# 68HC11 Training Lab
# Student Exercise Book

Date: 15 August, 1999

Document Revision:1.02

BiPOM Electronics

16301 Blue Ridge Road, Missouri City, Texas 77489 USA
Telephone: (713) 283-9970 Fax: (281) 416 - 2806
E-mail: info@bipom.com
Web: www.bipom.com

WARRANTY:

BiPOM Electronics warrants 68HC11 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 68HC11 Training Lab is to familiarize the student with developing practical applications for the Motorola 68HC11 series of micro-controllers. 68HC11 is one of the most popular micro-controllers on the market with applications ranging from industrial, medical, home automation to automotive.

The 68HC11 Training Kit consists of the following components:

- Motorola M68HC11 EVBU Micro-controller Board ( Training Board )
- Motorola M68HC11 Assembler (as11)
- Micro-IDE Integrated Development Environment
- Terminal Board
- Terminal Expansion Cable
- Serial Cable
- 1K Resistors ( 6 pieces )
- 5K Variable Resistor
- Red LED
- Yellow LED
- Green LED
- Numeric Keypad

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

- IBM Compatible Personal Computer (PC) running Windows 95/98 or NT 4.0.
  ( 68HC11 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.

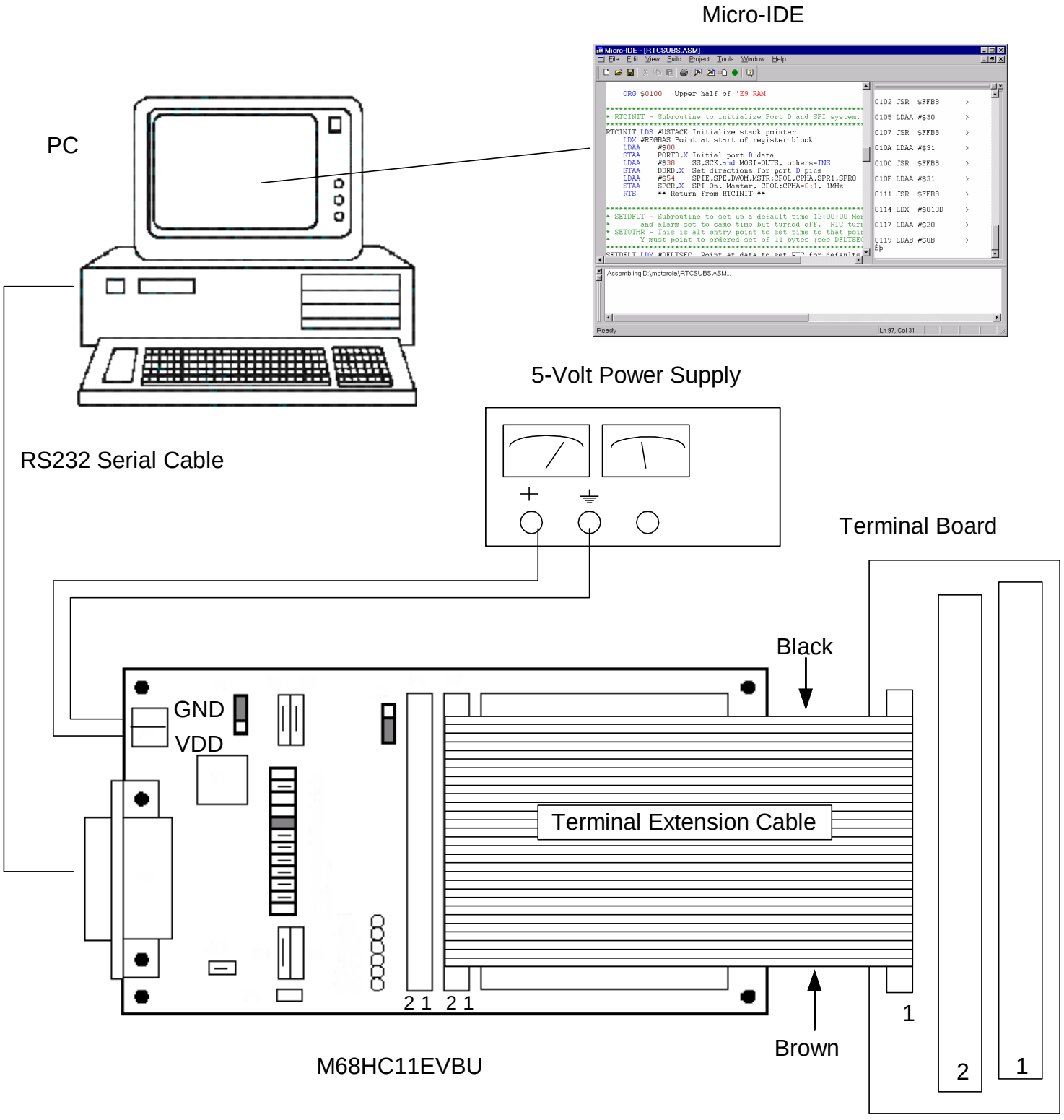

Micro-IDE
PC
5-Volt Power Supply
RS232 Serial Cable
Terminal Board
Black
GND
VDD
Terminal Extension Cable
2 1 2 1
1
Brown
2
1
M68HC11EVBU

Figure 1

Each student or group needs to log on to Windows before using the 68HC11 Training Kit. Enter the user name and password that your instructor has given to you before the Lab. Make sure to log out when you are done with the PC 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 ( Motorola AS11 Assembler )

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

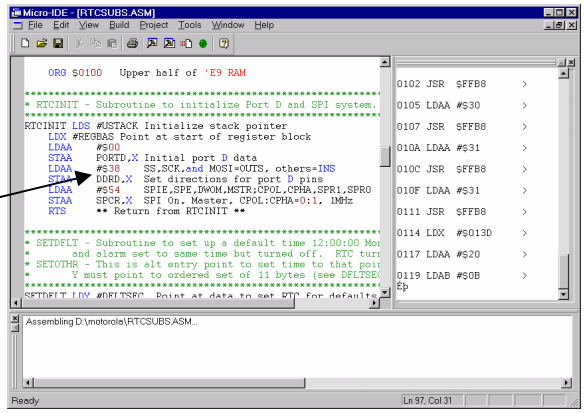
4. Run and debug the program on the Training Board using Micro-IDE Terminal Window.

Figure 2 illustrates these steps.

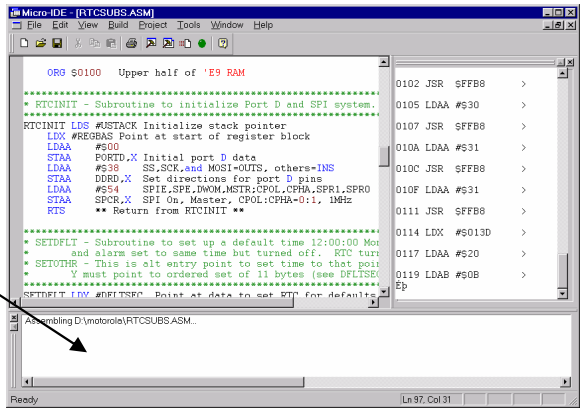
The 68HC11 micro-controller on the Training Board is factory programmed with a monitor/debugger environment called BUFFALO ( Bit User Fast Friendly Aid to Logical Operations ). You can send commands and run programs on the 68HC11 using BUFFALO features. BUFFALO communicates with the PC through Micro-IDE's Terminal Window.

Refer the M68HC11EVBU Universal Evaluation Board User's Manual to get familiar with the Training Board and the BUFFALO monitor commands.

