

**Faculty of Engineering & Technology**

**Electrical & Computer Engineering Department**

**COMPUTER DESIGN LABORATORY ENCS 4110**

**Experiment No. 3 -ARM Assembly**

**ARM's Flow Control Instructions**

**Report 1**

**Prepared by:**

Mohammad Abu Shams 1200549

**Instructor**: Dr. Abualseoud Hanani  
**Assistant**: Eng. Raha Zabadi

Section: 1

Date: 29-7-2022

Birzeit

# 

# **Abstract**

The experiment's goals include learning about ARM branch and compare instructions and how to use them for loops and conditional statements. In order to become more accustomed to and comprehend ARM assembly instructions, we will be dealing with strings by building some codes, debugging them, and then running them.

# **Table of Contents**

[Abstract I](#_Toc110275637)

[List of figures III](#_Toc110275638)

[List of tables IV](#_Toc110275639)

[Theory 1](#_Toc110275640)

[Setting Condition Code Flags 2](#_Toc110275641)

[The Encoding Format for Branch Instructions 2](#_Toc110275642)

[Branch and Control Instructions 5](#_Toc110275643)

[Examples of Compare Instructions 7](#_Toc110275644)

[Procedure 8](#_Toc110275645)

[Example 1 Using Branch Instruction 8](#_Toc110275646)

[Example 2 Using Branch Instruction 10](#_Toc110275647)

[Lab Assignment 12](#_Toc110275648)

[Task 14](#_Toc110275649)

[Conclusion 17](#_Toc110275650)

[Feedback 18](#_Toc110275651)

[References 19](#_Toc110275652)

# **List of figures**

[Figure 1: ARM register set [1] 1](#_Toc110020440)

[Figure 2: The encoding format for the branch instruction [3] 2](#_Toc110020441)

[Figure 3: ARM Instruction [5] 3](#_Toc110020442)

[Figure 4: Assembly code 1 8](#_Toc110020443)

[Figure 5: Debugging the first code 9](#_Toc110020444)

[Figure 6: Assembly code 2 10](#_Toc110020445)

[Figure 7: Debugging the second code 11](#_Toc110020446)

[Figure 8: Lab assignment code 12](#_Toc110020447)

[Figure 9: Debugging lab assignment code 13](#_Toc110020448)

[Figure 10: Task code 14](#_Toc110020449)

[Figure 11: Debugging task code 15](#_Toc110020450)

# **List of tables**

[Table 1: Condition in branch instruction 4](#_Toc110020389)

# 

# **Theory**

The default sequential execution is altered via ARM's flow control instructions. They manage the processor's functionality and the order in which instructions are executed. As seen in the diagram below, ARM contains 16 programmer-visible registers and a Current Program Status Register, or CPSR.

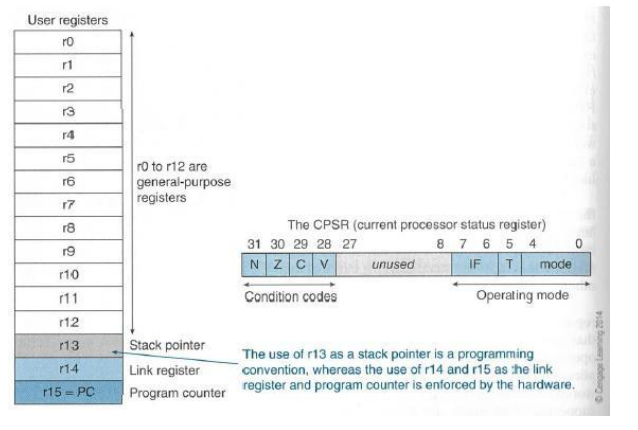


Figure 1: ARM register set [1]

R0 to R12 are the general-purpose registers.

R13 is reserved for the programmer to use it as the stack pointer.

R14 is the link register which stores a subroutine return address.

R15 contains the program counter and is accessible by the programmer.

**Conditonion code flags** in CPSR:

N - Negative or less than flag

Z - Zero flag

C - Carry or bowrrow or extendedflag

V - Overflow flag

The least-significant 8-bit of the CPSR are the control bits of the system. The other bits are reserved[2].

## **Setting Condition Code Flags**

Some instructions, such as Compare, given by CMP R1, R2 which performs the operation R1-R2 have the purpose of setting the condition code flags based on the result of the subtraction operation.

The arithmetic and logic instructions affect the condition code flags only if explicitly specified to do so by a bit in the OP-code field. This is indicated by appending the suffix S to the OP-code. For example, the instruction ADDS R0, R1, R2 sets the condition code flags. But ADD R0, R1, R2 does not.

