SMART CAR PARKING LOT

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

in

Electronics and Communication Engineering

By

Palak Nayyar(061088)

Deepak Semwal(061044)

Deepak Kumar(061043)

under the Supervision of

Prof. D.C. Kulshreshtha

May 2010

Jaypee University of Information Technology
Waknaghat, Solan - 173 234, Himachal Pradesh

Certificate

This is to certify that the project report entitled "**SMART CAR PARKING LOT**", submitted by Deepak Semwal (061044), Palak Nayyar (061088) and Deepak Kumar (061043) in partial fulfillment for the award of degree of Bachelor of Technology in Electronics and Communication Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

Date: 25/5/10

Prof. D.C. Kulshreshtha

Certified that this work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma

1. **Palak Nayyar**
   061088
2. **Deepak Semwal**
   061044
3. **Deepak Kumar**
   061043

ACKNOWLEDGEMENT

It has been a wonderful and intellectually stimulating experience working on the SMART CAR PARKING LOT which has much of practical implication in near future.

We gratefully acknowledge the management and administration of Jaypee University of Information Technology for providing the opportunity and hence the environment to initiate the project.

We would like to extend our first thanks to our HOD, Dr. S.V. Bhooshan, for helping us with the entire Lab works.

For Providing with the finest details of the subject ,We are greatly thankful to our project guide Prof. D.C. Kulshreshtha. He has provided us the way to get the job done ,not providing the exact way to do it ,but the complexities so that we can make better use of the existing knowledge and build up higher skills to meet the industry needs. His methodology of making the system strong from inside has taught us that output is not the end of project.

DATE

Deepak Semwal
(061044)

Palak Nayyar
(061088)

Deepak Kumar
(061043)

Table of Content

Abstract

In the recent years problems regarding availability of suitable parking spaces have increased by leaps and bounds. It's a common sighting that despite the availability of parking spaces, the parking lot is not utilized to its maximum potential. If there would have been a good alternative parking system ,the problems like time and hassle can solved to a great extent .Thus we have aimed our project to overcome all these problems and come up with a good parking system based on sensors and microcontrollers with less human intervention .Our project features a multistory lift based parking system which will further reduce problems regarding space and customer complexity .Since parking in the busiest of places is not very pleasing to the customers because of time it takes and overall chaos. The need for smart parking lots is growing everyday with ever increasing number of vehicles. This kind of parking system will grow up as the basis of upcoming urban society.

Chapter-1

Introduction and Components Used

1.1 Overview

With this project we have shown use of lift for multi level parking system. To make such a system efficient, a proper understanding about vacant parking spaces and appropriate floor to choose is needed .To carry this kind of job various kinds of sensors are being used .Every thing starts with Infrared Sensor which is employed at the gate to check entrance of a car.

When a car is being detected, a smart card already given to the user has to be swiped. The LCD will show the proper amount to be paid by customer according to his status. An ic 89cs52 micro controller is used to control the lift without any user intervention. A DC motor is used to rotate the lift in desired direction according to microcontroller. But at start a 'start' switch has to be pressed by the user. Few REED sensors are used to monitor the lift and few sensors to check the availability of the space. After pressing the start switch DC motor starts and when it reaches the first floor then firstly we sense the floor by REED sensor. REED sensor is a magnetic sensor, when magnet connected to the lift is near by the reed sensor then reed sensor is activated and provides a signal to the controller.

Controller sense the floor position and at the same time it checks the empty space position, if the empty space is there then lift stops automatically ,otherwise lift go forward to the next floor. In this project we use two floor and only two spaces for parking. Here in this project we monitor the floor and space every thing by the reed sensor. So we use magnetic proximity switches to monitor the space and position of the lift. We have designed this system for two floors only, but it can be expanded to as many floors as per designer's wish.

LCD and Microcontroller are two core components in our project. These are required to be programmed accordingly to get optimum results. Further insight into above mentioned will be in separate chapters dedicated to each of them.

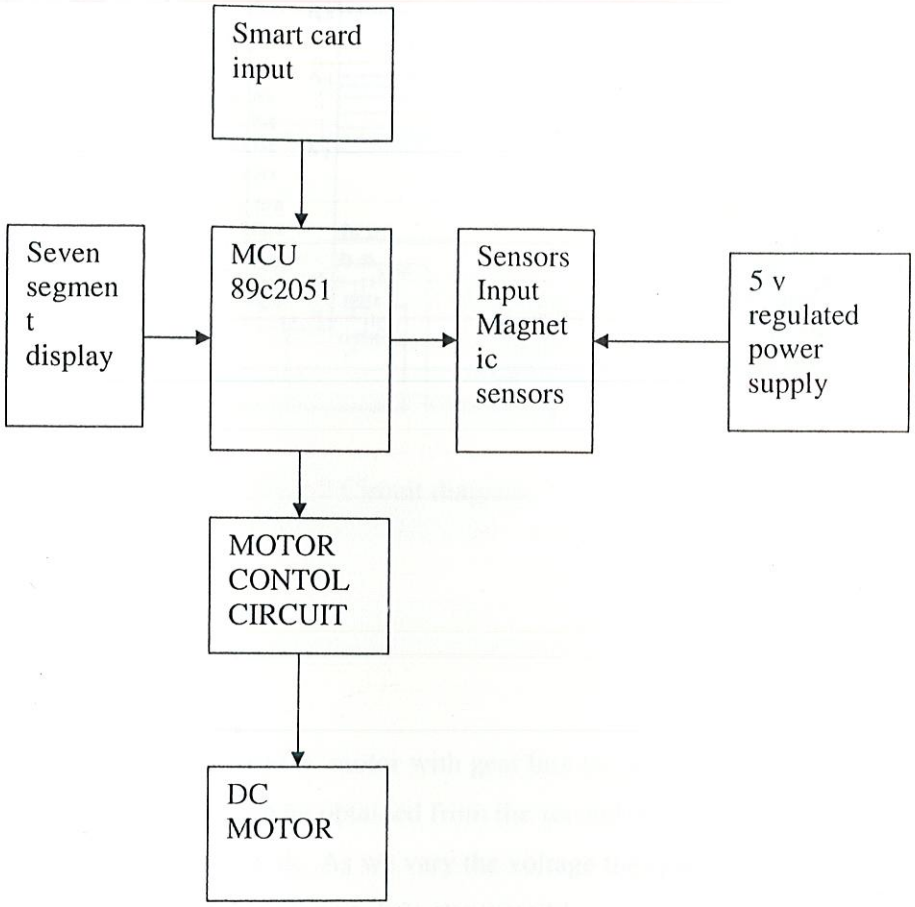

Fig1.1 Block diagram of car parking lift

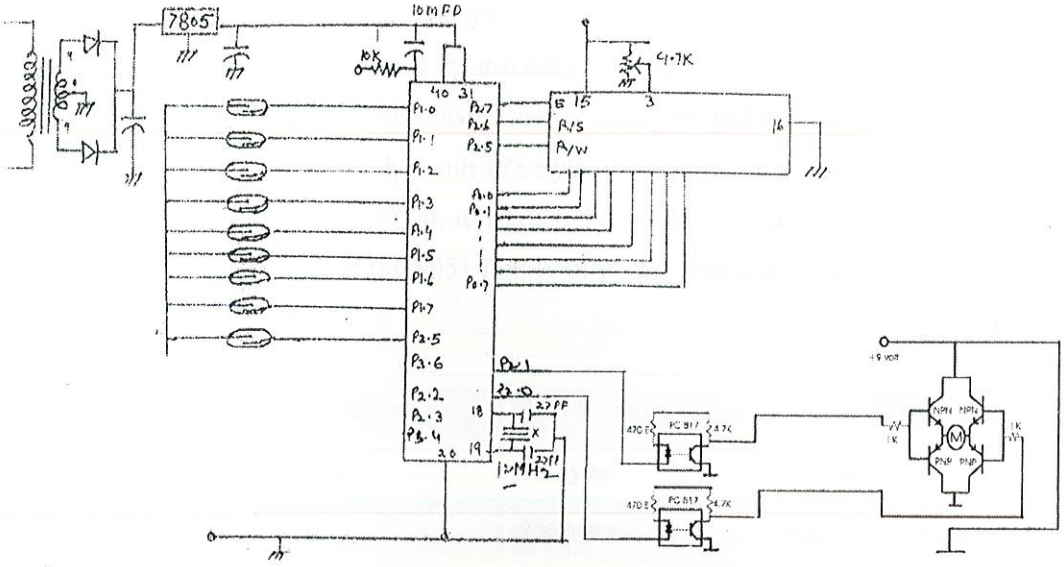

Fig 1.2 Circuit diagram

1.2 Components

1.2.1 DC Motor

Here in this project we use slow speed dc motor with gear box to reduce the speed of the platform. This type of gear motor can be obtained from the second hand machine. Supply voltage of this dc motor is 6 to 9 volt dc. As we vary the voltage the speed also varies. Current consumption of dc motor is 200 mas. It is also possible to use a stepper motor. If we use stepper motor then we require a high current supply. Normal stepper motor requires a minimum 1 A power supply. Limitations of brushed DC motors overcome by BLDC motors include lower efficiency and susceptibility of the commutator assembly to mechanical wear and consequent need for servicing, at the cost of potentionally less rugged and more complex and expensive control electronics.

