CSE 379
Lab #3
Part 2
Spring 2022

Description
Write four (4) ARM assembly language subroutines: *read_string*, *output_string*, *uart_init*, *int2string*, *string2int*, and *lab3*. These subroutines will allow the ARM processor to receive user-entered data from *PuTTy* and transmit data to be displayed in *PuTTy* via the UART. All user input should be echoed back to the display via your program. Your program will ask the user to enter two integers and report the sum of the two integers after the operation is performed on the two numbers. The numbers will be integers in the range of 0 through 99,999 (inclusive). When entering the numbers, commas will NOT be used for numbers greater than 999. The user should not be asked how many digits are in the numbers. The numbers should be entered in decimal. After the user enters each number, they should hit the *Enter* key to indicate that they have finished entering the number. Each integer user-entered integer should be stored as a NULL terminated ASCII string in memory at the labels *num_1_string* and *num_2_string* respectively. The result should be stored as a NULL terminated ASCII string in memory at the label *result*. After the result is displayed the user should be allowed to run the program again or quit.

Presentation is important. For example, when user data is expected, the program should prompt the user for data, and supply the appropriate instructions for doing so. When the program reports the results they should be descriptive, asking the user to enter each number, describing the result (sum) when displayed, and asking the user if they would like to run the program again.

The subroutines that need to be submitted (including those from Part #1) are defined as follows:

- *uart_init* initializes the user UART for use. This is your version (in assembly language) of the C function *serial_init*.
- *output_character* transmits a character from the UART to *PuTTy*. The character is passed in *r0*.
- *read_string* reads a string entered in *PuTTy* and stores it as a NULL-terminated ASCII string in memory. The user terminates the string by hitting *Enter*. The base address of the string should be passed into the routine in *r0*.
- *output_string* transmits a NULL-terminated ASCII string for display in *PuTTy*. The base address of the string should be passed into the routine in *r0*.
- *read_character* reads a character which is received by the UART from *PuTTy*, returning the character in *r0*.
- *int2string* stores the integer passed into the routine in *r0* as a NULL terminated ASCII string in memory at the address passed into the routine in *r1*.
- *string2int* converts the NULL terminated ASCII string pointed to by the address passed into the routine in *r0* to an integer. The integer should be returned in *r0*.
- *lab3* is the subroutine which handles getting the user data (number), calculating the sum, and displaying the result. This routine is called by your C program and is your top-level assembly language routine.

Skeleton Code
The following assembly language skeleton may be utilized to get you started. It has examples of how to store strings that can be used to prompt the user for input and describe the results that are displayed to the console in *PuTTy*. It also has a place for the two user entered strings and the sum (as a string). Each subroutine has also been setup for you in this example. The skeleton can be downloaded from the course website.

```
   .data

   .global prompt
   .global results
   .global num_1_string
   .global num_2_string

prompt: .string "Your prompts are placed here", 0
result: .string "Your results are reported here", 0
num_1_string:   .string "Place holder string for your first number", 0
num_2_string:   .string "Place holder string for your second number", 0

   .text

   .global lab3
U0FR:   .equ 0x18  ; UART0 Flag Register

ptr_to_prompt:     .word prompt
ptr_to_result:     .word result
ptr_to_num_1_string:  .word num_1_string
ptr_to_num_2_string:  .word num_2_string

lab3:
   PUSH {lr}   ; Store lr to stack
   ldr r4, ptr_to_prompt
   ldr r5, ptr_to_result
   ldr r6, ptr_to_num_1_string
   ldr r7, ptr_to_num_2_string

      ; Your code is placed here.  This is your main routine for
      ; Lab #3.  This should call your other routines such as
      ; uart_init, read_string, output_string, int2string, &
      ; string2int

   POP {lr}  ; Restore lr from stack
   mov pc, lr


read_string:
   PUSH {lr}   ; Store register lr on stack

      ; Your code for your read_string routine is placed here

   POP {lr}
   mov pc, lr


output_string:
   PUSH {lr}   ; Store register lr on stack

      ; Your code for your output_string routine is placed here

   POP {lr}
   mov pc, lr
```

```
read_character:
   PUSH {lr}   ; Store register lr on stack

      ; Your code for your read_character routine is placed here

   POP {lr}
   mov pc, lr


output_character:
   PUSH {lr}   ; Store register lr on stack

      ; Your code for your output_character routine is placed here

   POP {lr}
   mov pc, lr


uart_init:
   PUSH {lr}  ; Store register lr on stack

      ; Your code for your uart_init routine is placed here

   POP {lr}
   mov pc, lr

int2string:
   PUSH {lr}   ; Store register lr on stack

      ; Your code for your int2string routine is placed here

   POP {lr}
   mov pc, lr


string2int:
   PUSH {lr}   ; Store register lr on stack

      ; Your code for your string2int routine is placed here

   POP {lr}
   mov pc, lr

   .end
```

Testing

Use the C subroutine *serial_init* to get your program working. After your program works, replace *serial_init* with *uart_init*. By developing the program in this manner, it will be easier to identify and fix errors in *uart_init*.

Partners

You will work with the partner you signed up with during the week you worked on Lab #2.

Documentation

Your program must be clearly commented, and documentation must also be provided. The documentation must follow the guidelines covered in lecture (found on the *Lectures* webpage of the course website). Your comments should describe what *each* section of your program does. Before you may begin to work on Part #2 in Bonner 114, you MUST show flowcharts for your *output_string*, *read_string*, *int2string*, *string2int*, and *lab3* subroutines. The draft can be submitted on *timerlake.cse.buffalo.edu* using the submit command. It should be named *lab3_draft_flowchart.pdf*. Your code, along with your final documentation should be submitted on *timerlake* on or before the due date.

Submissions

Your source code (C and assembly) and your documentation (as a PDF) must be submitted online using the submit command (*submit_cse379 lab3wrapper.c lab3.s lab3documentation.pdf*) on *timberlake.cse.buffalo.edu* ***before 11:59 PM on Tuesday, March 1, 2022***. Your documentation will be used along with the code you submitted when you perform the debug exercise for Lab #3.