Khalifa University of Science, Technology and Research
Electronic Engineering Department

Microprocessor Systems Laboratory

ELCE333

Laboratory Report Experiment No.1

MICROCONTROLLER ASSEMBLY PROGRAM DEVELOPMENT

Lab Partners

Leena ElNeel	100037083
Muna Darweesh	100035522
Shamsah AlNabooda	100036984

Date Experiment Performed: 28/1/2015
Date Lab Report Submitted: 4/2/2015

Lab Instructors:

Mahmoud Khonji

Mohammed Ali Saif Al Zaabi

Spring 2015

Table of Contents

List of Figures and Tables

List of Figures

List of Tables

Summary

This report demonstrates with the use of figures and results the processes taken in this laboratory experiment to edit, assemble, simulate, and then execute an assembly program written using Freescale (Code Warrior IDE). This laboratory experiment includes three tasks. The first task focuses on the idea of creating an assembly code in order to add numbers. As a matter of fact the code was given and what was required is to know how to assemble and simulate the program by single stepping through it to record the content of the registers after each step. The second task focusses on modifying a program so it would allow the subtraction of numbers. Task three shows us how to open a file called the List File which contains many different beneficial details of the program. Task three in general requires the knowledge of techniques to find the memory contents for the developed program. Finally, the report is concluded through discussing the main achievements.

1. Introduction

After being introduced to FreeScale software and CodeWarrior, this experiment encapsulates the whole introduction idea by requiring us to create a file from start to end. CodeWarrior is a program that will be used to create projects and creating programs using assembly with simulating and debugging them. The registers are described and listed below:

Accumulator A: an 8 bit accumulator that is used as a source or destination for 8bit instructions.
Accumulator B: an 8 bit accumulator that is used as a source or destination for 8bit instructions.
Accumulator D: a 16 bit accumulator that is used as a source or destination for 16bit instructions. It is the combined accumulator A and B.
Register X: 16bit register used for addressing and arithmetic operations.
Register Y: 16bit register used for addressing and arithmetic operations.
Stack Pointer: a pointer that ensures the program remains in the RAM.
Program Counter: a counter that shows how much memory can be addressed.
Condition Code Register: 5bit register in which each bit has a significant cause. The bits are:

- **C:** when a borrow or carry occurs
- **V:** when a 2's compliment occurs
- **Z:** when result is zero
- **N:** when the most significant bit is set
- **H:** when a carry or borrow occur in bit three of the result

In assembly language, there are many important operations which are either data transfer instructions or assembly arithmetic instructions. Examples of the instructions are clear, shift, decrement and increment. For instance, we dealt with several mnemonics that represented instructions and such are DEC, INC, CLR, ABA, SBA, NEG, MUL and IDIV. In our lab we are dealing with HCS12 microcontroller; the code written will be saved in the ROM or to more precisely in the Flash EPROM. The variables of the program and the stack will be located in the RAM.

Aim:

The aim of this experiment is to introduce and familiarize students with the process of editing, assembling, simulating, downloading and subsequently executing a simple assembly program.

Objectives:

After doing this experiment, students will be able to:

- Understand how to edit an assembly program on HCS12.
- Generate a binary file and a list file by assembling the program.
- Realize how to download the program binary file to the Dragon Plus Trainer board.
- Learn how to go through the program using single-step.
- Trace a program execution and check the system registers.
- Understand a given program and modify it as required using HCS12 Instruction Set Manual.
- Learn how to realize the content of the program list file.

