uC Programming w/C II

ECE 3710

I used to have an open mind but my brains kept falling out.

- Steven Wright

configuring the uC in C

create vars situated at registers for GPIO Port D:

```
volatile int PD_DATA_R __attribute__((at(0x400073FC)));
volatile int PD_DIR_R __attribute__((at(0x40007400)));
volatile int PD_AF_R __attribute__((at(0x40007420)));
volatile int PD_DEN_R __attribute__((at(0x4000751C)));
volatile int RCGC2_R __attribute__((at(0x400FE108)));
```

`volatile`: use when uC might change contents of addr on you; e.g. when doing memory mapped I/O

(otherwise compiler might 'optimise' away functionality [like polling])

configuration using offsets

how to set initial addr of
what ptr points to:

```
unsigned char *d = (unsigned char *) 0x20000000;
```

use like C-array:

```
d[0x12]=0xAB;
```

points to
0x20000000+0x12
=0x20000012

memory(0x20000012)
=0xAB

configuration using offsets

tedious to write word

slight downside:

1. byte addressable (char for ptr)
2. addressing words (int for ptr)

indexing is off

example: configuration using offsets

final value of word at byte offset
0x60 should be 0x078E3B82:

```
//1. unsigned char *SYSCTL = (unsigned char *) 0x400FE000;
SYSCTL[0x60] = 0x82;
SYSCTL[0x61] = 0x3B;
SYSCTL[0x62] = 0x8E;
SYSCTL[0x63] = 0x07;
//2. unsigned int *SYSCTL = (unsigned int *) 0x400FE000;
SYSCTL[0x60/4] = 0x078E3B82;
```

ex: continuous serial tx of 0--255

1. configuration vars (global)

```
// system control base addr
unsigned char *SYSCTL = (unsigned char *) 0x400FE000;
//UART0 shares pins with Port A (RX: PA0, TX: PA1)
unsigned char *PA = (unsigned char *) 0x40004000;
// UART0 base
unsigned char *UART0 = (unsigned char *) 0x4000C000;
// var we write to for TX
unsigned char UART0_D __attribute__((at(0x4000C000)));
// var to poll while waiting for TX to finish
volatile unsigned int UART0_STAT __attribute__((at(0x4000C018)));
```

ex: continuous serial tx of 0--255

2. initialisation function

```
void UART0Init()
{
    // 0. set sysclk to main osc (8MHz crystal)
    // final value should be 0x078E3B82 (remember little endian)
  SYSCTL[0x60] = 0x82;
  SYSCTL[0x61] = 0x3B;
  SYSCTL[0x62] = 0x8E;
  SYSCTL[0x63] = 0x07;

  // 1. enable clock: uart then port
  SYSCTL[0x104] = 0x1; //uart0
  SYSCTL[0x108] = 0x1; //portA

  // 2. PA1: enable alt. func. and pin
  PA[0x420] = 0x2; //AFSEL
  PA[0x51C] = 0x2; //DEN

  // 3. disable uart0
  UART0[0x30] = 0x0;
...
```

ex: continuous serial tx of 0--255

2. initialisation function

```
void UART0Init()
{
...
  // 4. set baudrate divisor
  // BRD = 8e6/(16*9600)= 52.083
  // integer portion: int(52.083)=52
  UART0[0x24] = 0x34;
  // fractional portion: int(0.083*2^6+0.5)=6
  UART0[0x28] = 0x6;

  // 5. set serial parameters: 8-bit word, start/stop/parity bits
  UART0[0x2C] = 0x62;

  // 6. enable tx and uart
  UART0[0x30] = 0x01;
  UART0[0x31] = 0x1;
}
```

ex: continuous serial tx of 0--255

3. main

```
int main(void)
{
   unsigned char d;

   UART0Init();

   for(d=0;d<=255;d++)
   {
      UART0_D = d;
      while(UART0_STAT == 0x30);
   }
}
```

LCD Interfacing I

ECE 3710

w/o controller

LCDs considered:

