Computer Systems Architecture 2022/23
Laboratory Exercise – Cache controller simulator

1. Introduction

In this laboratory exercise you will write a C program to simulate the operation of a direct mapped cache controller on an embedded system. You will use your simulator to analyse the impact of cache memory size, cache memory block size and cache write policy on the performance of the embedded system when executing a bubble sort algorithm.

You will work individually on this laboratory exercise; submitting a C program, a results file and an individual report.

2. Embedded system memory architecture

The embedded system is interfaced to a 256 Ki x 16-bit external data memory using a 20-bit address bus and a 16-bit data bus. The embedded system contains a direct mapped cache controller for data accesses, and the controller can be configured in one of the 16 modes listed in Table 1.

Mode ID	Cache block size (16-bit words)	Number of cache blocks	Cache size (16-bit words)	Write Policy
1	16	8	128	WAWB
2	16	16	256	WAWB
3	16	32	512	WAWB
4	16	64	1024	WAWB
5	4	64	256	WAWB
6	8	32	256	WAWB
7	32	8	256	WAWB
8	64	4	256	WAWB
9	16	8	128	WAWT
10	16	16	256	WAWT
11	16	32	512	WAWT
12	16	64	1024	WAWT
13	4	64	256	WAWT
14	8	32	256	WAWT
15	32	8	256	WAWT
16	64	4	256	WAWT

Table 1. Cache controller configuration modes
(WAWB – Write Allocate/Write Back   WAWT – Write Allocate/Write Through)

3. Memory trace files

You are provided with a memory trace file, `bubble_sort_trace_`*`nnn`*`.trc`, that contains the addresses of all the read and write memory accesses generated by the CPU of the embedded system when sorting an array of 1500 random 16-bit values using a bubble sort algorithm. The memory trace file only records array accesses, as all other variables can are stored in processor registers.

The document, *Generating Trace Files*, describes how the memory trace files are generated, and provides a simplified example using the bubble sort algorithm.

Each student has their own unique trace file to analyse. You can find the value of the 3-digit ID number *nnn* identifying your unique trace file in the *TRACE FILE ID* column in Blackboard Gradebook.

The trace files can all be found in Dropbox folder https://tinyurl.com/2fzek9nd.

The memory trace files are ASCII text files. Each line begins with an 'R' or 'W' character to signify a memory read or memory write respectively. The 'R' or 'W' character is followed by a single space, and the memory address accessed. The address is represented as a hexadecimal number.

4. Cache controller simulator

Write a structured program in C to simulate the operation of the data cache controller in the embedded system. You may wish to first write a program to simulate the configuration where the cache has 8 blocks, a block size of 16 16-bit words and uses a Write Allocate/Write Back write policy. You can then enhance this program to simulate the other cache controller configuration modes shown in Table 1.

The cache controller simulator must be written as a single source file using ANSI standard C. The program must be suitable to be run from a terminal window without requiring user input and provide its formatted output to the terminal window.

The program **must** not be stored in a software repository, such as GitHub, with shared access.

The program must include a header which includes your name and student ID number. The program must be clearly commented and use descriptive function and variable names. The bubble sort algorithm presented in Appendix A is a good example of how to comment a program.

It is strongly recommended that you construct a software flowchart for the operation of the cache controller, and then directly code this in C.

The simulator must output its results in the comma separated variable (CSV) format described below. A sample set of results are presented in Appendix B.

trace_file_name, *mode_ID*, *NRA*, *NWA*, *NCRH*, *NCRM*, *NCWH*, *NCWM*

trace_file_name	The name of the trace file being analysed (without the folder path)
mode_ID	The ID number of the cache controller configuration mode (1 … 16)
NRA	Total number of read accesses to the external memory
NWA	Total number of write accesses to the external memory
NCRH	Number of cache read hits
NCRM	Number of cache read misses
NCWH	Number of cache write hits
NCWM	Number of cache write misses

The *mode_ID*, *NRA*, *NWA*, *NCRH*, *NCRM*, *NCWH* and *NCWM* values should all be formatted as unsigned integers.

5. Testing your simulator

A small test memory trace file, *test_trace.trc*, has been provided to help you test your trace file read function. As the memory trace files are ASCII text files with a very simple structure, you can create your own trace files to test your cache simulator. The document, *Testing your Cache Controller Simulator*, describes one approach to systematically testing your implementation of the cache controller simulator.

6. Cache controller performance simulations

For each of the configuration modes listed in Table 1, you should simulate the performance of the cache memory controller for the bubble sort algorithm using your allocated memory trace file.

7. Assessment

There are 3 elements to the assessment of the laboratory exercise:

- the cache controller simulation program,
- the results from the cache controller simulation program,
- a short, written report presenting the cache performance results and their analysis.

7.1. Cache simulation program

The cache simulation program must be submitted to Blackboard as a single ASCII file using the submission link that can be found in the *Laboratory Materials* folder. The filename must conform to the format `*familyname_studentIDnumber*_CSA_Simulator.c`

Example: `Green_1234567_CSA_Simulator.c`

The program will be compiled and run to verify that it executes correctly, and generates the submitted results file. The program must not require any command line parameters. The program will be submitted to a source code plagiarism detection tool.