## **The Encoding Format for Branch Instructions**

Conditional branch instructions contain a signed 24-bit offset that is added to the updated contents of the Program Counter to generate the branch target address. Here is the encoding format for the branch instructions:

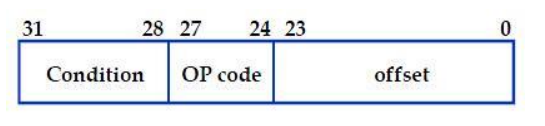


Figure 2: The encoding format for the branch instruction [3]

Offset is a signed 24-bit number. It is shifted left two-bit positions (all branch targets are aligned word addresses), signed extended to 32 bits, and added to the updated PC to generate the branch target address. The updated PC points to the instruction that is two words (8 bytes) forward from the branch instruction.

ARM instructions are conditionally executed depending on a condition specified in the instruction. The instruction is executed only if the current state of the processor condition code flag satisfies the condition specified in bits 31-28 of the instruction. Thus, the instructions whose condition does not meet the processor condition code flag are not executed. One of the conditions is used to indicate that the instruction is always executed.

Here is a more detailed description[4].

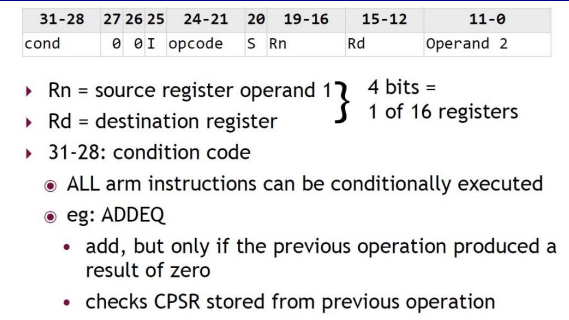


Figure 3: ARM Instruction [5]

All the ARM instructions are conditionally executed depending on a condition specified in the instruction (bits 31-28).

Table 1: Condition in branch instruction

|  |  |  |  |
| --- | --- | --- | --- |
| **CONDITION** |  | **Flags** | **Note** |
| **0000** | EQ | Z==1 | Equal |
| **0001** | NE | Z==0 | Not Equal |
| **0010** | HS/CS | C==1 | >= (u) / C=1 |
| **0011** | LO/CC | C==0 | < (u) / C=1 |
| **0100** | MI | N==1 | minus ( neg ) |
| **0101** | PL | N==0 | plus ( pos ) |
| **0110** | VS | V==1 | V set ( ovlf ) |
| **0111** | VC | V==0 | V clr |
| **1000** | HI | C==1&&Z==0 | > (u) |
| **1001** | LS | C==0&&Z==1 | <= (u) |
| **1010** | GE | N==V | >= |
| **1011** | LT | N!=V | < |
| **1100** | GT | Z==0&&N==V | > |
| **1101** | LE | Z==1&&N!=V | <= |
| **1110** | AL | always |  |
| **1111** | NE | never |  |

(u) = unsigned

The instruction is executed only if the current state of the processor condition code flag satisfies the condition specified in bits 31-28 of the instruction.

The instructions whose condition does not meet the processor condition code flag are not executed.

One of the conditions is used to indicate that the instruction is always executed.

## **Branch and Control Instructions**

Branch instructions are very useful for selection control and looping control.

Here is a list of the ARM processor's Branch and Control instructions.

B loopA ; branch to unconstrainedly label loopA

---------------------------------------------------------------------------------------------------------------------

BEQ target ; When Z = 1, conditionally branch to the target

--------------------------------------------------------------------------------------------------------------------

BNE AAA ; branch to AAA when Z = 0

---------------------------------------------------------------------------------------------------------------------

BMI BBB ; Branch to BBB when N = 1

---------------------------------------------------------------------------------------------------------------------

BPL CCC ;Branch to CCC from BPL when N = 0

---------------------------------------------------------------------------------------------------------------------

BLT labelAA ; Conditionally branch to label labelAA,

; N set, V clear, or N clear, V set

; i.e., N!= V

---------------------------------------------------------------------------------------------------------------------

BLE labelA ; Conditionally branch to label labelA,

; when less than or equal to, N set and V clear

; or N clear and V set

; i.e. Z = 1 or N!= V

---------------------------------------------------------------------------------------------------------------------

BGT labelAA ; Conditionally branch to label labelAA,

; Z clear and either N set and V set

; or N clear and V clear

; that is, Z = 0 and N = V

---------------------------------------------------------------------------------------------------------------------

BGE labelA ; Conditionally branch to label labelA,

; when greater than or equal to zero,

;Z set or N set and V clear; or N clear and V set

; that is, Z = 1 or N!=V

---------------------------------------------------------------------------------------------------------------------

BL funC ;Branch to function funC using a link (Call),

;return address is recorded in LR, register R14

---------------------------------------------------------------------------------------------------------------------

BX LR ;Return from function call

---------------------------------------------------------------------------------------------------------------------

BXNE R0 ;Conditionally branch to the address stored in R0

---------------------------------------------------------------------------------------------------------------------

BLX R0 ; Branch with link and exchange also Call to an address kept in R0.

