KHALIFA UNIVERSITY

ELCE333 - Microprocessor Systems Laboratory

Lab Report #2 : Development & Testing of HCS12 Programs Using Branching and Loops

Shamsah Al Ali - 100037059

Fatima Alzaabi - 100036968

Submitted on: Thursday, 12th, February 2015

Table of Contents

List of Figures and Tables

List of Figures:

List of Tables:

Summary

This report illustrates the results in the second experiment, which is regarding the development and testing process of the HCS12 microprocessor using loops and branching. In the first task the student was required to use the BEQ instruction as an If-Then-Else Statement. The second task is similar to the first task but the difference is the that BNE instruction was used not the BEQ. The third task is creating Nested If-Then-Else statements. The final task was simpler we were required to design simple loops.

1. Introduction

Branch instructions come in Different types, which are used to change execution flow in assembly codes. Changes accurse only when the branch condition is satisfied, otherwise the code sequence gets executed normally. loops with stacks are types of branches .The stack is used as temporary data storage ,for instance the Stack Pointer (SP) holds the address of the last stack location (16-bits). The SP is initialized in the beginning of the code and it grows downward from the pointed address where it gets decremented whenever a byte is pushed into it and it gets incremented whenever a byte is pulled from it.

Aim

The aim of this experiment is to gain experience in the design, development and testing of HCS12 assembly programs with conditional branching structures and loops.

Objectives

- Design flow-charts containing branching and loops.
- Implement flow-charts using HCS12 assembly code.
- Use Bcc instruction to implement branching and loops.
- Download, run, and test code on a Dragon Plus Trainer board.

2. Tasks and Results

Task1: If-Then-Else Statement using BEQ

In the first task, the aim of the assembly code we wrote was checking whether the content of memory location $1000 (X1=content of $1000) is zero. If true, then the content of Y2 (Y2=content of $1001) is assigned to X1. If (false) it is not equal to zero, then move Y1 (Y1=content of $1002) to X1.

```
                INCLUDE 'derivative.inc'
                XDEF Entry

          ORG  $4000
X1  EQU  $1000
Y1  EQU  $1001
Y2  EQU  $1002

Entry:

                        MOVB #$0 , X1
                        MOVB #$D1, Y1
                        MOVB #$D2, Y2
                        CLRA
                        CMPA X1
                        BNE Eqzero
                        MOVB Y2,X1
                        BRA Exit
Eqzero                  MOVB Y1,X1
Exit                            BRA Exit
```

Figure 1- Task1 code

As you can see in figure 3, first of all we need to clear register A and then compare its content with X1. Then, by using the branch instruction BEQ to branch to label Eqzero where the content of Y2 is moved to X1 if X1 equals zero. If it doesn't, then we move the content of Y1 to X1, and finally exit from the program. The Figures below illustrates the two cases. Table 1 shows the before and after running the program, and the two figures compare the two case.

Table 3- Testing results for Task1 code

	Before Program Start			After Program End		
	$1000	$1001	$1002	$1000	$1001	$1002
Test 1	0	D1	D2	D2	D1	D2
Test 2	5	D1	D2	D1	D1	D2

Task2: If-Then-Else Statement using BNE

In task 2, we were required to modify the code in task 1, so that we exchange the BEQ branch instruction with BNE. If the content of X1is not equal to zero, then the content of Y1 is moved to X1, and if it equals zero (the condition is false), then move the content of Y2 to X1. The rest of the code stays the same as shown in figure 2.

```
                    INCLUDE 'derivative.inc'
                    XDEF Entry

               ORG  $4000
X1  EQU  $1000
Y1  EQU  $1001
Y2  EQU  $1002

Entry:

                              MOVB #$0 , X1
                              MOVB #$D1, Y1
                              MOVB #$D2, Y2
                              CLRA
                              CMPA X1
                              BEQ Eqzero
                              MOVB Y1,X1
                              BRA Exit
Eqzero                        MOVB Y2,X1
Exit                          BRA Exit
```

Figure 2- Task2 code

Table 2- Testing results for task2

	Before Program Start			After Program End		
	$1000	$1001	$1002	$1000	$1001	$1002
Test 1	0	D1	D2	D2	D1	D2
Test 2	5	D1	D2	D1	D1	D2

