**Khalifa University of Science, Technology and Research**

**Electronic Engineering Department**

**ELCE332Microprocessor Systems laboratory**

**Laboratory Experiment 2**

**Development & testing of hcs12 programs using branching and loops**

**Lab Partners**

Mohammad Jassem 100037002

Khalid Al-Zarouni 100035499

Khalid Al-Hosani 100037003

**Date Experiment Performed**: 04-02-2015

**Date Lab Report Submitted**: 11-02-2015

**Lab Instructor:**

**Dr. Mahmoud Khonji, and Dr. Mohammad Al Azaabi**

**Spring 2015**

Table of Contents

[List of Figures and Tables 3](#_Toc411426492)

[Summary 4](#_Toc411426493)

[1. Introduction 5](#_Toc411426494)

[1.1 Aim 6](#_Toc411426495)

[1.2 Objectives 6](#_Toc411426496)

[2. Design and Results 7](#_Toc411426497)

[Task 1: If-Then-Else Statement using BEQ: 7](#_Toc411426498)

[Task 2: If-Then-Else Statement using BNE: 8](#_Toc411426499)

[Task 3: Nested If-Then-Else Statements 10](#_Toc411426500)

[Task 4: Simple Program with loops: 11](#_Toc411426501)

[3. Assignment Questions 13](#_Toc411426502)

[4. Conclusions and Recommendations 17](#_Toc411426503)

List of Figures and Tables

Figure 1:Flowchart of If-then-else statement using BEQ 7

Figure 2: Flowchart of If-then-else statement using BNE 8

Figure 3: Flow chart for Task 3 describes different conditions. 10

Figure 4: Flow chart describing the logical level of Task 4 12

Table 1: Status of memory locations before and after running the program of Task 1.....08

Table 2: Status of memory locations before and after running the program of Task 2.....09

Table3: Status of memory locations before and after running the program of Task 3.....10

Summary

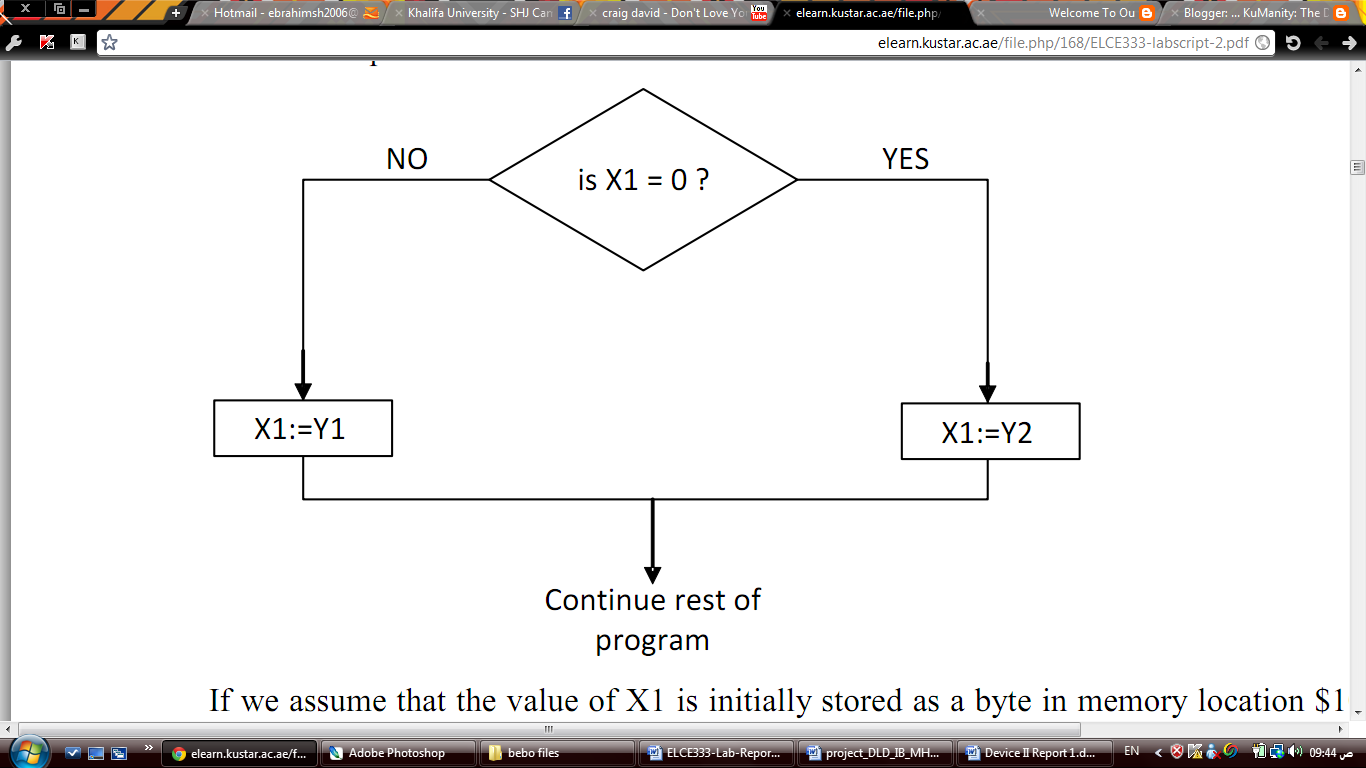
In this laboratory session, a further understanding of HCS12 assembly language is dealt with. However, conditional branching structures and loops are such functions that enable us for more functionality and ease of use. This leads us to the BEQ and BRA functions used in loops.