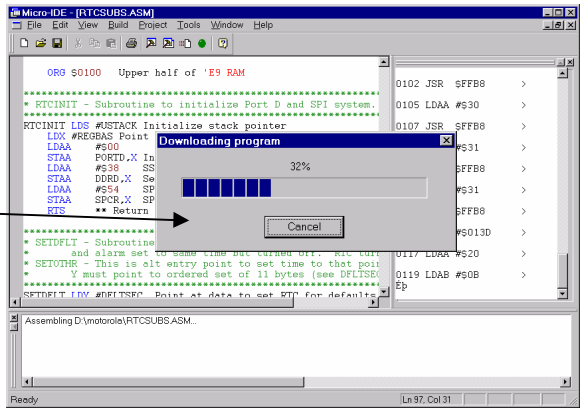
Create 68HC11 Assembly Programs




Assemble using AS11.EXE




Download to M68HC11EVBU




Execute program

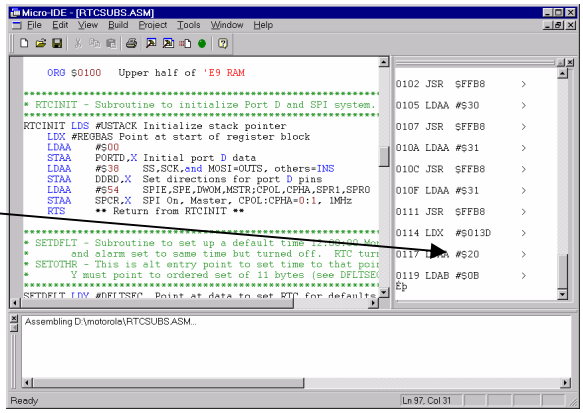



Figure 2

# LAB1

## Overview

The purpose of this lab is to familiarize the student with the Motorola M68HC11EVBU micro-controller board ( Training Board ) and the program development environment. In this lab, students will create a simple 68HC11 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 68HC11 to develop programs.

## Instructions

Type the following program using Micro-IDE editor. This program displays the string “hello” on the terminal screen and returns control to BUFFALO Monitor.

```
warmst  equ   $FF7C
outa    equ   $FFB8

     ORG  $0100

     LDAA #'H'
     JSR outa
     LDAA #'e'
     JSR outa
     LDAA #'l'
     JSR outa
     LDAA #'l'
     JSR outa
     LDAA #'o'
     JSR outa
     JSR warmst
```

Save the program as hello.asm.

Assemble the program by selecting Build and then Assemble from the menu. ( You can also assemble by clicking the Compile/Assemble button on the toolbar )

Download the program to the board by selecting Build and then Download from the menu. ( You can also download by clicking the Download button on the toolbar )

When the download is complete, you should see the message

**>done**

on the terminal screen.

Verify that the program is indeed downloaded by disassembling 68HC11 memory starting at address $0100.

Type **ASM 100**. This should display the first instruction in the program:

**0100 LDAA #$48**
**>**

By pressing ENTER several times, you can see the other instructions.

Press CTRL-C to exit the disassembly mode.

Now execute the program by using the G command ( short for GO ) on the BUFFALO Monitor. The syntax of G command is:

**G <start address>**

where <start address> is the starting address where the program execution begins.

Type G 100 to run the program hello.asm on the board.

You should see the following output:

**>G 100**
**Hello**
**>**

The program displays the string “Hello” and then returns to the BUFFALO monitor displaying the BUFFALO prompt ‘>’.

After completing Lab1, fill in the attached Lab1 Exercise Sheet and turn in to your instructor.

# LAB2

## Overview

The purpose of this lab is to familiarize the student with 68HC11 arithmetic and logic instructions. The student will also learn the BUFFALO Monitor's Trace commands to trace and debug the operation of a program.

The knowledge developed in this lab will be very useful in subsequent labs when working with the 68HC11 to develop programs.

## Information

BUFFALO 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 68HC11 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 $0100, RM command should be used to set the program counter to $0100.

## Instructions

Manually fill in the trace tables on the attached sheet titled 68HC11 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 68HC11 instructions

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  $0100
          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 ( $0100 ) of trace.asm. Use RM command in the Terminal window to set the program counter.

Use the T command to trace the program. BUFFALO will display the instruction that was executed and the contents of all the registers. If you enter T followed by a number, BUFFALO will execute that number of trace commands. For example, if you enter T3, BUFFALO 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 68HC11 to the outside world by developing programs to read digital inputs and write to digital outputs.

