Organization of Digital Computer Lab
EECS112L/CSE 132L

Assignment 2
Single-Cycle MIPS Processor - Datapath + Control

Prepared by: Team Big Test Icicles

Student name:
Michael Herrera
Kevin Ngo
Alexander Tran
Franklin Hool

Student ID:
47378920
25092065
64197583
71351119

EECS Department
Henry Samueli School of Engineering
University of California, Irvine

January, 24, 2016

1 Design

1.1 Processor

The processor is the top-level block of the design. It is responsible for connecting the instruction memory, register file, data memory, ALU, program counter, and controller together in order to create a MIPS processor.

1.2 Program Counter

The PC is used to tell the processor what instruction to read inside of the instruction memory. After each clock cycle, it is incremented by 1 in order for the next instruction to be read.

1.3 Instruction Memory

The instruction memory is the block that contains the instructions that will be performed by the processor. It is preloaded with instructions such as **add**, **sub**, **and**, and **or**. The instructions are preloaded for testing since that is the only way for the processor to receive instructions in this design.

1.4 Register File

The register file holds an array of registers which are used for reading and writing data. It is used for every operation of this simple MIPS processor.

1.5 ALU

The ALU takes in 2 inputs, A and B, and computes either arithmetic or logical operations with them and sends the result to the output. A is a 32-bit vector from the register file, while B can either be a 32-bit vector from the register file for an R-type instruction or an immediate value from instruction memory for an I-type instruction.

1.6 Data Memory

The data memory is where data is written to and stored. The address is calculated by the ALU, and the data being written to this address is from the register file. This block is used for **lw** and **sw** operations.

1.7 Controller

The controller is needed to determine which blocks are enabled for a given instruction since different instructions may have datapaths. For example, an **add** operation datapath differs from **lw** operation datapath because add does not need to access the data memory.

2 Testing

To test the MIPS processor, we first tested the individual components if possible. We tested the ALU with a testbench that had static inputs **A_in** and **B_in** that produced values for outputs **O_out**, **Branch_out**, and **Jump_out**. The operations tested were **add**, **sub**, **and**, **or**, **xor**, **nor**, **slt**, **bltz**, **bgez**, **jump**, **beq**, **bne**, **blez**, and **bgtz**. For this assignment, we were only required to have basic computations, so the control operations only produce outputs rather than change the location of memory of the program.