2. Design and Results

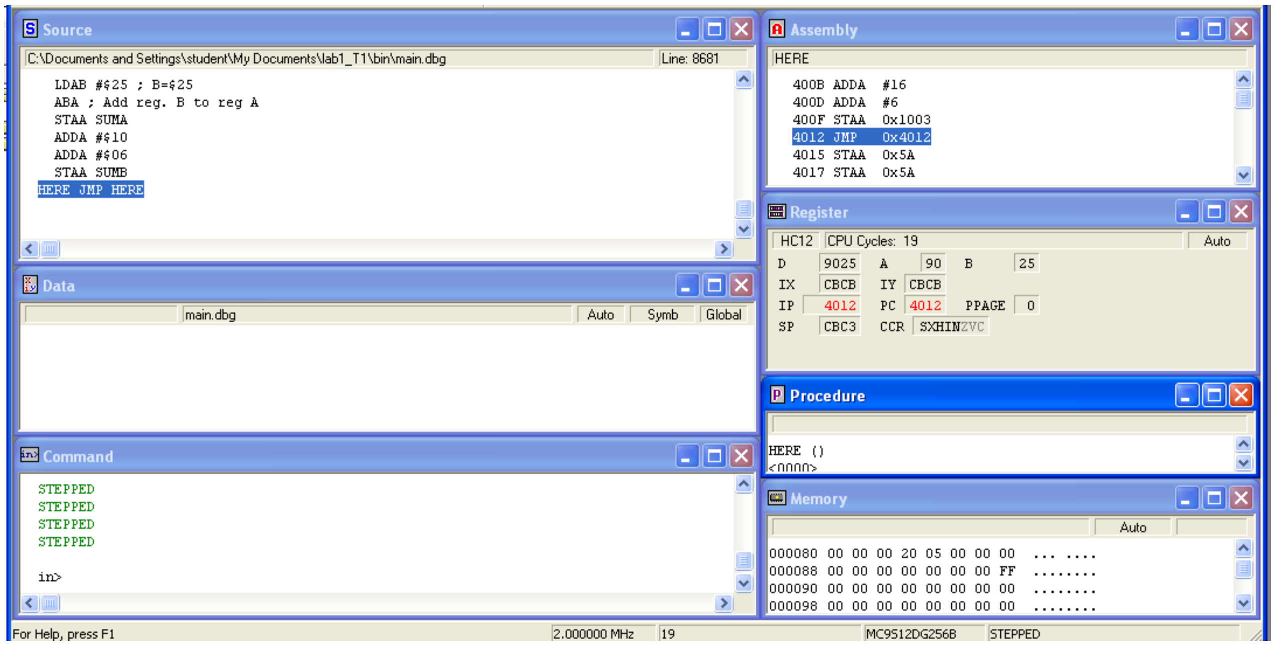
TASK 1: Adding Numbers

Task one summarizes the idea of creating a program from a given code. The code add the contents of the accumulator A and accumulator B after assigning the desired value in accumulator A which is $ 55 (note that $ is for hexadecimal) and for accumulator B which is $ 25. Then, the result stored into memory location $1002. The second part, the contents of accumulator A and $10 are added and the results are stored in accumulator A. After that $06 is added to the contents of accumulator A and the final results is stored in memory location $1003. The code is as the following.

```
          INCLUDE 'derivative.inc'
           XDEF Entry
SUMA         EQU   $1002        ; Location of SUMA
SUMB         EQU   $1003        ; Location of SUMB

             ORG   $4000        ; Set the program counter to$4000
Entry:
             CLRA              ; Clear accumulator A
             CLRB              ; Clear accumulator B
             LDAA #$55          ; the contents of accumulator A is $55
             LDAB #$25          ; the contents of accumulator B is $25
             ABA                ; Add accumulator B to accumulator A
             STAA  SUMA        ; Store the contents of A to memory location $1002
             ADDA  #$10         ; Add A to the immediate value $10 and store result in A
             ADDA #$06           ; Add A to the immediate value $10 and store result in A
             STAA  SUMB          ; Store the contents of A to memory location $1003

HERE         JMP   HERE
```



```
C:\Documents and Settings\student\My Documents\lab1_T1\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
```

```
main.dbg
```

```
STEPPED
STEPPED
STEPPED
STEPPED

in>
```

```
HERE

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

```
HC12   CPU Cycles: 19
D    9025   A    90    B      25
IX   CBCB   IY   CBCB
IP   4012   PC   4012  PPAGE  0
SP   CBC3   CCR  SXHINZVC
```

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

