



NEAR EAST UNIVERSITY

**GRADUATE SCHOOL OF APPLIED AND SOCIAL
SCIENCES**

**MICROCONTROLLER BASED AQUARIUM
CONTROL SYSTEM**

Mahmut Kısacık

Master Thesis

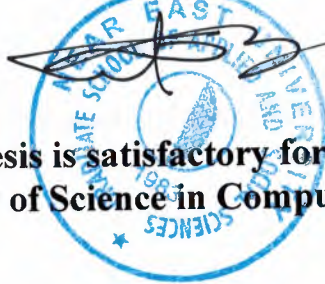
Department of Computer Engineering

Nicosia - 2006

**Approval of the Graduate School of Applied and
Social Sciences**


Prof. Dr. Fakhraddin Mamedov

Director





**We certify this thesis is satisfactory for the award of the
Degree of Master of Science in Computer Engineering**

Examining Committee in charge:


Assist. Prof. Dr. Kadri Buruncuk, Chairman , Electrical Engineering
Department, NEU

Assoc. Prof. Dr. Adil Amircanov, Member , Computer Engineering
Department, NEU


Assist. Prof. Dr. Özgür Özerdem, Member, Electrical Engineering
Department, NEU


Prof. Dr. Doğan Ibrahim, Supervisor, Computer Engineering
Department, NEU

ACKNOWLEDGEMENTS

First, I am very grateful to my family and my wife for their indulgence.

Second, I would like to thank my supervisor Prof. Dr. Doğan İbrahim for his invaluable advice and belief in my work and myself over the course of this MSc. Degree.

Finally, I would also like to thank all my friends for their help to collect my simulation parts.

ABSTRACT

This thesis is about the microcontroller based control of an aquarium. The thesis describes how to control the main three elements of an aquarium: the temperature, feeding the fish, and the lights. A temperature control system has been developed to keep the aquarium water temperature at the required level. Also, a microcontroller based automatic fish feeding mechanism has been developed which delivers food to the fish every day, at pre-specified times of the day. Proper lighting of an aquarium is very important for the healthy living of the fish and the plants inside the aquarium. The thesis describe the development of a microcontroller based lighting system which controls the aquarium lights every day, at pre-specified times of the day. The Peripheral Interface Controller (PIC) microcontrollers are very popular and these microcontrollers are used in the thesis.

The hardware of the system was built on several Personal Control Blocks (PCB), and the software has been developed using the assembler language of the PIC microcontrollers. The system developed was connected to real aquariums and its performance tested. The results obtained were highly satisfactory since a complete aquarium could be controlled automatically without any human intervention. The only maintenance required was to change the water at weekly intervals.

CONTENTS

ACKNOWLEDGEMENT.....	i
ABSTRACT.....	ii
CONTENTS.....	iii
LIST OF FIGURES.....	vii
LIST OF TABLES.....	ix
CHAPTER 1	
1. INTRODUCTION.....	1
CHAPTER 2	
2. MICROCOMPUTER SYSTEMS.....	3
2.1. Introduction to Microcomputer Systems.....	3
2.2. Microcontroller Systems.....	4
2.3. Why Microcontrollers are Used?.....	11
2.4. Microcontroller Memories.....	12
2.4.1. RAM.....	12
2.4.2. ROM.....	13
2.4.3. EPROM.....	13
2.4.4. EEPROM.....	13
2.4.5. Flash EEPROM.....	14
2.5. Microcontroller Features.....	14
2.5.1. Supply Voltage.....	14
2.5.2. The Clock.....	15
2.5.3. Timers.....	15
2.5.4. Watchdog.....	16

2.5.5. Reset Input.....	16
2.5.6. Interrupts.....	17
2.5.7. Brown-out Detector.....	17
2.5.8. Analogue-to-Digital Converter.....	18
2.5.9. Serial Input-Output.....	18
2.5.10. EEPROM Data Memory.....	19
2.5.11. LCD Drivers.....	19
2.5.12. Analogue Comparator.....	20
2.5.13. Real-time Clock.....	20
2.5.14. Sleep Mode.....	20
2.5.15. Power-on Reset.....	20
2.5.16. Low Power Operation.....	21
2.5.17. Current Sink/Source Capability.....	21
2.6. Microcontroller Architectures.....	21
2.6.1. RISC and CISC.....	22
2.7. Examples of Microcontrollers.....	23
2.8. PIC Microcontrollers.....	23
2.8.1. 12-bit Instruction word.....	27
2.8.2. 14-bit Instruction word.....	28
2.8.3. 16-bit Instruction word.....	35
2.9. Inside a PIC microcontroller.....	35
2.9.1. Program memory (Flash).....	35
2.9.2. Data memory (RAM).....	36
2.9.3. Register file map and special function registers.....	36
2.9.4. Option register.....	37

2.9.5. I/O Registers.....	39
2.9.6. Timer registers.....	40
2.9.7. TMR0 and Watchdog.....	41
2.9.8. INTCON Register.....	44
2.9.9. A/D Converter Registers.....	46
2.9.10. Oscillator circuits.....	52
2.9.11. Crystal operation.....	52
2.9.12. Resonator operation.....	53
2.9.13. RC oscillator.....	54
2.9.14. Reset Circuit.....	54
2.9.15. Interrupts.....	56
2.9.16. The configuration word.....	57
2.10. Hardware Programmers and Software Programs Used.....	58
2.11 The MPLAB Assembler.....	60
2.12 The Commands of MPASM Assembler.....	60

CHAPTER 3

3. AQUARIUMS.....	65
3.1. Types of Aquariums.....	65
3.2. Aquarium Material.....	68
3.3. Size.....	69
3.4. Cycling.....	70
3.5. Aquarium Furnishings.....	71
3.6. Heater.....	72
3.7. Air Pump.....	72
3.8. Filtration.....	72

3.9. Bucket and Siphon.....	73
3.10. Setting Up Equipment.....	73
3.10.1. Location of the Aquarium.....	74

CHAPTER 4

4. DEVELOPMENT OF A MICROCONTROLLER BASED AUTOMATIC AQUARIUM SYSTEM.....	75
4.1. Overview.....	75
4.1.1. The water temperature.....	75
4.1.2. Feeding the fish.....	76
4.1.3. Aquarium lighting.....	77
4.1.4. Cleaning the aquarium.....	78
4.1.5. Designing a computer based aquarium system.....	78
4.2. Development of the Aquarium Control System.....	82
4.2.1. The clock Microcontroller.....	84
4.2.2 Date Microcontrollers.....	86
4.2.3 Fish Feeding and Light Control.....	87
4.2.4. Ligth Flasher Display.....	89
4.2.5. Feed Flasher Display.....	89
4.2.6. Controlling the Aquarium Water Temperature.....	89
4.2.7. Results.....	96

CHAPTER 5

5. CONCLUSION.....	100
REFERENCES.....	102

APPENDICES

Appendix 1 – Year Program.....	103
--------------------------------	-----

Appendix 2 – Month and Day program.....	115
Appendix 3 – Clock Program.....	126
Appendix 4 – Light on-off flasher Program.....	134
Appendix 5 – Feed on-off flasher Program.....	147
Appendix 6 – Light and Feeding on-off Program.....	160
Appendix 7 – Digital thermometer and heater control program	167

LIST OF FIGURES

Figure 2.1.	Embedded microcontroller system	5
Figure 2.2.	Some microcontrollers.....	6
Figure 2.3.	Microcontroller based oven temperature control system.....	7
Figure 2.4.	Temperature control system with a keypad and LCD.....	8
Figure 2.5.	More sophisticated temperature controller.....	9
Figure 2.6.	The simplest microcontroller architecture.....	10
Figure 2.7.	The main idea of a microcontroller.....	11
Figure 2.8.	Von Neumann and Harvard architectures.....	22
Figure 2.9.	Pin-out diagram of PIC16F84.....	29
Figure 2.10.	Memory map of the PIC16F84 microcontroller.....	30
Figure 2.11.	Block diagram of the PIC16F84 microcontroller.....	31
Figure 2.12.	Pin-out diagram of the PIC16F877 microcontroller.....	32
Figure 2.13.	Block diagram of the PIC16F877 microcontroller.....	33
Figure 2.14.	Memory map of the PIC16F877 microcontroller.....	34
Figure 2.15.	OPTION_REG bit definitions.....	38
Figure 2.16.	TMR0 and watchdog circuit.....	42
Figure 2.17.	INTCON register bit definitions.....	44
Figure 2.18.	Multiplexed A/D structure.....	47
Figure 2.19.	ADCON0 bit definitions.....	47
Figure 2.20.	ADCON1 bit definitions.....	51
Figure 2.21.	Crystal oscillator circuit.....	53
Figure 2.22.	Resonator oscillator circuit.....	53

Figure 2.23.	RC oscillator circuit.....	54
Figure 2.24.	Using the power on reset.....	55
Figure 2.25.	Using an external reset button.....	56
Figure 2.26.	Usb PIC programmer photo.....	59
Figure 3.1.	Some type of aquariums.....	66
Figure 4.1.	Block diagram of the BAMU aquarium control system.....	79
Figure 4.2.	<i>Automated Aquarium Systems Inc.</i> aquarium control system.....	81
Figure 4.3.	<i>Neptun Systems</i> aquarium control system.....	82
Figure 4.4.	Block diagram of the system.....	83
Figure 4.5.	Block diagram of the Clock circuit.....	85
Figure 4.6.	Circuit diagram of the Clock circuit.....	85
Figure 4.7.	Block diagram of the fish feeding and light control.....	87
Figure 4.8.	Circuit diagram of the fish feeding and light control circuit.....	88
Figure 4.9.	Block diagram of the temperature control system.....	90
Figure 4.10.	Circuit diagram of the temperature control system.....	91
Figure 4.11.	Picture of the aquarium control system.....	95
Figure 4.12.	Picture of the aquarium control system.....	96

LIST OF TABLES

Table 4.1.	Aquarium lighting and feeding times.....	89
Table 5.1.	Results of the measurement using a real aquarium.....	99
Table 5.2.	Advantages and disadvantages of the thesis.....	99

CHAPTER 1

1. INTRODUCTION

Aquariums are very good adornments. They are placed in the best place in the houses, offices, restaurants, hotels etc. Aquariums make people relax, but setting up the ideal aquarium environment can be a challenge since there are fundamental principles about taking proper care of the fish that must be taken into serious consideration. It is necessary to exercise the utmost care in creating and maintaining an environmental balance. Most people like aquariums but they do not have the time to care for their aquariums. Aquariums need care throughout the year and they can not be left on their own when, for example, the owner goes on holiday.

The aim of this thesis is to automate the aquarium maintenance process using microcontroller based control systems. Popular PIC series of microcontrollers are used in the thesis. These are small integrated circuits in the form of single-chip computers with memories, timers, interrupt circuits, and, serial and parallel I/O. The thesis describes the development of control systems to control the 3 basic elements of an aquarium: the lighting control, the feeding control, and the temperature control of the aquarium water. Fish need certain amount of light each day for their healthy living. A light control system has been developed which turns the aquarium lights on and off at pre-defined times of the day. Fish also need food daily and a microcontroller based feeding system has been developed which provide the correct amount of food to the fish at pre-defined times of each day. Finally, the temperature of an aquarium is very important for the healthy living of its inhabitants (e.g. fish and plants). A microcontroller based temperature control system has been developed which keeps the aquarium temperature at a pre-defined value through the year.

The thesis is organized as follows: In Chapter 2, detailed explanation of microcomputer systems and the main parts of these systems are given.

Chapter 3 covers the explanation of aquarium types, parts of an aquarium, and the basic maintenance of aquariums. Also, in this chapter explains the aquarium materials, sizes, cycling, aquarium furnishings, heater, air pump, filtration and setting up equipment.

Chapter 4 the development of a microcontroller based aquarium system is described in detail. The described things are the water temperature, feeding the fish, aquarium lighting, cleaning the aquarium. The development of the aquarium control system describes the clock microcontroller, date microcontrollers, fish feeding and light control, light flasher display, feed flasher display, controlling the aquarium water temperature.

In chapter 5, the thesis results are described

Finally, in chapter 6, the conclusion and suggestions for future work are given at the end of the thesis.

CHAPTER 2

2. MICROCOMPUTER SYSTEMS

2.1. Introduction to Microcomputer Systems

In 1969 Bob Noyce and Gordon Moore set up the Intel Corporation to manufacture memory chips for the mainframe computer industry. Later in 1971, the first microprocessor chip 4040 was manufactured by Intel for a consortium of two Japanese companies. These chips were basically designed for a calculator named *Busicom* which was one of the first portable calculators. This was a very simple calculator which could only add and subtract numbers, 4 bits (a nibble) at a time. 4040 chip was so successful that it was soon followed by Intel's 8-bit 8008 microprocessor. This was a simple microprocessor with limited resources, poorly implemented interrupt mechanisms, and multiplexed address and data busses. This microprocessor had separate address and data busses with 64K byte of address space which was enormous in 1975 standards. 8080 microprocessor was the first microprocessor used in homes as a personal computer named *Altair*[12]. 8080 has been a very successful microprocessor but soon other companies began producing microprocessor chips. Motorola introduced the 8-bit 6800 chip which had a different architecture to the 8080 but has also been very popular. In 1976 Zilog introduced the Z80 microprocessor which was much more advanced than the 8080. The instruction set of Z80 was downward compatible with the 8080 and this made Z80 to be one of the most successful microprocessors of the time. Z80 was used in many microprocessor based applications, including home computers and games consoles. In 1976 Motorola created a microprocessor chip called 6801 which replaced a 6800 chip plus some of the chips required to make a complete computer system. This was a major step in the evolution of the microcontrollers which

are basically computers consisting of only one chip. In later years, we see many other microcontroller chips in the market, such as Intel 8048, 8049, 8051, Motorola 6809, Atmel 89C51 etc.

The term microcomputer is used to describe a system that includes a minimum of a microprocessor, program memory, data memory, and input-output (I/O)[1]. Some microcomputer systems include additional components such as timers, counters, analogue-to-digital converters, and so on. Thus, a microcomputer system can be anything from a large computer having hard disks, floppy disks, and printers, to a single chip embedded controller.

The consideration of the type of microcomputers that consists of a single silicon chip. Such microcomputer systems are also called microcontrollers and they are used in many household goods such as microwave ovens, TV remote control units, cookers, hi-fi equipment, CD players, personal computers, fridges, watering plants, etc.

2.2. Microcontroller Systems

A microcontroller is a single chip computer (see Figure 2.1). *Micro* suggests that the device is small, and *controller* suggests that the device can be used in control applications. Another term used for microcontrollers is *embedded controller*, since most of the microcontrollers are built into (or embedded in) the devices they control. An example embedded microcontroller system is shown in Figure 2.1.

A microprocessor differs from a microcontroller in many ways. The main difference is that a microprocessor requires several other components for its operation, such as program memory and data memory, input-output devices, and external clock

circuit. A microcontroller on the other hand has all the support chips incorporated inside the same chip. All microcontrollers operate on a set of instructions (or the user program) stored in their memory. A microcontroller fetches the instructions from its program memory one by one, decodes these instructions, and then carries out the required operations.

Microcontrollers have traditionally been programmed using the assembly language of the target device. Although the assembly language is fast, it has several disadvantages. An assembly program consists of mnemonics and it is difficult to learn and maintain a program written using the assembly language. Also, microcontrollers manufactured by different firms have different assembly languages and the user is required to learn a new language every time a new microcontroller is to be used. Microcontrollers can also be programmed using a high-level language, such as BASIC, PASCAL, and C. High-level languages have the advantage that it is much easier to learn a high-level language than the assembler. Also, very large and complex programs can easily be developed using a high-level language.



Figure 2.1.: Embedded microcontroller system

In general, a single chip is all that is required to have a running microcontroller system. In practical applications additional components may be required to allow a

microcomputer to interface to its environment. With the advent of the PIC family of microcontrollers the development time of an electronic project has reduced to several hours. Developing a PIC microcontroller based project simply takes no more than five or six steps:

- Type the program into a PC
- Assemble (or compile) the program
- Optionally simulate the program on a PC
- Load the program into PIC's program memory
- Design the hardware
- Test the project

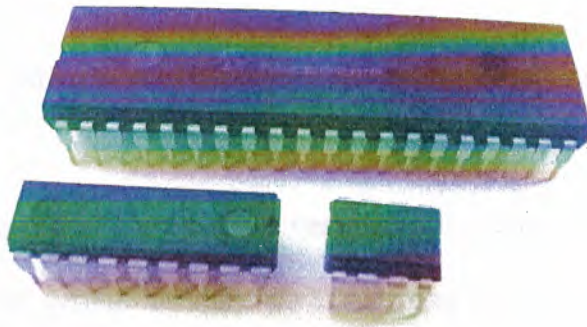


Figure 2.2.: Some microcontrollers

Basically, a microcomputer executes a user program which is loaded in its program memory. Under the control of this program data is received from external devices (inputs), manipulated and then sent to external devices (outputs). For example, in a microcontroller based oven temperature control system the temperature is read by

the microcomputer using a temperature sensor. The microcomputer then operates a heater or a fan to control and keep the temperature at the required value. Figure 2.4 shows the block diagram of our simple oven temperature control system.

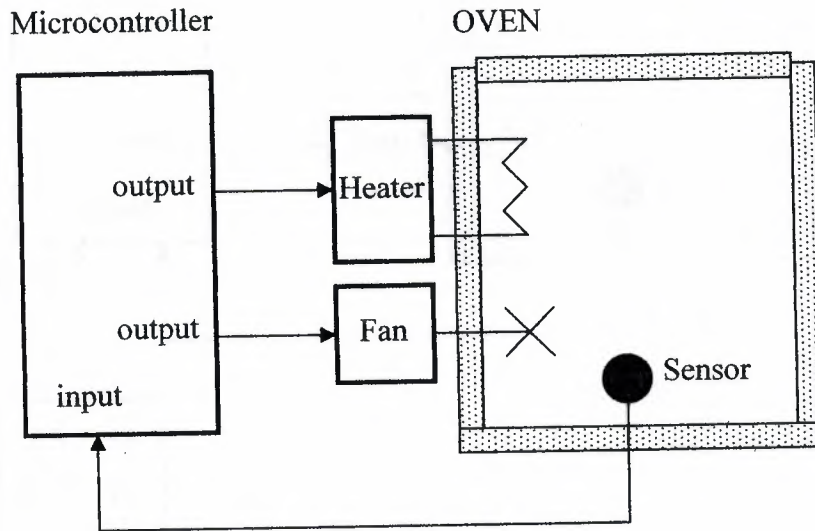


Figure 2.3.: Microcontroller based oven temperature control system

The system shown in Figure 2.3 is a very simplified temperature control system. In a more sophisticated system we may have a keypad to set the temperature, and a LCD to display the current temperature. Figure 2.4 shows the block diagram of this more sophisticated temperature control system.

For example the design even more sophisticated (see Figure 2.5) by adding an audible alarm to inform us if the temperature is outside the required values. Also, the temperature readings can be sent to a PC every second for archiving and further processing.

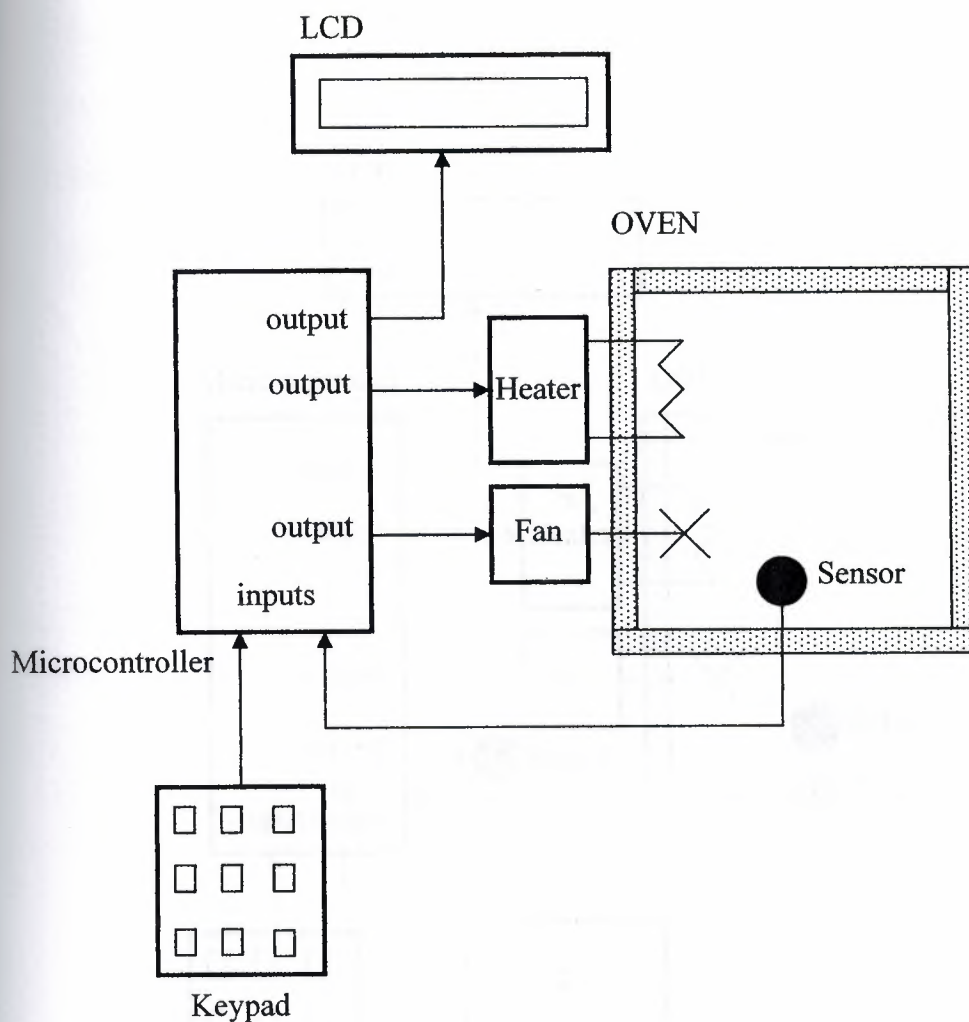


Figure 2.4.: Temperature control system with a keypad and LCD

For example, a graph of the daily temperature can be plotted on the PC. As you can see, because the microcontrollers are programmable it is very easy to make the final system as simple or as complicated as we like.

A microcontroller is a very powerful tool that allows a designer to create sophisticated input-output data manipulation under program control. Microcontrollers are classified by the number of bits they process. 8-bit microcontrollers are the most popular ones and are used in most microcontroller based applications.

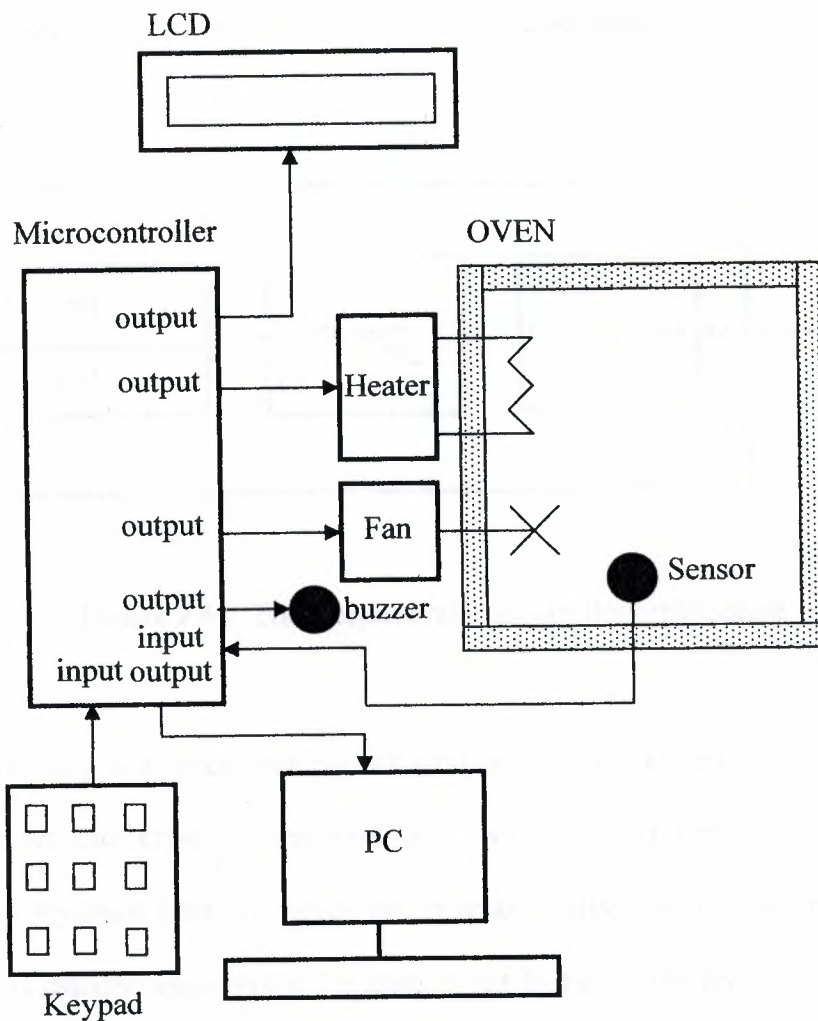


Figure 2.5.: More sophisticated temperature controller

16- and 32-bit microcontrollers are much more powerful, but usually more expensive and not required in many small to medium size general purpose applications where microcontrollers are generally used.

As shown in Figure 2.6, the simplest microcontroller architecture consists of a microprocessor, memory, and input-output. The microprocessor consists of a central processing unit (CPU), and the control unit (CU). The CPU is the brain of the

microcontroller and this is where all of the arithmetic and logic operations are performed. The control unit controls the internal operations of the microprocessor and sends out control signals to other parts of the microcontroller to carry out the required instructions.

