Faculty of Engineering and Technology
Electrical and Computer Engineering Department

Computer Design Laboratory
ENCS4110

Report of Experiment No. 3

ARM's Flow Control Instructions

Student's Name: Maha Mali

Student's ID: 1200746

Sec: 1

Instructor: Dr. Abualseoud Hanani

Teacher assistant: Eng. Raha Zabadi

Date: 29/7/2022

Abstract

The aim of this experiment is to explore and understand instructions and implement them in Keil uVision5. Also, to investigate how to use strings in Keil uVision5.

Contents

List Of Figures

1.Theory

1.1 ARM Register Set

ARM has 16 programmer-visible registers and a Current Program Status Register, CPSR. The registers are divided into two parts: General-purpose and Special-purpose registers are provided by ARM CPUs. The following registers are present and accessible in any processor mode on all ARM processors:

- 13 general-purpose registers(R0-R12).
- 3 Special-purpose registers(R13-R15).
- stack pointer (SP)--->(R13).
- Link Register (LR)--->(R14).
- program counter (PC)--->(R15)

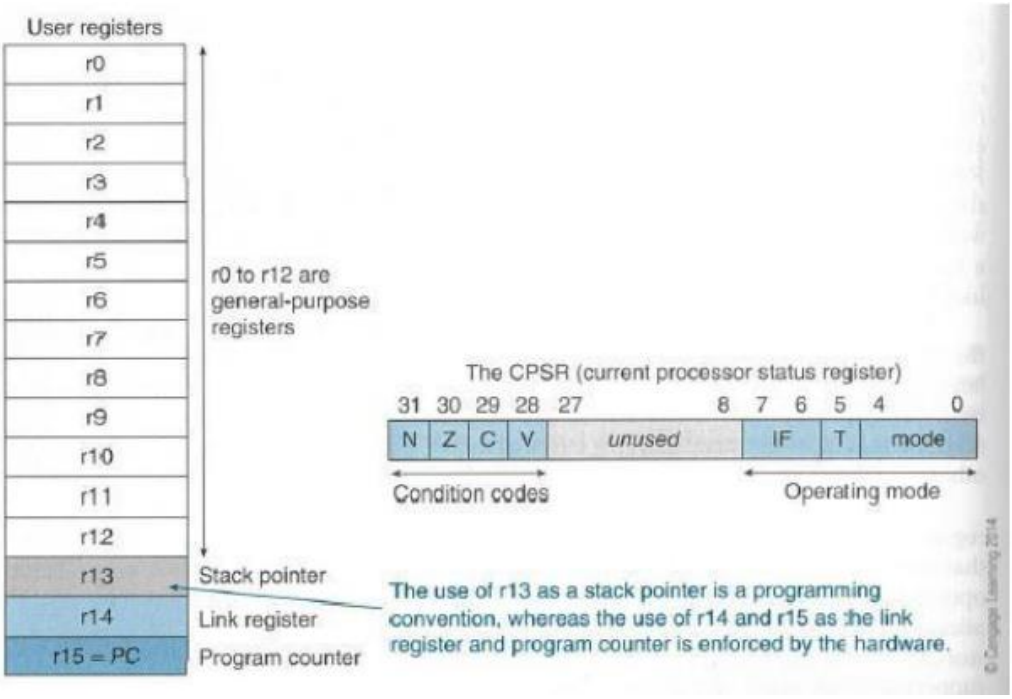

Figure1: ARM Registers

1.2. Setting Condition Code Flags

Some instructions, such as Compare, given by CMP R1, R2 which performs the operation R1-R2 have the sole purpose of setting the condition code flags based on the result of the subtraction operation (the instruction Compare just update the flags the result of subtraction does not store).

The arithmetic and logic instructions affect the condition code flags only 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 SUBS R0, R1, R2 sets the condition code flags. But SUB R0, R1, R2 does not.

1.3. The Encoding Format for Branch Instruction

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. Figure show the encoding format for the branch instructions.

