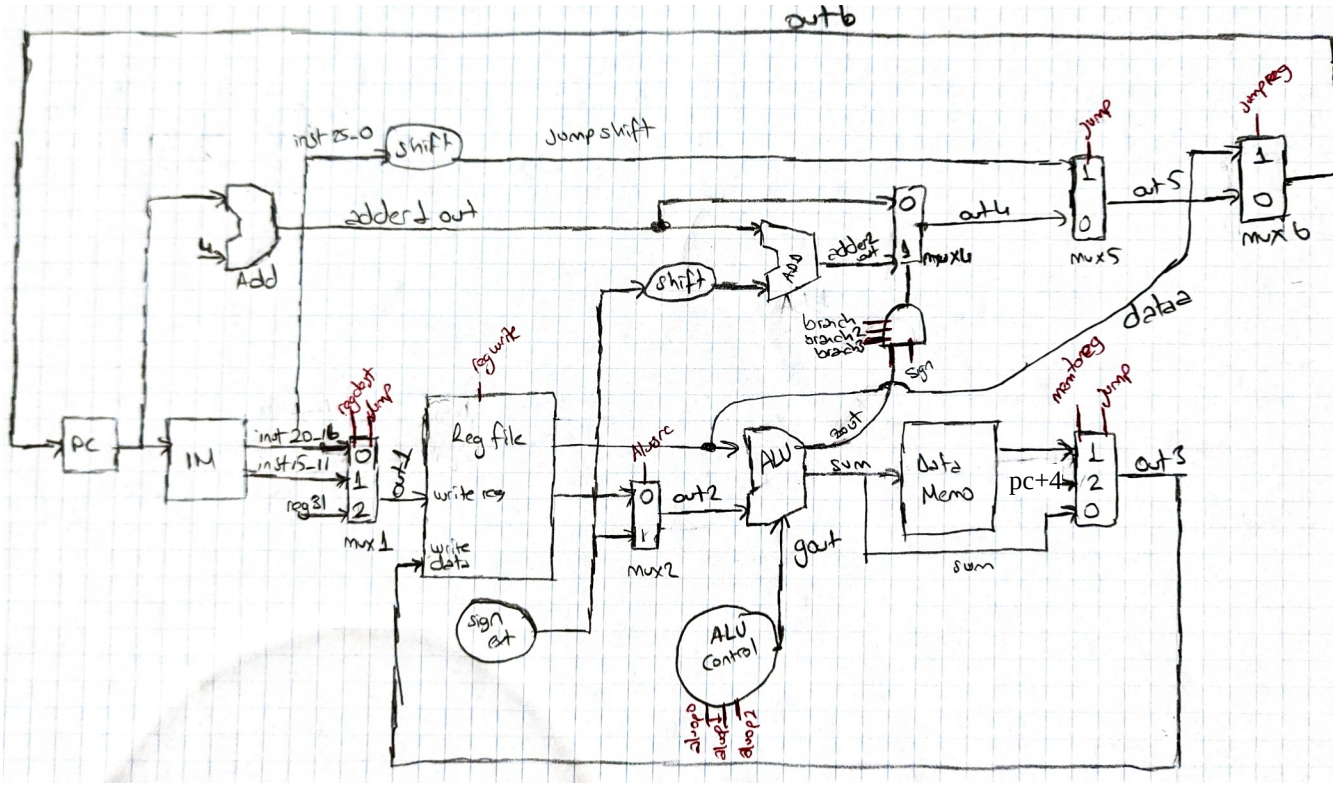
FURKAN ŞAHİN
250201042
CENG311- P3



The picture above includes name of some inputs,outputs and control bits. I developed the algorithm according to that picture. It helps to understand the documents easily.

Firstly I want to show you important changes in the code (please look at below) to understand algorithms easily. After that I explain all the instructions.

I changed the Mux1 and Mux3 which had one selection bit. I have new functions for mux1 and mux3 which have two selection bit.

Mult3_to_1_5 takes one output, two inputs and two selection bits and according to these two selection bits, I return the necessary input as the output. For example if S0 which corresponds regdest control bit is 0 and S1 which corresponds jump control bit is 1, I understand that jal instruction is being executed, so I have to set my write register as Register31.
Result of write register:

S0	S1	
0	1	= JAL → Write register is Reg31
1	0	= REGDEST → Write register is inst15_11
0	0	= NORMAL → Write register is inst20_16

Mult3_to_1_32 takes one output, three inputs and two selection bits and according to these two selection bits, I return the necessary input as the output. For example if S0 which corresponds memtoreg control bit is 0 and S1 which corresponds jump control bit is 1, I understand that jal instruction is being executed, so my output is set as i2 which is pc+8 and is written to Register31.
Result of output:

S0	S1	
0	1	= JAL → Output is pc+4
1	0	= MEMTOREG → Output is datapack
0	0	= NORMAL → Output is sum (aluresult)

```
module mult3_to_1_32(out,i0,i1,i2,s0,s1);
output [31:0] out;
input [31:0]i0,i1,i2; //i2 = pc+8
input s0,s1; //s0 = memtoreg , s1 = jal
reg  scontrol,scontrol2;

always @(s0,s1)
begin
if (s1&(~s0))
begin
        scontrol=1'b1;
        scontrol2=1'b0;
end
if ((~s1)&s0)
begin
        scontrol=1'b0;
        scontrol2=1'b1;
end
if ((~s1)&(~s0))
begin
        scontrol=1'b0;
        scontrol2=1'b0;
end
end
assign out = scontrol ? i2:(scontrol2 ? i1:i0);
endmodule

// S0,S1
// 01 = JAL
// 10 = MEMTOREG
// 00 = NORMAL
```

```
mult3_to_1_32 mult3(out3, sum,dpack,adderlout+32'h4,memtoreg,jump);
```

```
module mult3_to_1_5(out,i0,i1,s0,s1);
output [4:0] out;
input [4:0]i0,i1;
input s0,s1; //s0 = regdest , s1 = jal
reg  scontrol,scontrol2;
always @(s0,s1)
begin
if (s1&(~s0))
begin
        scontrol=1'b1;
        scontrol2=1'b0;
end
if ((~s1)&s0)
begin
        scontrol=1'b0;
        scontrol2=1'b1;
end
if ((~s1)&(~s0))
begin
        scontrol=1'b0;
        scontrol2=1'b0;
end
end
assign out = scontrol ? 5'b11111:(scontrol2 ? i1:i0);
endmodule

// S0,S1
// 01 = JAL
// 10 = REGDEST
// 00 = NORMAL
```

```
mult3_to_1_5  mult1(out1, instruc[20:16],instruc[15:11],regdest,jump);
```

I add a new module which takes sum (aluresult) and returns a signsrc control bit.
If sum is pozitive or zero, it returns 0. If sum is negative,it returns 1. I use that sign_control bit in the branch instructions (You see it in next pages)

```
module sign_control(out,sum);
output out;
input [31:0]sum;
reg  scontrol;
always @(sum)
begin
if (sum[31]==1'b1)
        scontrol=1'b1;
else
        scontrol=1'b0;
end
assign out = scontrol ? 1'b1:1'b0;
endmodule
```

I add necessary things to control. Rtfiled is for bgez,bgtz,blez,bltz controls. Funcfield is for JR control.

```
module control(in,rt,func,regdest,alusrc,memtoreg,regwrite,memread,memwrite,branch,branch2,branch3,aluop2,aluop1,aluop0,jump,jur
input [5:0] in,func;
input [4:0] rt;
output regdest,alusrc,memtoreg,regwrite,memread,memwrite,branch,branch2,branch3,aluop0,aluop1,aluop2,jump,jumpreg;
wire rformat,lw,sw,beq,bne,bgez,bgtz,blez,bltz,addi,andi,ori,j,jr,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];
assign beq=~in[5]& (~in[4])&(~in[3])&in[2]&(~in[1])&(~in[0]);
assign addi=(~in[5])& (~in[4])& in[3]& (~in[2])&(~in[1])&(~in[0]);
assign andi=(~in[5])& (~in[4])& in[3]& in[2]&(~in[1])&(~in[0]);
assign ori=(~in[5])& (~in[4])& in[3]& in[2]&(~in[1])&in[0];
assign bne=~in[5]& (~in[4])&(~in[3])&in[2]&(~in[1])&in[0];
assign bgez=~in[5]& (~in[4])&(~in[3])&(~in[2])&(~in[1])&in[0]&(~rt[4])&(~rt[3])&(~rt[2])&(~rt[1])&rt[0];
assign bgtz=~in[5]& (~in[4])&(~in[3])&in[2]&in[1]&in[0]&(~rt[4])&(~rt[3])&(~rt[2])&(~rt[1])&(~rt[0]);
assign blez=~in[5]& (~in[4])&(~in[3])&in[2]&in[1]&(~in[0])&(~rt[4])&(~rt[3])&(~rt[2])&(~rt[1])&(~rt[0]);
assign bltz=~in[5]& (~in[4])&(~in[3])&(~in[2])&(~in[1])&in[0]&(~rt[4])&(~rt[3])&(~rt[2])&(~rt[1])&(~rt[0]);
assign j= ~in[5]& (~in[4])&(~in[3])&(~in[2])&in[1]&(~in[0]);
assign jr= (~|in)& (~func[5])&(~func[4])&func[3]&(~func[2])&(~func[1])&(~func[0]);
assign jal= ~in[5]& (~in[4])&(~in[3])&(~in[2])&in[1]&in[0];
assign regdest=rformat;
assign alusrc=lw|sw|addi|andi|ori;
assign memtoreg=lw;
assign regwrite=rformat|lw|addi|andi|ori|jal;
assign memread=lw;
assign memwrite=sw;
assign branch=beq|bne|bgez|blez;
assign branch2=bgez|bgtz|bne;
assign branch3=blez|bltz|bne;
```

Most important thing is that : I have three branch control bits for branch instructions. You see the settings in the comments. I use that 3 bits of branches to set pcsrc.

```
assign branch=beq|bne|bgez|blez;
assign branch2=bgez|bgtz|bne;
assign branch3=blez|bltz|bne;
assign aluop2=andi|ori;
assign aluop1=rformat;
assign aluop0=beq|andi|bne|bgez|bgtz|blez|bltz;
assign jump=j|jal;
assign jumpreg=jr;
endmodule

//      branch  branch2 branch3
//      1       0       0       = for BEQ
//      1       1       0       = for BGEZ
//      0       1       0       = for BGTZ
//      1       0       1       = for BLEZ
//      0       0       1       = for BLTZ
//      1       1       1       = for BNE

//      aluop2  aluop1  aluop0
//      0       0       0       = addi -> alucont.v gout = b0010 add operation
//      0       0       1       = beq, bne, bgez, bqtz, blez, bltz -> alucont.v gout = b0110 sub operation
//      0       1       0       = R -> alucont.v gout = Rtype selection
//      0       1       1       = empty
//      1       0       0       = ori -> alucont.v gout = b0001 or operation
//      1       0       1       = andi -> alucont.v gout = b0000 and operation
//      1       1       0       = empty
//      1       1       1       = empty
```

```
//AND gate
assign pcsrc=
 (branch && (~branch2) && (~branch3) && zout) // for BEQ
|(branch && branch2 && branch3 && (~zout)) // for BNE
|(branch && branch2 && (~branch3) && zout && (~signsrc)) // for BGEZ
|(branch && branch2 && (~branch3) && (~zout) && (~signsrc)) // for BGEZ
|((~branch) && branch2 && (~branch3) && (~zout) && (~signsrc)) // for BGTZ
|(branch && (~branch2) && branch3 && zout &&(~signsrc)) // for BLEZ
|(branch && (~branch2) && branch3 && (~zout) && signsrc) // for BLEZ
|((~branch) && (~branch2) && branch3 && (~zout) && signsrc); // for BLTZ
```

For example, in control.v I examine the in input and rtfield input and I understand that current instruction is a BGTZ instruction. And I set branch, branch2, branch3 as you see in the comments.
In the processor code, I set pcsrc according to these 3 branch bit + zout bit + signsrc bit (signsrc is explained in above) According to situations I set pcsrc 1 or 0.

Example : bgtz r5, L and r5 is negative.
For bgtz : branch=0, branch2=1, branch3= 0, zout = 0, signsrc =1. So it does not correspond BGTZ and pcsrc is set to zero and just execute next instruction.
If r5 is pozitive : branch=0, branch2=1, branch3= 0, zout = 0, signsrc =0. So it corresponds BGTZ and pcsrc is set to one to jump given label.
If r5 is zero : branch=0, branch2=1, branch3= 0, zout = 1, signsrc =0. Not correspond, no jumping and just execute next instruction.

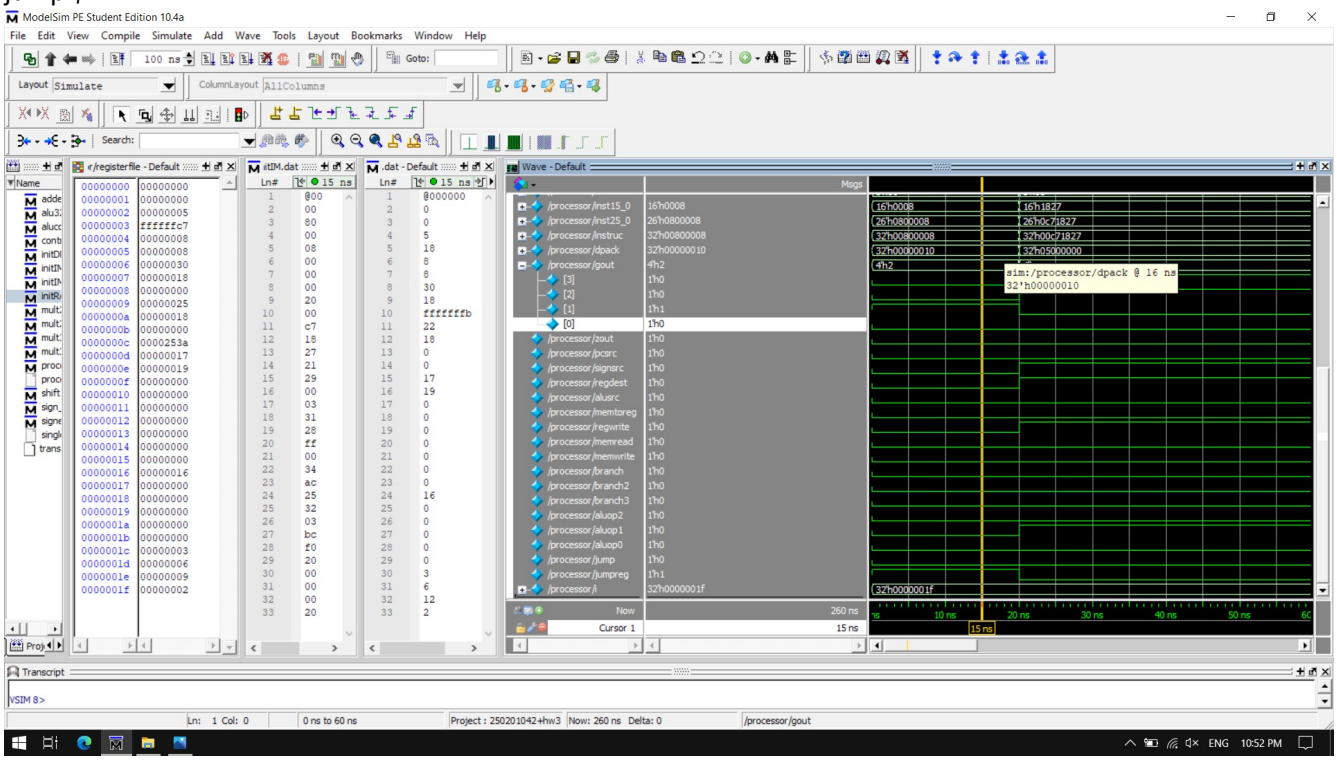
```
assign branch=beq|bne|bgez|blez;
assign branch2=bgez|bgtz|bne;
assign branch3=blez|bltz|bne;
assign aluop2=andi|ori;
assign aluop1=rformat;
assign aluop0=beq|andi|bne|bgez|bgtz|blez|bltz;
assign jump=j|jal;
assign jumpreg=jr;
endmodule

//      branch  branch2 branch3
//      1       0       0       = for BEQ
//      1       1       0       = for BGEZ
//      0       1       0       = for BGTZ
//      1       0       1       = for BLEZ
//      0       0       1       = for BLTZ
//      1       1       1       = for BNE

//      aluop2  aluop1  aluop0
//      0       0       0       = addi -> alucont.v gout = b0010 add operation
//      0       0       1       = beq, bne, bgez, bqtz, blez, bltz -> alucont.v gout = b0110 sub operation
//      0       1       0       = R -> alucont.v gout = Rtype selection
//      0       1       1       = empty
//      1       0       0       = ori -> alucont.v gout = b0001 or operation
//      1       0       1       = andi -> alucont.v gout = b0000 and operation
//      1       1       0       = empty
//      1       1       1       = empty
```

