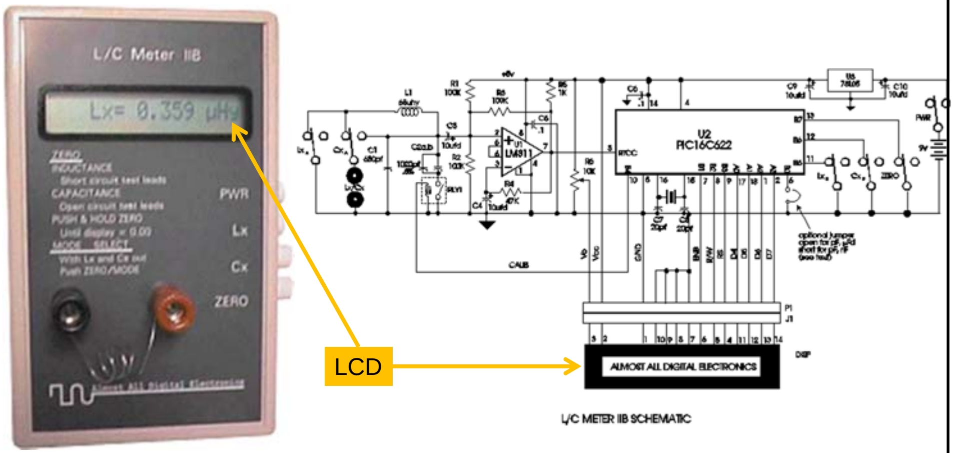
Embedded Systems

Lecture 10 - Liquid-Crystal Displays (LCDs)

ES Example – L/C Meter

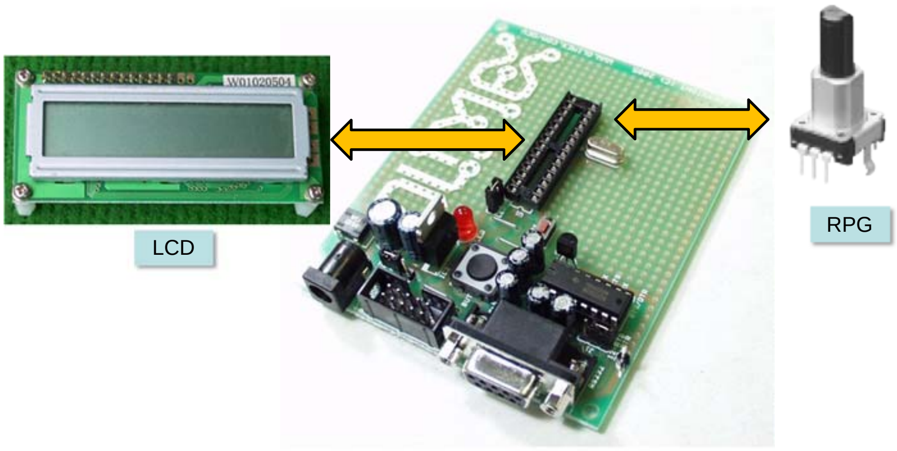


LCD

Start thinking about your project!

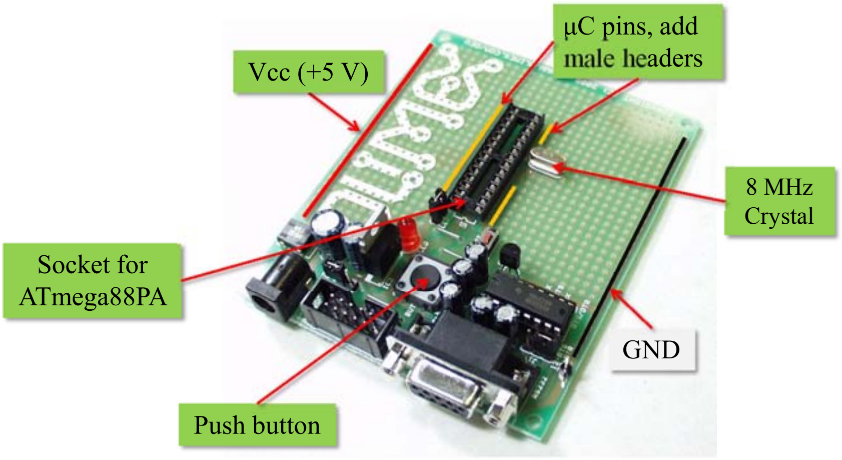
Lab 4 Overview/Preview

Lab 4 will utilize a push button, RPG, and LCD to generate a more elaborated user interface.



28-Pin AVR Development Board

Starting with Lab 4, we will start using a “bigger” controller → get “**Kit B”** from the Electronic Shop



ATmega88 Overview

Will be using ATmega88PA controller, which has more pins, and more resources.

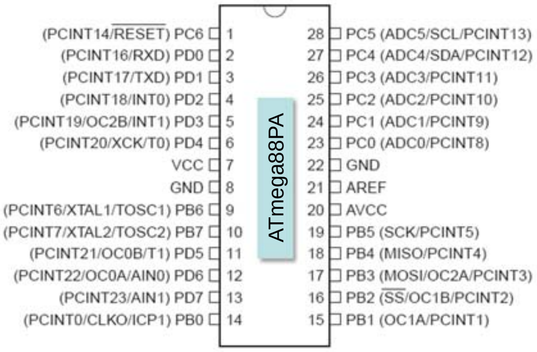
Two 8-bit Timer/Counters with separate Prescaler

$2^8 = 256$ counts

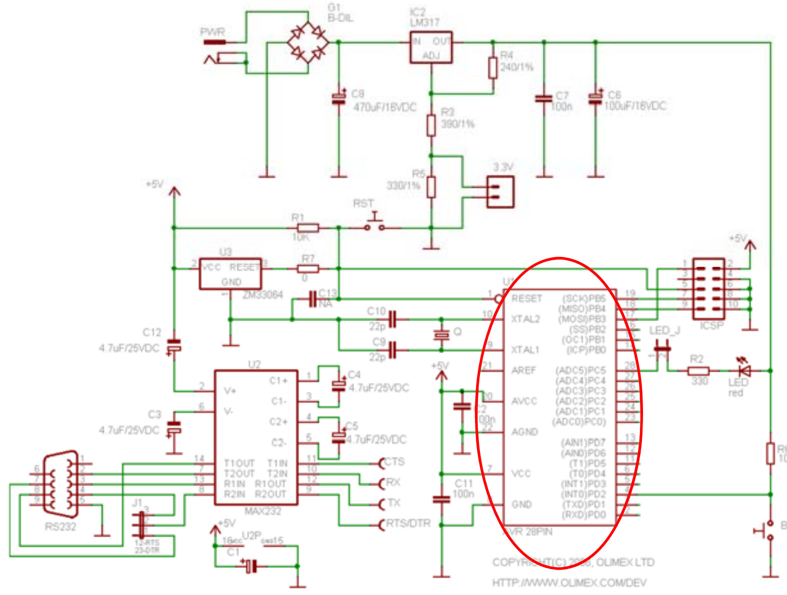
One 16-bit Timer/Counter with separate Prescaler

$2^{16} = 65{,}536$ counts

Will use interrupts



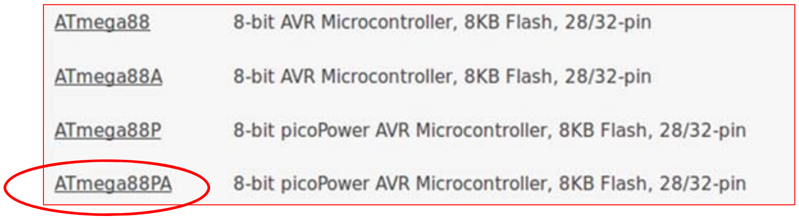
28-Pin AVR Development Schematic



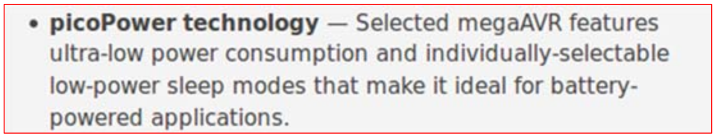
Versions of the ATmega88??

Versions of the ATmega88

- There are several versions of the ATmega88 microcontroller: ATmega88, ATmega88A, ATmega88P, ATmega88PA, etc.
- What are the differences?

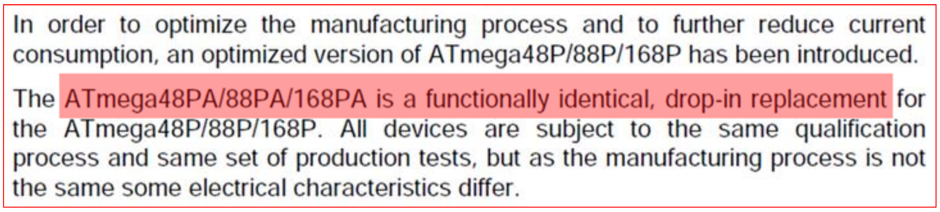


ATmega88	8-bit AVR Microcontroller, 8KB Flash, 28/32-pin
ATmega88A	8-bit AVR Microcontroller, 8KB Flash, 28/32-pin
ATmega88P	8-bit picoPower AVR Microcontroller, 8KB Flash, 28/32-pin
ATmega88PA	8-bit picoPower AVR Microcontroller, 8KB Flash, 28/32-pin



- **picoPower technology** — Selected megaAVR features ultra-low power consumption and individually-selectable low-power sleep modes that make it ideal for battery-powered applications.

ATmega88P vs ATmega88PA



In order to optimize the manufacturing process and to further reduce current consumption, an optimized version of ATmega48P/88P/168P has been introduced.

The ATmega48PA/88PA/168PA is a functionally identical, drop-in replacement for the ATmega48P/88P/168P. All devices are subject to the same qualification process and same set of production tests, but as the manufacturing process is not the same some electrical characteristics differ.

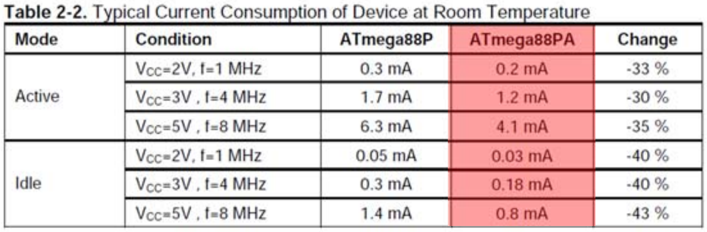
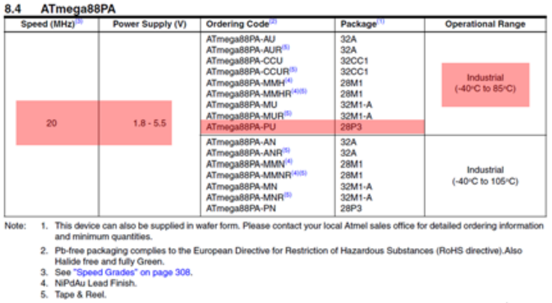


Table 2-2. Typical Current Consumption of Device at Room Temperature

Mode	Condition	ATmega88P	ATmega88PA	Change
Active	V_{CC}=2V, f=1 MHz	0.3 mA	0.2 mA	-33 %
	V_{CC}=3V , f=4 MHz	1.7 mA	1.2 mA	-30 %
	V_{CC}=5V , f=8 MHz	6.3 mA	4.1 mA	-35 %
Idle	V_{CC}=2V, f=1 MHz	0.05 mA	0.03 mA	-40 %
	V_{CC}=3V , f=4 MHz	0.3 mA	0.18 mA	-40 %
	V_{CC}=5V , f=8 MHz	1.4 mA	0.8 mA	-43 %

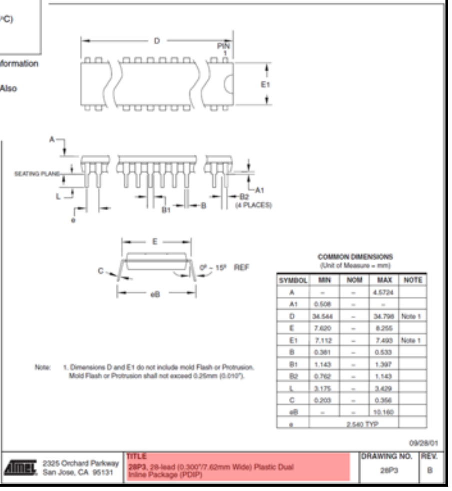
ATmega88PA - Packages



8.4 ATmega88PA

Speed (MHz)[3]	Power Supply (V)	Ordering Code[2]	Package[1]	Operational Range
20	1.8 - 5.5	ATmega88PA-AU ATmega88PA-AUR[5] ATmega88PA-CCU ATmega88PA-CCUR[5] ATmega88PA-MMH[4] ATmega88PA-MMHR[4][5] ATmega88PA-MU ATmega88PA-MUR[5] ATmega88PA-PU	32A 32A 32CC1 32CC1 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)
		ATmega88PA-AN ATmega88PA-ANR[5] ATmega88PA-MMN[4] ATmega88PA-MMNR[4][5] ATmega88PA-MN ATmega88PA-MNR[5] ATmega88PA-PN	32A 32A 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 105°C)

Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive).Also Halide free and fully Green.
3. See "Speed Grades" on page 308.
4. NiPdAu Lead Finish.
5. Tape & Reel.



COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	4.5724	
A1	0.508	–	–	
D	34.544	–	34.798	Note 1
E	7.620	–	8.255	
E1	7.112	–	7.493	Note 1
B	0.381	–	0.533	
B1	1.143	–	1.397	
B2	0.762	–	1.143	
L	3.175	–	3.429	
C	0.203	–	0.356	
eB	–	–	10.160	
e	2.540 TYP			

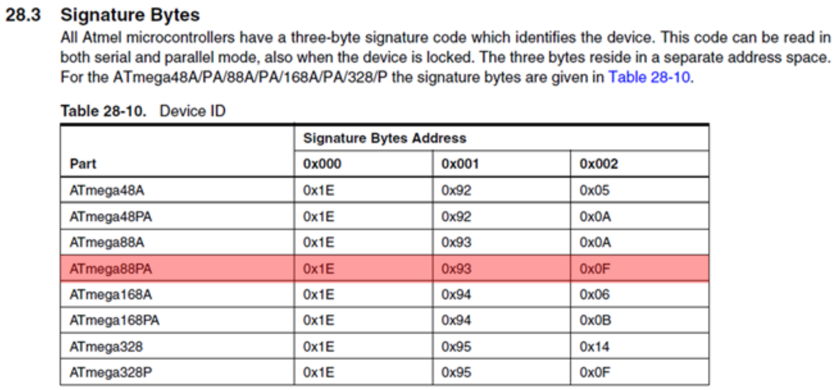
Note: 1. Dimensions D and E1 do not include mold Flash or Protrusion. Mold Flash or Protrusion shall not exceed 0.25mm (0.010").

09/28/01

2325 Orchard Parkway
San Jose, CA 95131

TITLE	DRAWING NO.	REV.
28P3, 28-lead (0.300"/7.62mm Wide) Plastic Dual Inline Package (PDIP)	28P3	B

Signature Bytes



28.3 Signature Bytes

All Atmel microcontrollers have a three-byte signature code which identifies the device. This code can be read in both serial and parallel mode, also when the device is locked. The three bytes reside in a separate address space. For the ATmega48A/PA/88A/PA/168A/PA/328/P the signature bytes are given in Table 28-10.

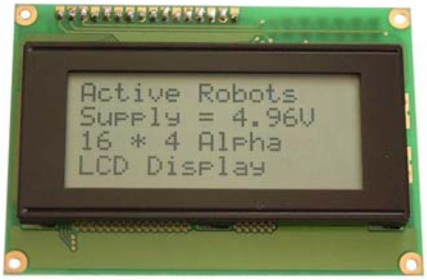
Table 28-10. Device ID

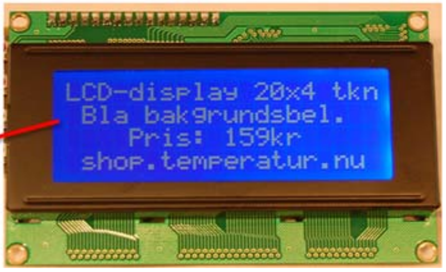
	Signature Bytes Address		
Part	**0x000**	**0x001**	**0x002**
ATmega48A	0x1E	0x92	0x05
ATmega48PA	0x1E	0x92	0x0A
ATmega88A	0x1E	0x93	0x0A
ATmega88PA	0x1E	0x93	0x0F
ATmega168A	0x1E	0x94	0x06
ATmega168PA	0x1E	0x94	0x0B
ATmega328	0x1E	0x95	0x14
ATmega328P	0x1E	0x95	0x0F

Important: select the correct µC version in Atmel Studio for ISP!

