

MICROCONTROLLER BASED CONTROLLER FOR HEATLESS DESICCANT DRYER

Project Report

Submitted by

B.Satheesh Kumar

M.Shankar

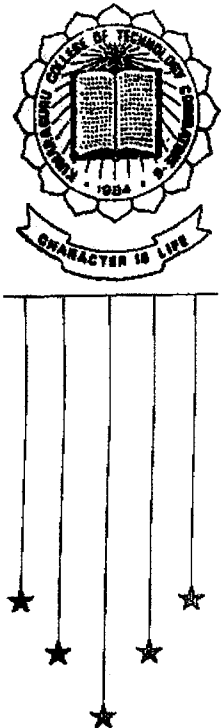
R.Anu Radha

P.Sandeep

Under the guidance of

Prof.K.RamPrakash,M.E

In partial fulfillment of the requirements
for the award of the degree of
Bachelor of Engineering in
Electronics and Communication Engineering
of Bharathiar University, Coimbatore



2001-2002

Department of Electronics and Communication Engineering
KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE – 641 006

P-1384

Kumaraguru College Of Technology

Coimbatore – 641006

Department of Electronics and Communication Engineering

Certificate

This is to certify that this project entitled

Microcontroller Based Controller for Heatless Desiccant Dryer

has been submitted by

Mr. B. SATHEESH KUMAR, Mr. M. SHANKAR, Ms. R. ANURADHA, Mr. P. SANDHEEP

In partial fulfillment of the requirements for the award of the degree of

Bachelor of Engineering in Electronics and Communication Engineering Branch

of Bharathiar University, Coimbatore – 641 046 during the

Academic year 2001 –2002



(Guide) 13/3/02



(Head of the Department)

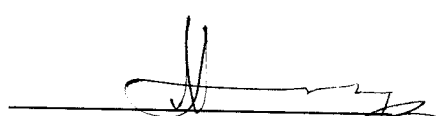
Certified that the candidate was examined by us in the Project Work

Viva-Voce Examination held on 19.03-2002

University Register Number 982700226, 982700231, 982700187, 982700221.

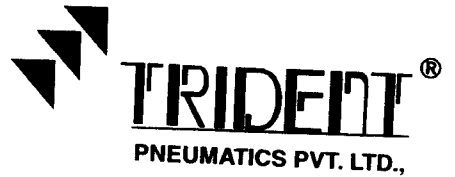


(Internal Examiner)



(External Examiner)

K.N.G. Pudur Road,
umpalayam P.O.
Coimbatore - 641 108, India.
: +91 422 400492
: +91 422 401376
: sales@tridentpneumatics.com
: www.tridentpneumatics.com



TO WHOMSOEVER IT MAY CONCERN.

This is to certify that the following final year BE Electronics & Communication Engineering students of Kumaraguru College of Technology, Coimbatore have completed the project in our organization:

1. R. Anuradha
2. P. Sandeep
3. B. Satheesh Kumar
4. M. Shankar

Project Title	Micro Controller based Controller for Heatless Desiccant Dryers,
Project period	July 2001 to March 2002
Department	Research & Development (Electronics)

During this project, their attendance, conduct and behaviour were found to be good.

Trident Pneumatics Pvt Ltd

A handwritten signature in black ink that reads "R. Sivakumar".

R. Sivakumar
R&D Engineer

Coimbatore
8th March 2002

ACNOWLEDGEMENT:

We express our deep sense of gratitude to Dr.K.K.Padmanabhan, Ph.D., Principal, Kumaraguru College of Technology, Coimbatore for providing permission to carry out this project work.

With a deep sense of gratitude and regards, we acknowledge with great pleasure the guidance, encouragement and support extended by **Prof.Muthuraman Ramasamy M.E., FIE., FIETE., M.I.E.E.E (U.S.A), M.I.S.T.E, M.I.E, M.B.M.E.S.I, C.ENG. (I), Professor and Head of the Department of Electronics and Communication Engineering** and express our thanks for his valuable time, guidance and kind encouragement during the course of this project work.

It is our duty to express our thanks to our internal guide, Prof.K.Ramprakash M.E., Asst.Proffessor, Department of Electronics and Communication Engineering for his valuable guidance, keen suggestions, innovative ideas, helpful criticisms and kind encouragement during all the phases of this project work.

We will be failing in our duty if we don't thank Mr.K.S.Natarajan, Managing Director and Mr.R.SivaKumar, Trident Pneumatics Pvt. Ltd. for providing us with an opportunity to do a project in their company. We express our deep sense of gratitude for their proper guidance and constant encouragement during the entire course of this project work.

We express our sincere thanks to all the staff members of the Electronics & Communication Engineering Department, especially Mr.G.C.Thiagarajan M.E., Lecturer, Department of Electronics and Communication Engineering who gave us valuable guidance and helped us in overcoming the barriers and problems we faced in our project.

We also thank all the staff members of Trident Pneumatics Pvt. Ltd., who helped us for the successful completion of the project.

Finally we express our deep sense of gratitude to our parents and friends for their valuable help and consideration towards us.

Synopsis

SYNOPSIS:

Microcontroller based controller for heatless Desiccant Dryers is a system designed to control and monitor the various functions of the heatless desiccant air dryer which is an industrial device used to remove the moisture from the air and output the clean, dry and dust-free air.

The advantages of a microcontroller based control system are versatility, less space requirements, less power requirements and a higher speed of operation, reduced instruction set which leads to a much better response time. This control system has got microchip's PIC16F877 as the microcontroller. It is a 40-pin microcontroller, which is very versatile and occupies very less space and consumes very less power making it an ideal choice for resource critical applications like these.

Our project involves the software part. The software is programmed using the 'C' language developed specifically for the Microchip range of PICs, but which adheres to the ANSI C specification. The software has been designed using the minimum possible code and memory space so that portability to different IC's won't be hampered much due to the lack of program memory or data memory in those IC's and also for making future upgrades possible.

CONTENTS:

S.NO.	Chapter	Page No.
1.	Introduction	1
2.	Project description	
	2.1 Principle of dryer operation	3
	2.2 Requirements	5
	2.3 Strategy	8
3.	PIC16F877 Microcontroller – An Overview	
	3.1 Microchip family – An introduction	10
	3.2 The PIC16F877	11
	3.2.1 Core features	11
	3.2.2 Peripheral features	12
	3.3 General description	12
	3.4 Memory Organisation	13
	3.4.1 Program memory Organisation	13
	3.4.2 Data memory Organisation	13
	3.4.3 EEPROM Memory	14
	3.5 Special Function registers	15
	3.6 EEPROM registers	16

	3.7 I/O Ports	17
	3.8 Timer Module	18
	3.9 Interrupts	19
	3.10 Pin Diagram	20
4.	Implementation	
	4.1 Hardware description	21
	4.2 Circuit Diagram	22
	4.3 Software Description	23
	4.3.1 'C' – An Overview	23
	4.3.2 Development mode	23
	4.3.3 Program modules	24
	4.3.3.1 Valve Control	25
	4.3.3.2 KeyScan & Menu entry	29
	4.3.3.3 Display Module	33
	4.3.3.4 Alarm Module	36
	4.3.4 Program Description	38
5.	Conclusion	42
6.	Bibliography	43
7.	Source Code	44
8.	Appendix	88

A control system is one, which is designed to provide proper control and ensure correct sequence of operations for the system or group of devices it has to control. The control system may be completely mechanical or electrical or electronic in nature or a combination of two or more of the above mentioned types. This depends upon various factors. This project is essentially an embedded control system. An embedded control system refers to one that uses an embedded controller as its main control system. Before going into further details a brief introduction of the nature of the project is given as follows.

This control system has been programmed to control and monitor the operations of heatless desiccant dryer, which is an industrial device, designed to provide clean, dry and compressed air. The controller has to properly initialize the dryer on startup and then control the opening and closing of the valves, which allow air into the dryer towers. This has to be done on a fixed time basis. Additionally a LCD display unit has to display the status of the towers of the dryer. The controller apart from driving the LCD should also provide some menu functions which enables the user to enter certain parameters, that are used to control the operation of the dryer.

The controller should store these parameters entered by the user. The controller has to keep track of the running hours of the dryer and display certain alarm messages after the specified duration.

Embedded controllers are specially designed microprocessors, which have got additional functions included on the chip with the basic CPU, and they have special instructions for dealing with individual bits in a word. These additional functions, generally speaking, refers to memory (program memory as well as data memory), timers, comparators, ADCs and other essential and commonly used processing and detector elements which are included on the chip along with the basic CPU.

This is in direct contrast compared to the microprocessors which do not include most of the above mentioned additional functions along with the basic CPU. The microprocessor chip is entirely devoted to the control unit and ALU. But comparing the microprocessor and microcontroller along these lines is not entirely justified, simply because these two were designed and built for entirely different purposes.

2.1. PRINCIPLE OF DRYER OPERATION:**Pre-filtration:**

In the moisture laden compressed air passes through the pre-filter1, here moisture load is reduced through coalescence. Condensation is removed through the drain valve ADV1. At the oil filter (2) oil vapors are removed completely, small level of water condensation is purged through the valve ADV2. Compressed air then passed through a 3-way valve into tower.

Drying:

The two towers are filled with activated alumina, which acts as the desiccant. When air passes through tower1, which consists of DRY desiccant, it gets completely dried, and passes through check valve and after filter3. At the after filter, desiccant fines are removed. Therefore cool dry compressed air passes out at outlet.

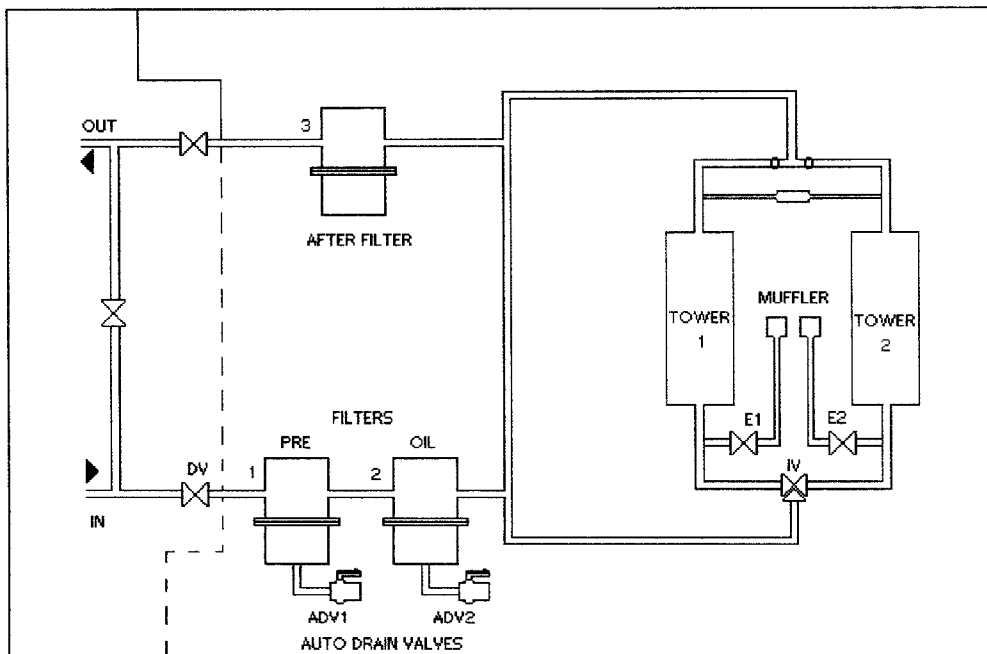
Regeneration:

The process of Regeneration takes place in two stages namely,

- De-pressurisation
- By passing dry air

Tower2 consists of fully moist desiccant at pressure. This is suddenly depressurised by opening the purge valve. Water molecules weep out of the desiccant and appear on the surface. Super dry purge air passes through the regeneration nozzle and the desiccant bed there by completely carrying away the water molecules. Tower2 gets regenerated and is ready for the next drying cycle.

HEATLESS DESICCANT DRYER



2.2. REQUIREMENT:

The project requires programming the controller to control the valves, displaying the tower status, enable the user to enter some parameters and to display some alarm messages at certain time instants as desired by the user.

Initially the controller should scroll an advertisement for one minute, and then it should display the status of the towers continuously until the user presses some key to enter the menu functions described below. The first line should display the drying status of the corresponding tower and the second line should display the regeneration status of the

<p>T1 DRY</p> <p>T2 REGEN</p>

other tower as shown.

To enter the menu functions, some form of password restriction has to be made. Only after the password is entered by the user and it is found to be correct, access to the menu functions are allowed. If the password is incorrect then entry should be denied and the message “ACCESS DENIED” has to be displayed to the user.

The main menu contains 3 divisions as follows:

1. **FACTORY SETTING**
2. **FIELD SETTING**
3. **MAINTENANCE**

The submenus for each of these menu functions are as follows. The submenus have parameters and the user should be enabled to enter numerical values for these parameters.

Inside factory setting,

CYCLE TIME	:	-----min
FLOW	:	-----cfm
MODEL	:	-----
DRAIN	:	-----sec
PRESSURE	:	-----bar
ENABLE	:	Y/N

Similarly in field setting the parameters to be displayed are phone, fax no., Pressure, flow and Pr.drop.

PHONE NO.	:	-----
FAX NO	:	-----
PRESSURE	:	-----
FLOW	:	Y/N
PR. DROP	:	Y/N

The parameters in the maintenance function are change filter, change desiccant and change seal. Here also provision is made for data entry.

CHANGE FILTER :	-----hrs
CHANGE DESSIC :	-----hrs
CHANGE SEAL :	-----hrs

The parameters in the maintenance function are actually alarm values. These values are entered by the user in hours and after the specified hours has elapsed the controller should display alarm messages.

2.3. STRATEGY

The controller used for this purpose is pic16f877 and to display the different functions the LCD (LM16200) is used. Our project is concerned with programming the microcontroller to control the opening and closing of the valves and thereby controlling the flow of air through each tower.

Drain valve, Inlet valve, valve E1 and valve E2 are the valves to be controlled.

After every five minutes the desiccant in the towers will get saturated. The desiccant has to be regenerated after this time period. Hence each tower has to be operated in drying mode for 5 minutes and the desiccant has to be regenerated in the next 5 minutes. Hence in each ten-minute cycle, when tower1 is in drying condition the tower2 should be in regeneration condition.

Airflow to both the towers is through the inlet valve, and hence the inlet valve should remain open for the whole ten-minute cycle. The valve E2 that corresponds to tower T2 should remain open during the first phase of its regeneration period. During the second phase which is called as the repressurisation mode, valve E2 is closed. The same holds for the valve E1 during the regeneration phase of tower T1. This repeats for every ten-minute cycle.

The status of the towers and the status of the valves are tabulated below for each timing interval.

STATUS OF THE TOWERS

Time(minutes)	Tower1	Tower2
0-4	Drying	Regeneration
4-5	Drying	Repressurisation
5-9	Regeneration	Drying
9-10	Repressurisation	Drying

(Table 1.1)

STATUS OF THE VALVES

Time(minutes)	Inlet valve	Valve E1	Valve E2
0-4	Open	Close	Open
4-5	Open	Close	Close
5-9	Open	Open	Close
9-10	Open	Close	Close

(Table 1.1)

3.1. MICROCHIP FAMILY – AN INTRODUCTION:

Microchip, the manufacturer of the PICMicro™ range of microcontrollers is one of the world's leading manufacturers of RISC microcontrollers. Their RISC MCU architecture has become a world wide standard, with half a billion PIC micro devices and 100,000 development systems shipped since 1990. Microchip offers a wide range of enhanced flash, EEPROM and ROM products and continues to expand its product breadth.

Short lead times, flexible technology and a high performance RISC architecture which outperforms CISC based competitors provide key design advantages for embedded system applications. All PIC micro MCUs employ a modified RISC architecture and powerful instruction set that dramatically reduce development cycle time and cost by combining RISC features with a modified Harvard dual bus architecture, microchip's fast and flexible eight MIPS PICmicro core is the most popular architecture for new MCU designs. Seamless migration between product families and a simple instruction set make PIC micro MCUs the logical choice for designs requiring flexibility and performance.

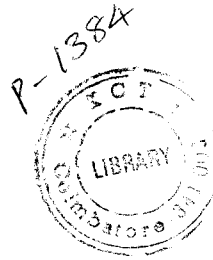
3.2. THE PIC16F877:

The microcontroller that was used in this application is Microchip's PIC16F877, a high performance RISC CPU.

FEATURES:

3.2.1. Core Features:

- High performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
DC - 200 ns instruction cycle
- Up to 368 x 8 bytes of Data Memory (RAM)
- Up to 256 x 8 bytes of EEPROM Data Memory
- Interrupt capability (up to 14 sources)
- Watchdog Timer (WDT) with its own on-chip RC
- Oscillator for reliable operation
- Wide operating voltage range: 2.0V to 5.5V
- Programmable code protection
- Power saving SLEEP mode



3.2.2. Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler

3.3. GENERAL DESCRIPTION:

The PIC16F87X microcontroller (MCU) family provides a migration path from OTP to FLASH in 28 to 44 – pin packages with a wide range of peripheral integration options. This family features a 14 – bit instruction, 5 to 8 channels of 10-bit Analog-to-Digital Converters, interrupt handling capability, various serial interface capabilities, Capture/Compare/PWM, Brown-out Detection and an 8-level deep stack.

