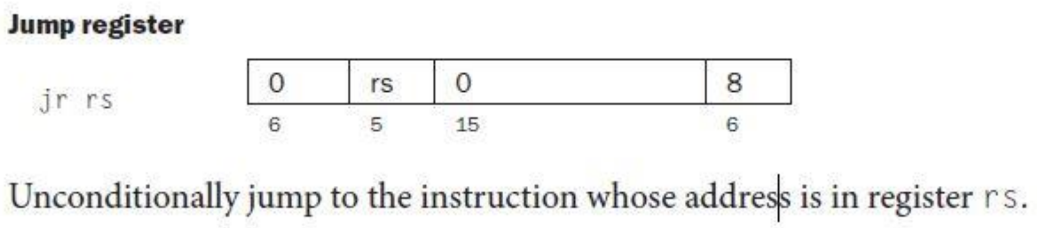
JUMP REGISTER (JR)

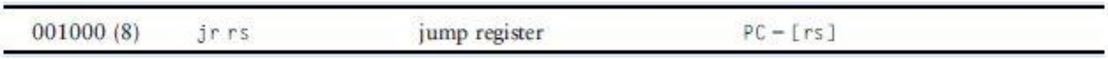


Jump register

jr rs

0	rs	0	8
6	5	15	6

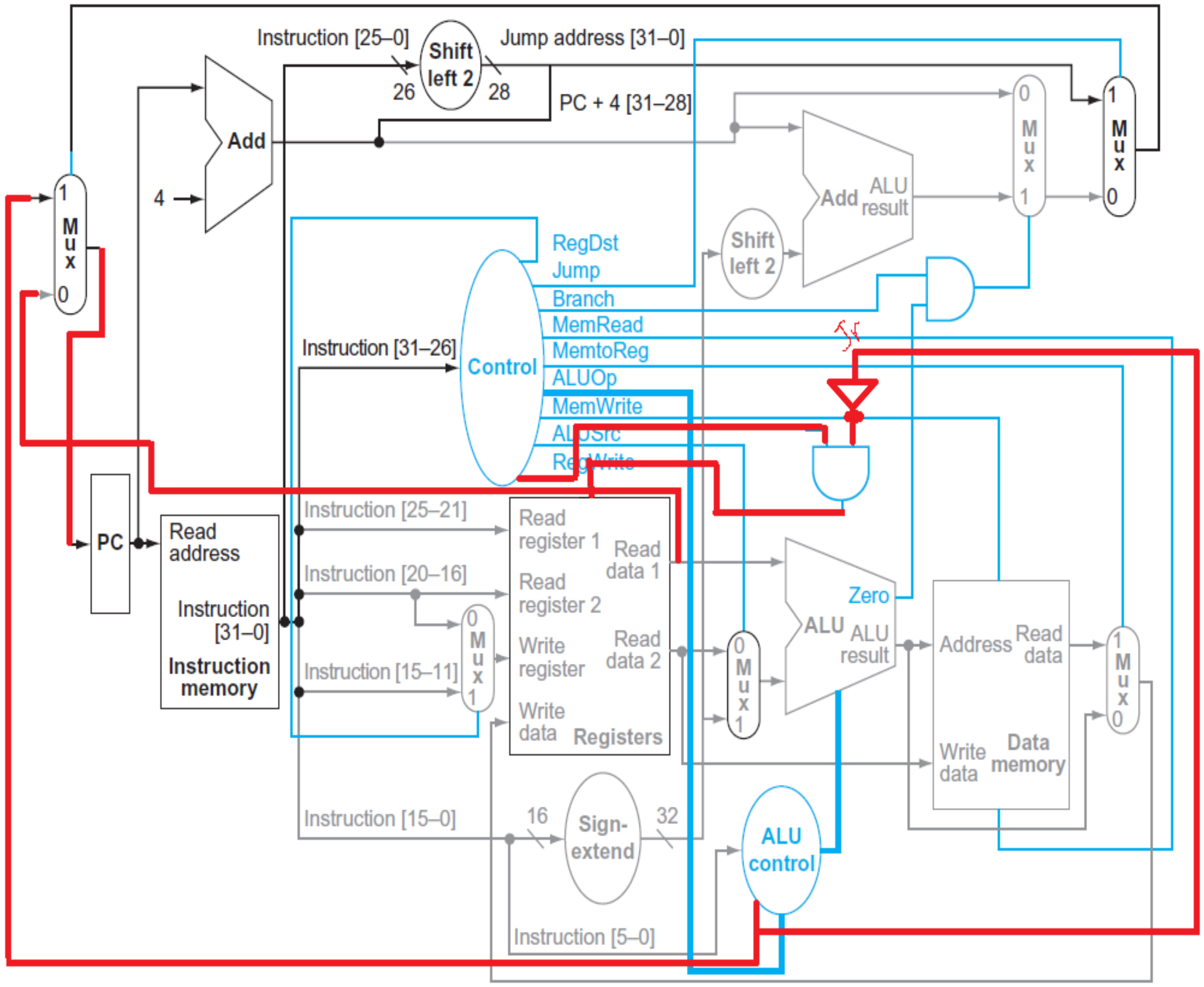
Unconditionally jump to the instruction whose address is in register rs.



