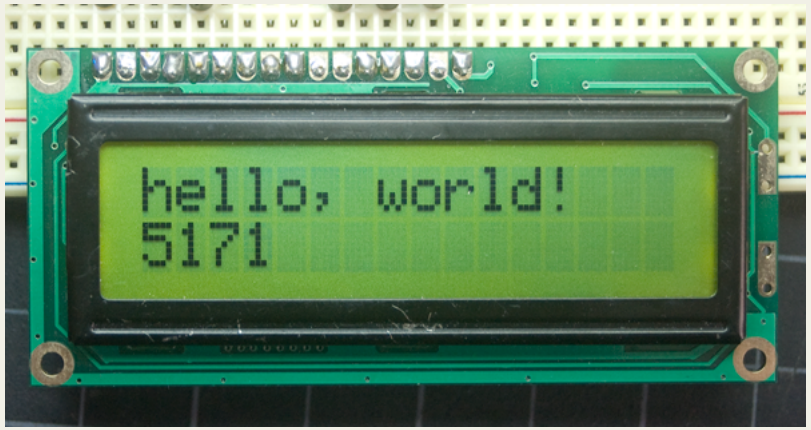


Physics 124: Lecture 4

LCD Text Display

Keypads and Time Slicing

Interrupts

2×16 LCD

- Typically 5×8 dots per character
- Note 16 pins: indicator of common interface

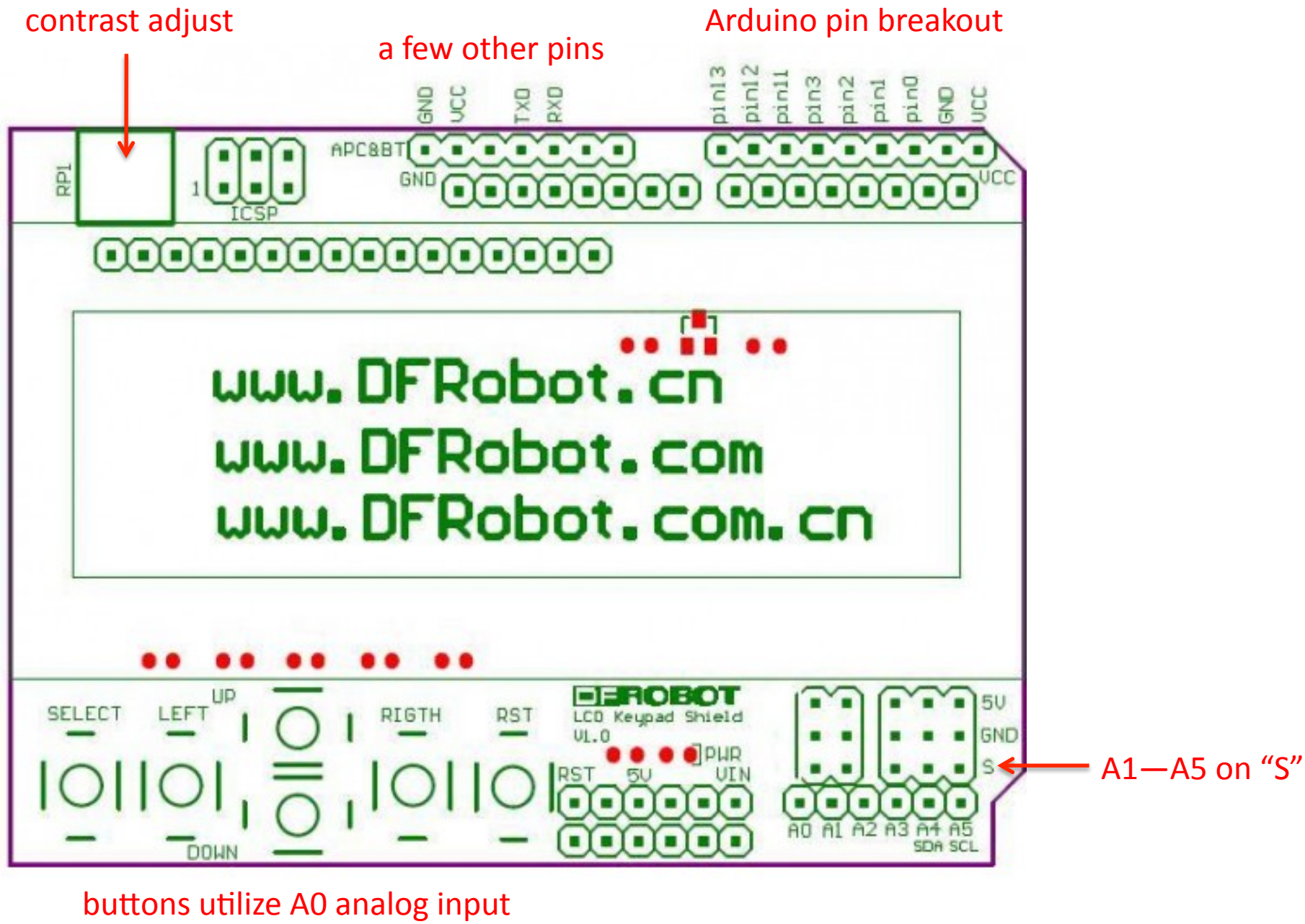
Typical LCD Unit pinout

pin	function	Arduino pin (shield)
1	ground	GND
2	+5 V	+5 V
3	VEE (contrast via potentiometer between 0 and 5 V)	pot on shield
4	RS (LOW = command; HIGH = data/characters)	8
5	RW (LOW = write; HIGH = read)	GND
6	E (enable strobe: toggle to load data and command)	9
7-14	data bus	4,5,6,7 → D4,D5,D6,D7
15	backlight +V	
16	backlight ground	