## **Examples of Compare Instructions**

Mnemonic Meaning

---------------------------------------------------------------------------------------------------------------------

CBZ R5, target ; If R5 is zero, take the forward branch

---------------------------------------------------------------------------------------------------------------------

CBNZ R0, target ; If R0 is not zero, take the forward branch

---------------------------------------------------------------------------------------------------------------------

CMP R2, R9 ; Update the N, Z, C, and V flags, R2 –R9

---------------------------------------------------------------------------------------------------------------------

CMN R0, #6400 ; R0 + #6400, update the N, Z, C, and flags

---------------------------------------------------------------------------------------------------------------------

CMPGT SP, R7, LSL #2 ; Update the N, Z, C, and V flags

---------------------------------------------------------------------------------------------------------------------

# **Procedure**

## **Example 1 Using Branch Instruction**

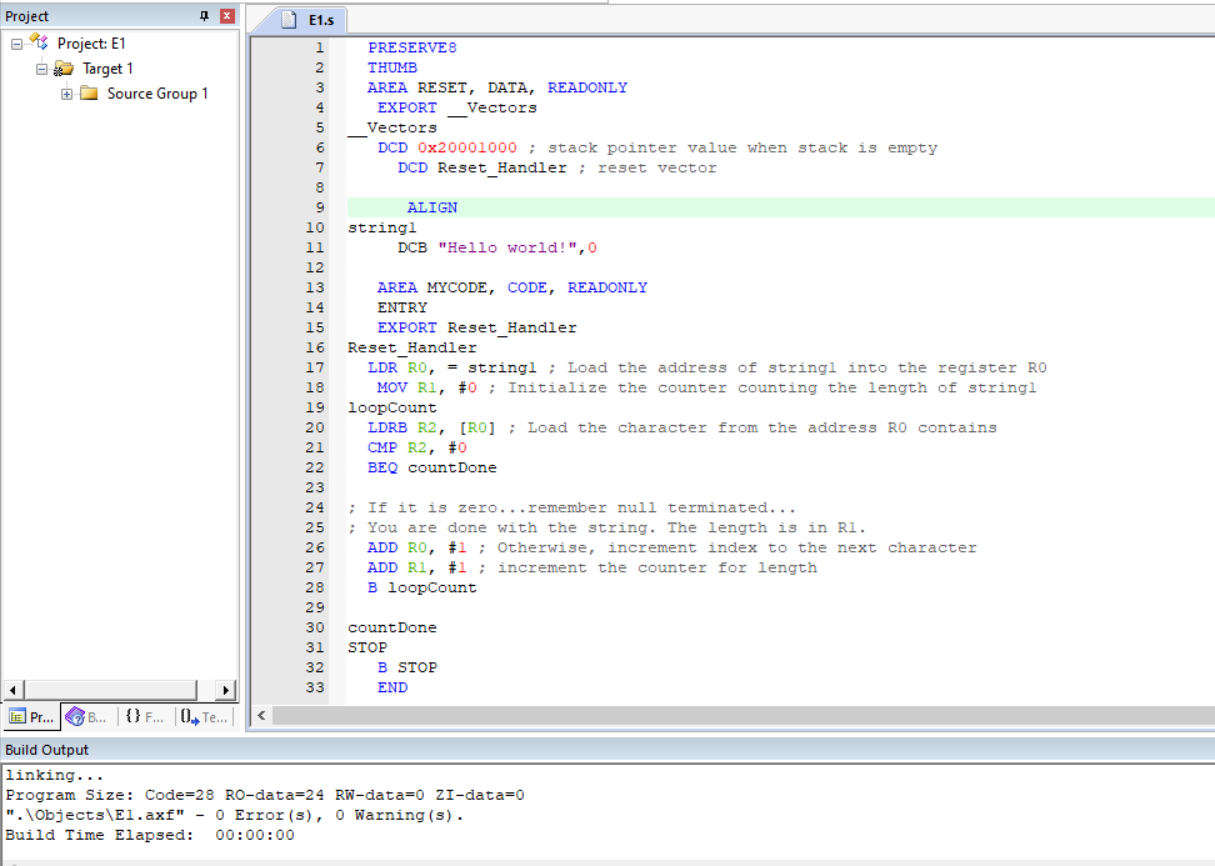


Figure 4: Assembly code 1

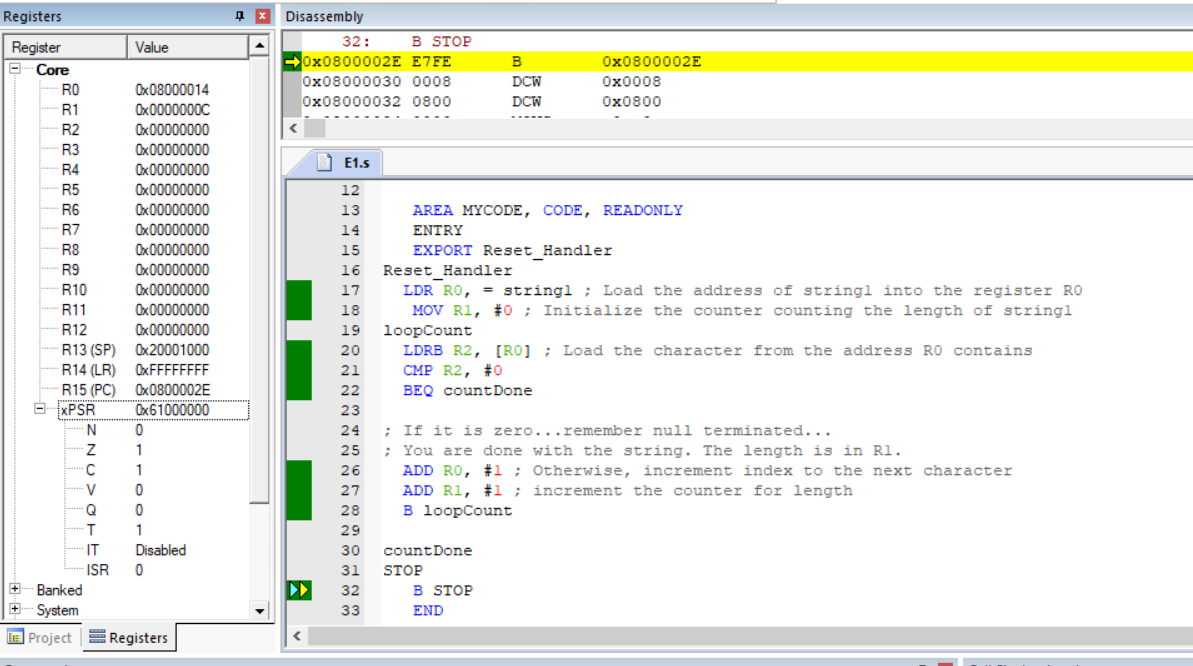


Figure 5: Debugging the first code

This code count the length of a string (“Hello world!”) and stored it into R1,, length of string is 12 characters, which equal C in hexadecimal that’s appear in the right side of screenshot in the register R1.

