PIC-an-LCD

User’s Guide
Version 0.9d
February 1998

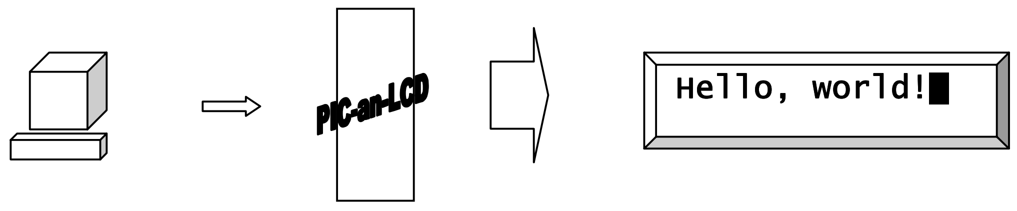
Written by Dale Wheat

Table of Contents

Table of Figures

1 Introduction

The “PIC-an-LCD” is a simple do-hickey that allows common alphanumeric LCD modules to be controlled by a serial interface.



PC* PIC-an-LCD Alphanumeric LCD module

Figure 1. Simple diagram showing where the “PIC-an-LCD” belongs

*A PC or any other device that can generate an asynchronous serial data stream, such as a modem or a microcontroller with a UART.

It also provides five (5) general purpose TTL-level outputs, which can be used for whatever you want, like LEDs, solid-state relays, or whatever.

2 Description

Overview

The “PIC-an-LCD” is a pre-programmed microcontroller, based on the Microchip PIC16C621. Its main purpose in life is to control an alphanumeric LCD (liquid crystal display) module, specifically those that use the Hitachi HD44780 chipset. It also sports a serial data input and five (5) general purpose TTL outputs.

Requirements

The “PIC-an-LCD”, like most other chips, has some simple requirements. First, it requires a regulated DC (direct current) power supply of five (5) volts at about 4-10 milliamperes, which is not very much at all. Second, it requires a quartz crystal to control its internal clock and provide an accurate time-base for the baud rate. Third, it needs an alphnumeric LCD module that uses the Hitachi HD44780 chipset (or equivalent). Finally, it works in a much more interesting manner if there is a source of serial data available, such as a PC, modem, or another microcontroller.

Device Pinouts

The “PIC-an-LCD” lives in an eighteen (18) pin plastic DIP (dual in-line package), whose pins are described by the diagram below.

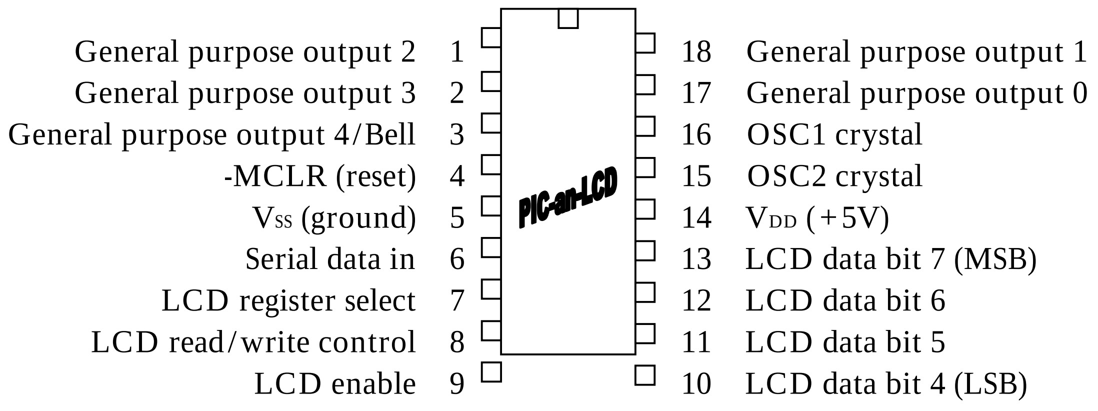


Figure 2. Device pinouts

Baud rate and crystal frequency

The baud rate for the serial interface is determined by the frequency of the quartz crystal used as the chip's oscillator, as shown in the table below.