Task 3: Nested If-Then-Else Statements

This task required to write a program that carries out the algorithm provided in the labscript as a flowchart. The flowchart starts with comparing a value in X1 if it's greater that 5, if the value is greater than 5 then the value of Y1 is stored in X2. If the value of X1 is less than 5, then the value of X1 will then be compared with -5. If the value was greater than -5 then the value of Y2 will be stored in X2, otherwise, the value of Y3 will be stored in X2. The code below represent the functionality of the flowchart.

```
                    INCLUDE 'derivative.inc'
                    XDEF Entry
          ORG $4000
X1 EQU $1000
Y1 EQU $1001
Y2 EQU $1002
Y3 EQU $1003
X2 EQU $1004
Entry:
                              MOVB #$7, X1 ;move value 7 to X1
                              MOVB #$D1, Y1   ;move value D1 to Y1
                              MOVB #$D2, Y2 ;move value D2 to Y2
                              MOVB #$D3, Y3 ;move value D3 to Y3
                              CLRA ;clear register A to 0
                              LDAA #$5 ;load value 5 to register A
                              CMPA X1 ;compare X1 to A
                              BLE greater ; X1 > 5?
                              NEGA ; ~A +1 -> A
                              CMPA X1 ; compare X1 to A
                              BLE ngreater ; X1> -5
                              MOVB Y3, X2 ;X2 = Y3
                              BRA Exit  ; exit
greater                       MOVB Y1, X2 ;X2 = Y1
                              BRA Exit
ngreater                              MOVB Y2,X2 ;X2 = Y2
                              BRA Exit
Exit                          BRA Exit
```

Figure 3- Task3 code

In the code above, the value tested was 7, this ran the assembly code with the branch ''greater'' and the value of Y1 was stored in X2. The value 3 was tested to check the other case. In order to have the value of -5, the operational code NEGA was used to provide the

2's compliment of the value in register A for the other condition. For this condition, the code branched to the value of Y2 was stored in X2. The last case tested was the value DF (FFFFFFFFFFFFFFDF) which is the hexadecimal of -33. This case lead to no branching and the value Y3 was stored in X2.

Task 4: Simple Program with Loops

This task required to write an assembly code to demonstrate the functionality of a loop program using the flowchart given in the labscript. The code uses a pointer that will point to the first value in the DC.B directive and go through an array of numbers, adding it in the a memory location called SUM. A counter is used to be used as a condition for the loop by decrementing its value every time it goes through it so once the counter reaches 0, the code exits the loop.

```
                INCLUDE 'derivative.inc'
                XDEF Entry
        ORG $4000
NUMBERS DC.B $12, $1A, $43, $15, $28

Entry:
                        LDAA #$5                ;load value 5 to register A
                        LDY #$0                 ;load 0 to Y
                        LDX #NUMBERS ; point at first number
                        CLRB                    ;clear B
                        BRA J                   ; branch to J and enter loop
J                       LDAB 1, X+              ;point to next address
                        ABY                     ;add the value to sum
                        DECA                    ;decrement counter
                        CMPA    #$0             ;compare whether counter is zero or not
                        BEQ FINISH              ;exit loop if counter is equal to zero
FINISH                  EXG A, Y                ;exchange values
                        STAA $100               ;store value from register A into memory location
                        BRA Exit                ;exit
Exit                    BRA Exit
```

Figure 4- Task4 code

3. Assignment Questions

1.

```
        INCLUDE 'derivative.inc'
                        XDEF Entry
                ORG $4000

Entry:
                        LDAA #$10
                        ;LDX #NUMBERS
                        CLRB
                        BRA J
J                       ;LDAB 1, X+
                        ADDB #$5
                        DECA
                        CMPA #$0
                        BNE J
                        BEQ Exit
Exit                    BRA Exit
```

Figure 5- Assignment question

4. Conclusion

This laboratory demonstrated the use of the branch instructions through linear execution and loops. The first task presented the use of the instruction BEQ, where the code branches when the value is zero. The second task presents the instruction code BNQ where the code branches when the value is not zero. The third task presents a nested if else statement. the last task presents a simple program to demonstrate the use of loops using assembly code. To conclude, this laboratory experiment emphasizes on the use of branching instructions and how it can be put to use in techniques such as if statement programs and loop programs.