CS151B/EE116C – Solutions to Homework #4

4.7
The instruction is:
101011 00011000100000000000010100
…which is a store word (sw) instruction.

4.7.1
Sign-extend: 00000000000000000000000000010100
- The lower sixteen bits of the instruction, sign extended to 32-bits

Jump's shift-left-2: 0001100010000000000001010000
- The lower 26-bits of the instruction, shifted left by 2 (two additional 0s added to the end)

4.7.2
ALUOp[1:0]: 00
- Based on the opcode (101011), the ALUOp is 00 in order to add.

Instruction[5:0]: 010100

4.7.3
New PC Path: PC → Add (PC+4) → Input 0 of the Branch Mux → Input 0 of the Jump Mux → PC
- Both Branch and Jump signals will be 0, resulting in PC = PC + 4

4.7.4
RegDst Mux: 2 or 0
- RegDst is a "don't care", which means the MUX may be outputting either of the inputs depending on the state of RegDst.

ALUSrc Mux: 20
- Because ALUSrc is 1 in an sw instruction, the selected input will be the sign extended immediate (000…0010100).

MemtoReg Mux: <Memory Read Data signal> or 17
- MemtoReg is a "don't care", which means the MUX may be outputting an arbitrary signal from the Read Data port of the Data Memory or the output from the ALU, which is 17.

Branch Mux: PC+4
Jump Mux: PC+4
- Both Branch and Jump control signals are 0, which selects PC + 4.

4.7.5
ALU Inputs: -3 and 20
- Read data 1 reads from register Rs which is 00011. Register r3 contains -3. Read data 2 comes from the ALUSrc MUX which is outputting the sign extended immediate for the sw instruction.

"PC + 4"Adder Inputs: PC and 4
- Imagine that

"Branch" Adder Inputs: PC+4 and 80

- The second input is the sign extended, twice left-shifted immediate. The sign extended immediate was 000…0010100 and the twice left-shifted immediate is 000..1010000 = 80

4.7.6

Read Register 1: 3
Read Register 2: 2
Write Register: 2 or 0

- RegDst is a "don't care"

Write Data: <Memory Read Data signal> or 17

- MemToReg is a "don't care"

