CSE3038 COMPUTER ORGANIZATION PROJECT REPORT#2

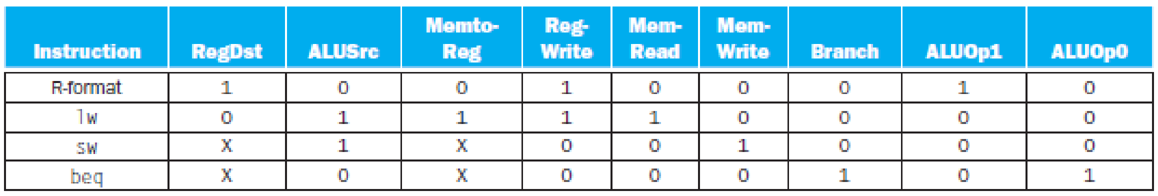
Prepared by

Serkan Aydın – 150114053

Deniz Arda Gürzihin – 150116065

Instruction Tableau

Before, we have started implementation, the instruction control signal was like this.

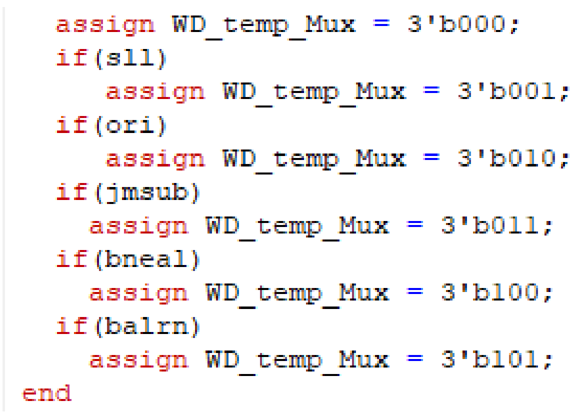


Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Components Which Has Been Changed Or Added

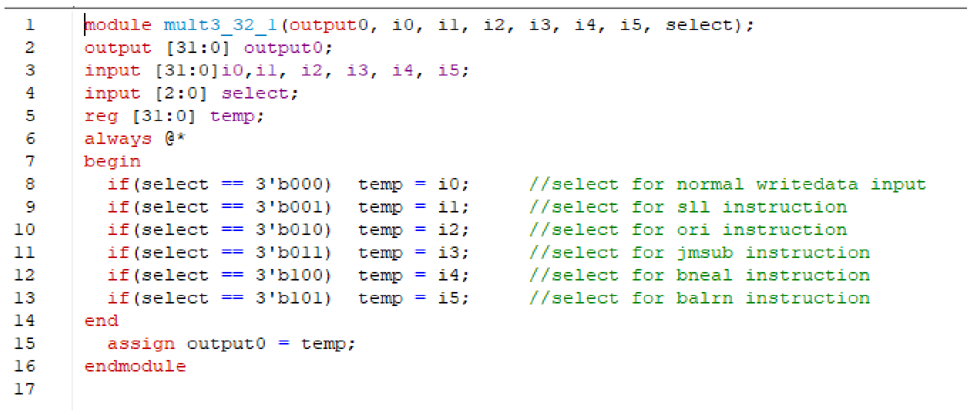
1) Write Data Multiplixer

After implementing the new instructions, there was a lot of instruction which wanted to write data to the registers, so that we have added a new multiplixer(3x8 and 32bits) before entering the write data port of the register files. The select bits of this multiplixer calculated with the current instruction signals. And can be shown like this;



```
   assign WD_temp_Mux = 3'b000;
   if(sll)
      assign WD_temp_Mux = 3'b001;
   if(ori)
      assign WD_temp_Mux = 3'b010;
   if(jmsub)
     assign WD_temp_Mux = 3'b011;
   if(bneal)
     assign WD_temp_Mux = 3'b100;
   if(balrn)
     assign WD_temp_Mux = 3'b101;
end
```

