PC-BASED SCROLLING MESSAGE DISPLAY

■ SURESH KUMAR

Controlling electronic devices from a PC is fun. Here is a scrolling message display that makes use of the PC's parallel port. The message typed from the keyboard of the PC is displayed on the light-emitting diodes arranged as 5×7 dot-matrix display in moving message format.

LED-based scrolling message displays are increasingly being used at railway stations, public places, colleges, universities, hospitals, general stores, etc for disseminating information. However, most displays lack in storage capacity and cannot display a large number of characters at a time.

This PC-based LED scrolling message display has the following features:

1. The message to be displayed is stored in a file and the message length to be displayed is limited only by free memory space on the hard disk of the computer.
2. The number of characters displayed at a time can be as high as 30.
3. The message stored in the file can be changed using any text editor including Notepad.
4. The running speed of the message displayed can be increased or decreased by pressing a few keys.

Here, the circuit is designed for displaying English characters on a 35 (5×7) LED dot-matrix display.

The PC's parallel port (LPT port) is used to output the display code and the clock signal for the scrolling message display.

The parallel port is terminated into a 25-pin D-type female connector at the back of the PC. IBM PCs usually come with one or two LPT ports. Each parallel port is actually made up of three ports, namely, data port, status port and control port. Here, only data port is used for this scrolling message display.

PARTS LIST

Semiconductors:

IC1	- 7805C 5V regulator
IC2-IC8	- 74174 hex D-type flip-flop
D1-D4	- 1N4007 rectifier diode
LED1-LED42	- Red LED
Resistors (all ¼-watt, ±5% carbon):	
R1- R42	- 150-ohm
Capacitors:	
C1	- 470µF, 16V electrolytic
Miscellaneous:	
X1	- 230V AC primary to 7.5V, 1A secondary transformer

Pins 2 through 9 form the 8-bit data output port. This is purely a write-only port, which means it can only output data. The base address of the first parallel port (LPT1) is '378H' or '888' (decimal).

Parallel-input parallel-output (PIPO) registers are used to shift the signal from right to left. The clock pulse and code signal are generated by the computer program and output from the parallel port (base address 0×378). Theoretically, we can add infinite number of PIPO registers but the maximum number of registers is actually limited to the current triggering value of the clock pulse. To add a large number of PIPO registers, amplify the clock pulse prior to connecting it to the PIPO ICs.

Circuit description

Fig. 1 shows the circuit for the scrolling message display. IC 74174 has been used as PIPO register, which comprises high-speed, hex D-type flip-flops. It is used as a 6-bit edge-triggered storage register. The data on the inputs of the flip-flop is transferred for storage during high-to-low transition of clock. Data lines D0 through D5 of the parallel port are connected to the input pins of the first flip-flop (IC2). The output of IC2 is fed to the next flip-flop IC input as well as LED. Data line D6 is fed to IC8, while data line D7 is connected to the clock inputs of IC2 through IC8. Clock pins of all the flip-flop ICs are connected together. Master reset pin 1 of all the flip-flops is connected to Vcc. Pins 18 through 25 of the parallel port are grounded. As data present on lines D0 through D6 shifts from the first stage to the next stage, and so on, the message appears as scrolling on the dot-matrix LED display.

The present circuit supports a display made of 42 LEDs comprising seven rows and six columns. Up to 30 such units can be added with no change in the circuit. However, to add these units, you need to amplify the clock pulse output. Note that each character is displayed in a matrix of 5 columns and 7 rows (explained later), hence the sixth-column LEDs form part of the next character (column 1).

Fig. 2 shows the power supply circuit. The AC mains is stepped down by transformer X1 to deliver a secondary output of 7.5V AC at 1A. The transformer output is rectified by a full-wave bridge rectifier comprising diodes D1 through D4, filtered by capacitor C1, then regulated by IC 7805C (IC1) to provide regulated 5V DC to the circuit.

An actual-size, single-side PCB for the circuits in Figs 1 and 2 is shown in Fig. 5 and its component layout in Fig. 6.

EFY note. Commercially 7×5 dot-matrix displays with discrete LEDs may not be easily available in the market, therefore a perforated board with holes for the LED leads may be used. The layout of such a board is shown in Fig. 7. The holes are used for passing the LED leads.

