CE-4
August 2015 QE

I. Processor (25 points) In modern out-of-order processor pipelines, there are many "pipeline loops" due to data/information generated in later pipeline stages and used in earlier stages (e.g., register bypass, branch resolution, and rename table update). Modern pipelines have many such loops each of which is relevant only to a subset of instructions.

Assume single-issue pipeline for the following questions.

1. (8 points) What is the increase in the cycles per instruction (CPI), if 80% of instructions incur a loop from the 8^{th} stage to the 8^{th} stage?
   Also, give a real example of such a loop.
2. (7 points) For a loop from the 15^{th} stage back to the 1^{st} stage, the pipeline employs prediction with accuracy 90% (you should figure out how prediction lessens the loops' impact on the CPI). What is the increase in the CPI if the loop is relevant to 20% of all instructions?
3. (10 points) For a loop from the 15^{th} stage back to the 1^{st} stage and another from the 10^{th} stage back to the 5^{th} stage, if 1% of instructions are affected by the former loop and 3% (including the previous 1%) are affected by the latter loop, what is the increase in the CPI?

II. Memory system (25 points)

1. (15 points) Assume 25% of all instructions are loads and stores, L1 miss rate is 20%, L2 miss rate is 20% (out of L1 misses), and L1, L2, main memory latencies are 3, 32, and 300 cycles, respectively, and their bandwidths are, respectively, 1 access (hit or miss) every 1, 8, and 30 cycles. What is the achievable IPC (instructions per cycle), if we assume abundant parallelism due to simultaneous multithreading?
2. (10 points) Inverted page tables reduce page table space overhead via hashing. However, how are hashing collisions handled? Explain any policy the hardware/operating system may have to use.

III. Multicore (25 points)

1. (10 points) Assuming non-atomic writes, add statements to the following code to show sequential consistency (SC) violation. You may add statements to one or both threads, and either before or after the statements shown below.

   Assume that before the following code segment runs X=Y=0 .

   Thread 1: write X = 10;

   Thread 2: write Y = 20;

   Show your new code and explain how SC may be violated.

2. (15 points) Assume the following multithreaded code where *mypid* is each thread's process id. The threads run on n cores with coherent private caches.

```
repeat {many times}
   for i = 1 to n
     A[mypid] += A[mypid]%mypid;
```

   (a) (5 points) Explain why this code is likely to perform poorly.

   (b) (10 points) Describe how to address this problem and show your pseudocode.

IV. Fundamentals (25 points)

A high-performance architecture uses ultra-wide simultaneously multi-threaded (SMT) processors (e.g., 128-way SMT processors) to target "embarrassingly" parallel, highly regular and loop-based codes with heavy branching (e.g., highly-conditional dense matrix applications). Due to the heavy branching, vector compiler scheduling is not effective. Instead, the compiler generates simple, parallel threads from the loop iterations each of which runs on an SMT context (e.g., 128 iterations run in parallel. While the branching disallows lock-step execution across the threads, the iterations are naturally load-balanced and access mostly sequentially-contiguous data (e.g., thread i accesses i^{th} array elements). A key challenge is designing memory system to support such processors which employ a cache hierarchy (memory system includes caches and main memory). Assume the code has enough reuse to justify a conventional cache hierarchy.

1. (5 points) Which memory-system parameters poses a key challenge for data access? Which parameters are much less relevant?

2. (7 points) Based on the code characteristics listed above, what would be your strategy to address the above challenge? Describe your strategy for both caches and main memory.

3. (13 points) What difficulty does the lack of lock-stepping cause to your strategy? How would you tackle the difficulty?