# *Mips instructions*

## JALR instruction

## Jump and link register

# Description:

- (JALR) is mips instruction

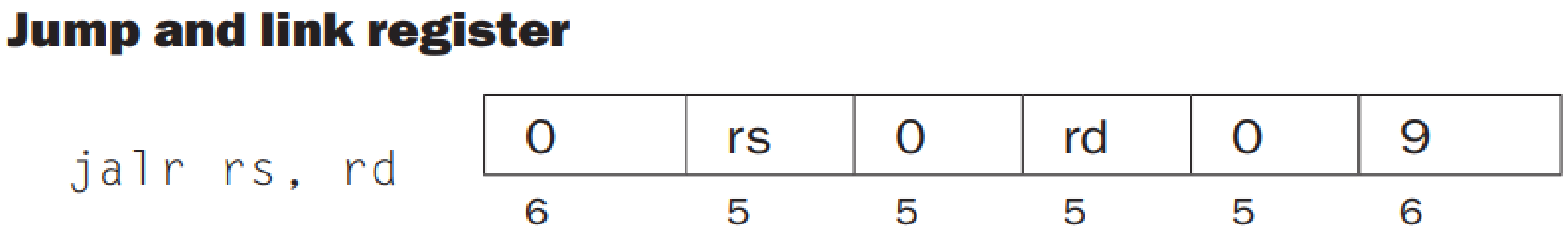
- same as the **jr(jump register)** instruction, except that the return

address is loaded into a specified register (*or $ra if not specified)*

# Format:

- Operation code (op code) : 000000 (R-type)

- Funct :001001 (9)



**Jump and link register**

```
jalr rs, rd
```

| | | | | | |
|---|---|---|---|---|---|
| 0 | rs | 0 | rd | 0 | 9 |
| 6 | 5 | 5 | 5 | 5 | 6 |

# Working principle

*Unconditionally* jump to the instruction whose address is in register rs.

Save the address of the next instruction in register rd (which defaults to 31).

This instruction performs a normal jump but we need to save the return address during the jump

This returned address (pc+4) is saved to register $ra(31)

We added a signal called link to perform this instruction

We added three muses (controlled by the signal link)

First mux to put (pc + 4 ) in the result signal.

second mux to make $ra the destination register (write result in $ ra)

