32-bit Single Cycle MIPS CPU

Elijah Malaby Tanmay Kotha Daniel Sawyer Conner Wulf

December 7, 2017

1 Design and Implementation

1.1 Data Path

The design of the CPU is a basic MIPS design with a subset of the instruction set. We implemented 16 instructions, 6 of which are R-type, 5 are I-type, and 3 are J-type. The general data path starts with the PC which feeds into instruction memory and gets the instruction, then goes into the register file and reads the source register(s) used for the instruction. Next it depends on the instruction type, with R-type both values come from the registers and go through the ALU to compute the instruction operation and then stores it into . With I-type, one of the operands is an immediate value instead of from the second register like the R-type's instructions. In both R and I types, they write the result of the ALU into the designated destination register except for the sw (store word) which saves into data memory. With J-type instructions, after the the result of the ALU (the new PC address) is computed, its sent back around to the PC at which point the PC control unit sends the proper selector for the mux to accept the branched or jump PC address. The control units control which path it will take based upon the instruction and feeds the appropriate control signal to where they need to go.

1.2 Components

The components for the whole CPU include the Register File, the PC module, the Instruction memory, the data memory, the ALU, the ALU Control, the PC control, and the Main control unit.

1.3 Control

There are three units for controlling the CPU: The Main Control unit, the ALU Control, and the PC Control. The Main Control unit handles interpreting the opcodes from the current instruction and enabling all of the relevant signals so that the rest of the processor will work as expected as described by the MIPS spec. Some of the signals of the Main Control unit are also fed into the PC Control unit (Jump, BEQ, BNE) and the ALU unit (ALUSrc, ALUOp). The PC Control unit handles getting the next address that the PC register should save for the next cycle of the processor, which will in turn dictate which instruction in instruction memory that is extracted and fed into the remaining components of the processor. The PC Control unit takes the signals of Jump, BNE, and BEQ, as well as ALU's Zero signal and computes from it a selection value to activate a channel in its multiplexer to select whether the PC will increment, jump, or branch. The ALU Control unit takes the signals ALUSrc and ALUOp (alternatively if the instruction handles immediate mode instructions, it will retrieve the opcode from the func bits in the original instruction retrieved from instruction memory) from the Main control unit, and based on those signals, retrieves two data pieces A and B, where A is always from a register, and B could be either from a register or directly from the immediate value embedded in the instruction as selected by ALUSrc, and then based on the ALUOp value the ALU Operation is then selected and then performed. Depending on the operation performed, there are additional signals that are created such as the ALU Zero signal.

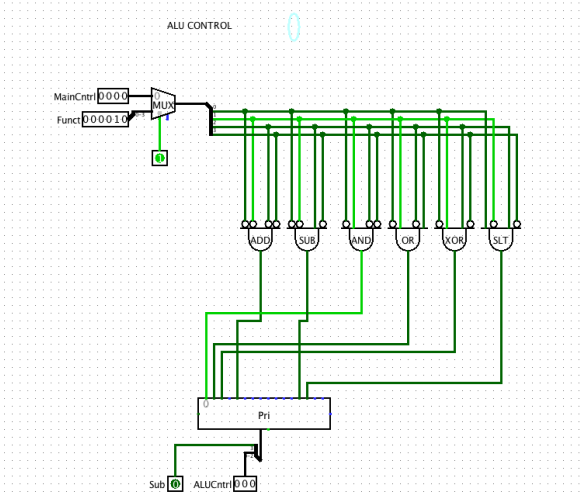
1.4 Implementation Details

There are three units for controlling the CPU: The Main Control unit, the ALU Control, and the PC Control. The Main Control unit handles interpreting the opcodes from the current instruction and enabling all of the relevant signals so that the rest of the processor will work as expected as described by the MIPS spec. Some of the signals of the Main Control unit are also fed into the PC Control unit (Jump, BEQ, BNE) and the ALU unit (ALUSrc, ALUOp). The PC Control unit handles getting the next address that the PC register should save for the next cycle of the processor, which will in turn dictate which instruction in instruction memory that is extracted and fed into the remaining

