LCD Technical FAQ

Contents:

3.5) Initialization for 4-bit operation

The module powers up in 8-bit mode. The initial startup instructions are sent in 8-bit mode, with the lower four bits (which are not connected) of each instruction as don't cares.

After the fourth instruction, which switches the module to 4-bit operation, the control bytes are sent on consecutive enable cycles (no delay is required between nybbles). Most significant is sent first, followed immediately by least significant nybble.

1. <POWER ON>

2. <Wait 15ms>

3.
```
(in 8-bit mode)
RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0   0   0   0   1   1  n/c n/c n/c n/c

n/c=not connected
set 8-bit mode (attention!)
```

4. <Wait 4.1ms>

5.
```
(in 8-bit mode)
RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0   0   0   0   1   1  n/c n/c n/c n/c

set 8-bit mode (attention!)
```

6. <Wait 100us>

7.
```
(in 8-bit mode)
RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0   0   0   0   1   1  n/c n/c n/c n/c

set 8-bit mode (attention!)
```

8. <Wait 4.1ms>

9.
```
(in 8-bit mode)
RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0   0   0   0   1   0  n/c n/c n/c n/c

set 4-bit operation
```

10. <Wait 40us>

```
====================
<Busy Flag cannot be read before this point>
====================
```

11.
```
(in 4-bit mode)
```

```
RS R/W DB7 DB6 DB5 DB4
0   0   0   0   1   0

RS R/W DB7 DB6 DB5 DB4
0   0   1   F   x   x

4-bit operation
1/16 duty cycle
F=font, 1 for 5x11 dot matrix
        0 for 5x8 dot matrix
x=don't care
```

12. <Wait 40us>

13.
```
RS R/W DB7 DB6 DB5 DB4
0   0   0   0   0   0

RS R/W DB7 DB6 DB5 DB4
0   0   1   0   0   0

Display off, cursor off, blink off
```

14. <Wait 40us>

15.
```
RS R/W DB7 DB6 DB5 DB4
0   0   0   0   0   0

RS R/W DB7 DB6 DB5 DB4
0   0   1   1   C   B

Display on, C=1 for cursor on,
B=1 for blinking cursor
```

16. <Wait 40us>

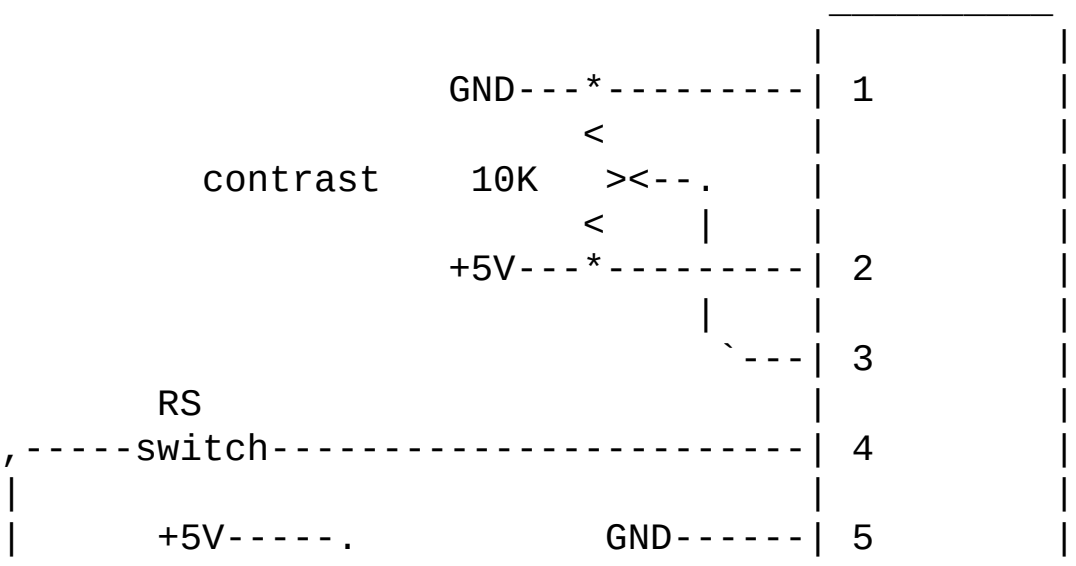
17.
```
RS R/W DB7 DB6 DB5 DB4
0   0   0   0   0   0

RS R/W DB7 DB6 DB5 DB4
0   0   0   1   I   S

Set entry mode,
I=1 for cursor moving right
I=0 for cursor moving left
S=1 for shift display
S=0 for no shift
```