If the instruction is the lw or rformat, control signal assigned as 3’b000, and for the other instructions, it can be seen in the picture how signal bit has assigned. Then, this select bit has an input to a new module called as mult3_32_1, this module was taking input from the lw, rformat, sll, ori, jmsub, bneal and balrn instructions, and also the select bits, and choosing the output corresponding to the select bits, following code is snipping the mult3_32_1 module.



```
module mult3_32_1(output0, i0, i1, i2, i3, i4, i5, select);
output [31:0] output0;
input [31:0]i0,i1, i2, i3, i4, i5;
input [2:0] select;
reg [31:0] temp;
always @*
begin
  if(select == 3'b000)  temp = i0;     //select for normal writedata input
  if(select == 3'b001)  temp = i1;     //select for sll instruction
  if(select == 3'b010)  temp = i2;     //select for ori instruction
  if(select == 3'b011)  temp = i3;     //select for jmsub instruction
  if(select == 3'b100)  temp = i4;     //select for bneal instruction
  if(select == 3'b101)  temp = i5;     //select for balrn instruction
end
  assign output0 = temp;
endmodule
```

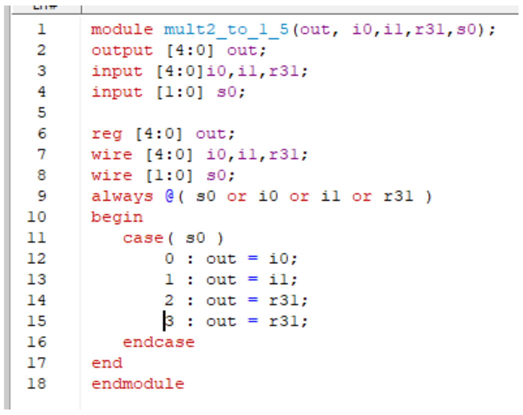
2) *The Multiplixer Controlled With RegDst Signal*

First of all we have changed the control signal which name is RegDst, we have made it two bits and calculated as

“assign regdest={linkadress,rformat};”

Linkadress in here refers to the instructions which were using register31 for writing, rformat is the control signal of the instruction if it rformat or not.

Then this control signal has been assigned as the control signal of the changed multiplixer module which name is “mult2_to_1_5”. With this module, we have controlled the register which will be modified according to the RegWrite signal. Here is the verilog code;



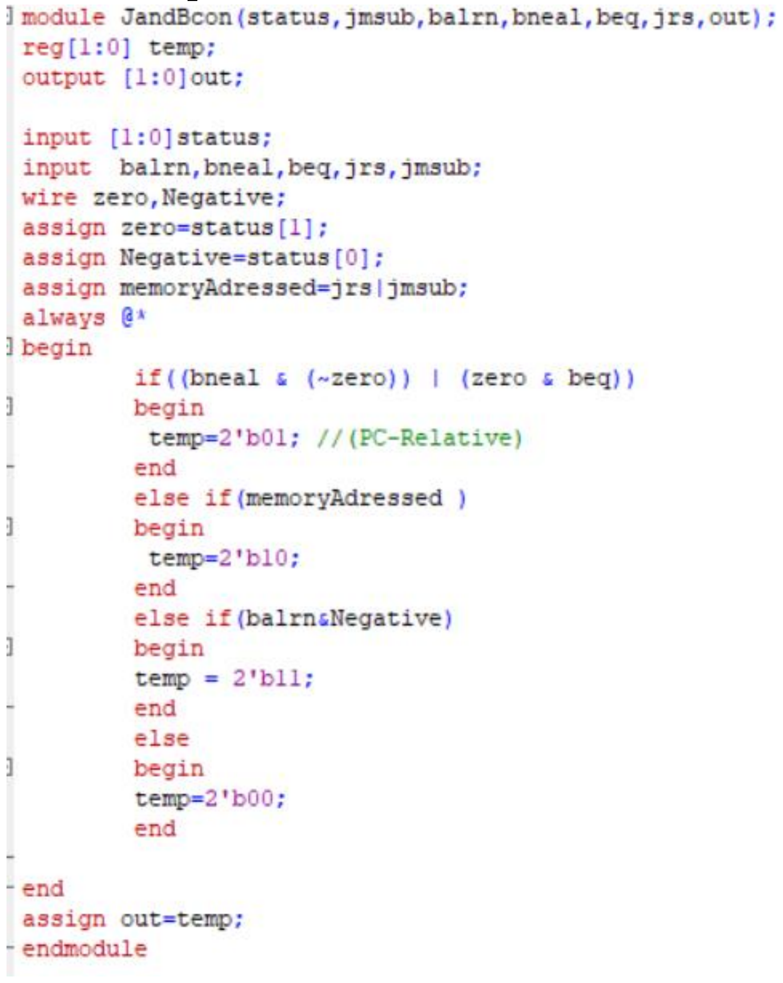
```
module mult2_to_1_5(out, i0,i1,r31,s0);
output [4:0] out;
input [4:0]i0,i1,r31;
input [1:0] s0;

reg [4:0] out;
wire [4:0] i0,i1,r31;
wire [1:0] s0;
always @( s0 or i0 or i1 or r31 )
begin
    case( s0 )
        0 : out = i0;
        1 : out = i1;
        2 : out = r31;
        3 : out = r31;
    endcase
end
endmodule
```

