



**BUS DETECTION DEVICE WITH KEYPAD
DESIGNED FOR THE BLIND**



A PROJECT REPORT

Submitted by

DEEPIKA K. 0710105015
MANIVANNAN T. 0710105032
RAMESH M. 0710105040
VINOTH KUMAR S. 0710105059

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRICAL AND ELECTRONICS ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY,

COIMBATORE – 641049

ANNA UNIVERSITY OF TECHNOLOGY : COIMBATORE

APRIL- 2011

ANNA UNIVERSITY OF TECHNOLOGY : COIMBATORE

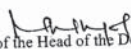
APRIL- 2011

BONAFIDE CERTIFICATE

Certified that this project report "**BUS DETECTION DEVICE WITH SPECIAL KEYPAD DESIGNED FOR THE BLIND**" is the bonafide work of


- | | |
|---------------------|------------------------|
| 1. Deepika.K | Register No.0710105015 |
| 2. Manivannan.T. | Register No.0710105032 |
| 3. Ramesh.M. | Register No.0710105040 |
| 4. Vinoth Kumar. S. | Register No.0710105059 |

who carried out the project work under my supervision.


Signature of the Head of the Department

Dr. Rani Thottungal
EEE

Signature of the supervisor


Mrs.D.Rajalakshmi
Asst. Professor

The candidates with University Register Nos.0710105015, 0710105032, 0710105040, 0710105059 was examined by us in the project viva-voce examination held on 18.04.11.


INTERNAL EXAMINER


EXTERNAL EXAMINER

ABSTRACT

In this project a bus detection device is designed using the GPS and GSM technologies to locate the current position of the bus and also a special keypad is designed for the visually disabled persons based on the Braille code.

The GPS receiver fixed in bus gives the current location of the bus. Using the GSM technology a microcontroller is programmed that sends the current location of the bus in SMS (Short Messaging Service) format. The location of the bus stand is known and constant. Hence a program that works on a Visual Basic platform is developed that finds the difference between the two different latitude-longitude positions and convert them to distance in kilometers. An average speed of 30kms/hr is fixed and with the known distance and speed, the time is calculated and given in the form of voice output.

The keypad for the visually disabled is designed that works based on a program in the PIC microcontroller to implement the Braille code. The distance to be covered along with the time the particular bus takes to reach the stand is given in the form of voice output.

ACKNOWLEDGEMENT

It is our bounden duty to thank contribution made in one form or the other by the individuals we hereby acknowledge.

We express our heart-felt gratitude and thanks to the Dean / HoD of Electrical and Electronics Engineering, **Dr. Rani Thottungal** for encouraging us and for being with us right from beginning of the project and fine tuning us at every step.

We wish to place on record our deep sense of gratitude and profound thanks to our guide **Ms.D.Rajalakshmi**, Asst. Professor, Electrical and Electronics Engineering Department, for her valuable guidance, constant encouragement and continuous support rendered throughout the project.

We are also thankful to all the reviewers for their valuable comments and encouragement.

Last but not the least, we extend our sincere thanks to all the teaching and non-teaching staff who have contributed their ideas and encouraged us for completing the project.

CONTENTS

Title	Page No.
Diploma Certificate	i
Abstract	ii
Acknowledgement	iii
Contents	iv
List of Figures	vii
List of Tables	vii
List of symbols and abbreviations	viii

CHAPTER 1: INTRODUCTION

1.1. Objective	1
1.2. Need of the project	1
1.3. Proposed system	1
1.4. Organisation of the report	2

CHAPTER 2: GPS AND GSM TECHNOLOGY

2.1.How GPS works	3
2.2.Signals	3
2.3.Timing & correction	4
2.4.Mapping	4
2.5.Accuracy	5

5.3.2 Three Terminal Voltage Regulators	12
5.4 Relay	13
5.5.PIC Microcontroller	15
5.5.1.PIC (16F877)	15
5.5.2.Core features	16
5.5.3.Peripheral features	17
5.5.4. Architecture of PIC 16F877	19
5.6.RS232 Communication	20
5.6.1.RS232	21
5.6.2.Scope of the standard	21
5.6.3.Circuit working description	21
5.7.Keypad	22
5.8.Experimental setup	24

CHAPTER 6: PROCESS AT THE COMPUTER TERMINAL

6.1. VB program to calculate the distance & time	26
6.2.Explanation	31
6.2. Output screen	32

CHAPTER 3:VISUAL BASIC

3.1.Visual Basic	6
3.2.Characteristics	6
3.3.Performance and other issues	7
3.4.Example code	7

CHAPTER 4: OVERVIEW OF THE PROJECT

4.1.Block diagram	8
4.2. Requirements of the system design	9
4.2.1Components for bus detection device	9
4.2.2.Components for keypad	9

CHAPTER 5: HARDWARE DESCRIPTION

5.1.Circuit diagram for Bus Detection Device	10
5.2.Circuit diagram for Keypad	11
5.3.Power Supply	
5.3..1 IC Voltage Regulators	12

CHAPTER 7: CONCLUSION AND SCOPE

7.1.Conclusion	33
7.2.Scope for Future	33

REFERENCES

APPENDIX A PIC 16F877A CODING	35
-------------------------------	----

APPENDIX B PIC 16F877A DATA SHEET	62
-----------------------------------	----

LIST OF FIGURES

Figure	Title	Page No.
4.1.1.	Block Diagram	8
5.1.1.	Circuit diagram for bus detection device	10
5.2.1.	Circuit diagram for Keypad	11
5.3.	Power supply circuit	12
5.3.2	Fixed positive voltage regulators	13
5.4.	Relay circuit	14
5.5.4.	Architecture of PIC 16F877	16
5.5.5	Pin diagram of PIC 16F877	17

LIST OF TABLES

Table	Title	Page No.
5.5.4	Specifications of PIC16F877	17
5.6	Function tables	21

LIST OF SYMBOLS AND ABBREVIATIONS

No.	Symbol/Abbreviation	Description
1.	GPS	Global positioning system
2.	GSM	Global system for mobile communication
3.	RS232	Voltage induced in inductor L1
4.	PIC	Programmable interface controller
5.	D	Diode
6.	GND	Ground
7.	V	Voltage
8.	C	Capacitor
9.	R	Resistor
10.	DC	Direct Current
11.	AC	Alternating Current

CHAPTER 1

INTRODUCTION

1.1.OBJECTIVE

The main aim of the project is to create a bus detection device that can tell the user the approximate time a bus takes to reach the bus stand from its current location in the form of voice output using the GPS and GSM technology and VB programming based on a formula.

The other objective of the project is to design a special keypad for the visually disabled to enter the bus number based on the Braille code.

1.2.NEED OF THE PROJECT

There is no system or technology solution that is applied to calculate the time a bus takes to reach the destination from its current position. People wait for a long time in the bus stand in order to reach their destination without knowing as to at what time the correct bus reaches the stand. There are visually disabled people who are dependent on others to enquire about the buses. Hence a system is needed that can offer a solution to the above explained. Thus a device that can calculate the time based on locations is required. Visually disabled persons always want to be independent. As the Braille code is their most comfortable language, we are required to develop a keypad that can provide a solution to them.

1.3.PROPOSED SYSTEM

The GPS receiver fixed in bus gives the current location of the bus. With the known location of the bus stand the distance close to real value is calculated. Software developed for the Computer will give the time the bus reaches the stand. BRAILLE keypads are

made and interfaced to the system through microcontroller. The informations such as time the bus reaches and distance to be covered are given as output through the speaker.

1.4.ORGANISATION OF THE REPORT

This report has been organized into seven chapters.

Chapter 1: Gives the objective and need of the project along with proposed system and the way the various chapters are organized.

Chapter 2: Explains the technology details of GPS and GSM that does the wireless transmission of data.

Chapter 3: Describes the concept of Visual Basic that does the real time calculation.

Chapter 4: Gives an overall view of the project design along with the requirements.

Chapter 5: Describes the hardware implementation of the project.

Chapter 6: Presents the Visual basic program that is used in the project.

Chapter 7: Concludes the project with the scope for future work.

CHAPTER 2

GPS AND GSM

2.1.HOW GPS WORKS

For those who are unfamiliar with the term, GPS stands for Global Positioning System, and is a way of locating a receiver in three dimensional space anywhere on the Earth, and even in orbit about it.

GPS is arguably one of the most important inventions of our time, and has so many different applications that many technologies and ways of working are continually being improved in order to make the most of it.

2.2.SIGNALS

In order for GPS to work, a network of satellites was placed into orbit around planet Earth, each broadcasting a specific signal, much like a normal radio signal. This signal can be received by a low cost, low technology aerial, even though the signal is very weak.

Rather than carrying an actual radio or television program, the signals that are broadcast by the satellites carry data that is passed from the aerial, decoded and used by the GPS software.

The information is specific enough that the GPS software can identify the satellite, it's location in space, and calculate the time that the signal took to travel from the satellite to the GPS receiver.

Using different signals from different satellites, the GPS software is able to calculate the position of the receiver. The principle is very similar to that which is used in orienteering – if you can identify three places on your map, take a bearing to where they are, and draw three lines on the map, then you will find out where you are on the map.

2.5.ACCURACY

GSM tracking relies on Cell Identification, or Cell-ID – the means by which the phone networks track individual phones from cell to cell. Positional accuracy relies on the number of mobile masts in the area, so it can be accurate to 100 metres or as imprecise as within 10km or more.

The lines will intersect, and, depending on the accuracy of the bearings, the triangle that they form where they intersect will approximate your position, within a margin of error.