1.2.2 Microcontroller Unit

A microcontroller is used to control the direction of motor with proximity switches and monitor the space of the parking. An IC 89c2051 controller is used to sense

all the logic. IC 89c2051 is a 20 pin member of the main 40 pin controller. We use 89c2051 because here in this project we use only 2 outputs for motor and control signal is only four. Out of four sensors, two sensors are for the floor and two sensors for the space. 89c2051 is a small micro controller with 128 byte of ram and 2 k byte of flash memory. To use efficiently the microcontroller, it needs to be suitably programmed.

There are two other member in the 8051 family of microcontrollers. They are the 8052 and the 8031.

Feature	8051	8052	8031
ROM	4K bytes	8K bytes	0K bytes
RAM	128 bytes	256 bytes	128
Timer	2	3	2
I/O pins	32	32	32
Serial port	1	1	1
Interrupt sources	6	8	6

Figure 1.3 Family members of 8051

1.2.3 Power Supply

The microcontroller we are using takes 12 volt as input supply so we have to use a step down transformer in order to convert voltage levels. As well as to convert AC to DC a rectifier is also employed.

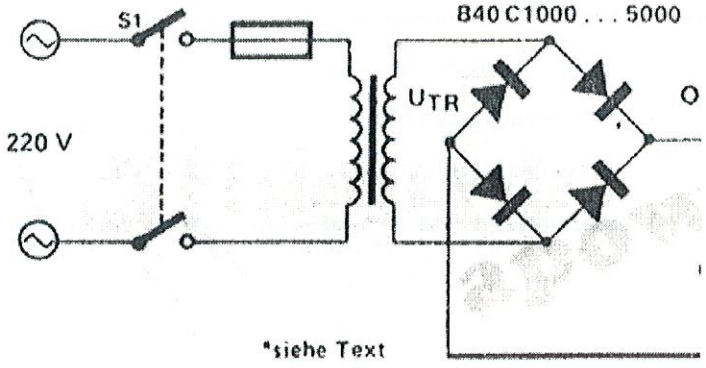

Figure 1.4 Power supply circuit

1.2.4 REED Sensors

The reed switch is an electrical switch operated by an applied magnetic field. It consists of a pair of contacts on ferrous metal reeds in a hermetically sealed glass envelope. The contacts may be normally open, closing when a magnetic field is present, or normally closed and opening when a magnetic field is applied. The reed switch contains a pair (or more) of magnetically, flexible, metal reeds whose end portions are separated by a small gap when the switch is open. Since the contacts of the reed switch are sealed away from the atmosphere, they are protected against atmospheric corrosion. The hermetic sealing of a reed switch make them suitable for use in explosive atmospheres where tiny sparks from conventional switches would constitute a hazard. One important quality of the switch is its sensitivity, the amount of magnetic field necessary to actuate it. Sensitivity is measured in units of Ampere-turns, corresponding to the current in a coil multiplied by the number of turns. Typical pull-in sensitivities for commercial devices are in the 10 to 60 AT range. The lower the AT, the more sensitive the reed switches. Also, smaller reed switches, which have smaller parts, are more sensitive to magnetic fields.

1.2.5 IR sensors

The emitted energy comes from an object and reaches the IR sensor through its optical system, which focuses the energy onto one or more photosensitive detectors. The detector then converts the IR energy into an electrical signal. The IR range falls between the visible portion of the spectrum and radio waves. IR wavelengths are usually expressed in microns, with the IR spectrum extending from 0.7 to 1000 microns. Only the 0.7-14 micron band is used for IR temperature measurement.

1.2.6 LCD

Frequently, an 8051 program must interact with the outside world using input and output devices that communicate directly with a human being. One of the most common devices attached to an 8051 is an LCD display. Some of the most common LCDs connected to the 8051 are 16x2 and 20x2 displays. This means 16 characters per line by 2 lines and 20 characters per line by 2 lines, respectively. Fortunately, a very popular standard exists which allows us to communicate with the vast majority of LCDs regardless of their manufacturer. The standard is referred to as HD44780U, which refers to the controller chip which receives data from an external source (in this case, the 8051) and communicates directly with the LCD.

The 44780 standard requires 3 control lines as well as either 4 or 8 I/O lines for the data bus. The user may select whether the LCD is to operate with a 4-bit data bus or an 8-bit data bus. If a 4-bit data bus is used, the LCD will require a total of 7 data lines (3 control lines plus the 4 lines for the data bus). If an 8-bit data bus is used, the LCD will require a total of 11 data lines (3 control lines plus the 8 lines for the data bus).

The three control lines are referred to as **EN**, **RS**, and **RW**.

1.2.7 Integrated Circuit

IC (Integrated Circuit) means that all the components of the circuit are fabricated on same chip. Digital ICs are a collection of resistors, diodes, and transistors fabricated on a single

piece of semiconductor, usually silicon called a substrate, which is commonly referred to as 'wafer'. The chip is enclosed in a protective plastic or ceramic package from which pins extend out connecting the IC to other device. Suffix N or P stands for dual-in-line (plastic package (DIP)) while suffix J or I stands for dual-in-lime ceramic package. Also the suffix for W stands for flat ceramic package.

The pins are numbered counter clockwise when viewed from the top of the package with respect to an identity notch or dot at one end of the chip.The manufacturer's name can usually be guessed from its logo that is printed on the IC. The IC type number also indicates the manufacturer's code. For e.g. DM 408 N SN 7404 indicates National Semiconductor and Texas Instruments.

Other examples are:

Fair Child	: UA, UAF
National Semiconductor	: DM, LM, LH, LF, and TA.
Motorola	: MC, MFC.
Sprague	: UKN, ULS, ULX.
Signetic	: N/s, NE/SE, and SU.
Burr-Brown	: BB.
Texas Instruments	: SN.

Various series with TTL logic family are:-

Standard TTL 74

Schottky TTL 74s

Low power Schottky 74LS

Advance Schottky 74AS

Advanced Low Power Schottky 74ALs

Also there are various series with CMOS logic family as metal state CMOS 40 or 140.

1.2.8 Diode

Diodes are polarized, which means that they must be inserted into the PCB the correct way round. This is because an electric current will only flow through them in one direction (like air will only flow one way trough a tyre valve). Diodes have two

connections, an anode and a cathode. The cathode is always identified by a dot, ring or some other mark. The PCB is often marked with a +sign for the cathode end. Diodes come in all shapes and sizes. They are often marked with a type number.

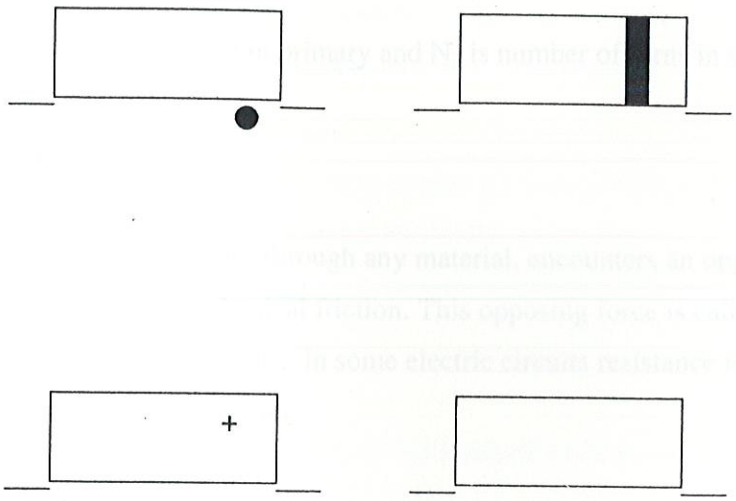

Figure 1.3 Diode

1.2.9 Transformer

Transformer works on the principle of mutual inductance. We know that if two coils or windings are placed on the core of iron, and if we pass alternating current in one winding, back emf or induced voltage is produced in the second winding. We know that alternating current always changes with the time. So if we apply AC voltage across one winding, a voltage will be induced in the other winding. Transformer works on this same principle. It is made of two windings wound around the same core of iron. The winding to which AC voltage is applied is called primary winding. The other winding is called as secondary winding.

Let V_1 volts be input alternating voltage applied to primary winding. I_1 Amp is input alternating current through primary winding. V_2 volt is output alternating voltage produced in the secondary. I_2 amp is the current flowing through the secondary.

Then relationship between input and output voltages is given by

$V_1/V_2 = N_1/N_2$

Relationship between input and output currents is

$I_1/I_2 = N_2/N_1$

(Where N_1 is no. of turns of coil in primary and N_2 is number of turns in secondary)

1.2.10 Resistors

The flow of charge (or current) through any material, encounters an opposing force similar in many respect to mechanical friction. This opposing force is called resistance of the material. It is measured in ohms. In some electric circuits resistance is deliberately introduced in the form of the resistor.

Resistors are of following types:

Wire wound resistors

Carbon resistors

Metal film resistors

- **Wire Wound Resistors:**

Wire wound resistors are made from a long (usually Ni-Chromium) wound on a ceramic core. Longer the length of the wire, higher is the resistance. So depending on the value of resistor required in a circuit, the wire is cut and wound on a ceramic core. This entire assembly is coated with a ceramic metal. Such resistors are generally available in power of 2 watts to several hundred watts and resistance values from 1ohm to 100k ohms. Thus wire wound resistors are used for high currents.

