California State University, Northridge

ECE 422L - Computer Architecture

Fall 2022

Lab 6

Matrix Multiplication with ARM Assembly

March 22, 2023

Instructor: Dr. Shahnam Mirzaei

Mark Attia

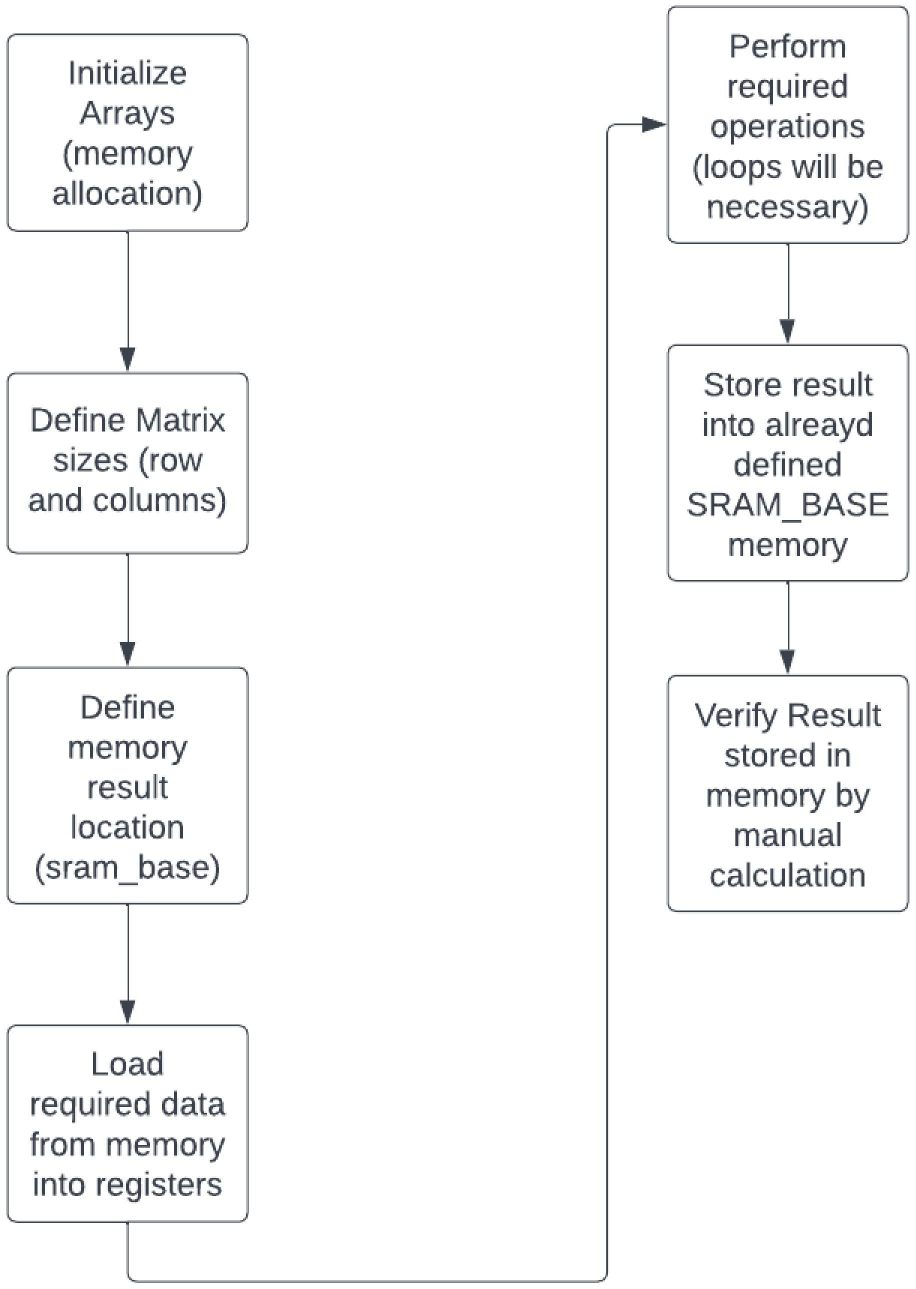
<u>Table of Contents</u>

Introduction

The purpose of this experiment is to learn how to manipulate matrices with our microcontroller. We will initialize them in memory and use previously learned ldr and str instructions to be able to perform different operations on these matrices. We will add, subtract, and multiply the matrices as they are fundamental data structures that we must be familiar with.