GPS software performs a similar kind of exercise, using the known positions of the satellites in space, and measuring the time that the signal has taken to travel from the satellite to Earth.

The result of the “trilateration” (the term used when distances are used instead of bearings) of at least three satellites, assuming that the clocks are all synchronized enables the software to calculate, within a margin of error, where the device is located in terms of its latitude (East-West) and longitude (North-South) and distance from the center of the Earth.

2.3.TIMING & CORRECTION

Since the satellites each contain atomic clocks which are extremely accurate, and certainly accurate with respect to each other, we can assume that most of the problem lies with the clock inside the GPS unit itself.

There are a few solutions. However the solution that was chosen uses a fourth satellite to provide a cross check in the trilateration process. Since trilateration from three signals should pinpoint the location exactly, adding a fourth will move that location; that is, it will not intersect with the calculated location.

This indicates to the GPS software that there is a discrepancy, and so it performs an additional calculation to find a value that it can use to adjust all the signals so that the four lines intersect.

2.4.MAPPING

With GPS, the traditional route has been that the user pays for the hardware, connection and applications separately. LBS tracking is structured differently; the user pays for the phone, while the services are provided by the network operator or applications provider.

CHAPTER 3

VISUAL BASIC

3.1.WHAT IS VB?

Visual Basic (VB) is the third-generation event-driven programming language and integrated development environment (IDE) from Microsoft for its COM programming model. Visual Basic is relatively easy to learn and use.^{[1][2]}

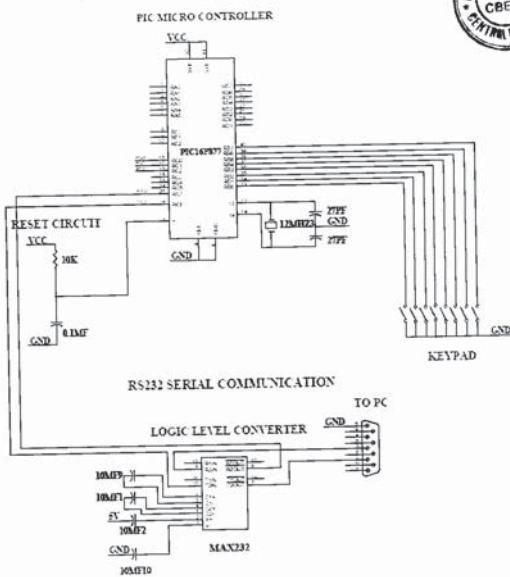
3.2.CHARACTERISTICS

Visual Basic has the following traits which differ from C-derived languages:

- Multiple assignment available in C language is not possible
- Boolean constant True has numeric value -1.^[4]
- Logical and bitwise operators are unified.
- Variable array base.
- OPTION BASE was introduced by ANSI, with the standard for ANSI Minimal BASIC in the late 1970s.
- Relatively strong integration with the Windows operating system and the Component Object Model. The native types for strings and arrays are the dedicated COM types, BSTR and SAFEARRAY.
- Banker's rounding as the default behavior when converting real numbers to integers with the Round function.^[6] ? Round(2.5, 0) gives 2, ? Round(3.5, 0) gives 4.
- Integers are automatically promoted to reals in expressions involving the normal division operator (/) so that division of one integer by another produces the intuitively correct result. There is a specific integer divide operator (\) which does truncate.
- By default, if a variable has not been declared or if no type declaration character is specified, the variable is of type Variant.

5.2.CIRCUIT DIAGRAM FOR KEYPAD

Fig.5.2.1.Circuit diagram for Keypad



P- 3423

5.3.POWER SUPPLY

The ac voltage, typically 220V rms, is connected to a transformer, which steps that ac voltage down to the level of the desired dc output. A diode rectifier then provides 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 variation.

Figure shows the basic connection of a three-terminal voltage regulator IC to a load. The fixed voltage regulator has an unregulated dc input voltage, V_i , applied to one input terminal, a regulated output dc voltage, V_o , from a second terminal, with the third terminal connected to ground. For a selected regulator, IC device specifications list a voltage range over which the input voltage can vary to maintain a regulated output voltage over a range of load current. The specifications also list the amount of output voltage change resulting from a change in load current (load regulation) or in input voltage (line regulation).

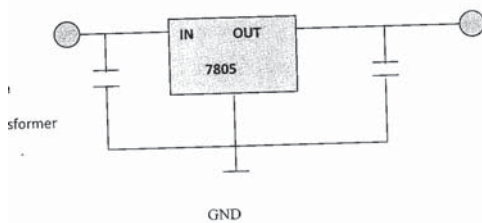
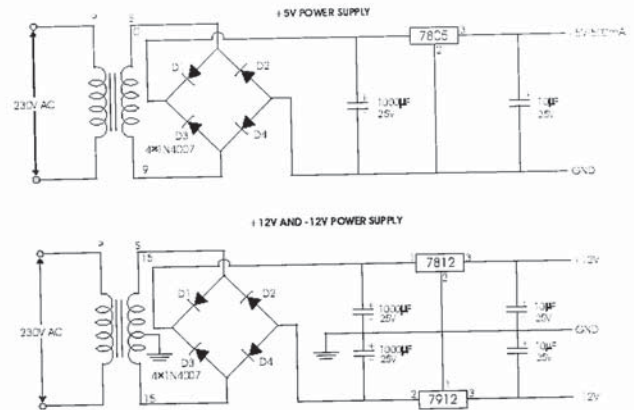


Fig 5.3.2 fixed positive voltage regulators

The series 78 regulators provide fixed regulated voltages from 5 to 24 V. Figure shows how one such IC, a 7812, is connected to provide voltage regulation with output from this unit of +12V dc. An unregulated input voltage V_i is filtered by capacitor C1 and connected to the IC's IN terminal. The IC's OUT terminal provides a regulated +12V which is filtered by capacitor C2 (mostly for any high-frequency noise). The third IC terminal is connected to ground (GND). While the input voltage may vary over some permissible voltage range, and the output load may vary over some acceptable range, the output voltage remains constant within specified voltage variation limits.

Fig 5.3.power supply circuit



A regulator circuit removes the ripples and also remains the same dc value even if the input dc voltage varies, or the load connected to the output dc voltage changes. This voltage regulation is usually obtained using one of the popular voltage regulator IC units.

5.3.1. IC VOLTAGE REGULATORS

A power supply can be built using a transformer connected to the ac supply line to step the ac voltage to a desired amplitude, then rectifying that ac voltage, filtering with a capacitor and RC filter, if desired, and finally regulating the dc voltage using an IC regulator. The regulators can be selected for operation with load currents from hundreds of milli amperes to tens of amperes, corresponding to power ratings from milliwatts to tens of watts.

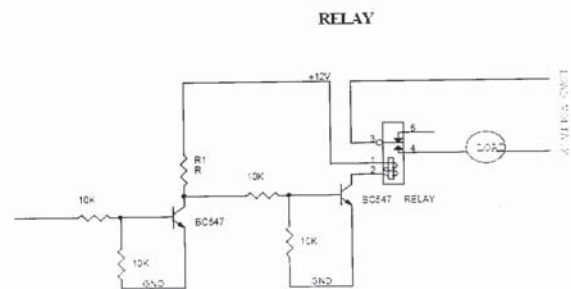
5.3.2. THREE-TERMINAL VOLTAGE REGULATORS

5.4.RELAY

A relay is an electrically operated switch. Current flowing through the coil of the relay creates a magnetic field which attracts a lever and changes the switch contacts. The coil current can be on or off so relays have two switch positions and they are double throw (changeover) switches.

Relays allow one circuit to switch a second circuit which can be completely separate from the first. For example a low voltage battery circuit can use a relay to switch a 230V AC mains circuit. There is no electrical connection inside the relay between the two circuits; the link is magnetic and mechanical.

Fig.5.4.Relay circuit



The coil of a relay passes a relatively large current, typically 30mA for a 12V relay, but it can be as much as 100mA for relays designed to operate from lower voltages. Most IC's (chips) cannot provide this current and a transistor is usually used to amplify the small IC

popular 555 timer IC is 200mA so these devices can supply relay coils directly without amplification.

5.5.PIC MICROCONTROLLER

The microcontroller that has been used for this project is from PIC series. PIC microcontroller is the first RISC based microcontroller fabricated in CMOS (complimentary metal oxide semiconductor) that uses separate bus for instruction and data allowing simultaneous access of program and data memory. The main advantage of CMOS and RISC combination is low power consumption resulting in a very small chip size with a small pin count. The main advantage of CMOS is that it has immunity to noise than other fabrication techniques.

5.5.1.PIC (16F877)

Various microcontrollers offer different kinds of memories. EEPROM, EPROM, FLASH etc. are some of the memories of which FLASH is the most recently developed. Technology that is used in pic16F877 is flash technology, so that data is retained even when the power is switched off. Easy Programming and Erasing are other features of PIC 16F877.

