Chapter 3  Instructions: Language of the Machine

3.3 [10] Assume that the code from Exercise 3.2 is run on a machine with a 500-MHz clock that requires the following number of cycles for each instruction:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>add,addi,slt</td>
<td>1</td>
</tr>
<tr>
<td>lw, bne</td>
<td>2</td>
</tr>
</tbody>
</table>

In the worst case, how many seconds will it take to execute this code?

3.4 [5] Show the single MIPS instruction or minimal sequence of instructions for this C statement:

```c
a = b + 100;
```

Assume that `a` corresponds to register `$t0` and `b` corresponds to register `$t1`.

3.5 [10] Show the single MIPS instruction or minimal sequence of instructions for this C statement:

```c
```

Assume that `c` corresponds to register `$t0` and the array `x` has a base address of 4,000,000.

3.6 [10] The following program tries to copy words from the address in register `$a0` to the address in register `$a1`, counting the number of words copied in register `$v0`. The program stops copying when it finds a word equal to 0. You do not have to preserve the contents of registers `$v1`, `$a0`, and `$a1`. This terminating word should be copied but not counted.

```
loop:    lw $v1,0($a0)  # Read next word from source
         addi $v0,$v0,1  # Increment count words copied
         sw $v1,0($a1)   # Write to destination
         addi $a0,$a0,1  # Advance pointer to next source
         addi $a1,$a1,1  # Advance pointer to next dest
         bne $v1,$zero,loop  # Loop if word copied ≠ zero
```

There are multiple bugs in this MIPS program; fix them and turn in a bug-free version. Like many of the exercises in this chapter, the easiest way to write MIPS programs is to use the simulator described in Appendix A. (Go to www.mkp.com/cod2c.htm to get a copy of this program.)

3.7 [15] Using the MIPS program in Exercise 3.6 (with bugs intact), determine the instruction format for each instruction and the decimal values of each instruction field.
3.11 Consider the following fragment of C code:

```c
for (i=0; i<=100; i++) { a[i] = b[i] + c;
```

Assume that a and b are arrays of words and the base address of a is in $a0$ and the base address of b is in $a1$. Register $t0$ is associated with variable $i$ and register $s0$ with $c$. Write the code for MIPS. How many instructions are executed during the running of this code? How many memory data references will be made during execution?

3.12 Given your understanding of PC-relative addressing, explain why an assembler might have problems directly implementing the branch instruction in the following code sequence:

```assembly
    here:  beq $t1, $t2, there
           ...  
    there:  add $t1, $t1, $t1
```

Show how the assembler might rewrite this code sequence to solve these problems.

3.13 Consider an architecture that is similar to MIPS except that it supports update addressing (like the PowerPC) for data transfer instructions. If we run gcc using this architecture, some percentage of the data transfer instructions shown in Figure 3.38 on page 189 will be able to make use of the new instructions, and for each instruction changed, one arithmetic instruction can be eliminated. If 25% of the data transfer instructions can be changed, which will be faster for gcc, the modified MIPS architecture or the unmodified architecture? How much faster? (You can assume that both architectures have CPI values as given in Exercise 3.16 and that the modified architecture has its cycle time increased by 10% in order to accommodate the new instructions.)

3.14 When designing memory systems, it becomes useful to know the frequency of memory reads versus writes as well as the frequency of accesses for instructions versus data. Using the average instruction-mix information for MIPS for the program gcc in Figure 3.38 on page 189, find the following:

a. The percentage of all memory accesses that are for data (vs. instructions).

b. The percentage of all memory accesses that are reads (vs. writes). Assume that two-thirds of data transfers are loads.

3.15 Perform the same calculations as for Exercise 3.14, but replace the program gcc with spice.