Alphanumeric LCDs

Alphanumeric LCDs



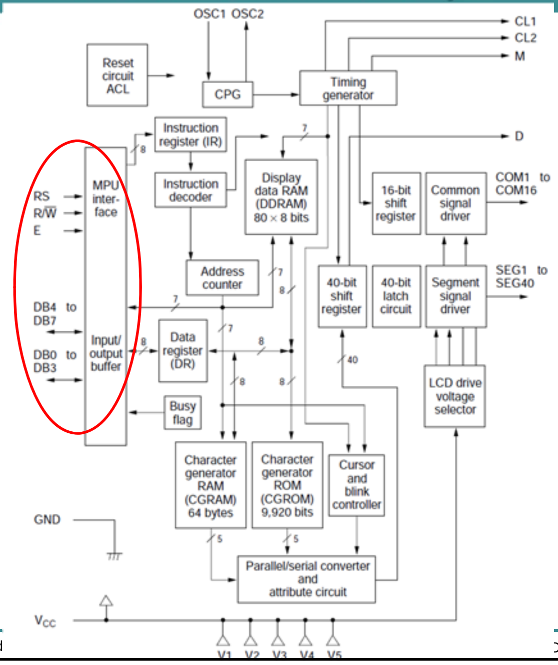


Back light

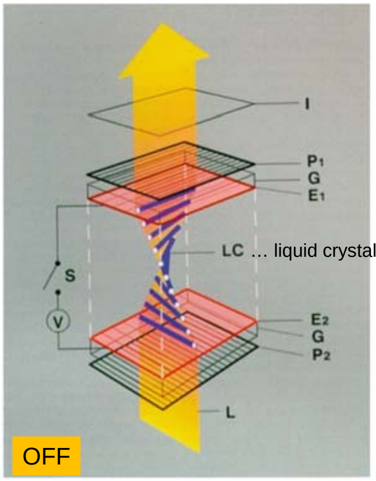
The HD44780 LCD Controller

- **Most low cost Character-based LCD modules use the Hitachi HD44780 controller chip**
- **De-facto industry standard**
  - **Typically 8, 16, 20, 24 or 40 characters/line**
  - **1, 2, or 4 lines**
  - **Handles up to 2^7 = 128 total characters/display**
- **“Standard” 14-pin interface**
- **In this course we will use the “1602A-1” LCD, which is HD44780-compatible**
- **It also has additional two pins for a back-light**
- **Data sheets and other LCD information are on class website under “Resources” & ICON**

Hitachi HD44780 Controller Chip

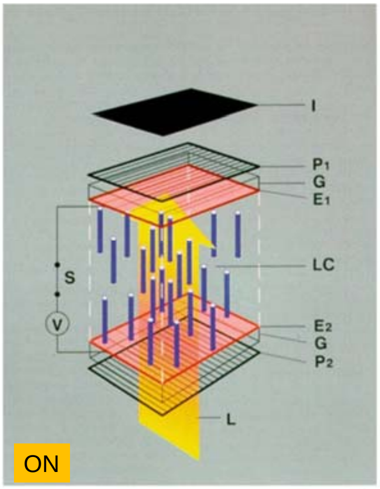


LCD Technology



LC … liquid crystal

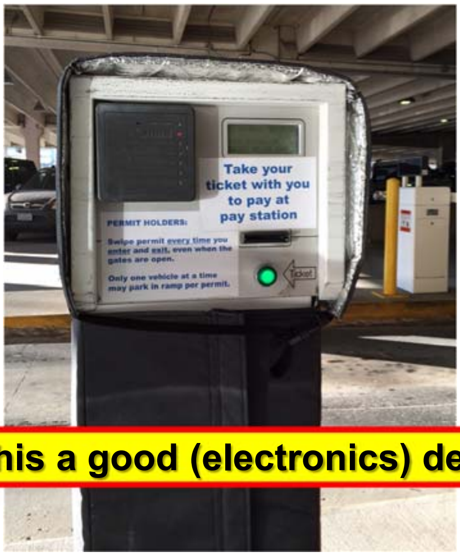
OFF



ON

The twisted nematic effect (TN-effect)

Limitations of LCDs


Take your ticket with you to pay at pay station
PERMIT HOLDERS:
Swipe permit every time you enter and exit, even when the gates are open.
Only one vehicle at a time may park in ramp per permit.
Ticket

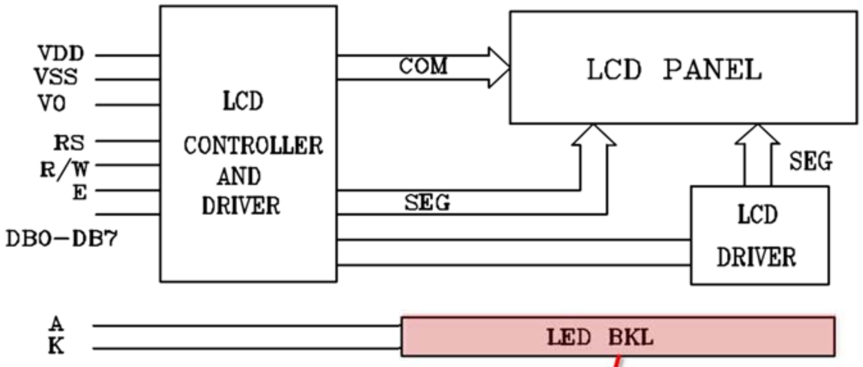
Q: Is this a good (electronics) design?

Limitations of LCDs

- **Typically, standard LCD character and graphics modules provide a temperature range of 0°C to +50°C.**
- **However, several display manufacturers offer extreme temperature models for industrial and military sectors:**
  - **-20°C to +70°C**
  - **-40°C to +85°C**

Typical LCD Display Block Diagram

Block diagram

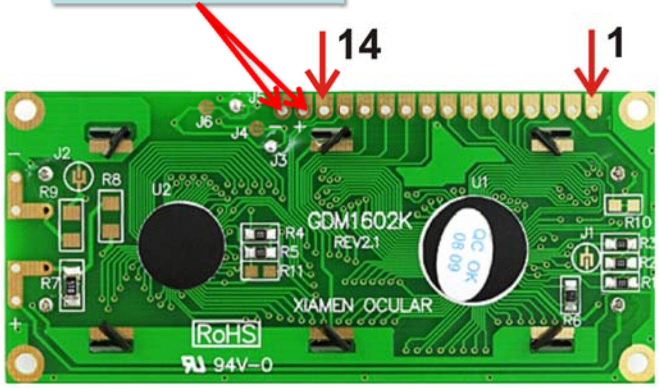


Back Light (pins 15 & 16).

LCD – Pins & Connection to µC

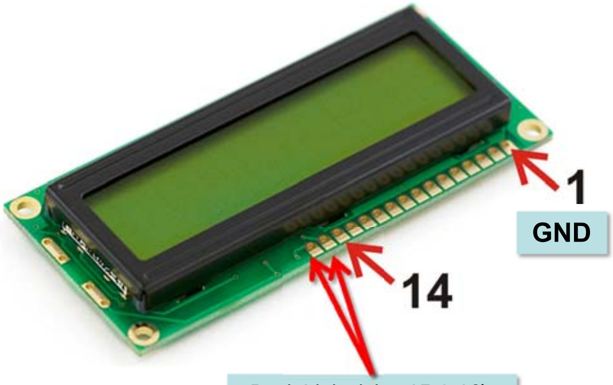
LCD Connections

Back Light (pins 15 & 16).
We will not use back light.



Typical LCD Connections

LCD Connections

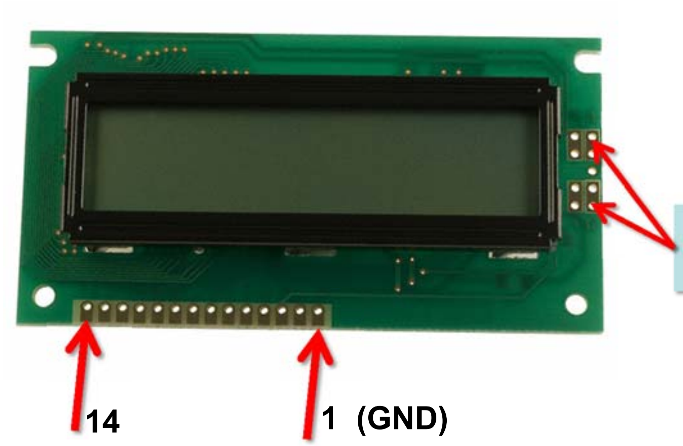


GND

Back Light (pins 15 & 16).
We will not use back light.

Typical LCD Connections

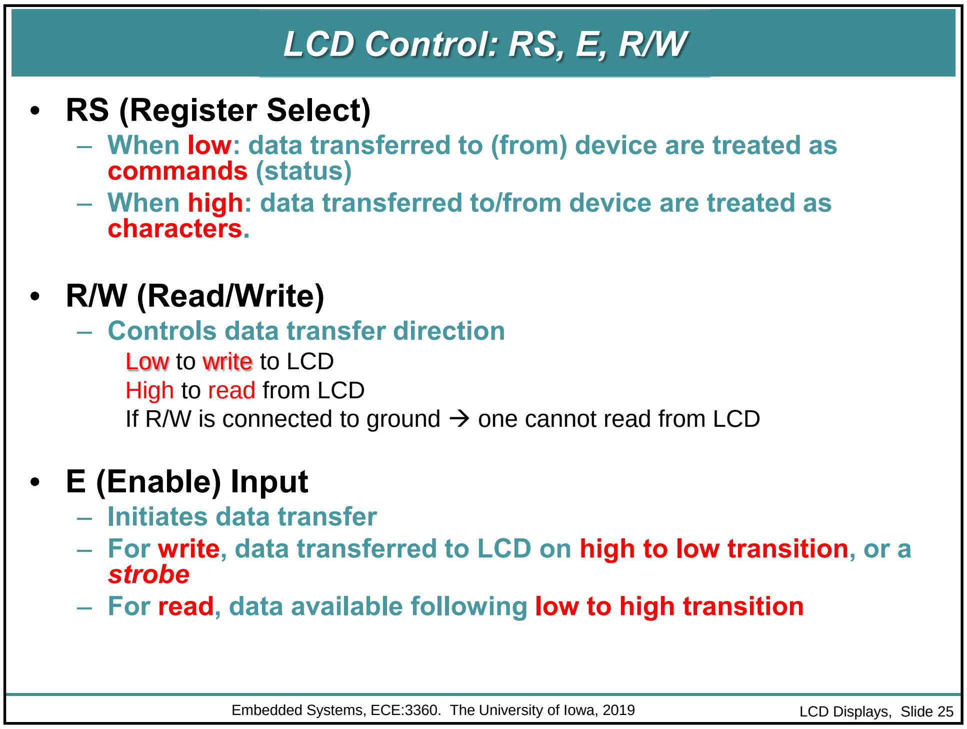
Lumex LCM-S01602DSR/B



Back Light (pins A & K). We will not use back light.

LCD Pinouts

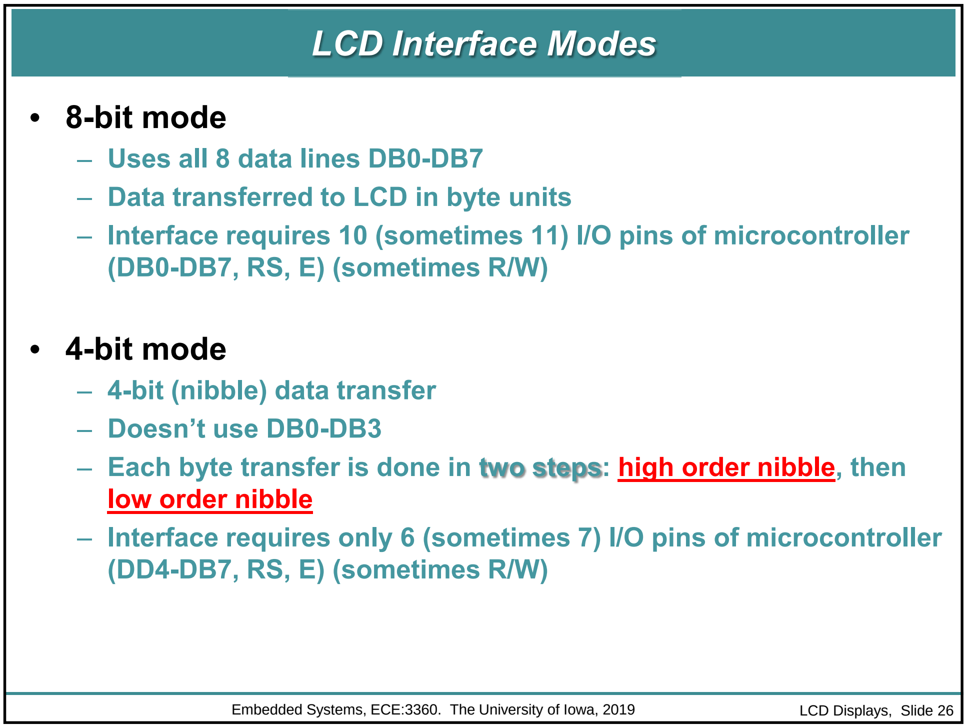
Pin number	Symbol	Level	I/O	Function
1	Vss	-	-	Power supply (GND)
2	Vcc	-	-	Power supply (+5V)
3	Vee	-	-	Contrast adjust
4	RS	0/1	I	0 = Instruction input 1 = Data input
5	R/W	0/1	I	0 = Write to LCD module 1 = Read from LCD module
6	E	1, 1→0	I	Enable signal
7	DB0	0/1	I/O	Data bus line 0 (LSB)
8	DB1	0/1	I/O	Data bus line 1
9	DB2	0/1	I/O	Data bus line 2
10	DB3	0/1	I/O	Data bus line 3
11	DB4	0/1	I/O	Data bus line 4
12	DB5	0/1	I/O	Data bus line 5
13	DB6	0/1	I/O	Data bus line 6
14	DB7	0/1	I/O	Data bus line 7 (MSB)



LCD Control: RS, E, R/W

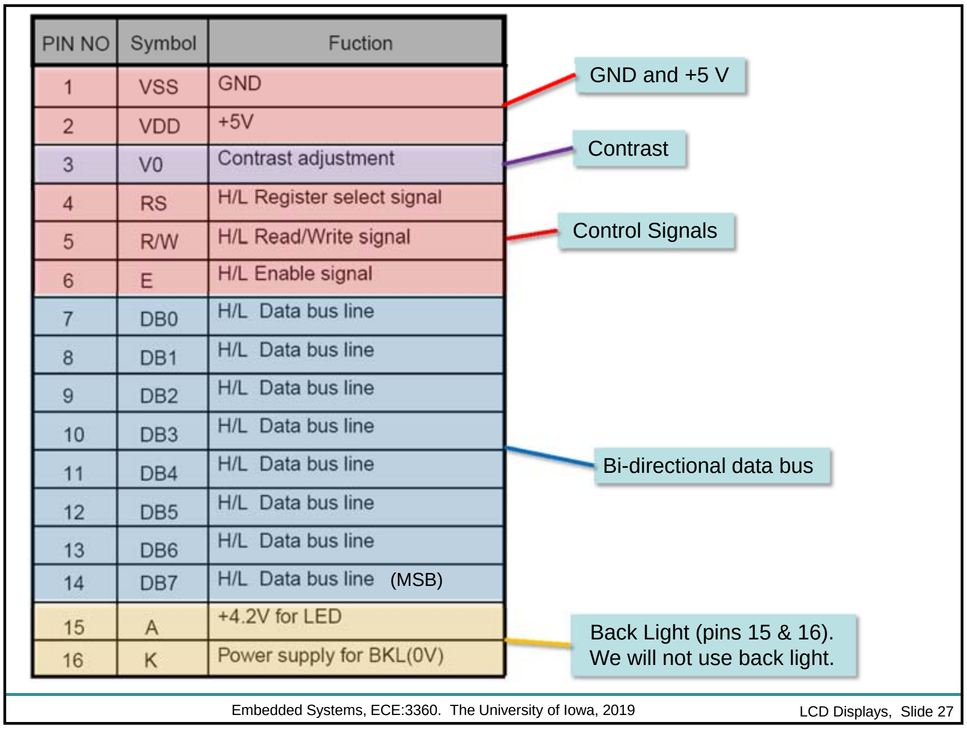
- **RS (Register Select)**
  - **When low: data transferred to (from) device are treated as commands (status)**
  - **When high: data transferred to/from device are treated as characters.**
- **R/W (Read/Write)**
  - **Controls data transfer direction**

    Low to write to LCD
    High to read from LCD
    If R/W is connected to ground → one cannot read from LCD
- **E (Enable) Input**
  - **Initiates data transfer**
  - **For write, data transferred to LCD on high to low transition, or a *strobe***
  - **For read, data available following low to high transition**



LCD Interface Modes

- **8-bit mode**
  - **Uses all 8 data lines DB0-DB7**
  - **Data transferred to LCD in byte units**
  - **Interface requires 10 (sometimes 11) I/O pins of microcontroller (DB0-DB7, RS, E) (sometimes R/W)**
- **4-bit mode**
  - **4-bit (nibble) data transfer**
  - **Doesn't use DB0-DB3**
  - **Each byte transfer is done in two steps: high order nibble, then low order nibble**
  - **Interface requires only 6 (sometimes 7) I/O pins of microcontroller (DD4-DB7, RS, E) (sometimes R/W)**



PIN NO	Symbol	Fuction
1	VSS	GND
2	VDD	+5V
3	V0	Contrast adjustment
4	RS	H/L Register select signal
5	R/W	H/L Read/Write signal
6	E	H/L Enable signal
7	DB0	H/L Data bus line
8	DB1	H/L Data bus line
9	DB2	H/L Data bus line
10	DB3	H/L Data bus line
11	DB4	H/L Data bus line
12	DB5	H/L Data bus line
13	DB6	H/L Data bus line
14	DB7	H/L Data bus line (MSB)
15	A	+4.2V for LED
16	K	Power supply for BKL(0V)

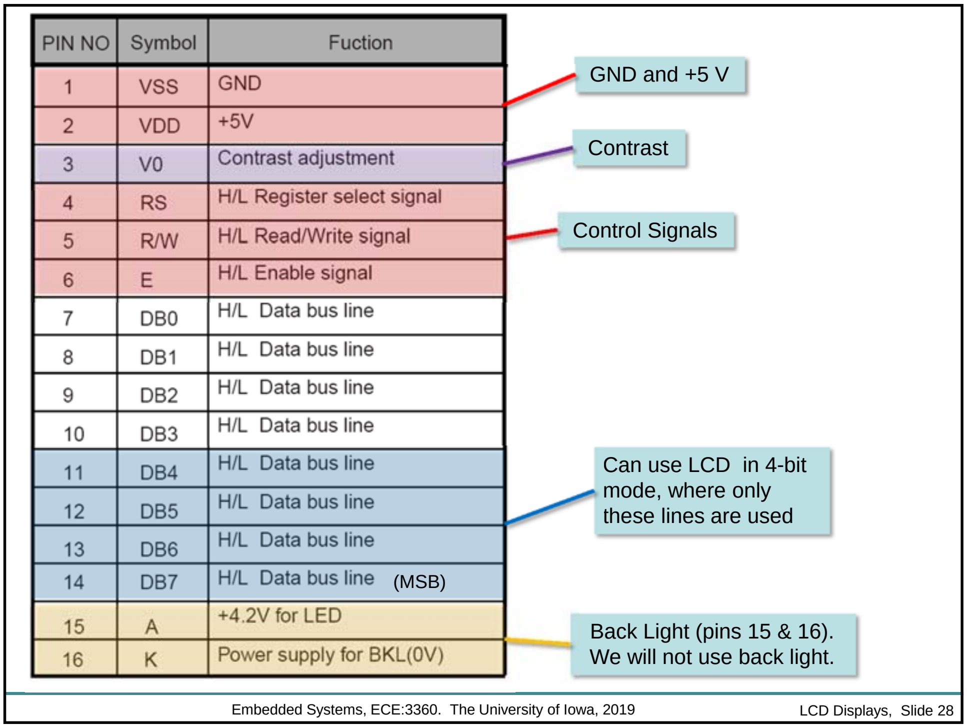
GND and +5 V

Contrast

Control Signals

Bi-directional data bus

Back Light (pins 15 & 16).
We will not use back light.



