Organization of Digital Computer Lab
EECS112L/CSE 132L

Assignment 5
Single-cycle MIPS - Complete

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

March 13, 2016

1 Design

This lab continues off of the previous assignment's single-cycle MIPS processor. In addition to the instructions from the last assignment, the datapath now supports unconditional jumps, conditional branches, and pipeline implementations.

Supported Instructions:
Branches - `BEQ, J`
Memory operation - `LW, SW`
Arithmetic - `ADD, SUB, AND, OR, XOR, NOR, SLT`

1.1 Processor

The processor is the top-level block of the design. It is responsible for connecting the program counter, instruction memory, controller, register file, ALU, data memory, hazard unit, pipeline registers, and all other supplementary components together in order to create the 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. With the addition of branching and jumping instructions, the PC's value can now be changed in order to access specific instructions in the instruction memory. For the pipeline implementation, the PC can be stalled when needed and is controlled by the hazard unit.

1.3 Instruction Memory

The instruction memory (ROM) is the block that contains the instructions that will be performed by the processor.

1.4 Controller

The controller is needed to determine which blocks are enabled for a given instruction since different instructions may have datapaths. It sends signals for multiplexer operations, regwrite, and memwrite. With the addition of I-Type instructions, it also decodes opcodes and translates them into functions for the ALU.

1.5 Register File

The register file holds an array of registers that are used for reading and writing data.

1.6 ALU

The ALU performs various arithmetic and logic functions with its ”A” and ”B” inputs and sends the result to the output. To support the new I-Type instructions, a 2-to-1 multiplexer is added to the ”B” input of the ALU. It is controlled by the controller, which decides whether to use the value from the register file or an immediate value in the instruction.

1.7 Data Memory

The data memory (RAM) is where data is written to and stored. It is a 512-word x 32-bit array whose size can be reconfigured at instantiation time using `dsize`.

1.8 Hazard Unit

The hazard unit controls whether to use the real control values from the controller or '0' when hazards are detected, such as data hazards, load-use hazard, and branch hazards. In the event of a hazard, depending on the type of hazard, the hazard unit will either stall the process or forward data. Data hazards, such as Read-After-Write or Write-After-Read, will cause forwarding in the pipeline implementation and stalling will occur in load-use hazard and branch hazards.

1.9 Pipeline Registers

With each clock cycle, data from the pipeline register is read first and then data coming into the register is stored so that it is read in the next cycle, imitating a pipeline implementation. When stall occurs in the fetch stage or decode stage, the value in the pipeline register is preserved when a `NOP|` instruction is executed.

2 Testing

To test the MIPS processor, we first tested the individual components if possible. The new pipeline registers were simulated individually to demonstrate the logic. When stall is off, the register will hold the input value present at the rising edge of the clock. Alternatively, it will read out the data stored in the register at any given time. The pipeline register for the fetch/decode was used for testing since it was the simplest one of the four.

