Lab Report 6
MIPS Single-cycle

Owner: Vishal J Parikh.

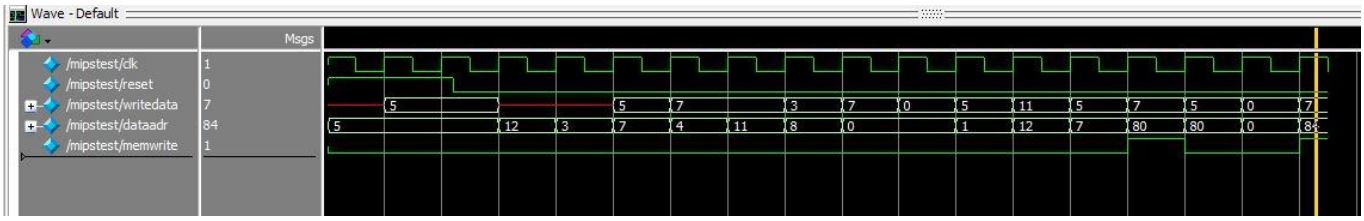
Part A:

1. A completed version of Table 1.

cycle	reset	pc	instruction	branch	srca	srcb	aluout	zero	pcsrc	write data	mem write	read data
1	1	0	addi $2,$0,5	0	0	5	5	0	0	5	0	x
2	0	4	addi $3, $0, 12	0	0	c	c	0	0	x	0	x
3	0	8	addi $7, $3, -9	0	x	-9	3	0	0	x	0	x
4	0	c	or $4, $7, $2	0	3	5	7	0	0	5	0	x
5	0	10	and $5, $3, $4	0	c	7	4	0	0	7	0	x
6	0	14	add $5, $5, $4	0	4	7	b	0	0	7	0	x
7	0	18	beq $5, $7, end	1	b	3	8	0	0	3	0	x
8	0	1c	slt $4, $3, $4	0	c	7	0	0	0	7	0	x
9	0	20	beq $4 $0, around	1	0	0	0	1	1	0	0	x
10	0	24	addi $5 $0, 0	0								
11	0	28	slt $4, $7, $2	0	3	5	1	0	0	5	0	x
12	0	2c	add $7, $4, $5	0	1	b	c	0	0	b	0	x
13	0	30	sub $7, $7, $2	0	c	5	7	0	0	5	0	x
14	0	34	sw $7, 68($3)	0	38	c	50	0	0	7	1	x
15	0	38	lw $2,80($0)	0	50	0	50	0	0	5	0	7
16	0	3c	j end	1			0	0	0	0	0	x
17	0	40	addi $2, $0, 1	0								
18	0	44	sw $2, 84($0)	0	54	0	54	0	0	7	1	x

2. An image of the simulation waveforms showing correct operation of the unmodified processor.

Does it write the correct value to address 84? : **YES**




```
# Simulation succeeded
# ** Note: $stop    : C:/intelFPGA_lite/18.1/ECE 469/mips/mipstest.sv(32)
#    Time: 125 ps  Iteration: 1  Instance: /mipstest
# Break in Module mipstest at C:/intelFPGA_lite/18.1/ECE 469/mips/mipstest.sv line 32
VSIM 2> run
```

Part B:

1. List of the new instructions implemented, their encoding, marked up versions of the Datapath schematic and decoder tables that include the new instructions.

BNE -- *Branch on not equal*

Description:	Branches if the two registers are not equal
Operation:	if $s != $t advance_pc (offset << 2)); else advance_pc (4);
Syntax:	bne $s, $t, offset
Encoding:	0001 01ss ssst tttt iiii iiii iiii iiii

ORI -- *Bitwise or immediate*

Description:	Bitwise ors a register and an immediate value and stores the result in a register
Operation:	$t = $s | imm; advance_pc (4);
Syntax:	ori $t, $s, imm
Encoding:	0011 01ss ssst tttt iiii iiii iiii iiii

3. The contents of your memfile2.dat containing your test2 machine language code.

```
34088000
20098000
350a8001
11090005
0128582a
15600001
08100009
01485022
350800ff
016a5820
01484022
ad680052
```