This laboratory session is divided into three main parts. After the introduction, we will discuss the main tasks of the experiment. First, If-Then-Else statement using BEQ will be manipulated. Then, If-Then-Else Statement using BNE will be tested. Finally, Nested If-Then-Else Statements and Simple Program with Loops will be finalized and compared to the previous tasks followed up with a conclusion summarizing the results of the laboratory session.

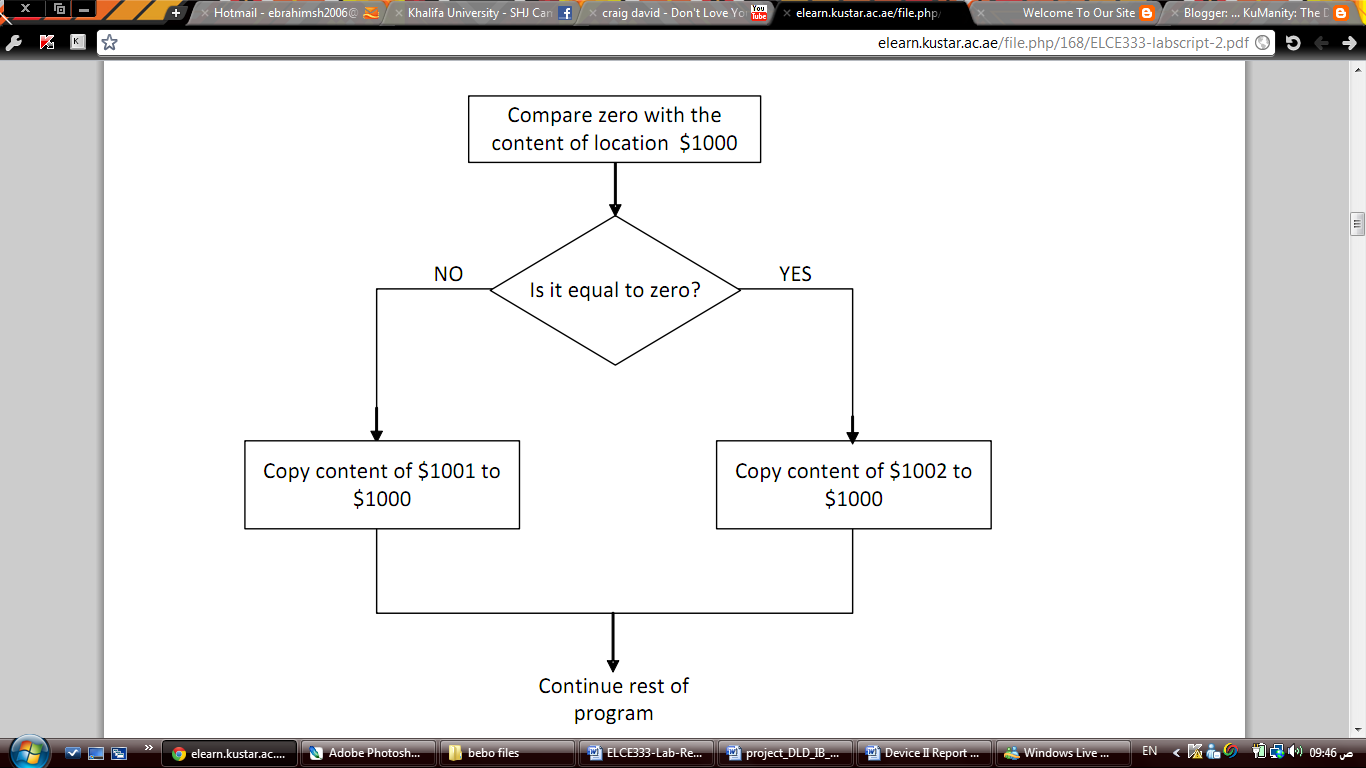
1. Introduction

Before starting the programming session, it is important to draw a schematic diagram to ease the steps for program. If we assume that a programmer wants to satisfy the following outcomes.

Using algorithms, the conditional program can be translatedinto the following diagram. To understand the methodology, the statement checks the value of X1, and then it judges on taking the suitable actions. The following flowchart summarizes the needed actions:



Let’s assume that we store a value of X1 as a byte in location $1000, Y1 in location $1001 and finally Y2 in location $1002. The following flowchart can be produced.



Sequence Control Instructions or Branch Instructions are special instructions used for such needed operations. HCS12 has a useful number of such operation for sequence control such as BEQ and BRA.

1.1 Aim

To gain experience in the design, development and testing of HCS12assembly programs with conditional branching structures and loops.

1.2 Objectives

On completion of this experiment the students should be able to:

1- Design flow-charts containing branching and loops.

2- Implement flow-charts using HCS12 assembly code.

3- Use Bcc instruction to implement branching and loops.

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

2. Design and Results

## Task 1: If-Then-Else Statement using BEQ:

To begin with, we modify the assembly code given in the lab script to apply the actions mentioned; that is, after judging the value of X1 and taking the needed action using BEQ (Branch Equal to Zero) In other words, using If-Then-Else statement to test 2 cases when X1=0 and X1≠0.The flow chart in Figure1 represents the mechanism.

