ECE 469 HDL
Project 2, Part 1
Tamanna Ravi Rupani
UIN: 665679988
MIPS

1. A completed version of Table 1.

cycle	reset	pc	instr	branch	srca	srcb	alu out	zero	pcsrc	write data	mem write	read data
1	1	00	addi $2, $0,5 20020005	0	0	5	5	0	0	0	0	x
2	0	04	addi $3, $0,12 2003000c	0	0	C	c	0	0	0	0	x
3	0	08	addi $7, $3, -9 2067fff7	0	C	-9	3	0	0	0	0	x
4	0	0C	or $4, $7, $2 00e220225	0	3	5	7	0	0	0	0	x
5	0	10	and $5, $3, $4 00642824	0	C	7	4	0	0	0	0	x
6	0	14	add $5, $5, $4 00a42820	0	4	7	b	0	0	0	0	x
7	0	18	beq $5, $7, end 10a7000a	1	B	3	8	0	0	0	0	x
8	0	1C	slt $4, $3, $4 0064202a	0	C	7	0	1	0	0	0	x
9	0	20	beq $4, $0, around 10800001	1	0	0	0	1	1	0	0	x
	0	24	addi $5, $0,0 20050000									
10	0	28	slt $4, $7, $2 00e2202a	0	3	5	1	0	0	0	0	x
11	0	2C	add $7, $4, $5 00853820	0	1	B	c	0	0	0	0	X
12	0	30	sub $7, $7, $2 00e23822	0	C	5	7	0	0	0	0	X
13	0	34	sw $7,68($3) ac670044	0	C	[80]	[80] = 7	0	0	7	1	X
14	0	38	lw $2,80($0) 8c020050	0	0	[80]	[80]	0	0	0	0	7
15	0	3C	j end 08000011	0	X	X	x	x	x	0	0	X
	0	40	addi $2, $0,1 20020001									
16	0	44	sw $2,84($0) ac020054	0	0	[84]	[84] =7	0	0	7	1	X

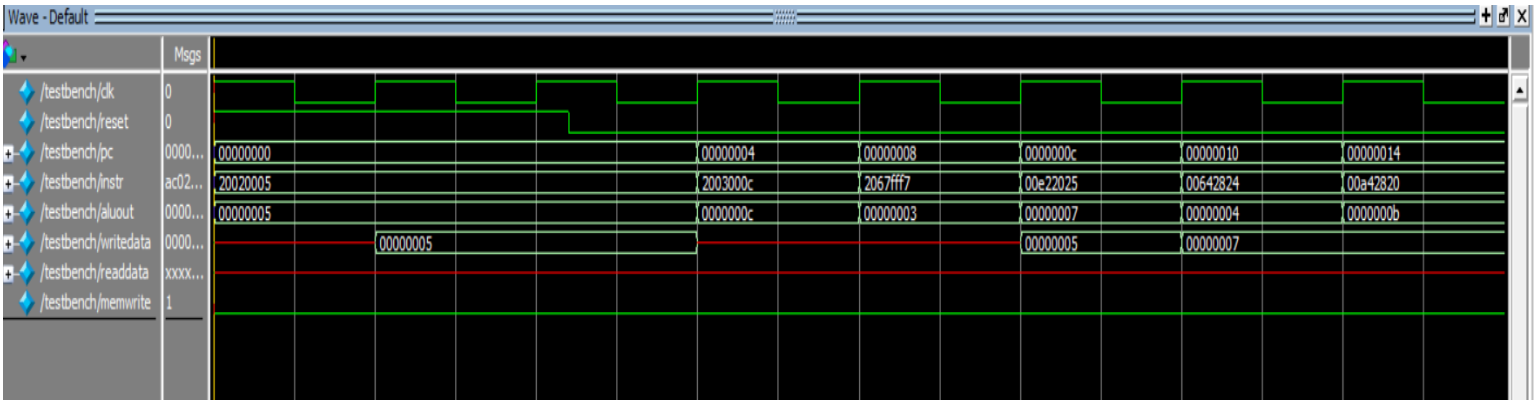
2. An image of the simulation waveforms showing correct operation of the unmodified processor. Does it write the correct value to address 84?

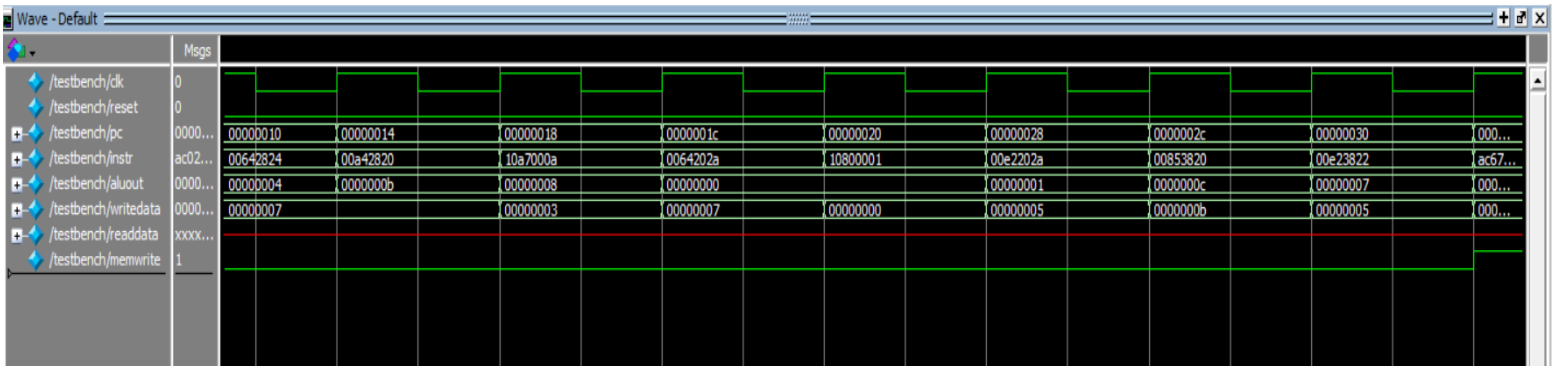
Yes, the output 7 is successfully written to address 84 and the simulation is successful.

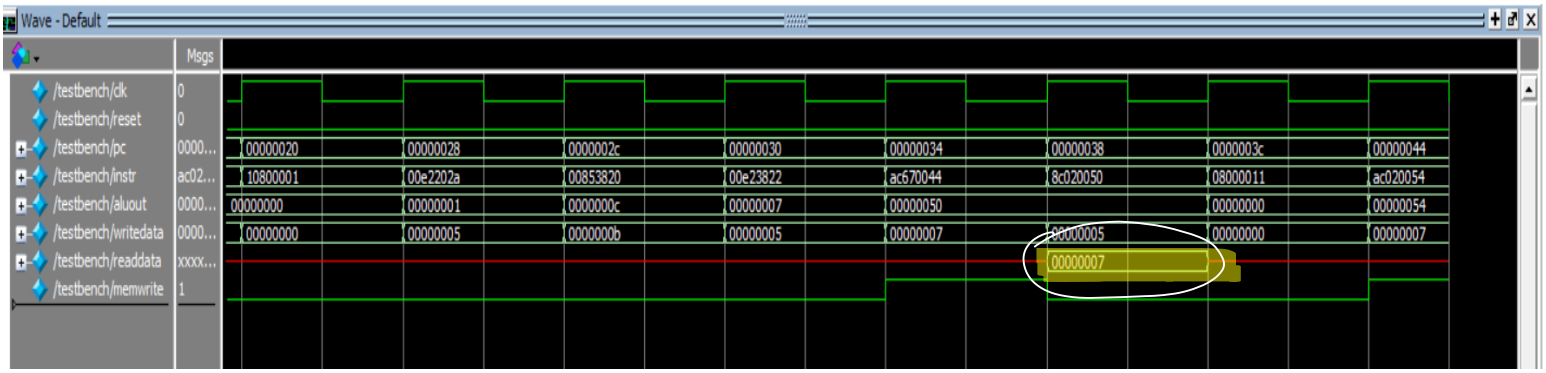
```
# ** Warning: (vsim-WLF-5000) WLF file currently in use: vsim.wlf
#           File in use by: taman  Hostname: DESKTOP-POIKV0U  ProcessID: 5144
#           Attempting to use alternate WLF file "./wlft2zj6g6".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
#           Using alternate file: ./wlft2zj6g6
VSIM 6> run
# Simulation succeeded
# ** Note: $stop    : C:/intelFPGA/17.1/testbench12.sv(24)
#    Time: 175 ps  Iteration: 1  Instance: /testbench
# Break in Module testbench at C:/intelFPGA/17.1/testbench12.sv line 24

VSIM 7>
```

Ln: 1 Col: 0 | 99 ps to 179 ps | Project : hdlproject2

Type here to search

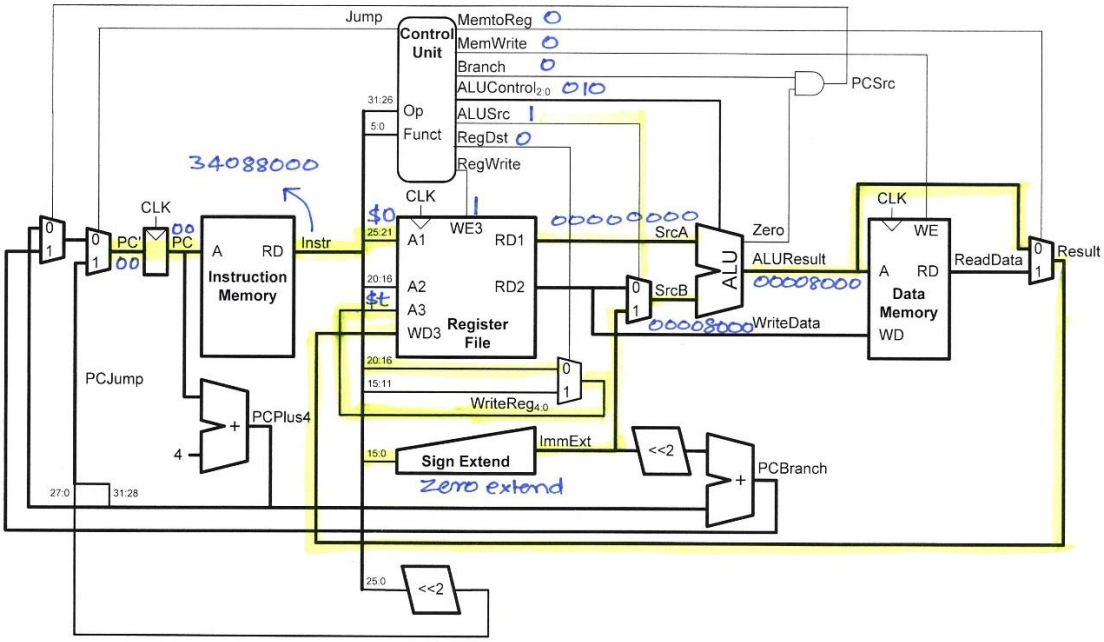






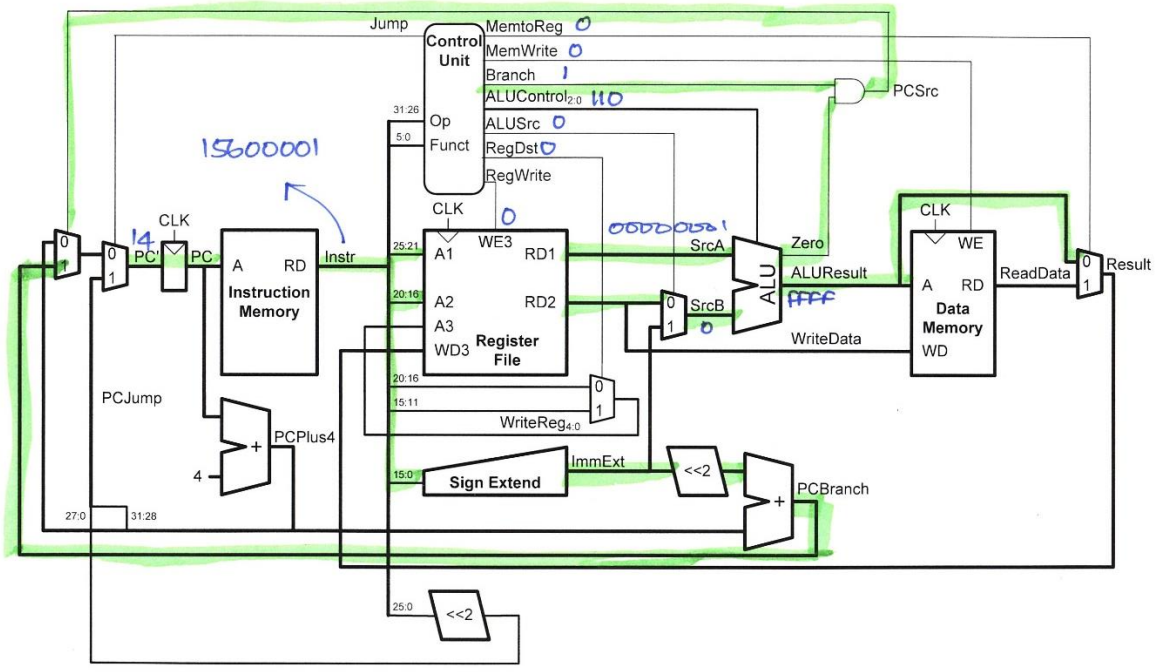
3. Marked up versions of the datapath schematic and decoder tables that adds the ori and bne instructions

Marked up version for ori



Single-cycle MIPS processor

Marked up version for bne



Single-cycle MIPS processor

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	X	0	0	10	0
Bne	000101	0	X	0	1	0	X	01	0

Extended functionality. ALU Decoder

$ALUOp_{1:0}$	Meaning
00	Add
01	Subtract
10	Look at Funct field
11	n/a

4. Your SystemVerilog code for your modified MIPS computer (including ori and bne functionality) with the changes highlighted and commented in the code.

