BIRZEIT UNIVERSITY

Faculty of Engineering & Technology
Electrical & Computer Engineering Department
COMPUTER DESIGN LABORATORY - ENCS 4110
Experiment#4: ARM Addressing Modes
Report#1

Prepared by: Rahaf Naser
ID: 1201319

Instructor: Dr. Abdelsalam sayyad
Teaching Assistant: Eng. Raha Zabadi

Section: 3
Date: 23/3/2024

Abstract

The aim of this experiment is to understand Register Addressing Mode, Register Indirect Addressing Mode, ARM's Autoindexing Pre-indexed Addressing Mode, ARM's Autoindexing Post-indexing Addressing Mode and Program Counter Relative (PC Relative) Addressing Mode. We'll practice all of this in the lab.

Table of contents

List of Figures

1.Theory

1.1.Literal Addressing Mode

The immediate or literal addressing mode is where a literal number appears as a parameter to an instruction, such as mov eax, 128 where 128 would be the literal or immediate value [1] .

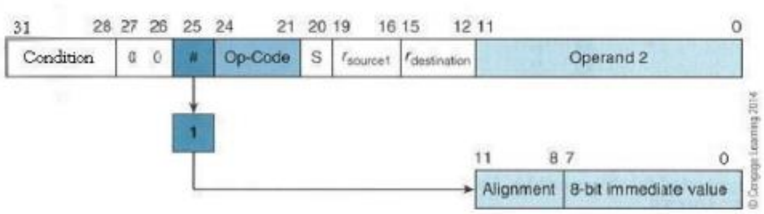


Figure 1: Literal Addressing Mode [4].

1.2.Register Indirect Addressing Mode

We use a processor register to hold a memory location's address wherever the operand has been placed. This addressing mode would allow the execution of a similar set of instructions for various different memory locations. It can be done if we increment the content of the register and, thereby, point to the new location every single time [2] .

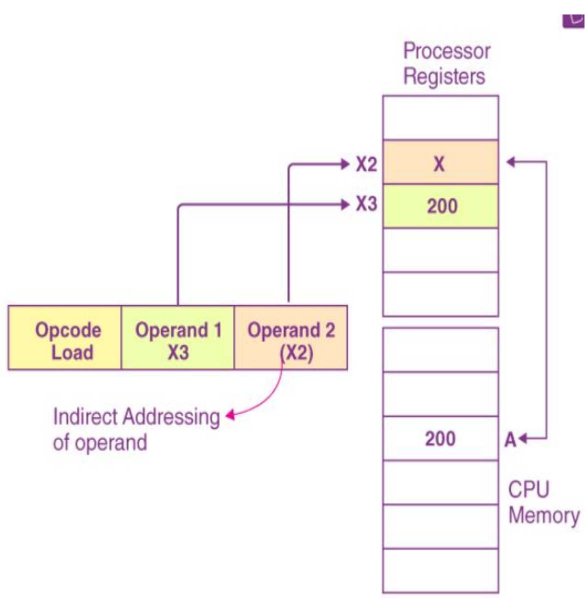


Figure 2 : Register Indirect Addressing Mode [2].

1.3.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 [3].

1.4.ARM's Autoindexing Pre-indexed Addressing Mode

This is used to facilitate the reading of sequential data in structures such as arrays, tables, and vectors. A pointer register is used to hold the base address. An offset can be added to achieve the effective address [4].

1.5.ARM's Autoindexing Post-indexing Addressing Mode

This is similar to the above, but it first accesses the operand at the location pointed by the base register, then increments the base register [4].

1.6.Program Counter Relative (PC Relative) Addressing Mode

Program counter relative addressing is a technique that allows you to access data or instructions relative to the current value of the program counter (PC) register. This can be useful for writing compact and portable code, as well as for implementing jump tables, switch statements, and loops [5].

1.7.ARM's Load and Store Encoding Format

Memory access operations have a conditional execution field in bit 31, 03, 29, and 28. The load and store instructions can be conditionally executed depending on a condition specified in the instruction [4].

