CS 352 – Prof. Mark End of semester Project Due May 4, 2007 5:00pm

Objective: Build a Level 1 data cache simulator and run a set of experiments using your simulator.

Authorship: You may use the pair programming approach for this assignment; this means actively working together on the assignment rather than just dividing the work in two. You can choose your partner; send email to the TA or to the class email list if you're having trouble finding someone who isn't matched up already. Alternatively, you may work on the project by yourself if you wish.

Due dates: Names of partners are due by the start of class on **April 12**. Send email to impjdi@cs.utexas.edu to name your partner or to indicate that you will be working on the assignment by yourself. The assignment itself is due at **5:00 pm on the last day of classes (Friday May 4th)**. If you turn in the assignment late, BOTH partners must have the necessary number of late days. Assignments turned in beyond the late-day limit might not be graded at all – you have been warned! With many programming assignments to grade in a short period of time before grades are due, we have to be strict about these deadlines.

Specifications:

1) You can write your program in either Java or C++ -- your choice. It must run on the CS department linux machines. For information on UTCS Java, see
http://facilities.cs.utexas.edu/htdoc/documentation/java.html
For information on UTCS C++ see
http://facilities.cs.utexas.edu/htdoc/faqs/programming.html#83

2) Your Level 1 Data Cache Simulator should accept the following command-line options:

-c <capacity>	with <capacity> in KB: 4, 8, 16, 32, or 64.
-b <blocksize>	with <blocksize> in bytes: 4, 8, 16, 32, 64, 128, 256, or 512.
-a <associativity>	where <associativity> is integer size of set: 1, 2, 4, 8, 16.

The baseline configuration (i.e. the first one that you should test) is:
capacity = 8K, blocksize = 16 bytes (i.e. 4 words), set associativity = 4
i.e. the command line is:

```
cache_sim -c8 -b16 -a4
```

or:

```
java cache_sim -c8 -b16 -a4
```

3) Additionally, your cache should have the following functionality:
* write back (for write hits)
* write allocate (for write misses) – read the cache line, modify it with the write, then mark it as dirty.
* LRU replacement

4) The input to the cache simulator will be a sequence of memory access traces, one per line (with 2-3 integers per line), terminated by end of file.

   For loads the format will be:
   0 <address>

   For stores the format will be:
   1 <address> <dataword>

   Each address will be the address of a 32-bit data word. The address will be word aligned (i.e. divisible by 4). The address and data are expressed in hexadecimal format.

5) To support your testing of the data cache simulator, you should also implement a simple model of main memory. The capacity of the memory should be 4 MB (1 megaword). At simulator start-up time, initialize the contents of each word to be the word's address. For example, the 32-bit word that starts at byte 0x00001004 should be initialized to 0x00001004.

6) The output of your program will consist of a set of statistics gathered during the simulation as well as the contents of the cache and memory at the end of the simulation. The format of the output is described in detail at the end of this document. You must use the code fragments available for download on the project web page
   http://www.cs.utexas.edu/users/billmark/teach/cs352-07-spring/project
   to print the output – this will guarantee a consistent output format that we can use for grading.

   Framework has been provided for both C++ and Java versions of this code. These files will help with input and output, letting you get to the interesting code faster.

   In C++:

   Download and use the following files: Makefile, *.h, *.c.

   In Java:

   Download and use the file cache_sim.java.

7) Your program will be graded for correctness and for design.

   For Design: I expect to see well chosen classes that reflect the application. Be sure that your method names and variable names reflect memory hierarchy terms. When done, anyone reading your code should have learned how a cache works. So, towards that end, it may help to define constants such as "dirty".

   For Correctness: You should create your own trace files to help you test and debug your cache configurations.

9) Additional detailed specifications:

   a) You should initialize all of the cache locations to be 'invalid' and initialize all of the tag and data fields to zero.

   b) All of the parameters on the command line are base 10 numbers (e.g. 32, 64, 128, etc.)

   c) The addresses and data in the trace files are expressed as base 16 (hexadecimal) numbers. Note that these numbers might (or might not) have extra zeros in front… e.g. 00de2a vs. de2a.

d) Read your input from 'stdin' and write your output to 'stdout'.
(i.e. read from the terminal and write to the terminal)

e) Use the unix shell's ability to redirect input and output to pull input from disk files and write output to disk files. For example:

csh> cache_sim –c 32 –b 16 –a 4 < test.trace > test.output

f) Your output should be formatted as described below. You are to print out the entire contents of the cache, and 1024 words of memory starting at address 0x003f7e00.

