Homework 1a - CIS501 Fall 2011

Instructor:	Prof. Milo Martin
Due:	Noon, Thursday, Sept 29th (at the start of class)
Instructions:	This is an *individual* work assignment. Sharing of answers or code is strictly prohibited.
Note:	This is part of a two-part assignment (HW1a and HW1b) due on the same day. Please print out your answers for each part and turn them in as **two distinct stapled documents**.

Overview

In this part of the assignment you will characterize a representative program in terms of some aspects of its ISA. This assignment will also prepare you for subsequent assignments that use the same trace file format.

We've used an x86 "functional" simulator to generate an instruction trace. Your task is to write a program to collect some key statistics of the instructions in the trace file. Your tasks is to use this representative trace to measure the number of micro-ops per macro-op (both average and a distribution), the size in bytes of instructions (both average and distribution), the distribution of branch distances, the mix of various type of instructions, and the performance impact of an ISA change (small branch offsets) and a micro-architectural technique (macro-op fusion).

To do this, you'll write a program that reads in the trace and measures the necessary information. We've provided a skeleton trace parser in both C and Java for you to use as a starting point. You may use languages other than C or Java, but that will require you writing your own parser (highly discouraged). Languages such as Python are likely too slow to process the trace in a reasonable amount of time.

We ask that you include a printout of your source code when you turn in the assignment, but the main focus of this assignment is the experimental results (and not the program source code).

Trace Format and Input Trace

The trace format, our parser code, and the location of the trace is fully described in the Trace Format document.

Question 1 - Micro-ops per Macro-op

a. Calculate the average number of micro-ops per macro-op. The parser code we provided for you already counts the number of micro-ops and macro-ops in the trace, so this is as simple as compiling the code we gave you, running it on the trace, and dividing these two numbers.

b. Modify the code to record the **distribution** of micro-ops per macro-op. Use the data to create a histogram plot of the distribution. The x-axis is the number of micro-ops per macro-op (from 1 to whatever the maximum number of micro-ops per instruction is in the trace). The y-axis is the **percentage** of dynamic macro-ops that have that number of micro-ops. The sum total height of all bars should be 100%. Re-calculate the average based on these numbers to make sure it matches the number you calculated in part (a) above.

Question 2 - Size of Instructions in Bytes

a. Calculate the average (mean) instruction size (in bytes) of the dynamic macro-op instructions in the trace. As this metric is for each macro-op, you should look only at the lines in the trace that indicate it is the first micro-op in a macro-op (that is, the first column is a 1).

   To calculate the instruction size, subtract the instruction's PC from the fallthough PC (both of which are fields in the trace).

b. As with part (b) above, record enough information to generate a histogram of the distribution of instruction sizes. The x-axis is the size in bytes (from 1 to whatever the maximum is). The y-axis is the **percentage** of dynamic macro-ops that are that many bytes in size. The sum total height of all bars should be 100%. Re-calculate the average based on these numbers to make sure it matches the number you calculated in part (a) above.

Question 3 - Distribution of Branch Distances

An important consideration in design of an ISA is how many bits in the instruction to provide to specify PC-relative branch targets. As x86 is a variable-length instruction set, it supports branch offsets of various sizes, including a 32-bit offset.

For each branch instruction (indicated by a non-zero "Target PC" field), calculate the number of bits needed to represent the offset as a signed number. The number of bits need

can be calculated as: 1 + floor(log2(abs(InstructionPC - TargetPC))).

a. As with the previous questions, plot the measured data (x-axis as number of bits needed, y-axis as percent of all branches). However, make this graph a **cumulative** representation of the distribution. This means that each bar represents the number of branches than can be encoded with *n* bits in the instruction. Thus, the left-most bar will be the shortest bar with each bar to the left getting taller. The right-most bar will be at 100%.

b. What percentage of branch targets can be encoded with: 8 bits (1 bytes)? What about 16 bits (2 bytes)?

Question 4 - Instruction Mix

When designing a processor, knowing the mixture of instructions is helpful. From the trace, classify the micro-ops into the following categories:

- **Loads**: If a micro-op's load/store field is 'L', it is a load, else if...
- **Stores**: If a micro-op's load/store field is 'S', it is a store, else if...
- **Unconditional branches**: If a micro-op's target PC field is not zero **and** its flags field is '-', it is an unconditional branch, else if...
- **Conditional branches**: If a micro-op's target PC field is not zero **and** its flags field is 'R', it is a conditional branch, else...
- **Other**: Otherwise, the micro-op is an "other" instruction (which includes adds, shifts, multiplies, etc.)

Plot a histogram of the percentage of each type of instruction (x-axis is labeled with each of these five types; the y-axis is percentage of all micro-operations). The height of all bars should sum to 100%.

Question 5 - Performance Calculation

Consider a modified version of x86 in which branches offsets are allowed to be at most 16-bits. Any branch with larger offsets would require the compiler to insert an extra instruction to first load the target into a register (single microop instruction).

Using the data collected from the previous two questions, how large of an increase in micro-ops would this result in? Give your answer as a percentage.

Question 6 - Operation Fusion

In class we discussed how Intel's processors perform various micro-op and macro-op fusion. Consider a similar optimization in which the processor can combine any "compare"

and "branch" pair in the dynamic instruction stream.

More specifically, any micro-op that writes the flags that is followed immediately by a conditional branch micro-op (reads the flags and has a non-zero target PC field). You can ignore macro-op boundaries when doing this measurement.

a. Modify your code to determine how frequently such pairs occur. As a percentage of all micro-ops, how many pairs of micro-ops are eligible for such fusion?

b. If all micro-ops take a single cycle to execute and the new fused operation all takes a single cycle, how much faster will the processor execute this program (give your answer as a percentage)?

What to Turn In

- **Short answers**. Turn in your answers to the above questions. You must type up your answers.
- **Graphs**. Turn in printouts of the following graphs. One per sheet of paper (to make them big enough to read). Label the axises and give each line a descriptive names in a legend.
  - Graph 1: Distribution of micro-ops per macro-op.
  - Graph 2: Distribution of instruction size in bytes.
  - Graph 3: Cumulative Distribution of branch target distances (in bits to encode).
  - Graph 4: Distribution of instruction mix.
- **Source Code**. Print out your final source code for performing the measurements.

Hints & Comments

- **Runtime**. Processing the traces should take less than a minute (less than 15 seconds with C code, more with Java). As all you are doing in this assignment is recording stats, your extra code shouldn't really slow down the processing significantly at all.
- **Computer resources**. If you wish to write the code by ssh'ing into a Linux box, you can log into minus.seas.upenn.edu. These machines are multi-core multi-Ghz 64-bit chips, so they are quite fast. However, if your jobs run for too long (say 10 minutes) the system may kill off the job (to make sure that users don't use these machines for very long running jobs). Of course, you're also welcome to write and run the code on your personal computers or the lab machines.
- **Code reuse**. In the end, the total amount of code needed to perform these simulations isn't actually that much. Don't go crazy in making the code super modular, but conversely there is significant potential for code reuse that shouldn't be ignored.
- **Total code**. My solution for this assignment added less than a hundred lines of code (including blank lines and comments). If you're writing a lot more code than that, you're doing something wrong.

Addendum

- None, yet.

Generated on: 2011-09-19 20:23 UTC.