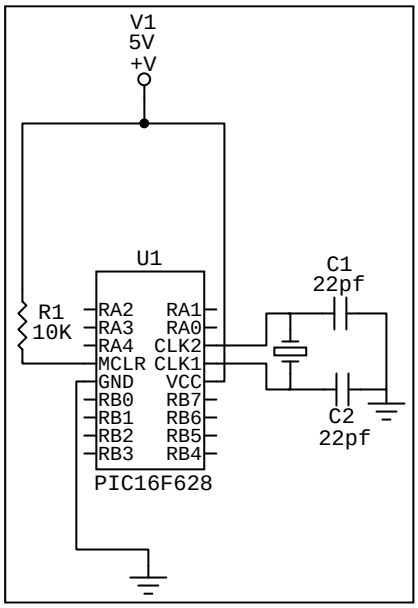
Appendix 1
Basic Electronics

The PIC Hardware

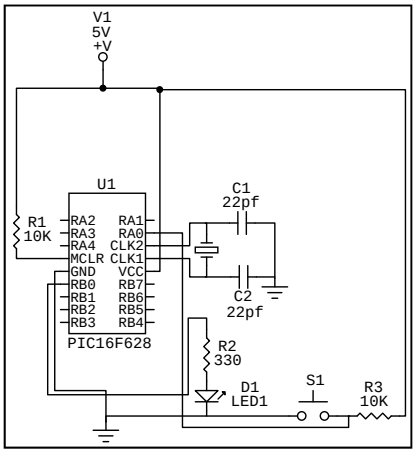
Well so far you have gained an insight about the various features of 1PIC microcontroller. Now is the time to understand how to use it in a project. In order to experiment with PIC Lab you don't need to do that, but it is an advantage to know, how the microcontroller based circuit is made. After all you are going to design your devices, using these microcontrollers and if you don't know how to put the thing into your circuit, of what use will be all this exercise. Certainly you can not use the PIC Lab motherboard in your every application.!



Basic circuit drawing is shown here. PIC microcontroller needs only four components to start functioning. A crystal oscillator of your choice is connected between CLK1 and CLK2. these pins will vary among various PICs, so always consult data sheet to locate pins. It is customary to talk in terms of pin names, rather than numbers in microcontrollers. The Vlk pins are grounded using 22pf capacitors. A 10K resistor is connected to Vcc at MCLR. Connecting a push switch to ground will provide a convenient Reset circuit. (don't remove 10K pull up) so that when switch is open MCLR gets Vcc. That's it. This is the basic circuit, and rest of the pins are all I/O you are free to use them, in whatever way you like. This circuit will run whatever program has been loaded into it. Since you will have to take the IC to programmer, try putting it in a socket.

Now lets see if we can make a blinking LED connected to RB0, and an input switch connected to RA0.

Although the PIC pins can both source and sink the load. It is customary to use them as source, so that a '1' on pin drives the load. These pins can give sufficient current to handle the load of an LED, still it is better to protect the pin from overloading by limiting the current flow using a current limiting resistor, usually a 330 Ω or 680 Ω .



A switch can be connected to vcc, giving '1' on the pin or connected to ground giving '0' when pushed. We prefer the second form, and the pin is connected using a 10K resistor to vcc to give logical '1' when switch is open and give logical '0' when switch is pressed.

PIC Lab uses switches in this form, so when a switch is pressed, it will deliver a logical '0' to the program.

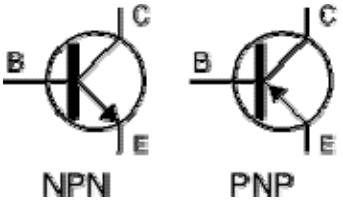
For driving low current circuits, like other digital devices, pins can be used directly. To drive a transistor, it is customary to use a current limiting resistor in series, like 2.2K with the base of transistor. The transistor can then be used to drive heavy loads, like switching a relay On.

Using Transistors (Basic Electronics)

There are two types of standard transistors, **NPN** and **PNP**, with different circuit symbols. The letters refer to the layers of semiconductor material used to make the transistor. Most transistors used today are NPN because this is the easiest type to make from silicon. We are going to talk about NPN transistors and if you are new to electronics it is best to start by learning how to use these first.

The leads are labeled **base** (B), **collector** (C) and **emitter** (E). These terms refer to the internal operation of

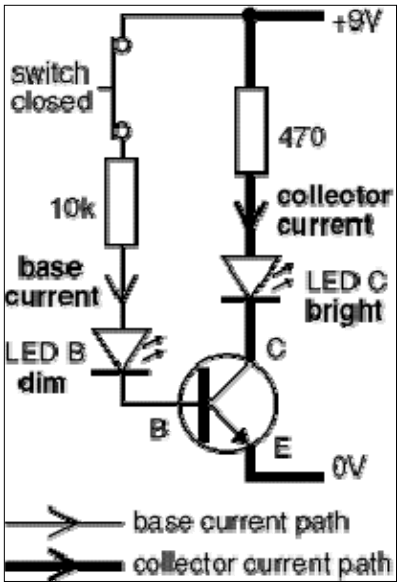
a transistor but they are not much help in understanding how a transistor is used, so just treat them as labels!



Transistor currents

The diagram shows the two current paths through a transistor. You can build this circuit with two standard 5mm red LEDs and any general purpose low power NPN transistor (BC108, BC182 or BC548 for example).

The small **base current** controls the larger **collector current**.



