lab report 7

Andrew Nady

900184042

Codes

- All code is in a separate folder
- Risc-v module:

```
`timescale 1ns / 1ps
module Full_dataPath( input clk,input fpga, input rst ,input [1:0] LedSel ,
input [3:0] ssdSel , output reg [7:0] leds, output  [6:0]SSDout, output[3:0]
anodes);

wire [31:0] adder1,adder2,PCmux,PCout;
wire cout1_ripple1,cout1_ripple2;
wire [31:0]ints_out;
wire [31:0]readdata1, readdata2 ;
wire  [31:0] gen_out; //should be reg || wire
wire branch;
 wire  memread;
  wire  memtoreg;
 wire [1:0]aluop;
  wire memwrite;
   wire  alusrc;
    wire regwrite;
    wire [3:0]aluS;
    wire [31:0]outmux_input_alu;
    wire [31:0]ALU_result;
    wire zero;
    wire [31:0] dataMem_out;
    wire [31:0] writingData;
    wire [31:0] shiftout;
    reg [12:0]num;
    // wires of piplened regs
    wire [31:0] IF_ID_PC, IF_ID_Inst; // input to REG1_ID
    //output of ID_EX
wire memtoreg_ID_out;
wire regwrite_ID_out;
wire memread_ID_out;
wire memwrite_ID_out;
   wire  branch_ID_out;
   wire [1:0]aluop_ID_out;
   wire alusrc_ID_out;
   wire [31:0]IF_ID_PC_ID_out;
   wire [31:0]readdata1_ID_out;
   wire [31:0]readdata2_ID_out;
   wire [31:0]gen_out_ID_out;
   wire [3:0]ints_out_ID_out_up;
   wire [4:0]ints_out_ID_out_down;
   //out of EX/MEM
  wire  memtoreg_EX_out;
  wire regwrite_EX_out;
  wire memread_EX_out;
```

```
   wire memwrite_EX_out;
   wire  branch_EX_out;
     wire [31:0]adder_branch_Ex_out;
     wire zero_EX_out;
     wire [31:0]ALU_result_EX_out;
     wire [31:0]readdata2_EX_out;
     wire [4:0]ints_out_EX_out_down;
   //out of MEM_WB
   wire memtoreg_MEM_out;
      wire  regwrite_MEM_out;
      wire  [31:0] dataMem_out_MEM_out;
      wire [31:0]ALU_result_MEM_out;
      wire [4:0]ints_out_MEM_out_down;
//instantiate 32bitreg PCin PCout
//module OneReg(input clk,input S,input [31:0] in,input rst, output [31:0] Q
);
OneReg  pc( clk,1, PCmux, rst,  PCout );
ripple_carry  ripplecar0(PCout,
32'd4,
0,
adder1,
cout1_ripple1
);
//adder branch  (DONE)
ripple_carry  ripplecar1(IF_ID_PC_ID_out,
shiftout,
0,
adder2,
cout1_ripple2
);

//mux
ThirtytwoMUX  mux3(adder1,adder2,branch_EX_out & zero_EX_out ,PCmux);
//register (PCmux, PCin)  //reload the PC

//instmem
InstMem instmem(PCout[7:2],ints_out);
//-------------------------------------If/ID-------------------------------
----
  sixtyFour_reg #(64) IF_ID  (clk,
1'b1,{PCout,ints_out},rst,{IF_ID_PC,IF_ID_Inst});
//control unit  (DONE)
Control_unit  controlunit(IF_ID_Inst[6:2], branch,
   memread, memtoreg,
aluop, memwrite,  alusrc, regwrite );
//alu_control (DONE)
ALU_Control   aluControl(aluop_ID_out,
ints_out_ID_out_up[2:0],ints_out_ID_out_up[3] ,aluS);
//registerfile DONE
RegisterFile regfile( clk, rst,
IF_ID_Inst[19:15], IF_ID_Inst[24:20], ints_out_MEM_out_down,
 writingData,
 regwrite_MEM_out,
```

```
