - f = g - A[B[4]]
\end{align*}
\]

2.4.2

For the C statement above, how many MIPS assembly instructions are needed?

5 instructions

2.4.3

For the C statement above, how many different registers are needed?

4 registers
Parts 2.4.4-6 deal with translating from MIPS to C. Assume that the variables $f$, $g$, $h$, $i$, and $j$ are assigned to registers $s0$, $s1$, $s2$, $s3$, and $s4$, respectively. Assume that the base address of the arrays $A$ and $B$ are in registers $s6$ and $s7$, respectively.

```
lw $s0, 4($s6)
```

### 2.4.4
For the MIPS assembly instructions above, what is the corresponding C statement?

$$f = A[1]$$

### 2.4.5
For the MIPS assembly instructions above, rewrite the assembly code to minimize the number of MIPS instructions (if possible) needed to carry out the same function.

No minimization possible.

### 2.4.6
How many registers are needed to carry out the MIPS assembly as written above? If you could rewrite the code above, what is the minimal number of registers needed?

2 registers, cannot be minimized.
2 Problem 2

Pseudo-instructions: Show how to implement the following pseudo-instructions as they would be implemented in a real machine (e.g., using $at):

2.1
	nop # Do nothing (show three different ways)

There are many different ways, for example:

1. add $zero, $zero, $zero
2. sll $zero, $zero, $zero
3. or $zero, $zero, $zero

2.2

li $s0, <32-bit constant> # Load 32-bit immediate value into s0

This instruction can be implemented a couple of different ways:

lui $s0, <16 upper bits>
ori $s0, $s0, <16 lower bits>

or

lui $s0, <16 upper bits>
addi $s0, $s0, <16 lower bits>

2.3

div $s0, $s1, $s2 # Integer division: s0 = s1/s2

This instruction can be implemented simply using the div instruction.

div $s1, $s2
mflo $s0
3 Problem 3

**New instructions:** Implement register indirect conditional branches (beqr and bner) as pseudo-instructions. Give a proposal for adding them to the ISA (i.e., describe how they could be encoded in the I-Type, R-Type, or J-Type format, and propose an opcode for them (use Figure B.10.2)). Give a (brief) argument for or against their inclusion.

One way to implement this for beqr $s0, $s1, $s2:

```
bne $s0, $s1, skip  # if (s0==s1) skip branch
add $zero, $zero, $zero # nop
jr $s2               # branch to $s2
add $zero, $zero, $zero # nop
```

```
skip:
```

bner is the same, but use beq instead for the first line.

We can encode this instruction as an R-Type instruction. As a R-Type instruction, the opcode would be 0, and we can use any funct value that has not yet been assigned.

Arguments for:
- reduces code size
- implementation in HW could save a branch

Arguments against:
- increase processor complexity
- might slow down decode logic or other parts of the pipeline
- can still achieve same functionality with existing instructions
4 Problem 4

Fibonacci sequence: Write MIPS assembly to compute the Nth Fibonacci number. Assume N is passed to your function in register $a0. Your output should be in register $v0 at the end of your function. Submit your code and a screenshot of Spim that shows the registers with correct output value for N=10, i.e., Fib(10) = 55 = 0x37. Note: You must implement Fibonacci recursively. The purpose of this assignment is to learn how to manipulate the stack correctly in MIPS.

One implementation of the fib function:

```mips
# The corresponding C code might look like this:
#
# int fib(int N) {
#   if (N == 0) return 0;
#   if (N == 1) return 1;
#   return fib(N-1) + fib(N-2);
# }
# input: N in $s0
# output: fib(N) in $s0

.text
.globl __start
__start:
or $a0, $zero, $s0  # Call fib(N)
jal fib
or $zero, $zero, $zero  # nop in delay slot
or $s0, $zero, $v0  # Save the returned value of fib(N)

#pragma Exit
addiu $v0, $zero, 10  # Prepare to exit (system call 10)
syscall  # Exit

#pragma Fib takes a single argument N in $a0, and returns fib(N) in $v0
#pragma Uses registers as follows:
#pragma   s0 - saved N (preserved across calls)
#pragma   s1 - fib(N-1)
#pragma   s2 - fib(N-2)

fib:
#pragma Save return address
addi $sp, $sp, -4
sw $ra, 0($sp)

#pragma fib(0) case (return 0)
or $v0, $zero, $zero
beq $a0, $zero, end
or $zero, $zero, $zero  # nop in delay slot

#pragma fib(1) case (return 1)
ori $v0, $zero, 1
beq $a0, $v0, end
or $zero, $zero, $zero  # nop in delay slot
```