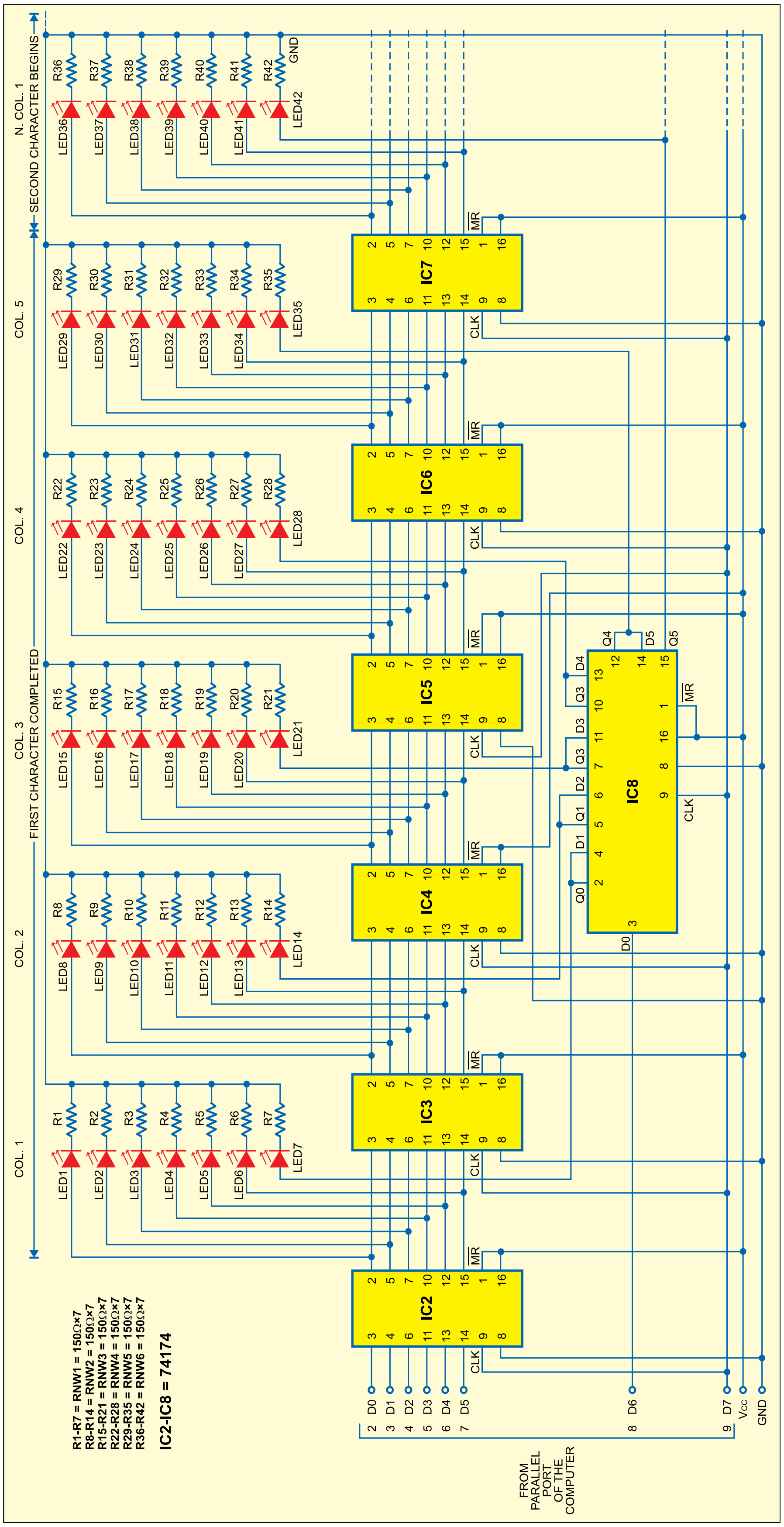


Fig. 1: Circuit of LED-based scrolling message display

The software

The software for the scrolling message display has been developed in 'C' language and compiled in 'Turbo C.' When you run the scroll.exe file, the program tries to open the message.txt file. If this file is not present in the same directory, it creates one with text "Welcome! You are watching running LED display..." and starts sending this message to the circuit via the parallel port for display on 5×7 dot-matrix pattern.