In here, i0 is coming from the instruction[20:16], i1 is coming from the instruction[15:11] and r31 is the oppcode of the 31th register of the register files.

Instruction	***Link Adress***	***Rformat***	***RegDest***
Rformat	*0*	*1*	*01*
Lw	*0*	*0*	*00*
Sll	*0*	*1*	*01*

Ori	*0*	*0*	*00*
Jmsub	*1*	*1*	*11*
Bneal	*1*	*0*	*10*
Balrn	*1*	*1*	*11*

3) JandB Component



```
module JandBcon(status,jmsub,balrn,bneal,beq,jrs,out);
reg[1:0] temp;
output [1:0]out;

input [1:0]status;
input  balrn,bneal,beq,jrs,jmsub;
wire zero,Negative;
assign zero=status[1];
assign Negative=status[0];
assign memoryAdressed=jrs|jmsub;
always @*
begin
        if((bneal & (~zero)) | (zero & beq))
        begin
         temp=2'b01; //(PC-Relative)
        end
        else if(memoryAdressed )
        begin
         temp=2'b10;
        end
        else if(balrn&Negative)
        begin
        temp = 2'b11;
        end
        else
        begin
        temp=2'b00;
        end

end
assign out=temp;
endmodule
```

This component decides pcsrc value which decides next pc value.

If current is instruction is bneal and ~zero output new pc value will be as pc relative addressing mode as beq(pcsrc will be 01 and output of mult4 will be adder2out)

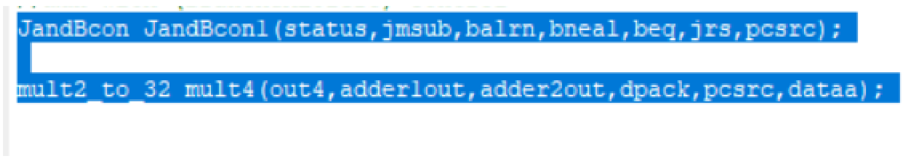
Else If current instruction jrs or jmsub new pc value will be as base addressing mode(pcsrc will be 10 and output of mult4 dpack)

Else if current instruction balrn and negative output is 1 new pc value will be as register addressed mode(pcsrc will be 11 and output of mult4 value of register $rs)

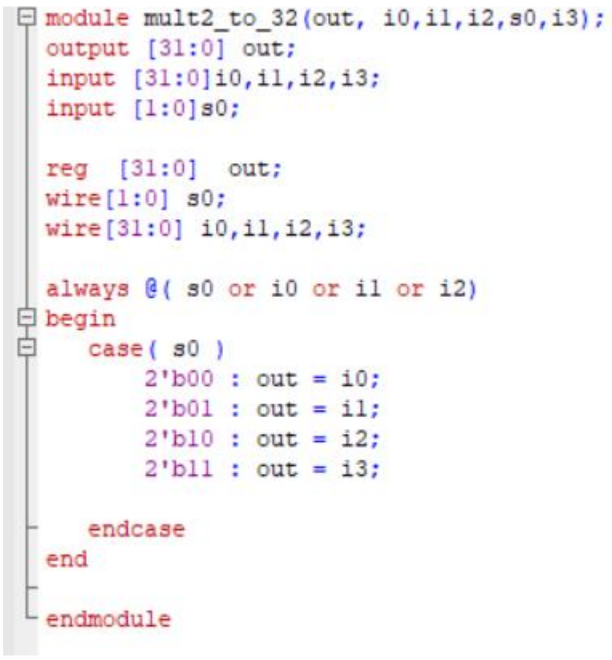
Else pc will be pc+4

pcsrc	New address
00	PC← PC+4
01	PC ← PC+4+(Label << 2)

