CSE30321 Computer Architecture I Fall 2009

Homework 04: MIPS Assembly Language (100 Points)
Assigned: September 22 Due: September 29

This homework will familiarize you with the details of MIPS assembly at a hands-on level. While it is important to talk about the theoretical aspects of an instruction set – for example, the merits of an accumulator machine vs. a three-operand format – it's easiest to grasp the impacts of these design choices, and to see how the features of a processor are used in real life, by writing real code. For these problems, you might find the green insert inside the cover of Patterson and Hennessy useful.

Problem 1

Part A (20 points):
Given the instructions below, note the instruction type (R, I or J) of each and then describe what you think this code does.

```
     addi $10, $0, 0
L1:  beq $8, $0, L2
     addi $10, $10, 1
     lw $8, 4($8)
     j L1
L2:
```

Part B (20 points):
Given the binary machine code below, note the instruction types, separate the instructions into fields and write the equivalent assembly. Then indicate what this code does.

```
0010 0000 0000 1001 0000 0000 0000 1111
0010 0000 0000 1010 0000 0000 0000 0001
0010 0000 0000 1011 0000 0000 0000 0000
0000 0001 0110 1010 0101 1000 0010 0000
0010 0001 0100 1010 0000 0000 0000 0001
0001 0101 0010 1010 < binary for -12  >
```

Problem 2

Part A (20 points):
Assume that you have a two-dimensional array *a* – i.e. a[m][n] – where *m* is 10 and *n* is 5. The data elements a[i][j] are stored in memory and the address of the first element – i.e. a[0][0] – is in $t0. Write an assembly language program using the MIPS instruction set that adds the 10 elements a[0][3], a[1][3], ... a[9][3]. You may assume that data is stored sequentially in memory as follows:

```
  X:  a[0][0]
X+4:  a[0][1]
X+8:  a[0][2]
      ....
```

Also, assume that m, and n are stored in registers $t1 and $t2 respectively. You must use a loop to accomplish this. For maximum partial credit, add descriptive comments to your answer – e.g. "$t3 = i" or "$t4 = accumulated sum + next array element", etc. The clock rate of this machine is 1 GHz.

Part B (10 points):
The FSM for a subset of MIPS instructions appears in Fig. 1. Using this information, calculate the execution time for your code. (Note: Extra credit will be given for the solution that offers the lowest overall execution time.)

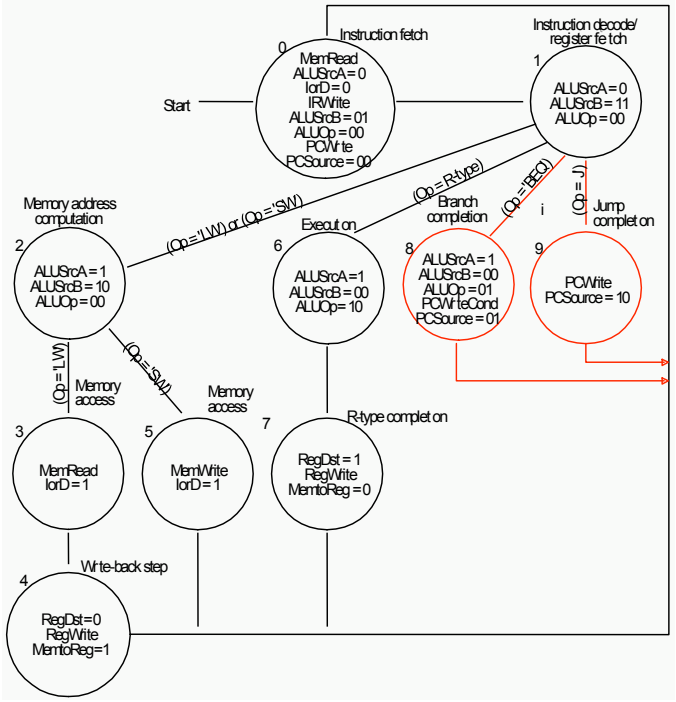

Figure 1: MIPS FSM.

