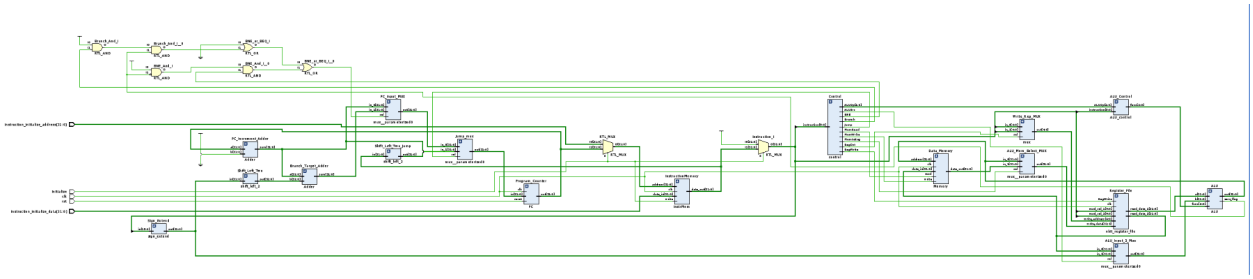
Mark Vinciguerra
EC413
Lab 6: Single Cycle CPU

Single Cycle CPU Schematic:



SLT:
Added additional R-type instruction in ALU Control. In the ALU we have the output being set to 1 if a is less than b.

ADDI:
Added an ADDI instruction to the control and set control items to store added immediate value to the destination register.

```
end else if (instruction == 6'b00_1000) begin //addi
ALUOp = 2'b10;
MemRead = 1'b0;
MemtoReg = 1'b0;
RegDst = 1'b0;
Branch = 1'b0;
ALUSrc = 1'b1;
MemWrite = 1'b0;
RegWrite = 1'b1;
```

J:

Added j instruction in control and jump variable to all the other control modules. We set jump to jump to address 0, the first instruction, where it does a loop going from the first instruction, the other instructions, then jumping again. ALUOp is don't cares because the ALU does not matter for this instruction. We also created a new mux where the input is the output of the old mux. The new mux output goes to the PC giving it its new instruction address.

```
end else if (instruction == 6'b00_0010) begin //jump
ALUOp = 2'b11;
MemRead = 1'b0;
MemtoReg = 1'b0;
RegDst = 1'b0;
Branch = 1'b0;
ALUSrc = 1'b0;
MemWrite = 1'b0;
RegWrite = 1'b0;
Jump = 1'b1;

mux #(32) PC_Input_MUX (PCSrc, PC_plus_4, Branch_target_address, PC_input_mux_output);

wire [31:0] jump_address_second_half;

shift_left_2 #(32) Shift_Left_Two_jump (instruction_mem_out, jump_address_second_half);
wire [31:0] jumpAddress = {PC_plus_4[31:28], jump_address_second_half[27:0]};

mux #(32) jump_mux (Jump, PC_input_mux_output, jumpAddress, PC_in);
```

Branch:
Added BNE instruction to control. Also added additional hardware to the CPU.

