Mips instructions

BLEZ instruction

Branch if less than or equal to zero

Description:

- (BLEZ) is mips instruction that jump to(go to) specific label (determined address) if a given register (rs) is less than or equal to 0 (rs <=0)

Format:

- Operation code (op code) : 000110

- 000110 (6)   blez rs , label   ----→   branch if less than or equal to zero

Working principle

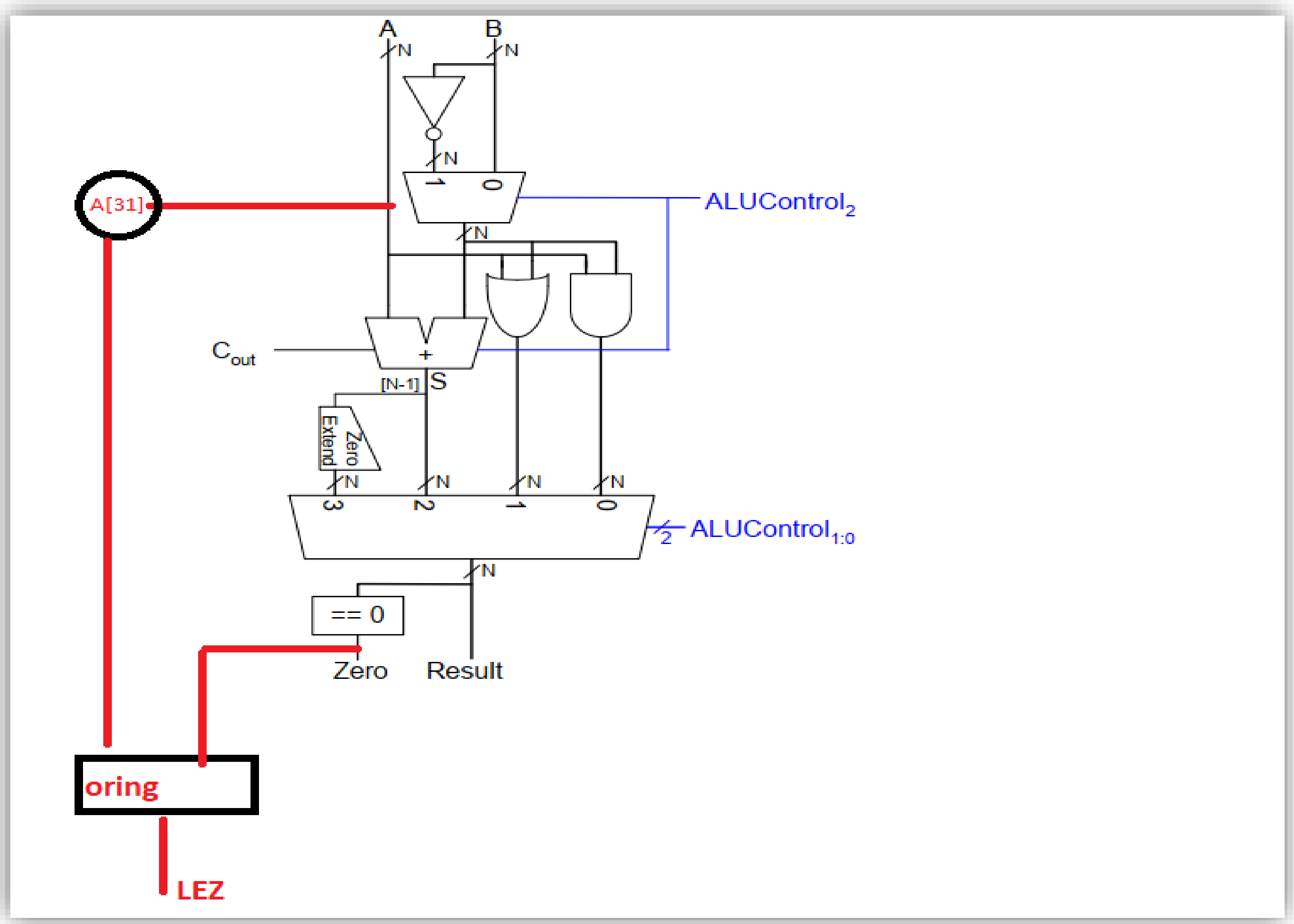
- This instruction take a register (operand one) and compare it to zero registrar (build in operand two)
- did this register less than or equal to 0 …..?
- the result of this comparison is true or false   (0 or 1)
- If true result going to a specific label (address) and execute the commands there
- If false result continue execute another commands (the commands out of condition outline)