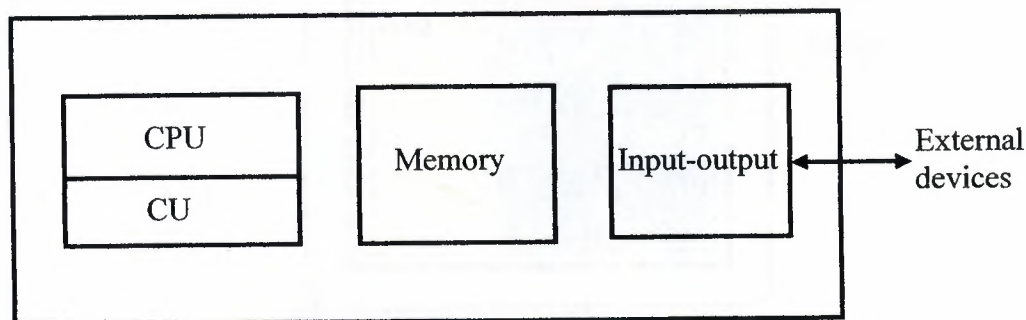


Figure 2.6: The simplest microcontroller architecture

Memory is an important part of a microcontroller system. Depending upon the type use we can classify memories into two groups: program memory, and data memory. Program memory stores the program written by the programmer and this memory is usually non-volatile. i.e. data is not lost after the removal of power. Data memory is where the temporary data used in the program are stored and this memory is usually volatile. i.e. data is lost after the removal of power. Figure 2.7. shows the main idea of a PIC.

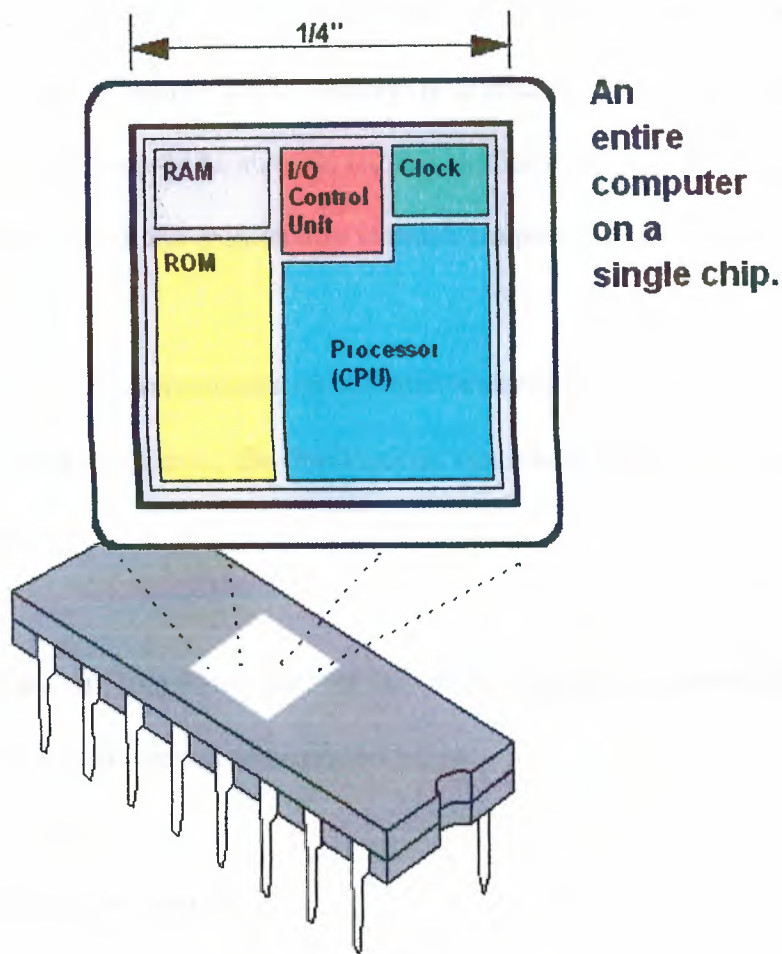


Figure 2.7.: The main idea of a microcontroller.

2.3. Why Microcontrollers are Used?

There are a lot of reasons why microcontrollers are incorporated in control systems:

- A. **Cost:** Microcontrollers with the supplementary circuit components are much cheaper than a computer with an analog and digital I/O .
- B. **Size and Weight:** Microcontrollers are compact and light compared to computers.

- C. **Simple applications:** If the application requires very few number of I/O and the code is relatively small, which do not require extended amount of memory and a simple LCD display is sufficient as a user interface, a microcontroller would be suitable for this application.
- D. **Reliability:** Since the architecture is much simpler than a computer it is less likely to fail.
- E. **Speed:** All the components on the microcontroller are located on a single piece of silicon. Hence, the applications run much faster than it does on a computer.

Memories are an important part of all microcontroller systems. There are basically five types of memories as summarised below.

2.4. Microcontroller Memories

In this section the properties of some commonly used semiconductor memories are given.

2.4.1. RAM

RAM means Random Access Memory. It is a general purpose memory which usually stores user the data used in a program. RAM is volatile. i.e. data is lost after the removal of power. Most microcontrollers have some amount of internal RAM. 256 bytes is a common amount, although some microcontrollers have more, some less. In general it is possible to extend the memory by adding external memory chips.

2.4.2. ROM

ROM is Read Only Memory. This type of memory usually holds program or fixed user data. ROM memories are programmed at factory during the manufacturing process and their contents can not be changed by the user. ROM memories are only useful if you have developed a program and wish to order several thousand copies of it.

2.4.3. EPROM

EPROM is erasable Programmable Read Only Memory. This is similar to ROM, but the EPROM can be programmed using a suitable programming device. EPROM memories have a small clear glass window on top of the chip where the data can be erased under UV light. Many development versions of microcontrollers are manufactured with EPROM memories where the user program can be stored. These memories are erased and re-programmed until the user is satisfied with the program. Some versions of EPROMs, known as OTP (One Time Programmable), can be programmed using a suitable programmer device but these memories can not be erased. OTP memories cost much less than the EPROMs. OTP is useful after a project has been developed completely and it is required to make many copies of the program memory.

2.4.4. EEPROM

EEPROM is Electrically Erasable Programmable Read Only Memory, which is a non-volatile memory. These memories can be erased and also be programmed under program control. EEPROMs are used to save configuration information, maximum and minimum values, identification data etc. Some microcontrollers have built-in EEPROM

memories (e.g. PIC16F84 contains a 64-byte EEPROM memory where each byte can be programmed and erased directly by software). EEPROM memories are usually very slow.

2.4.5. Flash EEPROM

This is another version of EEPROM type memory. This memory has become popular in microcontroller applications and is used to store the user program. Flash EEPROM is non-volatile and is usually very fast. The data is erased and then re-programmed using a programming device. The entire contents of the memory should be erased and then re-programmed.

2.5. MICROCONTROLLER FEATURES

Microcontrollers from different manufacturers have different architectures and different capabilities. Some may suit a particular application while others may be totally unsuitable. The hardware features of microcontrollers in general are described in this section.

2.5.1. Supply Voltage

Most microcontrollers operate with the standard logic voltage of +5V. Some microcontrollers can operate at as low as +2.7V and some will tolerate +6V without any problems. You should check the manufacturers' data sheets about the allowed limits of the power supply voltage.

A voltage regulator circuit is usually used to obtain the required power supply voltage when the device is to be operated from a mains adaptor or batteries. For example, a 5V regulator is required if the microcontroller is to be operated from a 5V supply using a 9V battery.

2.5.2. The Clock

All microcontrollers require a clock (or an oscillator) to operate. The clock is usually provided by connecting external timing devices to the microcontroller. Most microcontrollers will generate clock signals when a crystal and two small capacitors are connected. Some will operate with resonators or external resistor-capacitor pair. Some microcontrollers have built-in timing circuits and they do not require any external timing components. If your application is not time sensitive you should use external or internal (if available) resistor-capacitor timing components for simplicity and low cost.

An instruction is executed by fetching it from the memory and then decoding it. This usually takes several clock cycles and is known as the *instruction cycle*. In PIC microcontrollers the instruction cycle takes four clock periods. Thus, the microcontroller is actually operated at a clock rate which is a quarter of the actual oscillator frequency.

2.5.3. Timers

Timers are important parts of any microcontroller. A timer is basically a counter which is driven either from an external clock pulse or from the internal oscillator of the microcontroller. A timer can be 8-bits, or 16-bits wide. Data can be loaded into a timer under program control and the timer can be stopped or started by program control.

Most timers can be configured to generate an interrupt when they reach a certain count (usually when they overflow). The interrupt can be used by the user program to carry out accurate timing related operations inside the microcontroller.

Some microcontrollers offer capture and compare facilities where a timer value can be read when an external event occurs, or the timer value can be compared to a preset value and an interrupt can be generated when this value is reached.

It is typical to have at least one timer in every microcontroller. Some microcontrollers may have two, three, or even more timers where some of the timers can be cascaded for longer counts.

2.5.4. Watchdog

Most microcontrollers have at least one watchdog facility. The watchdog is basically a timer which is refreshed by the user program and a reset occurs if the program fails to refresh the watchdog. The watchdog timer is used to detect a system problem, such as the program being in an endless loop. A watchdog is a safety feature that prevents runaway software and stops the microcontroller from executing meaningless and unwanted code. Watchdog facilities are commonly used in real-time systems where it is required to regularly check the successful termination of one or more activities.

2.5.5. Reset Input

A reset input is used to reset a microcontroller. Resetting puts the microcontroller into a known state such that the program execution starts from address 0 of the program memory. An external reset action is usually achieved by connecting a

push-button switch to the reset input such that the microcontroller can be reset when the switch is pressed.

2.5.6. Interrupts

Interrupts are very important concepts in microcontrollers. An interrupt causes the microcontroller to respond to external and internal (e.g. a timer) events very quickly. When an interrupt occurs the microcontroller leaves its normal flow of program execution and jumps to a special part of the program, known as the *Interrupt Service Routine* (ISR)[13]. The program code inside the ISR is executed and upon return from the ISR the program resumes its normal flow of execution.

The ISR starts from a fixed address of the program memory. This address is also known as the *interrupt vector address*. For example, in a PIC16F84 microcontroller the ISR starting address is 4 in the program memory. Some microcontrollers with multi-interrupt features have just one interrupt vector address, while some others have unique interrupt vector addresses, one for each interrupt source. Interrupts can be nested such that a new interrupt can suspend the execution of another interrupt. Another important feature of a microcontroller with multi-interrupt capability is that different interrupt sources can be given different levels of priority.

2.5.7. Brown-out Detector

Brown-out detectors are also common in many microcontrollers and they reset a microcontroller if the supply voltage falls below a nominal value. Brown-out detectors are safety features and they can be employed to prevent unpredictable operation at low voltages, especially to protect the contents of EEPROM type memories.

2.5.8. Analogue-to-digital Converter

An analogue-to-digital converter (A/D) is used to convert an analogue signal such as voltage to a digital form so that it can be read by a microcontroller. Some microcontrollers have built-in A/D converters. It is also possible to connect an external A/D converter to any type of microcontroller. A/D converters are usually 8-bits, having 256 quantisation levels. Some microcontrollers have 10-bit A/D converters with 1024 quantisation levels. Most PIC microcontrollers with A/D features have multiplexed A/D converters where more than one analogue input channel is provided.

The A/D conversion process must be started by the user program and it may take several hundreds of microseconds for a conversion to complete. A/D converters usually generate interrupts when a conversion is complete so that the user program can read the converted data quickly.

A/D converters are very useful in control and monitoring applications since most sensors (e.g. temperature sensor, pressure sensor, force sensor etc.) produce analogue output voltages.

2.5.9. Serial Input-Output

Serial communication (also called RS232 communication) enables a microcontroller to be connected to another microcontroller or to a PC using a serial cable. Some microcontrollers have built-in hardware called USART (Universal Synchronous-Asynchronous Receiver-Transmitter) to implement a serial communication interface. The baud rate and the data format can usually be selected by the user program. If any serial input-output hardware is not provided, it is easy to

develop software to implement serial data communication using any I/O pin of a microcontroller.

Some microcontrollers incorporate SPI (Serial Peripheral Interface) or I²C (Integrated Inter Connect) hardware bus interfaces. These enable a microcontroller to interface to other compatible devices easily.

2.5.10. EEPROM Data Memory

EEPROM type data memory is also very common in many microcontrollers. The advantage of an EEPROM memory is that the programmer can store non-volatile data in such a memory, and can also change this data whenever required. For example, in a temperature monitoring application the maximum and the minimum temperature readings can be stored in an EEPROM memory. Then, if the power supply is removed for whatever reason, the values of the latest readings will still be available in the EEPROM memory.

Some microcontrollers have no built-in EEPROM memory, some provide only 16 bytes of EEPROM memory, while some others may have as much as 256 bytes of EEPROM memories.

2.5.11. LCD Drivers

LCD drivers enable a microcontroller to be connected to an external LCD display directly. These drivers are not common since most of the functions provided by them can be implemented in software.

2.5.12. Analogue Comparator

Analogue comparators are used where it is required to compare two analogue voltages. Although these circuits are implemented in most high-end PIC microcontrollers they are not common in other microcontrollers.

2.5.13. Real-time Clock

Real-time clock enables a microcontroller to have absolute date and time information continuously. Built-in real-time clocks are not common in most microcontrollers since they can easily be implemented by either using a dedicated real-time clock chip, or by writing a program.

2.5.14. Sleep Mode

Some microcontrollers (e.g. PIC) offer built-in sleep modes where executing this instruction puts the microcontroller into a mode where the internal oscillator is stopped and the power consumption is reduced to an extremely low level. The main reason of using the sleep mode is to conserve the battery power when the microcontroller is not doing anything useful. The microcontroller usually wakes up from the sleep mode by external reset or by a watchdog time-out.

2.5.15. Power-on Reset

Some microcontrollers (e.g. PIC) have built-in power-on reset circuits which keep the microcontroller in reset state until all the internal circuitry has been initialised. This feature is very useful as it starts the microcontroller from a known state on power-

up. An external reset can also be provided where the microcontroller can be reset when an external button is pressed.

2.5.16. Low Power Operation

Low power operation is especially important in portable applications where the microcontroller based equipment is operated from batteries. Some microcontrollers (e.g. PIC) can operate with less than 2mA with 5V supply, and around 15 μ A at 3V supply. Some other microcontrollers, especially microprocessor based systems where there could be several chips may consume several hundred mill amperes or even more.

2.5.17. Current Sink/Source Capability

This is important if the microcontroller is to be connected to an external device which may draw large current for its operation. PIC microcontrollers can source and sink 25mA of current from each output port pin. This current is usually sufficient to drive LEDs, small lamps, buzzers, small relays etc. The current capability can be increased by connecting external transistor switching circuits or relays to the output port pins.

2.6. Microcontroller Architectures

Usually two types of architectures are used in microcontrollers (see Figure 2.8): *Von Neumann* architecture and *Harvard* architecture[14]. Von Neumann architecture is used by a large percentage of microcontrollers and here all memory space is on the same bus and instruction and data use the same bus. In the Harvard architecture (used

by the PIC microcontrollers), code and data are on separate busses and this allows the code and data to be fetched simultaneously, resulting in an improved performance.

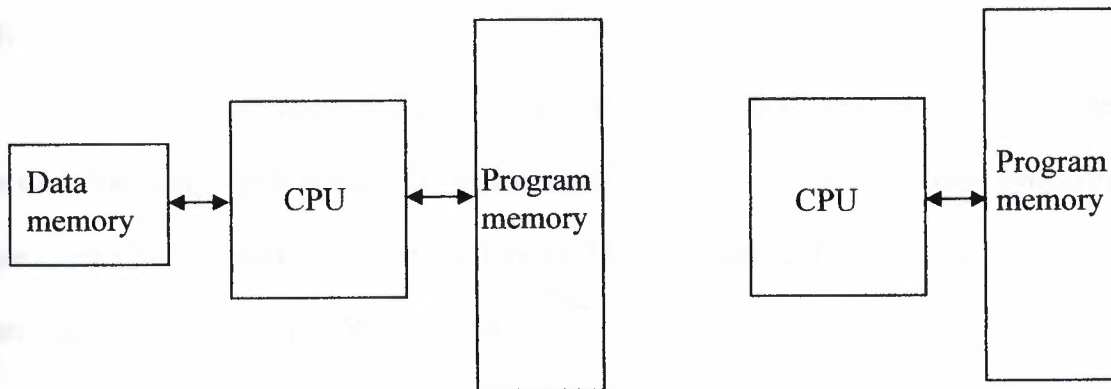


Figure 2.8.: Von Neumann and Harvard architectures

2.6.1. RISC and CISC

RISC (Reduced Instruction Set Computer) and CISC (Complex Instruction Computer) refer to the instruction set of a microcontroller. In an 8-bit RISC microcontroller, data is 8-bits wide but the instruction words are more than 8-bits wide (usually 12, 14 or 16-bits) and the instructions occupy one word in the program memory. Thus, the instructions are fetched and executed in one cycle, resulting in an improved performance. PIC microcontrollers are RISC based devices and they have no more than 35 instructions.

In a CISC microcontroller both data and instructions are 8-bits wide. CISC microcontrollers usually have over 200 instructions. Data and code are on the same bus and can not be fetched simultaneously.

2.7. Examples of Microcontrollers

There are a lot of PIC microcontrollers such as: 8 bit microcontrollers are start with PIC10, PIC12, PIC14, PIC16, PIC17, PIC18. 16 bit microcontrollers are start with PIC24.

PICs are small integrated circuits known correctly as micro controllers and can be used for many applications from traffic light control, digital thermometer, propeller clock, satellite encryption decoding and so on. They are manufactured by Microchip and can also take the form of EPROMs as well.

2.8. PIC Microcontrollers

The PIC microcontroller family of microcontrollers is manufactured by *Microchip Technology Inc.* Currently they are one of the most popular microcontrollers used in many commercial and industrial applications. Over 120 million devices are sold each year.

The PIC microcontroller architecture is based on a modified Harvard RISC (Reduced Instruction Set Computer) instruction set with dual-bus architecture, providing fast and flexible design with an easy migration path from only 6 pins to 80 pins, and from 384 bytes to 128K bytes of program memory.

PIC microcontrollers are available with many different specifications depending on:

- Memory Type
 - Flash
 - OTP (One-time-programmable)
 - ROM (Read-only-memory)

- ROMless
- I/O Pin Count
 - 4 – 18 pins
 - 20 -28 pins
 - 32 -44 pins
 - 45 and above pins
- Memory Size
 - 0.5K – 1K
 - 2K – 4K
 - 8K – 16K
 - 24K – 32K
 - 48K – 64K
 - 96K – 128K
- Special Features
 - CAN
 - USB
 - LCD
 - Motor Control
 - Radio Frequency

Although there are many models of PIC microcontrollers, the nice thing is that they are upward compatible with each other and a program developed for one model can very easily and in many cases with no modifications be run on other models of the family. The basic assembler instruction set of PIC microcontrollers consists of only 33 instructions and most of the family members (except the newly developed devices) use the same instruction set. This is why a program developed for one model can run on another model with similar architecture without any changes.

All PIC microcontrollers offer the following features:

- RISC instruction set with only a handful of instructions to learn
- Digital I/O ports
- On-chip timer with 8-bit prescaler
- Power-on reset
- Watchdog timer
- Power saving SLEEP mode
- High source and sink current
- Direct, indirect, and relative addressing modes
- External clock interface
- RAM data memory
- EPROM or Flash program memory

Some devices offer the following additional features:

- Analogue input channels
- Analogue comparators
- Additional timer circuits
- EEPROM data memory

- External and internal interrupts
- Internal oscillator
- Pulse-width modulated (PWM) output
- USART serial interface

Some even more complex devices in the family offer the following additional

features:

- CAN bus interface
- I²C bus interface
- SPI bus interface
- Direct LCD interface
- USB interface
- Motor control

Although there are several hundred models of PIC microcontrollers, choosing a microcontroller for an application is not a difficult task and requires taking into account these factors:

- Number of I/O pins required
- Required peripherals (e.g. USART, USB)
- The minimum size of program memory*
- The minimum size of RAM memory
- Whether or not EEPROM non-volatile data memory is required
- Speed
- Physical size
- Cost

The important point to remember is that there could be many models which satisfy all of the above requirements. You should always try to find the model which satisfies your minimum requirements and the one which does not offer more than you may need. For example, if you require a microcontroller with only 8 I/O pins and if there are two identical microcontrollers, one with 8 and the other one with 16 I/O pins, you should select the one with 8 I/O pins.

Although there are several hundred models of PIC microcontrollers, the family can be broken down into three main groups, which are:

- 12-bit instruction word (e.g. 12C5XX, 16C5X)
- 14-bit instruction word (e.g. 16F8X, 16F87X)
- 16-bit instruction word (e.g. 17C7XX, 18C2XX)

All three groups share the same RISC architecture and the same instruction set, with a few additional instructions available for the 14-bit, and many more instructions available for the 16-bit models. Instructions occupy only one word in memory, thus increasing the code efficiency and reducing the required program memory. Instructions and data are transferred on separate buses, thus the overall system performance is increased.

The features of some microcontrollers in each group are given in the following sections

2.8.1. 12-bit Instruction word

Some of the popular microcontrollers in this group are the PIC12C508 and PIC16C5X. Brief details of these microcontrollers are given below:

PIC12C508: This is a low-cost, 8-pin device with 512 x 12 EPROM program memory, and 25 bytes of RAM data memory. The device can operate at up to 4MHz clock input and the instruction set consists of only 33 instructions. The device features 6 I/O ports, 8-bit timer, power-on reset, watchdog timer, and internal 4MHz oscillator capability. One of the major disadvantages of this microcontroller is that the program memory is EPROM based and it can not be erased or programmed using the standard programming devices. The program memory has to be erased using an EPROM eraser device (an ultraviolet light source).

The "F" version of this family (e.g. PIC12F508) is based on flash program memory which can be erased and re-programmed using the standard PIC programmer devices. Similarly, the "CE" version of the family (e.g. PIC12CE518) offers an additional 16-byte non-volatile EEPROM data memory.

PIC16C5X: This is one of the earliest PIC microcontrollers. The device is 18-pin with a 384 x 12 EPROM program memory, 25 bytes of RAM data memory, 12 I/O ports, a timer, and a watchdog. Some other members in the family, e.g. PIC16C56 have the same architecture but more program memory (1024 x 12). PIC16C58A has more program memory (2048 x 12) and also more data memory (73 bytes of RAM).

2.8.2. 14-bit Instruction word

This is a big family including many models of PIC microcontrollers. These devices are supported by both the PicBasic and PicBasic Pro compilers. Most of the devices in this family can operate at up to 20MHz clock rate. The instruction set consists of 35 instructions. These devices offer advanced features such as internal and external interrupt sources. Some of the popular microcontrollers in this group are

PIC16C554 PIC16F84, PIC16F627, and PIC16F877. Brief details of these microcontrollers are given below:

PIC16C554: This microcontroller has similar architecture to the PIC16C54 but the instructions are 14 bits wide. The program memory is 512 x 14 and the data memory is 80 bytes of RAM. There are 13 I/O pins where each pin can source or sink 25mA current. Additionally, the device contains a timer, and a watchdog.

PIC16F84: This has been one of the most popular PIC microcontrollers for a very long time[2]. This is an 18-pin device and it offers 1024 x 14 flash program memory, 36 bytes of data RAM, 64 bytes of non-volatile EEPROM data memory, 13 I/O pins, a timer, a watchdog, and internal and external interrupt sources. The timer is 8-bits wide but can be programmed to generate internal interrupts for timing purposes. PIC16F84 can be operated from a crystal or a resonator for accurate timing. A resistor-capacitor can also be used as a timing device for applications where accurate timing is not required. PIC16F84 is used in the design in this thesis.

Pin-out description of the PIC16F84 is given in Figure 2.9

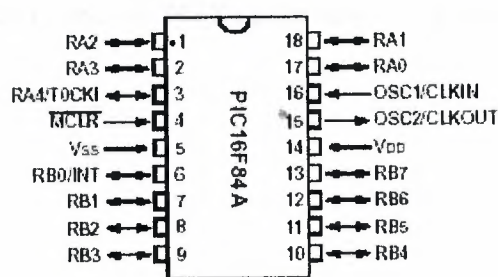


Figure 2.9: Pin-out diagram of PIC16F84

The data memory in PIC16F84 is partitioned into two areas. The first is the Special Function Registers (SFR) area, while the second is the General Purpose

Registers (GPR) area. The SFRs control the operation of the device. Portions of data memory are banked. This is for both the SFR area and the GPR area. The GPR area is banked to allow greater than 116 bytes of general purpose RAM. The banked areas of the SFR are for the registers that control the peripheral functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register. Figure 2.10. shows the memory map of the PIC16F84 microcontroller.

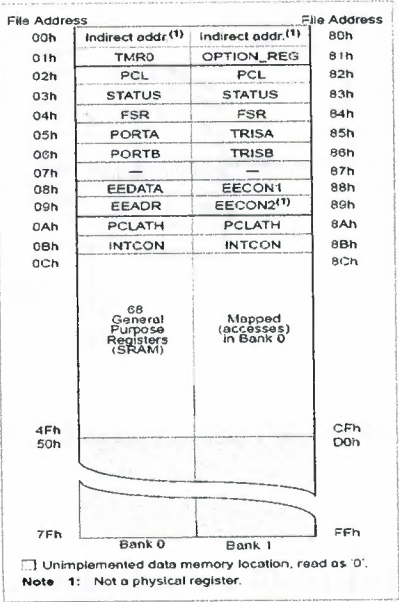


Figure 2.10.: Memory map of the PIC16F84 microcontroller

The block diagram of the PIC16F84 microcontrolelr is shown in Figuree 2.11

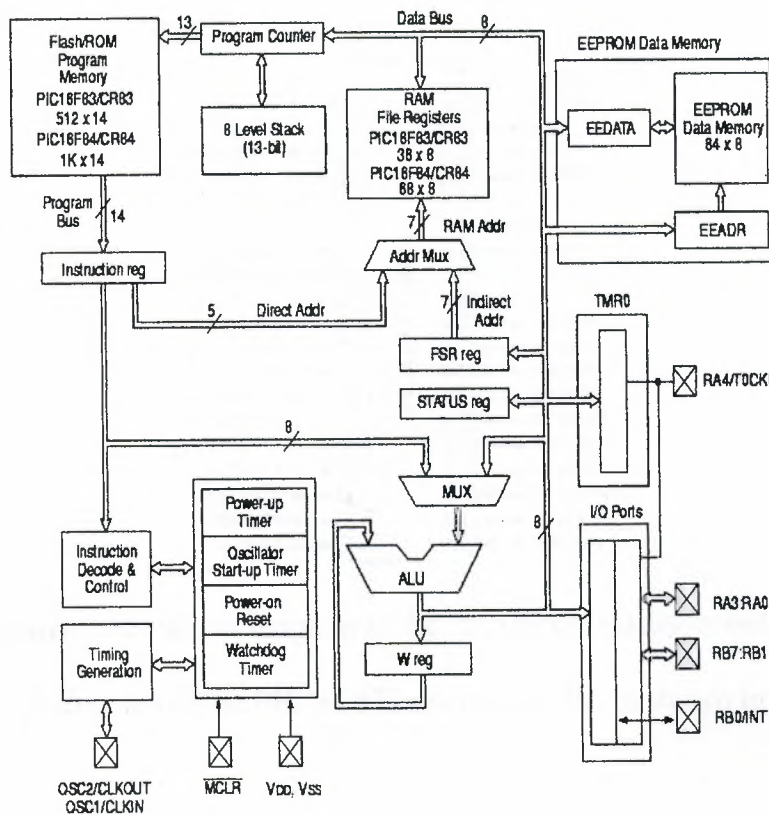


Figure 2.11 Block diagram of the PIC16F84 microcontroller

PIC16F877: This microcontroller is a 40-pin device and is one of the popular microcontrollers used in complex applications [3]. The device offers 8192 x 14 flash program memory, 368 bytes of RAM, 256 bytes of non-volatile EEPROM memory, 33 I/O pins, 8 multiplexed A/D converters with 10-bits resolution, PWM generator, 3 timers, analogue capture and comparator circuit, USART, and internal and external interrupt facilities. PIC16F877 is used in the design in this thesis.

