LCD Interfacing II

ECE 3710

how we write

two write functions:

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

to interface with LCD

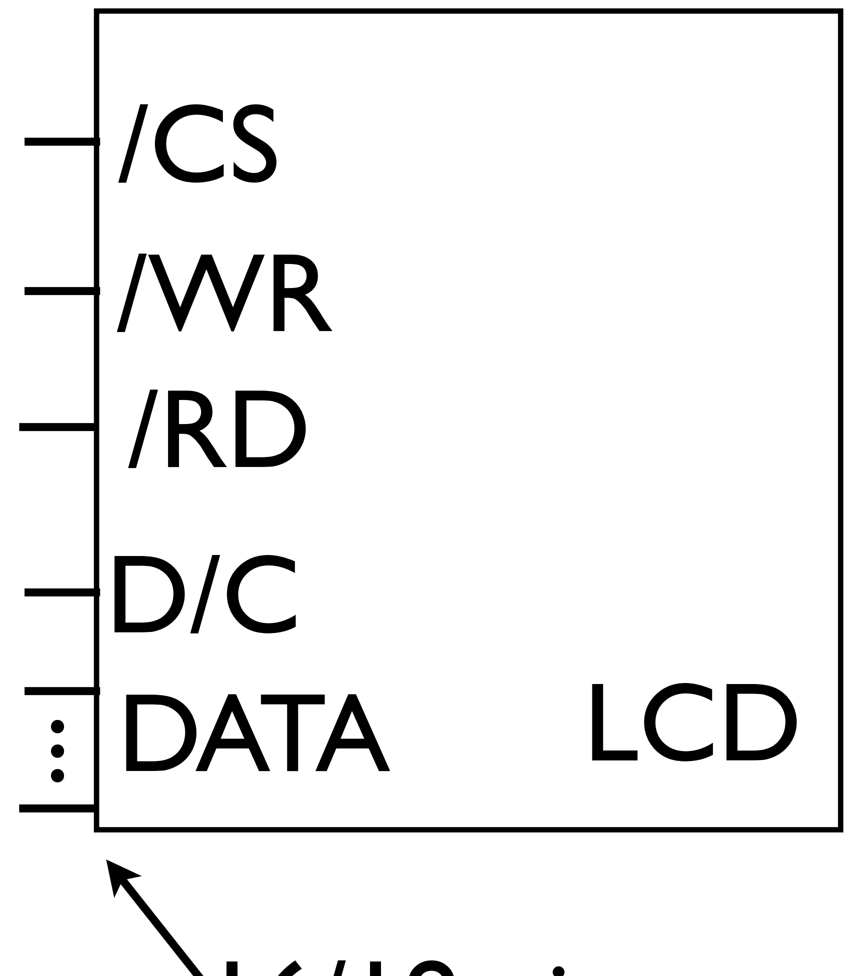
two step process (both writes):

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

how to think about LCD: Parallel

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

1. configuration
2. graphic/display



16/18 pins
(can use fewer)

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)

1. LCD: configuration

(8 bits = 8 lines)

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

(grey) Not connected pins

1st: DATA[7:0] = IB[15:8]
2nd: DATA[7:0] = IB[7:0]

remember: DATA is port/pins on uC
(no need to use D0)

1. LCD: configuration

(8 bits = 8 lines)