When the switch is closed a small current flows into the base (B) of the transistor. It is just enough to make LED B glow dimly. The transistor amplifies this small current to allow a larger current to flow through from its collector (C) to its emitter (E). This collector current is large enough to make LED C light brightly.

When the switch is open no base current flows, so the transistor switches off the collector current. Both LEDs are off.

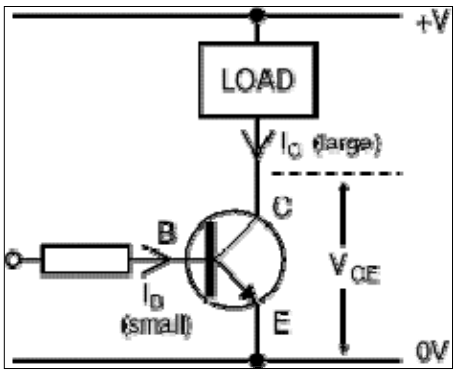
A transistor amplifies current and can be used as a switch.

This arrangement where the emitter (E) is in the controlling circuit (base current) and in the controlled circuit (collector current) is called **common emitter mode**. It is the most widely used arrangement for transistors so it is the one to learn first.

Thus if base of transistor is given a small current via a resistance in series and connected to microcontroller pin, a logical '1' on microcontroller will turn the transistor on, and a logical '0' will turn it off.

Using a transistor as a switch

When a transistor is used as a switch it must be either **OFF** or **fully ON**. In the fully ON state the voltage V_{CE} across the transistor is almost zero and the transistor is said to be **saturated** because it cannot pass any more collector current Ic. The output device switched by the transistor is usually called the 'load'.



The power developed in a switching transistor is very small:

In the **OFF** state: power = Ic × V_{CE}, but Ic = 0, so the power is zero.

In the **full ON** state: power = Ic × V_{CE}, but V_{CE} = 0 (almost), so the power is very small.

This means that the transistor should not become hot in use and you do not need to consider its maximum power rating. The important ratings in switching circuits are the **maximum collector current Ic (max)** and the **minimum current gain h_{FE}(min)**. The transistor's voltage ratings may be ignored unless you are using a supply voltage of more than about 15V. For information about the operation of a transistor please see the functional model above.

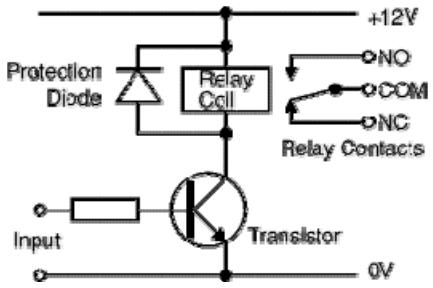
Protection diode

If the load is a **motor**, **relay** or **solenoid** (or any other device with a coil) a diode must be connected across the load to protect the transistor (and chip) from damage when the load is switched off. The diagram shows how this is connected 'backwards' so that it will normally NOT conduct. Conduction only occurs when the load is switched off, at this moment current tries to continue flowing through the coil and it is harmlessly diverted through the diode. Without the diode no current could flow and the coil would produce a damaging high voltage 'spike' in its attempt to keep the current flowing.

When to use a Relay

Transistors cannot switch AC or high voltages (such as mains electricity) and they are not usually a good choice for switching large currents (> 5A). In these cases a relay will be needed, but note that a low power

transistor may still be needed to switch the current for the relay's coil!



Advantages of relays:

Relays can switch **AC and DC**, transistors can only switch DC.

Relays can switch **high voltages**, transistors cannot.

Relays are a better choice for switching **large currents** (> 5A).

Relays can switch **many contacts** at once.

Disadvantages of relays:

Relays are **bulkier** than transistors for switching small currents.

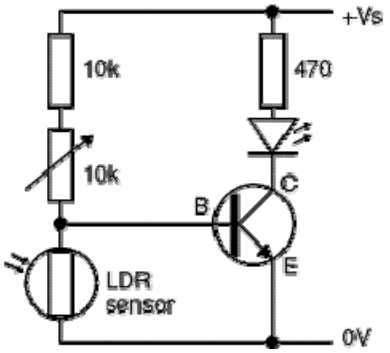
Relays **cannot switch rapidly**, transistors can switch many times per second.

Relays **use more power** due to the current flowing through their coil.

Relays **require more current than many chips can provide**, so a low power transistor may be needed to switch the current for the relay's coil.

Using a transistor switch with sensors

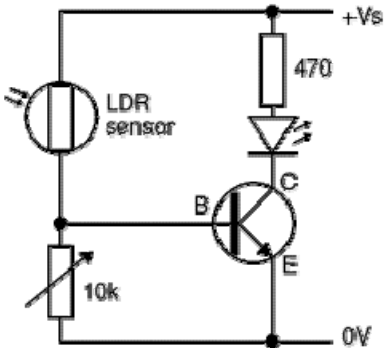
The circuit diagram shows an LDR (light sensor) connected so that the LED lights when the LDR is in darkness. The variable resistor adjusts the brightness at which the transistor switches on and off. Any general purpose low power transistor can be used in this circuit.



The 10k fixed resistor protects the transistor from excessive base current (which will destroy it) when the variable resistor is reduced to zero. To make this circuit switch at a suitable brightness you may need to experiment with different values for the fixed resistor, but it must not be less than 1k.

If the transistor is switching a load with a coil, such as a motor or relay, remember to add a protection diode across the load.