To increase the running speed of the message, press 'I' key, and to decrease the speed, press 'D' key. Press 'R' key for displaying the message from the beginning. When the program reaches the end of the message, it starts from the beginning again. To change the text being displayed, exit the program by pressing 'Esc' and edit the message.txt file using Notepad. After making changes to the message.txt file, save it and execute the scroll.exe file.

The program makes use of the outportb() function, which works perfectly only on Windows 95/98. However, the program may not work with the latest Window versions such as Windows 2000/XP .

When you try to save changes in the message.txt file, the window shows an error saying "Can't save message.txt. It is being used by some other application." This is because the scroll.exe file is running. So exit the program by pressing 'Esc' key, then save your changes made to the message. txt file and run the scroll.exe file. Now you can view your changes in the message being displayed.

The program does not show special characters like '/,' '\,'

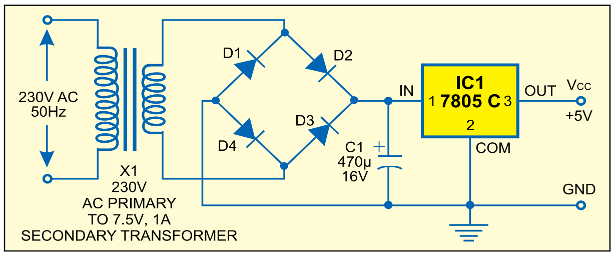


Fig. 2: Power supply

'~,' '@,' '#,' '$,' '%,' '^,' '(,' '),' '{,' '},' and ';.' It has been developed for displaying alphabets ('A' through 'Z'), digits ('0' through '9') and some special characters like '.,' ',,' '!,' '–,' '+' and '_.'

Other special characters can be added as follows: Suppose you want to display character 'A.' Draw 'A' on the 5×7 LED display as shown in Fig. 3. First, '7CH' data is available at the input of IC2 and the first flip-flop of IC8. When a clock pulse is received, this data (7CH) is output by IC2 and the first flip-flop of IC8 and new data '12H' arrives at the input pin of IC2 and the first flip-flop of IC8. The output data of IC2 and the first

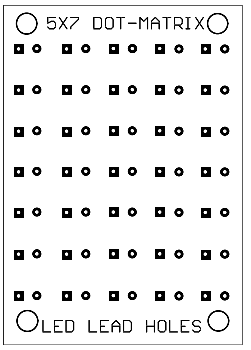


Fig. 7: Perforated board for 5x7 LEDs

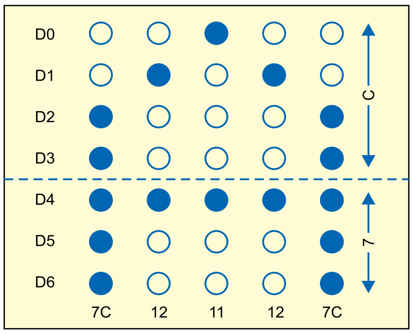


Fig. 3: Design of character 'A'

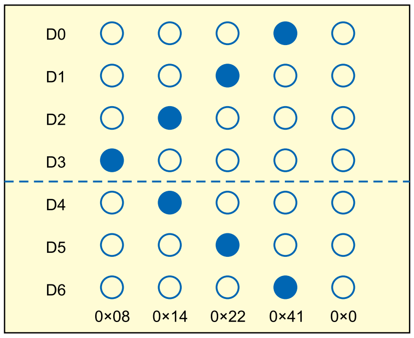


Fig. 4: Design of character '<'

