548 Midterm

Due 6pm Feb 18th - NO EXTENSIONS

ALL WORK MUST BE DONE INDIVIDUALLY

This is going to be an unusual, but I hope very thought provoking exam. This exam has a single question which you are free to apply to a variety of architectural features. The question is: write a program, preferably in C, but assembly may be required, to decipher the structure of a Pentium processor. Below is outlined the various processor features you can explore and a point value for each one – the points are proportional to their difficulty (I think!). Your goal on this midterm is to achieve a solid 100 points from the 520 available *with no less than 10 points coming from each category*. Do not do them all! My advice is you stop when you feel confident that you have 100 points, which may mean you solve for 150 or so.

When turning in the solutions you should provide: (1) the code, (2) the results from running the code, and (3) a write up with the results (graphs too if required) that articulates why the results elucidate the processor feature(s) in question.

This exam is to be done individually. Please wait until after the exam is over to discuss it. This exam is closed-net, as in I expect you not to google around for things. You are free to use any class material (i.e. anything linked from the website). If you have a question about something, please ask us (Andrew or I) and we will try and respond to the class list.

This midterm is really more about thought exploration than code. It may look like coding, but you should think carefully about the code you write and how it interacts with the microarchitecture. **Well reasoned explanations of incorrect code are scored higher than incorrect reasons and correct code.**

1 Memory system

- **1-a 20 points:** Cache Hierarchy. Write an application to determine the associativity, bank size, line size, and latency for all levels of the *data* cache hierarchy (including relevant parameters for main memory).
- **1-b 20 points:** Determine the L1 cache replacement policy (hint: its *not* LRU).
- **1-c 10 points:** Determine the *bandwidth* between each level of the data memory hierarchy.
- **1-d 30 points:** Determine the cache parameters of the internal trace cache on the Pentium 4.
- **1-e 10 points:** Determine the TLB size and associativity.
- **1-f 10 points:** How expensive is a TLB miss?

- **1-g 10 points:** Determine the size and characteristics of the victim buffer.
- **1-h 20 points:** Determine the size and characteristics (number of, prefetch strategies supported, etc) of the prefetch buffer.
- **1-i 20 points:** Determine the maximum number of basic blocks in a trace-cache line.

2 Instruction fetch

- **2-a 100 points:** Determine the branch predictor algorithms.
- **2-b 20 points:** Determine the branch predictor table size(s).
- **2-c 20 points:** Determine the BTB size.
- **2-d 10 points:** Determine if there is and how large the return stack depth predictor is?
- **2-e 10 points:** Determine the *external* x86 fetch bandwidth.
- **2-f 20 points:** Determine the *internal* u-op fetch bandwidth.
- **2-g: 10 points** Find 3 x86 instructions that utilize the "slow" decode process as opposed to the fast one.
- **2-h: 20 points** Find 3 x86 instructions (unique from the 3 used for the previous question if you answered that) that translate into more than 1 micro-op, and demonstrate the number of micro-ops they require.
- **2-i: 20 points** Determine the maximum number of speculative branches that can be in-flight.

3 Processing core

- **3-a 30 points:** Determine the maximum number of inflight instructions.
- **3-b: 20 points:** Determine the number of physical registers available in the machine.
- **3-c 10 points:** Determine the maximum number of outstanding dispatched memory operations.
- **3-d 20 points:** Determine the number and type of ALU cores in the system.
- **3-e: 20 points:** Determine the maximum number of in-flight instructions between the scheduler and execution phases of the core.
- **3-f 20 points:** Determine the maximum misspeculation penalty (the time from detecting an incorrect branch outcome to fetching the next correct instruction).