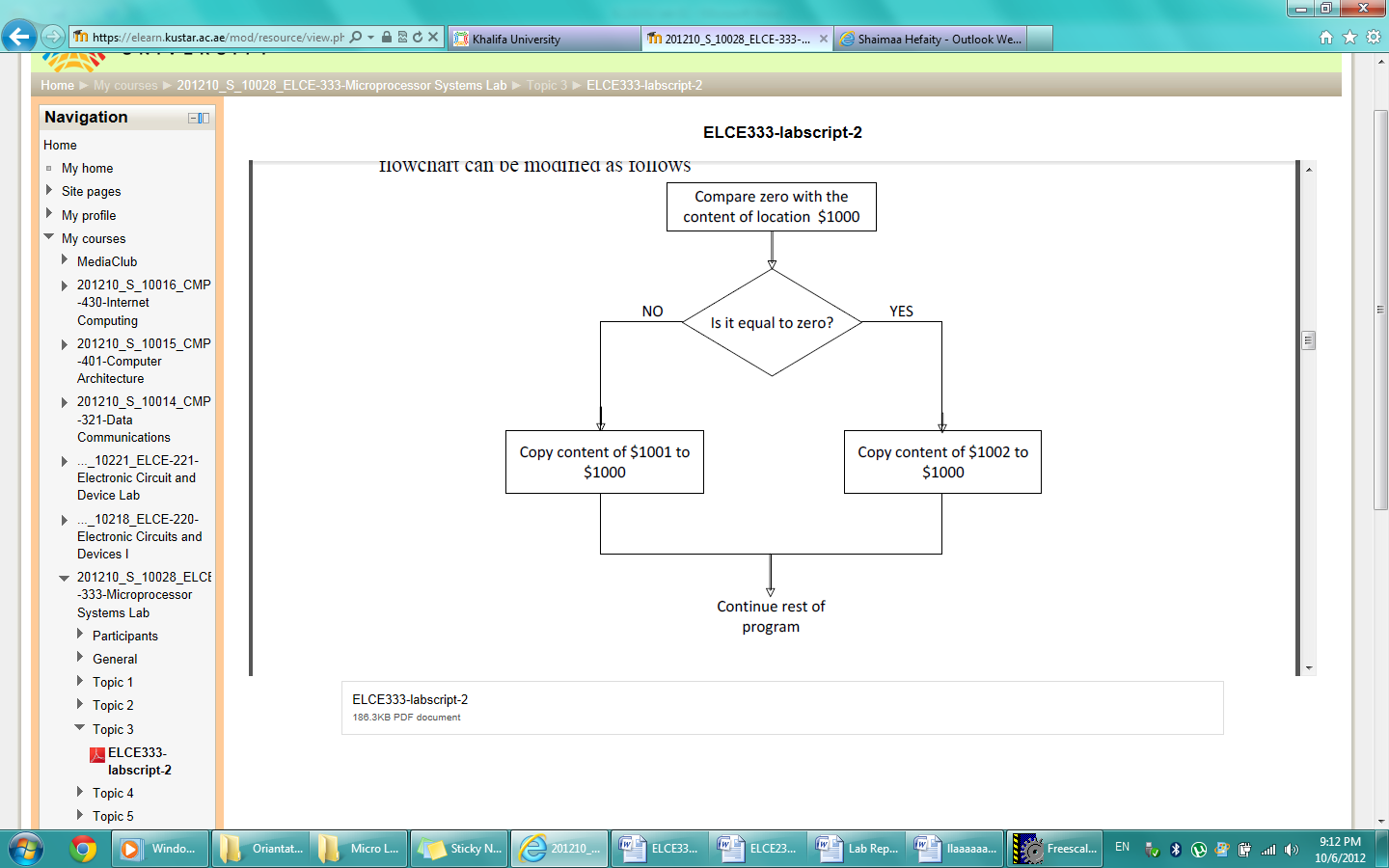


Figure 1: Flowchart of If-then-else statement using BEQ

The following code is a translation from the algorithm into a useful assembly code.

ABSENTRY Entry

INCLUDE 'derivative.inc'

ORG $4000 ;Flash ROM address for Dragon12+

X1 EQU $1000

Y1 EQU $1001

Y2 EQU $1002

Entry:

MOVB #$0,X1 ; when X1 is not zero then X1=Y1

MOVB #$A1,Y1 ; when X1 is not zero then X1=Y1

MOVB #$A2,Y2 ; when X1 is not zero then X1=Y1

CLRA

CMPA X1 ; compare zero with X1

BNE 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

Using the single step execution method, the table

Table 1: Status of memory locations before and after running the program of Task 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Before Program Start | | | After Program End | | |
| $1000 **(X1)** | $1001**(Y1)** | $1002**(Y2)** | $1000**(X1)** | $1001**(Y1)** | $1002**(Y2)** |
| Test1 (X1=0) | 00 | 10 | 20 | **20** | 10 | **20** |
| Test2 (X1≠0) | 10 | 10 | 20 | **10** | **10** | 20 |

If we think about the code, the code has only two states. This is exactly what we did in the previous table. In the first test, if the value of X1 is 0, Y2 is assigned to X1 which is 20. In the second test, the value of X1= 10 as the value of Y1, which is 10, was assigned to X1. This is what we predicted from the code.

