

现代计算机

'95.8

总第 43 期

原名《电脑与微电子技术》

现代计算机杂志社出版

MODERN COMPUTER



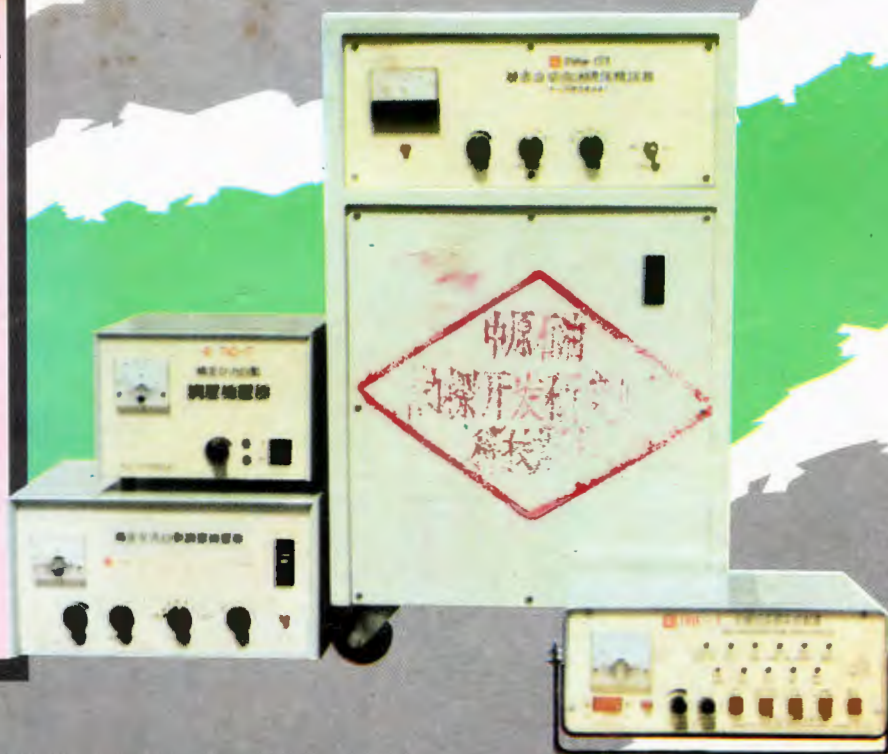
单、三相自动调压 精密交流稳压器

PRECISE AUTOMATIC AC VOLTAGE STABILIZERS

单相 TND-T 型系列
三相 TNS 型系列

中山大学电器设备厂是专业生产稳压电源的厂家，
是国家教委审定的合格企业，
自 80 年代初，
工厂一直依靠大学的先进科学技术，
秉承“信誉为本、
质量取胜”的宗旨，
精益求精地生产高质量的各种型号单、
三相精密交流自动调压稳压器，
其中 TND-T 型系列精密交流自动调压稳压器，
90 年度荣获国家教委科学技术进步奖二等奖。
产品广泛应用于各类高科技电子仪器
以及三资企业等用电设备领域。
在全国 20 多个省市设有经销维修网点。
十多年来，
真诚地服务於全社会，
受到用户的欢迎。

厂址：广州市新港西路 135 号中山大学西南区
电话：4186882
电挂：8775 电器设备厂
邮码：510275



中山大学电器设备厂

40M² LED 全彩色电视广告屏 矗立在中国出口商品交易会门前

广州中鸣公司隆重推出





广州市科教电脑设备有限公司

Guangzhou Science & Education Computer Equipment Co. Ltd.

COMPAQ 高档笔记本专卖
美国康柏电脑

不
求
价
格
第
一

- * COMPAQ LTE Elite 4/75(486DX4 - 75)
8M, 510M, 1.44M TFT
- * COMPAQ LTE Elite 4/50(486DX2 - 50)
8M, 510M, 1.44M TFT
- * 8M RAM For COMPAQ Elite
- * COMPAQ Smart Station Base For LTE Elite 4/40/50/75cx
- * COMPAQ 15" SVGA 151FS
- * COMPAQ Keyboard
- * COMPAQ Mouse
- * Kingston PCMCIA Fax/Modem 14.4K B/S
- * Xircom PCMCIA Ethernet II PS TP

但
求
服
务
最
好

地 址:五山路华师科技大楼 157—159 号, (邮政信箱 1233 号, 邮编:510630)

电 话:7549981、7549982、7549983、7549984、7549985、7549986、7549987、7549988 Fax:7549989

展销部:广州天河体育东路 39 号天宝大厦二楼新一代电脑城 A200 室 Tel:7548818

科教电脑多媒体中心

集 各种声霸卡、CD—ROM、音箱……

供 各种解压卡、视霸卡、TV 卡……

售 14" TOPCON 彩显、LQ—1600K、LQ300K 打印机……

欢迎批发、联销、合作!

地址:广州五山路科技街二栋二楼 222 号 Tel:7548485、5510446(Fax)

广州中鸣 一鸣惊人

——广州中鸣显示技术工程有限公司进展

中鸣显示技术工程公司是中山大学与香港得时利公司合作的结晶，专业研制、生产户内外的各种显示屏。由于拥有一批出色的科研人员和高效率的生产基地，公司在短短的几年里取得了很大的发展，产品遍布省内外。

OFC 系列户外全彩色 LED 显示屏是中鸣显示技术工程公司的最新产品，集最新发光材料、视频处理技术、多媒体技术、耐候处理技术于一身，被列为 1994 年国家级重点新产品。

全彩色 LED 显示屏是以红、绿、蓝三种颜色的发光二极管作为发光材料，它比其它的显示元件更适合于户外的工作环境，所以 LED 显示屏成为近年户外广告媒体的新宠。但是，由于技术上的限制，一直不能生产出适合实际使用的蓝色 LED，国内外的显示屏全都是红黄的色调，极大地限制了广告的制作，也影响了 LED 显示屏的发展。

随着半导体技术的发展，94 年下半年，中鸣公司与美国合作公司紧密配合，采用最新出品的蓝色 LED，研制成功了全彩色 LED 显示系统，填补了国内的空白，制作了国内第一块户外彩色 LED 显示屏。经测试，它具有亮度高、层次丰富、色彩平衡、性能稳定的特点，使用寿命长更是其它代用品所无法比拟的。它的出现，给电子显示行业带来了震动，同时也为之展示了更为广阔、更为美好的前景。

本着精益求精的原则，中鸣人潜心研究，对整个显示系统进行了科学论证，不断完善，特别是在结构和耐候方面，更是设计出适合不同纬度地区气候特点的系统，对耐热、耐寒、防紫外、抗老化等方面的性能有很大提高，使 OFC 系列户外全彩色 LED 显示屏更趋完善。

1994 年 11 月，中鸣公司参加了 94' 北京国际广告“四新”展示交易会。会上，作为唯一的 LED 全彩色显示屏，OFC—28 全彩色 LED 显示屏（5 平方米局部）吸引了众多的行家，并以丰富的色彩、生动的图象、逼人的光焰和良好的工艺、结构获得一致的好评。（王光英同志参观时特意驻足中鸣公司展位，留心听取了介绍，并给予肯定）。

1995 年 4 月，中鸣公司制作的全彩色 LED 显示屏矗立在广州外贸中心（交易会）的正门，并在春交会的开幕式上正式开播，以清晰、稳定的图像直播了春交会开幕式的盛况，博得众多来宾的赞誉。该屏开播以来，经历了风雨、雷电和酷暑的考验，每天播出十多小时，一直正常运行。

1995 年 5 月，中鸣公司通过美国代理公司参加了在多伦多举行的 NESA（Northern American Electronic Sign Association）展示会，在有美国、加拿大、日本、韩国、台湾等国家、地区专业公司参加的情况下，中鸣送展的全彩屏仍引起全场瞩目，为开拓海外市场迈出了成功的一步。

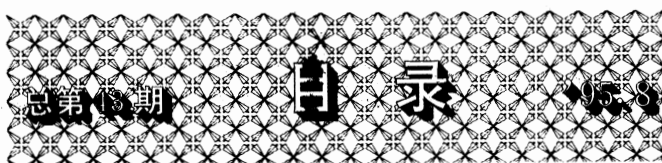
公开发行 1995年8月20日出版

主 办 中山大学
编辑出版 现代计算机杂志社
主 编 张纬铮
联系地址 广州新港西路 135 号中山大学
邮政编码 510275
电 话 4186300—6540
刊 号 CN44—1415/TP
邮发代号 46—121
广告许可 粤 010329 号
广告总代理: 广东省广告公司
电话: 7752254 传真: 7778225
印 刷 中山大学印刷厂
总 发 行 广东省报刊发行局
订 阅 处 全国各地邮局
每期订价 2.00 元 邮购价 3.00 元

本刊图文版权所有 未经允许不得转载

下 期 要 目

- 绘图软件中动态仿真调色板的设计
- BP 算法的模拟程序
- Net Ware 386 内存管理优化方法
- PC 机与 8031 的并行数据通讯及自校正标识符算法
- 多媒体视频与音频同步的探讨
- Drafting 实现机械制图辅助标注
- 通用数字图象处理软件在 PC 机上的研制
- 利用 CCED 进行彩色打印的尝试
- FOXBASE 伪编译文件的还原



☆ 研制·开发·应用 ☆

- 一种高精度温度测控系统的研制方法 夏益青②
灰色预测软件 朱诗兵 李迎春⑤

☆ 网络与通信 ☆

- 程控交换机与微机的串行通信 朱诗兵 李迎春⑦
用查表法实现快速循环冗余位校验计算 董 云⑨

☆ 软件纵横 ☆

- 如何实现显示字库的“零内存占用”
..... 杨 戈 王成义 苏庆会⑩
CMOS 及硬盘引导扇区读写示例 黄焕如⑭
非多媒体环境下 FLI 动画图像文件的应用
..... 李学春 苟列红⑱

☆ 实用技术 ☆

- 再谈《单片机与 RS-232 串行通讯口联接》
..... 高国权⑳
在 Turbo c 中实现窗口坐标绘图功能 习华勇㉑
高级语言直接在打印机上绘图 任铁良㉒
在 C 程序中使用 DOS 内部命令 杜小平 刘运㉓
微机系统掉电保护的实用技术 王成义㉔

☆ 编程技巧 ☆

- FOXBASE+ 数据库视窗编辑器的设计
及实现 张永强㉖

☆ 善工与利器 ☆

- 巧用 DOSKEY 黄中伟㉘

☆ 经验与交流 ☆

- 信息系统中库文件的设计原则与方法 韦沛文㉚
低主频微机使用 WPS 提高速度的技巧 林治洋㉛
ICL232 做液晶对比度电源使用一例
..... 刘文浩 李 华㉜
也谈键盘功能的自动切换 车光宏㉝
如何在 XENIX 系统下安全关机 谭仕谋㉞



一种高精度温度测控系统的研制方法

东南大学 动力工程系 [210018] 夏益青

提 要:本文介绍了一种高精度温度测量、控制系统的研制方法。采用可控硅调功、过零触发加热器和物流 PID 调节相结合的手段,达到使物流精确恒温的目的。

系统具有多点 I/O、超限报警、打印的功能和可靠的抗干扰能力。

关键字: 温度测量、温度控制、单片机应用

本系统为有限元温度场高精度控制而设计,实测准确度为 $\pm 0.5\%$ (测量准确度以满量程计,不包括传感器误差),线性度 $\leq 0.2\%$ 。其结构大体上可分为传感放大、A/D、D/A 转换、8031 单元、人机接口和执行部件几个部分,下面分述之。

一、传感放大与 A/D 转换

传感放大部分的传感器采用铂电阻,为消除导线电阻变化对测温系统的影响,接成三线制电路(图1)。选择某一被控温度作为基准温度,调节电桥使其输出值为0,当实际温度偏离此基准时,有电流从桥路输出。输出的多路温度信号经过

74LS151 后,送入隔离放大器 277。74LS151 是多路开关,其通道选择编码线 C、B、A 三端分别接数据总线的 D2、D1、D0。单片机 8031 通过这三根数据总线选择好某一路传感器后,由 P3.5 口发低电平使 74LS151 的 G 有效,对应的测值便出现在 Y 端。此信号经 277 隔离放大后,送入 A/D 部分。

隔离放大器 277 的结构可分为两个各自屏蔽的模块。前级模块中的输入运放和调制器将输入的直流信号放大后,调制成一定频率的交流信号输出,6、7、8 脚为运放的调 0 端;后级模块中的解调器和输出运放将耦合的交流信号变成直流信号,经

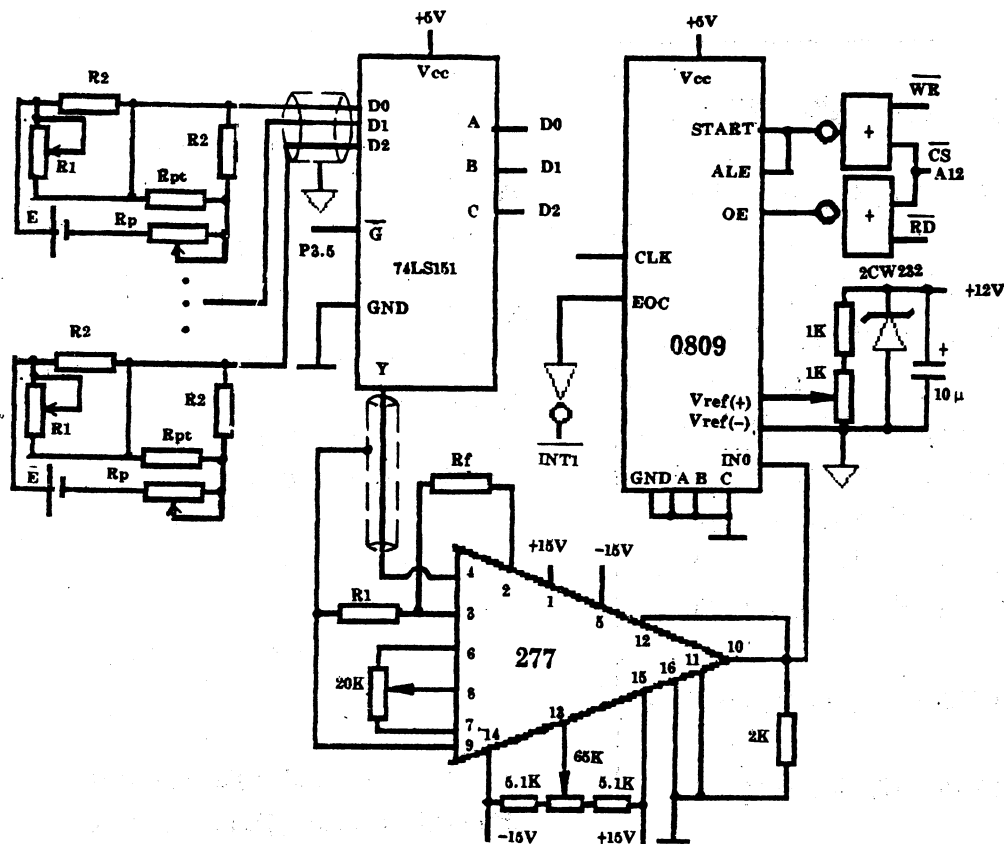


图 1

放大后输出。12、10 引脚间接不同的电阻可得到不同的闭环增益,如直接相连,则增益为 1。11 脚必须与外接电源地的 16 脚形成通路。由于该芯片隔离电阻很高,故共模信号在回路中产生的电流很小,从而提高了共模抑止比。图 1 中的 277 增益约为 $1 + R_f/R_1$ 。

ADC 选用 0809 八位转换芯片,片选信号 \overline{CS} 接地址总线的 A12。当 \overline{CS} 和 \overline{WR} 都为低电平有效时,起动转换。转换结束信号 EOC 经反向后,触发 8031 的 $\overline{INT1}$ 中断。8031 获知该中断以后,发读信号 \overline{RD} ,并选通 0809 的输出允许端 OE,使之将转换好的数字信号 D7~D0 传到数据总线上。

8031 单片机电路部分不赘述。

二、D/A 转换部分

DAC 选用 0832 芯片,它的 $\overline{WR1}$ 、 $\overline{WR2}$ 、 \overline{XFER} 都接地,为“直通”式,其片选信号 \overline{CS} 接地址总线的 A14。当 8031 选中该芯片后,D0~D7 数字量即由 OUT0、OUT1 口输出(图 2)。

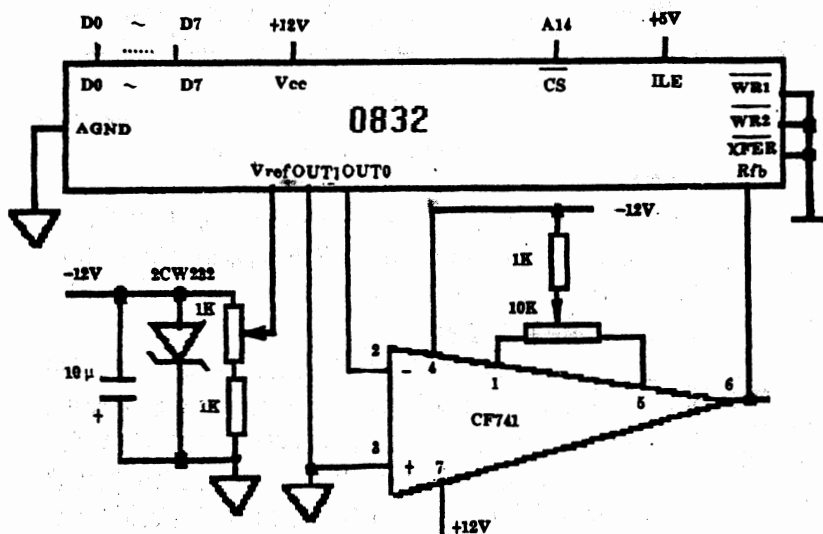


图 2

其中有两个地方需要加以重视:一是转换数据的有效保持时间不应小于 90 纳秒,否则将锁存错误数据。二是为了保持输出电流线性度,两个电流输出点的电位应尽可能接近地电位(0V)。按图 2 接法,当 $V_{ref}=10V$ 、 I_{out0} 和 I_{out1} 上若有 1mV 的电压时,将产生 0.01% 的线性误差。

CF741 是高精度通用运放,输出模拟量经它放大后,驱动物流控制电磁阀。该运放主要特点是不要求外部频率补偿,具有短路保护、失调电压调零能力,并有较宽的共模电压范围,不会在使用中出现自锁现象。

三、键盘输入部分

系统采用 8031 的串口线 RXD、TXD 扩展矩阵键盘(图 3)。设定串口工作于方式 0,成为同步移位寄存器,波特率 $= F_{osc}/12$ 。数据由 RXD 输出,同步移位时钟由 TXD 输出。74LS164 为串入并出移位寄存器,其对外输出由串行输入数据决定。8031 读入 P1.0、P1.1 口状态时,必须等待串行数据发送完毕,这只要检查 TI 是否置位,如置位则表示发送完毕。

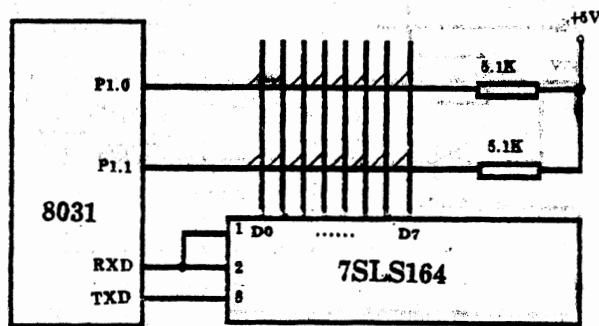


图 3

四、打印机接口

系统配有微型打印机接口,以便打印出当前各个测点值(实时打印功能),也可以打印出历史数据库值。系统用 6116RAM 作为历史数据库。

一般微型打印机的控制方式有查询和中断两种。本系统因 $\overline{INT0}$ 、 $\overline{INT1}$ 已分别被掉电中断、0809 转换结束中断占用,又不希望采取查询的方式,浪费机时,所以采用了基于查询的伪中断方式。

通常系统每次打印的内容,都有统一的格式,如:“时间,测点号,测值……”等,换言之,每次打印上述单条信息的时间是一个常数 T。所以在每次起动打印时,给定时/计数器 1 赋一个时间常数 T' , T' 略大于 T,这样每次定时/计数器 1 中断后,8031 查询打印机的“BUSY”线信号是否为“空闲”,一般情况下应该是“空闲”的,8031 继续向打印机发打印内容;反之,若“BUSY”线信号为“忙”,则要等到下一次定时/计数器 1 中断,8031 再查询“BUSY”线的状态。

五、可控硅调功、过零触发电路

本系统所用的加热器为电热丝,通过 TE03,过零触发可控硅(下简称 SCR),达到使物流恒温的目的。

灰色预测软件

国防科工委指挥技术学院 朱诗兵 李迎春

摘要: 本文通过阐述灰色建模的基本原理,编写了灰色预测软件,同时运用该软件对其进行实例分析。

关键词: 灰色系统,灰色预测,时间序列预测

一、引言

部分信息已知,部分信息未知的系统称为灰色系统。比如社会系统、经济系统、生态系统、农业系统、商业系统等抽象系统没有物理原型,不清楚系统的作用机制,很难判断信息的完备性,难以对系统关系、结构作精确描述。于是人们所建立的模型只能是原系统的代表,并且只在某种准则下成立。这类抽象系统称为本征性的灰色系统。

基于灰色系统理论的 GM(1,1)模型的预测,称灰色预测。灰色理论的微分方程型模型称为 GM 模型,G 表示 grey(灰),M 表示 Model(模型),GM(1,1)是 1 阶的、1 个变量的微分方程型模型。灰色预测可分五类,即:数列预测、灾变预测、季节灾变预测、拓扑预测、系统综合预测等。通过对这些方面的灰色预测,提高了预测精度,从而更好的采取措施,使问题得到更好的解决。

二、灰色建模的基本原理

设 k 为时间序号,给定下述的时间数据序列:

$$x^{(0)} = (x^{(0)}(1), x^{(0)}(2), \dots, x^{(0)}(k), \dots, x^{(0)}(n))$$

其中: $x^{(0)}(k) \geq 0, k=1, 2, \dots, n$

$x^{(0)}$ 表示对应于时间序号 k 的原始序列作它的一阶累加生成 AGO:

$$x^{(1)} = AGO x^{(0)} = (x^{(1)}(1), x^{(1)}(2), \dots, x^{(1)}(k), \dots, x^{(1)}(n))$$

其中: $x^{(1)}(k) = \sum_{i=1}^k x^{(0)}(i), k=1, 2, \dots, n$

作 $x^{(1)}$ 的一阶均值生成 AVG:

$$Z^{(1)} = AVG x^{(1)} = (Z^{(1)}(2), Z^{(1)}(3), \dots, Z^{(1)}(k), \dots, Z^{(1)}(n-1))$$

其中: $Z^{(1)}(k) = 0.5 * [x^{(1)}(k) + x^{(1)}(k-1)], k=2, \dots, n$

建立白化形式的微分方程 GM(1,1)

$$dx^{(1)}/dk + ax^{(1)} = u$$

其中: a, u 为待定参数。

解 GM(1,1), 记参数列 $a = \begin{pmatrix} a \\ u \end{pmatrix}$, 按最下二乘法解

得:

$$a = (B^T B)^{-1} B^T Y_N$$

$$\text{其中: } B = \begin{bmatrix} -Z^{(1)}(2) & 1 \\ -Z^{(1)}(3) & 1 \\ \dots & \dots \\ -Z^{(1)}(n) & 1 \end{bmatrix}, Y_N = \begin{bmatrix} x^{(0)}(2) \\ x^{(0)}(3) \\ \dots \\ x^{(0)}(n) \end{bmatrix}$$

则白化形式微分方程的解为:

$$x^{(1)}(k+1) = (x^{(0)}(1) - u/a)e^{-ak} + u/a$$

还原即进行累加生成的逆运算:

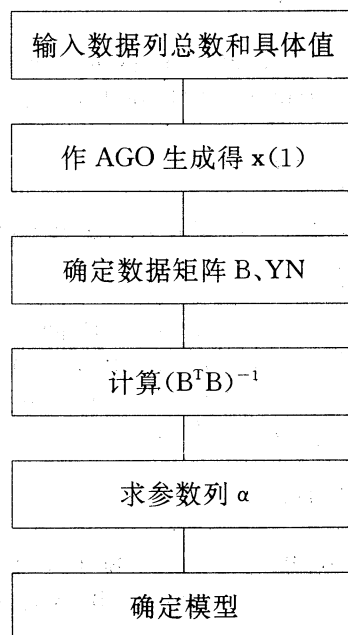
$$x^{(0)}(k) = x^{(1)}(k) - x^{(1)}(k-1)$$

所以 $x^{(0)}$ 的预测模型为:

$$x^{(0)}(k+1) = \begin{cases} (1-e^a)(x^{(0)}(1) - \frac{u}{a})e^{-ak}, & k > 0 \\ x^{(0)}(1), & k = 0 \end{cases}$$

三、软件编程

通过对灰色建模原理的分析,其相应的流程图如下:



部分程序清单如下:(input()函数是文本模式下的键盘接收函数)

```
#define NUMBER 10
main()
{
    register int i,j;
    double x0[NUMBER],x1[NUMBER],get2[NUMBER],Z[NUMBER];
    double u=0,a=0,Bt_B[2][2],BTB;
    char *get_data,ch[NUMBER];
    int key;
    printf("\n 请输入统计数据总数:");
    scanf("%d",&key);
    for(i=0;i<key;i++)
    {
        printf("\n 初始值[%d]=",i);
        get_data=input(NUMBER);
        for(j=0;j<strlen(get_data);j++)
            ch[j]=get_data[j];
        ch[j]='\0';
        x0[i]=(double)atof(ch);
    }
    /* AGO 生成 */
    x1[0]=x0[0];
    for(i=1;i<key;i++)
        x1[i]=x1[i-1]+x0[i];
    /* 确定数据矩阵 B */
    for(i=0;i<key-1;i++)
        Z[i]=(-0.5)*(x1[i]+x1[i+1]);
    Bt_B[0][0]=Bt_B[0][1]=Bt_B[1][0]=Bt_B[1][1]=0.0;
    for(i=0;i<key-1;i++)
    {
        Bt_B[0][0]=pow(Z[i],2)+Bt_B[0][0];
        Bt_B[0][1]=Bt_B[1][0]=Bt_B[0][1]+Z[i];
    }
    Bt_B[1][1]=key-1;
    /* 计算(BTB)-1 */
    BTB=(double)((Bt_B[0][0]*Bt_B[1][1])-(Bt_B[0][1]*Bt_B[1][0]));
    Bt_B[1][0]=Bt_B[0][0];
    Bt_B[0][0]=(double)Bt_B[1][1]/BTB;
    Bt_B[0][1]=(double)(-1)*Bt_B[0][1]/BTB;
    Bt_B[1][1]=(double)Bt_B[1][0]/BTB;
    Bt_B[1][0]=Bt_B[0][1];
    /* 求参数列 a */
    for(i=0;i<key-1;i++)
    {
        x1[i]=Bt_B[0][0]*Z[i]+Bt_B[0][1];
        get2[i]=Bt_B[1][0]*Z[i]+Bt_B[1][1];
    }
```

```
for(i=0;i<key-1;i++)
{
    a=a+x1[i]*x0[i+1];
    u=u+get2[i]*x0[i+1];
}
/* 确定模型 */
printf("\n 模型表达式:x(1)(t+1)=%f*e^%f*t\n",x0[0]-(double)u/a,
        (-1)*a,(double)u/a);
/* 进行预测 */
printf("\n 请输入预测时间参数:");
scanf("%d",&key);
printf("\n 生成模型计算数据:%f",
        (x0[0]-(double)u/a)*exp((double)(-1)*a*key)+(double)u/a);
printf("\n\n 预测值:%f", (x0[0]-(double)u/a)*
        (exp((double)(-1)*a*key)-exp((double)(-1)*a*(key-1))));
}
```

四、程序应用

运行编好的灰色预测软件,对文献[1]中的业务量值(P27)进行灰色预测,得到了灰色预测模型: $x(1)(t+1)=204.8759e^{0.1345856*t}-179.7759$ 。利用灰色预测模型所得数据与文献[1]中通过时间序列预测所得数据相比的结果如表:

年度	实际值	灰色预测模型所得值	误差值	时间序列预测所得值	误差值
1983	25.1	25.1	0.0	22.93	-2.17
1984	28.6	29.5	0.9	28.75	0.15
1985	32	31.78	-0.22	34.57	2.57
1986	43.2	40.64	-2.56	40.39	-2.81
1987	44	44.19	0.19	46.21	2.21
1988	49.7	50.49	0.79	52.03	2.33
1989	56.4	57.93	1.53	57.85	1.45
1990	67.4	66.17	-1.23	63.67	-3.73

从上表的误差值的大小可以看出灰色预测模型的精确度比时间序列预测的精确度要高。

该灰色预测软件对于采用灰色预测方法的大量数据计算起到了简便的作用,同时通过对该软件的了解,可以更加掌握灰色理论中的GM(1,1)模型。

参考文献:

- [1]李文海编,《电信网》,人民邮电出版社,1993年1月
- [2]邓聚龙著,《灰色预测与决策》,华中理工大学出版社,1986年6月
- [3]王军政编,《Turbo C 2.0 实用高级编程技巧》,北京科海培训中心,1992年5月 (本文收稿日期:1995.3.9)

程控交换机与微机的串行通信

国防科工委指挥技术学院[101407] 朱诗兵 李迎春

摘 要:本文从串行通信方式、串行链路连接和软件编程等方面研究了程控交换机与微机之间的串性通信。

关键词:程控交换机, 串行通信, 流量控制

一、引言

由于计算机技术与通信技术的结合,加强了通信的智能化控制,便于实行集中维护,促进了程控交换机的各项研究。程控交换机的维护管理单元都是由接口处理机及相应的接口系统组成,从目前的程控交换机来看,常常具有串行 RS-232 和并行 GP-IB 两种接口,供用户进行程控交换机维护和开发时使用。本文根据我们在研制 HICOM370 程控交换机计费系统中的体会,对程控交换机与微机的串行通信进行了探讨,在计费系统中仅仅把微机看作程控交换机的终端设备,从串口不断的接收来自程控交换机的呼叫记录信息。

二、串行通信方式

在计费系统中选用的微机为 IBM PC(包括 PC/XT, PC/AT 及兼容机)微型计算机。它一般的带有串行通信适配器,接口标准符合 EIA RS-232C, 使用 DB-9P(或 DB-25P)插头,并可按需要设定为 DTE 或 DCE 方式。而程控交换机也设置有 RS-232 接口,配备 DB-25P 插座,同样按要求亦可设定为 DTE 和 DCE 方式。当使用电缆将两者连接通信时,必须要先确定通讯双方的串行通信方式。

按照微机系统的软硬件配备情况,可将微机的串行通信分为以下三种形式:

- 1、利用 DOS 操作系统 INT 21H(AH 赋值 3、4)调用进行串行通信;
- 2、利用 BIOS 基本输入输出系统 INT 14H 调用进行串行通信;
- 3、按串行通信适配器的结构(8250、8251A)对寄存器直接编程进行串行通信。

在一般的点到点半双工计算机通信中,上述三种都是比较常用的串行通信方式,它们各自具有自己的通信特点。在计费系统中为了充分发挥 RS-

232C 的功能、能满足不同通信特殊情况下的要求,串行通信方式采用按照串行适配器结构直接编程方式,它直接对计算机硬件设备进行读写控制,在控制硬件握手信号方面显得更突出,在编程处理上可采用查询和多种中断的灵活方法。

三、串行链路连接

串行链路是微机与程控交换机的物理线路,它是计费系统串行通信的物质基础。微机与程控交换机之间的硬件连接是按照 EIA RS-232C 标准进行的,连接的主要原则是分别根据微机和程控交换机配置的串行接口所使用的 DB-25P 或 DB-9P 设置情况,分不同类型进行处理。串行链路接口连接时,应主要考虑程控交换机对各信号线的约定,一般来说,它们同 EIA RS-232 标准中的规定的各信号线存在差异,要特别注意。在 HICOM370 程控交换机计费系统中,由于通信双方的程控交换机使用 25 脚插座,而微机使用 9 脚插头,于是二者之间的连接如下图:

25 脚	8	接收线路信号检测	1	25 脚
	3	接收数据	2	
	2	发送数据	3	
	20	数据终端就绪	4	
	7	信号地	5	
	6	数传设备就绪	6	
	4	请求发送	7	
	5	清除发送	8	
	22	振铃指示	9	

四、软件编程

实现微机与程控交换机的串行通信要做两方

面的工作:首先是完成通信双方的线路连接;其次是根据通信方式编写通信程序。微机与程控交换机的通信都是由通信软件来实现的,对于串行通信的编程,结合微机与程控交换机的不同特点,应处理好下面三个问题:

1、串行通信的通信协议

串行通信的链路通信规程(协议)是保证传输信息正确可靠而遵循的规则。对于串行链路有 TTY、BSC、SDLC/HDLC 三种标准链路通信规程,在微机与程控交换机的通信中主要是用 BSC 规程,它设计的接口功能(DSR、DTR、CTS、RTS)是进行链路控制必不可少的,也是软件编程中必须遵循的规则。

2、查询与中断在编程中的选择

利用按串行适配器结构直接编程可分别采用查询式通信和中断式通信。查询式通信是监视通信链路状态内容来传输信息,而中断式通信是通过链路状态内容变化产生中断来传输信息。两种方法进行比较,中断方式可设置前后台工作方式,在通信的间隙中可进行其它内容的操作,提高利用效益。比较其优劣,计费系统的接收采用中断方式的方法。

3、链路中的流量控制

在微机与程控交换机的串行通信时,为防止有效信息的丢失,必须在链路中使用串行流量控制。为解决串行通信过程中在接口丢失信息的问题(即流量控制),实现通信双方间的流量控制,我们是利用程序借助于接口硬件实现的。这种方法是依靠通信双方必须遵守的 EIA RS-232C 标准的接口状态信息和控制功能来实现传输信息的流量控制。

4、部分程序清单

```
#include <stdio.h>
#include "ibmcom.h"
#define PORT 1 /* 微机接收端口 */
#define SPEED 4800 /* 通信速率 */
main() {
    int herb;
    int BZ_start=0;
    if (herb = com_install(PORT)) { /* 中断方式的驻留 */
        printf("com_install() error: %d\n", herb);
        return;
    }
    com_set_speed(SPEED);
    com_set_parity(COM_EVEN, 2); /* 2 bit 停止位 */
    com_raise_dtr(); /* DTR 为高 */
    clrscr();
```

```
while (1) {
    if (kbhit()) {
        herb = getch()+256;
        if (herb == 301 /* Alt-X */) {
            com_deinstall(); /* 驻留程序的释放 */
            com_lower_dtr(); /* DTR 为低 */
            exit();
        }
        com_tx(herb);
    }
    if ((herb=com_rx()) != '\0') {
        if (BZ_start == 1) putchar(herb);
        else if (herb == '*') { /* 判断特殊符 "*" */
            BZ_start=1;
            putchar(herb);
        }
    }
}
```

五、结束语

本文讨论了程控交换机与微机串行通信中的一些问题,按照上文所论述的内容,我们在 HICOM370 程控交换机与微机的串行通信中得到了很好的接收效果,使得研制的计费系统准确可靠。

参考文献:

- [1] 5700A Calibrator Operator Manual, FLUKE, 1989
- [2] 王士元等,《IBM PC/XT 接口技术及其应用》,南开大学出版社,1990 年
- [3] 王军政,《Turbo C 2.0 实用高级编程技巧》,北京科海培训中心,1992 年

(本文收稿日期:1994.12.29)

一种高精度温度测控系统的研制方法

(上接第4页)

化的影响;浮地屏蔽;软件动态校零、数字滤波、直接校正的标度变换等。

参考文献

- [1] 吴永生等,《热工测量及仪表》,水利电力出版社,1991.10
- [2] 方崇智,《过程控制计算机》,清华大学出版社,1990

(本文收稿日期:1995.1.17)

用查表法实现快速循环冗余位校验计算

中国海关干部管理学院计算机教研室 董 云

循环冗余位校验(Cyclical Redundancy Check 英文简称 CRC)的实现分为硬件和软件两种方法,其中软件实现的关键在于计算速度。如果单纯模拟硬件实现方法,则计算速度较慢。笔者在编制一个数据通讯软件中,运用了一种查表法计算 CRC,速度很快,效果极佳。

首先介绍其原理,如果每次参与 CRC 计算的信息为一个字节,该信息字节加到 16 位的累加器中去时,只有累加器的高 8 位或低 8 位与信息字节相互作用(异或),相互作用(异或)的结果记为组合值,那么累加器中的新值等于组合值加上(按模 2 异或)累加器中未改变的那一半即为新的 CRC 值。

组合值只有 256 种可能,因此可利用硬件模拟算法先算好它们的 CRC 值预先填入一张表中,该表的每一单元对应相对值的 CRC。这样就可以通过查表法来计算 CRC 值。下面给出用 C 语言编制的计算程序。

首先将 CRC 生成多项式及 CRC 值表定义为一个头文件 CRC.H:

```
#define CRC_CCITT 0x1021 //CCITT 多项式
#define REV_CCITT 0x8408
```

//反转 CCITT 多项式

```
#define CRC16 0x8005 //CRC16 多项式
```

```
#define REV_CRC16 0xa001
```

//反转 CRC16 多项式

```
unsigned short crc_tble[256]; //CRC 值表
```

注:16 位 CCITT 多项式($X^{16}+X^{12}+X^5+1$)和 16 位 CRC16 多项式($X^{16}+X^{15}+X^2+1$)为两种最常用的 CRC 多项式。反转多项式是指在数据通讯时,信息字节先传送或接收低位字节,如重新排位影响 CRC 计算速度,故设反转多项式。

造表和查表法 CRC 计算函数。

```
#include "crc.h"
```

```
void mk_crc_tble(unsigned short genpoly)
```

```
{ unsigned short crc_tble[256];
```

```
unsigned short accnum=0;
```

```
unsigned short i,j,k;
```

```
for(i=0,k=0;i<256;i++,k++){
```

```
    i<<=8;
```

```
    for(j=8;j>0;j--){
```

```
        if((i^accnum)&0x8000)
            accnum=(accnum<<=1)^genpoly;
        else accnum<<=1;
        i<<=1;
    }
    crc_tble[k]=accnum;
}
```

```
void crc_update(unsigned short data, unsigned short accnum)
```

```
{ accnum=(accnum<<=8)^crc_tble[(accnum>>8)^data]; }
```

注:genpoly 为 CRC 多项式,accnum 为累加器值(即为新的 CRC 值),data 为参与 CRC 计算的信息。以上两函数在 Borlandc++3.1 下编写。

参考文献:

- [1] 顾慰文,《纠错码及其在计算机系统中的应用》,人民邮电出版社,1980。
- [2] J. P. Roth, W. G. Bouricius: Programmed algorithms to compute teststo detect and and distinguish between failures in logic circuits, IEEE Trans. Electron. Comput., EC-16, No. 5, pp. 567-580(1977)
- [3] Joe Campell: C Programmer's Guide to Serial Communication. (1988)

(本文收稿日期:1995.4.10)

本刊致订户:

2~5 期退款通知

由于本刊易名《现代计算机》的办证注册在 6 月份,改大 16 开版出刊从 6 月份开始,补出 6 月以前的刊期已无可能,经与发行部门商定,2~5 期作停刊处理。特向订户深表歉意。凡在当地邮局订了 2、3、4、5 期的订户,请向当地邮局办理退款。

同时,95 年第 6 期(总第 41 期)第 19 页:“本刊公告”提出的出增刊计划相应取消。

敬请订户互相转告,请一定在年底前到当地邮局办理上述退款。

如何实现显示字库的“零内存占用”

郑州市解放军电子技术学院 301 室 杨戈 王成义 苏庆会

内容提要:本文首先介绍了扩展内存使用的一般性常识,然后通过一个实用程序,详细、深入地展示出了通过扩展内存使用显示字库的方法,从而,实现了“零内存占用”。

关键字:常规内存、扩展内存、INT 2FH、XMS 驱动程序。

现在,各种汉字系统软件在我国的计算机软件市场上十分畅销,如:天倚、天汇以及 NCDOS 和 UCDOS 等等,都是比较流行的优秀汉字系统软件。它们都有一共同的特点,即:现实了“零内存占用”,从而给用户一个完整的 640K 的常规内存,以便能够为执行其它任务腾出更多的自由空间。“零内存占用”究竟是怎样实现的?我们能不能在自己的程序中,实现“零内存占用”?回答是完全可以的。这就涉及到了扩展内存的使用技术问题。

扩展内存是指 DOS 所不能管理的其地址空间在 1M 以上的内存空间。扩展内存的管理是通过中断扩充 INT 2FH 来进行通讯的。但是,在链接之后,它将允许在它的内部码中使用向量来处理它本身。

首先,我们得检查是否已安装了 XMS 扩展内存管理驱动程序,即:

```
MOV AX,4300H
INT 2FH
CMP AL,80H
JE 驱动程序已安装
```

```
JMP 未安装 himem.sys 驱动程序
```

然后,我们就可获得此向量的地址 XMS_ADDRESS:

```
MOV AX,4310H
INT 2FH
XMS_ADDRESS=ES*0X10000+BX
```

这样我们把功能号送往寄存器 AH 中,然后通过语句 CALL XMS_ADDRESS 就可调用驱动程序中的控制代码了。

其次,分配扩展内存块的单位是 K 字节,分配成功后返回已分配的扩展内存块的句柄。一个柄就是一个整数,如同我们在 DOS 环境下打开文件时,使用的文件句柄一样。在一个扩展内存的句柄

使用完毕后,请一定要记住释放它。否则,即使程序已退出后,其它程序也不能使用此块扩展内存,除非重新启动机器。

另外,在扩展内存和常规内存之间传送数据时,必须使用一个结构:

```
struct xms_struct
{
    unsigned long count; //要传送的数据量
    int sourcehandle; //源数据块的句柄
    long int sourceoff; //源数据的偏移地址
    int desthandle; //目的块的句柄
    long int destinoff; //目的数据的偏移地址
}; //使用常规内存时,其句柄为 0
```

最后,请别忘记在 CONFIG.SYS 加入 XMS 管理驱动程序 DEVICE=C:\DOS\HIMEM.SYS,否则,以下程序是不能使用扩展内存的!

笔者在程序中所使用的是 UCDOS 2.0 中的 16*16 点阵的显示字库 cclib.dat。

下面就将笔者用 Borlandc C++ 3.1 编写的将显示字库驻留到扩展内存的源程序清单,奉献出来,以飨计算机专业同仁和计算机软件爱好者。

```
#include <stdlib.h>
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#include <bios.h>
#include <ctype.h>
#include <dos.h>
```

```
#define word_between 2
#define colum 16
```

```
struct xms_struct
```



```

{
    unsigned long count;
    int sourcehandle;
    long int sourceoff;
    int desthandle;
    long int destinoff;
}xms;
unsigned long xms_address;
unsigned xms_CCLIB_handle;

char xms_test(void);
int xms_alloc(int needsize);
char xms_move(void);
void putto_getfrom_xms(unsigned far * store, int put-
sign,
    unsigned long size, unsigned long offset, unsigned int han-
dle);
int load_cclib(char * filename);
void xms_free(int handle);
void checkerror();
int put_hz16(char qu, char wei, int x, int y, int hz_color);
void put_hz(char * hz_buf, int x, int y, int hz_color);

int main(void)
{
    int gdriver = VGA, gmode = VGAHI; //set VGA's
640 * 480 mode
    char hz_buf[] = "欢迎您使用本程序!";
    initgraph(&gdriver, &gmode, "");
    checkerror();
    if(! xms_test())
    { printf("XMS has not been installed ! \n"); exit(-
1); }
    if(! load_cclib("cclib.dat"))
    { printf("Error in file --- cclib16.dat ! \n"); exit(-
1); }
    put_hz(hz_buf, 200, 200, RED);
    getchar();
    getchar();
    xms_free(xms_CCLIB_handle);
    /* clean up */
    closegraph();
    return 1;
}

void checkerror()
{
    int errorcode;
    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != gtOk) /* an error occurred */

```

```

{
    printf("Graphics error: %s\n", grapherrormsg(error-
code));
    printf("Press any key to halt:");
    getch();
    exit(1); /* return with error code */
}
}

```

//首先,检查 XMS 驱动程序是否已存在
//然后,取得扩展内存管理驱动程序的代码的地址

```

char xms_test(void)
{
    union REGS reg;
    struct SREGS segreg;
    reg.x.ax = 0x4300;
    int86(0x2f, &reg, &segreg);
    if(reg.h.al != 0x80) return 0;
    else
    {
        reg.x.ax = 0x4310;
        int86(0x2f, &reg, &reg, &segreg);
        xms_address = segreg.es;
        xms_address * = 0x10000;
        xms_address += reg.x.bx;
        return 1;
    }
}

```

//从扩展内存中申请大小为 needsize (K bytes)的内存
//返回值,即为此块已申请的扩展内存的句柄

```

int xms_alloc(int needsize)
{
    int i;
    char errorstatus;
    asm MOV DX, needsize
    asm MOV AH, 6
    asm CALL xms_address
    asm MOV i, DX
    asm MOV errorstatus, BL
    switch(errorstatus)
    {
        case 0x80 : printf("Alloc function : Doesn't pass. \n");
return 0;
        case 0x81 : printf("Alloc function : Have vdisk device. \
n"); return 0;
        case 0xa0 : printf("Alloc function : All XMS have been
allocated. \n"); return 0;
        case 0xa1 : printf("Alloc function : All handle have been
used. \n"); return 0;
        default : break;
    }
}

```

```

    }
    return i;
}

//移动扩展内存块子程序
char xms_move(void)
{
    char far * p;
    char errorstatus;
    p=(char far *)&xms;
    asm MOV AH,11 //0BH
    asm PUSH DS
    asm POP ES
    asm PUSH DS
    asm LDS SI,p
    asm CALL xms_address
    asm POP DS
    asm MOV errorstatus,BL
    switch(errorstatus)
    {
        case 0x80: printf("Move function : Doesn't pass. \n");return (0);
        case 0x81: printf("Move function : Have vdisk device. \n");return (0);
        case 0x82: printf("Move function : A20 error. \n");return (0);
        case 0xa3: printf("Move function : Invalid source handle. \n");return (0);
        case 0xa4: printf("Move function : Invalid source offset. \n");return (0);
        case 0xa5: printf("Move function : Invalid destination handle. \n");return (0);
        case 0xa6: printf("Move function : Invalid destination offset. \n");return (0);
        case 0xa7: printf("Move function : Invalid size. \n");return (0);
        case 0xa8: printf("Move function : Move cross. \n");return (0);
        case 0xa9: printf("Move function : Parity error. \n");return (0);
    }
    return (1);
}

/* -----
   To free the memeory allocated.
   ----- */
//释放已分配的扩展内存的句柄
void xms_free(int handle)
{
    asm MOV BX,handle

```

```

    asm MOV CX,handle
    asm MOV DX,handle
    asm MOV AH,10 //0AH
    asm CALL xms_address
}

//往扩展内存中送数据或从扩展内存中取数据,其中,put_
sign=1 时,为送入数据;
//put_sign=0 时,为取数据.* store 为常规内存;size 为要
送入或要取出数据的大小;
//offset 为数据在扩展内存中偏移量;handle 为已申请到的
扩展内存的句柄。
void putto_getfrom_xms(unsigned far * store,int put_
sign,unsigned \
    long size,unsigned long offset,unsigned int handle)
{
    int i,k;
    unsigned bufsize=0x4000; //8k
    unsigned pagenum;
    unsigned lastpage;
    unsigned int myseg, myoff;
    unsigned long off_32b;
    pagenum = size / bufsize;
    lastpage = size % bufsize;
    pagenum ++;
    k=pagenum * 16;
    myseg=FP_SEG(store);
    myoff=FP_OFF(store);
    off_32b=myseg * 0x10000+myoff;
    if(put_sign)
    {
        xms.destinoff=offset;
        for(i=1;i<=pagenum;i++)
        {
            k=bufsize;
            if( i==pagenum )k=lastpage;
            xms.count=k;
            xms.sourcehandle=0; //formal memory
            xms.sourceoff=off_32b;
            xms.destinhandle=handle;
            if(xms_move()==0)exit(-1);
            xms.destinoff+=k;
        }
    } // if(put_sign)
    else
    {
        xms.sourceoff=offset;
        for(i=1;i<=pagenum;i++)
        {
            k=bufsize;
            if( i==pagenum )k=lastpage;