## **Example 2 Using Branch Instruction**

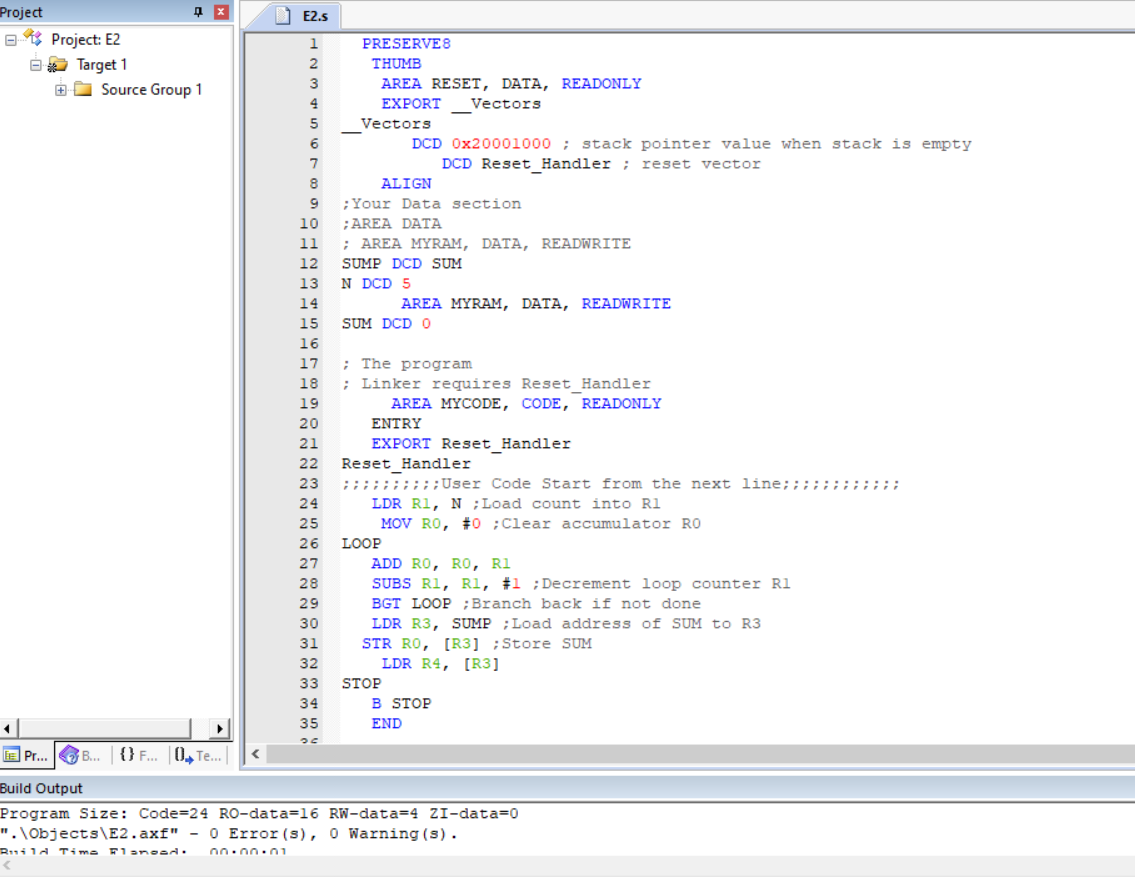


