

# NIS2331 《计算机组成与系统结构》

## ARM 汇编程序实验作业 1

### 1. 实验目标

通过 ARMv7 汇编程序实现两个矩阵的乘法操作。

### 2. 实验环境

KEIL  $\mu$ Vision 5.36 或更高版本。具体使用方法见课程课件《ARM 处理器简介与汇编基础》第四节“ARM 汇编程序实验”。

### 3. 实验方案

#### 3.1 文件说明

实验提供两个代码文件 main.c 和 matrix.s

**main.c:** 为 C 语言程序，程序入口，调用 matrix.s 中的 matrix\_mul 函数实现矩阵乘法。

1) 定义两个源矩阵规模及数据，结果矩阵的内存预留；

2) 定义了矩阵结构体 Matrix；

```
typedef struct Matrix {  
    int    row;        //Number of rows  
    int    column;     //Number of columns  
    const int* data;    //Address of matrix data  
}Matrix;
```

这里 data 成员指针指向一个二位数组的地址。

3) 定义了待实现的矩阵乘法函数 matrix\_mul；

// Matrix multiplication function declare

```
extern int matrix_mul(Matrix *results, const Matrix *source1, const Matrix *source2);
```

参数及返回值说明：

- a) 参数 `source1`: 源矩阵 1 结构体;
  - b) 参数 `source2`: 源矩阵 2 结构体;
  - c) 参数 `results`: 结果矩阵结构体;
  - d) 返回值: 0 表示执行成功; 1 表示执行失败。
- 4) 实现了一个打印矩阵的函数 `print_matrix`, 用以打印乘法结果矩阵;
- 5) `main` 函数
- a) 定义了两个源矩阵结构体 `source_matrix1` 和 `source_matrix2`, 使用 1) 中定义的规模和数据;
  - b) 定义了结果矩阵结构体 `results_matrix`, 使用 1) 中定义的规模和预分配空间;
  - c) 将结果矩阵数据部分内存空间清 0;
  - d) 调用 `matrix_mul` 对两个源矩阵进行乘法操作, 结果存放在结果矩阵中。对于成功的调用, 将打印结果矩阵; 对于失败的调用, 显示乘法失败。

**matrix.s:** 实现一个矩阵乘法函数 `matrix_mul`;

- 1) 定义了代码区域(AREA), 对外 Export 了 `matrix_mul` 符号。
  - 2) 提供了函数返回。
- 矩阵乘法功能待实验开发。

### 3.2 实验流程

- 1) 按照“ARM 汇编程序实验”的说明, 创建一个 C 语言与汇编混合编程的项目。将实验提供的 `main.c` 和 `matrix.s` 加入到项目中。
- 2) 在 `matrix.s` 的代码区域实现矩阵乘法函数 `matrix_mul`;
- 3) 编译成功后进行调试;
- 4) 运行到 `main` 函数的 `return` 语句时, 如果 `matrix_mul` 实现正确, 应能在 Debug (printf) Viewer 窗口中看到正确的结果矩阵输出;
- 5) 更改 `main.c` 中的源矩阵规模和数据, 测试不同规模的矩阵;
- 6) 提交 `matrix.s`。(不要提交 `main.c`)

### 3.3 实验要点

- 1) `matrix_mul` 的输入参数为结构体 `Matrix` 的指针。`Matrix` 结构体的大小为 12 字节，成员变量 `data` 大小为 4 个字节，保存真正的矩阵数据的地址。  
以参数 `source1` 为例，进入 `matrix_mul` 汇编代码后：
  - a) 寄存器 `R1` 保存了结构体 `source1` 的指针；
  - b) `R1 + 8` 的位置是 `data` 成员变量，保存的是源矩阵 1 的数据 `SOURCE_MATRIX1_DATA` 所在的地址；
  - c) 即通过 `R1` 访问矩阵元素需要两次内存访问。
- 3) 二维数组在内存中以 Row-Major 形式连续存放，所谓 Row-Major 指的是从第一行开始，一行接一行进行存放，见下图：

row,col	0,0	0,1	0,2
	1,0	1,1	1,2
	2,0	2,1	2,2

			0,0	0,1	0,2	1,0	1,1	1,2	2,0	2,1	2,2			
--	--	--	-----	-----	-----	-----	-----	-----	-----	-----	-----	--	--	--

所以可以通过 `const int* data` 访问整个二维数组，对于以 `data[2][3]` 为例，其地址就是 `data + (2 * columns) + 3`；

- 4) 在汇编程序中，地址的加减以字节为单位。由于每个 `int` 是 4 字节，所以假如寄存器 `R4` 存储的是数组首地址的话，那么 `[2][3]` 元素的地址应该是 `R4 + ((2 * columns) + 3) * 4`；
- 5) 输入参数使用了 `R0`, `R1` 和 `R2` 三个寄存器，`R11` 作为 `Frame Pointer` 不建议使用，故可以使用的寄存器范围是 `R3-R10`。如果不够，应考虑使用栈存储局部变量，参考《ARM 处理器简介与汇编基础》75 页起的“本地变量”部分。
- 6) ARM 指令集中还有很多课堂未讲授的指令，可以查阅 canvas 上“文件”→“ARM Programmer”下的 `ARM® Cortex™-A Series Version 4.0 Programmer's Guide.pdf` 文件，其中“Chapter 5”有各类指令的介绍，“Appendix A”有所有指令集的 Reference。丰富的指令集有利于更好地

完成实验。

- 7) 可以考虑写一个 C 语言的 `matrix_mul` 函数，然后对照实现 ARM 汇编的 `matrix_mul`，会容易一些。
- 8) 矩阵不一定是方阵，行和列可以不同。
- 9) 矩阵乘法有可能失败，从而返回 1。比如 `source1` 的列数不等于 `source2` 的行数，就无法进行乘法操作。