Note that most features are accessible using only the 4 MSB data pins

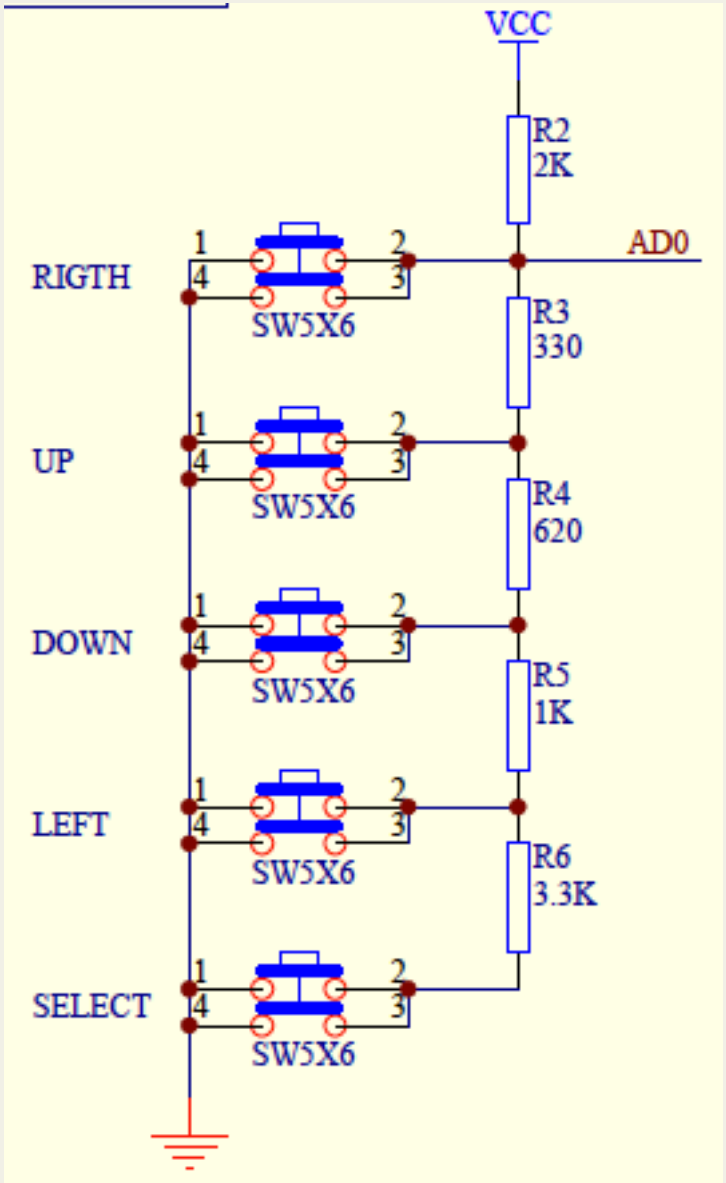
Arduino LCD Shield

- Handy package, includes buttons, contrast pot, some pins/headers for other connections
  - consumes Arduino pins 4, 5, 6, 7, 8, 9
  - leaves 0, 1 for Serial, 2, 3, 10, 11, 12, 13
    - fails to make pin 10 available on header, though