- **Carbon Resistors:**

Carbon resistors are divided into three types:

a. Carbon composition resistors are made by mixing carbon grains with Binding material (glue) and modulated in the form of rods. Wire leads are inserted at the two ends. After this an insulating material seals the resistor. Resistors are available in power ratings of 1/10, 1/8, 1/4, 1/2, 1.2 watts and values from 1 ohm to 20 ohms.

b. Carbon film resistors are made by deposition carbon film on a ceramic rod. They are cheaper than carbon composition resistors.

c. Cement film resistors are made of thin carbon coating fired onto a solid ceramic substrate. The main purpose is to have more precise resistance values and greater stability with heat. They are made in a small square with leads

- **Metal Film Resistors:**

They are also called thin film resistors. They are made of a thin metal coating deposited on a cylindrical insulating support. The high resistance values are not precise in value however such resistors are free of inductance effect that is common in wire wound resistors at high frequency.

- **Variable Resistors:**

Potentiometer is a resistor where values can be set depending on the requirement. Potentiometer is widely used in electronics systems. Examples are volume control, tons control, brightness and contrast control of radio or T.V. sets.

- **Fusible Resistors:**

These resistors are wire wound type and are used in T.V. circuits for protection. They have resistance of less than 15 ohms. Their function is similar to a fuse made to blow off whenever current in the circuit exceeds the limit. Resistance of a wire is directly proportional to its length and inversely proportional to its thickness. Resistance of a wire is directly proportional to its length and inversely proportional to its thickness.

COLOUR	NUMBER	MULTIPLIER	COLOUR	TOLERANCE
Black	0	10^0	Gold	5%
Brown	1	10^1	Silver	10%
Red	2	10^2		
Orange	3	10^3	No colour	20%
Yellow	4	10^4		
Green	5	10^5		
Blue	6	10^6		
Violet	7	10^7		
Grey	8	10^8		
White	9	10^9		
Gold		10^{-1}		
Silver		10^{-2}		

Table 1.1 Colour Coding of resistors

1.2.11 Capacitors

A capacitor can store charge, and its capacity to store charge is called capacitance. Capacitors consist of two conducting plates, separated by an insulating material (known as dielectric). The two plates are joined with two leads. The dielectric could be air, mica, paper, ceramic, polyester, polystyrene, etc. This dielectric gives name to the capacitor. Like paper capacitor, mica capacitor etc.

Capacitors can be broadly classified in two categories, i.e., Electrolytic capacitors and Non-Electrolytic capacitors.

- **Electrolytic Capacitor:**

Electrolytic capacitors have an electrolyte as a dielectric. When such an electrolyte is charged, chemical changes takes place in the electrolyte. If its one plate is charged positively, same plate must be charged positively in future. We call such capacitors as polarized.

- **Mica Capacitors:**

It is sandwich of several thin metal plates separated by thin sheets of mica. Alternate plates are connected together and leads attached for outside connections. The total assembly is encased in a plastic capsule or Bakelite case. Such capacitors have small capacitance value (50 to 500pf) and high working voltage (500V and above). The mica capacitors have excellent characteristics under stress of temperature variation and high voltage application. These capacitors are now replaced by ceramic capacitors.

- **Ceramic Capacitor:**

Such capacitors have disc or hollow tabular shaped dielectric made of ceramic material such as titanium dioxide and barium titanate. Thin coating of silver compounds is deposited on both sides of dielectric disc, which acts as capacitor plates. Leads are attached to each sides of the dielectric disc and whole unit is encapsulated in a moisture proof coating. Disc type capacitors have very high value up to 0.001uf. Their working voltages range from 3V to 60000V. These capacitors have very low leakage current. Breakdown voltage is very high.

- **Paper Capacitor:**

It consists of thin foils, which are separated by thin paper or waxed paper. The sandwich of foil and paper is then rolled into a cylindrical shape and enclosed in a paper tube or

encased in a plastic capsules. The lead at each end of the capacitor is internally attached to the metal foil. Paper capacitors have capacitance ranging from 0.0001uf to 2.0uf and working voltage rating as high as 2000V.

Item	Qty.	ID
OPTOCOUPLER	2	IC 817
Microcontroller Unit	1	IC 89C2051
CRYSTAL	1	12 MHZ
CERAMIC	2	27 PF
CONNECTING WIRES		
IC BASE	1	20 PIN
IC BASE	1	16 PIN
LEDS	5	RED 15 MM
REGULATOR	1	7805
CAP	1	1000 MICROFARAD
TR	2	TR548
TR	2	TR 558
MICRO SWITCHES	2	PUSH TO ON
REED SENSOR	6	MAGNETIC

Table 1.2 Components used in our project

CHAPTER -2

Micro Controller 8051

2.1 The 8051

The 8051 developed and launched in the early 80`s, is one of the most popular micro controller in use today. It has a reasonably large amount of built in ROM and RAM. In addition it has the ability to access external memory.

The generic term `8x51` is used to define the device. The value of x defining the kind of ROM, i.e. x=0, indicates none, x=3, indicates mask ROM, x=7, indicates EPROM and x=9 indicates EEPROM or Flash.

A note on ROM

The early 8051, namely the 8031 was designed without any ROM. This device could run only with external memory connected to it. Subsequent developments lead to the development of the PROM or the programmable ROM. This type had the disadvantage of being highly unreliable. The next in line, was the EPROM or Erasable Programmable ROM. These devices used ultraviolet light erasable memory cells. Thus a program could be loaded, tested and erased using ultra violet rays. A new program could then be loaded again. An improved EPROM was the EEPROM or the electrically erasable PROM. This does not require ultra violet rays, and memory can be cleared using circuits within the chip itself. Finally there is the FLASH, which is an improvement over the EEPROM. While the terms EEPROM and flash are sometimes used interchangeably, the difference lies in the fact that flash erases the complete memory at one stroke, and not act on the individual cells. This results in reducing the time for erasure

2.1.1 Different microcontrollers in market

- **PIC**

One of the famous microcontrollers used in the industries. It is based on RISC Architecture which makes the microcontroller process faster than other microcontroller.

- **INTEL**

These are the first to manufacture microcontrollers. These are not as sophisticated other microcontrollers but still the easiest one to learn.

- **ATMEL**

Atmel's AVR microcontrollers are one of the most powerful in the embedded industry. This is the only microcontroller having 1kb of ram even the entry stage. But it is unfortunate that in India we are unable to find this kind of microcontroller.

2.1.2 Intel 8051

Intel 8051 is CISC architecture which is easy to program in assembly language and also has a good support for High level languages. The memory of the microcontroller can be extended up to 64k.This microcontroller is one of the easiest microcontrollers to learn. The 8051 microcontroller is in the field for more than 20 years. There are lots of books and study materials are readily available for 8051.

2.1.3 Derivatives

The best thing done by Intel is to give the designs of the 8051 microcontroller to everyone. So it is not the fact that Intel is the only manufacture for the 8051 there more than 20 manufactures, with each of minimum 20 models. Literally there are hundreds of models of 8051 microcontroller available in market to choose. Some of the major manufactures of 8051 are

- Atmel
- Philips

- **Philips**

The Philips's 8051 derivatives has more number of features than in any microcontroller. The costs of the Philips microcontrollers are higher than the Atmel's which makes us to choose Atmel more often than Philips

- **Dallas**

Dallas has made many revolutions in the semiconductor market. Dallas's 8051 derivative is the fastest one in the market. It works 3 times as fast as a 8051 can process. But we are unable to get more in India.

- **Atmel**

These people were the one to master the flash devices. They are the cheapest microcontroller available in the market. Atmel's even introduced a 20pin variant of 8051 named 2051. The Atmel's 8051 derivatives can be got in India less than 70 rupees. There are lots of cheap programmers available in India for Atmel. So it is always good for students to stick with 8051 when you learn a new microcontroller.

2.2 Architecture

Architecture is must to learn because before learning new machine it is necessary to learn the capabilities of the machine. This is some thing like before learning about the car you cannot become a good driver. The architecture of the 8051 is given below. The 8051 doesn't have any special feature than other microcontroller. The only feature is that it is easy to learn. Architecture makes us to know about the hardware features of the microcontroller.