1.8.Encoding Format of ARM's load and store instructions

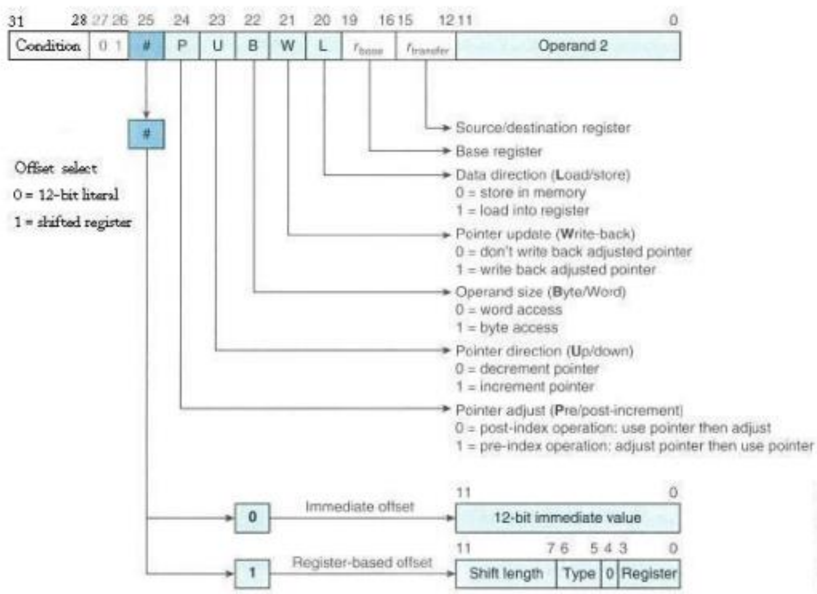


Figure 3 : Encoding Format of ARM's load and store instructions [4].

2.Procedure and Results

We practiced by running two examples and checking the results using debugging. Afterward, to ensure we understood the concept, we completed our lab assignment.

2.1 Example#1

2.1.1 Example#1 code

The code is to calculate the sum of numbers in array NUM1 and store sum in R0.

```
;;; Directives
      PRESERVE8
      THUMB
; Vector Table Mapped to Address 0 at Reset
; Linker requires __Vectors to be exported
      AREA RESET, DATA, READONLY
      EXPORT __Vectors
__Vectors
    DCD 0x20001000 ; stack pointer value when stack is
      DCD Reset_Handler ; reset vector

      ALIGN

;Your Data section
;AREA DATA
SUMP DCD SUM
N DCD 5
NUM1 DCD 3, -7, 2, -2, 10
POINTER DCD NUM1
        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 size of array -
; a counter for how many elements are left to process
     LDR R2, POINTER ; load base pointer of array
     MOV R0, #0 ; initialize accumulator

LOOP
     LDR R3, [R2], #4 ; load value from array,
; increment array pointer to next word
     ADD R0, R0, R3 ; add value from array to accumulator
     SUBS R1, R1, #1 ; decrement work counter
     BGT LOOP ; keep looping until counter is zero
     LDR R4, SUMP ; get memory address to store sum
     STR R0, [R4] ; store answer
     LDR R6, [R4] ; Check the value in the SUM

STOP
    B STOP
    END
```

