Experiment No 5

ARM assembly programming using control flow Instructions

Objectives:

- To learn the basic of various flow control Instructions in ARM Architecture
- Use these flow control instructions to perform conditional and repetitive tasks
- Operation of user defined subroutines in ARM Assembly

Flow control, or branching, instructions for jumps, loops and conditional jumps allow us to control the execution of program statements and to build high-level programming structures such as if-else statements, case statements, and loops in assembler programs.

CMP Instruction:

The CMP (Compare) instruction compares two operands and effects the flags only. CMN(Compare Negative) is also a similar instruction but it is used to compare with Negative numbers. The syntax of CMP instruction is as follows:

CMP{cond} Rn, Operand2

where:

cond

is an optional condition code.

Rn

is the ARM register holding the first operand

Operand2

is a flexible second operand

The instruction compares the value in a register with 'Operand2'. They update the condition flags on the result, but do not place the result in any register. The CMP instruction subtracts the value of 'Operand2' from the value in 'Rn'. This is the same as a SUBS instruction, except that the result is discarded. In ARM Assembly, CMP instruction is used before conditional execution of instructions or conditional branch. The flags (N, Z, C, V) are only affected by this instruction

TEQ Instruction:

The TEQ (Test Equivalence) instruction compares two operands by doing the 'AND' operation between two 32-Bit registers and effects the flags only. The syntax of TEQ instruction is as follows:

TEQ{cond} Rn, Operand2

where:

cond
is an optional condition code.

Rn
is the ARM register holding the first operand

Operand2
is a flexible second operand

The instruction compares the value in a register with 'Operand2'. They update the condition flags on the result, but do not place the result in any register. The TEQ instruction performs 'AND' operation between 'Rn' and 'Operand2'. In ARM Assembly, TEQ instruction is used before conditional execution of instructions or conditional branch. The flags (N, Z) are only affected by this instruction.

Conditions in ARM Assembly:

ARM, like many other architectures, implements conditional execution using a set of flags whichstore state information about a previous operation. The simplest way to set the condition flags is to use a comparison operation, such as CMP. This mechanism is common to many processor architectures, and the semantics (if not the details) of CMP will likely be familiar. In addition, we have already seen that many instructions (such as SUB in the example) can be modified to update the condition flags based on the result by adding an 'S' suffix.

Conditional execution can be done through:

1. Conditional Instructions.
2. Branch Instructions.

Conditional Instructions:

All the ARM Assembly instructions can be executed by adding a suffix to the instruction.

For Example:

```
CMP     R2, #0
ADDEQ   R2, R2, #50     ; Add if TRUE
MULEQ   R5, R5, #8      ; Multiply if TRUE
LSRNE   R8, R8, #4      ; Shift Right if TRUE
SUBLE   R3, R3, #5      ; Subtract if TRUE
```

These instructions will only execute if the condition is satisfied.

Figure 1: Suffix for Conditional Instruction

Suffix	Description	Flags tested
EQ	Equal (Unsigned & Signed)	Z=1
NE	Not Equal (Unsigned & Signed)	Z=0
MI	MInus (Negative)	N=1
PL	PLus (Positive or Zero)	N=0
VS	oVerflow Set	V=1
VC	oVerflow Clear	V=0
HI	Unsigned HIgher	C=1 & Z=0
LS	Unsigned Lower or Same	C=0 or Z=1
GE	Signed Greater or Equal	N=V
LT	Signed Less Than	N!=V
GT	Signed Greater Than	Z=0 & N=V
LE	Signed Less than or Equal	Z=1 or N!=V
AL	ALways	