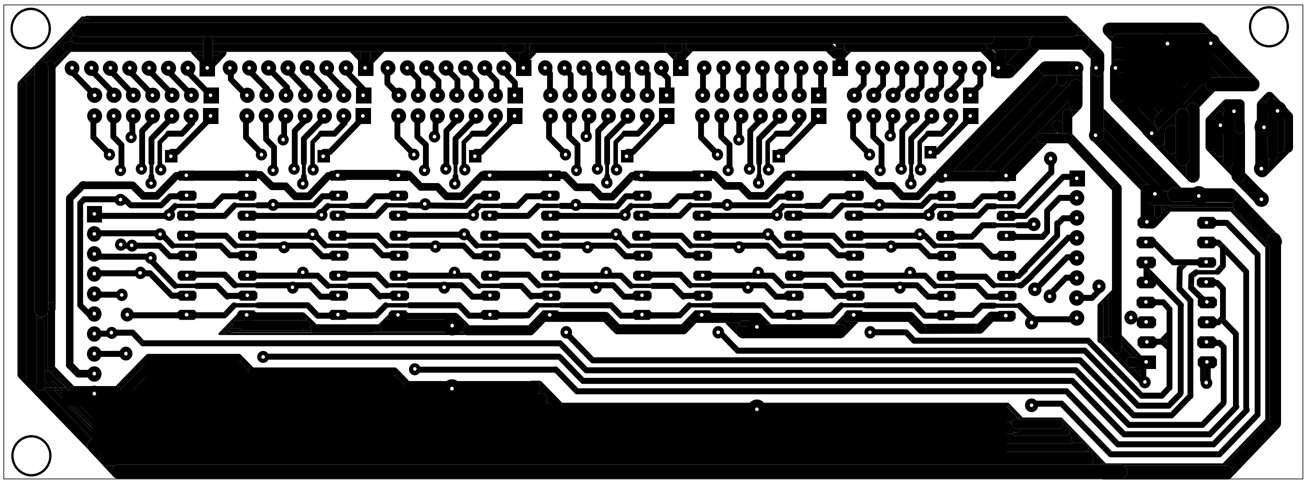
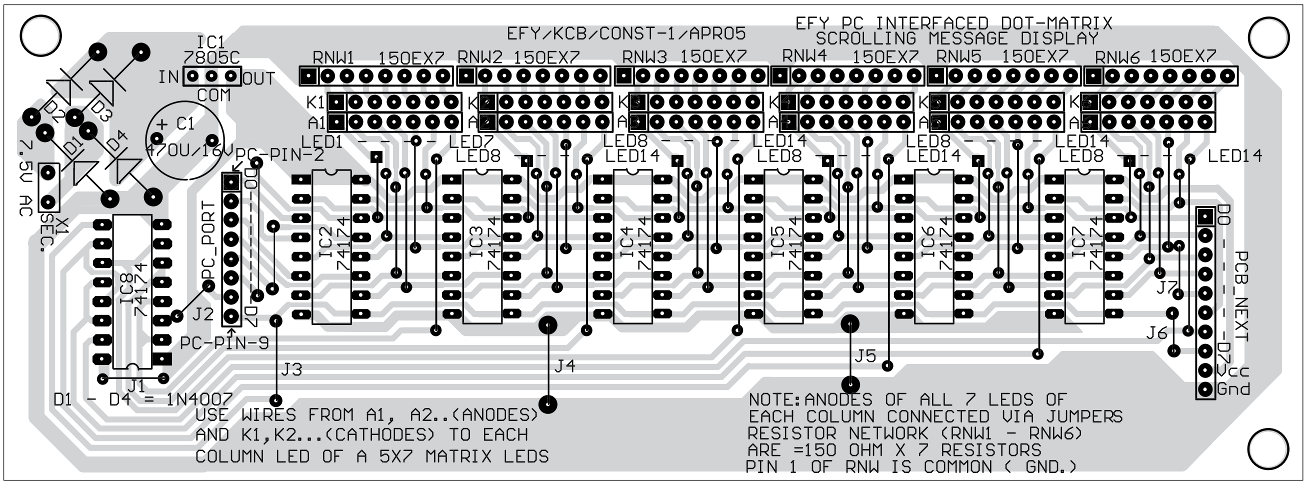


Fig. 5: Actual-size, single-side PCB for the LED-based scrolling message display including power supply



EFY/KCB/CONST-1/APR05
EFY PC INTERFACED DOT-MATRIX
SCROLLING MESSAGE DISPLAY
IC1
7805C
IN
OUT
COM
RNW1 150EX7
RNW2 150EX7
RNW3 150EX7
RNW4 150EX7
RNW5 150EX7
RNW6 150EX7
K1
A1
K
A
LED1
LED7
LED8
LED14
C1
470U/16V
PC-PIN-2
D0
D7
PC_PORT
J2
PC-PIN-9
J1
J3
J4
J5
J6
J7
D1
D2
D3
D4
7.5V AC
X1
SEC.
IC8
74174
IC2
74174
IC3
74174
IC4
74174
IC5
74174
IC6
74174
IC7
74174
PCB_NEXT
Vcc
Gnd
D1 - D4 = 1N4007
USE WIRES FROM A1, A2..(ANODES)
AND K1,K2...(CATHODES) TO EACH
COLUMN LED OF A 5X7 MATRIX LEDS
NOTE:ANODES OF ALL 7 LEDS OF
EACH COLUMN CONNECTED VIA JUMPERS
RESISTOR NETWORK (RNW1 - RNW6)
ARE =150 OHM X 7 RESISTORS
PIN 1 OF RNW IS COMMON (GND.)

