CMPE-250
Assembly and Embedded Programming
Spring 2021
Laboratory Exercise Four:
Iteration and Subroutines

This exercise investigates iteration and subroutines in a Cortex-M0+ assembly language program. The objective of this exercise is familiarization with Cortex-M0+ condition flags for conditional branching to support iteration and for output parameters, as well as subroutines without side effects. An assembly language program is created and simulated in Keil MDK-ARM to implement and test an integer division subroutine.

Prelab Work

Write a subroutine `DIVU` to perform unsigned integer division of the dividend in R1 by the divisor in R0, and to return the quotient in R0 and the remainder in R1.

R1 ÷ R0 = R0 remainder R1

If the subroutine is called with R0 = 0, it should leave the input parameters unchanged and set the C flag on return. Otherwise, the C flag should be cleared on return. You must write your subroutine such that no register other than the outputs and the condition flags has changed value after return. (Note that the correct result of 0 ÷ *n* for nonzero *n* is 0 remainder 0.) Below is a summary of the parameters for subroutine `DIVU`.

Application

Division is a commonly used mathematical operation. For binary computing systems, user interfaces that show decimal representations of binary numbers generated and stored by a processor require division by ten to convert from base 2 to base 10. However, the Cortex-M0+ instruction set does not include any divide instructions. This assignment develops a subroutine `DIVU` to perform unsigned integer division that will be used in subsequent assignments.

Program Specification

Write a properly commented and properly formatted Cortex-M0+ assembly language program to test the `DIVU` subroutine from prelab work. The program must meet the following requirements.

- Subroutine `DIVU` must be included, as specified for prelab work, to compute unsigned integer division: R0 remainder R1 = R1 ÷ R0.

  Input parameters:
  - R0: divisor (unsigned word by which to divide)
  - R1: dividend (unsigned word to divide)

  Output parameters:
  - R0: quotient (unsigned word result of integer division)
  - R1: remainder (unsigned word remainder from integer division)
  - C: APSR flag: 0 for valid result; 1 for invalid result

- Registers R6 and R7 *must not be modified*.
- `MAX_DATA` must be defined as an EQUate to 25.
- The following IMPORT directives must be in the `MyCode` AREA. (Suggestion: near the top, after the EXPORT `Reset_Handler` directive.)

```
IMPORT InitData
IMPORT LoadData
IMPORT TestData
```

- `P` and `Q` must be defined as word variables.
- `Results` must be defined as a word array with 2 × `MAX_DATA` elements.
- The following EXPORT directives must be in the `MyData` AREA.

```
EXPORT P
EXPORT Q
EXPORT Results
```

- The main program should perform the following steps.
  1. First call the subroutine `InitData`, which is provided in the `Exercise04_Lib.lib` library.
  2. Next, call the subroutine `LoadData`, also provided in the library, which will load `P` and `Q`. If the C flag is set after return, then there were no more data, and your program should quit.
  3. Otherwise, use your subroutine `DIVU` to divide `P` by `Q`, (i.e., initialize input parameters R0 and R1 correctly for it to compute `P` ÷ `Q`, and then call `DIVU`).

     R0 remainder R1 = P ÷ Q

  4. If the C flag is clear after return, the division result is valid, so store it in `P` and `Q` so that `P` contains the quotient and `Q` contains the remainder.

     P remainder Q = P ÷ Q

     Otherwise, if the C flag is set, store 0xFFFFFFFF in `P` and 0xFFFFFFFF in `Q`.
  5. Next, call the subroutine `TestData`, provided in the library, which will check the division results and store them to the `Results` array. If the division results are wrong, `TestData` will also increment R6.
  6. Repeat from step 2.

Lab Procedure

1. Create a new directory (folder) and Keil MDK-ARM project for this exercise.
2. From myCourses, download the `Exercise04_Lib.lib` library provided for this exercise, place it in the directory of your new project, and add it to the project's Source Group 1 by doing the following.
   a. Right-click on Source Group 1.
   b. Select Add Existing Files to Group 'Source Group 1'
   c. Select `Exercise04_Lib.lib`.
3. Write a properly commented and properly formatted Cortex-M0+ assembly language program according to the preceding program specification.
4. Assemble and build the program with Keil MDK-ARM to create a listing file and a map file.
5. Simulate the program in the debugger, and capture the final simulation results to include in your report. (Note: as the program runs, R6 keeps a count of the number of incorrect results detected by `TestData`. If R6 = 0 and R7 = 0xA0 when the program ends, your `DIVU` subroutine has computed and returned all correct results.)
6. Demonstrate steps 4–5 for your lab instructor.
7. Examine the memory map for your project, and determine the exact memory ranges, (i.e., the starting address and the address of the last byte), for the components listed below, to include and discuss in the results section of your report.
   a. Code you wrote for main program—not including code provided in the program template and not `DIVU` or other subroutine code. (Note: You will also need to examine the listing file to isolate only the addresses of instructions you wrote for main.)
   b. Exercise04_Lib.lib library code
   c. RAM used for variables
   d. RAM used for stack (Hint: Find the AREA where the stack is allocated.)
8. Submit your source file (.s), listing file (.lst), map file (.map), and report to the respective myCourses assignments for this assignment.

Baseline Metrics

The metrics in the table that follows 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[1]/ LOC	Code/ROM Size (bytes)	Execution Time (μs)[2]
main program	15	44	
DIVU (algorithm from notes)	28	64	
Assembly Image Total		**1248**	**2390.58**
DIVU (faster algorithm)	53	106	
Assembly Image Total		**1288**	**217.83**
C program: *Remainder = P % Q*	17	94	
C Image Total		**1448**	**357.58**
C program: *Remainder = P − (Quotient × Q)*	17	94	
C Image Total		**1448**	**308.17**
C program: stdlib div function that returns both quotient and remainder[3]	16	94	
C Image Total		**1464**	**322.25**
C program: stdlib div and lldiv functions that returns both quotient and remainder[4]	18	162	
C Image Total		**1924**	**382.75**
C program: DIVU assembly language subroutine (faster algorithm)	14	162	
C Image Total		**1192**	**326.17**

[1]Not including instructions from provided program template

[2]Xtal (MHz) = 12.0 in Target Options → Target tab

[3]stdlib div function provides 32-bit signed division (gives wrong results for operands greater than 0x7FFFFFFF)

[4]stdlib lldiv function provides 64-bit signed division (used instead of div for operands greater than 0x7FFFFFFF)

Report

Write a report consisting of the following sections. Your writing should follow the rules of professional technical writing and should meet the specifications in "Laboratory and Report Guidelines" on myCourses.

- Abstract
- Procedure
- Results.
  - ☞ Include the screen capture from step 5. (Remember to introduce and explain it in text rather than merely pasting it into the report.)
  - ☞ Include the memory ranges from step 7. (Discuss how you determined them and explain them.)
- Conclusion

Submission

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

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.