 readdata1, readdata2  );
//immGenerator    (DONE)
immediate_generator   immgen( gen_out, IF_ID_Inst );

//------------------------------ ID_EX--------------------------------
 sixtyFour_reg #(145) ID_EX (clk, 1'b1,
 {memtoreg,regwrite,memread,memwrite,
 branch,aluop,alusrc,IF_ID_PC,readdata1, readdata2
 ,gen_out,{IF_ID_Inst[30],IF_ID_Inst[14:12]},IF_ID_Inst[11:7] },rst,

{memtoreg_ID_out,regwrite_ID_out, memread_ID_out, memwrite_ID_out,
branch_ID_out,aluop_ID_out
,alusrc_ID_out,IF_ID_PC_ID_out, readdata1_ID_out,readdata2_ID_out,
  gen_out_ID_out
 ,ints_out_ID_out_up, ints_out_ID_out_down});


//shifting left  (done)
 shiftLeft sh(gen_out_ID_out ,shiftout);
//mux (DONE)
 ThirtytwoMUX mux (readdata2_ID_out,gen_out_ID_out,alusrc_ID_out
,outmux_input_alu);
 //ALU  (Done)
 ALU alu(
 readdata1_ID_out, outmux_input_alu,
 aluS,
 ALU_result,  zero );

//----------------------EX/MEM-----------------------------------------------
------
  sixtyFour_reg #(107) EX_MEM (clk, 1'b1,
 {memtoreg_ID_out,regwrite_ID_out, memread_ID_out, memwrite_ID_out,
branch_ID_out,
  adder2,zero,ALU_result,readdata2_ID_out,ints_out_ID_out_down },rst,

{ memtoreg_EX_out,
regwrite_EX_out,
   memread_EX_out,
    memwrite_EX_out,
     branch_EX_out,
      adder_branch_Ex_out,
      zero_EX_out,
     ALU_result_EX_out,
      readdata2_EX_out,
     ints_out_EX_out_down});

 //dataMem
 DataMem  datamem( clk, memread_EX_out, memwrite_EX_out,
  ALU_result_EX_out[7:2] , readdata2_EX_out,  dataMem_out);
  //----------------------MEM/WB-----------------------------------------------
--------
  sixtyFour_reg #(71) MEM_WB (clk, 1'b1,
   {regwrite_EX_out,memtoreg_EX_out,
```

```
     dataMem_out,ALU_result_EX_out,ints_out_EX_out_down},rst,

  {

       regwrite_MEM_out,
       memtoreg_MEM_out,
      dataMem_out_MEM_out,ALU_result_MEM_out,
             ints_out_MEM_out_down
 });



//mux after the datamem
ThirtytwoMUX mux_writing
(ALU_result_MEM_out,dataMem_out_MEM_out,memtoreg_MEM_out ,writingData);


//switch or if on input ledsel to generate outputs leds
always @(*)
begin
        case(LedSel)
        2'b00 :
        begin
         leds[0]= regwrite;
         leds[1]= alusrc;
         leds[3:2]=  aluop;
         leds[4]=  memread;
         leds[5]= memwrite;
         leds[6]=  memtoreg;
         leds[7]=  branch;
        end
        2'b01:
        begin
        leds[3:0]=aluS;
        leds [4]=zero;
        leds[5]=branch&zero;
        leds[7:6] =0;

        end
        2'b11:
             leds[7:0]= {1'b0,ints_out[6:0]}; //  monotring the opcode
        ////switch or if on input ssdsel to generate ssdnumber
        default : leds[7:0]= 0;
endcase

end
always@(*) begin
case (ssdSel)
4'b0000:num = PCout;
4'b0001: num =PCout+4;
4'b0010:  num = adder2;
4'b0011: num = PCmux;
4'b0100: num = readdata1;
```

