

LABORATORY MANUAL

COEN 311

Computer Organization and Software

By:

K. Vitoroulis

&

Dr. S. Tahar

Concordia University

Department of Electrical and Computer Engineering

Winter 2005

Revised: July 2010, T. Obuchowicz

Version 1.2

TABLE OF CONTENTS

	Page
LAB 1	1
LAB 2	7
LAB 3	9
LAB 4	12
LAB 5	13
APPENDICES	21
APPENDIX 1: Assembly programs	
APPENDIX 2: Selected CPU32Bug Debugger commands	
APPENDIX 3: Motorola S-Record format	
APPENDIX 4: Selected MC68000 instructions	
APPENDIX 5: LCD datasheet, ASCII table, parallel port documents	
APPENDIX 6: A sample MC68000 assembly language program	
APPENDIX 7: A sample MC68000 listing with a subroutine	

LAB 1

OBJECTIVES

In this experiment you will:

- become familiar with the CMM332 board
- become familiar with the AxIDE assembler
- create an ASCII text file containing a simple MC68000 assembly language program
- establish a serial interface communication between a computer and the CMM332 board
- program the CMM332 board
- single-step through a downloaded program in order to verify its correct functionality.

INTRODUCTION

The CMM332 board provided by Axiom is an extension of the Motorola MC68332 BCC (Business Card Computer) product.

The components of the CMM332 device are the following

- MC68332 Microcontroller Unit (MCU).
- Two 64K x 16 erasable programmable read only memories (EPROMs) that contain the M68CPU32BUG debugger software
- 32k x 16 bit, byte addressable random access memory (RAM)

The CMM332 can be connected to a Personal Computer via serial port. Any ANSI terminal emulator can be used to communicate with the device. In this lab will we be using the AxIDE software which is essentially an ANSI terminal emulator.

PRE-LAB:

The following is to be performed prior to attending the scheduled lab session:

- become familiar with the CPU332BUG debugger commands to be used in the Lab by reading Appendix 2.
- read Lab 1 in its entirety.

Part 1: Setting up serial communication with the board.

Connect the serial cable and the power connector to the CMM332 board. Inspect the serial cable connector from the PC to the board. Inspect the power connection from the power bar underneath the desk to the board.

To set up a serial communication interface to communicate with the CMM332 board you can use any ANSI compatible terminal emulator. On the Windows XP platform you can use either the AxIDE application (installed on the lab-cut) or HyperTerminal (comes with windows). The AxIDE application is preferred.

Once the ANSI terminal emulator is run, select File -> Options and set up the following parameters for the serial port:

```
Baud Rate:          9600
Parity:             None
Data Bits:          8
Stop Bits:          1
Handshaking Xon/Xoff : Off
```

Once the serial port is setup power up the CMM332 board and press the reset button. If the serial port is setup properly the following menu will appear on the terminal window:

```
-----
CMM332 AXM-0089 Utilities
-----

d. Debug Monitor - CPU32Bug
f. Flash EEPROM Utilities
t. Test Hardware
Select:
```

Part 2: First use of the debugger

From the main menu (see Part 1) select option (d) to run the debugger. The following prompt will appear:

```
CPU32Bug>
```

You can now issue commands to the debugger. A list of commands can be displayed by typing 'HE' on the prompt. Details for some selected commands and their syntax can be found in Appendix 2. The CPU32BUG debugger manual, available in the accompanying lab document set, contains the complete documentation of the debugger commands and features.

Perform the following:

- display the register contents.
- modify the register contents using the RM and the RS commands.
- display the memory content at locations \$0000 and \$3000.
- change the memory content at location \$3001 to the byte value \$CA.
- fill the memory starting at address \$4000 with the following byte sequence: \$474F4F44204A4F4212121202020. Display the memory content at location 4000 using the MD command.

Part 3: Compiling a program using the Windows AS32 assembler.

In your Unix account create an appropriate directory structure for the experiments. For example have a main 'COEN311' directory with subdirectories for each experiment (EXP1, EXP2 etc.). In the subdirectory for the first experiment create an ASCII text file (use any of the available text editors such as 'nedit', 'vi', 'emacs'. Alternatively a Windows text editor such as Microsoft Notepad may be used, save the file in your U:\ in an appropriately named folder) containing the following:

```
* Experiment 1
* Name: Ted Obuchowicz, put your name here
* Date: August 31, 2009
```

```
ORG $3000
```

```
CLR.L D1
```

```
CLR.L D2
```

```
CLR.L D3
```

```
CLR.L D4
```

```
MOVE.B    #$01,D1
```

```
MOVE.B    #$02,D2
```

```
MOVE.B    #$03,D3
```

```
START
```

```
ADD        D1,D1
```

```
ADD        D2,D2
```

```
ADD        D2,D3
```

```
MULS       D1,D3
```

```
DIVS       D2,D1
```

```
SUB        D0,D0
```

```
BEQ        START
```

```
END
```

Save this program under the filename "exp1.asm".

The AxIDE assembler is somewhat quirky. Here a few hints which will prevent assembly errors:

- lines containing only comments should begin in column 1 as in:

```
* Experiment 1
```

```
* T. Obuchowicz
```

```
* July 27, 2006
```

```
* comments should begin in column 1, this will cause an error
```

- mnemonics representing executable instructions and assembler directives should begin after a TAB as in:

```
ORG $3000
```

```
CLR.L D1
```

```
CLR.L D2    * executable instructions should start after a TAB
```

- The assembler is very particular about whitespace characters between operands or labels as in:

```
ADD D0,  D1    * this will assemble with 0 warnings 0 error
*
*              but the machine code generated will be
*              INCORRECT.
```

```
ADD D0,D1      * note no blank after the , this is CORRECT
*
*              and will produce correct machine code.
```

```
LOOP:
```

```
    <some instructions>
```

```
    DBRA D3,  LOOP * this will assemble, but the machine code
*
*                  will be incorrect but no errors will be
*                  reported by the assembler.
```

```
LOOP:
```

```
    <some instructions>
```

```
    DBRA D3,LOOP  * this is correct, note that there is no
*
*                  space after the ,
```

To assemble the program perform the following steps:

1. Open a command prompt (Start --> Accessories --> Command Prompt)
2. Issue the command: `path=%path%;"C:\Program Files\as32"` This command will add the location of the assembler program to the shell `path` variable.

