## Assignment 4

**Due: 1:25pm, Wed Sept 28th, 2022**

Note: Make reasonable assumptions where necessary and clearly state them. Feel free to discuss problems with classmates, but the only written material that you may consult while writing your solutions are the textbook and lecture slides/videos. Solutions should be uploaded as a single pdf file on Canvas. **Show your solution steps** so you receive partial credit for incorrect answers and we know you have understood the material. Don't just show us the final answer.

Every homework has an automatic penalty-free 1.5 day extension to accommodate any covid/family-related disruptions. In other words, try to finish your homework by Wednesday 1:25pm to keep up with the lecture content, but if necessary, you may take until Thursday 11:59pm.

1. **Loop unrolling and SW pipelining**

   Consider a basic in-order pipeline with bypassing (one instruction in each pipeline stage in any cycle). The pipeline has been extended to handle FP add. Assume the following delays between dependent instructions (note: these assumptions are different from the examples discussed in class):

   - Load feeding any instruction: 2 stall cycles
   - FP MUL feeding any instruction (except stores): 6 stall cycles
   - FP MUL feeding store: 5 stall cycles
   - Int add feeding a branch: 1 stall cycles
   - Int add feeding any other instruction: 1 stall cycle
   - A conditional branch has 1 delay slot (an instruction is fetched in the cycle after the branch without knowing the outcome of the branch and is executed to completion)

   Below is the source code and default assembly code for a loop.

   ```
   Source Code:
   for (i=1000; i>0; i--) {
   w[i] = x[i] * w[i];
   }

   Assembly Code:
   Loop:
   L.D F2, 0(R1) // Get w[i]
   L.D F4, 0(R2) // Get x[i]
   MUL.D F6, F2, F4 // Multiply two numbers
   S.D F6, 0(R1) // Store the result into w[i]
   DADDUI R1, R1, #-8 // Decrement R1
   DADDUI R2, R2, #-8 // Decrement R2
   BNE R1, R3, Loop // Check if we've reached the end of the loop
   NOP
   ```

   1. Show the schedule (what instruction issues in what cycle) for the default code. (15 points)
   2. How should the compiler order instructions to minimize stalls (without unrolling)(note that the execution of a NOP instruction is effectively a stall)? Show the schedule. How many cycles can you save per iteration, compared to the default schedule? (15 points)
   3. What is the minimum unroll degree to eliminate stall cycles? Show the schedule for the unrolled code. (20 points)
   4. Come up with a software-pipelined version of the code (no unrolling). Use appropriate register names and displacements in your code. When this code executes, will it experience any stalls? (20

points)

2. **Branch predictors (30 points)**

   Consider the following tournament branch predictor that employs a selector with 64K entries (2-bit saturating counters). The selector picks a prediction out of either a global predictor (16-bit global history is XOR-ed with 16 bits of branch PC to index into a table of 3-bit saturating counters) or a local predictor (8 bits of branch PC index into level-1, 11 bits of local history from level-1 are concatenated with 4 other bits of branch PC to generate the index into level-2 that has 3-bit saturating counters). What is the total capacity of the entire branch prediction system?

---