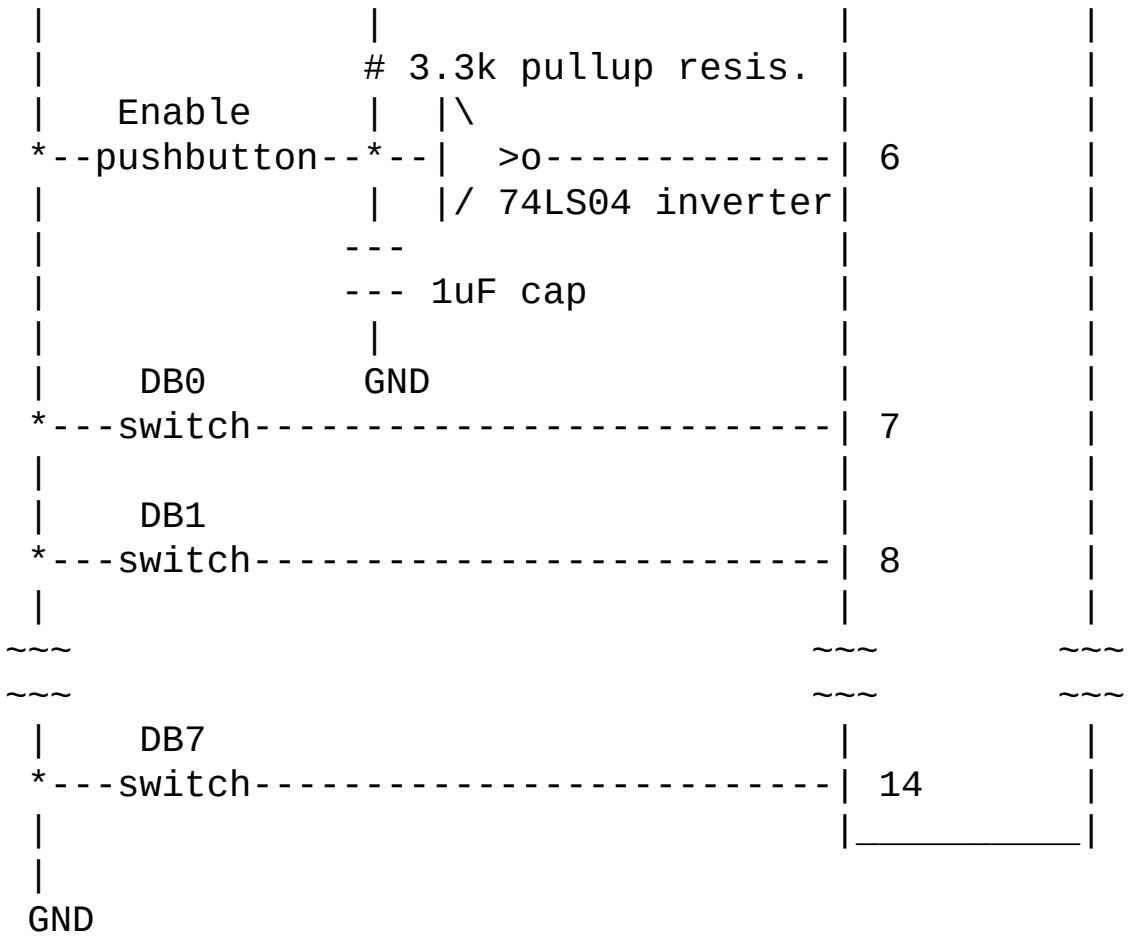
18. <Wait 40us>

19. <INITIALIZATION COMPLETE>

Chapter 4) Interfacing

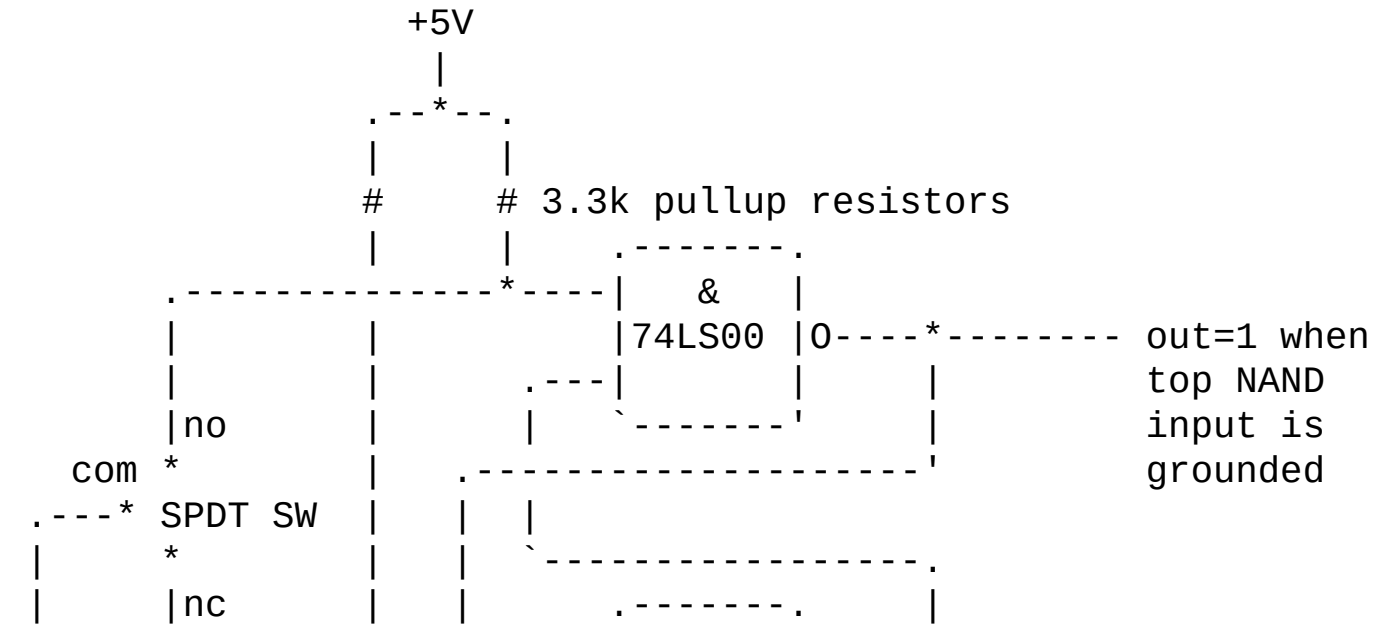
4.1) Manual test circuit

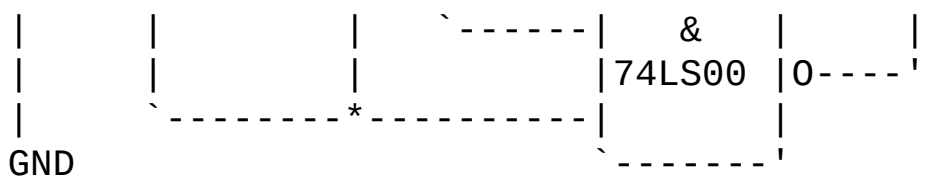




NOTES: Extended temperature range modules need the non-grounded end of the potentiometer tied to -7VDC instead of +5VDC.