3. Change the directory to the location of the source code by specifying U: to change to the U:\ drive then use the `cd <name_of_directory>` to the directory containing your source code.
4. Issue the command: `as32 -l exp1.asm > exp1.lst` Note that the `as32` command can be recognized because of the addition of its location to the `path` variable in step (2)

The correct execution of the AS32 command produces the following files in the same directory as the source code:

- **exp1.lst**: the listing file produced by the compilation of the source code. This code will contain the machine code given in hexadecimal notation and the source code of the program side by side. The contents of this file should be the following:

```

                                * Experiment 1
00003000                      ORG $3000

00003000 4281                  CLR.L D1
00003002 4282                  CLR.L D2
00003004 4283                  CLR.L D3
00003006 4284                  CLR.L D4

00003008 123c 0001             MOVE.B    #$01,D1
0000300c 143c 0002             MOVE.B    #$02,D2
00003010 163c 0003             MOVE.B    #$03,D3

                                START
00003014 d241                  ADD     D1,D1
00003016 d442                  ADD     D2,D2
00003018 d642                  ADD     D2,D3
0000301a c7c1                  MULS    D1,D3
0000301c 83c2                  DIVS    D2,D1
0000301e 9040                  SUB     D0,D0
00003020 67f2                  BEQ     START
                                END

====      0 Error(s)
====      0 Warning(s)

```

The numbers on the left hand side of the listing file specify main memory addresses in hexadecimal notation. The addresses start from the number specified with the ORG assembler directive (ORG is short hand notation for ORIGIN). Motorola 68000 instruc-

tions are of various sizes. For example, a CLR.L D1 instruction occupies two bytes in main memory. The numbers to the right of the main memory addresses specify the machine code in hexadecimal format. When the program is loaded into main memory, the PC register will be loaded with the starting address of the first instruction. As each instruction is fetched from main memory and executed by the microprocessor the value of PC will be updated to point to the next instruction to be fetched and executed.

• **exp1.s19:** this is the assembled program in a format appropriate for downloading to the board. It is an ASCII text file. Take some time to open the file and read its content. The S-record format will be studied in greater detail in Lab 3.

Part 4: Downloading the program to the CMM332 board.

To download the program go to the debugger window (the AxIDE software window) and do the following:

- Issue the command LO at the debugger prompt
- Click the 'Upload' button (or select the 'Upload' menu option) in the AxIDE software.
- In the window that appears specify either the "exp1.s19" or the "exp1b.s" file and click the OK button.
- if the prompt does not return to 'CPU32Bug>' press the <enter> key a few times.

The program should now be loaded at location \$3000 of the memory.

Part 5: Single step execution of a program in the debugger.

In Part 4 a program was downloaded to the board. To execute the program in single step mode do the following:

- set the program counter to location \$3000 (make use of the RS command)
- enter the command T at the debugger prompt. This command will execute an instruction, show the register content as well as the following instruction at the next location where the PC points.
- carefully examine the register contents and verify that the program runs as expected.

QUESTIONS

1. What is the definition of the following terms: CPU, MCU. What is the difference between a CPU and an MCU?
2. What is a register? How is a register implemented in hardware?
3. What is the purpose of the PC register, the SR register, the D0-D7 and A0-A7 registers?
4. What is the difference between a source code file (.asm) and a listing file (.lst)?
5. What is machine code?
6. What information does an S-record file contain? What format is an S-record file in?

LAB 2

OBJECTIVES

In this experiment you will:

- Assemble, and download a given assembly program.
- Change the program's functionality by directly altering the memory content.
- Learn the differences between the G (GO) debugger command and the T (TRACE) command.

INTRODUCTION

The CPU can only execute a program which is in machine language form and already stored in memory.

In Lab 1 we saw how a program is converted into machine language, stored in an S-Record file and loaded into the main memory for execution.

A programmer these days will very rarely, if ever, need to work directly with machine language. It is however important to have a good understanding of the process involved in the machine code generation and execution by the CPU.

To convert assembly into machine code one needs the op-code for the instructions. The instruction op-codes are well documented in the programmer's reference documentation of the CPU at hand. The 68000 programmer's documentation is available in the additional lab documents. Appendix 4 contains a very small portion of that documentation for some frequently used commands. The text book also contains the instructions along with their op-codes.

Take a look at some of the instructions provided in Appendix 4 and identify the Instruction Format and the tables that provide values for the Instruction Format fields.

In this experiment you will be downloading a program and then directly alter the memory content to change the functionality of that program.

PRELAB

The following is to be performed prior to attending the scheduled lab session:

- Read Lab 2 in its entirety.
- Create the text file containing the source code for this lab in Appendix 1 in your computer account.
- Familiarize yourself with Appendix 4 and the "hand-assembly" process. If necessary, refer to your textbook. Hand assemble the changes indicated in the source code for this lab.

Part 1: Assembly program compilation and download

Appendix 1 contains the assembly program for this experiment. Create the program using a text editor and save the file. Assemble, download and trace its execution.