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

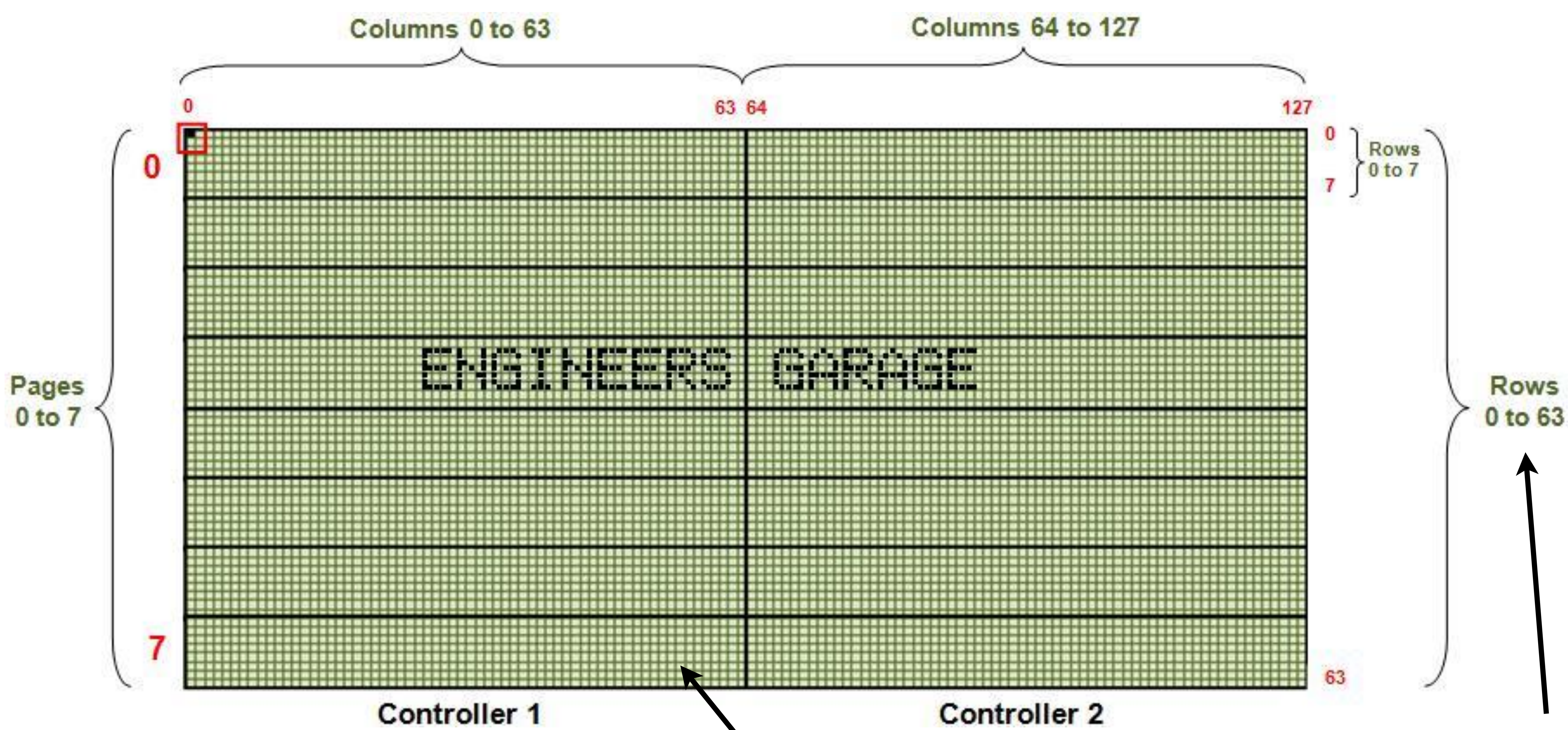
1st: DATA[7:0] = IB[15:8]
2nd: DATA[7:0] = IB[7:0]

remember: DATA is port/pins on uC
(no need to use D0 for our config)

Q: what is an LCD?

A: bunch of RGB pixels

you have 320

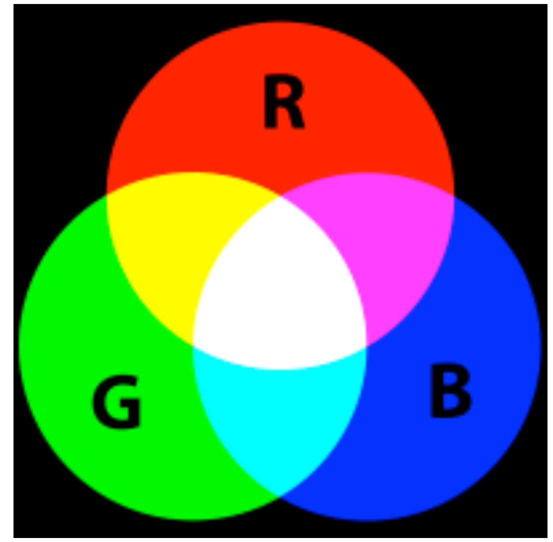


a pixel

you have 240

note: sometimes we divide LCD into columns and controllers to make it updating LCD easier/faster

think back to
elementary school:
any colour composed of



RGB = $\mathrm{R}_{\mathrm{ed}}\mathrm{G}_{\mathrm{reen}}\mathrm{B}_{\mathrm{lue}}$

use numbers to denote shade
of red, green,blue

e.g. R=83, G=133, B=204

8-bits each for of R,G,B => 24 bits
=>2^24 colours

SSD1298: specs

16-bit colour:

