ASSEMBLY LANGUAGE PROGRAMMING AND 9S12 PORTS

In this sequence of three labs, you will learn how to write simple assembly language programs for the MC9S12 microcontroller, and how to use general purpose I/O (input/output) ports.

WEEK 1

Introduction and Objectives

This laboratory will give you more experience with the tools we will use this semester — the MiniDRAGON+ evaluation board (EVB), the D-Bug12 monitor, and the **as12** assembler. Be sure to read through the entire lab and do the pre-lab for each section **before** coming to lab

1. Consider the program in Figure 1:

```
prog:     equ  $1000      ; put program at address 0x1000
data:     equ  $2000
          org  prog
          ldab #29        ; IMM (immediate) address mode
          ldaa #235
          sba
          std  result     ; EXT (extended) address mode
          swi

          org  data
result:   ds.w 1          ; Set aside one byte of memory for
                          ; result
```

Figure 1. Demo program for Part 1 of Lab 2.

Pre-Lab

- Hand-assemble this program, i.e., determine the op-codes the MC9S12 will use to execute this program.
- How many cycles will this take on the MC9S12? (Do not consider the swi instruction.)
- How long in time will this take. (Note: the MC912 executes 24 million cycles per second.)
- What will be the state of the N, Z, V and C bits after each instruction has been executed (ignore the swi instruction.)

- What will be in address 0x2000 and 0x2001 after the program executed?

a) Assemble the program using **as12**. Look at the lst and s19 files. You should be able to relate the op-codes from the pre-lab to the data in the s19 file. (A document showing the format of the Motorola S19 files is available in the document S_RECORD.TXT on the EE308 datasheet page.)

b) Trace through the program. Verify that the Z, N, V and C bits are what you expect after each instruction.

c) Look at the contents of address 0x2000. Does the value agree with your answer from the pre-lab?

d) You could change this program to add rather than subtract by changing the sba instruction to an aba instruction. Modify the program, assemble it and load the new program into the MC9S12.
   a. Find the address for the sba instruction.
   b. In the MC9S12 Core Users Guide, find the op code for the aba instruction.
   c. Go to the address of the sba instruction, and change the op code to that of the aba instruction.
   d. Run the program again, and verify that the program now adds rather than subtracts.

2. Consider the program in Figure 2, which is a program to divide a table of ten values by 2.

; MC9S12 demo program Bill Rison 1/15/03
; This is a program to take a table of data, and create a new table which is the original
; table divided by 2

```
prog:       equ   $1000        ; put program at 0x1000
data:       equ   $2000        ; put data at address 0x2000
count:      equ   10           ; number of entries in table
            org   prog         ; set program counter to 0x1000
            ldab  #count       ; ACC B holds number of entries left
            ldx   #table1      ; Reg X points to entry to process
repeat:     ldaa  0,x          ; get table1 entry into ACC A
            asra               ; divide by 2
            staa  count, x     ; save in table 2
            inx                ; Reg X points to next entry in table1
            decb               ; decrement number left to process
            bne   repeat       ; if not done, process next table1 entry
            swi                ; done - exit

            org   data
                               ; initialize table 1 (COUNT bytes long)
table1:     dc.b  $19,$a7,$53,$e3,$7a,$42,$93,$c8,$27,$cf
```

```
table2:          ds.b  count       ; reserve count bytes for table2
```

Figure 2. Demo program for part 2 of lab 2

a) Use the text editor to enter this program, assemble the program into an s19 file.
b) How may cycles will it take to execute the program take on the MC9S12?
c) How long will it take to execute the program?
d) Use the fill option to change the values in addresses 0x2000 through 0x2FFF to 0xff. Reload the s19 file.
e) Set a breakpoint at repeat.
f) Execute the program again. The program should stop the first time it reaches the repeat label, with 0x0a in acc b, and 0x2000 in x.
g) Continue running the program. It should stop each time it gets to the repeat label –b should be decremented by one, x should be incremented by one, and there should be a new entry in table2.

3. Consider the code of Figure 3. Do parts (a) and (b) below before coming to lab

```
                ldy   #100
loop1:          ldx   #25000
loop2:          dex
                bne   loop2 ; conditional branch to loop2
                dey
                bne   loop1 ; conditional branch to loop1
                swi
```

Figure 3. Demo program for part 3 of lab 2.

Question to answer before lab:

- How many cycles will this program take on the MC9S12? (Again, ignore the swi instruction.)
- How long will it take to execute this program?
- Use a text editor to enter the code into a program – you will have to add org statements and other assemble directives to make the program work.

a) Assemble the program and run it on the EVB. How long does it take to run? This time should match your answer to your pre-lab question.

WEEK 2

Introduction and Objectives

The purpose of this laboratory is to write a few assembly language programs and test them on your MC9S12.