SYSTEM VERILOG CODE FOR MODIFIED MIPS PROCESSOR

```
module top(input logic clk, reset,
        output logic [31:0] writedata, aluout,
        output logic memwrite,
        output logic [31:0] pc, instr, readdata);
                mips mips(clk, reset, pc, instr, memwrite, aluout, writedata, readdata);
                imem imem(pc[7:2], instr);
                dmem dmem(clk, memwrite, aluout, writedata, readdata);

endmodule

module dmem(input logic clk, we,
         input logic [31:0] a, wd,
         output logic [31:0] rd);

         logic [31:0] RAM[63:0];
         assign rd=RAM[a[31:2]];

         always_ff @(posedge clk)
         if (we) RAM[a[31:2]] <= wd;
```

```
endmodule

module imem(input logic [5:0] a,
	output logic [31:0] rd);

	logic [31:0] RAM[63:0];
	initial $readmemh("C:/intelFPGA/17.1/memfile2.dat", RAM);
	assign rd=RAM[a];
endmodule

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

	controller c(instr[31:26], instr[5:0], zero,aluout, memtoreg, memwrite, pcsrc, alusrc, regdst, regwrite, jump,
alucontrol); //aluout is added her for the modified code
	datapath dp(clk, reset, instr[31:26], 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,
		input logic [31:0] aluout, //This part is added for modification
		output logic memtoreg, memwrite,
		output logic pcsrc, alusrc,
		output logic regdst, regwrite,
		output logic jump,
		output logic [2:0] alucontrol);

	logic [1:0] aluop;
	logic branch;
	logic fun1;
	logic z;
	logic [5:0] fun2;

//This part is added for modification
always_comb
if ( op === 6'b001101)
```

```
fun1 = 1;
else
fun1 = 0;

always_comb
if ( fun1 === 1'b0 )
fun2 = funct;
else
fun2 = 6'b100101;

maindec md(op, memtoreg, memwrite, branch, alusrc, regdst, regwrite, jump, aluop);
aludec ad(fun2, aluop, alucontrol);

//This part is added for modification
always_comb
if ( op === 6'b000101 & aluout !== 32'b00000000000000000000000000000000 )
z = 1'b1;
else z = zero;

assign pcsrc = branch & z;

endmodule

module maindec(input logic [5:0] op,
               output logic memtoreg, memwrite,
               output logic branch, alusrc,
               output logic regdst, regwrite,
               output logic jump,
               output logic [1:0] aluop);

logic [8:0] controls;

assign {regwrite, regdst, alusrc, branch, memwrite, memtoreg, jump, aluop}=controls;
always_comb
case(op)
6'b000000: controls <= 9'b110000010; // RTYPE
6'b100011: controls <= 9'b101001000; // LW
6'b101011: controls <= 9'b001010000; // SW
6'b000100: controls <= 9'b000100001; // BEQ
6'b000101: controls <= 9'b000100001; // BNE
6'b001000: controls <= 9'b101000000; // ADDI
6'b001101: controls <= 9'b101000010; // ORI
6'b000010: controls <= 9'b000000100; // J
default: controls <= 9'bxxxxxxxxx; // illegal op
endcase
```

```
endmodule
module aludec(input logic [5:0] funct,
              input logic [1:0] aluop,
              output logic [2:0] alucontrol);

always_comb
case(aluop)
2'b00: alucontrol <= 3'b010; // add (for lw/sw/addi)
2'b01: alucontrol <= 3'b110; // sub (for beq)
default: case(funct) // R-type instructions
          6'b100000: alucontrol <= 3'b010; // add
          6'b100010: alucontrol <= 3'b110; // sub
          6'b100100: alucontrol <= 3'b000; // and
          6'b100101: alucontrol <= 3'b001; // or
          6'b101010: alucontrol <= 3'b111; // slt
          default: alucontrol <= 3'bxxx; // ???
          endcase
endcase

endmodule

module datapath(input logic clk, reset,
                input logic [5:0] opcode,
                input logic memtoreg, pcsrc,
                input logic alusrc, regdst,
                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;
logic [31:0] srca, srcb;
logic [31:0] result;
logic fun1;

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);
```

```
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);


//This part is added for modification
always_comb
if ( opcode === 6'b001101)
fun1 = 1;
else
fun1 = 0;

always_comb
if( fun1 === 1'b1)
signimm={16'b0, instr[15:0]};
else
signimm={{16{instr[15]}}, instr[15:0]};

//signext se(instr[15:0], signimm);
mux2 #(32) srcbmux(writedata, signimm, alusrc, srcb);
alu alu(srca, srcb, alucontrol, aluout, zero);

endmodule

module alu(input logic [31:0] a, b,
           input logic [2:0] F,
           output logic [31:0] Y,
           output logic zero);
logic [31:0] S , B;

assign B = F[2]? ~b:b;

assign S = a + B + F[2];

always_comb
case(F[1:0])
2'b00 : Y = a & B;
2'b01 : Y = a | B;
2'b10 : Y = S;
2'b11 : Y = {31'b0,S[31]};
endcase

always_comb
if ( Y === 32'b0)
zero = 1;
```

```
else zero =0;
endmodule
module regfile(input logic clk,
               input logic we3,
               input logic [4:0] ra1, ra2, wa3,
               input logic [31:0] wd3,
               output logic [31:0] rd1, rd2);

logic [31:0] rf[31:0];

always_ff @(posedge clk)
if (we3)
rf[wa3] <= wd3;

assign rd1=(ra1 !=0) ? rf[ra1] : 0;
assign rd2=(ra2 !=0) ? rf[ra2] : 0;

endmodule

module adder(input logic [31:0] a, b,
             output logic [31:0] y);

assign y=a+b;

endmodule

module sl2(input logic [31:0] a,
           output logic [31:0] y);

assign y={a[29:0], 2'b00};

endmodule

module flopr #(parameter WIDTH=8) (input logic clk, reset,
                                   input logic [WIDTH-1:0] d,
                                   output logic [WIDTH-1:0] q);
always_ff @(posedge clk, posedge reset)
if (reset) q <= 0;
else q <= d;
endmodule

module mux2 #(parameter WIDTH=8) (input logic [WIDTH-1:0] d0, d1,
                                  input logic s,
                                  output logic [WIDTH-1:0] y);
assign y=s ? d1 : d0;
endmodule
```

TESTBENCH CODE FOR THE MODIFIED MIPS PROCESSORS

```
module testbenchmod();
logic clk; logic reset;
logic [31:0] writedata, aluout;
logic memwrite;
logic [31:0] pc, instr, readdata;
topmod dut (clk, reset, writedata, aluout, memwrite,pc, instr, readdata);
initial
begin
reset <= 1; # 22; reset <= 0;
end

always
begin
clk <= 1; # 5; clk <= 0; # 5;
end

always @(negedge clk)
begin
        if (memwrite)
        begin
                if (aluout===84)
                begin
                $display("Simulation succeeded");
                $stop;
                end
                else if (aluout !==80)
                begin
                $display("Simulation failed");
                $stop;
                end
        end
end

endmodule
```

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

34088000

20098000

350A8001

11090005

0128582A

15600001

08000009

01485022

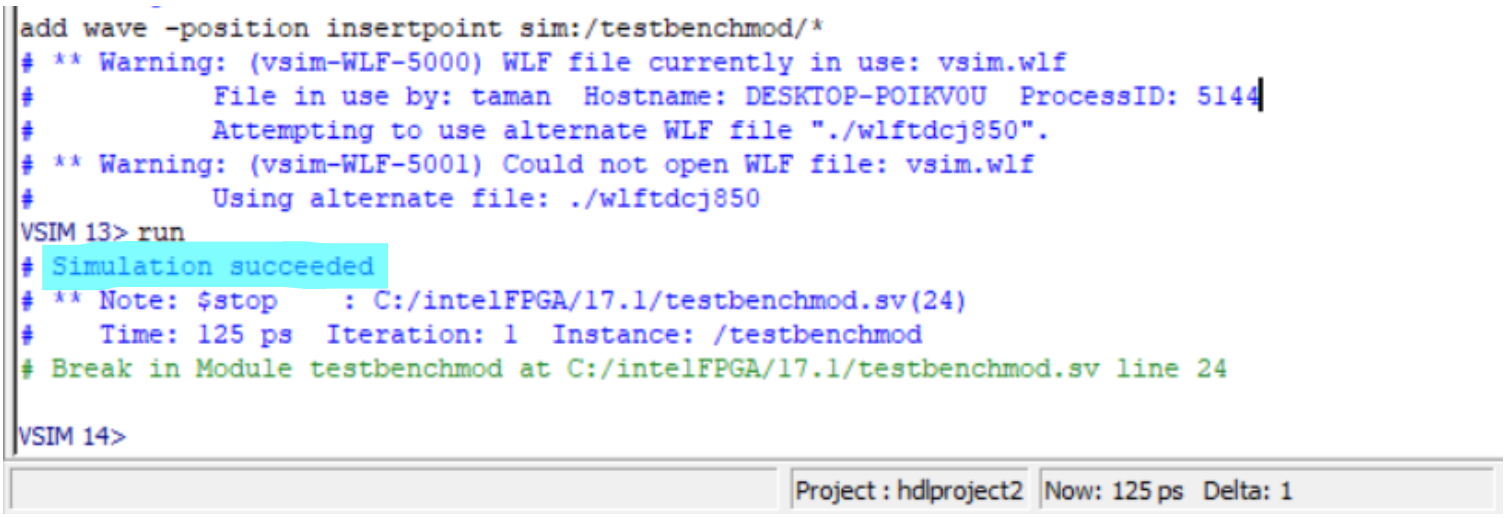
350800FF

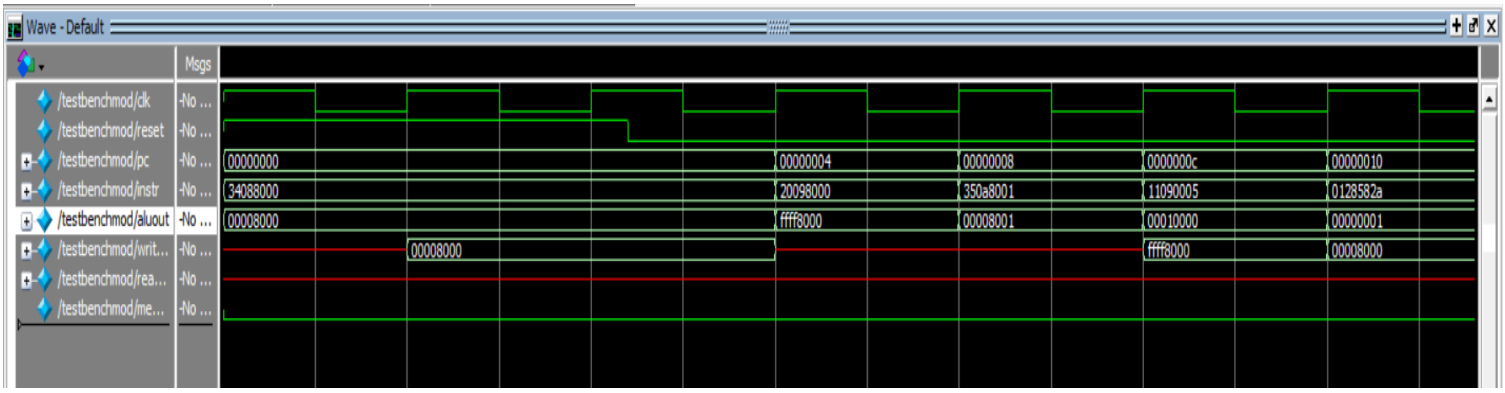
016A5820

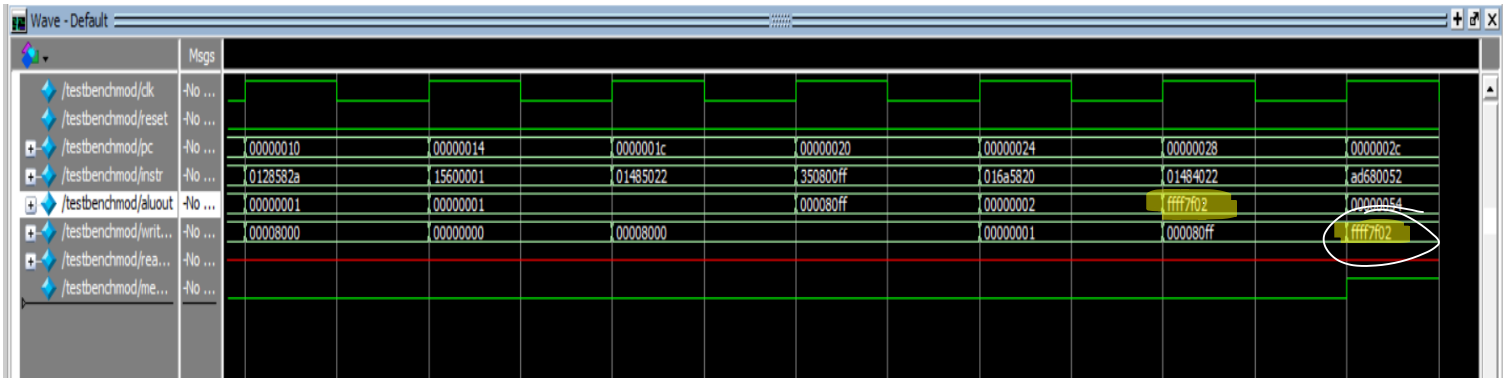
01484022

AD680052

6. An image of the simulation waveforms showing correct operation of your modified processor on the new program. What address and data value are written by the sw instruction?







