MS—Windows 3.0 环境下的汉字显示

Microsoft 公司于 90 年代初推出的 Windows 3.0 以其丰富的窗口函数、友好统一的图形用户等优良性能而风行一时，国内也有不少单位相继应用其汉化版本。如果手头只有西文的 Windows，能否同样地显示汉字呢？笔者通过把汉字字模直接映射到屏幕的方法，解决了此问题。

这种方法是以点阵汉字库为基础的，以 16×16 点阵的显示字库为例，每个汉字由 32 个字节表示，其排列如图 1 所示。这样的点阵结构就好比用一个 16×16 的网络罩在汉字上，相交位为 1，其他位置 0。所有这 32 个字节被称为汉字的字模。另一方面，在 Windows 的图形环境下，屏幕图形均由像素构成，如果能将字模的点阵信息映射到屏幕上对应的像素，就能显示出相应的汉字。这就是本方法的基本思想。

这里的关键是找出要显示汉字的字模来。众所周知，汉字库是按区位顺序存放的，每区有 94 个汉字或符号。设某个汉字的区号为 m，位号为 n，则它的字模在汉字库的起始位置应为：

$$((m-1)\times 94+n-1)\times 32$$

下面给出一个示例程序的完整清单，包括 MAKE 文件、资源文件、模块定义文件、头文件和 C 源程序。利用 MAKE 文件生成可执行程序 SHOW_CH.EXE，在 Windows 环境下执行该程序，能按屏显示汉字库的所有符号与汉字。笔者这里选用了 XSDOS 的显示字库 CCLIBJ.DOT。

程序清单如下：

```
E:\Windows>type show_ch.m
all:show_ch.exe

show_ch.res:show_ch.rc show_ch.h
    rc -r show_ch.rc
show_ch.obj:show_ch.c show_ch.h
    rc -c -AS -Gsw -Oas -Zpe show_ch.c

show_ch.exe:show_ch.obj show_ch.def show_ch.res
    link /NOD show_ch,,,slibcew libw,show_ch.def
    rc show_ch.res
E:\Windows>
E:\Windows>type show_ch.h
#define IDM_OPEN    103
#define IDM_PROC    110
#define IDM_ABOUT   100
#define IDM_CL      111
#define IDM_FMS     112
#define IDM_CLOSE   102
int PASCAL WinMain(HANDLE,HANDLE,LPSTR,int);
BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE,int);
long FAR PASCAL MainWndProc(HWND,unsigned,
    WORD,LONG);
void Show_Chinese(HDC,int,int);
E:\Windows>
E:\Windows>type show_ch.def
NAME    Show_ch

DESCRIPTION 'Show Chinese in Windows'

EXETYPE   Windows

STUB   'WINSTUB.EXE'

CODE   PRELOAD MOVEABLE DISCARDABLE
DATA   PRELOAD MOVEABLE MULTIPLE
HEAPSIZE     1024
STACKSIZE    5120

EXPORTS
    MainWndProc   @1

E:\Windows>

E:\Windows>type show_ch.rc
#include "windows.h"
#include "show_ch.h"

FlexibleMenu MENU
BEGIN
    POPUP     "&Help"
    BEGIN
        MENUITEM "&About Flexible...", IDM
ABOUT
    END

    POPUP     "&File"
    BEGIN
    MENUITEM "&Open", IDM_OPEN
    MENUITEM "&FMS", IDM_PROC
    MENUITEM "&Close", IDM_CLOSE
    MENUITEM "&About", IDM_ABOUT
    END
    POPUP "&Option"
    BEGIN
    MENUITEM "&FMS", IDM_FMS
    MENUITEM "&CL", IDM_CL
```

```
    END
END

E:\Windows>
E:\Windows>type show_ch.c
/* * * * * * * * * * * * * * * * * * * * *
  PROGRAM: Show_Ch.c

    WinMain() - main program
    InitApplication() - initializes window data and regis
        ters window
    InitInstance() - saves instance handle and creates main
        window
    MainWndProc() - processes messages
    About() - processes messages for "About" dialog box
* * * * * * * * * * * * * * * * * * * * * /

#include "windows.h"
#include "string.h"
#include "stdio.h"
#include "Show_Ch.h"

HANDLE hInst;
HPEN    hBlueSolidPen;
HPEN    hYellowSolidPen;
HPEN    hGreenSolidPen;
HPEN    hRedSolidPen;
HPEN    holdPen;
HBRUSH holdBrush;
HBRUSH hBlueBrush;

/* * * * * * * * * * * * * * * * * * * * *
  FUNCTION: WinMain(HANDLE, HANDLE, LPSTR,
            int)
  PURPOSE: calls initialization function, processes message
            loop
* * * * * * * * * * * * * * * * * * * * * /

int PASCAL WinMain(hInstance, hPrevInstance, lpCmd-
Line, nCmdShow)
HANDLE hInstance;
HANDLE hPrevInstance;
LPSTR lpCmdLine;
int nCmdShow;
{
MSG msg;

if (!hPrevInstance)
    if (!InitApplication(hInstance))
        return (FALSE);
if (!InitInstance(hInstance, nCmdShow))
    return (FALSE);
while (GetMessage(&msg, NULL, NULL, NULL)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return (msg.wParam);
}
/* * * * * * * * * * * * * * * * * * * * *
  FUNCTION: InitApplication(HANDLE)
  PURPOSE: Initializes window data registers window class
* * * * * * * * * * * * * * * * * * * * * /

BOOL InitApplication(hInstance)
HANDLE hInstance;
{
WNDCLASS    wc;

wc.style = NULL;
wc.lpfnWndProc = MainWndProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance = hInstance;
wc.hIcon = LoadIcon(NULL, IDC_APPLICATION);
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = GetStockObject(WHITE_
    BRUSH);
wc.lpszMenuName = "FlexibleMenu";
wc.lpszClassName = "FlexibleWClass";

return (RegisterClass(&wc));
}
/* * * * * * * * * * * * * * * * * * * * *
  FUNCTION: InitInstance(HANDLE, int)
  PURPOSE: Saves instance and creates main window
* * * * * * * * * * * * * * * * * * * * * /
BOOL InitInstance(hInstance, nCmdShow)
HANDLE    hInstance;
int       nCmdShow;
{
HWND      hWnd;

hInst = hInstance;
hWnd = CreateWindow(
    "FlexibleWClass",
    "Show Chinese in MS-Windows 3.0",
    WS_OVERLAPPEDWindow | WS_VSCROLL | WS
_ HSCROLL \ | ES _ AUTOHSCROLL | ES _ AU-
TOVSCROLL,
    0,0,
    GetSystemMetrics(SM_CXSCREEN),
    GetSystemMetrics(SM_CXSCREEN),
    NULL, NULL, hInstance, NULL);
if (!hWnd)
```

```
    return (FALSE);
ShowWindow(hWnd,nCmdShow);
UpdateWindow(hWnd);
return (TRUE);
}
/* * * * * * * * * * * * * * * * * * * * *
   FUNCTION: MainWndProc (HWND, unsigned, WORD,
             LONG)
   PURPOSE: Show Chinese in screen
* * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL MainWndProc (hWnd, message,
     wParam, lParam)
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
HDC hDC;
PAINTSTRUCT ps;
LPSTR FileName;
char bv, * Buffer, * tem;
int   i=0,j=0,l,f,n,x,y;
unsigned    pixel_value;
long    m;

switch (message) {
  case WM_CREATE:
    hRedSolidPen = CreatePen (0,1,RGB(250,0,0));
    hGreenSolidPen = CreatePen (0,1,RGB(0,250,0));
    hBlueSolidPen = CreatePen (0,1,RGB(0,0,200));
    hBlueBrush = CreateSolidBrush(RGB(0,0,200));
    break;

  case WM_PAINT:
      x=0;y=0;
    hDC = BeginPaint(hWnd,&ps);
    holdPen = SelectObject(hDC,hBlueSolidPen);
    tem = Buffer;
    FileName = "c:\\clibj.dot";
    holdBrush = SelectObject(hDC,hBlueBrush);
    Rectangle(hDC,0,0,640,420);
    for (m=0;m<8000;m++) {
        f = _lopen(FileName,READ);
        if (f==-1)
        _llseek(f,m*32,0);
        _lread(f,Buffer,32);
        _lclose(f);
        for (i=0;i<16;i++) {
          for (j=0;j<2;j++) {
            bv = *(Buffer++);
            for (l=0;l<8;l++)
              if (((bv>>(7-l))&0x01)==0)
                SetPixel(hDC,x+j*8+l,y+i,RGB(0,
                250,0));
            }
          }
        Buffer = tem;
        x += 16;
        if (x>580) {
          x = 0; y +=16;
        }
        if (y>410) {
          y = 0;
          Rectangle(hDC,0,0,640,420);
        }
      }
    }
    SelectObject(hDC,holdBrush);
    SelectObject(hDC,holdPen);
    EndPaint(hWnd,&ps);
    break;

case WM_DESTROY:
    DeleteObject(hBlueSolidPen);
    DeleteObject(hBlueBrush);
    PostQuitMessage(0);
    break;

default:
    return (DefWindowProc (hWnd, message, wParam,
            lParam));
}
return(NULL);
}
```

