EE 308 – 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:

• The Freescale MC9S12 assembler **as12**. This runs on the PC.

• The D-Bug12 monitor that runs on the MC9S12.

• The AsmIDE Integrated Development Environment used for assembling programs and interacting with your EVB.

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 **.s19** file which the MC9S12 monitor program can understand. In this lab we will use the Freescale **as12** assembler, which is on the PCs in the Digital/Microcontrollers lab. The as12 assembler is also included on the CD-ROM which comes with your Dragon12-Plus board. The as12 assembler 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.

Pre-Lab

You should read this entire lab and answer all of the questions in the pre-lab section before coming to lab. The pre-lab questions are repeated on a separate page to help you prepare for 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 MC9S12's 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 http://www.ee.nmt.edu/~rison/datasheets/DB12RG4.pdf, or in Sections 3.5 and 3.6 of Huang.

Questions to answer before lab:

1. What is the difference between the **`MD`** and **`MDW`** commands?

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

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.

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

Programs for the MC9S12 can be written in assembly language. An assembly language instruction will be converted by the assembler into a single machine instruction for the MC9S12. The assembly language instructions of interest to us for this lab are:

`LDAA, STAA, STAB, ADDA, LSRA, ASRA, CLRB` and **`DECB`**

Read the descriptions of these commands in the **S12CPUV2** manual (**http://www.freescale.com/files/microcontrollers/doc/ref_manual/S12CPUV2.pdf**). At this point you will not understand everything the manual says about these instructions, but you should understand enough to get you through this lab.

Figure 1 is a simple program for the MC9S12. We will use it to illustrate how to use the assembler, the simulator and D-Bug12. Note: We will always use the **SWI** instruction to end a program. This instruction will transfer control of the MC9S12 from the program to the D-Bug12 monitor.

```
; MC9S12 demo program
; EE 308
; 14 January 2009

; 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 for program
data:   equ     $1000       ; Starting address for data

        org     prog        ; Set initial program counter value

        ldaa    input1      ; Load first number into ACC A
        adda    input2      ; add second number
        adda    input3      ; add third number
        adda    input4      ; add fourth number
        lsra                ; divide by 2
        lsra                ; divide by 2 again
        staa    average     ; save result in memory
        swi

        org     data        ; Put data stating at this location
input1: dc.b    $15         ; First number
input2: dc.b    $63         ; Second number
input3: dc.b    $24         ; Third number
input4: dc.b    $3f         ; Fourth number

average: ds.b   1           ; Reserve one byte for result
```

Figure 1: An assembly language program used to get started with as12 and D-Bug12.

Question to answer before lab:

4. What are the contents of the A register after each instruction of the program shown in Figure 1 executes?

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** (available in the zip file http://www.freescale.com/files/microcontrollers/doc/data_sheet/9S12DP256B.zip). (Look 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**.

AsmIDE is an integrated development environment for working with assembly language files and the M9S12 microcontroller. The AsmIDE program is included on the CD which came with your Dragon12-Plus board, or you can download the latest version from http://www.ericengler.com/AsmIDE.aspx. The **getting started.pdf** file which comes on the CD with your Dragon12-Plus board shows how to install and use the AsmIDE software on you personal computer. Use of AsmIDE is also discussed in Section 3.6 of Huang.

Use the AsmIDE or a text editor to enter the program shown in Figure 1 before coming to lab, and save it under the name **lab01.asm**. To assemble the program double-click on the AsmIDE icon on your desktop. Open the file with the File:Open menu option. Assemble with the Build:Assemble menu option. Bring the **lab01.asm** file to lab on a flash drive (or put it onto a networked computer so you can download it in lab).

The as12 assembler will create a file called **lab01.lst** which shows what op codes were generated by the assembler, and a file called **lab01.s19**, which is the file you will use to load your program onto the MC9S12 evaluation board.

The Lab

On your account on the EE network, create a directory for this course (say, U:\EE308). Inside this directory create a subdirectory for this lab (say, U:\EE308\LAB01). Put the **lab01.asm** program into this directory, and assemble it with the commands shown above.

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 your **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 AsmIDE is running, and that the Terminal window is active. Power up the Dragon12-Plus board.

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.

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 Build:Download 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?