Ande Chen and Sunwoo Park
EC413
Lab 6: Single Cycle CPU
November 5, 2021

Lab 6 Report

Task Descriptions

Task 1: For task 1, we simulated the unmodified given files in a Vivado project with the CPU testbench to generate a waveform for the following instruction sequence:

000000_00000_00010_00001_00000_10_0000	(ADD R1, R0, R2)
000000_00100_00100_01000_00000_10_0010	(SUB R8, R4, $4)
000000_00101_00110_00111_00000_10_0101	(OR R7, R5, R6)
101011_00000_01001_00000_00000_00_1100	(SW R9, 12(R0))
100011_00000_01100_00000_00000_00_1100	(LW R12, 12(R0))
000100_00000_00000_00000_00000_00_0010	(BEQ R0, R0, 2)

From the waveform, we found that the output was

Task 2: SLT

We implemented the shift-less-than (SLT) operation into the CPUs ALU by adding a condition to the 'ALU_Control' module that checks if the 6-bit instruction code is 101010 (2A in hex), which corresponds to the SLT according to the MIPS Instruction Encoding table. If this condition is true, it sets the 3-bit function output to 5, which is the next open function code that is not already in use in the ALU. Next, in the ALU, we added a condition that checks if the input function from 'ALU_Control' is 5 and sets the output to 1 if input 1 is less than input 2 and 0 otherwise.

Task 3: ADDI

To implement add immediate (ADDI) into the ALU, we added a condition to the 'Control' module that checks if the 6-bit opcode is 001000, which corresponds to ADDI in the MIPS Instruction Encoding table. We set the Control output signals to the following:

- ALUOp is 2'b10 to make the ALU perform addition
- ALUSrc is 1 to use the immediate instead of the value of Read Data 2
- MemtoReg is 0 to send the ALU output back to the register file
- RegDst is 0 to select Read Register 2 as the register to update with the ALU output
- RegWrite is 1 to enable writing to registers in the Register File
- MemRead, MemWrite, and Branch are 0, because they are not used for ADDI

Task 4: J

Following the text's implementation of jump (J), we added a condition to the 'Control' module that checks if the 6-bit opcode is 000010, which corresponds to J in the MIPS Instruction Encoding table. We create the Control output signal for Jump and set it to 1. All the other Control outputs are 0 because they are not used. Then, we create an additional instance of the 'Shift Left 2' module to shift bits 25-0 of the instruction, which corresponds to the offset, left by 2 bits. The first 4-bits of the current program counter PC_out is appended to the front of the 26-bit shifted output, and 00 is appended to the end of the shifted output. This new 32-bit sequence is a possible new PC input. It is passed into a new mux that selects this new PC input if the Jump select signal is 1 and the PC input that is calculated by the Branch/PC Adder logic if the select is 0. The output of the mux is connected back to the Program Counter module.

Task 5: BNE

To add instruction (Branch if Not Equal) BNE, we added a condition to the 'Control' module that checks if the 6-bit opcode is 000101, which corresponds to BNE in the MIPS Instruction Encoding table. We created the Control output signal for BNE and set it to 1. Then, we connected the zero_flag wire, inverted the signal, and fed the BNE signal and the inverted zero_flag signal into an AND gate. Then, we sent the output of the AND gate into an OR gate with the output of the Branch control signal and the zero_flag. The output of the OR gate is then fed into the PC_input_MUX (as labeled in our diagram below) as the new PCSrc signal. This implementation allowed us to incorporate only a few logic gates and use the existing Branch signal in order to execute BNE in the single-cycle CPU.

Task 6: LUI

To add instruction (Load Upper Immediate) LUI, we added a condition to the 'Control' module that checks if the 6-bit opcode is 001111, which corresponds to LUI in the MIPS Instruction Encoding table. We created the Control output signal for LUI and set it to 1. Then, we took bits 15 through 0 of the instruction sequence and appended 16 bits of 0 to the end, effectively creating a 32-bit load-immediate bus. Then, we fed the existing 32-bit sign-extended immediate bus and our 32-bit load-immediate bus into a MUX, with the LUI control bit as the select. Then, we connected the output of the MUX to the signals that the sign-extended immediate originally fed into (as seen on our modified diagram below).