For alu operations I define three aluop bits which are aluop0, aluop1, aluop2 for aluoperations. You see the settings in the comments. I use that 3 bits of aluop to set gout in the alucont.v. According to gout it executes some operations in alu32.v

```
1   module alucont(aluop2,aluop1,aluop0,f3,f2,f1,f0,gout);//Figure 4.12
2   input aluop2,aluop1,aluop0,f3,f2,f1,f0;
3   output [3:0] gout;
4   reg [3:0] gout;
5   always @(aluop2 or aluop1 or aluop0 or f3 or f2 or f1 or f0)
6   begin
7   if(~(aluop2|aluop1|aluop0))gout=4'b0010;
8   if((~aluop1) & (~aluop2) & aluop0)gout=4'b0110;
9   if(aluop2 & (~aluop1) & aluop0)gout=4'b0000;
10  if(aluop2 & (~aluop1) &(~aluop0)) gout=4'b0001;
11  if((~aluop2) & aluop1 & (~aluop0))//R-type
12  begin
13          if (~(f3|f2|f1|f0))gout=4'b0010;        //function code=0000,ALU control=0010 (add)
14          if (f1&f3)gout=4'b0111;                 //function code=1x1x,ALU control=0111 (set on les
15          if (f1&~(f3))gout=4'b0110;              //function code=0x10,ALU control=0110 (sub)
16          if (f2&f0)gout=4'b0001;                 //function code=x1x1,ALU control=0001 (or)
17          if (f2&~(f0))gout=4'b0000;              //function code=x1x0,ALU control=0000 (and)
18          if ((~f3)&f2&f1&f0)gout=4'b1100;        //function code=0111,ALU control=1100 (nor)
19  end
20  end
21  endmodule
```

So we can look at the instructions now.

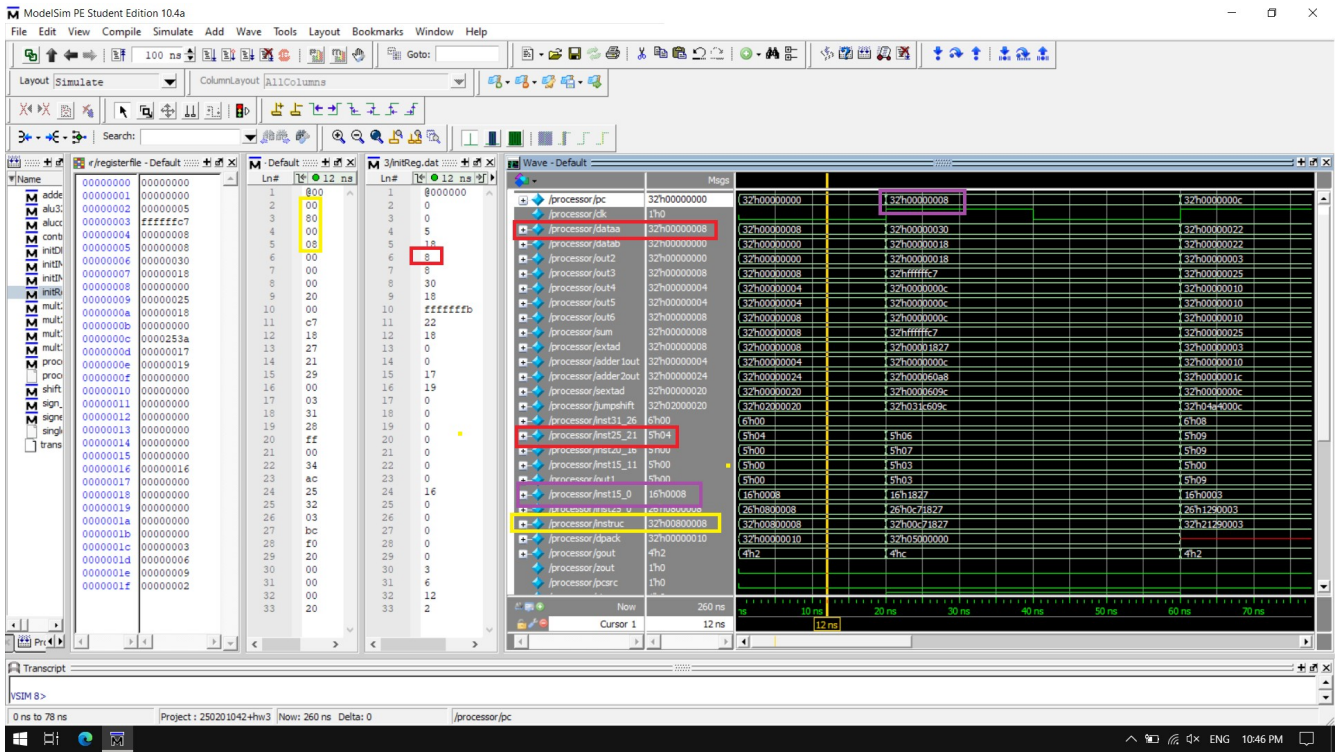
1- JR : IM.dat
000000 00100 00000 00000 00000 001000 | in hex = 00800008
jump $r4.



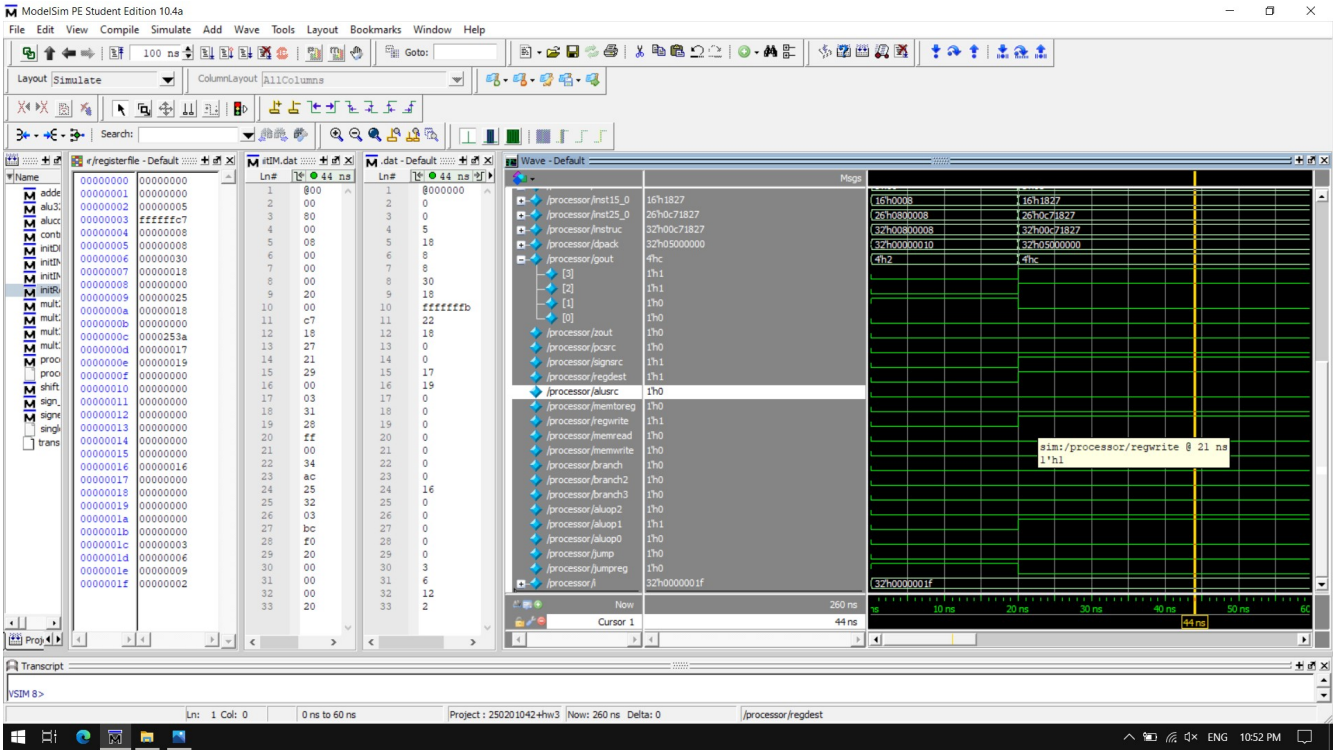
Control bits : jumpreg is 1 (it goes mux6). So dataa is assigned to out6. Others are zero.

Please look at next page for details

As you can see below, first instruction which is 00800008 is executed. $r4(inst25_21) which is sixth line (rX = (x+2)th line in reg.dat) in reg.dat and it is defined as 8 in reg.dat. And dataa is defined as 8. Mux4 and mux5 outputs are defined as pc+4 because of that mux4 and mux5 get zero as control bit. Mux6 get out5(pc+4),jumpadress(dataa) and jumreg control bit. Jumpreg control bit is 1 so it returns out6 as jump address and pc is defined as jumpaddress.



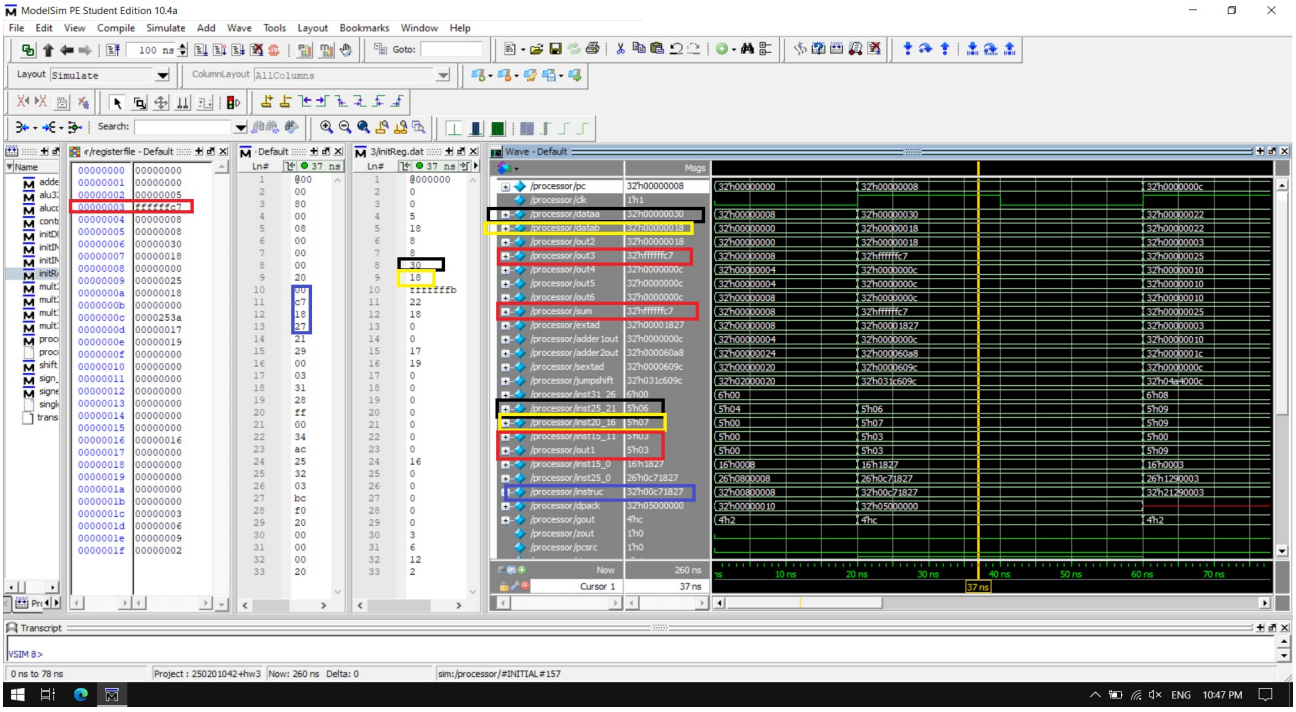
2- NOR: IM.dat
000000 00110 00111 00011 00000 100111 | in hex = 00C71827
nor $r3 $r6 $r7