Figure 6: Assembly code 2

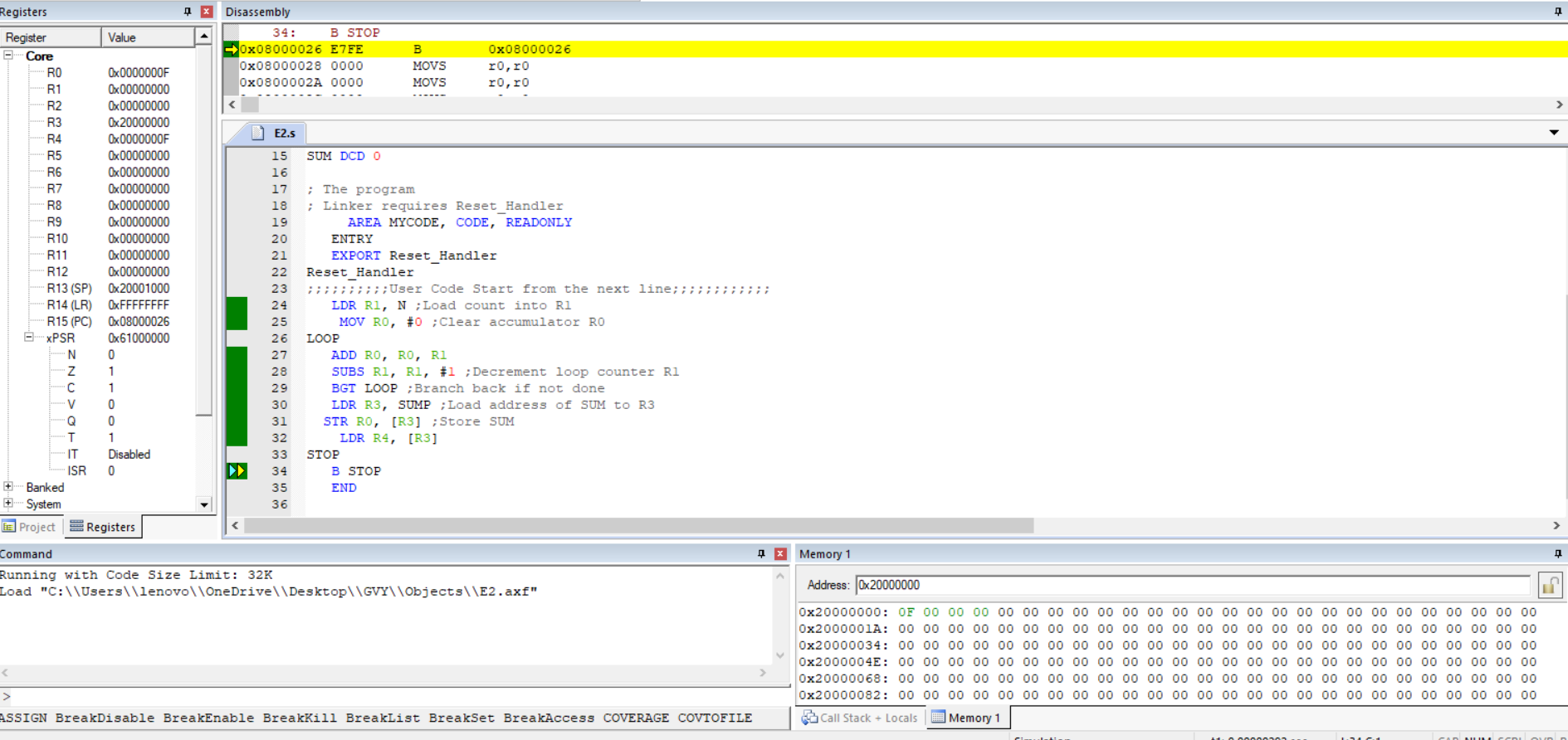
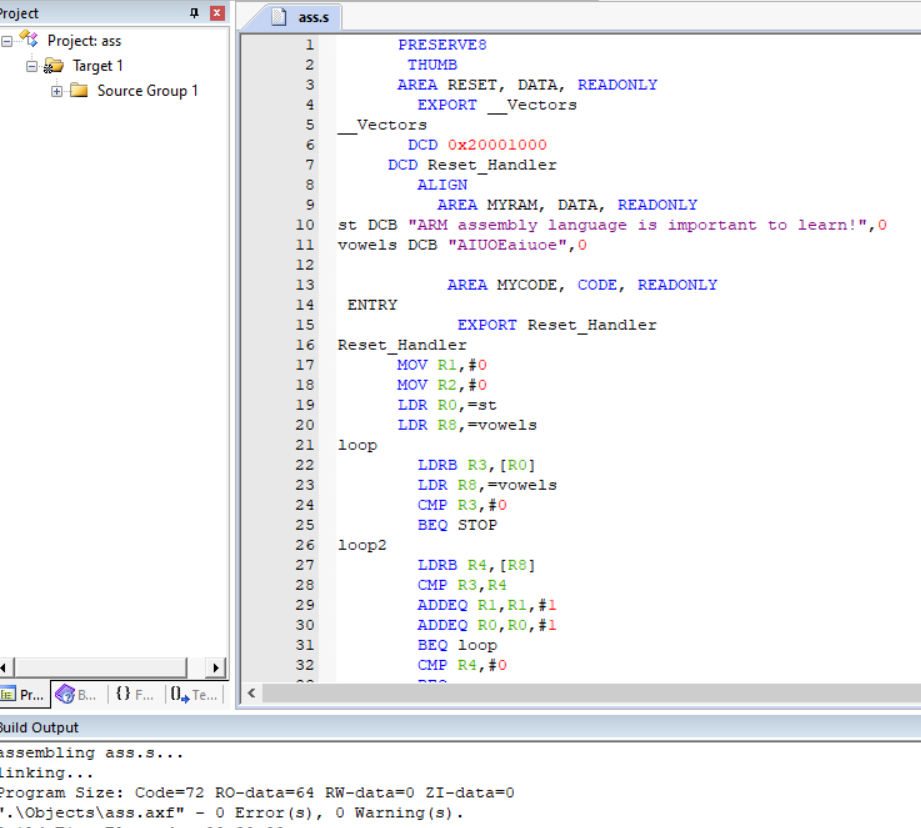