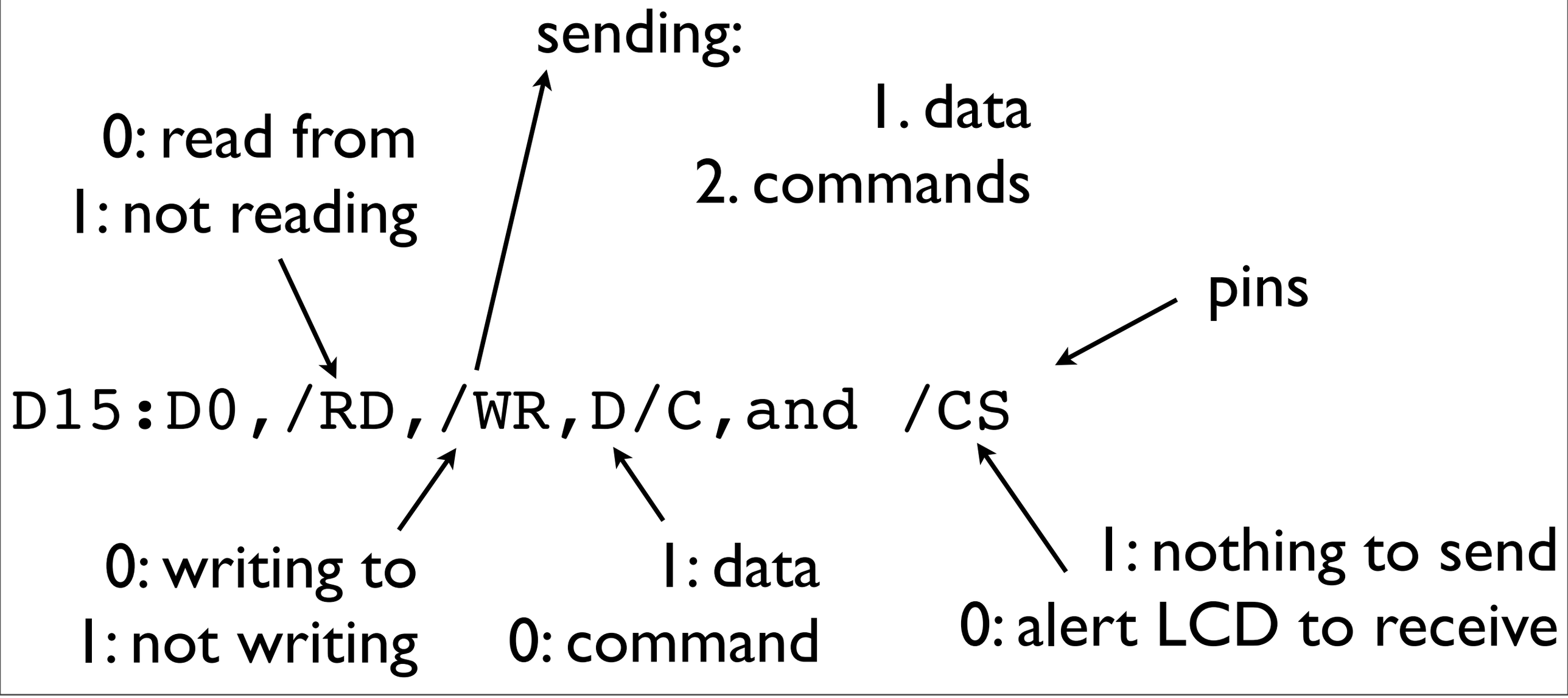
1. book

2. lab

w/ controller

guess which one is easier?

when communicating with LCD: Parallel



how to think about LCD: Parallel

(week one)

p70

to send to it:

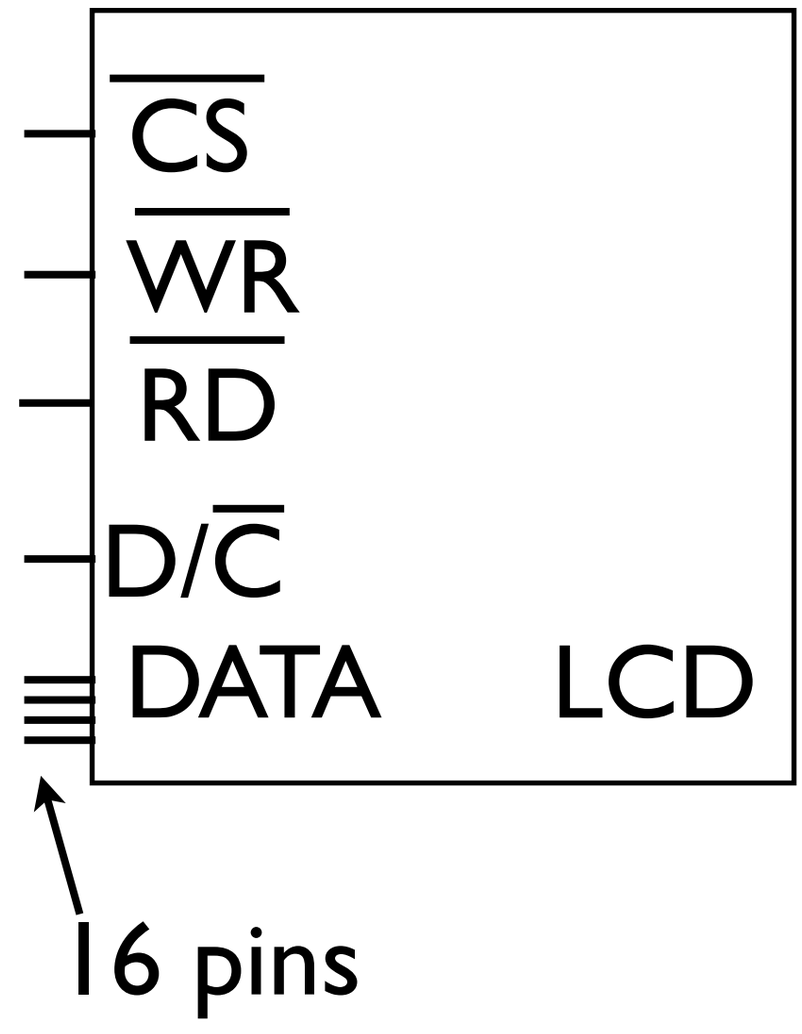
1. CS=0
2. WR=0
3. RD=1
4. D/C=0 command
   D/C=1 data

week one:

write C functions to send data and commands via parallel (GPIO)

a device with two chunks of memory

(chunk 1: D/C=0
chunk 2: D/C=1)



lcd lab:

some additional instruction required, else...

trying to figure out the datasheet w/o guidance:

DOOM!

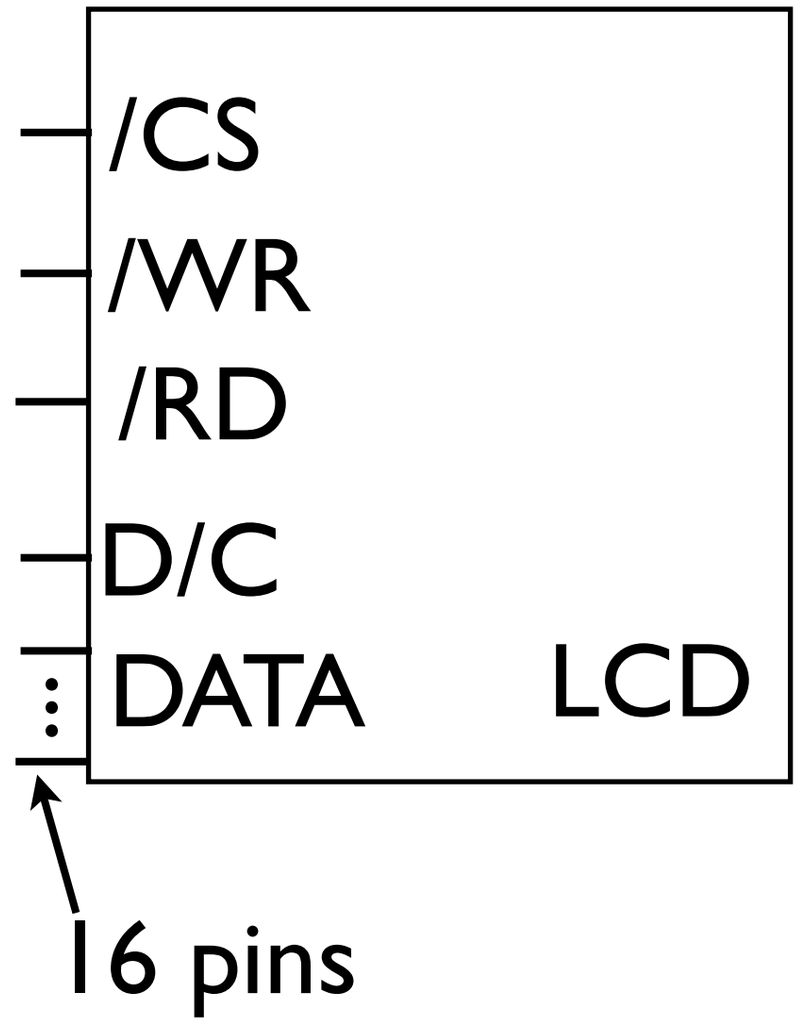
week one assistance:

1. header file with function definitions (you write them)
2. configuration to initialise LCD

how to think about LCD: Parallel

a device with two kinds of memory (location+data):

1. configuration
2. graphic/display



Q: how do we let this thing know what bits on DATA pins mean?

note: '/' denotes active-low
(to set line should be logic low)

how we write

two write functions:

1. *command*: which memory we write to and where
2. *data*: what is written

to interface with LCD

two step process (both writes):

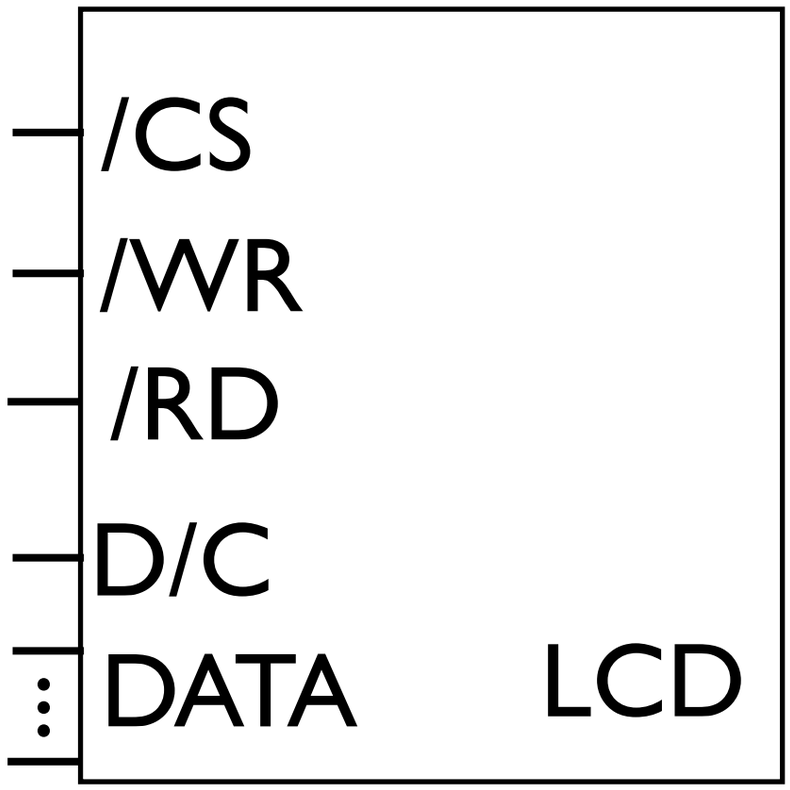
1. which memory location to write to (command)
2. what location should be set to (data)

`write*`

both send bits to LCD via DATA pins

to distinguish command from data:

1. `writeCmd(unsigned short cmd)`
2. `writeDat(unsigned short dat)`



we set:

1: D/C = 0

(DATA=`cmd`)

2: D/C = 1

(DATA=`dat`)

what is written

technical term

writes to:

1. configuration memory: *config stuff*
2. graphics data: *pixels*

remember: writeCmd and writeDat are needed in each case

Q: how do we make this happen?

