Team Pass It On

Phase Two: Register Unit, MUX, Sign-Extension, and ALU Implementation

Tennessee Technological University

ECE 4120 – Fundamentals of Computer Design

Department of Electrical and Computer Engineering

Written By: Josh Adair, Cameron Dempsey, Brantlee Garland, Hunter Goodson, and Randy Wilhoite

Table of Contents

[Modified Block Diagram: 2](#_Toc510692560)

[Phase Two Objectives: 3](#_Toc510692561)

[VHDL Implementation Elaboration: 4](#_Toc510692562)

[Synthesis Results: 5](#_Toc510692563)

[Test Bench Elaboration: 6](#_Toc510692564)

## Modified Block Diagram:

Figure 1 below shows the modified block diagram for phase two of the project. It includes the read/write unit (REG), the sign extension unit, the ALU, and the MUX for both the register unit and ALU unit. From the diagram we can see how many bits are input and output to each individual component in the diagram. We can also see the component-to-component data flow. It should be noted that this is essentially the same block diagram given with the exception of the multiplexer for the register unit. This was implemented to allow us to load the initial values into the registers using the I-type instruction addi, which allows us to add immediate values into registers.

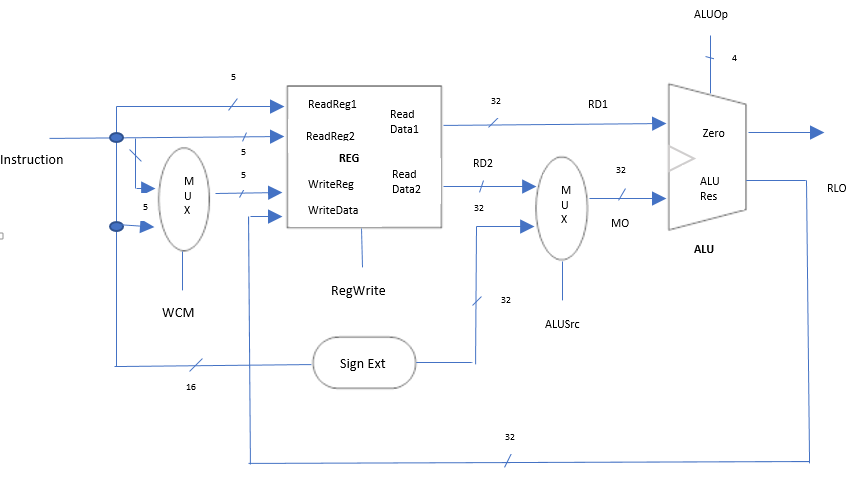


Figure 1. Modified Block Diagram

## Phase Two Objectives:

Phase two of this project required implementing multiple different units. These units are as follows: the register unit, the MUX, the sign-extension unit, and the ALU. The registers have 5 bits read/write inputs, a 32 bit write data port, and two 32 bit read data outputs. The read unit will contain thirty two 32 bit registers. The operand from the instruction memory that we implemented back in phase one is the select signal to the MUX. The write unit will have four different input signals. These signals are the clock, write, register number, and register data. Each register has three inputs, C, D, and clock. The register is enabled by the input C. The input D is the 32 bits that come from the data memory.

The multiplexer that is used for the ALU has two 32 bit inputs and one 32 bit output. A select signal is also used, which will be implemented in the final phase of this project. The sign-extension unit takes the 16th bit and makes a copy of it 16 times in order to make a 32 bit input for the ALU’s multiplexer. The ALU is a purely combinational block with two 32 bit input operands, a 4 bit input control signal, a 32 bit output, the result port and one bit output at the zero port. Once all of these units are each designed, created, tested, and then implemented together, a test bench will be required to make sure everything is in proper working order. This test bench will involve loading non-zero values into certain registers and doing some comparisons on the values in s1 and s2. Next the test bench will need to generate the required control signals and perform a set of provided MIPS instructions. This includes an ADD, SUB, and AND instruction. In addition to the required units needed to be implemented for this phase, we also chose to add an additional multiplexer for the register unit. The addition of this unit allows the initial values to be loaded in order to properly test the design.

## VHDL Implementation Elaboration:

The registers involved both a read and write unit. When completed, these two came together to create our overall register unit. The first entity is for reading and is called “ReadEntity” and contains two 5-bit inputs, the actual registers from the “WriteUnit”, and two 32-bit outputs. This unit required two multiplexers with 32 inputs that were each 32-bit. The second entity is for writing to the registers and is called “WriteUnit”. It contains both a 5 bit for selecting a register to write to, as well as a 32-bit input which contains the data to write to the register, and 32 32-bit outputs coming from the D-flip flops which act as the registers. The D-flip flops were controlled with the register number and the “Write” input ANDed together in order to control when the D-flip flops were enabled. Both the D-flip flops and the and gates were created using the generate function. We also had to create a package so that we could create and array of 32 standard logic vectors which were 32-bit in order to have the output of the D-flip flops from the “WriteUnit” and the input from the registers of the “ReadEntity”.