The switching action can be inverted, so the LED lights when the LDR is brightly lit, by swapping the LDR and variable resistor. In this case the fixed resistor can be omitted because the LDR resistance cannot be reduced to zero.



Note that the switching action of this circuit is not particularly good because there will be an intermediate brightness when the transistor will be **partly on** (not saturated). In this state the transistor is in danger of overheating unless it is switching a small current. There is no problem with the small LED current, but the larger current for a lamp, motor or relay is likely to cause overheating.

Other sensors, such as a thermistor, can be used with this circuit, but they may require a different variable resistor. You can calculate an approximate value for the variable resistor (Rv) by using a multimeter to find the minimum and maximum values of the sensor's resistance (Rmin and Rmax):

Variable resistor, Rv = square root of (Rmin × Rmax)

For example an LDR: Rmin = 100, Rmax = 1M, so Rv = square root of (100 × 1M) = 10k.

You can make a much better switching circuit with sensors connected to a suitable IC (chip). The switching action will be much sharper with no partly on state.

LED lights when the LDR is **dark**

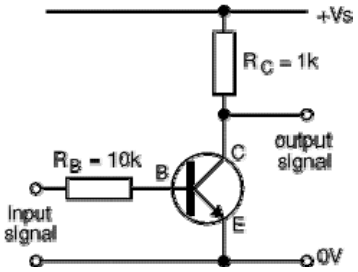
LED lights when the LDR is **bright**

A transistor inverter (NOT gate)

Inverters (NOT gates) are available on logic chips but if you only require one inverter it is usually better to use this circuit. The output signal (voltage) is the inverse of the input signal:

When the input is high (+Vs) the output is low (0V).

When the input is low (0V) the output is high (+Vs).



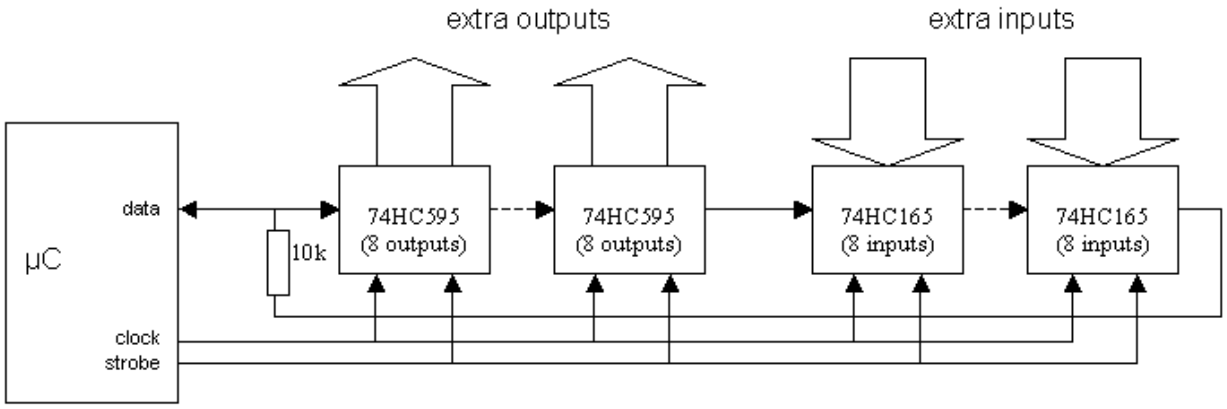
Any general purpose low power NPN transistor can be used. For general use R_B = 10k and R_C = 1k, then the inverter output can be connected to a device with an input impedance (resistance) of at least 10k such as a logic chip or a 555 timer (trigger and reset inputs).

If you are connecting the inverter to a CMOS logic chip input (very high impedance) you can increase R_B to 100k and R_C to 10k, this will reduce the current used by the inverter.

Appendix 2
Expanding Microcontroller I/O Lines

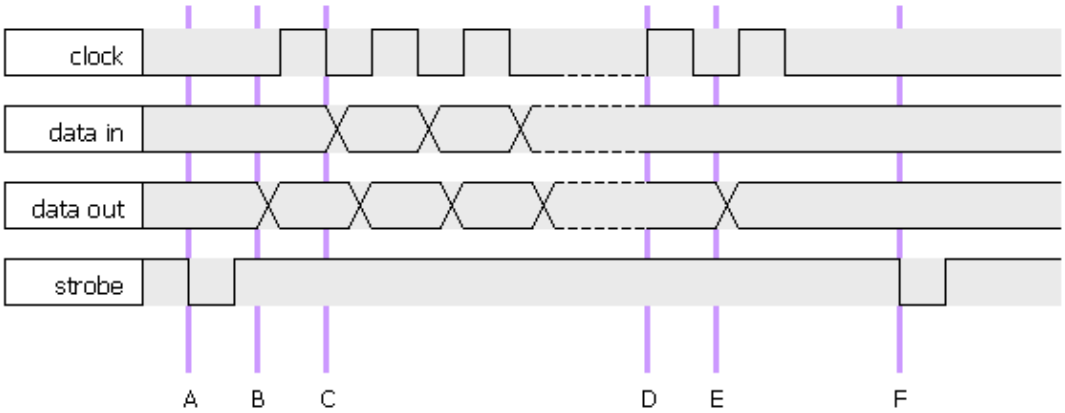
The most precious resource on a microcontroller is its I/O lines. Today's I/O line hungry applications require more and more lines from microcontrollers. For this reason many microcontrollers with more and more I/O lines are available. However remaining confined to your existing microcontroller, you can expand its I/O lines, by using Serial-In parallel Out, or parallel In serial Out Shift registers.

