National University of Singapore
Dept of Electrical & Computer Engineering (ECE)

CG2028 Computer Organization

Assignment (Semester 1, AY2018/19)
ARM v7-M Assembly Language, Machine Codes and Microarchitecture

I. Background Concepts

A function or subroutine can be programmed in assembly language and called from a C program. A well-written assembly language function usually executes faster than its C counterpart.

It is common in many engineering applications such as image processing and computer vision to do *classification*, which involves determining the *class* a given *pattern* comes from, e.g. determine from a photograph whether the person is male or female. In this case, there are 2 classes: *male* and *female*.

The performance of a classification method, for example a neural network pattern classifier, can be captured in a *confusion matrix* which indicates how many patterns, whose actual class is known, are correctly and wrongly classified.

In general, if we have M classes, the MxM confusion matrix would look like:

$$\mathrm{CM} = \begin{bmatrix} n_{11} & \cdots & n_{1M} \\ \vdots & \ddots & \vdots \\ n_{M1} & \cdots & n_{MM} \end{bmatrix}$$

where the element n_{ij} indicates the number of instances when patterns whose actual class is class i have been classified as class j. Hence, the number of instances when patterns from a particular class m have been correctly classified are indicated by the diagonal element n_{mm}. The confusion matrix is always a square matrix.

One common performance metric of a classification method is the *probability of detection* PD_m for class m, which is given by the formula:

$$\mathrm{PD}_m = \frac{n_{mm}}{\sum_{j=1}^{\mathrm{M}} n_{mj}}$$

II. Objectives

The objective of this assignment is to develop an ARMv7-M assembly language function `pdm(CM,M,m)` to compute the probability of detection PD_m for class m using the information in a given MxM confusion matrix `CM`. (You may assume that $M \leq 10$.)

You will also need to write down the machine code (8-digit hexadecimal, of the form 0xABCD1234) corresponding to each assembly language instruction as a comment next to the instruction in your assembly language function. Note the following.

- You can assume that the instruction memory starts at a location 0x00000000.
- Follow the encoding format given in Lecture 4. The actual ARMv7M encoding format is different, but we will be sticking to the encoding format given in Lecture 4 for simplicity.
- Assume all instructions are 32 bits long, and that the assembler places the instructions in successive word locations in the same order as they appear in assembly language.

- Only the machine codes for the following instructions need to be provided.
    - Data processing instructions such as ADD, SUB, MUL, MLA, AND, ORR, CMP etc., if they are used without shifts (i.e., the *Operand2* is either a register without shift or imm8).
    - Load and Store instructions in offset, PC-relative, pre-indexed and post-indexed modes.
    - Branch instructions - conditional and unconditional, i.e., of the form B{cond} LABEL.
- You do not need to provide machine codes for instructions which do not fall into the categories above, even if you have used them in your program (e.g., BX, MOVW, multiplication instructions with 64-bit products, division, data processing instructions where *Operand2* is a register with shift etc.).

You will also need to provide a paper design showing how the microarchitecture (datapath and control unit) covered in Lecture 4 can be modified to support MUL and MLA instructions. You can assume that a hardware multiplier block is available, which takes in two 32-bit inputs Mult_In_A and Mult_In_B, and provides a 32-bit output, Mult_Out_Product combinationally (i.e., without waiting for a clock edge).

III. Procedure

(a) Initial Configuration of Programs

The C program `main.c` in the project "CG2028AsmtS1AY201819" specifies a `CM` and the corresponding M value. This program calls `pdm(CM,M,m)` repeatedly for different values of m.

Currently in `main.c`, $M = 3$ and the `CM` is initialized with the following values:

$$\mathtt{CM} = \begin{bmatrix} 60 & 2 & 3 \\ 11 & 47 & 7 \\ 27 & 14 & 24 \end{bmatrix}$$

The elements of this matrix are stored row by row in consecutive words in memory in the manner shown in Table 1.

	Address*	Content	Remark
CM →	0x10007FA8	**0x0000003C**	CM element (1,1)
	0x10007FAC	**0x00000002**	CM element (1,2)
	0x10007FB0	**0x00000003**	CM element (1,3)
	0x10007FB4	**0x0000000B**	CM element (2,1)
	0x10007FB8	**0x0000002F**	CM element (2,2)
	0x10007FBC	**0x00000007**	CM element (2,3)
	0x10007FC0	**0x0000001B**	CM element (3,1)
	0x10007FC4	**0x0000000E**	CM element (3,2)
	0x10007FC8	**0x00000018**	CM element (3,3)
	0x10007FCC	**:**	:

Table 1. Contents of consecutive memory locations containing elements of the confusion matrix CM for *M*=3. **Note*: The actual memory addresses in your case may be different from those shown here.

Note: In C programs, the elements of each row and column of a square matrix are indexed from 0 to *M*-1, where *M* is the number of elements in each row and column. In this assignment, you do not need access the elements of the CM matrix in the C program. All necessary operations are to be done in the assembly language function.