最后补充一点，本文采用的字库是16×16点阵的显示字库，它为横向排列结构。如果选用精度更高的打印字库(常为24×24,32×32,48×48)，以上程序必须稍作修改，特别要注意的是，打印字库为纵向排列方式。

字节0	字节1
字节2	字节3
⋮	⋮
字节30	字节31

图1 显示字库的结构

上饶市 (93.1.6.131)

MASM 5.0中LINK时死机的原因及解决方法

MASM 5.0较之MASM 4.0增加了不少功能，但经常使用MASM 5.0的用户会发现这个问题，MASM 5.0编译通过的程序在LINK时却会"死机"。笔者在此讲解发现这个问题及寻找解决这个问题的过程，这样不单是解决了这一具体问题，用户还可从笔者解决这个问题的过程中可获得一种通用的解决这类问题的方法。

笔者曾经写过一个程序，将文件PAPER读入内存并对之进行处理，程序如下：

```
CODE  SEGMENT
  ASSUME CS:CODE,DS:CODE
  ORG   100H
START: JMP   BEGIN
FNAME: DB   'PAPER',0
BUFFER: DB   40000 DUP(0)
BEGIN: LEA   DX,FNAME
  MOV   AX,3D00H
  INT   21H
  JC    OPEN_ERR
  MOV   BX,AX
  LEA   DX,BUFFER
  MOV   AH,3FH
  INT   21H
  MOV   AH,3EH
  INT   21H
  .
  .
  .
CODE:  ENDS
  END   START
```

为节省篇幅，文中省略了打开文件出错的处理和文件的处理过程，

该程序在用MASM 5.0编译时通过，但在LINK时却死机。笔者怀疑是否LINK.EXE文件被破坏，但COPY一个好的LINK.EXE后，问题仍然存在。无奈，笔者怀疑是否文件的处理过程导致死机，因此将文件的处理过程去掉，只剩下打开文件、读文件和关闭文件，LINK时仍然死机。笔者又将读文件过程去掉，只剩下打开文件和关闭文件，在LINK时仍然死机。最后，笔者干脆使程序不做任何工作，而直接结束，程序如下：

```
CODE  SEGMENT
  ASSUME   CS:CODE,DS:CODE
  ORG    100H
START: JMP BEGIN
FNAME: DB   'PAPER',0
BUFFER:DB   40000 DUP(0)
BEGIN: MOV   AH,4CH
  INT   21H
CODE: ENDS
  END   START
```

结果LINK仍然死机。这时笔者又将语句FNAME DB 'PAPER',0去掉，只剩下语句BUFFER DB 40000 DUP(0)，结果仍然死机。此时笔者怀疑是否因为BUFFER定义较大所致，因此40000去掉一个0改为4000，LINK时经过短时间后通过，将4000改为10000，LINK时经过很长时间后通过。由此注意，笔者只是将缓冲区大小由4000改为10000，LINK的时间就大大延长（此时可以推测：缓冲区大小为10000时，程序并未死机，只是处理的时间太长）。笔者以前使用MASM 4.0时，从未遇到这种情况，于是笔者想是否将缓冲区定义为DUP(?)时情况也是如此呢？于是将DUP(0)改作DUP(?)，重新LINK时很快通过，又将10000改作40000，LINK时仍然很快通过。由此可知，这是由于定义缓冲区时缓冲区被赋予了初始值才导致LINK时"死机"的。

因为LINK使用的输入文件是.OBJ文件，可以推测：将缓冲区定义为DUP(0)时和将缓冲区定义为DUP(?)时，.OBJ文件是不相同的，于是笔者对这两种情况分别进行编译，缓冲区定义为DUP(0)时的.OBJ文件如下页所示(OBJ的大小为103)：

可以看出：DUP(0)时的.OBJ文件比DUP(?)时的.OBJ文件多了17个字节（从111H至151H），这17个字节构成了.OBJ文件的重复块代码记录，可见在DUP(?)时没有重复块代码记录。

至此，我们知道，如果程序中定义了一个有初值的较大缓冲区，LINK时就会"死机"。实际使用时，缓冲区大多数没有初值，因此在定义缓冲区时应把写成DUP(0)的习惯改为DUP(?)，但在某些特殊情况下，缓冲区确实需要初始化为一特定值，在此情况下为避免LINK时"死机"，可将缓冲区定义语句DUP(XX)改作DUP(?)，而在程序中使用下面的几条指令来实现缓冲区的初始化：

```
LEA DI,BUFFER
MOV CX,XXXXX  ;XXXXX为缓冲区的大小
MOV AL,XX
CLD
REP STOSB
```

这样，既初始化了缓冲区，又很快地通过LINK。

从笔者寻找这个问题的过程中可以看出，笔者使用的是逐步缩小范围以定位问题的方法。对这种莫名

其妙的问题，可采用这种方法找出解决问题的方法， 而不必去分析软件(比如本文的 LINK)。

```
C:\>DEBUG TL1.OBJ
-RCX      ;大小为103
CX 0067

-D100 167
36CA:0100 80 0E 00 0C 54 45 53 54-4C 49 4E 4B 2E 41 53 4D  ....TESTLINK.ASM
36CA:0110 E9 98 07 00 00 04 43 4F-44 45 44 98 07 00 60 47  ......CODEC...`G
36CA:0120 9D 02 01 01 19 88 04 00-00 A2 00 D1 A0 07 00 01  ..........Q...
36CA:0130 00 01 E9 00 00 6E 9C 08-00 84 01 00 01 01 43 9D  ...i..n.......C.
36CA:0140 F5 A2 0E 00 01 03 01 40-9C 01 00 01 00 00 00 01  u.......e.......
36CA:0150 00 6C A0 08 00 01 43 9D-B4 4C CD 21 89 8A 07 00  .l....C.4LM!....
36CA:0160 C1 00 01 01 00 01 AB E8                          A.....+h
```

将缓冲区定义为 DUP(?)时的.OBJ 文件如下(OBJ 文件的大小为86)：

```
C:\>DEBUG TL2/OBJ
-RCX
CX 0058          ;大小为86
D 100 156
36CA:0100 80 0E 00 0C 54 45 53 54-4C 49 4E 4B 2E 41 53 4D  ....TESTLINK.ASM
36CA:0110 E9 96 07 00 00 04 43 4F-44 45 44 98 07 00 60 47  ......CODED...`G
36CA:0120 9D 02 01 01 19 88 04 00-00 A2 00 D1 A0 07 00 01  ..........Q.....
36CA:0130 00 01 E9 00 00 6E 9C 08-00 84 01 00 01 01 43 9D  ....n.....C.
36CA:0140 F5 A0 08 00 01 43 9D B4-4C CD 21 89 8A 07 00 C1  u....C.4LM!.....A
36CA:0150 00 01 01 00 01 AB 43                             .....+C
```

王振祥 (93.1.6.139)

测试 CRT 的模式

目前个人计算机的显示器及其适配器的类型非常多，有单色的，也有彩色的。单色中又分 MDA、CGA、VGA 等等；彩色的也分 CGA、EGA、VGA 等等。在微机显示器及其适配器的开发与使用过程中，常常需要知道当前正使用的是什么模式？下面给出一个程序可以很方便地测出当前正使用的模式。

