PC-based Scrolling Message Display
Er. Suresh Kumar (www.ptupapers.tk)

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 diode (LED) display in moving 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 depends upon the size of 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 in 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 42 (6x7) LED display.

The PC's parallel port (LPT port) is used to output the display code and 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.

Pins 2 through 9 form the 8-bit data 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 signal are generated by the computer program and collected from the parallel port (base address 0x378) at the back of the computer. 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 current clock pulse prior to connecting it to the PIPO ICs.

Circuit description

Fig. 1 shows the circuit for the LED-based scrolling message display. IC 74174 has been used as PIPO register, which is a high-speed, hex D-type flip-flop. It is used as a 6-bit edge-triggered storage register. The data on the inputs of the flip-flop is transferred to storage during high-to-low transition. 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 D0 through D6 shifts from the first stage to the next stage, and so on, the message appears as scrolling on the LED display.

The display is made of 42 LEDs arranged in 6x7 matrix pattern. 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.

Fig. 2 shows the power supply circuit. The AC mains is stepped down by transformer X1 to deliver a secondary output of 6V AC at 500 mA. The transformer output is rectified by a full-wave bridge rectifier comprising diodes D1 through D4, filtered by capacitor C1, then regulated by IC 7805 (IC1) to provide regulated 5V DC to the circuit.

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, it creates one with text "Welcome You are watching running led display..." and starts sending the message to the parallel port, which is received by the circuit.

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 in 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 only on Windows 98. So the program does not work on any other operating system.

> 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 '/,' '\,' '~,' '@,' '#,' '$,' '%,' '^,' '&,' '(,' '),' '{,' '},' 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 5x7 LED display as shown in **Fig. 3**. First, '7CH' data is available at the input of IC2. When a clock pulse is received, this data (7CH) is outputed by IC2 and new data '12H' arrives at the input pin of IC2. The output data of IC2 becomes the input for IC3. When another clock pulse is received, '7CH' data becomes available at the output of IC3, '12H' is available at the output of IC2 and new data '11H' is available at the input of IC2. This process continues until the message completes.

Now let's assume that you want to display '<.' For this, first draw this symbol on the 5x7 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.

PART LIST:

Semiconductor:

IC1 - 7805 5V regulator

IC2-IC8 - 74174 hex D-type flip-flop

D1-D4 - 1N4007 rectifier diode

LED1- LED42 - Red LED

Resistor:

R1- R42 - 150-ohm

Capacitor:

C1 - 470µF, 16V electrolytic

Miscellaneous:

X1 - 230V AC primary to 6V, 1A secondary transformer

fig. 1 circuit diagram

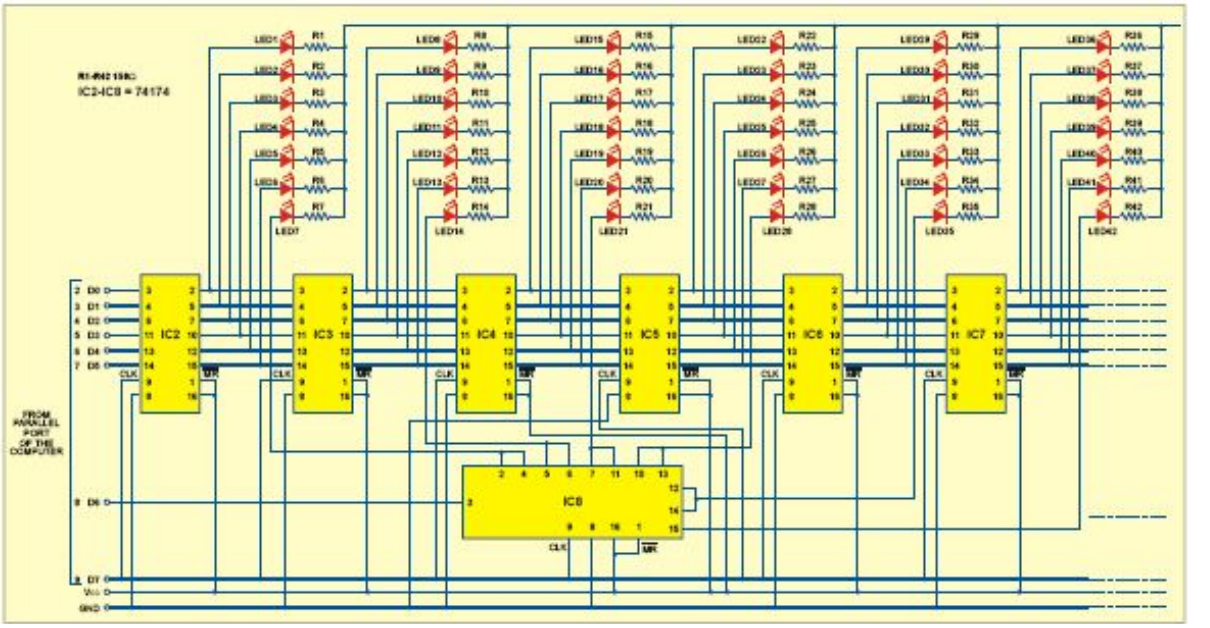


fig. 2 : DC Power Supply Unit

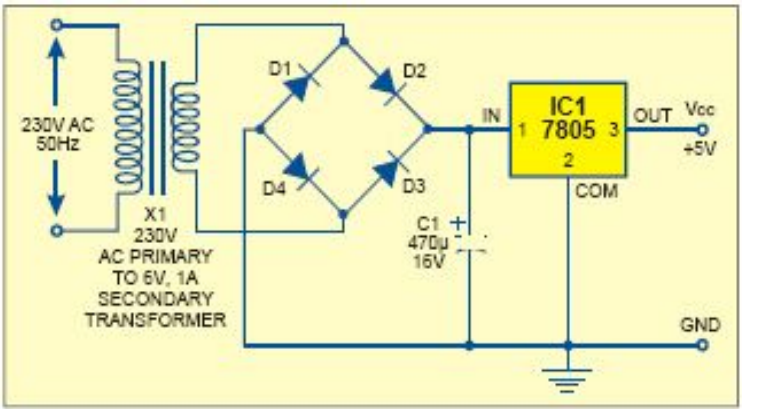
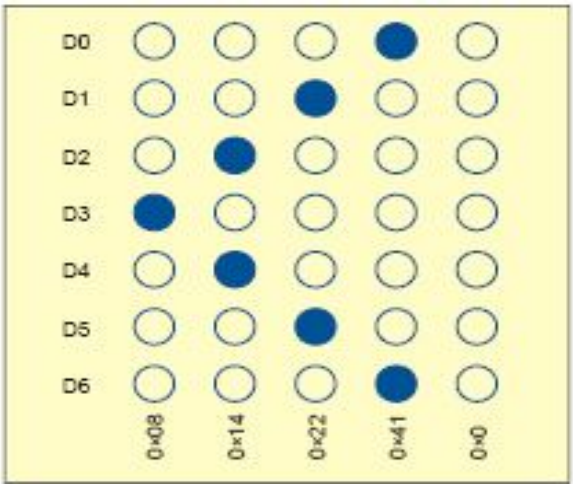
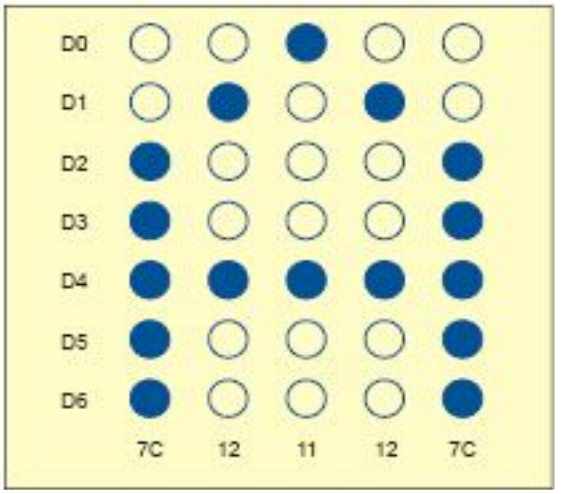


fig.3 & 4 : making of character '<' and 'A'





