**CMPE-250 Assembly and Embedded Programming**

**Laboratory Exercise Two**

**Basic Arithmetic Operations**

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 4th February 2021.

Submitted 11th February 2021.

Lab Section 1

Instructor: Muhammad Shaaban

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

Lecture Section 1

Lecture Instructor: Dr. Roy Melton

**Abstract**

The purpose of this exercise was to explore basic Arithmetic operations in an ARM Cortex-M0+ assembly language program using the Keil MDK-ARM IDE. The activity explores the movement and manipulation of data in the registers to perform the basic Cortex-M0+ arithmetic instructions.

**Procedure**

To perform arithmetic operations in Assembly Language all operands must be stored in Registers. This allows the system to have quick access to data whenever required by an instruction. This activity aims to solve equation 1:

*(45* + *6* + *([*−*13]* ÷ *4)* − *(7*× *3)* − *65* + *33) …(1)*

To solve equation 1, the operations required are: addition, subtraction, integer division, and multiplication. While Bits can be added and subtracted directly in assembly using their respective mnemonics, multiplication and integer division are not calculated directly. Multiplication is achieved by a Logical Left Shift operation, while integer division for a Signed number is only obtained by an Arithmetic Right Shift operation, and unsigned bit can be divided using a Logical Right Shift as well.

The code written to perform the activity solved the given equation from left to right order, while keeping the order of operations consistent. To initiate, value “45” was moved to a register using the “MOVS” function. The rest of the operations were then carried out by adding the required immediate value to the next register and then adding or subtracting its value to the first register. For instance, the first operation– (*45+6)* was carried out by using the “ADDS” function with an immediate value of *6*.

For multiplication and division, the value of the Shift Amounts for each shift was stored in memory using an EQUate statement. This was done as a good practice to ensure easy accessibility for reuse. Then using the Label for the division statement with an “ASRS” function, integer division was performed on *-13*, and stored in a separate register. The value of this register was then added to the original register. Similarly, multiplication of *7 by 3* was obtained, by first using the Label for multiplication statement with an “LSLS” function, and then adding another *7* to the value, finally adding it to the original register.

The process was continued, obtaining individual results in separate registers, and adding them to the original register to obtain the final answer.

**Results**

The Code written in the activity was executed and Debugged using the Build Tool and the Debugger Tool. Figure 1 shows the result of the final run. Register R0 shows the value 0xFFFFFFFA, which corresponds to *-6* in decimal.

Graphical user interface, text, application, email

Description automatically generated

Figure 1. Result

All the lines were executed as expected and the result obtained was in line with the calculations:

* *(45* + *6* + *([*−*13]* ÷ *4)* − *(7*× *3)* − *65* + *33)*
* *(51 + ([*−*13]* ÷ *4)* − *(7*× *3)* − *65* + *33)*
* *(51 + (*−*4)* − *(7*× *3)* − *65* + *33)*
* *(47* − *(7*× *3)* − *65* + *33)*
* *(47*− *21* − *65* + *33)*
* *(26* − *65* + *33)*
* *(*−*39* + *33)*
* −*6*

**Conclusion**

The activity was successful in performing basic arithmetic operations using Assembly Language. The activity also demonstrated good use of Registers and Memory usage in Assembly language to store and retrieve data. Storing data in registers allows quick and repetitive access to the data while ensuring legal bit width is maintained in the process. Memory usage to store frequently used data helps not only reduce instructions, but also the labels help to increase the overall understandability of the code. Overall, the practice successfully helped in increasing the understanding of how mathematical operations are carried out by an ARM Cortex-M0+ microprocessor.