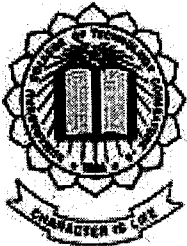


AUTOMATED ACCIDENT IDENTIFICATION SYSTEM



P-903

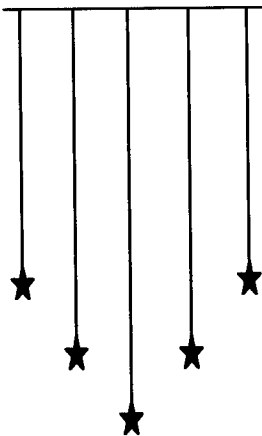
Project Report

Submitted by

**Bhavna. M
Gokila Priya.D
Arun Kumar.S
Naveed.N**

Guided by

**Mr. T. Vijayakumar, M.E.,
Lecturer,
Department of EEE.**



2002 - 2003

In partial fulfillment of the requirement
for the award of Degree of
BACHELOR OF ENGINEERING in
ELECTRICAL AND ELECTRONICS ENGINEERING

Department of Electrical and Electronics Engineering

**Kumaraguru College of Technology
Coimbatore - 641006.**



DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

Kum araguru College of Technology

COIMBATORE-641006.



ISO 9001:2000

CERTIFICATE

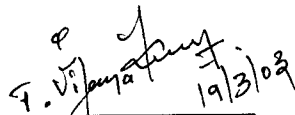
This is to certify that the project

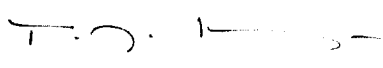
AUTOMATED ACCIDENT IDENTIFICATION SYSTEM

Has been submitted by


- | | |
|-------------------|---------|
| 1. Bhavna.M | 99EEE12 |
| 2. Gokila Priya.D | 99EEE16 |
| 3. Arun Kumar. S | 99EEE07 |
| 4. Naveed.N | 99EEE32 |

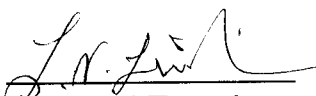
In partial fulfillment of the requirements for the award of degree of Bachelor of Engineering in Electrical and Electronics Engineering branch of the Bharathiar University, Coimbatore – 641006 during the academic year 2002 – 2003.


19/3/03
Guide


Professor and Head

Certified that the candidates mentioned above were examined in project work viva-voce held on 20-03-2003


Internal Examiner


External Examiner

ACKNOWLEDGEMENT

With deep sense of gratitude we express our heartfelt thanks to our guide Mr.T.Vijayakumar, M.E., Lecturer in EEE for his guidance, valuable suggestions and constant interest evinced by him throughout the course of the project work.

We are very much indebted to our Head of the Department Dr. T. M. Kameswaran, B.E., M.Sc., (Engg.), Ph.D., MISTE, Sr.M.I.E.E.E., FIE, for the encouragement and guidance he has given us. We are highly grateful to both our assistant professors, Mr.V.Duraisamy, M.E., MISTE, AMIE, MSSI, M.I.E.E.E. and Mr.K.Rajan, M.E.,MISTE., for their remarkable support.

Our sincere thanks is due to our principal Dr.K.K. Padmanabhan, B.Sc (Engg.), M. Tech., Ph. D., M.I.S.T.E., F.I.E. for having made available all the facilities to do this project.

We have no words to express our profound gratitude to our class advisor Mrs.R.Mahalakshmi, M.E. for her valuable help in doing the project.

We are indebted to the support, encouragement and help rendered by all the faculty members and non-teaching staff of EEE Department.

Last but not the least we thank our dear friends for helping us a lot with their innovative ideas.

SYNOPSIS

SYNOPSIS

The never-ending saga of accidents happening due to vehicles in India has led to the inevitable loss of life and property. The number of vehicles on road is increasing very rapidly and the numbers of accidents are also on the rise. Though, the requirements of safety is more, we can very much prevent the loss of life by knowing where exactly the accident has occurred so that we can get the injured to the hospital for treatment without any time delay. The project that we are presenting is a miniature model of how the location of the vehicles can be determined using infrared sensors.

Our project compromises these situations and provides suitable identification of any vehicle, which has met with an accident, and its location of accident with precision. The time at which the accident has taken place can also be determined. Incase an accident takes place here simulated by tapping the piezoelectric sensor; the signals given to the micro controller are sent to the PC via RS232C which can be monitored and thereby being able to provide the critical help required under such dire circumstances.

**DEDICATED TO OUR BELOVED PARENTS AND FRIENDS
FOR THEIR ENCOURAGEMENT AND SUPPORT.**

CONTENTS

CONTENTS

Chapter	Page no.
CERTIFICATE	
ACKNOWLEDGEMENT	
SYNOPSIS	
CHAPTER 1	1
1.1 INTRODUCTION	2
1.2 NEED FOR THE PROJECT	3
CHAPTER 2	5
2.1 MICRO CONTROLLER BLOCK DIAGRAM	6
2.2 HARDWARE DETAILS	7
2.3 FUNCTIONAL ASPECTS	25
CHAPTER 3	26
3.1 MAIN BLOCK DIAGRAM	27
3.2 OVERALL OPERATION	28
CHAPTER 4	30
4.1 HARDWARE DESCRIPTION	31
4.2 INFRARED CIRCUIT	32
4.3 PIEZOELECTRIC SENSOR	38

4.4 ACCIDENT ANNOUNCEMENT CIRCUIT	40
4.5 POWER SUPPLY BOARD	44
CHAPTER 5	47
5.1 NEED FOR RS232	48
5.2 HARDWARE DETAILS	50
5.3 FUNCTIONAL ASPECTS	53
CHAPTER 6	55
6.1 MICRO CONTROLLER CODING	56
6.2 C PROGRAM	70
CONCLUSION	80
OUTPUT	83
REFERENCES	84
APPENDIX	86

CHAPTER 1

CHAPTER 1

1.1 INTRODUCTION

Our project titled Automated Accident Identification System aims at providing a solution to reduce the number of lives lost due to accidents going unnoticed. The vehicle may be a two-wheeler or four-wheeler with a battery connection. The vehicle is simulated by means of two wheels with infrared sensors fixed on either side to monitor direction of movement of the vehicle. The direction is continuously fed into the micro controller and that will be transmitted to PC through digital modulation techniques. Also a vibration sensor is fixed and depending on the amplitude of vibrations an alarm will be set off. On the PC the location of the vehicle can be tracked based on the data being transmitted from the micro controller.

The micro controller that we are using is AT89C51. It is manufactured by ATMEL INC., USA. This micro controller is an advanced version of INTEL8051 micro controller. The advantage of using this micro controller is that it is cost effective, has low power consumption and a 4K flash memory.

1.2 NEED FOR THIS PROJECT

The rise in number of vehicles on the road nears 12.5% annually and human population is also steadily increasing. Life has become a race against time without even realizing it. All these factors have led to a drastic increase in the number of life threatening accidents thereby worsening the complications when lives are at stake. It was also analyzed that most of the people killed in such fatal accidents died before they received any basic medical treatment the main reason being the problem of identifying where exactly the accident has occurred and the time delay in taking the patient to the hospital.

STATISTICS TABLE

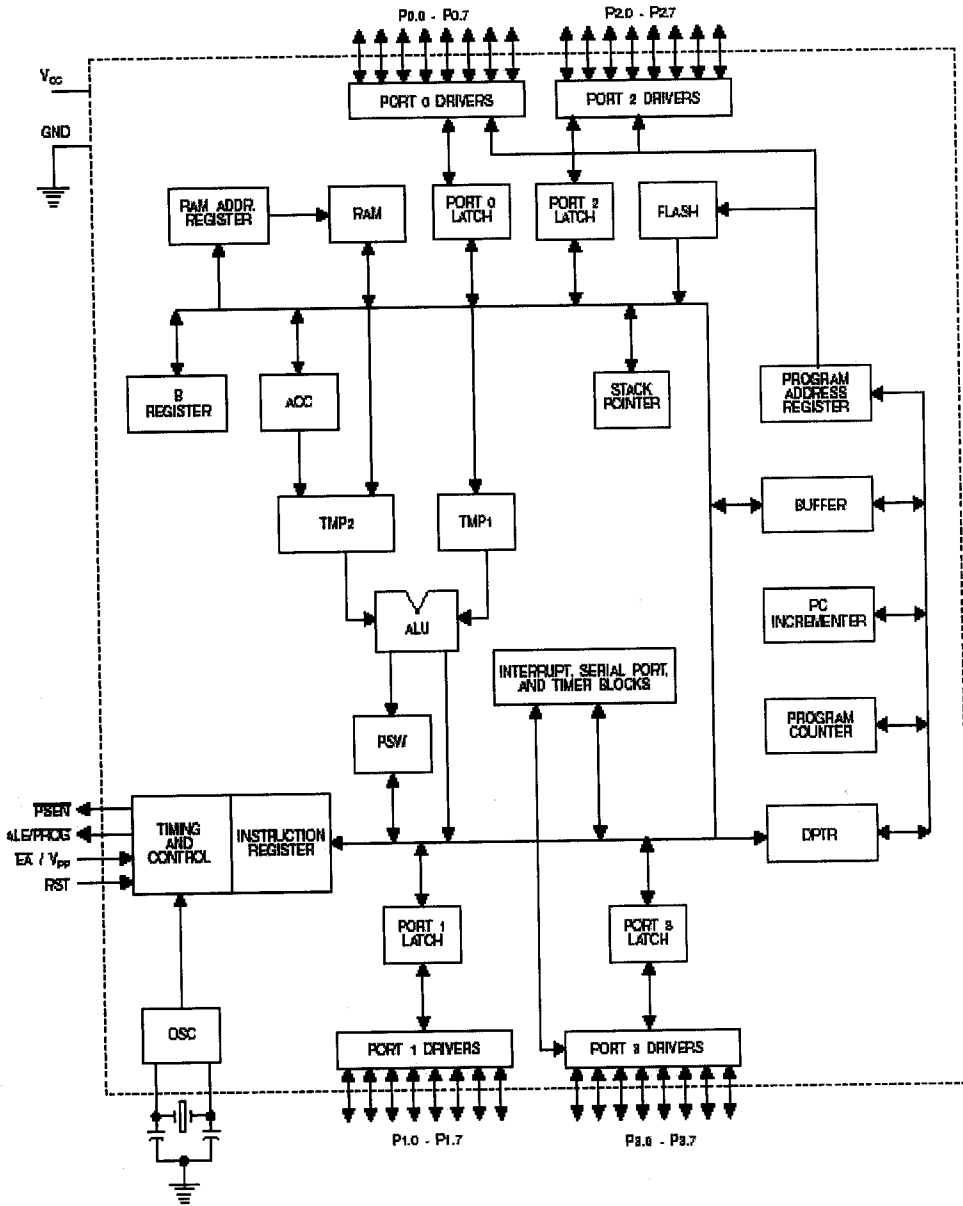
Nature of Accidents	During 1997, Number of accidents and people involved	During 1998 Number of accidents and people involved	During 1999 Number of accidents and people involved	During 2000 Number of accidents and people involved	During 2001 Number of accidents and people involved
Fatal	7947, 8756	8510,9801	8734, 9653	8269, 9300	8579,9571
Grievous	4542, 6557	6562, 8525	5276, 7287	5278, 8496	5442, 8354
Non Injured	8352	789	6845	6239	6994

A survey done by Government of TamilNadu State Transport Authority has been shown above. It has realized some harsh facts. Thereby this project aims at reducing the number of fatalities involved if not the number of accidents. Our project will not only allow the personals to know whether the accident has occurred or not but also the exact location of where the accident has occurred. For this we have done Automated Accident Identification System.

CHAPTER 2

CHAPTER 2

2.1 MICRO CONTROLLER BLOCK DIAGRAM:



2.2 HARDWARE DETAILS

The AT89C51 is a low power, high performance CMOS 8 bit microcomputer with 4k bytes of flash programmable and erasable read only memory (PEROM). The device has been manufactured using Atmel's high density, nonvolatile memory technology and is compatible with the industry standard MCS- 51 instruction set and pin out. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional non-volatile memory programmer. By combining an 8-bit CPU with Flash on a monolithic chip the Atmel AT89C51 is a powerful microcomputer, which provides a highly flexible and cost-effective solution to many embedded control applications.

SALIENT FEATURES:

1. 8- bit CPU optimized for control applications.
2. Extensive Boolean processing capabilities.
3. 64 k program memory address space.
4. 64 k program memory data space.
5. 4 k bytes of on- chip program memory.
6. 128 bytes of on- chip data RAM.
7. 32 bi-directional and individually addressable input/output lines.