Problem 3

(20 points):
Examining possible methods for compilers to optimize code is important for understanding high-level language performance. The following C code could be used to count the number of characters in a string (i.e., `strlen()`) and will be studied in this problem.

```
while (str[i] != 0)
  i++;
```

Look at the assembly below. The code uses both a conditional branch and an unconditional jump each time through the loop. Only inefficient compilers would produce code with this loop overhead. Now take the above C code, and translate it into mips assembly using only a single jump/branch instruction. How many instructions are executed before and after the optimization if `str[9] == 0`? *Note: We use load byte (*`lb`*) here rather than load word (*`lw`*), because a character takes a single byte of memory.*

```
        li   $s5, 10
Loop:   add  $t1, $s3, $s6   # $t1 = address of str[i]
        lb   $t0, 0($t1)     # Temp reg $t0 = str[i]
        beq  $t0, $s5, Exit  # go to Exit if str[i] == '\0'
        addi $s3, $s3, 1     # i = i + 1
        j    Loop
Exit:
```

With a RISC instruction set like the MIPS, the instructions are often so simple that it takes more than one to perform a common task. This is the point of a Reduced Instruction Set Computer: to break down the tasks that instructions perform into the smallest atomic chunks possible. This helps in high-performance designs (as we shall see later in the class). However, it means that the assembly programmer, or the compiler, produces common idioms a few instructions long to perform simple tasks.

Problem 4.1

(15 points):
Do Exercises 2.15.1b, 2.15.2b, 2.15.3b, 2.15.4a and 2.15.5a in Patterson and Hennessy on p. 193-194 (also reproduced below for your convenience). Again, the green insert in the text should prove helpful.

For 2.15.1b-2.15.3b, the xnor instruction is not included in the MIPS instruction set. How could the instruction xnor $t1, $t2, $t3 (bit wise exclusive NOR) be implemented?

2.15.1b:
If the value of $t2 = 0x00FFA5A5 and the value of $t3 = 0xFFF003C, what is the result in $t1?

2.15.2b:
This logical instruction is not included in the MIPS instruction set, but can be synthesized using one or more MIPS assembly instructions. Provide a minimal set of MIPS instructions that may be used in place of the hypothetical xnor.

2.15.3b:
For each sequence of instructions in 2.15.2b, show the bit-level representation of each instruction.

In the rest of this exercise, you will be asked to evaluate the statement and implement the C-statement: “A = B & C[0];” using MIPS assembly instructions.

2.15.4a:
If the memory location at c[0] contains the integer value 0x00001234, and the initial integer values of A and B are 0x00000000 and 0x00002222, what is the resulting value of A?

2.15.5a:
For the C statements in the table above, write a minimal sequence of MIPS assembly instructions that does the identical operation.

Problem 4.2

(15 points):
Do Exercises 2.16.4b, 2.16.5b, and 2.16.6b in Patterson and Hennessy on p. 195 (also reproduced below for your convenience). Again, the green insert in the text should prove helpful.

For these problems, $t0 = 0x20001400. You will be asked to evaluate the outcome of different branches.

2.16.4b:
Given the value of $t0 above, what is the value of $t2 after the following instructions?

```
       slt $t2, $t0, $t0
       bne $t2, $0, ELSE
       j   DONE
ELSE:  addi $t2, $t2, 2
DONE:
```

2.16.5b:
Given the value of $t0 above, what is the value of $t2 after the following instructions?

```
       sll $t0, $t0, 2
       slt $t2, $t0, $0
```

2.16.6b:
Suppose the program counter (PC) is set to 0x2000000. Is it possible to use the jump (j) MIPS assembly instruction to set the PC to the address as shown above? Is it possible to use the branch-on-equal (beq) MIPS assembly instruction to set the PC to the address as shown above?