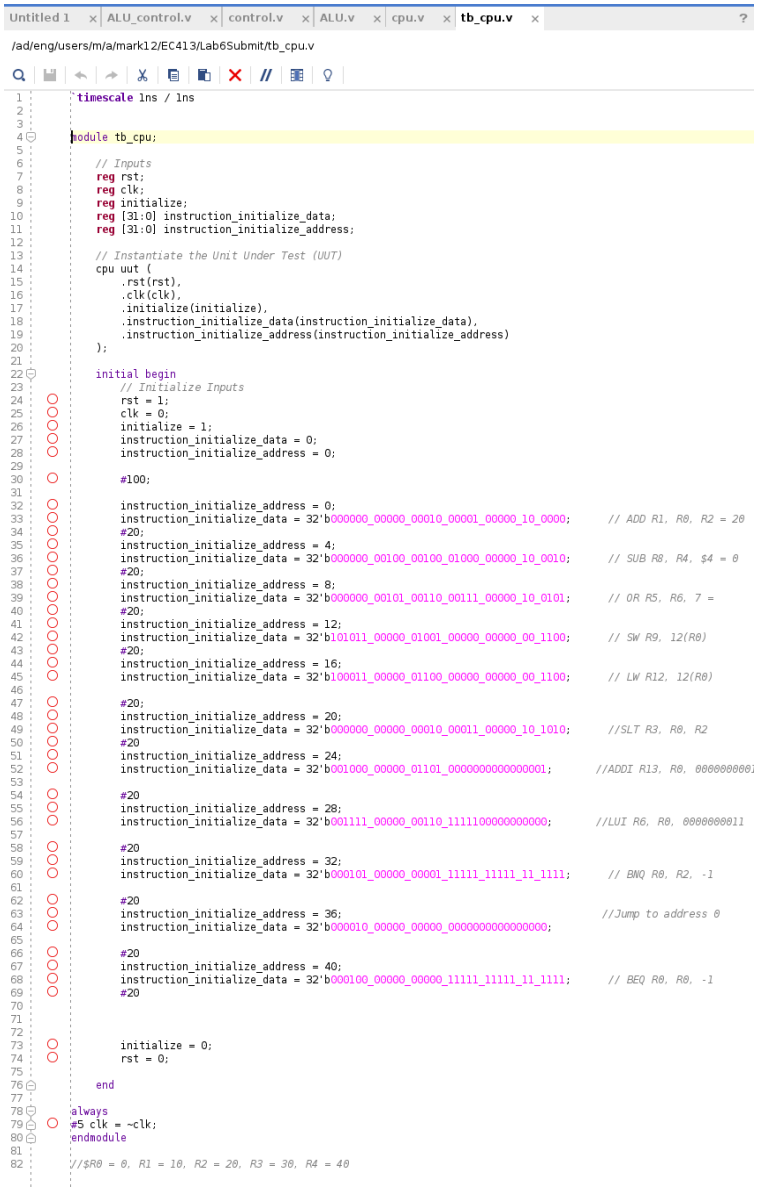
```
end else if (instruction == 6'b00_0101) begin   //BNE
ALUOp = 2'b01;
MemRead = 1'b0;
MemtoReg = 1'b0;
RegDst = 1'b0;
Branch = 1'b0;
ALUSrc = 1'b0;
MemWrite = 1'b0;
RegWrite = 1'b0;
Jump = 1'b0;
BNE = 1'b1;
```

```
wire [31:0] Branch_target_address;
wire [31:0] immediate_x_4;
shift_left_2 #(32) Shift_Left_Two (immediate, immediate_x_4);
Adder #(32) Branch_Target_Adder (PC_plus_4, immediate_x_4, Branch_target_address);


wire PCSrc1;
wire PCSrc2;
wire PCSrc;
wire [31:0] PC_input_mux_output;

and Branch_And (PCSrc1, Branch, zero_flag);
not BNE_not_zero (not_zero_flag, zero_flag);
and BNE_And (PCSrc2, not_zero_flag, BNE);
or BNE_or_BEQ(PCSrc, PCSrc1, PCSrc2);

mux #(32) PC_Input_MUX (PCSrc, PC_plus_4, Branch_target_address, PC_input_mux_output);

wire [31:0] jump_address_second_half;

shift_left_2 #(32) Shift_Left_Two_jump (instruction_mem_out, jump_address_second_half);
wire [31:0] jumpAddress = {PC_plus_4[31:28], jump_address_second_half[27:0]};

mux #(32) jump_mux (Jump, PC_input_mux_output, jumpAddress, PC_in);
```

LUI:
Added additional LUI instruction to the control. Added new ALUOp function for LUI:

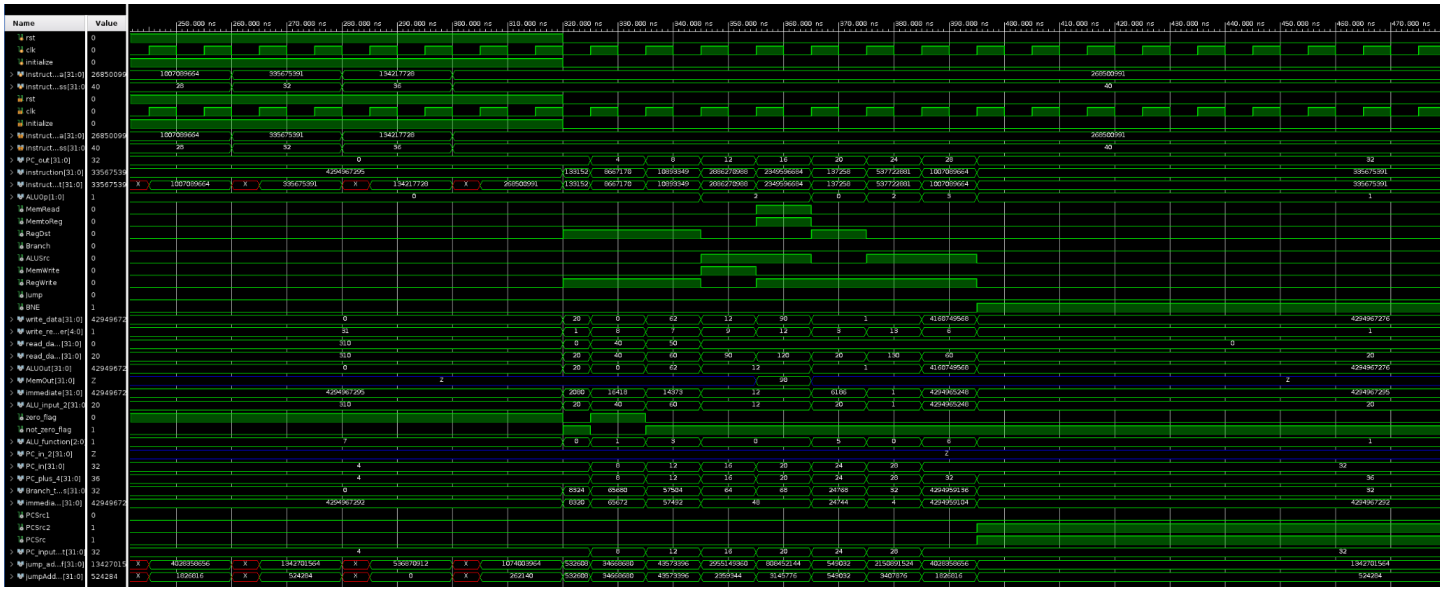
```
else if (func == 3'd6)
out = {b[15:0], 16'b0};
```

Testbench:



Address 20 uses SLT comparing R0 and R2 and stores result in R3. Address 24 adds R0 with immediate 1 in decimal and storing value in R13. Address 28 uses LUI loading the immediate 0000000011 into the most significant 16 bits of register R0 and storing this value in R6. Address 32 does BNQ comparing R0 and R2 and if not equal, it loops to the address before making infinite loop. Address 36 makes a jump to the first instruction also making an infinite loop as time goes on.

Simulation:


Name
Value
rst
clk
initialize
instruct...a[31:0]
instruct...ss[31:0]
PC_out[31:0]
instruction[31:0]
instruct...t[31:0]
ALUOp[1:0]
MemRead
MemtoReg
RegDst
Branch
ALUSrc
MemWrite
RegWrite
Jump
BNE
write_data[31:0]
write_re...er[4:0]
read_da...[31:0]
ALUOut[31:0]
MemOut[31:0]
immediate[31:0]
ALU_input_2[31:0]
zero_flag
not_zero_flag
ALU_function[2:0]
PC_in_2[31:0]
PC_in[31:0]
PC_plus_4[31:0]
Branch_t...s[31:0]
immedia...[31:0]
PCSrc1
PCSrc2
PCSrc
PC_input...t[31:0]
jump_ad...t[31:0]
JumpAdd...[31:0]