```



```

xms.count=k;
xms.sourcehandle=handle;
xms.destinhandle=0;
xms.destinoff=off_32b;
if(xms_move()==0)exit(-1);
xms.sourceoff+=k;
}
} // if(put_sign)...esle...

//往扩展内存中装入显示字库 * filename
int load_cclib(char * filename)
{
    FILE * fp;
    int i,k;
    unsigned char buf[0x4000];
    unsigned bufsize=0x4000; //16k
    unsigned pagenum;
    unsigned lastpage;
    unsigned int myseg, myoff;
    unsigned long off_32b;
    if( (fp=fopen(filename,"r+b")) == NULL )
    {
        printf("Can't open the file---%s !!!\n",filename);
        return 0;
    }
    fseek(fp,0L,SEEK_END);
    off_32b=ftell(fp);
    pagenum = off_32b / bufsize;
    lastpage = off_32b % bufsize;
    pagenum ++;
    fseek(fp,0L,SEEK_SET);
    k=pagenum * 16;
    xms_CCLIB_handle = xms_alloc(k); //KB Allocate to
handle
    myseg=FP_SEG(buf);
    myoff=FP_OFF(buf);
    off_32b=myseg * 0x10000+myoff;
    xms.destinoff=0;
    for(i=1;i<=pagenum;i++)
    {
        k=bufsize;
        if( i==pagenum )k=lastpage;
        fread(buf,k,1,fp);
        xms.count=k;
        xms.sourcehandle=0; //formal memory
        xms.sourceoff=off_32b;
        xms.destinhandle=xms_CCLIB_handle;
        if(xms_move()==0) return 0;
        xms.destinoff+=k;
    }
}

```

```

fclose(fp);
return 1;
}

//显示 16 * 16 点阵字库,其中,qu 和 wei 分别为区位码,x
和 y 分别为要显示的汉字
//在屏幕上的纵横座标,hz_color 为要显示汉字的颜色
int put_hz16(char qu,char wei,int x,int y,int hz_color){
    unsigned char buf[32];
    unsigned long offset;
    unsigned char c;
    int i,j,k;
    unsigned char mask[9]={0x80,0x40,0x20,0x10,0x08,
0x04,0x02,0x01};
    if(x<0||x>getmaxx()||y<0||y>getmaxy())
        {printf("X or Y's value out of range !\n");return 0;}
    offset=(unsigned long)((qu-1)*94+(wei-1))*
32;
    putto_getfrom_xms(buf,0,32,offset,xms_CCLIB_
handle);
    for(i=0;i<16;i++)
        for(j=0;j<2;j++)
            { c=buf[i*2+j];
                for(k=0;k<8;k++)
                    { if(c&mask[k])putpixel(x+j*8+k,y+i,hz_
color);}
            }
    return 1;
}

//从座标(x,y)处,用颜色 hz_color 显示内存 * hz_buf 中
的汉字
void put_hz(char * hz_buf,int x,int y,int hz_color)
{
    char qu,wei;
    char s[1];
    while(*hz_buf)
    {
        qu=*hz_buf++-0xa0;wei=*hz_buf++-
0xa0;
        put_hz16(qu,wei,x,y,hz_color);
        x+=(column+word_between);
    }
}

```

参考资料:

- [1]戈喻秋、梁健江编写,《微机图像实用技术与范例》,北京希望电脑公司出版。
- [2]《Borland C++ 3.1 程序员参考资料》。

(本文收稿日期:1995.2.11)

CMOS 及硬盘引导扇区读写示例

江西拖拉机发动机厂 黄焕如

内容提要:本文简要地介绍了 CMOS、硬盘主引导扇区和 DOS 引导扇区的概念及其主要内容,分别用 DEBUG、宏汇编和 C 语言列举了读写这些重要信息的示例程序,可供读者参考。

关键字:CMOS、硬盘主引导扇区、DOS 引导扇区、DEBUG、宏汇编、C

CMOS 和硬盘引导扇区(硬盘主引导扇区和 DOS 引导扇区)都是计算机正常运行必不可少的部分,而这些重要的信息又常常是病毒或其他因素破坏的目标,因此保存和恢复这些重要信息是一项很重要的工作。

目前有许多工具软件都能直接对 CMOS、硬盘主引导扇区和 DOS 引导扇区进行读或写,如 PC-TOOLS ver7.0 或 DOS5.0 的 MIRROR [/PARTN] 命令,将产生文件 PARTNSAV.FIL,然后可用 REBUILD 或 UNFORMAT [/PARTN] 命令恢复;CPAV ver1.30 的 BOOTSAFE [/M] 命令,将产生文件 CMOS.CPS,然后可用 BOOTSAFE [/R] 命令恢复;NORTON ver5.0/6.0 的 DISK-TOOL,选 CREAT RESCUE DISKETTE 项可将 CMOS、硬盘主引导扇区和 DOS 引导扇区全部保存在文件 CMOSINFO.DAT 上,然后可选 RESTORE RESCUE DISKETTE 项恢复;DOS ver5.0 的 FDISK/MBR(master boot record)也可以恢复硬盘主引导扇区信息。但是如果能直接编制程序来实现对上述重要的信息读写,不仅仅很方便可靠,而且能加深对计算机这类重要信息的理解,修改和扩充这些程序,能编制出保存和恢复计算机重要信息的更好的软件。以下将分 CMOS 信息、硬盘主引导程序和 DOS 引导程序三部分,分别使用 DEBUG、宏汇编和 C 语言,示例这些重要信息的读写。

一、CMOS 信息实际保存在其内部芯片 MC146818 上,其有两个端口:地址口 70H,数据口 71H,信息总共 64 字节。各信息地址定义如下:

0—05 秒、分、时及报警,06—09 星期、日、月、年,10—13 状态寄存器 A—D,14 诊断状态(无故障 0),15 电源状态,16 软驱类型(高 4 位为 A 盘,低 4 位为 B 盘,1 为 360K,2 为 1.2M,6 为 1.4M,7 为 720K),17 保留,18 C 或 D 硬盘类型(高 4 位 C

盘,低 4 位 D 盘,其值结合 19 号地址决定硬盘类型,19 硬盘类型辅助字节,20 设备状态,21—22 基本内存,23—24 扩展内存,25—26 C 和 D 类型号,27—45 保留,46—47 CMOS 信息校验,48—49 同 23—24,50 世纪值,51 信息标志,52 口令(无密码 0),53—63 口令字(注意:不同机型有不同的保留方式)。

CMOS 信息的读写通常是先向地址口发送地址号,以指明下一步在数据口读写的信息是哪个地址号上的数据,然后再进行读或写。宏汇编主要使用 out/in 指令,C 语言主要使用 outportb/inportb 函数。

1. 利用 DEBUG 读写 CMOS

利用 DEBUG 读写 CMOS,可直接使用 O/I 命令。如对于 AMI386 微型机的口令消除可执行:

```
C>DEBUG
-O70 10 (向 CMOS 地址端口送地址 10H)
-O71 00 (向 CMOS 数据端口送数据 00H,
设置口令无效)
-Q
```

2. 利用宏汇编读写 CMOS

/* 保存 CMOS 信息到文件中 CMOS.DAT */

```
seg_a segment byte public
assume cs:seg_a, ds:seg_a
org 100h
CMOS proc far
start: mov si,200h
mov ah,0
mov bx,40h
loc_1: mov al,ah
out 70h,al ; 端口 70 CMOS 地址
mov cx,5
loc_2: loop loc_2
in al,71h ; 端口 71 CMOS 数据
mov [si],al
cmp ah,0Eh
```

```

        jb      loc_3
        add     ah,80h
loc_3:   inc     ah
        inc     si
        dec     bx
        mov     cx,bx
        jnz     loc_1
        mov     ah,3Ch ;建立文件
        mov     dx,offset data_1
        mov     cx,20h
        int     21h
        mov     bx,ax
        mov     dx,200h
        mov     cx,40h
        mov     ah,40h ;写文件
        int     21h
        mov     ah,4Ch
        int     21h
data_1   db      'CMOS. DAT', 0
CMOS     endp
seg_a    ends
        end     start

/* 从 CMOS. DAT 恢复 CMOS 信息 */
seg_a    segment      byte public
        assume      cs:seg_a, ds:seg_a
        org         100h
CMOS     proc         far
start:   mov         ah,3Dh ;打开文件
        mov         dx,offset data_1
        mov         al,0
        int         21h
        mov         bx,ax
        mov         si,200h
        mov         dx,si
        mov         cx,40h
        mov         ah,3Fh ;读文件
        int         21h
        mov         ah,0
        mov         bx,41h
loc_1:   mov         al,ah
        out         70h,al ;端口 70 CMOS 地址
        mov         cx,5
loc_2:   loop        loc_2
        mov         al,[si]
        out         71h,al ;端口 71 CMOS 数据
        cmp         ah,0Eh
        jb         loc_3
        add         ah,80h
loc_3:   inc         ah
        inc         si
        dec         bx
        mov         cx,bx
        loop        loc_1
        mov         ah,4Ch

```

```

        int     21h
data_1   db      'CMOS. DAT', 0
CMOS     endp
seg_a    ends
        end     start

```

3. 利用 C 语言读写 CMOS

/* 保存 CMOS 信息到文件中 CMOS. DAT */

#include <io. h>

#include <stdio. h>

main()

```

{
    int i; FILE * fp;
    char buffer[64];
    fp=fopen("cmos. dat","wb");
    for(i=0; i<64; i++)
        {outportb(0x70,i); * (buffer+i)=inportb(0x71);}
    if(fwrite(buffer,1,64,fp)!=64) exit(1);
    fclose(fp);
}

```

/* 从 CMOS. DAT 恢复 CMOS 信息 */

#include <io. h>

#include <stdio. h>

main()

```

{
    int i; FILE * fp;
    char buffer[64];
    if(! (fp=fopen("cmos. dat","rb")))
        exit(1);
    fread(buffer,1,64,fp);
    fclose(fp);
    for(i=0; i<64; i++)
        { outportb(0x70,i); outportb(0x71, * (buffer+i)); }
}

```

二、硬盘主引导扇区中含有引导程序和分区表,总共占 100H 字节,其中 01BEH 后 40H 字节为分区信息表,自举有效标志 AA55H 占最后二个字节。分区信息表中每个分区信息从 XXXEH 开始到 XXXDH 字节止:

00H:可引导指示符(80 活动分区,00 不可引导),01—03H:分区起始地址,04H:操作系统代码(00=未知,01=DOS 系统 12 位 FAT,02=XENIX 系统,04=DOS 系统 16 位 FAT,05=逻辑盘),05—07H:分区终止地址,08—0BH:本分区前扇区数,0C—0FH:本分区扇区。

由于硬盘主引导扇区不属于 DOS 范围,宏汇编主要使用 int 13h 中断,C 语言主要使用 biosdisk 函数(实际上仍然是执行 int 13 中断)。

1. 利用 DEBUG 读写硬盘主引导扇区

保存硬盘主引导扇区到文件中 MBOOT. DAT
C>DEBUG

-A 100

```

MOV DX,80    (硬盘0号磁头)
MOV CX,1     (0道1扇区)
MOV AX,201   (读1扇区)
MOV BX,200   (缓冲区)
INT 13
INT 3

```

-G

-M200 FF0 100 (将数据移到起始位置)

-NMBOOT.DAT

-REX 0

-RCX 200

-W

从文件 MBOOT.DAT 恢复硬盘主引导扇区

C>DEBUG MBOOT.DAT

-M100 FF0 200 (将数据移到 200H)

-A100

```

MOV DX,80    (硬盘0号磁头)
MOV CX,1     (0道1扇区)
MOV AX,301   (写1扇区)
MOV BX,200   (缓冲区)
INT 13
INT 3

```

-G

-Q

2. 利用宏汇编读写硬盘主引导扇区

;保存硬盘主引导扇区到文件中 MBOOT.DAT

```

seg-a    segment    byte public
          assume     cs:seg-a, ds:seg-a
          org        100h
BOTT     proc        far
start:   jmp         short loc_1
data_1   db          'MBOOT.DAT', 0
loc_1:   mov         ax,201h    ;读一扇区
          mov         bx,200H
          mov         cx,1      ;0道1扇区
          mov         dx,80h    ;硬盘0号磁头
          int         13h
          mov         ah,3Ch    ;建文件
          mov         dx,offset data_1
          mov         cx,0      ;文件属性
          int         21h
          mov         cx,200h
          mov         bx,ax     ;写文件
          mov         dx,200H   ;缓冲区地址
          mov         ah,40h
          int         21h
          mov         ah,3Eh    ;关文件
          int         21h
          int         20h
BOTT     endp
ends
start

```

;从文件 MBOOT.DAT 恢复硬盘主引导扇区

```

seg-a    segment    byte public
          assume     cs:seg-a, ds:seg-a
          org        100h
BOOT     proc        far
start:   jmp         short loc_1
data_1   db          'mboot.dat', 0
loc_1:   mov         ax,3D00h    ;打开文件
          mov         dx,offset data_1
          int         21h
          mov         bx,ax
          mov         ah,3Fh    ;读文件
          mov         cx,200h   ;字节数
          mov         dx,200H   ;缓冲区地址
          int         21h
          mov         ax,301h   ;写一扇区
          mov         bx,200H
          mov         cx,1      ;0道第1扇区
          mov         dx,80h    ;硬盘0号磁头
          int         13h
          int         20h
BOOT     endp
nds
start
end

```

3. 利用 C 语言读写硬盘主引导扇区

/* 保存硬盘主引导扇区到文件中 MBOOT.DAT */

#include <bios.h>

#include <stdio.h>

main()

```

{
    FILE *fp; char buffer[512];
    fp=fopen("mboot.dat","wb");
    biosdisk(2,0x80,0,0,1,1,buffer);
    if(fwrite(buffer,1,512,fp)!=512) exit(1);
    fclose(fp);
}

```

/* 从文件 MBOOT.DAT 恢复硬盘主引导扇区 */

#include <bios.h>

#include <stdio.h>

main()

```

{
    FILE *fp; char buffer[512];
    if(! (fp=fopen("mboot.dat","rb")))
        exit(1);
    if(fread(buffer,1,512,fp)!=512) exit(1);
    fclose(fp);
    biosdisk(3,0x80,0,0,1,1,buffer);
}

```

三、在硬盘的 DOS 分区上有一引导记录,用来加载 DOS 系统的,它存放在硬盘的逻辑 0 扇区。DOS 引导区分三部分,第一部分为 BIOS 参数块 BPB,地址为 00H-2CH;第二部分为引导记录程

序主体,地址为 2EH—7DH;第三部分为错误提示信息,地址为 80H—FFH。

DOS 引导扇区的读写,宏汇编主要使用 int 25h/26h 中断,C 语言主要使用 int86 函数(实际上仍然是执行 int 25h/26h 中断)。

1. 利用 DEBUG 读写 DOS 引导扇区

保存 DOS 引导扇区到文件中 RBOOT.DAT

C>DEBUG

-L 100 2 0 1 (读硬盘逻辑 0 扇区)

-NRBOOT.DAT

-RCX 200

-W

从文件 RBOOT.DAT 恢复 DOS 引导扇区

C>DEBUG RBOOT.DAT

-W 100 2 0 1 (写硬盘逻辑 0 扇区)

-Q

2. 利用宏汇编读写 DOS 引导扇区;保存 DOS 引导扇区到文件中 RBOOT.DAT

```
seg-a segment byte public
    assume cs:seg-a, ds:seg-a
    org 100h
BOTT proc far
start: jmp short loc_1
data_1 db 'RBOOT.DAT', 0
loc_1: mov al,02h ;硬盘 C
    mov bx,200H
    mov cx,1 ;1 扇区
    mov dx,0h ;逻辑 0 扇区
    int 25h
    mov ah,3Ch ;建文件
    mov dx,offset data_1
    mov cx,0 ;文件属性
    int 21h
    mov cx,200h
    mov bx,ax ;写文件
    mov dx,200H ;缓冲区地址
    mov ah,40h
    int 21h
    mov ah,3Fh ;关文件
    int 21h
    int 20h
BOTT endp
seg-a ends
end start
```

;从文件 RBOOT.DAT 恢复 DOS 引导扇区

```
seg-a segment byte public
    assume cs:seg-a, ds:seg-a
    org 100h
BOOT proc far
start: jmp short loc_1
data_1 db 'rboot.dat', 0
```

```
loc_1: mov ax,3D00h ;打开文件
    mov dx,offset data_1
    int 21h
    mov bx,ax
    mov ah,3Fh ;读文件
    mov cx,200h ;字节数
    mov dx,200H ;缓冲区地址
    int 21h
    mov al,02h ;写硬盘 C
    mov bx,200H
    mov cx,1 ;1 扇区
    mov dx,0 ;逻辑 0 扇区
    int 26h
    int 20h
BOOT endp
seg-a ends
end start
```

3. 利用 C 语言读写 DOS 引导扇区

/* 保存 DOS 引导扇区到文件中 RBOOT.DAT */

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

```
main()
```

```
{
    union REGS in,out;
    FILE *fp; char buffer[512];
    fp=fopen("rboot.dat","wb");
    in.h.al=2;
    in.h.bx=buffer;
    in.h.cx=1;
    in.h.dx=0;
    int86(0x25,&in,&out);
    if(fwrite(buffer,1,512,fp)!=512) exit(1);
    fclose(fp);
}
```

/* 从文件 RBOOT.DAT 恢复 DOS 引导扇区 */

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

```
main()
```

```
{
    union REGS in,out;
    FILE *fp; char buffer[512];
    if(! (fp=fopen("rboot.dat","rb")))
        exit(1);
    if(fread(buffer,1,512,fp)!=512) exit(1);
    fclose(fp);
    in.h.al=2;
    in.h.bx=buffer;
    in.h.cx=1;
    in.h.dx=0;
    int86(0x26,&in,&out);
    exit(0);
}
```

(本文收稿日期:1995.2.22)

非多媒体环境下 FLI 动画图像文件的应用

石油大学华东 计算数学 李学春 苟列红

内容提要: 本文分析了多媒体环境下的 FLI 文件格式,给出了 FLI 动画图像文件的解压缩算法,并编写了在非多媒体环境下(DOS 环境下)显示 FLI 动画图像文件的 Turbo C 源程序。

关键词: 多媒体 文件头 帧 量化间隔 DCT 变换

随着多媒体技术的不断发展,人们对图形图像和视频技术的应用越来越广泛,特别是动画图像文件 FLI 的出现,使多媒体技术迅速的普及,它也是软件开放者可以利用动画和音频技术,更加清楚,更加优美的表达软件的各种功能,开发出用户界面优好的软件来。但是,若用户不清楚动画图像文件 FLI 的格式,平时只能在多媒体环境下显示 FLI 文件,在 DOS 环境下就不知道如何操作,这就限制了动画图像文件的充分应用。本文分析了多媒体环境下的 FLI 文件格式,并编写了在非多媒体环境下(DOS 环境下)显示 FLI 动画图像文件的 Turbo C 源程序。

一、FIL 动画图像文件结构

现有的图像文件格式很多,如 BMP, TIF, SPT, PCX, GIF, RIX 等,但这些图像文件一般都是静止的单幅图像,文件结构比较简单,而 FLI 图像文件是动画的,所以,它的结构比一般图像文件的结构复杂的多。FLI 文件可分为文件头和文件体两大部分;文件头又分为三部分,即 FLI 总文件头,每帧文件头和每块文件头,各部分的内容如下:

FLI 总文件头

地址偏移量	长度	内 容
0~3	4B	整个 FLI 文件的长度
4~5	2B	FLI 文件的标志
6~7	2B	FLI 文件所含的帧数
8~9	2B	FLI 文件每帧的宽度
10~11	2B	FLI 文件每帧的高度
12~14	3B	保留
15~16	2B	FLI 总文件头标志
17~18	2B	各帧之间的时间间隔
19~22	4B	FLI 文件第二帧的开始地址
23~26	4B	FLI 文件第一帧的开始地址
27~125	98B	彩色压缩表

每帧文件头

地址偏移量	长度	内 容
0~3	4B	该帧的长度
4~5	2B	该帧的标志
6~7	2B	该帧中所含的块数
8~15	8B	彩色压缩表

每块文件头

地址偏移量	长度	内 容
0~3	4B	本块的长度
4~5	2B	本块的数类型

头文件中的彩色表,是整个 FLI 图像文件的压缩颜色数据,它的值规定了每帧中图像有无背景值,而每帧中的彩色表说明该帧中各种颜色的搭配情况,其第 7 位说明该帧中有无彩色表,为 1 则有,为 0 没有,总头文件中 19~26 这八个字节,主要用来说明各帧图像的存放格式。根据它们的值,可以计算出帧与帧之间是错行存放或逐行存放,错行存放的 FLI 图像文件是按“之”字型,从高频到低频,从左上角到右下角的方式存储。

二、解压缩算法

FLI 动画图像文件压缩后的数据,最基本的存储单位是块,而块图像是按行来压缩的,所以,它没有一个通用的量化表,每行的量化间隔也不一样,它的量化表是在解压缩过程中动态生成的,而量化表的生成,又依赖于每块中头文件的第 4 和第 5 个字节,具体过程如下:

- 1、读出该块的 4,5 两个字节,确定初始码的长度和量化间隔
- 2、读出该块的前四个字节,确定该块的长度
- 3、读出该块的一行数据,根据初始码的长度和量化间隔,按 DCT(离散余弦变换)的逆变换,求出

当前行的第一个字符,若该字符为#号,则表示该块结束,否则,继续下去。

弄清了 FLI 动画图像文件的格式和解压缩方法后,我们可以方便的利用它,并且,可以在非多媒体环境下显示 FLI 文件。下面的 Turbo C 程序,就是我们为一般的 DOS 用户设计的,它可以按多种屏幕模式,在 DOS 下显示 FLI 动画图像文件,本程

序按如下方式执行:FLI [-r[count]] filename,其中可选项-r[count]为重复显示图像文件的次数,它支持 320X200X256 (模式代码为 13), 640X480X256 (模式代码为 5d) 和 1024X768X256 (模式代码为 62) 三种屏幕模式,有兴趣的读者不妨一试。本程序在 DEC 486/66 微机上调试通过

源程序清单

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <string.h>
#include <conio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include "showfli.h"
char far * date;

void SetMode(mode)
int mode;
{
    union REGS r;
    r.h.ah=0;
    r.h.al=mode;
    int86(0x10,&r,&r);
}

void color_pattern(pat)
unsigned char pat[256][3];
{
    union REGS r;
    struct SREGS s;
    r.h.ah=0x10;
    r.h.al=0x12;
    r.x.bx=0;
    r.x.cx=256;
    s.es=FP_SEG(pat);
    r.x.dx=FP_OFF(pat);
    int86x(0x10,&r,&r,&s);
}

void flick320()
{
    unsigned char far *q, far *buf;
    long int u;
    int n, m;
    int i, j;
    n = FHEIGHT;
    m = FWIDTH;
    q = date;
    buf = MK_FP(0xa000,0x0000);
    u = 0;
    outp(0x3C4, 14);
    outp(0x3C5, 0^2);
    for (i = 0; i < n; i++, q += FWIDTH)
    {
        memcpy(buf,q,m);
        u += 320;
        buf += 320;
    }
}

void flick640(void)
{
    unsigned char far *q, far *buf;
    long int u;
    int block, n, m;
    int i, j;
    n = FHEIGHT;
    m = FWIDTH;
    q = date;
    buf = MK_FP(0xa000,0x0000);
    u = 0;
    block = 0;
    outp(0x3C4, 14);
    outp(0x3C5, block^2);
    for (i = 0; i < n; i++, q += FWIDTH)
    {
        if(u<65536L)
        {
            memcpy(buf,q,m);
            buf += 1024;
            u += 1024;
        }
        else
        {
            u -= 65536L;
            buf = MK_FP(0xa000,0x0000);
            block++;
            outp(0x3C4,14);
            outp(0x3C5,block^2);
            u += 1024;
            memcpy(buf,q,m);
            buf += 1024;
        }
    }
}

void flick1024(void)
{
    unsigned char far *q, far *buf;
    long int u;
    int n, m;
    int i, j;
    n = FHEIGHT;
    m = FWIDTH;
    q = date;
    buf = MK_FP(0xa000,0x0000);
    u = 0;
    outp(0x3C4, 14);
    outp(0x3C5, 0^2);
    for (i = 0; i < n; i++, q += FWIDTH)
    {
        memcpy(buf,q,m);
        u += 640;
        if(u<65536L) buf += 640;
        else
        {
            u -= 65536L;
            buf = MK_FP(0xa000,0x0000);
            block++;
            outp(0x3C4,14);
            outp(0x3C5,block^2);
            if (m>640-u)
            {
                memcpy(buf,q+640-u,m-640+u);
                buf += u;
            }
        }
    }
}

int showflick(mode)
int mode;
{
    switch(mode)
    {
        case 0x13:
            flick320();
            break;
        case 0x5d:
            flick640();
            break;
        case 0x62:
            flick1024();
            break;
    }
}
```