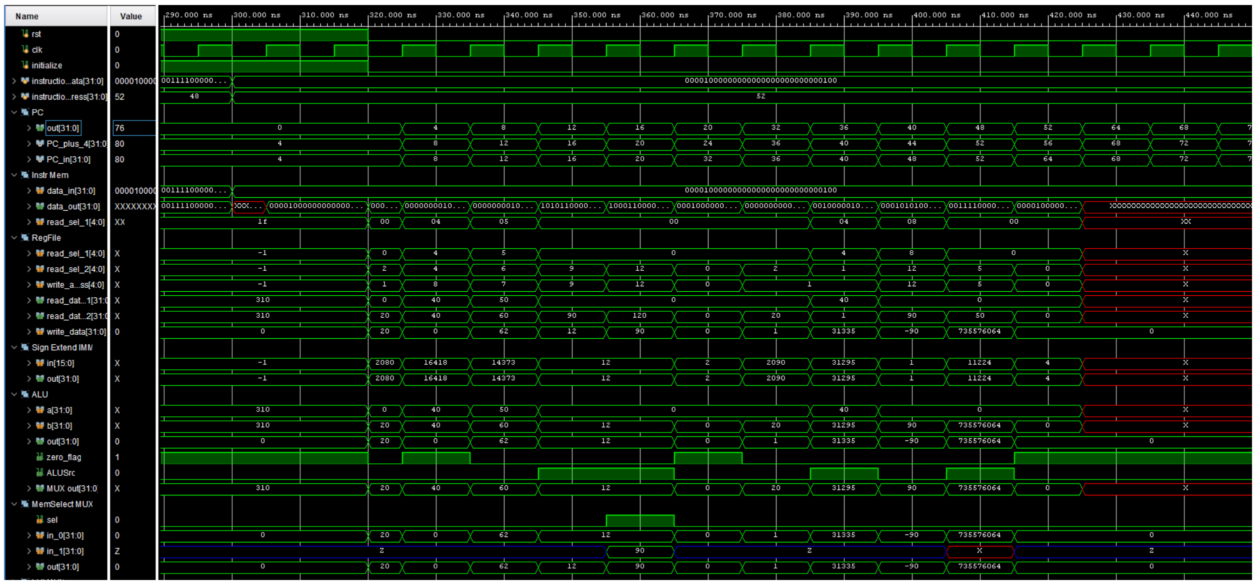
Testbench

The testbench instantiates the 'CPU' module and loads all the test case instructions into the CPU during a 100ns initialization period. Once the 100ns completes, the initialize signal becomes 0, the reset signal becomes 0, and the outputs of the instructions become available in the waveform. We tested the following cases:

000000_00000_00010_00001_00000_10_0000	ADD R1, R0, R2
000000_00100_00100_01000_00000_10_0010	SUB R8, R4, $4
000000_00101_00110_00111_00000_10_0101	OR R7, R5, R6
101011_00000_01001_00000_00000_00_1100	SW R9, 12(R0)
100011_00000_01100_00000_00000_00_1100	LW R12, 12(R0)
000100_00000_00000_00000_00000_00_0010	BEQ R0, R0, 2
000000_00000_00010_00001_00000_10_1010	SLT R1, R0(0)
001000_00100_00001_01111_01000_11_1111	ADDI R1, R4(40)
000101_01000_01100_00000_00000_00_0001	BNE R8, R12, 1
001111_00000_00101_00101_01111_01_1000	LUI R5, 16'h2BD8
000010_00000_00000_00000_00000_00_0100	J 4

