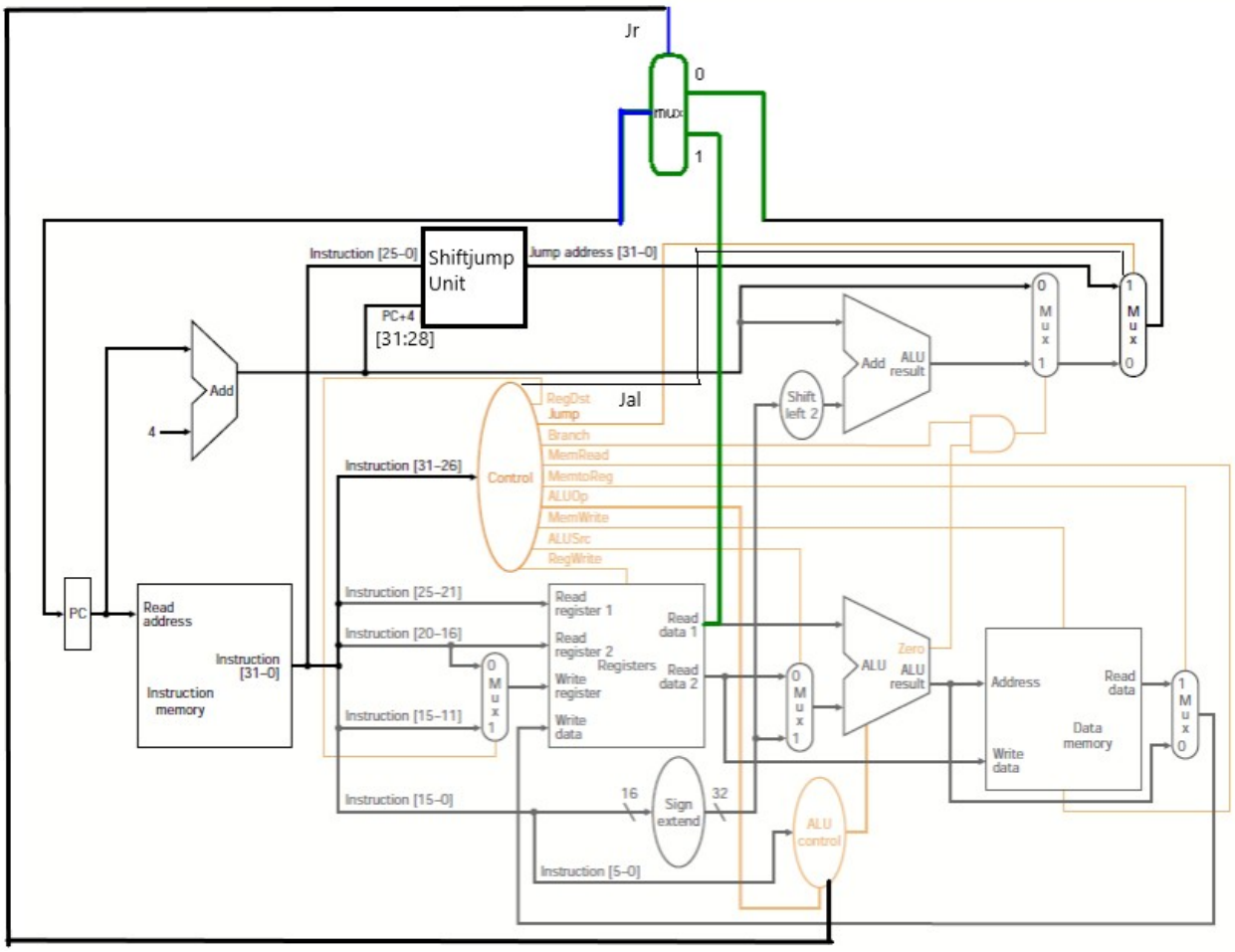
CENG311 PROGRAMMING ASSIGNMENT 3
REPORT

Processor Datapath



Implementation Details

Firstly I implemented new unit which is Shiftjump unit. This unit takes instructions last 26 bits and PC+4's most significant 4 bits. Then unit shifts 2 bits to the left the instruction bits and concatenate with PC+4[31:28]. Then it gives the output to the multiplexer 5.

```
module shiftjump(out,in,pcin);
output [31:0] out;
input [25:0] in;
input [3:0] pcin;
wire [27:0] shifted;
assign shifted=in<<2;
assign out = {pcin,shifted};
endmodule
```

I revised control unit. I added new control signals which are jump, jumpandlink and one additional aluop. I added new wires for checking new instructions' opcodes. I added new input to the control unit because we must check rt field of instruction to decide if it is bgez or bltz. These instructions' opcodes are equal but bgez's rt field is 1, bltz's rt field is 0.

```
module control(in, inrt, regdest, alusrc, memtoreg, regwrite, memread, memwrite, jump, jumpandlink, branch, aluop2, aluop1, aluop0);
 input [5:0] in;   // instruction op code
 input [4:0] inrt; // instruction rt field for bgez and bltz
 output regdest, alusrc, memtoreg, regwrite, memread, memwrite, jump, jumpandlink, branch, aluop2, aluop1, aluop0;

 wire rformat,lw,sw,beq,bne,bgez,bgtz,bltz,addi,andi,j,jal;

 assign rformat =~| in;

 assign lw = in[5]& (~in[4])&(~in[3])&(~in[2])&in[1]&in[0];

 assign sw = in[5]& (~in[4])&in[3]&(~in[2])&in[1]&in[0];

 // bgez opcode = 000001 rt = 00001
 assign bgez = ~in[5]& (~in[4])&(~in[3])&(~in[2])&(~in[1])&in[0]&(~inrt[4])& (~inrt[3])&(~inrt[2])&(~inrt[1])&inrt[0];
 // bltz opcode = 000001 rt = 00000
 assign bltz = ~in[5]& (~in[4])&(~in[3])&(~in[2])&(~in[1])&in[0]&(~inrt[4])& (~inrt[3])&(~inrt[2])&(~inrt[1])&(~inrt[0]);
 assign beq = ~in[5]& (~in[4])&(~in[3])&in[2]&(~in[1])&(~in[0]);     // 000100
 assign bne = ~in[5]& (~in[4])&(~in[3])&in[2]&(~in[1])& in[0];       // 000101
 assign bgtz = ~in[5]&(~in[4])&(~in[3])&in[2]&in[1]&in[0];           // 000111
 assign addi = ~in[5]&(~in[4])&in[3]&(~in[2])&(~in[1])&(~in[0]);     // 001000
 assign andi = ~in[5]&(~in[4])&in[3]&in[2]&(~in[1])&(~in[0]);        // 001100
 assign j = ~in[5]&(~in[4])&(~in[3])&(~in[2])&in[1]&(~in[0]);     // 000010
 assign jal = ~in[5]&(~in[4])&(~in[3])&(~in[2])&in[1]&in[0];     // 000011

 assign regdest = rformat;
 assign alusrc = lw|sw|addi|andi;
 assign memtoreg = lw;
 assign regwrite = rformat|lw|addi|andi;
 assign memread = lw;
 assign memwrite = sw;
 assign jump = j|jal;
 assign jumpandlink = jal;
 assign branch = beq|bne|bltz|bgtz|bgez; // branch operations
 assign aluop0 = rformat|beq|bgez|bltz; // xx1
 assign aluop1 = andi|beq|bgtz|bltz;    // x1x
 assign aluop2 = bne|bgez|bgtz|bltz;    // 1xx

endmodule
```

Thirdly I revised ALUControl unit. I added additional output signal which is jr. If instruction is jr, then I set it as 1. Then I changed gout to 4 bits because I gave new gout signal for every new instruction except addi.