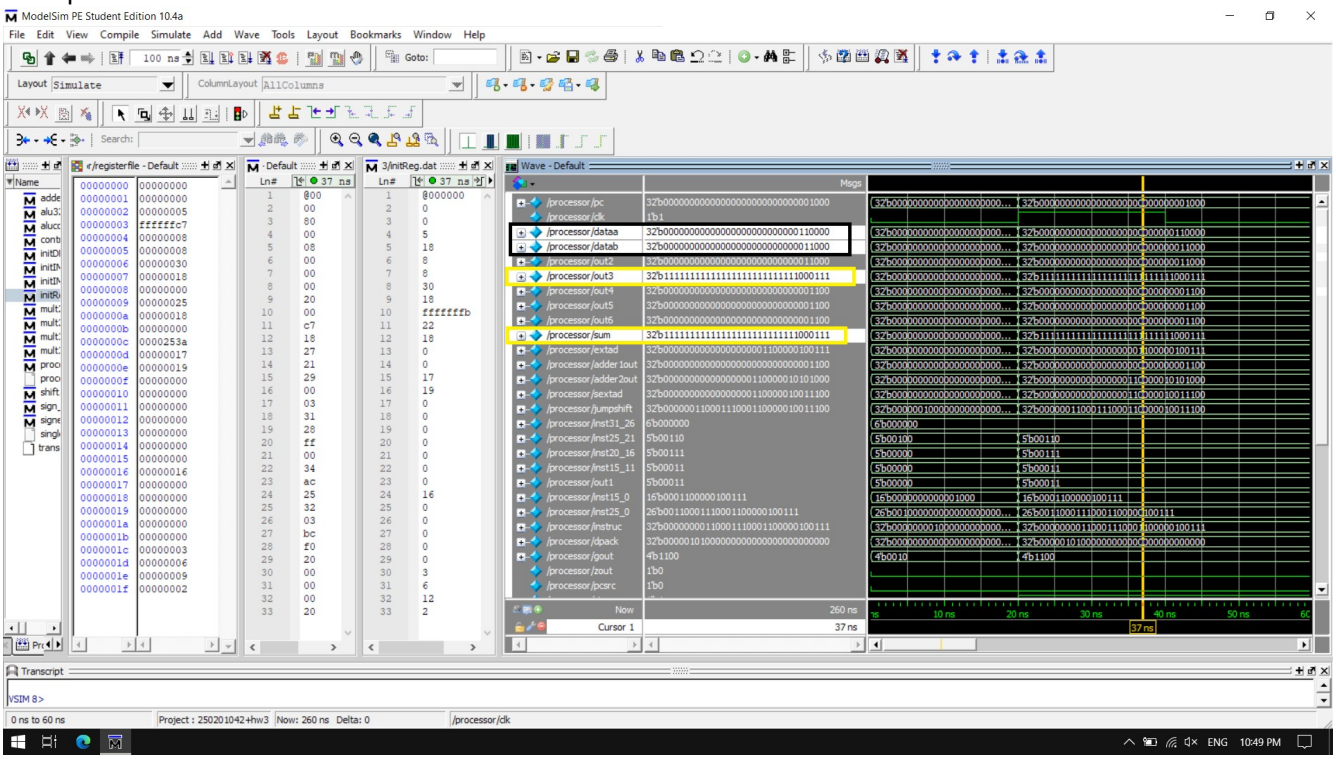
Control bits : regdest and regwrite are 1. aluop2=0,aluop1=1,aluop0=0 so aluop is 010 which is defined as R type and in the if condition of R-type : if ((~f3)&f2&f1&f0)gout=4'b1100 corresponds situation.
gout is defined as 1100 which corresponds NOR operation in alu32.v

Please look at next page for details

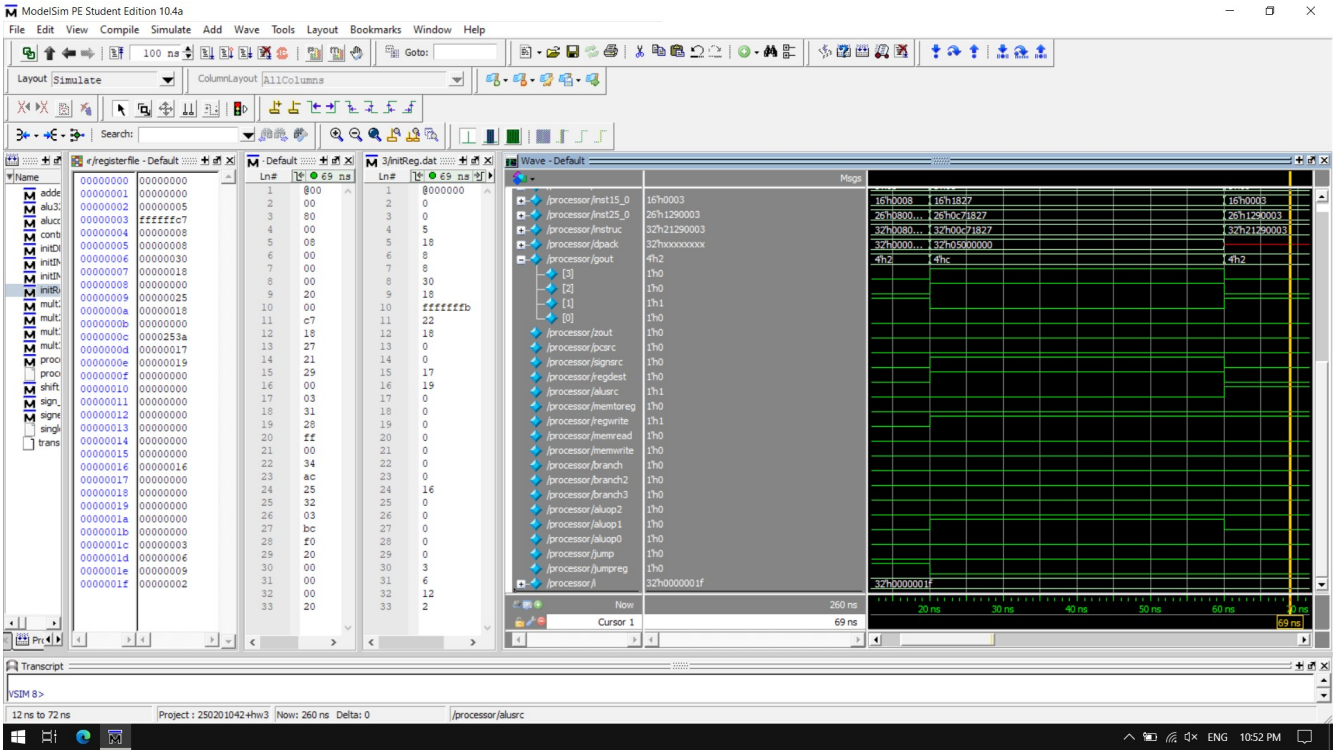
As you can see below, instruction which is 00C71827 is executed after jr $r4(8). $r3 is set as write register. $r6 is 30 which is defined as dataa, $r7 is 18 which is defined as datab. Alu executes ~(dataa | datab) and the result is assigned to sum. Mux3 output which is out3 is defined as sum result and sum result is written to the $r3.



Bit representation for NOR



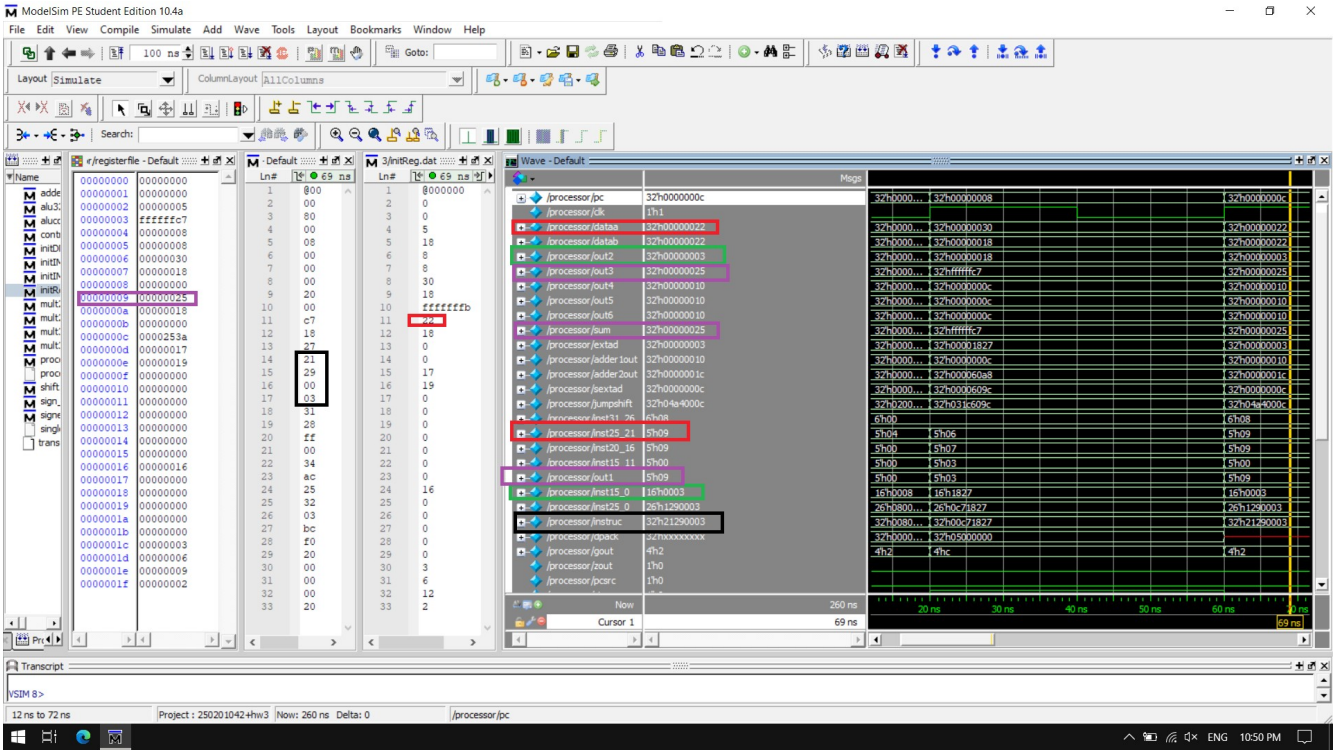
3- Addi: IM.dat
001000 01001 01001 0000000000000011 | in hex = 21290003
addi $r9 $r9 3 → $r9 = $r9 + 3



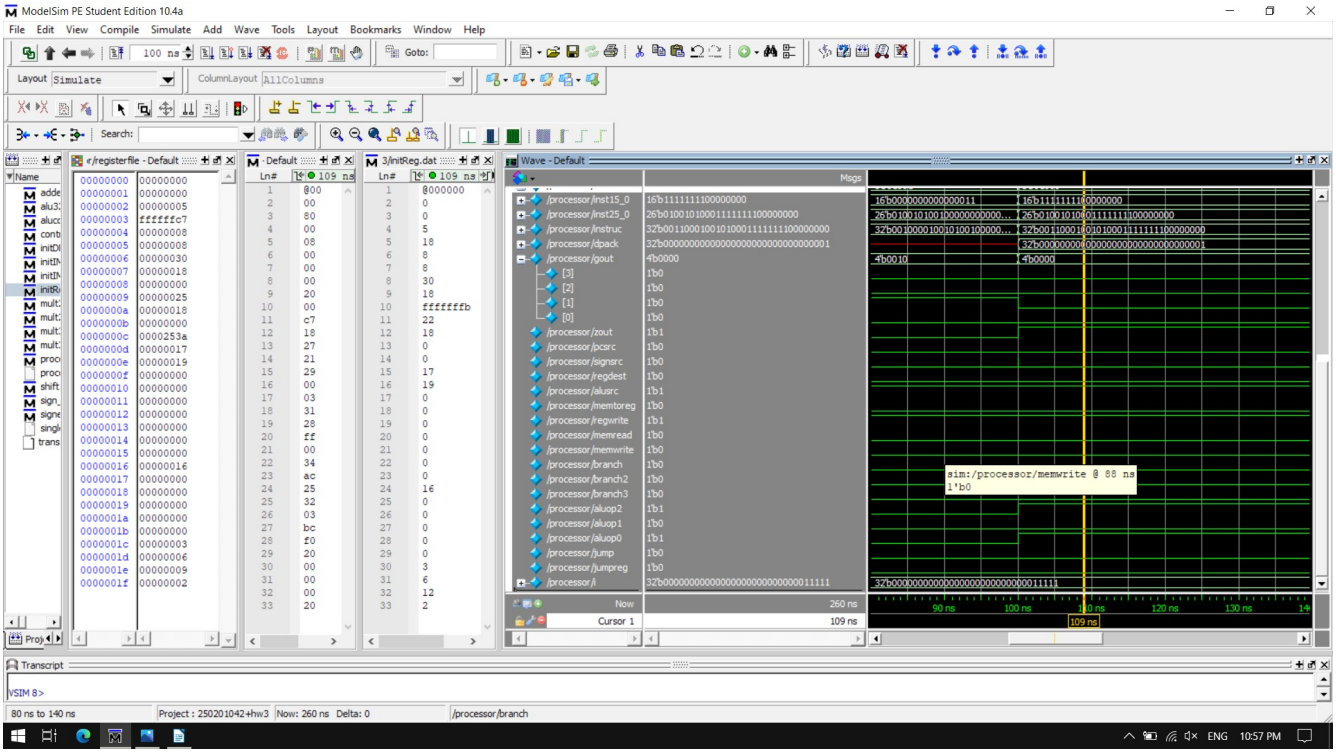
Control bits : alusrc and regwrite are 1. aluop2=0,aluop1=0,aluop0=0 so aluop is 000 which corresponds if(~(aluop2|aluop1|aluop0))gout=4'b0010 in alucont.v. gout is defined as 0010 which corresponds add operation in alu32.v

Please look at next page for details

As you can see below, instruction which is 21290003 is executed. $r9 is set as write register. $r9 is 22 which is defined as dataa, alusrc is 1 so mux2 returns inst15_0 which is 3 as out2. And gout says to alucontrol that it is an addition operation. dataa+out2 is defined in sum as 25. Mux3 returns sum as output3 because of that its control bits are 0. Lastly output3 goes to write data and it is written ro $r9 as 25.



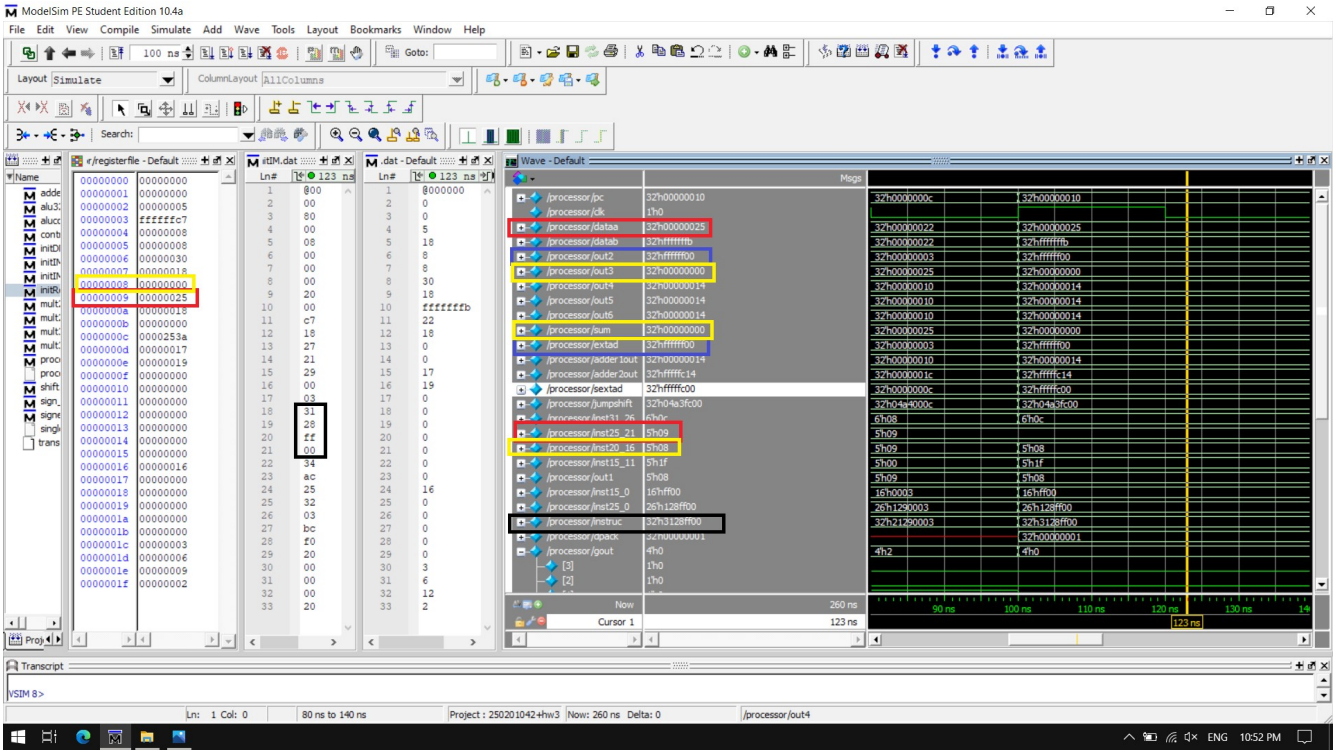
4- Andi: IM.dat
001100 01010 01000 1111111100000000 | in hex = 3128FF00
andi $r8 $r9 imm. → $r8 = $r9 & imm