Part 2: Machine code identification

Obtain the machine code for this program. You can use the MD command of the debugger to view the memory content after you download your program. You can also obtain the machine code from the listing file.

The memory content is displayed in hexadecimal format. The machine code is the binary version of the memory content.

Part 3: Direct modification of the program in memory.

Use the MC68000 instruction reference found in Appendix 4 and change the machine code to directly implement the changes outlined Appendix 1 listed in the source code for this lab. If you need information for commands not described in Appendix 4, use your textbook. Alternatively, one may edit the source code to contain the desired program modification, and then assemble the code to obtain the listing file which will contain the desired machine code. The appropriate memory locations may then be modified accordingly using either the MS or MM debugger commands.

Trace the execution of the modified program to verify that it functions correctly.

Part 4: Execution of the program with the GO debugger command.

Reload the machine code of the original program given in Appendix 1 and run the program using the GO debugger command instead of the T command. Make sure that you set the value of the PC to the starting address of the program before entering the GO command from the debugger prompt.

QUESTIONS

1. Which debugger commands are used in this lab?
2. Identify all addressing modes used in the program.
3. What does the original program perform? What does the modified program do?
4. Compare the opcode bits (the most significant 4 bits of the instruction) as specified in Appendix 4 for a SUB instruction with the opcode given in your modified listing as produced by the AS32 assembler. What can you conclude with the information given for the SUB instruction in Appendix 4?
5. What is the difference between the T and GO debugger commands?

LAB 3

OBJECTIVES

In this experiment you will:

- Manually produce the machine code and the S-Record for a given assembly program
- Program the CMM-332 board with your own generated code.

INTRODUCTION

Once assembly code is written, an assembler program is used to process the code and generate the corresponding machine language instructions.

The machine code is the only thing a CPU can execute. The machine code is accessed by the CPU only after it is loaded in the main memory. A loader program is usually used to transfer the machine code to the memory.

Motorola developed the S-record format to facilitate the transfer of data from a computer to a device (or another computer). For our case the S-record format is used to transfer data to the CMM332 board.

An S-record file is a simple ASCII text file that contains all the necessary information for the data to be transferred. Such information includes:

- target address location
- length of a block of data
- the data itself
- checksum information to ensure the data is transferred correctly

More detailed information about the S-Record format can be found in Appendix 3.

Once assembly code is generated, the assembler produces the appropriate S-record file which can be recognized by the `.s` or `.s19` extension to the filename. This file incorporates the machine language instructions together with other information that is necessary for loading the machine code into main memory.

For this experiment you can use Appendix 4, the 68000 programmer's reference manual as a reference for the op-code of the assembly instructions. Alternatively, the appropriate section of the textbook may be used as a reference.

PRELAB

The following is to be performed prior to attending the scheduled lab session:

- Read the contents of this lab in its entirety.

- Read Appendix 3 in its entirety.
- Generate the S-record file which is to be downloaded to the microcontroller board.

Part 1: Machine code generation

The assembly program for this experiment is in Appendix 1.

Generate the machine code that corresponds to this program into machine code in hexadecimal format.

Part 2: S-Record creation

Generate the S-record file for the machine code created in Part 1.

In order to generate the S-record file a thorough understanding of the S-record format is needed. Refer to the Appendix 3 for the details of the S-record format.

The following example illustrates the steps for the generation of an S-record. Note that the spaces between the numbers in the steps below are for readability and should not be there in the actual S-record file.

Example program:

```
ORG $4000
MOVE.L    #$1234,D0
MOVE.L    D0,D1
```

Step 1: Generate the machine code in byte code format:

```
20 3C 00 00 12 34
22 00
```

Step 2: Break the code into blocks. For this example, 2 blocks will be used, even though one would be sufficient.

```
Block 1: 20 3C 00 00 12 34
Block 2: 22 00
```

Step 3: Append to the left the starting address, or the address location where the blocks will be stored. Address \$4000 will be used to store this program as specified by the ORG command in the code:

```
40 00 20 3C 00 00 12 23
40 06 22 00
```

Step 4: Count the character pairs (bytes) of each block in step 3. Add 1 to this number and append the final result to the left of each block.

```
09 40 00 20 3C 00 00 12 34
05 40 06 22 00
```

Step 5: Add the values of the bytes of each block and append the one's complement of the least two significant bytes of the sum to the right side of the block:

```
09 40 00 20 3C 00 00 12 34 14
05 40 06 22 00 92
```

Step 6: complete the s-record with the header record, the header types and the termination records:

```
S0 10 00 00 47 4F 20 48 41 42 53 20 47 4F 21 20 20 04
S1 09 40 00 20 3C 00 00 12 34 14
S1 05 40 06 22 00 92
S9 03 00 00 FC
```

Comment: The header and termination records are the first and last line in the above listing of step 6. Details about the format of these records can be found in Appendix 3.

Part 3: Downloading to the board.

Download your S-record file to the board. More details on how to download a file to the board can be found in the first experiment.

Part 4: Contrast with assembler output

Assemble the program using the MC68332 assembler to obtain the listing and the S-record files.

Once the listing and the S-record files are obtained, compare them with your hand generated files.

Part 5: Run the program.

Single step execute your program and observe the results.

QUESTIONS

1. What does the given assembly program do?
2. What is the purpose of the "checksum" field in the s-record format?
3. Why is it possible to have many different S-record files for a given assembly program?
4. Elaborate on the tasks of an assembler and a loader program.
5. In the given S0 record (Step 6 of Part 2), what is the ASCII string represented by the 13 bytes appearing after the 2 byte address field of 00 00?

