(4.7)

Jason Less
404-640-155
CSM151B

CSM151B: HW #4

(4.7) Insn Word ⇒ | 101011 | 00011 | 00010 | 0000 0000 0001 0100 |
op rs rt imm/addr

(4.7.1) • The opcode is 101011 ⇒ which corresponds to an (sw)
• As sw is an I-type, the output of sign-extend can be extracted from the low-order 16-bits of the insn ⇒ 0000 0000 0001 0100
• Sign extending to 32-bits replicates the MSB (which is 0)
⇒ $\boxed{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0100}$
• The jump "Shift left 2":
• The low-order 26-bits are taken and expanded to 28-bits (via a left shift)
⇒ $\boxed{0001\ 1000\ 1000\ 0000\ 0000\ 0101\ 0000}$

(4.7.2) • The ALU Control Unit has 2 inputs ⇒ (1) The 2-bit ALUOp
(2) The lower 6-bits (i.e. funct for R-type)
• As the opcode corresponds to a sw ⇒ the $\boxed{\text{ALUOp} = 00}$
• The lower 6 bits ⇒ $\boxed{010100}$

(4.7.3) • The insn is a sw ⇒ and thus the PC will just advance to the next insn (i.e. PC+4) → $\boxed{PC+4}$
• Path ⇒ (1) PC output
(2) Through the ADD (PC+4)
(3) Through the branch mux (0)
(4) Through the jump mux (0)
(5) Wrap back around to PC input

(4.7.4) • The data path consists of 5 total MUXs
• As it is a sw, the PC will just advance to the next insn (i.e. PC+4)
• Thus, the branch MUX and jump MUX will select 0 (i.e. not branch nor jump)
• $\boxed{\text{Branch MUX} = \text{Jump MUX} = PC+4}$
• It is a sw, so RegDst is a dc, so the output of the Write register MUX could be from insn[20:16] or insn[15:11]
- $\boxed{\text{Write Reg MUX} = 2 \text{ or } 0}$
• ALUSrc is a 1, so the ALU Input MUX comes from the SE(I) = 10100
• $\boxed{\text{ALU Input MUX} = 20}$

4.7.4 (cont)
- The last MUX is the Data Memory correspondent
  - The insn is a sw, so no reading from memory
    - Therefore the output of the MUX is not used/needed
      - $\boxed{\text{Data Memory MUX = not used/needed}}$

4.7.5

<u>ALU</u>
- Input 1 comes from the contents of read data 1
  - Read reg 1 comes from insn[25:21] = 00011 or r3
  - r3 contains the value $\boxed{-3}$
- Input 2 comes from the SE(I) = $\boxed{20}$

