A PROGRAMMABLE CONTROL UNIT FOR INDUSTRIAL APPLICATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUSTAFA KEMAL GÜNGÖR

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2003

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Mübeccel Demirekler
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Mirzahan Hızal
Supervisor

Examining Committee Members

Prof. Dr. Ahmet Rumeli

Prof. Dr. Mirzahan Hızal

Prof. Dr. Arif Ertaş

Prof. Dr. Nevzat Özay

M.Sc. İlhan Koçar

ABSTRACT

A PROGRAMMABLE CONTROL UNIT FOR INDUSTRIAL APPLICATIONS

GÜNGÖR, Mustafa Kemal
M.Sc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Mirzahan Hızal

December 2003, 130 Pages

In this thesis, the automation of the long term and cyclic processes by using a programmable control unit is aimed.

To achieve this goal, timing relays and various microcontrollers are investigated. PIC microcontroller is chosen to implement the control unit due to its advantages like high speed, Harvard and RISC architecture, low cost and flexibility for programming. Theory of the PIC microcontrollers is studied and a controller unit to be used in the mentioned processes is designed. Some features are added to the device to widen the application fields and consequently a multi-purpose programmable controller is realized.

In the device, Microchip PIC16F877 is used as the microcontroller. The code of the controller is written in Assembly Language and is compiled with MPASM. This controller counts the signals coming from internal Timer 555 or external signals and activates ten different outputs in order. The operating times of the outputs can be changed by a keypad and shown in a display.

By keeping the number of the used ports of the microcontroller, as few as possible, room for the future improvements and additions is provided.

Keywords: Microcontroller, automation, PIC

ÖZ

ENDÜSTRİYEL UYGULAMALAR İÇİN PROGRAMLANABİLİR KONTROL ÜNİTESİ

GÜNGÖR, Mustafa Kemal
Yüksek Lisans , Elektrik-Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Mirzahan Hızal

Aralık 2003, 130 sayfa

Bu tezde, uzun süreli ve periyodik işlemlerin, programlanabilir bir kontrol ünitesi kullanılarak otomasyonu amaçlanmıştır.

Bu amaçla, zaman röleleri ve çeşitli mikrodenetleyiciler incelenmiştir. Kontrol ünitesini gerçekleştirmek için yüksek hız, Harvard ve RISC mimarisi, düşük maliyet ve programlanma kolaylığı gibi avantajları nedeniyle PIC mikrodenetleyici seçilmiştir. PIC'in teorisi üzerinde çalışılarak söz konusu işlemlerde kullanılabilecek bir kontrol ünitesi tasarlanmıştır. Cihazın kullanım alanlarını genişletmek için bazı fonksiyonlar eklenmiş ve sonuçta çok amaçlı programlanabilir bir kontrolörün yapımı gerçekleştirilmiştir.

Cihazda, mikrodenetleyici olarak Microchip PIC 16F877 kullanılmıştır. Denetleyicinin programı Assembly dilinde yazılmış ve MPASM yazılımı kullanılarak derlenmiştir. Kontrolör, bünyesinde bulunan Timer 555'in ürettiği veya harici olarak gelen sinyalleri saymakta ve on farklı çıkışı sırasıyla aktive etmektedir. Çıkışların çalışma süreleri klavye yardımıyla değiştirilebilirken göstergede de izlenebilmektedir.

Mikrodenetleyicinin mümkün olduğunca az portu kullanılarak ileride yapılabilecek değişiklik ve eklentilere olanak sağlanmıştır.

Anahtar Kelimeler: Mikrodenetleyici, otomasyon, PIC

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Prof. Dr. Mirzahan Hızal for his guidance and insight throughout this thesis study.

I would like to also thank my wife, Esra, for her great support and understanding.

TABLE OF CONTENTS

ABSTRACT iii

ÖZ iv

ACKNOWLEDGEMENTS v

TABLE OF CONTENTS vi

LIST OF TABLES x

LIST OF FIGURES xi

LIST OF ABBREVIATIONS xiii

CHAPTER

1. INTRODUCTION 1
2. PIC 16F877 MICROCONTROLLER 5

2.1 MICROCONTROLLERS – THE GENERAL 5
2.2 PIC MICROCONTROLLERS 6
2.2.1 General View 6
2.2.2 Architecture Overview 7
2.2.2.1 Harvard Architecture 8
2.2.2.2 Long Word Instructions 10
2.2.2.3 Single Word Instructions 10
2.2.2.4 Instruction Pipeline 10
2.2.2.5 Single Cycle Instructions 10
2.2.2.6 Reduced Instruction Set 11
2.2.2.7 Register File Architecture 11
2.2.2.8 Orthogonal (Symmetric) Instructions 11
2.3 PIC 16F877 ARCHITECTURE 11
2.3.1 Features of the PIC16F877 11
2.3.1.1 Microcontroller Core Features 11
2.3.1.2 Peripheral Features 12
2.3.2 Block Diagram and Pinout Description of PIC16F877 14
2.3.3 Memory Organization 17
2.3.3.1 Program Memory Organization 17
2.3.3.2 Data Memory Organization 18
2.3.3.3 PCL and PCLATH 19
2.3.3.4 Program Memory Paging 20
2.3.3.5 Indirect Addressing, INDF and FSR Registers 20

2.3.4 I/O Ports 21
2.3.4.1 PortA and the TRISA Register 23
2.3.4.2 PortB and the TRISB Register 24
2.3.4.3 PortC and the TRISC Register 25
2.3.4.4 PortD and the TRISD Register 25
2.3.4.5 PortE and the TRISE Register 25
2.3.4.6 Parallel Slave Port 26
2.3.5 Data EEPROM and Flash Program Memory 27
2.3.6 Timer0 Module 29
2.3.6.1 Timer0 Interrupt 29
2.3.6.2 Using Timer0 with an External Clock 30
2.3.6.3 Prescaler 30
2.3.7 Timer1 Module 30
2.3.7.1 Timer1 Operation in Timer Mode 31
2.3.7.2 Timer1 Counter Operation 31
2.3.7.3 Timer1 Operation in Synchronized Counter Mode 32
2.3.7.4 Timer1 Operation in Asynchronous Counter Mode 32
2.3.7.5 Timer1 Oscillator 33
2.3.8 Timer2 Module 33
2.3.8.1 Timer2 Prescaler and Postscaler 33
2.3.8.2 Timer2 Prescaler and Postscaler 34
2.3.9 Capture/Compare/PWM Modules 34
2.3.9.1 Capture Mode 35
2.3.9.2 Compare Mode 35
2.3.9.3 PWM Mode 35
2.3.10 Master Synchronous Serial Port (MSSP) Module 35
2.3.10.1 SPI Mode 36
2.3.10.2 MMSP I^{2}C Operation 37
2.3.10.3 Connection Considerations for I^{2}C Bus 38
2.3.11 Addressable Universal Synchronous Asynchronous Receiver Transmitter (USART) 39
2.3.12 Analog to Digital Converter (A/D) Module 40
2.3.13 Special Features of the CPU 41
2.3.13.1 Configuration Bits 42
2.3.13.2 Oscillator Configurations 42
2.3.13.3 Reset 43
2.3.13.4 Power-On Reset (POR) 44
2.3.13.5 Power-Up Timer (PWRT) 44
2.3.13.6 Oscillator Start-Up Timer (OST) 45
2.3.13.7 Brown-Out Reset (BOR) 45
2.3.13.8 Time-Out Sequence 45
2.3.13.9 Power Control/Status Register (PCON) 45
2.3.13.10 Interrupts 46
2.3.13.11 Context Saving During Interrupts 47
2.3.13.12 Watchdog Timer (WDT) 48
2.3.13.13 Power-Down Mode (SLEEP) 48
2.3.13.14 In-Circuit Debugger 50
2.3.13.15 Program Verification/Code Protection 51

2.3.13.16 ID Locations 51
2.3.13.17 In-Circuit Serial Programming 51
2.3.13.18 Low Voltage ICSP Programming 51

3. PIC PROGRAMMING 53

3.1 INTRODUCTION 53
3.2 MPLAB IDE 54
3.3 MPASM 54
3.3.1 Overview of Assembler 55
3.3.2 Assembler Input/Output Files 57
3.3.2.1 Source Code Format (.ASM) 58
3.3.2.2 Listing File Format (.LST) 59
3.3.2.3 Error File Format (.ERR) 59
3.3.2.4 Hex File Formats (.HEX, .HXL, .HXH) 60
3.3.2.5 Symbol and Debug File Format (.COD) 60
3.3.2.6 Object File Format (.O) 60
3.3.3 MPLAB Projects and MPASM 60
3.3.4 Directive Language 61
3.4 IC-PROG 61
3.4.1 General 61
3.4.2 Main View 62
3.4.3 Code Area 62
3.4.4 Data Area 63
3.4.5 Configuration Area 63
3.5 PROPIC II 64

4. THE CONTROL UNIT HARDWARE AND SOFTWARE 65

4.1 GENERAL 65
4.2 THE OPERATION OF THE DEVICE 66
4.2.1 Read Mode 66
4.2.2 Edit Mode 67
4.3 DESCRIPTION OF THE SOFTWARE 67
4.3.1 The Beginning of the Program 67
4.3.2 ISR Interrupt Service Routine 67
4.3.3 Timer0 Interrupt Service Routine 68
4.3.4 On Index Procedure 69
4.3.5 Load Compare Value Procedure 70
4.3.6 Relays Handler Procedure 71
4.3.7 LCD Handler Procedure 71
4.3.8 Clock Interrupt Service Routine 72
4.3.9 Main Routine 74
4.3.9.1 Read Mode 76
4.3.9.2 Edit Mode 76
4.4 HARDWARE OF THE CONTROLLER UNIT 76
4.4.1 MCU Unit 76
4.4.2 Relay Board 78
4.4.3 LCD Display 79
4.4.4 Keypad 79
4.4.5 Power Supply and Clock Generator 81

5. APPLICATIONS OF THE CONTROL UNIT:
POWER CABLE ACCESSORIES TESTS AND ELECTRO DISCHARGE MACHINING 83
5.1 Thermal Cycling Test 83
5.1.1 Installation 83
5.1.2 Method of the Test 84
5.1.3 Immersion test for outdoor terminations 85
5.1.3.1 Installation 85
5.1.3.2 Method 86
5.1.4 Automation method with the control unit 86
5.2 Screen Fault Current Initiation Test 86
5.2.1 Installation 87
5.2.2 Method 87
5.2.2.1 Solidly Earthed System 87
5.2.2.2 Unearthed or Impedance Earthed System 88
5.2.3 Automation method with the control unit 88
5.3 Electro Erosion Machining 89
5.3.1 Introduction 89
5.3.2 Advantages and Disadvantages of EDM 90
5.3.3 Automation method with the control unit 91
6. CONCLUSIONS 93
REFERENCES 95
APPENDICES
A. THE SOURCE CODE OF THE CONTROLLER 97
B. INSTRUCTION SET OF THE PIC 16F877 123
C. MPASM DIRECTIVE SUMMARY 128

LIST OF TABLES

TABLE

2.1 Key Features of PIC16F877 13
2.2 PIC16F877 Pinout Description 15
2.3 Data Memory Bank Select Bits 18
2.4 CCP Mode – Timer Resources Required 34
2.5 Debugger Resources 50
3.1 MPASM Default Extensions 57
4.1 EEPROM Organization 70
5.1 Sample values of the parameters in EDM process 92

LIST OF FIGURES

FIGURES

2.1 Microprocessor system contrasted with microcontroller system 6
2.2 Harvard vs. von Neumann Block Architectures 9
2.3 Block Diagram of PIC 16F877 14
2.4 Pin Diagram of PIC 16F877 15
2.5 Program Memory Map And Stack 17
2.6 Loading of PC in Different Situations 19
2.7 Direct/Indirect Addressing 21
2.8 Typical I/O Port 22
2.9 Timer1 Incrementing Edge 31
2.10 Sample Device Configuration For I^2C Bus 39
2.11 RC Oscillator Mode 43
3.1 Generating Absolute Code 55
3.2 Creating a Reusable Object Library 56
3.3 Generating Executable Code From Object Modules 57
3.4 Project Relationships – MPASM 60
3.5 Main View of IC-Prog 62
4.1 Flow Chart of the ISR Interrupt Service Routine 68
4.2 Flow Chart of the T0 Interrupt Service Routine 69
4.3 Flow Chart of the LCD Handler 71
4.4 Flow Chart of the Clock Interrupt Service Routine 73
4.5 Flow Chart of the Main Routine 74
4.6 Circuit Diagram of the MCU 77
4.7 Circuit Diagram of the Relay Board 78
4.8 Circuit Diagram of the LCD Module 80

4.9 Circuit Diagram of the Keyboard.. 81
4.10 Circuit Diagram of the Power Supply.. 82
5.1 Terminations tested in air.. 84
5.2 Joints tested in air.. 84
5.3 Thermal Cycle... 85
5.4 Diagram of the EDM Process 89
5.5 Diagram of the Wire EDM Process 90

LIST OF ABBREVIATIONS

PIC	: Peripheral Interface Controller
RISC	: Reduced Instruction Set Computer
CISC	: Complex Instruction Set Computer
CPU	: Central Processing Unit
MCU	: Microcontroller Unit
PWM	: Pulse Width Modulation
rms	: Root Mean Square
I^2C	: Inter-Integrated Circuit
IDE	: Integrated Development Environment
A/D	: Analog-to-Digital
ROM	: Read Only Memory
RAM	: Random Access Memory

CHAPTER 1

INTRODUCTION

As it is known, the microcontrollers have been used widely in industrial automation. Before the microprocessor-based systems, the automation was being made by using mechanical and electrical contactors and relays. They were very complicated, had many connections and components. When a modification or addition was required, the most of these connections needed to be changed. In most cases, this was even harder than making a new one. With the development of microprocessor-based systems, the required variations can be carried out by doing modifications only in software, without any change in hardware. Therefore, the systems which are easy to develop and have few components have taken place of the old ones that have more components resulting in more risk for failure.

Switching of the devices, such as energizing of a heater or switching off the power supply, is only possible by an operator if no automation system is used. In some applications, timing is very important and no error is allowed. However, some errors or delays may be caused by the operator. To reduce possibility of occurrence of these events and to decrease the dependency on him, a microcontroller-based control unit can be used.

In this thesis, a control unit by using a microcontroller is designed with the aim of the automation of the long term and/or periodic processes such as power cable accessories tests and electro discharge machining. To use this unit in the different applications, it is planned as a general purpose controller by additional features and improvements.

To achieve this objective, to automate the switching of the devices, using timing relays may be an alternative. They can be on-delay or off-delay type and they are connected and set to proper operating times. However, this method has some drawbacks. The cost of the system is higher than the microcontroller-based system and the size is bigger since multiple timing relays must be used. Moreover they can not count the external clock signals and upgrading of this unit is more complex.

A microcontroller-based control system is designed to realize this project. These systems are compact, cost is lower and they are more flexible for reprogramming.

The Microchip PIC 16F877 is used as a microcontroller in this study. The main reasons of choosing this chip are its architecture and price. PIC microcontrollers are RISC (Reduced Instruction Set Computer) machines with only thirty-odd instructions. Each instruction usually executes in one internal clock. PICs have reduced addressing modes (direct and indirect addressing). 2K x 14 words on PIC 16Fxx is approximately equivalent to 4Kx8 words on other 8-bit microcontrollers.

16F877 has Flash type program memory, so it does not have to be taken off the board to load the program. Flash memory can be electrically programmed on-board over 1.000.000 times.

EEPROM memory makes it easier to apply microcontrollers to devices where permanent storage of various parameters is needed.

On the other hand, CISC machines like Intel 80x86, 8051 or Motorola 68HC11 have many instructions usually more than 100, many addressing modes and it usually takes more than one internal clock cycle to execute instructions. For instance, Motorola 68HC12 can be used but unfortunately it would not be well suited for this project. It is too powerful, too expensive and too picky (i.e. it requires a very precise circuit layout in order to function properly).

Low cost, low consumption, high speed, easy handling, more development tools and flexibility make PIC 16F877 applicable in our project.

The reason of the using assembly language in the software is its high speed and low code size.

The device in this study counts the clock signals, increments the counter and if it reaches the values loaded before, activates the relevant relay. There are two alternatives for this clock signal. It can be produced by internal clock generator, so the control unit operates as a programmable timer which controls the different devices. The second case is that, signals come to the microcontroller externally.

Furthermore, the device includes an Up/Down signal input. This signal is useful if the decrement of the counter is required in the application. It can be employed in the automation of the systems which involve the two opposite process such as mechanical movement in opposite directions. Counter of the microcontroller increases or decreases according to this signal.

One of the application fields of the U/D input is systems at which clock signals are generated by a position sensor. Position sensor sends the clock signals to the microcontroller while a device is moving. But, due to the vibration, the device moves in the opposite direction, so some unwanted signals are produced and sent. In this situation, sensor changes the signal from Up to Down or vice versa, therefore the counter increments or decrements. This is the well-known industrial application.

The third input for this controller is an external reset. By using this input, it can operate periodically. To achieve it, one of the output relays is connected to this input and set the appropriate value. At this value, relay operates; its contacts are closed and the microcontroller resets, so the process starts from beginning. This feature is very useful in cyclic processes.

It is possible to change the loaded operating times of the relays by entering new values using keypad. The counted clock signals and the operating times of the relays can be shown on LCD display.

The organization of the thesis is as follows:

- In Chapter 2, the theory of the microcontroller PIC 16F877 is studied in detail.

- In Chapter 3, the programming of the PIC microcontrollers is described. The hardware and software required to achieve it are defined.

- In Chapter 4, subject is the hardware and the software of the controller unit. The flowcharts and the circuit diagrams are included in this chapter.

- In Chapter 5, power cable accessories tests are explained. The automation of these tests is an application of the control unit designed in this study.

- In Chapter 6, the final conclusions on this study are made and the further work on this area is proposed.

CHAPTER 2

PIC 16F877 MICROCONTROLLER

2.1 MICROCONTROLLERS – THE GENERAL

A digital computer typically consists of three major components: The Central Processing Unit (CPU), program and data memory, and an Input/Output (I/O) system. The CPU controls the flow of information among the components of the computer. It also processes the data by performing digital operations. A microprocessor is a CPU that is compacted into a single-chip semiconductor device and they are general-purpose devices, suitable for many applications.

A microcontroller is an entire computer manufactured on a single chip. The I/O and memory subsystems contained in a microcontroller specialize these devices so that they can be interfaced with hardware and control functions of the applications. Since microcontrollers are powerful digital processors, the degree of control and programmability they provide significantly enhances the effectiveness of the application. Microcontrollers are usually dedicated devices embedded within an application. In order to serve applications, they have a high concentration of on-chip facilities such as serial ports, parallel input-output ports, timers, counters, interrupt control, analog-to-digital converters, random access memory, and read only memory. Figure 2.1 shows a comparison between a microprocessor system and a microcontroller system.

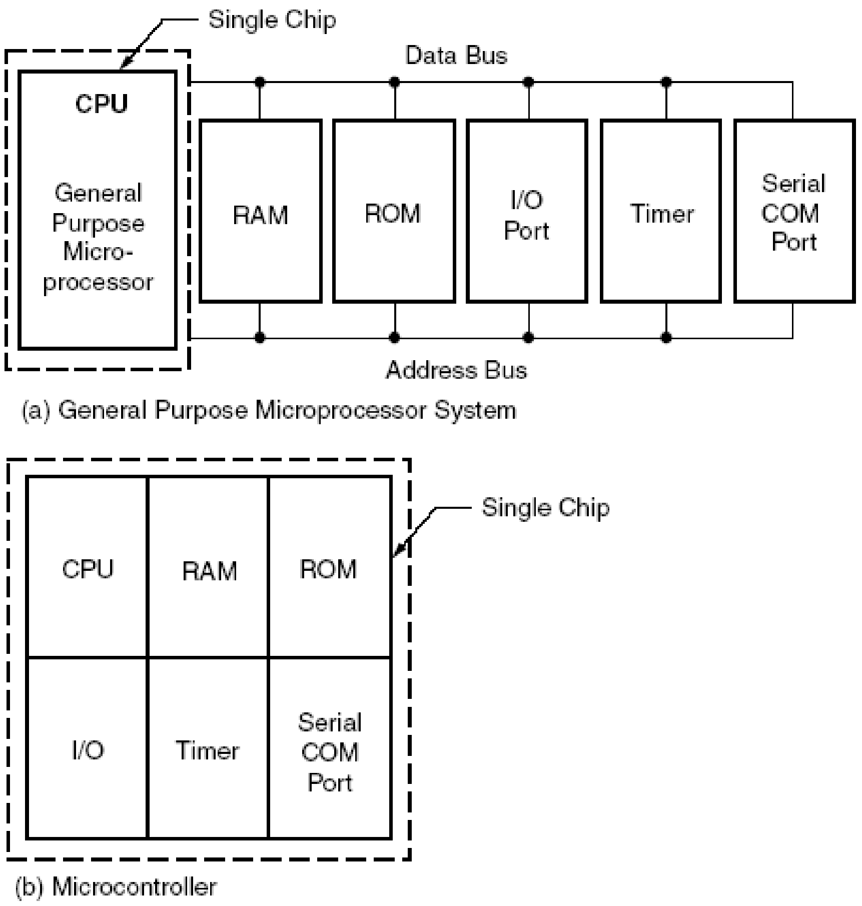


Figure 2.1 Microprocessor system contrasted with microcontroller system

Most of the microcontrollers have an 8-bit word size, at least 64 bytes of R/W memory, and 1 K byte of ROM. The range of I/O lines varies considerably, from 16 to 40 lines. However, most of these devices cannot be easily programmed unless they include EPROM on the chip.

2.2 PIC MICROCONTROLLERS

2.2.1 General View

The PICmicro was originally designed around 1980 by General Instrument as a small, fast, inexpensive embedded microcontroller with strong I/O capabilities. PIC stands for "**P**eripheral **I**nterface **C**ontroller". General Instrument recognized the potential for the little PIC and eventually spun off Microchip, to fabricate and market the PICmicro.

The PICmicro has some advantages in many applications over the older chips such as the Intel 8048/8051/8052 and its derivatives, the Motorola MC6805/6hHC11, and many others. Its unusual architecture is ideally suited for embedded control. Nearly all instructions execute in the same number of clock cycles, which makes timing control much easier. The PICmicro is a RISC (**R**educed **I**nstruction **S**et **C**omputer) design, with only thirty-odd instructions to remember; its code is extremely efficient, allowing the PIC to run with typically less program memory than its larger competitors.

Very important, though, is the low cost, high available clock speeds, small size, and incredible ease of use of the PIC. For timing-insensitive designs, the oscillator can consist of a cheap RC network. Clock speeds can range from low speed to 20MHz. Versions of the various PICmicro families are available that are equipped with various combinations ROM, EPROM, OTP (One-Time Programmable) EPROM, EEPROM, and FLASH program and data memory. An 18-pin PICmicro typically devotes 13 of those pins to I/O, giving the designer two full 8-bit I/O ports and an interrupt. In many cases, designing with a PICmicro is much simpler and more efficient than using an older, larger embedded microprocessor.

PICmicro processors are found in an incredible array of products. Remote controls, display panels, cars, appliances, weather stations, ham radio equipment, clocks, motor controllers, sensors, programmable thermostats, robots, toys, battery chargers, computer peripherals, almost anything using some sort of programmable logic.

2.2.2 Architecture Overview

PICmicro devices have a number of architectural features commonly found in RISC microprocessors. These include:

• Harvard Architecture

• Long Word Instructions

• Single Word Instructions

• Single Cycle Instructions

• Instruction Pipelining

• Reduced Instruction Set

• Register File Architecture

• Orthogonal (Symmetric) Instructions

2.2.2.1 Harvard Architecture

Harvard architecture has the program memory and data memory as separate memories and are accessed from separate buses. This improves bandwidth over traditional von Neumann architecture in which program and data are fetched from the same memory using the same bus. In other words, the von Neumann architecture uses the same bus for program memory, data memory, I/O, registers, etc. This makes it easy to bring the common bus out to device I/O pins for adding memory, but it limits the bus bandwidth that can be used for any one function since the bus is shared. Von Neumann processors are generally microcoded, CISC (**C**omplex **I**nstruction **S**et **C**omputer) designs (though there are exceptions). To execute an instruction, a von Neumann machine must make one or more (generally more) accesses across the 8-bit bus to fetch the instruction. Then data may need to be fetched, operated on, and possibly written. As can be seen from this description, that bus can be extremely congested. While with a Harvard architecture, the instruction is fetched in a single instruction cycle (all 14-bits). While the program memory is being accessed, the data memory is on an independent bus and can be read and written. These separated buses allow one instruction to execute while the next instruction is fetched. A comparison of Harvard vs. von-Neumann architectures is shown in Figure 2.2.

The Harvard architecture uses separate program memory and data memory busses. This makes it easy to design the processor for very efficient use of program memory, since the program memory bus can be of a much different width than the data memory. Instructions usually (always in the case of the PIC) take up only one program memory location, compared to one, two or even three in a typical von Neumann design. Harvard-architecture machines are generally non-microcoded, RISC (**R**educed **I**nstruction **S**et **C**omputer) designs (again, exceptions are to be found). One drawback to the Harvard architecture is that it is very difficult to bring the memory address and data busses out to device pins, so adding external program memory is difficult at best. For this reason, most Harvard machines have only internal program memory.

For example, the popular PIC16F84 contains 1K words of FLASH program memory, 68 bytes of data RAM, and 64 bytes of data EEPROM. While this seems like an extremely limited amount of code and data space, the PIC's incredibly compact code makes the most of it. 1024 instruction word memory actually means 1024 instructions, no less. Even immediate-mode instructions, where an operand is part of the instruction itself, takes only one memory location, as do CALL and GOTO instructions. There even exists a single-chip implementation of a TCP/IP stack and HTTP server written for a 16F84.

The PIC is also a non-microcoded design. In larger processors, each binary machine language instruction often is "executed" by a series of microcode steps. While this is a great approach for building large, complex processors with a wide range of instructions, it also leads to great complexity and takes up a lot of real estate. The PIC uses the instruction word itself, decoded by logic gates as it is read from program memory, to control the flow of data through the chip.

The seemingly odd 14-bit instruction word length is a direct result of the internal architecture of the processor itself. In the case of the 16F84 or 16C711, we need 13 bits just to address all of program memory. In the case of the smaller 16C54 with only 512 words of program memory and 25 bytes of RAM, we can get by with a 12-bit instruction word -- which is exactly what the 16C54 uses. Conversely, with more memory we would use a longer instruction word, like the 16 bits in the 18Cxxx family.

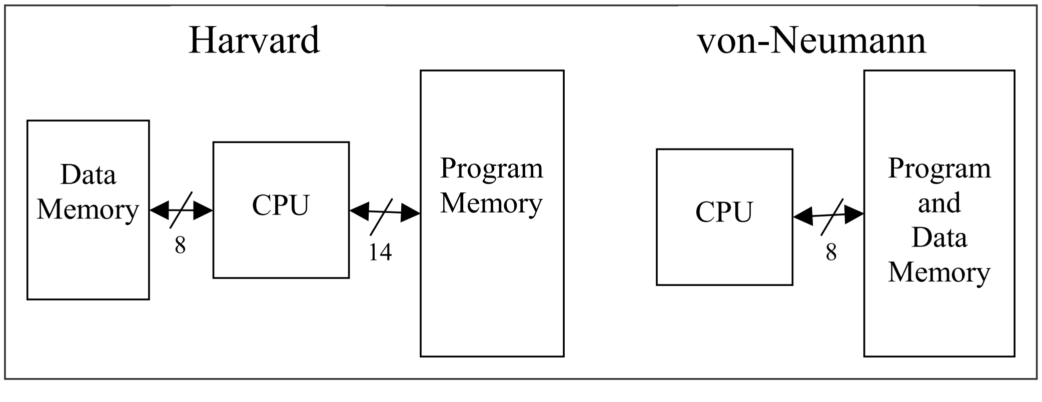


Figure 2.2 Harvard vs. von Neumann Block Architectures

2.2.2.2 Long Word Instructions

Long word instructions have a wider (more bits) instruction bus than the 8-bit Data Memory Bus. This is possible because the two buses are separate. This further allows instructions to be sized differently than the 8-bit wide data word which allows a more efficient use of the program memory, since the program memory width is optimized to the architectural requirements.

2.2.2.3 Single Word Instructions

Single Word instruction opcodes are 14-bits wide making it possible to have all single word instructions. A 14-bit wide program memory access bus fetches a 14-bit instruction in a single cycle. With single word instructions, the number of words of program memory locations equals the number of instructions for the device. This means that all locations are valid instructions.

Typically in the von Neumann architecture, most instructions are multi-byte. In general, a device with 4-KBytes of program memory would allow approximately 2K of instructions. This 2:1 ratio is generalized and dependent on the application code. Since each instruction may take multiple bytes, there is no assurance that each location is a valid instruction.

2.2.2.4 Instruction Pipeline

The instruction pipeline is a two-stage pipeline which overlaps the fetch and execution of instructions. The fetch of the instruction takes one machine cycle, while the execution takes another machine cycle. However, due to the overlap of the fetch of current instruction and execution of previous instruction, an instruction is fetched and another instruction is executed every single machine cycle.