The criteria for the assessment of the program, together with indicative percentage marks, are given below:

• Use of appropriate structured programming techniques	10%
• Readability, including clear comments	10%

7.2. Results from the cache simulation program

The formatted results from the cache simulation program must be submitted to Blackboard as a single ASCII file using the submission link that can be found in the *Laboratory Materials* folder. The filename must conform to the format `*familyname_studentIDnumber*_CSA_Results.csv`

Example: `Green_1234567_CSA_Results.csv`

The results file must be formatted as described in section 4 and will comprise 16 lines. A sample results file is presented in Appendix B.

The criteria for the assessment of the results, together with indicative percentage marks, are given below:

• Numerical correctness	30%

7.3. Cache performance report

You are required to write a short report presenting the performance results from the cache controller simulations together with your analysis of these results. The report must present the performance results in both tabular and graphical formats. There is no requirement to discuss method or background material.

The formatting requirements of the PDF document are described below. The font size requirement applies to all text in the document, including tables and graphs.

• Paper size:	A4
• Margins:	no smaller than 2cm
• Font:	Arial, Calibri or Verdana with the font size no smaller than 11pt
• Line spacing:	no less than 1.0
• Length:	no more than 5 A4 sides

The report must include your name and your student ID number. There is no requirement for a title page. The report must be submitted as a PDF document (via Turnitin) to Blackboard.

The criteria for the assessment of the report, together with indicative percentage marks, are given below:

• Clarity of results presentation (tabular and graphical)	10%
• Analysis of performance results	40%

Analysis

Your analysis should consider a range of cache performance metrics from hit/miss rates through to speed-up factors. For the purpose of calculating execution times, you should assume that the main memory has access time of 100 ns, and the cache memory has an access time of 10 ns.

When analysing / explaining the results you will need to refer to the pattern of memory accesses when executing the bubble sort algorithms on the embedded processor. The source code for the bubble sort algorithm is presented in Appendix A. You can deduce the memory addresses accessed from the trace files.

When presenting your analysis, include all relevant equations and clearly state all assumptions that have been made.

7.4. Submission Deadlines

You have been allocated to one of two laboratory groups. Group A has laboratory sessions scheduled in weeks 5 and 7, and Group B has laboratory sessions scheduled in weeks 6 and 8.

The submission deadlines for the cache simulation program and the results file are as follows. There are no automatic DASS extensions available for these deadlines.

Group A:	17:00 (UK time) on Friday 17th March 2023.
Group B:	17:00 (UK time) on Friday 24th March 2023.

The submission deadline for the report, for Group A and Group B, is 13:00 (UK time) on Monday 17th April 2023.

Peter R Green
Version 1.0 16th February 2023.

Appendix A – Source code for Bubble Sort Algorithm

```
/*
 * Filename:      BubbleSort.c
 * Author:        Jack Andrews
 * Student ID:    123456789
 * Date:          20 February 2019
 *
 */

void BubbleSort(short int *array, int length) {

      // Temporary variable used for swapping of elements
      short int temp;

      // Loop counters
      int i, j;

      /* Pass over the whole array on the first iteration. On subsequent
       * iterations, ignore the already sorted upper elements, achieved by
       * decrementing i on each iteration of the outer for() loop.
       */
      for (i = length-1; i >= 0; i--) {

            /* The inner for() loop iterates over the remaining array elements,
             * comparing each and swapping if necessary.
             */
            for (j = 0; j < i; j++) {

            /* Compare the value at index j in the array with the value at
             * index j+1. If array[j] > array[j+1], then swap the elements.
             */
                  if (array[j] > array[j+1]) {

                        // Swap the array elements
                        temp = array[j];
                        array[j] = array[j+1];
                        array[j+1] = temp;

                  }

            }
```

Appendix B – Sample Simulation Results File

```
bubble_sort_trace_100.trc,1,3577,7725,1011,1989,9032,6832
bubble_sort_trace_100.trc,2,5903,3947,475,7696,5925,7907
bubble_sort_trace_100.trc,3,7960,2985,6748,6853,618,9212
bubble_sort_trace_100.trc,4,6948,3475,5389,1393,8307,3630
bubble_sort_trace_100.trc,5,4438,1615,6699,200,8751,7206
bubble_sort_trace_100.trc,6,6440,506,3513,2216,9936,427
bubble_sort_trace_100.trc,7,1327,4258,7089,1787,8811,2414
bubble_sort_trace_100.trc,8,4597,1,7530,9829,9940,1620
bubble_sort_trace_100.trc,9,5903,3997,495,7696,5925,7907
bubble_sort_trace_100.trc,10,6940,506,3513,2216,9976,427
bubble_sort_trace_100.trc,11,7559,891,3333,8980,5511,2646
bubble_sort_trace_100.trc,12,1510,7884,6129,7691,1753,4692
bubble_sort_trace_100.trc,13,5510,84,9112,7691,1753,7692
bubble_sort_trace_100.trc,14,1510,224,632,7691,8753,492
bubble_sort_trace_100.trc,15,5210,3384,1712,7691,1753,6892
bubble_sort_trace_100.trc,16,5510,784,6012,7691,1753,892
```

Note: This sample results file uses random values for the *NRA*, *NWA*, *NCRH*, *NCRM*, *NCWH* and *NCWM* values