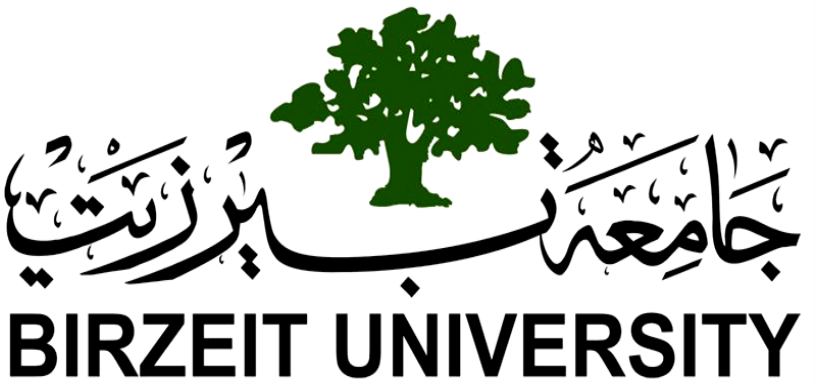

Faculty of Engineering & Technology Electrical & Computer Engineering Department

Computer Design Laboratory ENCS 4110

Experiment #7

ARM's Flow Control Instructions

Prepared by:
Hasan Hamed 1190496

Instructor: Dr. Abualseoud Hanani
Assistant: Eng. Moutasem Diab

Section: 1
Date: 14/8/2021

1. Abstract

The aim of the experiment is to understand how to deal with strings in ARM assembly language, and to understand the importance of conditions flags, and to learn ARM branch and compare instructions and how to use them for loops and conditional statements, and finally to create useful ARM assembly programs using all of those concepts.

Table of Contents

2. Theory

ARM's Flow Control Instructions modify the default sequential execution. They control the operation of the processor and sequencing of instructions. ARM has 16 programmer-visible registers and a Current Program Status Register, CPSR as shown below.

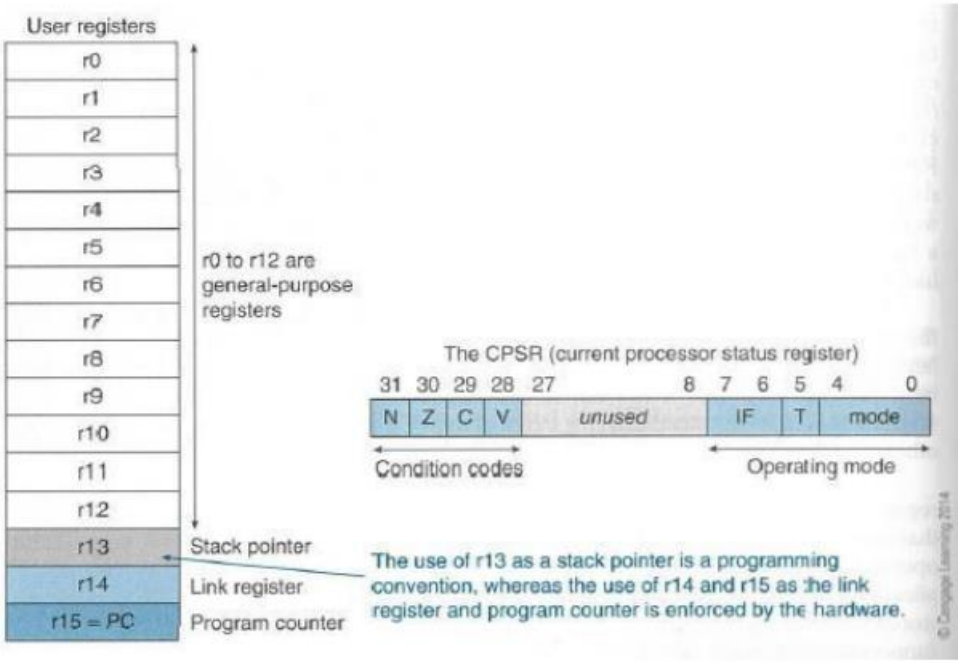

Figure 2.1: ARM Register Set

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.

Condition code flags in CPSR:

N - Negative or less than flag

Z - Zero flag

C - Carry or borrow or extended flag

V - Overflow flag

The least-significant 8-bit of the CPSR are the control bits of the system. The other bits are reserved.

2.1. 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.

2.2. 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.

