CMPE-250
Assembly and Embedded Programming
Spring 2021
Laboratory Exercise Five:
Polled Serial I/O

This exercise investigates serial input and output (I/O) on the NXP Freedom Board KL05Z using a universal asynchronous receiver/transmitter (UART). The objective of this exercise is to produce basic subroutines for polled serial I/O of characters and to test them. An assembly language program is created and is executed on the KL05Z board.

Prelab Work

First, write the subroutine Init_UART0_Polling to initialize the KL05 as discussed in class and presented in the class notes for polled serial I/O with UART0 through port B pins 1 and 2 using this format: eight data bits, no parity, and one stop bit at 9600 baud. You must write the subroutine so that no registers other than LR, PC, and PSR have changed values after return.

Next, following the specifications below, write subroutines for polled serial I/O of characters. You must write the subroutines so that no registers other than those used for output, LR, PC, and PSR have changed values after return. (Note: GetChar uses R0 for an output parameter, so R0's value must be changed by GetChar.) The notes from lecture give you the basis for GetChar and PutChar.

- GetChar: Reads a single character from the terminal keyboard into R0.
- PutChar: Displays the single character from R0 to the terminal screen.

Application

On an embedded development platform such as the NXP Freedom Board KL05Z used in lab, only terminal character input and output are generally available for providing a user interface. To minimize user effort required for input, one-character commands (rather than whole words) are often used in such interfaces. In this exercise, the developed GetChar and PutChar subroutines are applied to make a one-character command "menu," which can be adapted easily to other exercises throughout the semester.

The command menu for this exercise provides a basic interactive glossary of the ARM Cortex-M0+ APSR condition flags. When the name of a condition flag is typed, an explanation is shown to the user. Below are the entries in the glossary.

- C or c: Carry: set if there was a carry out of or no borrow into the msb.
- N or n Negative: set if result is negative for signed number; N = msb.
- V or v: Overflow: set if result is invalid for signed number.
- Z or z: Zero: set if result is zero.

Program Specification

To test your code from prelab, write a program to echo (on the terminal screen) characters typed on the terminal keyboard. Additionally, if the character belongs to the set of command characters {C, c, N, n, V, v, Z, z} execute the appropriate subroutine (provided in a library for this exercise) to show the glossary entry associated with that character. The program must perform the following sequence of operations.

1. Using Init_UART0_Polling, initialize the KL05 UART0 for polled serial I/O using a format of eight data bits, no parity, and one stop bit.
2. Call the provided subroutine PutPrompt. (Note: for the subroutine to work, you must add the Exercise05_Lib.lib provided for this exercise to your project, as directed in the lab procedure.)
3. Read a character typed on the terminal keyboard using GetChar.
4. Write the character from step 3 to the terminal screen using PutChar.
5. If the character is a lower-case alphabetic character, convert it to an upper-case alphabetic character. For example, if the user typed `c`, change it to `C`. (Having only one case of characters requires less code in the next step.)
6. If the character is `C`, `N`, `V`, or `Z`, execute the subroutine provided as indicated below, and then repeat the sequence starting from step 2. (Note: for the subroutine to work, you must add the Exercise05_Lib.lib provided for this exercise to your project, as directed in the lab procedure.)
    - `C:  BL Carry`
    - `N:  BL Negative`
    - `V:  BL Overflow`
    - `Z  BL Zero`
7. If the character is a carriage return character (<CR> = 0x0D), also output a line feed character (<LF> = 0x0A), and then repeat the sequence starting from step 2.
8. Otherwise, repeat the sequence starting from step 3.