```

    flick1024();
    break;
default:
    return(1);
}

read_flihead(fd, header)
int fd;
struct fli_header * header;
{
    char * saveb;
    char * buf = (char *) malloc
(SIZE_HEADER);
    saveb = buf;
    read(fd, buf, SIZE_HEADER);
    header->fhd_size = get_long
(buf);
    buf += LONGSIZE;
    header->fhd_magic = get_short
(buf);
    buf += SHORTSIZE;
    header->fhd_frames = get_short
(buf);
    buf += SHORTSIZE;
    buf += SHORTSIZE;
        /* 宽度 */
    buf += SHORTSIZE;
        /* 高度 */
    buf += SHORTSIZE;
        /* 保留 */
    buf += SHORTSIZE;
        /* 标志 */
    header->fhd_speed = get_short
(buf);
    free(saveb);
    return;
}

read_fheader(fd, fheader)
int fd;
struct frame_header * fheader;
{
    char data[SIZE_FHEADER];
    char * buf = data;
    read(fd, buf, SIZE_FHEADER);
    fheader->fr_size = get_long
(buf);
    buf += LONGSIZE;
    fheader->fr_magic = get_short
(buf);
    buf += SHORTSIZE;
    fheader->fr_chunks = get_short
(buf);
}

read_chunk(fd, chunk)
int fd;
struct chunk_header * chunk;
{
    char data[SIZE_CHEADER];
    char * buf = data;
    read(fd, buf, SIZE_CHEADER);
    chunk->ch_bytes = get_long
(buf);
    buf += LONGSIZE;
    chunk->ch_type = get_short
(buf);
    return;
}

main(argc, argv)
int argc;
char ** argv;
{
    int fd; /* 文件柄 */
    int i, j, s, l, k;
    short ndiff; /* 文件结束行 */
    struct fli_header header;
        /* 保留主文件头 */
    struct frame_header fheader;
        /* 保留每帧的文件头 */
    struct chunk_header chunk;
        /* 保留每块的文件头 */
    char * buf;
        /* 为每块保留内存 */
    int dptr, skchange, sicount;
        /* 保留文件运行前的状态 */
    unsigned char skcount;
    char * cptr;
        /* 主文件头缓冲区 */
    int npack;
        /* 本块中的数据量 */
    unsigned char by;
        /* 本块的字节数 */
    int curx, cury;
        /* 保存当前的 X, Y 坐标 */
    unsigned msec;
    long twopos;
        /* 保存下一块的偏移地址 */
    short repeatcount = 10;
        /* 默认的播放次数 */
    unsigned char notfirst = 0;
        /* 播放标志 */

    short farg = 1;
    int mode;
    unsigned char pat[256][3];
    int m;
    int loop;
    char far * datel;
    if(argc < 2)
    {
        USAGE;
        exit(0);
    }

    if(argv[1][0] == '-')
    {
        farg++;
        if(argv[1][1] != 'r')
        {
            USAGE;
            exit(0);
        }
        repeatcount = atol (& (argv[1]
[2]));
    }
    printf(" The Videomode: (5d, 62,
13)");
    scanf("%x", & mode);
    switch( mode )
    {
        case 0x5d;
            break;
        case 0x62;
            break;
        case 0x13;
            break;
        default;
            printf(" Not support this mode.\
n");
            exit(0);
    }
    date = (char far *) malloc(64000);
    if(date == NULL) {
        printf(" Cannot calloc memory\
n");
        exit(1);
    }
    fd = open(argv[farg], O_RDONLY
|O_BINARY);
    if (fd < 0)
    {
        fprintf(stderr, "Error opening %s
\n", argv[farg]);
        exit(1);
    }
    read_flihead(fd, & header);
    if ((header.fhd_magic & 0x0000ffff)
!= FLI_MAGIC)
    {
        fprintf(stderr, " Not a .fli file\
n");
        exit(0);
    }
    msec = (unsigned)((header.
fhd_speed * 1000.0)/70.0);
    SetMode(mode);
    while (repeatcount--)
        for (i = 0; i < header.fhd-
frames; i++)
        {
            if (((notfirst) && (i == 0)))
            {
                lseek(fd, twopos, 0);

```



```

continue;
}
if (! (notfirst) && (i==1))
{
    twopos = tell(fd);
    notfirst++;
}
read_fheader(fd, &fheader);
for (j=0; j < fheader.fr_chunks;
j++)
{
    read_chunk(fd, &chunk);
    if (chunk.ch_type != FLI-
COPY)
        buf = (char *)malloc(chunk.ch-
bytes);
    if (chunk.ch_type == FLI-
COPY)
    {
        datel=date;
        for(loop=0;loop<10;loop++)
        {
            read(fd, datel, 6400);
            datel += 6400;
        }
    }
    else
        s=read(fd, buf, chunk.ch_bytes
- 6);
    if (s < 0)
    {
        fprintf(stderr, "Error reading image
data. \n");
        perror("xflick");
        exit(0);
    }
    cptr = buf;
    if (chunk.ch_type == FLI-COPY)
    {
        showflick(mode);
    }
    if (chunk.ch_type == FLI-
BLACK)
    {
        for (l=0; l < 32000; l++)
        {
            date[l] = 0;
            date[3200+l]=0;
        }
        showflick(mode);
    }
    if (chunk.ch_type == FLI-COL-
OR)
    {
        npack = get_short(cptr);
        cptr += SHORTSIZE;
        for (l=0; l < npack; l++)
        {
            skcount = * (cptr++);

            /* # 表示颜色没有改变 */
            skchange = * (cptr++);
            /* # 表示颜色已经改变 */
            if (! skchange)
                /* 0 表示文件已经改变 */
                skchange=MAXCOL;
            if (skcount < 0)
                skcount = -skcount;
            m = skcount;
            while(skchange--)
            {
                pat[m][0] = * (cptr++);
                pat[m][1] = * (cptr++);
                pat[m][2] = * (cptr++);
                m++;
            }
            color_pattern(pat);
        }
        if ((chunk.ch_type == FLI-LC) |
| ((chunk.ch_type == FLI-
BRUN)))
        {
            dptr = 0;
            if (chunk.ch_type == FLI-LC)
            {
                curx = 0;
                cury = 0;
                cury += get_short(cptr);
                /* 跳过本行 */
                cptr += SHORTSIZE;
                ndiff = get_short(cptr);
                cptr += SHORTSIZE;
            }
            else
                ndiff = FHEIGHT;
            for (k=0; k < ndiff; k++)
                /* 逐行执行 */
            {
                npack = * (cptr++);
                /* 该行的数据量 */
                for (l=0; l < npack; l++)
                {
                    if (chunk.ch_type == FLI-LC)
                        skcount = * (cptr++);
                    else
                        skcount = 0;
                    if (skcount < 0)
                        curx -= skcount;
                    else
                        curx += skcount;
                    sicount = * (cptr++);
                    if (sicount > 0)
                    {
                        if (chunk.ch_type == FLI-LC)
                        {
                            memcpy(date + ((320 * cury) + curx), cptr,
                            sicount);
                            cptr += sicount;
                            dptr += sicount;
                        }
                        else
                        {
                            if (chunk.ch_type == FLI-LC)
                                by = * (cptr++);
                                while ((sicount++) < 0)
                                    date[(FWIDTH * cury) + (curx
                                    ++)] = by;
                                else
                                {
                                    memcpy (date + dptr, cptr, -
                                    sicount);
                                    cptr -= sicount;
                                    dptr -= sicount;
                                }
                            cury++;
                            curx=0;
                        }
                    }
                    showflick(mode);
                }
                if (chunk.ch_type != FLI-COPY)
                    free(buf);
            }
            delay(msecs);
        }
        SetMode(3);
    }
    /* * * * * 头文件 * * * * */
    struct iii_header
    {
        long fhd_size;
        /* 整个文件的长度 */
        short fhd_magic;
        short fhd_frames;
        short fhd_width;
        short fhd_height;
        short fhd_nothing;
        short fhd_flags; /* 一定为 0 */
        short fhd_speed;
        /* 各帧之间停留的时间 */
        long fhd_next; /* 一定为 0 */
    }
    (转下页)

```

再谈《单片机与 RS-232 串行通讯口联接》

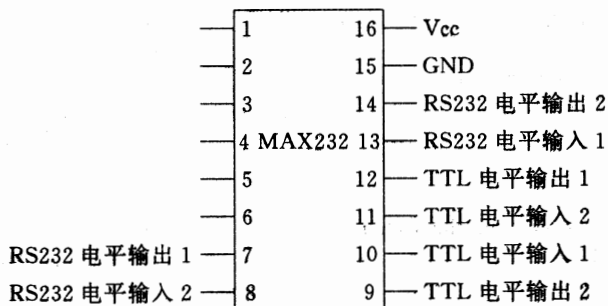
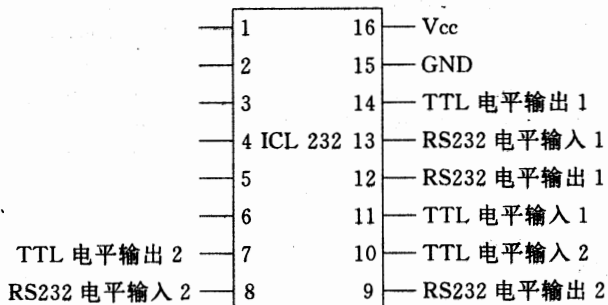
广东新会市供电局变电工区[529100]

高国权

读贵刊 94 年第 4 期第 13 页中关于单片机与 RS-232 串行通讯口联接的文章,笔者认为传统上实现这两种电平转换接口电路一般有两种,即用 MC1488、MC1499 及用分立元件组成的。但是分立元件组成的较为复杂,可靠性差,而用 MC1488、MC1499 组成的接口要±12V 及+5V 电源供电,为双电源。现在市场上已有专为此转换而生产的集成芯片。采用这些集成的电路实现,仅用一个+5V 电源即可了,而且一块芯片具有双向功能。它们完全符合 EIA RS-232 标准规范。这种电平转换芯片有 ICL232、MAX232、TC232 等。

下面简单就此介绍一下:

ICL232、MAX232 都是能在单电源+5V 供电下将 TTL 电平转换为 RS232 电平及将 RS232 电平转换为 TTL 电平,即双向性。使用只要在适当的管脚上加上几只电容及接地即行。下面两图分别为 ICL232 及 MAX232 芯片的主要引脚示意图。(详细用法及原理可由供应商处获得)



(本文收稿日期:1995.2.23)

(接上页)

```
long fhdl_frit; /* 一定为 0 */
char filler[98];
};
struct frame_header
{
    long fr_size; /* 帧的大小 */
    short fr_magic;
    short fr_chunks; /* 图像块数 */
    char fr_filler[8];
};
struct chunk_header
{
    long ch_bytes; /* 块的大小 */
    short ch_type; /* 块的类型 */
};
#define FLL_COLOR 11 /* 彩色压缩表 */
```

```
#define FLL_LC12 /* 压缩行 */
#define FLL_BLACK 13 /* 屏幕设置 */
#define FLL_BRUN 15
#define FLL_COPY 16 /* 无压缩为 64K */
#define FLL_MAGIC 0xaf11
#define FWIDTH 320
#define FHEIGHT 200
#define MAXCOL 256
#define SIZE_HEADER 128
#define SIZE_FHEADER 16
#define SIZE_CHEADER 6
#define USAGE fprintf(stderr, "Usage: %s [-r<count>] file\n", argv[0])
#define grab_short(p,l,h)\
((((char *) (p)) [l] &
```

```
0x000000ff) + (((char *) p)[h] & 0x000000ff) << 8)))
#define get_long(p) ((grab_short(p, 2, 3) << 16) + grab_short(p, 0, 1))
#define get_short(p) (grab_short(p, 0, 1) & 0x0000ffff)
#define SHORTSIZE 2
#define LONGSIZE 4
#define EXTRA_DELAY_FACTOR 3
#define EXTRA_DELAY_FACTOR 3
```

参考文献:

- [1] 王若望等,《Microsoft C/C++ V7.0 实用教程》,学苑出版社,1993.12
- [2] Creative, Microsoft Video for Windows User's Guide

(本文收稿日期:1995.2.15)

在 Turbo c 中实现窗口坐标绘图功能

华北水利水电学院[邯郸 056021] 习华勇

摘 要: 介绍为 Turbo c 设计的一个完整的窗口坐标绘图系统。

关键词: 绘图 视口 窗口坐标 物理坐标 分辨率

一、引 言

Turbo c 是一个使用非常广泛的 c 编译系统。在 Turbo c 中提供了许多绘图函数,但这些函数均采用物理坐标。物理坐标的 Y 轴是向下增加的,其方向和常用的实际坐标相反,且屏幕上的 X 轴和 Y 轴均只能取一个固定区域的整数值。例在 CGA 320×200 图形方式时,屏幕水平方向的取值区间固定为 0 到 319(左到右),垂直方向的取值区间固定为 0 到 199(上到下)。而实际所绘图形函数的自变量 X 及函数值 Y 的取值大部分都取实数值,且取值范围变化也很大。由于存在着这种差别,所以利用物理坐标绘图很不方便。

为 Turbo c 建立窗口坐标绘图系统的目的就是为了解决以上问题。利用本文介绍的系统在屏幕上建立窗口坐标后,坐标的 X 轴从左向右增加,Y 轴将从下向上增加,这和我们平时使用的实际平面坐标的方向相一致。另外屏幕上水平方向(X 轴)和垂直方向(Y 轴)的始点值和终点值可分别据函数自变量的定义域和函数值的范围来确定,且可取任意的实数值。例如当要画一个周期的 SIN(X) 函数图形时,可将屏幕水平方向(X 轴)定义为从 -3.14 到 +3.14,垂直方向(Y 轴)定义为从 -1.0 到 +1.0;而画二个周期的 $2 * \sin(X)$ 函数图形时,X 轴和 Y 轴可分别定义为从 -6.28 到 +6.28 和从 -2.0 到 +2.0。这样定义窗口坐标后,即可利用函数的实际坐标来描述图形中的各个点,而不必进行坐标转化。如绘一个周期的 SIN(X) 函数图形时可用下面的程序段:

```
x = -3.14; moveto_w(x, 0);
while (x <= 3.14)
{ x += 0.1; lineto_w(x, sin(x)); }
```

窗口坐标建立于视口内。视口是在计算机屏幕上定义的一个矩形绘图区域,其位置、大小均任意。视口定义的目的即为了将所绘图形限制在一定范围内,屏幕的其它位置另作它用。利用本系统定

义视口时可不必要知道屏幕的分辨率,其方法见下面 setviewport_w 函数的使用说明。

二、窗口坐标绘图系统函数功能及使用说明

程序 1 是本系统的完整实现。下面是各函数的使用说明。函数中的各参数(如 x、y、x1、y1 等)均为窗口坐标。

(1) unsigned far changex(float x)

功能 将窗口坐标 x 转化为物理坐标。

说明 本函数及 changey 函数一般由其它绘图函数调用,用户很少直接调用。

(2) unsigned far changey(float y)

功能 将窗口坐标 y 转化为物理坐标。

(3) void far setviewport_w(float x1, float y1, float x2, float y2, int clip)

功能 定义图形视口。

说明 定义图形视口时,按着屏幕左下角为 (0,0),右上角为 (1,1) 的窗口坐标来定义。参数 (x1,y1) 为视口的左下角, (x2, y2) 为视口的右上角。例将屏幕右半部定义为一个视口,可用语句: setviewport_w(0.5,0,1,1,0)。clip 为非零值时图形将在视区边缘被裁剪掉。

(4) void far setwindow(float xbegin, float xend, float ybegin, float yend)

功能 定义窗口坐标系统。

说明 xbegin 和 xend 分别是 x 轴的始点和终点, ybegin 和 yend 分别是 y 轴的始点和终点,本函数应在使用下面各绘图函数前首先被调用。

(5) void far putpixel_w(float x, float y, int pixelcolor)

功能 在窗口坐标 (x,y) 处画一点。

说明 点的颜色由 pixelcolor 定义,参见文献 [1]。

(6) float far getx_w(void)

功能 取当前图形指针的 X 窗口坐标。

(7) float far gety_w(void)

功能 取当前图形指针的 Y 窗口坐标。

(8)int far getpixel_w(x,y,real)

功能 送回(x,y)处象素的颜色值。

(9)void far moveto_w(float x,float y)

功能 移动当前图形指针到(x,y)。

(10)void far moverel_w(float dx,float dy)

功能 相对移动当前图形指针。

说明 (dx,dy)为相对于当前指针坐标的增量。

(11)void far lineto_w(float x,float y)

功能 从当前图形点画一条到(x,y)点的直线。

说明 所画线的颜色由 setcolor 设置,线型和线宽由 setlinestyle 设置。

(12)void far line_w(float x1,float y1, float x2, float y2)

功能 从(x1,y1)点到(x2,y2)点画一条直线。

(13)void far linerel_w(float dx,float dy)

功能 相对于当前点画直线。

说明 (dx,dy)为相对于当前指针坐标的增量。

(14)void far outtextxy_w(float x,float y,char *s)

功能 在图形屏幕上(x,y)点处显示一字符串。

说明 可利用此过程在绘出的图形上标注文字串。

(15)void far outtext_w(char *s)

功能 在当前指针位置显示字符串。

(16)void far rectangle_w(float x1,float y1, float x2,float y2)

功能 画长方形。

说明 (x1,y1)定义了长方形的左上角,(x2,y2)定义了右下角。

(17)void far floodfill_w(float x,float y,int border)

功能 填充一封闭区域。

说明 (x,y)是封闭区域中的一点,封闭区域的边界颜色由 border 指定。

(18)void far * getimage_w(float x1, float y1, float x2,float y2)

功能 将指定区域的位图放进缓冲区。

说明 (x1,y1)和(x2,y2)分别为区域的左上角和右下角,本过程能自动计算缓冲区的大小并申请缓冲区,增强了 Turbo c 中 getimage 的功能。

(19)void far putimage_w(float x,float y,void far * bitmap,int op)

功能 把存放于缓冲区中的位图输出到屏幕上。

说明 (x,y)是位图显示屏幕左上角的坐标。bitmap 是指向存放位图的指针,由 getimage_w 过程为其赋值。op 可取下列常量:

COPY_PUT、XOR_PUT、OR_PUT、AND_PUT、NOT_PUT

每个常量规定一个要显示的图形和屏幕上已有图形进行的操作,分别为 copy、xor、or、and 和 not 操作。

(20)void far ellipse_w(float x1,float y1,float x2, float y2,int stangle,int endangle)

功能 画椭圆弧。

说明 (x1,y1)为外切矩形的左上角,(x2,y2)为右下角。stangle 和 endangle 分别为弧的起始角和终止角,均用度表示。当 stangle 为 0,endangle 为 360 时将画一个完整的椭圆。也可用本函数画圆。

另外,可通过全局变量 xbegin0 和 xend0 得到屏幕上 x 轴的始点和终点值,通过 ybegin0 和 yend0 得到 y 轴的始点和终点值。

三、程序编制说明

设程序 1 存放在文件名为“GRAPH_WC.C”的磁盘文件中,使用本系统编程时,应在用户程序中加入#include“GRAPH_WC.C”语句。绘图程序应按下面的步骤编写:

(1)用 initgraph 将屏幕设置为图形模式。

(2)用 setviewport_w 设定视口。省略时缺省视口为整个屏幕。

(3)用 setwindow 定义窗口坐标系统。

(4)用窗口绘图系统中的各函数绘图。

(5)用 closegraph 关闭图形系统。

四、实例

程序 2 是本系统应用的一个示例程序,该程序中使用了系统中大部分的函数。其功能为首先将屏幕左边 2/3 区域定义为一个图形视口,然后不断地将坐标的 X 轴和 Y 轴给于不同的起点和终点,并画出一个周期 $(-\pi,\pi)$ 的 SIN(X)函数曲线。程序中也利用 getimage_w 和 putimage_w 函数实现了一个简单的图形动画。程序均在 TURBO C 2.0 系统下运行通过。(程度 1、2 附于下页)

参考文献

[1]潘金贵等. TURBO C 程序设计技术,南京大学出版社,1991.1


```

/*****
* 程序一 窗口坐标绘图系统源程序 *
* Program one; GRAPH_WC.C *
*****/
#include <graphics.h>
#include <alloc.h>
float xbegin0=0,xend0=1,ybegin0=0,yend0=1;
unsigned far changex(float x)
{ unsigned ax;struct viewporttype xy;
  getviewsettings(&xy);
  ax=(xy.right-xy.left)*(x-xbegin0)/
    (xend0-xbegin0);
  return(ax);
}
unsigned far changey(float y)
{ unsigned ay;struct viewporttype xy;
  getviewsettings(&xy);
  ay=(xy.bottom-xy.top)*(y-yend0)/
    (ybegin0-yend0);
  return(ay);
}
void far setviewport_w(float x1,float y1,float x2,float
y2,int clip)
{ int ax1,ax2,ay1,ay2;
  ax1=(getmaxx()*x1);ay1=-(getmaxy()*(y1-
1));
  ax2=(getmaxx()*x2);ay2=-(getmaxy()*(y2-
1));
  setviewport(ax1,ay2,ax2,ay1,clip);
}
void far setwindow(float xbegin,float xend,float ybegin,
float yend)
{ xbegin0=xbegin;xend0=xend;
  ybegin0=ybegin;yend0=yend;
}
void far putpixel_w(float x,float y,int pixel)
{ unsigned ax,ay;
  ax=changex(x);ay=change(y);
  putpixel(ax,ay,pixel);
}
float far getx_w(void)
{ float x;
  x=xbegin0+(getx()*(xend0-xbegin0))/
  getmaxx();
  return(x);
}
float far gety_w(void)
{ float y;
  y=yend0+(gety()*(xbegin0-xend0))/
  getmaxy();
  return(y);
}

```

```

}
int far getpixel_w(float x,float y)
{ int ax,ay;int r;
  ax=changex(x);ay=change(y);
  r=getpixel(ax,ay);
  return(r);
}
void far moveto_w(float x,float y)
{ unsigned ax,ay;
  ax=changex(x);ay=change(y);
  moveto(ax,ay);
}
void far moverel_w(float dx,float dy)
{ float x,y;
  x=getx_w();y=gety_w();
  moveto_w(x+dx,y+dy);
}
void far lineto_w(float x,float y)
{ unsigned ax,ay;
  ax=changex(x);ay=change(y);lineto(ax,ay);
}
void far line_w(float x1,float y1,float x2,float y2)
{ moveto_w(x1,y1);lineto_w(x2,y2);}
void far linerel_w(float dx,float dy)
{ float x,y;
  x=getx_w();y=gety_w();
  lineto_w(x+dx,y+dy);
}
void far outtextxy_w(float x,float y,char *s)
{ moveto_w(x,y);outtext(s);}
void far outtext_w(char *s)
{ outtext(s);}
void far rectangle_w(float x1,float y1,float x2,float y2)
{ unsigned ax1,ax2,ay1,ay2;
  ax1=changex(x1);ay1=change(y1);
  ax2=changex(x2);ay2=change(y2);
  rectangle(ax1,ay1,ax2,ay2);
}
void far floodfill_w(float x,float y,int border)
{ int ax,ay;
  ax=changex(x);ay=change(y);
  floodfill(ax,ay,border);
}
void far * getimage_w(float x1,float y1,float x2,float
y2)
{ int ax1,ay1,ax2,ay2;void far * bitmap;
  ax1=changex(x1);ay1=change(y1);
  ax2=changex(x2);ay2=change(y2);
  bitmap=malloc(imagesize(ax1,ay1,ax2,ay2));
  getimage(ax1,ay1,ax2,ay2,bitmap);
  return(bitmap);
}

```