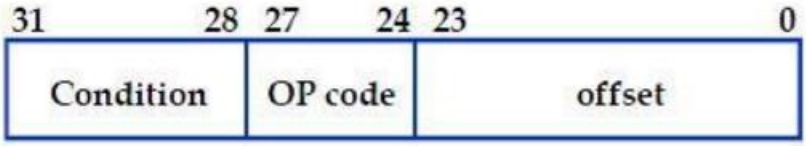

Figure 2.2: Encoding Format for Branch Instruction

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.

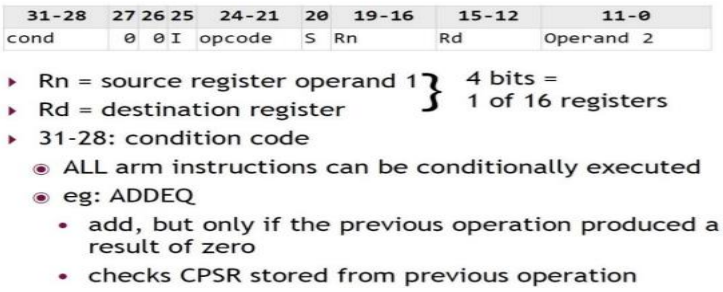

Figure 2.3: ARM Instruction

All the ARM instructions are conditionally executed depending on a condition specified in the instruction bits 31-28.

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	
			(U) = unsigned

Figure 2.4: ARM Conditions and Flags Affected

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.

2.3. 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.[1]

```
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 2.5: Examples of Branch Instruction

The Compare instruction can also be used as Branch by putting the condition after CMP instruction as shown below.

```
Mnemonic                  Meaning
CBZ R5, target          ; Forward branch if R5 is zero
CBNZ R0, target         ; Forward branch if R0 is not zero
CMP R2, R9              ; R2 - R9, update the N, Z, C and V flags
CMN R0, #6400           ; R0 + #6400, update the N, Z, C and V flags
CMPGT SP, R7, LSL #2    ; update the N, Z, C and V flags
```

Figure 2.6: Examples of Compare Instruction

3. Procedure

3.1. String Length Counter Program

1) The following code was put in Keil Uvision.

```
;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
```

Figure 3.1: Assembly Code to Count the Length of a String

2) The target was built using build target button and the debugger was started as shown below.

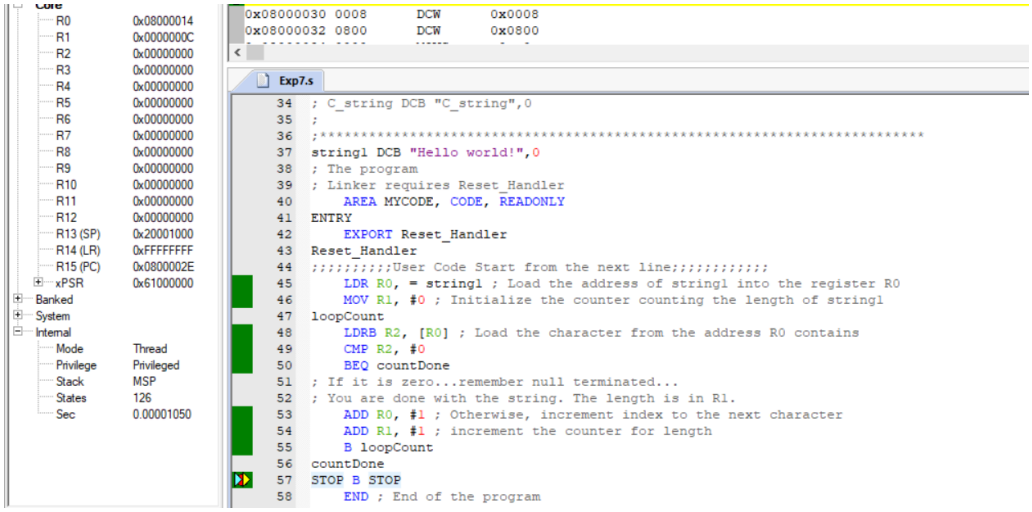

Figure 3.2: Debugging the Code

The program will count the length of the string and will save the number of characters in R1, and because the string in assembly is not null terminated the zero was used to stop counting if the string address pointer (R0) reaches the address of the number zero, after that the value of the register containing the character (R2) will become zero and the program will branch to finish.

3.2. Sum of Integers between 5 And 0 Program.

1) The following code was put to Keil Uvision program.

