Assignment 6

Alex Clemmer
Student number: u0458675

1 Problem 1:

	RegDst	*Branch*	*MemRead*	*MemtoReg*	*ALUOp*	*MemWrite*	*ALUSrc*	*RegWrite*
`ori`	0	0	0	0	`or`	0	1	1
`add`	1	0	0	0	`add`	0	0	1
`lw`	0	0	1	1	`add`	0	1	1
`sw`	x	0	0	x	`add`	1	1	0
`beq`	x	1	0	x	`sub`	0	0	0

2 Problem 2:

Here we add several things. First, we branch off immediately after fetching the instruction, and shift the lower 26 bits of the instruction left 2 bits. We then concatenate the upper 4 bits of the PC+4 and place those as the most significant bits in the instruction.

Why? In MIPS, `jal` is by offset, so we need to put the top of the PC onto the jump location. The low order bits are always a `00` for a jump.

A new mux should then mux this result with whatever the output of the branch/increment mux is, and the result should be that we end up routing this "new" address to the PC. To manage this new mux, we add a control line called "jump".

Also worth noting is that immediately to the left of the register file is a new mux that muxes the result of the mux that was already there with the value 31. This is provided because `jal` can't provide fields that specify the `$ra`. When this is selected, we are telling the register file that we want to write to register 31.

From there, we will proceed through the control path normally until we are ready to write back to the register. You will notice at the bottom is another mux, which muxes the output of the mux that determines whether we write the result of the ALU or memory to an address. This mux determines whether we will write that output, or the current PC+4.

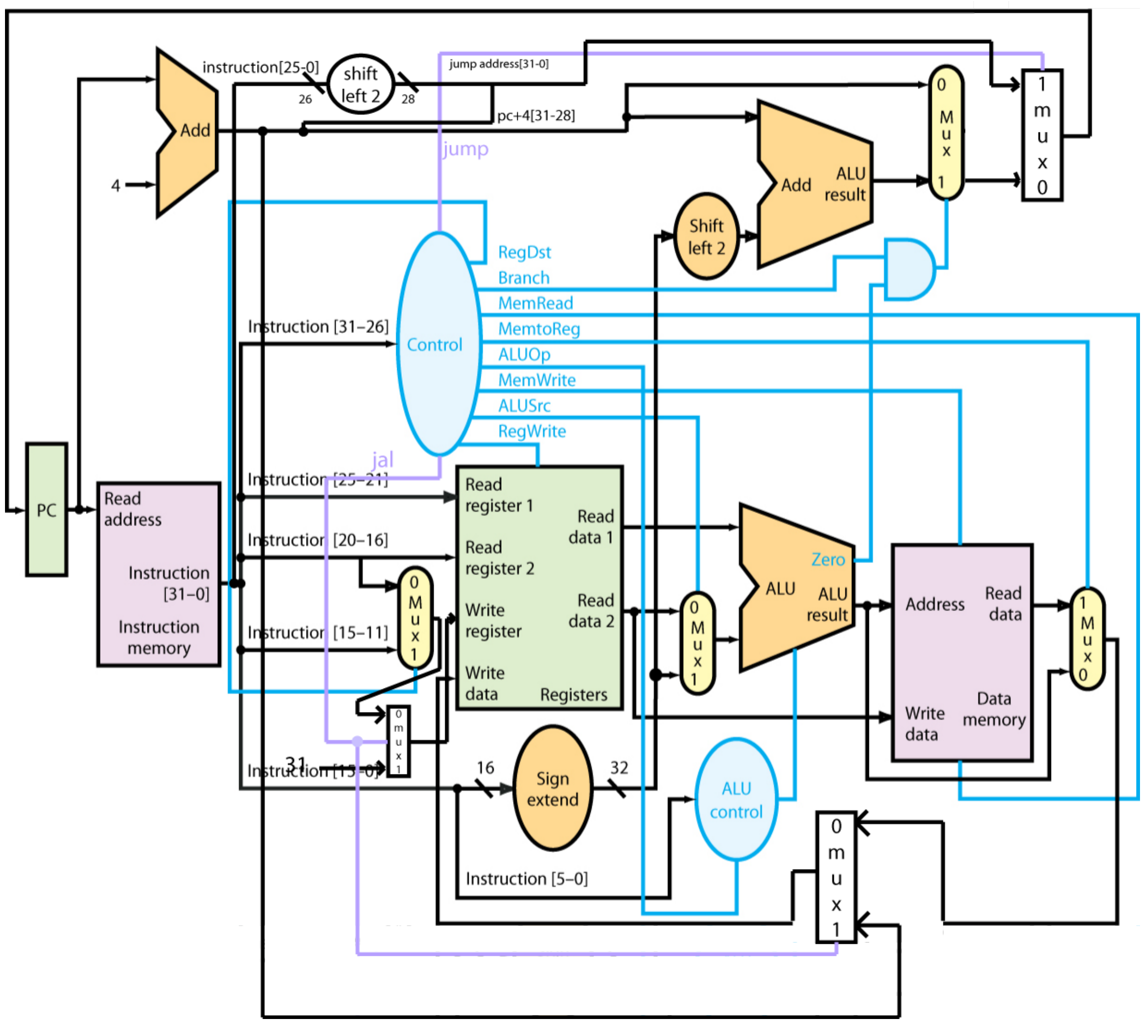
For the record, I chose to implement `jump` and `jal` separately, but we would also need to implement `jr` to use `$ra` to jump back. Oops!