```

    }
    void far putimage_w(float x,float y,void far * bitmap,
int op)
    { int ax,ay;
      ax=changex(x);ay=change(y);
      putimage(ax,ay,bitmap,op);
    }
    void far ellipse_w(float x1,float y1,float x2,
float y2,int stangle,int endangle)
    { int ax1,ay1,ax2,ay2,xr,yr;float x,y;
      ax1=changex(x1);ay1=change(y1);
      ax2=changex(x2);ay2=change(y2);
      xr=abs(ax1-ax2)/2;yr=abs(ay1-ay2)/2;
      x=x1+(x2-x1)/2;y=y1+(y2-y1)/2;
      ax1=changex(x);ay1=change(y);
      ellipse(ax1,ay1,stangle,endangle,xr,yr);
    }

```

```

/* * * * * *

```

```

* 程序二 示例主程序 *

```

```

* Program two;MAINDEMO.C *

```

```

* * * * *

```

```

#define COUNT 4

```

```

#include <graphics.h>

```

```

#include <math.h>

```

```

#include "graph-wc.c"

```

```

main()

```

```

{
    float x,y,r;int i,gd=DETECT,gm;
    char s[5];void far * buffer;
    initgraph(&gd,&gm,"");
    if (graphresult() != grOk) {
        printf("%s","Graphics Error!");
        exit(1);
    };
    setviewport_w(0,0,0.75,1,0);
    x=3.14+COUNT*0.8;y=1.0+COUNT*0.4;
    for(i=1;i<=COUNT+1;i++){
        setwindow(-x,x,-y,y);clearviewport();
        rectangle_w(-3.14,1,3.14,-1);
        line_w(-x,0,x,0);line_w(0,-y,0,y);
        outtextxy_w(0.05,-0.05,"0");
        outtextxy_w(0.05,1,"1");
        outtextxy_w(2.7,-0.05,"3.14");
        outtextxy_w(0.05,-1,"-1");
        outtextxy_w(-3.14,-0.05,"-3.14");
        sprintf(s,"%0.2f",y);
        outtextxy_w(0.05,y,s);
        sprintf(s,"%0.2f",x);
        outtextxy_w(x-x/7,-0.05,s);
        sprintf(s,"%0.2f",-y);

```

```

        outtextxy_w(0.05,-y+y/15,s);
        sprintf(s,"%0.2f",-x);
        outtextxy_w(-x,-0.05,s);
        r=-3.14;moveto_w(r,0);
        do{
            lineto_w(r,sin(r));r+=0.1;
        }while (r<=3.14+0.1);
        x-=0.8;y-=0.4;
        if(i==1){
            buffer=getimage_w(-3.14,1,3.5,-1);
        }
        outtextxy_w(-x,-y,"Press any key...");
        getch();
    }

```

```

rectangle_w(0.5,-0.1,2.5,-1);

```

```

ellipse_w(0.5,-0.1,2.5,-1,0,360);

```

```

floodfill_w(1.5,-0.5,getcolor());

```

```

y=1.1;

```

```

do{ x=-3.14;

```

```

    do{

```

```

        putimage_w(x,y,buffer,XOR_PUT);

```

```

        putimage_w(x,y,buffer,XOR_PUT);

```

```

        x+=0.3;

```

```

    }while(x<=3.14-1);

```

```

    y-=0.6;

```

```

}while(y>=-0.8);

```

```

cleardevice();closegraph();

```

```

}

```

(收稿日期:1995.3.6)

简 讯

国家信息中心隆重推出 《中国计算机应用进展专辑》

为推进中国计算机应用进程,国家信息中心隆重推出《中国计算机应用进展专辑》。《专辑》汇集了50余家国内外著名计算机公司客户的应用实例和30多个行业的应用典型以及中外著名计算机专家、企业界名人如 Bill Gates 等向计算机用户的赠言,共计40余万字,邹家华副总理为《专辑》题写了书名。

《专辑》可以帮助计算机用户更好地认识各计算机公司和计算机品牌的优势,从而选择最佳合作伙伴和产品;可以帮助计算机公司找到商业机会和工作方向,以制定发展战略;可以使人们更好地认识计算机的作用。该书售价30元/册,邮挂费3元。需要者请与以下地址联系:

联系单位:国家信息中心《经济与信息》杂志社

联系地址:北京市西城区三里河路58号

联系电话:(010)8526577 8502528

邮政编码:100045

开户行:中国银行总行营业部

帐户:《经济与信息》编辑部

帐号:5170100042

高级语言直接在打印机上绘图

冶金部鞍山热能研究院 辽宁鞍山[114004] 任铁良

摘要:利用打印机提供的图象方式输出命令,用高级语言编程,直接控制打印机,实现高级语言直接在打印机绘图;本文应用 C 语言编制出了打印机画点、线(框)、虚线、圆(椭圆、弧、扇形)的基本函数,利用这些基本函数,可在打印机上绘出各种复杂图形。最后给出一绘制三维图形的函数,该函数可绘制出任意给定的球坐标方程对应的三维立体图。

关键词:高级语言、C 语言、打印机、LQ1600 系列、图形

一、前言

目前,计算机用户的微机均配置有打印机,其打印机大多用于文字输出,若需打印输出图形,又无专用图形打印软件时,用户只能简单地硬拷显示屏上的图形;而绝大多数打印机命令集中又都有图象输出命令,应用高级语言,特别是无图形功能的高级语言,如 CLIPPER、FORTRAN 等,编程控制打印机进行绘图,则可以弥补该高级语言其图形功能的不足,又可充分发挥其打印机的功能。

数据库中的数据项经常需要以图形的形式打印输出,若所用的数据库语言无图形功能,一般常用的打印输出图形方法是,先用数据库语言(如 DBASE)编程生成图形文本文件,再进入有图形功能的高级语言(如 BASIC),读取文本文件中的数据,进行屏幕绘图(直方图、比例图、折线图等),需要打印机输出图形时,再进行硬拷贝。

上述方法虽然可在打印机上输出图形,但显得繁琐、效率低,且不够灵活。笔者通过打印机本身提供的图象命令,应用 C 语言编制了打印机绘图的基本函数,其中包括:设置打印图形内存缓冲区函数、和画点、线(框)、虚线、圆(椭圆、弧、扇形)函数及打印图形内存缓冲区函数。由这些基本图形函数,通过编程可在打印机上绘制由这些基本图素构成的复杂图形。

本文以 C 语言编程,并以 LQ1600 系列打印机为例,介绍图形打印原理和基本图形函数;也可用其它高级语言,如 FORTRAN 或 DBASE 等编程,实现打印机绘图。把打印控制命令稍加改动,程序可用于其它类型的打印机。

二、打印机输出图象原理

了解打印机输出图象原理,需要知道打印头

工作情况,LQ1600 系列打印机,打印头纵分布 24 根打印针,当打印头移过纸面时,一部分针被电流脉冲激发而撞击色带,在撞击点处色带与打印纸相碰,一个点即被印在打印纸上。

24 针图象模式是 LQ1600 系列打印机提供图象输出的一种方式,这种方式有 5 种密度,现以三倍密度为例,说明图象输出方法;三倍密度方式在水平方向每英寸打印 180 点,当打印头从纸上移过时,每隔 1/180 英寸其必收到有关 24 针击发的指令,在每一位置,其可击发从 0~24 间任意数目的打印针,即对应于所打印的每列,打印机必须接收 24 位数据,由于打印机在同计算机通讯中使用 8 位信息,故每个打印位置需要 3 个字节的信息。

为设定打印每列中哪些针击发,首先将纵列划分成各有 8 针的 3 个部分,由于对每部分的 8 针均有 256 种可能的组合,这可用一字节 of 的二进制数与其对应,其中二进制数的最高位对应于每部分 8 针的上针,二进制数的最低位对应于每部分 8 针的下针;要击发某针,可把二进制数对应该针的位置 1。

由于每列 24 针,每一位置必须用 3 个二进制数确定该位置针的动作,3 个 8 位(一字节)二进制数的顺序依次对应该位置的上、中、下 8 针,在三倍密度图象模式下,对打印机传送 3 个一字节二进制码,便可确定该位置的打印图象点阵。

设置打印机图象代码的同时,还要指定象素列数代码,打印机接收到列数代码后,它将以后接收到的 3 倍列数个字节的的信息译成图象模式(24 针图象),并将其打印出来。

LQ1600 系列打印机打印图象指令为:

ESC * m n1 n2 data

其中,ESC 为 ASCII 码 27(十六进制 1B),*

为 ASCII 码 42(十六进制 2A), m 表示图象方式选择码, 它确定图象打印针数和图象密度, 例如, 设置 24 针三倍密度模式(水平密度 180 点/英寸)时, m 为 39; $n1$ 和 $n2$ 确定图象列数, 图象列数为 $256 * n2 + n1$, $n1$ 和 $n2$ 取值范围为 $0 \sim 255$; $data$ 是图象数据, 24 针三倍密度图象时, 图象数据的字节数为 $3 * (256 * n2 + n1)$ 。

有关打印机图象指令请参阅所用操作手册。

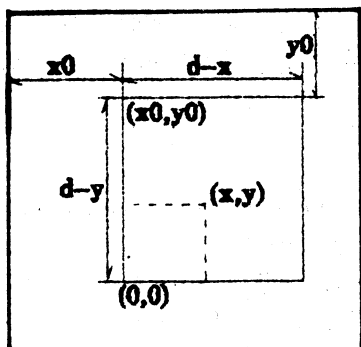


图 1 确定打印象素点

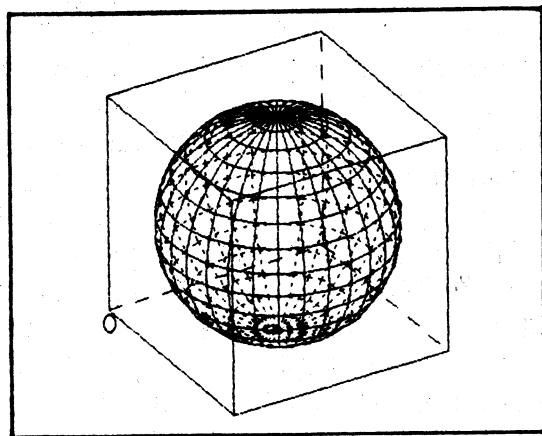


图 2 打印输出的球体

三、高级语言控制打印机绘图

利用高级语言的向打印机输出命令或过程、函数, 向打印机发送上述图象指令, 可把打印机置成所要求的图象模式, 并按其接收到的图象数据输出图形。

为在打印纸上绘出图形, 先在内存中开辟一块图形缓冲区, 对应所打印的图形, 对图形缓冲区中的数据进行预处理, 最后把处理后的图形缓冲区中的数据, 以图象方式打印, 即可在打印纸上输出所

需图形。

在打印纸上的 $(x0, y0)$ 处绘出一幅 $d-x * d-y$ 的图形($x0$ 和 $y0$ 分别为图幅左上角距纸最左边和最上边的距离, $d-x$ 和 $d-y$ 分别为图幅的宽和高, 单位均为点阵距离), 需要开辟一块 $4 * 2 + 3 * d-x * ([(d-y-1)/24] + 1)$ 个字节的内存空间, 其中前 8 个字节分别用于保存 $x0$ 和 $y0$ 及 $d-x$ 和 $d-y$, 其均为两字节整数, 第 9 个字节后用于存放图形数据, 图形数据的存储规则为: 先存放第 1 行(以 24 点阵为 1 行)第 1 列(以 1 点阵为 1 列)的上、中、下 8 点各一字节数据, 字节的高位对应上点, 字节的低位对应下点, 其次存放下列数据, 直到最后列, 再次存放下行数据, 直到最后行。

图 1 所示, 以图形图幅的左下角为坐标原点 $(0, 0)$, 在 (x, y) 处绘点, 对应图形缓冲区中的数据应进行如下设置:

首先确定 (x, y) 点对应缓冲区中的第几个字节 (m), 再确定其对应该字节的第几位 (n), 最后对该字节的该位置 1 (若原为 1, 则不变)。

经过计算得到:

$$m = 4 * 2 + 3 * ([yy/24] * d-x + x) + [(yy \bmod 24)/8]$$

$$n = yy \bmod 8$$

其中, $yy = d-y - y - 1$, $[]$ 为取整运算符, \bmod 为取模运算符。

用相应的高级语言可编制出对打印机画点的函数或过程, 通过调用画点函数或过程, 可编程实现打印机画线(框)、虚线、圆(椭圆、弧、扇形)的基本函数或过程; 有了这些基本函数或过程, 便可画出由这些图素构成的复杂图形的图形, 这样可在无图形功能的数据库中(如 CLPPER), 直接用数据库中的数据, 无需借助于屏幕, 即可在打印机上绘制出各种统计图(如直方图、比例图、折线图等)。

四、C 语言打印机绘图例程

附后给出的 C 程序包括: 设置打印图形内存缓冲区函数 `setprnxyz`, 打印图形内存缓冲区函数 `pgraphics`, 画点函数 `point`, 画线(框)函数 `pline`, 画虚线函数 `plineb`, 画圆(椭圆、弧、扇形)函数 `pcircle`。

本 C 程序编制了一个函数 `stereo`, 只要对其传递给定的球面坐标方程函数名和视角等参数, 即可打印出对应的三维立体图形。函数 `stereo` 能打印任意出球面坐标方程对应的立体图形。

程序仅以打印一球体(球面坐标函数为 `sphere`)为例, 其输出结果见图 2。

如在其它打印机上输出图形, 只需把以上涉及到打印控制部分的地方稍加改动即可。

以上程序在 486 微机 LQ1600K 和 BJ-10SX 喷墨打印机上及 Borland C++ 3.1 下运行通过, 可打印的最大图形幅面(以点阵距离为单位)尺寸为 $d_x * d_y = 8 * 64KB = 512KB$ 。

附 打印机绘图 C 程序

>TYPE PG.C

```
#include<stdio.h>
#include<alloc.h>
#include<stdlib.h>
#include<math.h>
void main(void)
{ void stereo(char *,float,float,float,int,float (*)(float,
float));
  float sphere(float,float);
  void pgraphics(char *);
  char *p, *setprnxyz(int,int,int,int,int);
  p=setprnxyz(100,50,500,400,0);
  stereo(p,acos(-0.5),asin(0.5),1.0,15,sphere);
  pgraphics(p);
}

/* 画点函数:
   参数    P -- 同函数 pgraphics
           x,y -- 点横纵坐标          */
void point(char *p,int x,int y)
{ int d_x=((int *)p)[2],d_y=((int *)p)[3],yy;
  char c=0x01;
  if(x>=0&&x<d_x&&y>=0&&y<d_y)
  { yy=d_y-y-1; c<=<=7-yy%8;
    p[sizeof(int)*4+3U*(yy/24*d_x+x)+yy%24/
8]|=c;
  }
}

/* 画线(框)函数:
   参数    p -- 同函数 pgraphics
           x1,y1 -- 线始点的横纵坐标
           x2,y2 -- 线终点的横纵坐标
           box -- 线框标志, 1 画线 0 画框          */
void pline(char *p,int x1,int y1,int x2,int y2,int box)
{ int d,i;
  float x,y,dx,dy;
  if(box)
  { if(abs(y2-y1)>abs(x2-x1))
    { d=abs(y2-y1);
      dx=(float)(x2-x1)/d; dy=y1>y2? -1.0:1.0;
    }
    else
    { d=abs(x2-x1);
      dx=x1>x2? -1.0:1.0; dy=d?(float)(y2-y1)/d:
0.0;
    }
    for(x=x1,y=y1,i=0;i<=d;i++,x+=dx,y+=
dy)
    point(p,(int)(x+0.5),(int)(y+0.5));
  }
```

```

}
else
{ pline(p,x1,y1,x2,y1,1);
  pline(p,x2,y1,x2,y2,1);
  pline(p,x2,y2,x1,y2,1);
  pline(p,x1,y2,x1,y1,1);
}
}

/* 画虚线函数:
   参数    p,x1,y1,x2,y2 -- 同函数 pline
           l1,l0 -- 实虚长度          */
void plineb(char *p,int x1,int y1,int x2,int y2,int l1,int
l0)
{ int t,l=l1+l0;
  float k,d,d1,x,y;
  if(l>0)
  { if(x1==x2)
    { if(y1>y2)
      { t=y2; y2=y1; y1=t;
      }
      while(y1<y2)
      { l1=y1+l1>y2? y2-y1:l1;
        pline(p,x1,y1,x1,y1+l1,1);
        y1+=l1;
      }
    }
    else
    { if(x1>x2)
      { t=x2; x2=x1; x1=t;
        t=y2; y2=y1; y1=t;
      }
      k=(float)(y2-y1)/(x2-x1);
      d=1*cos(atan(k)); d1=d*l1/l;
      x=x1; y=y1;
      while(x<(float)x2)
      { d1=x+d1>x2? x2-x:d1;
        pline(p,(int)(x+0.5),(int)(y+0.5),(int)(x+d1+
0.5),
          (int)(y+k*d1+0.5),1);
        x+=d; y+=k*d;
      }
    }
  }
}

/* 画圆(椭圆,弧,扇形)函数:
   参数    p -- 同函数 pgraphics
           x0,y0 -- 圆心坐标
           r -- 圆半径
           a1,a2 -- 起始和终止弧度, 负弧度到圆心连
线
           e -- 圆度, 1.0 圆          */
void pcircle(char *p,int x0,int y0,int r,float a1,float a2,
float e)
{ float const PI2=4*atan2(1.0L,0.0L);
  float er,t,ta1,ta2,ts;
```

```

if(x0>=0&& y0>=0&& r>0&& e>0.0)
{ er=e*r;
  ta1=fabs(a1); ta2=a1!=0.0||a2!=0.0? fabs(a2):
PI2;
  while(ta1>ta2) ta2+=PI2;
  if(a1<0)
    pline(p,x0+(int)(r*cos(ta1)+0.5),y0+(int)(er*
sin(ta1)+0.5),
    x0,y0,1);
  if(a2<0)
    pline(p,x0+(int)(r*cos(ta2)+0.5),y0+(int)(er*
sin(ta2)+0.5),
    x0,y0,1);
  ts=atan(1/sqrt(er>(float)r? 2*er*er:2.0*r*r-
1));
  for(t=ta1;t<ta2;t+=ts)
    point(p,x0+(int)(r*cos(t)+0.5),y0+(int)(er*sin
(t)+0.5));
}
}

```

/* 设置打印图形内存缓冲区函数

```

参数 x0,y0 -- 打印图幅相对打印纸左上角横纵
坐标
d_x,d_y -- 打印图幅的横纵长度
box -- 打印图幅边框标志, 0 有边框 */
char * setprnxyz(int x0,int y0,int d_x,int d_y,int box)
{ char * p;
  unsigned int m=sizeof(int)*4,n=3U*d_x*((d_y-
1)/24+1),mn,i;
  if((p=(char *)malloc(m+sizeof(char)*n))==
NULL)
  { printf("Not enough memory to allocate buffer. \n");
    exit(1);
  }
  ((int *)p)[0]=x0; ((int *)p)[1]=y0;
  ((int *)p)[2]=d_x; ((int *)p)[3]=d_y;
  mn=m+n;
  for(i=m+1;i<=mn;i++) p[i]=0;
  if(! box) pline(p,0,0,d_x-1,d_y-1,0);
  return p;
}

```

/* 打印图形内存缓冲区函数

```

参数 p -- 打印图形内存缓冲区指针 */
void pgraphics(char * p)
{ int i,j,m,n,n0,y0,d_y;
  union {int d_x; struct {unsigned char n1,n2;} cn;} xn,
xn0;
  FILE * pp;
  if((pp=fopen("prn","rb+"))==NULL)
  { printf("Printer not on line. \n");
    exit(1);
  }
  xn0.d_x=((int *)p)[0]; y0=((int *)p)[1];
  xn.d_x=((int *)p)[2]; d_y=((int *)p)[3];
  p+=sizeof(int)*4;
}

```

```

n0=3*xn0.d_x; n=3*xn.d_x; m=(d_y-1)/24+
1;
fputc(27,pp); fputc(51,pp); fputc(1,pp);
for(i=1;i<=y0;i++) fputc(10,pp);
fputc(27,pp); fputc(51,pp); fputc(24,pp);
for(i=1;i<=m;i++)
{ fputc(27,pp); fputc(42,pp); fputc(39,pp);
fputc(xn0.cn.n1,pp); fputc(xn0.cn.n2,pp);
for(j=1;j<=n0;j++) fputc(0,pp);
fputc(27,pp); fputc(42,pp); fputc(39,pp);
fputc(xn.cn.n1,pp); fputc(xn.cn.n2,pp);
for(j=1;j<=n;j++) fputc(*p++,pp);
fputc(10,pp);
}
fclose(pp);
}

```

/* 空间坐标结构

```

x_0,y_0 -- 三维坐标原点相对图幅左下角平面横
纵坐标
x0,y0,z0 -- 球坐标原点的三维坐标
lx,ly,lz -- 三维坐标在观察方向的投影比例
axz,ayz -- 在观察方向,X轴Y轴与Z轴的夹角
pff -- 函数(二变元)指针 */
struct sxyz {int x_0,y_0;float x0,y0,z0,lx,ly,lz,axz,ayz;
float (*pff)(float,float);};

```

/* 设置空间坐标结构并画三维坐标函数

```

参数 p,b,c,lp,pf -- 同函数 stereo
ps -- 空间坐标结构指针 */
void p_xyz(char * p,float b,float c,float lp,float (* pf)
(float,float),
struct sxyz * ps)
{ int k=((int)(b/atan2(1.0L,0.0L))%4+4)%4;
  d_x=((int *)p)[2],d_y=((int *)p)[3],dx,dy,
d,xd[4],yd[4],i,j;
  float l=2*lp,cc=cos(c),cx,cy,cz,tz,xc,yc,xs,ys,x,
y,z,r,ax,ay,az,n;
  dx=0.9*d_x; dy=0.9*d_y;
  x=cc*cos(b); y=cc*sin(b); z=sin(c);
  r=sqrt(x*x+y*y+z*z);
  ax=asin(x/r); ay=asin(y/r); az=asin(z/r);
  tz=tan(az);
  cx=fabs(cos(ax)); cy=fabs(cos(ay)); cz=fabs(cos
(az));
  ps->axz=(ay>0? 1:-1)*acos(-tan(ax)*tz);
  ps->ayz=(ax>0? 1:-1)*acos(-tan(ay)*tz);
  xc=cx*cos(ps->axz); yc=cy*cos(ps->ayz);
  xs=cx*sin(ps->axz); ys=cy*sin(ps->ayz);
  x=dx/(fabs(xs)+fabs(ys)); y=dy/(fabs(xc)+fabs
(yc)+cz);
  n=(x<y? x:y)/l;
  ps->lx=cx*n; ps->ly=cy*n; ps->lz=cz*n;
  n*=l;
  ps->x_0=((xs>0.0? 0.0:-xs)+(ys>0.0? ys:0.
0))*n;
  ps->x_0+=(d_x-dx/x*n)/2;
}

```

