Sequential Logic Design Review for Midterm #2

CS 64: Computer Organization and Design Logic Lecture #14

Ziad Matni
Dept. of Computer Science, UCSB
Administrative

• Midterm #2 is on **Thursday, 3/2**
  – Covers everything from Lecture 8 thru 14

• Today we’ll cover S-R Latches, then do a review for the midterm
Abstract Schematic of the MIPS CPU
Sequential Logic

- **Combinatorial Logic**
  - Combining multiple logic blocks
  - The output is a function *only* of the present inputs
  - There is no memory of past “states”

- **Sequential Logic**
  - Combining multiple logic blocks
  - The output is a function of *both* present and *past* inputs
  - There exists a memory of past “states”
The S-R Latch

- Only involves 2 NORs
- The outputs are fed-back to the inputs
- The result is that the output state (either a 1 or a 0) is maintained even if the input changes
How a Latch Works

• Note that if one NOR input is 0, the output becomes the inverse of the other input

• So, if output Q already exists and if S = 0, R = 0, then Q will remain at whatever it was before! (hold output state)

• If S = 0, R = 1, then Q becomes 0 (reset output)

• If S = 1, R = 0, then Q becomes 1 (set output)

• Making S = 1, R = 1 is not allowed (undetermined output)
Consequences?

- As long as $S = 0$ and $R = 0$, the circuit output holds memory of its prior value (state).

- To change the output, just make $S = 1$ (but also $R = 0$) to make the output 1 (set) OR $S = 0$ (but also $R = 1$) to make the output 0 (reset).

- Just avoid $S = 1$, $R = 1$...

<table>
<thead>
<tr>
<th>$S$</th>
<th>$R$</th>
<th>$Q_0$</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q*</td>
<td>Hold output</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Reset output</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Set output</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>Undetermined</td>
</tr>
</tbody>
</table>
About that $S = 1$, $R = 1$...

- What if we avoided it on purpose by making $R = \text{NOT} (S)$?
  - Where’s the problem?

- This, by itself, precludes a case when $R = S = 0$
  - You’d need that if you want to preserve the previous output state!

- Solution: the *clocked latch* and *the flip-flop*
  - *Next time on a new & exciting episode of.......CS.....64.....!*
Midterm #2 Review

**Topics**

- MIPS assembly programs
  - Loops
  - Memory manipulation (Arrays & Pointers)
  - Functions and using the Call Stack
  - Using MIPS convention rules (when asked to)
  - You still have to know the basics
    - Basic programming rules, syscall, if/then, etc...

- Logic Design
  - Basic logic blocks/functions
  - Combinatorial logic functions
  - Multiplexers (muxes)
  - S-R latches
Midterm #2 Details

- **Thursday, 3/2** in this classroom
- **Starts at 3:30pm **SHARP** **
  - Please start arriving 5-10 minutes before class
- **I will chose where you sit!**
  - Wait on my instruction...
- **Duration: 1 hour long**
- **Closed book**: no calculators, no phones, no computers
- **Only 1 sheet (single-sided) of written notes and the MIPS R.C.**
  - Must be no bigger than 8.5” x 11”
  - You have to turn it in with the exam
- **You will write your answers on the exam sheet itself.**
Midterm #2 Details

- Will have short answer types
  - Including logic circuit exercises

- Will have assembly language programming
  - Some with MIPS convention, some without

- No multiple choice questions

- PLEASE DON’T FORGET YOUR MIPS REFERENCE CARD!!!!
Permissible Code in the Exam

**Remember:**

- Your code must use **ONLY** those instructions and pseudoinstructions listed on the MIPS reference card.

