Lab 1: Introduction to ISA using SimpleScalar

Julian Garcia

1. Introduction

This lab has students grow accustomed to recording benchmarks using SimpleScalar, specifically using the sim-profile, which is a dynamic instruction profiler. Using this simulator, we're able to evaluate the amount of instructions in a given program as well as the percentages each type of instruction takes up in the total. This allows us to evaluate what instructions each operation is intensive in, as well as compare the amount of instructions each program has in relation to one another, this is especially helpful in evaluating the difference between Alpha and PISA configurations which is done later in this lab.

2. Method

To perform this lab, I repeatedly had to rely on the sim-profile simulator in Simple scalar, and the -iclass argument which enables instruction class profiling, providing the details I needed to fill out all relevant information. I also extracted the benchmark contents into the same directory as my simplescalar installation to be able to easily run the benchmarks.

PART 1:

For the first Part of the Lab I had to analyze 4 programs using methods provided in an included README file, the exact command lines I used for this part and information on each program are detailed below:

1) anagram: a program for finding anagrams for a phrase, based on a dictionary.

Command:

```
./sim-profile -iclass anagram.alpha words < anagram.in > OUT
```

2) compress95: (SPEC) compresses and decompresses a file in memory.

Command:

```
./sim-profile -iclass compress95.alpha < compress95.in > OUT
```

3) go: Artificial intelligence; plays the game of go against itself.

Command:

```
./sim-profile -iclass go.alpha 50 9 2stone9.in > OUT
```

4) cc1: (SPEC) limited version of gcc.

Command:

```
./sim-profile -iclass cc1.alpha -O lstmt.i
```

PART 2:

For the second Part of the Lab I had to analyze 4 programs using ALPHA configuration and PISA configuration.

Since the configuration in the first part of the lab was already using ALPHA, I ran the ALPHA benchmarks first, to change the configuration during the PISA section, I used the following commands in order:

```
make clean
```

```
make config-pisa
```

```
make
```

Information on each program and the command lines I used in both ALPHA and PISA benchmarks are detailed below:

1) test-math: performs various math computations mostly in integer and displays their result.

   For Alpha:

   ```
   ./sim-profile -iclass tests-alpha/bin/test-math
   ```

   For PISA:

   ```
   ./sim-profile -iclass tests-pisa/bin.little/test-math
   ```

2) test-fmath: performs various math functions mostly in integer.

   For Alpha:

   ```
   ./sim-profile -iclass tests-alpha/bin/test-fmath
   ```

For PISA:

```
./sim-profile -iclass tests-pisa/bin.little/test-fmath
```

3) test-llong: performs computations in long format.

For Alpha:

```
./sim-profile -iclass tests-alpha/bin/test-llong
```

For PISA:

```
./sim-profile -iclass tests-pisa/bin.little/test-llong
```

4) test-printf: displays various print statements.

For Alpha:

```
./sim-profile -iclass tests-alpha/bin/test-printf
```

For PISA:

```
./sim-profile -iclass tests-pisa/bin.little/test-printf
```

Recording information in the outputs of each of these commands allowed me to gather the results necessary to answer the lab questions.

3. Results and discussion

Include the items such as lab results and answers to questions included in the lab description. Summarize errors, pitfalls, and problems encountered while doing the lab. Comment on the results and anything requested in the lab description.

PART 1:

Benchmark	Total # of Instructions	Load %	Store %	Uncond Branch %	Cond Branch %	Integer Computation %	Floating pt Computation %
anagram.alpha	25593114	25.36	9.93	4.46	10.30	44.63	5.31
go.alpha	545812220	30.62	8.17	2.58	10.96	47.64	0.03
compress95.alpha	88013	1.54	79.34	0.19	5.68	13.23	0.0
cc1.alpha	48317	16.08	4.57	3.67	13.79	61.78	0.06

Listed below are the solutions to the questions for Part 1:

1) Is the benchmark memory intensive or computation intensive?

anagram.alpha : With an Integer computation value of 44.63% and a floating point computation value of 5.31%, this benchmark is computation intensive

go.alpha: With an integer computation value of 44.63% and a floating point value of 0.03%, this benchmark is computation intensive.

compress95.alpha: With a store value of 79.34%, and a Load value of 1.54%, this benchmark is memory intensive

cc1.alpha: With an integer computation value of 61.78%, and a floating point computation value of 0.06%, this benchmark is computation intensive

2) Is the benchmark mainly using integer or floating point computations?

anagram.alpha : This benchmark is mainly using floating point computations

go.alpha: This benchmark is mainly using integer computations.

compress95.alpha: This benchmark is mainly using integer computations.

cc1.alpha: This benchmark is mainly using integer computations.