5.5.2.CORE FEATURES

- High-performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input

- Commercial and Industrial temperature ranges

- Low-power consumption:

< 2mA typical @ 5V, 4 MHz

20mA typical @ 3V, 32 kHz

< 1mA typical standby current

5.5.3.PERIPHERAL FEATURES

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during sleep via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
 - Capture is 16-bit, max resolution is 12.5 ns,
 - Compare is 16-bit, max resolution is 200 ns,
 - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI. (Master Mode) and I2C. (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection.
- Brown-out detection circuitry for Brown-out Reset (BOR)

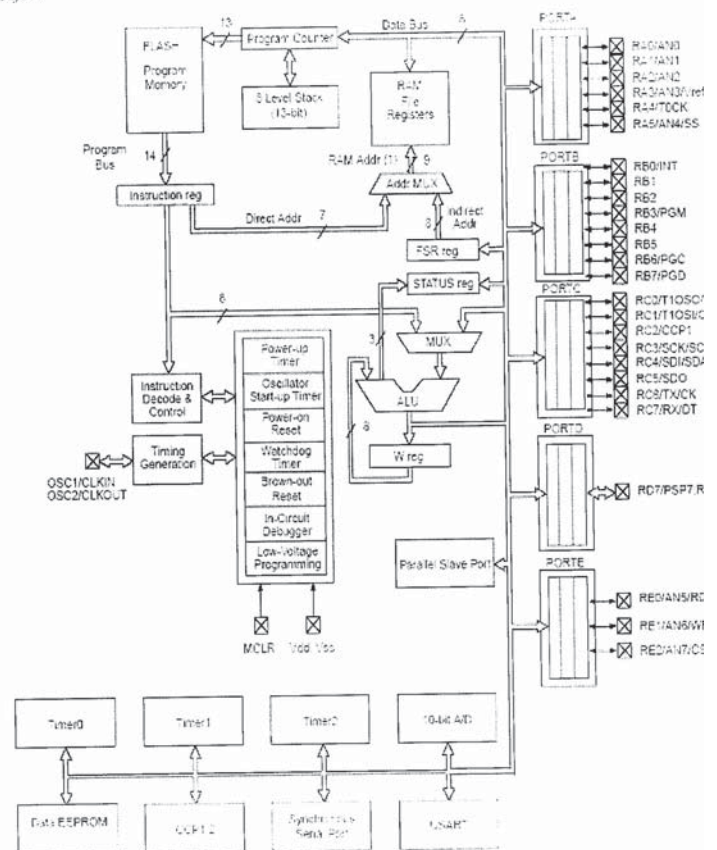
5.5.4.ARCHITECTURE OF PIC 16F877

The complete architecture of PIC 16F877 is shown in the fig 2.1. Table 2.1 gives details about the specifications of PIC 16F877. Fig 2.2 shows the complete pin diagram of the IC PIC 16F877.

DC - 200 ns instruction cycle

- Up to 8K x 14 words of Flash Program Memory
- Up to 368 x 8 bytes of Data Memory (RAM)
- Up to 256 x 8 bytes of EEPROM data memory
- Pin out compatible to the PIC16C73/74/76/77
- Interrupt capability (up to 14 internal/external)
- Eight level deep hardware stack
- Direct, indirect, and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC Oscillator for reliable operation
- Programmable code-protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low-power, high-speed CMOS EPROM/EEPROM technology
- Fully static design
- In-Circuit Serial Programming (ICSP) via two pins
- Only single 5V source needed for programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.5V to 5.5V
- High Sink/Source Current: 25 mA

fig 5.5.4 architecture of pic 16f877

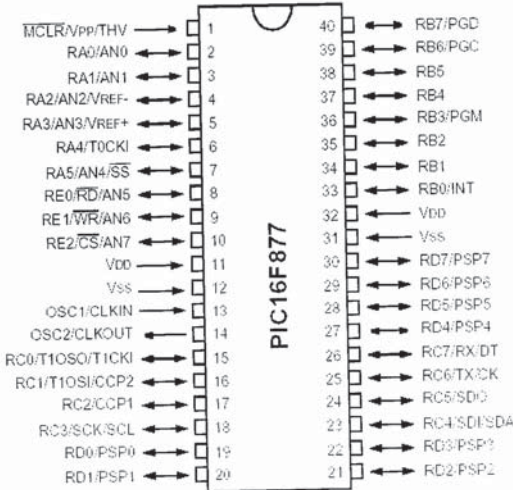


Note: 1 Higher order bits are from the STATUS register

table 5.5.4 specifications

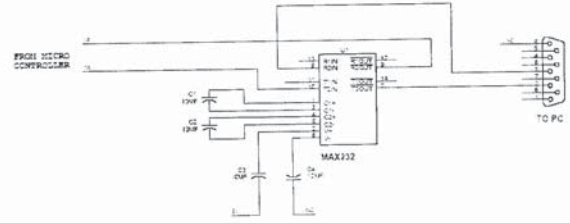
DEVICE	PROGRAM FLASH	DATA MEMORY	DATA EEPROM
PIC 16F877	8K	368 Bytes	256 Bytes

FIG 5.5.5 PIN DIAGRAM OF PIC 16F877



5.6.RS232 COMMUNICATION

Fig.5.6.1.MAX 232 circuit



In telecommunications, **RS-232** is a standard for serial binary data interconnection between a *DTE* (Data terminal equipment) and a *DCE* (Data Circuit-terminating Equipment). It is commonly used in computer serial ports.

5.6.1.SCOPE OF THE STANDARD

The Electronic Industries Alliance (EIA) standard RS-232-C [3] as of 1969 defines:

- Electrical signal characteristics such as voltage levels, signaling rate, timing and slew-rate of signals, voltage withstand level, short-circuit behavior, maximum stray capacitance and cable length
- Interface mechanical characteristics, pluggable connectors and pin identification
- Functions of each circuit in the interface connector
- Standard subsets of interface circuits for selected telecom applications

The standard does not define such elements as character encoding (for example, ASCII, Baudot or EBCDIC), or the framing of characters in the data stream (bits per character, start/stop bits, parity). The standard does not define protocols for error detection or algorithms for data compression.

The standard does not define bit rates for transmission, although the standard says it is intended for bit rates lower than 20,000 bits per second. Many modern devices can exceed this speed (38,400 and 57,600 bit/s being common, and 115,200 and 230,400 bit/s making occasional appearances) while still using RS-232 compatible signal levels.

Details of character format and transmission bit rate are controlled by the serial port hardware, often a single integrated circuit called a UART that converts data from parallel to serial form. A typical serial port includes specialized driver and receiver integrated circuits to convert between internal logic levels and RS-232 compatible signal levels.

5.6.2.CIRCUIT WORKING DESCRIPTION

In this circuit the MAX 232 IC used as level logic converter. The MAX232 is a dual driver/receiver that includes a capacitive voltage generator to supply EIA 232 voltage levels from a single 5v supply. Each receiver converts EIA-232 to 5v TTL/CMOS levels. Each driver converts TLL/CMOS input levels into EIA-232 levels.

Function Tables

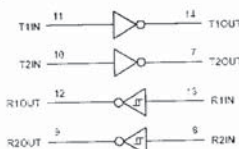
EACH DRIVER	
INPUT TIN	OUTPUT TOUT
L	H
H	L

H = High level, L = Low level

EACH RECEIVER	
INPUT RIN	OUTPUT ROUT
L	H
H	L

H = High level, L = Low level

logic diagram (positive logic)



In this circuit the microcontroller transmitter pin is connected in the MAX232 T2IN pin which converts input 5v TTL/CMOS level to RS232 level. Then T2OUT pin is connected to receiver pin of 9 pin D type serial connector which is directly connected to PC.

In PC the transmitting data is given to R2IN of MAX232 through transmitting pin of 9 pin D type connector which converts the RS232 level to 5v TTL/CMOS level. The R2OUT pin is connected to receiver pin of the microcontroller. Likewise the data is transmitted and received between the microcontroller and PC or other device vice versa.

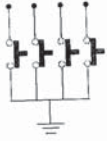
5.7.KEYPAD

A numeric keypad, or numpad for short, is the small, palm-sized, seventeen key section of a computer keyboard, usually on the very far right. The numeric keypad features digits 0 to 9, addition (+), subtraction (-), multiplication (*) and division (/) symbols, a decimal point (.) and Num Lock and Enter keys. Laptop keyboards often do not have a numpad, but may provide numpad input by holding a modifier key (typically labelled "Fn") and operating keys on the standard keyboard.

Particularly large laptops (typically those with a 17 inch screen or larger) may have space for a real numpad, and many companies sell separate numpads which connect to the host laptop by a USB connection.

Numeric keypads usually operate in two modes: when Num Lock is off, keys 8, 6, 2, 4 act like an arrow keys and 7, 9, 3, 1 act like Home, PgUp, PgDn and End; when Num Lock is on, digits keys produce corresponding digits. These, however, differ from the numeric keys at the top of the keyboard in that, when combined with the Alt key on a PC, they are used to enter characters which may not be otherwise available: for example, Alt-0169 produces the copyright symbol. These are referred to as Alt codes.

On Apple Computer Macintosh computers, which lack a Num Lock key, the numeric keypad always produces only numbers. The num lock key is replaced by the clear key.



Numeric keypads usually operate in two modes: when Num Lock is off, keys 8, 6, 2, 4 act like an arrow keys and 7, 9, 3, 1 act like Home, PgUp, PgDn and End; when Num Lock is on, digits keys produce corresponding digits. These, however, differ from the numeric keys at the top of the keyboard in that, when combined with the Alt key on a PC, they are used to enter characters which may not be otherwise available: for example, Alt-0169 produces the copyright symbol. These are referred to as Alt codes.