```
000080  00 00 00 20 05 00 00 00   ... ....
000088  00 00 00 00 00 00 00 FF   ........
000090  00 00 00 00 00 00 00 00   ........
000098  00 00 00 00 00 00 00 00   ........
```

```
For Help, press F1    2.000000 MHz    19    MC9S12DG256B    STEPPED
```

Figure 1: Registers at instruction STAA SUMB

We recorded the details required about each of the instructions shown in the code in Table 1.

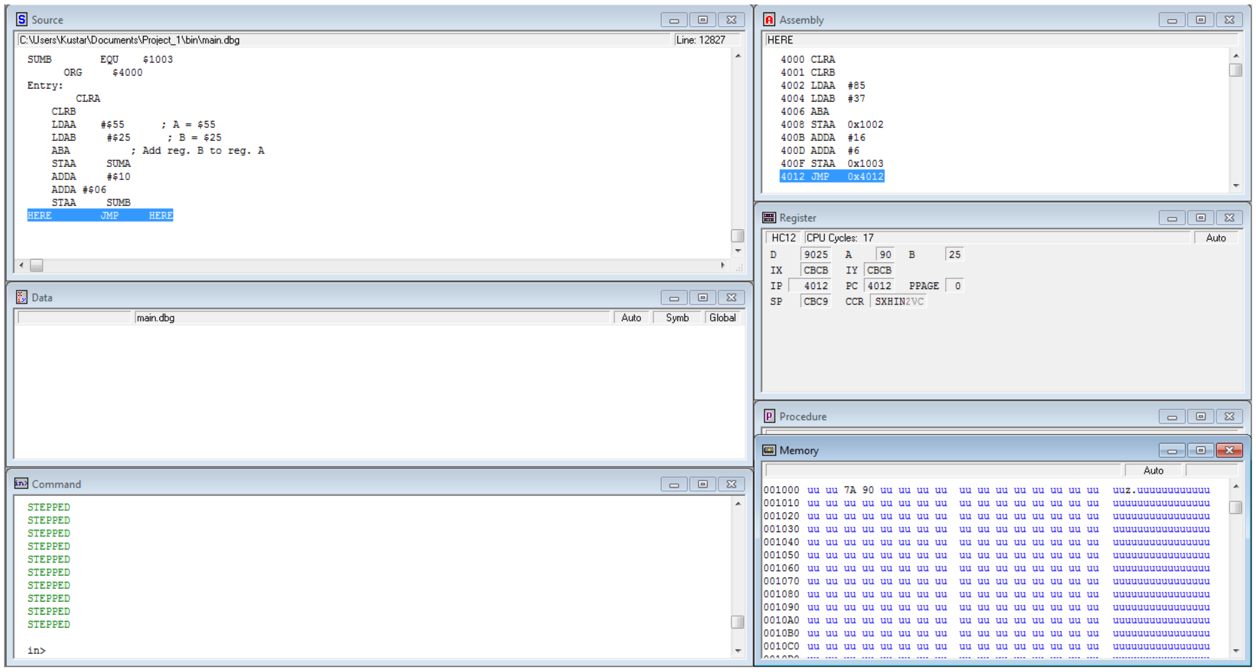
Table 1: Registers contents while single stepping in the program

Instruction	A	B	D	Comments
CLRA	0	25	25	D= 0+25= 0025, Clear acc. A
CLRB	0	0	00	D=0000, Clear acc. B
LDAA #$55	55	0	5500	D=55+0=5500, Load acc. A with the value $55
LDAB #$25	55	25	5525	D=55+25+5525, Load acc. B with the value $25
ABA	7A	25	7A25	D=7A+25=7A25, Add A to B and save the result in A
STAA SUMA	7A	25	7A25	D=7A+25=7A25, Store the value of A at SUMA
ADDA #$10	8A	25	8A25	D=8A+25=8A25, Add the value $10 to A
ADDA #$06	90	25	9025	D=90+25=9025, Add the value $06 to A
STAA SUMB	90	25	9025	D=90+25=9025, Store the value of A at SUMB

Finally, after the program is finished we examined both memory locations $1002 and $1003 and recorded the values before starting and after finishing as shown below in the table and in Figure 2.

Table 2: Memory Locations Contents

	$1002	**$1003**	**Comments**
Before	--	--	The addresses were empty
After	7A	90	The content of A is shown at the address 1002 after adding A&B while the final content of A is shown in 1003, after doing the second set of addition operation



Source — C:\Users\Kustar\Documents\Project_1\bin\main.dbg — Line: 12827

