COMPUTER ORGANIZATION (ELE408) LAB 2

The CPU Core and Memory Hierarchy of the TWR-K70F120M

1. Pre-lab Report

ϒ⁄ Read Chapters 3 and 4, 28 of the TWR-K70F120M Reference Manual.

ϒ⁄ Read this lab handout carefully. Take notes and prepare the programs required in this handout.

ϒ⁄ Study the Codewarrior design platform and review dBUG command set.

2. Objectives of this Lab

The purpose of this lab is to learn and gain first hand experiences on the CPU core and the memory hierarchy architecture of the Arm Cortex M4 processor. You will apply your knowledge and basic concepts of computer architecture that you have learnt in the lectures to the lab experiments. In particular, the concept of memory hierarchy and cache design is the main focus of this lab.

Specifically, you will learn and exercise the basics of

2.1. Cache design including mapping, replacement, configuration, and performance impact.

2.2. How to use programmable and high speed SRAM to control the execution time of a program.

2.3. What can be done to reduce power consumption?

3. Basics

The growing speed gap between memories and processing elements makes the memory system design become a crucial challenge to computer designers. High performance computers need sophisticated memory hierarchy to bridge the speed disparity between processors and the main memory. A large register file is usually small that can hardly hold the working set of a program. Register file also requires extra efforts in software to manage it. Highly interleaved memory may provide enough bandwidth, but the memory speed has to be extremely fast and the number of interleaved memory modules has to be excessively large in order to provide enough memory bandwidth. In addition, the interconnection network between processors and memories adds additional delay to each memory access.

High speed on chip cache memories have been successfully used in any processor today. A cache memory bridges the speed gap between the CPU and the off chip memory. Cache memories are generally transparent to programmers and controlled by hardware. For embedded processors, programmable SRAM is often used to selectively hold important program and data that need fast execution.

The Arm Cortex M4 processor has both on chip cache and SRAM for speeding up memory accesses. The cache memory can be configured many different ways by writing appropriate values to cache control register and access control register. The SRAM is a user configurable high speed memory to allow quick program execution. In this lab, you will configure both high speed memories to observe performance differences for different configurations.

4. **Experiment Procedures**

This lab consists of three parts:

1) Observation: read and debug given example program, observe both the code cache and system cache content in debug mode
2) Calculation: given a new pointer step value, calculate the cache line in processor, test and verify whether your calculation is right in debug mode
3) Programming: program a Matrix multiplication program. Test the run time in different cache mode.

4.1. Observation:

Preparation:

◆ Set workspace as "C:\Freescale\ELE408_lab", Import project k70-cache from directory (C:\Freescale\ELE408_Lab\build\cw\cache\k70_cache):

read the source code ‘cache.c’ under ‘Sources/cache’ very carefully. Answer following questions.

- ✧ `What’s the name and address of code cache control register and system cache control registers.`
- ✧ `What’s the default setting in the program?`
- ✧ `Write a flowchart for the function of both CacheWriteOperations and CacheReadOperations. Put your flowchart in your lab report.`

◆ Build the project, run in debug mode:

- ✧ `When “SAME_SET_ADDRESS_OFFSET” is “0X80”. What’s your observation of both code cache and system cache, print on screen your observation of each given cache operation. Explain your observation.`
- ✧ `What if “SAME_SET_ADDRESS_OFFSET” is“0X200” record your observation`

◆ Change the configuration of system cache control register by removing “LMEM_PSCCR_INVW0_MASK” and run in debug mode again

- ✧ `What’s the change of cache content? Screen print your observation and explain why.`

Hint: in debug mode, you’ll find a window named “cache”. Select either code cache or system cache for your observation.

4.2. Calculation:

◆ Calculate the cache location of each instruction when SAME_SET_ADDRESS_OFFSET is 0X1000

- ✧ `Write down your calculation in your lab report.`

◆ Build and debug the program, confirm your calculation

- ✧ `According to your result, find the cache location in code cache and system cache, confirm if your calculation is right, give the screen print in your lab report`

4.3 Programming:

◆ Design a C program to do intensive matrix computations based on the algorithm provided.

Get the run time in different cache mode:

- ✧ `Enable and disable code cache`
- ✧ `Enable and disable system cache`
- ✧ `Enable and disable both code cache and system cache`
- ✧ `Run time in Write-though mode`
- ✧ `Run time in write-back mode`

- ◆ Run your program for each of the 4 configurations and measure the execution times. You may change the program size, data size, or algorithm structure to observe performances. Is there any difference in your measurements? Why?
- ◆ Can you change the data size or program structure to improve the cache performance. Any idea or suggestions are welcome. Please put your thoughts and your results in your lab report.

5. Lab Report Requirements

In your lab report, you should discuss your designs, trade-offs between performance and power, Explanation and interpretation of your results are very important. The lab report will be graded based on your report and discussions. Total mark for the report is 100 points.

- ➢ Prelab report: 20 points
- ➢ Successful experiment: 50 points
- ➢ Results analysis, interpretation, and discussions on your design and engineering constraints: 30 points

In the following items, numbers inside each bracket ¡°[]¡± indicates the point you will earn on a satisfactory report and discussions.

In your discussion and explanations of your results, you should consider the following constraints:

- ➢ What knowledge of mathematics, science and engineering have you applied in this lab and what tools have you used in this lab?[5]

- Economic Constraint (performance/cost ratios) [5]
- Manufacturability, Modularity and Expandability Constraints, Environmental Constraint (power consumption) [5]
- Sustainability: Is your design and implementation sustainable? [5]
- What is the potential impact of your design on real time applications? [5]
- How would you utilize the memory hierarchy available to you to design real time applications? [5]

For each of the above programs hand in the debugged source code with comments; the machine code is not necessary. Be very specific with your comments that explain what you are doing and why you are doing it.

6. Reference C Program for Matrix Multiplication

To better utilize cache memory, each computer provides a set of cache-optimized libraries for common computational tasks.

The following example gives a Blocked Matrix Multiplication Algorithm that attempts to maximize cache hit ratio by increasing the reuse rate of small blocks of data. The block size should be chosen in such a way that it can be completely stored in the cache. An application programmer should select the block size based on the available cache size.

```
1. j := 1;
2. for jj = 1 to $N_2/B_2$ do begin  /* jj-loop */
3.   for kk = 1 to $N/B_1$ do    /* kk-loop */
4.    for i = 1 to $N_1$ do       /* i-loop */
5.     for k = $B_1(kk-1) + 1$ to $kkB_1$ do  /* k-loop */
6.       begin
7.       put X[i,k] in a register;
```

```
8.        for ii = j to j + $B_2$ - 1 do    /* ii-loop */
9.          Z[i,ii] := Z[i,ii] + X[i,k] * Y[k,ii];
10.       end;
11.    j := j + $B_2$;
12. end jj;
```

For simplicity, we assume that B_1 divides N and B_2 divides N2.