Modifications:

- We modified 3 things:
- Alu modifying
- Datapath modifying
- Main decoder modifying

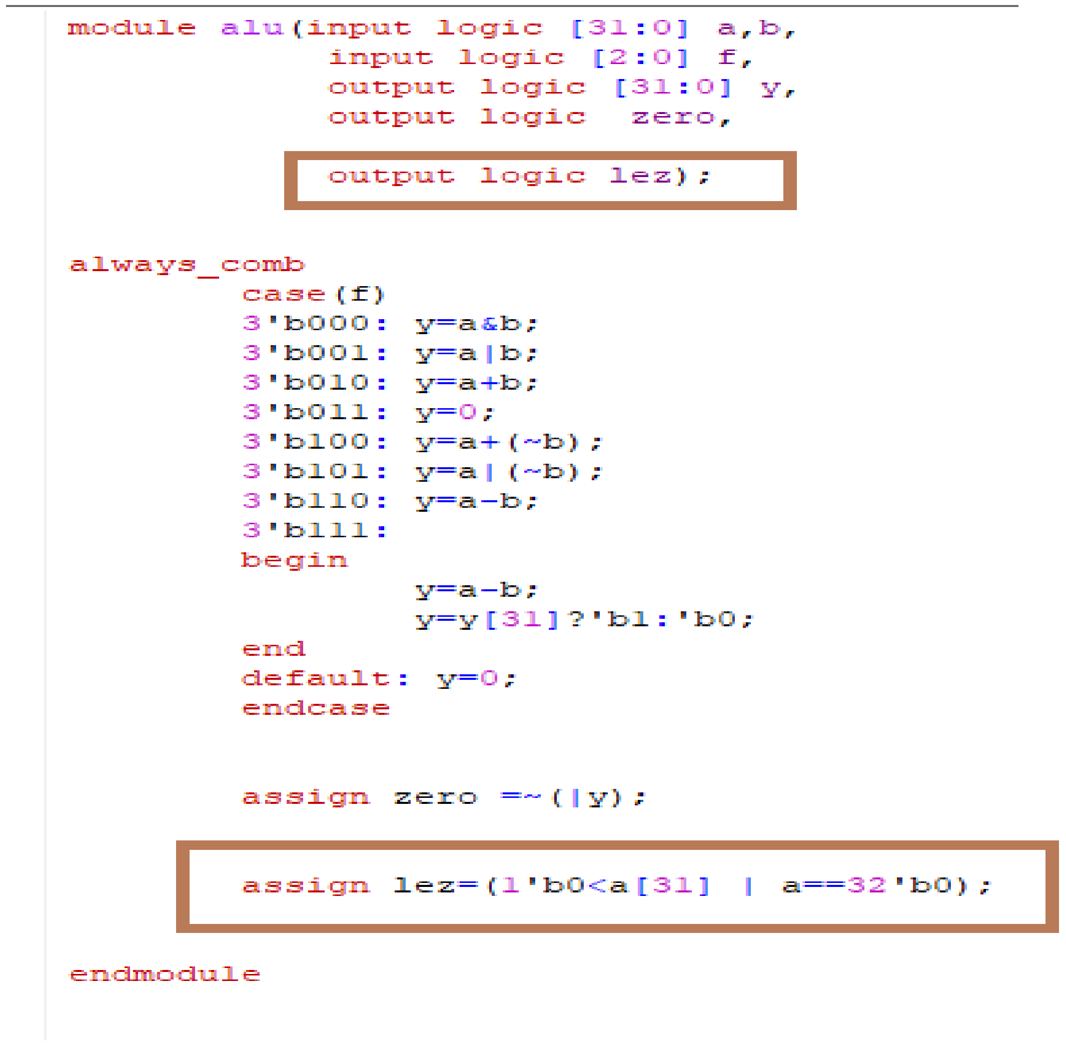
Alu modifications

- *Modifying on diagram :*



Alu modifications

- *Modifying on code :*



```
module alu(input logic [31:0] a,b,
            input logic [2:0] f,
            output logic [31:0] y,
            output logic  zero,

            output logic lez);

always_comb
        case(f)
        3'b000: y=a&b;
        3'b001: y=a|b;
        3'b010: y=a+b;
        3'b011: y=0;
        3'b100: y=a+(~b);
        3'b101: y=a|(~b);
        3'b110: y=a-b;
        3'b111:
        begin
                y=a-b;
                y=y[31]?'b1:'b0;
        end
        default: y=0;
        endcase


        assign zero =~(|y);

        assign lez=(1'b0<a[31] | a==32'b0);

endmodule
```