程序是用 TURBO C 2.0写成的。利用它所提供的函数，经编译、连接后，生成 EXE 文件，就可在命令行上直接键入 CRTMODE 来运行。程序已经在各种微机上使用过，包括 IBM－PC、XT、AT，以及各种兼容的286、386等微器。

```
#include <stdio.h>
#include <graphics.h>

#define P(note) printf(note)
#define PV(format,value) printf(format,(value))
#define PM printf(" mode is ")
#define PD printf("\n\tDetected graphics _driver is ")
void main(void)
{ int g_driver,g_error,g_mode;

  detectgraph(&g_driver,g_mode);
  if(g_driver<0)
  { p("No graphics hardware detected\n");
    exit(1);
  }
  switch(g_driver)
  { case 1: PD;P("CGA,");
      switch(g_mode)
      {case 0: PM; P("CGAC0 320 * 200");
       case 1: PM; P("CGAC1 320 * 200");
       case 2: PM; P("CGAC2 320 * 200");
       case 3: PM; P("CGAC3 320 * 200");
       case 4: PM; P("CGAHI 640 * 200");
       }
      break;
    case 2: PD; P("MCGA,");
      switch(g_mode)
      {case 0: PM; P("MCGAC0 320 * 200");
       case 1: PM; P("MCGAC1 320 * 200");
       case 2: PM; P("MCGAC2 320 * 200");
```

```
        case 3: PM1 P("MCGAC3 320 * 200");
        case 4: PM1 P("MCGAMED 640 * 200");
        case 5: PM1 P("MCGAHI 640 * 480");
        }
        break;
case 3: PD1P("EGA,");
        switch(g_mode)
        {case 0: PM1 P("EGALO 640 * 200");
         case 1: PM1 P("EGAHI 640 * 350");
        }
        break;
case 4: PD1P("EGA64,");
        switch(g_mode)
        {case 0: PM1 P("EGA64LO 640 * 200");
         case 1: PM1 P("EGA64HI 640 * 350");
        }
        break;
case 5: PD1P("EGAMONO,");
        PM1P("EGAMONOHI 640 * 350");
        break;
case 6: PD1P("HM8514,");
        switch(g_mode)
        {case 0: PM1 P("HM8... 1024 * 480");
         case 1: PM1 P("HM8... 1024 * 768");
        }
        break;
case 7: PD1P("HEROMONO,");
        PM1P("HEROMONO 720 * 348");
        break;
case 8: PD1P("ATT400,");
        switch(g_mode)
        {case 0: PM1 P("ATT400C0 320 * 200");
         case 1: PM1 P("ATT400C1 320 * 200");
         case 2: PM1 P("ATT400C2 320 * 200");
         case 3: PM1 P("ATT400C3 320 * 200");
         case 4: PM1 P("ATT400MCD 640 * 200");
         case 5: PM1 P("ATT400HI 640 * 200");
        }
        break;
case 9: PD1P("VGA,");
        switch(g_mode)
        {case 0: PM1 P("VGAIO 640 * 400");
         case 1: PM1 P("VGAMED 640 * 350");
         case 2: PM1 P("VGAHI 640 * 480");
        }
        break;
case 10: PD1P("PC3270,");
        PM1P("PC3270HI 720 * 350");
        break;

P("...THANK YOPU VERY MUCH FOR USING
THE CRTMODE PROGRAME!");
}
```

何　瑜　(93.1.6,141)

紫金多用户卡的故障排除

ZJ286 8I/O 卡(8串一并),串口采用 RS－232C 标准,并口采用 Centronics 标准.该卡适用于 DOS 系统和 Xenix 系统支持下的多用户.串口由通用可编程异步接口片(USART INS8250B 和一些电平转换芯片组成;并口是由一些中小规模 TTL 通用电路组成。

并口常见故障与维修

打印口出现故障一般有两大类:一类是插上 I/O 卡后主机无法启动;第二类是不影响主机启动,但不能打印.

(1)插上 I/O 卡后主机不能启动:

这种故障一般是 I/O 卡中被损坏的芯片将系统总线的地址线或数据线锁位成固定电平,或者是 I/O 卡上的＋5V 电源与地短路.用万用表和逻辑笔测量地址线和数据线正常时均应有脉冲.若电源、数据线和地址线均正常,可再查IOR、IOW、Busy、ACK信号是否正常,如果"忙"信号(Busy)和"回答"信号ACK出现故障,会导致主机死锁或等待。当上述信号均正常时,一般不影响主机启动.

(2)不影响主机启动,但联机不打印:

当打印机处在联机状态下,主机送出打印数据后,打印机不打印,这种现象一般分为锁死主机和不锁死主机两种状态.送出打印数据后锁死主机的情况较常出现,这种故障常常由命令译码器、数据收发器和控制信号几部分引起。

检查这类故障时,一般可先用随机诊断程序进行检查,但随机诊断程序功能不强,画面提示仅出现 PASS 或 ERROR,只能大致判断 I/O 卡是否有故障,而不能具体定位。但可借助汇编语言和系统资源,自己动手编制功能强的诊断程序,限于篇幅,本文不再叙述诊断程序的编制方法,现介绍一种用 DEBUG 对打印口定位的简易方法。在 DOS 操作系统下调用动

态调试程序 DEBUG，接着用 DEBUG 的读入命令 I 和输出命令 O 对三个寄存器的地址端口进行读写(状态寄存器为只读)，然后和表1进行比较，若和表1不同，则一定是相应地址端口的电路有问题，这样就使故障的范围大大缩小，且减小了对硬件电路检查的盲目性，具有较强的针对性。

表1　打印机各 I/O 口地址及正常初始值

	地址	脱机	联机
数据输出寄存器	378H	AA	AA
状态寄存器	379H	FF(7F)*	DA
控制寄存器	37AH	AC	AC

*7F 为并行端口上打印电缆在未联打印机时的状态。

```
A:DEBUG↙
I 378↙
00
-I 379↙
FF
-I 37A↙
AC
```

程序运行的结果表明数据通路有故障，用输出命令对数据进行修改，结果为：

```
-O378AA↙
-I 378↙
00
```

这表明数据7引不进去，很可能是数据锁存器 U8(74LS374)的控制信号有故障。用示波器检测时发现74LS374的11脚 CLK 为悬空电平，而该脚与命令译码器 U29(74LS155)的9脚相联，经检查 U29的2C、2G 端均正常。断开 U29的9脚引线后，9脚仍为悬空电平，故判断 U29损坏，更换 U29后故障排除。

(3)在 DOS 系统下打印机能正常工作，但在 Xenix 系统下打印机不能打印：

在 DOS 系统下能打印，但在 Xenix 系统下不能打印，原因是 DOS 系统使用查询方式(查忙信号 Busy)，Xenix 使用中断方式(中断7或中断5)，这时可检查打印中断线 U35(74LS04)的 8、9 脚，U41(74LS125)的4、5、6脚和 U31(74LS174)的第7脚是否正常。若中断线正常，但联机不打印，可在 Xenix 系统的超级用户状态下打入 LPinit 命令。根据画面提示对打印机进行初始化，经过初始化后，打印机就可正常运行了。

串口常见故障与维修

串口常见故障有两种形式：一种是整个串口均不能通讯；另一种是串口中的某一个串口不能通讯。

(1)整个串口不能工作：

整个串口不能工作时，可首先检查时钟线 UT(晶体振荡器1.8432MHz)的第8脚及 U31(7405)的第2和第4脚是否有输出，频率是否正确。时钟线正常时可以接着检查数据总线 U45(74LS245)的1和19脚、U40(74LS133)的9脚和 U38(74LS14)的12、13脚是否正常。正常时，U45的19脚应是低电平。若数据线正常可以检查读写控制线 U44(74LS125)的6、8、11脚和 U38(74LS14)的6、8脚，控制线也正常时可检查中断线 U1(74LS374)、U36(74LS245)、U37(74LS32)的3、6、8脚、U9、U10(74LS51)、U38(74LS14)的2、4脚，U44(74LS125)的3脚，U41(74LS125)的8脚。当中断线也正常时可检查接口地址译码器 U42、U30(74LS138)、U43(74LS21)的6、8脚，但两个译码器同时损坏的概率很小。