For the ALU we created an entity called “ALU”. This involved three inputs called A, B, and s, as well as an output called Compare and a buffer. S is the select and is used inside the architecture for ALU for deciding what F, the buffer, is set to. F is then compared and Compare is set to either 1 or 0. The multiplexer for the ALU has an entity called “mux\_21” since it’s a 2 to 1 multiplexer. Similarly to the ALU, it has three inputs, Sel, A, and B. A and B are both 32 bit inputs, while Sel is just a single bit. The output, X, is also 32 bits. Inside the architecture, X is assigned to A if Sel is equal to 1, otherwise X is assigned to B.

The sign extension unit has an entity called “SignExtendProject” and has only a single 16 bit input and single 32 bit output. The input and output are very fashionably named “input” and “output”. Inside the architecture is where the two signals, temp and extension, are used to take the 16th bit and copy it 16 times in order to obtain the 32 bit output.

## Synthesis Results:

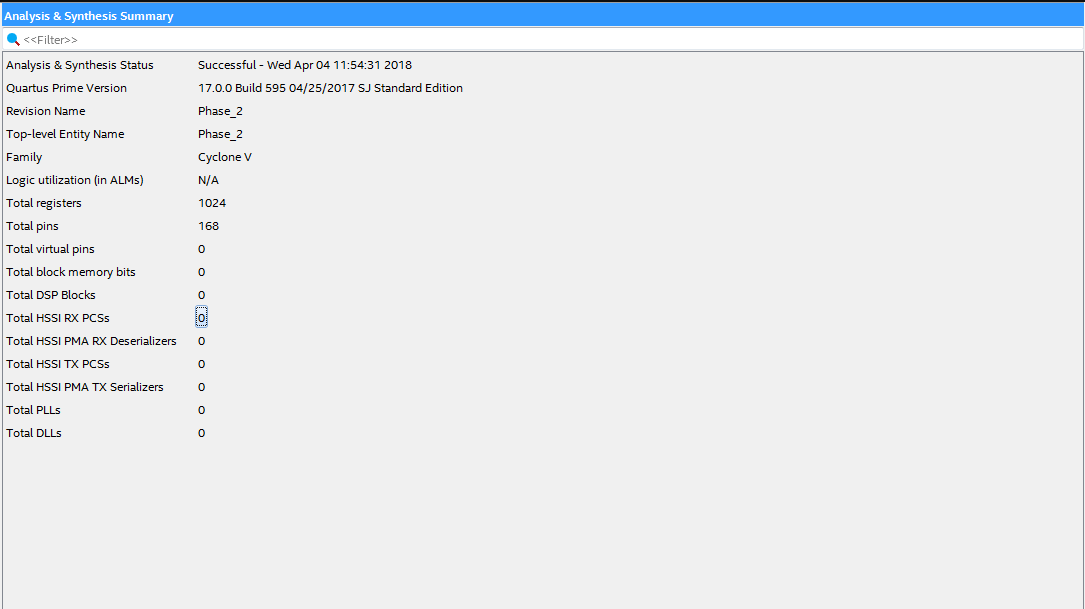


Figure 2. Synthesis results from Quartus.

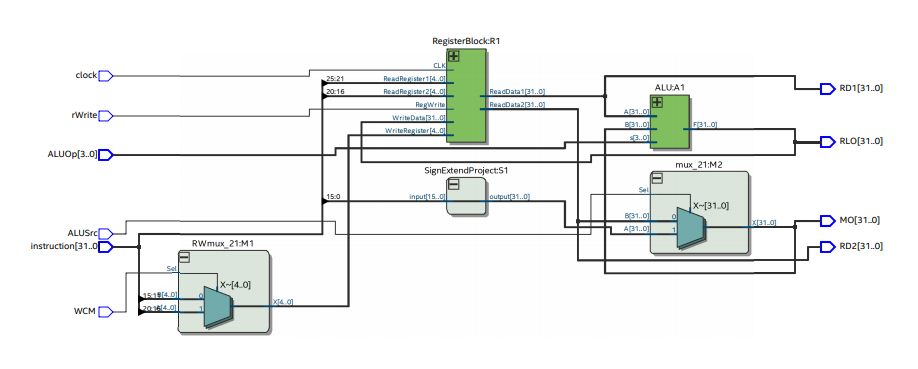


Figure 3. RTL Snapshot View

## Test Bench Elaboration:

Our completed test bench for the entirety of phase 2 is shown below. Ten signals are used, some of which were added specifically to keep track of certain outputs for the waveform. The instruction signal is used for our instructions, such as initially loading the value 5 into the register via the addi instruction.

The ALUOp signal is used to decide which operation to perform, for our purposes in this phase the operation was add. The clock signal simply starts the clock at 0. Due to using I-type instructions first, the ALUSrc signal must start at ‘1’ to allow us to read the immediate value. The WCM signal is the method of selecting the register we will be writing to, based on the type of instruction it is (‘1’ for I-type, and ‘0’ for R-type). The rWrite signal is a steady ‘1‘ since all of the instructions write to a register.