One more thing: A `jal` call would look like this.

RegDst	*Branch*	*MemRead*	*MemtoReg*	*ALUOp*	*MemWrite*	*ALUSrc*	*RegWrite*	*jump*	*jal*
x	x	x	x	x	0	x	1	1	1

It's helpful to note that, strictly speaking, we require none of those fields to be filled in, but they should be filled in and NOT accessed unless necessary.

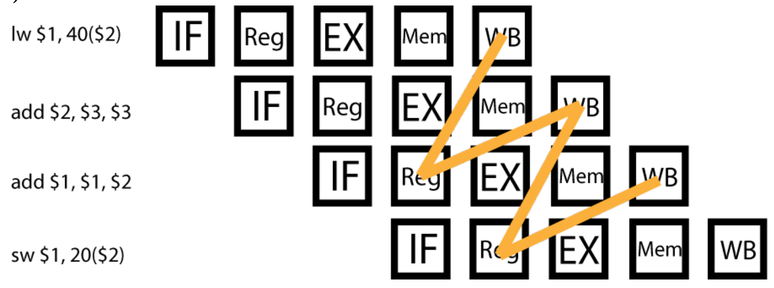
(picture on next page)



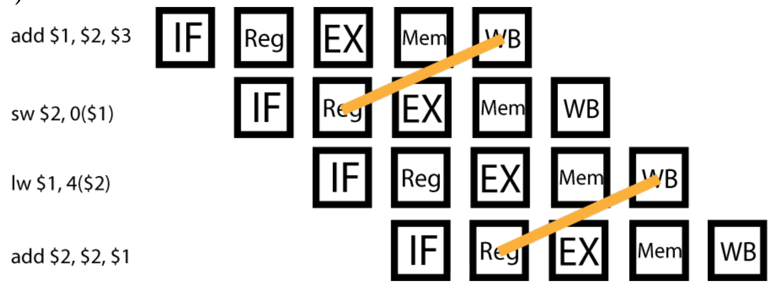
3 Problem 3:

Both of these are pretty straightforward:

(a)



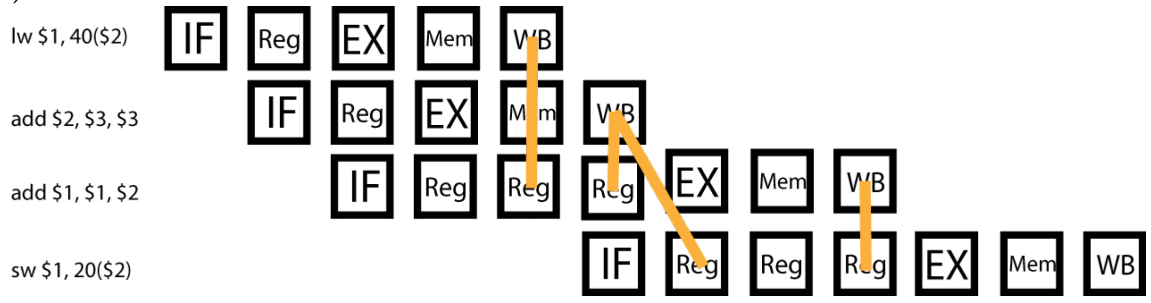
(b)



