Microprocessor Systems
Lab 1: Introduction to the EVB

This lab is intended for the first-time user of the M68HC12EVB evaluation board (referred to as EVB). The setup and operation of the EVB is explained. An introduction to the D-Bug 12 monitor program and its memory and register management commands is provided.

1.1 Equipment

- M68HC12EVB containing D-Bug 12 monitor program (**Please refrain from touching the EVB. Alert your TA if you need any assistance with the board**).
- Personal Computer containing MiniIDE software
- Power Supply

1.2 Introduction

The EVB contains a Motorola 68HC12 microcomputer and associated I/O and memory devices. The EVB allows the user to easily test systems and software designed for the 68HC12 microcomputer. On the board is an EEPROM, containing the D-Bug12 monitor program. The monitor program begins executing on power-up or reset. This program controls the communication between the EVB and a terminal (in this case a personal computer – PC). It also allows the user to examine or modify the contents of memory, and assemble, debug and execute programs.

1.2.1 EVB memory description

The monitor program, D-Bug 12, occupies the 32-kByte FLASH EEPROM area of the MCU's memory map (See table below). D-Bug 12 requires 512 bytes of on-chip RAM ($0A00 to $0BFF), for stack and variable storage. The remaining 512 bytes of on-chip RAM ($0800 to $09FF), are available for variable storage and stack space by user programs.
Note that all memory locations are expressed as 16-bit hexadecimal numbers, and memory values are expressed as 8-bit hexadecimal numbers.

Address Range	Usage	Description
$0000 - $01FF	CPU registers	On-chip registers
$0800 - $09FF	User code/data	512 bytes on-chip RAM
$0A00 - $0BFF	Reserved for D-Bug 12	512 bytes on-chip RAM
$0D00 - $0FFF	User code/data	768 bytes on-chip EEPROM
$8000 - $FFFF	D-Bug 12 code and functions	32 Kbytes on-chip FLASH EEPROM

The Axiom CML 12S-DP256 is an HC12 evaluation board developed by Axiom Manufacturing (www.axman.com), providing 256 kB of memory. Memory maps differ among evaluation boards with different amounts and distributions of memory. **When programming for the CML 12S-DP2256, you should be applying the following memory map.** I recommend that you use locations $4000-$7FFF or $8000-$BFFF for your program and data.

Axiom CML 12S-DP2256 Memory Map

ADDRESS	TYPE MEMORY	MEMORY APPLICATION
$C000 -$FFFF	FLASH	MON12, NOICE, and Utility firmware located in internal flash, Page $3F.
$8000 -$BFFF	External Ram	**User Paged Program Memory space**, Pages $20 - $2E. Note: Pages $30 - $3F reside in the internal flash.
$4000 -$7FFF	External Ram	**User Program Memory.** Emulate fixed page $3E.
$3F8C -$3FFD	Internal Ram	Ram Interrupt Vector Table
$3E00 -$3F8B	Internal Ram	Monitor reserved ram memory. Stacks and variables
$1000 -$3DFF	Internal Ram	**User Internal Ram memory**
$0400 -$0FEB	Internal EEprom	User EEprom memory, Monitor reserves $FEC - $FEF for Autostart, user should avoid $FF0 - $FFF memory use.
$0000 -$03FF	HCS12 Registers	**Monitor or user access to control registers.**

1. Memory manipulation commands

Review section 1.2.1 of this manual for EVB memory description. Recall that all memory locations are expressed as 16-bit hexadecimal numbers, and memory values are expressed as 8-bit hexadecimal numbers.

The Memory Display command (mm):

The monitor program allows the user to examine and modify the contents of memory. The "**md**" command is the "memory display" command and is used to display the contents of memory. The following example illustrates the "**md**" command. The bold-faced text indicates text that is typed by the user, and the second line indicates the response that you will see on your monitor. Type the following at the prompt:

```
>md 0800
 0800  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

This command displays a single line of memory, starting at location 0800 and ending at location 080F. In our example all memory locations from 0800 to 080F contain the value "FF".

The Memory Modify command (mm):
The “memory modify” command, "**mm**", can be used to alter the values stored in memory locations. The following example shows how the memory location 0800 would be altered. The bold-faced text indicates text that is typed by the user, and the second line indicates the response that you will see on your monitor.

```
>mm 0800
 0800  FF  01
```

In this example the value displayed, "FF", is the current value in memory location 0800. After “FF” the cursor appears and waits for you to enter a value then press the return key (ENTER). In our example we entered the value “**01**”. More than one consecutive memory locations can be modified if a “Return” is typed. For example,

```
>mm 0800
 0800  01  01  FF  02  FF  03  00  04  07  05  17
```

You can now use the “**mm**” command to view the changes you made to the memory locations.
Note: To exit MEMORY MODIFY mode type the period, “**.**”, as the first non-space character on the line. This will return you to the D-Bug12 prompt.

The Block Fill command (bf):
Filling large sections of memory with a constant value can be time consuming using the "**mm**" command. For example, it might be desirable to zero a large section of the on-chip RAM memory. The block fill command allows this type of operation. The following command zeros all of the user on-chip RAM memory (locations 6000 – 6030.

