**CMPE-250 Laboratory Exercise Four**

**Iteration and Subroutines**

By submitting this report, I attest that its contents are wholly my individual writing about this exercise and that they reflect the submitted code. I further acknowledge that permitted collaboration for this exercise consists only of discussions of concepts with course staff and fellow students; however, other than code provided by the instructor for this exercise, all code was developed by me.



Dean Trivisani

Performed September 19, 2017

Submitted September 26, 2017

Lab Section 01L2

Instructor: Melton

TA: Donald Mannise

Michael Baumgarten

Matthew Toro

Lecture Section 01

Professor: Melton

**Abstract**

This exercise utilized the Keil MDK-ARM Microcontroller Development Kit to write a Cortex-M0+ assembly language program that uses a subroutine and iteration to perform division for the purpose of demonstrating how complex operations can be implemented in assembly, as well as increasing familiarity with condition flags and conditional branching. A project was created in Keil MDK-ARM’s µVision IDE and a library file was added to the project. A division subroutine was written that would output the quotient and remainder for the division of two unsigned integers without changing the input parameters. The subroutines contained in the library file were used to test the functionality of the division subroutine. The program was loaded into a debugger simulation and the final register values examined. The simulation results showed that the program functioned correctly and performed all of the library test operations successfully.

**Procedure**

A project directory and project files were created in µVision using the NXP MKL46Z256VLL4 microcontroller on the NXP FRDM-KL46Z Freedom development platform, and a library file (Exercise04\_Lib.lib) containing subroutines for initializing, loading, and testing the subroutine for numerous different values was added to the project’s source group. MAX\_DATA was EQUated to 25 and the three aforementioned library subroutines (InitData, LoadData, and TestData, respectively) were imported in the MyCode AREA. Word variables P and Q were allocated word array with twice the value of MAX\_DATA in the MyData AREA. These variables and the array were then exported in the MyData AREA. The main program code was then written, using the library subroutines (in the order given) to first initialize and load the values for variables P and Q. If the Carry flag of the Application Program Status Register was set after these steps, the program was terminated. If the Carry flag was clear, the division subroutine DIVU was called and performed a division operation for dividend P and divisor Q. If the result of the operation was valid and the Carry flag was clear, the quotient and remainder were stored in P and Q, respectively. Otherwise, P and Q were stored with the value 0xFFFFFFFF. The TestData library subroutine was then called, which performed various division operations and check that the results were correct, storing the results in the Results word array. For any incorrect results, register R6 was incremented by 1. The program then returned to the beginning of the main program code and the process was repeated. The program was then loaded into a debugger simulation and the final register values for the simulation were observed. The final value of register R6 was 0, thus the program operated as intended and all results were correct.

**Results**

The program executed as intended, and the data test showed that all operations were run and no incorrect results were found. The final register values are shown in Figure 1. R0 and R1 were the registers that contained first the values of the divisor and dividend, then the quotient and remainder for each operation. R3 and R4 were temporary values used in the calculations. R6 contained the number of incorrect results given by the program, and R7 held the total number of operations tested. The xPSR values are shown as well.

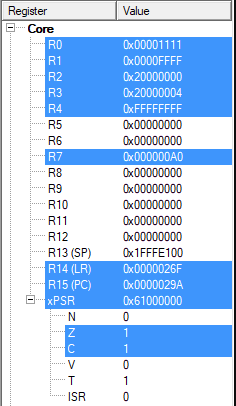


Figure 1: Final Register Contents of the Debugger Simulation

The memory ranges for MyCode, the library file code, and the variables and stack RAM are shown in Table 1. These values were found by examining the Memory Map for the simulation and recording the starting address for each area, then adding the size value less one (because the size of the area includes the starting byte) to the starting address to find the address of the final byte.

Table 1: Memory Ranges for Code Areas/RAM and Sizes

|  |  |  |  |
| --- | --- | --- | --- |
| Area | Start Address | Last Byte Address | Size |
| Exercise04\_Lib | 0x000000C0 | 0x00000263 | 0x000001A4 |
| MyCode | 0x00000264 | 0x000002F7 | 0x00000094 |
| Variable RAM | 0x20000000 | 0x20000039 | 0x0000003A |
| Stack RAM | 0x1FFFE000 | 0x1FFFE0FF | 0x00000100 |

**Conclusion**

This exercise gave insight into the use of subroutines and iteration for implementing complex operations, as well as improved familiarity with condition flags and by extension conditional branching. These are essential to computer engineering, as subroutines, iteration, and conditionals are necessary to create the systems that are integral to the industry.