68HC12 Training Lab
Student Exercise Book

Date: 13 September, 2000

Document Revision:1.01

BiPOM Electronics

11246 South Post Oak #205, Houston, Texas 77035
Telephone: (713) 661- 4214 Fax: (713) 661- 4201
E-mail: info@bipom.com
Web: www.bipom.com

WARRANTY:

BiPOM Electronics warrants 68HC12 Training Kit for a period of 1 year . If the Kit becomes defective during this period, BiPOM Electronics will at its option, replace or repair the Kit. This warranty is voided if the product is subjected to physical abuse or operated outside stated electrical limits. BiPOM Electronics will not be responsible for damage to any external devices connected to the Kit. BiPOM Electronics disclaims all warranties express or implied warranties of merchantability and fitness for a particular purpose. In no event shall BiPOM Electronics be liable for any indirect, special, incidental or consequential damages in connection with or arising from the use of this product. BiPOM's liability is limited to the purchase price of this product.

TABLE OF CONTENTS

Introduction

The purpose of the 68HC12 Training Lab is to familiarize the student with developing practical applications for the Motorola 68HC12 series of micro-controllers. 68HC12 is one of the most popular micro-controllers on the market with applications ranging from industrial, medical, home automation to automotive.

The 68HC12 Training Kit consists of the following components:

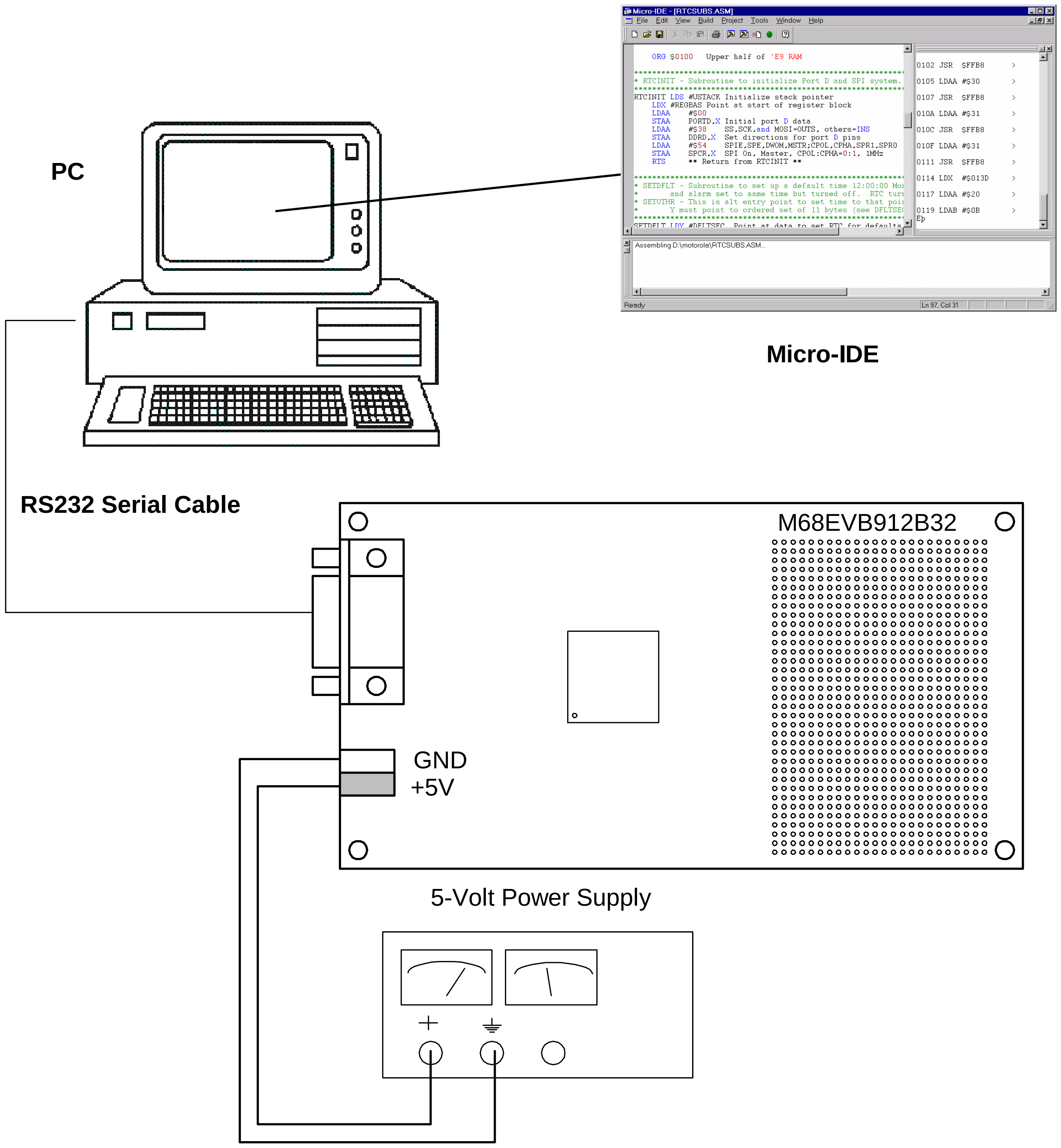
- Motorola M68EVB912B32 Micro-controller Board (Training Board)
- Micro-IDE Integrated Development Environment
- 68HC12 Assembler
- (Optional) Micro C Compiler and linker for the 68HC12
- Serial Cable