Fig. 6: Component layout for the PCB

flip-flop of IC8 becomes the input for IC3 and the second flip-flop of IC8. When the next clock pulse is received, '7CH' data becomes available at the output of IC3 and output of second flip-flop of IC8, '12H' is available at the output of IC2 and the first flip-flop of IC8 and new data '11H' is available at the input of IC2 and the first flip-flop of IC8. This process continues until the message completes.

Now let's assume that you want to display '<.' For this, first draw this symbol on the 5×7 matrix as shown in Fig. 4. Assuming glowing LED as '1,' convert the binary column sequence into hexadecimal for all the five columns as shown in the figure. Finally, add the following lines in the software program where the comment "Add your codes here" appears:

Case '<' :
str1[0]=0x00;str1[1]=0x41;str1[2]=0x22
;str1[3]=0x14;str1[4]=0x8;
break;

Save the file and compile the program again. On executing the program, you can watch '<' being displayed on the message display.

Other special characters can be added in the same way.

Download source code: http://www.efymag.com/admin/issuepdf/PC%20Scroll%20Display.zip

SCROLL.C

```
 /***********************************
 SCROLLING  MESSAGE  DISPLAY
 DEVELOPED BY : SURESH KUMAR
 FINAL YEAR, IITT COLLEGE OF ENGINEER-
ING, PUNJAB
 THANX TO ALL TEACHERS AND MY PAR-
ENTS
 *********************************/
 #include<stdio.h>
 #include<dos.h>
 #include<conio.h>
 #include<process.h>
 unsigned char str1[5],str2[13],str[5];
 int DELAY=100;
 void setcode();
 void sendcode();
 void getcode(char);
 void main()
 {
         FILE *fp;
         char line[150],ch;
         clrscr();
         fp=fopen("message.txt","r");
         if(fp==NULL)
         {
                 fp=fopen("message.
txt","w");
                 if(fp==NULL)
                 {
                         printf("\n\
nCAN'T CREATE MESSAGE.TXT CREATE A FILE
UNDER NAME MESSAGE.TXT YOURSELF");
                         exit(0);
                 }
                 fputs(" Welcome! You are
watching running LED display... ",fp);
                 fclose(fp);
                 fp=fopen("message.
txt","r");
                 if(fp==NULL)
                 {
                         printf("\
nCAN'T FIND OR OPEN \"message.txt\"");
                         exit(0);
                 }
         }
         clrscr();
         startagain:
         while(!kbhit())
         {
                 ch=fgetc(fp);
                 if(ch==EOF)
                 {
                         rewind(fp);
                         continue;
                 }
                 printf("\nSCROLLING
MESSAGE DISPLAY : Sending \'%c\'",ch);
                 getcode(ch);
                 setcode();
                 sendcode();
         }
         ch=getch();
         switch(ch)
         {
                 case 'i':
                 case 'I':
                 if(DELAY>10)
                 {
                         DELAY-=5;
                 }
                 else
                 {
                         DELAY-=1;
                 }
                 if(DELAY<0)
                 {
                         DELAY=0;
                 }
                 printf("\nSCROLLING
MESSAGE DISPLAY : Speed Increased");
                 break;
                 case 'd':
                 case 'D':
                 DELAY+=10;
                 printf("\nSCROLLING
MESSAGE DISPLAY : Speed Decreased");
                 break;
                 case 'r':
                 case 'R':
                 rewind(fp);
                 printf("\nSCROLLING
MESSAGE DISPLAY : Started from Begining");
                 break;
                 case 27:
                 clrscr();
                 printf("\nSCROLLING
MESSAGE DISPLAY : Exiting ");
                 fclose(fp);
                 delay(1000);
                 printf(". ");
                 delay(200);
                 printf(". ");