(b) Preparations

After completing the Self Familiarisation, you should have the "Lib_CMSISv1p30_LPC17xx" project and several other projects, in the Project Explorer pane of the LPCXpresso IDE (LXIDE).

The "Lib_CMSISv1p30_LPC17xx" project contains the Cortex Microcontroller Software Interface Standard (CMSIS) files.

Import the "CG2028AsmtS1AY201819.zip" archive file which contains the "CG2028Asmt1S1AY201516" project. Within the "src" folder of this project, there are 3 files:

- `cr_startup_lpc17.c`, which is part of the CMSIS and does not need to be modified;
- `pdm.s`, which presently does nothing and returns to the calling C program immediately - this is where you will write the assembly language instructions that implement the `pdm()` function; and
- `main.c`, which is a C program that calls the `pdm()` function with the appropriate parameters, scales down the returned results and prints out the results on the console pane. You do not need to modify this file unless you want to modify the value of *M* and/or the confusion matrix.

Note: Parameter passing between C program and assembly language function:

In general, parameters can be passed between a C program and an assembly language function through the ARM Cortex-M3 registers. In the following example:

```
extern int asm_fn(arg1, arg2, …);
```

arg1 will be passed to the assembly language function **asm_fn** in the R0 register, arg2 will be passed in the R1 register, and so on. The return value of the assembly language function can be passed back to the C program through the R0 register.

(c) Procedure

Compile the "CG2028AsmtS1AY201819" project and execute the program.
Explain the console window output that you see.

What is the address of the memory location containing element (1,1) of the CM matrix?
How do you determine this?

Write the code for the assembly language function `pdm()` after reading the following sub-section (d) carefully. If you have implemented the function correctly, the console output should be:

```
0.923000
0.723000
0.369200
```

Note: You will describe you and your partner's contributions towards the outcomes of this assignment through IVLE. More details will be provided later.

Verify the correctness of the results computed by the function you have written that appears at the console window of the LXIDE.

Show the assembly language program and actual console output to the Graduate Assistant (GA) during the assessment session (in Week 6). See Section IV below.

You will also need to: (i) write the machine code for each instruction as a comment next to the assembly language instruction, and (ii) come up with a microarchitecture capable of supporting MUL and MLA instructions as described in Section II above.

(d) Assembly Language function

You are required to write the assembly language code for the function `pdm()` that computes the probability of detection PD_m for class *m* whose formula is shown in Section I.

The assembly language program only needs to deal with integers. Note that the value returned by the `pdm()` is scaled down by a factor of 10,000 in `main.c` to arrive at the final PD_m value before it is printed out.

Write the ARM v7-M assembly language instructions that first determine the address of the memory location containing the requested data value before loading the data value from that memory location to a designated register.

Note: the assembly language program that you write should work for any value of *M* where $M \leq 10$. During the assessment for Assignment 1, the Graduate Assistant (GA) will modify the `main.c` program and specify a different value of *M* and another CM accordingly.

It is a good practice to push the contents in the affected general purpose registers onto the stack upon entry into the assembly language function, and to pop those values at the end of the assembly language function, just before returning to the calling main program.

For execution efficiency, do not use any subroutine in the assembly language function.

What else can be done to enhance the computation and/or storage efficiency of your programs?

IV. Assessment (Week 6)

For each group of 2 students:

Write a short report of about 4 pages long to answer the questions raised in Section III (c) above and explain how your assembly language program works, as well as the machine codes and microarchitecture design described in Section II above.

Write your names and matriculation numbers, and the name of Graduate Assistant (GA) supervising and assessing your group, clearly on the top left corner of the first page of your report.

Bring a hardcopy of your report when you attend the assessment session and hand it up to the GA assessing your group. Show the execution of your programs and the console output from the programs and explain how your programs work.

***Note*: your assembly language program will be copied onto the lab PC at the start of your assessment session. It will be compiled and tested on the lab PC.**

The assessing GA will modify your program slightly, e.g. specify a different value of *M* and another CM accordingly, to assess whether the correct PD_m values in that case can be obtained.

The GA will verify the machine codes for each instruction (that you wrote as comment) during the assessment.

You should also show and explain the paper design of your microarchitecture incorporating MUL and MLA instructions to the assessing GA.

The assessing GA(s) will ask each student some questions in turn during the assessment. Both students need to be familiar with all aspects of the assignment and your solution as the GA can choose any one of you to explain any part of the assignment. The GA will also ask you additional questions to test your understanding.

To test your understanding of instruction encodings / machine codes, the GA could also ask you to figure out encoding of any instruction whose format is specified in Lecture 4 even if you have not used that particular instruction in your program.

You will also need to fill up a statement of contributions on the IVLE, clearly stating the contributions of each group member towards the assignment. Instructions on how to do this will be sent to you by e-mail in due course.

END