R = 5 bits
G = 6 bits
B = 5 Bits
=16 => 2^16=65k colours

resolution:

320 pixels tall
240 pixels wide → 240x320
(horizontal-by-vertical; depending on orientation)

2. LCD: graphics data

set every pixel to `Color`:

```
void LCD_Clear( unsigned short Color )
{
    unsigned int i;

    LCD_SetCursor(0,0);
    ...
    writeCmd(0x0022);
    for( i=0; i< MAX_X*MAX_Y; i++ )
        writeDat(Color);
    ...
}
```

notice: only call `writeCmd` once for multiple `writeDat`

would seem to violate `cmd` for every `dat` principle

procedure:

1. select pixel
2. write RGB value

2. LCD: graphics data

```
void LCD_SetCursor( unsigned short x, unsigned int y )
{
    ...
    LCD_WriteReg(0x004E, x );
    LCD_WriteReg(0x004F, y );
}
```

cmd

dat

where the next write to graphics memory should go

RAM address set (R4Eh-R4Fh)

Reg#	R/W	DC	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0
R4Eh	W	1	0	0	0	0	0	0	0	0	XAD7	XAD6	XAD5	XAD4	XAD3	XAD2	XAD1	XAD0
	POR		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R4Fh	W	1	0	0	0	0	0	0	0	YAD8	YAD7	YAD6	YAD5	YAD4	YAD 3	YAD 2	YAD 1	YAD 0
	POR		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

XAD[7:0]: Make initial settings for the GDDRAM X address in the address counter (AC).
YAD[8:0]: Make initial settings for the GDDRAM Y address in the address counter (AC).

2. LCD: graphics data

graphics memory:

(p68)

y ↓

RL=1	S0	S1	S2	S3	S4	S5	S6	S7	S8	…	S714	S715	S716	S717	S718	S719
RL=0	S719	S718	S717	S716	S715	S714	S713	S712	S711	…	S5	S4	S3	S2	S1	S0
BGR=0	R	G	B	R	G	B	R	G	B	…	R	G	B	R	G	B
BGR=1	B	G	R	B	G	R	B	G	R	…	B	G	R	B	G	R

TB=1	TB=0							Vertical address
G0	G319	0000H,0000H	0000H, 0001H	0000H, 0010H	…	0000H, 00EEH	0000H, 00EFH	0
G1	G318	0001H,0000H	0001H, 0001H	0001H, 0010H	…	0001H, 00EEH	0001H, 00EFH	1
G2	G317	0010H,0000H	0010H, 0001H	0010H, 0010H	…	0010H, 00EEH	0010H, 00EFH	2
G3	G316	0011H,0000H	0011H, 0001H	0011H, 0010H	…	0011H, 00EEH	0011H, 00EFH	3
G4	G315	0100H,0000H	0100H, 0001H	0100H, 0010H	…	0100H, 00EEH	0100H, 00EFH	4
. . .	. . .	. . .			. . .	. . .	. . .	. . .
G316	G3	013CH, 0000H	013CH, 0001H	013CH, 0010H	…	013CH, 00EEH	013CH, 00EFH	316
G317	G2	013DH, 0000H	013DH, 0001H	013DH, 0010H	…	013DH, 00EEH	013DH, 00EFH	317
G318	G1	013EH, 0000H	013EH, 0001H	013EH, 0010H	…	013EH, 00EEH	013EH, 00EFH	318
G319	G0	013FH, 0000H	013FH, 0001H	013FH, 0010H	…	013FH, 00EEH	013FH, 00EFH	319
	Horizontal address	0	1	2	…	238	239	

each is a pixel

Remark : The address is in 00xxH,0yyyH format, where yyy is the vertical address and xx is the horizontal address

x →

2. LCD: graphics data

select pixel(123,210):

```
x:cmd = 0x004E
  dat = 0x007B

y:cmd = 0x004F
  dat = 0x00D2
```

2. LCD: graphics data

now at pixel, how to set RGB?

first issue write to GDATA

```
writeCmd(0x0022);
```

cmd

Write Data to GRAM (R22h)