                 delay(200);
                 printf(". ");
                 delay(200);
                 printf(". ");
                 delay(200);
                 exit(0);
         }
         goto startagain;
 }
 void getcode(char ch)
 {
         switch(ch)
         {
                 case 'a':
                 case 'A':
                 str1[0]=0x7c;str1[1]=0x12; st
r1[2]=0x11;str1[3]=0x12;str1[4]=0x7c;
                 break;
                 case 'b':
                 case 'B':
                 str1[0]=0x36;str1[1]=0x49; st
r1[2]=0x49;str1[3]=0x49;str1[4]=0x7F;
                 break;
                 case 'c':
                 case 'C':
                 str1[0]=0x22;str1[1]=0x41; st
r1[2]=0x41;str1[3]=0x41;str1[4]=0x3C;
                 break;
                 case 'd':
                 case 'D':
                 str1[0]=0x1C;str1[1]=0x22;s
tr1[2]=0x41; str1[3]=0x41;str1[4]=0x7F;
                 break;
                 case 'e':
                 case 'E':
                 str1[0]=0x41;str1[1]=0x41;st
r1[2]=0x49; str1[3]=0x49;str1[4]=0x7F;
                 break;
                 case 'f':
                 case 'F':
                 str1[0]=0x01;str1[1]=0x01;st
r1[2]=0x09; str1[3]=0x09;str1[4]=0x7F;
                 break;
                 case 'g':
                 case 'G':
                 str1[0]=0x3A;str1[1]=0x49;s
tr1[2]=0x41; str1[3]=0x41;str1[4]=0x3E;
                 break;
                 case 'h':
                 case 'H':
                 str1[0]=0x7F;str1[1]=0x08;st
r1[2]=0x08; str1[3]=0x08;str1[4]=0x7F;
                 break;
                 case 'i':
                 case 'I':
                 str1[0]=0x41;str1[1]=0x41;st
r1[2]=0x7F; str1[3]=0x41;str1[4]=0x41;
                 break;
                 case 'j':
                 case 'J':
                 str1[0]=0x7F;str1[1]=0x41;st
r1[2]=0x41; str1[3]=0x41;str1[4]=0x21;
                 break;
                 case 'k':
                 case 'K':
                 str1[0]=0x41;str1[1]=0x22;st
r1[2]=0x14; str1[3]=0x08;str1[4]=0x7F;
                 break;
                 case 'l':
                 case 'L':
                 str1[0]=0x40;str1[1]=0x40;st
r1[2]=0x40; str1[3]=0x40;str1[4]=0x7F;
                 break;
                 case 'm':
                 case 'M':
                 str1[0]=0x7F;str1[1]=0x02;st
r1[2]=0x04; str1[3]=0x02;str1[4]=0x7F;
                 break;
                 case 'n':
                 case 'N':
                 str1[0]=0x7F;str1[1]=0x08;st
r1[2]=0x04; str1[3]=0x02;str1[4]=0x7F;
                 break;
                 case 'o':
                 case 'O':
                 str1[0]=0x3E;str1[1]=0x41;st
r1[2]=0x41;  str1[3]=0x41;str1[4]=0x3E;
                 break;
                 case 'p':
                 case 'P':
                 str1[0]=0x06;str1[1]=0x09; st
r1[2]=0x09;str1[3]=0x09;str1[4]=0x7F;
                 break;
                 case 'q':
                 case 'Q':
                 str1[0]=0x3E;str1[1]=0x61; st
r1[2]=0x51;str1[3]=0x41;str1[4]=0x3E;
                 break;
                 case 'r':
                 case 'R':
                 str1[0]=0x46;str1[1]=0x29; st
r1[2]=0x19;str1[3]=0x09;str1[4]=0x7F;
                 break;
                 case 's':
                 case 'S':
                 str1[0]=0x32;str1[1]=0x49; st
r1[2]=0x49;str1[3]=0x49;str1[4]=0x26;
                 break;
                 case 't':
                 case 'T':
                 str1[0]=0x01;str1[1]=0x01; st
r1[2]=0x7F;str1[3]=0x01;str1[4]=0x01;
                 break;
                 case 'u':
                 case 'U':
                 str1[0]=0x3F;str1[1]=0x40; st
r1[2]=0x40;str1[3]=0x40;str1[4]=0x3F;
                        break;
                        case 'v':
                        case 'V':
                        str1[0]=0x1F;str1[1]=0x20;
str1[2]=0x40;str1[3]=0x20;str1[4]=0x1F;
                        break;
                        case 'w':
                        case 'W':
                        str1[0]=0x7F;str1[1]=0x20;
str1[2]=0x10;str1[3]=0x20;str1[4]=0x7F;
                        break;
                        case 'x':
                        case 'X':
                        str1[0]=0x63;str1[1]=0x14;
str1[2]=0x08;str1[3]=0x14;str1[4]=0x63;
                        break;
                        case 'y':
                        case 'Y':
                        str1[0]=0x03;str1[1]=0x04;
str1[2]=0x78;str1[3]=0x04;str1[4]=0x03;
                        break;
                        case 'z':
                        case 'Z':
                        str1[0]=0x03;str1[1]=0x04;
str1[2]=0x08;str1[3]=0x11;str1[4]=0x61;
                        break;
                        case '0':
                        str1[0]=0x1C;str1[1]=0x22
;str1[2]=0x41; str1[3]=0x22;str1[4]=0x1C;
                        break;
                        case '1':
                        str1[0]=0x40;str1[1]=0x40;
str1[2]=0x7F;str1[3]=0x42;str1[4]=0x44;
                        break;
                        case '2':
                        str1[0]=0x46;str1[1]=0x49;
str1[2]=0x51;str1[3]=0x61;str1[4]=0x42;
                        break;
                        case '3':
                        str1[0]=0x31;str1[1]=0x4B; st
r1[2]=0x45;str1[3]=0x49;str1[4]=0x21;
                        break;
                        case '4':
                        str1[0]=0x10;str1[1]=0x7F; st
r1[2]=0x12;str1[3]=0x14;str1[4]=0x18;
                        break;
                        case '5':
                        str1[0]=0x31;str1[1]=0x49; st
r1[2]=0x49;str1[3]=0x49;str1[4]=0x27;
                        break;
                        case '6':
                        str1[0]=0x32;str1[1]=0x49; st
r1[2]=0x49;str1[3]=0x51;str1[4]=0x3A;
                        break;
                        case '7':
                        str1[0]=0x07;str1[1]=0x79; st
r1[2]=0x01;str1[3]=0x01;str1[4]=0x01;
                        break;
                        case '8':
                        str1[0]=0x36;str1[1]=0x49; st
r1[2]=0x49;str1[3]=0x49;str1[4]=0x36;
                        break;
                        case '9':
                        str1[0]=0x3E;str1[1]=0x49; st
r1[2]=0x49;str1[3]=0x49;str1[4]=0x26;
                        break;
                        case '.':
                        str1[0]=0x60;str1[1]=0x60; st
r1[2]=0x00;str1[3]=0x00;str1[4]=0x00;
                        break;
                        case ' ':
                        str1[0]=0x00;str1[1]=0x00; st
r1[2]=0x00;str1[3]=0x00;str1[4]=0x00;
                        break;
                        case '!':
                        str1[0]=0x67;str1[1]=0x7F; st
r1[2]=0x00;str1[3]=0x00;str1[4]=0x00;
                        break;
                        case '-':
                        str1[0]=0x08;str1[1]=0x08; st
r1[2]=0x08;str1[3]=0x08;str1[4]=0x08;
                        break;
                        case '+':
                        str1[0]=0x08;str1[1]=0x08; st
r1[2]=0x3E;str1[3]=0x08;str1[4]=0x08;
                        break;
                        case '_':
                        str1[0]=0x40;str1[1]=0x40;st
r1[2]=0x40; str1[3]=0x40;str1[4]=0x40;
                        break;
 ////// ADD YOUR CODES HERE////////
                        default:
                        str1[0]=0x0;str1[1]=0x0;str1
[2]=0x0; str1[3]=0x0;str1[4]=0x0;
                        break;
            }
 }
 void setcode()
 {
            int i,k;
            for(i=0,k=0;i<10;i+=2,k++)
            {
                        str2[i]=str1[k];
                        str2[i+1]=str1[k]+128;
            }
                        str2[i]=0;
                        str2[i+1]=128;
 }
 void sendcode()
 {
            int i;
            for(i=0;i<12;i++)
            {
                        outportb(0x0378,str2[i]);
                        delay(DELAY);
            }
 }
```