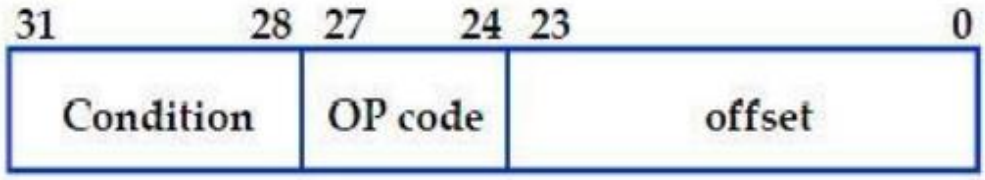

Figure2:Encoding format

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 b31-b28 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 3 that the instruction is always executed.

The figure 3 shows more detailed description.

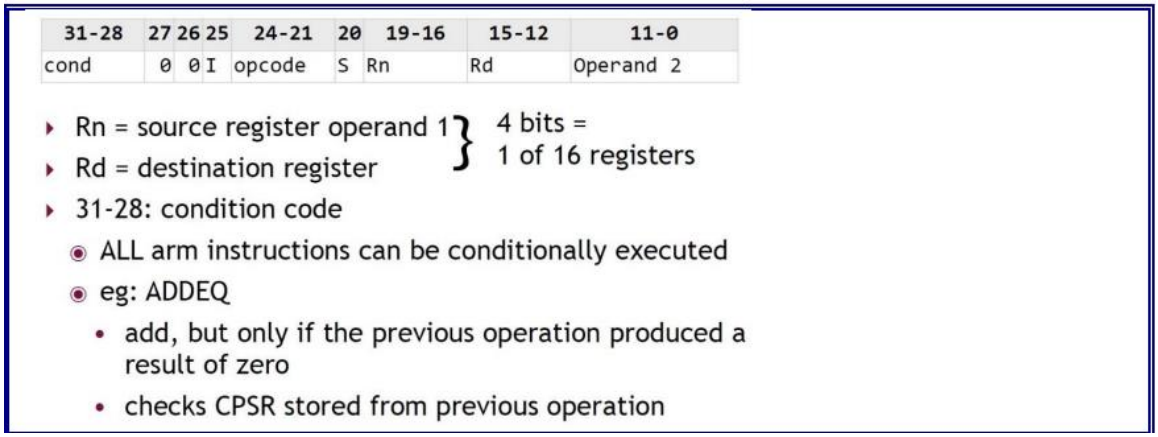

Figure3: Detailed Branch instruction

All the ARM instructions are conditionally executed depending on a condition specified in the instruction (bits 31-28). The figure 4 shows the condition in ARM 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(ovfl)
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

Figure4: Conditions in branch instructions

- The instruction is executed only if the current state of the processor condition code flag satisfies the condition specified in bits b31-b28 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

1.4. Branch and Control Instructions

Branch instructions are very useful for selection control and looping control. There are many Branch instructions as shown in figure 5. Such as:

- B loop A (Branch to label loop A unconditionally).
- BEQ target (Branch if equal: Conditionally branch to target, when Z = 1).
- BNE AAA (Branch if not equal: Branch to AAA when Z = 0).
- BX LR (Branch and exchange instruction set: Return from function call).

```
-----------------------------------------------------------------------
     B loopA         ; Branch to label loopA unconditioally
-----------------------------------------------------------------------
     BEQ target      ; Conditionally branch to target, when Z = 1
-----------------------------------------------------------------------

     BNE AAA         ; branch to AAA when Z = 0
-----------------------------------------------------------------------
     BMI BBB         ; branch to BBB when N = 1
-----------------------------------------------------------------------
     BPL  CCC        ; branch to CCC when N = 0
-----------------------------------------------------------------------
     BLT  labelAA    ; Conditionally branch to label labelAA,
                     ; N set and V clear  or  N clear and V set
                     ; i.e.  N != V
-----------------------------------------------------------------------
     BLE labelA      ; Conditionally branch to label labelA,
                     ; when less than or equal, Z set or 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
                     ; i.e.   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
                     ; i.e.  Z = 1 or N !=V
-----------------------------------------------------------------------
```

```
BL funC         ; Branch with link (Call) to function funC,
                ; return address stored in LR, the register R14
BX LR           ; Return from function call
BXNE R0         ; Conditionally branch to address stored in R0
BLX R0          ; Branch with link and exchange (Call)
                ; to a address stored in R0.
```

Figure5: Branch and Control Instructions

1.5. Examples Compare Instructions

Compare Instructions, given by CMP R1, R2 which performs the operation R1-R2 have the sole purpose of setting the condition code flags based on the result of the subtraction operation (the result of subtraction does not store just update the flag).

