MicroprocessorSystemsLaboratory(ELCE333)

LaboratoryExperimentNo.2

MARKINGSHEET

StudentNamesandIDNo's: **Amal AlQasimi 100036830**
Shaikha Albesher 100037064

LaboratorySection: **Wednesday** **ExperimentDate:** **February 4, 2015**

No.	**Criteria**	**Description**	**Weight %**	**Mark**	**Comments**
1	Pre-lab	Amark willbeallocated toeach studentthatreflectshis preparationsforthelab.	20		
2	Performance Inthelab	Amark will beallocated toeach student individually that reflects hisperformanceinthelab	10		
3	Results and Analysis	Documentation andanalysisofthe resultsforeachtaskperformed in thelab	30		
4	Summary/ Conclusions	Conclusions for each task performedinthelab	10		
5	Assignment Questions	Answerstoassignmentquestions	20		
6	Report Presentation	Overall presentationofthe reportincludingproperlayoutand clarity of figures,tables,andgraphs. CorrectuseofEnglishlanguage.	10		
	Total		**100**		

ELCE 333: Microprocessor Systems Laboratory

Lab Report- ExperimentNo.2

Experiment Title: Development & Testing of HCS12 Programs Using Branching and Loops

Completed by:

Amal Al Qasimi (100036830)

Shaikha Al Besher (100037064)

Lab instructors:

Mahmoud Khonji

Mohammed Al Zaabi

Spring 2015

Table of Contents

Summary

The following lab report will include the details of the second Microprocessor System's Laboratory lab session. It will first start off by stating the aims and objectives of the tasks assigned to us throughout the lab session. Following the aims and objectives, the report will discuss the purpose of each task separately with the results obtained upon the completion of the task. Based on the observations of the tasks completed, the analysis and interpretation section will include the explanation of why we obtained the results we did. Finally the report will end with a brief conclusion. The assignment questions given to us as part of the lab are answered after the lab repot.

1. Introduction

The aim of this lab was to further introducing us to the concepts in assembly instruction sets and in particularly focused on decision making as well as flow control instructions. The decision making instructions dealt with included data test and conditional branch. Data test is responsible for testing data in registers or memory and conditional branch is responsible for testing the CCR bits to branch within the code.Conditional execution in assembly language is accomplished by several looping and branching instructions. These instructions can change the flow of control in a program. Branch instructions are divided into two categories short conditional branch and long conditional branch. Short conditional branch instruction use a 8 bit relative addressing, this means that the branch must be in the range of -128 or +127 bytes from the instruction following the branch otherwise an error message reading “parameters out of range” will be given by the assembler program. The second category, long conditional branch, has a 16 bit relative addressing.

2. Aims and Objectives:

The following are the aims and objectives of the first Microprocessor Systems report:

Aim: This experiment aim is to be more familiar in the design, development and testing of HCS12 assembly programs with conditional branching structures and loops.

Objectives: **Upon the completion of this experiment we were expected to be able to:**

1. Create flow-charts containing branching and loops.
2. Execute flow-charts using HCS12 assembly code.
3. Apply branch instruction to execute branching and loops.
4. Download, run, and test code on a Dragon Plus Trainer board.

3. Lab tasks

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

The first task aims to restate the If Condition by using branch instruction (BEQ). Then testing the program using 2 data sets using the CodeWarrior IDE simulator to verify that the program fulfill the requirements. The program will check wither the X1=1 or not, accordingly it will follow different case shown in the flow chart below:

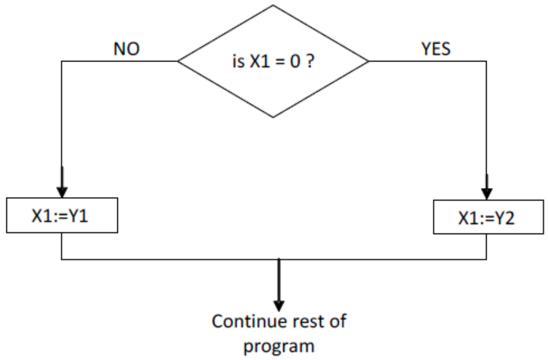


Figure 1: Flowchart of If-then-else statement using BEQ

The assembly code used to perform the operations shown in the flow chart above is:

```
; Include derivative-specific definitions
        INCLUDE 'derivative.inc'

; export symbols
        XDEF Entry
        ; we use export 'Entry' as symbol. This allows us to
        ; reference 'Entry' either in the linker .prm file
        ; or from C/C++ later on

        XREF __SEG_END_SSTACK      ; symbol defined by the linker for the end of the stack


X1 EQU $1000
Y1 EQU $1001
Y2 EQU $1002
            ORG $4000 ; Flash ROM address for Dragon12+
Entry:
    CLRA
    MOVB #$0,X1
    MOVB #$A1,Y1
    MOVB #$A2,Y2
    CMPA X1 ; compare zero with X1

BEQ Eqzero ; if X1 is zero, go to "Eqzero"
    MOVB Y1,X1 ; when X1 is not zero then X1=Y1
    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
HERE JMP HERE
```

After testing the code using two different sets, we tested the memory locations of X1 ($1000), Y1 ($1001), and Y2 ($1002) manually to verify the results, results are listed in table 1:

Table 1: outcome when X1=0 and when X1≠0 using BEQ instrcution

	Before Program Start			After Program End		
	$1000	**$1001**	**$1002**	**$1000**	**$1001**	**$1002**
Test 1 (X1=0)	00	A1	A2	A2	A1	A2
Test 2 (X1≠0)	2	A1	A2	A1	A1	A2

Conclusion:

The test proved that the code is working correctly. From the table above it is clear that when X1=0 the value of Y2 is assigned to X1. However, when X1≠0 the value of Y1 is assigned to X1.

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

In the second task, we modified the code used in task by using BNE instruction instead of BEQ and repeated the same steps as task 1.

The modified code is:

```
; Include derivative-specific definitions
        INCLUDE 'derivative.inc'

; export symbols
        XDEF Entry
        ; we use export 'Entry' as symbol. This allows us to
        ; reference 'Entry' either in the linker .prm file
        ; or from C/C++ later on

        XREF __SEG_END_SSTACK     ; symbol defined by the linker for the end of the stack

X1 EQU $1000
Y1 EQU $1001
Y2 EQU $1002
          ORG $4000 ; Flash ROM address for Dragon12+
Entry:
    CLRA
    MOVB #$0,X1
    MOVB #$A1,Y1
    MOVB #$A2,Y2
    CMPA X1 ; compare zero with X1

    BNE Nzero ; if X1 is NOT zero, go to "Nzero"
    MOVB Y2,X1 ; when X1 is zero then X1=Y2
    BRA Exit ; go to "exit", bypass "Nzero"
Nzero MOVB Y1,X1 ; when X1 not zer then make X1=Y1
Exit BRA Exit ; End your code here
HERE JMP HERE
```

After testing the code using two different sets, we tested the memory locations of X1 ($1000), Y1 ($1001), and Y2 ($1002) manually to verify the results, results are listed in table 2:

Table 2: outcome when X1=0 and when X1≠0 using BNE instruction

	Before Program Start			After Program End		
	$1000	**$1001**	**$1002**	**$1000**	**$1001**	**$1002**
Test 1 (X1=0)	00	A1	A2	A2	A1	A2
Test 2 (X1≠0)	02	A1	A2	A1	A1	A2

Conclusion:

The results generated are exactly in using BNE instruction are exactly same as the one generated task 1.

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

In this task we were asked to write a program that handle nested if-then algorithm which is the flow chart below.