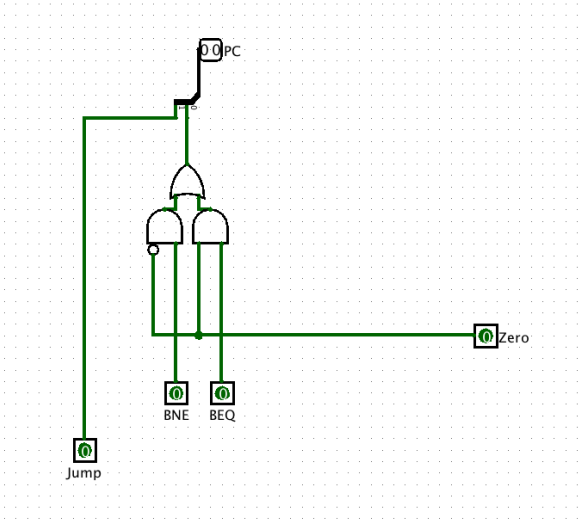
components of the processor. The PC Control unit takes the signals of Jump, BNE, and BEQ, as well as ALU's Zero signal and computes from it a selection value to activate a channel in its multiplexer to select whether the PC will increment, jump, or branch. The ALU Control unit takes the signals ALUSrc and ALUOp (alternatively if the instruction handles immediate mode instructions, it will retrieve the opcode from the func bits in the original instruction retrieved from instruction memory) from the Main control unit, and based on those signals, retrieves two data pieces A and B, where A is always from a register, and B could be either from a register or directly from the immediate value embedded in the instruction as selected by ALUSrc, and then based on the ALUOp value the ALU Operation is then selected and then performed. Depending on the operation performed, there are additional signals that are created such as the ALU Zero signal.

