Khalifa University for Sciences Technology and Research

Department of Electrical and Electronic Engineering

Module Name: Microprocessor Systems Laboratory
Module Code: ELCE333

Laboratory Experiment No. 2

Expriment Tite: Development & Testing of HCS12 Programs Using Branching and Loops

Group Members

Name: Afra Bin Fares, ID#:100033139
Name: Alya Humaid AlAlili ,ID#100037087
Name:Anoud Alshamsi ,ID#100035514

Instructors

Mohammed Ali Saif Al Zaabi
Mahmoud Khonji

Spring 2015

Table of Contents

Abstract 4

1. Introduction 4

2. Design and Results 6

Task1: If-Then-Else Statement using BEQ 6

Task2: If-Then-Else Statement using BNE 7

Task3: Nested If-Then-Else Statements 8

Task4: Simple Program with Loops 9

4. Conclusion & Recommdation 10

5. Assignment Questions 11

List of tables and figures

List of tables :

Table1: If-Then-Else Statement using BEQ values and results

Table2 If-Then-Else Statement using BNE values and results

Table3 Nested If-Then-Else Statements values and results

List of figures :

Figure 1: Flow chart of an example on simple Branch instruction

Figure 2: Nested If-Then-Else Statements flow chart

Figure 3: task 4 flow chart

Abstract

The laboratory of experiment two will continue to introduce the assembly instruction sets and and it will focus on decision making as well as flow control instructions such as data test. Moreover, this report will include the results of developing and testing the HCS12 microprocessor using branching and loops. Also, the laboratory experiment is divided into four tasks. The first task is about using the If-Then-Else Statement with the BEQ instruction in the program .In the second task, it is very similar to the first task but we need to use the BNE instead of BEQ instruction. However, the third task the Nested If-Then-Else statements need to applied. Finally, task four is about a simple program need to be designed with loops for summing five numbers.

1. Introduction

This laboratory experiment concentrate on a different branch types in order to change execution flow in assembly codes. These are the different types of branch instructions :

- **Short conditional** branch uses 8 bit relative addressing with range of -128 or +127 bytes from the instruction following the branch otherwise the assembler program will give an (parameter out of range) error.
- **Long conditional** branch has a 16 bit relative addressing. Moreover branch instructions distinguish between signed and unsigned number comparison.

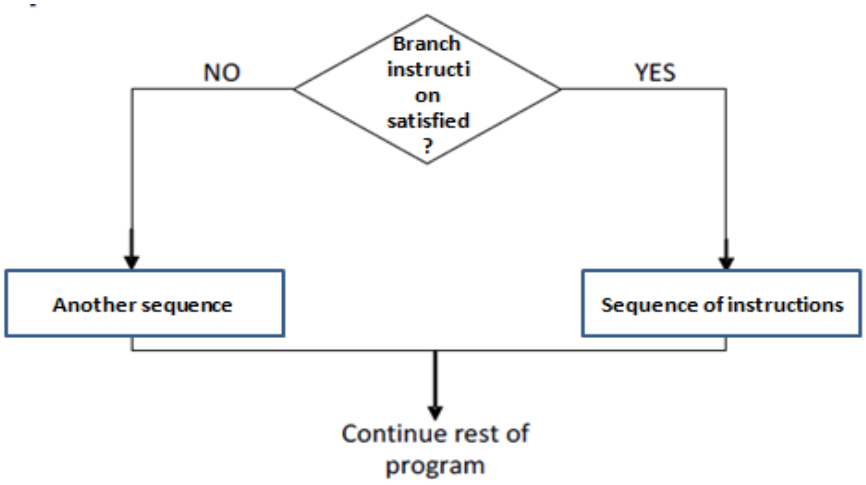


Figure 1: Flow chart of an example on simple Branch instruction

- **Stack pointer** is also a type of branching used in this laboratory experiment. A stack is a LIFO (last in, first out) which means the first one which will be popped is the last entry you push on the stack and this structure is used for holding stack frames. In addition, in order to hold the address of the last stack location we need to use Stack Pointer (SP).The SP is initialized at beginning of the code, and also it grows downward from the pointed address where it get incremented every time a byte when is pulled from it and decremented every time when a byte is pushed into it.

