

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

**Electronic Engineering Department**

**ELCE332 Microprocessor Systems laboratory**

**Laboratory Experiment 1**

**Microcontroller Assembly Program Development**

**Lab Partners**

Hamad Eisa Alhazami 100036985

Jasim Alhammadi 100035510

**Date Experiment Performed**: 28-1-2015

**Date Lab Report Submitted**: 4-2-2015

**Lab Instructor: Mr. Mohammed Alzaabi/ Mr. Mahmoud Khounji**

**Spring 2015**

Contents

[List of figures and tables 3](#_Toc410818627)

[Summary 4](#_Toc410818628)

[1. Introduction 5](#_Toc410818629)

[1.1 Aim 5](#_Toc410818630)

[1.2 Objectives 5](#_Toc410818631)

[2. Design and results 6](#_Toc410818632)

[**TASK - 1: Adding Numbers** 6](#_Toc410818633)

[**TASK - 2: Subtracting Numbers** 8](#_Toc410818634)

[**TASK - 3: Simple Program:** 10](#_Toc410818635)

[3. Assignment Questions 12](#_Toc410818636)

[4. Analysis and Interpretation 14](#_Toc410818637)

[5. Conclusions and Recommendations 15](#_Toc410818638)

# List of figures and tables

List of figures:

[Figure 1: First task generated list file 10](#_Toc410818737)

[Figure 2: screenshot of accumulators A, B, and D and CCR 13](#_Toc410818738)

List of tables:

[Table 1: Registers contents while single stepping in the program 7](#_Toc410818757)

[Table 2: Registers contents while single stepping in the program 9](#_Toc410818758)

[Table 3: The memory contents for the developed program 11](#_Toc410818759)

# Summary

The purpose of this report is to document the tasks conducted in the laboratory session that focused on editing, assembling, downloading and executing an assembly program. There are three main tasks in this experiment, the first task was basically using the CodeWarrior program to develop a new project and editing it into the shell of an assembly language program. The second task was about manipulating the code developed in the first task in such a way that it deducts bytes from different Accumulators. The third and final task was an introduction to a technique that aids in error detection. This technique is simply producing a list file; the list file can be used for debugging (finding and detecting errors) of assembly language programs. The report is concluded with the conclusions obtained after conducting the experiment and further recommendations.

# 1. Introduction

To begin with, a microcontroller is an integrated circuit composed of four components and two buses. The components are a Central Processing Unit (CPU), a non-volatile Read Only Memory (ROM), a volatile Random Access Memory (RAM) and Input/ Output (I/O) Interface with a data and address buses.

In this experiment, it is required to use the CodeWarrior IDE software to conduct the process of editing, assembling, downloading and executing an assembly codes for HCS12 Microcontroller.

1.1 Aim

The purpose of this experiment is to get familiarized on the process of editing, assembling, simulating, downloading and executing an assembly program.

1.2 Objectives

Edit and develop a microcontroller HCS12 program

Trace the program execution using single step debugging and check the system registers and memory contents.

Produce a list file and binary file from the assembly code

Understand the content of list file produced

Download binary file into Dragon Plus board

Use the HCS12 instruction manual as appropriate

# 2. Design and results

This part is divided into three parts, each part discusses and demonstrates a task and the steps taken to achieve it.

### **TASK - 1: Adding Numbers**

Starting with the first task, we created a new project and changed its program with the required program in the shell of the assembly language as it below:

**;Include derivative-specific definitions**

**INCLUDE 'derivative.inc'**

**;export symbols**

**XDEF Entry**

**;insert here your data definition.**

**SUMA EQU $1002**

**SUMB EQU $1003**

**;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**

The value of addresses $1002 and $1003 were checked before running the program. They were assigned to default values as (uu). For the modification, they were assigned to SUM1 and SUM2 in the code for registers $1002 and $1003 consecutively. The program was run step by step to observe carefully the value of Accumulators A, B and D for each instruction in the code. Values of the Accumulators mentioned before are shown in table 1. The values for registers $1002 and $1003 were modified after running the code to $1002= 7A , and $1003=90. Comments are shown as well for each instruction.

Table 1: Registers contents while single stepping in the program

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instruction | A | B | D | Comments |
| CLRA | 00 | CB | 00CB | Acc. A cleared |
| CLRB | 00 | 00 | 0000 | Acc. B cleared |
| LDAA #$55 | 55 | 00 | 5500 | A = 25 |
| LDAB #$25 | 55 | 25 | 5525 | B =55 |
| ABA | 7A | 25 | 7A25 | A + B 🡪 A=7A |
| STAA SUMA | 7A | 25 | 7A25 | store A in SUMA $1002 = 7A |
| ADDA #$10 | 8A | 25 | 8A25 | $10+A 🡪 A = 8A |
| ADDA #$06 | 90 | 25 | 9025 | $06+A 🡪 A = 90 |
| STAA SUMB | 90 | 25 | 9025 | store A in SUMB $1003 = 90 |

### **TASK - 2: Subtracting Numbers**

For the second task, it is required to modify the program in task 1 in which the byte in accumulator A is subtracted from the byte in B using SUB command. The assembly does not have a direct command to subtract the value of A from B. Thus EXG command is used to swap the values of A and B and then perform the subtraction. The code is shown below:

**; 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 ; Add reg. B to reg A**

**STAA SUMA**

**STAA SUMB**

**HERE JMP HERE**

The values of the registers $1003 and $1004 were checked before running the program and they were assigned to default (uu). Also they were assigned to the values SUMA and SUMB in the code consecutively. The program was run step by step to observe carefully the value of Accumulators A, B and D for each instruction in the code. Values of the Accumulators mentioned before are shown in table 1 as well as the comment for each instruction.

