CMPE-250
Assembly and Embedded Programming
Spring 2021
Laboratory Exercise Six:
Secure String I/O and Number Output

This exercise investigates serial string input and output (I/O) and number output on the NXP Freedom KL05Z using the KL05 universal asynchronous receiver/transmitter (UART) and preventing buffer overrun. The objective of this exercise is to produce secure subroutines for serial I/O of strings and to write a specified program to test them. An assembly language program is created and is executed on the KL05.

Prelab Work

Following the specifications below, write subroutines for secure I/O of strings, (i.e., preventing buffer overrun), and for output of the decimal representation of numbers. You must write the subroutines so that no registers other than those used for output parameters and PSR have changed values after return. When character I/O is required, your subroutines must use GetChar and PutChar from Lab Exercise Five.

- GetStringSB: Preventing overrun of the buffer capacity specified in R1, this subroutine inputs a string from the terminal keyboard to memory starting at the address in R0 and adds null termination. It ends terminal keyboard input when the user presses the ↵ enter key. For each of up to R1 – 1 characters typed on the terminal keyboard, it uses GetChar to input the character, uses PutChar to echo the character to the terminal screen, and stores the character at the next position in the string. For any character typed after the first R1 – 1 characters, it uses GetChar to input the character, but it neither stores the character in the string nor echoes the character to the terminal screen. When the carriage return character has been received, it null terminates the string, advances the cursor on the terminal screen to the beginning of the next line, and returns.
  Note: There is an extra credit opportunity if you write the subroutine to ignore control code input—see Extra Credit Opportunity section.
- PutStringSB: Preventing overrun of the buffer capacity specified in R1, this subroutine displays a null-terminated string to the terminal screen from memory starting at the address in R0. It uses PutChar to display characters from the string and leaves the terminal screen cursor positioned after the last character of the string.
- PutNumU: This subroutine displays the text decimal representation to the terminal screen of the unsigned word value in R0, using PutChar to output each decimal digit character. (For example, if R0 contains 0x00000100, then 256 should be output; 0000000256 would also be acceptable.)
  Hint: Use DIVU from Lab Exercise Four to determine decimal digit values.
  Note: There is an extra credit opportunity for not printing leading zeros, (e.g, 256 instead of 0000000256)—see Extra Credit Opportunity section.

Application

The program developed for this exercise adapts the one-character command "menu" developed in Lab Exercise Five to test the basic string I/O and number output subroutines developed in prelab work. The program uses string I/O and number output subroutines from prelab work for prompts, strings, and numbers, and these subroutines call GetChar and PutChar subroutines developed for Lab Exercise Five. In addtion, the program uses a provided library subroutine to calculate string length. The command menu implemented for this exercise works with one string: "the operational string." Below are the program commands for the operational string.

Command	Description
G or g	Get a string the user types on the terminal keyboard, and store it in the operational string.
I or i	Initialize the operational string to an empty string.
L or l	Print the length of the operational string to the terminal screen.
P or p	Print the operational string to the terminal screen.

Program Specification

Write a program that calls the subroutines from prelab work and from the provided library to perform the following sequence of operations. The program must meet the requirements that follow.

1. Initialize the KL05 UART0 for polled serial I/O at 9600 baud using a format of eight data bits, no parity, and one stop bit—as in Lab Exercise Five.
2. Output user instructions to the terminal screen, exactly as shown below.

```
Type a string command (G,I,L,P):
```

3. Read a character typed on the terminal keyboard.
4. If the character is a lower-case alphabetic character, convert it to an upper-case alphabetic character. For example, if the user typed g, change it to G. (Having only one case of characters requires fewer lines of code in the next step.)
5. If the character typed is not a valid command, (i.e., $\in$ {G, g, I, i, L, l, P, p}), ignore the character and repeat from step 3.
6. Otherwise, write the character from step 3 to the terminal screen and then output a carriage return character (0x0D) and a line feed character (0x0A) to move the cursor to the next line of the screen, as shown below for G.

```
Type a string command (G,I,L,P):G

```

7. Perform the command as specified below.
   - G or g: Print <, and then read the string entered from the terminal keyboard using GetStringSB, as shown in the example below for user input of `Test string.`↵, where ↵ indicates pressing the enter key.