EXAMPLE:

```
1       EXPORT SystemInit
2       EXPORT __main
3
4           AREA  myData, DATA, READWRITE
5   Array2 DCD 0,0,0,0,0
6
7           AREA     MDATA1, DATA, READONLY
8   Array1 DCD 4,-5,9,-10,8
9
10          AREA    MYCODE, CODE, READONLY
11
12  SystemInit FUNCTION
13      BX       LR
14
15      ENDFUNC
16
17  __main FUNCTION
18
19      LDR      R0,=Array1
20      LDR      R1,=Array2
21      mov      R2, #0
22
23      LDR      R4,[R0],#4
24      CMP      R4, #0
25      SUBLT    R4, R2, R4
26      STR      R4,[R1],#4
27
28      LDR      R4,[R0],#4
29      CMP      R4, #0
30      SUBLT    R4, R2, R4
31      STR      R4,[R1],#4
32
33      LDR      R4,[R0],#4
34      CMP      R4, #0
35      SUBLT    R4, R2, R4
36      STR      R4,[R1],#4
37
38      LDR      R4,[R0],#4
39      CMP      R4, #0
40      SUBLT    R4, R2, R4
41      STR      R4,[R1],#4
42
43      LDR      R4,[R0],#4
44      CMP      R4, #0
45      SUBLT    R4, R2, R4
46      STR      R4,[R1],#4
47
48      ;===================================
49
50
51      ENDFUNC
52      END
53
```

Branch Instructions:

Branch instructions are divided into further two types:

1. Unconditional Branch Instructions
2. Conditional Branch Instructions

Unconditional Branch (B) is followed by a 'Label' and it is always executed and theprogram execution jumps to the label.

For Example:

B Last Line

B SKIP

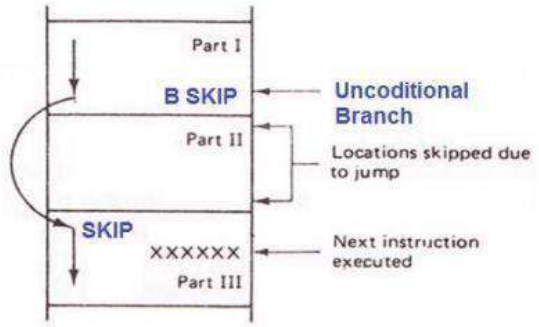


Figure 2: Flow of Instructions when using Unconditional Branch

Conditional Branch (B) is followed by a condition and a 'Label' and it is only executedwhen the condition is TRUE and the program execution jumps to the label.

For Example:

CMP R2, #0

BLE LastLine

BGE SKIP

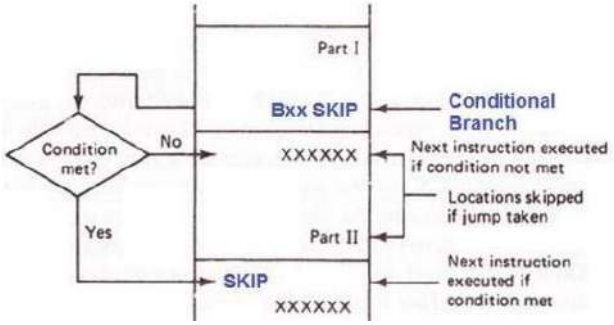


Figure 3: Flow of Instructions when using Conditional Branch

These instructions will only execute if the condition is satisfied.

Compare	Signed	Unsigned
==	BEQ	BEQ
!=	BNE	BNE
>	BGT	BHI
>=	BGE	BHS
<	BLT	BLO
<=	BLE	BLS

Figure 4: Some common Conditional Branch Instructions

Assembly Implementation of Loops:

Loop instruction jumps to the specified label until zero flag is not equal to zero. Loop structure usually consists of a condition statement and repetition structure.

There are three types of loops in High Level Languages like C:

- For Loop
- While Loop
- Do While Loop

Any task that has to be performed repeatedly will use loop instruction. Advantage of using loop instruction is that it will save your time and effort as well as the memory of the system in performing a certain repeated task.