5.8.EXPERIMENTAL SETUP

5.8.1.BUS DETECTION DEVICE



5.8.2.KEYPAD FOR BLIND



CHAPTER 6

PROCESS INVOLVED AT THE COMPUTER TERMINAL

6.1. VB PROGRAM TO CALCULATE THE DISTANCE & TIME

```

Dim var1, n, i As String
Dim gh, F, k1, temp As Integer
Dim a, b, c As Integer
Dim queryAddress As String
Dim voice As SpVoice
Const pi = 3.14152653589793
Private Sub Command1_Click()
On Error Resume Next
With MSComm1
.CommPort = Val(Text4.Text)
.Handshaking = comNone
.RThreshold = 1
.SThreshold = 8
.Settings = "9600,n,8,1"
.PortOpen = True
Mobile_Init
Timer2.Enabled = True
Timer3.Enabled = True
End With
End Sub

```

```
Private Sub Command2_Click()
```

```
End
```

```
End Sub
```

```
Private Sub Command3_Click()
```

```
Form2.Show
```

```
End Sub
```

```
Private Sub Command5_Click()
```

```
On Error Resume Next
```

```
With MSComm2
```

```
    .CommPort = Val(Text7.Text)
```

```
    .Handshaking = comNone
```

```
    .RThreshold = 1
```

```
    .SThreshold = 8
```

```
    .Settings = "9600,n,8,1"
```

```
    .PortOpen = True
```

```
End With
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
Set voice = New SpVoice
```

```
voice.Rate = 0
```

```
voice.Speak "BUS DETECTION TECHNIC FOR BLIND", SVSFlagsAsync
```

```
End Sub
```

```
Select Case comEvReceive
```

```
Case comEvReceive
```

```
    databuf = MSComm2.Input
```

```
    Text8.Text = databuf
```

```
    voice.Speak Mid(databuf, 1, 1), SVSFlagsAsync
```

```
End Select
```

```
End Sub
```

```
Private Sub Timer2_Timer()
```

```
MSComm1.Output = "AT+CMGR=1" & vbCrLf
```

```
Sleep 1000
```

```
'MSComm1.Output = "AT+CMGD=1" & vbCrLf
```

```
'Sleep 1000
```

```
End Sub
```

```
Private Sub FUN()
```

```
For k1 = 1 To 100
```

```
    If Mid(Text3.Text, k1, 1) = "*" Then
```

```
        Text11.Text = Mid(Text3.Text, k1 + 1, 2) + "." + Mid(Text3.Text, k1 + 3, 2) +  
Mid(Text3.Text, k1 + 7, 3)
```

```
        Text12.Text = Mid(Text3.Text, k1 + 12, 2) + "." + Mid(Text3.Text, k1 + 14, 2) +  
Mid(Text3.Text, k1 + 17, 3)
```

```
        Text1.Text = Mid(Text3.Text, k1 + 1, 2) + "" + Mid(Text3.Text, k1 + 3, 7) + ""N" +  
" "
```

```
Private Sub MSComm1_OnComm()
```

```
Dim databuf As String
```

```
Dim lat As String
```

```
Dim lon As String
```

```
Select Case comEvReceive
```

```
Case comEvReceive
```

```
    databuf = MSComm1.Input
```

```
    Text3.Text = Text3.Text + databuf
```

```
    If Len(Text3.Text) > 90 Then
```

```
        FUN
```

```
    End If
```

```
End Select
```

```
End Sub
```

```
Private Sub Mobile_Init()
```

```
MSComm1.Output = "AT" & vbCrLf
```

```
Sleep 500
```

```
MSComm1.Output = "AT+CMGF=1" & vbCrLf
```

```
Sleep 500
```

```
End Sub
```

```
Private Sub MSComm2_OnComm()
```

```
Dim databuf As String
```

```
    'queryAddress = "http://maps.google.com/maps?q="
```

```
    'lat = Text1.Text
```

```
    'queryAddress = queryAddress & lat
```

```
    'lon = Text2.Text
```

```
    'queryAddress = queryAddress & lon
```

```
    'WebBrowser1.Navigate queryAddress
```

```
    Text3.Text = ""
```

```
End If
```

```
Next k1
```

```
Text3.Text = ""
```

```
End Sub
```

```
Function distance(lat1, lon1, lat2, lon2)
```

```
Dim theta, dist
```

```
theta = lon1 - lon2
```

```
dist = Sin(deg2rad(lat1)) * Sin(deg2rad(lat2)) + Cos(deg2rad(lat1)) * Cos(deg2rad(lat2)) *  
Cos(deg2rad(theta))
```

```
dist = acos(dist)
```

```
dist = rad2deg(dist)
```

```
distance = dist * 60 * 1.1515 * 1.609344
```

```
End Function
```

```
Function deg2rad(deg)
```

```
deg2rad = CDbI(deg * pi / 180)
```

```
End Function
```

```
Function acos(rad)
```

```
acos = pi / 2 - Atn(rad / Sqr(1 - rad * rad))
```

```
Elseif rad = -1 Then
```

```
acos = pi
```

```
End If
```

```
End Function
```

```
Function rad2deg(rad)
```

```
rad2deg = Cdbl(rad * 180 / pi)
```

```
End Function
```

```
Private Sub Timer3_Timer()
```

```
If (Len(Text9.Text) > 6 And Len(Text10.Text) > 6) Then
```

```
Text14.Text = (distance(Text11.Text, Text12.Text, Text9.Text, Text10.Text))
```

```
Text15.Text = Val(Text14.Text) / 30
```

```
voice.Speak "distance", SVSFlagsAsync
```

```
voice.Speak Mid(Text14.Text, 1, 4), SVSFlagsAsync
```

```
voice.Speak Mid(Text15.Text, 1, 4), SVSFlagsAsync
```

```
voice.Speak "MINUTES", SVSFlagsAsync
```

```
End If
```

```
End Sub
```

6.3.EXPLANATION

The VB coding does the following

- It identifies the Braille input.
- It identifies the location of the corresponding bus and locates them on the Google map.
- With the known location of the bus stand the distance is calculated using the formula.
- Then the distance divided by speed gives the time.

CHAPTER 7

CONCLUSION AND SCOPE

7.1.CONCLUSION

In this project a bus detection device that can calculate the approximate time a bus takes to reach the destination from its current location has been designed based on the GPS and GSM technology. The other objective of designing a special keypad for the visually disabled has also been accomplished and for their aid the arrival time of the bus is given in the form of voice output.

7.2.FUTURE SCOPE

Visually disabled person should be guided in order to give the input. Thus a system that can guide those persons to not only be able to reach the system but also board the bus independently can be designed. The arrival time is however affected by the traffic in the city. Hence formulae for those limitation removal can be included in future design.

Keypad designed for the visually disabled can also be used in the ATM centres if further improvised in the future.

- The results are given in the form of voice output.

6.2.OUTPUT SCREEN

BUS DETECTION TECHNIC FOR BLIND

The screenshot shows a graphical user interface for a bus detection application. It features several input fields and buttons. At the top, there is a 'VALUE' section with a 'CHARACTER' input field. Below this, the interface is split into two columns: 'PC COMMUNICATION' and 'MOBILE'. Each column has 'COM PORT:' and 'BAUD RATE:' input fields, with 'EXT' and 'SET' buttons below them. The 'SET VALUE' section includes 'Latitude' and 'Longitude' input fields. The 'LOCATION' section also includes 'Latitude' and 'Longitude' input fields. At the bottom, there are 'DISTANCE' and 'HOUR' output fields, and a 'GOOGLE MAP' button.

The communication port numbers to which the keypad and the mobile are connected should be given as input. Thus the distance and time are displayed and when the user clicks the GOOGLE MAP button the exact current location of the bus is displayed.

REFERENCES

1. Patrick E. Lanigan, Aaron M. Paulos, Andrew W. Williams, Priyam Narasimhan, "Trinetra: Assistive Technologies for the Blind", Carnegie Mellon University Pittsburgh, PA 15213, May 1, 2006.
2. Bus detection device for the blind using RFID application Noor, M.Z.H. Ismail, I. Saaid, M.F. Fac. of Electr. Eng., Univ. Teknol. MARA, Shah Alam
3. MyBus: helping bus riders make informed decisions Maclean, S.D.; Dailey, D.J.; Washington Univ., Seattle, WA, USA
4. www.omniglot.com
5. www.wikipedia.com
6. www.vbtutor.net