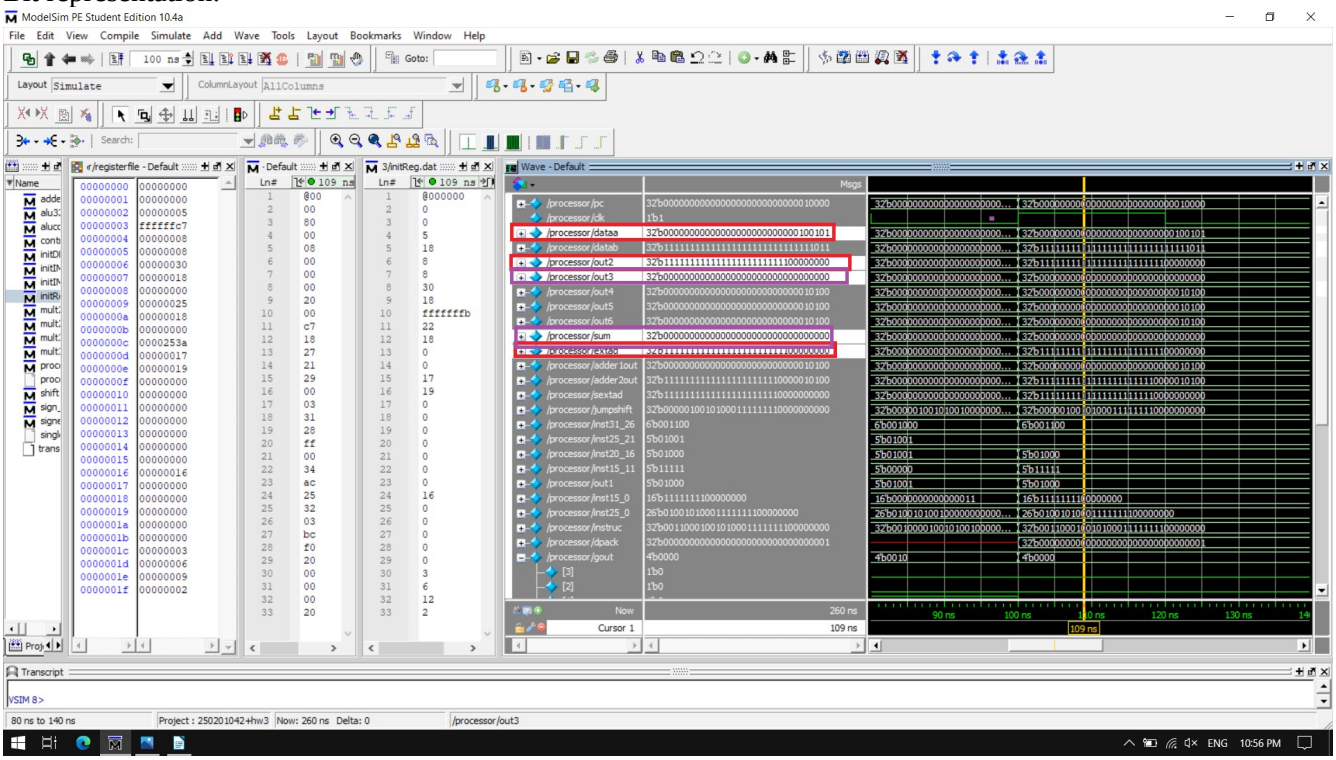
Control bits : alusrc and regwrite are 1. aluop2=1,aluop1=0,aluop0=1 so aluop is 101 which corresponds if(aluop2 & (~aluop1) & aluop0)gout=4'b0000 in alucont.v. gout is defined as 0000 which corresponds and operation in alu32.v

Please look at next page for details

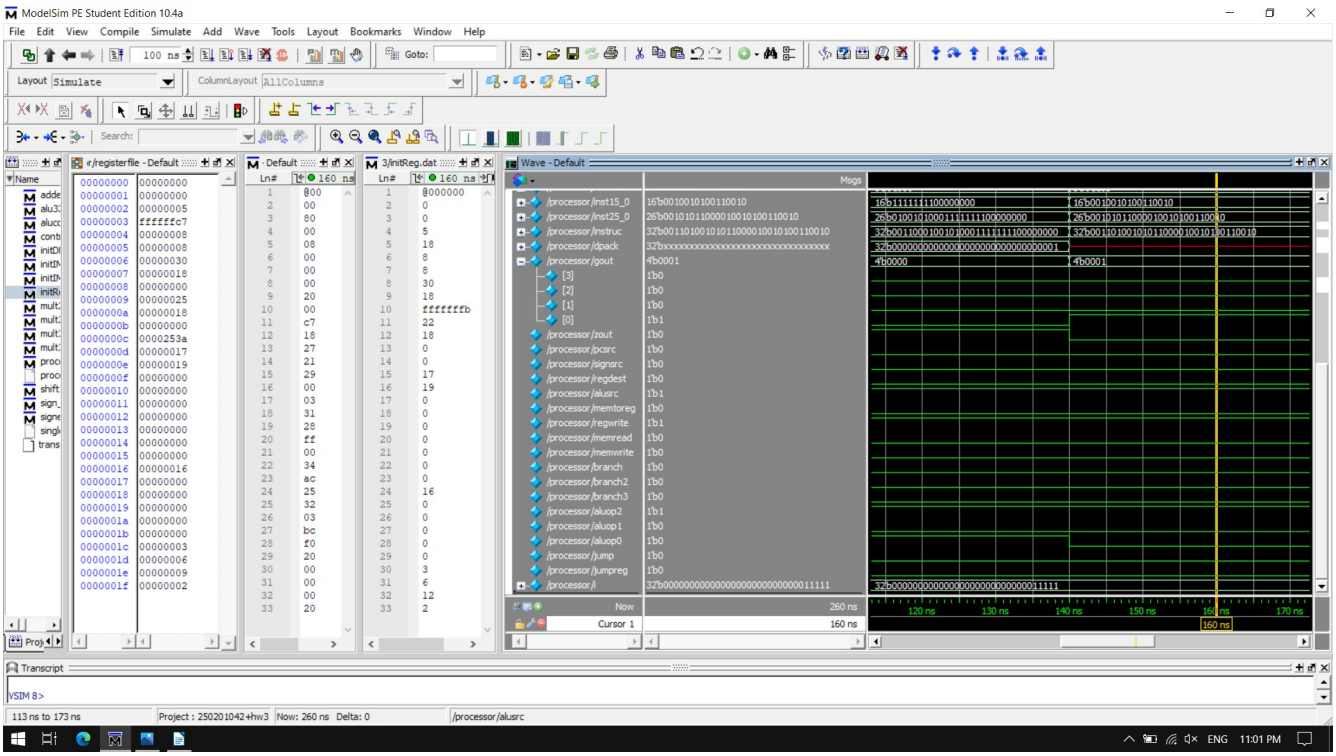
As you can see below, instruction which is 3128FF00 is executed. $r8 is set as write register. $r9 is 25 (after addi operation, $r9 is updated as 25) which is defined as dataa, alusrc is 1 so mux2 returns inst15_0 as out2 (extad) . And gout says to alucontrol that it is an and operation. Dataa & out2 is defined in sum as 0. Mux3 returns sum as output3 because of that its control bits are 0. Lastly output3 goes to write data and output3 is written to $r8.



Bit representation:



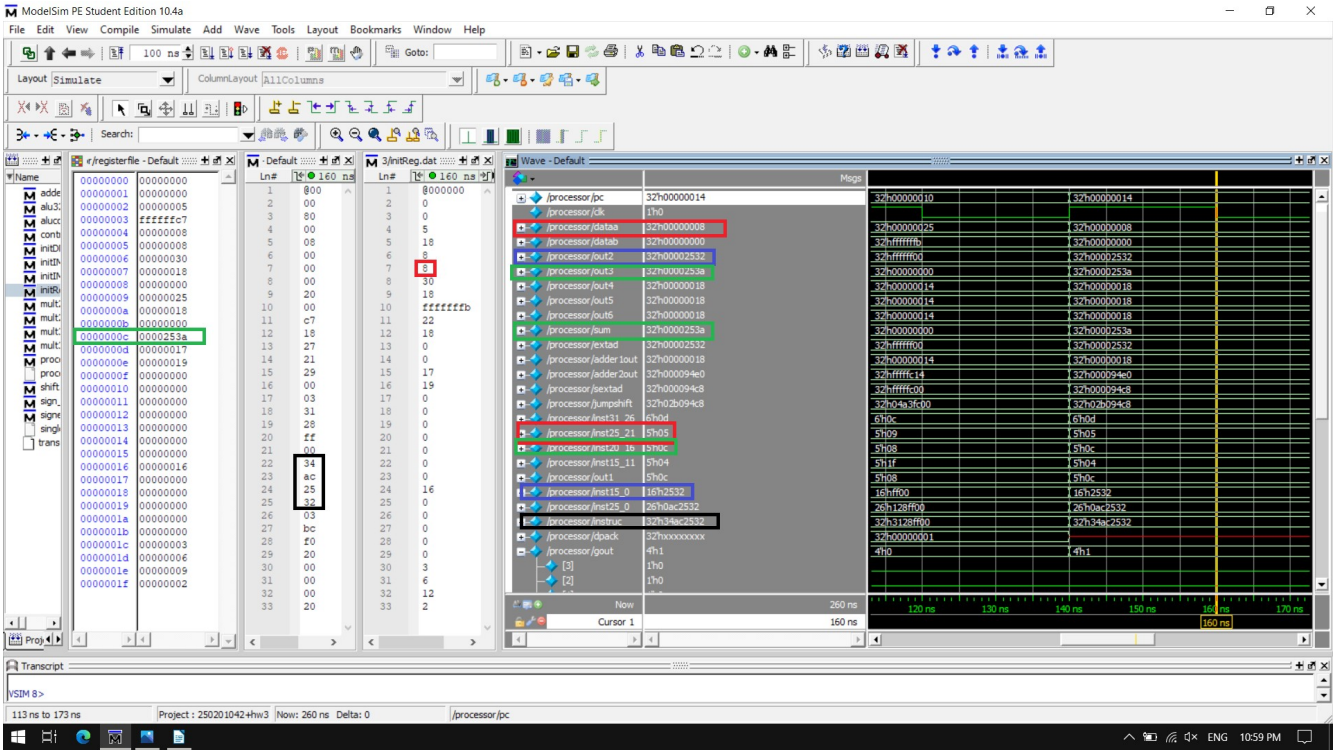
5- Ori : IM.dat
001101 00101 01100 0010010100110010 | in hex = 34AC2532
ori $r12 $r5 imm → $r12 = $r5 | imm



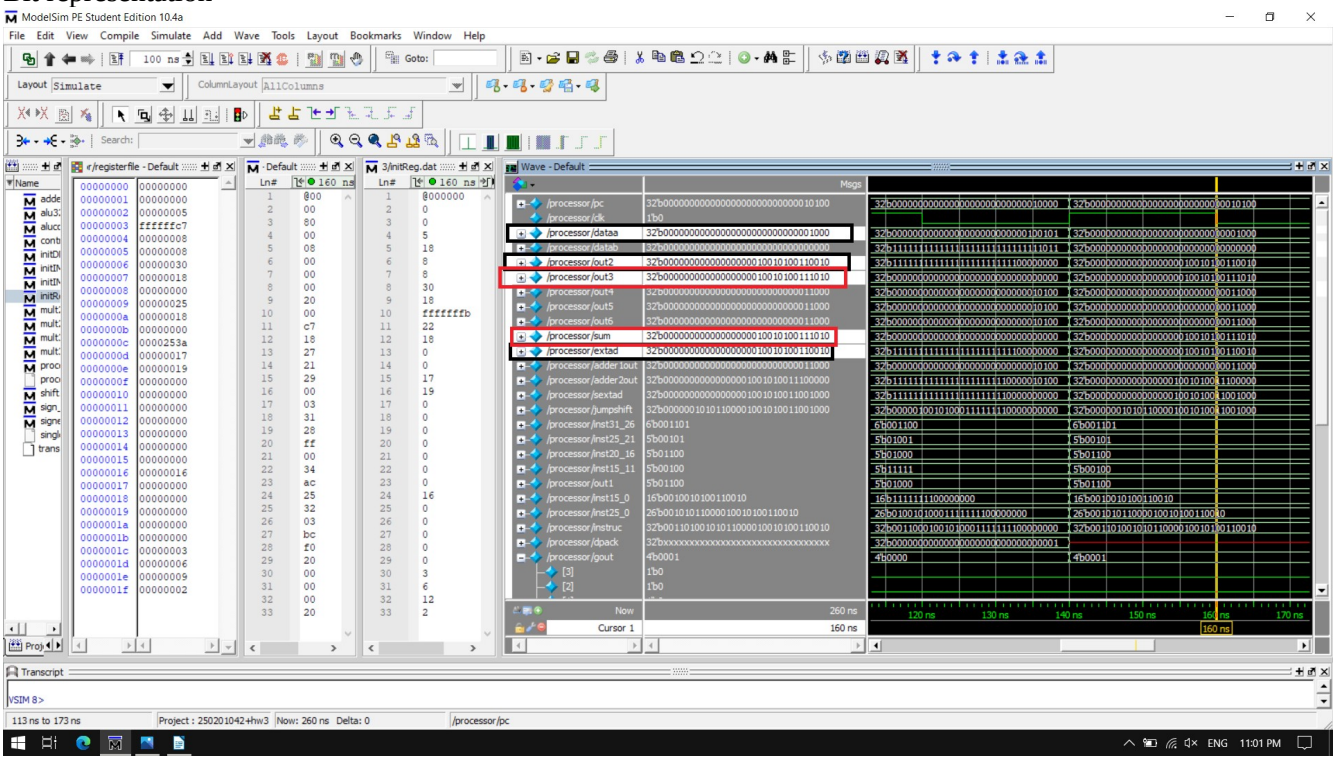
Control bits : alusrc and regwrite are 1. aluop2=1,aluop1=0,aluop0=0 so aluop is 100 which corresponds if(aluop2 & (~aluop1) & (~aluop0)) gout=4'b0001 in alucont.v. gout is defined as 0001 which corresponds or operation in alu32.v