```

ps->y_0=((xc<0.0?-xc:0.0)+(yc<0.0?-yc:0.0))*n;
ps->y_0+=(d_y-dy/y*n)/2;
ps->x_0=ps->y_0=ps->z_0=lp;
d=c*z*n; l=d/15;
xd[0]=x*n; xd[1]=-y*n; xd[2]=-xd[0]; xd[3]=-xd[1];
yd[0]=x*n; yd[1]=y*n; yd[2]=-yd[0]; yd[3]=-yd[1];
for(j=0;j<=2;j++)
for(d_x=ps->x_0,d_y=ps->y_0+(j==2?d:0),i=0;i<=3;i++)
{ dx=d_x+(j==1?0:xd[i]); dy=d_y+(j==1?d:yd[i]);
if((j!=2&&z>0.0||j!=0&&z<0.0)&&(i==k||j!=1&&i==(k+3)%4))
plineb(p,d_x,d_y,dx,dy,l,l);
else
pline(p,d_x,d_y,dx,dy,1);
d_x+=xd[i]; d_y+=yd[i];
}
pcircle(p,ps->x_0,ps->y_0-l/2,0.0,0.0,1.5);
ps->pff=pff;
}

```

/* 平面坐标结构

参数 ix,iy —— 平面横纵坐标(整数)

x,y —— 平面横纵坐标 */

```
struct sxy {int ix,iy;float x,y};
```

/* 计算空间坐标对应平面坐标函数

参数 ps —— 空间坐标结构指针

rb,rc —— 方向坐标(球坐标)

pso —— 平面坐标结构指针 */

```

void c_xy(struct sxyz *ps,float rb,float rc,struct sxy *pso)
{ float x,y,z,r=(ps->pff)(rb,rc),rcc;
rcc=r*cos(rc);
x=ps->lx*(ps->x_0+rcc*cos(rb));
y=ps->ly*(ps->y_0+rcc*sin(rb));
z=ps->lz*(ps->z_0+r*sin(rc));
pso->x=ps->x_0+x*sin(ps->axz)-y*sin(ps->ayz);
pso->y=ps->y_0+z*x*cos(ps->axz)+y*cos(ps->ayz);
pso->ix=pso->x+0.5; pso->iy=pso->y+0.5;
}

```

/* 设置空间坐标结构并画三维坐标函数

参数 p —— 同函数 pgraphics

b,c —— 观察方向(无穷远处的球坐标)

lp —— 球坐标的可视半径

n —— 纬线数目

pf —— 球坐标系方程函数(二变元)指针 */

```
void stereo(char *p,float b,float c,float lp,int n,
```

```

float (*pf)(float,float))
{ float const PI=3.14159;
float rb,rc,ar,al,au,ad,dbc=PI/n;
int l,l0,i,j,m=2*n;
struct sxyz s;
struct sxy so,sd,su,sl,sr;
p_xyz(p,b,c,lp,pf,&s);
l=lp*(s.lx+s.ly+s.lz)/3/n/2; l=l?l:1; l0=2*l;
for(i=1,rc=-PI/2+dbc;i<=n;i++)
{ c_xy(&s,0.0,rc,&sl);
c_xy(&s,dbc,rc,&so);
for(j=1,rb=dbc;j<=m;j++)
{ c_xy(&s,rb,rc+dbc,&su);
c_xy(&s,rb,rc-dbc,&sd);
c_xy(&s,rb+dbc,rc,&sr);
ad=sd.y!=so.y||sd.x!=so.x?atan2(sd.y-so.y,sd.x-so.x);ad;
al=sl.y!=so.y||sl.x!=so.x?atan2(sl.y-so.y,sl.x-so.x);al;
if(sin(al-ad)<0.0L)
{ au=su.y!=so.y||su.x!=so.x?atan2(su.y-so.y,su.x-so.x);au;
ar=sr.y!=so.y||sr.x!=so.x?atan2(sr.y-so.y,sr.x-so.x);ar;
if(sin(ad-ar)<0.0L) plineb(p,so.ix,so.iy,sd.ix,sd.iy,l,l0);
else pline(p,so.x,so.y,sd.x,sd.y,1);
if(sin(au-al)<0.0L) plineb(p,so.ix,so.iy,sl.ix,sl.ix,l,l0);
else pline(p,so.ix,so.iy,sl.ix,sl.iy,1);
}
else
{ pline(p,so.ix,so.iy,sd.ix,sd.iy,1);
pline(p,so.ix,so.iy,sl.ix,sl.iy,1);
}
sl.x=so.x; sl.y=so.y; so.x=sr.x; so.y=sr.y;
sl.ix=so.ix; sl.iy=so.iy; so.ix=sr.ix; so.iy=sr.iy;
rb+=dbc;
rc+=dbc;
}
}
}
/* 球坐标系中球体方程函数
参数 rb,rc —— 同函数 c_xy */
float sphere(float rb,float rc)
{ float r=1.0;
return r;
}

```

参考文献

- [1]任铁良,图形功能在 CLIPPER 中的实现,《电脑与微电子技术》,1994 年第 2 期

(本文收稿日期:1995.3.6)

在 C 程序中使用 DOS 内部命令

西南石油学院 杜小平
四川石油地调处子弟校 刘 运

摘 要:本文利用直接向键盘缓冲区送命令的方法,实现了在 C 语言中使用 DOS 内部命令,给出了相应的 C 函数。

关键词:DOS 内部命令 键盘缓冲区 C 语言

在开发应用程序时,我们有时会遇到在应用程序中使用 DOS 命令的情况。对于 DOS 外部命令,C 语言为我们提供了诸如 system、spawnl 等进程管理函数,用于加载 DOS 外部命令,从而实现了在应用程序中直接使用 DOS 外部命令的功能。对于 DOS 内部命令,C 语言没有提供相应的函数。若用户自己编写与 DOS 内部命令相应的程序将是一件工作量较大的事情。要执行 DOS 内部命令,首先必须加载 COMMAND.COM 文件,使用户暂离应用程序而回到操作系统,然后执行内部命令。但键入内部命令不应让用户完成而应由应用程序完成。本文采用在加载 COMMAND.COM 之前,将内部命令装入键盘缓冲区来模拟按键的方法实现了自动执行 DOS 内部命令。

键盘缓冲区位于内存 0040:001E~0040:003C 处,它被设计成先进先出的循环队列形式。缓冲区在内存中为每个键入字符提供两个字节。对一个字节的 ASCII 码来讲,第一个字节是 ASCII 码,第二个字节是字符的扫描码。将 ASCII 码输入键盘缓冲区时,仅需将其填入第一个字节,而不必管第二个字节。对于不能用 ASCII 码表示的按键或按键组合,第一个字节是零,第二个字节是数字代码,该数字代码一般是该键的扫描码。缓冲区内部没有固定的头和尾。其工作由头指针和尾指针控制,一面从尾指针处取走键码,一面从尾指针处填入键码。缓冲区头指针指向第一个键入字符的位置,尾指针指向最后一个键入字符后面的第一个位置。当要输入一个新的字符时就将新字符排在尾指针指示的位置上,尾指针随之改变。当缓冲区的最高位充满时,输入的新字符覆盖缓冲区的底部。头指针位于内存 0040:001A 处,尾指针位于 0040:001C 处。虽然指针占两个字节,但有用的只是低位字节。指针值在 0X1E 和 0X3C 之间变化,故可以存放 16 个

字符。将头指针与尾指针设置为相同值,即可清除缓冲区。附图表示了缓冲区内数据结构的可能情况。

本文提供的 DOS 内部命令执行函数 dos_com 的原型是: int dos_com(char * command, int mode)。其工作原理是:首先将头指针与尾指针均指向 0040:001E 处,清除缓冲区。接下来将 DOS 命令字符串 command 中的每一个字符依次输入当前尾指针所指向的地址。每输完一个字符,都将尾指针加 2。当命令字符串输入完毕后,再输入回车键(0XD)。然后加载 COMMAND.COM,退到 DOS 命令状态后即可自动执行输入的 DOS 内部命令。函数中提供了两种返回 C 程序的方式。当 mode=0 时,函数在输入命令字符串后,再向缓冲区输入“EXIT”命令和回车键。这样,当 DOS 内部命令执行完后,自动返回 C 程序。由于“EXIT”和两个回车键占有了 6 个字符,故命令串 COMMAND 的最大长度为 10 个字符。当命令串长度超过 10 个字符时,函数不向缓冲区送命令而直接返回 -1 值。当 mode>0 时,DOS 内部命令执行完后,由用户键入“EXIT”命令返回 C 程序。这样命令串的最大长度可达 15 个字符。超过这一长度时函数也不向缓冲区送命令而直接返回 -1 值。

本文所述方法也适用于 DOS 外部命令和其他可执行文件,建议采用 mode>0 的方式,以免在执行这些程序时因使用键盘输入打乱键盘缓冲区而无法自动返回 C 程序。所附程序 DOSCOM.C 中给出了在 C 程序中执行“TYPE DOSCOM.C”命令的例子。DOSCOM.C 是本文所附的 C 程序,执行程序时它应位于当前目录下。该程序在 COMPAQ DESKPRO 386/20e、AST PA4/50D 等机型上和 Turbo C 2.0 环境下通过。

0040:003C		
3A		
38		
36	'B'	
34	'U'	
32	'F'	
30	'F'	
2E	'E'	
2C	'R'	
2A		
28		
26		
24		
22		
20		
1E		
1C	2A	← 尾指针
0040:001A	36	← 头指针

附图 键盘缓冲区结构图

附 DOSCOM.C 程序清单

```

/* 示范程序 */
#include <conio.h>
main() {
    textbackground(LIGHTBLUE);
    clrscr();
    gotoxy(28,10);
    textcolor(LIGHTRED);
    cprintf("This is an example\n");
    gotoxy(5,13);
    textcolor(YELLOW);
    cprintf("Press any key to execute 'TYPE DOSCOM.
C',
and then press 'EXIT' to return\n");
    getch();
    dos_com("TYPE DOSCOM.C",1);
    /* 执行"TYPE DOSCOM.C" */
    clrscr();
    gotoxy(22,12);
    textcolor(LIGHTRED);
    cprintf("Now we have returned the program.\n");
    gotoxy(22,14);
    textcolor(YELLOW);
    cprintf("Press any key to exit the program.\n");
    getch();
    textbackground(0);

```

```

textcolor(15);
clrscr();
}

/* DOS 内部命令执行函数 */
int dos_com(char *command,int mode){
    unsigned char far * head = (unsigned char far *)
0x40001a;
    unsigned char far * tail = (unsigned char far *)
0x40001c;
    unsigned char far * tail_pointer = (unsigned char far
*)0x40001e;
    int com_length,i,x,y;
    char buffer[4096];
    unsigned char ccc[]="exit";
    gettext(1,1,80,25,buffer);
    x=wherex();
    y=wherey();
    window(1,1,80,25);
    clrscr();
    printf("\nEnter EXIT to return\n");
    com_length=strlen(command);
    if(mode&&com_length>15) return(-1);
    if(! mode&&com_length>10) return(-1);
    * head=0x1e; /* 清除缓冲区 */
    * tail=0x1e;
    for(i=0;i<com_length;i++){
        * tail_pointer=command[i];
        /* 在尾指针处存入键码 */
        tail_pointer+=2; /* 尾指针加 2 */
        * tail+=2;
    }
    * tail_pointer=0xd; /* 输入回车键 */
    tail_pointer+=2;
    * tail+=2;
    if(! mode){
        for(i=0;i<4;i++){ /* 输入 EXIT 命令 */
            * tail_pointer=ccc[i];
            tail_pointer+=2;
            * tail+=2;
        }
        * tail_pointer=0xd;
        * tail+=2;
    }
    system(""); /* 加载 COMMAND.COM */
    puttext(1,1,80,25,buffer);
    gotoxy(x,y);
}

```

参考文献

- [1] [美] Robert Jourdan,《IBM 编程指南》,北京:电子工业出版社,1988
- [2] 金泳,《用程序解释按键系列》,中国计算机用户,1994,(1)
- [3] 杜小平,《为 C 语言增添 SHELL 函数》,电脑与微电子技术,1994,(4) (本文收稿日期:1995. 3. 13)

微机系统掉电保护的实用技术

郑州解放军电子技术学院 王成义

内容摘要:本文主要介绍了微机系统电源掉电时的信息保护。具体分析了掉电保护的基本原理和基本电路,并对电源发生掉电后,微机系统的数据和状态信息的保护介绍了实现方法。

关键词:掉电 保护 数据

一、引言

在微机系统或在工业控制、测量和由单片机组成的控制系统中,一般存储器最大的缺点就是信息的易失性,电源的瞬间掉电,会使存在读写存储器中的信息全部丧失。由于条件的限制和系统本身的原因,微机系统电源故障是很难避免的。电源掉电将会使运行中的微机系统造成重复性的工作,输入了很长时间的程序、数据,一旦掉电,则前功尽弃,运行了很长时间的程序,掉电则只能从头开始。尤其是在野外作业对重要的数据采集、信号处理和作业控制,将造成不可弥补的损失,因此,在微机系统中采取掉电保护措施是非常必要的。

二、掉电保护的基本原理

图1所示,微机系统电源发生故障,其输出电压值并不是马上突变到0V,而是有一个延迟时间。掉电保护系统就是利用电源电压下降的短暂的延

迟时间,将RAM中的重要数据和运行中的CPU的各种状态信息加以保护。在电源正常供电时,使CPU中的各种状态信息可以恢复,RAM中的数据没有丢失,机器中的程序可以继续运行。在电源发生故障时,充分利用负载电容的充放电,使供给微机系统的直流电压按指数规律逐渐下降,同时掉电保护系统检测电源也下降,在电源电压下降到微机最小额定工作电压之前的延迟时间里,将重要数据和状态加以保护,之后主动停机。从以上可以看出掉电保护系统一般由三部分构成。

1、掉电检测电路。其功能一是用来检测电源故障,二是发出掉电信号,以控制掉电保护开始。

2、停机电路。其功能是在掉电保护工作结束后,使微机系统主动停机,以防止电源电压下降到最小额定工作电压以下后,工作不稳定而产生错误命令,导致不应有的操作。

3、数据和状态保护电路。利用不易失存储器芯片或者加后备电源的方法来保护数据和状态信息。

三、掉电保护的基本电路

(一)掉电检测电路

掉电检测电路是用来检测电源掉电的“瞬时”,使微机系统采取掉电保护措施并发出停机命令。

1、交流220V输入掉电检测器

利用交流掉电检测电路可以对交流供电系统的故障进行检测,交流掉电检测器输入为交流220V,输出的掉电信号POWER DOWN加到微机CPU的中断输入端上。当输入的交流电压一旦降到检测器的触发电平时,则检测器发出一个有效的低电平POWER DOWN信号,而CPU利用掉电瞬间直流供给的延迟时间,执行一个中断程序进行掉电保护。检测器的触发电平是可以调整的,一般调整在刚好为系统最低允许工作电压以下,以防止当交流电压的波动,达到最低工作电压时,检测器产生错误信号。一般情况下,对于额定工作电压为 $220 \pm 10\%$ 的系统,触发电压值调整在小于等于

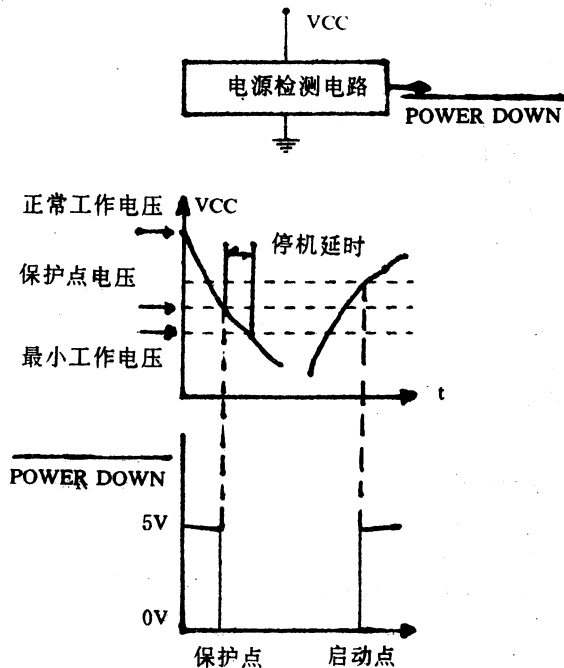


图1

198V, 而对于 $220V \pm 20\%$ 的系统, 则调整在小于等于 176V。这个电压的确定主要关系到微机的直流稳压电源的技术指标。

2、直流掉电检测器

图 2 为触发电平可调整的 VCC 直流掉电检测电路。电路以一个电压比较器为核心, 由于电压比较器可在 4.5—30V 范围内工作, 因此, 可使电路使用于 30V 以下电源掉电检测。图中的稳压二极管工作在 2.4V, 它作为电压比较器的输入基准电压 V_X , 比较输入电压 V_Y 正比于电源电压 V_{CC} , 其比例系数 K 与 R_2 、 R_3 和反馈电阻 R_H 及比较器输出有关。若比较器输出为高电平时, 比例系数为 K_H , 比较器输出为低电平时, 比例系数为 K_L , 很显然, $K_H > K_L$ 。由于反馈电阻 R_H 很大, 因此, 比例系数 K 的值主要取决于电阻 R_2 和 R_3 的分压, 即 $K = R_3 / (R_3 + R_2)$, 而正反馈电阻 R_H 的作用主要是加速比较器的翻转, 减小翻转过渡时间, 使输出有确定的逻辑值。从图示可以看出, 电压比较器为同相连接, 即 $V_Y > V_X$, 输出 $\text{POWER DOWN} = "1"$; $V_Y < V_X$ 时, 输出 $\text{POWER DOWN} = "0"$, 电路翻转门限为 $V_Y = V_X$ 。当翻转门限电平一定时, 掉电保护时的 V_{CC} 值为

$$V_{CC \text{ DOWN}} = V_X [(R_2 + R_3) / R_3]$$

因此调节 R_3 中的电位, 可得到适当的掉电保护电压值。

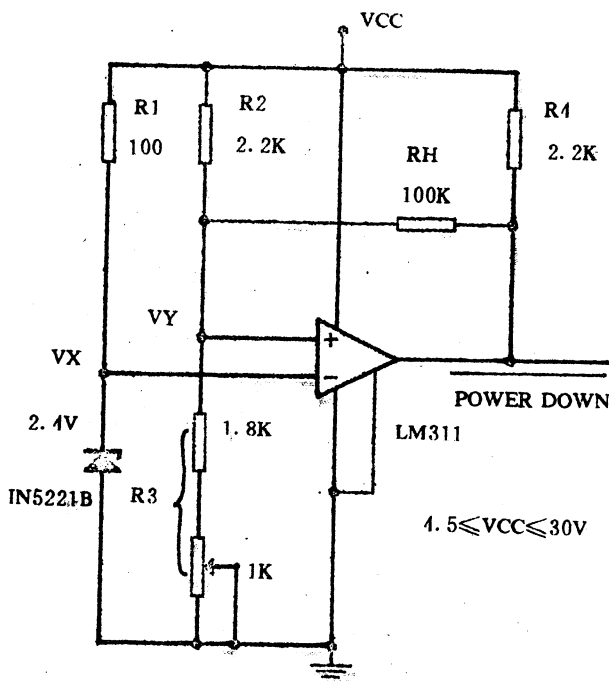


图 2

(二) 停机电路

停机电路使微机系统在掉电的延迟时间里做完掉电保护工作之后“受控”停机, 以避免由于电压下降的自然停机而造成错误操作。如图 3 所示为一种停机电路。这实际是两个或非门构成的双稳态触发器, 在掉电信号 POWER DOWN 有效之后, 只有收到微机发来的 ENABLE SHUT DOWN 有效信号, 电路才发出有效的停机信号 SHUT DOWN 。而在上电时, 只要 POWER DOWN 为高电平, 则 SHUT DOWN 失效, 机器即可重新启动。

(三) 数据与状态保护电路

数据与状态的保护可以用加后备电源和利用非易失 NVRAM 保护数据, 非易失 NVRAM 它实际是由静态 RAM 存储阵列和 EPROM 存储阵列共同构成的存储器, 它即可以象 RAM 存储器那样进行高速随机读、写操作, 又可以在电源掉电情况下, 利用 10MS 的时间把全部 RAM 阵列中的数据保存在不易失的 EPROM 阵列中。而在电源正常之后, 又可利用 10MS 时间, 把不易失存储阵列中的数据全部恢复到 RAM 阵列中。并且这种保存、恢复操作的初始化只需要一个存储器的写读周期。因此, 利用 NVRAM 来构成掉电保护电路是一种有效的方法。

四、利用 NVRAM 构成掉电保护电路

1、2004NVRAM 与系统总线接口

由于 2004NVRAM 芯片除控制保存/恢复操作的 $\overline{\text{NE}}$ 管脚之外, 地址、数据以及片选信号 $\overline{\text{CE}}$, 写控制信号 $\overline{\text{WE}}$, 读控制信号 $\overline{\text{OE}}$ 的设置完全与一般 SRAM 芯片一样, 因此接口方法一样。保存/恢复功能是 NVRAM 特有的性能, 因此接口中增加了保存/恢复操作控制电路。

保存操作是把 NVRAM 中 SRAM 阵列中的数据保存到不易失存储阵列中, 在控制上是在 $\overline{\text{NE}}$ 为低电平条件下的写操作, 即保存操作初始化时需同时使 2004 的 $\overline{\text{NE}}$ 、 $\overline{\text{CE}}$ 、 $\overline{\text{WE}}$ 为有效低电平。而恢复操作是把不易失存储阵列中的数据恢复到 SRAM 中, 是 $\overline{\text{NE}}$ 为低电平时的读操作, 所以在初始化操作时需同时使 $\overline{\text{NE}}$ 、 $\overline{\text{CE}}$ 、 $\overline{\text{OE}}$ 为有效低电平。实现以上要求的控制逻辑可用存储器寻址方式和 I/O 端口寻址方式。图 4 是利用 I/O 端口寻址方式来控制 $\overline{\text{NE}}$ 的示意图。从图可见, 用一位可寻址的数据锁存器的输出来发出控制 $\overline{\text{NE}}$ 的 $\overline{\text{NE ENABLE}}$ 信号。这个锁存器可以是任意并行接口中的一位, 图中是用 74LS74 中的一个 D 触发器来实现, 如果要使 $\overline{\text{NE}}$ 有效, 则向 74LS74 写一个数据 01H, $\overline{\text{NE ENABLE}} = "0"$ 。反之要使 $\overline{\text{NE}}$ 失效, 则向 74LS74 写一个数据

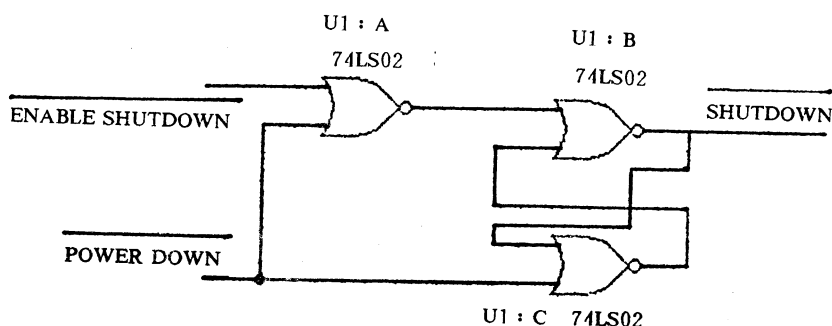


图 3

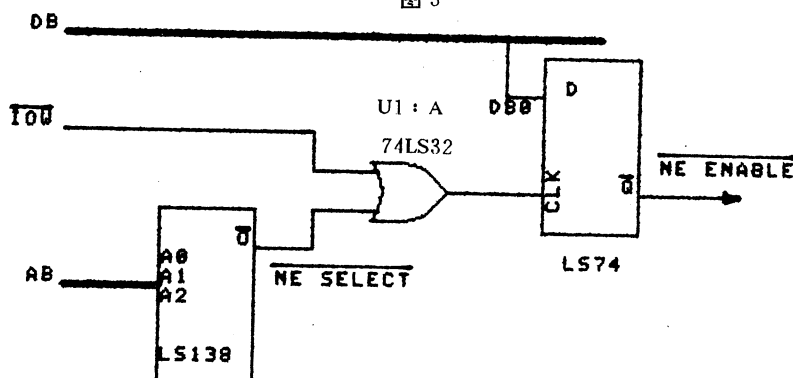


图 4

00H, NE ENABLE="1".

2、利用 2004NVRAM 构成的掉电保护电路

图 5 为用 4 片 2004NVRAM 构成的掉电保护电路,这是一个以 INTEL 8088 为 CPU 的准 16 位微机系统。4 片 2004 的片选信号由系

译码器 74LS138 给出,而 4 片 2004 的 $\overline{\text{NE}}$ 由电路中 74LS74 输出的 $\overline{\text{NE}} \text{ ENABLE}$ 统一控制。电路中,一个 220V 掉电检测器输出的 $\overline{\text{POWER DOWN}}$ 信号加到 8088 的不可屏蔽中断输入端。当交流电源掉电时,交流掉电检测器发出掉电信号,从而触发一个不可屏蔽的中断,利用交流掉电后的直流延迟时间,CPU 执行一个中断程序,把重要数据传送到 2004 中,之后对 2004 执行保存操作的初始化(先使 $\overline{\text{NE}}$ 有效,然后对每片 2004 执行一条写周期),只要系统的直流电源在交流电源掉电后至少保持 10MS 有效,则保存操作就全部完成。

参考文献

- [1] 张思东著,《微处理机接口技术》,1988年,中国计量出版社
- [2] 张明达著,《8086/8088 系列微型计算机及其应用》,1987年,中国工业大学出版社

(本文收稿日期:1994.12.18)

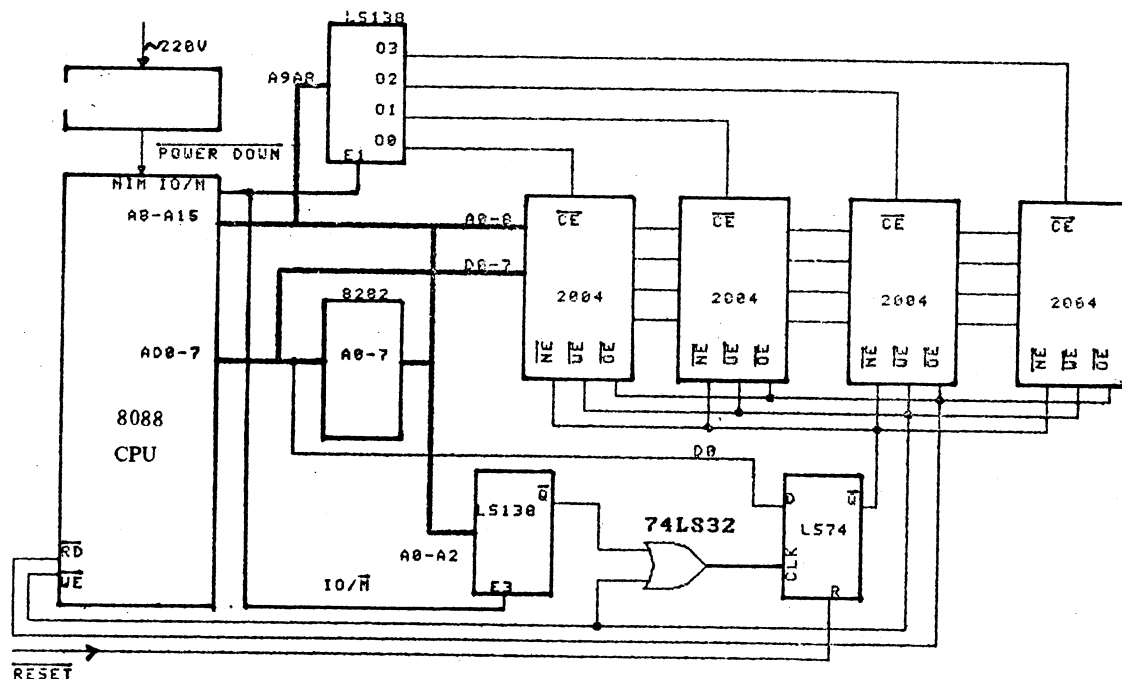


图 5

FOXBASE+数据库视窗编辑器的设计及实现

河北煤炭建工学院 张永强

摘要:本文介绍了用 FOXBASE+语言设计通用数据库视窗编辑器的思想,并给出了实现程序。

关键词:数据库 视窗 记录 锁定 过程

一. 引言

我们在设计应用软件时,经常需要对数据库进行浏览,或者对整个数据库进行全屏幕的增加记录、修改记录、插入记录、删除记录、恢复记录等编辑工作。但是 FOXBASE+系统本身提供的窗口显示及修改命令 BROWSE 却不能方便地应用到我们的应用程序当中,因为 BROWSE 将自动地对屏幕进行清除、对颜色进行重新设定,这样一来,不仅破坏了我们的系统界面,而且也不能在屏幕上显示一些必要的提示信息,操作画面既不美观生动,而且因为操作命令都是英文说明,故而亦不便操作,所以与应用系统不能有机地融为一体。为此,笔者设计了自己的数据库视窗编辑器程序(brow),该程序除具有原系统中 BROWSE 的绝大部分功能外,还增加了插入记录、删除记录、恢复记录以及改变编辑方向等功能,而且用户可以根据系统的需要,很灵活地在当前屏幕的任何地方,开辟任意大小的视窗,可以随意设定屏幕的前景色、背景色,可以将必要的提示或帮助信息放置在屏幕的任意位置上,更重要的是该程序可以方便地作为一个通用的外部过程或内部过程很灵活地加入到我们的应用程序当中。

二. 设计思想

1. 参数合法性检查

程序入口有五个参数需要输入,用来决定视

窗的屏幕位置。如果窗口的坐标参数给的过大,程序将默认为屏幕大小。

2. 获取数据库名称

首先,程序自动检查当前工作区中处于打开状态的数据库,如果存在,则自动获取该数据库的名称,否则,提示客户输入数据库名称。

然后,程序判断是否存在该数据库,如果存在,则打开该数据库的名称,并自动生成相应的结构描述库;否则,返回系统。

3. 窗口设计

首先,分析出数据库中各字段的名称和宽度,计算出视窗中能够显示的字段个数及其显示的起始坐标。

然后,根据当前记录位置,将若干条记录在视窗里显示出来。

4. 全屏幕编辑

利用 get-read 语句和 inkey()语句对数据库的记录进行编辑,实现了视窗内的光标的上下移动以及记录的上下滚动、上下换页、左右翻页等操作。

5. 工具菜单

利用工具菜单可以对若干个字段进行锁定操作;可以根据编辑需要灵活地改变编辑方向;可以在当前记录位置完成记录的插入、删除、恢复等工作。

三. 程序清单