若串口1通讯正常，串口2～8均不能通讯时，可检查 CT3的短接插头是否接触不良或脱落；若 CT3的短接插头脱落，则串口2～8均被截止。

(2)串口中某一个串口不能通讯：

串口中某一个串口或几个串口不能通讯时，在确定时钟和相应译码电路正常的条件下，可检查对应串口的 USART 及收发电平转换电路，因串口使用 USART 芯片，使电路结构比较简单，对故障的排除相对较易。如：主机启动后，串口不能进行 Login 状态，但键入命令无反映，即不能与主机通讯。根据现象分析是发送部分正常，故障出在接收部分。经实际检查为：U24(1489)接收电平转换器损坏，更换后故障排除。

为方便维修，现给出每个串口所使用的器件编号及脚号，以便出现故障时好对号入座，进行查找：

串口1：U28(8250)、U6(1488－6、8、11脚)、U7(1489)、U8(1489－3脚)、U44(LS125－3脚)

串口2：U23(8250)、U49(1489－3脚)、U6(1488－3脚)、U18(1488－8、11脚)、U22(1488－6、11脚)

串口3：U26(8250)、U4(1489－8脚)、U20(1489－3、6脚)、U24(1489)、U25(1488－3、8、11脚)

串口4：U27(8250)、U4(1489－6、8、11脚)、U5(1488－8、11脚)、U15(1488－3脚)、U19(1489－3、6脚)

串口5：U13(8250)、U14(1489－3、6、11脚)、U20(1489－8、11脚)、U22(1488－3、8脚)、U25(1488－6脚)

串口6：U12(8250)、U2(1489－8脚)、U3(1489)、U15(1488－6、8、11脚)

串口7：U11(8250)、U2(1489－3、6、11脚)、U5(1488－3、6脚)、U19(1489－8、11脚)、U21(1488－11脚)

串口8：U10(8250)、U17(1489)、U18(1489－6脚)、U21(1488－3、6、8脚)、U11(LS125－8脚)

(3)当以上检查均无问题时，可检查终端的通讯

参数是否设置正确：

西文系统：7位数据，1位停止位，偶校验、9600波特，双向传送方式。

中文系统：8位数据，1位停止位，不校验，9600波特，双向传送方式。

李家宏 (93.1.6,143)

HGC 卡单显环境下2.13H 系统存在的问题及解决方法

2.13H 系列汉字系统在 EGA 和 VGA 下运行良好，但在 CGA 卡单色显示器上使用却不令人满意，并且在 HGC 卡单色显示器环境下问题更多些。下面是我们对其作的几点修改。

2.13H 汉字系统在 HGC 卡单显卡可以调用三个显示模块（CH11. COM，CH21. COM 和 CH25. COM），分别进入三种中文屏显方式：11行、21行和25行汉字屏显，其中调入 CH25. COM 后屏幕乱套死机，系统无法正常运行，而调入 CH11或 CH21后汉字仅半高，按 CTRL－F7可从中文到西文状态，但再也无法从西文回到中文状态。

我们常用的是25行汉字方式。考察发现 CH25. COM 中开头 CRTC 参数位置不对，往后移两字节，汉字显示即可正常，但字符位置不对，将230D 处的2833改为2798（字符发生器首址），则 CH25. COM 基本可用，但仍有两点不尽如人意：

（1）字符显示仅汉字之一半高，不美观。我们知道，字符字模库中字符图形点阵为8×8，25行汉字屏显方式每行18线，原 CH25. COM 每行中字符显示的方式是6线空，8线字符，下面4线空，这样，字符就只有汉字的一半高。为此，将 CH25. COM 中字符显示有关部分修改如下：

```
DEBUG CH25.COH
U 232D
XXXX:232D MOV CX,8      每字符字模的字节数送入 CX
XXXX:2330 LODSB         取 SI 所指处字模字节送入 AL
XXXX:2331 TEST BL,70    是否反象显示?
XXXX:2334 JZ 2338       不反象转2338
XXXX:2336 NOT AL        取反象
XXXX:2338 STOSB         取 AL 送入字模缓冲区中 DI 所指单元中
XXXX:2339 STOSB         再取一次，纵向放大字符，使字符与汉字等高
XXXX:233A LOOP 2330     取完整个字符之字模到字模缓冲区中
XXXX:233E XOR AX,AX     AX 清0，作填冲空白线用
XXXX:2341 TEST BL,70
XXXX:2343 JZ 2345
XXXX:2344 NOT AX
XXXX:2345 STOSW
XXXX:2346 JMP 2356
```

（2）2.13系列汉字系统通过 CTRL－F7进行中西文状态转换，但 HGC 卡单显环境下却只能从中文到西文，而从西文回不了中文，究其原因，出在键盘管理模块 CCCC. COM 内 CTR－F7键的处理部分。

该处有关部分原为：

```
DEBUG CCCC.COM
U 9A79
XXXX.9A79 CMP AL,3      在西文文本方式吗?
XXXX.9A7b JBE 9A81      是，转9A81
XXXX.9A7D MOV AL,3      中文方式转为西文文本方式
XXXX.9A7F JMP 9A83
XXXX.9A81 MOV AL,6      西文文本方式转为中文方式
XXXX.9A83 XOR AH,AH     设置 AH＝0，准备显示方式
XXXX.9A84 INT 10        调用中断10H 的0号功能，改变屏显方式
```

……

众所周知，HGC 卡西文本方式编号为7，其它卡西文文本方式皆为4以下，上段程序只能从4以下编号转向中文方式号6。所以原 CCCC. COM 只对其它显示环境有效而对 HGC 卡中西文转换无效。

我们将该段改为：

```
U 9A79
XXXX:9A79 CMP AL,6      在中文方式吗?
XXXX:9A7b JZ 9A81       是，转9A81
XXXX:9A7d MOV AL,6      西文文本方式转为中文方式
XXXX:9A7F JMP 9A83
XXXX:9A81 MOV AL,7      中文方式转为西文文本方式
XXXX:9A83 XOR AH,AH
XXXX:9A84 INT 10
```

……

但这样修改的 CCCC. COM 对其它显示环境又不能进行中西文的互换，若要通用，必须对此段进行较大修改，笔者认为没有必要。

徐文兵 (93.1.13,99)

修改2.13H汉字系统中LQ1600打印机驱动程序单/双向打印控制命令错误

CCDOS2.13H汉字系统打印功能很强，但驱动LQ1600系列打印机时不能双向打印，本人通过分析打印机驱动程序，查出了其中的不足，用DEBUG调入PRTA.COM打印机驱动程序，在XXXX:0DD5H开始有如下程序：

```
C>DEBUG 213\PRTA.COM
-
-UDD5
24E2:0DD5 B03E      MOV AL,3E
24E2:0DD7 A2FF09    MOV [09FF],AL
24E2:0DDA 3C3E      CMP AL,3E
24E2:0DDC 7404      JZ 0DE2
24E2:0DDE B001      MOV AL,01
24E2:0DE0 EB02      JMP 0DE4
24E2:0DE2 B000      MOV AL,00
24E2:0DE4 50        PUSH AX
24E2:0DE5 B01B      MOV AL,1B
24E2:0DE7 E866F7    CALL 0550
24E2:0DEA B055      MOV AL,55
24E2:0DEC E866F7    CALL 0550
24E2:0DEF 58        POP AX
24E2:0DF0 E95DF7    JMP 0550
-
```

```
eDDB
3E,3C
-w
Writing 1083 bytes
-q
```

根据2.13H使用说明书，(ASCII码3E)为单向控制命令，(ASCII码3C)为双向控制命令，同时参阅LQ1600打印机说明书可知，打印机单/双向控制码序列为1B 55 N，N=0时双向打印，N=1时单向打印。从上面程序可以看出，当AL内容为3E(3E单向控制码)时，应将01赋给AL；AL为3C(双向控制码)，应将00赋给AL。实际程序却正好与此相反，即2.13H系统单/双向打印控制码与说明书相反，实际使用也确是如此，若想与说明书控制码一致，只需将DDBH的值3E改为3C即可。

另外，程序中调用的子程序实际是执行ROM-BIOS中的17号打印中断例程，此处的调用地址是不对的，真正的地址为1A0H。对于选择5号LQ1500系列打印机，在打印机驱动程序驻留内存过程中，将D80H起始的内容前移3B0H个字节，这样调用子程序地址就正确了。

张　平　(93.1.3.99)

