

**Faculty of Engineering & Technology Electrical and Computer Engineering Department**

**ENCS4110 Laboratory Report #1:**

**Experiment No 3. ARM's Flow Control Instructions**

**Prepared by:**

Mohammad Salem 1200651 **Instructor:** Abualseoud Hanani **Assistant:** Tareq Zidan

**Section:** 1

**Date:** March 20, 2024

# Abstract

The aim of this experiment is to learn more about ARM register sets, branch instructions and how to code with them. Also, to learn how to define and use strings and access the string character by character. We have also explored many ARM data-processing instructions, condition code flags used in the current process status register, and the encoding format for branch instructions. Finally, we practice coding ARM through examples, lab work and a to do task.

# Table of Contents

1. **Abstract** I
2. [**Theory**](https://docs.google.com/document/d/1bJtlnOT_Cf2HDBLa6Q0fTgGsQyMr-ABqseQClZ0zYYQ/edit#heading%3Dh.dmshv0kvip70)
   * [2.1 Review of ARM Register Set 1](#_TOC_250002)
   * [2.2 Current Processor Status Register and Condition Flags 2](#_TOC_250001)
   * [2.3 Branch and Control Instructions and Format 3](#_TOC_250000)
3. [Procedure](https://docs.google.com/document/d/1bJtlnOT_Cf2HDBLa6Q0fTgGsQyMr-ABqseQClZ0zYYQ/edit#heading%3Dh.i5b9td577uci)
   * 3.1 Counting Length of String 4
   * 3.2 Calculating Arithmetic Sum 5
   * 3.3 Lab Work Counting Vowels 6
   * 3.4 Todo – detect the same bit and find their count 8

[Conclusion](https://docs.google.com/document/d/1bJtlnOT_Cf2HDBLa6Q0fTgGsQyMr-ABqseQClZ0zYYQ/edit#heading%3Dh.2ajmbri2k2qt) 10

[References](https://docs.google.com/document/d/1bJtlnOT_Cf2HDBLa6Q0fTgGsQyMr-ABqseQClZ0zYYQ/edit#heading%3Dh.rb1nryd0zcyc) 11

# Table of Figures

* Figure 1:ARM register Set 1
* Figure 2: Encoding Format for Branch Instruction 3
* Figure 3: Conditions for Branch Instructions 3
* Figure 4: Finding Length of String Code 4
* Figure 5: Finding Sum of Arithmetic Series Code 5
* Figure 6: Counting Vowels and Non-Vowel Code 6
* Figure 7: Counting Vowels and Non-Vowel Continued Code 7
* Figure 8: ToDo Code 8
* Figure 9: ToDo case 2 9

# Theory

## Review of ARM Register Set

The assembly language used in this laboratory is the ARM instruction set which is a RISC (Reduced Instruction Set Computer) machine, executing a single instruction per cycle, making it simple and efficient especially in the implementation of circuits. It is designed to be both energy and power efficient. Thus, it is used in many devices such as smartphones, tablets, and wearables[1].

ARM has a total of 16 different registers and a Current Program Status Register CPSR, all of length 32 bits or one word. Registers from R0 to R12 are called general purpose registers since they are used for common purposes. Meanwhile, registers R13 to R15 are special purpose registers since they have a specific objective. R13 is the stack pointer, which has the address of the top of the stack and uses the commands “push” and “pop” to access the stack. R14 is the link register, which is when returning from a function. Finally, register R15 is the program counter that points at the next instruction. The figure below summarizes these registers [2].

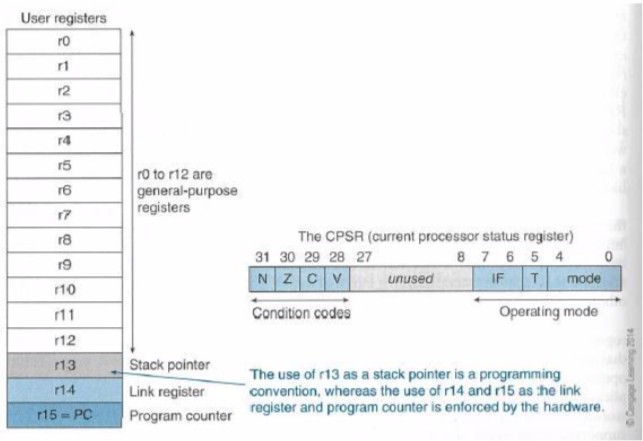


Figure 1: ARM register [2]

