

**Faculty of Engineering and Technology**

**Electrical And Computer Engineering Department**

**Computer Design Lab**

**ENCS4110**

**Assembly Addressing Modes Lab Report**

**Done By:** Amal Ziad\_1192141

**Instructor:** Ahmad Afaneh

**Assistant**: Hiba Qadah

**Sections:** 2

**Date:** 13/4/2022

# Abstract:

The aim of this experiment is to learn how to use arm assembly language to achieve programs by using the registers of the computer, specifically to learn arm addressing modes, all this by using **Keil uVision5.**

# Contents

[Abstract: 2](#_Toc101120279)

[Contents 3](#_Toc101120280)

[Theory: 4](#_Toc101120281)

[Program of Using Post-indexing Mode (#1): 5](#_Toc101120282)

[Program of Using Post-indexing Mode (#2): 6](#_Toc101120283)

[Lab Work #1: 7](#_Toc101120284)

[Lab Work #2: 8](#_Toc101120285)

[Conclusion: 11](#_Toc101120286)

# Theory:

We know that we have 16 registers we can use with arm assembly; registers from 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.

* **ARM addressing modes:**

The ARM instruction set architecture is a **Load/Store** architecture, which means that data values must be loaded into CPU registers before performing arithmetic or logic operations on them. The instructions that load data values from memory, or store data values in memory cannot alter the value.[[1]](#footnote-1)

* **Register Indirect Addressing Mode:**

Register indirect addressing means that the location of an operand is held in a register. It is also called indexed addressing or base addressing.

* **Register Indirect Addressing with an Offset :**

ARM supports a memory-addressing mode where the effective address of an operand is computed by adding the content of a register and a literal offset coded into load/store instruction.

Ex: LDR R0, [R1, #20] ; loads R0 with the word pointed at by R1+20

* **ARM's Autoindexing Pre-indexed Addressing Mode:**

As the previous but the only difference is ‘!” which means to increment the address to point at R + (#offset) the next time.

Ex: LDR R0, [R1, #4]! R1 + 4 ; loads R0 with the word pointed at by R1+4

; then update the pointer by adding 4 to R1

* **ARM's Autoindexing Post-indexing Addressing Mode:**

The same as above but it first accesses the operand at the location pointed by the base register, then increments the base register.

Ex: LDR R0, [R1], #4 R1 ; loads R0 with the word pointed at by R1

; then update the pointer by adding 4 to R1

# Program of Using Post-indexing Mode (#1):

The code (Procedure):

**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**

**NUM1 DCD 3, -7, 2, -2, 10**

**POINTER DCD NUM1**

**AREA MYRAM, DATA, READWRITE**

**SUM DCD 0**

**AREA MYCODE, CODE, READONLY**

**ENTRY**

**EXPORT Reset\_Handler**

**Reset\_Handler**

**LDR R1, N ;R1 is the counter for the array**

**LDR R2, POINTER**

**MOV R0, #0 ;where we will store the result**

**LOOP**

**LDR R3, [R2], #4**

**ADD R0, R0, R3**

**SUBS R1, R1, #1**

**BGT LOOP**

**LDR R4, SUMP**

**STR R0, [R4]**

**LDR R6, [R4] ; showing the final result in R6**

**STOP**

**B STOP**

**END**

Discussion: the program above will sum the numbers in the array, which is assigned to NUM1 variable. First the program will load n in R1 (which is for the index of the array (counter)), then loading POINTER in R2. After entering the loop, it will load the value of R2 in the memory (which is for the pointer (initially it will point at the first element in the array)), then increment the pointer by 4 to point to the next element (because we stored its values in a word which takes 4 bits), and so on. Then add the R0 (which is for the sum).Then decrementing the counter by 1 and compare it with zero, if it is greater than zero, branch to the beginning of the loop. And so on till the counter reaches zero so its done and after finishing the result will be stored in the byte address R4 and show the result in R6. As shown in fig.1