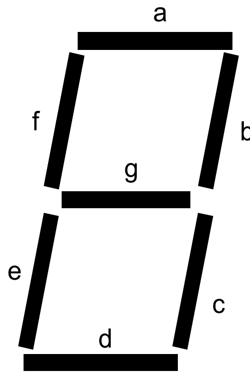
Pre-Lab

Make sure you have the programs written and clearly thought out before you come to the lab. You should put all your code starting at memory location 0x2000. You are encouraged to bring the programs in on a disk.

The Lab

As in last week's lab you will write some programs in assembly language and run the programs on the MC9S12. Write the program in Figure 1 and add necessary instructions to make it run.

The MniDRAGON+ has a seven-segment LED display connected to the MC9S12 Port H. A seven-segment display looks like this:



On the MiniDRAGON+, the ***a*** LED is connected to bit 0 of Port H, the ***b*** LED is connected to bit 1 of Port H, etc. (Bit 7 of Port H is connected to an infrared detector; it is not used to manipulate the seven segment LED display.) To display the number 3 on the LEDs, you need to turn on segments ***a***, ***b***, ***c***, ***d*** and ***g***. To do this, you need to write a 01001111_2, or 0x4f, to the Port H data register. (A 0xCF will also display a 3 on the LEDs, because the most significant bit can be either 0 or 1.) The following program will toggle the ***a*** LED of the seven segment display:

```
PTH         equ   $260
DDRH        equ   $262

            movb  #$ff,DDRH   ; make H an output port
            clr   PTH         ; clear port H data register

repeat:     bclr  PTH,$1
            bset  PTH,$1
            jmp   repeat
            swi
```

Figure 1. Demo program for part 2 of lab 2.

b) Test your program on the MC9S12. Trace through the loop to see what is happening.

Note: The program should cause one of the 7-segment display to toggle.

c) Modify the code to make all seven segments toggle in an alphabetical order.

d) Write a program to swap the last element of an array with the first element, the next-to-last element with the second element, etc. The array should have 0x20 eight-bit numbers and should start at 0x2000.

Check that your program works on the MC9S12. Use the following data for your test:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2000	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2010	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F

Write a program which puts the exclusive OR of the eight-bit numbers from memory locations 0x8000 through 0x8FFF into memory location 0x2003. (This operation is often used to generate a check sum to verify data transmission. The sending computer generates and transmits the check sum along with the data. The receiving computer calculates the check sum for the received data and compares it with the check sum sent by the sending computer. If the two values do not match, then there was an error in the transmission (The last byte in a line of an S19 file is a check sum used for this purpose.)

WEEK 3

Introduction and Objectives

In this week's lab you will write an assembly-language program to display various patterns on the 7-segment display. You will use the MC9S12's Port H as an output port to display the patterns, and 2 pushbutton switches via analog port ATD0 to decide which pattern to display.

For this week's lab, you will create programs to write to Port H, and read from the analog port 0. You will use information from switches connected to analog port to control the pattern you output on Port H. You will test your programs by changing the Port H output by modifying the switch settings connected to analog port 0.

Pre-Lab

We want to write a program which will display four different patterns on the seven segment LED display connected to Port H. We want to use two input lines to select which of the four patterns to display. To do this, we will use bits 1 and 0 of Port B.

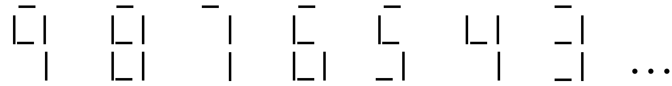
Write a program to set up Port H as an 8-bit output port, and to implement (i) a binary up counter, (ii) an decimal down counter, (iii) a flasher, and (iv) a message scrolling banner. For the binary up counter, just have Port H count 0, 1, 2, 3, 4, …. This will generate a random looking pattern on the LEDs. It should take 128 counts from the time all LEDs are off until the next time all are off. Samples from the other sequences that you should generate are shown in Fig. 1. Include an appropriate delay (about 250 ms) between changing the pattern so that you can easily and comfortably see them flash. Use a subroutine to implement the delay.

Set up Port B as an input port, and use bits 1 and 0 to control which of the Port H functions are performed as shown in Fig. 2. You will need to connect DIP switches (from your EE 231 lab kit) to the miniDRAGON+ protoboard, and connect the DIP switches to bits 1 and 0 of Port B. (Look at the pinout diagram for the 112 pin chip on page 54 of the MC9S12 Core Users Guideto determine which pins to use.)

When you switch between functions, the new function should start up where it ended when it was last activated, so set aside variables to save the states of the various patterns.

