Lab 5 EC413

Members: Pree Simphliphan, Rayan Syed

Components of Lab + Explanation:

SLT:

Since SLT is R-type we reused the instruction used for the add/xor etc. in the control module

In the ALU_control module we made sure that the instruction for SLT would choose the ALU function (function 6):

control:

```
else if (instruction == 6'h2a) //slt
func = 4'd6;
```

ALU_control:

```
else if (func == 4'd6)
out = a<b;
```

We implemented this just like the prelab's xor

Jump:

For jump we had to add a jump register output for control, to be passed to more modules in the overall cpu. The rest of this code chunk is the same as that for branch (in control.v)

```
    else if (instruction == 6'b000010) begin  //jump
    ALUOp = 2'b01;
    MemRead = 1'b0;
    MemtoReg = 1'b0;
    RegDst = 1'b0;
  Branch = 1'b1;
   ALUSrc = 1'b0;
    MemWrite = 1'b0;
    RegWrite = 1'b0;
    jump = 1;
```

This new code was added as per the discussion slides for the lab; this let us successfully implement jump:

```
//for jump
wire [31:0] jumpaddress;
shift_left_2 #(32) Shift_Left_Two2 (instruction[25:0], jumpaddress[27:0]);
assign jumpaddress = {PC_plus_4 [31:28],jumpaddress[27:0]};
mux #(32) jump_MUX (jump, PC_in1, jumpaddress, PC_in);
```

wire [31:0]

ADDI, ORI, LUI:

For these 3 immediate functions we simply had to make the first 6 instruction digits trigger the I-type signal in control to be dealt with in ALU_control. The rest of the code chunk is the same as R-type (excpet ALUSrc)

```
  else if (instruction == 6'b001000 || instruction == 6'b001101 || instruction == 6'b001111) begin              //IType
  ALUOp = 2'b01;
  MemRead = 1'b0;
  MemtoReg = 1'b0;
  RegDst = 1'b0;        // select R2 as written register
Branch = 1'b0;
 ALUSrc = 1'b1;          // choose immediate as R3
  MemWrite = 1'b0;
  RegWrite = 1'b1;
  jump = 0;
  end
```

The functions simply called the same ALU functions as add/or for addi/ori, but for lui we added a 7th function that simply shifted left 16 bits (will not attach screenshot since so simple):

```
if (instruction2 == 6'h08)
 func = 4'd0;
 //ori
 else if(instruction2==6'h0d)
 func = 4'd3;

 //lui
 else if(instruction2==6'h0f)
 func= 4'd7;
```

BNE:

While doing BNE, we also reimplemented BEQ. We essentially just added two functions to the ALU that would deal with beq and bne. If beq, the zero_flag would be 0; if if bne, the zero_flag would be 0. To do this, we increased the size of the func to 4 bits rather than 3 so we could have a function 9/10 for beq and bne respectively.

BEQ and BNE functions - this code falls under the ALU_control block:

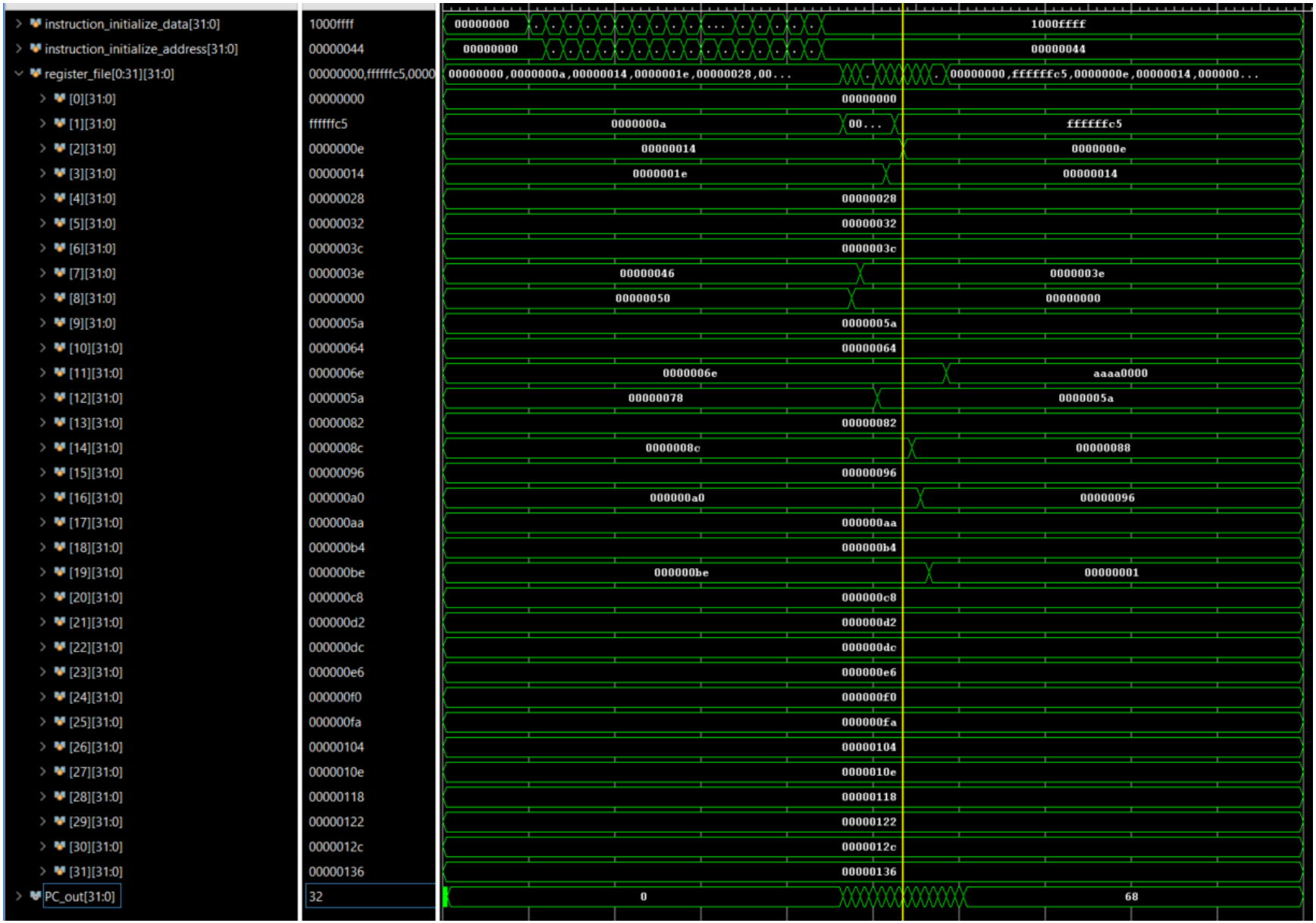
```
 //beq
 else if(instruction2== 6'h04)
 func = 4'd8;

 //bne
 else if (instruction2 == 6'h05)
 func = 4'd9;
```