Figure 2.12 shows the pin-out diagram of the PIC16F877 microcontroller.

PDIP

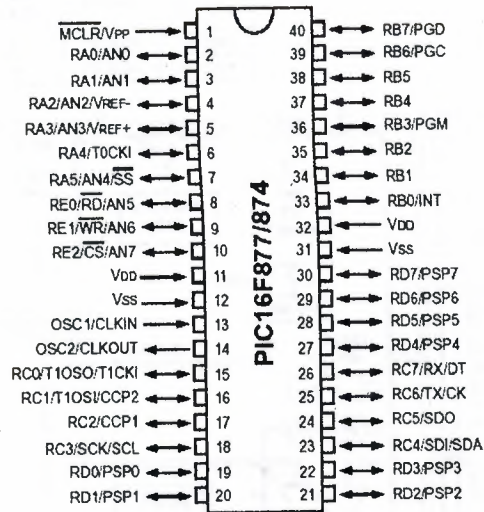


Figure 2.12 Pin-out diagram of the PIC16F877 microcontroller

The block diagram of the PIC16F877 microcontroller is shown in Figure 2.13.

Indirect addr. ⁽¹⁾		Indirect addr. ⁽¹⁾		Indirect addr. ⁽¹⁾		Indirect addr. ⁽¹⁾		File Address
TMR0	00h	OPTION_REG	80h	TMR0	100h	OPTION_REG	130h	
PCL	01h	PCL	81h	PCL	101h	PCL	131h	
STATUS	02h	STATUS	82h	STATUS	102h	STATUS	132h	
FSR	03h	FSR	83h	FSR	103h	FSR	133h	
PORTA	04h	TRISA	84h		104h		134h	
PORTB	05h	TRISB	85h	PORTB	105h	TRISE	135h	
PORTC	06h	TRISC	86h		106h		136h	
PORTD ⁽²⁾	07h	TRISD ⁽²⁾	87h		107h		137h	
PORTF ⁽²⁾	08h	TRISE ⁽²⁾	88h		108h		138h	
PCLATH	09h	PCLATH	89h	PCLATH	109h	PCLATH	139h	
INTCON	0Ah	INTCON	90h	INTCON	110h	INTCON	140h	
PIR1	0Bh	PIE1	91h	EECON1	111h	EECON1	141h	
PIR2	0Ch	PIE2	92h	EECON2	112h	EECON2	142h	
TMR1L	0Dh	PCON	93h	EEADRH	113h	Reserved ⁽²⁾	143h	
TMR1H	0Eh		94h		114h		144h	
T1CON	0Fh		95h		115h		145h	
TMR2	10h	SSPCON2	96h		116h		146h	
T2CON	11h	PR2	97h	General Purpose Register 16 Bytes	117h	General Purpose Register 16 Bytes	147h	
SSPBUF	12h	SSPADDD	98h		118h		148h	
SSPCON	13h	SSPSTAT	99h		119h		149h	
CCPR1L	14h		9Ah		120h		150h	
CCPR1H	15h		9Bh		121h		151h	
CCP1CON	16h		9Ch		122h		152h	
RCSTA	17h	TXSTA	9Dh		123h		153h	
TXREG	18h	SPBRG	9Eh		124h		154h	
RCREG	19h		9Fh		125h		155h	
CCPR2L	1Ah		90h		126h		156h	
CCPR2H	1Bh		91h		127h		157h	
CCP2CON	1Ch		92h		128h		158h	
ADRESH	1Dh	ADRESL	93h		129h		159h	
ADCON0	1Eh	ADCON1	94h		130h		160h	
	1Fh		95h		131h		161h	
	20h		96h		132h		162h	
General Purpose Register 96 Bytes	7Fh	General Purpose Register 80 Bytes	FFh	General Purpose Register 80 Bytes	17Fh	General Purpose Register 80 Bytes	1FFh	
Bank 0		Bank 1		Bank 2		Bank 3		
		accesses 70h-7Fh		accesses 70h-7Fh		accesses 70h-7Fh		

■ Unimplemented data memory locations, read as '0'.
 * Not a physical register.
 Note 1: These registers are not implemented on 28-pin devices.
 Note 2: These registers are reserved, maintain these registers clear.

Figure 2.14.: Memory map of the PIC16F877 microcontroller

PIC16F627: This is an 18-pin microcontroller with 1024 x 14 flash program memory. The device offers 224 bytes of RAM, 128 bytes of non-volatile EEPROM memory, 16 I/O pins, two 8-bit timers, one 16-bit timer, a watchdog and comparator circuits. This microcontroller is similar to PIC16F84, but offers more I/O pins, more program memory, and a lot more RAM memory. In addition, PIC16F627 is more suited to applications which require more than one timer.

2.8.3. 16-bit Instruction word

16-bit microcontrollers are at the high end of the PIC microcontroller family. Most of the devices in this group can operate at up to 40 MHz, have 33 I/O pins, and 3 timers. They have 23 instructions in addition to the 35 instructions found on the 14-bit microcontrollers.

2.9. Inside a PIC microcontroller

Although there are many models of microcontrollers in the PIC family, they all share some common features, such as program memory, data memory, input-output ports, and timers. Some devices have additional features such as A/D converters, USART and so on. Because of these common features, we can look at these attributes and cover the operation of most devices in the family.

2.9.1. Program memory (Flash)

In early microprocessors and microcontrollers the program memory was EPROM which meant that it had to be erased using UV light before it can be re-programmed. Most PIC microcontrollers nowadays are based on the flash technology where the memory chip can be erased or re-programmed using a programmer device. Most PIC microcontrollers can also be programmed without removing them from their circuits. This process (called in-circuit serial programming or ISP) speeds up the development cycle and lowers the development costs. Although the program memory is mainly used to store a program, there is no reason why it can not be used to store constant data used in programs.

PIC microcontrollers can have program memories from 0.5K to over 16K. It is interesting to note that PICs are known as 8-bit microcontrollers. This is actually true as far as the width of the data memory is concerned, which is 8-bits wide. Microchip calls the 14-bits a *word*, even though a *word* is actually 16-bits wide. When power is applied to the microcontroller or when the MCLR input is lowered to logic 0, execution start from the Reset Vector, which is the first word of the program memory. Thus, the first instruction executed after a reset is the one located at address 0 of the program memory. When the program is written in assembler language the programmer has to use special instructions (called ORG) so that the first executable instruction is loaded into address 0 of the program memory.

2.9.2. Data memory (RAM)

The data memory is used to store all of your program variables. This is a RAM memory which means that all the data is lost when power is removed. The width of the data memory is 8-bits wide and this is why the PIC microcontrollers are called 8-bit microcontrollers.

The data memory in a PIC microcontroller consists of banks where some models may have only 2 banks, some models 4 banks and so on. A required bank of the data memory can be selected under program control.

2.9.3. Register file map and special function registers

Register File Map (RFM) is a layout of all the registers available in a microcontroller and this is extremely useful when programming the device, especially when using an assembler language. The RFM is divided into two parts: the *Special*

Function Registers (SFR), and the *General Purpose Registers (GPR)*. For example, on a PIC16F84 microcontroller there are 68 GPR registers and these are used to store temporary data.

SFR is a collection of registers used by the microcontroller to control the internal operations of the device. Depending upon the complexity of the devices the number of registers in the SFR varies. It is important that the programmer understands the functions of the SFR registers fully since they are used both in assembly language and in high-level languages.

Depending on the model of PIC microcontroller used there could be other registers. Some of the important SFR registers that you may need to configure while programming using a high-level language are:

- OPTION register
- I/O registers
- Timer registers
- INTCON register
- A/D converter registers

The functions and the bit definitions of these registers are described in detail in the following sections.

2.9.4. Option register

This register is used to setup various internal features of the microcontroller and is named as OPTION_REG. This is a readable and writable register which contains

various control bits to configure the on-chip timer and the watchdog timer. This register is at address 81 (hexadecimal) of the microcontroller and its bit definitions are given in Figure 2.15. The OPTION_REG register is also used to control the external interrupt pin RB0. This pin can be setup to generate an interrupt for example when it changes from logic 0 to logic 1. The microcontroller then suspends the main program execution and jumps to the interrupt service routine to service the interrupt. Upon return from the interrupt, normal processing resumes.

7	6	5	4	3	2	1	0
RBPUP	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

Figure 2.15.: OPTION_REG bit definitions

Bit 7: PORTB Pull-up Enable

1: PORT B pull-ups disabled

0: PORT B pull-ups enabled

Bit 6: INT Interrupt Edge Detect

1: Interrupt on rising edge of INT input

0: Interrupt on falling edge of INT input

Bit 5: TMR0 Clock Source

1: T0CK1 pulse

0: Internal oscillator

Bit 4: TMR0 Source Edge Select

1: Increment on HIGH to LOW of T0CK1

0: Increment on LOW to HIGH of T0CK1

Bit 3: Prescaler Assignment

1: Prescaler Assigned to Watchdog Timer

0: Prescaler Assigned to TMR0

Bit 2-0: Prescaler Rate

000 1:2

001 1:4

010 1:8

011 1:16

100 1:32

101 1:64

110 1:128

111 1:256

2.9.5. I/O Registers

These registers are used for the input-output control. Every I/O port in the PIC microcontroller has two registers: *port data register* and *port direction control register*.

Port data register has the same name as the port it controls. For example, PIC16F84 microcontroller has two port data registers PORTA and PORTB. A PIC16F877 microcontroller has 5 port data registers PORTA, PORTB, PORTC,

PORTD, and PORTE. An 8-bit data can be sent to any port, or an 8-bit data can be read from the ports. It is also possible to read or write to individual port pins. For example, any bit of a given port can be set or cleared, or data can be read from one or more port pins at the same time.

Ports in a PIC microcontroller are bi-directional. Thus, each pin of a port can be used as an input or an output pin. Port direction control register configures the port pins as either inputs or outputs. This register is called the TRIS register and every port has a TRIS register named after its port name. For example, TRISA is the direction control register for PORTA. Similarly, TRISB is the direction control register for PORTB and so on.

2.9.6. Timer registers

Depending on the model used, some PIC microcontrollers have only one timer, and some may have up to 3 timers. In this section we shall look at the PIC16F84 microcontroller which has only one timer. The extension to several timers is similar and we shall see in the projects section how to use more than one timer.

The timer in the PIC16F84 microcontroller is an 8-bit register (called TMR0) which can be used as a timer or a counter. When used as a counter, the register increments each time a clock pulse is applied to pin T0CK1 of the microcontroller. When used as a timer, the register increments at a rate determined by the system clock frequency and a prescaler selected by register OPTION_REG. Prescaler rates vary from 1:2 to 1:256. For example, when using a 4MHz clock, the basic instruction cycle is 1 microsecond (the clock is internally divided by four). If we select a prescaler rate of 1:16, the counter will be incremented at every 16 microseconds. A timer interrupt is

generated when the timer overflows from 255 to 0. This interrupt can be enabled or disabled by our program. Thus, for example if we require generating interrupts at 200 microsecond intervals using a 4MHz clock, we can select a prescaler value of 1:4 and enable timer interrupts. The timer clock rate is then 4 microseconds. For a time-out of 200 microseconds, we have to send 50 clocks to the timer. Thus, the TMR0 register should be loaded with $256 - 50 = 206$. i.e. a count of 50 before an overflow occurs.

The watchdog timer's oscillator is independent from the CPU clock and the time-out is 18 msec. To prevent a time-out condition the watchdog must be reset periodically via software. If the watchdog timer is not reset before it times out, the microprocessor will be forced to jump to the reset address. The prescaler can be used to extend the time-out period and valid rates are 1, 2, 4, 8, 16, 32, 64 and 128. For example, when set to 128, the time out period is about 2 seconds ($18\text{msec} \times 128 = 2304\text{ msec}$). The watchdog timer can be disabled during programming of the device if it is not used.

Since the timer and watchdog are very important parts of the PIC microcontrollers more detailed information is given on their operation below.

2.9.7. TMR0 and Watchdog

TMR0 and a watchdog are found nearly in all PIC microcontrollers. Figure 2.16. shows the functional diagram of TMR0 and the watchdog circuit. The operation of the watchdog circuit is as described earlier and only the TMR0 circuit is described in this section.

The source of input for TMR0 is selected by bit T0CS of OPTION_REG and it can be either from the microcontroller oscillator f_{osc} divided by 4, or it can be an

external clock applied to the RA4/T0CK1 input. Here we will only look at using the internal oscillator. If a 4MHz crystal is used the internal oscillator frequency is $f_{osc} / 4 = 1\text{MHz}$ which corresponds to a period of $T = 1 / f = 10^{-6}$, or 1 μsec . TMR0 is then selected as the source for the prescaler by clearing PSA bit of OPTION_REG. The required prescaler value is selected by bits PS0 to PS2 as shown in Figure 2.8. Bit PSA should then be cleared to 0 to select the prescaler for the timer. All the bits are configured now and TMR0 register increments each time a pulse is applied by the internal oscillator. TMR0 register is 8-bits wide and it counts up to 255, then creates an overflow condition, and continues counting from 0. When TMR0 changes from 255 to 0 it generates a timer interrupt if timer interrupts and global interrupts are enabled (see INTCON register. Interrupt will be generated if GIE and TMR0 bits of INTCON are both set to 1). See the section on Interrupts for more information.

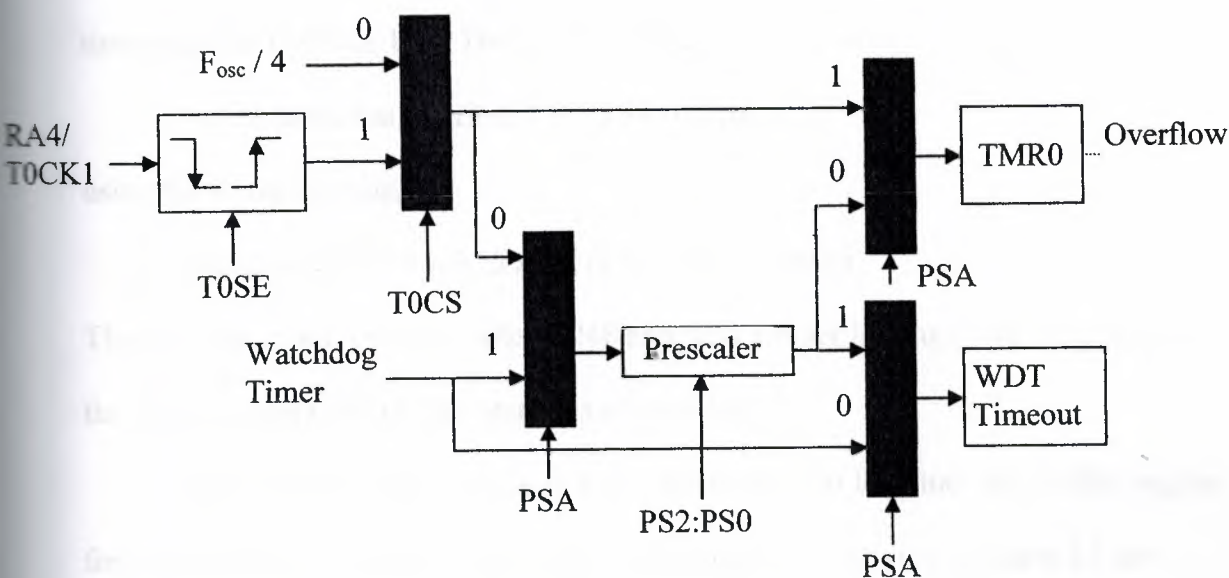


Figure 2.16.: TMR0 and watchdog circuit

By loading a value into the TMR0 register we can control the count until an overflow occurs. The formula given below can be used to calculate the time it will take for the timer to overflow (or to generate an interrupt) given the oscillator period, value loaded into the timer and the prescaler value.

$$\text{Overflow time} = 4 \times T_{\text{osc}} \times \text{Prescaler} \times (256 - \text{TMR0})$$

Where,

Overflow time is in μs

T_{osc} is the oscillator period in μs

Prescaler is the prescaler value chosen using OPTION_REG

TMR0 is the value loaded into TMR0 register

For example, assume that we are using a 4MHz crystal, and the prescaler chosen as 1:8 by setting bits PS2:PS0 to "010". Also assume that the value loaded into the timer register TMR0 is 100. The overflow time is then given by:

$$4\text{MHz clock has a period, } T = 1 / f = 0.25\mu\text{s}$$

using the above formula,

$$\text{Overflow time} = 4 \times 0.25 \times 8 \times (256 - 100) = 1248\mu\text{s}$$

Thus, the timer will overflow after 1.248msec and a timer interrupt will be generated if the timer interrupt and global interrupts are enabled.

What we normally want is to know what value to load into the TMR0 register for a required Overflow time. This can be calculated by modifying equation 2.1 as:

$$\text{TMR0} = 256 - (\text{Overflow time}) / (4 \times T_{\text{osc}} \times \text{Prescaler}) \quad (2.2)$$

For example, suppose that we want an interrupt to be generated after 500 μ s and the clock and the prescaler values are as before. The value to be loaded into the TMR0 register can be calculated using (2.1) as:

$$\text{TMR0} = 256 - 500 / (4 \times 0.25 \times 8) = 193.5 \quad (2.1)$$

The nearest number we can load into TMR0 register is 193.

2.9.8. INTCON Register

This is the interrupt control register. This register is at address 0B and 8B (hexadecimal) of the microcontroller RAM and the bit definitions are given in Figure 2.17.

7	6	5	4	3	2	1	0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

Figure 2.17.: INTCON register bit definitions

Bit 7: Global Interrupt Enable

1: Enable all un-masked interrupts

0: Disable all interrupts

Bit 6: EE Write Complete Interrupt

1: Enable EE write complete interrupt

0: Disable EE write complete interrupt

Bit 5: TMR0 Overflow Interrupt

- 1: Enable TMR0 interrupt
- 0: Disable TMR0 interrupt

Bit 4: INT External Interrupt

- 1: Enable INT External Interrupt
- 0: Disable INT External Interrupt

Bit 3: RB Port Change Interrupt

- 1: Enable RB port change interrupt
- 0: Disable RB port change interrupt

Bit 2: TMR0 Overflow Interrupt Flag

- 1: TMR0 has overflowed
- 0: TMR0 did not overflow

Bit 1: INT Interrupt Flag

- 1: INT interrupt occurred
- 0: INT interrupt did not occur

Bit 0: RB Port Change Interrupt Flag

- 1: One or more of RB4-RB7 pins changed state
- 0: None of RB4-RB7 changed state

2.9.9. A/D Converter Registers

The A/D converter is used to interface analogue signals to the microcontroller. The A/D converts analogue signals (e.g. voltage) into digital form so that they can be connected to a computer. A/D converter registers are used to control the A/D converter ports. On most PIC microcontrollers equipped with A/D, PORT A pins are used for analogue input and these port pins are shared between digital and analogue functions.

PIC16F876 includes 5 A/D converters. Similarly, PIC16F877 includes 8 A/D converters. There is actually only one A/D converter as shown in Figure 2.18 and the inputs are multiplexed and they share the same converter. The width of the A/D converter can be 8-bits or 10-bits. Both PIC16F876 and PIC16F877 have 10-bit converters. PIC16F73 has 8-bit converters. An A/D converter requires a reference voltage to operate. This reference voltage is chosen by programming the A/D converter registers and is typically +5V. Thus, if we are using a 10-bit converter (1024 quantisation levels) the resolution of our converter will be $5/1024 = 0.00488\text{V}$, or 4.88mV. i.e. we can measure analogue voltages with a resolution of 4.88mV. For example, if the measured analogue input voltage is 4.88mV we get the 10-bit digital number "0000000001", if the analogue input voltage is $2 \times 4.88 = 9.76\text{mV}$, the 10-bit converted number will be "0000000010", if the analogue input voltage is $3 \times 4.88 = 14.64\text{mV}$, the converted number will be "0000000011" and so on.

In a similar way, if the reference voltage is +5V and we are using an 8-bit converter (256 quantisation levels), the resolution of the converter will be $5/256 = 19.53\text{mV}$. For example, if the measured input voltage is 19.53mV we get the 8-bit number "00000001", if the analogue input voltage is $2 \times 19.53 = 39.06\text{mV}$ we get the 8-bit number "00000010" and so on.

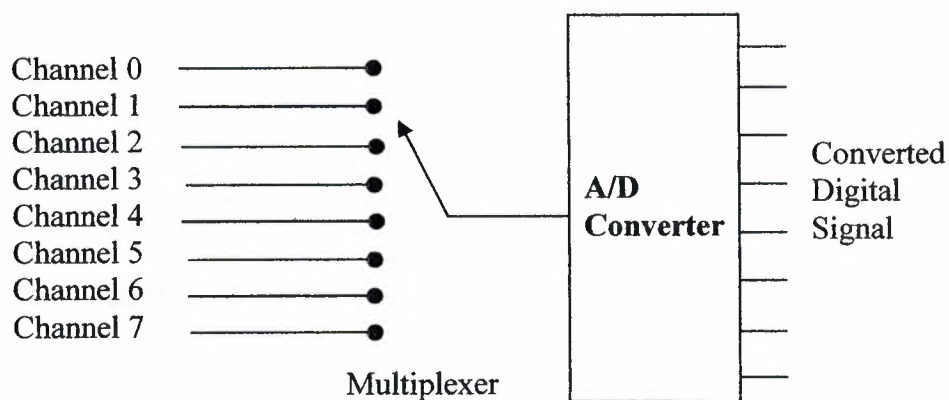


Figure 2.18.: Multiplexed A/D structure

The A/D converter is controlled by registers ADCON0 and ADCON1. The bit pattern of ADCON0 is shown in Figure 2.19. ADCON0 is split into four parts, the first part consists of the highest two bits ADCS1 and ADCS0 and they are used to select the conversion clock. The internal RC oscillator or the external clock can be selected as the conversion clock as in the following table:

00	External clock / 2
01	External clock / 8
10	External clock / 32
11	Internal RC clock

7	6	5	4	3	2	1	0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	-	ADON

Figure 2.19.: ADCON0 bit definitions

Bit 7-6: A/D Converter Clock Select

00	$f_{osc} / 2$
01	$f_{osc} / 8$
10	$f_{osc} / 32$
11	Internal RC oscillator

Bit 5-3: A/D Channel Select

000	Channel 0
001	Channel 1
010	Channel 2
011	Channel 3
100	Channel 4
101	Channel 5
110	Channel 6
111	Channel 7

Bit 2: GO/DONE bit

1: Start conversion

0: A/D conversion is complete

Bit 1: Not used

Bit 0: ADON bit

1: Turn ON A/D circuit

0: Turn OFF A/D circuit

The second part of ADCON0 consists of the three bits CHS2, CHS1 and CHS0.

These are the channel select bits, and set which input pin is routed to the A/D converter.

The selection is as follows:

CHS2:CHS1:CHS0

000 Channel 0

001 Channel 1

010 Channel 2

011 Channel 3

100 Channel 4

101 Channel 5

110 Channel 6

111 Channel 7

The third part of ADCON0 is the single GO/DONE bit. This bit has two functions: firstly by setting the bit it starts the A/D conversion. Secondly, the bit is cleared when the conversion is complete and this bit can be checked to see whether or not the conversion is complete.

The fourth part of ADCON0 is also a single bit ADON which is set to turn on the A/D converter circuitry.

ADRESH and ADRESL are the A/D converter result registers. ADRESL is the low byte and ADRESH is the upper 2 bits (if a 10-bit converter is used). We shall see how to configure the result of the conversion later.

ADCON1 is the second A/D control register. This register controls the format of converted data and mode of the PORT A inputs. The bit format of this register is shown in Figure 2.20. Bit 0 is called ADFM and when this bit 0 the result of the A/D

conversion is left justified, when it is 1, the result of the A/D conversion is right justified. If we have an 8-bit converter we can clear ADFM and just read ADRESH to get the 8-bit converted data. If we have a 10-bit converter we can set ADFM to 1 and the 8 bits of the result will be in ADRESL, 2 bits of the result will be in the lower bit positions of ADRESH. The remaining 6 positions of ADRESH (bit 2 to bit 7) will be cleared to zero.