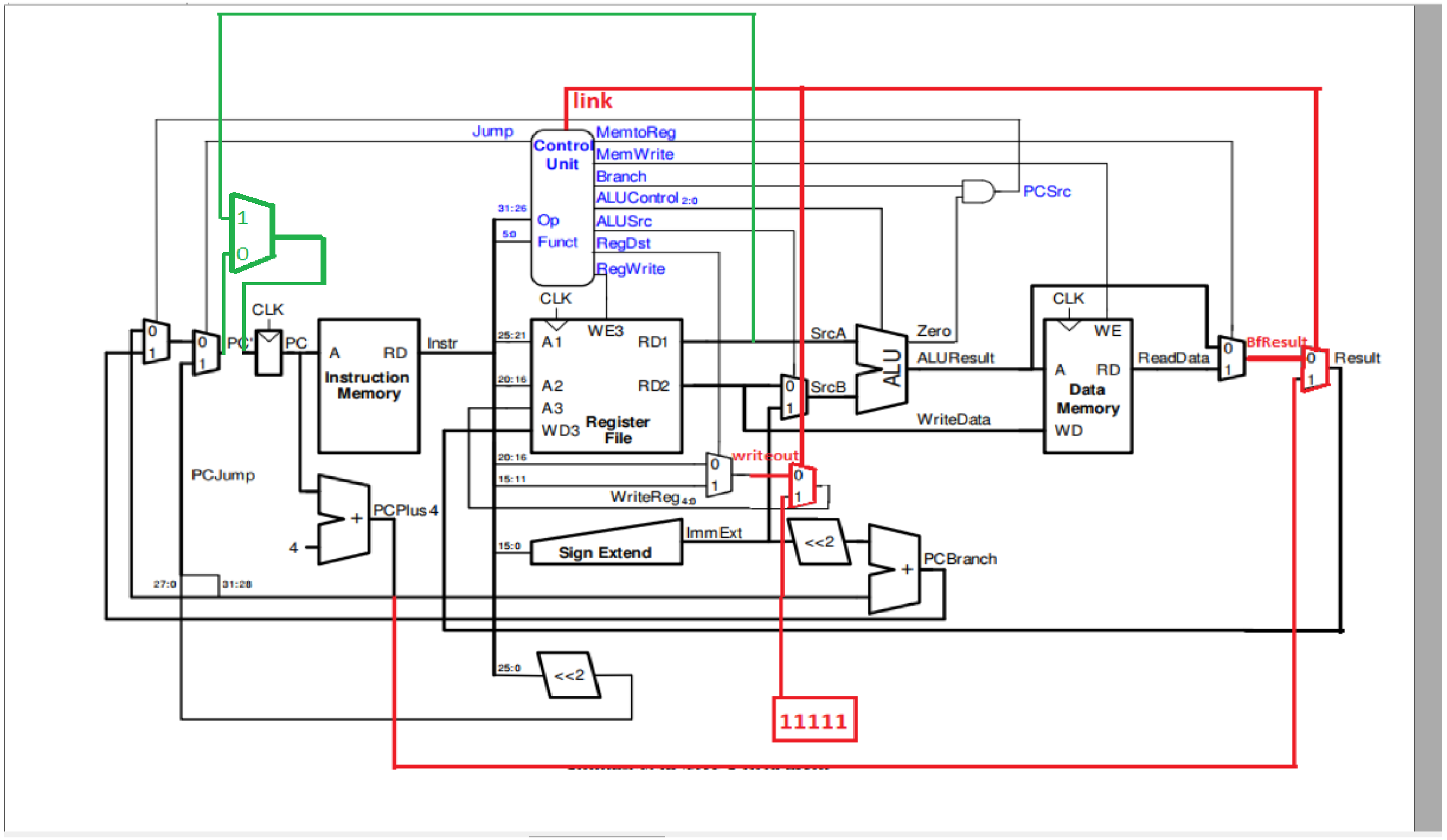
third mux to jump register (pc =rs);