10	PC ← M[$rs-$rt](for jrs $rt is $0)
11	PC ← R[rs]

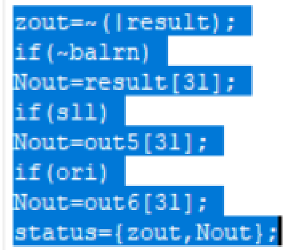


```
JandBcon JandBcon1(status,jmsub,balrn,bneal,beq,jrs,pcsrc);

mult2_to_32 mult4(out4,adder1out,adder2out,dpack,pcsrc,dataa);
```



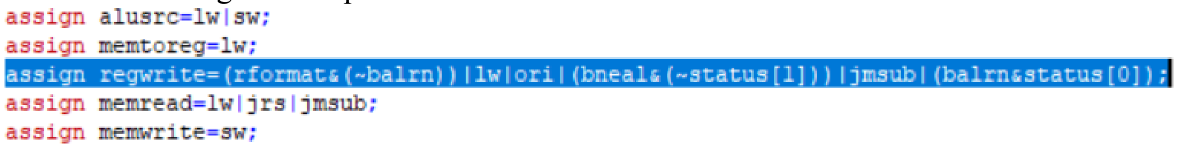
```
module mult2_to_32(out, i0,i1,i2,s0,i3);
output [31:0] out;
input [31:0]i0,i1,i2,i3;
input [1:0]s0;

reg  [31:0]  out;
wire[1:0] s0;
wire[31:0] i0,i1,i2,i3;

always @( s0 or i0 or i1 or i2)
begin
   case( s0 )
       2'b00 : out = i0;
       2'b01 : out = i1;
       2'b10 : out = i2;
       2'b11 : out = i3;

   endcase
end

endmodule
```

4) *Status Register On Alu*



```
zout=~(|result);
if(~balrn)
Nout=result[31];
if(sll)
Nout=out5[31];
if(ori)
Nout=out6[31];
status={zout,Nout};
```

Zero output is set as default but status[0] holds the zero output value at ALU. Negative output is set by result of ALU,sll instruction's output and ori instructions output. If current instruction is sll negative output value will be equal to out5[31] else if current instruction is ori negative output value will be equal to as out6[31] else if current instruction is not balrn negative output will be equal to ALU's result[31]. Balrn instruction doesn't change negative output, balrn instruction uses previous instrunction's negative output.



```
assign alusrc=lw|sw;
assign memtoreg=lw;
assign regwrite=(rformat&(~balrn))|lw|ori|(bneal&(~status[1]))|jmsub|(balrn&status[0]);
assign memread=lw|jrs|jmsub;
assign memwrite=sw;
```

Status register is used also to decide what will regwrite value. If zero output is zero and current instruction is bneal regwrite will be 1 to write link adress to register 31.

If current instruction balrn and negative output is 1 regwrite will be 1 to write link adress too.

5) Control Unit

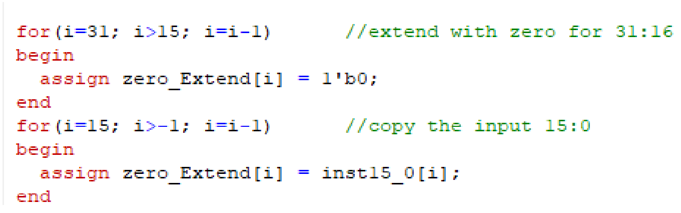
To control unit, we have added more inputs and outputs in order to control the flow of the proccesors. The input signals that have added is the “in2” which refers to the current instruction[5:0] to find the function code of the R format instruction. And we have added the following output signal which are one bit and helping for the jumping, branching and writing to the registers. These new output signals are “bneal”, “balrn”, “jrs”, “ori”, “jmsub”, “sll”, “beq” and “WD_Mux_Signal”. “WD_Mux_Signal” is 3 bit signal and selector bit of the write data multiplixer. One bit signals are becoming true if the current instruction is that instruction. And WD_Mux_Signal has the following truth table. Final instruction table can be seen on the last page.