8. Two sixteen bit timers/counters.
9. Full duplex UART.
10. 6- source/5- vector interrupt structure with two priority levels.
11. On- chip clock oscillator.

MEMORY ORGANISATION:

The AT89C51 has separate program and data memory. There are two types of memories available. One is read only memory (ROM) type where in you can only read the data's stored in it. Another type is Read Write memory or Random Access Memory (RAM). The program memory can only be read and not written into. The read strobe for external program memory is the PSEN (Program Store Enable). The data memory allows up to 64 k bytes of external RAM to be addressed. The CPU generates RD and WR signals as needed during external data memory access.

ACCESSING EXTERNAL MEMORY:

For fetching from the external program memory 16 I/O lines (port 0 and port 2) are dedicated for bus functions. The address size for accessing the program memory is 16-bit and for accessing the data memory

it can be either 16-bit or 8-bit depending on the instruction being used. The external memory program is accessed under two conditions:

1. When the EA (active low) is active.
2. When the program counter contains a number larger than 0FFFH

Port 0 provides the lower order 8 bits of the address and Port 2 provides the higher order 8 bits.

PORT 0:

It is an 8-bit open drain bi-directional I/O port. As an output port, each pin can sink 8 TTL inputs. If 1's are written to port 0 pin, the pin can be used as high impedance inputs. Port 0 can also be configured to act as a lower order data/address bus during access to external program and data memory. It also receives the code bytes during programming and outputs the code bytes during verification.

PORT 1:

Port 1 is an 8-bit bi-directional I/O port with internal pull-ups. The port 1 output buffer can sink /source four TTL inputs. When 1's are written to port 1 pins they are pulled high by the internal pull-ups and can be used as inputs. As inputs, port 1 pins that are externally being pulled low

will provide source current (I_{il}) because of internal pull-ups. Port 1 also receives the lower order bytes during flash programming and verification.

PORT 2:

It is an 8-bit bi-directional I/O port with internal pull-ups. If 1's are written to port 2 they can be used as inputs. As inputs, port 2 pins that are externally being pulled low will source current (I_{il}) because of the internal pull-ups. Port 2 emits the higher order address byte during fetch cycle from external program memory and during accessing from external data memory that also use 16 bit addresses. Port 2 also receives the higher order address bits and some control signals during flash programming and verification.

PORT 3:

It is an 8-bit bi-directional I/O port with internal pull-ups. The port 3-output buffer can sink four TTL inputs. When 1's are written to port 3 pins they are pulled high by the internal pull-ups and can be used as inputs.

Special functions of Port 3 pins:

P3.0- RXD (Serial input port)

P3.1- TXD (Serial output port)



P3.2- INT0 (Active low pin – External interrupt 0)

P3.3- INT1 (Active low pin – External interrupt 1)

P3.4- T0 (Timer 0 external input)

P3.5- T1 (Timer 1 external input)

P3.6- WR (active low)

P3.7- RD (active low)

RST:

Reset Input. A high on this pin for two machine cycles while the oscillator is running will reset the device.

ALE/PROG:

Address Latch Enable output pulse for latching the lower byte of the address during accesses to external memory. This pin is also the program pulse input during flash programming. In normal operation ALE is emitted at a constant rate of $1/6^{\text{th}}$ the oscillator frequency, and may also be used for external timing or clocking purpose. If desired, ALE operations can be disabled by setting bit 0 of SFR location 8Eh. With the bit set ALE is active only during MOV X instruction.

PSEN:

Program Store Enable is the read strobe to external program memory. While AT89C51 is executing code from external program memory, this pin is activated twice each machine cycle, except that two such activations are skipped during each access to external data memory.

EA/VPP:

External Access Enable. It must be strapped to GND in order to enable the device to fetch code from external program memory locations starting from 0000H to FFFFH. EA should be strapped to Vcc for internal program executions.

The AT89C51 also contains a number of special function registers. They are

ACCUMULATOR:

This a main register all data transfer instructions are carried out using this register. Data can be written and stored into the accumulator. It is a multipurpose register.

B REGISTER:

The B register is used during multiply and divide operations. For other instructions it can be used as a scratch pad register. The B register is used in most of the instructions and has almost equal importance.

PROGRAM STATUS WORD REGISTER:

This register contains program status information. It is an 8-bit register containing the following bits.

CY	AC	F0	RS1	RS0	OV	-	P
(MSB)							(LSB)

CY - Carry Flag

AC - Auxillary Carry

F0 – Available to the user for general purposes

RS1 - Register bank control bits 1 and 0

RS0 - Set/cleared by software to determine working register bank

OV - Overflow Flag

- - User defined flag

P- Parity Flag

STACK POINTER:

The stack pointer register is 8-bit wide. It is incremented before data is stored during PUSH and CALL instructions. The stack can reside at point in the RAM. The stack pointer is initialized to 07H.

DATA POINTER:

The data pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two 8-bit registers.

SERIAL DATA BUFFER:

The serial data buffer actually consists of two separate registers a receive buffer and a transmit buffer register. When data is moved to SBUF, it goes to the transmit buffer, where it is held for serial transmission. When data is moved from the SBUF it is from the receive buffer.

CONTROL REGISTERS:

There are a number of special function registers like TCON, TMOD, SCON, IE, IP etc. These registers contain control and status bits for the interrupt system, timers/counters and serial port.

TIMER REGISTERS:

Register pairs TH0 and TH1 & TL0 and TL1 are the 16-bit counter registers for the timers/counters 0 and 1 respectively.

TIMERS/COUNTERS:

The AT89C51 has two 16 bit timer/counter registers for Timer 0 and Timer1. They can be configured to work either as a timer or as a counter. As a timer the register is incremented every machine cycle. Thus the register counts machine cycles. Each machine cycle consists of 12 oscillator periods; the count rate is 1/12 of the oscillator frequency. As a counter the register increments if there is a 1 to 0 transition in its corresponding external input pin T0 and T1. There are no restrictions on the duty cycle of the external input signal but it should be held for at least one machine cycle to ensure that a given level is sampled at least once before it changes. Timer0 and Timer1 have four operating modes (13-bit timer, 16-bit timer, 8-bit auto-reload and split timer). The four modes are,

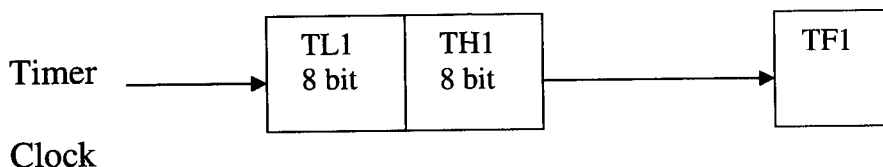
MODE 0:

Both the timers in mode 0 are 8-bit counters with a divide by 32 prescaler. The timer register acts as a 13-bit register. As the count rolls from

all 1's to 0's, it sets the timer interrupt flag TF1. The counted input is enabled to the timer when TR1=1 and GATE=0 or INT1=1. The 13bit registers consists of all 8 bits of TH1 and lower 5 bits of TL1. Setting the run flag TR1 does not clear the registers. Mode 0 operation is the same for Timer1 and Timer0. There are two different gate bits one for Timer1 and the other for Timer0.

MODE 1:

Mode 1 is the same as Mode 0 except that here all the 16 bits are used. The clock is applied to the combined high and low timer registers (TL1 and TH1). As clock pulses are received the timer counts up 0000H, 0001H etc. An overflow occurs on the FFFFH to 0000H flag. The timer continues to count.



MODE 2:

Mode 2 configures the Timer register as an 8-bit counter with automatic reload. Overflow from TL1 not only sets TF1, but also reloads

TL1 with the contents of TH1, which is preset by software. The reload leaves TH1 unchanged. Mode 2 operation is the same for Timer/Counter 0.

TCON REGISTER:

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
-----	-----	-----	-----	-----	-----	-----	-----

TF1- Timer 1 overflow flag

TR1- Timer 1 run control bit

TF0- Timer 0 overflow flag

TR0- Timer 0 run control bit

IE1- Interrupt 1 edge flag

IT1- Interrupt 1 type control bit

IE0- Interrupt 0 edge flag

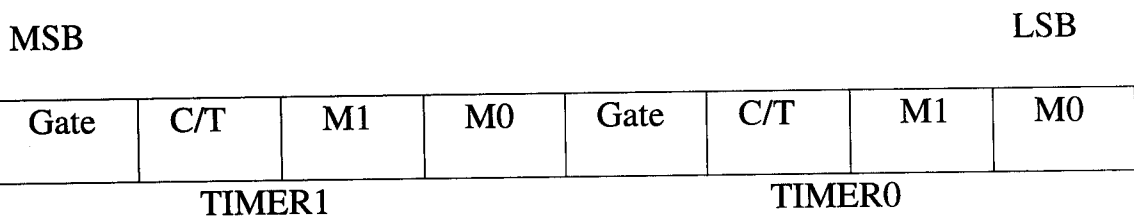
IT0- Interrupt 0 type control bit

MODE 3:

Timer1 in mode 3 simply holds its count. The effect is the same as setting TR1=0. Timer 0 in mode 3 establishes TL0 and TH0 as two separate counters. TL0 uses the Timer0 control bits. TH0 is locked into Timer function (counting machine cycles) over the use of TR1 and TF1 from

Timer1. TH0 now controls the Timer1 interrupt. With Timer0 in mode 3 in AT89C51 can appear to have three Timer/Counters. When Timer0 is in mode3, Timer1 can be turned on and off by switching it out of and into its own mode3. In this case, the serial port as a baud rate generator or in any application not requiring an interrupt can still use Timer1.

TMOD REGISTER:



Gate-gating control when set Timer/Counter X is enabled only while INT X pin is high and TRX control pin is set. When cleared, Timer X is enabled whenever TRX control bit is set.

C/T (T being active low)-Timer or Counter selector cleared for Timer operation (input from internal system clock). Set for Counter operation (input from TX input pin).

M1-mode bit 1

M0-mode bit 0

SERIAL INTERFACE:

The serial port is full duplex which means it can transmit and receive simultaneously. It is also receive-buffer which means it can begin receiving a second byte before a previously received byte is read. Writing to SBUF loads the transmit register and reading the SBUF accesses the physically separate receive register. It operates in four modes,

MODE 0:

Serial data enters and exits through RXD. TXD outputs the shift clock. Eight data bits are transmitted/received, with the LSB first. The baud rate is fixed at 1/12 of the oscillator frequency.

MODE1:

10 bits are transmitted through TXD or received through RXD. A start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive; the stop bit goes RB8 in special function register SCON. The baud rate is variable.

MODE 2:

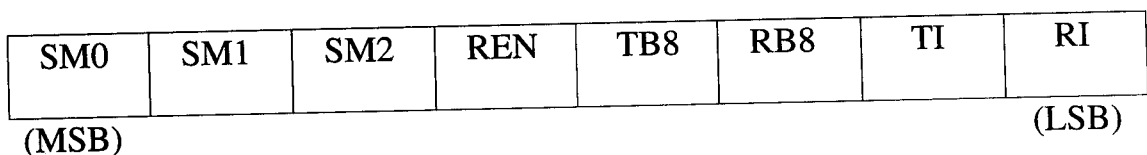
Here 11 bits are transmitted through TXD and received through RXD; a start bit (0), 8 data bits (LSB first) a programmable ninth data bit

(TB8 in SCON) can be assigned the value of 0 or 1. For example the parity bit can be moved into TB8. On receive the ninth data bit goes into RB8 in Special Function Register SCON, while the stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 of the oscillator frequency.

MODE 3:

11 bits are transmitted (through TXD) or received (through RXD); a start bit (0), 8 data bits (LSB first), a programmable ninth data bit and a stop bit (1). In fact, mode 3 is the same as mode 2 in all respects except baud rate, which is variable in mode 3. In all four modes transmission is initiated by an instruction that uses SBUF as a destination register. Reception is initiated in mode 0 by the condition R1=0 and REN=1. Reception is initiated in the other modes by incoming start bit if REN = 1.

SERIAL PORT CONTROL REGISTER:



SM0- Serial port mode bit 0

SM1- Serial port mode bit 1

SM2- Enables the multiprocessor communication in modes 2 and 3. In mode 2 or 3 if SM2 is set to 1 then R1 will not be activated if the received 9th data bit RB8 is 0. In mode 1 if SM2 = 1, then R1 will not be activated if valid stop bit was not received. In mode 0 SM2 should be zero.

REN- Enable serial reception. Set by software to enable reception and cleared by software to disable reception.

TB8- The 9th data bit that will be transmitted in modes 2 and 3. Set or cleared by the software.

RB8- In modes 2 and 3 the 9th data bit that was received. In mode 1 if SM2=0, RB8 is the stop bit that was received in mode 0.

