



BERZIET UNIVERSITY

Faculty of Information Technology Computer Systems

Engineering Department

COMPUTER DESIGN LAB

ENCS4110

ARM's Flow Control Instructions

Report NO.2/ Experiment _3(in Arm)

Prepared by:

Nour Naji- 1190270

Partners:

Rasha Dar Abu Zidan -1190547

Supervised by: Dr. Jamal Seyam

Teacher Assistant: Eng. Raha Zabadi

Section: 2

Date: 20 September 2021

Abstract

The aim of this experience is to learn how to work with ARM assembly language using application Keil uVision5, as well as to learn the ARM branch, execute instructions and Instruction Format, and understand the importance of condition flags, and use all of those to create ARM assembly programs.

Table of Contents

Abstract	i
Table of Contents	ii
List Of Figures	iii
1. Theory	1
1.1 ARM Register Set	1
1.2 ARM CPSR (Current Program Status Register)	2
1.2.1 C, the carry flag	2
1.2.2 Z, the zero flag	2
1.2.3 N, the negative flag.....	3
1.2.4 V, the overflow flag.....	3
1.3 ARM Instruction Format	3
1.4 Branch Instructions	3
2. Procedure & Discussion.....	5
2.1 String Length Counter Program	5
2.2 Sum of Integers from 1 To 5	6
3. TO DO - Lab Assignment-	7
4. Conclusion	8
5. References.....	9
6. Appendices.....	10
6.1 Appendix A	10
6.2 Appendix B	11
6.3 Appendix C	12

List Of Figures

Figure 1.1. 1:ARM Registers Data Size.....	1
Figure 1.1. 2:ARM Registers	2
Figure 1.2. 1:CPSR (Current Program Status Register)	2
Figure 1.4. 1:Branch Instructions.....	3
Figure 2.1. 1:Debugging the Code.....	5
Figure 2.2. 1:Debugging the Code-2.....	6
Figure 3. 1:Debugging the Code-3.....	7

1. Theory

Arm is a RISC (reduced instruction set computing) architecture developed by Arm Limited. This processor architecture is nothing new. It was first used in personal computers as far back as the 1980s.

The ARM processor is becoming the dominant CPU architecture in the computer industry. It is already the leading architecture in cell phones and tablet computers. With such a large number of companies producing ARM chips, it is certain that the architecture will move to the laptop, desktop and high-performance computers currently dominated by x86 architecture from Intel and AMD. Currently the PIC and AVR microcontrollers dominate the 8-bit microcontroller market. The ARM architecture will have a major impact in this area too as designers become more familiar with its architecture.[2]

1.1 ARM Register Set

The CPU, registers are used to store information temporarily. That information could be a byte of data to be processed, or an address pointing to the data to be fetched. All of ARM registers are 32-bit wide. The 32 bits of a register are shown in Figure [1.1.1]. These range from the MSB (most-significant bit) D31 to the LSB (least-significant bit) D0. With a 32-bit data type, any data larger than 32 bits must be broken into 32-bit chunks before it is processed. Although the ARM default data size is 32-bit many assemblers also support the single bit, 8-bit, and 16-bit data types. The 32-bit data size of the ARM is often referred as word. This is in contrast to x86 CPU in which word is defined as 16-bit. In ARM the 16-bit data is referred to as half-word. Therefore, ARM supports byte, half-word (two byte), and word (four bytes) data types.[2]

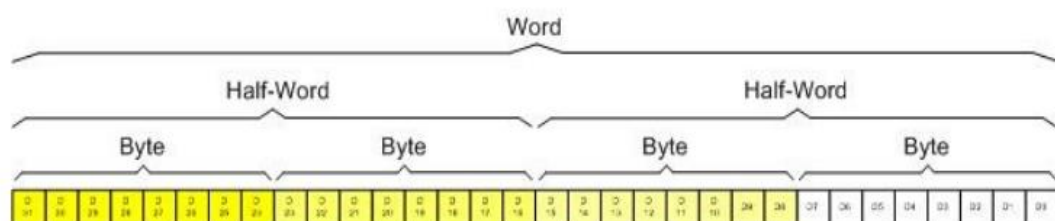


Figure 1.1. 1: ARM Registers Data Size

In ARM there are 13 general purpose registers. They are R0–R12 as shown in Figure [1.1.2], all of these registers are 32 bits wide, the general-purpose registers in ARM are the same as the accumulator in other microprocessors. They can be used by all arithmetic and logic instructions.

