CS M151B: Homework 4 Solutions

4.7

4.7.1

Sign-extend
00000000000000000000000000010100

Jump's shift-left-2
00011000100000000000001010000

4.7.2

ALUOp[1-0]
00

Instruction[5-0]
010100

4.7.3

New PC Path
PC to Add (PC+4) to branch Mux to jump Mux to PC

4.7.4

WrReg Mux: 2 or 0 (RegDst is X)

ALU Mux: 20

Mem/ALU Mux: (Mem) or 17

Branch Mux: PC+4

Jump Mux: PC+4

4.7.5

ALU: -3 and 20

Add (PC+4): PC and 4

Add (Branch): PC+4 and 80

4.7.6

Read Register 1: 3

Read Register 2: 2

Write Register: 2 or 0

Write Data: (Mem) or 17

RegWrite: 0

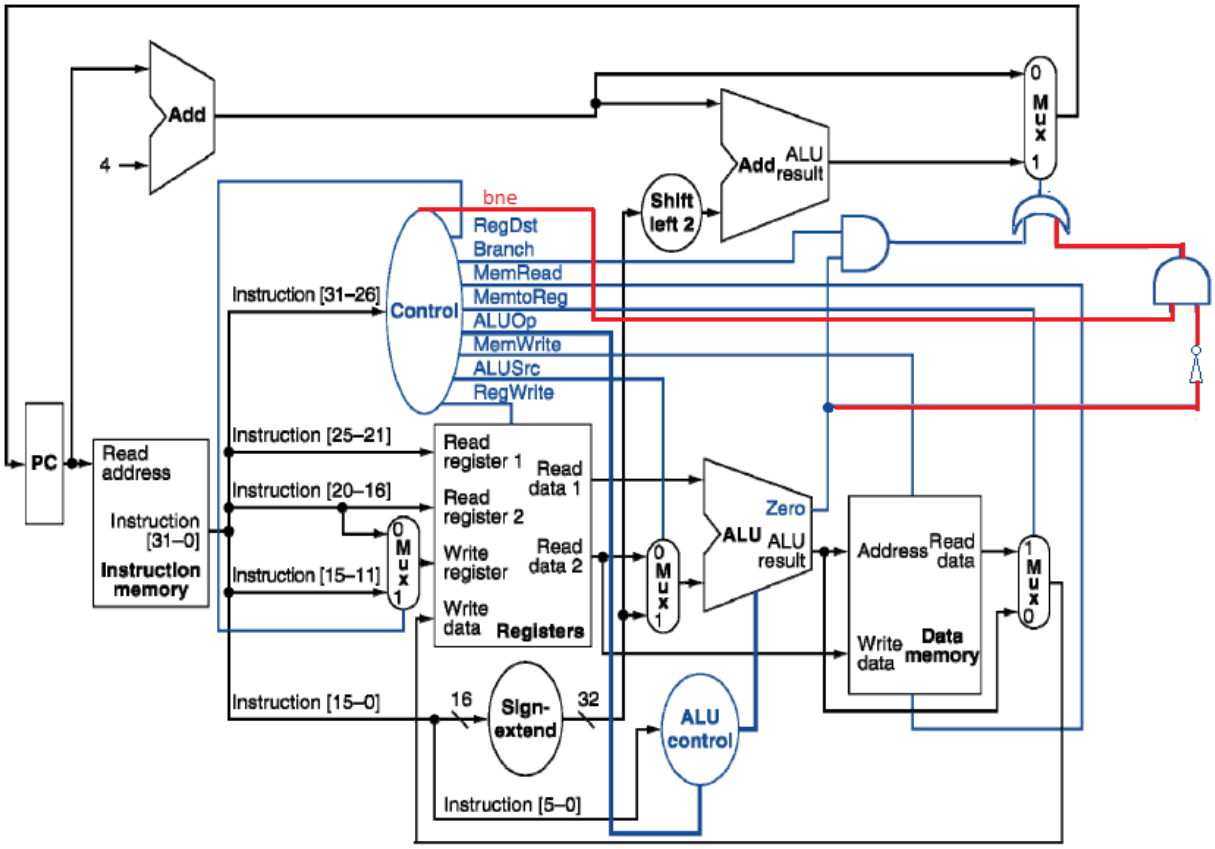
---

2. bne

if (R[rs] != R[rt])
  PC = PC + 4 + SE(I)
else
  PC = PC + 4

This is similar to the beq instruction except we would like to branch if the operands are NOT equal. This means we can have the ALU executing the same subtraction operation and use the same Zero signal, except now we will invert Zero so that the signal is 1 if the R[rs] and R[rt] are NOT equal. Similar to the beq operation, we will AND the ~Zero signal and a new bne control signal together which will be used to indicate that we want to branch.

Since the normal branch logic already allows for choosing between PC + 4 + SE(I) vs. PC + 4, we can reuse the logic by ORing the ANDed signals for beq and bne together. Alternatively, another solution would be to have another multiplexor before the PC.