TI- Transmit Interrupt flag. Set by hardware at the end of the 8th bit time in mode 0 or at the beginning of the stop bits in the other modes, in any serial transmission.

RI- Receive Interrupt flag. Set by hardware at the end of the 8th bit time in mode 0 or halfway through the stop bit time in the other modes in any serial reception.

INTERRUPTS:

The AT89C51 provides five interrupt sources, two external interrupts, two timer interrupts and a serial port interrupt. The external interrupts INT0 and INT1 can each be either level activated or transition activated, depending on bits IT0 and IT1 in register TCON. The flag that actually generates these interrupts are the IE0 and IE1 bits in TCON. When the service routine is vectored hardware clears the flag that generated the external interrupt only if the interrupt was transition activated.

The Timer0 and the Timer1 interrupts are generated by TF0 and TF1 which are set by a roll over in their respective Timer or Counter register. When a Timer interrupt is generated, the on-chip hardware clears the flag that generated it when the service routine is vectored.

The logical OR of RI and TI generate the serial port interrupt. Neither of these flags is cleared by hardware when the service routine is vectored. In fact the service routine must normally determine whether RI or TI generated the interrupt, and the bit must be cleared in software.

INTERRUPT ENABLE REGISTER (IE):

(MSB)

(LSB)

EA	—	ET2	ES	ET1	EX1	ET0	EX0
----	---	-----	----	-----	-----	-----	-----

EA - Global Enable/Disable

— -Undefined/ Reserved

ET2-Timer2 interrupt enable bit

ES -Serial port interrupt enable bit

ET1-Timer1 interrupt enable bit

EX1-External interrupt 1 enable bit

ET0-Timer 0 interrupt enable bit

EX0-External interrupt 0 enable bit.

INTERRUPT PRIORITY REGISTER (IP):

(MSB)

(LSB)

—	—	PT2	PS	PT1	PX1	PT0	PX0
---	---	-----	----	-----	-----	-----	-----

___ -Reserved

___ -Reserved

PT2 -Timer 2 interrupt priority bit

PS -Serial port interrupts priority bit

PT1 -Timer 1 interrupt priority bit

PX1 -External interrupt 1 priority bit

PT0 -Timer 0 interrupt priority bit

PX0 - External interrupt 0 priority bit

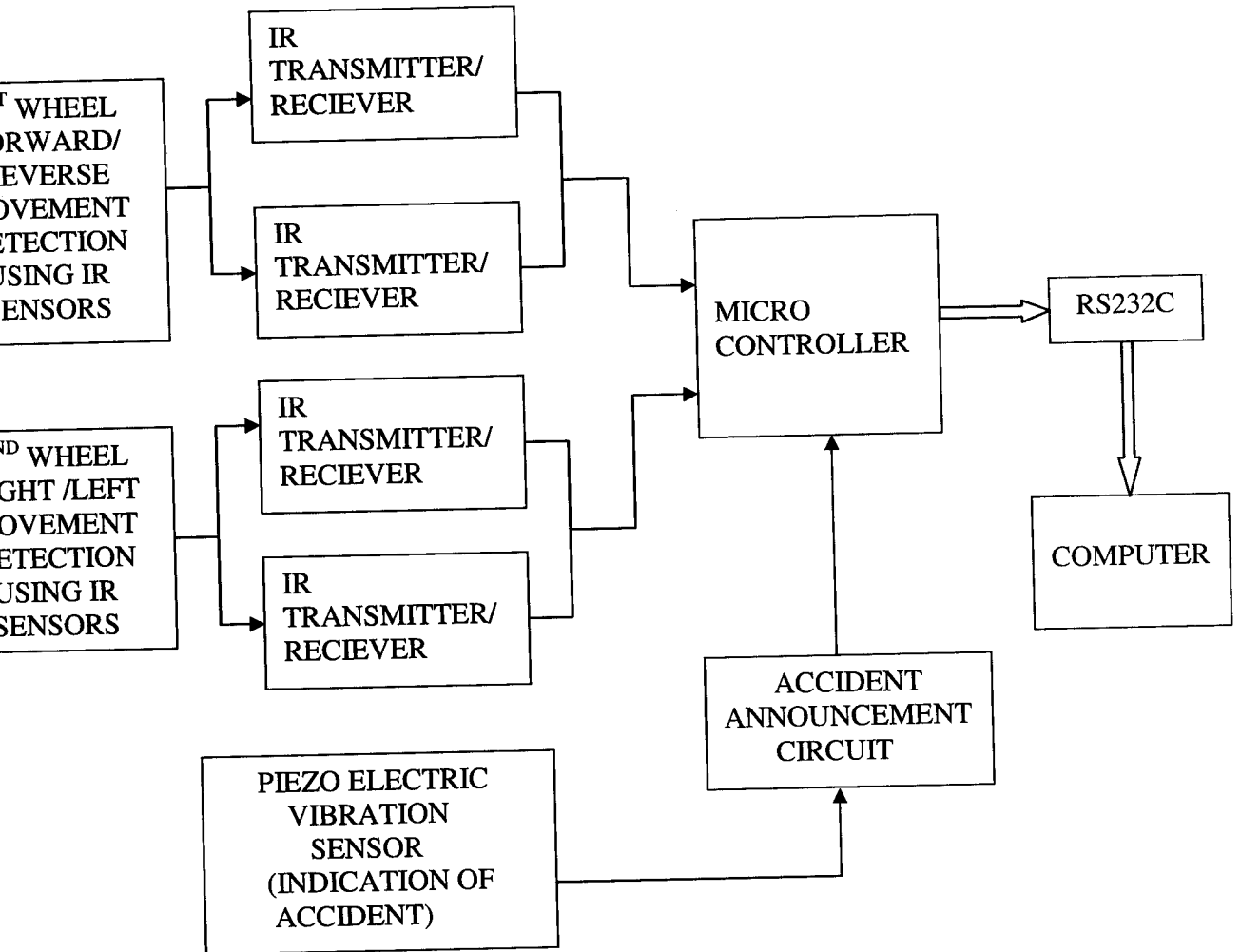
2.3 FUNCTIONAL ASPECTS:

The micro controller receives the various data from the IR sensors and the vibration sensor (piezo electric plate). The data concerning the direction of rotation of the wheels is given to the port pins P1.0 (Right), P1.1 (Left), P1.2 (Reverse) and P1.3 (Forward). The signal from the vibration sensor is given to the interrupt pin P3.2 (INT0). The port pins P3.0 and P3.1 are used for interfacing with RS-232-C. A crystal clock is connected to pins 18 (XTAL2) and 19(XTAL1) of the IC AT89C51.

CHAPTER 3

CHAPTER 3

3.1 MAIN BLOCK DIAGRAM



3.2 OVERALL DESCRIPTION

Our project titled Automated Accident Identification System provides a means of detecting and determining the location of any accident involving vehicles. Since there are practical difficulties in using an actual vehicle to present our idea we have made use of two wheels to simulate the required effect.

Infrared sensors are placed on either side of the wheels for detection of movement. When the wheels are made to rotate due to the hatching made on the wheel surface the following logical combinations of 0, 1 0,1 1, 0 1. This logic is obtained as a result of the hatching, which is in black and white. So when a black comes in between the sensors logic 0 is obtained and a white causes logic 1. Four sensors are made use of here and four directions can be determined forward, reverse, right and left. The signals obtained from the IR sensors are sent to the IR transmitter and receiver circuits which process the signals by comparing it to a particular threshold level and switching it to 5 V thus the relevant data is obtained. Based on these data's the direction of movement of the vehicle can be ascertained. These data's are continuously being sent to the micro controller through the port pins.

Forward logic – 0 0, 1 0, 1 1, 0 1

Reverse logic – 0 0, 0 1, 1 1, 1 0

Right logic - 0 0, 1 0, 1 1, 0 1

Left logic - 0 0, 0 1, 1 1, 1 0

Generally any sudden jerks will cause an abrupt increase in vibrations this concept is made use of here. A piezoelectric plate is made use of here. By tapping the vibration sensor (piezo electric plate) an accident can be simulated. The signal obtained from the piezoelectric plate is sent to the accident announcement circuit where the signal is amplified and compared. If the signal is above a particular level then indication of an accident will result. This signal is sent to the interrupt pin of the micro controller. If even after a particular period of time the vehicle does not move it will indicate that an accident has taken place and an alert will be sent. An RS232 is used for communication between the micro controller and the PC.

CHAPTER 4

CHAPTER 4

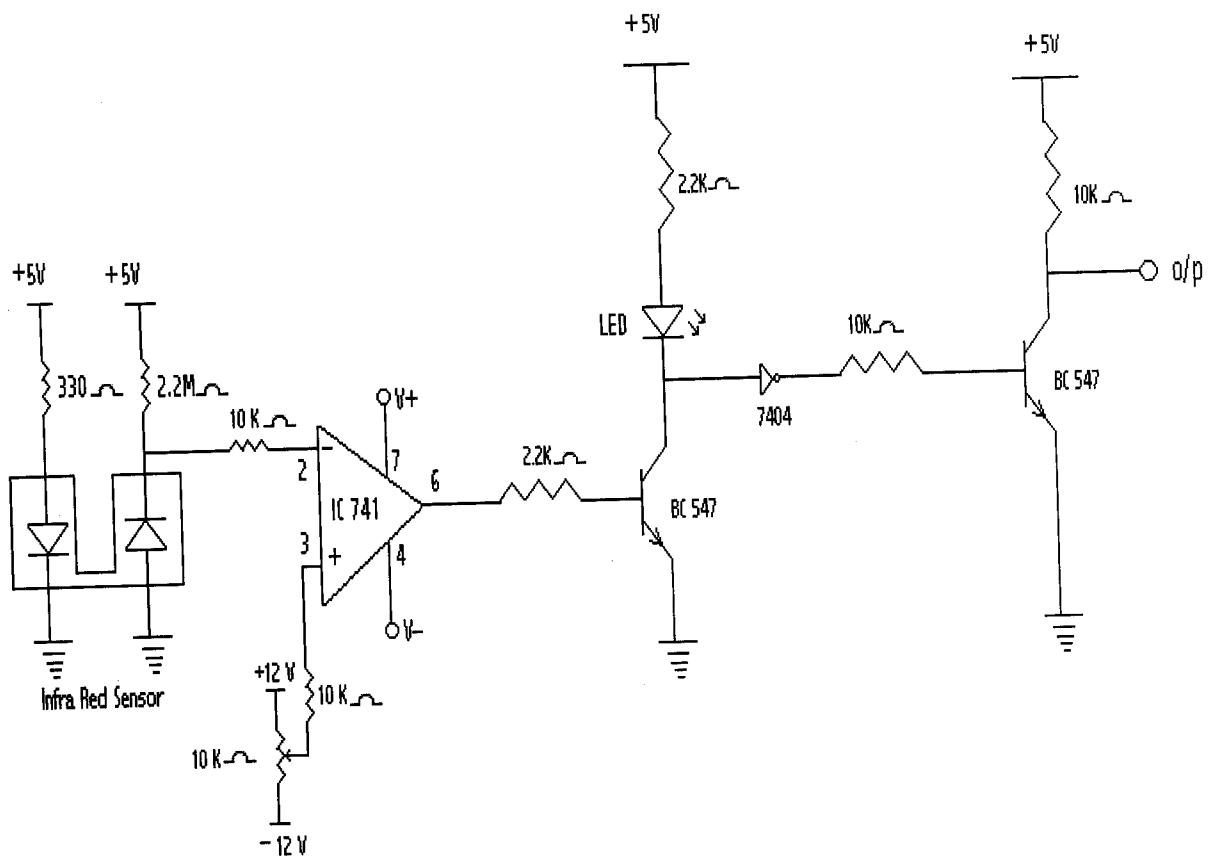
4.1 HARDWARE DESCRIPTION

The hardware basically consists of four important sections, which perform the processes of obtaining the signals and regulating them.

The four sections are the,

1. IR Circuit
2. Piezo Electric Plate
3. Accident Announcement Circuit
4. Power Supply Board

4.2 IR CIRCUIT



INFRARED DIODES:

Optoelectronics is the integration of optical principles and semiconductor electronics. Optoelectronic components are reliable, cost effective sensors. Infrared emitting diodes are solid-state gallium arsenide devices that emit a beam of radiant flux when forward biased. When the junction is forward biased, electron from the N region will recombine with excess holes of the P material in a specially designed recombination region sandwiched between the P&N type materials.

During this recombination process, energy is radiated away from the device in the form of photons. The generated photons will either be reabsorbed by the structure or leave the surface of the device as radiant energy. A few areas of application of such devices include card and paper tape readers, shaft encoder, data transmission systems and intrusion alarms.