Figure 4: Example1 code

2.1.2 Example1 result

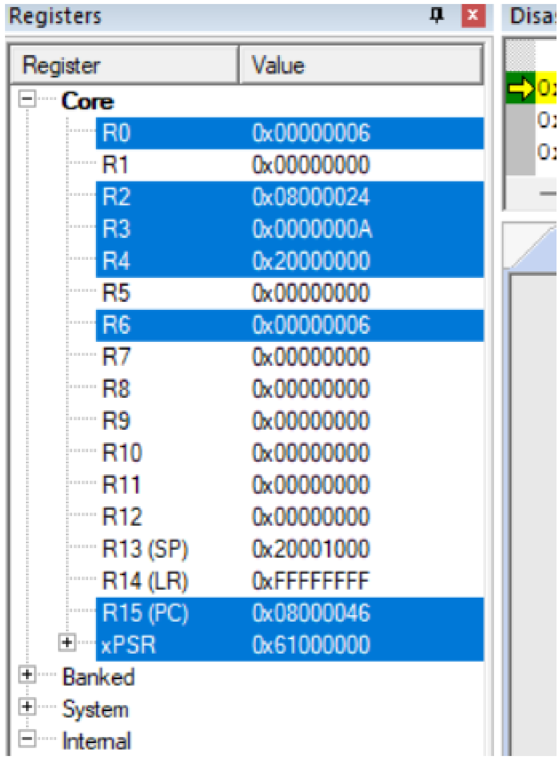


Figure 5: Example 1 result

2.1.3 Result discussion

This code is written in ARM assembly language and is designed to calculate the sum of elements in an array. At the beginning, it sets up a stack pointer and a reset vector for handling system resets. In the data section, it declares variables including an array of numbers (NUM1) and a pointer to this array. In the code section, it defines a reset handler that initializes necessary registers and starts a loop to iterate through the array. Within the loop, it loads each element from the array, adds it to an accumulator, and decrements a counter until all elements are processed. Finally, it stores the sum in memory and enters an infinite loop.
From the result the sum of array NUM1 is in R0 is 0x00000006.

2.2 Example#2

2.2.1 Example#2 code

The code is to count the length of the string and store the result in counter R1.

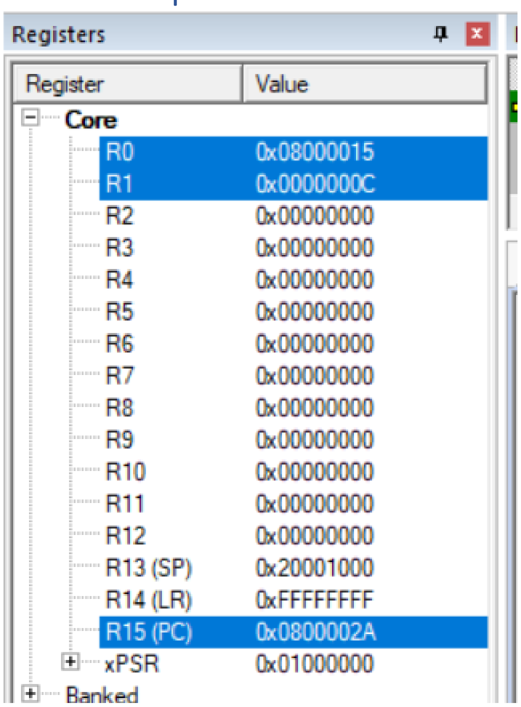
```
ex2.s
;;; Directives
        PRESERVE8
        THUMB
        AREA RESET, DATA, READONLY
        EXPORT __Vectors
__Vectors
      DCD 0x20001000 ; stack pointer value when stack is empty
        DCD Reset_Handler ; reset vector

        ALIGN
string1
          DCB "Hello world!",0

          AREA MYCODE, CODE, READONLY
          ENTRY
           EXPORT Reset_Handler
Reset_Handler
;;;;;;;;;;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], #1 ; Load the character from the address R0 contains
; and update the pointer R0
; using Post-indexed addressing mode
            CBZ R2, 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
         B countDone
         END ; End of the program
```

Figure 6: Example 2 code

2.2.2 Example#2 result


Registers

Register	Value
Core	
R0	0x08000015
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)	0x0800002A
xPSR	0x01000000
Banked	

Figure 7: Example 2 result

2.2.3 Result discussion

This ARM assembly code initializes a string "Hello world!" in memory and then calculates its length. The reset handler sets up the initial environment. The main functionality begins with loading the address of the string into register R0 and initializing a counter for the string's length in register R1. It enters a loop where it loads each character of the string using byte-wise loading, updating the address pointer, and incrementing the counter until it encounters a null terminator, indicating the end of the string.
The result above show that the length of string “Hello World” is 0x0000000C.