The ARM core has three special function registers of R13, R14, and R15, their jobs are as follows:

- 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.[1]

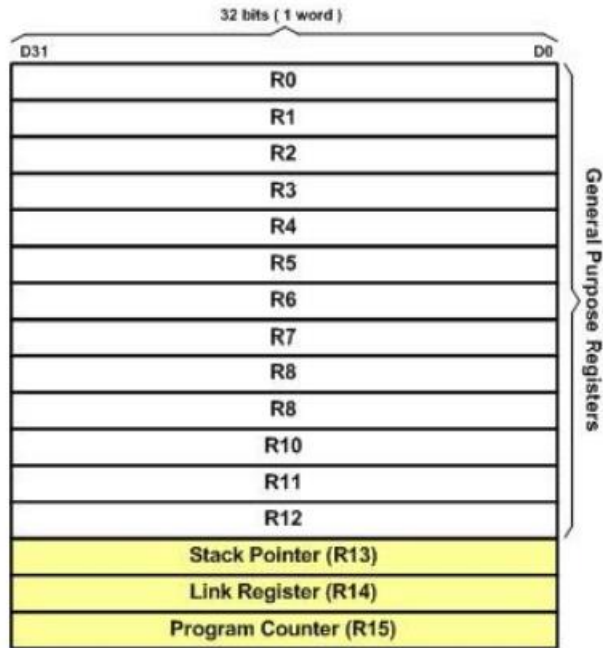


Figure 1.1. 2: ARM Registers

1.2 ARM CPSR (Current Program Status Register)

Like all other microprocessors, the ARM has a flag register to indicate arithmetic conditions such as the carry bit. The flag register in the ARM is called the current program status register (CPSR).

The status register is a 32-bit register, The bits C, Z, N, and V are called conditional flags, meaning that they indicate some conditions that result after an instruction is executed. Each of the conditional flags can be used to perform a conditional branch (jump), as shown in Figure [1.1.3].^[2]



Figure 1.2. 1: CPSR (Current Program Status Register)

1.2.1 C, the carry flag

This flag is set whenever there is a carry out from the D31 bit. This flag bit is affected after a 32-bit addition or subtraction.^[2]

1.2.2 Z, the zero flag

The zero flag reflects the result of an arithmetic or logic operation. If the result is zero, then $Z = 1$. Therefore, $Z = 0$ if the result is not zero.^[2]

1.2.3 N, the negative flag

Binary representation of signed numbers uses D31 as the sign bit. The negative flag reflects the result of an arithmetic operation. If the D31 bit of the result is zero, then $N = 0$ and the result is positive. If the D31 bit is one, then $N = 1$ and the result is negative. The negative and V flag bits are used for the signed number arithmetic operations.[2]

1.2.4 V, the overflow flag

This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit. In general, the carry flag is used to detect errors in unsigned arithmetic operations while the overflow flag is used to detect errors in signed arithmetic operations.[2]

1.3 ARM Instruction Format

The basic expression for arithmetic instructions is \rightarrow **Opcode Rd, Rn, Rm.**

For example :

- $\text{ADD R0, R2, R4} \rightarrow$ Performs the operation $R0 \leftarrow [R2] + [R4]$
- $\text{SUB R0, R6, R5} \rightarrow$ Performs the operation $R0 \leftarrow [R6] - [R5]$
- Immediate mode: $\text{ADD R0, R3, \#17} \rightarrow$ Performs the operation $R0 \leftarrow [R3] + 17$
- $\text{ADD R0, R1, R5, LSL \#4} \rightarrow$ the second operand stored in R5 is shifted left 4-bit positions (equivalent to $[R5] \times 16$), and it is then added to the contents of R1; the sum is placed in R0.

1.4 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 the format for the branch instructions is shown as below:

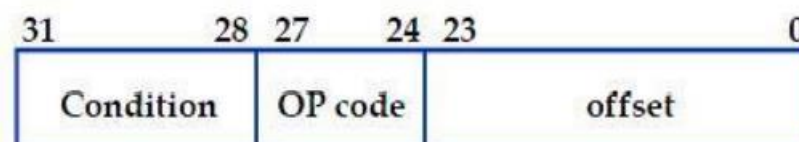


Figure 1.4. 1: Branch Instructions

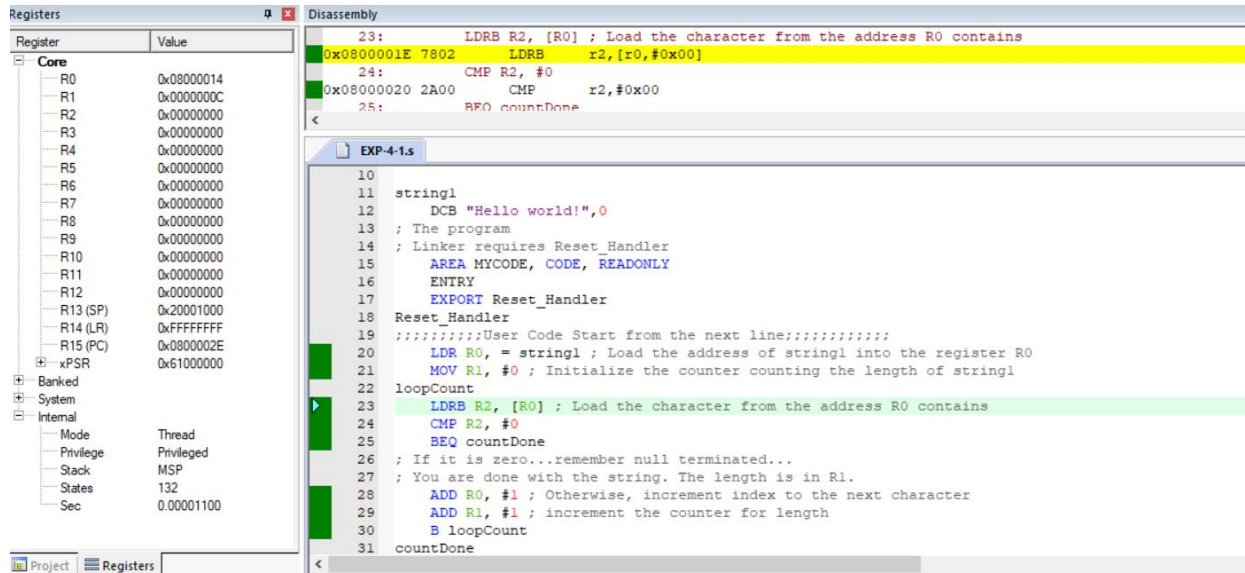
Offset is a signed 24-bit number. It is shifted left two-bit positions (all branch targets are aligned word addresses), signed extended to 32 bits, and added to the

updated PC to generate the branch target address, The updated points to the instruction that is two words (8 bytes) forward from the branch instruction.[1]

2. Procedure & Discussion

2.1 String Length Counter Program

The goal of this program is to calculate the length of the sentence, i.e., how many letters it consists of.



The screenshot shows a debugger interface with two main panes. The left pane, titled 'Registers', displays a list of registers and their values. The right pane, titled 'Disassembly', shows the assembly code for the program. The registers pane shows R0 with value 0x08000014, R1 with 0x0000000C, R2 with 0x00000000, and R3 through R15 with various values. The disassembly pane shows the following code:

```
23: LDRB R2, [R0] ; Load the character from the address R0 contains
0x0800001E 7802 LDRB r2, [r0,#0x00]
24: CMP R2, #0
0x08000020 2A00 CMP r2,#0x00
25: BEQ countDone

EXP-4-1.s
10
11 string1
12 DCB "Hello world!",0
13 ; The program
14 ; Linker requires Reset_Handler
15 AREA MYCODE, CODE, READONLY
16 ENTRY
17 EXPORT Reset_Handler
18 Reset_Handler
19 ;;;;User Code Start from the next line;;;;;
20 LDR R0, = string1 ; Load the address of string1 into the register R0
21 MOV R1, #0 ; Initialize the counter counting the length of string1
22 loopCount
23 LDRB R2, [R0] ; Load the character from the address R0 contains
24 CMP R2, #0
25 BEQ countDone
26 ; If it is zero...remember null terminated...
27 ; You are done with the string. The length is in R1.
28 ADD R0, #1 ; Otherwise, increment index to the next character
29 ADD R1, #1 ; increment the counter for length
30 B loopCount
31 countDone
```

Figure 2.1. 1: Debugging the Code

First, we notice in line 12 that 0 is placed at the end of the line, and the reason for this is to find out the end of the line and stop the loop.

The process begins by loading the address of string 1 into register R0, zeroing the counter, and creating a loop to count the characters, storing a character from R0 to R1, comparing this character with zero, if it is zero, the loop is exited and the program stops, and if it is not zero, the counter is incremented with one and move to the second character, and this is repeated until the end of the line is reached

We notice that the program ends when the register R2 becomes zero, and R1 will be 0x0000000C which means 12 and this is the length of the string (Hello World!), At the same time, R0 will store the address of the string "Hello World!" which is 0x08000014.

✓ Note: The full code can find in Appendix (A).