LAB 4

OBJECTIVES

In this experiment you will:

Investigate subroutines

INTRODUCTION

Subroutines in assembly language are constructs that allow the programmer to reuse code of a task that has already been coded in an easier fashion. The subject of subroutines is covered in depth in your textbook.

PRELAB

The following is to be performed prior to attending the scheduled lab session:

- Read the contents of this lab in its entirety.
- Read the appropriate sections of your textbook to become familiar with the subroutine call/return mechanism and parameter passing via the run-time stack in Motorola 68000 assembly language.
- Create the source code for this lab.

Part 1: Subroutines

Rewrite the program given in the appendix as a subroutine where parameters are passed using the stack. Ensure that your the state of the stack is the same before and after the subroutine has run (i.e. it will be necessary to adjust the Stack Pointer). Assemble, download and trace the execution of the program. Monitor and record the stack contents, the Stack Pointer register the Program Counter register on every step of the execution immediately before and after the subroutine execution.

QUESTIONS

1. Which register is used as the Stack Pointer in the 68000 microprocessor?
2. Why is it important to modify the stack pointer register upon a return from a subroutine when parameters have been pushed onto the stack?
3. What is another way of passing parameters to a subroutine other than pushing them onto the stack?

LAB 5

OBJECTIVES

In this experiment you will interface a light emitting diode (LED) to the parallel port of the CMM 332 board. Additionally, the parallel port will be used to control the operation of a liquid crystal display (LCD).

INTRODUCTION

Liquid Crystal Display -- LCD:

The LCDs are devices that allow the visual display of information to the user. One of the many LCD types is the Alphanumeric LCD which is capable of displaying character and number literals to its screen.

LCD displays contain a driver IC which controls their operation. By far the most common driver IC for an alphanumeric LCD is the HD44780 from Hitachi. Other companies provide compatible drivers to the HD44780 because of its popularity. Their operation is the same.

The HD44780 (or compatible) LCD driver is controlled by the following input signals:

Table 1: HD44780 LCD driver control signals.

Signal	Description
R/W*	This is the read/write signal for the display. When R/W* is HIGH, information is read from the display. When R/W* is LOW, information (commands/characters) is written to the display.
RS	This is a register select signal. Its value specifies which of the internal registers (instruction register or data register) is read or written to.
E	This is an enable signal. When the value of this signal transits from HIGH to LOW then the information is written to or read from the LCD. Note that it is only at the TRANSITION of this signal that the read or write operation takes place
D0 ~ D7	8-bit data bus. Character information or commands to the display are placed on this bus. Also information read from the display is written to this bus. The reading or writing operation takes place only at the transition of the enable signal E from HIGH to LOW.

CMM332 board parallel port interface:

The CMM332 microcontroller board that we use in this laboratory features parallel port interfaces which we will use in this experiment to communicate with the LCD display.

The parallel ports of the microcontroller are initialized by writing to control registers which are mapped to specific memory locations. A summary of these registers is shown in table 2 below. Appendix 5 contains a more detailed description of these registers.

We will be using parallel ports E and F of the microcontroller

To control a parallel port you need to perform the following steps:

1. Configure the pins associated with the parallel port for data Input/Output (I/O) operation. This is achieved by writing a value of '0' to the appropriate bits of the 'Port pin Assignment Register' (see table 2).
2. Specify the data direction of the port. In other words whether the port will read data from an external source (INPUT) or if it will output data from the micro controller (OUTPUT). This is done by setting the values of the port bits in the 'Port Direction Register' to '1' (see table 2).
3. Read the data from the port or write the data to the port. This is done by reading or writing to the memory location of the 'Port Data Register' (see table 2).

Steps (1) and (2) are initialization steps and need to be performed once.

The registers to control the ports are listed below:

Table 2: Control registers for ports E and F.

Memory location	Register	Information
\$FFFA16	PORT E PIN ASSIGNMENT REGISTER (PEPAR)	This register specifies whether the pins associated with the port are used for parallel port IO or not. Assign value \$00F0 to use the last 4 pins for parallel IO, port E.
\$FFFA14	PORT E DATA DIRECTION (DDRE)	This register specifies the direction of the port. Assign value \$00FF to set the port to write mode.
\$FFFA10	PORT E DATA REGISTER (PEDR)	This register holds the data of the port. If you are writing to the port write to this location your data. If you are reading from the port read the value of this location.
\$FFFA1E	PORT F PIN ASSIGNMENT REGISTER (PFPAR)	This register specifies whether the pins associated with the port are used for parallel port IO or not. Assign value \$0000 to use all 8 pins for parallel IO, port F.

Table 2: Control registers for ports E and F.

Memory location	Register	Information
\$FFFA1C	PORT F DATA DIRECTION (DDRF)	This register specifies the direction of the port. Assign value \$00FF to set the port to write mode.
\$FFFA18	PORT F DATA REGISTER (PFDR)	This register holds the data of the port. If you are writing to the port write to this location your data. If you are reading from the port read the value of this location.

Below is sample code that initializes port F and writes value \$AB to it:

```

PORTF0    EQU        $FFFA18
DDRF      EQU        $FFFA1C
PFPAR     EQU        $FFFA1E
          ORG         $5000
START     MOVE.W      #$0000, PFPAR
          MOVE.W      #$00FF, DDRF
          MOVE.W      #$00AB, PORTF0

```

Connection of the LCD to the display:

The connection of the LCD to the CMM332 board was made by connecting the pins of the microcontroller ports E and F directly to the connector of the LCD. The table below shows the pin to pin correspondence:

Table 3: LCD to parallel ports connection

Port bit	E3	E2	E1	E0	F7	F6	F5	F4	F3	F2	F1	F0
LCD signal		E	R/W*	RS	D7	D6	D5	D4	D3	D2	D1	D0

Sending commands to the LCD display:

The LCD datasheet in Appendix 5 contains a table with the commands that can be issued to the display. An ASCII character table of the display is also there for reference.