RLO, RD1, RD2, and MO were all additional outputs we used to keep track of the outputs mentioned above for the waveform. These are the 32-bit outputs for the ALU, read data 1, read data 2, and the ALU’s multiplexer respectively. Next the component is created for each signal. Finally, we get to the device under test (DUT).

Choosing to have a clock change every 10 ns, we converted the instructions to hexadecimal form and went through each instruction in 20 ns periods. The first four instructions were ADDI, immediate add, which we used to store our values. Following the addi instructions, we test the ADD, SUB, and AND instructions provided to us.



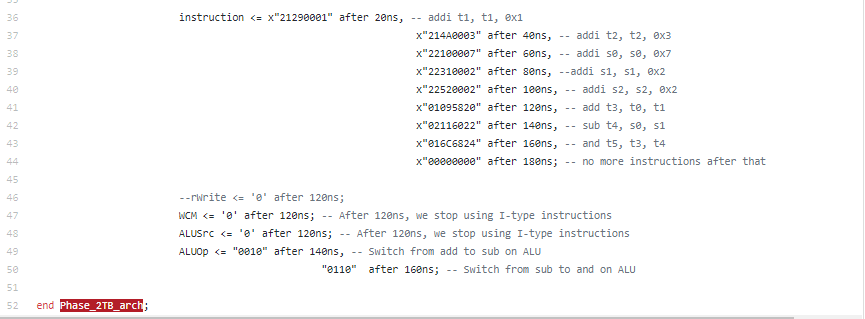
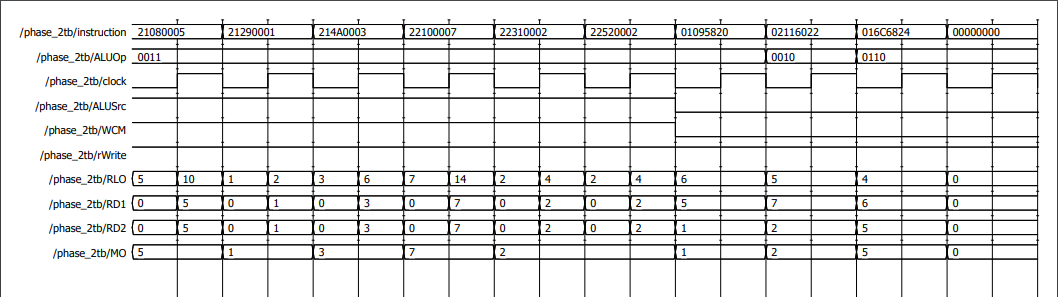


Figure 4. Test Bench used to test Phase 2

## Waveform Elaboration:

Figure 5 below shows our waveform and proves that our test bench and implementations are both correct. RD1 and RD2 are the read data outputs, and RLO and MO are, respectively, the ALU and ALU-multiplexer output. You can see that at first, both RLO and MO are 5, while RD1 and RD2 are 0. In the next half cycle, when the result is written back to the register, RD1 and RD2 both become 5, while RLO becomes 10. This continues until the 7th cycle, at around the 120 nanoseconds. Here we see RD1 and RD2 equal to 5 and 1. These are added together to provide the RLO value of 6. This is because of the ADD instruction in the test bench. The next cycle shows RD1 and RD2 being the values of 7 and 2. RLO becomes 5 because of the SUB instruction in the test bench. Lastly, 6 and 5, the RD1 and RD2 values get ANDed together to produce the value 4 for the RLO.



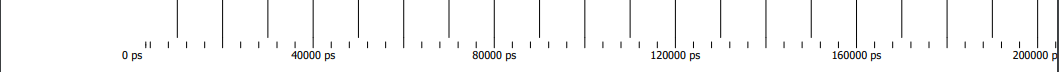


Figure 5. Waveform obtained from implementation

## Individual Contribution:

All of the members of the group met multiple times to complete this task; however, we did have certain people designated to specific tasks. Cameron handled the write unit of the register while Josh handled the read unit for the register. After finishing their side of the registers, they met together to complete the whole register unit. Hunter focused on the ALU, editing the appendix code as needed to ensure proper functionality, as well as worked with everyone to create the top-level design. Brantlee worked on the sign-extension unit for the multiplexer’s input. Randy created the ALU’s 2x1 multiplexer and created the initial draft of the report.

Though we all did have certain individual tasks, we came together as a group to ensure the overall task of phase two was completed. Once each unit was completed and individually tested, we met again so that we could all make the overall test bench together. This was a group effort where we ran into a few issues as to how we would initially load the values. This was solved by adding the register’s multiplexer so that we could use immediate instructions.

## Phase Two Conclusion:

We were able to complete phase two of the project successfully and do so in a way that met all project constraints. We designed and created the four input (three 5 bit, one 32 bit), two 32 bit output register read/write unit, our 16 bit input, 32 bit output sign-extension unit, the two 32 bit input, one 32 bit output multiplexer for the ALU, the ALU unit itself, and the additional multiplexer for the adding the immediate values to the registers in the beginning of our test bench. With all of these units combined into the TLD, our test bench with the provided ADD, SUB, and AND instructions compiled and proved our design correct.