ECE 252 / CPS 220 Homework #4
Due in class on Friday, Oct 30 (submission of Q6 due by 10:00 am)
Total Points 145

Explain all of your answers to get full credit!

<u>Static ILP</u>

1) (a) (10 points) Create a single hyperblock (using predication) from the following code. Assume that all instructions can be predicated and that you can use as many different predicate registers (p1, p2, etc.) as you need. Assume that you can set a predicate register (in this example, p1) with:

`setpeqz p1, r1 // p1 = 1 (true), if r1= 0`

Assume that you can predicate an instruction with the following syntax:

`add r1, r2, r3, p1 // r1 = r2 + r3, if p1 = 1 (true); else NOP`

Do not worry about performance (yet). Make sure that you rename registers appropriately, if necessary.

(b) (10 points) Schedule the hyperblock for optimal performance on a statically scheduled processor.

```
           add r3, r1, r2 // r3 = r1 + r2
           sub r7, r3, r1 // r7 = r3 - r1
           beqz NEXT, r7 // branch to NEXT, if r7==0
           sub r4, r3, r2
           and r3, r4, r4
           beqz NEXT, r4
           load r4, 0(r2) // r4 = Memory [r2]
  NEXT:    xor r4, r4, r3
           store r4, 0(r2) // Memory [r2] = r4
```

2) (10 points) The IA-64 ISA provides "speculative" instructions and "advanced" loads. For both of these ISA features, explain how they work and what purposes they serve.

<u>Memory Hierarchies and Caches</u>

3) (15 points) H&P 5.1a, b, and c (but not part d). Note that you should use the online version of Cacti at http://quid.hpl.hp.com:9081/cacti/

4) (10 points) Compare the cache hierarchies of the IBM POWER6 and the Intel Core i7. Discuss the sizes, associativities, bandwidths, and other interesting features of each level of cache in each processor.

5) (10 points) Write a couple paragraphs that describe "fully buffered DIMMs." Explain what they are and how they work.

Exploring Caches with SimpleScalar

6) (80 points) We will use `sim-outorder` for this set of three experiments. Use the default simulator parameters unless instructed otherwise.

Experiment #1: Explore the tradeoff between associativity and hit latency for a 4KByte L1 data cache (L1 D$) by comparing the performance of (a) 2-cycle direct-mapped cache, (b) 3-cycle 2-way set-associative cache, and (c) 4-cycle 4-way set-associative cache. Do not modify the other default characteristics of the L1 D$ (e.g., block size, total cache size, replacement policy). However, note that `sim-outorder` makes you specify the number of sets—make sure that the caches you specify are all 4 KB in total size (not including tags)! This experiment requires no modification to `sim-outorder`. What conclusions do you draw from this experiment?

Experiment #2: Add a 2-entry fully-associative victim cache (with LRU replacement) to the L1 D$. The victim cache can be accessed in parallel with the L1 D$, and a hit in the victim cache has no performance penalty (i.e., takes the same time as an L1 D$ hit). Assume all default simulator parameters except for the L1 D$'s set-associativity. How much does the victim cache help the direct-mapped cache? How about the 2-way and 4-way set-associative caches? What conclusions do you draw from this experiment?

Experiment #3: Explore the importance of non-blocking caching for out-of-order processors. Compare the performance of a processor with a non-blocking (lockup-free) cache (which is sim-outorder's default) to the performance of a processor with a blocking cache. You will need to modify sim-outorder to model a blocking cache. Assume all default simulator parameters. Do NOT use the victim cache from Experiment #2. Why do you think non-blocking makes such a big difference?

You should submit a single file called `homework4.c`. This file will include your code for both the victim cache (Experiment #2) and the blocking L1 D$ (Experiment #3). Add a compile time or runtime switch to disable either of these two features. SimpleScalar makes adding runtime switches fairly easy.