2.2.2.5 Single Cycle Instructions

With the Program Memory bus being 14-bits wide, the entire instruction is fetched in a single machine cycle. The instruction contains all the information required and is executed in a single cycle. There may be a one cycle delay in execution if the result of the instruction modified the contents of the Program Counter. This requires the pipeline to be flushed and a new instruction to be fetched.

2.2.2.6 Reduced Instruction Set

When an instruction set is well designed and highly orthogonal (symmetric), fewer instructions are required to perform all needed tasks. With fewer instructions, the whole set can be more rapidly learned.

2.2.2.7 Register File Architecture

The register files/data memory can be directly or indirectly addressed. All special function registers, including the program counter, are mapped in the data memory.

2.2.2.8 Orthogonal (Symmetric) Instructions

Orthogonal instructions make it possible to carry out any operation on any register using any addressing mode. This symmetrical nature and lack of “special instructions” make programming simple yet efficient. In addition, the learning curve is reduced significantly. The instruction set uses only two non-register oriented instructions, which are used for two of the cores features. One is the SLEEP instruction which places the device into the lowest power use mode. The other is the CLRWDT instruction which verifies the chip is operating properly by preventing the on-chip Watchdog Timer (WDT) from overflowing and resetting the device.

2.3 PIC 16F877 ARCHITECTURE

2.3.1 Features of the PIC16F877

2.3.1.1 Microcontroller Core Features

• RISC (Reduced Instruction Set Computer) CPU
• Only 35 single word instructions

• All single cycle instructions except for program branches which are two cycle

• Operating speed: DC - 20 MHz clock input

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

• Interrupt capability (up to 14 sources)

• 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 FLASH/EEPROM technology

• Fully static design

• In-Circuit Serial Programming (ICSP) via two pins

• Single 5V In-Circuit Serial Programming capability

• In-Circuit Debugging via two pins

• Processor read/write access to program memory

• Wide operating voltage range: 2.0V to 5.5V

• High Sink/Source Current: 25 mA

• Low-power consumption:

- < 0.6 mA typical at 3V, 4 MHz

- 20 μA typical at 3V, 32 kHz

- < 1 μA typical standby current

2.3.1.2 Peripheral Features

• Timer0: 8-bit timer/counter with 8-bit prescaler

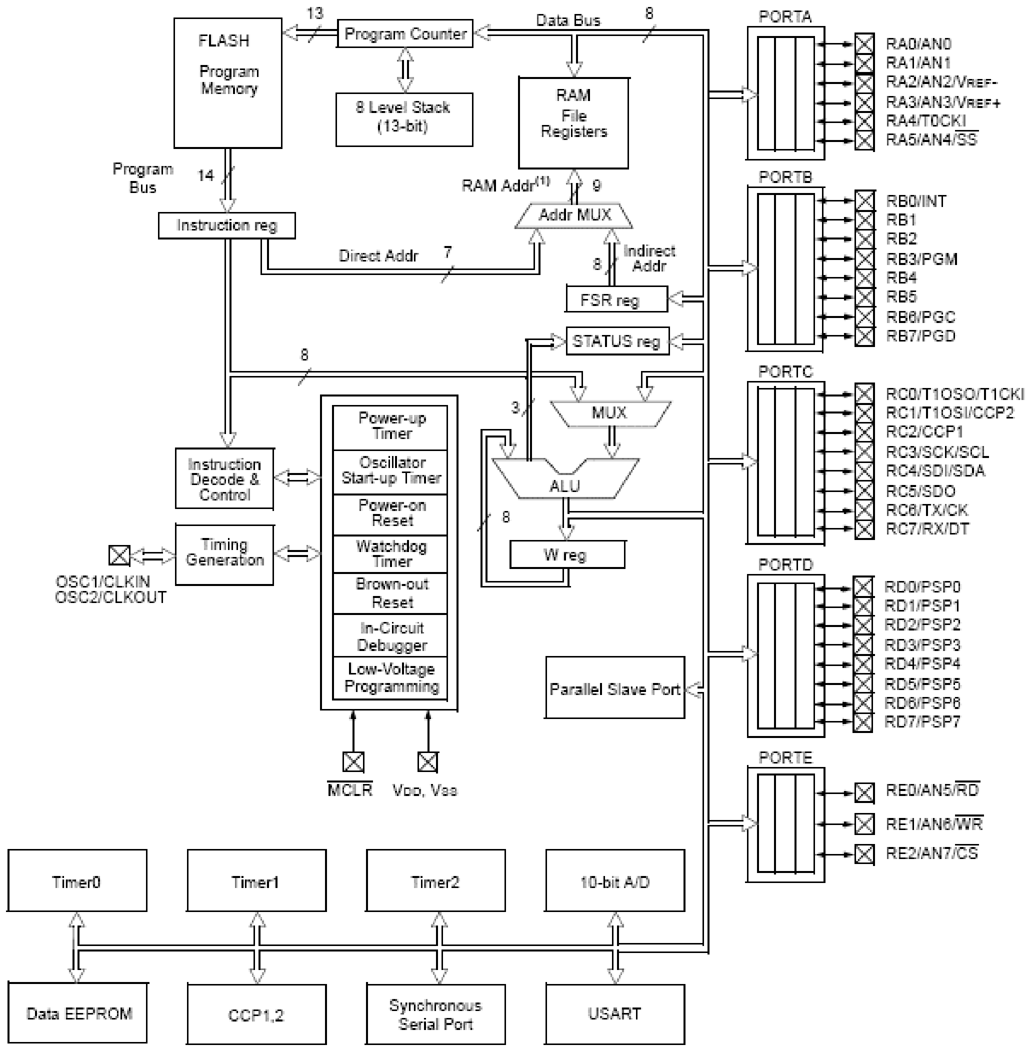
• Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock

- 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 I^2C (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

Table 2.1 Key Features of PIC16F877

Operating Frequency	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	8K
Data Memory (bytes)	368
EEPROM Data Memory	256
Interrupts	14
I/O Ports	Ports A,B,C,D,E
Timers	3
Capture/Compare/PWM Modules	2
Serial Communications	MSSP, USART
Parallel Communications	PSP
10-bit Analog-to-Digital Module	8 input channels
Instruction Set	35 instructions

2.3.2 Block Diagram and Pinout Description of PIC16F877



Note 1: Higher order bits are from the STATUS register.

Device	Program FLASH	Data Memory	Data EEPROM
PIC 16F877	8 K	368 Bytes	256 Bytes

Figure 2.3 Block Diagram of PIC 16F877

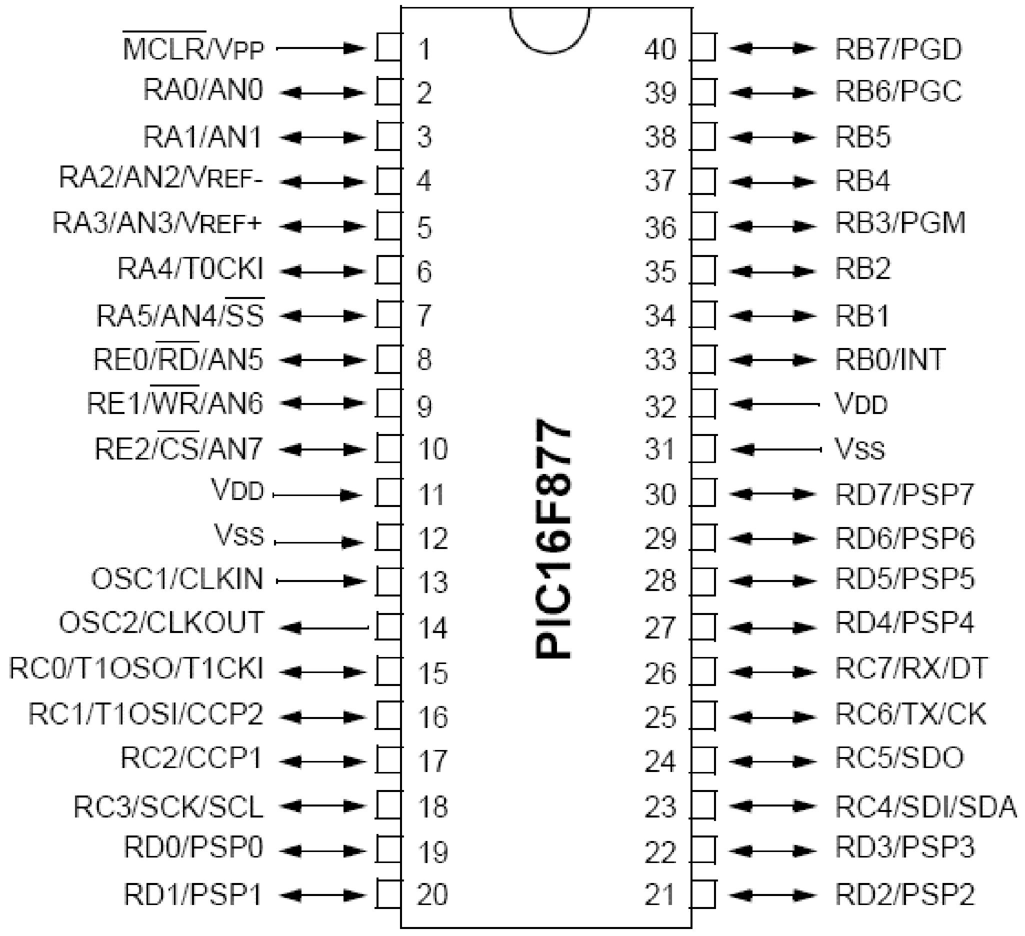


Figure 2.4 Pin Diagram of PIC 16F877

Table 2.2 PIC16F877 Pinout Description

Pin Name	Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	13	I	ST/CMOS$^{(4)}$	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	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.
$\overline{\text{MCLR}}$/Vpp	1	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
RA0/AN0	2	I/O	TTL	PORTA is a bi-directional I/O port. RA0 can also be analog input0.
RA1/AN1	3	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/VREF-	4	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.

Table 2.2 PIC16F877 Pinout Description (Continued)

RA4/T0CKI	6	I/O	ST	RA4 can also be the clock input to the Timer0 timer/counter. Output is open drain type.
RA5/$\overline{\text{SS}}$/AN4	7	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
RB0/INT	33	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	I/O	TTL	
RB2	35	I/O	TTL	
RB3/PGM	36	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	37	I/O	TTL	Interrupt-on-change pin.
RB5	38	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	39	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	40	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.
RC0/T1OSO/T1CKI	15	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/T1OSI/CCP2	16	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	17	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	18	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	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^{2}C mode).
RC5/SDO	24	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	25	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	26	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
RD0/PSP0	19	I/O	ST/TTL(3)	PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.
RD1/PSP1	20	I/O	ST/TTL(3)	
RD2/PSP2	21	I/O	ST/TTL(3)	
RD3/PSP3	22	I/O	ST/TTL(3)	
RD4/PSP4	27	I/O	ST/TTL(3)	
RD5/PSP5	28	I/O	ST/TTL(3)	
RD6/PSP6	29	I/O	ST/TTL(3)	
RD7/PSP7	30	I/O	ST/TTL(3)	
RE0/$\overline{\text{RD}}$/AN5	8	I/O	ST/TTL(3)	PORTE is a bi-directional I/O port. RE0 can also be read control for the parallel slave port, or analog input5.
RE1/$\overline{\text{WR}}$/AN6	9	I/O	ST/TTL(3)	RE1 can also be write control for the parallel slave port, or analog input6.
RE2/$\overline{\text{CS}}$/AN7	10	I/O	ST/TTL(3)	RE2 can also be select control for the parallel slave port, or analog input7.
Vss	12,31	P	—	Ground reference for logic and I/O pins.
VDD	11,32	P	—	Positive supply for logic and I/O pins.

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.

2.3.3 Memory Organization

There are three memory blocks in each of the PIC16F877 MCUs. The Program Memory and Data Memory have separate buses so that concurrent access can occur. The other data memory block is EEPROM.

2.3.3.1 Program Memory Organization

The PIC16F877 has a 13-bit program counter capable of addressing an 8K x 14 program memory space. It has 8K x 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.

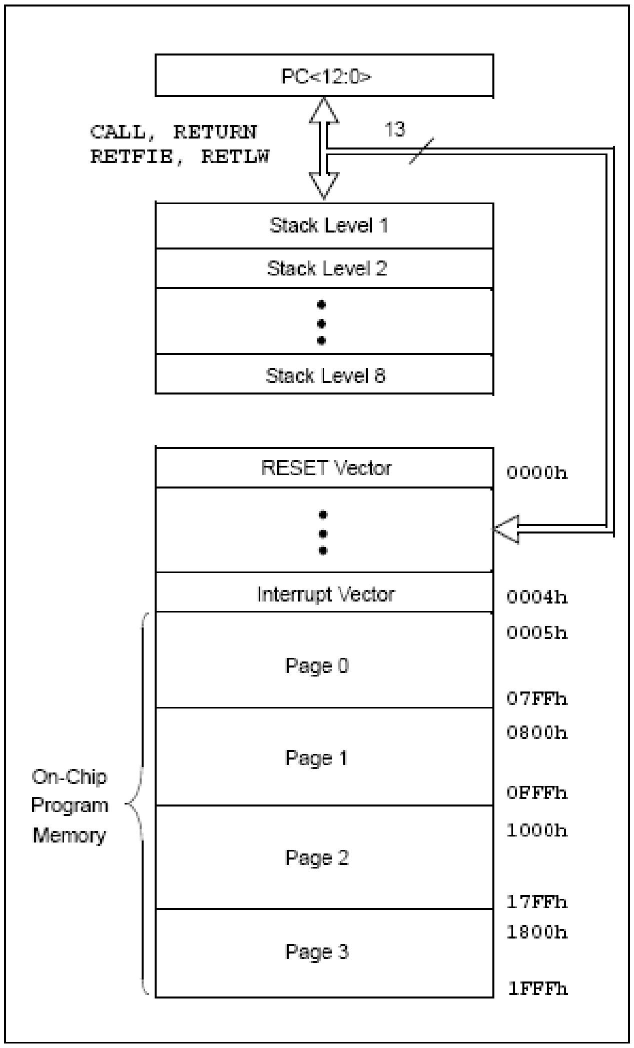


Figure 2.5 Program Memory Map And Stack

2.3.3.2 Data Memory Organization

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

Table 2.3 Data Memory Bank Select Bits

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

Each bank extends up to 7Fh (128 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.

- General Purpose Register File (GPR)

PIC 16F877 has banked memory in the GPR area. GPRs are not initialized by a Power-on Reset and are unchanged on all other resets.

The register file can be accessed either directly, or using the File Select Register FSR, indirectly. Some PICs have areas that are shared across the data memory banks, so a read / write to that area will appear as the same location (value) regardless of the current bank. This area is referred as the Common RAM.

- Special Function Registers

The Special Function Registers are registers used by the CPU and peripheral modules for controlling the desired operation of the device. These registers are implemented as static RAM.

The Special Function Registers can be classified into two sets: core (CPU) and peripheral.

PIC 16F877 has banked memory in the SFR area. Switching between these banks requires the RP0 and RP1 bits in the STATUS register to be configured for the desired bank. Some SFRs are initialized by a Power-on Reset and other resets, while other SFRs are unaffected.

2.3.3.3 PCL and PCLATH

The program counter (PC) is 13-bits wide. The low byte comes from the PCL register, which is a readable and writable register. The upper bits (PC<12:8>) are not readable, but are indirectly writable through the PCLATH register. On any RESET, the upper bits of the PC will be cleared. Figure 2.6 shows the two situations for the loading of the PC. The upper example in the figure shows how the PC is loaded on a write to PCL (PCLATH<4:0> → PCH). The lower example in the figure shows how the PC is loaded during a CALL or GOTO instruction (PCLATH<4:3> → PCH).

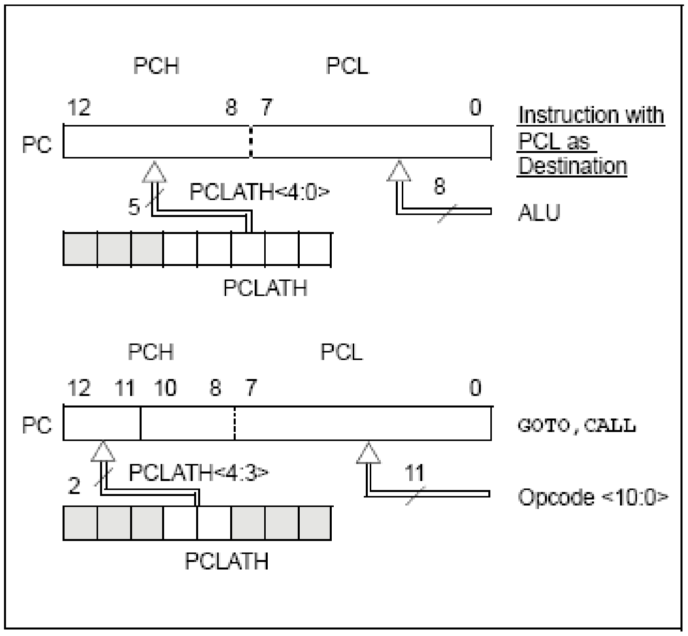


Figure 2.6 Loading of PC in Different Situations

- Computed GOTO

A computed GOTO is accomplished by adding an offset to the program counter (ADDWF PCL). When doing a table read using a computed GOTO method, care should be exercised if the table location crosses a PCL memory boundary (each 256 byte block).

- Stack

The PIC16F877 has an 8-level deep x 13-bit wide hardware stack. The stack space is not part of either program or data space and the stack pointer is not readable or writable. The PC is PUSHed onto the stack when a CALL instruction is executed, or an interrupt causes a branch. The stack is POPed in the event of a RETURN, RETLW or a RETFIE instruction execution. PCLATH is not affected by a PUSH or POP operation.

The stack operates as a circular buffer. This means that after the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on).

2.3.3.4 Program Memory Paging

16F877 is capable of addressing a continuous 8K word block of program memory. The CALL and GOTO instructions provide only 11 bits of address to allow branching within any 2K program memory page. When doing a CALL or GOTO instruction, the upper 2 bits of the address are provided by PCLATH<4:3>. When doing a CALL or GOTO instruction, the user must ensure that the page select bits are programmed so that the desired program memory page is addressed. If a return from a CALL instruction (or interrupt) is executed, the entire 13-bit PC is popped off the stack. Therefore, manipulation of the PCLATH<4:3> bits is not required for the return instructions (which POPs the address from the stack).

2.3.3.5 Indirect Addressing, INDF and FSR Registers

The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing. Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself, indirectly (FSR = ‘0’) will read 00h. Writing to the INDF register indirectly results in a no operation (although status bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 2-7.

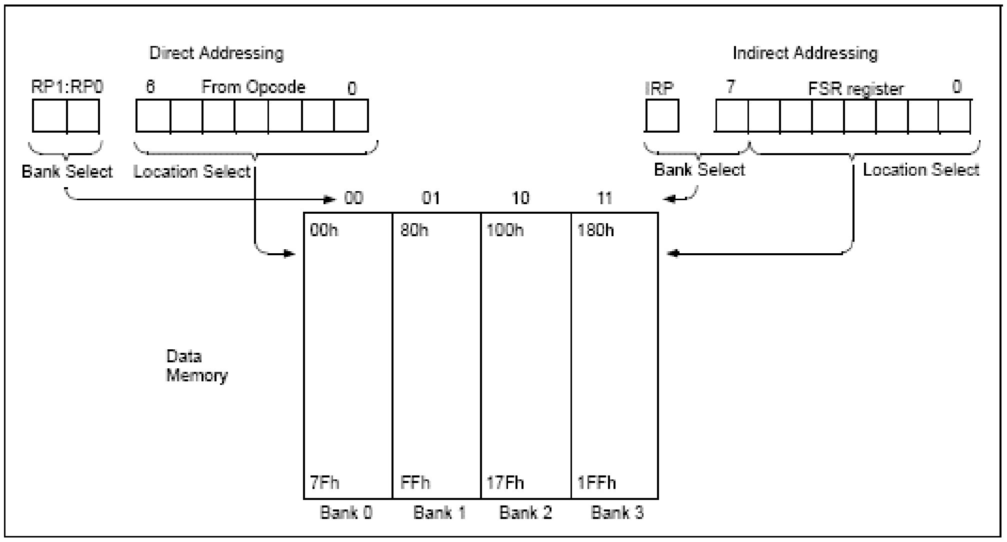


Figure 2.7 Direct/Indirect Addressing

2.3.4 I/O Ports

General purpose I/O pins can be considered the simplest of peripherals. They allow the PICmicro to monitor and control other devices. To add flexibility and functionality to a device, some pins are multiplexed with an alternate function(s). These functions depend on which peripheral features are on the device. In general, when a peripheral is functioning, that pin may not be used as a general purpose I/O pin.

For most ports, the I/O pin's direction (input or output) is controlled by the data direction register, called the TRIS register. TRIS<x> controls the direction of PORT<x>. A '1' in the TRIS bit corresponds to that pin being an input, while a '0' corresponds to that pin being an output. An easy way to remember is that a '1' looks like an I (input) and a '0' looks like an O (output).

The PORT register is the latch for the data to be output. When the PORT is read, the device reads the levels present on the I/O pins (not the latch). This means that care should be taken with read-modify-write commands on the ports and changing the direction of a pin from an input to an output.

Figure 2.8 shows a typical I/O port. This does not take into account peripheral functions that may be multiplexed onto the I/O pin. Reading the PORT register reads the status of the pins whereas writing to it will write to the port latch. All write operations (such as BSF and BCF instructions) are read-modify-write operations. Therefore a write to a port implies that the port pins are read; this value is modified, and then written to the port data latch.

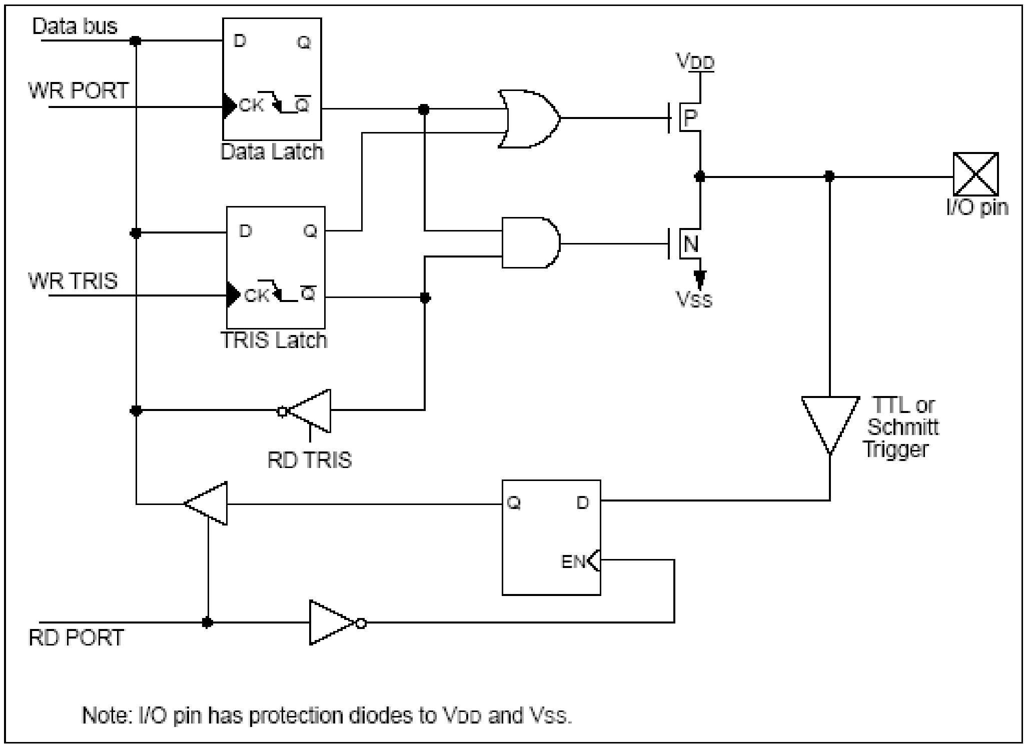


Figure 2.8 Typical I/O Port

When peripheral functions are multiplexed onto general I/O pins, the functionality of the I/O pins may change to accommodate the requirements of the peripheral module. Examples of this are the Analog-to-Digital (A/D) converter and LCD driver modules, which force the I/O pin to the peripheral function when the device is reset. In the case of the A/D, this prevents the device from consuming excess current if any analog levels were on the A/D pins after a reset occurred.

With some peripherals, the TRIS bit is overridden while the peripheral is enabled. Therefore, read-modify-write instructions (BSF, BCF, XORWF) with TRIS as destination should be avoided.

PORT pins may be multiplexed with analog inputs and analog V_{REF} input. The operation of each of these pins is selected, to be an analog input or digital I/O, by clearing/setting the control bits in the ADCON1 register (A/D Control Register1). When selected as an analog input, these pins will read as ‘0’s.

The TRIS registers control the direction of the port pins, even when they are being used as analog inputs. The user must ensure the TRIS bits are maintained set when using the pins as analog inputs.

2.3.4.1 PortA and the 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).

Reading the PORTA register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read; the value is modified and then written to the port data latch.

Pin RA4 is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin. The RA4/T0CKI pin is a Schmitt Trigger input and an open drain output. All other PORTA pins have TTL input levels and full CMOS output drivers.

Other PORTA pins are multiplexed with analog inputs and analog V_{REF} input. The operation of each pin is selected by clearing/setting the control bits in the ADCON1 register (A/D Control Register1).

The TRISA register controls the direction of the RA pins, even when they are being used as analog inputs. The user must ensure the bits in the TRISA register are maintained set when using them as analog inputs.

2.3.4.2 PortB and the 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.

Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit RBPU (OPTION_REG<7>). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

Four of the PORTB pins, RB7:RB4, have an interrupt-on-change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt-on-change comparison). The input pins (of RB7:RB4) are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of RB7:RB4 are OR'ed together to generate the RB Port Change Interrupt with flag bit RBIF (INTCON<0>).

This interrupt can wake the device from SLEEP. The user, in the Interrupt Service Routine, can clear the interrupt in the following manner:

a) Any read or write of PORTB. This will end the mismatch condition.

b) Clear flag bit RBIF.

A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition and allow flag bit RBIF to be cleared.

The interrupt-on-change feature is recommended for wake-up on key depression operation and operations where PORTB is only used for the interrupt-on-change feature. Polling of PORTB is not recommended while using the interrupt-on-change feature.

This interrupt-on-mismatch feature, together with software configurable pull-ups on these four pins, allow easy interface to a keypad and make it possible for wake-up on key depression.

RB0/INT is an external interrupt input pin and is configured using the INTEDG bit (OPTION_REG<6>).

2.3.4.3 PortC and the 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). Clearing a TRISC bit (= 0) will make the corresponding PORTC pin an output (i.e., put the contents of the output latch on the selected pin).

PORTC is multiplexed with several peripheral functions. PORTC pins have Schmitt Trigger input buffers.

When the I^2C module is enabled, the PORTC<4:3> pins can be configured with normal I^2C levels or with SMBus levels by using the CKE bit (SSPSTAT<6>).

When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin. Some peripherals override the TRIS bit to make a pin an output, while other peripherals override the TRIS bit to make a pin an input. Since the TRIS bit override is in effect while the peripheral is enabled, read-modify-write instructions (BSF, BCF, XORWF) with TRISC as destination, should be avoided.

2.3.4.4 PortD and the TRISD Register

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.

2.3.4.5 PortE and the 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.

The PORTE pins become the I/O control inputs for the microprocessor port when bit PSPMODE (TRISE<4>) is set. In this mode, the user must make certain that the TRISE<2:0> bits are set, and that the pins are configured as digital inputs. Also ensure that ADCON1 is configured for digital I/O. In this mode, the input buffers are TTL.

TRISE register controls the parallel slave port operation. PORTE pins are multiplexed with analog inputs. When selected for analog input, these pins will read as ’0’s.

TRISE controls the direction of the RE pins, even when they are being used as analog inputs. The user must make sure to keep the pins configured as inputs when using them as analog inputs.

2.3.4.6 Parallel Slave Port

PORTD operates as an 8-bit wide Parallel Slave Port (PSP) or microprocessor port, when control bit PSPMODE (TRISE<4>) is set. In Slave mode, it is asynchronously readable and writable by the external world through $\overline{\text{RD}}$ control input pin RE0/$\overline{\text{RD}}$ and $\overline{\text{WR}}$ control input pin RE1/$\overline{\text{WR}}$.

The PSP can directly interface to an 8-bit microprocessor data bus. The external microprocessor can read or write the PORTD latch as an 8-bit latch. Setting bit PSPMODE enables port pin RE0/$\overline{\text{RD}}$ to be the $\overline{\text{RD}}$ input, RE1/$\overline{\text{WR}}$ to be the $\overline{\text{WR}}$ input and RE2/$\overline{\text{CS}}$ to be the $\overline{\text{CS}}$ (chip select) input. For this functionality, the corresponding data direction bits of the TRISE register (TRISE<2:0>) must be configured as inputs (set). The A/D port configuration bits PCFG3:PCFG0 (ADCON1<3:0>) must be set to configure pins RE2:RE0 as digital I/O.

There are actually two 8-bit latches: one for data output, and one for data input. The user writes 8-bit data to the PORTD data latch and reads data from the port pin latch (note that they have the same address). In this mode, the TRISD register is ignored, since the external device is controlling the direction of data flow.