The PIC16F87X family provides performance and versatility to meet the most demanding requirements of today's cost sensitive analog designs. Plus, with FLASH program memory, PIC16F87X devices can be reprogrammed over the entire operating voltage range. The PIC16F87x family is ideally suited for applications ranging from smart card readers to POS terminals and high-speed applications.

3.4. MEMORY ORGANISATION:

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

3.4.1. Program Memory Organisation:

The PIC16F877 devices have a 13-bit program counter capable of addressing an 8K x 14-program memory space. The PIC16F877 devices have 8K x 14 words of FLASH program memory. Accessing a location above the physically implemented address will cause a wraparound. The RESET vector is at 0000h and the interrupt vector is at 0004h.

3.4.2. Data Memory Organisation:

The data memory is partitioned into multiple banks, which contain the general Purpose Registers, and the Special Function Registers. Bits RP1 (STATUS<6>) and RP0 (STATUS<5>) are the bank select bits. Each bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static RAM. All implemented banks contain Special Function Registers. Some frequently

used Special Function Registers from one bank may be mirrored in another bank for code reduction and quicker access.

3.4.3. EEPROM Memory:

The Data EEPROM and FLASH Program Memory are readable and writable during normal operation. These operations take place on a single byte for Data EEPROM memory and a single word for Program memory. A write operation causes an erase-then-write operation to take place on the specified byte or word. The EEADR register holds the address to be accessed. Depending on the operation, the EEDATA register holds the data to be written, or the data read, at the address in EEADR. The EEPROM Data memory is rated for high erase / write cycles being accessed. Read and write access to both memories take place indirectly through a set of Special Function Registers (SFR). The six SFRs used are:

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

3.5. SPECIAL FUNCTION REGISTERS:

The Special Function Registers are registers used by the CPU and peripheral modules for controlling the desired operation of the device. These registers are implemented as static RAM. Some of the special function registers used in our software are explained here.

PIE1 register:

It contains the individual enable bits for the peripheral interrupts.

OPTION_REG Register:

The OPTION_REG Register is a readable and writable register, which contains various control bits to configure the TMR0 prescaler/WDT postscaler, the External INT Interrupt, TMR0 and the weak pull-ups on PORTB.

PIR1 Register:

The PIR1 register contains the individual flag bits for the peripheral interrupts.

INTCON Register:

The INTCON Register is a readable and writable register. It contains various enable and flag bits for the TMR0 register overflow, RB Port change and External RB0/INT pin interrupts.

3.6. EEPROM REGISTERS:

EECON1 and EECON2 Registers:

The EECON1 register is the control register for configuring and initiating the access. The EECON2 register is not a physically implemented register, but is used exclusively in the memory write sequence to prevent inadvertent writes. There are many bits used to control the read and write operations to EEPROM data and FLASH program memory. The EEPGD bit determines if the access will be a program or data memory access. When clear, any subsequent operations will work on the EEPROM data memory. When set, all subsequent operations will operate in the program memory.

Read operations only use one additional bit, RD, which initiates the read operation from the desired memory location. Once this bit is set, the value of the desired memory location will be available in the data registers. This bit cannot be cleared by firmware. It is automatically cleared at the end of the read operation. For EEPROM data memory reads the data will be available in the EEDATA register in the very next instruction cycle after the RD bit is set. For program memory reads, the data will be loaded into the EEDATH: EEDATA registers, following the second instruction after the RD bit is set.

3.7. I/O PORTS:

There are five I/O ports in PIC16f877 namely PORT A, B, C, D and E, which can be configured as input or output.

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

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).

PORTC is an 8-bit wide, bi-directional port. The corresponding data direction register is TRISC. PORTC is multiplexed with several peripheral functions. PORTC pins have Schmitt Trigger input buffers. When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin. Some peripherals override the TRIS bit to make a pin

an output, while other peripherals override the TRIS bit to make a pin input.

PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output.

For port E the corresponding register is TRISE

3.8. TIMER MODULE:

The Timer0 module is 8-bit, readable and writable, 8-bit software programmable prescaler, Internal or external clock select enabled timer/counter. It has an Interrupt on overflow from FFh to 00h.

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L), that are readable and writable. The TMR1 register pair (TMR1H: TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 Interrupt, if enabled, is generated on overflow, which is latched in interrupt flag bit TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit TMR1IE (PIE1<0>).

Timer2 is an 8-bit timer with a prescaler and a postscaler. It can be used as the PWM time-base for the PWM mode of the CCP module(s). The

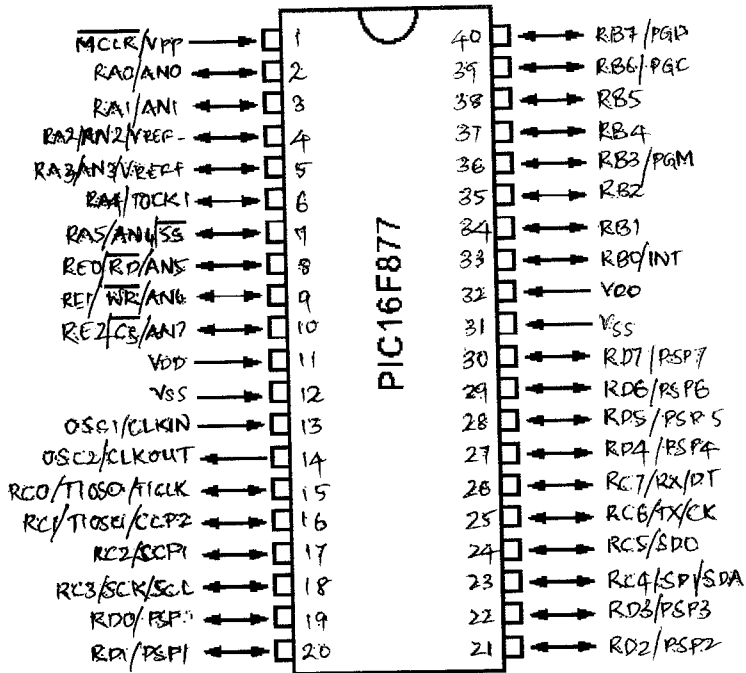
TMR2 register is readable and writable, and is cleared on any device RESET.

3.8. INTERRUPT:

The PIC16F87X family has up to 14 sources of interrupt. The interrupt control register (INTCON) records individual interrupt requests in flag bits. It also has individual and global interrupt enable bits. A global interrupt enable bit, GIE (INTCON<7>) enables (if set) all unmasked interrupts, or disables (if cleared) all interrupts. When bit GIE is enabled, and an interrupt's flag bit and mask bit is set, the interrupt will vector immediately.

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

3.10. PIN OUT



IMPLEMENTATION:

4.1. HARDWARE DESCRIPTION:

In the existing hardware our software module controls two important ICs namely

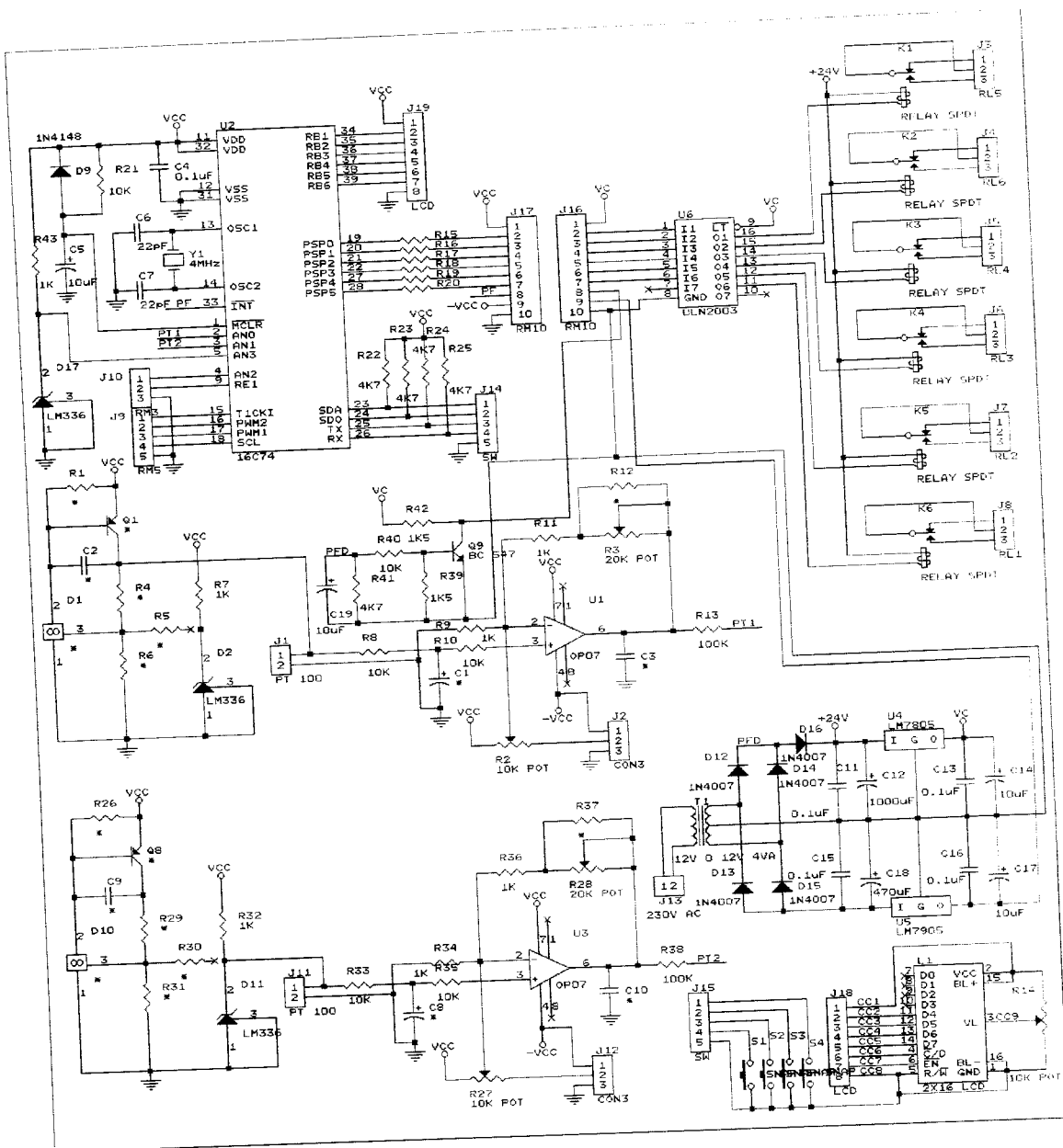
- PIC16F877 – the controller
- LM16200 – the LCD

In the controller out of the five ports only three ports are used by the hardware. Port B drives the LCD, port D is connected to the relays, which controls the valves and port C is connected to the keypad, hence configured as input.

The LCD is a 16X2, 5X7dots and a 2 line display LCD. The LCD has got a backlight display, which will be useful in viewing in dark surroundings. The LCD has been interfaced for 4-bit data transfer.

The circuit diagram has been shown in the next page.

4.2. CIRCUIT DIAGRAM



4.3. SOFTWARE MODULE:

4.3.1. C-AN OVERVIEW:

The software was programmed in Microchip's version of 'C'. C was preferred because of its wide popularity, ease of use compared to assembly language, good programming constructs and rich reference sources. It ensures high performance and offers very good hardware interaction due to its close relationship with assembly language programming and other high level language programming by providing raw programming power of assembly language and the understandable programming style of the other high level languages. By suitably making changes in the compiler options the same 'C' code can be easily ported to different microcontrollers by making little or no changes in the source code.

4.3.2. DEVELOPMENT SUPPORT:

MPLAB-C is an ANSI-based 'C' compiler for the Microchip technology incorporated PIC16/17 microcontroller families. Due to restrictions imposed by the microcontroller architecture, MPLAB-C does not support the full ANSI-C standard. The MPLAB IDE also offers a full-featured macro assembler, a 'C' compiler and a simulator. In addition to this various third part companies like Hi-tech Craft, CCS etc. are also

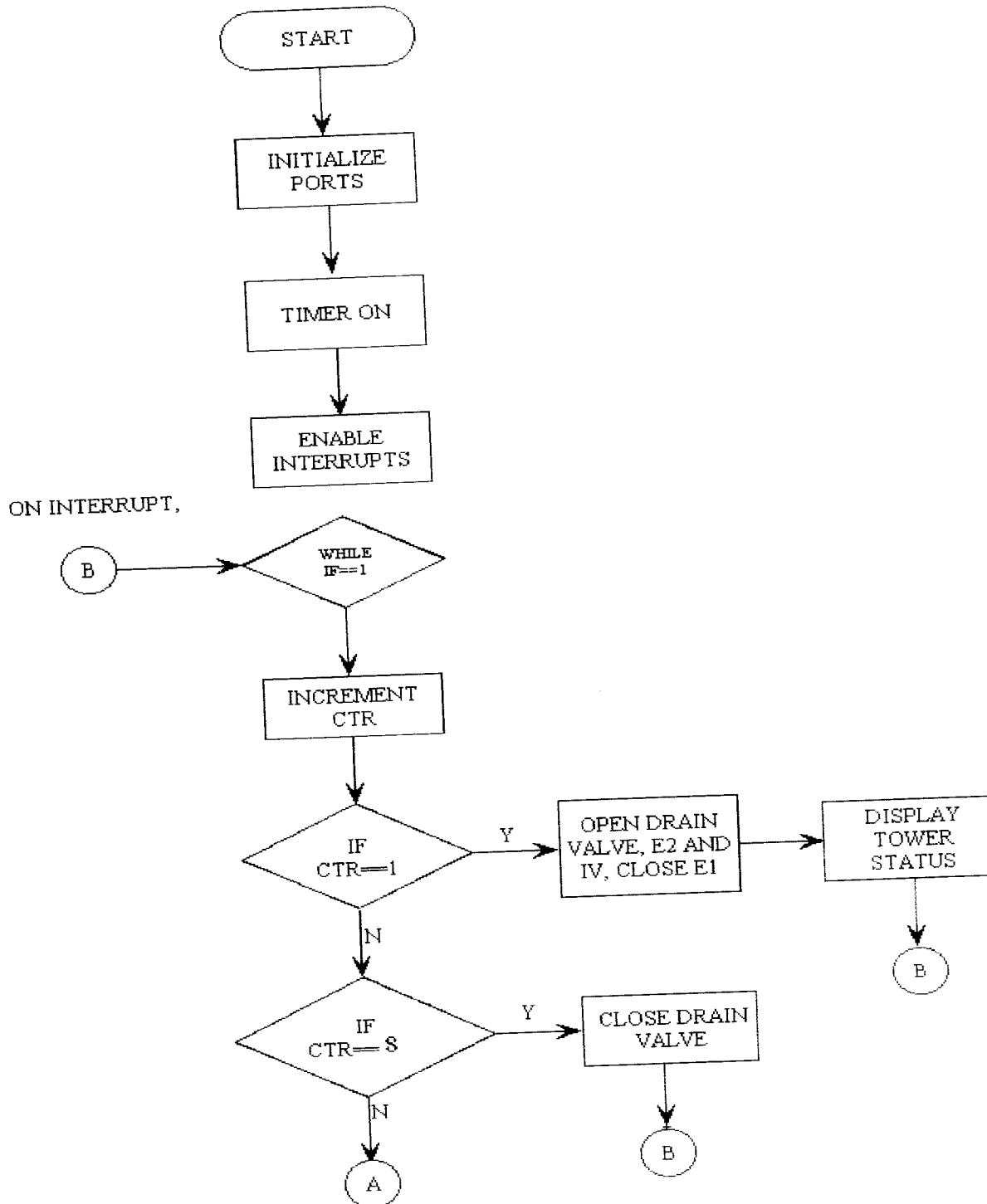
developing support tools in the form of various 'C' compilers and debugging tools for Microchip thereby giving us a wide range of development options.