2.3 LabWork1

2.3.1. LabWork1 code

The code add up all the numbers that are greater than 5 in the number array NUM1 and store the answer in R0.

labwork1.s*

```
    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 7
NUM1 DCD 3, -7, 2, -2, 10, 20, 30
POINTER DCD NUM1

        AREA myCode, DATA, READWRITE

SUM DCD 0

    AREA MYCODE, CODE, READONLY
    ENTRY
    EXPORT Reset_Handler
Reset_Handler

    LDR R1 ,POINTER
    LDR R2,N ; R2=NUMBER OF ARRAY
    MOV R0,#0
LOOP
    LDR R3,[R1],#4
    CMP R3,#5
    BGT L1
    SUBS R2,R2,#1
    BGT LOOP
L1
    ADD R0,R0,R3
    SUBS R2,R2,#1
    BGT LOOP
    LDR R4, SUMP ; get memory address to store sum
    STR R0, [R4] ; store answer
    LDR R6, [R4] ; Check the value in the SUM
STOP
    B STOP
```

Figure 8: labwork1 code

2.3.2. LabWork1 result

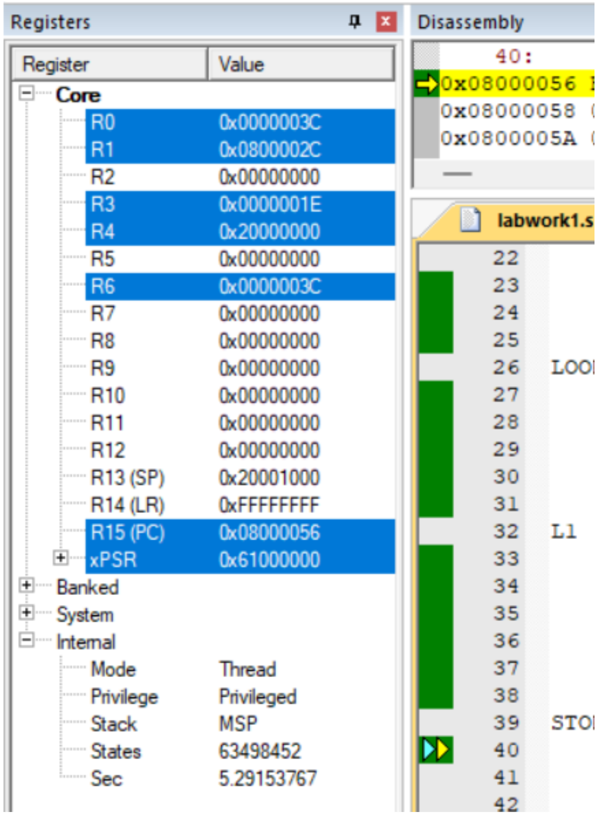


Figure 9 : labwork1 result

2.3.3. Result discussion

This ARM assembly code initializes an array of integers, **NUM1**, and calculates the sum of all elements in the array except for those that are greater than 5. It begins by loading the base address of the array into register R1 and the number of elements in the array into register R2. The accumulator register R0 is initialized to zero. Then, it enters a loop where it loads each element of the array and checks if it's greater than 5. If it's greater, it skips adding it to the accumulator. Otherwise, it adds it to the accumulator. After processing all elements, it stores the sum in memory. From the result R0 is 0x0000003C.

2.4 LabWork2

2.4.1 LabWork2 code

The code is to find the maximum value and the minimum value in the number array NUM1. And showing the Min in R5 and the Max in R6.

```

 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
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
; The program
; Linker requires Reset_Handler
    AREA MYCODE, CODE, READONLY
    ENTRY
    EXPORT Reset_Handler
```

```
Reset_Handler
 LDR R0, MaxP;
 LDR R1, MinP;
 LDR R2, N;
 LDR R3, POINTER;
 MOV R4, #0x80000000;
 MOV R5, #0; ;;min
 MOV R6, #0; ;;max
 LDR R5, [R3];
 LDR R6, [R3];
 MOV R10, #0xFFFFFFFF;
LOOP
 LDR R7, [R3];
 AND R8, R7, R4;
 CMP R8, R4;
 BEQ NEGATIVE1
 MOVGT R6, R7;
 B SKIP1
NEGATIVE1
 SUBEQ R7, R10, R7;
 ADDEQ R7, R7, #1;
 AND R8, R5, R4;
 CMP R8, R4;
 BEQ NEGATIVE2
 B SKIP2
SKIP2
 LDR R7, [R3];
 MOV R5, R7;
 B SKIP1
```

```
NEGATIVE2
 SUBEQ R9, R10, R5;
 ADDEQ R9, R9, #1;
 CMPEQ R7, R5;
 LDRGT R7, [R3];
 MOV R5, R7;
SKIP1 ADD R3, R3, #4;
 SUBS R2, R2, #1;
 BGT LOOP
HERE B HERE
ALIGN
END
 END
```