A write to the PSP occurs when both the $\overline{\text{CS}}$ and $\overline{\text{WR}}$ lines are first detected low. When either the $\overline{\text{CS}}$ or $\overline{\text{WR}}$ lines become high (level triggered), the Input Buffer Full (IBF) status flag bit (TRISE<7>) is set on the Q4 (fourth quarter) clock cycle, following the next Q2 cycle, to signal the write is complete. The interrupt flag bit PSPIF (PIR1<7>) is also set on the same Q4 clock cycle. IBF can only be cleared by reading the PORTD input latch. The Input Buffer Overflow (IBOV) status flag bit

(TRISE<5>) is set if a second write to the PSP is attempted when the previous byte has not been read out of the buffer.

A read from the PSP occurs when both the $\overline{\text{CS}}$ and $\overline{\text{RD}}$ lines are first detected low. The Output Buffer Full (OBF) status flag bit (TRISE<6>) is cleared immediately, indicating that the PORTD latch is waiting to be read by the external bus. When either the $\overline{\text{CS}}$ or $\overline{\text{RD}}$ pin becomes high (level triggered), the interrupt flag bit PSPIF is set on the Q4 clock cycle, following the next Q2 cycle, indicating that the read is complete. OBF remains low until data is written to PORTD by the user firmware.

When not in PSP mode, the IBF and OBF bits are held clear. However, if flag bit IBOV was previously set, it must be cleared in firmware.

An interrupt is generated and latched into flag bit PSPIF when a read or write operation is completed. PSPIF must be cleared by the user in firmware and the interrupt can be disabled by clearing the interrupt enable bit PSPIE (PIE1<7>).

2.3.5 Data EEPROM and Flash Program Memory

The Data EEPROM and FLASH Program Memory are readable and writable during normal operation over the entire VDD range. These operations take place on a single byte for Data EEPROM memory and a single word for Program memory. A write operation causes an erase-then-write operation to take place on the specified byte or word. A bulk erase operation may not be issued from user code (which includes removing code protection).

Access to program memory allows for checksum calculation. The values written to program memory do not need to be valid instructions. Therefore, up to 14-bit numbers can be stored in memory for use as calibration parameters, serial numbers, packed 7-bit ASCII, etc. Executing a program memory location containing data that form an invalid instruction, results in the execution of a NOP instruction.

The EEPROM Data memory is rated for high erase/write cycles. The FLASH program memory is rated much lower, because EEPROM data memory can be used to store frequently updated values. An on-chip timer controls the write time and it will vary with voltage and temperature, as well as from chip to chip.

A byte or word write automatically erases the location and writes the new value (erase before write). Writing to EEPROM data memory does not impact the operation of the device. Writing to program memory will cease the execution of instructions until the write is complete. The program memory cannot be accessed during the write. During the write operation, the oscillator continues to run, the peripherals continue to function and interrupt events will be detected and essentially "queued" until the write is complete. When the write completes, the next instruction in the pipeline is executed and the branch to the interrupt vector will take place, if the interrupt is enabled and occurred during the write.

Read and write access to both memories take place indirectly through a set of Special Function Registers (SFR). The six SFRs used are:

- EEDATA
- EEDATH
- EEADR
- EEADRH
- EECON1
- EECON2

The EEPROM data memory allows byte read and write operations without interfering with the normal operation of the microcontroller. When interfacing to EEPROM data memory, the EEADR register holds the address to be accessed. Depending on the operation, the EEDATA register holds the data to be written, or the data read, at the address in EEADR. The PIC16F874 has 128 bytes of EEPROM data memory and therefore, requires that the MSb of EEADR remain clear. The EEPROM data memory on this device does not wrap around to 0, i.e., 0x80 in the EEADR does not map to 0x00. The PIC16F877 has 256 bytes of EEPROM data memory and therefore, uses all 8-bits of the EEADR.

The FLASH program memory allows non-intrusive read access, but write operations cause the device to stop executing instructions, until the write completes. When interfacing to the program memory, the EEADRH:EEADR registers form a two-byte word, which holds the 13-bit address of the memory location being accessed. The register combination of EEDATH:EEDATA holds the 14-bit data for writes, or reflects the value of program memory after a read operation. Just as in

EEPROM data memory accesses, the value of the EEADRH:EEADR registers must be within the valid range of program memory, depending on the device: 0000h to 3FFFh for the PIC16F877. Addresses outside of this range do not wrap around to 0000h (i.e., 4000h does not map to 0000h on the PIC16F877).

2.3.6 Timer0 Module

The Timer0 module timer/counter has the following features:

• 8-bit timer/counter

• Readable and writable

• 8-bit software programmable prescaler

• Internal or external clock select

• Interrupt on overflow from FFh to 00h

• Edge select for external clock

Timer mode is selected by clearing bit T0CS (OPTION_REG<5>). In Timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If the TMR0 register is written, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

Counter mode is selected by setting bit T0CS (OPTION_REG<5>). In Counter mode, Timer0 will increment either on every rising, or falling edge of pin RA4/T0CKI. The incrementing edge is determined by the Timer0 Source Edge Select bit, T0SE (OPTION_REG<4>). Clearing bit T0SE selects the rising edge.

The prescaler is mutually exclusively shared between the Timer0 module and the Watchdog Timer. The prescaler is not readable or writable.

2.3.6.1 Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h. This overflow sets bit T0IF (INTCON<2>). The interrupt can be masked by clearing bit T0IE (INTCON<5>). Bit T0IF must be cleared in software by the Timer0 module Interrupt Service Routine before re-enabling this interrupt. The TMR0 interrupt cannot awaken the processor from SLEEP, since the timer is shut-off during SLEEP.

2.3.6.2 Using Timer0 with an External Clock

When no prescaler is used, the external clock input is the same as the prescaler output. The synchronization of T0CKI with the internal phase clocks is accomplished by sampling the prescaler output on the Q2 and Q4 cycles of the internal phase clocks. Therefore, it is necessary for T0CKI to be high for at least 2Tosc (and a small RC delay of 20 ns) and low for at least 2Tosc (and a small RC delay of 20 ns).

2.3.6.3 Prescaler

There is only one prescaler available, which is mutually exclusively shared between the Timer0 module and the Watchdog Timer. A prescaler assignment for the Timer0 module means that there is no prescaler for the Watchdog Timer, and vice-versa. This prescaler is not readable or writable.

The PSA and PS2:PS0 bits (OPTION_REG<3:0>) determine the prescaler assignment and prescale ratio.

When assigned to the Timer0 module, all instructions writing to the TMR0 register (e.g. CLRF 1, MOVWF 1, BSF 1,x....etc.) will clear the prescaler. When assigned to WDT, a CLRWDT instruction will clear the prescaler along with the Watchdog Timer. The prescaler is not readable or writable.

2.3.7 Timer1 Module

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (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 (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit TMR1IE (PIE1<0>).

Timer1 can operate in one of two modes:

• As a timer

• As a counter

The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).

In Timer mode, Timer1 increments every instruction cycle. In Counter mode, it increments on every rising edge of the external clock input.

Timer1 can be enabled/disabled by setting/clearing control bit TMR1ON (T1CON<0>). Timer1 also has an internal "RESET input". This RESET can be generated by either of the two CCP modules.

When the Timer1 oscillator is enabled (T1OSCEN is set), the RC1/T1OSI/CCP2 and RC0/T1OSO/T1CKI pins become inputs. That is, the TRISC<1:0> value is ignored, and these pins read as '0'.

2.3.7.1 Timer1 Operation in Timer Mode

Timer mode is selected by clearing the TMR1CS (T1CON<1>) bit. In this mode, the input clock to the timer is FOSC/4. The synchronize control bit $\overline{\text{T1SYNC}}$ (T1CON<2>) has no effect, since the internal clock is always in sync.

2.3.7.2 Timer1 Counter Operation

Timer1 may operate in either a Synchronous, or an Asynchronous mode, depending on the setting of the TMR1CS bit.

When Timer1 is being incremented via an external source, increments occur on a rising edge. After Timer1 is enabled in Counter mode, the module must first have a falling edge before the counter begins to increment.

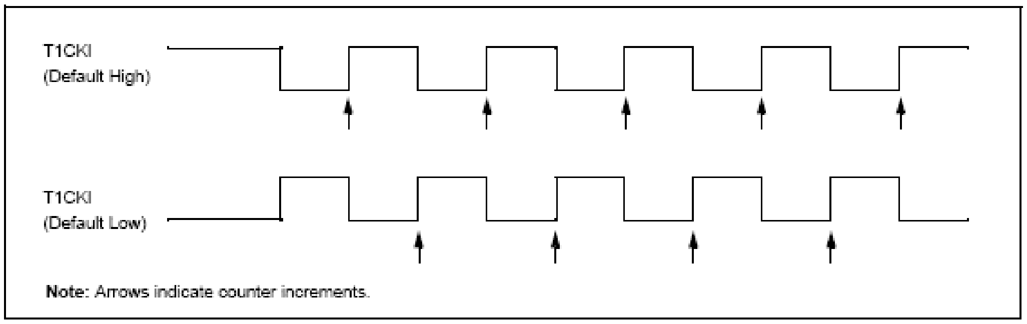


Figure 2.9 Timer1 Incrementing Edge

2.3.7.3 Timer1 Operation in Synchronized Counter Mode

Counter mode is selected by setting bit TMR1CS. In this mode, the timer increments on every rising edge of clock input on pin RC1/T1OSI/CCP2, when bit T1OSCEN is set, or on pin RC0/T1OSO/T1CKI, when bit T1OSCEN is cleared.

If $\overline{\text{T1SYNC}}$ is cleared, then the external clock input is synchronized with internal phase clocks. The synchronization is done after the prescaler stage. The prescaler stage is an asynchronous ripple-counter.

In this configuration, during SLEEP mode, Timer1 will not increment even if the external clock is present, since the synchronization circuit is shut-off. The prescaler, however, will continue to increment.

2.3.7.4 Timer1 Operation in Asynchronous Counter Mode

If control bit T1SYNC (T1CON<2>) is set, the external clock input is not synchronized. The timer continues to increment asynchronous to the internal phase clocks. The timer will continue to run during SLEEP and can generate an interrupt-on-overflow, which will wake-up the processor. However, special precautions in software are needed to read/write the timer.

In Asynchronous Counter mode, Timer1 cannot be used as a time-base for capture or compare operations.

Reading TMR1H or TMR1L while the timer is running from an external asynchronous clock, will guarantee a valid read (taken care of in hardware). However, the user should keep in mind that reading the 16-bit timer in two 8-bit values itself, poses certain problems, since the timer may overflow between the reads.

For writes, it is recommended that the user simply stop the timer and write the desired values. A write contention may occur by writing to the timer registers, while the register is incrementing. This may produce an unpredictable value in the timer register.

Reading the 16-bit value requires some care.

2.3.7.5 Timer1 Oscillator

A crystal oscillator circuit is built-in between pins T1OSI (input) and T1OSO (amplifier output). It is enabled by setting control bit T1OSCEN (T1CON<3>). The oscillator is a low power oscillator, rated up to 200 kHz. It will continue to run during SLEEP. It is primarily intended for use with a 32 kHz crystal.

The Timer1 oscillator is identical to the LP (Low Power Crystal) oscillator. The user must provide a software time delay to ensure proper oscillator start-up.

2.3.8 Timer2 Module

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 (FOSC/4) has a prescale option of 1:1, 1:4, or 1:16, selected by control bits T2CKPS1:T2CKPS0 (T2CON<1:0>).

The Timer2 module has an 8-bit period register, PR2. Timer2 increments from 00h until it matches PR2 and then resets to 00h on the next increment cycle. PR2 is a readable and writable register. The PR2 register is initialized to FFh 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 (latched in flag bit TMR2IF, (PIR1<1>)).

Timer2 can be shut-off by clearing control bit TMR2ON (T2CON<2>), to minimize power consumption.

2.3.8.1 Timer2 Prescaler and Postscaler

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
• any device RESET (POR, MCLR Reset, WDT Reset, or BOR)

TMR2 is not cleared when T2CON is written.

2.3.8.2 Timer2 Prescaler and Postscaler

The output of TMR2 (before the postscaler) is fed to the SSP module, which optionally uses it to generate shift clock.

2.3.9 Capture/Compare/PWM Modules

Each Capture/Compare/PWM (CCP) module contains a 16-bit register which can operate as a:

- 16-bit Capture register
- 16-bit Compare register
- PWM Master/Slave Duty Cycle register

Both the CCP1 and CCP2 modules are identical in operation, with the exception being the operation of the special event trigger.

CCP1 Module:

Capture/Compare/PWM Register1 (CCPR1) is comprised of two 8-bit registers: CCPR1L (low byte) and CCPR1H (high byte). The CCP1CON register controls the operation of CCP1. The special event trigger is generated by a compare match and will reset Timer1.

CCP2 Module:

Capture/Compare/PWM Register2 (CCPR2) is comprised of two 8-bit registers: CCPR2L (low byte) and CCPR2H (high byte). The CCP2CON register controls the operation of CCP2. The special event trigger is generated by a compare match and will reset Timer1 and start an A/D conversion (if the A/D module is enabled).

Table 2.4 CCP Mode – Timer Resources Required

CCP Mode	**Timer Resource**
Capture	Timer1
Compare	Timer1
PWM	Timer2

2.3.9.1 Capture Mode

In Capture mode, CCPR1H:CCPR1L captures the 16-bit value of the TMR1 register when an event occurs on pin RC2/CCP1. An event is defined as one of the following:

- Every falling edge
- Every rising edge
- Every 4th rising edge
- Every 16th rising edge

The type of event is configured by control bits CCP1M3:CCP1M0 (CCPxCON<3:0>). When a capture is made, the interrupt request flag bit CCP1IF (PIR1<2>) is set. The interrupt flag must be cleared in software. If another capture occurs before the value in register CCPR1 is read, the old captured value is overwritten by the new value.

2.3.9.2 Compare Mode

In Compare mode, the 16-bit CCPR1 register value is constantly compared against the TMR1 register pair value. When a match occurs, the RC2/CCP1 pin is:

- Driven high
- Driven low
- Remains unchanged

The action on the pin is based on the value of control bits CCP1M3:CCP1M0 (CCP1CON<3:0>). At the same time, interrupt flag bit CCP1IF is set.

2.3.9.3 PWM Mode

In Pulse Width Modulation mode, the CCPx pin produces up to a 10-bit resolution PWM output. Since the CCP1 pin is multiplexed with the PORTC data latch, the TRISC<2> bit must be cleared to make the CCP1 pin an output.

2.3.10 Master Synchronous Serial Port (MSSP) Module

The Master Synchronous Serial Port (MSSP) module is a serial interface, useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, shift registers, display drivers, A/D converters, etc. The MSSP module can operate in one of two modes:

• Serial Peripheral Interface (SPI)

• Inter-Integrated Circuit (I^2C)

2.3.10.1 SPI Mode

The SPI mode allows 8 bits of data to be synchronously transmitted and received simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

• Serial Data Out (SDO)

• Serial Data In (SDI)

• Serial Clock (SCK)

Additionally, a fourth pin may be used when in a Slave mode of operation:

• Slave Select ($\overline{\text{SS}}$)

When initializing the SPI, several options need to be specified. This is done by programming the appropriate control bits (SSPCON<5:0> and SSPSTAT<7:6>). These control bits allow the following to be specified:

• Master mode (SCK is the clock output)

• Slave mode (SCK is the clock input)

• Clock Polarity (Idle state of SCK)

• Data input sample phase (middle or end of data output time)

• Clock edge (output data on rising/falling edge of SCK)

• Clock Rate (Master mode only)

• Slave Select mode (Slave mode only)

To enable the serial port, MSSP Enable bit, SSPEN (SSPCON<5>) must be set. To reset or reconfigure SPI mode, clear bit SSPEN, re-initialize the SSPCON registers, and then set bit SSPEN. This configures the SDI, SDO, SCK and $\overline{\text{SS}}$ pins as serial port pins. For the pins to behave as the serial port function, some must have their data direction bits (in the TRIS register) appropriately programmed. That is:

• SDI is automatically controlled by the SPI module

• SDO must have TRISC<5> cleared

• SCK (Master mode) must have TRISC<3> cleared

• SCK (Slave mode) must have TRISC<3> set

• $\overline{\text{SS}}$ must have TRISA<5> set and register ADCON1 must be set in a way that pin RA5 is configured as a digital I/O

Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) register to the opposite value.

2.3.10.2 MMSP I^{2}C Operation

The MSSP module in I^{2}C mode, fully implements all master and slave functions (including general call support) and provides interrupts on START and STOP bits in hardware, to determine a free bus (multi-master function). The MSSP module implements the standard mode specifications, as well as 7-bit and 10-bit addressing.

A "glitch" filter is on the SCL and SDA pins when the pin is an input. This filter operates in both the 100 kHz and 400 kHz modes. In the 100 kHz mode, when these pins are an output, there is a slew rate control of the pin that is independent of device frequency.

Two pins are used for data transfer. These are the SCL pin, which is the clock, and the SDA pin, which is the data. The SDA and SCL pins are automatically configured when the I^{2}C mode is enabled. The SSP module functions are enabled by setting SSP Enable bit SSPEN (SSPCON<5>).

The MSSP module has six registers for I^{2}C operation. They are the:

• SSP Control Register (SSPCON)

• SSP Control Register2 (SSPCON2)

• SSP Status Register (SSPSTAT)

• Serial Receive/Transmit Buffer (SSPBUF)

• SSP Shift Register (SSPSR) - Not directly accessible

• SSP Address Register (SSPADD)

The SSPCON register allows control of the I^{2}C operation. Four mode selection bits (SSPCON<3:0>) allow one of the following I^{2}C modes to be selected:

• I^{2}C Slave mode (7-bit address)

• I^{2}C Slave mode (10-bit address)

• I^{2}C Master mode, clock = OSC/4 (SSPADD +1)

• I^2C firmware modes

Before selecting any I^2C mode, the SCL and SDA pins must be programmed to inputs by setting the appropriate TRIS bits. Selecting an I^2C mode by setting the SSPEN bit, enables the SCL and SDA pins to be used as the clock and data lines in I^2C mode. Pull-up resistors must be provided externally to the SCL and SDA pins for the proper operation of the I^2C module.

The CKE bit (SSPSTAT<6:7>) sets the levels of the SDA and SCL pins in either Master or Slave mode. When CKE = 1, the levels will conform to the SMBus specification. When CKE = 0, the levels will conform to the I^2C specification.

The SSPSTAT register gives the status of the data transfer. This information includes detection of a START (S) or STOP (P) bit, specifies if the received byte was data or address, if the next byte is the completion of 10-bit address, and if this will be a read or write data transfer.

SSPBUF is the register to which the transfer data is written to, or read from. The SSPSR register shifts the data in or out of the device. In receive operations, the SSPBUF and SSPSR create a doubled buffered receiver. This allows reception of the next byte to begin before reading the last byte of received data. When the complete byte is received, it is transferred to the SSPBUF register and flag bit SSPIF is set. If another complete byte is received before the SSPBUF register is read, a receiver overflow has occurred and bit SSPOV (SSPCON<6>) is set and the byte in the SSPSR is lost.

The SSPADD register holds the slave address. In 10-bit mode, the user needs to write the high byte of the address (1111 0 A9 A8 0). Following the high byte address match, the low byte of the address needs to be loaded (A7:A0).

2.3.10.3 Connection Considerations for I^2C Bus

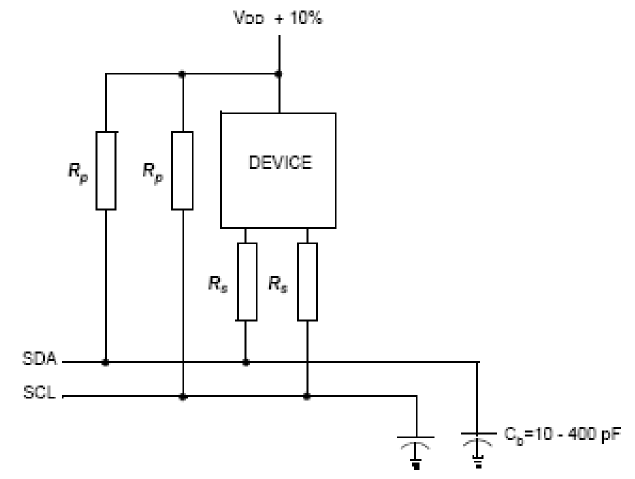
For standard-mode I^2C bus devices, the values of resistors Rp and Rs in Figure 2-10 depend on the following parameters:

- Supply voltage
- Bus capacitance
- Number of connected devices (input current + leakage current)

The supply voltage limits the minimum value of resistor Rp, due to the specified minimum sink current of 3 mA at V_{OL} max = 0.4V, for the specified output stages. For example, with a supply voltage of V_{DD} = 5V±10% and V_{OL} max = 0.4V at 3 mA, Rp min = (5.5-0.4)/0.003 = 1.7 kΩ. V_{DD} as a function of Rp is shown in Figure 2.10. The desired noise margin of $0.1V_{DD}$ for the low level limits the maximum value of Rs. Series resistors are optional and used to improve ESD susceptibility.

The bus capacitance is the total capacitance of wire, connections, and pins. This capacitance limits the maximum value of Rp due to the specified rise time (Figure 2.10).

The SMP bit is the slew rate control enabled bit. This bit is in the SSPSTAT register, and controls the slew rate of the I/O pins when in I^2C mode (master or slave).



Note: I^2C devices with input levels related to V_{DD} must have one common supply line to which the pull-up resistor is also connected.

Figure 2.10 Sample Device Configuration For I^2C Bus

2.3.11 Addressable Universal Synchronous Asynchronous Receiver Transmitter (USART)

The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the two serial I/O modules. (USART is also known as a Serial Communications Interface or SCI.) The USART can be configured as a full duplex asynchronous system that can communicate with peripheral devices such as CRT terminals and personal computers, or it can be configured as a half duplex synchronous system that can communicate with peripheral devices such as A/D or D/A integrated circuits, serial EEPROMs etc. The USART can be configured in the following modes:

• Asynchronous (full duplex)

• Synchronous - Master (half duplex)

• Synchronous - Slave (half duplex)

Bit SPEN (RCSTA<7>) and bits TRISC<7:6> have to be set in order to configure pins RC6/TX/CK and RC7/RX/DT as the Universal Synchronous Asynchronous Receiver Transmitter.

The USART module also has a multi-processor communication capability using 9-bit address detection.

2.3.12 Analog to Digital Converter (A/D) Module

The Analog-to-Digital (A/D) Converter module has five inputs for the 28-pin devices and eight for the other devices.

The analog input charges a sample and hold capacitor. The output of the 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 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, RA2, or RA3.

The A/D converter has a unique feature of being able to operate while the device is in SLEEP mode. To operate in SLEEP, the A/D clock must be derived from the A/D's internal RC oscillator.

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 Register0 (ADCON0)

• A/D Control Register1 (ADCON1)

The ADCON0 register controls the operation of the A/D module. The ADCON1 register configures the functions of the port pins. The port pins can be configured as analog inputs (RA3 can also be the voltage reference), or as digital I/O.

2.3.13 Special Features of the CPU

PIC16F877 has a host of features intended to maximize system reliability, minimize cost through elimination of external components, provide power saving operating modes and offer code protection. These are:

- Oscillator Selection
- RESET
  - Power-on Reset (POR)
  - Power-up Timer (PWRT)
  - Oscillator Start-up Timer (OST)
  - Brown-out Reset (BOR)
- Interrupts
- Watchdog Timer (WDT)
- SLEEP
- Code Protection
- ID Locations
- In-Circuit Serial Programming
- Low Voltage In-Circuit Serial Programming
- In-Circuit Debugger

PIC16F877 has a Watchdog Timer, which can be shut-off only through configuration bits. It runs off its own RC oscillator for added reliability.

There are two timers that offer necessary delays on power-up. One is the Oscillator Start-up Timer (OST), intended to keep the chip in RESET until the crystal oscillator is stable. The other is the Power-up Timer (PWRT), which provides a fixed delay of 72 ms (nominal) on power-up only. It is designed to keep the part in RESET while the power supply stabilizes. With these two timers on-chip, most applications need no external RESET circuitry.

SLEEP mode is designed to offer a very low current Power-down mode. The user can wake-up from SLEEP through external RESET, Watchdog Timer Wake-up, or through an interrupt.

Several oscillator options are also made available to allow the part to fit the application. The RC oscillator option saves system cost while the LP crystal option saves power. A set of configuration bits is used to select various options.

2.3.13.1 Configuration Bits

The configuration bits can be programmed (read as '0'), or left unprogrammed (read as '1'), to select various device configurations. The erased or unprogrammed value of the configuration word is 3FFFh. These bits are mapped in program memory location 2007h.

It is important to note that address 2007h is beyond the user program memory space, which can be accessed only during programming.

2.3.13.2 Oscillator Configurations

- Oscillator Types

The PIC16F877 can be operated in four different oscillator modes. The user can program two configuration bits (FOSC1 and FOSC0) to select one of these four modes:

• LP Low Power Crystal

• XT Crystal/Resonator

• HS High Speed Crystal/Resonator

• RC Resistor/Capacitor

- Crystal Oscillator/Ceramic Resonators

In XT, LP or HS modes, a crystal or ceramic resonator is connected to the OSC1/CLKIN and OSC2/CLKOUT pins to establish oscillation. The PIC16F877 oscillator design requires the use of a parallel cut crystal. Use of a series cut crystal may give a frequency out of the crystal manufacturers specifications.

When in XT, LP or HS modes, the device can have an external clock source to drive the OSC1/CLKIN pin.

- RC Oscillator

For timing insensitive applications, the "RC" device option offers additional cost savings. The RC oscillator frequency is a function of the supply voltage, the resistor (REXT) and capacitor (CEXT) values, and the operating temperature. In addition to this, the oscillator frequency will vary from unit to unit due to normal process parameter variation. Furthermore, the difference in lead frame capacitance between package types will also affect the oscillation frequency, especially for low CEXT values. The user also needs to take into account variation due to tolerance of external R and C components used. Figure 2.11 shows how the R/C combination is connected to the PIC16F877.

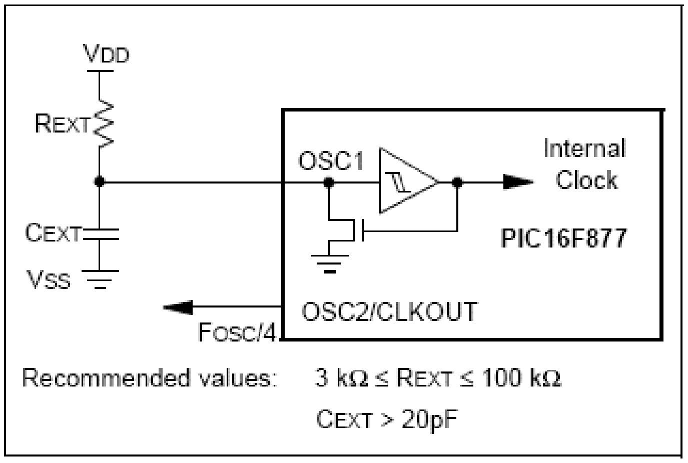


Figure 2.11 RC Oscillator Mode

2.3.13.3 Reset

The PIC16F877 differentiates between various kinds of RESET:

• Power-on Reset (POR)
• $\overline{\text{MCLR}}$ Reset during normal operation
• $\overline{\text{MCLR}}$ Reset during SLEEP
• WDT Reset (during normal operation)
• WDT Wake-up (during SLEEP)
• Brown-out Reset (BOR)

Some registers are not affected in any RESET condition. Their status is unknown on POR and unchanged in any other RESET. Most other registers are reset to a “RESET state” on Power-on Reset (POR), on the $\overline{\text{MCLR}}$ and WDT Reset, on $\overline{\text{MCLR}}$ Reset during SLEEP, and Brown-out Reset (BOR). They are not affected by a WDT Wake-up, which is viewed as the resumption of normal operation. The TO and PD bits are set or cleared differently in different RESET situations. These bits are used in software to determine the nature of the RESET.

There is a $\overline{\text{MCLR}}$ noise filter in the $\overline{\text{MCLR}}$ Reset path. The filter will detect and ignore small pulses.

It should be noted that a WDT Reset does not drive $\overline{\text{MCLR}}$ pin low.

2.3.13.4 Power-On Reset (POR)

A Power-on Reset pulse is generated on-chip when V_{DD} rise is detected (in the range of 1.2V - 1.7V). To take advantage of the POR, tie the $\overline{\text{MCLR}}$ pin directly (or through a resistor) to V_{DD}. This will eliminate external RC components usually needed to create a Power-on Reset. A maximum rise time for V_{DD} is specified.

When the device starts normal operation (exits the RESET condition), device operating parameters (voltage, frequency, temperature,...) must be met to ensure operation. If these conditions are not met, the device must be held in RESET until the operating conditions are met. Brown-out Reset may be used to meet the start-up conditions.

2.3.13.5 Power-Up Timer (PWRT)

The Power-up Timer provides a fixed 72 ms nominal time-out on power-up only from the POR. The Power-up Timer operates on an internal RC oscillator. The chip is kept in RESET as long as the PWRT is active.

The PWRT’s time delay allows V_{DD} to rise to an acceptable level. A configuration bit is provided to enable/disable the PWRT. The power-up time delay will vary from chip to chip due to V_{DD}, temperature and process variation.