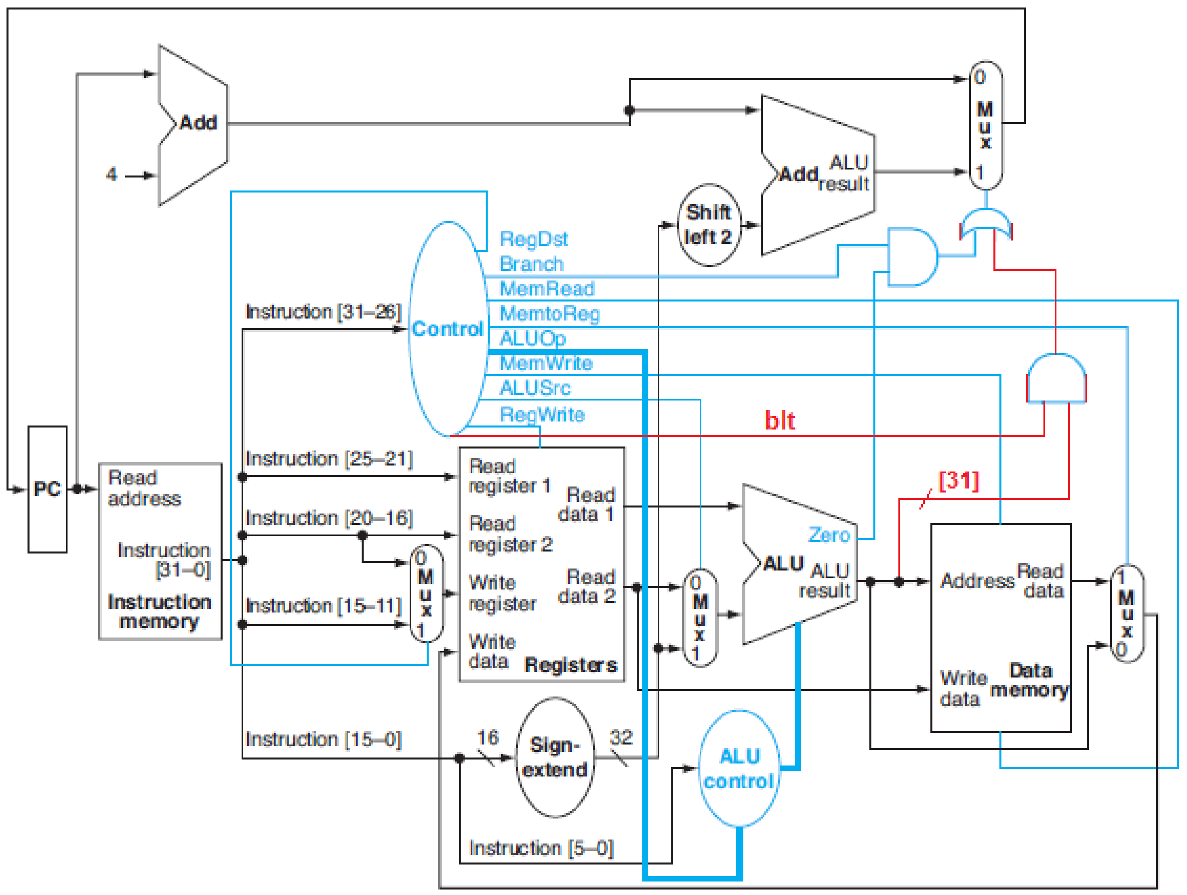
RegWrite: 0

---

2. blt

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

This is similar to the beq instruction except we need to perform a subtraction and branch if R[Rs] is less than R[Rt]. This means we can have the ALU execute a subtraction operation and the output of the ALU will be negative if R[Rs] is less than R[Rt]. One solution is to extract bit 31 of the ALU result (ie. the sign bit which will be 1 if the result is less than 0/if R[Rs] is less than R[Rt]) and AND it with a new blt control signal which will be 1 if the instruction that is being executed is a blt. Since the normal branch logic already allows for choosing between PC + 4 + SE(I) or PC + 4, we can reuse the multiplexer by ORing the ANDed signals for beq and blt together. An alternative solution would be to insert another multiplexer between the Branch MUX and the PC which takes as inputs PC+4 and PC+4+SE(I) whose control signal is blt AND the most significant bit of the ALU output. This new instruction will be an I-Type instruction in order to make use of an immediate field.



Main Controller:
Input Opcode: [Some new I-Type opcode]
Outputs:
  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)
  blt: 1

Another solution for this problem is to let the ALU do SLT operation. The difference in the data path will be the ALU result will output bit 0 instead of bit 31 to the AND gate.

Input Opcode: [Some new I-Type opcode]
Outputs:
  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: 1
ALUOp2: 1 (Have ALU SLT)
blt: 1

Add another column in the ALU Controller table:

Opcode	ALUOp	instruction	function	ALU Action	ALUCtrl
BLT	11	branch less than	xxxx	SLT	111

The difference between the SLT and subtract solution is that using subtract will yield wrong result in terms of comparison if overflow is taken into the consideration. For example, if the R[rs] is -2147483648 (-2^32) and R[rt] is -1 and you use the subtraction to compare them, then the most significant bit will be 0 due to overflow but actually R[rs] is smaller than R[rt]. A correct implementation of the SLT will take the overflow situation into consideration. The subtraction solution is still considered to be correct given that as said in the video lecture, the overflow of the ALU will be ignored.