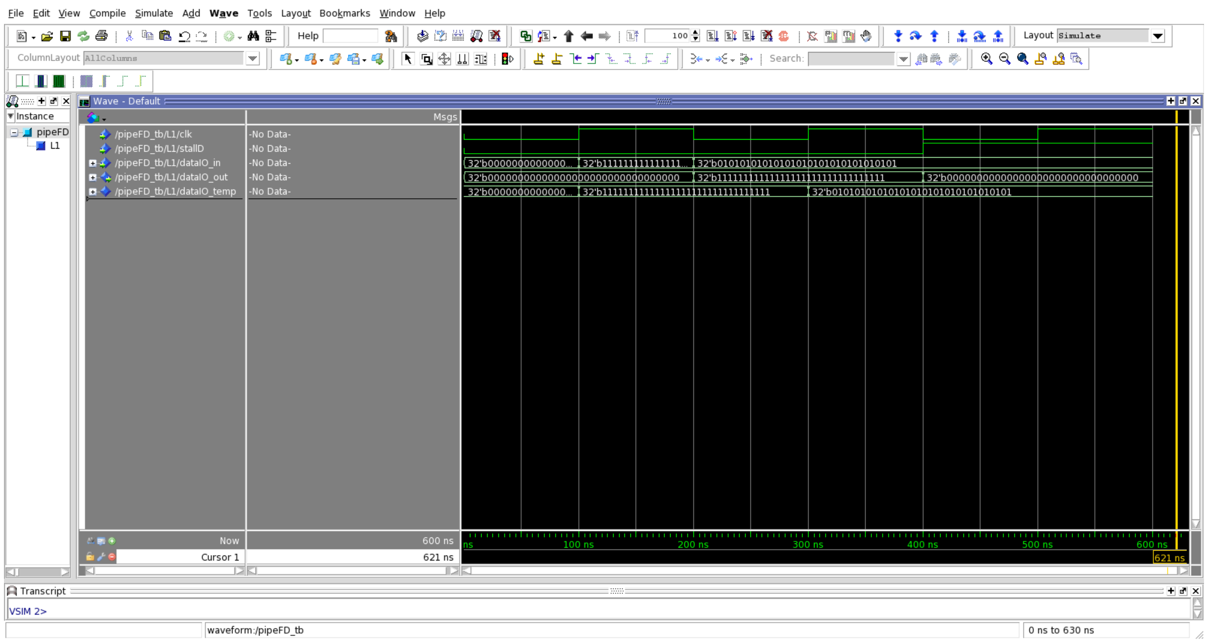
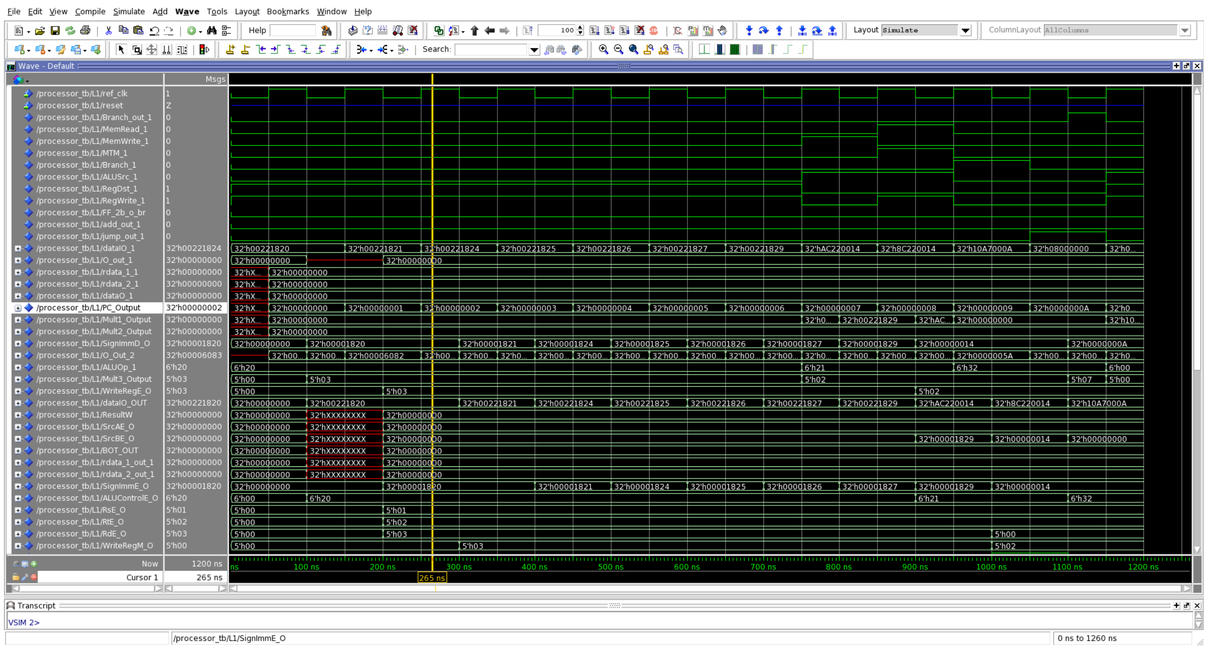


Figure 1: pipeFD waveform

To test the processor, we preloaded the ROM with the instructions given in the assignment specificiations. Those instructions are listed below:

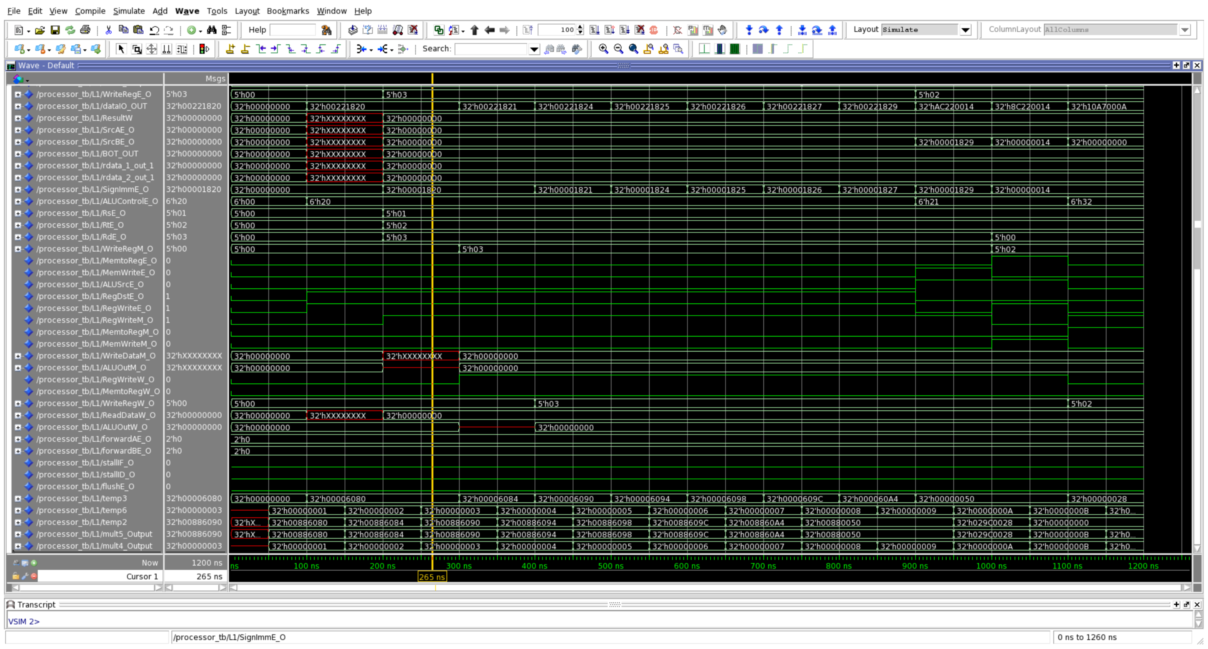
1. `00000000001000100001100000100000 - add R3 R1 R2`
2. `00000000001000100001100000100001 - sub R3 R1 R2`
3. `00000000001000100001100000100100 - and R3 R1 R2`
4. `00000000001000100001100000100101 - or R3 R1 R2`
5. `00000000001000100001100000100110 - xor R3 R1 R2`
6. `00000000001000100001100000100111 - nor R3 R1 R2`
7. `00000000001000100001100000101001 - slt R1 R2 R3`
8. `10101100001000100000000000010100 - sw R2 20(R1)`
9. `10001100001000100000000000010100 - lw R2 20(R1)`
10. `00010000101001110000000000001010 - beq R5 R7 10`
11. `00001000000000000000000000000000 - j 0`



Signal	Msgs
/processor_tb/L1/ref_clk	1
/processor_tb/L1/reset	Z
/processor_tb/L1/Branch_out_1	0
/processor_tb/L1/MemRead_1	0
/processor_tb/L1/MemWrite_1	0
/processor_tb/L1/MTM_1	0
/processor_tb/L1/Branch_1	0
/processor_tb/L1/ALUSrc_1	0
/processor_tb/L1/RegDst_1	1
/processor_tb/L1/RegWrite_1	1
/processor_tb/L1/FF_2b_o_br	0
/processor_tb/L1/add_out_1	0
/processor_tb/L1/jump_out_1	0
/processor_tb/L1/dataIO_1	32'h00221824
/processor_tb/L1/O_out_1	32'h00000000
/processor_tb/L1/rdata_1_1	32'h00000000
/processor_tb/L1/rdata_2_1	32'h00000000
/processor_tb/L1/dataO_1	32'h00000000
/processor_tb/L1/PC_Output	32'h00000002
/processor_tb/L1/Mult1_Output	32'h00000000
/processor_tb/L1/Mult2_Output	32'h00000000
/processor_tb/L1/SignImmD_O	32'h00001820
/processor_tb/L1/O_Out_2	32'h00006083
/processor_tb/L1/ALUOp_1	6'h20
/processor_tb/L1/Mult3_Output	5'h03
/processor_tb/L1/WriteRegE_O	5'h03
/processor_tb/L1/dataIO_OUT	32'h00221820
/processor_tb/L1/ResultW	32'h00000000
/processor_tb/L1/SrcAE_O	32'h00000000
/processor_tb/L1/SrcBE_O	32'h00000000
/processor_tb/L1/BOT_OUT	32'h00000000
/processor_tb/L1/rdata_1_out_1	32'h00000000
/processor_tb/L1/rdata_2_out_1	32'h00000000
/processor_tb/L1/SignImmE_O	32'h00001820
/processor_tb/L1/ALUControlE_O	6'h20
/processor_tb/L1/RsE_O	5'h01
/processor_tb/L1/RtE_O	5'h02
/processor_tb/L1/RdE_O	5'h03
/processor_tb/L1/WriteRegM_O	5'h00
Now	1200 ns
Cursor 1	265 ns

```
VSIM 2>
```