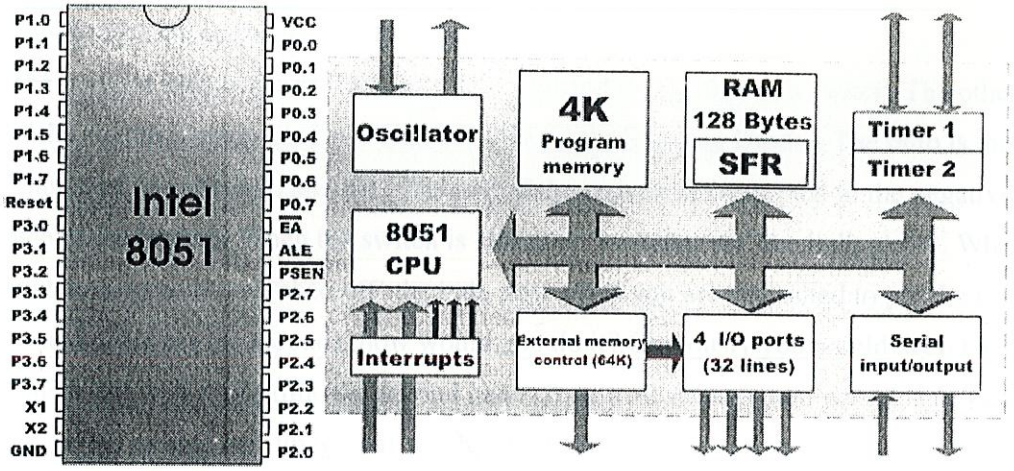

Figure 2.1 pin diagram of 8051 microcontroller

The features of the 8051 are

- 4K Bytes of Flash Memory
- 128 x 8-Bit Internal RAM
- Fully Static Operation: 1 MHz to 24 MHz
- 32 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- Six Interrupt Sources (5 Vectored)
- Programmable Serial Channel

- Low Power Idle and Power Down Modes

The 8051 has a 8-Bit CPU that means it is able to process 8 bit of data at a time. 8051 has 235 instructions.Let's now move on to a practical example. We shall work on a simple practical application and using the example as a base, shall explore the various features of the 8051 microcontroller.

Consider an electric circuit as follows,

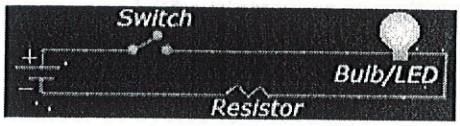

Figure 2.2 Simple electric Bulb Circuit

The positive side (+ve) of the battery is connected to one side of a switch. The other side of the switch is connected to a bulb or LED (Light Emitting Diode). The bulb is then connected to a resistor, and the other end of the resistor is connected to the negative (-ve) side of the battery. When the switch is closed or 'switched on' the bulb glows. When the switch is open or 'switched off' the bulb goes off.if you are instructed to put the switch on and off every 30 seconds, how would you do it? Obviously you would keep looking at your watch and every time the second hand crosses 30 seconds you would keep turning the switch on and off. Imagine if you had to do this action consistently for a full day. Do you think you would be able to do it? Now if you had to do this for a month, a year?? No way, you would say!

The next step would be, then to make it automatic. This is where we use the Microcontroller. But if the action has to take place every 30 seconds, how will the microcontroller keep track of time?

2.3 Execution time

Look at the following instruction

clr p1.0

this is an assembly language instruction. It means we are instructing the microcontroller

to put a value of 'zero' in bit zero of port one. This instruction is equivalent to telling the microcontroller to switch on the bulb. The instruction then to instruct the microcontroller to switch off the bulb is set p1.0

Set p1.0

This instructs the microcontroller to put a value of 'one' in bit zero of port one.
Don't worry about what bit zero and port one means. We shall learn it in more detail as we proceed. There are a set of well defined instructions, which are used while communicating with the microcontroller. Each of these instructions requires a standard number of cycles to execute. The cycle could be one or more in number.
How is this time then calculated?

The speed with which a microcontroller executes instructions is determined by what is known as the crystal speed. A crystal is a component connected externally to the microcontroller. The crystal has different values, and some of the used values are 6MHZ, 10MHZ, and 11.059 MHz etc.
Thus a 10MHZ crystal would pulse at the rate of 10,000,000 times per second.
The time is calculated using the formula
No of cycles per second = Crystal frequency in HZ / 12.
For a 10MHZ crystal the number of cycles would be,
10,000,000/12=833333.33333 cycles.
This means that in one second, the microcontroller would execute 833333.33333 cycles.
Therefore for one cycle, what would be the time? Try it out.
The instruction clr p1.0 would use one cycle to execute. Similarly, the instruction setb p1.0 also uses one cycle.
So go ahead and calculate what would be the number of cycles required to be executed to get a time of 30 seconds!
Getting back to our bulb example, all we would need to do is to instruct the microcontroller to carry out some instructions equivalent to a period of 30 seconds, like counting from zero upwards, then switch on the bulb, carry out instructions equivalent to 30 seconds and switch off the bulb. Just put the whole thing in a loop, and you have a

never ending on-off sequence.

2.4 Features of the 8051 core

Let us now have a look at the features of the 8051 core, keeping the above example as a reference

2.4.1 8-bit CPU.(Consisting of the 'A' and 'B' registers)

Most of the transactions within the microcontroller are carried out through the 'A' register, also known as the Accumulator. In addition all arithmetic functions are carried out generally in the 'A' register. There is another register known as the 'B' register, which is used exclusively for multiplication and division.

Thus an 8-bit notation would indicate that the maximum value that can be input into these registers is '11111111'. Puzzled?

The value is not decimal 111, 11,111! It represents a binary number, having an equivalent value of 'FF' in Hexadecimal and a value of 255 in decimal.We shall read in more detail on the different numbering systems namely the Binary and Hexadecimal system in our next module.

2.4.2 4K on-chip ROM

Once you have written out the instructions for the microcontroller, where do you put these instructions? Obviously you would like these instructions to be safe, and not get deleted or changed during execution. Hence you would load it into the 'ROM'.

The size of the program you write is bound to vary depending on the application, and the number of lines. The 8051 microcontroller gives you space to load up to 4K of program size into the internal ROM. 4K, that's all? Well just wait. You would be surprised at the

amount of stuff you can load in this 4K of space. Of course you could always extend the space by connecting to 64K of external ROM if required.

2.4.3 128 bytes on-chip RAM

This is the space provided for executing the program in terms of moving data, storing data etc.

2.4.4 32 I/O lines. (Four- 8 bit ports, labeled P0, P1, P2, P3)

In our bulb example, we used the notation p1.0. This means bit zero of port one. One bit controls one bulb. Thus port one would have 8 bits. There are a total of four ports named p0, p1, p2, p3, giving a total of 32 lines. These lines can be used both as input or output.

2.4.5 Two 16 bit timers / counters

A microcontroller normally executes one instruction at a time. However certain applications would require that some event has to be tracked independent of the main program. The manufacturers have provided a solution, by providing two timers. These timers execute in the background independent of the main program. Once the required time has been reached, (remember the time calculations described above?), they can trigger a branch in the main program. These timers can also be used as counters, so that they can count the number of events, and on reaching the required count, can cause a branch in the main program.

2.4.6 Full Duplex serial data receiver / transmitter

The 8051 microcontroller is capable of communicating with external devices like the PC etc. Here data is sent in the form of bytes, at predefined speeds, also known as baud rates. The transmission is serial, in the sense, one bit at a time.

2.4.7 5 interrupt sources with two priority levels (Two external and three internal)

During the discussion on the timers, we had indicated that the timers can trigger a branch in the main program. However, what would we do in case we would like the microcontroller to take the branch, and then return back to the main program, without having to constantly check whether the required time / count has been reached?
This is where the interrupts come into play. These can be set to either the timers, or to some external events. Whenever the background program has reached the required criteria in terms of time or count or an external event, the branch is taken, and on completion of the branch, the control returns to the main program. Priority levels indicate which interrupt is more important, and needs to be executed first in case two interrupts occur at the same time.

2.4.8 On-chip clock oscillator

This represents the oscillator circuits within the microcontroller. Thus the hardware is reduced to just simply connecting an external crystal, to achieve the required pulsing rate.

2.4.9 Micro controller used in our project

In this project we use micro controller to control the direction of motor with proximity switches and monitor the space of the parking. Here in this project we use ic 89c2051 controller to sense all the logic. 89c2051 is a 20 pin member of the main 40 pin controller. We use 89c2051 because here in this project we use only 2 outputs for motor and control signal is only four. Out of four sensors, two sensors are for the floor and two sensors for the space. 89c2051 is a small micro controller with 128 byte of ram and 2 k byte of flash memory.

2.5 PIN FUNCTION OF IC 89C51

- Supply pin of this ic is pin no 40. Normally we apply a 5 volt regulated dc power supply to this pin. For this purpose either we use step down transformer power supply or we use 9 volt battery with 7805 regulator.
- Ground pin of this ic is pin no 20. Pin no 20 is normally connected to the ground pin (normally negative point of the power supply).
- XTAL is connected to the pin no 18 and pin no 19 of this ic. The quartz crystal oscillator connected to XTAL1 and XTAL2 PIN .These pins also needs two capacitors of 30 pf value .One side of each capacitor is connected to crystal and other pin is connected to the ground point. Normally we connect a 12 MHz or 11.0592 MHz crystal with this ic. But we use crystal up to 20 MHz to this pins
- RESET PIN. Pin no 9 is the reset pin of this ic. It is an active high pin. On applying a high pulse to this pin, the micro controller will reset and terminate all activities. This is often referred to as a power on reset. The high pulse must be high for a minimum of 2 machine cycles before it is allowed to go low.
- PORT0 occupies a total of 8 pins. Pin no 32 to pin no 39. It can be used for input or output. We connect all the pins of the port 0 with the pull up resistor (10 k ohm) externally. This is due to fact that port 0 is an open drain mode. It is just like an open collector transistor.
- PORT1.All the ports in micro controller is 8 bit wide pin no 1 to pin no 8 because it is a 8 bit controller. All the main register and SFR all is mainly 8 bit wide. Port 1 is also occupies 8 pins. But there is no need of pull up resistor in this port. Upon reset port 1 acts as a input port. Upon reset all the ports act as a input port
- PORT2.port 2 also has 8 pins. It can be used as an input or output. There is no need of any pull up resistor to this pin.
- PORT 3. Port3 occupies a total 8 pins from pin no 10 to pin no 17. It can be used as input or output. Port 3 does not require any pull up resistor. The same as port 1 and port2. Port 3 is configured as an output port on reset. Port 3 has the additional

function of providing some important signals such as interrupts. Port 3 also use for serial communication.