## Information

68HC11 has 5 input/output ports with 38 general-purpose input/output lines. Ports are grouped in units of 8 and are assigned a port letter. The following are the ports on the 68HC11:

PA0 through PA7
PB0 through PB7
PC0 through PC7
PD0 through PD7
PE0 through PE7 ( Port E can be used as both digital ports or analog inputs )

On the Training Board, the following ports are already used:

PA3 - Connected to XIRQ signal for Tracing programs.
PD0 - RS232 Serial port Transmit Line ( Used for communicating with the board )
PD1 - RS232 Serial port Receive Line (Used for communicating with the board )

These ports should not be used for user programs.

Each port has a memory-mapped address in the 68HC11's address space. The following are the addresses for the ports:

| Register name | Description | Address |
|---|---|---|
| PORTA | Port A Data Register | $1000 |
| PACTL | Pulse Accumulator Control | $1026 |
| PORTB | Port B Data Register | $1004 |
| PORTC | Port C Data Register | $1003 |
| PORTCL | Port C Latched | $1005 |
| DDRC | Port C Data Direction Register | $1007 |
| PORTD | Port D Data Register | $1008 |
| DDRD | Port D Data Direction Register | $1009 |
| PORTE | Port E Data Register | $100A |

Ports can be made input or output using the Port Direction Registers. 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 the switches and Light emitting Diodes (LED's) to the Training Board using the following wiring diagram:

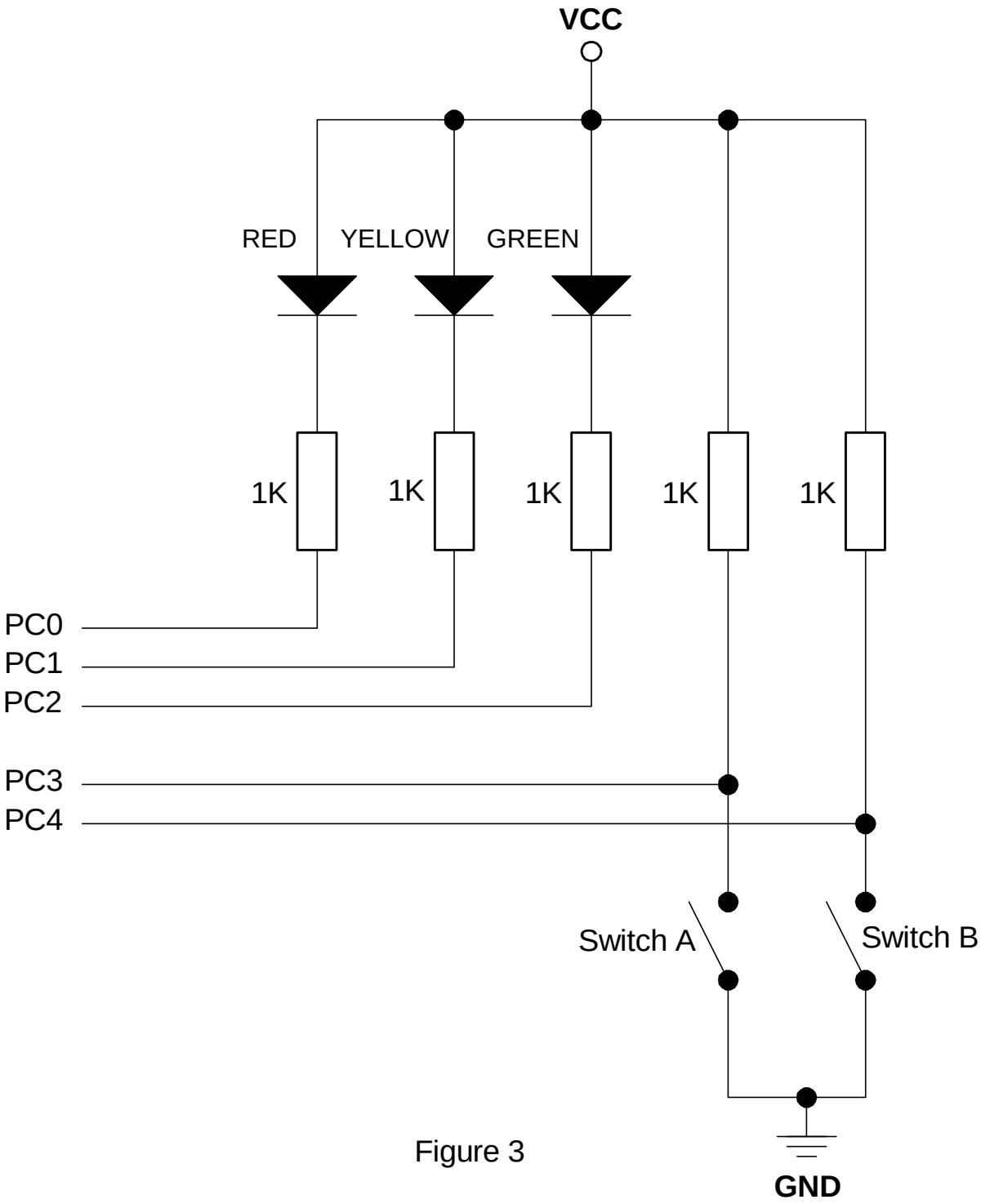



Figure 3

Write a 68HC11 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 68HC11's Electrically Erasable Read Only Memory ( EEPROM).

The lab is separated into two sections:

1. Downloading a program into EEPROM and running the program from EEPROM.
2. Writing a program to read to, write from and erase the EEPROM.

## Information

68HC11 has 512 bytes of EEPROM starting at address $B600 and ending at address $B7FF. 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.

The other advantage of using the EEPROM is that the 512-byte EEPROM has twice storage as the 256-byte RAM available for user programs.

Refer to the Subtitle 'EEPROM' under Section 'Operating Modes and On-Chip Memory' in Motorola M68HC11 E Series Technical Data manual.

## Instructions

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.

Type the following simple program using Micro-IDE:

```
warmst    equ   $FF7C
outa      equ   $FFB8

main      org $b600
          ldaa #'*'
          jsr outa
          jsr warmst
```

Save the program as lab4.asm.

Note that the entry point of the program is $B600 ( instead of $0100 ) which is the starting address of the 512-byte EEPROM inside 68HC11.

Since writing to EEPROM is slow, download speed must be decreased by changing the baud rate of the Training Board to 300 baud.

In BUFFALO monitor, type MM 102B and press ENTER. BUFFALO displays

102B 30

Type 35 and press ENTER. BUFFALO will not respond because it is now set to 300 baud but our Terminal window is still set to 9600 baud ( default value ).

Change the baud rate for the Terminal window. To do this, first disconnect from the Training Board by selecting Toolkit, Terminal, Disconnect. Then select Toolkit, Options and then Terminal tab. Change the baud rate to 300. Do not change other parameters. Click OK.

Connect back to the Training Board by selecting Toolkit, Terminal and Connect. Press Enter. Training Board should now respond because both the Terminal window and the Training Board are now communicating at 300 baud. Note that the Terminal window is now updating slower due to lower communications speed.

Change the Loader baud rate also to 300 baud by selecting Toolkit, Options and then Loader tab. Change the baud rate to 300. Do not change other parameters. Click OK.

Now build lab2.asm as lab2.s19 and download to the Training Board. If the program is built and downloaded successfully, you should see the message

**>done**

on the Terminal window.

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

Run the program in the EEPROM using the G command.

When you are finished, change the Terminal, Loader and Training Board baud rates back to 9600:

To change the Training Board baud rate, type MM102B and press ENTER. BUFFALO will display 102B 35. Type 30 and press ENTER. BUFFALO will not respond because it is now set to 9600 baud but our Terminal window is still set to 300 baud ( default value ).