1.5 Component Circuits and Overall Datapath

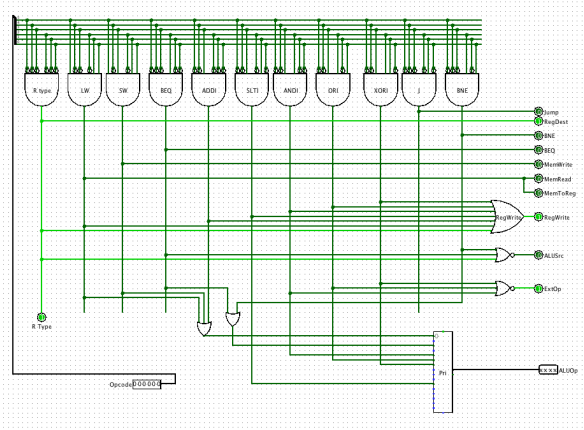
ALU Controller



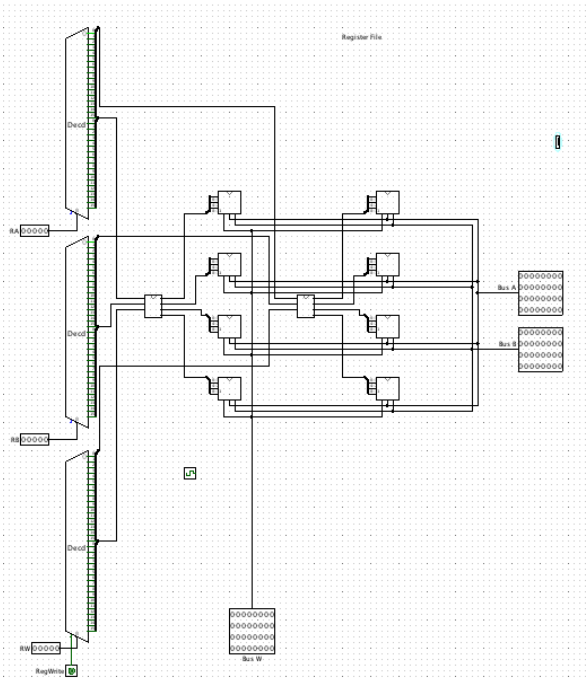
PC Controller



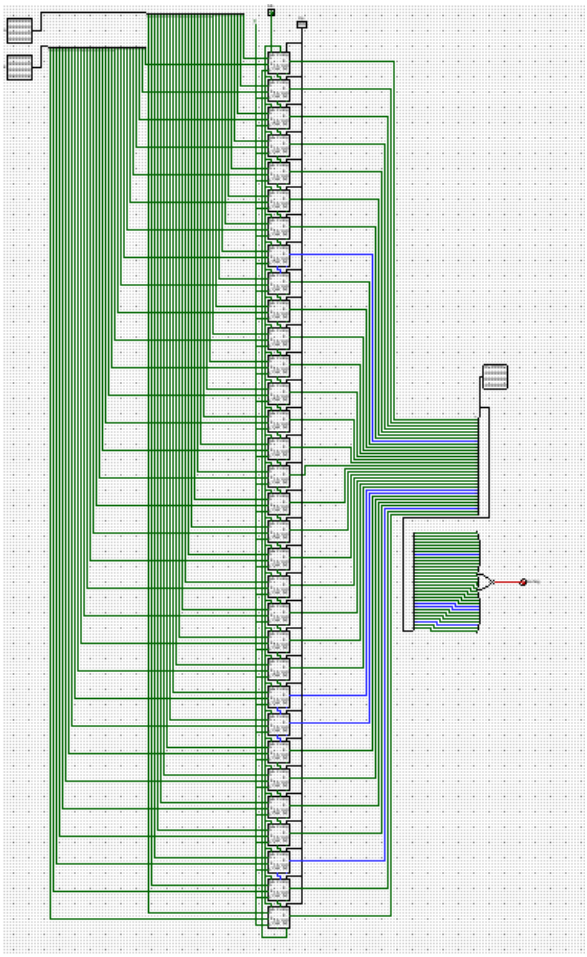
Main Controller



Register File



ALU



1.6 Main Control Table

See Table 1. Some shorthands are used for the columns (the signals): RD, for RegDest, MW for MemWrite, MR for MemRead, MTR for MemToReg, and RW for RegWrite.

1.7 ALU Control Table

See Table 2.

1.8 PC Control Table

See Table 3.

1.9 Logic Equations

Here are the logic equations from our program.

$$\begin{aligned} Jump &= J \\ RegDest &= RType \\ BNE &= BNE \\ BEQ &= BEQ \\ MemWrite &= SW \\ MemRead &= LW \\ MemToReg &= LW \\ RegWrite &= RType + LW + ADDI + SLTI \\ &\quad + ANDI + ORI + XORI \\ ALUSrc &= \sim (BNE + BEQ + RType) \\ ExtOp &= \sim (XORI + ORI + ANDI) \end{aligned} \tag{1}$$

2 Simulation and Testing

2.1 Carry out the simulation of the processor developed using Logisim.

Shown in the video

2.2 Describe the test programs that you used to test your design with enough comments describing the program, its inputs, and its expected output.

For testing we used three different programs. The first two were from this website and are included in our zip file: `https://cs.nyu.edu/courses/fall15/CSCI-UA.0436-001/lecture10.html`. The first two programs rigorously check the following eight instructions: and, or, sub, lw, sw, slt, beq, add. We included hex text files with preloaded memory files that are compatible with logisim. The third program we wrote by hand and loaded into MARS and then created the hex text file then loaded it into logisim. The third program tested all of the other instructions which include: xor, addi, slti, andi, ori, xori, bne. All of the instructions worked flawlessly without error. We test if the instruction works. Then we made sure the branch instructions do not branch when they shouldn't. We tried diligently to break all of the instruction but thankfully, we could not. We include the program testings in the video included in the zip file.

Table 1: Main Control Table

	Jump	RD	BNE	BEQ	MW	MR	MTR	RW	ALUSrc	ExtOp	ALUOp
add	X	1	X	X	X	X	X	X	X	0	0000
sub	X	1	X	X	X	X	X	1	1	0	0010
and	X	1	X	X	X	X	X	1	1	0	0100
or	X	1	X	X	X	X	X	1	1	0	0101
xor	X	1	X	X	X	X	X	1	1	0	0110
slt	X	1	X	X	X	X	X	1	1	0	0101
addi	X	X	X	X	X	X	X	1	0	1	0000
slti	X	X	X	X	X	X	X	1	0	1	1010
andi	X	X	X	X	X	X	X	1	0	0	0100
ori	X	X	X	X	X	X	X	1	0	0	0101
xori	X	X	X	X	X	X	X	1	0	0	0110
lw	X	X	X	X	X	1	1	1	0	0	0000
sw	X	X	X	X	1	X	X	X	0	0	0000
beq	X	X	X	1	X	X	X	X	1	0	0010
bne	X	X	1	X	X	X	X	X	1	0	0010
j	1	X	X	X	X	X	X	X	0	0	X

Table 2: ALU Control Signals Table

Function	ALU Op Code	Sub Flag (fed in from ALU Control)
and	000	0
or	001	0
xor	010	0
add	100	0
sub	100	1
slt	101	1

