

# 80X86 微机原理及接口技术

## 实验教程



西安唐都科教仪器公司

Copyright Reserved 2015

## 版 权 声 明

本实验教程的版权归西安唐都科教仪器开发有限责任公司所有，保留一切权利。未经本公司书面许可，任何单位和个人不得擅自摘抄、复制本实验教程的部分或全部，并以任何形式传播。

西安唐都科教仪器开发有限责任公司，2015(C)，All Right Reserved.

80X86 微机原理及接口技术实验教程

©版权所有 未经许可 严禁复制

唐都公司网址：<http://www.tangdu.com>

技术支持邮箱：[tangdukejiao@126.com](mailto:tangdukejiao@126.com)

技术支持 QQ：[826435224](https://www.qq.com/number/826435224)

# 目 录

目 录.....	I
第 1 章 实模式下的 80X86 机器组织.....	1
1.1 80X86 寄存器.....	1
1.2 80X86 存储器寻址.....	3
1.3 80X86 指令集.....	4
第 2 章 16 位微机原理及其程序设计实验 .....	14
2.1 系统认识实验 .....	14
2.2 数制转换实验 .....	19
2.3 运算类编程实验 .....	28
2.4 分支程序设计实验 .....	34
2.5 循环程序设计实验 .....	37
2.6 排序程序设计实验 .....	41
2.7 子程序设计实验 .....	45
2.8 查表程序设计实验 .....	50
2.9 输入输出程序设计实验.....	53
第 3 章 32 位指令及其程序设计实验 .....	57
3.1 80X86 指令及程序设计 .....	57
3.2 32 位指令及寻址实验 .....	59
第 4 章 80X86 微机接口技术及其应用实验 .....	69
4.1 静态存储器扩展实验 .....	69
4.2 8259 中断控制实验.....	73
4.3 8255 并行接口实验.....	86
4.4 DMA 特性及 8237 应用实验.....	95
4.5 8254 定时/计数器应用实验.....	108
4.6 8251 串行接口应用实验.....	116
4.7 A/D 转换实验 .....	131
4.8 D/A 转换实验 .....	135
4.9 键盘扫描及显示设计实验.....	140
4.10 电子发声设计实验 .....	146
4.11 点阵 LED 显示设计实验 .....	151
4.12 图形 LCD 显示设计实验.....	160
4.13 步进电机实验 .....	169
4.14 直流电机闭环调速实验.....	173
4.15 温度闭环控制实验 .....	186

<b>第 5 章 保护模式下的 80X86 机器组织</b> .....	<b>200</b>
5.1 实模式和保护模式 .....	200
5.2 寄存器组织 .....	200
5.3 保护模式下的分段存储管理机制 .....	202
5.4 任务管理的概念 .....	208
5.5 任务内的控制转移 .....	211
5.6 任务间的控制转移 .....	214
5.7 中断/异常管理 .....	215
5.8 80X86 保护模式程序设计 .....	218
<b>第 6 章 保护模式微机原理及其程序设计实验</b> .....	<b>226</b>
6.1 描述符及描述符表实验 .....	226
6.2 特权级变换实验 .....	242
6.3 任务切换实验 .....	262
6.4 中断与异常处理实验 .....	293
<b>第 7 章 80X86 虚拟存储器的组织及其管理</b> .....	<b>318</b>
7.1 分段管理机制 .....	318
7.2 分页管理机制 .....	325
<b>第 8 章 保护模式下的存储器扩展及其应用实验</b> .....	<b>331</b>
8.1 无分页机制的存储器扩展实验 .....	331
8.2 具有分页机制的存储器扩展实验 .....	336
<b>附录 1 Wmd86 联机软件使用说明</b> .....	<b>344</b>
附 1.1 菜单功能 .....	344
附 1.2 工具栏功能介绍 .....	347
附 1.3 专用图形显示 .....	349
附 1.4 示波器 .....	352
附 1.5 Debug 调试命令 .....	354
<b>附录 2 系统实验程序清单</b> .....	<b>356</b>
<b>附录 3 系统编程信息</b> .....	<b>359</b>
附 3.1 地址分配情况 .....	359
附 3.2 常用 BIOS 及 DOS 功能调用说明 .....	360
<b>附录 4 I386EX 系统板引出管脚排列及名称</b> .....	<b>362</b>

# 第 1 章 实模式下的 80X86 机器组织

微处理器发展是从 8086/8088 开始，经 80286、80386、80486、80586 直到现在的 Pentium 及 Core2 等微处理器。无论哪种微处理器，从 80386 开始都统称为 80X86 系列微机。80X86 支持实模式和保护模式两种运行模式。在实模式下，80X86 相当于一个可以进行 32 位处理的快速 8086/8088，所有为 8086/8088 设计的程序几乎都可适用于 80X86 处理器。

## 1.1 80X86 寄存器

80X86 寄存器的宽度大多是 32 位，可分为如下几组：通用寄存器、段寄存器、指令指针及标志寄存器、系统地址寄存器、调试寄存器和测试寄存器。应用程序主要使用前三组寄存器，只有系统程序才会使用各种寄存器。这些寄存器是 80X86 系统微处理器先前处理器（8086/8088、80186 和 80286）寄存器的超集，所以，80X86 包含了先前微处理器的全部 16 位寄存器。8086/8088 没有系统地址寄存器和控制寄存器等。

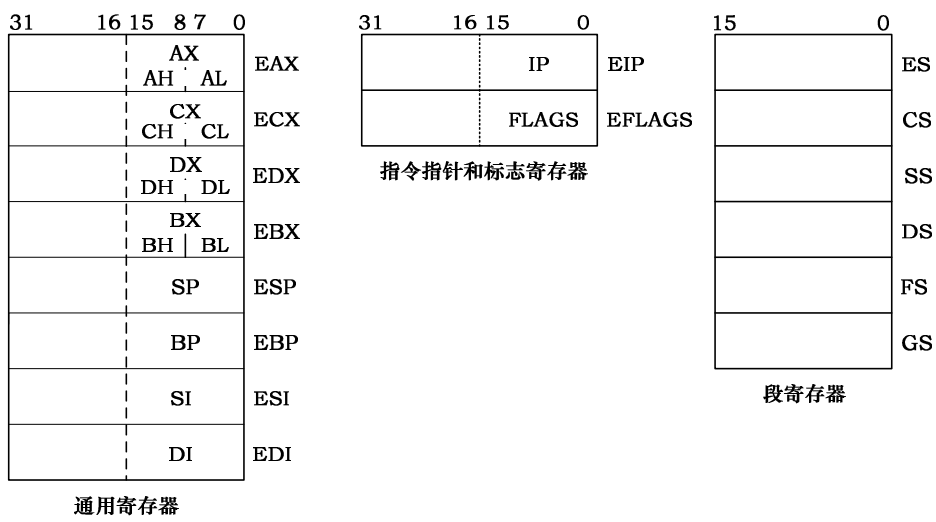


图1.1 80X86的部分寄存器

### 1.1.1 通用寄存器

80X86 有 8 个 32 位通用寄存器，这 8 个寄存器分别命名为 EAX、ECX、EDX、EBX、ESP、EBP、ESI 和 EDI。它们是原先的 16 位通用寄存器的扩展，请参考图 1.1。这些通用

寄存器的低 16 位可以作为 16 位的寄存器独立存取，并把它们命名为 AX、CX、DX、BX、SP、BP、SI 和 DI，它们也就是 X86 系列微处理器先前的 8 个 16 位通用寄存器。

存取这些 16 位的寄存器时，相应的 32 位通用寄存器的高 16 位不受影响。与先前的微处理器一样，AX、BX、CX、DX 这 4 个 16 位的数据寄存器的高 8 位和低 8 位可以被独立存取，分别命名为 AH、AL、BH、BL、CH、CL、DH、DL。在存取这些 8 位寄存器时，相应的 16 位寄存器的其它位不受影响，相应的通用寄存器的其它位也不受影响。

这些 32 位通用寄存器不仅可以传送数据、暂存数据、保存数据，而且还可以在基址和变址寻址时，存放地址。例如：

```
MOV    EAX, 12345678H
MOV    [EBX], EAX
ADD    EAX, [EBX+ESI+1]
MOV    AL, [ECX+EDI+1234]
SUB    CX, [EAX-12]
```

在以前的微处理器中，只有 BX、BP、SI 和 DI 可以在基址和变址寻址时存放地址，而现在 80X86 的 8 个 32 位通用寄存器都可以作为指针寄存器使用，所以说这些 32 位通用寄存器更具有通用性。

### 1.1.2 段寄存器

80X86 有 6 个 16 位段寄存器，分别命名为 CS、SS、DS、ES、FS 和 GS。在实模式下，代码段寄存器 CS、堆栈段寄存器 SS、数据段寄存器 DS 和附加段寄存器 ES 的功能与以前微处理器中对应段寄存器的功能相同。FS 和 GS 是 80X86 新增加的段寄存器。因此，80X86 上运行的程序可同时访问多达 6 个段。

在实模式下，内存单元的逻辑地址仍然是“段值：偏移”形式。为了访问一个给定内存段中的数据，可直接把相应的段值装入某个段寄存器中。例如：

```
MOV    AX, SEG BUFFER
MOV    FS, AX
MOV    AX, FS: [BX]
```

### 1.1.3 指令指针和标志寄存器

80X86 的指令指针和标志寄存器也是以前微处理器的指令指针 IP 和标志寄存器 FLAG 的 32 位扩展。

#### 1. 指令指针寄存器

80X86 的指令指针寄存器扩展到 32 位，记为 EIP。EIP 的低 16 位是 16 位的指令指针 IP，它与以前微处理器中的 IP 相同。IP 寄存器提供了用于执行 8086 和 80286 代码的指令指针。由于实模式下段的最大范围是 64K，所以 EIP 中的高 16 位必须是 0，仍然相当于只有低

16 位的 IP 起作用。

2. 标志寄存器

80X86 的指令寄存器也扩展到 32 位，记为 EFLAGS。与 8086/8088 的 16 位标志寄存器相比，增加了 4 个控制标志，分别为：IO 特权标志 IOPL、嵌套任务标志 NT、重启动标志 RF、虚拟 8086 方式标志 VM。它们分别是：

- (1) IO 特权标志 IOPL (I/O Privilege Level)：位 12、13，按特权级从高到低取值：0，1，2 和 3。只有当前特权级 CPL 在数值上小于或等于 IOPL，I/O 指令才可以执行。
- (2) 嵌套任务标志 NT (Nested Task)：位 14，在保护模式下中断和 CALL 指令可以引起任务切换，任务切换时令 NT=1，否则 NT 清零。在中断返回指令 IRET 执行时，如果 NT=1，则中断返回引起任务切换，否则只产生任务内的控制转移。
- (3) 重启动标志 RF (Restart Flag)：位 16，重启动标志控制是否接受调试故障。
- (4) 虚拟 86 方式标志 VM (Virtual 8086 Mode)：位 17，在保护模式下 VM=1 时，32 位处理器工作在虚拟 86 模式下。

以上 4 个控制标志位在实模式下不起作用，从 80386 开始的 32 位处理器都有。还有 4 个标志位：对齐检查标志 AC (位 18)、虚拟中断允许标志 VIF、虚拟中断挂起标志 VIP、标识标志位 CD，后三个只对 Pentium 有效。32 位标志寄存器的内容如图 1.2 所示。

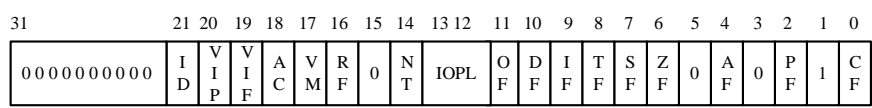


图 1.2 32 位标志寄存器

1.2 80X86 存储器寻址

80X86 支持以前微处理器的各种寻址方式。在立即寻址方式和寄存器寻址方式中，操作数可达 32 位宽。在存储器寻址方式中，不仅操作数可达 32 位，而且寻址范围和方式更加灵活。

1.2.1 存储器寻址

80X86 继续采用分段的方法管理存储器。存储器的逻辑地址由段基地址和段内偏移两部分表示，存储单元的地址由段地址加上段内偏移所得。段寄存器指示段基地址，各种寻址方式决定段内偏移。

在实模式下，段基地址是 16 的倍数，段的最大长度是 64K。段寄存器内所含的是段地址对应的段值，存储单元的物理地址是段寄存器内的段值乘 16 加上段内偏移。所以，80X86 在实模式下与 8086/8088 相似。

DS 寄存器是主要的数据段寄存器，对于访问除堆栈外的数据段它是一个默认的段寄存器。

在以 BP 或 EBP 或 ESP 作为基址寄存器访问堆栈时，默认的段寄存器是 SS。某些字符串操作指令总是使用 ES 段寄存器作为目标操作数的段寄存器。此外 CS、SS、ES、FS 和 GS 也都可以作为访问数据时引用的段寄存器，但必须显式地在指令中指定。

一般的，使 DS 含有最经常访问的数据段的段值，而用 ES、FS 和 GS 含有那些不经常使用的数据段的段值。例如：

```
MOV    EAX, [SI]           ; 默认段寄存器 DS
MOV    [BP+2], EAX         ; 默认段寄存器 SS
MOV    AL, FS: [BX]        ; 显式指定段寄存器 FS
MOV    GS: [BP], DX        ; 显式指定段寄存器 GS
```

## 1.2.2 存储器寻址方式

80X86 支持以前微处理器所支持的各种存储器寻址方式，各种存储器寻址方式表示的都是有效地址。

80X86 不仅支持各种 16 位偏移的存储器寻址方式，而且还支持 32 位偏移的存储器寻址方式。80X86 允许内存地址的偏移可以由三部分内相加构成：一个 32 位基址寄存器，一个可乘上比例因子 1、2、4 或 8 的 32 位变址寄存器，及一个 8 位或 32 位的常数偏移量。如果含变址寄存器，那么变址寄存器中的值先按给定的比例因子放大，再加上偏移。

在所有寻址方式中，对数据的访问所默认引用的段寄存器取决于所选择的基址寄存器。如果基址寄存器是 ESP 或者 EBP，那么默认的段寄存器从通常的 DS 改为 SS。对于别的基址寄存器的选择，包括没有基址寄存器的情况，DS 仍然是默认的段寄存器。

## 1.2.3 支持各种数据结构

80X86 支持的“基址+变址+位移量”寻址方式能进一步满足各高级语言支持的数据结构的需要。标量变量、记录、数组、记录的数组和数组的记录等数据结构可方便地利用 80X86 的这种寻址方式实现。

# 1.3 80X86 指令集

80X86 的指令集包含了 8086/8088、80186 和 80286 指令集。可分为如下：数据传送指令、算术运算指令、逻辑运算和移位指令、控制转移指令、串操作指令、高级语言支持指令、条件字节设置指令、位操作指令、处理器控制指令和保护方式指令。

80X86 是 32 位处理器，其指令的操作数长度可以是 8 位、16 位或者是 32 位。对于 80X86 而言，32 位操作数是对 16 位操作数的扩展。80X86 既支持 16 位存储器操作数地址，又支持 32 位的存储器操作数有效地址的扩展。所以，80X86 支持的 32 位操作数的指令往往就是对



相应支持 16 位操作数指令的扩展；80X86 的 32 位存储器操作数有效地址方式往往就是对 16 位存储器操作数有效地址寻址方式的扩展。

### 1.3.1 数据传送指令

数据传送指令实现在寄存器、内存单元或 I/O 端口之间传送数据和地址。80X86 的数据传送指令仍分成四种：通用数据传送指令、累加器专用传送指令、地址传送指令和标志传送指令。

#### 1. 通用传送指令组

80X86 的通用传送指令组含有如下十条指令：数值传送指令 MOV、符号扩展指令 MOVSX、零扩展指令 MOVZX、交换指令 XCHG、进栈指令 PUSH、PUSHA、PUSHAD、退栈指令 POP、POPA、POPAD。

##### (1) 数值传送指令 MOV

MOV 指令与 8086/8088 的 MOV 指令相同，可传送 8 位、16 位或 32 位数据。

##### (2) 符号扩展指令 MOVSX 和零扩展指令 MOVZX

符号扩展指令的格式如下：

MOVSX     DST, SRC

该指令功能是把源操作数 SRC 的内容送到目的操作数 DST，目的操作数空出的位用源操作数的符号位填补。

零扩展指令的格式如下：

MOVZX     DST, SRC

该指令功能是把源操作数 SRC 的内容送到目的操作数 DST，目的操作数空出的位用零填补。

符号扩展指令和零扩展指令中的目的操作数 DST 必须是 16 位或 32 位寄存器，源操作数 SRC 可以是 8 位或 16 位寄存器，也可以是 8 位或 16 位存储器操作数。如果源操作数和目的操作数都是字，那么就相当于 MOV 指令。

这两条指令各不影响标志。

##### (3) 交换指令 XCHG

XCHG 指令与 8086/8088 的 XCHG 指令相同，可传送 8 位、16 位或 32 位数据。

##### (4) 进栈指令 PUSH

进栈指令 PUSH 与 8086/8088 格式一样，但功能增强了，压入堆栈的操作数还可以是立即数。从 80X86 开始，操作数长度还可以达 32 位，那么堆栈指针减 4。

##### (5) 出栈指令 POP

POP 指令与 8086/8088 的 POP 指令相同，可弹出 32 位操作。

##### (6) 16 位全进栈指令 PUSHA 和全出栈指令 POPA

PUSHA 指令和 POPA 指令提供了压入或弹出 8 个 16 位通用寄存器的有效手段，它们的格式如下：

PUSHA

## POPA

PUSHA 指令将所有 8 个通用寄存器（16 位）内容压入堆栈，其顺序是：AX、CX、DX、BX、SP、BP、SI、DI，然后堆栈指针寄存器 SP 的值减 16，所以 SP 进栈的内容是 PUSHA 执行之前的值。

POPA 指令从堆栈弹出内容以 PUSHA 相反的顺序送到这些通用寄存器，从而恢复 PUSHA 之前的寄存器内容。但堆栈指针寄存器 ESP 的值不是由堆栈弹出，而是通过增加 16 来恢复。这两条指令各不影响标志。

### (7) 32 位全进栈指令 PUSHAD 和全出栈指令 POPAD

PUSHAD 指令和 POPAD 指令提供了压入或弹出 8 个 32 位通用寄存器的有效手段，它们的格式如下：

PUSHAD

POPAD

PUSHAD 指令将所有 8 个通用寄存器（32 位）内容压入堆栈，其顺序是：EAX、ECX、EDX、EBX、ESP、EBP、ESI、EDI，然后堆栈指针寄存器 ESP 的值减 32，所以 ESP 进栈的内容是 PUSHAD 执行之前的值。

POPAD 指令从堆栈弹出内容以 PUSHAD 相反的顺序送到这些通用寄存器，从而恢复 PUSHAD 之前的寄存器内容。但堆栈指针寄存器 SP 的值不是由堆栈弹出，而是通过增加 32 来恢复。

这两条指令各不影响标志。

## 2. 地址传送指令组

### (1) 装入有效地址指令 LEA

装入有效地址指令的格式和功能同 8086/8088。源操作数仍然必须是存储器操作数，且的操作数是 16 位或者 32 位通用寄存器。当目的操作数是 16 位通用寄存器时，那么只装入有效地址的低 16 位。

### (2) 装入指针指令组

装入指针指令组有 5 条指令，格式如下：

LDS REG, OPRD

LES REG, OPRD

LFS REG, OPRD

LGS REG, OPRD

LSS REG, OPRD

这 5 条指令的功能是将操作数 OPRD 所指内存单元的 4 个或 6 个相继字节单元的内容送到指令助记符给定的段寄存器和目的操作数 REG 中。目的操作数必须是 16 位或 32 位通用寄存器，源操作数是存储器操作数。

如果目的操作数是 16 位通用寄存器，那么源操作数 OPRD 含 32 位指针。如果目的操作数是 32 位通用寄存器，那么源操作数 OPRD 含 48 位指针。如：

LSS SP, SPVAR ; SPVAR 是含有堆栈指针的双字

这些指令各不影响标志。

## 3. 标志传送指令组

80X86 的标志传送指令组含有以下 6 条指令: LAHF、SAHF、PUSHF、PUSHFD、POPF 和 POPFD。

指令 LAHF、SAHF、PUSHF 和 POPF 指令格式和功能与 8086/8088 相同。

32 位标志寄存器进栈和出栈指令的格式如下:

PUSHFD

POPFD

PUSHFD 指令将整个标志寄存器的内容压入堆栈; POPFD 指令将栈顶的一个双字弹出到 32 位的标志寄存器中。这两条指令是 PUSHF 和 POPF 指令的扩展。

PUSHFD 指令不影响各标志, POPFD 指令影响各标志。

#### 4. 累加器专用传送指令组

80X86 累加器专用传送指令组含有如下指令: IN、OUT 和 XLAT。

输入指令 IN、OUT 与 8086/8088 相同, 但可以通过累加器 EAX 输入、输出一个双字。

如:

IN EAX, 20H ; 从 20H 端口输入一个双字

OUT 20H, EAX ; 输出一个双字到 20H 端口

表转换指令 XLAT 的格式和功能与 8086/8088 相同。但是从 80X86 开始存放基值的寄存器可以是 EBX。也就是说, 扩展的 XLAT 指令以 EBX 为存放基值的寄存器, 非扩展的 XLAT 指令以 BX 为存放基值的寄存器。

### 1.3.2 算术运算指令

80X86 算术运算指令的操作数可以扩展到 32 位, 同时与 8086/8088 相比还增强了有符号数乘法指令的功能。

#### 1. 加法和减法指令组

加法和减法指令组的功能与 8086/8088 相同, 有 8 条指令: **ADD**、**ADC**、**INC**、**SUB**、**SBB**、**DEC**、**CMP** 和 **NEG**。但在 80X86 下指令的操作数可以扩展到 32 位, 如:

ADD EAX, ESI

ADC EAX, DWORD PTR [BX]

INC EBX

SUB ESI, 4

SBB DWORD PTR [EDI], DX

DEC EDI

CMP EAX, EDX

NEG ECX

#### 2. 乘法和除法指令组

乘法和除法指令组含有 4 条指令: **MUL**、**DIV**、**IMUL** 和 **IDIV**。

(1) 无符号数乘法和除法指令

无符号数乘法 MUL 指令和除法指令 DIV 指令的格式没有变。指令中只给出一个操作数,

自动根据给出的操作数确定另一个操作数。当指令中给出的源操作数为字节或字时，它们与 8086/8088 相同。

在源操作数为双字的情况下，乘法指令 MUL 默认的另一个操作数是 EAX，其功能是把 EAX 内容乘上源操作数内容所得积送入 EDX: EAX 中，若结果的高 32 位为 0，那么标志 CF 和 OF 被清 0，否则被置 1；除法指令 DIV 默认的被除数是 EDX: EAX，其功能是把指令中给出的操作数作为除数，所得的商送 EAX，余数送 EDX。

### (2) 有符号数乘法和除法指令

原有的有符号数乘法指令 IMUL 和除法指令 IDIV 继续保持，但操作数可以扩展到 32 位。当操作数为 32 位时，它与无符号数乘法指令相同。

另外，80X86 还提供了新形式的有符号数乘法指令。如：

IMUL DST, SRC

IMUL DST, SRC1, SRC2

上述第一种格式是将目的操作数 DST 与源操作数 SRC 相乘，结果送到目的操作数 DST 中；第二种格式是将 SRC1 和 SRC2 相乘，结果送到目的操作数 DST 中。

### 3. 符号扩展指令组

80X86 的符号扩展指令有 4 条：CBW、CWD、CWDE 和 CDQ。

其中 CBW 和 CWD 的功能没有发生变化；指令 CWDE 和 CDQ 是 80X86 新增的指令，它们的格式如下：

CWDE

CDQ

指令 CWDE 将 16 位寄存器 AX 的符号位扩展到 32 位寄存器 EAX 的高 16 位中。该指令是指令 CBW 的扩展。

指令 CDQ 将寄存器 EAX 的符号位扩展到 EDX 的所有位。该指令是指令 CWD 的扩展。这些指令均不影响各标志。

### 4. 十进制调整指令组

十进制调整指令 DAA、DAS、AAA、AAS、AAM 和 AAD，这 6 条指令的功能与 8086/8088 相同。

## 1.3.3 逻辑运算和移位指令

80X86 的逻辑运算和移位指令包括逻辑运算指令、一般移位指令、循环移位指令和双精度移位指令。

### 1. 逻辑运算指令组

逻辑运算指令 NOT、AND、OR、XOR 和 TEST 这 5 条指令，除了其操作数可以扩展到 32 位外，其它功能与 8086/8088 相同。

### 2. 一般移位指令组

一般移位指令组含有 3 条指令：SAL/SHL、SAR 和 SHR。算术左移指令 SAL 和逻辑左移指令 SHL 是相同的。

从 80X86 开始, 操作数可扩展到 32 位。尽管这些指令的格式没有变化, 但移位位数的表达增强了。实际移位位数的变化范围是 0 至 31。

### 3. 循环移位指令组

循环移位指令组有 4 条指令: **ROL**、**ROR**、**RCL** 和 **RCR**。

从 80X86 开始, 对循环指令 **ROL** 和 **ROR** 而言, 实际移位的位数将根据被移位的操作数的长度取 8、16 或 32 位的模; 对带进位循环移位指令 **RCL** 和 **RCR** 而言, 移位位数先取指令中规定的移位位数的低 5 位, 再根据被移位的操作数的长度取 9、17 或 32 位的模。

### 4. 双精度移位指令组

双精度移位指令 **SHLD** 和 **SHRD** 从 80X86 开始才有, 其格式如下:

**SHLD** OPRD1, OPRD2, m

**SHRD** OPRD1, OPRD2, m

其中, OPRD1 可以是 16 位通用寄存器、16 位存储单元、32 位通用寄存器或者 32 位存储单元; 操作数 OPRD2 的长度必须与操作数 OPRD1 和长度一致, 并且只能是 16 位通用寄存器或者是 32 位通用寄存器; m 是移位位数, 或者是 8 位立即数, 或者是 CL。

双精度左移指令 **SHLD** 的功能是把操作数 OPRD1 左移指定的 m 位, 空出的位用操作数 OPRD2 高端的 m 位填补, 但操作数 OPRD2 的内容不变, 最后移出的位保留在进位标志 CF 中。如果只移 1 位, 当进位标志和最后的符号位不一致是, 置溢出标志 OF, 否则清 OF。

双精度右移指令 **SHRD** 的功能是把操作数 OPRD1 右移指定的 m 位, 空出的位用操作数 OPRD2 低端的 m 位填补, 但操作数 OPRD2 的内容不变, 最后移出的位保留在进位标志 CF 中。当移位位数是 1 时, OF 标志受影响, 否则清 OF。

## 1.3.4 控制转移指令

控制转移指令可分为以下 4 组: 转移指令、循环指令、过程调用和返回指令、中断调用指令和中断返回指令。

### 1. 转移指令组

#### (1) 无条件转移指令

无条件转移指令 **JMP** 在分为段内直接、段内间接、段间直接和段间间接四类的同时, 还具有扩展形式, 扩展的无条件转移指令的转移目的地址偏移采用 32 位表示, 段间转移目的地址采用 48 位全指针形式表示。

在实模式下, 无条件转移指令 **JMP** 的功能几乎没有提高。尽管 80X86 的无条件转移指令允许把 32 位的段内偏移送到 EIP, 但在实模式下段最大 64K, 段内偏移不能超过 64K, 所以不需要使用 32 位的段内偏移。

#### (2) 条件转移指令

80X86 的条件转移指令 (除 **JCXZ** 和 **JECXZ** 指令处) 允许用多字节来表示转移目的地偏移与当前偏移之间的差, 所以转移范围可起出  $-128 \sim +127$ 。

在 80X86 中, 当寄存器 CX 的值为 0 时, 转移的指令 **JCXZ** 可以被扩展到 **JECXZ**, 如:

**JECXZ**      **OK**

它表示当 32 位寄存器 ECX 为 0 时，转移到标号 OK 处。

## 2. 循环指令组

循环指令组含有 3 条指令：LOOP、LOOPZ/LOOPE 和 LOOPNZ/LOOPNE。这三条循环指令的非扩展形式保持原功能。它们的扩展形式使用 ECX 作为计数器，即从 CX 扩展到 ECX。

## 3. 过程调用和返回指令组

过程调用指令 CALL 在分为段内直接、段内间接、段间直接和段间间接四种的同时，还具有扩展形式。扩展的调用指令的转移目的地址偏移采用 32 位表示。对于扩展的段间调用指令，转移目的地址采用 48 位全指针形式表示，而且在把返回地址的 CS 压入堆栈时扩展成高 16 位为 0 的双字，这样会压入堆栈 2 个双字。

过程返回指令 RET 在分为段内返回和段间返回的同时，还分别具有扩展形式。扩展的过程返回指令要从堆栈弹出双字作为返回地址的偏移。如果是扩展的段间返回指令，执行时要从堆栈弹出包含 48 位返回地址全指针的 2 个双字。

在实模式下，段内过程调用指令和返回指令 RET 的非扩展形式，它们与 8086/8088 的 CALL 和 RET 相同。

## 4. 中断调用和中断返回指令组

在实模式下，中断调用指令 INT 和中断返回指令 IRET 的功能与 8086/8088 的相同。

# 1.3.5 串操作指令

从 80X86 开始，串操作的基本单位在字节和字的基础上增加了双字。

## 1. 基本串操作指令

对应于字节和字为元素的基本串操作指令没有变化。对应于双字为元素的基本串操作指令格式为：

LODSD	；串装入指令
STOSD	；串存储指令
MOVSD	；串传送指令
SCANS	；串扫描指令
CMPSD	；串比较指令

其中，LODSD、STOSD 和 SCANS 指令使用累加器 EAX；在 DF=0 时，每次执行串操作后相应指针加 4，在 DF=1 时，每次串操作后相应指针减 4。

这些以双字为元素的基本串操作指令的功能和使用方法与以字节或字为元素的基本串操作指令一样。它们分别是对应以字为元素的串操作指令的扩展。

## 2. 重复前缀

重复前缀 REP、REPZ/REPE 和 REPNZ/REPNE，在仍采用 16 位地址偏移指针的情况下以 CX 作为重复计数器，在采用 32 位地址偏移的扩展情况下以 ECX 作为重复计数器。由于实模式下通常采用 16 位指针，所以一般仍以 CX 作为计数器。

## 3. 串输入指令

串输入指令的格式如下：



INSB                ; 输入字节 BYTE  
 INSW              ; 输入字 WORD  
 INSD              ; 输入双字 DWORD

串输入指令从由 DX 给出端口地址的端口读入一字符，并送入由 ES: DI (或 EDI) 所指的串中，同时根据方向标志 DF 和字符类型调整 DI (或 EDI)。在汇编语言中，三条串输入指令的格式可统一如下一种格式：

INS     DSTS, DX

#### 4. 串输出指令

串输出指令的格式如下：

OUTSB            ; 输出字节 BYTE  
 OUTSW           ; 输出字 WORD  
 OUTSD           ; 输出双字 DWORD

串输出指令是把 DS: SI (或 ESI) 所指的源串中的一个字符，输出到由 DX 给出的端口，同时，根据方向标志 DF 和字符类型调整 SI (或 ESI)。在汇编语言中，三条串输入指令的格式可统一如下一种格式：

OUTS     DX, SRCS

### 1.3.6 条件字节设置指令

从 80X86 开始新增加了一组条件字节设置指令。这些指令根据一些标志位设置某个字的内容为 1 或 0。

条件字节设置指令的一般格式为：

SET\*\*        OPRD

共有以下 30 个指令：

SETZ	SETE	SETNZ	SETNE	SETS	SETNS
SETO	SETNO	SETP	SETPE	SETNP	SETPO
SETB	SETNAE	SETC	SETNB	SETAE	SETNC
SETBE	SETNA	SETNBE	SETA	SETL	SETNGE
SETNL	SETGE	SETLE	SETNG	SETNLE	SETG

### 1.3.7 位操作指令

从 80X86 开始增加了位操作指令。这些位操作指令可以直接对一个二进制位进行测试、设置和扫描等操作。利用这些指令可以更有效地进行位操作。

位操作指令可分为位扫描指令和位测试及设置指令组。

#### 1. 位扫描指令组

位扫描指令组含有 2 条指令：顺向位扫描 BSF 指令和逆向位扫描 BSR 指令。其格式如下：

BSF            OPRD1, OPRD2

BSR            OPRD1, OPRD2

其中操作数 OPRD1 和 OPRD2 可以是 16 位或 32 位通用寄存器和 16 位或 32 位存储器单元；但操作数 OPRD1 和 OPRD2 的位数长度必须相等。

顺向位扫描 BSF 指令的功能是从右向左扫描字或者双字操作数 OPRD2 中第一个含“1”的位的位号送到操作数 OPRD1。

逆向位扫描 BSR 指令的功能是从左向右扫描字或者双字操作数 OPRD2 中第一个含“1”的位的位号送到操作数 OPRD1。

如果字或双字操作数 OPRD2=0，那么零标志 ZF 被置 1，操作数 OPRD1 的值不确定；否则零标志 ZF 被清 0。

## 2. 位测试及设置指令组

位测试及设置指令含有 4 条指令，其格式如下：

BT            OPRD1, OPRD2

BTC           OPRD1, OPRD2

BTR           OPRD1, OPRD2

BTS           OPRD1, OPRD2

其中操作数 OPRD1 可以是 16 位或 32 位通用寄存器和 16 位或 32 位存储单元，用于指定要测试的内容；操作数 OPRD2 必须是 8 位立即数或者操作数 OPRD1 长度相等的通用寄存器，用于指定要测试的位。

## 1.3.8 处理器控制指令

处理器控制指令用于设置标志、空操作和与外部事件同步等。

### 1. 设置标志指令组

设置进位标志 CF 的指令 CLC、STC 和 CMC 保持原先相同。

设置方向标志 DF 的指令 CLD 和 STD 保持原先相同。

设置中断允许标志 IF 的指令 CLI 和 STI 的功能在实模式下保持与原先相同。在保持模式下它们是 I/O 敏感指令。

### 2. 空操作指令组

空操作指令 NOP 的一般格式如下：

NOP

空操作指令的功能是什么都不干，该指令就一个字节的操作码。

### 3. 外同步指令和前缀

#### (1) 等待指令 WAIT

等待指令 WAIT 的一般格式如下：

WAIT

该指令的功能是等待直到 BUSY 引脚为高。BUSY 由数值协处理器控制，所以该指令的功能是等待数值协处理器，以便与它同步。



## (2) 封锁前缀 LOCK

封锁前缀 LOCK 可以锁定其后指令的目的操作数确定的存储单元，这是通过使 LOCK 信号在指令执行期间一直保持有效而实现的。

## 第 2 章 16 位微机原理及其程序设计实验

本章主要介绍汇编语言程序设计，通过实验来学习 80X86 的指令系统、寻址方式以及程序的设计方法，同时掌握联机软件的使用。

### 2.1 系统认识实验

#### 2.1.1 实验目的

掌握 TD 系列微机原理及接口技术教学实验系统的操作，熟悉 Wmd86 联机集成开发调试软件的操作环境。

#### 2.1.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

#### 2.1.3 实验内容

编写实验程序，将 00H~0FH 共 16 个数写入内存 3000H 开始的连续 16 个存储单元中。

#### 2.1.4 实验步骤

1. 运行 Wmd86 软件，进入 Wmd86 集成开发环境。
2. 根据程序设计使用语言的不同，通过在“设置”下拉列表来选择需要使用的语言和寄存器类型，这里我们设置成“汇编语言”和“16 位寄存器”，如图 2.1、图 2.2 所示。设置选择后，下次再启动软件，语言环境保持这次的修改不变。本章选择 16 位寄存器。