2.2 Sum of Integers from 1 To 5

The goal of this program is to calculate the sum of the numbers from 1 to N.

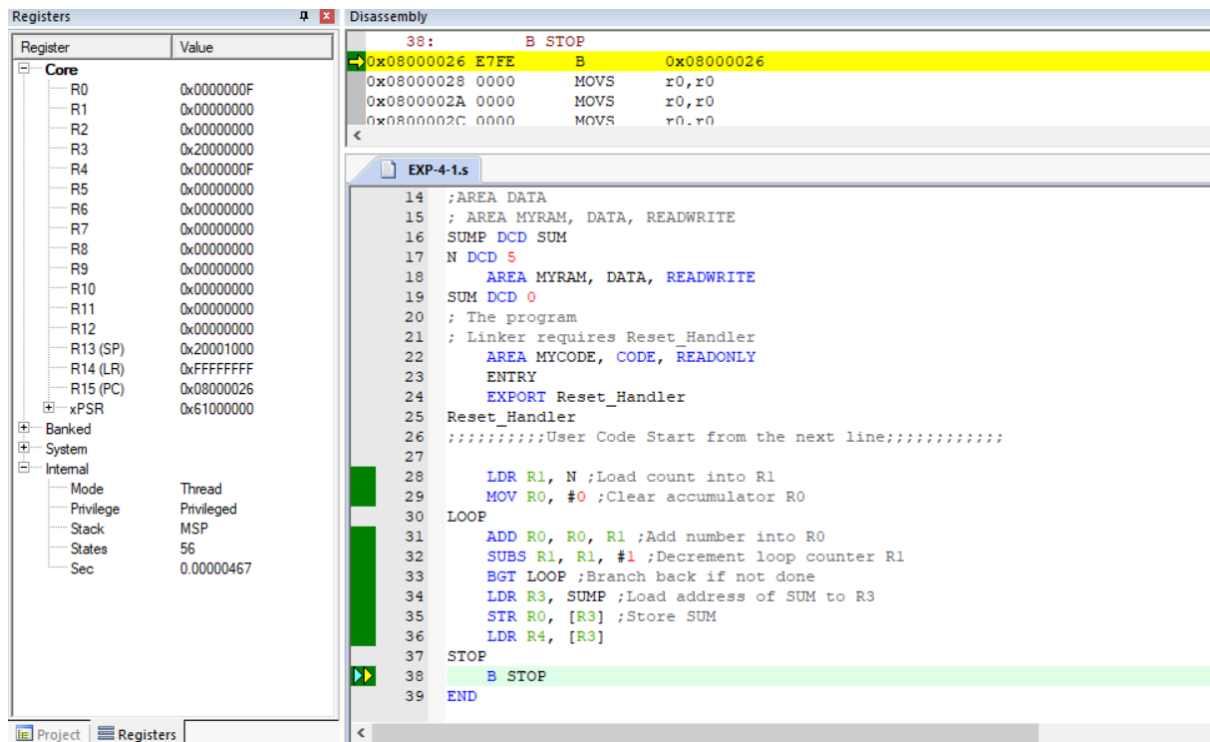


Figure 2.2. 1: Debugging the Code-2

Initially the number 5 is stored in N, so the program calculates the sum of the numbers from 1 to 5 \rightarrow (1+2+3+4+5).

The number 5 is stored in R1, and R0 is zeroed (where the sum is stored), a loop is created, it ends when the number stored in N becomes zero.

First, the number stored in the register R1 is added to the number stored in the register R0, and the result is stored in R0, then 1 is subtracted from R1, If R1 is zero, the loop is exited, and if it is not zero, an address is loaded from SUM to R3, and Store SUM in R0, and load it in R4.

As shown from figure 2.2.1, the sum was in R0 and its 0x0000000F which is 15.

✓ Note: The full code can find in Appendix (B).