2.3.13.6 Oscillator Start-Up Timer (OST)

The Oscillator Start-up Timer (OST) provides a delay of 1024 oscillator cycles (from OSC1 input) after the PWRT delay is over (if PWRT is enabled). This helps to ensure that the crystal oscillator or resonator has started and stabilized.

The OST time-out is invoked only for XT, LP and HS modes and only on Power-on Reset or Wake-up from SLEEP.

2.3.13.7 Brown-Out Reset (BOR)

The configuration bit, BODEN, can enable or disable the Brown-out Reset circuit. If V_{DD} falls below V_{BOR} (about 4V) for longer than T_{BOR} (about 100µS), the brown-out situation will reset the device. If V_{DD} falls below V_{BOR} for less than T_{BOR}, a RESET may not occur.

Once the brown-out occurs, the device will remain in Brown-out Reset until V_{DD} rises above V_{BOR}. The Power-up Timer then keeps the device in RESET for T_{PWRT} (about 72mS). If V_{DD} should fall below V_{BOR} during T_{PWRT}, the Brown-out Reset process will restart when V_{DD} rises above V_{BOR} with the Power-up Timer Reset. The Power-up Timer is always enabled when the Brown-out Reset circuit is enabled, regardless of the state of the PWRT configuration bit.

2.3.13.8 Time-Out Sequence

On power-up, the time-out sequence is as follows: The PWRT delay starts (if enabled) when a POR Reset occurs. Then OST starts counting 1024 oscillator cycles when PWRT ends (LP, XT, HS). When the OST ends, the device comes out of RESET.

If $\overline{\text{MCLR}}$ is kept low long enough, the time-outs will expire. Bringing $\overline{\text{MCLR}}$ high will begin execution immediately. This is useful for testing purposes or to synchronize more than one PIC16F877 operating in parallel.

2.3.13.9 Power Control/Status Register (PCON)

The Power Control/Status Register, PCON, has up to two bits depending upon the device.

Bit0 is Brown-out Reset Status bit, BOR. Bit BOR is unknown on a Power-on Reset. It must then be set by the user and checked on subsequent RESETS to see if bit BOR cleared, indicating a BOR occurred. When the Brown-out Reset is disabled, the state of the BOR bit is unpredictable and is, therefore, not valid at any time.

Bit1 is POR (Power-on Reset Status bit). It is cleared on a Power-on Reset and unaffected otherwise. The user must set this bit following a Power-on Reset.

2.3.13.10 Interrupts

The PIC16F877 has up to 14 sources of interrupt. The interrupt control register (INTCON) records individual interrupt requests in flag bits. It also has individual and global interrupt enable bits.

A global interrupt enable bit, GIE (INTCON<7>) enables (if set) all unmasked interrupts, or disables (if cleared) all interrupts. When bit GIE is enabled, and an interrupt's flag bit and mask bit are set, the interrupt will vector immediately. Individual interrupts can be disabled through their corresponding enable bits in various registers. Individual interrupt bits are set, regardless of the status of the GIE bit. The GIE bit is cleared on RESET.

The "return from interrupt" instruction, RETFIE, exits the interrupt routine, as well as sets the GIE bit, which re-enables interrupts.

The RB0/INT pin interrupt, the RB port change interrupt, and the TMR0 overflow interrupt flags are contained in the INTCON register.

The peripheral interrupt flags are contained in the special function registers, PIR1 and PIR2. The corresponding interrupt enable bits are contained in special function registers, PIE1 and PIE2, and the peripheral interrupt enable bit is contained in special function register INTCON.

When an interrupt is responded to, the GIE bit is cleared to disable any further interrupt, the return address is pushed onto the stack and the PC is loaded with 0004h. Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid recursive interrupts.

For external interrupt events, such as the INT pin or PORTB change interrupt, the interrupt latency will be three or four instruction cycles. The exact latency depends when the interrupt event occurs. The latency is the same for one or two-cycle instructions. Individual interrupt flag bits are set, regardless of the status of their corresponding mask bit, PEIE bit, or GIE bit.

- INT Interrupt

External interrupt on the RB0/INT pin is edge triggered, either rising, if bit INTEDG (OPTION_REG<6>) is set, or falling, if the INTEDG bit is clear. When a valid edge appears on the RB0/INT pin, flag bit INTF (INTCON<1>) is set. This interrupt can be disabled by clearing enable bit INTE (INTCON<4>). Flag bit INTF must be cleared in software in the Interrupt Service Routine before re-enabling this interrupt. The INT interrupt can wake-up the processor from SLEEP, if bit INTE was set prior to going into SLEEP. The status of global interrupt enable bit, GIE, decides whether or not the processor branches to the interrupt vector following wake-up.

- TMR0 Interrupt

An overflow (FFh → 00h) in the TMR0 register will set flag bit T0IF (INTCON<2>). The interrupt can be enabled/disabled by setting/clearing enable bit T0IE (INTCON<5>).

- PortB INTCON Change

An input change on PORTB<7:4> sets flag bit RBIF (INTCON<0>). The interrupt can be enabled/disabled by setting/clearing enable bit RBIE (INTCON<4>).

2.3.13.11 Context Saving During Interrupts

During an interrupt, only the return PC value is saved on the stack. Typically, users may wish to save key registers during an interrupt, (i.e., W register and STATUS register). This will have to be implemented in software.

Since the upper 16 bytes of each bank are common in the PIC16F877, temporary holding registers W_TEMP, STATUS_TEMP, and PCLATH_TEMP should be placed in here. These 16 locations don’t require banking and therefore, make it easier for context save and restore.

2.3.13.12 Watchdog Timer (WDT)

The Watchdog Timer is a free running on-chip RC oscillator which does not require any external components. This RC oscillator is separate from the RC oscillator of the OSC1/CLKIN pin. That means that the WDT will run, even if the clock on the OSC1/CLKIN and OSC2/CLKOUT pins of the device has been stopped, for example, by execution of a SLEEP instruction.

During normal operation, a WDT time-out generates a device RESET (Watchdog Timer Reset). If the device is in SLEEP mode, a WDT time-out causes the device to wake-up and continue with normal operation (Watchdog Timer Wake-up). The $\overline{\text{TO}}$ bit in the STATUS register will be cleared upon a Watchdog Timer time-out.

The WDT can be permanently disabled by clearing configuration bit WDTE. WDT time-out period values for the WDT prescaler (actually a postscaler, but shared with the Timer0 prescaler) may be assigned using the OPTION_REG register.

2.3.13.13 Power-Down Mode (SLEEP)

Power-down mode is entered by executing a SLEEP instruction. If enabled, the Watchdog Timer will be cleared but keeps running, the PD bit (STATUS<3>) is cleared, the TO (STATUS<4>) bit is set, and the oscillator driver is turned off. The I/O ports maintain the status they had before the SLEEP instruction was executed (driving high, low, or hi-impedance).

For lowest current consumption in this mode, place all I/O pins at either V_{DD} or V_{SS}, ensure no external circuitry is drawing current from the I/O pin, power-down the A/D and disable external clocks. Pull all I/O pins that are hi-impedance inputs, high or low externally, to avoid switching currents caused by floating inputs. The T0CKI input should also be at V_{DD} or V_{SS} for lowest current consumption. The contribution from on-chip pull-ups on PORTB should also be considered.

The $\overline{\text{MCLR}}$ pin must be at a logic high level (V_{IHMC}).

- Wake-Up from SLEEP

The device can wake-up from SLEEP through one of the following events:

1. External RESET input on $\overline{\text{MCLR}}$ pin.

2. Watchdog Timer Wake-up (if WDT was enabled).

3. Interrupt from INT pin, RB port change or peripheral interrupt.

External $\overline{\text{MCLR}}$ Reset will cause a device RESET. All other events are considered a continuation of program execution and cause a "wake-up". The $\overline{\text{TO}}$ and $\overline{\text{PD}}$ bits in the STATUS register can be used to determine the cause of device RESET. The $\overline{\text{PD}}$ bit, which is set on power-up, is cleared when SLEEP is invoked. The $\overline{\text{TO}}$ bit is cleared if a WDT time-out occurred and caused wake-up.

The following peripheral interrupts can wake the device from SLEEP:

1. PSP read or write.
2. TMR1 interrupt. Timer1 must be operating as an asynchronous counter.
3. CCP Capture mode interrupt.
4. Special event trigger (Timer1 in Asynchronous mode using an external clock).
5. SSP (START/STOP) bit detect interrupt.
6. SSP transmit or receive in Slave mode (SPI/I^2C).
7. USART RX or TX (Synchronous Slave mode).
8. A/D conversion (when A/D clock source is RC).
9. EEPROM write operation completion

Other peripherals cannot generate interrupts since during SLEEP, no on-chip clocks are present.

When the SLEEP instruction is being executed, the next instruction (PC + 1) is pre-fetched. For the device to wake-up through an interrupt event, the corresponding interrupt enable bit must be set (enabled). Wake-up is regardless of the state of the GIE bit. If the GIE bit is clear (disabled), the device continues execution at the instruction after the SLEEP instruction. If the GIE bit is set (enabled), the device executes the instruction after the SLEEP instruction and then branches to the interrupt address (0004h). In cases where the execution of the instruction following SLEEP is not desirable, the user should have a NOP after the SLEEP instruction.

- Wake-Up Using Interrupts

When global interrupts are disabled (GIE cleared) and any interrupt source has both its interrupt enable bit and interrupt flag bit set, one of the following will occur:

• If the interrupt occurs before the execution of a SLEEP instruction, the SLEEP instruction will complete as a NOP. Therefore, the WDT and WDT postscaler will not be cleared, the $\overline{\text{TO}}$ bit will not be set and $\overline{\text{PD}}$ bits will not be cleared.

• If the interrupt occurs during or after the execution of a SLEEP instruction, the device will immediately wake-up from SLEEP. The SLEEP instruction will be completely executed before the wake-up. Therefore, the WDT and WDT postscaler will be cleared, the $\overline{\text{TO}}$ bit will be set and the $\overline{\text{PD}}$ bit will be cleared.

Even if the flag bits were checked before executing a SLEEP instruction, it may be possible for flag bits to become set before the SLEEP instruction completes. To determine whether a SLEEP instruction executed, test the $\overline{\text{PD}}$ bit. If the $\overline{\text{PD}}$ bit is set, the SLEEP instruction was executed as a NOP.

To ensure that the WDT is cleared, a CLRWDT instruction should be executed before a SLEEP instruction.

2.3.13.14 In-Circuit Debugger

When the DEBUG bit in the configuration word is programmed to a ’0’, the In-Circuit Debugger functionality is enabled. This function allows simple debugging functions when used with MPLAB ICD. When the microcontroller has this feature enabled, some of the resources are not available for general use. Table 2.5 shows which features are consumed by the background debugger.

Table 2.5 Debugger Resources

I/O pins	RB6, RB7
Stack	1 level
Program Memory	Address 0000h must be NOP
	Last 100h words
Data Memory	0x070 (0x0F0, 0x170, 0x1F0) 0x1EB - 0x1EF

To use the In-Circuit Debugger function of the microcontroller, the design must implement In-Circuit Serial Programming connections to $\overline{\text{MCLR}}$/VPP, VDD, GND, RB7 and RB6. This will interface to the In-Circuit Debugger module.

2.3.13.15 Program Verification/Code Protection

If the code protection bit(s) have not been programmed, the on-chip program memory can be read out for verification purposes.

2.3.13.16 ID Locations

Four memory locations (2000h - 2003h) are designated as ID locations, where the user can store checksum or other code identification numbers. These locations are not accessible during normal execution, but are readable and writable during program/verify. It is recommended that only the 4 Least Significant bits of the ID location are used.

2.3.13.17 In-Circuit Serial Programming

PIC16F877 microcontroller can be serially programmed while in the end application circuit. This is simply done with two lines for clock and data and three other lines for power, ground, and the programming voltage. This allows users to manufacture boards with unprogrammed devices, and then program the microcontroller. This also allows the most recent firmware, or a custom firmware to be programmed.

When using ICSP, the part must be supplied at 4.5V to 5.5V, if a bulk erase will be executed. This includes reprogramming of the code protect, both from an on-state to off-state. For all other cases of ICSP, the part may be programmed at the normal operating voltages. This means calibration values, unique user IDs, or user code can be reprogrammed or added.

2.3.13.18 Low Voltage ICSP Programming

The LVP bit of the configuration word enables low voltage ICSP programming. This mode allows the microcontroller to be programmed via ICSP using a V_{DD} source in the operating voltage range. This only means that V_{PP} does not have to be brought to V_{IHH}, but can instead be left at the normal operating voltage. In this mode, the RB3/PGM pin is dedicated to the programming function and ceases to be a general purpose I/O pin. During programming, V_{DD} is applied to the $\overline{MCLR}$ pin.

To enter Programming mode, V_{DD} must be applied to the RB3/PGM, provided the LVP bit is set. The LVP bit defaults to on (‘1’).

If Low Voltage Programming mode is not used, the LVP bit can be programmed to a '0' and RB3/PGM becomes a digital I/O pin. However, the LVP bit may only be programmed when programming is entered with V_{IHH} on $\overline{MCLR}$. The LVP bit can only be charged when using high voltage on $\overline{MCLR}$.

It should be noted, that once the LVP bit is programmed to 0, only the High Voltage Programming mode is available and only High Voltage Programming mode can be used to program the device.

When using low voltage ICSP, the part must be supplied at 4.5V to 5.5V, if a bulk erase will be executed. This includes reprogramming of the code protect bits from an on-state to off-state. For all other cases of low voltage ICSP, the part may be programmed at the normal operating voltage. This means calibration values, unique user IDs, or user code can be reprogrammed or added.

CHAPTER 3

PIC PROGRAMMING

3.1 INTRODUCTION

To write a program for a PIC Microcontroller and to load it, there must be some software and hardware:

• an ASCII editor

• an assembler

• a simulator

• a programmer

• a software for programmer

The first step for programming is the writing of the program to use a text editor. The alternatives are:

- EDIT in DOS
- NOTEPAD in WINDOWS
- MPLAB Editor

After writing the program, it must be saved in .asm type of file.

In the second stage, this .asm file is converted to hex file which machine can understand. At this point a simulator can be used for simulate the code.

The last step is to send the hex file to microcontroller by a device programmer and its software.

3.2 MPLAB IDE

MPLAB IDE is a Windows-based Integrated Development Environment (IDE) for the PICmicro microcontroller (MCU) families. MPLAB IDE is used for writing, debugging, and optimizing PICmicro MCU applications for firmware product designs. MPLAB IDE includes a text editor for creating assembly language source code and an assembler to convert the source code into a form which can be programmed into the PIC microcontroller, simulator, and project manager.

The MPLAB IDE features the following:

• MPLAB Project Manager

  Organizes the different files under one 'project'

  Interfaces between the editor, assembler, linker, and simulator

• MPLAB-SIM Software Simulator

  Features debug capabilities: unlimited breakpoints, trace, examine/modify registers, watch variables and time-stamp

  Simulates core functions and peripherals

• MPLAB Editor

  Programmer's editor to write and edit source files

• MPASM Universal Assembler

  Has macro capabilities, conditional assembly

  Builds the HEX file (machine language)

The most important component of the IDE is the integrated software simulator, which allows a designer to trace through their assembly code and watch the registers, RAM, ROM, and I/O ports.

3.3 MPASM

MPASM is a DOS or Windows-based PC application that provides a platform for developing assembly language code for PICmicro microcontroller (MCU) families. Generically, MPASM will refer to the entire development platform including the macro assembler and utility functions.

MPASM provides a solution for developing assembly code for all of Microchip's 12-bit, 14-bit, 16-bit, and enhanced 16-bit core PICmicro microcontrollers. Notable features include:

• All PICmicro MCU Instruction Sets

• Command Line Interface

• Command Shell Interfaces

• Directive Language

• Flexible Macro Language

• MPLAB Compatibility

3.3.1 Overview of Assembler

MPASM can be used in two ways:

• To generate absolute code that can be executed directly by a microcontroller.

• To generate object code that can be linked with other separately assembled or compiled modules.

Absolute code is the default output from MPASM. This process is shown in Figure 3.1.

When a source file is assembled in this manner, all values used in the source file must be defined within that source file, or in files that have been explicitly included. If assembly proceeds without errors, a HEX file will be generated, containing the executable machine code for the target device. This file can then be used in conjunction with a device programmer to program the microcontroller.

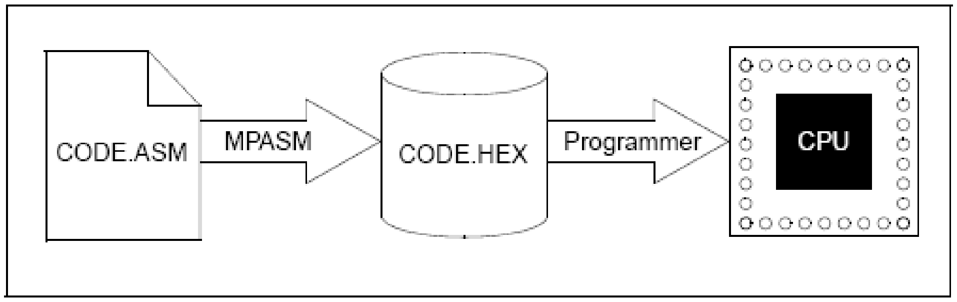


Figure 3.1 Generating Absolute Code

MPASM also has the ability to generate an object module that can be linked with other modules using Microchip's MPLINK linker to form the final executable code. This method is very useful for creating reusable modules that do not have to be retested each time they are used. Related modules can also be grouped and stored together in a library using Microchip's MPLIB Librarian. Required libraries can be specified at link time, and only the routines that are needed will be included in the final executable.

A visual representation of this process is shown in Figure 3.2 and Figure 3.3.

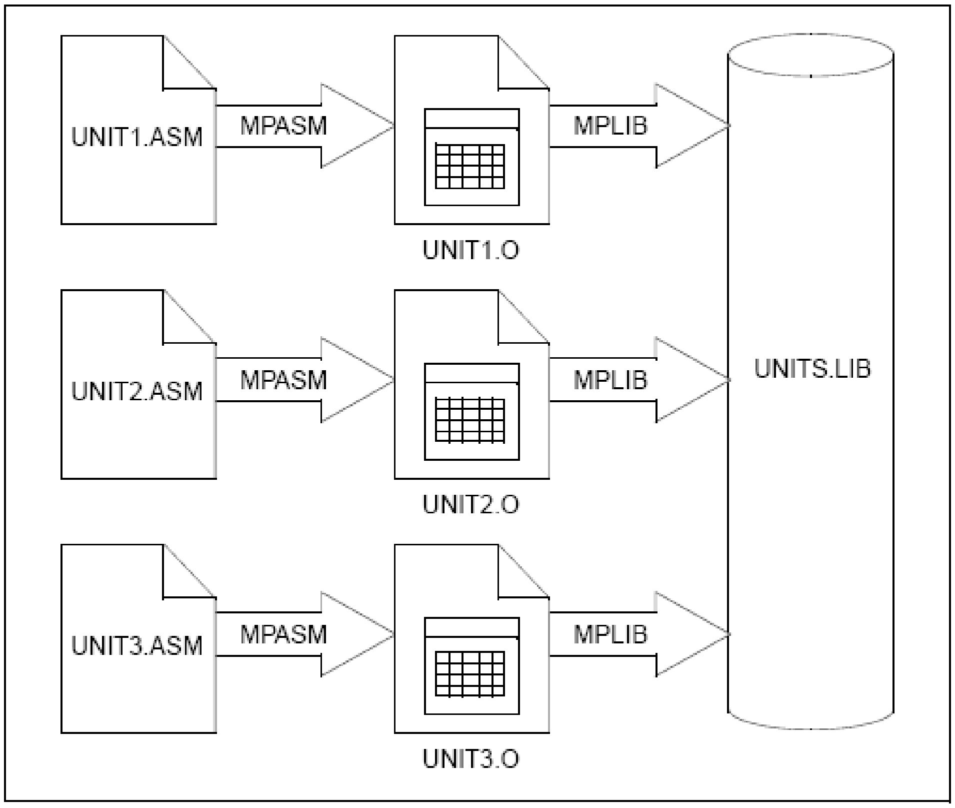


Figure 3.2 Creating a Reusable Object Library

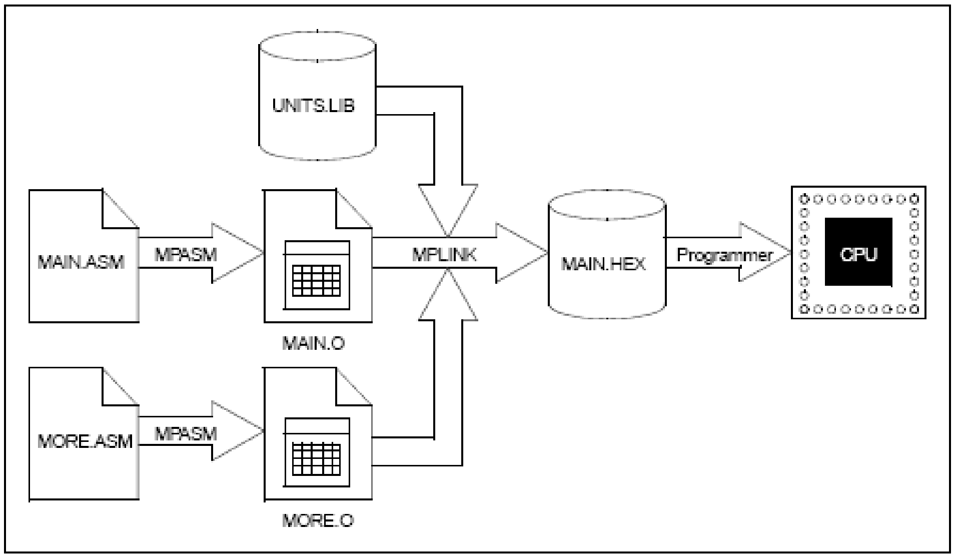


Figure 3.3 Generating Executable Code from Object Modules

3.3.2 Assembler Input/Output Files

These are the default file extensions used by MPASM and the associated utility functions.

Table 3.1 MPASM Default Extensions

Extension	Purpose
`.ASM`	Default source file extension input to MPASM: `<source_name>.ASM`
`.LST`	Default output extension for listing files generated by MPASM: `<source_name>.LST`
`.ERR`	Output extension from MPASM for error files: `<source_name>.ERR`
`.HEX`	Output extension from MPASM for hex files (see Appendix A), `<source_name>.HEX`
`.HXL/` `.HXH`	Output extensions from MPASM for separate low byte and high byte hex files: `<source_name>.HXL, <source_name>.HXH`
`.COD`	Output extension for the symbol and debug file. This file may be output from MPASM or MPLINK: `<source_name>.COD`
`.O`	Output extension from MPASM for object files: `<source_name>.O`

3.3.2.1 Source Code Format (.ASM)

The source code file may be created using any ASCII text file editor. It should conform to the following basic guidelines.

Each line of the source file may contain up to four types of information:

• labels

• mnemonics

• operands

• comments

The order and position of these are important. Labels must start in column one. Mnemonics may start in column two or beyond. Operands follow the mnemonic. Comments may follow the operands, mnemonics or labels, and can start in any column. The maximum column width is 255 characters.

Whitespace or a colon must separate the label and the mnemonic, and the mnemonic and the operand(s). Multiple operands must be separated by a comma.

- Labels

A label must start in column 1. It may be followed by a colon (:), space, tab or the end of line.

Labels must begin with an alpha character or an under bar (_) and may contain alphanumeric characters, the under bar and the question mark.

Labels may be up to 32 characters long. By default they are case sensitive, but case sensitivity may be overridden by a command line option. If a colon is used when defining a label, it is treated as a label operator and not part of the label itself.

- Mnemonics

Assembler instruction mnemonics, assembler directives and macro calls must begin in column two or greater. If there is a label on the same line, instructions must be separated from that label by a colon, or by one or more spaces or tabs.

- Operands

Operands must be separated from mnemonics by one or more spaces, or tabs. Multiple operands must be separated by commas.

- Comments

MPASM treats anything after a semicolon as a comment. All characters following the semicolon are ignored through the end of the line. String constants containing a semicolon are allowed and are not confused with comments.

3.3.2.2 Listing File Format (.LST)

The listing file format produced by MPASM is straight forward:

The product name and version, the assembly date and time, and the page number appear at the top of every page.

The first column of numbers contains the base address in memory where the code will be placed. The second column displays the 32-bit value of any symbols created with the SET, EQU, VARIABLE, CONSTANT, or CBLOCK directives. The third column is reserved for the machine instruction. This is the code that will be executed by the PICmicro MCU. The fourth column lists the associated source file line number for this line. The remainder of the line is reserved for the source code line that generated the machine code.

Errors, warnings, and messages are embedded between the source lines, and pertain to the following source line.

The symbol table lists all symbols defined in the program. The memory usage map gives a graphical representation of memory usage. 'X' marks a used location and '-' marks memory that is not used by this object. The memory map is not printed if an object file is generated.

3.3.2.3 Error File Format (.ERR)

MPASM by default generates an error file. This file can be useful when debugging the code. The MPLAB Source Level Debugger will automatically open this file in the case of an error. The format of the messages in the error file is:

<type>[<number>] <file> <line> <description>

For example:

Error[113] C:\PROG.ASM 7 : Symbol not previously defined (start)

3.3.2.4 Hex File Formats (.HEX, .HXL, .HXH)

MPASM is capable of producing different hex file formats.

3.3.2.5 Symbol and Debug File Format (.COD)

When MPASM is used to generate absolute code, it produces a COD file for use in MPLAB debugging of code.

3.3.2.6 Object File Format (.O)

Object files are the relocatable code produced from source files.

3.3.3 MPLAB Projects and MPASM

MPLAB projects are composed of nodes (Figure 3.4). These represent files used by a generated project.

• Target Node – Final Output
  - HEX File
• Project Nodes – Components
  - Assembly Source Files

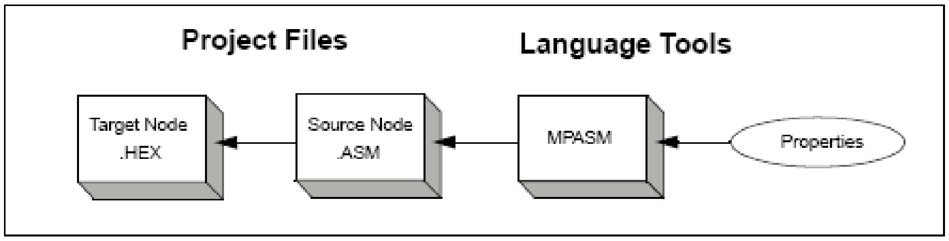


Figure 3.4 Project Relationships – MPASM

Projects are used to apply language tools, such as assemblers, compilers and linkers, to source files in order to make executable (.HEX) files. This diagram shows the relationship between the final .HEX file and the component .ASM files used to create it.

3.3.4 Directive Language

Directives are assembler commands that appear in the source code but are not translated directly into opcodes. They are used to control the assembler; its input, output, and data allocation.

There are five basic types of directives provided by MPASM:

- Control Directives – Control directives permit sections of conditionally assembled code.
- Data Directives – Data Directives are those that control the allocation of memory and provide a way to refer to data items symbolically, that is, by meaningful names.
- Listing Directives – Listing Directives are those directives that control the MPASM listing file format. They allow the specification of titles, pagination, and other listing control.
- Macro Directives – These directives control the execution and data allocation within macro body definitions.
- Object File Directives – These directives are used only when creating an object file.

3.4 IC-PROG

3.4.1 General

IC-Prog is a Windows based software to control a development programmer for PIC microcontrollers. It is used to send the .HEX file to the MCU.

In order for this software to operate a programmer has to be attached to the computer and hardware and software have to be set up appropriately.

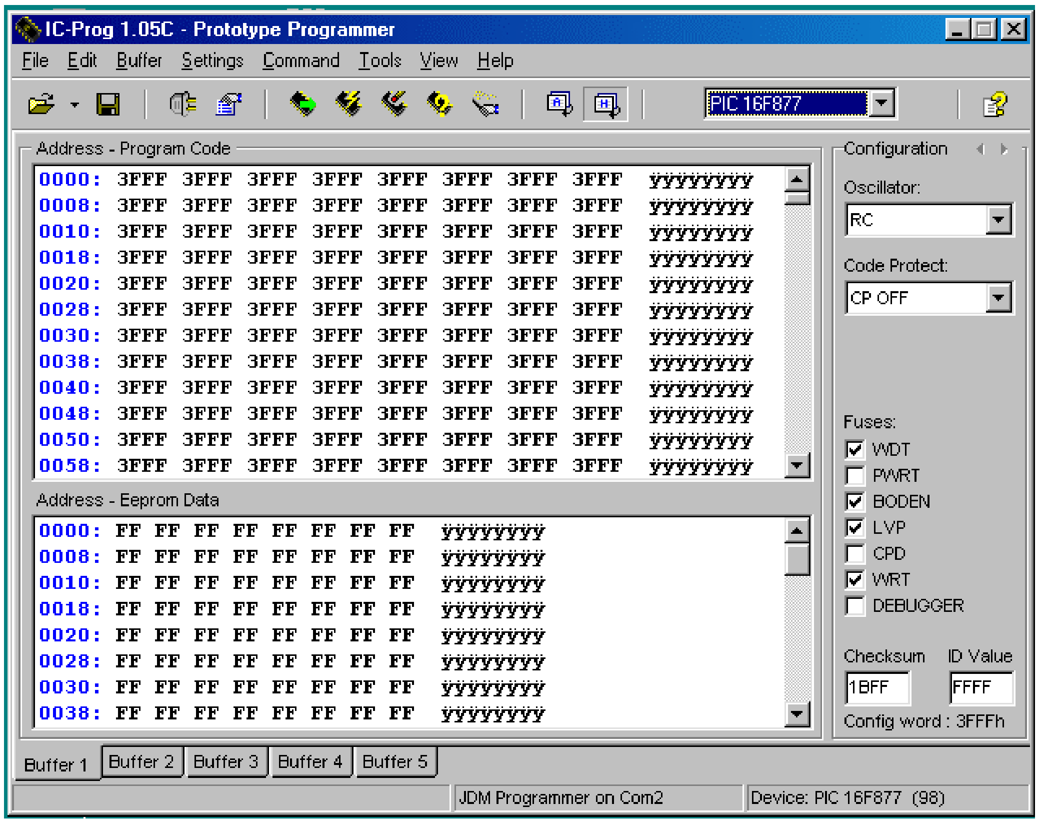
The main area of IC-Prog shows the information that needs to be programmed into the specified device. All devices at least have a Code area where information can be stored. Devices like EEPROMs only have this Code area.

