Faculty of Engineering & Technology – Electrical & Computer Engineering Department

First Semester 2021 – 2022

COMPUTER DESIGN LABORATORY

ENCS4110.

Experiment No. 5 – ARM's Flow control instructions

Name: Ahmaide Al-Awawdah.

ID: 1190823

Section: 3

Instructor: Dr. Ayman Haroub

TA: Eng.Raha Zabadi

Date: 19th – 24th October 2021

1. Abstract

This Experiment will be done using Keli uVision5.

Objectives:

This experiment aims to explore and apply the branch instructions using the condition flags in ARM which can be used as strings and if statements in higher lever programming languages.

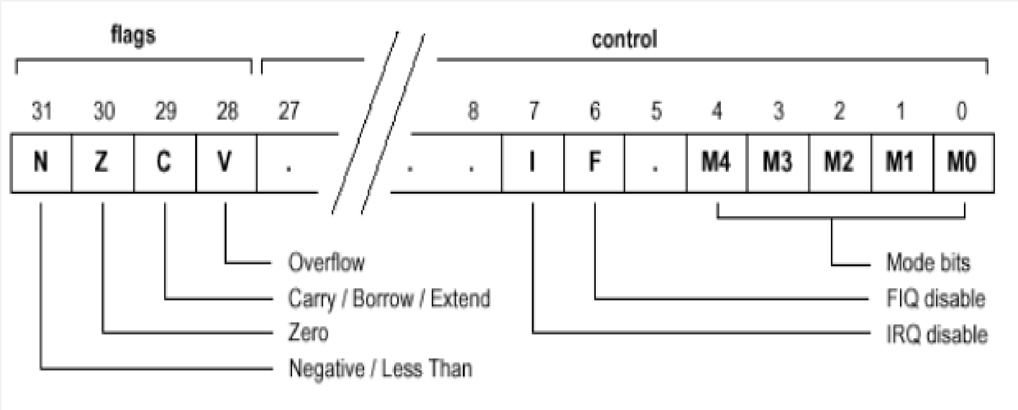


Figure 1: The Flags in ARM Registers [2]

❖Table of Content

1. Abstract 2
2. Theory 5
   2.1. Arm Register Set 5
   2.2. Condition Code Flags 5
   2.3. Branch Instructions 6
3. Procedure 9
   3.1. Example 1 9
   3.2. Example 2 10
   3.3. Lab Assignment 11
4. Conclusion 12
5. Appendix 13
   5.1. Appendix 1 13
   5.2. Appendix 2 14
   5.3. Appendix 3 15
6. References 17

❖Table of Figures

Figure 1: The Flags in ARM Registers [2]……………………………...………………………………...……2

Figure 2: ARM Registers…………………………………………………….……………………...……...…5

Figure 3: ARM ALU [4]…………………………………………………………………………...…………...6

Figure 4: Offset register……………………………………………………………………………...…......6

Figure 5: ARM instructions…………………………………………………………………………..……..8

Figure 6: The Difference Between the Registers In The Beginning & End Of The Loop Of Example 1….9

Figure 7: Memory Location Of the String In Example 1……………………………...…………………....9

Figure 8: The Difference Between the Registers In The Beginning & End Of The Loop Of Example 2...10

Figure 9: Memory Location Of The Output In Example 2……………………………...………...……10

Figure 10: Register values of the vowels and nun-vowels……………………………………...…....11

2. Theory

2.1. Arm Register Set

As ARM has 16 registers (R0 – R15), the first 13 are general-purpose registers as they can be used by the user, However the other registers (R13, R14, R15) are used for other things, R13 is used as a stack pointer, R14 is a link register and R15 is the program counter. [3]