Example:

```
    EXPORT SystemInit
    EXPORT __main

        AREA  myData, DATA, READWRITE
Average DCD 0

        AREA    MDATA1, DATA, READONLY
Array1 DCD 4,5,9,10,3,1,6,7,15,13

        AREA   MYCODE, CODE, READONLY

SystemInit FUNCTION
    BX      LR

    ENDFUNC

__main FUNCTION

    LDR     R0,=Array1
    LDR     R1,=Average

    mov     R2, #0      ;running sum
    mov     R3, #10     ;Loop Count

START

    LDR     R4,[R0],#4
    ADD     R2, R2, R4
    SUB     R3, R3, #1

    CMP     R3, #0
    BGT     START

    MOV     R3, #10
    SDIV    R2, R2, R3
    STR     R2,[R1]

    ;================================


    ENDFUNC
    END
```

User-Defined Subroutines in Assembly:

A program is made up of instructions which implement the solution to a problem. In a given program, it is often needed to perform a particular sub-task many times on different data values. So, we split the program into smaller units which solve a particular part of the problem. These task specific sub-units are termed as functions, procedures or subroutines. In assembly language, we use the word subroutine for all subprograms to distinguish between functions used in other programming languages and those used in assembly languages. The block of instructions that constitute a subroutine can be included at every point in the main program when that task is needed. However, this would result in unnecessary waste of memory space. Rather, only one copy of the instructions that constitute the

subroutine is placed in memory and can be accessed repeatedly.

Syntax of assembly language procedure is given below:

```
sample FUNCTION
    .
    .
    BX LR
ENDFUNC
```

Figure 5: Structure of User-Defined Subroutine

- FUNCTION & ENDFUNC directives mark the start and end of procedure.
- An Assembly Program may have one procedure or multiple procedures to do small tasks.

The program must follow these six steps:

- Put function arguments in a place where subroutine can access them
- Transfer control to the subroutine
- Acquire the storage for the procedure
- Perform the desired task
- Put the result in a place where main function can access
- Return control to the point of origin

Subroutine Call:

In ARM assembly, Branch and Link (BL) Instruction is used to call the subroutine:

BL MyFunction ; Call instruction

On execution of instruction, return address is copied into link register (R14) and program flowjumps to the start of subroutine.

- **Link Register = PC + 4 ; LR = R14, PC = R15**

And the program counter (R15) is updated accordingly. On subroutine call, the 'BL' instruction returns the address of instruction after 'BL' is copied to link register (R14) and Program Counter (R15) is updated with address of first instruction of subroutine.

Examples:

- **BL AddNum**
- **BL Factorial**
- **BL Subroutine**

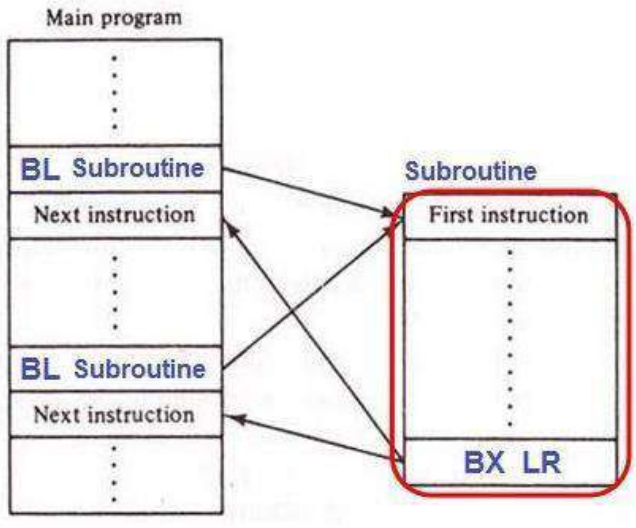


Figure 6: Working of a Subroutine

You should be able to call the subroutine from anywhere.

Returning from Subroutine:

In ARM Assembly, once the subroutine is complete, it should return back to the place that calledthe subroutine. To return from subroutine, branch (BX) instruction is used:

```
BX      LR      ; return from subroutine (LR = R14)
```

On execution of this instruction, the return address contained in link register (R14) moves toprogram counter (R15) and program flow jumps to point of origin to return from subroutine.