图 2.1 语言环境设置界面

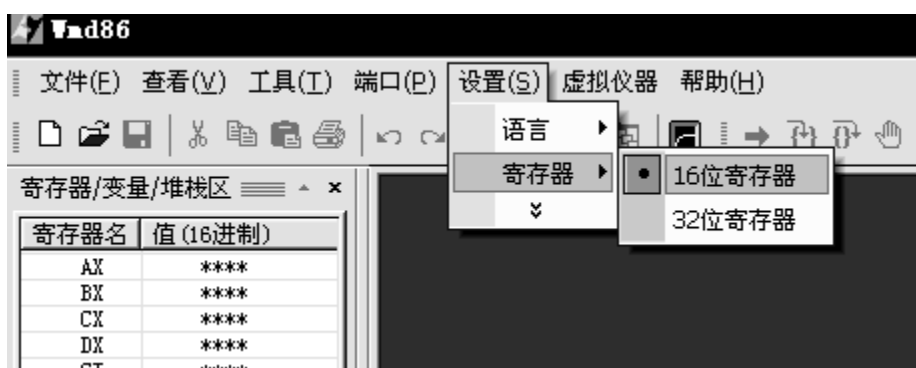


图 2.2 寄存器设置界面

3. 语言和寄存器选择后，点击新建或按 Ctrl+N 组合键来新建一个文档，如图 2.3 所示。默认文件名为 Wmd861。

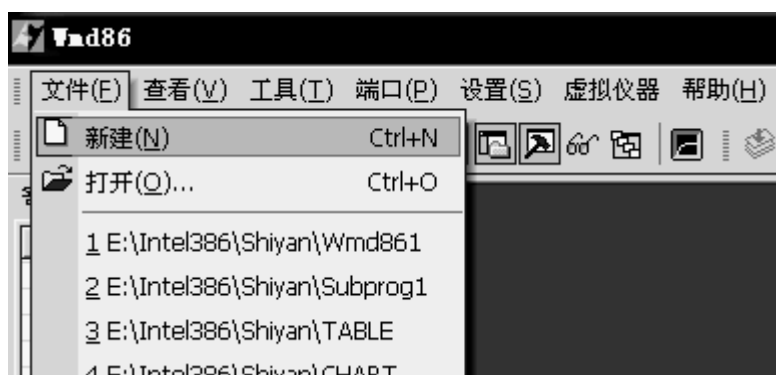


图 2.3 新建文件界面

4. 编写实验程序，如图 2.4 所示，并保存，此时系统会提示输入新的文件名，输完后点击保存。

```

SSTACK  SEGMENT STACK                ;定义堆栈段
        DW 32 DUP(?)
SSTACK  ENDS

CODE     SEGMENT
        ASSUME CS:CODE, SS:SSTACK
START:   PUSH DS
        XOR AX, AX
        MOV DS, AX
        MOV SI, 3000H                ;建立数据起始地址
        MOV CX, 16                   ;循环次数
AA1:     MOV [SI], AL
        INC SI                        ;地址自加1
        INC AL                        ;数据自加1
        LOOP AA1
        MOV AX, 4C00H
        INT 21H                       ;程序终止
CODE     ENDS
        END START

```

图 2.4 程序编辑界面





5. 点击 , 编译文件, 若程序编译无误, 则可以继续点击  进行链接, 链接无误后方可加载程序。编译、链接后输出如图 2.5 所示的输出信息。



图 2.5 编译输出信息界面

6. 连接 PC 与实验系统的通讯电缆, 打开实验系统电源。

7. 编译、链接都正确并且上下位机通讯成功后, 就可以下载程序, 联机调试了。可以通过端口列表中的“端口测试”来检查通讯是否正常。点击  下载程序。  为编译、链接、下载

组合按钮，通过该按钮可以将编译、链接、下载一次完成。下载成功后，在输出区的结果窗中会显示“加载成功！”，表示程序已正确下载。起始运行语句下会有一条绿色的背景。如图 2.6 所示。




图 2.6 加载成功显示界面

8. 将输出区切换到调试窗口，使用 D0000:3000 命令查看内存 3000H 起始地址的数据，如图 2.7 所示。存储器在初始状态时，默认数据为 CC。



图 2.7 内存地址单元数据显示

9. 点击按钮  运行程序，待程序运行停止后，通过 D0000:3000 命令来观察程序运行结果。如图 2.8 所示。

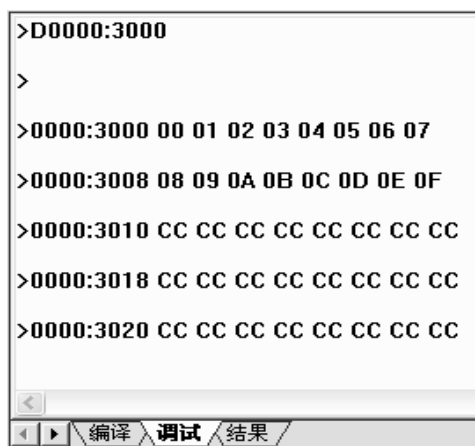


图 2.8 运行程序后数据变化显示

10. 也可以通过设置断点，断点显示如图 2.9 所示，然后运行程序，当遇到断点时程序会停下来，然后观察数据。可以使用 E0000:3000 来改变该地址单元的数据，如图 2.10 所示，输入 11 后，按“空格”键，可以接着输入第二个数，如 22，结束输入按“回车”键。

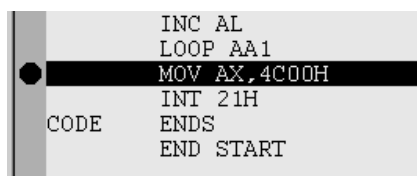


图 2.9 断点设置显示

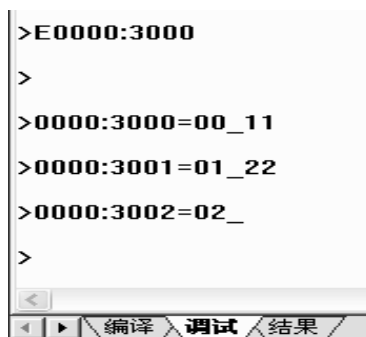


图 2.10 修改内存单元数据显示界面

实验例程文件名为 Wmd861.asm。

## 2.1.5 操作练习

编写程序，将内存 3500H 单元开始的 8 个数据复制到 3600H 单元开始的数据区中。通过调试验证程序功能，使用 E 命令修改 3500H 单元开始的数据，运行程序后使用 D 命令查看 3600H 单元开始的数据。

## 2.2 数制转换实验

### 2.2.1 实验目的

1. 掌握不同进制数及编码相互转换的程序设计方法，加深对数制转换的理解。
2. 熟悉程序调试的方法。

### 2.2.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 2.2.3 实验内容及步骤

计算机输入设备输入的信息一般是由 ASCII 码或 BCD 码表示的数据或字符，CPU 一般均用二进制数进行计算或其它信息处理，处理结果的输出又必须依照外设的要求变为 ASCII 码、BCD 码或七段显示码等。因此，在应用软件中，各类数制的转换是必不可少的。计算机与外设间的数制转换关系如图 2.11 所示，数制对应关系如表 2.1 所示。

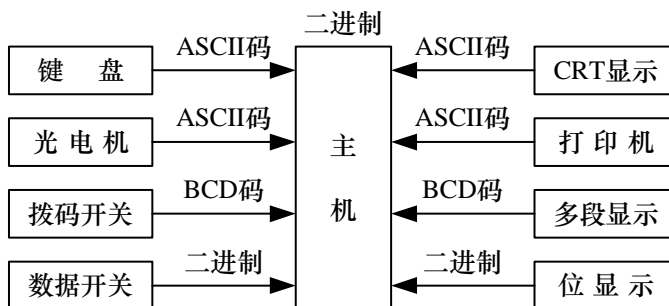


图 2.11 数制转换关系

1. 将 ASCII 码表示的十进制数转换为二进制数  
十进制表示为：

$$D_n \times 10^n + D_{n-1} \times 10^{n-1} + \dots + D_0 \times 10^0 = \sum_{i=0}^n D_i \times 10^i \quad (1)$$

$D_i$  代表十进制数 0, 1, 2, ..., 9;

上式转换为：

$$\sum_{i=0}^n D_i \times 10^i = (\cdots((D_n \times 10 + D_{n-1}) \times 10 + D_{n-2}) \times 10 + \cdots + D_1) \times 10 + D_0 \quad (2)$$

由式 (2) 可归纳十进制数转换为二进制数的方法：从十进制数的最高位  $D_n$  开始作乘 10 加次位的操作，依次类推，则可求出二进制数的结果。

表 2.1 数制对应关系表

十六进制	BCD 码	二进制 机器码	ASCII 码	七段码	
				共阳	共阴
0	0000	0000	30H	40H	3FH
1	0001	0001	31H	79H	06H
2	0010	0010	32H	24H	5BH
3	0011	0011	33H	30H	4FH
4	0100	0100	34H	19H	66H
5	0101	0101	35H	12H	6DH
6	0110	0110	36H	02H	7DH
7	0111	0111	37H	78H	07H
8	1000	1000	38H	00H	7FH
9	1001	1001	39H	18H	67H
A		1010	41H	08H	77H
B		1011	42H	03H	7CH
C		1100	43H	46H	39H
D		1101	44H	21H	5EH
E		1110	45H	06H	79H
F		1111	46H	0EH	71H

程序流程图如图 2.12 所示，实验程序参考例程。



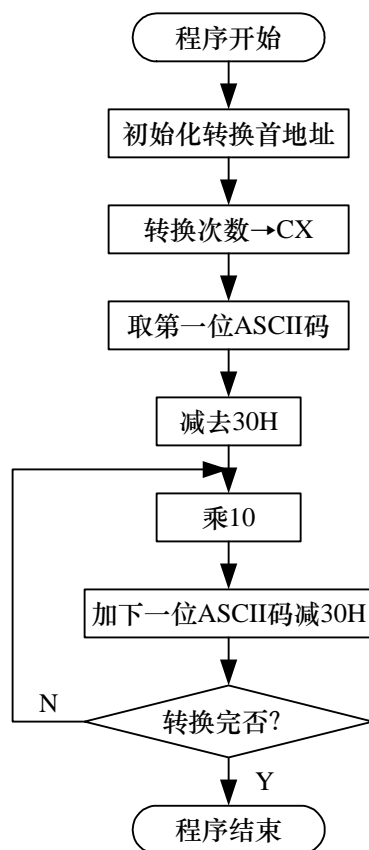


图 2.12 转换程序流程图

### 实验步骤

(1) 绘制程序流程图，编写实验程序（例程文件名：A2-1.ASM），经编译、链接无误后装入系统。

(2) 待转换数据存放于数据段，根据自己要求输入，默认为 30H，30H，32H，35H，36H。

(3) 运行程序，然后停止程序。

(4) 查看 AX 寄存器，即为转换结果，应为：0100。

(5) 反复试几组数据，验证程序的正确性。

### 实验程序清单

```

;=====
; 文件名:   A2-1.ASM
; 功能描述: 将 ASCII 码表示的十进制转换为二进制
;=====
PUBLIC  SADD
SSTACK  SEGMENT STACK

```

```

        DW 64 DUP(?)
SSTACK  ENDS

DATA    SEGMENT
SADD    DB  30H,30H,32H,35H,36H          ;十进制数:00256
DATA    ENDS

CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA

START:  MOV AX, DATA
        MOV DS, AX
        MOV AX, OFFSET SADD
        MOV SI, AX
        MOV BX, 000AH
        MOV CX, 0004H
        MOV AH, 00H
        MOV AL, [SI]
        SUB AL, 30H
A1:     IMUL BX
        MOV DX, [SI+01]
        AND DX, 00FFH
        ADC AX, DX
        SBB AX, 30H
        INC SI
        LOOP A1
A2:     JMP A2
CODE    ENDS
        END START

```

## 2. 将十进制数的 ASCII 码转换为 BCD 码

从键盘输入五位十进制数的 ASCII 码，存放于 3500H 起始的内存单元中，将其转换为 BCD 码后，再按位分别存入 350AH 起始的内存单元内。若输入的不是十进制的 ASCII 码，则对应存放结果的单元内容为“FF”。由表 2.1 可知，一字节 ASCII 码取其低四位即变为 BCD 码。

### 实验步骤

(1) 自己绘制程序流程图，然后编写程序（例程文件名：A2-2.ASM），编译、链接无误后装入系统。

(2) 在 3500H~3504H 单元中存放五位十进制数的 ASCII 码，即：键入 E3500 后，

输入 31, 32, 33, 34, 35。

(3) 运行程序, 待程序运行停止。

(4) 在调试窗口键入 D350A, 显示运行结果, 应为:

0000:350A 01 02 03 04 05 CC ...

(5) 反复测试几组数据, 验证程序功能。

#### 实验程序清单

```
=====
; 文件名:   A2-2.ASM
; 功能描述: 将十进制数的 ASCII 码转换为 BCD 码
=====
```

```
SSTACK   SEGMENT STACK
          DW 64 DUP(?)
SSTACK   ENDS
```

```
CODE SEGMENT
      ASSUME CS:CODE
```

```
START: MOV CX, 0005H           ;转换位数
      MOV DI, 3500H           ;ASCII 码首地址
A1:    MOV BL, 0FFH           ;将错误标志存入 BL
      MOV AL, [DI]
      CMP AL, 3AH
      JNB A2                   ;不低于 3AH 则转 A2
      SUB AL, 30H
      JB  A2                   ;低于 30H 则转 A2
      MOV BL, AL
A2:    MOV AL, BL              ;结果或错误标志送入 AL
      MOV [DI+0AH],AL         ;结果存入目标地址
      INC DI
      LOOP A1
      MOV AX, 4C00H
      INT 21H                 ;程序终止
CODE   ENDS
      END START
```

### 3. 将十六位二进制数转换为 ASCII 码表示的十进制数

十六位二进制数的值域为 0~65535, 最大可转换为五位十进制数。

五位十进制数可表示为：

$$N = D_4 \times 10^4 + D_3 \times 10^3 + D_2 \times 10^2 + D_1 \times 10 + D_0$$

$D_i$ ：表示十进制数 0~9

将十六位二进制数转换为五位 ASCII 码表示的十进制数，就是求  $D_1 \sim D_4$ ，并将它们转换为 ASCII 码。自行绘制程序流程图，实验程序参考例程。例程中源数存放于 3500H、3501H 中，转换结果存放于 3510H~3514H 单元中。

#### 实验步骤

- (1) 编写程序（例程文件名：A2-3.ASM），经编译、链接无误后，装入系统。
- (2) 在 3500H、3501H 中存入 0C 00。
- (3) 运行程序，待程序运行停止。
- (4) 检查运行结果，键入 D3510，结果应为：30 30 30 31 32。
- (5) 可反复测试几组数据，验证程序的正确性。

#### 实验程序清单

```

;=====
; 文件名：  A2-3.ASM
; 功能描述：将十六位二进制数转换为 ASCII 码表示的十进制数
;=====

```

```

SSTACK  SEGMENT STACK
          DW 64 DUP(?)
SSTACK  ENDS

```

```

CODE  SEGMENT
      ASSUME CS:CODE

```

```

START: MOV SI,3500H      ;源数据地址
      MOV DX,[SI]
      MOV SI,3515H      ;目标数据地址
A1:    DEC SI
      MOV AX,DX
      MOV DX,0000H
      MOV CX,000AH      ;除数 10
      DIV CX             ;得商送 AX, 得余数送 DX
      XCHG AX,DX
      ADD AL,30H         ;得 Di 得 ASCII 码
      MOV [SI],AL        ;存入目标地址

```

```

        CMP DX,0000H
        JNE A1          ;判断转换结束否，未结束则转 A1
A2:     CMP SI,3510H    ;与目标地址得首地址比较
        JZ A3          ;等于首地址则转 A3，否则将剩余地址
        DEC SI         ;中填 30H
        MOV AL,30H
        MOV [SI],AL
        JMP A2
A3:     MOV AX,4C00H
        INT 21H        ;程序终止

CODE    ENDS
        END START

```

#### 4. 十六进制数转换为 ASCII 码

由表 2.1 中十六进制数与 ASCII 码的对应关系可知：将十六进制数 0H~09H 加上 30H 后得到相应的 ASCII 码，AH~FH 加上 37H 可得到相应的 ASCII 码。将四位十六进制数存放于起始地址为 3500H 的内存单元中，把它们转换为 ASCII 码后存入起始地址为 350AH 的内存单元中。自行绘制流程图。

#### 实验步骤

- (1) 编写程序（例程文件名为 A2-4.ASM），经编译、链接无误后装入系统。
- (2) 在 3500H、3501H 中存入四位十六进制数 203B，即键入 E3500，然后输入 3B 20。
- (3) 先运行程序，待程序运行停止。
- (4) 键入 D350A，显示结果为：0000:350A 32 30 33 42 CC …。
- (5) 反复输入几组数据，验证程序功能。

#### 实验程序清单

```

;=====
; 文件名:   A2-4.ASM
; 功能描述: 十六进制数转换为 ASCII 码
;=====

SSTACK    SEGMENT STACK
            DW 64 DUP(?)
SSTACK    ENDS

CODE      SEGMENT
            ASSUME CS:CODE

```

```

START: MOV CX,0004H
        MOV DI,3500H      ;十六进制数源地址
        MOV DX,[DI]
A1:     MOV AX,DX
        AND AX,000FH      ;取低 4 位
        CMP AL,0AH
        JB A2             ;小于 0AH 则转 A2
        ADD AL,07H        ;在 A~FH 之间, 需多加上 7H
A2:     ADD AL,30H         ;转换为相应 ASCII 码
        MOV [DI+0DH],AL   ;结果存入目标地址
        DEC DI
        PUSH CX
        MOV CL,04H
        SHR DX,CL         ;将十六进制数右移 4 位
        POP CX
        LOOP A1
        MOV AX,4C00H
        INT 21H           ;程序终止
CODE    ENDS
        END START

```

### 5. BCD 码转换为二进制数

将四个二位十进制数的 BCD 码存放于 3500H 起始的内存单元中, 将转换的二进制数存入 3510H 起始的内存单元中, 自行绘制流程图并编写程序。

#### 实验步骤

- (1) 编写程序 (例程文件名为: A2-5.ASM), 经编译、链接无误后装入系统。
- (2) 将四个二位十进制数的 BCD 码存入 3500H~3507H 中, 即:  
先键入 E3500, 然后输入 01 02 03 04 05 06 07 08。
- (3) 先运行程序, 待程序运行停止。
- (4) 键入 D3510 显示转换结果, 应为: 0C 22 38 4E。
- (5) 反复输入几组数据, 验证程序功能。

#### 实验程序清单

```

;=====
; 文件名:   A2-5.ASM
; 功能描述: BCD 码转换为二进制数
;=====

```

```
SSTACK    SEGMENT STACK
            DW 64 DUP(?)
SSTACK    ENDS
```

```
CODE SEGMENT
        ASSUME CS:CODE
```

```
START: XOR AX, AX
        MOV CX, 0004H
        MOV SI, 3500H
        MOV DI, 3510H
A1:     MOV AL, [SI]
        ADD AL, AL
        MOV BL, AL
        ADD AL, AL
        ADD AL, AL
        ADD AL, BL
        INC SI
        ADD AL, [SI]
        MOV [DI], AL
        INC SI
        INC DI
        LOOP A1
        MOV AX, 4C00H
        INT 21H                ;程序终止
CODE    ENDS
        END START
```

## 2.2.4 思考题

1. 实验内容 1 中将一个五位十进制数转换为二进制数（十六位）时，这个十进制数最小可为多少，最大可为多少？为什么？
2. 将一个十六位二进制数转换为 ASCII 码十进制数时，如何确定  $D_i$  的值？
3. 在十六进制转换为 ASCII 码时，存转换结果后，为什么要把 DX 向右移四次？
4. 自编 ASCII 码转换十六进制、十六进制小数转换二进制、二进制转换 BCD 码的程序，并调试运行。

## 2.3 运算类编程实验

### 2.3.1 实验目的

- 1. 掌握使用运算类指令编程及调试方法。
- 2. 掌握运算类指令对各状态标志位的影响及其测试方法。
- 3. 学习使用软件监视变量的方法。

### 2.3.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 2.3.3 实验内容及步骤

80X86 指令系统提供了实现加、减、乘、除运算的基本指令，可对表 2.2 所示的数据类型进行算术运算。

表 2.2 数据类型算术运算表


数制	二进制		BCD 码	
	带符号	无符号	组合	非组合
运算符	+、-、×、÷		+、-	+、-、×、÷
操作数	字节、字、多精度		字节（二位数字）	字节（一位数字）

#### 1. 二进制双精度加法运算

计算  $X+Y=Z$ ，将结果  $Z$  存入某存储单元。实验程序参考例程。

本实验是双精度（2 个 16 位，即 32 位）加法运算，编程时可利用累加器  $AX$ ，先求低 16 位的和，并将运算结果存入低地址存储单元，然后求高 16 位的和，将结果存入高地址存储单元中。由于低 16 位运算后可能向高位产生进位，因此高 16 位运算时使用  $ADC$  指令，这样在低 16 位相加运算有进位时，高位相加会加上  $CF$  中的 1。

#### 实验步骤

- (1) 编写程序（例程文件名为：A3-1.ASM），经编译、链接无误后装入系统。
- (2) 程序装载完成后，点击‘变量区’标签将观察窗切换到变量监视窗口。
- (3) 点击，将变量  $XH, XL, YH, YL, ZH, ZL$  添加到变量监视窗中，然后修改  $XH$ ,



XL, YH, YL 的值, 如图 2.13 所示, 修改 XH 为 0015, XL 为 65A0, YH 为 0021, YL 为 B79E。

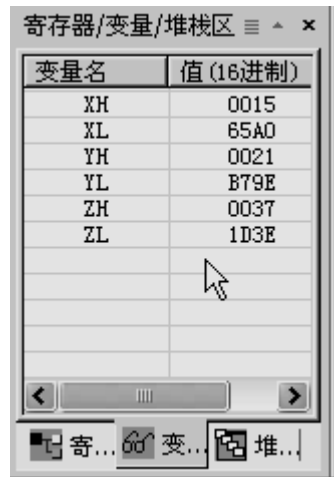


图 2.13 变量监视窗口

(4) 在 JMP START 语句行设置断点, 然后运行程序。

(5) 当程序遇到断点后停止运行, 查看变量监视窗口, 计算结果 ZH 为 0037, ZL 为 1D3E。

(6) 修改 XH, XL, YH 和 YL 的值, 再次运行程序, 观察实验结果, 反复测试几组数据, 验证程序的功能。

#### 实验程序清单

```
=====
; 文件名:   A3-1.ASM
; 功能描述: 二进制双精度加法运算
=====
```

```
SSTACK   SEGMENT STACK
          DW 64 DUP(?)
SSTACK   ENDS
```

```
PUBLIC XH, XL, YH, YL
PUBLIC ZH, ZL
```

```
DATA     SEGMENT
```

```
XL       DW ?           ;X 低位
XH       DW ?           ;X 高位
YL       DW ?           ;Y 低位
YH       DW ?           ;Y 高位
ZL       DW ?           ;Z 低位
ZH       DW ?           ;Z 高位
```

```
DATA    ENDS
```

```
CODE    SEGMENT
```

```
    ASSUME CS:CODE, DS:DATA
```

```
START:  MOV AX, DATA
```

```
    MOV DS, AX
```

```
    MOV AX, XL
```

```
    ADD AX, YL      ;X 低位加 Y 低位
```

```
    MOV ZL, AX      ;低位和存到 Z 的低位
```

```
    MOV AX, XH
```

```
    ADC AX, YH      ;高位带进位加
```

```
    MOV ZH, AX      ;存高位结果
```

```
    JMP START
```

```
CODE    ENDS
```


```
    END START
```

## 2. 十进制的 BCD 码减法运算

计算  $X - Y = Z$ ，其中 X、Y、Z 为 BCD 码。实验程序参考例程。

### 实验步骤

(1) 输入程序（例程文件名为 A3-2.ASM），编译、链接无误后装入系统。

(2) 点击  将变量 X, Y, Z 添加到变量监视窗中，并为 X, Y 赋值，假定存入 40 与 12 的 BCD 码，即 X 为 0400，Y 为 0102。

(3) 在 JMP START 语句行设置断点，然后运行程序。

(4) 程序遇到断点后停止运行，观察变量监视窗，Z 应为 0208。

(5) 重新修改 X 与 Y 的值，运行程序，观察结果，反复测试几次，验证程序正确性。

### 实验程序清单

```
=====
; 文件名:   A3-2.ASM
; 功能描述: 十进制的 BCD 码减法运算
=====

SSTACK    SEGMENT STACK
            DW 64 DUP(?)
SSTACK    ENDS
```

```

PUBLIC X, Y, Z
DATA SEGMENT
X      DW ?
Y      DW ?
Z      DW ?
DATA  ENDS

CODE SEGMENT
      ASSUME CS:CODE, DS:DATA

START: MOV AX, DATA
      MOV DS, AX
      MOV AH, 00H
      SAHF
      MOV CX, 0002H
      MOV SI, OFFSET X
      MOV DI, OFFSET Z
A1:   MOV AL, [SI]
      SBB AL, [SI+02H]
      DAS
      PUSHF
      AND AL, 0FH
      POPF
      MOV [DI], AL
      INC DI
      INC SI
      LOOP A1
      JMP START
CODE  ENDS
      END START

```

### 3. 乘法运算

实现十进制数的乘法运算，被乘数与乘数均以 BCD 码的形式存放在内存中，乘数为 1 位，被乘数为 5 位，结果为 6 位。实验程序参考例程。

#### 实验步骤

(1) 编写程序（例程文件名为 A3-3.ASM），编译、链接无误后装入系统。

(2) 查看寄存器窗口获得 CS 的值，使用 U 命令可得到数据段地址 DS，然后通过 E 命令为被乘数及乘数赋值，如被乘数：01 02 03 04 05，乘数：01，方法同实验内容 1。

(3) 运行程序，待程序运行停止。

(4) 通过 D 命令查看计算结果，应为：00 01 02 03 04 05；当在为被乘数和乘数赋值时，如果一个数的低 4 位大于 9，则查看计算结果将全部显示为 E。

(5) 反复测试几组数据，验证程序的正确性。

#### 实验程序清单

```

;=====
; 文件名:   A3-3.ASM
; 功能描述: 乘法运算
;=====

SSTACK    SEGMENT STACK
            DW 64 DUP(?)
SSTACK    ENDS

DATA      SEGMENT
DATA1 DB 5 DUP(?)           ;被乘数
DATA2 DB ?                  ;乘数
RESULT   DB 6 DUP(?)        ;计算结果
DATA     ENDS

CODE      SEGMENT
            ASSUME CS:CODE,DS:DATA

START:    MOV AX,DATA
            MOV DS,AX
            CALL INIT          ;初始化目标地址单元为 0
            MOV SI,OFFSET DATA2
            MOV BL,[SI]
            AND BL,0FH          ;得到乘数
            CMP BL,09H
            JNC ERROR
            MOV SI,OFFSET DATA1
            MOV DI,OFFSET RESULT
            MOV CX,0005H
A1:       MOV AL,[SI+04H]
            AND AL,0FH
            CMP AL,09H
            JNC ERROR

```

```
    DEC SI
    MUL BL
    AAM                                ;乘法调整指令
    ADD AL,[DI+05H]
    AAA
    MOV [DI+05H],AL
    DEC DI
    MOV [DI+05H],AH
    LOOP A1
A2:   MOV AX,4C00H
      INT 21H                          ;程序终止

;===将 RESULT 所指内存单元清零===
INIT:  MOV SI,OFFSET RESULT
      MOV CX,0003H
      MOV AX,0000H
A3:   MOV [SI],AX
      INC SI
      INC SI
      LOOP A3
      RET

;===错误处理===
ERROR: MOV SI,OFFSET RESULT ;若输入数据不符合要求
      MOV CX,0003H          ;则 RESULT 所指向内存单
      MOV AX,0EEEEH         ;元全部写入 E
A4:   MOV [SI],AX
      INC SI
      INC SI
      LOOP A4
      JMP A2
CODE  ENDS
      END    START
```

## 2.4 分支程序设计实验

### 2.4.1 实验目的

1. 掌握分支程序的结构。
2. 掌握分支程序的设计、调试方法。

### 2.4.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 2.4.3 实验内容

设计一数据块间的搬移程序。设计思想：程序要求把内存中一数据区（称为源数据块）传送到另一存储区（成为目的数据块）。源数据块和目的数据块在存储中可能有三种情况，如图 2.14 所示。

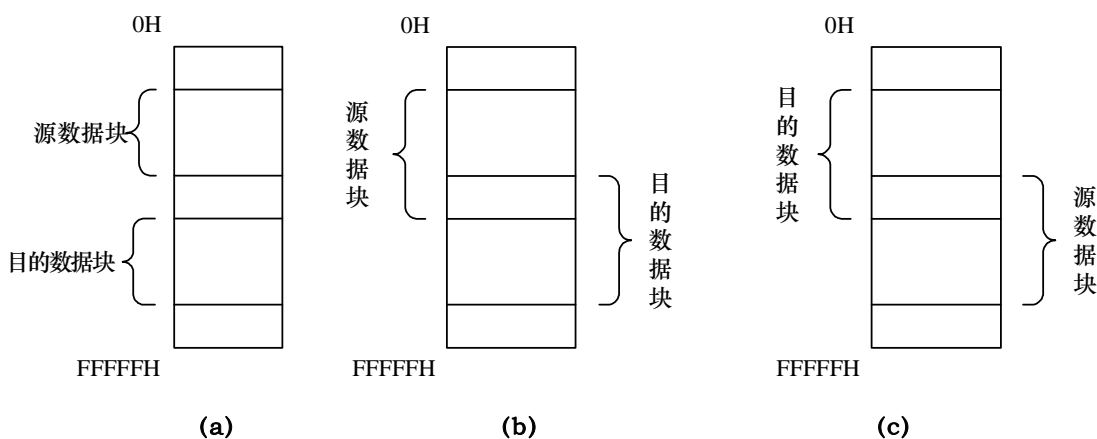


图 2.14 源数据块与目的数据块在存储中的位置情况

对于两个数据块分离的情况，如图 2.14 (a)，数据的传送从数据块的首地址开始，或从数据块的末地址开始均可。但是对于有重叠的情况，则要加以分析，否则重叠部分会因“搬移”而遭到破坏，可有如下结论：

当源数据块首地址 < 目的块首地址时，从数据块末地址开始传送数据，如图 2.14 (b) 所示。

当源数据块首地址 > 目的块首地址时，从数据块首地址开始传送数据，如图 2.14 (c) 所

示。

实验程序流程图如图 2.15 所示。

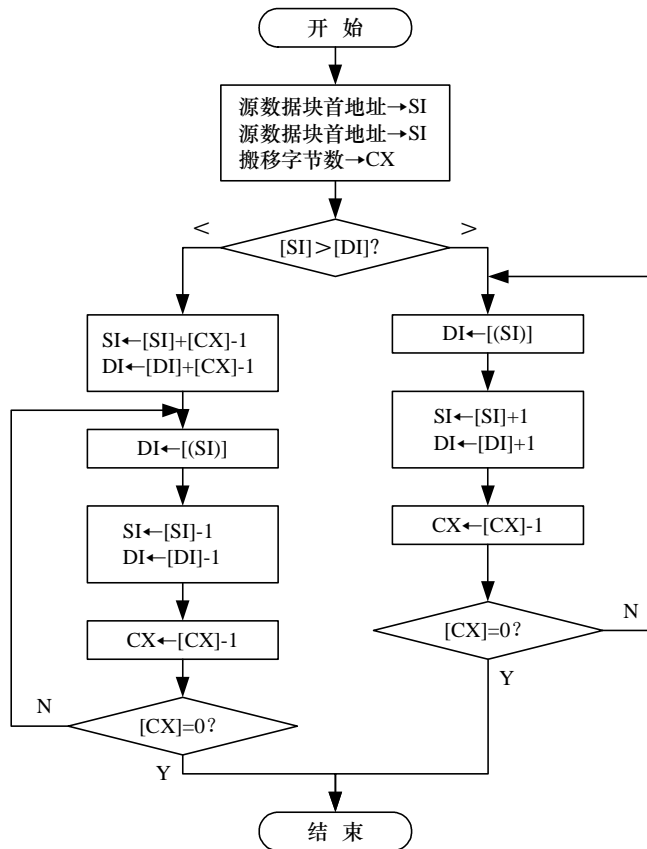


图 2.15 程序流程图

## 2.4.4 实验步骤

1. 按流程图编写实验程序（例程文件名为：A4-1.ASM），经编译、链接无误后装入系统。
2. 用 E 命令在以 SI 为起始地址的单元中填入 16 个数。
3. 运行程序，待程序运行停止。
4. 通过 D 命令查看 DI 为起始地址的单元中的数据是否与 SI 单元中数据相同。
5. 通过改变 SI、DI 的值，观察在三种不同的数据块情况下程序的运行情况，并验证程序的功能。

实验程序清单

```
;=====
; 文件名: A4-1.ASM
```

；功能描述：分支程序的设计

；=====

```
SSTACK    SEGMENT STACK
            DW 64 DUP(?)
SSTACK    ENDS
```

```
CODE SEGMENT
        ASSUME CS:CODE
```

```
START: MOV CX, 0010H
        MOV SI, 3100H
        MOV DI, 3200H
        CMP SI, DI
        JA A2
        ADD SI, CX
        ADD DI, CX
        DEC SI
        DEC DI
```

```
A1:     MOV AL, [SI]
        MOV [DI], AL
        DEC SI
        DEC DI
        DEC CX
        JNE A1
        JMP A3
```

```
A2:     MOV AL, [SI]
        MOV [DI], AL
        INC SI
        INC DI
        DEC CX
        JNE A2
```

```
A3:     MOV AX, 4C00H
        INT 21H
```

；程序终止

```
CODE ENDS
        END START
```



## 2.5 循环程序设计实验

### 2.5.1 实验目的

1. 加深对循环结构的理解。
2. 掌握循环结构程序设计的方法以及调试方法。

### 2.5.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 2.5.3 实验内容及步骤

1. 计算  $S=1+2\times 3+3\times 4+4\times 5+\cdots+N(N+1)$ ，直到  $N(N+1)$  项大于 200 为止。编写实验程序，计算上式的结果，参考流程图如图 2.16 所示。