Figure 2: Processor waveform 1



Signal	Msgs
/processor_tb/L1/WriteRegE_O	5'h03
/processor_tb/L1/dataIO_OUT	32'h00221820
/processor_tb/L1/ResultW	32'h00000000
/processor_tb/L1/SrcAE_O	32'h00000000
/processor_tb/L1/SrcBE_O	32'h00000000
/processor_tb/L1/BOT_OUT	32'h00000000
/processor_tb/L1/rdata_1_out_1	32'h00000000
/processor_tb/L1/rdata_2_out_1	32'h00000000
/processor_tb/L1/SignImmE_O	32'h00001820
/processor_tb/L1/ALUControlE_O	6'h20
/processor_tb/L1/RsE_O	5'h01
/processor_tb/L1/RtE_O	5'h02
/processor_tb/L1/RdE_O	5'h03
/processor_tb/L1/WriteRegM_O	5'h00
/processor_tb/L1/MemtoRegE_O	0
/processor_tb/L1/MemWriteE_O	0
/processor_tb/L1/ALUSrcE_O	0
/processor_tb/L1/RegDstE_O	1
/processor_tb/L1/RegWriteE_O	1
/processor_tb/L1/RegWriteM_O	1
/processor_tb/L1/MemtoRegM_O	0
/processor_tb/L1/MemWriteM_O	0
/processor_tb/L1/WriteDataM_O	32'hXXXXXXXX
/processor_tb/L1/ALUOutM_O	32'hXXXXXXXX
/processor_tb/L1/RegWriteW_O	0
/processor_tb/L1/MemtoRegW_O	0
/processor_tb/L1/WriteRegW_O	5'h00
/processor_tb/L1/ReadDataW_O	32'h00000000
/processor_tb/L1/ALUOutW_O	32'h00000000
/processor_tb/L1/forwardAE_O	2'h0
/processor_tb/L1/forwardBE_O	2'h0
/processor_tb/L1/stallF_O	0
/processor_tb/L1/stallD_O	0
/processor_tb/L1/flushE_O	0
/processor_tb/L1/temp3	32'h00006080
/processor_tb/L1/temp6	32'h00000003
/processor_tb/L1/temp2	32'h00886090
/processor_tb/L1/mult5_Output	32'h00886090
/processor_tb/L1/mult4_Output	32'h00000003
Now	1200 ns
Cursor 1	265 ns

```
VSIM 2>
```

Figure 3: Processor waveform 2

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.

4 Synthesis

Area = 3750.139959
Power = 2.95e+03
Maximum Frequency = 2.778

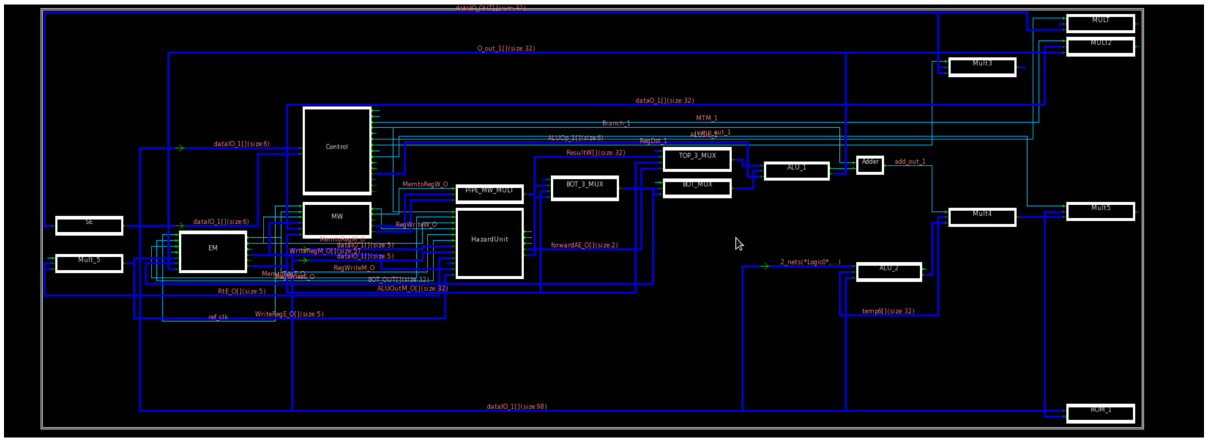


Figure 4: Processor synthesis