The following external items are required for each training kit station:

- IBM Compatible Personal Computer (PC) running Windows 95/98/NT 4.0 or 2000.
  (68HC12 Training Kit will not work with DOS, Windows 3.1 or lower).
  Minimum 16MB memory and 10 MB of available hard disk space.
  One available RS232 Serial port.

- 5-Volt regulated power supply.

Figure 1 shows how all the components are connected together.


PC
Micro-IDE - [RTCSUBS.ASM]
File Edit View Build Project Tools Window Help
Micro-IDE
RS232 Serial Cable
M68EVB912B32
GND
+5V
5-Volt Power Supply

```
    ORG $0100   Upper half of 'E9 RAM

****************************************************************
* RTCINIT - Subroutine to initialize Port D and SPI system.
****************************************************************
RTCINIT LDS #USTACK Initialize stack pointer
    LDX #REGBAS Point at start of register block
    LDAA    #$00
    STAA    PORTD,X Initial port D data
    LDAA    #$38    SS,SCK,and MOSI=OUTS, others=INS
    STAA    DDRD,X  Set directions for port D pins
    LDAA    #$54    SPIE,SPE,DWOM,MSTR;CPOL,CPHA,SPR1,SPR0
    STAA    SPCR,X  SPI On, Master, CPOL:CPHA=0:1, 1MHz
    RTS     ** Return from RTCINIT **

****************************************************************
* SETDFLT - Subroutine to set up a default time 12:00:00 Mon
*       and alarm set to same time but turned off.  RTC turn
* SETOTHR - This is alt entry point to set time to that poin
*       Y must point to ordered set of 11 bytes (see DFLTSEC
****************************************************************
SETDFLT LDY #DFLTSEC  Point at data to set RTC for defaults
```

```
0102 JSR  $FFB8     >
0105 LDAA #$30      >
0107 JSR  $FFB8     >
010A LDAA #$31      >
010C JSR  $FFB8     >
010F LDAA #$31      >
0111 JSR  $FFB8     >
0114 LDX  #$013D    >
0117 LDAA #$20      >
0119 LDAB #$0B      >
Éþ
```

```
Assembling D:\motorola\RTCSUBS.ASM...
```

Ready   Ln 97, Col 31

Figure 1

Each student or group needs to log on to Windows before using the 68HC12 Training Kit. Enter the user name and password that your instructor gave you before the Lab. Make sure to log out at the end of the Lab. Do not share your username or password with anybody.

All programs are written as Assembly Language Program files which have the extension **.ASM** but they are plain text files. Assembly Language Program files are created using Micro-IDE Integrated Development Environment for Windows.

You can edit and save programs and download to the Training Board using Micro-IDE. Creating programs and running them on the Training Board consists of the following steps:

1. Edit an existing or create a new program using Micro-IDE Program Editor.

2. Assemble the program using Micro-IDE Build Toolkit

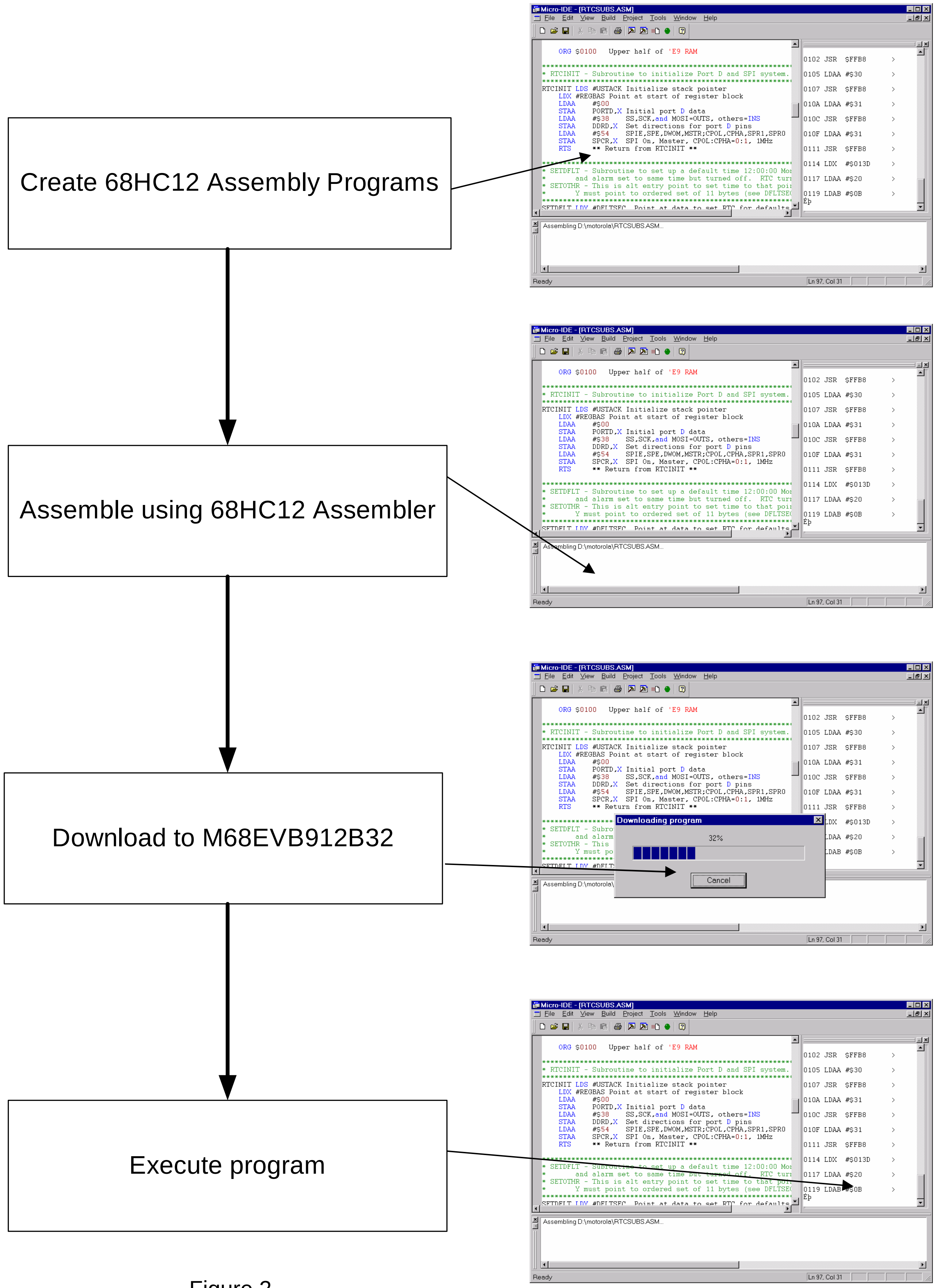
3. Download the program to the Training Board using Micro-IDE Serial Loader.

4. Run and debug the program on the Training Board using Micro-IDE Terminal Window.

Figure 2 illustrates these steps.

The 68HC12 micro-controller on the Training Board is factory programmed with a monitor/debugger environment called D-Bug12. You can send commands and run programs on the 68HC12 using D-Bug12 features. D-Bug12 communicates with the PC through Micro-IDE's Terminal Window.

Refer the M68EVB912B32 Evaluation Board User's Manual to get familiar with the Training Board and the D-Bug12 monitor commands.


Create 68HC12 Assembly Programs
Assemble using 68HC12 Assembler
Download to M68EVB912B32
Execute program
Micro-IDE - [RTCSUBS.ASM]
File Edit View Build Project Tools Window Help
Downloading program
32%
Cancel
Ready
Ln 97, Col 31

```
    ORG $0100   Upper half of 'E9 RAM

****************************************************************
* RTCINIT - Subroutine to initialize Port D and SPI system.
****************************************************************
RTCINIT LDS #USTACK Initialize stack pointer
    LDX #REGBAS Point at start of register block
    LDAA     #$00
    STAA     PORTD,X Initial port D data
    LDAA     #$38     SS,SCK,and MOSI=OUTS, others=INS
    STAA     DDRD,X  Set directions for port D pins
    LDAA     #$54     SPIE,SPE,DWOM,MSTR;CPOL,CPHA,SPR1,SPR0
    STAA     SPCR,X  SPI On, Master, CPOL:CPHA=0:1, 1MHz
    RTS      ** Return from RTCINIT **

****************************************************************
* SETDFLT - Subroutine to set up a default time 12:00:00 Mon
*       and alarm set to same time but turned off.  RTC turn
* SETOTHR - This is alt entry point to set time to that poin
*       Y must point to ordered set of 11 bytes (see DFLTSEC
****************************************************************
SETDFLT LDY #DFLTSEC  Point at data to set RTC for defaults
```

```
0102 JSR  $FFB8    >
0105 LDAA #$30     >
0107 JSR  $FFB8    >
010A LDAA #$31     >
010C JSR  $FFB8    >
010F LDAA #$31     >
0111 JSR  $FFB8    >
0114 LDX  #$013D   >
0117 LDAA #$20     >
0119 LDAB #$0B     >
Éþ
```

```
Assembling D:\motorola\RTCSUBS.ASM...
```

Figure 2

LAB1

Overview

The purpose of this lab is to familiarize the student with the Motorola M68EVB912B32 micro-controller board (Training Board) and the program development environment. In this lab, students will create a simple 68HC12 program in Assembly language, assemble the program, download the program to the board and execute the program.