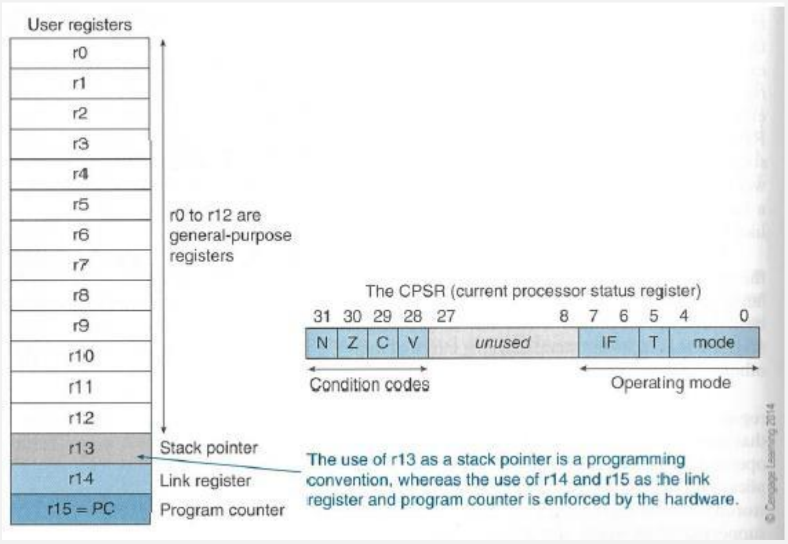


Figure 2: ARM Registers

2.2. Condition Code Flags

Flags are conditional bits which presents conditions coming from instructions after performing operations such as (adding, subtracting, comparing, etc.), these flags are the last 4 bits in the CPSR (current processor status register), and they are:

- Z: zero flag (when the output of an operation as zero or when two equal registers are compared)
- N: negative or less than flag (When the output is negative or when the first compared register has a value smaller than the second compared register)

- C: carry or barrow or extended flag (Used when the output needs to present an extra value of ‘1’ in binary)
- V: overflow flag (used when an overflow accrues in an operation)

Figure 3 below shows the ARM ALU the shows how the flags are set:

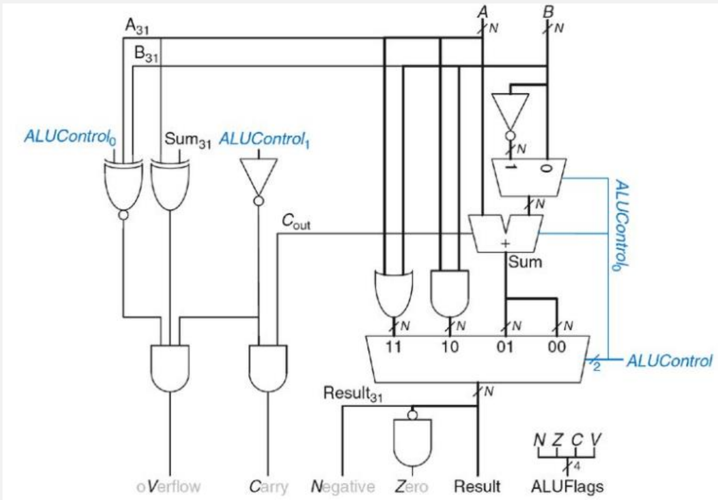


Figure 3: ARM ALU [4]

2.3. Branch Instructions

Branches are used in order to teleport to further or previous line or instruction in an ARM assembly code, conditional branch instructions contain a signed 24-bit offset as it is shown in figure 5.

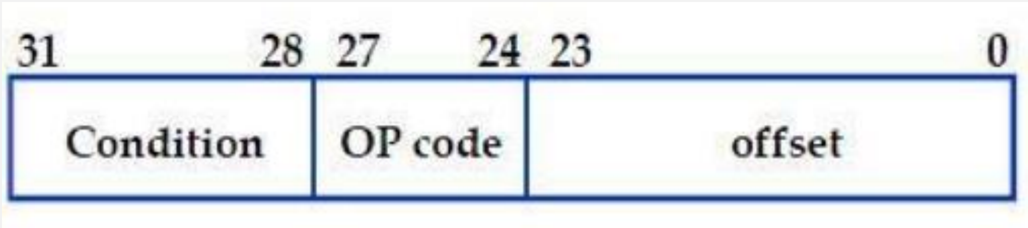


Figure 4: Offset register

Since the conditional branch changes the target, the offset updates the program counter in order to generate the new target's address, depending on the condition of the instruction as it will execute only if the flag's current conditions satisfy the condition from bit 28 – 31 in figure 5 above, otherwise it will go to the next instruction, this can create loops where the instructions can be done more than once as the conditions are satisfies so when it reaches the end of the loop it will see check the flag condition and teleport to the beginning of the loop.

