KHALIFA UNIVERSITY

**Module Name: Microprocessor Systems Laboratory**
**Module Code: ELCE333**

# Laboratory Experiment No. 1

**Expriment Tite: Microcontroller Assembly Program Development**
**Lab Report**

## 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

### *Table of Contents*

# List of tables and figures

## List of tables:

## List of Figures:

## Abstract

The main goal of this experiment is to edit, assemble, simulate, download, and then execute an assembly program. In the first task, we need to create a project in CodeWarrior and editing it into the shell of an assembly language program. The next task is to modify a program using SUB command to subtract bytes. The third task we use the list file which to check and correct errors and finally the conclusion.

# 1. Introduction

Assembly language is a language that allows users and microcontrollers to converse between each other and it's the basic language for processors. The user deals with instructions that are based on the physical CPU. CodeWarrior IDE is used in this lab, which is a software tool that sets up the connection with the micro controller and run the assembly language accordingly. The communication needs several elements to be present in order for it to be successful and error free. Those elements are:

**a. Assembly language syntax**

Those are the regulations that state how expressions are formed into statements that consist of the assembly language.

**b. Labels**

A text that represent the address in ROM or RAM

**c. Instruction**

Defined by the manufacturer of the micro controller which means the programmer needs to learn the rules of the micro controller he's using. The instructions performed in this lab are arithmetic and data transfer instructions.

**d. Operands**

It is the argument or value on which the instruction performs at like a variable, register, label or memory address.

A micro controller is a compact computer created to monitor the operation of an embedded system. micrcontrollers nowadays can be found everywhere in all items in everyday life. They are designed to do a very specific task therefore the micro controller size can be simplified that decreases the costs of production. Because it needs to operate at a high speed it contains the following:

**Read Only Memory (ROM)**

It's a type of memory that can be only read and to be written to. It saves the data permanently without being deleted when its turned off so it's not lost. The data in the memory cannot be altered again after it's initially saved.

**Random Access Memory (RAM)**

It's a memory that is used temporary store data and is accessed in a random fashion that a byte is accessed in memory without going through it's preceding bytes. This type of memory is volatile so when the power is shut off the memory is cleared.

**Electrically Erasable Programmable ROM (EEPROM)**

This type of memory has its contents changed when executing a program but saves the data permanently like the ROM by exposing the memory to an electric charge but it's not as fast as the RAM.

**Special Function Registers (SFR)**

Those registers are a part of RAM memory where their purposes are predined by the manufacturer. They control and regulate the functions of the microprocessor .

**Program counter**

This is a register that has the location of the instruction executed right now. When the power is off, the program counter is set to 0. Program counter can be referred to address pointer too.

**Central Processor Unit (CPU)**

This unit governs all the processes in the micro controller and it performs the instructions that deal with it.

**Aim:** The main goal of this experiment is to deal more with the assembly language, and to learn and be familiar with process of editing, assembling, simulating, downloading and executing a certain program.

**Objectives: for this lab experiment it is expected to:**

1. Writing and editing an assembly program on HCS12 microcontroller.
2. Generating a binary file and a list file from an assembly program.
3. Downloading the generated binary file to the Dragon Plus Trainer board.
4. Using the single-step function at the CodeWarrier program.
5. Tracing the program execution process and checking the system registers.
6. Utilizing the HCS12 Instruction Set Manual to learn, understand and perform a given program.
7. Understanding the content of the program list file.

## 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 three tasks. The first task is about adding numbers. In the second task numbers will be subtracted. And in the final part a file called a list file will be viewed and studied by using the code written in task 1.

### 2.1. Task1: Adding numbers

In this part, a project named lab1task1 was created using CodeWarrier software and the following program was edited and added into the shell of an assembly language window:

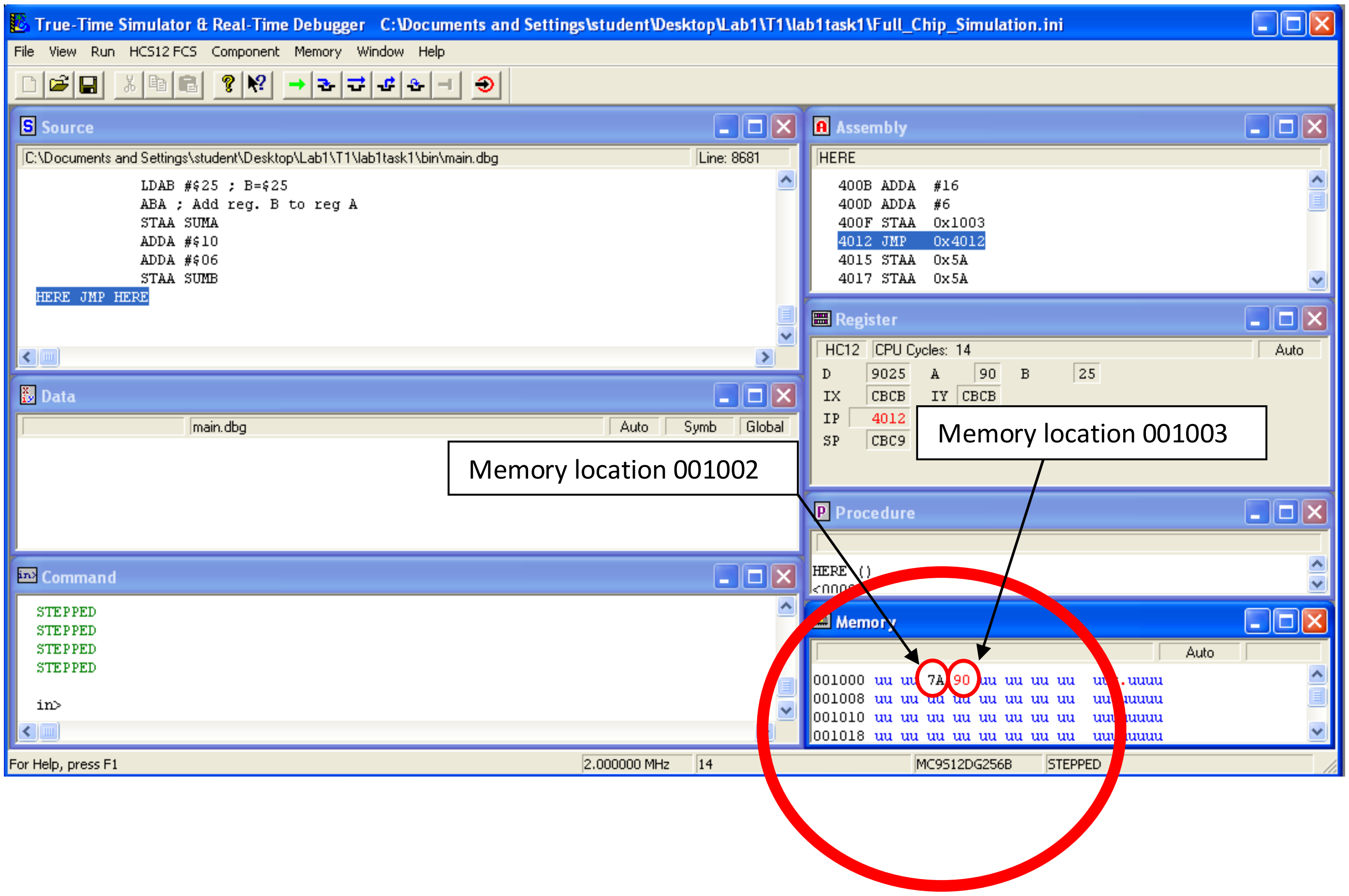
```
; Include derivative-specific definitions
INCLUDE 'derivative.inc'
; export symbols
XDEF Entry
; Insert here your data definition.
SUMA EQU $1002  ; initiate location/address $1002 in the memory to store the value of SUMA
SUMB EQU $1003  ; initiate location/address $1002 in the memory to store the value of SUMB
; code section
ORG $4000
Entry: CLRA ;
CLRB ;
LDAA #$55 ; A=$55 ;
LDAB #$25 ; B=$25 ;
ABA ; Add reg. B to reg A
STAA SUMA ;
ADDA #$10 ;
ADDA #$06 ;
STAA SUMB ;
HERE JMP HERE
```

This program was assembled and simulated by stepping though it, to observe and study the content of registers (Accumulator A , Accumulator B and Accumulator D), table 1 represents these result of the registers content after each step.

