CS152 Section 7

Q1: Branch Prediction

Consider the following code:

```
for (int i = 0; i < 4; i++)       // BRANCH_A
    for (int j = 0; j < 4; j++)   // BRANCH_B
        if (j % 2)                // BRANCH_C
            sum += i
```

```
     li x4, 4
     li x1, 0
loop1:
     li x2, 0
loop2:
     andi x5, x2, 0x1
     beqz x5, skip         // BRANCH_C
     add x3, x3, x1
skip:
     addi x2, x2, 1
     bne x2, x4, loop2     // BRANCH_B
     addi x1, x1, 1
     bne x1, x4, loop1     // BRANCH_A
```

Q1.1: Bimodal Counters

Assume a BHT that is indexed by PC, where the PCs of these branches do not alias to the same entry.

What is the branch prediction accuracy for each branch if the BHT uses 1-bit counters? The counters are initialized to 0 (not taken).

What is the branch prediction accuracy for each branch if the BHT uses 2-bit counters? The counters are initialized to 00 (strongly not taken).

Q1.2: Two-level Predictors

At least how many bits of local branch history are required to perfectly predict each branch?

At least how many bits of global branch history are required to perfectly predict each branch?

Q2. VLIW and Software Pipelining

Consider the following loop which computes a dot product:

```
for (i = 0; i < N; i++) {
      C += A[i] * B[i];
}
```

A and B are arrays of double-precision floating-point numbers. The result is accumulated into C.

The code is compiled into the following assembly. x1 and x2 are initialized to the base addresses of arrays A and B, respectively. x3 contains a pointer to the end of array A. f0 holds C.

```
loop:
      fld f1, 0(x1)
      fld f2, 0(x2)
      fmul f3, f1, f2
      fadd f0, f0, f3
      addi x1, x1, 8
      addi x2, x2, 8
      bne x1, x3, loop
```

Q2.1: VLIW Scheduling

Schedule the code onto an in-order VLIW machine with the following execution units:

- One integer ALU, 1-cycle latency, also used for branches
- One load/store unit, 2-cycle latency
- One floating-point adder, 3-cycle latency
- One floating-point multiplier, 4-cycle latency

All functional units are fully pipelined and latch their operands at issue. Instructions are statically scheduled with no interlocks; all latencies are exposed in the ISA. All register operands are read before any writes from the same instruction take effect (i.e., no WAR hazards between operations within a single VLIW instruction). Assume that no exceptions arise during execution.

Do not apply loop unrolling or software pipelining. Entries for NOPs can be left blank.

Label	INT	MEM	FADD	FMUL

Q2.2: Software Pipelining

Schedule operations with VLIW instructions using only software pipelining (no loop unrolling). Include the prologue and epilogue to initiate and drain the software pipeline.

If possible, use different colors to distinguish between instructions from different iterations. Entries for NOPs can be left blank.

Label	INT	MEM	FADD	FMUL