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

List of figures

List of tables

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.

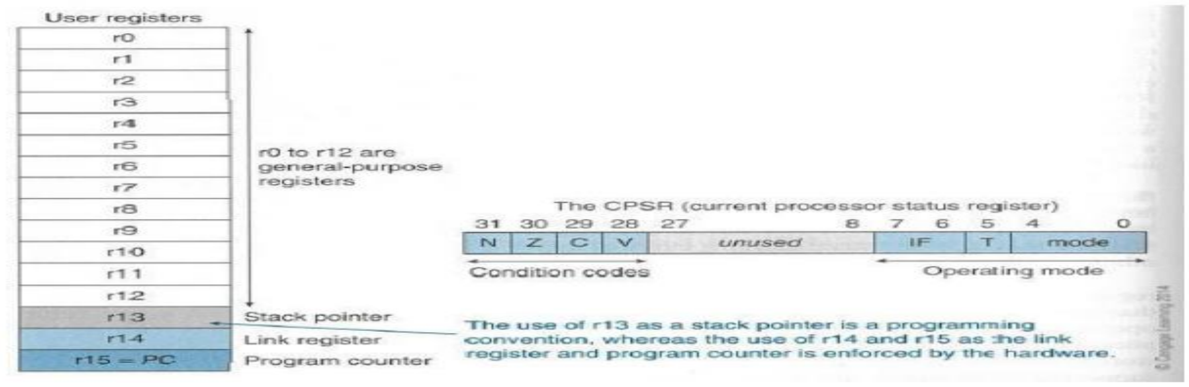

User registers
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13
r14
r15 = PC
r0 to r12 are general-purpose registers
Stack pointer
Link register
Program counter
The CPSR (current processor status register)
31 30 29 28 27 8 7 6 5 4 0
N Z C V unused IF T mode
Condition codes
Operating mode
The use of r13 as a stack pointer is a programming convention, whereas the use of r14 and r15 as the link register and program counter is enforced by the hardware.

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:

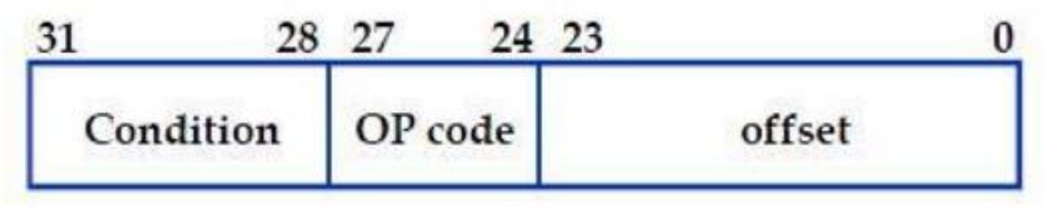
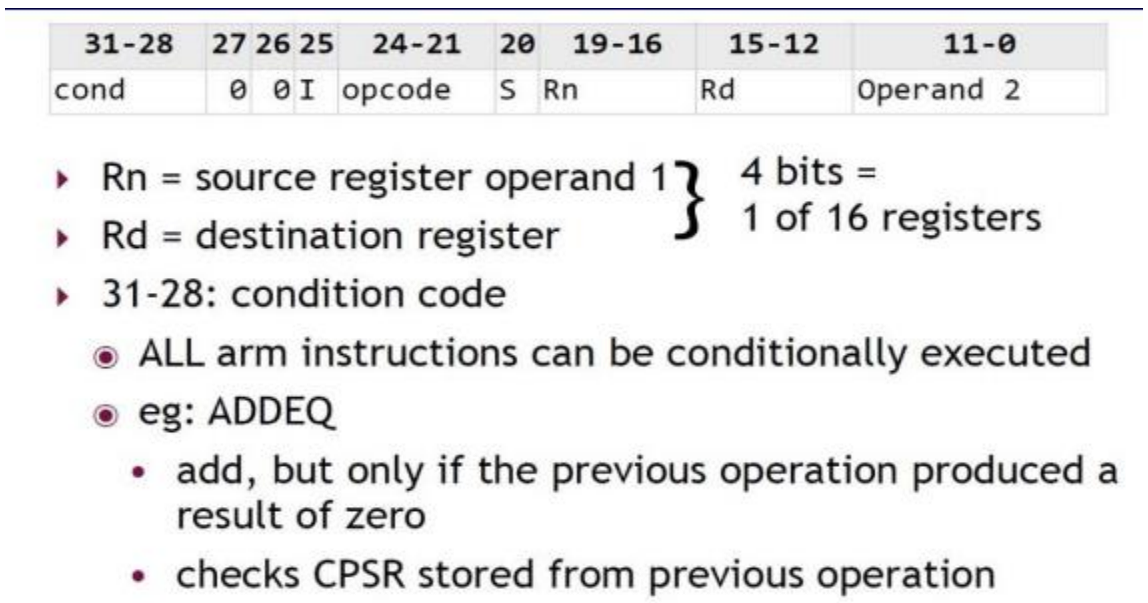


Figure 2: The encoding format for the branch instruction [3]

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].



31-28	27	26	25	24-21	20	19-16	15-12	11-0
cond	0	0	I	opcode	S	Rn	Rd	Operand 2

- Rn = source register operand 1 } 4 bits = 1 of 16 registers
- Rd = destination register
- 31-28: condition code
  - ALL arm instructions can be conditionally executed
  - eg: ADDEQ
    - add, but only if the previous operation produced a result of zero
    - checks CPSR stored from previous operation

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

------------------------------------------------------------------------------------------
```

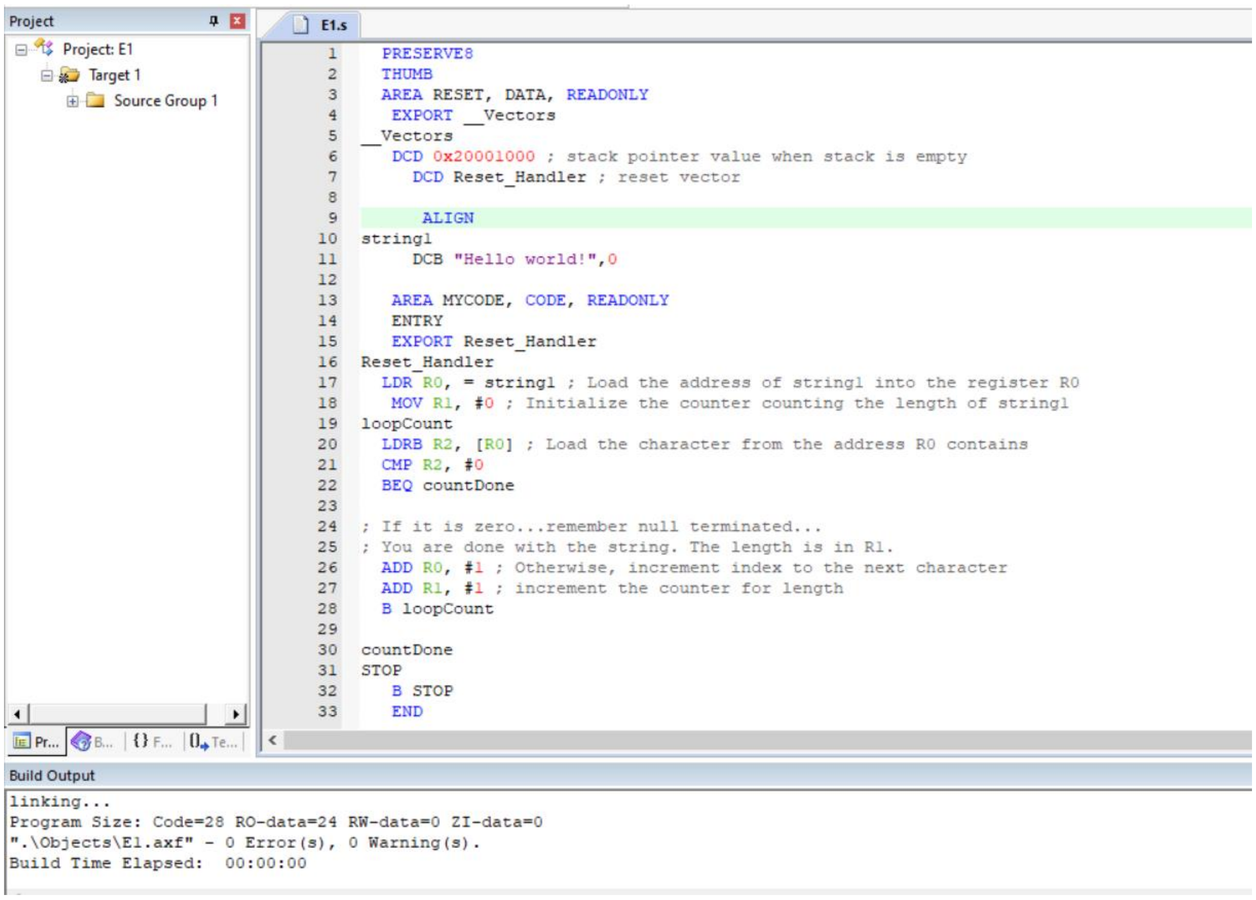
```
BGT labelAA                 ; Conditionally branch to label labelAA,
                            ; Z clear and either N set and V set
                            ; or N clear and V clear
                            ; that is, Z = 0 and N = V
--------------------------------------------------------------------------------
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

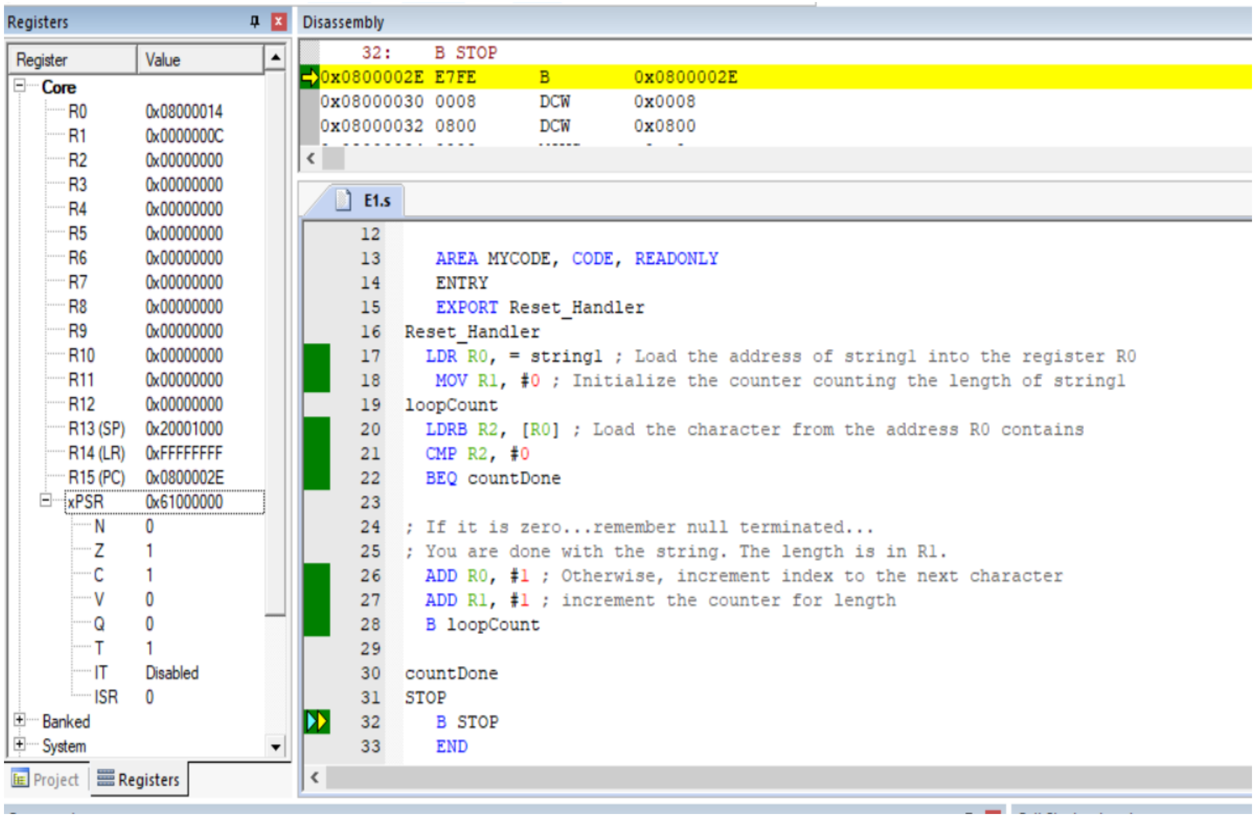
Example 1 Using Branch Instruction



```
   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 "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
STOP
    B STOP
    END
```

```
linking...
Program Size: Code=28 RO-data=24 RW-data=0 ZI-data=0
".\Objects\E1.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:00
```

Figure 4: Assembly code 1



Register	Value
Core	
R0	0x08000014
R1	0x0000000C
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0800002E
xPSR	0x61000000
N	0
Z	1
C	1
V	0
Q	0
T	1
IT	Disabled
ISR	0
Banked	
System	

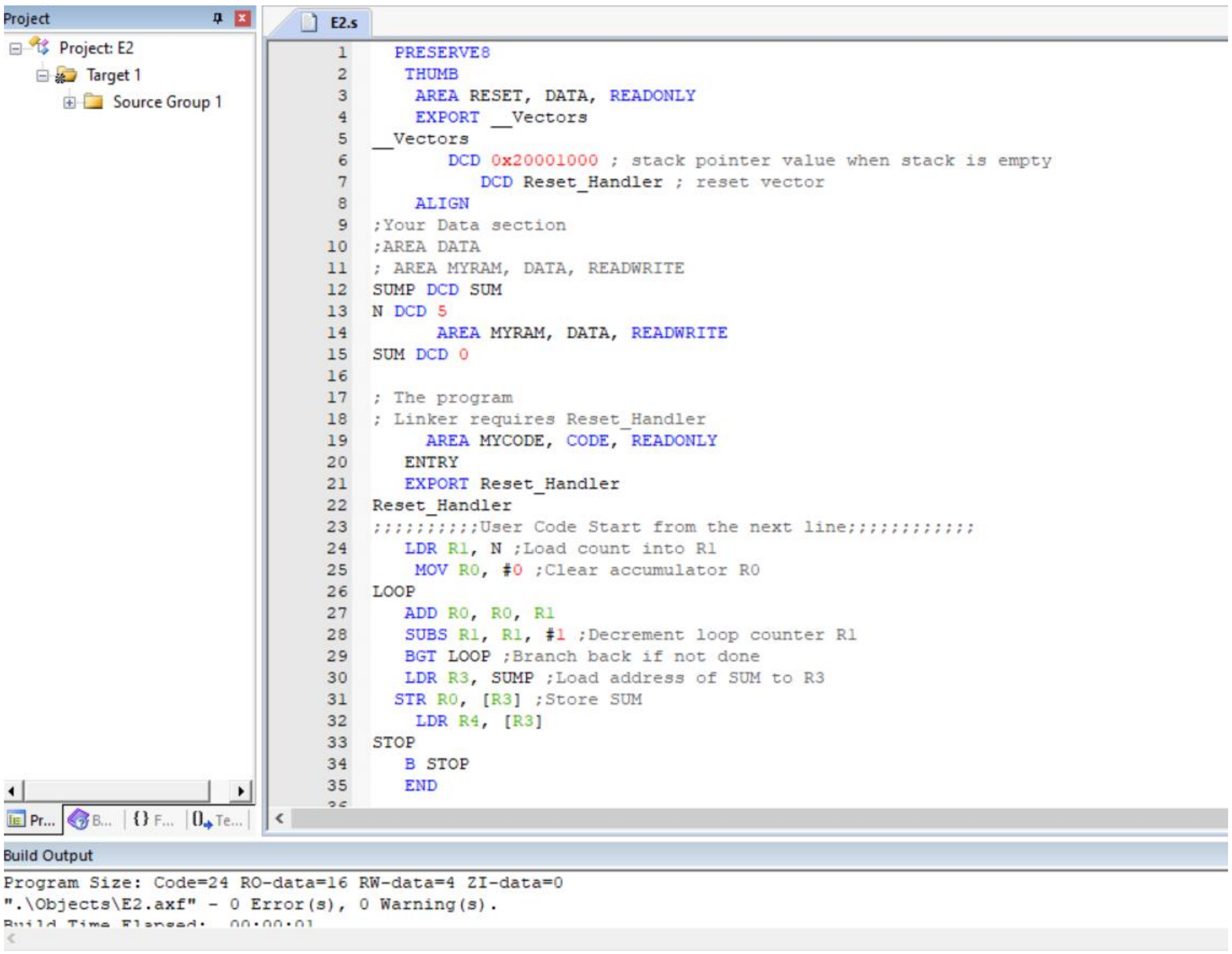
```
    32:     B STOP
0x0800002E E7FE      B        0x0800002E
0x08000030 0008      DCW      0x0008
0x08000032 0800      DCW      0x0800
```

```

   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
STOP
   B STOP
   END
```

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.

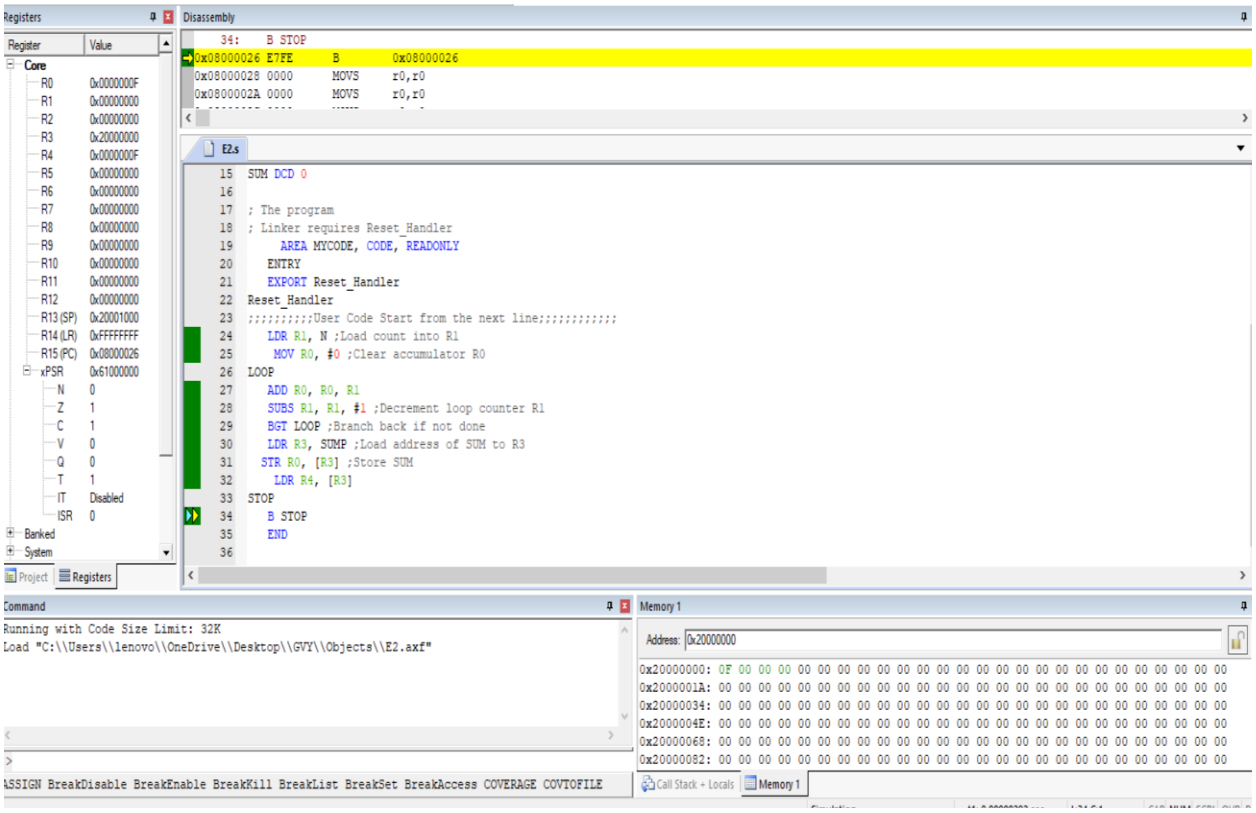
Example 2 Using Branch Instruction



```
  PRESERVE8
   THUMB
    AREA RESET, DATA, READONLY
    EXPORT __Vectors
__Vectors
       DCD 0x20001000 ; stack pointer value when stack is empty
          DCD Reset_Handler ; reset vector
    ALIGN
;Your Data section
;AREA DATA
; AREA MYRAM, DATA, READWRITE
SUMP DCD SUM
N DCD 5
      AREA MYRAM, DATA, READWRITE
SUM DCD 0

; The program
; Linker requires Reset_Handler
     AREA MYCODE, CODE, READONLY
   ENTRY
   EXPORT Reset_Handler
Reset_Handler
;;;;;;;;;;User Code Start from the next line;;;;;;;;;;;;
   LDR R1, N ;Load count into R1
    MOV R0, #0 ;Clear accumulator R0
LOOP
   ADD R0, R0, R1
   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
```

```
Program Size: Code=24 RO-data=16 RW-data=4 ZI-data=0
".\Objects\E2.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:01
```

Figure 6: Assembly code 2



Register	Value
Core	
R0	0x0000000F
R1	0x00000000
R2	0x00000000
R3	0x20000000
R4	0x0000000F
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x08000026
xPSR	0x61000000
N	0
Z	1
C	1
V	0
Q	0
T	1
IT	Disabled
ISR	0
Banked	
System	

```
    34:     B STOP
0x08000026 E7FE      B        0x08000026
0x08000028 0000      MOVS     r0,r0
0x0800002A 0000      MOVS     r0,r0
```

```
SUM DCD 0

; The program
; Linker requires Reset_Handler
     AREA MYCODE, CODE, READONLY
   ENTRY
   EXPORT Reset_Handler
Reset_Handler
;;;;;;;;;;User Code Start from the next line;;;;;;;;;;;;
   LDR R1, N ;Load count into R1
    MOV R0, #0 ;Clear accumulator R0
LOOP
   ADD R0, R0, R1
   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
```

```
Running with Code Size Limit: 32K
Load "C:\\Users\\lenovo\\OneDrive\\Desktop\\GVY\\Objects\\E2.axf"
```

```
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
```

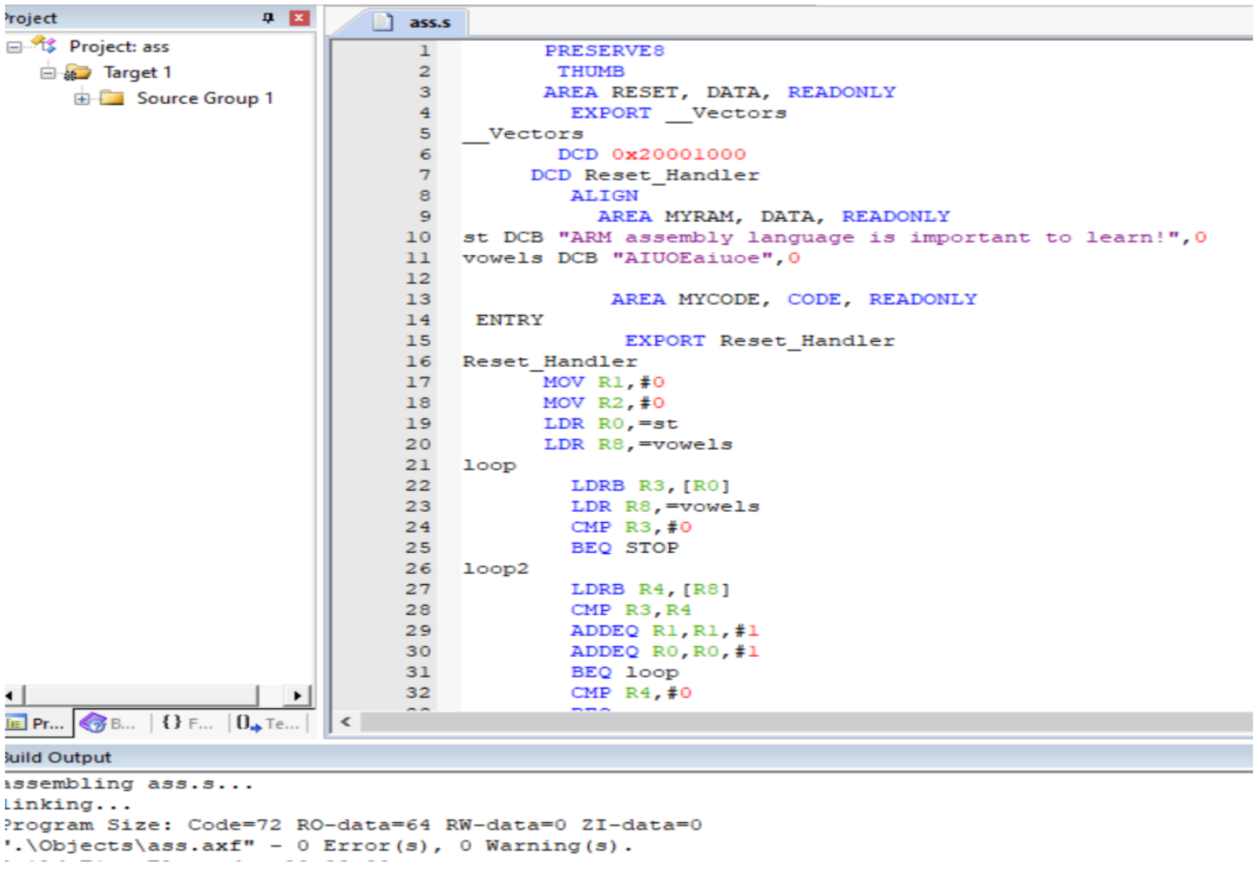
Address: 0x20000000

```
0x20000000: 0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000001A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000034: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000004E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000082: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 7: Debugging the second code

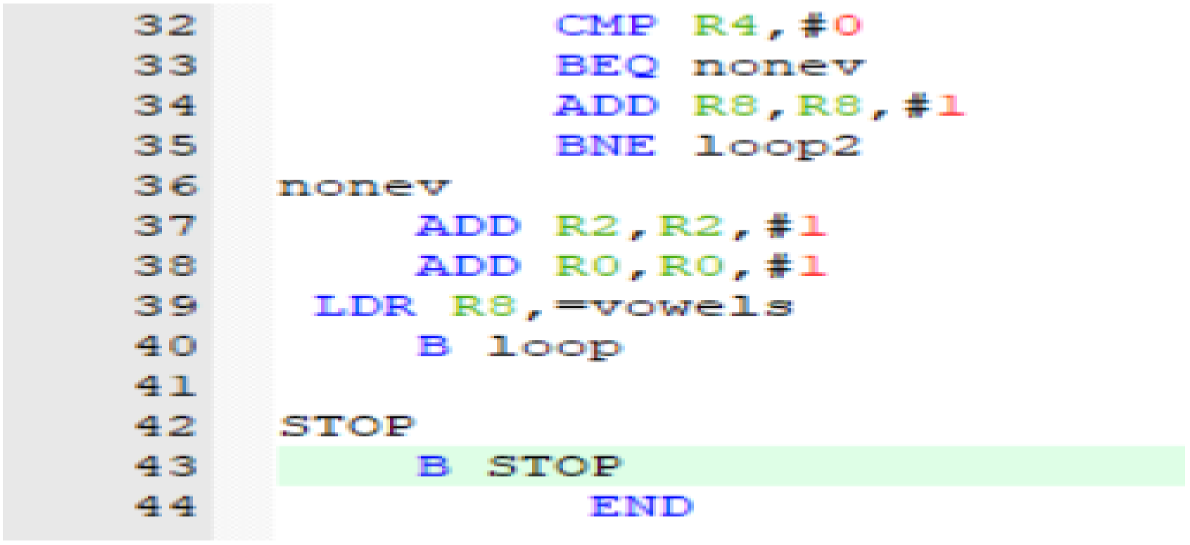
This code count the integer between 0 and N ,, but in this code N=5, so it adding the numbers between 0-5 which is 1+2+3+4+5 and the result equal 15 which equal F in hexadecimal and this result stored in register R0 and load into a register R4.

Lab Assignment



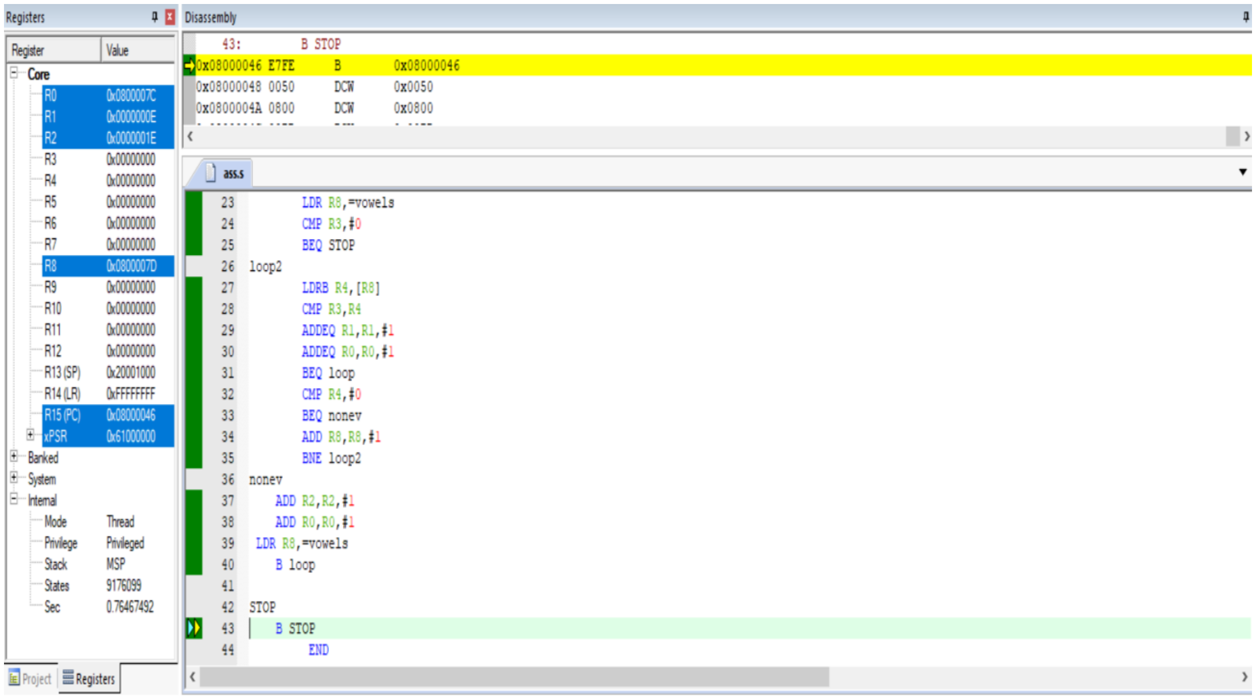
```
        PRESERVE8
         THUMB
        AREA RESET, DATA, READONLY
          EXPORT __Vectors
__Vectors
         DCD 0x20001000
       DCD Reset_Handler
          ALIGN
            AREA MYRAM, DATA, READONLY
st DCB "ARM assembly language is important to learn!",0
vowels DCB "AIUOEaiuoe",0

             AREA MYCODE, CODE, READONLY
 ENTRY
              EXPORT Reset_Handler
Reset_Handler
        MOV R1,#0
        MOV R2,#0
        LDR R0,=st
        LDR R8,=vowels
loop
          LDRB R3,[R0]
          LDR R8,=vowels
          CMP R3,#0
          BEQ STOP
loop2
          LDRB R4,[R8]
          CMP R3,R4
          ADDEQ R1,R1,#1
          ADDEQ R0,R0,#1
          BEQ loop
          CMP R4,#0
```

```
assembling ass.s...
Linking...
Program Size: Code=72 RO-data=64 RW-data=0 ZI-data=0
".\Objects\ass.axf" - 0 Error(s), 0 Warning(s).
```



```
                    CMP R4,#0
                    BEQ nonev
                    ADD R8,R8,#1
                    BNE loop2
nonev
          ADD R2,R2,#1
          ADD R0,R0,#1
   LDR R8,=vowels
          B loop

STOP
          B STOP
                 END
```

Figure 8: Lab assignment code



Register	Value
Core	
R0	0x0800007C
R1	0x0000000E
R2	0x0000001E
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x0800007D
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x08000046
xPSR	0x61000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	9176099
Sec	0.76467492

```
    43:         B STOP
0x08000046 E7FE      B        0x08000046
0x08000048 0050      DCW      0x0050
0x0800004A 0800      DCW      0x0800
```

ass.s

```
        LDR R8,=vowels
        CMP R3,#0
        BEQ STOP
loop2
        LDRB R4,[R8]
        CMP R3,R4
        ADDEQ R1,R1,#1
        ADDEQ R0,R0,#1
        BEQ loop
        CMP R4,#0
        BEQ nonev
        ADD R8,R8,#1
        BNE loop2
nonev
    ADD R2,R2,#1
    ADD R0,R0,#1
 LDR R8,=vowels
    B loop

STOP
    B STOP
          END
```

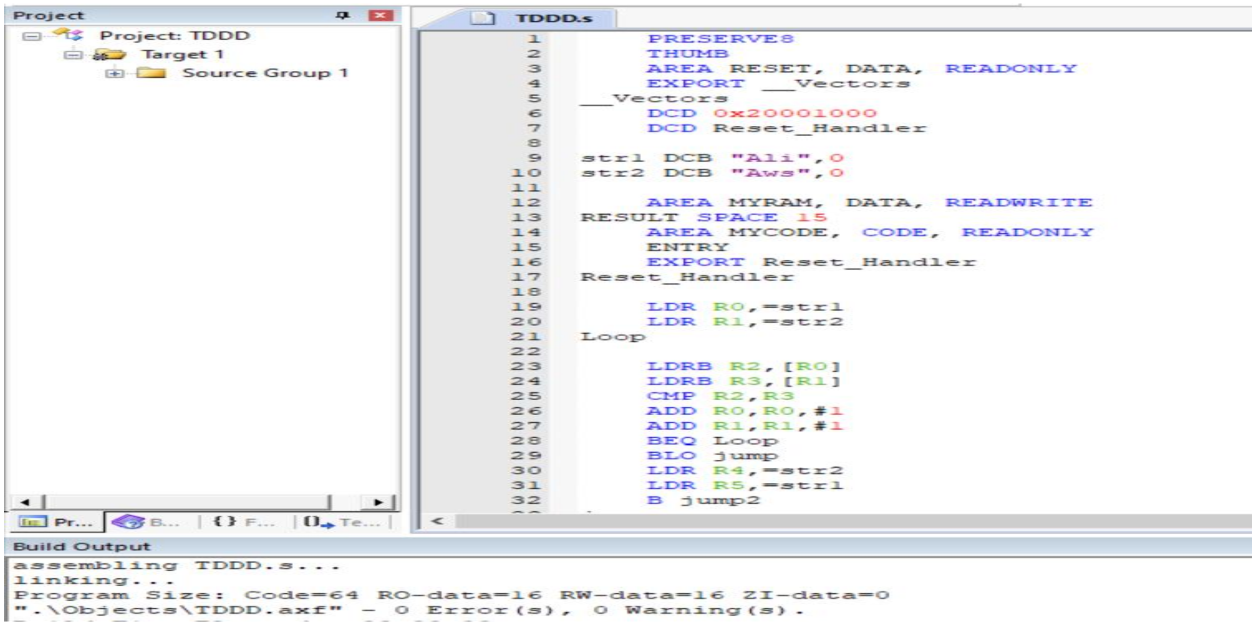
Figure 9: Debugging lab assignment code

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.

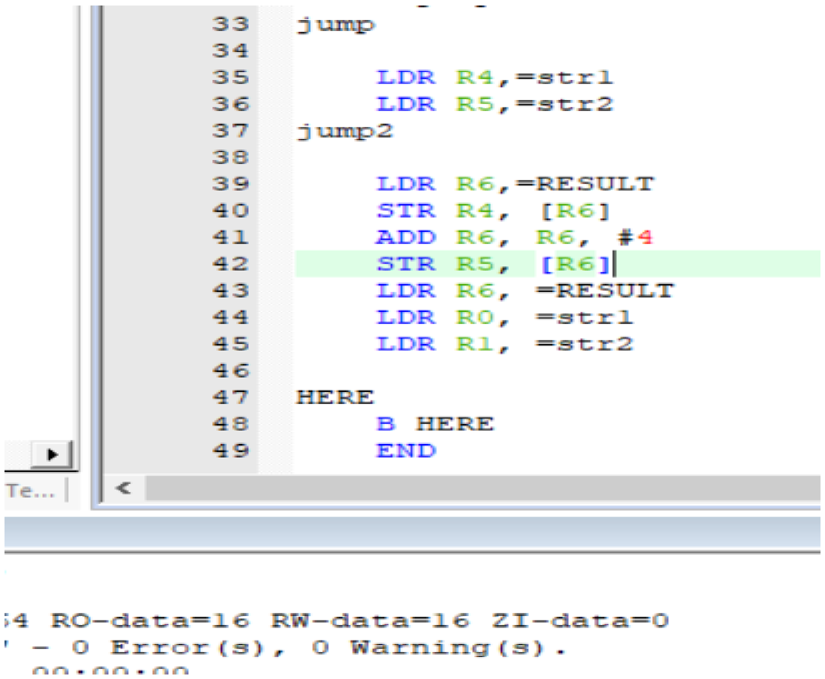
Task

Given Two strings sort them alphabetically and store them in memory



```
        PRESERVE8
        THUMB
        AREA RESET, DATA, READONLY
        EXPORT __Vectors
__Vectors
        DCD 0x20001000
        DCD Reset_Handler

str1 DCB "Ali",0
str2 DCB "Aws",0

        AREA MYRAM, DATA, READWRITE
RESULT SPACE 15
        AREA MYCODE, CODE, READONLY
        ENTRY
        EXPORT Reset_Handler
Reset_Handler

        LDR R0,=str1
        LDR R1,=str2
Loop

        LDRB R2, [R0]
        LDRB R3, [R1]
        CMP R2,R3
        ADD R0,R0,#1
        ADD R1,R1,#1
        BEQ Loop
        BLO jump
        LDR R4,=str2
        LDR R5,=str1
        B jump2
```

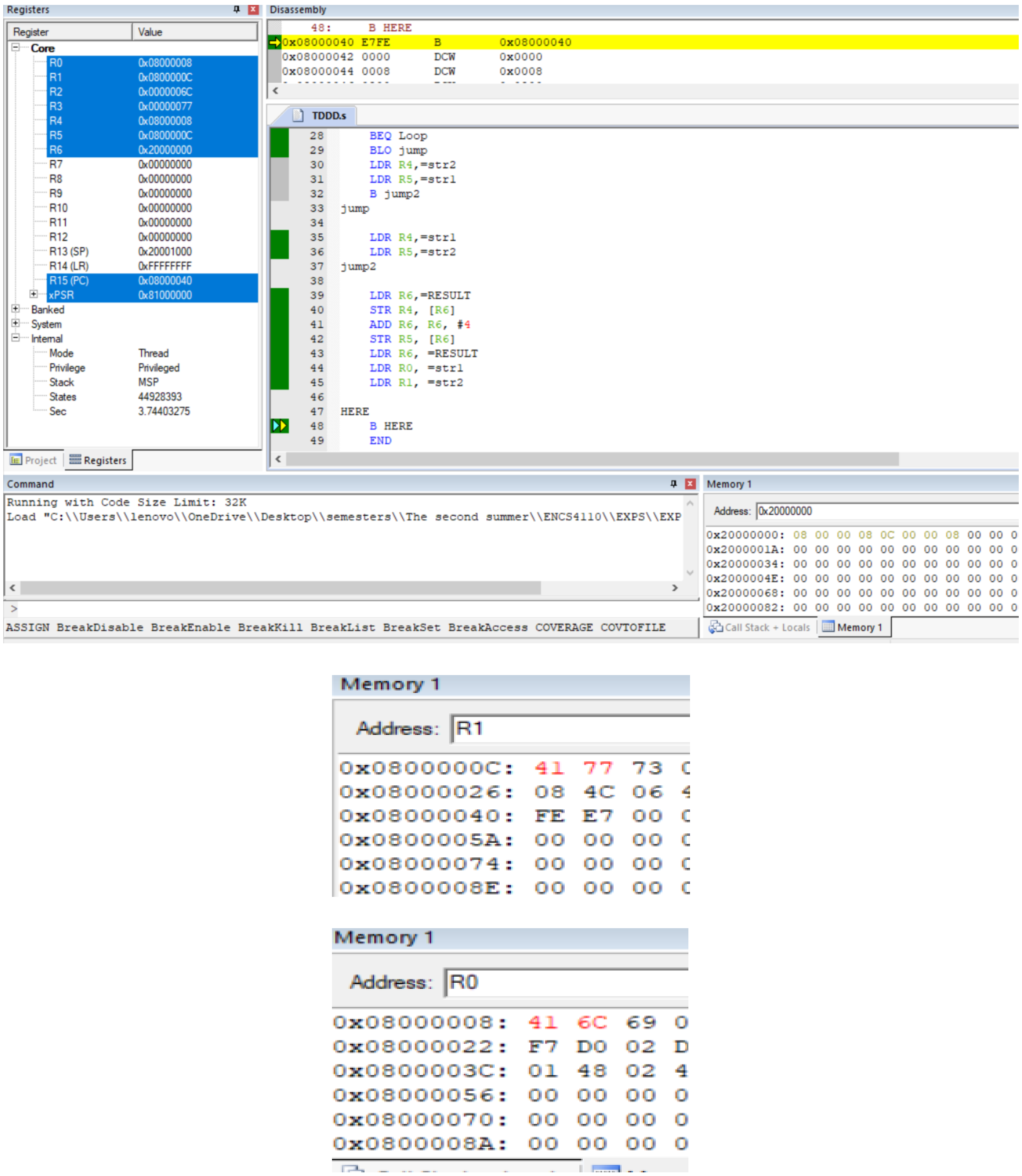
```
assembling TDDD.s...
linking...
Program Size: Code=64 RO-data=16 RW-data=16 ZI-data=0
".\Objects\TDDD.axf" - 0 Error(s), 0 Warning(s).
```



```
jump

        LDR R4,=str1
        LDR R5,=str2
jump2

        LDR R6,=RESULT
        STR R4, [R6]
        ADD R6, R6, #4
        STR R5, [R6]
        LDR R6, =RESULT
        LDR R0, =str1
        LDR R1, =str2

HERE
        B HERE
        END
```

```
4 RO-data=16 RW-data=16 ZI-data=0
' - 0 Error(s), 0 Warning(s).
```

Figure 10: Task code



```
Registers
Register        Value
Core
  R0            0x08000008
  R1            0x0800000C
  R2            0x0000006C
  R3            0x00000077
  R4            0x08000008
  R5            0x0800000C
  R6            0x20000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x20001000
  R14 (LR)      0xFFFFFFFF
  R15 (PC)      0x08000040
  xPSR          0x81000000
Banked
System
Internal
  Mode          Thread
  Privilege     Privileged
  Stack         MSP
  States        44928393
  Sec           3.74403275
```

```
Disassembly
    48:        B HERE
0x08000040 E7FE      B        0x08000040
0x08000042 0000      DCW      0x0000
0x08000044 0008      DCW      0x0008
```

```
TDDD.s
       BEQ Loop
       BLO jump
       LDR R4,=str2
       LDR R5,=str1
       B jump2
jump

       LDR R4,=str1
       LDR R5,=str2
jump2

       LDR R6,=RESULT
       STR R4, [R6]
       ADD R6, R6, #4
       STR R5, [R6]
       LDR R6, =RESULT
       LDR R0, =str1
       LDR R1, =str2

HERE
       B HERE
       END
```

```
Command
Running with Code Size Limit: 32K
Load "C:\\Users\\lenovo\\OneDrive\\Desktop\\semesters\\The second summer\\ENCS4110\\EXPS\\EXP
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
```

```
Memory 1
Address: 0x20000000
0x20000000: 08 00 00 08 0C 00 00 08 00 00 0
0x2000001A: 00 00 00 00 00 00 00 00 00 00 0
0x20000034: 00 00 00 00 00 00 00 00 00 00 0
0x2000004E: 00 00 00 00 00 00 00 00 00 00 0
0x20000068: 00 00 00 00 00 00 00 00 00 00 0
0x20000082: 00 00 00 00 00 00 00 00 00 00 0
```

```
Memory 1
Address: R1
0x0800000C: 41 77 73
0x08000026: 08 4C 06
0x08000040: FE E7 00
0x0800005A: 00 00 00
0x08000074: 00 00 00
0x0800008E: 00 00 00
```

```
Memory 1
Address: R0
0x08000008: 41 6C 69 0
0x08000022: F7 D0 02 D
0x0800003C: 01 48 02 4
0x08000056: 00 00 00 0
0x08000070: 00 00 00 0
0x0800008A: 00 00 00 0
```

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