ECE 2330 DIGITAL LOGIC DESIGN
UNIVERSITY OF VIRGINIA
STUDIO ASSIGNMENT 12

Standard honor statement. ***Your work should be your own.***

OBJECTIVE

This assignment will allow you to better understand how a processor works.

PROBLEM DESCRIPTION

Add 2 new instructions to our 4-instruction processor: **Load-Constant** and **Jump-if-zero.**

DESIGN PROCEDURE

Chapter 8, specifically section 8.4, describes the changes needed to add 2 new instructions to our instruction set. Read this section carefully, and study the 4-instruction processor *Logisim* file to see the relationship between it and the book's rendition. There are a few minor differences between the book's design and what is implemented in Logisim: the opcodes are different, the names of the signals are different, minor things like that.

Once you understand the extensions as described in the book (and on the slides used in class on Frida April 22) you can implement these changes in the 4-processor *Logisim* file.

There is a 6-instruction skeleton file in which most of the work has been done for you. Use this skeleton file as the starting point for your implementation. The program that is loaded in the ROM is listed below. It's figure 8.14 from the text.

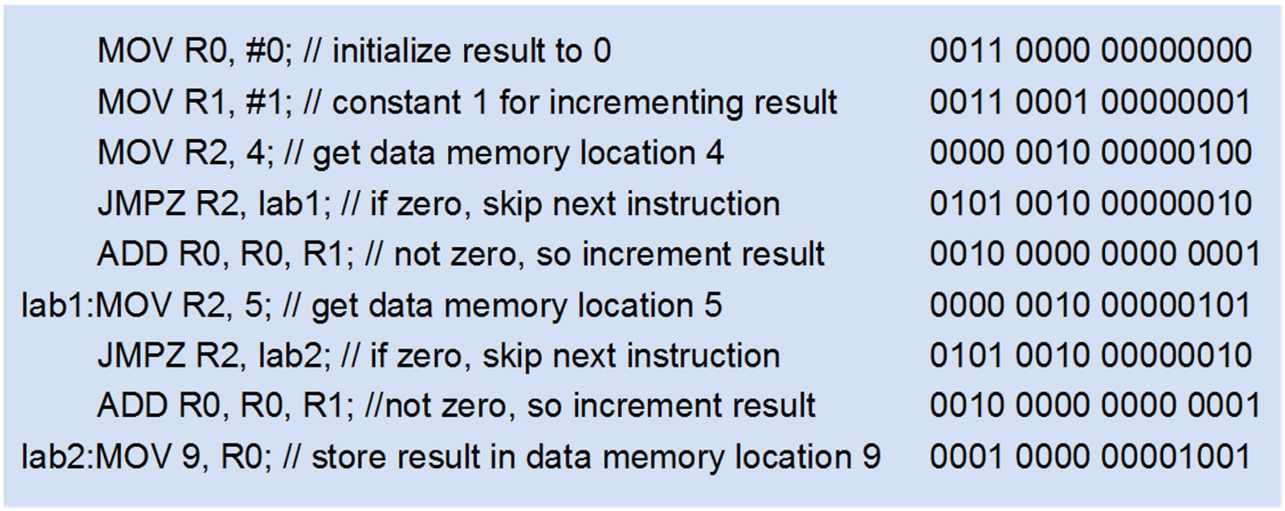
PRE-WORK

Your design should be complete before you come to the studio. All state outputs must use tri-state buffers.

Make sure you completely understand the 6-instruction processor and how the changes were implemented from the 3-instruction or 4-instruction processor. There are changes in the datapath and changes in the controller. Make sure you understand these changes. Be prepared to answer questions in the studio.

GRADING

- 2 points each for the correct output functions in the 3 added states. The outputs should be complete and correct with nothing extra. Explanation of the outputs is included in these points.
- 3 points for demonstrating that your design works and for being able to make program and data changes as requested by the TA.
- 1 point for professionalism



```
      MOV R0, #0; // initialize result to 0                  0011 0000 00000000
      MOV R1, #1; // constant 1 for incrementing result      0011 0001 00000001
      MOV R2, 4; // get data memory location 4               0000 0010 00000100
      JMPZ R2, lab1; // if zero, skip next instruction       0101 0010 00000010
      ADD R0, R0, R1; // not zero, so increment result       0010 0000 0000 0001
lab1:MOV R2, 5; // get data memory location 5                0000 0010 00000101
      JMPZ R2, lab2; // if zero, skip next instruction       0101 0010 00000010
      ADD R0, R0, R1; //not zero, so increment result        0010 0000 0000 0001
lab2:MOV 9, R0; // store result in data memory location 9    0001 0000 00001001
```

(a) (b)