VHDL Lab 3 ISA Implementation

By, Nicholas Zahabiun and Jonathan Barnes

**Abstract:**

The goal of this lab was to implement an 8-bit ISA we created in VHDL. This lab teaches us many skills. We practiced advanced programing in VHDL. We studied methods on how data paths are transferred in processors. We also learned the integrities of delays in computer logic. In the end, we achieved a fully functioning 8-bit ISA implementation. Although an 8-bit ISA is not very practical in the real world we were able to achieve a great learning experience. Our ISA was created following the guide lines set in place to keep our implementation synthesizable.

**Division of Labor:**

This lab had multiple stages and each stage had multiple parts. The two main stages of this lab are the design of our ISA implementation on paper and the implementation of our ISA in VHDL.

The first stage is designing our RTL gate-level circuit schematics. This stage was broken down by component. Nicholas was tasked with designing the full ISA data path, ALU, and Display register. John was tasked with designing the register file and controller. Since Nicholas is neater than John he was tasked with redrawing some of the designs to be more presentable.

For the second stage, we had to implement our ISA in VHDL. For this task, we decided it would be best to remain working on our complimentary component schematics from the first stage. Nicholas was tasked with coding the ALU, Display, and test benches. John was tasked with coding the register file, controller, and full data path.

For each part of this lab we both worked together and helped each other along the way. Nicholas with John created the final lab report and submission.

**Detailed Strategy:**

This lab involved a lot of planning ahead and utilizing our RTL gate-level circuit schematics. By creating detailed schematics beforehand, we could have a strong idea of what our code should look like beforehand. This gave us a detrimental advantage as opposed to beginning the VHDL with a schematic to follow.

Creating the ALU involved two main parts. The first part was setting the does rs equal rt. The second part was the adder and subtractor. When creating the ALU I took the most concise approach possible. I did this by always running the adder, subtractor, and the does equal logic. I also did this by storing and manipulating data in 2’s compliment.

Storing the data in 2’s compliment with leading ones for negative sign extension was a very good idea. In two’s compliment, one can add both positive and negative numbers with an ordinary adder. This means for the subtraction instruction I only needed to change the sign of one input. Fortunately, in 2’s compliment the algorithm for changing a number from positive to negative is the same as the algorithm for changing a number from negative to positive. This means no extra work had to be done for cases like subtracting a negative number. When the ALU determines if we are adding numbers or subtracting them it only looks at one bit in the op code and always created an output no mater the instruction.

The if statement that sets the beq bit to 1 if the two ALU inputs equal and to zero if they do not is always running. This makes things importune for the controller that only samples the bit when the beq instruction is ran.

The display component is very special because it involves library functions. These functions are used to convert a standard logic vector to multiple intermediary values to a string to be used with the report function. The report function prints strings to the console in VHDL. This method allowed me to print the desired value with one line of code and some imported libraries. The display component also outputs the display value for the sake of having a more robust ISA implementation.