Address and data value are written by the sw instruction is,

Address: 00000084

Data: ffff7f02

7. Workload report: How many hours have you spent on this part? Which activity takes the most significant amount of time? This will not affect your grade (unless omitted).

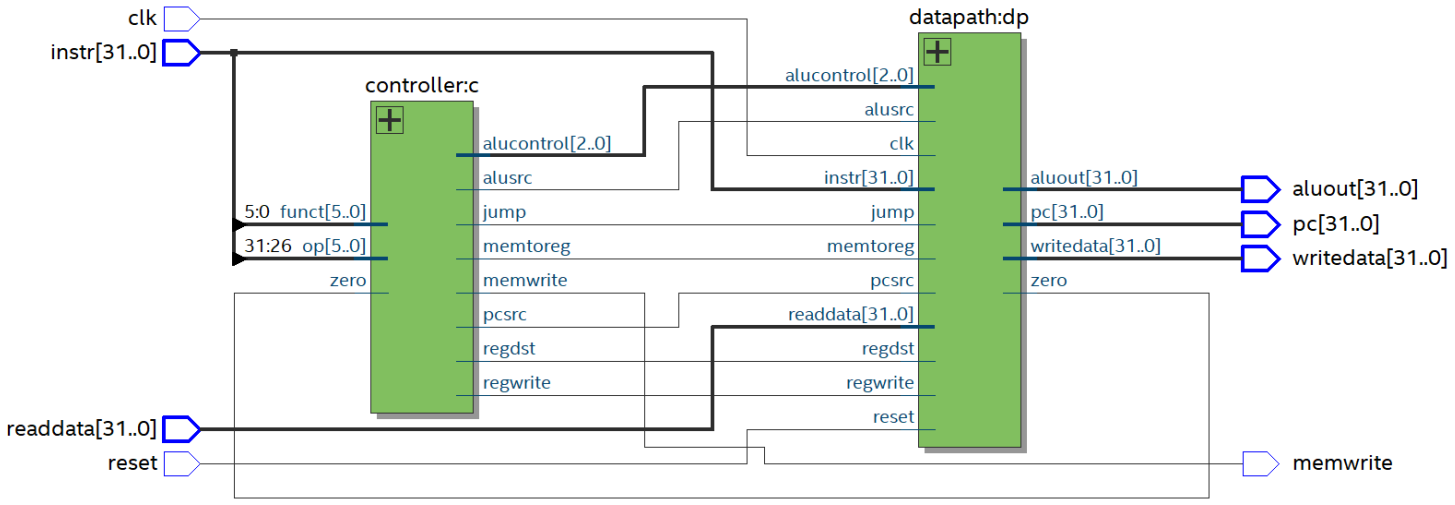
Activity	Time
Studying chapter 6 and 7, understanding MIPS concept	6 hours
Filling the table 1, decoder table, alu table	2 hours
Understanding the SystemVerilog code	2-3 hours
Execution of SystemVerilog code and debugging errors	3+ hours
Modifying new machine language code	2-3 hours
New SystemVerilog code for modified MIPS	2 hours
Execution of new modified code	2 hours
Quartus part	1 hour
Making the report	4 hours
Total time	24+ hours

The activity of executing and debugging the processor code takes the most amount of time, along with modifying the code for the new modified processor.

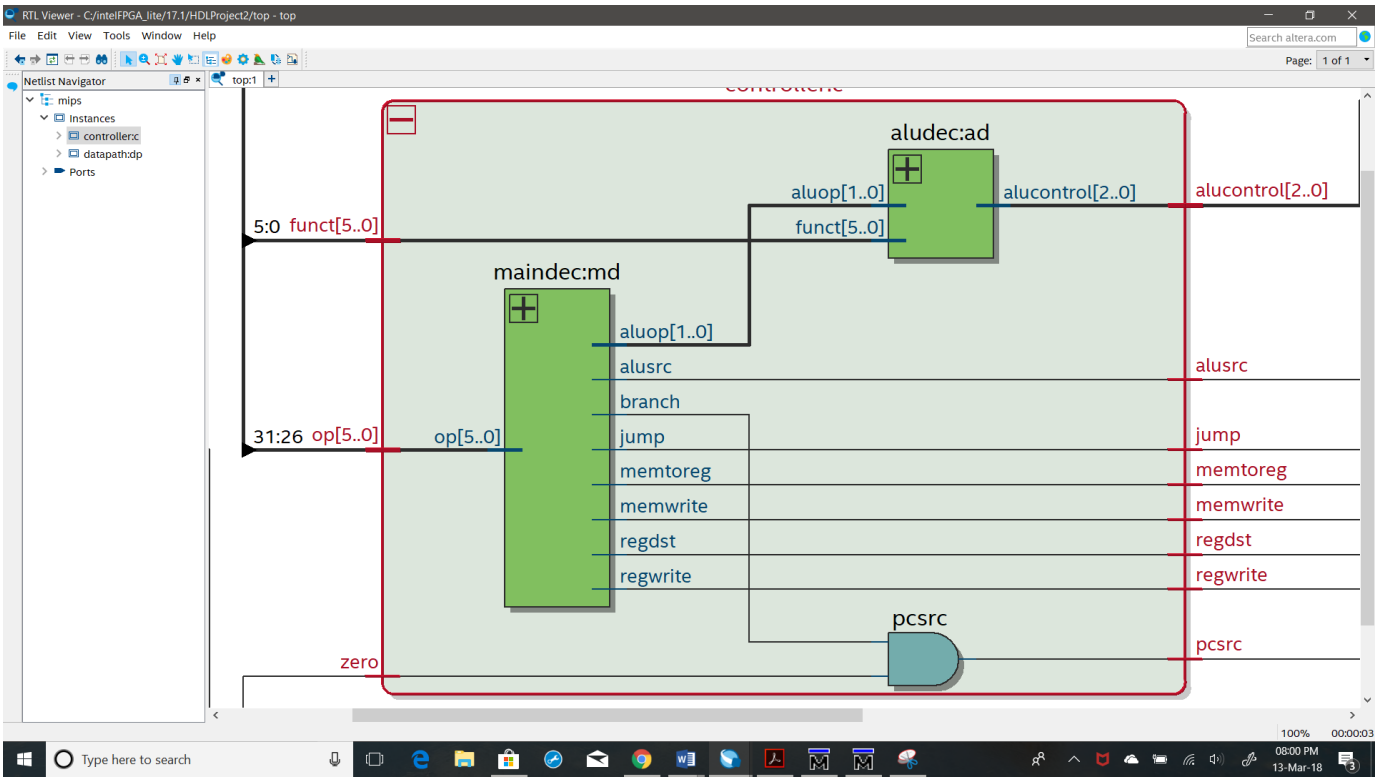
8. (extra credit) RTL view from Quartus' compilation result of your MIPS processor: before and after the modification. Can you identify the changes?