- ALE is an output pin and is active high. When connecting an 8031 to external memory, port 0 provides both address and data. In other words, the 8031 multiplexes address and data through port 0 to save pins. The ALE pin is used for demultiplexing the address and data by connecting to the ic 74ls373 chip.
- PSEN stands for program store enable. In an 8031 based system in which an external rom holds the program code, this pin is connected to the OE pin of the rom.
- EA In 89c51 8751 or any other family member of the ateml 89c51 series all come with on-chip rom to store programs, in such cases the EA pin is connected to the Vcc. For family member 8031 and 8032 is which there is no on chip rom, code is stored in external memory and this is fetched by 8031. In that case EA pin must be connected to GND pin to indicate that the code is stored externally.

2.6 SPECIAL FUNCTION REGISTER (SFR) ADDRESSES

ACC	ACCUMULATOR	0E0H
B	B REGISTER	0F0H
PSW	PROGRAM STATUS WORD	0D0H
SP	STACK POINTER	81H
DPTR	DATA POINTER 2 BYTES	
DPL	LOW BYTE OF DPTR	82H
DPH	HIGH BYTE OF DPTR	83H
P0	PORT0	80H
P1	PORT1	90H
P2	PORT2	0A0H

P3	PORT3	0B0H
TMOD	TIMER/COUNTER MODE CONTROL	89H
TCON	TIMER/ COUNTER CONTROL	88H
TH0	TIMER 0 HIGH BYTE	8CH
TLO	TIMER 0 LOW BYTE	8AH
TH1	TIMER 1 HIGH BYTE	8DH
TL1	TIMER 1 LOW BYTE	8BH
SCON	SERIAL CONTROL	98H
SBUF	SERIAL DATA BUFFER	99H
PCON	POWER CONTROL	87H

2.7 Instructions

2.7.1 Single bit instructions

SETBBIT	SET THE BIT =1
CLRBIT	CLEAR THE BIT =0
CPLBIT	COMPLIMENT THE BIT 0 =1, 1=0
JBBIT, TARGET	JUMP TO TARGET IF BIT =1
JNBBIT, TARGET	JUMP TO TARGET IF BIT =0
JBCBIT, TARGET	JUMP TO TARGET IF BIT =1 &THEN CLEAR THE BIT

2.7.2 MOV Instructions

MOV D, S	Copy the data from(S) source to D (destination)
MOV R0, A	Copy contents of A into Register R0
MOV R1, A	Copy contents of A into register R1

MOV A, R3 Copy contents of Register R3 into Accumulator.

Direct loading through MOV

MOV A, #23 H ; Direct load the value of 23h in A

MOV R0, #12h ; direct load the value of 12h in R0

MOV R5, #0F9H ; Load the F9 value in the Register R5

2.7.3 ADD Instruction

ADD instruction adds the source byte to the accumulator (A) and place the result in the Accumulator.

MOV A, #25H

ADD A, #42H ; BY this instruction we add the value 42h in Accumulator (42H+ 25H)

ADDA, R3 ; by this instructions we move the data from register r3 to accumulator and then add the contents of the register into accumulator.

2.7.4 SUB ROUTINE CALL FUNCTION

ACALL, TARGET ADDRESS

By this instruction we call subroutines with a target address within 2k bytes from the current program counter.

LCALL, TARGET ADDRESS.

ACALL is a limit for the 2 k byte program counter, but for up to 64k byte we use LCALL instructions.. Note that LCALL is a 3 byte instruction. ACALL is a two byte instruction.

AJMP TARGET ADDRESS

AJMP stand for absolute jump. It transfers program execution to the target address unconditionally. The target address for this instruction must be within 2 k byte of program memory.

LJMP is also for absolute jump. It transfer program execution to the target address un conditionally. This is a 3 byte instructions LJMP jump to any address within 64 k byte location.

2.7.5 Instructions related to carry

JC TARGET JUMP TO THE TARGET IF CY FLAG =1

JNC TARGET JUMP TO THE TARGET ADDRESS IF CY FLAG IS = 0

2.7.6 Instructions related to jump with accumulator

JZ TARGET JUMP TO TARGET IF A = 0

JNZ TARGET JUMP IF ACCUMULATOR IS NOT ZERO

This instruction jumps if register A has a value other than zero

2.7.7 Instructions related to rotate

RL A ROTATE LEFT THE ACCUMULATOR

BY this instruction we rotate the bits of A left. The bits rotated out of A are rotated back into A at the opposite end

RR A

By this instruction we rotate the contents of the accumulator from right to left from LSB to MSB

RRC A

This is same as RR A but difference is that the bit rotated out of register first enter in to carry and then enter into MSB

RLC A ROTATE A LEFT THROUGH CARRY

Same as above but shift the data from MSB to carry and carry to LSB

RET

This is return from subroutine. This instruction is used to return from a subroutine previously entered by instructions LCALL and ACALL.

RET1

This is used at the end of an interrupt service routine. We use these instructions after interrupt routine,

PUSH.

This copies the indicated byte onto the stack and increments SP by. This instruction supports only direct addressing mode.

POP. POP FROM STACK

This copies the byte pointed to be SP to the location whose direct address is indicated, and decrements SP by 1. Notice that this instruction supports only direct addressing mode.

2.7.8 DPTR Instructions

MOV DPTR, #16 BIT VALUE

LOAD DATA POINTER

This instructions load the 16 bit dptr register with a 16 bit immediate value

MOV C A,@A+DPTR

This instruction moves a byte of data located in program ROM into register A. This allows us to put strings of data, such as look up table elements.

MOVC A,@A+PC

This instruction moves a byte of data located in the program area to A. the address of the desired byte of data is formed by adding the program counter (PC) register to the original value of the accumulator.

INC BYTE

This instruction adds 1 to the register or memory location specified by the operand.

INC A

INC Rn

INC DIRECT

DEC BYTE

This instruction subtracts 1 from the byte operand. Note that CY is unchanged

DEC A

DEC Rn

DEC DIRECT

2.7.9 Arithmetic Instructions

ANL dest-byte, source-byte

This perform a logical AND operation

This performs a logical AND on the operands, bit by bit, storing the result in the destination. Notice that both the source and destination values are byte –size only

DIV AB

This instruction divides a byte accumulator by the byte in register B. It is assumed that both register A and B contain an unsigned byte. After the division the quotient will be in register A and the remainder in register B.

2.15 Arithmetic Operations

Mnemonic	Description	Size	Cycles
ADD A, Rn	Add register to Accumulator (ACC)	1	1
ADD A, direct	Add direct byte to ACC	2	1
ADD A, @Ri	Add indirect RAM to ACC	1	1
ADD A, #data	Add immediate data to ACC	2	1
ADDC A, Rn	Add register to ACC with carry	1	1
ADDC A, direct	Add direct byte to ACC with carry.	2	1
ADDC A, @Ri	Add indirect RAM to ACC with carry.	1	1
ADDC A, #data	Add immediate data to ACC with carry.	2	1
SUBB A, Rn	Subtract register from ACC with borrows.	1	1
SUBB A, direct	Subtract direct byte from ACC with borrow	2	1
SUBB A, @RI	Subtract indirect RAM from ACC with borrow.	1	1
SUBB A, #data	Subtract immediate data from ACC with borrow.	2	1
INC A	Increment ACC	1	1
NC Rn	Increment register	1	1
INC direct	Increment direct byte	2	1
@Ri	Increment indirect RAM	1	1

DEC A	Decrement ACC	1	1
DEC Rn	Decrement register	1	1
DEC direct	Decrement direct byte	2	1
DEC @Ri	Decrement indirect RAM	1	1
INC DPTR	Increment data pointer.	1	2
MUL AB	Multiply A and B Result: A <- low byte, B <- high byte.	1	4
DIV AB	Divide A by B Result: A <- whole part, B <- remainder.	1	4
DA A	Decimal adjusts ACC.	1	1

Logical Operations

Mnemonic	Description	Size	Cycles
ANL A, Rn	AND Register to ACC.	1	1
ANL A, direct	AND direct byte to ACC	2	1
ANL A, @RI	AND indirect RAM to ACC	1	1
ANL A, #data	AND immediate data to ACC	2	1
ANL direct, A	AND ACC to direct byte.	2	1
ANL direct, #data	AND immediate data to direct byte	3	2
ORL A, Rn	OR Register to ACC.	1	1
ORL A, direct	OR direct byte to ACC	2	1
ORL A, @RI	OR indirect RAM to ACC	1	1
ORL A, #data	OR immediate data to ACC	2	1
ORL direct, A	OR ACC to direct byte.	2	1
ORL direct, #data	OR immediate data to direct byte	3	2

