Computer Systems Architecture
Assignment 6
Due: Tuesday, May 27

The purpose of this assignment is to learn to analyze data that is typically obtainable from a hardware simulation run and to use the data to answer an architectural design question. You will be presented with two alternate viewpoints on whether speculation on an SMT processor is a good idea and a set of data that compares SMT with an out-of-order superscalar with comparable hardware. Your job will be to decide which hypothesis is supported by the data.

For this assignment try to work in teams of two, and for the same reason as before – so that you'll benefit from discussions with your partner.

The Issue: to speculate or not to speculate on an SMT

Instruction speculation is a crucial contributor to the performance of modern superscalar processors. (Our measurements show that 70% of committed instructions on an aggressively designed out-of-order superscalar started out speculative!) Instruction speculation hides branch latencies, and thereby boosts performance, by executing the likely branch path without stalling. Branch predictors, which provide accuracies up to 96% (excluding OS code), are the key to effective speculation. The primary disadvantage of speculation is that some processor resources are invariably allocated to useless, wrong-path instructions that must be flushed from the pipeline. However, since resources are often underutilized on superscalars because of low single-thread instruction-level parallelism (ILP), the benefit of speculation far outweighs this disadvantage and the decision to speculate as aggressively as possible is an easy one.

In contrast to superscalars, simultaneous multithreading (SMT) processors operate with high processor utilization, because they issue and execute instructions from multiple threads each cycle and all threads dynamically share most hardware resources. If some threads have low ILP, utilization is improved by executing instructions from additional threads; if only one or a few threads are executing, then all critical hardware resources are available to them. Consequently, instruction throughput on a fully loaded SMT processor is up to four times higher than on a superscalar with comparable hardware on a variety of workloads (integer, scientific, database, web service, and the operating system).

With its high hardware utilization, speculation on an SMT may *harm* rather than improve performance, particularly with all hardware contexts occupied. Speculative (and potentially wasteful) instructions from one thread may compete with useful, non-speculative instructions from other threads for highly utilized hardware resources, and in some cases could displace them, lowering performance.

Given these lines of argument, one that argues for speculation on an SMT and one that argues against it, analyze the performance data below to decide whether an SMT processor should speculate. Back up your decision with a data analysis, i.e., don't just say SMT should speculate or SMT should not speculate – tell us why and base the reasons on your data analysis, quoting the figures that prove your point. Really mine the data. It is the quality of your analysis that will determine your grade.

The SMT simulations

The SMT processor used in the simulations was configured according to the parameters in the table below. The superscalar was identical, except that it had 2 fewer pipeline stages, since it reads and writes to a smaller register file. The workload was SPECInt95, which is comprised of commonly used integer applications. The results are averaged over the SPECInt95 applications.

CPU	
Thread Contexts	8
Pipeline	9 stages, 7 cycle misprediction penalty.
Fetch Policy	8 instructions per cycle from up to 2 contexts (the ICOUNT scheme)
Functional Units	6 integer (including 4 load/store and 2 synchronization units) 4 floating point
Instruction Queues	32-entry integer and floating point queues
Renaming Registers	100 integer and 100 floating point
Retirement bandwidth	12 instructions/cycle
Branch Predictor	hybrid predictor (shared among all contexts)
Local Predictor	4K-entry prediction table, indexed by 2K-entry history table
Global Predictor	8K entries, 8K-entry selection table, gshare (xor'd)
Branch Target Buffer	256 entries, 4-way set associative (shared among all contexts)
Cache Hierarchy	
Cache Line Size	64 bytes
Icache	128KB, 2-way set associative, dual-ported, 2 cycle latency
Dcache	128KB, 2-way set associative, dual-ported (from CPU, r&w), single-ported (from the L2), 2 cycle latency
L2 cache	16MB, direct mapped, 23 cycle latency, fully pipelined (1 access per cycle)
MSHR	32 entries for the L1 cache, 32 entries for the L2 cache
Store Buffer	32 entries
ITLB & DTLB	128-entries, fully associative
L1-L2 bus	256 bits wide
Memory bus	128 bits wide
Physical Memory	128MB, 90 cycle latency, fully pipelined

