Lab #2: Single-Cycle RISC Processor

CEG 3156: Computer Systems Design

Winter 2021

School of Electrical Engineering and Computer Science

University of Ottawa

Professors name:

Dr. Rami Abielmona

Student Name and Student #: Ritvik Johar, 300074686
Student Name and Student #: Gianluca Coletti, 300065278

Table of Contents

List of Figures

1.0 Introduction

1.1 Purpose

The purpose of this lab is to design and implement a single-cycle RISC processor. In our case, we our looking at the implementation of MIPS processor and its following functionalities:

- Memory-reference instructions (lw and sw)
- Arithmetic logic instructions (add, sub, and, or and slt)
- Control flow instruction (beq and j)

All circuits are programmed in very high speed hardware description language (VHDL) at the structural level.

1.2 Problem

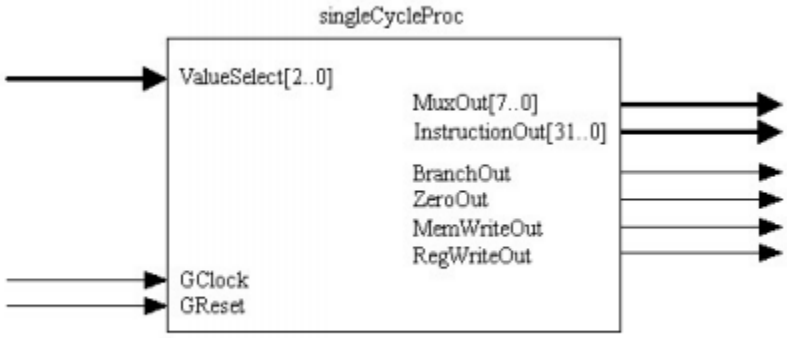

Figure 1: Single-cycle processor entity
Accessed from Laboratory #2: Single-cycle RISC processor, R. Abielmona

2.0 Functioning of the processor

To execute an instruction, the CPU needs to complete the following steps: instruction fetching, instruction decoding, the decoded instruction on arithmetic logic unit(ALU), and the memory is fetched according to the fetch address calculated by the ALU. Then, write the fetched result back to memory and so on.

- Instruction Fetch:
  → The content of the program counter(PC) is the address of the instruction.
  → Use PC content as address, access instruction memory to obtain instruction code

- Increment PC:
  → When executed sequentially; PC<=PC+4
  → When branch instruction; PC<= branch target address

Different types of instructions(R,I,J) may go through different processes because their datapath is different, which is described briefly below.

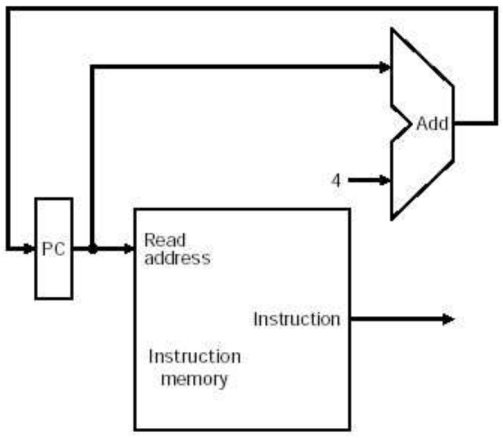

Figure 2: Fetch and Increment datapath
Accessed from Laboratory #2: Single-cycle RISC processor, R. Abielmona

2.1 R-Type Instruction Datapath

- Fetch the instruction from the instruction register(instr MEM), and the PC will be incremented.
- The contents of registers 1 and 2 are read.
- The ALU determines the operation mode depending upon the opcode, and determines the data read from the registers.
- The result of ALU is written in the register file for later use, the address of register file storage is determined by rd.