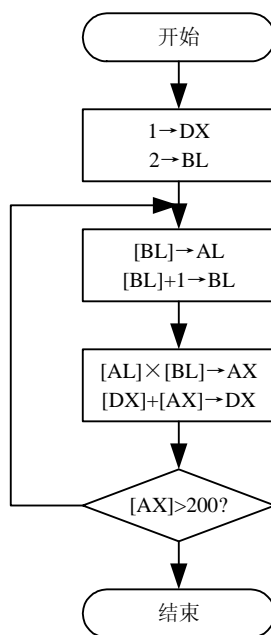


图 2.16 程序流程图

## 实验步骤

- (1) 编写实验程序（例程文件名为：A5-1.ASM），编译、链接无误后装入系统。
- (2) 运行程序，待程序运行停止。
- (3) 运算结果存储在寄存器 DX 中，查看结果是否正确。
- (4) 可以改变  $N(N+1)$  的条件来验证程序功能是否正确，但要注意，结果若大于 0FFFFH 将产生数据溢出。

## 实验程序清单

```

;=====
; 文件名:   A5-1.ASM
; 功能描述: 计算  $S=1+2\times 3+3\times 4+4\times 5+\dots+N(N+1)$ ,
;           直到  $N(N+1)$ 项大于 200 为止。
;=====

```

```

SSTACK   SEGMENT STACK
          DW 64 DUP(?)
SSTACK   ENDS

```

```

CODE     SEGMENT
          ASSUME CS:CODE

```

```

START: MOV DX,0001H
        MOV BL,02H
A1:     MOV AL,BL
        INC BL
        MUL BL
        ADD DX,AX           ;结果存于 DX 中
        CMP AX,00C8H       ;判断  $N(N+1)$ 与 200 的大小
        JNA A1
        MOV AX,4C00H
        INT 21H             ;程序终止
CODE     ENDS
        END START

```

## 2. 求某数据区内负数的个数

设数据区的第一单元存放区内单元数据的个数，从第二单元开始存放数据，在区内最后一个单元存放结果。为统计数据区内负数的个数，需要逐个判断区内的每一个数据，然后将所有数据中凡是符号位为 1 的数据的个数累加起来，即得到区内所包含负数的个数。

实验程序流程图如图 2.17 所示。

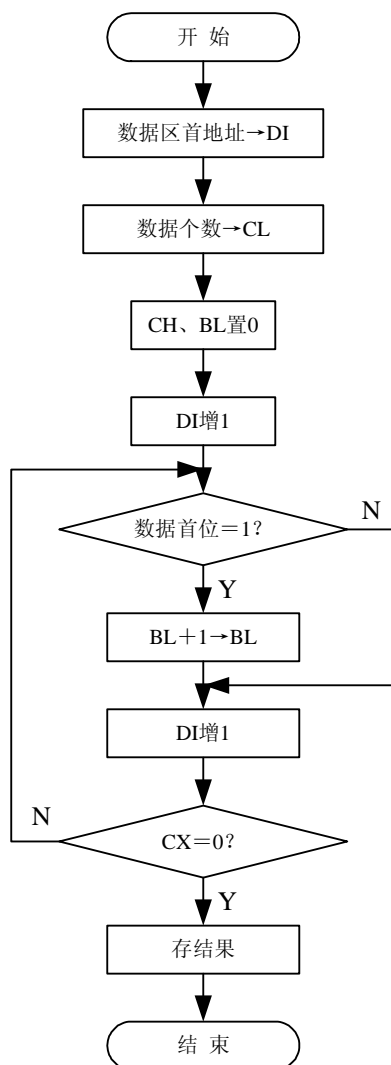


图 2.17 程序流程图

## 实验步骤

- (1) 按实验流程编写实验程序（例程文件名为：A5-2.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 键入 E3000，输入数据如下：  
 3000=06 （数据个数）  
 3001=12  
 3002=88  
 3003=82  
 3004=90  
 3005=22  
 3006=33
- (4) 先运行程序，待程序运行停止。

(5) 查看 3007 内存单元或寄存器 BL 中的内容，结果应为 03。

(6) 可以进行反复测试来验证程序的正确性。

### 实验程序清单

```

;=====
; 文件名:   A5-2.ASM
; 功能描述: 求某数据区内负数的个数
;=====

```

```

SSTACK    SEGMENT STACK
            DW 64 DUP(?)
SSTACK    ENDS

```

```

CODE SEGMENT
        ASSUME CS:CODE

```

```

START: MOV DI, 3000H      ;数据区首地址
        MOV CL, [DI]      ;取数据个数
        XOR CH, CH
        MOV BL, CH
        INC DI            ;指向第一个数据
A1:     MOV AL, [DI]
        TEST AL, 80H      ;检查数据首位是否为 1
        JE A2
        INC BL            ;负数个数加 1
A2:     INC DI
        LOOP A1
        MOV [DI], BL      ;保存结果
        MOV AX, 4C00H
        INT 21H           ;程序终止
CODE    ENDS
        END START

```

## 2.6 排序程序设计实验

### 2.6.1 实验目的

1. 掌握分支、循环、子程序调用等基本的程序结构。
2. 学习综合程序的设计、编制及调试。

### 2.6.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 2.6.3 实验内容及步骤

#### 1. 气泡排序法

在数据区中存放着一组数，数据的个数就是数据缓冲区的长度，要求采用气泡法对该数据区中的数据按递增关系排序。

设计思想：

(1) 从最后一个数（或第一个数）开始，依次把相邻的两个数进行比较，即第  $N$  个数与第  $N-1$  个数比较，第  $N-1$  个数与第  $N-2$  个数比较等等；若第  $N-1$  个数大于第  $N$  个数，则两者交换，否则不交换，直到  $N$  个数的相邻两个数都比较完为止。此时， $N$  个数中的最小数将被排在  $N$  个数的最前列。

(2) 对剩下的  $N-1$  个数重复 (1) 这一步，找到  $N-1$  个数中的最小数。

(3) 再重复 (2)，直到  $N$  个数全部排列好为止。

实验步骤

(1) 分析参考程序（例程文件名为：A6-1.ASM），绘制流程图并编写实验程序。

(2) 编译、链接无误后装入系统。

(3) 键入 E3000 命令修改 3000H~3009H 单元中的数，任意存入 10 个无符号数。

(4) 先运行程序，待程序运行停止。

(5) 通过键入 D3000 命令查看程序运行的结果。

(6) 可以反复测试几组数据，观察结果，验证程序的正确性。

实验程序清单

```
;=====
```

; 文件名: A6-1.ASM

; 功能描述: 气泡法排序

;=====

SSTACK SEGMENT STACK

DW 64 DUP(?)

SSTACK ENDS

CODE SEGMENT

ASSUME CS:CODE

START: MOV CX, 000AH

MOV SI, 300AH

MOV BL, 0FFH

A1: CMP BL, 0FFH

JNZ A4

MOV BL, 00H

DEC CX

JZ A4

PUSH SI

PUSH CX

A2: DEC SI

MOV AL, [SI]

DEC SI

CMP AL, [SI]

JA A3

XCHG AL, [SI]

MOV [SI+01H], AL

MOV BL, 0FFH

A3: INC SI

LOOP A2

POP CX

POP SI

JMP A1

A4: MOV AX, 4C00H

INT 21H

;程序终止

CODE ENDS

END START

## 2. 学生成绩名次表

将分数在 1~100 之间的 10 个成绩存入首地址为 3000H 的单元中, 3000H+I 表示学号为 I 的学生成绩。编写程序, 将排出的名次表放在 3100H 开始的数据区, 3100H+I 中存放的为学号为 I 的学生名次。

### 实验步骤

- (1) 绘制流程图, 并编写实验程序 (例程文件名为: A6-2.ASM)。
- (2) 编译、链接无误后装入系统。
- (3) 将 10 个成绩存入首地址为 3000H 的内存单元中。
- (4) 调试并运行程序。
- (5) 检查 3100H 起始的内存单元中的名次表是否正确。

### 实验程序清单

```

;=====
; 文件名:   A6-2.ASM
; 功能描述: 实现学生成绩名次表
;=====

```

```

SSTACK    SEGMENT STACK
            DW 64 DUP(?)
SSTACK     ENDS

```

```

CODE SEGMENT
        ASSUME CS:CODE

```

```

START: MOV AX,0000H
        MOV DS,AX
        MOV ES,AX
        MOV SI,3000H    ;存放学生成绩
        MOV CX,000AH    ;共 10 个成绩
        MOV DI,3100H    ;名次表首地址
A1:     CALL BRANCH      ;调用子程序
        MOV AL,0AH
        SUB AL,CL
        INC AL
        MOV BX,DX
        MOV [BX+DI],AL
        LOOP A1
        MOV AX,4C00H

```

INT 21H

;程序终止

;===扫描成绩表，得到最高成绩者的学号===

BRANCH: PUSH CX

MOV CX,000AH

MOV AL,00H

MOV BX,3000H

MOV SI,BX

A2: CMP AL,[SI]

JAE A3

MOV AL,[SI]

MOV DX,SI

SUB DX,BX

A3: INC SI

LOOP A2

ADD BX,DX

MOV AL,00H

MOV [BX],AL

POP CX

RET

CODE ENDS

END START



## 2.7 子程序设计实验

### 2.7.1 实验目的

- 1. 学习子程序的定义和调用方法。
- 2. 掌握子程序、子程序的嵌套、递归子程序的结构。
- 3. 掌握子程序的程序设计及调试方法。

### 2.7.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 2.7.3 实验内容及步骤

1. 求无符号字节序列中的最大值和最小值  
设有一字节序列，其存储首地址为 3000H，字节数为 08H。利用子程序的方法编程求出该序列中的最大值和最小值。程序流程图如图 2.18 所示。

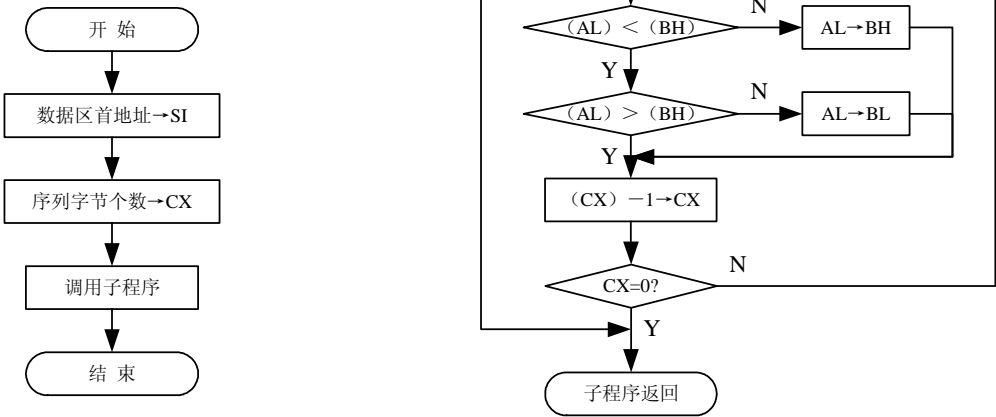


图 2.18 程序流程图

## 实验步骤

- (1) 根据程序流程图编写实验程序（例程文件名为：A7-1.ASM）。
- (2) 经编译、链接无误后装入系统。
- (3) 键入 E3000 命令，输入 8 个字节的数据，如：D9 07 8B C5 EB 04 9D F9。
- (4) 运行实验程序。
- (5) 点击停止按钮，停止程序运行，观察寄存器窗口中 AX 的值，AX 应为 F9 04，其中 AH 中为最大值，AL 中为最小值。
- (6) 反复测试几组数据，检验程序的正确性。

程序说明：该程序使用 BH 和 BL 暂存现行的最大值和最小值，开始时初始化成首字节的内容，然后进入循环操作，从字节序列中逐个取出一个字节的內容与 BH 和 BL 相比较，若取出的字节内容比 BH 的内容大或比 BL 的内容小，则修改之。当循环操作结束时，将 BH 送 AH，将 BL 送 AL，作为返回值，同时恢复 BX 原先的内容。

## 实验程序清单

```

;=====
; 文件名:   A7-1.asm
; 功能描述: 子程序实验，利用子程序求一组数中的最大
;           值和最小值
;=====
; 实验方法:
;   使用 E0000:3000 命令，改变连续 8 个地址单元的值，然后
;   运行程序，再点击停止，查看 AX 寄存器中的内容，AH 中
;   为最大值，AL 中为最小值。
;=====

SSTACK  SEGMENT STACK
          DW 64 DUP(?)
SSTACK  ENDS

CODE  SEGMENT
        ASSUME CS:CODE

START: MOV AX, 0000H
        MOV DS, AX
        MOV SI, 3000H    ; 数据区首址
        MOV CX, 0008H
        CALL BRANCH      ; 调用子程序

```

```

HERE: JMP HERE
;=====
; 子程序，出口参数在 AX 中
;=====
BRANCH PROC NEAR
    JCXZ A4
    PUSH SI
    PUSH CX
    PUSH BX
    MOV BH, [SI]
    MOV BL, BH
    CLD
A1:    LODSB
    CMP AL, BH
    JBE A2
    MOV BH, AL
    JMP A3
A2:    CMP AL, BL
    JAE A3
    MOV BL, AL
A3:    LOOP A1
    MOV AX, BX
    POP BX
    POP CX
    POP SI
A4:    RET
BRANCH ENDP
CODE ENDS
END START

```

## 2. 求 $N!$

利用子程序的嵌套和子程序的递归调用，实现  $N!$  的运算。根据阶乘运算法则，可以得：

$$N! = N (N-1)! = N (N-1) (N-2)! = \dots$$

$$0! = 1$$

由此可知，欲求  $N$  的阶乘，可以用一递归子程序来实现，每次递归调用时应将调用参数减 1，即求  $(N-1)$  的阶乘，并且当调用参数为 0 时应停止递归调用，且有  $0! = 1$ ，最后将每次调用的参数相乘得到最后结果。因每次递归调用时参数都送入堆栈，当  $N$  为 0 而程序开始返回时，应按嵌套的方式逐层取出相应的调用参数。

定义两个变量  $N$  及  $RESULT$ ， $RESULT$  中存放  $N!$  的计算结果， $N$  在  $00H \sim 08H$  之间取

值。

实验步骤

- (1) 依据设计思想绘制程序流程图，编写实验程序（例程文件名为：A7-2.ASM）。
- (2) 经编译、链接无误后装入系统。
- (3) 将变量 N 及 RESULT 加入变量监视窗口，并修改 N 值，N 在 00~08H 之间取值。
- (4) 在 JMP START 语句行设置断点，然后运行程序。
- (5) 当程序遇到断点后停止运行，此时观察变量窗口中 RESULT 的值是否正确，验证程序的正确性。
- (6) 改变变量 N 的值，然后再次运行程序，当程序停止在断点行后观察实验结果。

表 2.3 阶乘表

N	0	1	2	3	4	5	6	7	8
RESULT	1	1	2	6	18H	78H	02D0H	13B0H	9D80H

实验程序清单

```
=====
; 文件名:   A7-2.ASM
; 功能描述: 求 N!
;=====

SSTACK   SEGMENT STACK
            DW 64 DUP(?)
SSTACK   ENDS

PUBLIC N, RESULT      ;设置全局变量
DATA     SEGMENT
N         DB ?         ;N 的范围在 1~8 之间
RESULT    DW ?         ;N!的结果存于该变量中
DATA     ENDS

CODE     SEGMENT
            ASSUME CS:CODE, DS:DATA

START: MOV AX, DATA
            MOV DS, AX
            MOV AX, OFFSET RESULT
            PUSH AX
            MOV AL, N
```

```
        MOV AH, 00H
        PUSH AX
        MOV DI, 0000H
        CALL branch
        JMP START           ;在此处设置断点，观察变量
;===子程序===
branch: PUSH BP
        MOV BP,SP
        PUSH BX
        PUSH AX
        MOV BX,[BP+DI+06H]
        MOV AX,[BP+DI+04H]
        CMP AX,0000H
        JZ A1
        PUSH BX
        DEC AX
        PUSH AX
        CALL branch        ;递归调用
        MOV BX,[BP+DI+06H]
        MOV AX,[BX]
        PUSH BX
        MOV BX,[BP+DI+04H]
        MUL BX
        POP BX
        JMP A2
A1:     MOV AX, 0001H
A2:     MOV RESULT, AX
        POP AX
        POP BX
        POP BP
        RET 0004H
CODE    ENDS
        END START
```

## 2.8 查表程序设计实验

### 2.8.1 实验目的

学习查表程序的设计方法。

### 2.8.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 2.8.3 实验内容

所谓查表，就是根据某个值，在数据表格中寻找与之对应的一个数据，在很多情况下，通过查表比通过计算要使程序更简单，更容易编制。

通过查表的方法实现十六进制数转换为 ASCII 码。根据 2.2 章节的表 2.1 可知，0~9 的 ASCII 码为 30H~39H，而 A~F 的 ASCII 码为 41H~46H，这样就可以将 0~9 与 A~F 对应的 ASCII 码保存在一个数据表格中。当给定一个需要转换的十六进制数时，就可以快速的在表格中找出相应的 ASCII 码值。

### 2.8.4 实验步骤

1. 根据设计思想绘制程序流程图，编写实验程序（例程文件名为：A8-1.ASM）。
2. 经编译、链接无误后，将目标代码装入系统。
3. 将变量 HEX，ASCH，ASCL 添加到变量监视窗口中，并修改 HEX 的值，如 12。
4. 在语句 JMP AA1 处设置断点，然后运行程序。
5. 程序会在断点行停止运行，并更新变量窗口中变量的值，查看变量窗，ASCH 应为 31，ASCL 应为 32。
6. 反复修改 HEX 的值，观察 ASCH 与 ASCL 的值，验证程序功能。

实验程序清单

```
;=====
; 文件名:   A8-1.ASM
; 功能描述: 通过查表的方法实现十六进制到 BCD 码
```

```

;          的转换
;=====
; 实验方法:
;  程序下载完成后, 首先查看寄存器 CS 的值, 根据 CS 的
;  值使用反汇编 U 命令查看 DS 的值, 然后更改 DS 段 3000H
;  处的值, 即需转换的十六进制数, 转换结果存放在
;  3001H(高 4 位)和 3002H(低 4 位)中。
;=====

SSTACK    SEGMENT STACK
            DW 32 DUP(?)
SSTACK    ENDS

;=====
; 十六进制数 0~9 与 A,B,C,D,E,F 对应 ASC 码表
;=====
PUBLIC ASCH, ASCL, HEX
DATA      SEGMENT
TAB        DB 30H,31H,32H,33H,34H,35H,36H,37H,38H,39H
            DB 41H,42H,43H,44H,45H,46H
HEX        DB ?
ASCH       DB ?
ASCL       DB ?
DATA      ENDS

CODE      SEGMENT
            ASSUME CS:CODE, DS:DATA
START:    PUSH DS
            XOR AX, AX
            MOV AX, DATA
            MOV DS, AX
AA1:      MOV AL, HEX          ;需转换的十六进制数
            MOV AH, AL
            AND AL, 0F0H
            MOV CL, 04H
            SHR AL, CL
            MOV BX, OFFSET TAB ;表首地址存放于 BX 中
            XLAT
            MOV ASCH, AL      ;存放十六进制数高 4 位的 BCD 码

```

```
MOV AL, AH
AND AL, 0FH
XLAT
MOV ASCL, AL      ;存放十六进制数低 4 位的 BCD 码
NOP
JMP AA1
CODE ENDS
END START
```



## 2.9 输入输出程序设计实验

### 2.9.1 实验目的

1. 了解 INT 21H 各功能调用模块的作用及用法。
2. 掌握 Wmd86 软件界面下数据输入和输出的方法。

### 2.9.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 2.9.3 实验内容及步骤

INT 21H 功能调用使用说明如下：

- (1) 入口：AH=00H 或 AH=4CH  
功能：程序终止
- (2) 入口：AH=01H  
功能：读键盘输入到 AL 中并回显
- (3) 入口：AH=02H，DL=数据  
功能：写 DL 中的数据到显示屏
- (4) 入口：AH=08H  
功能：读键盘输入到 AL 中无回显
- (5) 入口：AH=09H，DS:DX=字符串首地址，字符串以 '\$' 结束  
功能：显示字符串，直到遇到 '\$' 为止
- (6) 入口：AH=0AH，DS:DX=缓冲区首地址，(DS:DX)=缓冲区最大字符数，  
(DS:DX+1)=实际输入字符数，(DS:DX+2)=输入字符串起始地址  
功能：读键盘输入的字符串到 DS:DX 指定缓冲区中并以回车结束

1. 编写实验程序，在显示器上的输出窗口显示 A~Z 共 26 个大写英文字母。

实验步骤

- (1) 编写实验程序（例程文件名为：A9-1.ASM 或 CDISPLAY.C），经编译、链接无误后装入系统。
- (2) 运行实验程序，观察实验结果。

(3) 修改实验程序，在显示器上显示 ‘GOOD AFTERNOON’，要求使用 AH=09 功能（显示一字符串功能块）完成。

#### 实验程序清单

```
=====
;
; 文件名:   A9-1.ASM
; 功能描述: 使用 INT 21H 功能调用实现显示 A--Z 共 26 个字母
;
=====
```

```
SSTACK   SEGMENT STACK
          DW 64 DUP(?)
SSTACK   ENDS
```

```
CODE SEGMENT
      ASSUME CS:CODE
```

```
START: MOV CX,001AH
        MOV DL,41H
        MOV AL,DL
A1:     MOV AH,02H
        INT 21H           ;功能调用
        INC DL
        PUSH CX
        MOV CX,0FFFFH
A2:     LOOP A2
        POP CX
        DEC CX
        JNZ A1
        MOV AX,4C00H
        INT 21H           ;程序终止
CODE ENDS
      END START
```

## 2. INT 21H 功能调用示例程序实验。

### 实验步骤

(1) 参考附录中 INT 21H 功能调用使用说明，编写实验程序（例程文件名为：A9-2. ASM），经编译、链接无误后装入系统。

(2) 运行实验程序，观察实验结果，可根据需要设断点观测实验现象。

(3) 仔细分析实验内容，理解 INT 21H 各功能调用的用法。

#### 实验程序清单

```

=====
; 文件名:   A9-2.ASM
; 功能描述: INT 21H 功能调用示例程序
=====
DATA1 SEGMENT
MES1  DB 'This is tangdu INT 21H !','$'
DATA1 ENDS
;-----
DATA2 SEGMENT
MES2  DB 0FFH DUP(?)
DATA2 ENDS
;-----
SSTACK  SEGMENT STACK
        DW 64 DUP(?)
SSTACK  ENDS
;-----
CODE  SEGMENT
        ASSUME CS:CODE
;-----
START:
        MOV AH,08H
        INT 21H                ;读键盘输入到 AL 中无回显

        MOV AH,01H
        INT 21H                ;读键盘输入到 AL 中并回显

        CALL ENTERR

        MOV CX,04H
        MOV DL,41H
AA:     MOV AH,02H
        INT 21H
        INC DL
        LOOP AA                ;将 DL 中的数据显示出来

        CALL ENTERR

```

```
MOV AX,DATA1
MOV DS,AX
MOV DX,OFFSET MES1
MOV AH,09H
INT 21H           ;显示数据段 DATA1 中的字符串

CALL ENTERR

MOV AX,DATA2
MOV DS,AX
MOV DX,OFFSET MES2
MOV AH,0AH
INT 21H           ;读入字符串放到数据段 DATA2 中,以回车结束

ADD DX,02H
MOV AH,09H
INT 21H           ;将数据段 DATA2 中的字符串显示出来

MOV AX,4C00H
INT 21H           ;程序终止

ENTERR:
MOV AH,02H
MOV DL,0DH
INT 21H           ;回车
MOV AH,02H
MOV DL,0AH
INT 21H           ;换行
RET
;-----
CODE ENDS
END START
```

## 第 3 章 32 位指令及其程序设计实验

在实模式下, 80X86 相当于一个可进行 32 位处理的快速 8086; 在实模式下为 80X86 编写的程序可利用 32 位的通用寄存器, 可使用新的指令, 可采用扩展寻址方式, 但段的最大长度仍是 64K。

### 3.1 80X86 指令及程序设计

#### 1. 说明处理器类型的伪指令

在缺省情况下, MASM 和 TASM 只识别 8086/8088 的指令, 为了让其识别 80X86 新增的指令或功能增强的指令, 必须告诉汇编程序处理器的类型, 如:

```
.386           ; 支持对 80386 非特权指令的汇编
.386P          ; 支持对 80386 所有指令的汇编
.386C          ; 支持对 80386 非特权指令的汇编
```

只有在使用说明处理器类型是 80X86 伪指令后, 汇编程序才识别表示 32 位寄存器的符号和表示始于 80X86 的指令的助记符。

#### 2. 关键段属性类型的说明

在实模式下, 80X86 的段保持与 8086/8088 兼容, 所以段的最大长度仍是 64K, 这样的段称为 16 位段。但在保护模式下, 段长度可达到 4G, 这样的段称为 32 位段。为了兼容, 在保护模式下, 也可使用 16 位段。

完整段定义的一般格式如下:

段名 SEGMENT[定位类型] [组合类型] [‘类别’] [属性类型]

属性类型说明符号是“USE16”和“USE32”。各表示 16 位段和 32 位段。在使用“.386”等伪指令指示处理器类型 80X86 后, 缺省的属性类型是 USE32; 如果没有指示处理器类型 80X86, 那么缺省的属性类型是 USE16。

例如定义一个 32 位段:

```
CSEG SEGMENT PARA USE32
.....
.....
CSEG ENDS
```

例如定义一个 16 位段

```
CSEG SEGMENT PARA USE16
.....
.....
CSEG ENDS
```

#### 3. 操作数和地址长度前缀

虽然在实模式下只能使用 16 位段, 但可以使用 32 位操作数, 也可使用以 32 位形式表示

的存储单元地址，这是利用操作数长度前缀 66H 和存储器地址长度前缀 67H 来表示的。

在 16 位代码段中，正常操作数的长度是 16 位或 8 位。在指令前加上操作数长度前缀 66H 后，操作数长度就成为 32 位或 8 位，也即原来表示 16 位操作数的代码成为表示 32 位操作数的代码。一般情况下，不在源程序中直接使用操作数长度前缀，而是直接使用 32 位操作数，操作数长度前缀由汇编程序在汇编时自动加上。

试比较如下在 16 位代码段中的汇编格式指令和对应的机器码（注释部分）：

```
.386
TEST16    SEGMENT PARA    USE16
          .....
                                ; 66H
MOV        EAX, EBX            ; 8BH, C3H
MOV        AX, BX              ; 8BH, C3H
MOV        AL, BL              ; 8AH, C3H
          .....
TEST16     ENDS
```

32 位代码段情况恰好相反。在 32 位代码段中，正常操作数长度是 32 位或 8 位。在指令前加上操作数长度前缀 66H 后，操作数长度就成为 16 位或 8 位。不在 32 位代码的源程序中直接使用操作数长度前缀 66H 表示使用 16 位操作数，而是直接使用 16 位操作数，操作数长度前缀由汇编程序在汇编时自动加上。

试比较如下在 32 位代码段中的汇编格式指令和对应的机器码（注释部分）：

```
.386
TEST32     SEGMENT PARA    USE32
          .....
MOV        EAX, EBX            ; 8BH, C3H
                                ; 66H
MOV        AX, BX              ; 8BH, C3H
MOV        AL, BL              ; 8AH, C3H
          .....
TEST32     ENDS
```

通过存储器地址长度前缀 67H 区分 32 位存储器地址和 16 位存储器地址的方法与上述通过操作数长度前缀 66H 区分 32 位操作数和 16 位操作数的方法类似。在源程序中可根据需要使用 32 位地址，或者 16 位地址。汇编程序在汇编程序时，对于 16 位的代码段，在使用 32 位存储器地址的指令前加上前缀 67H；对于 32 位代码段，在使用 16 位存储器地址的指令前加上前缀 67H。

在一条指令前能既有操作数长度前缀 66H，又有存储器地址长度前缀 67H。

## 3.2 32 位指令及寻址实验

### 3.2.1 实验目的

1. 熟悉 32 位通用寄存器的使用。
2. 熟悉部分新增指令的使用。
3. 熟悉部分扩展寻址方式的使用。

### 3.2.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 3.2.3 实验内容一

编写一个汇编程序，学习 32 位寄存器和 32 位指令使用的基本用法，对存储区中的一组双字进行排序，并将排序结果显示在屏幕上。

#### 1. 实验步骤

- (1) 运行 Wmd86 集成操作软件，进入 Wmd86 集成开发环境。
- (2) 在菜单“设置\语言”栏，选择“汇编语言”，如图 3.1：

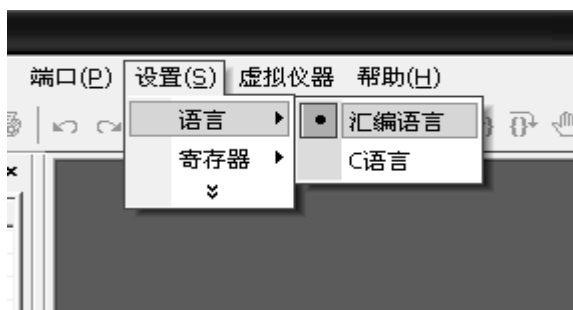


图 3.1 语言设置环境界面

- (3) 在菜单“设置\寄存器”栏，选择“32 位寄存器”（本章实验选择 32 位寄存器），如图 3.2：

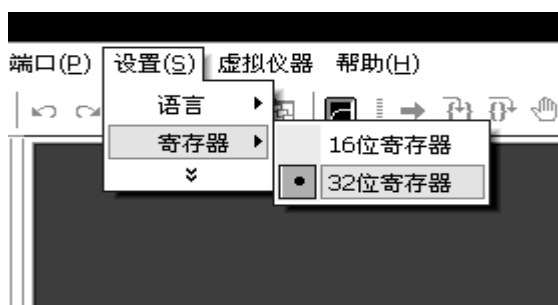


图 3.2 寄存器设置环境界面

设置好以后，下次再启动软件，设置栏将保持这次的修改不变。

(4) 设置完毕后，点击新建或按 Ctrl+N 组合键来新建一个文档，如图 3.3 所示，默认文件名为 Wmd861。

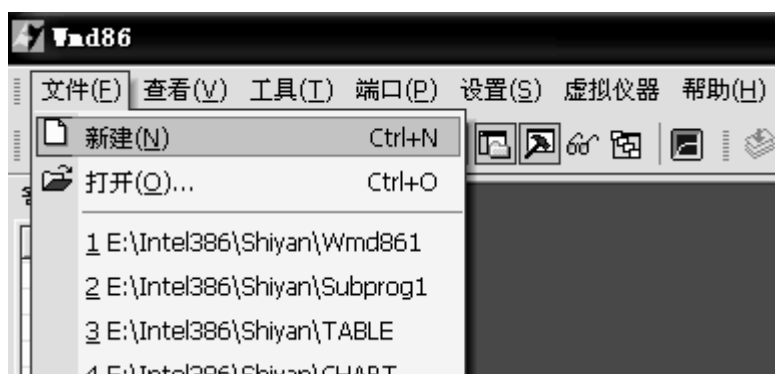


图 3.3 新建文档界面

(5) 编写实验程序（例程文件名为：3-2-1.ASM），如图 3.4 所示，并保存，此时系统会提示输入新的文件名，输完后点击保存。



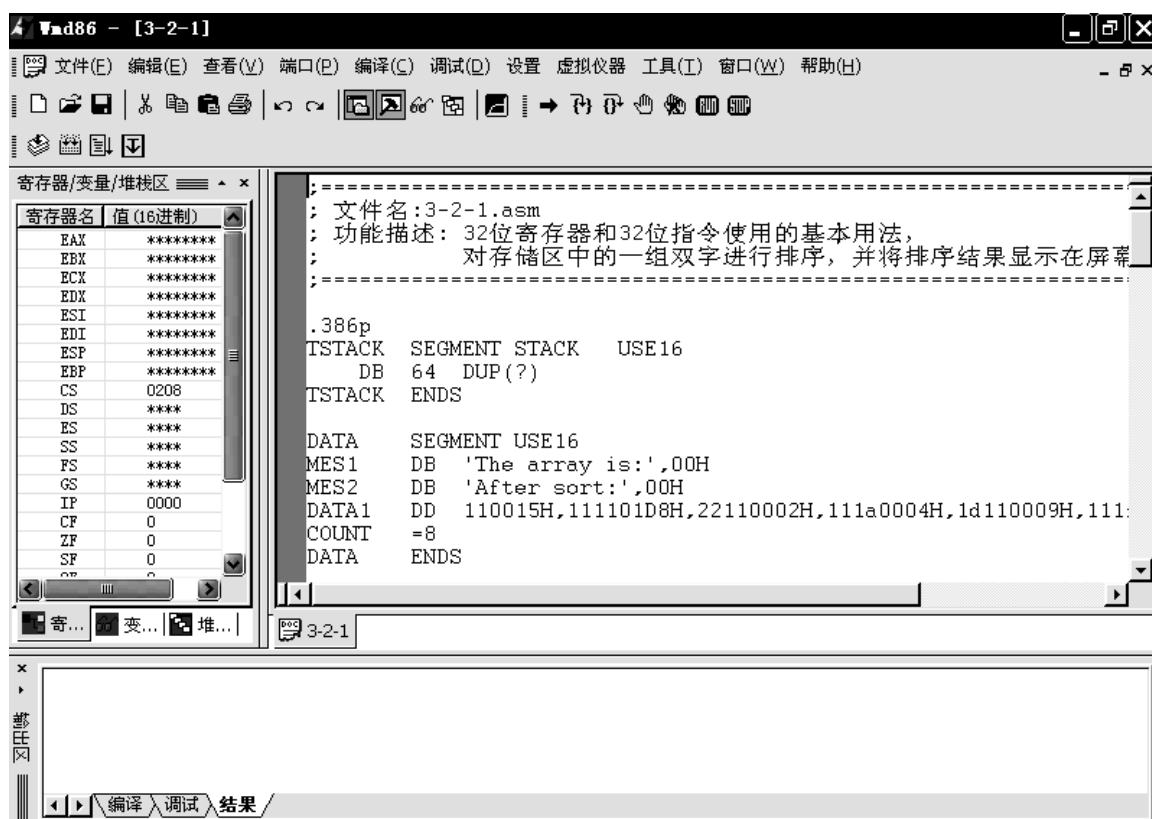




图 3.4 保存文件图

(6) 点击 ，编译文件，若程序编译无误，则可以继续点击  进行链接，链接无误后方可加载程序。编译、链接后输出如图 3.5 所示的输出信息。

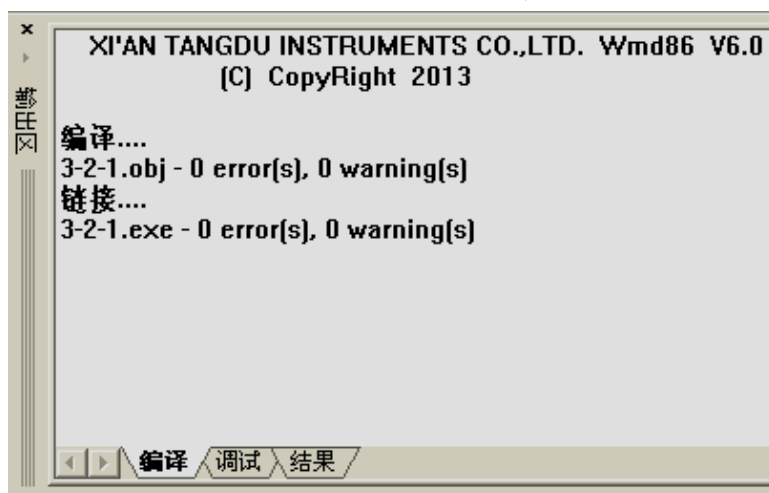




图 3.5 编译链接信息界面

(7) 连接 PC 与实验系统的通讯电缆，打开实验系统电源。

(8) 编译、链接都正确并且上下位机通讯成功后, 就可以下载程序, 联机调试了。可以通过端口列表中的“端口测试”来检查通讯是否正常。点击下载程序。为编译、链接、下载组合按钮, 通过该按钮可以将编译、链接、下载一次完成。下载成功后, 在输出区的结果窗中会显示“加载成功!”, 表示程序已正确下载。起始运行语句下会有一条绿色的背景。如图 3.6 所示:

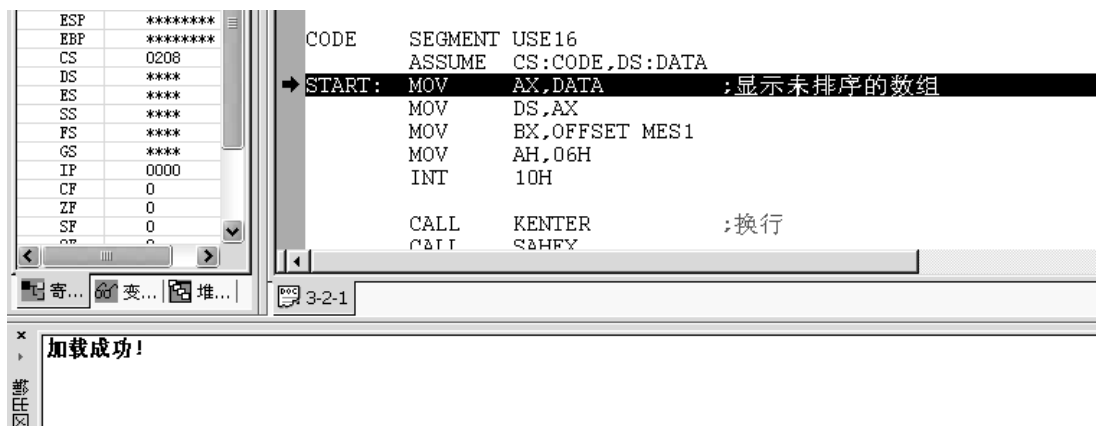



图 3.6 开始运行界面图

(9) 点击按钮运行程序, 待程序运行停止后, 观察运行结果。如图 3.7:

程序运行结果为:

The array is:

00000032 11D10203 0111F044 1D110009 111A0004 22110002 111101D8  
00110015

After sort:

00000032 00110015 0111F044 111101D8 111A0004 11D10203 1D110009  
22110002

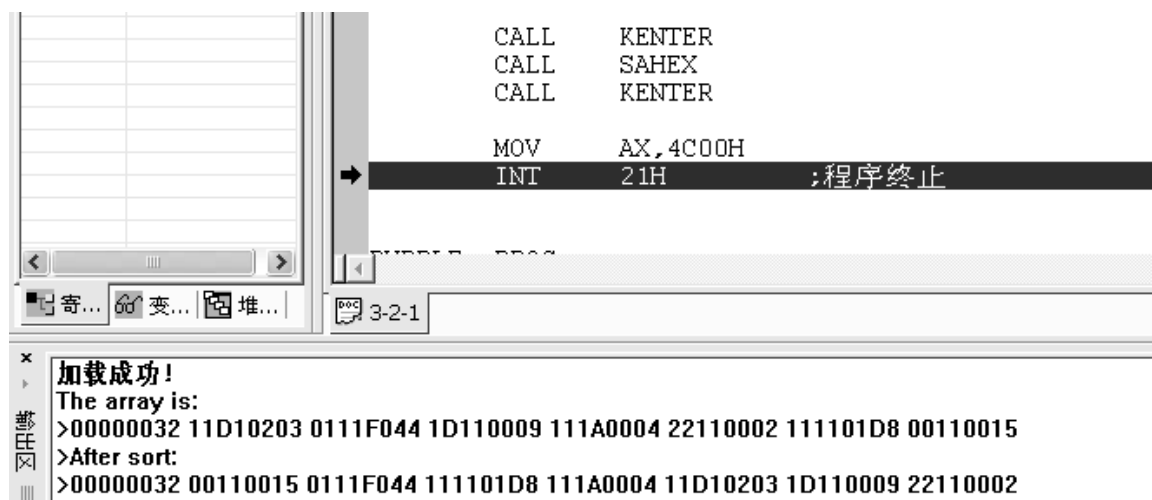


图 3.7 程序运行结果

## 实验程序清单

```

=====
;
; 文件名:3-2-1.asm
; 功能描述: 32 位寄存器和 32 位指令使用的基本用法,
;           对存储区中的一组双字进行排序, 并将排序结果显示在屏幕上。
;
=====
;

.386p
TSTACK  SEGMENT  STACK USE16
        DB 64  DUP(?)
TSTACK  ENDS

DATA  SEGMENT  USE16
MES1  DB 'The array is:', '$'
MES2  DB 'After sort:', '$'
DATA1 DD
        110015H,111101D8H,22110002H,111a0004H,1d110009H,111f044H,11d1020
3H,32H
COUNT =8
DATA  ENDS

CODE  SEGMENT  USE16

```

```

        ASSUME    CS:CODE,DS:DATA
START:  MOV     AX,DATA           ;显示未排序的数组
        MOV      DS,AX
        MOV     DX,OFFSET MES1
        MOV     AH,09H
        INT     21H

        CALL    KENTER           ;换行
        CALL    SAHEX
        CALL    KENTER

        CALL    BUBBLE           ;显示排序后的数组
        MOV     DX,OFFSET MES2
        MOV     AH,09H
        INT     21H

        CALL    KENTER
        CALL    SAHEX
        CALL    KENTER

        MOV     AX,4C00H
        INT     21H              ;程序终止

```

```

BUBBLE  PROC
        XOR     ESI,ESI
        XOR     ECX,ECX
        MOV     SI,OFFSET DATA1
        MOV     CX,COUNT
L1:     XOR     EBX,EBX
L2:     CMP     EBX,ECX
        JAE     LB
        MOV     EAX,[ESI+EBX*4+4]
        CMP     [ESI+EBX*4],EAX
        JGE     LNS
        XCHG    [ESI+EBX*4],EAX
        MOV     [ESI+EBX*4+4],EAX
LNS:    INC     EBX
        JMP     L2
BUBBLE  ENDP

```

```

LB:    LOOP  L1
      RET
BUBBLE ENDP

SAHEX PROC  NEAR
      XOR     ESI,ESI
      XOR     ECX,ECX
      MOV     SI,OFFSET DATA1
      MOV     CX,COUNT*4
C1:    MOV     EBX,ECX
      DEC     EBX
      MOV     AL,DS:[ESI+EBX]
      AND     AL,0F0H           ;取高 4 位
      SHR     AL,4
      CMP     AL,0AH           ;是否是 A 以上的数
      JB      C2
      ADD     AL,07H
C2:    ADD     AL,30H
      MOV     DL,AL
      MOV     AH,02H
      INT     21H              ;显示字符
      MOV     AL,DS:[ESI+EBX]
      AND     AL,0FH           ;取低 4 位
      CMP     AL,0AH
      JB      C3
      ADD     AL,07H
C3:    ADD     AL,30H
      MOV     DL,AL            ;显示字符
      MOV     AH,02H
      INT     21H
      TEST    EBX,03H
      JNZ     C4
      MOV     DL,20H
      MOV     AH,02H
      INT     21H              ;空格
C4:    LOOP    C1
      RET
SAHEX ENDP

```

```

KENTER    PROC    NEAR
            MOV     AH,02H
            MOV     DL,0DH
            INT     21H                ;回车
            MOV     AH,02H
            MOV     DL,0AH
            INT     21H                ;换行
            RET
KENTER    ENDP

CODE      ENDS
            END      START

```

### 3.2.4 实验内容二

编写一个汇编程序，学习 32 位寄存器和 32 位指令使用的基本用法，将一组 ASCII 字符转换成 16 进制数码，并在屏幕上显示出来。

#### 1. 实验步骤

- (1) 编写实验程序（例程文件名为：3-2-2.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 调试并运行程序。
- (4) 运行结果：

在输出区的结果栏将会显示：

This is tangdu speaking!

Show this sentence as hex:

54,68,69,73,20,74,61,6E,67,64,75,20,73,70,65,61,6B,69,6E,67,21

实验程序清单

```

;=====
; 文件名:3-2-2.asm
; 功能描述: 32 位寄存器和 32 位指令使用的基本用法,
;           将一组 ASCII 字符转换成 16 进制数码，并在屏幕上显示出来。
;=====

```

.386P

```

TSTACK    SEGMENT STACK USE16
            DB 64 DUP(?)

```

```
TSTACK    ENDS
```

```
DATA      SEGMENT USE16
```

```
MES0      DB 'This is tangdu speaking!','$'
```

```
MES1      DB 'Show this sentence as hex:$'
```

```
;54,68,69,73,20,74,61,6E,67,64,75,20,73,70,65,61,6B,69,6E,67,21
```

```
BUF       DB 65 DUP(?)
```

```
DATA      ENDS
```

```
CODE      SEGMENT USE16
```

```
    ASSUME    CS:CODE,DS:DATA
```

```
START:    MOV     AX,DATA
```

```
    MOV       DS,AX
```

```
    MOV       DX,OFFSET MES0 ;Show "This is tangdu speaking!"
```

```
    MOV       AH,09H
```

```
    INT       21H
```

```
    CALL      KENTER
```

```
    MOV       DX,OFFSET MES1 ;Show Sentence as hex
```

```
    MOV       AH,09H
```

```
    INT       21H
```

```
    CALL      KENTER
```

```
    CALL      SAHEX
```

```
    MOV       DX,OFFSET BUF
```

```
    MOV       AH,09H
```

```
    INT       21H
```

```
    CALL      KENTER
```

```
    MOV       AX,4C00H
```

```
    INT       21H ;程序终止
```

```
SAHEX PROC NEAR
```

```
    CBYTE =24
```

```
    PUSHAD ;将所有 32 位寄存器压栈
```

```
    MOV       DI,OFFSET MES0
```

```
    MOVZX     EDI,DI ;零扩展指令
```

```
    MOV       AX,DATA
```

```
    MOV       GS,AX ;使用 GS 段
```

```
    MOV       SI,OFFSET BUF
```

```
    MOVZX     ESI,SI
```

```
    MOV       ECX,CBYTE
```

```
C1:    MOV     AL,DS:[EDI]
```

```

        AND     AL,0F0H           ;取高 4 位
        SHR     AL,4
        CMP     AL,0AH           ;是否是 A 以上的数
        JB      C2
        ADD     AL,07H
C2:     ADD     AL,30H
        MOV     GS:[ESI],AL
        MOV     AL,DS:[EDI]
        AND     AL,0FH           ;取低 4 位
        CMP     AL,0AH
        JB      C3
        ADD     AL,07H
C3:     ADD     AL,30H
        MOV     GS:[ESI+1],AL
        MOV     BYTE PTR GS:[ESI+2],20H ;在每个字符间加入空格
        ADD     ESI,3
        INC     EDI
        LOOP    C1

        MOV     AL,'$'
        MOV     GS:[ESI],AL

        POPAD           ;弹出所有寄存器值
        RET
SAHEX ENDP

KENTER  PROC  NEAR
        MOV     AH,02H
        MOV     DL,0DH
        INT     21H           ;回车
        MOV     AH,02H
        MOV     DL,0AH
        INT     21H           ;换行
        RET
KENTER  ENDP
CODE  ENDS
        END      START

```



## 第 4 章 80X86 微机接口技术及其应用实验

接口技术是把由处理器、存储器等组成的基本系统与外部设备连接起来,从而实现 CPU 与外部设备通信的一门技术。微机的应用是随着外部设备的不断更新和接口技术的不断发展而深入到各行各业,任何微机应用开发工作都离不开接口的设计、选用及连接。微机应用系统需要设计的硬件是一些接口电路,所要编写的软件是控制这些接口电路按要求工作的驱动程序。因此,接口技术是微机应用中必不可少的基本技能。

### 4.1 静态存储器扩展实验

#### 4.1.1 实验目的

1. 了解存储器扩展的方法和存储器的读/写。
2. 掌握 CPU 对 16 位存储器的访问方法。

#### 4.1.2 实验设备

PC 机一台, TD-PITE 实验装置一套。

#### 4.1.3 实验内容

按照规则字写存储器,编写实验程序,将 0000H~000FH 共 16 个数写入 SRAM 的从 0000H 起始的一段空间中,然后通过系统命令查看该存储空间,检测写入数据是否正确。

#### 4.1.4 实验原理

存储器是用来存储信息的部件,是计算机的重要组成部分,静态 RAM 是由 MOS 管组成的触发器电路,每个触发器可以存放 1 位信息。只要不掉电,所储存的信息就不会丢失。因此,静态 RAM 工作稳定,不要外加刷新电路,使用方便。但一般 SRAM 的每一个触发器是由 6 个晶体管组成,SRAM 芯片的集成度不会太高,目前较常用的有 6116 (2K×8 位),6264 (8K×8 位) 和 62256 (32K×8 位)。本实验平台上选用的是

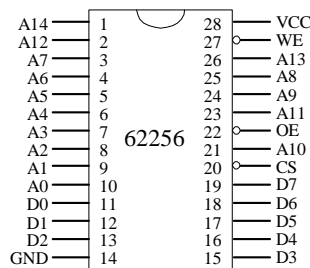


图 4.1.1 62256 引脚图

62256, 两片组成  $32\text{K} \times 16$  位的形式, 共  $64\text{K}$  字节。62256 的外部引脚图如图 4.1.1 所示。

本系统采用准 32 位 CPU, 具有 16 位外部数据总线, 即  $\text{D0}$ 、 $\text{D1}$ 、 $\dots$ 、 $\text{D15}$ , 地址总线为  $\text{BHE}\#$  ( $\#$  表示该信号低电平有效)、 $\text{BLE}\#$ 、 $\text{A1}$ 、 $\text{A2}$ 、 $\dots$ 、 $\text{A20}$ 。存储器分为奇体和偶体, 分别由字节允许线  $\text{BHE}\#$  和  $\text{BLE}\#$  选通。

存储器中, 从偶地址开始存放的字称为规则字, 从奇地址开始存放的字称为非规则字。处理器访问规则字只需要一个时钟周期,  $\text{BHE}\#$  和  $\text{BLE}\#$  同时有效, 从而同时选通存储器奇体和偶体。处理器访问非规则字却需要两个时钟周期, 第一个时钟周期  $\text{BHE}\#$  有效, 访问奇字节; 第二个时钟周期  $\text{BLE}\#$  有效, 访问偶字节。处理器访问字节只需要一个时钟周期, 视其存放单元为奇或偶, 而  $\text{BHE}\#$  或  $\text{BLE}\#$  有效, 从而选通奇体或偶体。写规则字和非规则字的简单时序图如图 4.1.2 所示。

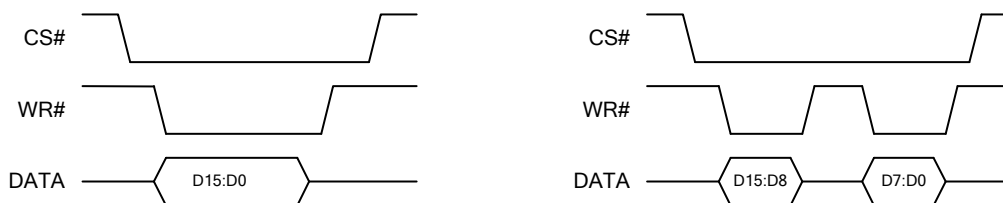


图 4.1.2 写规则字 (左) 和非规则字 (右) 简单时序图

实验单元电路如图 4.1.3 所示。

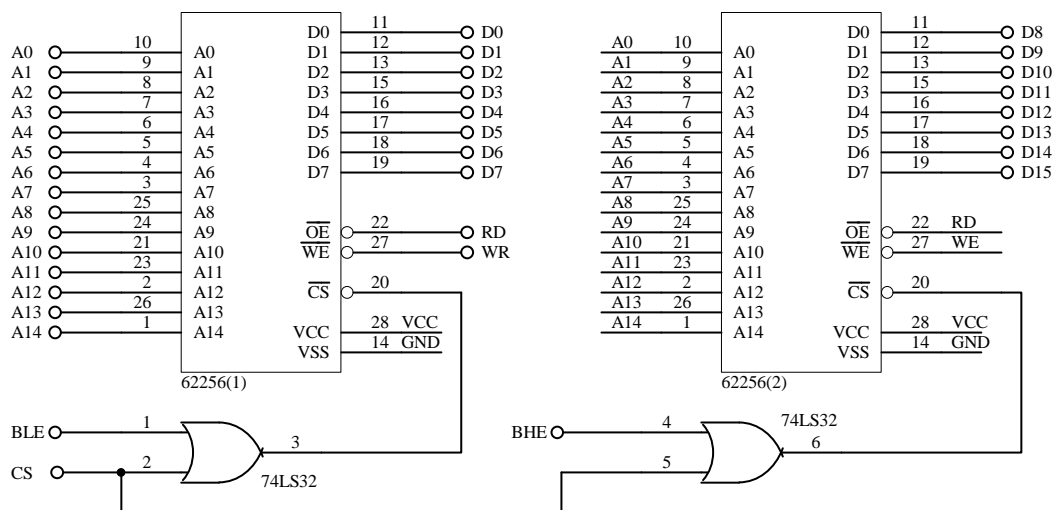


图 4.1.3 SRAM 单元电路图

### 4.1.5 实验步骤

(注：本章实验选择 16 位寄存器)

1. 实验接线图如图 4.1.4 所示，按图接线。

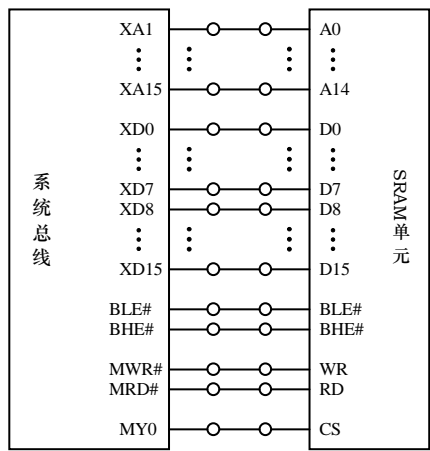


图 4.1.4 SRAM 实验接线图

2. 编写实验程序（例程文件名为：MEM.ASM），经编译、链接无误后装入系统。

3. 先运行程序，待程序运行停止。

4. 通过 D 命令查看写入存储器中的数据：

D8000: 0000 回车，即可看到存储器中的数据，应为 0000、0001、0002、…、000F 共 16 个字。

实验程序清单

```
=====
;
; 文件名: MEM.ASM
; 功能描述: 扩展存储器实验，访问 16 位存储器，将 16 个数写入
;           从 8000:0000H 开始的连续地址单元，然后使用 D 命令查看
;           数据是否被正确写入
;
=====
```

```
SSTACK   SEGMENT STACK
          DW 32 DUP(?)
SSTACK   ENDS
CODE     SEGMENT
START    PROC FAR
          ASSUME CS:CODE
          MOV AX, 8000H           ; 存储器扩展空间段地址
```

```
        MOV DS, AX
AA0:    MOV SI, 0000H    ; 数据首地址
        MOV CX, 0010H
        MOV AX, 0000H
AA1:    MOV [SI], AX
        INC AX
        INC SI
        INC SI
        LOOP AA1
        MOV AX, 4C00H
        INT 21H          ;程序终止
START   ENDP
CODE    ENDS
        END START
```

5. 改变实验程序，按非规则字写存储器，观察实验结果。

给 SI 寄存器赋奇地址数，

```
MOV SI,0001H
```

即为非规则字写存储器。

6. 改变实验程序，按字节方式写存储器，观察实验现象。

```
AA0:    MOV SI, 0000H
        MOV CX, 0010H
        MOV AL, 00H
AA1:    MOV [SI], AL
        INC AL
        INC SI
        LOOP AA1
```

## 4.2 8259 中断控制实验

### 4.2.1 实验目的

1. 掌握 8259 中断控制器的工作原理。
2. 学习 8259 的应用编程方法。
3. 掌握 8259 级联方式的使用方法。

### 4.2.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 4.2.3 实验内容

1. 利用系统总线上中断请求信号 MIR7,设计一个单一中断请求实验。
2. 利用系统总线上中断请求信号 MIR6 和 MIR7,设计一个双中断优先级应用实验，观察 8259 对中断优先级的控制。
3. 利用系统总线上中断请求信号 MIR7 和 SIR1，设计一个级连中断应用实验。

### 4.2.4 实验原理

#### 1. 中断控制器 8259 简介

在 Intel 386EX 芯片中集成有中断控制单元 (ICU)，该单元包含有两个级联中断控制器，一个为主控制器，一个为从控制器。该中断控制单元就功能而言与工业上标准的 82C59A 是一致的，操作方法也相同。从片的 INT 连接到主片的 IR2 信号上构成两片 8259 的级联。

在 TD-PITE 实验系统中，将主控制器的 IR6、IR7 以及从控制器的 IR1 开放出来供实验使用，主片 8259 的 IR4 供系统串口使用。8259 的内部连接及外部管脚引出如图 4.2.1:

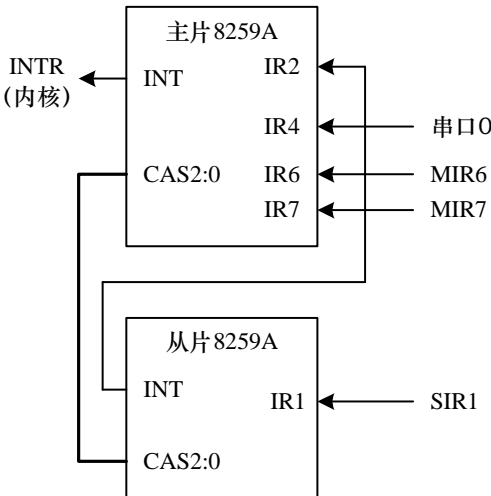


图 4.2.1 8259 内部连续及外部管脚引出图

表 4.2.1 列出了中断控制单元的寄存器相关信息。

表 4.2.1 ICU 寄存器列表

寄存器	口地址	功能描述
ICW1 (主)	0020H	初始化命令字 1: 决定中断请求信号为电平触发还是边沿触发。
ICW1 (从) (只写)	00A0H	
ICW2 (主)	0021H	初始化命令字 2: 包含了 8259 的基址中断向量号, 基址中断向量是 IR0 的向量号, 基址加 1 就是 IR1 的向量号, 依此类推。
ICW2 (从) (只写)	00A1H	
ICW3 (主) (只写)	0021H	初始化命令字 3: 用于识别从 8259 设备连接到主控制器的 IR 信号, 内部的从 8259 连接到主 8259 的 IR2 信号上。
ICW3 (从) (只写)	00A1H	
ICW4 (主)	0021H	初始化命令字 4: 选择特殊全嵌套或全嵌套模式, 使能中断自动结束方式。
ICW4 (从) (只写)	00A1H	
OCW1 (主)	0021H	操作命令字 1: 中断屏蔽操作寄存器, 可屏蔽相应的中断信号。
OCW1 (从) (读/写)	00A1H	
OCW2 (主)	0020H	操作命令字 2: 改变中断优先级和发送中断结束命令。
OCW2 (从) (只写)	00A0H	
OCW3 (主)	0020H	操作命令字 3: 使能特殊屏蔽方式, 设置中断查询方式, 允许读出中断请求寄存器和当前中断服务寄存器。
OCW3 (从) (只写)	00A0H	
IRR (主)	0020H	中断请求: 指出挂起的中断请求。
IRR (从) (只读)	00A0H	
ISR (主)	0020H	当前中断服务: 指出当前正在被服务的中断请求。
ISR (从) (只读)	00A0H	

POLL (主)	0020H	查询状态字： 表明连接到 8259 上的设备是否需要服务，如果有中断请求，该字表明当前优先级最高的中断请求。
POLL (从)	0021H	
POLL (只读)	00A0H 00A1H	

初始化命令字 1 寄存器 (ICW1) 说明见图 4.2.2 所示。

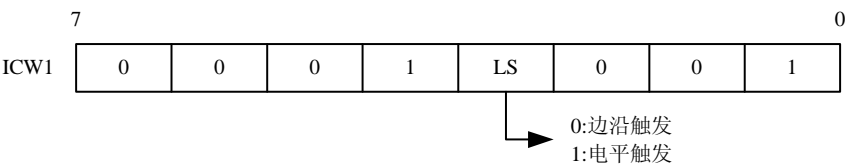


图 4.2.2 初始化命令字 1 寄存器

初始化命令字 2 寄存器 (ICW2) 说明见图 4.2.3 所示。

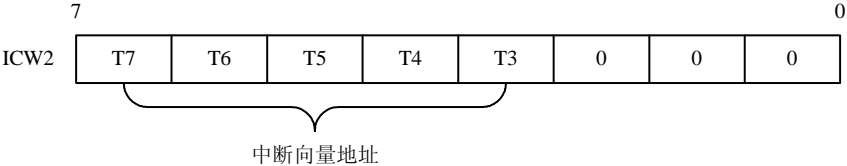


图 4.2.3 初始化命令字 2 寄存器

初始化命令字 3 寄存器 (ICW3) 说明，主片见图 4.2.4，从片见图 4.2.5。

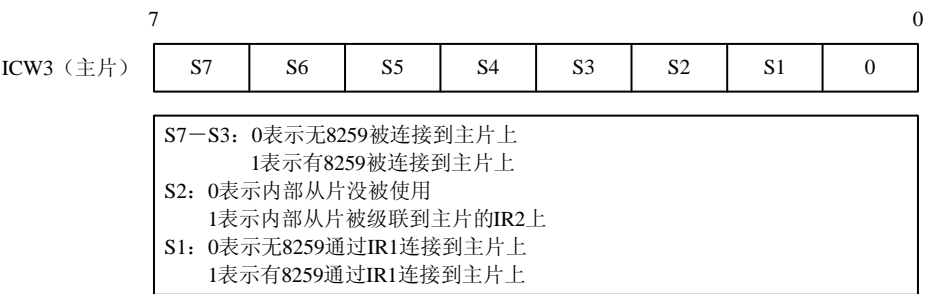


图 4.2.4 主片初始化命令字 3 寄存器

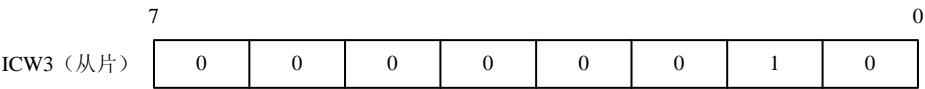


图 4.2.5 从片初始化命令字 3 寄存器

初始化命令字 4 寄存器 (ICW4) 说明见图 4.2.6。

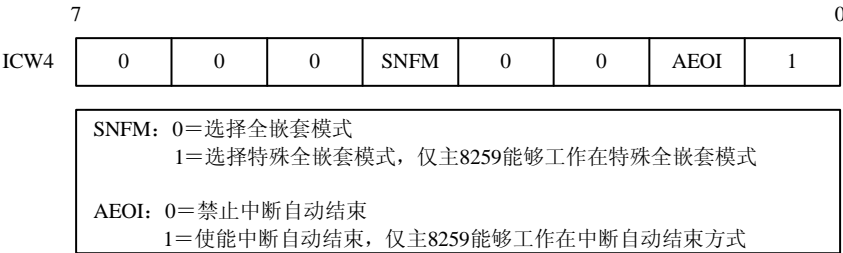


图 4.2.6 初始化命令字 4 寄存器

操作命令字 1 寄存器 (OCW1) 说明见图 4.2.7。

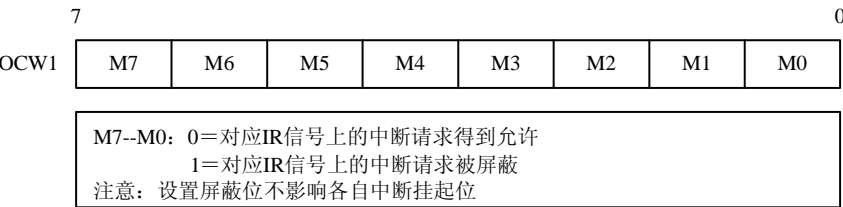


图 4.2.7 操作命令字 1 寄存器

操作命令字 2 寄存器 (OCW2) 说明如图 4.2.8 所示。

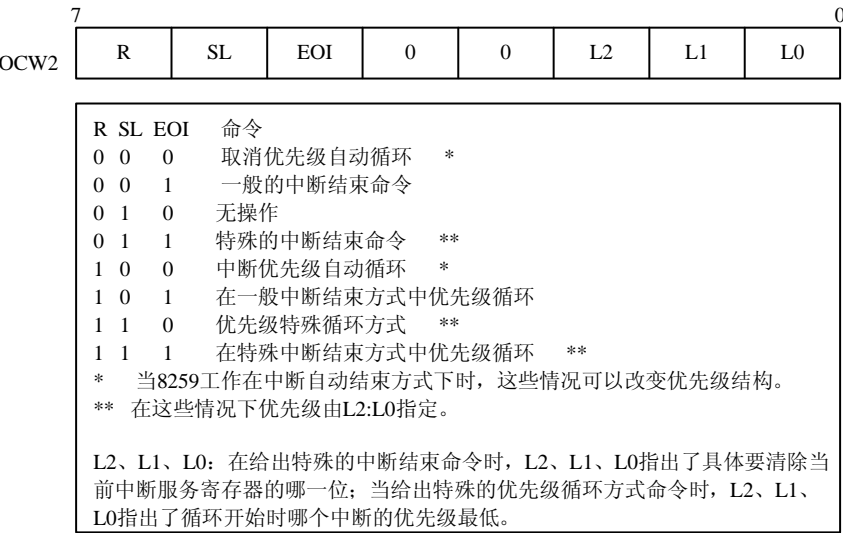


图 4.2.8 操作命令字 2 寄存器

操作命令字 3 寄存器 (OCW3) 说明如图 4.2.9 所示。



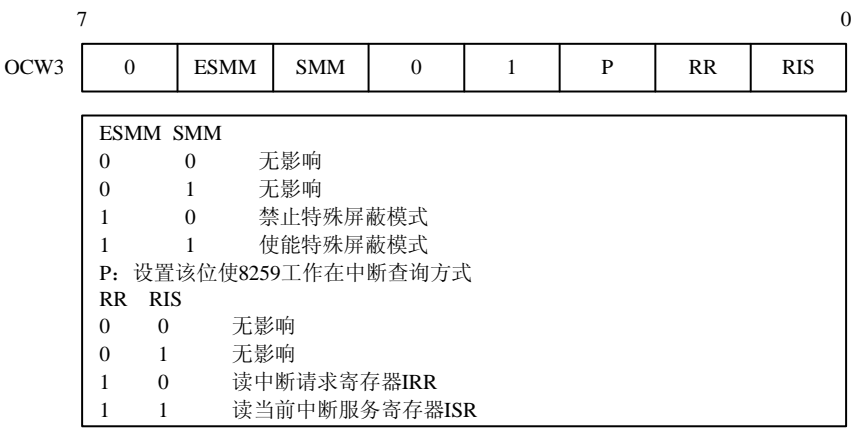


图 4.2.9 操作命令字 3 寄存器

查询状态字 (POLL) 说明如图 4.2.10 所示。

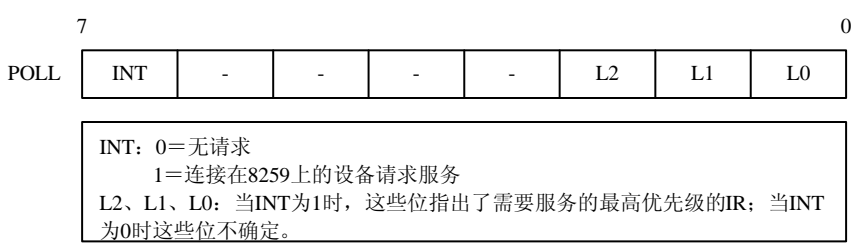


图 4.2.10 程序状态字寄存器

在对 8259 进行编程时, 首先必须进行初始化。一般先使用 CLI 指令将所有的可屏蔽中断禁止, 然后写入初始化命令字。8259 有一个状态机控制对寄存器的访问, 不正确的初始化顺序会造成异常初始化。在初始化主片 8259 时, 写入初始化命令字的顺序是: ICW1、ICW2、ICW3、然后是 ICW4, 初始化从片 8259 的顺序与初始化主片 8259 的顺序是相同的。

系统启动时, 主片 8259 已被初始化, 且 4 号中断源 (IR4) 提供给与 PC 联机的串口通信使用, 其它中断源被屏蔽。中断矢量地址与中断号之间的关系如下表所示:

主片中断序号	0	1	2	3	4	5	6	7
功能调用	08H	09H	0AH	0BH	0CH	0DH	0EH	0FH
矢量地址	20H~23H	24H~27H	28H~2BH	2CH~2FH	30H~33H	34H~37H	38H~3BH	3CH~3FH
说明	未开放	未开放	未开放	未开放	串口	未开放	可用	可用
从片中断序号	0	1	2	3	4	5	6	7
功能调用	30H	31H	32H	33H	34H	35H	36H	37H
矢量地址	C0H~C3H	C4H~C7H	C8H~CBH	CCH~CFH	D0H~D3H	D4H~D7H	D8H~DBH	DCH~DFH
说明	未开放	可用	未开放	未开放	未开放	未开放	未开放	未开放

### 4.2.5 实验步骤

#### 1. 8259 单中断实验

实验接线图如图 4.2.11 所示，单次脉冲输出与主片 8259 的 IR7 相连，每按动一次单次脉冲，产生一次外部中断，在显示屏上输出一个字符“7”。

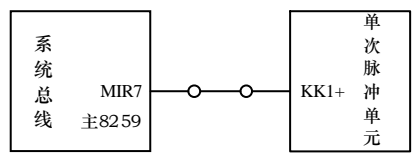



图 4.2.11 8259 单中断实验接线图

#### 实验步骤

- (1) 按图 4.2.11 连接实验线路。
- (2) 编写实验程序（例程文件名为：A82591.ASM），经编译、链接无误后装入系统。
- (3) 单击  按钮，运行实验程序，重复按单次脉冲开关 KK1+，在界面的输出区会显示字符“7”，说明响应了中断。实验现象结果如图 4.2.12 所示。