PIN NO	Symbol	Fuction
1	VSS	GND
2	VDD	+5V
3	V0	Contrast adjustment
4	RS	H/L Register select signal
5	R/W	H/L Read/Write signal
6	E	H/L Enable signal
7	DB0	H/L Data bus line
8	DB1	H/L Data bus line
9	DB2	H/L Data bus line
10	DB3	H/L Data bus line
11	DB4	H/L Data bus line
12	DB5	H/L Data bus line
13	DB6	H/L Data bus line
14	DB7	H/L Data bus line (MSB)
15	A	+4.2V for LED
16	K	Power supply for BKL(0V)

GND and +5 V

Contrast

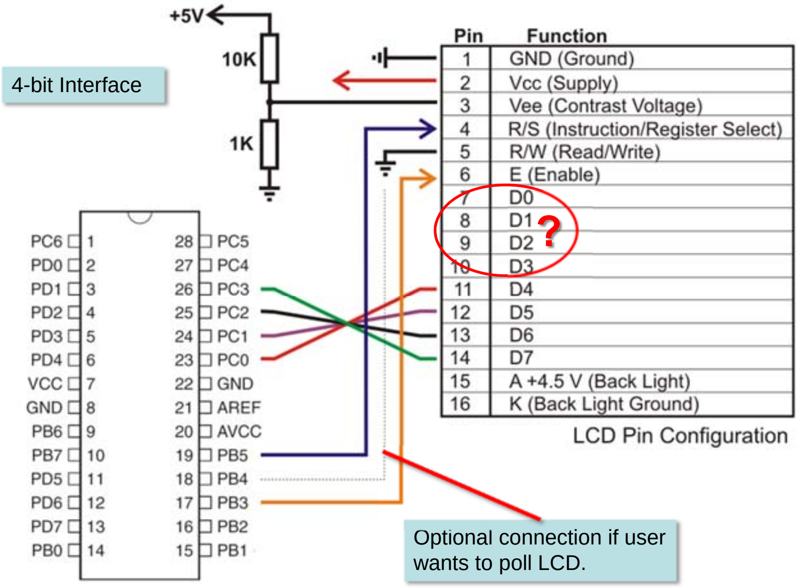
Control Signals

Can use LCD in 4-bit mode, where only these lines are used

Back Light (pins 15 & 16).
We will not use back light.

LCD Connections

4-bit Interface

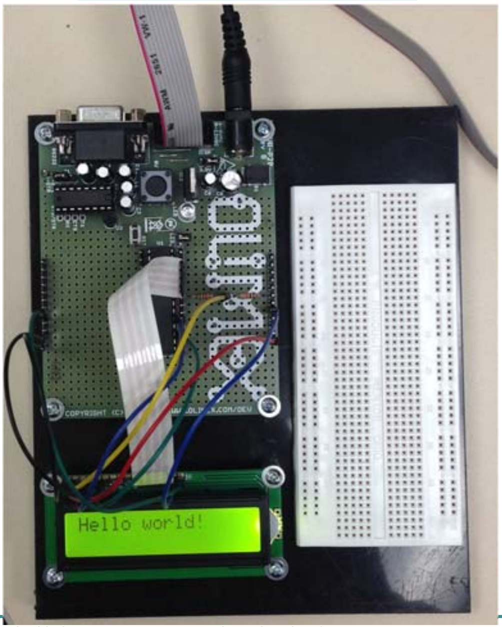


Pin	Function
1	GND (Ground)
2	Vcc (Supply)
3	Vee (Contrast Voltage)
4	R/S (Instruction/Register Select)
5	R/W (Read/Write)
6	E (Enable)
7	D0
8	D1
9	D2
10	D3
11	D4
12	D5
13	D6
14	D7
15	A +4.5 V (Back Light)
16	K (Back Light Ground)

LCD Pin Configuration

Optional connection if user wants to poll LCD.

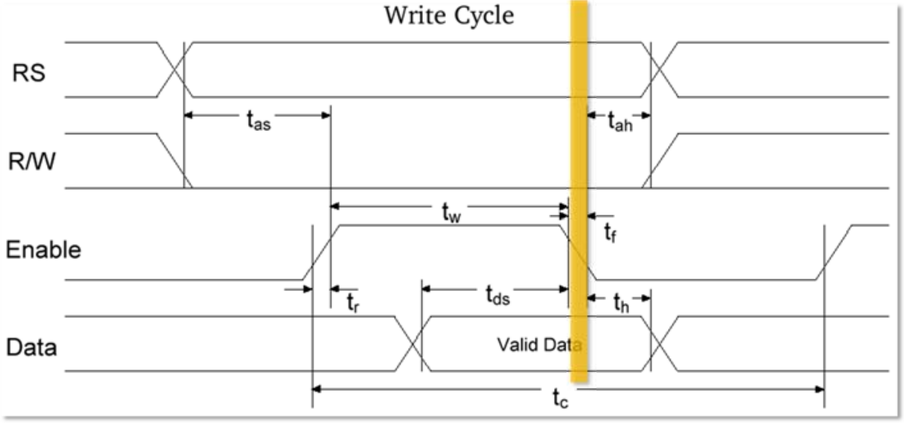
LCD Connections

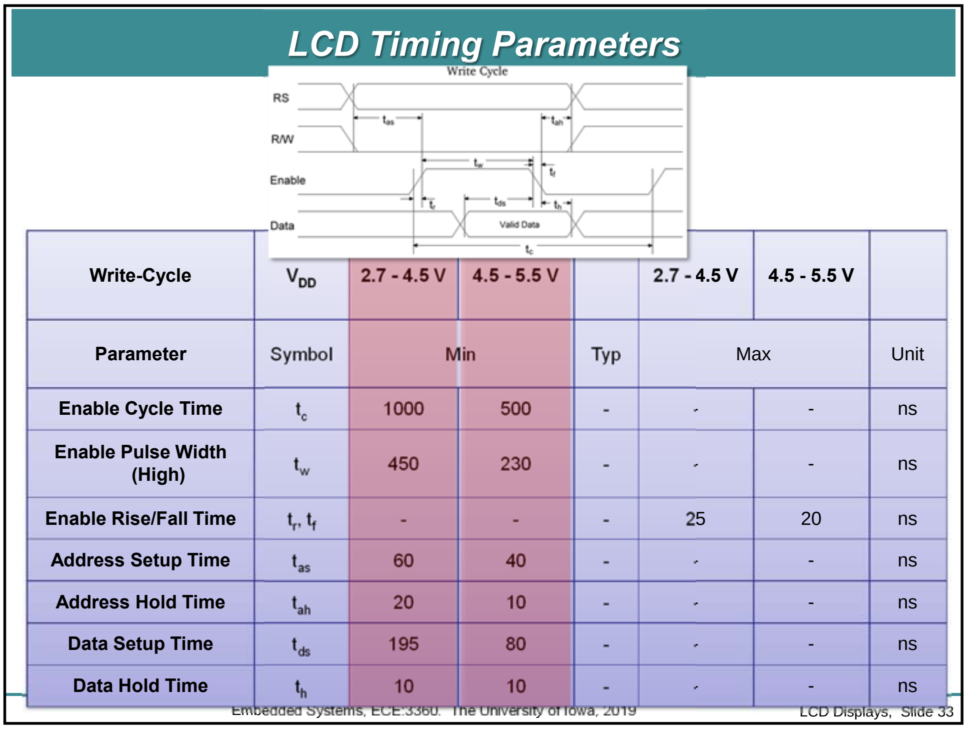


LCD Signal Timing

LCD Timing (Write Cycle)

Data is stored in the LCD

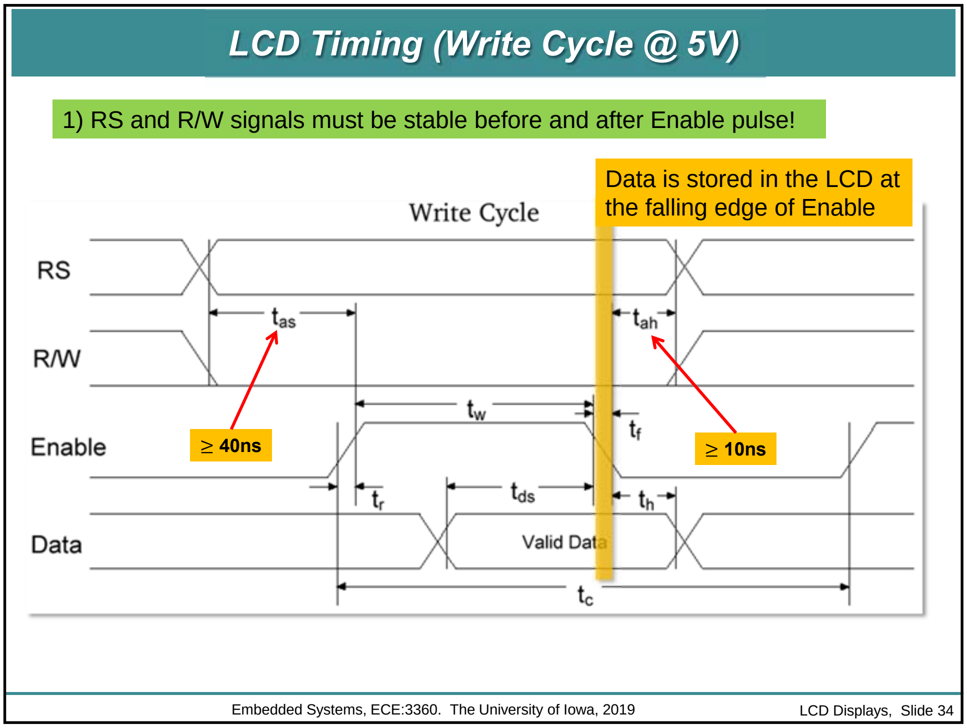




LCD Timing Parameters

Write Cycle
RS
R/W
Enable
Data
t_{as}
t_{ah}
t_w
t_f
t_r
t_{ds}
t_h
Valid Data
t_c

Write-Cycle	V_{DD}	2.7 - 4.5 V	4.5 - 5.5 V		2.7 - 4.5 V	4.5 - 5.5 V	
Parameter	Symbol	Min		Typ	Max		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



LCD Timing (Write Cycle @ 5V)

1) RS and R/W signals must be stable before and after Enable pulse!

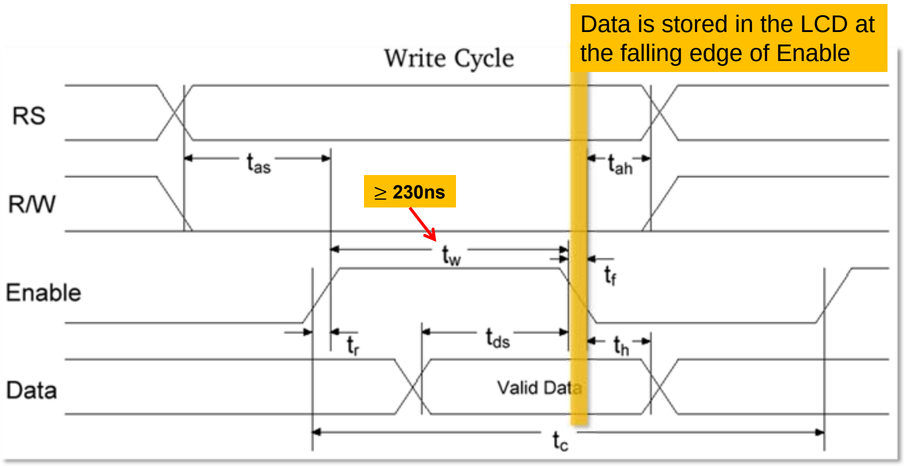
Data is stored in the LCD at the falling edge of Enable

≥ 40ns

≥ 10ns

LCD Timing (Write Cycle @ 5V)

2) The Enable pulse must have a minimum length!

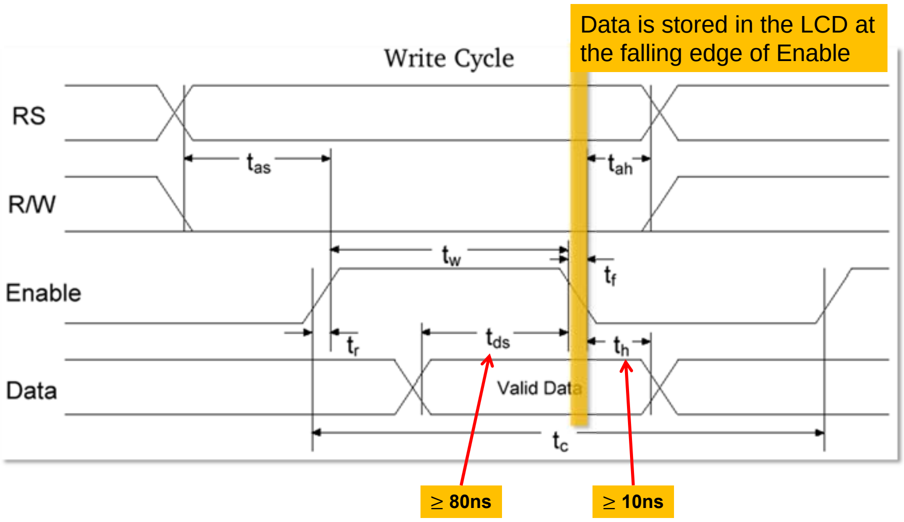


Data is stored in the LCD at the falling edge of Enable

≥ 230ns

LCD Timing (Write Cycle @ 5V)

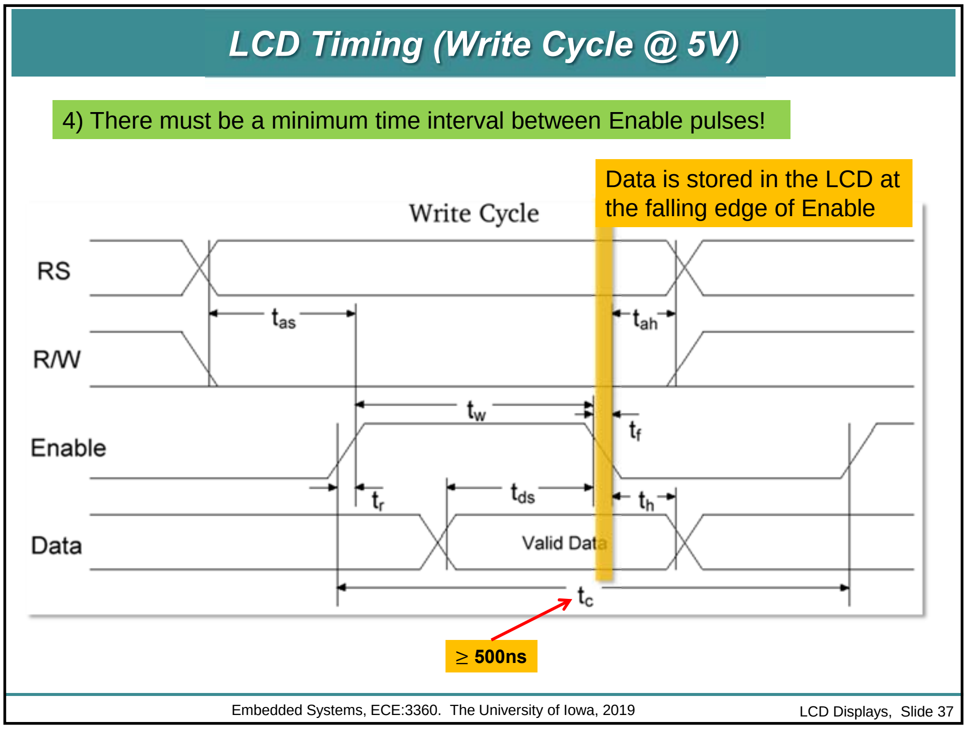
3) Data must be stable before and after the falling edge of Enable!



Data is stored in the LCD at the falling edge of Enable

≥ 80ns

≥ 10ns

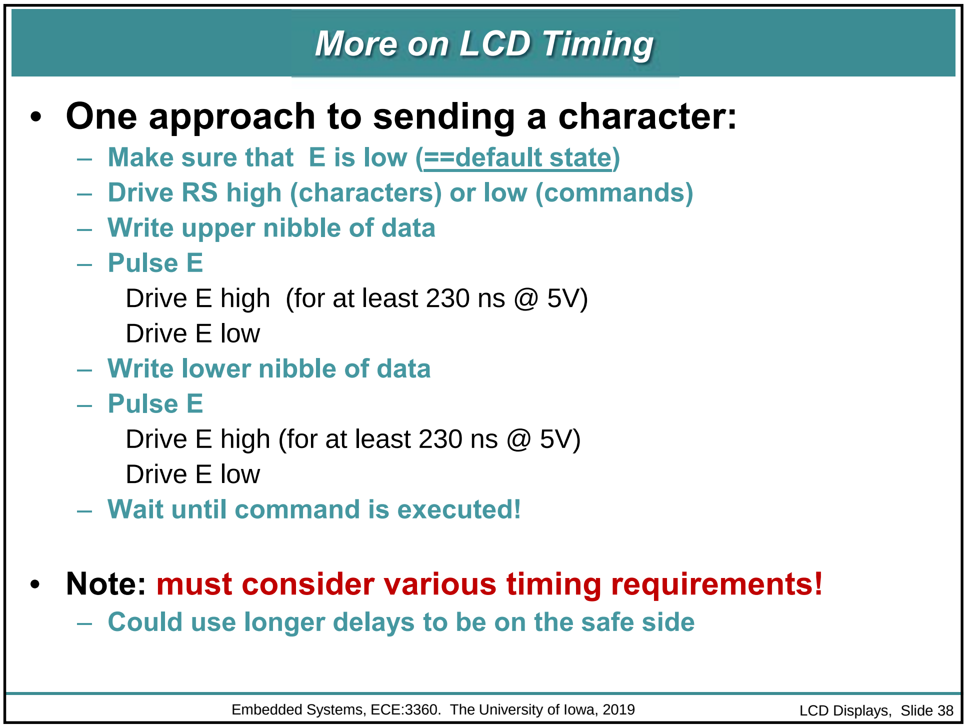


LCD Timing (Write Cycle @ 5V)

4) There must be a minimum time interval between Enable pulses!