```
SUMB        EQU    $1003
      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
```

Assembly — HERE

```
4000 CLRA
4001 CLRB
4002 LDAA  #85
4004 LDAB  #37
4006 ABA
4008 STAA  0x1002
400B ADDA  #16
400D ADDA  #6
400F STAA  0x1003
4012 JMP   0x4012
```

Register — HC12 — CPU Cycles: 17 — Auto

```
D    9025   A   90    B     25
IX   CBCB   IY  CBCB
IP   4012   PC  4012  PPAGE  0
SP   CBC9   CCR SXHINZVC
```

Data — main.dbg — Auto — Symb — Global

Command

```
STEPPED
STEPPED
STEPPED
STEPPED
STEPPED
STEPPED
STEPPED
STEPPED
STEPPED
STEPPED

in>
```

Memory — Auto

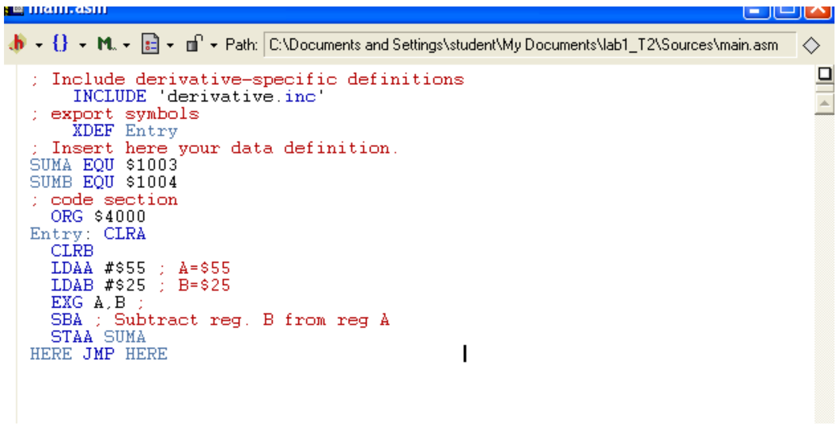
```
001000 uu uu 7A 90 uu uu uu uu  uu uu uu uu uu uu uu uu  uuz.uuuuuuuuuuuu
001010 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
001020 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
001030 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
001040 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
001050 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
001060 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
001070 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
001080 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
001090 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
0010A0 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
0010B0 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
0010C0 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
```

Figure 2: The memory contents for $1002 and $1003 memory address

Comment: The addresses $1002 and $1003 were assigned for the labels SUMA and SUMB. In 1002, the addition of $ 55 (decimal value: 85) and $ 25 (decimal value: 37) is 7A (decimal value: 122). Then, the result stored into memory location $1002, as shown above. For the second set, the address $1003, $ 55+ $ 25=$ 80 and then this is added to $ 10 which will result in $ 90, as shown above.

TASK 2: Subtraction Numbers

Task two is about modifying the code given in task one –which adds numbers- to create a program that subtracts numbers instead. We used the SBA instruction to subtracted and final results were saved in $1003. We did not use the memory location $1004 because we exchanged the contents of A and B. Figure 3 below shows the modified code.

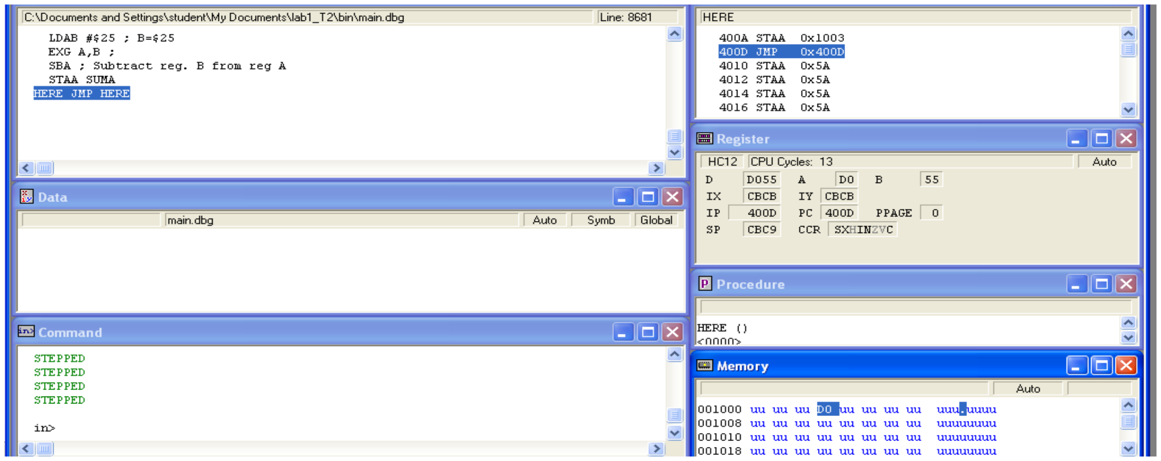


Path: C:\Documents and Settings\student\My Documents\lab1_T2\Sources\main.asm