Baud rate	Crystal frequency
9600	14.318 MHz
4800	7.159 MHz
2400	3.579 MHz
1200	1.790 MHz

Figure 3. Baud rate vs. crystal frequency

Serial data format

The serial data input format is fixed at one (1) start bit, eight (8) data bits, one (1) stop bit and no parity bits. This format is very common, and is sometimes referred to as "8N1".

Serial data signal levels

The input of the "PIC-an-LCD" is compatible with a wide range of input levels. It will work directly with TTL (transistor-transistor logic) levels, or RS-232C levels through a current-limiting resistor or level-shifting receiver, such as the SN74189 or MAX232.

The polarity of the signal is determined automatically at device power-up and reset. The incoming signal is assumed to be in the "make" state (as opposed to the "break" state) for a short period of time after power is applied to the device.

Further information

Since the "PIC-an-LCD" is derived from the PIC16C621 programmable microcontroller, all device specifications will ultimately resolve to that device's specifications. Microchip publishes these specifications, and much more information, on their web site. Next time you're on the Internet, surf on by http://www.microchip.com and check them out.

Microchip also has sales offices around the world, so you shouldn't have much of a problem getting their device specifications.

3 Installation and assembly

Note: This section assumes that you're going to wire up the "PIC-an-LCD" yourself, as opposed to sticking it in a prefabricated circuit board. If you are so lucky as to have a PC board already made for the "PIC-an-LCD", then you can skip this section.

Here is a step-by-step guide to hooking up the "PIC-an-LCD". It's not the only possible way to hook things up, but it is the fastest and probably the most reliable.

Power and ground

The "PIC-an-LCD" requires a regulated five (5) volt DC power supply. Althought the "PIC-an-LCD" draws very little power, a poorly regulated supply can introduce weird and pesky problems.

It's always a good idea to hook up ground first, and power last. The ground pin (V_{SS}) on the "PIC-an-LCD" is pin 5, and the power pin (V_{DD}) is pin 14. It wouldn't hurt to have at least a couple of 0.1µF (or smaller) decoupling capacitors across the power supply, especially if other circuits share this supply. If you're running the "PIC-an-LCD" from the power supply in your PC, then the decoupling capacitors are not optional. Try to place them as close as possible to the "PIC-an-LCD" chip.

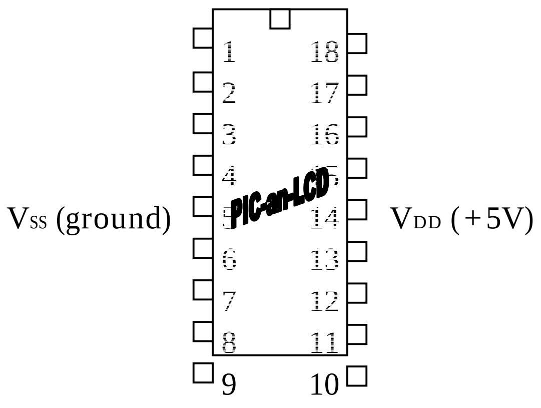


Figure 4. Power and ground pins

I'm not going to tell you how to build a regulated DC power supply here, (sorry - it's just really beyond the scope of this manual), but I will give you a couple of hints. A really simple power supply can be made with a 9V alkaline battery and a small fixed positive voltage regulator, such as the LM78L05. It's not really the most efficient way to go (the regulator draws as much current as the "PIC-an-LCD" and most LCD modules combined), but it will work in a pinch if nothing else is available. Ask your favorite wire-head how to hook it up. Also, four (4) rechargeable NiCd cells (NOT alkalines) in series will give around five (5) volts (while they're charged).

PIC16C621 support circuitry

The "PIC-an-LCD" was designed to require a bare minimum of external components to operate. The quartz crystal that determines the baud rate is the only other required component (with the obvious exception of the LCD module). It is connected between pins 16 (OSC1) and 15 (OSC2) of the "PIC-an-LCD" chip. Microchip suggests bypassing these two pins with 15pf – 33pf capacitors to ground, but they are not strictly required.