Familiarize yourself with these tables to understand how to control and send data to the LCD module.

The character table is easy to interpret: the value of a character is formed by combining the column number and row number. For example character “A” has a hexadecimal value of \$41 and character “m” has a value of \$6D.

The LCD command table is also easy to read. For example the second row of the table describes the LCD instruction ‘Clear Display’. The RS, R/W*, and D0-D7 signals for that instruction are given. They are: RS R/W* D7~D0 = 0 0 0 0 0 0 0 0 1.

In order to send this ‘Clear Display’ instruction to the display, the following steps are needed:

1. write the binary value 00000001 (or in HEX: \$01) to the port F. This is because the port F is directly connected to signals D7-D0 (see table 3).
2. write value 00000100 (or in HEX: \$04) in port E. This is because according to table 3, bits 0 1 and 2 of this port are connected to the LCD signals RS R/W* and E respectively. We want to make signal E ‘1’.
3. write value 00000000 (or in HEX: \$00) in port E. This is because we want to cause a TRANSITION from HIGH to LOW on signal E of the LCD which is connected to bit 2 of port E

Important notes:

1. You must turn the LCD display on AFTER the CMM332 board has been powered up and reset. This is because during power on the board performs a self diagnostic which will fail if the LCD (which is connected to the pins of the microcontroller) is turned on.
2. If you wish to run your program using the GO command of the debugger, you would need to slow down the rate at which the commands are given to the LCD display. This is because the microcontroller is too fast for the LCD driver...

You can do this by inserting a *delay loop* after each command send to the LCD. A delay loop is nothing more than subroutine in which a register is decremented from \$EFFF to 0.

Also if you want to use the GO command to run your program you must end it with the appropriate TRAP command to return control to the debugger. End your program with the following two instructions:

```
TRAP #15
DC.W $63
```

3. The LCD will be disabled once you power it on. To initialize it properly send a command to clear it. Note that before you writing any data to the LCD module it is necessary to first send the “DISPLAY ON” command:

```

*      prepare to write to the LCD display

      MOVE.W #$0004,PEDR * bit 3  2  1  0
*                        0  1  0  0
                        E  R/W RS
      MOVE.W #$000F,PFDR * send the DISPLAY ON command to the LCD

*
*      RS  RW   D7  D6  D5  D4  D3  D2  D1  D0
*  Display control  0   0   0   0   0   0   1   D   C   B
*
*      D  =  1  = display ON
*      D  =  0  = display OFF
*      -----
*      C  =  1  = cursor ON
*      C  =  0  = cursor OFF
*      -----
*      B  =  1  = cursor BLINKING
*      B  =  0  = cursor NOT BLINKING

```

Once the LCD module has been initialized with the DISPLAY ON command, ASCII data may be written it using the following sequence of instructions:

```

*  WRITE ASCII DATA TO PORT F and STROBE using E=0
*  and DELAY in between strobes
*  WRITE "T" ASCII 54H
      MOVE.W #$0005,PEDR * bit 3  2  1  0
*                        0  1  0  1
*                        E  R/W  RS   note RS = 1 for
*                        writing data into LCD
      MOVE.W #$0054,PFDR * send "T" to the LCD
      BSR DELAY

```

```

MOVE.W #$0001,PEDR      * make E= 0  (info strobed to Display)
*
*           bit 3  2  1  0
*           0  0  0  1
*
*           E  R/W  RS   note RS = 1 for
*
*           writing data into LCD
BSR DELAY

```

PRELAB

The following is to be performed prior to attending the scheduled lab session:

- Read the contents of this lab in its entirety.
- Read and understand the LCD datasheet given in Appendix 5
- Write the required MC68000 programs.

Part 1: Requirement

Write an MC68000 program which will turn on and off a light emitting diode. The LED is to be connected to bit 0 of Port F and ground. Refer to Figure 1 for the details of connecting the LED to the CMM 332 board. The LED is to be ON for approximately 0.5 seconds and OFF for 0.5 seconds. Use a delay subroutine to control these ON and OFF times. The main program is to initialize the ports and then repeatedly turn ON and OFF bit 0 of Port F. The printed circuit board containing the main CMM 332 board and the LCD display contains a 14 pin header which provides access to all 8 bits of Port F and the three low-order bits of Port E. It is necessary to turn on the power switch which provides power to the LCD module in order to be able to access the pins of the header. The pseudocode is as follows:

```

    configure Port F as parallel output
loop: write a logic '1' to port F, bit 0
      delay for 0.5 seconds
      write a logic '0' to port F, bit 0
      delay for 0.5 seconds
      branch to loop