1. LCD configuration: easy

to initialise LCD:

```
LCD_WriteReg(0x0000,0x0001);    delay_ms(50);   /* Enable LCD Oscillator */
LCD_WriteReg(0x0003,0xA8A4);    delay_ms(50);   // Power control(1)
LCD_WriteReg(0x000C,0x0000);    delay_ms(50);   // Power control(2)
LCD_WriteReg(0x000D,0x080C);    delay_ms(50);   // Power control(3)
LCD_WriteReg(0x000E,0x2B00);    delay_ms(50);   // Power control(4)
LCD_WriteReg(0x001E,0x00B0);    delay_ms(50);   // Power control(5)
LCD_WriteReg(0x0001,0x2B3F);    delay_ms(50);   // Driver Output Control /* 320*240 0x2B3F */
LCD_WriteReg(0x0002,0x0600);    delay_ms(50);   // LCD Drive AC Control
LCD_WriteReg(0x0010,0x0000);    delay_ms(50);   // Sleep Mode off
LCD_WriteReg(0x0011,0x6070);    delay_ms(50);   // Entry Mode
LCD_WriteReg(0x0005,0x0000);    delay_ms(50);   // Compare register(1)
LCD_WriteReg(0x0006,0x0000);    delay_ms(50);   // Compare register(2)
LCD_WriteReg(0x0016,0xEF1C);    delay_ms(50);   // Horizontal Porch
LCD_WriteReg(0x0017,0x0003);    delay_ms(50);   // Vertical Porch
LCD_WriteReg(0x0007,0x0133);    delay_ms(50);   // Display Control
LCD_WriteReg(0x000B,0x0000);    delay_ms(50);   // Frame Cycle control
LCD_WriteReg(0x000F,0x0000);    delay_ms(50);   // Gate scan start position
LCD_WriteReg(0x0041,0x0000);    delay_ms(50);   // Vertical scroll control(1)
LCD_WriteReg(0x0042,0x0000);    delay_ms(50);   // Vertical scroll control(2)
LCD_WriteReg(0x0048,0x0000);    delay_ms(50);   // First window start
LCD_WriteReg(0x0049,0x013F);    delay_ms(50);   // First window end
LCD_WriteReg(0x004A,0x0000);    delay_ms(50);   // Second window start
LCD_WriteReg(0x004B,0x0000);    delay_ms(50);   // Second window end
LCD_WriteReg(0x0044,0xEF00);    delay_ms(50);   // Horizontal RAM address position
LCD_WriteReg(0x0045,0x0000);    delay_ms(50);   // Vertical RAM address start position
LCD_WriteReg(0x0046,0x013F);    delay_ms(50);   // Vertical RAM address end position
LCD_WriteReg(0x0030,0x0707);    delay_ms(50);   // gamma control(1)
LCD_WriteReg(0x0031,0x0204);    delay_ms(50);   // gamma control(2)
LCD_WriteReg(0x0032,0x0204);    delay_ms(50);   // gamma control(3)
LCD_WriteReg(0x0033,0x0502);    delay_ms(50);   // gamma control(4)
LCD_WriteReg(0x0034,0x0507);    delay_ms(50);   // gamma control(5)
LCD_WriteReg(0x0035,0x0204);    delay_ms(50);   // gamma control(6)
LCD_WriteReg(0x0036,0x0204);    delay_ms(50);   // gamma control(7)
LCD_WriteReg(0x0037,0x0502);    delay_ms(50);   // gamma control(8)
LCD_WriteReg(0x003A,0x0302);    delay_ms(50);   // gamma control(9)
LCD_WriteReg(0x003B,0x0302);    delay_ms(50);   // gamma control(10)
LCD_WriteReg(0x0023,0x0000);    delay_ms(50);   // RAM write data mask(1)
LCD_WriteReg(0x0024,0x0000);    delay_ms(50);   // RAM write data mask(2)
LCD_WriteReg(0x0025,0x8000);    delay_ms(50);   // Frame Frequency
LCD_WriteReg(0x004f,0);                         // Set GDDRAM Y address counter
LCD_WriteReg(0x004e,0);                         // Set GDDRAM X address counter
```

at the thought of
coming up with that, I quit...