怎样使WordStar使用更方便

WordStar使用相当普遍，但该软件有几点美中不足之处：

1. 移光标到行首和移光标到行尾分别用组合键^QS和组合键^QD，都要击键三次，远不如其它一些软件如SK用Home键和End键来得方便。

2. Del键只删除光标左边的字符，而删除光标所在处的字符要用组合键^G。

3. Backspace只将光标前移一个位置，而不符合人们用退格键删除光标左边字符的习惯。

针对上述问题，本人仔细分析了WordStar的结构，找到了一种解决问题的简单办法，WordStar在屏幕编辑状态下，用16H号中断检测键盘状态。然后根据输入键的ASCII码值和扫描码值区分功能，分别转向不同的处理子程序。而各键功能处理子程序的入口地址表存放在CWS.COM或者WS.COM的第380H字节开始处。这样，只要找到相应的入口地址加以修改，即可方便地达到目的了。

下面介绍具体的修改方法。该方法对于中文版不管是11行的还是25行的都适用，对于西文版只试验了版本3.3。

首先用DEBUG装入主程序WS.COM

对于西文版：

C:>DEBUG　　WS.COM↙

对于中文版：

C:>DEBUG　　CWS.COM↙

1. 改Home键为移光标到行首：

```
-E4BF
* * * * : 04BF 64. BA 7H. 7E↙
```

2. 改End键为移光标到行尾：

```
E4BB↙
* * * * :04BB 8F. 95 7F. 7E↙
```

3. 修改Del键为删光标所在处字符：

E52B↙

＊ ＊ ＊ ＊ ：052B AE. D6↙

4. 修改退格键为删光标左边的字符：

－E49B↙

＊ ＊ ＊ ＊ ：049B 0D. AE 7E. 83↙

5. 由于 WordStar 把小键盘区的光标左移键与退格键视为等同，在改了退格键以后，光标左移键也随之改变，这是我们所不希望的，为此应将其与退格键区别对待：

－E6E6↙

＊ ＊ ＊ ＊ ：06E6 08. 13↙

至此已全部修改完毕，存盘退出即可：

－W↙

Q↙

当然读者也可以只改其中的一、两项，但退格键和光标左移键要同时修改。

卢殿森 （1993. 1. 13，103）

在有效期外运行规定有效期软件的通用程序

一些软件的使用有日期限制，要运行这类软件必须在具有效期内，如中文制表软件 CCED，当系统日期设置在1989年12月31日以后时，在使用时屏幕会显示："CCED 使用期限已到，需要更新版本？"然后即退出 CCED，此时 CCED 将无法使用；另外配 PUC 汉卡的主机，若系统日期设置在1991年7月31日以后，在进入 WPS 后屏幕会显示："你目前所使用的 WPS 版本 2.1 软件将于1991年7月31日到期"等信息，如果此时继续使用 WPS 软件，文件 WPS2.OVL 将被删除。为了运行这类软件通常的作法是：先用 DOS 的 DATE 命令将系统日期改在软件的使用期限以内，然后再运行这些有日期限制的软件，运行完后再用 DATE 命令将日期改回为当前实际日期。整个过程较为麻烦，并且在使用这类软件时若忘记事先修改系统日期，可能对应用软件和我们的编程序造成破坏。实际上只要能实现对系统当前日期的临时保存与释放，以上过程即可由批处理自动完成。现介绍批处理文件 RTIME. BAT（见程序一）和 CURRENT. BAT（见程序二）在运行规定有效期的软件时，完成系统当前日期的临时保存与释放的过程。

在批处理文件 RTIME. BAT 中，第3行使用的 FOR 命令是允许多输入转向到同一标号的较好方法，当 RTIME. BAT 的%1参数接受小写字符串 cced、wps 或大写字符串 CCED、WPS 的输入时，将转向标号 MOTION，而其它输入将会遭到拒绝，此条件判断仅由一行 FOR 命令即可完成，从而避免了使用长串的 IF 命令来控制批处理文件转向。第7行用伪变量 MOTION 来保存%1参数，第8行把回车键送到临时文件 TEMP1 中，第9行利用 DOS 管道命令"|"将 TEMP1 中的回车符送到 DATE 命令中，DATE 命令执行后即输出类似于以下两行的信息：

Current date is Fri 5－1 5－1992

Enter new date (mm－dd－yy)：

利用 DOS 的重定向功能将这两行信息添加到文件 RQ. BAT 中，第10行用 DOS 的 CALL 命令调用文件 RQ. BAT，也即执行上述两行信息中的第一行，DOS 将 Current 赋给可替换参数%0，开始调用批处理文件 CURRENT. BAT，并将 date 赋给可替换参数%1，is 赋给可替换参数%2，Fri 赋给可替换参数%3，系统当前日期5－1 5－1992赋给可替换参数%4。

批处理文件 CURRENT. BAT 中，第2行将系统当前日期保存到临时文件 temp2中，第3行、第4行将系统日期修改为1－01－1980，以便调整系统日期在所有规定有效期的软件的有效期以内，第5行释放伪变量 MOTION，用 CALL 命令调用规定有效期的软件名，第6行从临时文件 temp2中恢复系统的当前实际日期，第7行用命令"SET MOTION＝"来删掉伪变量 MOTION 以节省环境空间，注意等号后无空格。建好批处理文件 RTIME. BAT 和 CURRENT. BAT 后，便可正常运行各类规定有效期的软件而不必将日期限制的问题。例如，在系统日期为1992年12月31日到期的 CCED 软件，只要在 DOS 提示符下键入"C＞RTIME CCED＜Enter＞"，利用本程序除了能运行 CCED、WPS 等软件外，还可运行其它有日期限制的软件，只要在 RTIME. BAT 中第3行 FOR 命令的括号内加入相应软件名的大小写即可。

程序一：

```
C>TYPE RTIME.BAT
1:@echo off
2:cls
3:for %%f in (cced CCED wps WPS) do if @%1==@%%f goto motion
4:goto retry
5: :motion
6:echo ＊ ＊ ＊ 在有效期外运行%1软件 ＊ ＊ ＊
7:set motion=%1
8:echo+ >temp1
9:type temp1 date rq.bat>nul
```

```
10:call rq.bat
11:del temp1
12:del rq.bat
13:echo ＊＊ %1软件运行结束,系统日期恢复正常 ＊＊
14:goto end
15: :retry
16:echo please input filename
17:echo+
18:echo such as "C:>RTIME CCED"
19: :end
```

程序二：

```
C:\TYPE CURRENT.BAT
1:@echo off
2:echo %4>temp2
3:echo 12-30-1980>temp3
4:date<temp3
5:call %motion%
6:date<temp2 >nul
7:set motion=
8:del temp2
9:del temp3
```

宋　捷　(93.1.13.103)

硬盘系统参数的获取

硬盘由五部分组成：主引导扇区、引导扇区、文件分配表、根目录表和文件数据区。其系统参数因硬盘类型和DOS版本的不同而不同。在实际应用中，获取相应的硬盘系统参数有时是必要的。笔者用汇编语言编写了一段程序，可自动获得不同硬盘类型和DOS版本的硬盘系统数据。