Pin 4 is the Master Clear (-MCLR), or reset pin. The "PIC-an-LCD" automatically resets itself on power-up, so no external circuit is required. You may simply tie pin 4 to Vcc, either directly or though a resistor. A resistor is recommended if you want to temporarily short the –MCLR pin to ground to affect a chip reset for some reason. Otherwise, you'd be shorting out your power supply, which can cause other problems (smoke, fire, burns, cussing, etc.).

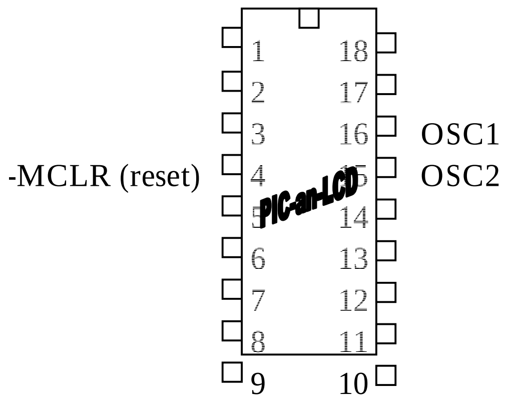


Figure 5. Support circuitry

LCD power and contrast

Most common alphanumeric LCD modules require *two* power supplies: one for the logic (usually +5V, just like the "PIC-an-LCD") and another for the display itself, called V_{EE}. V_{EE} is adjusted to vary the contrast, or viewing angle, of the LCD module. Since it draws very little power, a simple voltage-divider circuit can be used to set the display's contrast. This is built by simply hooking one end of a relatively large value potentiometer (10KΩ or more) to +5V, the other end to ground, and, *voilá*, the middle leg is your V_{EE}. If you just can't seem to find a potentiometer laying around handy, try a 330Ω resistor to ground. This works with *some* modules. I really recommend using a potentiometer, though, because you get much better control of the display.

Note: A lot of people get discouraged when using LCD modules because they read in the product specifications that the V_{EE} voltage has to be adjusted somewhere between +5V and –12V, and they have no negative supply readily available. This is only true if the LCD module is built using "extended temperature range fluid" in the display (yes, there really is a liquid in a liquid crystal display). For "normal" temperature range devices, a negative bias is not required, and a single +5V supply will suffice. If you are stuck with an extended temperature range device, try the single-supply voltage-divider anyway; you just might get lucky. If you're not the lucky type, throw in an ICL7660 and a couple of capacitors and, presto-changeo, you have a negative supply. Also, if you're using a MAX232 for your RS-232 interface, pin 6 provides around –10 volts.

Most LCD modules have similar pinouts. A large number of displays use the first three pins for power, as shown in the table below.

Pin	Description
1	V_{SS} (ground)
2	V_{CC} (+5V)
3	V_{EE} (display drive)

Figure 6. LCD power connections

Note: Don't take my word for it. You should consult the data sheet for your module (if available) for the correct pins for power, ground and display bias.

"PIC-an-LCD" to LCD connections

There are seven (7) signal lines going from the "PIC-an-LCD" to the LCD module. They should be connected as described in the following table.

"PIC-an-LCD"		LCD module	
Pin	Description	Pin	Description
7	LCD register select	4	RS
8	LCD read/write control	5	R/W

9	LCD enable	6	E
10	LCD data bit 4 (LSB)	11	DB4
11	LCD data bit 5	12	DB5
12	LCD data bit 6	13	DB6
13	LCD data bit 7 (MSB)	14	DB7

Figure 7. LCD data connections

Serial data in

Pin 6 of the "PIC-an-LCD" is the serial data input. It can be connected to a variety of serial interfaces. TTL-level signals can be connected directly. RS-232C level signals should be connected through a 1MΩ resistor. Remember to connect a common ground between your data source and the "PIC-an-LCD". It won't work without it.