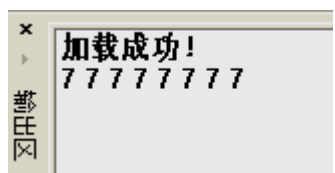


图 4.2.12 8259 单中断实验结果图

#### 实验程序清单

```
=====
; 文件名: A82591.ASM
; 功能描述: 8259 中断实验，中断源为主片 8259 的 IRQ7
;           每产生一次中断输出显示一个字符 7
;
=====

SSTACK    SEGMENT STACK
            DW 32 DUP(?)
SSTACK    ENDS
CODE      SEGMENT
            ASSUME CS:CODE
START:    PUSH DS
            MOV AX, 0000H
```

```

MOV DS, AX
MOV AX, OFFSET MIR7      ;取中断入口地址
MOV SI, 003CH             ;中断矢量地址
MOV [SI], AX              ;填 IRQ7 的偏移矢量
MOV AX, CS                ;段地址
MOV SI, 003EH
MOV [SI], AX              ;填 IRQ7 的段地址矢量
CLI
POP DS
;初始化主片 8259
MOV AL, 11H
OUT 20H, AL               ;ICW1
MOV AL, 08H
OUT 21H, AL               ;ICW2
MOV AL, 04H
OUT 21H, AL               ;ICW3
MOV AL, 01H
OUT 21H, AL               ;ICW4
MOV AL, 6FH               ;OCW1  0110 1111  开放 4 号中断串口用,
7 号中断实验用
OUT 21H, AL
STI
AA1:  NOP
      JMP AA1
MIR7: STI
      CALL DELAY
      MOV AX, 0137H
      INT 10H                ;显示字符 7
      MOV AX, 0120H
      INT 10H
      MOV AL, 20H
      OUT 20H, AL            ;中断结束命令
      IRET
DELAY: PUSH CX
      MOV CX, 0F00H
AA0:  PUSH AX
      POP  AX
      LOOP AA0
      POP CX

```

```
RET
CODE ENDS
END START
```

2. 8259 双中断优先级实验


实验接线图如图 4.2.13 所示，KK1+和 KK2+分别连接到主片 8259 的 IR7 和 IR6 上，当按一次 KK1+时，显示屏上显示字符“7”，按一次 KK2+时，显示字符“6”。编写程序。



图 4.2.13 8259 单中断实验接线图

实验步骤

- (1) 按图 4.2.13 连接实验线路。
- (2) 编写实验程序（例程文件名为：A82592.ASM），经编译、链接无误后装入系统。

(3) 单击  按钮，运行实验程序，重复按单次脉冲开关 KK1+和 KK2+，在界面的输出区会显示字符“7”和“6”，说明响应了中断。

(4) 尝试先按 KK1+,再快速按 KK2+，观察 MIR7 和 MIR6 两个中断请求的优先级，分析实验结果。实验结果现象如图 4.2.14 所示。

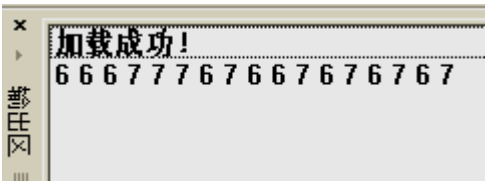


图 4.2.14 8259 双中断优先级实验结果图

实验程序清单

```
=====
; 文件名: A82592.ASM
; 功能描述: 8259 中断优先级应用实验
=====

SSTACK SEGMENT STACK
        DW 32 DUP(?)
SSTACK ENDS
```

```
CODE      SEGMENT
          ASSUME CS:CODE
START:    PUSH DS
          MOV AX, 0000H
          MOV DS, AX
          MOV AX, OFFSET MIR7      ;取中断入口地址
          MOV SI, 003CH            ;中断矢量地址
          MOV [SI], AX             ;填 IRQ7 的偏移矢量
          MOV AX, CS               ;段地址
          MOV SI, 003EH
          MOV [SI], AX            ;填 IRQ7 的段地址矢量
          MOV AX, OFFSET MIR6
          MOV SI, 0038H
          MOV [SI], AX
          MOV AX, CS
          MOV SI, 003AH
          MOV [SI], AX
          CLI
          POP DS
          ;初始化主片 8259
          MOV AL, 11H
          OUT 20H, AL              ;ICW1
          MOV AL, 08H
          OUT 21H, AL              ;ICW2
          MOV AL, 04H
          OUT 21H, AL              ;ICW3
          MOV AL, 01H
          OUT 21H, AL              ;ICW4
          MOV AL, 2FH
          OUT 21H, AL              ;主 8259 OCW1
          STI
AA1:      NOP
          JMP AA1
MIR7:     STI
          CALL DELAY
          MOV AX, 0137H
          INT 10H                  ;显示字符 7
          MOV AX, 0120H
```

```

        INT 10H
        MOV AL, 20H
        OUT 20H, AL           ;中断结束命令
        IRET
MIR6:   STI
        CALL DELAY
        MOV AX, 0136H
        INT 10H               ;显示字符 6
        MOV AX, 0120H
        INT 10H
        MOV AL, 20H
        OUT 20H, AL
        IRET
DELAY:  PUSH CX
        MOV CX, 0F000H
AA0:    PUSH AX
        POP  AX
        LOOP AA0
        POP CX
        RET
CODE   ENDS
        END   START

```

### 3. 8259 级连中断实验

实验接线图如图 4.2.15 所示, KK1+ 连接到主片 8259 的 IR7 上, KK2+ 连接到从片 8259 的 IR1 上, 当按一次 KK1+ 时, 显示屏上显示字符 “M7”, 按一次 KK2+ 时, 显示字符 “S1”。编写程序。

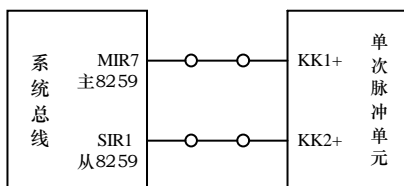



图 4.2.15 8259 级连实验接线图

#### 实验步骤

- (1) 按图 4.2.15 连接实验线路。
- (2) 输入程序（例程文件名为：A82593.ASM），编译、链接无误后装入系统。
- (3) 单击  按钮，运行实验程序，重复按单次脉冲开关 KK1+ 和 KK2+，在界面的输出

区会显示字符“M7”和“S1”，说明响应了中断，验证实验程序的正确性。

(4) 尝试先按 KK1+, 再快速按 KK2+, 观察 MIR7 和 SIR1 两个级连中断请求的优先级, 分析实验结果。实验结果现象如图 4.2.16 所示。

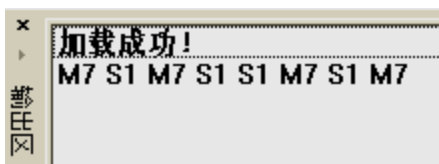


图 4.2.16 8259 级连实验结果图

#### 实验程序清单

```

;=====
; 文件名: A82593.ASM
; 功能描述: 8259 级联中断实验, 中断源为主片 8259 的 IR7,
;           从片 8259 的 IR1。从片 8259 通过主片 8259 的 IR2
;           进行级联。
;           主片每产生一次中断输出显示一个字符 M7, 从片
;           每产生一次中断输出显示一个字符 S1。
;=====

```

```

SSTACK    SEGMENT STACK
            DW 32 DUP(?)
SSTACK    ENDS
CODE      SEGMENT
            ASSUME CS:CODE
START:    PUSH DS
            MOV AX, 0000H
            MOV DS, AX
            MOV AX, OFFSET MIR7    ;取中断入口地址
            MOV SI, 003CH          ;中断矢量地址
            MOV [SI], AX           ;填 IRQ7 的偏移矢量
            MOV AX, CS              ;段地址
            MOV SI, 003EH
            MOV [SI], AX           ;填 IRQ7 的段地址矢量
            MOV AX, OFFSET SIR1
            MOV SI, 00C4H
            MOV [SI], AX
            MOV AX, CS

```

```
MOV SI, 00C6H
MOV [SI], AX
CLI
POP DS
;初始化主片 8259
MOV AL, 11H
OUT 20H, AL                ;ICW1
MOV AL, 08H
OUT 21H, AL                ;ICW2
MOV AL, 04H
OUT 21H, AL                ;ICW3
MOV AL, 01H
OUT 21H, AL                ;ICW4
;初始化从片 8259
MOV AL, 11H
OUT 0A0H, AL               ;ICW1
MOV AL, 30H
OUT 0A1H, AL               ;ICW2
MOV AL, 02H
OUT 0A1H, AL               ;ICW3
MOV AL, 01H
OUT 0A1H, AL               ;ICW4
MOV AL, 0FDH
OUT 0A1H, AL               ;OCW1 = 1111 1101
MOV AL, 6BH
OUT 21H, AL                ;主 8259 OCW1
STI
AA1:  NOP
      JMP AA1
MIR7: STI
      CALL DELAY
      MOV AX, 014DH
      INT 10H                ;显示字符 M
      MOV AX, 0137H
      INT 10H                ;显示字符 7
      MOV AX, 0120H
      INT 10H
      MOV AL, 20H
      OUT 20H, AL            ;中断结束命令
```



```
        IRET
SIR1:   STI
        CALL DELAY
        MOV AX, 0153H
        INT 10H                ;显示字符 S
        MOV AX, 0131H
        INT 10H                ;显示字符 1
        MOV AX, 0120H
        INT 10H
        MOV AL, 20H
        OUT 0A0H, AL
        OUT 20H, AL
        IRET
DELAY:  PUSH CX
        MOV CX, 0F00H
AA0:    PUSH AX
        POP  AX
        LOOP AA0
        POP CX
        RET
CODE    ENDS
        END  START
```

## 4.3 8255 并行接口实验

### 4.3.1 实验目的

1. 学习并掌握 8255 的工作方式及其应用。
2. 掌握 8255 典型应用电路的接法。
3. 掌握程序固化及脱机运行程序的方法。

### 4.3.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 4.3.3 实验内容

1. 基本输入输出实验。编写程序，使 8255 的 A 口为输出，B 口为输入，完成拨动开关到数据灯的数据传输。要求只要开关拨动，数据灯的显示就发生相应改变。
2. 流水灯显示实验。编写程序，使 8255 的 A 口和 B 口均为输出，数据灯 D7~D0 由左向右，每次仅亮一个灯，循环显示，D15~D8 与 D7~D0 正相反，由右向左，每次仅点亮一个灯，循环显示。
3. 方式 1 输入输出实验。编写程序，使 8255 工作在方式 1 控制下的 A 口输入，B 口输出。

4.3.4 实验原理

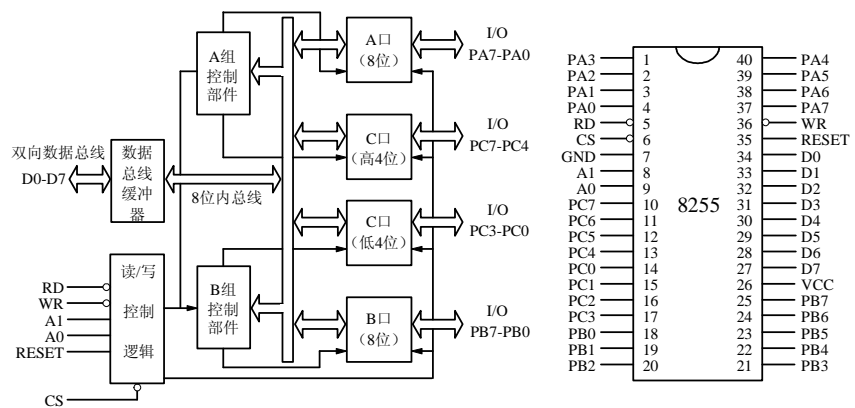


图 4.3.1 8255 内部结构及外部引脚图

并行接口是以数据的字节为单位与 I/O 设备或被控制对象之间传递信息。CPU 和接口之间的数据传送总是并行的，即可以同时传递 8 位、16 位或 32 位等。8255 可编程外围接口芯片是 Intel 公司生产的通用并行 I/O 接口芯片，它具有 A、B、C 三个并行接口，用+5V 单电源供电，能在以下三种方式下工作：方式 0--基本输入/输出方式、方式 1--选通输入/输出方式、方式 2--双向选通工作方式。8255 的内部结构及引脚如图 4.3.1 所示，8255 工作方式控制字和 C 口按位置位/复位控制字格式如图 4.3.2 所示。

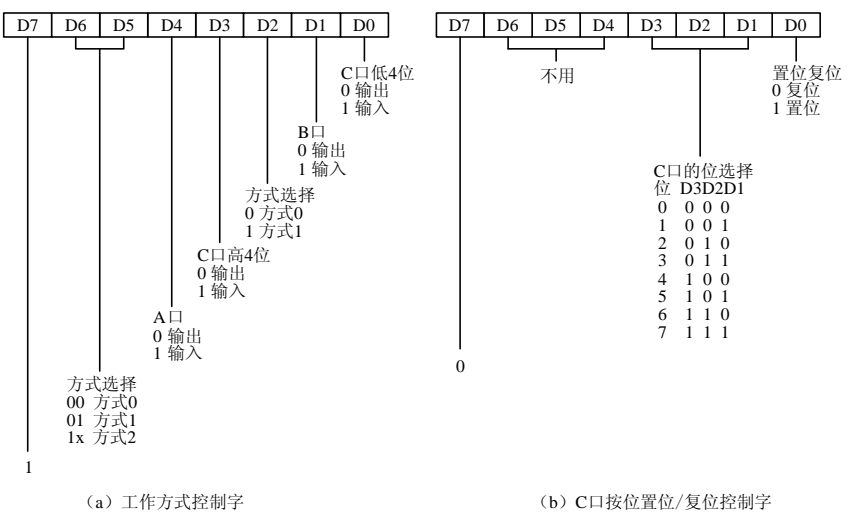


图 4.3.2 8255 控制字格式

8255 实验单元电路图如图 4.3.3 所示：

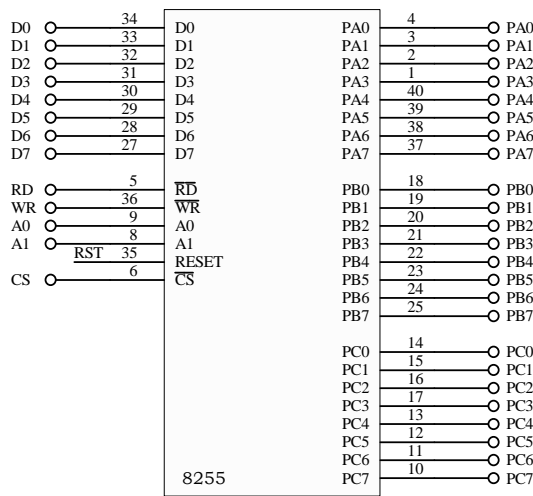


图 4.3.3 8255 实验单元电路图

### 4.3.5 实验步骤

#### 1. 基本输入输出实验

本实验使 8255 端口 A 工作在方式 0 并作为输出口，端口 B 工作在方式 0 并作为输入口。用一组开关信号接入端口 B，端口 A 输出线接至一组数据灯上，然后通过对 8255 芯片编程来实现输入输出功能。具体实验步骤如下，其中第（4）步到第（6）步固化功能可选作：

- （1）实验接线图如图 4.3.4 所示，按图连接实验线路图。
- （2）编写实验程序（例程文件名为：A82551.ASM），经编译、连接无误后装入系统。
- （3）运行程序，改变拨动开关，同时观察 LED 显示，验证程序功能。
- （4）点击“调试”下拉菜单中的“固化程序”项，将程序固化到系统存储器中。
- （5）将 386EX 单板机系统的短路跳线 JDBG 短接到 RUN 端，然后按复位按键，观察程序是否正常运行；关闭实验箱电源，稍等后再次打开电源，看固化的程序是否运行，验证程序功能。
- （6）实验完毕后，请将短路跳线 JDBG 的短路块短接到 DBG 端，以方便下次联机实验。

小提示：I386EX CPU 单板机支持联机调试模式和脱机独立运行模式。两种模式的切换是通过 I386EX CPU 单板机单元的右下角下层基板处的短路跳线 JDBG 来实现，短路块短接到 DBG 档，CPU 与软件处于联机调试模式，该模式下，通过软件界面可对 CPU 进行下载程序，单步、断点、连续运行等调试，通过固化功能菜单，可将加载到 CPU 单板机存储器中的程序固化到 FLASH 存储器中。固化完成后，将短路块短接到 RUN 档，并复位或另加电，CPU 将启动 FLASH 存储器中的程序进行独立运行，此时 I386EX CPU 单板机就工作在脱机独立运行模式了。

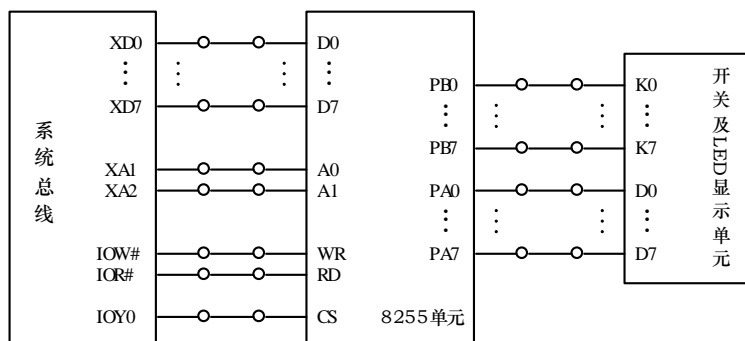


图 4.3.4 8255 基本输入输出实验接线图

## 实验程序清单

```

=====
; 文件名: A82551.ASM
; 功能描述: A 口为输入, B 口为输出, 将读入的数据输出显示
;          IOY0
=====

```

```

IOY0      EQU    0600H          ;片选 IOY0 对应的端口始地址
MY8255_A   EQU    IOY0+00H*2    ;8255 的 A 口地址
MY8255_B   EQU    IOY0+01H*2    ;8255 的 B 口地址
MY8255_C   EQU    IOY0+02H*2    ;8255 的 C 口地址
MY8255_MODE EQU    IOY0+03H*2   ;8255 的控制寄存器地址

```

```

SSTACK    SEGMENT STACK
            DW 32 DUP(?)
SSTACK     ENDS
CODE       SEGMENT
            ASSUME CS:CODE
START:     MOV DX, MY8255_MODE
            MOV AL, 82H
            OUT DX, AL
AA1:       MOV DX, MY8255_B
            IN  AL, DX
            CALL DELAY
            MOV DX, MY8255_A
            OUT DX, AL
            JMP AA1

```

```

DELAY: PUSH CX
      MOV CX, 0F00H
AA2:  PUSH AX
      POP  AX
      LOOP AA2
      POP  CX
      RET
CODE  ENDS
      END  START
    
```

## 2. 流水灯显示实验

使 8255 的 A 口和 B 口均为输出，数据灯 D7~D0 由左向右，每次仅亮一个灯，循环显示，D15~D8 与 D7~D0 正好相反，由右向左，每次仅点亮一个灯，循环显示。实验接线图如图 4.3.5 所示。实验步骤如下所述：

- (1) 按图 4.3.5 连接实验线路图。
- (2) 编写实验程序（例程文件名为：A82552.ASM），经编译、链接无误后装入系统。
- (3) 运行程序，观察 LED 灯的显示，验证程序功能。
- (4) 自己改变流水灯的方式，编写程序。
- (5) 固化程序并脱机运行（可选做）。

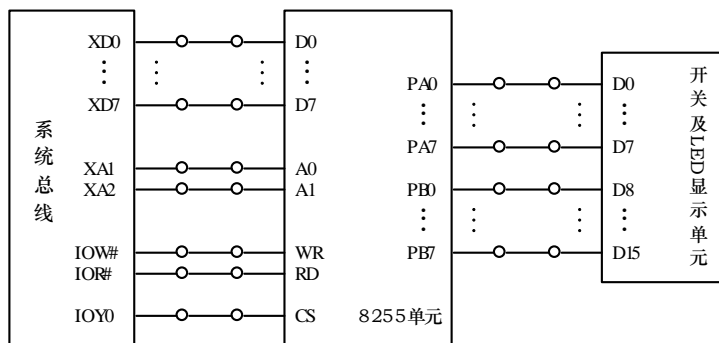


图 4.3.5 8255 流水灯实验接线图

## 实验程序清单

```

=====
; 文件名: A82552.ASM
; 功能描述: A 口为输出, B 口为输出, 流水灯显示
=====
    
```

```

IOY0      EQU    0600H      ;片选 IOY0 对应的端口始地址
MY8255_A   EQU    IOY0+00H*2 ;8255 的 A 口地址
    
```

```

MY8255_B    EQU    IOY0+01H*2    ;8255 的 B 口地址
MY8255_C    EQU    IOY0+02H*2    ;8255 的 C 口地址
MY8255_MODE EQU    IOY0+03H*2    ;8255 的控制寄存器地址

```

```

SSTACK    SEGMENT STACK
            DW 32 DUP(?)
SSTACK    ENDS
CODE SEGMENT
            ASSUME CS:CODE
START: MOV DX, MY8255_MODE
            MOV AL, 80H
            OUT DX, AL
            MOV BX, 8001H
AA1:  MOV DX, MY8255_A
            MOV AL, BH
            OUT DX, AL
            MOV DX, MY8255_B
            MOV AL, BL
            OUT DX, AL
            ROL BL, 1
            CALL DELAY
            CALL DELAY
            JMP AA1
DELAY: PUSH CX
            MOV CX, 0F000H
AA2:  PUSH AX
            POP  AX
            LOOP AA2
            POP  CX
            RET
CODE ENDS
            END  START

```

### 3. 方式 1 输入输出实验

本实验使 8255 端口 A 工作在方式 0 并作为输出口，端口 B 工作在方式 1 并作为输入口，则端口 C 的 PC2 成为选通信号输入端 STBB，PC0 成为中断请求信号输出端 INTRB。当 B 口数据就绪后，通过发 STBB 信号来请求 CPU 读取端口 B 数据并送端口 A 输出显示。用一组开关信号接入端口 B，端口 A 输出线接至一组数据灯上。具体实验步骤如下：

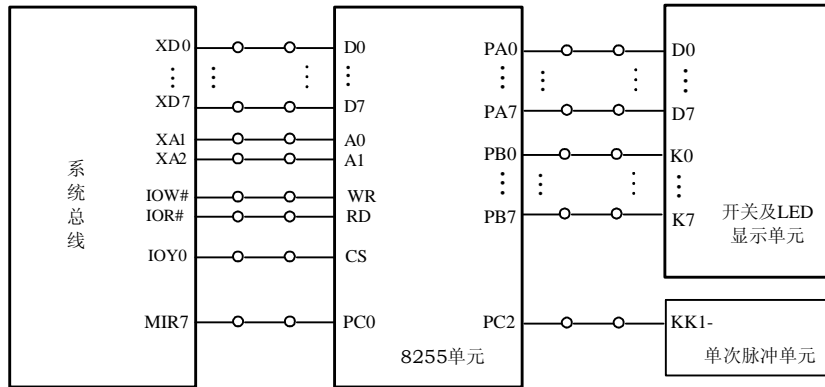


图 4.3.6 8255 方式 1 输入输出实验接线图

- (1) 按图 4.3.6 连接实验线路图。
- (2) 编写实验程序（例程文件名为：A82553.ASM），经编译、链接无误后装入系统。
- (3) 运行程序，然后改变拨动开关，准备好后，按动 KK1，同时观察数据灯显示，应与开关组信号一致。

#### 实验程序清单

```

=====
; 文件名: A82553.ASM
; 功能描述: 本实验使 8255 端口 A 工作在方式 0 并作为输出口,
;           端口 B 工作在方式 1 并作为输入口
=====

IOY0      EQU    0600H          ;片选 IOY0 对应的端口始地址
MY8255_A   EQU    IOY0+00H*2     ;8255 的 A 口地址
MY8255_B   EQU    IOY0+01H*2     ;8255 的 B 口地址
MY8255_C   EQU    IOY0+02H*2     ;8255 的 C 口地址
MY8255_MODE EQU    IOY0+03H*2    ;8255 的控制寄存器地址

STACK1     SEGMENT STACK
            DW 256 DUP(?)
STACK1     ENDS
CODE       SEGMENT
            ASSUME CS:CODE
START:     MOV DX,MY8255_MODE      ;初始化 8255 工作方式
            MOV AL,86H             ;工作方式 1, A 口输出, B 口输入
            OUT DX,AL
    
```



```

MOV DX,MY8255_MODE           ;C 口 PC2 置位
MOV AL,05H
OUT DX,AL
PUSH DS
MOV AX, 0000H
MOV DS, AX
MOV AX, OFFSET MIR7           ;取中断入口地址
MOV SI, 003CH                 ;中断矢量地址
MOV [SI], AX                  ;填 IRQ7 的偏移矢量
MOV AX, CS                    ;段地址
MOV SI, 003EH
MOV [SI], AX                  ;填 IRQ7 的段地址矢量
CLI
POP DS
;初始化主片 8259
MOV AL, 11H
OUT 20H, AL                   ;ICW1
MOV AL, 08H
OUT 21H, AL                   ;ICW2
MOV AL, 04H
OUT 21H, AL                   ;ICW3
MOV AL, 01H
OUT 21H, AL                   ;ICW4
MOV AL, 6FH                   ;OCW1
OUT 21H, AL
STI
AA1:  NOP
      JMP AA1
MIR7:  PUSH AX
      MOV DX,MY8255_B           ;读 B 口
      IN  AL,DX
      MOV DX,MY8255_A           ;写 A 口
      OUT DX,AL
      MOV AL,20H
      OUT 20H,AL
      POP AX
      IRET
DELAY: PUSH CX
      MOV CX, 0F00H

```

```
AA0:  PUSH AX
      POP  AX
      LOOP AA0
      POP CX
      RET
CODE  ENDS
      END START
```

## 4.4 DMA 特性及 8237 应用实验

### 4.4.1 实验目的

1. 掌握 8237DMA 控制器的工作原理。
2. 了解 DMA 特性及 8237 的几种数据传输方式。
3. 掌握 8237 的应用编程。

### 4.4.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 4.4.3 实验内容

1. 将存储器 1000H 单元开始的连续 10 个字节的数据复制到地址 0000H 开始的 10 个单元中，实现 8237 的存储器到存储器传输。
2. I/O 到存储器 DMA 传输实验。利用 8237、8255 和扩展存储器单元，设计一个 DMA 传输，将 8255 读并行接口数据传输到扩展存储器中。
3. 存储器到 I/O DMA 传输实验。利用 8237、8255 和扩展存储器单元，设计一个 DMA 传输，将扩展存储器中数据传输到 8255 写并行接口。

### 4.4.4 实验原理

直接存储器访问 (Direct Memory Access, 简称 DMA)，是指外部设备不经过 CPU 的干涉，直接实现对存储器的访问。DMA 传送方式可用来实现存储器到存储器、I/O 接口到存储器、存储器到 I/O 接口之间的高速数据传送。

1. 8237 芯片介绍

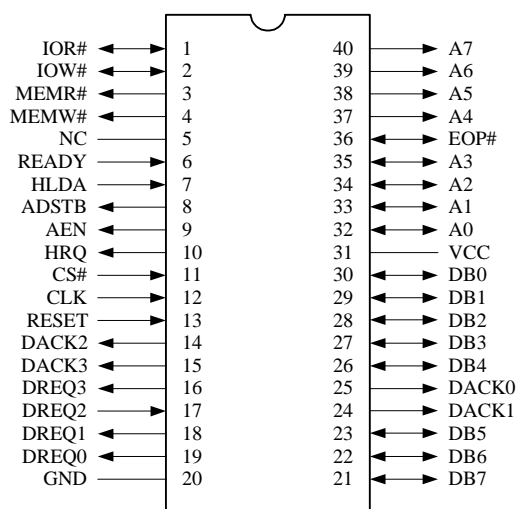


图 4.4.1 8237 外部引脚图

8237 是一种高性能可编程 DMA 控制器，芯片有 4 个独立的 DMA 通道，可用来实现存储器到存储器、存储器到 I/O 接口、I/O 接口到存储器之间的高速数据传送。8237 的各通道均具有相应的地址、字数、方式、命令、请求、屏蔽、状态和暂存寄存器，通过对它们的编程，可实现 8237 初始化，以确定 DMA 控制的工作类型、传输类型、优先级控制、传输定时控制及工作状态等。8237 的外部引脚如图 4.4.1 所示。

8237 的内部寄存器分为两类：

4 个通道共用的寄存器。包括命令、方式、状态、请求、屏蔽和暂存寄存器。4 个通道专用的寄存器。包括地址寄存器（基地址及当前地址寄存器）和字节计数器（基本字节计数器和当前字节计数器）。

8237 的内部结构图如图 4.4.2 所示。

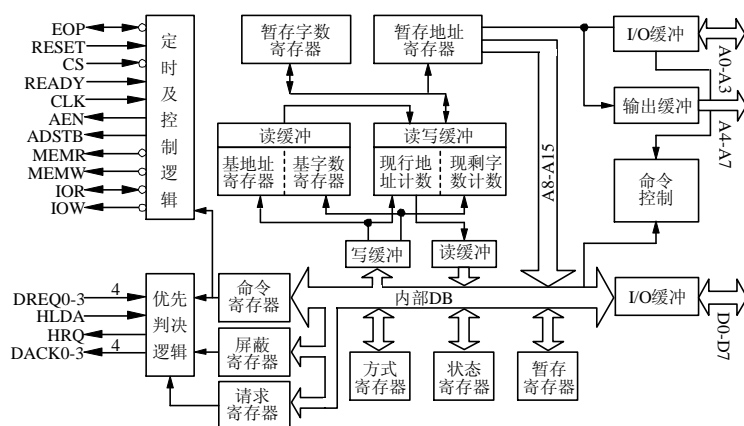


图 4.4.2 8237 内部结构图

寄存器格式如图 4.4.3~图 4.4.7 所示。

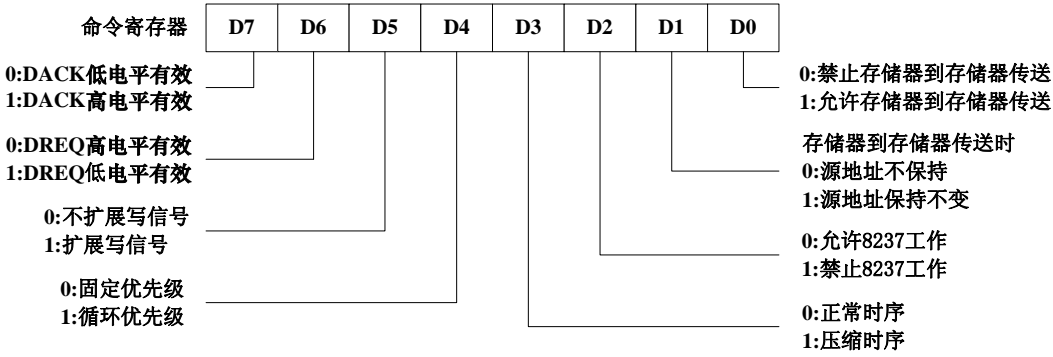


图 4.4.3 命令寄存器格式

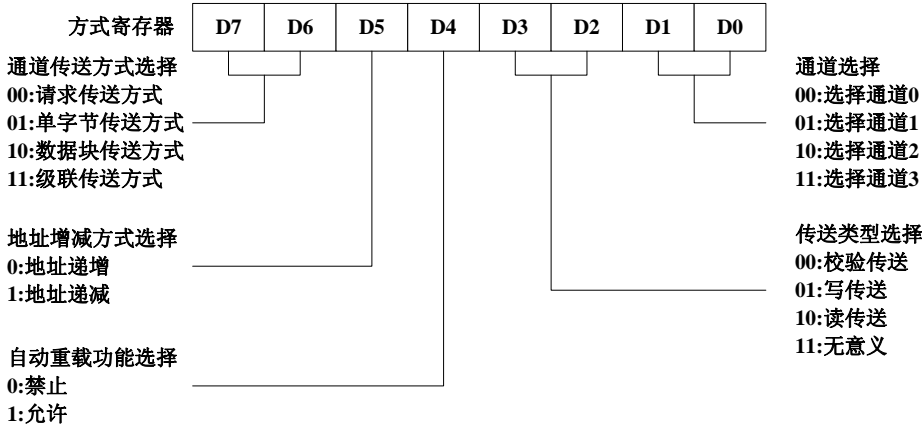


图 4.4.4 方式寄存器格式

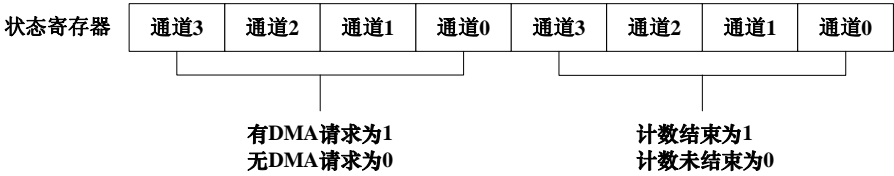


图 4.4.5 8237 状态寄存器

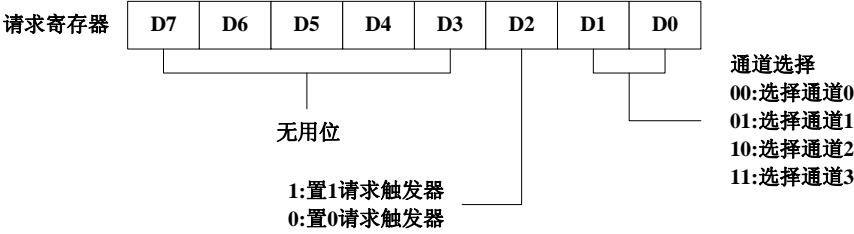


图 4.4.6 8237 请求寄存器格式

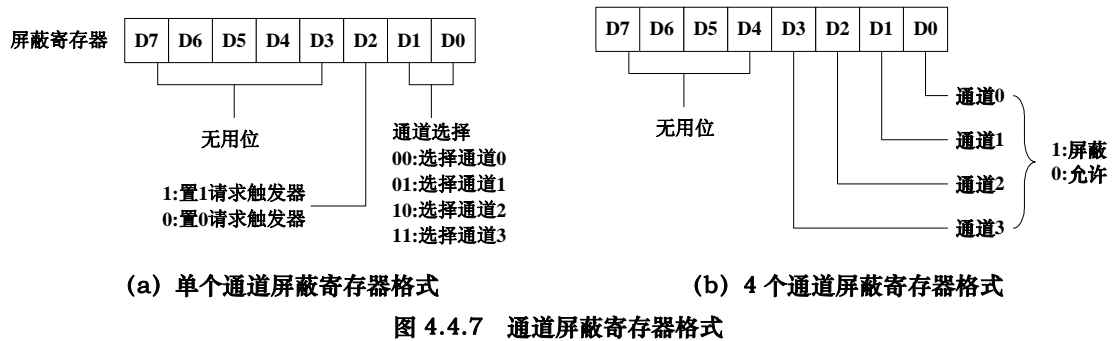


图 4.4.7 通道屏蔽寄存器格式

表 4.4.1 列出了 8237 内部寄存器和软命令及其操作信息。

表 4.4.1 8237 内部寄存器和软命令及其读写操作一览表

寄存器名		位长	操作	片选逻辑 (CS # =0)				对 应 端口号	先/后 触发器	操作字节						
				IOR#	IOR#	A3	A2				A1	A0				
基地址寄存器 (4 个)		16	写	1	0	0	A2	A1	0							
当前地址寄存器 (4 个)		16	写	0	1	通道选择		0H	0	低 8 位						
			读					2H	1	高 8 位						
								4H	0	低 8 位						
								8H	1	高 8 位						
基字节数寄存器(4 个)		16	写	1	0	通道选择		0	A2	A1	1					
当前字节数寄存器 (4 个)		16	写					1H	0	低 8 位						
								3H	1	高 8 位						
			读					5H	0	低 8 位						
				7H	1	高 8 位										
命令寄存器		8	写	1	0	1	0	0	0	8H	-	-				
状态寄存器		8	读	0	1											
请求寄存器		4	写	1	0					1			0	0	1	9H
写单个屏蔽位寄存器		4	写	1	0					1			0	1	0	AH
方式寄存器 (4 个)		6	写	1	0	1	0	1	1	BH						
暂存寄存器		8	读	0	1	1	1	0	1	DH						
软命令	主清除	-	写	1	0											
	清先/后触发器	-	写	1	0					1			1	0	0	CH
	清屏蔽寄存器	-	写	1	0					1			1	1	0	EH
写 4 通道屏蔽位寄存器		4	写	1	0	1	1	1	1	FH						
地址暂存寄存器		16	与 CPU 不直接发生关系													
字节数暂存寄存器		16														