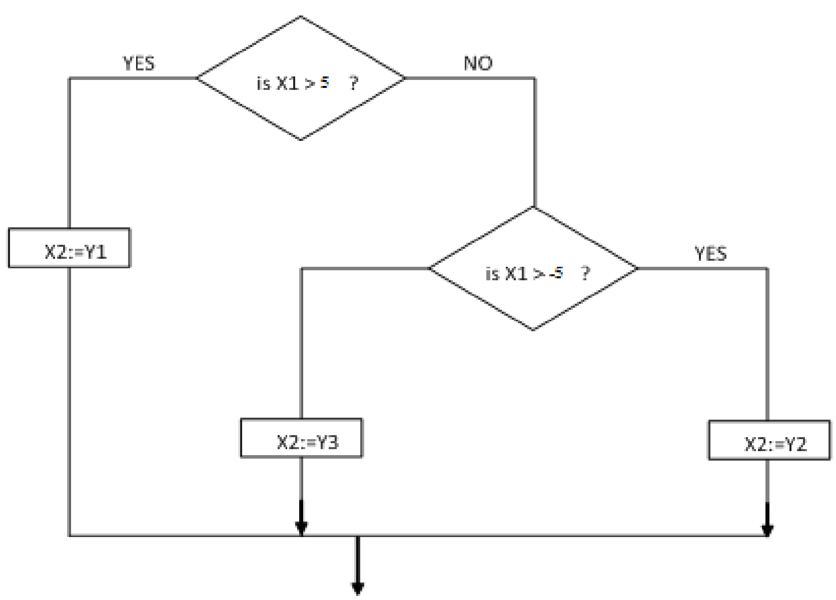


Figure 2: Nested if else flow chart

The program will simply have five variables which are X1, X2, Y1, Y2, and Y3. It will start by checking wither X1 is greater or less than 5. If it is less than 5 then it will check if it is greater or less than -5, if it is greater the program then will assign Y2 to X2 otherwise it will assign Y3 to X2. And if X1 greater than 5 then Y1 will be assigned to X2.

The code below was developed using branch instructions to perform the requirement:

```
; Include derivative-specific definitions
        INCLUDE 'derivative.inc'
; export symbols
        XDEF Entry
        ; we use export 'Entry' as symbol. This allows us to
        ; reference 'Entry' either in the linker .prm file
        ; or from C/C++ later on

        XREF __SEG_END_SSTACK     ; symbol defined by the linker for the end of the stack
X1 EQU $1000
X2 EQU $1001
Y1 EQU $1002
Y2 EQU $1003
Y3 EQU $1004
 ORG $4000                         ;Flash ROM address for Dragon12+
Entry:
     CLRA
     CLRB
     MOVB #$A1,Y1
     MOVB #$A2,Y2
     MOVB #$A3,Y3
     LDAA #$3
     CMPA X1                              ; compare 5 with X1
     BLE GREAT                  ; if X1 is greater than 5, go to GREAT
     NEGA
     CMPA X1                              ; compare X1 with -5
     BLE GREAT2    ; if X1 greater than -5 got to GREAT2
     MOVB Y3,X2                            ; if none of the conditions satisfied do X2:=Y3
     BRA Exit                              ; go to "exit"
GREAT  MOVB Y1,X2                         ; when X1›5  then make X2:=Y1
     BRA Exit                   ; exit bypass "GREAT"
GREAT2 MOVB Y2,X2                          ;when X1›-5 then make X2:=Y2
Exit   BRA Exit                            ; End of nested if-else
HERE   JMP HERE
```

To verify that the program is written correctly, we tested on the 3 cases, where X1>5,5> X1>-5, and X1 < -5, the results are shown in the table below:

Table 2:Task3 results

	Initial testing values					Final values				
	X1	**X2**	**Y1**	**Y2**	**Y3**	**X1**	**X2**	**Y1**	**Y2**	**Y3**
Case 1 X1 >5	10	1	2	3	4	10	2	2	3	4
Case 2 5>X1 > -5	-1	1	2	3	4	-1	3	2	3	4
Case 3 X1<-5	-7	1	2	3	4	-7	4	2	3	4

Conclusion:

The provided flow chart ease the understanding of creating a program that runs if-else statements. Basically, the program works as visualized in the flow chart and checks two statements, and works nested to assign the right value regarding the results. Table 3 demonstrates how the program works in the three different cases; theshaded cells are identical in each row, indicating that the assignment of value follows the coding.