Shift registers like 74HC595 require three I/O lines, one for serial data, one for clock signals to shift data and one for latching the data from internal registers to output lines of shift registers. 74HC595 is an 8 bit shift register, which means you get 8 lines (for Output) by sacrificing three lines of microcontroller, a gain of 5 lines. However the most beautiful thing is that they can be chained together, in definitely, so that you can shift out 16, 24 or even 32 bits of data, just by using three I/O lines.



2.2. How to drive it

The circuit description covers use of the 74HC595. Use of the 74HC4094 (which may be cheaper and easier to find) is covered later in this document. First the strobe line is dipped low and back high again. This latches all the inputs into the 74HC165 input shift registers. Then the clocking begins. With each clock pulse, the data line is set to an output, and the appropriate data bit is presented on the line to be clocked into the output shift register. On the same clock pulse, the input shift register presents its next data bit to the mi-



crocontroller data pin. The data pin is set to an input, and this data bit read.

The number of clock pulses required is the larger of the number of inputs and the number of outputs. After this number of clock pulses, all the required output states have been shifted into position in the 74HC597 output shift registers, and another dipping of the "strobe" line is performed to set these states on the shift register output pins.

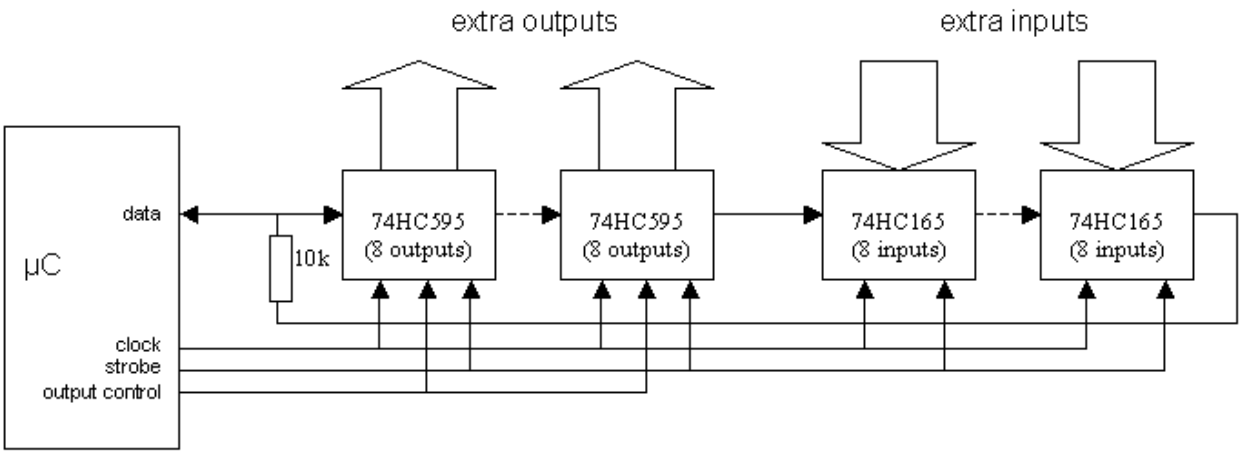
The points in this diagram are:

- A: STROBE is pulsed low to latch inputs.
- B: DATA line is set to an output, and appropriate data placed on it. CLOCK is pulsed high to shift this data bit into the chain, and get next bit from the chain.
- C: DATA line is set to an input, and the data bit is read from the chain.
- D: In this example there are more outputs than inputs, so from this point, the rest of the outputs are clocked out.
- E: This is the rest of the outputs being clocked out.
- F: Finally, the STROBE line is pulsed again to latch the outputs onto the output shift register output pins.

You can make the chain any length you want, with as many output stages or input stages as required. It may be all input stages, or it may be all output stages too. There are constants in the code where the dimensions of the shift register are defined, and they control how the software drives it.

2.3. Protected outputs

The outputs will be in an undefined state when the circuit is powered up. This may be dangerous if the lines that drive the output must not be turned on unless special circumstances are observed. Therefore, the output shift registers chosen have three-state outputs. This means that their outputs can be turned off (not high or low but effectively open-circuit). They will then need to be pulled high or low as required by the circuit that the output is connected to. Using this three-state option requires a further output line from the microcontroller as shown below:



For practical project on shift registers see our 8 x 32 Matrix LED project, which uses 4 shift registers.

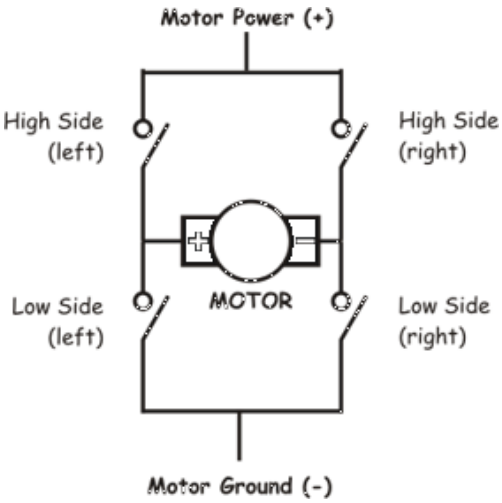
Appendix 3
H-Bridge and DC Motors

Introduction

A number of web sites talk about H-bridges, they are a topic of great discussion in robotics clubs and they are the bane of many robotics hobbyists. I periodically chime in on discussions about them, and while not an expert by a long shot I've built a few over the years. Further, they were one of my personal stumbling blocks when I was first getting into robotics. This section is devoted to the theory and practice of building H-bridges for controlling brushed DC motors (the most common kind you will find in hobby robotics ...) .

