**CSE370 Fall '99**
**Assignment 6**
Distributed: 11/8/99
Due: 11/15/99

**Reading:**

Katz, Chapter 6 through 6.4.3, The rest of Chapter 6 is optional
Verilog Introduction: http://www.eg.bucknell.edu/~cs320/1995-fall/verilog-manual.html
Recommend sections 2.2-2.7 (Same as last week)
Design Works Howto: http:…370/98au/admin/Tools/DesignWorks/Verilog.html

The purpose of this assignment is to design a simple 8-bit ripple carry ALU that has all the functionality needed run a small computer. The ALU is specified as follows:

**Arithmetic Operations (All are assumed to be in 2's Complement)**

| | |
|---|---|
| ADD | F = A + B |
| INC | F = A + 1 |
| DEC | F = A – 1 |
| SUB | F = A – B |
| CMP | F = A – B |
| PASS | F = A |
| NEG | F = –A |

**Bitwise Logic Operations**

| | | |
|---|---|---|
| XOR | Fi = Ai xor Bi | |
| XNOR | Fi = Ai xnor Bi | |
| OR | Fi = Ai or Bi | |
| AND | Fi = Ai and Bi | |
| NOT | Fi = ~Ai | |
| SHL | Fi = Ai-1 | (F0 = 0, C8 = A7) (Shift Left) |
| SHR | Fi = Ai+1 | (F7 = 0) (Shift Right) |

In addition to implementing these functions, the system must produce the following Condition Code outputs:

| | |
|---|---|
| V | Result of operation is a 2's Complement overflow |
| C | Carry (Unsigned) |
| Z | Result of operation is Zero |
| N | Result of operation is Negative (2's Complement) |

The V,Z,N outputs are generated assuming 2's complement representation, the C output is generated assuming unsigned representation.

**Part A. Design a single bit slice of the ALU.**

Determine what control lines you need, define the role of each of each one, and make a table showing how the system implements each of the 14 operations above. The columns in your table should show the control lines, the operation being performed, and the pre- or post-processing that is done so that the + operation will produce the desired result (similar to what we did in lecture). Define the control lines to simplify the logic as much as possible. The inputs to each slice should include your control lines along with at least Ai, Bi, Ci. You may need some other inputs as well. The outputs should be Ci+1, Fi. Turn in the table and the bit-slice schematic.

**Part B. Control Encoding and Decoding.**
Define a unique four-bit opcode for each instruction. In Verilog, implement a block that converts the each opcode to the corresponding control lines values. You may use any style of Verilog that you choose. Turn in the Verilog model along with any explanations that are needed.

**Part C. Complete the System**
Using the ALU bit-slices from Part A and the Verilog decoder from part B along with any other logic needed, build an 8-bit ripple carry adder that implements the all the functions **and the condition codes** defined above. Connect A and B inputs to xA5 and x55 respectively and test your circuit on each of the 14 instructions. Turn the complete schematic with a spreadsheet showing the encoded control inputs with **verified expected** values for all outputs. **Use the Group Numbered Signals feature of the spreadsheet tool (under "get header" command) it works!** Determine the length of the longest delay path from any input to any output, assuming that muxes and your Verilog decoder require two gate delays. Mark the path in RED on your schematic and note the number of gates in path.

**Part D. Optional Design Contest**
Complete the gate level implementation of the your decoder from part B. Count the number of gates and the worst case delay for the ENTIRE SYSTEM and submit those numbers with your assignment . If the best design from the class is as good or better that best that the Deepak, Sorin, and I can come up with we will throw a pizza party for the entire class. This should be quite doable, especially since we are sooo busy and there really aren't that many ways to do it. This is like GOLF, you are responsible for the score you submit. If it is in error, your submission will not be counted. In the case of the tie on gate count, the fastest design wins. For this contest, Muxes and other PRIMLOG types of devices count as two delays and the number of gates in the SOP implementation of the device.