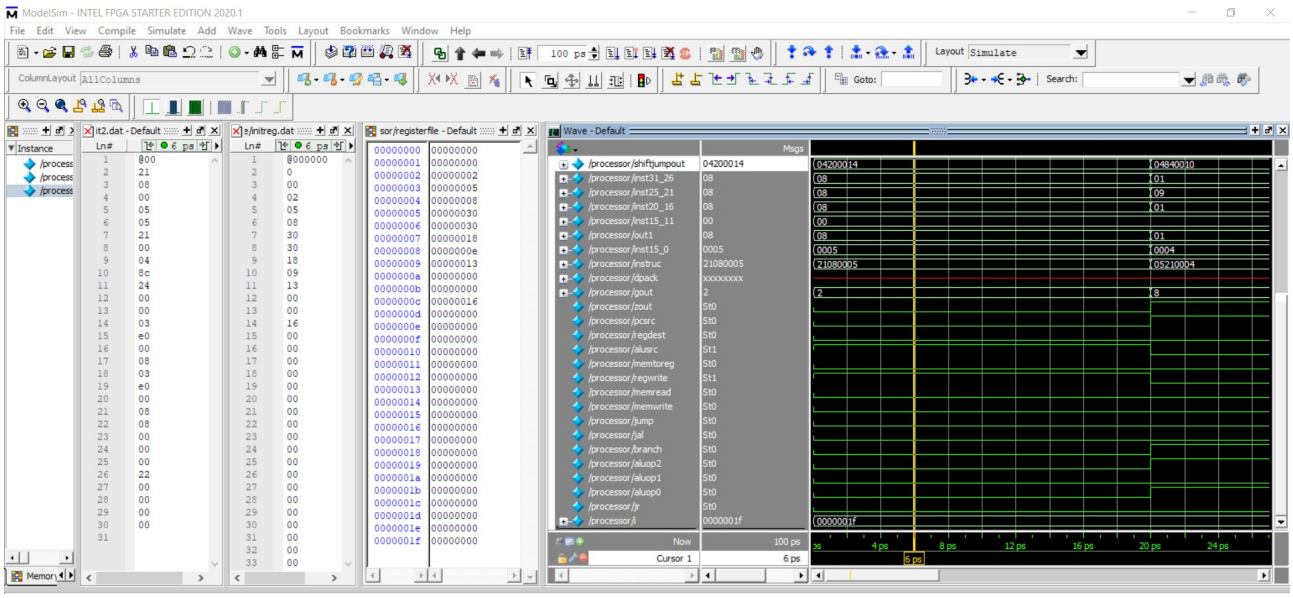
```
input aluop2,aluop1,aluop0,f3,f2,f1,f0;
output [3:0] gout;
reg [3:0] gout;
output reg jr; // jr signal

always @(aluop2 or aluop1 or aluop0 or f3 or f2 or f1 or f0)
begin
        jr=0; //initialize
        if((~aluop2)&(~aluop1)&(~aluop0)) // lw/sw/addi
                gout=4'b0010;
        if((~aluop2)&aluop1&(~aluop0)) // andi
                gout=4'b0000;
        if((~aluop2)&aluop1&aluop0)  // beq
                gout=4'b0110;
        if(aluop2&(~aluop1)&(~aluop0))  //bne
                gout=4'b1011;
        if(aluop2&(~aluop1)&aluop0)  // bgez
                gout=4'b1000;
        if(aluop2&aluop1&(~aluop0)) // bgtz
                gout=4'b1001;
        if(aluop2&aluop1&aluop0) // bltz
                gout=4'b1010;
        if((~aluop2)&(~aluop1)&aluop0) // r-type
        begin
                if (f3&(~f2)&(~f1)&(~f0))
                begin
                        jr = 1;
                        gout=4'b0000;  // just only initialize
                end
                if (~(f3|f2|f1|f0))
                        gout=4'b0010;        // Funct field = 0000, add
                if (f1 & f3)
                        gout=4'b0111;        // Funct field = 1010, slt
                if (f1 &~(f3))
                        gout=4'b0110;        // Funct field = 0010, sub
                if (f2 & f0)
                        gout=4'b0001;        // Funct field = 0101, or
                if (f2 &~(f0))
                        gout=4'b0000;        // Funct field = 0100, and
                if (f2 & f1 & f3 & f0)
                        gout=4'b0011;        // Funct field = 1111, mul
                if (~(f3) & f2 & f1 & f0)
                        gout=4'b1100;        // Funct field = 0111, nor
        end
end
```

Lastly I revised ALU32 to perform new operations according to each incoming gout. In bgez, bgtz, bltz instructions, I need to check sign bit because these instructions have restraints. If instruction is a branch type, then I set zout as 1.

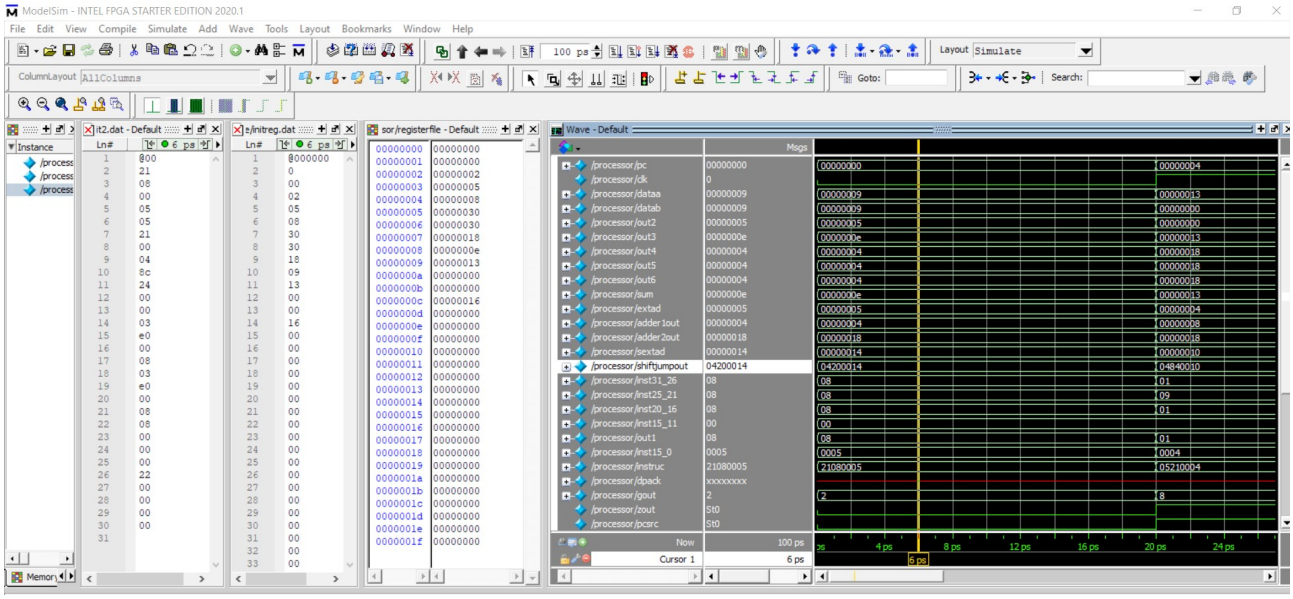
```
module alu32(alu_out, a, b, zout, alu_control);
output reg [31:0] alu_out;
input [31:0] a,b;
input [3:0] alu_control;

reg [31:0] less;
output zout;
reg zout;

always @(a or b or alu_control)
begin
zout = 0;
        case(alu_control)
        4'b0000: alu_out = a & b; // and
        4'b0001: alu_out = a|b; // or
        4'b0010: alu_out = a+b; // add
        4'b0011: alu_out = a * b; //mul
        4'b0110: begin alu_out = a+1+(~b); // sub, beq
                        zout=~(|alu_out);
                end
        4'b0111: begin less = a+1+(~b); //slt,
                        if (less[31]) alu_out=1;
                        else alu_out=0;
                end
        4'b1000: begin zout = (a==0 ? 1:((~a[31]) ? 1:0)); // bgez
                        alu_out = a;  // I checked 32nd bit because sign bit must be zero
                end
        4'b1001: begin zout = ((~a[31]) ? (a!=0 ? 1:0):0); // bgtz
                        alu_out = a;  // now sign bit is zero but value is not zero
                end
        4'b1010: begin zout = (a==0 ? 1:(a[31] ? 1:0)); // bltz
                        alu_out = a;  // now sign bit is one and value is zero
                end
        4'b1011: begin alu_out = a+1+(~b); // bne
                        zout=(|alu_out);
                end
        4'b1100: alu_out=~(a|b); // nor
        default: alu_out=31'bx;
        endcase
end
endmodule
```

Instructions

1- addi $t0, $t0, 5
Bit Representation: 001000 01000 01000 0000000000000101
Hex: 0x21080005