7	6	5	4	3	2	1	0
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0

Bit 7: A/D Converter Result Format Select

1: A/D converter output is right justified

0: A/D converter output is left justified

Bit 6: Not used

Bit 5: Not used

Bit 4: Not used

Bit 3-0: Port Assignment and Reference Voltage Selection

(see Table below)

PCFG3- PCFG0	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	Vref+	Vref-
0000	A	A	A	A	A	A	A	A	Vdd	Vss
0001	A	A	A	A	Vref+	A	A	A	RA3	Vss
0010	D	D	D	A	A	A	A	A	Vdd	Vss
0011	D	D	D	A	Vref+	A	A	A	RA3	Vss
0100	D	D	D	D	A	D	A	A	Vdd	Vss
0101	D	D	D	D	Vref+	D	A	A	RA3	Vss
0110	D	D	D	D	D	D	D	D	Vdd	Vss
0111	D	D	D	D	D	D	D	D	Vdd	Vss
1000	A	A	A	A	Vref+	Vref-	A	A	RA3	RA2
1001	D	D	A	A	A	A	A	A	Vdd	Vss
1010	D	D	A	A	Vref+	A	A	A	RA3	Vss
1011	D	D	A	A	Vref+	Vref-	A	A	RA3	RA2
1100	D	D	D	A	Vref+	Vref-	A	A	RA3	RA2
1101	D	D	D	D	Vref+	Vref-	A	A	RA3	RA2
1110	D	D	D	D	D	D	D	A	Vdd	Vss
1111	D	D	D	D	Vref+	Vref-	D	A	RA3	RA2

Figure 2.20.: ADCON1 bit definitions

Bits PCFG0-3 control the mode of PORT A pins. As seen in Figure 2.20, a PORT A pin can be programmed to be a digital pin or an analogue pin. For example, if

we set PCFG0-3 to "0110" then all PORT A pins will be digital I/O pins. PCFG0-3 bits can also be used to define the reference voltage for the A/D converter. As we shall see in the projects section of the book, the reference voltage V_{ref+} is usually set to be equal to the supply voltage (V_{dd}), and V_{ref-} is set to be equal to V_{ss} . This makes the A/D reference voltage to be +5V.

2.9.10. Oscillator circuits

An Oscillator circuit is used to provide a microcontroller with a clock. A clock is needed so that the microcontroller can execute a program. PIC microcontrollers have built-in oscillator circuits and this oscillator can be operated in one of five modes:

- LP – Low power crystal
- XT – Crystal/resonator
- HS – High speed crystal/resonator
- RC resistor – capacitor
- No external components (only on some PIC microcontrollers)

2.9.11. Crystal operation

As shown in Figure 2.21, in this mode of operation an external crystal and two capacitors are connected to the OSC1 and OSC2 inputs of the microcontroller. The capacitors should be chosen to be between 22-33 Pico farads.

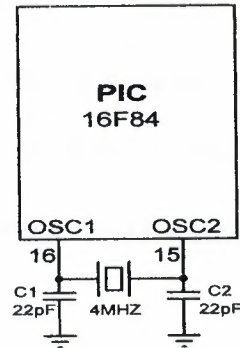


Figure 2.21.: Crystal oscillator circuit

2.9.12. Resonator operation

Resonators are available from 4MHz to about 8 MHz. They are not as accurate as crystal based oscillators. Resonators are usually 3 pin devices and the two pins at either sides are connected to OSC1 and OSC2 inputs of the microcontroller. The middle pin is connected to the ground. Figure 2.22 shows how a resonator can be used in a PIC microcontroller circuit.

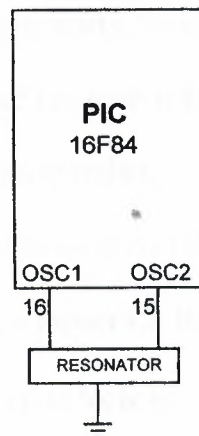


Figure 2.22.: Resonator oscillator circuit

2.9.13. RC oscillator

For applications where the timing accuracy is not important we can connect an external resistor and a capacitor to the OSC1 input of the microcontroller as in Figure 2.23. The oscillator frequency depends upon the values of the resistor and capacitor.

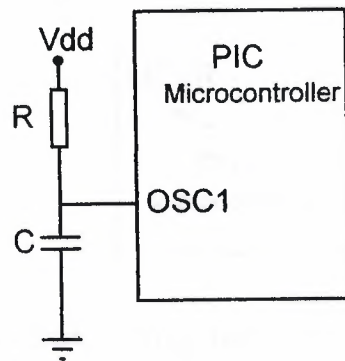


Figure 2.23.: RC oscillator circuit

2.9.14. Reset Circuit

Reset is used to put the microcontroller into a known state. Normally when a PIC microcontroller is reset execution starts from address 0 of the program memory. This is where the first executable user program resides. The reset action also initialises various SFR registers inside the microcontroller.

PIC microcontrollers can be reset when one of the following conditions occur:

- Reset during power on (POR – Power On Reset)
- Reset by lowering MCLR input to logic 0
- Reset when the watchdog overflows

As shown in Figure 2.24, a PIC microcontroller is normally reset when power is applied to the chip and when the MCLR input is tied to the supply voltage through a 4.7K resistor.

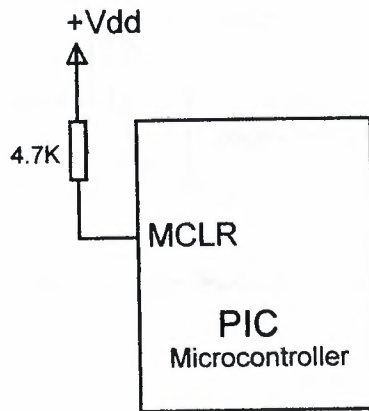


Figure 2.24.: Using the power on reset

There are many applications where we want to reset the microcontroller, e.g. by pressing an external button. The simplest circuit to achieve an external reset is shown in Figure 2.25. In this circuit, the MCLR input is normally at logic 1 and the microcontroller is operating normally. When the reset button is pressed this pin goes to logic 0 and the microcontroller is reset. When the reset button is released the microcontroller starts executing from address 0 of the program memory.

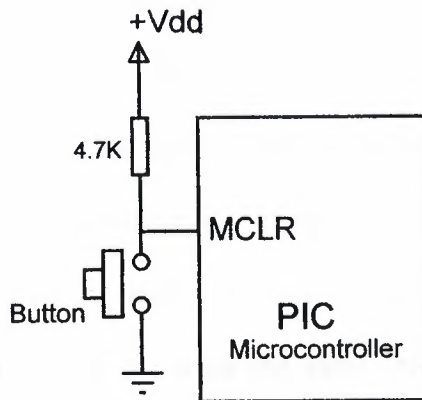


Figure 2.25.: Using an external reset button

2.9.15. Interrupts

Interrupts are an important feature of all microcontrollers. An interrupt can either occur asynchronously or synchronously. Asynchronous interrupts are usually external events which interrupt the microcontroller and request service. For example, pin INT (RB0) of a PIC16F84 microcontroller is the external interrupt pin and this pin can be used to interrupt the microcontroller asynchronously. i.e. the interrupt can occur at any time independent of the program being executed inside the microcontroller. Synchronous interrupts are usually timer interrupts, such as the timer overflow generating an interrupt.

Depending on the model used, different PIC microcontrollers may have different number of interrupt sources. For example, PIC16F84 microcontroller has the following four sources of interrupt:

- External interrupt from INT (RB0) pin
- TMR0 interrupt caused by timer overflow
- External interrupt when the state of RB4, RB5, RB6 or RB7 pins change

- Termination of writing data to the EEPROM

Interrupts are enabled and disabled by the INTCON register. Each interrupt source has two bits to control it. One enables interrupts, the other one detects when an interrupt occurs. There is a common bit called GIE (see INTCON register bit definitions) which can be used to disable all sources of interrupts. The completion status has to be cleared to zero if we want the same interrupt source to be able to interrupt again.

Assuming that we wish to use the external interrupt (INT) input, and interrupts should be accepted on the low to high transition of the INT pin, the steps before and after an interrupt are summarised below:

- Set the direction of the external interrupt to be on rising edge by setting $INTEDG = 1$ in register OPTION_REG
- Enable INT interrupts by setting $INTE = 1$ in register INTCON
- Enable global interrupts by setting $GIE = 1$ in register INTCON
- Carry out normal processing. When interrupt occurs program will jump to the ISR
- Carry out the required tasks in the ISR routine
- At the end of the ISR, re-enable the INT interrupts by clearing $INTF = 0$

2.9.16. The configuration word

PIC microcontrollers have a special register called the *Configuration Word*. This is a 14-bit register and is mapped in program memory 2007 (hexadecimal). This address is beyond the user program memory space and can not be directly accessed in a program. This register can be accessed during the programming of the microcontroller.

The configuration word stores the following information about a PIC microcontroller:

- Code protection bits. These bits are used to protect blocks of memory so that they can not be read.
- Power-on timer enable bit.
- Watchdog (WDT) timer enable bit.
- Oscillator selection bits. The oscillator can be selected as XT, HS, LP, RC, or internal (if supported by the microcontroller).

For example, in a typical application we can have the following configuration word selection during the programming of the microcontroller:

- Code protection OFF
- XT oscillator selection
- WDT disabled
- Power-up timer enables

2.10. Hardware Programmers and Software Programs Used

The microcontroller used in this thesis is the PIC series of microcontrollers manufactured by Microchip Inc. In order to develop a microcontroller based system, one needs an assembler (or a compiler), and a chip programmer. There are many assemblers for the PIC microcontrollers, but perhaps MPLAB is one of the best one available for the developers. This assembler has been developed by the Microchip company. MPLAB also provides a simulator which helps the programmer to simulate his design before implementing it in hardware.

A chip programmer is a hardware device which enables the system developer to download his code to the memory of a microcontroller. The chip programmer is

normally connected to a PC using a serial line, a parallel line, or the USB port of the PC. In a typical application the chip programmer hardware is connected to the PC, the user then inserts the microcontroller to be programmed into the socket on the chip programmer. A program is then run on the PC which transfers the program code to the memory of the microcontroller.

There are many types of chip programmers in the market. The one used in this thesis was the DELAB D128, which is a USB based programmer, and it can be used to program most types of PIC microcontrollers. Figure 2.26 shows a picture of the DELAB D128 chip programmer.



Figure 2.26.: USB PIC chip programmer

2.11. The MPLAB Assembler

The MPLAB assembler is developed by the Microchip Inc. and it is used in this thesis to develop the various programs. The current version of the compiler is MPLAB[11] version 03.20, and it can be downloaded free of charge from the web-site: <http://www.microchip.com/1000/pline/tools/picmicro/devenv/mplabi/>.

There are basically two methods of writing assembly programs:

- 1- Use a text editor and convert this txt file to an ".ASM" file.
- 2- Use the built-in editor of the MPLAB assembler.

In this thesis the first option was selected as it was easier to develop the program outside the assembler environment. After the program was developed, it was compiled using the MPLAB assembler. The assembler generates a number of output files. The list file (extension .LST) contains the assembled program with the memory allocations and the error messages. This file is useful when the program contains errors as it enables the programmer to locate and correct these errors. The hexadecimal output file (extension .HEX) is used to download the program code to the memory of the target microcontroller using a chip programmer.

2.12. The Commands of MPASM Assembler

In this section the basic commands of the MPLAB assembler used during the development of the programs in this thesis are given.

LIST: list [<list_option>, ..., <list_option>]

p=<type> Set processor type; for example, PIC16F84 or PIC16F877.

INCLUDE: #include "<include_file>"

The specified file is read in as source code. For example a include file is
pic16f84.inc or pic16f877.inc

CONFIG: `__config <expr>`

Sets the processor's configuration bits to the value described by `<expr>`.

_RC_OSC: RC Resistor/Capacitor type Oscillator.

_XT_OSC: XT Crystal/Capacitor type oscillator. This oscillator type is used in this
thesis.

_WDT: The Watchdog Timer is a free running On-Chip RC Oscillator Which does not
require any external components.

_PWRTE: Power up timer.

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

EQU: `<label> equ <expr>`

The value of `<expr>` is assigned to `<label>`. For example `four equ four`.

ORG: `[<label>] org <expr>`

Set the program origin for subsequent code at the address defined in `<expr>`.

CLRF: Clear f. The contents of register 'f' are cleared and the Z bit is set. Z bit is a
Status affected.

PORTB: 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).

PORTA: PORTA is a 5-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).

MOVLW: [label] MOVLW k

Move Literal to W. The eight-bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.

MOVWF: [label] MOVWF f

Move W to f. Move data from W register to register 'f'.

BSF: [label] BSF f,b

Bit Set f. Bit 'b' in register 'f' is set.

BCF: [label] BCF f,b

Bit Clear f. Bit 'b' in register 'f' is cleared.

RPO: Data memory is partitioned into two banks which contain the general purpose registers and the special function registers. Bank 0 is selected by clearing the RP0 bit (STATUS<5>). Setting the RP0 bit selects Bank1. Each Bank extends up to 7Fh (128 bytes).

CALL: [label] CALL k

Call Subroutine. First, return address (PC+1) is pushed onto the stack. The eleven-bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH. CALL is a two-cycle instruction.

DECFSZ: [label] DECFSZ f,d

Decrement f, Skip if 0. The contents of register 'f' are decremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'. If the result is 1, the next instruction is executed. If the result is 0, then a NOP is executed instead, making it a 2TCY instruction.

GOTO: [label] GOTO k

GOTO is an unconditional branch. The eleven-bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a two cycle instruction.

ADDLW: [label] ADDLW k

The contents of the W register are added to the eight bit literal 'k' and the result is placed in the W register

ANDLW: [label] ANDLW k

The contents of W register are AND'ed with the eight bit literal 'k'. The result is placed in the W registers.

BTFSC: [label] BTFSC f,b

If bit 'b' in register 'f' is 1 then the next instruction is executed. If bit 'b' in register 'f', is '0' then the next instruction is discarded, and a NOP is executed instead, making the instruction.

BTFSS: [label] BTFSS f,b

If bit 'b' in register 'f' is 0 then the next instruction is discarded and NOP is executed instead, making the instruction.

INCF: [label] INCF f,d

The contents of register 'f' are incremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

INCFSZ: [label] INCFSZ f,d

The contents of register 'f' are incremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'. If the result is 1, the next instruction is executed.

MOVF: [label] MOVF f,d

The contents of register f is moved to a destination dependant upon the status of d. If d=0, destination is W register. If d=1, the destination is file register f itself. d=1 is useful to test a file register since status flag Z is affected.

NOP: [label] NOP

No operation.

RETLW: [label] RETLW k

The W register is loaded with the eight bit literal 'k'. The program counter is loaded from the top of the stack (the return address). This is a two cycle instruction.

RETURN: [label] RETURN

Return from subroutine.

RLF: [label] RLF f,d

The contents of register 'f' are rotated one bit to the left through the carry flag. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is stored back in register 'f'. If 'd' is 1 the result is placed back in register 'f'.

RRF: [label] RRF f,d

The contents of register 'f' are rotated one bit to the right through the carry flag. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

CHAPTER 3

3. AQUARIUMS

3.1 Types of Aquariums

The word aquarium is taken from the Latin *aqua*, meaning *water*, and *rium*, meaning *place* or *building*[15]. Keeping fish in artificial environments for display purposes dates back to Sumerians. But the development of glass aquariums and the keeping of aquarium as a hobby dates back to the 18th century. Aquarium became popular as houses became universally electrified after World War 1. With electricity great improvements were made in aquarium technology, allowing artificial lighting as well as the filtration, and heating of the water temperature. There are currently estimated 60 million aquarium hobbyists worldwide. The hobby is widely accepted in Europe, Asia, and North America. In the United States, a large minority (40%) of aquarists maintain two or more tanks at any one time.

The types of aquariums are divided into two parts. Freshwater and saltwater are the types of aquariums. There are too many types of aquariums to list them all for the shapes. It is mostly based on personal taste and budget. Standard aquariums are in the rectangular shape and come in sizes from 2 gallons to custom sized that are over 1,000 gallons. Other shapes are pentagonal, hexagonal, octagonal, flat-back, euro, corner, high, long, desktop, and table to name a few.

There are two types of aquariums (see Figure 3.1): freshwater aquariums, and saltwater aquariums. The common freshwater aquarium maintained by a home aquarist typically includes a filtration system, an artificial lighting system, air pumps, and a heater. In addition, some freshwater tanks use power heads to increase water circulation.



Figure 3.1. Some types of aquariums

Combined biological and mechanical filtration systems are now common; these are designed to remove potentially dangerous build up of nitrogenous wastes and phosphates dissolved in the water, as well as particulate matter. Filtration systems are the most complexly engineered component of most home aquaria, and various designs are used. Most systems use pumps to remove a small portion of the tank's water to an external pathway where filtration occurs; the filtered water is then returned to the aquarium. Protein skimmers, filtration devices that remove proteins and other waste from the water, only work in salt water aquaria. ^a

Air pumps are employed to adequately oxygenate the water. These devices, once universal, are now somewhat less commonly used as some newer filtration systems create enough surface agitation to supply adequate gas exchange at the surface. Aquarium heaters are designed to act as thermostats to regulate water temperature at a level designated by the aquarist when prevailing temperature of air surrounding the aquarium is below the desired water temperature. Coolers are also available for use in

cold water aquaria or in parts of the world where the ambient room temperature is above the desired tank temperature.

An aquarium's physical characteristics form another aspect of aquarium design. Size, lighting conditions, density of floating and rooted plants, placement of bogwood, creation of caves or overhangs, type of substrate, and other factors can all affect the behavior and survivability of tank inhabitants.

The solute content of water is perhaps the most important aspect of water conditions, as total dissolved solids and other constituents can dramatically impact basic water chemistry, and therefore how organisms are able to interact with their environment. Salt content, or salinity, is the most basic classification of water conditions. An aquarium may have fresh water, simulating a lake or river environment; salt water, simulating an ocean or sea environment; or brackish water, simulating environments lying between fresh and salt, such as estuaries.

Home aquarists typically use modified tap water supplied through their local water supply network to fill their tanks. For freshwater aquaria, additives formulated to remove chlorine or chloramines are often all that is needed to make the water ready for aquarium use. Brackish or saltwater aquaria require the addition of a mixture of salts and other minerals, which are commercially available for this purpose. More sophisticated aquarists may make other modifications to their base water source to modify the water's alkalinity, hardness, or dissolved content of organics and gases, before adding it to their aquaria. In contrast, public aquaria with large water needs often locate themselves near a natural water source such as a river, lake, or ocean in order to have easy access to a large volume of water that does not require further treatment.

Secondary water characteristics are also important to the success of an aquarium.

The temperature of the water forms the basis of one of the two most basic aquarium classifications; tropical vs cold water. Most fish and plant species tolerate only a limited range of water temperatures: Tropical or warm water aquaria, with an average temperature of about 25°C, are much common, and tropical fish are among the most popular aquarium denizens. Cold water aquaria are those with temperatures below what would be considered tropical; a variety of fish are better suited to this cooler environment.

3.2. Aquarium Material

Basically two types of materials used in aquarium construction: glass and acrylic. The advantages and disadvantages of both types of aquariums are given below.

Scratching: Glass is very difficult to scratch. Acrylic is highly scratchable. An acrylic aquarium can be scratched by a person brushing it with their clothing, jewelry, purse, or bag.

Weight: Glass is denser and therefore heavier than acrylic. A glass tank is in general 4-10 times as much as an acrylic tank of the same volume.

Breaking and cracking: Glass can be broken or shattered with a sharp impact. Acrylic is much more difficult to break or shatter.

Shape: Glass is relatively rigid and brittle and because of this, it is difficult to make fish tanks from glass that is not rectangular or spherical in shape. Acrylic on the other hand is easier to bend and can easily be molded and formed into almost any shape.

Support and rigidity: Glass can support considerably more than its own weight. Because of this, glass aquariums are kept on stands with an open or incomplete top with little or no risk. Acrylic tanks require a stand that will support the entire bottom of the tank, or else the bottom of the tank may break from the weight of the water.

Refraction of light: Glass has different index of refraction than water. This means that as light passes through the air, then the glass, then the water to bounce off a fish and get reflected back through the water, the light is bent four times. Each time the light is bent, the image is distorted. Acrylic has nearly same index of refraction as water. This means that when we see a fish in an acrylic tank, the light has only been bent once or twice. Because of this, the only distortion we are likely to see is that the fish is slightly misplaced, but the size and colour are true.

Clarity: Glass maintains its clarity over time. Many types of acrylic will yellow with age, particularly if they are kept under a full spectrum light or exposed to direct sunlight.

Cost: Glass tanks are in general cheaper than acrylic ones.

3.3. Size

An aquarium can range from a small glass bowl containing less than a liter of water to massive tanks built in public aquaria which are limited only by engineering constraints and can house entire ecosystems. In general, larger aquarium systems are typically recommended to hobbyists due to their resistance to rapid fluctuations of temperature, allowing greater system stability.

Aquaria kept in homes by hobbyists can be as small as 11 liters. This size is widely considered the smallest practical system with filtration and other basic systems. Public aquaria designed for exhibition of large species or environments can be dramatically larger than any home aquaria. Some public aquaria can be as large as 60 feet long by 20 feet high (e.g. the Monterey Bay Aquarium).

Generally, the bigger the tank the better it is because a larger aquarium will tend to have much more stable water parameters. For example, take a 5 gallon versus a 55 gallon tank. In the 5 gallon tank the temperature may fluctuate up to 10 degrees Fahrenheit every day whereas the temperature isn't going to fluctuate as much in the 55 gallon. Having more water will usually buy us more time to correct anything that should happen.

3.4. Cycling

Newly built aquaria do not usually have the required populations of bacteria for the handling of nitrogen waste. In a process called cycling, aquarists cultivate these bacteria as fish and other producers of nitrogen waste are gradually added to the tank over the course of several weeks. Aquarists use several different methods to jump start this process, including the use of water additives containing small populations of the bacteria, or seeding a new tank with a mature bacteria colony removed from another aquarium.

Other cycling methods that have gained popularity in recent years are the fishless cycle and the silent cycle. As the name of the former implies, no fish are kept in a tank undergoing a fishless cycle. Instead, small amounts of ammonia are added to the tank to feed the bacteria being cultured. During this process, ammonia nitrite, and

nitrate levels are tested to monitor progress. The silent cycle is basically nothing more than densely stocking the aquarium with fast growing aquatic plants and relying on them to consume the nitrogen products rather than bacteria.

Improperly cycled aquaria can quickly accumulate toxic concentrations of nitrogen waste and kill their inhabitants.

3.5. Aquarium Furnishings

Several substances are used as aquarium furnishing material. Some commonly used material are:

Gravel: The gravel in an aquarium serves both an aesthetic and a practical purpose. The composition of gravel is important and if plants are kept in the tank, the gravel should be 2-5mm in diameter. The gravel can be added to the tank and arranged levelly or terraced.

Rocks: The tank should be furnished with rocks and the rocks used should not dissolve or crumble in water, nor release calcium.

Wood: Wood provides a refuge, a spawning site, and can be used to adjust the acidity of the water.

Lighting: The type of lighting is not especially important if plants are not grown. Plants require light in order to carry out photosynthesis and grow. Several different types of lights can be used. E.g. fluorescent tubes or mercury lamps. Fluorescent tubes are the most commonly used lights. They consume little power, produce little heat, and are

bright. Mercury vapor lamps are not very common and they are usually used in tanks with a depth greater than 50cm.

3.6. Heater

Correct temperature of an aquarium is very important for the healthy living of the fish. The most popular means to heat the aquarium is a glass immersion heater. Most heaters include a thermostat, so that once the temperature is set it remains at the desired level. Heaters using electronic thermostats and timers are now very commonly used. A thermometer should be used to check the water temperature at regular intervals to make sure that the water temperature is normal.

3.7. Air Pump

The air pump is an important part of the aquarium as they produce surface disturbance for oxygenation. The major drawback to air pumps is the noise they produce.

3.8. Filtration

The filter is one of the most important pieces of equipment in the aquarium. The filter used in a tank should be large enough to handle the amounts of wastes produced by the tank. Many filters are rated in terms of litres per hour. There are basically 3 types of filtration methods; mechanical filtration, chemical filtration, and biological filtration.

Mechanical filtration: This refers to the filtering of water through a mechanical strainer to remove particles from the water.

Chemical filtration: This is the filtration method where chemicals are used for the filtration. One popular material for chemical filtration is ammonia.

Biological filtration: In this type of filtration break down from organic wastes are used for the filtration of the water.

3.9. Bucket and Siphon

A bucket and a siphon are needed for water changes and adding fresh water to the tank. The bucket should be used only for the aquarium.

3.10. Setting up Equipment

The heater should be installed but should not be plugged in until the water is about the right temperature. Hook up the filter and any other equipment you have, then top off the aquarium water to just under the hood lip. Place your hood and tank light on the aquarium and then check your power cords to be sure that they are free of water. The recommend think is using a drip loop on all of the power cords to be extra cautious. Plug all of the equipment into a power strip and then "turn on" the aquarium.

The time is now to add some fish to the aquarium. But, in order to do this right, we have to wait until the aquarium has cycled before adding any fish. Only add one or two fish at a time. Adding a couple fish at a time gives the filtration system the time needed to take on the increased biological load that the new fish introduce. When you

bring the fish home let the bag float in the tank for about 15 minutes so that the fish can become acclimated to the temperature and pH of the aquarium water. After 5 minutes of floating the bag you should add some of the aquarium water to the bag so that the fish can become acclimated to the pH level in the aquarium. This will help reduce the amount of stress imposed on the fish. Stressed fish often leads to dead or diseased fish. Don't feed your fish on the first day. They probably wouldn't eat any food on the first day anyway.

3.10.1 Location of the Aquarium

The aquarium should be placed in an area where the light and temperature of the tank won't be affected by external sources such as windows and heater vents. Sunlight that enters the room through an unshaped window could affect the temperature of your tank. This could also lead to green algae problems for your tank down the road. You will want to place your aquarium on a stand that will be able to hold its total weight. You also want to be sure that the floor is able to support the total weight of the aquarium and stand.

A fish tank is just like having a dog or a cat when it comes to the amount of effort on your part. In order to have a successful fish tank you will have to work at it. Once a week, or at most once every two weeks, you will need to perform some kind of maintenance on the tank. Most of the time you will be performing water changes. You will also have to feed your fish at least once a day.

CHAPTER 4

4. DEVELOPMENT OF A MICROCONTROLLER BASED AUTOMATIC AQUARIUM SYSTEM

4.1 Overview

Aquariums are very good adornments but they require maintenance and daily care. Maintenance involves mainly cleaning the inside and outside of the aquarium, changing the water, and making sure that the aquarium equipment are in good working order. Maintenance is usually carried out once a week in small aquariums, and once every 2-3 weeks in larger aquariums.

Aquariums care consists of 4 types of work:

- Controlling the aquarium temperature
- Feeding the fish
- Providing light to the aquarium
- Cleaning and changing the water of the aquarium

4.1.1 The Water Temperature

The water temperature of an aquarium is very important for the healthy living of the fish and the plants inside the aquarium. The ideal aquarium temperature for freshwater fish is around 29°C. Too hot or too cold water temperature call kill the fish and also effect the growth of the plants inside the aquarium. In this thesis a microcontroller based temperature control system has been developed for the aquarium to keep the temperature at the required level.