2. Your SystemVerilog code for your modified MIPS with the changes

mips.sv

```
// files needed for simulation:
//  mipsttest.sv   mipstop.sv, mipsmem.sv,  mips.sv,  mipsparts.sv

// single-cycle MIPS processor
module mips(input  logic        clk, reset,
            output logic [31:0] pc,
            input  logic [31:0] instr,
            output logic        memwrite,
            output logic [31:0] aluout, writedata,
            input  logic [31:0] readdata);

  logic        memtoreg, branch,
               pcsrc, zero,
               regdst, regwrite, jump;
  logic [1:0]  alusrc;
  logic [2:0]  alucontrol;

  controller c(instr[31:26], instr[5:0], zero,
               memtoreg, memwrite, pcsrc,
               alusrc, regdst, regwrite, jump,
               alucontrol);
  datapath dp(clk, reset, memtoreg, pcsrc,
              alusrc, regdst, regwrite, jump,
              alucontrol,
              zero, pc, instr,
              aluout, writedata, readdata);
endmodule

module controller(input  logic [5:0] op, funct,
                  input  logic       zero,
                  output logic       memtoreg, memwrite,
                  output logic       pcsrc,
                  output logic [1:0] alusrc,
                  output logic       regdst, regwrite,
                  output logic       jump,
                  output logic [2:0] alucontrol);

  logic [1:0] aluop;
  logic       branch;

  maindec md(op, memtoreg, memwrite, branch,
             alusrc, regdst, regwrite, jump,
             aluop);
  aludec  ad(funct, aluop, alucontrol);
```

```
49
50  module maindec(input  logic [5:0] op,
51                 output logic       memtoreg, memwrite,
52                 output logic       branch,
53                 output logic [1:0] alusrc,
54                 output logic       regdst, regwrite,
55                 output logic       jump,
56                 output logic [1:0] aluop);
57
58    logic [9:0] controls;
59
60    assign {regwrite, regdst, alusrc,
61            branch, memwrite,
62            memtoreg, jump, aluop} = controls;
63
64    always_comb
65      case(op)
66        6'b000000: controls = 10'b1100000010; //Rtype
67        6'b100011: controls = 10'b1001001000; //LW
68        6'b101011: controls = 10'b0001010000; //SW
69        6'b000100: controls = 10'b0000100001; //BEQ
70        6'b000101: controls = 10'b0100100001; //BNE
71        6'b001000: controls = 10'b1001000000; //ADDI
72        6'b001101: controls = 10'b1010000011; //ORI
73        6'b000010: controls = 10'b0000000100; //J
74        default:   controls = 10'bxxxxxxxxxx; //???
75      endcase
76  endmodule
77
78  module aludec(input  logic [5:0] funct,
79                input  logic [1:0] aluop,
80                output logic [2:0] alucontrol);
81
82    always_comb
83      case(aluop)
84        2'b00: alucontrol = 3'b010;  // add
85        2'b01: alucontrol = 3'b110;  // sub
86        2'b11: alucontrol = 3'b001;  // OR
87        default: case(funct)          // RTYPE
88            6'b100000: alucontrol = 3'b010; // ADD
89            6'b100010: alucontrol = 3'b110; // SUB
90            6'b100100: alucontrol = 3'b000; // AND
91            6'b100101: alucontrol = 3'b001; // OR
92            6'b101010: alucontrol = 3'b111; // SLT
```

```
endmodule

module datapath(input  logic        clk, reset,
                input  logic        memtoreg, pcsrc, regdst,
                input  logic [1:0]  alusrc,
                input  logic        regwrite, jump,
                input  logic [2:0]  alucontrol,
                output logic        zero,
                output logic [31:0] pc,
                input  logic [31:0] instr,
                output logic [31:0] aluout, writedata,
                input  logic [31:0] readdata);

  logic [4:0]  writereg;
  logic [31:0] pcnext, pcnextbr, pcplus4, pcbranch;
  logic [31:0] signimm, signimmsh, zeroimm;
  logic [31:0] srca, srcb;
  logic [31:0] result;

  // next PC logic
  flopr #(32) pcreg(clk, reset, pcnext, pc);
  adder       pcadd1(pc, 32'b100, pcplus4);
  sl2         immsh(signimm, signimmsh);
  adder       pcadd2(pcplus4, signimmsh, pcbranch);
  mux2 #(32)  pcbrmux(pcplus4, pcbranch, pcsrc, pcnextbr);
  mux2 #(32)  pcmux(pcnextbr, {pcplus4[31:28], instr[25:0], 2'b00},
                    jump, pcnext);

  // register file logic
  regfile     rf(clk, regwrite, instr[25:21],
                 instr[20:16], writereg,
                 result, srca, writedata);
  mux2 #(5)   wrmux(instr[20:16], instr[15:11], regdst, writereg);
  mux2 #(32)  resmux(aluout, readdata, memtoreg, result);
  signext     se(instr[15:0], signimm);
  zeroext     ze(instr[15:0], zeroimm);

  // ALU logic
  mux4to1     srcbmux(writedata, signimm, zeroimm, alusrc, srcb);
  alu         alu(.A(srca), .B(srcb), .F(alucontrol), .Y(aluout), .zero(
endmodule
```

mipsparts.sv

```
module zeroext(input  logic [15:0] a,
               output logic [31:0] y);

  assign y = {16'b0, a};
endmodule

module signext(input  logic [15:0] a,
               output logic [31:0] y);

  assign y = {{16{a[15]}}, a};
endmodule
```