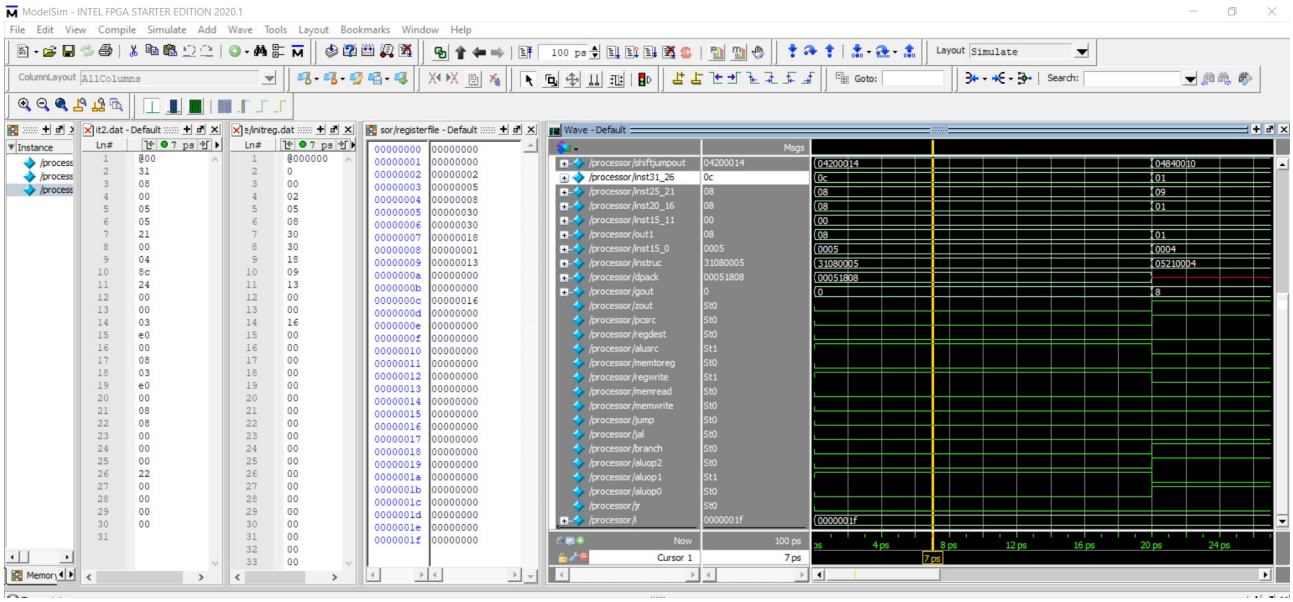


Control bits: regwrite = 1, alusrc = 1, aluop = 000, gout = 0x2 (0010)

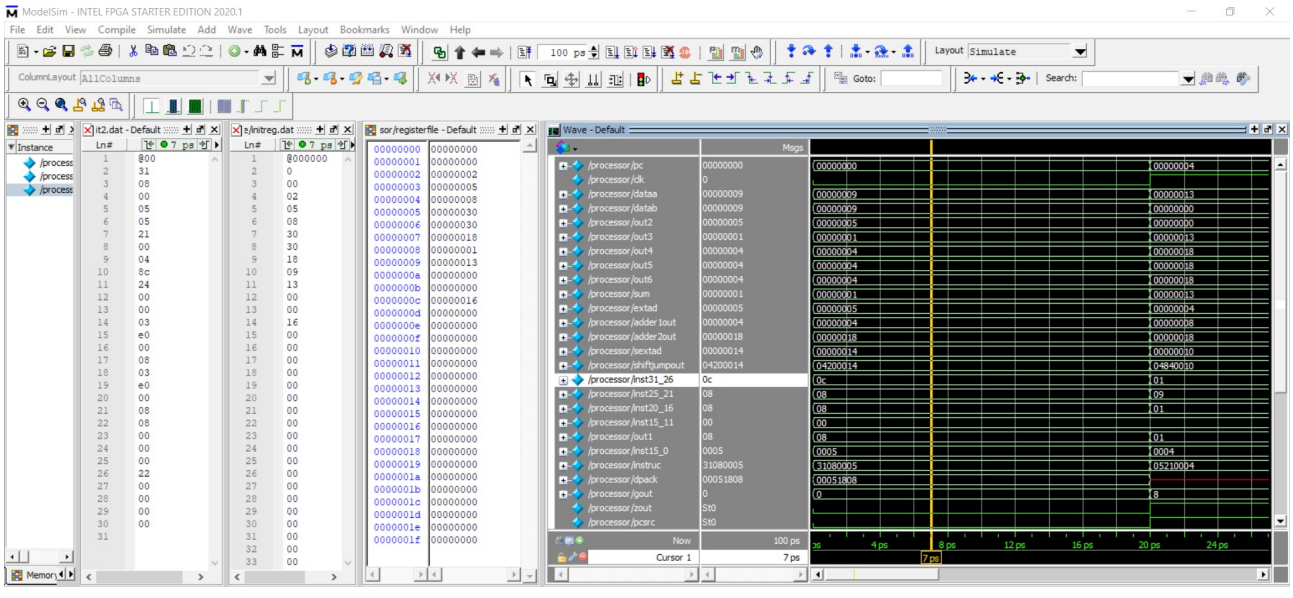


out1 is set as 08 which is corresponding to $r8($t0). dataa = 09 (value in $t0 as you can see in initreg.dat file). out2 is set as instruc[15:0] because alusrc is 1. Then sum is set 14(0xe) which is corresponding to dataa+out2. out3 is equal to sum because memtoreg control bit 0. Then $r8($t0) is set as 14(0xe) as you can see in registerFile.

2- andi $t0, $t0, 5
Bit Representation: 001100 01000 01000 0000000000000101
Hex: 0x31080005



Control bits: regwrite = 1, alusrc = 1, aluop = 010, gout = 0x0 (0000)

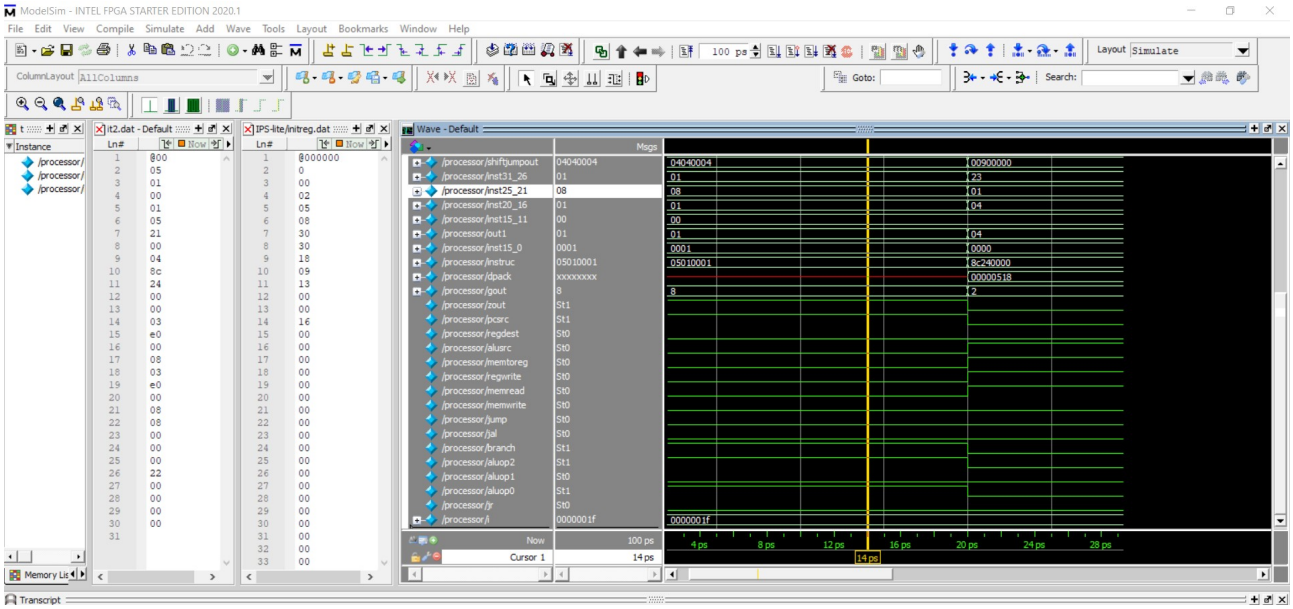


out1 is set as 08 which is corresponding to $r8($t0). dataa = 09 (value in $t0 as you can see in initreg.dat file). out2 is set as instruc[15:0] because alusrc is 1. Then sum is set 14(0xe) which is corresponding to dataa&out2. out3 is equal to sum because memtoreg control bit 0. Then $r8($t0) is set as 1(0x1) as you can see in registerFile.