Table : Registers contents while single stepping in the program

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instruction | A | B | D | Comments |
| CLRA | 00 | CB | 00CB | Acc. A cleared |
| CLRB | 00 | 00 | 0000 | Acc. B cleared |
| LDAA #$55 | 55 | 00 | 5500 | Load acc. A = 55 |
| LDAB #$25 | 55 | 25 | 5525 | Load acc. B = 25 |
| EXG A,B | 25 | 55 | 2555 | Swap values of acc. A and acc. B |
| SBA | D0 | 55 | D055 | Subtract the value of acc. B from acc. A |
| STAA SUMA | D0 | 55 | D055 | Store the value of acc. A in $1003 = D0 |
| STAA SUMB | D0 | 55 | D055 | Store the value of acc. A in $1004 = D0 |

### **TASK - 3: Simple Program:**

In the third task, it is required to produce the list file that contains:

* 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 list file for task 1 is created by following the steps mentioned in the lab script. The figure in file below shows the list of the instructions and its details:

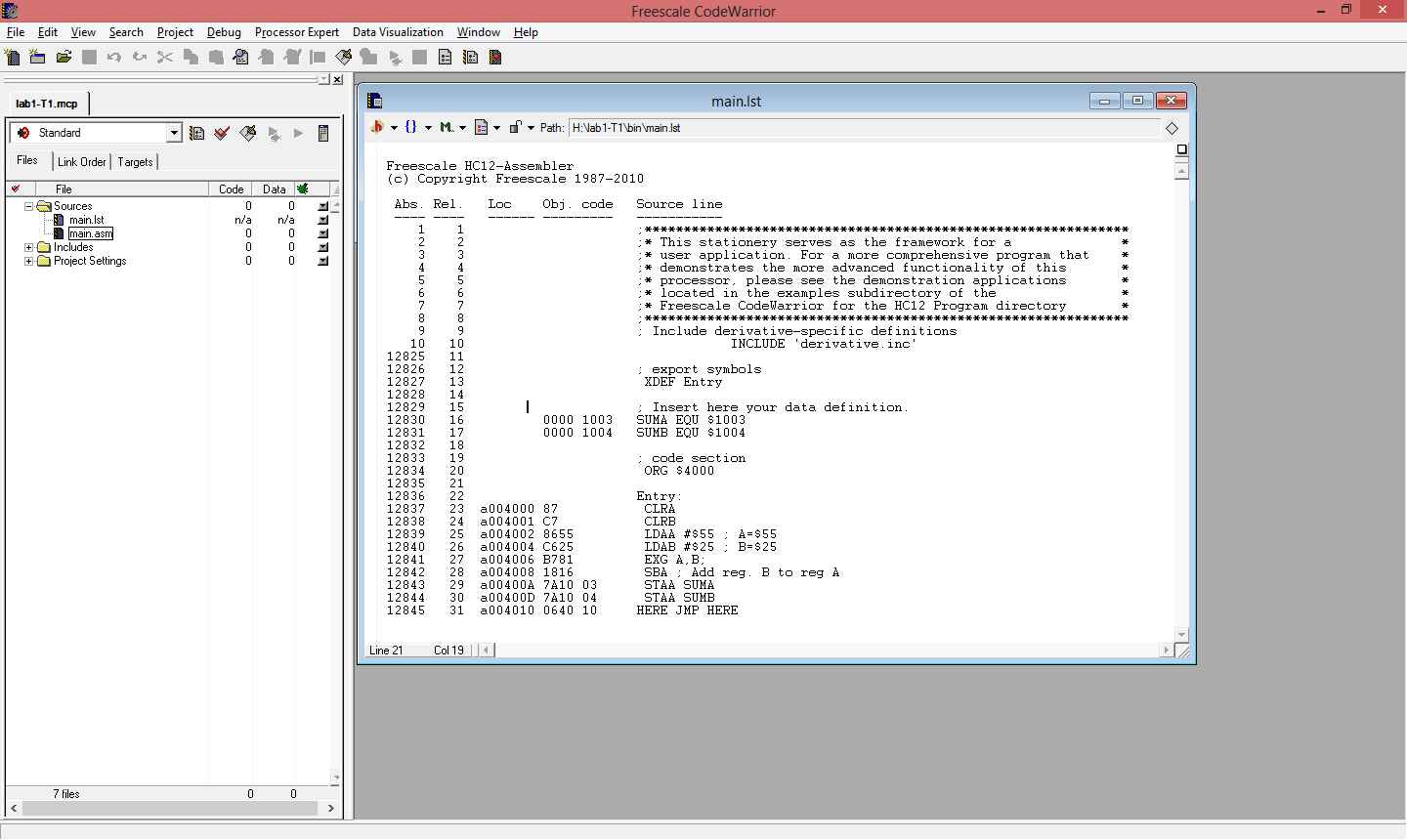


Figure 1: First task generated list file

The results have been studied in order to analyze each instruction to follow the table below:

Table 3: The memory contents for the developed program

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instruction | Start Address | No. of Bytes | Instruction Code | Comments |
| CLRA | a004000 | 1 | 87 | 1 Byte, it does not use memory location |
| CLRB | a004001 | 1 | C7 | 1 byte, it does not use memory location |
| LDAA #$55 | a004002 | 2 | 8655 | 2 Bytes:  1 Byte op-code  1 Byte operand |
| LDAB #$25 | a004004 | 2 | C625 | 2 Bytes:  1 Byte op-code  1 Byte operand |
| ABA | a004006 | 2 | 1806 | 2 Bytes:  1 Byte op-code  1 Byte operand |
| STAA R1 | a004008 | 3 | 7A10 02 | 3Bytes,  1Byte op-code  2Bytes operand |
| ADDA #$10 | a00400B | 2 | 8B10 | 2 Bytes:  1 Byte op-code  1 Byte operand |
| ADDA #$06 | a00400D | 2 | 8B06 | 2 Bytes:  1 Byte op-code  1 Byte operand |
| STAA R2 | a00400F | 3 | 7A10 03 | 3Bytes,  1Byte op-code  2Bytes operand |

The list enables the students to observe instructions and how the program started from the address 4000 by the instruction ORG. The length of each instruction can be noticed by the difference in the address i.e. the first address is 4000 and its length 1 byte, thus, the second address will be 4001.

# 3. Assignment Questions

* 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.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | instructions | C | V | H | N | Z | Acc. A |  |
| A | LDAA #$A5 | 0 | 0 | 0 | 1 | 0 | A5 | Negative number |
| ADDA #89 | 1 | 1 | 0 | 0 | 0 | 2E | Carry + Overflow |
| B | LDAA #$80 | 1 | 0 | 0 | 1 | 0 | 80 | Carry + negative number |
| ADDA #$80 | 1 | 1 | 0 | 0 | 1 | 0 | Carry + Overflow + Negative number |
| C | LDAB #$A5 | 1 | 0 | 0 | 1 | 0 | 0 | Carry +­ Negative number |
| SUBB #$68 | 0 | 1 | 0 | 0 | 0 | 0 | Overflow |
| D | LDAB #$65 | 0 | 0 | 0 | 0 | 0 | 0 | - |
| SUBB #$65 | 0 | 0 | 0 | 0 | 1 | 0 | zero result |

* 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.

**;Include derivative-specific definitions**

**INCLUDE 'derivative.inc'**

**;export symbols**

**XDEF Entry**

**;insert here your data definition.**

**SUMA EQU $1002**

**SUMB EQU $1003**

**;code section**

**ORG $4000**

**Entry: CLRA**

**CLRB**

**LDAA #$F4 ; A = $F4**

**LDAB #$F2 ; B = $F2**

**MUL**

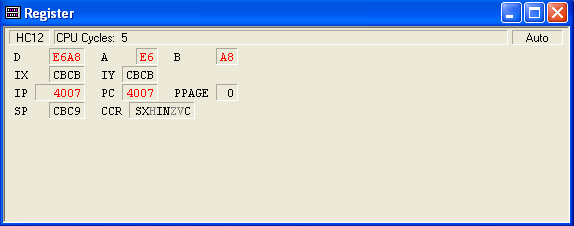
****

Figure 2: screenshot of accumulators A, B, and D and CCR

The result in Acc. D is E6A8 (59048 in decimal) which is the result of multiplying F4 (244 in Decimal) in Acc. A and F2 (242 in Decimal) in Acc. B. The CCR registers H,Z, and V are highlighted.

# 

# 4. Analysis and Interpretation

The first task highlighted the usage of CodeWarrior program to develop a new project and edit it into the shell of an assembly language program. Also it focused of tracing each instruction and noticing the change in accumulators and registers.

The second task was about manipulating the code developed in the first task in such a way that it deducts bytes from different Accumulators. It highlighted the usage of data-movement instructions in addition to the data modification instructions in order to achieve the operation purpose.

The third and final task was an introduction to a technique that aids in error detection. This technique is simply producing a list file; the list file can be used for debugging (finding and detecting errors) of assembly language programs. Each instruction has its own address, and size. The size can be calculated from the address by looking at the next instruction’s address and calculate the difference.

5. Conclusions and Recommendations

The main achievements of this experiment were learning how to edit, assemble, simulate, download and then execute an assembly program using an assembly source code of microcontroller HCS12 and learning how to create a list file relevant to any code. The list file contains information about the memory locations and the Op-code of the instructions.