---

3. jal

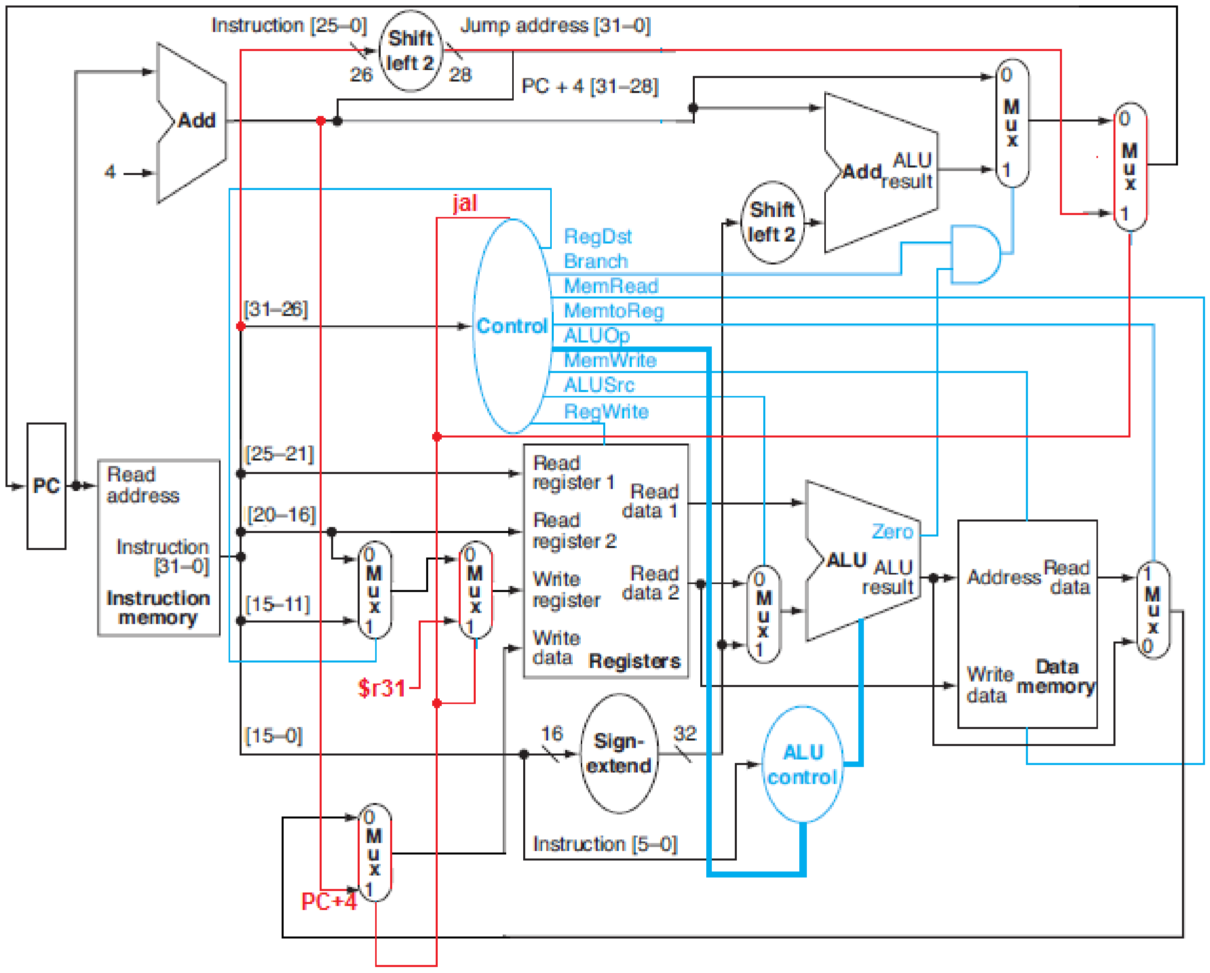
R[$r31] = PC+4
PC = [31..28](PC+4) | [27..0] (I<<2)

First, we need 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 multiplexer before the PC to select 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 multiplexer before write register port of the register file, also controlled by jal.

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

This new instruction will be a J-Type in order to make use the 26-bit address field.