## Current Processor Status Register and Condition Flags

As can be seen in figure 1 the CPSR (Current Processor Status Register) is a register of length 32 bits that has many sections. The three sections are condition codes, unused sections and an operating mode section. The condition code keeps track of the different flags, each given one bit. The “N” stands for the negative or less than flag. The “Z” stands for the zero flag. The “C” stands for the carry or borrow or extended flag. The “V” stands for the overflow flag [2].

These flags are used with arithmetic and logical instructions by adding the letter “s” at the end of the OP-code field. For Example, the instruction ADDS R0, R1, R2 will set the condition flag but this ADD R0, R1, R2 instruction does not. However, there are instructions that don't need the suffix “s” to be added to its op-code field, but still automatically affect the condition flags. These instructions are not arithmetic or logical instructions such as the CMP, TST and TEQ instructions [2].

## Branch and Control Instructions and Format

Figure 2 shows the encoding format for the branch instruction. The format consists of three sections: the condition, OP code field, and the offset. The Branch instruction allows the program to move completely to different instructions either above or below the current instruction. Thus, the offset is a signed 24-bit offset. Since branch targets are aligned word addresses they need to be shifted left by 2 bit positions and signed extended to 32 bits. After that the PC will be modified by adding the branch 32 bit induction with the PC to generate the intended address to move to [2].

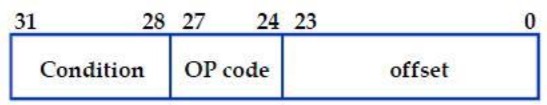


Figure 2: Encoding Format for Branch Instruction [2]

In the figure above, the condition field determines if an instruction will be executed depending if it meets the processor condition code flag. The following figure shows instructions that are executed depending on a certain condition.

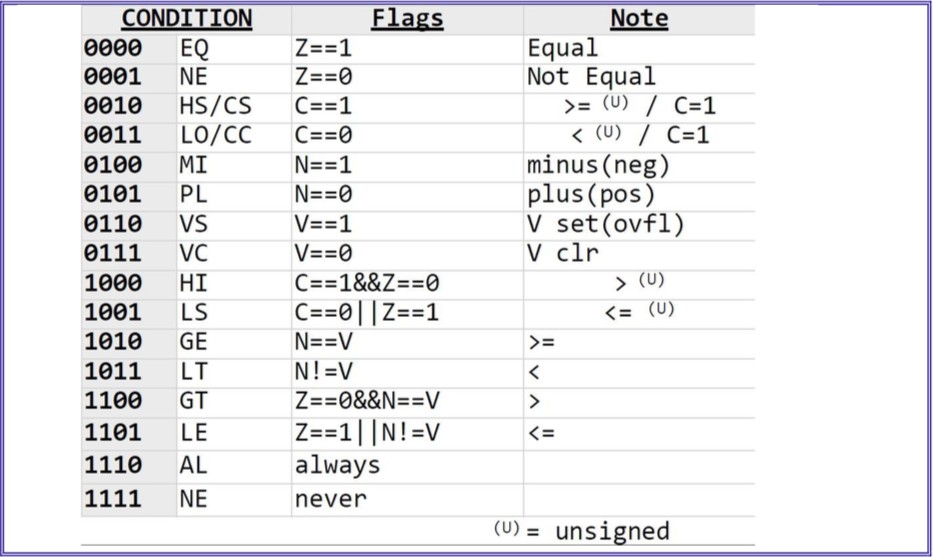


Figure 3: Conditions for Branch Instructions[2]

# Procedure

## Counting Length of String

The following figure is an ARM program that counts the length of the String including the spaces as well. First, a string is defined using DCB (define constant byte) and its value is “Hello world!”, as seen in line 24. Then, the address of the string is loaded in Register R0 where it points to the first byte of the string. Register R1 will hold the value of the length, thus it is initialized to zero at first. Then the code enters a loop to go through each character by loading it into register R2 and comparing that character with the null pointer. If the character is not null then the counter is incremented by one, as well as, the address position register R0 to move to the next character. Once the null character is reached the code breaks out of the loop and branches to the label “countDone” and ends the program. The final result is stored in register R1. In the below case “Hello world!” has 12 characters which is equivalent to 0xC in hexadecimal.

