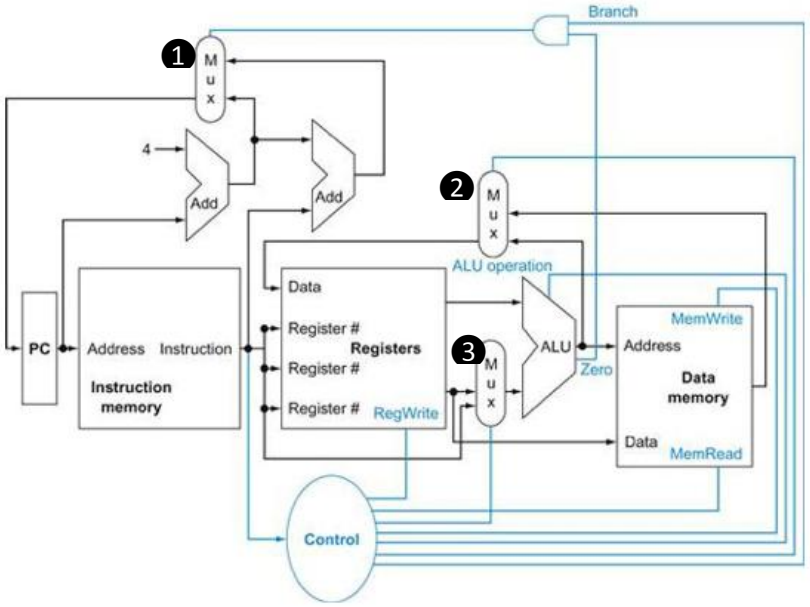
CPSC 3300 – Spring 2018 – Homework 3
Due at class time on Thursday, March 29



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
(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.) (24 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

3. Consider microprogramming the Eckert datapath from Project 2 for a load negative constant instruction, coded as 8zz, where 8 is the hex opcode and zz is an unsigned 8-bit value. The instruction will overwrite whatever is in the ACC and B registers such that the 8-bit value becomes a 12-bit positive value in the B register and a 12-bit negative value in the ACC register. For example, the instruction 801 places 0x001 in B and 0xfff in ACC, and the instruction 8f5 places 0x0f5 in B and 0xf0b in ACC. Note that the instruction must work correctly regardless of the prior values in the ACC and B registers. Note also that the ALU temporary register (i.e., ALU_TMP) is invisible to the machine language programmer, so we do not care what value is placed in it. Show the register transfers needed to execute the load negative constant instruction without making any extensions to the datapath. (26 pts.)

4. For the MIPS instruction sequence below, complete the data dependency diagram. The destination register is listed first except for the sw instruction; sw writes into memory rather than a register. Each dependency arc should be labeled with the register and dependency type: RAW for read-after-write, WAR for write-after-read, and WAW for write-after-write. (26 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]
```