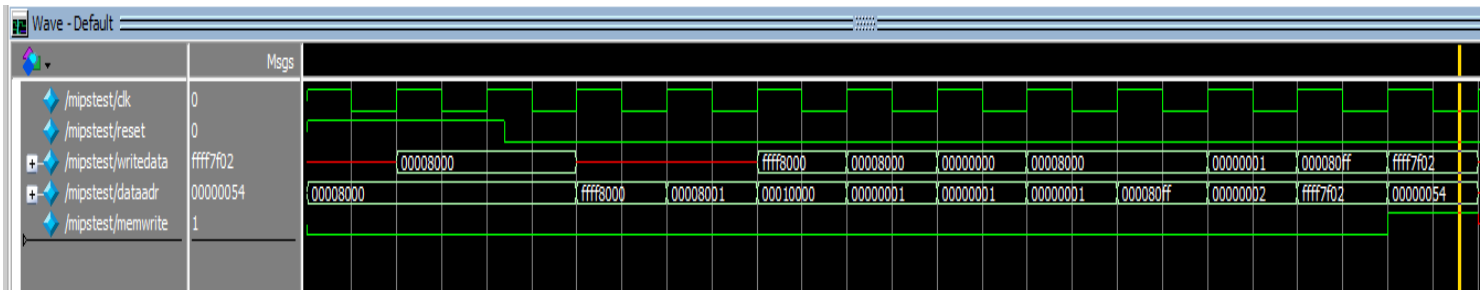
```
module mux4to1(
      input logic [31:0] A, B, C,
      input logic [1:0] S,
      output logic [31:0] Y);

   always_comb
   case(S)
      2'b00: Y = A;
      2'b01: Y = B;
      2'b10: Y = C;
      default: Y = 32'bx;
   endcase
endmodule
```

\

mipstest.sv

```
// Example testbench for MIPS processor

module mipstest();

  logic        clk;
  logic        reset;

  logic [31:0] writedata, dataadr;
  logic        memwrite;

  // instantiate device to be tested
  mipstop dut(clk, reset, writedata, dataadr, memwrite);

  // initialize test
  initial
    begin
      reset <= 1; # 22; reset <= 0;
    end

  // generate clock to sequence tests
  always
    begin
      clk <= 1; # 5; clk <= 0; # 5;
    end

  // check that 7 gets written to address 84
  always@(negedge clk)
    begin
      if(memwrite) begin
        if(dataadr === 84 & writedata === -33022) begin
          $display("Simulation succeeded");
          $stop;
        end else if (dataadr !== 82) begin
          $display("Simulation failed");
          $stop;
        end
      end
    end
endmodule
```

5. Simulation waveforms Part B




```
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# Simulation succeeded
# ** Note: $stop    : C:/intelFPGA_lite/18.1/ECE 469/mips/mipstest.sv(32)
#    Time: 125 ps  Iteration: 1  Instance: /mipstest
# Break in Module mipstest at C:/intelFPGA_lite/18.1/ECE 469/mips/mipstest.sv line 32
VSIM 2> run
```

Extended functionality. Main Decoder:

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump			
R-type	000000	1	1	0	0	0	0	10	0			
lw	100011	1	0	1	0	0	1	00	0			
sw	101011	0	X	1	0	1	X	00	0			
beq	000100	0	X	0	1	0	X	01	0			
addi	001000	1	0	1	0	0	0	00	0			
j	000010	0	X	X	X	0	X	XX	1			
ori	001101	1	0	1	0	0	0	11	0			
bne	000101	0	1	0	1	0	X	01	0			

