CS 6810 Homework #2

Due: 9:10 a.m. Sept. 22 (no late submissions will be graded)

General instructions: You will hand in your homework's at the beginning of class in paper form. That way the TA can make comments during the grading process. Note that if we can't read your writing then you will be penalized. Similarly, if you just write the answer without supporting it with your reasoning, you will be penalized. Hence you are encouraged to type your answers where this makes sense. Note this is often not easy when doing algebraic calculations unless you have a good function editor, or use MatLab or whatever. The key is that your solutions must be readable in order to receive credit. Organize your work accordingly. The general intent is to make the problem statement clear but there will be times when this goal unfortunately isn't met. Clarifications can be sought by email but if you're in a middle of the night scramble the chances of an instant answer are low. If you make reasonable assumptions and clearly state them then you won't be penalized. "Googling for answers" is unfortunately a common practice, and given the broad use of your textbook in other universities it is not unlikely that you will find problems and answers similar to these somewhere. If you rely on this mechanism you will likely lose when it comes to exam time. Make sure that what you turn in is your own work – see the cheating policy on the class web page. However you are encouraged to discuss solution strategies with your classmates, just make sure that you solve the problems on your own.

Problem 1: [10 points] Pipeline Efficiency

Consider two versions of the pedagogical 5-stage MIPS pipeline:
uMIPS – the unpipelined version
pMIPS – the pipelined version

Assuming an L1 hit for both cache and data each of the 5 stages takes 20 ns – namely the latency to complete any single instruction on the uMIPS is 100ns. The pMIPS is built by adding pipeline registers between the stages. The propagation delay for these registers is 3ns, set-up time is 1 ns, and hold time is 1 ns.

1.1 [2 points] In a perfect world where there are no stalls how much faster is pMIPS than uMIPS?
1.2 [2 points] Branch instructions occur 12% of the time. The pipeline organization of pMIPS is such that there is only one branch delay slot, and the slot is utilized 33% of the time due to either compiler or dynamic hardware mechanisms. How much slower is this pMIPS situation over the idealized world in problem 1.1?
1.3 [5 points] Now consider the effect of data hazards on pMIPS performance. Loads occur 8% of the time in the workload, and instruction reordering only

works 50% of the time to eliminate the one cycle stall that will result if there is a forwarding path from MEM to EX. 60% of the time instructions are simple REG-REG instructions and 4% of the time a REG-REG consumer instruction (preceded by the REG-REG producer instruction) must be stalled since there is no forwarding path for this case. How much worse is this pMIPS case from the idealized pMIPS case in problem 1.1

1.4 [1 point] Givien that uMIPS has a normalized performance of 1, what is the normalized performance against uMIPS, for the pMIPS under control hazards as specified in problem 1.2. Do the same projection for the 1.3 restrictions, and finally for a pMIPS under both 1.2 & 1.3 considerations.

Problem 2: [50 points] Pipeline Efficiency

It's impossible to pass up the chance to learn from Bob Colwell (formerly a chief architect at Intel, now retired, and always a smart but good guy) so this question is a slight variation of the Chapter 2 case study in your book. The conditions expressed in the initial paragraph on page 142 of your text still apply.

Consider the following code sequence, assume all registers that are not written to by code in the loop already have been initialized long enough before loop entry to not create stalls:

```
Loop:  L.D      F2, 0(R9)
IO:    DIVD     F8, F2, F0
I1:    MULTD    F2, F6, F2
I2:    L.D      F4, 0(R10)
I3:    ADDD     F4, F0, F4
I4:    ADDD     F10, F8, F2
I5:    ADDI     R9, R9, #8
I6:    ADDI     R10, R10, #8
I7:    S.D      F4, 0(R10)
I8:    SUB      R20, R4, R9
I9     BNZ      R20, Loop
```

Where the load-use latencies (e.g. the number of intervening instructions between producing and consuming instruction) are:

```
L.D    +4
S.D    +1
SUB    +0
ADDI   +0
BNZ    +1
ADDD   +1
MULTD  +5
```

DIVD +12

2.1 [5 points] Assuming that the loop runs forever, with no instruction reordering, in a 5 stage MIPS pipeline, how many cycles does it take per loop iteration?

2.2 [5 points] Assume that this code is the only thing that runs and the loop runs for 20 iterations, and you should only re-order/schedule the code to fill the branch delay slot if possible. What is your new code sequence? How many cycles does it take to complete the 20 iterations?

2.3 [10 points] Assume you can fetch & decode 2 instructions per cycle and that you have 2 full execution pipelines, each capable of executing any instruction, but only a single MEM stage so only one S.D or L.D instruction can be handled in this stage on any 1 cycle. In WB you can write to any 2 different registers in the same cycle. Use the above code sequence without instruction reordering. Your architecture is fully forwarded both between pipelines and within each pipeline. Assuming an initially empty pipeline and that the loop will happen more than 3 times, after which cycle does the initial LD F2, 0(R9) instruction complete the execution stage for the third time? [Hint – this instruction first completes IF after cycle 1.] NOTE: since bookkeeping is important here make sure you clearly show how you are keeping track of cycles.