OVERALL DESCRIPTION:

At the transmitting side, when the IR emits a beam of radiant flux, a voltage will be developed in one input of the comparator. Let that voltage be V_1 . A reference voltage V_2 is developed across the other input. The comparator compares these two voltages (V_1 & V_2) and the output of

the comparator is positive and negative going pulses. These pulses are given to the switching circuit to get logic 1 and logic 0. The circuit consists of the following

1. Voltage Divider Circuit
2. Comparator
3. Switching Circuit

VOLTAGE DIVIDER:

A potential or voltage divider provides a convenient way of getting a variable voltage from a fixed voltage supply. In general, if two resistors with values R_1 and R_2 are connected in series across a supply voltage V and the voltages developed across each are V_1 and V_2 respectively, then, if I is the current flowing, we can say:

$$V_1 = I \times R_1 \dots\dots(1)$$

$$V_2 = I \times R_2 \dots\dots(2)$$

$$V = V_1 + V_2 = I (R_1 + R_2) \dots\dots(3)$$

Dividing (1) by (3) we obtain:

$$V_1 / V = (I \times R_1) / (I \times (R_1 + R_2))$$

Multiplying both sides by V gives:

$$V_1 = (R_1 * V) / (R_1 + R_2)$$

Similarly from (2) and (3) we get:

$$V_2 = (R_2 * V)/(R1+R2)$$

COMPARATOR:

A comparator is a circuit, which compares a signal voltage applied at one input of an op-amp with a known reference voltage at the other input. It is basically an open loop op-amp with an output $\pm V_{\text{sat}}$ ($= V_{\text{cc}}$). It may be seen that the change in the output state takes place with an increment in input V_1 of only 2 mV. This is the uncertainty region where output cannot be directly defined. There are basically two types of comparators:

1. Inverting comparator
2. Non-Inverting comparator

In case of inverting comparator a fixed reference voltage V_{ref} is applied to (+) input and a time varying signal V_i is applied to (-) input. The output voltage is at $+V_{\text{sat}}$ for $V_i < V_{\text{ref}}$ and V_o goes to $-V_{\text{sat}}$ for $V_i > V_{\text{ref}}$.

In case of non-inverting comparator a fixed reference voltage V_{ref} is applied to (-) input and a time varying signal V_i is applied to (+) input. The output voltage is at $-V_{sat}$ for $V_i < V_{ref}$ and V_o goes to $+V_{sat}$ for $V_i > V_{ref}$.

SWITCHING CIRCUIT:

Many solid-state devices are also used in power control applications, and the simplest of these is the discrete bipolar transistor, which is usually used in the switching mode. In the case of the NPN transistor the switch load is wired between collector and supply positive, and in the case of PNP device it is wired between collector and the zero volt. In both cases the switch-driving signal is applied to base via R_1 , which has a typical resistance about twenty times greater than the load resistance value.

In the NPN circuit Transistor Q1 is cut off (acting like an open switch), with its output at the positive supply voltage value, with zero input signal applied, but can be driven to saturation (so that it acts like a closed switch and passes current from collector to emitter) by applying a large positive input voltage, under which condition the output equals Q1's saturation voltage value (typically 200mV to 600 mV). The action of the PNP circuit is the reverse of that described above, and Q1 is driven to

saturation (with its output a few hundred millivolts below the supply voltage value) and passes current from emitter to collector with zero input drive voltage applied, and is cut off (with its output at zero volts) when the input equals the positive supply rail value.

4.3 PIEZO ELECTRIC SENSOR

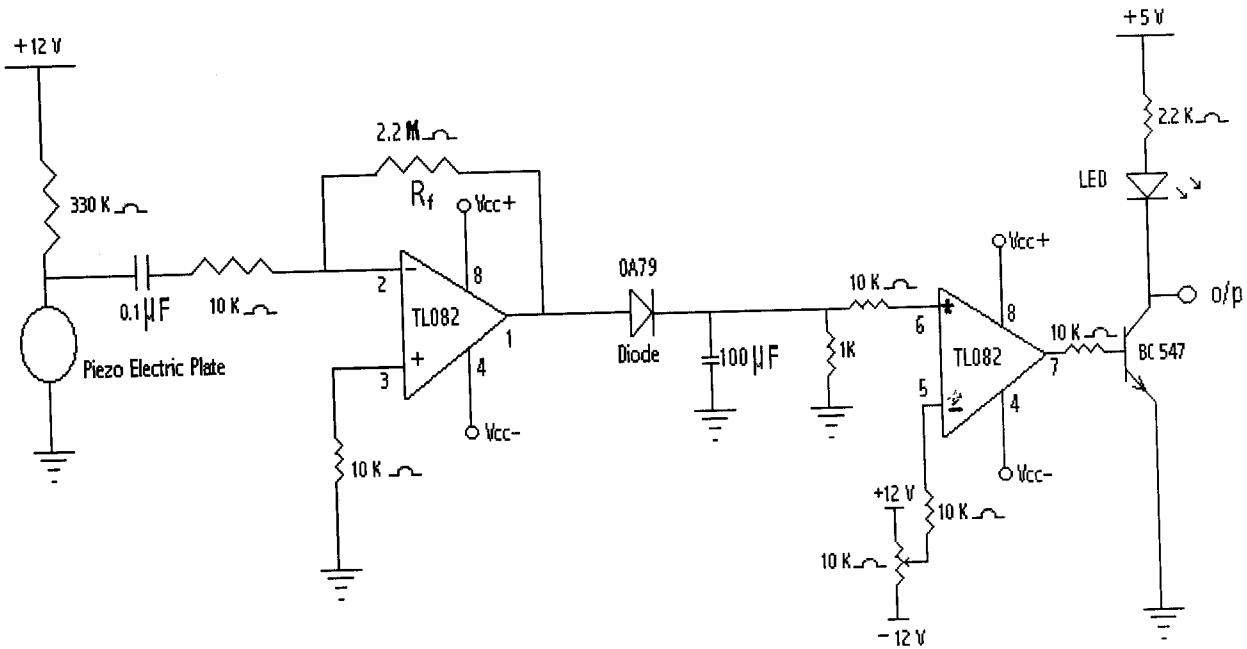
The piezo electric material is one in which electric potential appears across a certain surface of crystal if the dimensions of the crystal are changed by applying a mechanical force. The potential developed is due to the displacement of the charges. This effect is known as piezo electric effect.

The piezoelectric materials include quartz, ceramics A and B, Rochelle salt, Ammonium dihydrogen phosphate, dipotassium tartarate, lithium sulphate. Except for quartz and ceramics A and B the rest are man made crystals grown from aqueous solution under carefully controlled condition. The ceramic materials are polycrystalline in nature basically made of barium titanate. They do not have piezo electric properties in their original states but these properties are produced by special polarizing treatment.

The piezo electric effect can be made to respond to mechanical deformations of the material in many different modes. The modes are thickness expansion, transverse expansion, thickness shear and face shear. A piezo electric element used for converting mechanical motion to electrical signals maybe thought as charge generator and capacitor. Mechanical

deformation generates a charge and this charge appears as voltage across the terminals. Piezo electric effect is also direction sensitive. Tensile force produces a voltage of one polarity while compressive force produces a voltage of another polarity.

4.4 ACCIDENT ANNOUNCEMENT CIRCUIT



This accident announcement circuit is used to determine whether an accident has occurred or not. The accident is simulated by means of tapping the vibration sensor (piezo electric plate). Based on the intensity of vibration a signal will be sent to the micro controller depending upon the output of this circuit. It consists of the following sections.

1. Amplifier
2. Half wave rectifier
3. Comparator
4. Voltage divider
5. Switching circuit

AMPLIFIER:

The piezo electric plate will generate voltage in the range of 5 to 10 mV whenever a mechanical force is applied. This signal is given to the amplifier circuit (non inverting operational amplifier) and gets amplified to about 100 times the original value. The operational amplifier used here is TL082.

HALFWAVE RECTIFIER:

The amplified signal is sent to the rectifier, which consists of a diode in which the positive half of the ac signal is allowed to pass and the remaining is cut off. To remove any further ripple content a filter is used.

COMPARATOR:

A comparator is a circuit, which compares a signal voltage applied at one input of an op-amp with a known reference voltage at the other input. It is basically an open loop op-amp with an output $\pm V_{\text{sat}} (= V_{\text{cc}})$. It may be seen that the change in the output state takes place with an increment in input V_1 of only 2 mV. This is the uncertainty region where output cannot be directly defined. There are basically two types of comparators:

1. Inverting comparator

2. Non - Inverting comparator

The operational amplifier used here also is TL082 this is actually a dual operational amplifier and so it performs the operation of both amplifier and comparator in this accident announcement circuit.

SWITCHING CIRCUIT:

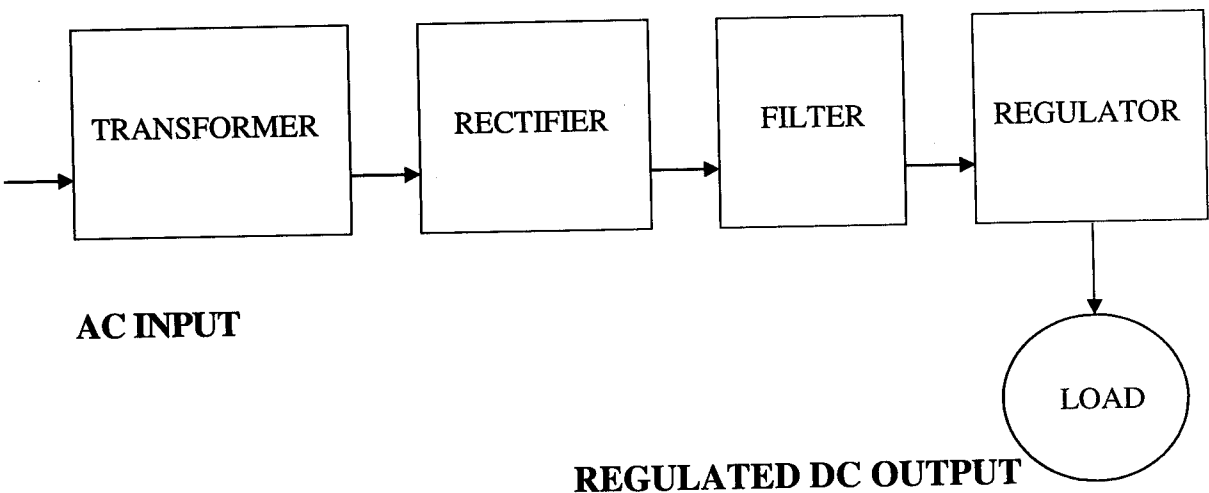
Many solid-state devices are also used in power control applications, and the simplest of these is the discrete bipolar transistor, which is usually used in the switching mode. In the case of the NPN transistor the switch load is wired between collector and supply positive, and in the case of PNP device it is wired between collector and the 0V. In both cases the switch-driving signal is applied to base via R_1 , which has a typical resistance about twenty times greater than the load resistance value.

In the NPN circuit Transistor Q1 is cut off (acting like an open switch), with its output at the positive supply voltage value, with zero input signal applied, but can be driven to saturation (so that it acts like a closed switch and passes current from collector to emitter) by applying a large positive input voltage, under which condition the output equals Q1's saturation voltage value (typically 200mV to 600 mV).

4.5 POWER SUPPLY UNIT

Most electronic circuits work only with low DC voltages so we need a power supply unit that will provide the appropriate voltage. This unit consists of transformer, rectifier, filter and regulator. AC voltage typically 230V rms is given to the primary side of the transformer, which steps the voltage down to the desired level of AC voltage. A diode rectifier is used to obtain a full-wave rectified voltage that is initially filtered by a simple capacitor filter to produce a DC voltage. This resulting DC voltage usually has some ripple or AC voltage variations. A regulator circuit is used to reduce the ripple content and also to maintain the output voltage value constant in spite of variations in the input or at the load.

BLOCK DIAGRAM:



TRANSFORMER:

A transformer is a static (or stationary) device for transforming electrical energy from one alternating circuit to another, without changing the frequency. It works with the principle of mutual induction. It can increase or decrease the voltage with a corresponding increase or decrease in current. A transformer can change low voltage to high voltage and high voltage to low voltage but in both cases the frequency remains unchanged. The transformer has no rotating parts hence it is often called a static transformer. A step down transformer can be used for providing necessary supply to the electronic circuits. In our project we are using a 15-0-15 center tapped transformer.

RECTIFIER:

A rectifier is a device used to convert AC voltage to DC voltage. The DC level obtained from a sinusoidal input can be improved 100% using a process called full-wave rectification. The rectifier uses 4 diodes in a bridge configuration. From the basic bridge configuration we see that during the positive half cycle of the input two diodes (say D2 & D3) are conducting while the other two diodes (D1 & D4) are in "off" state during the period $t = 0$ to $T/2$. Accordingly during the negative half cycle of the input the

conducting diodes are D1 & D4 and D2 and D3 are in the “off state”. Thus the polarity across the load is the same.