Before Modification

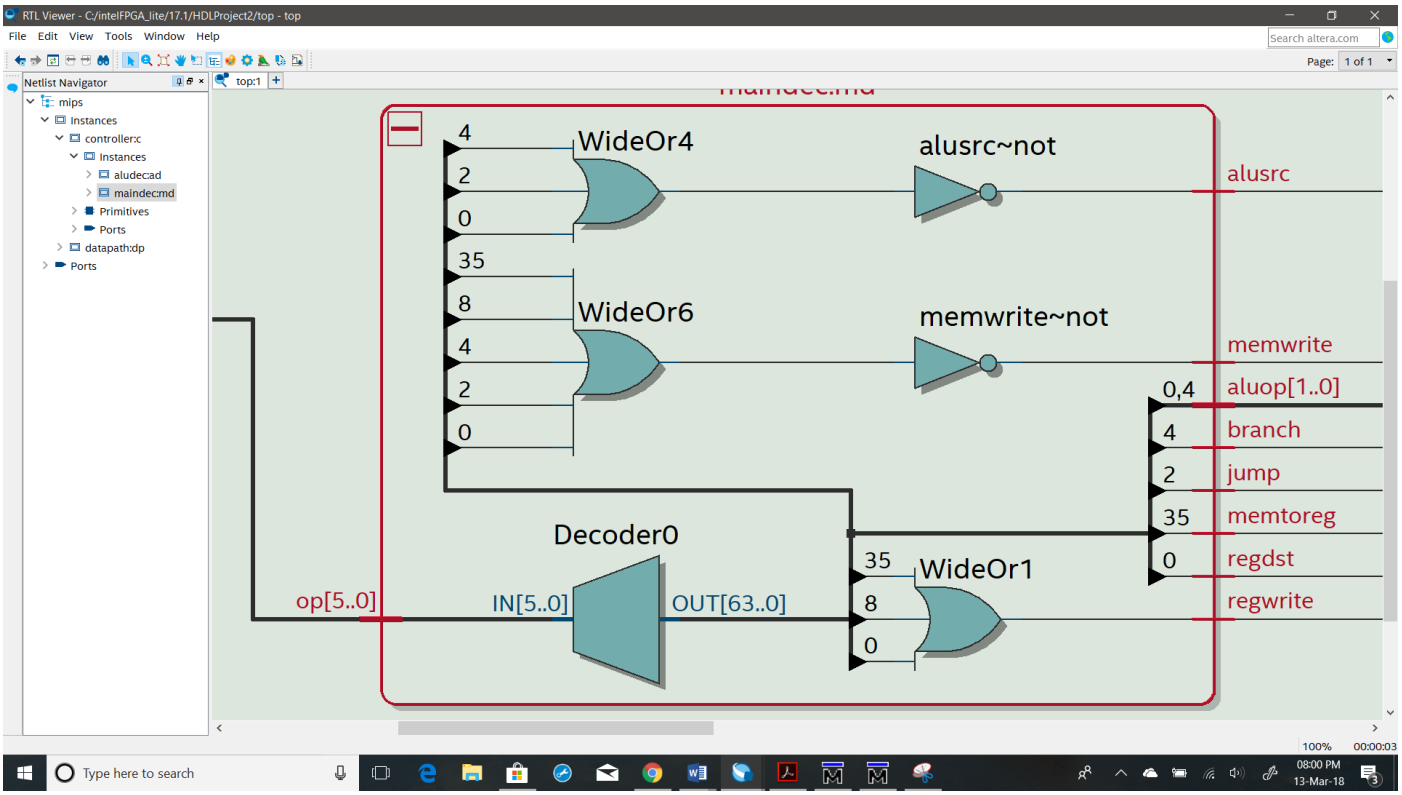
Normal view



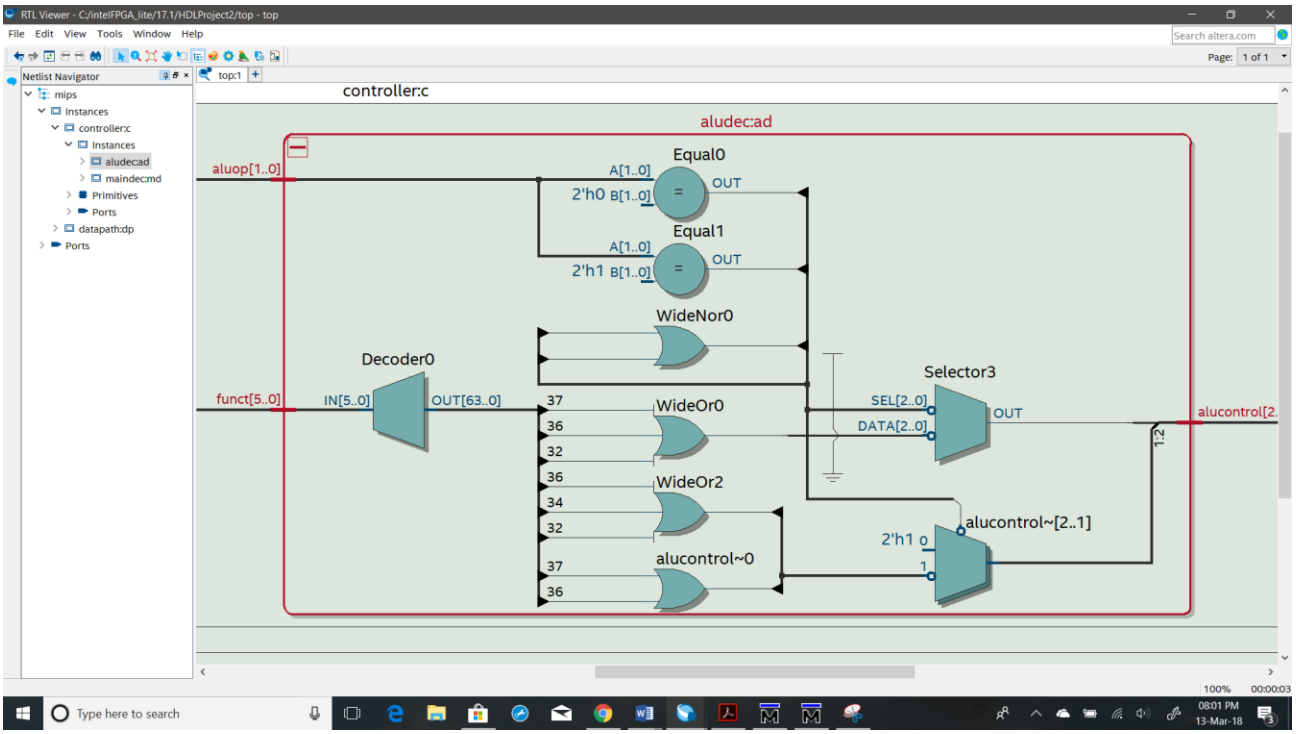
Inside controller:c



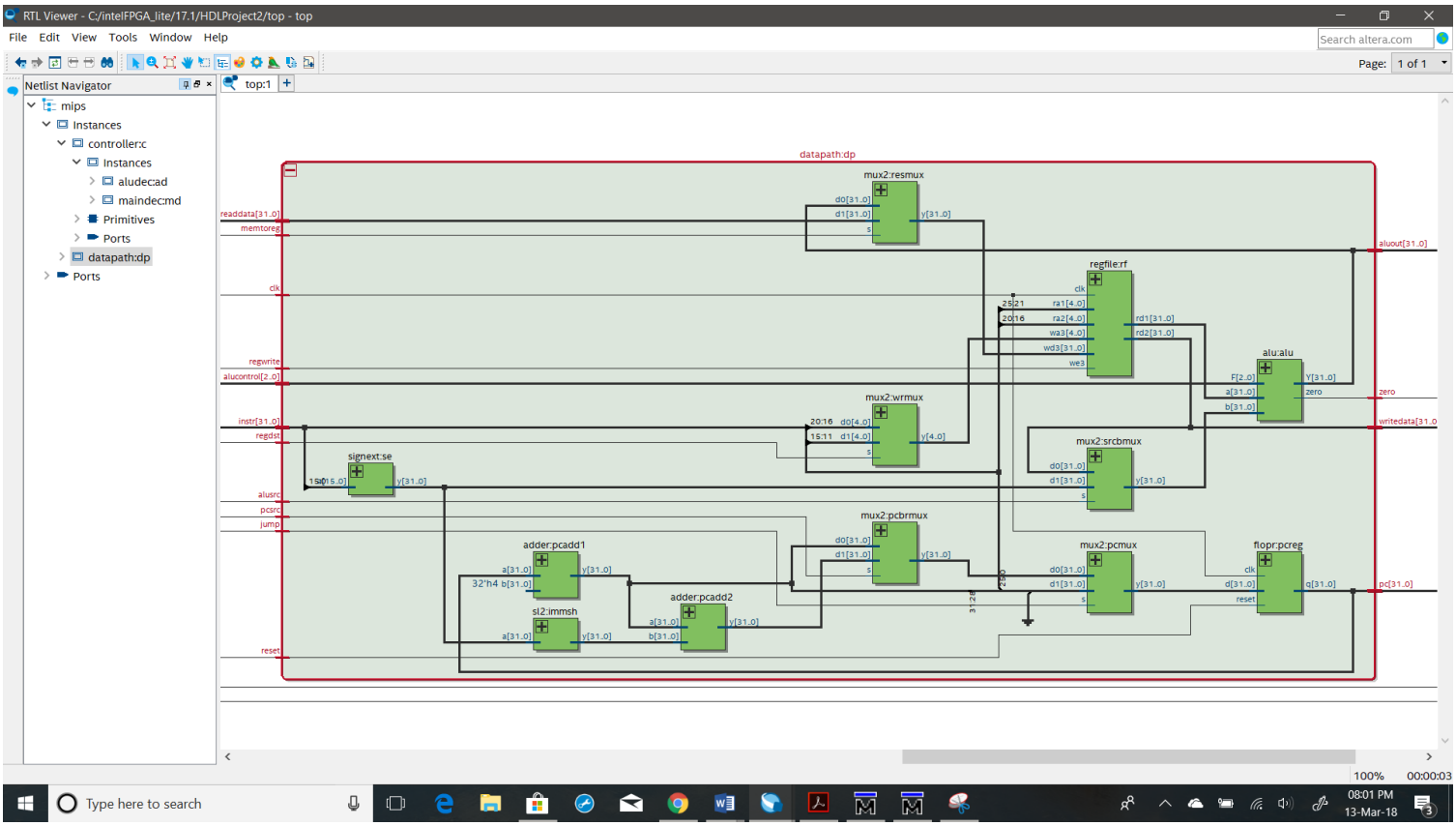
Inside maindec:md



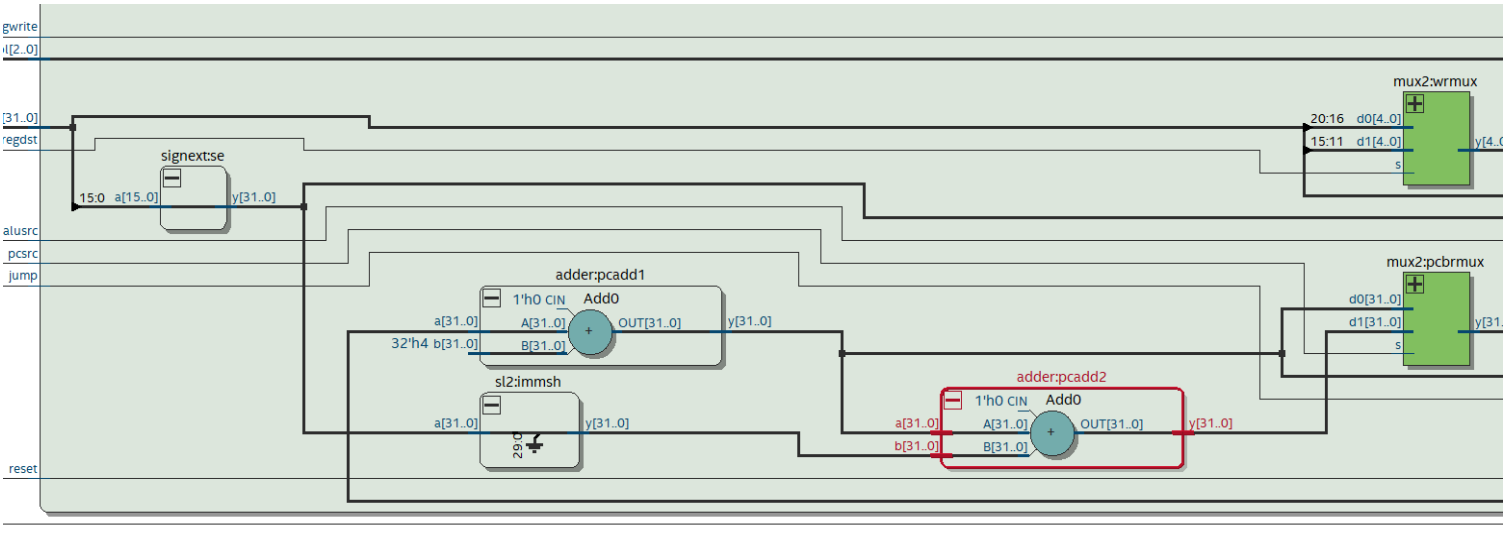
Inside aludec:ad

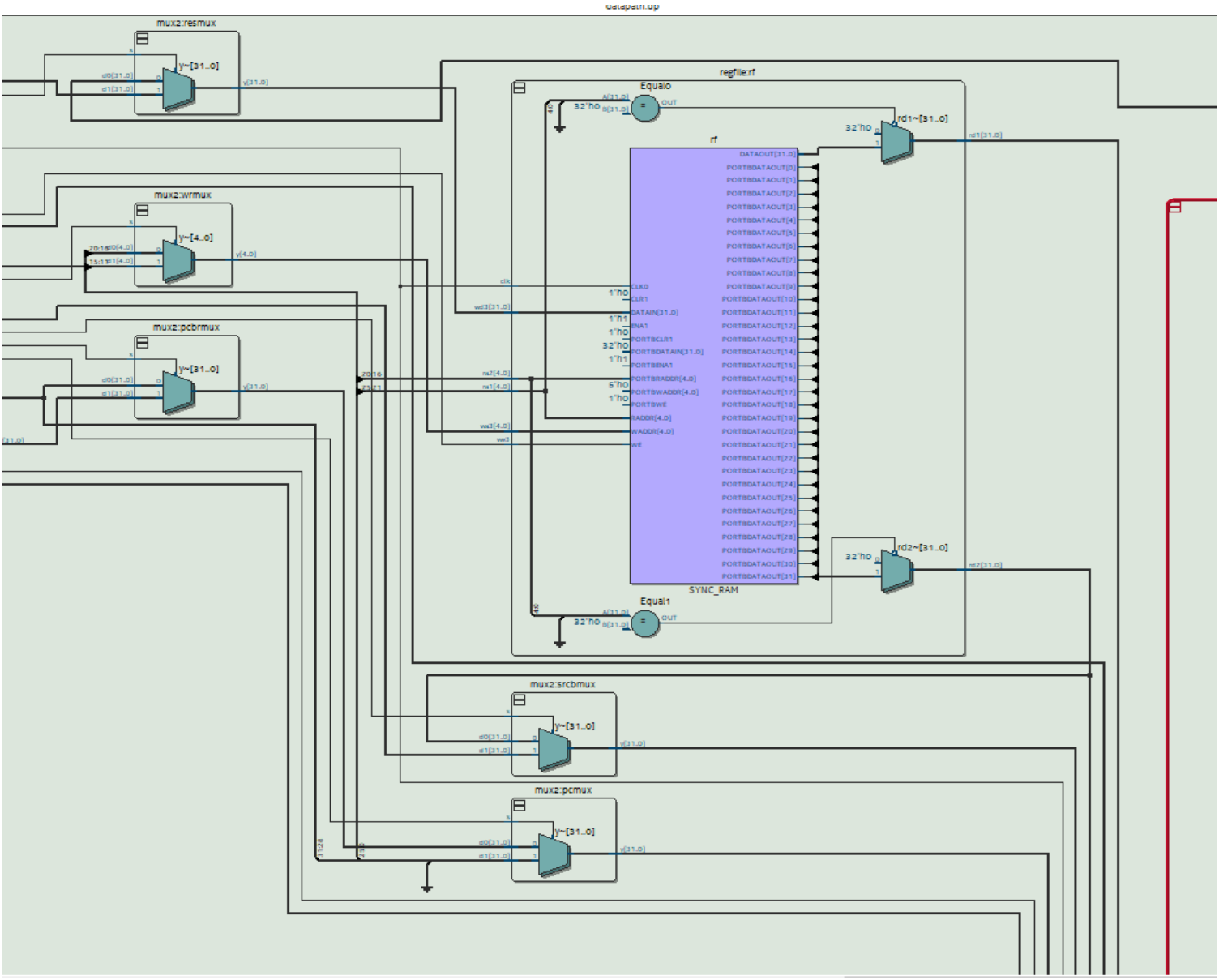


Inside datapath:dp

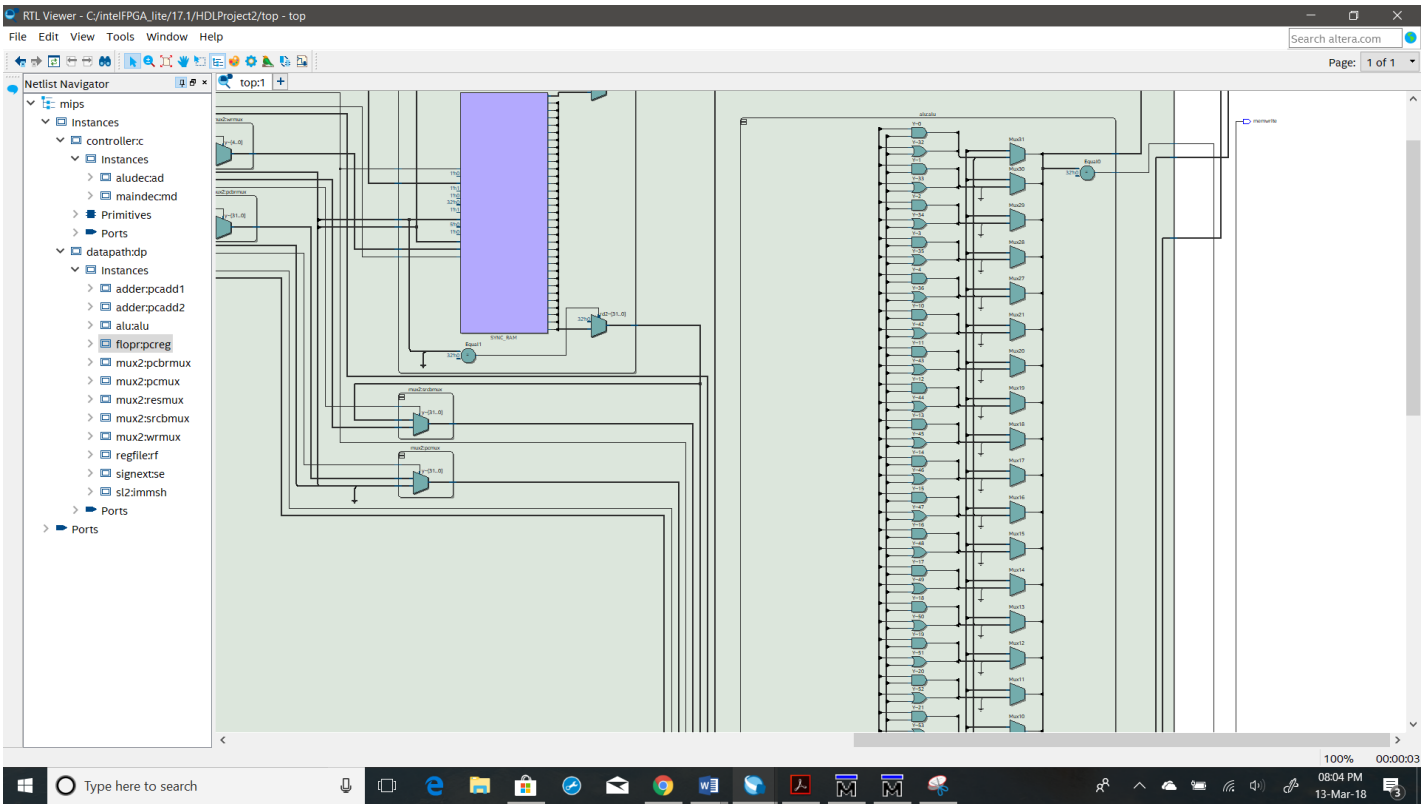


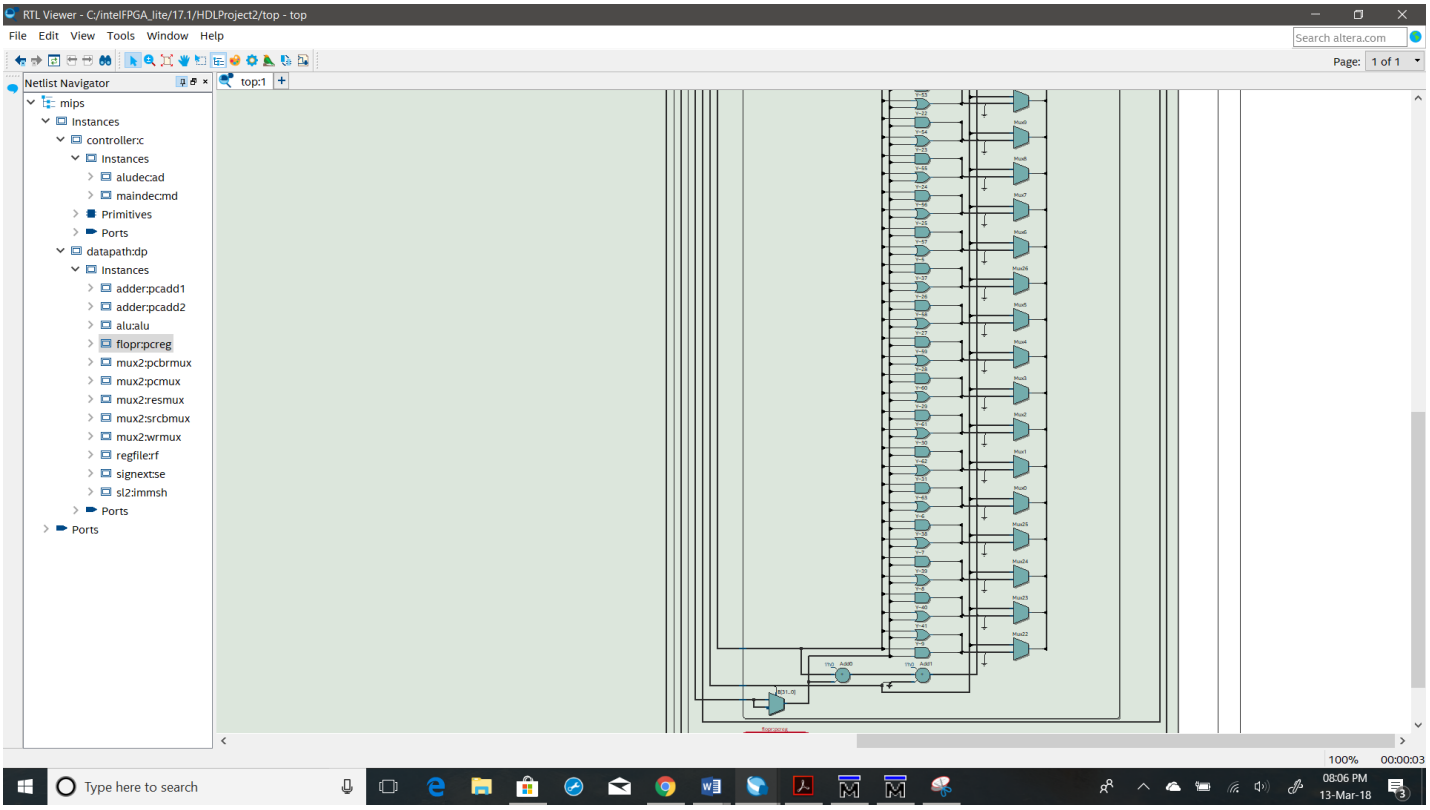
Zoomed view inside datapath modules



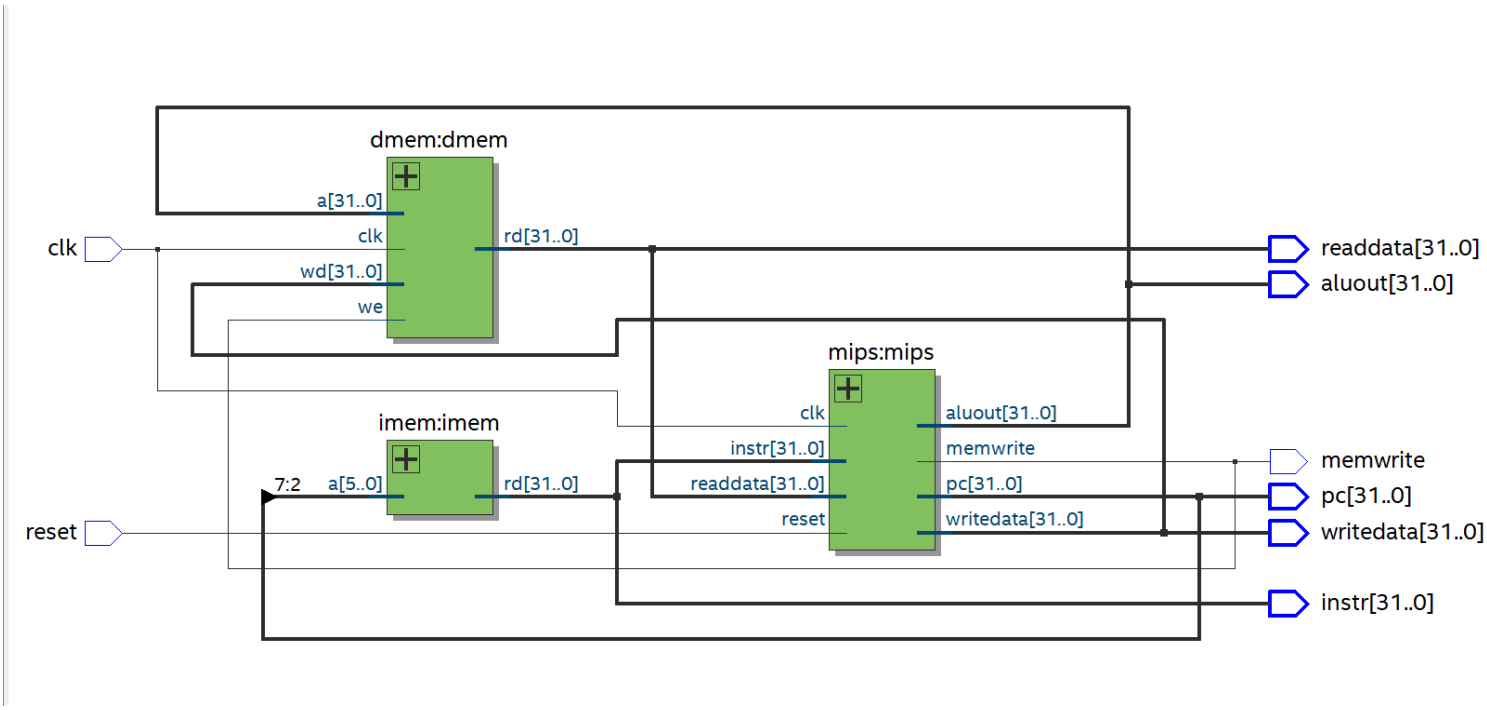

datapath.bp
mux2:resmux
y~[31..0]
regfile:rf
Equal0
rf
rd1~[31..0]
mux2:wrmux
y~[4..0]
mux2:pcbrmux
