

**Electrical and Computer Engineering**

**Computer Design Lab – ENCS4110**

**ARM's Flow Control Instructions**

Student Name: Maisam Alaa

Student ID: 1200650

Instructor: Ayman Hroub

Section: 4

Date: 24\11\2022

# **Abstract:**

The aim of this experiment is to explore ARM branch, execute instructions and

Instruction Format and implement them in Keil uVision5, and to investigate how to use

strings in Keil uVision5.

# **Contents**

**Theory1**

ARM Registers Set1

Condition code suffixes and related flags2

Branch and Control instructions4

Examples on ARM using Branch and flags5

**Procedure** **6**

Part1: Exercise on Keil uVision56

Part2: Lab Work10

Part3: TODO13

**Conclusion** **16**

**References** **17**

## **Figures:**

[Figure 1 ARM Registers Set 2](#_Toc120208319)

[Figure 2 Branch and Exchange instructions 4](#_Toc120208320)

[Figure 3 Examples in ARM using branch 5](file:///C:\Users\maisam\Desktop\report%201\report.docx#_Toc120208321)

[Figure 4 Exercise 1 on Keil uVision 5 6](#_Toc120208322)

[Figure 5 Registers set with the output shown 7](#_Toc120208323)

[Figure 6 Exercise 2 on Keil uVision 5 8](#_Toc120208324)

[Figure 7 Registers set with the output shown 9](#_Toc120208325)

[Figure 8 Lab Work code part 1 10](#_Toc120208326)

[Figure 9 Lab Work code part 2 11](#_Toc120208327)

[Figure 10 Lab Work output 12](#_Toc120208328)

[Figure 11 TODO code part 1 13](#_Toc120208329)

[Figure 12 TODO code part 2 14](#_Toc120208330)

[Figure 13 TODO output 15](#_Toc120208331)

## **Tables:**

[Table 1 Condition code suffixes and related flags 3](#_Toc120208354)

[Table 2 Conditional Branch instructions 5](#_Toc120208355)

# **Theory:**

## **ARM Registers Set [1]**

ARM processors provide general-purpose and special-purpose registers. Some additional registers are available in privileged execution modes.

In all ARM processors, the following registers are available and accessible in any processor mode.

* 13 general-purpose registers R0-R12.
* One Stack Pointer (SP).
* One Link Register (LR).
* One Program Counter (PC).
* One Application Program Status Register (APSR).

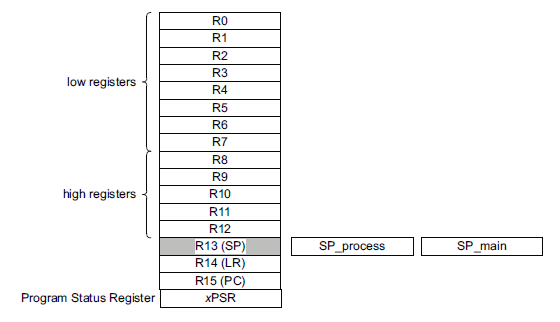


Figure 1 ARM Registers Set

## **Condition code suffixes and related flags [1]**

The condition flags are held in the APSR. They are set or cleared as follows:

**N**

Set to 1 when the result of the operation is negative, cleared to 0 otherwise.

**Z**

Set to 1 when the result of the operation is zero, cleared to 0 otherwise.

**C**

Set to 1 when the operation results in a carry, or when a subtraction results in no borrow, cleared to 0 otherwise.

**V**

Set to 1 when the operation causes overflow, cleared to 0 otherwise.

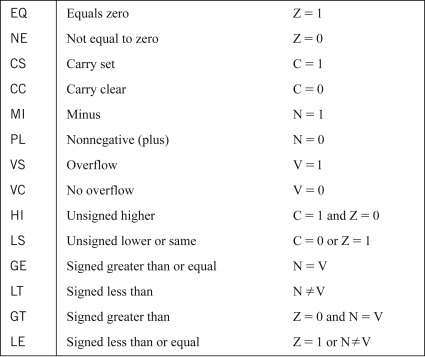
Note

Most instructions update the condition flags only is the S suffix is specified. The instructions CMP, CMN, TEQ, and TST always update the flags.

Condition code suffixes define the conditions that must be met for the instruction to execute.

The following table shows the condition codes that you can use and the flag settings they depend on.

Table 1 Condition code suffixes and related flags



## **Branch and Control instructions [4]**

This instruction is only executed if the condition is true. This instruction performs a branch by copying the contents of a general register, Rn, into the program counter, PC. The branch causes a pipeline flush and refill from the address specified by Rn. This instruction also permits the instruction set to be exchanged. When the instruction is executed, the value of Rn[0] determines whether the instruction stream will be decoded as ARM or THUMB instructions.