```

*****
* 数据库视窗编辑器程序 brow *
*****
para row_s,col_s,row_e,col_e,flag
    && 窗口的左上角 row_s,col_s;右
    下角 row_e,col_e;
    && 修改标志 flag=0 允许修改,flag
    =1 不允许修改

set menu off
set talk off
set stat off
set scor off
set safe off
set esca off
set proc to brow
set colo to gr+/1,7/0,4
clea

if row_s<3
    row_s=3
endi
if row_e>22
    row_e=22
endi
if col_e>77
    col_e=77
endi

```

```

if len(dbf())=0
    name=spac(8)
    @23,0 say [请输入数据库名称:]
get name
read
if name=spac(8)
    retu
endi
name=name-['.dbf']
if ! file(name)
    @24,0 say [文件不存在]
    retu
endi
use &name
endi
@23,0 clear to 24,79
fp=dbf()
r=recn()
copy to ddds stru exte
use ddds
num=recc()
dime zdmc(num), zcdc(num), zdxsl
(num)
do while ! eof()
    zdmc(recn())=field_name
    zcdc(recn())=iif(len(field_name)
    <= field_len, field_len, len(field_
name))
    skip
endd
use &fp
go r
@0,0 say [数据库视窗编辑器]
set colo to gr+/3,7/0
@row_s-1,col_s,row_e,col_e box
chr(201)+chr(205)+chr(187)+chr
(186),
+chr(188)+chr(205)+chr(200)+
chr(186)+' '
set colo to 7/0
@row_s,col_e+1 clear to row_e,col_e
+3
@row_e+1,col_s+2 clear to row_e+
1,col_e+3
row_s=row_s+1
row_e=row_e-1
col_s=col_s+1
col_e=col_e-1
do xsjlh    && 显示记录号
zdsdcd=col_s    && 字段锁定长度
zdxscd=col_s    && 字段显示长度
zdsd=0          && 每屏显示的字段
个数
zdsd=0          && 字段锁定数
bjfx=.t.        && 编辑方向
page=15         && 翻屏行数
row=row_s
n=1
do xsdqy
do bjdqzd
pack
set proc to
retu
*****
* 显示当前页          *
*****
proc xsdqy
priv i,j,r
r=recn()
skip -row+row_s
set colo to 7/4
@row_s-1,zdsdcd clear to row_s-1,
col_e
zdxscd=zdsdcd
    && 字段显示开始=字段锁定长度
j=n
do while j<=num && 字段总个数
    if zdxscd+zcdc(j)+2<col_e
        zdxsl(j)=zdxscd+3
        @row_s-1,zdxsl(j) say zdmc(j)
        zdxscd=zdxscd+zcdc(j)+2
        j=j+1
    else
        exit
    endif
endd
i=row_s    && 起始行显示
zd0=j-1
set colo to gr+/3
@row_s,zdsdcd clear to row_e,col_e
do while ! eof().and.i<=row_e
    if dele()
        set colo to 0/3
        @i,col_s say [ * ]
    else
        set colo to gr+/3
        @row,col_s clear to row,col_s+1
    endi
    j=n
do while j<=zd0
    && 本屏最后一个字段
    mc=zdmc(j)
    @i,zdxsl(j) say &mc pict [ @z ]
    j=j+1
endd
i=i+1
skip
endd
go r
zdxscd=zdsdcd
    && 字段显示开始=字段锁定长度
retu
*****
* 编辑当前字段          *
*****
proc bjdqzd
do while .t.
    if eof()
        go bott
    endi
    if zdxscd+zcdc(n)+2<col_e
        zdxsl(n)=zdxscd+3
        mc=zdmc(n)
        set colo to gr+/5,7/0
        @row,zdxsl(n) get &mc pict
        [ @z ]
        if flag=0
            read
        else
            clear gets
            set colo to gr+/1,7/1
            k=[ ]
            @24,79 get k
            read
        endi
    endif
    k=mod(read(),256)
    if dele()
        set colo to 0/3
    else
        set colo to gr+/3
    endi
    @row,zdxsl(n) say &mc pict [ @z ]
do case
    case k=2
        if n>1
            n=n-1
            zdxscd=zdxscd-zcdc(n)-2
        endi
    case k=3
        if n<zd0
            n=n+1
            zdxscd=zdxscd+zcdc(n)+2

```

```

endi
case k=4      && 上移一行
do syyh
case k=5      && 下移一行
do xyyh
case k=6      && 上翻一页
do sfyy
case k=7      && 下翻一页
do xfyy
case k=15     && 回车
if bjfx
if n<zd0
n=n+1
zdxscd=zdxscd+zdcn(n)+2
else
n=1
zdxscd=col_s
do xyyh
endif
else
do xyyh
endi
case k=33     && 工具菜单
do gjcd
case k=34     && 左移窗口
do zyck
case k=35     && 右移窗口
do yyck
case k=14
retu
endc
do xsjhl
endd
retu
*****
* 下移一行 *
*****
proc xyyh
if recn()>=recc()
do zjll
retu
else
skip
endi
if row<row_e
row=row+1
else
scro row_s,col_s,row_e,col_e,1
do xsyh
endif
retu

```

```

*****
* 上移一行 *
*****
proc syyh
if recn()=1
retu
endi
skip -1
if row>row_s
row=row-1
else
scro row_s,col_s,row_e,col_e,-1
do xsyh
endif
retu
*****
* 显示一行 *
*****
proc xsyh
priv j
if dele()
set colo to 0/3
@row,col_s say [ * ]
else
set colo to gr+/3
@row,col_s clea to row,col_s+1
endi
j=zdsd+1
do while j<=zd0
mc=zdmc(j)
@row,zdxsl(j) say &mc pict '@z'
j=j+1
endd
retu
*****
* 下翻一页 *
*****
proc xfyy
priv i
if recn()=recc()
do zjll
endi
i=0
do while i<page
if recn()<recc()
skip
if row<row_e
row=row+1
else
scro row_s,col_s,row_e,col_e,1
do xsyh

```

```

endi
endi
i=i+1
endd
retu
*****
* 上翻一页 *
*****
proc sfyy
priv i
i=0
do while i<page
if recn()>1
skip -1
if row>row_s
row=row-1
else
scro row_s,col_s,row_e,col_e,-1
do xsyh
endi
endi
i=i+1
endd
retu
*****
* 右移窗口 *
*****
proc yyck
if n>=num
retu
endi
r=recn()
row0=row
n=n+1
do xsdqy
go r
row=row0
retu
*****
* 左移窗口 *
*****
proc zyck
if n<=zdsd+1.or.n=1
retu
endi
r=recn()
row0=row
n=n-1
do xsdqy
go r
row=row0

```



```

retu
* * * * *
* 工具菜单 *
* * * * *
proc gjcd
priv o,i,k,y0,m,tx(7)
m=7
dime tx(m)
tx(1)='文件顶部'
tx(2)='文件尾部'
tx(3)='删除记录'
tx(4)='恢复记录'
tx(5)='插入记录'
tx(6)='字段锁定'
tx(7)='编辑方向'
set colo to 0/7,7/4
@0,18 clea to 0,79
i=1
do while i<=m
    @0,9+i*9 prom tx(i)
    i=i+1
endd
menu to k
set colo to 7/1
@0,18 clea to 0,79
if k=0
    retu
endi
k=str(k,1)
do p&k
retu
* * * * *
* 文件顶部 *
* * * * *
proc p1
go top
set colo to 7/3
do xsdq
row=row_s
retu
* * * * *
* 文件尾部 *
* * * * *
proc p2
go bott
skip -15
if bof()
    go top
endif
set colo to 7/3
do xsdq

row=row_s
retu
* * * * *
* 删除记录 *
* * * * *
proc p3
if ! dele()
    dele
endi
do xsyh
retu
* * * * *
* 恢复记录 *
* * * * *
proc p4
if dele()
    reca
endi
do xsyh
retu
* * * * *
* 插入记录 *
* * * * *
proc p5
if flag=1
    retu
endi
set colo to gr+/3
scro row,col_s,row_e,col_e,-1
inse blan befo
do xsyh
n=1
zdxscd=col_s
retu
* * * * *
* 字段锁定 *
* * * * *
proc p6
priv i
set color to 7/1,0/3
zdsd=0
@0,62 say [字段锁定数] get zdsd pict
'@z 99999' rang 1,num
read
@0,62 clear to 0,78
if zdsd<1.or.zdsd>zdsd-1
    zdsd=0
endif
zdxscd=col_s    && 字段显示长度
i=1
do while i<=zdsd    && 字段锁定个数

if zdxscd+zdcde(i)+2<col_e
    zdxscd=zdxscd+zdcde(i)+2
    i=i+1
else
    exit
endif
endd
zdsdcd=zdxscd
n=zdsd+1
retu
* * * * *
* 编辑方向 *
* * * * *
proc p7
bjfx=iif(bjfx,.f.,.t.)
retu
* * * * *
* 增加记录 *
* * * * *
proc zjjl
priv k
if flag=1
    retu
endi
set colo to 7/1,7/5
k=[ ]
@24,col_s say [增加新的记录吗(Y/N)?] get k vali k $[YyNn]
read
@24,col_s clea to 24,col_s+25
if k $[Yy]
    set colo to gr+/3
    scro row_s,col_s,row,col_e,1
    appe blan
    do xsyh
    n=1
    zdxscd=col_s
endi
retu
* * * * *
* 显示记录号 *
* * * * *
proc xsjlh
set colo to 7/1
@row_s-3,col_s say [库名]+fp
@row_s-3,col()+5 say [记录号:]+
ltrim(str(rece(),5))+[/];
+ltrim(str(recc(),5))+spac(5)
retu

```

(本文收稿日期:1995.2.15)

巧用 DOSKEY

杭州商学院 黄中伟

摘要: 本文全面详细地介绍了 DOSKEY 这一内存驻留程序的使用方法和技巧

关键字: 驻留程序, 历史命令, 缓冲区, 宏

DOSKEY 是 MS-DOS5.0 以上版本提供的一个实用性很强的内存驻留程序, 加载后约占 4KB 内存。DOS 在没有提供 DOSKEY 程序以前, 就象在 FOXBASE 出现以前一样, 广大计算机用户对 DBASE 那种在某一命令发错或想再使用以前的命令(称为历史命令)时, 都必须重新输入一次的做法深表遗憾。而有了 DOSKEY 程序后, 不仅可以非常方便、灵活地调用和编辑历史命令, 并且可以建立批处理和宏命令, 这对广大 DOS 用户来说, 确实方便了许多。笔者通过参看有关资料, 并结合自己的一些实践经验, 总结了一些 DOSKEY 程序的使用技巧, 供大家参考。

一. DOSKEY 命令的格式和用法

格式: d: path \DOSKEY [/reinstall] [/bufsize = size] [/macros] [/history] [/insert | overstrike] [macroname = [text]]

其中:

/reinstall 重新安装一份 DOSKEY 程序, 并清理 DOSKEY 缓冲区

/bufsize = size 指定用于存贮命令队列和宏命令的缓冲区大小。其默认值为 512 字节, 最小值为 250 字节

/macros 显示所有宏命令。可缩写为 /M, 也可用 ">" 将其输入到一个文件保存起来

/history 显示所有历史命令。可缩写为 /H, 也可用 ">" 将其输入到一个文件加以保存

/insert 使键盘处于默认的插入状态

/overstrike 使键盘处于默认的改写状态

macroname = [text] 建立一条宏命令用来执行一条或多条 DOS 命令。其中 macroname 为宏命令名, 由用户自己定义, text 为一条或多条 DOS 命令

/? 显示帮助信息

用法: 使用时只要运行一下 DOSKEY 程序即

可, 至于参数和其他 DOS 命令一样, 可以按需单独或组合选择。

二. 历史命令的重复使用

只要运行过 DOSKEY 命令, 以后输入的所有 DOS 命令都将被按顺序记忆保存在 DOSKEY 缓冲区内, 形成一个有序的命令队列。然后通过一些功能键对这命令队列中的任何命令进行调用或编辑后执行。

下面提供一些常用的调用功能键及其用法。

↑	调用上一条 DOS 命令
↓	调用下一条 DOS 命令
PaUp	调用运行 DOSKEY 程序后的第一条 DOS 命令
PaDn	调用最后用过的一条 DOS 命令
F7	显示命令队列中的所有命令及其序号。和 DOSKEY/H 的执行结果相似
ALT+F7	从缓冲区中清除命令队列中的所有命令
F8	查找和显示符合条件的命令。用此命令时先指定要查找命令的头一个或头几个字符, 然后按 F8 键即可按序往前显示相匹配的命令
F9	显示指定序号的命令。用时可先用 F7 进行显示查看该命令的序号。

而当要执行一条和某一历史命令相近的命令时, 同样需要先对此命令进行编辑。一些常用的编辑键如下:

F1~F6	这6个功能键的功能与安装 DOSKEY 程序前一样
←	向左移动一个字符, 但不会将此字符删除
→	向右移动一个字符
End	快速移动光标到命令行尾

「工欲善其事必先利其器」强调的是利器, 其实反之亦然: 「器欲利其事必先善其工」。试问《三国演义》中的关公刀和关公的功力: 「酒尚温时斩华雄」, 不就是善工与利器的典范么? 从这一期起独辟这个栏目《善工与利器》, 以飨读者。

编者的话

Home 快速移动光标到命令行首

同时,编辑时为了使键盘处于默认的插入模式,可执行 DOSKEY /insert 命令。当然此时仍可以通过“INSERT”键来临时更换插入或改写模式。

另外,运行了 DOSKEY 程序后,还可以将多条 DOS 命令放在一行内连续执行,而中间只须用 Ctrl+T 键分开。

例如: DIR *.COM<Ctrl+T>PAUSE<Ctrl+T>DIR *.EXE

三. 巧建批处理

虽然 MS-DOS5.0 以后的版本增加了 EDIT 命令,可以方便地用来建立批处理,但在没有 EDIT 文件时,还得用其他软件或 COPY CON[:]命令来建立批处理。而使用 DOSKEY 命令的可选项 /H 把命令队列重新定向到一个 BAT 文件的方法,为建立批处理提供了一条捷径。

下面通过一个在“ABC”用户文件子目录下,调用 UCDOS 中的 WPS 的实例来说明建立批处理的方法。

① 先用 ALT+F7 键清除掉 DOSKEY 缓冲区中的命令队列,若刚安装了 DOSKEY 程序,则此步可省略;

② 按序键入以下命令:

```
@ ECHO OFF
CD \ABC
PATH C:\UCDOS
RD16
KNL
WB
SP
RDSL
WPS
QUIT
CD \
```

若 UCDOS 子目录下有现成的包含括号内命令的批处理,用 CALL 调用即可;

③ 键入 DOSKEY/H>WPS.BAT (将命令队列保存到 WPS.BAT 中);

④ 用 EDIT 或其他办法将 WPS.BAT 中的最后一句“DOSKEY/H>WPS.BAT”删除;

⑤ 运行 WPS.BAT 即可进入 UCDOS 中 WPS。

四. 宏命令的建立

如果经常以某种参数运行某一 DOS 命令,或

者经常执行某一条较长的 DOS 命令,则可用 DOSKEY macroname=[text] 命令把这些 DOS 命令定义为一个易记忆的宏命令,使之简单化。并且可将多个 DOS 命令放在一起,中间用 \$T 分隔。

例如:键入 DOSKEY ML=MD ABC \$TCD ABC \$TCOPY C:*.COM, 然后执行 ML 宏命令,则计算机依次执行建子目录、转子目录、拷贝文件等 DOS 命令。

并且宏命令中也可象批处理中那样使用替换参数,但要注意的是这些替换参数不是 %1~%9,而是 \$1~\$9。

例如上述宏命令可用 DOSKEY ML=MD \$1 \$TCD \$1 \$TCOPY C:*.COM 来代替,但执行时须键入 ML ABC。

同时,在定义宏命令时,还有其他一些特殊的字符可用,提供如下:

```
$G      重新定向输出,相当于“>”
$G$G    将输出附加到一个文件的末尾,相当于“>>”
$L      重新定向输入,相当于“<”
$B      将宏命令的输出作为另一个命令的输入,即管道功能,相当于“|”
```

例如: DOSKEY AA = DIR/W *.COM \$GABC, 执行 AA 宏命令后,ABC 文件中存放的即为当前目录下所有的 COM 文件名。

另外,由于宏命令只保存在内存中,因此关机后这些定义好的宏命令也随之消失。而用户想将这些宏命令加以保存的话,就可用“DOSKEY/M>filename”来加以保存。然后用 EDIT 命令或其他办法来编辑这个文件名,在每一行前加一个 DOSKEY,下次开机后运行一下此文件名,即可执行这些宏命令。当然,在建立需保存的宏命令前,最好先用 ALT+F10 删除缓冲区中所有宏命令,这样就可以将这些有保留价值的宏命令单独清楚地保存在文件中了。

值得注意的是 DOS 执行时是按宏命令、COM、EXE、BAT 的顺序执行同名程序的;并且 DOS 执行宏命令时,宏命令前不应有空格,而 DOS 命令是允许的。因此宏命令完全可以和 DOS 命令同名,执行时只要通过是否在命令前加空格,就可将它们区分开。

参考资料:

周建军 罗文虎等,MS-DOS6.0 基础及其操作技术

(本文收稿日期:1995.3.18)

信息系统中库文件的设计原则与方法

广州中山大学管理学院 韦沛文

随着国民经济的高速发展,信息管理计算机化在我国正逐步普及。目前我国大多数的管理软件是用 DBASE Ⅲ、FOXBASE、FOXPRO 等微机数据库管理系统开发的。一般每一个子系统或每一个独立模块都需要设计若干个数据库文件。库文件个数多少?各库文件字段数多少?设什么字段?其类型宽度如何确定等等,即库文件的设置及其库结构的设计,是管理软件开发中很重要的一个环节,本文将就其设计的原则和方法谈谈自己的体会。

一、信息系统中数据库文件的类型

一般来说,信息系统中每一个独立模块(指包含数据输入、数据处理、信息查询和打印输出等基本功能的一个相对独立的功能部分)都要设置若干个数据库文件,以便存贮输入的各种数据和运算处理所得的中间或最后结果。例如会计信息系统中的工资核算模块,一般要设置存贮职工个人姓名、职务、基本工资、应发工资、实发工资……等主要数据的库文件,存贮单位内各部门名称及其代码对照表等基本数据的库文件,存贮计算钞票张数、工资分配结果等计算结果的库文件。相应地我们可以分别把它们称为主数据库文件,基本数据库文件和结果数据库文件。为了简便,把主数据库文件简称为主库。一般来说,一个模块只有一个主库,它在本模块中是最重要、一个库文件。数据处理主要是对主库文件数据的处理,结果数据库文件的许多字段的数据往往都是由主库文件数据处理得到的结果。

主库在模块所有库文件中起着“奠基石”或“纲”那样的关键作用。纲举目张。因此,把主库的结构设计好是库文件设计至关重要的。

二、库文件结构设计的基本原则与方法

考虑库文件结构的设计,主要有下面一些因素要考虑到,才能使设计的库结构比较符合实际的需要:

1、每个库文件应设计什么字段,主要是由要从该库文件打印出的报表应有什么栏目来决定。

直觉上,习惯认为库文件设什么字段是由要输入的凭证上的数据决定的,凭证上有什么数据,库文件就应能存贮什么数据,因而就应有什么字段。这当然有一定道理。但仔细想想,手工编制的凭证、数据表中有很多数据,我们要输入、存贮到计算机的,并非其全部。一般是那些要打印输出或查询的

信息中不能由已存贮数据通过运算处理得到的最原始数据才需要从外部输入计算机,存贮于库文件中。而要查询的数据绝大部分已反映在打印输出的报表中,所以,库文件应设置什么字段,最主要是由要从该库文件的数据打印出的报表需要打印什么栏目来决定的。所有要打印的栏目中,有些是独立的,有些是其他栏目数据计算得到的。那些独立的栏目就是库文件中应设置的字段。例如工资表中有职工编号、姓名、基本工资等栏目,工资主库就应该有这些字段,以便输入、存贮这些数据。

2、设置一些存放中间或最后处理结果的字段,以方便程序设计和提高程序运行的速度。

中间或最后结果,虽然都可以从独立的字段计算得到,从节省存贮空间、减少数据冗余、降低数据出现不一致性几率的观点,是不需要在库文件中设置单独的字段来存贮它们的。但是中间结果和最后结果,往往是许多其他计算、处理要用到的数据,也是查询、打印经常要用到的数据,如果每次用时都要由独立的原始字段计算后才能得到,就大大增加了程序的复杂性和执行的时间,降低了运行效率。所以,对于一些重要的中间和最后结果,是应该在库文件中设置相应字段来存放它们的。例如工资主库中一般应设置应发工资、实发工资等字段,它们是打印、查询经常用到的数据。

3、设置一些方便运算处理的字段。

这些字段不是为存贮原始数据而设,而主要是为方便运算处理而设。例如,电算会计系统中有许多要进行数据汇总的运算,如科目发生额的汇总,部门或单位工资汇总,材料、产品出入库汇总,销售汇总等等。不同的计算机语言汇总时有不同的方法和具体命令。DBASE 等有一条 TOTAL 命令,对汇总很方便。但这条命令只能以某一字段为关键字汇总,而有时我们要汇总的关键字可能没有这样的单独字段,有时可能要以几个字段组合成一个关键字来汇总。我们可以在库文件中设置一个单独的字段,专门存放要汇总的关键字的值,以便能以此字段为关键字用 TOTAL 命令快速汇总。

4、设置一些控制程序处理过程、时间或处理次数的字段,以方便程序控制。

许多处理是过程性的,或有时时间性的。例如,在工资核算模块中,我们一般每月都用同一个工资主

库存放职工的工资数据,因而在每月开始输入当月工资变动数据之前,先要把上月的工资数据(包括上月输入的、计算后得到的结果等)复制备份,然后把库文件整理为符合本月输入数据前的状态。我们把这一整理工作称为本月初始化。工资核算只有进行了月初始化后,才能输入本月的工资数据,只有本月每个职工的数据均已输入修改好并确认正确无误才能进行工资计算,只有计算好工资才能进行工资自动转帐和工资表打印。我们可以用一个内存变量来记忆各步是否已进行并完成了?进行到哪一步了?在上步未完成前就拒绝执行下一步骤,这就是程序控制问题。但是,用一个内存变量不能很方便地记忆和判断每一记录是否都已修改好当月数据了,如果主库中增加一个“标志”字段,每个记录月初始化后;修改好其工资数据后;计算完其应发、实发工资后,都分别把该记录的“标志”字段值置为某一确定值,就可以很好地十分准确地记下各记录的实际状态了。只有全部记录的数据已正确输入修改好才允许程序进行计算工资,这样数据的准确性就有保障了。信息系统中的其它模块都有类似的情况。所以,当有必要时,库文件中就要设置一些控制处理过程、时间、次数等用于程序控制的字段。

5、主库中要设置一些其它库可能用到的字段,以方便库文件之间的联系。

有些模块中,主库不一定要有某个字段,但其它库文件有此字段,而它们的某些字段的数据要由主库的有关字段计算出来的。例如,工资分配库中应该有“应借科目”字段,以便存储工资自动转帐时的应借科目。为了从主库中统计出各部门应借某一科目的工资总数,我们必须给主库增加部门码、应借科目这样的字段,以便对每个职工(即主库中的每个记录)都能方便地知道其所属部门,其应发工资应借的科目。

有些模块中,为了方便两个库文件之间进行联接(物理联接或逻辑联系)、数据传送、修改(如用UPDATE命令更新),这两个库文件都必须设置一个相同名字的关键字段,否则就不能用相应的命令编程进行这些操作了。

总之,主库文件和基本数据库、结果数据库等其它库文件之间往往要考虑增加一些对主库来说本来是不需要的关键字段。这也是一种为方便程序运算处理而设置的字段。

6、检查所设置字段是否能满足查询、数据输入的需要了,如遗漏了某些查询、输入必须的字段,增加之。

因为我们主要由打印输出的报表所需数据来设计库文件应设立什么字段,所以经过上面这些步

骤而设定了一个库文件的大多数字段以后,我们最后应检查一下,所设立的字段是否对打印、查询、输入都足够了。各库文件都设计好后还应检查各库之间方便计算的关键字段是否都已足够了?不够的应补上。有时可能到设计程序流程图时还会发现应增设某个字段才能方便某部分处理程序的设计。

7、各字段类型的设计

各字段的类型可按这样的原则考虑:

存储字母、汉字的字段当然只能是字符型的;存放日期的字段尽量定义为日期型字段,以便进行日期的运算、不同日期格式的转换、显示等。定义为日期型另一个明显的好处是当输入日期时若键入一个不可能存在的日期(如二月三十日之类)时,系统能自动发现输入错误而拒绝接收该数据。要存放金额的字段,也全部设置为数值型,以方便汇总统计。

8、各字段宽度、小数位数的设计

字段宽度设计的原则是:既要放得下该字段可能出现的位数最多的一个值,又要尽可能短,以节省存储空间。因此,字符型字段按该字段最长的一个值设定;但对数值型字段,还要考虑该字段汇总(求合计)时可能会达到的最大值,按汇总时可能的最大值设置宽度,否则将不能用TOTAL命令进行汇总;对金额字段,其小数位一律定为两位。

9、实践证明,数据库文件记录越少,字段数越少,每个字段宽度越小,对库文件操作的速度越快。

因此,在满足上面各条要求的前提下,每个库文件字段数、记录数尽可能最小化。例如对不经常一起运算使用的字段,可分为几个库文件设计,但对经常同时用到的字段,则以放在同一库文件中为好,以免计算时要打开多个库文件,增加编程难度,容易出错,运算时要在多个库中取数,也会减低运算的速度。

10、库文件结构设计底稿中,应对一些字段加上简要的说明。

每个库文件的结构设计之后加上一些适当的说明来说明某些字段的用途、数据来源、编码格式、数据间的计算关系、处理过程等,就使库文件设计成了数据词典、处理说明、代码设计等合在一起了,既一目了然,又减少了分开设计这些内容的巨大工作量。这样设计出来的库文件,是编程时画程序流程图和编写源程序的主要参考。

以上所谈的库文件设计原则和方法,对所有管理软件都是适用的。库文件结构对编程是有重大影响的,库结构设计得好,编程会相对容易,程序的运行效果也会更好。

(本文收稿日期:1995.1.18)

低主频微机使用 WPS 提高速度的技巧

安徽省蚌埠市汽车管理学院机械室(233011) 林治洋

许多机关早期购置的微机其主频较低,用这种微机作字处理,软件运行速度慢。但只要采取一些适当的措施,仍可以提高编辑速度,满足工作的要求。下面以较流行的 WPS 的运行操作方法作说明。

一、巧设窗口

一般在输入文字前已设定标尺(左右界默认值分别为 1 和 73 个 ASC II 字符),这样在输满设定字符数时会自动换行。一般用户,在文章输入完毕,各段难免有错误,需要修改。在增加内容使一行字符数超过设定右界、或减少一行时,会重写光标以下屏幕。有时由于有表格或其它需要,设定的右界超过 80,这时

时用←、→移光标至 80 列以后时,及其它一些操作也会使屏幕各行重写,把屏幕写满后才能执行下一操作。当文章较长错误较多时,用于写屏的时间相当长,有时令人不能容忍。可以采取下列措施:按下 F6 键设置一个水平窗口,并将需要进行修改的文件所在的窗口用 Ctrl-KO 尽量缩小,然后转入此窗口(Ctrl-QN)进行修改。此时,用于写屏时间大大缩短,可使效率提高。

二、巧用标志移光标 我们都有体会,要将光标移到文章中间某位置,用光标移动键↑、↓或 PgUp、PgDn 很费时间,有时打印一份长文件后,发现有几处错误,修改时翻页查找时间较长,可在适当位置事先定义一个块(如在文章的三分之一处定义块首,在三分之二处定义块尾),这样,可把整篇文章分成三大部分,在纸上找出错误或需改正的文字离文章头、尾、块首、块尾哪儿最近,然后用 Ctrl-Home、End、QB、QK 等命令移动光标,最后在小范围内查找,要比直接一页一页翻或一行一行查找

要快的多。也可用文章本身各自然段前的数字或其它特殊字符作为标志,用 F7 来查找(例如用 F7 查找目前光标所在位置前 10 自然段,用 Ctrl-P Ctrl-M 回答,再用 10BN 回答,就可把光标移到要求的位置。参见下文四)。建议在打印校对稿时,当计算机问“以上参数需要改变吗?”,将打印时重排参数置为“不排”,或直接按源代码打印。这样,根据校对稿可找出错误处对应的是屏幕上的第几行,然后用 Ctrl-QL 移动光标,这是最便捷的方法。

三、文章自然段中间建议不要按回车键 在实际操作中,常见到有的用户录入文字时,用按回车键的方法来保证屏幕上各行对齐。在修改错误以后,若各行不齐就无法用段落重排功能,重新排版就需要人工重排。建议只在自然段结束时打入回车键,这样以后重新排版要方便的多,并且当打印时,将参数“打印时重排:”置为“重排”可在各种宽度的纸上按需要打印出满意的文章。

四、巧用替换命令 如果已经习惯在屏幕上一行末尾按回车键,不妨在重新排版之前用替换功能 Ctrl-QA 将硬回车改为软回车(Ctrl-P Ctrl-M 表示硬回车,Ctrl-P Ctrl-J 表示软回车,自然段尾的硬回车后面是下一自然段前的空格,而自然段中各硬回车后面紧跟字符),然后排版。其他字符、汉字或打印控制符都可用替换命令来替换。标点符号也可在半角状态下直接用键盘上的标点输入,在输入完成后用替换操作把全篇的标点符号改为全角符号。

五、输入汉字若中间夹带英文字母,大写字母 可先按下 Shift 键,然后按下字母键;小写字母可先按下 Alt 键,然后按下字母键。标点符号如果想和汉字占同样宽度也可在全角方式下输入键盘上的标点符号(句号为键盘上的“.”;顿号为键盘上的“/”;省略号为键盘上的“\”),也可以用 Alt-F9 找到需要的符号。在多次重复录入相同内容的几行(段)及大块移动文字位置时,尽量用块操作。

六、如果内存允许,可设置虚拟盘,提高运行速度。

七、用 WPS 打印功能时,若用自定义纸张,下一次再打印时,机器询问“以上参数需要改变吗?” 应回答“Y”,否则不给出自定义纸张尺寸过程,打印出错。另外建议多用模拟打印显示来调整打印版面,又快又节约纸张。

以上各点对于主频较高的机器及熟练用户,也许不算什么技巧,但是对于主频较慢的机器及不十分熟练的用户还是很实用的,不妨一试。

(本文收稿日期:1994.12.9)

ICL232 做液晶对比度电源使用一例

山东省计算中心[250014] 刘文浩 李华

在使用单片微机的系统中,一般为使系统小型化而只使用单电源芯片(通常是+5V)且仅有一个电源,但对于RS-232通讯、液晶显示器等,会需要 $\pm 10V$ 左右的电源。本文介绍的就是如何使用一类收发器芯片来达到此目的。

ICL232是为完成TTL电平和RS-232电平的转换而设计的,它只需单+5V电源(类似的有RS-422、RS-485等通讯需要设计),其DIP/SO封装的引脚排列见图1, LCC封装引脚排列见图2。利用此芯片,可满足简单的RS-232通讯的需要,如图3所示可提供4条RS-232电平信号线(2个输入、2个输出)和4条TTL电平信号线(2个输入、2个输出)。

注意此类芯片的V+端和V-端,它们分别提供一个2VCC和-2VCC的电压,此端子有一定的负载能力(在一定范围内正比于电容)。一般情况下,液晶显示器所需对比度电源电流为 μA 级,此类芯片的V+和V-端完全可以胜任。我们就曾在工作中利用此芯片的V-端做MGLS-12032A液晶显示器的对比度电源,取得了令人满意的效果。这样做,即提供了一个RS-232界面,也充分利用了此类芯片的资源,解决了小规模系统中的电源问题。原理图见图4,图中电容可以取 $10\mu F$,电阻为 $10K\Omega$ 左右的可变电阻。

(本文收稿日期:1995.3.29)

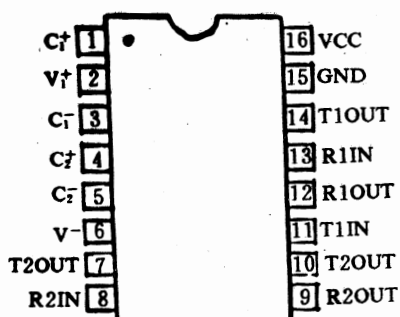


图1 DIP/SO 封装引脚排列图

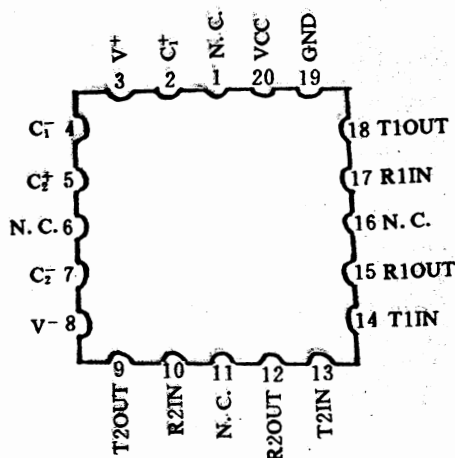


图2 LCC 封装引脚排列图

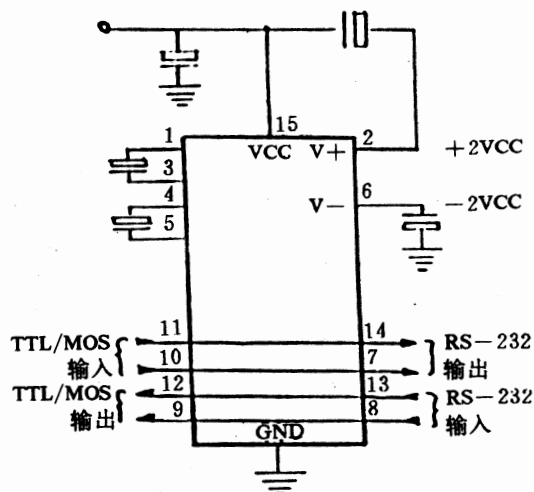


图3 ICL 示例

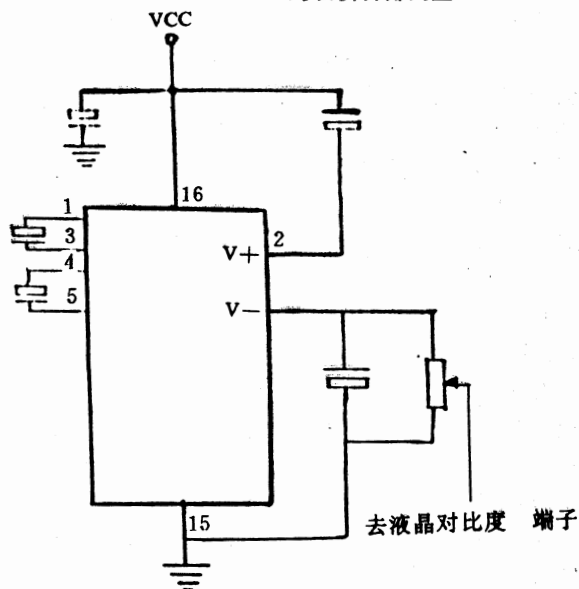


图4 液晶对比度电源一例

也谈键盘功能的自动切换

安徽财贸学院[233041] 车光宏

贵刊 94 年第四期《谈键盘功能的自动切换》一文,介绍了在程序中自动模拟按功能(组合)键的方法。在汉字 DOS 下运行的应用程序中自动模拟按某些功能(组合)键实现各种状态间的转换,无疑会给用户带来很大方便。但是该文仅说明了在 CCDOS2.13 中使用其特殊显示功能来实现的方法,而在目前使用也较为普遍的其它一些汉字 DOS 中一般不具有象 CCDOS2.13 那样的特殊显示功能。显然,该文介绍的方法在这样的 CCDOS 中就无能为力了。那么,在其它 CCDOS 中能否实现功能(组合)键的自动模拟呢?回答是肯定的。对于那些没有占用 BIOS 键盘中断(INT 16H)的 CCDOS(CCDOS2.13 占用了此中断),一般都可用 BIOS 键盘中断(INT 16H)的写键盘功能(功能 5)来完成模拟按功能(组合)键的功能。

使用 INT 16H 的写键盘功能来模拟按功能(组合)键的汇编程序如下:

```
MOV AH,5 ;设置功能号
MOV CH,<功能(组合)键的扫描码>
MOV CL,0 ;若模拟按字符键,则 CL 为字符的 ASCII 码
```

INT 16H

在高级语言(比如 FOXBASE,BASIC 等)中利用 BIOS 键盘中断的写键盘功能来模拟按功能(组合)键以实现状态转换的方法是:编一个有参调用的机器码子程序,该子程序利用 BIOS 键盘中断的写键盘功能模拟按键,高级语言程序在需要转换状态的地方用适当参数调用一下该机器码子程序即可。

例如,程序 1 是按 FOXBASE 调用格式写出的汇编子程序。先用 DEBUG 将其制作成二进制文件 ALT_FK.BIN,以后,若 FOXBASE 程序中需要自动实现中西文输入方式的转换,就在应用程序的开始处用 LOAD ALT_FK 命令将 ALT_FK.BIN 装入,当需要按 ALT+F2 时就发个调用命令

```
CALL ALT_FK WITH '1'
```

当需要按 ALT+F10 时就发个调用命令

```
CALL ALT_FK WITH '9'
```

程序 1:

```
MOV CH,[BX]
ADD CH,38H
XOR CL,CL
MOV AH,05
INT 16H
RETF
```

用 DEBUG 将程序 1 制作成二进制文件 ALT_FK.BIN 的操作步骤如下:

```
C: >DEBUG (进入 DEBUG)
-A (输入汇编程序)
XXXX:0100 MOV CH,[BX]
XXXX:0102 ADD CH,38
XXXX:0105 XOR CL,CL
XXXX:0107 MOV AH,05
XXXX:0109 INT 16
XXXX:010B RETF
XXXX:010C
-R CX (确定文件长度)
CX 0000
:0C (0C 个字节)
-N ALT_FK.BIN (为文件命名)
-W (文件写盘)
Writing 0000C bytes
-Q (退出 DEBUG)
```

至此,建立了名为 ALT_FK.BIN 的二进制文件。

若是在 BASIC 语言中使用这一功能,可将程序 2 译成机器码,在程序开始处将机器码子程序装入内存并取名为 ALT_FK。以后,当需要按 ALT+F2 时就发个调用语句

```
CALL ALT_FK(1)
```

当需要按 ALT+F10 时就发个调用语句

```
CALL ALT_FK(9)
```

由于 BASIC 语言的机器码子程序制作和装入都较为繁琐,所以笔者已将程序 2 对应的机器码译出并写出了将该机器码子程序装入内存的 BASIC 程序(程序 3),只要把程序 3 放到自己程序的开始处,即可用上述的 CALL ALT_FK(1)等语句实现键盘功能的自动切换。

如何在 XENIX 系统下安全关机

广西大化县建行 谭仕谋

一般情况下,操作员只能在超级用户 ROOT 下直接键入 haltsys 或 shutdown 关机,在超级用户下进行操作,很危险,ROOT(根)是系统管理员即特权(超级)用户注册进入系统的名字,与操作系统有关的文件都存放在这里,这是 XENIX 系统最重要的部分,普通用户不应在 ROOT 目录下操作,即使是系统管理员,不在必须对系统进行维护时,也不要以 ROOT 名注册进入系统,因为如果不小心(或者异常关机)根被破坏了,整个系统就有可能无法工作.下面介绍一种不用在 ROOT 下操作便可正常关机的方法:

1. 给系统增加一个用户

```
#mkuser
```

请给出新用户的注册名

键入新用户的注册名:close

你希望用系统指导的用户标识符吗?(y/n/

q):y

你想使用缺省组吗?(y/n/q):y

下面是当前系统中 shell(命令解释器)的配置情况:

```
(1) rsh
```

```
(2) sh
```

```
(3) uucp
```

```
(4) vsh
```

```
(5) csh
```

选择一种 shell: 2(或 4)

键入口令: 直接回车,不用口令

请键入注释: >.....

>——可直接回车

2. 用 vi 修改 cd/usr/close 下的文件 .profile 最后一行,把 "exec/usr/bin/vsh"

修改为"exec/etc/haltsys"

3. 修改 /etc 下 passwd 文件,把 passwd 文件最后一行修改为"close:00:00:./usr/close:./bin/sh"

4. 直接在 "login:" 状态下键入 close 便可正常关机:

```
login:close
```

```
* * Normal System Shutdown * *
```

```
* * Safe to Power off * *
```

```
--or--
```

```
* * Press Any Key to Reboot * *
```

注:已在中西文兼容 cc_xenix v/386 ver 2.3.

2 系统下通过!

(收稿日期:1995.3.15)

(接上页)

程序 2:

```
PUSH BP
```

```
MOV BP,SP
```

```
MOV SI,[BP+06]
```

```
MOV CH,[SI]
```

```
ADD CH,68H
```

```
XOR CL,CL
```

```
MOV AH,05
```

```
INT 16H
```

```
POP BP
```

```
RETF 2
```

程序 3:

```
10 DATA &H55,&H89,&HE5,&H8B,&H76,&H06,
```

```
20 DATA &H8A,&H2C,&H80,&HC5,&H68,&H30,
```

```
30 DATA &HC9,&HB4,&H05,&HCD
```

```
40 DATA &H16,&H5D,&HCA,&H02,&H00
```

```
50 CLEAR ,55000!
```

```
60 FOR J=0 TO 20
```

```
70 READ A%
```

```
80 POKE 55000! +J,A%
```

```
90 NEXT J
```

```
100 ALTFC=55000!
```

注:使用以上方法编写的具有自动切换键盘功能的 FOXBASE 程序和 BASIC 程序均在 Super CCDOS 5.1 和 UC DOS 3.1 环境下的 FOXBASE 和 GWBASIC 中运行通过。

(本文收稿日期:1995.4.1)