APPENDIX A

PIC 16F877 CODING

Code for Braille :

```
#include<at89x51.h>
#include"smcl_lcd8.h"
#include"AT_serial.h"
```

```
sbit key1=P1^0;
sbit key2=P1^1;
sbit key3=P1^2;
sbit key4=P1^3;
sbit key5=P1^4;
sbit key6=P1^5;
sbit ent=P1^7;
```

```
void display(unsigned char);
```

```
unsigned char a=0,b=0,c=0,d=0,e=0,f=0,sum;
```

```
void main()
{
    Lcd8_Init();
    Serial_Init(9600);
    Lcd8_Display(0x80,"Bus Detection ",16);
    Lcd8_Display(0xC0,"System For Blind",16);
    Delay(40000);Delay(40000);Delay(40000);
    Delay(40000);Delay(40000);Delay(40000);
    Lcd8_Command(0x01);
    Lcd8_Display(0x80,"Charactor:",10);
    while(1)
    {
        if(!key1)a=1;
```

```
        Serial_Out('b');
        break;
    }
    case 3:
    {
        Lcd8_Write(0x8A,'c');
        Serial_Out('c');
        break;
    }
    case 11:
    {
        Lcd8_Write(0x8A,'d');
        Serial_Out('d');
        break;
    }
    case 9:
    {
        Lcd8_Write(0x8A,'e');
        Serial_Out('e');
        break;
    }
    case 7:
    {
        Lcd8_Write(0x8A,'f');
        Serial_Out('f');
        break;
    }
    case 15:
    {
        Lcd8_Write(0x8A,'g');
        Serial_Out('g');
        break;
```

```
        if(!key2)b=2;
        //else b=0;
        if(!key3)c=4;
        //else c=0;
        if(!key4)d=8;
        //else d=0;
        if(!key5)e=16;
        //else e=0;
        if(!key6)f=32;
        //else f=0;
```

```
        if(!ent)
        {
            sum=a+b+c+d+e+f;
            a=b=c=d=e=f=0;
            display(sum);
        }
    }
}
void display(unsigned char num)
{
    switch(num)
    {
        case 1:
        {
            Lcd8_Write(0x8A,'a');
            Serial_Out('a');
            break;
        }
        case 5:
        {
            Lcd8_Write(0x8A,'e');
```

```
        case 13:
        {
            Lcd8_Write(0x8A,'h');
            Serial_Out('h');
            break;
        }
        case 6:
        {
            Lcd8_Write(0x8A,'i');
            Serial_Out('i');
            break;
        }
        case 14:
        {
            Lcd8_Write(0x8A,'j');
            Serial_Out('j');
            break;
        }
        case 17:
        {
            Lcd8_Write(0x8A,'k');
            Serial_Out('k');
            break;
        }
        case 21:
        {
            Lcd8_Write(0x8A,'l');
            Serial_Out('l');
            break;
        }
        case 19:
        {
            Lcd8_Write(0x8A,'m');
```

```

Serial_Out('m');
break;
}
case 27:
{
Lcd8_Write(0x8A,'n');
Serial_Out('n');
break;
}
case 25:
{
Lcd8_Write(0x8A,'o');
Serial_Out('o');
break;
}
case 23:
{
Lcd8_Write(0x8A,'p');
Serial_Out('p');
break;
}
case 31:
{
Lcd8_Write(0x8A,'q');
Serial_Out('q');
break;
}
case 29:
{
Lcd8_Write(0x8A,'r');
Serial_Out('r');
break;

```

```

case 22:
{
Lcd8_Write(0x8A,'s');
Serial_Out('s');
break;
}
case 30:
{
Lcd8_Write(0x8A,'t');
Serial_Out('t');
break;
}
case 49:
{
Lcd8_Write(0x8A,'u');
Serial_Out('u');
break;
}
case 53:
{
Lcd8_Write(0x8A,'v');
Serial_Out('v');
break;
}
case 46:
{
Lcd8_Write(0x8A,'w');
Serial_Out('w');
break;
}
case 51:
{

```

```

Serial_Out('x');
break;
}
case 59:
{
Lcd8_Write(0x8A,'y');
Serial_Out('y');
break;
}
case 57:
{
Lcd8_Write(0x8A,'z');
Serial_Out('z');
break;
}
case 56:
{
Lcd8_Write(0x8A,'a');
Serial_Out('0');
break;
}
case 4:
{
Lcd8_Write(0x8A,'a');
Serial_Out('1');
break;
}
case 20:
{
Lcd8_Write(0x8A,'a');
Serial_Out('2');
break;

```

```

case 12:
{
Lcd8_Write(0x8A,'a');
Serial_Out('3');
break;
}
case 44:
{
Lcd8_Write(0x8A,'a');
Serial_Out('4');
break;
}
case 36:
{
Lcd8_Write(0x8A,'a');
Serial_Out('5');
break;
}
case 28:
{
Lcd8_Write(0x8A,'a');
Serial_Out('6');
break;
}
case 60:
{
Lcd8_Write(0x8A,'a');
Serial_Out('7');
break;
}
case 52:
{
Lcd8_Write(0x8A,'a');

```

```

        Serial_Out('8');
        break;
    }
case 24:
    {
        Lcd8_Write(0x8A,'a');
        Serial_Out('9');
        break;
    }
}
}

```

Code for ADC :

```

void Adc_Init()
{
    ADMUX=(1<<REFS0);           // For Aref=AVcc;
    ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); //Rrescalar div factor
    =128
}

```

```

uint16_t Adc_Cha10(uint8_t ch)
{
    if(ch==0)
    {
        //Select ADC Channel ch must be 0-7
        ch=0b00000000;
        ADMUX=ch;

        //Start Single conversion
        ADCSRA|=(1<<ADSC);
    }
}

```

```

//Select ADC Channel ch must be 0-7
ch=0b00000010;
ADMUX=ch;

```

```

//Start Single conversion
ADCSRA|=(1<<ADSC);

```

```

//Wait for conversion to complete
while(!(ADCSRA & (1<<ADIF)));

```

```

//Clear ADIF by writing one to it
//Note you may be wondering why we have write one to clear it
//This is standard way of clearing bits in io as said in datasheets.
//The code writes '1' but it result in setting bit to '0' !!!

```

```

ADCSRA|=(1<<ADIF);

```

}

```

else if(ch==3)

```

```

{
    //Select ADC Channel ch must be 0-7
    ch=0b00000011;
    ADMUX=ch;

```

```

//Start Single conversion
ADCSRA|=(1<<ADSC);

```

```

//Wait for conversion to complete
while(!(ADCSRA & (1<<ADIF)));

```

```

//Clear ADIF by writing one to it

```

```

while(!(ADCSRA & (1<<ADIF)));

```

```

//Clear ADIF by writing one to it
//Note you may be wondering why we have write one to clear it
//This is standard way of clearing bits in io as said in datasheets.
//The code writes '1' but it result in setting bit to '0' !!!

```

```

ADCSRA|=(1<<ADIF);
}

```

```

else if(ch==0)

```

```

{
    //Select ADC Channel ch must be 0-7
    ch=0b00000001;
    ADMUX=ch;

```

```

//Start Single conversion
ADCSRA|=(1<<ADSC);

```

```

//Wait for conversion to complete
while(!(ADCSRA & (1<<ADIF)));

```

```

//Clear ADIF by writing one to it
//Note you may be wondering why we have write one to clear it
//This is standard way of clearing bits in io as said in datasheets.
//The code writes '1' but it result in setting bit to '0' !!!

```

```

ADCSRA|=(1<<ADIF);

```

}

```

else if(ch==2)

```

```

//Note you may be wondering why we have write one to clear it
//This is standard way of clearing bits in io as said in datasheets.
//The code writes '1' but it result in setting bit to '0' !!!

```

```

ADCSRA|=(1<<ADIF);

```

}

```

else if(ch==4)

```

```

{
    //Select ADC Channel ch must be 0-7
    ch=0b00000100;
    ADMUX=ch;

```

```

//Start Single conversion
ADCSRA|=(1<<ADSC);

```

```

//Wait for conversion to complete
while(!(ADCSRA & (1<<ADIF)));

```

```

//Clear ADIF by writing one to it
//Note you may be wondering why we have write one to clear it
//This is standard way of clearing bits in io as said in datasheets.
//The code writes '1' but it result in setting bit to '0' !!!

```

```

ADCSRA|=(1<<ADIF);

```

}

```

else if(ch==5)

```

```

{
    //Select ADC Channel ch must be 0-7

```

```

ch=0b00000101;
ADMUX=ch;

//Start Single conversion
ADCSRA|=(1<<ADSC);

//Wait for conversion to complete
while(!(ADCSRA & (1<<ADIF)));

//Clear ADIF by writing one to it
//Note you may be wondering why we have write one to clear it
//This is standard way of clearing bits in io as said in datasheets.
//The code writes '1' but it result in setting bit to '0' !!!

ADCSRA|=(1<<ADIF);

}

else if(ch==6)
{
//Select ADC Channel ch must be 0-7
ch=0b00000110;
ADMUX=ch;

//Start Single conversion
ADCSRA|=(1<<ADSC);

//Wait for conversion to complete
while(!(ADCSRA & (1<<ADIF)));

//Clear ADIF by writing one to it
//Note you may be wondering why we have write one to clear it
//This is standard way of clearing bits in io as said in datasheets.
//The code writes '1' but it result in setting bit to '0' !!!

```

```

#define SDA_L PORTC &= ~0x02
#define SCL_H PORTC |= 0x01
#define SCL_L PORTC &= ~0x01

void Eeprom_Write(unsigned char,unsigned char);
Eeprom_Read(unsigned char);
void Eeprom_rd_wr_sub();
void Eeprom_Init();
void Eeprom_Start();
void Eeprom_Tx();
void Eeprom_Rx();
void Eeprom_Stop();
void Eeprom_Ack();

unsigned int eeprom_add_wr,eeprom_add_rd;
unsigned int d_eeprom,datain_eeprom,in_eeprom,temp_eeprom,dat_eeprom;
unsigned int flag_eeprom,c;

void Eeprom_Init()
{
    eeprom_add_wr=0xa0;
    eeprom_add_rd=0xa1;
}

void Eeprom_Write(unsigned char zig,unsigned char zag)// program to write to EEPROM
{
    dat_eeprom=zig;
    temp_eeprom=zag;
    Eeprom_rd_wr_sub();
above:

```

```

//This is standard way of clearing bits in io as said in datasheets.
//The code writes '1' but it result in setting bit to '0' !!!

ADCSRA|=(1<<ADIF);

}

else if(ch==7)
{
//Select ADC Channel ch must be 0-7
ch=0b00000111;
ADMUX=ch;

//Start Single conversion
ADCSRA|=(1<<ADSC);

//Wait for conversion to complete
while(!(ADCSRA & (1<<ADIF)));

//Clear ADIF by writing one to it
//Note you may be wondering why we have write one to clear it
//This is standard way of clearing bits in io as said in datasheets.
//The code writes '1' but it result in setting bit to '0' !!!

ADCSRA|=(1<<ADIF);

}

return(ADC);

}
Code for EEPROM :
//Eeprom Add Wr PortC |= 0x02?

