Implementation Details of Instructions

- **R-Type Instructions**

  - **jr:** I have added another input named “jr” to the control unit. This input is 5_0 bits of instruction. jr is R-type instruction and has unique function code. I check the function code in control unit and if it is jr jump1 becomes “1” and regwrite becomes “0”.

  - **nor:** I just updated the alu32 to nor instruction.

- **I-Type Instructions**
  - **addi:** alusrc and regwrite signals must be “1” and aluop must be “0000”.
  - **andi:** alusrc and regwrite signals must be “1” and aluop must be “0010”.
  - **ori:** alusrc and regwrite signals must be “1” and aluop must be “0011”.
  - **bne, beq, bgtz, blez, bltz:** aluop must be “0001” for these instructions. I have added new inputs named c0, c1, c2 to the ALU Control unit to distinguish between these instructions. c2c1c0 = instruction[28-26]. Briefly, I pass the opcodes of these instructions to the ALU Control unit to send a more specific gout signal to the alu32.
  - **bgez:** aluop must be “1001” for this instruction because it’s opcode is same with bltz. Difference between them is rt register. So, to check rt I have added new input to the control input which named “rt”. I have added aluop3 signal to check only this instruction.

- **J-Type Instructions**
  - **j:** I have added jump0 signal to implement j instruction. If opcode is j instruction opcode, then jump0 signal is “1”. So jump1jump0 = 01 and PCSrc multiplexer will send input1 to the out5 wire.
  - **jal:** I have added jump1 signal to implement jal instruction. If instruction is jal instruction then jump1jump0 = 10. In this case, PCSrc multiplexer must send input2 to the out5 wire. Also, memtoreg1 must be “1”. Because memtoreg multiplexer must send input2 to the out3. Also, regdest1 must be “1” because we must write to the register ra.

Other Details

I have needed some multi-bit control lines to implement the instructions. So, I change some parts of control unit in source code.

memtoreg -> memtoreg1, memtoreg0

jump -> jump1, jump0

regdest -> regdest1, regdest0

aluop1, aluop0 -> aluop3, aluop2, aluop1, aluop0

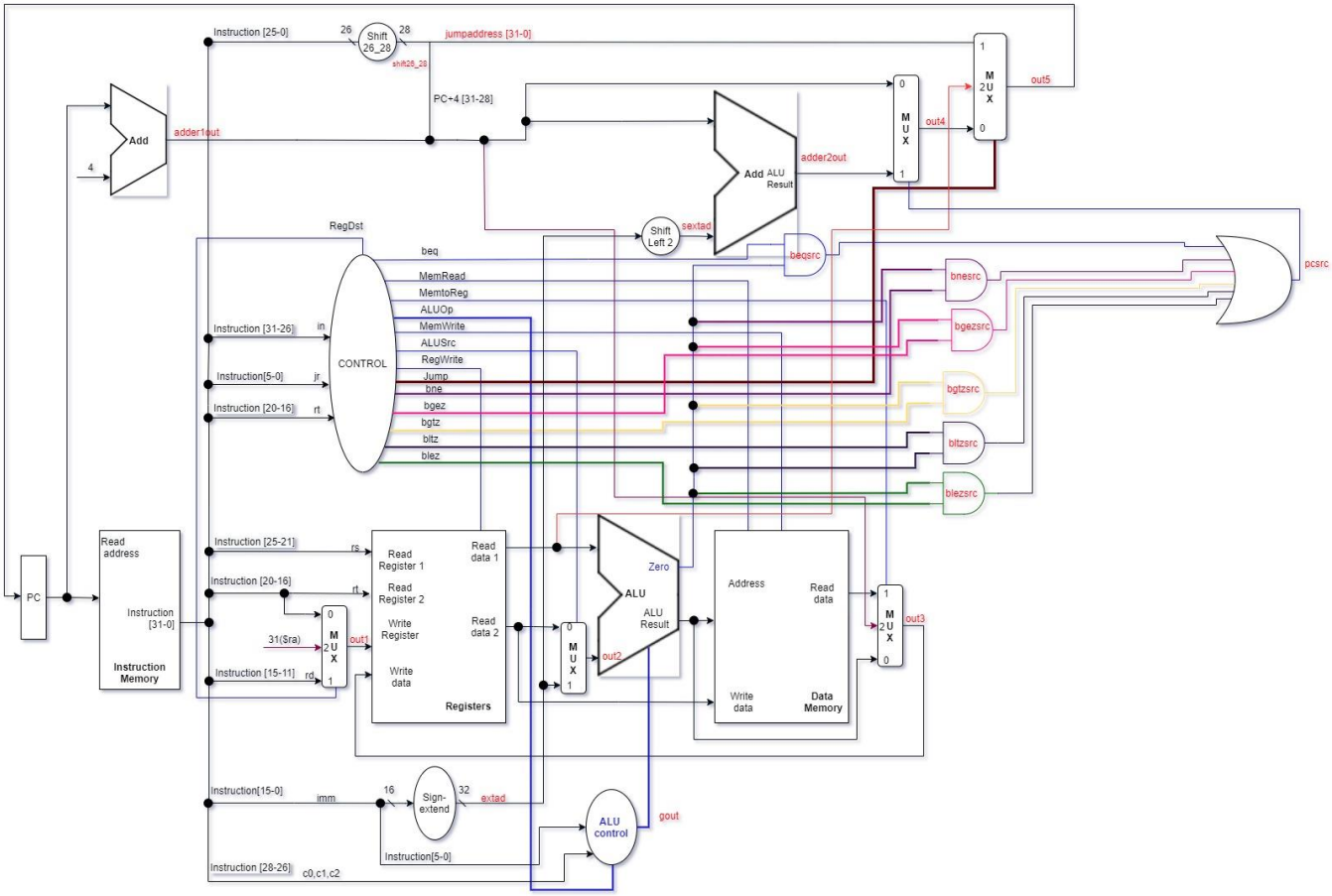
I used 4-bit ALUOp signal to distinguish between instructions' opcodes.

I used some 3:1 multiplexers to implement some instructions.

Control Signals for Instructions

	Regdest	alusrc	Memtoreg	regwrite	memread	memwrite	Aluop	jump	Beq	Bne	Bgtz	Bgez	Bltz	Blez
jr	xx	x	xx	0	0	0	xxxx	10	0	0	0	0	0	0
nor	01	0	00	1	0	0	0110	00	0	0	0	0	0	0
addi	xx	1	xx	1	x	x	0000	00	0	0	0	0	0	0
andi	00	1	00	1	0	0	0010	00	0	0	0	0	0	0
ori	00	1	00	1	0	0	0011	00	0	0	0	0	0	0
bne	00	0	00	0	0	0	0001	00	0	1	0	0	0	0
bgez	00	0	00	0	0	0	1001	00	0	0	0	1	0	0
bgtz	00	0	00	0	0	0	0001	00	0	0	1	0	0	0
blez	00	0	00	0	0	0	0001	00	0	0	0	0	0	1
bltz	00	0	00	0	0	0	0001	00	0	0	0	0	1	0
jal	10	0	10	1	0	0	xxxx	10	0	0	0	0	0	0
j	00	0	0	00	0	0	xxxx	01	0	0	0	0	0	0

Datapath for Instructions



In the code I add comments to explain what I did.