Main Controller:
Opcode: [Some new I-Type opcode]

RegDst: X (not writing to register)
ALUSrc: 0 (comparing R[Rs] and R[Rt])
MemtoReg: X (not writing to register)
RegWrite: 0 (don't write to register)
MemRead: 0 (don't read from memory)
MemWrite: 0 (don't write to memory)
Branch: 0 (don't branch on equal)
ALUOp1: 0
ALUOp2: 1 (Have ALU subtract)
bne: 1

---

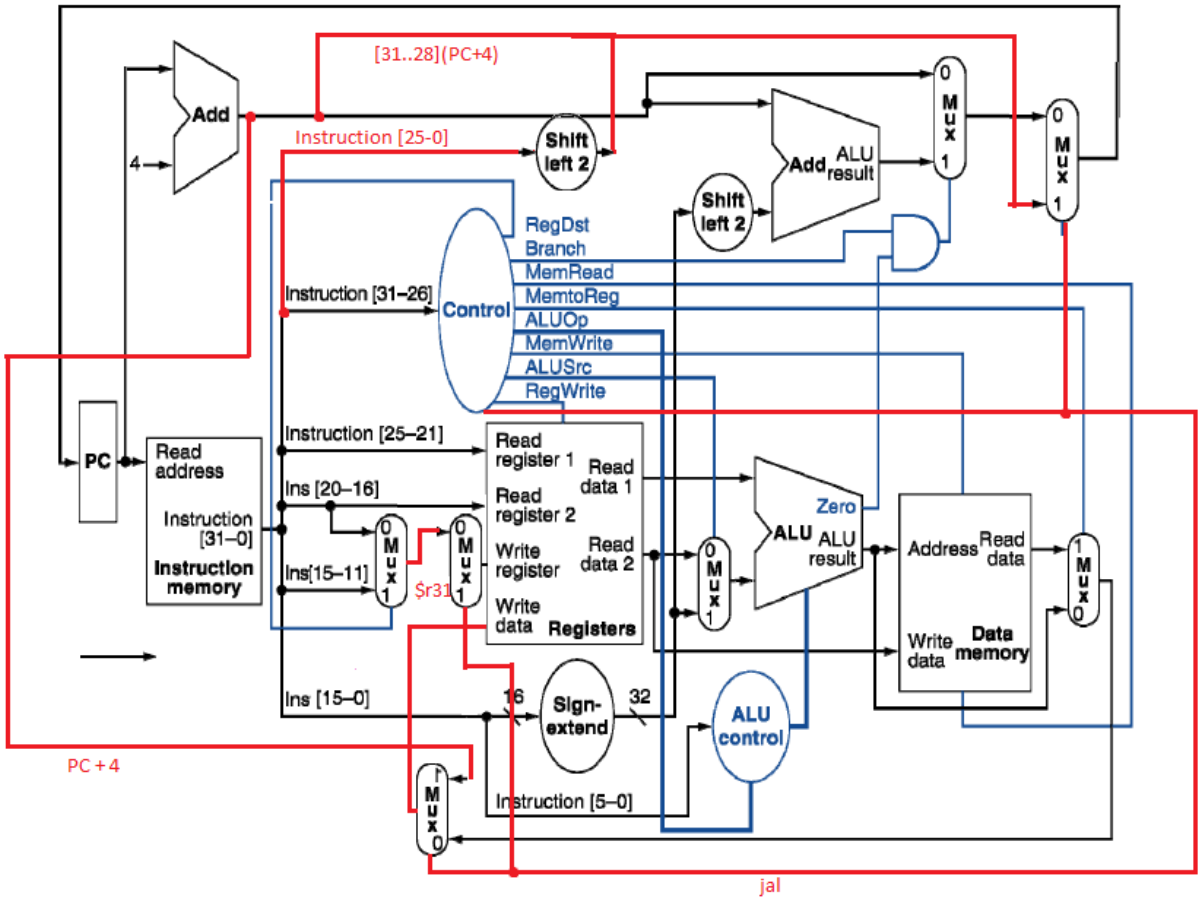
3. jal

R[$r31] = PC+4
PC = [31..28](PC+4) | [27..0] (I<<2)

First, we must have a way to choose the new PC as being [31..28](PC+4) | [27..0] (I<<2) rather than the typical options. As a result, we must extract bits 0 to 25 from the instruction, shift that by 2, and then merge that with bits 28 to 31 of the PC + 4. Then, we have to add an additional multiplexor before the PC to select between the new jump value. This mux will be controlled by a new control signal jal.

Additionally, we must be able to select register $r31 as the write register destination rather than the rt or rd. This will require a multiplexor before write register port of the register file.

Finally, we must be able to select PC + 4 as the write data rather than the output of the ALU/Memory. This will require yet another multiplexor before the write data port of the register file.