Data path modification

- *Modifying on diagram :*

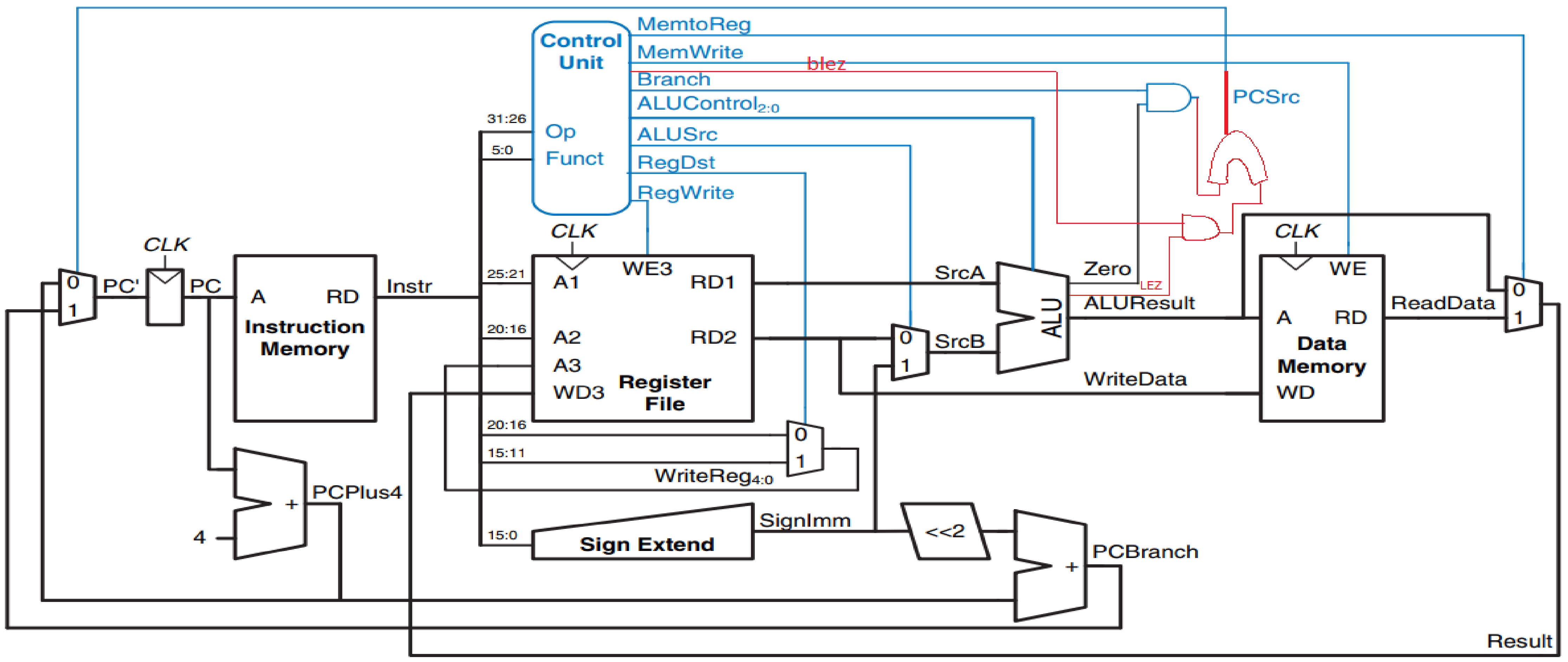
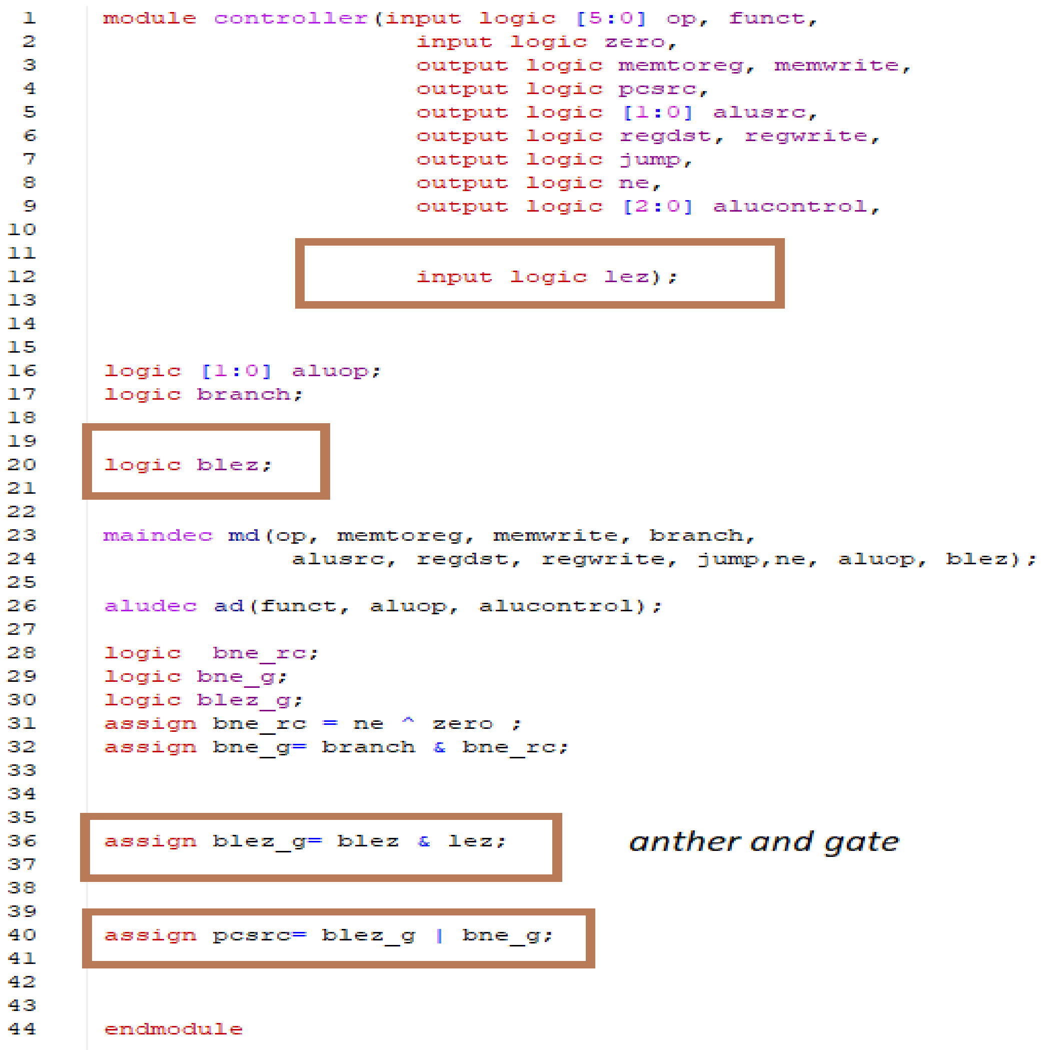