Test Cases:

We are providing four small test cases on the project web page. We are not providing the solutions, but the test cases are small enough that you can hand calculate the results. Once your program is complete and passes the simple test cases, you can use the trace file from the compress benchmark (cs352.trace) for a "stress test" and for your experiment as described below.

We are also providing a perl script on the web page to check the format of your program's output. THE OUTPUT FROM YOUR PROGRAM MUST PAST THIS CHECKER SCRIPT, because we will use a variation of this script to grade your program! Substantial points will be deducted from your grade if your output does not pass the checker script, even if your program is otherwise correct. This policy is necessary to avoid total chaos when grading this complex project.

Experiment:

After you finish and test your cache simulator, use it as a tool to evaluate several cache configurations, to determine which one results in the lowest miss rate for the compress benchmark. The trace file for the compress benchmark is available from the project web page.

Please do not copy this (large) file into your own home directory. Instead, you can access it through the unix filesystem, by using its full path:

/u/billmark/public/cs352/project/cs352.trace

Rather than running all possible combinations of cache parameters, you should develop a plan that tests just a partial set of combinations.

The result of your experiments should be a list of the five best cache configurations for the compress benchmark, with the best one listed first, second best one listed second, etc.

Write a 3-page report summarizing the design of your simulator and your experimental results.

1) Describe your experiences in the development of the program:
   - how did you structure your code and why?
   - what were the challenges in simulating a cache and memory?
   - Describe how your partitioned the 32 bit addresses into tag, index, and offset when given a particular configuration.

2) Describe the experiment and its results:
   - List the five best configurations you found, in rank order.
   - Specify the criteria you used to choose the five best configurations
   - Provide a table of the statistics you collected from each of the five best configurations

3) Discuss factors we did not consider in these experiments:
Discuss other factors that architects must consider before selecting a specific cache configuration, and how these factors would influence the selection. You should consider potential cache access time (i.e. hit time), the die area that the cache might occupy, etc.

Turn in:

You will use both electronic and hardcopy turn in for this assignment.

1) **Electronic:** (note that your last submission to the turnin directory will determine the 'timestamp' of your submission)
    1)a. All of your source files, header files, and makefiles (if used). All of these files must be well documented with comments.
    1)b. If your submission is in C++, then the command "make" should build your executable binary. If your submission is in Java, then "javac cache_sim.java" should build your java binary.
    1)c. The compiled binary that can run on the departmental linux machines. The binary file must be named "cache_sim" and it must accept command line arguments in the exact form specified above.
    1)d. Five output files: The output from simulating test cases 1 through 4, and the stress test (cs352.trace using the baseline configuration)
    1)e. An electronic copy of your report (in either ASCII text-file format, named 'report.txt' or PDF format, named 'report.pdf')
    1)f. You will use the following command to turn in your files electronically:
    csh> /p/bin/turnin --submit impjdi cs352-project <filename>

    To verify that your files have been successfully submitted, use:

    csh> /p/bin/turnin --list impjdi cs352-project

    Only one of the two partners should submit the project. It doesn't matter which partner does it.

2) **Hardcopy:** (turned in to the drop box outside my door, ACES 2.118. But, KEEP A COPY of your assignment for yourself. If you turn this portion late, using late days, then ACES will likely be locked. If that is the case, turn it in to the department drop box and EMAIL THE TA (impjdi@cs.utexas.edu). If you do not, the written portion will not be collected and graded.
    2)a. One page with the statistics for test cases 1,2,3,4 and the stress testcase
    2)b. A printout of your source files, header files, and makefiles (if used). This is just the hardcopy version of what is specified above.
    2)c. A printout of the cache contents and the 1024 words of memory contents for ONLY Test Case 3
    2)d. Your three page report.

Format of Output:

Key to the following description:

base of output: (10 = decimal and 16 refers to hex)
rate should be a real number, with 4 digits of accuracy
0/1: print either 1 for True or 0 for False

STATISTICS:
Misses: <Total (10)> <DataReads (10)> <DataWrites (10) >
Miss Rate: <for total> <for dataread> <for datawrites>
Number of Dirty Blocks Evicted From the Cache: <Total (10)>

CACHE CONTENTS:

Set_Number	Valid	Tag	Dirty	Word0	Word1	…
<base 16>	0/1	<16>	0/1	<16>	<16>	...
.....						

MAIN MEMORY: {print 1024 words of memory starting at address address 0x003f7f00. Print 8 words of memory per line, all values should be base 16}

<address of word0>	<word0>	<word1>	<word2>	....	<word7>
<address of word8>	<word8>	<word9>	<word10>	....	<word15>
…					
<address of word1016>	<word1016>	<word1017>	<word1018>	....	<word1023>