Problem 4.7

10101100011000100000000000010100

4.7.1

Output of sign-extend [: 0000000000000000 (16) 0000000000010100 [15-0]

Output of jump shift left 2: 0000 00011000100000000000010100 [25-0] 00

4.7.2

Values of ALU control unit's inputs

101011 -> 1+2+8+32 = 43 -> sw $rt, <offset>$rs (register to memory)

Instruction [5-0]: 010100 -> not relevant for loads or stores (don't care)

Loads and stores add -> ALU Op 00

Therefore ALU control input 0010

4.7.3

New PC address after execution

Highlight path through which value is determined

PC outputs goes into ALU to be added to 4 -> PC+4

This output gets directed into a MUX into the 0 port

The branch control is set to 0 so PC+4 (on the 0 port)

moves through. This output goes to the 0 port of the next MUX

Not a jump instructions so jump control set to 0 -> so PC+4 moves

through the MUX again. Output gets put back into PC so PC has been

set to PC+4.

4.7.4

For each MUX, show value of data output during execution and register values

Write Register MUX would have control of DC because we are storing into memory

so not touching register file.

Therefore, we don't care what comes out of the MUX because we won't be using

the write register.

ALU MUX would have control of 1 because we want to use the immediate, not the second register for the ALU.

Output of sign-extend [: 0000000000000000 (16) 0000000000010100 [15-0]

This would be our output for the ALU MUX -> 20

The ALU/Mem MUX would have control of DC because we won't be writing anything to the register file, so we don't care what goes into the write data port.

Therefore, we don't care what comes out of the MUX.

From the previous parts, we said that the Branch and Jump MUX will both output PC+4.

4.7.5

For ALU and two add units, what are data input values?

For the top ALU unit, the inputs are PC+4 and sign-extended 16-bit immediate shifted left 2. We found that the 16-bit immediate was 20 so shifting left 2 would result in 20*4 = 80.

For the bottom ALU unit, one of the inputs is Read data 1. Read register 1 is 00011 -> register 3. From the table, read data 1 is -3. The other input will be the sign-extended 16-bit immediate, which we found to be 20.

The PC adder unit will have inputs of PC and 4.

4.7.6

Values of all inputs for the "Registers" unit?

Read register 1 will take bits 25-21 as input, which are 00011 = 3

Read register 2 will take bits 20-16 as input, which are 00010 = 2

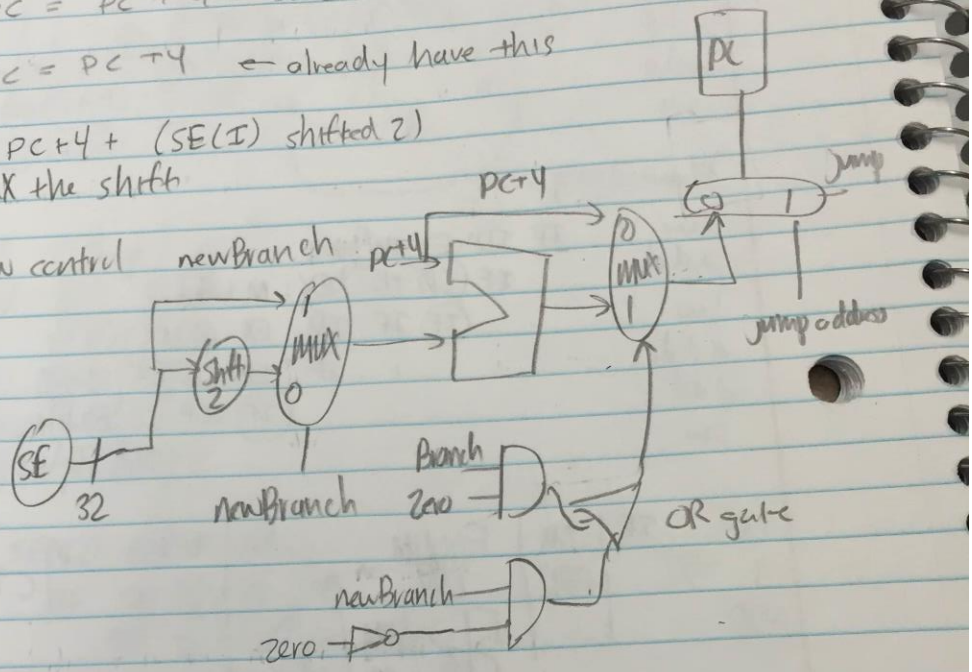
Since we are doing a store (from register to memory) we don't care what goes into the write register or the write data ports (because we aren't writing to the register file). It follows that the RegWrite control will be 0.

