

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 μ 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) 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。
- 4) 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。
- 5) 矩阵不一定是方阵，行和列可以不同。
- 6) 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。

3.4 参考文献

Canvas 中，“文件” → “ARM Programming” 目录下：

ARM® Cortex™-A Series Version 4.0 Programmer's Guide.pdf 中给出了 ARMv7 Cortex-A 的开发人员指引，包含体系结构、寄存器、指令集等内容的说明，可以参考该文献，使用更为丰富的指令集完成本实验。

4. 实验评估

- 1) 评估环境：

简单来说，我们通过 Docker 运行一个 ubuntu20.04 的环境，为了能在上面跨架构地编译和运行 Arm 架构的程序，我们安装 GCC-ARM 工具链进行交叉编译，安装 `qemu-arm-static` 来执行 ARM 程序。

2) 文件清单:

- a) Dockerfile: 用于构建评估所用 docker 环境
- b) arm2gas.pl: 用于将 ARMASM 汇编语法转换为 GNU 的 GAS 语法
- c) main.c: 主体 C 文件, 包括读取矩阵数据和调用 matrix_mul 函数
- d) matrix.s: 实验实现的汇编程序 (ARMASM 语法)
- e) grade.py: 评分程序。这里提供了三类 case 供调试

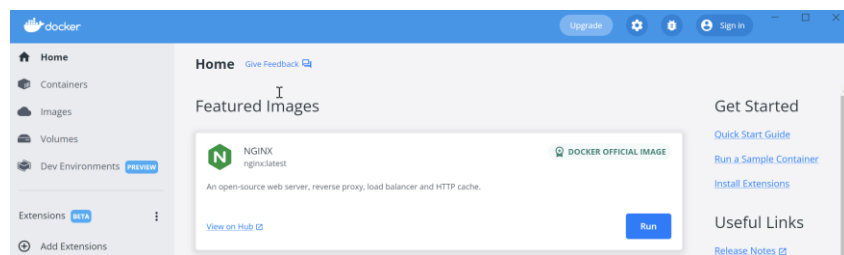
3) 环境准备:

a) 安装 Docker

➤ Window 下 Docker 的安装

参考 [Docker 官方文档](#)

- i. 在上面给出的链接下载 Docker Desktop for Windows
- ii. 下载完成后运行 Docker Desktop Installer.exe, 直接使用默认选项安装, 安装完后重启即可使用



- iii. Docker 命令的使用可以采用 Command Line 或 Powershell 或 WSL2。这里给出 Powershell 的使用截图, 表示正确运行:

```
PS C:\> docker ps
```

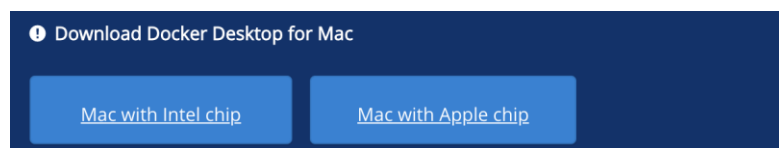
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

➤ Ubuntu 下 Docker 的安装

参考 [Docker 官方文档](#), 文档非常详细, 一步步操作即可, 这里不进行赘述。

➤ macOS 下 Docker 的安装

参考 Docker 官方文档, 下载 Docker Desktop for Mac 即可。



b) Docker 容器内的配置

主要是安装了 GCC-ARM 的工具链、qemu 以及 python3.8。

安装使用的命令均写在 Dockerfile，无需在容器内手动输入。

4) 使用方式：

- a) 在 Dockerfile 目录下构建 docker: ``docker build -t ubuntu:arm .``

```
PS C:\armasm_lab> ls

Directory: C:\armasm_lab

Mode                LastWriteTime         Length Name
----                -
d-----          5/6/2022   9:35 PM             .git
-a-----          5/5/2022   7:44 PM          14622 arm2gas.pl
-a-----          5/5/2022   7:44 PM           457 Dockerfile
-a-----          5/5/2022   7:44 PM          3470 grade.py
-a-----          5/5/2022   7:44 PM          2110 main.c
-a-----          5/5/2022   7:44 PM           208 Makefile
-a-----          5/5/2022   7:44 PM          4204 matrix.s
-a-----          5/5/2022   8:12 PM          2140 README.md

PS C:\armasm_lab> docker build -t ubuntu:arm .
[+] Building 101.0s (10/10) FINISHED
```

使用 ``docker images`` 命令能够看到我们刚刚构建的映像：

```
PS C:\armasm_lab> docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        arm       733287173333  2 minutes ago  926MB
```

