CS152 Section 8

Q1: Parallelism and Utilization

The goal of every one of the various architectures we have studied in this module is to improve the utilization of the functional units built into the design. Achieving perfect saturation is often impossible, and in general we classify the wasted cycles as either vertical waste (due to long or variable latency instructions) or horizontal waste (due to limitations on the number of instructions that can issue or execute on a given cycle). Utilization is improved by exploiting parallelism, but the ways and times at which this parallelism is expressed vary radically between these architectures.

	How is vertical waste reduced?	How is horizontal waste reduced?	Limitations or disadvantages compared to an in-order superscalar RISC machine?
Out-of-Order Superscalar Execution			
VLIW			

Vector			
Vertical Multithreading			
Simultaneous Multithreading			

Q2: Fine-Grained Multithreading

In this problem, we would like to investigate the performance of the following C program on a multithreaded architecture. The arrays A, B, and C contain double-precision floating-point numbers.

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

```
loop: fld  f1, 0(x1)
      fld  f2, 0(x2)
      fadd f3, f1, f2
      fsd  f3, 0(x3)
      addi x1, x1, 8
      addi x2, x2, 8
      addi x3, x3, 8
      addi x4, x4, -1
      bnez x4, loop
```

To split the work across N threads, we rewrite the loop so that each thread executes every Nth iteration of the loop.

```
// TID is the thread ID (0 to N-1)
for (int i = TID; i < M; i += N) {
    C[i] = A[i] + B[i];
}
```

```
loop: fld  f1, 0(x1)
      fld  f2, 0(x2)
      fadd f3, f1, f2
      fsd  f3, 0(x3)
      addi x1, x1, 8N
      addi x2, x2, 8N
      addi x3, x3, 8N
      addi x4, x4, -1
      bnez x4, loop
```

We execute the code on a single-issue in-order processor. Integer operations take 1 cycle to execute, floating-point arithmetic operations take 3 cycles, and memory operations take 2 cycles. The processor uses fine-grained multithreading and switches to a new thread every cycle using fixed round-robin scheduling. Assume perfect branch prediction.

Q2.1: Utilization
How many threads are needed to fully utilize the pipeline?

Q2.2: Peak Performance
What is the peak performance in FLOPs/cycle for the multithreaded program?

Q2.3: Instruction Reordering
Can peak performance be reached with fewer threads by rearranging instructions in the loop?

Q3: Simultaneous Multithreading

Q3.1: Shared Resources

In an SMT processor, some resources are shared between threads, while others are specific to a single thread. For each of the following resources, indicate whether they can be shared by all threads or must be duplicated for each thread.

Program Counter	
Fetch Unit	
Rename Table	
Physical Register File	
Issue Window	
Functional Units	
Reorder Buffer	

Q3.2: Icount Policy

When choosing which thread to fetch from in an SMT processor, we use the Icount algorithm, which prioritizes the thread with the fewest instructions in flight. Why would we expect this to improve throughput?