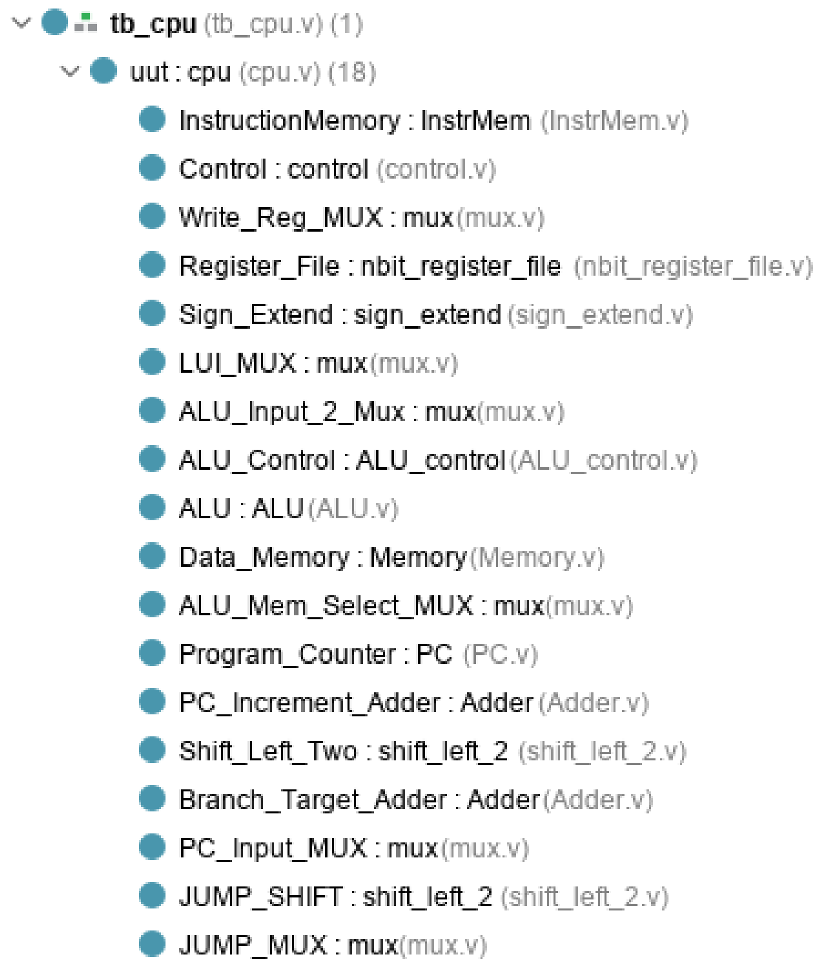
For the provided BEQ testcase, we changed the instruction to branch forward 2 instructions, because the given branch of -1 makes the instruction infinitely loop. Since we branch forward 2 instructions, and BEQ's instruction address is 20, we put the next instruction SLT at address 32. We followed the same reasoning when setting the branch of BNE and the address of LUI.