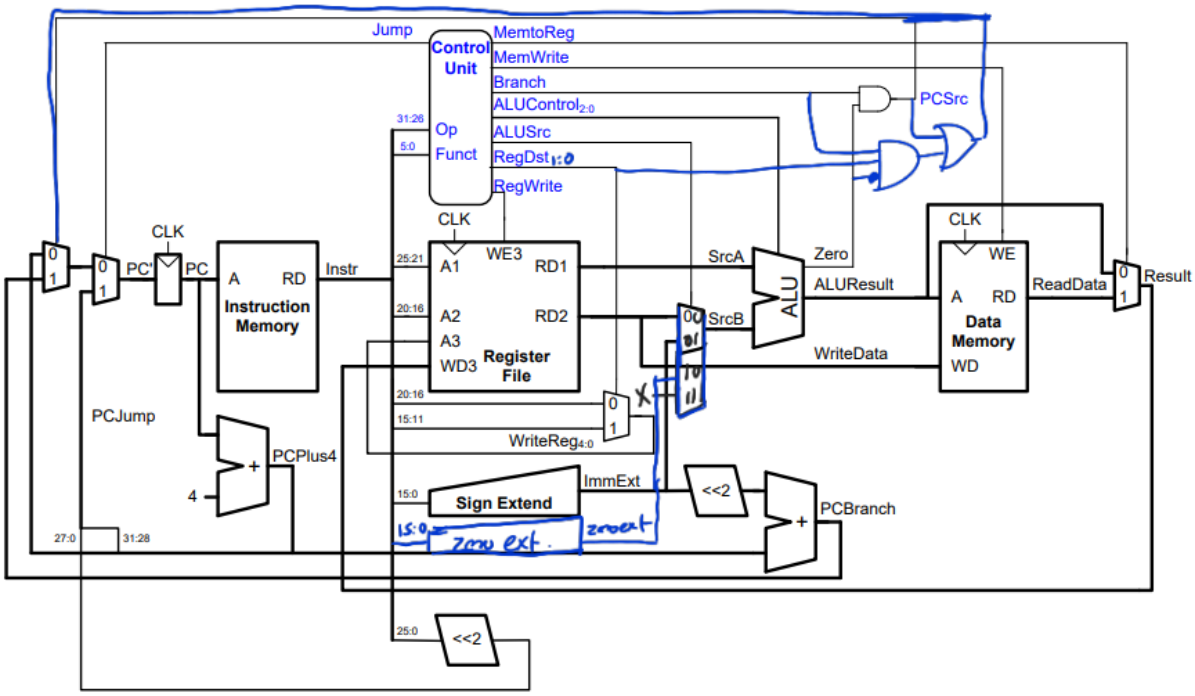
Extended functionality. ALU Decoder:

$ALUOp_{1:0}$	Meaning
00	Add
01	Subtract
10	Look at `funct` field
11	Ori

Lab Report 6
MIPS Single-cycle

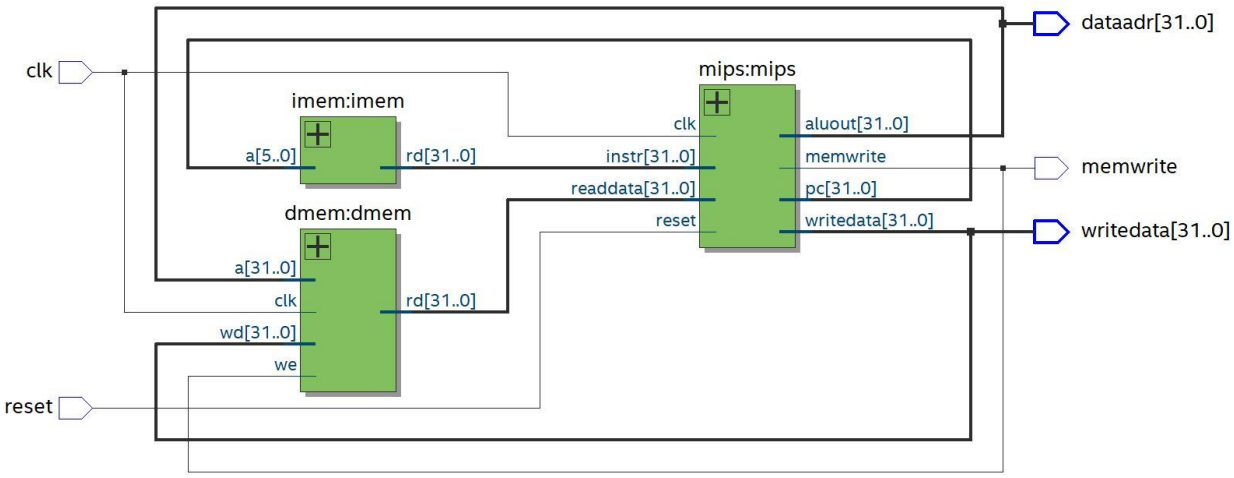
Overall:

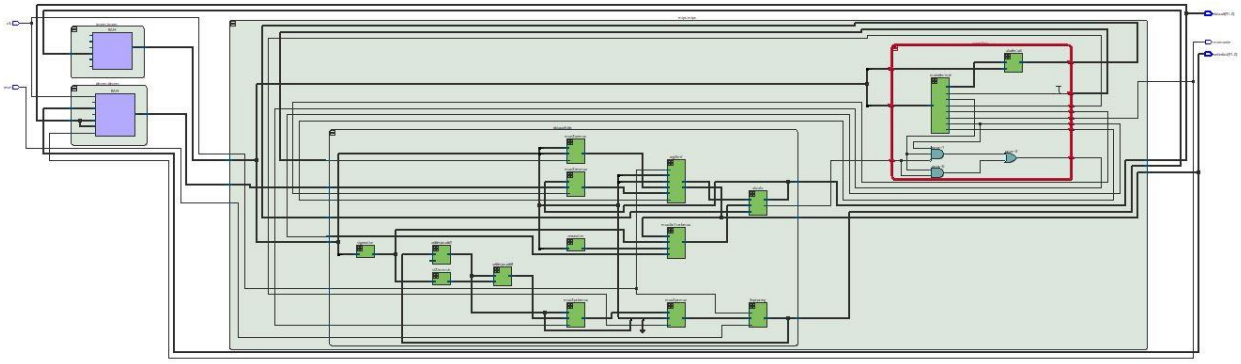
1. RTL view from Quartus' compilation result of your MIPS processor: before and after the modification. Identify and highlight the changes of your modification parts.

