John Delaney
Anthony Pasquariello
Boston University
EC413 - Spring 2016
Multicycle CPU Design

I. Introduction

The objective of this project was to design and implement a multicycle CPU using Verilog. The design of this multicycle CPU varies somewhat from a multicycle CPU that runs the MIPS instruction set. The implemented instruction set uses a different order of registers in the machine language instructions. With this new instruction set in mind, the design of the multicycle CPU was set and is shown in *Figure 1.*

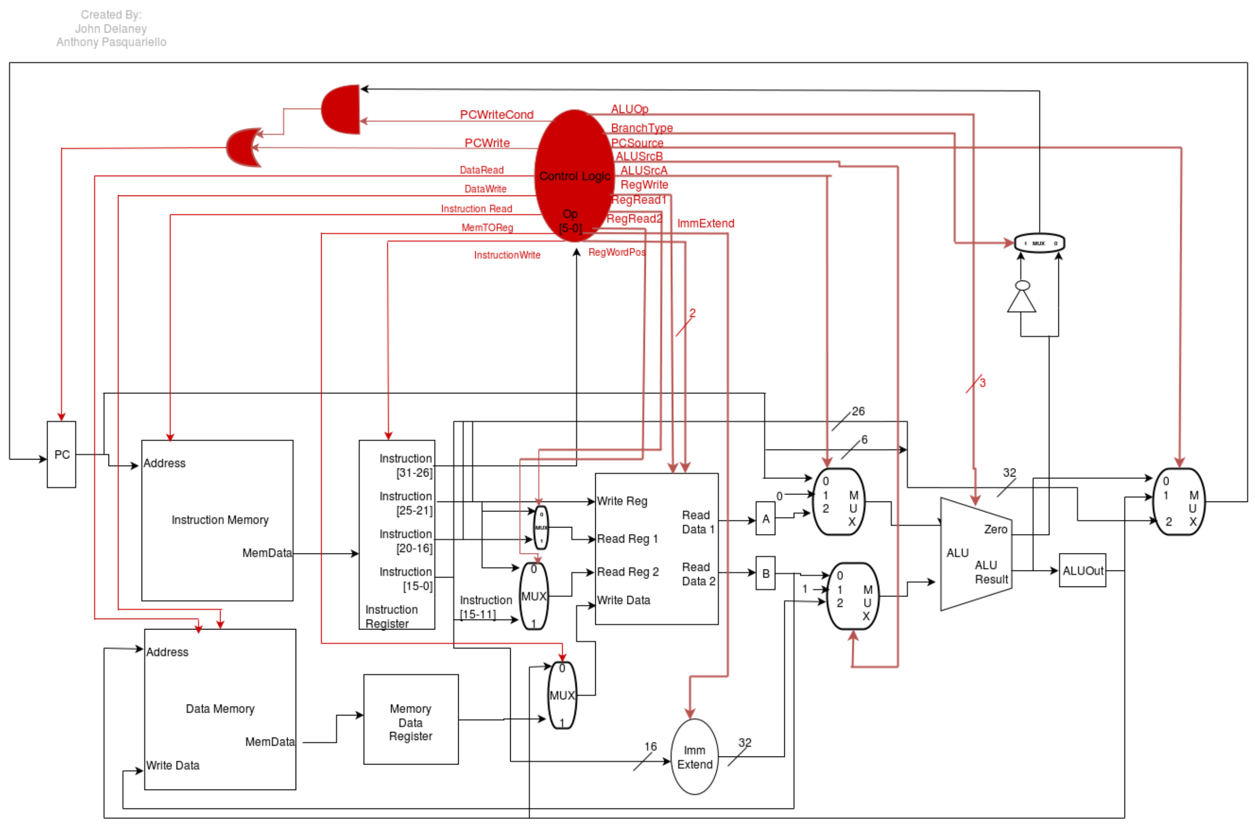

Figure 1: Final design of Multicycle CPU (see included PDF for high res image of this)

II. Design Alterations

The design of the multicycle CPU changed slightly throughout the building and testing phases. Our initial design considerations were as follows:

> *"We used the Multi-Cycle CPU from the textbook as a base. We added some functionality to our CPU that wasn't in the textbook's design. We added a 2x1 MUX that selects between Instruction[25:21] and Instruction[20:16] depending on the ALU OPCode. We also got rid of the shift left two after the sign extend. We also added a 2x1 MUX from the ALU Zero output to select between 0 and 1 for the branch type."*

By the end of our design, implementation, and testing phases the new design had some different aspects from the original design. These alterations are highlighted below:

1. We added a control signal called "RegWordPos" that controls the Register block. This control signal enables the LUI functionality, by allowing the CPU to select between three functionalities: LW, LUI, or neither. The register block was then changed to write to these specific blocks.
2. In our original design, we forgot to add in NOP functionality, so an additional state change was added from 1 to 0 if a NOP was encountered.
3. We removed the "InstrMemRead" control signal because the IMEM given doesn't use a Read Enable.
4. The ideal ALU lacked the "Zero" output, so it was added.

III. **Extra Credit Instructions**

We implemented the follow instructions that were deemed as 'extra credit' on the Spec Sheet.

1. LUI - We added a 2 bit block select enable to the register file. When it is a LI, we write to the lower 16 DFFs. If it is a LUI, we write to the upper 16 DFFs. In all other scenarios, both blocks are selected.
2. BEQ - We added a mux to the ALUZero, that selects between zero and the inverse of zero. BEQ selects the normal zero, so when it is high the contents are equal and it should branch.
3. LW - Nothing was added for load word, the functionality already existed control signals just had to be set. LWI sets ALUInput 1 to 0, whereas LW sets ALUInput1 to the data stored in the selected register.
4. SW- Implemented in the same manner as LW.