Waveform



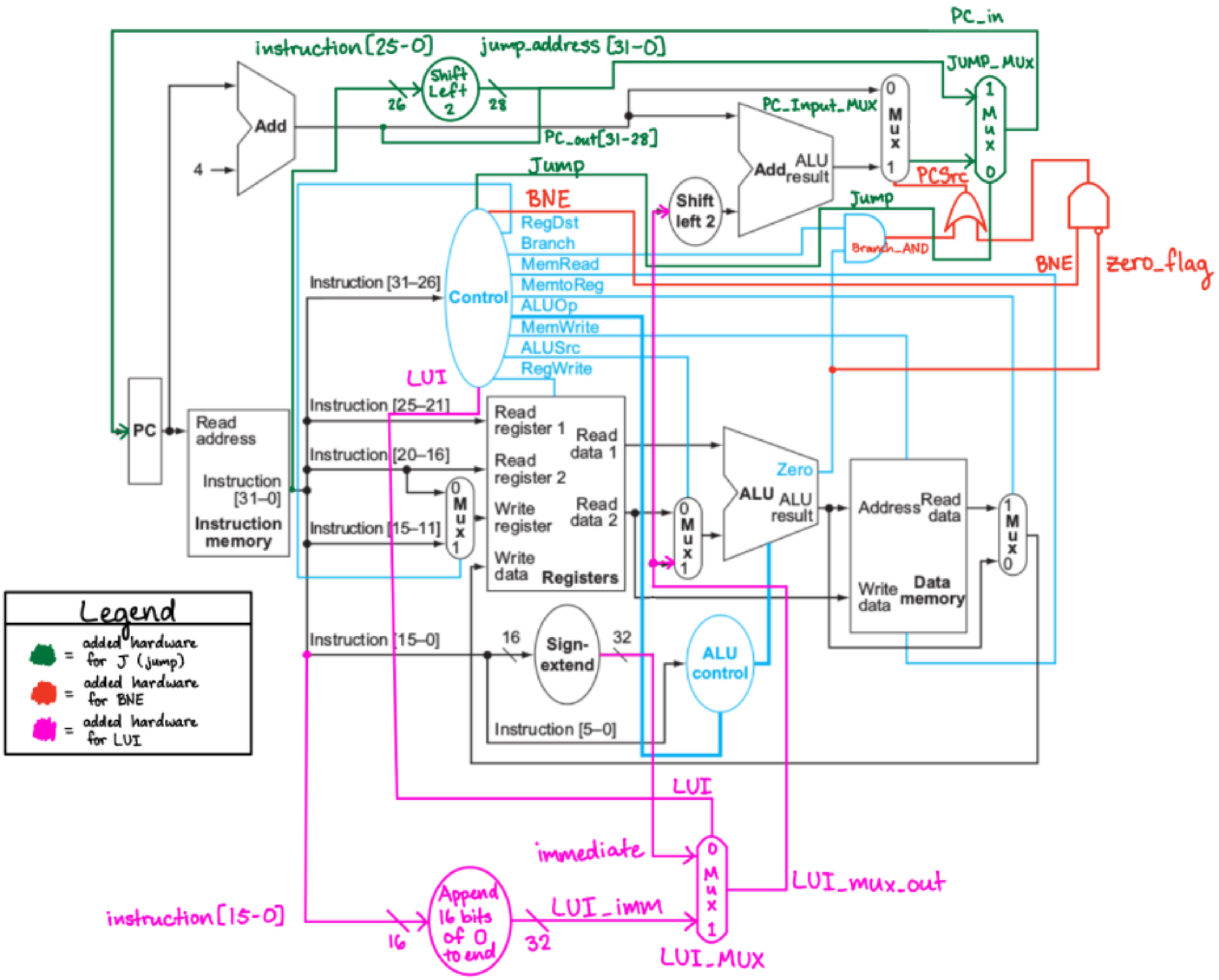
Please see the included .jpg file or the .wcfg for more detail

Design Hierarchy



```
tb_cpu (tb_cpu.v) (1)
    uut : cpu (cpu.v) (18)
        InstructionMemory : InstrMem (InstrMem.v)
        Control : control (control.v)
        Write_Reg_MUX : mux(mux.v)
        Register_File : nbit_register_file (nbit_register_file.v)
        Sign_Extend : sign_extend (sign_extend.v)
        LUI_MUX : mux(mux.v)
        ALU_Input_2_Mux : mux(mux.v)
        ALU_Control : ALU_control (ALU_control.v)
        ALU : ALU (ALU.v)
        Data_Memory : Memory(Memory.v)
        ALU_Mem_Select_MUX : mux(mux.v)
        Program_Counter : PC (PC.v)
        PC_Increment_Adder : Adder (Adder.v)
        Shift_Left_Two : shift_left_2 (shift_left_2.v)
        Branch_Target_Adder : Adder(Adder.v)
        PC_Input_MUX : mux(mux.v)
        JUMP_SHIFT : shift_left_2 (shift_left_2.v)
        JUMP_MUX : mux(mux.v)
```

]**Diagram of Modified CPU**



PC_in
instruction[25-0]
jump_address[31-0]
JUMP_MUX
Shift Left 2
26
28
PC_out[31-28]
PC_Input_MUX
Add
4
Jump
BNE
PCSrc
Branch_AND
zero_flag
Shift left 2
ALU result
Mux
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
LUI
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Zero
ALU
Address
Read data
Write data
Data memory
Instruction [15–0]
16
Sign-extend
32
Instruction [5–0]
ALU control
Legend
added hardware for J (jump)
added hardware for BNE
added hardware for LUI
immediate
instruction[15-0]
Append 16 bits of 0 to end
LUI_imm
LUI_mux_out
LUI_MUX