R/W	DC	D[17:0]
W	1	WD[17:0] mapping depends on the interface setting

WD[17:0]: Transforms all the GDDRAM data into 18-bit, and writes the data. Format for transforming data into 18-bit depends on the interface used. SSD1289 selects the grayscale level according to the GDDRAM data. After writing data to GDDRAM, address is automatically updated according to AM bit and ID bit. Access to GDDRAM during stand-by mode is not available.

```
dat=[D17:D0]
```

then send data

2. LCD: graphics data

(16 bit)

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

how to send RGB values

Interface	Color mode	Cycle	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
			Hardware pins																	
18 bits	262k		R5	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B5	B4	B3	B2	B1	B0
16 bits	262k	1st	R5	R4	R3	R2	R1	R0	x	x		G5	G4	G3	G2	G1	G0	x	x	
		2nd	B5	B4	B3	B2	B1	B0	x	x		R5	R4	R3	R2	R1	R0	x	x	
		3rd	G5	G4	G3	G2	G1	G0	x	x		B5	B4	B3	B2	B1	B0	x	x	
		1st	R5	R4	R3	R2	R1	R0	x	x		G5	G4	G3	G2	G1	G0	x	x	
		2nd	x	x	x	x	x	x	x	x		B5	B4	B3	B2	B1	B0	x	x	
		1st	R5	R4	R3	R2	R1	R0	x	x		G5	G4	G3	G2	G1	G0	x	x	
		2nd	B5	B4	B3	B2	B1	B0	x	x		x	x	x	x	x	x	x	x	
	65k		R4	R3	R2	R1	R0	G5	G4	G3		G2	G1	G0	B4	B3	B2	B1	B0	
9 bits	262k	1st										R5	R4	R3	R2	R1	R0	G5	G4	G3
		2nd										G2	G1	G0	B5	B4	B3	B2	B1	B0
8 bits	262k	1st										R5	R4	R3	R2	R1	R0	x	x	
		2nd										G5	G4	G3	G2	G1	G0	x	x	
		3rd										B5	B4	B3	B2	B1	B0	x	x	
	65k	1st										R4	R3	R2	R1	R0	G5	G4	G3	
		2nd										G2	G1	G0	B4	B3	B2	B1	B0	

Remark : x Don't care bits
Not connected pins

DATA[7:0] = D[8:1]

DATA[15:8] = D[17:10]

still only sending 16 bits as two pins not connected

2. LCD: graphics data

(8 bit)

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

how to send RGB values

			Hardware pins																	
Interface	Color mode	Cycle	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
18 bits	262k		R5	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B5	B4	B3	B2	B1	B0
16 bits	262k	1st	R5	R4	R3	R2	R1	R0	x	x		G5	G4	G3	G2	G1	G0	x	x	
		2nd	B5	B4	B3	B2	B1	B0	x	x		R5	R4	R3	R2	R1	R0	x	x	
		3rd	G5	G4	G3	G2	G1	G0	x	x		B5	B4	B3	B2	B1	B0	x	x	
		1st	R5	R4	R3	R2	R1	R0	x	x		G5	G4	G3	G2	G1	G0	x	x	
		2nd	x	x	x	x	x	x	x	x		B5	B4	B3	B2	B1	B0	x	x	
		1st	R5	R4	R3	R2	R1	R0	x	x		G5	G4	G3	G2	G1	G0	x	x	
		2nd	B5	B4	B3	B2	B1	B0	x	x		x	x	x	x	x	x	x	x	
	65k		R4	R3	R2	R1	R0	G5	G4	G3		G2	G1	G0	B4	B3	B2	B1	B0	
9 bits	262k	1st										R5	R4	R3	R2	R1	R0	G5	G4	G3
		2nd										G2	G1	G0	B5	B4	B3	B2	B1	B0
8 bits	262k	1st										R5	R4	R3	R2	R1	R0	x	x	
		2nd										G5	G4	G3	G2	G1	G0	x	x	
		3rd										B5	B4	B3	B2	B1	B0	x	x	
	65k	1st										R4	R3	R2	R1	R0	G5	G4	G3	
		2nd										G2	G1	G0	B4	B3	B2	B1	B0	

for lab (no need to connect D0 [only use 8 pins])

Remark : x Don't care bits

Not connected pins

1st: DATA[7:0] = D[8:1]

2nd: DATA[7:0] = D[8:1]