1.1. Aim:

This laboratory experiment aims to familiarize students with the design, development and testing of HCS12 assembly programs with conditional branching structures and loops.

1.2. Objectives:

After finishing the laboratory experiment the student should know how to:

1- Plan flow-charts containing branching and loops.

2- Apply flow-charts using HCS12 assembly code.

3- Use branch instruction to implement branching and loops.

4- Download, run, and test code on a Dragon Plus Trainer board.

2. Design, Results and Analysis

In this part of the report the tasks performed in the experiment will be discussed by explaining the steps of these tasks, listing, explaining and analyzing the results obtained from this experiment. The experiment is divided into four tasks. The first task is about using the BEQ instruction in If-Then-Else statement. In the second the program used in task1 will be modified by replacing the BEQ instruction with BNE. The third part is about writing a nested If-Then-Else stating program .And in the final part a program with loops was created and assembled to perform a specific task.

2.1. TASK-1: If-Then-Else Statement using BEQ

In this part of the experiment it was required to edit, assemble and step though the following program:

```
; Include derivative-specific definitions
        INCLUDE ' derivative.inc'
X1 EQU $1000 ;X1 is associated with the memory location $1000
Y1 EQU $1001  ;Y1is associated with the memory location $1001
Y2 EQU $1002 ;Y2 is associated with the memory location $1002
; export symbols
    XDEF Entry
; Insert here your data definition.
ORG $4000 ;Flash ROM address for Dragon12+
Entry:
   MOVB #$0,$1000  ;immediate value to memory byte move the program was simulated twice using 0 and 3 for X
   MOVB #$A1,$1001 ; immediate value to memory byte move
   MOVB #$A2,$1002 ; immediate value to memory byte move

CLRA ; clear the content in accumulator A by assigning it to zero
CMPA X1 ; compare zero in accumulator A  with X1
 BEQ Eqzero ; if X1 is zero, go to "equalzero"
MOVB Y1,X1 ; else then  assign X1=Y1 and store it in X1
BRA Exit ; go to "exit", bypass "equal zero"
Eqzero MOVB Y2,X1 ; when X1=0 then make X1=Y2
 Exit BRA Exit ; End your code here
```

Using the following sets of data assigned for each memory location (shown in the table below) ;where the value of X was zero at the first test , and a non zero value at the second test; the program was simulated and the results shown in the table were obtained after the end of the program:

Table1: If-Then-Else Statement using BEQ values and results

	Before Program Start			After Program End		
	$1000	$1001	$1002	$1000	$1001	$1002
Test 1	0	A1	A2	A2	A1	A2
Test 2	3	A1	A3	A1	A1	A2

2.2. TASK-2 : If-Then-Else Statement using BNE

In this task, the program used in task 1 was modified by using the BNE instead of BEQ, and the following program was created, assembled and stepped though:

```
; Include derivative-specific definitions
        INCLUDE ' derivative.inc'
X1 EQU $1000 ;X1 is associated with the memory location $1000
Y1 EQU $1001  ;Y1is associated with the memory location $1001
Y2 EQU $1002 ;Y2 is associated with the memory location $1002
; export symbols
    XDEF Entry
; Insert here your data definition.
ORG $4000 ;Flash ROM address for Dragon12+
Entry:
   MOVB #$0,$1000  ;immediate value to memory byte move the program was simulated twice using 0 and 3 for X
   MOVB #$A1,$1001 ; immediate value to memory byte move
   MOVB #$A2,$1002 ; immediate value to memory byte move

CLRA ; clear the content in accumulator A by assigning it to zero
CMPA X1 ; compare zero in accumulator A  with X1
 BNE Eqnonzero ; if X1 is not zero, go to "equalzero"
MOVB Y1,X1 ; else then  assign X1=Y1 and store it in X1
BRA Exit ; go to "exit", bypass "equal zero"
Eqnonzero MOVB Y2,X1 ; when X1=0 then make X1=Y2
 Exit BRA Exit ; End your code here
```

Using the following sets of data assigned for each memory location (shown in the table below) ;where the value of X was zero at the first test , and a non zero value at the second test; the program was simulated and the results shown in the table were obtained after the end of the program:

Table2 If-Then-Else Statement using BNE values and results

	Before Program Start			After Program End		
	$1000	$1001	$1002	$1000	$1001	$1002
Test 1	0	A1	A2	A1	A1	A2
Test 2	3	A1	A2	A2	A1	A2

2.3. TASK-3: Nested If-Then-Else Statements

In this part of the experiment it was required to write a program to carry out the algorithm shown in figure2 below, and make our own decision of where the variables are initially stored, their initial values, and where the results are stored at.