**Table1 Registers contents while single stepping in the program**

| **instruction** | **A** | **B** | **D =AB** | **Comment** |
|---|---|---|---|---|
| **CLRA** | 0 | CB | CB | Clear the content of the accumulator A by setting it to 0 |
| **CLRB** | 0 | 0 | 0 | Clear the content of the accumulator B by setting it to 0 |
| **LDAA #$55** | 55 | 0 | 5500 | Initialize Accumulator A with the immediate value 0x55 |
| **LDAB #$25** | 55 | 25 | 5525 | Initialize Accumulator B with the immediate value 0x25 |
| **ABA** | 7A | 25 | 7A25 | Add the content of Accumulator B to Accumulator A and store it in Accumulator A |
| **STAA SUMA** | 7A | 25 | 7A25 | Store the value of Accumulator A 0x7A to in SUMA in the memory location $1002 |
| **ADDA #$10** | 8A | 25 | 8A25 | Add the immediate value #$10 to the content of Accumulator A and store it in Accumulator A |
| **ADDA #$06** | 90 | 25 | 9025 | Add the immediate value #$06 to the content of Accumulator A and store it in Accumulator A |
| **STAA SUMB** | 90 | 25 | 9025 | Store the value of Accumulator A 0x7A to in SUMB in the memory location $1003 |

Once these steps of the program were finished, the locations $1002(which represents SUMA) and $1003 (which represents SUMB) in the memory were inspected and observed as shown in figure 1, the memory location $1002 has the value 7A which represent the value of A at which instruction **STAA SUMA** was executed, and the memory location $1003 has the value 90 which represents the value of A at which instruction **STAA SUMB** was executed.


True-Time Simulator & Real-Time Debugger C:\Documents and Settings\student\Desktop\Lab1\T1\lab1task1\Full_Chip_Simulation.ini
File View Run HCS12 FCS Component Memory Window Help
Source
Assembly
Register
Data
Procedure
Command
Memory
Memory location 001002
Memory location 001003

C:\Documents and Settings\student\Desktop\Lab1\T1\lab1task1\bin\main.dbg Line: 8681

```
            LDAB #$25 ; B=$25
            ABA ; Add reg. B to reg A
            STAA SUMA
            ADDA #$10
            ADDA #$06
            STAA SUMB
HERE JMP HERE
```

```
HERE
  400B ADDA  #16
  400D ADDA  #6
  400F STAA  0x1003
  4012 JMP   0x4012
  4015 STAA  0x5A
  4017 STAA  0x5A
```

```
HC12  CPU Cycles: 14                 Auto
D   9025   A   90   B   25
IX  CBCB   IY  CBCB
IP  4012
SP  CBC9
```

main.dbg Auto Symb Global

```
STEPPED
STEPPED
STEPPED
STEPPED

in>
```

```
HERE ()
<000
```

```
                                          Auto
001000 uu uu 7A 90 uu uu uu uu   uu .uuuu
001008 uu uu uu uu uu uu uu uu   uuuuuuuu
001010 uu uu uu uu uu uu uu uu   uuuuuuuu
001018 uu uu uu uu uu uu uu uu   uuuuuuuu
```

For Help, press F1    2.000000 MHz    14    MC9S12DG256B    STEPPED

**Figure1:memory content after executing the whole program**

### 2.2. Task2: Subtraction Numbers

In this part, the same steps as the previous task were followed, and the program used in task1 was used and edited in order to perform subtraction operation. The following program was added into the shell of an assembly language window:

```
; Include derivative-specific definitions
        INCLUDE 'derivative.inc'
; export symbols
          XDEF Entry
; Insert here your data definition.
VALUE EQU $1003; initiate location/address $1002 in the memory to store the value of VALUE
RESULT EQU $1004; initiate location/address $1002 in the memory to store the value of RESULT
; code section
          ORG $4000
Entry: CLRA
          CLRB
          LDAA #$55 ; A=$55
          LDAB #$25 ; B=$25
          STAA VALUE
          SUBB VALUE
          STAB RESULT
HERE JMP HERE
```