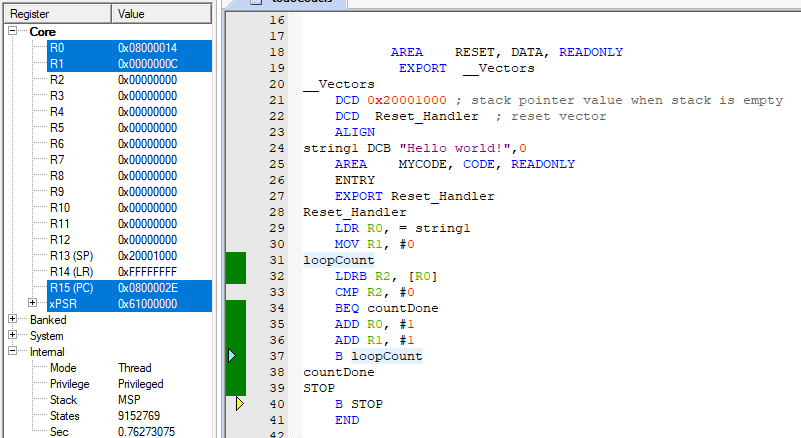


Figure 4: Finding Length of String Code

## Calculating Arithmetic Sum

The following figure is an ARM program that calculates the sum of all the numbers from N to 1. For example, in the figure below N is equal to 5, thus the sum from 5 to 1 is 5+4+3+2+1 which is equal to 15. The program starts by loading the number in register R1 and initializing register R0 which will hold the result at the end. The code enters a loop to add the number to the sum and then decrementing the value of register R1 by one. This loop continues until register R1 is 0. Then the result in R0 is stored in the memory at the address held in register R3. The result is then loaded from the memory to register R4. As can be seen in the register set, R4 has the value of 0xF in hexadecimal which is equivalent to 15 in decimal.

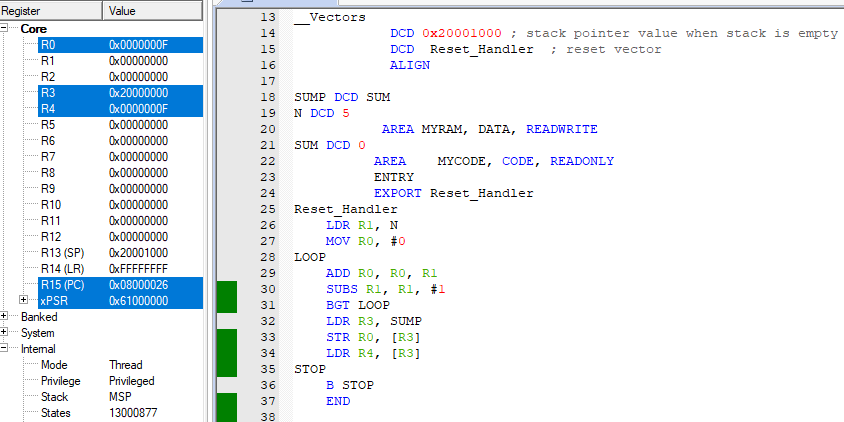


Figure 5: Finding Sum of Arithmetic Series Code

## Lab Work Counting Vowels

The ARM program below counts the number of vowels and of non-vowels in a given string. First, the String is defined in line 13 using DCB(define constant byte). Then the address of the string is loaded in R0, and R1 and R2 are initialized to hold the results of the number of vowels and non-vowels respectively. The loop starts by loading the first character and compares it with the null character to check if the end of the string is reached. It also compares it with a space character as seen in line 53. If the character is a space character then the code will branch to “skip” that will just move on to the next character. Then the code continues with a number of checks and branches that compares the character with all the vowels. If the character is a vowel, then the code branches to the label “vowel” that will increment register R1 as seen in figure 7. If a certain character is an explanation mark or a question mark then it will also branch to the label “skip”. If the letter is none of the above then automatically it is a non-vowel character so the code will branch to label “NonVoule” which will increment the value of register R2 as seen in figure 7.