2. LCD: graphics data

```
void LCD_Clear( unsigned short Color )
{
    unsigned int i;

    LCD_SetCursor(0,0);
    ...
    writeCmd(0x0022);
    for( i=0; i< MAX_X*MAX_Y; i++ )
        writeDat(Color);
    ...
}
```

notice: only call `writeCmd` once for multiple `writeDat`

note: cmd=`0x0022` tells LCD that all subsequent `dat` is RGB values

2. LCD: graphics data

`cmd=0x0022` tells LCD that all subsequent `dat` is RGB values

GRAM x,y position
must auto update
after each write to GRAM

Entry Mode (R11h) (POR = 6830h)

R/W	DC	IB15	IB14	IB13	IB12	IB11	IB10	IB9	IB8	IB7	IB6	IB5	IB4	IB3	IB2	IB1	IB0
W	1	VSMode	DFM1	DFM0	TRANS	OEDef	WMode	DMode1	DMode0	TY1	TY0	ID1	ID0	AM	LG2	LG1	LG0
POR		0	1	1	0	1	0	0	0	0	0	1	1	0	0	0	0

controls what auto
update does

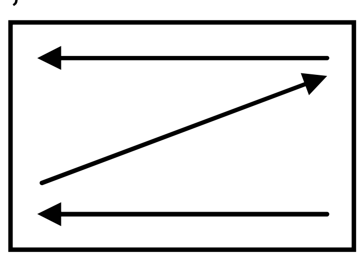
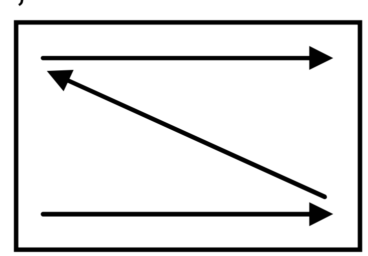
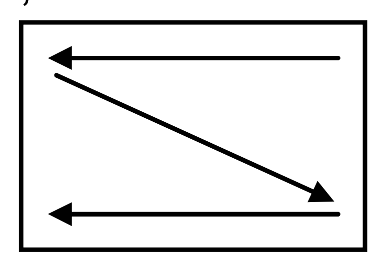
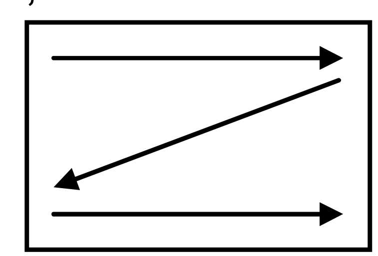
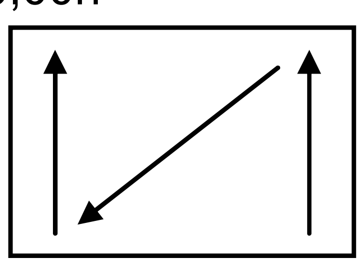
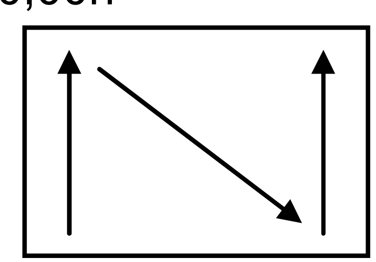
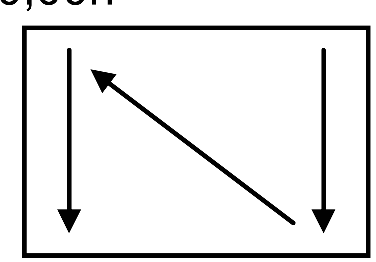
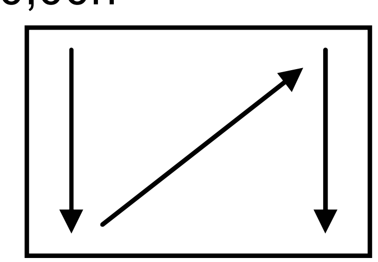
2. LCD: graphics data

assuming:

1. `writeCmd(0x0022);`
2. each `writeDat(...);`

updates GRAM set addr

according to ID,AM

	ID[1:0]="00" Horizontal: decrement Vertical: decrement	ID[1:0]="01" Horizontal: increment Vertical: decrement	ID[1:0]="10" Horizontal: decrement Vertical: increment	ID[1:0]="11" Horizontal: increment Vertical: increment
AM="0" Horizontal	00,00h  13F,EFh	00,00h  13F,EFh	00,00h  13F,EFh	00,00h  13F,EFh
AM="1" Vertical	00,00h  13F,EFh	00,00h  13F,EFh	00,00h  13F,EFh	00,00h  13F,EFh

set GRAM pos to 0,0

(next GRAM data write at 0,0)

240*320=76,800

(all pixels on LCD)

```
LCD_SetCursor(0,0);
...
writeCmd(0x0022);
for( i=0; i< MAX_X*MAX_Y; i++ )
    writeDat(Color);
```

1. write colour
2. increment pos
3. goto 1

Interrupts I

ECE 3710

It doesn't matter what temperature the room is, it's always room temperature.

