CSc 252: Computer Organization
Fall 18 (Lewis)

Test 2
Thu 20 Sep 2018

Solutions

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Answer, Page 1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Short Answer, Page 2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Truth Tables and Sum-of-Products</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Masks</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Writing MIPS, part 1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Writing MIPS, part 2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1. For each question below, give a short answer - a few words or symbols, maybe a sentence or two.  
   (a) (10 points) For each task below, give a single instruction which will accomplish it.

   Multiply $s0$ by 8, and put the result into $t3$. (Remember that you're not allowed to use any actual “multiply” instruction yet!)

   **Solution:**  
   \[
   \text{addi } $v0, $zero, 1  
   \text{add } $a0, $s0, $zero  
   \text{syscall}
   \]

   Decrement $s7$ by $s6$.

   **Solution:**  
   \[
   \text{sub } $s7, $s7, $s6
   \]

   Assuming that we have an address already stored in $s2$, store the contents of register $s4$ to that location.

   **Solution:**  
   \[
   \text{sw } $s4, 0($s2)
   \]

   Place the address of the variable asdf into $s0$.

   **Solution:**  
   \[
   \text{la } $s0, asdf
   \]

   Compare $s0$ to $s1$. Branch to the label GO_HERE if they are different.

   **Solution:**  
   \[
   \text{bne } $s0,$s1, GO_HERE
   \]

   (b) (5 points) Suppose that you have an integer stored in $s0$. Give the MIPS code to print that integer. (You don’t have to print anything other than the integer - not even a trailing newline!)

   **Solution:**  
   \[
   \text{addi } $v0, $zero, 1  
   \text{add } $a0, $s0, $zero  
   \text{syscall}
   \]
2. (a) (10 points) Each of the following instructions is **invalid**. Explain what is wrong with each one.

```asm
la $t0, 0($t1)
```

**Solution:** The `LA` instruction has the wrong format. This is the proper format for a `LW` instruction.

```asm
add $s0, $s1, 1
```

**Solution:** This has a constant as the second input; this needs to be an `ADDI` instruction.

```asm
lw $s3, foobar
```

**Solution:** This is a Load-Word but it uses the Load-Address format.

```asm
subi $t3, $t3, 10
```

**Solution:** There is no `SUBI`. Use `ADDI` with a negative value instead.

```asm
add $t2, $t3, $t4, $t5
```

**Solution:** There are too many parameters; MIPS instructions allow at most 3 registers.

(b) (5 points) Explain the difference between the `.text` and `.data` sections of a MIPS program. What sort of things can you put into each section?

**Solution:** The `.text` section contains code; it contains instructions like `ADD`, `LW`, etc. The `.data` section contains data; it contains global variables (like `.word`) and string constants.
3. (a) (10 points) Fill in the following truth table. The outputs have the following rules:

- $X$ is true if $A,B,C,D$ are all identical.
- View the inputs $A,B$ as a two-bit unsigned integer ($A$ is the MSB), and $C,D$ as another ($C$ is the MSB). $Y$ is true if $A,B$ is greater than $C,D$.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
<th>$X$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
(b) (10 points) Now, write a sum-of-products expression for each of the outputs. 
You do not need to draw the circuit diagram for either of these outputs.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Solution: 
\[ X = \overline{A} \overline{B} \overline{C} \overline{D} + A B C D \]
\[ Y = \overline{A} B \overline{C} \overline{D} + \overline{A} B \overline{C} D + A B \overline{C} \overline{D} + A B \overline{C} D + A B C \overline{D} \]
4. (15 points) In this problem, you will use a mask to read out certain bits from $s4$, and store them into $s5$. You may modify any $tX$ register, but do not modify any $sX$ register (other than $s5$).

First give the mask, then give the code.

- Keep bits 8 through 11, and 20 through 23 (inclusive).
- Use only the instructions and, andi, addi, and sll.
- For full credit, do this in four instructions.

REMEMBER: The ‘I’ instructions (andi, addi, etc.) can only take 16-bit constants!!!

Solution: Mask:

```
0000 0000 1111 0000 0000 1111 0000 0000
3  2  2   1  8
1  3  0   1
```

Mask, in hex (not required): 0x00f0
0f00

```
addi $s5, $zero,0xf0  # s5 = 0xf0
sll $s5, $s5,16       # s5 = 0xf0_0000
addi $s5, $s5, 0x0f00  # s5 = 0x00f0_0f00
and $s5, $s4,$s5
```

5. (15 points) In this problem, assume that we have two integer variables, named king and ashen. Write MIPS code which implements the following C code. When the C code writes to a variable, make sure to update the related variable in memory!

```
king = king*2 + ashen;
ashen = 17;
```

Special Limitations:

- You may use la for each variable once - but do not use it more than once per variable.

Solution:

```
la $t0, king    # t0 = &king
la $t1, ashen   # t1 = &ashen
lw $t2, 0($t0)  # t2 = king
lw $t3, 0($t1)  # t3 = ashen

add $t4, $t2,$t2 # t4 = 2*king
add $t4, $t4,$t3 # t4 = 2*king+ashen
sw $t4, 0($t0)  # king = 2*king+ashen

addi $t5, $zero,17
sw $t5, 0($t1)  # ashen = 17
```
6. (20 points) This question assumes some MIPS code (on the last page of this exam). The code sets up memory locations bee, spider, whale, tiger, charles. The code then loads the values of some of these variables into the indicated MIPS registers. In answering this question, you can assume this code has already been executed, and that the value of some of the variables are already in the indicated registers.

**Special Limitations:**

- You may need to read from memory - but do **not** write to memory unless specifically instructed.
- Do **not** modify any sX register, unless specifically instructed.

See the last page for the list of allowable instructions.

If (bee+harold < whale-tiger), then store the value of bee into the memory location harold; otherwise, store the value of whale into tiger.

```
Solution:
We had to add a note on the overhead: “Assume that there is a word ‘harold’, and that it is not loaded into any register.”

la   $t0, harold  # t0 = &harold
la   $t1, tiger   # t1 = &tiger
lw   $t2, 0($t0)  # t2 = harold
lw   $t3, 0($t1)  # t3 = tiger
add  $t4, $s5,$t2  # t4 = bee+harold
sub  $t5, $s6,$t3  # t5 = whale-tiger
slt  $t6, $t4,$t5  # t6 = (bee+harold < whale-tiger)
beq  $t6,$zero, FALSE  # if (bee+harold >= whale-tiger) skip ahead
sw   $s5, 0($t0)  # harold = bee
j    AFTER

FALSE:
sw   $s6, 0($t1)  # tiger = whale
j    AFTER

AFTER:
```

Page 7
# values are hidden so that you can’t hardcode the answers!

.data
bee: .word xxx
spider: .word xxx
whale: .word xxx
tiger: .word xxx
charles: .word xxx

.text
main:
   # set $s3 = charles
   la $s3, charles
   lw $s3, 0($s3)

   # set $s5 = bee
   la $s5, bee
   lw $s5, 0($s5)

   # set $s6 = whale
   la $s6, whale
   lw $s6, 0($s6)

Allowable Instructions
When writing MIPS assembly, the only instructions that you are allowed to use (so far) are:

- add, addi, sub
- beq, bne, j
-slt, slti
- and, or, nor, andi, ori, xor, xori
- sll, srl, sra
- lw, lh, lb, sw, sh, sb
- la
- syscall