Please look at next page for details

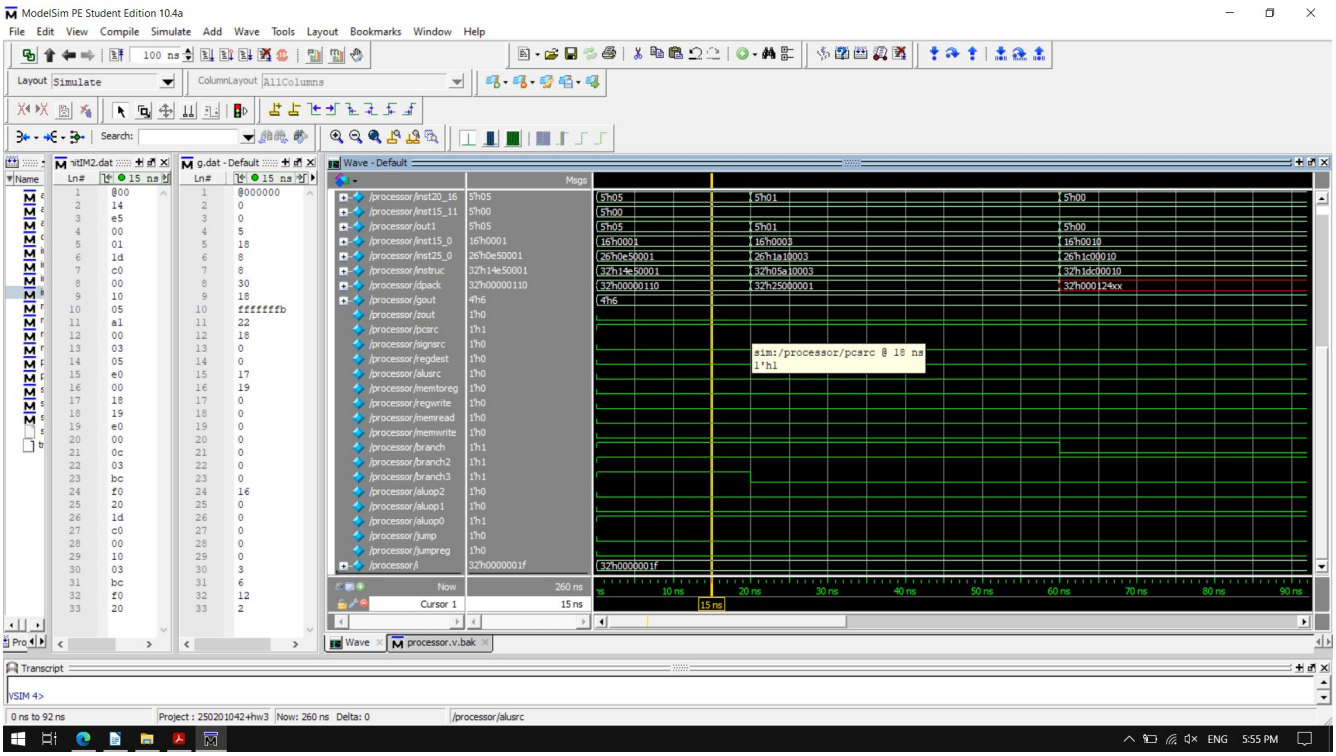
As you can see below, instruction which is 34AC2532 is executed. $r12 (0c in hexa) is set as write register. $r5 is 8 in hex which is defined as dataa, alusrc is 1 so mux2 returns inst15_0 as out2 (extad) . And gout says to alucontrol that it is an or operation. Dataa | out2 is defined in sum. Mux3 returns sum as output3 because of that its control bits are 0. Lastly output3 goes to write data and output3 is written to $r12.



Bit representation



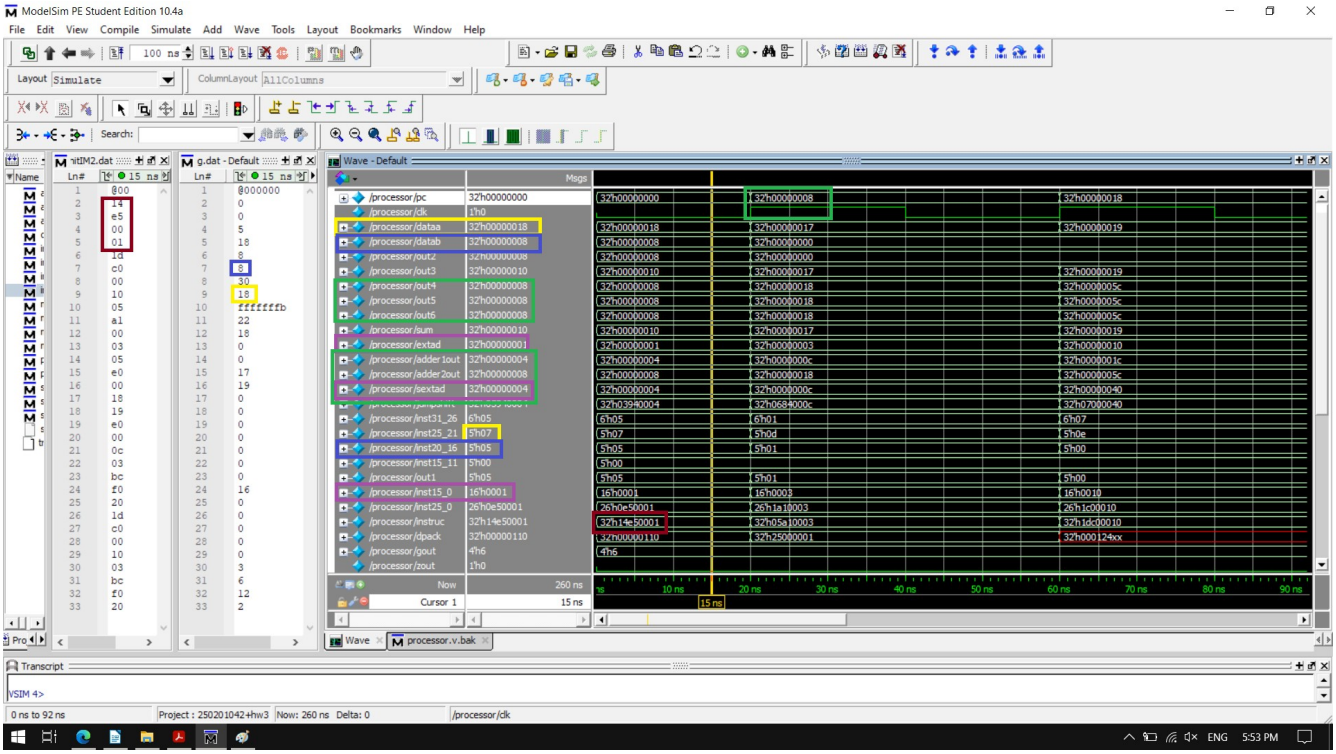
6- Bne : IM2.dat
000101 00111 00101 0000000000000001 | in hex = 14E50001
bne $r7 $r5 1



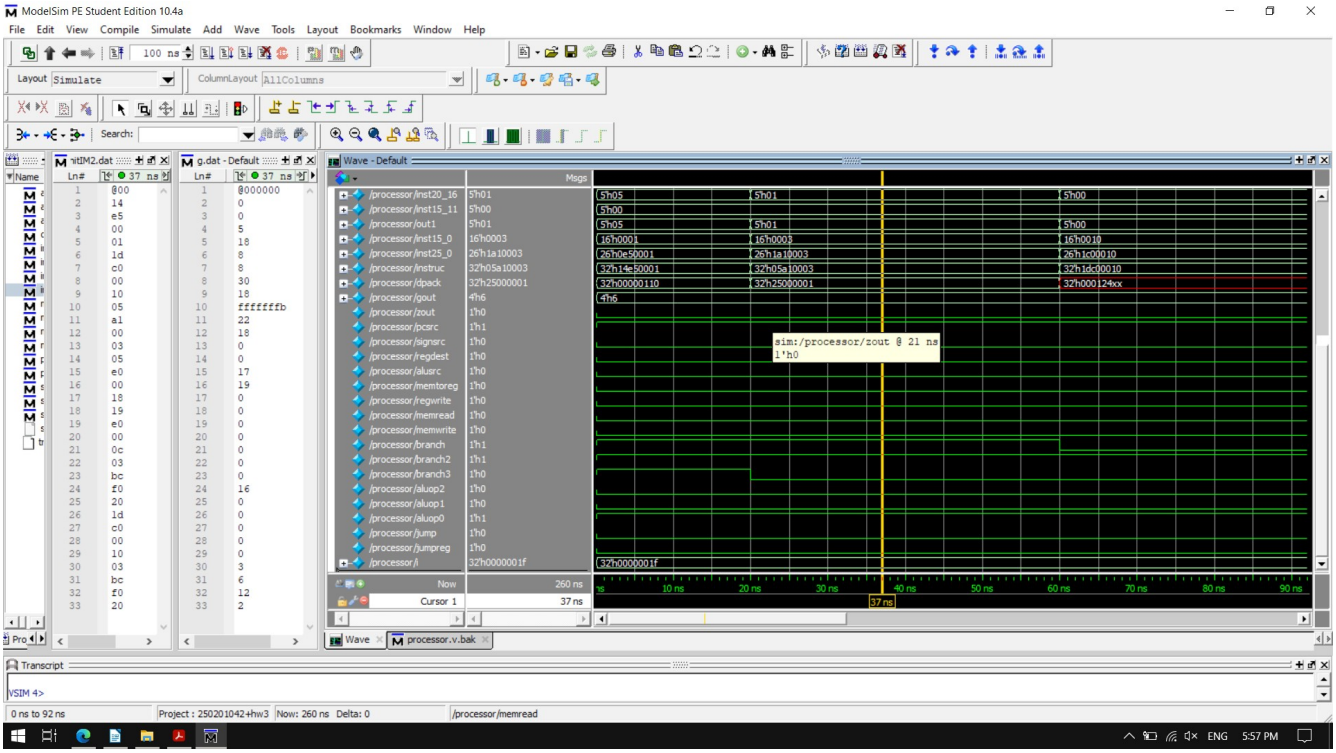
Control bits : branch=1 branch2=1 branch3= 1, so branch is 111 which corresponds BNE.
aluop2=0,aluop1=0,aluop0=1 so aluop is 001 which corresponds if((~aluop2) & (~aluop1) & aluop0)
gout=4'b0110 in alucont.v. gout is defined as 0110 which corresponds subtraction operation in alu32.v

Please look at next page for details

As you can see below, instruction which is 14E50001 is executed. $r7 is 18 in hex which is defined as dataa. $r5 is 8 in hex which is defined as datab. Our extad is 1 and shifted value sextad is 4. And gout says to alucontrol that it is an sub operation. Dataa - datab is defined in sum which is 10 in hex. Sum is not zero and they are not equal, zout is defined as 0. So pcsrc returns 1 in that situation and it jumps to [adder1out(pc +4) + sextad] = adder2out. (please look at pcsrc situations in 4th pages)



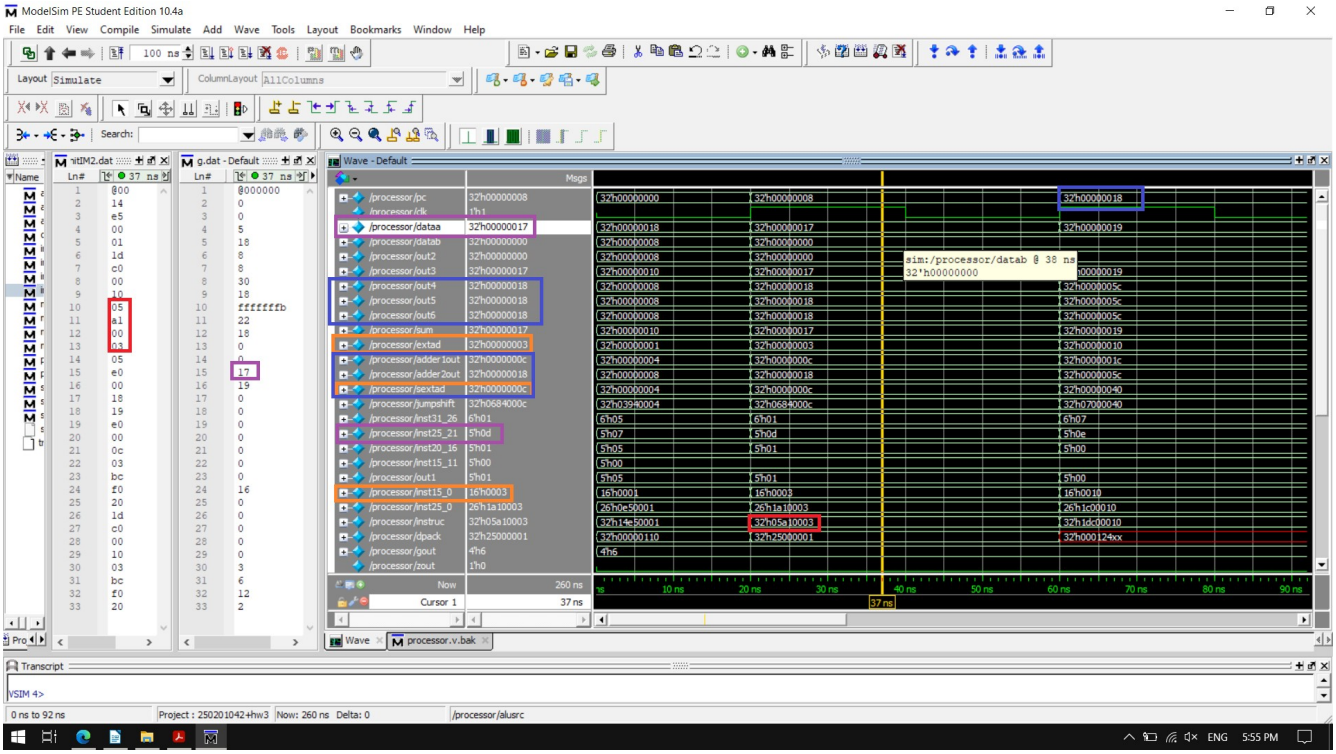
7- Bgez : IM2.dat
000001 01101 00001 0000000000000003 | in hex = 05A10003
bgez $r13 3