There are many instructions for compare as shown in figure 6. Such as:

- CBZ R5, target (Compare and Branch on Zero: Forward branch if R5 is zero).
- CBNZ R0, target (Compare and Branch on Non-Zero: Forward branch if R0 is not zero).
- CMP R2, R9 (Compare: R2 - R9, update the N, Z, C and V flags).

```
Mnemonic                Meaning
CBZ R5, target          ; Forward branch if R5 is zero
CBNZ R0, target         ; Forward branch if R0 is not zero
CMP R2, R9              ; R2 - R9, update the N, Z, C and V flags
CMN R0, #6400           ; R0 + #6400, update the N, Z, C and V flags
CMPGT SP, R7, LSL #2    ; update the N, Z, C and V flags
```

Figure6: Compare Instructions

2.Procedure and Discussion

2.1. Example 1[1]

The aim of this program is to count the length of a string1. In the data section, we defined the string1 which is: "Hello world!",0. The ,0 at the end null terminates the character string. Also, the zero value of the null allows you to tell when the string ends.

First, we have loaded the address of string1 into register R0 by using LDR. And we have initialized the counter R1 counting the length of string1. Then, we start loading the character from the address R0 contains to R2 by using LDR, and check the last character if it equals 0 or not equal zero by using CMP. If it does not equal zero then increment the counter for length. Else if it equals “zero” that means we have reached the end of the string, and we have finished the program.

The count the length of a string1"Hello world!" is 12 in Decimal which is 0x0000000C Hexa-Decimal. And the register R1 prove that as shown in figure 7.

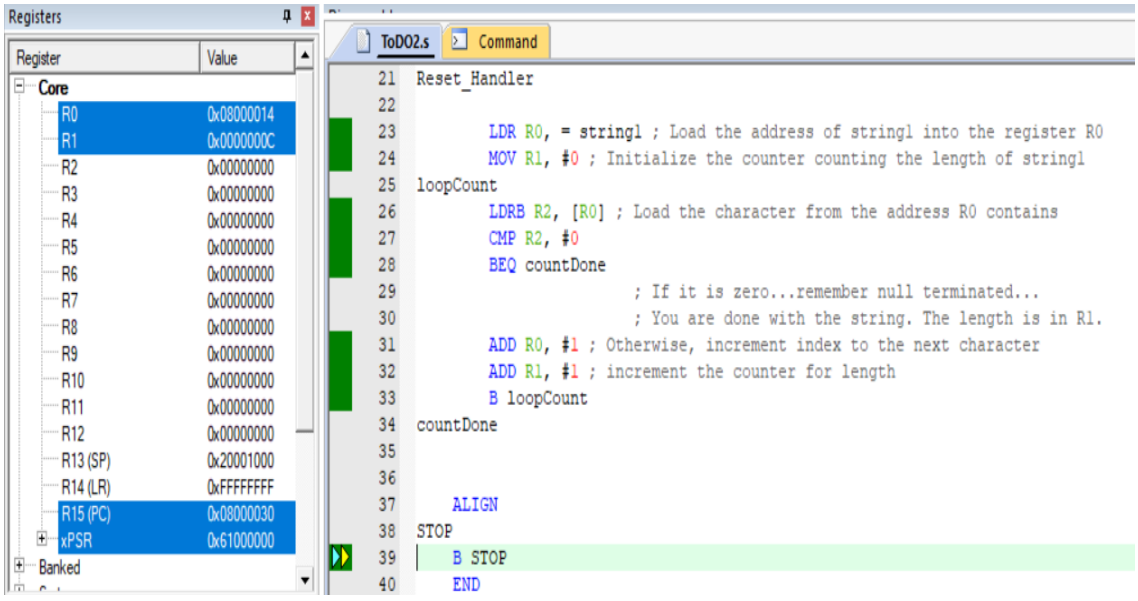

Figure7: Example 1 Simulation

[1] The written code with comments can be found in the Manual.

2.2. Example2[2]

The program is to find the sum of the number from 5 to 1(5+4+2+3+1=15), the result of sum in Decimal is 15, and in the Hexa-Decimal is 0x0000000F. And the register R0 prove that as shown in figure8.

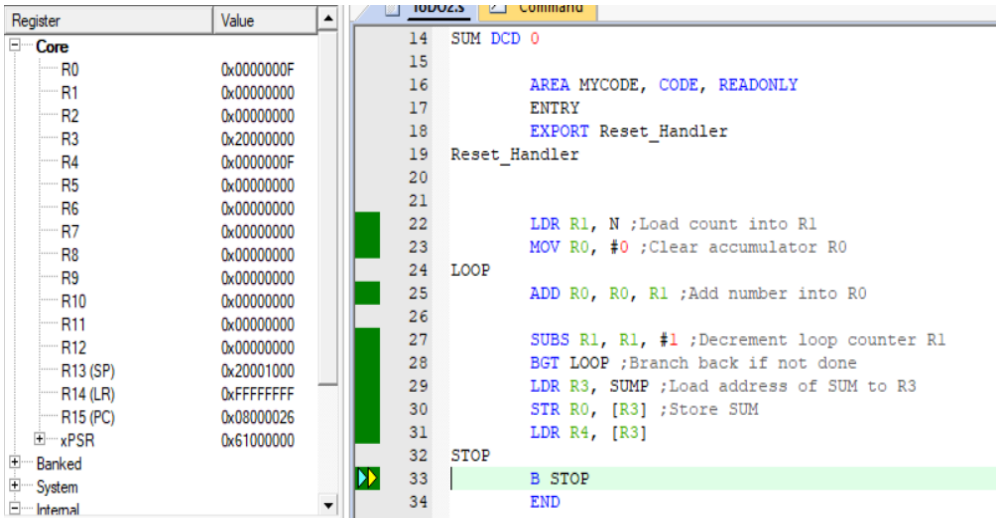

Figure8: Example 2 Simulation

First, we have loaded the count into R1 by using (LDR R1, N which mean: R1=5 because N DCD 5). And we have cleared the accumulator R0 with 0 (R0=0). Then, we start add the values from R1 add them to the accumulator R0 and store the result in R0(R0=R0+R1). Then, decrement the loop counter R1 by 1 and store the result in R1(SUBS R1, R1, #1: R1=R1-1). After decrementing the counter R1 value, we check if it counts greater than 0 or not. If the value of the count did not reach "zero" then continue in the loop this done by using BGT (Branch if greater than). Else if it equals "zero" that means we have added all number between 5 to 1. After finishing the loop, the answer will be stored in the memory by using LDR R0, [R3]; R0 = answer, [R4] = memory address of the sum.

[2] The written code with comments can be found in the Manual.

2.3. Lab Assignment [3]

Required: Write an ARM assembly language program to count how many vowels and how many non-vowels are in the following string: "ARM assembly language is important to learn!"

Vowels = (a e i o u AEIOU). The rest of the letters are not vowels.

Number of vowels in the following string: "ARM assembly language is important to learn!" is 14 in decimal which is 0x0000001E Hexa-Decimal. And the number of non-vowels is 30 in decimal which is 0x0000000E in Hexa-Decimal. And the register R1 and R2 prove that as shown in figure8.

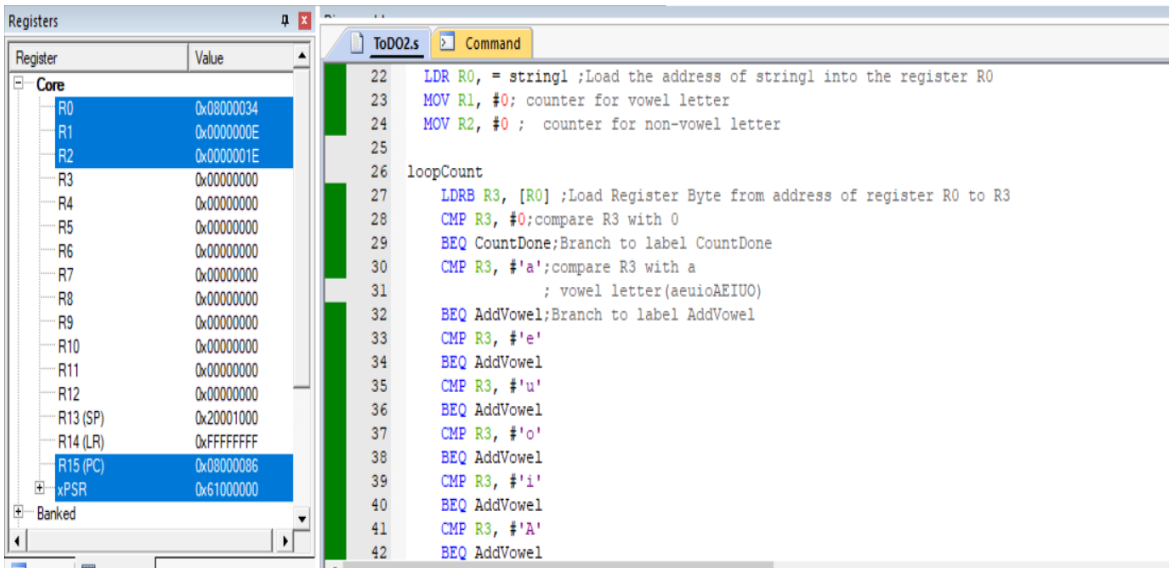

Figure9: Lab Assignment Simulation

In this exercise, we have defined String1 contain "ARM assembly language is important to learn!",0. The ,0 at the end null terminates the character string. Also, the zero value of the null allows you to tell when the string ends.

After Loading the address of string1 into the register R0, we define register R1 which is counter for vowels and register R2 for non- vowels.in addition to that we loading

[3] The written code with comments can be found in the appendix.

the register byte from address of register R0 to R3. This is done by LDRB (Load Register Byte from address of register R0 to R3).

we start checking each letter of the string if it’s vowel or non. This is done by using CMP with the lowercase and uppercase vowels (Example: CMP R3, #'a’) if the register R3 contain lowercase or uppercase vowels Branch to label AddVowel and increment the counter of vowel letter by 1. Also, if R3 does not contain lowercase or uppercase vowels branch to label AddNonVowel and increment the counter of non-vowel letter by 1.

2.4. Task[4]

Required: Given two string sort them alphabetically and store them in memory.

In this exercise, we defined two strings: String1 contain “aseel“and String2 contain maher. So “aseel” should store in the first in memory than “maher” because it came first alphabetically. The ASCII code for “aseel” is (61,73,65,65,6C). And ascii code for “maher” is (6D,61,68,65,72). And the figure 10 show that .

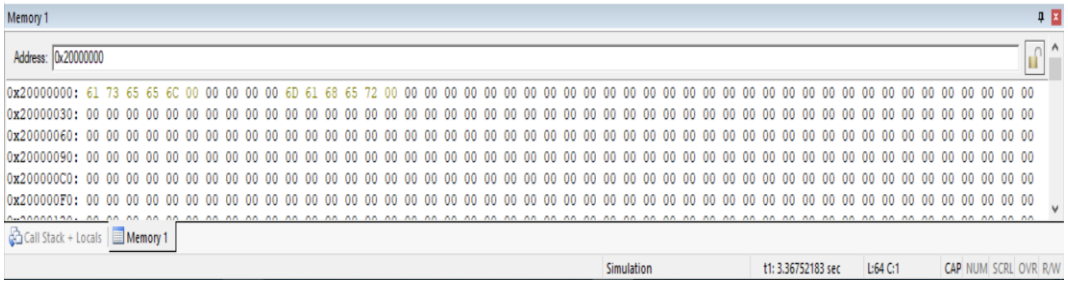

Figure10:Task store the result

After that we loaded the address of memory space First, Second to R10, R9 respectively also the address of string1, string 2 into the registers R0, R1 respectively. Also, we initialized accumulator R5 with initial value 0. In the loopCounter we loaded Register Byte from memory address R0+ R5 to R2(which mean get the character in the address R0+R3) and this is done by Using LDRB (LDRB R3, [R0+R2]). Also loaded Register Byte from memory address R1+ R5 to R3. Then increment the accumulator R5 by 1.

In addition to that compare string1 which is in register R2 and compare string2 which is in R3. Branch to label STORE_FIRST if R3 less than and Branch to label STORE_SECOND if R2 greater than R3. Check if string1 end or not this done by CMP R2, #0 if string1 not end branch to loopCounter and if string1 finish end the program. And the same for string2.

In label STORE_FIRST set accumulator to 0, we loaded Register Byte from memory address R0+ R5 to R2 than store the character in R10, increment the accumulator

[4] The written code with comments can be found in the appendix.

by 1 ,compare if string1 finish or not. And the same for label STORE_SECOND but the result store in R9.

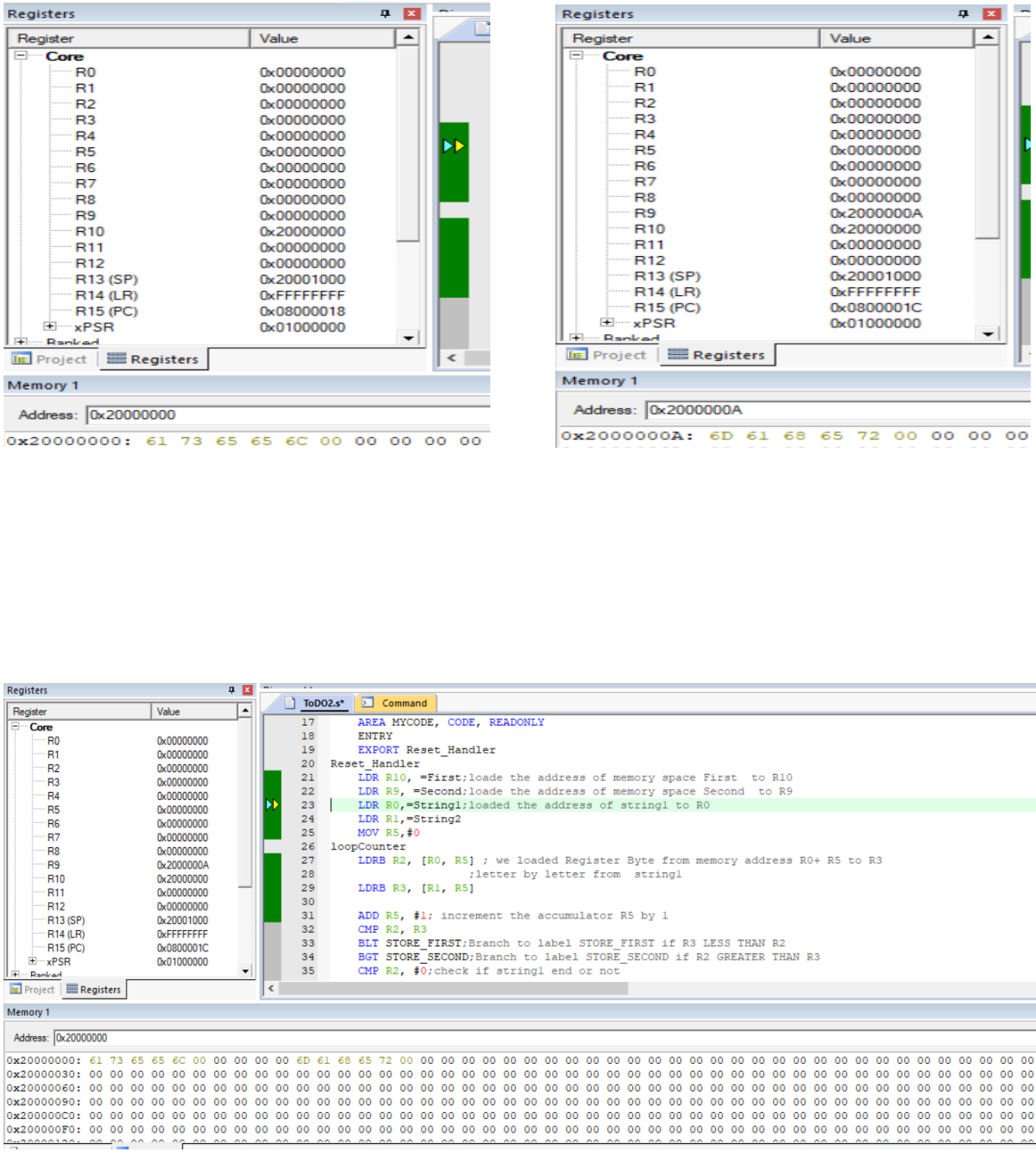

Figure11:Task simulation

3. Conclusion

In this experiment, we learned about different ARM Branch instruction such as: branch if equal (BEQ when Z=1), branch if greater than (BGT when Z=0, N=V) …etc. In addition to that learned about different Compare instruction such as: CMP R2, R9 which mean (R2-R9 update the N, Z, C and V flags, but the result of subtraction does not store just update the flags) …. etc. Then we write many codes to apply what we learned.

4. References

[1] https://www.codesexplorer.com/2017/06/arm-code-length-of-carriage-return-terminated-string.html. Accessed on 26-07-2022 at 1:23PM.

[2] https://iitd-plos.github.io/col718/ref/arm-instructionset.pdf . Accessed on 26-07-2022 at 2:14PM.

[3] https://cseweb.ucsd.edu/classes/su14/cse30-b/lectures/PI_CSE30_lecture_6.pdf. Accessed on 26-07-2022 at 5:34PM.

[4] https://developer.arm.com/ . Accessed on 26-07-2022 at 7:12PM.

5. Appendix

5.1. Example 1 code

```
; This program will count the length of a string.
;Name:Maha Maher Mali
;ID:1200746
        PRESERVE8
        THUMB
    AREA RESET, DATA, READONLY
    EXPORT __Vectors
__Vectors
  DCD 0x20001000
    DCD Reset_Handler
    ALIGN
string1
        DCB "Hello world!",0
    AREA MYCODE, CODE, READONLY
  ENTRY
    EXPORT Reset_Handler
Reset_Handler

        LDR R0, = string1; Load the address of string1 into the register R0
        MOV R1, #0; Initialize the counter counting the length of string1
loopCount
        LDRB R2, [R0]; Load the character from the address R0 contains
        CMP R2, #0
        BEQ countDone
                        ; If it is zero...remember null terminated...
```

```
                                        ; You are done with the string. The length is in  R1
                ADD R0, #1 ; Otherwise, increment index to the next character
                ADD R1, #1 ; increment the counter for length
                B loopCount
countDone
        ALIGN
STOP
        B STOP
        END
```

5.2. Example 2 code

```
;Name:Maha Maher Mali

;ID:1200746

;this program to find the sum of number between 5 to 1

        PRESERVE8
                THUMB
                AREA RESET, DATA, READONLY
                EXPORT __Vectors
__Vectors
        DCD 0x20001000 ; stack pointer value when stack is empty
                DCD Reset_Handler ; reset vector


                ALIGN


SUMP DCD SUM
N DCD 5
                AREA MYRAM, DATA, READWRITE
SUM DCD 0


                AREA MYCODE, CODE, READONLY
                ENTRY
                EXPORT Reset_Handler
Reset_Handler



                LDR R1, N ;Load count into R1
                MOV R0, #0 ;Clear accumulator R0
LOOP
```

```
        ADD R0, R0, R1 ;Add number into R0

        SUBS R1, R1, #1 ;Decrement loop counter R1
        BGT LOOP ;Branch back if not done
        LDR R3, SUMP ;Load address of SUM to R3
        STR R0, [R3] ;Store SUM
        LDR R4, [R3]
STOP
        B STOP
        END
```

5.3. Lab Assignment code

```
;count number of vowel letter and number of non vowel letter
;Name:Maha Maher Mali
;ID:1200746
                PRESERVE8
                THUMB


        AREA RESET, DATA, READONLY
    EXPORT __Vectors
__Vectors
    DCD 0x20001000 ; stack pointer value when stack is empty
    DCD Reset_Handler ; reset vector
    ALIGN
string1
    DCB "ARM assembly language is important to learn!",0


    AREA MYCODE, CODE, READONLY
    ENTRY
    EXPORT Reset_Handler
Reset_Handler


  LDR R0, = string1 ;Load the address of string1 into the register R0
  MOV R1, #0; counter for vowel letter
```

```
 MOV R2, #0 ;  counter for non-vowel letter


loopCount
        LDRB R3, [R0] ;Load Register Byte from address of register R0 to R3
        CMP R3, #0;compare R3 with 0
        BEQ CountDone;Branch to label CountDone
        CMP R3, #'a';compare R3 with a
                                ; vowel letter(aeuioAEIUO)
        BEQ AddVowel;Branch to label AddVowel
        CMP R3, #'e'
        BEQ AddVowel
        CMP R3, #'u'
        BEQ AddVowel
        CMP R3, #'o'
        BEQ AddVowel
        CMP R3, #'i'
        BEQ AddVowel
        CMP R3, #'A'
        BEQ AddVowel
        CMP R3, #'E'
        BEQ AddVowel
        CMP R3, #'U'
        BEQ AddVowel
```

```
        CMP R3, #'O'
        BEQ AddVowel
        CMP R3, #'I'
        BEQ AddVowel
        BNE AddNonVowel;Branch if not equal to label AddNonVowel
AddVowel
  ADD R1,#1;increment the conter of  vowel letter by 1
  ADD R0,#1;;go to the next letter in string
  B loopCount


AddNonVowel
  ADD R2,#1;increment the conter of non vowel letter by 1
  ADD R0,#1;go to the next letter in string
  B loopCount

CountDone
STOP
        B STOP
        END
```

5.4. Task code

```
;Name:Maha Maher Mali

;ID:1200746

;Given two string sort them alphabetically and store them in memory.

                PRESERVE8

        THUMB

        AREA RESET, DATA, READONLY

        EXPORT __Vectors

__Vectors

        DCD 0x20001000 ; stack pointer value when stack is empty

        DCD Reset_Handler ; reset vector

String1 DCB "aseel",0

String2 DCB "maher",0

        ALIGN


        AREA MYRAM, DATA, READWRITE


First space 10

Second space 10


        AREA MYCODE, CODE, READONLY

        ENTRY

        EXPORT Reset_Handler
```

```
Reset_Handler
        LDR R10, =First;loade the address of memory space First  to R10
        LDR R9, =Second;loade the address of memory space Second  to R9
        LDR R0,=String1;loaded the address of string1 to R0
        LDR R1,=String2
        MOV R5,#0
loopCounter
        LDRB R2, [R0, R5] ; we loaded Register Byte from memory address R0+ R5 to
R3
                                ;letter by letter from  string1
        LDRB R3, [R1, R5]

        ADD R5, #1; increment the accumulator R5 by 1
        CMP R2, R3
        BLT STORE_FIRST;Branch to label STORE_FIRST if R3 LESS THAN R2
        BGT  STORE_SECOND;Branch  to  label  STORE_SECOND  if  R2  GREATER
THAN R3
        CMP R2, #0;check if string1 end or not
        BNE loopCounter
        BEQ STOP
        CMP R3, #0;check if string2 end or not
        BNE loopCounter
        B STOP
```

```
STORE_FIRST ; STORE R0 in first and R1 in second.
        MOV R5,#0

loop1
        LDRB R2, [R0, R5];we loaded Register Byte from memory address R0+ R5 to R32
        STRB R2,[R10];store the charecter in R10
        ADD R5,#1;increment the acumlator
        ADD R10,#1; increment the memory space
        CMP R2,#0
        BNE loop1
        B STORE_SECOND

STORE_SECOND ; STORE R1 in first and R0 in second.
        MOV R5,#0
loop2
        LDRB R2, [R1, R5]
        STRB R2,[R9]
        ADD R5,#1
        ADD R9,#1
        CMP R2,#0
        BNE loop2
```

```
STOP
        B STOP
        END
```