```
; Include derivative-specific definitions
    INCLUDE 'derivative.inc'
; export symbols
    XDEF Entry
; Insert here your data definition.
SUMA EQU $1003
SUMB EQU $1004
; code section
  ORG $4000
Entry: CLRA
  CLRB
  LDAA #$55 ; A=$55
  LDAB #$25 ; B=$25
  EXG A,B ;
  SBA ; Subtract reg. B from reg A
  STAA SUMA
HERE JMP HERE
```

Figure 3: Subtraction code

Figure 4 shows the results we obtained from this task. Using the single step facility, we were able to view the registers at each instruction and when tested, the output we got was D0 which gives a hexadecimal value in correspondence to the subtraction value. As seen in the memory window, the $1000 row and fourth column shows the location of $1003 which as previewed is D0.



C:\Documents and Settings\student\My Documents\lab1_T2\bin\main.dbg   Line: 8681

```
   LDAB #$25 ; B=$25
   EXG A,B ;
   SBA ; Subtract reg. B from reg A
   STAA SUMA
HERE JMP HERE
```

Data: main.dbg   Auto   Symb   Global

Command

```
STEPPED
STEPPED
STEPPED
STEPPED

in>
```

HERE

```
400A STAA  0x1003
400D JMP   0x400D
4010 STAA  0x5A
4012 STAA  0x5A
4014 STAA  0x5A
4016 STAA  0x5A
```

Register

```
HC12  CPU Cycles: 13                 Auto
D    D055   A    D0     B     55
IX   CBCB   IY   CBCB
IP   400D   PC   400D   PPAGE  0
SP   CBC9   CCR  SXHINZVC
```

Procedure

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

Memory   Auto