2.4 [5 points] Problem 2.3 has some unrealistic assumptions and one very constraining assumption – describe why this is true?

2.5 [25 points] Now consider a different version of the problem in 2.3. The same restrictions apply for IF, ID, and MEM as in 2.3 but for both EX and WB the resources for floating point and integer operations are separate meaning at any cycle 1 FP and 1 Int can be concurrently in the EX stage, and in the WB stage 2 concurrent register writes can be done as long as at most one of them is has an integer register target and at most one of them has a FP register target. Also you are allowed to reorder the instructions by moving at most 3 instructions in the original code to make your code run faster. A ssuming an initially empty pipeline and that the loop will happen more than 3 times, after which cycle does the initial instruction of your loop complete the execution stage? Again. be sure to clearly show your work.

Problem 3: [40 points] More complex instruction scheduling.

You've been given the job of optimizing a scientific application on a new RISC microprocessor architecture. The architecture is fully pipelined (including the FPU), but hazards can still arise from data-dependencies with penalties described below:

Producing Instruction	Using Instruction	Latency (stalls)
FPU Op	FPU Op	+3
FPU Op	Store	+2
Load	FPU or ALU Op	+1
FPU Compare	BFP{T or F}	+1
FPU Op	FPU Compare	+3

All branches (jumps, calls, condition branches) have a single delay slot. The majority of the application time is spent executing the following code fragment which takes two lists of data-points and calls some function foo() for all pairs of points that are closer than some specified distance max.

```
for (i = 0; i < n1; i++)
  for (j = 0; j < n2; j++) {
    dx = x[j] - x[i];
    dy = y[j] - y[i];
    r2 = dx*dx + dy*dy;
    if (r2 < max)
      foo();
  }
```

The following registers have been preloaded: R2 = &x[0], R3 = &y[0], R4 = n1, R5 = n2, and F30=max. In the following machine code you will notice some differences in what you're used to in terms of MIPS 64 all of the floating point registers are used as even numbers indicating doubles and the opcodes don't have the MIPS64 D at the end of the mneumonic. LTD – less than double is a FP compare for less than which sets a FP predicate which is used for branching similar to the BNEZ and BEQ instructions that you're familiar with – for example BFPF branches if the FP predicate is false.

The machine code is:

```
     ADD R6, R2, R0 ; R6 = &x[0]
     ADD R7, R3, R0 ; R7 = &y[0]
loop1: ADD R8, R2, R0 ; R8 = &x[0]
     ADD R9, R3, R0 ; R9 = &y[0]
     ADD R10, R5, R0 ; R10 = n2 (j loop counter)
loop2: LD F0, 0(R8) ; load x[j]
     LD F2, 0(R6) ; load x[i]
     FSUB F0, F0, F2 ; F0 = x[j]-x[i]
     LD F4, 0(R9) ; load y[j]
     LD F6, 0(R7) ; load y[i]
     FSUB F4, F4, F6 ; F4 = y[j]-y[i]
     FMUL F0, F0, F0 ; dx*dx
     FMUL F4, F4, F4 ; dy*dy
     FADD F0, F0, F4 ; r2 = dx*dx + dy*dy
     LTD F0, F30 ; r2 < max (sets FP status bit).
(*)  BFPF inc2 ; FP status false then skip this next
part
     NOP
     CALL foo ; Call foo();
     NOP
inc2: ADDI R8,R8,#8 ; increment x[j] pointer
     ADDI R9,R9,#8 ; increment y[j] pointer
     SUBI R10, R10, #1
     BNZ R10, loop2 ; continue with j loop
     NOP
     ADDI R6,R6,#8 ; increment x[i] pointer
     ADDI R7,R7,#8 ; increment y[i] pointer
     SUBI R4,R4,#1 ; decrement i counter
     BNZ R4, loop1
     NOP
```

Assume the branch location at * is taken 65% of the time and ignore the number of cycles that are executed in the function foo() for now.

3.1 [8 points] How many integer instructions, memory instructions, control instructions and floating point instructions are executed by this code fragment for n1=n2=100?

3.2 [8 points] Identify all of the hazards in the code and describe how many stall cycles result from each hazard.

3.3 [4 points] How many cycles are spent executing this code? What percentage of these cycles is caused by stalls?

3.4 [20 points] Optimize the code as much as possible by unrolling the inner loop 2x, rescheduling instructions to eliminate hazards, filling branch delay slots, and eliminating redundant instructions. [Hint – unless you are a wizard, this could be tricky so look VERY carefully at the options.]