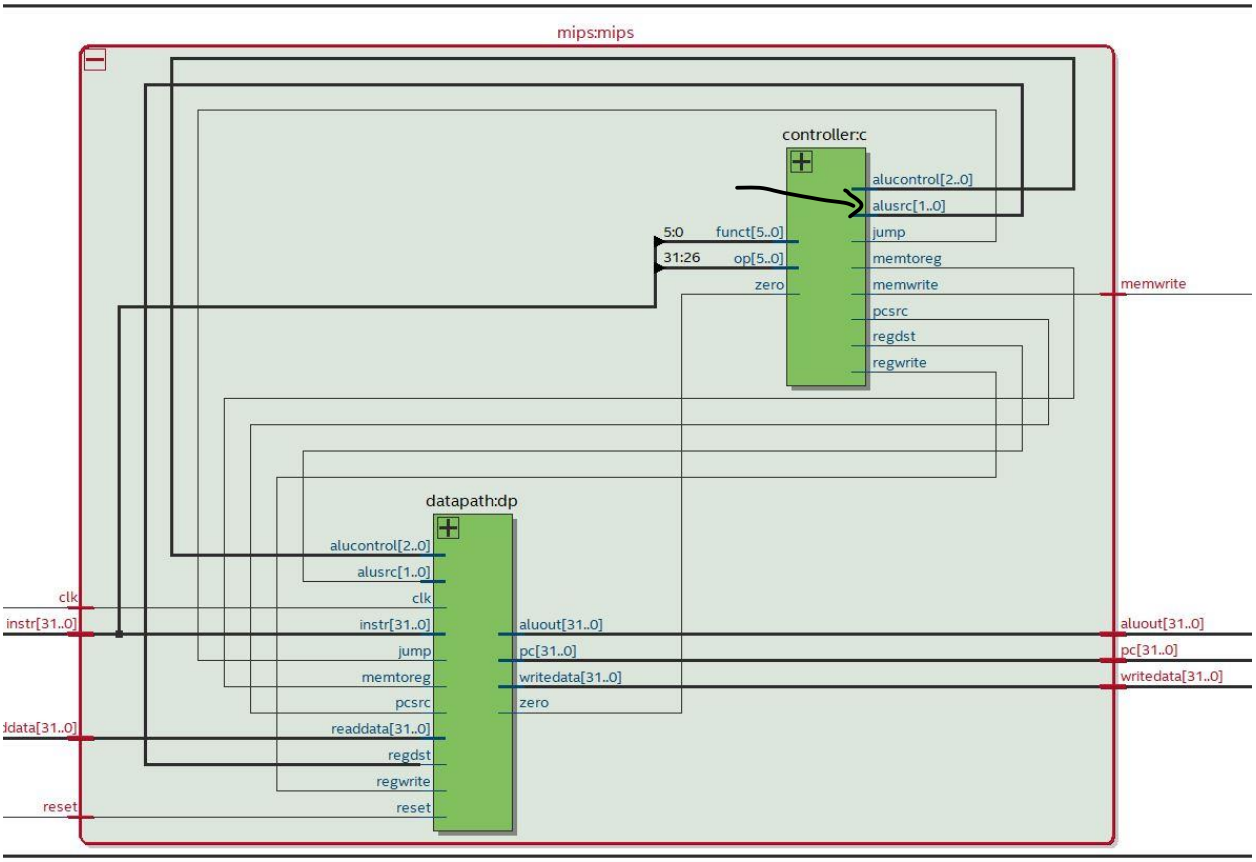


Single-cycle MIPS processor

Mipstop

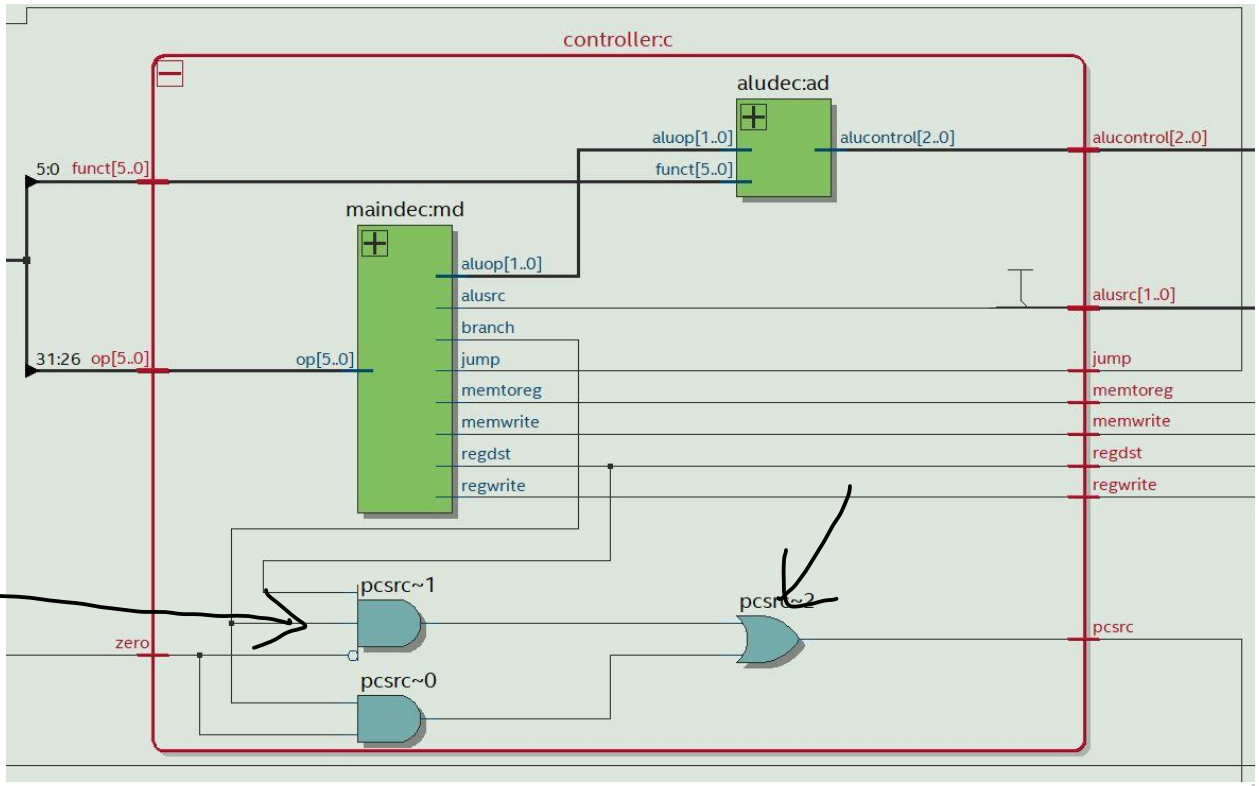


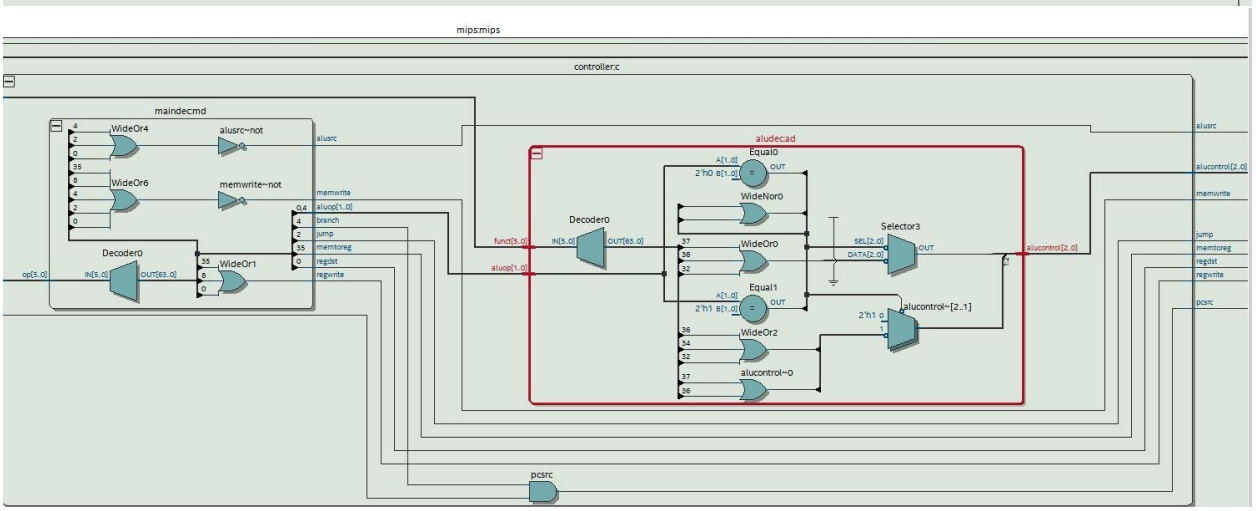



mips:mips
controller:c
alucontrol[2..0]
alusrc[1..0]
jump
memtoreg
memwrite
pcsrc
regdst
regwrite
5:0
funct[5..0]
31:26
op[5..0]
zero
memwrite
datapath:dp
alucontrol[2..0]
alusrc[1..0]
clk
instr[31..0]
jump
memtoreg
pcsrc
readdata[31..0]
regdst
regwrite
reset
aluout[31..0]
pc[31..0]
writedata[31..0]
zero
clk
instr[31..0]
ddata[31..0]
reset
aluout[31..0]
pc[31..0]
writedata[31..0]