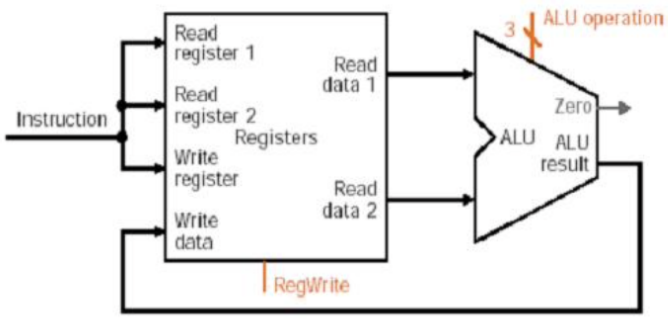

Figure 3: R-Type Instruction datapath
Accessed from Laboratory #2: Single-cycle RISC processor, R. Abielmona

2.2 I-Type Instruction Datapath

- Fetch the instruction from the instruction register(instr MEM), and the PC will be incremented.
- The contents of register 1 are read from the register file.
- The ALU adds data read from the register 1 to lower 16 bit value of sign extended instruction.
- The result of ALU operation is written into the register file, and the address of the register file is determined by rt.

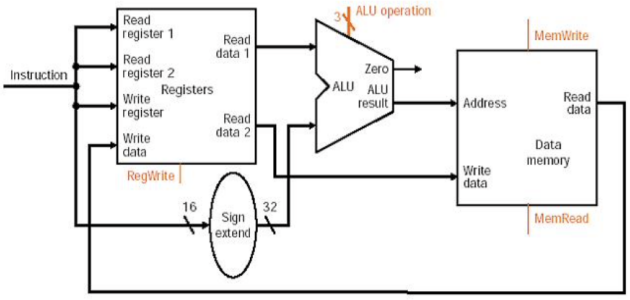

Figure 4: I-Type Instruction datapath
Accessed from Laboratory #2: Single-cycle RISC processor, R. Abielmona

2.3 J-Type Instruction Datapath

- In case of a branch instruction, the register file outputs the contents of two operands, and the ALU continues to compare the two numbers.
- The zero status flag will be asserted if the two numbers are equal.
- The control unit decides whether to take the branch or not. If the branch is taken, the branch target will include base address (PC+4) and the sign-extended offset, left shifted by 2.

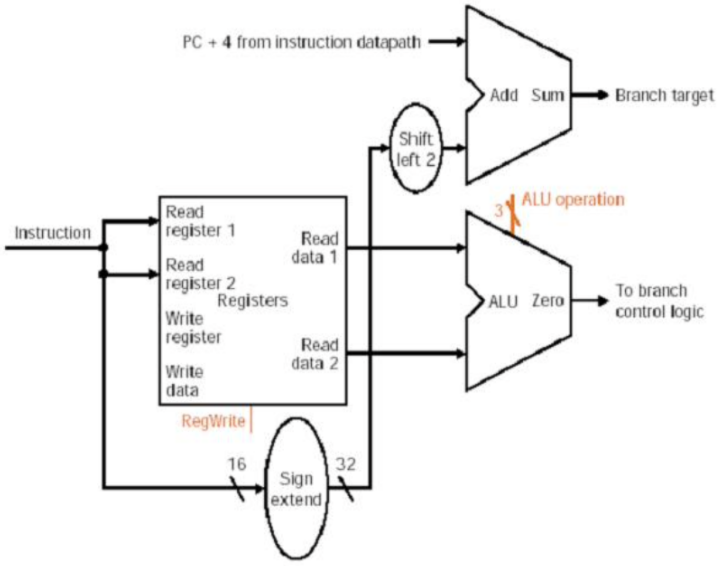

Figure 5: J-Type Instruction datapath
Accessed from Laboratory #2: Single-cycle RISC processor, R. Abielmona

3.0 VHDL Solution

The above described data paths were realized in the form of components. Only structural VHDL and RTL logic was used (with some exceptional Behavioural in memory units). To ensure the correct results, each component was simulated and verified. The components created are as follows:

- ALU
- ALU Decoder
- Control Unit
- Instruction Decoder
- Instruction Memory
- 2-to-1 Multiplexer (5 bit)
- 2-to-1 Multiplexer (32 bit)
- 4-to-1 Multiplexer (32 bit)
- PC
- Register File
- Shift Left by 2 bits (SL2)
- Select MUX
- Sign Extend
- Data Memory
- Full Adder (1 bit)
- 4 to 1 MUX (32 bit)
- Ripple Adder (8 bit)
- Ripple Adder (32 bit)
- Single cycle processor (*Top level entity*)
- Zero Extend