```

```

Eeprom_Tx();
if(c==1)goto above;
c=0;
Eeprom_Stop();
}

Eeprom_Read(unsigned char zig)
{
    dat_eeprom=zig;
    Eeprom_rd_wr_sub();
    Eeprom_Start();
be:
    d_eeprom=eeprom_add_rd;
    Eeprom_Tx();
    if(c==1)goto be;
    Eeprom_Rx();
    Eeprom_Ack();
    c=0;
    Eeprom_Stop();
    return(datain_eeprom);
}

void Eeprom_Start()
{
    SDA_H;
    SCL_H;
    SDA_L;
    SCL_L;
}

```

```
void Eeprom_Stop()
```

```
{  
    SDA_L;  
    SCL_H;  
    SDA_H;  
}
```



P-3423

```
void Eeprom_Tx()// program to send the device address, read/write address,data to be written
```

```
{  
    signed char i_eeeprom;  
    for(i_eeeprom=7;i_eeeprom>=0;i_eeeprom--)// should necessarily be initialised as signed  
    char.  
    {  
        c=(d_eeeprom>>i_eeeprom)&0x01;  
        //sda_eeeprom=c;  
        //(PINC&0x02)=c;  
        if(c)SDA_H;  
        else SDA_L;  
        SCL_H;// clock is essential inorder to write or read  
        SCL_L;// clk should be alternated  
    }  
    SDA_H;  
    SCL_H;  
    c=PINC&0x02;  
    SCL_L;  
}
```

```
void Eeprom_Rx()// program read the data from the EEPROM
```

```
{
```

```
    Eeprom_Tx();  
    if(c==1)goto here1;
```

```
again1:  
    d_eeeprom=dat_eeeprom;// the address from which data is to be read/written is to be  
    passed  
    Eeprom_Tx();  
    if(c==1)goto again1;  
}
```

```
void Eeprom_cardcheck()// to check whether the card has been entered
```

```
{  
    unsigned char e_eeeprom;  
    Eeprom_Start();  
    for (e_eeeprom=0;e_eeeprom<5;e_eeeprom++)  
    {  
        d_eeeprom=0xa0;// to send the device address  
        Eeprom_Tx();  
        if(c==1)flag_eeeprom=0;  
        else  
        {  
            flag_eeeprom=1;  
            goto breac;  
        }  
    }  
    breac;  
}
```

Code for serial communication :

```
SDA_H;  
DDRC=0XFD;  
for (l_eeeprom=0;l_eeeprom<=7;l_eeeprom++)  
{  
    SCL_H;  
    in_eeeprom=in_eeeprom<<1;  
    in_eeeprom|=PINC&0x02;  
    SCL_L;  
}
```

```
in_eeeprom=in_eeeprom>>1;  
datain_eeeprom=in_eeeprom;  
in_eeeprom=0;  
DDRC=0xFF;  
PORTC=0;  
}
```

```
void Eeprom_Ack()// this is to intimate the EEPROM that the read operation is over
```

```
{  
    SDA_H;  
    SCL_H;  
    SCL_L;  
}
```

```
void Eeprom_rd_wr_sub()// this routine will be used by both the read & write operations to  
send the device address & the address at which the corresponding action is to be taken
```

```
{  
    Eeprom_Start();
```

```
    here1:
```

```
    //Eeprom_Start(); // device address is passed
```

```
void Serial0_Init(unsigned int);  
void Serial1_Init(unsigned int);  
void Serial0_Out(unsigned char);  
void Serial1_Out(unsigned char);  
void Serial0_Conout(const unsigned char *,unsigned char);  
void Serial1_Conout(const unsigned char *,unsigned char);
```

```
void Serial0_Init(unsigned int BAUD)
```

```
{  
    /* Set baud rate */  
    UBRR0H = (unsigned char) (((F_CPU/(BAUD*16UL))-1)>>8);  
    UBRR0L = (unsigned char) (F_CPU/(BAUD*16UL))-1;  
    //UBRR0H = (unsigned char) (F_CPU/(BAUD*16UL))-1;  
(UART_BAUD_CALC(UART_BAUD_RATE,F_OSC)>>8);  
    //UBRR0L = (unsigned char) UART_BAUD_CALC(UART_BAUD_RATE,F_OSC);  
    //TX, RX enable and RX Complete interrupt enable  
    UCSR0B = (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0);  
    /* Set frame format: 8data, 2stop bit */  
    //UCSR0C = (1<<URSEL0)|(1<<USBS0)|(3<<UCSZ00);  
    UCSR0C = (3 << UCSZ00);  
}
```

```
void Serial1_Init(unsigned int BAUD)
```

```
{  
    UBRR1H = (unsigned char) (((F_CPU/(BAUD*16UL))-1)>>8);  
    UBRR1L = (unsigned char) (F_CPU/(BAUD*16UL))-1;  
    UCSR1B = (1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1);  
    UCSR1C = (3 << UCSZ10);  
}
```

```
void Serial0_Out(unsigned char data0)
```

```
{  
    //UCSR0A = (1<<UDR00); //Wait for empty transmit buffer
```



```

    UDR0 = data0; //Start transimtion
}
void Serial1_Out(unsigned char data1)
{
    while ( !(UCSR1A & (1<<UDRE1)) ); //Wait for empty transmit buffer
    UDR1 = data1; //Start transimtion
}

void Serial0_Conout(const unsigned char *dat,unsigned char n)
{
    unsigned char ser_j;
    for(ser_j=0;ser_j<n;ser_j++)
    {
        Serial0_Out(dat[ser_j]);
    }
}

void Serial1_Conout(const unsigned char *dat,unsigned char n)
{
    unsigned char ser_j;
    for(ser_j=0;ser_j<n;ser_j++)
    {
        Serial1_Out(dat[ser_j]);
    }
}

```

Code for LCD

```

#define First_Line 0x80
#define Second_Line 0xc0
#define Curser_On 0x0f
#define Curser_Off 0x0c
#define Clear_Display 0x01
#define Data_Port P0

```

```

Lcd8_Command(com);

Data_Port=lr; // Data
Lcd_en=Lcd_rs=1;
Lcd_rw=0;
Delay(125);
Lcd_en=0;
Delay(125);
}

void Lcd8_Display(unsigned char com,const unsigned char *word,unsigned int n)
{
    unsigned char Lcd_i;

    for(Lcd_i=0;Lcd_i<n;Lcd_i++)
    {
        Lcd8_Write(com+Lcd_i,word[Lcd_i]);
    }
}

void Lcd8_Decimal2(unsigned char com,unsigned char val)
{
    unsigned int Lcd_hr,Lcd_t,Lcd_o;

    Lcd_hr=val%100;
    Lcd_t=Lcd_hr/10;
    Lcd_o=Lcd_hr%10;

    Lcd8_Write(com,Lcd_t+0x30);
    Lcd8_Write(com+1,Lcd_o+0x30);
}

```

```

sbit Lcd_rs = P2^7;
sbit Lcd_rw = P2^6;
sbit Lcd_en = P2^5;

void Lcd8_Init();
void Lcd8_Command(unsigned char);
void Lcd8_Write(unsigned char,unsigned char);
void Lcd8_Display(unsigned char,const unsigned char*,unsigned int);
void Lcd8_Decimal2(unsigned char,unsigned char);
void Lcd8_Decimal3(unsigned char,unsigned char);
void Lcd8_Decimal4(unsigned char,unsigned int);
void Delay(unsigned int);

```

```

void Lcd8_Init()
{
    Lcd8_Command(0x38); //to select function set
    Lcd8_Command(0x06); //entry mode set
    Lcd8_Command(0x0c); //display on
    Lcd8_Command(0x01); //clear display
}

```

```

void Lcd8_Command(unsigned char com)
{
    Data_Port=com;
    Lcd_en=1;
    Lcd_rs=Lcd_rw=0;
    Delay(125);
    Lcd_en=0;
    Delay(125);
}

```

```

void Lcd8_Write(unsigned char com,unsigned char lr)
{

```

```

void Lcd8_Decimal3(unsigned char com,unsigned char val)
{
    unsigned int Lcd_h,Lcd_hr,Lcd_t,Lcd_o;

    Lcd_h=val/100;
    Lcd_hr=val%100;
    Lcd_t=Lcd_hr/10;
    Lcd_o=Lcd_hr%10;

    Lcd8_Write(com,Lcd_h+0x30);
    Lcd8_Write(com+1,Lcd_t+0x30);
    Lcd8_Write(com+2,Lcd_o+0x30);
}

```

```

void Lcd8_Decimal4(unsigned char com,unsigned int val)
{
    unsigned int Lcd_th,Lcd_thr,Lcd_h,Lcd_hr,Lcd_t,Lcd_o;

    val = val%10000;
    Lcd_th=val/1000;
    Lcd_thr=val%1000;
    Lcd_h=Lcd_thr/100;
    Lcd_hr=Lcd_thr%100;
    Lcd_t=Lcd_hr/10;
    Lcd_o=Lcd_hr%10;

    Lcd8_Write(com,Lcd_th+0x30);
    Lcd8_Write(com+1,Lcd_h+0x30);
    Lcd8_Write(com+2,Lcd_t+0x30);
    Lcd8_Write(com+3,Lcd_o+0x30);
}

```

```

void Delay(unsigned int del)