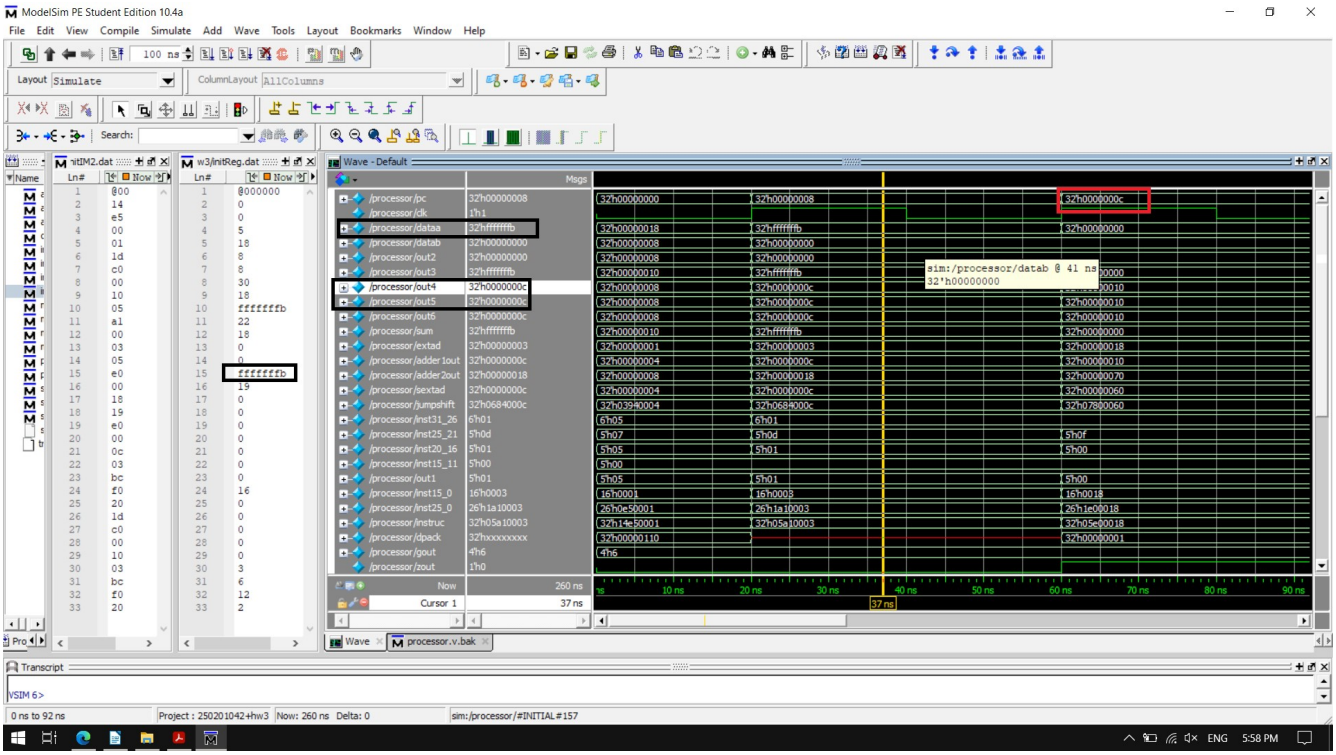


Control bits : branch=1 branch2=1 branch3= 0, so branch is 110 which corresponds BGEZ.
aluop2=0,aluop1=0,aluop0=1 so aluop is 001 which corresponds if((~aluop2) & (~aluop1) & aluop0)
gout=4'b0110 in alucont.v. gout is defined as 0110 which corresponds subtraction operation in alu32.v

Please look at next page for details

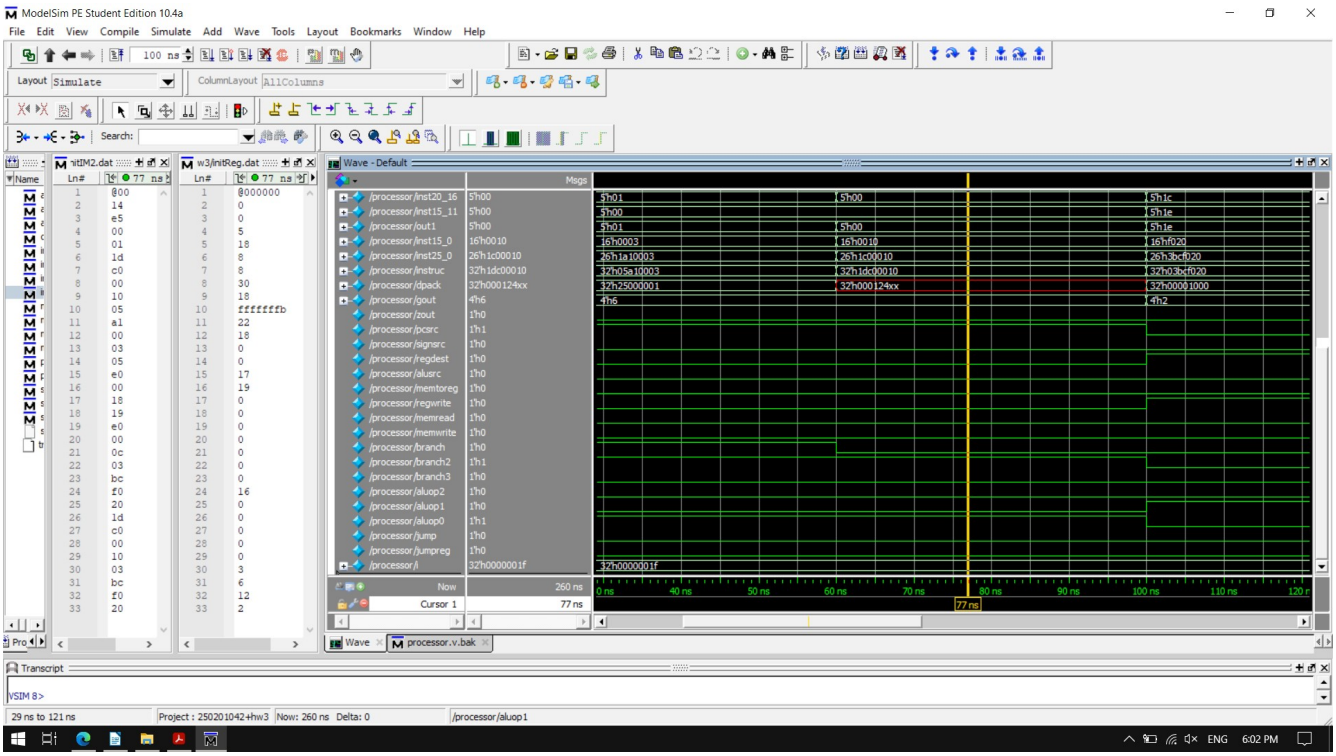
As you can see below, instruction which is 05A10003 is executed. $r13 is 17 in hex which is defined as dataa. $r1 is 0 in hex which is defined as datab. And gout says to alucontrol that it is an sub operation. Dataa - datab is defined in sum which is 17 in hex. Sum is not zero, zout is defined as 0. And sign of sum is positive, So pcsrc returns 1 in that situation and it jumps to [adder1out(pc +4) + sextad] = adder2out. (please look at pcsrc situations in 4th pages). For $r13=0 → zout is 1 and signsrc is 0. That situation also makes pcsrc 1 and it jumps.





For negative value, it is not jumping and makes pc+4 because of that signsrc is 1 and it makes pcsrc be zero. Just positive values and zero value can make it jump.

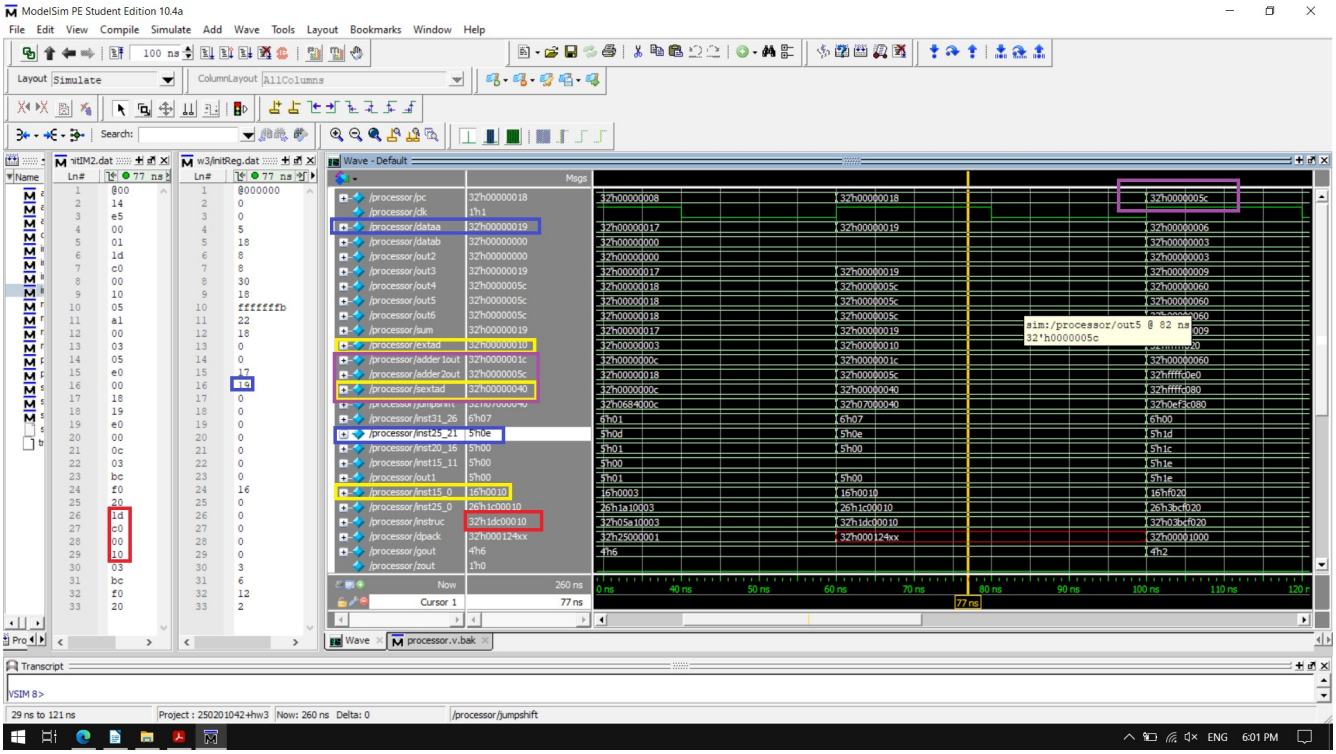
8- Bgtz : IM2.dat
000111 01110 00000 0000000000010000 | in hex = 1DC00010
bgtz $r14 16

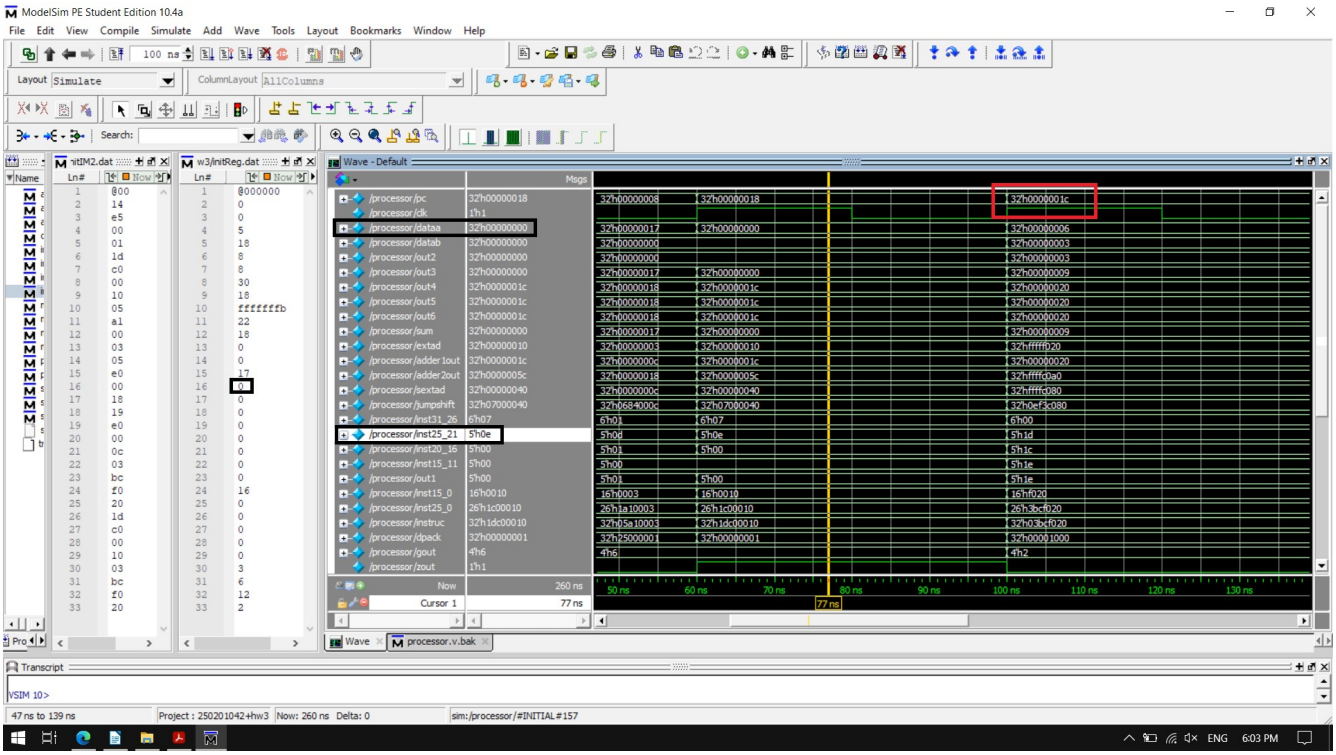


Control bits : branch=0 branch2=1 branch3= 0, so branch is 010 which corresponds BGTZ.
aluop2=0,aluop1=0,aluop0=1 so aluop is 001 which corresponds if((~aluop2) & (~aluop1) & aluop0)
gout=4'b0110 in alucont.v. gout is defined as 0110 which corresponds subtraction operation in alu32.v

Please look at next page for details

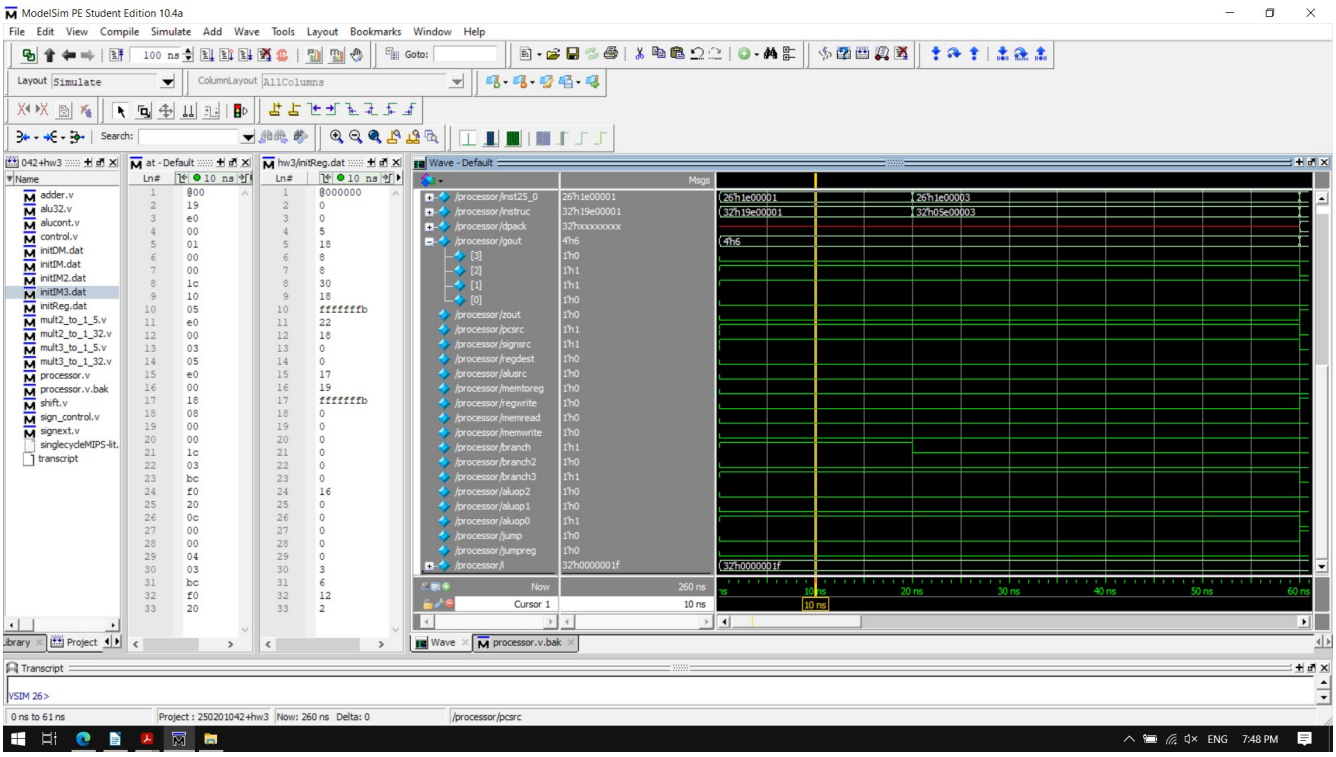
As you can see below, instruction which is 1DC00010 is executed. $r14 is 19 in hex which is defined as dataa. $r0 is 0 in hex which is defined as datab. And gout says to alucontrol that it is an sub operation. Dataa - datab is defined in sum which is 19 in hex. Sum is not zero, so zout is defined as 0. And sign of sum is positive, so signsrc returns 0. So pcsrc returns 1 in that situation and it jumps to [adder1out(pc +4) + sextad] = adder2out. (please look at pcsrc situations in 4th pages).