5
# Recursion

# Save s0
addi $sp, $sp, -8
sw $s0, 0($sp)
sw $s1, 4($sp)

# Recover N from a0
or $s0, $zero, $a0

# Get fib(N-1)
addi $a0, $s0, -1  # N-1
jal fib
or $zero, $zero, $zero  # nop in delay slot
or $s1, $v0, $zero  # Save fib(N-1)

# Get fib(N-2)
addi $a0, $s0, -2  # N-2
jal fib
or $zero, $zero, $zero  # nop in delay slot
or $s2, $v0, $zero  # Save fib(N-2)

# Return fib(N-1) + fib(N-2)
add $v0, $s1, $s2

# Restore s0, s1, s2
lw $s0, 0($sp)
lw $s1, 4($sp)
addi $sp, $sp, 8

del:
lw $ra, 0($sp)
addi $sp, $sp, 4  # Note: Fills load delay
jr $ra
or $zero, $zero, $zero  # nop in delay slot
5 Problem 5

Performance Processors: The table below describes the performance of two processors, the rAlpha and the c86, and two compilers on a common 'benchmark' program.

<table>
<thead>
<tr>
<th></th>
<th>GHz</th>
<th>Cost</th>
<th>Compiler A</th>
<th>Compiler B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Instructions</td>
<td>Average CPI</td>
</tr>
<tr>
<td>rAlpha</td>
<td>3.4</td>
<td>$100</td>
<td>7000</td>
<td>1.2</td>
</tr>
<tr>
<td>c86</td>
<td>2.6</td>
<td>$100</td>
<td>1500</td>
<td>2.2</td>
</tr>
</tbody>
</table>

5a.
Which is the best compiler-machine combination?

Using the execution time of the program as the performance metric, where

\[
\text{seconds} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{clock cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{clock cycle}}
\]

<table>
<thead>
<tr>
<th></th>
<th>Compiler A</th>
<th>Compiler B</th>
</tr>
</thead>
<tbody>
<tr>
<td>rAlpha</td>
<td>(\frac{7000 \times 1.2}{3.4 \times 10^9} = 2.47 \times 10^{-6}) seconds</td>
<td>(\frac{5000 \times 1.5}{3.4 \times 10^9} = 2.20 \times 10^{-6}) seconds</td>
</tr>
<tr>
<td>c86</td>
<td>(\frac{1500 \times 2.2}{2.6 \times 10^9} = 1.26 \times 10^{-6}) seconds</td>
<td>(\frac{1000 \times 4.0}{2.6 \times 10^9} = 1.53 \times 10^{-6}) seconds</td>
</tr>
</tbody>
</table>

5b.
Both hardware vendors release quad-core versions of the processor for 1.7x the cost of the single core version. The results on the two new systems:

<table>
<thead>
<tr>
<th></th>
<th>GHz</th>
<th>Cost</th>
<th>Compiler A</th>
<th>Compiler B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Instructions</td>
<td>Average CPI</td>
</tr>
<tr>
<td>rAlphaX</td>
<td>3.4</td>
<td>$170</td>
<td>2250/core</td>
<td>1.5</td>
</tr>
<tr>
<td>c86x4</td>
<td>2.6</td>
<td>$170</td>
<td>625/core</td>
<td>3.2</td>
</tr>
</tbody>
</table>