To change the Terminal baud rate, select Toolkit, Options and then Terminal tab. Set the baud rate to 9600. Click OK.

To change the Loader baud rate, select Toolkit, Options and then Loader tab. Set the baud rate to 9600. Click OK.

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.

Turn off the Training Board. Move Jumper J2 from pins 2 and 3 to pins 1 and 2. This will cause the program to be started from the EEPROM when the Training Board is turned on or when the Training Board is reset through the red RESET button on the board.

Turn on the Training Board. The program in the EEPROM should run. Press the RESET button on the Training Board. The program in the EEPROM should run again.

Write a program that copies the string “Hello” to the beginning of the EEPROM. Download the program to RAM at address $0100 and execute. Check to see if the string was copied to EEPROM using MD command.

# LAB5

## Overview

The purpose of this lab is to read voltage values using the M68HC11's internal Analog-To-Digital Converter.

## Information

One of the core components of M68HC11 is an 8-channel Analog-To-Digital Converter ( ADC ). The ADC has 8 bits of resolution which means that the input range is represented by values 0 through 255 ( 2^8 – 1 ). ADC input voltage range is 0 to 5 Volts. A value of 0 corresponds to 0 Volts and a value of 255 corresponds to 5 Volts.

ADC is accessed through a control register (ADCTL) and 4 data registers ( ADR1. ADR2, ADR3 and ADR4 ).

## Instructions

Connect a variable resistor to one of the analog input channels on Port E as shown in Figure 4.

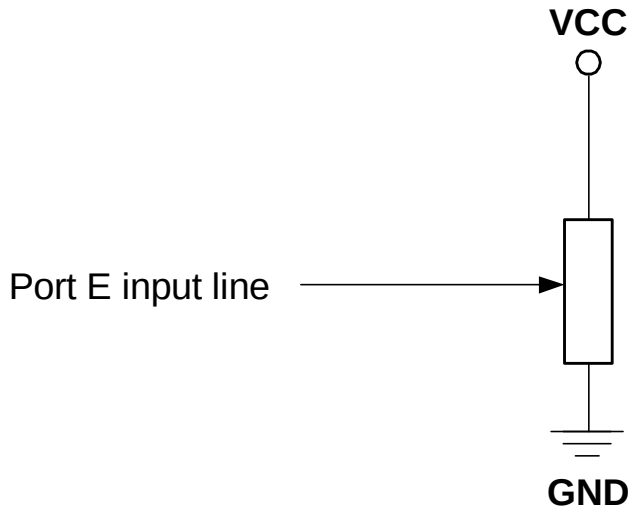



Figure 4

Write a program that reads the voltage continuously from the analog channel that you selected and displays it on the Terminal Window.

Connect a digital voltmeter to the middle terminal of the variable resistor. Turn the knob to obtain the following voltage reading on the voltmeter:

0 Volts ( or as close to 0 as possible )
2.5 Volts
5 Volts

Record the readings from the Terminal window for each input voltage. Is the ADC linear ?

# LAB6

## Overview

The purpose of this lab is to program the interrupt handling capability of the M68HC11.

## Information

There are 18 interrupt vectors that support 22 interrupt sources on the 68HC11. 15 of the interrupts are generated by sources within the 68HC11.

The interrupts are listed in Figure 3:

| Interrupt Source | Interrupt Vector Address |
|---|---|
| Reserved | FFC0 – FFD5 |
| SCI Serial port<br>• SCI Receive Data Register Full<br>• SCI Receive Overrun<br>• SCI Transmit Data Register Empty<br>• SCI Transmit Complete<br>• SCI Idle Line Detect | FFD6, FFD7 |
| SPI Serial Transfer Complete | FFD8, FFD9 |
| Pulse Accumulator Input Edge | FFDA, FFDB |
| Pulse Accumulator Overflow | FFDC, FFDD |
| Timer Overflow | FFDE, FFDF |
| Timer Input Capture 4/Output Compare 5 | FFE0, FFE1 |
| Timer Output Compare 4 | FFE2, FFE3 |
| Timer Output Compare 3 | FFE4, FFE5 |
| Timer Output Compare 2 | FFE6, FFE7 |
| Timer Output Compare 1 | FFE8, FFE9 |
| Timer Input Capture 3 | FFEA, FFEB |
| Timer Input Capture 2 | FFEC, FFED |
| Timer Input Capture 1 | FFEE, FFEF |
| Real-Time Interrupt | FFF0, FFF1 |
| /IRQ | FFF2, FFF3 |
| /XIRQ | FFF4, FFF5 |
| Software Interrupt | FFF6, FFF7 |
| Illegal Opcode | FFF8, FFF9 |
| COP Failure | FFFA, FFFB |
| Clock Monitor Fail | FFFC, FFFD |
| RESET | FFFE, FFFF |