Data is stored in the LCD at the falling edge of Enable

≥ 500ns



More on LCD Timing

- **One approach to sending a character:**
  - **Make sure that E is low (==default state)**
  - **Drive RS high (characters) or low (commands)**
  - **Write upper nibble of data**
  - **Pulse E**

    Drive E high (for at least 230 ns @ 5V)

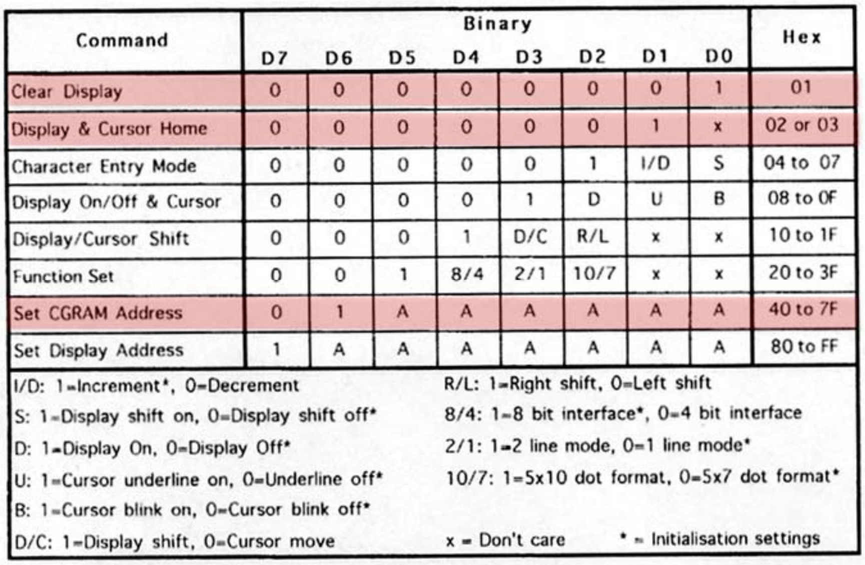
    Drive E low
  - **Write lower nibble of data**
  - **Pulse E**

    Drive E high (for at least 230 ns @ 5V)

    Drive E low
  - **Wait until command is executed!**
- **Note: must consider various timing requirements!**
  - **Could use longer delays to be on the safe side**

LCD Instructions

LCD Commands



Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	I/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

I/D: 1=Increment*, 0=Decrement

S: 1=Display shift on, 0=Display shift off*

D: 1=Display On, 0=Display Off*

U: 1=Cursor underline on, 0=Underline off*

B: 1=Cursor blink on, 0=Cursor blink off*

D/C: 1=Display shift, 0=Cursor move

R/L: 1=Right shift, 0=Left shift

8/4: 1=8 bit interface*, 0=4 bit interface

2/1: 1=2 line mode, 0=1 line mode*

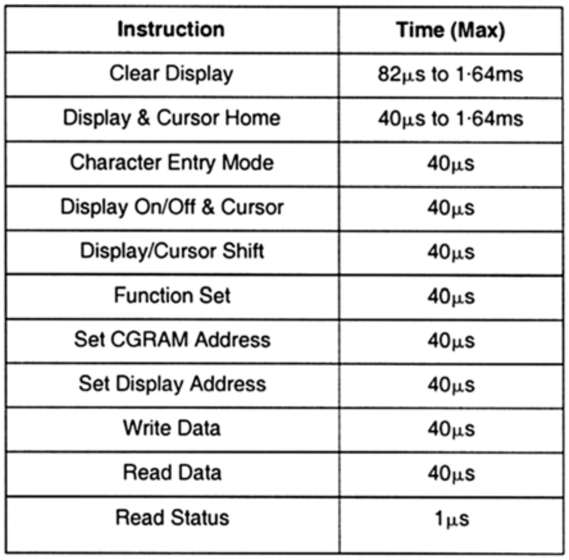
10/7: 1=5x10 dot format, 0=5x7 dot format*

x = Don't care * = Initialisation settings

Command Execution Times

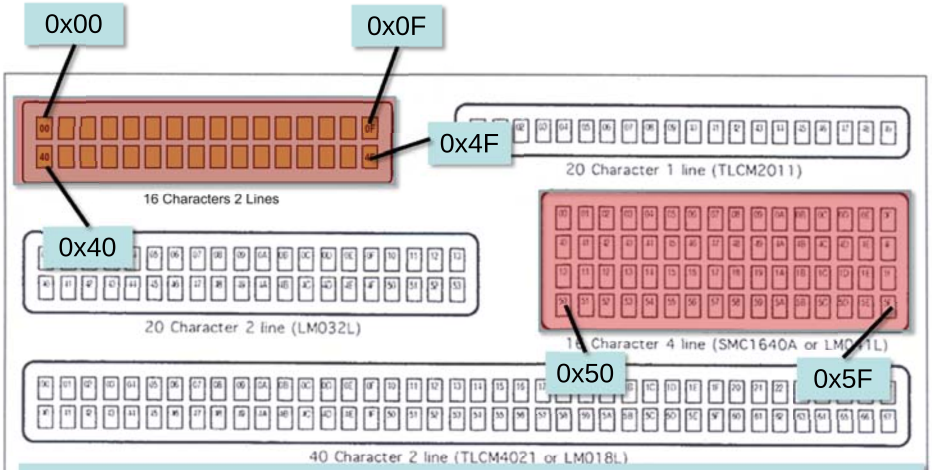
- **Most HD44780 commands take 40 µs to execute**
- **The Clear Display and Cursor Home commands can take much longer (up to 1.64 ms!)**
- **Can't issue another command until previous one has finished**
- **Two options**
  - **Busy-wait:**
    - After issuing a command, continuously monitor HD44780 status until device is not busy
    - On the schematic shown earlier, R/W is connected to ground—i.e., one can't read from LCD
  - **Insert at least a 40 µs (or, in some cases, much longer) delay between commands**

LCD Command Execution Times



Instruction	Time (Max)
Clear Display	82µs to 1·64ms
Display & Cursor Home	40µs to 1·64ms
Character Entry Mode	40µs
Display On/Off & Cursor	40µs
Display/Cursor Shift	40µs
Function Set	40µs
Set CGRAM Address	40µs
Set Display Address	40µs
Write Data	40µs
Read Data	40µs
Read Status	1µs

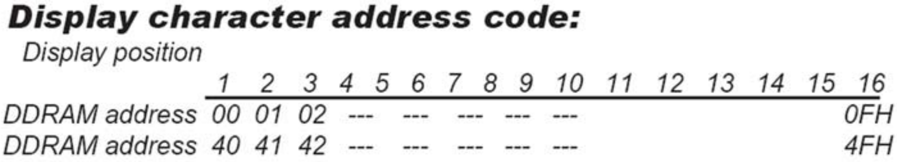
LCD Cursor Position - Display Addresses


0x00
0x0F
0x4F
16 Characters 2 Lines
0x40
20 Character 1 line (TLCM2011)
20 Character 2 line (LM032L)
16 Character 4 line (SMC1640A or LM041L)
0x50
0x5F
40 Character 2 line (TLCM4021 or LM018L)

Note: The HD44780 always maintains an internal buffer of 128 character positions. For a given LCD, not all of these are displayable. You can still write characters to these positions, but they won't appear on the screen

The Cursor Position Map for the LCD

The LCD used in this course is a 16×2 character LCD



Display character address code:

Display position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	00	01	02	---		---	---		---	---						0FH
DDRAM address	40	41	42	---		---	---		---	---						4FH

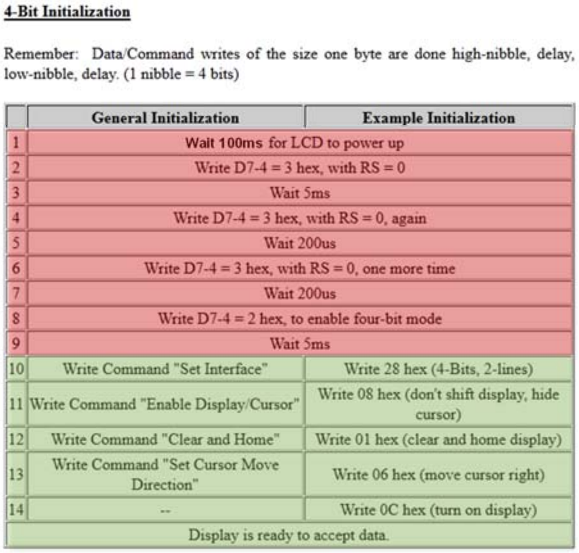
LCD Initialization

Initialization

- The HD44780 has some initialization quirks.
- The recommended initialization sequence for the 4-bit mode (default: 8-bit mode), following power-up is:
  - Wait for 100 ms
  - Set the device to 8-bit mode
  - Wait 5 ms
  - Set the device to 8-bit mode
  - Wait at least $200\ \mu s$
  - Set the device to 8-bit mode
  - Wait at least $200\ \mu s$
  - Set device to 4-bit mode
  - Wait at least 5 ms
  - Complete additional device configuration

    Clear screen, Entry Mode, display shift, …

Initialization - 4-Bit Mode



4-Bit Initialization

Remember: Data/Command writes of the size one byte are done high-nibble, delay, low-nibble, delay. (1 nibble = 4 bits)

	General Initialization	Example Initialization
1	Wait 100ms for LCD to power up	
2	Write D7-4 = 3 hex, with RS = 0	
3	Wait 5ms	
4	Write D7-4 = 3 hex, with RS = 0, again	
5	Wait 200us	
6	Write D7-4 = 3 hex, with RS = 0, one more time	
7	Wait 200us	
8	Write D7-4 = 2 hex, to enable four-bit mode	
9	Wait 5ms	
10	Write Command "Set Interface"	Write 28 hex (4-Bits, 2-lines)
11	Write Command "Enable Display/Cursor"	Write 08 hex (don't shift display, hide cursor)
12	Write Command "Clear and Home"	Write 01 hex (clear and home display)
13	Write Command "Set Cursor Move Direction"	Write 06 hex (move cursor right)
14	--	Write 0C hex (turn on display)
	Display is ready to accept data.	

8-bit MODE! Write only upper nibble!

4-bit MODE! Write upper & lower nibble!

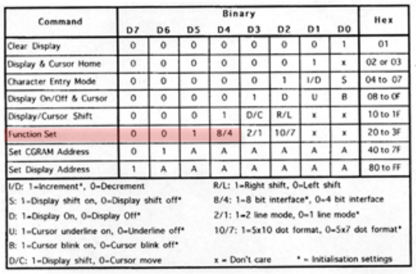
Credits: http://joshuagalloway.com/lcd.html

Initialization, Continued

The LCD can be properly initialized by writing the following command string to the controller, a nibble at a time (Must be in **Command** mode (RS=0); also, must wait 0.1 s first, and observe delays as indicated on previous slide):

```
LCDstr:.db   0x33,0x32,0x28,0x01,0x0c,0x06
```

Function set 8-bit mode

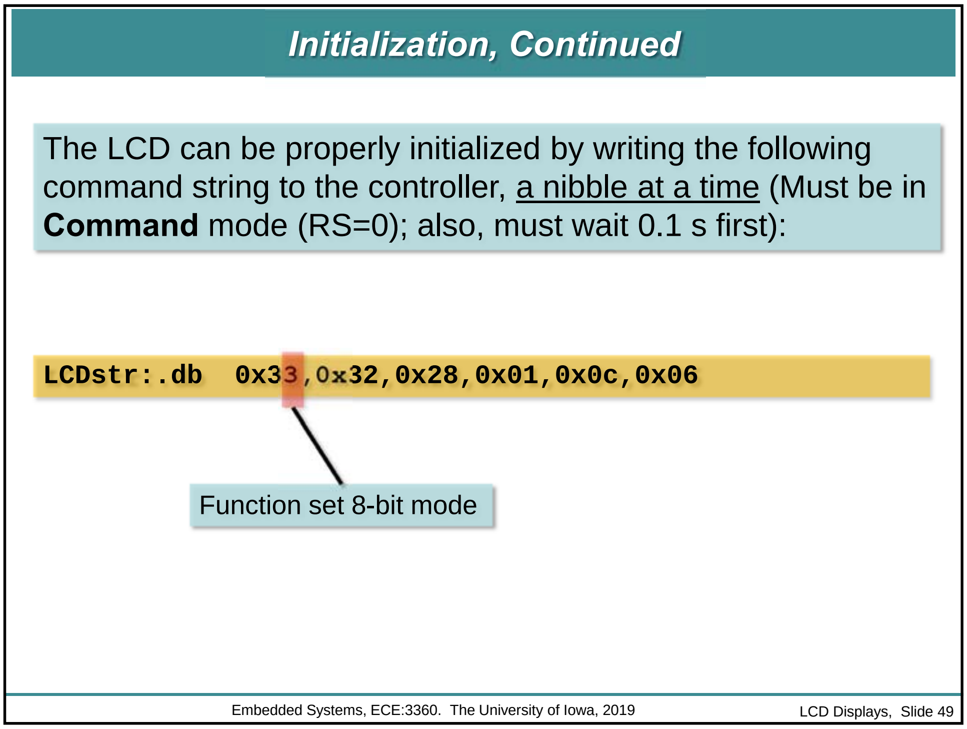


Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	I/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

I/D: 1=Increment*, 0=Decrement
S: 1=Display shift on, 0=Display shift off*
D: 1=Display On, 0=Display Off*
U: 1=Cursor underline on, 0=Underline off*
B: 1=Cursor blink on, 0=Cursor blink off*
D/C: 1=Display shift, 0=Cursor move

R/L: 1=Right shift, 0=Left shift
8/4: 1=8 bit interface*, 0=4 bit interface
2/1: 1=2 line mode, 0=1 line mode*
10/7: 1=5x10 dot format, 0=5x7 dot format*

x = Don't care   * = Initialisation settings

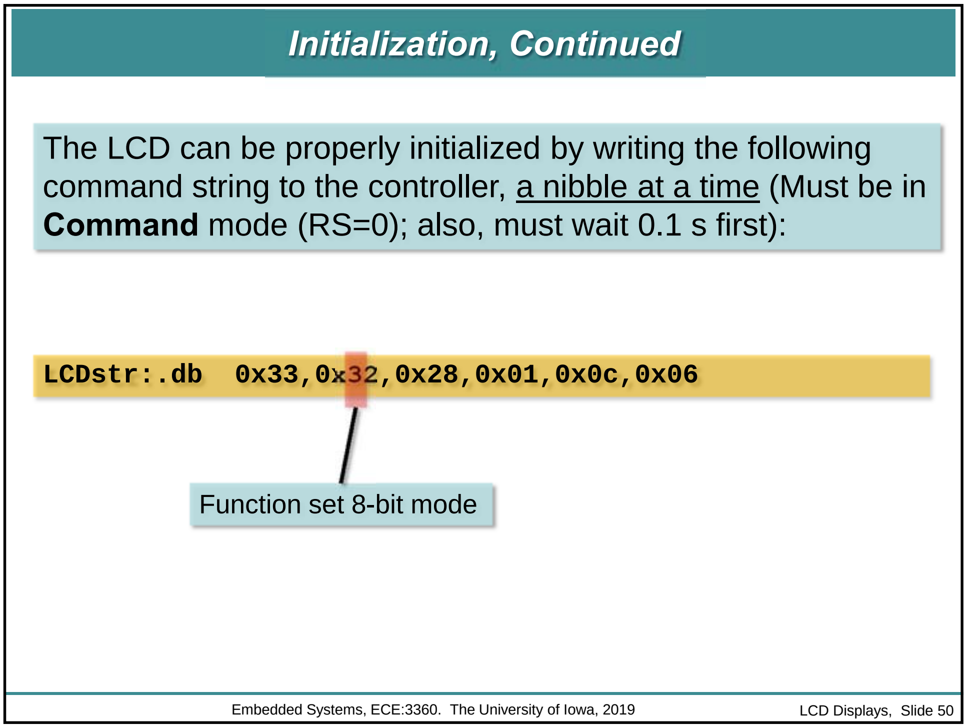


Initialization, Continued

The LCD can be properly initialized by writing the following command string to the controller, a nibble at a time (Must be in **Command** mode (RS=0); also, must wait 0.1 s first):

```
LCDstr:.db  0x33,0x32,0x28,0x01,0x0c,0x06
```

Function set 8-bit mode



Initialization, Continued

The LCD can be properly initialized by writing the following command string to the controller, a nibble at a time (Must be in **Command** mode (RS=0); also, must wait 0.1 s first):

```
LCDstr:.db  0x33,0x32,0x28,0x01,0x0c,0x06
```

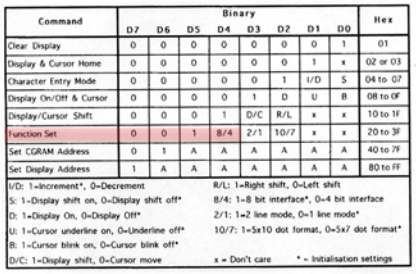
Function set 8-bit mode

Initialization, Continued

The LCD can be properly initialized by writing the following command string to the controller, a nibble at a time (Must be in **Command** mode (RS=0); also, must wait 0.1 s first):

```
LCDstr:.db  0x33,0x32,0x28,0x01,0x0c,0x06
```

Function set 4-bit mode



Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	I/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

