CMPE-250
Assembly and Embedded Programming
Spring 2021
Laboratory Exercise Three
Memory, Conditional Branching, and Debugging Tools

This exercise investigates memory use and conditional branching in a Cortex-M0+ assembly language program, as well as use of the Keil MDK-ARM's debugger. The objective of this exercise is familiarization with Cortex-M0+ condition codes and branch instructions, as well as creation and access of constants and variables in memory. An assembly language program is created and simulated in Keil MDK-ARM.

Prelab Work

For the three expressions and two sets of inputs below (from the program specification and lab procedure step 4, respectively) calculate and write the results of each arithmetic operation in evaluating the expressions *F*, *G*, and *Result* from left to right, observing standard algebraic order of operations. Note where byte overflow would occur for each operation in evaluating an expression; if overflow would occur, stop calculation for that expression, and write zero for the result of that expression. In lab, these calculations will be used to verify the execution of your program, so you need to perform two calculations for *F*, *G*, and *Result*: once with input set 1 and then again with input set 2. Include these calculations in your documentation to validate your program's results.

$$F = 3P + 2Q - 75$$
$$G = 2P - 4Q + 63$$
$$Result = F + G$$

Input Set 1: $P = 9$; $Q = 4$

Input Set 2: $P = 13$; $Q = -14$

Application

Many devices that might be controlled or might be accessed by a microcontroller may have a different word size from the microcontroller. For example, some devices work with only 8-bit or 16-bit data, whereas the Cortex-M0+ word size is 32 bits. This assignment involves verifying that the results of 32-bit signed computations performed by the Cortex-M0+ are valid for output to an 8-bit device, (i.e., that they are valid signed byte results).

Program Specification

Write a properly commented and properly formatted Cortex-M0+ assembly language program to compute the expressions below.

$$\begin{aligned} F &= 3P + 2Q - 75 \\ G &= 2P - 4Q + 63 \\ Result &= F + G \end{aligned}$$

In addition to computing the correct arithmetic result, the program must meet the following requirements.

- *F*, *G*, *P*, *Q*, and *Result* must be word variables in memory.
- The constant operands required must be *word* constants in ROM allocated using assembler DCD directives, as follows.

$$\begin{aligned} const\_F &= 75 \\ const\_G &= 63 \end{aligned}$$

- Multiplication operations must be implemented with a combination of shift and addition instructions. Assembler EQUate directives must be used to define the shift amounts for the multiplication operations, (e.g., `MUL2 EQU 1`, `MUL4 EQU 2`, etc.).
- The program must first load the registers indicated below for *P* and *Q* before computing the expressions.

R1: *P*
R2: *Q*

- The expressions, as written without any simplification, must be evaluated from left to right, observing standard algebraic order of operations.
- The program must check for byte overflow after every operation, (i.e., a result greater than +127 or less than −128.) If byte overflow occurs during the computation of any output variable, (i.e., *F*, *G*, or *Result*), the computation of that variable should be stopped, and its value should be set to zero. (Note: word variables, constants, and instruction operands are used in the program; however, the purpose of this requirement is to ensure that all results can be represented as valid byte values, [i.e., all significant bits of a two's complement result fit within one byte].)
  - *F* must be computed using the exact values in variables *P* and *Q*, and overflow for each arithmetic (or shift) operation in the computation of *F* must be determined by checking whether the result is in the range [−128, +127].
  - *G* must be computed using the values in variables *P* and *Q* shifted to the most significant byte of the word operands for arithmetic operations, and overflow for each arithmetic operation in the computation of *G* must be determined from the V condition code flag. (Note: V cannot be used to determine overflow for shift operations, so you may choose to perform shift operations in the computation using values in either the most significant byte or the least significant byte of word operands.)
  - *Result* may be computed using any algorithm that gives the correct result from the values in variables *P* and *Q*, and overflow for each arithmetic (or shift) operation in the computation of *Result* may be determined by any correct means.

Lab Procedure

1. Create a new directory (folder) and Keil MDK-ARM project for this exercise.
2. Write a properly commented and properly formatted Cortex-M0+ assembly language program according to the preceding program specification.
3. Assemble and build the program with Keil MDK-ARM to create a listing file and a map file.
4. For each set of input variables, (i.e., *P* and *Q*), shown below, simulate the program in the simulator, and obtain a screen capture of the final simulation results to include in your documentation. (Each screen capture should clearly show contents of all registers and of memory variables *F*, *G*, and *Result*.) To set the values of the variables, modify the memory contents in the simulator, (i.e., so you do not have to reassemble the program for different inputs).

   Input Set 1: $P = 9; Q = 4$

   Input Set 2: $P = 13; Q = -14$
5. Demonstrate step 4 for your lab instructor. Be prepared to demonstrate additional input sets your lab instructor may request to verify correct operation.
6. Submit your source file (.s), listing file (.lst), and documentation to the respective myCourses assignments for this assignment.

Baseline Metrics

The metrics in the table below can be used to compare your solution to a baseline solution. They are provided solely for your information and personal curiosity about the efficiency of your solution. There are no grading criteria associated with how your code compares with them.

Code	Instructions/ LOC	Code Size (bytes)	Image Size (bytes)
Assembly*	77	154	716
C	50	326	860

*Not including instructions from provided program template.

Documentation

Prepare a document (in Microsoft Word or PDF format) containing the following.

- The screen captures from step 4. (Each screen capture should clearly show contents of all registers and of memory variables *F*, *G*, and *Result*.)
- The calculations from prelab work that validate the results obtained in step 4.
- The answer to the following question.
  Could you reduce or eliminate overflow by changing the order of operations within the expressions (*F*, *G*, and/or *Result*)? Explain why or why not.

Submission

myCourses Assignments
Separate assignment for each item below • Documentation (including cover sheet) • Source code (.s) • Listing (.lst)

As specified in "Laboratory and Report Guidelines," late submissions are penalized per day late, where "day late" is calculated by rounding submitted time to the nearest greater or equal integer number of days late, (i.e., ceiling function).

Note: To receive grading credit for this lab exercise you must earn points for *both* the demonstration *and* the report.