y~[31..0]
SYNC_RAM
Equal1
rd2~[31..0]
mux2:srcbmux
y~[31..0]
mux2:pcmux
y~[31..0]

Inside alu

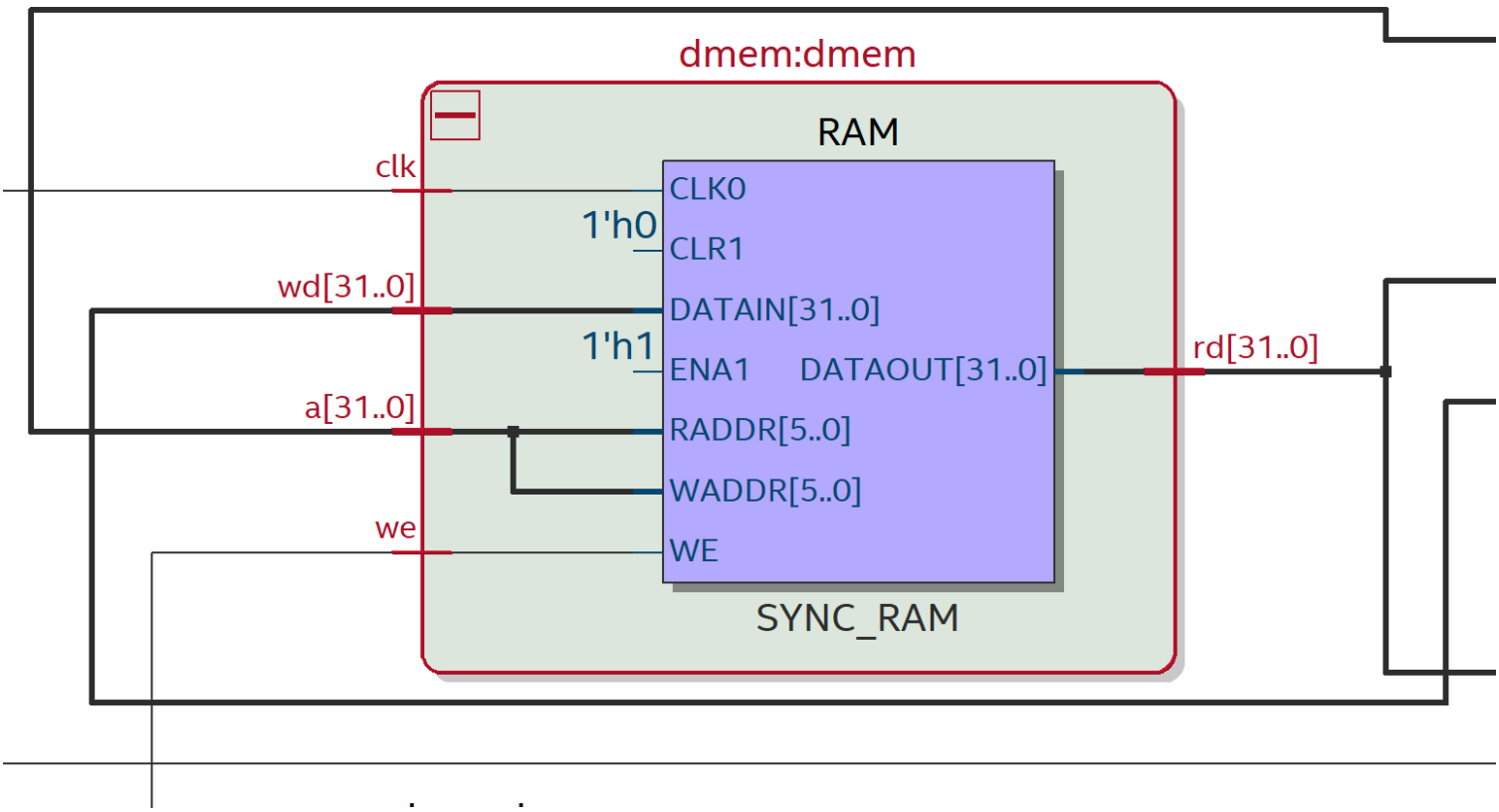

RTL Viewer - C:/intelFPGA_lite/17.1/HDLProject2/top - top
File Edit View Tools Window Help
Search altera.com
Page: 1 of 1
Netlist Navigator
top:1
mips
Instances
controller:c
Instances
aludec:ad
maindec:md
Primitives
Ports
datapath:dp
Instances
adder:pcadd1
adder:pcadd2
alu:alu
flopr:pcreg
mux2:pcbrmux
mux2:pcmux
mux2:resmux
mux2:srcbmux
mux2:wrmux
regfile:rf
signext:se
sl2:immsh
Ports
Ports
100%
00:00:03
Type here to search
08:04 PM
13-Mar-18