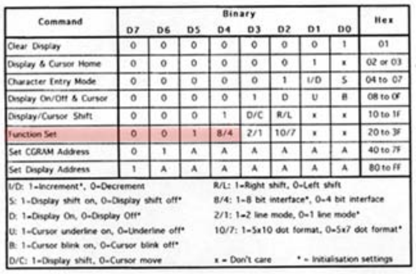
I/D: 1=Increment*, 0=Decrement
S: 1=Display shift on, 0=Display shift off*
D: 1=Display On, 0=Display Off*
U: 1=Cursor underline on, 0=Underline off*
B: 1=Cursor blink on, 0=Cursor blink off*
D/C: 1=Display shift, 0=Cursor move
R/L: 1=Right shift, 0=Left shift
8/4: 1=8 bit interface*, 0=4 bit interface
2/1: 1=2 line mode, 0=1 line mode*
10/7: 1=5x10 dot format, 0=5x7 dot format*
x = Don't care
* = Initialisation settings

Initialization, Continued

The LCD can be properly initialized by writing the following command string to the controller, a nibble at a time (Must be in **Command** mode (RS=0); also, must wait 0.1 s first):

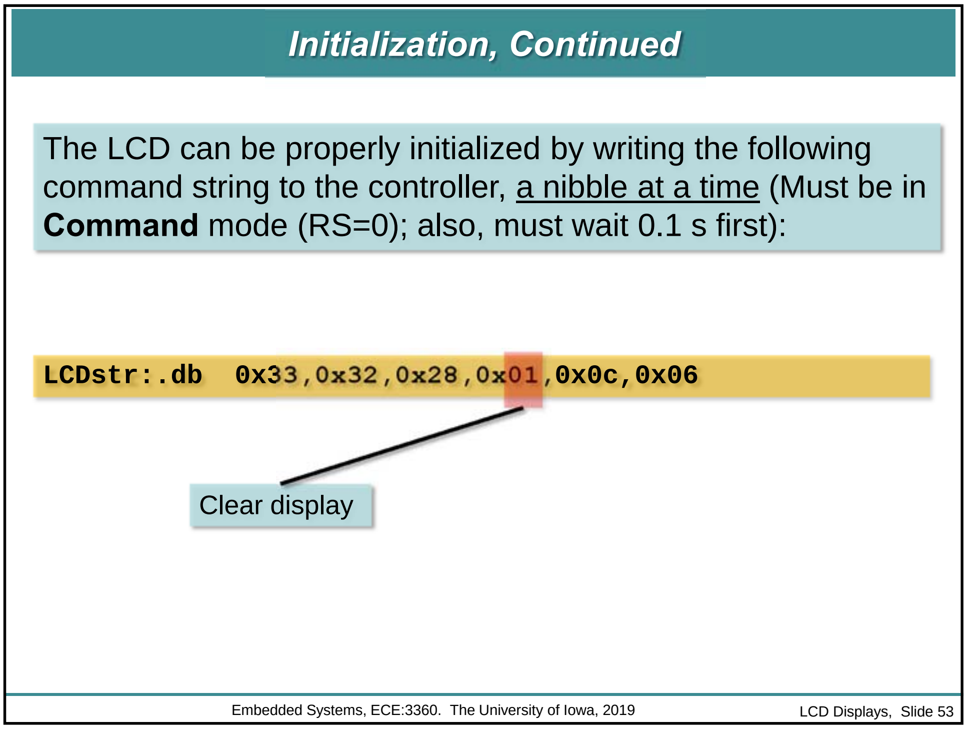
```
LCDstr:.db  0x33,0x32,0x28,0x01,0x0c,0x06
```

Function set 4 bit mode, two rows, 5x7 characters



Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	I/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

I/D: 1=Increment*, 0=Decrement
S: 1=Display shift on, 0=Display shift off*
D: 1=Display On, 0=Display Off*
U: 1=Cursor underline on, 0=Underline off*
B: 1=Cursor blink on, 0=Cursor blink off*
D/C: 1=Display shift, 0=Cursor move
R/L: 1=Right shift, 0=Left shift
8/4: 1=8 bit interface*, 0=4 bit interface
2/1: 1=2 line mode, 0=1 line mode*
10/7: 1=5x10 dot format, 0=5x7 dot format*
x = Don't care
* = Initialisation settings



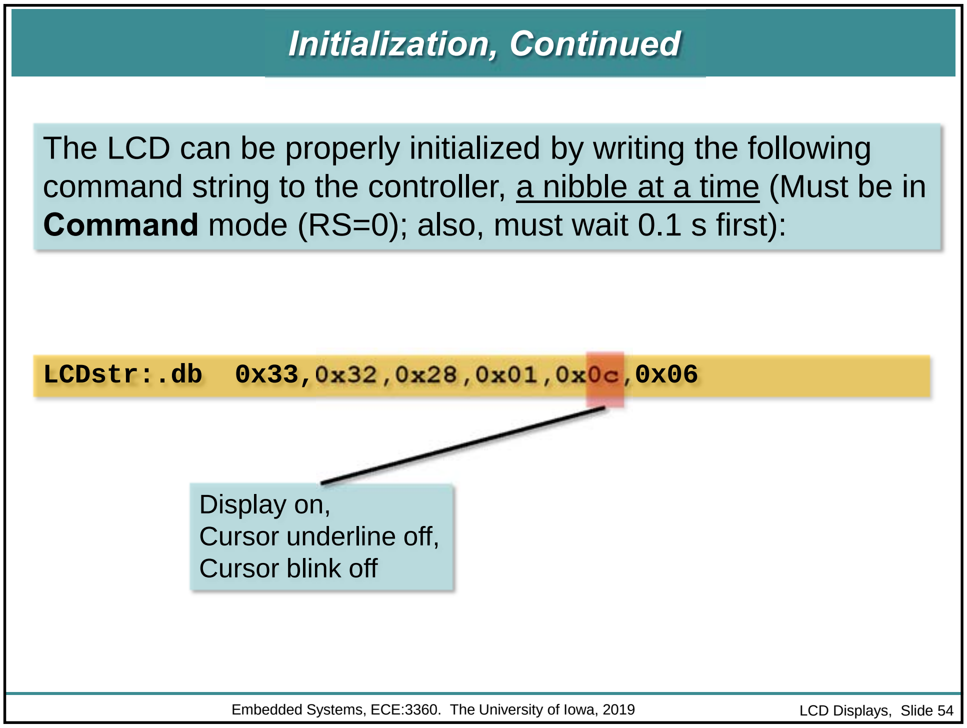
Initialization, Continued

The LCD can be properly initialized by writing the following command string to the controller, a nibble at a time (Must be in **Command** mode (RS=0); also, must wait 0.1 s first):

```
LCDstr:.db  0x33,0x32,0x28,0x01,0x0c,0x06
```

Clear display

Embedded Systems, ECE:3360. The University of Iowa, 2019 LCD Displays, Slide 53



Initialization, Continued

The LCD can be properly initialized by writing the following command string to the controller, a nibble at a time (Must be in **Command** mode (RS=0); also, must wait 0.1 s first):

```
LCDstr:.db  0x33,0x32,0x28,0x01,0x0c,0x06
```

Display on,
Cursor underline off,
Cursor blink off

Embedded Systems, ECE:3360. The University of Iowa, 2019 LCD Displays, Slide 54

Initialization, Continued

The LCD can be properly initialized by writing the following command string to the controller, a nibble at a time (Must be in **Command** mode (RS=0); also, must wait 0.1 s first):

```
LCDstr:.db  0x33,0x32,0x28,0x01,0x0c,0x06
```

Display shift off,
Address increment

This causes display address (cursor position) to be automatically incremented following each character write.
Can also set controller to automatically decrement the address

Sending Characters in 4-bit Mode

Sending Characters to LCD, 4-bit Mode

Assume LCD has been initialized in 4-bit mode. To send data, one sends upper nibble, then lower nibble. Assume **PC0…PC3** are connected to **D4…D7**.

Let's send the ASCII character 'E' which is equivalent to '0x45'

Important → set RS line!

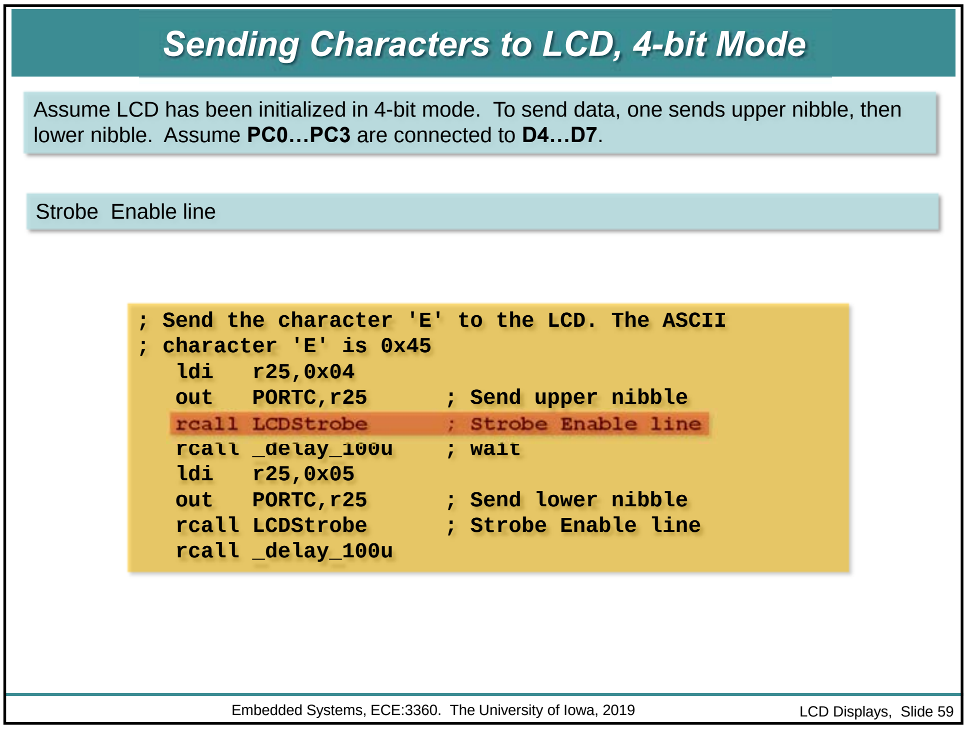
```
; Send the character 'E' to the LCD. The ASCII
; character 'E' is 0x45
   ldi   r25,0x04
   out   PORTC,r25      ; Send upper nibble
   rcall LCDStrobe      ; Strobe Enable line
   rcall _delay_100u    ; wait
   ldi   r25,0x05
   out   PORTC,r25      ; Send lower nibble
   rcall LCDStrobe      ; Strobe Enable line
   rcall _delay_100u
```

Sending Characters to LCD, 4-bit Mode

Assume LCD has been initialized in 4-bit mode. To send data, one sends upper nibble, then lower nibble (**RS=1**). Assume **PC0…PC3** are connected to **D4…D7**.

Load upper nibble and send it to **PORTC** (where the LCD is connected)

```
; Send the character 'E' to the LCD. The ASCII
; character 'E' is 0x45
   ldi   r25,0x04
   out   PORTC,r25      ; Send upper nibble
   rcall LCDStrobe      ; Strobe Enable line
   rcall _delay_100u    ; wait
   ldi   r25,0x05
   out   PORTC,r25      ; Send lower nibble
   rcall LCDStrobe      ; Strobe Enable line
   rcall _delay_100u
```

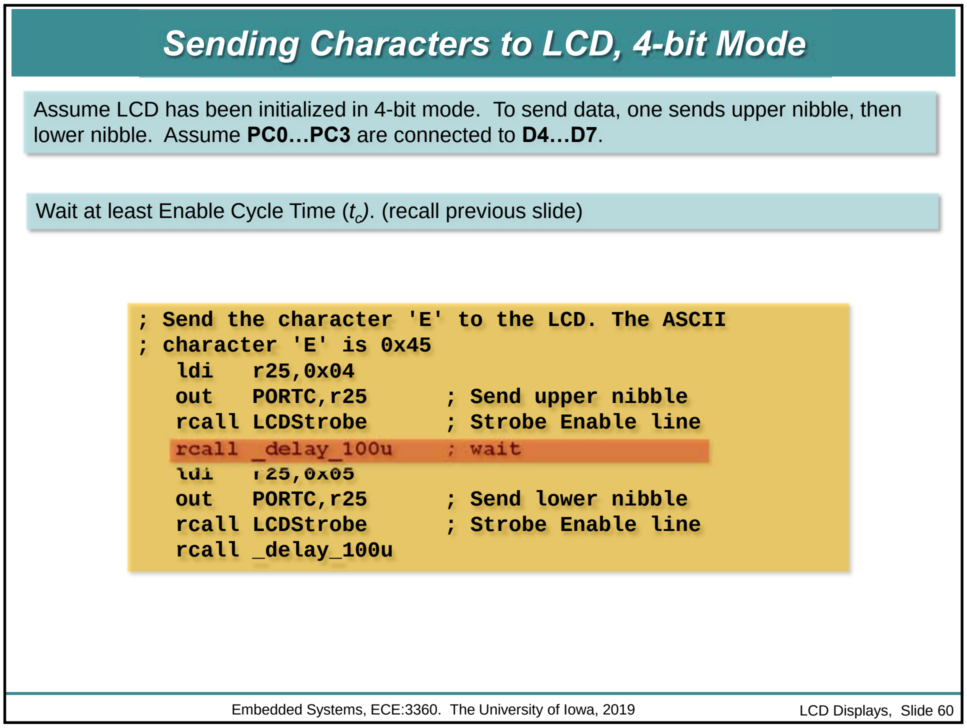


Sending Characters to LCD, 4-bit Mode

Assume LCD has been initialized in 4-bit mode. To send data, one sends upper nibble, then lower nibble. Assume **PC0…PC3** are connected to **D4…D7**.

Strobe Enable line

```
; Send the character 'E' to the LCD. The ASCII
; character 'E' is 0x45
   ldi   r25,0x04
   out   PORTC,r25      ; Send upper nibble
   rcall LCDStrobe      ; Strobe Enable line
   rcall _delay_100u    ; wait
   ldi   r25,0x05
   out   PORTC,r25      ; Send lower nibble
   rcall LCDStrobe      ; Strobe Enable line
   rcall _delay_100u
```

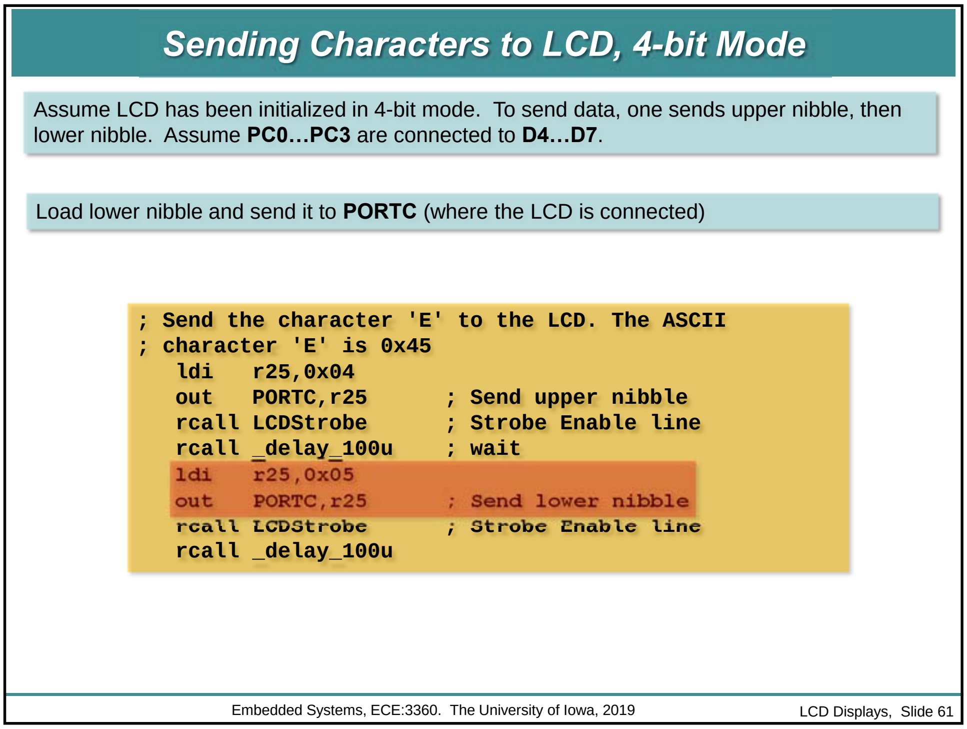


Sending Characters to LCD, 4-bit Mode

Assume LCD has been initialized in 4-bit mode. To send data, one sends upper nibble, then lower nibble. Assume **PC0…PC3** are connected to **D4…D7**.