## Task 2: If-Then-Else Statement using BNE:

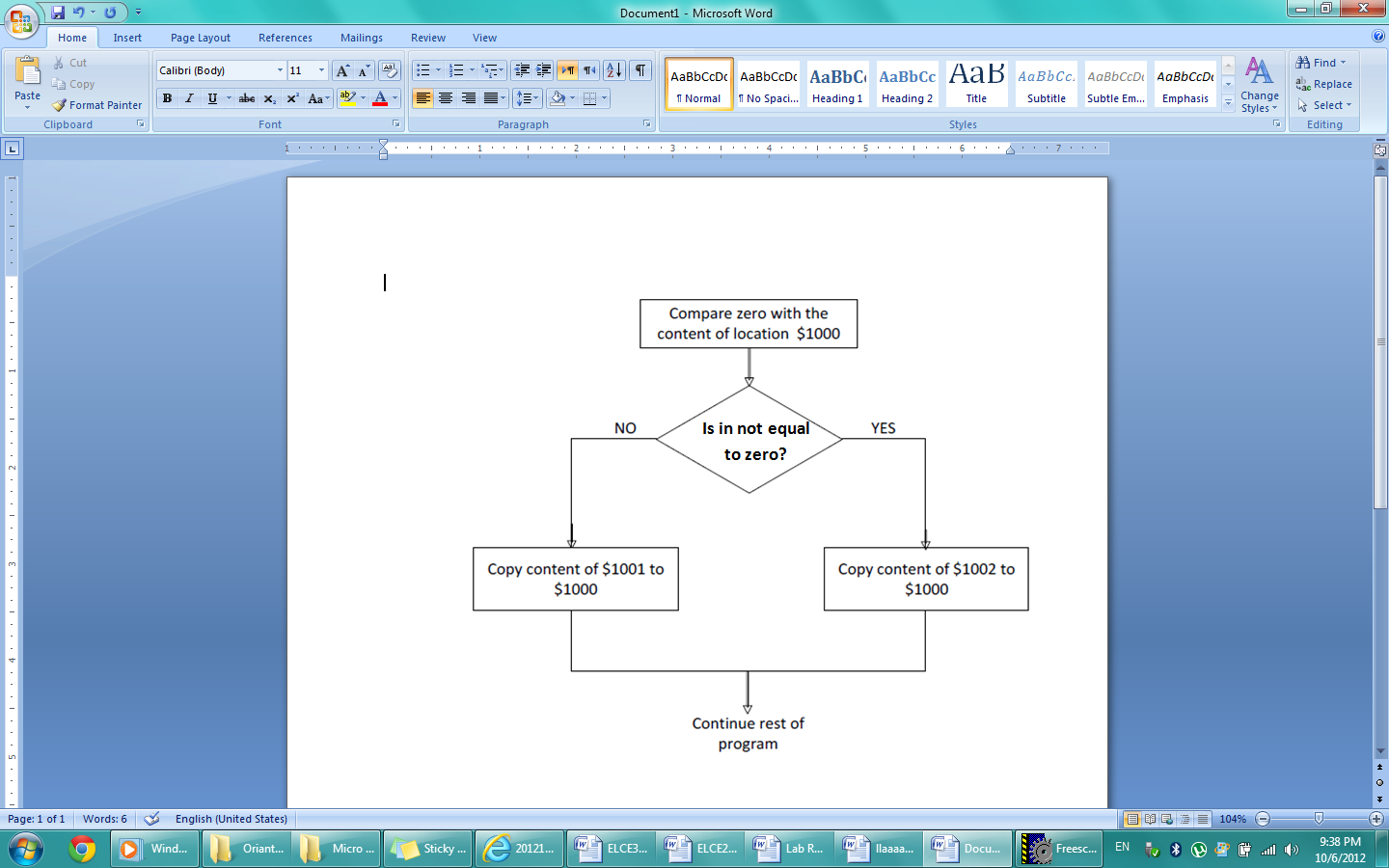
In this task, we use Branch if not equal to zero compared to the previous task where we used Branch if equal to zero. However, the small modification on the previous task is pretty simple. We simply shift the condition code between the **if** and **else** function as follows.

Figure 2: Flowchart of If-then-else statement using BNE

ABSENTRY Entry

INCLUDE 'derivative.inc'

ORG $4000 ;Flash ROM address for Dragon12+

X1 EQU $1000

Y1 EQU $1001

Y2 EQU $1002

Entry:

MOVB #$3,X1 ; when X1 is not zero then X1=Y1

MOVB #$A1,Y1 ; when X1 is not zero then X1=Y1

MOVB #$A2,Y2 ; when X1 is not zero then X1=Y1

CLRA

CMPA X1 ; compare zero with X1

BEQ Eqzero ; if X1 is zero, go to "equalzero"

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

We can see that the single step results are the same as the previous task as shown in table Table 2.

Table 2: Status of memory locations before and after running the program of Task 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Before Program Start | | | After Program End | | |
| $1000 **(X1)** | $1001**(Y1)** | $1002**(Y2)** | $1000**(X1)** | $1001**(Y1)** | $1002**(Y2)** |
| Test1 (X1=0) | 00 | 10 | 20 | **20** | 10 | **20** |
| Test2 (X1≠0) | 10 | 10 | 20 | **10** | **10** | 20 |

This is exactly what we did in the previous task. In the first test, if the value of X1 is 0, Y2 is assigned to X1, which is 20. In the second test, the value of X1= 10 as the value of Y1, which is 10, was assigned to X1. This is what we predicted from the code.