Here are all the ARM conditions with their flags in Table 1:

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	

Table 1: Conditions Flags

The list below shows ARM control and branch instructions:

```
------------------------------------------------------------------------
     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.
------------------------------------------------------------------------
```

Figure 5: ARM instructions

.

3. Procedure

3.1. Example 1

The code can be found in Appendix 1

In this code Register R2 stores temporarily the ASCII in binary of the string character, where R0 is pointer to the memory location of the String and R1 is a counter that counts the number of the characters in the string, when R0 reaches the 0 after the end of the string it jumps to the (countDone) instruction, while loopCount is to keep looping while R2 isn't zero, figure 7 shows the difference of the ARM registers after the first and last loop, figure 8 shows the stored string ASCII in the memory.

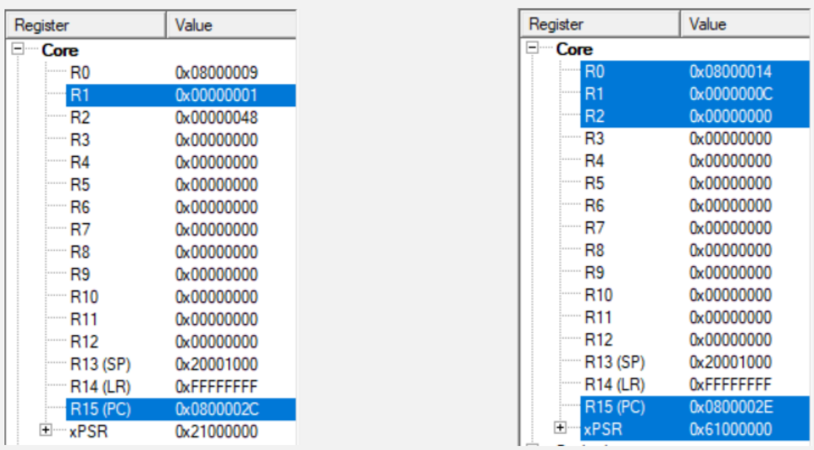


Figure 6: The Difference Between the Registers In The Beginning & End Of The Loop Of Example 1

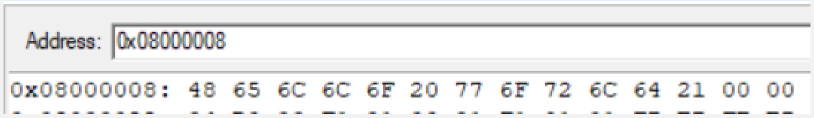


Figure 7: Memory Location Of the String In Example 1

As shown in figure 7, R1 contains the number of characters in "Hello world!" which is 12.

3.2. Example 2

The code can be found in Appendix 2

This example does the equation $\sum_{k=0}^{N}(N-k)$, where N =5, so the output will be 15 (F in hex), the value is stored to R1 and then its decremented each time after being added to R0 which is initially 0, the loop (Loop) ends when the value of R1 equals 0 then the wanted location's address in the memory is stored to R3 (SUMP) in order to store in it the output value in R0, figure 9 shows the changes of the registers before and after the loop, figure 10 shows the memory location of the output (SUMP).