001000 (8)	jr rs	jump register	PC – [rs]

- JR instruction sets the PC to the content of the register, so we have to provide a way for this data from the register file (Read data 1 port).
- a mux is needed to control whether the PC will take the value coming from the register file via the added wire or not.
- a signal has to be led from the controller to the newly added mux to control it.
- in case Jr instruction we are not need to write in register file so, we added an and gate.

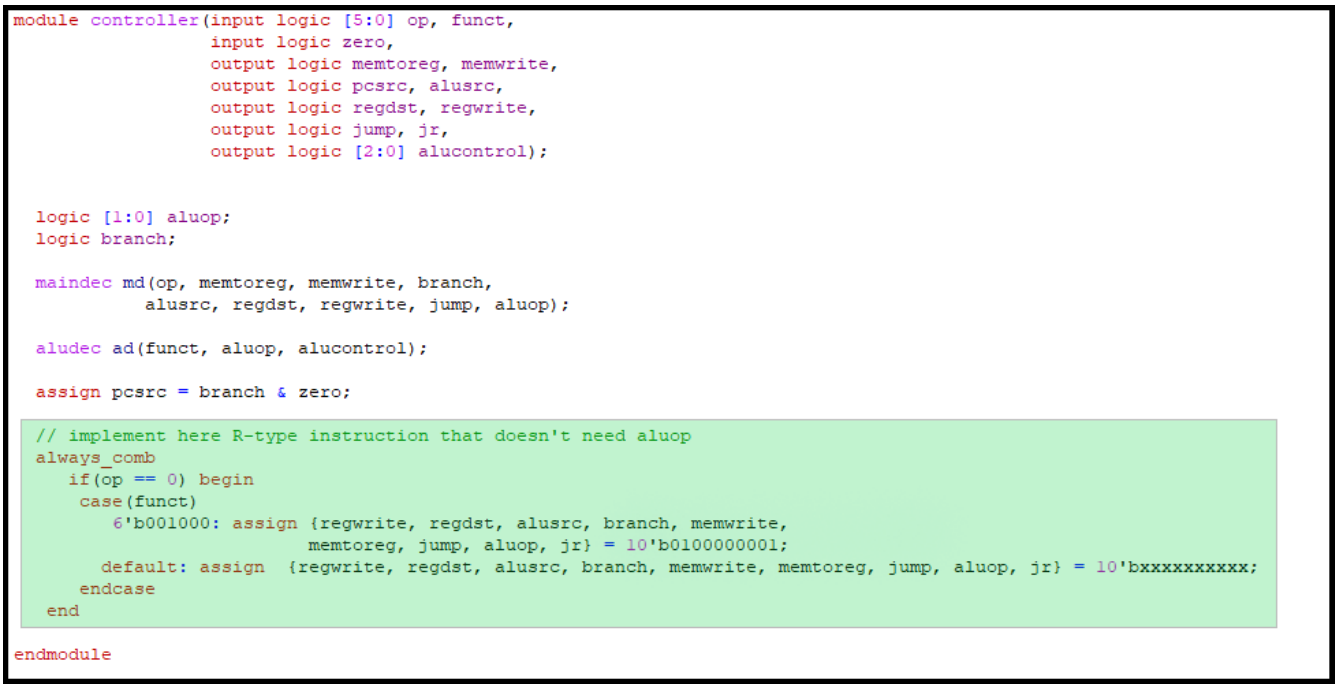
Datapath modified:



Control :

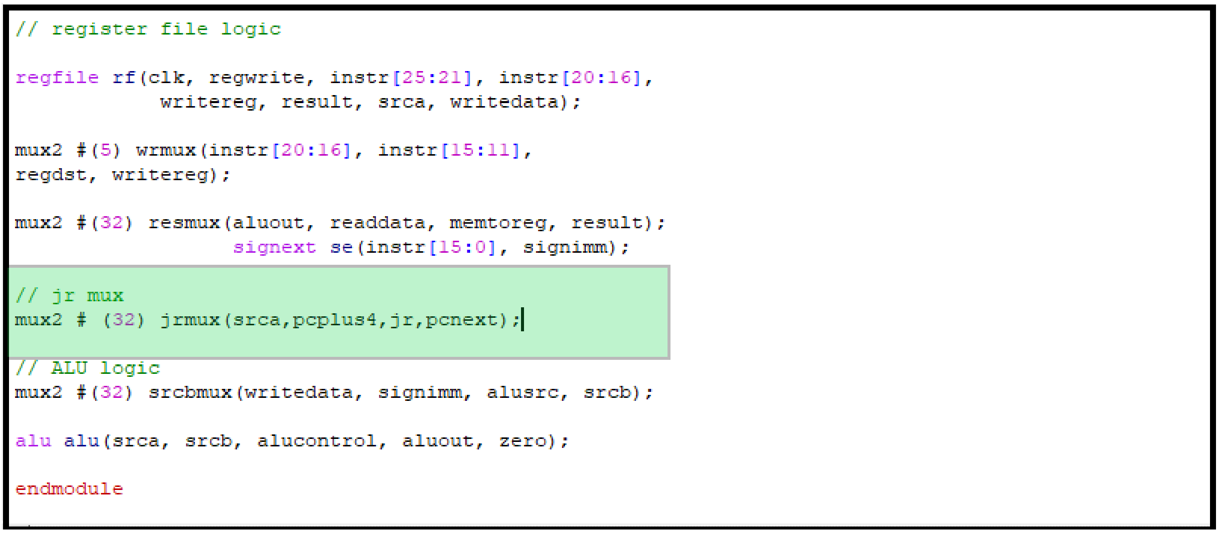
Jump	MemtoReg	MemWrite	Branch	AluControl	AluSrc	RegDst	RegWrite	Jr
0	0	0	0	00	0	1	1	1

Changes to the controller module:



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


  logic [1:0] aluop;
  logic branch;

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

  aludec ad(funct, aluop, alucontrol);

  assign pcsrc = branch & zero;

  // implement here R-type instruction that doesn't need aluop
  always_comb
     if(op == 0) begin
      case(funct)
         6'b001000: assign {regwrite, regdst, alusrc, branch, memwrite,
                           memtoreg, jump, aluop, jr} = 10'b0100000001;
        default: assign  {regwrite, regdst, alusrc, branch, memwrite, memtoreg, jump, aluop, jr} = 10'bxxxxxxxxxx;
      endcase
   end

endmodule
```

Changes made to the datapath module:



```
// register file logic

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

mux2 #(5) wrmux(instr[20:16], instr[15:11],
regdst, writereg);

mux2 #(32) resmux(aluout, readdata, memtoreg, result);
                  signext se(instr[15:0], signimm);

// jr mux
mux2 # (32) jrmux(srca,pcplus4,jr,pcnext);
// ALU logic
mux2 #(32) srcbmux(writedata, signimm, alusrc, srcb);

alu alu(srca, srcb, alucontrol, aluout, zero);

endmodule
```