4. Task 4:Simple Program with Loops

Branching instructions can be used to implement loops in assembly language. The aim of this task is to write a simple program that sum 5 numbers which are stored in the memory. We started the code by initializing a counter to indicate how many numbers are there and store it in the register A. then we initialized sum to zero in order to use it later to store the sum. Then we started adding data from the memory to the sum and decrease the counter accordingly. If the counter reaches zero this means that all numbers have been added to sum and the program cans end. At this stage, the sum can be moved from the data register to memory for storage.The flow chart below describes the loop used to add a list of five numbers.

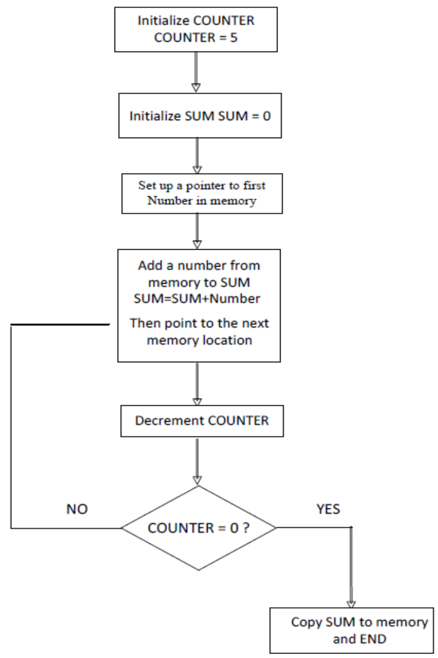


Figure 3: Counter Loop flow chart

The following code demonstrates the loop shown in the flow chart above:

```
; Include derivative-specific definitions
        INCLUDE 'derivative.inc'

; export symbols
        XDEF Entry
        ; we use export 'Entry' as symbol. This allows us to
        ; reference 'Entry' either in the linker .prm file
        ; or from C/C++ later on

        XREF __SEG_END_SSTACK      ; symbol defined by the linker for the end of the stack


      ORG $4000 ;Flash ROM address for Dragon12+
Entry:
      CLRA
      CLRB
      LDX #NUMBERS             ; loading register X with the numbers initialized in NUMBERS
      LDAA #5         ;loading register A with number
LOOP    ADCB 1,X+                 ; (B)+(M) ? (B) and then point to the next memory location
      DECA                              ; decrement A
      CMPA #0           ; compare A with zero
      BEQ EQZERO      ; if A=0, move to branch EQZERO
      BRA LOOP        ; if not, go to branch LOOP
EQZERO  STAB X                  ; if A=0, store the value of B in register X
EXIT    BRA EXIT
      ORG   $1000
NUMBERS DC.B $12,$1A ,$43 , $15 , $28
```

Conclusion:

The program is developed using loops, and the data were stored in memory locations $1000-$1004. The pointer is used to point the next element to be added to the sum and accordingly the counter will decrement, once the counter reaches zero the sum that was stored temporarily in register B ($1005) will be moved to register X. The following figure shows the values stored in memory from location $1000 to $1004, and the result which AC stored in $1005.

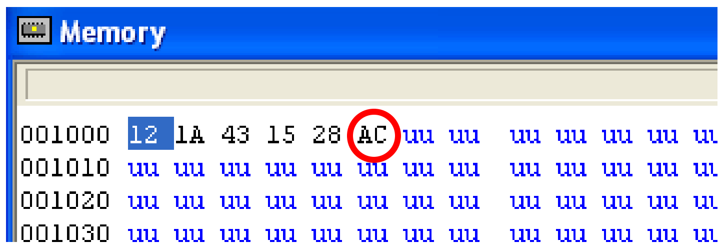


Figure 4: result of task 4

5. ANALYSIS AND INTERPRETATION

This laboratory experiment consists of 4 tasks related to branching and loops:

1- Task 1, using BEQ command stands for "branch if equal" which means "jump to given address if zero flag is set".

2- Task 2, using BNQ stands for “branch if not equal” which means "jump to given address if not zero flag is set". It is opposite to BEQ.

3- Task 3, impeded loops were used. One to test if the number is greater than 5 amd the other to check if the number is less than -5 (FB).

4- Task 4, is about branch loops. The loop was used to sum 5 numbers from memory locations into register B, and resister A was used as a counter.