Below is an example of the terminal screen output from pressing the following keys, where ↵ represents the enter key (to generate a carriage return character).

nZcVaB↵7zCNv

```
Type a letter for APSR glossary lookup.
>n
Negative:  set if result is negative for signed number; N = msb.
Type a letter for APSR glossary lookup.
>Z
Zero:  set if result is zero.
Type a letter for APSR glossary lookup.
>c
Carry:  set if there was a carry out of or no borrow into the msb.
Type a letter for APSR glossary lookup.
>V
Overflow:  set if result is invalid for signed number.
Type a letter for APSR glossary lookup.
>aB
Type a letter for APSR glossary lookup.
>7z
Zero:  set if result is zero.
Type a letter for APSR glossary lookup.
>C
Carry:  set if there was a carry out of or no borrow into the msb.
Type a letter for APSR glossary lookup.
>N
Negative:  set if result is negative for signed number; N = msb.
Type a letter for APSR glossary lookup.
>v
Overflow:  set if result is invalid for signed number.
Type a letter for APSR glossary lookup.
>
```

The program must use the subroutines from prelab work for the character I/O operations. The subroutines must comply with the prelab specifications. Also, each subroutine must have a comment header block that documents these aspects:

- Description of subroutine functionality,
- Input parameter list,
- Output parameter list, and
- Register modification list.

The required subroutines from prelab work have the following parameter specifications.

- GetChar
  - Input parameter:
    - (none)
  - Output parameter:
    - R0: character received from terminal (unsigned byte ASCII code)
- Init_UART0_Polling
  - Input parameter:
    - (none)
  - Output parameter:
    - (none)

- PutChar
  Input parameter:
  R0: character to send to terminal (unsigned byte ASCII code)
  Output parameter:
  (none)

Lab Procedure

1. Create a new directory (folder) and Keil MDK-ARM project for this exercise.
2. Place copies of the MKL05Z4.s and Start.s files in your project directory, as directed in the tutorial for this exercise. Furthermore, add Start.s—but not MKL05Z4.s—to your project's Source Group 1.
3. From myCourses, download the `Exercise05_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.
4. Write a properly commented and properly formatted KL05 assembly language program according to the preceding specification.
5. At the beginning of the `MyCode` AREA, along with EXPORT `Reset_Handler` and IMPORT `Startup`, add the following EXPORT and IMPORT directives needed to use the `Exercise05_Lib.lib` library provided for this exercise.
   ```
   EXPORT PutChar
   IMPORT Carry
   IMPORT Negative
   IMPORT Overflow
   IMPORT PutPrompt
   IMPORT Zero
   ```
6. Build the program with Keil MDK-ARM to assemble it and create a listing file and to link it and create a linker map file.
7. Test your program, which must be demonstrated to the instructor.
8. Demonstrate your program for the lab instructor. Following demonstration, capture the terminal screen output.
9. Submit your source (.s) file, listing (.lst) file, map (.map) file, and documentation to the respective myCourses assignments for this exercise.

Baseline Metrics

A Keil MDK-ARM C project solution is provided in Exercise06_C_Project.zip. The metrics in the table below can be used to compare your solution to a baseline assembly solution as well as the C project. 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.

Language	Routine	Instructions/LOC	Code Size (bytes)
Assembly	main program[1]	32	82
C	main	29	118
Assembly	Init_UART0_Polling	54	108
C	Init_UART0_Polling	18	120
Assembly	GetChar[2]	11	22
C	GetChar[2]	4	26
Assembly	PutChar	9	18
C	PutChar	3	20
Assembly	**Total RO Size**[3]	—	1236
C	**Total RO Size**[3]	—	1440

[1]Not including instructions from provided program template.
[2]Includes code to clear possible receive overrun condition.
[3]Includes Exercise05_Lib and Start.

Documentation

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

- The screen capture from step 8.
- The answer to the following question.
  From the listing file and KL05 memory map produced for your program, what is the exact memory range, (i.e., the starting address and the address of the last byte), of the following blocks of code?
  1. `MyCode` AREA
  2. Executable code in `MyCode` AREA
     (Hint: to determine size, look in listing file for offset indicated at `;>>>>> end subroutine code <<<<<` comment.)
  3. `main` program
     (Hint: see `Reset_Handler` in map file.)
  4. `GetChar` subroutine
  5. `PutChar` subroutine

Submission

myCourses Assignments
Separate assignment for each item below • Documentation (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.