For negative value (signsrc is 1) and zero value (zout is 1) and it makes pcsrc be zero. Just positive values can make it jump. For example $r14 = 0 and it just goes pc+4

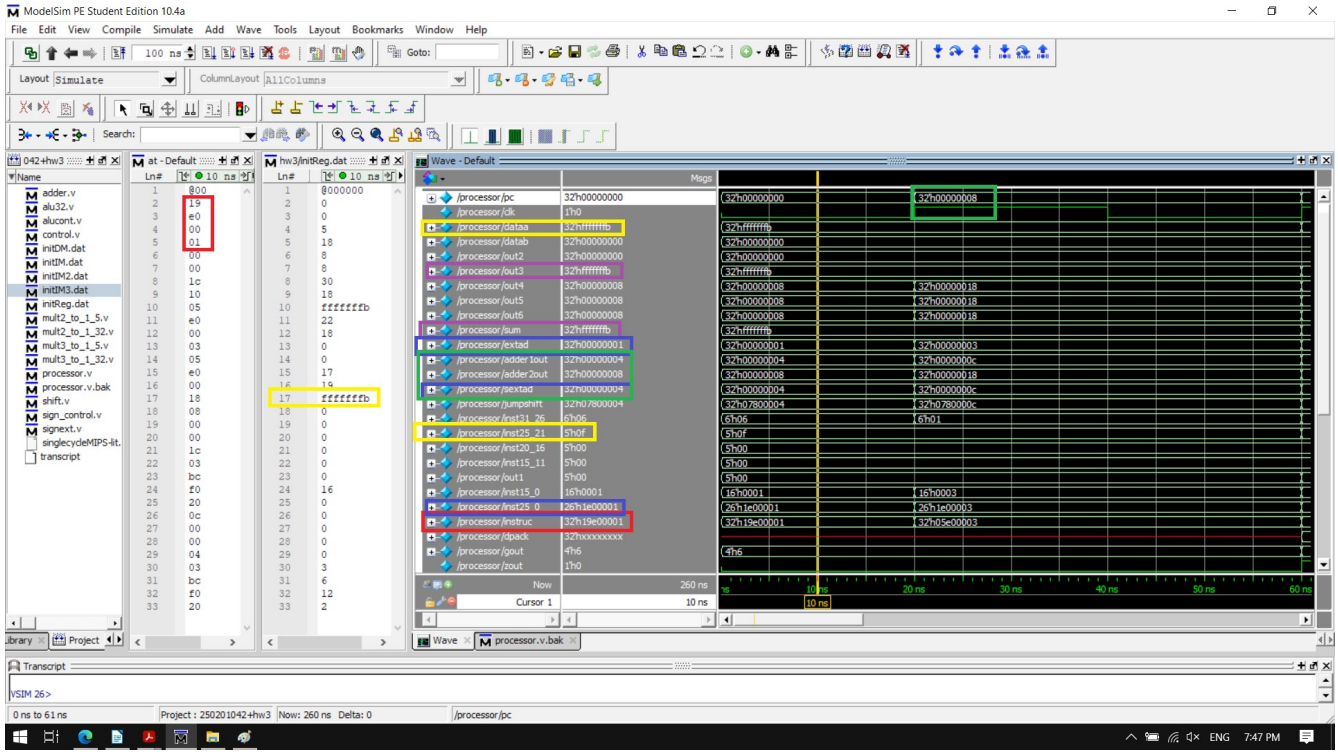
9- BLEZ : IM3.dat
000110 01111 00000 0000000000000001 | in hex 19E00001
blez $r15 1

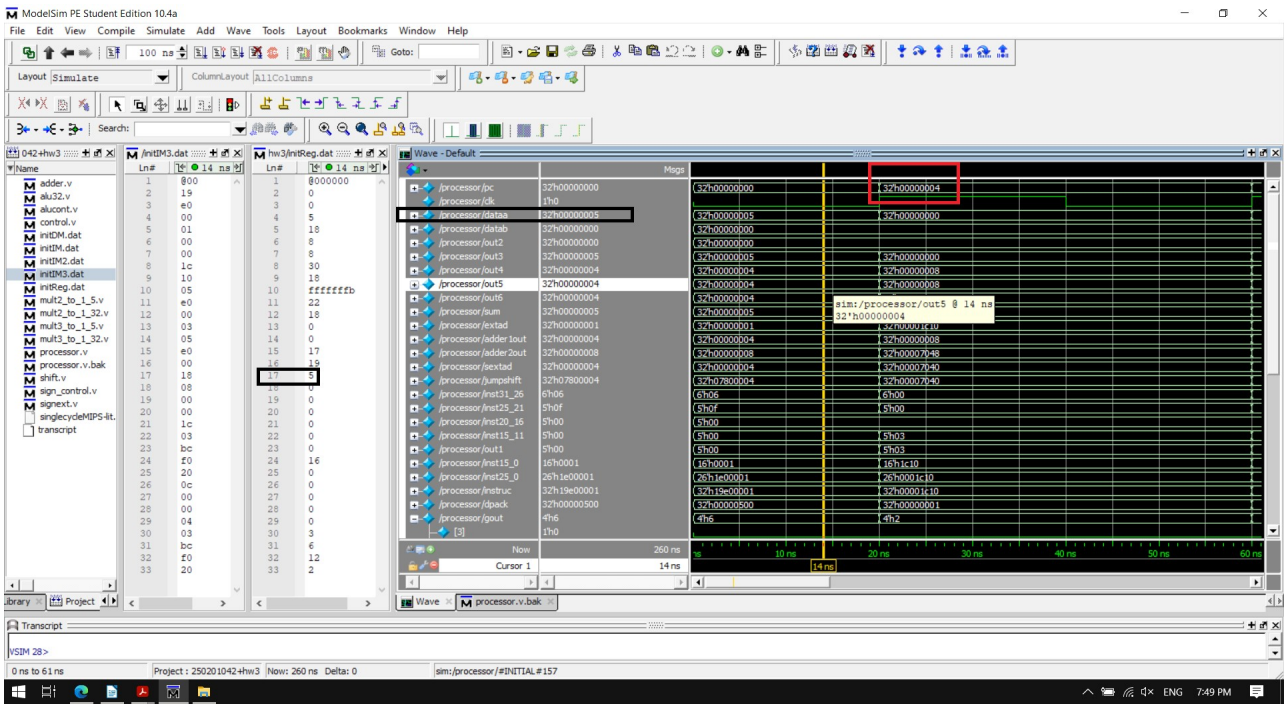


Control bits : branch=1 branch2=0 branch3= 1, so branch is 101 which corresponds BLEZ.
aluop2=0,aluop1=0,aluop0=1 so aluop is 001 which corresponds if((~aluop2) & (~aluop1) & aluop0)
gout=4'b0110 in alucont.v. gout is defined as 0110 which corresponds subtraction operation in alu32.v

Please look at next page for details

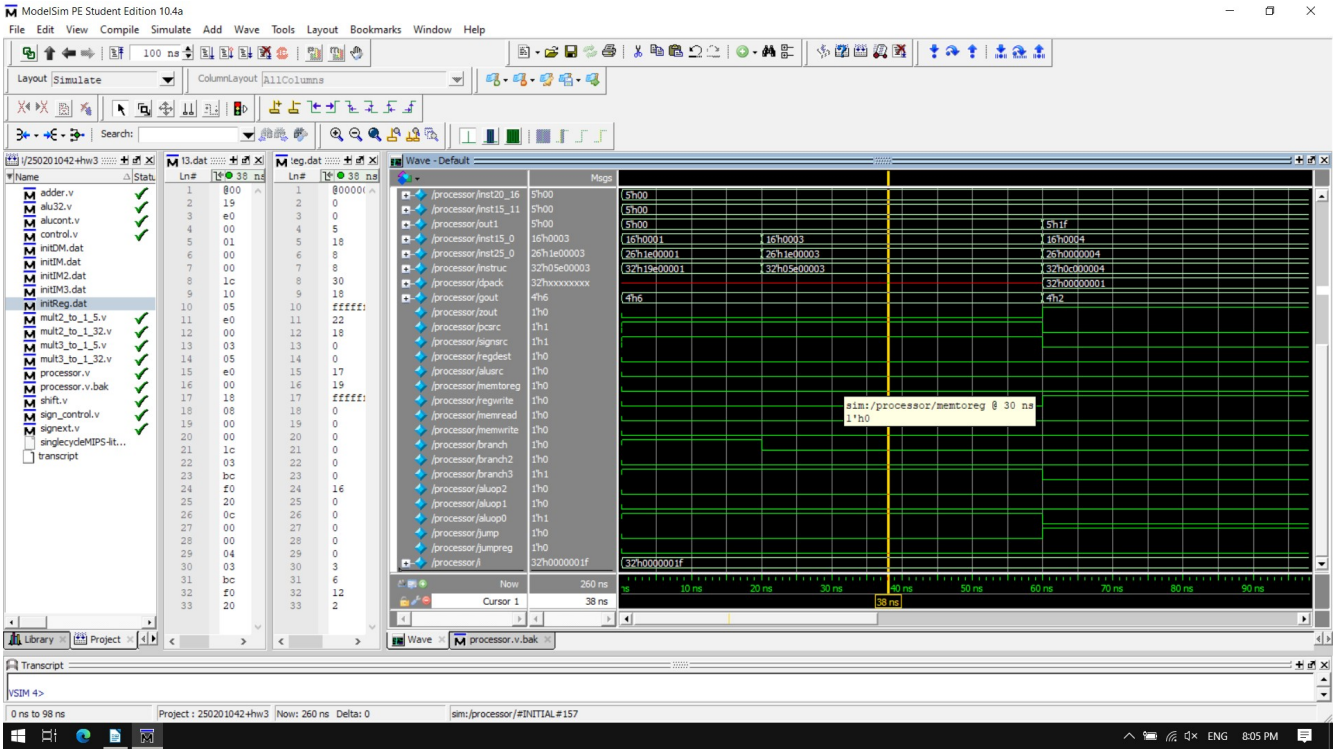
As you can see below, instruction which is 19E00001 is executed. $r15 is fffffffb(-5) in hex which is defined as dataa. $r0 is 0 in hex which is defined as datab. And gout says to alucontrol that it is an sub operation. Dataa - datab is defined in sum which is fffffffb(-5) in hex. Sum is not zero, so zout is defined as 0. And sign of sum is negative, so signsrc returns 1. So pcsrc returns 1 in that situation and it jumps to [adder1out(pc +4) + sextad] = adder2out. (please look at pcsrc situations in 4th pages). For $r15=0 → zout is 1 and signsrc is 1. That situation also makes pcsrc 1 and it jumps.





For positive value (signsrc is 0) it makes pcsrc be zero. Just negative and zero values can make it jump. For example $r14 = 5 and it just goes pc+4

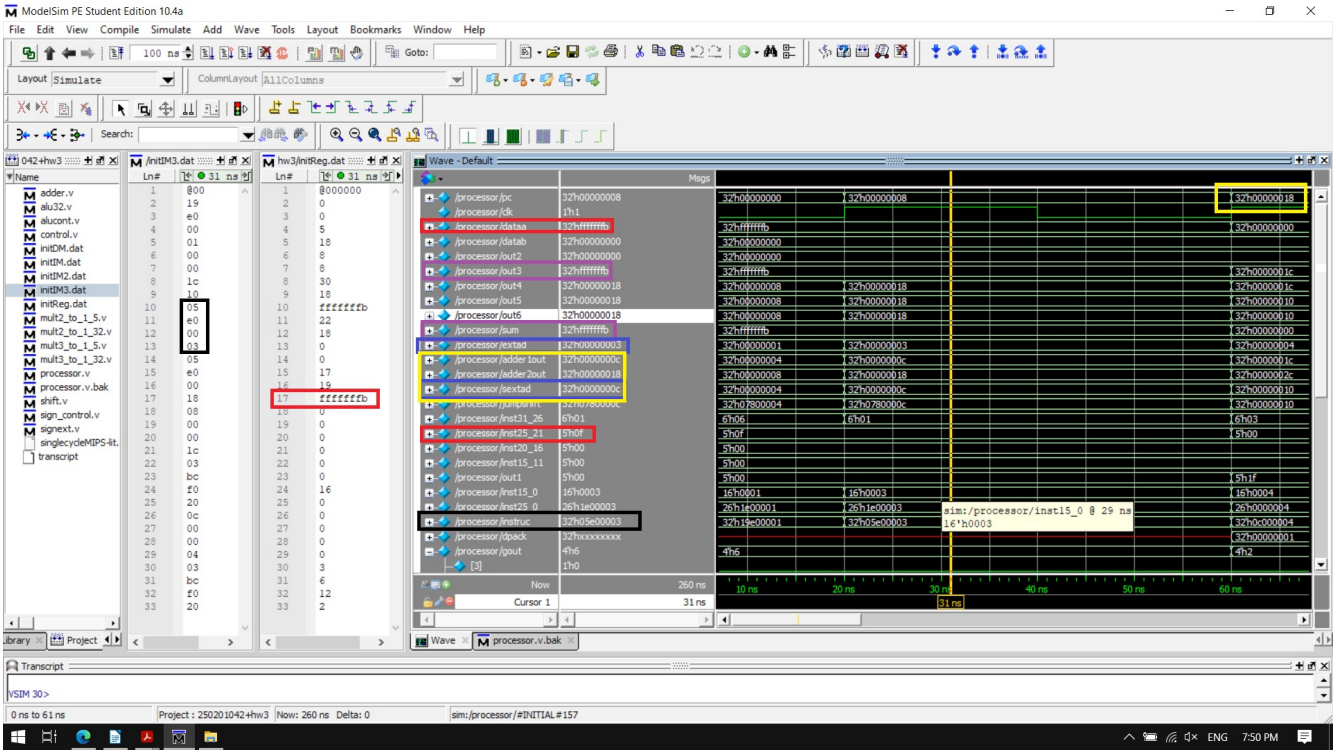
10- BLTZ : IM3.dat
000001 01111 00000 0000000000000011 | in hex 05E00003
bltz $r15 3