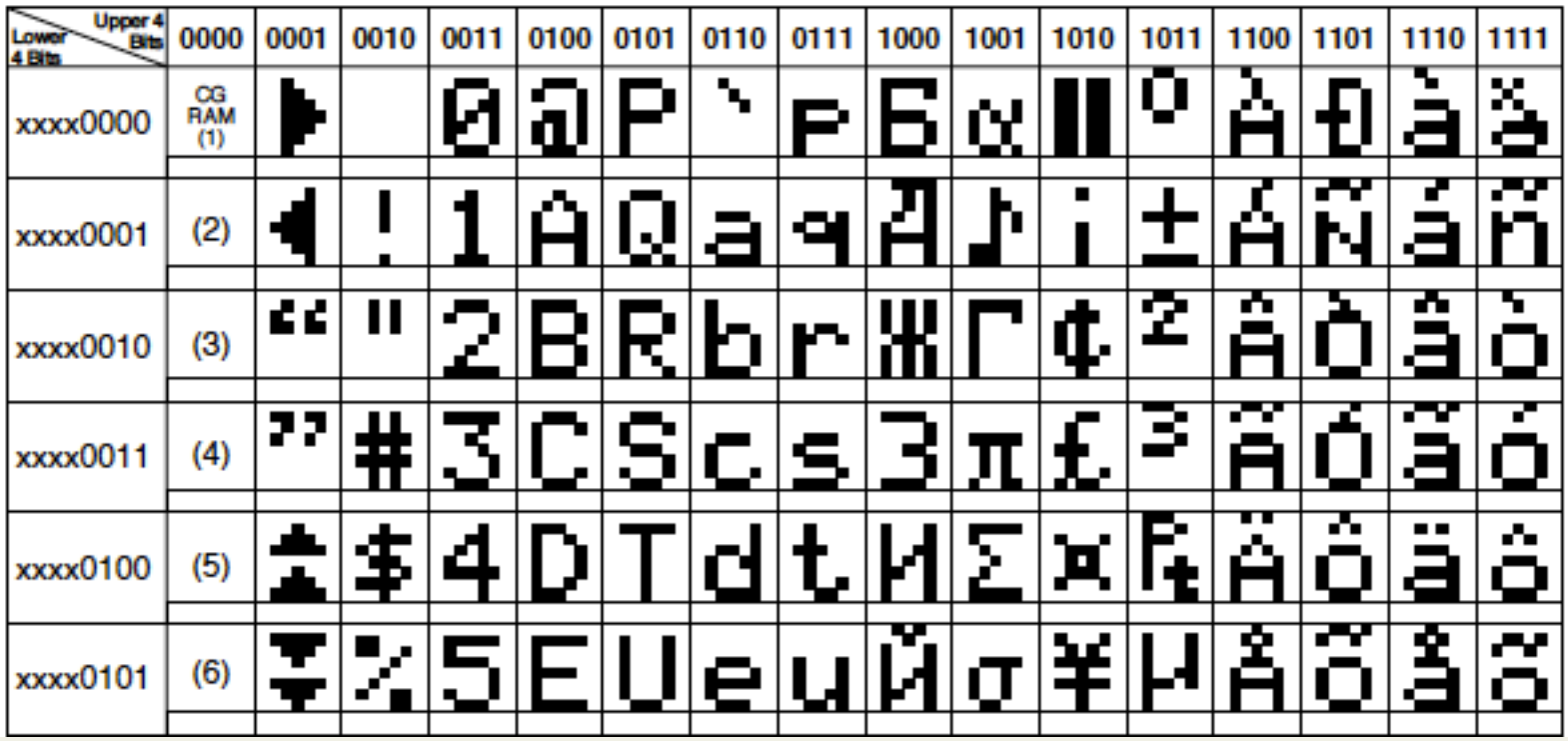

contrast adjust
a few other pins
Arduino pin breakout
GND
VCC
TXD
RXD
pin13
pin12
pin11
pin3
pin2
pin1
pin0
GND
VCC
RP1
1
ICSP
APC&BT
GND
VCC
www.DFRobot.cn
www.DFRobot.com
www.DFRobot.com.cn
SELECT
LEFT
UP
DOWN
RIGTH
RST
DFROBOT
LCD Keypad Shield
V1.0
PWR
RST
5V
VIN
5V
GND
S
A0 A1 A2 A3 A4 A5
SDA SCL
A1—A5 on “S”
buttons utilize A0 analog input

Buttons

- The buttons use a voltage divider tree to present an analog voltage to A0
  - note “RIGTH” typo made it onto printed circuit board!
- I measure the following:
  - none: 4.95 V
  - SELECT: 3.59 V
  - LEFT: 2.44 V
  - DOWN: 1.60 V
  - UP: 0.70 V
  - RIGHT: 0.0 V
- Easily distinguishable



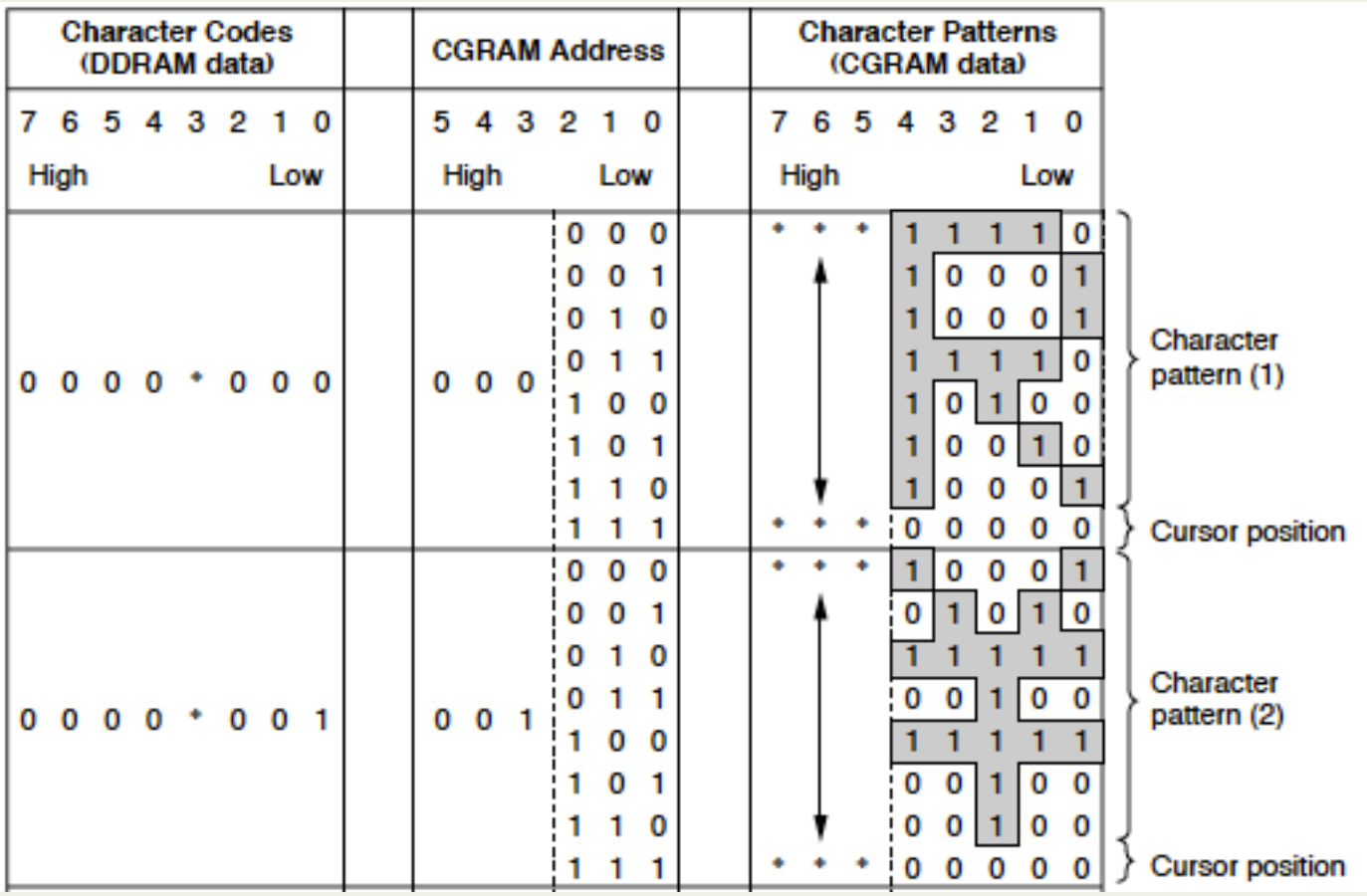
LCD Datasheet



Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)															
xxxx0001	(2)															
xxxx0010	(3)															
xxxx0011	(4)															
xxxx0100	(5)															
xxxx0101	(6)															