## Task 3: Nested If-Then-Else Statements

In this task, the following chart is translated into an assembly program:

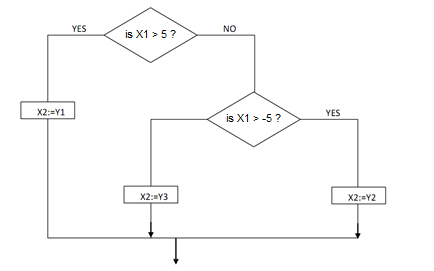


Figure 3: Flow chart for Task 3 describes different conditions.

The If-Then-Else nested Statements are satisfied using any of the branching instructions.The following table represents the results obtained from the test that used BLT (Branch if Less Than) and BRA (Branch Always) to apply and jump across the unused statements to exit respectively.

Table 3: Status of memory locations before and after running the program of Task 3

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Before Program Start | | | | | After Program End | | | | |
| $1000  X1 | $1001  X2 | $1002  Y1 | $1003  Y2 | $1004  Y3 | $1000  X1 | $1001  X2 | $1002  Y1 | $1003  Y2 | $1004  Y3 |
| Case1 | 2 | 10 | 20 | 30 | 40 | 2 | 30 | 20 | 30 | 40 |
| Case2 | -6 | 10 | 20 | 30 | 40 | -6 | 40 | 20 | 30 | 40 |
| Case3 | 6 | 10 | 20 | 30 | 40 | 6 | 20 | 20 | 30 | 40 |

The following is the assembly language source code written to satisfy the results:

ABSENTRY Entry

INCLUDE 'derivative.inc'

X1 EQU $1000

X2 EQU $1001

Y1 EQU $1002

Y2 EQU $1003

Y3 EQU $1004

Entry:

MOVB #$07, X1

MOVB #$10, X2

MOVB #$20, Y1

MOVB #$30, Y2

MOVB #$40, Y3

LDAA #$05

CMPA X1

BLT biggerfive

LDAA #$FB

CMPB X1

BLT biggermin

MOVB Y3, X2

BRA Exit

biggerfive

MOVB Y1, X2

BRA Exit

biggermin

MOVB Y2,X2

BRA Exit

Exit BRA Exit

## Task 4: Simple Program with loops:

The last task, was about writing a code that adds a specific number of different numbers to each other and then store in a memory location. To do that, a counter should be initialized to the number of the numbers that are needed to be summed in our case it is 5 numbers and the SUM to 0. We loaded the numbers in Register X, then we took each number for Register X to Register B, after that we add Register B with Register Y, we repeat the same steps until all numbers are added. Finally we stored it in the memory.

