CSE30321 Computer Architecture I
Lab 3: Processor Performance Analysis and Extending the Simple Processor
Assigned: September 15. Due: September 24

A. Objectives

- Become familiar with benchmarking processor performance
- Gain understanding in the impact of compiler optimizations on performance
- Practice processor design in Verilog

B. References

- A basic Linux tutorial (links on course web page)
- Textbook by Patterson and Hennessy
- Benchmark Documentation (http://www.spec.org/cpu/CINT2000/)
- Textbook by Vahid (Ch. 8, 9)
- CSE20221 Handouts on Verilog (links on course web page)

C. Analyzing Processor Performance

C.1 What to Accomplish

- Execute given benchmarks and collect data
- Analyze collected data for performance comparisons

C.2 Procedure

1. Copy the three benchmark directories (dhry, parser, gzip) from
   (/afs/nd.edu/courses/cse/cse30321.01/www/labs/lab3/) to a directory in your AFS space. You should find the following three benchmarks:
   - dhrystone - a synthetic integer benchmark
   - gzip - a compression tool using Lempel-Ziv coding
   - parser - a link grammar syntactic parser for English
2. Compile and run all benchmarks without compiler optimizations. To assist the compilation, you are given a sample Makefile for each benchmark. Make any necessary adjustments before using it. Use the following input(s) for each benchmark. Run the dhrystone and gzip benchmarks first so that you get a feel for how to do this from the command line. Also practice

with the `time` command to time program execution times. (Consult the Linux tutorial posted for more information or talk to a TA.)

Benchmark	piped input	input file	input value
DHRYSTONE			50 million
GZIP		input.graphic	
PARSER	test.in	2.1.dict	

If you are running the bash shell, it has a different built-in time command. Use the `time` command located at /usr/bin/time. Always pipe the output of your tests to file so that you can have a record and so that you can go back to the results later. Record user time for all runtimes.

Once you get the hang of it, compile and test each benchmark on two different machines of your choice. (Make sure that the machines contain different processor models.) Modify Makefile to help with this process. For each program and machine, record 3 tests in the following tables.

Iteration	Machine 1 MIPS Rating	Machine 2 MIPS Rating
1st Run		
2nd Run		
3rd Run		
Average		

For Machine 1:

Iteration	GZIP Runtime	PARSER Runtime
1st Run		
2nd Run		
3rd Run		
Average		

For Machine 2:

Iteration	GZIP Runtime	PARSER Runtime
1st Run		
2nd Run		
3rd Run		
Average		

3. Record the executable file size of the parser benchmark compiled for Machine 1. Recompile the parser benchmark with two different compiler optimizations (O2 and Os ("Oh-2" and "Oh-s")). For each recompile, record the compile time, the new executable size, and the

benchmark runtime.

Opt. Level	1st Runtime	2nd Runtime	3rd Runtime	Avg. Runtime
No Opt.				
-02 Opt.				
-0s Opt.				

Opt. Level	Compile Time	File Size
No Opt.		
-02 Opt.		
-0s Opt.		

C.3 Analysis

1. Based on the MIPS rating you calculated from the Dhrystone benchmark and the clock frequency, compute the CPI for both machines.

Platform	Clock Freq.	Avg. MIPS Rating	Functional CPI
Machine 1			
Machine 2			

2. Using the MIPS rating for each machine and the average runtimes for the non-optimized tests, calculate the approximate number of instructions executed for each benchmark.

   For Machine 1:

Benchmark	Avg. MIPS Rating	Avg. Runtime	Approx. Instr.
GZIP			
PARSER			

   For Machine 2:

Benchmark	Avg. MIPS Rating	Avg. Runtime	Approx. Instr.
GZIP			
PARSER			

C.4 Questions

1. Using the average O2 optimized runtime for the parser benchmark and the MIPS rating, calculate a new instruction count for the benchmark on each machine. How does it compare to the earlier instruction count? What could account for the difference? What does this say about the MIPS rating and the compiler?

2. Is CPI always a constant for the same machine? Is it possible that Intel runs favorable benchmarks to produce lower than "normal" CPI when producing its marketing material. Considering the clock rate and CPI, what can you say about the two machines?

3. Look at the differences in runtime and executable size based on the different compiler optimizations. What does this say about the different optimizations? Is there an optimization level that is superior in all categories?

4. For each benchmark there are many input files given. For example, the gzip benchmark comes with five input files called "input.graphic", "input.log", "input.program", "input.random", and "input.source". Based on you knowledge of the gzip benchmark, why are different input files included and why these specific five?

5. Take a look at the current list of SPEC programs on p.260 of the textbook. Consider how you typically use your computer. What classes of programs are irrelevant or missing? Why do you think they were or were not included in the SPEC2000 collection?

D. Extending a Given ISA

D.1 What to Accomplish

- Add two new instructions `jmpn` and `jmp` to the six-instruction ISA discussed in class.
- Verify and debug the new processor design with ModelSim or the ISE simulator.

D.1 Procedure

1. Open ISE and create a new project called "lab3" in an empty directory. Download and extract the zipped Verilog files from the course web page into a different directory. In ISE, add the Verilog files by clicking "Add Copy of Source..." under the "Project" menu.
2. Add the new instructions to the processor by adding states to the controller and making any other necessary changes.
3. We have discussed `jmp` in class. For `jmpn`, it is used as `jmpn Ra OFFSET` and accomplishes the following:

   if Rf[a] < 0
   PC=PC+OFFSET
   else
   PC=PC+1

4. Write a simple program in assembly to test the new opcodes with your Verilog processor.
5. Test your program in ModelSim or the ISE simulator and print out parts of the waveform that demonstrate the functionality of the instructions. If the instructions do not work as intended, use the simulator to determine why, and then fix the problems.
6. Record the # of cycles each new instruction takes.

E. What to Turn In

For this lab, no demonstration is required. Your report should include the following:

- For the performance analysis part, a short summary of the experiments, details of your results and calculations, and discussion of the questions raised.
- For the processor design part, a description of the lab process, problems encountered, and how they were resolved,
- The waveforms with appropriate explanation demonstrating that you properly implemented the new instructions. Be sure to annotate all waveforms to show key points of the simulation.
- Cycle counts for the new instructions.