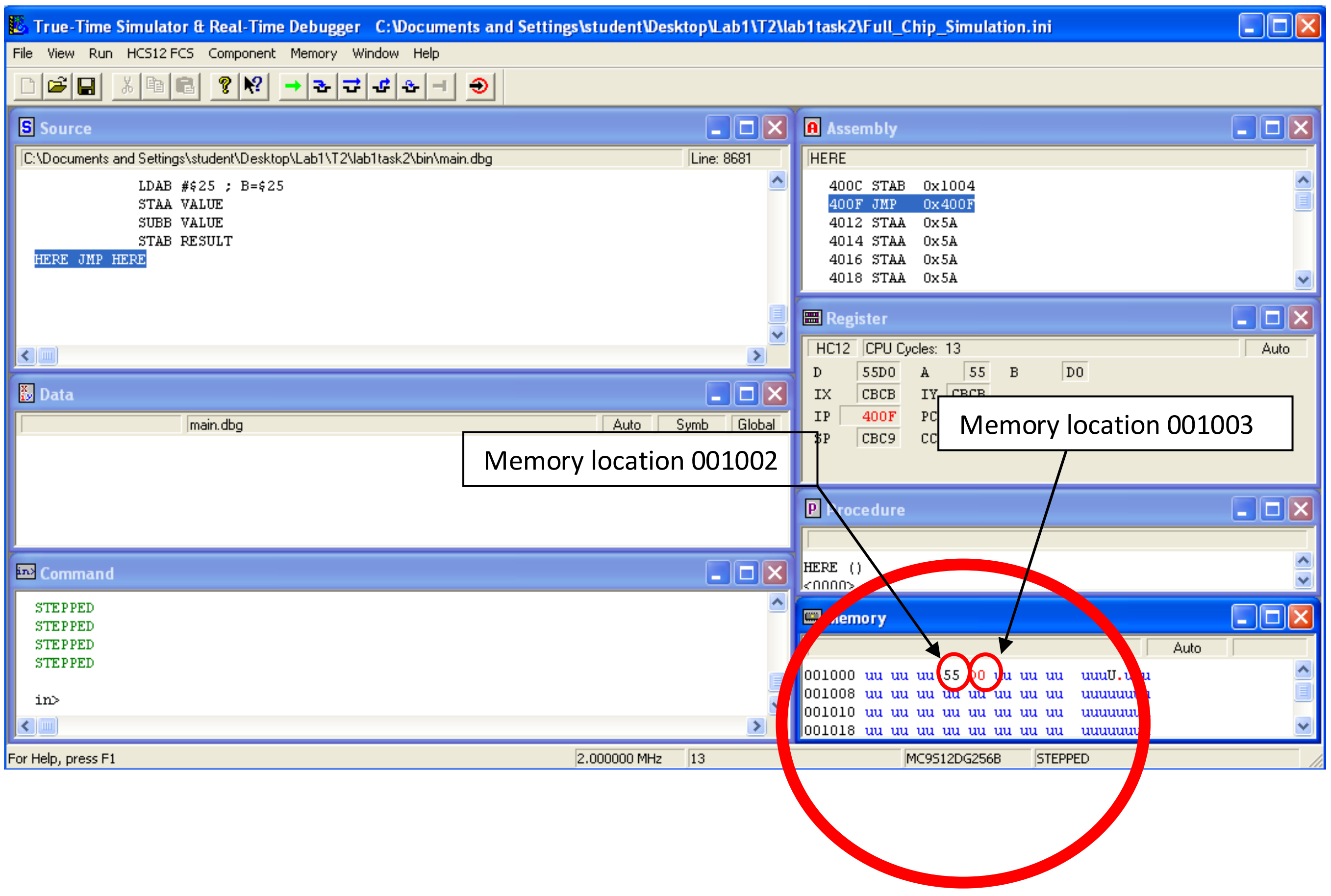
This

program was assembled and simulated by stepping though it, to observe and study the content of registers (Accumulator A , Accumulator B and Accumulator D), table 2 represents these result of the registers content after each step.