2. DMA 实验单元电路图、存储器译码单元电路图

实验系统中提供的 8237 单元电路原理如图 4.4.8。

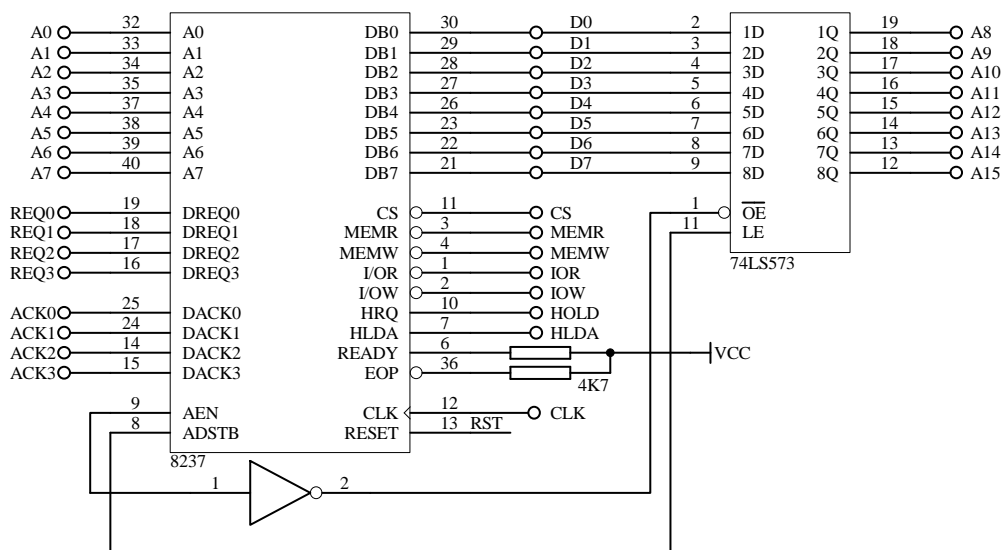


图 4.4.8 DMA 实验单元电路图

实验系统的系统总线单元提供了 MY0 和 MY1 两个存储器译码信号，译码空间分别为 80000H~9FFFFH 和 A0000H~BFFFFH。在做 DMA 实验时，CPU 会让出总线控制权，而 8237 的寻址空间仅为 0000H~FFFFH，8237 无法寻址到 MY0 的译码空间，故系统中将高位地址线 A19~A17 连接到固定电平上，在 CPU 让出总线控制权时，MY0 会变为低电平，即 DMA 访问期间，MY0 有效。具体如下图所示。

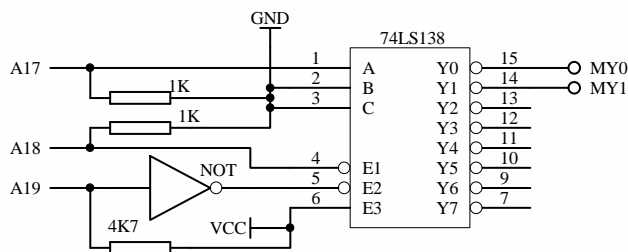


图 4.4.9 存储器译码单元电路图

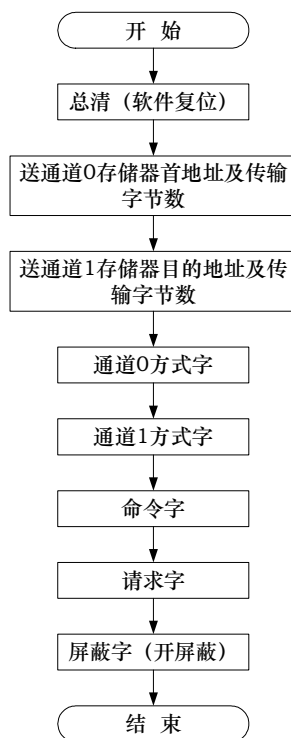


图 4.4.10 DMA 实验流程图

### 4.4.5 实验步骤

#### 1. 存储器到存储器 DMA 传输实验

将存储器 1000H 单元开始的连续 10 个字节的数据复制到地址 0000H 开始的 10 个单元中，实现 8237 的存储器到存储器传输。

(1) 根据实验要求，参考流程图 4.4.10 编写实验程序（例程文件名为：A82371.ASM）；实验接线如图 4.4.11 所示，按图连接实验线路。

(2) 编译、链接程序无误后，将目标代码装入系统。

(3) 初始化首地址中的数据，通过 E8000:2000 命令来改变。（注：思考为何通道中送入的首地址值为 1000H，而 CPU 初始化时的首地址为 2000H。）

```
E8000:2000=11
E8000:2002=22
E8000:2004=33
E8000:2006=44
E8000:2008=55
E8000:200A=66
```



E8000:200C=77  
E8000:200E=88  
E8000:2010=99  
E8000:2012=00

(4) 运行程序，待程序运行停止。

(5) 通过 D8000:0000 命令查看 DMA 传输结果，是否与首地址中写入的数据相同，可反复验证。

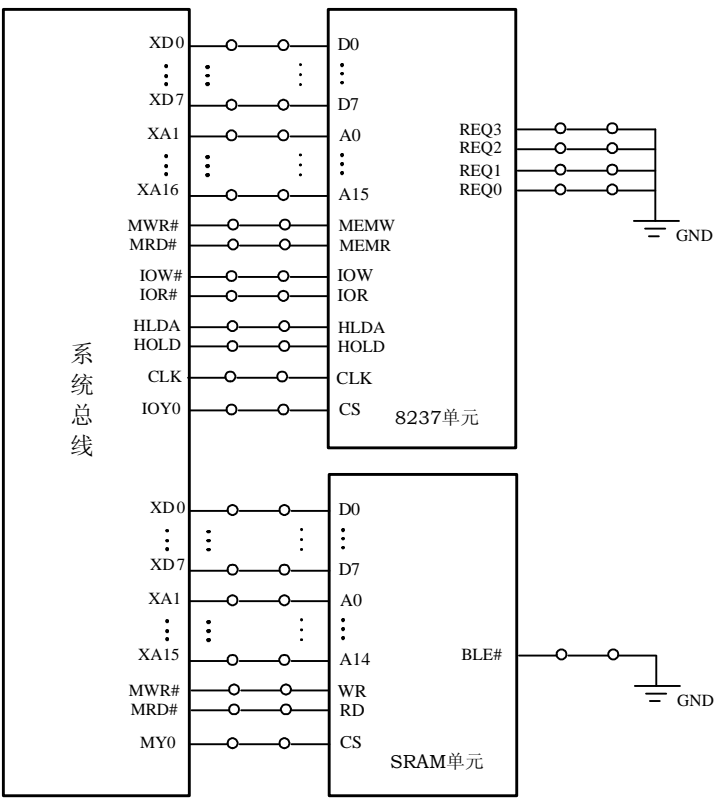


图 4.4.11 8237 实现存储器到存储器传输实验接线图

```
实验程序清单
;=====
; 文件名: A82371.ASM
; 功能描述: 8237DMA 传送实验，源地址为 1000H，目标地址为
;           0000H，通过 E 命令改变 8000:2000 连续 10 个存储单元的值，
;           运行程序后，通过 D8000:0000 查看传送结果。
;=====

IOY0      EQU    0600H      ;IOY0 起始地址
```

MY8237_0	EQU	IOY0+00H*2	;通道 0 当前地址寄存器
MY8237_1	EQU	IOY0+01H*2	;通道 0 当前字节计数寄存器
MY8237_2	EQU	IOY0+02H*2	;通道 1 当前地址寄存器
MY8237_3	EQU	IOY0+03H*2	;通道 1 当前字节计数寄存器
MY8237_8	EQU	IOY0+08H*2	;写命令寄存器/读状态寄存器
MY8237_9	EQU	IOY0+09H*2	;请求寄存器
MY8237_B	EQU	IOY0+0BH*2	;工作方式寄存器
MY8237_D	EQU	IOY0+0DH*2	;写总清命令/读暂存寄存器
MY8237_F	EQU	IOY0+0FH*2	;屏蔽位寄存器

STACK SEGMENT STACK

DW 64 DUP(?)

STACK ENDS

CODE SEGMENT

ASSUME CS:CODE

START: MOV AL, 00

MOV DX, MY8237\_D

OUT DX, AL ;写总清命令

AA1: MOV AL, 00H

MOV DX, MY8237\_0 ;写通道 0 当前地址寄存器

OUT DX,AL

MOV AL,10H

OUT DX,AL

MOV AL,00H

MOV DX, MY8237\_2 ;写通道 1 当前地址寄存器

OUT DX,AL

MOV AL,00H

OUT DX,AL

MOV AL,0AH

MOV DX, MY8237\_1 ;写通道 0 当前字节计数寄存器

OUT DX,AL

MOV AL,00H

OUT DX,AL

MOV AL,0AH

MOV DX, MY8237\_3 ;写通道 1 当前字节计数寄存器

OUT DX,AL

MOV AL,00H

OUT DX,AL

MOV AL,88H

```
MOV DX, MY8237_B      ;写通道 0 工作方式寄存器
OUT DX,AL
MOV AL,85H             ;写通道 1 工作方式寄存器
OUT DX,AL
MOV AL,81H
MOV DX, MY8237_8       ;写命令寄存器
OUT DX,AL
MOV AL,04H
MOV DX, MY8237_9       ;写请求寄存器
OUT DX,AL
MOV AL,00H
MOV DX, MY8237_F       ;写屏蔽位寄存器
OUT DX,AL
MOV AX,4C00H
INT 21H                ;程序终止
CODE ENDS
END START
```

## 2. I/O 到存储器 DMA 传输实验

在实验 1 基础上增加 8255 初始化为 B 口输入，A 口输出。B 口输入数据由拨动开关模拟。修改 8237 初始化方式，将 B 口所连接开关指示的数据传输到存储器相应的数据单元中。

(1) 实验接线如图 4.4.12 所示，按图连接实验线路。

(2) 编写实验程序（例程文件名为：A82372.ASM），编译、链接程序无误后，将目标代码装入系统。

(3) 拨动 8255 的 B 口所连开关组，设置好一个数据。

(4) 运行程序，待程序运行停止。

(5) 在“Memory”的地址栏输入“8000: 0642”，回车，查看 DMA 传输结果，是否与前面开关所设置数据相同，可反复验证。

注：本实验中，8255 使用 IOY1 地址空间，B 口的端口地址为 0642H，所以，数据传输到扩展存储器偏移为 0642H 地址单元。对于 8237 来讲，实际偏移地址为 0321H。想想看，是不是这样？

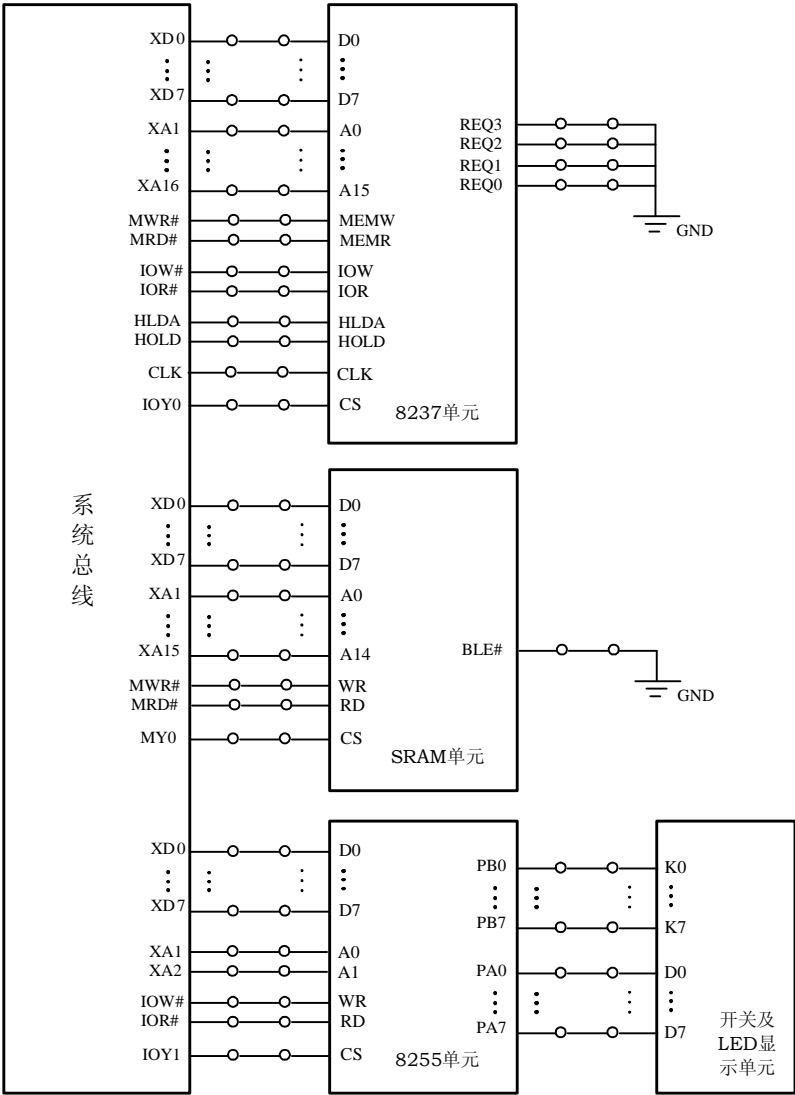


图 4.4.12 8237 实现存储器与 I/O 间 DMA 传输实验接线图

实验程序清单

```
=====
;
; 文件名: A82372.ASM
; 功能描述: 8237DMA 传送实验, IO-存储器
;
=====
```

```
IOY0      EQU 0600H      ;IOY0 起始地址
IOY1      EQU 0640H      ;IOY1 起始地址
```

MY8237_0	EQU	IOY0+00H*2	;通道 0 当前地址寄存器
MY8237_1	EQU	IOY0+01H*2	;通道 0 当前字节计数寄存器
MY8237_2	EQU	IOY0+02H*2	;通道 1 当前地址寄存器
MY8237_3	EQU	IOY0+03H*2	;通道 1 当前字节计数寄存器
MY8237_8	EQU	IOY0+08H*2	;写命令寄存器/读状态寄存器
MY8237_9	EQU	IOY0+09H*2	;请求寄存器
MY8237_B	EQU	IOY0+0BH*2	;工作方式寄存器
MY8237_D	EQU	IOY0+0DH*2	;写总清命令/读暂存寄存器
MY8237_F	EQU	IOY0+0FH*2	;屏蔽位寄存器
MY8255_A	EQU	IOY1+00H*2	;8255 的 A 口地址
MY8255_B	EQU	IOY1+01H*2	;8255 的 B 口地址
MY8255_C	EQU	IOY1+02H*2	;8255 的 C 口地址
MY8255_MODE	EQU	IOY1+03H*2	;8255 的控制寄存器地址

STACK1     SEGMENT STACK

    DW 256 DUP(?)

STACK1 ENDS

CODE SEGMENT

    ASSUME CS:CODE

START:     MOV DX,MY8255\_MODE

    MOV AL,82H

    OUT DX,AL

    MOV DX,MY8237\_D             ;写总清命令

    OUT DX,AL

    MOV DX,MY8237\_2             ;写通道 1 当前地址寄存器

    MOV AL,21H             ;总线地址是 0321H\*2=0642H

    OUT DX,AL

    MOV AL,03H

    OUT DX,AL

    MOV DX,MY8237\_B             ;写通道 1 工作方式寄存器

    MOV AL,45H

    OUT DX,AL

    MOV DX,MY8237\_8             ;写命令寄存器

    MOV AL,80H

    OUT DX,AL

    MOV DX,MY8237\_F             ;写屏蔽位寄存器

    MOV AL,00H

    OUT DX,AL

    MOV DX,MY8237\_9             ;写请求寄存器

```

MOV AL,05H
OUT DX,AL
QUIT: MOV AX,4C00H           ;结束程序退出
      INT 21H
CODE  ENDS
      END START

```

### 3. 存储器到 I/O DMA 传输实验

在实验 1 基础上增加 8255 初始化为 B 口输入，A 口输出。A 口输出数据连接到数码二极管组显示。修改 8237 初始化方式，将存储器相应数据单元中的数据传输到 A 口，由数码二极管组显示。

(1) 实验接线如图 4.4.12 所示，按图连接实验线路。

(2) 编写实验程序（例程文件名为：A82373.ASM），编译、链接程序无误后，将目标代码装入系统。

(3) 在“Memory”窗口中，设置地址“8000: 0640H”中的数据。

(4) 运行程序，待程序运行停止。

(5) 查看发光二极管组显示数据，是否与前面写入的数据相同，可反复验证。

注：本实验中，8255 使用 IOY1 地址空间，A 口的端口地址为 0640H，所以，我们设置是扩展存储单元中偏移为 0640H 的数据。对于 8237 来讲，实际偏移地址为 0320H。想想看，是不是这样？

#### 实验程序清单

```

;=====
; 文件名: A82373.ASM
; 功能描述: 8237DMA 传送实验，存储器-IO
;=====

```

IOY0	EQU	0600H	;IOY0 起始地址
IOY1	EQU	0640H	;IOY1 起始地址
MY8237_0	EQU	IOY0+00H*2	;通道 0 当前地址寄存器
MY8237_1	EQU	IOY0+01H*2	;通道 0 当前字节计数寄存器
MY8237_2	EQU	IOY0+02H*2	;通道 1 当前地址寄存器
MY8237_3	EQU	IOY0+03H*2	;通道 1 当前字节计数寄存器
MY8237_8	EQU	IOY0+08H*2	;写命令寄存器/读状态寄存器
MY8237_9	EQU	IOY0+09H*2	;请求寄存器
MY8237_B	EQU	IOY0+0BH*2	;工作方式寄存器
MY8237_D	EQU	IOY0+0DH*2	;写总清命令/读暂存寄存器
MY8237_F	EQU	IOY0+0FH*2	;屏蔽位寄存器
MY8255_A	EQU	IOY1+00H*2	;8255 的 A 口地址

```
MY8255_B    EQU  IOY1+01H*2    ;8255 的 B 口地址
MY8255_C    EQU  IOY1+02H*2    ;8255 的 C 口地址
MY8255_MODE EQU  IOY1+03H*2    ;8255 的控制寄存器地址
```

```
STACK1      SEGMENT STACK
```

```
    DW 256 DUP(?)
```

```
STACK1      ENDS
```

```
CODE SEGMENT
```

```
    ASSUME CS:CODE
```

```
START:      MOV DX,MY8255_MODE
```

```
    MOV AL,82H
```

```
    OUT DX,AL
```

```
    MOV DX,MY8237_D            ;写总清命令
```

```
    OUT DX,AL
```

```
    MOV DX,MY8237_0            ;写通道 0 当前地址寄存器
```

```
    MOV AL,20H                 ;总线地址是 0320H*2=0640H
```

```
    OUT DX,AL
```

```
    MOV AL,03H
```

```
    OUT DX,AL
```

```
    MOV DX,MY8237_B            ;写通道 0 工作方式寄存器
```

```
    MOV AL,48H
```

```
    OUT DX,AL
```

```
    MOV DX,MY8237_8            ;写命令寄存器
```

```
    MOV AL,80H
```

```
    OUT DX,AL
```

```
    MOV DX,MY8237_F            ;写屏蔽位寄存器
```

```
    MOV AL,00H
```

```
    OUT DX,AL
```

```
    MOV DX,MY8237_9            ;写请求寄存器
```

```
    MOV AL,04H
```

```
    OUT DX,AL
```

```
QUIT:      MOV AX,4C00H        ;结束程序退出
```

```
    INT 21H
```

```
CODE ENDS
```

```
    END START
```

## 4.5 8254 定时/计数器应用实验

### 4.5.1 实验目的

1. 掌握 8254 的工作方式及应用编程。
2. 掌握 8254 典型应用电路的接法。

### 4.5.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 4.5.3 实验内容

1. 计数应用实验。编写程序，应用 8254 的计数功能，使用单次脉冲模拟计数，使每当按动‘KK1+’5 次后，产生一次计数中断，并在屏幕上显示一个字符‘M’。
2. 定时应用实验。编写程序，应用 8254 的定时功能，产生一个 1s 的方波，并用本装置的示波器功能来观察。

### 4.5.4 实验原理

8254 是 Intel 公司生产的可编程间隔定时器。是 8253 的改进型，比 8253 具有更优良的性能。8254 具有以下基本功能：

- (1) 有 3 个独立的 16 位计数器。
- (2) 每个计数器可按二进制或十进制 (BCD) 计数。
- (3) 每个计数器可编程工作于 6 种不同工作方式。
- (4) 8254 每个计数器允许的最高计数频率为 10MHz (8253 为 2MHz)。
- (5) 8254 有读回命令 (8253 没有)，除了可以读出当前计数单元的内容外，还可以读出状态寄存器的内容。
- (6) 计数脉冲可以是有规律的时钟信号，也可以是随机信号。计数初值公式为：
$$n = f_{CLKi} \div f_{OUTi}$$
，其中  $f_{CLKi}$  是输入时钟脉冲的频率， $f_{OUTi}$  是输出波形的频率。

图 4.5.1 是 8254 的内部结构框图和引脚图，它是由与 CPU 的接口、内部控制电路和三个计数器组成。8254 的工作方式如下述：

- (1) 方式 0：计数到 0 结束输出正跃变信号方式。



- (2) 方式 1：硬件可重触发单稳方式。
- (3) 方式 2：频率发生器方式。
- (4) 方式 3：方波发生器。
- (5) 方式 4：软件触发选通方式。
- (6) 方式 5：硬件触发选通方式。

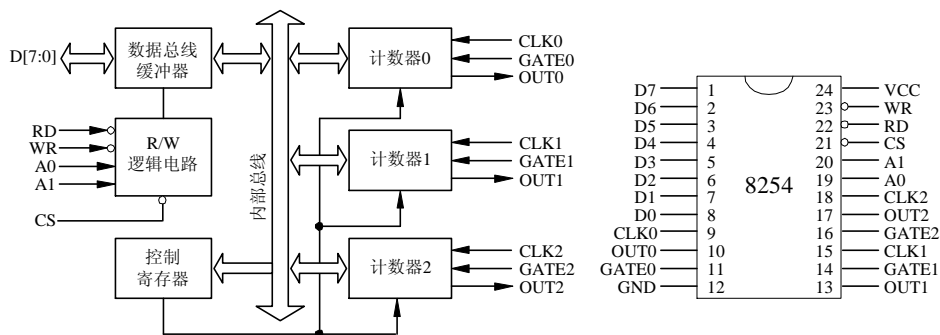


图 4.5.1 8254 的内部接口和引脚

8254 的控制字有两个：一个用来设置计数器的工作方式，称为方式控制字；另一个用来设置读回命令，称为读回控制字。这两个控制字共用一个地址，由标识位来区分。控制字格式如表 4.5.1—4.5.3 所示。

表 4.5.1 8254 的方式控制字格式

D7	D6	D5	D4	D3	D2	D1	D0
计数器选择		读/写格式选择		工作方式选择		计数码制选择	
00—计数器 0		00—锁存计数值		000—方式 0		0—二进制数	
01—计数器 1		01—读/写低 8 位		001—方式 1		1—十进制数	
10—计数器 2		10—读/写高 8 位		010—方式 2			
11—读出控制字标志		11—先读/写低 8 位 再读/写高 8 位		011—方式 3			
				100—方式 4			
				101—方式 5			

表 4.5.2 8254 读出控制字格式

D7	D6	D5	D4	D3	D2	D1	D0
1	1	0—锁存计数值	0—锁存状态信息	计数器选择（同方式控制字）			0

表 4.5.3 8254 状态字格式

D7	D6	D5	D4	D3	D2	D1	D0
OUT 引脚现行状态 1—高电平 0—低电平	计数初值是否装入 1—无效计数 0—计数有效	计数器方式（同方式控制字）					

8254 实验单元电路图如下图所示：

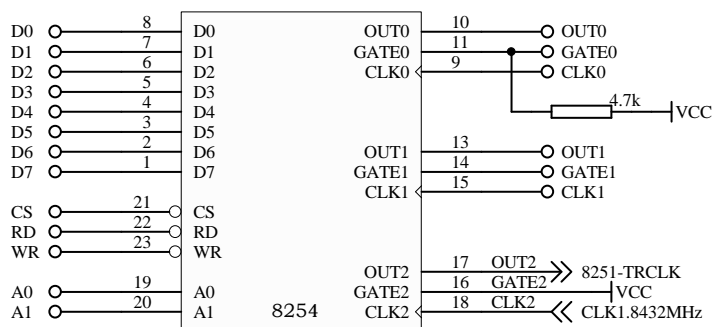


图 4.5.2 8254 实验电路原理图

### 4.5.5 实验步骤

#### 1. 计数应用实验

将 8254 的计数器 0 设置为方式 3，计数值为十进制数 4，用单次脉冲 KK1+ 作为 CLK0 时钟，OUT0 连接 MIR7，每当 KK1+ 按动 5 次后产生中断请求，在屏幕上显示字符“M”。

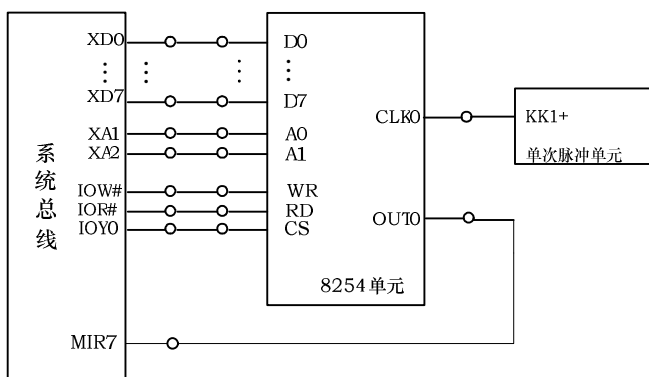



图 4.5.3 8254 计数应用实验接线图

实验步骤：

(1) 实验接线如图 4.5.3 所示(由于 8254 单元中 GATE0 信号已经上拉+5V,所以 GATE0 不用接线)。

(2) 编写实验程序(例程文件名为: A82541.ASM), 经编译、链接无误后装入系统。

(3) 单击  按钮, 运行实验程序, 每连续按动 5 次 KK1+, 在界面的输出区会显示字符“M”, 观察实验现象。实验现象结果如图 4.5.4 所示。

(4) 改变计数值, 验证 8254 的计数功能。



图 4.5.4 8254 计数实验结果图

## 实验程序清单

```

;=====
; 文件名: A82541.ASM
; 功能描述: 通过对计数器 0 进行计数, 计数初值为 4,
;           当计数满后, 产生正跳变触发中断, 中断
;           程序显示 M(每按 5 次输出一个 M)
;=====

IOY0    EQU    0600H                ;IOY0 起始地址
A8254    EQU    IOY0+00H*2
B8254    EQU    IOY0+01H*2
C8254    EQU    IOY0+02H*2
CON8254 EQU    IOY0+03H*2

SSTACK   SEGMENT STACK
          DW 32 DUP(?)
SSTACK   ENDS

CODE     SEGMENT
          ASSUME CS:CODE, SS:SSTACK
START:   PUSH DS
          MOV AX, 0000H
          MOV DS, AX
          MOV AX, OFFSET IRQ7        ;取中断入口地址
          MOV SI, 003CH               ;中断矢量地址
          MOV [SI], AX               ;填 IRQ7 的偏移矢量
          MOV AX, CS                  ;段地址
          MOV SI, 003EH
          MOV [SI], AX               ;填 IRQ7 的段地址矢量
          CLI
          POP DS
          ;初始化主片 8259

```

```

        MOV AL, 11H
        OUT 20H, AL           ;ICW1
        MOV AL, 08H
        OUT 21H, AL           ;ICW2
        MOV AL, 04H
        OUT 21H, AL           ;ICW3
        MOV AL, 01H
        OUT 21H, AL           ;ICW4
        MOV AL, 6FH           ;OCW1
        OUT 21H, AL
        ;8254
        MOV DX, CON8254
        MOV AL, 10H           ;计数器 0, 方式 0
        OUT DX, AL
        MOV DX, A8254
        MOV AL, 04H
        OUT DX, AL
        STI
AA1:    JMP AA1
IRQ7:   MOV DX, A8254
        MOV AL, 04H
        OUT DX, AL
        MOV AX, 014DH
        INT 10H               ;显示字符 M
        MOV AX, 0120H
        INT 10H
        MOV AL, 20H
        OUT 20H, AL           ;中断结束命令
        IRET
CODE    ENDS
        END    START

```

## 2. 定时应用实验

将 8254 的计数器 0 和计数器 1 都设置为方式 3, 用信号源 1MHz 作为 CLK0 时钟, OUT0 为波形输出 1ms 方波, 再通过 CLK1 输入, OUT1 输出 1s 方波。

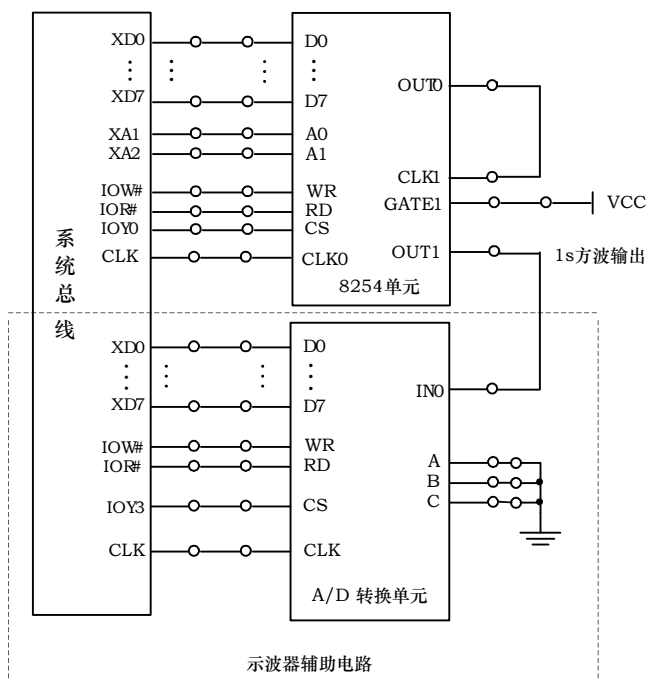






图 4.5.5 8254 定时应用实验接线图

实验步骤:

- (1) 接线图如图 4.5.5 所示。
- (2) 根据实验内容, 编写实验程序 (例程文件名为: A82542.ASM), 经编译、链接无误后装入系统。
- (3) 单击  按钮, 运行实验程序, 8254 的 OUT1 会输出 1s 的方波, 可用软件自带的示波器功能进行观察。
- (4) 用示波器观察波形的方法: 单击虚拟仪器菜单中的  示波器按钮或直接单击工具栏的  按钮, 在新弹出的示波器界面上单击  按钮运行示波器, 就可以观测出 OUT1 输出的波形。本实验现象结果如图 4.5.6 所示。

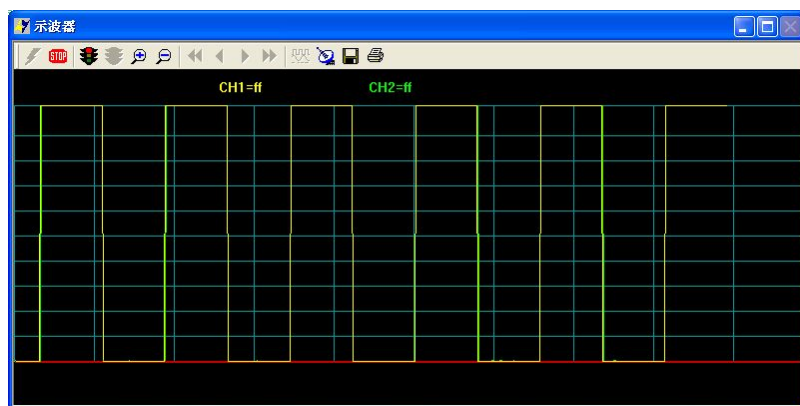


图 4.5.6 8254 定时应用实验结果图

## 实验程序清单

```

=====
; 文件名: A82542.ASM
; 功能描述: 产生 1s 方波, 输入时钟为 1MHz, 使用计数器 0 和 1
;           计数初值均为 03E8H
=====

```

```

IOY0    EQU    0600H           ;IOY0 起始地址
A8254    EQU    IOY0+00H*2
B8254    EQU    IOY0+01H*2
C8254    EQU    IOY0+02H*2
CON8254  EQU    IOY0+03H*2

```

```

SSTACK   SEGMENT STACK
          DW 32 DUP(?)
SSTACK   ENDS

```

```

CODE SEGMENT
        ASSUME CS:CODE
START:  MOV DX, CON8254           ;8254
        MOV AL, 36H              ;计数器 0, 方式 3
        OUT DX, AL
        MOV DX, A8254
        MOV AL, 0E8H
        OUT DX, AL
        MOV AL, 03H
        OUT DX, AL

```

```
MOV DX, CON8254      ;8254
MOV AL, 76H           ;计数器 1, 方式 3
OUT DX, AL
MOV DX, B8254
MOV AL, 0E8H
OUT DX, AL
MOV AL, 03H
OUT DX, AL
AA1:  JMP AA1
CODE  ENDS
END   START
```