Instruction	***WD_Mux_Signal***
R format(basic)	*000*
Lw	*000*
Sw	*000*
Sll	*001*
Ori	*010*
Jmsub	*011*
Bneal	*100*
Balrn	*101*
Jrs	*000*

NEW INSTRUCTIONS IMPLEMENTATION

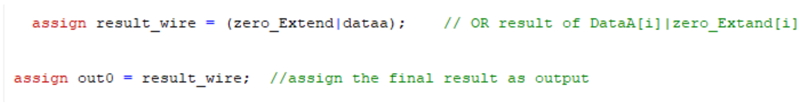
1) ORI

For ori instruction, we have designed a new component, we have not used the ALU component for this calculation. The component, we have designed has taken two inputs which are the “dataa”, “inst15_0” and given “out0” as output. In here, “dataa” input was the data from the register which given in the instruction[25:21], “inst15_0” was the instruction[15:0] which is an immediate value. For calculating the “out0”, firstly we have zero extend the “inst15_0” via following code



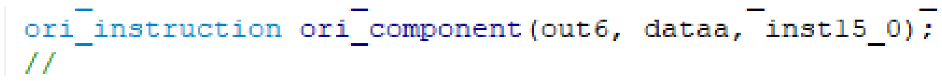
```
for(i=31; i>15; i=i-1)        //extend with zero for 31:16
begin
  assign zero_Extend[i] = 1'b0;
end
for(i=15; i>-1; i=i-1)        //copy the input 15:0
begin
  assign zero_Extend[i] = inst15_0[i];
end
```

Then we have simply or the zero extended inst15_0 with “dataa” input. And assign it as the “out0”.



```
  assign result_wire = (zero_Extend|dataa);    // OR result of DataA[i]|zero_Extand[i]

assign out0 = result_wire;  //assign the final result as output
```

And we have implement this component on the processor with defining this module.



```
ori_instruction ori_component(out6, dataa, inst15_0);
//
```

Additionally, the instruction control signal was like this;

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	AluOp1	AluOp2
Ori	00	0	0	1	0	0	0	0	0

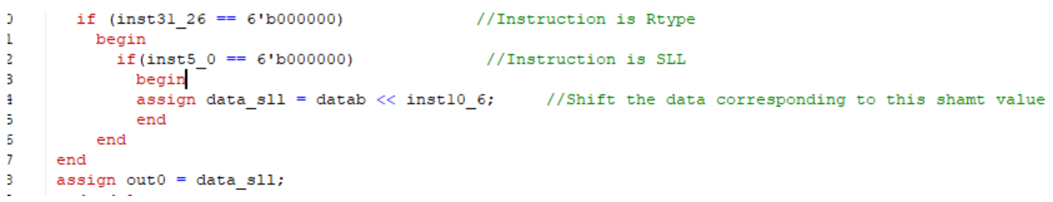
Then the output of this component has been connected with the multiplixer in front of the write data port of the register files which was designed by us and written in to the register defined in the inst[20:16].

2) SLL

The control signal of the SLL instruction has determined like this;

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	AluOp1	AluOp2
Sll	01	0	0	1	0	0	0	1	0

For sll instruction, we have also designed a new component so we have not used the ALU component for calculation. The component, we have designed was taken 4 input which name was “inst31_26”, “inst5_0”, datab, “inst10_6” respectively and giving one output which name was “out0”. “Inst31_26” was referring to the instruction[31:26], “inst5_0” was referring to the instruction[5:0], “datab” was referring to the data coming from the Inst[20:16] registers and finally “inst10_6” was referring to the instruction[10:6]. For calculating the “out0”, first we have check the instruction is Rformat or not then we have checked if it is sll function code which is “000000”. If this conditions was meet up, then we have found “out0” as the datab shifted left by the inst10_6 times. The following code is snipping to that part.



```
  if (inst31_26 == 6'b000000)              //Instruction is Rtype
    begin
      if(inst5_0 == 6'b000000)              //Instruction is SLL
        begin
        assign data_sll = datab << inst10_6;     //Shift the data corresponding to this shamt value
        end
    end
end
assign out0 = data_sll;
```

Then the output of this component has been connected with the multiplixer in front of the write data port of the register files which was designed by us and written in to the register defined in the inst[15:11].

