**Worksheet 7**

**Q1. Parallelism and Utilization**

The goal of every one of the various architectures we have studied in this unit 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 in-order RISC machine? |
| Out of Order Execution |  |  |  |
| VLIW |  |  |  |
| Vector |  |  |  |
| Vertical Multithreading |  |  |  |
| Simultaneous Multithreading |  |  |  |

**Q2. 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 < 500; 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 < 500; 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 with no bypassing**. Integer instructions take 1 cycle to execute, floating point instructions take 3 cycles, and memory instructions take 2 cycles. The processor used fine-grained multithreading and switches to a new thread every cycle using fixed round-robin scheduling. Assume perfect branch prediction.

1. How many threads do we need so that the pipeline is fully utilized?
2. What will be the peak performance in flops/cycle (load/store don’t count as flop) for this program?
3. Can we reach peak performance with fewer threads by rearranging instructions in the loop?

**Q3. Simultaneous Multithreading**

1. 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 are shared or not.

|  |  |
| --- | --- |
| Program counter |  |
| Fetch Unit |  |
| Rename Table |  |
| Physical Register File |  |
| Issue Window |  |
| Functional Units |  |
| ROB |  |

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