1. LCD initialisation: let someone else figure it out

```
LCD_WriteReg(0x0000,0x0001);    delay_ms(50);   /* Enable LCD Oscillator */
LCD_WriteReg(0x0003,0xA8A4);    delay_ms(50);   // Power control(1)
LCD_WriteReg(0x000C,0x0000);    delay_ms(50);   // Power control(2)
LCD_WriteReg(0x000D,0x080C);    delay_ms(50);   // Power control(3)
LCD_WriteReg(0x000E,0x2B00);    delay_ms(50);   // Power control(4)
LCD_WriteReg(0x001E,0x00B0);    delay_ms(50);   // Power control(5)
LCD_WriteReg(0x0001,0x2B3F);    delay_ms(50);   // Driver Output Control /* 320*240 0x2B3F */
LCD_WriteReg(0x0002,0x0600);    delay_ms(50);   // LCD Drive AC Control
LCD_WriteReg(0x0010,0x0000);    delay_ms(50);   // Sleep Mode off
LCD_WriteReg(0x0011,0x6070);    delay_ms(50);   // Entry Mode
LCD_WriteReg(0x0005,0x0000);    delay_ms(50);   // Compare register(1)
LCD_WriteReg(0x0006,0x0000);    delay_ms(50);   // Compare register(2)
LCD_WriteReg(0x0016,0xEF1C);    delay_ms(50);   // Horizontal Porch
LCD_WriteReg(0x0017,0x0003);    delay_ms(50);   // Vertical Porch
LCD_WriteReg(0x0007,0x0133);    delay_ms(50);   // Display Control
LCD_WriteReg(0x000B,0x0000);    delay_ms(50);   // Frame Cycle control
LCD_WriteReg(0x000F,0x0000);    delay_ms(50);   // Gate scan start position
LCD_WriteReg(0x0041,0x0000);    delay_ms(50);   // Vertical scroll control(1)
LCD_WriteReg(0x0042,0x0000);    delay_ms(50);   // Vertical scroll control(2)
LCD_WriteReg(0x0048,0x0000);    delay_ms(50);   // First window start
LCD_WriteReg(0x0049,0x013F);    delay_ms(50);   // First window end
LCD_WriteReg(0x004A,0x0000);    delay_ms(50);   // Second window start
LCD_WriteReg(0x004B,0x0000);    delay_ms(50);   // Second window end
LCD_WriteReg(0x0044,0xEF00);    delay_ms(50);   // Horizontal RAM address position
LCD_WriteReg(0x0045,0x0000);    delay_ms(50);   // Vertical RAM address start position
LCD_WriteReg(0x0046,0x013F);    delay_ms(50);   // Vertical RAM address end position
LCD_WriteReg(0x0030,0x0707);    delay_ms(50);   // gamma control(1)
LCD_WriteReg(0x0031,0x0204);    delay_ms(50);   // gamma control(2)
LCD_WriteReg(0x0032,0x0204);    delay_ms(50);   // gamma control(3)
LCD_WriteReg(0x0033,0x0502);    delay_ms(50);   // gamma control(4)
LCD_WriteReg(0x0034,0x0507);    delay_ms(50);   // gamma control(5)
LCD_WriteReg(0x0035,0x0204);    delay_ms(50);   // gamma control(6)
LCD_WriteReg(0x0036,0x0204);    delay_ms(50);   // gamma control(7)
LCD_WriteReg(0x0037,0x0502);    delay_ms(50);   // gamma control(8)
LCD_WriteReg(0x003A,0x0302);    delay_ms(50);   // gamma control(9)
LCD_WriteReg(0x003B,0x0302);    delay_ms(50);   // gamma control(10)
LCD_WriteReg(0x0023,0x0000);    delay_ms(50);   // RAM write data mask(1)
LCD_WriteReg(0x0024,0x0000);    delay_ms(50);   // RAM write data mask(2)
LCD_WriteReg(0x0025,0x8000);    delay_ms(50);   // Frame Frequency
LCD_WriteReg(0x004f,0);                         // Set GDDRAM Y address counter
LCD_WriteReg(0x004e,0);                         // Set GDDRAM X address counter
```

we'll let you have this one...but what does it all mean?