```

```

while(del-);
}
////////////////////////////////////////////////////AT-MEGA164--TWO SERIALS//////////////////////////////////////

#define F_CPU 8000000UL

void Serial0_Init(unsigned int);
void Serial1_Init(unsigned int);
void Serial0_Out(unsigned char);
void Serial1_Out(unsigned char);
void Serial0_Conout(const unsigned char *,unsigned char);
void Serial1_Conout(const unsigned char *,unsigned char);

void Serial0_Init(unsigned int BAUD)
{
    /* Set baud rate */
    UBRR0H = (unsigned char) (((F_CPU/(BAUD*16UL))-1)>>8);
    UBRR0L = (unsigned char) (F_CPU/(BAUD*16UL))-1;
    //UBRR0H = (unsigned char) (F_CPU/(BAUD*16UL))-1;
    //UBRR0L = (unsigned char) (F_CPU/(BAUD*16UL))-1;
    (UART_BAUD_CALC(UART_BAUD_RATE,F_OSC)>>8);
    //UBRR0L = (unsigned char) UART_BAUD_CALC(UART_BAUD_RATE,F_OSC);
    //TX, RX enable and RX Complete interrupt enable
    UCSR0B = (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0);
    /* Set frame format: 8data, 2stop bit */
    //UCSR0C = (1<<URSEL0)|(1<<USBS0)|(3<<UCSZ00);
    UCSR0C = (3 << UCSZ00);
}

void Serial1_Init(unsigned int BAUD)
{
    UBRR1H = (unsigned char) (((F_CPU/(BAUD*16UL))-1)>>8);
    UBRR1L = (unsigned char) (F_CPU/(BAUD*16UL))-1;
    //UBRR1H = (unsigned char) (F_CPU/(BAUD*16UL))-1;
    //UBRR1L = (unsigned char) (F_CPU/(BAUD*16UL))-1;
}

```

```

UCSR1B = (1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1);
UCSR1C = (3 << UCSZ10);
}

void Serial0_Out(unsigned char data0)
{
    while ( !(UCSR0A & (1<<UDRE0)) ); //Wait for empty transmit buffer
    UDR0 = data0; //Start transmission
}

void Serial1_Out(unsigned char data1)
{
    while ( !(UCSR1A & (1<<UDRE1)) ); //Wait for empty transmit buffer
    UDR1 = data1; //Start transmission
}

void Serial0_Conout(const unsigned char *dat,unsigned char n)
{
    unsigned char ser_j;
    for(ser_j=0;ser_j<n;ser_j++)
    {
        Serial0_Out(dat[ser_j]);
    }
}

void Serial1_Conout(const unsigned char *dat,unsigned char n)
{
    unsigned char ser_j;
    for(ser_j=0;ser_j<n;ser_j++)
    {
        Serial1_Out(dat[ser_j]);
    }
}

```

APPENDIX B

16F877A PIC MICROCONTROLLER

PIN OUT DESCRIPTION

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O-P Type	Buffer Type	Description
OSC1/CLKIN	13	14	30	I	ST/CMOS ⁽¹⁾	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	15	31	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
MCLR/Vpp/THV	1	2	18	I/P	ST	Master clear (reset) input or programming voltage input, or high voltage test mode control. This pin is an active-low reset to the device.
RA0/AN0	2	3	19	I/O	TTL	PORTA is a bi-directional I/O port. RA0 can also be analog input0.
RA1/AN1	3	4	20	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/Vref-	4	5	21	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/Vref+	5	6	22	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.
RA4/T0CKI	6	7	23	I/O	ST	RA4 can also be the clock input to the Timer0 timer counter. Output is open drain type.
RA5/SS/AN4	7	8	24	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
RB0/INT	33	36	8	I/O	TTL/ST ⁽¹⁾	PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs. RB0 can also be the external interrupt pin.
RB1	34	37	9	I/O	TTL	
RB2	35	38	10	I/O	TTL	
RB3/PGM	36	39	11	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	37	41	14	I/O	TTL	Interrupt on change pin.
RB5	38	42	15	I/O	TTL	Interrupt on change pin.
RB6/PGC	39	43	16	I/O	TTL/ST ⁽²⁾	Interrupt on change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	43	44	17	I/O	TTL/ST ⁽²⁾	Interrupt on change pin or In-Circuit Debugger pin. Serial programming data.

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O-P Type	Buffer Type	Description
RC0/T1OSC/T1CKI	16	16	32	I/O	ST	PORTC is a bi-directional I/O port. RC0 can also be the Timer1 oscillator output or a Timer1 clock input.
RC1/T1OS/ICP2	16	18	35	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/ICP1	17	19	36	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	18	20	37	I/O	ST	RC3 can also be the synchronous serial clock input/output for both SPI and I ² C modes.
RC4/SDI/SDA	23	25	42	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I ² C mode).
RC5/SDO	24	26	43	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TXCK	25	27	44	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RXD/T	26	29	1	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
RD0/PSP0	19	21	38	I/O	ST/TTL ⁽¹⁾	PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.
RD1/PSP1	20	22	39	I/O	ST/TTL ⁽¹⁾	
RD2/PSP2	21	23	40	I/O	ST/TTL ⁽¹⁾	
RD3/PSP3	22	24	41	I/O	ST/TTL ⁽¹⁾	
RD4/PSP4	27	30	2	I/O	ST/TTL ⁽¹⁾	
RD5/PSP5	28	31	3	I/O	ST/TTL ⁽¹⁾	
RD6/PSP6	29	32	4	I/O	ST/TTL ⁽¹⁾	
RD7/PSP7	30	33	5	I/O	ST/TTL ⁽¹⁾	
RE0/RD/AN5	6	9	25	I/O	ST/TTL ⁽²⁾	PORTE is a bi-directional I/O port. RE0 can also be read control for the parallel slave port, or analog input5.
RE1/WR/AN6	9	10	26	I/O	ST/TTL ⁽²⁾	RE1 can also be write control for the parallel slave port, or analog input6.
RE2/CS/AN7	10	11	27	I/O	ST/TTL ⁽²⁾	RE2 can also be select control for the parallel slave port, or analog input7.
Vss	12,31	13,34	6,29	P	—	Ground reference for logic and I/O pins.
Vpp	11,32	12,35	7,28	P	—	Positive supply for logic and I/O pins.
NC	—	1,17,28,30	12,13,33,34	—	—	These pins are not internally connected. These pins should be left unconnected.

Legend: I = input O = output I/O = input/output P = power

— = Not used TTL = TTL input ST = Schmitt Trigger input

Note :

1. This buffer is a Schmitt Trigger input when configured as an external interrupt.
2. This buffer is a Schmitt Trigger input when used in serial programming mode.
3. This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
4. This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a

CMOS input otherwise.

I/O PORTS

Some pins for these I/O ports are multiplexed with an alternate function for the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

Additional Information on I/O ports may be found in the IC micro™ Mid-Range Reference Manual,

PORTA AND TRISA REGISTER

PORTA is a 6-bit wide bi-directional port. The corresponding data direction register is TRISA. Setting a TRISA bit (=1) will make the corresponding PORTA pin an input, i.e., put the corresponding output driver in a Hi-impedance mode. Clearing a TRISA bit (=0) will make the corresponding PORTA pin an output, i.e., put the contents of the output latch on the selected pin.

PORTB AND TRISB REGISTER

PORTB is an 8-bit wide bi-directional port. The corresponding data direction register is TRISB. Setting a TRISB bit (=1) will make the corresponding PORTB pin an input, i.e., put the corresponding output driver in a hi-impedance mode. Clearing a TRISB bit (=0) will make the corresponding PORTB pin an output, i.e., put the contents of the output latch on the selected pin. Three pins of PORTB are multiplexed with the Low Voltage Programming function; RB3/PGM, RB6/PGC and RB7/PGD. The alternate functions of these pins are described in the Special Features Section. Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups.

PORTC AND TRISC REGISTER

PORTC is an 8-bit wide bi-directional port. The corresponding data direction register is TRISC. Setting a TRISC bit (=1) will make the corresponding PORTC pin an input, i.e., put the corresponding output driver in a hi-impedance mode

This section is not applicable to the 28-pin devices. PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output. PORTD can be configured as an 8-bit wide microprocessor Port (parallel slave port) by setting control bit PSPMODE (TRISE<4>). In this mode, the input buffers are TTL.

PORTE AND TRISE REGISTER

PORTE has three pins RE0/RD/AN5, RE1/WR/AN6 and RE2/CS/AN7, which are individually configurable as inputs or outputs. These pins have Schmitt Trigger input buffers.

MEMORY ORGANISATION

There are three memory blocks in each of the PIC16F877 MUC's. The program memory and Data Memory have separate buses so that concurrent access can occur.

PROGRAM MEMORY ORGANISATION

The PIC16F877 devices have a 13-bit program counter capable of addressing 8K *14 words of FLASH program memory. Accessing a location above the physically implemented address will cause a wraparound.

The RESET vector is at 0000h and the interrupt vector is at 0004h.

DATA MEMORY ORGANISATION

The data memory is partitioned into multiple banks which contain the General Purpose Registers and the special functions Registers. Bits RP1 (STATUS<6>) and RP0 (STATUS<5>) are the bank selected bits.

RP1:RP0	Banks
00	0
01	1
10	2
11	3

Each bank extends up to 7Fh (1238 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static RAM. All implemented banks contain special function registers. Some frequently used special function registers from one bank may be mirrored in another bank for code reduction and quicker access.

