Computer Systems Architecture
Assignment 2
Part I due: Monday, January 17
Parts II and III due: Wednesday, January 26

This assignment is meant to acquaint you with a type of standard architectural performance tool (Atom and/or Etch) and to evaluate:

- The dynamic instruction mix in the ISA of the Alpha 21164 or the Pentium Pro
- Today's dynamic branch prediction techniques

You are expected to work in teams of 2 people.

PART I: Instrumentation

1. Pick an architecture to work on: either the Alpha or x86. For the Alpha, you will use the instrumentation tool called Atom (that was developed at Compaq Western Research Lab), and for the Pentium, a similar tool called Etch (that was developed here by some of your fellow grad students, Dennis Lee and Geoff Voelker). Note that Etch is still relatively new and is not as stable as Atom. We recommend Etch only if you're somewhat familiar with coding in an NT environment. Pointers to documentation for the ISA and the instrumentation tool are available on the course web page (or will be there soon). As soon as you have decided, email your decision along with the names of your group members to Luke.

   For the really adventurous, a third option is to use tools developed for the Kimera project (by grad student Gun Sirer) with Java applications. Please contact Luke if you are interested in exploring this idea. (But I will say that no one chose this option for a similar project last year.)

2. Pick a non-trivial and interesting program to instrument. For architecture, "non-trivial" means that the program is not too short (> 10^9 instructions) and that its dataset does not fit into modern L1 caches (assume 32KB). You don't want too long a program either, since instrumentation will slow it down by a factor of 10 to 100. "Interesting" programs means staying away from the typical SPEC type applications, and choosing one that should utilize some of the new instructions we have studied. Graphics programs, object-oriented programs (e.g., C++, Cecil, Java), compilers (e.g., gcc, vortex), image manipulation (jpeg, mpeg, etc.), games, ghostview and the like are good candidates.

3. Instrument your program with the appropriate tool. Report on:
   - Total number of instructions executed
   - Frequency of various instruction types, such as:
     - integer or FP arithmetic and logical instructions
     - data transfer instructions

- conditional and unconditional branches
- subroutine calls & returns
- the "new" instructions

4. For conditional branches, record the frequencies of each element of the Cartesian product: (*forward*, *backward*) x (*taken*, *not taken*). Comment on what you find with respect to the rationale for static branch prediction algorithms.
5. Make histograms of:
   - the (dynamic) length of basic blocks
     Basic block size is a static count -- it relies only on the source instructions, and not on whether the branch/jump is taken during execution. Note, however, that the "average" size should be computed dynamically. For example, suppose a program has 10 basic blocks, 9 of which are of size 4 and the 10th of size 2. Suppose 4 of the size 4 blocks are never executed during a run, the other 5 are executed once and the size 2 block is executed 15 times (say, a loop). Then, the dynamic average basic block size is (5*1*4 + 15*2)/20 = 2.5 whereas, the static average size is (9*4 + 2)/10 = 3.8
   - the (dynamic) length of sequences of code uninterrupted by a transfer of control (branch/jump)
     The "uninterrupted code sequence" is a dynamic count in a different sense. Here, it matters whether or not branches are taken. If a branch is taken, the code sequence is interrupted (same for jumps, return, etc.) If, however, it falls through, then it is uninterrupted.
6. Measure something else you find of interest that isn't on this list.

PART II: Branch prediction

Compare one static prediction technique (of your choice) with two dynamic branch prediction techniques. One dynamic branch predictor can be simple, such as two-state branch prediction, and will serve as your baseline dynamic branch prediction performance. The other should be more state-of-the-art, such as correlated dynamic branch prediction or dynamic branch prediction with confidence predictors. We will discuss what these predictors look like in class and you'll have reading on them. Simulate these techniques on your program (using Atom/Etch). Record the number of correct predictions and mispredictions. Be sure to update your branch prediction tables on a misprediction.

