# EEL 6935 Embedded Systems – Fall 2011

# Assignment 1

# SimpleScalar

Assigned: 9/24/11

Due date: 10/14/11 @ 8 PM via Sakai

Since this assignment is very hands-on and you cannot learn the tools unless you actually follow the steps, all work must be done ***individually***. ***No group work allowed***.

In this assignment you will:

1. Install SimpleScalar [4] and the SimpleScalar gcc cross-compiler
2. Compile and test the MiBench [2] embedded systems benchmarks
3. Run SimpleScalar to gather statistics and analyze benchmarks
4. Modify SimpleScalar to gather additional statistics

## 1. Installing SimpleScalar and the gcc cross-compiler

The installation instructions are based on [1] and have been tested on the grid.ecel.ufl.edu Redhat x86_64 machines. You are welcome to install these tools on your own Linux OS machine or on Cygwin, however, the TA will not give you installation help for problems associated with these installations. You will have to debug your installation using web resources.

### *Preparation*

Create a "*simplescalar*" folder in your home directory and store the following files in your *simplescalar* directory.

| | |
|---|---|
| **simpletools-2v0.tgz** | http://www.simplescalar.com/tools.html |
| **simplesim-3v0d.tar.gz** | http://www.igoy.in/wp-content/uploads/2009/09/simplesim-3v0d.tgz |
| **simpleutils-990811.tar.gz** | http://www.igoy.in/wp-content/uploads/2009/09/simpleutils-990811.tar.gz |
| **gcc-2.7.2.3.ss.tar.gz** | http://american.cs.ucdavis.edu/RAD/gcc-2.7.2.3.ss.tar.gz |

Set up environment variables for installation:

**$ export IDIR=/home/username/simplescalar**
**$ export HOST=i686-linux-gnu**
**$ export TARGET=sslittle-na-sstrix**

Note: If you do not use grid.ecel.ufl.edu you may be required to install/update *build-essential, bison,* and *flex* using "sudo apt-get install <package name>".

### *Install SimpleTools*

**$ cd $IDIR**
**$ tar xvfz simpletools-2v0.tgz**
**$ rm -rf gcc-2.6.3**

### *Install SimpleUtils*

**$ cd $IDIR**
**$ tar xvfz simpleutils-990811.tar.gz**
**$ cd simpleutils-990811**

Update the *simpleutils-990811/ld/ldlex.l* file before compiling:
Replace all instances of *yy_current_buffer* with *YY_CURRENT_BUFFER*

**$ ./configure --host=$HOST --target=$TARGET --with-gnu-as --with-gnu-ld --prefix=$IDIR**
**$ make CFLAGS=-O**
**$ make install**

### *Install Simulators sim-**

**$ cd $IDIR**
**$ tar xvfz simplesim-3v0d.tgz**
**$ cd simplesim-3.0**
**$ make config-pisa**
**$ make**

To test the installation you may use:
**$ ./sim-safe tests/bin.little/test-math**

### *Install the gcc cross-compiler*

**$ cd $IDIR**
**$ tar xvfz gcc-2.7.2.3.ss.tar.gz**
**$ cd gcc-2.7.2.3**
**$ ./configure –-host=$HOST –-target=$TARGET –-with-gnu-as –-with-gnu-ld –-prefix=$IDIR**
**$ chmod –R +w .**

Modify the following files:

- Append *–I/usr/include* to the *Makefile* at line 130
- Replace *#include <varargs.h>* with *#include <stdarg.h>* in *protoize.c* at line 60
- Change **((void **)__o->next_free)++=((void *)datum);* to **((void **)__o->next_free++)=((void *)datum);* in *obstack.h* at line 341

Copy the following files:

**$ cp ./patched/sys/cdefs.h ../sslittle-nasstrix/include/sys/cdefs.h**
**$ cp ../sslittle-na-sstrix/lib/libc.a ../lib/**
**$ cp ../sslittle-na-sstrix/lib/crt0.o ../lib/**

Build the compiler using:

**$ make LANGUAGES=c CFLAGS=-O CC="gcc -m32"**

To fix the compilation errors:

- Append '\' to *insn-output.c* at lines 675, 750, and 823
- Execute **make LANGUAGES=c CFLAGS=-O CC="gcc -m32"** again
- Fix any other errors you find and run **make LANGUAGES=c CFLAGS=-O CC="gcc -m32"** again
- Finally, remove lines 2978-2979 in *cxxmain.c*

To finish installing the cross-compiler:

```
$ make LANGUAGES=c CFLAGS=-O CC="gcc -m32"
$ make enquire
$ ../simplesim-3.0/sim-safe ./enquire -f > float.h-cross
$ make LANGUAGES=c CFLAGS=-O CC="gcc -m32" install
```

The cross-compiler and tools are found in your *$IDIR/bin* folder. To test your cross-compiler create a test program such as hello.c

```
#include<stdio.h>
main()
{
printf("Hello World!\n");
}
```

Compile for SimpleScalar

```
$ $IDIR/bin/sslittle-na-sstrix-gcc  –o hello hello.c
```

Run on sim-safe

```
$ $IDIR/simplesim-3.0/sim-safe hello
```

### *Questions*

1. What is the purpose of appending *–I/usr/include* to the *Makefile*?
2. Why do we need the –m32 gcc compiler option? If you did not use grid.ecel.ufl.edu or a similar machine, explain any gcc options you used.
3. Explain why *#include <varargs.h>* should be replaced with *#include <stdarg.h>.*

## 2. Compiling and testing MiBench

Download and extract the source code for the Network and Telecomm MiBench benchmarks from the website http://www.eecs.umich.edu/mibench/. This website also contains the compilation instructions for MiBench and sample outputs.