The C Program :

```
/************************************
SCROLLING   MESSAGE   DISPLAY
DEVELOPED BY : SURESH KUMAR
FINAL YEAR, IITT COLLEGE OF ENGINEERING, PUNJAB
THANX TO ALL TEACHERS AND MY PARENTS
WWW.PTUPAPERS.TK
*************************************/
#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;str1[2]=0x11;str1[3]=0x12;str1[4]=0x7c;
```

```
break;
case 'b':
case 'B':
str1[0]=0x7F;str1[1]=0x49;str1[2]=0x49;str1[3]=0x49;str1[4]=0x36;
break;
case 'c':
case 'C':
str1[0]=0x3C;str1[1]=0x41;str1[2]=0x41;str1[3]=0x41;str1[4]=0x22;
break;
case 'd':
case 'D':
str1[0]=0x7F;str1[1]=0x41;str1[2]=0x41;str1[3]=0x22;str1[4]=0x1C;
break;
case 'e':
case 'E':
str1[0]=0x7F;str1[1]=0x49;str1[2]=0x49;str1[3]=0x41;str1[4]=0x41;
break;
case 'f':
case 'F':
str1[0]=0x7F;str1[1]=0x09;str1[2]=0x09;str1[3]=0x01;str1[4]=0x01;
break;
case 'g':
case 'G':
str1[0]=0x3E;str1[1]=0x41;str1[2]=0x41;str1[3]=0x49;str1[4]=0x3A;
break;
case 'h':
case 'H':
str1[0]=0x7F;str1[1]=0x08;str1[2]=0x08;str1[3]=0x08;str1[4]=0x7F;
break;
case 'i':
case 'I':
str1[0]=0x41;str1[1]=0x41;str1[2]=0x7F;str1[3]=0x41;str1[4]=0x41;
break;
case 'j':
case 'J':
str1[0]=0x21;str1[1]=0x41;str1[2]=0x41;str1[3]=0x41;str1[4]=0x7F;
break;
case 'k':
case 'K':
str1[0]=0x7F;str1[1]=0x08;str1[2]=0x14;str1[3]=0x22;str1[4]=0x41;
break;
case 'l':
case 'L':
str1[0]=0x7F;str1[1]=0x40;str1[2]=0x40;str1[3]=0x40;str1[4]=0x40;
break;
case 'm':
case 'M':
str1[0]=0x7F;str1[1]=0x02;str1[2]=0x04;str1[3]=0x02;str1[4]=0x7F;
```

```
break;
case 'n':
case 'N':
str1[0]=0x7F;str1[1]=0x02;str1[2]=0x04;str1[3]=0x08;str1[4]=0x7F;
break;
case 'o':
case 'O':
str1[0]=0x3E;str1[1]=0x41;str1[2]=0x41;str1[3]=0x41;str1[4]=0x3E;
break;
case 'p':
case 'P':
str1[0]=0x7F;str1[1]=0x09;str1[2]=0x09;str1[3]=0x09;str1[4]=0x06;
break;
case 'q':
case 'Q':
str1[0]=0x3E;str1[1]=0x41;str1[2]=0x51;str1[3]=0x61;str1[4]=0x3E;
break;
case 'r':
case 'R':
str1[0]=0x7F;str1[1]=0x09;str1[2]=0x19;str1[3]=0x29;str1[4]=0x46;
break;
case 's':
case 'S':
str1[0]=0x26;str1[1]=0x49;str1[2]=0x49;str1[3]=0x49;str1[4]=0x32;
break;
case 't':
case 'T':
str1[0]=0x01;str1[1]=0x01;str1[2]=0x7F;str1[3]=0x01;str1[4]=0x01;
break;
case 'u':
case 'U':
str1[0]=0x3F;str1[1]=0x40;str1[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]=0x61;str1[1]=0x11;str1[2]=0x08;str1[3]=0x04;str1[4]=0x03;
break;
case '0':
str1[0]=0x1C;str1[1]=0x22;str1[2]=0x41;str1[3]=0x22;str1[4]=0x1C;
break;
case '1':
str1[0]=0x44;str1[1]=0x42;str1[2]=0x7F;str1[3]=0x40;str1[4]=0x40;
break;
case '2':
str1[0]=0x42;str1[1]=0x61;str1[2]=0x51;str1[3]=0x49;str1[4]=0x46;
break;
case '3':
str1[0]=0x21;str1[1]=0x49;str1[2]=0x45;str1[3]=0x4B;str1[4]=0x31;
break;
case '4':
str1[0]=0x18;str1[1]=0x14;str1[2]=0x12;str1[3]=0x7F;str1[4]=0x10;
break;
case '5':
str1[0]=0x27;str1[1]=0x49;str1[2]=0x49;str1[3]=0x49;str1[4]=0x31;
break;
case '6':
str1[0]=0x3A;str1[1]=0x51;str1[2]=0x49;str1[3]=0x49;str1[4]=0x32;
break;
case '7':
str1[0]=0x01;str1[1]=0x01;str1[2]=0x01;str1[3]=0x79;str1[4]=0x07;
break;
case '8':
str1[0]=0x36;str1[1]=0x49;str1[2]=0x49;str1[3]=0x49;str1[4]=0x36;
break;
case '9':
str1[0]=0x26;str1[1]=0x49;str1[2]=0x49;str1[3]=0x49;str1[4]=0x3E;
break;
case '.':
str1[0]=0x60;str1[1]=0x60;str1[2]=0x00;str1[3]=0x00;str1[4]=0x00;
break;
case ' ':
str1[0]=0x00;str1[1]=0x00;str1[2]=0x00;str1[3]=0x00;str1[4]=0x00;
break;
case '!':
str1[0]=0x67;str1[1]=0x7F;str1[2]=0x00;str1[3]=0x00;str1[4]=0x00;
break;
case '-':
str1[0]=0x08;str1[1]=0x08;str1[2]=0x08;str1[3]=0x08;str1[4]=0x08;
break;
case '+':
```

```
                    str1[0]=0x08;str1[1]=0x08;str1[2]=0x3E;str1[3]=0x08;str1[4]=0x08;
                    break;
                    case '_':
                    str1[0]=0x40;str1[1]=0x40;str1[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]=str1[k];
        }
                str2[i]=0;
                str2[i+1]=128;
}
void sendcode()
{
        int i;
        for(i=0;i<12;i++)
        {
                outportb(0x0378,str2[i]);
                delay(DELAY);
        }
}
```