For the dynamic predictors, vary the size of the prediction table to gauge the sensitivity of branch prediction performance to prediction table size or vary the size of the confidence thresholds to gauge the sensitivity of branch prediction performance to how confident branch predicting confidence should be.

Measure the performance of your application with each hardware technique. Gather component metrics in addition to baseline performance, so that you can richly analyze your results. If you feel that your metrics do not give you a good indication of performance, what metrics would give you a better view?

PART III: Comparison

Find another group that is using either:

- the same application on a different architecture
- a different application on the same architecture
- the same application on the same architecture

Compare the results of your Parts I and II. When your architectures or applications are different, what conclusions can you draw about the architecture and/or applications used? When they are the same, whose branch predictors did the best and why? This means you will have to scope out your competitors when you choose an application/architecture combination for the first part of the assignment.

The Write-ups

Each write-up should not be more than about 3 pages long. You should pretend you're writing a paper for publication (but omit an abstract). This means that, for instance, you should have an introduction, a methodology section in which you describe your experiments and a results and results analysis section. While we will not necessarily be checking the accuracy of your results, you should still provide enough information to allow us to duplicate your results. For grading purposes, the focus is on what you do with your measurements: What conclusions can you draw? What explanations can you give?

You don't have to restrict yourself to these questions, but we just wanted to give you some idea of what you should be thinking about when doing the write-up. (Again, imagine you're writing a technical paper...) In general, all your HW/project write-ups will have this flavor.

Q1: Explain why you chose the benchmark and why did you think it was interesting? In other words, have some sort of hypothesis here.

Q2: What is your experimental methodology for the measurements you will be taking?

Q3: What do the numbers you measure tell you about the program?

Q4: (already in the handout) Comment on what you find wrt the rationale for static branch prediction schemes.

Q5: What do you conclude from the similarities/differences between the two histograms?

Q6: Say why your measurement is interesting and what that tells you about the program.

Consider this a heads up that you should spend time not just on the implementation but also on the write-up!

Computer Systems Architecture
Assignment 2
Part I due: Monday, January 17
Parts II and III due: Wednesday, January 26

This assignment is meant to acquaint you with a type of standard architectural performance tool (Atom and/or Etch) and to evaluate:

- The dynamic instruction mix in the ISA of the Alpha 21164 or the Pentium Pro
- Today's dynamic branch prediction techniques

You are expected to work in teams of 2 people.

PART I: Instrumentation

1. Pick an architecture to work on: either the Alpha or x86. For the Alpha, you will use the instrumentation tool called Atom (that was developed at Compaq Western Research Lab), and for the Pentium, a similar tool called Etch (that was developed here by some of your fellow grad students, Dennis Lee and Geoff Voelker). Note that Etch is still relatively new and is not as stable as Atom. We recommend Etch only if you're somewhat familiar with coding in an NT environment. Pointers to documentation for the ISA and the instrumentation tool are available on the course web page (or will be there soon). As soon as you have decided, email your decision along with the names of your group members to Luke.

   For the really adventurous, a third option is to use tools developed for the Kimera project (by grad student Gun Sirer) with Java applications. Please contact Luke if you are interested in exploring this idea. (But I will say that no one chose this option for a similar project last year.)

2. Pick a non-trivial and interesting program to instrument. For architecture, "non-trivial" means that the program is not too short ($> 10^9$ instructions) and that its dataset does not fit into modern L1 caches (assume 32KB). You don't want too long a program either, since instrumentation will slow it down by a factor of 10 to 100. "Interesting" programs means staying away from the typical SPEC type applications, and choosing one that should utilize some of the new instructions we have studied. Graphics programs, object-oriented programs (e.g., C++, Cecil, Java), compilers (e.g., gcc, vortex), image manipulation (jpeg, mpeg, etc.), games, ghostview and the like are good candidates.