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

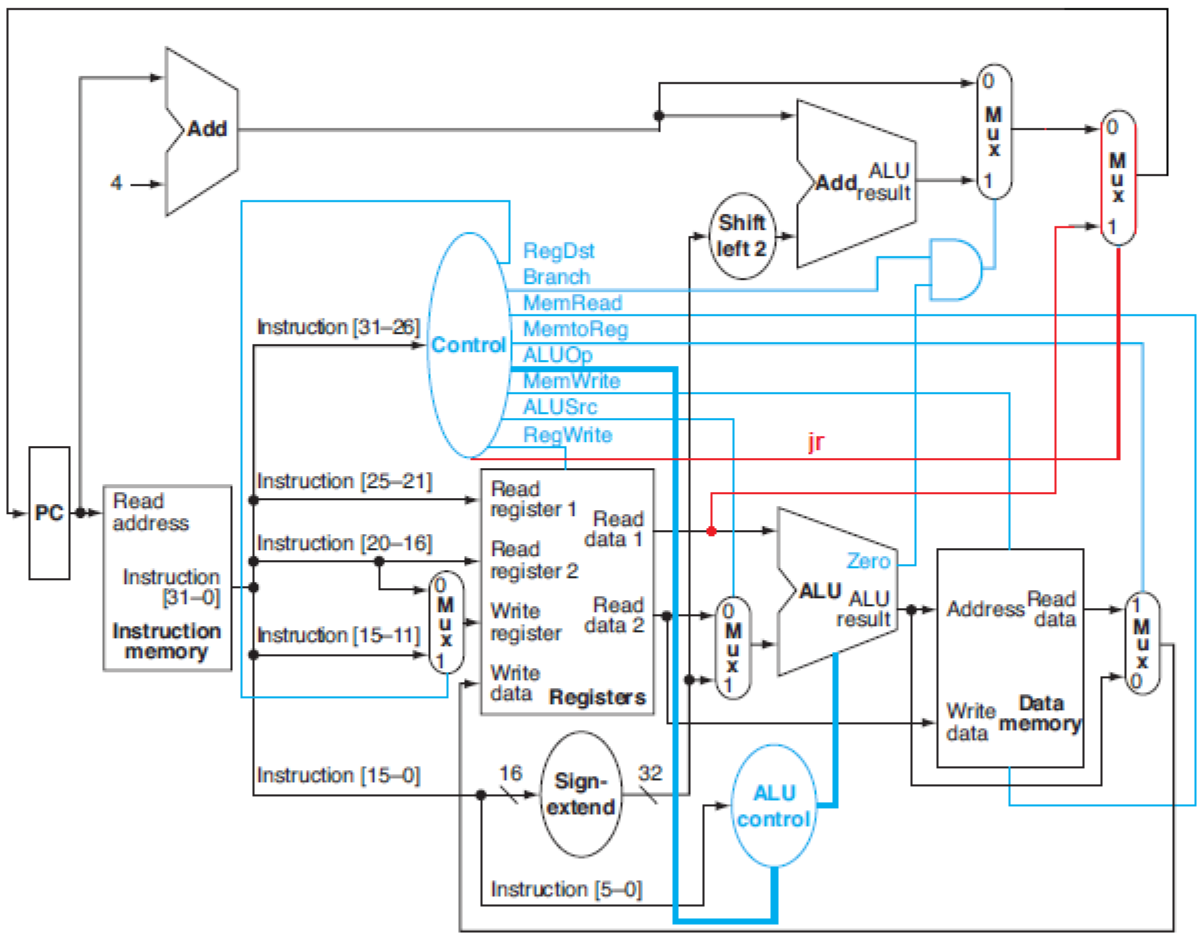
Output:
RegDst: X (overridden by new mux)
ALUSrc: X (not using ALU)
MemtoReg: X (overridden by new mux)
RegWrite: 1 (write to register $31)
MemRead: 0 (don’t read from memory)
MemWrite: 0 (don’t write to memory)
Branch: X (overridden by new mux)
ALUOp1: X
ALUOp2: X (not using ALU)
jal: 1

4. jr

PC = R[rs]

We must be able to select R[rs] as the new PC rather than just 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 multiplexer in front of the PC. Additionally, a new control signal “jr” will be required to control the multiplexer.

This can be implemented as an R-Type instruction or an I-Type instruction, but since R-Type instructions all share opcode 000000, it will be simpler to implement as an I-Type instruction where a new opcode can be defined and the new control signal will be determined by the Control module.



Main Controller:
Input Opcode: [Some new I-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: X (overwritten by new mux)
ALUOp1: X
ALUOp2: X (not using ALU)
jr: 1