Assignment
Data Cache Simulator

Objectives. Learn how data references are stored and accessed in a data cache.

Your assignment is to write a program that simulates the behavior of a data cache. The program will read a trace of references from standard input and produce statistics about the trace to standard output. The cache simulator should first determine the configuration of the data cache. This is accomplished by reading the configuration information from a *trace.config* file in the current directory. These lines in this file specify the number of cache sets, the number of lines within each set (associativity level), and the size of a cache line in bytes. The format of the configuration information is shown below (note that the value specified after the colon on each line is the portion that can change). You may assume that the number of sets does not exceed 8192 and the associativity level does not exceed 8. You can also assume that the number of sets and line size is a power of two and the line size is greater than or equal to 8 bytes. You may also assume that the format and numbers specified in these three lines are valid.

```
Number of sets: 8
Set size: 1
Line size: 8
```

The trace of references follow the configuration information and has the following format:

<accesstype>:<size>:<hexaddress>

where <accesstype> can be the characters:

R - indicates a read access
W - indicates a write access

<size> is the size of the reference in bytes, and <hexaddress> is the starting byte address of the reference expressed as a hexadecimal number. You may assume that the format and addresses in the trace of references are valid. You should check that the size of each data reference is 1, 2, 4, or 8 bytes. You should also check if each reference's address is properly aligned. If the data reference size or alignment is not correct, then print a warning message to *stderr* and ignore the reference. I have provided an example trace of references below, containing the addresses that correspond to the example references described on page 386 of your text (note the addresses below are hexadecimal byte addresses rather than decimal block addresses shown in the text).

```
R:4:b0
R:4:d0
R:4:b0
R:4:d0
R:4:80
R:4:18
R:4:80
R:4:90
R:4:80
```

The cache configuration information for this example is available in the file *trace.config* and the trace data is available in the file *trace.dat*. You can read the <accesstype>, <size>, and <hexaddress> into a character and two integers, respectively, using formatted I/O with the following type of scanf statement.

```
scanf("%c:%d:%x", &accesstypevariable, &sizevariable, &hexaddressvariable);
```

Your cache simulator should use an LRU replacement algorithm. For data writes, it should employ a write-back and write-allocate policy. The cache simulator output should first print information about the cache configurations used. Next, it should print information for each reference. This information is the reference number, access type (read or write), address, tag, index, offset, result (hit or miss), and the number of memory references (copies of

the cache line to/from memory) caused by the reference (0, 1, or 2). Note that the address and the tag should be printed in hexadecimal. The index and offset should be printed in decimal. This information will help you debug problems with your simulator. Though it is not required, you may also wish to write a function to dump the state of the caches to help debug problems. After the last reference the cache simulator should output the following summary information:

a. the number of hits
b. the number of misses
c. total accesses to the cache
d. hit ratio
e. miss ratio

The output below and the file *trace.stats* contains the output of my cache simulator after pro-cessing only the references in the trace file *trace.dat*.

```
Cache Configuration

   8 1-way set associative entries
   of line size 8 bytes


Results for Each Reference

Ref  Access Address    Tag   Index Offset Result Memrefs
---- ------ -------- ------- ----- ------ ------ -------
   1   read       b0       2     6      0   miss       1
   2   read       d0       3     2      0   miss       1
   3   read       b0       2     6      0    hit       0
   4   read       d0       3     2      0    hit       0
   5   read       80       2     0      0   miss       1
   6   read       18       0     3      0   miss       1
   7   read       80       2     0      0    hit       0
   8   read       90       2     2      0   miss       1
   9   read       80       2     0      0    hit       0

Simulation Summary Statistics
-----------------------------
Total hits       : 4
Total misses     : 5
Total accesses   : 9
Hit ratio        : 0.444444
Miss ratio       : 0.555556
```

You should make your output match mine exactly to facilitate grading of the assignment.

You can invoke an executable I have provided at *~professor/datacache.exe* to check that your output is correct. For instance, you could use the following command on *server* to run my executable if the name of the input file is *trace.dat* and the name of the output file is to be *trace.stats*.

(I have access to this exe to test if the output is correct, however the professor has it read only on the server, so I can not download a local copy. If needed I can test your code and let you know of any bugs during your testing)

```
datacache.exe < trace.dat > trace.stats
```

You should use good programming style, which includes descriptive variable names, abstraction, consistent and readable indentation, and comments. You should comment your program so that others (e.g. the grader) can understand it. You should place comments before each function and each major block of code. You should also have comments at the top of the file indicating your name, this course, the assignment, and the command used to compile your program. For example:

```
/*************************************************************
 *                                                           *
 *       NAME: JOHN DOE                                      *
 *      CLASS: CSD3355                                       *
 * ASSIGNMENT: Data Cache Simulator                          *
 *    Compile: "gcc -g -o datacache datacache.c"             *
 *                                                           *
 *************************************************************/
```

Assignment 6 Suggestions

I recommend that you accomplish assignment 6 in steps. After you implement each step, compare your output with the output generated by my executable. For each step compare your output with the output generated by my executable using the Unix *diff* command to ensure your output exactly matches mine.

(1) Copy *the trace.config* file to your test directory. Read in this configuration file, calculate the number of index and offset bits for the different portions of the memory hierarchy, and print out the configuration information. Change the information in the configuration file and retest to ensure that the correct configuration information is printed.

(2) Take as input the *trace.dat* file. Enhance your program to print out the headings, the reference number, the access type, and the address.

(3) Still take as input the *trace.dat* file. Enhance your program to also print out the tag, index, and offset.

(4) Use the same configuration and create your own trace data sets, where all references are reads. Implement the simulation of the data cache for a direct-mapped organization (set size 1). Print out the result and number of memory references for each reference. Be sure to test for some conflicts to the same line.

(5) Enhance the data cache simulation to deal with associativity levels that are greater than 1, where all references are still reads. Be sure to have more references to a single set than the specified set size in the *trace.config* file.

(6) Implement the simulation of the data cache where some of the references include writes.

(7) Increment counters during the simulation and print out the summary statistics at the end of the simulation.

(8) Test for error conditions, such as an invalid reference size or an invalid reference alignment.