```
001000 uu uu uu D0 uu uu uu uu   uuu.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
```

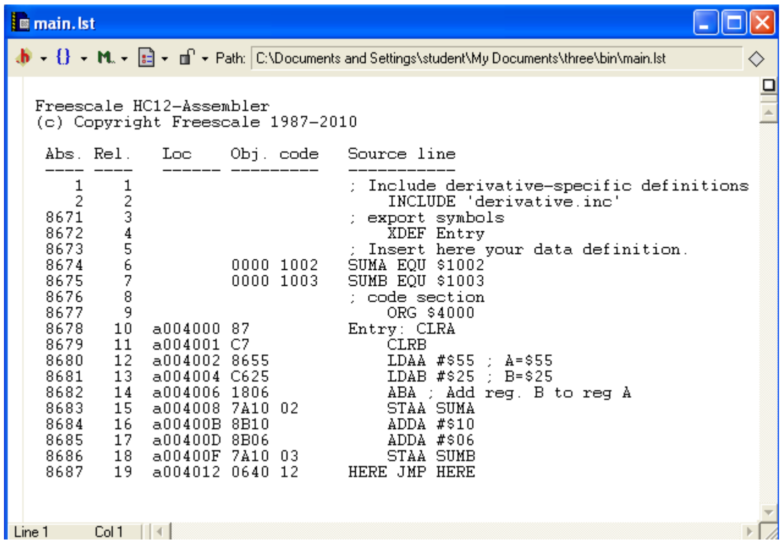
Figure 4: Result of subtraction program

Comment: The addresses $1003 has the contents of D0 which is the result of the subtraction of accumulator A and accumulator B.

TASK 3: Simple Program

Task three concepts about dealing with list files with extension ".lst". This file provides the following with the source-code program, The assembled code in hexadecimal format (including memory locations), any error messages and a list of user-defined symbols used and their values.

We were asked to copy the same given code in task 1 and use the "create a listing file" checkbox to create the file we need. It is saved in the "bin" file and opened from that source. When we opened the file, several columns appeared. A screenshot of the file was taken and shown in Figure 4 in the next page. We were required to note down several details about every instruction and the information are jotted in Table 3.


main.lst
Path: C:\Documents and Settings\student\My Documents\three\bin\main.lst
Line 1
Col 1