Figure 7.11 **Complete single-cycle MIPS processor**

Data path modification

- *Modifying on code :*

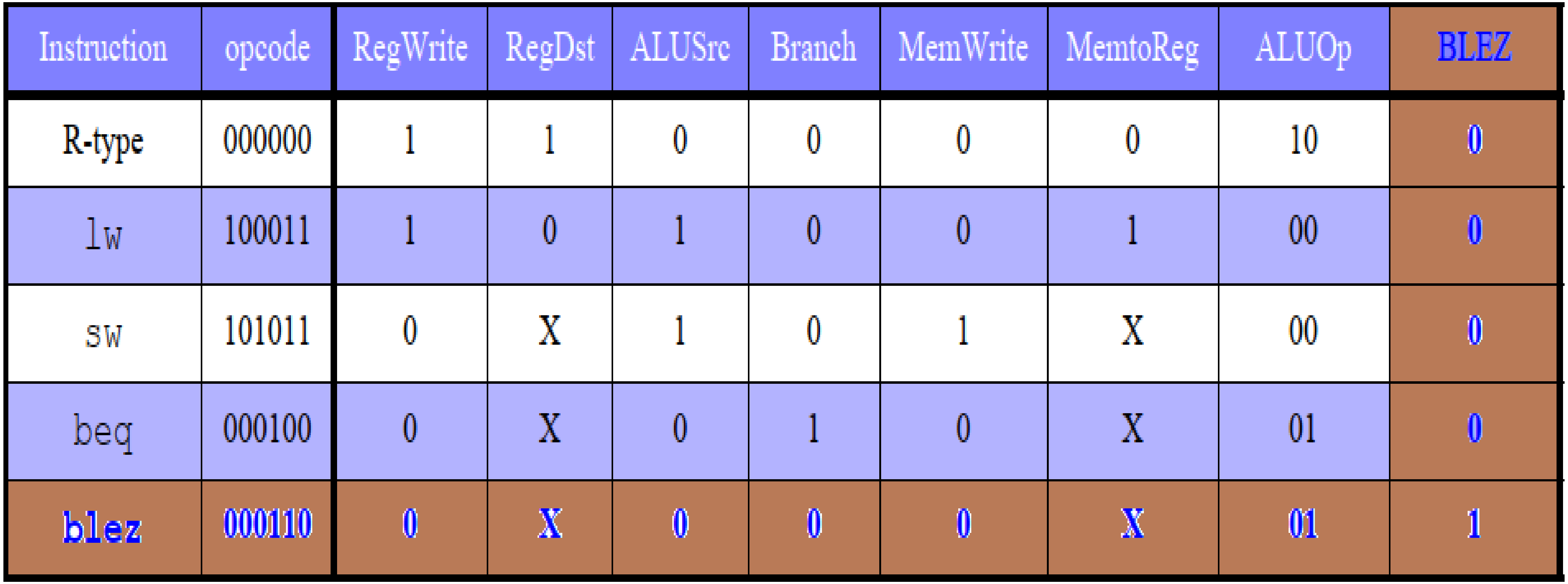


```
module controller(input logic [5:0] op, funct,
                    input logic zero,
                    output logic memtoreg, memwrite,
                    output logic pcsrc,
                    output logic [1:0] alusrc,
                    output logic regdst, regwrite,
                    output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


                    input logic lez);



logic [1:0] aluop;
logic branch;


logic blez;


maindec md(op, memtoreg, memwrite, branch,
            alusrc, regdst, regwrite, jump,ne, aluop, blez);

aludec ad(funct, aluop, alucontrol);

logic  bne_rc;
logic bne_g;
logic blez_g;
assign bne_rc = ne ^ zero ;
assign bne_g= branch & bne_rc;



assign blez_g= blez & lez;



assign pcsrc= blez_g | bne_g;



endmodule
```

anther and gate

Main decoder modification

- *We add anther op code to determine blez instruction :*



Instruction	opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp	BLEZ
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
blez	**000110**	**0**	**X**	**0**	**0**	**0**	**X**	**01**	**1**

finally

- We know that there is anther modifications on codes that we don’t need to represent it here as these modifications are logically ….
- like passing anther parameter when using modified modules or these for adding machine code for blez on meme file to do that we modified.