Lab 2 Report

1 Introduction

In this lab, we will write a program in LC-3 assembly language to check whether two strings are anagrams.

2 Solution

To check whether the two given strings are anagrams, we use iterators twice. First we make a array that contains the frequency of every character in the string. In the first iteration we add 1 to the corresponding array. And we plus 1 to the character counter. In the second iteration, we minus 1 to the corresponding array. And we minus 1 to the character counter. In the end, we only check whether all the arrays and the character counter are all 0. If they are, then they are anagrams. We output "YES". Otherwise they are not, we output "NO".

2.1 Algorithm

```
procedure Anagram_Checker(str1, str2)
a[26] : integer;
c := 0 // initialize the character counter to 0
for i := 0 to 25 do
    a[i] = 0
x := *str1 // get the first character of the first string
while x != 0 do
    if x != 32
        c := c + 1 // count the number of the first string
        if x > 96 // check whether it's a lowercase
            x := x - 32
        a[x - 65] = a[x - 65] + 1
        str1 := str1 + 1 // move to the next character
    x := *str1
y := *str2 // get the second string
while y != 0 do
    if y!= 32
        c := c-1 // count the number of the second string
        if y > 96 // check whether it's a lowercase
            y := y - 32
        a[y - 65] := a[y - 65] - 1
    str2 := str2 + 1
    y := *str2
if c = 0 and a[i] = 0 (0 <= i <= 25)
    output "YES"
else
    output "NO"
```

2.2 Essential part of the code

The following part is the first loop which traverses every character in the first string and fills the frequency array.

When we convert the character to a 0-25 index, it's worth noting that the ASCII code of every uppercase character is in the form of 0b010X XXXX, which increase from 65 to 90. If we use an AND instruction to the ASCII code and 0b0001 1111 and then subtract the result by one, we'll convert the character to a 0-25 index. Then we add this value to R2, we get the address of the current element. Every time we get the corresponding address, we add 1 to the value and load it back to this address.

```
        LDR R4, R0, #0; R4 points to the first character
        LDR R6, R4, #0; R6 contains the character R4 points to
loop1   ADD R5, R6, #-16
        ADD R5, R5, #-16
        BRz SKIP1; judge whether it's a space
        ADD R3, R3, #1; count the number of the character
; then we check whether the character is in lowercase.
        ADD R5, R5, #-16
        ADD R5, R5, #-16
        ADD R5, R5, #-16
        ADD R5, R5, #-16
        BRn SKIP01; if it's uppercase, the ASCII code is smaller than 96
        ADD R6, R6, #-16
        ADD R6, R6, #-16; lowercase is 32 smaller than the uppercase
SKIP01  AND R5, R5, #0
        ADD R5, R5, #15
        ADD R5, R5, #15
        ADD R5, R5, #1
        AND R5, R6, R5
        ADD R5, R5, #-1; convert the character to a 0-25 index
        ADD R5, R5, R2; R5 contains the address of the current element
        LDR R7, R5, #0; R7 contains the current frequency count
        ADD R7, R7, #1; increment the frequency count
        STR R7, R5, #0; store the updated frequency count
SKIP1   ADD R4, R4, #1; move to the next character
        LDR R6, R4, #0
        BRz loop2; check if we reached the end of the first string
        BR loop1; otherwise, we continue the iteration
```

3 Q & A

> Q: How do you convert lowercase letters to uppercase letters?

A: First we check whether the ASCII code of this letter larger than 96 or not. If the answer is yes, we know it's lowercase. Secondly, we subtract 32 from its ASCII code. The result is the ASCII code of its uppercase.

> Q: How do you keep track of the number of times each letter appears?

A: We established an array to keep track of the frequency of every letters. In the first loop we add 1 to the corresponding element in the array every time we meet a letter. In the second loop, we subtract 1 from the corresponding element. At the end, we only need to check whether all the elements in the array are zero. It it is, they are anagrams.

> Q: How do you skip spaces in the given strings?

A: Every time we arrive at a new address, we first check whether the value of it equals 32. If so, it means this address stores a space. So we move to the next address.

> Q: How can you tell if you finished this string or not?

A: After we move to the next address, we first check if the value of the address is 0. If it is, that means we've finished this string. Then we jump out of the loop.