The SMT data

```
IPC: 5.18


IPC per pipe stage:

            spec(%)   true-path spec(%) wrong-path spec(%)   nonspec(%)  total
fetch     3.49 (56.5%)   2.82 (45.6%)     0.67 (10.9%)      2.69 (43.5%)  6.19
decode    3.31 (56.0%)   2.75 (46.6%)     0.56 (9.4%)       2.60 (44.0%)  5.91
rename    3.21 (55.4%)   2.74 (47.2%)     0.47 (8.13)       2.59 (44.6%)  5.80
issue     2.92 (53.4%)   2.72 (49.6%)     0.20 (3.7%)       2.56 (46.6%)  5.48
read1     2.92 (53.3%)   2.72 (49.7%)     0.20 (3.7%)       2.56 (46.5%)  5.45
read2     2.86 (52.9%)   2.71 (50.1%)     0.15 (2.8%)       2.55 (47.1%)  5.41
execute   2.78 (52.4%)   2.68 (50.6%)     0.10 (1.8%)       2.52 (47.6%)  5.30
write     2.75 (52.2%)   2.68 (50.9%)     0.07 (1.4%)       2.51 (47.8%)  5.26
commit    2.67 (51.6%)   2.67 (51.5%)     0.00 (0.0%)       2.51 (48.4%)  5.18


percentage of instructions squashed (from branch speculations): 9.7
cycles simulated: 200000000

avg. number of contexts that are speculating in any given cycle: 4.7



Fetch statistics
========================================
avg. number of contexts ready to be fetched each cycle: 6.6
avg. number of speculating contexts ready to be fetched each cycle: 4.6
avg. number of contexts not speculating ready to be fetched each cycle: 2.0

avg. number of instructions fetched per cycle: 6.2
avg. number of speculative instructions fetched per cycle: 3.5
avg. number of nonspeculative instructions fetched per cycle: 2.7

avg. number of consecutive speculative branches per context: 0.8

total instructions fetched: 1237550208
speculative instructions fetched (% of fetched instructions): 61.4
nonspeculative instructions fetched (% of fetched instructions): 38.6

avg. branch delay (cycles/branch): 10.0

avg. number of instructions squashed per misprediction: 8.8

avg. fetch-to-fetch delay for a thread (the number of cycles between two
consecutive fetches from the same context): 5.1

branch prediction accuracy: 87.7%

useful (true-path) instructions (% of maximum fetch bandwidth): 68.5
```

```
percentage of forward-taken branches: 31.1
percentage of backward taken branches: 17.9
percentage of forward not taken branches: 45.9
percentage of backward not taken branches: 5.1



IQ statistics
========================================
avg. number of contexts present in IQ: 5.3

avg. IQ size (instructions): 21.8

avg. number of speculative instructions in IQ: 12.9
avg. number of nonspeculative instructions in IQ: 8.6

avg. number of speculative instructions ready to issue: 4.4
avg. number of nonspeculative instructions ready to issue: 4.1

IQ full (percentage of cycles): 4.3
avg. number of speculative instructions in full IQ (instructions/backup): 21.3
avg. nonspeculative instructions in full IQ (instructions/backup): 10.7

avg. speculative instructions held up by full IQ (instructions/backup): 1.2
avg. nonspeculative instructions help up by full IQ (instructions/backup): 0.9

out of renaming registers (percentage of cycles): 0.3
avg. number of speculative instructions held up by a lack of registers: 1.8
avg. number of nonspeculative instructions help up by a lack of registers: 1.8

Where in the pipeline instructions are squashed (mostly from branch
speculations but a small number from other factors, such as exceptions):
        per stage           accumulative
0:    12338755 (8.9%)      12338755 (8.9%)
1:    18641892 (13.4)      30980648 22.2%)
2:    56457388 (40.5%)     87438032 (62.7%)
3:     1762941 (1.3%)      89200976 (64.0%)
4:      195106 (0.1%)      89396080 (64.1%)
5:    11986018 (8.6%)     101382096 (72.7%)
6:    16690893 (12.0%)    118072992 (84.7%)
7:     6255869 (4.5%)     124328864 (89.2%)
8:    15058986 (10.8%)    139387856 (100.0%)
```