3) What % of the instructions executed are conditional branches? Given this %, how many instructions on average does the processor execute between each pair of conditional branch instructions (do not include the conditional branch instructions)

anagram.alpha : 14.76% of instructions executed are conditional branches, given this %
Since (100-14.76) / 14.76 = 5.775,
5.775 instructions are executed by the processor between each pair of conditional branch instructions on average.

go.alpha: 13.54% of instructions executed are conditional branches, given this %
Since (100-13.54) / 13.54 = 6.385

6.385 instructions are executed by the processor between each pair of conditional branch instructions on average.

compress95.alpha: 5.87% of instructions executed are conditional branches, given this %
Since (100-5.87) / 5.87 = 16.036
16.036 instructions are executed by the processor between each pair of conditional branch instructions on average.

cc1.alpha: 17.46% of instructions executed are conditional branches, given this %
Since (100-17.46) / 17.46 = 4.727
4.727 instructions are executed by the processor between each pair of conditional branch instructions on average.

PART 2:

Alpha Results

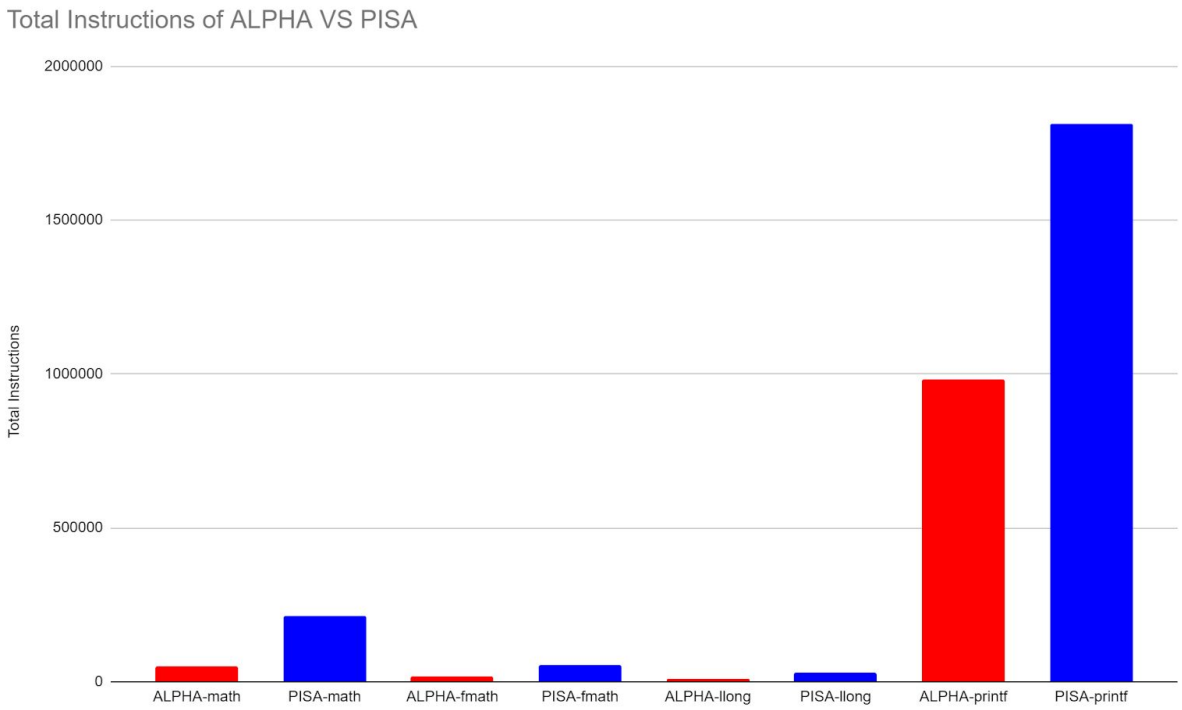
ALPHA Benchmark	Total # of instructions	Load %	Store %	Uncond branch %	Cond branch %	Integer Compute %	Floating compute %
test-math	49280	17.13	10.44	3.95	11.01	55.42	1.88
test-fmath	19369	17.62	12.60	4.73	11.13	53.33	0.43
test-llong	10497	17.62	14.78	5.49	12.15	49.68	0.10
Test-printf	983343	17.99	10.74	4.82	11.39	54.85	0.09

