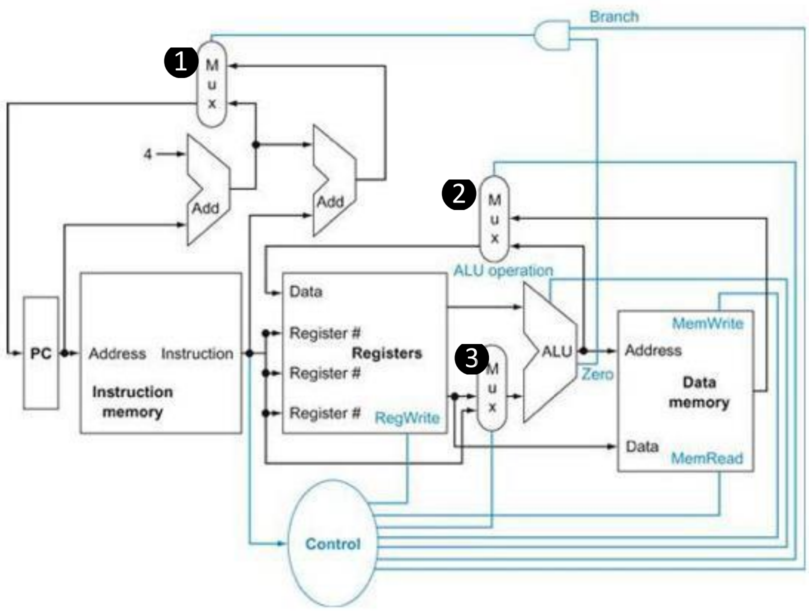
Patrick Woodrum

CPSC 3300 – Homework 3
Due 11:59PM Tuesday, March 9
Submit your answers to Canvas



1. Consider the MIPS “load word” instruction as implemented on the datapath above (Figure 4. from textbook):

   lw R2, 8(R1) // Reg[2] <- memory[Reg[1] + 8]

   Circle the correct value 0 or 1 for the control signals (a-d) and circle whether each of the three muxes (e-g) selects its upper input, lower input, or don't care. For the ALU operation (h) circle one of the function names. (The Zero condition signal will be assumed to be 0.) (20 pts.)

   (a) Branch = 0 1
   (b) MemRead = 0 1
   (c) MemWrite = 0 1
   (d) RegWrite = 0 1
   (e) Mux1 (upper left; output to PC) = upper, lower, don't care
   (f) Mux2 (upper middle; output to Data port of Regs) = upper, lower, don't care
   (g) Mux3 (lower middle; output to bottom leg of ALU) = upper, lower, don't care
   (h) ALU operation = and, or, add, subtract, set-on-less-than, nor

2. Consider the MIPS “store word” instruction as implemented on the datapath above (Figure 4.2 from textbook):

   sw R4, -12(R3) // Memory[Reg[3] + signextended(-12)] <- Reg[4]

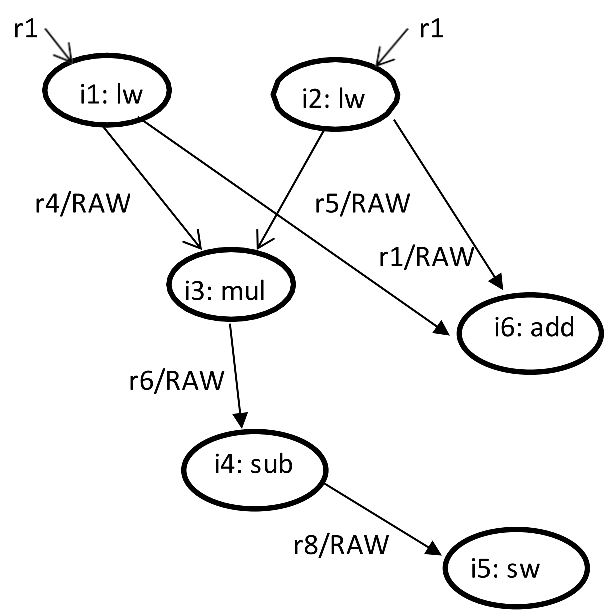
   Circle the correct value 0 or 1 for the control signals (a-d) and circle whether each of the three muxes (e-g) selects its upper input, lower input, or don't care. For the ALU operation (h) circle one of the function names. (The Zero condition signal will be assumed to be 0.) (20 pts.)