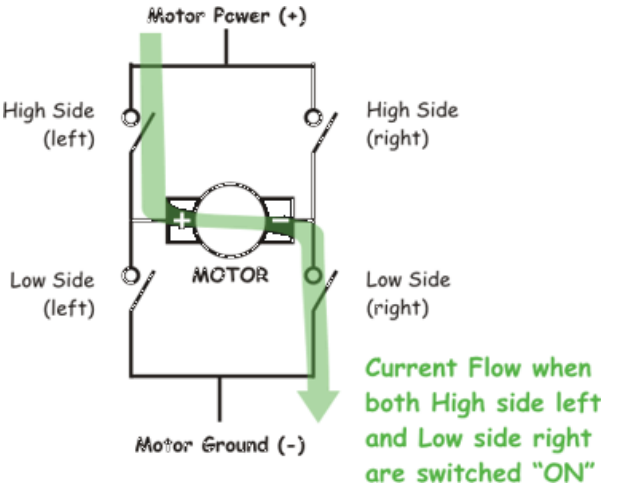
Basic Theory

Let's start with the name, H-bridge. Sometimes called a "full bridge" the H-bridge is so named because it has four switching elements at the "corners" of the H and the motor forms the cross bar. The basic bridge is shown in the figure to the right.



Of course the letter H doesn't have the top and bottom joined together, but hopefully the picture is clear. The key fact to note is that there are, in theory, four switching elements within the bridge. These four elements are often called, high side left, high side right, low side right, and low side left (when traversing in clockwise order).

The switches are turned on in pairs, either high left and lower right, or lower left and high right, but never both switches on the same "side" of the bridge. If both switches on one side of a bridge are turned on it creates a short circuit between the battery plus and battery minus terminals. This phenomena is called shoot through in the Switch-Mode Power Supply (SMPS) literature. If the bridge is sufficiently powerful it will absorb that load and your batteries will simply drain quickly. Usually however the switches in question melt.



To power the motor, you turn on two switches that are diagonally opposed. In the picture to the right, imagine that the high side left and low side right switches are turned on. The current flow is shown in green.

The current flows and the motor begins to turn in a "positive" direction. What happens if you turn on the high side right and low side left switches? You guessed it, current flows the other direction through the motor and the motor turns in the opposite direction.

Pretty simple stuff right? Actually it is just that simple, the tricky part comes in when you decide what to use for switches. Anything that can carry a current will work, from four SPST switches, one DPDT switch, relays, transistors, to enhancement mode power MOSFETs.

One more topic in the basic theory section, quadrants. If each switch can be controlled independently then you can do some interesting things with the bridge, some folks call such a bridge a "four quadrant device" (4QD get it?). If you built it out of a single DPDT relay, you can really only control forward or reverse. You can build a small truth table that tells you for each of the switch's states, what the bridge will do. As each switch has one of two states, and there are four switches, there are 16 possible states. However,

since any state that turns both switches on one side on is "bad" (smoke issues forth), there are in fact only four useful states (the four quadrants) where the transistors are turned on.

H-Bridge Driver Chips

A few driver chips are available, which contain the H-bridge based upon heavy duty switching transistors. One such chip is L298 which contains 2 drivers for two DC motors.

Appendix 4
Stepper Motors

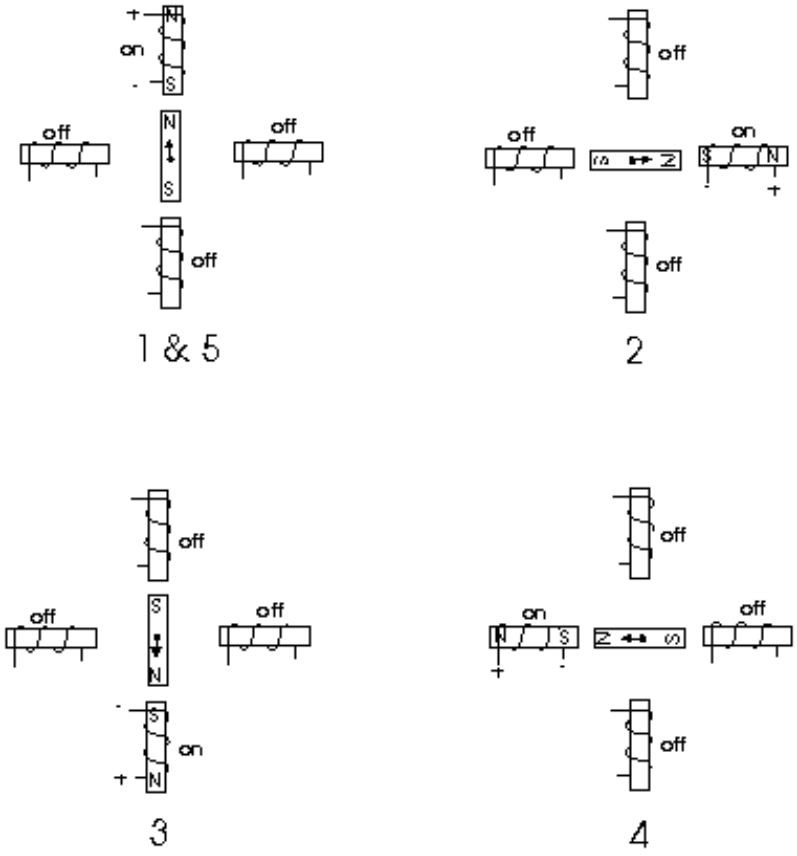
A **stepper motor** is a brushless, synchronous electric motor that can divide a full rotation into a large number of steps. The motor's position can be controlled precisely, without any feedback mechanism.