# Modifications:

- We modified 2 things:
- Datapath modifying
- Main decoder modifying

# Data path modification

- *Modifying on diagram :*



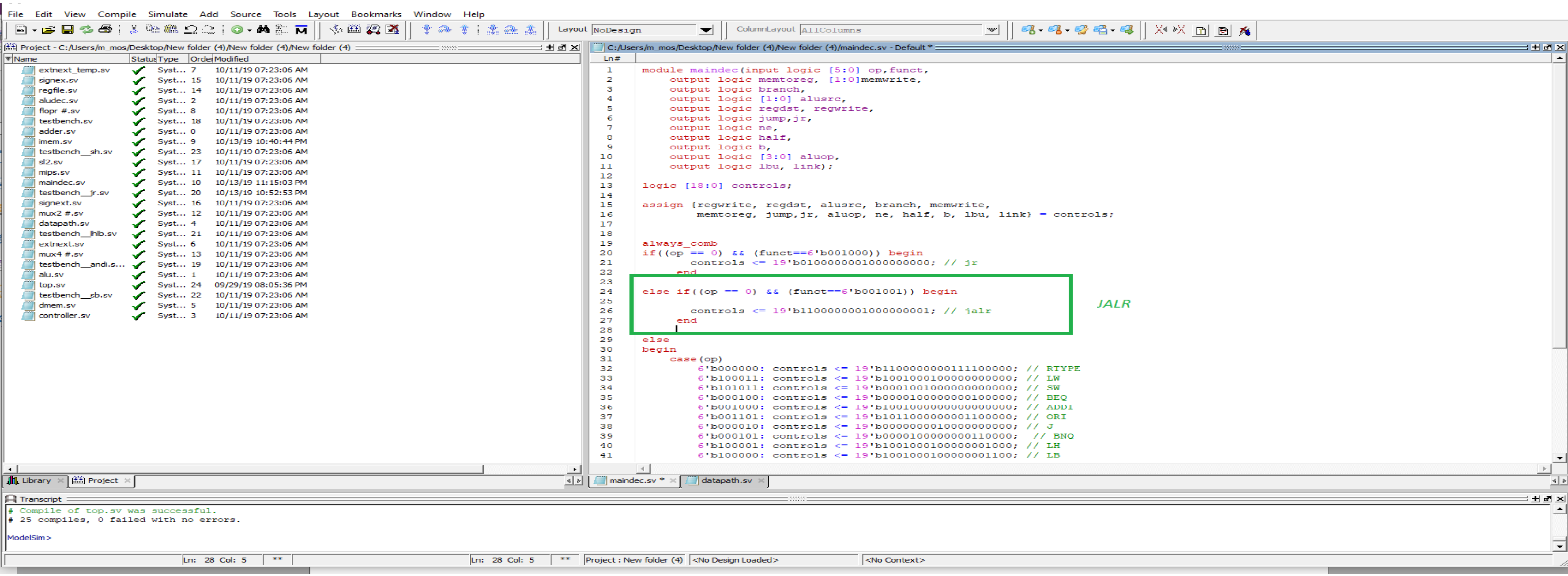
# Data path modification

- *Modifying on code :*

```
flopr #(32) pcreg(clk, reset, pcnext, pc);

adder pcadd1(pc, 32'b100, pcplus4); //normal +4

sl2 immsh(signimm, signimmsh); //jumb

adder pcadd2(pcplus4, signimmsh, pcbranch); //branch or jumb

//half
    signext se2(result_T[15:0], half_result_extended); //extend sign
//mux after the halfword
mux2 #(32) halfmux(result_T,half_result_extended,half,hw_dataMemeoryOutput);
// one byte
signex #(24,8) se3(result_T[7:0], one_byte_result_sign_extended);
//mux after the one byte word
mux2 #(32) ob_mux(hw_dataMemeoryOutput,
                  one_byte_result_sign_extended,
                  b,
                  bfresult);
mux2 #(32) jal_resmux(bfresult, pcplus4, link, result);          Mux 1

mux2 #(32) pcbrmux(pcplus4, pcbranch, pcsrc, pcnextbr);
mux2 #(32) pcmux(pcnextbr, {pcplus4[31:28],
                  instr[25:0], 2'b00}, jump, pcnextj);
mux2 #(32) pcjrmux(pcnextj, srca, jr, pcnext);                  Mux 3

regfile rf(clk, regwrite, instr[25:21], instr[20:16],
           writereg, result, srca, writedata);

mux2 #(5) wrmux(instr[20:16], instr[15:11],
                  regdst, outwrite);
mux2 #(5) linkmux(outwrite, 5'b11111, link, writereg);          Mux 2
// mux2 #(32) resmux(aluout, readdata, memtoreg, result_T);
mux4 #(32) resmux(aluout, readdata, {24'b0,readdata[7:0]},{32'bx}, {lbu,memtoreg},result_T);////hey....:)from mux 2 to 4 and zer

signext se(instr[15:0], signimm); //extend sign

logic [31:0] extimm;
logic [31:0] zeroimm;
extnext ex(instr[15:0], zeroimm);
// ALU logic
```