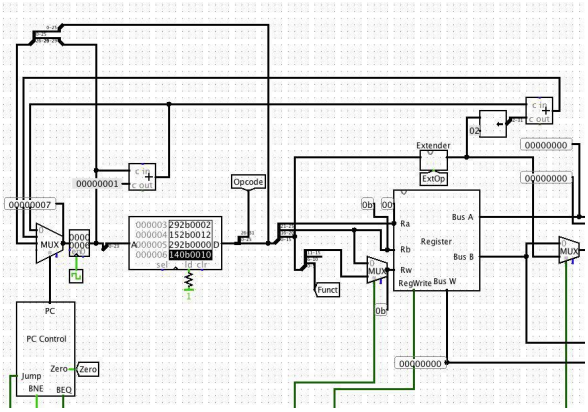
Table 3: PC Control Signals Table

Op	Op Code	Jump	Bne or Beq
PC++	00	0	0
Branch	01	0	1
Jump	10	1	0

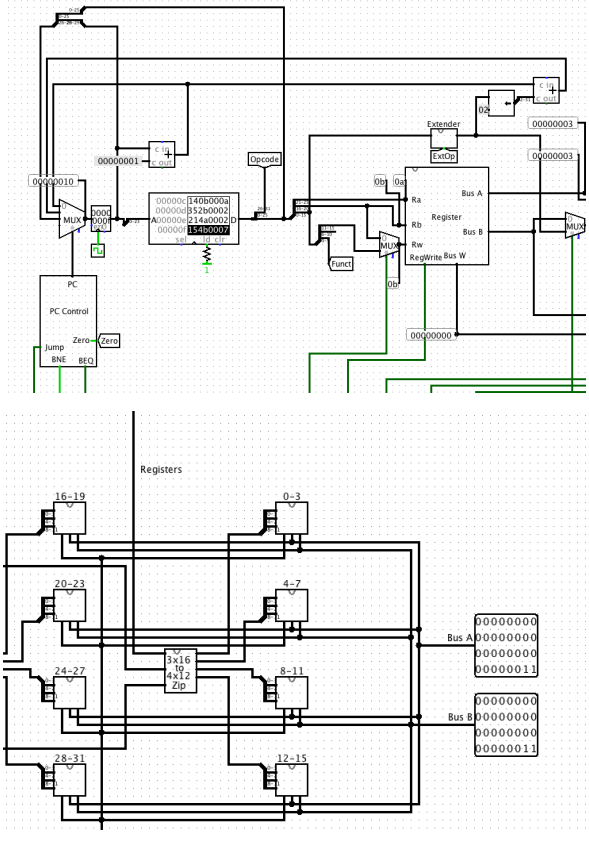
2.3 List all the instructions that were tested and work correctly.

Based on the results of the three tests, two of which are shown in the video included with this project, as well as the auxiliary tests included, we are able to show that all of the instructions have been implemented and operate correctly.

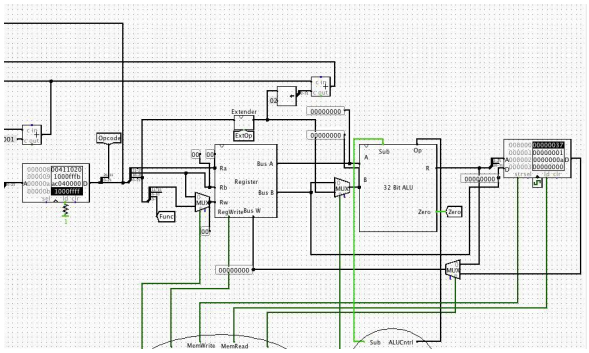
2.4 Provide snapshots of the Simulator window with your test program loaded and showing the simulation output results



This instruction is testing slti, which should place 0 in `$t3` and prevent the following bne from branching.



This instruction is testing ori, which should place 0 in `$t3` and prevent the following bne from branching.



This is a successful test of the sw instruction, shown at the end of program 1.

3 Teamwork

Throughout the semester, the team has been performing most of the design process remotely via an IM messaging service, and sharing the logisim design files between each other in a peer to peer repository. Throughout the early month of November, most of the basic design of the processor was heavily researched and discussed remotely, due to various scheduling issues arising in each individual member's time. The development of the ALU circuit module was developed by both Daniel and Conner, while the Register file was developed by Elijah and Tanmay. The full debugging process for both components were delayed until both components were brought up to basic feature parity to be tested together. But for the most part, individually the sub teams for the components were split up into circuit designer and circuit debugger roles. One person would handle designing the circuit and explaining the schematic to their partner, while the partner would go through and attempt to get the circuit to work while relaying any issues arising from the circuit, while the designer fixed any issues that arose. Once part 1 of the processor was complete, in order to complete part 2, the team opted for in person meetings on campus in order to fully collaborate on what parts of the processor datapath should be implemented. The first phase of the part 2 meetings were implementing the remainder of the components, namely the memory, control units, and the PC register logic. Elijah was able to quickly get the memory and PC register up and running. From there the control unit work was split up between the ALU Control developed by Daniel and Tanmay, and the PC control and the Main control unit was developed by Conner and Elijah. From there Control unit debugging was handled by Daniel, and Processor testing was handled by Elijah, and the ALU was tested by Tanmay.