Problem Set #2
Issued Thursday, 2/7/08; Due Thursday, 2/21/08

Homework Information: Some problems are probably too long to be done the night before the due date, so plan accordingly. Late homework will be penalized according to the rules given on the course web site. Feel free to get help from others, but the work you hand in should be your own.

Problem 1. “Matters of State”

When specifying a state machine’s transition table it is often convenient to specify that a particular state “does not care” about the value of an input. In our Roboant example, the “–” was used to indicate this. This is actually a shorthand that shortens the number of table rows.

(A) Rewrite the default Roboant state machine from lecture without using any –’s.

<table>
<thead>
<tr>
<th>Current</th>
<th>Left</th>
<th>Right</th>
<th>Next</th>
<th>TL</th>
<th>TR</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>lost</td>
<td>0</td>
<td>0</td>
<td>lost</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>lost</td>
<td>1</td>
<td>-</td>
<td>rccw</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>lost</td>
<td>0</td>
<td>1</td>
<td>rccw</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>rccw</td>
<td>1</td>
<td>-</td>
<td>rccw</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rccw</td>
<td>0</td>
<td>1</td>
<td>rccw</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rccw</td>
<td>0</td>
<td>0</td>
<td>wall1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>wall1</td>
<td>-</td>
<td>1</td>
<td>wall2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>wall1</td>
<td>-</td>
<td>0</td>
<td>wall1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>wall2</td>
<td>0</td>
<td>0</td>
<td>wall1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>wall2</td>
<td>0</td>
<td>1</td>
<td>wall2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>wall2</td>
<td>1</td>
<td>-</td>
<td>rccw</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(B) Our default Roboant has four states (requiring 2-bits to encode), two inputs, and three outputs. How many possible ways can a Roboant with these specs be programmed (not all programs need to make sense)?

(C) Specifying a state machine via a transition table resembles writing a computer program. What columns of the state transition able determine the operations that Roboant can perform? What columns determine the order that operations are executed?

(D) One trick that is often used when designing state machines is to make outputs play a dual role, by using them to encode the state. Which two outputs of Roboant could be used to uniquely encode the four states {lost, rccw, wall1, wall2}?
Problem 2. “Shortcomings of the UNC-101”

The UNC-101 has a small, but deceptive instruction set. Many common operations can be performed using a single instruction. For the following give a single instruction that performs the specified operation.

- (A) neg $d$, $a$ Negate the contents of register a and place the result in register d. (Negation implies computing the 2’s complement).
- (B) com $d,a$ Complement each bit of register a and place the result in register d. (replaces each 1 with a 0, and each 0 with a 1.
- (C) sgn $d,a$ If the contents of register a are negative load register d with -1; otherwise, load register d with 0.
- (D) clr $d$ Clear the contents of register d (set it to 0).
- (E) br label Unconditionally branch to the specified label.

Problem 3. “Manual Assembly Required”

Give the binary encoding for the following UNC-101 instructions:

1. add $0,0,0$
2. shl $10,10,4$
3. xor $12,11,12$
4. subi $1,1,1$
5. beq $0,1,0,0x0200$

Problem 4. “Go Forth and Multiply”

Only addition and subtraction are provided as arithmetic instructions in the UNC-101. More complicated operations, like multiplication and division must be written as procedures. The following code fragments multiply the contents of register $1$ by the contents of register $2$, and place the result in register $3$. Enter each program into the UNC-101 simulator and answer the following questions:

(A) How many words in length is each procedure? How many steps does each multiply procedure require to compute its product?

(B) Does either MultA or MultB support multiplication with a negative number (consider both arguments)? If not, suggest how they might be fixed to handle this case (just explain your approach, you don’t need to write the code).
Project #2: UNC-101

Use the simulator demonstrated in lecture to write a procedure for finding the largest number in a list. When called, the address of the list will be in register $1$, and its length in register $2$. Upon return, the largest value should be in register $3$.

The URL for the UNC-simulator, and its documentation can be found at:

http://www.unc.edu/courses/2008spring/comp/101/001/PS2/

You can use the following code fragment to test your procedure:

```assembly
# find the maximum value in an array
main:   addi $1,$0,array
        addi $2,$0,10
        beq  $15,$0,$0,max
halt:   beq  $0,$0,$0,halt

array:  .data 634,32,460,902,343,956,28,587,460,202

# your code goes here
```