Assume the benchmarks used are perfectly parallelizable i.e. the cores process equal parts of the program independently and at the same time. Now which combination performs best?
Using the same performance metric as 5a.,

<table>
<thead>
<tr>
<th></th>
<th>Compiler A</th>
<th>Compiler B</th>
</tr>
</thead>
<tbody>
<tr>
<td>rAlpha</td>
<td>(\frac{2250 \times 1.5}{3.4 \times 10^9} = 9.93 \times 10^{-7} \text{ seconds per program})</td>
<td>(\frac{1750 \times 1.8}{3.4 \times 10^9} = 9.26 \times 10^{-7} \text{ seconds per program})</td>
</tr>
<tr>
<td>c86</td>
<td>(\frac{625 \times 3.2}{2.6 \times 10^9} = 7.69 \times 10^{-7} \text{ seconds per program})</td>
<td>(\frac{500 \times 5.0}{2.6 \times 10^9} = 9.61 \times 10^{-7} \text{ seconds per program})</td>
</tr>
</tbody>
</table>

5c.

Using the metric \((\text{Cycles/s}) \times \text{Cores}/(\text{Dollars}^2)\), which is the best machine? Is this metric useful?

\[
\text{rAlphaX: } \frac{(3.4 \times 10^9) \times 4}{170^2} = 4.71 \times 10^5 \frac{\text{cycles}\times\text{core}}{\text{seconds}\times\text{dollars}^2}
\]

\[
\text{c86x4: } \frac{(2.6 \times 10^9) \times 4}{170^2} = 3.60 \times 10^5 \frac{\text{cycles}\times\text{core}}{\text{seconds}\times\text{dollars}^2}
\]

Using this metric, rAlpha has the highest value. However, this metric is not very useful because 5b. shows that the c86x4 can run programs faster with the same number of cores and cost. It is not indicative of a good computer for its price.

5d.

The four processors have the following power ratings:

- rAlpha: 20 W
- c86: 35 W
- rAlphaX: 50 W
- c86x4: 80 W

If you wanted to show that the first processor had the best power efficiency for performance, what metric would you use? For this part, use values only from Compiler A.

Either \(\frac{1}{\text{latency} \times \text{power}}\) where bigger is better or \(\text{latency} \times \text{power}^2\) where smaller is better would be better metrics. If power is not valued more, the metric does not show rAlpha as the best processor.
6 Problem 6

Chapter 1: Exercise 1.5. Part (b) only (i.e., 1.5.1b-1.5.6b):

Consider two different implementations, P1 and P2, of the same instruction set. There are five classes of instructions (A, B, C, D, and E) in the instruction set. The clock rate and CPI of each class is given below.

<table>
<thead>
<tr>
<th>Machine</th>
<th>Clock rate</th>
<th>CPI A</th>
<th>CPI B</th>
<th>CPI C</th>
<th>CPI D</th>
<th>CPI E</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1.0 GHz</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>1.5 GHz</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

1.5.1

Assume that peak performance is defined as the fastest rate that a computer can execute any instruction sequence. What are the peak performances of P1 and P2 expressed in instructions per second?

\[
\text{P1 (using CPI A): } \frac{1.0 \times 10^9 \text{ cycles}}{1 \text{ second}} \times \frac{1 \text{ instruction}}{1 \text{ cycle}} = 1 \times 10^9 \text{ inst/s}
\]

\[
\text{P2 (using CPI A): } \frac{1.5 \times 10^9 \text{ cycles}}{1 \text{ second}} \times \frac{1 \text{ instruction}}{1 \text{ cycle}} = 1.5 \times 10^9 \text{ inst/s}
\]

1.5.2

If the number of instructions executed in a certain program is divided equally among the classes of instructions except for class A, which occurs twice as often as each of the others: Which computer is faster? How much faster is it?