3.1 Calculation of Max Clock Frequency

The original processor has delay:

$$= lw\,delay = PC + instruction\,mem + register\,file + mux + ALU + data\,memory + mux + regwrite$$
$$= 1 + 10 + 10 + 2 + 15 + 10 + 2 + 10 = 60\,ns$$

Now have additional 2 mux + 1 ALU+ 1 mux = 51+ 4+ 15+2 = 81 ns

The fastest CLK Rate = 1/81ns = <u>12.35 MHz</u>

3.2 Calculation of CPU Execution Time

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

Thus,

$$\text{CPU time} = (12 \times 1) \div 12.35 \cdot 10^{3} = 9.72 \times 10^{-4} \text{ seconds}$$

3.3 Calculation of ALU Worst Path Delay

In a 32-bit [ripple carry] adder, there are 32 full adders, so the critical path (worst case) delay is:

$31 \times 2(for\ carry\ propagation) + 3(for\ sum) = 65$ gate delays

Therefore,

$$\text{Worst case delay} = 65 \times 0.01\ ns = \underline{0.65\ ns}$$

3.4 Discussion of Tool

The tools that were used for this lab were ModelSim and Quartus II; which were used in combination with the Altera Cyclone DE2 board. ModelSim is a multi-language HDL simulation environment which allows us to design and simulate the VHDL files. These VHDL files were then compiled in Quartus II and used to help build our VHDL components and create our top level entity for the design. These design files were programmed onto the FPGA and used to verify and debug our code in real time functional simulations.

3.5 Discussion of Challenging Problems

Challenging problems faced in this lab consisted of debugging of the ALU and control unit of the MIPS single-cycle processor. These problems are addressed in section 4.2.

4.0 Verification

4.1 Simulation Results

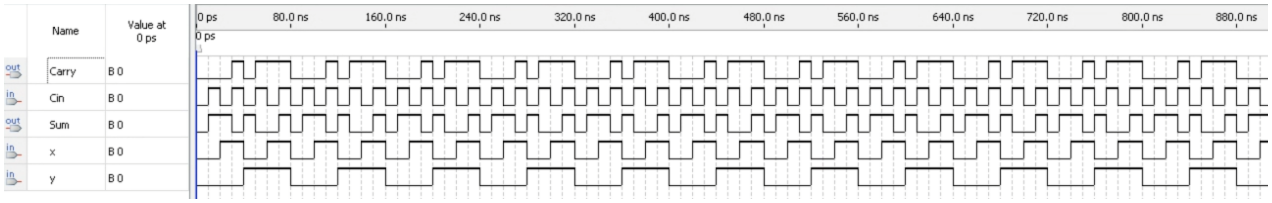

Figure 6: Simulation of full adder (1 bit).

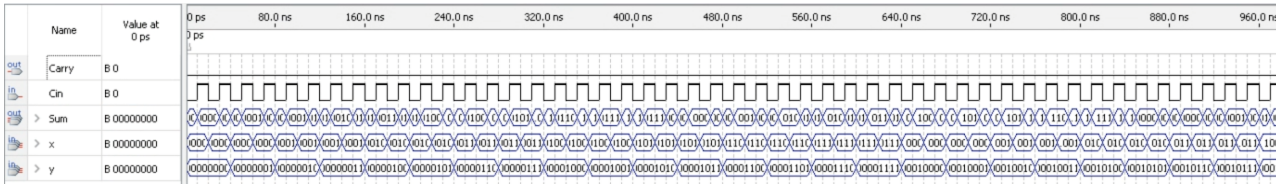

Figure 7: Simulation of ripple adder (8 bit).

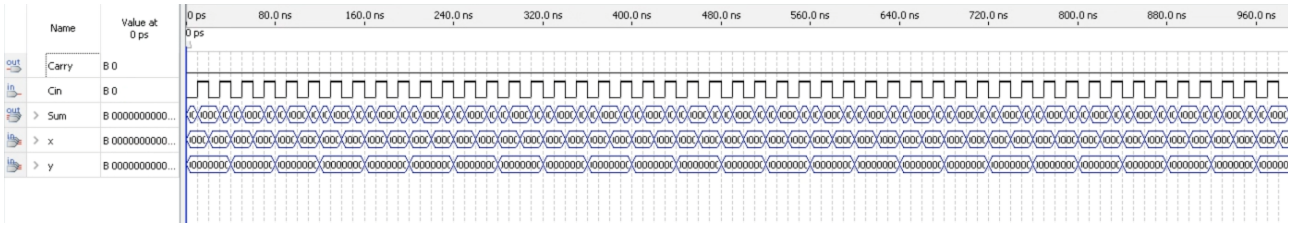

Figure 8: Simulation of ripple adder (32 bit).