**Table2  Registers contents while single stepping in the program**

| **instruction** | **A** | **B** | **D** | **Comment** |
|---|---|---|---|---|
| **CLRA** | 0 | CB | CB | Clear the content of the accumulator A by setting it to 0 |
| **CLRB** | 0 | 0 | 0 | Clear the content of the accumulator B by setting it to 0 |
| **LDAA #$55** | 55 | 0 | 5500 | Initialize Accumulator A with the immediate value 0x55 |
| **LDAB #$25** | 55 | 25 | 5525 | Initialize Accumulator B with the immediate value 0x25 |
| **STAA VALUE** | 55 | 25 | 5525 | Store the value of Accumulator A 0x55 in VALUE in the memory location $1003 |
| **SUBB VALUE** | 55 | D0 | 55D0 | Subtract the content of VALUE stored in the memory location $1003 from B and store it in Accumulator B |
| **STAB RESULT** | 55 | D0 | 55D0 | Store the value of Accumulator B 0xD0 in RESULT in the memory location $1004 |

Once these steps of the program were finished, the locations $1003(which represents VALUE) and $1004 (which represents RESULT) in the memory were inspected and observed as shown in figure 2, the memory location $1003 has the value 55 which represent the value of A at which instruction **STAA VALUE** was executed, and the memory location $1004 has the value D0 which represents the value of B at which instruction **STAB RESULT** was executed.


True-Time Simulator & Real-Time Debugger   C:\Documents and Settings\student\Desktop\Lab1\T2\lab1task2\Full_Chip_Simulation.ini
File  View  Run  HCS12 FCS  Component  Memory  Window  Help
Source
Assembly
Register
Data
Procedure
Command
Memory
Memory location 001002
Memory location 001003

```
C:\Documents and Settings\student\Desktop\Lab1\T2\lab1task2\bin\main.dbg    Line: 8681
            LDAB #$25 ; B=$25
            STAA VALUE
            SUBB VALUE
            STAB RESULT
HERE JMP HERE
```

```
HERE
  400C STAB   0x1004
  400F JMP    0x400F
  4012 STAA   0x5A
  4014 STAA   0x5A
  4016 STAA   0x5A
  4018 STAA   0x5A
```

```
HC12   CPU Cycles: 13                    Auto
D    55D0    A    55    B    D0
IX   CBCB    IY   CBCB
IP   400F    PC
SP   CBC9    CC
```

```
main.dbg                    Auto   Symb   Global
```

```
HERE ()
<0000>
```

```
STEPPED
STEPPED
STEPPED
STEPPED

in>
```

```
                                    Auto
001000 uu uu uu 55 D0 uu uu uu   uuuU.uuu
001008 uu uu uu uu uu uu uu uu   uuuuuuuu
001010 uu uu uu uu uu uu uu uu   uuuuuuuu
001018 uu uu uu uu uu uu uu uu   uuuuuuuu
```

```
For Help, press F1          2.000000 MHz   13          MC9S12DG256B   STEPPED
```

**Figure 2:memory content after executing the whole program**

The program written by the team worked perfectly, it was suggested in the lab script to use SUBA to perform A-B operation and store it at A; however this program could have been written in a better way to consume storage in the memory, instead of initiating a temporary VALUE and store the value of accumulator B in it and then subtract A-VALUE, the instruction SBA could have been used to perform this operation directly and save memory.