The following instruction can also be used in ARM assembly to return from the function:

```
MOV   PC, LR          ; return from subroutine (PC = R15 , LR = R14)
```

As discussed above, there are special instructions for transferring control to subroutines and restoring control to the main program. The instruction that transfers the control is usually termed as call, jump or branch to subroutine. The calling program is called Caller and the subroutine called is known as Called. The instruction that transfer control back to the caller is known as Return.

Conventions for Subroutine parameters and return value:

All parameter values are placed in R0, R1, R2, R3 by parent procedure before calling the child procedure. Extra arguments, if any, will be pushed onto Stack. child procedure will use the parameters to compute the return value. Before returning, child procedure will place return value in R0. The parent procedure will assume return value from child procedure in R0 register.

Example:

```
MOV     R0, Val1
MOV     R1, Val2
BL      addnums
```

Example Code:

```
1      EXPORT SystemInit
2      EXPORT __main
3
4      AREA  myData, DATA, READWRITE
5   ANSWER DCD 0
6
7      AREA MDATA1, DATA, READONLY
8   VAL1 DCD 10
9   VAL2 DCD 20
10
11     AREA    MYCODE, CODE, READONLY
12
13  SystemInit FUNCTION
14     BX      LR
15     ENDFUNC
16  ;  Average Subroutine
17
18  AVGFUNC FUNCTION
19
20     ADD  R0, R0, R1
21     MOV  R2, #2
22     SDIV  R0, R0, R2
23     BX LR
24
25     ENDFUNC
26  __main FUNCTION
27
28     LDR  R0,VAL1
29     LDR  R1,VAL2
30     LDR  R4,=ANSWER
31
32     BL AVGFUNC
33     STR  R0,[R4]
34
35     ENDFUNC
36     END
```

Lab Task (Psychomotor)

Activity 01:

Write an Assembly Program that will load marks of a student from memory & compute Grade according to the scheme shown below:

Marks >= 85	**A Grade**
Marks >= 70	**B Grade**
Marks >= 60	**C Grade**
Marks < 60	**F Grade**

The computed grade will be stored in the memory.

Activity 02:

Write an assembly Function/ Subroutine to find Maximum of three numbers. Three numbers will be passed as input argument to the function and the function will return the Maximum. Call the function from main function. Main Function will load the three numbers from memory and will also stores the Maximum in memory.

Lab Task (Cognitive)

Question 1:

Consider the following ARM program fragment:

```
      MOV   R0, #5
      MOV   R1, #0
loop  ADD   R1, R1, R0
      SUBS  R0, R0, #1
      BNE   loop
```

a) Trace the execution and determine the final values of R0 and R1. **[3 Marks]**

b) State how many times the loop body executes and which status flag is tested by BNE. **[3 Marks]**

c) Write the instruction pair that branches to the label HIGH when the signed value in R0 is greater than 10. **[4 Marks]**

Lab Exercise and Summary

Summary should cover Introduction, Procedure, Data Analysis and Evaluation.

Student's Signature: _____ **Date: _____**

LABORATORY SKILLS ASSESSMENT (Cognitive)

Obtained Marks	**/10**

Instructor's Signature: _____ Date: _____

LABORATORY SKILLS ASSESSMENT (Psychomotor)

Psychomotor Activity 1: Experimental Setup

S. No	**Criteria**	**Performance**	**Description**	**Marks**
1	**Keil Assembly Setup (4)**	Level 4	Correctly prepares the required Keil assembly setup for the experiment independently and according to the laboratory requirements.	4
		Level 3	Correctly prepares the required Keil assembly setup with minor errors or assistance.	3
		Level 2	Partially prepares the required Keil assembly setup and requires considerable assistance.	2
		Level 1	Shows limited ability to prepare the required Keil assembly setup and requires substantial assistance.	1
2	**Program Structure & Subroutine Setup (4)**	Level 4	Correctly prepares the required program structure and subroutine setup for the experiment independently and according to the laboratory requirements.	4
		Level 3	Correctly prepares the required program structure and subroutine setup with minor errors or assistance.	3
		Level 2	Partially prepares the required program structure and subroutine setup and requires considerable assistance.	2
		Level 1	Shows limited ability to prepare the required program structure and subroutine setup and requires substantial assistance.	1
3	**Debug & Breakpoint Parameters (2)**	Level 2	Correctly sets the required debug and breakpoint parameters for the experiment.	2
		Level 1	Sets the basic debug and breakpoint parameters with minor errors or assistance.	1
Total Obtained				**/10**