     ```
     Type a string command (G,I,L,P):g
     <Test string.
     ```

   - I or i: Initialize the operational string to an empty string, (i.e., store null in the first byte), and then advance to the beginning of the next line, as shown below.

     ```
     Type a string command (G,I,L,P):i
     ```

   - L or l: Print `Length:`, use LengthStringSB from the provided library to determine the length of the operational string, use PutNumU to print the length, and then advance to the beginning of the next line, as shown below.
     Note: LengthStringSB has two input parameters: R0 is the address of the string buffer, and R1 specifies the buffer capacity in bytes. It returns the length of the string in R0. For LengthStringSB to work, you must add the Exercise06_Lib.lib provided for this exercise to your project, as directed in lab procedure steps 2 and 4.)

     ```
     Type a string command (G,I,L,P):g
     <Test string.
     Type a string command (G,I,L,P):l
     Length:12
     ```

   - P or p: Print >, use PutStringSB to print the operational string on the terminal screen, print >, and then advance to the beginning of the next line, as shown below.

     ```
     Type a string command (G,I,L,P):g
     <Test string.
     Type a string command (G,I,L,P):p
     >Test string.>
     ```

8. Repeat from step 2.

In addition, the program must meet these requirements.

- It must use subroutines from prelab work for string I/O and for number output. The following is a summary of the parameter specifications for each of these subroutines. Note: For output by reference, (e.g., string buffer for GetStringSB), the reference, (i.e., address), must be passed into the subroutine—looks like an input parameter before call.
  - GetStringSB
    Input parameter:
    R1: bytes in string buffer where R0 points (unsigned word value)
    Output parameter:
    R0: string buffer in memory for input from user (unsigned word address)
  - PutNumU
    Input parameter:
    R0: number for output to terminal (unsigned word value)
    Output parameter:
    (none)
  - PutStringSB
    Input parameter:
    R0: string buffer in memory for output to simulated output stream (unsigned word address)
    R1: bytes in string buffer where R0 points (unsigned word value)
    Output parameter:
    (none)
- It must use Exercise06_Lib.lib library subroutine LengthStringSB for string length.
- It must use MAX_STRING, (defined with EQUate as 79, which is the maximum number of string characters, including the null termination character), wherever the maximum number of string characters is needed in the program code, (e.g., parameter for SPACE to allocate a string buffer, and buffer size input parameter for GetStringSB and PutStringSB).
- It must have properly commented subroutines, which each have a comment header block that includes these components.
  - Description of functionality
  - List of subroutines called
  - Lists of register parameters: input, output, and modified

Lab Procedure

1. Create a new directory and Keil MDK-ARM project for this exercise.
2. Copy the Exercise06_Lib.lib file (provided on myCourses for this exercise) to your project directory and add it to your project.)
3. Write a properly commented and properly formatted KL05 assembly language program according to the preceding specification.
4. Add the following directive needed for the Exercise06_Lib library in the MyCode AREA of your program after the EXPORT Reset_Handler directive already in the program template. (The IMPORT directive for the Exercise06_Lib subroutine makes it available for your program to use.)
   ```
   IMPORT LengthStringSB
   ```
5. Build the program with Keil MDK-ARM to create a listing file and a linker map file.
6. Test program, including these test cases.
   - Uninitialized string, (i.e., at program start): l and p
   - User string of between 1 and 77 characters (inclusive): g, l, and p
   - User empty string, (i.e., only ↵ enter key pressed): g, l, and p
   - User string of 78 characters: g, l, and p
   - Initializing string to empty string: i, l, and p
   - User string attempting more than 78 characters: g, l, p
7. Examine the KL05 memory map and listing produced for your program, 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 in the documentation for this exercise.
   a) Executable code in main program
   b) MyCode AREA,
   c) Constants (including all prompt and annotation strings), and
   d) Variables (including the string buffer for the operational string).
8. Also from the KL05 memory map produced for your program, note the code size, (i.e., number of bytes), for each of the following subroutines you wrote, to include in the documentation for this exercise.
   a) GetStringSB,
   b) PutNumU, and
   c) PutStringSB.
9. Demonstrate procedure steps 5 and 6 for your lab instructor. Following demonstration, capture the terminal screen output to include and discuss in the results section of your report.
10. Submit your source (.s) file, listing (.lst) file, map (.map) file, and documentation to the myCourses assignments for this exercise.