```

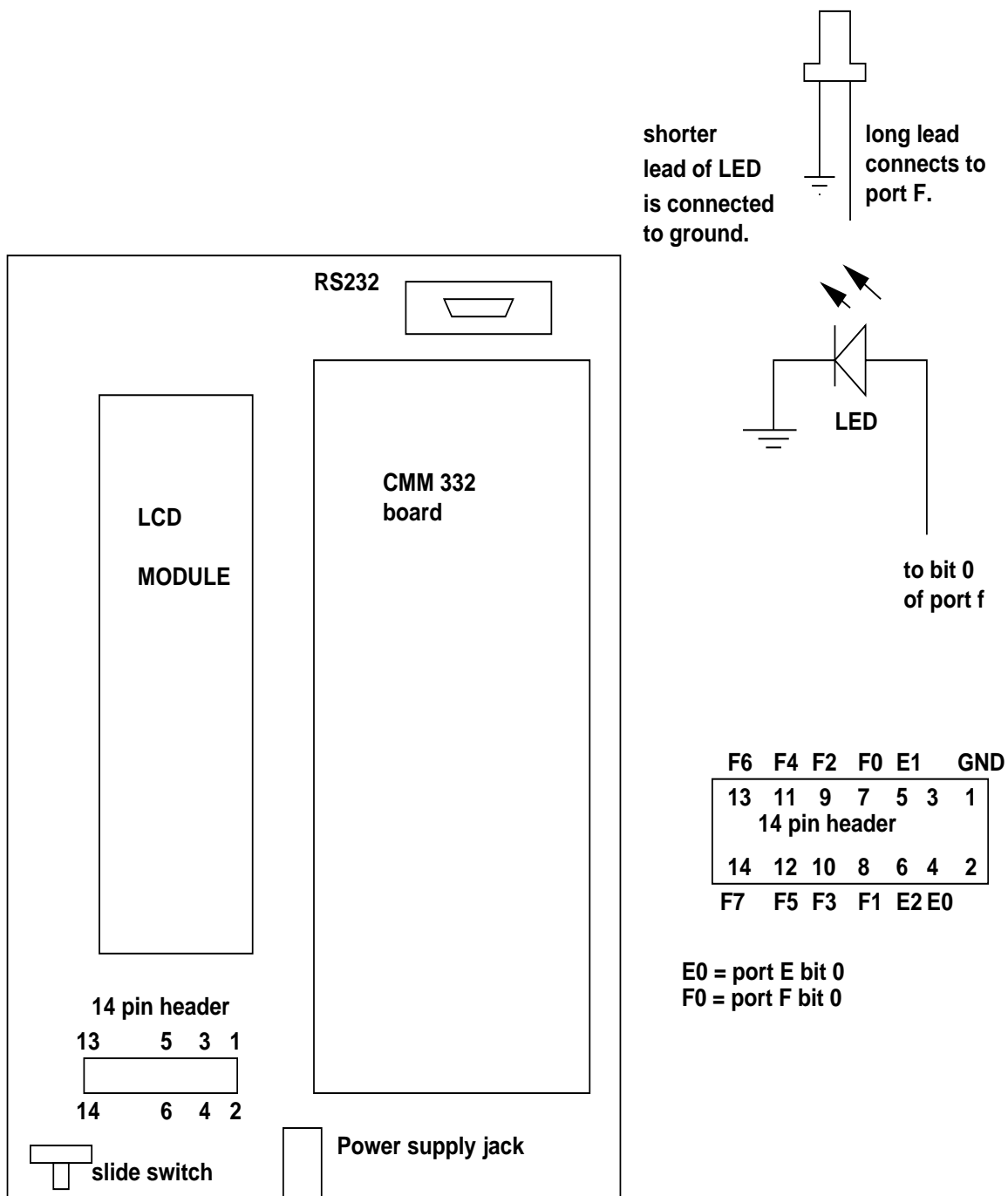


Figure 1: Port header connections.

Part 2: Requirement

Write an MC68000 program that will initialize the LCD and display your name. Your program is to make use of a delay subroutine as well as a subroutine which displays an ASCII character stored in register D0 on the LCD. Store the characters composing your

name in a buffer in main memory. This can be easily done with the DC.B (Declare Constant) assembler directive as in:

```
ORG      $4000
BUF      DC.B  'j','A','c','K',' ','f','L','a','S','h','#','l'
```

Use a loop structure to move the characters from the memory buffer into register D0.

Once the character is in register D0, make a call to your subroutine which handles the writing of the character to the LCD display.

APPENDIX 1

Contains:

- Assembly programs for all experiments

*ASSEMBLY PROGRAM FOR EXPERIMENT #2

```
ORG $4000
CLR      D1
CLR      D2
CLR      D3
MOVE.W   #$200A,D2
MOVE.W   #$1000,D3
ADD      D2,D1
ADD      D3,D1
TRAP     #15
DC.W     $63
```

MODIFICATIONS TO BE PERFORMED TO THIS PROGRAM BY DIRECTLY
CHANGING THE MEMORY:

1. Make the program load value 15ff instead of 200A in register D2
2. Make the program load value 00FF instead of 1000 in register D3
3. Make the program subtract value D3 from D1 (instead of adding it)

*ASSEMBLY PROGRAM FOR EXPERIMENT #3

```
ORG $4000
MOVE.W    #$4,D0
MOVE.W    #$5,D1
CLR       D2
SUB       #1,D0
ADD       D1,D2
ADD       D0,D2
TRAP #15
DC.W      $63
```

*OP-CODE FOR THE TRAP INSTRUCTION IS 4E4X, WHERE X IS THE VECTOR NUMBER IN HEX.

*IF X IS #15 THEN THE TRAP INSTRUCTION WILL EXECUTE THE USER ROUTINE
*SPECIFIED IN THE FOLLOWING DC.W INSTRUCTION.

*THE DC.W \$63 SPECIFIES THAT CONTROL OF THE EXECUTION IS RETURNED TO THE
*CPU32 DEBUGGER.

*ASSEMBLY PROGRAM FOR EXPERIMENT #4

*THIS PROGRAM USES THREE NUMBERS CONTAINED IN
*REGISTERS D0, D1 AND D2 TO PERFORM CALCULATIONS
*AND STORES THE RESULT IN REGISTER D3.
*THE REGISTERS D0, D1, D2 WILL CONTAIN THEIR
*INITIAL CONTENT AFTER THE EXECUTION OF THIS PROGRAM.