I've breadboarded the inverter debounce circuit and it seems to work. However, some experimenters have problems with it. Here is a foolproof debounce circuit:

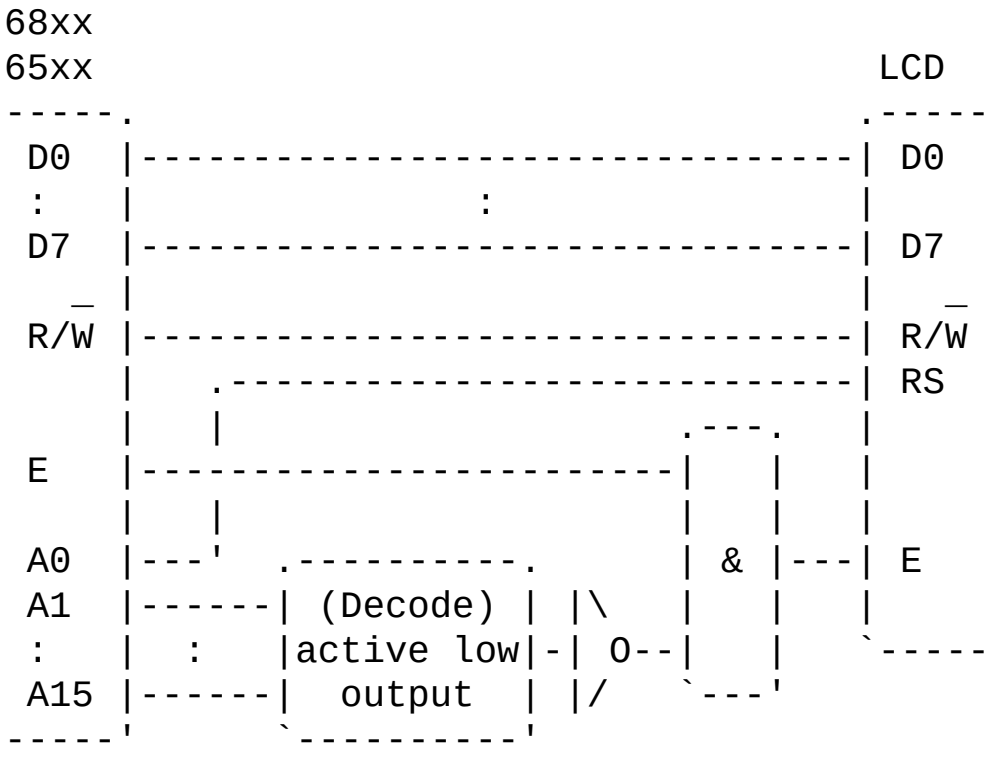




4.2) Interfacing to a CPU Bus

These modules use an interface like those in Motorola (68xx) and MOS Technology (65xx) systems. R/W on the module can come from the R/W line on the CPU, which is set up about the same time as the address, while RS can be selected by the low order address line A0. Select Enable by ANDing the output from your address decoding and the E clock. Address decoding can be done with a magnitude comparator like the 74LS688, or if you have address space to spare, with partial address decode like running A13-A15 into a 74LS138 3-to-8 demux, or just the inverse of the high order address line (hogging a big chunk of address space).