# Main decoder modification

- *We add anther funct code to determine jalr instruction :*



Project - C:/Users/m_mos/Desktop/New folder (4)/New folder (4)/New folder (4)

| Name | Status | Type | Order | Modified |
|---|---|---|---|---|
| extnext_temp.sv | ✓ | Syst... | 7 | 10/11/19 07:23:06 AM |
| signex.sv | ✓ | Syst... | 15 | 10/11/19 07:23:06 AM |
| regfile.sv | ✓ | Syst... | 14 | 10/11/19 07:23:06 AM |
| aludec.sv | ✓ | Syst... | 2 | 10/11/19 07:23:06 AM |
| flopr #.sv | ✓ | Syst... | 8 | 10/11/19 07:23:06 AM |
| testbench.sv | ✓ | Syst... | 18 | 10/11/19 07:23:06 AM |
| adder.sv | ✓ | Syst... | 0 | 10/11/19 07:23:06 AM |
| imem.sv | ✓ | Syst... | 9 | 10/13/19 10:40:44 PM |
| testbench__sh.sv | ✓ | Syst... | 23 | 10/11/19 07:23:06 AM |
| sl2.sv | ✓ | Syst... | 17 | 10/11/19 07:23:06 AM |
| mips.sv | ✓ | Syst... | 11 | 10/11/19 07:23:06 AM |
| maindec.sv | ✓ | Syst... | 10 | 10/13/19 11:15:03 PM |
| testbench__jr.sv | ✓ | Syst... | 20 | 10/13/19 10:52:53 PM |
| signext.sv | ✓ | Syst... | 16 | 10/11/19 07:23:06 AM |
| mux2 #.sv | ✓ | Syst... | 12 | 10/11/19 07:23:06 AM |
| datapath.sv | ✓ | Syst... | 4 | 10/11/19 07:23:06 AM |
| testbench__lhlb.sv | ✓ | Syst... | 21 | 10/11/19 07:23:06 AM |
| extnext.sv | ✓ | Syst... | 6 | 10/11/19 07:23:06 AM |
| mux4 #.sv | ✓ | Syst... | 13 | 10/11/19 07:23:06 AM |
| testbench__andi.s... | ✓ | Syst... | 19 | 10/11/19 07:23:06 AM |
| alu.sv | ✓ | Syst... | 1 | 10/11/19 07:23:06 AM |
| top.sv | ✓ | Syst... | 24 | 09/29/19 08:05:36 PM |
| testbench__sb.sv | ✓ | Syst... | 22 | 10/11/19 07:23:06 AM |
| dmem.sv | ✓ | Syst... | 5 | 10/11/19 07:23:06 AM |
| controller.sv | ✓ | Syst... | 3 | 10/11/19 07:23:06 AM |

C:/Users/m_mos/Desktop/New folder (4)/New folder (4)/maindec.sv - Default *

```
module maindec(input logic [5:0] op,funct,
     output logic memtoreg, [1:0]memwrite,
     output logic branch,
     output logic [1:0] alusrc,
     output logic regdst, regwrite,
     output logic jump,jr,
     output logic ne,
     output logic half,
     output logic b,
     output logic [3:0] aluop,
     output logic lbu, link);

logic [18:0] controls;

assign {regwrite, regdst, alusrc, branch, memwrite,
         memtoreg, jump,jr, aluop, ne, half, b, lbu, link} = controls;


always_comb
if((op == 0) && (funct==6'b001000)) begin
         controls <= 19'b0100000001000000000; // jr
       end

else if((op == 0) && (funct==6'b001001)) begin

         controls <= 19'b1100000001000000001; // jalr
       end

else
begin
     case(op)
          6'b000000: controls <= 19'b1100000000111100000; // RTYPE
          6'b100011: controls <= 19'b1001000100000000000; // LW
          6'b101011: controls <= 19'b0001001000000000000; // SW
          6'b000100: controls <= 19'b0000100000000100000; // BEQ
          6'b001000: controls <= 19'b1001000000000000000; // ADDI
          6'b001101: controls <= 19'b1011000000001100000; // ORI
          6'b000010: controls <= 19'b0000000010000000000; // J
          6'b000101: controls <= 19'b0000100000000110000;  // BNQ
          6'b100001: controls <= 19'b1001000100000001000; // LH
          6'b100000: controls <= 19'b1001000100000001100; // LB
```

JALR

Transcript

```
# Compile of top.sv was successful.
# 25 compiles, 0 failed with no errors.

ModelSim>
```