The result:

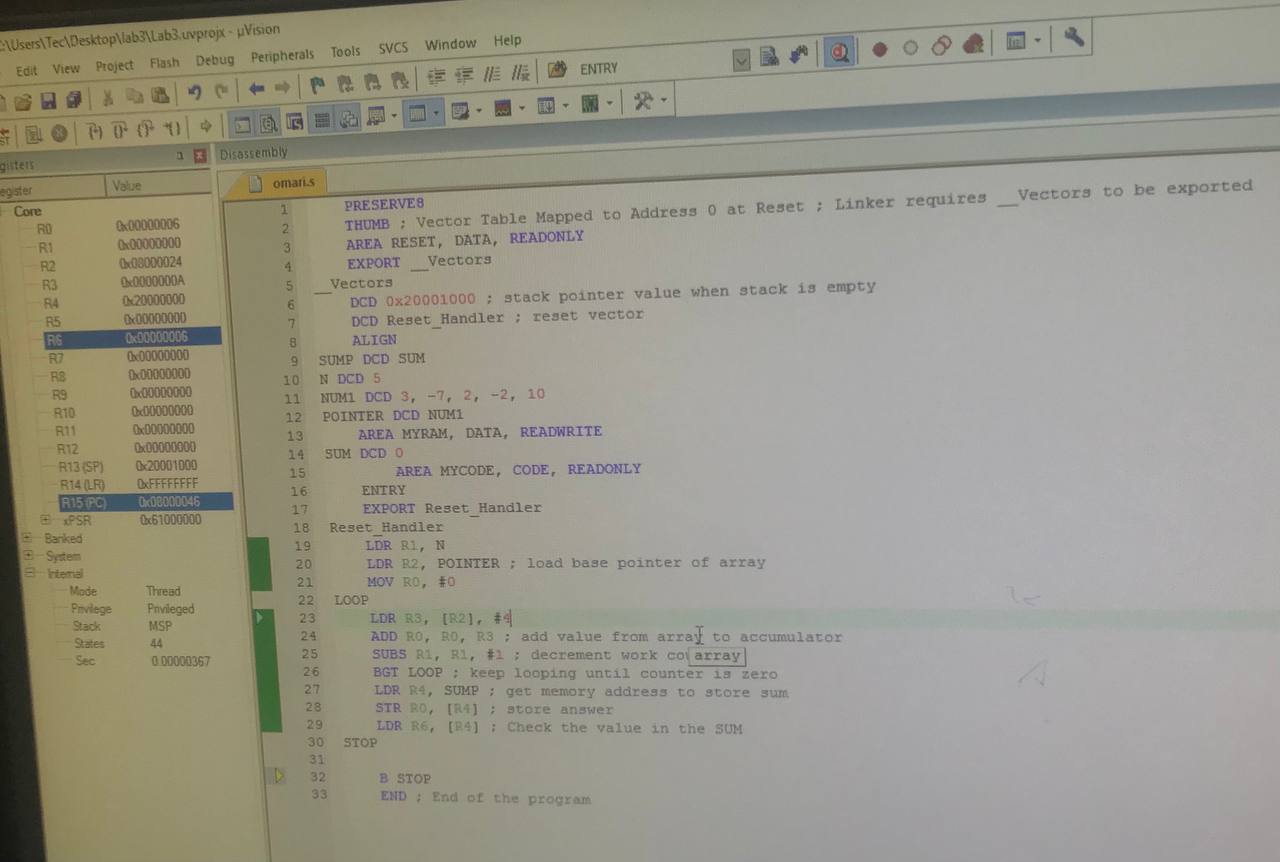


Fig.1

# Program of Using Post-indexing Mode (#2):

The code:

**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 MYRAM, DATA, READWRITE**

**SUM DCD 0**

**AREA MYCODE, CODE, READONLY**

**ENTRY**

**EXPORT Reset\_Handler**

**Reset\_Handler**

**LDR R0, = string1**

**MOV R1, #0 ;the counter of the array**

**loopCount**

**LDRB R2, [R0], #1**

**CBZ R2, STOP**

**ADD R0, #1 ;to the next character**

**ADD R1, #1 ;calculating length**

**B loopCount**

**STOP**

**B STOP**

**END**

Discussion: this program will count the length of the string. After assigning the string “Hello world!” to the variable string1, the program will load it in R0, then start its operations in the loop by get the first byte of the string stored in R0 and increment the pointer by 1. If the pointer gives zero then the string is done and exits the loop, if not then increment R1 by 1 (which is the counter of the string), and R1 by 1 to the next character.

The answer we have gotten is R1=C.

# Lab Work #1:

The code (Procedure):

**PRESERVE8**

**THUMB**

**AREA RESET, DATA, READONLY**

**EXPORT \_\_Vectors**

**\_\_Vectors**

**DCD 0x20001000 ; stack pointer value when stack is empty**

**DCD Reset\_Handler ; reset vector**

**ALIGN**

**SUM DCD 0**

**SUMP DCD SUM**

**N DCD 7**

**NUM1 DCD 3, -7, 2, -2, 10, 20, 30**

**POINTER DCD NUM1**

**AREA MYCODE, CODE, READONLY**

**ENTRY**

**EXPORT Reset\_Handler**

**Reset\_Handler**

**LDR R0, POINTER**

**MOV R1, #0 ;the sum will be here**

**LDR R3, N**

**loopCount**

**LDR R2, [R0], #4 ;**

**CMP R2, #5**

**BGT doAdd**

**return**

**SUBS R3,R3, #1**

**BLT done**

**B loopCount**

**doAdd**

**ADD R1,R1,R2**

**B return**

**done**

**END**

Discussion: the program will sum the numbers greater than 5. After assigning variables, first loading pointer in R0 which points to the first element in the array. Then N in R3 (which is for number of elements in the array). In the loop, get the value of R0 in R2 and increment the pointer by 4 (because its stored as word) to get the next value and compare it with 5, if it is greater than 5 then branch to doAdd and add the value of R2 to R1 (R1 is for the required sum). Then decrement R3 by 1, if R3 equals zero then the program is done. As shown in fig.2.

The result: R3=3C=60 (true)