Wait at least Enable Cycle Time (t_c). (recall previous slide)

```
; Send the character 'E' to the LCD. The ASCII
; character 'E' is 0x45
   ldi   r25,0x04
   out   PORTC,r25      ; Send upper nibble
   rcall LCDStrobe      ; Strobe Enable line
   rcall _delay_100u    ; wait
   ldi   r25,0x05
   out   PORTC,r25      ; Send lower nibble
   rcall LCDStrobe      ; Strobe Enable line
   rcall _delay_100u
```

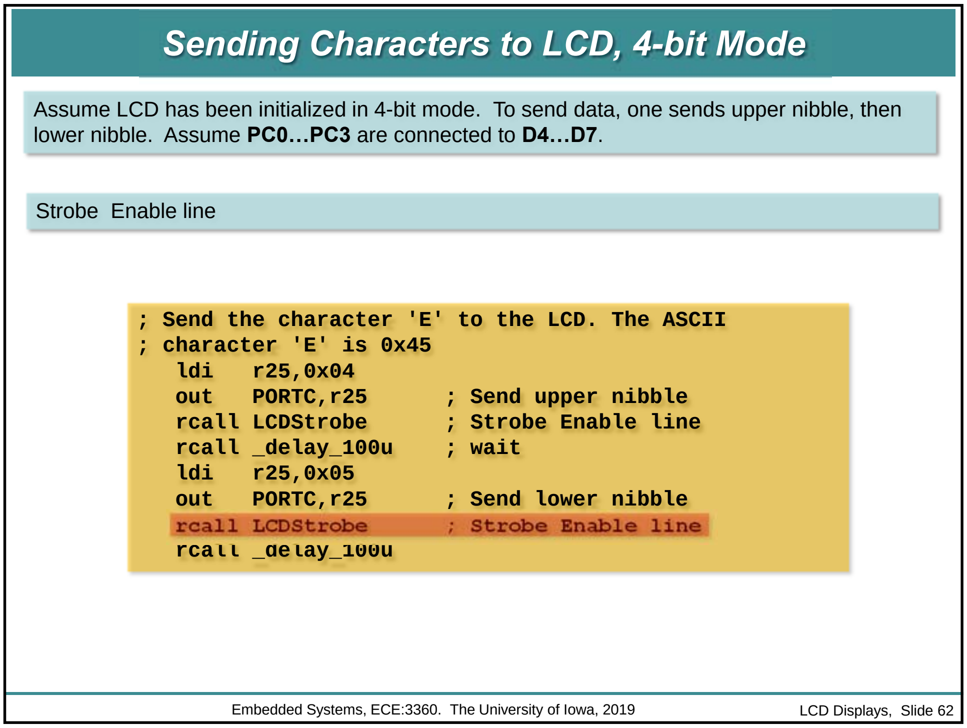


Sending Characters to LCD, 4-bit Mode

Assume LCD has been initialized in 4-bit mode. To send data, one sends upper nibble, then lower nibble. Assume **PC0…PC3** are connected to **D4…D7**.

Load lower nibble and send it to **PORTC** (where the LCD is connected)

```
; Send the character 'E' to the LCD. The ASCII
; character 'E' is 0x45
   ldi   r25,0x04
   out   PORTC,r25       ; Send upper nibble
   rcall LCDStrobe       ; Strobe Enable line
   rcall _delay_100u     ; wait
   ldi   r25,0x05
   out   PORTC,r25       ; Send lower nibble
   rcall LCDStrobe       ; Strobe Enable line
   rcall _delay_100u
```

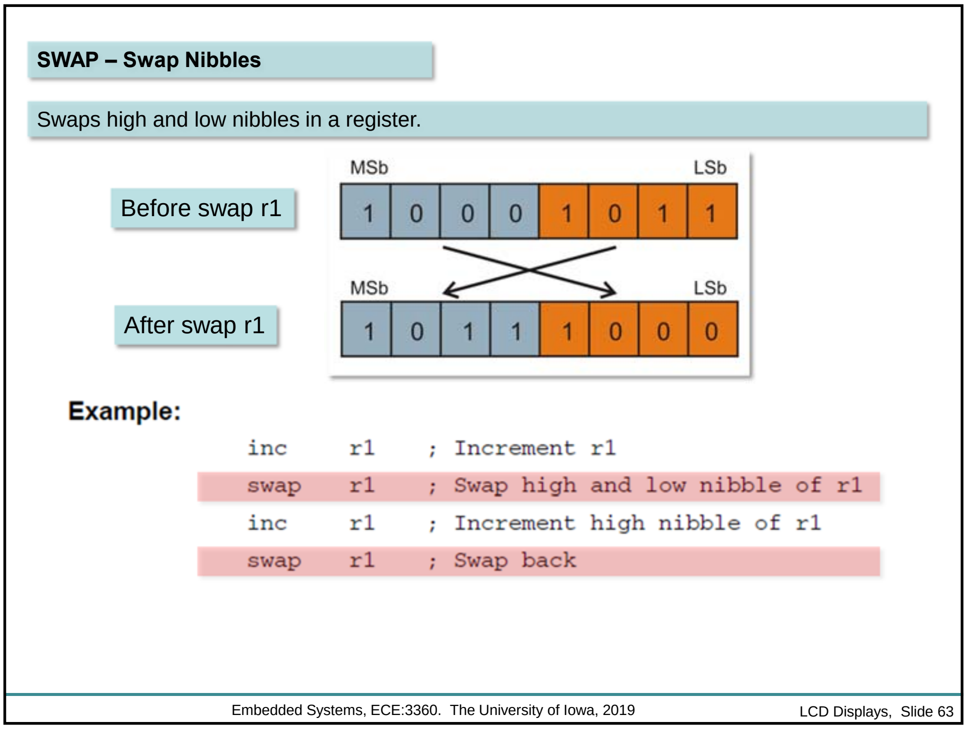


Sending Characters to LCD, 4-bit Mode

Assume LCD has been initialized in 4-bit mode. To send data, one sends upper nibble, then lower nibble. Assume **PC0…PC3** are connected to **D4…D7**.

Strobe Enable line

```
; Send the character 'E' to the LCD. The ASCII
; character 'E' is 0x45
   ldi   r25,0x04
   out   PORTC,r25       ; Send upper nibble
   rcall LCDStrobe       ; Strobe Enable line
   rcall _delay_100u     ; wait
   ldi   r25,0x05
   out   PORTC,r25       ; Send lower nibble
   rcall LCDStrobe       ; Strobe Enable line
   rcall _delay_100u
```



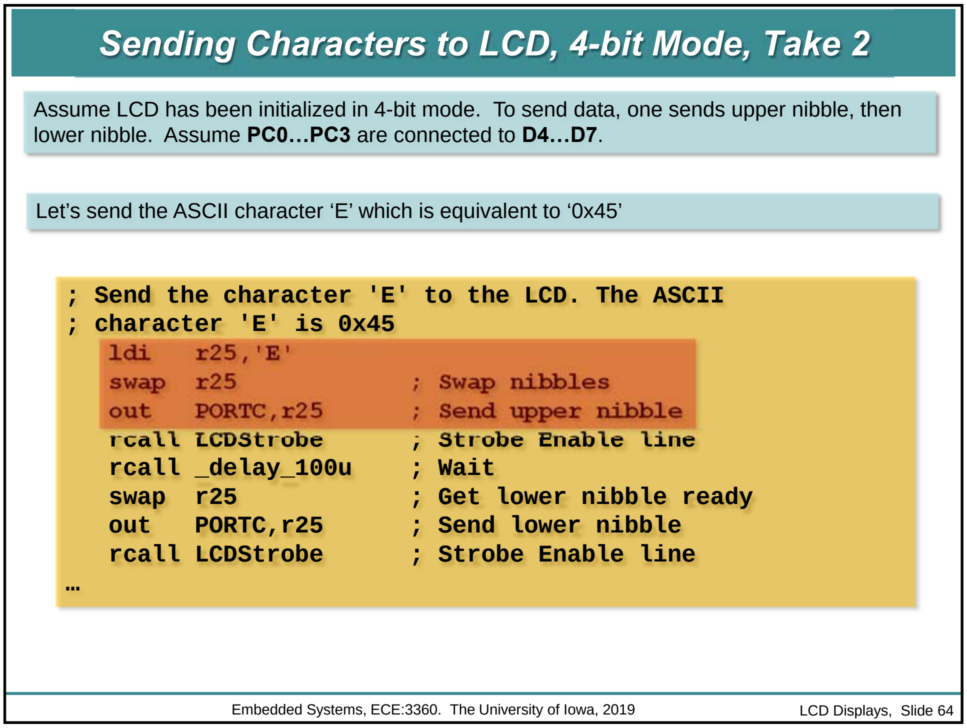
SWAP – Swap Nibbles

Swaps high and low nibbles in a register.

Before swap r1
MSb
LSb
1 0 0 0 1 0 1 1
After swap r1
MSb
LSb
1 0 1 1 1 0 0 0

Example:

```
inc     r1    ; Increment r1
swap    r1    ; Swap high and low nibble of r1
inc     r1    ; Increment high nibble of r1
swap    r1    ; Swap back
```



Sending Characters to LCD, 4-bit Mode, Take 2

Assume LCD has been initialized in 4-bit mode. To send data, one sends upper nibble, then lower nibble. Assume **PC0…PC3** are connected to **D4…D7**.

Let's send the ASCII character 'E' which is equivalent to '0x45'

```
; Send the character 'E' to the LCD. The ASCII
; character 'E' is 0x45
   ldi   r25,'E'
   swap  r25              ; Swap nibbles
   out   PORTC,r25        ; Send upper nibble
   rcall LCDStrobe        ; Strobe Enable line
   rcall _delay_100u      ; Wait
   swap  r25              ; Get lower nibble ready
   out   PORTC,r25        ; Send lower nibble
   rcall LCDStrobe        ; Strobe Enable line
…
```

Sending Characters to LCD, 4-bit Mode, Take 2

Assume LCD has been initialized in 4-bit mode. To send data, one sends upper nibble, then lower nibble. Assume **PC0…PC4** are connected to **D4…D7**.

Let's send the ASCII character 'E' which is equivalent to '0x45'

```
; Send the character 'E' to the LCD. The ASCII
; character 'E' is 0x45
   ldi   r25,'E'
   swap  r25             ; Swap nibbles
   out   PORTC,r25       ; Send upper nibble
   rcall LCDStrobe       ; Strobe Enable line
   rcall _delay_100u     ; Wait
   swap  r25             ; Get lower nibble ready
   out   PORTC,r25       ; Send lower nibble
   rcall LCDStrobe       ; Strobe Enable line
…
```

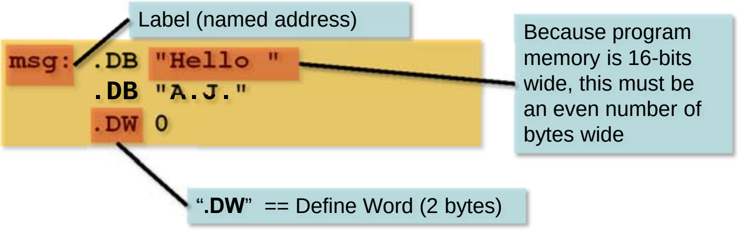
Writing a Character String

To Write a String of Characters LCD

- Drive RS low (command mode)
- Send a Set Display Address command to the LCD to establish initial display position
- Drive RS high (character mode)
- Send first character to LCD
- Send second character to LCD
- etc.
- The display position will automatically increment or decrement depending upon how you configured the LCD with the Character Entry Command
- Need to wait at least 100 µs between characters

Tables and Static Strings in Program Memory

- Data (RAM) memory is limited, one does not want to use this for storing constants, tables etc.
- One can create tables in program (Flash) memory during assembly. “Tables” can be quite large.
- The lpm (load program memory) instruction allows one to access data stored in program memory



```
msg: .DB "Hello "
     .DB "A.J."
     .DW 0
```

Label (named address)

Because program memory is 16-bits wide, this must be an even number of bytes wide

“**.DW**” == Define Word (2 bytes)

Tables and Static Strings in Program Memory

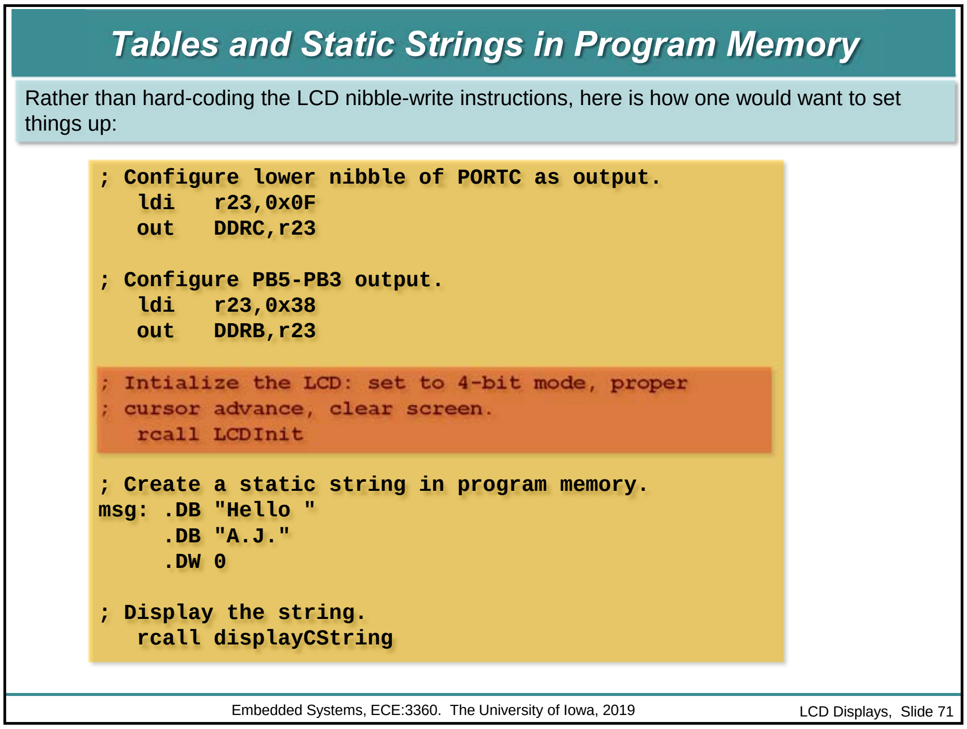
Rather than hard-coding the LCD nibble-write instructions, here is how one would want to set things up:

```
; Configure lower nibble of PORTC as output.
   ldi   r23,0x0F
   out   DDRC,r23

; Configure PB5-PB3 output.
   ldi   r23,0x38
   out   DDRB,r23

; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0

; Display the string.
   rcall displayCString
```

Tables and Static Strings in Program Memory

Rather than hard-coding the LCD nibble-write instructions, here is how one would want to set things up:

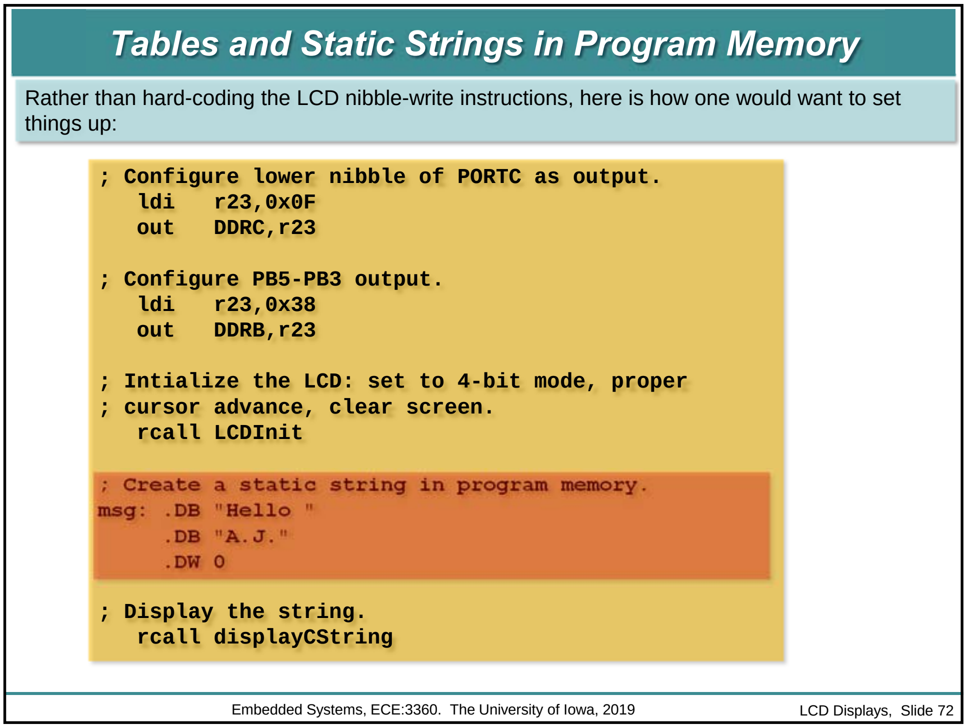
```
; Configure lower nibble of PORTC as output.
   ldi   r23,0x0F
   out   DDRC,r23

; Configure PB5-PB3 output.
   ldi   r23,0x38
   out   DDRB,r23

; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0

; Display the string.
   rcall displayCString
```



Tables and Static Strings in Program Memory

Rather than hard-coding the LCD nibble-write instructions, here is how one would want to set things up:

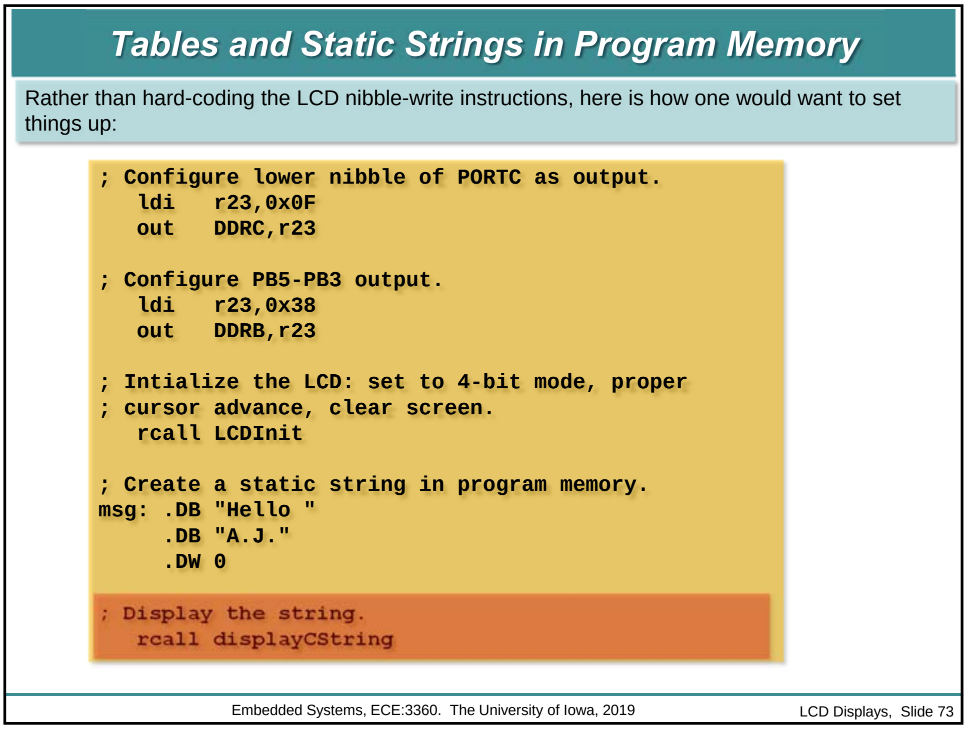
```
; Configure lower nibble of PORTC as output.
   ldi   r23,0x0F
   out   DDRC,r23

; Configure PB5-PB3 output.
   ldi   r23,0x38
   out   DDRB,r23

; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0

; Display the string.
   rcall displayCString
```



