Computer Science 146/246
Homework #3

Due 11:59 P.M. Thursday, March 2nd, 2023

Suggested Readings Computer Architecture: A Quantitative Approach, *Fifth* Edition, Chapter 3.8-3.12

1 Pin Assignment: Branch Predictor [50 pts]

1.1 Summary

You will write code to simulate several branch predictors from a classic paper [2]. We have a template for you under `hw3.cpp`, which currently implements a 1-bit Branch Prediciton Buffer, mentioned as *Strategy 6* in another classic paper [1].

When your code is ready to compile, type "make", which will generate your hw3.so tool in obj-intel64/. The current template will add instrumentation that keeps track of all branches in the program and their direction (taken / not taken). As a brief recap, here is how to run hw3 and Pin on `/bin/ls`, listing files in the current directory while tracking branch activity:

```
$ $PIN_ROOT/pin -t obj-intel64/hw3.so  -- /bin/ls
```

1.2 Benchmarks

While in the previous assignment you ran /bin/ls, this time we will use two sample programs from the SPEC benchmarking suite. Please present your results for the following programs. They are run naively as follows:

```
$ $HOME/shared_data/benchmarks/libquantum_O3 400 25
$ $HOME/shared_data/benchmarks/dealII_O3 10
```

Pass the above application commands to the Pin tool in place of "/bin/ls". Both benchmarks run for approximately 40 seconds each without Pin instrumentation.

1.3 Evaluation

There are four parts to this assignment:

A. Implement Automaton A2 in Figure 2 from the Yeh paper (2-bit saturating counter) with the Branch Prediction Buffer scheme. Currently the homework template implements Automaton Last-Time (LT) shown in Figure 2. [10 pts]

B. Implement the 2-level adaptive training scheme (see Figure 1) with Automaton A2 (see Figure 2) using a Hash History Register Table. [10 pts]

C. Lookup the processor model listed in /proc/cpuinfo, find out what kind of branch prediction technique is being implemented for the processor model used in your machine and describe it briefly. [5 pts]

   For the branch prediction implementations, you must parameterize the following key pieces (a) the number of entries in the Hash History Register Table (HHRT) (b) the number of entries in the Pattern Table (PT), as that will help you complete the next part of the assignment.

D. Evaluate your designs:

   (a) Provide branch prediction accuracy plots for the local predictor across the two benchmarks, 1-bit vs. 2-bit counter sweeping Branch Prediction Buffer size. [5 pts]

   (b) Provide prediction accuracy results for the 2-level predictor across the two benchmarks, sweeping the number of entries of HHRT and PT. [10 pts]

   (c) Provide branch prediction accuracy reported by `likwid-perfctr` on the server. To integrate the average prediction accuracy with the two previous plots, compute the average prediction accuracy and draw a horizontal line to see under what circumstances the simulated predictor approaches the performance of the real branch predictor. [10 pts]

The last part will require you to sweep a range of values. This range must be sufficiently large to quantitatively draw conclusions on the performance of one predictor versus another. You should also comment on the hardware resources used by each design in that range. Please provide accuracy plots, as the paper does.

Sweeping parameters is pretty common in architecture – after all, our textbook is called "A quantitative approach". It is a good idea to automate that process – there is no need to manually run the same benchmark / predictor configuration and babysit every run. When you parameterize your simulations, you expose parameters to the command line with the Pin KNOB library. A simple bash script can run all your simulations in one fell swoop.

2 Tomasulo and Hardware Speculation [20 pts]

Consider the following code fragment:

```
LOOP:   LD      F0, 0(R1)
        DIVD    F2, F0, F6
        LD      F6, 8(R1)
        DIVD    F6, F6, F2
        SD      F6, 16(R1)
        DADDI   R1, R1, #-32
        BNEQZ   R1, LOOP
```

For this problem, consider the following architecture specifications:

Functional Unit Type	Cycles in EX	Number of Functional Units
Integer	1	1
FP Divider	15	1

A. Assume you have unlimited reservation stations.

B. Memory accesses use the integer functional unit to perform effective address calculation during the EX stage. For stores, memory is accessed during the EX stage (Tomasulo's algorithm without speculation) or commit stage (Tomasulo's algorithm with speculation). All loads access memory during EX stage.

C. Loads and stores stay in EX for 1 cycle.

D. Functional units are not pipelined.

E. If an instruction moves to its WB stage in cycle x, then an instruction that is waiting on the same functional unit (due to a structural hazard) can start executing in cycle x.

F. An instruction waiting for data on the CDB can move to its EX stage in the cycle after the CDB broadcast.

G. Only one instruction can write to the CDB in one clock cycle. Branches and stores do not need the CDB.

H. Whenever there is a conflict for a functional unit or the CDB, assume that the oldest (by program order) of the conflicting instructions gets access, while others are stalled.

I. Assume that BNEQZ occupies the integer functional unit for its computation and spends one cycle in EX.

J. Assume that the result from the integer functional unit is also broadcast on the CDB and forwarded to dependent instructions through the CDB (just like any floating point instruction).

2.1 Tomasulo with Speculation

Complete the pipeline summary table using Tomasulo's algorithm assuming hardware speculation. That is, assume that an instruction can issue even before branch has completed (or started) its execution (as with perfect branch and target prediction).

Additionally, assume that you have as large a reorder buffer as you need. Only one instruction can commit each cycle. Also assume that stores only calculate target addresses in EX and perform memory accesses during the Commit stage, and do nothing during the WB stage.

Fill in the cycle numbers in each pipeline stage for each instruction in the first two iterations of the loop, assuming the **branch is always taken**.

Show your complete work using a cycle by cycle breakdown PowerPoint presentation, similar to what was shown in lecture using a table for Instruction Status, Reorder Buffer and Register Result Status / Reorder Number.

The entries for the first instruction in the first iteration are filled in already.

	Iteration 1				Iteration 2			
Instruction	Issue	EX	WB	CMT	Issue	EX	WB	CMT
`LP:LD F0, 0(R1)`	1	2-3	4	5				
`DIVD F2, F0, F6`	2	5-						
`LD F6, 8(R1)`	3	4-5						
`DIVD F6, F6, F2`	4							
`SD F6, 16(R1)`	5							
`DADDI R1, R1, #-32`								
`BNEQZ R1, LP`								

3 Submission

For the branch predictor assignment parts A and B, submit all source code for implementing the two branch predictor schemes. For part C and D, include your response and accuracy plots from your Pin assignment as a PDF file.

Put your PDF files and all source code (including makefiles so I can easily compile it) in one folder titled "branch-predictor". Please upload your zipped folder via Canvas. Please make sure your code compiles before uploading!

Please include solutions to Tomasulo with Speculation in a separate PDF. Only include your completed Instruction Status table in the pdf (so I don't need to go hunting for your answers).

In a separate document, include your cycle-by-cycle PowerPoint showing your complete work. Please write down any assumptions you make and any other notes you'd like me to read justifying your solution / points of confusion.

Please upload the "branch-predictor" folder, pdf and powerpoint to canvas in a .zip file. Title the submission using the following format `cs146_246_hw3_[first]_[last].zip`

References

[1] James E Smith. A study of branch prediction strategies. In *Computer Architecture (ISCA)*, 1981.

[2] Tse-Yu Yeh and Yale N Patt. Two-level adaptive training branch prediction. In *Microarchitecture (MICRO)*, 1991.

Updated February 24, 2023, Matt Adiletta