本文所述的硬盘系统参数包括以下几个方面：

1：根目录的起始扇区。

2：根目录的最大项数。

3：文件分配表占用的扇区数。

4：每磁扇区数。

5：磁盘簇号和其对应扇区的换算公式。

6：硬盘的总扇区数。

7：文件分配表FAT连续字的字节数。

本程序在东海0520F、东海0530、长城386、SUPER286等机器上调试通过。

反汇编程序清单如下：

```
-C>debug ypxx.com

3E56:0100 b80080  MOV  AX,8000
3E56:0103 8ED8    MOV  DS,AX
3E56:0105 31D8    XOR  BX,BX
3E56:0107 B90100  MOV  CX,0001
3E56:010A 31D2    XOR  DX,DX
3E56:010C B002    MOV  AL,02
3E56:010E CD25    INT  25
3E56:0110 3E      DS:
3E56:0111 A11600  MOV  AX,[0016]
3E56:0114 D0E0    SHL  AL,1
3E56:0116 FEC0    INC  AL
3E56:0118 E8E500  CALL  0200
3E56:011B BB1503  MOV  BX,0315
3E56:011E E82F01  CALL  0250
3E56:0121 A11100  MOV  AX,[0011]
3E56:0124 E8D900  CALL  0200
3E56:0127 BB2F03  MOV  BX,032F
3E56:012A E82301  CALL  0250
3E56:012D 88E0    MOV  AL,AH
3E56:012F E8CE00  CALL  0200
3E56:0132 BB2D03  MOV  BX,032D
3E56:0135 E81801  CALL  0250
3E56:0138 A11100  MOV  AX,[0011]
3E56:013B D1E8    SHR  AX,1
3E56:013D D1E8    SHR  AX,1
3E56:013F D1E8    SHR  AX,1
3E56:0141 D1E8    SHR  AX,1
3E56:0143 40      INC  AX
3E56:0144 8B1E1600  MOV  BX,[0016]
3E56:0148 D1E3    SHL  BX,1
3E56:014A D1D8    ADD  AX,BX
3E56:014C E8B100  CALL  0200
3E56:014F BB9703  MOV  BX,0397
3E56:0152 E8EB00  CALL  0250
3E56:0155 A01600  MOV  AL,[0016]
3E56:0158 E8A500  CALL  0200
3E56:015B BB4E03  MOV  BX,034E
3E56:015E E8EF00  CALL  0250
3E56:0161 A11300  MOV  AX,[0013]
3E56:0164 E89900  CALL  0200
3E56:0167 BBB403  MOV  BX,03B4
3E56:016A E8E300  CALL  0250
3E56:016D 88E0    MOV  AL,AH
3E56:016F E88E00  CALL  0200
3E56:0172 BBB203  MOV  BX,03B2
3E56:0175 E8DB00  CALL  0250
3E56:0178 A00D00  MOV  AL,[000D]
3E56:017B E88200  CALL  0200
```

```
3E56:017E BB6203   MOV   BX,0362
3E56:0181 E8CC00   CALL   0250
3E56:0184 BB9203   MOV   BX,0392
3E56:0187 E8C600   CALL   0250
3E56:018A 90   NOP
3E56:018B B80080   MOV   AX,8000
3E56:018E 8EC0   MOV   ES,AX
3E56:0190 31DB   XOR   BX,BX
3E56:0192 BA8000   MOV   DX,0080
3E56:0195 B90100   MOV   CX,0001
3E56:0198 B80102   MOV   AX,0201
3E56:019B CD13   INT   13
3E56:019D 26   ES:
3E56:019E 803EF20101   CMP   BYTE PTR [01F2],01
3E56:01A3 740A   JZ   01AF
3E56:01A5 0E   PUSH   CS
3E56:01A6 1F   POP   DS
3E56:01A7 3E   DS:
3E56:01A8 C606DB0332   MOV   BYTE PTR [03DB],32
3E56:01AD EB0E   JMP   01BD
3E56:01AF 0E   PUSH   CS
3E56:01B0 1F   POP   DS
3E56:01B1 3E   DS:
3E56:01B2 C706DB03312E   MOV WORD PTR [03DB],
2E31
3E56:01B8 C606DD0335   MOV BYTE PTR [03DD],35
3E56:01BD BAC002   MOV   DX,02C0
3E56:01C0 B409   MOV   AH,09
3E56:01C2 CD21   INT   21
3E56:01C4 CD20   INT   20
-

3E56:0200 50   PUSH   AX
3E56:0201 88C5   MOV   CH,AL
3E56:0203 240F   AND   AL,0F
3E56:0205 0430   ADD   AL,30
3E56:0207 88C1   MOV   CL,AL
3E56:0209 D0ED   SHR   CH,1
3E56:020B D0ED   SHR   CH,1
3E56:020D D0ED   SHR   CH,1
3E56:020F D0ED   SHR   CH,1
3E56:0211 80E50F   AND   CH,0F
3E56:0214 80C530   ADD   CH,30
3E56:0217 80FD3A   CMP   CH,3A
3E56:021A 7C03   JL   021F
3E56:021C 80C507   ADD   CH,07
3E56:021F 80FD3A   CMP   CL,3A
3E56:0222 7C03   JL   0227
3E56:0224 80C107   ADD   CL,07
3E56:0227 58   POP   AX
3E56:0228 C3   RET
-
3E56:0250 1E   PUSH   DS
3E56:0251 0E   PUSH   CS
3E56:0252 1F   POP   DS
3E56:0253 3E   DS:
3E56:0254 882F   MOV   [BX],CH
3E56:0256 43   INC   BX
3E56:0257 886F   MOV   [BX],CL
3E56:0259 1F   POP   DS
3E56:025A C3   RET
-d2c0 40f

3E56:02C0 20 20 0D 0A 0D 0A 20 20-20 20 20 20 20 20 20 20   ....
3E56:02D0 20 3D 3D 3D 3D 3D 3D 3D-3D 20 D3 B2 20 C5 CC 20- = = = = = = = = .  . .
3E56:02E0 CF B5 20 CD B3 20 B2 CD-20 CA FD 20 3D 3D 3D 3D.  .  .  .  .  .  .  = = = =
3E56:02F0 3D 3D 3D 3D 0D 0A 20 20-20 20 20 20 20 20 0D 0A  = = = = .        .  .
3E56:0300 31 2E 20 B8 F9 C4 BF C2 -BC B5 C4 C6 F0 CA BC C9 1. .  .  .  .  .  .  .  .  .  .  .  .
3E56:0310 C8 C7 F8 3A 20 20 20 20-0D 0A 32 2E 20 B8 F9 C4  .  .  .  :        .  .  2. .  .  .  .
3E56:0320 BF C2 BC D7 EE B4 F3 CF-EE CA FD 3A 20 20 20 20.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  :
3E56:0330 20 0D 0A 33 2E 20 CE C4- BC FE B7 D6 C5 E4 B1 ED .  .  3. .  .  .  .  .  .  .  .  .  .
3E56:0340 D5 BC D3 C3 B5 C4 C9 C8-C7 F8 CA FD 3A 20 20 20.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  :
3E56:0350 20 0D 0A 34 2E 20 C3 BF -B4 D8 C9 C8 C7 F8 CA FD .  .  4. .  .  .  .  .  .  .  .  .  .
3E56:0360 3A 20 20 20 20 20 0D 0A-35 2E 20 B4 D8 BA C5 BA :        .  .  5. .  .  .  .
3E56:0370 CD C6 E4 B6 D4 D3 A6 C9-C8 C7 F8 B5 C4 BB B8 CB.  .  .  .  .  .  .  .  .  .  .  .  .
3E56:0380 E3 B9 AB CA BD 3A 20 28-B4 D8 BA C5 2D 32 29 20.  .  .  .  .  :  (  .  .  .  .  -  2  )
3E56:0390 78 20 20 20 20 2B 20 20-20 20 20 20 0D 0A 36 2E  x        +        .  .  6.
3E56:03A0 20 D3 B2 C5 CC B5 C4 D7-DC C9 C8 C7 F8 CA FD 20.  .  .  .  .  .  .  .  .  .  .  .  .  .
3E56:03B0 3A 20 20 20 20 20 20 0D-0A 37 2E 20 CE C4 BC FE :        .  .  7. .  .  .  .
3E56:03C0 B7 D6 C5 E4 B1 ED 20 46-41 54 20 C1 AC BD D3 D7.  .  .  .  .  .  FAT .  .  .  .  .
3E56:03D0 D6 B5 C4 D7 D6 BD DA CA-FD 3A 20 20 20 20 20 D7.  .  .  .  .  .  .  .  :        .
3E56:03E0 D6 BD DA 0D 0A 0D 0A 20-20 20 D7 A2 3A 20 D2 D4.  .  .  .  .  .  .  .  .  .  .  .  .  .
3E56:03F0 C9 CF CA FD BE DD BE F9-CE AA CA AE C1 F9 BD F8.  .  .  .  .  .  .  .  .  .  .  .  .  .
```

```
3E56:0400 D6 C6 20 20 20 01 0A 24-00 00 00 00 00 00 00 00  ... .$........
```

运行结果如下：

```
** 硬盘系统参数 **
1. 根目录的起始扇区:11
2. 根目录最大项数:0400
3. 文件分配表占用的扇区数:08
4. 每簇扇区数:10
5. 簇号和其对应扇区的换算公式:
      (簇号-2)×10+51
6. 硬盘的总扇区数:0317
7. 文件分配表 FAT 连接字的字节数:1.5字节
注:以上数据均为十六进制
C>-
```