0

0 0

0

0

1

0

1

0

0

0

Cond

1

Rn

0

3

4

7

8

11

12

15

16

19

20

23

24

27

28

31

**Operand register**

If bit 0 of Rn = 1, subsequent instructions decoded as THUMB instructions

If bit 0 of Rn = 0, subsequent instructions decoded as ARM instructions

**Condition Field**

1

1

1

1

1

1

1

1

1

1

1

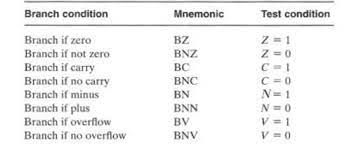
1

Figure 2 Branch and Exchange 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.[3]

Table 2 Conditional Branch instructions



## **Examples on ARM using Branch and flags [2]**

MOV R0, #0x90

WHILE\_START

TST R0, #0x80

TSTNE R0, #0x01

BNE WHILE\_DONE

SUB R0, R0, #1

B WHILE\_START

WHILE\_DONE

MOV R0, #0

MOV R1, #10

START\_FOR

CMP R1, #0

BLE END\_FOR

ADD R0, R0, R1

SUB R1, R1, #1

B START\_FOR

END\_FOR

Figure 3 Examples in ARM using branch

# **Procedure:**

## **Part1: Exercise on Keil uVision5**

This part goal is to practice on the program, and solve some example using Branch Instructions.

This program object is to count the length of a string. We first load the string into R0 then created a loop to count and check to end.

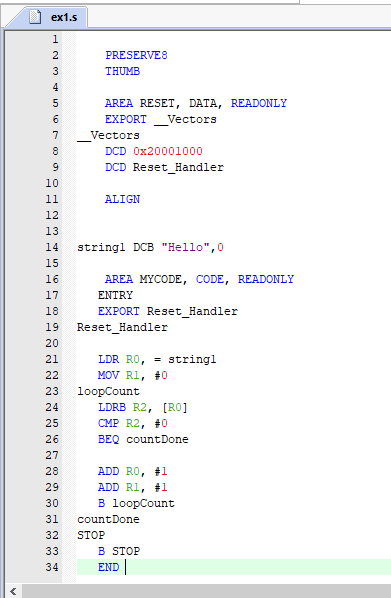


Figure 4 Exercise 1 on Keil uVision 5

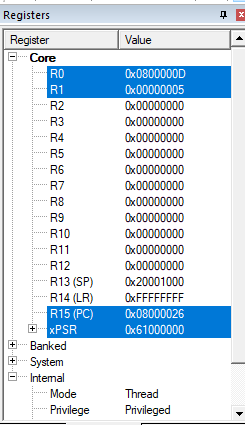


Figure 5 Registers set with the output shown

Note that the result is stored in R1, and its shown 0x5 as in “HELLO”.

This program object is to find N+(N-1)+(N-2)+…….+1. We first load the number to R1 and created a loop to add and check until reaching the number 0 to stop program. The result is loaded in R4 after storing it in R0.

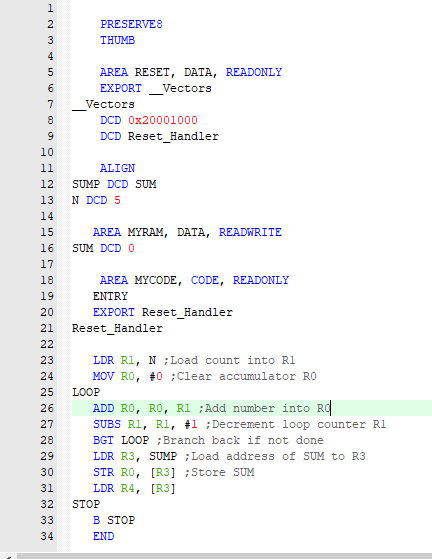


Figure 6 Exercise 2 on Keil uVision 5

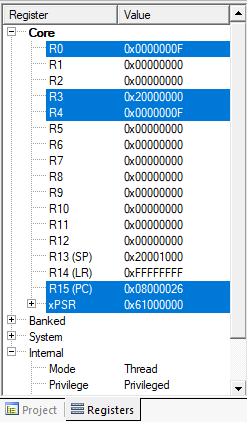


Figure 7 Registers set with the output shown

- Note that the result is shown in R4 and R0.

- 5+4+3+2+1= 15 in decimal = 0xF in Hexa.

## **Part2: Lab Work**

This part object is to write an ARM program CountVowelsOne.s to count how many vowels and how many non-vowels are in the following string.