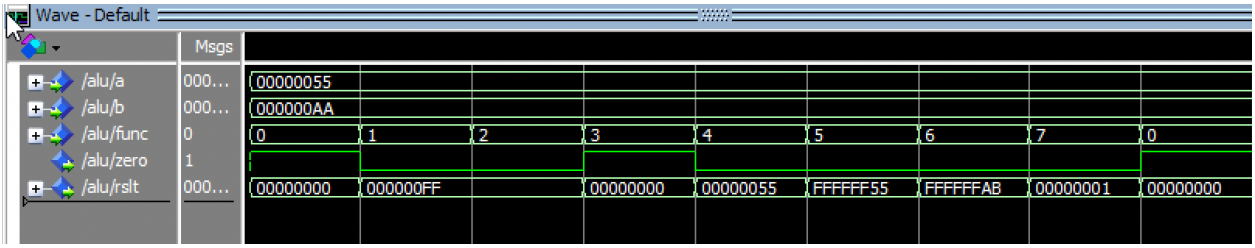

Figure 9: Simulation of ALU unit.

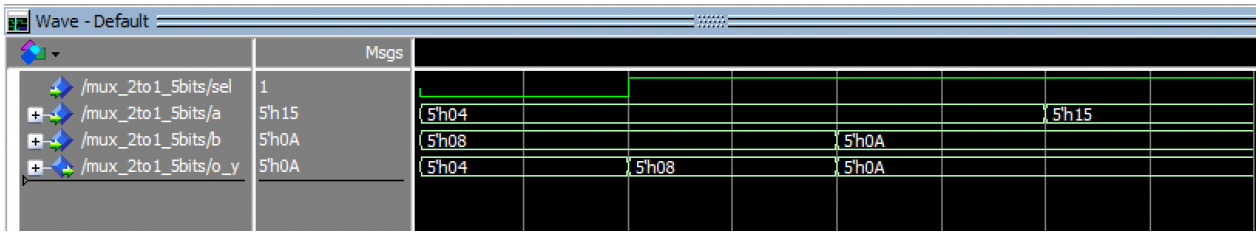

Figure 10: Simulation of MUX 2-to-1 (5 bit).

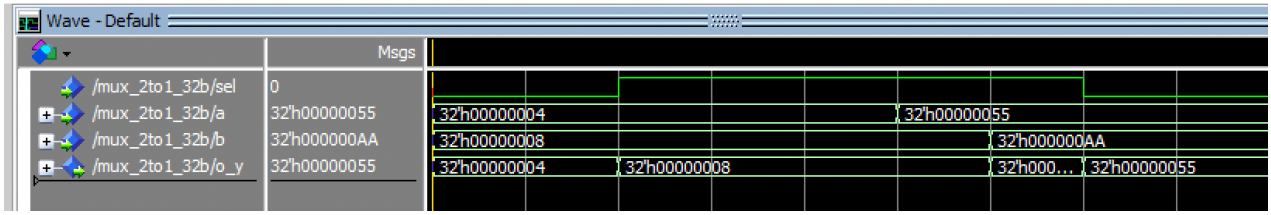

Figure 11: Simulation of MUX 2-to-1 (32 bit).