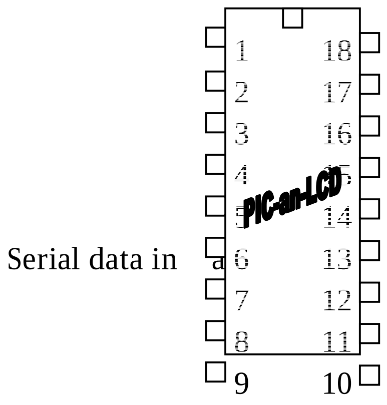


Figure 8. Serial data in

General purpose outputs (optional)

The "PIC-an-LCD" had five (5) extra I/O lines left over when we got through inventing it. So instead of clipping off all those unused pins, we decided to make them useful. They are TTL-level outputs that can be set or cleared by sending the appropriate codes to the "PIC-an-LCD". Their use is not in any way required, and you don't have to hook them up, if you don't want to. If you have a use for them, then by all means, hook em up. They will drive 20mA and sink 25mA of current.

A simple device to attach to the General purpose outputs is an LED (light emitting diode). Simply connect the anode of the LED to the "PIC-an-LCD", connect the cathode of the LED to a current-limiting resistor (330Ω or more) and the other end of the resistor to ground. Now, when you send the appropriate codes to the "PIC-an-LCD", your LED will turn on. A fancier trick is to connect a bi-colored LED between two General purpose outputs (don't forget the current-limiting resistor!) and get two different colors by setting one output on and the other off, and vice versa.

General purpose output 4 (pin 3) also serves as a 'bell' output, although it really sounds more like a 'beep'. If the "PIC-an-LCD" receives the ASCII code for 'bell' (Ctrl-G), it will toggle the pin 3 on and off for one tenth of a second at 1KHz. If you've connected a speaker between the pin and +5V (through at least a 330Ω resistor), then you can hear it. Otherwise, you won't. Likewise, if you want to use

General purpose output 4 for something more general purpose in nature, then go ahead. Just don’t send the ASCII ‘bell’ character.

Also note that General purpose outputs 0-3 are TTL-type outputs, but General purpose output 4/Bell is an open-collector output, and will only sink current.

4 Testing and troubleshooting

Now that you've assembled the "PIC-an-LCD" circuit, you probably want to put it in a drawer and go mow the yard. Or you might want to check it out. Option #2 is explored below.

Pre-test checklist

First, check out your power supply. Make sure it's +5V and NO MORE (well, it can be a *little* more, but not much). Then make sure power goes where power is supposed to go; likewise ground. Nobody gets hurt if the data signals get crossed, but the "PIC-an-LCD" and maybe your LCD module go quietly and quickly 'up in smoke' if the power supply is hooked up backwards. Double-check your wiring!

Initial power up test

You don't need a serial data source for the first test. Just hook up the power supply to the "PIC-an-LCD" + LCD module circuit, cross your fingers, and turn it on.

Here's what's *supposed* to happen when the circuit is powered up: The LCD is initialized and the current revision level is displayed for a second. The speaker, if attached, will beep.

Otherwise, when looking at the LCD module, you were probably greeted by a big, fat nothing. It's OK. Quit crying. Try adjusting the contrast knob (remember the V_{EE} potentiometer that controls the LCD module's viewing angle? You did use a potentiometer, didn't you?) Nine times out of ten the contrast is not set properly. Try adjusting it until you see a blinking block cursor in the upper left corner of your LCD module. When you see this, you are entitled to jump up and down and do a little victory dance. Proceed to the next test.

If, on the othe hand, all you get is a row of blocks, or no display at all, then something isn't quite right. If you get the 'row of blocks', then at least you know your LCD module is working and has its power hooked up right. It's just not getting initialized by the "PIC-an-LCD" (the first thing it does is initialize the LCD module, clear the screen and set the cursor to blinking). This is probably caused by crossed data lines from the "PIC-an-LCD". Double-check you wiring, or try a different LCD module, if you have one.

Even with no "PIC-an-LCD" circuit attached to it, you should be able to get the 'row of blocks' display on an LCD module, if the power supply is hooked up right and the appropriate bias is applied for the contrast adjustment.

Data transmission test

Now that you have a blinking cursor visible in the upper left corner of the display, turn everything off and connect your data source to the “PIC-an-LCD”. Double check that you used a series resistor (1MΩ) if you are connecting directly to an RS-232C source. And make sure that you didn’t forget to hook up the ground wire to your data source.

Now turn the “PIC-an-LCD” back on. You should see the cursor blinking right away. Now start sending data to the “PIC-an-LCD”. If your baud rate and data format (eight (8) data bits, one (1) stop bit, and no parity) are correct on the sending side, you should be rewarded with the very impressive sight of your text being displayed on your LCD module. Your work here is done.