Fundamentals of operation

Stepper motors operate much differently from normal DC motors, which rotate when voltage is applied to their terminals. Stepper motors, on the other hand, effectively have multiple "toothed" electromagnets arranged around a central gear-shaped piece of iron. The electromagnets are energized by an external control circuit, such as a microcontroller. To make the motor shaft turn, first one electromagnet is given power, which makes the gear's teeth magnetically attracted to the electromagnet's teeth. When the gear's teeth are thus aligned to the first electromagnet, they are slightly offset from the next electromagnet. So when the next electromagnet is turned on and the first is turned off, the gear rotates slightly to align with the next one, and from there the process is repeated. Each of those slight rotations is called a "step." In that way, the motor can be turned a precise angle.

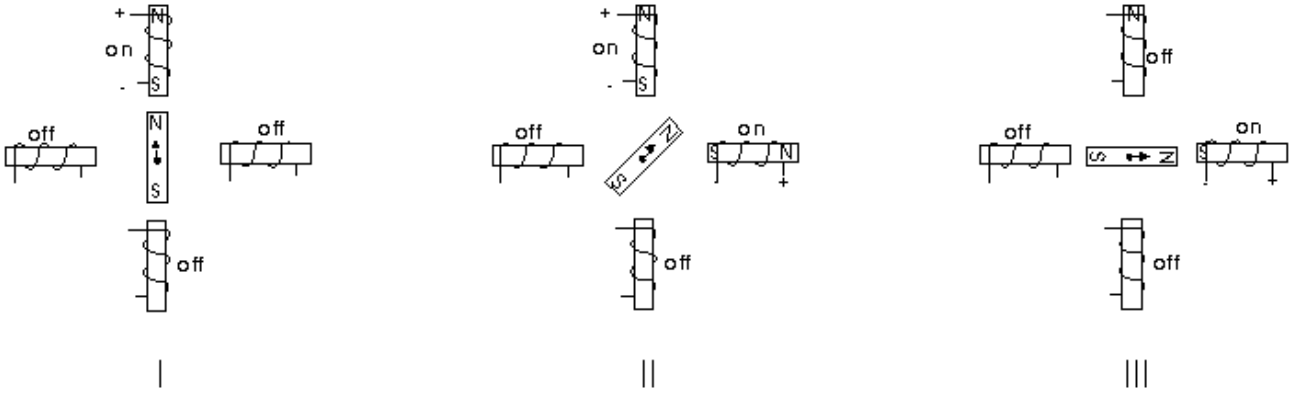
How Stepper Motors Work

Stepper motors consist of a permanent magnet rotating shaft, called the rotor, and electromagnets on the stationary portion that surrounds the motor, called the stator. Fig illustrates one complete rotation of a stepper motor. At position 1, we can see that the rotor is beginning at the upper electromagnet, which is currently active (has voltage applied to it). To move the rotor clockwise (CW), the upper electromagnet is deactivated and the right electromagnet is activated, causing the rotor to move 90 degrees CW, aligning itself with the active magnet. This process is repeated in the same manner at the south and west electromagnets until we once again reach the starting position.



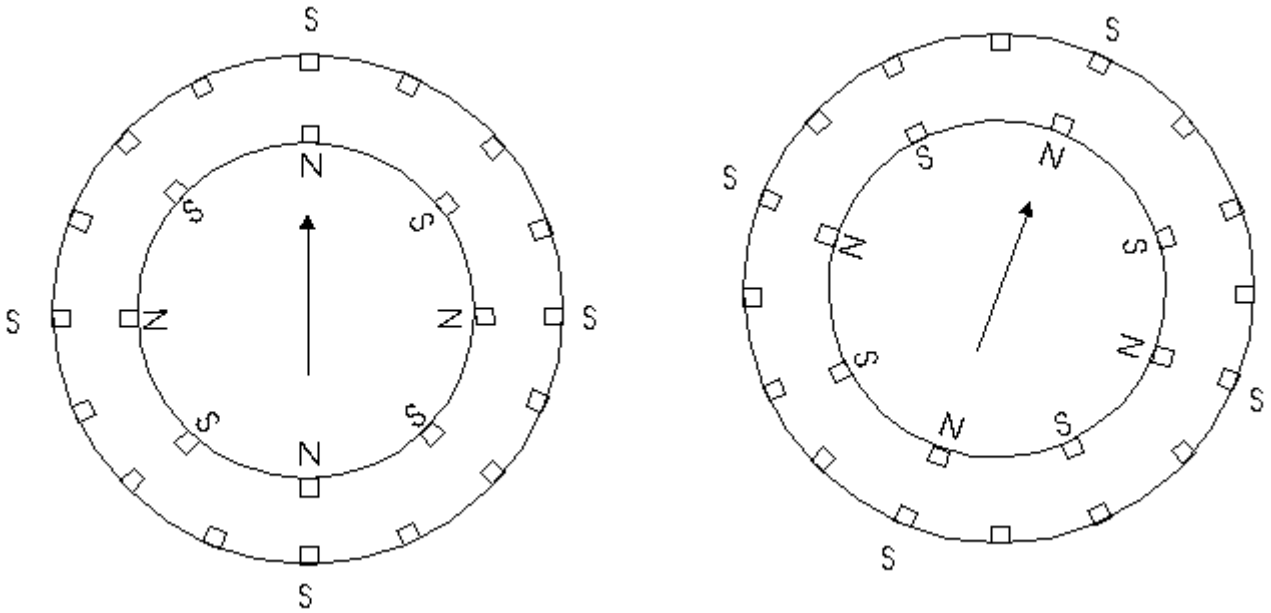
In the above example, we used a motor with a resolution of 90 degrees or demonstration purposes. In reality, this would not be a very practical motor for most applications. The average stepper motor's resolution -- the amount of degrees rotated per pulse -- is much higher than this. For example, a motor with a resolution of 5 degrees would move its rotor 5 degrees per step, thereby requiring 72 pulses (steps) to complete a full 360 degree rotation.