- In addition, **ONLY** the following pseudoinstructions are allowed:
  - blt (Branch Less Than)
  - bgt (Branch Greater Than)
  - ble (Branch Less Than or Equal)
  - bge (Branch Greater Than or Equal)
  - li (Load Immediate)
  - la (Load Address)
  - move (Move)

- See this for full details:
  [http://www.cs.ucsb.edu/~zmatni/cs64w17/documentation/mips_instruction_policy.html](http://www.cs.ucsb.edu/~zmatni/cs64w17/documentation/mips_instruction_policy.html)
Loops 1

for (int t0 = 10; t0 > 5; t0--)
    print ("Loop");
print ("The End");

Examples used today will be made available on the class website.
int t0 = 5; // or something else
int t1 = 0;

if (t0 < 8) {
    while (t1 <= 5) {
        print ("Loop ");
        t1 ++;
    }
}
else
    print("No Loop");

.data
str1: .asciiz "Loop 
str2: .asciiz "No Loop"
.text
main:
li $t0, 5  # or something else
li $t1, 0
addi $t4, $zero, 8
slt $t5, $t0, $t4
beq $t5, $zero, noloop
loop:
addi $t2, $zero, 6
slt $t3, $t1, $t2
beq $t3, $zero, end
la $a0, str1
li $v0, 4
syscall
addi $t1, $t1, 1
j loop
noloop:
lc $a0, str2
syscall
end:
li $v0, 10
syscall
Arrays
from review sheet

array[0] = 0;
array[1] = 1;
unsigned int s0, s1 = 4;

for (s0 = 2; s0 <= s1; s0++) {
    array[s0] = array[s0 - 1] +
    array[s0 - 2];
}

s2 = array[s1];

main:
    la $t0, array
    sw $zero, 0($t0)
    li $t1, 1
    sw $t1, 4($t0)
    li $s0, 2

loop:
    slt $t1, $s1, $s0
    bne $t1, $zero, after_loop
    sll $t2, $s0, 2
    addu $t3, $t0, $t2
    lw $t4, -4($t3)
    lw $t5, -8($t3)
    addu $t6, $t4, $t5
    sw $t6, 0($t3)
    addiu $s0, $s0, 1
    j loop

after_loop:
    sll $t1, $s1, 2
    addu $t2, $t1, $t0
    lw $s2, 0($t2)
The function **bar** is implemented in MIPS assembly, which takes two parameters, which we'll name x and y, respectively. The function is supposed to return the result of \((x \times y) - (x + y)\). However, there is a bug in the function, with respect to the MIPS calling convention. What's wrong with the code, and how could it be fixed?

```
bar:
    mult $a0, $a1
    mflo $s0
    add  $s1, $a0, $a1
    sub  $v0, $s0, $s1
    jr   $ra
```

**ANSWER:**
The code uses registers $s0 and $s1, which are **supposed to be preserved across a call according to the MIPS calling convention**. As such, in order to safely use $s0 and $s1, **the code needs to first save their values on the stack, and then later restore their values** from the stack just before returning from bar.

However, the code above does not do this, so it is buggy. **The solution is to either save and restore registers $s0 and $s1 as described, or use registers $t0 and $t1 instead, which do not need to be preserved across calls.**
Create the assembly code from the following C program using MIPS convention

```c
int gamma(int x) {
    if (x < 0) {
        return x;
    } else {
        return gamma(x - 1);
    }
}
```

```
gamma:
    addiu $sp, $sp, -4
    sw $ra, 0($sp)
    bge $a0, $0, else
    move $v0, $a0
    j return
else:
    addi $a0, $a0, -1
    jal gamma
    lw $ra, 0($sp)
return:
    addiu $sp, $sp, 4
    jr $ra
```
Simplify the Following Expressions  
*Without* the Use of K-Map

1. \[ F = (A.B + 1).(AB + 0) \]
   \[ \Rightarrow F = A.B \]

2. \[ F = !(X + Y.(!Z)) \quad \text{--- solution must have at least 1 OR} \]
   \[ \Rightarrow F = !X . !(Y.(!Z)) = !X . (!Y + !(!Z)) = !X.(!Y + Z) \]

3. \[ M = A'.B.C + A'.B.C.D + A'.B.C' \]
   \[ \Rightarrow M = A'.B.C (1 + D) + A'.B.C' = A'.B (C + C') = A'.B \]
Find the Simplest Function and Draw the Circuit

Out(A, B, C, D) = C'.D' + B.D' = D'.(B + C')

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Matni, CS64, Wi17
Your To-Dos

• Study for the Midterm! 😊
  — Good luck!