Lab 1
MC9S12 Assembler and Monitor

Introduction and Objectives

The purpose of this lab is to help you become familiar with your Dragon12-Plus Evaluation Board (EVB), and some of the software tools which you will use to write programs for this course. This laboratory introduces you to the following MC9S12 assembly language programming tools:

- Freescale CodeWarrior.
- The D-Bug12 monitor that runs on the MC9S12.

An assembler takes assembly language code in a form that a human can reasonably read and write (with a little practice), translates it to machine code which a microprocessor can understand, and stores the machine code in a file which can be uploaded to the microprocessor (Motorola uses a format called “S19” for its machine code files). In this lab we will use the Freescale CodeWarrior, which is on the PCs in the Digital/Microcontrollers lab. The assembler of the CodeWarrior produces an output file with an .s19 extension which can be loaded into and run on the MC9S12. The D-Bug12 monitor, running from MC9S12 Flash memory, loads S19 records into the MC9S12 and provides some tools for debugging loaded programs.

1. Prelab

You should read this entire lab and **answer all of the questions in the pre-lab section before coming to lab**.

The D-Bug12 monitor allows you to interact with the MC9S12 microcontroller. You can use it to load programs, to find out what is in the MC9S12s registers and memory, to change the contents of the registers and memory, and many other things. The D-Bug12 commands of interest for this lab are:

ASM, BF, BR, NOBR, G, HELP, LOAD, MD, MDW, MM, MMW, RM, RD, and T.

Read the descriptions of these commands in Chapter 5 of the D-Bug12 Reference Guide, or in Sections 3.5 and 3.6 of Huang.

1.1 Questions to Answer Before Lab

1. What is the difference between the MM and MMW commands?

2. How would you change the value of the X accumulator to 0x55AA? (There are several ways to do this; you only need to find one way.)

3. The Dragon12-Plus has a two-line LCD display, and there is a way for you to identify your board by putting your name (up to 16 characters) on the first line of the display. You can do this by converting your name into ASCII, and putting these ASCII data into a region of the EEPROM on your board. (If your name has less than 16 characters, pad your name –at the beginning and/or end –with spaces.)

Convert you name to ASCII. For example, Jane Smith would become 0x4A 0x61 0x6E 0x65 0x20 0x53 0x6D 0x69 0x74 0x68 0x20 0x20 0x20 0x20 0x20 0x20. You will enter these ASCII characters into you board in lab in Step 7 of Section 2.1.

4. What will be the contents of the A register after each instruction of the program shown in Program 1 executes?

Program 1 An assembly language program used to get started with HyberTerminal and D-Bug12.

```
; MC9S12 demo program
; EE 308

; This is a program to add four numbers in memory from $1000 through $1003,
; divide the sum by four, and store the result in address $1004
prog:   equ     $2000           ; Starting address from program
data:   equ     $1000           ; Starting address for data
        org     prog            ; Set initial program counter value
        ldaa    input1          ; Load first number into ACCA
        adda    input2          ; add second number
        adda    input3          ; add third number
        adda    input4          ; add fourth number
        lsra                    ; divide by 2
        lsra                    ; divide by 2
        staa    average         ; save result in memory
```

```
        swi

        org     data        ; Put data starting at this location
input1: dc.b    $32         ; First number
input2: dc.b    $45         ; Second number
input3: dc.b    $3c         ; Third number
input4: dc.b    $2f         ; Fourth number
average: ds.b   1           ; Reserve one byte for results
```

2. The Lab

We will use D-Bug12 to explore the memory of the MC9S12 on your evaluation board. The memory map for your MC9S12 is shown on page 24 of the MC9S12DP256B Device Users Guide (zip file). (Look for the figure labeled Normal Single Chip.) Some of the memory is used by the D-Bug12 monitor. For this class you can use RAM from 0x1000 to 0x3BFF, and EEPROM from 0x0400 to 0x0EFF. Normally, you should use the RAM from 0x2000 through 0x3BFF for programs, and from 0x1000 through 0x1FFF for data. In later labs, you will load programs into the EEPROM so the programs will remain on you board after cycling the power. Do not use RAM from 0x3C00 through 0x3FFF or EEPROM from 0x0F00 through 0x0FFF.

On your account on the EE network, create a directory for this course (say, U:\EE308). Create a project using the following steps:

1. Start CodeWarrior.

2. Start a new project.

3. Under Device and Connection:

   • Select the appropriate device which is *HCS12D Family:MC9S12DP256B*.

   • Under connections select *Full Chip Simulation*.

4. Under Project Parameters:

   • Choose an appropriate name for the project and place it in U:\EE308\LAB01).