Or other things might happen. If the LCD module fills with garbage when you send data, you should double-check your sending baud rate and data format. If you get no response at all when you send data, check your cable and any adapters that might be in-line with it. Make sure you’re going out the right port, if you have more than one.

5 Sending data and commands

The “PIC-an-LCD” responds to the data sent to it in one of two ways; either as ‘printable’ characters, or as commands. ‘Printable’ characters are the letters, numbers and punctuation that are displayed on the LCD. Commands are the codes that tell the LCD what to do, like clear the screen, move to the beginning of the line, or change the appearance of the cursor. In this way it works very much like a video terminal or teletype.

‘Printable’ characters

The ‘printable’ characters are those whose ASCII codes (see Appendix) are in the range 0x20 to 0x7f (mostly Latin characters) or 0xa0 to 0xff (mostly Japanese characters). Your LCD’s data sheet should contain a chart called a ‘Font Table’ or ‘Character Generator Table’ that shows all the possible characters that can be displayed by your LCD.

When the “PIC-an-LCD” receives a ‘printable’ character, it sends it directly to the LCD module, which places it at the cursor’s current position. Normally, the cursor is then advanced to the next position, but this behavior can be changed.

Commands

The invisible (or ‘unprintable’) characters are interpreted by the “PIC-an-LCD” as commands. The commands tell the “PIC-an-LCD” what to do, and it in turn tells the LCD module what to do. For example, most LCD modules do not understand the concepts of ‘carriage return’ and ‘line feed’. Since LCD modules are solid state devices with no moving parts, and are noticeably lacking in both a carriage and a platen, this is not such a grievous omission. The “PIC-an-LCD” knows what both the ‘carriage return’ and ‘line feed’ commands are expected to do on a mechanical terminal, and emulates their action on the LCD module.

The commands that the “PIC-an-LCD” understands are listed in detail in the Appendix. Some of the basic ones are described in this section.

The ‘carriage return’ (Ctrl-M, hex 0x0D, decimal 13) command tells the “PIC-an-LCD” to move the LCD module’s cursor to the beginning, or left end, of the line. It does not automatically move the cursor to the next line. That’s what the ‘line feed’ (Ctrl-J, hex 0x0A, decimal 10) command does.

The ‘backspace’ (Ctrl-H, hex 0x08, decimal 8) command moves the cursor back one space and erases the character in that spot. This is known as a ‘destructive backspace’.

It does not move the remaining right-hand characters back. If the cursor was already at the beginning of a line, it moves to the end, or right end, of the previous line.

The ‘form feed’ (Ctrl-L, hex 0x0C, decimal 12) command clears the display of the LCD module and moves the cursor to the upper left, or ‘home’ position.

The ‘bell’ (Ctrl-G, hex 0x07, decimal 7) command beeps the speaker, if one is attached. It has no effect on the LCD module.

6 Special Features

Multiline LCD modules

The “PIC-an-LCD” supports LCD modules that have 1, 2 or 4 lines of characters. All LCD modules that use the Hitachi HD44780 chipset support eighty (80) characters of display RAM, each of which may correspond to a display position. These RAM locations are always addressed as 0-39 and 64-103. The “PIC-an-LCD” handles the addressing calculations needed for most common LCD modules, including the following formats:

4 lines x up to 20 characters (default)
2 lines x up to 40 characters
1 line x up to 40 characters

The ‘carriage return’ and ‘line feed’ commands need to operate differently depending on what the configuration of the LCD module is. The ‘Set number of lines’ command (Ctrl-S, hex 0x13, decimal 19) is followed by the number of lines that the LCD module has. For example, if you have a two (2) line module, then send a Ctrl-S followed by the digit ‘2’ (hex 0x32, decimal 50). This allows the “PIC-an-LCD” to know how to handle carriage returns and line feeds. The “PIC-an-LCD” is originally configured to support the addressing calculations of a four (4) line module. If you are using a four (4) line module, then no extra commands are required for proper operation.