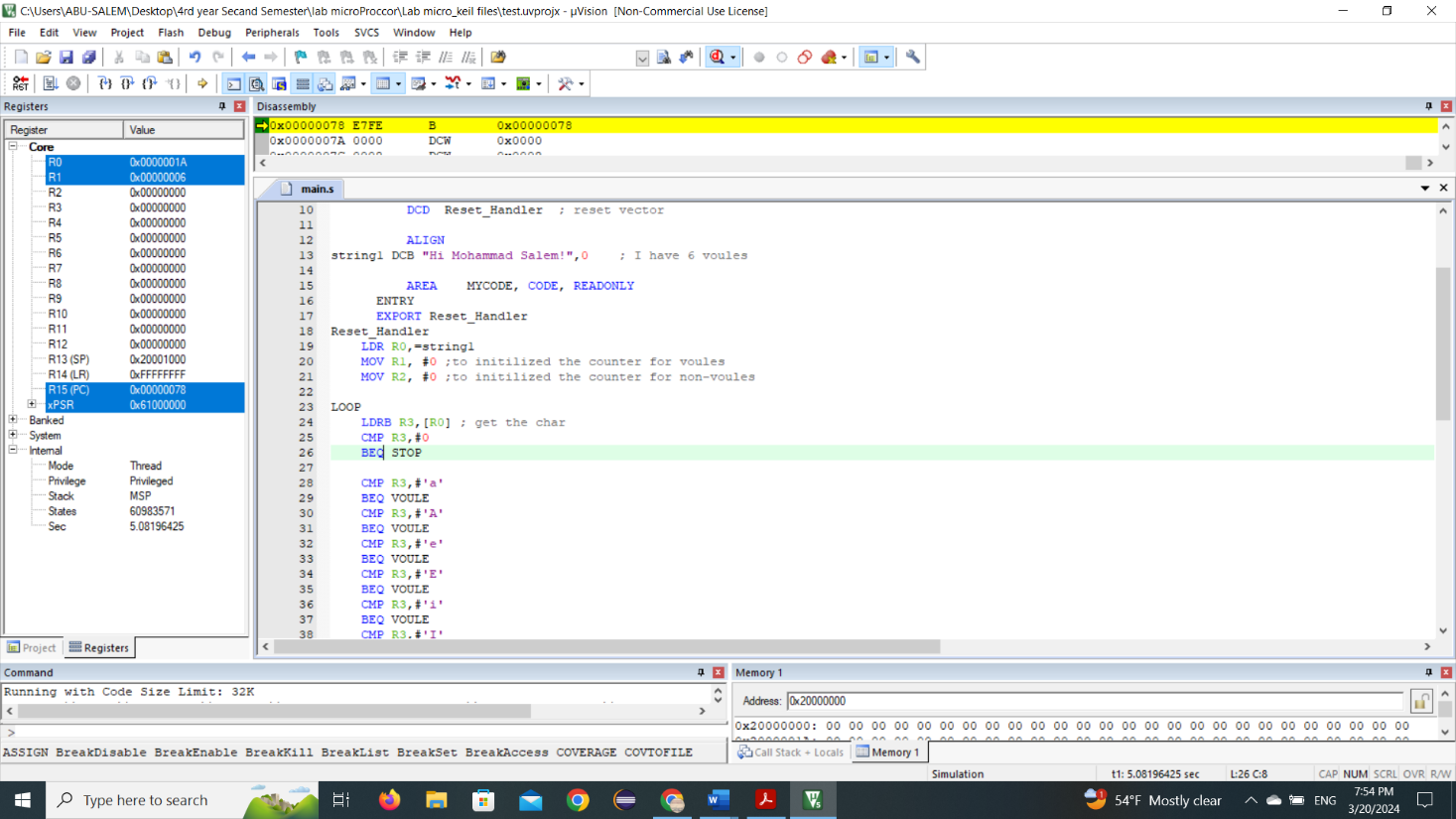


Figure 6: Counting Vowels and Non-Vowel Code

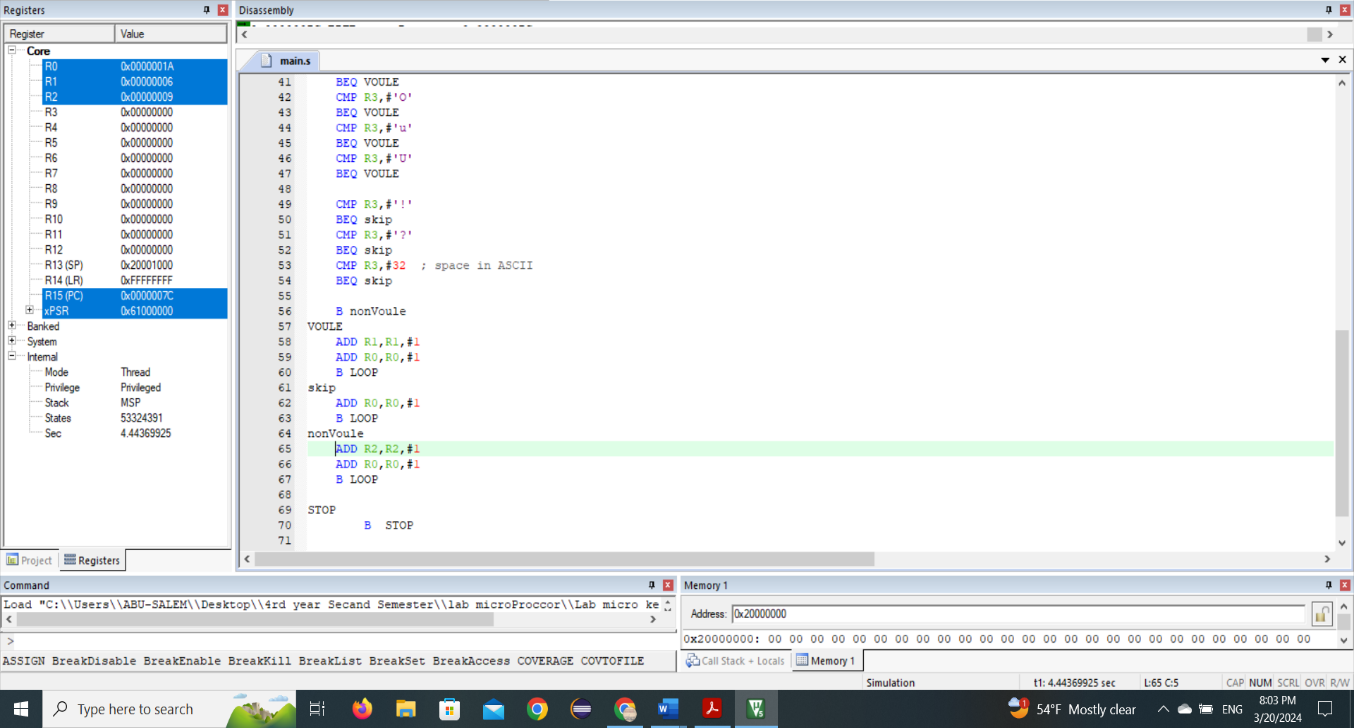


Figure 7: Counting Vowels and Non-Vowel Continued Code

It is important to note that I have assumed that the string will only contain letters and special characters such as the space, explanation mark and question mark. In my code, the string is given the value “Hi Mohammad Salem!” which contains 6 vowels ‘i’, ‘o’, ‘a’, ‘a’, ‘a’, ‘e’ Also, the other are not vowel. As seen in the register file, R1 has the value 6 for the number of vowels and R2 has the number 9 for the number of non-vowels.

**Todo - detect the same bit and find their count.**

In the following code as shown below at Figure 8. First, initialization of R0 and R1 of the same value to delete all matches it is 32 bit, at R5 loop counter to count decreasing from 32 until R5 reach at 0 it is express the number of iteration from up to down by Descending. Also, I and my partner did not find ENOR, we Mack EOR "xor" as shown in line 35, then we think about the Right shift but the RRXS version to effected on flags.

If we were to interpret "RRXS R2, R2" in a hypothetical scenario, it might suggest rotating the contents of register R2 to the right, extending the rightmost bit into the carry flag, and then swapping the result back into register R2 itself.