6. CONCLUSION AND RECOMMENDATIONS

This experiment is about developing and testing HCS12 Programs Using Branching and Loops. The used commands are BNQ, BEQ, BLT, and BRA. The aim and objectives of the lab were achieved after completing the tasks, including designing flow charts with branches and loops, the ability to understand an implement HCS12 assembly code.

7. 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.

```
; 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
```

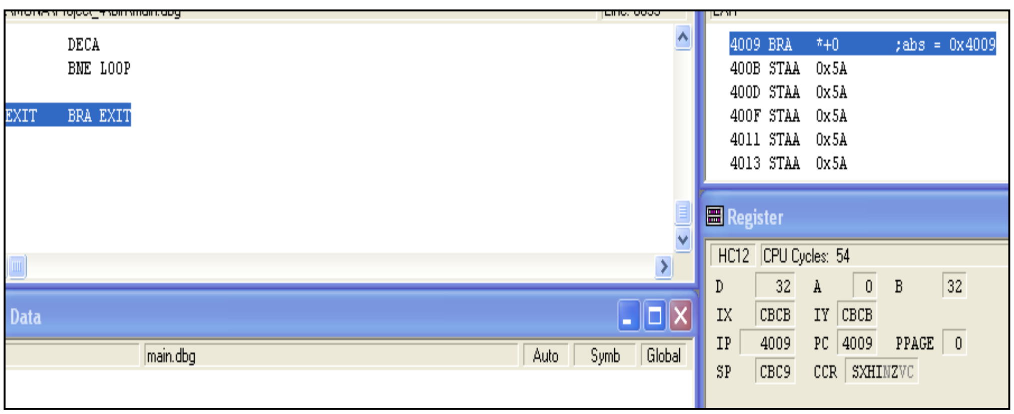
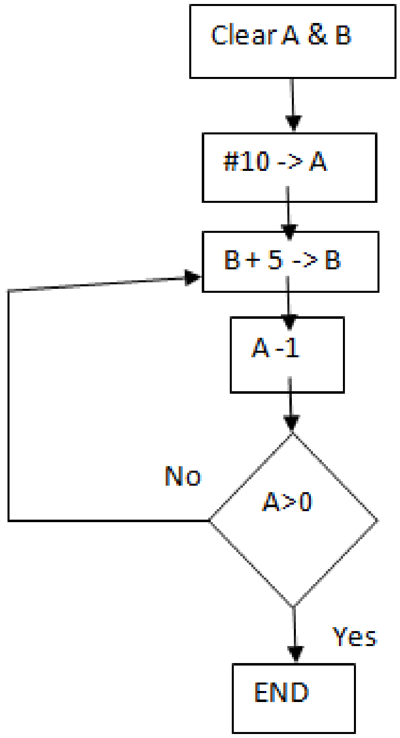
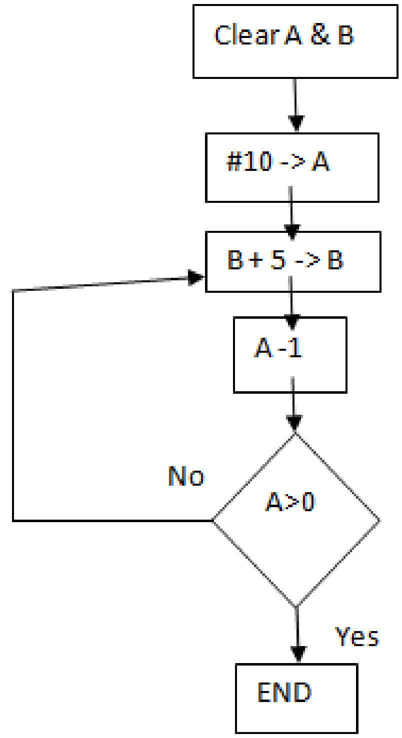


Figure 5: final content of Acc B

Comment: after running the code, the content of Accumulator B was \$32 = 50 in decimal. This shows the sum of X (5), 10 times (the loop of A).





References

1. CourseTextBook
2. http://www.evbplus.com/
3. MC9S12DP256User'sManual