4.1.2 Feeding The Fish

It is very important that the fish should be fed daily, and once a day. The amount of food and the time of feeding are important factors affecting the health of the fish. Too much food can kill the fish.. Similarly, the fish could starve and die if enough food is not supplied to the aquarium. In general, enough food should be provided to last the fish for about 2 minutes. The most common reasons for premature fish death are over-feeding and inadequate maintenance. Also, excess food in the aquarium will begin to decay, and in the limited capacity of a aquarium, the water can become toxic rapidly.

In this thesis an automatic microcontroller based fish feed mechanism has been developed which deliver food to the fish on a daily basis and at the same time in each day.

There are a lot of types of fish foods are exist. Feeding your fish an improper diet is as common a mistake as overfeeding. Providing the correct diet is essential for fish growth and health. Dietary deficiencies will not only shorten the lifespan of fish and cause many diseases, but will also contribute to a deteriorating water quality by polluting the water.

The diet of fish varies based on their individual nutritional needs. Some require meaty foods (carnivores), some plants (herbivores) and some a combination of both (omnivores).

Besides requiring specific dietary and nutritional needs, fish also have varying feeding habits. Generally three feeding groups can be identified. There are bottom feeders, mid-water feeders, and top or surface feeders. Fish are usually easily categorized by their mouths. While bottom feeders have downward positioned mouth, an upward facing mouth is easily distinguishable for surface feeders.

To accommodate feeding preferences and habits, fish food is available in many shapes and sizes: flakes, pellets (sinking and floating), live foods, frozen, dried food, small, medium, and large for all species in various sizes and forms.

In nature, fish scavenge for food all day long, detecting food by various techniques such as movement, smell, visually, taste, color, and flavor. In aquariums fish need to be "trained" to eat food that otherwise would not be part of their diet.

Fish food consists basically of proteins, carbohydrates, fiber, fats, vitamins, and minerals. The diet is composed based on nutritional needs; Values are in %, average moisture content 6-10% and ~ 40% of carbohydrates.

Fish need protein mainly for their growth. Fish in general need less food to grow than other animals. It has to be noted that adult fish will not utilize all the protein provided in the food. It is important to use less protein rich foods for adult fish because the excess protein is being excreted as ammonia.

Fry need more protein, so do fast moving fish. High protein diets are recommended to make fish spawn as more protein is needed for egg production.

Less protein is required for cool water species like goldfish as the metabolism slows significantly the lower the temperature.

4.1.3 Aquarium Lighting

An aquarium needs light for about 4-6 hours a day. Different plants require different levels of aquarium lighting, usually measured in watts per gallon of aquarium water. If we are growing live plants or live corals, then we should increase the light duration to 6-8 hours per day. Light helps the plants to make photosynthesis. CO₂ is

also required for photosynthesis. CO_2 can come from fish respiration process and is released into the tank. However, if we have a heavily planted tank, we may want to get CO_2 injector for the aquarium because we will not get enough CO_2 from the fish. Algae is the bane of every aquarium owner. Algae grows in every healthy aquarium, no matter how well, or how often the tank is cleaned. There is absolutely no safe way to completely prevent or stop the growth of algae. Once it starts to grow in an aquarium, it does so very rapidly. A small patch of algae can triple in size in a few days. It is normal for algae to begin growing back in an aquarium within three weeks after the tank is cleaned. So, the light and nutrients in the water are the major causes of algae growth.

In this thesis a microcontroller based light control system has been developed. The system controls the aquarium lights at predefined times of the day.

4.1.4 Cleaning The Aquarium

Aquarium cleaning consists of cleaning the inside and the outside of the aquarium, and changing the water inside the aquarium. How often the water must be changed, depends on how many fish we have and the quality of our filtration system. In lightly stocked tanks, the recommended changing is 10 percent of the water once a week. We could probably get by with vacuuming the gravel once every two weeks depending on the population of our tank. Heavier stocked tanks will need larger (25% or more) weekly water changes and gravel vacuuming.

4.1.5 Designing a computer based aquarium system

The idea of designing a computer based aquarium system is not new. Evans [7] reports an aquarium management system called BAMU. This system is based on a NEC EV9835 microcontroller development system (see the block diagram of BAMU is Figure 4.1). The system controls the aquarium lighting, controls the aquarium

temperature, and also provides automatic food to the fish. This system in addition monitors the state of the air pressure pump in the aquarium. In addition, BAMU provides various alarms such as the failure of the heater, detection of low water inside the tank, failure of the lighting system and so on.

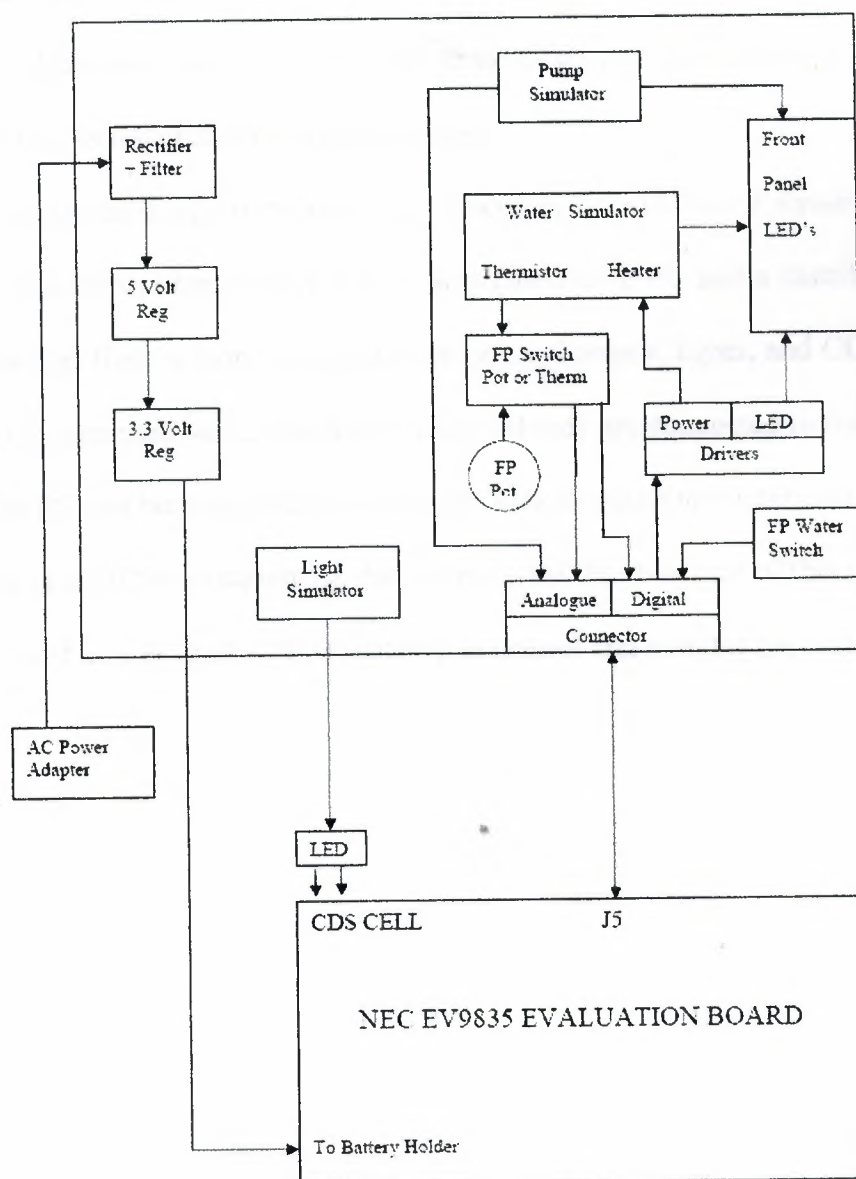


Figure 4.1. Block diagram of the BAMU aquarium control system

Carstensen [8] reports an 8031 microcontroller based aquarium control system. The microcontroller runs on its own but is connected to a PC with a terminal program using the RS232 communications protocol. The microcontroller is equipped with a 4-digit LCD display which shows the pH of the water, the water temperature, air temperature, time of the day, and the conductivity of the aquarium water. In addition, a feed switch is provided which is turned on at specified times of a day in order to provide food to the fish. The microcontroller receives instructions from the PC and then monitors and controls the aquarium. One disadvantage of this system is that it is PC based and a PC is required for its programming.

An automated aquarium system is described by *Automated Aquarium Systems Inc.* [9]. This system (see Figure 4.2) is also based on a PC and a distribution panel. Sensors such as flow sensor, temperature sensor, pH sensor, lights, and CO₂ sensor are placed in the aquarium water circulation path and they are connected to the distribution panel. The PC can be programmed to control various aquarium parameters and also to display the state of the aquarium on the screen. One disadvantage of this system is that it is also based on a PC and a PC is required to control and monitor the system.

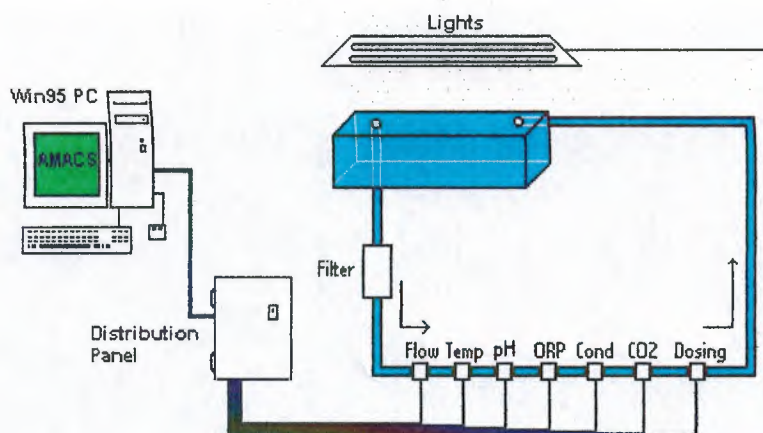


Figure 4.2. *Automated Aquarium Systems Inc.* aquarium control system

Neptune Systems [10] aquarium control system is based on a small portable unit with a LCD display as shown in Figure 4.3. The unit is used to monitor and also to control the pH, ORP, and the temperature of the aquarium water. A timer is provided which enables the user to start and stop various activities. The unit can be connected to a PC for programming. One advantage of this system is that the sunrise and sunset can be simulated, moon cycle can be simulated, and the pumps can be controlled to provide a waves in the water.



Figure 4.3. *Neptun Systems* aquarium control system

4.2 Development of the Aquarium Control System

Figure 4.4 shows the block diagram of the overall system. The system consists of seven microcontrollers. Six PIC16F84 type microcontrollers are used to control the date and time, automatic feeding, and flashing mechanisms. One PIC16F877 type microcontroller is used to control the temperature of the aquarium water. Parts of the system are described in more detail in the following sections.

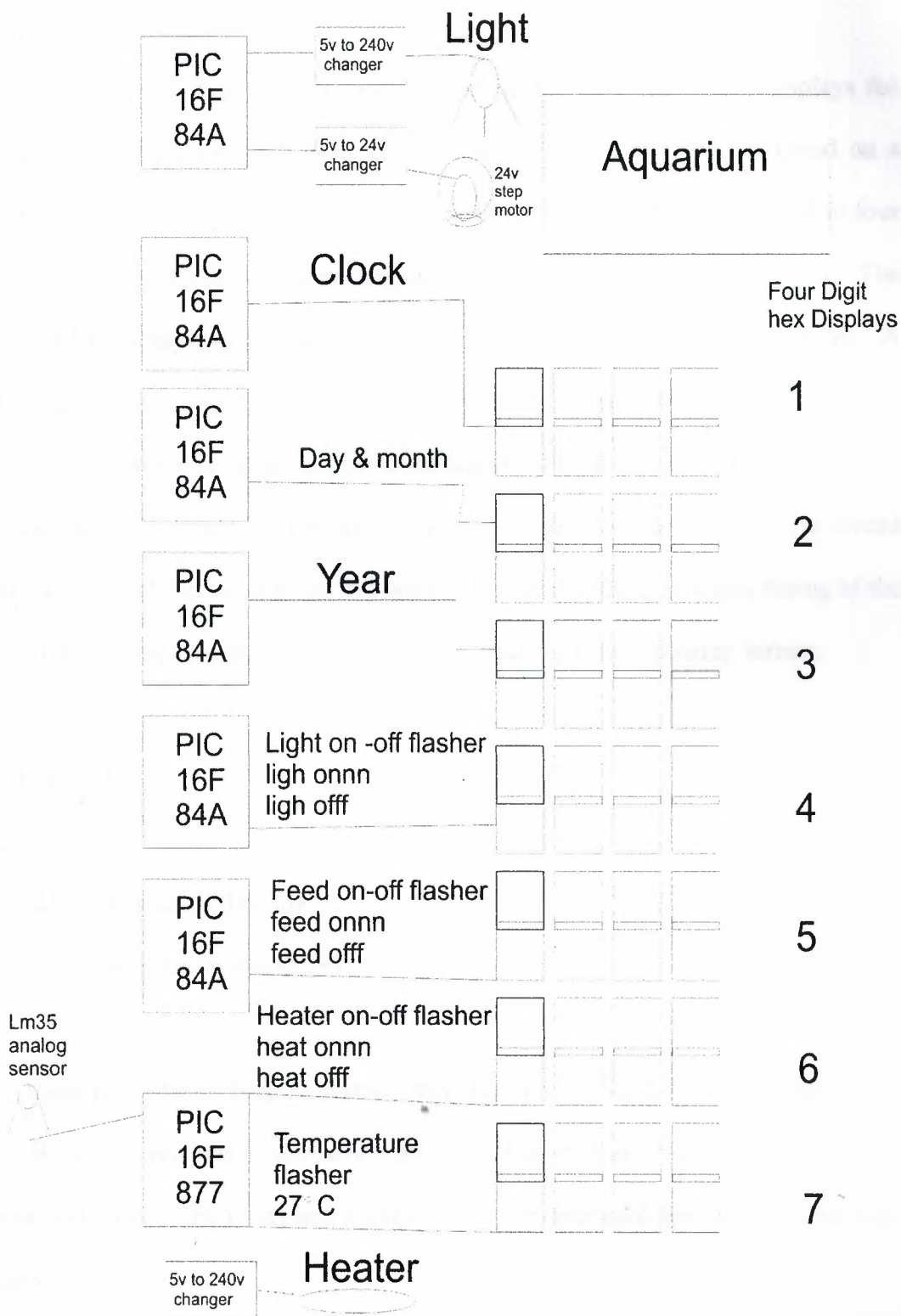


Figure 4.4. Block diagram of the system

4.2.1. The Clock Microcontroller

The system clock provides real time clock to the system and also displays the hours and minutes on a 4-digit, 7-segment display. The clock circuit is based on a PIC16F84 type microcontroller working with 4MHz crystal and it is connected to four seven segment hex displays. Seven segment hex display consists of 8 LEDs. The individual LEDs segments are turned ON when logic 0 is applied to the segment. A digit is selected by applying a logic 1 to the required digit. The displays are time multiplexed and each digit of the display is turned ON for about one millisecond.

The block diagram of the clock circuit is shown in Figure 4.5. The circuit diagram of the clock circuit is given in Figure 4.6. The assembly program listing of the clock circuit is given in Appendix A1. Time is displayed in the following format:

HH:MM

where,

HH is the hours (00 to 23)

MM is the minutes (00 to 59)

Basically a timer loop is formed and the minutes field is incremented every minute. When the minutes field reached 59, it is cleared to zero and the hours field is incremented by one. The hours and minutes fields are displayed and updated whenever necessary.

The clock displays are flashed ON and OFF with an interval of one second.

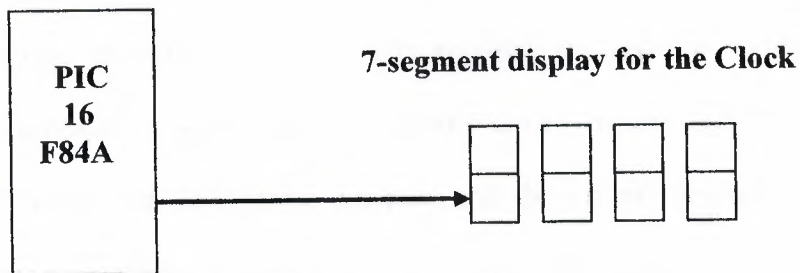


Figure 4.5. Block diagram of the Clock circuit

As shown in Figure 4.6, the microcontroller is powered from a +5V supply and is timed from a 4MHz crystal. PORT B of the microcontroller controls the LED segments. Similarly, PORT A of the PIC16F84 microcontroller controls the digits of the 7-segment LED display.

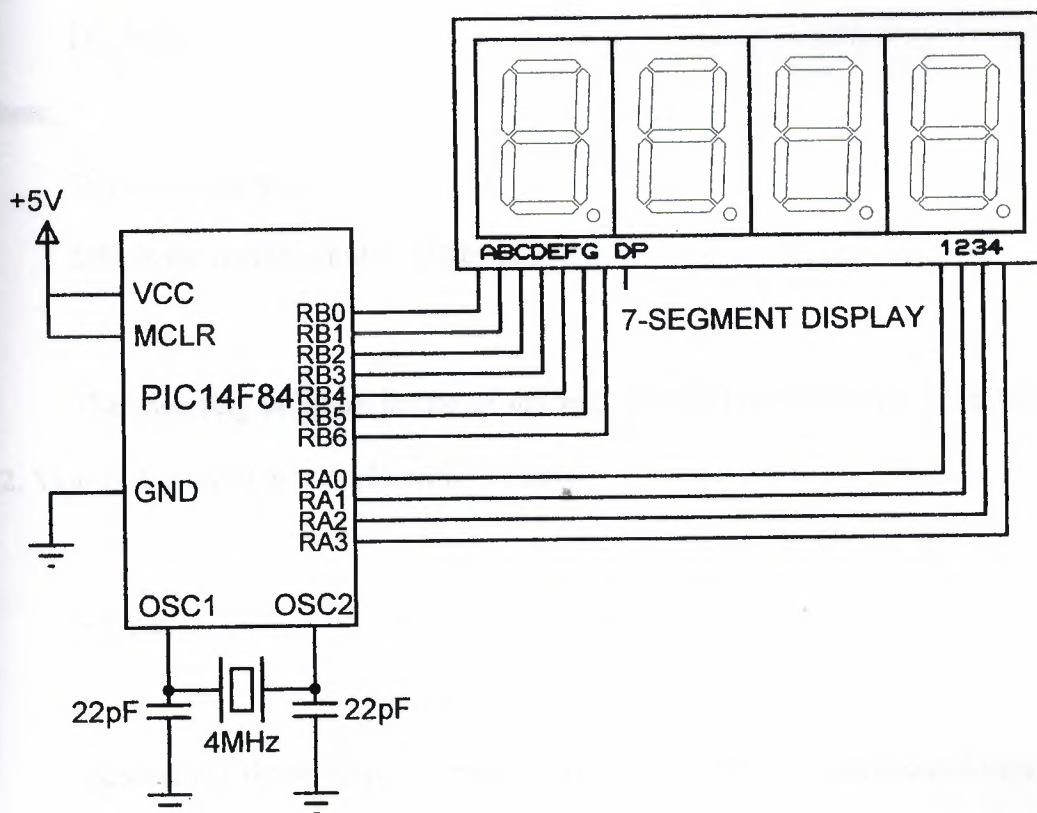


Figure 4.6. Circuit diagram of the Clock circuit

The display is flashed ON and OFF at one second intervals.

4.2.2 Date Microcontrollers

Two PIC16F84 type microcontrollers and two 7-segment hex displays are used for the real-time date information. The date microcontrollers provide real-time day and month information to the system. The first display shows the day and month, and the second display shows the year. The block diagram and the circuit diagram of the date microcontroller is as in Figure 4.5 and Figure 4.6 respectively, but two microcontrollers are used instead of one and each microcontroller drives a separate 7-segment hex display.

The date microcontroller displays the current month and the date in the following format:

DD:MM

where,

DD is the day number (01 to 31)

MM is the month number (01 to 12)

The assembly program listing of the year display program is given in Appendix A2. Year is displayed in the following format:

YYYY

Basically a timer loop is formed and the day field is incremented every day. When the day field reached 30 or 31, it is cleared to zero and the month field is

incremented by one. The days and minutes fields are displayed and updated whenever necessary. The displays are flashed ON and OFF at one second intervals.

4.2.3 Fish Feeding And Light Control

Automatic fish feeding control system is based on a motor and a glass which holds the food, and is controlled from a PIC16F84 type microcontroller. The microcontroller is powered from a +5V supply and is timed from a 4MHz crystal oscillator. Figure 4.7 shows the block diagram of the fish feeding control system.

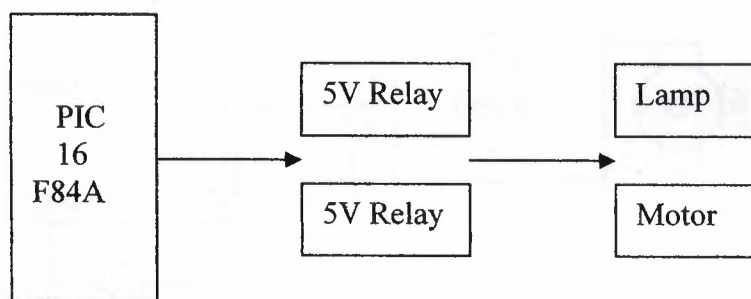


Figure 4.7. Block diagram of the fish feeding and light control

The motor rotates under the control of the microcontroller and this pushed the food out of the glass and into the aquarium. The motor operates from a 12V standard supply and thus a relay is used to control the motor from the microcontroller. The relay is connected to port pin RB1 of the microcontroller (bit 1 of PORT B).

The aquarium light control system is also controlled from the same PIC 16F84 microcontroller. The lights are programmed to turn ON at 17:00 o'clock every afternoon, and then turn OFF at 22:00 o'clock in the evening. Lights operate normally

with a 240V mains supply, and thus, a relay is used to control the lights. The relay is connected to port pin RB0 of the microcontroller (bit 0 of PORT B).

Figure 4.8 shows the circuit diagram of the fish feeding and light control circuit. The assembly program listing of the light control system is given in Appendix A4. Also, the assembly listing of the automatic feed control system is given in Appendix A3.

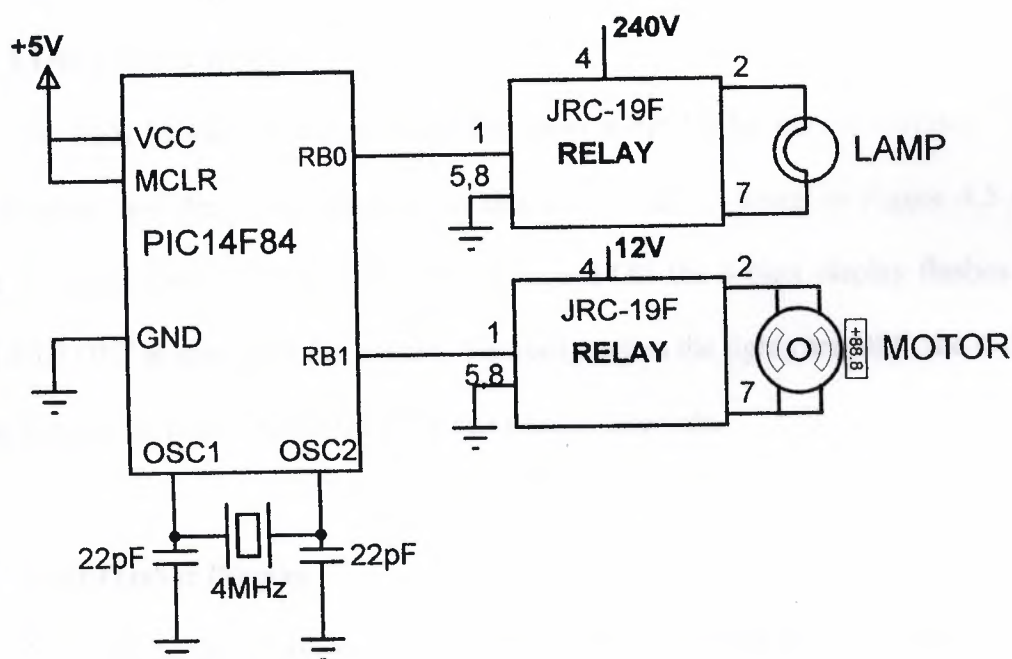


Figure 4.8. Circuit diagram of the fish feeding and light control circuit

The program has been adjusted so that the lighting and the feed times are as shown in Table 4.1. Notice in this table that the feeding starts at 17:00 hours and then stops at 17:01 hours. Similarly, lights are turned on between the hours of 17:00 – 22:00

HOURS	LIGHTS	FEEDER
00:00 – 17:00	OFF	OFF
17:00 – 17:01	ON	ON
17:01 – 22:00	ON	OFF
22:00 – 23:00	OFF	OFF

Table 4.1. Aquarium lighting and feeding times

4.2.4. Light Flasher Display

The light flasher display is controlled from a PIC16F84 microcontroller. The block diagram and the circuit diagram of this system are as given in Figure 4.5 and Figure 4.6 respectively. When the lights are turned ON the 4-digit display flashes the text “LIGH ON” at one second intervals. Similarly, when the lights are OFF, the 4-digit display flashes the text “LIGH OFFF” at one second intervals.

4.2.5. Feed Flasher Display

The feed flasher display is controlled from a PIC16F84 microcontroller. The block diagram and the circuit diagram of this system are as given in Figure 4.5 and Figure 4.6 respectively. When the feed mechanism is ON the 4-digit display flashes the text “FEED ON” at one second intervals. Similarly, when the mechanism is OFF, the 4-digit display flashes the text “FEED OFFF” at one second intervals.

4.2.6. Controlling The Aquarium Water Temperature

The last step is to make a digital thermometer and to control the temperature of the aquarium water. In this thesis, PIC-16F877 microcontroller is used with 4 MHz oscillator and the microcontroller is powered from a +5V supply.

Temperature is measured using a LM35 type analog sensor[4]. LM35 is a popular temperature sensor integrated circuit which can operate from a single +5V supply and can measure the temperature between the range 0°C to +100°C. The sensor provides a voltage which is directly proportional to the ambient temperature. The output voltage of the LM35 sensor is 10mV/°C. For example, if the ambient temperature is 15°C then the output voltage of LM35 will be 150mV. Similarly, if the ambient temperature is 30°C, the output voltage of the sensor will be 300mV and so on.