Procedure

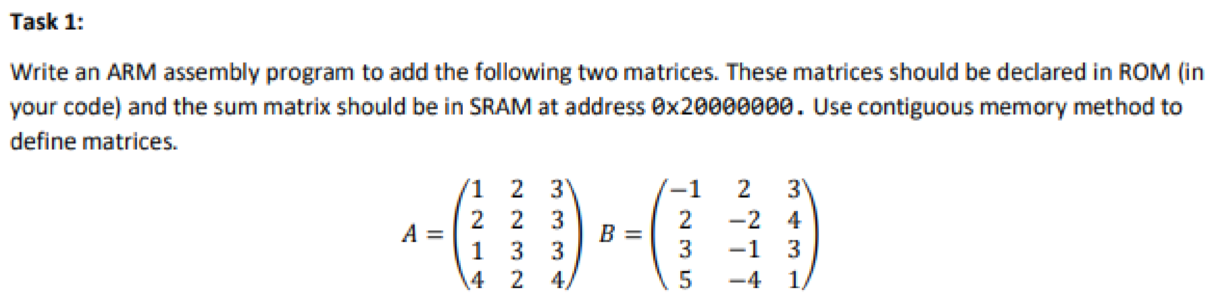
1. Create Project Lab5
2. Ensure all details are correct for compiling code
3. Simulation debugger
4. Drivers for board installed -Using Tiva Core
5. Use sample program provided in manual and test for proper execution
6. Carefully monitor memory activity in step-by-step debug mode
7. Complete task 1.
8. Complete task 2
9. Complete task 3


Initialize Arrays (memory allocation)
Define Matrix sizes (row and columns)
Define memory result location (sram_base)
Load required data from memory into registers
Perform required operations (loops will be necessary)
Store result into alreayd defined SRAM_BASE memory
Verify Result stored in memory by manual calculation

Testing Strategy

The testing strategy for this experiment comes from being able to understand what should be happening inside of our memory based on what we have learned about using ldr and str to pull from and send to memory. Keil has a useful feature of memory view, where we can simply enter any address we'd like (usually SRAM_BASE) and view the values at that location. This feature will be our primary method to verify that our matrix operations are correct.