**Instructions**

Apply the following correction from Motorola to the M68HC11EVBU Universal Evaluation Board User's Manual before starting the lab:

The M68HC11EVBU Universal Evaluation Board User[1]s Manual contains a printing error regarding the pseudo-interrupt vectors to be used when running the BUFFALO monitor. These vectors are located in $0XXX space, not $EXXX space as the manual states. Go to Table 3-2 ( titled 'Interrupt Vector Jump Table' ) in the manual and change $E's with $0's with a pen. The rest of the information is correct.

Write an interrupt handler program to generate a Timer interrupt every second and display the message "Hello" on the Terminal window.

# LAB7

## Overview

The purpose of this lab is to program the M68HC11 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.

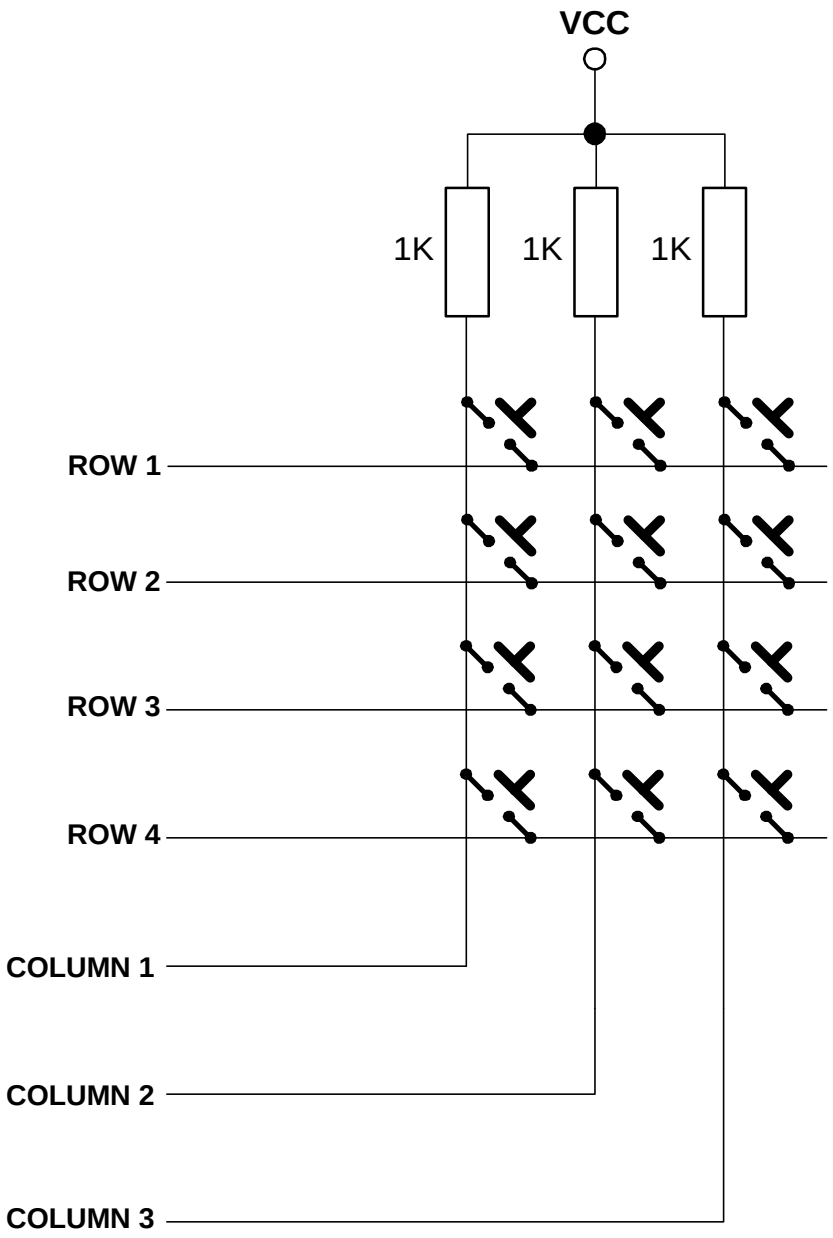



Figure 4

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 Lab7 Exercise Sheet.

Fill in the Keypad Scan Mask table on the attached Lab7 Exercise Sheet. This table shows what hexadecimal number will be used on the row outputs to scan each column.

Write a 68HC11 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.

# Appendix A: List of available BUFFALO Subroutines

| Name | Address | Description |
|---|---|---|
| .BPCLR | $FF7F | Clear Breakpoint Table |
| .BUFFAR | $FF88 | Read 4-digit Hex argument from input buffer into SHFTREG ($009C - $009D) |
| .CHGBYT | $FF8E | Write value from SHFTREG+1 ($00D9) to memory location pointed to by X |
| .CHKABR | $FF9D | Monitor input for (CTL)X, (DEL), OR (CTL)W |
| .DCHEK | $FFA6 | Test character in A and return with CCR Z bit set if character is a delimiter (carriage return or white-space) |
| .DECBUF | $FF97 | Decrement pointer into input buffer |
| .HEXBIN | $FF85 | Convert ASCII character in A to 4-bit binary number placing it into SHFTREG ($009C - $009D) |