You may double the resolution of some motors by a process known as "half-stepping". Instead of switching the next electromagnet in the rotation on one at a time, with half stepping you turn on both electromagnets, causing an equal attraction between, thereby doubling the resolution. As you can see in Figure 2, in the first position only the upper electromagnet is active, and the rotor is drawn completely to it. In position 2, both the top and right electromagnets are active, causing the rotor to position itself between the two active poles. Finally, in position 3, the top magnet is deactivated and the rotor is drawn all the way right. This process can then be repeated for the entire rotation.



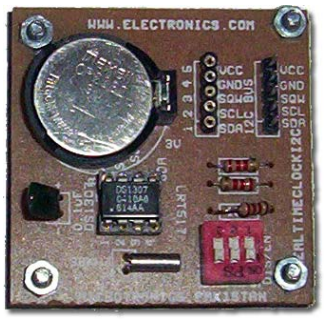
There are several types of stepper motors. 4-wire stepper motors contain only two electromagnets, however the operation is more complicated than those with three or four magnets, because the driving circuit must be able to reverse the current after each step. For our purposes, we will be using a 6-wire motor.

Unlike our example motors which rotated 90 degrees per step, real-world motors employ a series of mini-poles on the stator and rotor to increase resolution. Although this may seem to add more complexity to the process of driving the motors, the operation is identical to the simple 90 degree motor we used in our example. An example of a multipole motor can be seen in Figure 3. In position 1, the north pole of the rotor's perminant magnet is aligned with the south pole of the stator's electromagnet. Note that multiple positions are alligned at once. In position 2, the upper electromagnet is deactivated and the next one to its immediate left is activated, causing the rotor to rotate a precise amount of degrees. In this example, after eight steps the sequence repeats.



Appendix 5
Real Time Clock

Some applications need to keep track of time. Although timing can be achieved by PIC microcontroller, yet when microcontroller is busy in some task, it can not precisely update the system time. Moreover time will be implemented using memory variables, when power is turned off, these variables reset. Most applications that require an on board clock / calendar implement it using a standalone real time chip. There are numerous chips out there which can be used with microcontroller to keep the clock/calendar function. We are going to introduce here a popular chip, DS1307. although you can make the circuit yourself, Microtronics has a smart board, for this chip, that implements it as a stand alone device to be used with any microcontroller.



DS1307 uses I2C communication, and you can use this board on I2C bus, or on any other I/O lines, implementing I2C communication using software routines. We will implement this on PORTB, as the connector on board contains an additional pin called SQW, which is strictly speaking not part of I2C and therefore to connect the board on I2C bus on PIC Lab-II a slightly modified connector has to be made.

GENERAL DESCRIPTION

The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially through an I2C, bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24- hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply.

DETAILED DESCRIPTION

The DS1307 is a low-power clock/calendar with 56 bytes of battery-backed SRAM. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The DS1307 operates as a slave device on the I2C bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until a STOP condition is executed. When V_{CC} falls below 1.25 x V_{BAT}, the device terminates an access in progress and resets the device address counter. Inputs to the device will not be recognized at this time to prevent erroneous data from being written to the device from an out-of tolerance system. When V_{CC} falls below V_{BAT}, the device switches into a low-current battery-backup mode. Upon power -up, the device switches from battery to V_{CC} when V_{CC} is greater than V_{BAT} +0.2V and recognizes inputs when V_{CC} is greater than 1.25 x V_{BAT}. The block diagram in Figure 1 shows the main elements of the serial RTC.

RTC AND RAM ADDRESS MAP

Table 2 shows the address map for the DS1307 RTC and RAM registers. The RTC registers are located in address locations 00h to 07h. The RAM registers are located in address locations 08h to 3Fh. During a multi byte access, when the address pointer reaches 3Fh, the end of RAM space, it wraps around to location 00h, the beginning of the clock space.

CLOCK AND CALENDAR

The time and calendar information is obtained by reading the appropriate register bytes. Table 2 shows the RTC registers. The time and calendar are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the BCD format. The day-of-week register increments at

midnight. Values that correspond to the day of week are user-defined but must be sequential (i.e., if 1 equals Sunday, then 2 equals Monday, and so on.) Illogical time and date entries result in undefined operation. Bit 7 of Register 0 is the clock halt (CH) bit. When this bit is set to 1, the oscillator is disabled. When cleared to 0, the oscillator is enabled.

Note that the initial power-on state of all registers is not defined. Therefore, it is important to enable the oscillator (CH bit = 0) during initial configuration.