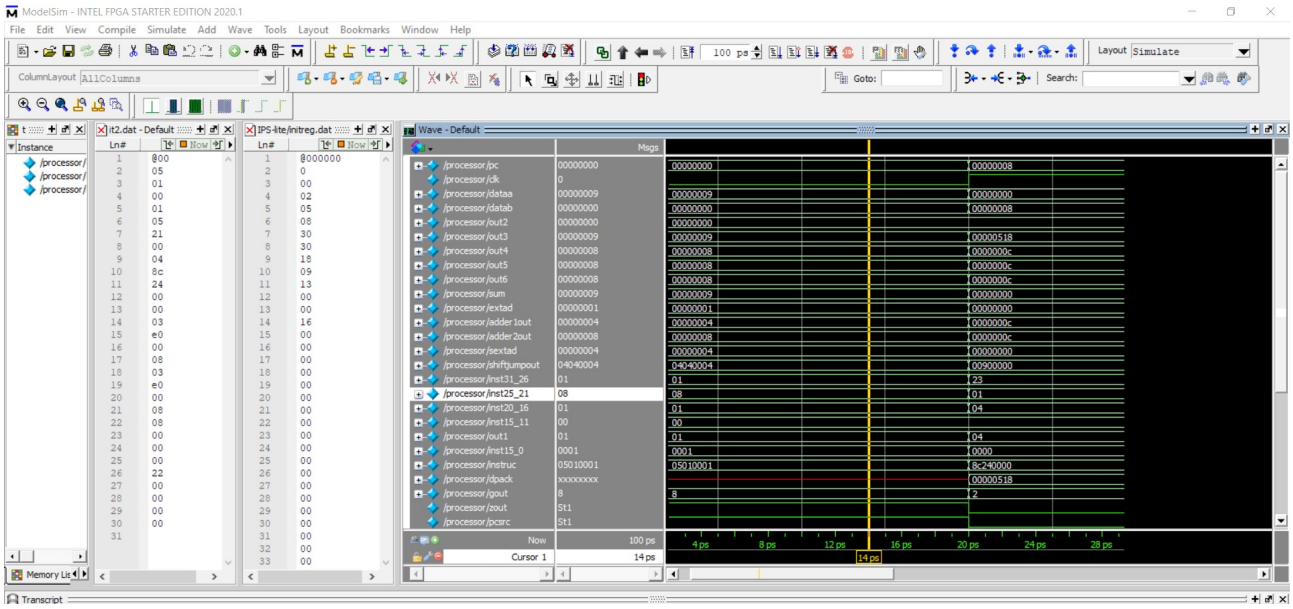
3- bgez $t0, 0x0001

Bit Representation: 000001 01000 00001 0000000000000001

Hex: 0x05010001



Control Bits: branch = 1, pcsrc = 1, zout = 1, aluop = 101, gout = 0x8(1000)

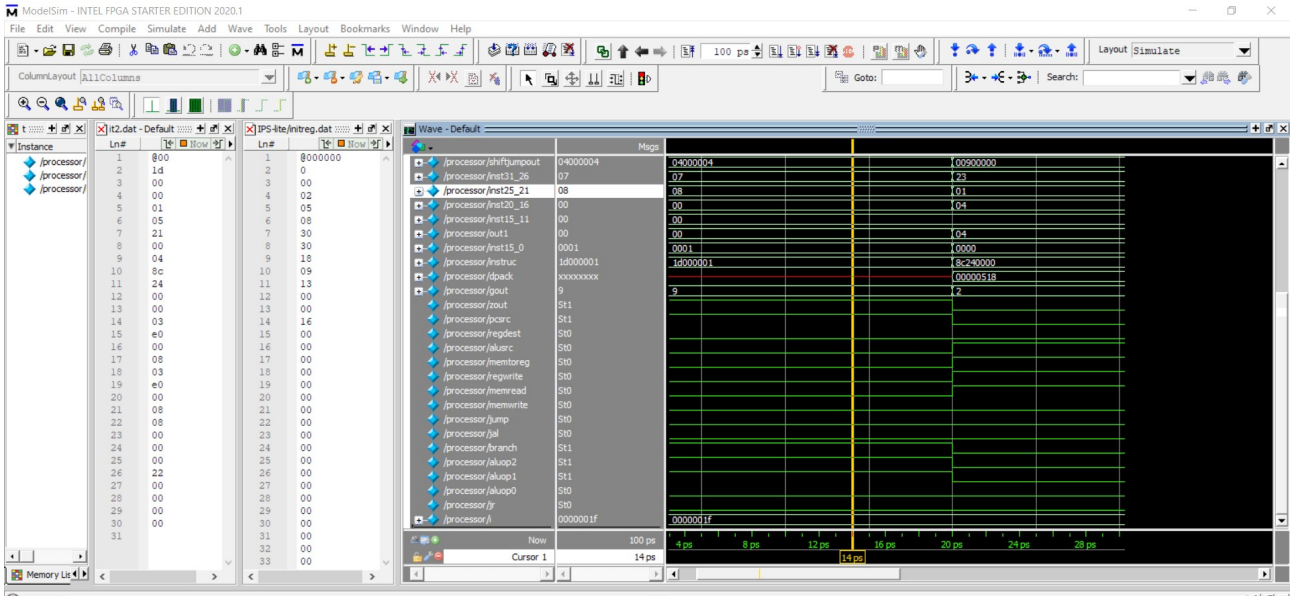


dataa is set as 0x9 ($r8 = $t0 = 09), datab is set as 0 because it is $r1's value. Extad is set as 1(offset), then its shifted value, sextad, is 0x4. Gout says it is bgez operation to alu32. Alu32 sets zout as 1 and sets sum as dataa. And pcsrc is 1 so program jumps to adder2out.

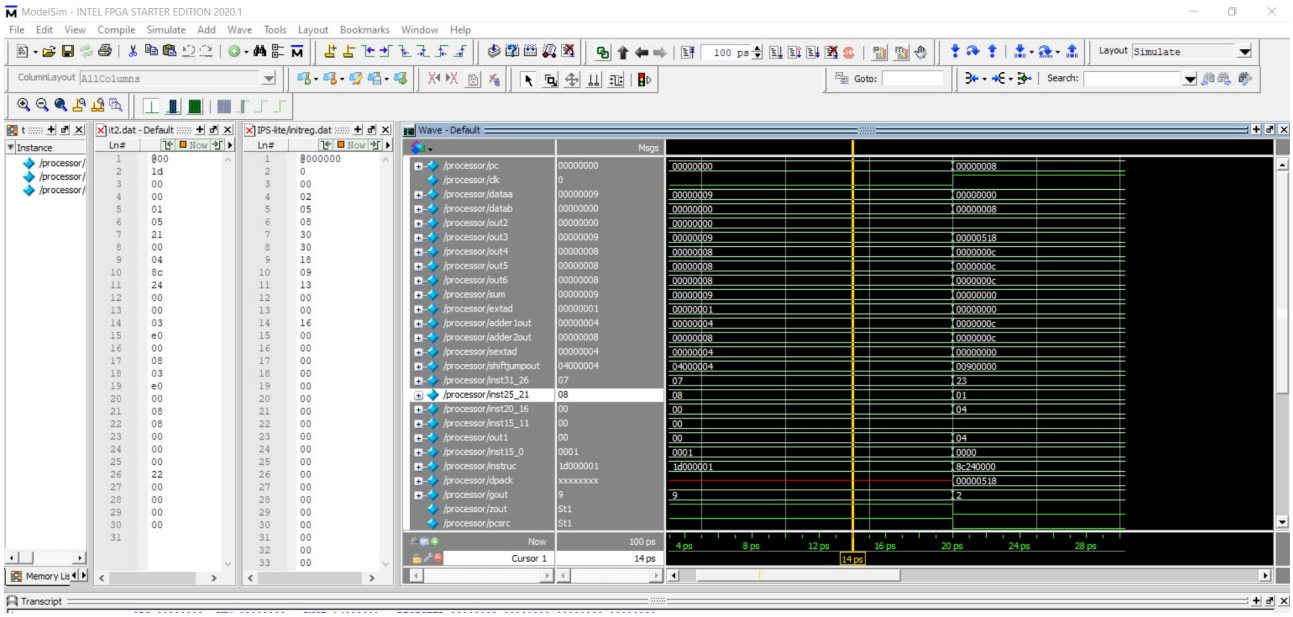
4- bgtz $t0, 0x0001

Bit Representation: 000111 01000 00000 0000000000000001

Hex: 0x1D000001

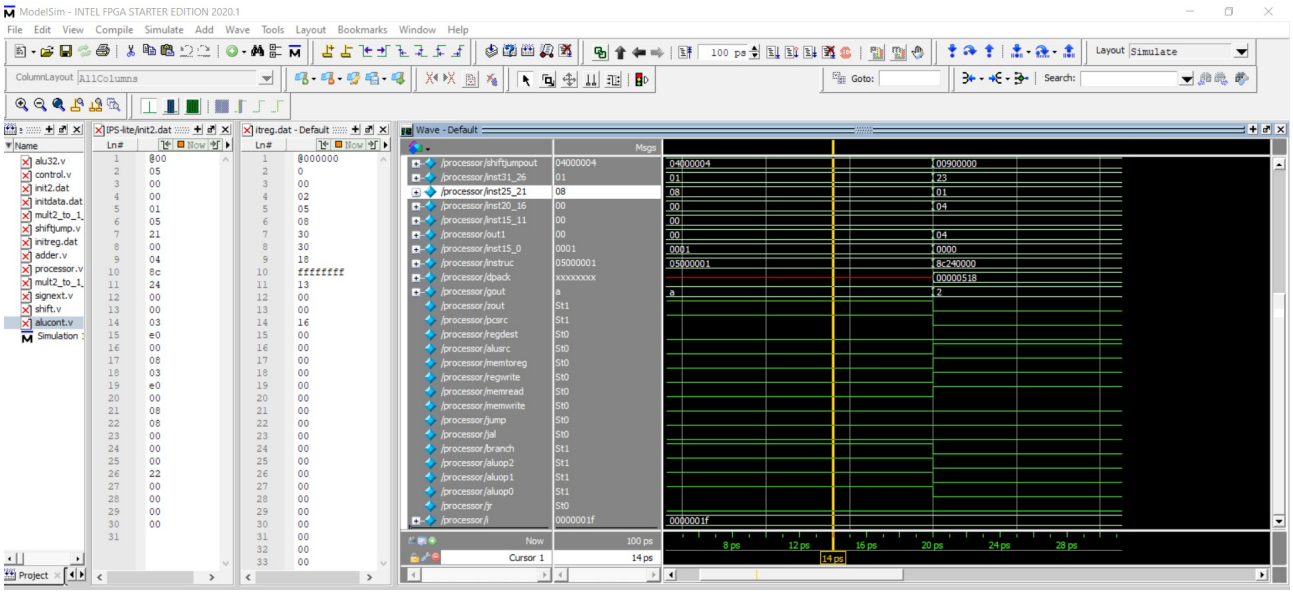


Control Bits: branch = 1, pcsrc = 1, zout = 1, aluop = 110, gout = 0x9(1001)