3. Instrument your program with the appropriate tool. Report on:
   - Total number of instructions executed
   - Frequency of various instruction types, such as:
     - integer or FP arithmetic and logical instructions
     - data transfer instructions

- conditional and unconditional branches
- subroutine calls & returns
- the "new" instructions

4. For conditional branches, record the frequencies of each element of the Cartesian product: (*forward*, *backward*) x (*taken*, *not taken*). Comment on what you find with respect to the rationale for static branch prediction algorithms.
5. Make histograms of:
   - the (dynamic) length of basic blocks
     Basic block size is a static count -- it relies only on the source instructions, and not on whether the branch/jump is taken during execution. Note, however, that the "average" size should be computed dynamically. For example, suppose a program has 10 basic blocks, 9 of which are of size 4 and the 10th of size 2. Suppose 4 of the size 4 blocks are never executed during a run, the other 5 are executed once and the size 2 block is executed 15 times (say, a loop). Then, the dynamic average basic block size is (5*1*4 + 15*2)/20 = 2.5 whereas, the static average size is (9*4 + 2)/10 = 3.8
   - the (dynamic) length of sequences of code uninterrupted by a transfer of control (branch/ jump)
     The "uninterrupted code sequence" is a dynamic count in a different sense. Here, it matters whether or not branches are taken. If a branch is taken, the code sequence is interrupted (same for jumps, return, etc.) If, however, it falls through, then it is uninterrupted.
6. Measure something else you find of interest that isn't on this list.

PART II: Branch prediction

Compare one static prediction technique (of your choice) with two dynamic branch prediction techniques. One dynamic branch predictor can be simple, such as two-state branch prediction, and will serve as your baseline dynamic branch prediction performance. The other should be more state-of-the-art, such as correlated dynamic branch prediction or dynamic branch prediction with confidence predictors. We will discuss what these predictors look like in class and you'll have reading on them. Simulate these techniques on your program (using Atom/Etch). Record the number of correct predictions and mispredictions. Be sure to update your branch prediction tables on a misprediction.

For the dynamic predictors, vary the size of the prediction table to gauge the sensitivity of branch prediction performance to prediction table size or vary the size of the confidence thresholds to gauge the sensitivity of branch prediction performance to how confident branch predicting confidence should be.

Measure the performance of your application with each hardware technique. Gather component metrics in addition to baseline performance, so that you can richly analyze your results. If you feel that your metrics do not give you a good indication of performance, what metrics would give you a better view?

PART III: Comparison

Find another group that is using either:

- the same application on a different architecture
- a different application on the same architecture
- the same application on the same architecture

Compare the results of your Parts I and II. When your architectures or applications are different, what conclusions can you draw about the architecture and/or applications used? When they are the same, whose branch predictors did the best and why? This means you will have to scope out your competitors when you choose an application/architecture combination for the first part of the assignment.

The Write-ups

Each write-up should not be more than about 3 pages long. You should pretend you're writing a paper for publication (but omit an abstract). This means that, for instance, you should have an introduction, a methodology section in which you describe your experiments and a results and results analysis section. While we will not necessarily be checking the accuracy of your results, you should still provide enough information to allow us to duplicate your results. For grading purposes, the focus is on what you do with your measurements: What conclusions can you draw? What explanations can you give?

You don't have to restrict yourself to these questions, but we just wanted to give you some idea of what you should be thinking about when doing the write-up. (Again, imagine you're writing a technical paper...) In general, all your HW/project write-ups will have this flavor.

Q1: Explain why you chose the benchmark and why did you think it was interesting? In other words, have some sort of hypothesis here.

Q2: What is your experimental methodology for the measurements you will be taking?

Q3: What do the numbers you measure tell you about the program?

Q4: (already in the handout) Comment on what you find wrt the rationale for static branch prediction schemes.

Q5: What do you conclude from the similarities/differences between the two histograms?

Q6: Say why your measurement is interesting and what that tells you about the program.

Consider this a heads up that you should spend time not just on the implementation but also on the write-up!