Other devices like most microcontrollers have additional storage areas, like the Data area. Normally the Code area contains code that will be executed by the microcontroller and the data area contains some fixed data like tables for calculating etc.

Most microcontrollers, like PIC, have a Configuration area as well. This configuration information will configure the microcontroller with some initial settings at bootup. This configuration information is different and unique for each microcontroller.

3.4.2 Main View

IC-Prog intends to display the information in all areas in such a way, that it is easy to see what the information means. The Code and Data area will display the information in hexadecimal values and character values: The left part of the Code and Data area contains the address on which the information is stored. The middle part contains the information in hexadecimal values and the right part contains the SAME information, but displayed in character values.


IC-Prog 1.05C - Prototype Programmer
File Edit Buffer Settings Command Tools View Help
PIC 16F877
Buffer 1
Buffer 2
Buffer 3
Buffer 4
Buffer 5

Address - Program Code

```
0000: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0008: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0010: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0018: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0020: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0028: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0030: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0038: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0040: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0048: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0050: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
0058: 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF 3FFF  ÿÿÿÿÿÿÿÿ
```

Address - Eeprom Data

```
0000: FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
0008: FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
0010: FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
0018: FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
0020: FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
0028: FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
0030: FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
0038: FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
```

Configuration

Oscillator: RC

Code Protect: CP OFF

Fuses:
- [✓] WDT
- [ ] PWRT
- [✓] BODEN
- [✓] LVP
- [ ] CPD
- [✓] WRT
- [ ] DEBUGGER

Checksum: 1BFF   ID Value: FFFF

Config word : 3FFFh

JDM Programmer on Com2   Device: PIC 16F877 (98)

Figure 3.5 Main View of IC-Prog

3.4.3 Code Area

Each row in the Code area will display 8 words. So each row the address will be incremented with 8. A word is normally 16 bits wide so IC-Prog will show 0000 to FFFF as a hexadecimal value. Some devices only have 14 bit, 12 bit or 8 bit words, so the maximum hexadecimal value will be 3FFF, 0FFF or 00FF, but IC-prog will always display a hexadecimal value using 4 digits. The character values only use the lower 8 bits of the 16 bit data word, because the standard character range only runs from 0 to 255 (8 bits). (Figure 3.5)

3.4.4 Data Area

Each row in the Data area will also display 8 words, but these words are by default always 8 bits. They will always be displayed as 2 digit hexadecimal words with a minimum of 00 and a maximum of FF. (Figure 3.5)

3.4.5 Configuration Area

The configuration area of device will be shown with drop down combo boxes and checkboxes. The user can select the desired configuration, and IC-Prog will calculate the according configuration word. This calculated configuration word is also displayed at the bottom of the configuration area.

These configuration elements are device specific, and so this configuration area will look different for each selected device.

Often a specific configuration element can only be enabled or disabled. This results in IC-Prog calculation a zero for a specific bit in the configuration word or a one. This can be the other way around if a specific configuration element is inverted inside the device. IC-Prog automatically inverts a configuration element if this is needed. To enable a configuration element, a mark is placed in the checkbox. To disable a configuration element, the mark is removed from a checkbox.

Some devices have so much configuration elements; the configuration area cannot contain them all. Then IC-Prog will create a second (or a third) configuration area, which user can select by using the arrows in the upper right corner of this area. If a device has only 1 configuration area, then these arrows will be disabled.

When user selects a device from the menu, IC-Prog will automatically determine the device type and adjust the main view. All devices have a Code area at least, so this part of the main view is always visible. Some devices (like EEPROMs) do not have a Data area, so IC-Prog will not show this area. Instead it will extend the Code area to get a full view. Some devices also don't have configuration information, and IC-Prog will leave it blank. Only the checksum value will be visible.

3.5 PROPIC II

Propic II is one of the hardware alternatives that is used to transfer the code to MCU. It operates with a software like IC-Prog.

CHAPTER 4

THE CONTROL UNIT HARDWARE AND SOFTWARE

4.1 GENERAL

In this thesis study, a multi-purpose programmable control unit is designed and implemented. The Microchip's PIC 16F877 is used as a microcontroller and the program of the controller is written in Assembly language.

MPLAB Editor is used for writing the source code. This source code is saved as .asm file. Then this .asm file is compiled and converted to .hex file (machine code) by using MPASM. The source code of the program is included in this document as attachment.

The hardware of the unit is composed of MCU Unit, Relay Board, Keypad, LCD Display and a clock generator.

Power input and clock generator which produce clock signals in two different frequencies, are on a board. Microcontroller is on the board that is connected to Keypad and LCD. Ten relays, the outputs, are on another one.

4.2 THE OPERATION OF THE DEVICE

In this control unit, there are four inputs:

• Clock Signal Input
• U/D (Up/Down) Signal Input
• Index (External Reset) Input
• Keypad

The outputs are the ten relays and LCD Display.

This controller counts the clock signals coming to its interrupt input and it displays them on the LCD. When it reaches the values saved before, the corresponding relay is activated and the LED for this relay is ON.

With the coming clock signals, the counter increases. If U/D input goes to down (0), the counter decreases with the clocks. This feature was developed for that the device may be connected to a position sensor or another device generating these two signals; clock and U/D. U/D signal refers to two opposite movement direction.

The ten relays are activated in order. So the operating times must be in order. That is firstly relay A is activated and then B, C…. The time and the status of the next relay to be operated are displayed on the LCD.

The other feature of the device is that relays can be wanted to be ON continuously or ON until the next relay operation. In other words the status of relays may be temporary or continuous.

The above mentioned external reset input can be used at the end of the operation. The other alternative is that to operate the device periodically, this input can be connected to one of the relays. For example for the cable tests that will be mentioned in the next chapter, it must be run periodically and so one of the relay is connected to external reset. This will be explained in the next chapter in detail.

4.2.1 Read Mode

The controller allows the user to read and to change the operating times of the relays. If the (*) key is pressed on the keypad, the device wants from the user to enter the relay number. After pressing the number, associated relay time and status is displayed on the LCD. This is the read mode.

4.2.2 Edit Mode

On the other hand, if (#) is pressed, again the relay number is wanted from the user. After entering the number, the user must enter a valid operating time for the relevant relay. This value has to be higher than the previous relay and lower than the value of the next relay. The maximum number of signals the device can count is 999.999. And the last step for this write mode is to enter the status of the relay. This status can be continuous or temporary. If “0” is pressed the status of the relay is continuous, if “1” is pressed it is temporary.

4.3 DESCRIPTION OF THE SOFTWARE

The program of the device is written in assembly language and it can be examined part by part. It includes interrupt service routines, procedures and main routine.

4.3.1 The Beginning of the Program

In the beginning of the program, the microcontroller which is 16F877 is declared by using the LIST directive and the format of the .hex file is chosen as Intel hex format. This format produces one 8-bit hex file with a low byte, high byte combination. Since each address can only contain 8 bits in this format, all addresses are doubled. This file format is useful for transferring PICmicro series code to programmers.

With the CONFIG directive, the configuration bits are set. In the program PWRT and BODEN are set as ON, the others are set to OFF.

At the second part, the addresses of the registers and the ports are defined.

4.3.2 ISR Interrupt Service Routine

At the ISR Interrupt Service Routine, the values of the W and STATUS registers are saved for normal program flow, so the MCU may continue without error. PortD is saved for correct key input routines. Key input routines may be interrupted, so saving of PortD is vital.

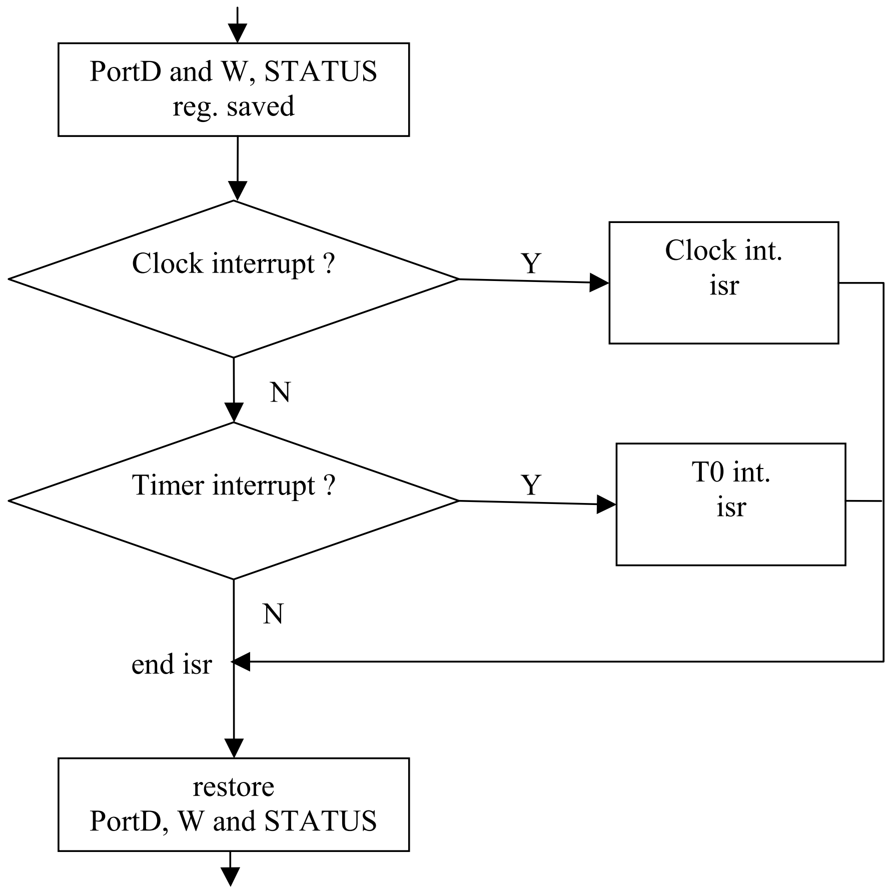


Figure 4.1 Flow Chart of the ISR Interrupt Service Routine

4.3.3 Timer0 Interrupt Service Routine

Timer0 Interrupt Service Routine checks if the index is on. If it is on, then on_index procedure is performed. If it is not on, TIMER value increments and LCD handler is done. When TIMER reaches to eight, relay handler is called. This is to satisfy that about four relays refresh cycle corresponds to one complete LCD refresh cycle. One complete LCD refresh cycle is completed about 30-40 LCD handler routines.

LCD refresh and relay refresh are put into interrupt service routine. The temporary registers of LCD and relays are loaded within the execution cycles so no delays occur while refreshing LCD. One character is send every Timer0 interrupt and the time for “LCD to execute the command” is used for normal program execution of MCU.

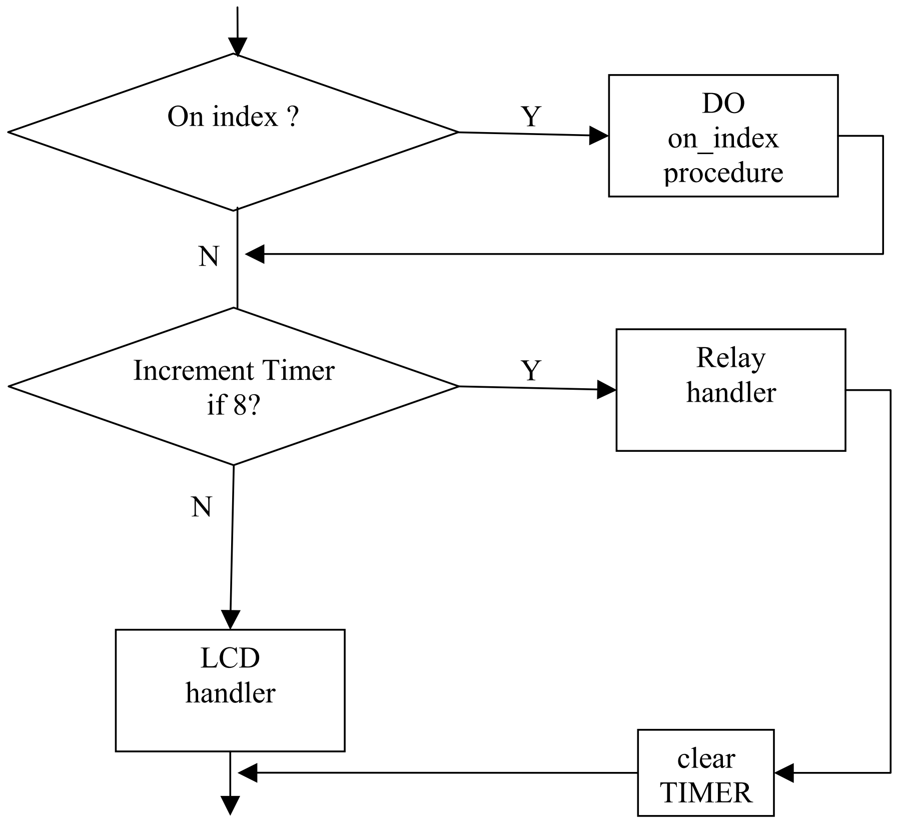


Figure 4.2 Flow Chart of the T0 Interrupt Service Routine

4.3.4 On Index Procedure

On index is a position that the device stays at ‘zero’ condition and does not count.

In the on_index procedure:

• Count is reset to ‘000000’. (Count_lo, Count_md and Count_hi registers)

• Comparison index is reset. (Comp Index register)

• Comparison value is reset. (Comp_lo, Comp_md and Comphi registers)

• All relays are reset (Switch_hi and switch_lo registers)

• The first comparison value is loaded (by calling load_comp_val procedure)

4.3.5 Load Compare Value Procedure

In this device, all relay switching values are not compared every time. Only the operating time value of the relay which is coming after the last switched relay is compared. So the switching time values of the relays must be from smallest to largest for proper execution. Comp_index register is used to determine to know which relay will be switched next.

This procedure loads the current comparison values to previous registers:

$\text{COMP\_XX} \longrightarrow \text{PREV\_XX}$

Then new comparison values are loaded from EEPROM.

Table 4.1 EEPROM Organization

HI	MD	LO	STATUS	COMP INDEX
0x00	0x01	0x02	0x03	0
0x04	0x05	0x06	0x07	1
0x08	0x09	0x0A	0x0B	2
0x0C	0x0D	0x0E	0x0F	3
0x10	0x11	0x12	0x13	4
0x14	0x15	0x16	0x17	5
0x18	0x19	0x1A	0x1B	6
0x1C	0x1D	0x1E	0x1F	7
0x20	0x21	0x22	0x23	8
0x24	0x25	0x26	0x27	9

In the Table 4.1, addresses of switching values of the relays and their status (continuous or temporary) data can be shown. 16 values are found in EEPROM but the first ten is directed to relays.

4.3.6 Relays Handler Procedure

This procedure loads the status of the relays to hardware.

-	-	-	9	8	7	6	5
bit7							bit0

SWITCH_HI

-	-	-	4	3	2	1	0
bit7							bit0

SWITCH_LO

4.3.7 LCD Handler Procedure

LCD Handler work is divided into the sections. At each Timer0 interrupt, one of the sections is carried out. LHCOUNT register saves the number of the step of the routine. According to this variable, corresponding section is executed.

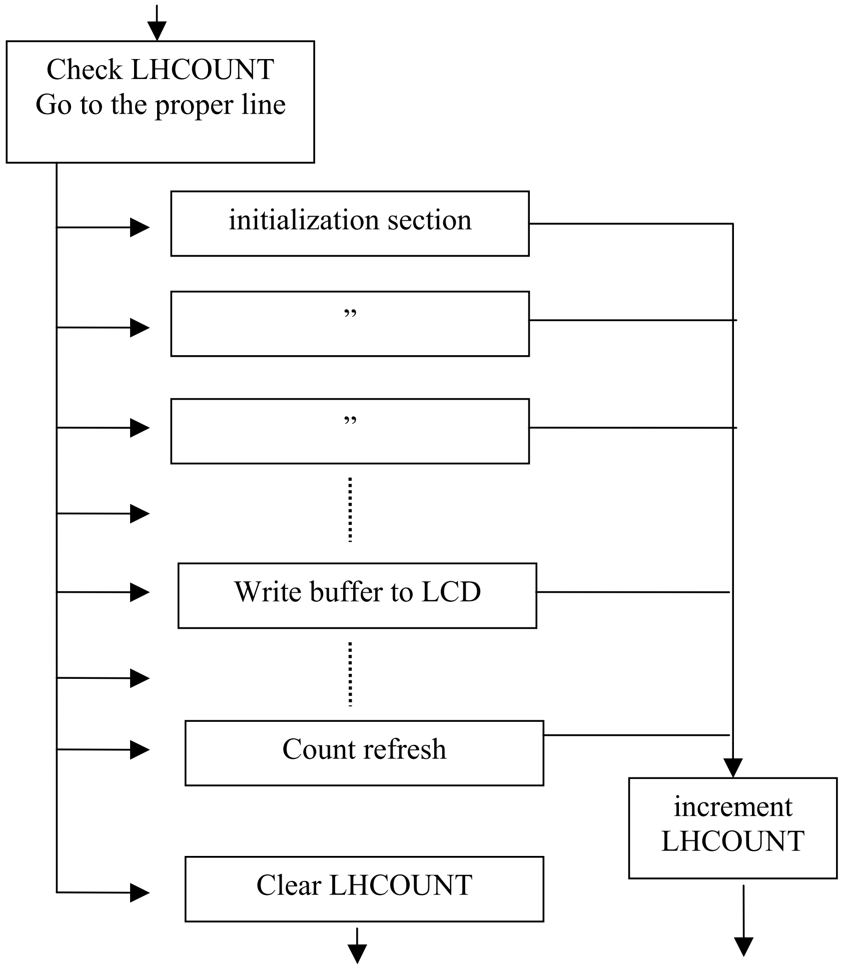


Figure 4.3 Flow Chart of the LCD Handler

Initialization sections are used to make LCD ready for data write. After initialization, the display data buffer DDxx registers are transferred. Then, the count is refreshed that means the actual count value is loaded to display data buffer and at last LHCOUNT is cleared, so the process starts from the first step.

In the LCD handler routine, data or command is loaded to W register, then related subroutine is called:

• bit_8_load	→	8 bit command sending routine
• bit_4_load	→	4 bit command sending routine
• disp_load	→	4 bit data sending routine
• refr_count	→	Transfers count value to DD01…DD06

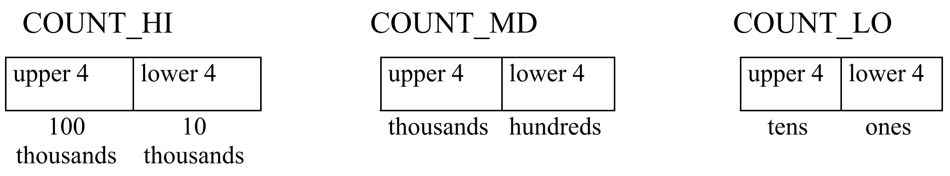
4.3.8 Clock Interrupt Service Routine

This is the interrupt routine for the pin RB0. RB0 is the clock input of the controller. This routine checks if on index, else counts up or down according to direction. Then it compares the switching values of the relays. If the limit value is reached, it loads the new data to compare registers and relays.

In the figure 4.4, the flowchart of the clock interrupt service routine is shown.

• Check direction checks the direction input of the device which is RB1 pin.

• Increase or decrease count modifies the COUNT_HI, COUNT_MD and COUNT_LO registers



• Compare is done between COUNT_xx and COMP_xx registers.

• Setting of relays is performed according to COMP INDEX, reset is done according to COMP INDEX – 1 and PREV_ST registers. Modifications are done on SWITCH_HI and SWITCH_LO registers. Then ISR sends them to the hardware.

• The loading new comparison values increases the COMP_INDEX that is the relay number to be compared and loads values to COMP_xx registers. The previous comparison values saved in PREV_xx registers.

• COMP_ST and PREV_ST registers hold the data that the relay is temporary or continuous.

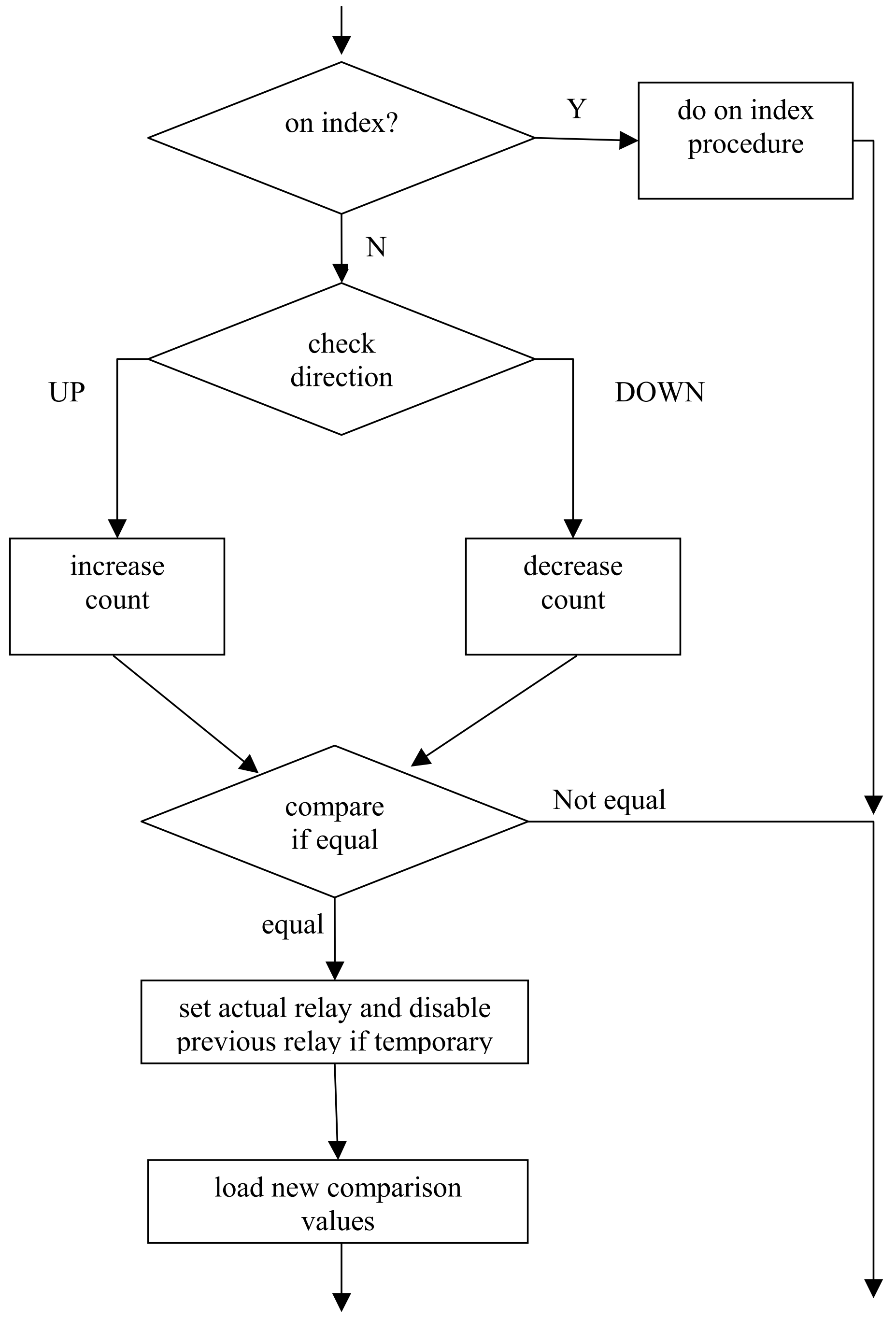


Figure 4.4 Flow Chart of the Clock Interrupt Service Routine

4.3.9 Main Routine

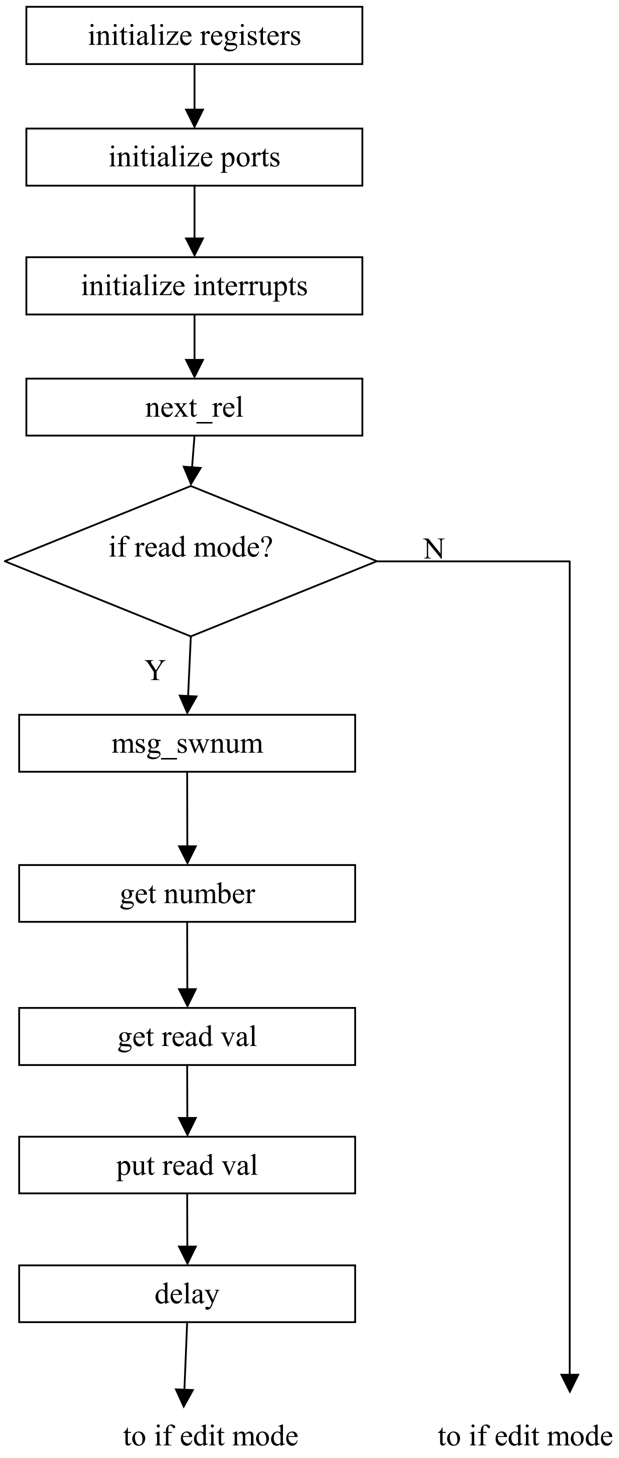


Figure 4.5 Flow Chart of the Main Routine

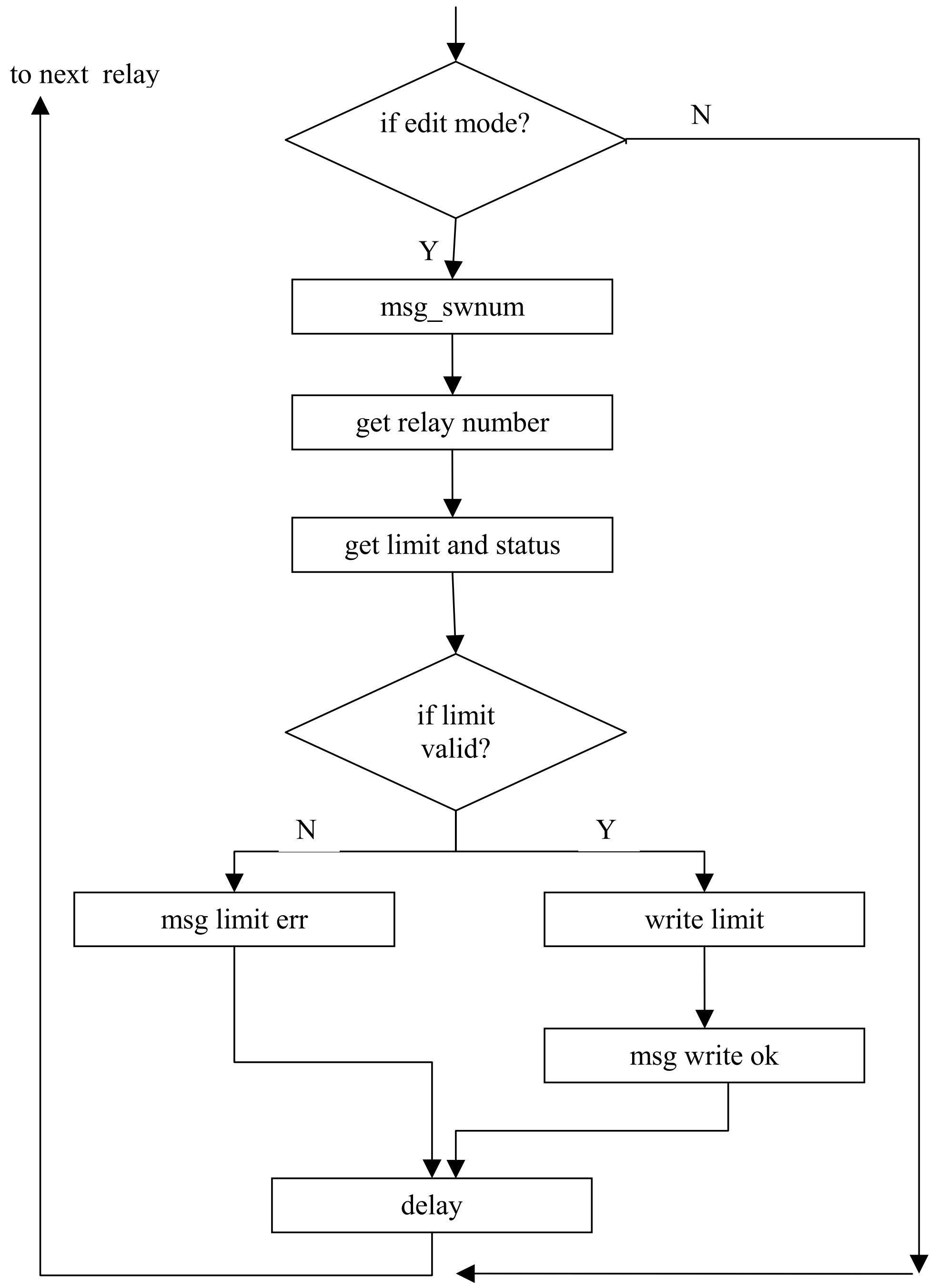


Figure 4.5 Flowchart of the Main Routine (continued)

4.3.9.1 Read Mode

If the key (*) is pressed by keypad, the device enters the read mode and wants from the user the relay number. Get number procedure takes this value to EDIT_NUM register. Then according to EDIT_NUM, data is loaded from the EEPROM and send to LCD buffer.