LCD commands

LCD modules based on the Hitachi HD44780 chipset offer a wide variety of operating modes and configurations. The “PIC-an-LCD” allows access to these features by using two special commands: ‘Send LCD Instruction’ (Ctrl-Q, hex 0x11, decimal 17) and ‘Send LCD Data’ (Ctrl-R, hex 0x12, decimal 18). Each of these commands would be followed by instructions and data, respectively. For example, the LCD instruction to clear the display is 01 (Ctrl-A, hex 0x01, decimal 1). To send it to the LCD module’s instruction register, you would first send the ‘Send LCD Instruction’ command (Ctrl-Q, hex 0x11, decimal 17) to the “PIC-an-LCD”, then send the instruction byte, 0x01 (Ctrl-A, hex 0x01, decimal 1). The “PIC-an-LCD” will automatically forward this LCD instruction to the correct place in the LCD module, thus executing the instruction and clearing the display. Of course, the ‘form feed’ command (Ctrl-L, hex 0x0C, decimal 12) does just that, and is more intuitive. There are other instructions, however, that are more complex and are not implemented directly by the “PIC-an-LCD”.

A similar LCD instruction is ‘Home Cursor’, 0x02 (Ctrl-B, hex 0x02, decimal 2), which simply moves the cursor to the upper left corner of the display. It does not clear the display or change the appearance of the cursor.

Downloading custom characters

The Hitachi HD44780 LCD controller supports eight (8) downloadable characters. These characters are normally random garbage on power up. Also, any downloaded characters will be lost when the unit loses power, and will need to be programmed again.

Custom characters can be sent to the LCD module via the “PIC-an-LCD” chip by using the ‘Send LCD instruction’ (Ctrl-Q) and ‘Send LCD data’ (Ctrl-R) commands.

Appendix

Control codes

Below is a list of control codes that the “PIC-an-LCD” understands.

Cursor address (Ctrl-A, hex 0x01, decimal 1)
The first forty (40) RAM locations of the LCD module constitute the first line of a two (2) line display, and the first and third lines of a four (4) line display. These are addressed as locations 0-39. The other locations (the lower line of a two (2) line display or the second and fourth lines of a four (4) line display) are addressed as 64-103. The cursor can be placed at any desired location by using the ‘Cursor address’ (Ctrl-A) command. First the command is sent, then the binary address for the cursor (0-39 or 64-103). Note that the binary address is a single byte, such as 0x00 for location zero (0), not the ASCII digits for the address. Subsequent output will be placed at the cursor’s location.

Home cursor (Ctrl-B, hex 0x02, decimal 2)
The cursor is returned to the upper, left corner of the display. The contents of the display are not affected.

Cursor left (Ctrl-C, hex 0x03, decimal 3)
The cursor is moved one space to the left. The character under the cursor is not affected. This is known as a ‘non-destructive backspace’.

Cursor right (Ctrl-D, hex 0x04, decimal 4)
The cursor is moved one space to the right. The character under the cursor is not affected.

Save cursor position (Ctrl-E, hex 0x05, decimal 5)
The current position of the cursor is saved. Any previously saved cursor position is lost.

Restore cursor position (Ctrl-F, hex 0x06, decimal 6)
The previously saved cursor position is restored. This command can be repeated any number of times to allow the cursor to be returned to a known location.

Bell (Ctrl-G, hex 0x07, decimal 7)
The speaker (if attached to General purpose output 4/Bell) is sounded. The display is not affected. The frequency and duration of the tone are determined by the crystal oscillator.

Backspace (Ctrl-H, hex 0x08, decimal 8)
The cursor is moved back (to the left) one space, and the character now under the cursor is replace with a space (ASCII code hex 0x20, decimal 32). This is known as a ‘destructive backspace’.

Horizontal tab (Ctrl-I, hex 0x09, decimal 9)
The cursor is advanced to the next tab stop. Tab stops are defined every four (4) spaces.

Line feed (Ctrl-J, hex 0x0A, decimal 10)
The cursor is advanced to the same location in the next lower line. If the cursor was already on the bottom line (of a multi-line display), then the contents of the display are scrolled upwards one (1) line, and the bottom line is cleared.