1. LCD: configuration

first we write

(tell LCD want to write at memory location 0x0)

Enable LCD Oscillator:

1. `cmd = 0x0000`
2. `dat = 0x0001`

then we write

(set memory(0x0)=0x1)

how do we know to write this?

1. LCD: configuration

we look here

Section 9 (p27) of SSD1298:

cmd

Oscillator (R00h) (POR = 0000h)

R/W	DC	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0
W	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	OSCEN
POR		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

OSCEN: The oscillator will be turned on when OSCEN = 1, off when OSCEN = 0.

oscillator:

1. on: dat=0x1
2: off: dat=0x0

dat=[IB15:IB0]

1. LCD: configuration

(16 bits)

what should go out on DATA pins (`dat`):

how to send config values

		Hardware pins																	
Interface	**Cycle**	**D17**	**D16**	**D15**	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
18 bits		IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	x	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	x
16 bits		IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8		IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	
9 bits	1st										IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	x
	2nd										IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	x
8 bits	1st										IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	
	2nd										IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	

Remark : x Don't care bits

Not connected pins

DATA[7:0] = IB[7:0]

DATA[15:8] = IB[15:8]

remember: DATA is port/pins on uC

1. LCD: configuration

(16 bits)

replace IB15:IB0 with cmd value

note: cmd follows same form for DATA pins:

		Hardware pins																	
Interface	**Cycle**	**D17**	**D16**	**D15**	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
18 bits		IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	x	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	x
16 bits		IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8		IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	
9 bits	1st										IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	x
	2nd										IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	x
8 bits	1st										IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	
	2nd										IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0	

Remark : x Don't care bits

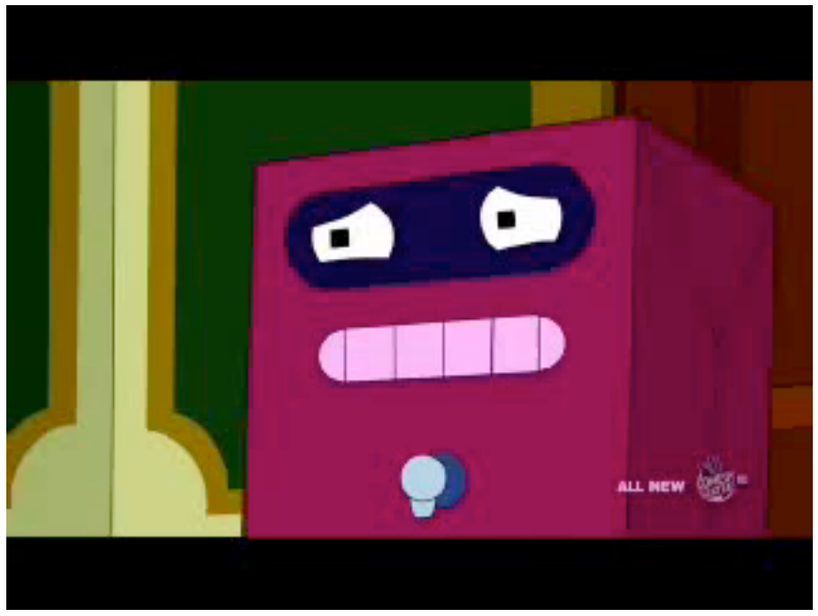
Not connected pins

DATA[7:0] = IB[7:0]

DATA[15:8] = IB[15:8]

remember: DATA is port/pins on uC

tell me graphic
data is easier:



well...probably not worse