EECE2323 **Digital Logic Design Lab** **Prelab 8-1**

Adding Branch Logic to the Datapath to Complete your Computer

Begin by reading your lab manual. Then use the Instruction Set document uploaded on Canvas as a road-map for answering the following questions.

1 Design a PC logic with branching capability

Write a Verilog module which accepts the clock, a reset, the immediate value from the instruction word (least significant byte), and the zero output from the ALU as inputs and generates an 8-bit Program Counter (PC) for the output.

Note that in this architecture when we have a branch, the next PC value should be the current PC value plus the offset which is extracted from the branch instruction. The offset is represented in two's complement, so the range of branch target is from PC - 128 to PC + 127. Note that the value of PC should not exceed 0xFF as we have a 256-deep instruction memory. You do not need to check for this condition in your hardware.

2 A Signed Multiplier in Assembly

Based on the instruction set document write an assembly program which multiplies two signed numbers (in two's complement system).

The multiplier and multiplicand could be any number within the 8 bit data range and with any sign. Make sure your assembly program is generic.

Note that the in this lab you have branching hardware so you can insert conditions and labels in your program.

3 Generate the Machine Codes using the Assembler

Use the assembler available on Canvas to generate the machine codes out of your assembly program. Use different sign combinations for multiplier and multiplicand, such as both being positive or negative, or having opposite sign for them. Use immediate values to initialize your data inputs.

Verify the output of the assembler by comparing the output machine codes with the instruction structure described in the instruction set document.

Turn in the output file of the assembler.

Note: You can use an online C IDE from `http://www.tutorialspoint.com/codingground.htm` for your code development, compilation and running the assembler.