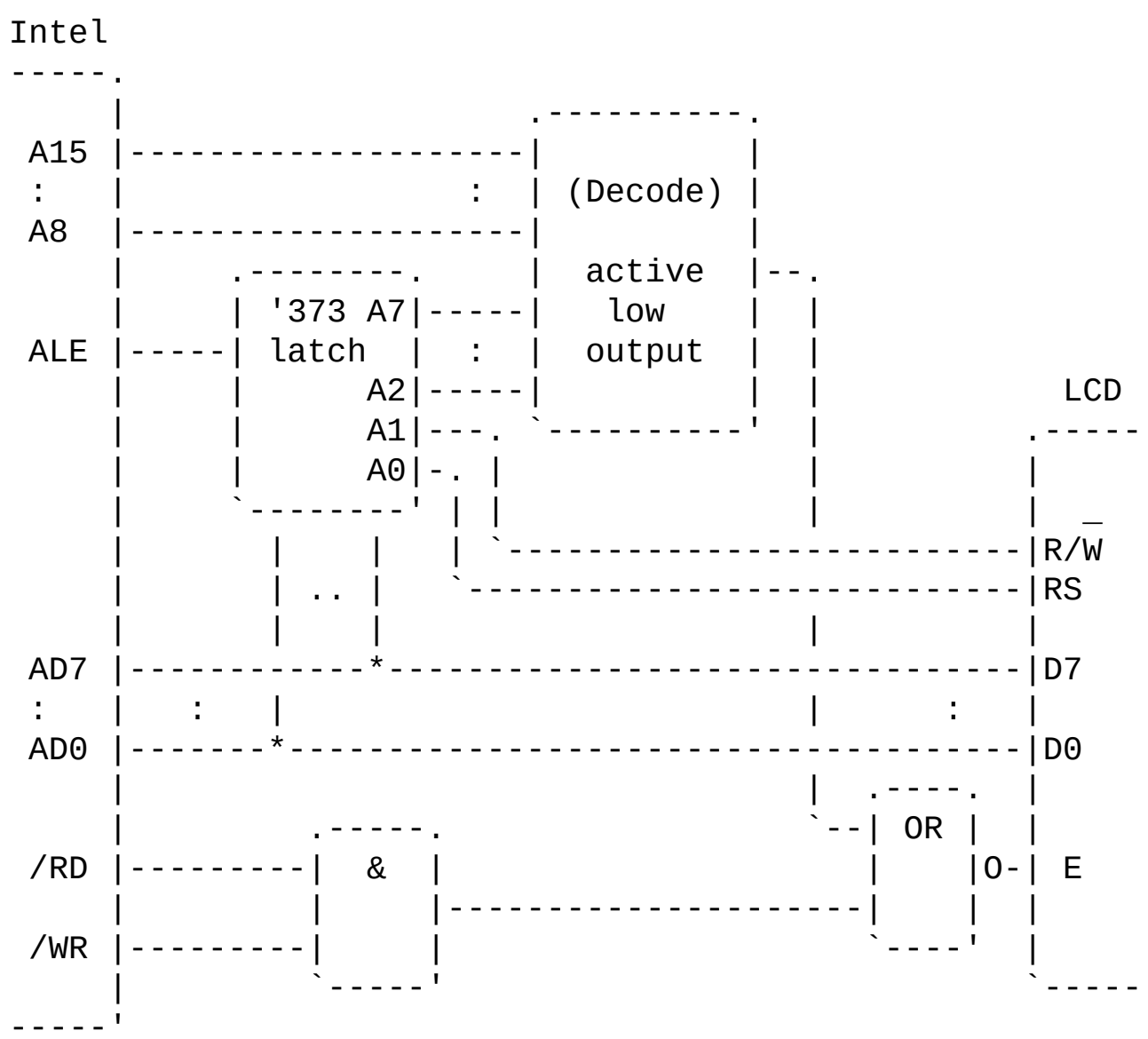
4.2.1) Motorola



Motorola bus interface

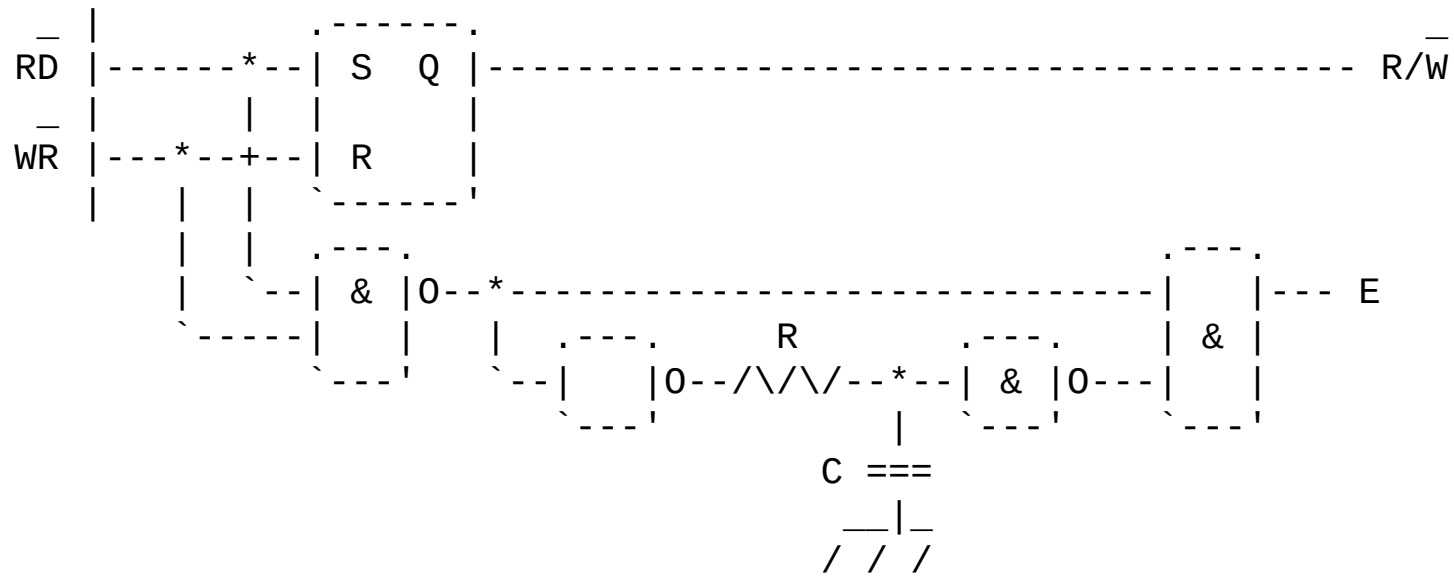
4.2.2) Intel

Because Intel (e.g. 8080, 8085, 8048) and Zilog (e.g. Z80) CPUs use separate read and write lines, and they are not set up ahead of time, R/W as well as RS must be set via address lines. Then the Enable signal may be generated from NOT(/RD AND /WR):



4.2.3) Zilog

Z80



NB. R & C must be chosen to delay the rising edge of the enable pulse at least 140ns.

Using the data bus method limits CPU clock speed because of the tDDR read delay and the TcycE and PWEH requirements of the Hitachi controller.

(Section 5.2 by Doug Girling and Chris Burian)

4.3) Interfacing to a CPU Port

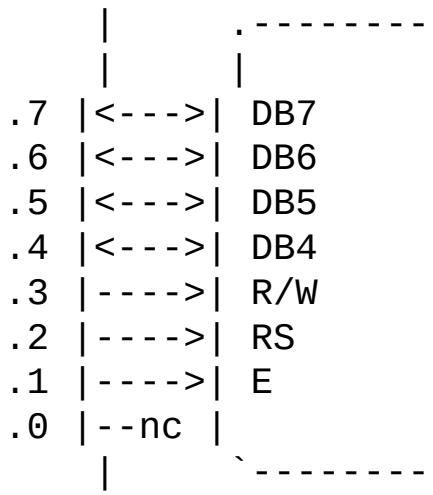
In the case where a direct bus interface is not desired, or is impractical (e.g., many microcontrollers don't provide an external data/address bus, or do so at the expense of reassigning too many pins), the LCD module can be connected via an I/O port.

Whether you use a 4-bit or an 8-bit interface, those port pins driving the LCD's data lines must be bidirectional if you wish to read from the LCD. Remember too that you'll have to write your code to switch the port's data direction bits whenever you switch from reading to writing or visa versa. If you don't expect to ever be reading back from the LCD, you can conserve resources by grounding R/W, saving a pin, and thus using digital outputs instead of bidirectional ports.

You may be able to eliminate the need for a software-generated enable (E) signal if the particular port you are using automatically generates a handshake strobe. In practical terms, most ports only generate a stobe on output, and expect a strobe on input, so this approach is most likely applicable to write-only configurations.

4.3.1) 4-Bit Port Interface

This interface is to drive the module in 4-bit mode using 7 I/O bits from a port.



First, put the most significant nybble on P7-P4, and the appropriate R/W and RS signals on P3 and P2, and P1 low. Then toggle P1 high. Then toggle P1 low again. Then put the least significant nybble on P7-P4, toggle P1 high, toggle P1 low. Forty microseconds later, you can send the next character.

Sample in-line assembly code, by Jordan Nicol, for implementation under Dunfield's Micro-C for the Miniboard can be found on ftp cher.media.mit.edu. It uses port pins PA7-PA3 for 4-bit data and PC6 and 7 for RS and E.

Please check attribution section for Author of this document! This article was written by **filipg@repairfaq.org** [Feedback Form] [mailto]. The most recent version is available on the WWW server http://www.repairfaq.org/filipg/ [Copyright] [Disclaimer]