- Steven Wright

examples of polling (waiting on):

timer expiration

```
wait
  ldr R0,[R1,#0x1C]
  ands R0,#0x1
  beq wait
```

```
while(UART0_STAT == 0x30);
```

serial transmission

a little polling is OK...not this, though:

interrupts: a better way...

interrupts

interrupt controller

polls devices for uC, lets it know
when devices ‘ready’

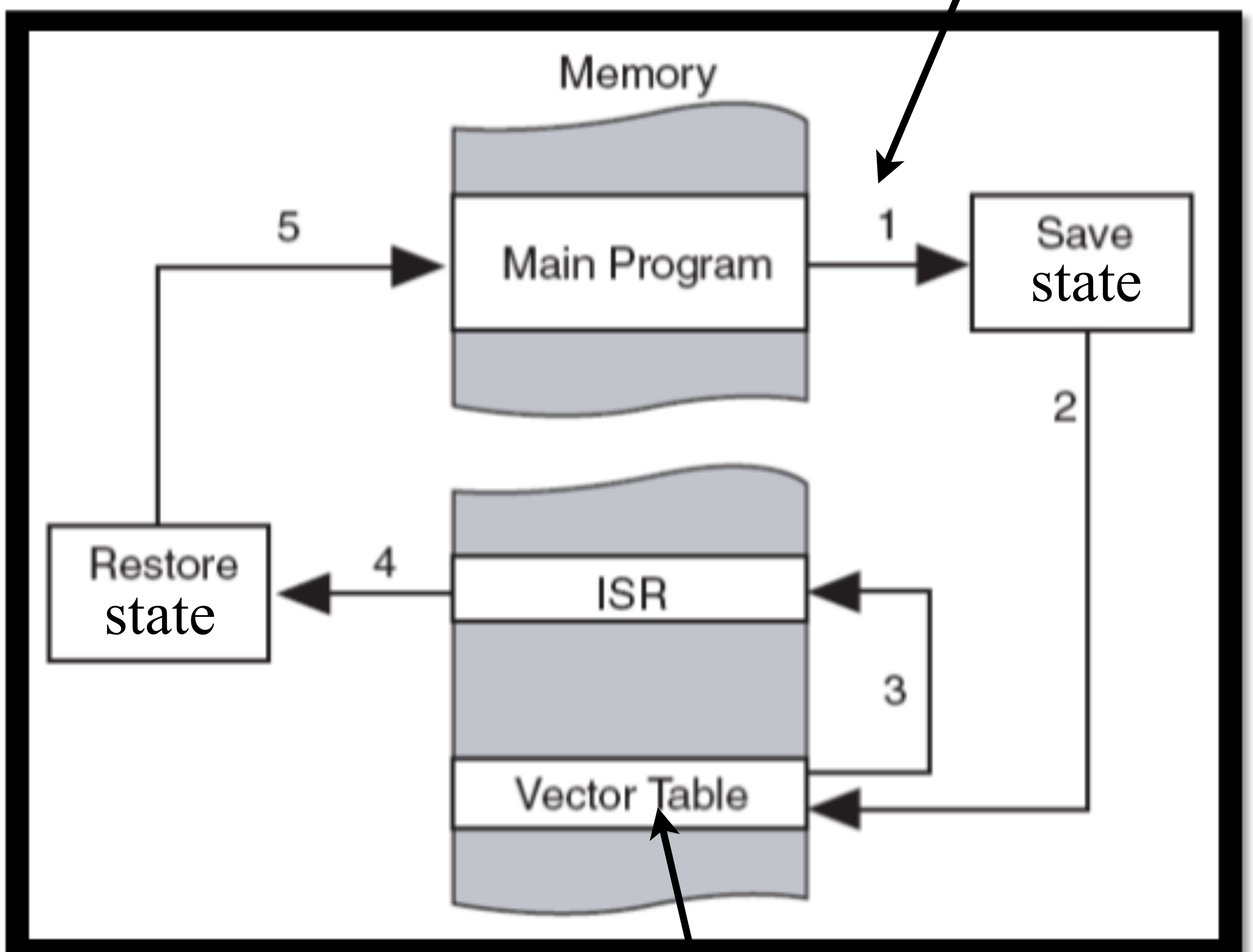
uC can do other stuff

each has own ‘interrupt’