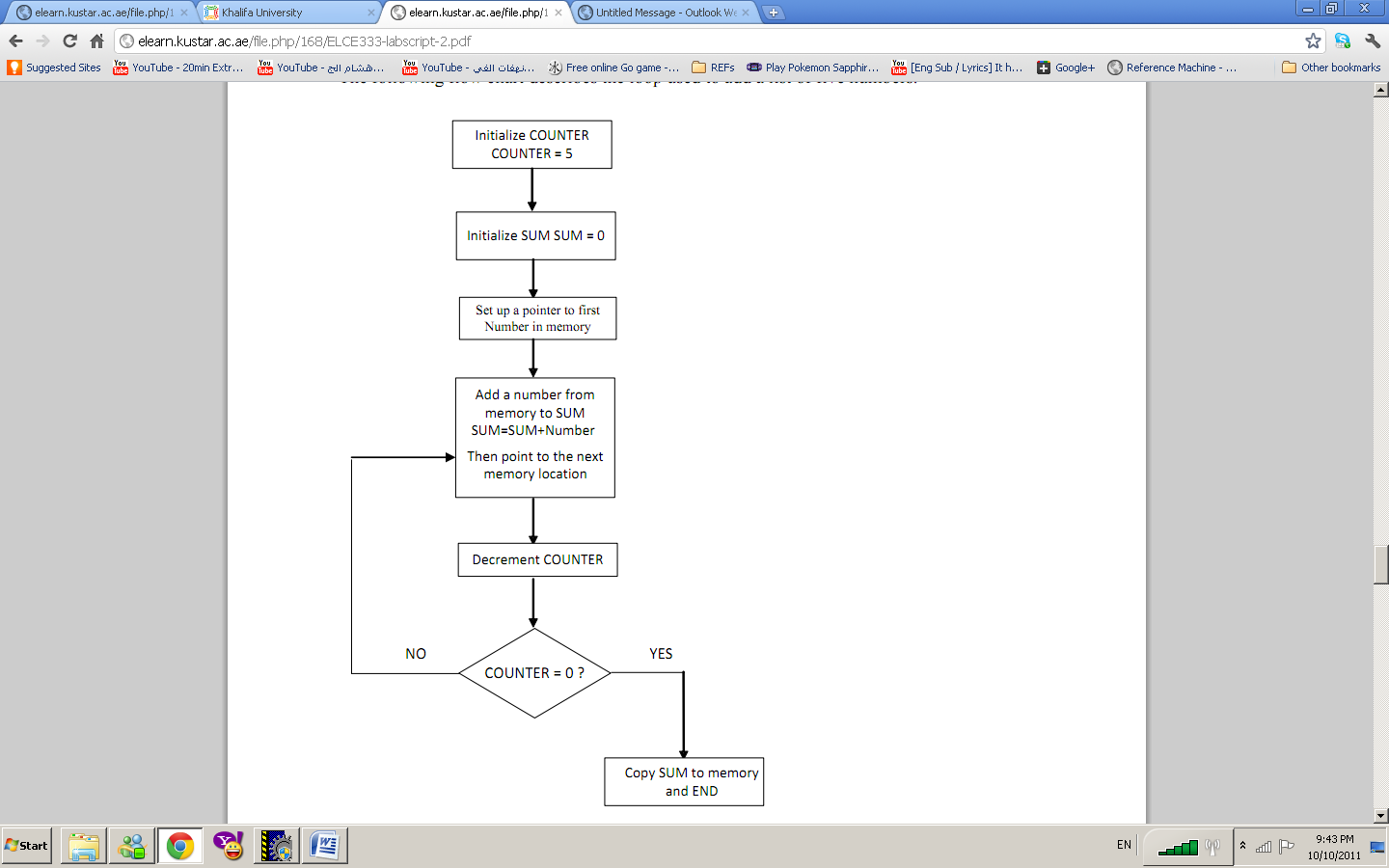


Figure 4: Flow chart describing the logical level of Task 4

The Source code that performs the above algorithm is shown below:

ABSENTRY Entry

INCLUDE 'derivative.inc'

SUM EQU $1000

ORG $1010

FINAL\_SUM EQU $1001

NUMBERS DC.B $12,$1A,$43,$15,$28

Entry:

CLRA

CLRB

LDY #$0000

LDAA #5

LDX #NUMBERS

More LDAB 1,X+

ABY

DECA

CMPA #$0

BNE More

STY SUM

Exit JMP Exit

END

# 3. 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.inc***

*INCLUDE 'derivative.inc'*