PISA RESULTS:

PISA Benchmark	Total # of instructions	Load %	Store %	Uncond branch %	Cond branch %	Integer Compute %	Floating compute %
test-math	213703	15.96	10.67	4.22	13.84	54.42	0.88
test-fmath	53459	16.14	14.43	4.24	15.09	49.95	0.11
test-llong	29642	16.34	18.02	4.36	15.42	45.81	0.00
test-printf	1813891	19.22	9.28	5.13	17.01	49.33	0.01

Now I compared the two ISAs using histogram plots. I used google sheets to create the histogram. The histograms and my conclusions are listed below:

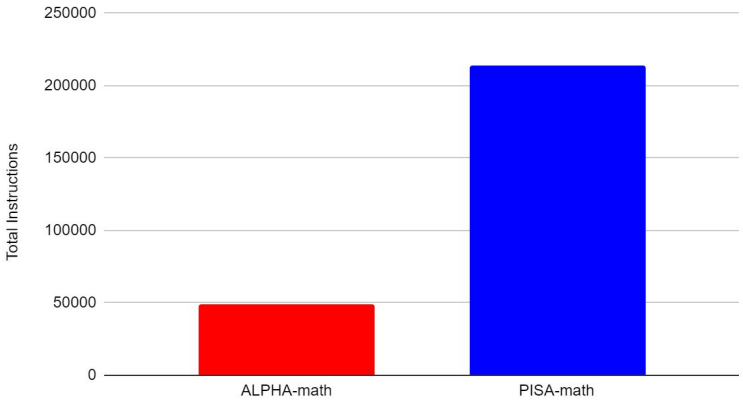
Overall Histogram:



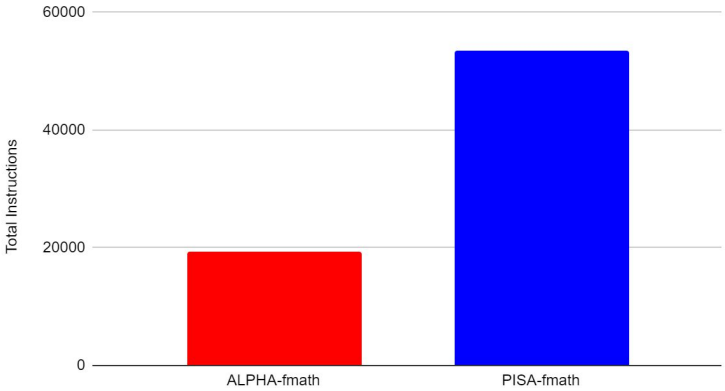
In this Histogram we can see that PISA consistently takes more instructions to complete than ALPHA, and that printf takes the most instructions to complete overall by a wide margin. To better evaluate the difference in PISA and ALPHA, there are Histograms below comparing each program individually.

Individual Histograms:

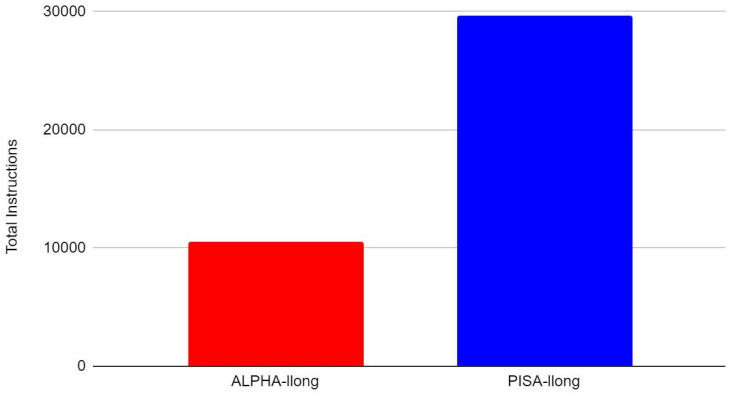
# of Instructions: Pisa Math vs ALPHA Math