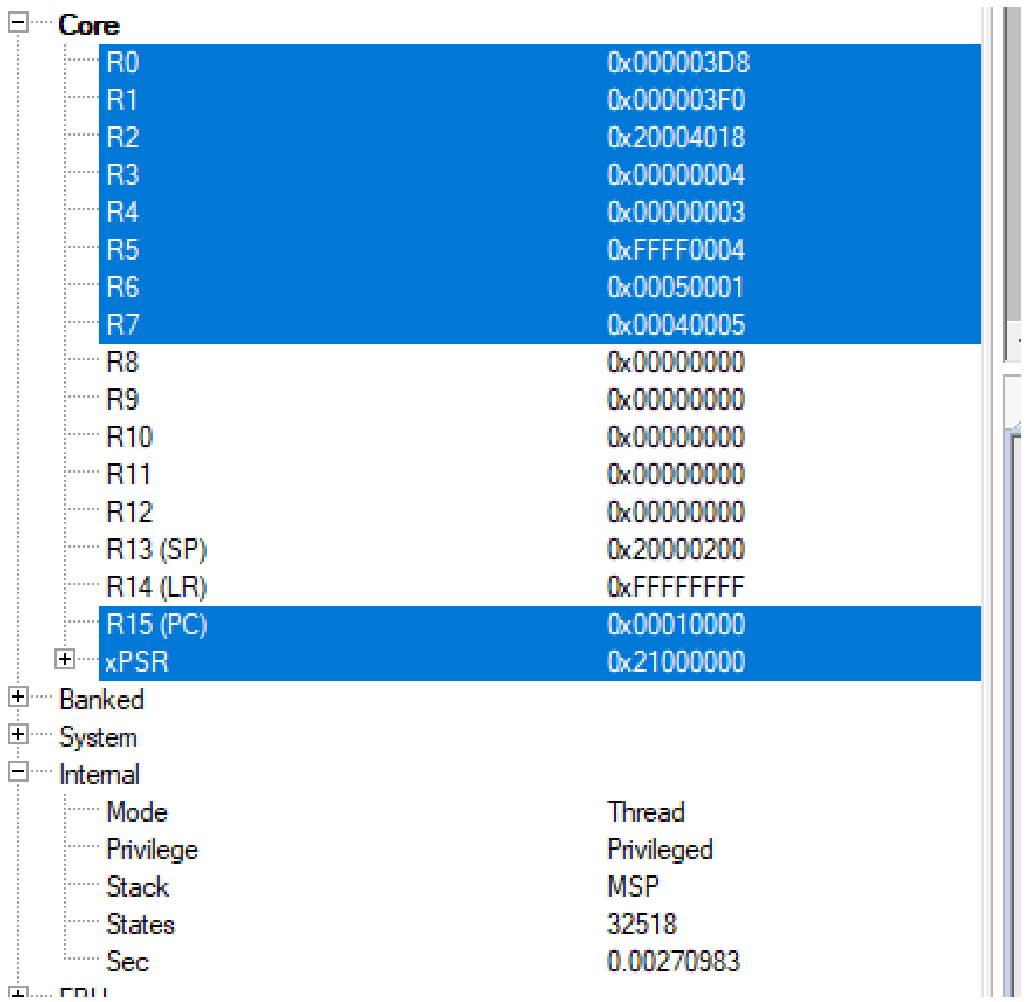
Results



Task 1:

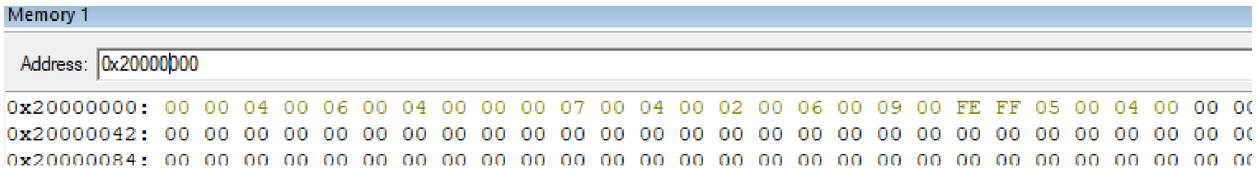
Write an ARM assembly program to add the following two matrices. These matrices should be declared in ROM (in your code) and the sum matrix should be in SRAM at address `0x20000000`. Use contiguous memory method to define matrices.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & 3 \\ 1 & 3 & 3 \\ 4 & 2 & 4 \end{pmatrix} \quad B = \begin{pmatrix} -1 & 2 & 3 \\ 2 & -2 & 4 \\ 3 & -1 & 3 \\ 5 & -4 & 1 \end{pmatrix}$$



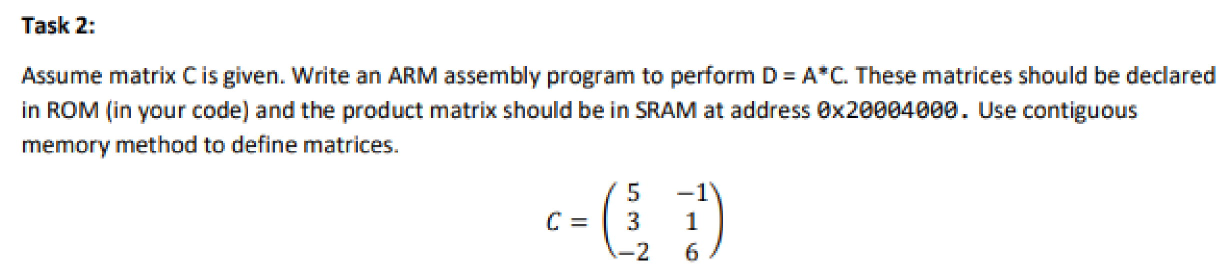
Core	
R0	0x000003D8
R1	0x000003F0
R2	0x20004018
R3	0x00000004
R4	0x00000003
R5	0xFFFF0004
R6	0x00050001
R7	0x00040005
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000200
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00010000
xPSR	0x21000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	32518
Sec	0.00270983