XRL A,Rn	Exclusive OR Register to ACC	1	1
XRL A, direct	Exclusive OR direct byte to ACC	2	1
XRL A, @ RI	Exclusive OR indirect RAM to ACC.	1	1
XRL A,#data	Exclusive OR immediate data to ACC.	2	1
XRL direct, A	Exclusive OR ACC to direct byte.	2	1
XRL direct, #data	XOR immediate data to direct byte	3	2
CLR A	Clear ACC (set all bits to zero).	1	1
CPL A	Compliment ACC	1	1
RL A	Rotate ACC left.	1	1
RLC A	Rotate ACC left through carry.	1	1
RR A	Rotate ACC right	1	1
RRC A	Rotate ACC right through carry.	1	1
SWAP A	Swap nibbles within ACC.	1	1

Data Transfer

Mnemonic	**Description**	**Size**	**Cycles**
MOV A, Rn	Move register to ACC	1	1
MOV A, direct	Move direct byte to ACC	2	1
MOV A, @Ri	Move indirect RAM to ACC	1	1
MOV A, #data	Move immediate data to ACC	2	1
MOV Rn, A	Move ACC to register.	1	1
MOV Rn, direct	Move direct byte to register.	2	2

MOV Rn, #data	Move immediate data to register.	2	1
MOV direct, A	Move ACC to direct byte.	2	1
MOV direct, Rn	Move register to direct byte.	2	2
MOV direct, direct	Move direct byte to direct byte	3	2
MOV direct, @Ri	Move indirect RAM to direct byte.	2	2
MOV direct, #data	Move immediate data to direct byte	3	2
MOV @Ri, A	Move ACC to indirect RAM.	1	1
MOV @Ri, direct	Move direct byte to indirect RAM.	2	2
MOV @ RI, #data	Move immediate data to indirect RAM.	2	1
MOV DPTR,#data16	Move immediate 16 bit data to data pointer register.	3	2
MOVC A,@A+DPTR	Move code byte relative to DPTR to ACC (16 bit address)	1	2
MOVC A,@A+PC	Move code byte relative to PC to ACC	1	2
MOVX A,@Ri	Move external RAM to ACC (8 bit address).	1	2
MOVX A,@DPTR	Move external RAM to ACC (16 bit address).	1	2
MOVX @Ri,A	Move ACC to external RAM (8 bit address).	1	2
MOVX @DPTR,A	Move ACC to external RAM (16 bit address).	1	2
PUSH direct	Push direct byte onto stack.	2	2
POP direct	Pop direct byte from stack.	2	2
XCH A,Rn	Exchange register with ACC.	1	1
XCH A,direct	Exchange direct byte with ACC.	2	1
XCH A,@Ri	Exchange indirect RAM with ACC.	1	1
XCHD A,@Ri	Exchange low order nibble of indirect RAM with low order nibble of ACC	1	1

Boolean Variable Manipulation

Mnemonic	Description	Size	Cycles
CLR C	Clear carry flag.	1	1
CLR bit	clear direct bit.	2	1
SETB C	Set carry flag.	1	1
SETB	bit Set direct bit	2	1
CPL C	Compliment carry flag.	1	1
CPL bit	Compliment direct bit.	2	1
ANL C,bit	AND direct bit to carry flag.	2	2
ANL C,/bit	AND compliment of direct bit to carry.	2	2
ORL C,bit	OR direct bit to carry flag.	2	2
ORL C,/bit	OR compliment of direct bit to carry.	2	2
MOV C,bit	Move direct bit to carry flag.	2	1
MOV bit,C	Move carry to direct bit.	2	2
JC rel	Jump if carry is set.	2	2
JNC rel	Jump if carry is not set.	2	2
JB bit,rel	Jump if direct bit is set.	3	2
JNB bit,rel	Jump if direct bit is not set.	3	2
JBC bit,rel	Jump if direct bit is set & clear bit.	3	2

Program Branching

Mnemonic	Description		Size	Cycles
ACALL	addr11	Absolute subroutine call	2	2
LCALL	addr16	Long subroutine call	3	2
RET	Return from subroutine.		1	2
RETI	Return from interrupt.		1	2
AJMP addr11	Absolute jump.		2	2
LJMP addr16	Long jump.		3	2
SJMP rel	Short jump (relative address)		2	2
JMP @A+DPTR	Jump indirect relative to the DPTR.		1	2
JZ rel	Jump relative if ACC is zero.		2	2
JNZ rel	Jump relative if ACC is not zero.		2	2
CJNE A,direct,rel	Compare direct byte to ACC and jump if not equal.		3	2
CJNE A,#data,rel	Compare immediate byte to ACC and jump If not equal		3	2
CJNE Rn,#data,rel	Compare immediate byte to register and jump If not equal		3	2
CJNE @Ri,#data,rel	Compare immediate byte to indirect and jump If not equal		3	2
DJNZ Rn,rel	Decrement register and jump if not zero.		2	2
DJNZ direct,rel	Decrement direct byte and jump if not zero.		3	2

2.20 Other Instructions

Mnemonic	Description	Size	Cycles
NOP	No operation	1	1

2.21 Software:-

Keil compiler or UMPS for programming

Window xp

Chapter 3
LCD Display

3.1 Introduction

Frequently, an 8051 program must interact with the outside world using input and output devices that communicate directly with a human being. One of the most common devices attached to an 8051 is an LCD display. Some of the most common LCDs connected to the 8051 are 16x2 and 20x2 displays. This means 16 characters per line by 2 lines and 20 characters per line by 2 lines, respectively. Fortunately, a very popular standard exists which allows us to communicate with the vast majority of LCDs regardless of their manufacturer. The standard is referred to as HD44780U, which refers to the controller chip which receives data from an external source (in this case, the 8051) and communicates directly with the LCD.

The 44780 standard requires 3 control lines as well as either 4 or 8 I/O lines for the data bus. The user may select whether the LCD is to operate with a 4-bit data bus or an 8-bit data bus. If a 4-bit data bus is used, the LCD will require a total of 7 data lines (3 control lines plus the 4 lines for the data bus). If an 8-bit data bus is used, the LCD will require a total of 11 data lines (3 control lines plus the 8 lines for the data bus).

The three control lines are referred to as EN, RS, and RW.

The EN line is called "Enable." This control line is used to tell the LCD that you are sending it data. To send data to the LCD, your program should first set this line high (1) and then set the other two control lines and/or put data on the data bus. When the other lines are completely ready, bring EN low (0) again. The 1-0 transition tells the 44780 to take the data currently found on the other control lines and on the data bus and to treat it as a command. The RS line is the "Register Select" line. When RS is low (0), the data is to be treated as a command or special instruction (such as clear screen, position cursor,

etc.). When RS is high (1), the data being sent is text data which should be displayed on the screen. For example, to display the letter "T" on the screen you would set RS high.

The RW line is the "Read/Write" control line. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading) the LCD. Only one instruction ("Get LCD status") is a read command. All others are write commands--so RW will almost always be low. Finally, the data bus consists of 4 or 8 lines (depending on the mode of operation selected by the user). In the case of an 8-bit data bus, the lines are referred to as DB0, DB1, DB2, DB3, DB4, DB5, DB6, and DB7.

AN EXAMPLE HARDWARE CONFIGURATION

As we've mentioned, the LCD requires either 8 or 11 I/O lines to communicate with. For the sake of this tutorial, we are going to use an 8-bit data bus--so we'll be using 11 of the 8051's I/O pins to interface with the LCD. Let's draw a sample psuedo-schematic of how the LCD will be connected to the 8051.

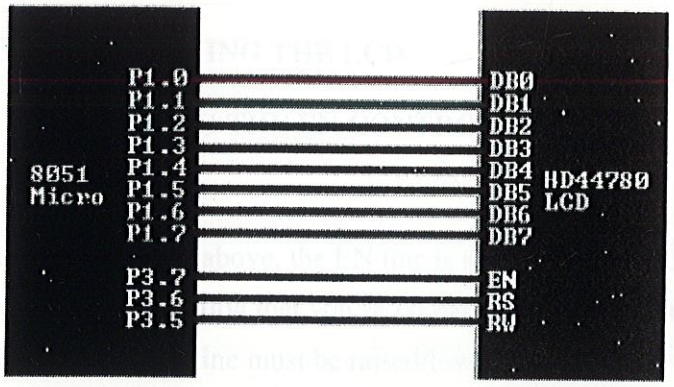

Figure 3.1 .Connection if display with microcontroller

As you can see, we've established a 1-to-1 relation between a pin on the 8051 and a line on the 44780 LCD. Thus as we write our assembly program to access the LCD, we are going to equate constants to the 8051 ports so that we can refer to the lines by their 44780 name as opposed to P0.1, P0.2, etc. Let's go ahead and write our initial equates:

```
DB0 EQU P1.0
DB1 EQU P1.1
DB2 EQU P1.2
DB3 EQU P1.3
DB4 EQU P1.4
DB5 EQU P1.5
DB6 EQU P1.6
DB7 EQU P1.7
EN EQU P3.7
RS EQU P3.6
RW EQU P3.5
DATA EQU P1
```

Having established the above equates, we may now refer to our I/O lines by their 44780 name. For example, to set the RW line high (1), we can execute the following instruction:

```
SETB RW
```

3.2 PROGRAMING THE LCD

3.2.1 HANDLING THE EN CONTROL LINE

As we mentioned above, the EN line is used to tell the LCD that you are ready for it to execute an instruction that you've prepared on the data bus and on the other control lines. Note that the EN line must be raised/lowered before/after each instruction sent to the LCD regardless of whether that instruction is read or write, text or instruction. In short, you must always manipulate EN when communicating with the LCD. EN is the LCD's way of knowing that you are talking to it. If you don't raise/lower EN, the LCD doesn't know you're talking to it on the other lines. Thus, before we interact in any way with the LCD we will always bring the EN line high with the following instruction:

```
SETB EN
```

And once we've finished setting up our instruction with the other control lines and data bus lines, we'll always bring this line back low:

CLR EN

Programming Tip: The LCD interprets and executes our command at the instant the EN line is brought low. If you never bring EN low, your instruction will never be executed. Additionally, when you bring EN low and the LCD executes your instruction, it requires a certain amount of time to execute the command. The time it requires to execute an instruction depends on the instruction and the speed of the crystal which is attached to the 44780's oscillator input.