```
;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
```

Figure 3.3: Program Code

2) The target was built using the Build button and the debugger was started as show below.

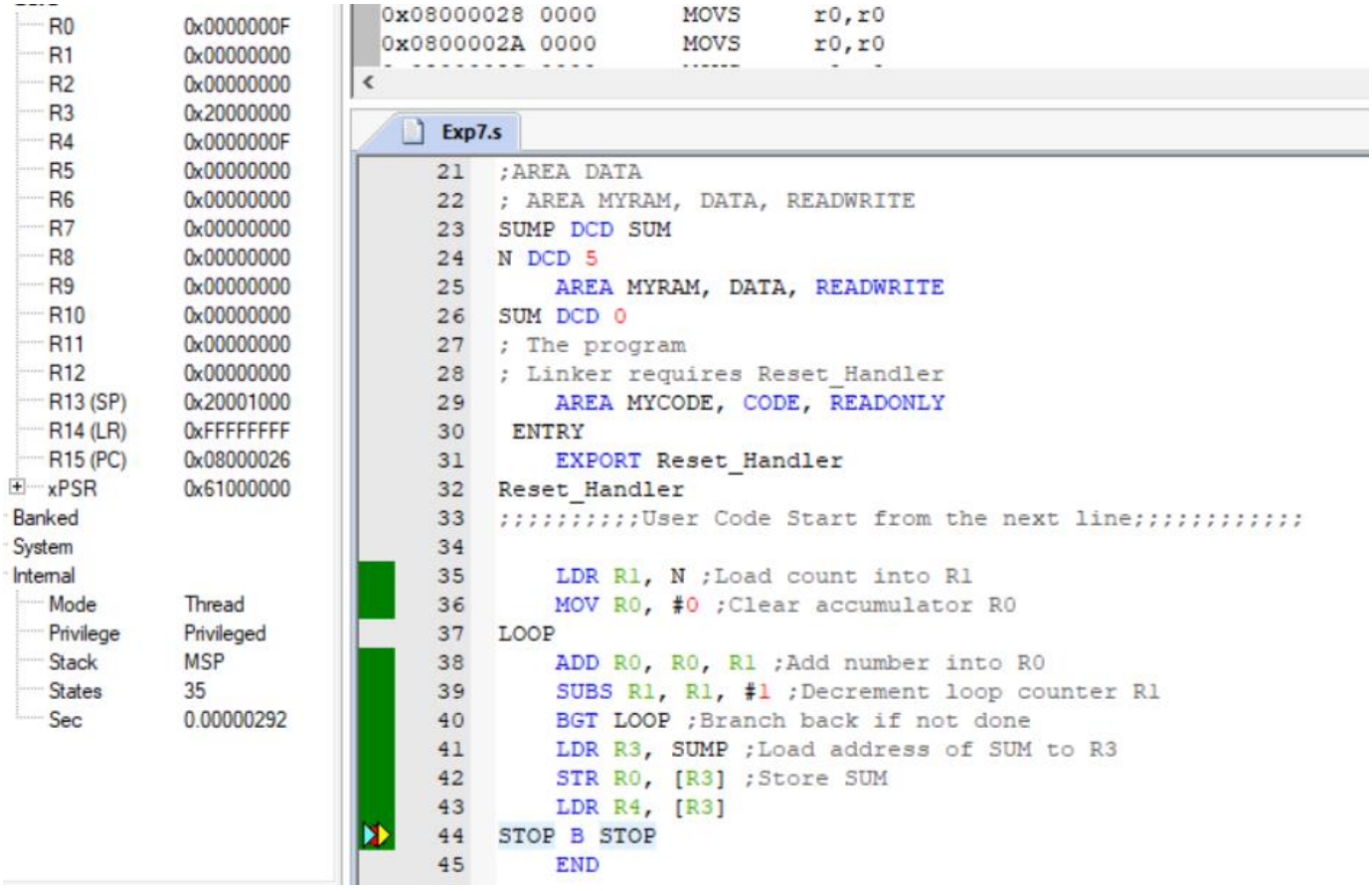

Figure 3.4: Program Debugging

The program will count the integers between n and 0 and in this case the number is 5 so the program will add (5+4+3+2+1) and will save it in R0, after that the sum will be stored in the memory at the address that sum pointer pointing at, and finally it will load R4 with the sum that is stored previously in the memory.

4. Conclusion

In conclusion, we understood how to deal with strings in ARM assembly, and we understood that the strings in assembly are not null terminated as in C or JAVA or other high level languages, and we understood the importance of Branch and Compare instructions and how to use them to make loops and conditional statements (If … else , while loop), and finally we used all those concepts to make programs such as counters.

5. References

[1] Computer Design Laboratory Manual.