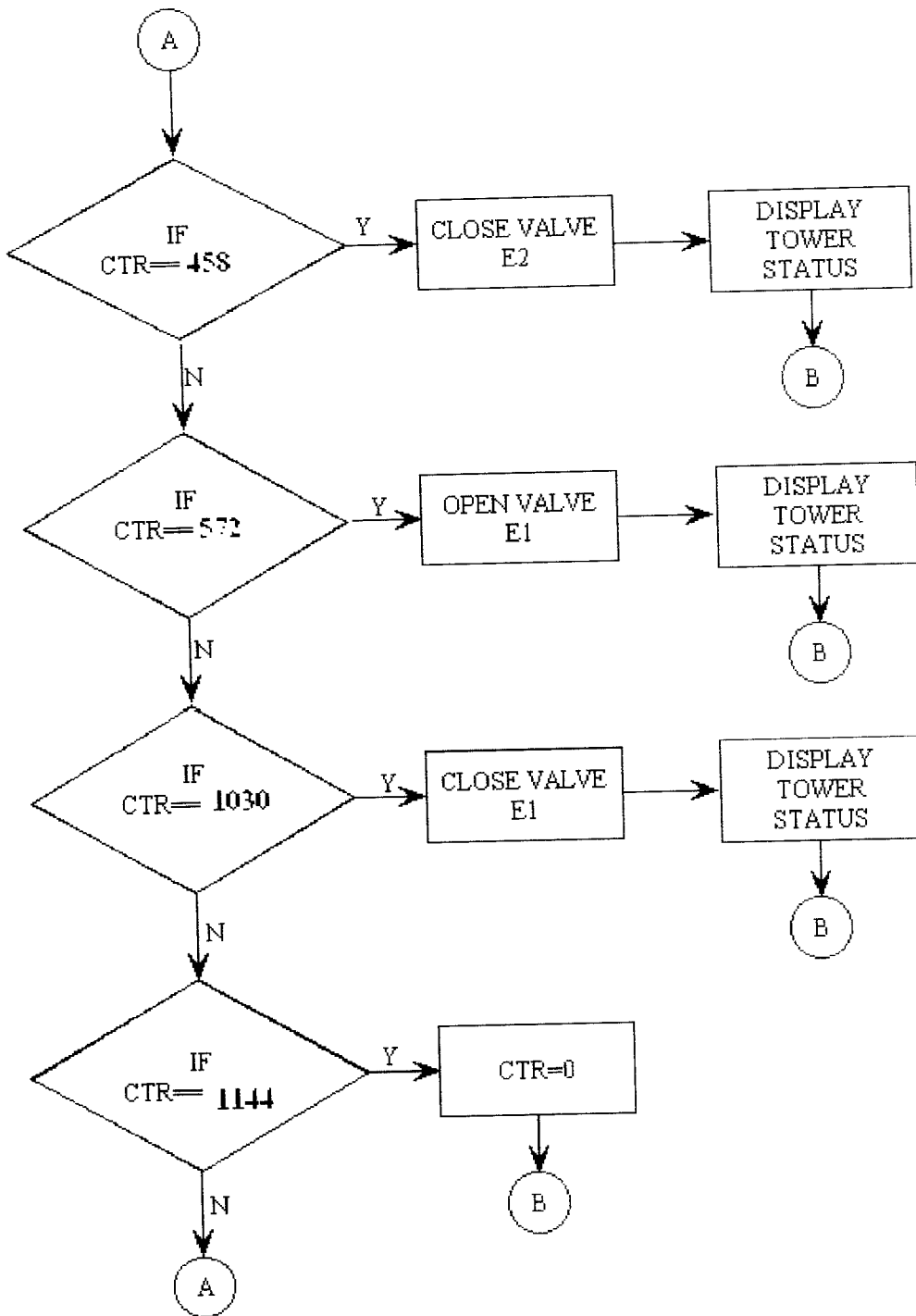
4.3.3. PROGRAM MODULES:

Functionally the software can be divided into four modules each carrying out a particular function. They are,

- Valve control
- Key scan and Menu entry
- Alarm module
- Display module

4.3.3.1. VALVE CONTROL





Description:

This module performs the task of controlling the valves i.e., opening and closing the respective valves on a time basis. For timing this module we use the timer0 interrupt routine.

The valves are controlled by the output of port D. The drain valve, inlet valve (IV), valve E1 and E2 are controlled by pins 0,1,2 & 3 of port D respectively. Initially all these pins are configured to be output pins by setting logic high level. By changing the logic levels to high or low, the valves can be opened or closed.

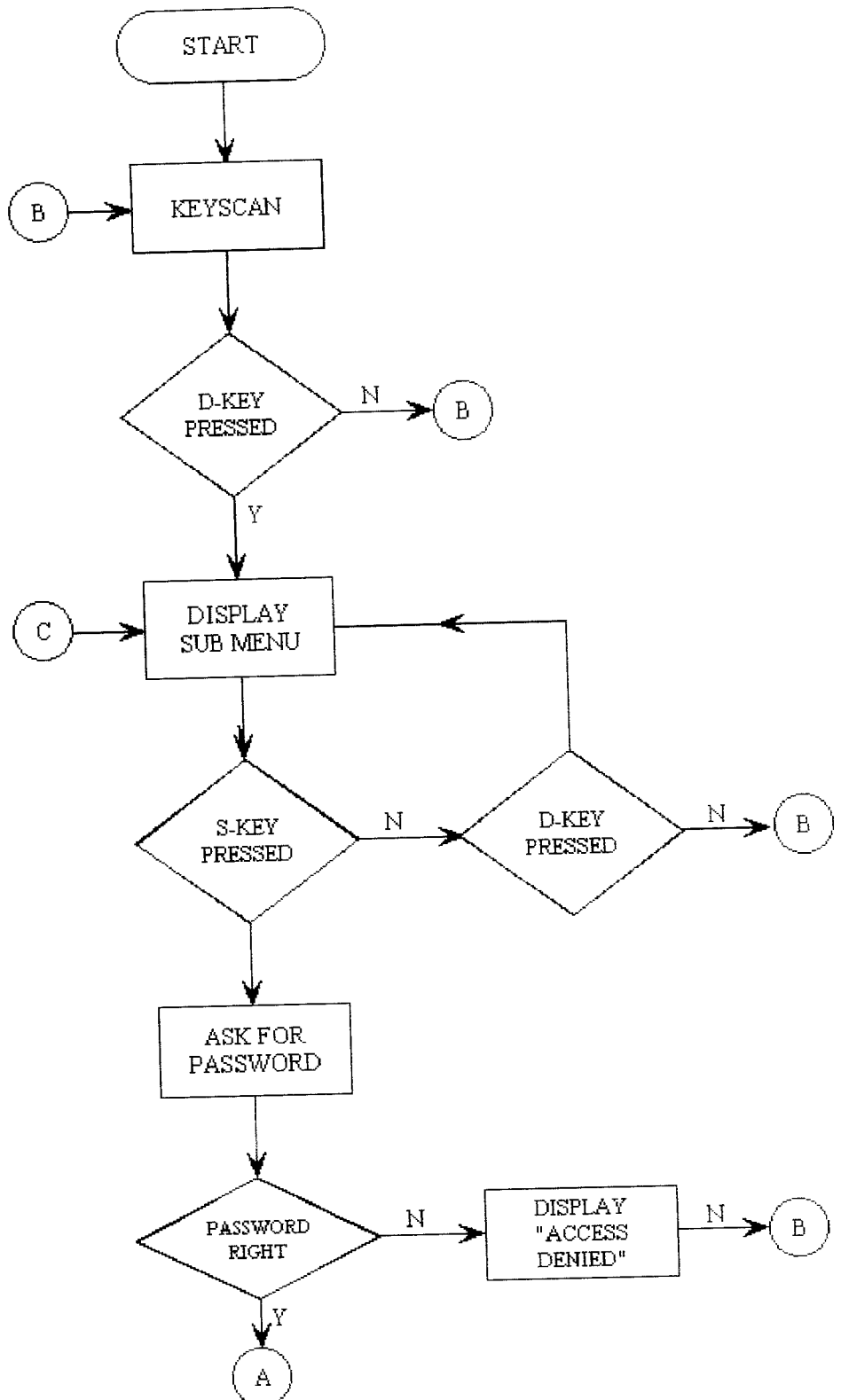
The timing cycle for the entire 10 minutes has been defined in the previous section. As explained already, for the first four seconds drain valve should remain open after which it should remain closed for the entire 10-minute cycle. To calculate the time initially the timer1 is turned on by configuring the T1CON register. The 0th bit of T1CON register is set so that the timer is enabled. Next the timer1 interrupt is enabled by setting the 0th bit of PIE1 register. Every time the timer overflows the TMRIF (0th bit) of PIR1 register is set and whenever this flag is set the control branches to the vector location 0004h. In this vector location, the routine for controlling the valves at the required time intervals is written.

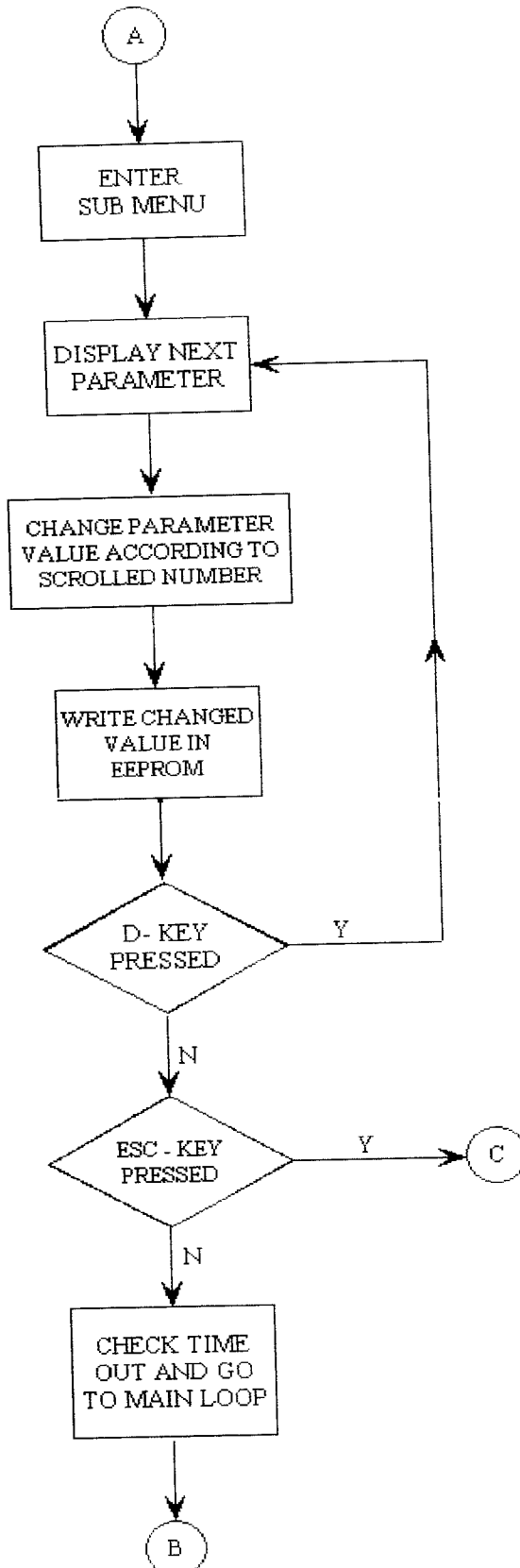
In this interrupt service routine, a counter is incremented every time, the control branches to that location. Every timer overflow consumes (Including prescalar). At the beginning of each cycle IV, drain valve & valve E2 should be opened. So when the counter is 1 these valves are opened. At this juncture the tower 1 is in drying status and tower 2 is in regeneration status. This status is displayed by calling the display function. Next, after duration of four seconds, i.e., when the count reaches 8 the drain valve is closed by making the 0th pin of port D low.

After closing this valve, the control branches back to the main routine. The process of incrementation goes on and every time the control goes to the ISR, the count is checked. When the count reaches 458 (which is nothing but 4 minutes) the valve E2 is closed. Now the status of the tower 2 changes to repressurisation. The function to indicate this change in the display is called.

Likewise after every time interval shown in table 1.1, the valve control is changed by making the corresponding pins high or low. The tower status is also updated regularly (table 1.2) by calling the corresponding display functions.

4.3.3.2. KEYSKAN AND MENU ENTRY





Description:

This module involves continuously scanning the port connected to the keypad (in our case pins 4,5,6 & 7 of port C) and displaying and providing various functionality according to the key pressed. Under normal conditions the tower status will be displayed. Firstly during a keyscan if it is found that the down key is pressed i.e., if port C 5th pin goes high then, a function to display the first submenu i.e., FACTORY SETTINGS is called. On further pressing the down key each of the submenus are displayed one by one for each down key press by calling the corresponding functions. If then the select key is pressed then a function to verify the password is called.

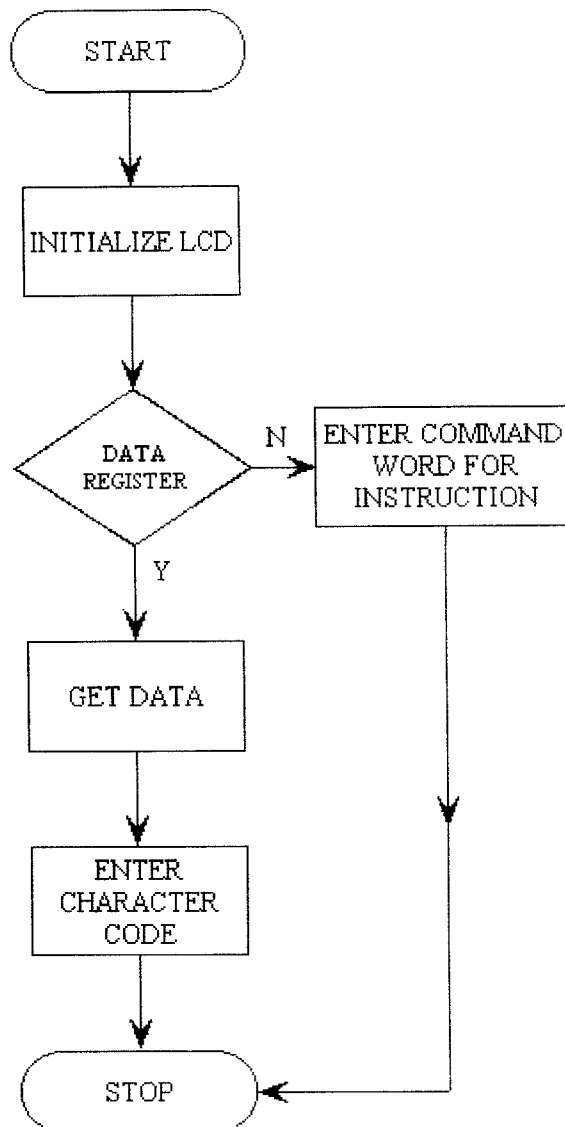
Now if the password combination i.e., the up key and select key is pressed together correctly the password is accepted. This is achieved by checking continuously whether the corresponding ports are high and subsequently allowing a delay for the required amount of time, and checking whether the ports remain high throughout the delay. After the password is accepted the function to display the parameters of the corresponding submenu is called. Again the port corresponding to the down key is scanned and for each key press the next parameter is displayed.

All the display requirements in this module are accomplished using the same functions, but using different arguments according to the display details. Now for each parameter the values remain stored in the EEPROM. These values are initially read from EEPROM and displayed. Now after entering the submenu, to change these values the down/up key is made use of. For each digit there are separate locations allotted in the EEPROM memory. For each digit, by sensing the number of times the down/up key is pressed the corresponding character code of the number being selected is written in the EEPROM and the display module actually takes this value and displays. The same procedure is repeated for all the digits. This task is done by a separate function, which takes in the number of digits as an argument. Another function is used to display the units for each parameter.

After entering all the digits if the down key is pressed then the control goes to the next parameter which involves the same procedure. Now if the escape key is pressed, the display goes back to the main menu. If no key is pressed a delay loop is called and even after that no key is pressed then the display or control goes to the main loop. There are three sub menus and each of them can be accessed by pressing the down key and then the select key. Thereafter using the same procedure the different parameters are written to EEPROM and the same is read and displayed. At

any point of time a timeout will cause the controller to switch its control to the main loop and therefore the tower status will be displayed.

4.3.3.3. DISPLAY MODULE



Description:

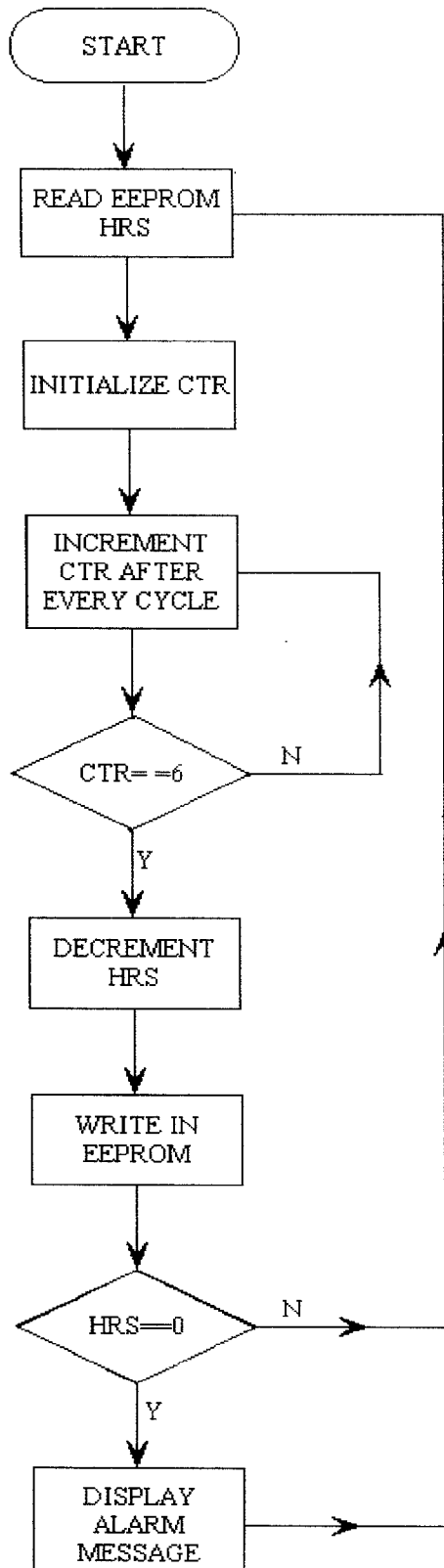
The LCD module is used to display the status of the towers, the functions, the parameters and their values. The same algorithm is used to display all these modules. To accomplish this, first the LCD should be initialized.