***; export symbols***

*XDEF Entry*

***; the needed code along with loop.***

*ORG $4000*

*Entry:*

*CLRB*

*LDAA #$A ; 10 in hex, to set a counter.*

*LOOP1:*

*ADDB #$5 ; 5 to be added 10 times.*

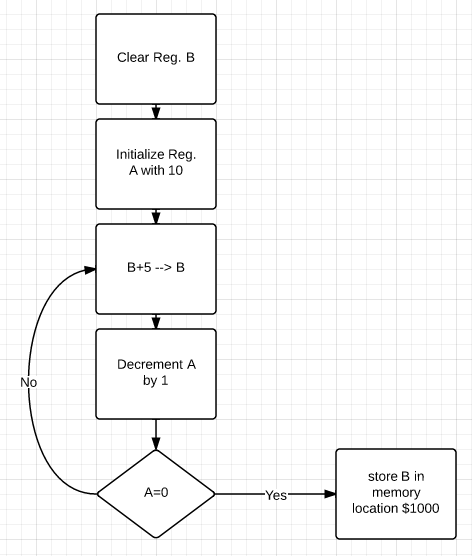
*DECA ; set a counter*

*CMPA #0 ; compare to 0*

*BNE LOOP1; branch if higher than*

*STAB $1000; store the value of Reg. B in $1000*

*Exit JMP Exit*

To translate the code into a flowchart, we do the following using a spacific programe.