FILTER:

The filter circuit used here is the capacitor filter circuit where a capacitor is connected at the rectifier output, and a DC is obtained across it. The filtered waveform is essentially a DC voltage with negligible ripples, which is ultimately fed to the load.

REGULATOR:

The output voltage from the capacitor filter will still have ripple content, which has to be further regulated. The voltage regulator is a device, which maintains the output voltage constant irrespective of the change in supply variations, load variation and temperature changes. Here we use two fixed voltage regulators namely LM 7812, LM 7805 and LM7912. The IC 7812 is a +12V regulator IC 7912 is a -12V regulator and IC 7805 is a +5V regulator.

CHAPTER 5

CHAPTER 5

5.1 NEED FOR RS-232-C:

The RS-232-C was originally set to standardize the interconnections of terminals and host computers through public telephone networks. Modems were used to translate the digital data signals from the computer equipment to analog audio signals suitable for transmission on the telephone network, and back to digital signals at the receiving end.

At that time, each manufacturer of equipment used a different configuration for interfacing a DTE (Data Terminal Equipment) with a DCE (Data Communications Equipment). In 1969, EIA with Bell Laboratories and other parties established a recommended standard for interfacing terminals and data communications equipment. The object of this standard was to simplify the interconnection of equipment manufactured by different firms.

The standard defines electrical, mechanical, and functional characteristics. This standard shortly became RS-232-C (Recommended Standard number 232, revision C from the Electronic Industry Association).

RS-232-C was widely adopted by manufacturers of terminals and computer equipment.

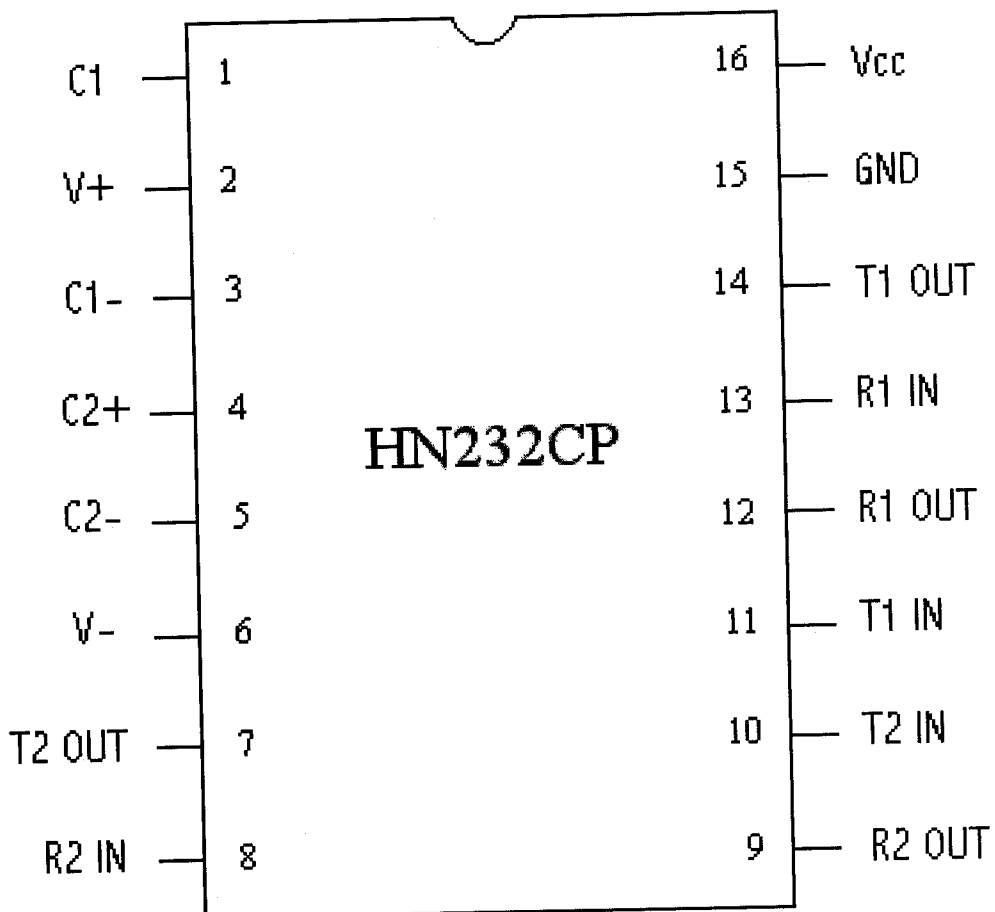
In the 1980' s, therapidly growing microcomputer industry found the RS-232-C standard cheap (compared to parallel connections) and suitable for connecting peripheral equipment to microcomputers. RS-232-C quickly became a standard for connecting microcomputers to printers, plotters, backup tape devices, terminals, programmed equipment and other microcomputers. The RS-232-C is so widely available that it is certain to stay with us for some time to come.

By using RS-232-C digital transmission of data is possible, that is transmission of data in the form of 1's and 0's. These increases the speed with which data is transmitted is highly improved thereby providing better communication conditions.

The voltage levels for all RS-232-C signals are as follows. A logic high, or mark, is a voltage between -3V and -15 V under load (-25 V no load). A logic low or space is a voltage between $+3\text{ V}$ and $+15$ under load ($+25\text{ V}$ no load). Voltages such as $\pm 12\text{ V}$ are commonly used.

5.2 HARDWARE DETAILS:

PIN DIAGRAM



The various pins in the IC H.N232CP are

V_{cc} - Power supply input to the IC 5V +/- 10%

$V+$ - Internally generated positive supply

$V-$ - Internally generated negative supply

GND- Connected to 0V.

$C1+$ - External capacitor (positive terminal) is connected to this lead

$C1-$ - External capacitor (negative terminal) is connected to this lead

$C2+$ - External capacitor (positive terminal) is connected to this lead

$C2-$ - External capacitor (negative terminal) is connected to this lead

T_{1IN} and T_{2IN} - Transmitter inputs

T_{1OUT} and T_{2OUT} - Transmitter outputs

R_{1IN} and R_{2IN} - Receiver inputs

R_{1OUT} and R_{2OUT} - Receiver outputs

The salient features of RS-232-C are

1. Requires only a single 5V supply
2. High data rate of 120kbps
3. Low power consumption
4. Low power shutdown function
5. Multiple Drivers

6. Onboard Voltage/Doubler

7. Multiple Receivers these feature hysteresis to greatly improve noise rejection. They can handle up to +/- 30V.

8. It can be used for any device that requires a communication port like

a. Computer – Mainframe, Laptop, Portable

b. Peripherals – Printers and Terminals

c. Instrumentation

d. Modems.

5.3 FUNCTIONAL ASPECTS:

In our project RS-232-C is used as communication medium between micro controller and PC. A Max level translator is used within the RS-232-C to convert the voltage level (5V) obtained from the micro controller. The UART board in the PC receives the translated voltage (12V) for it's functioning. The pins T_{2IN} and R_{2OUT} are used for interfacing between the micro controller and RS232C. The pins T_{2OUT} and R_{2IN} is used for communication between RS-232-C and the PC.

The basic circuit is divided into three sections: The charge pump, Transmitter and Receiver.

CHARGE PUMP:

The charge pump consists of two sections: the voltage doubler and the voltage inverter sections. Each section is driven by a two-phase, internally generated clock to generate +10V and -10V. The normal clock frequency is 16KHz. Each of the capacitors get charged in sequence thereby obtaining the required output. It accepts input voltages up to 5.5V. The output impedance of voltage divider section (V+) is approximately 200

ohms and the output impedance voltage inverter section (V-) is approximately is 450 ohms.

TRANSMITTER:

They are TTL/CMOS compatible inverters, which translate the inputs to RS232C inputs. The input logic threshold is about 26% of V_{cc} or 1.3 V for $V_{cc} = 5$ V. A logic 1 at the input results in a voltage between $-5V$ and $-V$ at output and a logic 0 results in a voltage between $+5V$ and $+V$. Each transmitter input has an internal 400 Kohms pull up resistor so any unused input can be left unconnected and its output remains in its low state. The transmitters have an internally limited output slew rate, which is less than 30V/microseconds. The outputs are short-circuited and can be shorted to ground definitely.

RECIEVERS:

The receiver inputs accept up to $\pm 30V$ while presenting the required 3 Kohms to 7 Kohms input impedance even if the power is off. The output is 0V to V_{cc} . The output will be low whenever the input is greater than 2.4V and high whenever the input is floating or driven between $+0.8V$ and $-30V$. The receivers feature 0.5V hysteresis to improve noise rejection.

CHAPTER 6

CHAPTER 6

6.1 MICRO CONTROLLER CODING

P0 DATA 080H
P1 DATA 090H
P2 DATA 0A0H
P3 DATA 0B0H
T0 BIT 0B0H.4
AC BIT 0D0H.6
T1 BIT 0B0H.5
EA BIT 0A8H.7
IE DATA 0A8H
RD BIT 0B0H.7
ES BIT 0A8H.4
IP DATA 0B8H
RI BIT 098H.0
INT0 BIT 0B0H.2
CY BIT 0D0H.7
TI BIT 098H.1
INT1 BIT 0B0H.3
PS BIT 0B8H.4
SP DATA 081H
OV BIT 0D0H.2
WR BIT 0B0H.6
SBUF DATA 099H
PCON DATA 087H
SCON DATA 098H
TMODDATA 089H
TCON DATA 088H
IE0 BIT 088H.1
IE1 BIT 088H.3
B DATA 0F0H
ACC DATA 0E0H
ET0 BIT 0A8H.1
ET1 BIT 0A8H.3
TF0 BIT 088H.5
TF1 BIT 088H.7
RB8 BIT 098H.2
TH0 DATA 08CH
EX0 BIT 0A8H.0
IT0 BIT 088H.0
TH1 DATA 08DH
TB8 BIT 098H.3

EX1 BIT 0A8H.2
 IT1 BIT 088H.2
 P BIT 0D0H.0
 SM0 BIT 098H.7
 TL0 DATA 08AH
 SM1 BIT 098H.6
 TL1 DATA 08BH
 SM2 BIT 098H.5
 PT0 BIT 0B8H.1
 PT1 BIT 0B8H.3
 RS0 BIT 0D0H.3
 TR0 BIT 088H.4
 RS1 BIT 0D0H.4
 TR1 BIT 088H.6
 PX0 BIT 0B8H.0
 PX1 BIT 0B8H.2
 DPH DATA 083H
 DPL DATA 082H
 REN BIT 098H.4
 RXD BIT 0B0H.0
 TXD BIT 0B0H.1
 F0 BIT 0D0H.5
 PSW DATA 0D0H

?PR?main?TRACKING SEGMENT CODE

?PR?wheel2?TRACKING SEGMENT CODE

?PR?int0?TRACKING SEGMENT CODE

?PR?ser_init?TRACKING SEGMENT CODE

?PR?del?TRACKING SEGMENT CODE

?PR?delay?TRACKING SEGMENT CODE

?C_INITSEG SEGMENT CODE

?DT?TRACKING SEGMENT DATA
 EXTRN CODE (?C_STARTUP)
 PUBLIC s
 PUBLIC p
 PUBLIC k
 PUBLIC j
 PUBLIC i
 PUBLIC arr

```
PUBLIC tx4
PUBLIC tx3
PUBLIC tx2
PUBLIC rx4
PUBLIC tx1
PUBLIC rx3
PUBLIC rx2
PUBLIC rx1
PUBLIC delay
PUBLIC del
PUBLIC ser_init
PUBLIC int0
PUBLIC wheel2
PUBLIC main
```

RSEG ?DT?TRACKING

```
rx1: DS 1
rx2: DS 1
rx3: DS 1
tx1: DS 1
rx4: DS 1
tx2: DS 1
tx3: DS 1
tx4: DS 1
arr: DS 4
i: DS 2
j: DS 2
k: DS 2
p: DS 1
s: DS 2
```

RSEG ?C_INITSEG

```
DB 002H
DB k
DW 00000H
```

```
DB 001H
DB rx1
DB 000H
```

```
DB 001H
DB p
DB 000H
```

```
;  
; #pragma src;  
; #include<reg51.h>
```

```

; void wheel2();
; void ser_init();
; unsigned int i,j,k=0;
; void ser_init();
; void ser_out();
; void delay();
; void del();
; unsigned char tx1,tx2,tx3,tx4,rx1=0,rx2,rx3,rx4,arr[4],p=0;
; unsigned int s,j;
; main()