Figure 7: Debugging the second code

This code count the integer between 0 and N ,, but in this code N=5, so it adding the numbers between 0-5 which is 1+2+3+4+5 and the result equal 15 which equal F in hexadecimal and this result stored in register R0 and load into a register R4.

## **Lab Assignment**



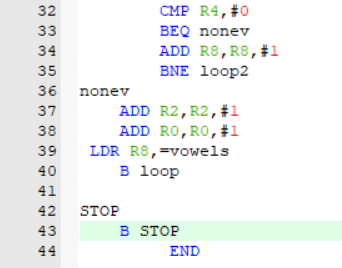


Figure 8: Lab assignment code

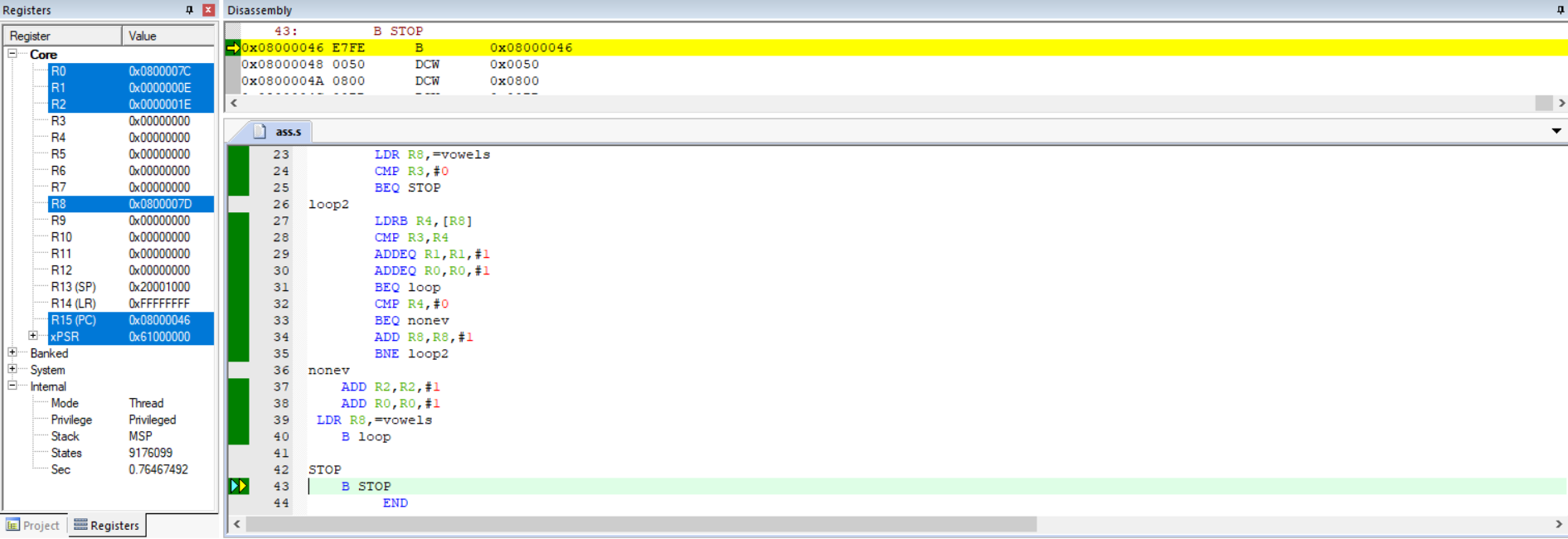


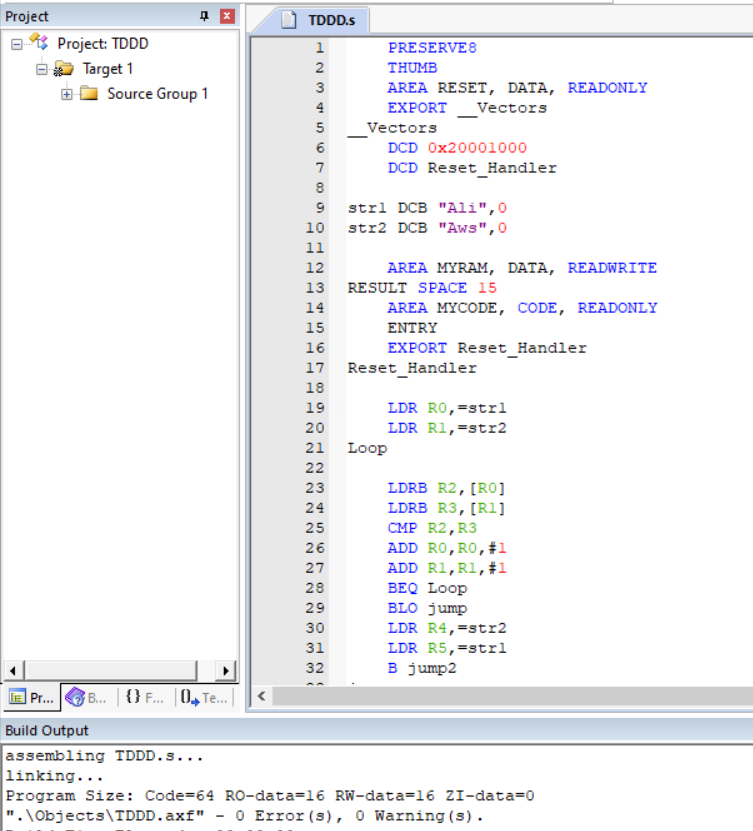
Figure 9: Debugging lab assignment code

After debugging the code the register R1 have a value E in hexadecimal which equal 14 in decimal which equal number of vowels.

And the register R21 have a value 1E in hexadecimal which equal 30 in decimal which equal number of non-vowels.

