

1. (50 points) **Tomasulo's Algorithm/Superscalar/Speculation.** In this exercise, we will look at how variations on Tomasulo's algorithm perform when running a common vector loop. Assume the following code fragment:

```
loop:  L.D F2, 0(R1)
      L.D F0, 0(R4)
      MULT.D F4, F2, F0
      LD.D F6, 0(R2)
      ADD.D F6, F4, F6
      S.D 0(R2), F6
      DADDUI R1, R1, #8
      DADDUI R2, R2, #8
      DADDUI R4, R4, #8
      DSGTUI R3, R1, exit
      BEQZ R3, loop
```

Effectively, the above code fragment computes $Y_i = A_i X_i + Y_i$, where A , X and Y are vectors of length 100.

Here are some assumptions/clarifications:

- Initially, R1, R2 and R4 contain the start address of X , Y and A , respectively.
- You may assume that the three vectors are non-overlapping.
- The instruction mnemonic DSGTUI stands for “set if greater than unsigned immediate.” In particular, DSGTUI R3, R1, exit does the following: set R3 to one if the value in R1 is greater than the immediate constant “exit”; otherwise, set R3 to zero. The next instruction then checks the value of R3 and branches to the beginning of the loop if R3 has been set to zero. Note: “exit” is a constant integer.
- Function units are not pipelined in part (a), but are fully pipelined (*i.e.*, can accept a new set of operands every clock cycle) in parts (b) and (c).
- The pipeline is single-issue for part (a), but two-issue for parts (b) and (c).
- There is no forwarding between function units; results are communicated via the CDB.
- The execution stage (EX) does both the effective address computation and the memory access for loads and stores. Thus, there is no explicit memory stage (MEM); instead, the pipeline is: Fetch (IF), Decode (ID), Issue (IS), Execute (EX), and CDB Write. Each of these pipeline stages, except for the EX stage, takes 1 clock cycle.
- For the EX stage, assume the following latencies: Integer operations take 1 cycle; FP adder takes 4 cycles; FP multiplier takes 15 cycles. What this means is that, if an FP addition operation is issued on cycle 1, it completes execution on cycle 5, and its result may be used on cycle 6.
- There are five load buffer slots and 5 store buffer slots.
- Assume that at most one instruction can write the CDB in one clock cycle.
- For architectures with reorder buffer, assume at most one instruction can commit per clock cycle.

- (a) For this part, assume the single-issue Tomasulo pipeline. Run this machine for two iterations of the loop, and show the execution as a space-time chart (with a row for each instruction, and columns showing which clock cycles are spent in doing IF, ID, IS, EX and Write phases of its execution.
- (b) Repeat, but this time assume the FP add and FP multiply operations are fully pipelined, *i.e.*, they can admit a new pair of operands to enter each clock cycle. Once again show the execution using a space-time chart.
- (c) Now assume a Tomasulo machine *with speculation* (Fig 2.9 on Page 94 of textbook). Assume branches are predicted as taken. Show the execution using a space-time chart.

2. (30 points) **Multiple-Issue: VLIW.** Suppose we have a VLIW processor that could issue three memory references, one floating-point operation, and one integer operation or branch in every clock cycle. Assume the following:

- *Load to add latency:* If a L.D instruction is issued in cycle 1, then a dependent ADD.D instruction, which needs the value just loaded, must be issued in cycle 3 or later.
- *Add to store latency:* If an ADD.D instruction is issued in cycle 1, then a dependent S.D instruction, which saves the result to memory, must be issued in cycle 4 or later.
- There is one branch-delay slot.
- An unlimited number of registers are available.

Show an unrolled version of the following loop for such a processor, and schedule it to execute efficiently. Unroll the loop the minimum number of times necessary to eliminate all stalls, but no more than *six* times. *For full credit, your unrolled code must execute as fast as possible. However, do not perform any software pipelining.*

```

Loop:  L.D    F0, 0(R1)
       L.D    F2, 1000(R1)
       ADD.D F4, F0, F2
       S.D    0(R1), F4
       SUBI   R1, R1, #8
       BNEZ   R1, Loop

```

You may enter your answer in the following format.

Mem Ref 1	Mem Ref 2	Mem Ref 3	FP op	Integer/branch

How many clock cycles does your unrolled loop require for each iteration of the original loop?

3. (20 points) **Loop-Carried Dependencies.** Consider the following program fragment:

```
for (i=1; i<100; i=i+1) {  
    a[i] = b[i-1] + c[i];  
    b[i] = c[i] + d[i];  
    b[i+1] = a[i] + e[i];  
}
```

- (a) List all of the dependencies for the above fragment, and for each dependency indicate its class (*i.e.*, RAW, WAW or WAR) and whether it is loop-carried (*i.e.*, whether it is a dependency between instruction instances *across* loop iterations).
- (b) Rewrite the loop so that it is “parallel,” *i.e.*, rewrite so as to eliminate all loop-carried dependencies. Indicate and classify all of the dependencies in the rewritten loop.

Hint: This is an easy question!