```

RSEG ?PR?main?TRACKING

```

main:
    USING      0
                ; SOURCE LINE # 12
; {
                ; SOURCE LINE # 13
; P1=0xff;
                ; SOURCE LINE # 14
    MOV P1,#0FFH
; EA=1;
                ; SOURCE LINE # 15
    SETB EA
; EX0=1;
                ; SOURCE LINE # 16
    SETB EX0
; ser_init();
                ; SOURCE LINE # 17
    LCALL ser_init
; while(1)
                ; SOURCE LINE # 18
; {
                ; SOURCE LINE # 19
; l4:
                ; SOURCE LINE # 20
?main?l4:
; wheel2();
                ; SOURCE LINE # 21
    LCALL wheel2
; tx1=P1&0x0c;
                ; SOURCE LINE # 22
    MOV A,P1
    ANL A,#0CH
    MOV tx1,A
; if(tx1!=0)
                ; SOURCE LINE # 23

```

```

    JNZ    ?main?l4
; {goto l4;}
; SOURCE LINE # 24

;l1:
; SOURCE LINE # 25

?main?l1:
; wheel2();
; SOURCE LINE # 26
    LCALL   wheel2
; tx2=P1&0x0c;
; SOURCE LINE # 27
    MOV    A,P1
    ANL   A,#0CH
    MOV   tx2,A
; if(tx2==tx1)
; SOURCE LINE # 28
    XRL   A,tx1
    JZ    ?main?l1
; {goto l1;}
; SOURCE LINE # 29

;l2:
; SOURCE LINE # 30

?main?l2:
; wheel2();
; SOURCE LINE # 31
    LCALL   wheel2
; tx3=P1&0x0c;
; SOURCE LINE # 32
    MOV    A,P1
    ANL   A,#0CH
    MOV   tx3,A
; if(tx3==tx2)
; SOURCE LINE # 33
    XRL   A,tx2
    JZ    ?main?l2
; {goto l2;}
; SOURCE LINE # 34

;l3:
; SOURCE LINE # 35

?main?l3:
; wheel2();
; SOURCE LINE # 36
    LCALL   wheel2
; tx4=P1&0x0c;
; SOURCE LINE # 37
    MOV    A,P1

```

```

    ANL A,#0CH
    MOV tx4,A
; if(tx4==tx3)
    ; SOURCE LINE # 38
    XRL A,tx3
    JZ ?main?13
; {goto l3;}
    ; SOURCE LINE # 39
?C0010:
;
; if((tx2==0x04)&&(tx3==0x0c)&&(tx4==0x08))
    ; SOURCE LINE # 41
    MOV A,tx2
    CJNE A,#04H,?C0011
    MOV A,tx3
    CJNE A,#0CH,?C0011
    MOV A,tx4
    CJNE A,#08H,?C0011
; {
    ; SOURCE LINE # 42
; SBUF=' F' ;
    ; SOURCE LINE # 43
    MOV SBUF,#046H
; delay();
    ; SOURCE LINE # 44
    LCALL delay
; SCON=0x40;
    ; SOURCE LINE # 45
; del();
    ; SOURCE LINE # 46
    LCALL L?0036
; }
    ; SOURCE LINE # 47
?C0011:
; if((tx2==0x08)&&(tx3==0x0c)&&(tx4==0x04))
    ; SOURCE LINE # 48
    MOV A,tx2
    CJNE A,#08H,?main?14
    MOV A,tx3
    XRL A,#0CH
    JNZ ?main?14
    MOV A,tx4
    XRL A,#04H
    JNZ ?main?14
; {
    ; SOURCE LINE # 49

```

```

; SBUF=' R' ;
; SOURCE LINE # 50
MOV SBUF,#052H
; delay();
; SOURCE LINE # 51
LCALL delay
; SCON=0x40;
; SOURCE LINE # 52
; del();
; SOURCE LINE # 53
LCALL L?0036
; }
; SOURCE LINE # 54
; }
; SOURCE LINE # 55
SJMP ?main?14
; END OF main

; }
;
; void wheel2()

RSEG ?PR?wheel2?TRACKING
wheel2:
USING 0
; SOURCE LINE # 58
; {
; SOURCE LINE # 59
; if(k==0)
; SOURCE LINE # 60
MOV A,k+01H
ORL A,k
JNZ ?C0014
; {
; SOURCE LINE # 61
; rx1=P1&0x03;
; SOURCE LINE # 62
MOV A,P1
ANL A,#03H
MOV rx1,A
; if(rx1==0x00)
; SOURCE LINE # 63
JNZ ?C0022
; {
; SOURCE LINE # 64
; k=1;

```



```

; SOURCE LINE # 65
MOV k,A
MOV k+01H,#01H
; goto retr;
; SOURCE LINE # 66
RET
; }
; SOURCE LINE # 67
; goto retr;
; SOURCE LINE # 68
; }
; SOURCE LINE # 69
?C0014:
; if(k==1)
; SOURCE LINE # 70
MOV A,k+01H
XRL A,#01H
ORL A,k
JNZ ?C0022
; {
; SOURCE LINE # 71
; rx2=P1&0x03;
; SOURCE LINE # 72
MOV A,P1
ANL A,#03H
MOV rx2,A
; if(rx2!=rx1)
; SOURCE LINE # 73
XRL A,rx1
JZ ?C0022
; {
; SOURCE LINE # 74
; arr[p]=rx2;
; SOURCE LINE # 75
MOV A,#LOW(arr)
ADD A,p
MOV R0,A
MOV @R0,rx2
; rx1=rx2;
; SOURCE LINE # 76
MOV rx1,rx2
; p++;
; SOURCE LINE # 77
INC p
; if(p==3)
; SOURCE LINE # 78

```



```

;   SCON=0x40;
      MOV SCON,#040H      ; SOURCE LINE # 91
;   del();
      LCALL del           ; SOURCE LINE # 92
;   }
      ; SOURCE LINE # 93
?C0021:
;   p=0;
      CLR A
      MOV p,A
;   k=0;
      MOV k,A
      MOV k+01H,A
;   }
      ; SOURCE LINE # 96
;   }
      ; SOURCE LINE # 97
;   goto retr;
      ; SOURCE LINE # 98
;   }
      ; SOURCE LINE # 99
;   retr:
      ; SOURCE LINE # 100
?wheel2?retr:
;;
;   }
      ; SOURCE LINE # 102
?C0022:
      RET
;   END OF wheel2

CSEG AT 00003H
      LJMP int0

;
;   void int0(void) interrupt 0

      RSEG ?PR?int0?TRACKING
      USING 0

int0:
      PUSH ACC
      PUSH B

```

```

PUSH DPH
PUSH DPL
PUSH PSW
MOV PSW,#00H
PUSH AR0
PUSH AR1
PUSH AR2
PUSH AR3
PUSH AR4
PUSH AR5
PUSH AR6
PUSH AR7
USING      0
           ; SOURCE LINE # 104
; {
; ser_init();
           ; SOURCE LINE # 106
      LCALL ser_init
; SBUF=' X' ;
           ; SOURCE LINE # 107
      MOV SBUF,#058H
; delay();
           ; SOURCE LINE # 108
      LCALL delay
; SCON=0x40;
           ; SOURCE LINE # 109
      MOV SCON,#040H
; del();
           ; SOURCE LINE # 110
      LCALL del
; }
           ; SOURCE LINE # 111
      POP AR7
      POP AR6
      POP AR5
      POP AR4
      POP AR3
      POP AR2
      POP AR1
      POP AR0
      POP PSW
      POP DPL
      POP DPH
      POP B
      POP ACC
      RETI

```

```
; END OF int0
```

```
;  
;
```

```
; void ser_init()
```

```
    RSEG ?PR?ser_init?TRACKING
```

```
ser_init:
```

```
    USING      0  
                ; SOURCE LINE # 114
```

```
; {  
                ; SOURCE LINE # 115
```

```
; TH1=0x72;  
                ; SOURCE LINE # 116
```

```
    MOV TH1,#072H
```

```
; TMOD=0x20;  
                ; SOURCE LINE # 117
```

```
    MOV TMOD,#020H
```

```
; TR1=1;  
                ; SOURCE LINE # 118
```

```
    SETB TR1
```

```
; delay();  
                ; SOURCE LINE # 119
```

```
    LCALL      delay
```

```
; SCON=0x40;  
                ; SOURCE LINE # 120
```

```
    MOV SCON,#040H
```

```
; }  
                ; SOURCE LINE # 121
```

```
    RET
```

```
; END OF ser_init
```

```
;
```

```
; void del()
```

```
    RSEG ?PR?del?TRACKING
```

```
L?0036:
```

```
    USING      0  
    MOV SCON,#040H
```

```
del:  
                ; SOURCE LINE # 123
```

```
; {  
                ; SOURCE LINE # 124
```

```
; for(s=0;s<=8000;s++)  
                ; SOURCE LINE # 125
```

```
    CLR  A
```

```

MOV s,A
MOV s+01H,A
?C0025:
; {}
; SOURCE LINE # 126
INC s+01H
MOV A,s+01H
JNZ ?C0033
INC s
?C0033:
CJNE A,#041H,?C0025
MOV A,s
CJNE A,#01FH,?C0025
; }
; SOURCE LINE # 127
?C0028:
RET
; END OF del
;
; void delay()
RSEG ?PR?delay?TRACKING
delay:
; SOURCE LINE # 129
; {
; SOURCE LINE # 130
; for(j=0;j<=120;j++)
; SOURCE LINE # 131
CLR A
MOV j,A
MOV j+01H,A
?C0029:
; {}
; SOURCE LINE # 132
INC j+01H
MOV A,j+01H
JNZ ?C0035
INC j
?C0035:
XRL A,#079H
ORL A,j
JNZ ?C0029
; }
; SOURCE LINE # 133
?C0032:

```

RET
; END OF delay

END

6.2 C PROGRAM

```
#include <stdlib.h>
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<process.h>
#include<ctype.h>
#include <graphics.h>
void screen(void);
void map(void);
void main()
{
    struct time t;
    int a,ah,al,result,in=1,mode=2,flag=0,t1,t2;
    int gdriver = DETECT, gmode;
    void *one;
    unsigned int size1;
    int key,x1=50,x2,y2,y1=50, maxx, maxy,style=0,userpat;
    initgraph(&gdriver, &gmode, "\\TC\\BGI");
    maxx = getmaxx() + 1;
    maxy = getmaxy() + 1;
    x2=x1=276;//maxx/2;
    y2=y1=298;//maxy/2;
```



```
setcolor(12);  
circle(40,40,3);  
setfillstyle(1,5);  
floodfill(41,41,12);  
size1=imagesize(36,36,44,44);  
one=malloc(size1);  
getimage(36,36,44,44,one);  
cleardevice();  
userpat = 100;  
setlinestyle(style, userpat,3);  
rectangle(10,10,maxx-10,maxy-10);  
screen();  
setlinestyle(style, userpat,1);  
map();  
while(!kbhit())  
{  
    setcolor(15);  
    _AH=0x03;  
    _DX=0x00;  
    geninterrupt(0x14);  
    ah=_AH;  
    al=_AL;  
    gettime(&t);
```

```
t1=t.ti_sec;
if(flag==0)
{
t2=t1;
flag=1;
}
if(abs(t1-t2)>20 && t1<50 && t2<50 && a=4)
if( (ah & 0x01) == 0x01)
{
gettime(&t);
t2=t.ti_sec;
result = inportb(0x3f8);
a=result;
switch (a)
{
case 65:
key=54;
break;
case 66:
key=52;
break;
case 70:
key=50;
```

```
break;

case 82:

key=56;

break;

case 88:

key=0;

break;

//default:

//printf("%d",a);

}

switch (key)

{

case 0:

gotoxy(35,12);

printf("Problem");

break;

case 56:

if(y1>=1)

y1=y1-in;

//putpixel(x1,y1,15);

//putimage(x1,y1,one,mode);

line(x2,y2,x1,y1);

break;
```

case 54:

```
if(x1<maxx)
```

```
x1=x1+in;
```

```
//putpixel(x1,y1,15);
```

```
//putimage(x1,y1,one,mode);
```

```
line(x2,y2,x1,y1);
```

```
break;
```

case 50:

```
if(y1<maxy)
```

```
y1=y1+in;
```

```
//putpixel(x1,y1,15);
```

```
//putimage(x1,y1,one,mode);
```

```
line(x2,y2,x1,y1);
```

```
break;
```

case 52:

```
if(x1>=1)
```

```
x1=x1-in;
```

```
//putpixel(x1,y1,15);
```

```
//putimage(x1,y1,one,mode);
```

```
line(x2,y2,x1,y1);
```

```
break;
```

