Lab 04 - Caches

Introduction

In this lab, you will be exploring cache design trade-offs. You will build a number of different caches, and see how these design choices affect the number of memory accesses. A sample simulation program, written in C, can be downloaded here and a version in C++ can be downloaded here.

Note: You can use a modified version of the PTLSIM simulator to get an executable's memory trace. This executable should work on any compiled C/C++ application. To use it call `./ptlsim` with your executable as an argument (i.e. "`$./ptlsim a.out`"). The simulator should output two files `ptlsim.log`, and `ptlsim.cache`. The `.cache` file will hold a trace of instruction and data loads for your executable. PTLSIM is used to generate traces of any compiled program you have installed. You can generate traces if you want to test your simulator on multiple programs, but we are only grading the performance on the trace below.

Deliverables

You should build a cache simulator that reads one address trace file and simulates multiple cache architectures and reports the miss rate (# misses / total accesses). Your simulator should support all the following attributes:

- 32-bit addresses
- **Block Size**: 16 elements
- **Replacement Policies**: LRU, FIFO
- **Cache Sizes**: 1024, 2048, 4096, 8192, 16384 locations
- **Associativity**: Direct Mapped, 2-way, 4-way, and 8-way

The input to your simulator will be the following, given as arguments on the command line:

- Associativity as 1, 2, 4, or 8
- Cache size as 1024, 2048, 4096, 8192 or 16384
- Replacement policy as LRU or FIFO (your program only need work with these exact cases)
- An input file produced by PTLSIM with a trace of memory locations that is redirected to your executable.

An example of the command line would be:

```
./cache_example 2 8192 LRU < trace
```

This command line would then output the miss rate, as specified below, for a 2-way cache of size 8192 blocks using the LRU replacement policy for the addresses in the file name trace.

Your program should only validate the input as far as not crashing if the command line specifies a value other than those specified above.

The output from your simulator should have the following format.

- First output the associativity, 1, 2, 4, or 8, on it's own line
- Next output the cache size, 1024, 2048, 4096, 8192 or 16384, on it's own line
- Next output the replacement policy, LRU or FIFO, on it's own line
- Finally, output the miss rate as a percentage for example 5.72 for a 5.72% miss rate

You will build your simulator in C/C++. If you want to use a different language please check with the TA first. Your program should read from standard input, and write to standard output. To test your simulation you can use the memory trace file here. The output for this file should look like for the above give command line:

```
associativity:      8
cache size:         16384
replacement policy: LRU
miss rate:          5.72
```

You may use any data you want (as produced by the PTLSIM tool mentioned above, but to test if you're simulating the caches correctly the following values will be used as part of the autograder. The best practice would be to test your code for all the possible inputs against the given memory trace and make sure it matches exactly the values below.

LRU Replacement Policy					
	1024	**2048**	**4096**	**8192**	**16384**
1	55.01	42.07	29.30	20.74	13.91
2	51.58	36.44	23.70	13.98	08.49
4	48.85	33.97	20.04	11.33	06.37
8	47.44	32.20	18.63	10.02	05.72

FIFO Replacement Policy					
	1024	**2048**	**4096**	**8192**	**16384**
1	55.01	42.07	29.30	20.74	13.91
2	53.31	38.32	25.47	15.37	09.49
4	51.86	37.07	23.11	13.67	07.86
8	51.14	35.98	22.40	12.79	07.44

Turn-In

Each student should turn in one zip file to Gradescope. Your work can be (and should be) identical to the other members of your group. The contents of which should be:

- A README file with the group members names, and any incomplete or incorrect functionality
- The source code for your simulator
- A makefile to compile your code on the school's server. When compiling I should only have to type "make"..