## **Task**

Given Two strings sort them alphabetically and store them in memory



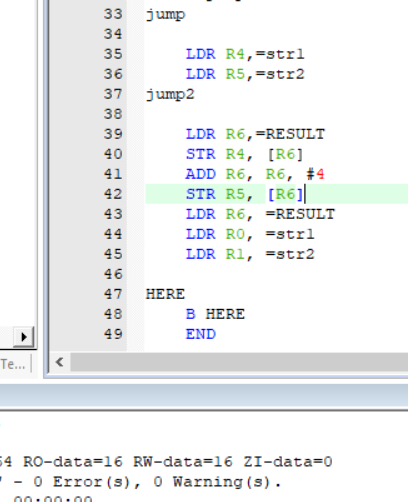
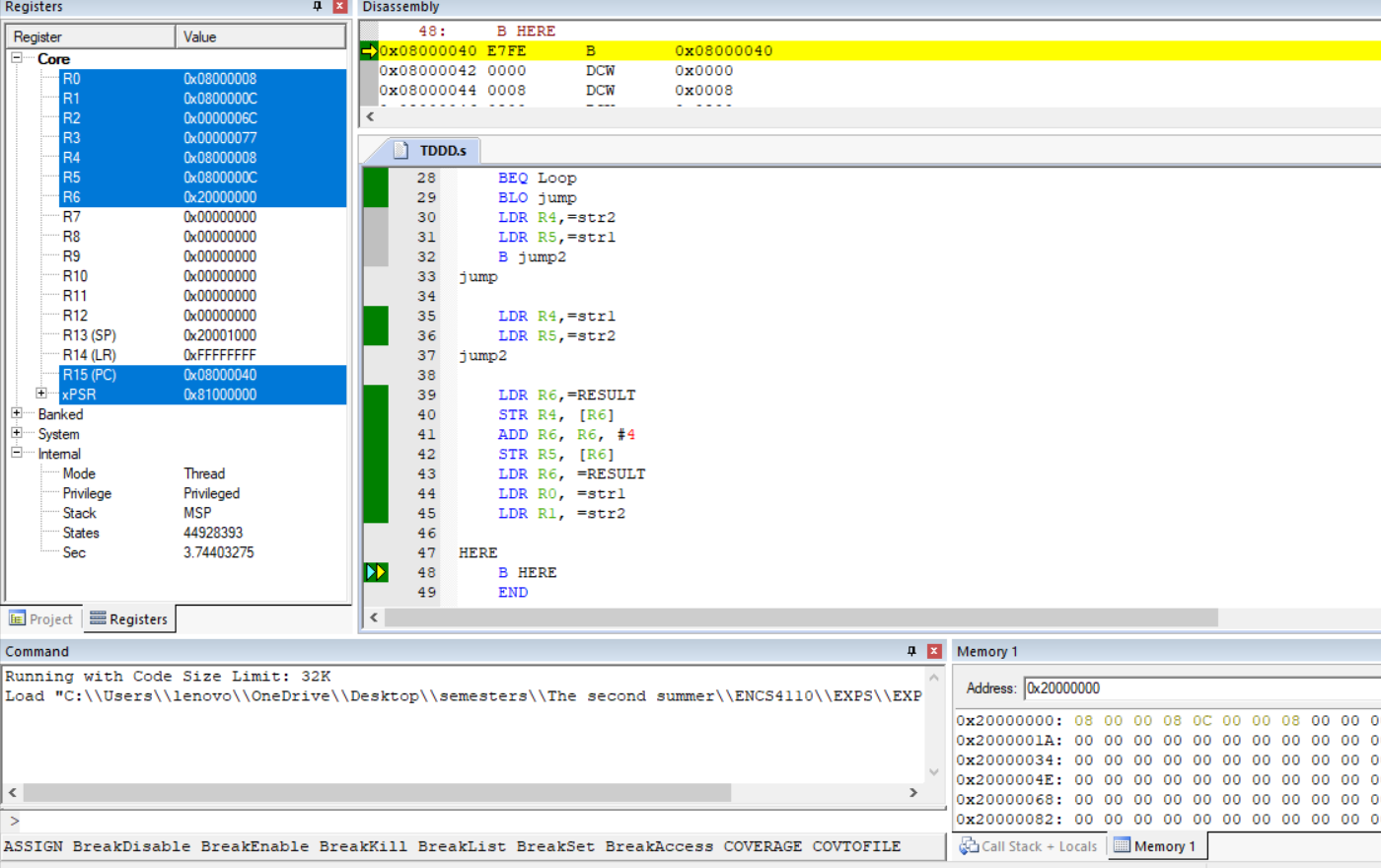
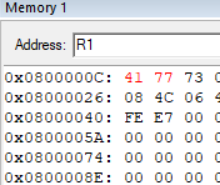


Figure 10: Task code





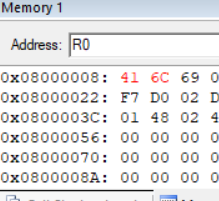


Figure 11: Debugging task code

The result is in the address of register R6, and ( Aws ) and ( Ali ) they are equal in (A) so we compare with the second letter and ( l ) is greater than ( w ) alphabetically, so Ali stored first in memory in the register R0 with address 0x08000008, and Aws stored in register R1 with address 0x0800000C, and when we entered the address of register R0 we found character of Ali appeared :

A = 65 in decimal = 41 hexadecimal

l = 108 in decimal = 6C in hexadecimal

i = 105 in decimal = 69 in hexadecimal

and when I entered the address of register R1 I found character of Aws appeared:

A = 65 in decimal = 41 hexadecimal

w = 119 in decimal = 77 in hexadecimal

s = 115 in decimal = 73 in hexadecimal

# **Conclusion**

In this experiment, I learnt about some new branch instruction and strings and I wrote sum codes on this topic including: count sum of numbers and find the length of string and count number of vowels