ASC Ⅱ

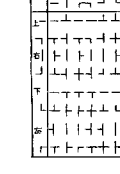
应武斌 (93.1.13,107)

多窗口重叠时窗口边界的绘制

在应用程序的界面设计中，窗口的使用是很普遍的。当在屏幕上重叠多个窗口时，窗口的边界该如何绘制呢？我们一般用制表符“┌”“┐”、“┘”、“└”、“─”、“│”来绘制窗口边界，如图1所示。窗口2部分覆盖了窗口1，在边界的相交处(A,B)窗口2的边界符“ ”和“│”分别覆盖了窗口1的边界符“│”和“ ”。由于制表符“ ”和“│”之间留有空隙，所以这样的画面不能令人满意。若在A、B两个相交点处分别用制表符“┴”和“┤”代之，如图2所示，则效果大为改善。

图1　　图2

表1　制表符的各种组合情况



为了达到这样的重叠效果，我们在画窗口边界时就不能简单处理，而应加上一定的判断，根据当前位置的字符作相应的处理。若当前位置为非制表符，则可直接以窗口边界符覆盖该字符；若当前位置为制表符，则应根据该制表符和要画的边界符决定正确的制表符。表1给出了不同组合时制表符的选择，其中纵栏为要画的边界符，横栏为当前位置制表符，交汇点为应画的制表符。例如，要画边界符“┐”，当前位置为制表符“┌”，从表中查到应画制表符“┬”。

笔者在实际应用中根据上述原理用C语言编制了绘制窗口边界的程序，并在GW386上用MSC5.0调试通过。该程序能在多窗口重叠的情况下正确绘制窗口边界。

```
#include<dos.h>
static union REGS regs;
void set_cur(row,col)
int row,col;
{
    regs.h.ah=2;
    regs.h.dh=row;
    regs.h.dl=col;
    regs.h.bh=0; /* 0 page */
    int86(0x10,&regs,&regs);
}
void get_chr()
{
regs.h.ah=8
regs.h.bh=0
int86(0x10,&regs,&regs);
}
void put_chr(code,attr)
int attr;
char code;
{
    regs.h.ah=9;
    regs.h.al=code;
    regs.h.bl=attr;
    regs.h.bh=0;
    regs.x.cx=1;
    int86(0x10,&regs,&regs);
}
void draw_border(brow,bcol,erow,ecol,attr)
int brow,bcol,erow,ecol;
}
```

```
    register int i;
    int ch;
/* ┌   */
    set_cur(brow,bcol);
    get_chr();
    ch=regs.h.al;
    switch(ch){
        case 191;
        case 196;
        case 194:put_chr(194,attr);break;
        case 192;
        case 179;
        case 195:put_chr(195,attr);break;
        case 217;
        case 193;
        case 180;
        case 197:put_chr(197,attr);break;
        default :put_chr(218,attr);break;
    }
/* up — */
    for(i=bcol+1;i<=ecol;i++);
    set_cur(brow,i);
    get_chr();
    ch=regs.h.al;
    switch(ch){
        case 217;
        case 192;
        case 179;
        case 197;
        case 180;
        case 195;
        case 193:put_chr(193,attr);break;
        default :put_chr(196,attr);,break;
    }
/* ┐   */
    set_cur(brow,ecol);
    get_chr();
    ch=regs.h.al;
    switch(ch)
        case 218;
        case 196;
        case 194:put_chr(194,attr);break;
        case 179;
        case 217;
        case 180:put_chr(180,attr);break;
        case 192;
        case 193;
        case 195;
        case 197:put_chr(197,attr),;break;
        default :put_chr(191,attr),;break;
    }
/* right | */
    for(i=brow+1;i<=erow;i++){
        set_cur(i,ecol);
        get_chr();
        ch=regs.h.al;
        switch(ch){
        case 218;
        case 192;
        case 196;
        case 197;
        case 193;
        case 194;
        case 195:put_chr(195,attr);break;
        default :put_chr(179,attr),;break;
        }
    }
/* ┘   */
    set_cur(erow,ecol);
    get_chr();
    ch=regs.h.al;
    switch(ch){
        case 196;
        case 192;
        case 193:put_chr(193,attr);break;
        case 191;
        case 179;
        case 180:put_chr(180,attr);break;
        case 218;
        case 194;
        case 195;
        case 197:put_chr(197,attr);break;
        default :put_chr(193,attr);break;
    }
/* down — */
    for(i=ecol-1;i>bcol;i--){
      set_cur(erow,i);
      get_chr()
      ch=regs.h.al;
      switch(ch){
        case 218;
        case 191;
        case 179;
        case 197;
        case 180;
        case 195;
        case 194:put_chr(194,attr);break;
        default :put_chr(196,attr);break;
        }
    }
/* └   */
    set_cur(erow,bcol);
    get_chr();
```

```
    ch=regs.h.al;
    switch(ch){
        case 218;
        case 179;
        case 195;put_chr(195,attr);break;
        case 196;
        case 217;
        case 193;put_chr(193,attr);break;
        case 191;
        case 194;
        case 180;
        case 197;put_chr(197,attr);break;
        default ;put_chr(192,attr);break;
    }
/* left |*/
for(i=ecow-1;i>brow;i--){
    set_cur(i,bcol);
    get_chr();
    ch=regs.h.al;
    switch(ch){
        case 191;
        case 217;
        case 196
        case 197;
        case 194;
        case 193;
        case 180;put_chr(180,attr);break;
        default ;put_chr(179,attr);break;
    }
}
```

陈　剑　(93.1.13.109)

AST/286内存板故障的维修

由于AST/286在总线结构设计上，增设了特快扩展槽，因此，它的内存部分并不像其它AT兼容机那样，直接设计在主机板上，而是提供了一块可实现零等待状态的快速内存板，插在主机板带有特快扩展槽的I/O通道上。

标准的AST/286微机，一般为用户提供1MB内存。其内存板上使用的RAM芯片是41256－10，内存的控制线路主要使用74系列芯片和PAL芯片。AST/286的512KB内存，即RAM芯片U1—U18，是直接焊在内存板上的，512KB以上的RAM芯片都装有插座。

在内存自检过程中，若显示错误信息：MEMORY PARITY ERROR AT ××：××××；×××× FOUND FFFF EXPECTED FFFF，或者显示内存容量与实际配置不符，这类内存板故障，是由512KB以上RAM芯片损坏所致，这部分RAM芯片由于装有插座，因此，采用替换法可以很快查出故障芯片。应该注意的是，如果每次显示错误信息的地址代码不固定，或者显示的内存容量与实际配置不符，且每次显示的内存容量也不固定，这样的内存板故障，不在上述范围之内。

现通过实例说明，AST/286内存板其它故障的维修。

例

故障现象

一台AST/286微机，开机后显示屏无任何信息。

故障分析及维修

更换这台微机的内存板，主机工作正常，因此确定内存板有故障。

对于AST/286内存板，如果512KB内存以上的RAM芯片有故障，不会造成主机开机后无显示；所以根据上述故障现象，基本可排除512KB以上RAM芯片有故障的可能性，问题发生在512KB内存，即RAM芯片U1　U18或者内存控制芯片上。首先，用万用表测量内存控制芯片未发现有管脚短路或开路的情况。接着，用示波器检测RAM芯片U1－U18，发现其输入端的数据、地址信号均正常，而输出端，即RAM芯片（41256－10）的第14管脚，为3V左右的高电平。由此推测，故障的产生是由于某块RAM芯片损坏，造成输出信号失常所致，但由于U1－U18的输出端均为3V左右的高电平，所以，一时无法确定哪块芯片有故障。

用示波器探针抵住U1的第14管脚，重新启动主机，在开机的瞬间，U1的第14管脚为3V左右的高电平，接着，产生一个脉冲信号，但这个脉冲信号瞬间便消失，又变成3V左右的高平。使用上述方法检测其它RAM芯片，检测结果，除U6外，其余芯片的管脚输出情况与U1相同。而U6芯片的输出管脚，在开机的瞬间产生一个3V左右的高电平后，不再有任何变化，没有信号输出，因此，断定该芯片坏。更换U6，主机工作正常，故障排除。