case 55:

```
if(x1>=1)
```

```
x1=x1-in;
if(y1>=1)
y1=y1-in;
//putpixel(x1,y1,15);
//putimage(x1,y1,one,mode);
line(x2,y2,x1,y1);
break;
case 57:
if(x1<maxx)
x1=x1+in;
if(y1>=1)
y1=y1-in;
//putpixel(x1,y1,15);
//putimage(x1,y1,one,mode);
line(x2,y2,x1,y1);
break;
case 49:
if(x1>=1)
x1=x1-in;
if(y1<maxy)
y1=y1+in;
//putpixel(x1,y1,15);
//putimage(x1,y1,one,mode);
```

```
line(x2,y2,x1,y1);  
break;  
case 51:  
if(x1<maxx)  
x1=x1+in;  
if(y1<maxy)  
y1=y1+in;  
//putimage(x1,y1,one,mode);  
//putpixel(x1,y1,15);  
line(x2,y2,x1,y1);  
break;  
default:  
break;  
}  
x2=x1;  
y2=y1;  
}  
}  
getch();  
}
```

```
void screen(void)
```

```
{  
    //setfillstyle(1,1);  
    //floodfill(100,100,12);  
    setcolor(13);  
    settextstyle(1,0,2);  
    outtextxy(546,50,"N");  
    setcolor(11);  
    settextstyle(1,0,3);  
    outtextxy(170,50,"VEHICLE TRACKING SYSTEM");  
    setcolor(13);  
    outtextxy(172,48,"VEHICLE TRACKING SYSTEM");  
    setcolor(3);  
    line(520,100,580,100);  
    line(550,70,550,170);  
}
```

```
void map(void)
```

```
{  
    int r=5;  
    setcolor(2);  
    circle(20,450,r);  
    setfillstyle(1,10);
```

```
floodfill(21,453,2);  
setcolor(14);  
settextstyle(1,0,1);  
outtextxy(21,448,"CRISR");  
setcolor(2);  
circle(80,350,r);  
setfillstyle(1,10);  
floodfill(81,353,2);  
setcolor(14);  
outtextxy(81,348,"Erode");  
setcolor(2);  
circle(280,300,r);  
setfillstyle(1,10);  
floodfill(281,303,2);  
setcolor(14);  
outtextxy(281,308,"Namakkal");  
setcolor(2);  
circle(230,130,r);  
setfillstyle(1,10);  
floodfill(232,133,2);  
setcolor(14);  
outtextxy(231,138,"Dharmapuri");  
setcolor(2);
```



```
circle(280,230,r);
setfillstyle(1,10);
floodfill(281,233,2);
setcolor(14);
outtextxy(281,238,"Salem");
setcolor(2);
circle(420,300,r);
setfillstyle(1,10);
floodfill(421,303,2);
setcolor(14);
outtextxy(421,308,"AMSEC");
setcolor(2);
circle(550,450,r);
setfillstyle(1,10);
floodfill(551,453,2);
setcolor(14);
outtextxy(551,448,"Trichy");
setcolor(2);
circle(530,250,r);
setfillstyle(1,10);
floodfill(531,253,2);
setcolor(14);
outtextxy(531,248,"Thuraiyur"); }
```

CONCLUSION

CONCLUSION

The concepts and the software and hardware logic used in this project have been successfully implemented. All the circuits have been tested and the results were satisfactory. Thus by using an embedded system all the initial difficulties faced by the authorities like

- Determining whether an accident has taken place.
- Determining the location of the accident.
- Determining the time of the accident
- Not being able to give the required emergency medical help, which is a must in most accident cases.

can be overcome.

SCOPE FOR DEVELOPMENTS:

The project that we have implemented is a wired version a further development on that would be to make the communication between the vibration sensor and the micro controller wireless thereby giving a better perspective. Implementing GPS for determining the location of the vehicle could be a future development for our method of automation. We have used

only one vibration sensor in our project but in real time applications more than one sensor will have to be placed at strategic and vulnerable locations of the vehicle and this depends on the make and design of the vehicle. To conclude the implementation of this concept will be very beneficial to mankind and will be revolutionary in alleviating a lot of undue suffering. We as novices have taken sincere efforts to make this project a successful one.

OUTPUT

100

100

ACCIDENT

IF YOU ARE INTERESTED

100

100

100

REFERENCES

REFERENCES

1. The 8051 Micro Controller Architecture and Applications, Kenneth. J. Ayala.
2. Handbook of 8 bit micro controller, Intel Corporation 1989 USA.
3. Linear Integrated Circuits, D. Roy Choudhury & Shail Jain, New Age International P. Ltd, 1991.
4. Microprocessors and Microcomputer Based System Design, Mohamed Rafiquzzman, UBS Publisher's Distributors Ltd.
5. www.atmel.com
6. www.Iweil.com/micro/8051/8951atmel.com
7. Statistics on road accidents-TamilNadu state transport authority.htm

APPENDIX

PIN OUT

ATMEL 89C51

P1.0	□	1		40	□	VCC
P1.1	□	2		39	□	P0.0 (AD0)
P1.2	□	3	A	38	□	P0.1 (AD1)
P1.3	□	4		37	□	P0.2 (AD2)
P1.4	□	5	T	36	□	P0.3 (AD3)
P1.5	□	6		35	□	P0.4 (AD4)
P1.6	□	7	8	34	□	P0.5 (AD5)
P1.7	□	8		33	□	P0.6 (AD6)
RST	□	9	9	32	□	P0.7 (AD7)
(RXD) P3.0	□	10		31	□	$\overline{EA/VPP}$
(TXD) P3.1	□	11	C	30	□	$\overline{ALE/PROG}$
(INT0) P3.2	□	12		29	□	\overline{PSEN}
(INT1) P3.3	□	13	5	28	□	P2.7 (A15)
(T0) P3.4	□	14		27	□	P2.6 (A14)
(T1) P3.5	□	15	1	26	□	P2.5 (A13)
(WR) P3.6	□	16		25	□	P2.4 (A12)
(RD) P3.7	□	17		24	□	P2.3 (A11)
XTAL2	□	18		23	□	P2.2 (A10)
XTAL1	□	19		22	□	P2.1 (A9)
GND	□	20		21	□	P2.0 (A8)

controller Instruction Set

For interrupt response time information, refer to the hardware description chapter.

Operations that Affect Flag Settings⁽¹⁾

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
	X	X	X	CLR C	0		
	X	X	X	CPL C	X		
	X	X	X	ANL C,bit	X		
	0	X		ANL C,/bit	X		
	0	X		ORL C,bit	X		
	X			ORL C,/bit	X		
	X			MOV C,bit	X		
	X			CJNE	X		
3 C	1						

- Operations on SFR byte address 208 or bit addresses 209-215 (that is, the PSW or bits in the PSW) also affect flag settings.

Instruction Set and Addressing Modes

	Register R7-R0 of the currently selected Register Bank.
Address	8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR [i.e., I/O port, control register, status register, etc. (128-255)].
Address	8-bit internal data RAM location (0-255) addressed indirectly through register R1 or R0.
Constant	8-bit constant included in instruction.
Constant 16	16-bit constant included in instruction.
Destination 16	16-bit destination address. Used by LCALL and LJMP. A branch can be anywhere within the 64K byte Program Memory address space.
Destination 11	11-bit destination address. Used by ACALL and AJMP. The branch will be within the same 2K byte page of program memory as the first byte of the following instruction.
Offset	Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.
Source	Direct Addressed bit in Internal Data RAM or Special Function Register.



Instruction Set



Instruction Set Summary

	0	1	2	3	4	5	6	7
	NOP	JBC bit, rel [3B, 2C]	JB bit, rel [3B, 2C]	JNB bit, rel [3B, 2C]	JC rel [2B, 2C]	JNC rel [2B, 2C]	JZ rel [2B, 2C]	JNZ rel [2B, 2C]
	AJMP (P0) [2B, 2C]	ACALL (P0) [2B, 2C]	AJMP (P1) [2B, 2C]	ACALL (P1) [2B, 2C]	AJMP (P2) [2B, 2C]	ACALL (P2) [2B, 2C]	AJMP (P3) [2B, 2C]	ACALL (P3) [2B, 2C]
	LJMP addr16 [3B, 2C]	LCALL addr16 [3B, 2C]	RET [2C]	RETI [2C]	ORL dir, A [2B]	ANL dir, A [2B]	XRL dir, a [2B]	ORL C, bit [2B, 2C]
	RR A	RRC A	RL A	RLC A	ORL dir, #data [3B, 2C]	ANL dir, #data [3B, 2C]	XRL dir, #data [3B, 2C]	JMP @A + DPTR [2C]
	INC A	DEC A	ADD A, #data [2B]	ADDC A, #data [2B]	ORL A, #data [2B]	ANL A, #data [2B]	XRL A, #data [2B]	MOV A, #data [2B]
	INC dir [2B]	DEC dir [2B]	ADD A, dir [2B]	ADDC A, dir [2B]	ORL A, dir [2B]	ANL A, dir [2B]	XRL A, dir [2B]	MOV dir, #data [3B, 2C]
	INC @R0	DEC @R0	ADD A, @R0	ADDC A, @R0	ORL A, @R0	ANL A, @R0	XRL A, @R0	MOV @R0, #data [2B]
	INC @R1	DEC @R1	ADD A, @R1	ADDC A, @R1	ORL A, @R1	ANL A, @R1	XRL A, @R1	MOV @R1, #data [2B]
B	INC R0	DEC R0	ADD A, R0	ADDC A, R0	ORL A, R0	ANL A, R0	XRL A, R0	MOV R0, #data [2B]
9	INC R1	DEC R1	ADD A, R1	ADDC A, R1	ORL A, R1	ANL A, R1	XRL A, R1	MOV R1, #data [2B]
A	INC R2	DEC R2	ADD A, R2	ADDC A, R2	ORL A, R2	ANL A, R2	XRL A, R2	MOV R2, #data [2B]
B	INC R3	DEC R3	ADD A, R3	ADDC A, R3	ORL A, R3	ANL A, R3	XRL A, R3	MOV R3, #data [2B]
C	INC R4	DEC R4	ADD A, R4	ADDC A, R4	ORL A, R4	ANL A, R4	XRL A, R4	MOV R4, #data [2B]
D	INC R5	DEC R5	ADD A, R5	ADDC A, R5	ORL A, R5	ANL A, R5	XRL A, R5	MOV R5, #data [2B]
E	INC R6	DEC R6	ADD A, R6	ADDC A, R6	ORL A, R6	ANL A, R6	XRL A, R6	MOV R6, #data [2B]
F	INC R7	DEC R7	ADD A, R7	ADDC A, R7	ORL A, R7	ANL A, R7	XRL A, R7	MOV R7, #data [2B]

Key: [2B] = 2 Byte, [3B] = 3 Byte, [2C] = 2 Cycle, [4C] = 4 Cycle, Blank = 1 byte/1 cycle

Instruction Set

Instruction Set Summary (Continued)