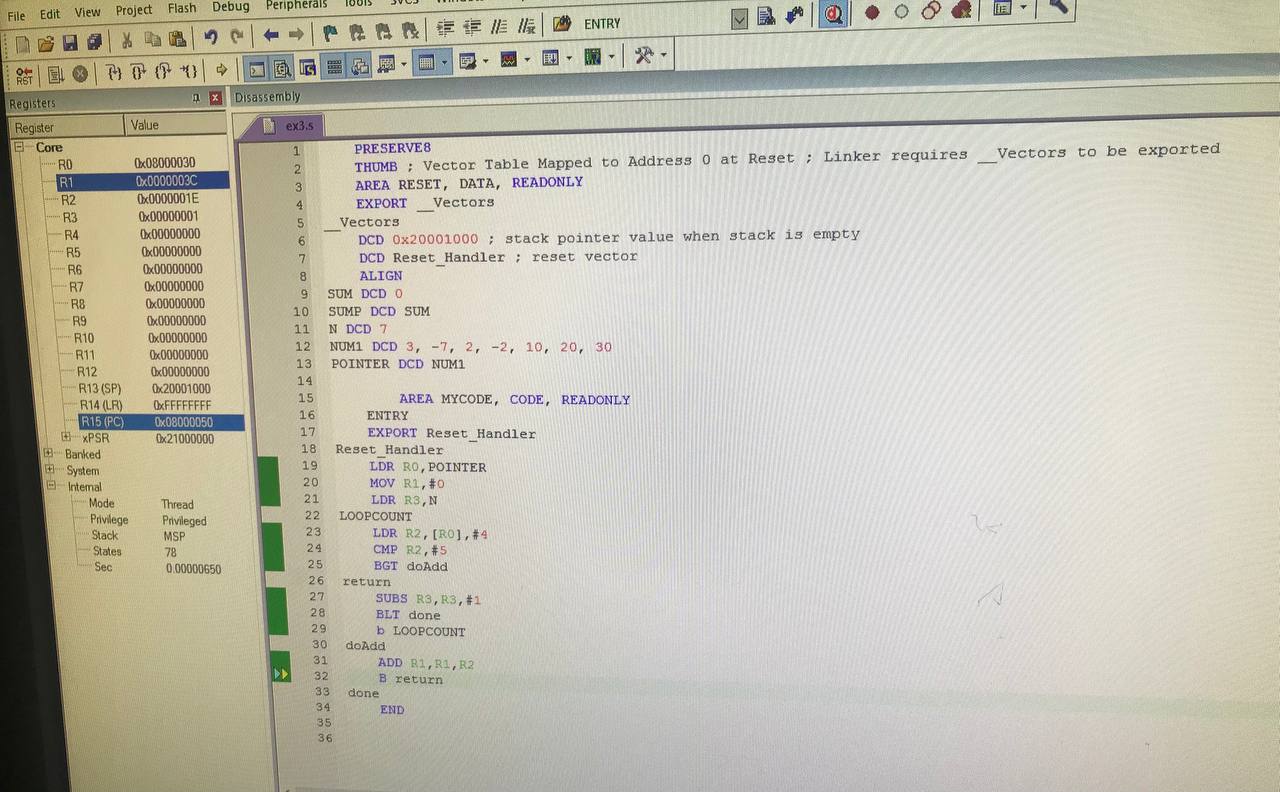


Fig.2

# Lab Work #2:

The code (Procedure):

**PRESERVE8**

**THUMB**

**AREA RESET, DATA, READONLY**

**EXPORT \_\_Vectors**

**\_\_Vectors**

**DCD 0x20001000 ; stack pointer value when stack is empty**

**DCD Reset\_Handler ; reset vector**

**ALIGN**

**Max DCD 0**

**MaxP DCD Max**

**Min DCD 0**

**MinP DCD Min**

**N DCD 12**

**NUM1 DCD 3, -7, 2, -2, 10, 20, 30, 15, 32, 8, 64, 66**

**POINTER DCD NUM1**

**AREA MYCODE, CODE, READONLY**

**ENTRY**

**EXPORT Reset\_Handler**

**Reset\_Handler**

**LDR R0, POINTER**

**LDR R3, N ;R3 counter of the array elements**

**MOV R4,#0 ;for max value**

**MOV R5,#0 ;for min value**

**LOOP**

**LDR R2, [R0], #4**

**CMP R2, R4**

**BGT MAX**

**CMP R2,R5**

**BLT MIN**

**CMP R3,#0**

**BEQ done**

**B LOOP**

**MAX**

**MOV R4,R2**

**SUBS R3,R3,#1**

**B LOOP**

**MIN**

**MOV R5,R2**

**SUBS R3,R3,#1**

**B LOOP**

**done**

**END**

Discussion: the program will get the max and min value of array elements. After assigning variables especially the array and the pointer. After entering the loop, the first element of the array will be in R2, then increment R0 address by 4 to get the next element. Then comparing the present value in R2 with R4 (which is for the max value), if it is greater than then replace it with the new value. If not then compare it with the min value in R5, if less than then replace it with the new value in R5. And so on till N in R3 becomes zero. As shown in fig.3.1 and fig.3.2.

The result: max value R4=(42)hex=(66)dec & min value R5=(FFFFFFF9)hex=(-7)dec (true answer).