4.3.9.2 Edit Mode

If the (#) key is pressed, this is the edit mode for the controller. In this condition Zero bit of the STATUS register is set. The device waits till a number key is pressed, then return in W. The limit numbers are saved to TEDIT1…TEDIT6 and relay status is saved to TEDITST register, if the limit is valid.

4.4 HARDWARE OF THE CONTROLLER UNIT

The hardware of the unit is composed of the MCU Unit, Relay Board, Power Supply, LCD Display, Keyboard and Clock Generator.

4.4.1 MCU Unit

The MCU Board includes the controller PIC 16F877 and connections to all other hardware units.

- The clock input of the device is RB0 pin on the PIC. This is logic ‘0’ as default, if it goes to ‘1’, interrupt service routine starts.
- RB1 pin is the direction input. It is logic ‘1’ and counter increments. When it goes to ‘0’, counter decrements.
- RB2 is the index (or external reset) input. It is ‘0’, otherwise reset occurs.
- The Vss or Ground pins of the controller are 31 and 12. V_{DD} or power inputs to controller are 11 and 32.
- RD4 (Fourth pin of the Port D) is reserved for LCD command and RD6 is reserved for relay latch.
- Master Clear ($\overline{MCLR}$) is ‘1’ normally. This pin is an active low Reset to the device. If this reset occurs, the values in the registers return to default. Some of them take random values.

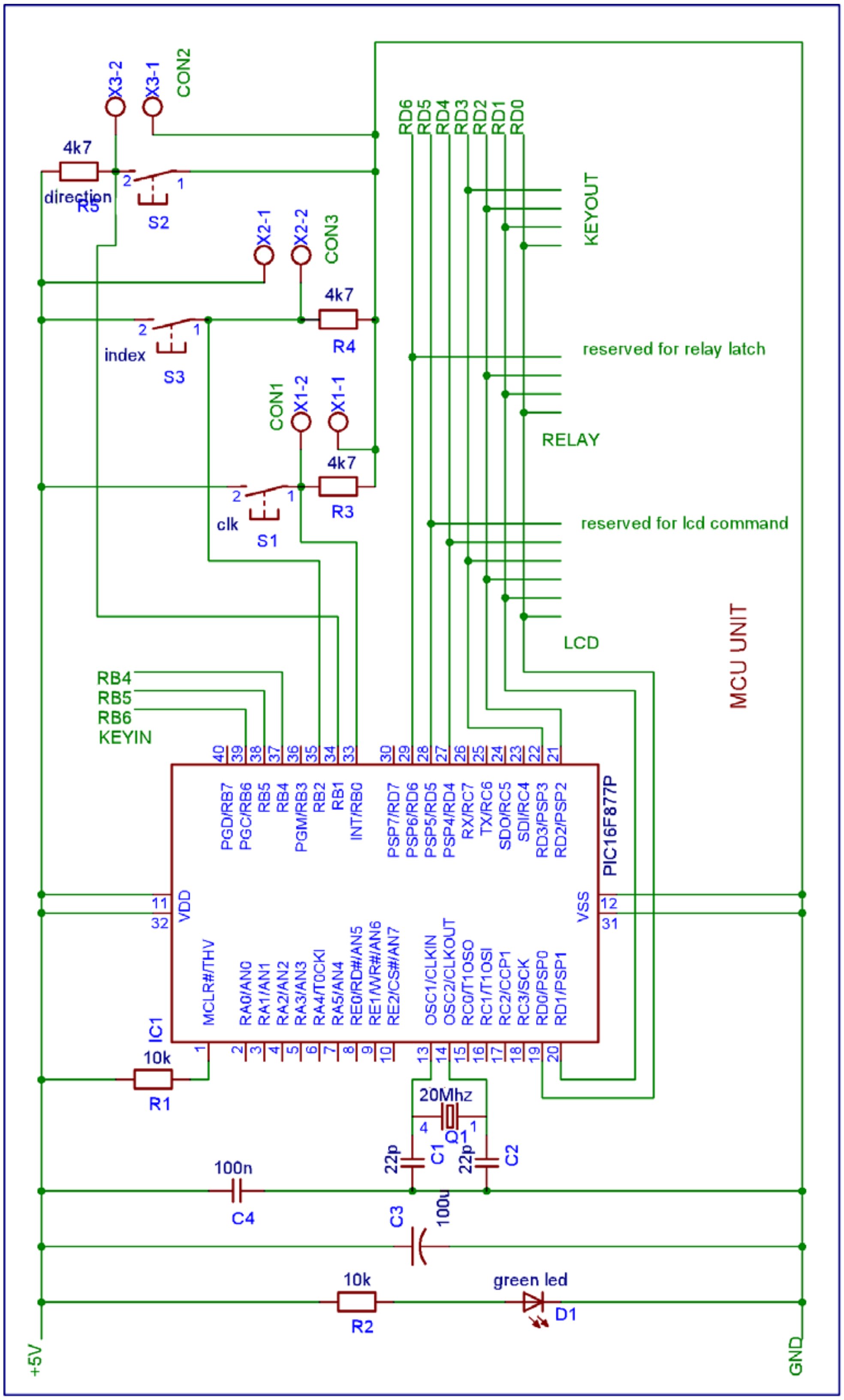


Figure 4.6 Circuit Diagram of the MCU

• Crystal is used to generate the clocks for the controller. Since this crystal produce clock of frequency 20 MHz, five million instructions are executed in one second.

• C4 capacitor is connected to filter the noise.

• D1 green led is for ON/OFF indicator.

• Ports A, C and E is free. The Port A has the A/D converter.

4.4.2 Relay Board

The relay board has ten relays which are the outputs of the controller unit. To indicate the relay operation, ten leds are used. This board also includes two shift registers.

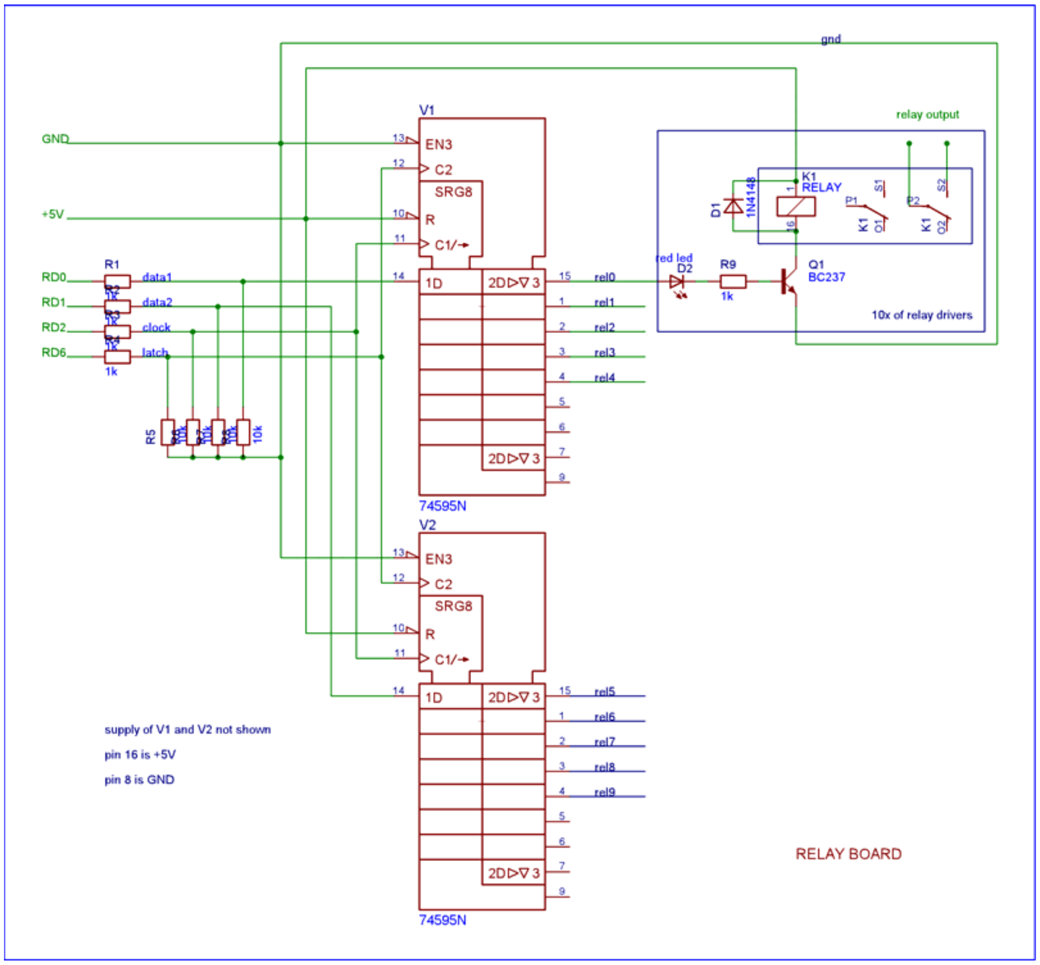


Figure 4.7 Circuit Diagram of the Relay Board

• The power supplies of the shift registers are not shown in the circuit. Pin 16 is +5V and pin 8 is Ground.

• The values in the SWITCH_HI and SWITC_LO registers are loaded to shift registers firstly. Then by giving a latch input, datas are transferred to relays.

• If the logic ‘1’ comes to the relay, transistor operates and contact is closed.

4.4.3 LCD Display

The LCD in this unit is 1x24 and HD44780U compatible.

• LCD has 11 pins, 8 pins for data and 3 for control.

• 6 pins of 11 are used in this unit.

• Since the device only writes to LCD, do not read from, R/W pin is connected to ground.

• LCD Display can be loaded as 4bit plus 4bit in two times or 8bit in one time. In this device, it is loaded as 4bit-4bit to use less number of ports.

• The R8 resistor is used for setting the contrast of the LCD.

• To prevent the noise resistors are used. (R2…R7)

• The capacitors C1 and C2 are the filter for supply.

4.4.4 Keypad

• One of the keyouts from the microcontroller is done logic ‘1’. The others are ‘0’. If a key is pressed, one of the RB4, RB5 and RB6 is activated. In other words, controller reads from keypad in row by row.

• The reason of using the diodes is to prevent the short circuit in case pressing two keys in different rows at the same time.

• R1, R2 and R3 resistors are used to accelerate the response.

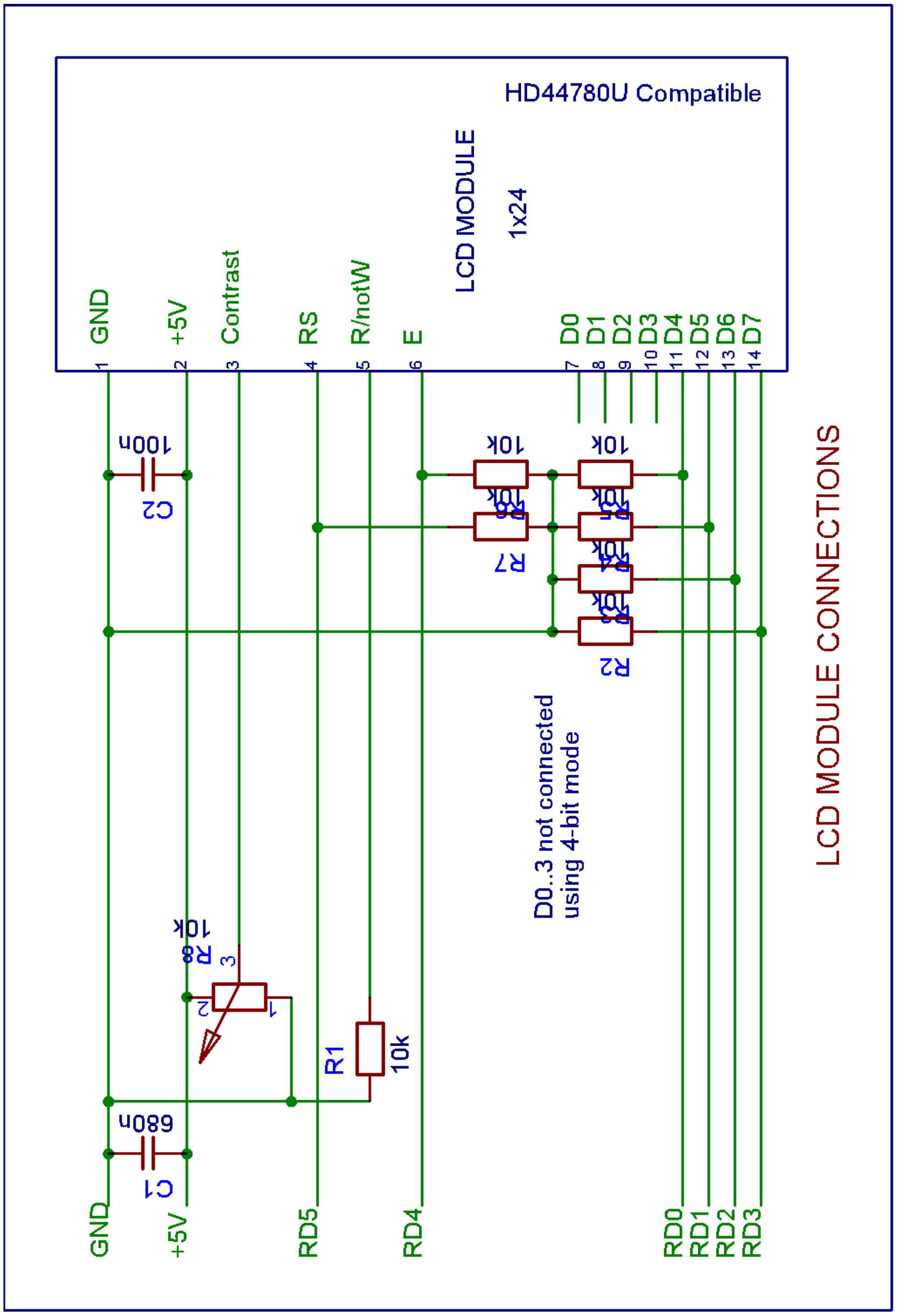


Figure 4.8 Circuit Diagram of the LCD Module

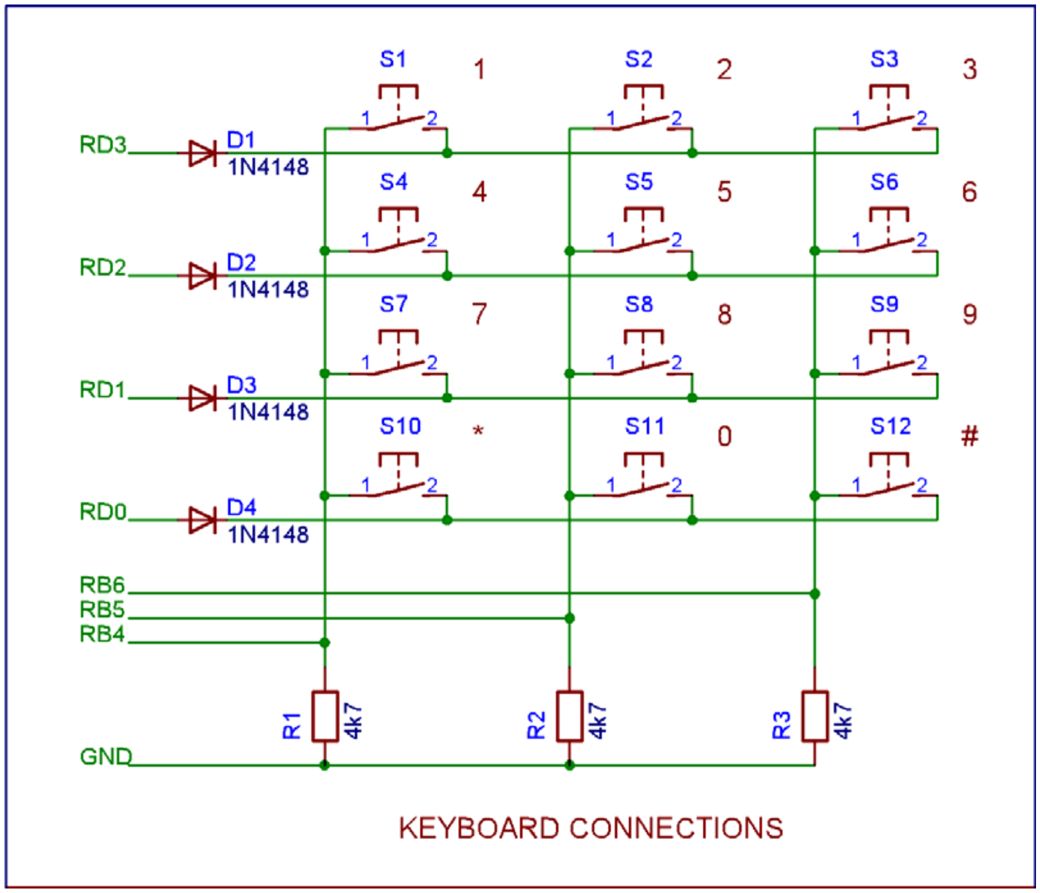


Figure 4.9 Circuit Diagram of the Keyboard

4.4.5 Power Supply and Clock Generator

• The two 7805s are the voltage regulators. They are used to provide 5 V.

• One of the regulators is free. If a battery is needed, this regulator can be used. In this case, relay board is connected to OUT port of it. So, the battery supplies only MCU.

• Timer 555 is a clock generator. It produces clocks in two different frequencies. If the switch is closed, capacitance increases. Increasing capacitance results in low frequency.

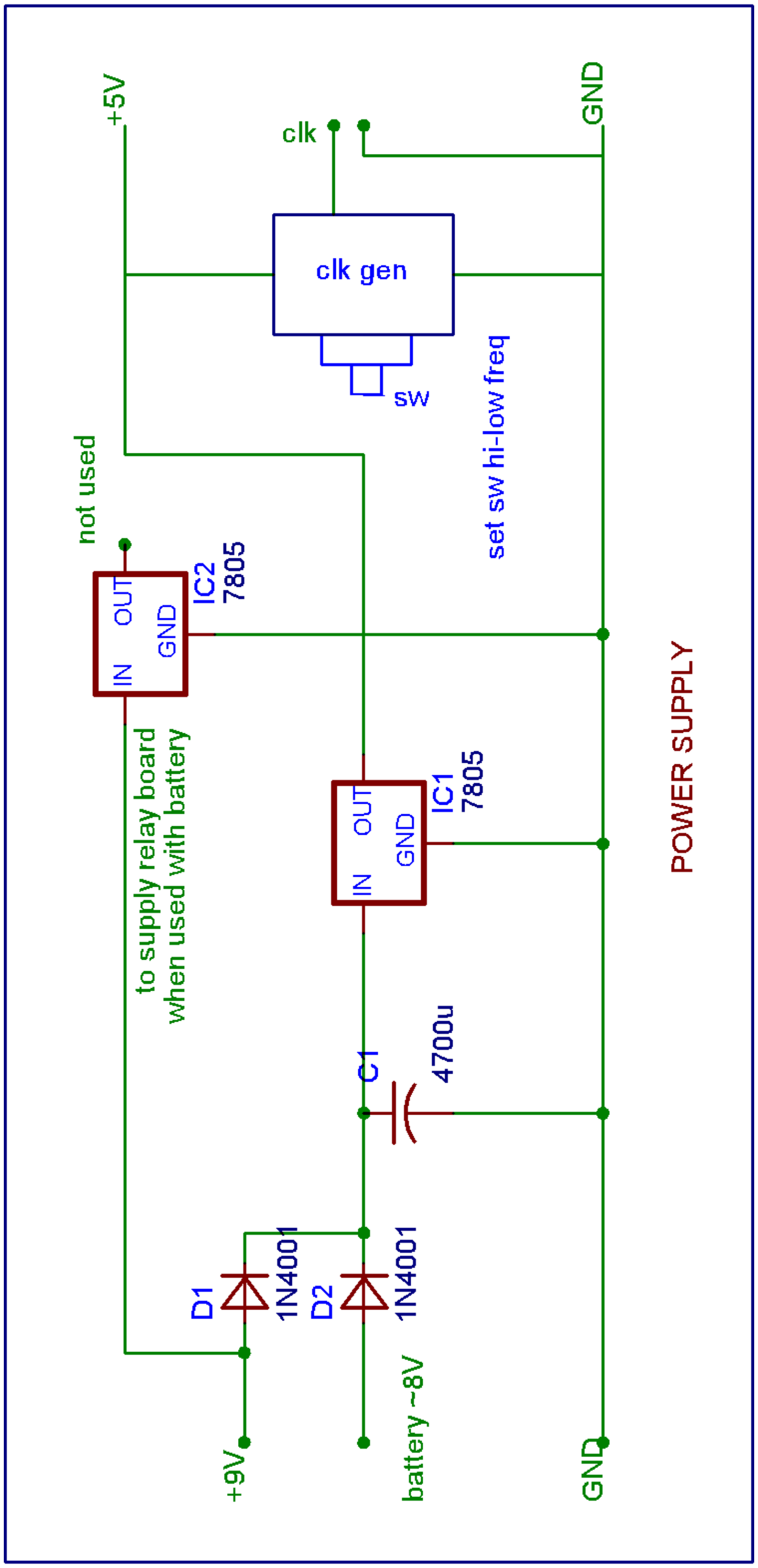


Figure 4.10 Circuit Diagram of the Power Supply

CHAPTER 5

APPLICATIONS OF THE CONTROL UNIT: POWER CABLE ACCESSORIES TESTS AND ELECTRO DISCHARGE MACHINING

The two applications of the control unit implemented in this study are the automation of the power cable accessories tests and electro erosion machining. In these processes, switching of some devices is required. Ten output relays are employed in the unit for this operation. Clock signals are prodeced by internal Timer 555. Microcontroller counts these signals and activates the outputs in order.

For some applications, the periodic operation is needed. This is done easily by using this controller. To achieve it, one of the outputs is used to reset the microcontroller. The others are set to proper operating times.

The power cable accessories tests which are described herewith are Thermal Cycling Test and Screen Fault Current Initiation Test. Methods for these tests are specified for accessories for extruded insulation cables.

5.1 Thermal Cycling Test

5.1.1 Installation

The arrangement for test in air shall be as shown in the figure. For the thermal cycling in water, terminations or separable connectors must be installed in a container to have a water level 1 m above the all accessories.

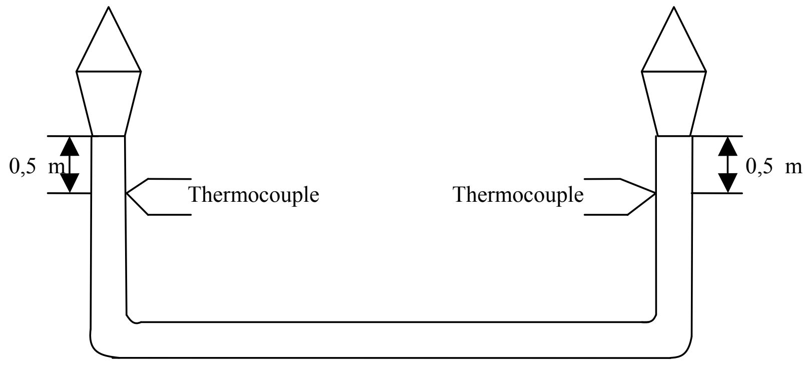


Figure 5.1 Terminations tested in air

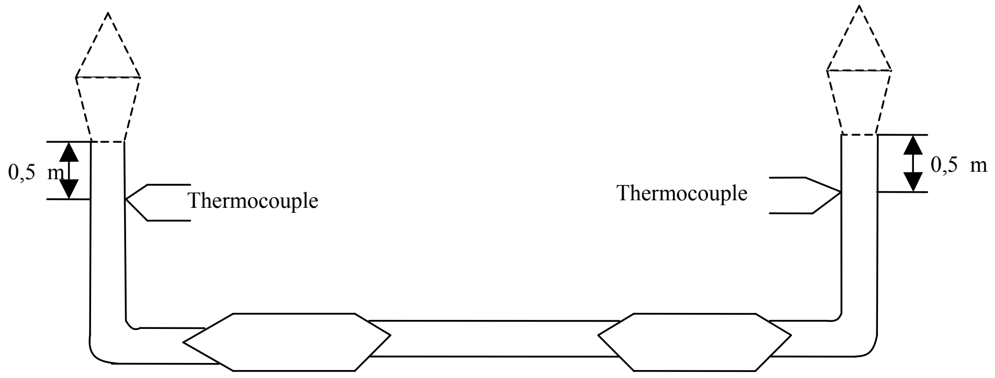


Figure 5.2 Joints tested in air

5.1.2 Method of the Test

Each thermal cycle in air or in water shall be of 8 hours duration with at least 2 hours at a steady temperature:

- 5 K to 10 K above the maximum cable conductor temperature in normal operation for extruded insulation cables.
- 0 K to 5 K above the maximum cable conductor temperature in normal operation for paper insulated cables

followed by at least 3 hours of natural cooling to within 10 K of ambient temperature.

The test assemble shall be subjected to the required number of thermal cycles, energized at the voltage given in the relevant standard.

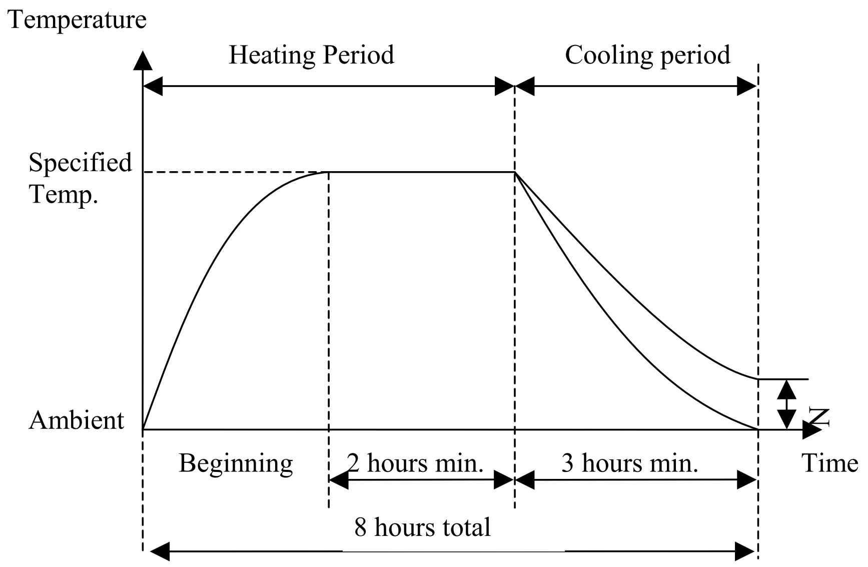


Figure 5.3 Thermal Cycle

5.1.3 Immersion test for outdoor terminations

This test shall replace the last ten cycles of the thermal cycling test in air, thereby keeping the total number of cycles the same for all accessories.

