**CMPE-250 Assembly and Embedded Programming**

**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. Other than code provided by the instructor for this exercise, all code was developed by me.

Shubhang Mehrotra

Performed 18th February 2021.

Submitted 25th February 2021.

Lab Section 1

Instructor: Muhammad Shaaban

|  |  |  |
| --- | --- | --- |
| TA: |  | Anthony Bacchetta  Aidan Trabuco |
|  |  | Sam Myers |
|  |  | Payton Burak |

Lecture Section 1

Lecture Instructor: Dr. Roy Melton

**Abstract**

The purpose of this exercise was to explore conditional loops through subroutines in a Cortex-M0+ assembly language program. The scope of the activity was to write assembly code to divide a number to obtain a quotient and a remainder in separate registers. The written code was then tested in a provided test bench against multiple test cases. Secondary learning goals of the activity were to understand memory mapping of assembly code instructions and variable usage.

**Procedure**

A subroutine was written to perform the division using two inputs and provide two outputs for the activity. The two inputs were– the dividend in register R1 and the divisor in the register R0. The quotient was then stored in register R0, followed by the remainder in register R1.

Moreover, the subroutine was written in such a manner than it can handle the edge cases of both division of zero, and division by zero. When division by zero was called, the subroutine left the inputs unchanged and set the C flag on return, which was clear on all other cases. In the case of division of zero, both outputs were set to zero, and hence the Z flag was also set to zero and C flag was clear. It was ensured, that none of the other registers or flags were changed by the code, during the process.

The activity also introduced the concept of importing from library files and exporting data back to them. The provided library file contained the test cases against which the subroutine was tested. To handle communication with the test bench, variables were exported to the library file and “InitData”, “LoadData”, and “TestData” subroutines were imported and called. The code written was then built and debugged in Keil IDE environment to obtain the results.

**Results**

The resources included in the activity comprised of a library file which had the required test cases for the code written in the exercise. The test was run, and the register values shown in Figure 1 were obtained.

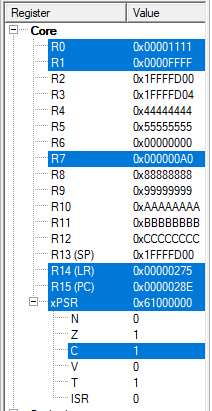


Figure 1. Register content.

While register R0 and R1 depict the quotient and the remainder of a test case; registers R6 and R7 pertain to the test data and give information about the number of failed test cases and the total number of test cases, respectively. Looking at the values of registers in Figure 1, it can be concluded that 240 tests were done, none of which failed.

Moreover, the figure also depicts that none of the other registers or PSR flags were affected in the process. This verifies that the code written was successful in its implementation.

The translation of the code generates a listing file for the assembly code, and building it generates the map file. The contents of the two files were analyzed to obtain the memory addresses of various objects in the code. The data obtained is listed in Table 1.

Table 1. Memory ranges

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Memory Address  Start | Memory Address  End | Size |
| Main Program | 0x00000004 | 0x0000002A | 0x00000026 |
| Library Code | 0x000000C0 | 0x00000264 | 0x000001A4 |
| Variables | 0x1FFFFD00 | 0x1FFFFD3A | 0x0000003A |
| Stack | 0x1FFFFC00 | 0x1FFFFD00 | 0x00000100 |

From Table 1, the instructions written for the main program began at memory address 0x00000004 and took 38 bits in the memory, whereas the provided library took 420 bits starting from address 0x000000C0.

The variables and the stack occupied memory in the RAM. While variables occupied addresses 0x1FFFFD00 to 0x1FFFFD3A, the stack used to maintain the variable registers took a total memory of 256 bits in the RAM in the addresses 0x1FFFFC00 to 0x1FFFFD00.

**Conclusion**

The activity was successful in performing unsigned integer division operation using a subroutine and conditional looping. The given test cases were successfully executed, and the results obtained depicted that the code passed all general and edge cases. The memory map provided an insight about how the instructions are stored in Read-only memory, and variables occupy the RAM.