3) JMSUB

The control signal of the JMSUB instruction has determined like this;

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	AluOp1	AluOp2
Jmsub	11	0	0	1	1	0	0	1	0

The design of the jmsub instruction was pretty simple, we have just send the memory output of the processor to a multiplixer which was controlled by “pcsrc” select bits, and in this condition it was 2’b10. Which is calculated by the JandBcon modüle.

Simply jmsub was, taking the two registers differences which was “dataa-datab” and sending it to the memory and finding result was “dpack”. And “dpack” is sending to the multiplixers as input.

4) BNEAL (I-format)

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	AluOp1	AluOp2
bneal	10	0	0	x	1	0	0	0	1

I-type opcode=45 bneal $rs, $rt, Target

Bneal instruction works as bne instruction but it stores link adress in register 31.The instruction looks zero output if zero output(status[1]) equals 0 branches else not branches.RegWrite equals If bneal signal equals to 1 and zero output equals to 0 pcsrc is set by 01 and regwrite is set by 1.So the next PC is calculated as beq instruction.

5) JRS(I-format)

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	AluOp1	AluOp2
jrs	10	0	0	1	1	0	0	0	0

I-type opcode=18
jrs $rs jumps to address found in memory where the memory address is written in register $rs.
It stores link adress in register 31.pcsrc is set as 10. If pcsrc equal to 10 new pc value is set as memory output.

6) BALRN (R-format)

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	AluOp1	AluOp2
balrn	11	0	0	x	0	0	0	1	0

R-type funct=23 balrn $rs, $rd if Status [N] = 1, branches to address found in register $rs link address is stored in $rd (which defaults to 31)

Negative output is set by previous instruction. If previous instruction’s result in ALU is negative negative output is set by 1 else 0.

If control signal “balrn” equals to 1 and negative output(status[0]) equals to 1 pcsrc is set by 11 and regwrite is set by 1 then branches to address found in register $rs.

FINAL TABLE OF THE PROCESSORS

Instruction	***Opp Code***	***RegDst***	***AluSrc***	***MemToReg***	***RegWrite***	***MemRead***	***MemWrite***	***Branch***
R Format	*000000*	*01*	*0*	*0*	*1*	*0*	*0*	*0*
Lw	*100011*	*00*	*1*	*1*	*1*	*1*	*0*	*0*
Sw	*101011*	*00*	*1*	*0*	*0*	*0*	*1*	*0*
Beq	*000100*	*00*	*0*	*0*	*0*	*0*	*0*	*1*
Sll	*000000*	*01*	*0*	*0*	*1*	*0*	*0*	*0*
Ori	*001101*	*00*	*0*	*0*	*1*	*0*	*0*	*0*
Jmsub	*000000*	*11*	*0*	*0*	*1*	*1*	*0*	*0*
Bneal	*101101*	*10*	*0*	*0*	*1 if(Z=0) 0 if(Z=1)*	*0*	*0*	*0*
Balrn	*000000*	*11*	*0*	*0*	*1 if(N=1) 0 if(N=0)*	*0*	*0*	*0*
Jrs	*010010*	*00*	*0*	*0*	*0*	*1*	*0*	*0*

Instruction	AluOp1	AluOp2	Sll	Ori	Jmsub	Bneal	Balrn	Jrs	WD_Mux_Signal
R Format	1	0	0	0	0	0	0	0	000
Lw	0	0	0	0	0	0	0	0	000
Sw	0	0	0	0	0	0	0	0	000
Beq	0	1	0	0	0	0	0	0	000
Sll	1	0	1	0	0	0	0	0	001
Ori	0	0	0	1	0	0	0	0	010
Jmsub	1	0	0	0	1	0	0	0	011
Bneal	1	1	0	0	0	1	0	0	100
Balrn	1	0	0	0	0	0	1	0	101
Jrs	0	0	0	0	0	0	0	1	000