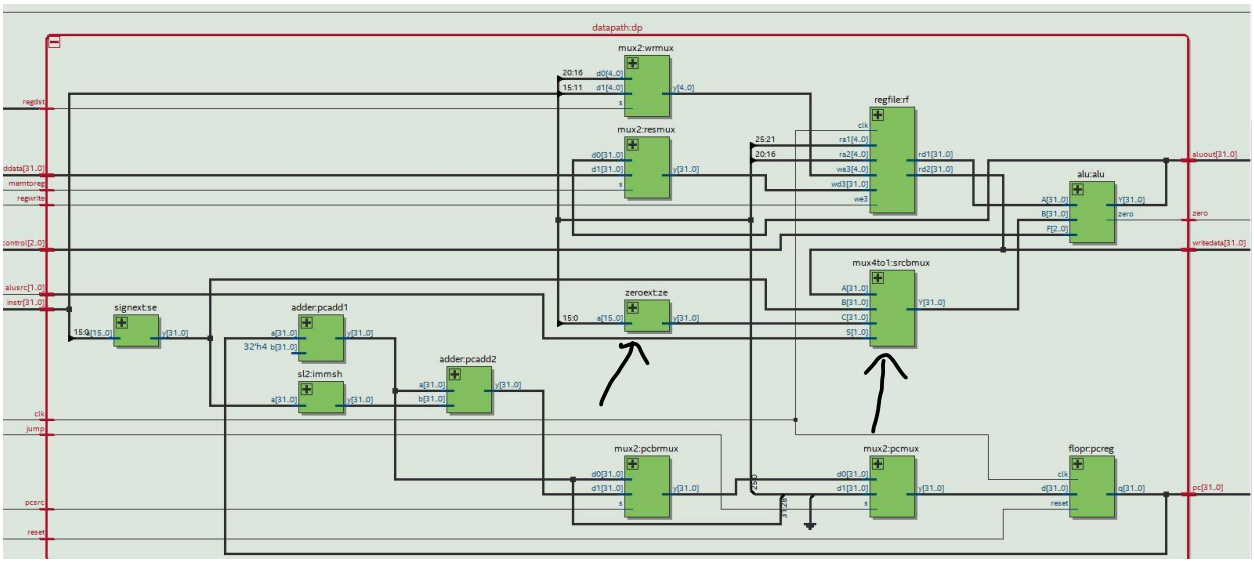
Lab Report 6
MIPS Single-cycle

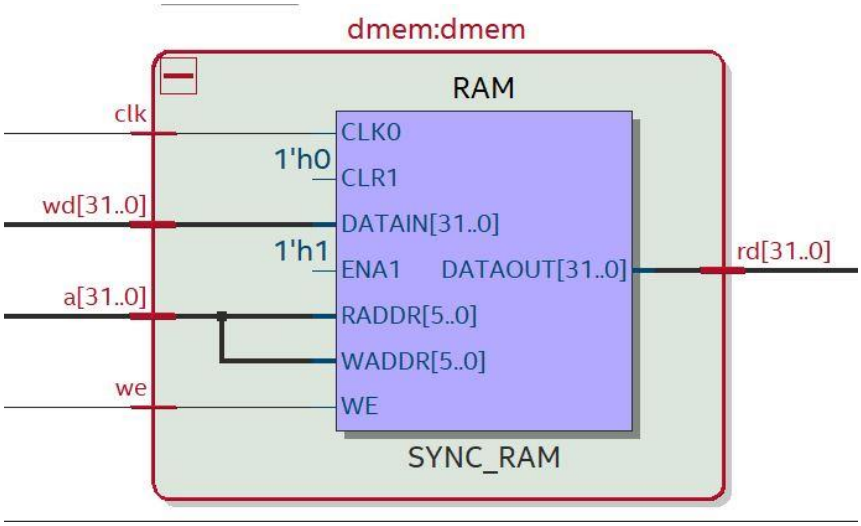
Controller

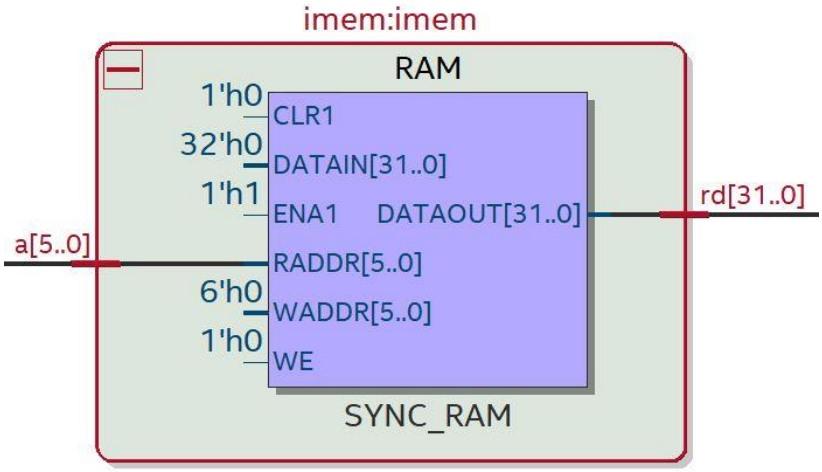




Datapath


datapath:dp
mux2:wrmux
mux2:resmux
regfile:rf
alu:alu
mux4to1:srcbmux
zeroext:ze
signext:se
adder:pcadd1
sl2:immsh
adder:pcadd2
mux2:pcbrmux
mux2:pcmux
flopr:pcreg
regdst
memtoreg
regwrite
alusrc[1..0]
instr[31..0]
clk
jump
pcsrc
reset
aluout[31..0]
zero
writedata[31..0]
pc[31..0]


dmem:dmem
RAM
clk
1'h0
CLK0
CLR1
wd[31..0]
DATAIN[31..0]
1'h1
ENA1
DATAOUT[31..0]
rd[31..0]
a[31..0]
RADDR[5..0]
WADDR[5..0]
we
WE
SYNC_RAM


imem:imem
RAM
1'h0
CLR1
32'h0
DATAIN[31..0]
1'h1
ENA1
DATAOUT[31..0]
rd[31..0]
a[5..0]
RADDR[5..0]
6'h0
WADDR[5..0]
1'h0
WE
SYNC_RAM

2. Workload report:
   How many hours have you spent for this lab?
   - More than a week (avg 4 hours every day during this time)

   Which activity takes the most significant amount of time? This will not affect your grade (unless omitted).
   - Figuring out the code, connections other than logic part was fine.