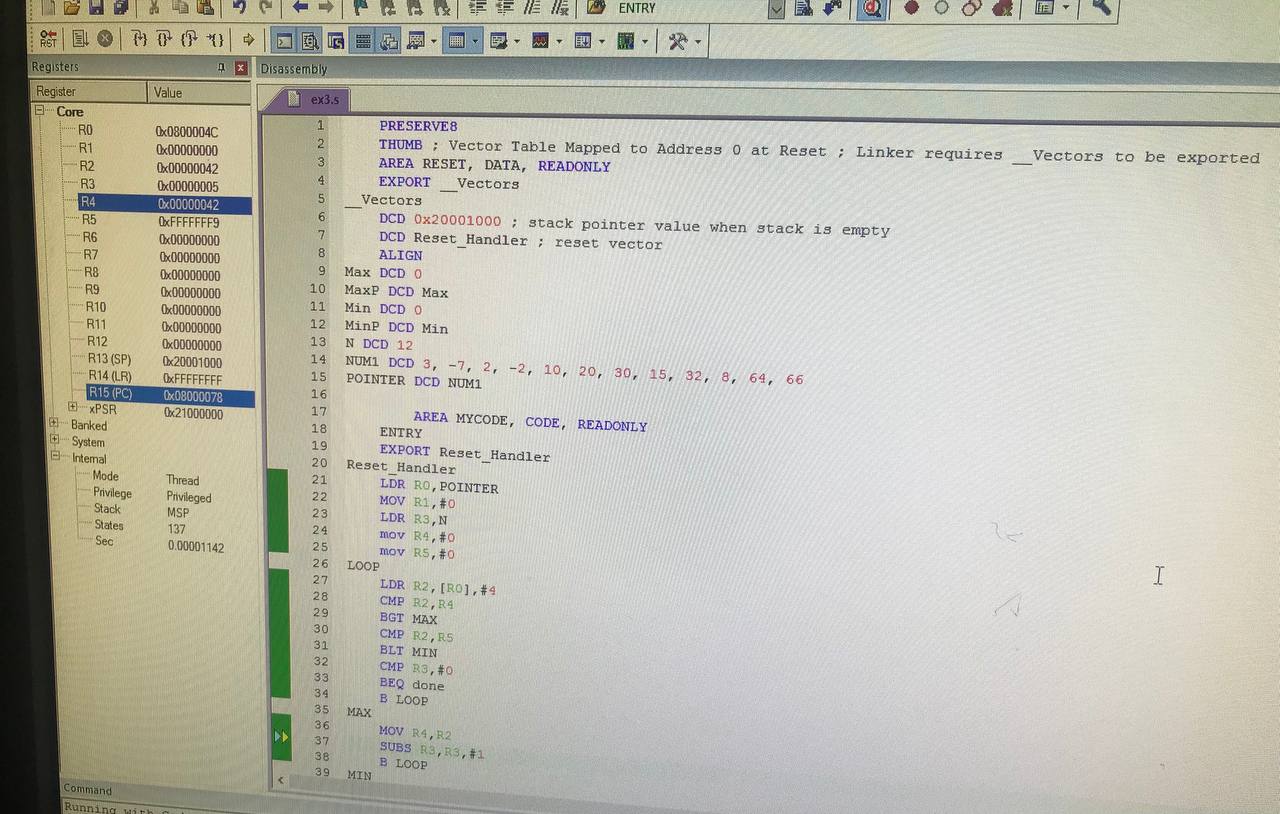


Fig.3.1

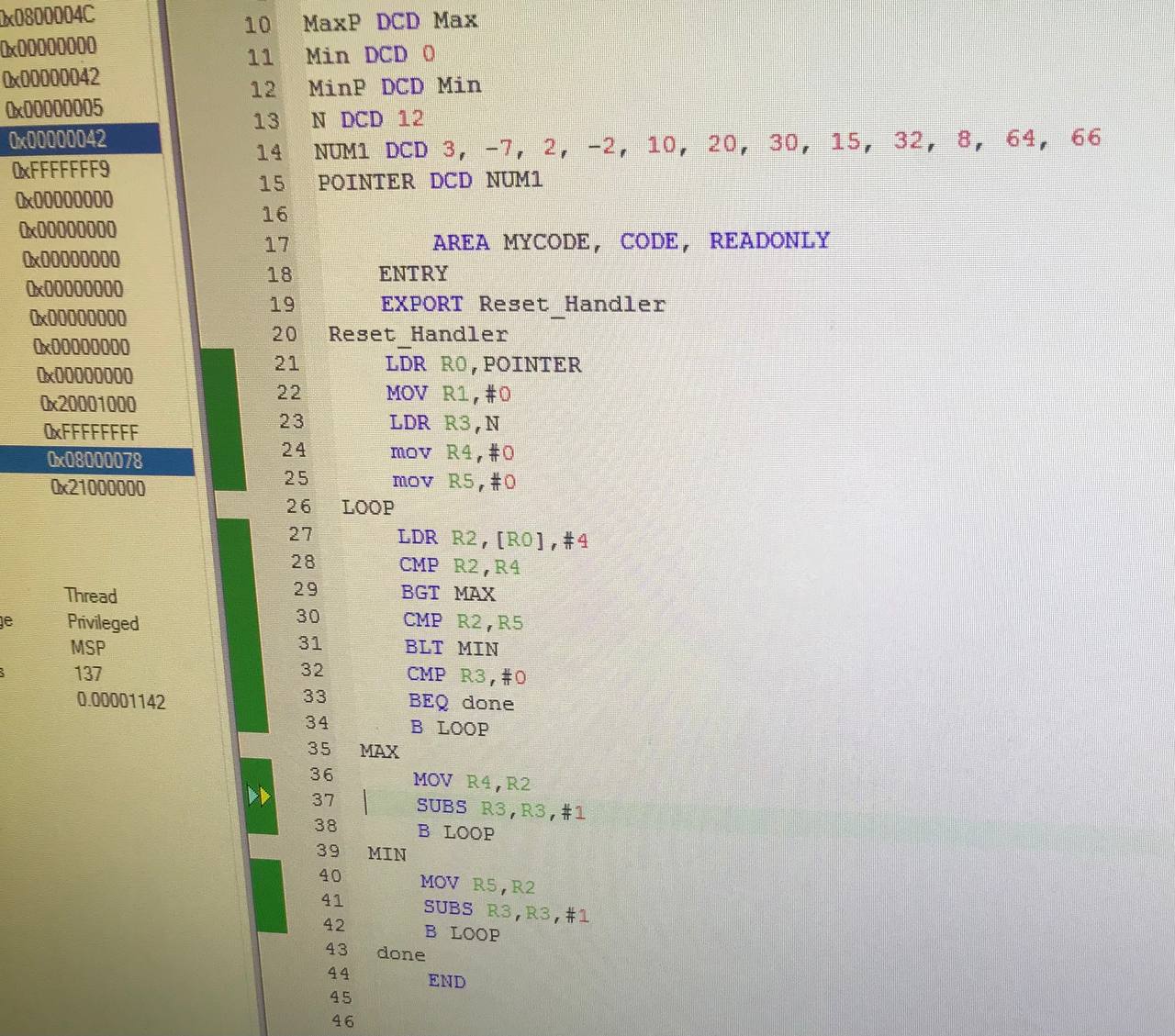


Fig.3.2

# Conclusion:

In conclusion, we used to deal with arm assembly in general, and get to know the exact concept of addressing modes in particular. Finally, we learned how to appoint our knowledge to build programs depending on registers.

1. <https://bob.cs.sonoma.edu/IntroCompOrg-RPi/sec-addr-mode.html> 13/4/2022 at 8:15 pm [↑](#footnote-ref-1)