   • unselect *C* and select *Absolute assembly*.

5. Click Finish.

6. Under Edit/Standard Settings/Target/Assembler for HC12: Click on options and select *Generate a listing file*.

7. In the left-hand panel will be a tree of the project with a file called *main.asm*. Double-click on *main.asm*, and edit the file to match Program 1. (You can edit the file at home, and replace the main.asm file in the Sources subdirectory with that file).

8. Save and assemble the file. (To assemble the file, click on the Project drop-down menu, and select Make.) This will generate a Project.abs.s19 file that you can download onto your Dragon12 board and a Project.abs.lst which is a listing of your program after making the project. The two files are in the bin subdirectory.

9. Edit the Project.abs.s19 file, remove the first line then save it.

2.1 Answer the Following During Lab

Here are some questions on the output of the assembler. Be sure to answer these questions in you lab book.

1. Look at the lab01.lst file. Where in memory will the machine code for the instruction staa average be stored?

2. What machine code is generated for the staa average mnemonic? Is this what you expected? (Look up the STAA instruction in MC9S12 CPU12 Core Users Guide, to determine what code this instruction should generate.)

3. At what address will the variable *average* be located in the MC9S12 memory?

Connect your Dragon12-Plus board to your computer using the included serial cable. Make sure HyperTerminal is running with the following setting and then power up the Dragon12-Plus board. (Note: In the Digital Lab, the built-in USB-to-Serial adapter on the Dragon12-Plus board will appear as com4. If you conned the adapter to your personal computer, it may appear with a different number. If you do not have HyperTerminal on your personal computer, you can use a different program, such AsmIDE and MiniIDE as discussed in the text.)

- **serial port – com4**
- **9600 Baud**
- **N81 (no-parity - 8bits data - 1 stop bit).**

- **Xon/Xoff flow control**

4. Use the **MM** command to put 0x55 into memory location 0x2000 and 0xAA into memory location 0x2001. Use the MD command to verify that this was done. To exit this mode, type a period before hitting ENTER.

5. Use the **MMW** command to put 0x55 into memory location 0x2100 and 0xAA into memory location 0x2101. Use the **MD** command to verify that this was done.

6. Use the **BF** command to load 0x55 into memory locations 0x2800 to 0x2FFF. Use the **MD** command to verify that it worked.

7. Use the **MM** command to put the ASCII value of **W** into address 0xFDE and the ASCII value of C into address 0xFDF. Then put the ASCII representation for your name into addresses 0xFE0 through 0xFEF. Use the **MD** command to verify these addresses hold the correct values. Push the Reset button. Your name should now be in the first line of the LCD display.

8. Use the **ASM** 2000 command to enter the following simple program at address 0x2000. (What does this program do?)

```
ldaa    #$1C
adda    #$A5
clrb
decb
staa    $1000
stab    $1001
swi
```

9. You can trace (execute your program one step at a time) through your program by setting the **PC** (Program Counter) to the address of the first instruction of your program, and then use the **T** (Trace) command. Use the **RM** command to change the value of the Program Counter to 0x2000, the address of the first instruction of the simple program. Trace through your program and observe what is happening to the registers and memory. (Use the **MD** command to display memory.) Verify that the values in A and B are what you expect after each instruction.

10. Use the **G 2000** command to run your entire program.

11. Load your program lab01.s19 into the EVB. To do this, type **LOAD** into the terminal window, then use the Transfer:Send Text File. . .menu option to send the program to the EVB.

12. In the Terminal window type **ASM 2000** followed by the ENTER key. You should see the first instruction of your program, along with its address and machine code. Each time you hit ENTER you should see the next instruction in the program. To exit this mode, type a period before hitting ENTER.

13. Trace through your program. How do the contents of the A register compare to what you predicted in the Pre-lab after the execution of each instruction?

14. When you are done tracing through your program, reload it and run the entire program by giving the command: **G 2000**. Verify that the program worked correctly.

15. When debugging a long program, it is impractical to step through the entire program to get to the section of code which is giving problems. A breakpoint is a way to run a program up to the point in the code you want to debug, and stop there. After stopping at the breakpoint, you can then trace through your code to try to resolve the problem. You can set a breakpoint with the **BR** command. The format is **BR xxxx**, where **xxxx** is the address you want to break at. Find the address of the first lsra instruction, and set a breakpoint there. Give the command **G 2000**, and your program should run until it is ready to execute the first lsra instruction. You can trace through your program after that to see what the instructions do. Do this. When done, use the **NOBR** command to remove the breakpoint.

16. Change the two lsra instructions to asra instructions. Rerun your program. Does it give a different result? Why?