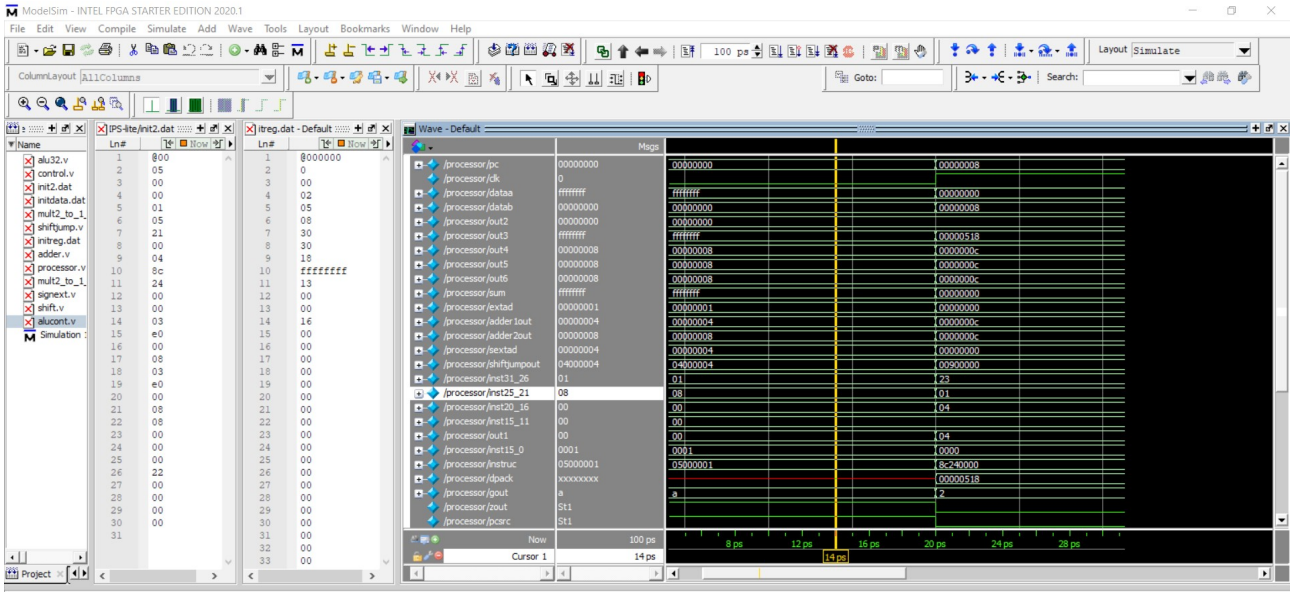


dataa is set as 0x9 ($r8 = $t0 = 09), datab is set as 0 because it is $r0's value. Extad is set as 1(offset), then its shifted value, sextad, is 0x4. Gout says it is bgtz operation to alu32. Alu32 sets zout as 1 and sets sum as dataa. And pcsrc is 1 so program jumps to adder2out.

5- bltz $t0, 0x00001
Bit Representation: 000001 01000 00000 0000000000000001
Hex: 0x05000001

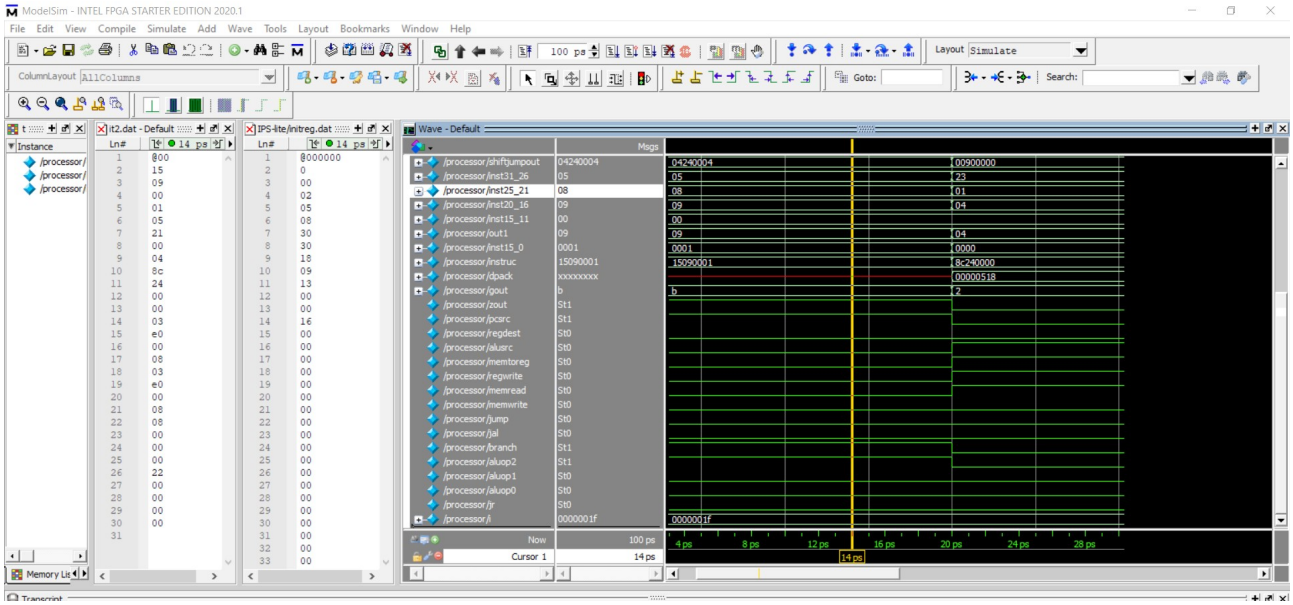


Control Bits: branch = 1, pcsrc = 1, zout = 1, aluop = 111, gout = 0xa(1010)

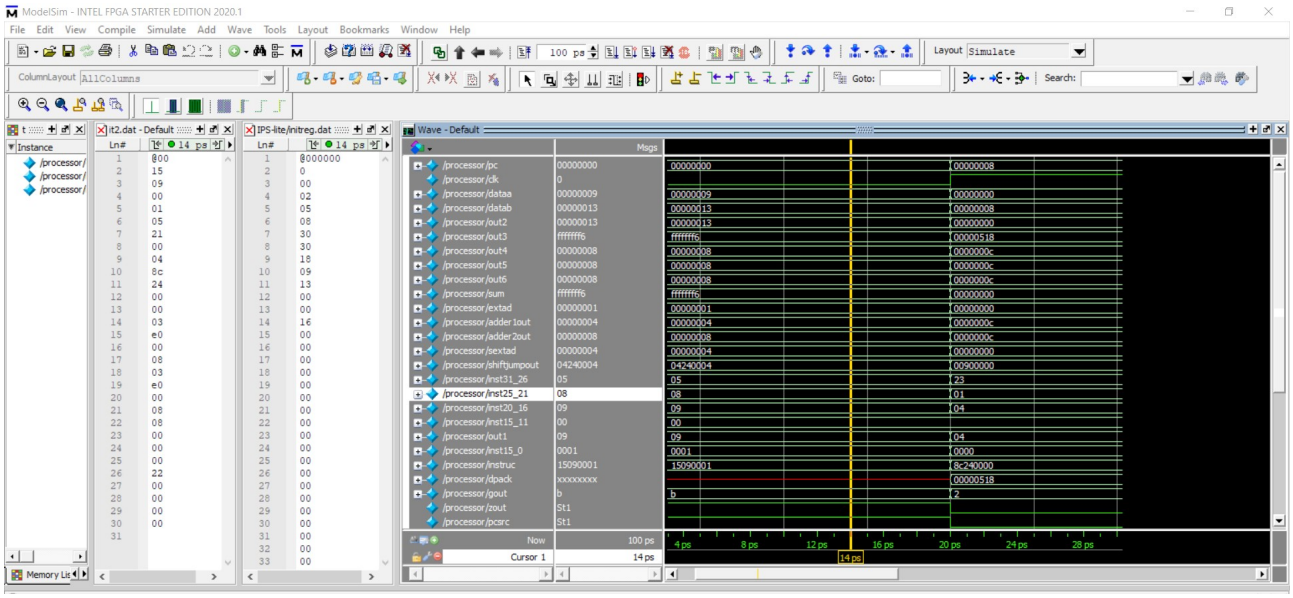


dataa is set as 0xffffffff ($r8 = $t0 = -1), datab is set as 0 because it is $r0's value. Extad is set as 1(offset), then its shifted value, sextad, is 0x4. Gout says it is bltz operation to alu32. Alu32 sets zout as 1 and sets sum as dataa. And pcsrc is 1 so program jumps to adder2out.