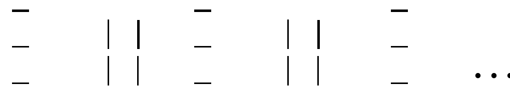
Start writing the program before coming to lab. You should at least have an outline (or flowchart) of the flow control of the program, and pseudocode for how you plan to implement your functions. Be sure to write the program using structured, easy-to-read code.

b) A decimal down counter:



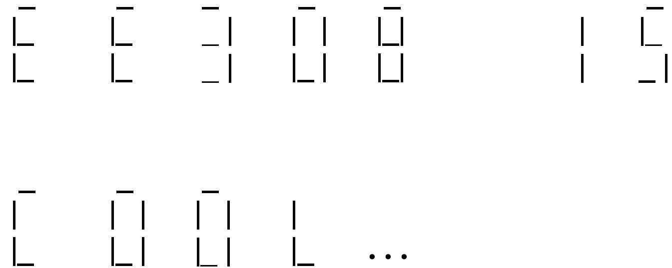
Continue counting down

c) A flasher:



Repeat the above sequence

d) Generate a message scrolling banner.



Repeat the above message

Figure 1. Samples of the functions to be performed using Port H as an output.

PB1	PB0		Port H Function
0	0		Binary Counter
0	1		Decimal Counter
1	0		Flasher
1	1		Message

Figure 2. Port B inputs to control the Port A functions.

The Lab

Run your program on the MC9S12. If you have difficulty getting your program to work, start by trying to implement one function only – say, the binary counter. Once this works, start working on your next functions.

Set a break point at the first line of your delay subroutine. When the breakpoint is reached, check the value of the stack pointer, and the data on the stack. Make sure you understand what these mean.

When you get your program to work, have your lab instructor or TA verify the program operation.

The MC9S12 has EEPROM (Electrically Erasable Programmable Read Only Memory) functionality. If you put your program into EEPROM the program will remain there when you turn off power.

The EEPROM is located at address 0x400. You can just change the origin statement of your assembly language program, then reassemble, and reload your program. If you type “G 400”, you will run your program out of EEPROM, and it should work the same as it did when you ran it out of RAM. (Try it.) You can power cycle your board, and then type “G 400”, and again your program will run correctly. (Try it.) For some applications it would be nice if you could run your program without having to type “G 400” – if your board is controlling a robot, and no computer is connected to it, it would be impossible to start the program by typing “G 400”. D-Bug12 has a special mode to allow you to run a program out of EEPROM without having to type “G 400”. If you move jumper J9 to the left, and power cycle the board, the program will run immediately out of EEPROM. (Try it.) You will notice that the program runs much slower – actually, three times slower than it did when you ran it by typing “G 400”. This is because D-Bug12 does some system initialization which is bypassed when you move jumper J7 and run your program directly from EEPROM. In particular, the miniDRAGON+ board has a 16 MHz clock, and the MC9S12 runs at half the clock frequency, or 8 MHz. The MC9S12 has a built-in phases lock loop (PLL) which allows the chip to generate a faster clock internally, and run with a 24 MHz frequency. In order to get the chip to run at the higher frequency, you must do the initialization which enables the PLL. Here is some code which will do that initialization:

```
     org    $400

; Program the PLL to multiply the oscillator clock by 3
; Top get a 24 MHz clock from a 16 MHz crystal,
; multiply by 6 and divide by 4 (25 = 16 * 6 / 4)

     bclr   $36,#$80           ; Use oscillator rather than PLL
     ldab   #$03               ; Set clock divider to 4
     stab   $0035
     ldab   #$05               ; Set clock multiplier to 6
     stab   $0034
l1:  brclr  $0037,#$08,l1      ; Wait for PLL to lock
     bset    $39,#$80          ; Switch to PLL
```

Add the above code to your program, right after the "org $400" line and before the first line of your program. Load this new code into EEPROM. (Be sure to move jumper J7 to the right in order to get back to the D-Bug12 monitor so you can load new code into memory.) Now move jumper J7 to the left, power cycle your board, and your program should run at the same speed it did when starting by "G 400".

Note: The document "readme_EEPROM.pdf" which came on the miniDRAGON+ CD says that you need to convert your S1 code (in the S19 file) to S2 code to successfully load a program into EEPROM. This is because the MC9S12 EEPROM must be programmed with an even number of bytes, and must be programmed starting at an even address. However, I have had no problem loading a program which starts on an odd address or has an odd number of bytes. I think that, when D-Bug12 sees that a user wants to load a program which starts on an odd address or contains an odd number of bytes, it automatically adds the bytes needed to make the programming start on an even address or to contain an even number of bytes. If you have trouble getting an EEPROM program to load correctly, you should try converting your S1 code to S2 code as discussed in the "readme_EEPROM.pdf" document.