## 4.6 8251 串行接口应用实验

### 4.6.1 实验目的

1. 掌握 8251 的工作方式及应用。
2. 了解有关串口通讯的知识。

### 4.6.2 实验设备

PC 机一台，TD-PITE 实验装置一套或两套。

### 4.6.3 实验内容

1. 数据信号的串行传输实验，循环向串口发送一个数，使用示波器测量 TXD 引脚上的波形，以了解串行传输的数据格式。
2. 自收自发实验，将 3000H 起始的 10 个单元中的初始数据发送到串口，然后自接收并保存到 4000H 起始的内存单元中。
3. 双机通讯实验，本实验需要两台实验装置，其中一台作为接收机，一台作为发送机，发送机将 3000H~3009H 内存单元中共 10 个数发送到接收机，接收机将接收到的数据直接在屏幕上输出显示。

### 4.6.4 实验原理

#### 1. 8251 的基本性能

8251 是可编程的串行通信接口，可以管理信号变化范围很大的串行数据通信。有下列基本性能：

- (1) 通过编程，可以工作在同步方式，也可以工作在异步方式。
- (2) 同步方式下，波特率为 0~64K，异步方式下，波特率为 0~19.2K。
- (3) 在同步方式时，可以用 5~8 位来代表字符，内部或外部同步，可自动插入同步字符。
- (4) 在异步方式时，也使用 5~8 位来代表字符，自动为每个数据增加 1 个启动位，并能够根据编程为每个数据增加 1 个、1.5 个或 2 个停止位。
- (5) 具有奇偶、溢出和帧错误检测能力。
- (6) 全双工，双缓冲器发送和接收器。



注意，8251 尽管通过了 RS-232 规定的基本控制信号，但并没有提供规定的全部信号。

## 2. 8251 的内部结构及外部引脚

8251 的内部结构图如图 4.6.1 所示，可以看出，8251 有 7 个主要部分，即数据总线缓冲器、读/写控制逻辑电路、调制/解调控制电路、发送缓冲器、发送控制电路、接收缓冲器和接收控制电路，图中还标识出了每个部分对外的引脚。

8251 的外部引脚如图 4.6.2 所示，共 28 个引脚，每个引脚信号的输入输出方式如图中的箭头方向所示。

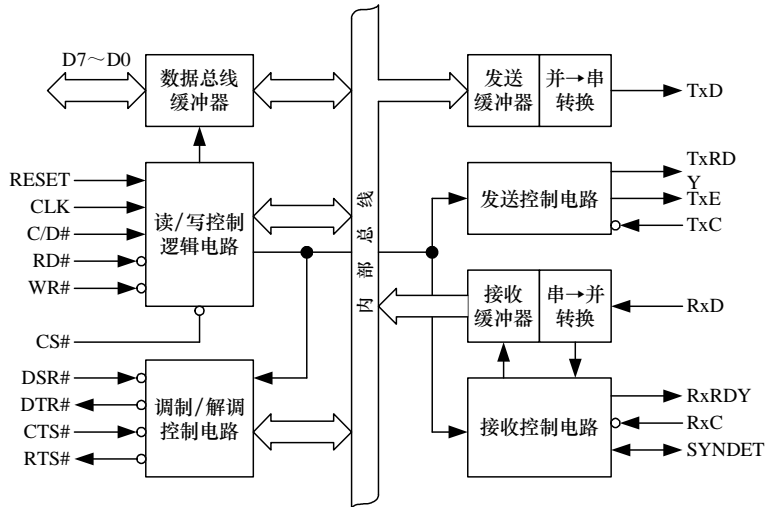


图 4.6.1 8251 内部结构图

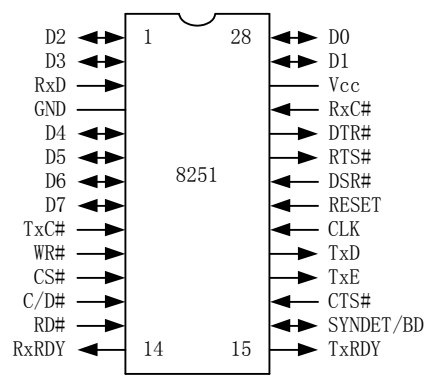


图 4.6.2 8251 外部引脚图

## 3. 8251 在异步方式下的 TXD 信号上的数据传输格式

图 4.6.3 示意了 8251 工作在异步方式下的 TXD 信号上的数据传输格式。数据位与停止位的位数可以由编程指定。

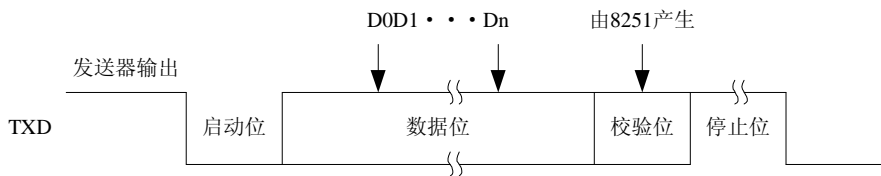


图 4.6.3 8251 工作在异步方式下 TXD 信号的数据传输格式

4. 8251 的编程

对 8251 的编程就是对 8251 的寄存器的操作,下面分别给出 8251 的几个寄存器的格式。

(1) 方式控制字

方式控制字用来指定通信方式及其方式下的数据格式,具体各位的定义如图 4.6.4 所示。

D7	D6	D5	D4	D3	D2	D1	D0
SCS/S2	ESD/S1	EP	PEN	L2	L1	B2	B1
同步/停止位		奇偶校验		字符长度		波特率系数	
同步 (D1D0=00)		异步 (D1D0≠0)		X0=无校验 01=奇校验 11=偶校验		00=5 位 01=6 位 10=7 位 11=8 位	
X0=内同步 X1=外同步 0X=双同步 1X=单同步		00=不用 01=1 位 10=1.5 位 11=2 位		异步 00=不用 01=01 10=16 11=64		同步 00=同步 方式标志	

图 4.6.4 8251 方式控制字

(2) 命令控制字

命令控制字用于指定 8251 进行某种操作 (如发送、接收、内部复位和检测同步字符等)

或处于某种工作状态,以便接收或发送数据。图 4.6.5 所示的是 8251 命令控制字各位的定义。

D7	D6	D5	D4	D3	D2	D1	D0
EH	IR	RTS	ER	SBRK	RxE	DTR	TxE
进入搜索 1=允许搜索	内部复位 1=使 8251 返回方式控制字	请求发送 1=使 RTS 输出 0	错误标志复 位使错误标志 PE、OE、FE 复位	发中止字符 1=使 TXD 为低 0=正常工作	接收允许 1=允许 0=禁止	数据终端准备好 1=使 DTR 输出 0	发送允许 1=允许 0=禁止

图 4.6.5 8251 命令控制字格式

(3) 状态字

CPU 通过状态字来了解 8251 当前的工作状态,以决定下一步的操作,8251 的状态字如图 4.6.6 所示。

D7	D6	D5	D4	D3	D2	D1	D0
DSR	SYNDET	FE	OE	PE	TxE	RxRDY	TxRDY
数据装置就绪: 当 DSR 输入为 0 时,该位为 1	同步检测	帧错误: 该标志仅用于异步方式,当在任一字符的结尾没有检测到有效的停止位时,该位置 1。此标志由命令控制字中的位 4 复位。	溢出错误: 在下一个字符变为可用前, CPU 没有把字符读走,此标志置 1。此错误出现时上一字符已丢失。	奇偶错误: 当检测到奇偶错误时此位置 1。	发送器空	接收就绪为 1 表明接收到一个字符。	发送就绪为 1 表明发送缓冲器空。

图 4.6.6 8251 状态字格式

(4) 系统初始化

8251 的初始化和操作流程如图 4.6.7 所示。

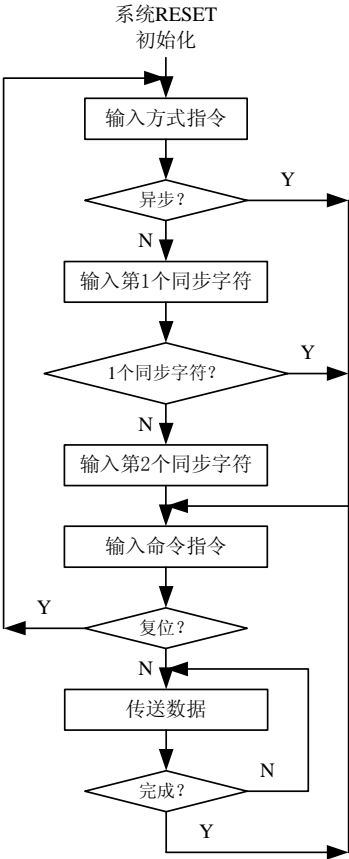


图 4.6.7 8251 初始化流程图

5. 8251 实验单元电路图

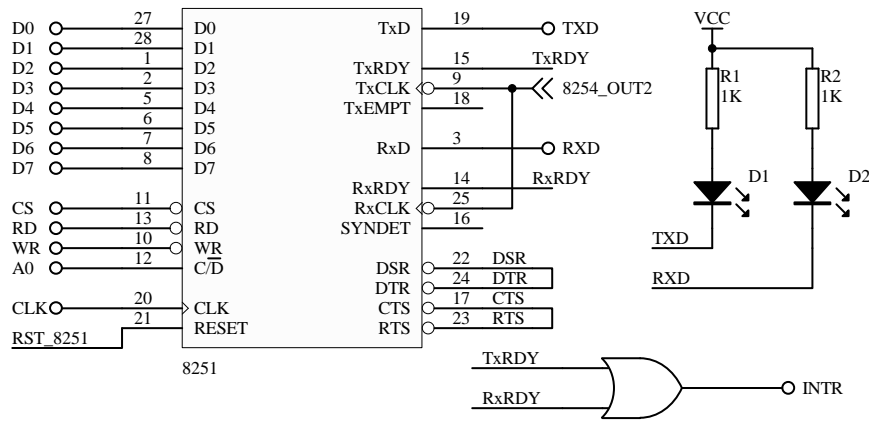


图 4.6.8 8251 实验单元电路图

## 4.6.5 实验步骤

### 1. 数据信号的串行传输

发送给串口的数据会以串行格式从 TXD 引脚输出，编写程序，观察串行输出的格式。实验步骤如下：

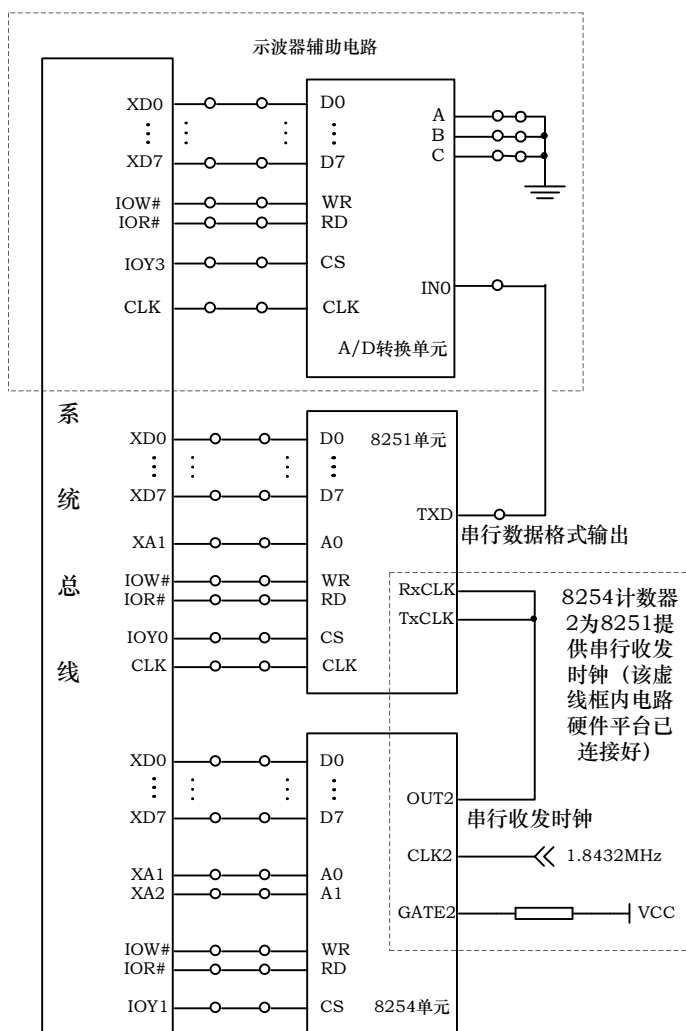






图 4.6.9 8251 数据串行传输实验线路图

- (1) 按图 4.6.9 连接实验接线。
- (2) 编写实验程序（例程文件名为：A82511.ASM），经编译、链接无误后装入系统。
- (3) 单击  按钮，运行实验程序，TXD 引脚上输出串行格式的数据波形。
- (4) 用示波器观察波形的方法：单击虚拟仪器菜单中的  示波器 按钮或直接单击工具栏

的按钮，在新弹出的示波器界面上单击按钮运行示波器，观测实验波形，分析串行数据传输格式。

### 实验程序清单

```
=====
; 文件名: A82511.ASM
; 功能描述: 研究了解串行传输的数据格式
=====
```

```
IOY0          EQU 0600H          ;IOY0 起始地址
IOY1          EQU 0640H          ;IOY1 起始地址
M8251_DATA    EQU IOY0+00H*2
M8251_CON     EQU IOY0+01H*2
M8254_2       EQU IOY1+02H*2
M8254_CON     EQU IOY1+03H*2
```

```
SSTACK  SEGMENT STACK
        DW 64 DUP(?)
```

```
SSTACK  ENDS
```

```
CODE  SEGMENT
```

```
        ASSUME CS:CODE
```

```
START:  CALL INIT
```

```
A1:     CALL SEND
        MOV CX, 0001H
```

```
A2:     MOV AX, 0F00H
```

```
A3:     DEC AX
```

```
        JNZ A3
```

```
        LOOP A2
```

```
        JMP A1
```

```
INIT:   MOV AL, 0B6H          ; 8254, 设置通讯时钟
```

```
        MOV DX, M8254_CON
```

```
        OUT DX, AL
```

```
        MOV AL, 1BH
```

```
        MOV DX, M8254_2
```

```
        OUT DX, AL
```

```
        MOV AL, 3AH
```

```
        OUT DX, AL
```

```
        CALL RESET          ; 对 8251 进行初始化
```

```
CALL DALLY
MOV AL, 7EH
MOV DX, M8251_CON      ; 写 8251 方式字
OUT DX, AL
CALL DALLY
MOV AL, 34H
OUT DX, AL              ; 写 8251 控制字
CALL DALLY
RET
RESET: MOV AL, 00H      ; 初始化 8251 子程序
MOV DX, M8251_CON      ; 控制寄存器
OUT DX, AL
CALL DALLY
OUT DX, AL
CALL DALLY
OUT DX, AL
CALL DALLY
MOV AL, 40H
OUT DX, AL
RET
DALLY: PUSH CX
MOV CX, 5000H
A4:    PUSH AX
POP AX
LOOP A4
POP CX
RET
SEND:  PUSH AX
PUSH DX
MOV AL, 31H
MOV DX, M8251_CON
OUT DX, AL
MOV AL, 55H
MOV DX, M8251_DATA      ; 发送数据 55H
OUT DX, AL
POP DX
POP AX
RET
CODE  ENDS
```

END START

2. 自收自发实验

通过自收自发实验，可以验证硬件及软件设计，常用于自测试。具体实验步骤如下：

- (1) 参考实验接线图如图 4.6.10 所示，按图连接实验线路。
- (2) 编写实验程序（例程文件名为：A82512.ASM），编译、链接无误后装入系统。
- (3) 使用 E 命令更改 4000H 起始的 10 个单元中的数据。
- (4) 运行实验程序，待程序运行停止。
- (5) 查看 3000H 起始的 10 个单元中的数据，与初始化的数据进行比较，验证程序功能。

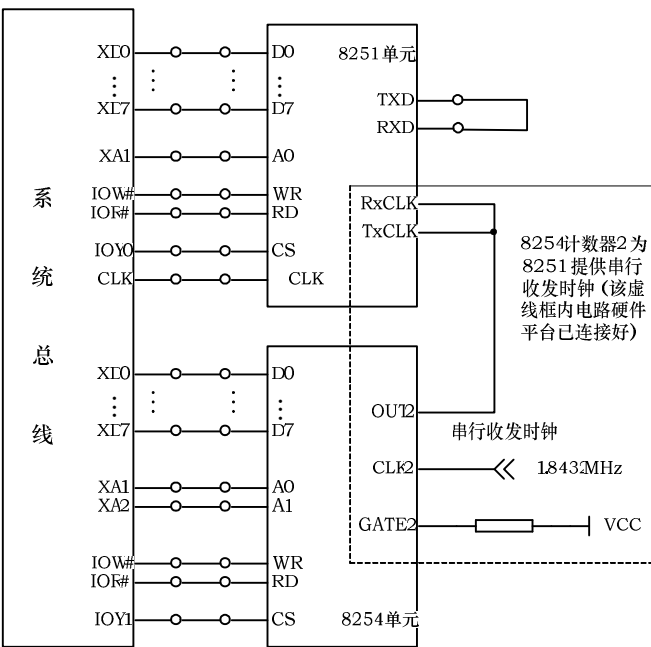


图 4.6.10 自收自发实验接线图

实验程序清单

```
=====
;
; 文件名: A82512.ASM
; 功能描述: 自收自发实验程序, 源地址 4000H, 目的地址 3000H
;=====

IOY0      EQU    0600H      ;IOY0 起始地址
IOY1      EQU    0640H      ;IOY1 起始地址
M8251_DATA EQU    IOY0+00H*2
M8251_CON EQU    IOY0+01H*2
M8254_2   EQU    IOY1+02H*2
```

M8254\_CON EQU IOY1+03H\*2

SSTACK SEGMENT STACK

DW 64 DUP(?)

SSTACK ENDS

CODE SEGMENT

ASSUME CS:CODE

START: MOV AX, 0000H

MOV DS, AX

;初始化 8254, 得到收发时钟

MOV AL, 0B6H

MOV DX, M8254\_CON

OUT DX, AL

MOV AL, 0CH

MOV DX, M8254\_2

OUT DX, AL

MOV AL, 00H

OUT DX, AL

;复位 8251

CALL INIT

CALL DALLY

;8251 方式字

MOV AL, 7EH

MOV DX, M8251\_CON

OUT DX, AL

CALL DALLY

;8251 控制字

MOV AL, 34H

OUT DX, AL

CALL DALLY

MOV DI, 3000H

MOV SI, 4000H

MOV CX, 000AH

A1: MOV AL, [SI]

PUSH AX

MOV AL, 37H

MOV DX, M8251\_CON

OUT DX, AL

POP AX



```
        MOV DX, M8251_DATA
        OUT DX, AL                ;发送数据
        MOV DX, M8251_CON
A2:     IN AL, DX                  ;判断发送缓冲是否为空
        AND AL, 01H
        JZ A2
        CALL DALLY
A3:     IN AL, DX                  ;判断是否接收到数据
        AND AL, 02H
        JZ A3
        MOV DX, M8251_DATA
        IN AL, DX                  ;读取接收到的数据
        MOV [DI], AL
        INC DI
        INC SI
        LOOP A1
        MOV AX, 4C00H
        INT 21H                    ;程序终止
INIT:   MOV AL, 00H                ;复位 8251 子程序
        MOV DX, M8251_CON
        OUT DX, AL
        CALL DALLY
        OUT DX, AL
        CALL DALLY
        OUT DX, AL
        CALL DALLY
        MOV AL, 40H
        OUT DX, AL
        RET
DALLY:  PUSH CX
        MOV CX, 3000H
A5:     PUSH AX
        POP AX
        LOOP A5
        POP CX
        RET
CODE    ENDS
        END START
```

3. 双机通讯实验

使用两台实验装置，一台为发送机，一台为接收机，进行两机间的串行通讯。实验步骤如下：

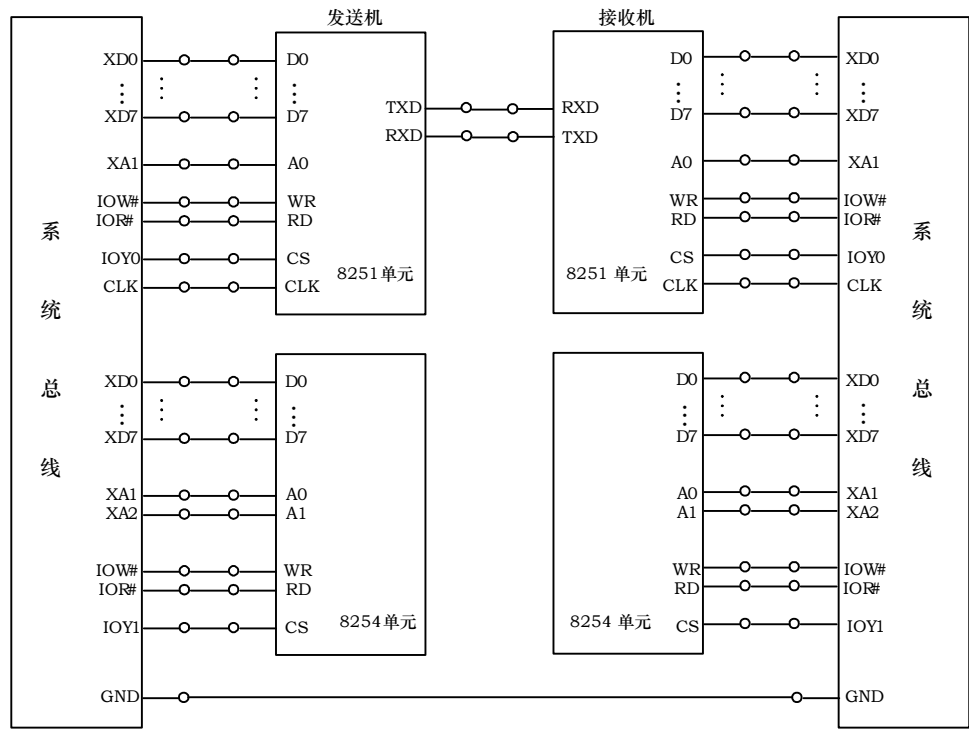


图 4.6.11 双机通讯实验接线图

- (1) 按图 4.6.11 连接实验线路。
- (2) 为两台机器分别编写实验程序（接收机例程文件名为：A82513.ASM，发送机例程文件名为：A82514.ASM），编译、链接后装入系统。
- (3) 为发送机初始化发送数据。在发送机 3000H~3009H 内存单元写入 ASCII 值：30，31，32，33，34，35，36，37，38，39 共 10 个数。
- (4) 首先运行接收机上的程序，等待接收数据，然后运行发送机上的程序，将数据发送到串口。
- (5) 观察接收机端屏幕上的显示是否与发送机端初始的数据相同，验证程序功能。屏幕将会显示字符：0123456789

实验程序清单

```
=====
; 文件名: A82513.ASM
; 功能描述: 接收机接收程序
=====

IOY0      EQU    0600H      ;IOY0 起始地址
```

```
IOY1      EQU 0640H      ;IOY1 起始地址
M8251_DATA EQU IOY0+00H*2
M8251_CON  EQU IOY0+01H*2
M8254_2    EQU IOY1+02H*2
M8254_CON  EQU IOY1+03H*2

SSTACK    SEGMENT STACK
          DW 64 DUP(?)
SSTACK    ENDS
CODE      SEGMENT
          ASSUME CS:CODE
START:    MOV AL, 0B6H      ;初始化 8254
          MOV DX, M8254_CON
          OUT DX, AL
          MOV AL, 0CH
          MOV DX, M8254_2
          OUT DX, AL
          MOV AL, 00H
          OUT DX, AL

          ;CLI
          CALL INIT         ;复位 8251
          CALL DALLY
          MOV AL, 7EH
          MOV DX, M8251_CON
          OUT DX, AL
          CALL DALLY
          MOV AL, 34H
          OUT DX, AL
          CALL DALLY
          MOV AX, 0152H      ;输出显示字符 R
          INT 10H
          MOV DI, 3000H
          MOV CX, 000AH
A1:       MOV DX, M8251_CON
          IN AL, DX
          AND AL, 02H
          JZ A1
          MOV DX, M8251_DATA
          IN AL, DX
```

```

        AND AL, 7FH
        MOV [DI],AL
        INC DI
        LOOP A1
        MOV AL, 00H
        MOV SI, 300AH
        MOV [SI], AL
        MOV AH, 06H
        MOV BX, 3000H
        INT 10H                ;输出显示接收到的数据
                                ;STI
A2:     JMP A2
INIT:   MOV AL, 00H            ;复位 8251 子程序
        MOV DX, M8251_CON
        OUT DX, AL
        CALL DALLY
        OUT DX, AL
        CALL DALLY
        OUT DX, AL
        CALL DALLY
        MOV AL, 40H
        OUT DX, AL
        RET
DALLY:  PUSH CX
        MOV CX, 3000H
A3:     PUSH AX
        POP AX
        LOOP A3
        POP CX
        RET
CODE    ENDS
        END START

;=====
; 文件名: A82514.ASM
; 功能描述: 发送机的发送程序
;=====

IOY0    EQU    0600H          ;IOY0 起始地址

```

```
IOY1      EQU 0640H      ;IOY1 起始地址
M8251_DATA EQU IOY0+00H*2
M8251_CON  EQU IOY0+01H*2
M8254_2    EQU IOY1+02H*2
M8254_CON  EQU IOY1+03H*2

SSTACK    SEGMENT STACK
          DW 64 DUP(?)
SSTACK    ENDS
CODE      SEGMENT
          ASSUME CS:CODE
START:    MOV AL, 0B6H      ;初始化 8254, 得到收发时钟
          MOV DX, M8254_CON
          OUT DX, AL
          MOV AL, 0CH
          MOV DX, M8254_2
          OUT DX, AL
          MOV AL, 00H
          OUT DX, AL
          CALL INIT          ;复位 8251
          CALL DALLY
          MOV AL, 7EH
          MOV DX, M8251_CON
          OUT DX, AL          ;8251 方式字
          CALL DALLY
          MOV AL, 34H
          OUT DX, AL          ;8251 控制字
          CALL DALLY
          MOV DI, 3000H
          MOV CX, 000AH
A1:       MOV AL, [DI]
          CALL SEND
          CALL DALLY
          INC DI
          LOOP A1
A2:       JMP A2
INIT:     MOV AL, 00H          ;复位 8251 子程序
          MOV DX, M8251_CON
          OUT DX, AL
```

```
        CALL DALLY
        OUT DX, AL
        CALL DALLY
        OUT DX, AL
        CALL DALLY
        MOV AL, 40H
        OUT DX, AL
        RET
DALLY:  PUSH CX
        MOV CX, 3000H
A4:     PUSH AX
        POP AX
        LOOP A4
        POP CX
        RET
SEND:   PUSH AX                ;数据发送子程序
        PUSH DX
        MOV AL, 31H
        MOV DX, M8251_CON
        OUT DX, AL
        POP AX
        MOV DX, M8251_DATA
        OUT DX, AL
        MOV DX, M8251_CON
A3:     IN AL, DX
        AND AL, 01H
        JZ A3
        POP DX
        RET
CODE    ENDS
        END START
```

## 4.7 A/D 转换实验

### 4.7.1 实验目的

1. 学习理解模/数信号转换的基本原理。
2. 掌握模/数转换芯片 ADC0809 的使用方法。

### 4.7.2 实验设备

PC 机一台，TD-PITE 实验装置一套，万用表一个。

### 4.7.3 实验内容

编写实验程序，将 ADC 单元中提供的 0V~5V 信号源作为 ADC0809 的模拟输入量，进行 A/D 转换，转换结果通过变量进行显示。

### 4.7.4 实验原理

ADC0809 包括一个 8 位的逐次逼近型的 ADC 部分，并提供一个 8 通道的模拟多路开关和联合寻址逻辑。用它可直接输入 8 个单端的模拟信号，分时进行 A/D 转换，在多点巡回检测、过程控制等应用领域中使用非常广泛。ADC0809 的主要技术指标为：

- 分辨率：8 位
- 单电源：+5V
- 总的不可调误差： $\pm 1\text{LSB}$
- 转换时间：取决于时钟频率
- 模拟输入范围：单极性 0~5V
- 时钟频率范围：10KHz~1280KHz

ADC0809 的外部管脚如图 4.7.1 所示，地址信号与选中通道的关系如表 4.7.1 所示。

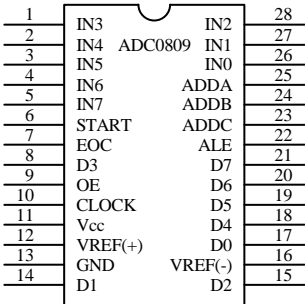


图 4.7.1 ADC0809 外部引脚图

表 4.7.1 地址信号与选中通道的关系

地 址			选中通道
A	B	C	
0	0	0	IN0
0	0	1	IN1
0	1	0	IN2
0	1	1	IN3
1	0	0	IN4
1	0	1	IN5
1	1	0	IN6
1	1	1	IN7

模/数转换单元电路图如图 4.7.2 所示：

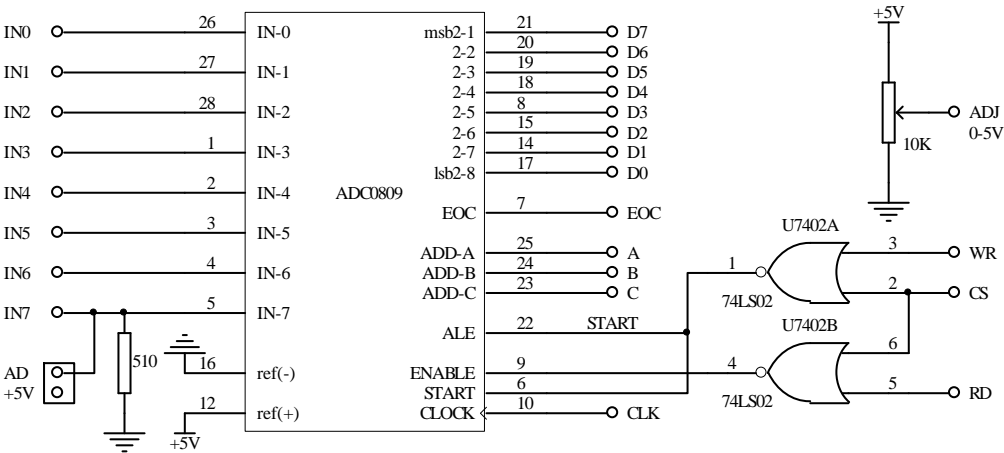


图 4.7.2 模/数转换电路图



### 4.7.5 实验步骤

- 1. 按图 4.7.4 连接实验线路。
- 2. 编写实验程序（例程文件名为：AD0809.ASM），经编译、链接无误后装入系统。
- 3. 将变量 VALUE 添加到变量监视窗口中。

方法如下：打开设置\变量监控，出现如图 4.7.3 的界面，选中要监视的变量“VALUE”，单击“加入监视”后确定，就会在软件左侧栏的“变量区”出现该值。



图 4.7.3 AD 转换实验接线图

- 4. 在 JMP START 语句行设置断点，使用万用表测量 ADJ 端的电压值，计算对应的采样值，然后运行程序。
- 5. 程序运行到断点处停止运行，查看变量窗口中 VALUE 的值，与计算的理论值进行比较，看是否一致（可能稍有误差，相差不大）。
- 6. 调节电位器，改变输入电压，比较 VALUE 与计算值，反复验证程序功能。

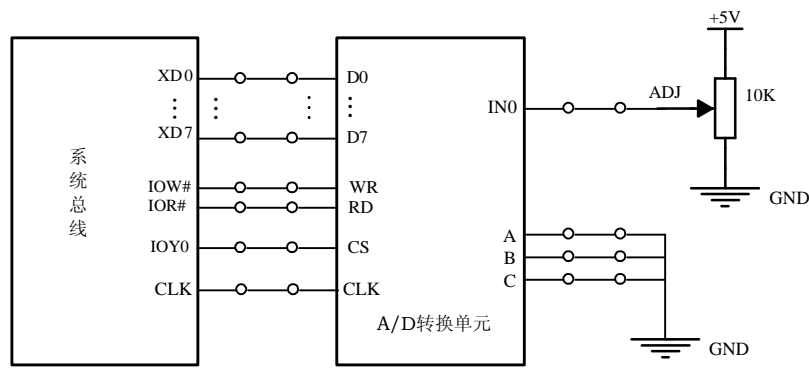


图 4.7.4 AD 转换实验接线图

#### 实验程序清单

=====

```
; 文件名: AD0809.ASM
; 功能说明: 进行 AD 采样, 将结果显示。 片选为 IOY0
;=====
```

```
IOY0      EQU    0600H
AD0809     EQU    IOY0+00H*2      ;AD0809 的端口地址

SSTACK     SEGMENT STACK
            DW 64 DUP(?)
SSTACK     ENDS
PUBLIC VALUE      ;设置全局变量以便变量监视
DATA      SEGMENT
VALUE DB ?          ;AD 转换结果
DATA      ENDS
CODE      SEGMENT
            ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
            MOV DS, AX
            MOV DX, AD0809      ;启动 AD 采样
            OUT DX, AL
            CALL DALLY
            IN  AL, DX          ;读 AD 采样结果
            MOV VALUE, AL       ;将结果送变量
            JMP START           ;在此处设置断点, 观察变量窗口中的 VALUE 值
DALLY: PUSH CX                ;延时程序
            PUSH AX
            MOV CX, 100H
A5:  MOV AX, 0800H
A6:  DEC AX
            JNZ A6
            LOOP A5
            POP AX
            POP CX
            RET
CODE      ENDS
            END START
```

## 4.8 D/A 转换实验

### 4.8.1 实验目的

- 1. 学习数/模转换的基本原理。
- 2. 掌握 DAC0832 的使用方法。

### 4.8.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 4.8.3 实验内容

设计实验电路图实验线路并编写程序，实现 D/A 转换，要求产生锯齿波、脉冲波，并用示波器观察电压波形。

### 4.8.4 实验原理

D/A 转换器是一种将数字量转换成模拟量的器件，其特点是：接收、保持和转换的数字信息，不存在随温度、时间漂移的问题，其电路抗干扰性较好。大多数的 D/A 转换器接口设计主要围绕 D/A 集成芯片的使用及配置响应的外围电路。DAC0832 是 8 位芯片，采用 CMOS 工艺和 R-2RT 形电阻解码网络，转换结果为一对差动电流 Iout1 和 Iout2 输出，其主要性能参数如表 4.7 示，引脚如图 4.8.1 所示。

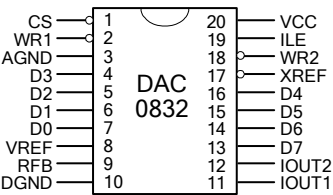


图 4.8.1 DAC0832 引脚图

表 4.8.1 DAC0832 性能参数

性能参数	参数值
分辨率	8 位
单电源	+5V~ +15V
参考电压	+10V~-10V
转换时间	1Us
满刻度误差	± 1LSB
数据输入电平	与 TTL 电平兼容

图 4.8.2 D/A 实验单元电路图

### 4.8.5 实验步骤



Figure 1-1-1 is a schematic diagram of a waveform auxiliary circuit. It features a central '系统总线' (System Bus) on the left. Two main functional blocks are connected to this bus: a 'D/A转换单元' (D/A Conversion Unit) and an 'A/D转换单元' (A/D Conversion Unit).  
 The 'D/A转换单元' has data inputs XD0 through XD7 and control inputs IOW# and IOY0. It provides data outputs D0 through D7, control outputs WR and CS, and a single output line labeled OUT.  
 The 'A/D转换单元' has data inputs XD0 through XD7 and control inputs IOW#, IOR#, IOY3, and CLK. It provides data outputs D0 through D7, control outputs WR and RD, and a control input CS. It also has a control output CLK.  
 The OUT line from the D/A unit and the INO line from the A/D unit are connected to a common horizontal line. This line leads to a terminal labeled '波形输出' (Waveform Output).  
 Additionally, the A/D unit has three inputs labeled A, B, and C, which are connected to a common ground symbol.