and non-vowels in a string, also I wrote a task code that sort string alphabetically. After this I can deal with strings and branch now, and I understood the importance of branch instructions and how to use them to make conditional statements and loops.

# **Feedback**

This experiment is a nice one, I learnt some new ideas in Assembly languages and the time of this experiment was perfect and I finished my tasks and work before ending time.

# **References**

*[1] ENCS4110 Lab Manual Page 1*. (2022, 7 29). Retrieved from file:///C:/Users/lenovo/Downloads/Documents/Exp3\_ARM%20Flow%20Control%20Instructions.pdf

*[2] ENCS4110 Lab Manual Page 2*. (2022, 7 29). Retrieved from file:///C:/Users/lenovo/Downloads/Documents/Exp3\_ARM%20Flow%20Control%20Instructions.pdf

*[3] ENCS4110 Lab Manual Page 2*. (2022, 7 29). Retrieved from file:///C:/Users/lenovo/Downloads/Documents/Exp3\_ARM%20Flow%20Control%20Instructions.pdf

*[4] ENCS4110 Lab Manual page 2*. (2022, 7 29). Retrieved from file:///C:/Users/lenovo/Downloads/Documents/Exp3\_ARM%20Flow%20Control%20Instructions.pdf

*[5] ENCS4110 Lab Manual Page 3*. (2022, 7 29). Retrieved from file:///C:/Users/lenovo/Downloads/Documents/Exp3\_ARM%20Flow%20Control%20Instructions.pdf