CS146: Computer Architecture
Fall 2019
Homework #1
Due September 18, 2019 (Wednesday) In Class

Collaboration Policy

I think that the following policy is more or less standard in most Harvard CS courses so we will adopt something similar for CS146.

These homework sets will be extremely valuable as tools for learning the material and for doing well on the midterm and final. You are free to verbally collaborate with other students although you should do the following:

(a) State who you discussed the problems with and on which problems
(b) Each student should write out their solution independently and in their own words
(c) Same applies to programming assignments – you can discuss but everyone should do their own coding.

Above all, make sure that you understand the solution to these homework problems. They really are assigned to help you understand the material and be prepared for the types of problems on the midterm and final!

1. **Performance, Speedup**

Programmers are debating about what is the best way to write a program. The three options are:

1) Create a standard, sequential C program and run it on a uniprocessor workstation
2) Create a standard, sequential C program and compile it using a parallelizing compiler, and run it on a 4-way shared memory multiprocessor.
3) Hand-parallelize the problem and run it on a 16-processor shared memory multiprocessor

Consider the table:

Approach	Time to Create Program	Time to compile program once	Time to run program once
1	2 hours	1 second	1 hour
2	2 hours	5 minutes	30 minutes
3	4 hours	5 seconds	6 minutes

a. What is the runtime speedup of approach 2 compared to approach 1?
b. What is the runtime speedup of approach 3 compared to approach 1?
c. If programming, compiling, and run time are all considered, what is the total overall speedup of approach 2 compared to approach 1?

d. If programming, compiling, and run time are all considered, what is the total overall speedup of approach 3 compared to approach 1?
e. If parallelizing the program (as in either approach 2 or 3) can only improve the runtime of the program, then what is the best possible total overall speedup compared to approach #1?

2. **Amdahl's Law – H&P 1.3 (Page 75)**

3. **Multimedia ISAs – Block Match**

Motion estimation in MPEG-1 and MPEG-2 often requires efficient block match algorithms. Block match accumulates the absolute magnitude of the difference of two corresponding values from two 16x16 blocks of data, accumulating the results in a register (Sum of Absolute Differences). Here is an example for two 2x2 blocks of data.

5	4
4	3

3	7
1	3

Block Match = |(5 – 3)| + |(4 – 7)| + |(4 – 1)| + |(3 – 3)| = 8

a. Write standard assembly code to perform the Block Match without using Multimedia Instructions. Assume that the two 16x16 blocks of data are already in memory. Do not worry about performing loop unrolling or software pipelining.

b. Write assembly code to perform the Block Match with Multimedia Instructions (use the MAX-2 ISA instructions) using Loop Vectorization and In-Line Conditional Execution (Saturating Arithmetic) to speedup the computation.

c. Compare the number of instructions needed in both implementations.

4. **Basic Pipelines – H&P A.1 (Page A-81)**

Assume that branches resolve in the MEM phase for this question.

5. **SimpleScalar Assignment**

a. Go to http://www.simplescalar.com look at the "Documentation" under the Support category. Browse through the documents available there – in particular, the user's guide (v2 is fine) and hacker's guide (v2) will be useful.

b. We will run the tools off of the FAS compute systems for this first assignment (nice.fas.harvard.edu and ice.fas.harvard.edu). The tools should work fairly seamlessly on either the Linux boxes or the Alpha's. The tools and benchmarks are both hosted under the cs146 directory. You will need a local

copy of the tools, but the benchmarks can run out of the course directory. These are the files/directories of interest:

- ~cs146/simplesim-3v0d.tgz (SimpleScalar v3.0d sources)
- ~cs146/benchmarks/ (4 benchmarks with reduced inputs, used for this assignment, both PISA and Alpha binaries)
- ~cs146/spec2000binaries.alpha/ (SPEC2000 pre-compiled Alpha binaries)
- ~cs146/spec2000binaries.ss/ (SPEC2000 pre-compiled PISA binaries)
- ~cs146/xbenchmarks/ (Pre-compiled Alpha binaries for Doom, Povray, and some other X11 Benchmarks)
- ~cs146/quake106/ (Pre-compiled Alpha binaries for Quake)

For this assignment we will only need the first two of these.

Make a copy of simplesim-3v0d.tgz and gunzip/un-tar it into a local directory in your account. The tools can be compiled to support simulation of PISA or Alpha-ISA binaries.

To compile for Alpha-ISA use the command "make config-alpha" and run "make". This will generate sim-* where * is bpred, cache, cheetah, eio, fast, outorder, profile, and safe. Make a copy of sim-profile. You can also change the Makefile to support a higher optimization setting (-O3) with the gcc compilation.

For this assignment we will use four simple benchmarks, "anagram", "cc1" (gcc), "compress", and "go". These are in ~cs146/benchmarks – the README in this file describes how to run the benchmarks. Some of these benchmarks may take a bit of time to run, depending on which simulator is used.
You may want to write a shell/perl script to automate running the benchmarks.

c. Use sim-profile to collect instruction profiles, instruction-class profiles, branch profiles, and address mode profiles for the four benchmarks for the Alpha-ISA binaries. Summarize the results in your report -- do not just print out the dump from the simulator – mention some interesting characteristics (top 10 instructions, popular addressing modes, etc) that these benchmarks utilize.