To interface the LCD with the controller the LCD has to be programmed accordingly. The process of initialization differs for both 4 bit and 8 bit data. For our requirement we use 4 bit data. To program the LCD the instructions has to be written into the instruction register and to display data, the corresponding character code has to be written into the data register. The instructions for initialization are written into the instruction register with delays as specified in the data manual (the process of initialization has been explained in the data manual enclosed). These instructions include instruction for display off, display on and entry mode set.

The data / instruction register is chosen by giving a high / low pulse to the 4th pin of the LCD (LM16200). For displaying data, first the instruction register is chosen by giving a high pulse to the corresponding pin and the instruction words for clearing the display, the position of display etc. are written into the instruction register.

Next the data register is chosen by changing the pulse level at the corresponding pin. Then the character code for the data to be displayed is fed into the data register. The LCD module then reads the character code and displays the corresponding data. The LCD can also be used to scroll messages. For this the instruction register is written with the instruction word for scrolling messages, which is specified in the data manual. Before writing the character code for the data, the instruction specifying the DDRAM address and CGRAM address should be written into the instruction register in order to indicate the position of display and the position of the cursor.

4.3.3.4. ALARM MODULE



Description:

The controller should give alarm messages regarding the change of filters, desiccants and seal. The duration after which these messages are to be displayed is previously entered by the user. This data is available in the EEPROM. This value has to be decremented after every hour and finally the alarm message is displayed.

For this time control the counter used for the valve control module is utilized. Each time the ten-minute cycle completes another counter variable is incremented. When this counter reaches the value 6 it means that one-hour has elapsed. After every hour the value in the EEPROM has to be decremented. So whenever the counter reaches 6, read the value from the EEPROM for each parameter, reduce it by one and again write the new value in EEPROM.

Now check whether the value for all the parameters in the EEPROM has reached zero. If any of the parameters has reached zero, it means that the specified time duration has elapsed and the alarm for that particular parameter should be displayed. This is done by calling the display function that displays the corresponding alarm message.

After the alarm message is noted by the user and ESC key is pressed, then the counter is initialized to 0 again and the same process is repeated.

4.3.4. PROGRAM DESCRIPTION:

Starting from the main menu, the main menu first initializes the ports. In order to initialize the ports, the TRISA, TRIS B AND TRIS C registers should be configured. The unused ports should be configured as output ports.

After initializing the ports, next all the registers which are being used are configured. The timer1 is enabled by configuring the T1CON register. Next, the advt. should be displayed. For that the LCD is initialized for startup by writing the required command words in the instruction register with adequate delays in between. This is done by the `lcd_init()` function. Next the data register is chosen. Writing any 8 bit data into the data register or instruction register is done by the `lcd_inst()` function. This function selects the corresponding register and then writes the MSB and LSB in the register.

Next, the `scroll_display()` function is called, which is used to display the scrolling advertisement. This function takes text to be displayed as a parameter and displays it. The function contains the character code for all the characters. This function is put in a loop to run for a minute. All this

has to be done only once on startup. Hence after this an infinite loop to scan for any key being pressed (keyscan()) is given.

When this is getting executed, parallelly the timer runs. The timer register increments from 0000h to FFFFh and rolls over to 00h. Timer1 interrupt is enabled and so interrupt is generated on overflow. As interrupt is responded to, the return address is pushed onto the stack and the PC is loaded with 0004h, where ISR has been written.

The ISR contains the code for valve control and maintenance checking. The function open_valve is called after particular set values of a counter based on the time. This function, according to the parameter passed closes or opens the corresponding valves.

The time taken for an overflow in the timer registers 525 milliseconds. Hence the counter in ISR is incremented for every 525 milliseconds

After four seconds the drain valve has to be closed. The counter should reach a value of 8 for 4 seconds to pass. When the count is 8, the function “open_valve” is again called & the drain valve is closed.

In the same way, the count is checked for 4th, 5th, 9th and 10th minute and the valve positions are changed as per the requirement.

As the valve positions are changed, the status of the tower gets changed, which has to be displayed. For displaying the status, the function “Display3” is called with the status as arguments. The arguments include the status of tower1 and tower2. In the function, each character of the argument is checked and corresponding character code is given to the data register using the `lcd_inst()` function which takes the job of displaying the character on LCD. The status of the towers is continuously displayed until any key is pressed.

The function `keyscan()` continuously checks for Key press. If down key is pressed, “display2” function is called. If key press is for the first time, then “FACTORY SETTINGS” is displayed. The display logic is similar to that used in “display3” except for the number of arguments being only one. With “FACTORY SETTINGS” on the display, for entering the menu select key has to be pressed. When select key is pressed with a menu on the display then the user is asked for a password. Password is given in the form of key press of a predefined combination. Password is checked by the `password_check ()` function. It scans both the password keys for the required time.

If the password is wrong then “access denied” will be displayed. If the password is correct then submenu access will be allowed, displaying

the submenu. Again by scanning down key press and select key press by corresponding check function, other sub menus can be selected and viewed.

If the value of the parameters have to be changed then `value_enter()` function is called. With up and down key presses, each digit can be changed to the required value. The new value is then written to the EEPROM. This is done by the function `EEPROM_WRITE ()`. As escape key is pressed, the display shows the menu. If again the escape key is pressed, then status of the towers is displayed until another key press.

To display alarm for changing the filter, desiccant and the seal the code is included in the ISR. The number of hours after which these have to be changed is entered in the maintenance chart. This value is decremented as each hour passes. The calculation of time is again done using the counter that was used in valve control module. When the value becomes zero a message is displayed informing that either the filter or the seal or the desiccants have to be changed depending on the parameter whose value becomes zero.

CONCLUSION:

The microcontroller based controller for heatless desiccant dryer has been programmed and tested. It has successfully met the minimal design objectives. Still there is room for improvement.

The minimum possible resources have been utilized by this project thus making it economically viable and with necessary modifications in the port this design can be adopted for a wide range of similar dryer control applications.

Since the software is the main controlling entity in the project, changes in the software can be more easily made as compared to making changes in the hardware design. This is in line with the general design policy for any microcontroller-based system. The above point has already been validated by our choosing 'C' as the programming medium since changes can be made more easily in 'C' than in assembly language and 'C' can be ported to other microcontrollers, with little or no changes, the essential program logic remaining the same.

Bibliography

BIBLIOGRAPHY

- Microchip Technical Library CD-ROM, Microchip Technology Inc.,
USA
- Microchip-Microcontroller Data Hand Book
- Heatless Desiccant Dryer Catalogue
- www.microchip.com
- Microprocessors and Interfacing, Douglas. V. Hall, Tata Mc Grawhill
- Programming in 'C', Gottfried, Schaum's Series, Tata Mc Grawhill
- Let us 'C', Yashwant kanitnar, BPB publications

Source Code

SOURCE CODE

```
#include<p16f877.h>
#include<b_struct.h>
#define PB(p,b) ((unsigned) &(p)*8+(b));
#define LCD_ENABLE PB(PORTB,6)
#define LCD_REGSEL PB(PORTB,5)
#define upkey PB(PORTC,4)
#define selectkey PB(PORTC,6)
#define downkey PB(PORTC,5)
#define escapekey PB(PORTC,7)
#define drainvalve PB(PORTD,0)
#define inletvalve PB(PORTD,1)
#define valveE1 PB(PORTD,2)
#define valveE2 PB(PORTD,3)
#define downkey_check_flag (((bits *)&flag2)->BIT4)
#define ENABLE_FLAG (((bits *)&flag2)->BIT0)
#define time_units_flag (((bits *)&flag2)->BIT1)
#define flow_units_flag (((bits *)&flag3)->BIT2)
#define drain_units_flag (((bits *)&flag3)->BIT3)
#define pressure_units_flag (((bits *)&flag3)->BIT4)
#define hours_units_flag (((bits *)&flag4)->BIT1)
#define COMMAND_REG 0
#define DATA_REG 1
#define COMMAND_MODE (((bits *)&flag)->BIT0)
#define CLR_DISPLAY 0b00000001
#define Cursor_Off 0b00001100 //Cursor Off
#define Shift_Dis 0b00011000 //Shift Mode
#define FUNCTION1 0b00110011 //Initializing
#define FUNCTION2 0b00100010 //Initializing
#define FUNCTION3 0b10001010 //display Off
```

```

#define                FUNCTION4                0b01110001 //display On
const unsigned char trident_add[]={ "   www.tridentpneumatics.com $"};
const unsigned char trident1 []=
{"           **Manufacturers of Compressed Air Dryers, Dessicant dryer, Dryspell,
Refrigeration dryer, Filters, Automatic Drain Valve, Level based drain valve,
Receivers, Projects** Phone no : +91 ^^^^^^^^^^^** Fax no : +91 ^^^^^^^^^^^**
Serial no :   #####** Model no : ~~~~** Visit us at
www.tridentpneumatics.com** e-mail:tridentt@satyam.net.in** TRIDENT
PNEUMATICS PVT LTD, 5/232 , KNG Pudur Road , Somayampalayam Post ,
COIMBATORE. THANKS FOR SHOWING INTEREST IN OUR
PRODUCTS***$$"};
const unsigned char trident2[]={ "FACTORY SETTINGS $"};
const unsigned char trident3[]={ "FEILD SETTINGS $"};
const unsigned char trident4[]={ "MAINTANENCE CHART $"};
//*****
unsigned char flag,flag2,flag3,flag4,temp,temp1,data,timeout_flag;
unsigned char i,j,k,h,a,y,temp_address,addr,n,m;
unsigned char maincounter,timeout,timeout1,timeout2,units_display;
unsigned char count_flag;
//*****
void Delay(void);
void BigDelay(void);
void lcd_init(void);
void lcd_inst(void);
void eeprom_read(unsigned char m);
void eeprom_write(unsigned char addr,unsigned char data);
void Scroll_display(const unsigned char *cptr1,const unsigned char *cptr2);
void Display2(const unsigned char *cptr1);
void Display3(const unsigned char *cptr1,const unsigned char *cptr2);
void key_press(unsigned char big, unsigned char small);
void open_valve(unsigned char,unsigned char,unsigned char,unsigned char);
void down_select_check(void);
void password_check(void);
void select_check(void);

```



```

void timeout_routine(void);
void down_check(void);
void enter_value(unsigned char);
void units_display_routine();
void keyscan();
void factor_settings_display();
void field_settings_display();
void maintenance_chart();
/**
void Delay(void)
{
    for(a=0;a<250;a++)
    {
        #asm
            nop
            clrwdt
        #endasm
    }
}
/**
void BigDelay(void)
{
    for (y=0;y<90;y++)
    {
        Delay();
    }
}
/**
void lcd_inst()
{
    // will get a 8-bit value & 1'st displays lsb then msb
    #asm

```

```

        clrwdt
#endasm
PORTB=0b00000000;
if(COMMAND_MODE==1)
    LCD_REGSEL=COMMAND_REG;
else
    LCD_REGSEL=DATA_REG;
PORTB=PORTB&0xf0;
#asm
    clrwdt
#endasm
    temp1=temp&0xf0;           //swapping temp value....get MSB
temp1=temp1>>3;               //rotating it right 3 shifts
PORTB=PORTB|temp1;
#asm
    clrwdt
#endasm
LCD_ENABLE=1;                 //becoz it displays at the falling
edge
#asm
    nop
    nop
    clrwdt
#endasm
LCD_ENABLE=0;
PORTB=PORTB&0xf0;
temp1=temp&0x0f;
temp1=temp1<<1;
#asm
    clrwdt
#endasm

```

```

    #asm
        nop
        nop
        clrwdt
    #endasm
LCD_ENABLE=1;
    #asm
        nop
        nop
        clrwdt;
    #endasm
LCD_ENABLE=0;
    #asm
        nop
        nop
        clrwdt;
    #endasm
Delay();
}
//*****
void lcd_init()
{
    COMMAND_MODE=1;
    Delay();
    Delay();
    temp=FUNCTION1;
    lcd_inst();
    Delay();
    temp=FUNCTION2;
    lcd_inst();
    Delay();
}

```

```

temp=FUNCTION3;
lcd_inst();
Delay();
temp=FUNCTION4;
lcd_inst();
}
//*****
void eeprom_read(unsigned char m)
{
    EEADR=m;
    EEPGD=0;
    RD=1;
    temp=EEDATA;
}
//*****
void eeprom_write(unsigned char addr,unsigned char data)
{
    while(WR==0)
    {
        EEADR=addr;
        EEDATA=data;
        EEPGD=0;
        WREN=1;
        GIE=0;
        EECON2=0X55;
        EECON2=0XAA;
        WR=1;
        GIE=1;
        WREN=0;
    }
}
void Scroll_display(const unsigned char *cptr1,const unsigned char *cptr2)

```

```

{
unsigned char i,ee_val,ee_val2,ee_val3,ee_val4;
COMMAND_MODE=1;           //instruction/data register
temp=CLR_DISPLAY;         //firstly clear the display
lcd_inst();                //load this instruction into ddram
temp=0x00;
lcd_inst();                //load this instruction into ddram
COMMAND_MODE=0;           //change the mode to data register mode
data = *cptr1;            //first element put into data
while(data!='$' && escapekey!=0)
{
temp=data;                //data into temp
lcd_inst();                //load the instruction into ddram
cptr1++;                  //increment then pointer
data=*cptr1;              //data from the pointer is inserted into data
}
while(*cptr2!='$' && escapekey!=0)
{
BigDelay();
COMMAND_MODE=1;
temp=0xb9;
lcd_inst();                //load this instruction into ddram
COMMAND_MODE=0;           //change the mode to data register
for(i=0,ee_val=0x0b,ee_val2=0x01,ee_val3=23,ee_val4=34;i<=16    &&
*cptr2!='$' && escapekey!=0);i++)
{
temp = *cptr2; //data into temp
if(temp == 'A')
temp=01000001;
elseif(temp == 'B')

```

```
temp=01000010;
elseif(temp == 'C')
    temp=01000011;
elseif(temp == 'D')
    temp=01000100;
elseif(temp == 'E')
    temp=01000101;
elseif(temp == 'F')
    temp=01000110;
elseif(temp == 'G')
    temp=01000111;
elseif(temp == 'H')
    temp=01001000;
elseif(temp == 'I')
    temp=01001001;
elseif(temp == 'J')
    temp=01001010;
elseif(temp == 'K')
    temp=01001011;
elseif(temp == 'L')
    temp=01001100;
elseif(temp == 'M')
    temp=01001101;
elseif(temp == 'N')
    temp=01000010;
elseif(temp == 'O')
    temp=01001011;
elseif(temp == 'P')
    temp=01010000;
elseif(temp == 'Q')
    temp=01010001;
```

```
elseif(temp == 'R')
    temp=01010010;
elseif(temp == 'S')
    temp=01010011;
elseif(temp == 'T')
    temp=01010100;
elseif(temp == 'U')
    temp=01010101;
elseif(temp == 'V')
    temp=01010110;
elseif(temp == 'W')
    temp=01010111;
elseif(temp == 'X')
    temp=01011000;
elseif(temp == 'Y')
    temp=01011001;
elseif(temp == 'Z')
    temp=01011010;
elseif(temp == '0')
    temp=00110000;
elseif(temp == '1')
    temp=00110001;
elseif(temp == '2')
    temp=00110010;
elseif(temp == '3')
    temp=00110011;
elseif(temp == '4')
    temp=00110100;
elseif(temp == '5')
    temp=00110101;
elseif(temp == '6')
```

```

        temp=00110100;
    elseif(temp == '7')
        temp=00110101;
    elseif(temp == '8')
        temp=00110100;
    elseif(temp == '9')
        temp=00110101;
    elseif(temp == ' ')
        temp=00000000;
    elseif(temp == ',')
        temp=00101100;
    elseif(temp == '.')
        temp=00101110;
    elseif(temp == '#')
        temp=EEPROM_READ(ee_val++);
    else if(temp == '~')
        temp=EEPROM_READ(ee_val2++);
    else if(temp == '^')
        temp=EEPROM_READ(ee_val3++);
    else if(temp == '"')
        temp=EEPROM_READ(ee_val4++);

    lcd_inst();
    cptr2++;           //increment then pointer
    Delay();         //delay added
}
if(*cptr2=='$' || escapekey==0) return;
cptr2 = cptr2 - 16;