	8	9	A	B	C	D	E	F
	SJMP REL [2B, 2C]	MOV DPTR,# data 16 [3B, 2C]	ORL C, /bit [2B, 2C]	ANL C, /bit [2B, 2C]	PUSH dir [2B, 2C]	POP dir [2B, 2C]	MOVX A, @DPTR [2C]	MOVX @DPTR, A [2C]
	AJMP (P4) [2B, 2C]	ACALL (P4) [2B, 2C]	AJMP (P5) [2B, 2C]	ACALL (P5) [2B, 2C]	AJMP (P6) [2B, 2C]	ACALL (P6) [2B, 2C]	AJMP (P7) [2B, 2C]	ACALL (P7) [2B, 2C]
	ANL C, bit [2B, 2C]	MOV bit, C [2B, 2C]	MOV C, bit [2B]	CPL bit [2B]	CLR bit [2B]	SETB bit [2B]	MOVX A, @R0 [2C]	MOVX wR0, A [2C]
	MOVC A, @A + PC [2C]	MOVC A, @A + DPTR [2C]	INC DPTR [2C]	CPL C	CLR C	SETB C	MOVX A, @R1 [2C]	MOVX @R1, A [2C]
	DIV AB [2B, 4C]	SUBB A, #data [2B]	MUL AB [4C]	CJNE A, #data, rel [3B, 2C]	SWAP A	DA A	CLR A	CPL A
	MOV dir, dir [3B, 2C]	SUBB A, dir [2B]		CJNE A, dir, rel [3B, 2C]	XCH A, dir [2B]	DJNZ dir, rel [3B, 2C]	MOV A, dir [2B]	MOV dir, A [2B]
	MOV dir, @R0 [2B, 2C]	SUBB A, @R0	MOV @R0, dir [2B, 2C]	CJNE @R0, #data, rel [3B, 2C]	XCH A, @R0	XCHD A, @R0	MOV A, @R0	MOV @R0, A
	MOV dir, @R1 [2B, 2C]	SUBB A, @R1	MOV @R1, dir [2B, 2C]	CJNE @R1, #data, rel [3B, 2C]	XCH A, @R1	XCHD A, @R1	MOV A, @R1	MOV @R1, A
	MOV dir, R0 [2B, 2C]	SUBB A, R0	MOV R0, dir [2B, 2C]	CJNE R0, #data, rel [3B, 2C]	XCH A, R0	DJNZ R0, rel [2B, 2C]	MOV A, R0	MOV R0, A
	MOV dir, R1 [2B, 2C]	SUBB A, R1	MOV R1, dir [2B, 2C]	CJNE R1, #data, rel [3B, 2C]	XCH A, R1	DJNZ R1, rel [2B, 2C]	MOV A, R1	MOV R1, A
	MOV dir, R2 [2B, 2C]	SUBB A, R2	MOV R2, dir [2B, 2C]	CJNE R2, #data, rel [3B, 2C]	XCH A, R2	DJNZ R2, rel [2B, 2C]	MOV A, R2	MOV R2, A
	MOV dir, R3 [2B, 2C]	SUBB A, R3	MOV R3, dir [2B, 2C]	CJNE R3, #data, rel [3B, 2C]	XCH A, R3	DJNZ R3, rel [2B, 2C]	MOV A, R3	MOV R3, A
	MOV dir, R4 [2B, 2C]	SUBB A, R4	MOV R4, dir [2B, 2C]	CJNE R4, #data, rel [3B, 2C]	XCH A, R4	DJNZ R4, rel [2B, 2C]	MOV A, R4	MOV R4, A
	MOV dir, R5 [2B, 2C]	SUBB A, R5	MOV R5, dir [2B, 2C]	CJNE R5, #data, rel [3B, 2C]	XCH A, R5	DJNZ R5, rel [2B, 2C]	MOV A, R5	MOV R5, A
	MOV dir, R6 [2B, 2C]	SUBB A, R6	MOV R6, dir [2B, 2C]	CJNE R6, #data, rel [3B, 2C]	XCH A, R6	DJNZ R6, rel [2B, 2C]	MOV A, R6	MOV R6, A
	MOV dir, R7 [2B, 2C]	SUBB A, R7	MOV R7, dir [2B, 2C]	CJNE R7, #data, rel [3B, 2C]	XCH A, R7	DJNZ R7, rel [2B, 2C]	MOV A, R7	MOV R7, A

Key: [2B] = 2 Byte, [3B] = 3 Byte, [2C] = 2 Cycle, [4C] = 4 Cycle, Blank = 1 byte/1 cycle



AT89 Instruction Set Summary⁽¹⁾

Mnemonic	Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS			
A,R _n	Add register to Accumulator	1	12
A,direct	Add direct byte to Accumulator	2	12
A,@R _i	Add indirect RAM to Accumulator	1	12
A,#data	Add immediate data to Accumulator	2	12
A,R _n	Add register to Accumulator with Carry	1	12
A,direct	Add direct byte to Accumulator with Carry	2	12
A,@R _i	Add indirect RAM to Accumulator with Carry	1	12
A,#data	Add immediate data to Acc with Carry	2	12
A,R _n	Subtract Register from Acc with borrow	1	12
A,direct	Subtract direct byte from Acc with borrow	2	12
A,@R _i	Subtract indirect RAM from ACC with borrow	1	12
A,#data	Subtract immediate data from Acc with borrow	2	12
A	Increment Accumulator	1	12
R _n	Increment register	1	12
direct	Increment direct byte	2	12
@R _i	Increment direct RAM	1	12
A	Decrement Accumulator	1	12
R _n	Decrement Register	1	12
direct	Decrement direct byte	2	12
@R _i	Decrement indirect RAM	1	12
DPTR	Increment Data Pointer	1	24
AB	Multiply A & B	1	48
AB	Divide A by B	1	48
A	Decimal Adjust Accumulator	1	12

1. All mnemonics copyrighted © Intel Corp., 1980.

Mnemonic	Description	Byte	Oscillator Period	
LOGICAL OPERATIONS				
ANL	A,R _n	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@R _i	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,R _n	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@R _i	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,R _n	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@R _i	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
LOGICAL OPERATIONS (continued)				

Instruction Set

Mnemonic	Description	Byte	Oscillator Period
RA	Rotate Accumulator Right	1	12
RA	Rotate Accumulator Right through the Carry	1	12
RA	Swap nibbles within the Accumulator	1	12
DATA TRANSFER			
MOV A,R _n	Move register to Accumulator	1	12
MOV A,direct	Move direct byte to Accumulator	2	12
MOV A,@R _i	Move indirect RAM to Accumulator	1	12
MOV A,#data	Move immediate data to Accumulator	2	12
MOV R _n ,A	Move Accumulator to register	1	12
MOV R _n ,direct	Move direct byte to register	2	24
MOV R _n ,#data	Move immediate data to register	2	12
MOV direct,A	Move Accumulator to direct byte	2	12
MOV direct,R _n	Move register to direct byte	2	24
MOV direct,direct	Move direct byte to direct	3	24
MOV direct,@R _i	Move indirect RAM to direct byte	2	24
MOV direct,#data	Move immediate data to direct byte	3	24
MOV @R _i ,A	Move Accumulator to indirect RAM	1	12
MOV @R _i ,direct	Move direct byte to indirect RAM	2	24
MOV @R _i ,#data	Move immediate data to indirect RAM	2	12
MOV DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOV A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOV A,@A+PC	Move Code byte relative to PC to Acc	1	24
MOVX A,@R _i	Move External RAM (8-bit addr) to Acc	1	24

DATA TRANSFER (continued)

Mnemonic	Description	Byte	Oscillator Period
MOVX A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX @R _i ,A	Move Acc to External RAM (8-bit addr)	1	24
MOVX @DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH direct	Push direct byte onto stack	2	24
POP direct	Pop direct byte from stack	2	24
XCH A,R _n	Exchange register with Accumulator	1	12
XCH A,direct	Exchange direct byte with Accumulator	2	12
XCH A,@R _i	Exchange indirect RAM with Accumulator	1	12
XCHD A,@R _i	Exchange low-order Digit indirect RAM with Acc	1	12
BOOLEAN VARIABLE MANIPULATION			
CLR C	Clear Carry	1	12
CLR bit	Clear direct bit	2	12
SETB C	Set Carry	1	12
SETB bit	Set direct bit	2	12
CPL C	Complement Carry	1	12
CPL bit	Complement direct bit	2	12
ANL C,bit	AND direct bit to CARRY	2	24
ANL C,/bit	AND complement of direct bit to Carry	2	24
ORL C,bit	OR direct bit to Carry	2	24
ORL C,/bit	OR complement of direct bit to Carry	2	24
MOV C,bit	Move direct bit to Carry	2	12
MOV bit,C	Move Carry to direct bit	2	24
JC rel	Jump if Carry is set	2	24
JNC rel	Jump if Carry not set	2	24
JB bit,rel	Jump if direct Bit is set	3	24
JNB bit,rel	Jump if direct Bit is Not set	3	24
JBC bit,rel	Jump if direct Bit is set & clear bit	3	24
PROGRAM BRANCHING			



Op Code	Description	Byte	Oscillator Period
addr11	Absolute Subroutine Call	2	24
addr16	Long Subroutine Call	3	24
	Return from Subroutine	1	24
	Return from interrupt	1	24
addr11	Absolute Jump	2	24
addr16	Long Jump	3	24
rel	Short Jump (relative addr)	2	24
@A+DPTR	Jump indirect relative to the DPTR	1	24
rel	Jump if Accumulator is Zero	2	24
rel	Jump if Accumulator is Not Zero	2	24
A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
R _n ,#data,rel	Compare immediate to register and Jump if Not Equal	3	24
@R _i ,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
R _n ,rel	Decrement register and Jump if Not Zero	2	24
direct,rel	Decrement direct byte and Jump if Not Zero	3	24
	No Operation	1	12

Instruction Set

Instruction Opcodes in Hexadecimal Order

	Number of Bytes	Mnemonic	Operands
	1	NOP	
	2	AJMP	code addr
	3	LJMP	code addr
	1	RR	A
	1	INC	A
	2	INC	data addr
	1	INC	@R0
	1	INC	@R1
	1	INC	R0
	1	INC	R1
	1	INC	R2
	1	INC	R3
	1	INC	R4
	1	INC	R5
	1	INC	R6
	1	INC	R7
	3	JBC	bit addr,code addr
	2	ACALL	code addr
	3	LCALL	code addr
	1	RRC	A
	1	DEC	A
	2	DEC	data addr
	1	DEC	@R0
	1	DEC	@R1
	1	DEC	R0
	1	DEC	R1
	1	DEC	R2
	1	DEC	R3
	1	DEC	R4
	1	DEC	R5
	1	DEC	R6
	1	DEC	R7
	3	JB	bit addr,code addr
	2	AJMP	code addr
	1	RET	
	1	RL	A
	2	ADD	A,#data
	2	ADD	A,data addr

Hex Code	Number of Bytes	Mnemonic	Operands
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr,code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2



	Number of Bytes	Mnemonic	Operands
	1	ORL	A,R3
	1	ORL	A,R4
	1	ORL	A,R5
	1	ORL	A,R6
	1	ORL	A,R7
	2	JNC	code addr
	2	ACALL	code addr
	2	ANL	data addr,A
	3	ANL	data addr,#data
	2	ANL	A,#data
	2	ANL	A,data addr
	1	ANL	A,@R0
	1	ANL	A,@R1
	1	ANL	A,R0
	1	ANL	A,R1
	1	ANL	A,R2
	1	ANL	A,R3
	1	ANL	A,R4
	1	ANL	A,R5
	1	ANL	A,R6
	1	ANL	A,R7
	2	JZ	code addr
	2	AJMP	code addr
	2	XRL	data addr,A
	3	XRL	data addr,#data
	2	XRL	A,#data
	2	XRL	A,data addr
	1	XRL	A,@R0
	1	XRL	A,@R1
	1	XRL	A,R0
	1	XRL	A,R1
	1	XRL	A,R2
	1	XRL	A,R3
	1	XRL	A,R4
	1	XRL	A,R5
	1	XRL	A,R6
	1	XRL	A,R7
	2	JNZ	code addr

Hex Code	Number of Bytes	Mnemonic	Operands
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr,data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0

Instruction Set

	Number of Bytes	Mnemonic	Operands
	1	SUBB	A,@R1
	1	SUBB	A,R0
	1	SUBB	A,R1
	1	SUBB	A,R2
	1	SUBB	A,R3
	1	SUBB	A,R4
	1	SUBB	A,R5
	1	SUBB	A,R6
	1	SUBB	A,R7
	2	ORL	C,/bit addr
	2	AJMP	code addr
	2	MOV	C,/bit addr
	1	INC	DPTR
	1	MUL	AB
		reserved	
	2	MOV	@R0,data addr
	2	MOV	@R1,data addr
	2	MOV	R0,data addr
	2	MOV	R1,data addr
	2	MOV	R2,data addr
	2	MOV	R3,data addr
	2	MOV	R4,data addr
	2	MOV	R5,data addr
	2	MOV	R6,data addr
	2	MOV	R7,data addr
	2	ANL	C,/bit addr
	2	ACALL	code addr
	2	CPL	bit addr
	1	CPL	C
	3	CJNE	A,#data,code addr
	3	CJNE	A,data addr,code addr
	3	CJNE	@R0,#data,code addr
	3	CJNE	@R1,#data,code addr
	3	CJNE	R0,#data,code addr
	3	CJNE	R1,#data,code addr
	3	CJNE	R2,#data,code addr
	3	CJNE	R3,#data,code addr
	3	CJNE	R4,#data,code addr

Hex Code	Number of Bytes	Mnemonic	Operands
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0



	Number of Bytes	Mnemonic	Operands
	1	MOVX	A,@R1
	1	CLR	A
	2	MOV	A,data addr
	1	MOV	A,@R0
	1	MOV	A,@R1
	1	MOV	A,R0
	1	MOV	A,R1
	1	MOV	A,R2
	1	MOV	A,R3
	1	MOV	A,R4
	1	MOV	A,R5
	1	MOV	A,R6
	1	MOV	A,R7
	1	MOVX	@DPTR,A
	2	ACALL	code addr
	1	MOVX	@R0,A
	1	MOVX	@R1,A
	1	CPL	A
	2	MOV	data addr,A
	1	MOV	@R0,A
	1	MOV	@R1,A
	1	MOV	R0,A
	1	MOV	R1,A
	1	MOV	R2,A
	1	MOV	R3,A
	1	MOV	R4,A
	1	MOV	R5,A
	1	MOV	R6,A
	1	MOV	R7,A