Tables and Static Strings in Program Memory

Rather than hard-coding the LCD nibble-write instructions, here is how one would want to set things up:

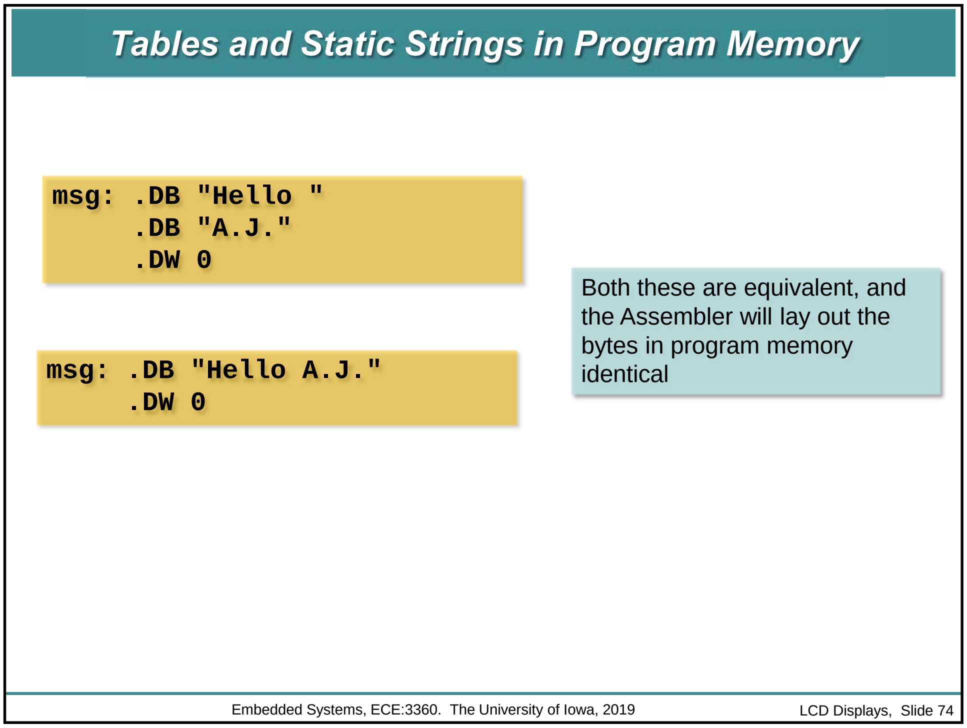
```
; Configure lower nibble of PORTC as output.
   ldi   r23,0x0F
   out   DDRC,r23

; Configure PB5-PB3 output.
   ldi   r23,0x38
   out   DDRB,r23

; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0

; Display the string.
   rcall displayCString
```



Tables and Static Strings in Program Memory

Rather than hard-coding the LCD nibble-write instructions, here is how one would want to set things up:

```
; Configure lower nibble of PORTC as output.
   ldi   r23,0x0F
   out   DDRC,r23

; Configure PB5-PB3 output.
   ldi   r23,0x38
   out   DDRB,r23

; Intialize the LCD: set to 4-bit mode, proper
; cursor advance, clear screen.
   rcall LCDInit

; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0

; Display the string.
   rcall displayCString
```



Tables and Static Strings in Program Memory

```
msg: .DB "Hello "
     .DB "A.J."
     .DW 0
```

```
msg: .DB "Hello A.J."
     .DW 0
```

Both these are equivalent, and the Assembler will lay out the bytes in program memory identical

Tables and Static Strings in Program Memory

```
msg:  .DB "Hi A.J."
```

This will generate a warning, since there are an odd number of bytes.

The assembler will pad (add) a zero byte at the end to get 16-bit alignment

```
LCD01.asm(30): warning: .cseg .db misalignment - padding zero byte
```

Tables and Static Strings in Program Memory

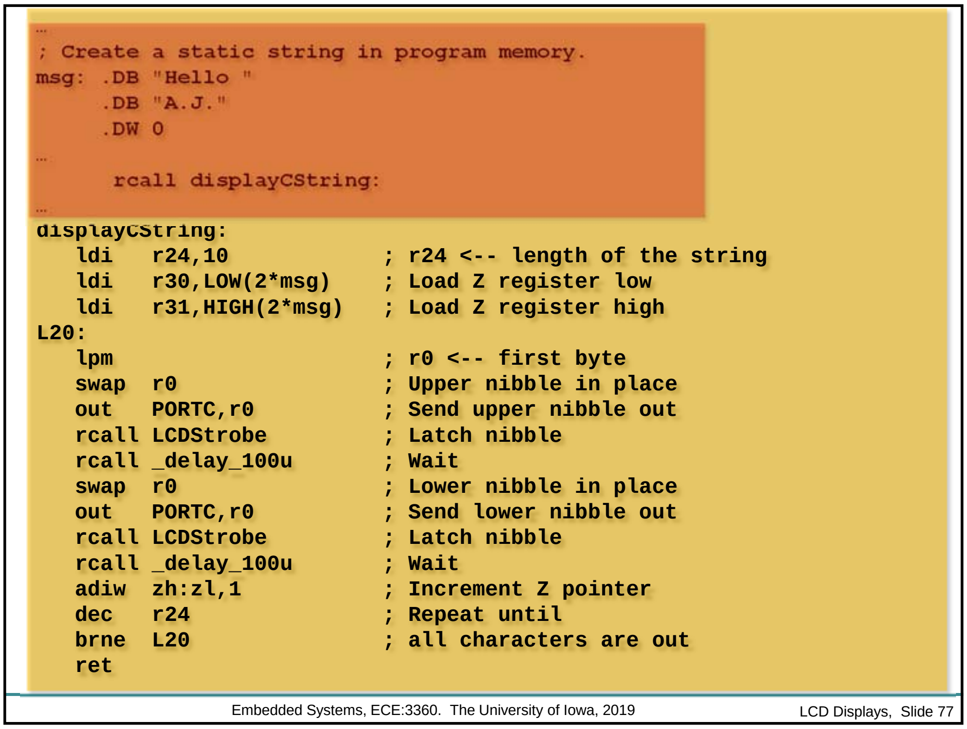
The **lpm** (load program memory) instruction is used for accessing tables and strings from program (Flash) memory.

First, one loads the address on the string/table into the **Z** pointer register. Recall, the **R30** and **R31** register pair forms the **Z** pointer register.

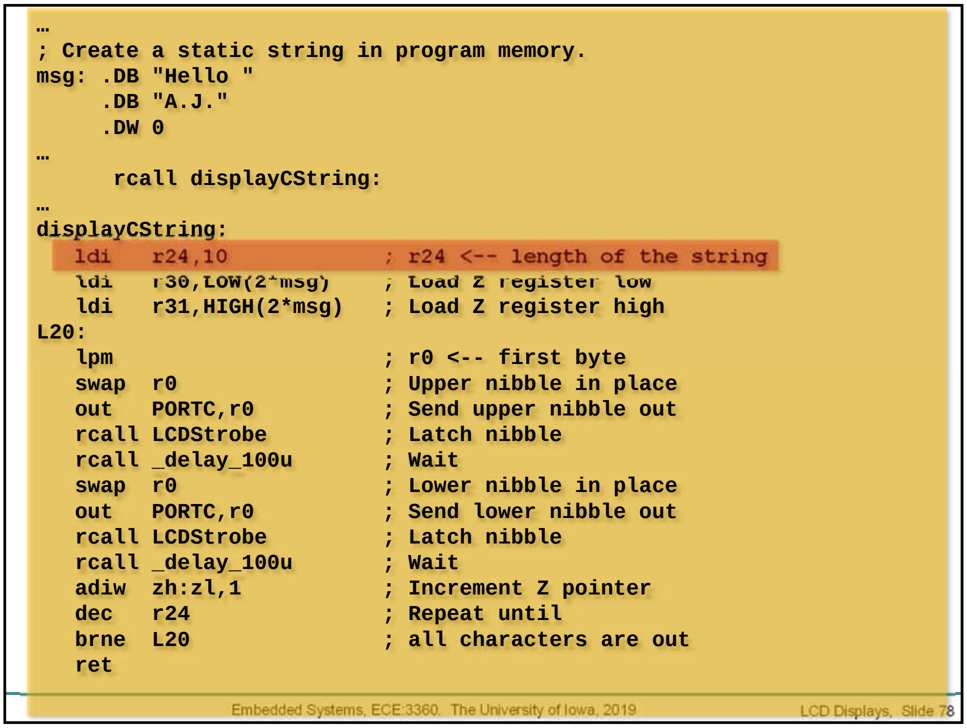
Next, execute the **lpm** instruction. This copies what **Z** register points to, into **R0**.

To load another byte, increment the **Z** register and repeat.

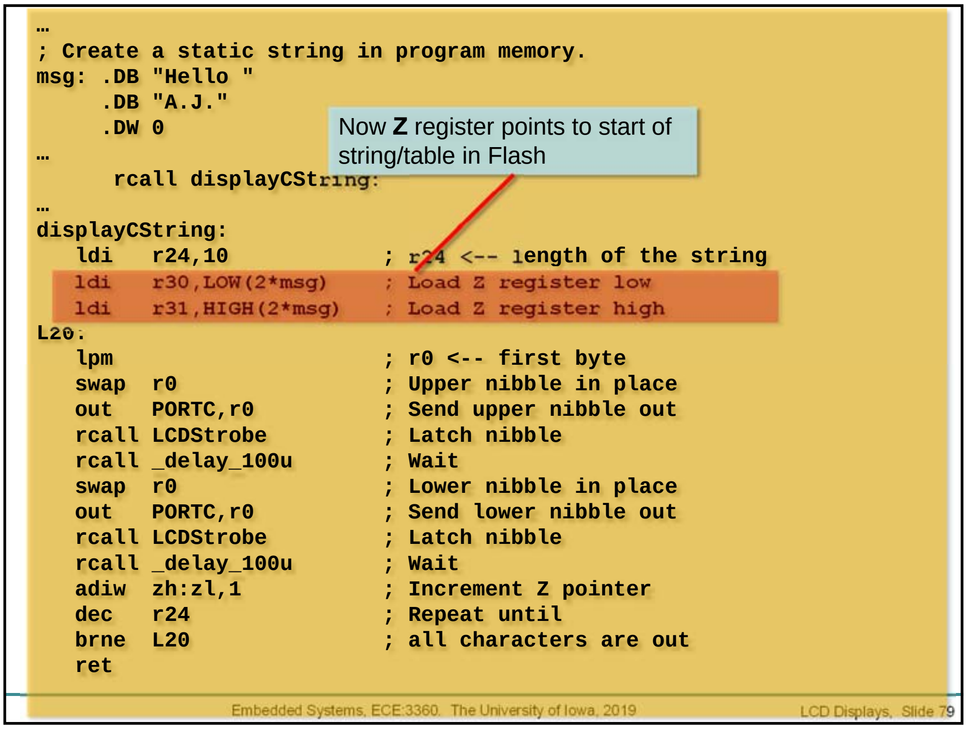
```
msg:  .DB "Hello "
   …
   ldi    r30,LOW(2*msg)     ; Load Z register low
   ldi    r31,HIGH(2*msg)    ; Load Z register high
 ->lpm                       ; r0 <-- load byte
   …
   adiw   zh:zl,1            ; Increment Z pointer
```



```
…
; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0
…
      rcall displayCString:
…
displayCString:
   ldi   r24,10            ; r24 <-- length of the string
   ldi   r30,LOW(2*msg)    ; Load Z register low
   ldi   r31,HIGH(2*msg)   ; Load Z register high
L20:
   lpm                     ; r0 <-- first byte
   swap  r0                ; Upper nibble in place
   out   PORTC,r0          ; Send upper nibble out
   rcall LCDStrobe         ; Latch nibble
   rcall _delay_100u       ; Wait
   swap  r0                ; Lower nibble in place
   out   PORTC,r0          ; Send lower nibble out
   rcall LCDStrobe         ; Latch nibble
   rcall _delay_100u       ; Wait
   adiw  zh:zl,1           ; Increment Z pointer
   dec   r24               ; Repeat until
   brne  L20               ; all characters are out
   ret
```

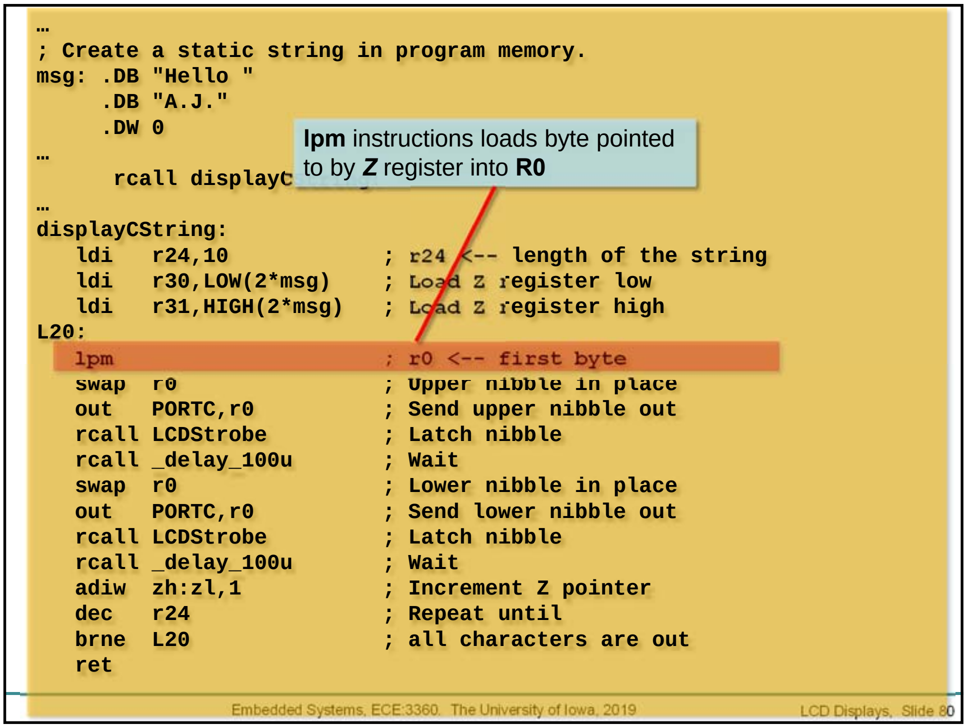


```
…
; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0
…
      rcall displayCString:
…
displayCString:
   ldi   r24,10            ; r24 <-- length of the string
   ldi   r30,LOW(2*msg)    ; Load Z register low
   ldi   r31,HIGH(2*msg)   ; Load Z register high
L20:
   lpm                     ; r0 <-- first byte
   swap  r0                ; Upper nibble in place
   out   PORTC,r0          ; Send upper nibble out
   rcall LCDStrobe         ; Latch nibble
   rcall _delay_100u       ; Wait
   swap  r0                ; Lower nibble in place
   out   PORTC,r0          ; Send lower nibble out
   rcall LCDStrobe         ; Latch nibble
   rcall _delay_100u       ; Wait
   adiw  zh:zl,1           ; Increment Z pointer
   dec   r24               ; Repeat until
   brne  L20               ; all characters are out
   ret
```



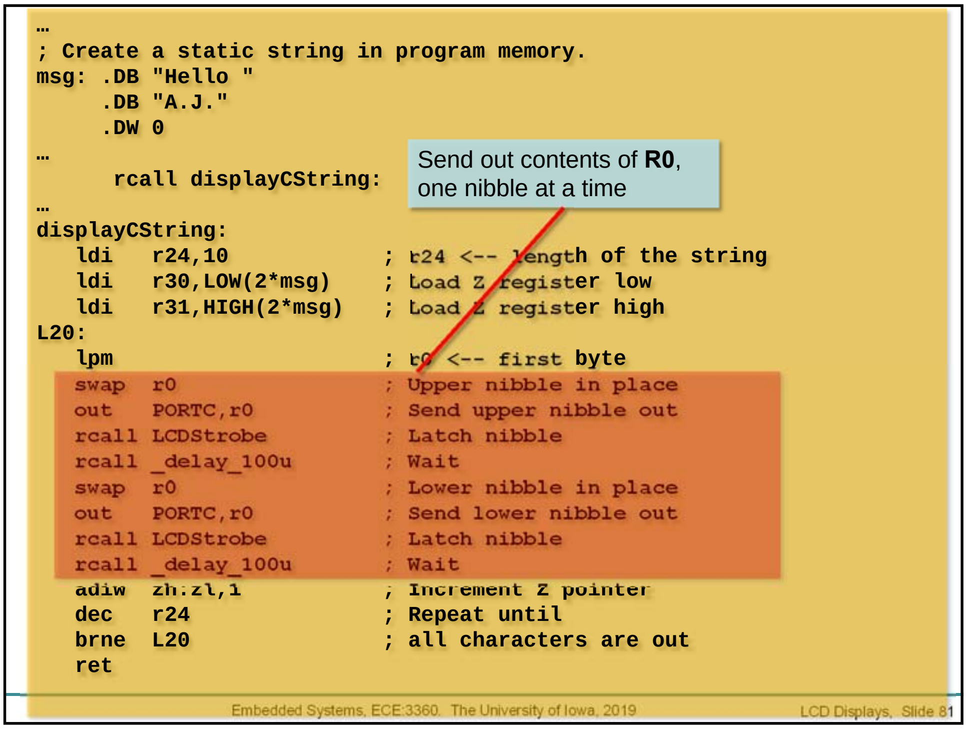
```
…
; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0
…
      rcall displayCString:
…
displayCString:
   ldi   r24,10              ; r24 <-- length of the string
   ldi   r30,LOW(2*msg)      ; Load Z register low
   ldi   r31,HIGH(2*msg)     ; Load Z register high
L20:
   lpm                       ; r0 <-- first byte
   swap  r0                  ; Upper nibble in place
   out   PORTC,r0            ; Send upper nibble out
   rcall LCDStrobe           ; Latch nibble
   rcall _delay_100u         ; Wait
   swap  r0                  ; Lower nibble in place
   out   PORTC,r0            ; Send lower nibble out
   rcall LCDStrobe           ; Latch nibble
   rcall _delay_100u         ; Wait
   adiw  zh:zl,1             ; Increment Z pointer
   dec   r24                 ; Repeat until
   brne  L20                 ; all characters are out
   ret
```