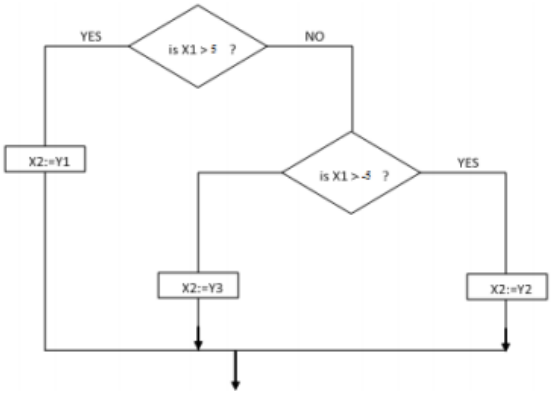


FIGURE 2: Nested If-Then-Else Statements flow chart

The following program was created, assembled and stepped though in order to perform the algorithm needed to be executed:

```
; Include derivative-specific definitions
        INCLUDE 'derivative.inc'
X1 EQU $1000 ;X1 is associated with the memory location $1000
Y1 EQU $1001 ;Y1 is associated with the memory location $1001
Y2 EQU $1002;Y2 is associated with the memory location $1002
Y3 EQU $1003 ;Y3 is associated with the memory location $1003
; export symbols
    XDEF Entry
; Insert here your data definition.
 ORG $4000 ;Flash ROM address for Dragon12+
Entry:
   MOVB #$FA,$1000 ; immediate value to memory byte move this value will be changed according to the conditions given
   MOVB #$A1,$1001; immediate value to memory byte move
   MOVB #$A2,$1002; immediate value to memory byte move
   MOVB #$A3,$1003; immediate value to memory byte move
   CLRA ; clear the content in accumulator A by assigning it to zero
   LDAA #$05 ; load accumulator A with the immediate value 5
   CMPA X1 ; compare 5 with X1
   BLT Eqfive ; if 5 is less than X1
   LDAA #$FB ; load accumulator A with the immediate value of -5 ( in decimal which is equivalent to FB in hex)
   CMPA X1  ; compare -5 with X1
   BLT Eqnegfive; if -5 is less than X1
   MOVB Y3,X1 ; if not ,i.e. X1 is less than -5 them make X1=Y3
   BRA Exit ; go to "exit", bypass "equal zero"
Eqnegfive MOVB Y2,X1  when X1 is more than -5 then make X1=X2
   BRA Exit ; go to "exit", bypass "equal zero"
Eqfive MOVB Y1,X1 ; when X1 is more than 5 then make X1=Y1
Exit BRA Exit ; End your code here
```

Using the following sets of data assigned for each memory location (shown in the table below) , the program was simulated and the results shown in the table were obtained after the end of the program:

Table3 Nested If-Then-Else Statements values and results								
	Before Program Start				After Program End			
	$1000	$1001	$1002	$1003	$1000	$1001	$1002	$1003
Test 1	6	A1	A2	A3	A1	A1	A2	A3
Test 2	0	A1	A2	A3	A2	A1	A2	A3
Test3	-6(FA in hex)	A2	A2	A3	A3	A1	A2	A3

2.4. TASK-4: Simple Program with Loops

In this part of the experiment we are required to find the sum of five numbers which are stored in the memory.

The program should do the following by the same order:

1) Initializing a counter to indicate how many numbers there are, which is implemented using a data register.

2) Initializing the sum to zero so that we can add the numbers accumulatively to this sum, that is implemented using a second data register.

3) Adding data from memory to the sum while decrementing the counter by 1 each time.

4) When the counter reaches zero, this means that all the numbers have been used and the program can finish. At this point the sum will be moved from the data register to memory for storage.

