**CMPE-250 Laboratory Exercise 02**

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

John Judge

Performed 2/4/16

Submitted 2/11/16

Lab Section 02

Instructor: Dr.Shaaban

TAs: Peter Muller

Stephen Moore

Connor Goss

Lecture Section 01

Professor: Allesandro Sarra

**Abstract**

In this exercise, a given equation was solved using the Cortex-M0+ assembly language. The equation was solved while respecting certain limitations that were also given. The objective of the exercise was familiarization of basic arithmetic using the Cortex-M0+. The exercise included; movement of values into registers, use of constants, and Cortex-M0+ arithmetic instructions.

**Design Methodology**

In this exercise a given equation was solved using the ARM Cortex-M0+ assembly language program. This taught basic arithmetic using the movement of data into registers, constants, and basic Cortex-M0+ instructions. The equation was given in the decimal (base 10) number system but first had to be solved by converting the values into hexadecimal (base 16) number system.

As seen in Equation (1.1) the given equation used the arithmetic operations of division, multiplication, addition, and subtraction. In order to correctly verify the results later on, the equation had to be rewritten using the hexadecimal system and solved with regards to the order of operations.

Once the equation was rewritten using the hexadecimal format, as seen in Equation (1.2), then the equation cold be solved. What Equation (1.2) does not show is the signed values to represent the negative numbers such as the “-5” at the beginning of the equation. In hexadecimal, negative values are represented by first taking the absolute value of the number. Each bit is then subtracted from the value 1510 (F16), except the most insignificant bit, which is subtracted from 1610. (It is important to note that in Cortex-M0+ all values stored in registers are represented in 32-bits. As a result, in this exercise, all values whether signed or unsigned were represented in 32-bits.) The result is referred to as a signed value and represents a negative number

When using the hexadecimal system the “order of operations” is the same as the decimal system. In this example, however, order of operations was not completely observed for simplicity. This way, the integers could be written into the program as read from left to right. The result was the same had the order of operations been observed.

When preforming arithmetic in Cortex-M0+ operations can not be preformed the same way they computed mentally (except for addition, which is preformed the same as it is done mentally). That being, multiplication by an odd number (as seen by the in Eq. (1.2)) must be represented by multiplying by powers of two and then adding the operand to the result. When dividing, any equation that would result in a fraction (as seen by the in Eq. (1.2)) is represented by truncating the number to only represent an integer value. Subtraction is represented by addition of the operand the negative signed value of the operator.

In order to compute Equation (1.2) the value of each operand was stored in registers. By using Cortex-M0+ operations and equate statements the final value of the equation was found, stored in a register, and verified to be correct.

**Procedure**

Since this exercise was preformed in ARM Cortex-M0+ assembly language the program was written in the Keil MDK-ARM IDE. Using a template made available in a previous exercise, only the main program code and two equate statements had to be written. This main program code the arithmetic was preformed one operation at a time.

To preform the first operation, the addition of , first the value of had to be stored in register R2 using the MOVS command. Once stored in R2 the value then had to be negated. This operation was preformed by RSBS command, which takes the value of a register and subtracts that value from zero. The result is a negative value of the same magnitude and this value was stored in register R1. Then by using the MOVS command, once again, the value was stored in register R2. To add the values together the ADDS command was used to combine the contents of R1 and R2 and the final value was stored in R0.

The next operation that was preformed was the division of . In order to preform this operation an equate statement was necessary to meet the guidelines. The equate statement, called DIV4, told the assembler to shift the value it was called on by two bits and truncate and reminder. The equate statement reduced the amount of code needed in the main program code. Once a value of was stored in R1 using the MOVS command, the equate statement was called on the contents of R1 and the result was stored in R1. The result was a truncated value of . Finally this value was subtracted from the contents of R0 using the SUBS command. The result was stored in R0.

To preform multiplication, which was necessary for the operation , another equate statement was written. This one being named MULT8, what this command did was shift the value called on by 3 bits to the right. This multiplied the number by 8. Once a value of was stored in R1 the equate statement was called and the value of the operation was stored in R2. Since the operation needed was to multiply by an additional ADDS command was necessary. This essentially multiplied the initial value by 9. The result was stored in R1 and then the value of R1 was subtracted from the contents of R0.

Finally, the last two operations were written. These operations were . To preform each one of these operations the values of the operators were stored in register R1, one at a time, and then by using the ADDS command the values were, once at a time, added to the final value in register R0. The result was the final value of Equation (1.1).

**Results**

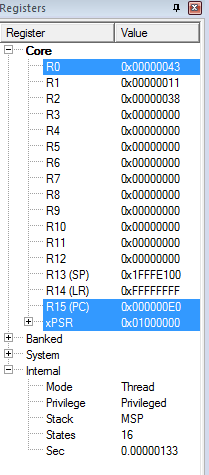
Once assembled, built, and verified to be debugged the simulation was preformed. By stepping through the code, one line at a time, and reading the contents of the registers in the simulator window it was verified that each operation was correctly solved preformed. 

Figure (1.1): Simulation Final Result

Figure (1.1) shows the final result, which was stored in register R0, and this value was verified to be the correct result. Since the computer preforms the operations in hexadecimal, the result of Equation (1.2) has to be fist computed by hand. The final value stored in R0 was then verified by comparing it to this value and once verified it was confirmed that the all correct operations had taken place.

**Conclusion**

In this exercise, a given equation was solved using the Cortex-M0+ assembly language. The equation was solved while respecting certain limitations that were also given. Such as a limitation on the how many registers could be used and the mandates of equate statements for multiplication and division. The objective of the exercise was familiarization of basic arithmetic using the Cortex-M0+ and this objective can be considered achieved. The correct results and implementation of the computation prove that exercise was successful.