Figure 4.9 shows the block diagram of the temperature control system.

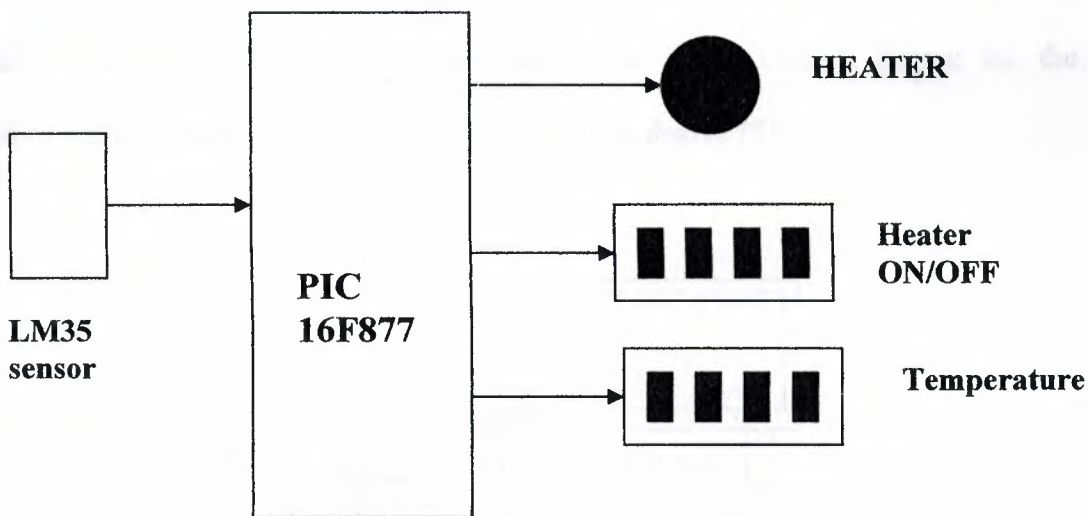


Figure 4.9. Block diagram of the temperature control system

The main reason of using PIC-16F877 microcontroller is it has an analog to digital converter. LM35 analog sensor is connected to one of the analog to digital (A/D) converter legs of the PIC16F877 microcontroller as shown in Figure 4.10.

LM35 analog sensor has 3 legs. One of the legs is connected to ground, the other leg is connected to the +5V supply, and the third leg is the output leg which is connected to analog input AN0 of the PIC16F877 microcontroller. The temperature is

measured and then displayed on a 4-digit 7-segment display every second. In addition, if the temperature is below the required value then the heater is operated to heat-up the aquarium water. If on the other hand the aquarium water temperature is above the required value then the heater is turned off.

The PIC16F877 microcontroller has 8 analog to digital converter (A/D) each of them is 10 bit. Fundamentally, it has only one A/D but analog legs are multiplexed and used the same A/D. Port A legs are used both digital or analog signaling. If these legs are used for analog, they are named with AN0-AN7. A/D normally works with 5V. For this reason, 10 bit input signal is to come into 1024 levels. Every level has $5000/1024 = 4.88\text{mV}$. The temperature sensor gives 10mV for every Celsius degree so, the sensitivity of the measuring temperature is 0.5 Celsius degree [5].

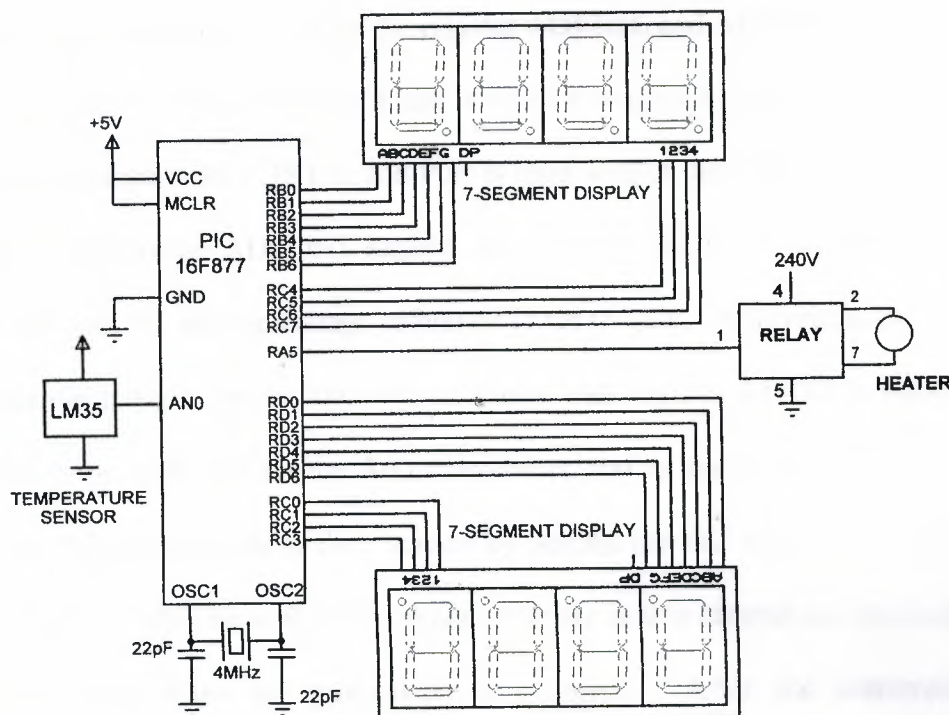


Figure 4.10. Circuit diagram of the temperature control system

The A/D conversion operation is carried out as follows:

- ADCON1 register is used for setting up the ports for analog or digital. For example, if ADCON1 register is '10001110' (0x8E), the AN0 (RA0) leg of port A comes analog and the rest of RA1-RA7 legs comes to digital input/output. In addition, the voltage of analog port is chosen by +VDD and ADRESH and ADRESL are rotated to the right and keep the 10 bit conversion.
- ADCON0 register is used for choosing analog channel and clock sequence. Afterwards, make active the analog to digital converter. If 4 MHz crystal is used and clock sequence is Fosc/8, the ADCON1 register must be '01000001' (0x41).
- To start the conversion, the GO/DONE bit must be logic 1.
- Wait until the conversion finished. The GO/DONE bit is logic 0, if the conversion is finished.
- Read the converted value from the register ADRESL and ADRESH.

The assembly listing of the program is given in Appendix A5. In the program, first of all we configure the PORT A and Port B input-output directions. Then, we load hexadecimal 0x8E to the ADCON1 register and RA0 is configured to be an analog input. In addition, the analog voltage reference is set to +5V. Afterwards, BANK0 is selected and the hexadecimal value 0x41 is loaded into register ADCON0 register in order to select the clock rate for the A/D conversion, and to enable the A/D converter circuit. The A/D conversion is then started by setting the GO bit of the ADCON0 register to logic 1. This bit is then tested continuously as it is cleared automatically by the microcontroller when the conversion is complete. After the conversion is completed, the 10 bit result is available in registers ADRESH and ADRESL. The high

2 bits of the converted data is stored in register ADRESH. Similarly, the low 8 bits of the converted data is stored in register ADRESL.

The analog to digital converter of the PIC16F877 microcontroller is 10 bit wide. In this case, the numerical value is between 0 and 1023. The reference voltage used is +5V (5000mV). So, a converted “n” numerical value has the real voltage value of “n x 5000/1024 mV”. For example, if the output value of the analog to digital converter is 256, the real value of the voltage read can be calculated as (n = 256): $256 \times 5000 / 1024 = 1250\text{mV}$. In this case, the analog voltage read at the AN0 leg of the microcontroller is +1.25 V. The sensor that used gives $10\text{mV}/^{\circ}\text{C}$. So the result temperature comes like this

$$V = \frac{5000 \times T}{1024 \times 10}$$

Or approximately, this is equivalent to T/2. In this case, the obtained result can be divided by two to reach the actual value of the voltage read at the analog input. The LM35 analog sensor read the temperature between 0 and 100°C . For example, if the temperature is 100°C , it gives 1000mV . The numerical conversion of this value is $1000 \times 1024 / 5000 = 204$. It is possible to keep this value in 8 bit register. In this case, high-byte is always 0 and low-byte gives the temperature. To summarize this, only taking 8 low bit and divided by two of converted numerical value gives the real temperature in $^{\circ}\text{C}$. The easiest way to divide a number by two is to rotate the register to the right by one place:

Divide_two

bcf status, C

rrf result, 1

return

At the end of this, the temperature is read from the LM35 sensor[6]. This read signal is then sent to the seven segment hex displays by using PORT D and PORT C. The actual process works like this: send a signal to the first seven segment hex display for a millisecond, and then send the other signal to the other seven segment hex display and repeat this process in a loop. The temperature program also controls the heater to keep the temperature at 29° Celsius which is the ideal temperature for the fish.

The two 7-segment 4-digit displays connected to the temperature control circuit display the current temperature of the aquarium and the status of the heater. The first display shows the text “CELS” and then after one second the temperature of the aquarium in °C. This is repeated continuously. For example, if the aquarium temperature is 29°C, the first display will show alternately: CELS 29°C CELS 29°C CELS and so on.

The second 4-digit display shows the status of the heater. If the heater is turned ON the text “HEAT” and then after one second the text “Onnn” is displayed. Similarly, if the heater is turned OFF, the text “HEAT” and then “OFFF” is displayed at one second intervals. This display is repeated continuously.

Figure 4.11 and Figure 4.12 show the picture of the aquarium control system.

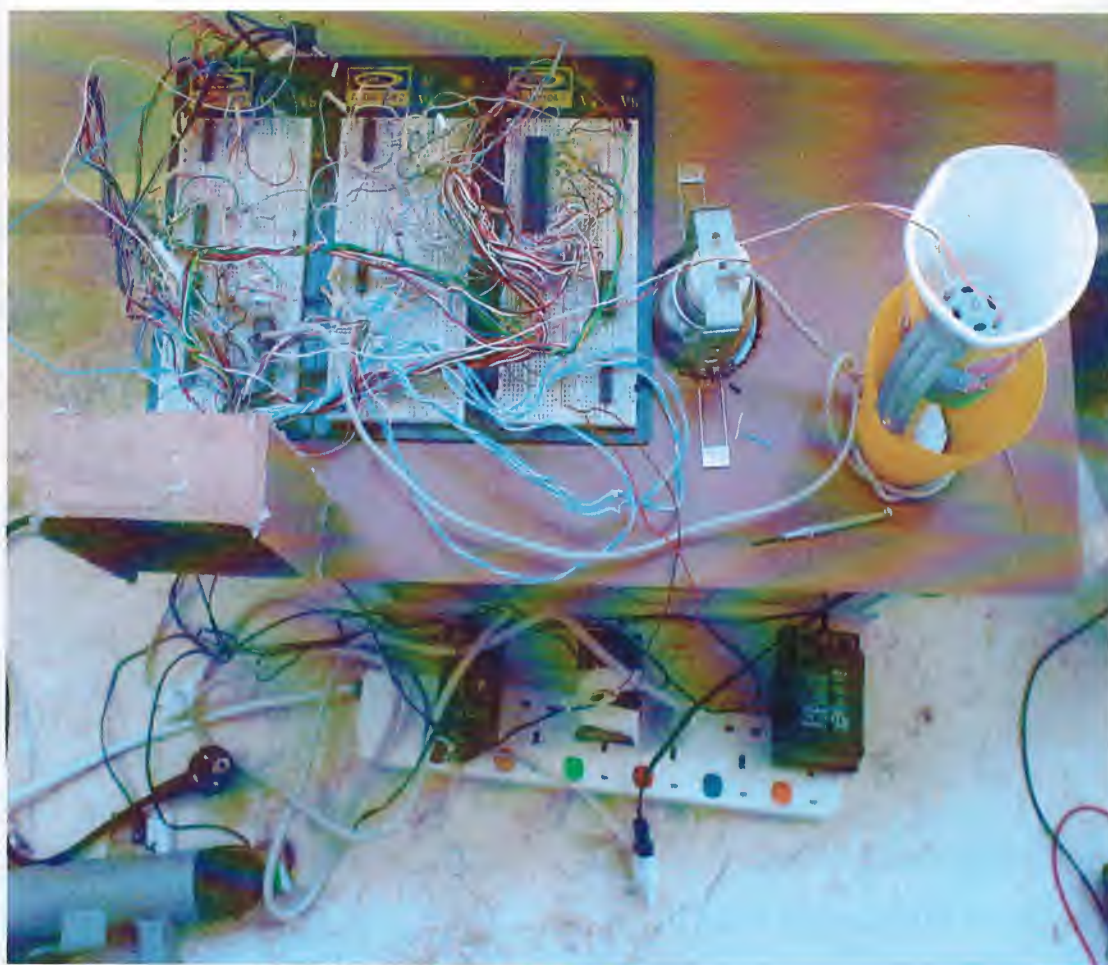


Figure 4.11. Picture of the aquarium control system

In the above picture the feed mechanism is shown on the right hand side. The microcontroller circuit built on a breadboard is shown at on the back left hand side. The 4-digit hex displays are at the front left hand side of the picture.



Figure 4.12. Picture of the aquarium control system

In Figure 4.12 the feed mechanism is shown at the right hand side of the picture. The 4-digit hex displays are shown stacked at the left hand side of the picture. The microcontroller circuit is shown in the middle of the picture.

4.2.7. Results

This thesis is about the development of a microcontroller based automatic aquarium control system.

Basically, an aquarium requires 4 types of services: feeding the fish, providing light to the aquarium, controlling the temperature of the aquarium, and changing the

water of the aquarium. In this thesis a microcontroller based system has been developed to control the first three services of an aquarium.

An automatic microcontroller based food feed mechanism has been developed which provides food to the fish at pre-specified times of a day. This system is based on a motor and two cups.

Aquarium lights are controlled from a microcontroller whereby 6-8 hours of light is provided to the aquarium daily.

The temperature of the aquarium is controlled using a microcontroller. A temperature sensor integrated circuit is used which measures the temperature of the aquarium water and gives this signal to a microcontroller. The microcontroller attempts to control the temperature of the water so that it is around 29°C which is the ideal temperature for the fish. If the temperature is below this value then a heater is operated to increase the temperature. If on the other hand the temperature of the aquarium water is above this value then the heater is turned off.

In addition to the control circuitry, a number of 7-segment LED displays are used to give information to the user, such as the time and date, state of the heater, and the state of the feed mechanism.

Table 5.1 shows the results obtained from a typical real aquarium. The temperature of the aquarium water, the state of the lights, and the feed mechanism have all been observed over a 24 hour period. As shown in the table the results are satisfactory and as expected. The lights and the feed mechanism were operated at the required times of the day. The average temperature of the aquarium was kept at 29°C despite of the changes in the external ambient temperature.

HRS	LIGHTS	FEED	AQUARIUM TEMP	AMBIENT TEMP
00:00	OFF	OFF	29	22
01:00	OFF	OFF	29	21
02:00	OFF	OFF	29	20
03:00	OFF	OFF	29	21
04:00	OFF	OFF	29	22
05:00	OFF	OFF	29	23
06:00	OFF	OFF	29	24
07:00	OFF	OFF	29	25
08:00	OFF	OFF	29	26
09:00	OFF	OFF	29	27
10:00	OFF	OFF	29	28
11:00	OFF	OFF	29	29
12:00	OFF	OFF	29	29
13:00	OFF	OFF	29	29
14:00	OFF	OFF	29	29
15:00	OFF	OFF	29	29
16:00	OFF	OFF	29	28
17:00	ON	ON	29	27
17:01	ON	OFF	29	26
18:00	ON	OFF	29	25
19:00	ON	OFF	29	24

20:00	ON	OFF	29	24
21:00	ON	OFF	29	23
22:00	OFF	OFF	29	22
23:00	OFF	OFF	29	22

Table 5.1 Results of the measurement using a real aquarium.

The advantages and disadvantages of this thesis are shown in the table 5.2.

Reliability	Seven microcontrollers are used. If one is not working the others are continue to work. The other systems stop completely.
Cost	Seven microcontrollers are used; it's not good for the cost.
Ease	It has clock and date. It automatically makes feeding, controls temperature of the water and controls the light.

Table 5.2 Advantages and disadvantages of the thesis.

CHAPTER 5

5. CONCLUSION

This thesis has described the development of a microcontroller based automatic aquarium control system. The system is based on seven PIC type microcontrollers. The system developed can control the temperature of the aquarium water so that the water temperature stays at the optimum value for the healthy living of the fish. The system can also deliver food to the fish automatically at predefined times of the day. Proper feeding of the fish in an aquarium is very important since over-feeding could kill the fish, and under-feeding could starve the fish to death. Another feature of the system is that the aquarium lights can be turned on or off at predefined times of a day. Proper lighting of an aquarium is very important for the maintenance of the aquarium material, and for the healthy living of the fish and the plants inside the aquarium.

The microcontroller based aquarium control system developed in this thesis also uses a number of 7-segment LED displays to show the current status of the system. For example, the date and time is displayed continuously, the temperature of the aquarium is also displayed continuously. When the feed mechanism operates, or when the lights are turned on or off, a flashing display indicates the state of the aquarium.

The microcontroller based automatic aquarium control system described in this thesis can be developed further. For example, a timer based automatic changing of the aquarium water system can be developed so that the water can be changed at predefined days of the week. Also, an external keypad can be provided to program the required temperature, feed times and the time that the lights should be turned on and off.

Another suggestion is to use a more powerful microcontroller with multi-tasking capabilities. This way, only one microcontroller could be used to control all aspects of an aquarium. Also, LCD based displays could be used instead of the simple 7-segment displays used in the thesis. It should be possible to display the date and time and the various aquarium parameters on a single LCD display. Controlling the heater by using a red lamp is very important for the fish safety. Because the fish is dead, if the heater is not working. Measuring the Ph degree of the water and, if the water Ph degree is low, the program changes the water of the aquarium automatically. These types of alarms are valid for feeding and light control units. At the end, these are all future work of this thesis.

REFERENCES

- [1] Computer desktop encyclopedia, the computer language co. inc.,1998,
www.computerlanguage.com/techweb.html.
- [2] PIC16F84A datasheet, microchip company, www.microchip.com
- [3] PIC16F877 Datasheet, microchip company, www.microchip.com
- [4] LM35 datasheet, www.national.com/JPN/ds/LM/LM35.pdf.
- [5] Doğan İbrahim and Hamit İ Mustafa, PIC Programming and PIC Projects, Bilesim
Yayincilik, 36, Turkey, 2001.
- [6] Barbaros Asuroğlu, Antrak Magazine, 43,2001, www.antrak.org.tr.
- [7] Brian Evans, Aquarium Management System (BAMU), EDN, July 12, 2006.
- [8] Thomas Cartensen, Computerized Tank, www.thekrib.com.
- [9] Automated Aquarium Systems Inc., www.automatedaquariums.com.
- [10] Neptun Systems, www.neptunesys.com/aquaController2.htm
- [11] MPLAB integrated development environment,
www.microchip.com/stellent/idcplg?IdcService=SS.
- [12] MITS Altair 8800, 1974, <http://www.weller.to/com/comp-mits-altair.htm>.
- [13] Information about PIC microcontroller ,www.electronic-engineering.ch/microchip.
- [14] Von neuman and Harvard architecture,
www.esacademy.com/automation/faq/primer/4.htm.
- [15] Information about aquariums, <http://www.aquariumfish.net/pages/information.htm>.

APPENDICES

Appendix 1 – Year Program

```
;  
;  
;    MAHMUT KISACIK 20034186 YEAR USING PIC16F84A  
;  
;  
LIST p=16f84A                ;MPLAB command  
#include p16f84A.inc          ;MPLAB command  
__CONFIG _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF ;crystal osc  
DIGIT1 EQU h'0D'              ;initializations  
DIGIT2 EQU h'0E'  
COUNT EQU h'0F'  
COUNTER11 EQU h'12'  
COUNTER22 EQU h'13'  
DIGIT4 EQU h'14'  
DIGIT3 EQU h'15'  
COUNTER12 EQU h'16'  
DIGIT5 EQU h'17'  
DIGIT6 EQU h'18'  
DIGIT7 EQU h'19'  
COUNTERS EQU h'20'  
D1 EQU h'21'  
D2 EQU h'22'  
D3 EQU h'23'  
D4 EQU h'24'
```


COUNTX EQU h'25'

GUN1 EQU h'26'

GUN2 EQU h'27'

AY1 EQU h'28'

AY2 EQU h'29'

YIL1 EQU h'30'

YIL2 EQU h'31'

YIL3 EQU h'32'

YIL4 EQU h'33'

INIT

BSF STATUS,5 ;Jump Bank1

CLRF TRISB ;All pins of PortB is output

CLRF TRISA ;All pins of PortA is output

BCF STATUS,5 ;Jump Bank0

CLRF PORTA ;All pins are at logic 0 level

CLRF PORTB ;All pins are at logic 0 level

MAIN

MOVLW h'02' ;D4=02

MOVWF D4

MOVLW h'03' ;D3=03

MOVWF D3

MOVLW h'05' ;D2=05

MOVWF D2

MOVLW h'09' ;D1=09 initial time is 23:59

MOVWF D1

MOVLW h'03'

MOVWF GUN1 ;Day =31

MOVLW h'01'

MOVWF GUN2

MOVLW h'01'

MOVWF AY1 ;Month=12

MOVLW h'02'

MOVWF AY2

MOVLW h'02'

MOVWF YIL4

MOVLW h'00' ;Year=2006

MOVWF YIL3

MOVLW h'00'

MOVWF YIL2

MOVLW h'06'

MOVWF YIL1

GOTO LOOPN99 ;jump to loopn99

LOOPA1

INCFSZ YIL1,F ;year increment

MOVLW h'00'

MOVWF AY1 ;clearing the first digit of month

```

MOVLW h'01'
MOVWF AY2           ; pass to the first month
MOVLW h'00'         ;clearing the first digit of days
MOVWF GUN1
MOVLW h'01'         ;pass to the first day
MOVWF GUN2
GOTO LOOPA5

LOOPA2
MOVLW h'00'         ;clearing the first digit of month
MOVWF AY2

LOOPA3
MOVLW h'00'         ;clearing the second digit of day
MOVWF GUN1

LOOPA4
MOVLW h'00'         ;clearing the first digit of days
MOVWF GUN2

LOOPA5
MOVLW h'00'         ;clearing the first digit of hours
MOVWF D4

LOOPN77
MOVLW h'00'         ;clearing the second digit of hours
MOVWF D3

LOOPN66
MOVLW h'00'         ;clearing the first digit of minutes

```

```

MOVWF D2

LOOPN55
    MOVLW h'00'           ;clearing the second digit of minutes
    MOVWF D1

LOOPN99
    MOVLW 0X048           ;loop for real time clock speed
    MOVWF COUNTERS

LOOPV
    MOVLW 0X037
    MOVWF COUNTX

LOOPVV
    MOVLW h'00'           ;clearing porta
    MOVWF PORTA
    MOVLW h'C0'           ;c0 means zero to portb
    MOVWF PORTB
    CALL DELAY
    CALL DELAY
    CALL DELAY
    CALL DELAY

    DECFSZ COUNTX,F
    GOTO LOOPVV           ;loop for a second without led light
    MOVLW 0X038

```


MOVWF COUNTER12

LOOP123

MOVLW h'00' ;clearing porta

MOVWF PORTA

MOVF YIL1,W ;entering yil1 to w register

CALL SEVNS_TABL ;changing it to the seven segment for the same

number

MOVWF PORTB ;output this number by using portb

MOVLW h'01' ;opening first digit

MOVWF PORTA ;by using porta

CALL DELAY ;give a milisecond

MOVLW h'00' ;clearing porta

MOVWF PORTA

MOVF YIL2,W ;entering yil2 to w register

CALL SEVNS_TABL ;changing it to the seven segment for the same

number

MOVWF PORTB ;output this number by using portb

MOVLW h'02' ;opening second digit

MOVWF PORTA ;by using porta

CALL DELAY ;give a milisecond

MOVLW h'00' ;clearing porta

MOVWF PORTA

MOVF YIL3,W ;entering yil3 to w register

CALL SEVNS_TABL ;changing it to the seven segment for the same
number

MOVWF PORTB ;output this number by using portb

MOVLW h'04' ;opening third digit

MOVWF PORTA ;by using porta

CALL DELAY ;give a milisecond

MOVLW h'00' ;clearing porta

MOVWF PORTA ; entering yil4 to w register

MOVF YIL4,W

CALL SEVNS_TABL ;changing it to the seven segment for the same

number

MOVWF PORTB ;output this number by using portb

MOVLW h'08' ;opening fourth digit

MOVWF PORTA ;by using porta

CALL DELAY ;give a milisecond

DECFSZ COUNTER12,F

GOTO LOOP123 ;loop for a second 4 digit at the same time

DECFSZ COUNTERS,F

GOTO LOOPV ;loop continues about a minute

BSF PORTA,4

MOVLW h'09' ;test if the first digit of minutes is 9

```

SUBWF D1,W
BTFSS STATUS,2
GOTO ART221           ;if it not 9
GOTO LOOP1           ;if it is 9
ART221
INCFSZ D1,F           ;increment the first digit of the minutes
GOTO LOOPN99         ;go back and continue for one minute

LOOP1
    MOVLW h'05'       ;test if the second digit of minutes is 5
    SUBWF D2,W
    BTFSS STATUS,2
    GOTO LOOP2       ;if it is not 5
    GOTO LOOPN2      ;if it is 5

LOOP2
    INCFSZ D2,F       ;increment the second digit of minutes
    GOTO LOOPN55      ;go back and continue for one minute

LOOPN2
    MOVLW h'03'       ;test if the first digit of hours is 3
    SUBWF D3,W
    BTFSS STATUS,2
    GOTO LOOP3       ;if it is not 3
    GOTO LOOPN3      ;if it is 3

LOOP3

```

LOOPN6	INCFSZ D3,F	;increment the first digit of hours
	GOTO LOOPN66	;go back and continue for one minute
LOOPN3		
	MOVLW h'02'	;test if the second digit of hours is 2
	SUBWF D4,W	
	BTFSS STATUS,2	
LOOPN4	GOTO LOOP5X	;if it is not 2
	GOTO LOOPN5	;if it is 2
LOOP5X		
LOOPN7	INCFSZ D4,F	;increment the second digit of hours
	GOTO LOOPN77	;go back and continue for one minute
LOOPN5		
LOOPN8	MOVLW h'01'	;test if the first digit of days is 1
	SUBWF GUN2,W	
	BTFSS STATUS,2	
	GOTO LOOPT2	;if it is not 1
	GOTO LOOPT3	;if it is 1
LOOPT3		
LOOPN9	MOVLW h'03'	;test if the second digit of days is 3
	SUBWF GUN1,W	
	BTFSS STATUS,2	
	GOTO LOOPT4	;if it is not 3
	GOTO LOOPT5	;if it is 3
LOOPT2		