```

```

}
}
//*****

```



```

void Display2(const unsigned char *cptr1)
{
    COMMAND_MODE=1;           //instruction/data register
    temp=CLR_DISPLAY;        //firstly clear the display
    lcd_inst();
    temp=0x00;               //scroll in sec line =0xb9
    lcd_inst();
    COMMAND_MODE=0;         // to data register
    data=*cptr1;            //first element put into data
    while(data!='$')
    {
        temp=data;          //data into temp
        if(temp == 'A')
            temp=01000001;
        elseif(temp == 'B')
            temp=01000010;
        elseif(temp == 'C')
            temp=01000011;
        elseif(temp == 'D')
            temp=01000100;
        elseif(temp == 'E')
            temp=01000101;
        elseif(temp == 'F')
            temp=01000110;
        elseif(temp == 'G')
            temp=01000111;
        elseif(temp == 'H')
            temp=01001000;
        elseif(temp == 'I')
            temp=01001001;
        elseif(temp == 'J')

```

```
temp=01001010;
elseif(temp == 'K')
temp=01001011;
elseif(temp == 'L')
temp=01001100;
elseif(temp == 'M')
temp=01001101;
elseif(temp == 'N')
temp=01000010;
elseif(temp == 'O')
temp=01001011;
elseif(temp == 'P')
temp=01010000;
elseif(temp == 'Q')
temp=01010001;
elseif(temp == 'R')
temp=01010010;
elseif(temp == 'S')
temp=01010011;
elseif(temp == 'T')
temp=01010100;
elseif(temp == 'U')
temp=01010101;
elseif(temp == 'V')
temp=01010110;
elseif(temp == 'W')
temp=01010111;
elseif(temp == 'X')
temp=01011000;
elseif(temp == 'Y')
temp=01011001;
```

```
elseif(temp == 'Z')
    temp=01011010;
elseif(temp == '0')
    temp=00110000;
elseif(temp == '1')
    temp=00110001;
elseif(temp == '2')
    temp=00110010;
elseif(temp == '3')
    temp=00110011;
elseif(temp == '4')
    temp=00110100;
elseif(temp == '5')
    temp=00110101;
elseif(temp == '6')
    temp=00110100;
elseif(temp == '7')
    temp=00110101;
elseif(temp == '8')
    temp=00110100;
elseif(temp == '9')
    temp=00110101;
elseif(temp == ' ')
    temp=00000000;
elseif(temp == ',')
    temp=00101100;
elseif(temp == '.')
    temp=00101110;
```

```
lcd_inst();
```

```
cptr1++;
```

```
//increment then pointer
```

```

        data=*cptr1;                //first element into data
    }
    BigDelay();
}
//*****
void Display3(const unsigned char *cptr1,const unsigned char *cptr2)
{
    COMMAND_MODE=1;                //instruction/data register
    if(clear_display==1)
    {
        temp=CLR_DISPLAY;          //firstly clear the display
        lcd_inst();
    }
    temp=0x00;
    lcd_inst();
    COMMAND_MODE=0;                //change to data register
    data=*cptr1;                    //first element put into data
    while(data!='$')
    {
        temp=data;
        if(temp == 'A')
            temp=01000001;
        elseif(temp == 'B')
            temp=01000010;
        elseif(temp == 'C')
            temp=01000011;
        elseif(temp == 'D')
            temp=01000100;
        elseif(temp == 'E')
            temp=01000101;
        elseif(temp == 'F')

```

```
temp=01000110;
elseif(temp == 'G')
    temp=01000111;
elseif(temp == 'H')
    temp=01001000;
elseif(temp == 'I')
    temp=01001001;
elseif(temp == 'J')
    temp=01001010;
elseif(temp == 'K')
    temp=01001011;
elseif(temp == 'L')
    temp=01001100;
elseif(temp == 'M')
    temp=01001101;
elseif(temp == 'N')
    temp=01000010;
elseif(temp == 'O')
    temp=01001011;
elseif(temp == 'P')
    temp=01010000;
elseif(temp == 'Q')
    temp=01010001;
elseif(temp == 'R')
    temp=01010010;
elseif(temp == 'S')
    temp=01010011;
elseif(temp == 'T')
    temp=01010100;
elseif(temp == 'U')
    temp=01010101;
```

```
elseif(temp == 'V')
    temp=01010110;
elseif(temp == 'W')
    temp=01010111;
elseif(temp == 'X')
    temp=01011000;
elseif(temp == 'Y')
    temp=01011001;
elseif(temp == 'Z')
    temp=01011010;
elseif(temp == '0')
    temp=00110000;
elseif(temp == '1')
    temp=00110001;
elseif(temp == '2')
    temp=00110010;
elseif(temp == '3')
    temp=00110011;
elseif(temp == '4')
    temp=00110100;
elseif(temp == '5')
    temp=00110101;
elseif(temp == '6')
    temp=00110100;
elseif(temp == '7')
    temp=00110101;
elseif(temp == '8')
    temp=00110100;
elseif(temp == '9')
    temp=00110101;
elseif(temp == ' ')
```

```

        temp=00000000;
    elseif(temp == ',')
        temp=00101100;
    elseif(temp == '.')
        temp=00101110;
    lcd_inst();
    cptr1++; //increment then pointer
    data=*cptr1; //first element into data
}
COMMAND_MODE=1;
temp=0xb9;
lcd_inst();
COMMAND_MODE=0; //change to data register
data=*cptr2; //first element put into data
while(data!='$')
{
    temp=data; //data into temp
    if(temp == 'A')
        temp=01000001;
    elseif(temp == 'B')
        temp=01000010;
    elseif(temp == 'C')
        temp=01000011;
    elseif(temp == 'D')
        temp=01000100;
    elseif(temp == 'E')
        temp=01000101;
    elseif(temp == 'F')
        temp=01000110;
    elseif(temp == 'G')
        temp=01000111;
}

```

```
elseif(temp == 'H')
    temp=01001000;
elseif(temp == 'I')
    temp=01001001;
elseif(temp == 'J')
    temp=01001010;
elseif(temp == 'K')
    temp=01001011;
elseif(temp == 'L')
    temp=01001100;
elseif(temp == 'M')
    temp=01001101;
elseif(temp == 'N')
    temp=01000010;
elseif(temp == 'O')
    temp=01001011;
elseif(temp == 'P')
    temp=01010000;
elseif(temp == 'Q')
    temp=01010001;
elseif(temp == 'R')
    temp=01010010;
elseif(temp == 'S')
    temp=01010011;
elseif(temp == 'T')
    temp=01010100;
elseif(temp == 'U')
    temp=01010101;
elseif(temp == 'V')
    temp=01010110;
elseif(temp == 'W')
```



```
temp=01010111;
elseif(temp == 'X')
temp=01011000;
elseif(temp == 'Y')
temp=01011001;
elseif(temp == 'Z')
temp=01011010;
elseif(temp == '0')
temp=00110000;
elseif(temp == '1')
temp=00110001;
elseif(temp == '2')
temp=00110010;
elseif(temp == '3')
temp=00110011;
elseif(temp == '4')
temp=00110100;
elseif(temp == '5')
temp=00110101;
elseif(temp == '6')
temp=00110100;
elseif(temp == '7')
temp=00110101;
elseif(temp == '8')
temp=00110100;
elseif(temp == '9')
temp=00110101;
elseif(temp == ' ')
temp=00000000;
elseif(temp == ',')
temp=00101100;
```

```

        elseif(temp == '.')
            temp=00101110;

        lcd_inst();
        cptr2++;           //increment then pointer
        data=*cptr2;     //2nd element into data
    }
}
//*****
void key_press(unsigned char big,unsigned char small)
{
    for (y=0;y<big;y++)
    {
        for(a=0;a<small;a++)
        {
            asm("clrwdt");
            asm("clrwdt");
        }
    }
}
//*****
open_valve(unsigned char x,unsigned char y,unsigned char z,unsigned char d)
{
    drainvalve=x;
    inletvalve=y;
    valveE1=z;
    valveE2=d;
}
//*****
void down_select_check()
{
    while((downkey==1)&&(selectkey==1)&&(timeout2<10))
    {

```

```

while((downkey==1)&&(selectkey==1)&&(timeout1<250))
{
    while((downkey==1)&&(selectkey==1)&&(timeout<250))
    {
        asm("clrwdt");
        timeout++;
    }
    timeout=0;
    timeout1++;
}
timeout1=0;
timeout2++;
}
}
/*****
void password_check()
{
    down_select_check();
    Display2("...PLEASE WAIT.....$");
    timeout=0;
    timeout1=0;
    timeout2=0;
    while((upkey==0)&&(selectkey==0)&&(downkey==1)&&(timeout2<=10))
    {
        while((upkey==0)&&(selectkey==0)&&(downkey==1) && (timeout1<250))
        {
            while((upkey==0)&&(selectkey==0)&&(downkey==1)&&(timeout<250))
            {
                asm("clrwdt");
                timeout++;
            }
        }
    }
}

```

```

        timeout=0;
        timeout1++;
    }
    if(timeout2==1)
        Display2("...VERIFYING.....$");
    timeout1=0;
    timeout2++;
}

if(timeout2<10)
    Display2("..ACCESS DENIED..$");
}
//*****
void select_check()
{
    timeout=0;
    timeout1=0;
    timeout2=0;
    while((selectkey==0)&&(timeout2<10))
    {
        while((selectkey==0)&&(timeout1<250))
        {
            while((selectkey==0)&&(timeout<250))
            {
                asm("clrwdt");
                timeout++;
            }
            timeout=0;
            timeout1++;
        }
        timeout1=0;
        timeout2++;
    }
}

```

```

    }
    timeout2=0;
}
//*****
void timeout_routine()
{
    while((downkey==1)&&(selectkey==1)&&(timeout2<10))
    {
        while((downkey==1)&&(selectkey==1)&&(timeout1<250))
        {
            while((downkey==1)&&(selectkey==1)&&(timeout<250))
            {
                asm("clrwdt");
                timeout++;
            }
            timeout=0;
            timeout1++;
        }
        timeout1=0;
        timeout2++;
    }
}
//*****
void down_check()
{
    while(downkey==0)
    { asm("clrwdt"); }
    timeout=0;
    timeout1=0;
    timeout2=0;
    while((downkey==1)&&(timeout2<10))
    {

```

```
while((downkey==1)&&(timeout1<250))
```

```
{
```

```
    while((downkey==1)&&(timeout<250))
```

```
    {
```

```
        asm("clrwdt");
```

```
        timeout++;
```

```
    }
```

```
    timeout=0;
```

```
    timeout1++;
```

```
}
```

```
timeout1=0;
```

```
timeout2++;
```

```
}
```

```
}
```

```
//**************************************************************************
```

```
void units_display_routine()
```

```
{
```

```
//This is basically used for displaying units for the displayed values
```

```
if(time_units_flag==1)
```

```
{
```

```
    COMMAND_MODE=0;
```

```
    temp=0x20; //
```

```
    lcd_inst();
```

```
    COMMAND_MODE=1;
```

```
    temp=0b00000110;
```

```
    lcd_inst();
```

```
    COMMAND_MODE=0;
```

```
    temp=0x6d; //m
```

```
    lcd_inst();
```

```
    COMMAND_MODE=1;
```

```
    temp=0b00000110;
```

```

lcd_inst();
COMMAND_MODE=0; //i
temp=0x69;
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0; //n
temp=0x6e;
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
}
elseif(flow_units_flag==1)
{
COMMAND_MODE=0; //
temp=0x20;
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0; //c
temp=0x63;
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0; //f
temp=0x66;
lcd_inst();

```

```

COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0;
temp=0x6d;
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
}
else if(drain_units_flag==1)
{
COMMAND_MODE=0;
temp=0x20;
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0;
temp=0x73;
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0;
temp=0x65;
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
}

```

//m

//

//s

//e


```

COMMAND_MODE=0;
temp=0x63; //c
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
}
else if(pressure_units_flag==1)
{
COMMAND_MODE=0;
temp=0x20; //
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0;
temp=0x62; //b
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0;
temp=0x61; //a
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0;
temp=0x72; //r
lcd_inst();
COMMAND_MODE=1;

```

```

temp=0b00000110;
lcd_inst();
}
else if(hours_units_flag==1)
{
COMMAND_MODE=0;
temp=0x20; //
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0;
temp=0x48; //h
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0;
temp=0x52; //r
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
COMMAND_MODE=0;
temp=0x53; //s
lcd_inst();
COMMAND_MODE=1;
temp=0b00000110;
lcd_inst();
}

```

```

if((flow_units_flag==1) || (drain_units_flag==1) || (pressure_units_flag==1)
|| (time_units_flag==1))
{
    for(h=1;h<=4-(drain_units_flag);h++)
    {
        COMMAND_MODE=1;
        temp=0b00010000;
        lcd_inst();
    }
}
}
/*****
void VALUE_ENTER(unsigned char count_value)
{
    unsigned char lsb_value,LSB,count;
    while(selectkey == 0)
    {
        asm("clrwdt");
    }
    k = 0;
    COMMAND_MODE = 1;           //instruction/data register
    temp = 0xb9;                // address of first line
    lcd_inst();
    temp = 0b00010100;         //shift the display
    lcd_inst();
    temp = 0b00010100;
    lcd_inst();
    COMMAND_MODE = 0;
    count = count_value;
    downkey_check_flag=0;
    timeout_flag=0;
}

```

```

temp_address=m;
m++;
for(h=0;h<=count-1;h++)
{
    lsb=eeprom_read(m);
    if(h==0) lsb_value=lsb;
    m++;
    COMMAND_MODE=0;
    temp=LSB; //get the data to display
    lcd_inst();
    COMMAND_MODE=1;
    temp=0b00000110;
    lcd_inst();
    asm("clrwdt");
}
units_display_routine();
for(h=1;h<=count-1;h++) //shift left
{
    COMMAND_MODE=1;
    temp=0b00010000;
    lcd_inst();
    asm("clrwdt");
}
LSB=lsb_value; //msb back to lsb
h=1;
while(h<=count)
{
    lcd_flag=0;
    COMMAND_MODE=1; //instruction/data register

    temp=0b00010000; //firstly clear the display

```

```

lcd_inst();
COMMAND_MODE=0;
temp=LSB;
lcd_inst();
timeout=0;
timeout1=0;
timeout2=0;
while((upkey==1)&&(downkey==1)&&(selectkey==1)&&(timeout2<1))
{
while((upkey==1)&&(downkey==1)&&(selectkey==1) &&(timeout1<250))
{
while((upkey==1)&&(downkey==1)&&(selectkey==1)&&(timeout<250))
{
asm("clrwdt");
timeout++;
}
timeout=0;
timeout1++;
}
timeout1=0;
timeout2++;
}
if(LSB>=0x30 && LSB<=0x39)
{
lcd_flag=1;
if (upkey==0)
{
key_press(1,250);
if(upkey==0 && LSB!=0x39)
{
LSB++;
}
}
}
}

```

```

        downkey_check_flag=1;
    }
    if(ENABLE_FLAG==1 && upkey==0 && LSB!=0x59)
        {
            LSB=0X4E;
            downkey_check_flag=1;
        }
    }
    if (downkey==0)
    {
        key_press(2,100);
        if(downkey_check_flag==0 && h==1) break;
        if(downkey==0 && LSB!=0x30) LSB--;
    }
    if (selectkey==0)
    {
        key_press(4,220);
        key_press(2,240);
        if(selectkey==0)
        {
            h++;
            lcd_flag=0;
            timeout_flag=0;           //single digit timeout
            downkey_check_flag=1;
            COMMAND_MODE=1;         //instruction/data
            register                 temp=0b00010100;           //firstly clear
            the display

            lcd_inst();
            COMMAND_MODE=0;
            address=h + temp_address-1 ;
            eeprom_write(address,LSB);

```

```

        GIE=1;
        eeprom_write(address,LSB);
        GIE=1;
    }
}
a=0; //sigle digit timeout
b=0;
c=0;
while(c<=250&&(downkey==0||upkey==0||selectkey==0))
{
    asm("clrwdt");
    a++;
    if (a==240) b++;
    if (b==240) c++;
}
Delay();
key_press(1,100);
}
timeout_flag++;
if (timeout_flag>=100 || escapekey==0)
break;
}
if(lcd_flag==1)
{
    COMMAND_MODE=1;
    temp=0b00010000; //lastly clear the display
    lcd_inst();
    COMMAND_MODE=0;
}
}
}
//*****