Testbench:

We updated the testbench to include test cases for all the new functions.

We pulled out PC_out to see where the current instruction was.

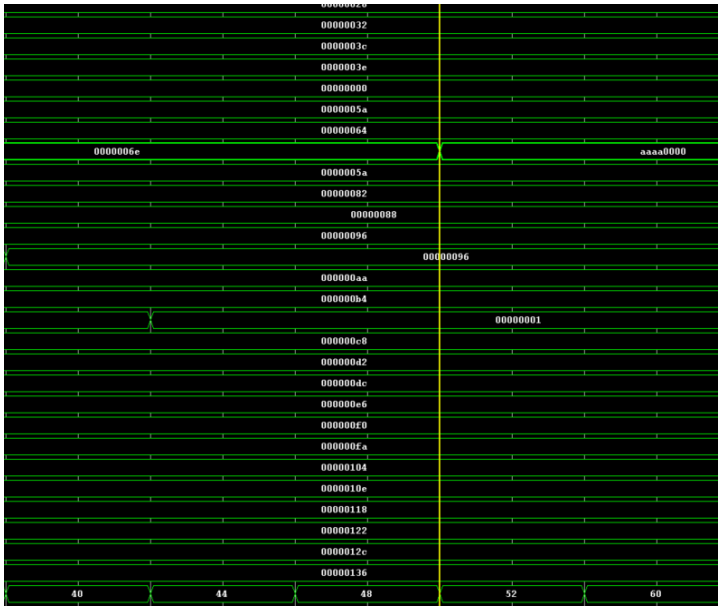
Here is the relevant waveform:


instruction_initialize_data[31:0]
1000ffff
instruction_initialize_address[31:0]
00000044
register_file[0:31][31:0]
00000000,ffffffc5,0000
[0][31:0]
00000000
[1][31:0]
ffffffc5
[2][31:0]
0000000e
[3][31:0]
00000014
[4][31:0]
00000028
[5][31:0]
00000032
[6][31:0]
0000003c
[7][31:0]
0000003e
[8][31:0]
00000000
[9][31:0]
0000005a
[10][31:0]
00000064
[11][31:0]
0000006e
[12][31:0]
0000005a
[13][31:0]
00000082
[14][31:0]
0000008c
[15][31:0]
00000096
[16][31:0]
000000a0
[17][31:0]
000000aa
[18][31:0]
000000b4
[19][31:0]
000000be
[20][31:0]
000000c8
[21][31:0]
000000d2
[22][31:0]
000000dc
[23][31:0]
000000e6
[24][31:0]
000000f0
[25][31:0]
000000fa
[26][31:0]
00000104
[27][31:0]
0000010e
[28][31:0]
00000118
[29][31:0]
00000122
[30][31:0]
0000012c
[31][31:0]
00000136
PC_out[31:0]
32
00000000
1000ffff
00000044
00000000,0000000a,00000014,0000001e,00000028,00...
00000000,ffffffc5,0000000e,00000014,000000...
0000000a
00...
ffffffc5
00000014
0000000e
0000001e
00000014
00000046
0000003e
00000050
00000000
0000006e
aaaa0000
00000078
0000008c
00000088
000000a0
00000096
000000be
00000001
0
68

For SLT, you can see that R19 changes to 1 because at that instruction R17 < R18 based on our testbench:

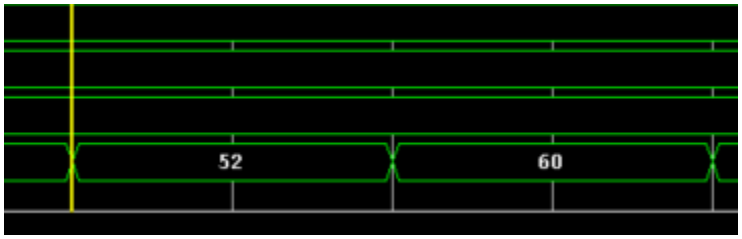
This happens at PC 40 because this is when we made it occur on the testbench.

For LUI, the contents of immediate we put in the instruction are shifted 16 bits to the left and assigned it to register 11 as seen here in the waveform:



This occurs at PC 48 because this is when it is supposed to according to our testbench.

For BNE, it branches from PC_out 52 to 56 which means it skips one instruction as we indicated in testbench



Here is a picture of instruction (PC) 56 being skipped, as the actual register values are irrelevant to testing this.

For J, it jumps from PC_out 60 to 68 which means it skips one instruction as we indicated in testbench: