

Assembly Programming Assignment

Due date: 24:00 20/06/2021

Total marks: 7

The objectives of this assignment are (a) practicing writing assembly language programs, (b) understanding the conversion between ASCII code and binary value, and (c) manipulating information at bit level.

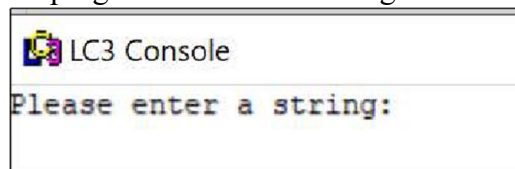
Part 1 (2 marks)

You are required to write an LC3 assembly language program to accept a sequence of letters entered by the user. Your program should carry out the following tasks:

1. Prompt the user to enter a string with message "Please enter a string:"
2. User enters a sequence of lowercase letters. The user ends the input with a LF character. The LF character is entered by pressing the "Enter/Return" key. The ASCII code of the LF character is 0x0a. It should be assumed that the user would enter at least one lowercase letter.
3. Your program should echo each of the letters entered by the user.
4. Name your program as p1.asm.

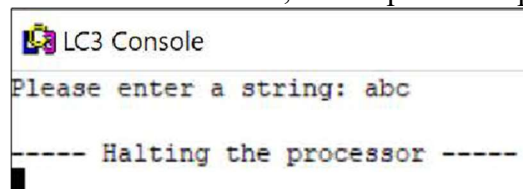
The following diagrams show a possible execution of the program.

When the program is run, the program shows the message as below:



```
LC3 Console
Please enter a string:
```

After the user enters "abc" and the LF character, the output of the program should be as below:



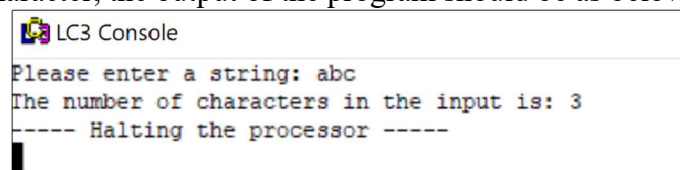
```
LC3 Console
Please enter a string: abc
---- Halting the processor ----
```

Part 2 (2 marks)

This part is based on Part 1. Your program should satisfy the following requirements:

1. After the user completes the input sequence, the program should display the number of letters that have been entered by the user. For this part, it should be assumed that the user enters at least one and at most nine letters.
2. Name your program as p2.asm.

The following diagram show a possible execution of the program. After the user enters "abc" and the new line character, the output of the program should be as below:



```
LC3 Console
Please enter a string: abc
The number of characters in the input is: 3
---- Halting the processor ----
```

Part 3 (2 marks)

This part is based on Part 2. Write a LC-3 assembly language program to (a) read in one string, (b) check whether the string is a palindrome, and, (c) display the result of the checking. The detailed requirements are as below:

1. A string is a palindrome if it can be read the same way in either direction. For example, “abcba” and “abba” are palindromes while “abca” and “abc” are not palindromes.
2. Like in Part 2, (a) the string consists of lowercase letters only, and (b) each string has at least one letter and at most nine letters.
3. Name your program as p3.asm.

The screenshots of some executions of the program are given below.

```
LC3 Console
Please enter a string: abc
The number of characters in the input is: 3
The input string is not a palindrome.
----- Halting the processor -----
```

```
LC3 Console
Please enter a string: abba
The number of characters in the input is: 4
The input string is a palindrome.
----- Halting the processor -----
```

Part 4 (1 mark)

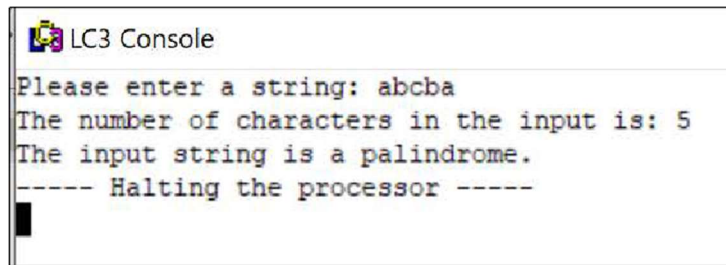
This part is based on Part 3. The detailed requirements are as below:

1. Like in Part 3, the program checks whether the input string is a palindrome. In this part, the input string has at least one and at most 19 lowercase letters.
2. Name your program as p4.asm.

The screenshots of some executions of the program are given below.

```
LC3 Console
Please enter a string: abcdefghijk
The number of characters in the input is: 11
The input string is not a palindrome.
----- Halting the processor -----
```

```
LC3 Console
Please enter a string: abcdefgfedcba
The number of characters in the input is: 13
The input string is a palindrome.
----- Halting the processor -----
```



```
LC3 Console
Please enter a string: abcba
The number of characters in the input is: 5
The input string is a palindrome.
----- Halting the processor -----
```

Submission

1. You MUST thoroughly test your program in the lab before submission. Programs that cannot be assembled or run on a lab machine will NOT get any mark.
2. Put p1.asm, p2.asm, p3.asm, and p4.asm in a folder and compress the folder in zip format. Name the zip file as YourStudentNumber-YourName-Assembly.zip and submit it to cloudcampus.

Some Hints

You are strongly encouraged to attempt each part first without reading these hints.

There are many ways to write the programs that satisfy the specifications. The hints given here are just some of the possible implementations. You can implement your programs differently as long as the outputs satisfy the specifications.

These are hints rather than a detailed step by step descriptions on how to implement the assignment. So, you still need to bridge the gaps that are not explicitly stated.

Part 1

- In order to obtain a sequence of letters in the input, you need to use the GETC instruction inside a loop. GETC does not echo the input character. So, you need to use the OUT instruction to echo the input character. Example 2b of Chapter 7 shows how to do this.
- To check the end of the input, you need to check whether the ASCII code of the input character is equal to the ASCII code of the LF character. Example 1 of Chapter 7 shows how to check the end of an input sequence.

Part 2

- You need to have a counter (i.e. a register) to record the number of letters entered by the user. According to the assumption for this part, the value of the counter must be between 1 and 9. So, the value is a single digit number.
- When you print out a number, you need to convert the number to its ASCII code before calling the OUT instruction to display the number. For example, to display number 9, the ASCII code of number 9 is 0x39. This is because device can only display ASCII characters. Example 1 of Chapter 7 shows how to convert the ASCII characters of digits to the value of the digits. This part is the reverse of the process in example 1. That is, you need to add 0x30 to the value of the counter.

Part 3

- To check for palindrome, you need to store the input string in the memory. So, you need to use “.blkw” directive to reserve some memory to hold the letters in the input string. Since there are at most 9 letters in the input string. You need to reserve at least 9 words.
- To check whether the string is a palindrome, you need to compare the corresponding letters at the start and the end of the string.
 - For example, for string “abba”, you need to compare the first letter and the last letter, i.e. the first “a” and the last “a”. If they are the same, you carry on to compare the second letter and the second to last letter, and so on.
- The address of the first letter of the string is the address of the label associated with the “.blkw” directive that you use to reserve memory locations for storing the letters in the string. You can obtain the address of this label using the “lea” instruction (refer to the example on LEA instruction in the slides of chapter 7). The address of the last letter of the string can be obtained according to the address of the first letter and the number of letters in the input string using the formula below:
$$(\text{the address of the first letter}) + (\text{the number of letters in the input string}) - 1$$
 - For example, if the starting address for storing string “abc” is x4000, the address of letter “a” is x4000 and the address of letter “c” is x4002.
- Example 3 of chapter 7 shows how to compare two values stored in the memory.
- As there are many letters in the string, you need to use a loop to compare each pair of corresponding letters at the beginning and the end of the string. Let p1 and p2 be two

pointers (i.e. registers) holding the addresses of the letters at the beginning and the end of the string respectively.

- If the two letters being pointed by p1 and p2 are not equal, the string is not a palindrome. So, we can terminate the loop.
- If the two letters being pointed by p1 and p2 are equal, we need to make p1 and p2 point to the next pair of letters to be compared. This is done by incrementing p1 and decrementing p2 because the beginning of the string is stored at the memory location with lower address compared to the end of the string.
 - After updating p1 and p2, if $p1=p2$, it means that p1 and p2 are pointing to the same letter. This means that the string must be a palindrome (why? Hint: Consider string “aba”, and p1 and p2 are both pointing to letter “b”).
 - The other case is $p1>p2$. This means all the corresponding letters that have been checked are all the same and all the corresponding letters have been checked (why? Hint: Consider string “abba”, and p1 is pointing to the second letter “b” while p2 is pointing to the first letter “b”).
 - Under the two conditions above, the loop can terminate.
 - Otherwise, you carry on to check the next pair of corresponding letters.

Part 4

- The only difference between Part 3 and 4 is the constraint on the number of letters in the input string. In Part 4, you can have up to 19 letters in the input string. As mentioned in Part 2, we have a counter to record the number of letters in the input string. As the number of letters might exceed 9, you cannot simply add 0x30 to the counter to obtain the value of the ASCII code of the digit that you want to display.
- First, you check whether the value of the counter is less than 10.
 - If it is, you just carry on to print out the value of the counter as you did in Part 2.
 - Otherwise, the value of the tens digit of the counter is 1, and the value of the ones digit is (the value of the counter – 10).
 - In this case, you print out the tens digit and the ones digit separately by adding 0x30 to the values of the tens and ones digits respectively.