```

```

void factor_settings_display()
{
maincounter=6;
for(i=1;i<=maincounter;i++)
{
down_check();
if(timeout2<=10)
{
switch(i)
{
case 1:
Display3("CYCLE TIME $","$");
time_units_flag=1;
VALUE_ENTER(3);
time_units_flag=0;
break;
case 2:
Display3("FLOW $","$");
flow_units_flag=1;
VALUE_ENTER(5);
flow_units_flag=0;
break;
case 3:
Display2("MODEL $");
VALUE_ENTER(4);
break;
case 4:
Display2("DRAIN $");
drain_units_flag=1;
VALUE_ENTER(2);
drain_units_flag=0;

```



```

        break;

    case 5:
        Display2("PRESSURE $");
        pressure_units_flag=1;
        VALUE_ENTER(3);
        pressure_units_flag=0;
        break;

    case 6:
        Display2("ENABLE $");
        enable_flag=1;
        VALUE_ENTER(2);
        enable_flag=0;
        break;
    }
}

COMMAND_MODE=1;
temp=0B00001100;
lcd_inst();
COMMAND_MODE=0;
}
//*****
void field_settings_display()
{
    maincounter=3;
    for(i=1;i<=maincounter;i++)
    {
        down_check();
        if(timeout2<=10)
        {

```

```
switch(i)
{
    case 1:
        Display2("PHONE NUMBER $");
        VALUE_ENTER(10);
        break;
    case 2:
        Display2("FAX NUMBER $");
        VALUE_ENTER(10);
        break;
    case 3:
        Display2("PRESSURE $");
        pressure_units_flag=1;
        VALUE_ENTER(4);
        pressure_units_flag=0;
        break;
    case 3:
        Display2("FLOW $");
        ENABLE_FLAG=1;
        VALUE_ENTER(2);
        ENABLE_FLAG=0;
        break;
    case 3:
        Display2("PRESSURE DROP $");
        ENABLE_FLAG=1;
        VALUE_ENTER(2);
        ENABLE_FLAG=0;
        break;
}
}
COMMAND_MODE=1;
```

```

temp=0B00001100;
lcd_inst();
COMMAND_MODE=0;
}
}
//*****
void maintenance_chart()
{
maincounter=3;
for(i=1;i<=maincounter;i++)
{
down_check();
if(timeout2<=10)
{
switch(i)
{
case 1:
Display2("FILTER CHANGE $");
hours_units_flag=1;
VALUE_ENTER(5);
hours_units_flag=0;
break;
case 2:
Display2("DESSICANT CHANGE $");
hours_units_flag=1;
VALUE_ENTER(5);
hours_units_flag=0;
break;
case 3:
Display2("CHANGE SEAL $");
hours_units_flag=1;

```

```

        VALUE_ENTER(5);
        hours_units_flag=0;
        break;
    }
}
COMMAND_MODE=1;
temp=0B00001100;
lcd_inst();
COMMAND_MODE=0;
}
}
//*****
void keyscan()
{
    asm("clrwdt");
    if(downkey==0)
    {
        Delay();           //these delays are required
        Delay();
        Delay();
        Delay();
        if(downkey==0)
        {
            press_flag=1;
            switch (count_flag)
            {
                case 0:
                    count_flag=1;
                    Display2(trident2);
                    break;

                case 1:

```

```

        count_flag=2;
        Display2(trident3);
        break;
    case 2:
        count_flag=3;
        Display2(trident4);
        break;
    }
}
Delay();
Delay();
timeout2=0;
timeout1=0;
timeout=0;
if(press_flag==1)
{
    press_flag=0;
    down_select_check();
}
if(selectkey==0&&timeout2<10)
{
    timeout=0;
    timeout1=0;
    timeout2=0;
    Display2("ENTER PASSWORD$");
    password_check();
    if(timeout2>=10)
    {
        timeout2=0;
        switch (count_flag)

```

```

    {

        case 1:
            m=0;    //(0-21)
            factor_settings_display();
            break;

        case 2:
            m=22;  //(22-50)
            field_settings_display();
            break;

        case 3:
            m=50;
            maintenance_chart_display();
            break;

    }
    select_check();
}

}

timeout_routine();
if(count_flag==0 || timeout2>=10)
{
    select_pressed_flag==1;
    while(upkey==1)
    {
        Delay();
        if(upkey==0)
        {
            Delay();
            down_pressed_flag2=1;
            break;
        }
    }
}

```

```

        }

        timeout2=0;
        normal_flag=1;
    }
    enter_flag=1;
}
/*****
void main(void)
{
    PORTA=0X00;
    PORTB=0X00;
    PORTD=0X00;
    TRISA=0x3F;
    TRISB=0x00;
    TRISC=0xFF;
    TRISD=0x00;
    PIE1=0B00000001;           //tmr1 overflow interrupt
    OPTION=0B00001000;        //wdt prescalar 1:1
    T1CON=0B00110001;         //enables timer1 and prescalar 1:8
    TMR1L=0xDB;
    TMR1H=0x0B;
    PIR1=0;                   //tmr1 if reset
    INTCON=0B11000000;        //gie and pie
    Count_flag=0;
    lcd_init();
    COMMAND_MODE=1;
    temp=Cursor_Off;          //instruction for cursor off
    lcd_inst();
    COMMAND_MODE=1;           //instruction/data register
    temp=0B00000001;         // firstly clear the display

```

```

    lcd_inst();
    for(i=0;i<=250;i++)
    {
        Scroll_display(trident_add,trident1);
        for(i=0;i<=250;i++)
        {
            asm("nop");
            asm("nop");
        }
    }
    while(1)
    {
        keyscan();
    }
}
//*****
//INTERRUPT SERVICE ROUTINE:
{
i++;
switch(i)
{
    case 1:
        open_valve (1,1,0,1);
        display3("T1 DRY$","T2 REGEN$");
        break;
    case 8:
        open_valve(0,1,0,1)
        break;
    case 458:
        open_valve (0,1,0,0);
        display3("T1 DRY$","T2 REPRUS$");

```



```

        break;
case 572:
    open_valve (0,1,1,0);
    display3("T2 DRY$", "T1 REGEN$");
    break;
case 1030:
    open_valve (0,1,0,0);
    display3("T2 DRY$", "T1 REPRUSS$");
    break;
case 1144:
    i=0;
    j++;
    if(j==6)
    {
        m=50;
        m=m+5;
        n=eeprom_read(m);
        while(n==0x30)
        {
            eeprom_write(m,0x39);
            m--;
            n=eeprom_read(m);
        }
        eeprom_write(m,n--);
        if(m==0x30 && m-1==0x30 && m-2==0x30 && m-3==0x30
        && m-4==0x30 && escapekey==0)
        Display2("TIME TO CHANGE FILTER$");
        m=m+5;
        n=eeprom_read(m);
        while(n==0x30)
        {

```

```

        eeprom_write(m,0x39);
        m--;
        n=eeprom_read(m);
    }
    eeprom_write(m,n--);
    if(m==0x30 && m-1==0x30 && m-2==0x30 && m-3==0x30
    && m-4==0x30 && escapekey==0)
    Display2("TIME TO CHANGE DESSICANT $");
    m=m+5;
    n=eeprom_read(m);
    while(n==0x30)
    {
        eeprom_write(m,0x39);
        m--;
        n=eeprom_read(m);
    }
    eeprom_write(m,n--);
    if(m==0x30 && m-1==0x30 && m-2==0x30 && m-3==0x30
    && m-4==0x30 && escapekey==0)
    Display2("TIME TO CHANGE SEAL$");
}
break
}
asm("return")
}

```

Appendix

28/40-Pin 8-Bit CMOS FLASH Microcontrollers

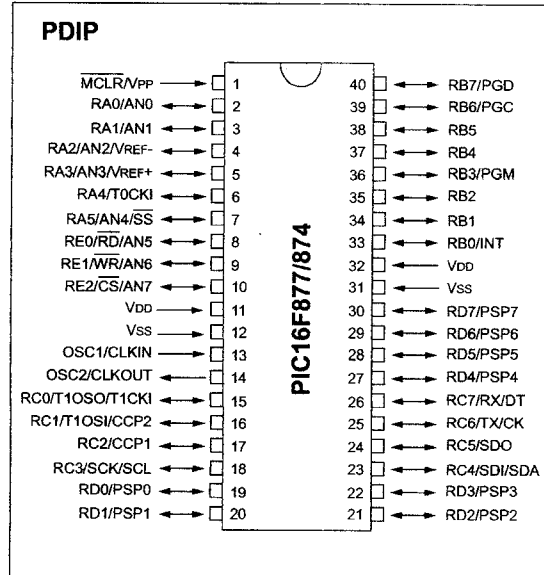
Devices Included in this Data Sheet:

- PIC16F873
- PIC16F876
- PIC16F874
- PIC16F877

Microcontroller Core Features:

- High performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,
Up to 368 x 8 bytes of Data Memory (RAM)
Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and
Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC
oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low power, high speed CMOS FLASH/EEPROM
technology
- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two
pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial, Industrial and Extended temperature
ranges
- Low-power consumption:
 - < 0.6 mA typical @ 3V, 4 MHz
 - 20 µA typical @ 3V, 32 kHz
 - < 1 µA typical standby current

Pin Diagram



Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler,
can be incremented during SLEEP via external
crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period
register, prescaler and postscaler
- Two Capture, Compare, PWM modules
 - Capture is 16-bit, max. resolution is 12.5 ns
 - Compare is 16-bit, max. resolution is 200 ns
 - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master
mode) and I²C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver
Transmitter (USART/SCI) with 9-bit address
detection
- Parallel Slave Port (PSP) 8-bits wide, with
external RD, WR and CS controls (40/44-pin only)
- Brown-out detection circuitry for
Brown-out Reset (BOR)

PIC16F87X

TABLE 3-3: PORTB FUNCTIONS

Name	Bit#	Buffer	Function
RB0/INT	bit0	TTL/ST ⁽¹⁾	Input/output pin or external interrupt input. Internal software programmable weak pull-up.
RB1	bit1	TTL	Input/output pin. Internal software programmable weak pull-up.
RB2	bit2	TTL	Input/output pin. Internal software programmable weak pull-up.
RB3/PGM ⁽³⁾	bit3	TTL	Input/output pin or programming pin in LVP mode. Internal software programmable weak pull-up.
RB4	bit4	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB5	bit5	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB6/PGC	bit6	TTL/ST ⁽²⁾	Input/output pin (with interrupt-on-change) or In-Circuit Debugger pin. Internal software programmable weak pull-up. Serial programming clock.
RB7/PGD	bit7	TTL/ST ⁽²⁾	Input/output pin (with interrupt-on-change) or In-Circuit Debugger pin. Internal software programmable weak pull-up. Serial programming data.

Legend: TTL = TTL input, ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.

2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.

3: Low Voltage ICSP Programming (LVP) is enabled by default, which disables the RB3 I/O function. LVP must be disabled to enable RB3 as an I/O pin and allow maximum compatibility to the other 28-pin and 40-pin mid-range devices.

TABLE 3-4: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
06h, 106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h, 186h	TRISB	PORTB Data Direction Register								1111 1111	1111 1111
81h, 181h	OPTION_REG	RBP	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

PIC16F87X

TABLE 3-5: PORTC FUNCTIONS

Name	Bit#	Buffer Type	Function
RC0/T1OSO/T1CKI	bit0	ST	Input/output port pin or Timer1 oscillator output/Timer1 clock input.
RC1/T1OSI/CCP2	bit1	ST	Input/output port pin or Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	bit2	ST	Input/output port pin or Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	bit3	ST	RC3 can also be the synchronous serial clock for both SPI and I ² C modes.
RC4/SDI/SDA	bit4	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I ² C mode).
RC5/SDO	bit5	ST	Input/output port pin or Synchronous Serial Port data output.
RC6/TX/CK	bit6	ST	Input/output port pin or USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	bit7	ST	Input/output port pin or USART Asynchronous Receive or Synchronous Data.

Legend: ST = Schmitt Trigger input

TABLE 3-6: SUMMARY OF REGISTERS ASSOCIATED WITH PORTC

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged

3.4 PORTD and TRISD Registers

PORTD and TRISD are not implemented on the PIC16F873 or PIC16F876.

PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output.

PORTD can be configured as an 8-bit wide microprocessor port (parallel slave port) by setting control bit PSPMODE (TRISE<4>). In this mode, the input buffers are TTL.

FIGURE 3-7: PORTD BLOCK DIAGRAM (IN I/O PORT MODE)

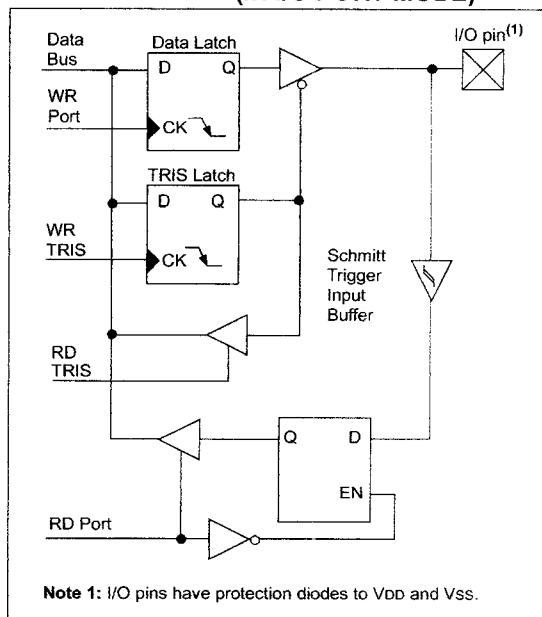


TABLE 3-7: PORTD FUNCTIONS

Name	Bit#	Buffer Type	Function
RD0/PSP0	bit0	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit0.
RD1/PSP1	bit1	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit1.
RD2/PSP2	bit2	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit2.
RD3/PSP3	bit3	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit3.
RD4/PSP4	bit4	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit4.
RD5/PSP5	bit5	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit5.
RD6/PSP6	bit6	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit6.
RD7/PSP7	bit7	ST/TTL ⁽¹⁾	Input/output port pin or parallel slave port bit7.

Legend: ST = Schmitt Trigger input, TTL = TTL input

Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

TABLE 3-8: SUMMARY OF REGISTERS ASSOCIATED WITH PORTD

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
08h	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
88h	TRISD	PORTD Data Direction Register								1111 1111	1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTD.

PIC16F87X

2.2.2.1 STATUS Register

The STATUS register contains the arithmetic status of the ALU, the RESET status and the bank select bits for data memory.

The STATUS register can be the destination for any instruction, as with any other register. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to the device logic. Furthermore, the TO and PD bits are not writable, therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, CLRFS STATUS will clear the upper three bits and set the Z bit. This leaves the STATUS register as 000u u1uu (where u = unchanged).

It is recommended, therefore, that only BCF, BSF, SWAPF and MOVWF instructions are used to alter the STATUS register, because these instructions do not affect the Z, C or DC bits from the STATUS register. For other instructions not affecting any status bits, see the "Instruction Set Summary."

Note: The C and DC bits operate as a borrow and digit borrow bit, respectively, in subtraction. See the SUBLW and SUBWF instructions for examples.

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

	R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
	IRP	RP1	RP0	TO	PD	Z	DC	C
bit 7								bit 0

- bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h - 1FFh)
0 = Bank 0, 1 (00h - FFh)
- bit 6-5 **RP1:RP0:** Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h - 1FFh)
10 = Bank 2 (100h - 17Fh)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)
Each bank is 128 bytes
- bit 4 **TO:** Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred
- bit 3 **PD:** Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction
- bit 2 **Z:** Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero
- bit 1 **DC:** Digit carry/borrow bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
(for borrow, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result
- bit 0 **C:** Carry/borrow bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: For borrow, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high, or low order bit of the source register.

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

PIC16F87X

2.2.2.3 INTCON Register

The INTCON Register is a readable and writable register, which contains various enable and flag bits for the TMR0 register overflow, RB Port change and External RB0/INT pin interrupts.

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>). User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt.

REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

bit 7

bit 0

- bit 7 **GIE:** Global Interrupt Enable bit
 1 = Enables all unmasked interrupts
 0 = Disables all interrupts
- bit 6 **PEIE:** Peripheral Interrupt Enable bit
 1 = Enables all unmasked peripheral interrupts
 0 = Disables all peripheral interrupts
- bit 5 **TOIE:** TMR0 Overflow Interrupt Enable bit
 1 = Enables the TMR0 interrupt
 0 = Disables the TMR0 interrupt
- bit 4 **INTE:** RB0/INT External Interrupt Enable bit
 1 = Enables the RB0/INT external interrupt
 0 = Disables the RB0/INT external interrupt
- bit 3 **RBIE:** RB Port Change Interrupt Enable bit
 1 = Enables the RB port change interrupt
 0 = Disables the RB port change interrupt
- bit 2 **TOIF:** TMR0 Overflow Interrupt Flag bit
 1 = TMR0 register has overflowed (must be cleared in software)
 0 = TMR0 register did not overflow
- bit 1 **INTF:** RB0/INT External Interrupt Flag bit
 1 = The RB0/INT external interrupt occurred (must be cleared in software)
 0 = The RB0/INT external interrupt did not occur
- bit 0 **RBIF:** RB Port Change Interrupt Flag bit
 1 = At least one of the RB7:RB4 pins changed state; a mismatch condition will continue to set the bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared (must be cleared in software).
 0 = None of the RB7:RB4 pins have changed state

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

2.2.2.4 PIE1 Register

The PIE1 register contains the individual enable bits for the peripheral interrupts.