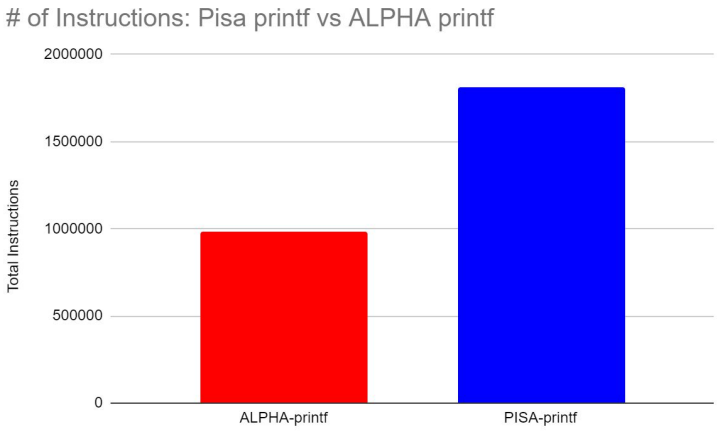


# of Instructions: Pisa FMath vs ALPHA FMath



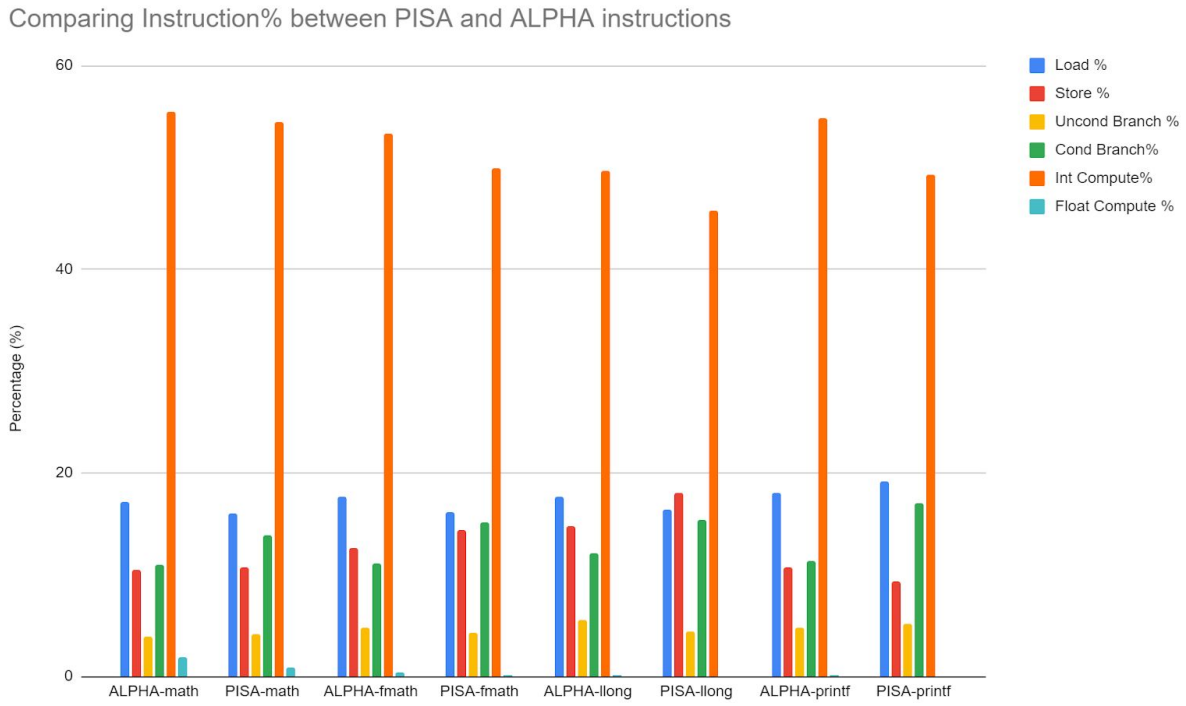
# of Instructions: Pisa llong vs ALPHA llong





After comparing ALPHA and PISA through each of these histograms, it becomes much more clear that PISA takes roughly twice the amount of instructions on average than ALPHA.

Comparison of Instruction Percentages:



In comparing instruction percentages, we see that the distribution of instructions is roughly the same for both ALPHA and PISA.

4. Conclusion

The majority of what I learned in this lab had to do with how to use SimpleScalar to run benchmarks. From installation, to learning the details of sim-profile and which arguments I needed to use to get sim-profile to give me the information I needed to obtain the relevant benchmark data. In finding the arguments I needed for sim-profile, I also learned how the other simulators differed from sim-profile and the exact types of info I could extract from sim-profile. Otherwise, the actual results I obtained from sim-profile taught me how to evaluate benchmarks, and most importantly showed me the difference between PISA and Alpha configurations when it comes to the number of total instructions. The biggest conclusion I obtained from this lab is that PISA configuration costs roughly twice the amount of instructions as the Alpha configuration. I can also conclude that sim-profile is a great method of conducting a benchmark which provides information on which instructions are being performed more than others.