when an interrupt occurs

(uC is doing something)

interrupt here



1. push state onto stack
2. uC looks up address of routine associated with that interrupt
3. PC set to that routine
4. routine finishes: original state restored

each interrupt has own entry

1--4 happens automatically

(less work for us)

some interrupt sources

interrupt service routines (ISR)

what happens when interrupt occurs

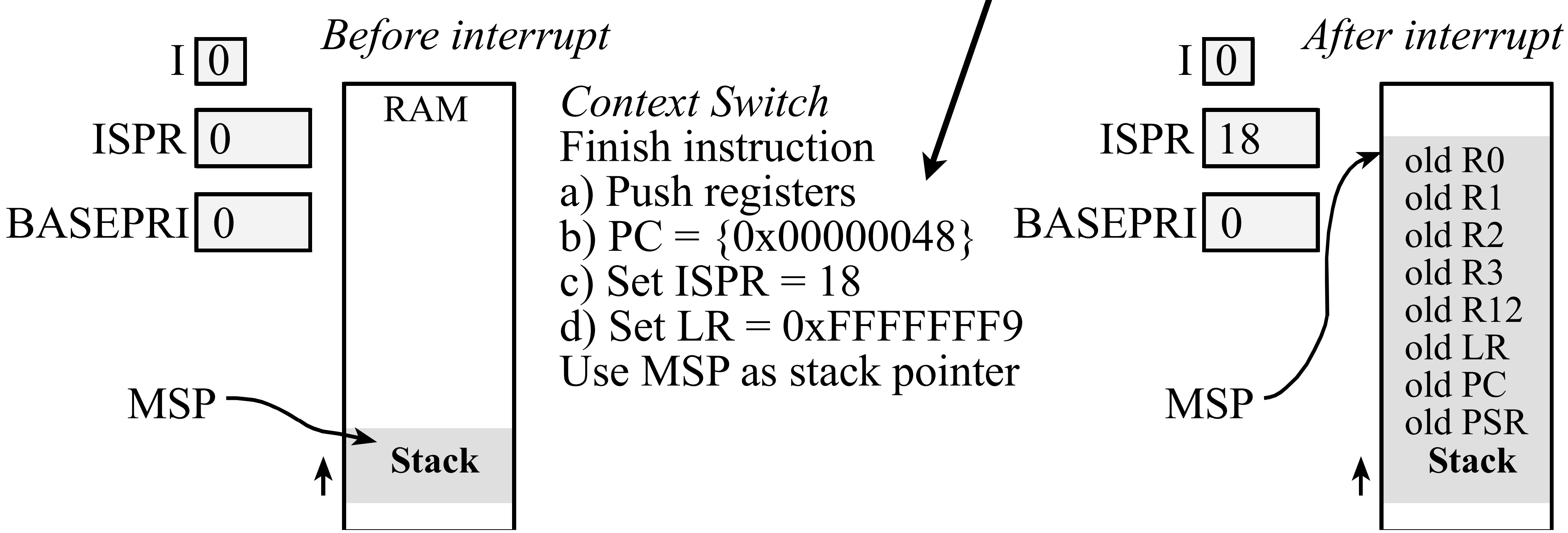
Vector address	Number	IRQ	ISR name in **Startup.s**
0x00000038	14	-2	`PendSV_Handler`
0x0000003C	15	-1	`SysTick_Handler`
0x00000040	16	0	`GPIOPortA_Handler`
0x00000044	17	1	`GPIOPortB_Handler`
0x00000048	18	2	`GPIOPortC_Handler`
0x0000004C	19	3	`GPIOPortD_Handler`
0x00000050	20	4	`GPIOPortE_Handler`

contains address of your ISR code

context (state) switch

for interrupt by GPIOC:

GPIOC ISR addr
located here



end of interrupt service routine (ISR): `bx lr`

set to 0xF...F9
(by uC)

causes registers to be popped off