CLR.L	D3
ADD.L	D0,D3
ADD.L	D1,D3
SUB.L	D2,D3

APPENDIX 2

Contains:

- Selected Debugger commands for the CPU32BUG debugger. The complete documentation of the debugger provided as supplemental lab material contains the complete command documentation

APPENDIX 3

Contains:

- The Motorola S-Record format documentation.

APPENDIX 4

Contains:

- Selected Motorola 68000 instructions. These pages were taken from the MC68000 programmer's reference documentation which is available in the additional lab materials set of documents. Chapter 4 of that documents contains all the MC68000 instructions.

APPENDIX 5

Contains

- LCD datasheet for the HD44780 commands
- ASCII character table for the LCD characters
- Excerpt from the 68332 documentation on the parallel port control register descriptions. The complete 68332 documentation is available in the additional lab materials set of documents.

APPENDIX 6

A sample MC68000 assembly language program.

```
* Program 1 - Find the maximum value of an array of number
* T. Obuchowicz
* October 29, 2008
```

```
        ORG $3000
RSLT    DS.B 1          * reserve 1 byte of storage to hold the result
ARRAY   DC.B 2,5,1,4,3 * declare byte sized constants

        ORG $5000
CLR.L   D0
CLR.L   D1
MOVE.L  #4,D0          * setup D0 as a counter
MOVEA.L #ARRAY,A0
MOVE.B  (A0),D1        * get first element of array
LOOP    CMP.B (A0)+,D1
        BGT BIGGER
        MOVE.B -1(A0),D1
BIGGER  DBRA D0,LOOP
        MOVE.B D1,RSLT
        END
```

NOTE: If you intend the run the program using the GO command, add the two lines

```
TRAP #15
DC.W $63
```

as the last two instructions before the END assembler directive.

APPENDIX 7

A MC68000 assembly language program listing containing assembler directives to define memory constants, a subroutine definition, and a main program which calls the subroutine.

```
* Program 1 - Adds two numbers using a subroutine
* makes use of parameter passing using CPU registers
* T. Obuchowicz
* August 29, 2010

* define a subroutine which adds two numbers
* the addend and augend are expected to be in registers D0 and D1
* the sum is returned via register D1
00003000                                ORG $3000
00003000 d240                          MYSUB  ADD D0,D1
00003002 4e75                          RTS

* define some memory locations which hold the
* numbers to be added and place to store the
* sum
00004000                                ORG $4000
00004000 0001                          MICK   DC.W 1
00004002 0002                          KEITH  DC.W 2
00004004 0000                          RON    DC.W ?

* The main program
00005000                                ORG $5000
00005000 3038 4000                      MOVE  MICK,D0
00005004 3238 4002                      MOVE  KEITH,D1
```


00005008 6100 dff6

BSR MYSUB

0000500c 31c1 4004

MOVE D1,RON

END

==== 0 Error(s)

==== 0 Warning(s)

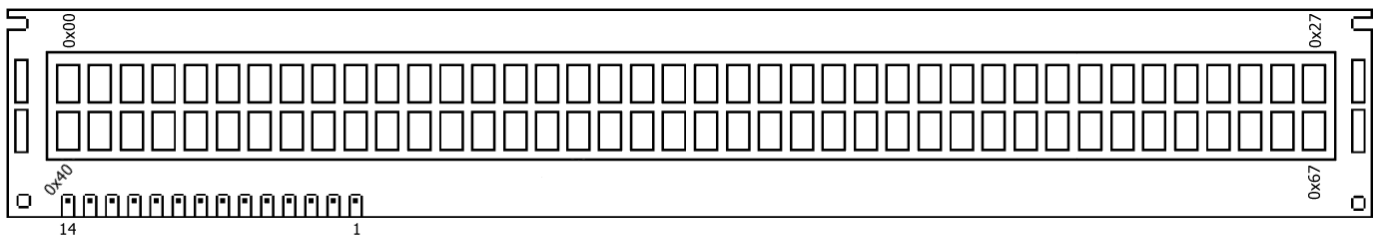
The *Extended Concise* LCD Data Sheet

for HD44780

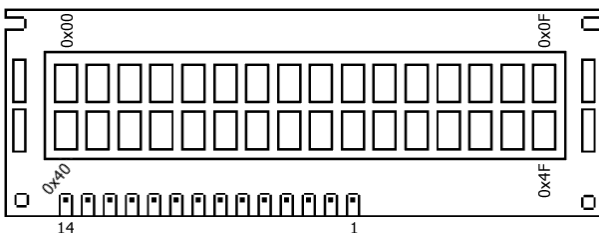
Version: 25.6.1999

Instruction	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Description	Clock-Cycles
NOP	0	0	0	0	0	0	0	0	0	0	No Operation	0
Clear Display	0	0	0	0	0	0	0	0	0	1	Clear display & set address counter to zero	165
Cursor Home	0	0	0	0	0	0	0	0	1	x	Set adress counter to zero, return shifted display to original position. DD RAM contents remains unchanged.	3
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Set cursor move direction (I/D) and specify automatic display shift (S).	3
Display Control	0	0	0	0	0	0	1	D	C	B	Turn display (D), cursor on/off (C), and cursor blinking (B).	3
Cursor / Display shift	0	0	0	0	0	1	S/C	R/L	x	x	Shift display or move cursor (S/C) and specify direction (R/L).	3
Function Set	0	0	0	0	1	DL	N	F	x	x	Set interface data width (DL), number of display lines (N) and character font (F).	3
Set CGRAM Address	0	0	0	1	CGRAM Address					Set CGRAM address. CGRAM data is sent afterwards.		3
Set DDRAM Address	0	0	1	DDRAM Address					Set DDRAM address. DDRAM data is sent afterwards.		3	
Busy Flag & Address	0	1	BF	Address Counter					Read busy flag (BF) and address counter		0	
Write Data	1	0	Data					Write data into DDRAM or CGRAM		3		
Read Data	1	1	Data					Read data from DDRAM or CGRAM		3		
x : Don't care	I/D	1 0	Increment Decrement					R/L	1 0	Shift to the right Shift to the left		
	S	1 0	Automatic display shift					DL	1 0	8 bit interface 4 bit interface		
	D	1 0	Display ON Display OFF					N	1 0	2 lines 1 line		
	C	1 0	Cursor ON Cursor OFF					F	1 0	5x10 dots 5x7 dots		
	B	1 0	Cursor blinking					DDRAM : Display Data RAM CGRAM : Character Generator RAM				
	S/C	1 0	Display shift Cursor move									