6- bne $t0, $t1, 0x0001
Bit Representation: 000101 01000 01001 0000000000000001
Hex: 0x15090001

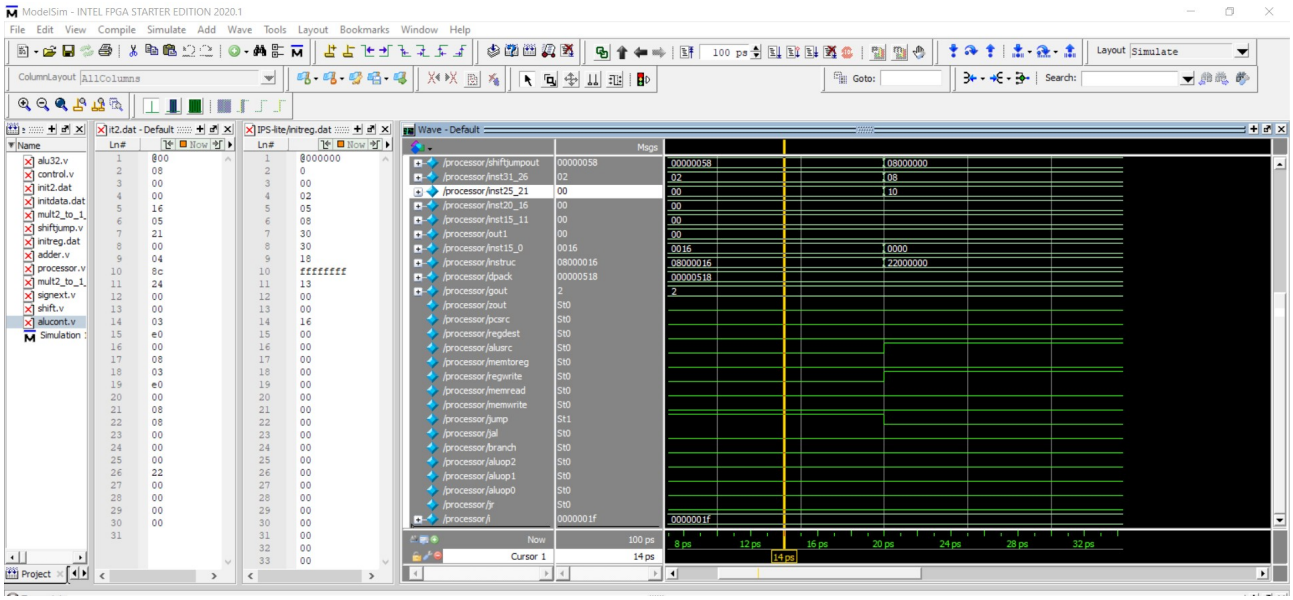


Control Bits: branch = 1, pcsrc = 1, zout = 1, aluop = 100, gout = 0xb(1011)

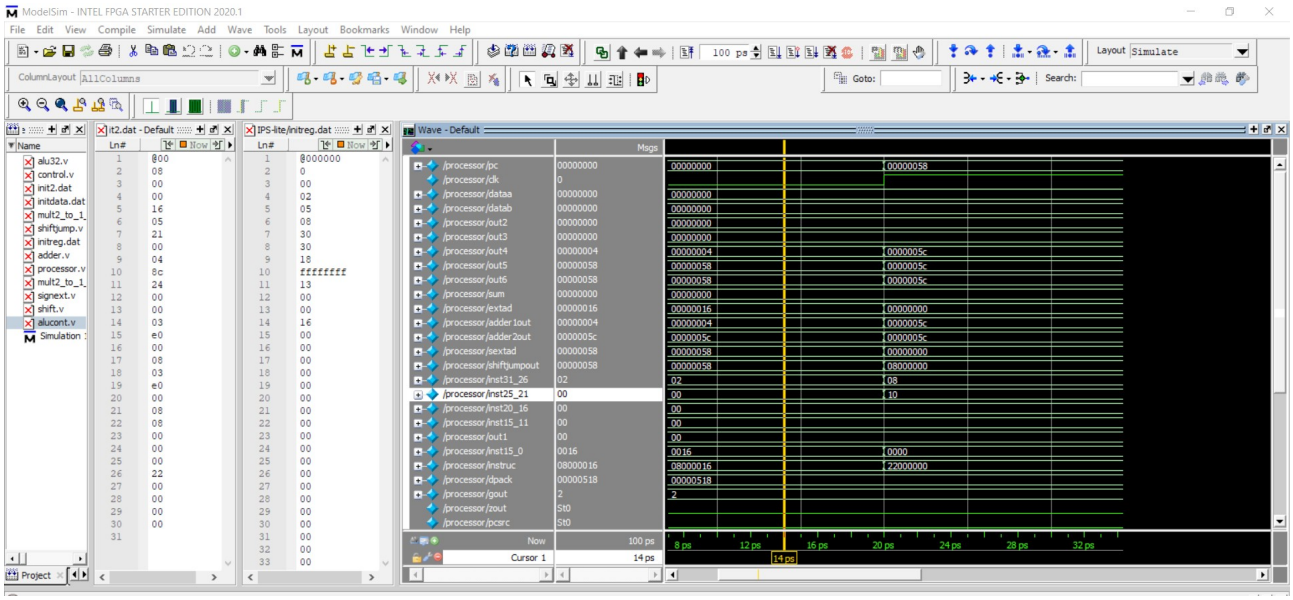


dataa is set as 0x9 ($r8 = $t0 = 0x9). datab is set as 0x13 ($r9 = $t1 = 0x13). Extad is set as 1(offset), then its shifted value, sextad, is 0x4. Gout says it is bne operation to alu32. Alu32 sets zout as 1 and sets sum as dataa-datab. And pcsrc is 1 so program jumps to adder2out.

7- j 0x0000016
Bit Representation: 000010 00000000000000000000010110
Hex: 0x08000016