- For behind-the-scenes control of the LCD display, see the datasheet
  - http://www.physics.ucsd.edu/~tmurphy/phys124/labs/doc/LCD_HD44780.pdf
- Above is just one snippet of the sort of things within

And one other snippet from LCD datasheet



Character Codes (DDRAM data)	CGRAM Address	Character Patterns (CGRAM data)	
7 6 5 4 3 2 1 0 (High … Low)	5 4 3 2 1 0 (High … Low)	7 6 5 4 3 2 1 0 (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 0 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

- Datasheets: they build character (at least characters)

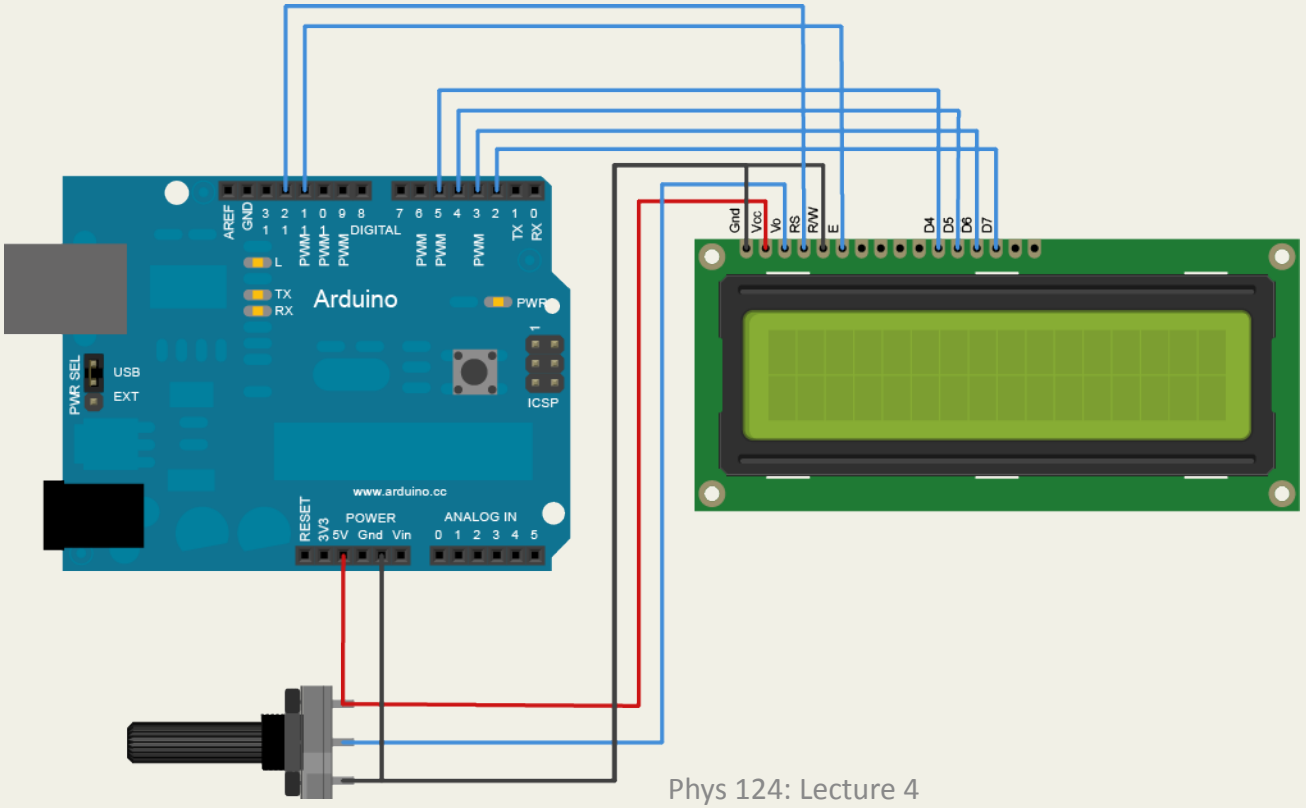
The LiquidCrystal Library

- This is one place I'm not itching for low-level control
  - or wait—where's the fun/challenge in *that* attitude?
- Library makes simple

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);  // matches shield config

void setup() {
  lcd.begin(16, 2);         // # columns & rows
  lcd.print("Phys 124 Rules!");
}