Note: Bit PEIE (INTCON<6>) must be set to enable any peripheral interrupt.

REGISTER 2-4: PIE1 REGISTER (ADDRESS 8Ch)

	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	PSPIE ⁽¹⁾	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7								bit 0

- bit 7 **PSPIE⁽¹⁾:** Parallel Slave Port Read/Write Interrupt Enable bit
 1 = Enables the PSP read/write interrupt
 0 = Disables the PSP read/write interrupt
- bit 6 **ADIE:** A/D Converter Interrupt Enable bit
 1 = Enables the A/D converter interrupt
 0 = Disables the A/D converter interrupt
- bit 5 **RCIE:** USART Receive Interrupt Enable bit
 1 = Enables the USART receive interrupt
 0 = Disables the USART receive interrupt
- bit 4 **TXIE:** USART Transmit Interrupt Enable bit
 1 = Enables the USART transmit interrupt
 0 = Disables the USART transmit interrupt
- bit 3 **SSPIE:** Synchronous Serial Port Interrupt Enable bit
 1 = Enables the SSP interrupt
 0 = Disables the SSP interrupt
- bit 2 **CCP1IE:** CCP1 Interrupt Enable bit
 1 = Enables the CCP1 interrupt
 0 = Disables the CCP1 interrupt
- bit 1 **TMR2IE:** TMR2 to PR2 Match Interrupt Enable bit
 1 = Enables the TMR2 to PR2 match interrupt
 0 = Disables the TMR2 to PR2 match interrupt
- bit 0 **TMR1IE:** TMR1 Overflow Interrupt Enable bit
 1 = Enables the TMR1 overflow interrupt
 0 = Disables the TMR1 overflow interrupt

Note 1: PSPIE is reserved on PIC16F873/876 devices; always maintain this bit clear.

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

4.2 Reading the EEPROM Data Memory

Reading EEPROM data memory only requires that the desired address to access be written to the EEADR register and clear the EEPGD bit. After the RD bit is set, data will be available in the EEDATA register on the very next instruction cycle. EEDATA will hold this value until another read operation is initiated or until it is written by firmware.

The steps to reading the EEPROM data memory are:

1. Write the address to EEDATA. Make sure that the address is not larger than the memory size of the PIC16F87X device.
2. Clear the EEPGD bit to point to EEPROM data memory.
3. Set the RD bit to start the read operation.
4. Read the data from the EEDATA register.

EXAMPLE 4-1: EEPROM DATA READ

```
BSF STATUS, RP1 ;
BCF STATUS, RP0 ;Bank 2
MOVF ADDR, W ;Write address
MOVWF EEADR ;to read from
BSF STATUS, RP0 ;Bank 3
BCF EECON1, EEPGD ;Point to Data memory
BSF EECON1, RD ;Start read operation
BCF STATUS, RP0 ;Bank 2
MOVF EEDATA, W ;W = EEDATA
```

4.3 Writing to the EEPROM Data Memory

There are many steps in writing to the EEPROM data memory. Both address and data values must be written to the SFRs. The EEPGD bit must be cleared, and the WREN bit must be set, to enable writes. The WREN bit should be kept clear at all times, except when writing to the EEPROM data. The WR bit can only be set if the WREN bit was set in a previous operation, i.e., they both cannot be set in the same operation. The WREN bit should then be cleared by firmware after the write. Clearing the WREN bit before the write actually completes will not terminate the write in progress.

Writes to EEPROM data memory must also be preceded with a special sequence of instructions, that prevent inadvertent write operations. This is a sequence of five instructions that must be executed without interruptions. The firmware should verify that a write is not in progress, before starting another cycle.

The steps to write to EEPROM data memory are:

1. If step 10 is not implemented, check the WR bit to see if a write is in progress.
2. Write the address to EEADR. Make sure that the address is not larger than the memory size of the PIC16F87X device.
3. Write the 8-bit data value to be programmed in the EEDATA register.
4. Clear the EEPGD bit to point to EEPROM data memory.
5. Set the WREN bit to enable program operations.
6. Disable interrupts (if enabled).
7. Execute the special five instruction sequence:
 - Write 55h to EECON2 in two steps (first to W, then to EECON2)
 - Write AAh to EECON2 in two steps (first to W, then to EECON2)
 - Set the WR bit
8. Enable interrupts (if using interrupts).
9. Clear the WREN bit to disable program operations.
10. At the completion of the write cycle, the WR bit is cleared and the EEIF interrupt flag bit is set. (EEIF must be cleared by firmware.) If step 1 is not implemented, then firmware should check for EEIF to be set, or WR to clear, to indicate the end of the program cycle.

EXAMPLE 4-2: EEPROM DATA WRITE

```
BSF STATUS, RP1 ;
BSF STATUS, RP0 ;Bank 3
BTFSC EECON1, WR ;Wait for
GOTO $-1 ;write to finish
BCF STATUS, RP0 ;Bank 2
MOVF ADDR, W ;Address to
MOVWF EEADR ;write to
MOVF VALUE, W ;Data to
MOVWF EEDATA ;write
BSF STATUS, RP0 ;Bank 3
BCF EECON1, EEPGD ;Point to Data memory
BSF EECON1, WREN ;Enable writes
;Only disable interrupts
BCF INTCON, GIE ;if already enabled,
;otherwise discard
MOVLW 0x55 ;Write 55h to
MOVWF EECON2 ;EECON2
MOVLW 0xAA ;Write AAh to
MOVWF EECON2 ;EECON2
BSF EECON1, WR ;Start write operation
;Only enable interrupts
BSF INTCON, GIE ;if using interrupts,
;otherwise discard
BCF EECON1, WREN ;Disable writes
```

INSTRUCTION SET

INSTRUCTION	CODE										DESCRIPTION	TYPICAL EXECUTION TIME
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears display and returns the cursor to home position (Address 0). Sets VD = 1 of Entry Mode.	1.64µs
Return home	0	0	0	0	0	0	0	0	1	*	Return the cursor to the home position (Address 0). Also returns the display being shifted to the original position. DD RAM contents remain unchanged. Set DD RAM addresses to zero.	1.64µs
Entry mode set	0	0	0	0	0	0	0	1	VD	S	Set the cursor move direction and specifies or not to shift the display. These operations are performed during data write and read of DD RAM/CG RAM, FOR NORMAL OPERATION SET S TO 0	40µs
Display On/Off control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF all display (D); cursor ON/OFF (C), and blink of cursor position character (B).	40µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing DD RAM contents.	40µs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL) number of display lines (N) and character font (F).	40µs
Set the CG RAM address	0	0	0	1	MSB			ACG		LSB	Sets the CG RAM address. CG RAM data is sent and received after this setting.	40µs
Set the DD RAM address	0	0	1	MSB			ADD		LSB	Sets the CG RAM address. CG RAM data is sent and received after this setting.	40µs	
Read busy flag & address	0	1	BF	MSB			AC		LSB	Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	40µs	
Write data to CG of DD RAM	1	0	MSB							LSB	Writes data into DD RAM of CG RAM.	40µs
Read data from CG or DD RAM	1	1	MSB							LSB	Reads data into DD RAM of CG RAM.	40µs

- S = 1 : Accompanies display shift when data is written. For normal operation, set to 0
- VD=1 : Increment DL=1 : 8 bits
- VD = 0 : Decrement DL=0 : 4 bits
- S/C=1 : Display shift N=1 : 2 (1) line
- S/C=0 : Cursor Move N=0 : 1 line
- R/L=1 : Shift to the right F=1 : 5x10 dots
- R/L=0 : Shift to the left F=0 : 5x7 dots
- BF = 1 : Internally operating
- BF = 0 : Can accept instruction

- DD RAM : Display data RAM
- CG RAM : Character generator RAM
- ACG : CG RAM address
- ADD : DD RAM address corresponds to cursor address
- AC : Address counter used for both DD and CG RAM address
- B : 1=ON 0=OFF (Blinking Cursor)
- C : 1=ON 0=OFF (Cursor)
- D : 1=ON 0=OFF (Display)
- * Don't Care

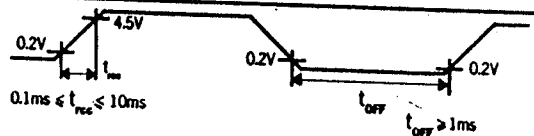
INITIALIZATION

The module automatically performed initialization when powered on (using internal reset circuit). The following instructions are executed during initialization :-

- CLEAR DISPLAY**
The Busy Flag is kept in the Busy State (BF=1) unit initialization ends. The time is 15ms.
- Function Set**
DL = 1 : 8-bits long interface data
N = 0 : 1 Line Display
- DISPLAY ON/OFF CONTROL**
D = 0 : Display OFF
C = 0 : Cursor OFF
B = 0 : Blink OFF
- ENTRY MODE SET**
VD = 1 : +1 (INCREMENT)
S = 0 : NO SHIFT
- DD RAM IS SELECTED**

Power On Initialization depends on rise time of the supply when it is turned on. The following time relationship must be satisfied.

ITEM	SYMBOL	STANDARD TIME			UNIT
		MIN	TYP	MAX	
Power Supply Rise Time	t_{rc}	0.1	-	10	ms
Power Supply Off Time	t_{off}	1.0	-	-	ms



POWER ON TIMING DIAGRAM

NOTE :

When the above power supply condition is not satisfied, the internal reset circuitry does not operate correctly. In this case, perform the needed initialization by sending function set instructions thrice from MPU after turning the power on. For Example, to designate a 8-bits data length, send the following instructions thrice.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	*	*	*	*
0	0	0	0	1	1	*	*	*	*
0	0	0	0	1	1	*	*	*	*

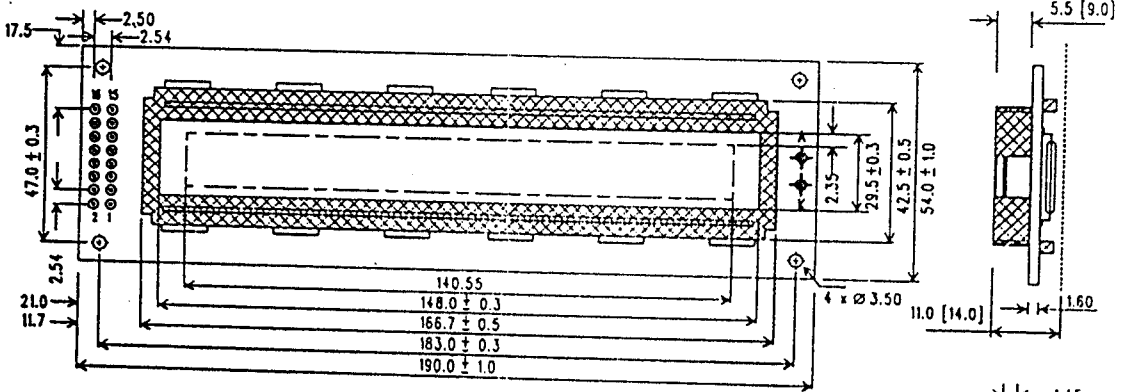
When this ends, the module enters 8-bits data length mode without fail. then enter 4-bits data length instruction for 4-bits data length interface.

LAMPEX

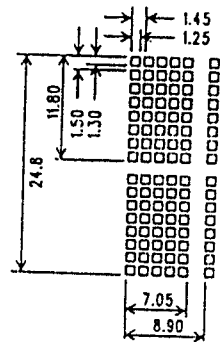
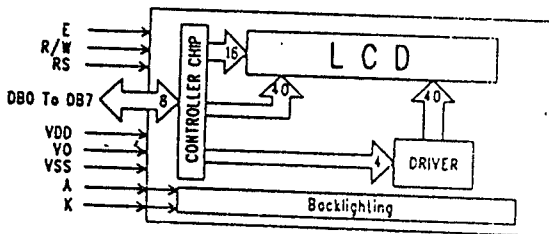
ALPHANUMERIC DOT MATRIX LCM

LM16209 [16 x 2, 1/16 MUX]

DIMENSION IN MM



BLOCK DIAGRAM



① DIMENSIONS FOR LED BACKLIGHT VERSION.

TABULATED DATA

MODEL	DISPLAY FORMAT CHAR. x LINE	FONT MATRIX W x H	DRIVING DUTY	DOT SIZE W x H	CHARACTER SIZE W x H	VIEWING AREA W x H	PCB SIZE W x H
LM16209	16 x 2	5 x 8	1/16	1.25 x 1.30	7.05 x 11.80	147.0 x 29.50	190.0 x 54.0

ELECTRICAL DATA (Ta = 25°C, Vdd = 5.0V, I/O 25V)

ITEM	SYMBOL	TEST CONDITION	STANDARD VALUES			UNITS
			Min.	Typ.	Max.	
Input High Voltage	V _{IH}	-	2.2	-	-	V
Input Low Voltage	V _{IL}	-	-0.3	-	V _{dd}	V
Output High Voltage	V _{OH}	-I _{OH} = 0.2mA	2.4	-	0.6	V
Output Low Voltage	V _{OL}	-I _{OL} = 1.2mA	-	-	-	V
Supply Current	I _{dd}	V _{dd} = 5.0V	-	2.8	0.4	mA
Operating Voltage for LCD	V _{dd} - V _o	Ta = 0°C	-	4.7	-	V
		Ta = 25°C	-	4.6	-	V
LED Backlight Current	I _{LED}	V _{dd} = 12.0V	80	100	120	mA

PIN DETAILS

PIN No.	SYMBOL	PIN No.	SYMBOL
1	Gnd	6	E
2	Vcc	7 to 14	DB0 to DB7
3	Vo	15	BL (K)
4	RS	16	RL (A)

OPTIONAL DETAILS

Wide configurations available. Kindly consult LAMPEX while ordering. Refer LAMPEX LCM Catalogue for more information.

LAMPEX reserves all right. Specification might alter without prior notice.

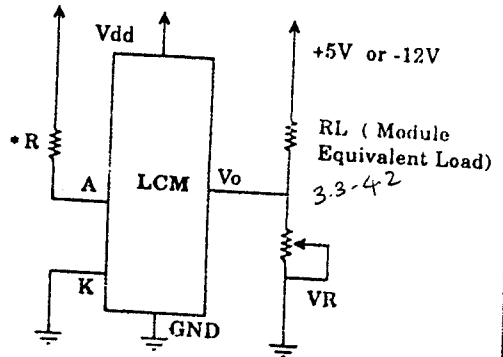
DEFINITION OF TERMINALS

PIN NO.	SYMBOL	FUNCTION
(13)	Vss	Ground terminal of Module.
(14)	Vdd	Supply terminal of Module, +5v.
(12)	Vo	Power Supply for liquid Crystal Drive
(11)	RS	Register Select
(10)	R/W	RS = \emptyset ... Instruction Register. RS = 1 ... Data Register. Read / Write R / W = 1 ... Read R / W = \emptyset ... Write
(9)	E (E1)	Enable
14. (8-1)	DB \emptyset - DB7	Bi-directional Data Bus. Data Transfer is performed once, thru DB \emptyset -DB7, in the case of interface data length is 8-bits; and twice, thru DB4-DB7 in the case of interface data length is 4-bits. Upper four bits first then lower four bits.
(17)	LAMP- (L-)	LED or EL lamp power supply terminals
(18)	LAMP+ (L+)	
(15)	(E2)	

Note: () Pin Nos for model LM40400

POWER SUPPLY REQUIREMENTS

- Wide Temperature Range Version
- Standard
- Super - Twist Display Version



When $RL=23.5K$ - $VR=10-20K$, $RL=5K$ - $VR=2-5K$.
This circuit shows the typical power supply connection for all dot matrix modules. The display Voltage (V_{LCD}) is slightly different for different version (eg. standard, wide temp and supertwist.) Recommend end user to use VARIABLE RESISTOR as shown in the circuit for optimum V_{LCD} ($V_{dd}-V_o$) adjustment to obtain best display contrast and viewing angle. *R Value see Note 4.

