Take Home Quiz I
Voluntary and Collaborational

Due: On or before Wednesday, November 14, 4:00 PM (in class)

This quiz requires you to execute the code DLX code fragment on the next page using a variety of assumptions regarding the treatment of control hazards. The code is to be executed using the following modification of a Multicycle DLX *pipelined* machine, similar to that presented in section 3.7.

Functional Units: There are separate computational pipes for integer and floating point ALU computations. Except for the otherwise indivisible divide, all floating point units are fully pipelined, meaning one stage per clock cycle. The floating point divide takes 25 clock cycles; the floating point adder/subtractor takes 4 clock cycles, and, the floating point multiply unit takes 7 clock cycles.

Registers: Register **writes** occur in the first half of the clock cycle, and register **reads** occur in the 2nd half of the clock cycle.

Memory: A Harvard Architecture is assumed, in which separate instruction and data memories are used. A perfect cache is also assumed.

Latencies: Latencies correspond to those given in section 3.7.

Structural Hazards: Should two ALU instructions complete their EX stages simultaneously and both require the MEM stage, the instruction that was started earlier gets the MEM stage, and the other instruction stalls in its last stage prior to the MEM stage.

RAW Hazards: Assume that normal forwarding, bypassing, and load interlocks are implemented to minimize stalls due to RAW hazards. Stalls for RAW hazards occur after the ID stage, until the pipeline can proceed safely without any further delays to prevent this hazard.

WAW Hazards: The 2nd instruction involved in a potential WAW hazard is stalled after the ID stage, waiting for the hazard to clear.

Control Hazards: Each problem modifies the problem specification or design assmptions associated with the pipeline's treatment of branches. So, make sure that you indicate your assumptions clearly when doing your work to maximize your partial credit.

Formatting your work.

To make life easy, use the columnar format below, indicating the clock cycles spent by each instruction in each stage, and explaining any stalls that occur.

Instruction	IF	ID	EX	MEM	WB	Comments
LF F8, 0(R6)	1	2	3	4	5	
LF F9, 4(R6)	2	3	4	5	6	
ADDF F10, F8, F9	3	4-5	6-9	10	11	Stall for RAW hazard with F9
MULTF F20,F10, F10	4-5	6-9	10-16	17	18	Stall for RAW hazard with F10
SF 8(R6), F20	6-9	10-16	17	18	19	Stall for RAW with F20 in ID stage.

The DLX Code Fragment

```
        ADDI     R4, R0, # 5200   ; make a float 5200
        MVI2FP   F4, R4
        CVTI2FP  F4, F4           ; F4 has a float constant
        ADD      R1,R0,R0         ; init counter to 0
Loop:   LF       F2,100(R1)       ; F2 is array element,
                                  ; R1 has offset of lowest unused array element
        LF       F3,500(R1)       ; F3 holds array element
        SUBF     F5,F3,F2         ; perform subtraction
        ADDF     F5,F5,F4         ; perform addition of a constant
        SF       1000(R1),F5      ; store the result
        ADDI     R1,R1,#4         ; increment pointer
        SUBI     R5,R1,#400       ; check pointer
        BNEZ     R5,Loop          ; branch while not done
        STOP
```

Problem 1

Compute the CPI for one iteration of the loop assuming perfect branch prediction. That is, there should be no stalls due to branching in your calculations. your analysis. Use this to estimate the total CPI for the loop, again assuming perfect branch prediction.

Problem 2

Again, compute the CPI for one iteration of the loop, but this time assume that the machine stalls until the target is known, using a branch penalty of 2 clock cycles before the correct next instruction can be selected. Use this to estimate the total CPI for the loop.

Problem 3

This time, unroll the loop four times and reschedule to minimize stalls, except that this time, a two-instruction branch delay slot is used. One more time, compute the CPI for one iteration of the loop, using a two-instruction branch delay slot. (And, if you can't find two instructions to execute safely after the branch, using a no-op is allowed.)

Extra Credit, Dude:

Now, this is simpler than it looks, and does NOT require calculators.

Which is bigger?

$$(10)^{\frac{1}{10}} or 2^{\frac{1}{3}}$$

And, yes the calculator will tell you the answer. But, only algebra II is required to prove this, and certain no worse than calculus-like reasoning.