\[
\text{P1 cycles: } 2 \times 1 + 1 + 2 + 3 + 2 = 10
\]

\[
\text{Execution time on P1: } \frac{10}{1} = 10 \text{ ns}
\]

\[
\text{P2 cycles: } 2 \times 1 + 2 + 3 + 4 + 3 = 14
\]

\[
\text{Execution time on P2: } \frac{14}{1.5} = 9.33 \text{ ns}
\]

\[
\frac{\text{P2 performance}}{\text{P1 performance}} = \frac{\text{P1 execution time}}{\text{P2 execution time}} = \frac{10}{9.33} = 1.072
\]

P2 is 1.072 times faster than P1.
1.5.3

If the number of instructions executed in a certain program is divided equally among the classes of instructions except for class E, which occurs twice as often as each of the others: Which computer is faster? How much faster is it?

\[
P1 \text{ cycles: } 1 + 1 + 2 + 3 + 2 \times 2 = 11
\]

Execution time on P1: \(\frac{11}{1} = 11\) ns

\[
P2 \text{ cycles: } 1 + 2 + 3 + 4 + 2 \times 3 = 16
\]

Execution time on P2: \(\frac{16}{1.5} = 10.67\) ns

\[
\frac{\text{P2 performance}}{\text{P1 performance}} = \frac{\text{P1 execution time}}{\text{P2 execution time}} = \frac{11}{10.67} = 1.031
\]

P2 is 1.031 faster than P1.

The table below shows instruction-type breakdown for different programs. Using this data, you will be exploring the performance tradeoffs with different changes made to a MIPS processor.

<table>
<thead>
<tr>
<th>Program</th>
<th>Compute</th>
<th>Load</th>
<th>Store</th>
<th>Branch</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 4</td>
<td>1500</td>
<td>300</td>
<td>100</td>
<td>100</td>
<td>1750</td>
</tr>
</tbody>
</table>

1.5.4

Assuming that computes take 1 cycle, loads and store instructions take 10 cycles, and branches take 3 cycles, find the execution time of this program on a 3 GHz MIPS processor.

Program 4 cycles: \(1500 \times 1 + 300 \times 10 + 100 \times 10 + 100 \times 3 = 5800\) cycles

Program 4 execution time: \(5800\) cycles \(\times \frac{1\text{ second}}{3 \times 10^9\text{ cycles}} = 1.93 \times 10^{-6}\) s

1.5.5

Assuming that computes take 1 cycle, loads and store instructions take 2 cycles, and branches take 3 cycles, find the execution time of this program on a 3 GHz MIPS processor.

Program 4 cycles: \(1500 \times 1 + 300 \times 2 + 100 \times 2 + 100 \times 3 = 2600\) cycles

Program 4 execution time: \(2600\) cycles \(\times \frac{1\text{ second}}{3 \times 10^9\text{ cycles}} = 8.67 \times 10^{-7}\) s
Assuming that computes take 1 cycle, loads and store instructions take 2 cycles, and branches take 3 cycles, what is the speed-up of a program if the number of compute instructions can be reduced by one-half?

New Program 4 cycles: \( 750 \times 1 + 300 \times 2 + 100 \times 2 + 100 \times 3 = 1850 \) cycles

New Program 4 execution time: \( 1850 \text{ cycles} \times \frac{1 \text{ second}}{3 \times 10^9 \text{ cycles}} = 6.17 \times 10^{-7} \text{ s} \)

Speedup: \( \frac{\text{Old execution time}}{\text{New execution time}} = \frac{8.67 \times 10^{-7} \text{ s}}{6.17 \times 10^{-7} \text{ s}} = 1.41 \) times or 41% speedup.
7 Problem 7

Amdahl’s Law: Exercises 1.16.1b - 1.16.3b, 1.16.4.

The following table shows the execution time of five routines of a program running on different numbers of processors.

<table>
<thead>
<tr>
<th># Proc</th>
<th>Rtn A (ms)</th>
<th>Rtn B (ms)</th>
<th>Rtn C (ms)</th>
<th>Rtn D (ms)</th>
<th>Rtn E (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4</td>
<td>14</td>
<td>2</td>
<td>12</td>
<td>2</td>
</tr>
</tbody>
</table>

1.16.1

Find the total execution time, and how much it is reduced if the time of routines A, C, & E is improved by 15%.