The knowledge developed in this lab will be very useful in subsequent labs when working with the 68HC12 to develop programs. It is recommended that the student read the M68EVB912B32 Evaluation Board User's Manual and become familiar with the M68HC912 micro-controller and the D-Bug12 environment before starting this lab.

Instructions

Part I

Create and execute a simple 68HC12 program using the D-Bug12 environment by taking the following steps:

1. If you have not already logged into Windows, login now by specifying your username and password.

2. Start Micro-IDE by selecting Start, Programs and Micro-IDE. Select the Micro-IDE option under Micro-IDE folder.

3. Check communications to the M68EVB912B32 board. Start the Terminal program by selecting Terminal under View menu. A blank terminal screen will appear on the right side of the Micro-IDE window. (You can resize the terminal screen by selecting the left edge of the terminal window with the left mouse button and dragging to the right.) Select Tools, Terminal and Connect. This will establish communications to the M68EVB912B32.

4. Press the RESET button on the Training Board. The following message should appear on the terminal screen.

```
D-Bug12 v2.0.2
Copyright 1996 - 1997 Motorola Semiconductor
For Commands type "Help"

>
```

Depending on the version of your M68EVB912B32 board, this message may be slightly different. This message indicates that you have established communications with the board.

5. Click the left mouse button once in the Terminal window to set the input focus to that window. Press <ENTER> once. Another '>' prompt will be displayed. This means you are now ready to enter commands.

At any point, you can obtain quick help on the available commands and their syntax by typing HELP and then pressing Enter.

6. Press the Reset button (S1) on the M68EVB912B32 board and enter RD command to view the current values of micro-controller registers. Make a note of these values in Table 1 below:

Register	Hexadecimal Value
PC	
SP	
X	
Y	
A	
B	
D (A:B)	
CCR	

Table 1

7. The memory map of the 64K address space of the M68HC912B32 micro-controller is shown in Table 2:

Type	Address Range
Micro-controller registers	$0000 - $01FF
RAM	$0800 - $0BFF
EEPROM	$0D00 - $0FFF
Flash EEPROM	$8000 - $FFFF

Table 2

We will use RAM at location $0800 for writing and executing simple programs. RAM locations from $0800 to $09FF is available for user programs and data. RAM locations from $0A00 and $0BFF are used by D-Bug12 and should not be altered by the user.

Use the ASM command to start entering your program at address $0800. Type

```
ASM 0800
```

D-Bug12 will display the current instruction at address $0800 followed by the > prompt. Replace this current instruction by typing

```
LDAA #$01
```

and pressing Enter.

D-Bug12 will now display

```
0800   8601   LDAA #$01
```

followed by the instruction at address $0802. Note that next to 0800, 8601 is displayed; this is the machine code equivalent of LDAA #$01.

Now, type RTS and press Enter. D-Bug12 will display

```
0802   3D      RTS
```

and the instruction at address $0803. Note that the machine code of 3D which is the equivalent of RTS instruction is also displayed.

Exit the assembly by typing '.' and pressing Enter. D-Bug12 will take you back to the > prompt. You have now entered the following simple program

```
LDAA #$01
RTS
```

This program loads the accumulator A with a value of 1 and returns control to D-Bug12 using the RTS instruction.

8. Execute this program by typing

```
call 0800
```

and pressing Enter. This will pass control to your simple program at address 0800, execute it and when the program completes, D-Bug12 will display the current values of the micro-controller registers. Make a note of these values in Table 3 below:

Register	Hexadecimal Value
PC	
SP	
X	
Y	
A	
B	
D (A:B)	
CCR	

Table 3

How is Table 1 different from Table 3 ? What is the value of accumulator A after you have executed your simple program ?

Part II

1. For the second part of the lab, first erase the RAM to make sure that the simple program that we entered is removed. To do this, type

   ```
   bf 0800 09ff ff
   ```

   and press Enter. bf is the block fill command which will fill the memory locations from $0800 to $09ff with values of $ff . Verify that the program is removed by typing

   ```
   asm 0800
   ```

   and pressing Enter. This should display

   ```
   0800      FFFFFF      LDS   $FFFF       >
   ```

   which means that the RAM area was successfully filled with values of $ff. Type '.' And press Enter to exit the assembly mode. You should be back at the > prompt now.