The DS1307 can be run in either 12-hour or 24-hour mode. Bit 6 of the hours register is defined as the 12-hour or 24-hour mode-select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10-hour bit (20 to 23 hours). The hours value must be re-entered whenever the 12/24-hour mode bit is changed. When reading or writing the time and date registers, secondary (user) buffers are used to prevent errors when the internal registers update. When reading the time and date registers, the user buffers are synchronized to the internal registers on any I2C START. The time information is read from these secondary registers while the clock continues to run. This eliminates the need to re-read the registers in case the internal registers update during a read. The divider chain is reset whenever the seconds register is written. Write transfers occur on the I2C acknowledge from the DS1307. Once the divider chain is reset, to avoid rollover issues, the remaining time and date registers must be written within one second.

Table 2. Timekeeper Registers

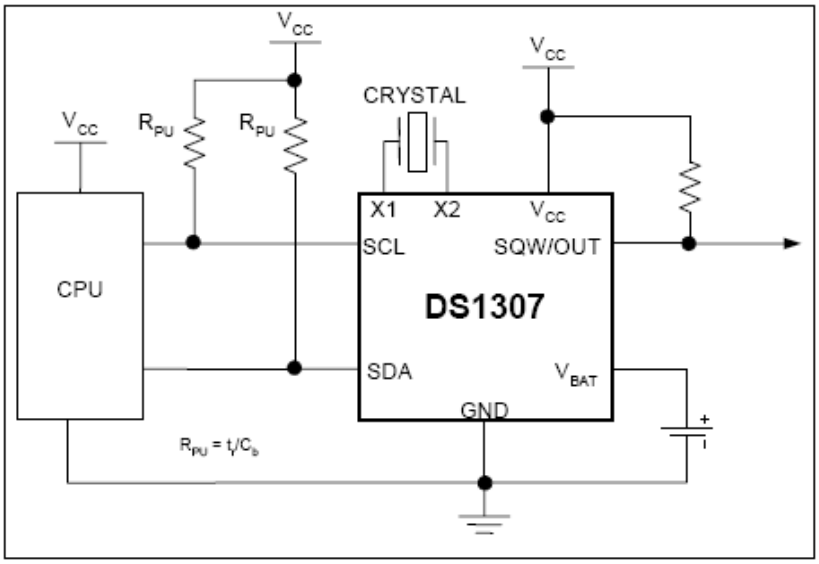
ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00H	CH	10 Seconds			Seconds				Seconds	00–59
01H	0	10 Minutes			Minutes				Minutes	00–59
02H	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03H	0	0	0	0	0	DAY			Day	01–07
04H	0	0	10 Date		Date				Date	01–31
05H	0	0	0	10 Month	Month				Month	01–12
06H	10 Year				Year				Year	00–99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08H-3FH									RAM 56 x 8	00H–FFH

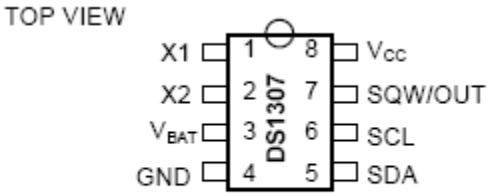
The DS1307 may operate in the following two modes:

1. **Slave Receiver Mode (Write Mode):** Serial data and clock are received through SDA and SCL. After each byte is received an acknowledge bit is transmitted. START and STOP conditions are recognized as the beginning and end of a serial transfer. Hardware performs address recognition after reception of the slave address and direction bit (see Figure 4). The slave address byte is the first byte received after the master generates the START condition. The slave address byte contains the 7-bit DS1307 address, which is 1101000, followed by the direction bit (R/W), which for a write is 0. After receiving and decoding the slave address byte, the DS1307 outputs an acknowledge on SDA. After the DS1307 acknowledges the slave address + write bit, the master transmits a word address to the DS1307. This sets the register pointer on the DS1307, with the DS1307 acknowledging the transfer. The master can then transmit zero or more bytes of data with the DS1307 acknowledging each byte received. The register pointer automatically increments after each data byte are written. The master will generate a STOP condition to terminate the data write.
2. **Slave Transmitter Mode (Read Mode):** The first byte is received and handled as in the slave receiver mode. However, in this mode, the direction bit will indicate that the transfer direction is reversed. The DS1307 transmits serial data on SDA while the serial clock is input on SCL. START and STOP conditions are recognized as the beginning and end of a serial transfer. The slave address byte is the first byte received after the START condition is generated by the master. The slave address byte contains the 7-bit DS1307 address, which is 1101000, followed by the direction bit (R/W), which is 1 for a read. After

receiving and decoding the slave address the DS1307 outputs an acknowledge on SDA. The DS1307 then begins to transmit data starting with the register address pointed to by the register pointer. If the register pointer is not written to before the initiation of a read mode the first address that is read is the last one stored in the register pointer. The register pointer automatically increments after each byte are read. The DS1307 must receive a Not Acknowledge to end a read.

TYPICAL OPERATING CIRCUIT





PDIP (300 mils)

Writing applications is fairly simple, however keep in mind that the data obtained from registers is in BCD format and not as binary number. Like seconds, say if its 49 seconds, bits 0 to 3 will contain 9 and bits 4,5,6 will contain 4 so you have to extract the numbers from the byte read.

Secondly and most importantly, when the chip is used for the very first time, always make sure that CH bit of register 0 is cleared, so that clock starts. After that always make sure whenever setting seconds, not to set this bit as logical 1 as this will halt the clock.