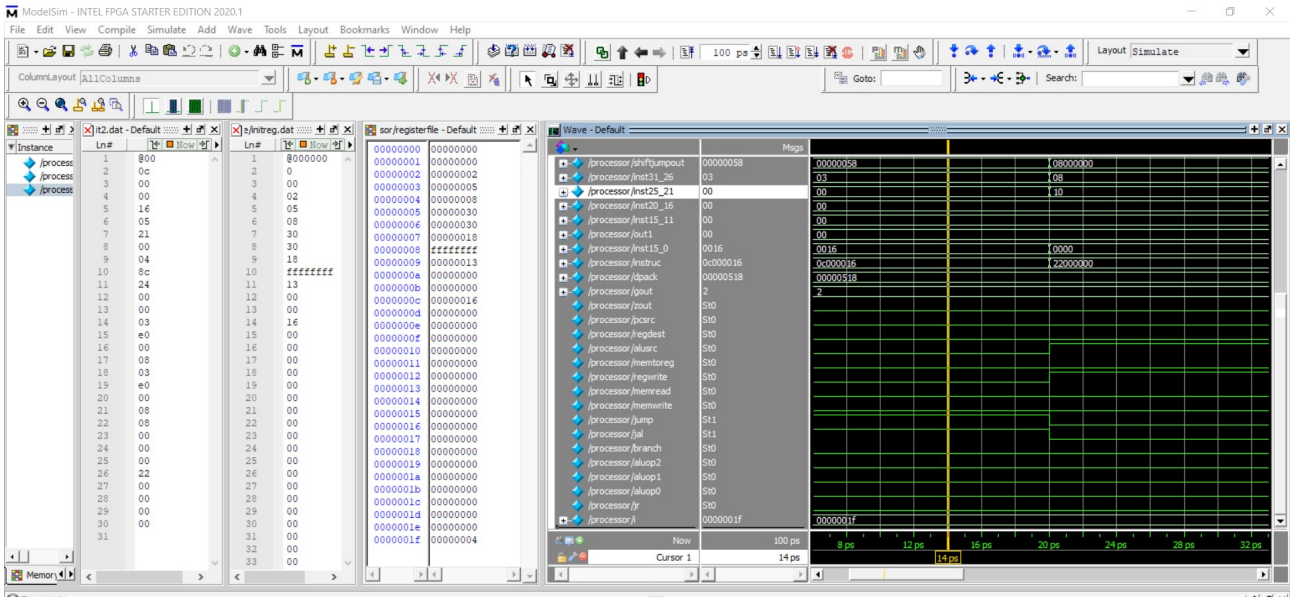


Control Bits: jump = 1, jr = 0

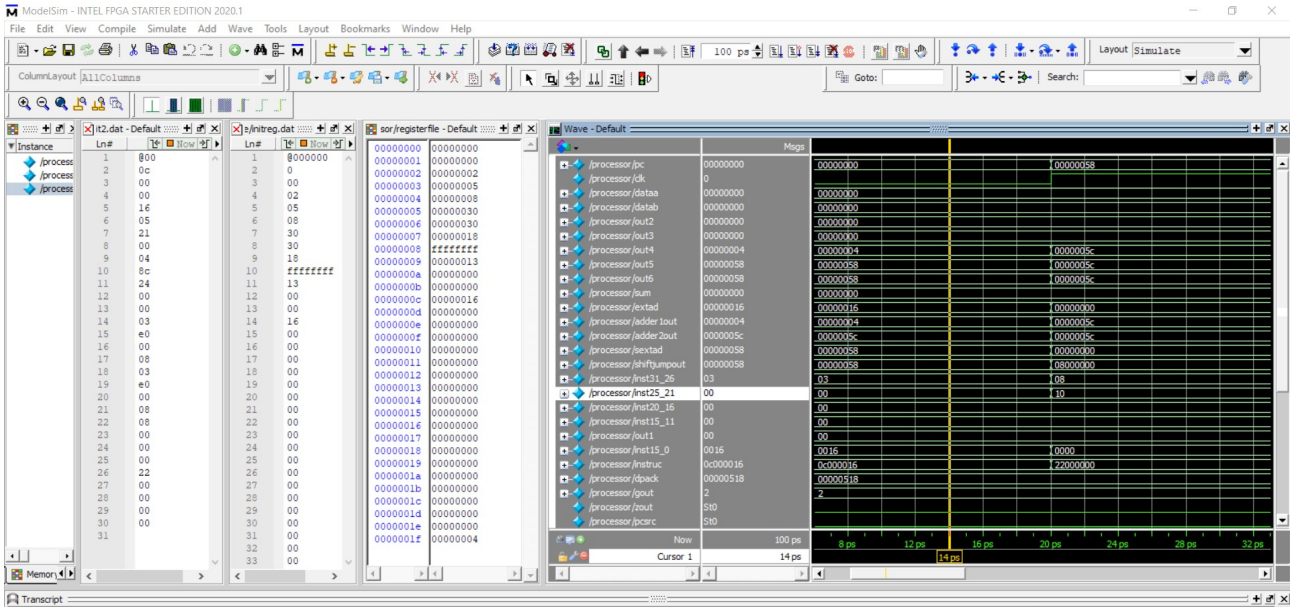


Immediate value is 0x16, shiftjump unit shifts 2 to the left immediate value and concatenate with PC+4[31:28] and jump bit is 1 so multiplexer 5 selects shiftjumpout value (out5). The jr signal is 0 so multiplexer 6 selects multiplexer 5 result (out6). And program jumps to 58.

8- jal 0x0000016
Bit Representation: 000011 00000000000000000000010110
Hex: 0x0C000016



Control Bits: jump = 1, jal = 1, jr = 0

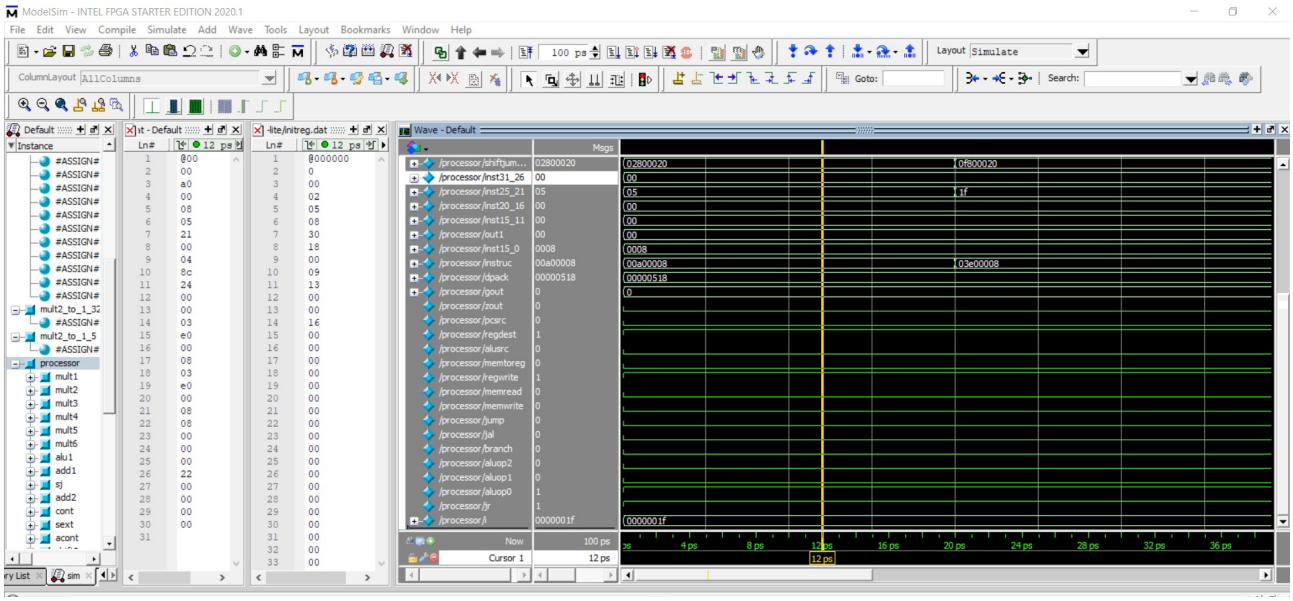


Immediate value is 0x16, shiftjump unit shifts 2 to the left immediate value and concatenate with pc+4[31:28] and jal bit is 1 so multiplexer 5 selects shiftjumpot value (out5).the jr signal is 0 so multiplexer 6 selects multiplexer 5 result(out6). And program jumps to 58. and $ra register value is PC+4 as you can see in the registerFile.