RTL Viewer - C:/intelFPGA_lite/17.1/HDLProject2/top - top
File Edit View Tools Window Help
Search altera.com
Page: 1 of 1
Netlist Navigator
top:1
mips
Instances
controller:c
Instances
aludec:ad
maindec:md
Primitives
Ports
datapath:dp
Instances
adder:pcadd1
adder:pcadd2
alu:alu
flopr:pcreg
mux2:pcbrmux
mux2:pcmux
mux2:resmux
mux2:srcbmux
mux2:wrmux
regfile:rf
signext:se
sl2:immsh
Ports
Ports
100%
00:00:03
Type here to search
08:06 PM
13-Mar-18

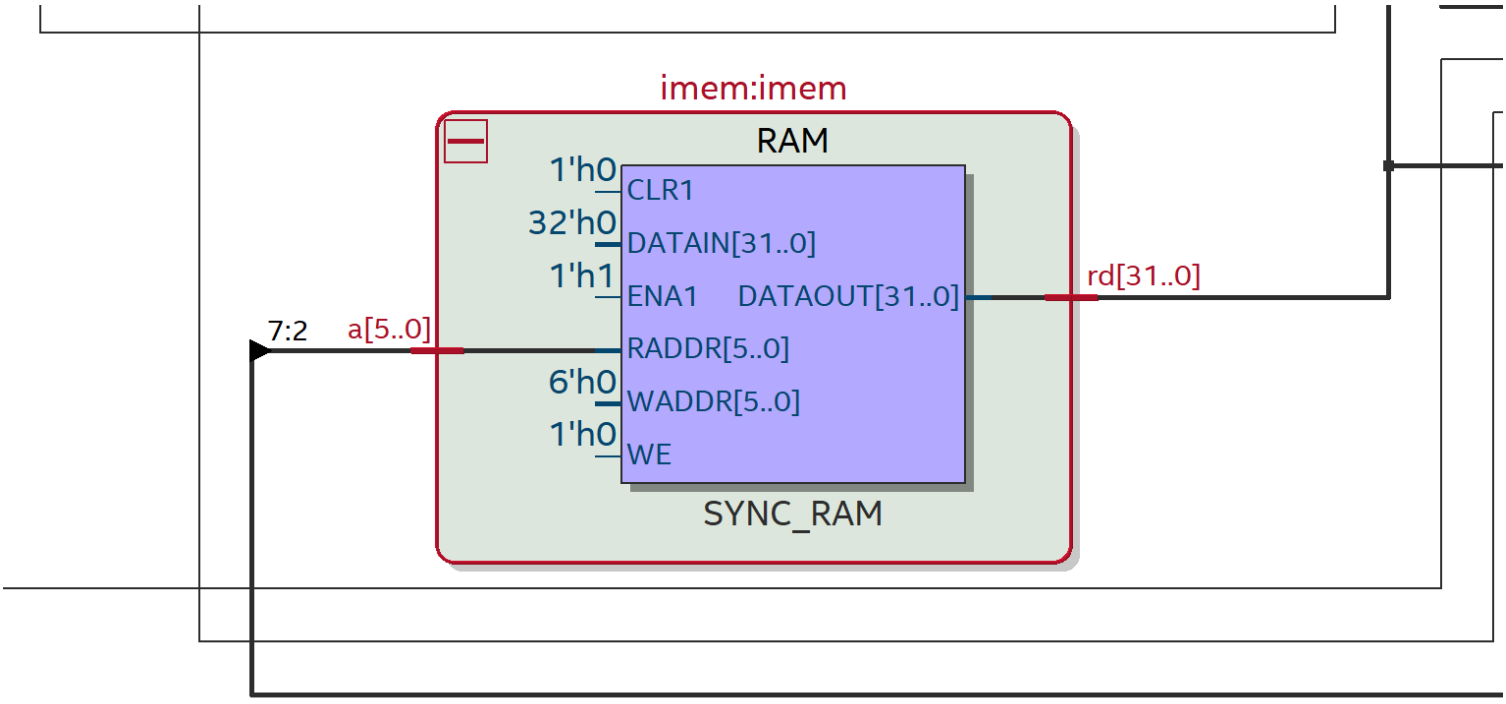
After modification



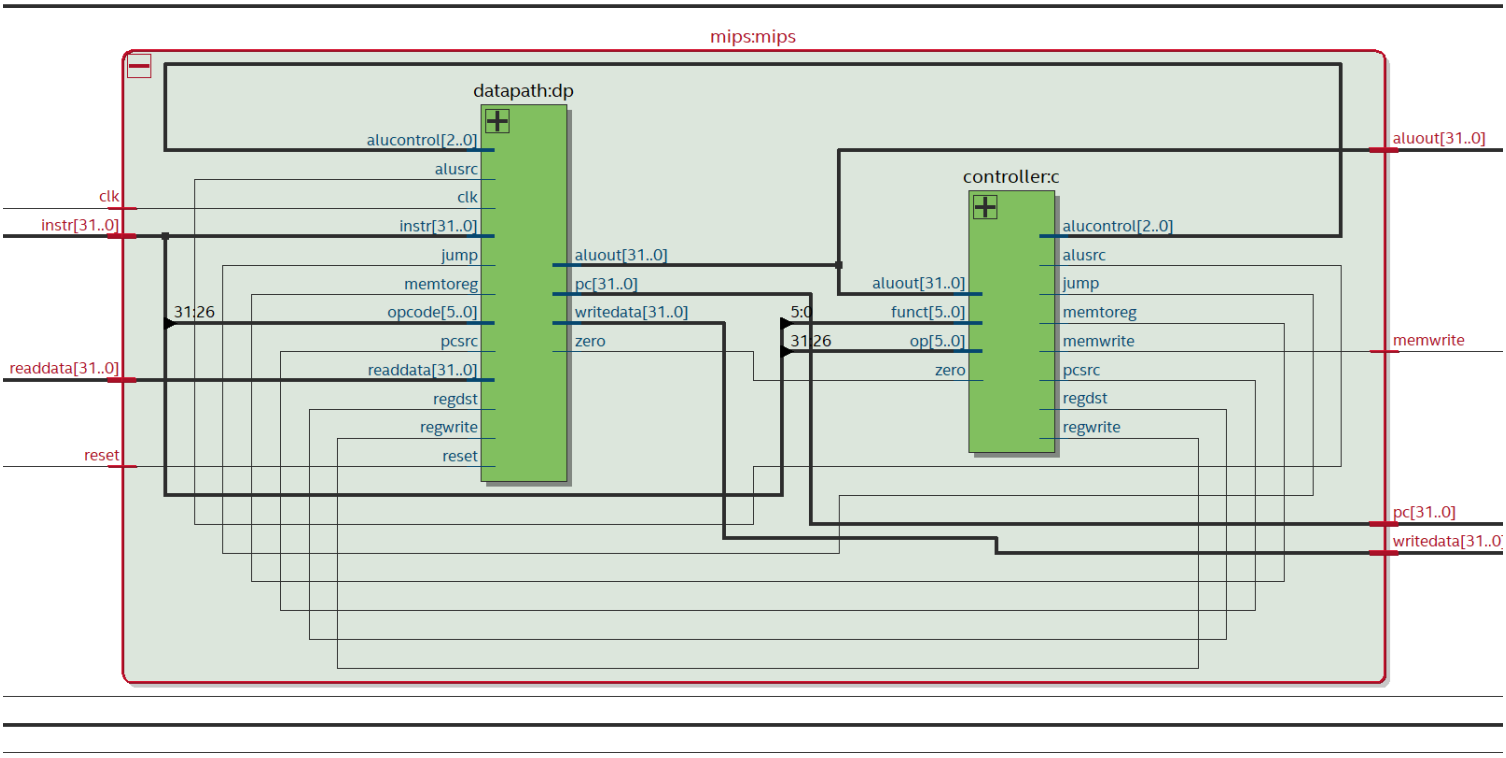
Inside dmem



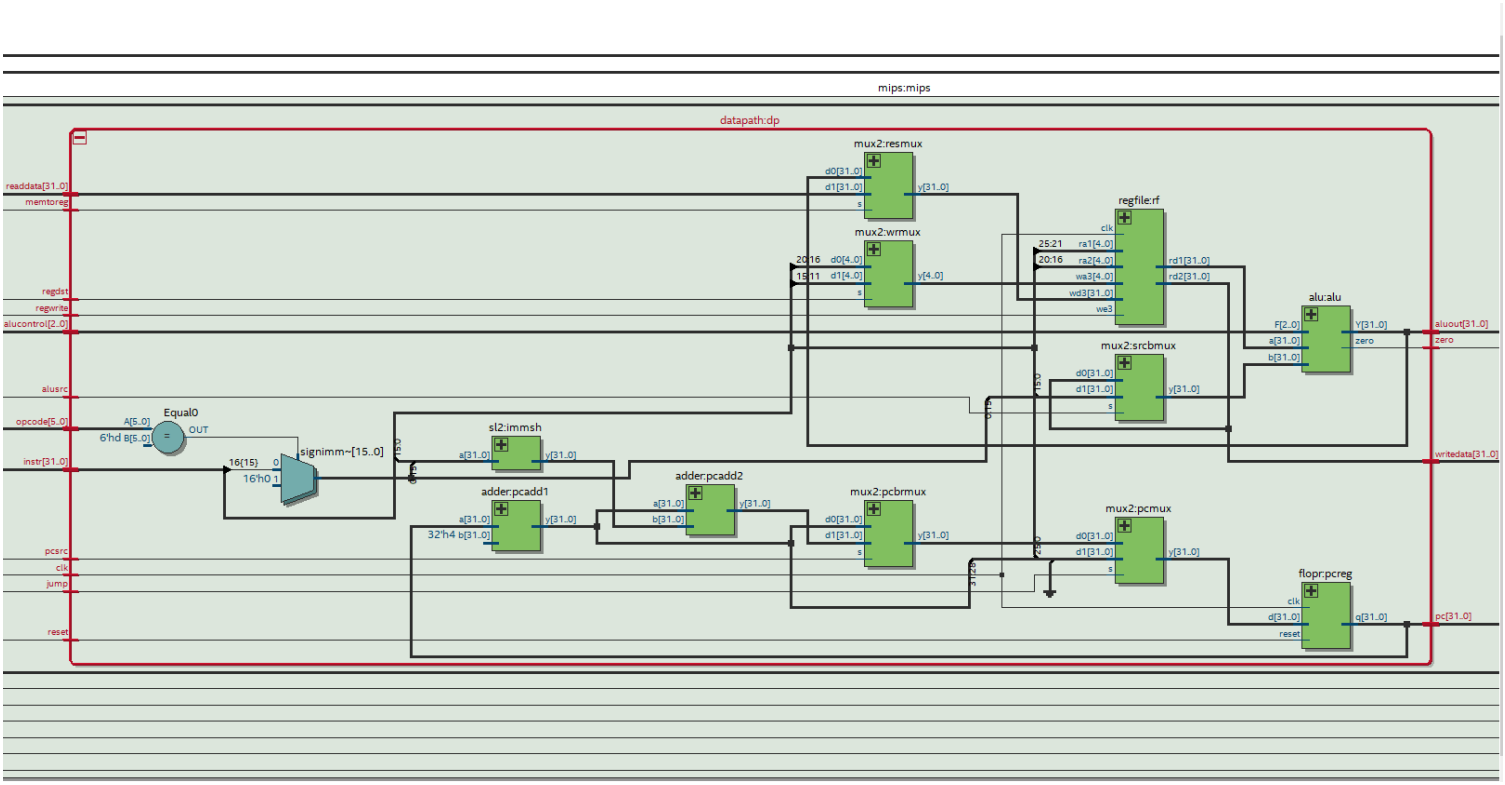
Inside imem



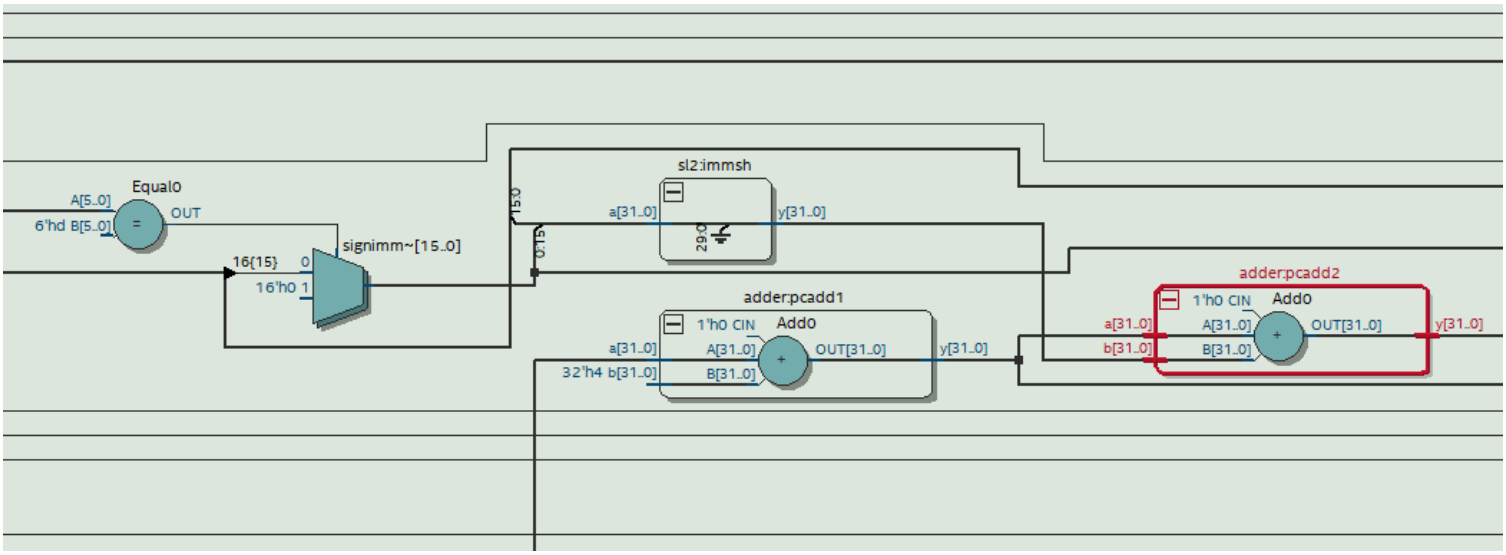
Inside mips

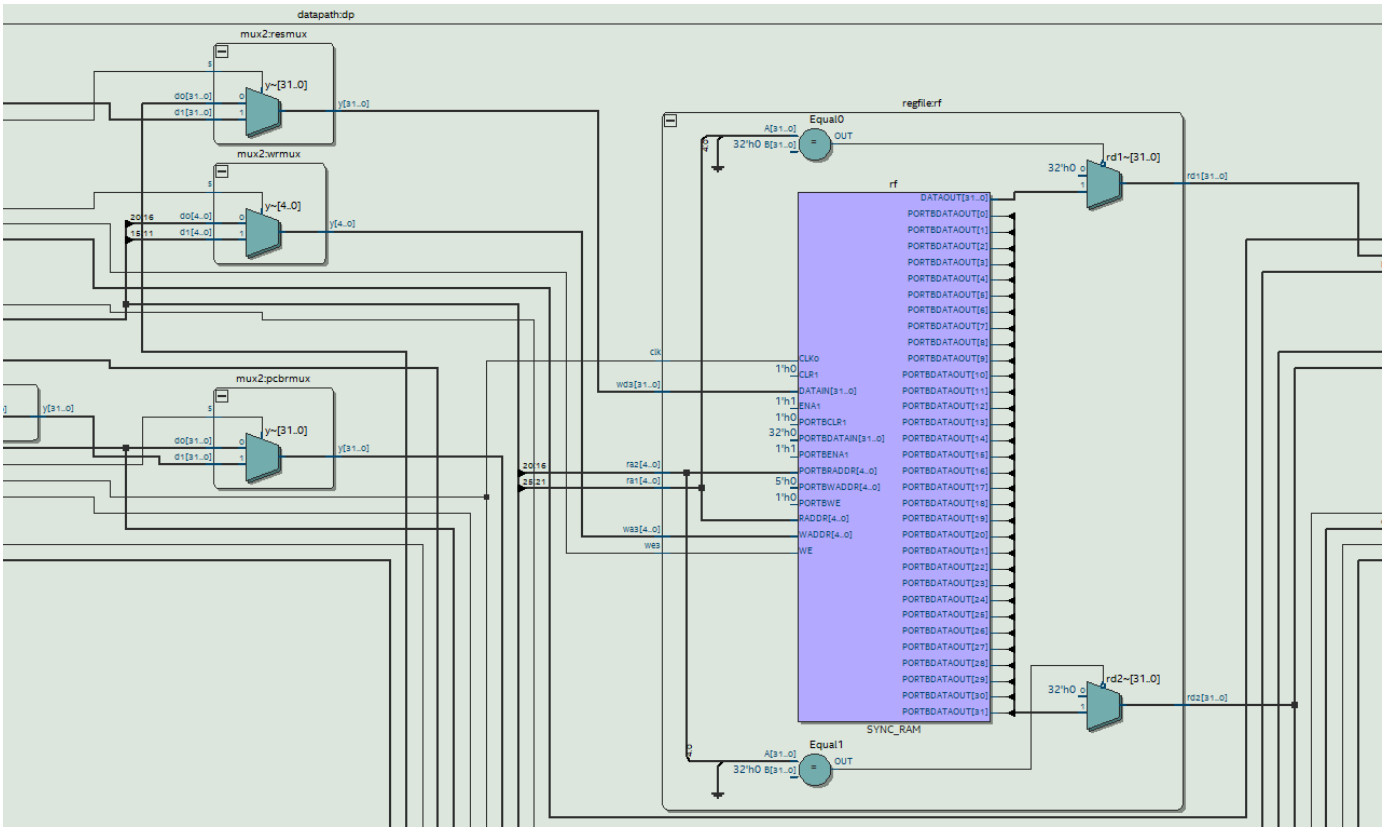


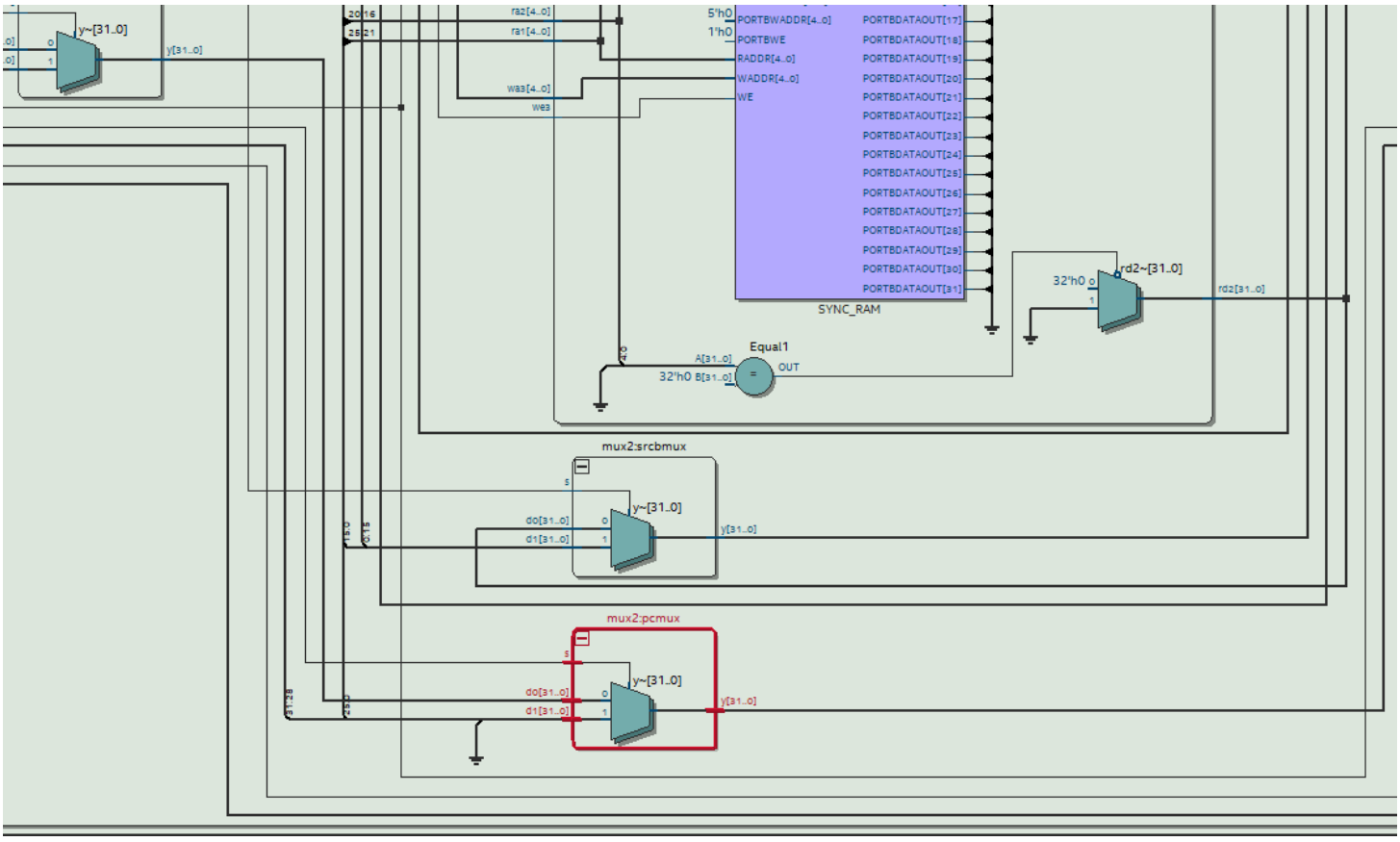
Inside datapath

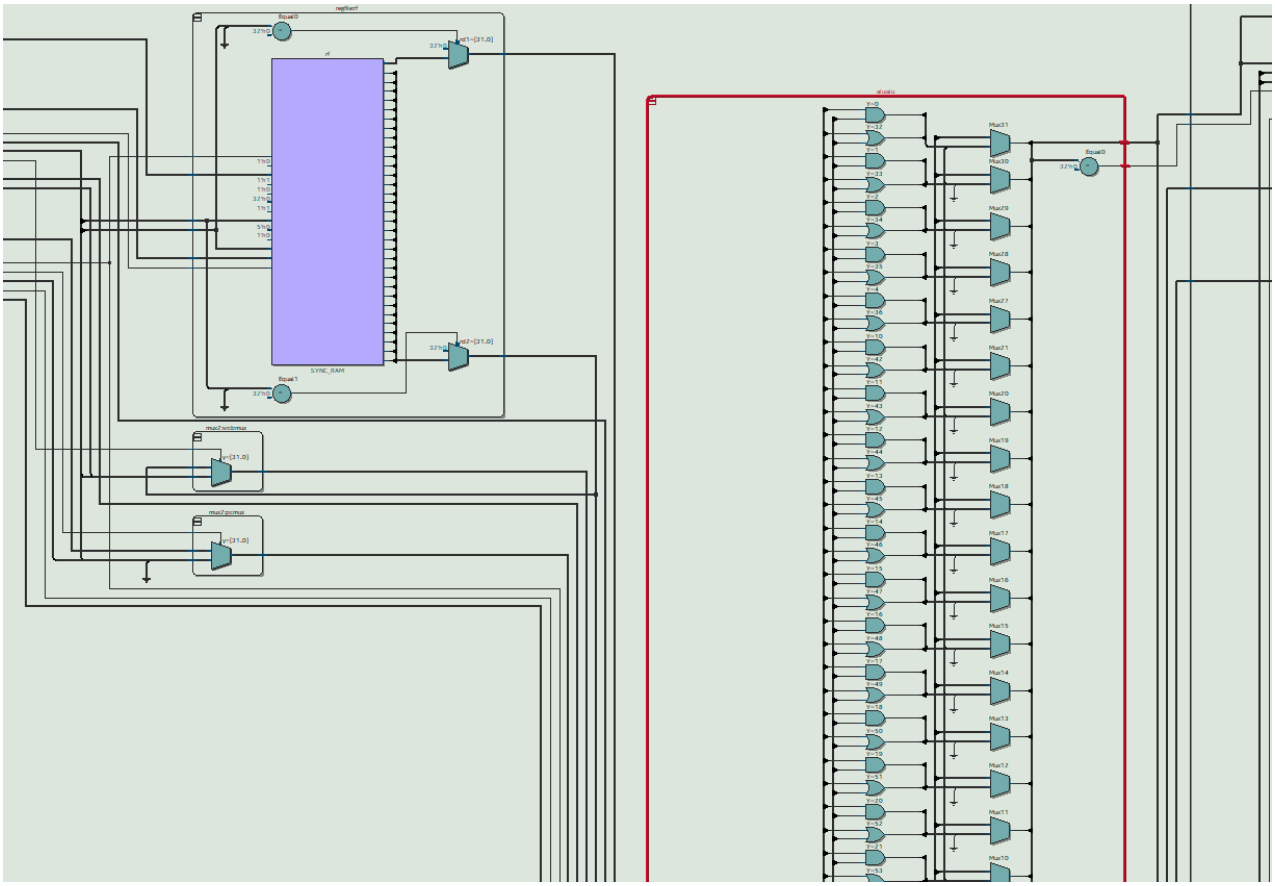


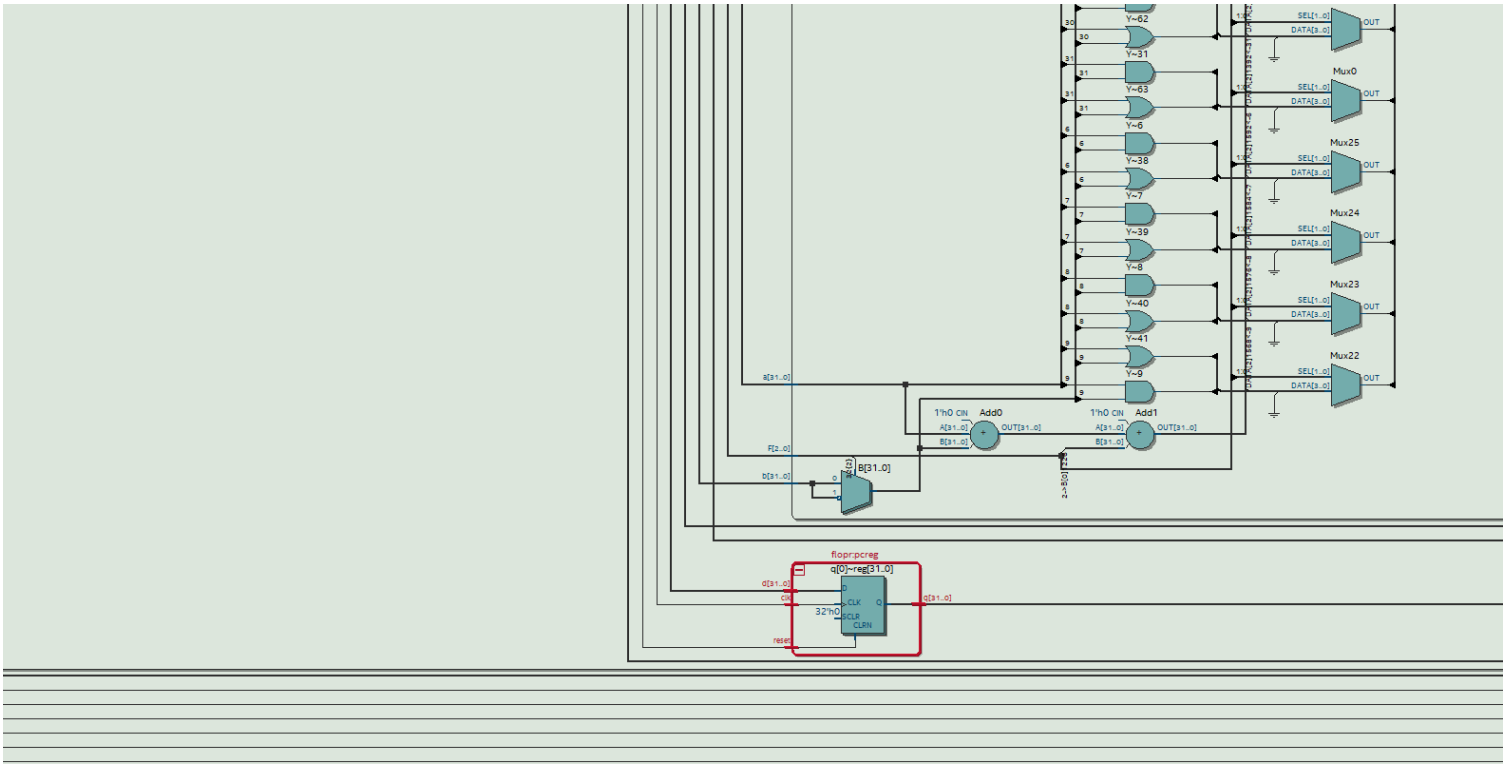
Zoomed view inside the datapath



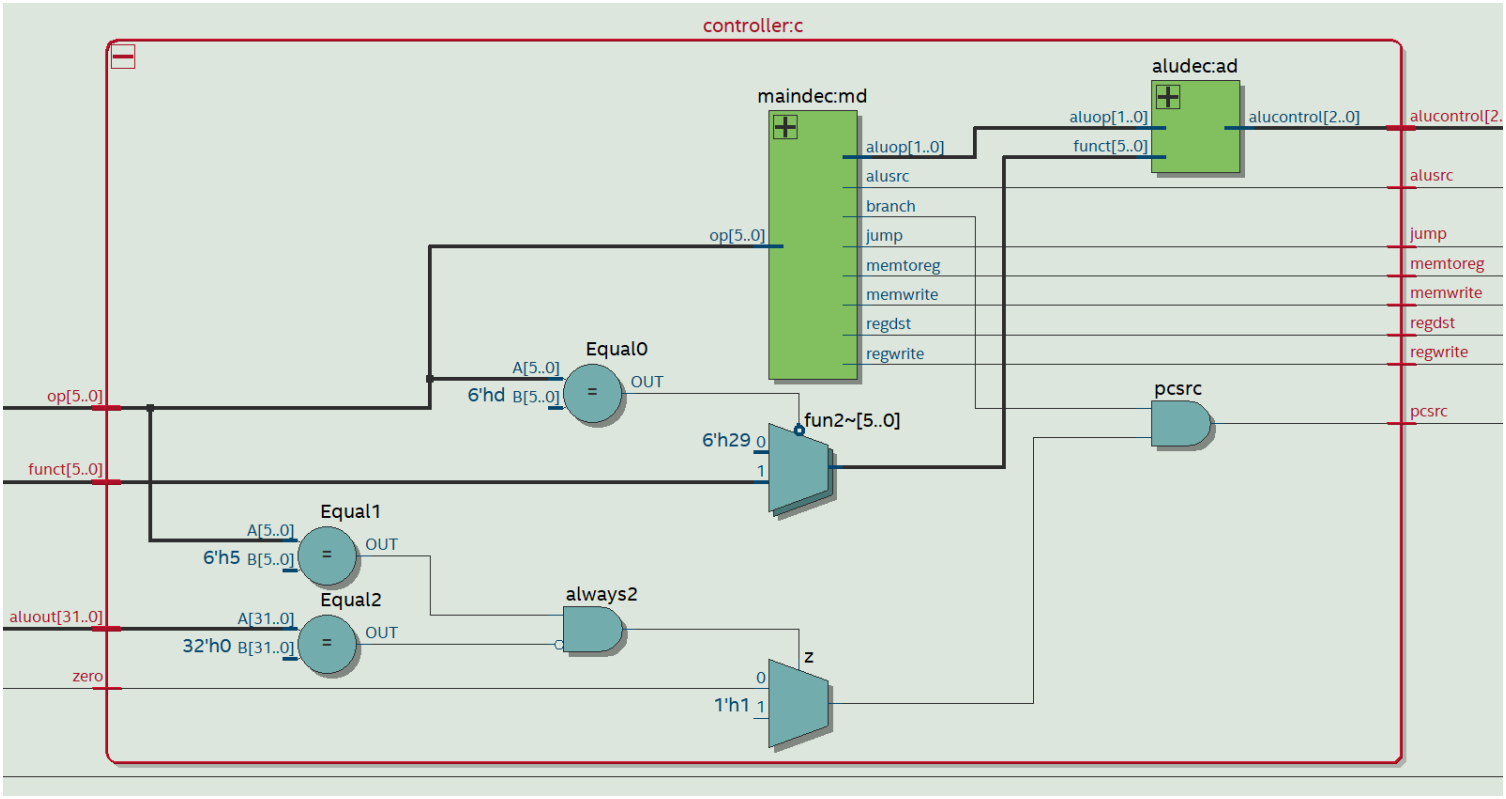

datapath:dp
mux2:resmux
y~[31..0]
do[31..0]
d1[31..0]
y[31..0]
mux2:wrmux
y~[4..0]