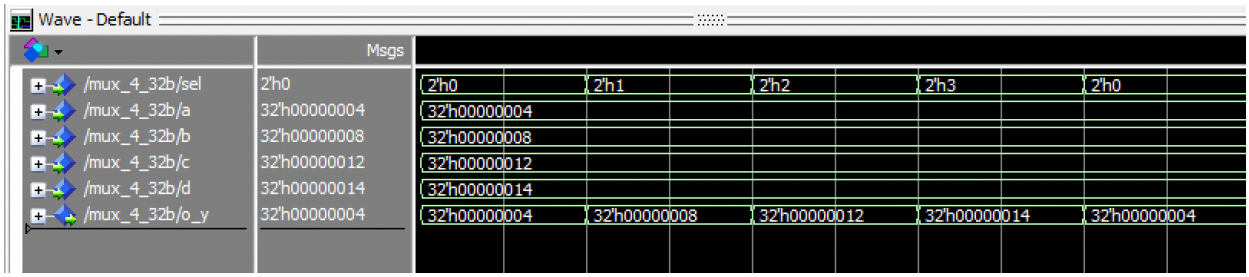

Figure 12: Simulation of MUX 4-to-1 (32 bit).

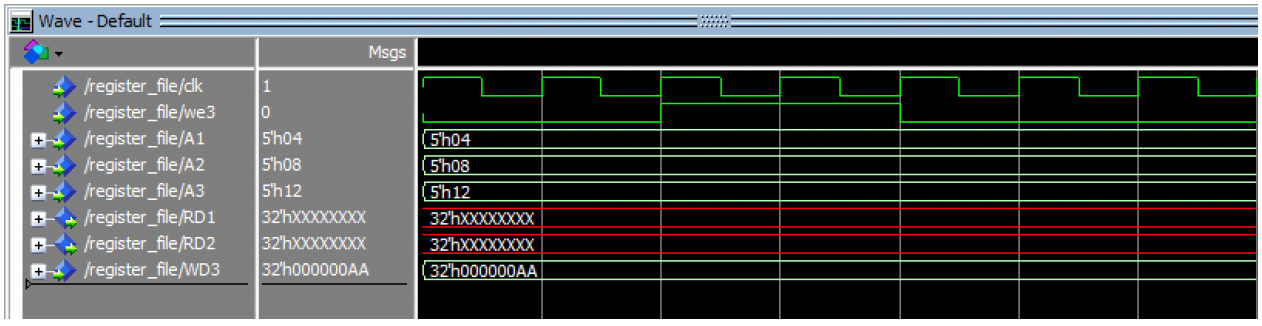

Figure 13: Simulation of register file unit.

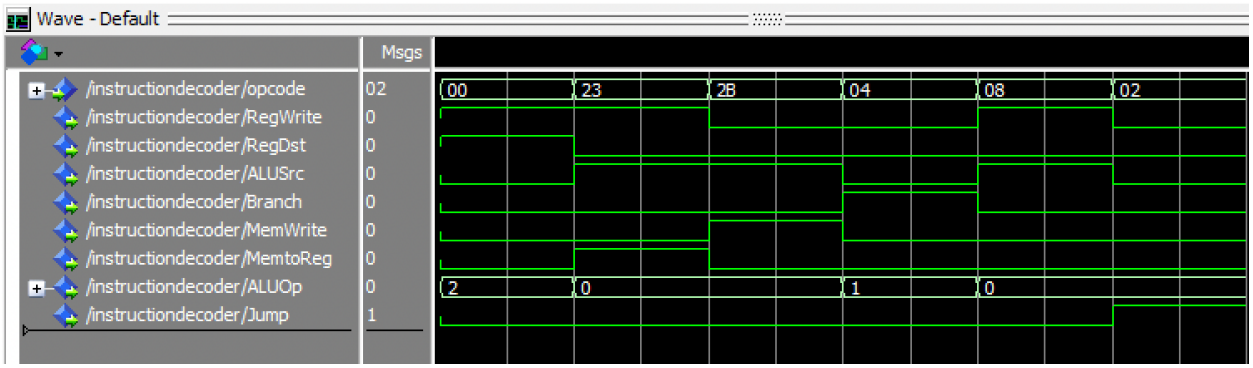

Figure 14: Simulation of instruction decoder.

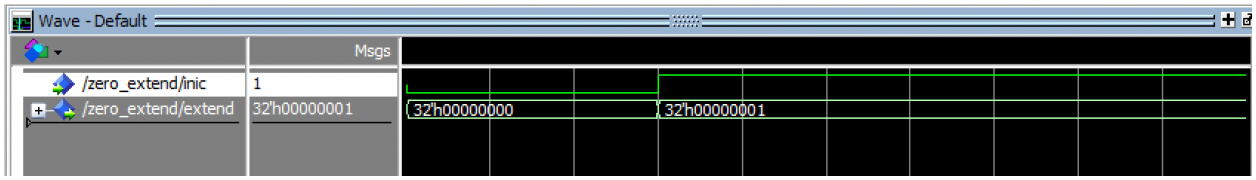

Figure 15: Simulation of zero extender unit.