LOOP4

```
MOV LW h'09'      ;test if the first digit of days is 9
SUBWF GUN2,W
BTFSS STATUS,2
GOTO LOOP6        ;if it is not 9
GOTO LOOP7        ;if it is 9
```

LOOP6

```
INCFSZ GUN2,F      ;increment the first digit of days
GOTO LOOPA5        ;go back and continue for one day
```

LOOP7

```
INCFSZ GUN1,F      ;increment the second digit of days
GOTO LOOPA4        ; go back and continue for one day
```

LOOP5

```
MOV LW h'02'      ;test if the first digit of month is 2
SUBWF AY2,W
BTFSS STATUS,2
GOTO LOOPM2        ;if it is not 2
GOTO LOOPM3        ;if it is 2
```

LOOPM3

```
MOV LW h'01'      ;test if the second digit of month is 1
SUBWF AY1,W
BTFSS STATUS,2
GOTO LOOPM4        ;if it is not 1
GOTO LOOPA1        ;if it is 1
```

```

LOOPM2                                ;NOT NECESSARY

LOOPM4

    MOVLW h'09'

    SUBWF AY2,W                       ;test if the first digit of month is 9

    BTFSS STATUS,2

    GOTO LOOPM6                       ;if it is not 9

    GOTO LOOPM7                       ;if it is 9

LOOPM6

    INCFSZ AY2,F                      ;increment the first digit of days

    GOTO LOOPA3                       ; go back and continue for one day

LOOPM7

    INCFSZ AY1,F                      ;increment the second digit of days

    GOTO LOOPA2                       ; go back and continue for one day

SEVNS_TABL

    ADDWF PCL,F ;PCL=W(h'04')+PCL.

    RETLW h'C0' ;0

    RETLW h'F9' ;1

    RETLW h'A4' ;2

    RETLW h'B0' ;3

    RETLW h'99' ;4

    RETLW h'92' ;5

    RETLW h'82' ;6

    RETLW h'F8' ;7

    RETLW h'80' ;8

```

```

    RETLW h'90' ;9
    RETLW h'88' ;A
    RETLW h'83' ;B
    RETLW h'C6' ;C
    RETLW h'A1' ;D
    RETLW h'86' ;E
    RETLW h'8E' ;F
    RETLW h'7F' ;.
DELAY
    MOVLW h'04'
    MOVWF COUNTER11      ;delayl loop for a milisecond
LOOP9X
    MOVLW h'FF'
    MOVWF COUNTER22
LOOP99
    DECFSZ COUNTER22,F
    GOTO LOOP99
    DECFSZ COUNTER11,F
    GOTO LOOP9X
    RETURN
    END

```

Appendix 2 – Month and Day program.

```
;
;    MAHMUT KISACIK 20034186 MONTH AND DAY USING PIC16F84A
;

LIST p=16f84A      ;mplab command

#include p16f84A.inc  ;mplab command

__CONFIG _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF  ;crystal osc

DIGIT1 EQU h'0D'

DIGIT2 EQU h'0E'

COUNT EQU h'0F'

COUNTER11 EQU h'12'      ;initializations

COUNTER22 EQU h'13'

DIGIT4 EQU h'14'

DIGIT3 EQU h'15'

COUNTER12 EQU h'16'

DIGIT5 EQU h'17'

DIGIT6 EQU h'18'

DIGIT7 EQU h'19'

COUNTERS EQU h'20'

D1 EQU h'21'

D2 EQU h'22'

D3 EQU h'23'

D4 EQU h'24'

COUNTX EQU h'25'
```


GUN1 EQU h'26'

GUN2 EQU h'27'

AY1 EQU h'28'

AY2 EQU h'29'

INIT

BSF STATUS,5 ;Jump Bank1

CLRF TRISB ;All pins of PortB is output

CLRF TRISA ;All pins of PortA is output

BCF STATUS,5 ;Jump Bank0

CLRF PORTA ;All pins are at logic 0 level

CLRF PORTB ;All pins are at logic 0 level

MAIN

MOVLW h'02' ;d4=02

MOVWF D4

MOVLW h'03' ;d3=03

MOVWF D3 ;clock=23:59

MOVLW h'05' ;d2=05

MOVWF D2

MOVLW h'09' ;d1=09

MOVWF D1

MOVLW h'03'

MOVWF GUN1 ;day=31

MOVLW h'01'

MOVWF GUN2

MOVLW h'01'

MOVWF AY1 ;month=12

MOVLW h'02'

MOVWF AY2

GOTO LOOPN99

LOOPA1

MOVLW h'00' ;clearing the first digit of month

MOVWF AY1

MOVLW h'01' ;clearing the second digit of month and initial it by 1

MOVWF AY2

MOVLW h'00' ;clearing the first digit of day

MOVWF GUN1

MOVLW h'01' ;clearing the second digit of day and initial it by 1

MOVWF GUN2

GOTO LOOPA5

LOOPA2

MOVLW h'00' ;clearing the second digit of month

MOVWF AY2

LOOPA3

MOVLW h'00' ;clearing the first digit of day

MOVWF GUN1

LOOPA4

MOVLW h'00' ;clearing the second digit of day

MOVWF GUN2

```

LOOPA5                ;clearing the first digit of hours

    MOVLW h'00'

    MOVWF D4

LOOPN77

    MOVLW h'00'        ;clearing the second digit of hours

    MOVWF D3

LOOPN66

    MOVLW h'00'        ;clearing the first digit of minutes

    MOVWF D2

LOOPN55

    MOVLW h'00'        ;clearing the second digit of minutes

    MOVWF D1

LOOPN99

    MOVLW 0X048        ;loop for one minute

    MOVWF COUNTERS

LOOPV

    MOVLW 0X037

    MOVWF COUNTX

LOOPVV

    MOVLW h'00'        ;clears porta, clears selection of hex displays

    MOVWF PORTA

    MOVLW h'C0'        ;outputs 0

    MOVWF PORTB

    CALL DELAY

```

```

CALL DELAY
CALL DELAY
CALL DELAY
DECFSZ COUNTX,F      ;loop for a second without led light
GOTO LOOPVV
BCF PORTA,4
MOVLW 0X038
MOVWF COUNTER12
LOOP123
    MOVLW h'00'      ;clears porta, clears selection of hex displays
    MOVWF PORTA
    MOVF AY2,W        ;entering ay2 to w register
    CALL SEVNS_TABL   ;changing it to the seven segment for the same
number
    MOVWF PORTB       ;output this number by using portb
    MOVLW h'01'       ;selecting the first hex display
    MOVWF PORTA       ;by using porta
    CALL DELAY        ;give a milisecond
    MOVLW h'00'
    MOVWF PORTA       ;clears porta, clears selection of hex displays
    MOVF AY1,W        ;entering ay2 to w register
    CALL SEVNS_TABL   ;changing it to the seven segment for the same
number
    MOVWF PORTB       ;output this number by using portb

```


MOVLW h'02'	;selecting the second hex display
MOVWF PORTA	;by using porta
CALL DELAY	;give a milisecond
MOVLW h'00'	;clears porta, clears selection of hex displays
MOVWF PORTA	
MOVF GUN2,W	;entering gun2 to w register
CALL SEVNS_TABL	;changing it to the seven segment for the same

number

MOVWF PORTB	;output this number by using portb
MOVLW h'04'	;selecting the third hex display
MOVWF PORTA	; by using porta
CALL DELAY	;give a milisecond
MOVLW h'00'	;clears porta, clears selection of hex displays
MOVWF PORTA	;entering gun1 to w register
MOVF GUN1,W	
CALL SEVNS_TABL	;changing it to the seven segment for the same

number

MOVWF PORTB	;output this number by using portb
MOVLW h'08'	;selecting the fourth hex display
MOVWF PORTA	; by using porta
CALL DELAY	
DECFSZ COUNTER12,F	
GOTO LOOP123	;loop for see all hex display at the same time for

one sec

```

DECFSZ COUNTERS,F

GOTO LOOPV           ;loop for one minute

BSF PORTA,4

MOVLW h'09'          ;test if first digit of minutes is 9

SUBWF D1,W

BTFSS STATUS,2

GOTO ART221           ;if it is not 9

GOTO LOOP1           ;if it is 9

ART221

    INCFSZ D1,F        ;increment the first digit of minutes

    GOTO LOOPN99       ;go back to the loop for one minute

LOOP1

    MOVLW h'05'        ;test if the second digit of the minutes is 5

    SUBWF D2,W

    BTFSS STATUS,2

    GOTO LOOP2         ;if it is not 5

    GOTO LOOPN2        ;if it is 5

LOOP2

    INCFSZ D2,F        ;increment the second digit of minutes

    GOTO LOOPN55       ;go back to the loop for one minute

LOOPN2

    MOVLW h'03'        ;test if the first digit of the hours is 3

    SUBWF D3,W

    BTFSS STATUS,2

```

GOTO LOOP3 ;if it is not 3

GOTO LOOPN3 ;if it is 3

LOOP3

INCF SZ D3,F ;increment the first digit of hours

GOTO LOOPN66 ;go back to the loop for one minute

LOOPN3

MOVLW h'02' ;test if the second digit of the hours is 2

SUBWF D4,W

BTFSS STATUS,2

GOTO LOOP5X ;if it is not 2

GOTO LOOPN5 ;if it is 2

LOOP5X

INCF SZ D4,F ;increment the first digit of hours

GOTO LOOPN77 ;go back to the loop for one minute

LOOPN5

MOVLW h'01' ;test if the first digit of the days is 1

SUBWF GUN2,W

BTFSS STATUS,2

GOTO LOOPT2 ;if it is not 1

GOTO LOOPT3 ;if it is 1

LOOPT3

MOVLW h'03' ;test if the second digit of the days is 3

SUBWF GUN1,W

BTFSS STATUS,2

GOTO LOOPT4 ;if it is not 3

GOTO LOOPT5 ;if it is 3

LOOPT2

LOOPT4

MOVLW h'09' ;test if the first digit of the days is 3

SUBWF GUN2,W

BTFSS STATUS,2

GOTO LOOPT6 ;if it is not 9

GOTO LOOPT7 ;if it is 9

LOOPT6

INCFSZ GUN2,F ;increment the first digit of days

GOTO LOOPA5 ;go back to the loop for one minute

LOOPT7

INCFSZ GUN1,F ;increment the second digit of days

GOTO LOOPA4 ;go back to the loop for one minute

LOOPT5

MOVLW h'02' ;test if the first digit of the months is 2

SUBWF AY2,W

BTFSS STATUS,2

GOTO LOOPM2 ;if it is not 2

GOTO LOOPM3 ;if it is 2

LOOPM3

MOVLW h'01' ;test if the second digit of the months is 1

SUBWF AY1,W


```

    BTFSS STATUS,2
    GOTO LOOPM4           ;if it is not 1
    GOTO LOOPA1           ;if it is 1
LOOPM2                   ;NOT NECESSARY
LOOPM4
    MOVLW h'09'           ;test if the first digit of the months is 9
    SUBWF AY2,W
    BTFSS STATUS,2
    GOTO LOOPM6           ;if it is not 9
    GOTO LOOPM7           ;if it is 9
LOOPM6
    INCFSZ AY2,F           ;increment the first digit of months
    GOTO LOOPA3           ;go back to the loop for one minute
LOOPM7
    INCFSZ AY1,F           ;increment the second digit of months
    GOTO LOOPA2           ;go back to the loop for one minute
SEVNS_TABL
    ADDWF PCL,F ;PCL=W(h'04')+PCL.
    RETLW h'C0' ;0
    RETLW h'F9' ;1
    RETLW h'A4' ;2
    RETLW h'B0' ;3
    RETLW h'99' ;4
    RETLW h'92' ;5

```

APR RETLW h'82' ;6

RET LW h'F8' ;7

RET LW h'80' ;8

RET LW h'90' ;9

RET LW h'88' ;A

RET LW h'83' ;B

RET LW h'C6' ;C

RET LW h'A1' ;D

RET LW h'86' ;E

RET LW h'8E' ;F

RET LW h'7F' ;.

DELAY

DA MOVLW h'04'

DM MOVWF COUNTER11

LOOP9X

DM MOVLW h'FF'

DM MOVWF COUNTER22

LOOP99 ;delay loop for a milisecond

DM DECFSZ COUNTER22,F

DM GOTO LOOP99

DM DECFSZ COUNTER11,F

DM GOTO LOOP9X

DM RETURN

DM END

Appendix 3 – Clock program.

```
;  
;    MAHMUT KISACIK 20034186 CLOCK USING PIC16F84A  
;  
  
LIST p=16f84A           ;mplab command  
  
#include p16f84A.inc     ;mplab command  
  
__CONFIG _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF ;crystal osc  
  
DIGIT1 EQU h'0D'  
  
DIGIT2 EQU h'0E'  
  
COUNT EQU h'0F'  
  
COUNTER11 EQU h'12'      ;initializations  
  
COUNTER22 EQU h'13'  
  
DIGIT4 EQU h'14'  
  
DIGIT3 EQU h'15'  
  
COUNTER12 EQU h'16'  
  
DIGIT5 EQU h'17'  
  
DIGIT6 EQU h'18'  
  
DIGIT7 EQU h'19'  
  
COUNTERS EQU h'20'  
  
D1 EQU h'21'  
  
D2 EQU h'22'  
  
D3 EQU h'23'  
  
D4 EQU h'24'  
  
COUNTX EQU h'25'
```

INIT

BSF STATUS,5 ;Jump Bank1

CLRF TRISB ;All pins of PortB is output

CLRF TRISA ;All pins of PortA is output

BCF STATUS,5 ;Jump Bank0

CLRF PORTA ;All pins are at logic 0 level

CLRF PORTB ;All pins are at logic 0 level

MAIN

MOVLW h'02'

MOVWF D4

MOVLW h'03'

MOVWF D3 ;initialization clock 23:59

MOVLW h'05'

MOVWF D2

MOVLW h'09'

MOVWF D1

GOTO LOOPN99

LOOPN88

MOVLW h'00' ;clearing second digit of hours

MOVWF D4

LOOPN77

MOVLW h'00' ;clearing first digit of hours

MOVWF D3

LOOPN66


```

    MOVLW h'00'           ;clearing second digit of minutes
    MOVWF D2

LOOPN55
    MOVLW h'00'           ;clearing first digit of minutes
    MOVWF D1

LOOPN99
    MOVLW 0X048           ;loop for one minute
    MOVWF COUNTERS

LOOPV
    MOVLW 0X037
    MOVWF COUNTX

LOOPVV
    MOVLW h'00'           ;clears porta, clears selection of hex displays
    MOVWF PORTA
    MOVLW h'C0'           ;output 0
    MOVWF PORTB
    CALL DELAY
    CALL DELAY
    CALL DELAY
    CALL DELAY

    DECFSZ COUNTX,F
    GOTO LOOPVV           ;delay loop for a second without led light
    BCF PORTA,4

```

MOVLW 0X038

MOVWF COUNTER12

LOOP123

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D1,W ;entering d1 to w register

CALL SEVNS_TABL ;changing it to the seven segment for the same

number

MOVWF PORTB ;output this number by using portb

MOVLW h'01' ;selecting the first hex display

MOVWF PORTA ;by using porta

CALL DELAY ;give a millisecond delay

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D2,W ;entering d2 to w register

CALL SEVNS_TABL ;changing it to the seven segment for the same

number

MOVWF PORTB ;output this number by using portb

MOVLW h'02' ;selecting the second hex display

MOVWF PORTA ;by using porta

CALL DELAY ;give a millisecond delay

MOVLW h'00'	;clears porta, clears selection of hex displays
MOVWF PORTA	
MOVF D3,W	;entering d3 to w register
CALL SEVNS_TABL	;changing it to the seven segment for the same
number	

MOVWF PORTB	;output this number by using portb
MOVLW h'04'	;selecting the third hex display
MOVWF PORTA	;by using porta
CALL DELAY	;give a millisecond delay

MOVLW h'00'	;clears porta, clears selection of hex displays
MOVWF PORTA	
MOVF D4,W	;entering d4 to w register
CALL SEVNS_TABL	;changing it to the seven segment for the same
number	

MOVWF PORTB	;output this number by using portb
MOVLW h'08'	;selecting the fourth hex display
MOVWF PORTA	;by using porta
CALL DELAY	;give a millisecond delay
DECFSZ COUNTER12,F	
GOTO LOOP123	;one second delay to see four hex display at the
same time	

DECFSZ COUNTERS,F	
GOTO LOOPV	;loop for one minute

```

BSF PORTA,4
MOVLW h'09'           ;test if first digit of minutes is 9
SUBWF D1,W
BTFSS STATUS,2
GOTO ART221           ;if it is not 9
GOTO LOOP1            ;if it is 9
ART221
INCFSZ D1,F           ;increment the first digit of minutes
GOTO LOOPN99          ;go back to the one minute loop

LOOP1
MOVLW h'05'           ;test if second digit of minutes is 5
SUBWF D2,W
BTFSS STATUS,2
GOTO LOOP2            ;if it is not 5
GOTO LOOPN2           ;if it is 5

LOOP2
INCFSZ D2,F           ;increment the second digit of minutes
GOTO LOOPN55          ;go back to the one minute loop

LOOPN2
MOVLW h'03'           ;test if first digit of hours is 3
SUBWF D3,W
BTFSS STATUS,2
GOTO LOOP3            ;if it is not 3
GOTO LOOPN3           ;if it is 3

```


LOOP3

INCFSZ D3,F ;increment the first digit of hours
GOTO LOOPN66 ;go back to the one minute loop

LOOPN3

MOVLW h'02' ;test if second digit of hours is 2
SUBWF D4,W
BTFSS STATUS,2
GOTO LOOP5X ;if it is not 2
GOTO LOOPN5 ;if it is 2

LOOP5X

INCFSZ D4,F ;increment the second digit of hours
GOTO LOOPN77 ;go back to the one minute loop

LOOPN5

GOTO LOOPN88 ;go back and initializing clock

SEVNS_TABL

ADDWF PCL,F ;PCL=W(h'04')+PCL.
RETLW h'C0' ;0
RETLW h'F9' ;1
RETLW h'A4' ;2
RETLW h'B0' ;3
RETLW h'99' ;4
RETLW h'92' ;5
RETLW h'82' ;6
RETLW h'F8' ;7

RETLW h'80' ;8

RETLW h'90' ;9

RETLW h'88' ;A

RETLW h'83' ;B

RETLW h'C6' ;C

RETLW h'A1' ;D

RETLW h'86' ;E

RETLW h'8E' ;F

RETLW h'7F' ;.

DELAY

MOVLW h'04'

MOVWF COUNTER11

LOOP9X

;delay loop for a milisecond

MOVLW h'FF'

MOVWF COUNTER22

LOOP99

DECFSZ COUNTER22,F

GOTO LOOP99

DECFSZ COUNTER11,F

GOTO LOOP9X

RETURN

END

Appendix 4 – Light on-off flasher program.

```
;
; MAHMUT KISACIK 20034186 LIGHT ON OFF FLASHER USING PIC16F84A
; all working principle is same with clock except the outputs

LIST p=16f84A

#include p16f84A.inc

__CONFIG _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF

DIGIT1 EQU h'0D'
DIGIT2 EQU h'0E'
COUNT EQU h'0F'
COUNTER11 EQU h'12'
COUNTER22 EQU h'13'
DIGIT4 EQU h'14'
DIGIT3 EQU h'15'
COUNTER12 EQU h'16'
DIGIT5 EQU h'17'
DIGIT6 EQU h'18'
DIGIT7 EQU h'19'
COUNTERS EQU h'20'
D1 EQU h'21'
D2 EQU h'22'
D3 EQU h'23'
D4 EQU h'24'
D11 EQU h'25'
```

D12 EQU h'26'
D13 EQU h'27'
D14 EQU h'28'
COUNTX EQU h'29'

test EQU h'30'

D21 EQU h'31'

D22 EQU h'32'

D23 EQU h'33'

D24 EQU h'34'

INIT

BSF STATUS,5 ;Jump Bank1

CLRF TRISB ;All pins of PortB is output

CLRF TRISA ;All pins of PortA is output

BCF STATUS,5 ;Jump Bank0

CLRF PORTA ;All pins are at logic 0 level

CLRF PORTB ;All pins are at logic 0 level

MAIN

bcf PORTA,0

bcf PORTA,1 ;clearing all porta 0,1,2,3

bcf PORTA,2

bcf PORTA,3

bcf PORTB,0

bcf PORTB,1

bcf PORTB,2 ;clearing all portb 0,1,2,3,4,5,6,7


```

    bcf PORTB,3
    bcf PORTB,4
    bcf PORTB,5
    bcf PORTB,6
    bcf PORTB,7

    MOVLW h'02'
    MOVWF D4
    MOVLW h'03'
    MOVWF D3      ;initializing clock 23:59
    MOVLW h'05'
    MOVWF D2
    MOVLW h'09'
    MOVWF D1
    GOTO LOOPN99

LOOPN88
    movlw h'01'      ;test=01
    movwf test
    ;BSF PORTB,0      ;open portb 0 and 1
    ;BSF PORTB,1      ;so,open the light and feeder this for the on of program
    MOVLW h'00'      ;clearing the second digit of hours
    MOVWF D4

LOOPN77
    MOVLW h'00'      ;clearing the first digit of hours

```

MOVWF D3

LOOPN66

MOVLW h'00' ;clears the second digit of minutes

MOVWF D2

LOOPN55

MOVLW h'00' ;clears the first digit of minutes

MOVWF D1

LOOPN99

MOVLW 0X048 ;loop for a minute

MOVWF COUNTERS

LOOPV

MOVLW 0X037 ;loop for a second

MOVWF COUNTX

LOOPVV

MOVLW h'00' ;d11=00=L

MOVWF D11

MOVLW h'01' ;d12=01=I

MOVWF D12

MOVLW h'02' ;d13=02=G

MOVWF D13

MOVLW h'03' ;d14=03=H

MOVWF D14

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D11,W ;enters d11 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'01' ;selecting the first hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D12,W ;enters d12 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'02' ;selecting the second hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D13,W ;enters d13 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'04' ;selecting the third hex display

MOVWF PORTA ;by using porta

```

CALL DELAY      ;loop for a milisecond
MOVLW h'00'     ;clears porta, clears selection of hex displays
MOVWF PORTA
MOVF D14,W      ;enters d14 to w register
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTB     ;output this number by using portb
MOVLW h'08'     ; selecting the fourth hex display
MOVWF PORTA     ;by using porta
CALL DELAY      ;loop for a milisecond

```

```

DECFSZ COUNTX,F
GOTO LOOPVV     ;loop for a second
BCF PORTA,4
MOVLW 0X038
MOVWF COUNTER12 ;loop for a second

```

LOOP123

```

MOVLW h'01'     ;check if test =1
subwf test,w
BTFSS STATUS,Z
goto off        ;if it not 1 pass and goto off section
MOVLW h'04'     ;continue if it is 1, d24=04=0
MOVWF D21
MOVLW h'05'     ;d22=05=n

```


MOVWF D22

MOVLW h'05' ;d23=05=n

MOVWF D23

MOVLW h'05' ;d24=05=n

MOVWF D24

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA ;enters d21 to w register

MOVF D21,W

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'01' ; selecting the first hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D22,W ;enters d22 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'02' ; selecting the second hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

```

MOVLW h'00'      ;clears porta, clears selection of hex displays
MOVWF PORTA
MOVF D23,W       ;enters d23 to w register
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTB      ;output this number by using portb
MOVLW h'04'      ; selecting the third hex display
MOVWF PORTA      ;by using porta
CALL DELAY       ;loop for a milisecond

```

```

MOVLW h'00'      ;clears porta, clears selection of hex displays
MOVWF PORTA
MOVF D24,W       ;enters d24 to w register
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTB      ;output this number by using portb
MOVLW h'08'      ;selecting the fourth hex display
MOVWF PORTA      ;by using porta
CALL DELAY       ;loop for a milisecond
goto dong        ;pass the off section

```

```

off              ;off section starts

```

```

MOVLW h'04'      ;d21=04=O
MOVWF D21
MOVLW h'06'      ;d22=06=F
MOVWF D22
MOVLW h'06'      ;d23=06=F

```

```

MOVWF D23

MOVLW h'06'      ;d24=06=F

MOVWF D24

MOVLW h'00'      ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D21,W       ;enters d21 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB      ;output this number by using portb

MOVLW h'01'      ;selecting the first hex display

MOVWF PORTA      ;by using porta

CALL DELAY       ;loop for a milisecond


MOVLW h'00'      ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D22,W       ;enters d22 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB      ;output this number by using portb

MOVLW h'02'      ;selecting the second hex display

MOVWF PORTA      ;by using porta

CALL DELAY       ;loop for a milisecond


MOVLW h'00'      ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D23,W       ;enters d23 to w register

```

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'04' ;selecting the third hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D24,W ;enters d24 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'08' ;selecting the fourth hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

goto dong

dong

DECFSZ COUNTER12,F

GOTO LOOP123 ;loop for a second

DECFSZ COUNTERS,F

GOTO LOOPV ;loop for a minute

MOVLW h'01' ;check if the D1 is 1

SUBWF D1,W

BTFSS STATUS,2

GOTO LOOPKK ;if it is not 1
BCF PORTB,1 ;if it is 1 turn off the feeder

LOOPKK

MOVLW h'09' ;check if the first digit of minutes is 1
SUBWF D1,W
BTFSS STATUS,2
GOTO ART221 ;if it is not 1
GOTO LOOP1 ;if it is 1

ART221

INCFSZ D1,F ;increment the first digit of minutes
GOTO LOOPN99 ;go back to the one minute loop

LOOP1

MOVLW h'00' ;test=0
MOVWF test
BCF PORTB,0 ;turn off the light
MOVLW h'05' ;check if the second digit of minutes is 5
SUBWF D2,W
BTFSS STATUS,2
GOTO LOOP2 ;if it not 5
GOTO LOOPN2 ;if it is 5

LOOP2

INCFSZ D2,F ;increment the second digit of minutes
GOTO LOOPN55 ;go back to the one minute loop

LOOPN2

MOVLW h'03' ;check if the first digit of hours is 3

SUBWF D3,W

BTFSS STATUS,2

GOTO LOOP3 ;if it is not 3

GOTO LOOPN3 ;if it is 3

LOOP3

INCFSZ D3,F ;increment the first digit of hours

GOTO LOOPN66 ;go back to the one minute loop

LOOPN3

MOVLW h'02' ;check if the second digit of the hours is 2

SUBWF D4,W

BTFSS STATUS,2

GOTO LOOP5X ;if it is not 2

GOTO LOOPN5 ;if it is 2

LOOP5X

INCFSZ D4,F ;increment the second digit of hours

GOTO LOOPN77 ;go back to the one minute loop

LOOPN5

GOTO LOOPN88 ;go back to the one minute loop

SEVNS_TABL

ADDWF PCL,F ;PCL=W(h'04')+PCL.