Total execution time = 4 + 14 + 2 + 12 + 2 = 34 ms

New execution time = \[ \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Unaffected execution time} \]

= 0.85(4 + 2 + 2) + (34 − 4 − 2 − 2) = 32.8 ms

Speedup: \[ \frac{\text{Old execution time}}{\text{New execution time}} = \frac{34 \text{ ms}}{32.8 \text{ ms}} = 1.037 \]

1.16.2

By how much is the total time reduced if routine B is improved by 10%?

New execution time = \[ \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Unaffected execution time} \]

= 0.9(14) + (34 − 14) = 32.6 ms

Speedup: \[ \frac{\text{Old execution time}}{\text{New execution time}} = \frac{34 \text{ ms}}{32.6 \text{ ms}} = 1.043 \]

1.16.3

By how much is the total time reduced if routine D is improved by 10%?

New execution time = \[ \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Unaffected execution time} \]

= 0.9(12) + (34 − 12) = 32.8 ms

Speedup: \[ \frac{\text{Old execution time}}{\text{New execution time}} = \frac{34 \text{ ms}}{32.8 \text{ ms}} = 1.037 \]
1.16.4

Execution time in a multiprocessor system can be split into computing time for the routines plus routing time spent sending data from one processor to another. Consider the execution time and routing time given in the following table.

<table>
<thead>
<tr>
<th># Proc</th>
<th>Rtn A (ms)</th>
<th>Rtn B (ms)</th>
<th>Rtn C (ms)</th>
<th>Rtn D (ms)</th>
<th>Rtn E (ms)</th>
<th>Routing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td>78</td>
<td>9</td>
<td>65</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>44</td>
<td>4</td>
<td>34</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>23</td>
<td>3</td>
<td>19</td>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
<td>3</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>26</td>
</tr>
</tbody>
</table>

For each doubling of the number of processors, determine the ratio of new to old computing time and the ratio of new to old routing time.

<table>
<thead>
<tr>
<th># Proc</th>
<th>Computing Time</th>
<th>Computing Time Ratio</th>
<th>Routing Time Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>176</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>96</td>
<td>$\frac{96}{176} = 0.545$</td>
<td>$\frac{13}{11} = 1.181$</td>
</tr>
<tr>
<td>8</td>
<td>49</td>
<td>$\frac{49}{96} = 0.510$</td>
<td>$\frac{17}{13} = 1.308$</td>
</tr>
<tr>
<td>16</td>
<td>30</td>
<td>$\frac{30}{49} = 0.612$</td>
<td>$\frac{22}{17} = 1.294$</td>
</tr>
<tr>
<td>32</td>
<td>14</td>
<td>$\frac{14}{30} = 0.467$</td>
<td>$\frac{23}{22} = 1.045$</td>
</tr>
<tr>
<td>64</td>
<td>6.5</td>
<td>$\frac{6.5}{14} = 0.464$</td>
<td>$\frac{26}{23} = 1.13$</td>
</tr>
</tbody>
</table>

8 Problem 8

MIPS ISA: Exercises 2.12.1 - 2.12.3. Parts c, d. are not from the book!

In the following problems, you will investigate the impact of certain modifications to the MIPS ISA.

2.12.1

If the instruction set of the MIPS processor is modified, the instruction format must also be changed. For each of the suggested changes, show the size of the bit fields of an R-type format instruction. What is the total number of bits needed for each instruction?
a. 8 Registers
26 bits.
Register bits: \( \log_2(8) = 3 \)

\[
\text{opcode (6)} \quad \text{rs (3)} \quad \text{rt (3)} \quad \text{rd (3)} \quad \text{shamt (5)} \quad \text{funct(6)}
\]

b. 10 bit immediate constants
No change.

