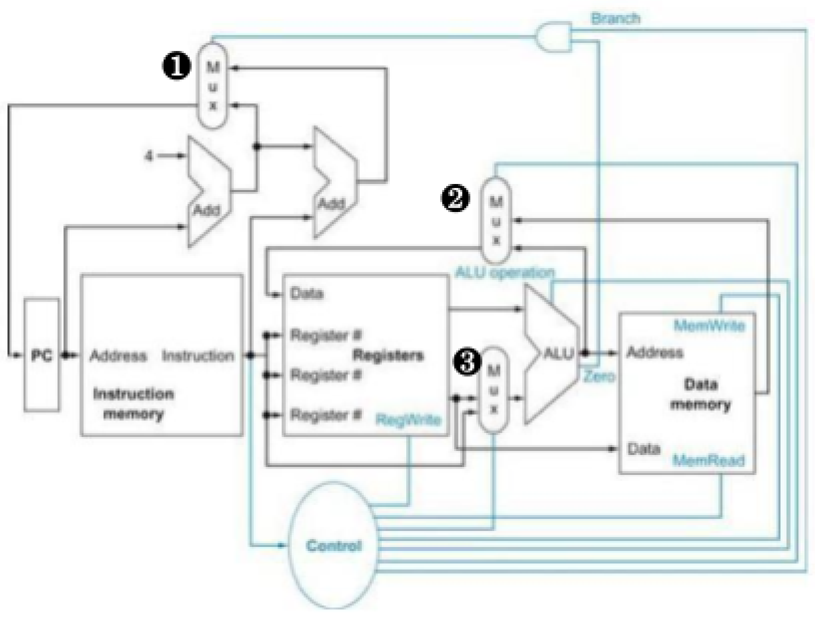
CPSC 3300 – Homework 3

Due 11:59PM, October 8. Using all 5 late days on this. Had something come up :(



1. Consider the MIPS “load word” instruction as implemented on the datapath above (Figure 4.2 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.) (24 pts.)

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

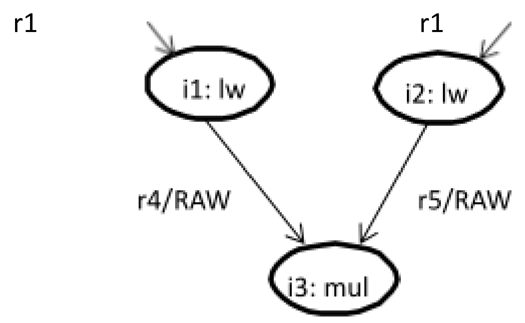
2. Consider the MIPS “store word” instruction as implemented on the datapath above (Figure 4.2 from textbook): sw R4, -12(R3) // Memory[ Reg[3] + sign extended(-12) ] <- Reg[4]

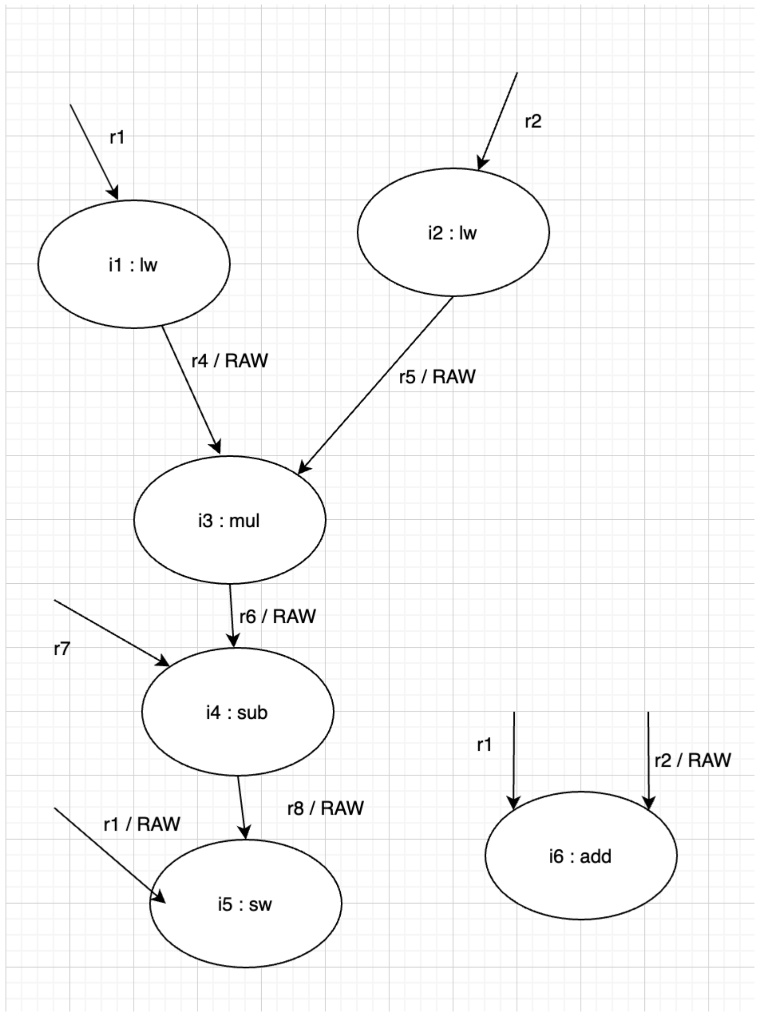
Circle the correct value 0 or 1 for the control signals (a-d) and circle whether each of the three muxes (e-g) selects its upper input, lower input, or don't care. For the ALU operation (h) circle one of the function names. (The Zero condition signal will be assumed to be 0.) (24 pts.)

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

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]
```





4. For the following MIPS instruction sequence, complete the pipeline cycle diagram for the standard 5- stage pipeline without forwarding. Assume register file writes occur in the first half cycle and reads in the second half cycle. (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]
```

Table was too small unless I did this entirely wrong?

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i1: | IF | ID | EX | MEM | WB | | | | | | | | | | | |
| i2 | | IF | ID | EX | MEM | WB | | | | | | | | | | |
| i3 | | | IF | STALL | ID | EX | MEM | WB | | | | | | | | |
| i4 | | | | IF | STALL | STALL | STALL | STALL | ID | EX | MEM | WB | | | | |
| i5 | | | | | IF | STALL | STALL | STALL | STALL | STALL | STALL | ID | EX | MEM | WB | |
| i6 | | | | | | IF | STALL | STALL | STALL | STALL | STALL | STALL | ID | EX | MEM | WB |

5. For the following MIPS instruction sequence, complete the pipeline cycle diagram for the standard 5- stage pipeline with forwarding. Assume register file writes occur in the first half cycle and reads in the second half cycle. (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]
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i1: | IF | ID | EX | MEM | WB | | | | | | | | | | |
| i2 | | IF | ID | EX | MEM | WB | | | | | | | | | |
| i3 | | | IF | ID | EX | MEM | WB | | | | | | | | |
| i4 | | | | IF | ID | EX | MEM | WB | | | | | | | |
| i5 | | | | | IF | ID | EX | MEM | WB | | | | | | |
| i6 | | | | | | IF | ID | EX | MEM | WB | | | | | |