Extra Credit Opportunity

GetStringSB (10 points total)
Control characters (5 points). Design GetStringSB so that it does not accept any control characters, (i.e., 0x00–0x1F and 0x7F), that are not explicitly handled by program code. Note: Of these control characters, the program specification for this lab exercise requires program code to handle only carriage return, 0x0D.
Optional: For an even more robust GetStringSB, you might also consider filtering terminal escape sequences, which begin with escape (<ESC>, ASCII code 0x1B) followed by [along with one or more additional characters, and which end with ~. (For example, pressing the Home key results in the escape sequence <ESC>[1~. Note that by default, PuTTY does not correctly pass all terminal escape sequences, so some specialized keyboard keys will still cause problems, even with this additional input filtering. For testing purposes, the Escape, Home, and Page Up keys should work correctly.)

Backspace functionality (5 points). Design GetStringSB so that backspace (<BS>, ASCII code 0x08) works correctly, (i.e., so that characters can be erased from the end of the string typed so far.) You will first need to make sure PuTTY is set to send <BS> when the backspace key is pressed by looking at PuTTY configuration settings for Keyboard options under the Terminal category to make sure Control-H is selected for the backspace key.
Note: PutChar of <BS> to the terminal results in moving the cursor back one character. Full backspace functionality requires these additional actions: the character needs to be overwritten with a space, the cursor needs to be moved back to the overwritten location, and the length of the string needs to be adjusted. For correct operation, make sure you also consider the special cases of empty and full strings.

PutNumU (10 points). Design PutNumU so that it does not output any leading zeroes to the left of the first significant digit, as illustrated by the table below.

Value	Output (Basic)	Output (Extra Credit)
0	0000000000	0
1	0000000001	1
10	0000000010	10
100	0000000100	100

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.

Language	Routine	Instructions/LOC	Code Size (bytes)
Assembly	main program*	56	154
C	main	40	216
Assembly	GetStringSB (basic)	25	60
C	GetStringSB (basic)	15	90
Assembly	GetStringSB (extra credit)	57	136
C	GetStringSB (extra credit)	24	138
Assembly	PutNumU (basic)	18	44
C	PutNumU (basic)	6	62
Assembly	PutNumU (extra credit)	15	34
C	PutNumU (extra credit)	8	54
Assembly	PutStringSB	14	30
C	PutStringSB	5	38
Assembly	**Total RO Size (basic)**	—	1236
C	**Total RO Size (basic)**	—	1720
Assembly	**Total RO Size (extra credit)**	—	1236
C	**Total RO Size (extra credit)**	—	1760

*Not including instructions from provided program template.

Documentation

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

- The screen capture from procedure step 9.
- The memory ranges from procedure step 7.
- The subroutine code sizes from procedure step 8.
- The answers to the following questions.
  1. Why does the number of user-typed characters stored in the string have to be one fewer than the number of bytes allocated for the string?
  2. Why does `MAX_STRING` need to be defined as an EQUate rather than as a DCD in the `MyConst` AREA, (i.e., a constant)?

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.