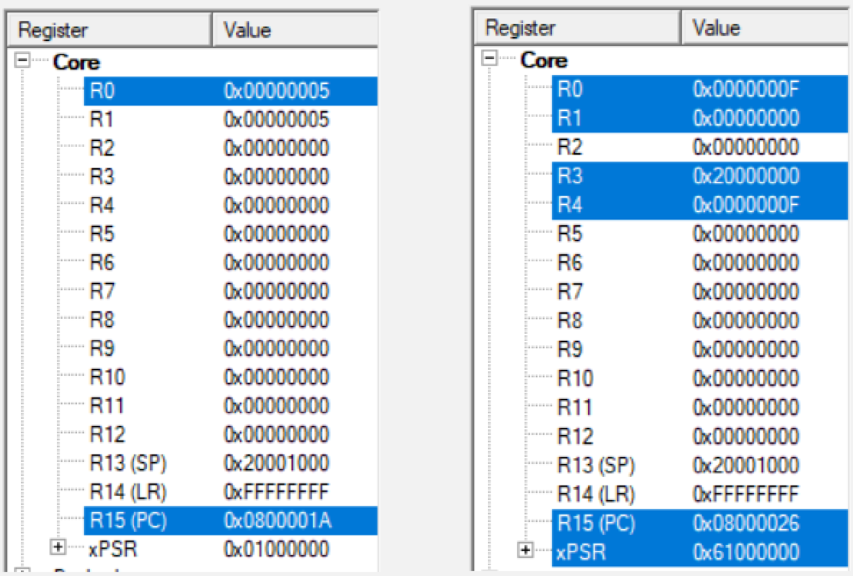


Figure 8: The Difference Between the Registers In The Beginning & End Of The Loop Of Example 2

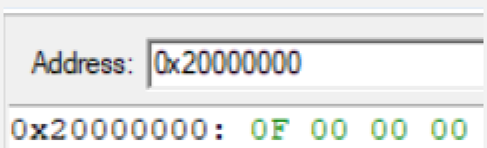


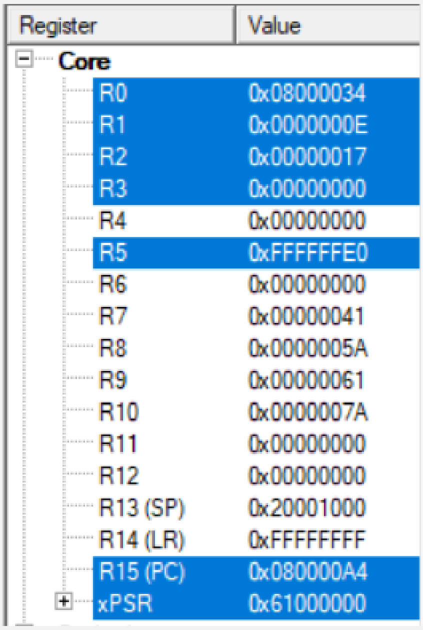
Figure 9: Memory Location Of The Output In Example 2

As shown in figure 10 the output is F which is equal to 15 in decimal = 5 + 4 + 3 + 2 + 1

3.3. Lab Assignment

The code can be found in Appendix 3

This code requires to count the number of vowels and nun- vowels in a string, in order to count the vowels and save their amount in register R1, a Z branch is called in each time the value of the register R3 that stores the string ASCII is compared with a vowel where it will be compared with all capital and small vowels, if z =0 it will branch to just increase the value of R1 and then restart the loop, in case R3 wasn't an ASCII of a vowel it will be checked to see if its letter or not by seeing if its ASCII between (65-90) as a Capital letter, or between (97- 122) as a small letter, if its neither then it will skip the add 1 to R2 instruction and branch to the end of the loop where the pointer R0 is increased and loading the value into R3, the main loop that keeps checking the characters will end when R0 reaches the address of 0 after the string and the 0 will be stored in R3 and since it will be compared with 0, the z flag will equal 1 and the BNE branch wont teleport to the beginning of the loop which means that it will reach the end as the job is done and R1 will have the number of the vowels and R2 will have the nun-vowels, figure 11 shows the values in the registers after the loop is over.