do[4..0]
d1[4..0]
y[4..0]
mux2:pcbrmux
regfile:rf
Equal0
A[31..0]
32'h0 B[31..0]
OUT
rd1~[31..0]
rf
DATAOUT[31..0]
PORTBDATAOUT[0]
PORTBDATAOUT[1]
PORTBDATAOUT[2]
PORTBDATAOUT[3]
PORTBDATAOUT[4]
PORTBDATAOUT[5]
PORTBDATAOUT[6]
PORTBDATAOUT[7]
PORTBDATAOUT[8]
PORTBDATAOUT[9]
PORTBDATAOUT[10]
PORTBDATAOUT[11]
PORTBDATAOUT[12]
PORTBDATAOUT[13]
PORTBDATAOUT[14]
PORTBDATAOUT[15]
PORTBDATAOUT[16]
PORTBDATAOUT[17]
PORTBDATAOUT[18]
PORTBDATAOUT[19]
PORTBDATAOUT[20]
PORTBDATAOUT[21]
PORTBDATAOUT[22]
PORTBDATAOUT[23]
PORTBDATAOUT[24]
PORTBDATAOUT[25]
PORTBDATAOUT[26]
PORTBDATAOUT[27]
PORTBDATAOUT[28]
PORTBDATAOUT[29]
PORTBDATAOUT[30]
PORTBDATAOUT[31]
CLKo
CLR1
DATAIN[31..0]
ENA1
PORTBCLR1
PORTBDATAIN[31..0]
PORTBENA1
PORTBRADDR[4..0]
PORTBWADDR[4..0]
PORTBWE
RADDR[4..0]
WADDR[4..0]
WE
SYNC_RAM
rd2~[31..0]
rd2[31..0]
Equal1
clk
wd3[31..0]
ra2[4..0]
ra1[4..0]
wa3[4..0]
we3


y~[31..0]
y[31..0]
ra2[4..0]
ra1[4..0]
wa3[4..0]
we3
5'h0
1'h0
PORTBWADDR[4..0]
PORTBWE
RADDR[4..0]
WADDR[4..0]
WE
PORTBDATAOUT[17]
PORTBDATAOUT[18]