- b) 运行容器: ``docker run -it ubuntu:arm /bin/bash``

- c) 编译程序: ``make``

- d) 测试: ``python3 grade.py``

运行后会生成三组测试用例 `case_1.txt`、`case_2.txt`、`case_3.txt`，这三组测试用例仅供参考，评分时会采用其他测试用例。测试用例规范如下：

- i. 第一行两个数字分别表示矩阵 1 的行数（`r1`）和列数（`c1`）；
- ii. 接下来跟着的 `r1` 行数据（每行 `c1` 个数字）是矩阵 1 的元素；
- iii. 下一行的两个数字分别表示矩阵 1 的行数（`r2`）和列数（`c2`）；
- iv. 紧跟着的 `r2` 行数据（每行 `c2` 个数字）是矩阵 2 的元素；

windows 示例如下：

在 docker 中运行 `make`，成功后可以得到 `matrix` 文件，若实现正确，在运行 `grade.py` 后顺利通过测试用例。也可以单独运行 `matrix` 可执行文件，命令行传入输入矩阵的文件，可以得到计算结果的输出。

```

PS C:\armasm_lab> docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        arm       733287173333   2 minutes ago  926MB
PS C:\armasm_lab> docker run -it ubuntu:arm /bin/bash
root@5d5752a7ef85:/armasm_lab# ls
Dockerfile  Makefile  README.md  arm2gas.pl  grade.py  main.c  matrix.s
root@5d5752a7ef85:/armasm_lab# make
./arm2gas.pl -i -c matrix.s
WARN: matrix.s.out:50 -> matrix.s.out:50: Implicit shift is not supported in GAS, converting to explicit shift
WARN: matrix.s.out:62 -> matrix.s.out:63: Implicit shift is not supported in GAS, converting to explicit shift
WARN: matrix.s.out:88 -> matrix.s.out:90: Implicit shift is not supported in GAS, converting to explicit shift
mv matrix.s.out matrix_gnu.s
arm-linux-gnueabi-gcc matrix_gnu.s main.c -static -g -o matrix
main.c: In function 'main':
main.c:55:9: warning: initialization of 'int *' from incompatible pointer type 'int (*)(size_t)(rows))[(size_t)(columns))' [-Wincompatible-pointer-types]
    55 |         &data_1
       |         ^
main.c:55:9: note: (near initialization for 'source_matrix1.data')
main.c:65:9: warning: initialization of 'int *' from incompatible pointer type 'int (*)(size_t)(rows))[(size_t)(columns))' [-Wincompatible-pointer-types]
    65 |         &data_2
       |         ^
main.c:65:9: note: (near initialization for 'source_matrix2.data')
main.c:75:9: warning: initialization of 'int *' from incompatible pointer type 'int (*)(size_t)(source_matrix1.row))[(size_t)(source_matrix2.column))' [-Wincompatible-pointer-types]
    75 |         &results_matrix_data
       |         ^
main.c:75:9: note: (near initialization for 'results_matrix.data')
root@5d5752a7ef85:/armasm_lab# python3 grade.py
[*] Test 1 succeed.
[*] Test 2 succeed.
[*] Test 2 succeed.
[*] Final score is 30.

root@5d5752a7ef85:/armasm_lab# ./matrix case.1.txt
204 204 204 204 204 204 204 204
240 240 240 240 240 240 240 240
276 276 276 276 276 276 276 276
312 312 312 312 312 312 312 312
348 348 348 348 348 348 348 348
384 384 384 384 384 384 384 384
420 420 420 420 420 420 420 420
456 456 456 456 456 456 456 456

```

同样地，在 ubuntu 上也是一样，在此不重复展示 docker 命令的使用：

```

root@f8285059a7c1:/armasm_lab# python3 grade.py
[*] Test 1 succeed.
[*] Test 2 succeed.
[*] Test 2 succeed.
[*] Final score is 30.

root@f8285059a7c1:/armasm_lab# ls
Dockerfile  Makefile  arm2gas.pl  case.1.txt  case.2.txt  case.3.txt  grade.py  main.c  matrix  matrix.s  matrix_gnu.s
root@f8285059a7c1:/armasm_lab# ./matrix case.1.txt
204 204 204 204 204 204 204 204
240 240 240 240 240 240 240 240
276 276 276 276 276 276 276 276
312 312 312 312 312 312 312 312
348 348 348 348 348 348 348 348
384 384 384 384 384 384 384 384
420 420 420 420 420 420 420 420
456 456 456 456 456 456 456 456

```

5) 评分标准：

- a) 程序正确性 (70%)：能够顺利通过编译并通过测试用例。测试环境提供的三组测试用例仅供参考，实际评估时会采用其他的测试用例。
- b) 程序可读性 (30%)：规范编写，并能适当注释提高可读性，程序简洁。