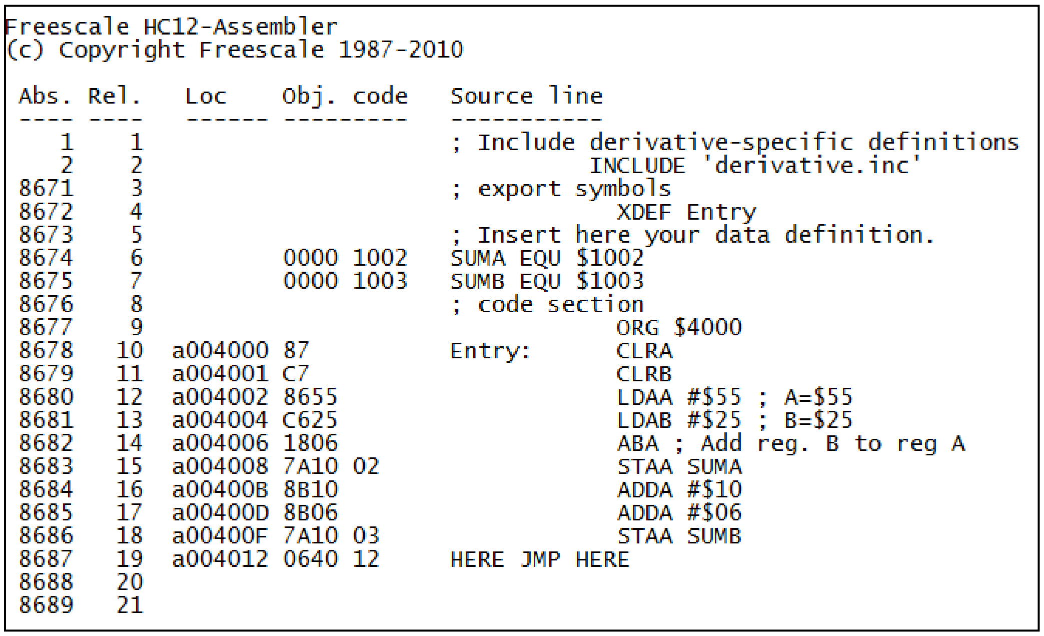
## 2.2. Task3: Simple Program

In this part of the experiment, the program of task1 will be used in order to create a file called a list file that contains the following:

- The source-code program
- The assembled code in hexadecimal format (including memory locations)
- Any error messages
- A list of user-defined symbols used and their values

The steps for creating this file (which were listed and explained fully in the lab script) were followed. And a file named From this file lots of information can be obtained, like the Start address from column Loc and the Instruction code from the column Obj. code. This file was studied in order to fully understand what it contains. The following table explains some of the file content:

main.lst was created in the bin folder of the program( as seen in figure 2 ).



```
Freescale HC12-Assembler
(c) Copyright Freescale 1987-2010

 Abs. Rel.   Loc    Obj. code   Source line
 ---- ----   ------ ---------   -----------
    1    1                      ; Include derivative-specific definitions
    2    2                                INCLUDE 'derivative.inc'
 8671    3                      ; export symbols
 8672    4                                  XDEF Entry
 8673    5                      ; Insert here your data definition.
 8674    6          0000 1002   SUMA EQU $1002
 8675    7          0000 1003   SUMB EQU $1003
 8676    8                      ; code section
 8677    9                                  ORG $4000
 8678   10  a004000 87          Entry:      CLRA
 8679   11  a004001 C7                      CLRB
 8680   12  a004002 8655                    LDAA #$55 ; A=$55
 8681   13  a004004 C625                    LDAB #$25 ; B=$25
 8682   14  a004006 1806                    ABA ; Add reg. B to reg A
 8683   15  a004008 7A10 02                 STAA SUMA
 8684   16  a00400B 8B10                    ADDA #$10
 8685   17  a00400D 8B06                    ADDA #$06
 8686   18  a00400F 7A10 03                 STAA SUMB
 8687   19  a004012 0640 12     HERE JMP HERE
 8688   20
 8689   21
```

**Figure 3:main.lst file content**

| **Table 3 The memory contents for the developed program** | | | | |
|---|---|---|---|---|
| **Instruction** | **Start Address** | **No. of Bytes** | **Instruction Code** | **Comments** |
| **CLRA** | a004000 | 1 | 87 | Start Address: this start address was initiated in the program (ORG $4000) so starts saving the content of the program at this memory address.<br>No.of Bytes:this is obtained either by counting the number of bytes in the instruction code or the difference between the next Start address and the current start address .by obtaining it in the instruction code. Each digit in the instruction code has 4 bits, so the number of bytes in 1 Byte.<br>Using the start address: a004001-a004000=1Byte |
| **CLRB** | a004001 | 1 | C7 | Start Address: since we have one byte to store in the previous step, so the new start address is a004000+1 = a004001<br>No.of Bytes:here we can obtain the number of bytes using both mentioned ways explained in the previous comment.<br>C7 = 8Bits =1Byte, a004002- a004001=1Byte |
| **LDAA #$55** | a004002 | 2 | 8655 | Start Address: since we have one byte to store in the previous step, so the new start address is a004001+1 = a004002<br>No.of Bytes: 8655 = 16 Bits =1Byte, a004004-a004002=2Bytes |
| **LDAB #$25** | a004004 | 2 | C625 | Start Address: since we have two bytes to store in the previous step, so the new start address is a004002+2 = a004004<br>No.of Bytes: C625=16Bits=2Bytes, a004006-a004004=2Bytes |
| **ABA** | a004006 | 2 | 1806 | Start Address: since we have two bytes in the previous step, so the new start address is a004004+2 = a004006<br>No.of Bytes: 1806=16 Bits=2Bytes,a004008-a004006=2Bytes |
| **STAA SUMA** | a004008 | 3 | 7A1002 | Start Address: since we have two bytes in the previous step, so the new start address is a004006+2 = a004008<br>No.of Bytes: 7A1002=12 Bite=3Bytes,a00400B- a004008=3Bytes |
| **ADDA #$10** | a00400B | 2 | 8B10 | Start Address: since we have three bytes in the previous step, so the new start address is a004008+3 = a00400B<br>No.of Bytes: 8B10=16 Bits=2Bytes, a00400D-a00400B=2Bytes |
| **ADDA #$06** | a00400D | 2 | 8B06 | Start Address: since we have two bytes in the previous step, so the new start address is a00400B+2 = a00400D<br>No.of Bytes: 8B06=16 Bits=2Bytes, a00400F- |