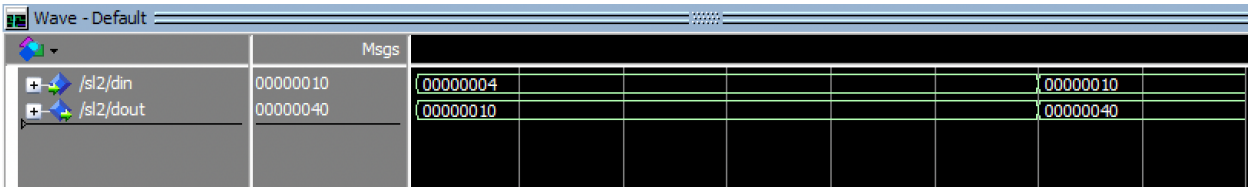

Figure 16: Simulation of arithmetic shift left by 2 bits unit.

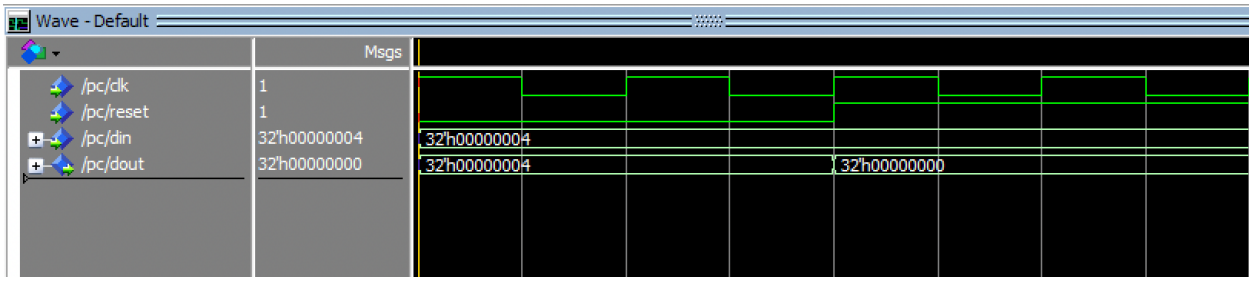

Figure 17: Simulation of PC increment component.

Figure 18: Simulation of the MIPS control unit.

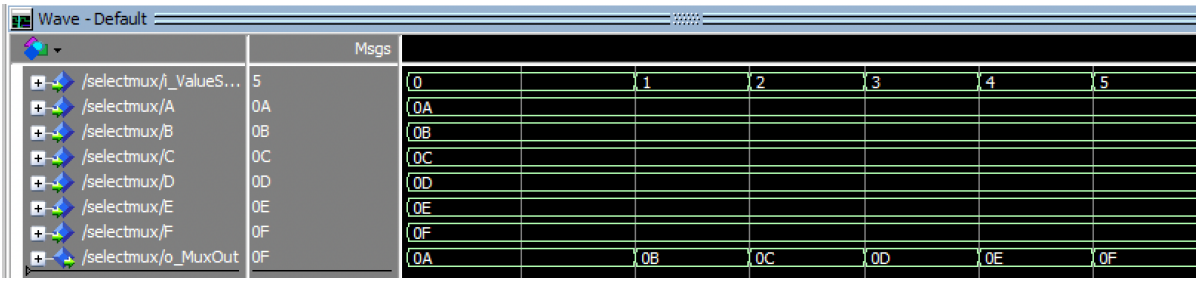

Figure 19: Simulation of SelectMux.