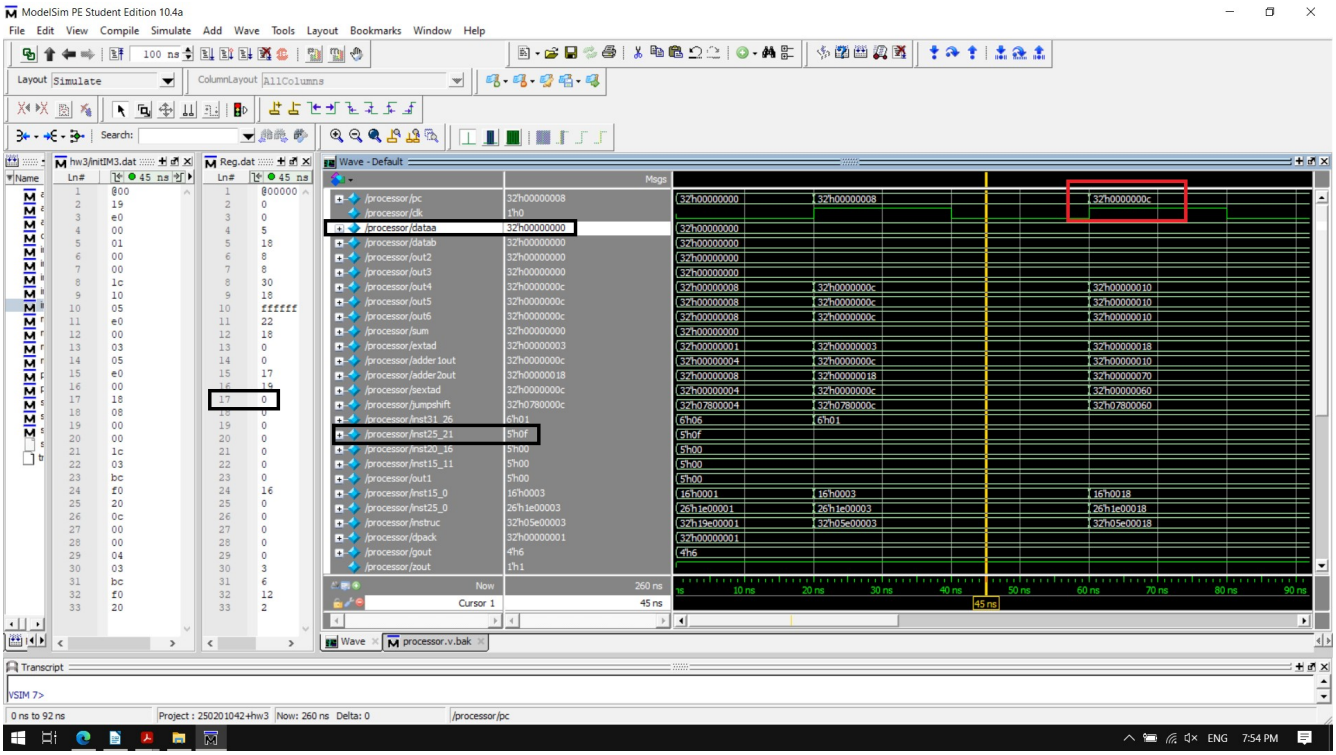


Control bits : branch=0 branch2=0 branch3= 1, so branch is 001 which corresponds BLTZ.
aluop2=0,aluop1=0,aluop0=1 so aluop is 001 which corresponds if((~aluop2) & (~aluop1) & aluop0)
gout=4'b0110 in alucont.v. gout is defined as 0110 which corresponds subtraction operation in alu32.v

Please look at next page for details

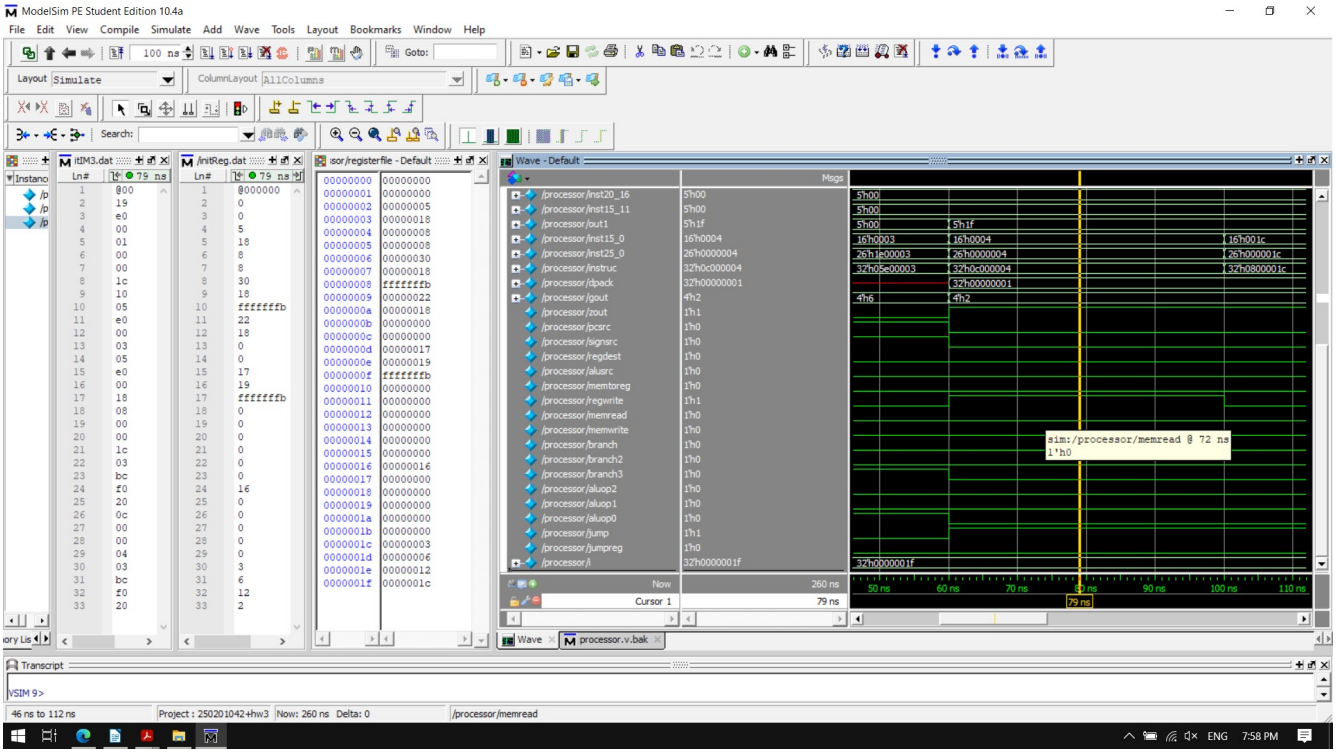
As you can see below, instruction which is 05E00003 is executed. $r15 is fffffffb(-5) in hex which is defined as dataa. $r0 is 0 in hex which is defined as datab. And gout says to alucontrol that it is an sub operation. Dataa - datab is defined in sum which is fffffffb(-5) in hex. Sum is not zero, so zout is defined as 0. And sign of sum is negative, so signsrc returns 1. So pcsrc returns 1 in that situation and it jumps to [adder1out(pc +4) + sextad] = adder2out. (please look at pcsrc situations in 4th pages).





For positive value (signsrc is 0) and zero value (zout is 1) and it makes pcsrc be zero. Just negative values can make it jump. For example $r15 = 0 and it just goes pc+4.

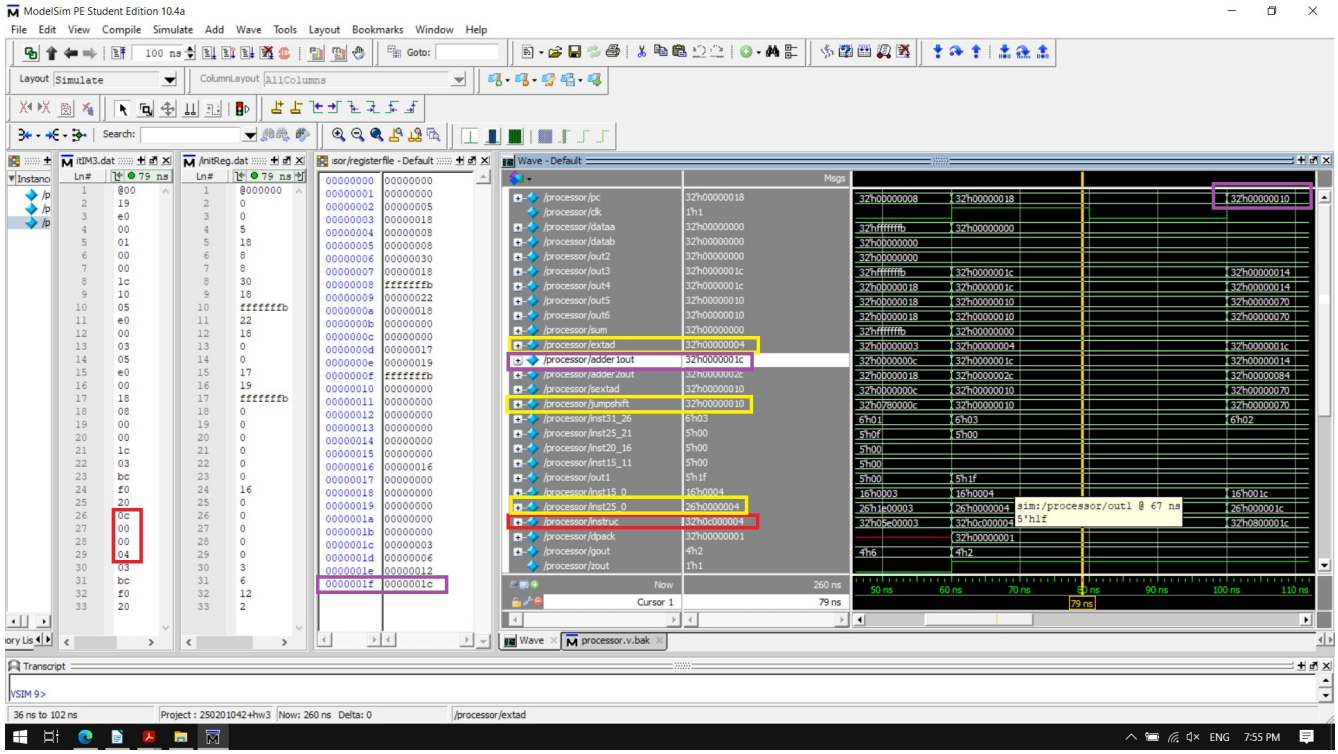
11- JAL : IM3.dat
000011 00000000000000000000000100 | in hex 0C000004
jal 4



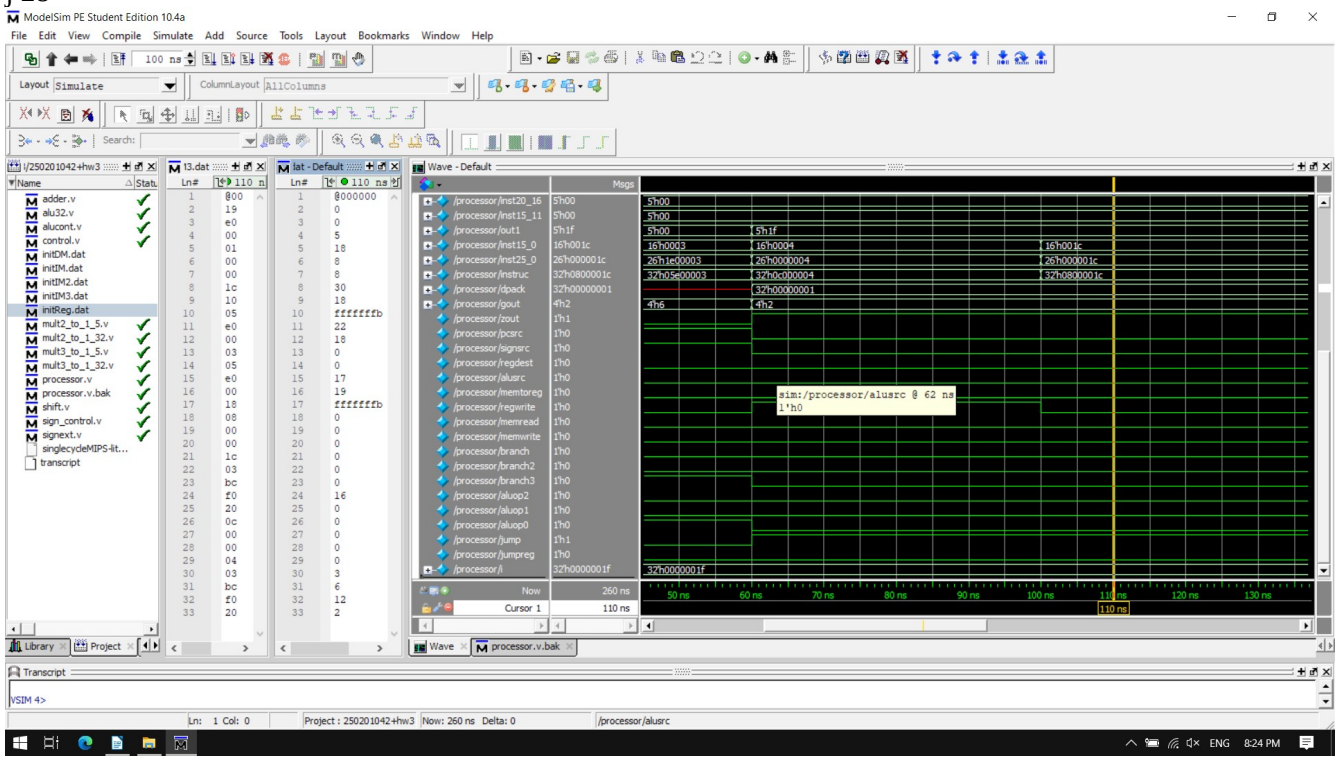
Control bits : jump is 1 and regwrite is 1 (to write pc+4 value to $r31). Memtoreg is zero so mux3 selects i2 as output which holds pc+4 value. Regdest is zero so mux1 select $r31 as write register. Also mux5 selects jumpshift value.

Please look at next page for details

As you can see below, instruction which is 0C000004 is executed. Immediate value is 4 in hex and it is shifted. Shifted value is jumpshift which is 10 in hex. Jump control bit is 1 so mux5 selects jumpshift as output. As you can see, it jumps to 10 (16 in decimal).
jump is 1 and memtoreg is zero so mux3 selects i2 as output which holds pc+4 value (18 in hex = 24 in decimal. 24+4=28 in decimal = 1c in hex)
Regdest is zero and jump is 1 so mux1 select $r31 as write register and pc+4 (1c) value is written to the $r31



12- J : IM3.dat
000010 00000000000000000000011100 | in hex 0800001C
j 28



Control bits : jump is 1. So Mux5 selects jumpshift value as output.

Please look at next page for details

As you can see below, instruction which is 0800001C is executed. Immediate value is 1c in hex and it is shifted. Shifted value is jumpshift which is 70 in hex. Jump control bit is 1 so mux5 selects jumpshift as output. As you can see, it jumps to 70.