Figure 10: labwork2 code

2.4.2 LabWork2 result

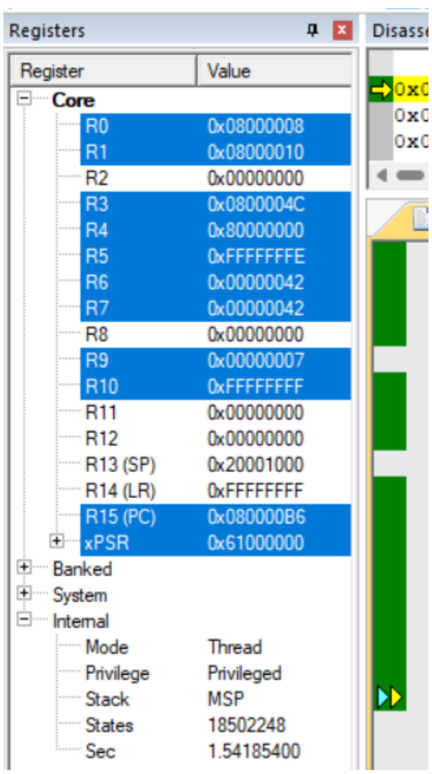


Figure 11: labwork2 result

2.4.3 Result discussion

This ARM assembly code aims to find the maximum and minimum values in an array of integers. It initializes variables to store the maximum and minimum values, as well as their pointers, and the number of elements in the array. Within the main routine, it loads necessary pointers and initializes registers. It then iterates through the array, examining each element. For each element, it checks if it's negative, updates the maximum or minimum accordingly, and moves to the next element. It utilizes conditional branching and bitwise operations to handle negative numbers appropriately.
From the result Min is 0xFFFFFFFE and Max is 0x00000042.

2.5 Todo

The above code is to determine if the first string is a substring from the second string.

```
        AREA    RESET, DATA, READONLY
           EXPORT  __Vectors

__Vectors  DCD  0x20001000     ; stack pointer value when s
           DCD  Reset_Handler  ; reset vector

        ALIGN
string1
           DCB "string",0

string2
           DCB "second_string",0

      AREA mycode, CODE, READONLY

      ENTRY
      EXPORT Reset_Handler
Reset_Handler


      LDR R0, =string1
      LDR R1, =string2

loop
  LDRB R2, [R0, #1]!
  LDRB R3, [R1, #1]!
  CMP R2, #97
  BLT skip1
  CMP R2, #122
  BGT skip1
  SUBS R2, R2, #32  ;convert to upper case

skip1
  CMP R3, #97
  BLT skip2
  CMP R3, #122
```

```
skip1
  CMP R3, #97
  BLT skip2
  CMP R3, #122
  BGT skip2
  SUBS R3, R3, #32  ;convert to upper c

skip2
  CMP R2, R3
  BNE not_part
  B loop

not_part
  MOV R0, #0

    END
```

Figure 12: Todo code

3.Conclusion

In this experiment, we practiced using different types of ARM Addressing Modes that help us make decisions in our programs. We learned each of these types Register Addressing Mode, Register Indirect Addressing Mode, ARM's Autoindexing Pre-indexed Addressing Mode, ARM's Autoindexing Post-indexing Addressing Mode, Program Counter Relative (PC Relative) Addressing Mode. Finally, we used our knowledge to solve the tasks.

4.References

[1] https://www.syncfusion.com/succinctly-free-ebooks/assemblylanguage/addressing-modes#:~:text=The%20immediate%20or%20literal%20addressing,the%20literal%20or%20immediate%20value.[Accessed on 23 March 2024]

[2] https://byjus.com/gate/register-indirect-addressing-mode-notes/.[Accessed on 23 March 2024]

[3] https://study.com/academy/lesson/addressing-modes-definition-types-examples.html. [Accessed on 23 March 2024]

[4] Lab manual, fourth experiment, ARM Addressing Mode. [Accessed on 23 March 2024]

[5] https://www.geeksforgeeks.org/difference-between-pc-relative-and-base-register-addressing-modes/.[Accessed on 23 March 2024]