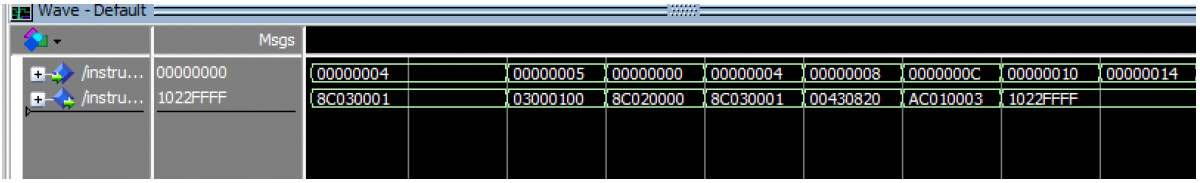

Figure 20: Simulation of instruction memory unit.

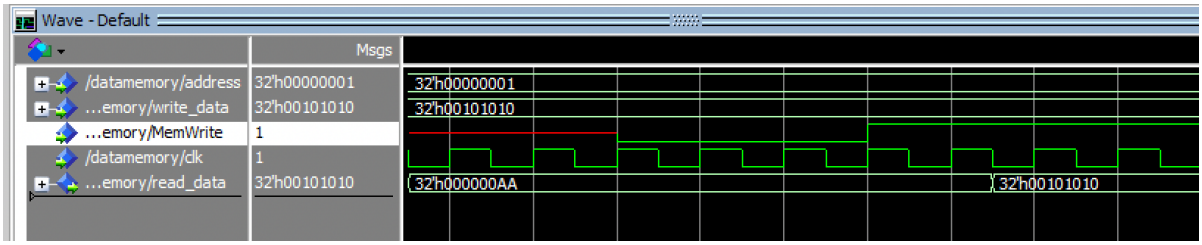

Figure 21: Simulation of data memory file.

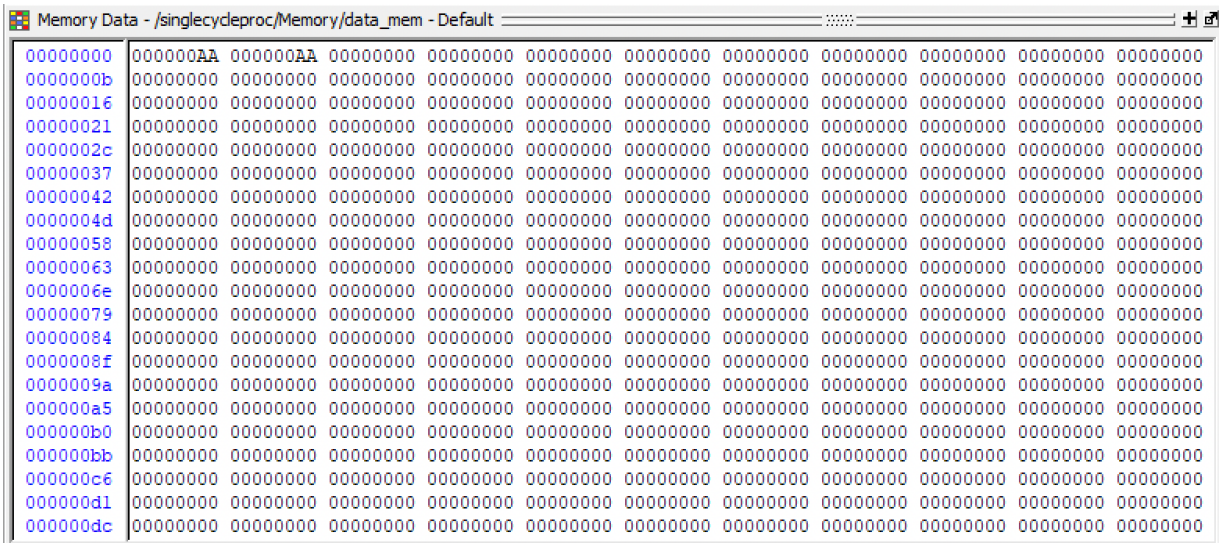

Figure 22: Data memory after instructions execution (32 bit).

4.2 Discussion

Most of the errors pertaining to our design for this lab consisted of the ALU adding the wrong two values denoted by *srca* and *srcb* for the ALU Mux. This resulted in simulations and real-time demonstration on the Altera DE2 board to show incorrect hexadecimal value *AA* by adding 55 + 55 rather than the intended 55+AA = FF (this can be seen in Figure 22). We believe this error is from a discrepancy between the design of the internal signals in our ALU unit, which causes identical values to be added instead of the values loaded into the MIPS registers, and the intended value.