Figure 1: Registers Task 1



```
Memory 1
Address: 0x20000000
0x20000000: 00 00 04 00 06 00 04 00 00 00 07 00 04 00 02 00 06 00 09 00 FE FF 05 00 04 00 00
0x20000042: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000084: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

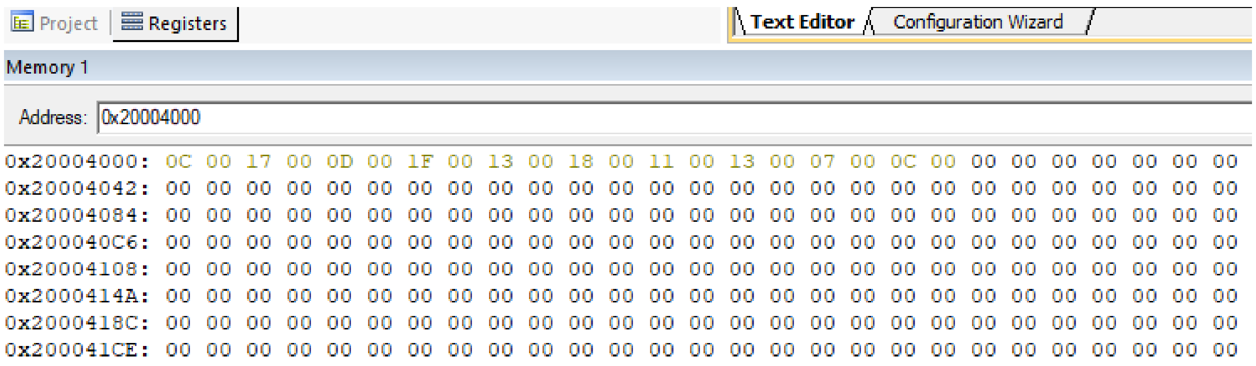
Figure 2: Memory Result SRAM_ BASE



Task 2:

Assume matrix C is given. Write an ARM assembly program to perform D = A*C. These matrices should be declared in ROM (in your code) and the product matrix should be in SRAM at address `0x20004000`. Use contiguous memory method to define matrices.

$$C = \begin{pmatrix} 5 & -1 \\ 3 & 1 \\ -2 & 6 \end{pmatrix}$$



```
Memory 1
Address: 0x20004000
0x20004000: 0C 00 17 00 0D 00 1F 00 13 00 18 00 11 00 13 00 07 00 0C 00 00 00 00 00 00 00 00
0x20004042: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20004084: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200040C6: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20004108: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000414A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000418C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200041CE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 2: Matrix Multiplication Result

Logic stays identical except for adding a delay to debounce button presses.

https://photos.app.goo.gl/fs1UeRV7MT87dotH9

Questions

There are benefits and drawbacks for either method of performing these operations. If we use multiple blocks, it allows the CPU to access multiple sets of data at once (in parallel) and therefore increases speed and most likely reduces conflicts as everything has its own dedicated space. However it could still end up being slower as the CPU will have to access the memory more times to reach all of the data that it needs which could be bad if the CPU needs more than one memory location for a single operation. Overall, it would mostly depend on the use case for which one was better, so my preference would usually be to use the one that is easier for my specific application.

Conclusion

This experiment allowed us to get refreshed with our ldr and str instructions, as well as begin manipulating larger datasets. It helped us bring together all the skills we’ve been working on in ARM assembly programming and add to it using very import data type of matrices. Doing this will act as a steppingstone to future experiments where we must manipulate even larger sets of data using similar concepts such as loops, ldr, str, and DCW/DCD (memory allocation).

https://github.com/EgyptiansFTW/Computer-Architecure-Lab.git