5.1.3.1 Installation

Two terminations of a test loop shall be immersed in water at ambient temperature with a height of water of 0,03 m above every part of the termination. The test loop shall be installed upside down in a water tank, at ambient temperatures, in such a way that the terminations are fully immersed in water, including the end of the sealing element.

5.1.3.2 Method

The test loop shall be subjected to ten cycles under the thermal conditions of sub-title 5.1.2 of this chapter. The test loop shall not be energized.

5.1.4 Automation method with the control unit

In the thermal cycling test, we need four relays for automation. Two of them are connected to contactors which control the devices used for heating. The first and second relays open or close the contactors. The loaded values and the statuses which determine the operating time of the relays are as follows:

- First relay : operates at 81120 signals and the status is continuous.
- Second relay : operates at 81121 signals and the status is temporary.
- Third relay : operates at 101400 signals and the status does not matter.
- Fourth relay : operates at 162240 signals and the status does not matter.

The Timer 555 in the control unit can produce clock signals in two different frequencies. The slower one is used in this application. 338 signals are sent to microcontroller in one minute.

The first two relays are set to 81120 and 81121, so 81120 : 338 = 240 minutes = 4 hours. After this period, two devices connected to these relays operate for heating. (Beginning period in Figure 5.3 starts). One hour later third relay operates and second one is deactivated due to its temporary status. So, one heating device left for keeping temperature constant. (The second part of the Figure 5.3 starts). After three hours the fourth relay is activated and the cooling period starts. The fourth relay is connected to external reset input of the unit. This resets the unit and operation goes on periodically.

5.2 Screen Fault Current Initiation Test

The purpose of this test is;

a) In the case of a solidly earthed system, to demonstrate the ability of the separable connector screen to initiate a fault to earth which produces sufficient current to operate the circuit protection, should the insulation fail.

b) In the case of an unearthed or impedance earthed system, to demonstrate that a separable connector, which has failed, can be clearly recognized as being faulty.

The test is applicable only to screened separable connectors and shall be carried out with the connectors installed as in service.

The test for screened separable connectors with a metal housing is under consideration.

5.2.1 Installation

A separable connector shall be assembled on a cable in accordance with the manufacturer's instructions. All parts of the separable connector which are normally earthed shall be connected to the cable screen, including the bushing screen.

For testing separable connectors used in solidly earthed systems, the faulting rod shall be of erosion resistant metal, approximately 10mm in diameter, threaded at one end to engage the accessory metal connector through a drilled hole. The rod shall be in contact with the inner and outer screens and shall not protrude beyond the outer screen surface.

Test arrangement for screen fault current initiation test for separable connectors used in unearthed systems or impedance earthed systems, the faulting rod shall be replaced by a copper wire of approximately 0.2 mm diameter. The wire shall be in contact with the inner and outer screens and shall not protrude beyond the outer screen surface.

5.2.2 Method

5.2.2.1 Solidly Earthed System

The test shall be carried out at ambient temperature. The circuit shall be adjusted to impose the separable connector phase to earth voltage Uo on the test specimen and a short circuit current of 10 kA r.m.s. The test specimen shall be subjected to two tests that cause initiation of a fault current arc to earth, each operation having a minimum current flow duration of 0,2s. Between the two tests, the test sample shall be allowed to cool to a temperature less than 10 K above its temperature prior to the first test.

5.2.2.2 Unearthed or Impedance Earthed System

The test shall be carried out at ambient temperature. The circuit shall be adjusted to impose the separable connector phase to earth voltage Uo on the test specimen and a short circuit current of at least 10 A.

The current for the short-circuit test is to be determined taking into account the actual short-circuit conditions of the network.

The test voltage and current shall be recorded continuously during the entire period. The sequence of the test shall be as follows:

1. Voltage switched on for 1 s
2. Voltage switched off for 2 min
3. Voltage switched on for 2 min
4. Voltage switched off for 2 min
5. Voltage switched on for 1 min
6. Voltage switched off

5.2.3 Automation method with the control unit

To carry out this test for unearthed or impedance earthed system, the operating times and statuses of relays are as follows:

- First relay	: operates at 1 signal and the status is temporary.
- Second relay	: operates at 7 signals and the status is temporary.
- Third relay	: operates at 683 signals and the status is temporary.
- Fourth relay	: operates at 1359 signals and the status is temporary.
- Fifth relay	: operates at 2035 signals and the status is temporary.
- Sixth relay	: operates at 2373 signals and the status is temporary.

The relays are connected in parallel to control the contactor. With the first, third and fifth relays the voltage switched on and other relays are for switching off. Six signals come from Timer 555 in one second. So the sequence of test mentioned in previous pages is achieved.

5.3 Electro Erosion Machining

5.3.1 Introduction

Electric Discharge Machining (EDM) is a metal removal process where two electrodes are used to produce a spark in a dielectric liquid medium. The cathode (negative electrode) is the workpiece itself and the anode (positive electrode) is the tool, shaped with the inverse of the detail required. The two electrodes never actually come in contact and a small gap is maintained between them at all times by servo control. The two electrodes are submerged in a dielectric fluid, which allows a path for the electric discharge to be made, cools the tool and workpiece and removes the waste products. The resistance of the dielectric fluid is important in producing a stable operating mode since it affects the sparking conditions.

The discharges are produced by a D.C. power supply, which is connected to the two electrodes. Thermal energy is produced in the form of localised heat and the temperature reaches approximately 12000°C, which is sufficient to melt and vaporise almost all metals and alloys. Thousands of discharges occur each second, each removing a small particle from the workpiece, and hence the area under the tool is gradually eroded. Of course, the tool is also eroded like the workpiece as tool material is also subjected to the intense heating. However this is minimized by selecting a proper tool material and also the correct polarity and duration of spark voltage. Tool wear is measured as the percent ratio of tool material removed to workpiece material removed and this can vary greatly depending on tool and workpiece materials used - from about 1 to 1/1000.
Below is a simplified diagram of the basic EDM process

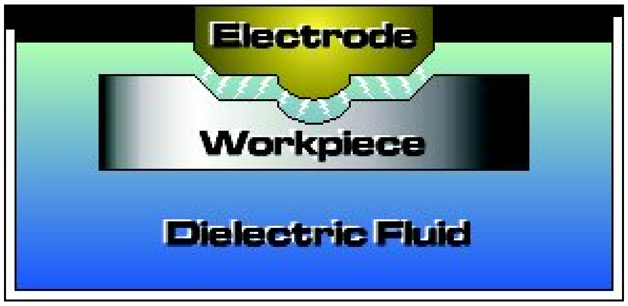


Figure 5.4 Diagram of the EDM process

Common materials used for the electrodes are graphite, copper and copper tungsten as these are good conductors with high melting point. Example of dielectric fluid is hydrocarbon oils.

Wire EDM is another type of electric discharge machining. Rather than using a large electrode, wire EDM used a long thin electrode (usually brass wire) which is constantly being renewed to produce the sparks. This avoids the effects of the wear in the tool as it is used. Otherwise this process is exactly the same as EDM. It can machine to an accuracy of ±0.002 inches. Figure 5.5 shows a simplified diagram of the wire EDM process.

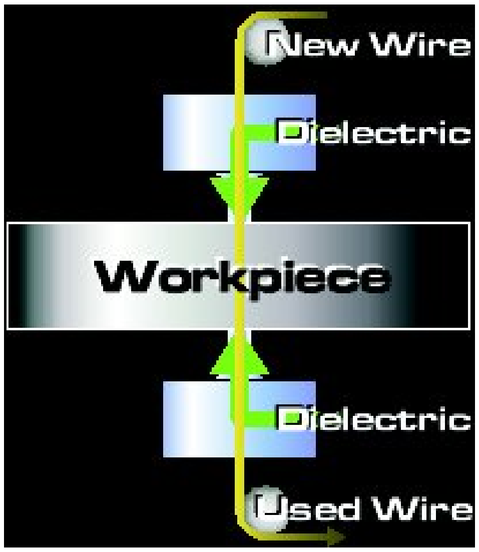


Figure 5.5 Diagram of the wire EDM process

5.3.2 Advantages and Disadvantages of EDM

Advantages

- Accuracy - EDM of all types is very accurate compared to other types of machining processes.
- Flexibility - EDM can create shapes and contours that other forms of machining are not capable of.
- Functionality - EDM can be used on any conductive materials whether they are hard or soft including metals, alloys and carbides which are to hard to machine using traditional methods.

- Prototyping - EDM is often used in prototyping because of the speed in which the designs can be changed.
- Delicate Machining - The tool never touches the workpiece so it cannot destroy it.
- Finishes - EDM can produce very fine surface finishes eliminating the need for grinding.

Disadvantages

- Cost - The tool is subjected to wear and so it may need replacing if the same component is to be reproduced many times.
- Material – Only conductive materials can be machined.

5.3.3 Automation method with the control unit

There are four inputs in the electro erosion process. They are :

1. Electrode gap voltage (eg. 0 – 20 Volt)
2. Electrode mean current (eg. 0- 50 Amper)
3. Dielectric liquid conduction (eg. 0 – 50 mA at 24 Volts)
4. Electrode position (eg. 0 – 100 mm in 0.01 mm steps)

The controlled parameters in this process are :

1. On time
2. Off time
3. Electrode position
4. Magnitude of the current

At least one of these parameters (generally all of them) is changed at each mode variation. Ton, Toff and electrode position determine the actual frequency.

In fixed pulse width (Ratio of On Time to Off Time), the higher of current, the higher the metal removal rate. However if the current value per square cm is over 25 A, difficulty in keeping the liquid in the spark gap clean will be experienced. This may lead to the loss of operation stability and increased electrode wear. So when in small hole or small area machining, low electrode currents should be used.

In fine machining, the current intensity and the spark duration are decreased gradually. Hence weak but high frequency sparks are used.

In the electro erosion process, the modes of the machine and the electrode position can be like that:

	Electrode position
1. mode (roughing)	0 - 9.50 mm
2. mode (finishing)	9.50 - 9.70 mm
3. mode (finishing)	9.70 - 9.85 mm
4. mode (finishing)	9.85 - 10.00 mm

In the EDM system, to change the Ton and Toff, timer capacitances are added to the timer circuits. Magnitude of the current can be increased by adding parallel resistors. Peak value of the electrode current decreases for example from 20 Amper to 1 Amper with the changing mode from 1 to 4. At the same time, Ton changes from 2000 μs to 20 μs and Toff chages from 500 μs to 10 μs.

To automate the electro discharge machining process with the control unit, a linear decoder at z-axis can be used. The position of the electrode changes and the clock signals are sent to the controller. The unit counts these signals to know where the electrode is. The movement resolution which can be controlled is 0.01 mm.

Table 5.1 Sample values of the parameters in EDM process

Mode	Ton (μs)	Toff (μs)	Servo (V)	Gap (mm)	Current Intensity (A)
1	2000	500	1.35	0.100	20
2	200	100	1.40	0.080	5
3	50	20	1.45	0.070	2
4	20	10	1.50	0.050	1

The controller switches the electro erosion machine according to electrode position and the mode is changed from roughing to finishing. The change of the parameters may be as in the Table 5.1.

CHAPTER 6

CONCLUSIONS

In this thesis, a control unit is designed, implemented and programmed to automate the long term processes which involve cyclic adjustment of a number of parameters. In the present study, this unit was applied to control high voltage power cable accessory tests.

To achieve this objective, alternatives for control systems are investigated. Using timing relays are higher in cost and size. They are not flexible for programming, variations and future developments.

A microcontroller is used for this project. They are used to control almost all automated processes. A microcontroller can be used to perform almost any function. It is small, relatively cheap and reliable. More importantly replacing it with a similar system built from discrete components would involve a very difficult and complex design task; an enormously complex, very large and unreliable electronic system, and a lot of maintenance.

All the systems that are built or investigated using discrete components have their function decided by the connections between the various components. They are hard wired; their function can be changed by rewiring them.

As in this project, a programmable system such as a microcontroller however, can have its function changed without changing any of its connections. To communicate with the microcontroller, instructions must be given it so that it can function properly.

Using PICs can make the products more flexible. Their features are programmed into the chip, not built into electronic hardware, so they can be developed and changed quickly and easily. A simple change to the PIC program can achieve what is required without the need to alter any of the components on the board.

The main disadvantage of PICs is that they have only a very low power output, of a few milliamps. They therefore require interfacing to drive higher current loads.

The microcontroller employed in this device uses reprogrammable “flash memory” which can be written and rewritten to with ease. This is a very important feature. It has a maximum frequency of 20 MHz, causing execution time for one instruction is 200 ns and therefore it executes 5.000.000 instructions in a second. High operating speed is one of the main characteristic of the microcontroller which is an advantage while counting high frequency clock signals and refreshing LCD display.

The control unit implemented in this study is designed by using as few pins as possible of the microcontroller for future developments. Therefore, addition of inputs, outputs and memory devices etc. is possible.

It is a system suitable for future improvements. The number of outputs can easily be increased since sixteen memory locations are reserved in the EEPROM for the operating times and statuses of the relays. The six more relays can be connected to the unit if required.

By using additional EEPROM and software modification, various programs including different operating times and statuses of the relays can be saved in the EEPROM memory. Therefore, user calls the any program from the memory and the relevant operating times and statuses for relays are loaded. The user must enter only the program number.

With the features mentioned in this thesis report, the application field of the device is diversified; it can be also used widely in industrial applications. By using this device for automation, there will be considerable decrease in the dependency on the operator attention which shall be needed only when data is loaded or changed.

REFERENCES

[1] John B. Peatman, “Design with PIC Microcontrollers”, Prentice Hall, 1997.

[2] Myke Predko, “Programming and Customizing PICmicro Microcontrollers”, McGraw-Hill, 2000.

[3] David W. Smith “PIC in Practice”, Newnes, 2002

[4] David Benson “Easy PIC’n”, Square One Electronics, 1999

[5] “PICmicro Mid-Range MCU Family Reference Manual”, Microchip Technology Inc., 1997

[6] “PIC16F87X Data Sheet 28/40 Pin, 8-Bit CMOS FLASH Microcontrollers”,

Microchip Technology Inc., 2001

[7] “MPLAB IDE User’s Guide”, Microchip Technology Inc., 2000

[8] “MPASM User’s Guide with MPLINK and MPLIB”, Microchip Technology Inc., 1999

[9] “EEPROM Memory Programming Specification”, Microchip Technology Inc., 2002

[10] David Benson, “PIC’n up the Pace”, Square One Electronics, 1999

[11] Carl J. Bergquist “Guide to PICmicro Microcontrollers”, Delmar Learning, 2000

[12] Sid Katzen “The Quintessential PIC Microcontroller”, Springer Verlag, 2001

[13] “EEPROM Memory Programming Specification”, Microchip Technology Inc., 2002

[14] Martin Bates “Introduction to Microelectronic Systems: The PIC 16F84 Microcontroller”, Butterworth – Heinemann, 2001

[15] "Test Methods for accessories for power cables with rated voltage from 3,6/6 kV (Um=7,2 kV) up to and including 20,8/36 kV (Um=42 kV)" CENELEC Standard No. HD 628 S1 (1996) + A1 (2001)

[16] E. P. De Garmo, J. T. Black and R. A. Kohser "Materials and Processes in Manufacturing" 8th edition, Prentice Hall, 1988

[17] S. Kalpakjian "Manufacturing Processes for Engineering Materials" 3rd edition, Addison Wesley, 1997

[18] M. Burns "Automated Fabrication: Improving productivity in manufacturing" Prentice Hall, 1993