Now ***Z*** register points to start of string/table in Flash



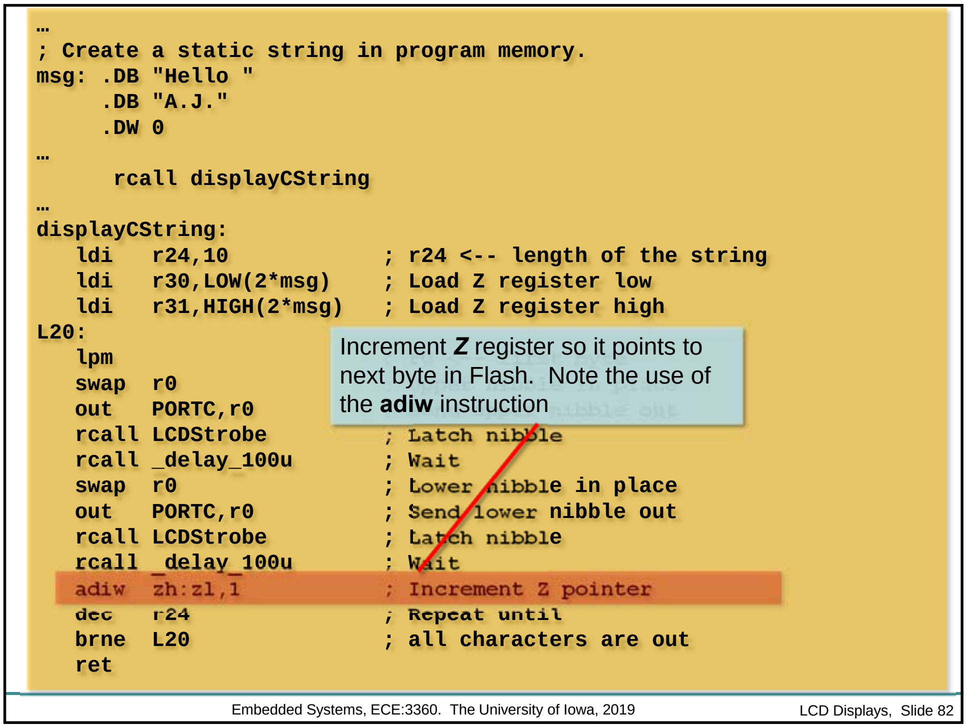
```
…
; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0
…
      rcall displayC
…
displayCString:
   ldi   r24,10              ; r24 <-- length of the string
   ldi   r30,LOW(2*msg)      ; Load Z register low
   ldi   r31,HIGH(2*msg)     ; Load Z register high
L20:
   lpm                       ; r0 <-- first byte
   swap  r0                  ; Upper nibble in place
   out   PORTC,r0            ; Send upper nibble out
   rcall LCDStrobe           ; Latch nibble
   rcall _delay_100u         ; Wait
   swap  r0                  ; Lower nibble in place
   out   PORTC,r0            ; Send lower nibble out
   rcall LCDStrobe           ; Latch nibble
   rcall _delay_100u         ; Wait
   adiw  zh:zl,1             ; Increment Z pointer
   dec   r24                 ; Repeat until
   brne  L20                 ; all characters are out
   ret
```

lpm instructions loads byte pointed to by ***Z*** register into **R0**



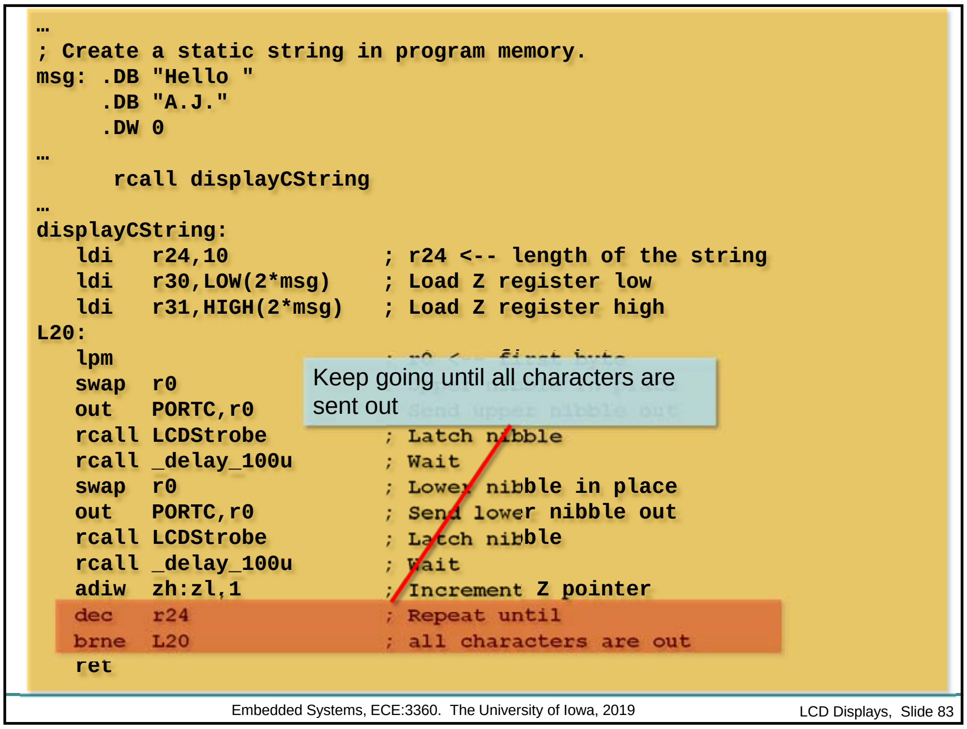
```
…
; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0
…
      rcall displayCString:
…
displayCString:
   ldi    r24,10             ; r24 <-- length of the string
   ldi    r30,LOW(2*msg)     ; Load Z register low
   ldi    r31,HIGH(2*msg)    ; Load Z register high
L20:
   lpm                       ; r0 <-- first byte
   swap   r0                 ; Upper nibble in place
   out    PORTC,r0           ; Send upper nibble out
   rcall  LCDStrobe          ; Latch nibble
   rcall  _delay_100u        ; Wait
   swap   r0                 ; Lower nibble in place
   out    PORTC,r0           ; Send lower nibble out
   rcall  LCDStrobe          ; Latch nibble
   rcall  _delay_100u        ; Wait
   adiw   zh:zl,1            ; Increment Z pointer
   dec    r24                ; Repeat until
   brne   L20                ; all characters are out
   ret
```

Send out contents of **R0**, one nibble at a time



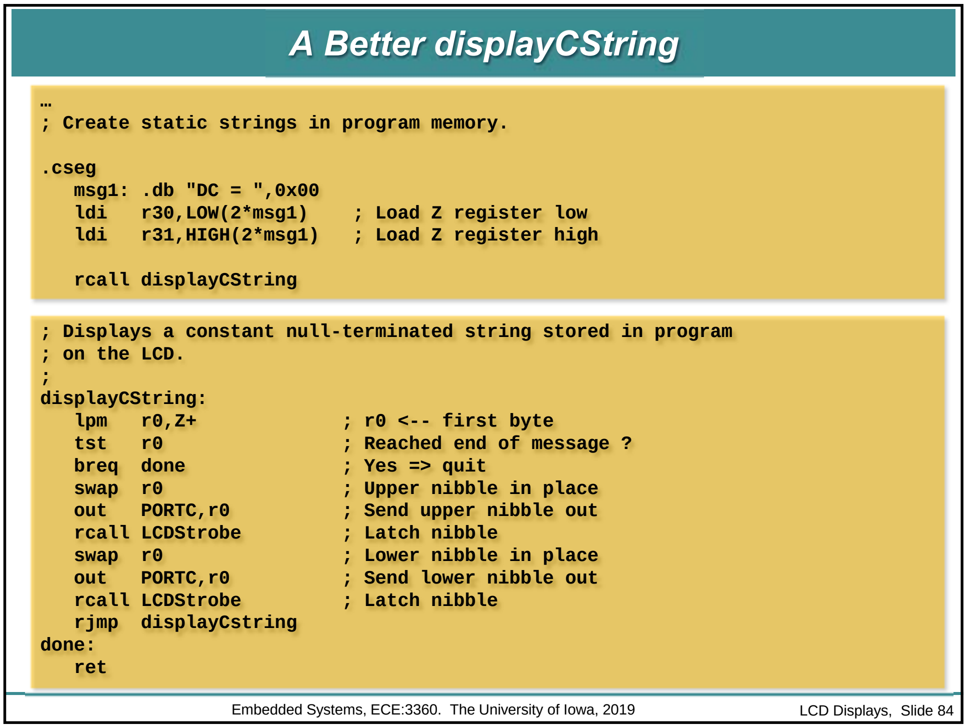
```
…
; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0
…
      rcall displayCString
…
displayCString:
   ldi    r24,10             ; r24 <-- length of the string
   ldi    r30,LOW(2*msg)     ; Load Z register low
   ldi    r31,HIGH(2*msg)    ; Load Z register high
L20:
   lpm
   swap   r0
   out    PORTC,r0
   rcall  LCDStrobe          ; Latch nibble
   rcall  _delay_100u        ; Wait
   swap   r0                 ; Lower nibble in place
   out    PORTC,r0           ; Send lower nibble out
   rcall  LCDStrobe          ; Latch nibble
   rcall  _delay_100u        ; Wait
   adiw   zh:zl,1            ; Increment Z pointer
   dec    r24                ; Repeat until
   brne   L20                ; all characters are out
   ret
```

Increment ***Z*** register so it points to next byte in Flash. Note the use of the **adiw** instruction



```
…
; Create a static string in program memory.
msg: .DB "Hello "
     .DB "A.J."
     .DW 0
…
      rcall displayCString
…
displayCString:
   ldi   r24,10             ; r24 <-- length of the string
   ldi   r30,LOW(2*msg)     ; Load Z register low
   ldi   r31,HIGH(2*msg)    ; Load Z register high
L20:
   lpm                      ; r0 <-- first byte
   swap  r0
   out   PORTC,r0             Send upper nibble out
   rcall LCDStrobe          ; Latch nibble
   rcall _delay_100u        ; Wait
   swap  r0                 ; Lower nibble in place
   out   PORTC,r0           ; Send lower nibble out
   rcall LCDStrobe          ; Latch nibble
   rcall _delay_100u        ; Wait
   adiw  zh:zl,1            ; Increment Z pointer
   dec   r24                ; Repeat until
   brne  L20                ; all characters are out
   ret
```

Keep going until all characters are sent out



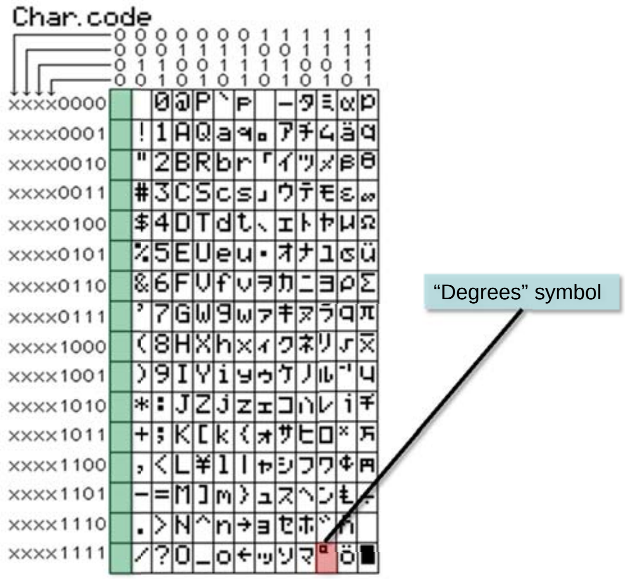
A Better displayCString

```
…
; Create static strings in program memory.

.cseg
   msg1: .db "DC = ",0x00
   ldi   r30,LOW(2*msg1)    ; Load Z register low
   ldi   r31,HIGH(2*msg1)   ; Load Z register high

   rcall displayCString
```

```
; Displays a constant null-terminated string stored in program
; on the LCD.
;
displayCString:
   lpm   r0,Z+             ; r0 <-- first byte
   tst   r0                ; Reached end of message ?
   breq  done              ; Yes => quit
   swap  r0                ; Upper nibble in place
   out   PORTC,r0          ; Send upper nibble out
   rcall LCDStrobe         ; Latch nibble
   swap  r0                ; Lower nibble in place
   out   PORTC,r0          ; Send lower nibble out
   rcall LCDStrobe         ; Latch nibble
   rjmp  displayCstring
done:
   ret
```

Character Codes & User-Defined Characters

HD44780 Character Codes



Char.code

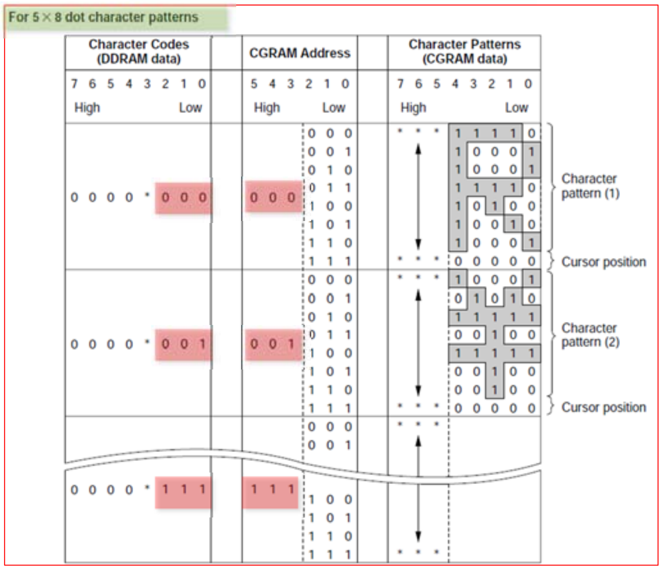
	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
xxxx0000			0	@	P	`	p		ー	タ	ミ	α	p
xxxx0001		!	1	A	Q	a	q	。	ア	チ	ム	ä	q
xxxx0010		"	2	B	R	b	r	「	イ	ツ	メ	β	θ
xxxx0011		#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
xxxx0100		$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
xxxx0101		%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
xxxx0110		&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111		’	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
xxxx1000		(	8	H	X	h	x	ィ	ク	ネ	リ	√	x̄
xxxx1001		)	9	I	Y	i	y	ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010		*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
xxxx1011		+	;	K	[	k	{	ォ	サ	ヒ	ロ	ˣ	万
xxxx1100		,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
xxxx1101		-	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
xxxx1110		.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
xxxx1111		/	?	O	_	o	←	ッ	ソ	マ	゜	ö	█

“Degrees” symbol

User-defined characters

- HD44780 supports up to 8 user-defined characters
- Character codes 0x00 – 0x07 or 0x08 – 0x0F
- Before use, the characters must be defined by storing the pixel pattern in a character-generating RAM (CGRAM) on the controller chip

CGRAM Address Map



For 5 × 8 dot character patterns

Character Codes (DDRAM data)	CGRAM Address		Character Patterns (CGRAM data)		
7 6 5 4 3 2 1 0	5 4 3	2 1 0	7 6 5	4 3 2 1 0	
High Low	High	Low	High	Low	
0 0 0 0 * 0 0 0	0 0 0	0 0 0	* * *	1 1 1 1 0	Character pattern (1)
		0 0 1		1 0 0 0 1	
		0 1 0		1 0 0 0 1	
		0 1 1		1 1 1 1 0	
		1 0 0		1 0 1 0 0	
		1 0 1		1 0 0 1 0	
		1 1 0		1 0 0 0 1	
		1 1 1	* * *	0 0 0 0 0	Cursor position
0 0 0 0 * 0 0 1	0 0 1	0 0 0	* * *	1 0 0 0 1	Character pattern (2)
		0 0 1		0 1 0 1 0	
		0 1 0		1 1 1 1 1	
		0 1 1		0 0 1 0 0	
		1 0 0		1 1 1 1 1	
		1 0 1		0 0 1 0 0	
		1 1 0		0 0 1 0 0	
		1 1 1	* * *	0 0 0 0 0	Cursor position
		0 0 0	* * *		
		0 0 1			
0 0 0 0 * 1 1 1	1 1 1	1 0 0			
		1 0 1			
		1 1 0			
		1 1 1	* * *		

Writing to the CGRAM requires using the **Set CGRAM Address** command (01XX XXXX)

LCD - Checklist

- **Connect LCD to µC**
- **Test LCD setup by downloading test program from ES website**
- **Implement software (subroutines) for:**
  - **Performing LCD initialization sequence (clear, set mode, …)**
  - **Displaying strings**
- **Integrate LCD subroutines with other software …**

- **Important: must meet all timing requirements!**
  - **Note: longer delays could simplify software design**

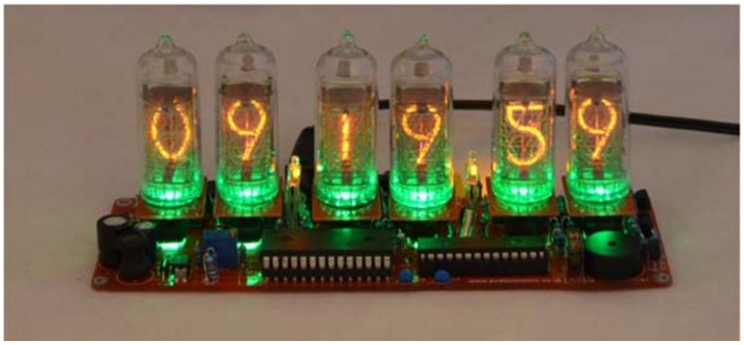
Resources

"Google" LCD Simulator

DigiKey Electronics - Electronic Components Distributor
http://www.digikey.com

SparkFun Electronics
https://www.sparkfun.com/

…



… EOL