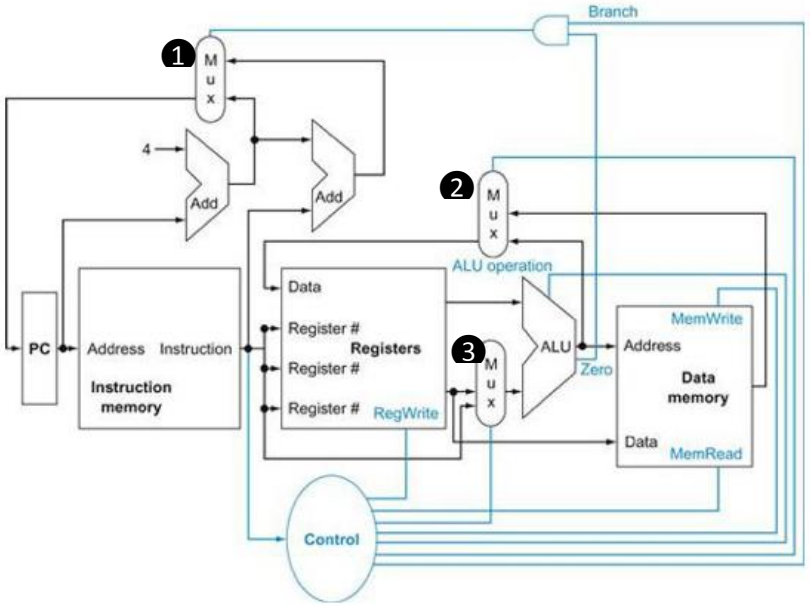
CPSC 3300 - Spring 2015
Homework 5
Due at class time on Monday, March 9



1. Consider the MIPS "and" instruction as implemented on the datapath above (Figure 4.2 from textbook):

```
and   R3, R1, R2   //  Reg[3] <- Reg[1] & Reg[2]
```

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

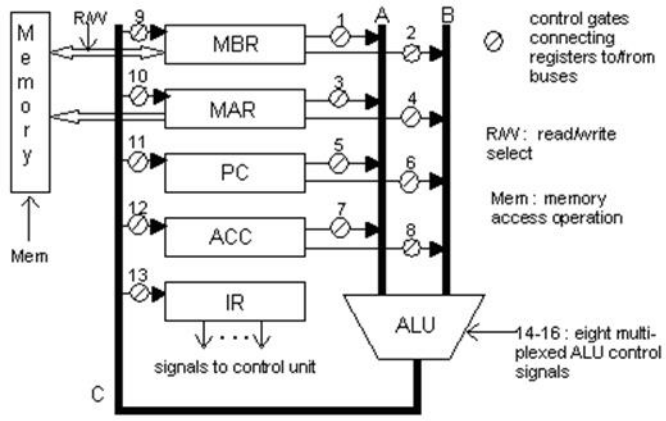
2. Consider the MIPS "sw" instruction as implemented on the datapath above (Figure 4.2 from textbook):

```
sw   R5, 8(R4)   //  Memory[ Reg[4] + 8 ] <- Reg[5]
```

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 the following datapath. (Assume all registers are edge-triggered and thus immune from races.) Control signal identifiers are given for the in and out control points of the registers. Additional control signals include memory signals Mem, R (read), W (write), and 3-bit ALU function field F.




```
ALU functions (three-bit F field)
---------------------------------
000: C = A + B     100: C = A - B
001: C = A         101: C = not A
010: C = A + 1     110: C = A - 1
011: C = A << 1    111: C = A >> 1
```

Complete the step-by-step RTL and the control signal sequence to fetch and execute an increment memory instruction "incr A". Assume that the instruction is composed of two memory words: a one-word opcode followed by a one-word address. Assume also that the address of the instruction is in the PC, and that the memory is word-addressable. The actions of the instruction are memory[A] <- memory[A] + 1, for the memory address A given in the second word of the instruction. (16 pts.)

// fetch opcode and place in IR	// control signals		
MAR <- PC	5 (A=PC),	F=001 (C=A),	10 (MAR=C)
PC <- PC + 1	5 (A=PC),	F=010 (C=A+1),	11 (PC=C)
MBR <- memory[MAR]	Mem/R		
IR <- MBR	1 (A=MBR),	F=001 (C=A),	13 (IR=C)
// fetch operand address and place in MAR			
MAR <- PC	5 (A=PC),	F=001 (C=A),	10 (MAR=C)
PC <- PC + 1	5 (A=PC),	F=010 (C=A+1),	11 (PC=C)
MBR <- memory[MAR]	Mem/R		
MAR <- MBR	1 (A=MBR),	F=001 (C=A),	10 (MAR=C)

4. Using the datapath and control signals for the class example (people.cs.clemson.edu/~mark/uprog.html), show the step-by-step RTL and the corresponding control signal sequence needed to implement a jump to subroutine instruction and a return instruction (i.e., indirect jump). The first word in the subroutine holds the return address. (36 pts.)

```
jsub  -- mem[addr] <- updated pc; pc <- addr + increment        // assume pcincr signal provides increment
jmpi  -- pc <- mem[addr]
```

Use of these instructions in assembly code would look like:

```
start:  load  a
        jsub  subr
        store c
        halt
subr:    .word 0          // reserved for return address
        sub   b
        jmpi  subr
```