2. Create a simple project using Micro-IDE. This project will contain a simple 68HC912 Assembly program. From the Micro-IDE menu, select Project and New Project. Enter **test** as the project name and type **c:\test** for the Location. Select

   ASM12 Assembler for 68HC12

3. Now create a simple Assembly file and add to the test project. To do this, select File and New. Select ASM File and type the name **test.asm**. Location should already be **c:\test**; if not, enter the correct Location as **c:\test**.

   Click OK. This will create the empty assembly file test.asm.

   Type the following simple 68HC912 Assembly program:

   ```
           ORG $0800

           LDAA #1
           RTS
   ```

   This program loads the accumulator A with the value of 1 and returns. The origin directive (ORG) tells the assembler that the start address of program in memory is hexadecimal 0800. Address 0800 is mapped to RAM on the M68EVB912B32 board.

Important Note: Enter a tab or spaces to the left of each assembly instruction; otherwise the assembler will give Unknown Instruction errors.

4. Assemble the file by selecting Build and Build test.asm. This will assemble the test.asm source file and create the test.hex file to be downloaded to the board. Test.hex is in Motorola S Record format. Details of the hex format can be found in the M68EVB912B32 Evaluation Board User's Manual.

   If test.asm is assembled without errors, the following message appears in the Output window of Micro-IDE:

   Assembling c:\test\test.asm…
   First pass… Second pass… 0 error(s).

5. Download the test.hex file to M68EVB912B32 board by selecting Build and Download test.hex.

   You should see a progress indicator during download. When the download is finished, you should see the message

   >

   on the Terminal screen. This means that the board has received the program successfully. Since our test program starts at address 0800 (because of the ORG $0800 directive), it will automatically be loaded to 0800.

   At the > prompt, execute the **test.asm** program by typing

   call 0800

   What is the value of accumulator A ? What value should it be ?

LAB2

Overview

The purpose of this lab is to familiarize the student with 68HC12 arithmetic and logic instructions. The student will also learn the D-Bug12 Monitor's Trace commands to trace and debug the operation of a program.

Information

D-Bug12 Monitor on the Training Board allows tracing through the programs one instruction at a time using the Trace command (T). Trace command also displays the 68HC12 register contents after each instruction is executed. Before using Trace command, the Program counter should be set to the starting address of the program to be traced. For example, to start tracing a program at address $0800, PC command should be used to set the program counter to $0800.

Instructions

Manually fill in the trace tables on the attached sheet titled 68HC12 Trace Tables before starting the lab. For each instruction, indicate which registers are effected by filling in the value of that register after executing the instruction.

Type the program shown below and save as trace.asm using Micro-IDE. Build and download the program to the Training Board.

```
* This program demonstrates various 68HC12 instructions

        ORG $0900

VAR1      FCB  $A1
VAR2      FCB  $FF
VAR3      FCB  $69
VAR4      FCB  $73
VAR5      FCB  $1D
VAR6      FCB  $3A
VAR7      FCB  $CC
VAR8      FCB  $44
VAR9      FCB  $84
VAR10     FCB  $00
VAR11     FCB  $11


MAIN    ORG  $0800
        LDX  VAR4
        LDD  VAR10
        LDY  VAR11
        STAA $07,Y
        ADDA VAR5
        ADCB #100
        XGDX
        INY
        DECB
        ABY
        ADDB VAR9
        SUBD VAR7
        NEGA
        STX  VAR8
        COM  VAR10
        MUL
        STD  VAR6
        LDD  VAR10
        LDX  VAR9
        IDIV
        STD  VAR4
        STX  VAR2
LOOP    BRA  LOOP
```

Before using Trace command, set the Program counter to the starting address ($0800) of trace.asm. Use PC 0800 command in the Terminal window to set the program counter.

Use the T command to trace the program. D-Bug12 will display the instruction that was executed and the contents of all the registers. If you enter T followed by a number, D-Bug12 will execute that number of trace commands. For example, if you enter T 3, D-Bug12 executes 3 traces. For this lab, trace the program only one instruction at a time. Use MD command to watch memory contents after executing an instruction.

As you trace through the program, compare the register and memory values to those in the Trace Table that you manually filled in at the beginning of the lab.

LAB3

Overview

The purpose of this lab is to interface the 68HC12 to the outside world by developing programs to read digital inputs and write to digital outputs.