The Superscalar data

```
IPC:          2.78


IPC per pipe stage:

          spec (%)   true-path spec (%) wrong-path spec (%)  nonspec (%) total
fetch   3.52 (93.2%)    2.55 (67.4%)      0.97 (25.74%)      0.26 (6.8%)  3.76
decode  3.35 (93.1%)    2.53 (70.4%)      0.82 (22.68%)      0.25 (6.9%)  3.60
rename  3.23 (92.9%)    2.53 (72.8%)      0.70 (20.05%)      0.25 (7.1%)  3.48
issue   2.84 (92.0%)    2.53 (81.8%)      0.31 (10.17%)      0.25 (8.0%)  3.09
read    2.85 (92.0%)    2.53 (81.8%)      0.31 (10.17%)      0.25 (8.0%)  3.09
execute 2.74 (91.7%)    2.53 (84.7%)      0.21 (6.99%)       0.25 (8.3%)  2.99
commit  2.53 (91.1%)    2.53 (91.1%)      0.00 (0.00%)       0.25 (8.9%)  2.78


percentage of instructions squashed (from branch speculations): 25.8
cycles simulated: 200000000

avg. number of contexts that are speculating in any given cycle: 0.9

avg. basic block size: 4.8



Fetch statistics
========================================
avg. number of contexts ready to be fetched each cycle: 0.97
avg. number of speculating contexts ready to be fetched each cycle: 0.95
avg. number of contexts not speculating ready to be fetched each cycle: 0.02

avg. number of instructions fetched per cycle: 3.8
avg. number of speculative instructions fetched per cycle: 3.5
avg. number of nonspeculative instructions fetched per cycle: 0.3

avg. number of consecutive speculative branches per context: 3.3

total instructions fetched: 755177536
speculative instructions fetched (% of fetched instructions): 94.2
nonspeculative instructions fetched (% of fetched instructions): 5.8

avg. branch delay (cycles/branch): 1.3

avg. number of instructions squashed per misprediction: 35.2

avg. fetch-to-fetch delay for a thread (cycles): 1.4

branch prediction accuracy: 92.4%

useful (true-path) instructions (% of maximum fetch bandwidth): 34.9
```

```
IQ statistics
========================================
avg. IQ size (instructions/cycle): 9.3

avg. number of speculative instructions in IQ: 8.4
avg. number of nonspeculative instructions in IQ: 0.6

avg. number of speculative instructions ready to issue: 3.3
avg. number of nonspeculative instructions ready to issue: 0.3

IQ full (percentage of cycles): 1.7
avg. number of speculative instructions in a full IQ: 31.6
avg. number of nonspeculative instructions in a full IQ: 0.4

avg. number of speculative instructions held up by a full IQ: 2.1
avg. number of nonspeculative instructions help up by a full IQ: 0.01

out of renaming registers (percentage of cycles): 0.2
avg. number of speculative instructions held when there are no registers: 3.8
avg. number of nonspeculative instructions help up when there are no
registers: 0.01



Where in the pipeline instructions are squashed (mostly from branch
speculations but a small number from other factors, such as exceptions):
        per stage          accumulated
0:   23731896 (12.7%)     23731896 (12.7%)
1:   23796264 (12.7%)     47528160 (25.4%)
2:   76448264 (40.9%)    123976424 (66.3%)
3:        332 (0.0%)     123976752 (66.3%)
4:         70 (0.0%)     123976824 (66.3%)
5:   21156748 (11.3%)    145133568 (77.6%)
6:   41778032 (22.4%)    186911600 (100.0%)


Speculative Instructions: 711180385 (94.2%)
```

Part II: The report

Your report should follow the usual report guidelines, and 2-3 pages of text should do it. This time you can write only a results section that argues whether SMT should speculate or not. The premise comes from the assignment itself.

Part III: Turning it in

Turn in your report via e-mail or paper handouts. Use postscript or pdf (it’s hard for us to read MS Word documents).