PORTBDATAOUT[19]
PORTBDATAOUT[20]
PORTBDATAOUT[21]
PORTBDATAOUT[22]
PORTBDATAOUT[23]
PORTBDATAOUT[24]
PORTBDATAOUT[25]
PORTBDATAOUT[26]
PORTBDATAOUT[27]
PORTBDATAOUT[28]
PORTBDATAOUT[29]
PORTBDATAOUT[30]
PORTBDATAOUT[31]
SYNC_RAM
rd2~[31..0]
32'h0
rd2[31..0]
Equal1
A[31..0]
32'h0 B[31..0]
OUT
mux2:srcbmux
do[31..0]
d1[31..0]
mux2:pcmux

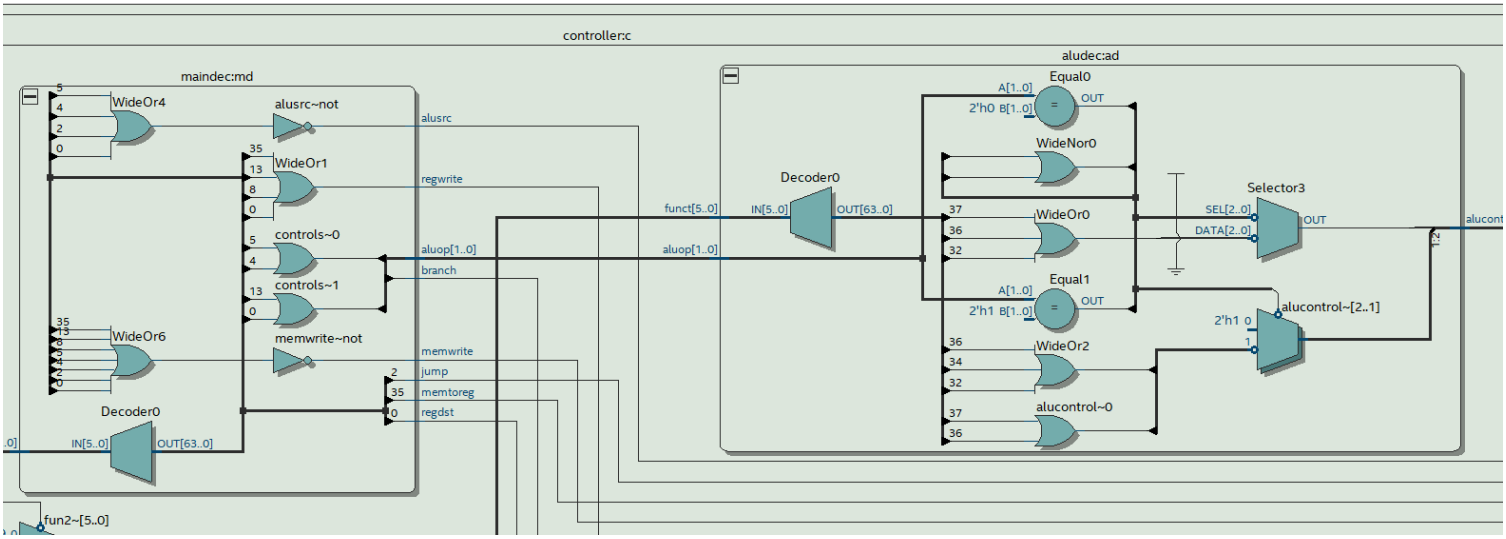



Mux0
Mux25
Mux24
Mux23
Mux22
Add0
Add1
B[31..0]
floprpcreg
q[0]~reg[31..0]
SYNC_RAM

Inside controller



Inside maindec and aludec



Changes identified are:

There is a change in the maindec module between the modified and unmodified code. The modified code has extra number of OR gates as control 0 and control 1.

Another change is in the aludec module. The modified aludec has changes in the positioning of the gates which changes the paths to be taken.
The basic working and the diagrammatic representation remains the same for both the MIPS processors, only that specific changes take place in sub-modules as states above.