Information

68HC912B32 has 8 input/output ports with a total of 57 general-purpose input/output lines. Ports are grouped in units of 8 input/output lines (except for PORT DLC which has 7 input/output lines) and are assigned a port letter. The following are the ports on the 68HC912B32:

PORT	Function	Alternate Function
PA0-PA7	General purpose input/output	ADDR8-ADDR15/Data8-Data15
PB0-PB7	General purpose input/output	ADDR0-ADDR7/Data0-Data7
PE0-PE7	General purpose input/output (PE0 and PE1 are inputs)	LITE INTEGRATION MODULE
PAD0-PAD7	General purpose inputs	Analog/Digital Converter
PDLC0-PDLC6	General purpose input/output	Serial port
PP0-PP7	General purpose input/output	Pulse-Width-Modulation
PS0-PS7	General purpose input/output	Serial port
PT0-PT7	General purpose input/output	Timer and pulse accumulator

On the Training Board, these ports can be accessed through connector blocks P2, P3, P4 and P5.

Each port has a memory-mapped address in the 68HC12's address space. The following are the addresses for the ports:

Register name	Description	Address
PORTA	Port A Data Register	$0000
DDRA	Port A Data Direction Register	$0002
PORTB	Port B Data Register	$0001
DDRB	Port B Data Direction Register	$0003
PORTE	Port E Data Register	$0008
DDRE	Port E Data Direction Register	$0009
PORTAD	Port AD Data Register	$006F
PORTDLC	Port DLC Data Register	$00FE
DDRDLC	Port DLC Data Direction Register	$00FF
PORTP	Port P Data Register	$0056
DDRP	Port P Data Direction Register	$0057
PORTS	Port S Data Register	$00D6
DDRS	Port S Data Direction Register	$00D7
PORTT	Port T Data Register	$00AE
DDRT	Port T Data Direction Register	$00AF

Ports can be made input or output using the Port Direction Registers (with the exception of PAD0-PAD7, PE0 and PE1; these lines are always inputs. After setting the port as either an input or output, the Port value can be read from or written to using the Port Data Register.

Instructions

Connect switches and Light emitting Diodes (LED's) to the Training Board using the following wiring diagram:

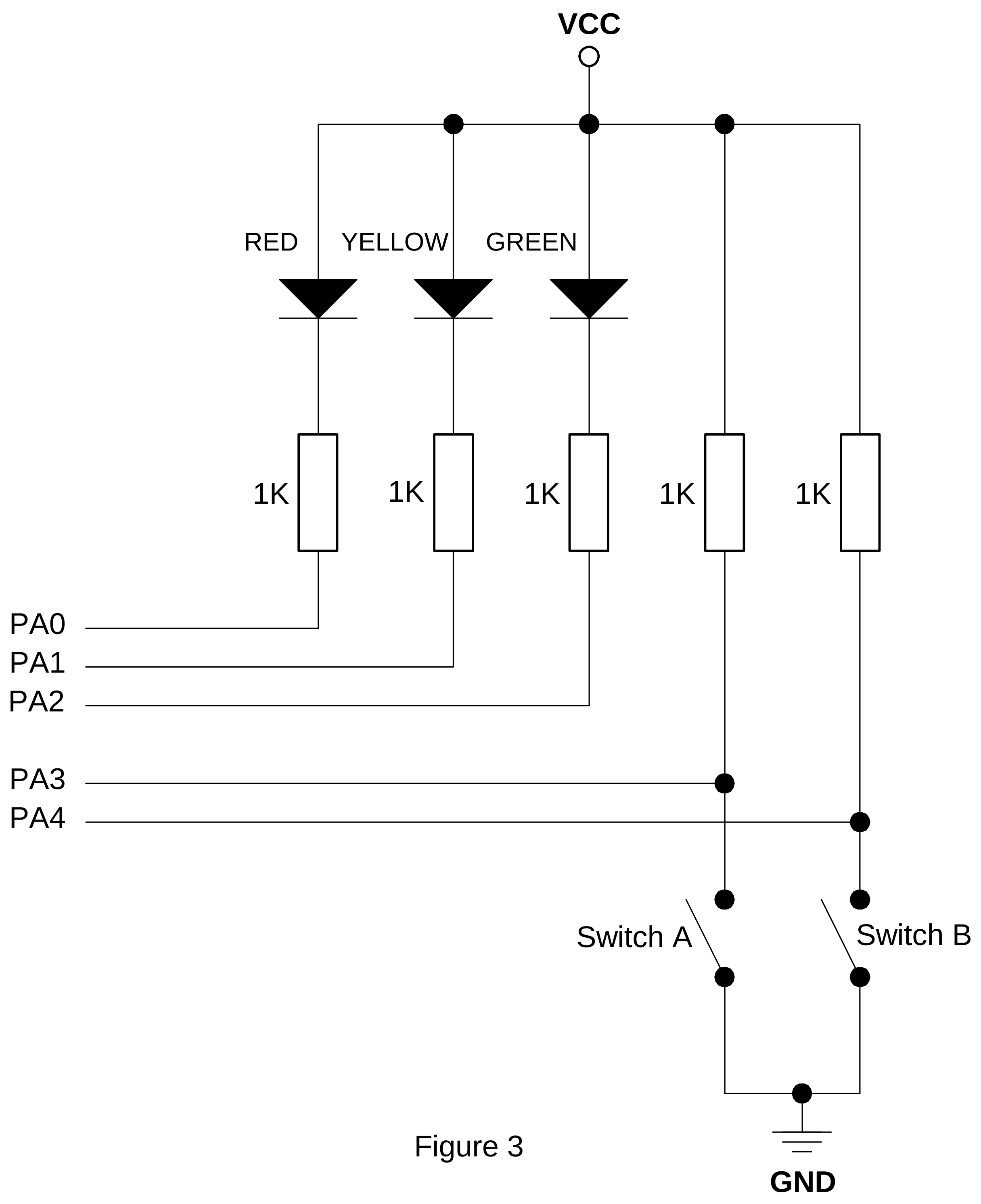

VCC
RED
YELLOW
GREEN
1K
1K
1K
1K
1K
PA0
PA1
PA2
PA3
PA4
Switch A
Switch B
GND

Figure 3

Write a 68HC12 program to generate the following results:

Switch A	Switch B	Red LED	Yellow LED	Green LED
Open	Open	OFF	OFF	ON
Open	Closed	ON	ON	OFF
Closed	Open	OFF	OFF	ON
Closed	Closed	OFF	OFF	OFF

Build the program and download to the Training Board. Trace through the program and watch the LED’s as you step through each instruction. Open and close Switch A and Switch B to change the LED’s to make sure that your program is handling every case correctly.

LAB4

Overview

The purpose of this lab is to program 68HC12's Electrically Erasable Read Only Memory (EEPROM).

Information

68HC12 has 768 bytes of EEPROM starting at address $0D00 and ending at address $0FFF. EEPROM can be used for storing both programs and data. The information that is programmed in the EEPROM stays even if the power to the Training Board is turned off.

Refer to the Section 'EEPROM' in Motorola M68HC12B Family Advance Information manual for more information on the EEPROM.

Instructions

Part I

Check the contents of the EEPROM using MD command. If the EEPROM is blank, all bytes should be $FF (all bits high).

If the EEPROM is not blank, erase the EEPROM using **BULK** Command.

Use MD command to verify that EEPROM is blank.

Create a project called **lab4** and type the following simple program using Micro-IDE:

```
ORG $0D00

LDAB #$55
RTS
```

Save the program as lab4.asm.

Note that the entry point of the program is $0D00 (instead of $0800) which is the starting address of the 768-byte EEPROM inside 68HC12.

Now build lab2.hex and download to the Training Board. If the program is built and downloaded successfully, you should see the message

>

on the Terminal window.

Important Note: BULK command should be used to erase the EEPROM before each download.

Use the MD command to display the contents of the EEPROM.

Run the program in the EEPROM using the

call 0d00

command.

Turn off the Training Board. Turn on the Training board. Connect back to the Training Board by selecting Toolkit, Terminal and Connect. Check to see if your program is still in the EEPROM.

Part II

Using Micro-IDE, create a project that contains a C program that prints “Hello” on the terminal screen. Build this program. After successful build, download to the board and execute from 0d00. Watch the terminal to see if it prints “hello” on the terminal.

LAB5

Overview

The purpose of this lab is to program the M68HC12 to detect keystrokes from a keypad.

Information

Many keypads are wired as a matrix of rows and columns. Figure 4 shows the internal connections of a 4 row by 3 column keypad.