LCD Display with 2 lines x 40 characters :



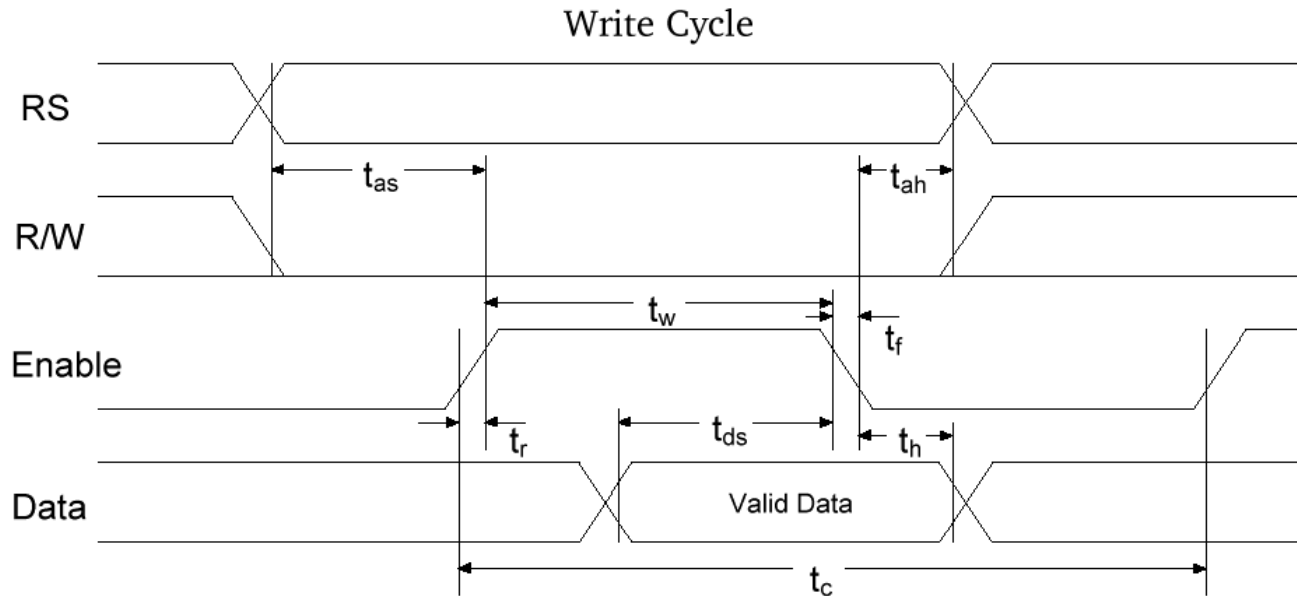
LCD Display with 2 lines x 16 characters :



Pin No	Name	Function	Description
1	Vss	Power	GND
2	Vdd	Power	+ 5 V
3	Vee	Contrast Adj.	(-2) 0 - 5 V
4	RS	Command	Register Select
5	R/W	Command	Read / Write
6	E	Command	Enable (Strobe)
7	D0	I/O	Data LSB
8	D1	I/O	Data
9	D2	I/O	Data
10	D3	I/O	Data
11	D4	I/O	Data
12	D5	I/O	Data
13	D6	I/O	Data
14	D7	I/O	Data MSB

Bus Timing Characteristics

(Ta = - 20 to + 75°C)



Write-Cycle	V _{DD}	2.7 - 4.5 V ⁽²⁾	4.5 - 5.5 V ⁽²⁾		2.7 - 4.5 V ⁽²⁾	4.5 - 5.5 V ⁽²⁾	
Parameter	Symbol	Min ⁽¹⁾		Typ ⁽¹⁾	Max ⁽¹⁾		Unit
Enable Cycle Time	t _c	1000	500	-	-	-	ns
Enable Pulse Width (High)	t _w	450	230	-	-	-	ns
Enable Rise/Fall Time	t _r , t _f	-	-	-	25	20	ns
Address Setup Time	t _{as}	60	40	-	-	-	ns
Address Hold Time	t _{ah}	20	10	-	-	-	ns
Data Setup Time	t _{ds}	195	80	-	-	-	ns
Data Hold Time	t _h	10	10	-	-	-	ns

(1) The above specifications are indications only (based on Hitachi HD44780). Timing will vary from manufacturer to manufacturer.

(2) Power Supply : HD44780 S : V_{DD} = 4.5 - 5.5 V
 HD44780 U : V_{DD} = 2.7 - 5.5 V

This data sheet refers to specifications for the Hitachi HD44780 LCD Driver chip, which is used for most LCD modules.

Common types are :

- 1 line x 20 characters
- 2 lines x 16 characters
- 2 lines x 20 characters
- 2 lines x 40 characters
- 4 lines x 20 characters
- 4 lines x 40 characters