```
>bf   6000  6030    00
```

Try this command and then use "**md**" to display this section of memory. Now, type the command **"md 6000** " and display the changes.

The Move command (move):
The "move" command copies entire sections of memory from one location to another. To use the move command, the starting address of the source, the ending address of the source, and the starting address of the destination must be supplied.
In order to copy locations 0800 - 08ff to locations 0D00-0Dff, the following command can be used.

```
> mov  0800  08ff  0D00
```

1.2 Registers and Register Modification

Registers are small storage locations to which special operations can be applied. The 68HC12 microprocessor has seven registers, the program counter (PC), the stack pointer (SP), two index registers (X and Y), two eight bit accumulators (A and B) and a condition register (CCR).

The Register Display command (rd):

This command is used to display the CPU12's registers. For example

>**RD**

PC SP X Y A B CCR

This command displays all the registers and their current values.

The Register Modify command (rm):

The contents of the registers can be examined and altered with the "**rm**" command. Typing "**rm**" displays the contents of the registers and allows them to be altered starting with the PC register. Type the following at the prompt

>**RM**

Now modify all the registers then verify that your changes took effect.

Hint: To exit REGISTER MODIFY mode type the period, ".", as the first non-space character on the line. This will return you to the D-Bug12 prompt.

Note: All exercises must be conducted during the lab, and the solutions submitted with your lab report.

Create a table that summarizes all the commands you used (D-Bug12 and Kermit) during the lab. The following should serve as an example:

2. Entering and Running a Simple Program (Using the MiniIDE Software to Develop a Program)

The following assembly language code is a simple program to get you started. The code loads the number 5 into accumulator A of the 68HC12, copies that value from accumulator A to accumulator B, and then stores the value in accumulator B in memory location $4000.

```
ORG     $8000       ; starting address of the program in memory
LDAA    #$05        ;loads accumulator A with value 5
TAB                 ;transfers contents of A to B
STAB    $4000       ;stores contents of B in memory location 4000 (hex)
RTS                 ;returns control to monitor program (RTS or END)
```

For all of your exercises and problems, you will be using MiniIDE to create the program on the PC and then download the program to the HC12 EVB where you can run it. The use of MiniIDE and downloading of a program is discussed in sections 3.7.4, 3.7.5, and 3.7.6 of your textbook. The steps to using MiniIDE to execute your code are as follows: First we generate, assemble, and download the program.

1. Generate the program using the MiniIDE editor (section 3.7.4).
2. Assemble the text program to generate the "machine" code used by the HC12 microprocessor (section 3.7.5).

3. Download the program (machine code) to the EVB (see "load" instruction on page 96 - your offset will be zero and the optional offset can be ignored, as in Figure 3.7).

Then, once the program has been compiled and loaded into memory, it can be run with the "**CALL**" command. This effectively calls a **subroutine** and then returns to the D-Bug12 monitor. The "**RTS**" command in the program above is a "return to subroutine" command.

2.1 Exercise

Enter the above program into MiniIDE's texteditor. After you have completed typing up the code, compile the code then load it into memory using the relevant buttons on the menu bar of MiniIDE. You can ask your TA for help in locating these menu buttons.

2.2 Exercise

Enter CALL 8000 at the D-Bug 12 prompt. (*Question: Why **not** CALL 4000?).*
Your program now executes up to the RTS command. If you left out the RTS command, the program will execute up to the first illegal command and then exit. You will not be sure of where the execution ended if the last line happens to be a code representing an actual instruction.
The contents of the registers will be displayed when the program finishes executing. Both accumulators should contain 05. Examine memory location 4000; it also should contain 05.

3. Laboratory Project Description

Using MiniIDE, enter and run the programs discussed in the following examples/assignments in your textbook. Adjusting of memory locations is illustrated below for the first example only. It will be easiest if you simply readjust the memory locations as needed for your program and data before typing in the program.

3.1.1 Example 2.4 (Page 37): This program sets the starting address of the instructions at $1000, and stores the final result in memory location $900. Change its starting address to $8000 and correct all data address with $4XXX, For example, an address $802 is replaced by $4002.
Then the program becomes:

```
org     $8000
ldaa    $4000
adda    $4002
suba    $4004
staa    $40F0
rts
```

Complete the following steps for this example.

a. Using the D-Bug12, read and **record** the data stored in memory locations $4000, $4002, $4004, and $40F0.

b. Enter the program for this example in the MiniIDE TextEditor.

c. Run the program.

d. Once more, read and record the data stored in memory locations $4000, $4002, $4004, and $40F0.

e. What changed and what did not change? Explain why changes did or did not occur.

f. Read and record the contents of the registers. What data is stored in accumulator A. Is this the same as the data stored in memory location $40F0? Explain the result.

g. Using the D-Bug12 monitor program, change the data in memory locations $4000, $4002, and $4004 to new values (your choice) and rerun the program. Is it necessary for us to also recompile the program? Explain your answer. Repeat steps **d** and **e** and explain the new results.