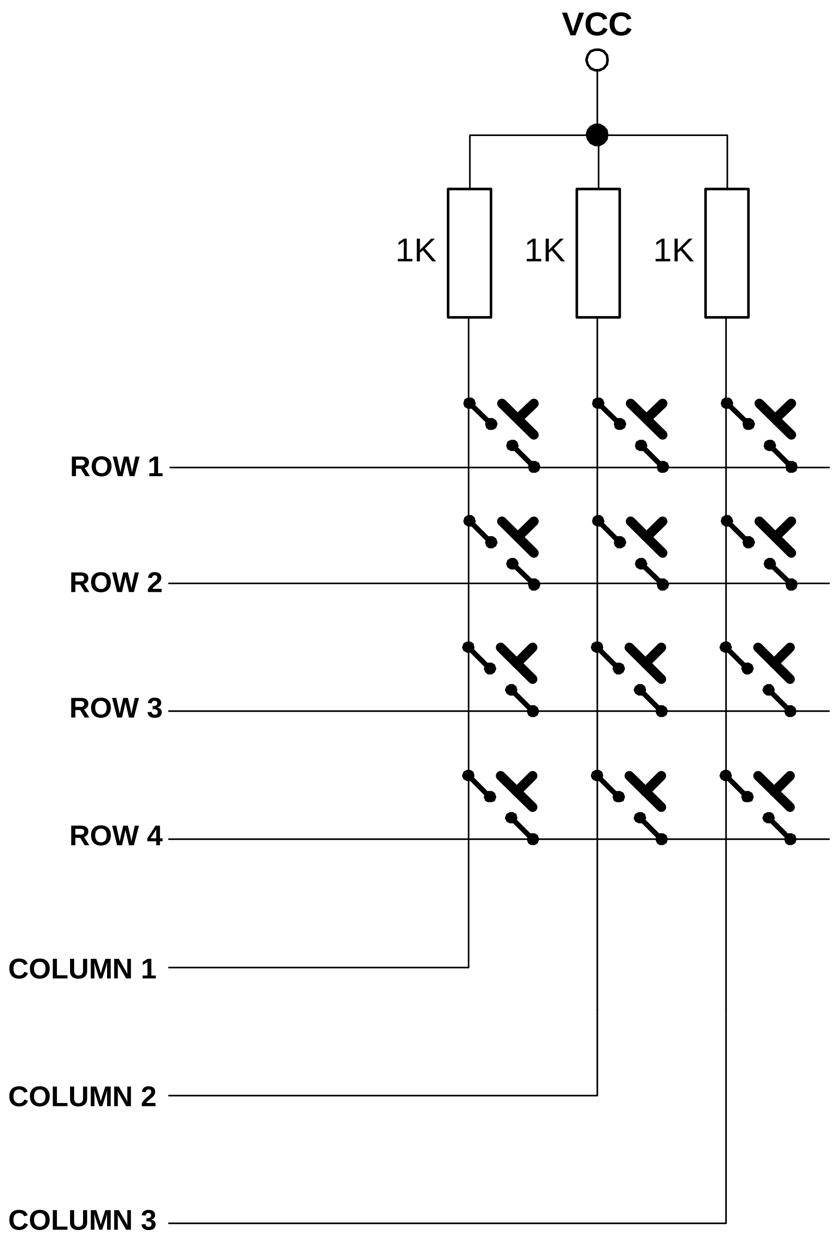


Figure 4

Matrix connection saves on the number of connections and micro-controller port lines. For example, a 3 row by 4 column keypad would require 13 wires (12 + ground) if each key was individually connected to micro-controller ports. Using the matrix approach and scanning the keypad under software control reduces the number of wires and port pins to 7 (3 rows + 4 columns).

When a key is pressed, the row for that key will be physically connected to the column for that key. Therefore, the port input for the column will be at the same logic level as the port output for the row.

Since the columns (inputs) are normally at the HIGH logic level due to pull-up resistors, the only way to make a column LOW will be press a key and make the row for that key LOW. By periodically strobing each row LOW one row at a time, and reading the column input levels during each strobe, one can determine which key is pressed.

This is illustrated in the table below for the 3 by 4 keypad. In the Row Mask, Row 1 is assigned to the Most Significant Bit and Row 4 is assigned to the Least Significant Bit. Similarly in the Column Mask, Column 1 is assigned to the Most Significant Bit and Column 3 is assigned to the Least Significant Bit.

Action	**Row Mask**	**Column Mask**
No keys were pressed	XXX	111
Row 1 Column 1 key pressed	0111	011
Row 1 Column 2 key pressed	0111	101
Row 1 Column 3 key pressed	0111	110
Row 2 Column 1 key pressed	1011	011
Row 2 Column 2 key pressed	1011	101
Row 2 Column 3 key pressed	1011	110
Row 3 Column 1 key pressed	1101	011
Row 3 Column 1 key pressed	1101	101
Row 3 Column 1 key pressed	1101	110
Row 4 Column 1 key pressed	1110	011
Row 4 Column 1 key pressed	1110	101
Row 4 Column 1 key pressed	1110	110

A matrix keypad can be scanned for keystrokes using the following algorithm:

```
Assign a micro-controller port line to each row and column.
Configure rows as output and columns as input.

For all the rows
{
     Make one of the rows LOW
     Read the columns
     If any of the columns is LOW, then
     {
          The key that connects that column and current row is pressed.
          Display the key that was pressed to the Terminal Window.
     }
}
```

Instructions

Determine the internal wiring of the Numeric Keypad that comes with the Training Kit using an Ohmmeter.

Fill in the tables of Keypad Row and Column assignments on the attached Lab5 Exercise Sheet.

Fill in the Keypad Scan Mask table on the attached Lab5 Exercise Sheet. This table shows what hexadecimal number will be used on the row outputs to scan each column.

Write a 68HC12 program that implements the keypad scan algorithm that is given in the Information section of this lab. Save the program as keypad.asm.

Build the program and download to the Training Board. Each time a key is pressed on the Numeric Keypad, the program should display the key that was pressed.