EEPROM

EEPROM (electrically erasable, programmable read only memory) technology supplies Nonvolatile storage of variables to a PIC-controlled device or instrument. That is variables stored in an EEPROM will remain there even after power has been turned off and then on again. Some instruments use an EEPROM to store calibration data during manufacture. In this way, each instrument is actually custom built, with customization that can be easily automated. Other instruments use an EEPROM to allow a user to store several sets of setup information. For an instrument requiring a complicated setup procedure, this permits a user to retrieve the setup required for any one of several very different measurements. Still other devices use an EEPROM in a way that is transparent

To a user, providing backup of setup parameters and thereby bridging over power outages

The data EEPROM and flash program memory are readable and writable during normal operation over the entire VDD range. A bulk erase operation may not be issued from user code (which includes removing code protection. The data memory is not directly mapped in the register file space. Instead it is indirectly addressed through the special function registers (SFR).

There are six SFRS used to read and write the program and data EEPROM memory.

These registers are:

EECON1

EECON2

EEDATA

EEDATH

EEADR

EEADRH

EEDATA holds the 8-bit data for read/write and EEADRH holds the address of the EEPROM location being accessed. The 8-bit EEADR register can access up to 256 locations of data EEPROM.

TIMERS

There are three timers used Timer 0, Timer1 and Timer2

Timer 0

8-bit timer/counter

Software programmable prescaler

Internal or external clock select

Readable writable

Interrupt on overflow

Edge selects for external clock

Timer 1

Timer 1 can be used as timer or counter

It is 16-bit register

Software programmable prescaler

Interrupt on overflow

Readable and writable

The timer-1 module is a 16-bit timer/counter consisting two 8-bit register (TMR1H) and TMR1L), which are readable and writable. The TMR1 register pair (TMR1H:TMR1L)

increments from 0000h to FFFFh and rolls over to 0000h. The tmr1 interrupt, if enabled, is generated on overflow, which is latched in interrupt flag bit tmr1IF. This interrupt can be enabled/disabled by setting/clearing tmr1 interrupt enable bit tmr1IE.

Timer-2

Timer2 is an 8-bit timer with a prescaler and a postscaler. IT can be used as the PWM

Time-base for the PWM mode of the CCP module(s). The TMR2 register is readable and writable, and is cleared on any device reset.

The input clock ($F_{osc}/4$) has a prescale option of 1:1, 1:4 OR 1:16, selected by control bits.

The timer2 module has an 8-bit period register PR2. Timer2 increments from 00h until it match PR2 and then resets to 00h on the next increment cycle. PR2 is a readable and writable register. The PR2 register is initialized to Fh upon reset.

The match output of TMR2 goes through a 4-bit postscaler (which gives a 1:1 to 1:16 scaling inclusive) to generate a tmr2 interrupt

Timer 2 can be shut off by clearing control bit tmr2on to minimize power consumption.

The prescaler and postscaler counters are cleared when any of the following occurs:

A write to the tmr2 register

A write to the t2con register

An any-device reset

Tmr2 is not cleared when t2con is written

ANALOG TO DIGITAL CONVERTER (ADC)

There are two types of analog to digital converter is present in this IC. We use 10-bit ADC. The ADC module can have up to eight analog inputs for a device. The analog input charges a sample and hold capacitor. The output of sample and hold capacitor is the input into the converter. The converter then generates a digital result of this analog level via successive approximation. The A/D conversion of the analog input signal results in a

INTERRUPTS

The PIC16F87X FAMILY HAS UPTO 14 SOURCES OF INTERRUPT. The interrupt control register (INTCON) records individual interrupt requests in flag bits. IT also has individual interrupt requests in flag bits. IT also has individual and global interrupt enables bits.

Additionally if the device has peripheral interrupts, then it will have registers to enable the peripheral interrupts and registers to hold the interrupt flag bits

PIE1

PIE2

PIR1

PIR2

The following table shows which devices have which interrupts.

Device	TMR1IF	INTF	RBIF	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	EEIF	BCLIF	CCP2IF
PIC16F876/873	Yes	Yes	Yes	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
PIC16F877/874	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

ADDRESSING MODES:

DIRECT ADDRESSING:

In direct addressing, the operand specified by an 8-bit address field in the instruction. Only internal data RAM and SFR's can be directly addressed.

INDIRECT ADDRESSING:

In Indirect addressing, the instruction specifies a register that contains the address of the operand. Both internal and external RAM can indirectly address.

The address register for 8-bit addresses can be either the Stack Pointer or R0 or R1 of the selected register Bank. The address register for 16-bit addresses can be only the 16-bit data pointer register, DPTR.

INDEXED ADDRESSING

corresponding 10-bit digital number. The A/D module has high and low voltage reference input that is software selectable to some combination of VDD, VSS, and RA2 or RA3.

The A/D module has four registers. These registers are

A/D result high register (ADRESH)

A/D RESULT LOW REGISTER (ADRESL)

A/D CONTROL REGISTER 0 (ADCON0)

A/D CONTROL REGISTER 1 (ADCON1)

Program memory can only be accessed via indexed addressing this addressing mode is intended for reading look-up tables in program memory.

REGISTER INSTRUCTION

The register banks, which contains registers R0 through R7, can be accessed by instructions whose opcodes carry a 3-bit register specification. Instructions that access the registers this way make efficient use of code, since this mode eliminates an address byte. When the instruction is executed, one of four banks is selected at execution time by the row bank select bits in PSW.

IMMEDIATE CONSTANTS

The value of a constant can follow the opcode in program memory For example. MOV A, #100 loads the Accumulator with the decimal number 100. The same number could be specified in hex digit as 64h.

OSCILLATOR AND CLOCK CIRCUIT

XTAL1 and XTAL2 are the input and output respectively of an inverting amplifier which is intended for use as a crystal oscillator in the pierce configuration, in the frequency range of 1.2 Mhz to 12 Mhz. XTAL2 also the input to the internal clock generator.

To drive the chip with an internal oscillator, one would ground XTAL1 and XTAL2. Since the input to the clock generator is divide by two flip flop there are no requirements on the duty cycle of the external oscillator signal. However, minimum high and low times must be observed.

The clock generator divides the oscillator frequency by 2 and provides a tow phase clock signal to the chip. The phase 1 signal is active during the first half to each clock period and the phase 2 signals are active during the second half of each clock period.

CPU TIMING

A machine cycle consists of 6 states. Each state is divided into a phase / half, during which the phase 1 clock is active and phase 2 half. Arithmetic and Logical operations take place during phase1 and internal register - to register transfer take place during phase 2

OPCODE FIELD DESCRIPTIONS F877A INSTRUCTION

SET

Mnemonic, Operands	Description	Cycles	14-Bit Opcode		Status Affected	Notes
			MSb	LSb		
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF f,d	Add W and f	1	00	0111 dfff ffff	C,DC,Z	1,2
ANDWF f,d	AND W with f	1	00	0101 dfff ffff	Z	1,2
CLRF f	Clear f	1	00	0001 1fff ffff	Z	2
CLRWF -	Clear W	1	00	0001 0xxx xxxxxx	Z	1,2
COMF f,d	Complement f	1	00	1001 dfff ffff	Z	1,2
DECf f,d	Decrement f	1	00	0011 dfff ffff	Z	1,2
DECFSZ f,d	Decrement f, Skip if 0	1(2)	00	1011 dfff ffff	Z	1,2,3
INCF f,d	Increment f	1	00	1010 dfff ffff	Z	1,2
INCFSZ f,d	Increment f, Skip if 0	1(2)	00	1111 dfff ffff	Z	1,2,3
IORWF f,d	Inclusive OR W with f	1	00	0100 dfff ffff	Z	1,2
MOVF f,d	Move f	1	00	1000 dfff ffff	Z	1,2
MOVWF f	Move W to f	1	00	0000 1fff ffff		
NOP -	No Operation	1	00	0000 0xxx 0000		
RLF f,d	Rotate Left through Carry	1	00	1101 dfff ffff	C	1,2
RRF f,d	Rotate Right through Carry	1	00	1100 dfff ffff	C	1,2
SUBWF f,d	Subtract W from f	1	00	0010 dfff ffff	C,DC,Z	1,2
SWAPF f,d	Swap nibbles in f	1	00	1110 dfff ffff		1,2
XORWF f,d	Exclusive OR W with f	1	00	0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF f,b	Bit Clear f	1	01	0xxx bfff ffff		1,2
BSF f,b	Bit Set f	1	01	01xx bfff ffff		1,2
BTFSC f,b	Bit Test f, Skip if Clear	1(2)	01	10xx bfff ffff		3
BTFSS f,b	Bit Test f, Skip if Set	1(2)	01	11xx bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW k	Add literal and W	1	11	111x kxxx kxxx	C,DC,Z	
ANDLW k	AND literal with W	1	11	1001 kxxx kxxx	Z	
CALL k	Call subroutine	2	10	0xxx kxxx kxxx		
CLRWDT -	Clear Watchdog Timer	1	00	0000 0110 0100		TOPD
GOTO k	Go to address	2	10	kxxx kxxx kxxx		
IORLW k	Inclusive OR literal with W	1	11	1000 kxxx kxxx	Z	
MOVLW k	Move literal to W	2	00	0000 0000 1001		
RETFIE -	Return from interrupt	2	11	01xx kxxx kxxx		
RETLW k	Return with literal in W	2	00	0000 0000 1000		
RETURN -	Return from Subroutine	1	00	0000 0110 0011		TOPD
SLEEP -	Go into standby mode	1	11	11xx kxxx kxxx	C,DC,Z	
SUBLW k	Subtract W from literal	1	11	1010 kxxx kxxx	Z	
XORLW k	Exclusive OR literal with W	1	11	1010 kxxx kxxx	Z	

- Note 1: When an I/O register is modified as a function of itself (e.g. MOVF PORTB, 1), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.
- Note 2: If the instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the "Timer" Module.
- Note 3: If Program Counter (PC) is specified and a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.