|  |  |  |  |  |
|---|---|---|---|---|
|  |  |  |  | a00400D=2Bytes |
| **STAA SUMB** | a00400F | 3 | 7A1003 | Start Address: since we have two bytes in the previous step, so the new start address is a00400D+2 = a00400F<br>No.of Bytes: 7A1003 =12 Bite=3Bytes ,<br>Here we cannot obtain the number of Bytes by using the difference method since the next address is unknown. |

## 3. Conclusion

We are able to achieve all the goals of the Laboratory experiment 1 and now we are familiar with using CodeWarrier program for executing assembly programs. Also, we are now able to editing, assembling, simulating, downloading and subsequently. Moreover, we used the Dragon Plus for detecting the changes in the memory in each step. Finally, we end up this experiment with grateful result and without facing any problems.

## 4. Assignment Questions and Answers

1)

|  | **Instructions** | **C** | **V** | **H** | **N** | **Z** | **Acc A** | **Acc B** | **Comment** |  |
|---|---|---|---|---|---|---|---|---|---|---|
| **A** | LDAA #$A5 |  |  |  | 1 |  | A5 |  | $A5 - > A | Negative |
|  | ADDA #$89 | 1 | 1 |  |  |  | 2E |  | A +$89 - > A | Carry + Over flow |
| **B** | LDAA #$80 |  |  |  | 1 |  | 80 |  | $80 - > A | Carry + Negative |
|  | ADDA #$80 | 1 | 1 |  |  | 1 | 0 |  | A + $80 - > A | Carry + Overflow + Zero |
| **C** | LDAB #$A5 |  |  |  | 1 |  | CB | A5 | $A5 - > A | Negative |
|  | SUBB #$68 |  | 1 |  |  |  | CB | 3D | B - $68 - > B | Over flow |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **D** | LDAB #$65 | | | | | | CB | 65 | $65 - > B | |
| | SUBB #$65 | | | | | 1 | CB | 0 | B – $65 - > B | Zero |

2)

```
; Include derivative-specific definitions
          INCLUDE 'mc9s12dg256.inc'

; export symbols
    XDEF Entry

; Insert here your data definition.

Entry:

 CLRA
 CLRB
 LDAB    #$F4
 LDAA    #$F2
 MUL
```

| Instruction | C | V | H | N | Z | Acc A | Acc B | Acc D |
|---|---|---|---|---|---|---|---|---|
| CLRA | 0 | 0 | 0 | 0 | 1 | 0 | F4 | F4 |
| CLRB | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| LDAA #$F4 | 0 | 0 | 0 | 1 | 0 | 0 | F4 | F4 |
| LDAA #$F2 | 0 | 0 | 0 | 1 | 0 | F2 | F4 | F2F4 |
| MUL | 1 | 0 | 0 | 1 | 0 | E6 | A8 | E6A8 |

**Comment:** This instruction set multiplies two unsigned 8 bit numbers A and B in register F$ and F2 where they are loaded into. The D register then contains both registers multiplied by each other F2F4, after a unit step it changed to the result in hexadecimal which is E6A8.