例二

故障现象

一台 AST/286微机，有时开机后显示屏无任何信息；有时在使用过程中，随机性地出现下述错误信息：

RAM PARITY ERROR，CHECKING FOR SEGMENT ADDRESS…

OFFENDING SEGMENT：××××

同时，主机死锁，且每次显示的地址代码不固定。经诊断，确定为内存板故障。

故障分析及维修

由于这种故障带有随机性，不容易发现问题的症结，维修起来有一定难度，因此，称之谓"软故障"。

分析故障现象，因为每次出现错误信息时，地址代码不固定，所以不像 RAM 芯片本身故障，问题可能出在内存控制电路上。在机器出现故障现象时，用示波器检测 RAM 芯片的第14管脚，发现它们在主机启动时，均有脉冲信号输出，因此证明上述分析是正确的。下是，进一步仔细检测内存控制电路，发现机器工作正常时，DL-一 100ns 延迟线的管脚波形很规整，当机器出现故障时，DL 各管脚波形起初很杂乱，进而又变得十分微弱。

内存工作时，列选通信号 CAS 要在行选通信号 RAS 之后发送，这一滞后过程是通过延迟线 DL 实现的。如果延迟线性能不好，便会造成行、列地址的选通信号失常。鉴于上述原因，更换了 DL，结果，故障排除。

例三

故障现象

一台 AST/286微机，不能读软盘。主机自检时，屏幕提示如下信息：

DISK BOOT FAILURE，INSERT SYSTEM DISK AND PRESS ENTER，

故障分析及维修

从故障现象上看，似乎是软盘驱动器或主机板上软驱接口的问题，但在更换 AST/286内存板后，主机恢复正常，因此，确定为内存板故障。

AST/286内存板上只有 RAM 芯片和内存控制芯片，插上此故障内存板后，不能读软盘，这说明该内存板影响了数据、地址总线，造成主机板的外围设备接口线路失常，从而使软驱不能读盘。

根据上述分析，决定从检测主机板的 I/O 通道入手，找寻受内存板影响而失常的总线信号，然后顺藤摸瓜，查出内存板上的故障芯片。当用示波器检测到 I/O 槽 A29(地址总线 SA2)时，发现该总线信号波形幅度偏低、关机后，用万用表测量，这条总线与内存板上的 U105(74LS682)芯片的第17管脚及 U102(74F157)芯片的第2管脚相通。测量这两个管脚的内阻，发现它们对地短路(内阻偏低，但未与地完全导通)，至此，将故障范围缩小到这两块芯片上。接着，断开 U105的第17管脚与 U102的第2管脚之间的连线，再测量内阻，U105的第17管脚内阻恢复正常，而 U102第2管脚仍短路，因此确定 U102有故障。

更换 U102后，加电测试，发现机器不再提示前文所述的错误信息，但软驱磁头归零后，便不再动作，屏幕上只有一个光标。继续检测 I/O 总线，再没有找出明显失常的信号，致使维修工作陷入困境。

起初怀疑，U102的损坏，将与之相连的 U105也损坏，但更换 U105后，故障并未排除。进一步分析上述维修过程，数据选择器 U102(74F157)第2管脚对地短路，造成地址总线 SA2失常，致使不能读软盘。更换 U102后，虽未能彻底解决问题，但故障情况有所改善，由此推测，是否内存板上别的数据选择器也有损坏，从而影响其它地址总线，于是，试验性地更换了与 U102相邻的另一块数据选择器 U103，果然不出所料，更换 U103后，软驱工作正常，故障完全排除。

范广宇 (93.1.13.115)

怎样实现 LEVEL Ⅱ—COBOL 索引文件到 MS—COBOL 索引文件的转换

我行开始用微机办理储蓄事后监督业务时，有的储蓄所已经把微机应用于柜台业务上，这就为计算机软件人员提出能否解决储蓄业务数据一次录入再次利用的问题。

储蓄柜台业务数据是以 LEVEL Ⅱ—COBOL 索引文件形式存入微机，储蓄事后监督业务数据是以 MS—COBOL 索引文件形式存入微机，笔者在 M290单用户微机和 M300多用户微机下实现了 LEVEL Ⅱ

—COBOL 索引文件到 MS—COBOL 索引文件的转换。

转换的基本方法

1. 在由 CCDOS2.1操作系统支持的 M290微机中建立一个中间顺序文件，读取 LEVEL Ⅱ—COBOL 索引文件的所有记录并写入中间文件中，程序结构和执行过程如下：

BXA 为顺序文件，BXC 为索引文件。

```
FILE SECTION.
FD BXA LABEL RECORD IS STANDARD.
01 BXA—REC.
    02 BXA010 PIC X(1).
    02 BXA020 PIC 9(1).
    02 BXA030 PIC 9(1).
    02 BXA040 PIC 9(1).
    02 BXA050 PIC 9(9).
    02 BXA060 PIC X(8).
    02 BXA070 PIC 9(7)V99.
    02 BXA080 PIC 9(7)V99.
    02 BXA090 PIC 9(5)V999.
    02 BXA100 PIC 9(6).
FD BXC LABEL RECORD IS STANDARD.
01 BXC—REC.
    02 BXC010 PIC X(1).
    02 BXC020 PIC 9(1).
    02 BXC030 PIC 9(9) COMP.
    02 BXC040 PIC X(8).
    02 BXC050 PIC 9(7)V99 COMP.
    02 BXC060 PIC 9(7)V99 COMP.
    02 BXC070 PIC 9(5)V99 COMP.
    02 BXC080 PIC 9(6) COMP.
    02 BXC090 PIC 9(2) COMP.
    02 BXC100 PIC 9(2) COMP.
    02 BXC110 PIC 9(6) COMP.
PROCEDURE DIVISION.
  ACCEPT I.
  DISPLAY SPACE AT 0101 DISPLAY " AT 0101.
  OPEN INPUT BXC OUTPUT BXA.
ST1.  READ BXC NEXT AT END GO TO END1.
    IF BXC010 = "1" OR "2" GO TO ST1.
    MOVE BXC010 TO BXA010.
    IF BXC010 = "2" OR "5" OR "6" MOVE "1" TO
BXA010.
    IF BXC020 = OR 1 MOVE BXC020 TO BXA020.
    MOVE 0 TO BXA030 BXA040.
    MOVE BXC030 TO BXA050.
    MOVE BXC040 TO BXA060.
    MOVE BXC050 TO BXA080.
    MOVE BXC070 TO BXA090.
    MOVE BXC110 TO BXA100.
    MOVE 0 TO BXA070.
    WRITE BXA—REC.
    GO ST1.
END1. CLOSE BXC BXA.
    STOP RUN.
```

在由汉化 XENIX System V 支持的 M300机中，把读取中间文件的内容写入 MS—COBOL 索引文件中，程序结构和执行过程如下：BXB 为索引文件。

```
  FILE SECTION.
    FD BXB LABEL RECCORD IS STANDARD
      VALUE OF FILE—ID IS F—BXB.
    01 BXB—REC.
    02 BXB010 PIC X(1).
    02 BXB020 PIC 9(1).
    02 BXB030 PIC 9(1).
    02 BXB040 PIC 9(1).
    02 BXB050 PIC 9(9).
    02 BXB060 PIC X(8).
    02 BXB070 PIC 9(7)V99 COMP—3.
    02 BXB080 PIC 9(7)V99.
    02 BXB090 PIC 9(5)V999 COMP—3.
    02 BXB100 PIC 9(6)      COMP—3.
    02 BXB110 PIC 9(4) COMP—3.
PROCEDURE DIVISION.
FF1.
    OPEN INPUT BXA OUTPUT BXB.
    READ BXA AT END GO TO EE.
    MOVE ZERO TO BXB—REC.
    MOVE BXA010 TO BXB010.
    MOVE BXA020 TO BXB020.
    MOVE BXA030 TO BXB030.
    MOVE BXA040 TO BXB040.
    MOVE BXA050 TO BXB050.
    MOVE BXA060 TO BXB060.
    MOVE BXA070 TO BXB070.
    MOVE BXA080 TO BXB080.
    MOVE BXA090 TO BXB090.
    MOVE BXA100 TO BXB100.
    MOVE 0      TO BXB110.
    WRITE BXB—REC INVALID KEY "写文件出错;"
    GO TO FF1.
EE.
    CLOSE BXA BXB.
    STOP RON.
```

孙玉刚 (93.1.20.99)