RETLW h'C7' ;L

RETLW h'F9' ;I

RETLW h'82' ;G

RETLW h'89' ;H

RETLW h'C0' ;O

RETLW h'AB' ;n

RETLW h'8E' ;F

DELAY

MOVLW h'04'

MOVWF COUNTER11

LOOP9X

MOVLW h'FF'

MOVWF COUNTER22 ;delay loop for a milisecond

LOOP99

DECFSZ COUNTER22,F

GOTO LOOP99

DECFSZ COUNTER11,F

GOTO LOOP9X

RETURN

END

Appendix 5 – Feed on-off flasher program.

```
;
;    MAHMUT KISACIK 20034186 FEED ON-OFF FLASHER USING PIC16F84A
;

LIST p=16f84A           ;mplab command

#include p16f84A.inc      ;mplab command

__CONFIG _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF

DIGIT1 EQU h'0D'
DIGIT2 EQU h'0E'
COUNT EQU h'0F'
COUNTER11 EQU h'12'      ;initializations
COUNTER22 EQU h'13'
DIGIT4 EQU h'14'
DIGIT3 EQU h'15'
COUNTER12 EQU h'16'
DIGIT5 EQU h'17'
DIGIT6 EQU h'18'
DIGIT7 EQU h'19'
COUNTERS EQU h'20'
D1 EQU h'21'
D2 EQU h'22'
D3 EQU h'23'
D4 EQU h'24'
D11 EQU h'25'
```


D12 EQU h'26'
D13 EQU h'27'
D14 EQU h'28'
COUNTX EQU h'29'
test EQU h'30'
D21 EQU h'31'
D22 EQU h'32'
D23 EQU h'33'
D24 EQU h'34'
INIT

BSF STATUS,5 ;Jump Bank1

CLRF TRISB ;All pins of PortB is output

CLRF TRISA ;All pins of PortA is output

BCF STATUS,5 ;Jump Bank0

CLRF PORTA ;All pins are at logic 0 level

CLRF PORTB ;All pins are at logic 0 level

MAIN

bcf PORTA,0

bcf PORTA,1

bcf PORTA,2 ;clearing all ports of porta

bcf PORTA,3

bcf PORTB,0

bcf PORTB,1

bcf PORTB,2

```

bcf PORTB,3
bcf PORTB,4      ;clearing all ports of portb
bcf PORTB,5
bcf PORTB,6
bcf PORTB,7
MOVLW h'02'
MOVWF D4
MOVLW h'03'
MOVWF D3      ;initializing clock 23:59
MOVLW h'05'
MOVWF D2
MOVLW h'09'
MOVWF D1

GOTO LOOPN99

LOOPN88
    movlw h'01'      ;test=01
    movwf test
    BSF PORTB,0      ;this commands work in on-off program
    BSF PORTB,1      ;this commands work in on-off program
    MOVLW h'00'      ;clears the second digit of hours
    MOVWF D4

LOOPN77
    MOVLW h'00'      ;clears the first digit of hours

```

MOVWF D3

LOOPN66

MOVLW h'00' ;clears the second digit of minutes

MOVWF D2

LOOPN55

MOVLW h'00' ;clears the first digit of minutes

MOVWF D1

LOOPN99

MOVLW 0X048 ;loop for one minute

MOVWF COUNTERS

LOOPV

MOVLW 0X037 ;loop for one second

MOVWF COUNTX

LOOPVV

MOVLW h'00' ;d11=00=F

MOVWF D11

MOVLW h'01' ;d12=01=E

MOVWF D12

MOVLW h'02' ;d13=02=E

MOVWF D13

MOVLW h'03' ;d14=03=d

MOVWF D14

```

MOVLW h'00'      ;clears porta, clears selection of hex displays
MOVWF PORTA
MOVF D11,W       ;enters d11 to w register
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTB      ;output this number by using portb
MOVLW h'01'      ;selecting the first hex display
MOVWF PORTA      ;by using porta
CALL DELAY       ;loop for a milisecond

```

```

MOVLW h'00'      ;clears porta, clears selection of hex displays
MOVWF PORTA
MOVF D12,W       ;enters d12 to w register
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTB      ;output this number by using portb
MOVLW h'02'      ;selecting the first hex display
MOVWF PORTA      ;by using porta
CALL DELAY       ;loop for a milisecond

```

```

MOVLW h'00'      ;clears porta, clears selection of hex displays
MOVWF PORTA
MOVF D13,W       ;enters d13 to w register
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTB      ;output this number by using portb
MOVLW h'04'      ;selecting the third hex display

```



```

MOVWF PORTA      ;by using porta
CALL DELAY        ;loop for a milisecond

MOV LW h'00'      ;clears porta, clears selection of hex displays
MOVWF PORTA
MOV F D14,W       ;enters d14 to w register
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTB       ;output this number by using portb
MOV LW h'08'      ;selecting the fourth hex display
MOVWF PORTA       ;by using porta
CALL DELAY        ;loop for a milisecond

DECFSZ COUNTX,F
GOTO LOOPVV       ;loop for a second
BCF PORTA,4
MOV LW 0X038
MOVWF COUNTER12   ;loop for a second

LOOP123
MOV LW h'01'      ;test=01
subwf test,w
BTFSS STATUS,Z
goto off          ;if test is not 1 pass to the off loop
MOV LW h'04'      ;d21=04=0

```

MOVWF D21

MOVLW h'05' ;d22=05=F

MOVWF D22

MOVLW h'05' ;d23=05=F

MOVWF D23

MOVLW h'05' ;d24=05=F

MOVWF D24

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D21,W ;enters d21 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'01' ;selecting the first hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D22,W ;enters d22 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'02' ;selecting the second hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D23,W ;enters d23 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'04' ;selecting the third hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D24,W ;enters d24 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'08' ;selecting the fourth hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

goto dong

off

MOVLW h'04'

MOVWF D21

MOVLW h'06'

MOVWF D22

MOVLW h'06'

MOVWF D23

MOVLW h'06'

MOVWF D24

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D21,W ;enters d21 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'01' ;selecting the first hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D22,W ;enters d22 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'02' ;selecting the secoond hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D23,W ;enters d23 to w register

CALL SEVNS_TABL ;changing it to the seven segment for the same

number

MOVWF PORTB ;output this number by using portb

MOVLW h'04' ;selecting the third hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears porta, clears selection of hex displays

MOVWF PORTA

MOVF D24,W ;enters d24 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'08' ;selecting the fourth hex display

MOVWF PORTA ;by using porta

CALL DELAY ;loop for a milisecond

goto dong

dong

DECFSZ COUNTER12,F

GOTO LOOP123 ;loop for a second

DECFSZ COUNTERS,F

GOTO LOOPV ;loop for a minute

MOVLW h'01' ;check if d1 is 1

SUBWF D1,W

BTFSS STATUS,2

GOTO LOOPKK ;if it is not 1

MOVLW h'00' ;if it is 1

MOVWF test ;make test=0

BCF PORTB,1 ;turn off the feed

LOOPKK

MOVLW h'09' ;check if the first digit of minutes is 9

SUBWF D1,W

BTFSS STATUS,2

GOTO ART221 ;if it is not 9

GOTO LOOP1 ;if it is 9

ART221

INCFD1,F ;increment the first digit of minutes

GOTO LOOPN99 ;go back to the one minute loop

LOOP1

BCF PORTB,0 ;turn off the light

MOVLW h'05' ;check if the second digit of the minutes is 5

SUBWF D2,W

BTFSS STATUS,2

GOTO LOOP2 ;if it is not 5

GOTO LOOPN2 ;if it is 5

LOOP2

INCFSZ D2,F ;increment the second digit of minutes
GOTO LOOPN55 ;go back to the one minute loop

LOOPN2

MOVLW h'03' ;check if the first digit of the hours is 3
SUBWF D3,W
BTFSS STATUS,2
GOTO LOOP3 ;if it not 3
GOTO LOOPN3 ;if it is 3

LOOP3

INCFSZ D3,F ;increment the first digit of the hours
GOTO LOOPN66 ;go back to the one minute loop

LOOPN3

MOVLW h'02' ;check if the second digit of the hours is 2
SUBWF D4,W
BTFSS STATUS,2
GOTO LOOP5X ;if it is not 2
GOTO LOOPN5 ;if it is 2

LOOP5X

INCFSZ D4,F ;increment the second digit of the hours
GOTO LOOPN77 ;go back to the one minute loop

LOOPN5

GOTO LOOPN88 ;go back to the one minute loop

SEVNS_TABL

ADDWF PCL,F ;PCL=W(h'04')+PCL.

RETLW h'8E' ;F

RETLW h'86' ;E

RETLW h'86' ;E

RETLW h'A1' ;d

RETLW h'C0' ;O

RETLW h'AB' ;n

RETLW h'8E' ;F

DELAY

MOVLW h'04'

MOVWF COUNTER11

LOOP9X ;delay a milisecond

MOVLW h'FF'

MOVWF COUNTER22

LOOP99

DECFSZ COUNTER22,F

GOTO LOOP99

DECFSZ COUNTER11,F

GOTO LOOP9X

RETURN

END

Appendix 6 – Light and feeding On-off program.

```
;
;    MAHMUT KISACIK 20034186 LIGHT AND FEEDING ON-OFF
;MECHANISM USING PIC16F84A
;
LIST p=16f84A      ;mplab command
#include p16f84A.inc ;mplab command
__CONFIG _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF ;crystal osc

DIGIT1 EQU h'0D'
DIGIT2 EQU h'0E'
COUNT EQU h'0F'
COUNTER11 EQU h'12' ;initializations
COUNTER22 EQU h'13'
DIGIT4 EQU h'14'
DIGIT3 EQU h'15'
COUNTER12 EQU h'16'
DIGIT5 EQU h'17'
DIGIT6 EQU h'18'
DIGIT7 EQU h'19'
COUNTERS EQU h'20'
D1 EQU h'21'
D2 EQU h'22'
D3 EQU h'23'
D4 EQU h'24'
```

COUNTX EQU h'25'

INIT

```
BSF STATUS,5      ;Jump Bank1
CLRF TRISB        ;All pins of PortB is output
CLRF TRISA        ;All pins of PortA is output
BCF STATUS,5      ;Jump Bank0
CLRF PORTA        ;All pins are at logic 0 level
CLRF PORTB        ;All pins are at logic 0 level
```

MAIN

```
bcf PORTA,0
bcf PORTA,1      ;clear all ports of porta
bcf PORTA,2
bcf PORTA,3
bcf PORTB,0
bcf PORTB,1
bcf PORTB,2
bcf PORTB,3
bcf PORTB,4      ;clear all ports of portb
bcf PORTB,5
bcf PORTB,6
bcf PORTB,7
MOVLW h'02'
MOVWF D4
MOVLW h'03'
```

MOVWF D3 ;initializing clock=23:59

MOVLW h'05'

MOVWF D2

MOVLW h'09'

MOVWF D1

GOTO LOOPN99

LOOPN88

BSF PORTB,0 ;turn on feed

BSF PORTB,1 ;turn on light

MOVLW h'00' ;clears the first digit of hours

MOVWF D4

LOOPN77

MOVLW h'00' ;clears the second digit of hours

MOVWF D3

LOOPN66

MOVLW h'00' ;clears the first digit of minutes

MOVWF D2

LOOPN55

MOVLW h'00' ;clears the second digit of minutes

MOVWF D1

LOOPN99

MOVLW 0X048 ;loop for one minute

MOVWF COUNTERS

LOOPV

MOVLW 0X037

MOVWF COUNTX

LOOPVV

CALL DELAY ;make some delay to work synchronously

CALL DELAY ;to the other programs

CALL DELAY

CALL DELAY

DECFSZ COUNTX,F

GOTO LOOPVV

MOVLW 0X038

MOVWF COUNTER12

LOOP123

CALL DELAY

CALL DELAY ;same that above

CALL DELAY

CALL DELAY

DECFSZ COUNTER12,F

GOTO LOOP123

DECFSZ COUNTERS,F

GOTO LOOPV

MOVLW h'01' ;check if the first digit of the clock is 1

SUBWF D1,W

BTSS STATUS,2

GOTO LOOPKK ;if it not 1


```

    BCF PORTB,1      ;if it is 1,turn off the feed

LOOPKK
    MOVLW h'09'      ;check if the first digit of minutes is 9
    SUBWF D1,W
    BTFSS STATUS,2
    GOTO ART221      ;if it is not 9
    GOTO LOOP1       ;if it is 9

ART221
    INCFSZ D1,F      ;increment the first digit of the minutes
    GOTO LOOPN99     ;go back to the one minute loop

LOOP1
    BCF PORTB,0      ;turn off the light
    MOVLW h'05'      ;check if the second digit of the minutes is 5
    SUBWF D2,W
    BTFSS STATUS,2
    GOTO LOOP2       ;if it not 5
    GOTO LOOPN2      ;if it is 5

LOOP2
    INCFSZ D2,F      ;increment the second digit of the minutes
    GOTO LOOPN55     ;go back to the one minute loop

LOOPN2
    MOVLW h'03'      ;check if the first digit of the hours is 3
    SUBWF D3,W
    BTFSS STATUS,2

```

GOTO LOOP3 ; if it is not 3

GOTO LOOPN3 ; if it is 3

LOOP3

INCF SZ D3,F ; increment the first digit of the hours

GOTO LOOPN66 ; go back to the one minute loop

LOOPN3

MOVLW h'02' ; check if the second digit of the hours is 2

SUBWF D4,W

BTFSS STATUS,2

GOTO LOOP5X ; if it not 2

GOTO LOOPN5 ; if it is 2

LOOP5X

INCF SZ D4,F ; increment the second digit of the hours

GOTO LOOPN77 ; go back to the one minute loop

LOOPN5

GOTO LOOPN88 ; go back to the one minute loop

SEVNS_TABL

ADDWF PCL,F ;PCL=W(h'04')+PCL.

RETLW h'C0' ;0

RETLW h'F9' ;1

RETLW h'A4' ;2

RETLW h'B0' ;3

RETLW h'99' ;4

RETLW h'92' ;5

RETLW h'82' ;6

RETLW h'F8' ;7

RETLW h'80' ;8

RETLW h'90' ;9

RETLW h'88' ;A

RETLW h'83' ;B

RETLW h'C6' ;C

RETLW h'A1' ;D

RETLW h'86' ;E

RETLW h'8E' ;F

RETLW h'7F' ;.

DELAY ;delay loop for a milisecond

MOVLW h'04'

MOVWF COUNTER11

LOOP9X

MOVLW h'FF'

MOVWF COUNTER22

LOOP99

DECFSZ COUNTER22,F

GOTO LOOP99

DECFSZ COUNTER11,F

GOTO LOOP9X

RETURN

END

Appendix 7 – Digital thermometer and heater control program.

```
;  
;    MAHMUT KISACIK 20034186 DIGITAL THERMOMETER & HEATER  
CONTROL USING ;PIC16F877  
;  
LIST p=16f877      ;mplab command  
#include p16f877.inc    ;mplab command  
__CONFIG _XT_OSC & _WDT_OFF & _PWRTE_ON & _CP_OFF ;crystal osc  
COUNTER11 EQU h'0F'  
COUNTER22 EQU h'10'  
COUNTER12 EQU h'11'  
V7    EQU h'12'      ;initializations  
D1    EQU h'13'  
D2    EQU h'14'  
D3    EQU h'15'  
COUNTX EQU h'16'  
Z1    EQU h'17'  
Z2    EQU h'18'  
RESULTL EQU h'19'  
GEC    EQU h'20'  
GECX   EQU h'21'  
G1     EQU h'22'  
G2     EQU h'23'  
G3     EQU h'24'
```


G4 EQU h'25'
G5 EQU h'26'
G6 EQU h'27'
V8 EQU h'28'
V9 EQU h'29'
V10 EQU h'30'

INIT

BSF STATUS,5 ;Jump Bank1

movlw b'00000001' ; input for analogue temp sensor

movwf TRISA

CLRF TRISB ;All pins of PortB is output

CLRF TRISC ;All pins of Portc is output

CLRF TRISD ;All pins of Portd is output

BCF STATUS,5 ;Jump Bank0

CLRF PORTC ;All pins are at logic 0 level

CLRF PORTD ;All pins are at logic 0 level

CLRF PORTB ;All pins are at logic 0 level

CLRF PORTA ;All pins are at logic 0 level

MAIN

BSF STATUS,RP0 ;jump bank 1

MOVLW 0x01 ;porta 0 is analog output

MOVWF TRISA

MOVLW 0x8E ;ref=vdd ,adfm=1

MOVWF ADCON1

```

BCF STATUS,RP0    ;jump bank 0

MOVLW 0X41        ;fosc/8 because of 4mhz crystal,A/D is active now

MOVWF ADCON0

CEVIRT

CLRF TRISD

CLRF TRISC

BSF ADCON0,GO     ;start to A/D converter

MOVLW h'0A'

MOVWF GECX

ART                ;small delay loop for getting temp correctly

    DECFSZ GECX,F

    GOTO ART

BAKLE

    BTFSC ADCON0,2 ;is converting finished or not

    GOTO BAKLE     ;if it is not

    BSF STATUS,5   ;finished, jump to bank1

    MOVF ADRESL,w  ;read low 8 bit

    BCF STATUS,5   ;jump bank0

    MOVWF RESULTL  ; keep the bits in resultl

    BCF STATUS,C

    RRF RESULTL,1  ;divide the number by two

    INCF RESULTL,f ;get the correct temperature by six times adding

    INCF RESULTL,f

    INCF RESULTL,f

```

INCF RESULTL,f

INCF RESULTL,f

INCF RESULTL,f

CLRF D1

CLRF D2

CLRF D3

MOVF RESULTL,w ;enters resultl to w register

MOVWF GEC ;enters w to gec

MOVLW .100

B1 ;if the number is 037 degree d1=0 d2=3 d3=7

INCF D1,f ;increment the first character

SUBWF GEC,f

BTFSC STATUS,C

GOTO B1

DECF D1,f ;decrement the first character

ADDWF GEC,f

MOVLW .10

B2

INCF D2,f ;increment the second character

SUBWF GEC,f

BTFSC STATUS,C

GOTO B2

DECF D2,f ;decrement the second character

ADDWF GEC,f

```

MOVF GEC,w
MOVWF D3
MOVLW h'0A'      ;z1=0A=0 for celsius
MOVWF Z1
MOVLW h'0B'      ;z2=0B=C
MOVWF Z2
MOVLW h'11'      ;g6=11=F
MOVWF G6
BCF PORTA,5;      ;turn off the heater
MOVLW h'01'      ;check if the second digit is 01
SUBWF D2,W
BTFSS STATUS,2
GOTO ART22X      ;if it is not 1
GOTO LOOP2      ;if it is 1

ART22X
MOVLW h'02'      ;check if the second digit is 2
SUBWF D2,W
BTFSS STATUS,2
GOTO ART221      ;if it is not 2
GOTO LOOP1      ;if it is 2

ART221
MOVLW 0X060      ;loop for one second
MOVWF COUNTX

LOOPVV

```



```

MOVLW h'00'      ;clears portc, clears selection of hex displays
MOVWF PORTC
MOVLW h'0B'      ;enters d24 to w register
MOVWF V7
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTD      ;output this number by using portd
MOVLW h'01'      ;selecting the first hex display
MOVWF PORTC      ;by using portc
CALL DELAY       ;loop for a milisecond
MOVLW h'00'      ;clears portc, clears selection of hex displays
MOVWF PORTC
MOVLW h'0D'
MOVWF V8          ;enters v8 to w register
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTD      ;output this number by using portd
MOVLW h'02'      ;selecting the second hex display
MOVWF PORTC      ;by using portc
CALL DELAY       ;loop for a milisecond
MOVLW h'00'      ;clears portc, clears selection of hex displays
MOVWF PORTC
MOVLW h'12'      ;enters v9 to w register
MOVWF V9
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTD      ;output this number by using portd

```

```

MOVLW h'04'      ;selecting the third hex display
MOVWF PORTC      ;by using portc
CALL DELAY       ;loop for a milisecond
MOVLW h'00'      ;clears portc, clears selection of hex displays
MOVWF PORTC
MOVLW h'05'      ;enters v10 to w register
MOVWF V10
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTD      ;output this number by using portd
MOVLW h'08'      ;selecting the fourth hex display
MOVWF PORTC      ;by using portc
CALL DELAY       ;loop for a milisecond
MOVLW h'00'      ;clears portc, clears selection of hex displays
MOVWF PORTC
MOVLW h'0C'      ;enters g1 to w register
MOVWF G1
CALL SEVNS_TABL;changing it to the seven segment for the same number
MOVWF PORTB      ;output this number by using portb
MOVLW h'10'      ;selecting the fifth hex display
MOVWF PORTC      ;by using portc
CALL DELAY       ;loop for a milisecond
MOVLW h'00'      ;clears portc, clears selection of hex displays
MOVWF PORTC
MOVLW h'0D'      ;enters g2 to w register

```

MOVWF G2

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'20' ;selecting the sixth hex display

MOVWF PORTC ;by using portc

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears portc, clears selection of hex displays

MOVWF PORTC

MOVLW h'0E' ;enters g3 to w register

MOVWF G3

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'40' ;selecting the seventh hex display

MOVWF PORTC ;by using portc

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears portc, clears selection of hex displays

MOVWF PORTC

MOVLW h'0F' ;enters g4 to w register

MOVWF G4

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB ;output this number by using portb

MOVLW h'80' ;selecting the eighth hex display

MOVWF PORTC ;by using portc

CALL DELAY ;loop for a milisecond

DECFSZ COUNTX,F

GOTO LOOPVV ;loop for a second

MOVLW 0X062

MOVWF COUNTER12

LOOP123

MOVLW h'00' ;clears portc, clears selection of hex displays

MOVWF PORTC

MOVF D2,W ;enters d2 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTD ;output this number by using portd

MOVLW h'01' ;selecting the first hex display

MOVWF PORTC ;by using portc

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears portc, clears selection of hex displays

MOVWF PORTC

MOVF D3,W ;enters d3 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTD ;output this number by using portd

MOVLW h'02' ;selecting the second hex display

MOVWF PORTC ;by using portc

CALL DELAY ;loop for a milisecond

MOVLW h'00' ;clears portc, clears selection of hex displays

MOVWF PORTC

MOVF Z1,W ;enters z1 to w register


```

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTD    ;output this number by using portd

MOVLW h'04'    ;selecting the third hex display

MOVWF PORTC    ;by using portc

CALL DELAY     ;loop for a milisecond

MOVLW h'00'    ;clears portc, clears selection of hex displays

MOVWF PORTC

MOVF Z2,W      ;enters z2 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTD    ;output this number by using portd

MOVLW h'08'    ;selecting the fourth hex display

MOVWF PORTC    ;by using portc

CALL DELAY     ;loop for a milisecond

MOVLW h'00'    ;clears portc, clears selection of hex displays

MOVWF PORTC

MOVLW h'00'

MOVWF G5       ;enters g5 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB    ;output this number by using portb

MOVLW h'10'    ;selecting the fifth hex display

MOVWF PORTC    ;by using portc

CALL DELAY     ;loop for a milisecond

MOVLW h'00'    ;clears portc, clears selection of hex displays

MOVWF PORTC

```

```

MOVWF G6,W      ;enters g6 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB     ;output this number by using portb

MOVLW h'20'     ;selecting the sixth hex display

MOVWF PORTC     ;by using portc

CALL DELAY      ;loop for a milisecond

MOVLW h'00'     ;clears portc, clears selection of hex displays

MOVWF PORTC

MOVWF G6,W      ;enters g6 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB     ;output this number by using portb

MOVLW h'40'     ;selecting the seventh hex display

MOVWF PORTC     ;by using portc

CALL DELAY      ;loop for a milisecond

MOVLW h'00'     ;clears portc, clears selection of hex displays

MOVWF PORTC

MOVWF G6,W      ;enters g6 to w register

CALL SEVNS_TABL;changing it to the seven segment for the same number

MOVWF PORTB     ;output this number by using portb

MOVLW h'80'     ;selecting the eighth hex display

MOVWF PORTC     ;by using portc

CALL DELAY      ;loop for a milisecond

DECFSZ COUNTER12,F

GOTO LOOP123    ;loop for a second

```

GOTO MAIN ;go back and take the new temp

LOOP1

MOVLW h'09' ;check if the third digit is 9

SUBWF D3,W

BTFSS STATUS,2

GOTO LOOP2 ;if it is not 9

GOTO ART221 ;if it is 9

LOOP2

BSF PORTA,5 ;turn on heater

MOVLW h'10'

MOVWF G6 ;G6=10

GOTO ART221

SEVNS_TABL

addwf PCL,f ; W + PCL -> PCL

RETLW h'C0' ;0

RETLW h'F9' ;1

RETLW h'A4' ;2

RETLW h'B0' ;3

RETLW h'99' ;4

RETLW h'92' ;5

RETLW h'82' ;6

RETLW h'F8' ;7

RETLW h'80' ;8

RETLW h'90' ;9

```

RETLW h'9C' ;A o
RETLW H'C6' ;B C
RETLW H'89' ;c H
RETLW H'86' ;D E
RETLW H'88' ;E A
RETLW H'8F' ;F I-
RETLW H'AB' ;10 n
RETLW H'8E' ;11 F
RETLW H'C7' ;12 L

DELAY ;delay for a milisecond

    MOVLW h'04'
    MOVWF COUNTER11

LOOP9X
    MOVLW h'FF'
    MOVWF COUNTER22

LOOP99
    DECFSZ COUNTER22,F
    GOTO LOOP99
    DECFSZ COUNTER11,F
    GOTO LOOP9X
    RETURN
    END

```