ECE 462/562
Homework 1

Due: Wednesday, February 12, 11.59pm.

- *Work in your project groups. Only one submission is required per group, but I recommend that each group member does the full assignment.*
- *This homework assumes some basic knowledge of Linux commands (e.g., cd, tar, etc.) If you don't know how to do something, Google it, or Bing it, or ask your colleagues, or post on Piazza.*

GEM5 simulator is one of the most popular open-source computer architecture simulators that provides a robust way to model and evaluate new designs. In this homework, you will familiarize yourself with the basics of GEM5 (and architecture simulation) and how to use it for simple simulations and application profiling.

1. **Read** about the GEM5 simulator on the GEM5 website: http://gem5.org/Introduction

2. **Download the Virtual Machine** image containing the GEM5 installation from https://uweb.engr.arizona.edu/~tosiron/ece562_spring20/downloads.php. You can run the image using any virtual machine client of your choice, e.g., VMWare or Virtual Box (free, recommended) or Windows 10 Hyper-V (I'm not crazy about this one). If you choose to use the virtual machine, it saves you the need to install GEM5 and its dependencies.

   If you're not familiar with virtual machines, Google is your friend, and it's a pretty nifty invention! ☺

   Username: **comparch**
   Password: **awesomet**

   You may also install GEM5 on your own using the instructions in *Step 3*. If you use the virtual machine, you may skip *Step 3.*

3. **Install GEM5** on your Linux computer or virtual machine. You may also use the ECE3 (ece3.ece.arizona.edu) server, but I suggest you exhaust all other alternatives before resorting to the ECE3 server (lots of issues with that server!!). To remotely access the ECE3 server, I recommend PuTTY for SSH and WinSCP for FTP.

   You may also use a virtual machine like VMWare or Virtual Box on which a Linux OS (I recommend Ubuntu) is installed.

   I do not recommend trying to install directly on Windows or Mac OS (this could lead to a lot of painful experiences and disappointment in your choices).

   To install GEM5 dependencies, several of them can be installed using build-essential:

```
>apt install build-essential
```

   Others like *m4, zlib1g-dev, scons, python-six, python-dev* can be install directly using apt install.

```
>apt install m4 zlib1g-dev scons python-six python-dev
```

GEM5 documentation can be found on the GEM5 website: http://gem5.org/Download. Install the simulator as follows:

- *Clone the GEM5 repository somewhere in your home directory*

```
> git clone https://gem5.googlesource.com/public/gem5
```

- *Install the **ARM simulator***

```
> cd gem5
> scons build/ARM/gem5.opt
```

This installation will take several minutes (> 1hr on the ECE3 server), so you may take a power nap while installation takes place.

- *Test the simulator*

```
> build/ARM/gem5.opt configs/example/se.py -c tests/test-
progs/hello/bin/arm/linux/hello
```

If everything works correctly, this should print out "Hello world!" and an Exiting @ tick… message. A stats.txt file, containing the execution statistics, should be stored in *m5out* folder.

4. Test Benchmarks

If you're installing GEM5 on your own, perform *steps a. to c*. If you're using the provided virtual machine, those steps have already been performed, so you may skip to *step e*.

a. Download the benchmarks.zip file from Piazza and unzip into your home directory. The provided virtual machine already contains the benchmarks in the /home//benchmarks directory.
b. Replace the configs/example/se.py file with the provided se.py file on Piazza (I recommend that you keep a copy of the old one). Some changes have been made to include benchmark options. The file is commented with where the changes are made ("#Tosi's changes for ECE 462/562 from here!!!").
c. Modify Line 150 to point to the full path to your benchmarks directory from the prior step. On the server, the path would be something like `/home/netID/benchmarks/.` Don't forget to include the '/' at the end.
d. Five benchmarks from two benchmark suites are provided: two from SPEC2006 (*mcf* and *libquantum*) and three from EEMBC (*a2time01, cacheb01,* and *bitmnp01*). These benchmarks perform different functions, which you may read about on the SPEC and EEMBC websites (Bing them!).

   The benchmark binaries provided have been cross-compiled for the ARM ISA, since we are simulating an ARM architecture.

e. Test the benchmarks using the following command (from the gem5 directory):

```
>build/ARM/gem5.opt  configs/example/se.py  --maxinsts=1000000  --bench
<benchmark>
```

Replace <benchmark> with the name of a benchmark (e.g., *mcf*). Maxinsts specifies a maximum number of instructions to run. This command uses a small number to test. You can see other possible simulation options by running gem5.opt and se.py as shown above, without any options.

If everything works fine, the simulation should end with "Exiting @ tick… because a thread reached the max instruction count"

The stats are stored by default in m5out/stats.txt (this can be changed, if you want, using the '--stats-file' option)

5. **Profile Benchmarks/CPU – PART 1**

   Now, run all five benchmarks using a more complex single core (n=1) processor model, called O3CPU (some details here: http://gem5.org/O3CPU), which is an *out of order* (OoO) CPU. We will discuss how OoO CPUs work in class later.

   Using the following instruction, run simulations for all five provided benchmarks:

```
>build/ARM/gem5.opt --stats-file=<$statsFileName> --dump-config=<$location/config.ini>
configs/example/se.py --caches --l1i_size=32kB --l1i_assoc=4 --l1d_size=32kB --l1d_assoc=4
--cacheline_size=64 --cpu-clock=1.6GHz --cpu-type=O3_ARM_v7a_3 -n 1 --maxinsts=100000000 --
bench <$benchmark>
```

   For the cache configurations, use 32kB size, 4-way set associative, and 64B line size for both instruction and data caches. Maxinsts is 100million instructions (still a very small portion of instructions, especially for the SPEC benchmarks, but suffices for this homework).

   The specified options override the defaults. Only the options in '< >' need to be changed. Cache sizes should be in 'kB' (case sensitive). Line size (in bytes) and associativity (in ways) should both have *no* unit specified. The '--stats-file' and '--dump-config' are optional, but recommended; they specify where to store stats and config files for each simulation.

   Using your simulations, fill out all the requested information in **PART 1** of the homework1_deliverables.docx file. A Word file is provided so that you can directly write into the tables. You may also print and handwrite your answers.

6. **Impact of the L2 Cache – PART 2**

   In this part of the assignment, you will explore the impact of augmenting the processor design with a level two (L2) cache. Since one of the goals of a multi-level cache hierarchy is to reduce the average memory access time (refer to slides on cache optimizations), we want to investigate if, and by how much, an L2 cache improves the performance for the different benchmarks.

   Using the five provided benchmarks, run the simulations with an L2 cache as follows:

```
>build/ARM/gem5.opt --stats-file=<$statsFileName> --dump-config=<$location/config.ini>
configs/example/se.py --caches --l1i_size=32kB --l1i_assoc=4 --l1d_size=32kB --l1d_assoc=4
--cacheline_size=64 --l2cache --l2_size=1MB --l2_assoc=8 --cpu-clock=1.6GHz --cpu-
type=O3_ARM_v7a_3 -n 1 --maxinsts=100000000 --bench <$benchmark>
```

   Note that the instruction now includes the l2 cache commands. We are assuming a 1MB L2 cache with 8-way set associativity, and 64B line size.

   Compare the new results with your results from PART 1. For each benchmark, fill in **PART 2** of *homework1_deliverables.docx* file with the required information to illustrate the benefits of the L2 cache.

   DELIVERABLE: Convert the *homework1_deliverables.docx* file to a **pdf** file with the same name before submitting in the appropriate D2L Assignments folder. ***Only pdf files will be graded***.