3. TO DO - Lab Assignment-

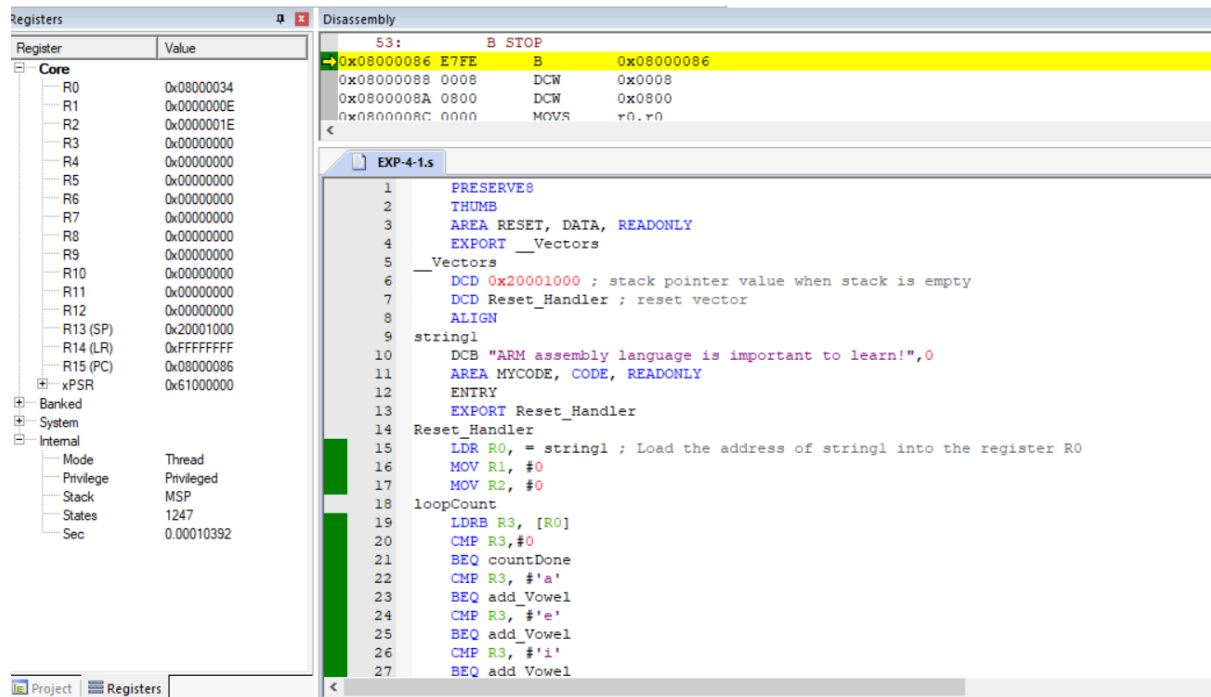


Figure3. 1:Debugging the Code-3

The goal of the program is to count the number of vowels and the number of non-vowels in the string "ARM assembly language is important to learn!".

Initially, the address of the string was put in the memory register R0, and two counters have been created, R1 counter for vowels and R2 counter for non-vowels, a loop is created, inside the loop, will compare the characters if there are vowels or not, if the character is one of these characters (a, i, o, u, e) then its vowels and R1 is incremented by 1, other is not vowels and R2 is incremented by 1.

As shown from figure 3.1, in R1 the value is 0x0000000E which is 14 and this is the number of vowels character, on another hand, in R2 the value is 0x0000001E which is 30 and this the number of non-vowels character.

✓ Note: The full code can find in Appendix (C).

4. Conclusion

We learnt how to work with strings and numbers in an ARM assembly, and learnt about the various instructions and their respective functions, and how to use branch and compare instructions to create loops and conditional statements, and used all of these concepts to create a different program.

5. References

- [1] Computer Design Laboratory Manual.
- [2] ARM Assembly Language_ Programming and Architecture

6. Appendices

6.1 Appendix A

```
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
; The program
; Linker requires Reset_Handler
    AREA MYCODE, CODE, READONLY
    ENTRY
    EXPORT Reset_Handler
Reset_Handler
;;;;;;;;;;User Code Start from the next line;;;;;;;;;;
    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 ; End of the program

6.2 Appendix B

PRESERVE8

THUMB

; Linker requires __Vectors to be exported

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

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

6.3 Appendix C

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

MOV R2, #0

loopCount

LDRB R3, [R0]

CMP R3,#0

BEQ countDone

CMP R3, #'a'

BEQ add_Vowel

CMP R3, #'e'

BEQ add_Vowel

CMP R3, #'i'

BEQ add_Vowel

CMP R3, #'o'

BEQ add_Vowel

CMP R3, #'u'

BEQ add_Vowel

CMP R3, #'A'

BEQ add_Vowel


```
CMP R3, #'E'  
BEQ add_Vowel  
CMP R3, #'I'  
BEQ add_Vowel  
CMP R3, #'O'  
BEQ add_Vowel  
CMP R3, #'U'  
BEQ add_Vowel  
BNE notVowel  
add_Vowel  
    ADD R1, #1 ; Increment the vowel counter by 1  
    ADD R0, #1  
    B loopCount  
notVowel  
    ADD R2, #1  
    ADD R0, #1  
    B loopCount  
countDone  
STOP  
    B STOP  
END
```