If a student is absent:

Criteria	**Description**	**Marks**
Absent	Student is absent from the laboratory activity.	**0**

Psychomotor Activity 2: Implementation + Results

S. No	Criteria	Performance	Description	Marks
1	**Branch, Loop & Subroutine Implementation (4)**	Level 4	Correctly implements the assigned branch, loop and subroutine program independently while following appropriate laboratory practices.	4
		Level 3	Implements the assigned branch, loop and subroutine program with minor errors or assistance and generally follows proper laboratory practices.	3
		Level 2	Partially implements the assigned branch, loop and subroutine program and requires considerable assistance or repeated guidance.	2
		Level 1	Implementation of the assigned branch, loop and subroutine program is largely incomplete, requires substantial assistance, or shows poor adherence to proper laboratory and safety practices.	1
2	**Execution & Debugging (2)**	Level 2	Correctly performs the required execution and debugging and verifies the expected operation.	2
		Level 1	Performs the basic execution and debugging with minor errors or assistance.	1
3	**Control-Flow & Result Analysis (4)**	Level 4	Obtains correct control-flow and result analysis and appropriately verifies and interprets the required experimental results.	4
		Level 3	Obtains mostly correct control-flow and result analysis with minor errors in verification or interpretation.	3
		Level 2	Obtains partially correct control-flow and result analysis with noticeable errors or omissions.	2
		Level 1	Control-flow and result analysis is largely incorrect or incomplete and requires substantial assistance.	1
Total Obtained				**/10**

If a student is absent:

Criteria	Description	Marks
Absent	Student is absent from the laboratory activity.	**0**

Instructor's Signature: _____ Date: _____

LABORATORY SKILLS ASSESSMENT (Affective)

Affective Activity: Lab Report + Viva Voce

S. No	Criteria	Performance	Description	Marks
1	**Lab Report Completeness & Organization (4)**	Level 4	Report is complete, well-organized, properly presented, and demonstrates responsible completion of the assigned laboratory work.	4
		Level 3	Report is complete with minor omissions or organizational issues and demonstrates satisfactory responsibility for the assigned work.	3
		Level 2	Report is partially complete with several omissions and demonstrates limited responsibility for the assigned work.	2
		Level 1	Report has major omissions or poor organization and demonstrates minimal responsibility for the assigned work.	1
2	**Control-Flow Output & Conclusion (2)**	Level 2	The control-flow output and conclusion are correctly presented and discussed, demonstrating ownership of the assigned work.	2
		Level 1	The control-flow output and conclusion are presented with minor errors or omissions.	1
3	**Viva Voce (4)**	Level 4	Clearly explains the assigned experiment, branch instructions, loops, subroutines, program explanation, working, and results, demonstrating individual responsibility, participation, and effective communication.	4
		Level 3	Explains most of the assigned experiment and results correctly, with minor difficulty, and demonstrates good individual participation.	3
		Level 2	Explains basic aspects of the assigned experiment with some prompting and demonstrates basic participation.	2
		Level 1	Demonstrates limited understanding or involvement and requires substantial prompting to explain the assigned work.	1
Total Obtained				**/10**

The student has not submitted the report and has not appeared in the Viva Voce.

Criteria	Description	Marks
Lab Report and Results Not Submitted	Student has not submitted the report and results file.	**0**
Absent in Viva Voce	Student is absent from the viva voce.	**0**

Instructor's Signature: _____ Date: _____