Homework 4

1) BNE instruction (I-type)
   if (R[rs] != R[rt])
      PC = PC + 4 + SE(I)
   else
      PC = PC + 4 ← already have this

Have PC+4 + (SE(I) shifted 2)
   MUX the shift

Create new control newBranch



RegDst X (not writing to RF)
Jump 0 (do not want to jump)
Branch 0 (must be 0, don't want AND gate to output 1)
MemRead 0 (don't want to read random stuff)
MemtoReg X (not writing to RF)
ALUOp 01 (subtract like normal branch)
MemWrite 0 (don't want write random stuff)
ALUSrc 0 (want to compare 2 registers, not immediate)
RegWrite 0 (don't write to RF)
newBranch 1 → (should be 0 for all other instructions)

2) jal instruction – J type instruction

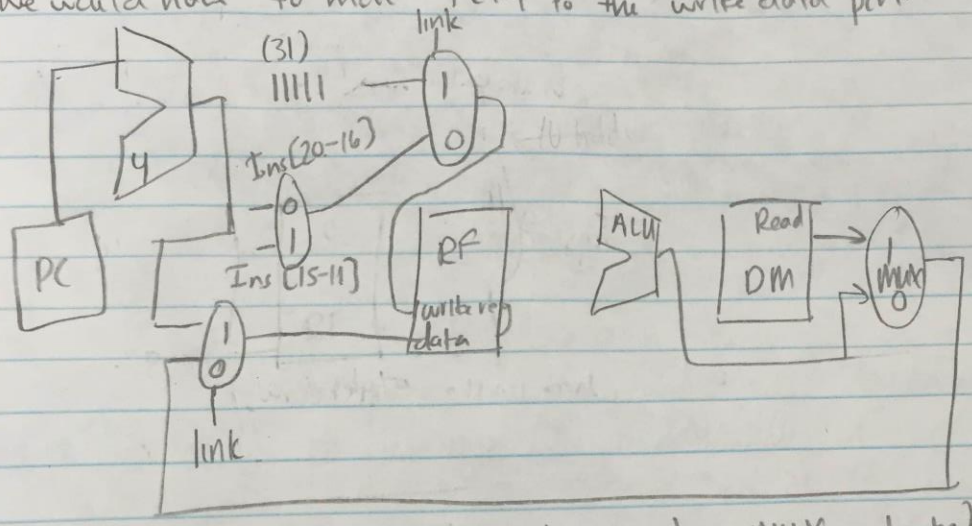
R[$r31] = PC+4

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

(jump instruction covered but also (always) writes PC+4 to $r31

Therefore if Jump control = 1, our PC will be correct

We would have to move PC+4 to the write data port



RegDst X (doesn't matter which reg that MUX selects)

Jump 1 (want the PC to jump)

Branch X (doesn't matter b/c will always take the jump input in 2nd

MemRead 0 (don't want to read nonsense)

MemtoReg X (doesn't matter what comes out of memory, also not reading

ALUOp X (not using the ALU to calculate anything)

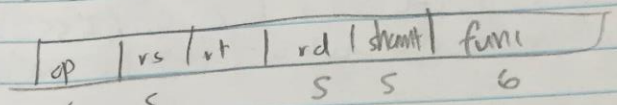
MemWrite 0 (don't want to write nonsense)

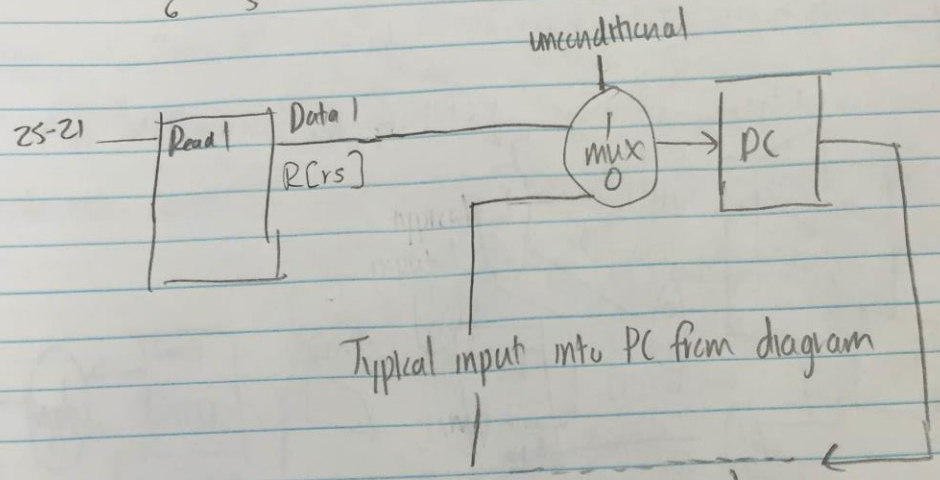
ALUSrc X → Same

RegWrite 1 (want to write to $r31)

Link 1 (0 for all other commands)

3) jr instruction - R type instruction
PC = R[rs]





RegDst X (Don't care what write register is)
Jump X (Will affect typical input into PC, which not selected in MU
Branch X (Same " ——————→)
MemRead 0 (Don't want to read nonsense)
MemtoReg X (Don't care what goes to write data)
ALUOp X (Not using ALU)
MemWrite 0 (Don't want to write nonsense)
ALUSrcA X (Not using ALU)
RegWrite 0 (Don't want to write to RF)
unconditional 1 (0 for rest of instructions)