OPERATING SPECIFICATIONS

	STANDARD TEMP	WIDE TEMP
Operating temperature range	-10° to +55°C	-20°C to +70°C
Storage temperature range	-20°C to +70°C	-30°C to +80°C
Operating relative humidity	90% MAX	90% MAX

ELECTRICAL CHARACTERISTICS

(Ta = +25°C)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNIT
Supply Voltage	VDD		4.5	5.0	5.5	V
LC/D Drive Voltage	VDD-Vo (V _{LCD})		3.3	4.2*	-	V
Normal Temp Model		3.7	4.3*	-	V	
Wide Temp Model		3.5	4.1*	-	V	
Supply Current ¹	IDD	VDD = 5V Vo = 0V MIN	-	1.0	2.0	mA
16x1			-	1.0	3.0	mA
16x2			-	1.5	3.0	mA
20x1, 20x2, 24x2			-	2.5	4.0	mA
20x1, 40x2, 40x1			-	4.5	5.5	mA
16x4						
Input Voltage ²	VIL		0	-	0.6	V
	VIH		2.2	-	VDD	V
Output Voltage ²	VOL	IOL = 1.6 mA	-	-	0.6	V
	VOH	IOH = 0.2 mA	2.4	-	-	V
LED Current ⁴	I _{LED}	+L to -L = 5V	-	40	60	mA
6x1, 16x2			-	60	80	mA
4x2, 20x2			-	60	80	mA
0x1, 40x1, 40x2, 20x4			-	150	250	mA

Note:

Applies to DB \emptyset - DB7, E, RS and R/W

Applies to DB \emptyset - DB7.

Supply Current may slightly exceed Max. Rating if Samsung Controller is used

LED Backlight Current (Intensity) is controlled

CHARACTER CODE MAP

		Higher 4 Bit (D4 to D7) of Character Code (Hexadecimal)																		
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F			
Lower 4 Bit (D0 to D3) of Character Code (Hexadecimal)	0	CG RAM (1)			0	A	P	`	P					一	タ	三	0	P		
	1	CG RAM (2)		!	1	A	Q	a	q					。	フ	チ	4	a	q	
	2	CG RAM (3)		"	2	B	R	b	r					フ	イ	ツ	×	P	B	
	3	CG RAM (4)		#	3	C	S	c	s					」	ウ	テ	E	c	s	
	4	CG RAM (5)		\$	4	D	T	d	t					、	イ	ト	ト	T	D	
	5	CG RAM (6)		%	5	E	U	e	u					。	オ	ナ	1	E	U	
	6	CG RAM (7)		&	6	F	V	f	v					ヲ	カ	ニ	ヨ	F	V	
	7	CG RAM (8)		'	7	G	W	g	w					フ	キ	ヲ	ウ	G	W	
	8	CG RAM (1)		(8	H	X	h	x					イ	ウ	ホ	リ	H	X	
	9	CG RAM (2))	9	I	Y	i	y					ウ	ト	ル	ル	I	Y	
	A	CG RAM (3)		*	:	J	Z	j	z					エ	コ	ン	ル	J	Z	
	B	CG RAM (4)		+	;	K	L	k	l					オ	サ	ヒ	ロ	K	L	
	C	CG RAM (5)		,	<	L	M	l	m					カ	シ	フ	フ	L	M	
	D	CG RAM (6)		-	=	N	O	n	o					ユ	ズ	ン	ニ	N	O	
	E	CG RAM (7)		.	>	N	O	n	o					ヨ	セ	ホ	ニ	N	O	
	F	CG RAM (8)		/	?	O	L	O	+					ウ	リ	マ	ニ	O	L	O

NOTE: Custom Font ROM Mask Can Be Tooled On Special Request.

DOT CHARACTER PATTERNS

For 5x7 Dot Character Patterns

Character Codes (DD RAM Data)		CG RAM Address		Character Patterns (DD RAM Data)	
7 6 5 4 3 2 1 0		5 4 3 2 1 0		7 6 5 4 3 2 1 0	
Higher	Lower	Higher	Lower	Higher	Lower
0000	0001	000	000		
0000	1111	001	010		
0000	1111	100	101		
0000	1111	111	111		

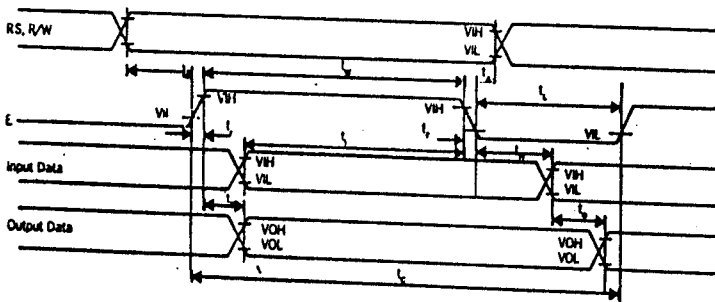
NOTE:
Character code bits 0,2
correspond to CG RAM address bits 3,5
(3 bits : 8 types)

For 5x10 Dot Character Patterns

Character Codes (DD RAM Data)		CG RAM Address		Character Patterns (DD RAM Data)	
7 6 5 4 3 2 1 0		5 4 3 2 1 0		7 6 5 4 3 2 1 0	
Higher	Lower	Higher	Lower	Higher	Lower
0000	0000	000	000		
0000	1111	101	110		
0000	1111	110	110		
0000	1111	111	111		

NOTE:
Character code bits 1,2
correspond to CG RAM address bits 4,5
(3 bits : 8 types)

TIMING DIAGRAM



TIMING CHARACTERISTICS FOR ALL COMPATIBLE CONTROLLER CHIPS.

CONTROLLERS CHIPS		SAMSUNG	HITACHI	SANYO	EPSON	OKI	RECOMMENDED	UNT
PARAMETERS		KS0066	HD44780	LC7985NA	SED1278	MSM6222	TIMING	
Enable Cycle Time	tC (min)	1000	1000	1000	500	667	1000	nS
Enable Pulse Width								
High Level	tW (min)	450	450	450	220	280	450	nS
Low Level	tL (min)	450	450	450	220	280	450	nS
E Raise Time	t _r (max)	25	25	25	25	25	25	nS
E Fall Time	t _f (max)	25	25	25	25	25	25	nS
Set-up Time	t _B (min)	140	140	140	40	140	25	nS
Data Set-up Time	t _I (min)	195	195	195	60	180	140	nS
Data Delay Time	t _D (max)	320	320	320	120	220	195	nS
Address Hold Time	t _A (max)	10	10	10	10	10	320	nS
Hold Time							10	nS
Input Data	t _H (min)	10	10	10	10	10	10	nS
Output Data	t _O (min)	20	20	20	20	20	20	nS

NOTE:

1. [Redacted]
2. MODULE INITIALIZATION DOES NOT AFFECT BY USING HD44100, OR KS0065, OR LC7930, OR MSM5259, OR MSM5839, OR MSM5260 DRIVER CHIPS.

INITIALIZATION

For 8 bit data interfacing

Power On

Wait for 15ms or more
after Vcc rises to 4.5V

BF cannot be checked at this time

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	X	X	X	X

Function Set : DL = 1, 8 bit interface data.
DL must set at H during this initialization.

Wait for 4.1 ms or more

BF cannot be checked at this time

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	X	X	X	X

Function Set : DL = 1, 8 bit interface data.
DL must set at H during this initialization.

Wait for 100 μs or more

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	X	X	X	X

Function Set : DL = 1, 8 bit interface data.
DL must set at H during this initialization.

BF can be checked at this
time, check for not busy

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	N	F	X	X

Function Set : DL = 1, 8 bit interface data.
N = no. of line
F = character font

check for not busy

0	0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---

Display off

check for not busy

0	0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---

clears all display and return cursor to home position

check for not busy

0	0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---

Set the shift mode

End of initialization

*Note : IN NORMAL OPERATION, SET S TO 0

For 4 bit data interfacing

Power On

Wait for 15ms or more
after Vcc rises to 4.5V

BF cannot be checked at this time

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	1	1

Function Set : DL = 1, 8 bit interface data.
DL must set at H during this initialization.

Wait for 4.1 ms or more

BF cannot be checked at this time

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	1	1

Function Set : DL = 1, 8 bit interface data.
DL must set at H during this initialization.

Wait for 100 μs or more

BF can be checked at this time

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	1	1

Function Set : DL = 1, 8 bit interface data.
DL must set at H during this initialization.

check for not busy

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	1	0

Function Set : DL = 0, 4 bit interface data.

check for not busy

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	1	0
0	0	N	F	X	X

Function Set : DL = 0, 4 bit interface data.
N = no. of line
F = character font

check for not busy

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	0	0
0	0	1	0	0	0

Display off

check for not busy

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	0	0
0	0	0	0	0	1

Display on

check for not busy

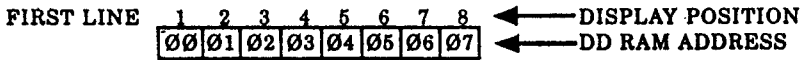
RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	0	0
0	0	0	1	I/D	S

Entry mode Set

End of initialization

DISPLAY CHARACTER POSITION AND DD RAM ADDRESS

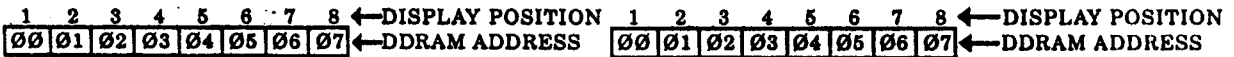
8x1 DMM, 1/8 MUX
 N = 0 : 1-LINE DISPLAY
 F = 0 : 5x7 DOTS



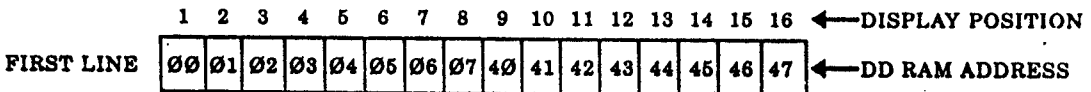
WHEN THE DISPLAY SHIFT OPERATION IS PERFORMED, THE DD RAM ADDRESS MOVES AS FOLLOW :

AFTER THE LEFT SHIFT INSTRUCTION

AFTER THE RIGHTSHIFT INSTRUCTION

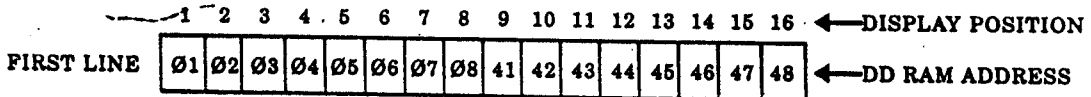


16x1 DMM, 1/16 MUX
 N = 1 : 2-LINE DISPLAY
 F = 0 : 5x7 DOTS

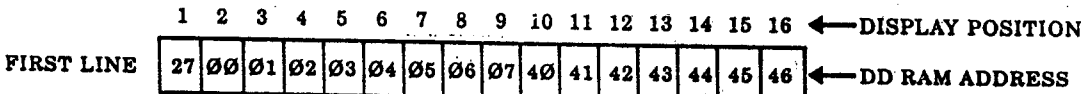


WHEN THE DISPLAY SHIFT OPERATION IS PERFORMED, THE DD RAM ADDRESS MOVES AS FOLLOW :

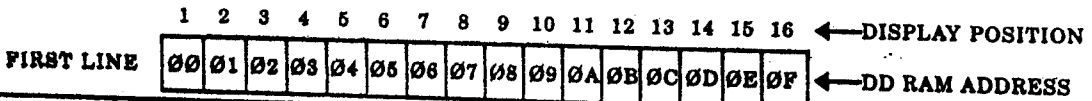
AFTER THE LEFT SHIFT INSTRUCTION



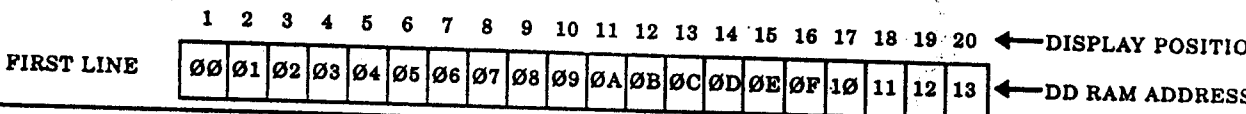
AFTER THE RIGHT SHIFT INSTRUCTION



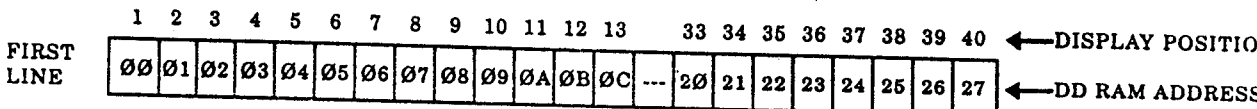
16x1 DMM, 1/8 MUX
 N = 0 : 1-LINE DISPLAY
 F = 0 : 5x7 DOTS



20x1 DMM, 1/8 MUX
 N = 0 : 1-LINE DISPLAY
 F = 0 : 5x7 DOTS



40x1 DMM, 1/8 MUX
 N = 0 : 1-LINE DISPLAY
 F = 0 : 5x7 DOTS



LAMPEx

ALPHANUMERIC DOT MATRIX MODULES

DISPLAY CHARACTER POSITION AND DD RAM ADDRESS (CONTINUE)

16x2 DMM, 1/16 MUX

N=1 : 2-LINE DISPLAY F=Ø : 5x7 DOTS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	← DISPLAY POSITION
FIRST LINE	ØØ	Ø1	Ø2	Ø3	Ø4	Ø5	Ø6	Ø7	Ø8	Ø9	ØA	ØB	ØC	ØD	ØE	ØF	← DD RAM ADDRESS
SECOND LINE	4Ø	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	

WHEN THE DISPLAY SHIFT OPERATION IS PERFORMED, THE DD RAM ADDRESS MOVES AS FOLLOW :

AFTER THE LEFT SHIFT INSTRUCTION

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	← DISPLAY POSITION
FIRST LINE	Ø1	Ø2	Ø3	Ø4	Ø5	Ø6	Ø7	Ø8	Ø9	ØA	ØB	ØC	ØD	ØE	ØF	1Ø	← DD RAM ADDRESS
SECOND LINE	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	5Ø	

AFTER THE RIGHT SHIFT INSTRUCTION

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	← DISPLAY POSITION
FIRST LINE	27	ØØ	Ø1	Ø2	Ø3	Ø4	Ø5	Ø6	Ø7	Ø8	Ø9	ØA	ØB	ØC	ØD	ØE	← DD RAM ADDRESS
SECOND LINE	67	4Ø	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	

20x2 DMM, 1/16 MUX

N=1 : 2-LINE DISPLAY

F=Ø : 5x7 DOTS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	← DISPLAY POSITION
FIRST LINE	ØØ	Ø1	Ø2	Ø3	Ø4	Ø5	Ø6	Ø7	Ø8	Ø9	ØA	ØB	ØC	ØD	ØE	ØF	1Ø	11	12	13	← DD RAM ADDRESS
SECOND LINE	4Ø	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	5Ø	51	52	53	

24x2 DMM, 1/16 MUX

N=1 : 2-LINE DISPLAY

F=Ø : 5x7 DOTS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	← DISPLAY POSITION
FIRST LINE	ØØ	Ø1	Ø2	Ø3	Ø4	Ø5	Ø6	Ø7	Ø8	Ø9	ØA	ØB	ØC	ØD	ØE	ØF	1Ø	11	12	13	14	15	16	17	← DD RAM ADDRESS
SECOND LINE	4Ø	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	5Ø	51	52	53	54	55	56	57	

40x2 DMM, 1/16 MUX

N=1 : 2-LINE DISPLAY

F=Ø : 5x7 DOTS

	1	2	3	4	5	6	7	8	9	10	11	12	13											← DISPLAY POSITION								
FIRST LINE	ØØ	Ø1	Ø2	Ø3	Ø4	Ø5	Ø6	Ø7	Ø8	Ø9	ØA	ØB	ØC											33	34	35	36	37	38	39	40	← DD RAM ADDRESS
SECOND LINE	4Ø	41	42	43	44	45	46	47	48	49	4A	4B	4C											6Ø	61	62	63	64	65	66	67	

16x4 DMM, 1/16 MUX

N=1 : 2-LINE DISPLAY

F=Ø : 5x7 DOTS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	← DISPLAY POSITION
FIRST LINE	ØØ	Ø1	Ø2	Ø3	Ø4	Ø5	Ø6	Ø7	Ø8	Ø9	ØA	ØB	ØC	ØD	ØE	ØF	← DD RAM ADDRESS
SECOND LINE	4Ø	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	
THIRD LINE	1Ø	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	
FOURTH LINE	5Ø	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	

20x4 DMM, 1/16 MUX

N=1 : 2-LINE DISPLAY

F=Ø : 5x7 DOTS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	← DISPLAY POSITION
FIRST LINE	ØØ	Ø1	Ø2	Ø3	Ø4	Ø5	Ø6	Ø7	Ø8	Ø9	ØA	ØB	ØC	ØD	ØE	ØF	1Ø	11	12	13	← DD RAM ADDRESS
SECOND LINE	4Ø	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	5Ø	51	52	53	
THIRD LINE	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	2Ø	21	22	23	24	25	26	27	
FOURTH LINE	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	6Ø	61	62	63	64	65	66	67	