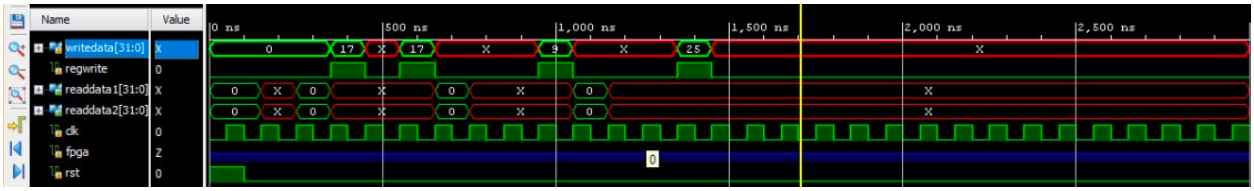
```
4'b0101: num= readdata2;
4'b0110: num=  writingData;
4'b0111: num= gen_out;
4'b1000: num= shiftout;
4'b1001:num=outmux_input_alu;
4'b1010: num= ALU_result;
4'b1011:num=dataMem_out;
default : num = 0;
endcase
  end
//instantiate 7 seg

seven ssd ( fpga, num,anodes,SSDout);

endmodule
```

Simulation screenshots

Here is the data out from memory

Here is the writing data

Here is the writing data and the reg write signal



Table

	Pc Output	Pc input (branch adder or PC+4)	Writing data (Reg file)	Writing data (data mem)
`lw x1, 0(x0)`	4	8	17	17
`lw x2, 4(x0)`	20	24	9	9
`lw x3, 8(x0)`	36	40	25	25
`or x4, x1, x2`	52	56	25	0
`beq x4, x3, 16`	68	16*2+68=100	0	0
`add x3, x1, x2`				
`add x5, x3, x2`	100	104	25+9=34	0
`sw x5, 12(x0)`	116	120	0	0
`lw x6, 12(x0)`	132	136	34	34
`and x7, x6, x1`	150	154	34&17=0	0
`sub x8, x1, x2`	164	170	17-9=8	0
`add x0, x1, x2`	180	184	0	0
`add x9, x0, x1`	196	200	17	0

As x4, x3 are equal so the PC will jump

minimize the number of NOP

- **as know that and, add, … needs 2 bubbles (2 Nop) -> data hazard**
- **load word need 3 bubbles (3 NOP) ->data hazard**
- **beq need 3 bubbles (3 nop)-> control hazard**

```
add x0, x0, x0
lw x1, 0(x0)
lw x2, 4(x0)
lw x3, 8(x0)
NOP ->for x1(data hazard)
NOP -> for x2(data hazard)
or x4, x1, x2
NOP -> (data hazard)
NOP -> (data hazard)
beq x4, x3, 16
NOP -> (control hazard)
NOP -> (control hazard)
NOP -> (control hazard)
add x3, x1, x2
NOP -> (data hazard)
NOP -> (data hazard)
add x5, x3, x2
NOP -> (data hazard)
NOP -> (data hazard)
sw x5, 12(x0)
lw x6, 12(x0)
NOP -> (data hazard)
NOP -> (data hazard)
NOP -> (data hazard)
and x7, x6, x1
sub x8, x1, x2
add x0, x1, x2
add x9, x0, x1
```

Modifying the program

```
add x0, x0, x0
lw x1, 0(x0)
lw x2, 4(x0)
lw x3, 8(x0)
NOP -> (data hazard)
NOP -> (data hazard)
or x4, x1, x2
NOP -> (data hazard)
NOP -> (data hazard)
beq x4, x3, 16
NOP -> (control hazard)
NOP -> (control hazard)
NOP -> (control hazard)
add x3, x1, x2
NOP -> (data hazard)
NOP -> (data hazard)
add x5, x3, x2
NOP -> (data hazard)
NOP -> (data hazard)
sw x5, 12(x0)
lw x6, 12(x0)
sub x8, x1, x2
add x0, x1, x2
add x9, x0, x1
and x7, x6, x1
```

My program

```
lw t4, 0(zero)
lw t5, 4(zero)
lw t6, 8(zero)
loop:
beq t4,t6,exit
add  t4,t4,t5
beq zero,zero,loop

exit:
sw t4, 12(zero)
```

Modifying the program

```
lw t4, 0(zero)
lw t5, 4(zero)
lw t6, 8(zero)
NOP
NOP
NOP
loop:
beq t4,t6,exit
NOP
NOP
NOP
add  t4,t4,t5       1
beq zero,zero,loop
NOP
NOP
NOP
exit:
sw t4, 12(zero)
```