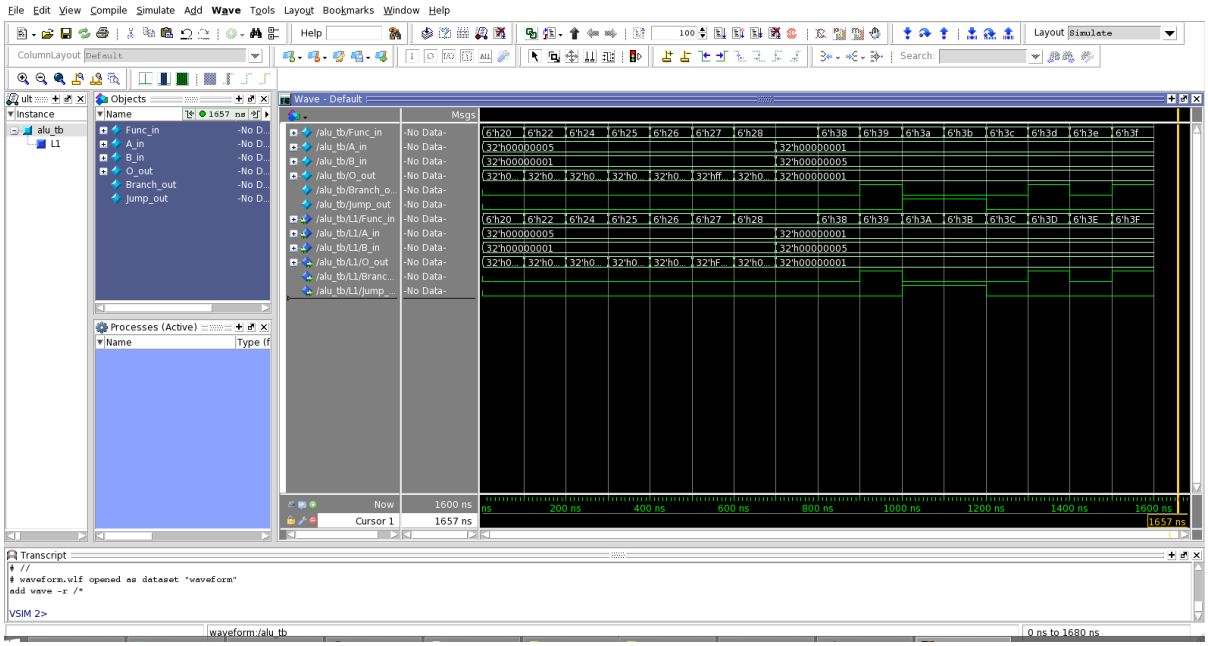

Figure 1: ALU waveform

To test the register file, we first tested it with **rst_s** set to **1** and **we** set to **0** to see if the read data and write data is equal to 0, and then tested it with **rst_s** set to **0** and **we** set to **1** to see that the read data from the given read address was written to the given write address in the testbench.

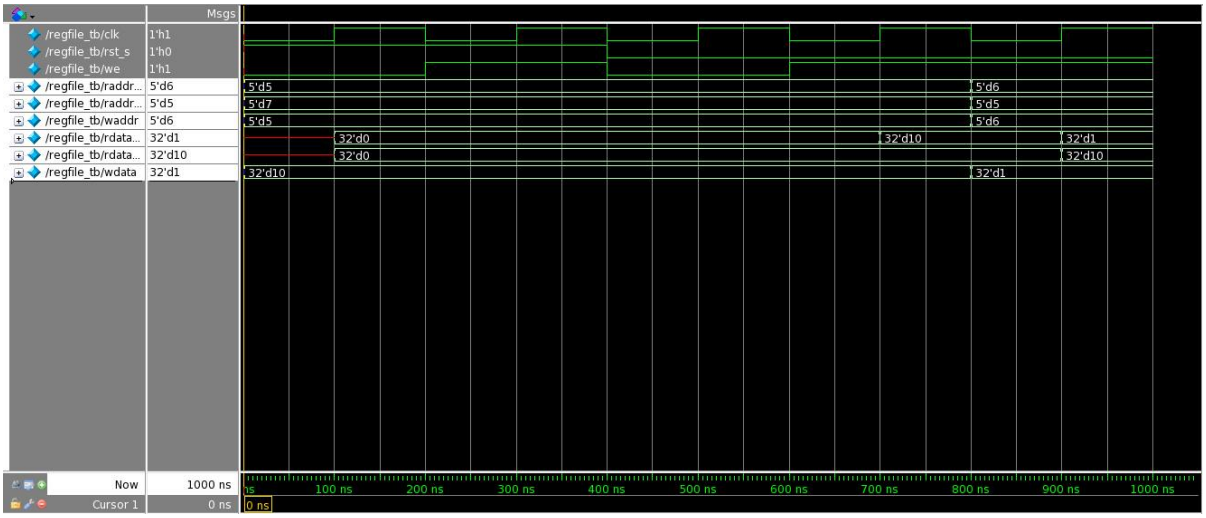

Figure 2: Register file waveform

To test the program counter, we simply just had the clock run for a few cycles and watched the output value increase by 1 for each cycle.

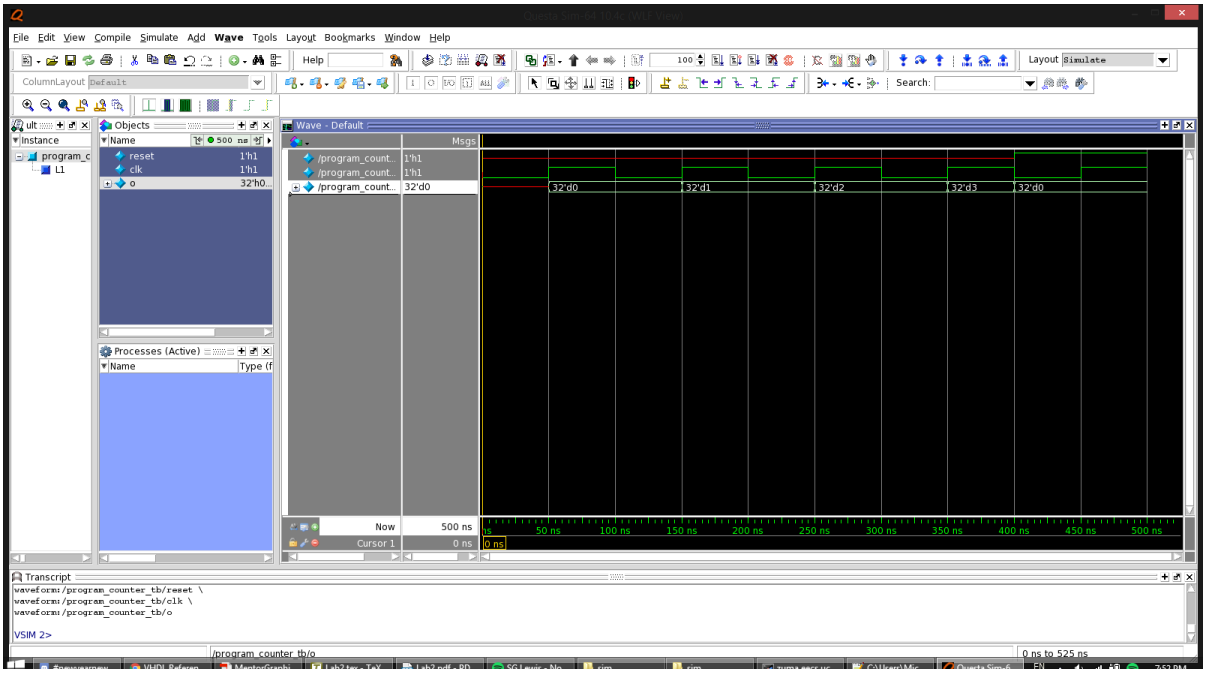

Figure 3: Program counter waveform

To test the controller, we gave it opcodes for each case and looked at the resulting enable bit outputs. We tested opcodes **000000**, **100011**, and **101011**.

Finally, after extensive portmap debugging, we connected all of the blocks together to form the processor. It computes preloaded instructions from the instruction memory, which are chosen by the

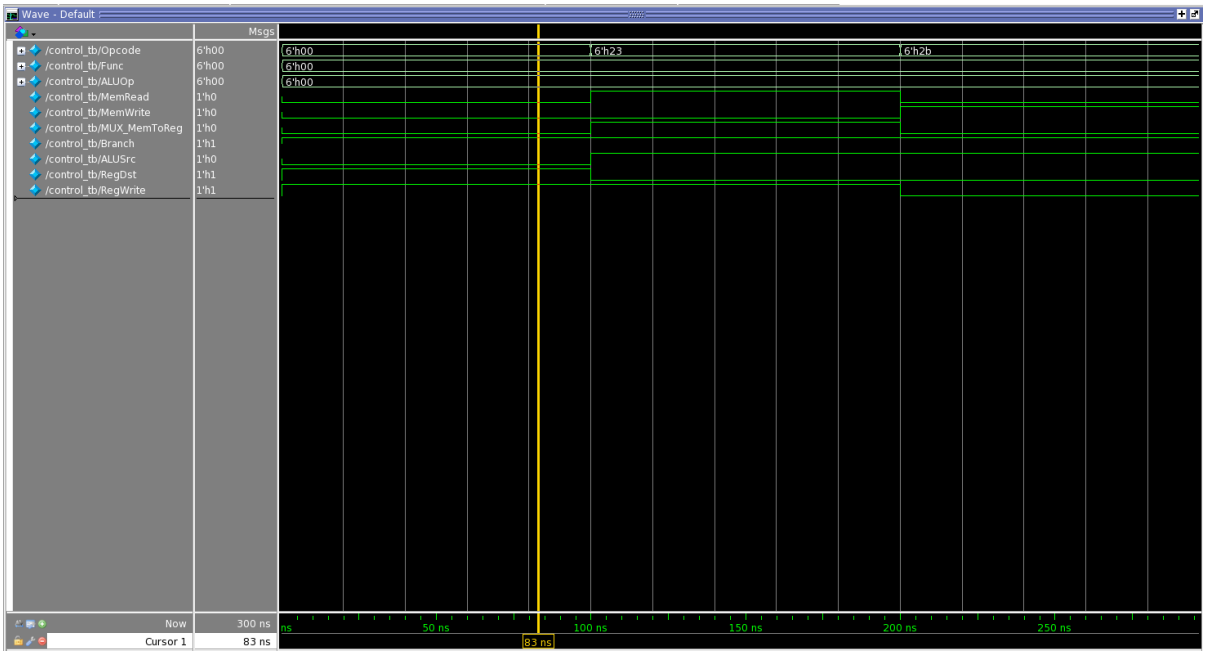

Figure 4: Controller waveform

incrementing program counter. The instruction is then decoded and moved through the register file, where the data continues through its corresponding datapath. The preloaded instructions, in order, are:

1. add R3, R1, R2
2. sub R3, R1, R2
3. and R3, R1, R2
4. or R3, R1, R2
5. xor R3, R1, R2
6. nor R3, R1, R2
7. slt R1, R2, R3
8. sw R2, 20(R1)
9. lw R2, 20(R1)

3 Bugs

Our testing for the processor is not extensive due to time constraints. Therefore, we are not able to discern any bugs at its current state. If more time was available, the processor could be further tested by comparing every resulting bit to its expected bit at every output to verify that they match up completely to the MIPS architecture's logic.

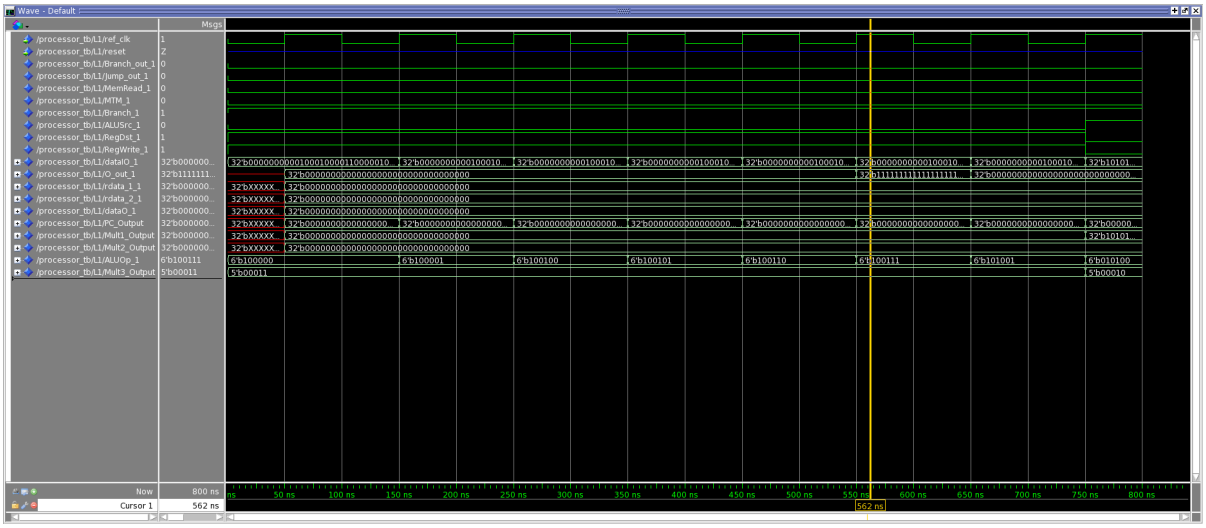

Figure 5: Processor waveform