The following flow chart below describes this loop used to add a list of five numbers:

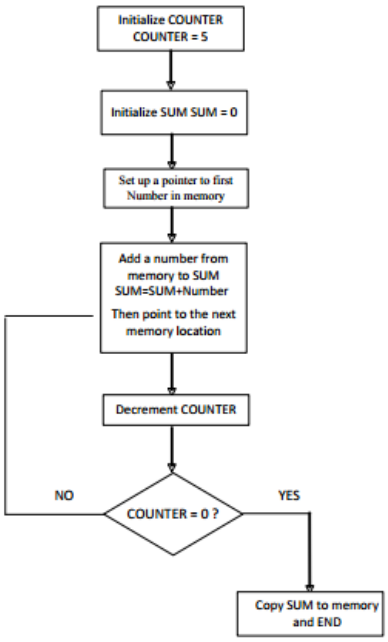


Figure 3: task 4 flow chart

The following program was created, assembled and stepped though in order to perform the algorithm needed to be executed:

```
; Include derivative-specific definitions
        INCLUDE 'derivative.inc'
; export symbols
    XDEF Entry
X1 EQU $1008 ;X1 is associated with the memory location $1008 here where the final value will be stored at
; Insert here your data definition.
NUMBERS DC.B $12,$1a,$43,$15,$28
 ORG $1000 ;Flash ROM address for Dragon12+
Entry:
   CLRA; clear the content in accumulator A by assigning it to zero
   CLRB; clear the content in accumulator B by assigning it to zero
   LDY #$0000 ;load index to register Y this represents the sum
   LDX #NUMBERS ;load index to register X which is NUMBERS
   LDAA #5 ; load accumulator A with the immediate value 5
LOOP   LDAB 1,X+ ; load accumulator B by the next value of numbers
   ABY ;add accumulator B to index register Y
   DECA ; decrement A
   CMPA #0 ; compare A to the immediate value 0
   BEQ eqz ; branch if A is equal to zero
   BRA LOOP   ;go to loop if not
eqz STY X1  ;store index register Y
EXIT BRA EXIT ; end here
```

3. Conclusion and Recommendation

We are able to achieve all the goals of the Laboratory experiment two and we noticed the reason behind designing a flow charts that's contain branch and loops before the implementation thats because it does helps to understand and implement the assembly code in the correct way and then run the program . Finally, we end up this experiment with grateful result and without facing any problems.

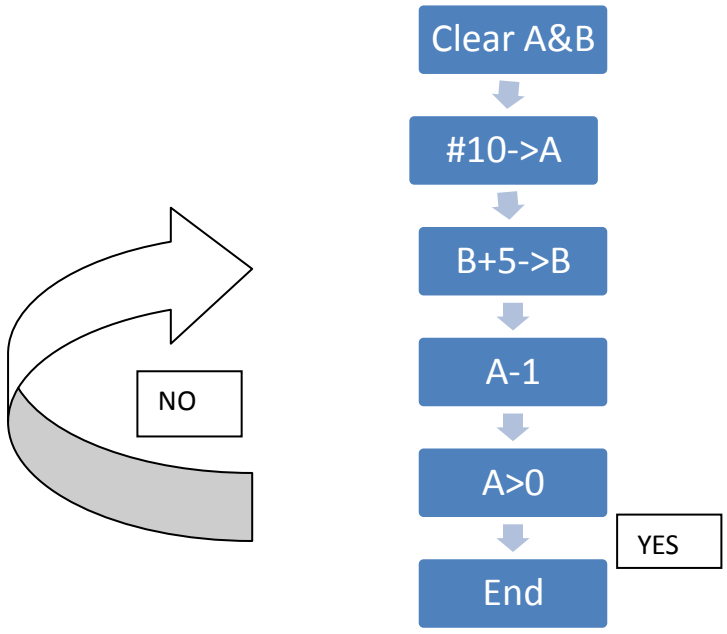
4. Assignment Questions

1. Write a program to clear the accumulator B, and then add 5 to Acc B 10 times using the loop concept. Use the zero flag and BNE with DECA. Draw the program flowchart.

Code:

```
; Include derivative-specific definitions
  INCLUDE 'derivative.inc'
  ORG $4000 ;Flash ROM address for Dragon12+
Entry:
      CLRA
      CLRB
      LDAA #10
LOOP ADDB  #5
      DECA
      BNE LOOP
EXIT    BRA EXIT
     END
```

Diagram:



Comment: The accumulator B was $30 = 50 in decimal