```
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
```

Figure 5: list file

The table shows that the ORG instruction forces the program to start from address 4000. The difference between the start address of the required instruction and the next instruction shows the number of bytes.

Table 3: The memory contents for the developed program:

Instruction	Start Address	No. of Bytes	Instruction Code
CLRA	a004000	1	87
CLRB	a004001	1	C7
LDAA #$55	a004002	2	8655
LDAB #$25	a004004	2	C625
ABA	a004006	2	1806
STAA SUMA	a004008	3	7A1002
ADDA #$10	a00400B	2	8B06
ADDA #$06	a00400D	3	7A1003
STAA SUMB	a00400F	3	064012

Comment: The list file shows the operation that must be done and the operands that are being operated upon or each instruction in the assembly language. If we take the third instruction in the table which is LDAA #$55, the first byte which is 86 is a unique code for each operation to be executed by the microcontroller; this is known as the opcode byte. The second byte is for the operands of the operation.

3. Assignment Questions

1. Find the status of CCR flags - C, V, H, N, Z and Acc A after executing A, B, C and D individually and comment on the results.

Table 4- Results of simulation for question 1

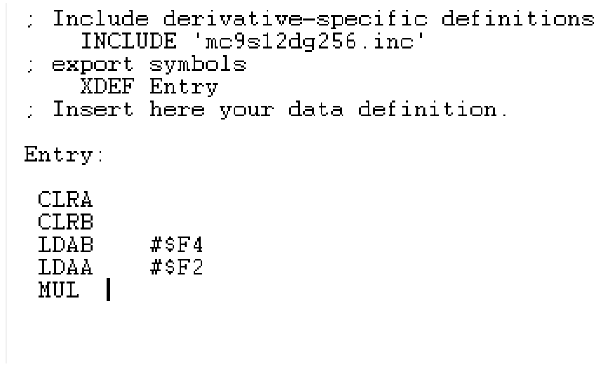
	Instructions	**C**	**V**	**H**	**N**	**Z**	**Acc A**	**Acc B**	**Comment**
A	**LDAA #$A5**	-	0	0	1	0	A5	▨	➢ *Addressing mode*: Immediate ➢ *Operation*:($A5) → A ➢ *C*:Executing LDAA instructions doesn't change the carry bit ➢ *V:* 2's complement overflow is set to zero ➢ *H:* no half carry occurred ➢ *N*:Because the msb is 1 $(A5)_{16}=(10100101)_2$ ➢ *Z:* The value contents of acc. A isn't zero
	ADDA #$89	1	1	0	0	0	12E	▨	➢ *Addressing mode*: Immediate ➢ *Operation*: ($A5) + ($89) → A ➢ *C*:the result of the instruction is $(12E)_{16}=(100101110)_2$ which is 9 bits so we have a carry ➢ *V:* the binary representation shows a 2's complement overflow ➢ *H:* no half carry occurred ➢ *N*:the msb (the 8th bit) is zero ➢ *Z:* The value contents of acc. A isn't zero
B	**LDAA #$80**	-	0	0	1	0	80	▨	➢ *Addressing mode*: Immediate ➢ *Operation*:($80) → A ➢ *C*:Executing LDAA instructions ➢ doesn't change the carry bit

								▨	➢ *V:* 2’s complement overflow is set to zero ➢ *H:* no half carry occurred ➢ *N*:msb is zero $(80)_{16}=(10000000)_2$ ➢ *Z:* The value contents of acc. A isn’t zero
	ADDA #$80	1	1	0	0	1	0	▨	➢ *Addressing mode*: Immediate ➢ *Operation*: ($80) + (80) → A ➢ *C*:the result of the instruction is $(100)_{16}=(100000000)_2$ which is 9 bits so we have a carry ➢ *V:* the binary representation shows a 2’s complement overflow ➢ *H:* no half carry occurred ➢ *N*:the msb (the 8th bit) is zero ➢ *Z:* the bits of acc. A are zeros and the 9th overflow bit is stored in next memory address
C	**LDAB #$A5**	-	0	0	1	0	▨	A5	➢ *Addressing mode*: Immediate ➢ *Operation*:($A5) → B ➢ *C*:Executing LDAB instructions doesn’t change the carry bit ➢ *V:* 2’s complement overflow is set to zero ➢ *H:* no half carry occurred ➢ *N*:Because the msb is 1 $(A5)_{16}=(10100101)_2$ ➢ *Z:* The value contents of acc. A isn’t zero
	SUBB #$68	0	1	0	0	0	▨	3D	➢ *Addressing mode*: Immediate ➢ *Operation*: ($A5) - (68) → B ➢ *C*:no carry $(3D)_{16}=(00111101)_2$ ➢ *V:* 2’s complement is used for this instruction ➢ *H:* no half carry occurred

									➢ *N*:the msb (the 8th bit) is zero *Z:* the result isn't zero
D	**LDAB #$65**	-	0	0	0	0		65	➢ *Addressing mode*: Immediate ➢ *Operation*:($65) $\rightarrow$ B ➢ *C*:Executing LDAB instructions doesn't change the carry bit ➢ *V:* 2's complement overflow is set to zero ➢ *H:* no half carry occurred ➢ *N*:Because the msb is 1 $(65)_{16}=(01100101)_2$ ➢ *Z:* The value contents of acc. A isn't zero
	SUBB #$65	0	0	0	0	1		0	➢ *Addressing mode*: Immediate ➢ *Operation*: ($65) – ($65) $\rightarrow$ B ➢ *C*:no carry occurred ➢ *V:* no 2's complement overflow occurred ➢ *H:* no half carry occurred ➢ *N*: msb is 0 $(0)_{16}=(00000000)_2$ ➢ *Z:* The value contents of acc. A is zero

2. Write an instruction sequence to multiply two 8-bit numbers using MUL instruction (unsigned).Initialize Accumulator A and Accumulator B with $F4 and $F2 respectively using the load instructions. Highlight the result in the D register and the CCR flags.

The code:



```
; Include derivative-specific definitions
    INCLUDE 'mc9s12dg256.inc'
; export symbols
    XDEF Entry
; Insert here your data definition.

Entry:

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

Table 5- Results of simulation for question 2

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

4. Conclusion

In the first task it was required to assemble and simulate a code for addition using ABA. Same program of task one was modified in the second task in order to do the subtraction operation and the results are stored in specific memory location.

The purpose of the last task was to learn some further details and features about the assembled program. Start Address, number of bytes, Instruction Code are essential tools in every program for debugging and finding the errors in the program. Actually, a computer instruction is the combination of an operation (what the computer is to do) and zero, or more operands (what the computer is going to do to it).