void loop() {
lcd.setCursor(0, 1);        // first col, second row (0 base)
  // print the number of seconds since reset:
  lcd.print(millis()/1000);
}
```

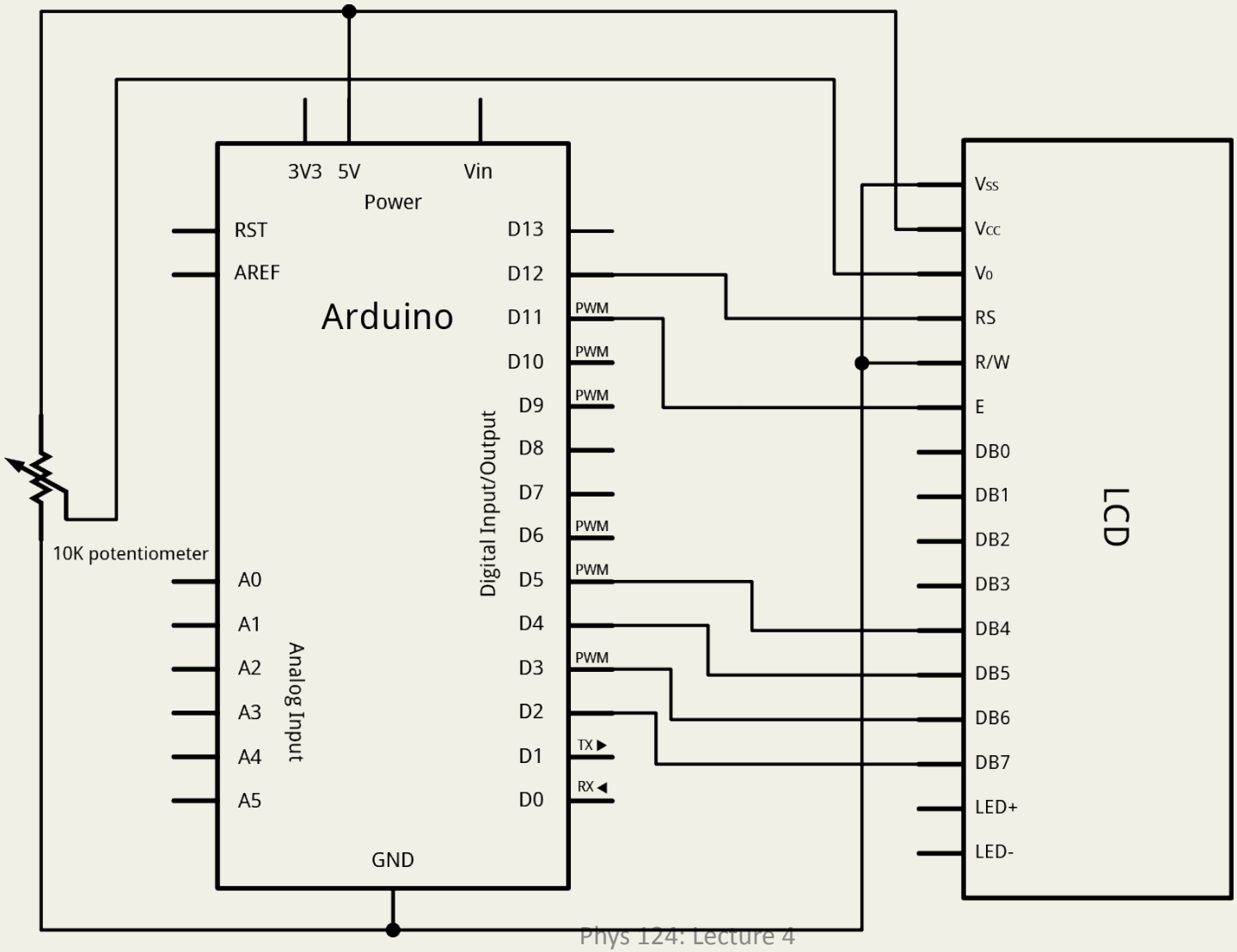
The setup call

- Arguments in LiquidCrystal type are:
  - pins corresponding to: RS, Enable, D4, D5, D6, D7
  - don't need shield at all; just those 6 pins and power/gnd
  - here's one without shield: must hook R/W to gnd; rig pot



Same thing in schematic form

- Note this pinout is different than shield's mapping



Explore the library

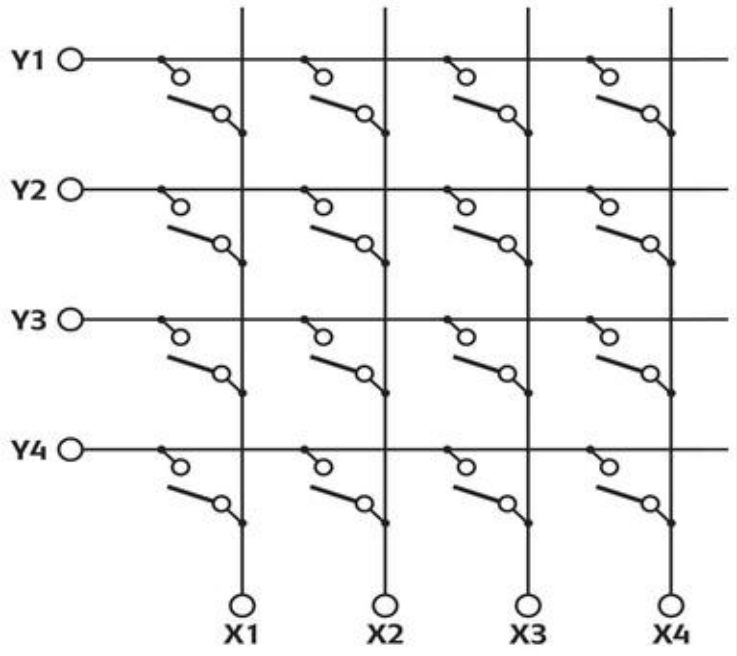
- Can do a lot with a few functions, but more available
  - LiquidCrystal() must use
  - begin() must use
  - clear()
  - home()
  - setCursor() almost certainly use
  - write()
  - print() almost certainly use
  - cursor()
  - noCursor()
  - blink()
  - noBlink()
  - display()
  - noDisplay()
  - scrollDisplayLeft()
  - scrollDisplayRight()
  - autoscroll()
  - noAutoscroll()
  - leftToRight()
  - rightToLeft()
  - createChar()

LCD References

- Good general intro to LCD control
  - http://spikenzielabs.com/SpikenzieLabs/LCD_How_To.html
- Arduino page
  - http://arduino.cc/en/Tutorial/LiquidCrystal
- See links on course site:
  - http://www.physics.ucsd.edu/~tmurphy/phys124/labs/useful_links.html
    - http://www.physics.ucsd.edu/~tmurphy/phys124/labs/doc/LCD-shield-schem.pdf
    - http://www.physics.ucsd.edu/~tmurphy/phys124/labs/doc/LCD_HD44780.pdf

Keypads

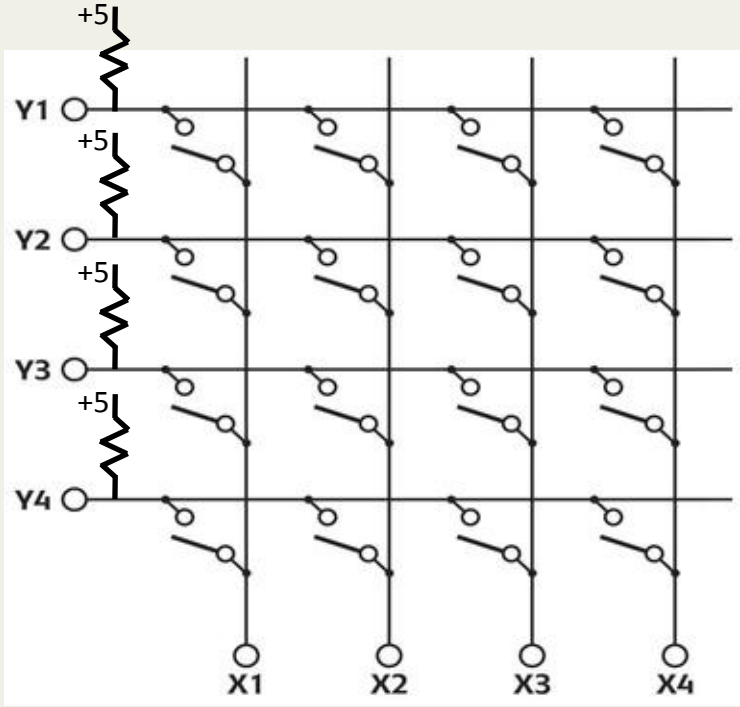
- Most keypads are matrix form: row contact and column contact
  - pressing button connects one row to one column



note crossings do not connect:
dots indicate connection

Reading the keypad

- Imagine we hooked the rows (Y) to four digital inputs with pull-up resistors
  - and hooked the columns (X) up to digital outputs
- Now cycle through X, putting each to zero (LOW) in turn
  - otherwise enforce high state
- Read each row value and see if any inputs are pulled low
  - means switch closed, button pressed
- Called time-slicing



Those Pesky Pullups

- Arduino has a `pinMode` option to engage internal pullup resistors
  - `pinMode(`*pin*`, INPUT_PULLUP);`
  - does just what we want
- Let's start by defining our pins (example values)
  - and our key characters