<u>ADD(PC+4)</u>
- Input 1 = $\boxed{4}$
- Input 2 = $\boxed{PC}$

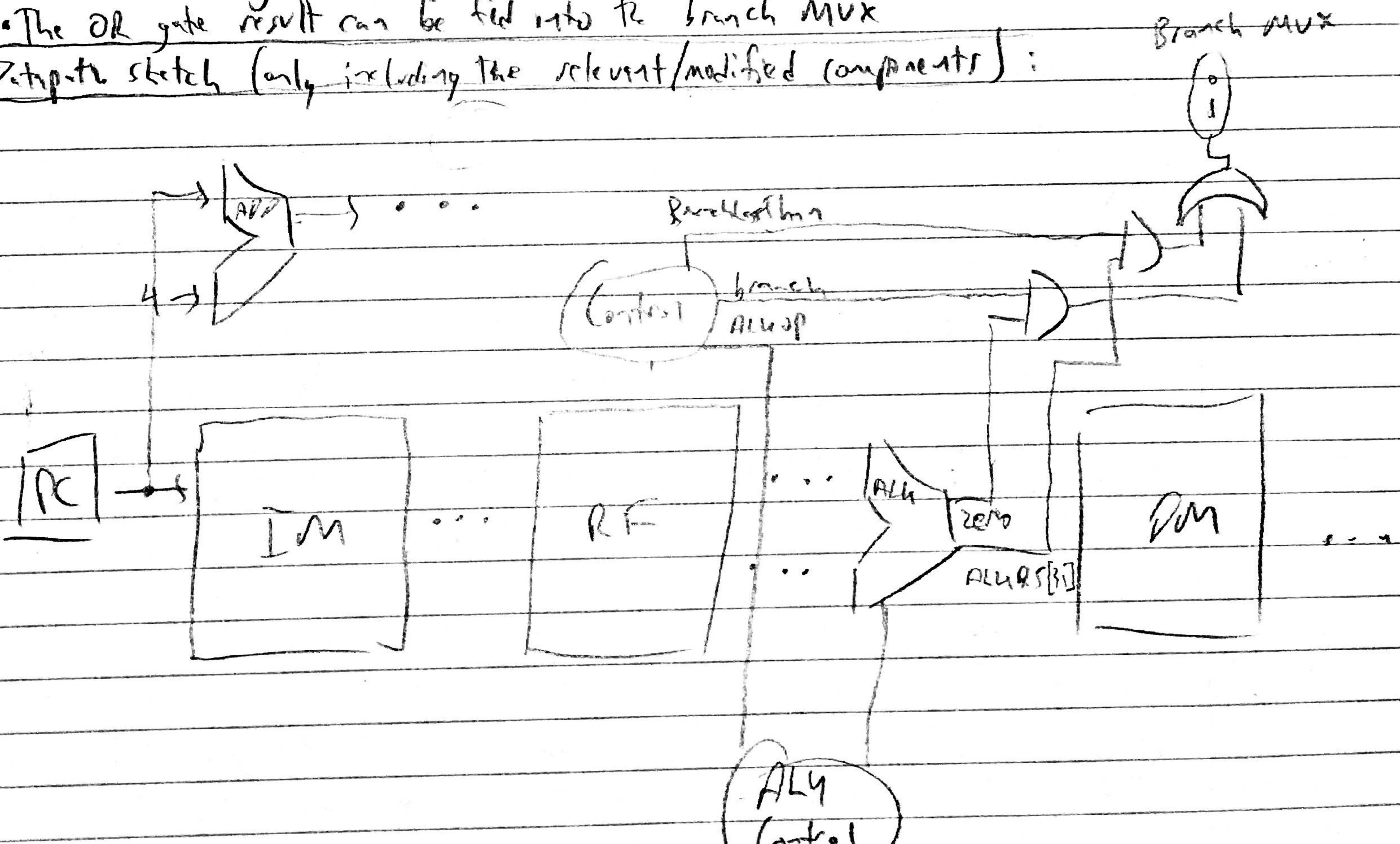
<u>ADD (branch)</u>
- Input 1 comes from ADD(PC+4) = $\boxed{PC+4}$
- Input 2 comes from SE(I) << 2 = 20 << 2 = 20*4 = $\boxed{80}$

4.7.6
- <u>Read reg</u> 1 comes from Insn[25:21] = 00011 = $\boxed{3}$
- <u>Read reg</u> 2 comes from Insn[20:16] = 00010 = $\boxed{2}$
- <u>Write reg</u> comes from Insn[20:16] or Insn[15:11] as RegDst is dc and the MUX can be 0 or 1 => 00010 or 00000 => $\boxed{2 \text{ or } 0}$
- <u>RegWrite</u> = $\boxed{0}$ [As it is a sw insn]
- <u>Write data</u> => insn is a sw, so $\boxed{\text{not writing}}$ anything to the RF