"ARM assembly language is important to learn!",0

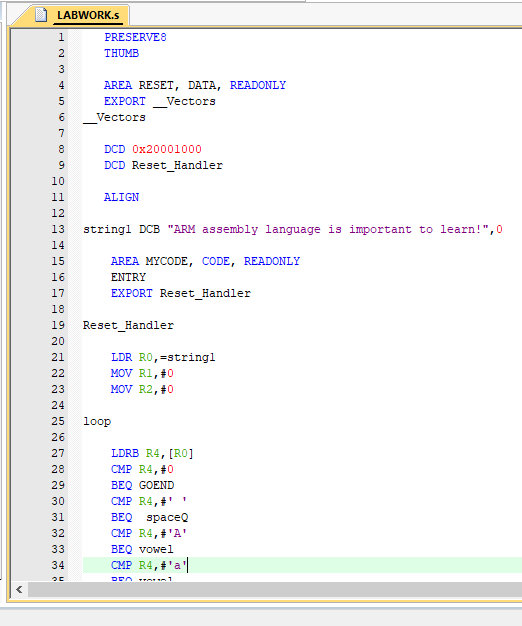


Figure 8 Lab Work code part 1

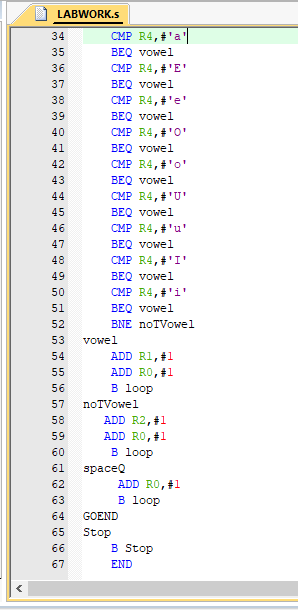


Figure 9 Lab Work code part 2

Description: we first loaded the string to R0, then created a loop to track the string length and compare every litter to see if its vowel or non-vowel.

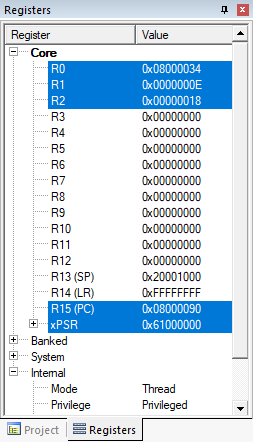


Figure 10 Lab Work output

Note: The result of the vowel count is stored in R1 which is 0xE in hexa( 14 in decimal ), and the result of the non-vowel count is stored in R2 which is 0x18 in hexa( 24 in decimal ).

## **Part3: TODO**

This part object is to write an ARM program to find the Factorial on an integer .

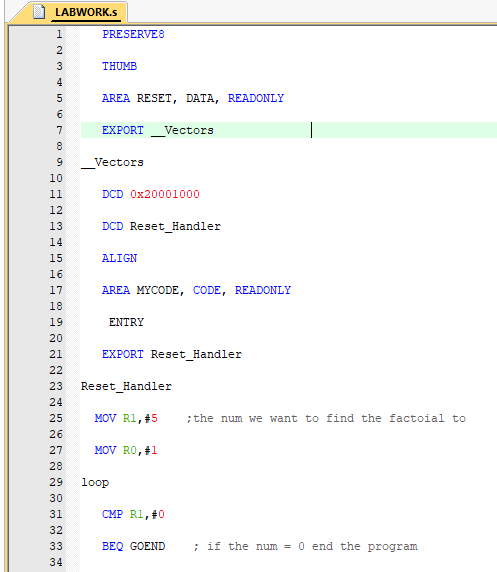


Figure 11 TODO code part 1

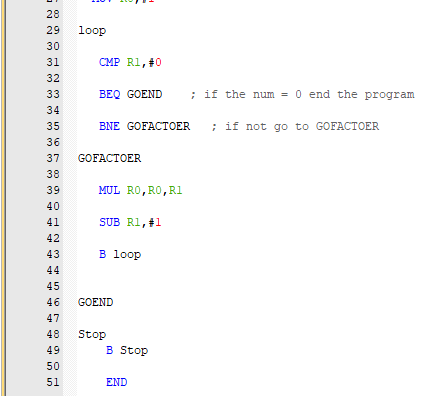


Figure 12 TODO code part 2

Description: we first selected a number to find its factorial, then created a loop to multiply it with (n-1)\*(n-2)\*…….\*1.

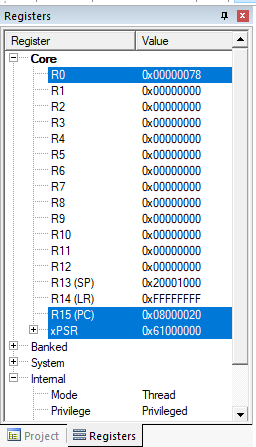


Figure 13 TODO output

-Note that the result shown in register R0 is 0x78 in hexa which is 120 in decimal.

- 5! = 5\*4\*3\*2\*1 = 120 .

# **Conclusion:**

In the end of this experiment we explored ARM branch instructions and learned more about strings and how to deal with them in Keil uVision5, we also impart more knowledge about loops and how branch makes them easier and understand the importance of condition flags, and use all of those to create ARM assembly programs.

# **References:**

[1]: <https://developer.arm.com/documentation/dui0473/m/overview-of-the-arm-architecture/arm-registers>

[2]: <https://ece353.engr.wisc.edu/arm-assembly/branches/>

[3]: <https://web.uettaxila.edu.pk/CMS/AUT2012/ectCARbs/notes%5CLecture-8%20%28Computer%20Architecture%29%203rd%20Semester%202k11.pdf>

[4]: <https://iitd-plos.github.io/col718/ref/arm-instructionset.pdf>