   (e) Branch = 0 1
   (f) MemRead = 0 1
   (g) MemWrite = 0 1
   (h) RegWrite = 0 1
   (e) Mux1 (upper left; output to PC) = upper, lower, don't care
   (f) Mux2 (upper middle; output to Data port of Regs) = upper, lower, don't care
   (g) Mux3 (lower middle; output to bottom leg of ALU) = upper, lower, don't care
   (h) ALU operation = and, or, add, subtract, set-on-less-than, nor

Patrick Woodrum

3. For the MIPS instruction sequence below, complete the data dependency diagram. (Destination register is listed first except for sw instruction; sw writes into memory rather than a register.) (15 pts.)

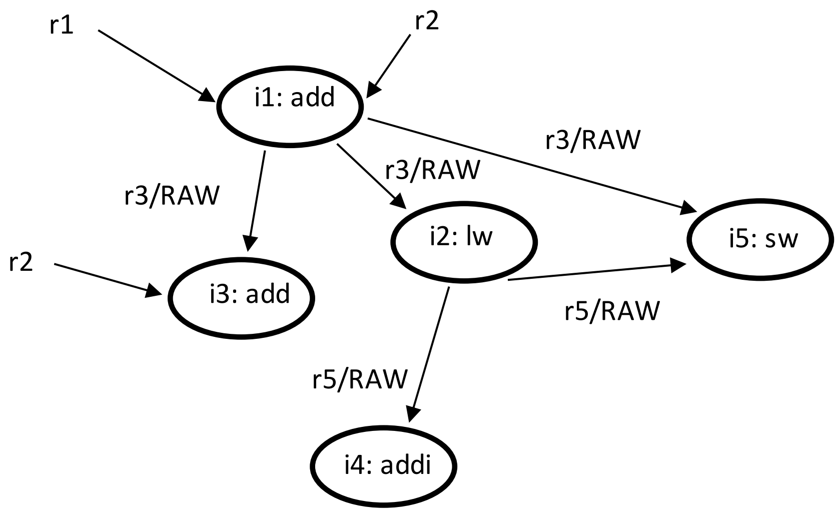
i1: lw r4, 0(r1) // reg[4] ← memory[reg[1] + 0]
i2: lw r5, 4(r1) // reg[5] ← memory[reg[1] + 4]
i3: mul r6, r4, r5 // reg[6] ← reg[4] * reg[5]
i4: sub r8, r6, r7 // reg[8] ← reg[6] - reg[7]
i5: sw r8, 8(r1) // memory[reg[1] + 8] ← reg[8]
i6: add r1, r1, r2 // reg[1] ← reg[1] + reg[2]



Patrick Woodrum

4. Draw the dependency diagram for the following MIPS code (15 pts.)

   i1: add r3, r1, r2 // reg[3] ← reg[1] + reg[2]
   i2: lw r5, 0(r3)   // reg[5] ← memory[reg[3] + 0]
   i3: add r3, r3, r6 // reg[3] ← reg[3] + reg[6]
   i4: addi r5, r5, 1   // reg[5] ← reg[5] + 1
   i5: sw r5, 0(r3)   // memory[reg[3] + 0] ← reg[5]



Patrick Woodrum

5. For the following MIPS instruction sequence, complete the pipeline cycle diagram for the standard 5- stage pipeline <u>without forwarding</u>. Assume register file writes occur in the first half cycle and reads in the second half cycle. (15 pts.)

   i1: lw r1, 0(r5) // reg[1] ← memory[reg[5] + 0]
   i2: add r3, r1, r2 // reg[3] ← reg[1] + reg[2]
   i3: addi r4, r3, 1 // reg[4] ← reg[3] + 1

i1:lw	IF	ID	EX	MEM	WB		
i2:add		IF	ID	EX	MEM	WB	
i3:addi			IF	ID	EX	MEM	WB

6. For the following MIPS instruction sequence, complete the pipeline cycle diagram for the standard 5- stage pipeline <u>with forwarding</u>. Assume register file writes occur in the first half cycle and reads in the second half cycle. (15 pts.)

   i1: lw r1, 0(r5) // reg[1] ← memory[reg[5] + 0]
   i2: add r3, r1, r2 // reg[3] ← reg[1] + reg[2]
   i3: addi r4, r3, 1 // reg[4] ← reg[3] + 1

i1:lw	IF	ID	EX	MEM	WB			
i2:add			IF	ID	EX	MEM	WB	
i3:addi				IF	ID	EX	MEM	WB