There should be six benchmarks in all: Network has two benchmarks (dijkstra and patricia) and Telecomm has four benchmarks (adpcm, CRC32, FFT, and gsm).

Choose five of the six benchmarks and cross-compile them for SimpleScalar. Execute each benchmark on sim-safe with the small data set and verify that the benchmarks execute correctly by comparing your output with the sample outputs from the MiBench website.

Most outputs are in .txt or .dat formats so you can compare the contents of your test output to the sample output. If the output is in a format which cannot be opened, such as a .pcm file, you should at least verify that the test output file is the same size as the sample output file.

Create a file called *verify_mibench.txt* to record the following information for each benchmark:
The benchmark name
The command executed: sim-safe <benchmark name, options, etc.>
The number of instructions executed
The changes made to makefile
Did you verify that the benchmark executed correctly (yes or no)?

## 3. Analyzing benchmarks

Your goal is to optimize the cycles per instruction (CPI) for your five benchmarks.

### *Simulations and results*

Execute each benchmark on sim-outorder varying the following parameters:
dL1 cache size: 2K, 4K, 8K
iL1 cache size: 2K, 4K, 8K
processing: in order, out of order
branch prediction: not taken, taken, perfect
All other parameters must be kept constant.

Report the configuration chosen for your unified L2 cache, and the line size and associativity chosen for your L1 caches.

For each benchmark create a file *benchmarkname_analysis.txt.* In that file report the optimal, lowest CPI, configuration for the benchmark, as well as the CPI, number of instructions, dL1 cache miss rate, and iL1 cache miss rate for the 54 configurations.

I suggest writing a script to automate your benchmark execution and to create your *benchmarkname_analysis.txt* files. Perl scripts are fairly easy to create (you can find a link to a comprehensive Perl user guide in [3]), however, you are free to use any scripting language. A sample Perl script *run_sim_safe.pl* for executing the *hello* executable on sim-safe is shown below:

```
#!/usr/bin/perl

$bmark = "/home/username/hello";
system ( "/home/username/simplescalar/simplesim-3.0/sim-safe $bmark" );
```

### *Analysis*

Write a brief report (no more than one page) analyzing the benchmarks and CPI optimization. Your report should answer questions such as: Does varying branch prediction have a large effect on CPI? Why or why not? What about the dL1 cache size? Which parameter has the largest impact on CPI? Why? etc.

## 4. Modifying SimpleScalar

In the final part of this assignment you will add a command line option, statistic, and formula to the sim-cache simulator.

You system will have:
dL1 cache – an 8KB, 4-way associative, 64 byte line size
iL1 cache – 4-way associative, 64 byte line size. You will vary the cache size – 2KB, 4KB, 8KB

L2 cache – none

Modify sim-cache to calculate the "instruction cache cycles per instruction (icache_CPI)" where *icache_CPI* is defined as *icache_cycles* / *number of instructions executed*.

You will add a counter called *icache_cycles* which must be incremented for each instruction:
For a 2KB cache – increment *icache_cycles* by 1 for an icache hit, and by 10 for an icache miss
For a 4KB cache – increment *icache_cycles* by 2 for an icache hit, and by 15 for an icache miss
For an 8KB cache – increment *icache_cycles* by 4 for an icache hit, and by 20 for an icache miss

Modify sim-cache to include *icache_cycles* in the statistics printed at the end of the simulation.

Since *icache_cycles* is based on the instruction cache size, you must add a command line option with arguments to specify the instruction cache size (in bytes) at run time. For example
-icache_size:bytes 2048
for the 2KB instruction cache.

Finally, modify sim-cache to include *icache_CPI* in the statistics printed at the end of the simulation. Hint: do this by registering a formula called *icache_CPI* which uses *icache_cycles* and *sim_num_insn*.

Execute your five benchmarks on the modified sim-cache for a 2KB, 4KB, and 8KB instruction cache.

## 5. What to turn in

You must submit the following via Sakai in a zipped file named *Lastname_Firstname_SS*:

- Answers to the questions in section 1.
- The *verify_mibench.txt* benchmark report from section 2.
- Your *benchmarkname_analysis.txt* files and analysis for section 3.
- The following for section 4:
  - Your modified *sim-cache.c* file (commented) along with any other files you decide to modify
  - A summary of changes. Sample format: *sim-cache.c* line 130 – added *icache_cycles* counter
  - For each benchmark create a file *benchmarkname_modified.txt.* The file should contain:
    - the command executed: sim-cache <benchmark name, options, etc.>
    - the number of instructions, il1 cache hits, il1 cache misses, icache_cycles, and icache_CPI as it is printed by sim-cache

## References


[1] Brorson, Mats. Simplescalar instruction guide. http://www.kth.se/polopoly_fs/1.36445!/Simplescalar-installation-instructions.pdf
[2] MiBench a free, commercially representative embedded benchmark suite http://www.eecs.umich.edu/mibench/
[3] Marshall, A. D. Practical Perl Programming. http://www.cs.cf.ac.uk/Dave/PERL/
[4] SimpleScalar Tools and Documentation http://www.simplescalar.com/