```
#define ROW1 12          // or whatever pin is hooked to row1
etc.
#define COL1 8
etc.
#define ROWS 4
#define COLS 4
char keys[ROWS][COLS] = {      // handy map of keys
  {'1','2','3','A'},           // black 4x4 keypad
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
```

Now set up pins in `setup()`

```
pinMode(ROW1, INPUT_PULLUP);
etc.
pinMode(COL1, OUTPUT);
etc.
digitalWrite(COL1, HIGH);    // def. state is high; start high
```

- Now in loop()

```
pressed = 0;                          // value for no press

digitalWrite(COL1, LOW);              // assert col 1 low
if (digitalRead(ROW1) == LOW)
  pressed = 0x11;                     // upper digit is row
if (digitalRead(ROW2) == LOW)
  pressed = 0x21;                     // lower digit is col
etc.
digitalWrite(COL1, HIGH);             // reset col1 to high
```

etc. for all 4 columns; the scheme for `pressed` is just one way, my first impulse

Piecing together at end of loop

```
if (pressed != 0 && pressed != last)
{
  row = pressed >> 4;          // drop 4 LSB, look at upper 4
  col = pressed & 0x0f;        // kill upper 4 bits; keep 4 LSB
  ch = keys[row-1][col-1];     // what character, from map
  if (ch != '#')               // treat # as newline
    Serial.print(ch);
  else
    Serial.println("");        // just want return
}
last = pressed;                // preserve knowledge
delay(40);                     // debounce delay
```

- print only if new press, new line if '#' pressed
  - note >> bit shift row look at high nibble;
  - and mask lower 4 bits for isolating lower nibble
  - thus decode into row and column (at least this is *one* way)

Cleaning up code

- Repeating the sweep four times during the loop is a bit clumsy, from a coding point of view
  - begs to be function()-ized

```
int readCol(int column)
{
  int row_press = 0;
  digitalWrite(column, LOW);
  if (digitalRead(ROW1) == LOW)
    row_press = 1;
  if (digitalRead(ROW2) == LOW)
    row_press = 2;
  etc.
  digitalWrite(column, HIGH);

  return row_press;
}
```

Now a function to sweep columns

```
int sweepCols()
{
  int row_press;
  pressed = 0;

  row_press = readCol(COL1);
  if (row_press > 0)
    pressed = (row_press << 4) + 1;
  etc.
  row_press = readCol(COL4);
  if (row_press > 0)
    pressed = (row_press << 4) + 4;

  return pressed;
}
```

now in main loop, just: `pressed = sweepCols();` and otherwise same

And, there's a Library

- Of course there is…
  - http://playground.arduino.cc/code/Keypad
  - installed in sketch folder `libraries/` directory