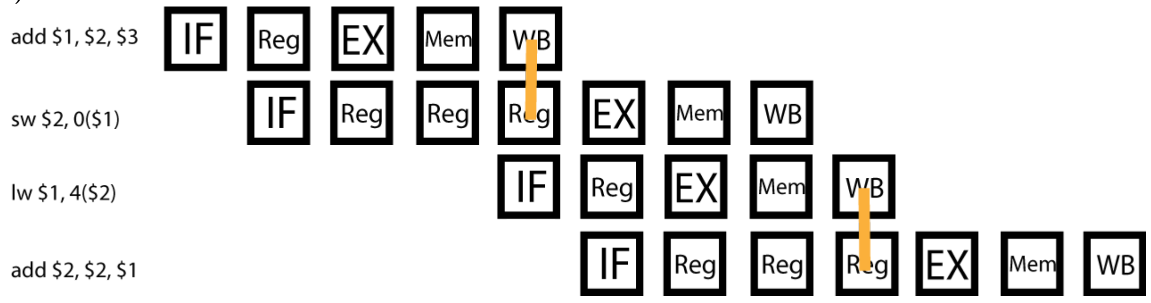
4 Problem 4:

These, once again, are pretty straightforward:

(a)



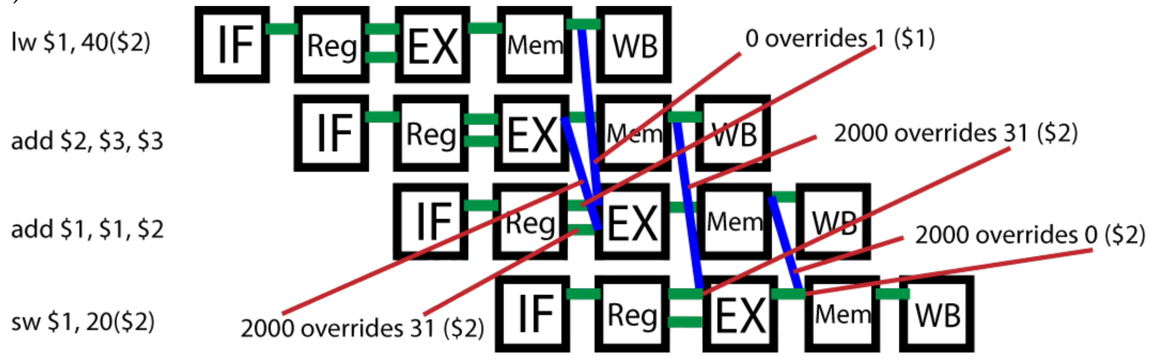
(b)



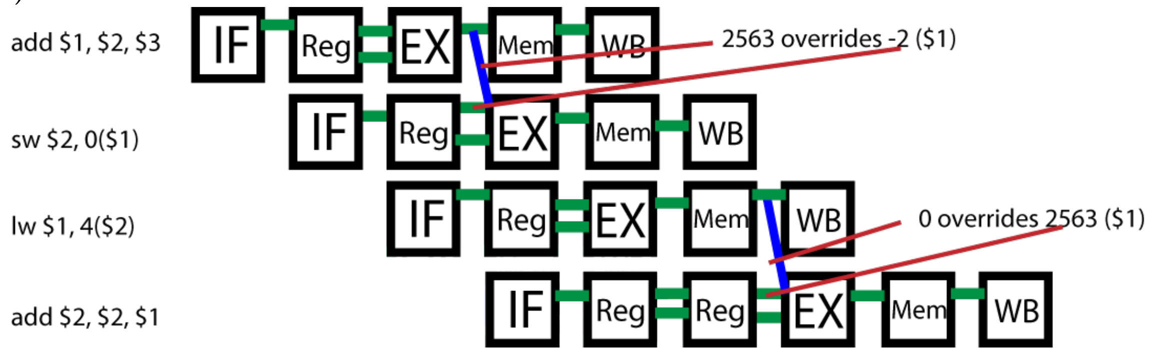
5 Problem 5 and 6:

NOTE that I've combined them into one diagram for simplicity.

(a)



(b)


add $1, $2, $3
sw $2, 0($1)
lw $1, 4($2)
add $2, $2, $1
IF
Reg
EX
Mem
WB
2563 overrides -2 ($1)
0 overrides 2563 ($1)