3.2.2 CHECKING THE BUSY STATUS OF THE LCD

As previously mentioned, it takes a certain amount of time for each instruction to be executed by the LCD. The delay varies depending on the frequency of the crystal attached to the oscillator input of the 44780 as well as the instruction which is being executed. While it is possible to write code that waits for a specific amount of time to allow the LCD to execute instructions, this method of "waiting" is not very flexible. If the crystal frequency is changed, the software will need to be modified. Additionally, if the LCD itself is changed for another LCD which, although 44780 compatible, requires more time to perform its operations, the program will not work until it is properly modified.
A more robust method of programming is to use the "Get LCD Status" command to determine whether the LCD is still busy executing the last instruction received.
The "Get LCD Status" command will return to us two tidbits of information the information that is useful to us right now is found in DB7. In summary, when we issue the "Get LCD Status" command the LCD will immediately raise DB7 if it's still busy executing a command or lower DB7 to indicate that the LCD is no longer occupied. Thus our program can query the LCD until DB7 goes low, indicating the LCD is no longer busy. At that point we are free to continue and send the next command.

Since we will use this code every time we send an instruction to the LCD, it is useful to make it a subroutine. Let's write the code:

```
WAIT_LCD:
SETB EN ;Start LCD command
CLR RS ;It's a command
SETB RW ;It's a read command
MOV DATA,#0FFh ;Set all pins to FF initially
MOV A,DATA ;Read the return value
JB ACC.7,WAIT_LCD ;If bit 7 high, LCD still busy
CLR EN ;Finish the command
CLR RW ;Turn off RW for future commands
RET
```

Thus, our standard practice will be to send an instruction to the LCD and then call our WAIT_LCD routine to wait until the instruction is completely executed by the LCD. This will assure that our program gives the LCD the time it needs to execute instructions and also makes our program compatible with any LCD, regardless of how fast or slow it is.

Programming Tip: The above routine does the job of waiting for the LCD, but was it to be used in a real application a very definite improvement would need to be made: as written, if the LCD never becomes "not busy" the program will effectively "hang," waiting for DB7 to go low. If this never happens, the program will freeze. Of course, this should never happen and won't happen when the hardware is working properly. But in a real application it would be wise to put some kind of time limit on the delay--for example, a maximum of 256 attempts to wait for the busy signal to go low. This would guarantee that even if the LCD hardware fails, the program would not lock up.

3.2.3 INITIALIZING THE LCD

Before you may really use the LCD, you must initialize and configure it. This is accomplished by sending a number of initialization instructions to the LCD. The first instruction we send must tell the LCD whether we'll be communicating with it with an 8-bit or 4-bit data bus. We also select a 5x8 dot character font. These two options are

selected by sending the command 38h to the LCD as a command. As you will recall from the last section, we mentioned that the RS line must be low if we are sending a command to the LCD. Thus, to send this 38h command to the LCD we must execute the following 8051 instructions:

```
SETB EN
CLR RS
MOV DATA,#38h
CLR EN
LCALL WAIT_LCD
```

Programming Tip: The LCD command 38h is really the sum of a number of option bits. The instruction itself is the instruction 20h ("Function set"). However, to this we add the values 10h to indicate an 8-bit data bus plus 08h to indicate that the display is a two-line display.

We've now sent the first byte of the initialization sequence. The second byte of the initialization sequence is the instruction 0Eh. Thus we must repeat the initialization code from above, but now with the instruction. Thus the next code segment is:

```
SETB EN
CLR RS
MOV DATA,#0Eh
CLR EN
LCALL WAIT_LCD
```

Programming Tip: The command 0Eh is really the instruction 08h plus 04h to turn the LCD on. To that an additional 02h is added in order to turn the cursor on.

The last byte we need to send is used to configure additional operational parameters of the LCD. We must send the value 06h.

```
SETB EN
CLR RS
MOV DATA,#06h
```

```
CLR EN
LCALL WAIT_LCD
```

Programming Tip: The command 06h is really the instruction 04h plus 02h to configure the LCD such that every time we send it a character, the cursor position automatically moves to the right. So, in all, our initialization code is as follows:

```
INIT_LCD:

SETB EN
CLR RS
MOV DATA,#38h
CLR EN
LCALL WAIT_LCD
SETB EN
CLR RS
MOV DATA,#0Eh
CLR EN
LCALL WAIT_LCD
SETB EN
CLR RS
MOV DATA,#06h
CLR EN
LCALL WAIT_LCD
RET
```

Having executed this code the LCD will be fully initialized and ready for us to send display data to it.

3.2.4 CLEARING THE DISPLAY

When the LCD is first initialized, the screen should automatically be cleared by the 44780 controller. However, it's always a good idea to do things yourself so that you can

be completely sure that the display is the way you want it. Thus, it's not a bad idea to clear the screen as the very first operation after the LCD has been initialized.

An LCD command exists to accomplish this function. Not surprisingly, it is the command 01h. Since clearing the screen is a function we very likely will wish to call more than once, it's a good idea to make it a subroutine:

```
CLEAR_LCD:

SETB EN
CLR RS
MOV DATA,#01h
CLR EN
LCALL WAIT_LCD
RET
```

How that we've written a "Clear Screen" routine, we may clear the LCD at any time by simply executing an LCALL CLEAR_LCD.

Programming Tip: Executing the "Clear Screen" instruction on the LCD also positions the cursor in the upper left-hand corner as we would expect.

3.2.5 WRITING TEXT TO THE LCD

Now we get to the real meat of what we're trying to do: All this effort is really so we can display text on the LCD. Really, we're pretty much done.

Once again, writing text to the LCD is something we'll almost certainly want to do over and over--so let's make it a subroutine.

```
WRITE_TEXT:

SETB EN
SETB RS
MOV DATA,A
CLR EN
```

```
LCALL WAIT_LCD
RET
```

The WRITE_TEXT routine that we just wrote will send the character in the accumulator to the LCD which will, in turn, display it. Thus to display text on the LCD all we need to do is load the accumulator with the byte to display and make a call to this routine.

- **.A "HELLO WORLD" PROGRAM**

Now that we have all the component subroutines written, writing the classic "Hello World" program--which displays the text "Hello World" on the LCD is a relatively trivial matter. Consider:

```
LCALL INIT_LCD
LCALL CLEAR_LCD
MOV A,#'H'
LCALL WRITE_TEXT
MOV A,#'E'
LCALL WRITE_TEXT
MOV A,#'L'
LCALL WRITE_TEXT
MOV A,#'L'
LCALL WRITE_TEXT
MOV A,#'O'
LCALL WRITE_TEXT
MOV A,#' '
LCALL WRITE_TEXT
MOV A,#'W'
LCALL WRITE_TEXT
MOV A,#'O'
LCALL WRITE_TEXT
MOV A,#'R'
LCALL WRITE_TEXT
MOV A,#'L'
```

```
LCALL WRITE_TEXT
MOV A,#'D'
LCALL WRITE_TEXT
```

The above "Hello World" program should, when executed, initialize the LCD, clear the LCD screen, and display "Hello World" in the upper left-hand corner of the display.

3.2.6 CURSOR POSITIONING

The above "Hello World" program is simplistic in the sense that it prints its text in the upper left-hand corner of the screen. However, what if we wanted to display the word "Hello" in the upper left-hand corner but wanted to display the word "World" on the second line at the tenth character? This sounds simple--and actually, it is simple. However, it requires a little more understanding of the design of the LCD.

The 44780 contains a certain amount of memory which is assigned to the display. All the text we write to the 44780 is stored in this memory, and the 44780 subsequently reads this memory to display the text on the LCD itself. This memory can be represented with the following "memory map":

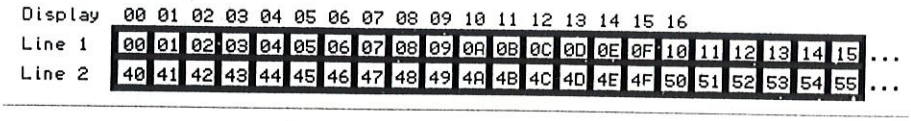

Figure 3.2 .memory map of display

Thus, the first character in the upper left-hand corner is at address 00h. The following character position (character #2 on the first line) is address 01h, etc. This continues until we reach the 16th character of the first line which is at address 0Fh.

However, the first character of line 2, as shown in the memory map, is at address 40h. This means if we write a character to the last position of the first line and then write a second character, the second character will not appear on the second line. That is because the second character will effectively be written to address 10h--but the second line begins at address 40h.