Vertical tab (Ctrl-K, hex 0x0B, decimal 11)
The display is scrolled upwards one (1) line, and the bottom line is cleared. The cursor remains in the same location.

Form feed (Ctrl-L, hex 0x0C, decimal 12)
The display is cleared, and the cursor is returned to the upper, left corner.

Carriage return (Ctrl-M, hex 0x0D, decimal 13)
The cursor is returned to the left-most edge of the current line.

Shift left (Ctrl-N, hex 0x0E, decimal 14)
The display is shifted one (1) space to the left.

Shift right (Ctrl-O, hex 0x0F, decimal 15)
The display is shifted one (1) space to the right.

Send LCD instruction (Ctrl-Q, hex 0x11, decimal 17)
The next character sent is forwarded to the LCD module’s instruction register. This allows infinite flexibility in configuring and controlling the LCD module itself.

Send LCD data (Ctrl-R, hex 0x12, decimal 18)
The next character sent is forwarded to the LCD module’s data register. This allows for any data at all to be sent directly to the LCD module, including codes that would otherwise be interpreted as control codes.

Set number of lines (Ctrl-S, hex 0x13, decimal 19)

The lower three (3) bits of the next character sent is used to determine the number of lines of the LCD module. The number of lines determines many calculations that are made in cursor positioning, such as line feeds and carriage returns. The valid values are zero (0) for no formatting, one (1) for single line displays, two (2) for two line displays and four (4) for four line displays. Because only the lower three (3) bits of this parameter are used, either binary data, such as 0x01 for a single line display, or ASCII data, such as a '4' (hex 0x34, decimal 52) can be sent.

General purpose output (Ctrl-U, hex 0x15, decimal 21)
The lower five (5) bits of the next character sent is used to set the General purpose output pins. All five (5) outputs are updated simultaneously. The least significant bit (LSB) of the parameter sent corresponds to General purpose output 0. Note that General purpose output 4/Bell is an open-collector output, and is affected by the 'Bell' (Ctrl-G) command.

Print signed decimal number (Ctrl-V, hex 0x16, decimal 22)
The next character sent is interpreted as a signed, eight (8) bit number, in the range of –128 to 127, and 'printed' at the current cursor location. If the number is negative, it is preceded by a minus sign ('-'). If it is zero (0) or positive, it is preceded with a space.

Print unsigned decimal number (Ctrl-W, hex 0x17, decimal 23)
The next character sent is interpreted as an unsigned, eight (8) bit number, in the range of 0 to 255, and 'printed' at the current cursor location.

Set cursor display options (Ctrl-Y, hex 0x19, decimal 25)
The lower two (2) bits of the next character sent are used to determine the cursor's display attributes. The cursor can be an underline, a blinking block, or disabled.

ASCII codes

The following table describes the ASCII (American Standard Code for Information Interchange) codes as used by the "PIC-an-LCD". Not all of the standard meanings for the codes are used by the "PIC-an-LCD".

ASCII	Hex	Dec	Description
Ctrl-A	0x01	1	Cursor address
Ctrl-B	0x02	2	Home cursor
Ctrl-C	0x03	3	Cursor left
Ctrl-D	0x04	4	Cursor right
Ctrl-E	0x05	5	Save cursor position
Ctrl-F	0x06	6	Restore cursor position

Ctrl-G	0x07	7	Bell
Ctrl-H	0x08	8	Backspace
Ctrl-I	0x09	9	Horizontal tab
Ctrl-J	0x0A	10	Line feed
Ctrl-K	0x0B	11	Vertical tab (scroll screen up one (1) line)
Ctrl-L	0x0C	12	Form feed (clear screen)
Ctrl-M	0x0D	13	Carriage return
Ctrl-N	0x0E	14	Shift display left
Ctrl-O	0x0F	15	Shift display right
Ctrl-Q	0x11	17	Send LCD instruction
Ctrl-R	0x12	18	Send LCD data
Ctrl-S	0x13	19	Set number of lines
Ctrl-U	0x15	21	General purpose output
Ctrl-V	0x16	22	Print signed decimal number
Ctrl-W	0x17	23	Print unsigned decimal number
Ctrl-Y	0x19	25	Set cursor display option

Figure 9. ASCII codes