The result is if Carry flag equal zero the bits is match

Example: let R0=0011 R1=0101

then R0 xor R1 = 0110 the R0 xnor R1 = 1001 by RRXS the bit in C flag indicate if match occur or not , in line 39 BCC it is match, and increment the R3 number of matches by 1 , and so on so for , in the update count block of code change the value of counter at R5, and if R5 reach at 0 that means the 32bit are completely checked. I know what you think the not match case? it is at figure 9 the C flag always equal 1.

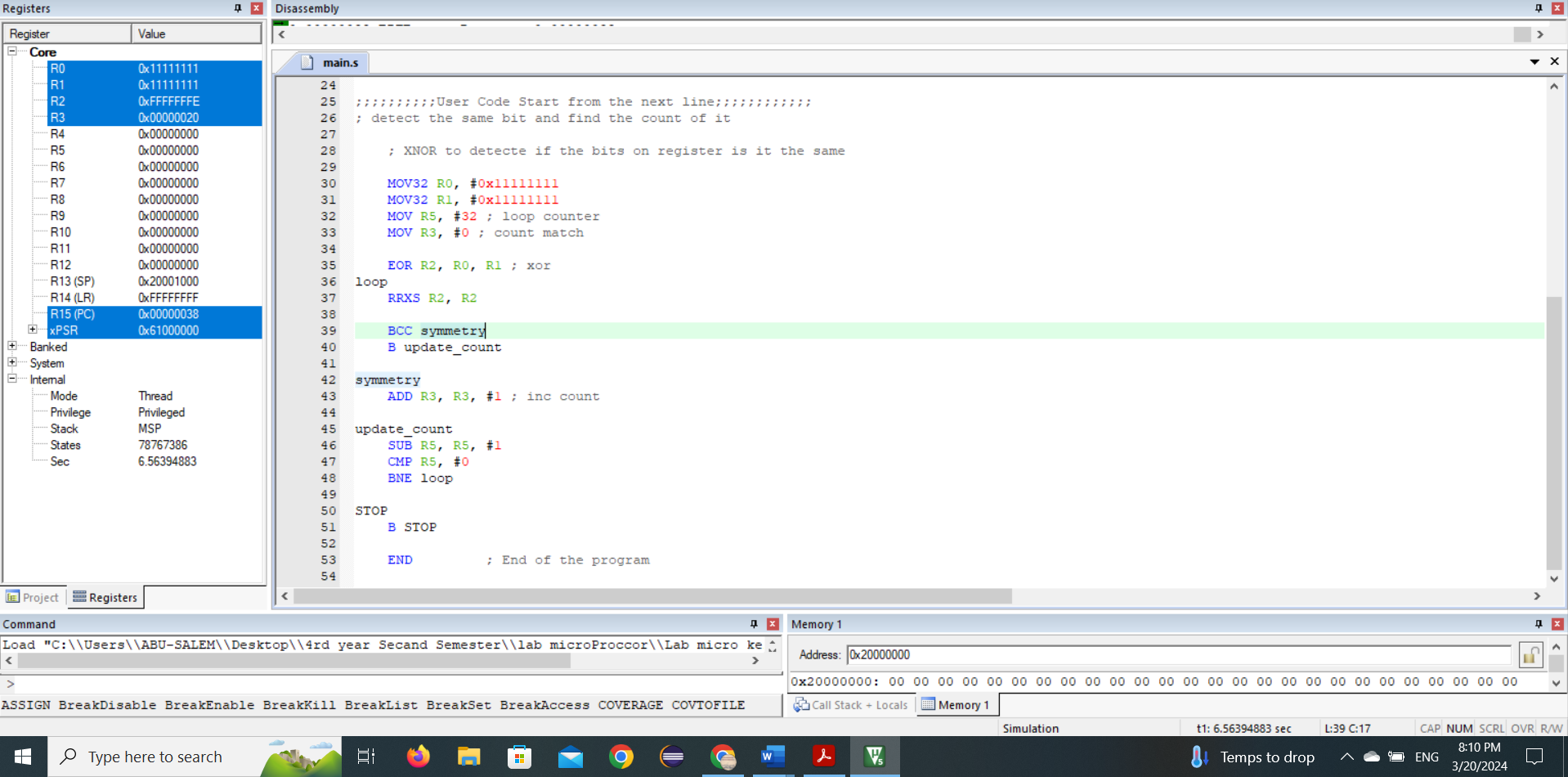


Figure 8: ToDo Code

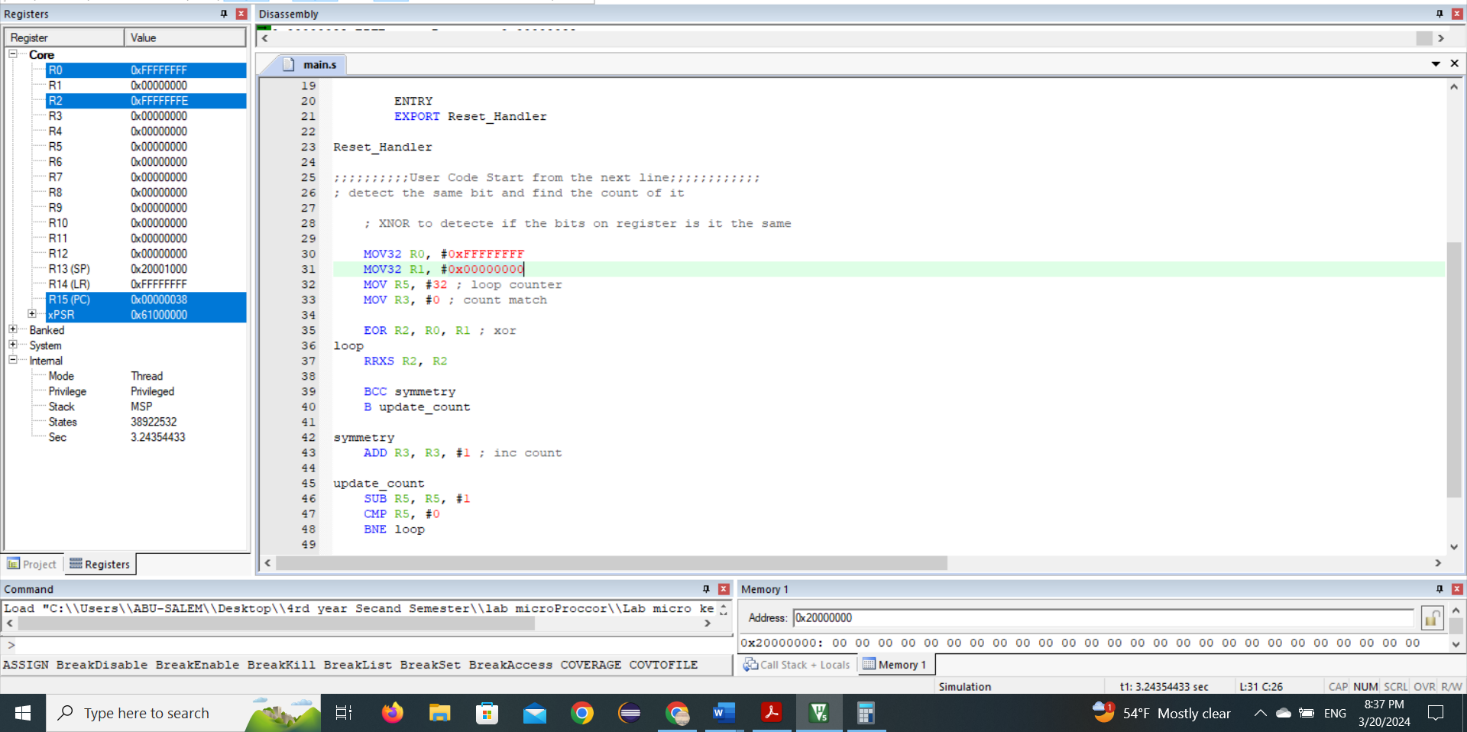


Figure 9: ToDo case 2

# Conclusion

In this lab, we have learned about many different instructions including arithmetic, logical, branching and control instructions. We have also seen conditional flags and used instructions for the purpose of changing these flags, such as the RRXS,BCC and CMP instructions that were used in the todo task. We have also learned how to deal with strings and load characters one by one with the program that counts the length of a string. Moreover, we used arithmetic instructions and branch instruction in the program that calculates the sum series.

# References

1. :<https://www.geeksforgeeks.org/arm-processor-and-its-features/>
2. : Lab-Manual Experiment 3