Main Controller:
Opcode: [Some new J-type opcode]

RegDst: X (overridden by new mux)
ALUSrc: X (not using ALU)
MemtoReg: X (overridden by new mux)
RegWrite: 1 (write to register $31)
MemRead: 0 (don’t read from memory)
MemWrite: 0 (don’t write to memory)
Branch: 0 (don’t branch on equal)
ALUOp1: X
ALUOp2: X (not using ALU)
jal: 1

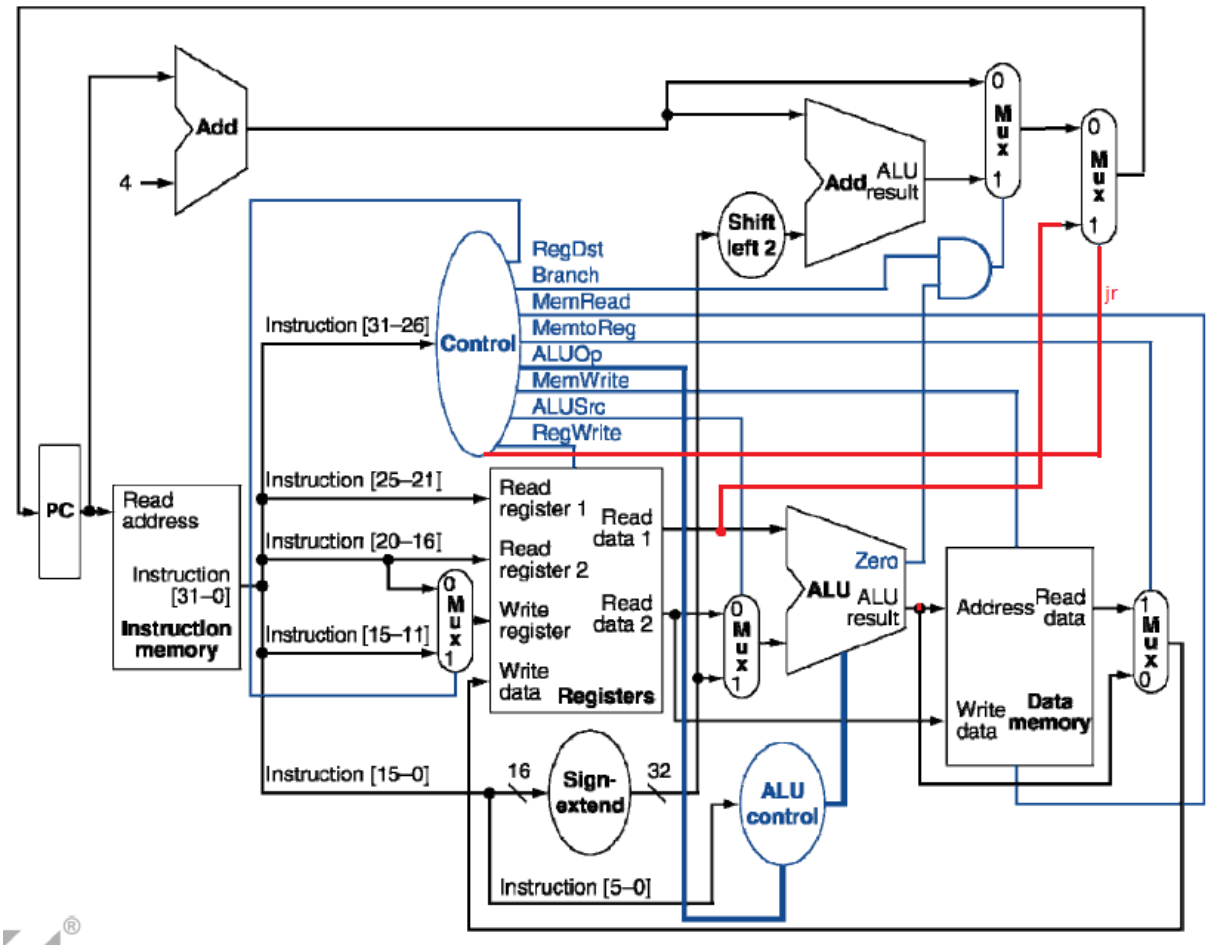
---

4. jr

PC = R[rs]

We must be able to select R[rs] rather than PC + 4 or PC + 4 + SE(I). This will require a link from the Read Data 1 port of the register file leading to a multiplexor before the PC that allows us to choose R[rs] as the new PC. This requires a new control signal jr.

Note that this is an R-type Instruction which has an opcode of 000000, which means that normally the control unit would not be sufficient in setting the control signals. Some solutions would be to have the ALU Control consider the funct field and then somehow change the control signals. Additionally, you could also have the control unit read in the funct field. However, in accordance to what the professor said in class, you are permitted to define a new unique opcode for this R-Type instruction.



Main Controller:
Opcode: [Some new R-type opcode]

RegDst: X (not writing to register)
ALUSrc: X (not using ALU)

MemtoReg: X (not writing to register)
RegWrite: 0 (don’t write to register)
MemRead: 0 (don’t read from memory)
MemWrite: 0 (don’t write to memory)
Branch: 0 (don’t branch on equal)
ALUOp1: X
ALUOp2: X (not using ALU)
jr: 1