c. 128 Registers
38 bits.
Register bits: \( \log_2(128) = 7 \)

\[
\text{opcode (6)} \quad \text{rs (7)} \quad \text{rt (7)} \quad \text{rd (7)} \quad \text{shamt (5)} \quad \text{funct(6)}
\]

d. All arithmetic instructions can use base addressing mode with the last argument (Example: add $a0, $a2, 0[$t1])
48 bits.

\[
\text{opcode (6)} \quad \text{rs (5)} \quad \text{rt (5)} \quad \text{rd (5)} \quad \text{shamt (5)} \quad \text{funct(6)} \quad \text{immediate (16)}
\]

R-type instructions use the opcode 0 and the funct to denote the operation. Using an unused opcode (there are many) and the same funct combinations, all instructions with base addressing can be added. The length of the immediate was unspecified -- 16 is a good choice since it is a half-word, and the size used in I-type instructions.

However, no instruction that uses the shamt field can have base addressing as asked in the question, so it is possible to use the shamt field and allow only 5-bit offsets and retain the instruction length.

2.12.2
If the instruction set of the MIPS processor is modified, the instruction format must also be changed. For each of the suggested changes, show the size of the bit fields of an I-type format instruction. What is the total number of bits needed for each instruction?
a. 8 Registers
28 bits.
Register bits: \( \log_2(8) = 3 \)

```
| opcode (6) | rs (3) | rt (3) | immediate (16) |
```

b. 10 bit immediate constants
26 bits.

```
| opcode (6) | rs (5) | rt (5) | immediate (10) |
```

c. 128 Registers
36 bits.
Register bits: \( \log_2(128) = 7 \)

```
| opcode (6) | rs (7) | rt (7) | immediate (16) |
```
d. All arithmetic instructions can use base addressing mode with the last argument (Example: add $a0, $a2, 0[$t1])

Two acceptable answers:

1. No change using the assumption that the definition of an immediate must be an integer constant.

2. 37 bits for an instruction of the form addi $a0, $s1, 0($t1). We will need to add another 5 bits to incorporate the register data.

2.12.3

Why could the suggested change decrease the size of a MIPS assembly program? Why could the suggested change increase the size of a MIPS assembly program?

a. 8 Registers

Smaller instruction encoding may mean smaller programs -- this may be difficult in practice since the reductions aren’t byte-aligned, so addressing subsequent instructions is difficult. It also breaks the MIPS tenet of fixed-length instructions. Programs may increase in size because fewer registers means more data will ’spill’ into memory, leading to more load/store instructions.

b. 10 bit immediate constants

Smaller instruction encoding may mean smaller programs -- this may be difficult in practice since the reductions aren’t byte-aligned, so addressing subsequent instructions is difficult. It also breaks the MIPS tenet of fixed-length instructions. Programs may increase in size because to use immediate numbers larger than 10-bits, you need additional shift/or instructions.

c. 128 Registers

Longer instruction-encoding means longer programs that do the same operations. The instruction formats may be even larger than indicated above since the increases aren’t byte-aligned and addressing subsequent instructions is difficult. It also breaks the MIPS tenet of fixed-length instructions. Programs may decrease in size because more registers means you can ’hold’ onto more data in the processor without going out to memory, leading to fewer load/store instructions.

d. All arithmetic instructions can use base addressing mode with the last argument (Example: add $a0, $a2, 0[$t1])

A larger instruction format means programs that cannot make use of the new addressing mode will be longer. This may be further exacerbated by extending I and J-type instructions to 48-bits as well, to keep instructions fixed-length. However, based-addressing allows many load operations to be combined into an arithmetic instruction -- this is a reduction from 2*32 bytes to 48 bytes for each such reduction. A program that heavily uses this might grow
smaller. If I and J-types are made 48-bytes as well, 32-bit immediate constants and longer
jump offsets may also contribute to reducing the program length (though unlikely).

If the shamt field is used, the program will not grow larger with this addition, though it
may grow smaller. However, adding base-addressing still adds complexity to the processor
that may slow the program down. It is also less useful with just 5-bit offsets.