2. Write an HCS12 assembly program that reads the following string of 8-bit small letter characters located in memory starting at location $1010.

String : “ilovemicrocontrollers”

The end of the sequence is indicated by a null. The program should count the number of vowel letters and store them in registers A and the number of constant letters and store them in registers B. The program should also copy the vowel letters to memory stating at location $1030, and the constant letters to memory starting at location $1050. Draw the program flowchart.

Hints:

1) Your string is defined using DC.B ‘ilovemicrocontrollers’,$00

2) Small vowel letters ascii equivalents are: a=0x61, e=0x65, i=0x69, o=0x6F, u=0x75

**INCLUDE 'derivative.inc'**

XDEF Entry

VowelCOUNTER EQU $1000 **;Storing the number of vowel letters**

ConstCOUNTER EQU $1060 **;Storing the number of constant letters**

STOREvowels EQU $1030 **; Storing the vowel letters starting at $1030**

STOREconstants EQU $1050 **;Storing the constant letter starting at $1050**

ORG $4000

Entry:

CLR $1060 **;Vowel counter**

CLR $1000 **;Const counter**

CLRA

CLRB

LDX #STRING

LDY #STOREvowels

loop1 LDAA X ;(X)-->A

CMPA #$00 **;Check if the string has ended**

BEQ exit **;If we checked the whole string then end the program**

CMPA #$69 **; Compare character with 'i'**

BEQ vowel

CMPA #$65 **;Compare character with 'e'**

BEQ vowel

CMPA #$61 **;Compare character with 'a'**

BEQ vowel

CMPA #$6F **;Compare character with 'o'**

BEQ vowel

CMPA #$75 **;Compare character with 'u'**

BEQ vowel

;a=0x61, e=0x65, i=0x69, o=0x6F, u=0x75

constant:

STAA STOREconstants **;(A)--> Starting at $1050**

INX

INC ConstCOUNTER

BRA loop1

vowel:

STAA Y

INC VowelCOUNTER

INY

INX

BRA loop1

exit:

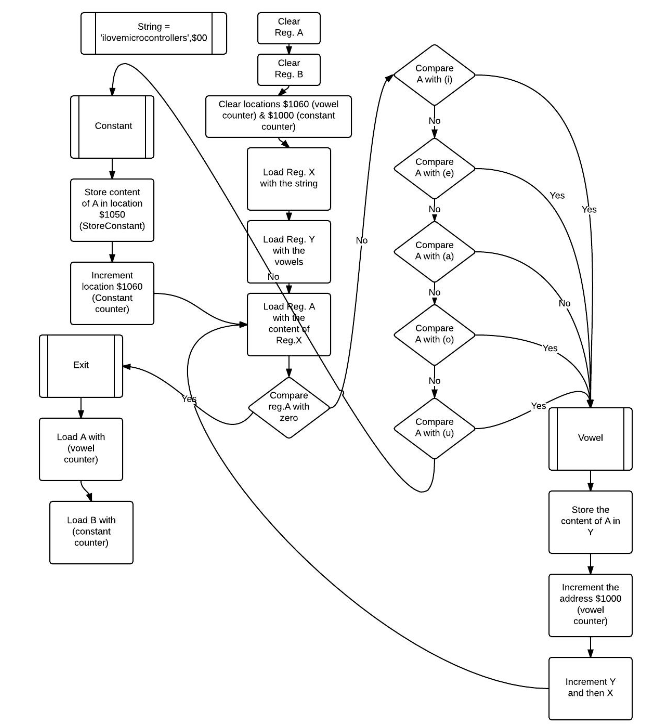
LDAA VowelCOUNTER **;Store the number of vowels in Acc A**

LDAB ConstCOUNTER **;Store the number of constants in Acc B**

Exit BRA Exit

ORG $1010

STRING DC.B 'ilovemicrocontrollers',$00



4. Conclusions and Recommendations

To sum up, the simulated results of the written codes agree with exact needed action. Branching and looping have been clearly understood and effectively employed depending on the needed results. Students are now able to produce flowcharts, create the suitable mechanism and condition codes. The students are now familiar with producing the code, testing the code by simulation and finally practically implement on the dragon Board. In conclusion, the tasks have familiarized the students with The Code Warrior program. The main objective of the laboratory session have been achieved along with the sub aims