```
#include <Keypad.h>
const byte ROWS = 4; //four rows
const byte COLS = 3; //three columns
char keys[ROWS][COLS] = {{'1','2','3'}, {'4','5','6'},
                         {'7','8','9'}, {'#','0','*'}};
byte rowPins[ROWS] = {5, 4, 3, 2}; //conn. to the row pins of the keypad
byte colPins[COLS] = {8, 7, 6}; //conn. to the col pins of the keypad

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

void setup(){
  Serial.begin(9600);}

void loop(){
  char key = keypad.getKey();
  if (key != NO_KEY)
    Serial.println(key);
}
```

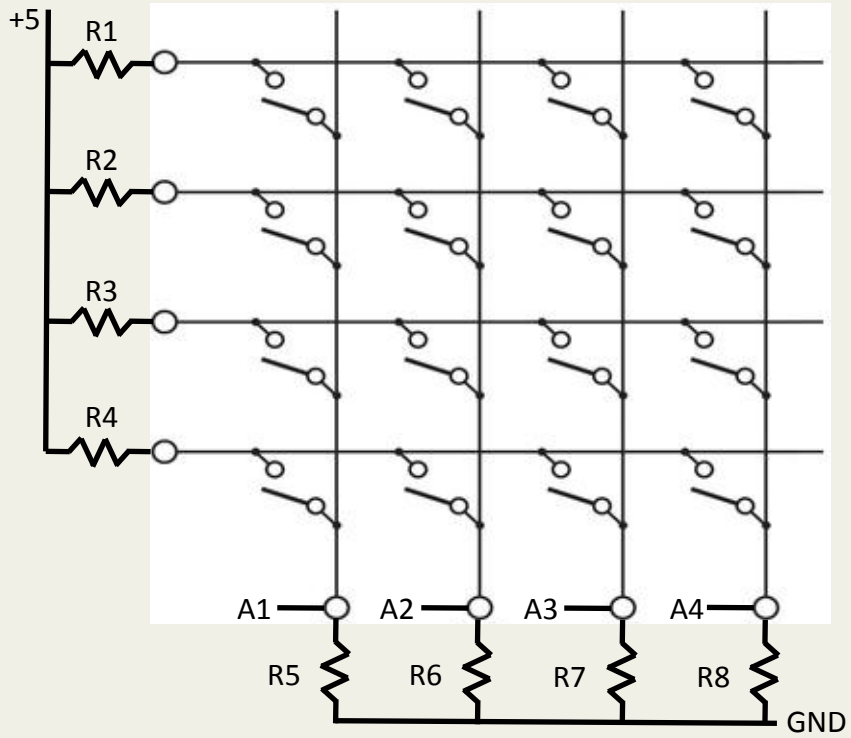
Some Notes on the Keypad Library

- Note that the key map is taken seriously by Keypad.h
  - if any character appears twice, it messes up
  - therefore more than a printing convenience; a core functional element of the operation
- Functions
  - `void begin(makeKeymap(userKeymap))`
  - `char waitForKey()`
  - `char getKey()`
  - `KeyState getState()`
  - `boolean keyStateChanged()`
  - `setHoldTime(unsigned int time)`
  - `setDebounceTime(unsigned int time)`
  - `addEventListener(keypadEvent)`
- Consult link on previous slide for descriptions

Combining LCD and Keypad?

- The LCD uses six digital pins
- A 4x4 keypad needs 8 pins
- Uno has 14, but pins 0 and 1 are used by Serial
  - could forgo serial communications, and max out pins
- Need a better way, *less greedy*
- Take a page from LCD shield buttons: use analog input
- Many schemes are possible
  - generally: +5 V on rows/cols, GND on other, resistors between
  - could have all 16 buttons map to a *single* analog input
    - interesting problem in designing appropriate network
  - or make it easier and map to four analog inputs

Four-Input Scheme



- R1 thru R4 could be 10 kΩ, 4.7 kΩ, 2.2 kΩ, 1 kΩ
- R5 thru R8 could be all 3.3 kΩ, or in that ballpark
  - voltages will be 0 (nothing pressed), 1.25 V (top row), 2.06V; 3 V; and 3.8 V for resp. rows — lots of separation
- Poll each A# input to ascertain keypress

Interrupts

- Sometimes we can't afford to miss a critical event, while the main loop is busy, or in a delay, etc.
- Interrupts demand immediate attention
- Uno has two interrupts
  - int.0 on pin 2; int.1 on pin 3
  - Mega has 6 available interrupts
- You can exempt some of loop from interruption
  - may be rare that you need to do this, but…

```
void loop()
{
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

Easily implemented

- Just have to attach an interrupt to a service routine
  - `attachInterrupt(`*int#*, *function*, *trigger_type*`);`
  - the interrupt number is 0 or 1 on Uno (pins 2 or 3)
  - the function is some function you've created to service the interrupt: name it whatever makes sense
  - trigger_type can be
    - RISING: detects edge from logic low to logic high
    - FALLING: detects falling edge
    - CHANGE: any change between high/low (watch out for bounce!)
    - LOW: a low state will trigger an interrupt
  - note that `delay()` will not work within the service routine
    - need `delayMicroseconds()`, only good up to 16383 μs
    - but not often interested in delay in interrupt routine

Simple example

- Turn on/off LED via interrupt; note volatile variable

```
int pin = 13;
volatile int state = LOW;

void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}

void loop()
{
  digitalWrite(pin, state);
}

void blink()
{
  state = !state;
}
```