5.0 Conclusion

5.1 Summary and conclusions

In conclusion, we have a better understanding of single-cycle RISC processors. By discussing the data paths of different Instruction types (R,I,J) separately we have a better idea of the MIPS instructions and the MIPS single-cycle control units functionality with these datapaths.

6.0 Prelab

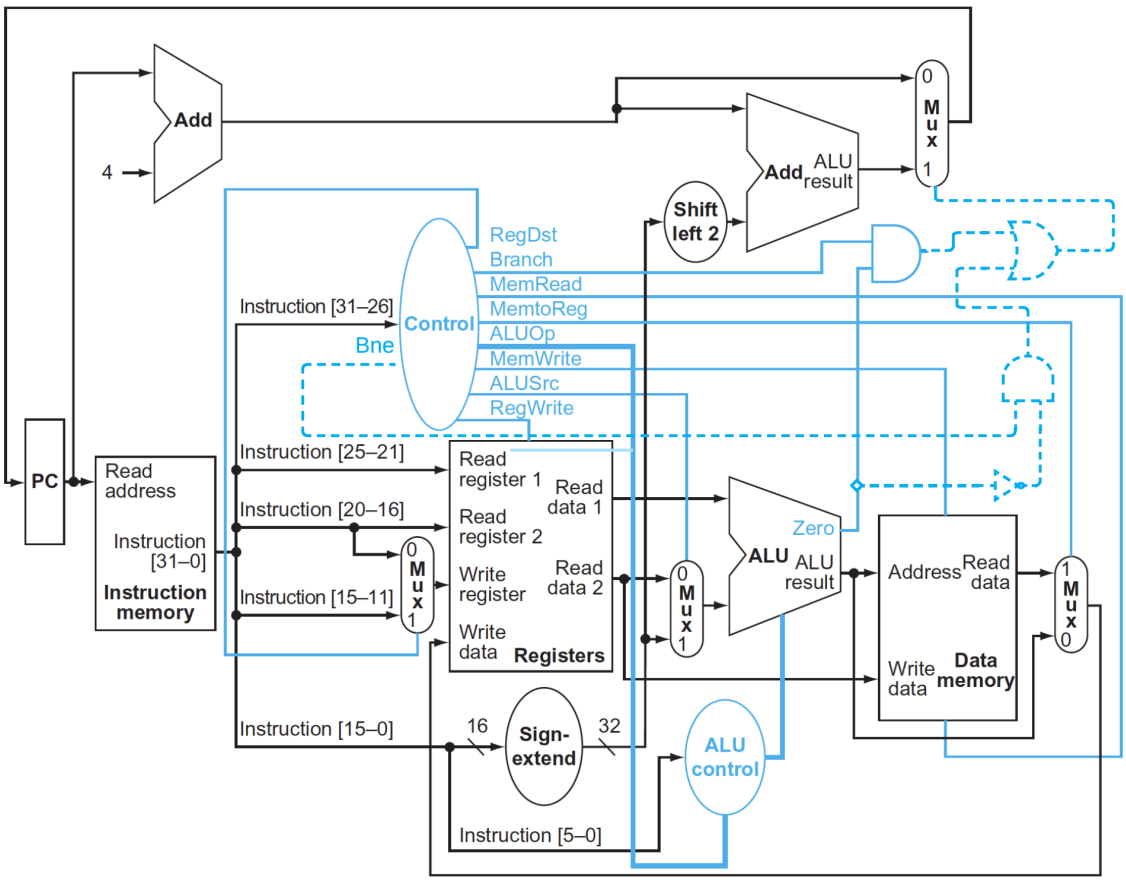

Figure 23: Revised MIPS single-cycle architecture with BNE instruction supported.

As seen in Figure 23, the modification includes an OR-gate to combine inputs (*Branch* and *Zero*) and (*BNE* and $\overline{Zero}$) which becomes the new input for the branch ALU mux select signal. The control logic for the new BNE control signal is shown in Figure 24.

$$\mathrm{BNE} = \mathrm{ALUop1\ and\ ALUop2}$$

Figure 24: Control logic for Branch Not Equal signal.