Thus we need to send a command to the LCD that tells it to position the cursor on the second line. The "Set Cursor Position" instruction is 80h. To this we must add the address of the location where we wish to position the cursor. In our example, we said we wanted to display "World" on the second line on the tenth character position. Referring again to the memory map, we see that the tenth character position of the second line is address 4Ah. Thus, before writing the word "World" to the LCD, we must send a "Set Cursor Position" instruction--the value of this command will be 80h (the instruction code to position the cursor) plus the address 4Ah. 80h + 4Ah = C4h. Thus sending the command C4h to the LCD will position the cursor on the second line at the tenth character position:

```
SETB EN
CLR RS
MOV DATA,#0C4h
CLR EN
LCALL WAIT_LCD
```

The above code will position the cursor on line 2, character 10. To display "Hello" in the upper left-hand corner with the word "World" on the second line at character position 10 just requires us to insert the above code into our existing "Hello World" program. This results in the following:

```
LCALL INIT_LCD
LCALL CLEAR_LCD
MOV A,#'H'
LCALL WRITE_TEXT
MOV A,#'E'
LCALL WRITE_TEXT
MOV A,#'L'
LCALL WRITE_TEXT
MOV A,#'L'
LCALL WRITE_TEXT
MOV A,#'O'
LCALL WRITE_TEXT
```

```
SETB EN
CLR RS
MOV DATA,#0C4h
CLR EN
LCALL WAIT_LCD
MOV A,#'W'
LCALL WRITE_TEXT
MOV A,#'O'
LCALL WRITE_TEXT
MOV A,#'R'
LCALL WRITE_TEXT
MOV A,#'L'
LCALL WRITE_TEXT
MOV A,#'D'
LCALL WRITE_TEXT
```

3.3 PIN WISE DETAIL OF LCD

1.	Vss	GROUND
2.	Vcc	+5VOLT SUPPLY
3	Vee	POWER SUPPLY TO CONTROL CONTRAST
4.	RS	RS = 0 TO SELECT COMMAND REGISTER
		RS = 1 TO SELECT DATA REGISTER
5.	R/W	R/W = 0 FOR WRITE
		R/W = 1 FOR READ
6	E	ENABLE
7	DB0	

8 DB1

9. DB2

10. DB3

11. DB4

12. DB5

13. DB6

14. DB7

15, 16 FOR BACK LIGHT DISPLAY

3.4 LCD COMMAND CODES

1. CLEAR DISPLAY SCREEN

2. RETURN HOME

4 DECREMENT CURSOR (SHIFT CURSOR TO LEFT)

5 SHIFT DISPLAY RIGHT.

6. INCREMENT CURSOR (SHIFT CURSOR TO RIGHT)

7. SHIFT DISPLAY LEFT

8. DISPLAY OFF, CURSOR OFF

A DISPLAY OFF CURSOR ON

C DISPLAY ON CURSOR OFF

E DISPLAY ON CURSOR BLINKING

F. DISPLAY ON CURSOR BLINKING.

10. SHIFT CURSOR POSITION TO LEFT

14. SHIFT CURSOR POSITION TO RIGHT

18. SHIFT THE ENTIRE DISPLAY TO THE LEFT

1C SHIFT THE ENTIRE DISPLAY TO THE RIGHT

80 FORCE CURSOR TO BEGINNING OF IST LINE

C0 FORCE CURSOR TO BEGINNING OF 2^{ND} LINE

LINES AND 5 X 7 MATRIX

Chapter-4

Programming Code

4.1 Programming code

The assembly level programming code for our project is mentioned below

```
; Program to --------- Car LIFT----------
        lcd equ p0
        rs equ p2.5
        rw equ p2.6
        en equ p2.7
        LED equ p3
        d7 equ p0.7
org 33h
acall mainlp
mainlp:
        setb p3.7
                        ;show car is parked or not on led
        acall carparking            ;calling LCd commands
        acall express
        acall gotogrd   ;displau car parking
        acall switch
express:                ; moving auto carparking on lcd
        acall fcbfl
        mov dptr,#TABLE1
```

```
        acall write
        ret
switch:
        jnb p3.7,j1 ; checking ir sensor
        jnb p1.7,motoraction6
        ret
start1:                 ; moving motor forward and backward

        jb p1.0,stop1 ;if car is not on ground floor
        jb p1.1,motoraction1
        jb p1.2,motoraction2
        jb p1.3,motoraction3
        jnb p1.4,j2
        acall motoraction4
        j2:
        acall nospace
        acall delay
        ret
nospace:
        acall clrlcd
        acall fcbfl
         mov dptr,#TABLE7
         ACALL write
         acall delay
```

```
acall delay
        ret
stop1:
clr p2.0
setb p2.1
acall delay
acall gotogrd
acall delay
acall delay
acall delay
jnb p1.0,stop
jb p1.0,stop1
ret
stop:
setb p2.0
setb p2.1
acall delay
acall delay
ret
        motoraction1:
                acall fcbsl
                mov dptr,#TABLE3
                acall write
                setb p2.0
```

```
                clr p2.1
                acall delay1
                acall delay1
                acall delay1
                jb p1.5,motoraction1
                acall stop1
                ret
motoraction6:
                acall fcbsl
                mov dptr,#TABLE8
                acall write
                setb p2.0
                clr p2.1
                acall delay1
                acall delay1
                acall delay1
                jb p3.6,motoraction6
                acall stop1
                ret
motoraction2:
                acall fcbsl
                mov dptr,#TABLE4
                acall write
                setb p2.0
```

```
                clr p2.1
                acall delay1
                acall delay1
                acall delay1
                jb p1.5,motoraction2
                acall stop1
                ret
motoraction3:
                acall fcbsl
                mov dptr,#TABLE5
                acall write
                setb p2.0
                clr p2.1
                acall delay1
                acall delay1
                acall delay1
                jb p1.6,motoraction3
                acall delay
                jnb p1.6,stop1
                ret
motoraction4:
                acall fcbsl
                mov dptr,#TABLE6
                acall write
```

```
                setb p2.0
                clr p2.1
                acall delay1
                acall delay1
                acall delay1
                setb p1.5
                 jb p1.6,motoraction1
                acall stop1
                ret
                gotogrd:

        acall fcbsl
        mov dptr,#TABLE2
        acall write
        ret
clrlcd:
 mov A,#01h                ;              Entry mode, Set increment
        acall command
        acall delay
        ret
carparking:                     ; LCD routines
        acall delay
        mov A,#01h              ;              Entry mode, Set increment
        acall command
```

```
        acall delay
        mov A,#06h              ;       Entry mode, Set increment
        acall command
        acall delay
        mov A,#0Eh              ;       Entry mode, Set increment
        acall command
        acall delay
        ret
display:            ;WRITE COMMAND TO LCD
        mov lcd,a
        setb rs
        clr rw
        setb en
        acall delay
        clr en
        ret
command:            ;INSTRUCTION COMMAND TO LCD
        mov lcd,a
        clr rs
        clr rw
        setb en
        acall delay
        clr en
        ret
```

```
write:                  ;WRITE FROM DATATABLES
        clr a
        movc a,@a+dptr
        acall display
        inc dptr
        jz again
      acall delay
sjmp write
again: acall delay1
        ret
fcbfl:
        mov a,#80h          ;CURSOR AT BEGINNING OF IST LINE
        acall command
        acall delay
        ret
fcbsl:
        mov a,#0C0h         ;CURSOR AT BEGINNING OF IIND LINE
     acall delay
        acall command
        ret
delay:                      ;lcd delay
        mov r3,#50
here2: mov r4,#255
here:   djnz r4,here
```

```
djnz r3,here2
        ret
delay1:                         ;motor delay
mov r0,#0ffh
her: mov r1,#0ffh
 h1: djnz r1,h1
djnz r0,her
ret
;DATA TABLE
TABLE1:  DB "CAR LIFT",0
TABLE2:  DB 'GOTOGROUND',0
TABLE3:  DB "GOTO***F1*",0
TABLE4:  DB "GOTO***F2*",0
TABLE5:  DB "GOTO***S1*",0
TABLE6:  DB "GOTO***S2*",0
TABLE7:  DB "NO SPACE",0
TABLE8:  DB "VIP******",0
end
```

Bibliography

1. HAND BOOK OF ELECTRONICS — A.K. MAINI.

2. HAND BOOK OF ELECTRONICS — GUPTA & KUMAR.

3. LET US C — YASHWANT KANITKAR.

4. SHUAM SERIES — TATA MC GRILL

5. DIGITAL SYSTEMS PRINCIPLES AND APPLICATION — RONALD LTOCCI

(Sixth addition)

6. ELECTRONICS FOR YOU

7. DIGITAL DESIGN — MORIS MANO.

(Second addition)

8. MODERN ALL ABOUT MOTHERBOARD — LOTHIA, M.

(Bpb-publishers)

9. POWER SUPPLY FOR ALL OCCASION — SHARMA, MC.

(Bpb-publishers)

10. CMOS DATA BOOK (74SERIES) — ECA.

(Bpb-publishers)

11. PRACTICAL VALUE AND TRANSISTOR DATA — POPE.

(Bpb-publishers)

12. PRACTICAL TRANSFORMER DESIGN HAND BOOK — LABON. E.

13. MODERN IC — MANAHAR LOTIA