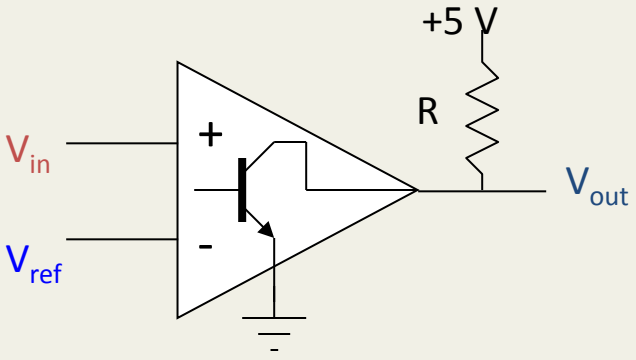
Interrupt Notes

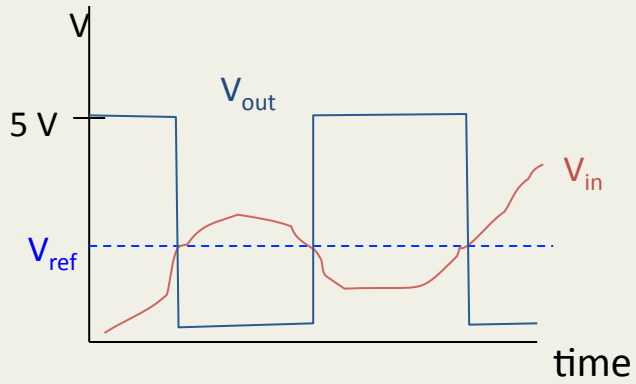
- *Inside the attached function, delay() won't work and the value returned by millis() will not increment. Serial data received while in the function may be lost. You should declare as volatile any variables that you modify within the attached function.*
- See the page for `attachInterrupts()`:
  - http://arduino.cc/en/Reference/AttachInterrupt

Interrupts from analog?

- What if we need to make a digital interrupt out of an analog signal like the analog-scheme keypad?
- Can use a *comparator* to sense if we're above or below some threshold voltage
  - output is digital state
  - could also use a high-pass (differentiator) to sense any significant *change* in the analog level, fed into a comparator

Comparator Basics

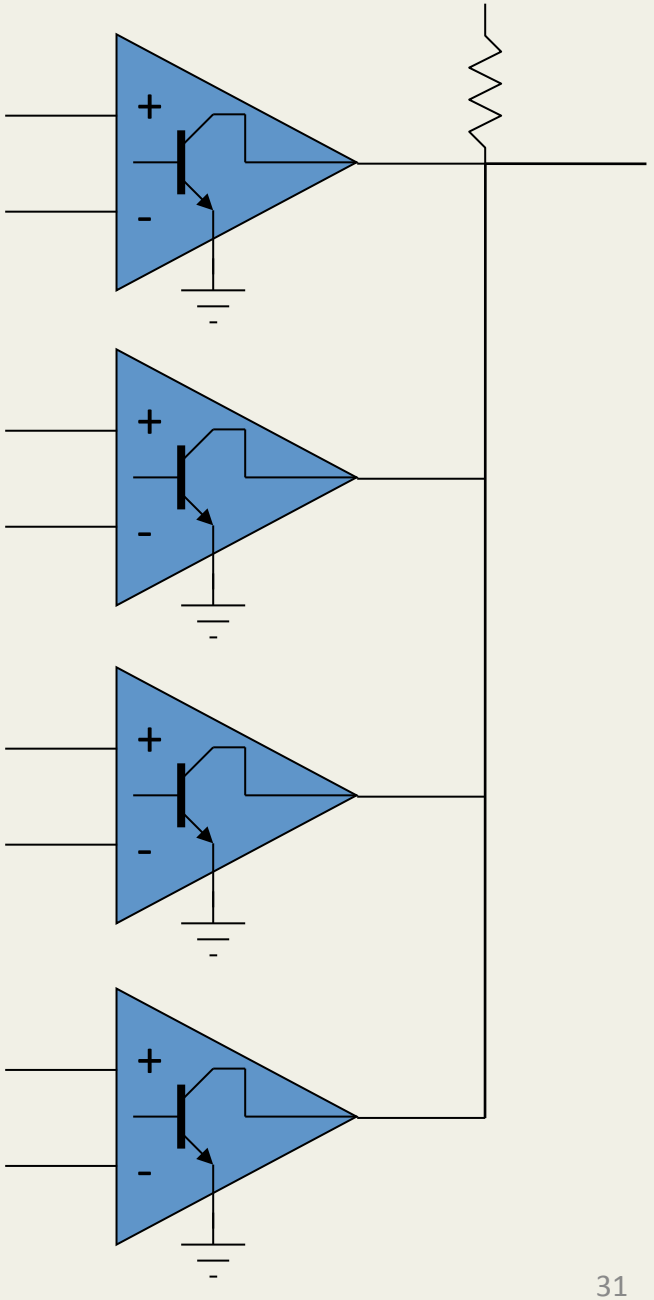




- Scheme is: when + input larger than − input, transistor driven to ON
  - then current flows through transistor and output is pulled low
- When $V_{in} < V_{ref}$, V_{out} is pulled high (through the pull-up resistor—usually 1 kΩ or more)
  - this arrangement is called "open collector" output: the output is basically the collector of an npn transistor: in saturation it will be pulled toward the emitter (ground), but if the transistor is not driven (no base current), the collector will float up to the pull-up voltage
- The output is a "digital" version of the signal
  - with settable low and high values (here ground and 5V)

Can Gang Open-Collector Comparators into Chain

- Put same (or different) threshold values on – inputs and four different analog signals on +
  - tie all four open collectors together with common pull-up
  - if any comparator activates, the associated transistor will pull the combined output low, and the other (off) transistors won't care
- The "311" comparator is standard



Upcoming Lab

- Monday is a holiday, so this is it for lab prep!
- In Week 3 lab, we will:
  - make an LCD analog voltage meter
  - read a 4x4 keypad using the time-slice method and 8 pins
  - combine the keypad, LCD, and interrupts into a party