Lab 9 (All Sections) Prelab: ALU Control and Data Path

Name:

Sign the following statement:

On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work

1 Objective

In this lab you will be implementing part of the processor data and control paths. You are expected to be familiar with the MIPS single cycle processor from sections 4.2, 4.3 and 4.4 of the textbook.

2 Introduction

The data path is the “brawn” of a processor, since it implements the fetch-decode-execute cycle. The general procedure data path design is to:

1. Determine the instruction classes and formats in the instruction set;

2. Design data path components and interconnections for each instruction class or format;

3. Compose the data path segments designed in Step 2 to yield a composite data path.

Simple data path components include Instruction memory (stores the current instruction), program counter or PC (stores the address of current instruction), and ALU (executes...
current instruction). Implementation of the data path for I- and J-format instructions requires two more components - a data memory and a sign extender.

In this lab, you will be adding more pieces to the data path. You have already designed the ALU and sign extender, and next you will design the data memory, ALU control, and PC logic.

3 Questions

1. The data memory stores ALU results and operands, including instructions, and has two enabling inputs (MemWrite and MemRead). Can both these inputs (MemWrite and MemRead) be active at the same time? Explain why or why not?

2. List the functional units (Instruction memory, Register file, ALU, Data memory) that will be used by the following instruction classes:
   (a) R-type
   (b) Load Word
   (c) Store Word
   (d) Branch
   (e) Jump
3. How would you extend the single cycle data path in Figure 1 to implement bne (Branch on not equal) instruction? This is similar to beq except that the branch is taken when the values are not equal. Explain your work briefly.

Fig. 1: Control Datapath
4. The processor needs to compute the next PC after each instruction. Write a verilog module for computing the next PC? Include the \texttt{j}, and \texttt{beq} cases. You may use the output of the “Control” module as shown in Figure 1.

5. How would you extend the single cycle data and control paths in Figure 1 to implement the \texttt{jr} (Jump To Register) instruction?
6. Investigate the possibility of adding a new instruction \texttt{srjr} that combines two existing instructions. For example, \texttt{srjr $ra, $sp, 16} will implement:

\begin{verbatim}
1  addi $sp, $sp, 16
2  jr $ra
\end{verbatim}

List any additional control or data paths than need to be added above what is needed in \texttt{jr}. What will be the values of \texttt{RegDst}, \texttt{ALUSrc}, \texttt{MemToReg}, \texttt{RegWrite}, \texttt{MemRead}, \texttt{MemWrite}, \texttt{Branch}, \texttt{JumpToRegister}, and any control signals you added in the previous problem and added for this problem.