① blt insn ⇒ [I-type] ⇒ if (R[rs] < R[rt]) PC = PC + 4 + SE(I)
else PC = PC + 4

Concept

- This idea is similar to the SLT insn in that we can use the subtraction operation in the ALU to test if R[RS] < R[RT]
  - If R[RS] − R[RT] < 0, then R[RS] < R[RT]
  - We only need to look at the sign bit of the result of the subtraction operation
- Thus, we can do a subtraction op in the ALU and only look at the MSB of the result (or ALURS[31])
  - This result can be combined with a new control signal (say BranchLessThan) to form an AND gate
    - This handles the less than portion
- To handle the branch can use the existing branch control signal, and zero output and create a 2-input OR gate, whose inputs are the outputs of the 2-input AND gates
  - The OR gate result can be fed into the branch MUX
- Datapath sketch (only including the relevant/modified components):

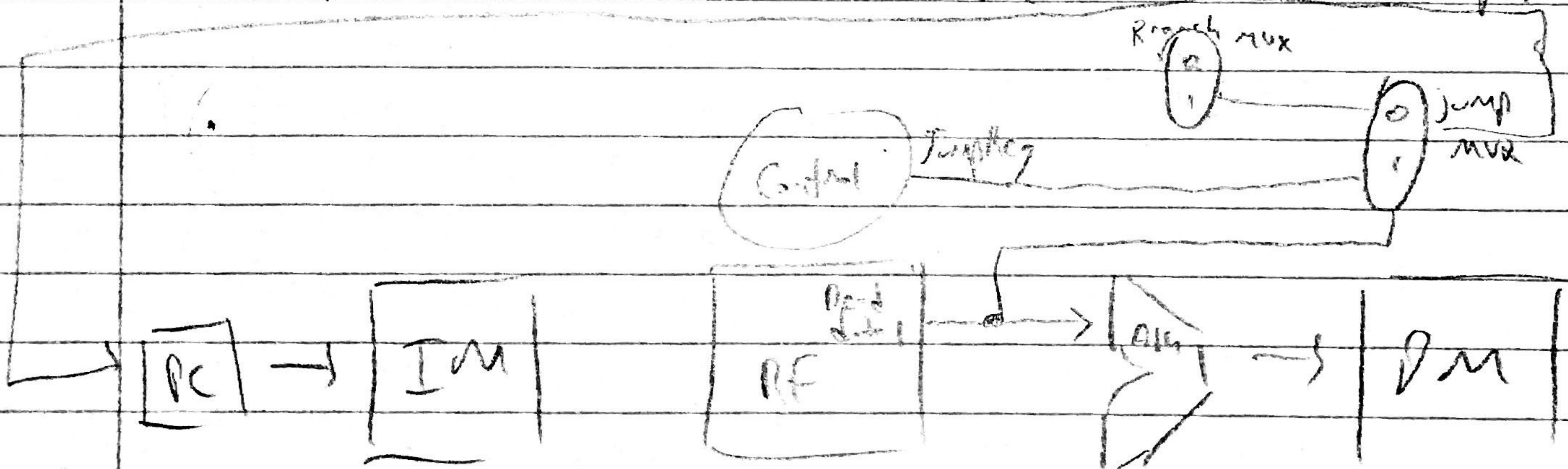


1, cont

Control Outputs	blt
RegDst	X
ALUSrc	0
MemToReg	X
RegWrite	0
MemRead	0
MemWrite	0
Branch	0
ALUOp1	0
ALUOp2	1
BranchLessThan	1

3 jr isn => R-type isn => PC = R[rs]

- This isn is simply setting the PC to R[rs] rather than it being PC+4 or PC+4+SE(I)
- The PC being PC+4 or PC+4+SE(I) is set at the final (upper) jump MUX
- Thus, we still want to utilize the jump MUX (set to 1) in order to jump, but will create a new control signal (say JumpReg) to set the MUX, and then input to the MUX the value R[RS] (from Read data 1 output wire)



Control Outputs	jr
RegDst	X
ALUSrc	0
MemToReg	x
RegWrite	0
MemRead	0

Control Outputs	jr
MemWrite	0
Branch	0
ALUOP1	1
ALUOp2	0
JumpReg	1