APPENDIX A

THE SOURCE CODE OF THE CONTROLLER

```
list      p=16F877;f=inhx8m           ;list directive to define processor
#include <p16F877.inc>                ;processor specific variable definitions
__CONFIG 0x3D71                       ;pwrt, boden, ON remaining fuses OFF, XT

;*****************************************************************************
;

W_TEMP              equu    0x70              ;W, STATUS and PORTD
STATUS_TEMP         equ     0x71              ;interrupt save registers
PORTD_TEMP          equ     0x72

DATA_EE             equ     0x74              ;eeprom variables
ADDR_EE             equ     0x75

LCD_DATA_P          equ     PORTD             ;lcd port definitions
lcd_e               equ     4
lcd_rs              equ     5

TIMER               equ     0x20              ;TIMER0 isr program variable

IN_PORT             equ     PORTB             ;clock input port definitions
direction           equ     1
index               equ     2

REL_PORT            equ     PORTD             ;relay port definitions
data_lo             equ     0
data_hi             equ     1
rel_clk             equ     2
rel_latch           equ     6

KEY_OUTPORT         equ     PORTD             ;keyboard port definitions
KEY_INPORT          equ     PORTB

DELAY1              equ     0x21              ;delay routine variables
DELAY2              equ     0x22
DELAY3              equ     0x23
LHCOUNT             equ     0x24              ;lcd handler routine counter

DD01                equ     0x25              ;display data buffer
DD02                equ     0x26              ;disp is refreshed by these
```

```
DD03            equ     0x27            ;registers by timer interrupt
DD04            equ     0x28
DD05            equ     0x29
DD06            equ     0x2A
DD07            equ     0x2B
DD08            equ     0x2C
DD09            equ     0x2D
DD10            equ     0x2E
DD11            equ     0x2F
DD12            equ     0x30
DD13            equ     0x31
DD14            equ     0x32
DD15            equ     0x33
DD16            equ     0x34
DD17            equ     0x35
DD18            equ     0x36
DD19            equ     0x37
DD20            equ     0x38
DD21            equ     0x39
DD22            equ     0x3A
DD23            equ     0x3B
DD24            equ     0x3C

SWITCH_HI       equ     0x3D            ;relays status buffers
SWITCH_LO       equ     0x3E

LD_TEMP         equ     0x3F            ;lcd data temp reg

COUNT_LO        equ     0x40            ;main counter
COUNT_MD        equ     0x41
COUNT_HI        equ     0x42

COMP_HI         equ     0x43            ;values to be compared
COMP_MD         equ     0x44
COMP_LO         equ     0x45
COMP_ST         equ     0x46
PREV_HI         equ     0x47            ;previous comparison values
PREV_MD         equ     0x48
PREV_LO         equ     0x49
PREV_ST         equ     0x4A
COMP_INDEX      equ     0x4B            ;index of relays
TEMP_C_I        equ     0x4C            ;temp reg for index calculate

LHCOUNT2        equ     0x4D            ;LCD init handler counter

EDIT_NUM        equ     0x4E            ;data input variables
TEDIT1          equ     0x4F            ;used for keyboard data input
TEDIT2          equ     0x50
TEDIT3          equ     0x51
TEDIT4          equ     0x52
TEDIT5          equ     0x53
TEDIT6          equ     0x54
TEDITST         equ     0x55

UPPER_HI        equ     0x56            ;check limit variables
UPPER_MD        equ     0x57
```

```
UPPER_LO            equ     0x58
LOWER_HI            equ     0x59
LOWER_MD            equ     0x5A
LOWER_LO            equ     0x5B
;*****************************************************************************
                    ORG     0x0000                      ;processor reset vector
                    clrf    STATUS                      ;bank0 for ram
                    goto    main
;*****************************************************************************
;interrupt service routine - saves registers and
;restores the pre-interrupt status

                    ORG     0x0004
isr                 movwf   W_TEMP                      ;push W and STATUS and PORTD
                    swapf   STATUS,W
                    clrf    STATUS
                    movwf   STATUS_TEMP
                    movf    PORTD,W
                    movwf   PORTD_TEMP

                    btfsc   INTCON,INTF                 ;check if clk
                    goto    clock_int_isr

                    btfsc   INTCON,T0IF                 ;check if timer
                    goto    t0_int_isr

end_isr             movf    PORTD_TEMP,W                ;pop pushed registers
                    iorwf   PORTD,F
                    swapf   STATUS_TEMP,W
                    movwf   STATUS
                    swapf   W_TEMP,F
                    swapf   W_TEMP,W

                    retfie

;*****************************************************************************
;TIMER0             interrupt routine - checks if on_index then continues with
;lcd refresh and relays status refresh

t0_int_isr          bcf     INTCON,T0IF                 ;clear interrupt flag

                    btfss   IN_PORT,index               ;check if on index
                    goto    lcd_relais

                    clrf    COUNT_HI                    ;clear count values
                    clrf    COUNT_MD
                    clrf    COUNT_LO
                    clrf    COMP_INDEX
                    clrf    COMP_HI
                    clrf    COMP_MD
                    clrf    COMP_LO
                    clrf    COMP_ST
                    clrf    SWITCH_HI                   ;disable all relays
                    clrf    SWITCH_LO

                    call    load_comp_val               ;load start value of compare
```

```
                movlw   a'-'                        ;write '------' on the count
                movwf   DD01
                movwf   DD02
                movwf   DD03
                movwf   DD04
                movwf   DD05
                movwf   DD06

lcd_relais      incf    TIMER,F                     ;increase TIMER
                btfss   TIMER,3                     ;till reaches 8
                goto    lcd_handler                 ;if not reached do lcd work

rel_handler     bcf     REL_PORT,rel_clk            ;load data to HC595s
                bcf     REL_PORT,data_lo
                btfsc   SWITCH_LO,4
                bsf     REL_PORT,data_lo
                bcf     REL_PORT,data_hi
                btfsc   SWITCH_HI,4
                bsf     REL_PORT,data_hi
                bsf     REL_PORT,rel_clk
                bcf     REL_PORT,rel_clk

                bcf     REL_PORT,data_lo
                btfsc   SWITCH_LO,3
                bsf     REL_PORT,data_lo
                bcf     REL_PORT,data_hi
                btfsc   SWITCH_HI,3
                bsf     REL_PORT,data_hi
                bsf     REL_PORT,rel_clk
                bcf     REL_PORT,rel_clk

                bcf     REL_PORT,data_lo
                btfsc   SWITCH_LO,2
                bsf     REL_PORT,data_lo
                bcf     REL_PORT,data_hi
                btfsc   SWITCH_HI,2
                bsf     REL_PORT,data_hi
                bsf     REL_PORT,rel_clk
                bcf     REL_PORT,rel_clk

                bcf     REL_PORT,data_lo
                btfsc   SWITCH_LO,1
                bsf     REL_PORT,data_lo
                bcf     REL_PORT,data_hi
                btfsc   SWITCH_HI,1
                bsf     REL_PORT,data_hi
                bsf     REL_PORT,rel_clk
                bcf     REL_PORT,rel_clk

                bcf     REL_PORT,data_lo
                btfsc   SWITCH_LO,0
                bsf     REL_PORT,data_lo
                bcf     REL_PORT,data_hi
                btfsc   SWITCH_HI,0
                bsf     REL_PORT,data_hi
                bsf     REL_PORT,rel_clk
                bcf     REL_PORT,rel_clk

                bsf     REL_PORT,rel_latch          ;send loaded data to outputs
                bcf     REL_PORT,rel_latch

                clrf    TIMER                       ;clr timer after 7 lcd cycles
                goto    end_isr                     ;exec relays routine

lcd_handler     bcf     STATUS,C                    ;initializes and sends data
                rlf     LHCOUNT,W
                addwf   PCL,F
                movlw   b'00110000'                 ;8 bit mode
                goto    bit_8_load
                incf    LHCOUNT,F
                goto    end_isr
                incf    LHCOUNT,F
                goto    end_isr
                incf    LHCOUNT,F
                goto    end_isr
                incf    LHCOUNT,F
                goto    end_isr
                movlw   b'00110000'
                goto    bit_8_load
                incf    LHCOUNT,F
                goto    end_isr
                movlw   b'00110000'
                goto    bit_8_load
                movlw   b'00100000'                 ;4 bit mode
                goto    bit_8_load
                movlw   b'00100100'                 ;func set
                goto    bit_4_load
                movlw   b'00100000'                 ;func set
                goto    bit_4_load
                movlw   b'00000110'                 ;entry mode
                goto    bit_4_load
                movlw   b'00001100'                 ;display on
                goto    bit_4_load
                movlw   b'00010100'                 ;cursor shift
                goto    bit_4_load
                movlw   b'00000010'                 ;return home
                goto    bit_4_load
                incf    LHCOUNT,F
                goto    end_isr
                movlw   b'10000000'                 ;set ddram address
                goto    bit_4_load
                movf    DD01,W                      ;send buffer to lcd
                goto    disp_load
                movf    DD02,W
                goto    disp_load
                movf    DD03,W
                goto    disp_load
                movf    DD04,W
                goto    disp_load
                movf    DD05,W
                goto    disp_load
                movf    DD06,W
                goto    disp_load
                movf    DD07,W
                goto    disp_load
                movf    DD08,W
                goto    disp_load
                movf    DD09,W
                goto    disp_load
                movf    DD10,W
                goto    disp_load
                movf    DD11,W
                goto    disp_load
                movf    DD12,W
                goto    disp_load
                movf    DD13,W
                goto    disp_load
                movf    DD14,W
                goto    disp_load
                movf    DD15,W
                goto    disp_load
                movf    DD16,W
                goto    disp_load
                movf    DD17,W
                goto    disp_load
                movf    DD18,W
                goto    disp_load
                movf    DD19,W
                goto    disp_load
                movf    DD20,W
                goto    disp_load
                movf    DD21,W
                goto    disp_load
                movf    DD22,W
                goto    disp_load
                movf    DD23,W
                goto    disp_load
                movf    DD24,W
                goto    disp_load
                incf    LHCOUNT,F               ;after write refresh count
                goto    refr_count

                incf    LHCOUNT2,F              ;init at every 32 lcd_rout
                btfsc   LHCOUNT2,5
                goto    $+4
                movlw   d'14'                   ;data refresh start index
                movwf   LHCOUNT
                goto    end_isr
                clrf    LHCOUNT2
                clrf    LHCOUNT
                goto    end_isr

bit_8_load      bcf     LCD_DATA_P,lcd_rs       ;8bit command routine for lcd
                movwf   LCD_DATA_P
                bsf     LCD_DATA_P,lcd_e
                nop
                nop
                bcf     LCD_DATA_P,lcd_e

                incf    LHCOUNT,F
                goto    end_isr

bit_4_load      bcf     LCD_DATA_P,lcd_rs           ;4bit command routine for lcd
                goto    $+2

disp_load       bsf     LCD_DATA_P,lcd_rs           ;4bit data routine for lcd

                movwf   LD_TEMP
                movlw   b'11110000'
                andwf   LCD_DATA_P,F
                swapf   LD_TEMP,W
                andlw   b'00001111'
                iorwf   LCD_DATA_P,F
                bsf     LCD_DATA_P,lcd_e
                nop
                nop
                bcf     LCD_DATA_P,lcd_e
                movlw   b'11110000'
                andwf   LCD_DATA_P,F
                movf    LD_TEMP,W
                andlw   b'00001111'
                iorwf   LCD_DATA_P,F
                bsf     LCD_DATA_P,lcd_e
                nop
                nop
                bcf     LCD_DATA_P,lcd_e

                incf    LHCOUNT,F
                goto    end_isr

refr_count      swapf   COUNT_HI,W                  ;refresh display count value
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD01
                movf    COUNT_HI,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD02

                swapf   COUNT_MD,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD03
                movf    COUNT_MD,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD04

                swapf   COUNT_LO,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD05
                movf    COUNT_LO,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD06

                goto    end_isr

;*****************************************************************************
;interrupt routine for RB0 (clock input from the sensor)
;checks if on index else counts up or down then compares switching values
;if limit reached loads new data to comp registers and relays (switch_xx)

clock_int_isr   bcf     INTCON,INTF                 ;clear interrupt flag

                btfss   IN_PORT,index               ;check if on index
                goto    count_rout

                clrf    COUNT_HI
                clrf    COUNT_MD
                clrf    COUNT_LO
                clrf    COMP_INDEX
                clrf    COMP_HI
                clrf    COMP_MD
                clrf    COMP_LO
                clrf    COMP_ST
                clrf    SWITCH_HI
                clrf    SWITCH_LO

                call    load_comp_val

                goto    end_isr

count_rout      btfsc   IN_PORT,direction           ;check the direction of count
                goto    inc_count

dec_count       movlw   b'00001111'                 ;decrease the count
                andwf   COUNT_LO,W
                btfsc   STATUS,Z
                goto    $+3
                decf    COUNT_LO,F
                goto    compare
                movlw   0x09
                iorwf   COUNT_LO,F
                movlw   b'11110000'
                andwf   COUNT_LO,W
                btfsc   STATUS,Z
                goto    $+4
                movlw   0x10
                subwf   COUNT_LO,F
                goto    compare
                movlw   0x90
                iorwf   COUNT_LO,F

                movlw   b'00001111'
                andwf   COUNT_MD,W
                btfsc   STATUS,Z
                goto    $+3
                decf    COUNT_MD,F
                goto    compare
                movlw   0x09
                iorwf   COUNT_MD,F
                movlw   b'11110000'
                andwf   COUNT_MD,W
                btfsc   STATUS,Z
                goto    $+4
                movlw   0x10
                subwf   COUNT_MD,F
                goto    compare
                movlw   0x90
                iorwf   COUNT_MD,F

                movlw   b'00001111'
                andwf   COUNT_HI,W
                btfsc   STATUS,Z
                goto    $+3
                decf    COUNT_HI,F
                goto    compare
                movlw   0x09
                iorwf   COUNT_HI,F
                movlw   b'11110000'
                andwf   COUNT_HI,W
                btfsc   STATUS,Z
                goto    $+4
                movlw   0x10
                subwf   COUNT_HI,F
                goto    compare
                movlw   0x90
                iorwf   COUNT_HI,F
                goto    compare

inc_count       movlw   b'00001111'                     ;increase the count
                andwf   COUNT_LO,W
                sublw   0x09
                btfsc   STATUS,Z
                goto    $+3
                incf    COUNT_LO,F
                goto    compare
                movlw   b'11110000'
                andwf   COUNT_LO,F
                movlw   b'11110000'
                andwf   COUNT_LO,W
                sublw   0x90
                btfsc   STATUS,Z
                goto    $+4
                movlw   0x10
                addwf   COUNT_LO,F
                goto    compare
                movlw   b'00001111'
                andwf   COUNT_LO,F

                movlw   b'00001111'
                andwf   COUNT_MD,W
                sublw   0x09
                btfsc   STATUS,Z
                goto    $+3
                incf    COUNT_MD,F
                goto    compare
                movlw   b'11110000'
                andwf   COUNT_MD,F
                movlw   b'11110000'
                andwf   COUNT_MD,W
                sublw   0x90
                btfsc   STATUS,Z
                goto    $+4
                movlw   0x10
                addwf   COUNT_MD,F
                goto    compare
                movlw   b'00001111'
                andwf   COUNT_MD,F

                movlw   b'00001111'
                andwf   COUNT_HI,W
                sublw   0x09
                btfsc   STATUS,Z
                goto    $+3
                incf    COUNT_HI,F
                goto    compare
                movlw   b'11110000'
                andwf   COUNT_HI,F
                movlw   b'11110000'
                andwf   COUNT_HI,W
                sublw   0x90
                btfsc   STATUS,Z
                goto    $+4
                movlw   0x10
                addwf   COUNT_HI,F
                goto    compare
                movlw   b'00001111'
                andwf   COUNT_HI,F

compare         movf    COMP_HI,W                       ;comparison routine
                subwf   COUNT_HI,W
                btfss   STATUS,Z
                goto    not_equal
                movf    COMP_MD,W
                subwf   COUNT_MD,W
                btfss   STATUS,Z
                goto    not_equal
                movf    COMP_LO,W
                subwf   COUNT_LO,W
                btfss   STATUS,Z
                goto    not_equal

                bcf     STATUS,C                        ;equal then set relays
                bsf     PCLATH,0
                rlf     COMP_INDEX,W
                addwf   PCL,F
                bsf     SWITCH_LO,0
                goto    end_sw
                bsf     SWITCH_LO,1
                goto    end_sw
                bsf     SWITCH_LO,2
                goto    end_sw
                bsf     SWITCH_LO,3
                goto    end_sw
                bsf     SWITCH_LO,4
                goto    end_sw
                bsf     SWITCH_HI,0
                goto    end_sw
                bsf     SWITCH_HI,1
                goto    end_sw
                bsf     SWITCH_HI,2
                goto    end_sw
                bsf     SWITCH_HI,3
                goto    end_sw
                bsf     SWITCH_HI,4
                goto    end_sw
                bsf     SWITCH_LO,5
                goto    end_sw
                bsf     SWITCH_LO,6
                goto    end_sw
                bsf     SWITCH_LO,7
                goto    end_sw
                bsf     SWITCH_HI,5
                goto    end_sw
                bsf     SWITCH_HI,6
                goto    end_sw
                bsf     SWITCH_HI,7

end_sw          btfss   PREV_ST,0                   ;disables relays if prev needs
                goto    end_sw_off
                decf    COMP_INDEX,W
                movwf   TEMP_C_I
                bcf     STATUS,C
                rlf     TEMP_C_I,W
                addwf   PCL,F
                bcf     SWITCH_LO,0
                goto    end_sw_off
                bcf     SWITCH_LO,1
                goto    end_sw_off
                bcf     SWITCH_LO,2
                goto    end_sw_off
                bcf     SWITCH_LO,3
                goto    end_sw_off
                bcf     SWITCH_LO,4
                goto    end_sw_off
                bcf     SWITCH_HI,0
                goto    end_sw_off
                bcf     SWITCH_HI,1
                goto    end_sw_off
                bcf     SWITCH_HI,2
                goto    end_sw_off
                bcf     SWITCH_HI,3
                goto    end_sw_off
                bcf     SWITCH_HI,4
                goto    end_sw_off
                bcf     SWITCH_LO,5
                goto    end_sw_off
                bcf     SWITCH_LO,6
                goto    end_sw_off
                bcf     SWITCH_LO,7
                goto    end_sw_off
                bcf     SWITCH_HI,5
                goto    end_sw_off
                bcf     SWITCH_HI,6
                goto    end_sw_off
                bcf     SWITCH_HI,7

end_sw_off      bcf     PCLATH,0                ;for computed goto, set before

                incf    COMP_INDEX,F            ;new comparison values
                call    load_comp_val

not_equal       goto    end_isr

;****************************************************************************
;main routine, first makes the initializations then runs the prog section

main            call    delay00                 ;wait for power on delays

                clrf    PORTA

                movlw   a' '                    ;nothing but ' ' in display
                movwf   DD07

                clrf    LHCOUNT                 ;init lcd rout variable

                clrf    SWITCH_LO               ;disable all relays
                clrf    SWITCH_HI

                clrf    COUNT_HI                ;reset count value
                clrf    COUNT_MD
                clrf    COUNT_LO

                clrf    COMP_INDEX              ;set actual and previous
                clrf    COMP_HI                 ;comparison values
                clrf    COMP_MD
                clrf    COMP_LO
                clrf    COMP_ST
                call    load_comp_val

init_ports      bsf     STATUS,RP0              ;set PORTA as digital inputs
                movlw   0x06
                movwf   ADCON1
                movlw   b'00111100'             ;1 - input and 0 - output
                movwf   TRISA

                movlw   b'11111111'             ;set PORTB inputs and outputs
                movwf   TRISB

                movlw   b'11111111'             ;set PORTC inputs and outputs
```

```
            movwf  TRISC

            movlw  b'10000000'              ;set PORTD inputs and outputs
            movwf  TRISD

            movlw  b'00000111'              ;set PORTE inputs and outputs
            movwf  TRISE
            bcf    STATUS,RP0

init_int  clrf  TMR0                        ;assign prescaler to TMR0
            clrwdt                          ;prescaler ratio 1/32
            bsf    STATUS,RP0
            movlw  b'11000100'
            movwf  OPTION_REG
            bcf    STATUS,RP0

            movlw  b'10110000'              ;enable (RB0) and TMR0 int
            movwf  INTCON

prog_begin  call   next_rel                 ;display the next rel value

            call   ifreadmode               ;if readmode entered then
            btfss  STATUS,Z                 ;get rel # and show its limit
            goto   edit_mode

            call   msg_swnum
            call   getnumber
            movwf  EDIT_NUM

            call   getreadval
            call   putreadval

            call   view_delay000

edit_mode   call   ifeditmode               ;if editmode entered
            btfss  STATUS,Z
            goto   prog_begin

            call   msg_swnum                ;get the rel #
            call   getnumber
            movwf  EDIT_NUM

            call   msg_getlimit             ;get limit and status
            call   getnumber
            movwf  TEDIT1
            addlw  d'48'
            movwf  DD17
            call   getnumber
            movwf  TEDIT2
            addlw  d'48'
            movwf  DD18
            call   getnumber
            movwf  TEDIT3
            addlw  d'48'
            movwf  DD19
            call   getnumber
```

```
                movwf  TEDIT4
                addlw  d'48'
                movwf  DD20
                call   getnumber
                movwf  TEDIT5
                addlw  d'48'
                movwf  DD21
                call   getnumber
                movwf  TEDIT6
                addlw  d'48'
                movwf  DD22
                call   getnumber
                movwf  TEDITST
                btfss  TEDITST,0
                goto   $+4
                movlw  a't'
                movwf  DD24
                goto   $+3
                movlw  a'c'
                movwf  DD24

                call   iflimitvalid               ;if a valid value then set
                btfss  STATUS,Z
                goto   limiterr
                bcf    INTCON,GIE
                call   write_limit
                bsf    INTCON,GIE
                call   msg_wrt_ok
                call   view_delay000

                goto   prog_begin

limiterr  call  msg_limiterr
                call   view_delay000

                goto   prog_begin

;***************************************************************************

view_delay000   movlw  d'60'                      ;delay about 2.4 sec
                movwf  DELAY3                     ;@ 20 Mhz

delay000        decfsz DELAY1,F                   ;about 10 sec
                goto   $-1
                decfsz DELAY2,F
                goto   $-3
                decfsz DELAY3,F
                goto   $-5

                return

;***************************************************************************

m_delay00       movlw  d'100'                     ;about 16 ms
                movwf  DELAY2
```

```
delay00         decfsz  DELAY1,F                    ;about 40 ms
                goto    $-1
                decfsz  DELAY2,F
                goto    $-3

                return

;******************************************************************************

key_delay0      movlw   d'10'                       ;about 6 us
                movwf   DELAY1

delay0          decfsz  DELAY1,F                    ;about 154 us
                goto    $-1

                return

;******************************************************************************
;reads comparison values according to COMP_INDEX from EEPROM

load_comp_val   movf    COMP_HI,W                   ;make current values previous
                movwf   PREV_HI
                movf    COMP_MD,W
                movwf   PREV_MD
                movf    COMP_LO,W
                movwf   PREV_LO
                movf    COMP_ST,W
                movwf   PREV_ST

                bcf     STATUS,C                    ;load current comparison
                rlf     COMP_INDEX,W                ;values from EEPROM
                bsf     STATUS,RP1
                movwf   EEADR
                bcf     STATUS,C
                rlf     EEADR,F
                bsf     STATUS,RP0
                bcf     EECON1,EEPGD
                bsf     EECON1,RD
                bcf     STATUS,RP0
                movf    EEDATA,W
                bcf     STATUS,RP1
                movwf   COMP_HI

                bsf     STATUS,RP1
                bsf     EEADR,0
                bsf     STATUS,RP0
                bsf     EECON1,RD
                bcf     STATUS,RP0
                movf    EEDATA,W
                bcf     STATUS,RP1
                movwf   COMP_MD

                bsf     STATUS,RP1
                bcf     EEADR,0
                bsf     EEADR,1
                bsf     STATUS,RP0
                bsf     EECON1,RD
                bcf     STATUS,RP0
                movf    EEDATA,W
                bcf     STATUS,RP1
                movwf   COMP_LO

                bsf     STATUS,RP1
                bsf     EEADR,0
                bsf     STATUS,RP0
                bsf     EECON1,RD
                bcf     STATUS,RP0
                movf    EEDATA,W
                bcf     STATUS,RP1
                movwf   COMP_ST

                return

;*****************************************************************************
;look if '#' pressed

ifeditmode      clrf    KEY_OUTPORT

                bsf     KEY_OUTPORT,0
                call    key_delay0

                bcf     STATUS,Z
                btfsc   KEY_INPORT,6
                bsf     STATUS,Z

                return

;*****************************************************************************
; look if '*' pressed

ifreadmode      clrf    KEY_OUTPORT

                bsf     KEY_OUTPORT,0
                call    key_delay0

                bcf     STATUS,Z
                btfsc   KEY_INPORT,4
                bsf     STATUS,Z

                return

;*****************************************************************************
;read the key pressed and return the value

getnumber       movlw   b'00001111'
                iorwf   KEY_OUTPORT,F
                movf    KEY_INPORT,W
                andlw   b'01110000'
                btfss   STATUS,Z
                goto    getnumber
                call    delay00
```

```
              clrf    KEY_OUTPORT

              bsf     KEY_OUTPORT,3
              call    key_delay0
              btfsc   KEY_INPORT,4
              retlw   1
              btfsc   KEY_INPORT,5
              retlw   2
              btfsc   KEY_INPORT,6
              retlw   3
              bcf     KEY_OUTPORT,3

              bsf     KEY_OUTPORT,2
              call    key_delay0
              btfsc   KEY_INPORT,4
              retlw   4
              btfsc   KEY_INPORT,5
              retlw   5
              btfsc   KEY_INPORT,6
              retlw   6
              bcf     KEY_OUTPORT,2

              bsf     KEY_OUTPORT,1
              call    key_delay0
              btfsc   KEY_INPORT,4
              retlw   7
              btfsc   KEY_INPORT,5
              retlw   8
              btfsc   KEY_INPORT,6
              retlw   9
              bcf     KEY_OUTPORT,1

              bsf     KEY_OUTPORT,0
              call    key_delay0
              btfsc   KEY_INPORT,5
              retlw   0
              bcf     KEY_OUTPORT,0

              goto    getnumber

;****************************************************************************

msg_limiterr  movlw   a'd'
              movwf   DD08
              movlw   a'a'
              movwf   DD09
              movlw   a't'
              movwf   DD10
              movlw   a'a'
              movwf   DD11
              movlw   a'_'
              movwf   DD12
              movlw   a'e'
              movwf   DD13
              movlw   a'r'
              movwf   DD14
```

```
            movlw  a'r'
            movwf  DD15

            return

;***************************************************************************

msg_swnum   movlw  a'e'
            movwf  DD08
            movlw  a'n'
            movwf  DD09
            movlw  a't'
            movwf  DD10
            movlw  a'e'
            movwf  DD11
            movlw  a'r'
            movwf  DD12
            movlw  a' '
            movwf  DD13
            movlw  a'r'
            movwf  DD14
            movlw  a'e'
            movwf  DD15
            movlw  a'l'
            movwf  DD16
            movlw  a'a'
            movwf  DD17
            movlw  a'i'
            movwf  DD18
            movlw  a's'
            movwf  DD19
            movlw  a' '
            movwf  DD20
            movlw  a'n'
            movwf  DD21
            movlw  a'u'
            movwf  DD22
            movlw  a'm'
            movwf  DD23
            movlw  a' '
            movwf  DD24

            return

;***************************************************************************
;reads limit values according to EDIT_NUM for displaying

getreadval  movf   EDIT_NUM,W
            movwf  ADDR_EE
            bcf    STATUS,C
            rlf    ADDR_EE,F
            rlf    ADDR_EE,F
            call   read_eeprom
            movwf  TEDIT1
            incf   ADDR_EE,F
            call   read_eeprom
```

```
                movwf  TEDIT3
                incf   ADDR_EE,F
                call   read_eeprom
                movwf  TEDIT5
                incf   ADDR_EE,F
                call   read_eeprom
                movwf  TEDITST

                return

;******************************************************************************
;eeprom reading by ADRESS and return in W

read_eeprom     bsf    STATUS,RP1
                movf   ADDR_EE,W
                movwf  EEADR
                bsf    STATUS,RP0
                bcf    EECON1,EEPGD
                bsf    EECON1,RD
                bcf    STATUS,RP0
                movf   EEDATA,W
                bcf    STATUS,RP1

                return

;******************************************************************************
;display the value of limit of the selected relay @ readmode

putreadval      movlw  a'r'
                movwf  DD08
                movlw  a'e'
                movwf  DD09
                movlw  a'a'
                movwf  DD10
                movlw  a'd'
                movwf  DD11
                movlw  a'm'
                movwf  DD12
                movlw  a'o'
                movwf  DD13
                movlw  a'd'
                movwf  DD14
                movlw  a'e'
                movwf  DD15
                movlw  a' '
                movwf  DD16

                movf   EDIT_NUM,W                   ;add 65 to convert to chr
                addlw  d'65'
                movwf  DD17

                swapf  TEDIT1,W                     ;add 48 for decimal ascii
                andlw  b'00001111'
                addlw  d'48'
                movwf  DD18
                movf   TEDIT1,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD19

                swapf   TEDIT3,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD20
                movf    TEDIT3,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD21

                swapf   TEDIT5,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD22
                movf    TEDIT5,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD23

                btfss   TEDITST,0                       ;print status of relays
                goto    $+4                             ;0 continuous, 1 temporary

                movlw   a't'
                movwf   DD24
                return

                movlw   a'c'
                movwf   DD24
                return

;***************************************************************************
;display the value of limit of the next relay

next_rel movlw  a'n'
                movwf   DD08
                movlw   a'e'
                movwf   DD09
                movlw   a'x'
                movwf   DD10
                movlw   a't'
                movwf   DD11
                movlw   a'_'
                movwf   DD12
                movlw   a'r'
                movwf   DD13
                movlw   a'e'
                movwf   DD14
                movlw   a'l'
                movwf   DD15
                movlw   a' '
                movwf   DD16

                movf    COMP_INDEX,W
                addlw   d'65'
                movwf   DD17

                swapf   COMP_HI,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD18
                movf    COMP_HI,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD19

                swapf   COMP_MD,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD20
                movf    COMP_MD,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD21

                swapf   COMP_LO,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD22
                movf    COMP_LO,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD23

                btfss   COMP_ST,0
                goto    $+4

                movlw   a't'
                movwf   DD24
                return

                movlw   a'c'
                movwf   DD24
                return

;****************************************************************************
;print display 'XXXXXX sw_lim_X ______@_'

msg_getlimit    movlw   a's'
                movwf   DD08
                movlw   a'w'
                movwf   DD09
                movlw   a'_'
                movwf   DD10
                movlw   a'l'
                movwf   DD11
                movlw   a'i'
                movwf   DD12
                movlw   a'm'
                movwf   DD13
                movlw   a'_'
                movwf   DD14
                movf    EDIT_NUM,W
                addlw   d'65'
                movwf   DD15
                movlw   a' '
                movwf   DD16
                movlw   a'_'
                movwf   DD17
                movlw   a'_'
                movwf   DD18
                movlw   a'_'
                movwf   DD19
                movlw   a'_'
                movwf   DD20
                movlw   a'_'
                movwf   DD21
                movlw   a'_'
                movwf   DD22
                movlw   a'@'
                movwf   DD23
                movlw   a'_'
                movwf   DD24

                return

;***************************************************************************
;print display 'XXXXXX Write_OK XXXXXXXX'

msg_wrt_ok      movlw   a'W'
                movwf   DD08
                movlw   a'r'
                movwf   DD09
                movlw   a'i'
                movwf   DD10
                movlw   a't'
                movwf   DD11
                movlw   a'e'
                movwf   DD12
                movlw   a'_'
                movwf   DD13
                movlw   a'O'
                movwf   DD14
                movlw   a'K'
                movwf   DD15
                movlw   a' '
                movwf   DD16

                movf    EDIT_NUM,W
                addlw   d'65'
                movwf   DD17

                swapf   TEDIT1,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD18
                movf    TEDIT1,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD19

                swapf   TEDIT3,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD20
                movf    TEDIT3,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD21

                swapf   TEDIT5,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD22
                movf    TEDIT5,W
                andlw   b'00001111'
                addlw   d'48'
                movwf   DD23

                btfss   TEDITST,0
                goto    $+4

                movlw   a't'
                movwf   DD24
                return

                movlw   a'c'
                movwf   DD24
                return

;******************************************************************************
;arrange the key input to TEDIT 1-3-5 HI-MD-LO
;then load upper and lower boundaries, compare and return if valid

iflimitvalid    swapf   TEDIT1,F
                movf    TEDIT2,W
                addwf   TEDIT1,F

                swapf   TEDIT3,F
                movf    TEDIT4,W
                addwf   TEDIT3,F

                swapf   TEDIT5,F
                movf    TEDIT6,W
                addwf   TEDIT5,F

                movf    EDIT_NUM,W
                btfss   STATUS,Z
                goto    $+7
                clrf    LOWER_HI
                clrf    LOWER_MD
                clrf    LOWER_LO
                movlw   1
                call    upperlimit
                goto    comp_valid

                movf    EDIT_NUM,W
                sublw   d'15'
                btfss   STATUS,Z
                goto    $+8
                movlw   d'14'
                call    lowerlimit
                movlw   0x99
                movwf   UPPER_HI
                movwf   UPPER_MD
                movwf   UPPER_LO
                goto    comp_valid

                decf    EDIT_NUM,W
                call    lowerlimit
                incf    EDIT_NUM,W
                call    upperlimit

comp_valid      movf    TEDIT1,W
                subwf   LOWER_HI,W
                btfss   STATUS,C
                goto    valid_lo
                btfsc   STATUS,Z
                goto    $+2
                goto    invalid

                movf    TEDIT3,W
                subwf   LOWER_MD,W
                btfss   STATUS,C
                goto    valid_lo
                btfsc   STATUS,Z
                goto    $+2
                goto    invalid

                movf    TEDIT5,W
                subwf   LOWER_LO,W
                btfss   STATUS,C
                goto    valid_lo
                goto    invalid
valid_lo
                movf    UPPER_HI,W
                subwf   TEDIT1,W
                btfss   STATUS,C
                goto    valid_hi
                btfsc   STATUS,Z
                goto    $+2
                goto    invalid

                movf    UPPER_MD,W
                subwf   TEDIT3,W
                btfss   STATUS,C
                goto    valid_hi
                btfsc   STATUS,Z
                goto    $+2
                goto    invalid

                movf    UPPER_LO,W
                subwf   TEDIT5,W
                btfss   STATUS,C
                goto    valid_hi

invalid         bcf     STATUS,Z
                return

valid_hi bsf    STATUS,Z
                return

;*****************************************************************************

;read upperlimit from EEPROM to upper registers

upperlimit      movwf   ADDR_EE
                bcf     STATUS,C
                rlf     ADDR_EE,F
                rlf     ADDR_EE,F
                call    read_eeprom
                movwf   UPPER_HI
                incf    ADDR_EE,F
                call    read_eeprom
                movwf   UPPER_MD
                incf    ADDR_EE,F
                call    read_eeprom
                movwf   UPPER_LO

                return

;*****************************************************************************
;read lowerlimit from EEPROM to lower registers

lowerlimit      movwf   ADDR_EE
                bcf     STATUS,C
                rlf     ADDR_EE,F
                rlf     ADDR_EE,F
                call    read_eeprom
                movwf   LOWER_HI
                incf    ADDR_EE,F
                call    read_eeprom
                movwf   LOWER_MD
                incf    ADDR_EE,F
                call    read_eeprom
                movwf   LOWER_LO

                return

;*****************************************************************************
;writes data to EEPROM from arranged data of iflimitvalid

write_limit     movf    EDIT_NUM,W
                movwf   ADDR_EE
                bcf     STATUS,C
                rlf     ADDR_EE,F
                rlf     ADDR_EE,F
                movf    TEDIT1,W
                call    write_eeprom
                incf    ADDR_EE,F
                movf    TEDIT3,W
                call    write_eeprom
                incf    ADDR_EE,F
                movf    TEDIT5,W
                call    write_eeprom
                incf    ADDR_EE,F
                movf    TEDITST,W
                call    write_eeprom

                return

;*****************************************************************************
;EEPROM write cycle

write_eeprom    movwf   DATA_EE
                bsf     STATUS,RP0
                bsf     STATUS,RP1
                btfsc   EECON1,WR
                goto    $-1
                bcf     STATUS,RP0
                movf    ADDR_EE,W
                movwf   EEADR
                movf    DATA_EE,W
                movwf   EEDATA
                bsf     STATUS,RP0
                bcf     EECON1,EEPGD
                bsf     EECON1,WREN
                movlw   0x55
                movwf   EECON2
                movlw   0xAA
                movwf   EECON2
                bsf     EECON1,WR
                bcf     EECON1,WREN
                bcf     STATUS,RP0
                bcf     STATUS,RP1

                return

;*****************************************************************************

        END                             ;directive 'end of program'
```

APPENDIX B

INSTRUCTION SET OF THE PIC 16F877

1. addlw number
adds a number with the number in the working register.

2. addwf FileReg, f
adds the number in the working register to the number in a file register (FileReg) and puts the result in the file register.

addwf FileReg, w
adds the number in the working register to the number in a file register (FileReg) and puts the result back into the working register, leaving the file register unchanged.

3. andlw number
ANDs a number with the number in the working register, leaving the result in the working reg.

4. andwf FileReg, f
ANDs the number in the working register with the number in a file register (FileReg) and puts the result in the file register

5. bcf FileReg, bit
clears a bit in a file register (FileReg), i.e. makes the bit 0

6. bsf FileReg, bit
sets a bit in a file register (FileReg), i.e. makes the bit 1

7. btfsc FileReg, bit

tests a bit in a file register (FileReg) and skips the next instruction if the result is clear (i.e. if that bit is 0).

8. btfss FileReg, bit

tests a bit in a file register (FileReg) and skips the next instruction if the result is set (i.e. if that bit is 1).

9. call AnySub

makes the chip call a subroutine(AnySub), after which it will return to where it left off.

10. clrf FileReg

clears (makes 0) the number in a file register (FileReg).

11. clrw

clears the working register.

12. clrwdt

clears the watchdog timer.

13. comf FileReg, f

complements (inverts - ones become zeroes, zeroes become ones) the number in a file register (FileReg), leaving the result in the file register.

14. decf FileReg, f

decrements (subtracts one from) a file register (FileReg) and puts the result in the file register.

15. decfsz FileReg, f

decrements a file register (FileReg) and if the result is zero it skips the next instruction. The result is put in the file register.

16. goto Anywhere

makes the chip go to somewhere in the program which have been labelled 'Anywhere'.

17. incf FileReg, f

increments (adds one to) a file register (FileReg) and puts the result in the file register.

18. incfsz FileReg, f

increments a file register (FileReg) and if the result is zero it skips the next instruction. The result is put in the file register.

19. iorlw number

inclusive ORs a number with the number in the working register.

20. iorwf FileReg, f

inclusive ORs the number in the working register with the number in a file register (FileReg) and puts the result in the file register.

21. movfw FileReg or movf FileReg, w

moves (copies) the number in a file register (FileReg) in the working register

22. movlw number

moves (copies) a number into the working register.

23. movwf FileReg

moves (copies) the number in the working register into a file register (FileReg).

24. nop

this stands for : no operation, in other words - do nothing.

25. retfie

returns from a subroutine and enables the Global Interrupt Enable bit.

26. retlw number

returns from a subroutine with a particular number (literal) in the working register.

27. return

returns from a subroutine.

28. rlf FileReg, f

rotates the bits in a file register (FileReg) to the left, putting the result in the file register.

29. rrf FileReg, f

rotates the bits in a file register (FileReg) to the right, putting the result in the file register.

30. sleep

sends the PIC to sleep, a lower power consumption mode.

31. sublw number

subtracts the number in the working register from a number.

32. subwf FileReg, f

subtracts the number in the working register from the number in a file register (FileReg) and puts the result in the file register.

33. swapf FileReg, f

swaps the two halves of the 8 bit binary number in a file register (FileReg), leaving the result in the file register.

34. xorlw number

exclusive ORs a number with the number in the working register.

35. xorwf FileReg, f

exclusive ORs the number in the working register with the number in a file register (FileReg) and puts the result in the file register.

APPENDIX C

MPASM DIRECTIVE SUMMARY

1. **_ _ BADRAM –** Specify invalid RAM locations
2. **BANKISEL –** Generate RAM bank selecting code for indirect addressing
3. **BANKSEL –** Generate RAM bank selecting code
4. **CBLOCK –** Define a Block of Constants
5. **CODE –** Begins executable code section
6. **_ _ CONFIG –** Specify configuration bits
7. **CONSTANT –** Declare Symbol Constant
8. **DA –** Store Strings in Program Memory
9. **DATA –** Create Numeric and Text Data
10. **DB –** Declare Data of One Byte
11. **DE –** Define EEPROM Data
12. **#DEFINE –** Define a Text Substitution Label
13. **DT –** Define Table
14. **DW –** Declare Data of One Word
15. **ELSE –** Begin Alternative Assembly Block to IF
16. **END –** End Program Block
17. **ENDC –** End an Automatic Constant Block
18. **ENDIF –** End conditional Assembly Block
19. **ENDM –** End a Macro Definition
20. **ENDW –** End a While Loop
21. **EQU –** Define an Assembly Constant
22. **ERROR –** Issue an Error Message
23. **ERRORLEVEL –** Set Error Level

24. EXITM – Exit from a Macro

25. EXPAND – Expand Macro Listing

26. EXTERN – Declares an external label

27. FILL – Fill Memory

28. GLOBAL – Exports a defined label

29. IDATA – Begin an Object File Initialized Data Section

30. _ _IDLOCS – Set Processor ID Locations

31. IF – Begin Conditionally Assembled Code Block

32. IFDEF – Execute If Symbol has Been Defined

33. IFNDEF – Execute If Symbol has not Been Defined

34. INCLUDE – Include Additional Source File

35. LIST – Listing Options

36. LOCAL – Declare Local Macro Variable

37. MACRO – Declare Macro Definition

38. _ _MAXRAM – Define Maximum RAM Location

39. MESSG – Create User Defined Message

40. NOEXPAND – Turn off Macro Expansion

41. NOLIST – Turn off Listing Output

42. ORG – Set Program Origin

43. PAGE – Insert Listing Page Eject

44. PAGESEL – Generate Page Selecting Code

45. PROCESSOR – Set Processor Type

46. RADIX – Specify Default Radix

47. RES – Reserve Memory

48. SET – Define an Assembler Variable

49. SPACE – Insert Blank Listing Lines

50. SUBTITLE – Specify Program Subtitle

51. TITLE – Specify Program Title

52. UDATA – Begin an Object File Uninitialized Data Section

53. UDATA_ACS – Begin an Object File Access Uninitialized Data Section

54. UDATA_OVR – Begin an Object File Overlayed Uninitialized Data Section

55. UDATA_SHR – Begin an Object File Shared Uninitialized Data Section

56. #UNDEFINE – Delete a Substitution Label

57. VARIABLE – Declare Symbol Variable

58. WHILE – Perform Loop While Condition is True