Register	Value
Core	
R0	0x08000034
R1	0x0000000E
R2	0x00000017
R3	0x00000000
R4	0x00000000
R5	0xFFFFFFE0
R6	0x00000000
R7	0x00000041
R8	0x0000005A
R9	0x00000061
R10	0x0000007A
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x080000A4
xPSR	0x61000000

Figure 10: Register values of the vowels and nun-vowels

4. Conclusion

In conclusion it was noticed that there are 4 flags that control looping, and redoing and skipping in ARM assembly (Zero, Negative, Overflow, Carry) these flags are very useful in order to make large operations in the memory, and to do multi instructions more than once, they can also be very helpful in denying other instruction that cannot be wanted depending on the situation that they will be in, Branches are very useful in making loops and making if statements and going pass arrays with the use of simple conditions in ARM assembly.

5.Appendix

5.1. Appendix 1

```
;The semicolon is used to lead an inline documentation
;
;When you write your program, you could have your info at the top document block
;For Example: Your Name, Student Number, what the program is for, and what it does
etc.
;
; This program will count the length of a string.
;
;;; 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 empty
            DCD Reset_Handler ; reset vector

            ALIGN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Byte array/character string
; DCB type declares that memory will be reserved for consecutive bytes
; You can list comma separated byte values, or use "quoted" characters.
; The ,0 at the end null terminates the character string. You could also use "\0".
; The zero value of the null allows you to tell when the string ends.
;
; The DCB directive allocates one or more bytes of memory, and defines the initial
; runtime contents of the memory.
;
; Example
; Unlike C strings, ARM assembler strings are not null-terminated.
; You can construct a null-terminated C string using DCB as follows:
; C_string DCB "C_string",0
;
;*****************************************************************************
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
```

5.2. Appendix 2

```
;The semicolon is used to lead an inline documentation
;When you write your program, you could have your info at the top document block
;For Example: Your Name, Student Number, what the program is for, and what it does etc.
;
; See if you can figure out what this program does
;
;;; 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 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 ;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. Appendix 3

```
;The semicolon is used to lead an inline documentation
;
;When you write your program, you could have your info at the top document block
;For Example: Your Name, Student Number, what the program is for, and what it does etc.
;
; This program will count the length of a string.
;
;;; 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 empty
            DCD Reset_Handler ; reset vector

            ALIGN

string1     DCB "ARM assembly language is important to learn!",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
        MOV R7, #65
        MOV R8, #90
        MOV R9, #97
        MOV R10, #122
        LDRB R3, [R0]
Loop
        CMP R3, #'A'
        BEQ vowel
        CMP R3, #'E'
        BEQ vowel
        CMP R3, #'I'
        BEQ vowel
        CMP R3, #'O'
        BEQ vowel
        CMP R3, #'U'
        BEQ vowel
        CMP R3, #'a'
        BEQ vowel
        CMP R3, #'e'
        BEQ vowel
        CMP R3, #'i'
        BEQ vowel
        CMP R3, #'o'
        BEQ vowel
        CMP R3, #'u'
        BEQ vowel
        SUBS R5, R8, R3
        BMI notCapital
        SUBS R5, R3, R7
        BMI No
        ADD R2, R2, #1
        B No
notCapital      SUBS R5, R10, R3
        BMI No
        SUBS R5, R3, R9
        BMI No
        ADD R2, R2, #1
        B No
vowel   ADD R1, R1, #1
No      ADD R0, R0, #1
        LDRB R3, [R0]
        CMP R3, #0
        BNE Loop

STOP
        B STOP
        END
```

6. References

[1]. Exp3_ARM Flow Control Instructions. Oct 19th at 7:22 PM

[2]. https://smist08.wordpress.com/2019/12/02/arm-processor-modes/psr/ Oct 19th at 7:25 PM

[3]. https://www.cs.uregina.ca/Links/class-info/301/ARM-control/lecture.html/ OCT 23rd at 10:35 PM

[4]. http://slideplayer.com/slide/17804675/ OCT 23rd at 11:21 PM