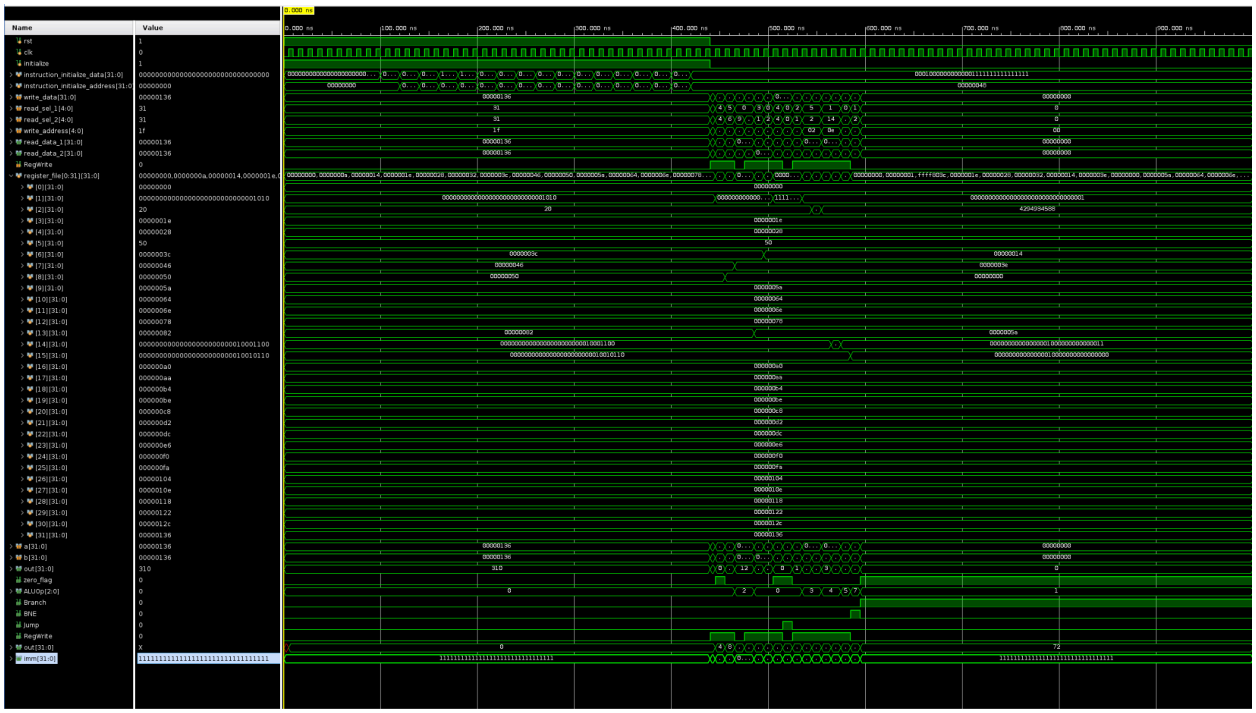
EC413 - LAB 7

Siara Patel & Vanshika Chaddha

Source Files

- cpu.v
- InstrMem.v
- control.v
- mux.v
- nbit_register_file.v
- sign_extend.v
- zero_extend.v
- three_input_mux.v
- ALU_control.v
- ALU.v
- Memory.v
- PC.v
- Adder.v
- shift_left_2.v

Waveform



Additions/Changes:

Implementing NOR, AND, XOR

To add Nor, And, and Xor functionality, in the ALU_control module we check the ALUOp code set in the control module. In this case, it is R-type corresponding to 3'b000. For R-type instruction, we also check the last five bits of the instruction input to and set func to a decimal value. Func then corresponds to options in the ALU module and executes the ALU functionality. We implemented these functions using behavioral verilog.

Implementing J-Type Instructions

In order to implement the jump functionality, we went into the control module and added in a jump signal. We created another instruction code as per the MIPS sheet (2 hex) and then went into the CPU and created two wires: jump_adress and node. OUr shifffty_2 takes in the last 25 bits of our instruction and shifts it left by 2 (stored into node). The jump_address is assigned by concatenating PC_plus_4[31:28], node[27:0]. Then, this gets inputted into mux_boiiii where Jump (the signal is set to 1) and jump_adress is selected, allowing for the PC to jump to the specified address.

Implementing I-Type Instructtructions changed ALUOp size from two bits to three bits to configure for more I-Type instructions.

To implement more I-type instructions, we

Addi → to implement Addi we still use the ALU module and add functionality. However, to do this we must use a mux to choose between read data 2 (RD2) and the sign extended immediately. We do this using a three input mux called ALU_Input_2_Mux. ALUScr selects between RD2 and the immediate and is set in the control module based on the instruction type.

Ori → to implement Ori we still use the ALU module and or functionality. Again we use ALU_Input_2_Mux because we must choose between RD2 and the immediate. It works the same as functionality to ori.

Lui → to implement Lui, in the CPU module we created a wire called left_ex_imm and shifted it to 16 bits. We then pass this through the alu this just sets the ALU out to the left extended immediate.

Now because we are computing three different types of immediates, we created a four input MUX to choose between different immediate values depending on the ALUOp/Instruction type.
Sign extend → Addi
Zero extend → Ori
Shifted Left 16 → Lui