| .INCBUF | $FF94 | Increment pointer into input buffer |
| .INCHAR | $FFCD | Input ASCII character to A and echo back. Wait till character is received. |
| .INIT | $FFA9 | Initialize I/O device |
| .INPUT | $FFAC | Read I/O device |
| .OUT1BS | $FFBE | Convert byte at address in X to two ASCII characters and outputs followed by a space. Returns with X pointing to next byte. |
| .OUT1BY | $FFBB | Convert byte at address in X to two ASCII characters and outputs. Returns with X pointing to next byte. |
| .OUT2BS | $FFC1 | Convert word at address in X to four ASCII characters and outputs followed by a space. Returns with X pointing to next byte. |
| .OUTA | $FFB8 | Output A (ASCII character) |
| .OUTCRT | $FFC4 | Output ASCII carriage return followed by a line feed |
| .OUTLHL | $FFB2 | Convert left nibble of A to ASCII and output to terminal |
| .OUTPUT | $FFAF | Write I/O device |
| .OUTRHL | $FFB5 | Convert right nibble of A to ASCII and output to terminal |
| .OUTST0 | $FFCA | Output string of ASCII bytes pointed to by address in X until character is an EOT ($04) No leading carriage return. |
| .OUTSTR | $FFC7 | Output string of ASCII bytes pointed to by address in X until character is an EOT ($04) |
| .READBU | $FF91 | Read next character from INBUFF ($0071) |
| .RPRINT | $FF82 | Display the user's Registers |
| .TERMAR | $FF8B | Read 4-digit Hex argument from terminal device into SHFTREG ($009C - $009D) |
| .UPCASE | $FFA0 | If character in A is lower case alpha make it upper case |
| .VECINIT | $FFD0 | Used to initialize the indirect interrupt vector area of RAM. (Needed when starting at $B600) |
| .WARMST | $FF7C | Go to the ">" prompt |
| .WCHEK | $FFA3 | Test character in A and return with CCR Z bit set if character is a white-space (space, comma, tab) |
| .WSKIP | $FF9A | Read input buffer until non-white-space character found |