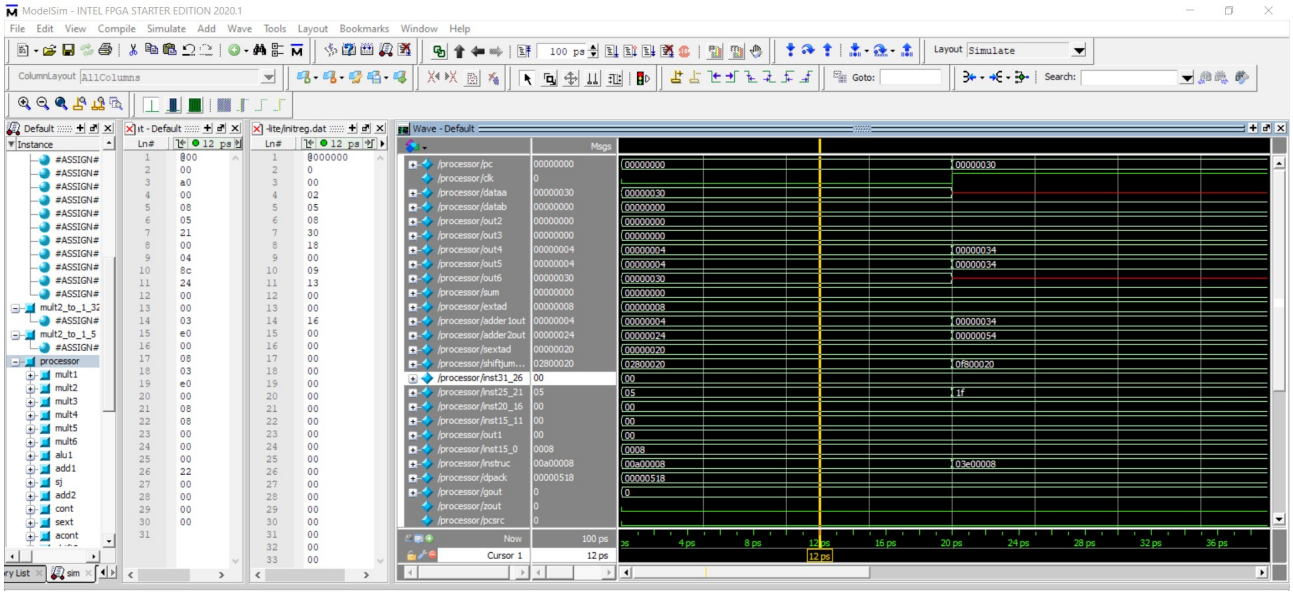
9- jr $a1

Bit Representation: 000000 00101 000000000000000 001000

Hex: 0x00A00008

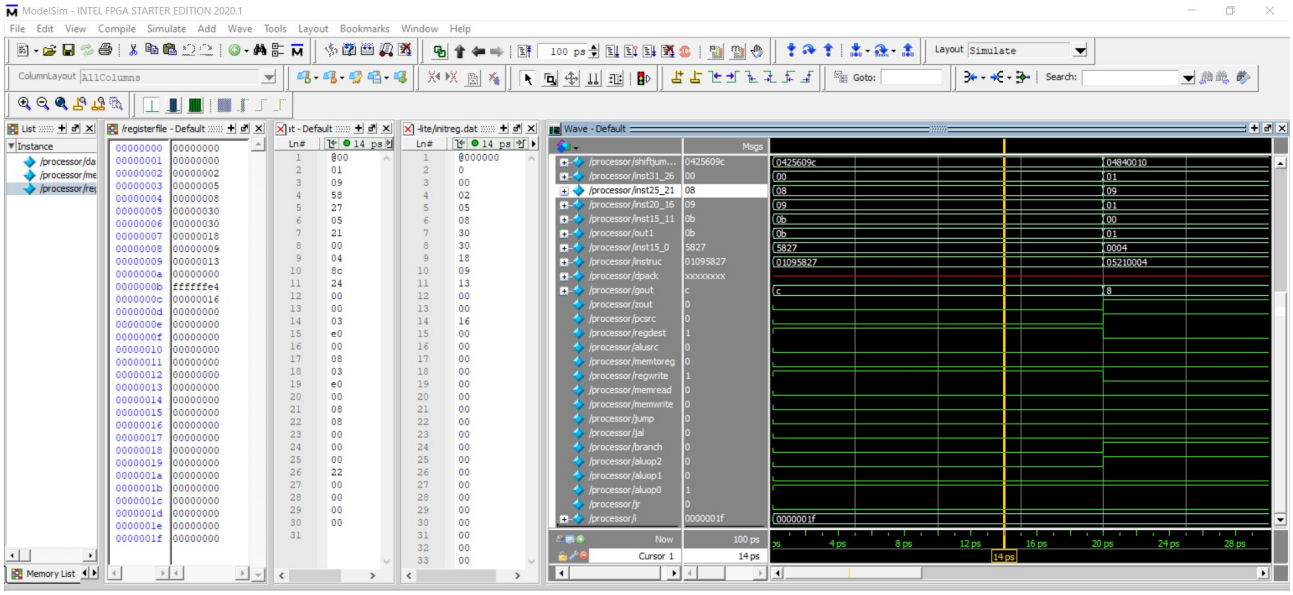


Control Bits: regdest = 1, regwrite = 1, aluop = 001, jr = 1

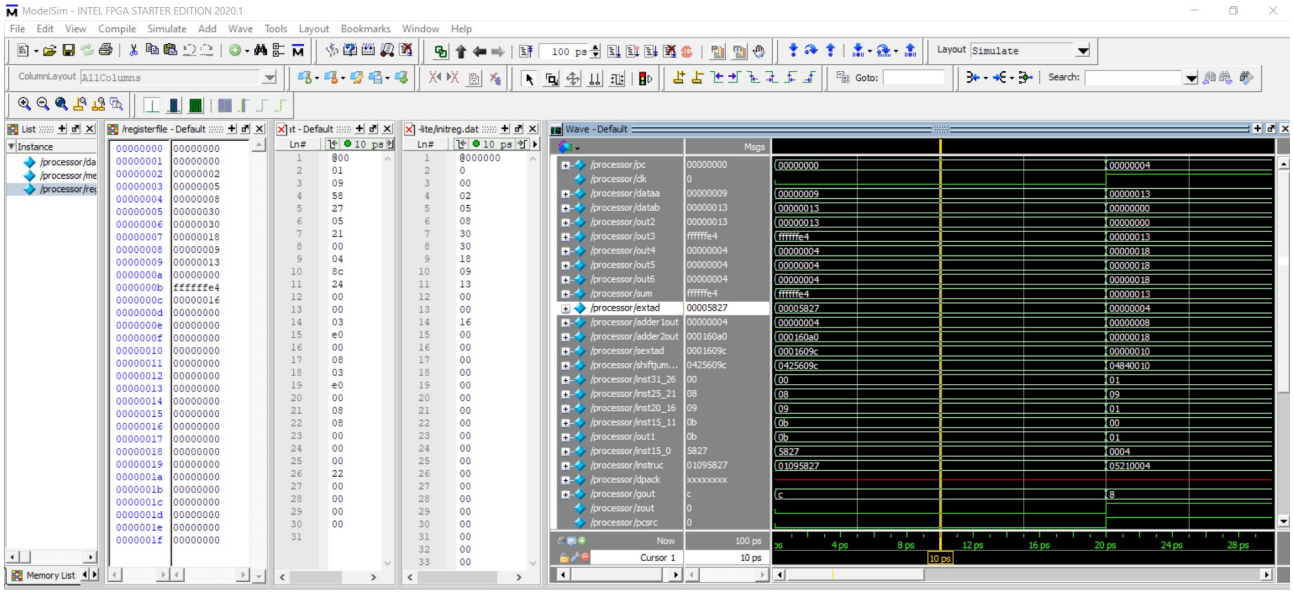


dataa is set to value in $r5($a1), out4 and out5 PC+4 because zout is 0. Multiplexer 6 selects out5 or dataa with checking jr signal. Jr bit is 1 so it selects dataa and out 6 is dataa. PC is set to dataa(0x30).

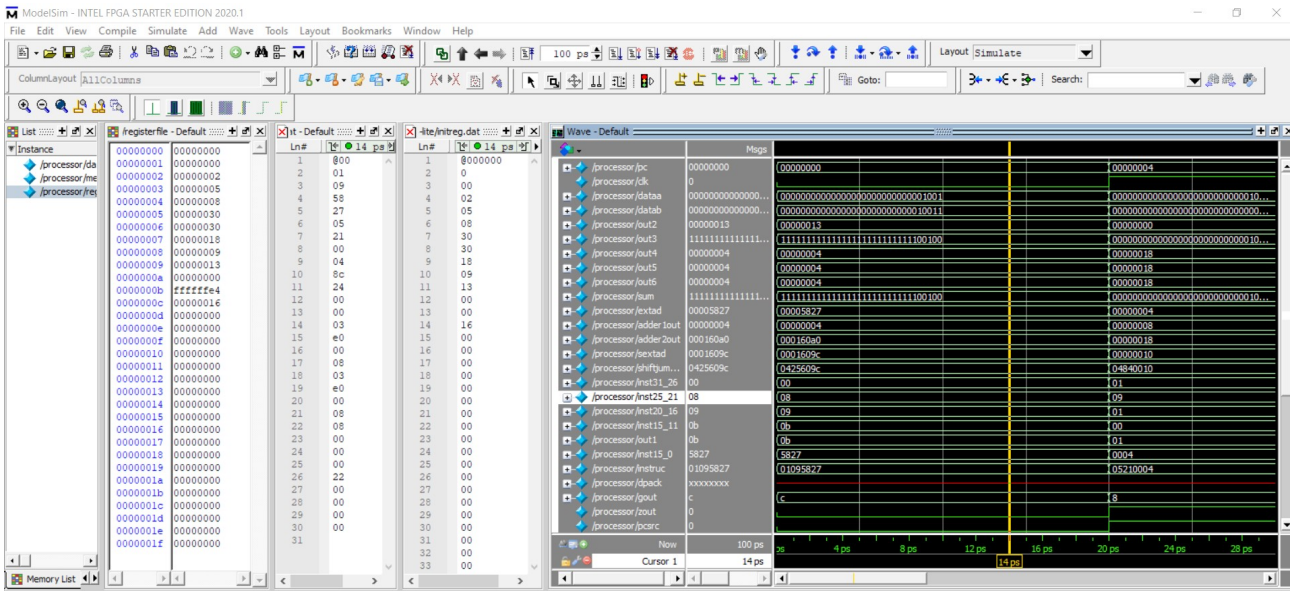
10- nor $t3, $t0, $t1
Bit Representation: 000000 01000 01001 01011 00000 100111
Hex: 0x01095827



Control Bits: regdest = 1, regwrite = 1, aluop = 001, gout = 0xc(1100)



dataa is set to 0x9 ($r8 = $t0 = 0x9). datab is set to 0x13 ($r9 = $t1 = 0x13). Out1 is set to 0xb(11) which is corresponding to $r11($t3). Sum is set to ~(dataa|datab). out3 is equal to sum because memtoreg control bit 0. Then $r11($t3) is set as ffffffe4 as you can see in registerFile.



This is the bit representation of datas in nor instruction.

Onur Cihangir
250201049