图 4.8.3 D/A 实验接线图

2. 编写实验程序（锯齿波例程文件名为：DA08321.ASM，方波例程文件名为：DA08322.ASM），经编译、链接无误后装入系统。

3. 单击  按钮，运行实验程序，用示波器测量 DA 的输出，观察实验现象。

4. 用示波器观察波形的方法：单击虚拟仪器菜单中的  示波器 按钮或直接单击工具栏的

 按钮，在新弹出的示波器界面上单击  按钮运行示波器，观测实验波形。

5. 本实验现象结果如图 4.8.4 和图 4.8.5 所示。

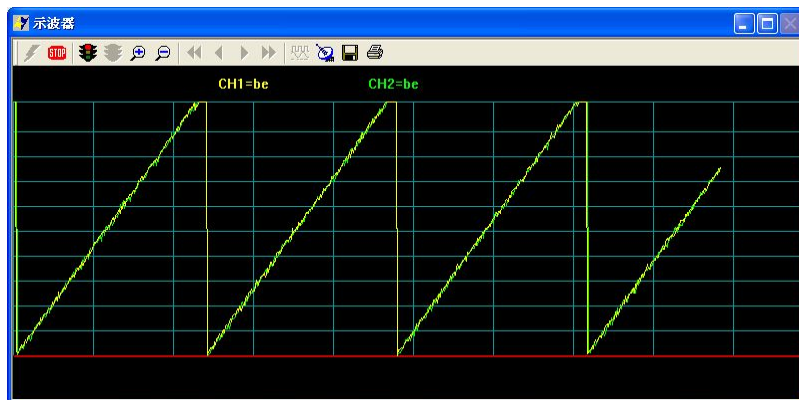


图 4.8.4 DA0832 产生锯齿波实验结果图

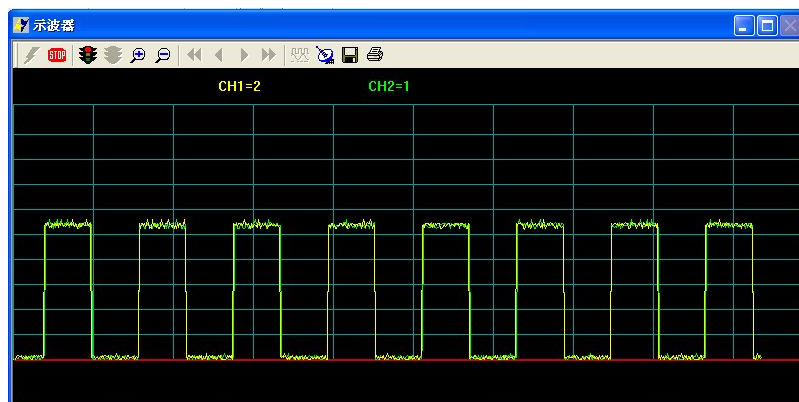


图 4.8.5 DA0832 产生方波实验结果图

### 实验程序清单

```
=====
; 文件名: DA08321.ASM
; 功能描述: 利用 DA0832 产生锯齿波，通过延时变化波形
=====
```

```
IOY0      EQU    0600H
DA0832    EQU    IOY0+00H*2
```

```
STACK SEGMENT STACK
        DW 32 DUP(?)
```

```
STACK ENDS
```

```
CODE SEGMENT
```

```
ASSUME  CS:CODE, SS:STACK
```

```
START: MOV AX, 00H ; 产生锯齿波
```

```
        MOV DX, DA0832
```

```
        MOV AL, 00H
```

```
AA1:    OUT DX, AL
```

```
        CALL DELAY
```

```
        INC AL
```

```
        JMP AA1
```

```
DELAY:  PUSH CX
```

```
        MOV CX, 03FFH
```

```
AA2:    PUSH AX
```

```
        POP  AX
```

```
        LOOP AA2
```

```
        POP CX
```

```
        RET
```

```
CODE ENDS
```

```
END START
```

```
;=====
```

```
; 文件名: DA08322.ASM
```

```
; 功能描述: 利用 DA0832 产生方波, 通过延时变化波形
```

```
;=====
```

```
IOY0      EQU    0600H
```

```
DA0832    EQU    IOY0+00H*2
```

```
SSTACK SEGMENT STACK
        DW 32 DUP(?)
```

```
SSTACK ENDS
```

```
CODE SEGMENT
```

```
        ASSUME CS:CODE
```

```
START: MOV AX, 00H ; 产生方波
```

```
        MOV DX, DA0832
AA1:    MOV AL, 00H
        OUT DX, AL
        CALL DELAY
        MOV AL, 7FH
        OUT DX, AL
        CALL DELAY
        JMP AA1
DELAY:  PUSH CX
        MOV CX, 0FF00H
AA2:    PUSH AX
        POP  AX
        LOOP AA2
        POP CX
        RET
CODE    ENDS
        END START
```

## 4.9 键盘扫描及显示设计实验

### 4.9.1 实验目的

了解键盘扫描及数码显示的基本原理，熟悉 8255 的编程。

### 4.9.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 4.9.3 实验内容

将 8255 单元与键盘及数码管显示单元连接，编写实验程序，扫描键盘输入，并将扫描结果送数码管显示。键盘采用  $4 \times 4$  键盘，每个数码管显示值可为 0~F 共 16 个数。实验具体内容如下：将键盘进行编号，记作 0~F，当按下其中一个按键时，将该按键对应的编号在一个数码管上显示出来，当再按下一个按键时，便将这个按键的编号在下一个数码管上显示出来，数码管上可以显示最近 4 次按下的按键编号。

键盘及数码管显示单元电路图如图 4.9.1 所示。8255 键盘及显示实验参考接线图如图 4.9.2 所示。



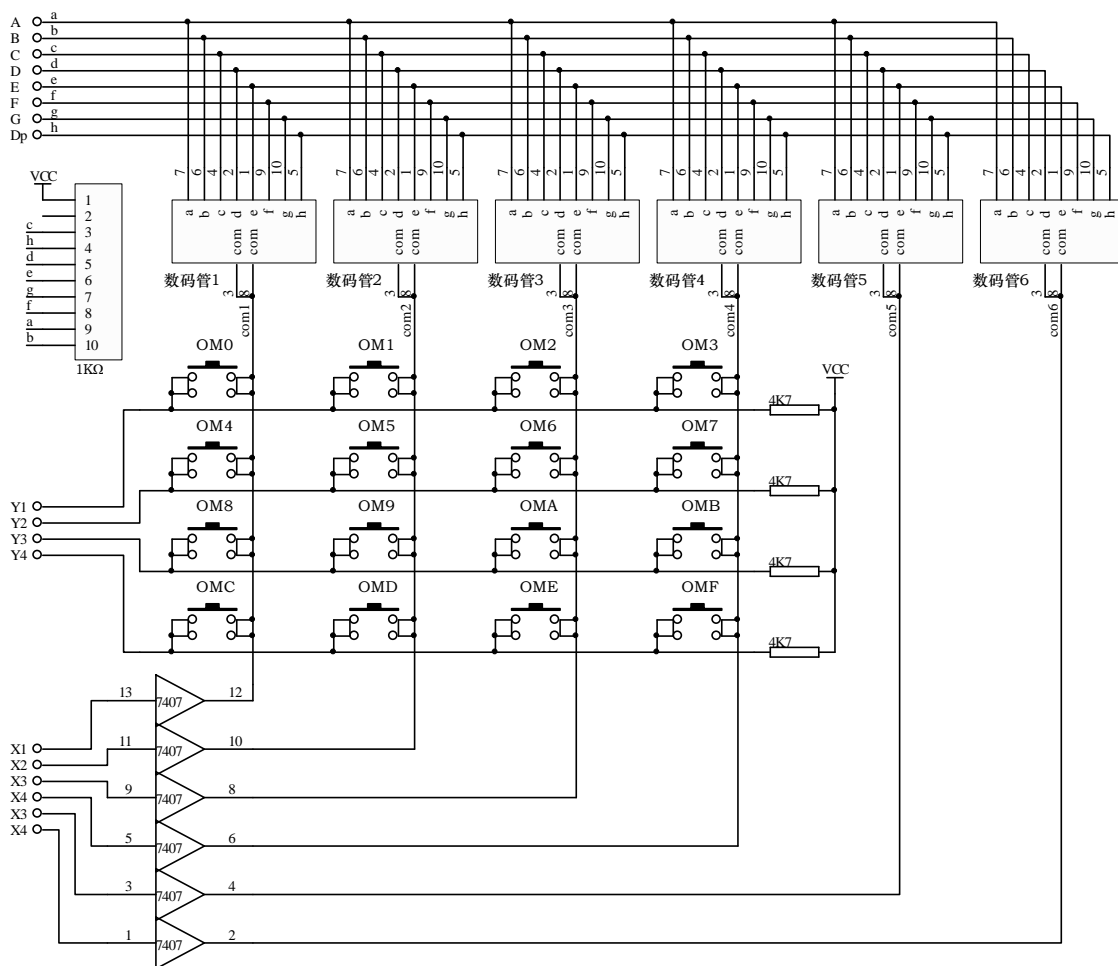


图 4.9.1 键盘及数码管显示单元电路图

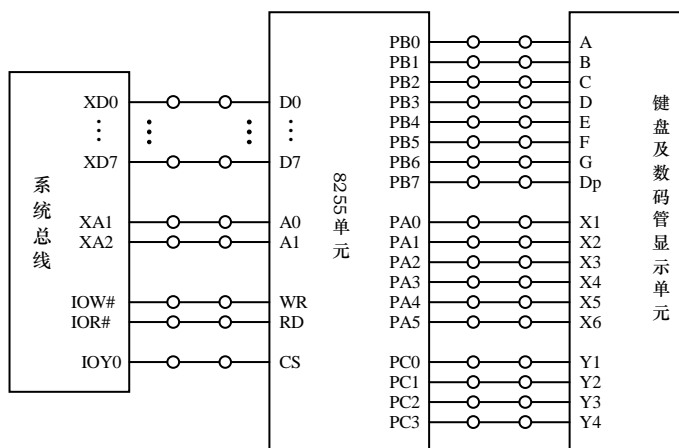


图 4.9.2 8255 键盘扫描及数码管显示实验线路图

## 4.9.4 实验步骤

1. 按图 4.9.2 连接线路图。
2. 编写实验程序（例程文件名为：KEYSCAN.ASM），检查无误后编译、连接并装入系统。
3. 运行程序，按下按键，观察数码管的显示，验证程序功能。
4. 固化程序，然后脱机运行程序。

### 实验程序清单

```

;=====
; 文件名: Keyscan.asm
; 功能描述: 键盘及数码管显示实验，通过 8255 控制。
;      8255 的 B 口控制数码管的段显示，A 口控制键盘列扫描
;      及数码管的位驱动，C 口控制键盘的行扫描。
;      按下按键，该按键对应的位置将按顺序显示在数码管上。
;=====

```

```

IOY0      EQU    0600H      ;片选 IOY0 对应的端口始地址
MY8255_A   EQU    IOY0+00H*2 ;8255 的 A 口地址
MY8255_B   EQU    IOY0+01H*2 ;8255 的 B 口地址
MY8255_C   EQU    IOY0+02H*2 ;8255 的 C 口地址
MY8255_CON EQU    IOY0+03H*2 ;8255 的控制寄存器地址

```

```

SSTACK    SEGMENT STACK
            DW 16 DUP(?)
SSTACK    ENDS

```

```

DATA      SEGMENT
DTABLE    DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H
            DB 7FH,6FH,77H,7CH,39H,5EH,79H,71H
DATA      ENDS

```

```

CODE      SEGMENT
            ASSUME CS:CODE,DS:DATA
START:    MOV AX,DATA
            MOV DS,AX
            MOV SI,3000H
            MOV AL,00H
            MOV [SI],AL      ;清显示缓冲

```

```

MOV [SI+1],AL
MOV [SI+2],AL
MOV [SI+3],AL
MOV [SI+4],AL
MOV [SI+5],AL
MOV DI,3005H
MOV DX,MY8255_CON      ;写 8255 控制字
MOV AL,81H
OUT DX,AL
BEGIN: CALL DIS          ;调用显示子程序
      CALL CLEAR         ;清屏
      CALL CCSCAN        ;扫描
      JNZ INK1
      JMP BEGIN
INK1:  CALL DIS
      CALL DALLY
      CALL DALLY
      CALL CLEAR
      CALL CCSCAN
      JNZ INK2           ;有键按下，转到 INK2
      JMP BEGIN
;确定按下键的位置
INK2:  MOV CH,0FEH
      MOV CL,00H
COLUM: MOV AL,CH
      MOV DX,MY8255_A
      OUT DX,AL
      MOV DX,MY8255_C
      IN AL,DX
L1:    TEST AL,01H        ;is L1?
      JNZ L2
      MOV AL,00H         ;L1
      JMP KCODE
L2:    TEST AL,02H        ;is L2?
      JNZ L3
      MOV AL,04H         ;L2
      JMP KCODE
L3:    TEST AL,04H        ;is L3?
      JNZ L4

```

```
        MOV AL,08H                ;L3
        JMP KCODE
L4:      TEST AL,08H                ;is L4?
        JNZ NEXT
        MOV AL,0CH                ;L4
KCODE:   ADD AL,CL
        CALL PUTBUF
        PUSH AX
KON:     CALL DIS
        CALL CLEAR
        CALL CCSCAN
        JNZ KON
        POP AX
NEXT:    INC CL
        MOV AL,CH
        TEST AL,08H
        JZ KERR
        ROL AL,1
        MOV CH,AL
        JMP COLUM
KERR:    JMP BEGIN

CCSCAN:  MOV AL,00H                ;键盘扫描子程序
        MOV DX,MY8255_A
        OUT DX,AL
        MOV DX,MY8255_C
        IN  AL,DX
        NOT AL
        AND AL,0FH
        RET

CLEAR:   MOV DX,MY8255_B          ;清屏子程序
        MOV AL,00H
        OUT DX,AL
        RET

DIS:     PUSH AX                  ;显示子程序
        MOV SI,3000H
        MOV DL,0DFH
        MOV AL,DL
AGAIN:   PUSH DX
```

```
        MOV DX,MY8255_A
        OUT DX,AL
        MOV AL,[SI]
        MOV BX,OFFSET DTABLE
        AND AX,00FFH
        ADD BX,AX
        MOV AL,[BX]
        MOV DX,MY8255_B
        OUT DX,AL
        CALL DALLY
        INC SI
        POP DX
        MOV AL,DL
        TEST AL,01H
        JZ  OUT1
        ROR AL,1
        MOV DL,AL
        JMP AGAIN
OUT1:   POP AX
        RET
DALLY:  PUSH CX                                ;延时子程序
        MOV CX,0006H
T1:     MOV AX,009FH
T2:     DEC AX
        JNZ T2
        LOOP T1
        POP CX
        RET
PUTBUF:  MOV SI,DI                            ;存键盘值到相应位的缓冲中
        MOV [SI],AL
        DEC DI
        CMP DI,2FFFFH
        JNZ GOBACK
        MOV DI,3005H
GOBACK:  RET

CODE    ENDS
        END START
```

## 4.10 电子发声设计实验

### 4.10.1 实验目的

学习用 8254 定时/计数器使蜂鸣器发声的编程方法。

### 4.10.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 4.10.3 实验内容

根据实验提供的音乐频率表和时间表，编写程序控制 8254，使其输出连接到扬声器上能发出相应的乐曲。

### 4.10.4 实验说明及步骤

一个音符对应一个频率，将对应一个音符频率的方波通到扬声器上，就可以发出这个音符的声音。将一段乐曲的音符对应频率的方波依次送到扬声器，就可以演奏出这段乐曲。利用 8254 的方式 3——“方波发生器”，将相应一种频率的计数初值写入计数器，就可产生对应频率的方波。计数初值的计算如下：

$$\text{计数初值} = \text{输入时钟} \div \text{输出频率}$$

例如输入时钟采用 1MHz，要得到 800Hz 的频率，计数初值即为  $1000000 \div 800$ 。音符与频率对照关系如表 4.10.1 所示。对于每一个音符的演奏时间，可以通过软件延时来处理。首先确定单位延时时间程序（根据 CPU 的频率不同而有所变化）。然后确定每个音符演奏需要几个单位时间，将这个值送入 DL 中，调用 DALLY 子程序即可。

```
;单位延时时间  
DALLY PROC  
D0: MOV CX, 0010H  
D1: MOV AX, 0F00H  
D2: DEC AX  
JNZ D2
```

```

LOOP D1
RET
DALLY ENDP
; N 个单位延时时间 (N 送至 DL)
DALLY PROC
D0: MOV CX, 0010H
D1: MOV AX, 0F00H
D2: DEC AX
    JNZ D2
    LOOP D1
    DEC DL
    JNZ D0
    RET
DALLY ENDP

```

表 4.10.1 音符与频率对照表 (单位: Hz)

音符 音调	1	2	3	4	5	6	7
A	221	248	278	294	330	371	416
B	248	278	312	330	371	416	467
C	131	147	165	175	196	221	248
D	147	165	185	196	221	248	278
E	165	185	208	221	248	278	312
F	175	196	221	234	262	294	330
G	196	221	248	262	294	330	371
音符 音调	1	2	3	4	5	6	7
A	441	495	556	589	661	742	833
B	495	556	624	661	742	833	935
C	262	294	330	350	393	441	495
D	294	330	371	393	441	495	556
E	330	371	416	441	495	556	624
F	350	393	441	467	525	589	661
G	393	441	495	525	589	661	742
音符 音调	1	2	3	4	5	6	7
A	882	990	1112	1178	1322	1484	1665
B	990	1112	1248	1322	1484	1665	1869
C	525	589	661	700	786	882	990
D	589	661	742	786	882	990	1112
E	661	742	833	882	990	1112	1248
F	700	786	882	935	1049	1178	1322
G	786	882	990	1049	1178	1322	1484

下面提供了乐曲《友谊地久天长》实验参考程序。程序中频率表是将曲谱中的音符对应的频率值依次记录下来 (B 调、四分之二拍), 时间表是将各个音符发音的相对时间记录下来 (由曲谱中节拍得出)。

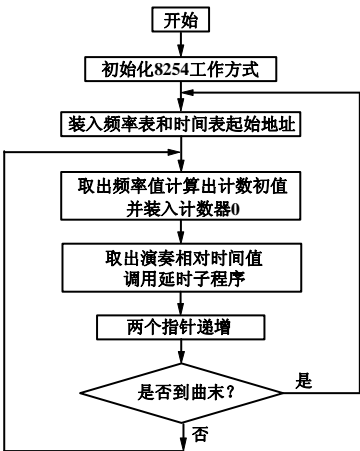


图 4.10.1 实验参考流程图

频率表和时间表是一一对应的，频率表的最后一项为 0，作为重复的标志。根据频率表中的频率算出对应的计数初值，然后依次写入 8254 的计数器。将时间表中相对时间值带入延时程序来得到音符演奏时间。实验参考程序流程如图 4.10.1 所示。

电子发声电路图如图 4.10.2 所示。

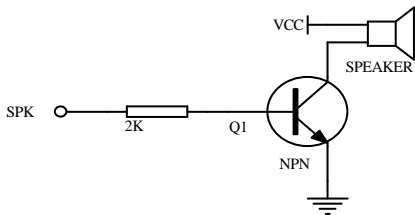


图 4.10.2 电子发声单元电路图

实验步骤如下：

1. 参考图 4.10.3 所示连接实验线路。
2. 编写实验程序（例程文件名为：SOUND.ASM），经编译、连接无误后装入系统。
3. 运行程序，听扬声器发出的音乐是否正确。
4. 固化程序，然后脱机运行程序。

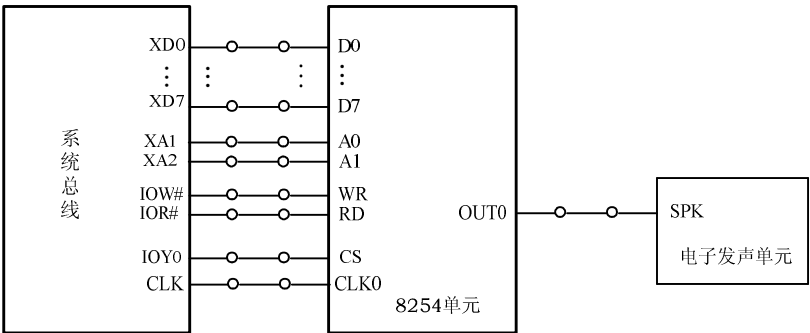


图 4.10.3 8254 电子发声实验接线图



## 实验程序清单

```

;=====
; SOUND.asm
; 电子发声设计实验
;=====

; 端口定义
IOY0          EQU 0600H
MY8254_COUNT0 EQU IOY0+00H*2      ;8254 计数器 0 端口地址
MY8254_COUNT1 EQU IOY0+01H*2      ;8254 计数器 1 端口地址
MY8254_COUNT2 EQU IOY0+02H*2      ;8254 计数器 2 端口地址
MY8254_MODE    EQU IOY0+03H*2      ;8254 控制寄存器端口地址
STACK1         SEGMENT STACK
                DW 256 DUP(?)
STACK1         ENDS
DATA          SEGMENT
FREQ_LIST DW 371,495,495,495,624,556,495,556,624      ;频率表
            DW 495,495,624,742,833,833,833,742,624
            DW 624,495,556,495,556,624,495,416,416,371
            DW 495,833,742,624,624,495,556,495,556,833
            DW 742,624,624,742,833,990,742,624,624,495
            DW 556,495,556,624,495,416,416,371,495,0
TIME_LIST DB 4, 6, 2, 4, 4, 6, 2, 4, 4      ;时间表
            DB 6, 2, 4, 4, 12, 1, 3, 6, 2
            DB 4, 4, 6, 2, 4, 4, 6, 2, 4, 4
            DB 12, 4, 6, 2, 4, 4, 6, 2, 4, 4
            DB 6, 2, 4, 4, 12, 4, 6, 2, 4, 4
            DB 6, 2, 4, 4, 6, 2, 4, 4, 12
DATA          ENDS

CODE          SEGMENT
                ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
        MOV DS, AX
        MOV DX, MY8254_MODE      ;初始化 8254 工作方式
        MOV AL, 36H              ;定时器 0、方式 3
        OUT DX, AL
BEGIN: MOV SI, OFFSET FREQ_LIST  ;装入频率表起始地址

```

```

MOV DI,OFFSET TIME_LIST      ;装入时间表起始地址
PLAY: MOV DX,0FH              ;输入时钟为 1MHz, 1M = 0F4240H
MOV AX,4240H
DIV WORD PTR [SI]            ;取出频率值计算计数初值, 0F4240H / 输出频
率
MOV DX,MY8254_COUNT0
OUT DX,AL                    ;装入计数初值
MOV AL,AH
OUT DX,AL
MOV DL,[DI]                  ;取出演奏相对时间, 调用延时子程序
CALL DALLY
ADD SI,2
INC DI
CMP WORD PTR [SI],0          ;判断是否到曲末?
JE BEGIN
JMP PLAY
DALLY PROC                    ;延时子程序
D0:  MOV CX,0010H
D1:  MOV AX,0F00H
D2:  DEC AX
JNZ D2
LOOP D1
DEC DL
JNZ D0
RET
DALLY ENDP
CODE ENDS
END START

```

## 4.11 点阵 LED 显示设计实验

### 4.11.1 实验目的

1. 了解 LED 点阵的基本结构。
2. 学习 LED 点阵扫描显示程序的设计方法。

### 4.11.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 4.11.3 实验内容及原理

编写程序，控制点阵向上卷动显示“西安唐都科教仪器公司!”。

实验系统中的  $16 \times 16$  LED 点阵由四块  $8 \times 8$  LED 点阵组成，如图 4.11.1 所示， $8 \times 8$  点阵内部结构图如图 4.11.2 所示。由图 4.11.2 可知，当行为“0”，列为“1”，则对应行、列上的 LED 点亮。图 4.11.3 为点阵外部引脚图。汉字显示如图 4.11.4 所示。

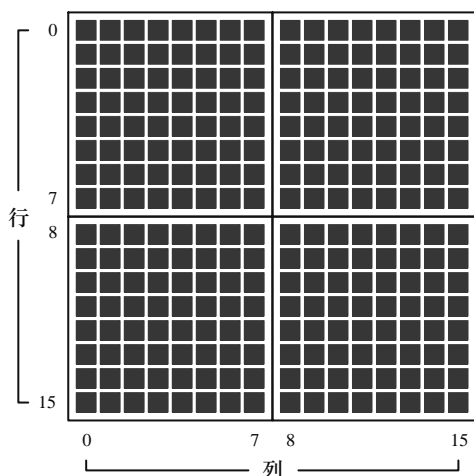


图 4.11.1  $16 \times 16$  点阵示意图

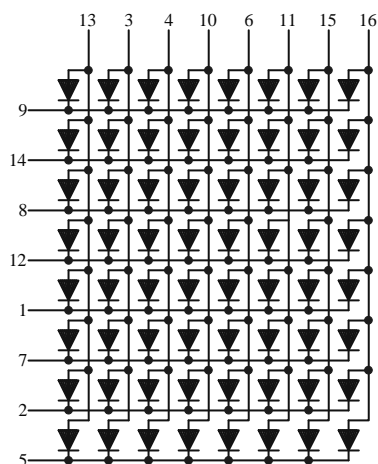


图 4.11.2 点阵内部结构图

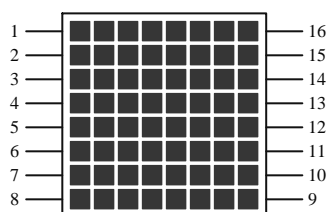


图 4.11.3 点阵外部引脚图

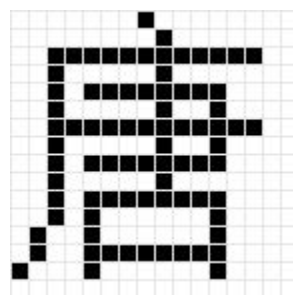


图 4.11.4 显示示例

点阵实验单元电路图如图 4.11.5 所示。由于 2803 输出反向，所以行为 1，列为 0 时对于点的 LED 点亮。

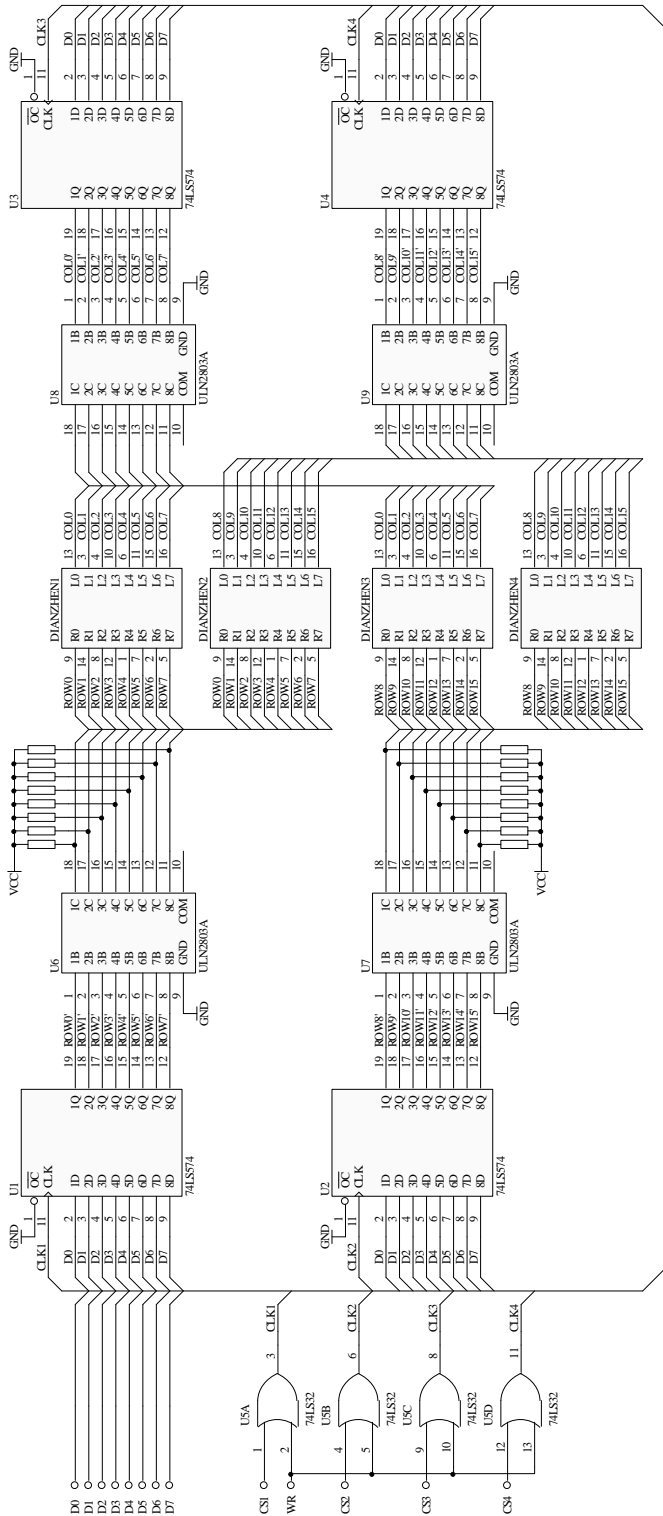


图 4.11.5 点阵实验单元电路图

点阵实验接线图如图 4.11.6 所示。

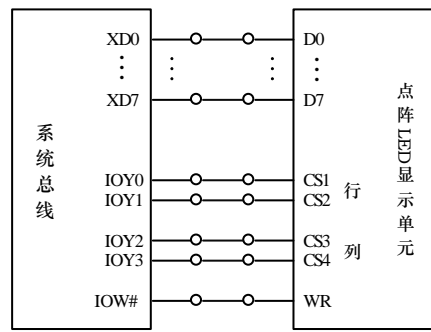


图 4.11.6 点阵显示实验接线图

4.11.4 实验步骤

- 1. 按图 4.11.6 连接实验电路图。
- 2. 编写实验程序（例程文件名为：LED16.ASM），检查无误后，编译、链接并装入系统。
- 3. 运行实验程序，观察点阵的显示，验证程序功能。
- 4. 固化实验程序，然后脱机运行。
- 5. 自己可以设计实验，使点阵显示不同的符号。

使用点阵显示符号时，必须首先得到显示符号的编码，这可以根据需要通过不同的工具获得。在本例子中，我们首先得到了显示汉字的字库文件，然后将该字库文件修改后包含到主文件中。参考 4.11.5 节所述。

实验程序清单

```
;=====
; 文件名称: LED16.ASM
; 功能描述:
;   行: CS1(600H), CS2(640H)  列: CS3(680H), CS4(6C0H)
;=====
ROW1      EQU 0600H      ;端口定义 IOY0
ROW2      EQU 0640H      ;端口定义 IOY1
COL1      EQU 0680H      ;端口定义 IOY2
COL2      EQU 06C0H      ;端口定义 IOY3
STACK1    SEGMENT STACK
          DW 256 DUP(?)
STACK1    ENDS
```

;定义为数据段

INCLUDE HZDOTht.ASM

;数据字段为汉字点阵库, 在 HZDOTht.ASM 文

件中

CODE SEGMENT

ASSUME CS:CODE, DS:DATA

START: MOV AX, DATA

MOV DS, AX

MOV DX, ROW1

MOV AL, 00H

OUT DX, AL

MOV DX, ROW2

OUT DX, AL

MOV AL, 0ffH

MOV DX, COL1

OUT DX, AL

MOV DX, COL2

OUT DX, AL

BG0: MOV AX, 160

MOV SI, OFFSET HZDOTht

BG1: CALL DISP

ADD SI, 2

DEC AX

JZ BG0

JMP BG1

;=====显示汉字子程序=====

;入口参数: SI = 存放汉字起始地址

DISP: MOV CX, 000FH

PUSH AX

ML0: PUSH CX

MOV BL, 01H

MOV CX, 0008H

ML1: MOV DX, ROW1 ;控制 0--7 行

MOV AL, 00H

OUT DX, AL

MOV AL, [SI]

NOT AL

MOV DX, COL1 ;0--7 列

OUT DX, AL

```
    INC SI
    MOV AL, [SI]
    NOT AL
    MOV DX, COL2      ;8--15 列
    OUT DX, AL
    INC SI
    MOV DX, ROW1      ;控制 0--7 行
    MOV AL, BL
    OUT DX, AL
    ROL BL, 1

    CALL DELAY
    LOOP ML1
    MOV DX, ROW1
    MOV AL, 00H
    OUT DX, AL
    MOV CX, 0008H
ML2:  MOV DX, ROW2      ;控制 8--15 行
    MOV AL, 00H
    OUT DX, AL
    MOV AL, [SI]
    NOT AL
    MOV DX, COL1      ;0--7 列
    OUT DX, AL
    INC SI
    MOV AL, [SI]
    NOT AL
    MOV DX, COL2      ;8--15 列
    OUT DX, AL
    INC SI
    MOV DX, ROW2      ;控制 8--15 行
    MOV AL, BL
    OUT DX, AL
    ROL BL, 1
    CALL DELAY
    LOOP ML2
    MOV DX, ROW2
    MOV AL, 00H
    OUT DX, AL
```




```

SUB SI, 32
POP CX
LOOP ML0
POP AX
RET
DELAY: PUSH CX           ;延时子程序
      MOV CX, 0100H
DL1:  PUSH AX
      POP AX
      LOOP DL1
      POP CX
      RET

CODE  ENDS
      END START

```

#### 4.11.5 字符提取方法

1. 将 HZDotReader 文件夹拷贝到硬盘上，然后双击文件  运行程序；
2. 在“设置”下拉菜单中选择“取模字体”选项，设置需要显示汉字的字体；

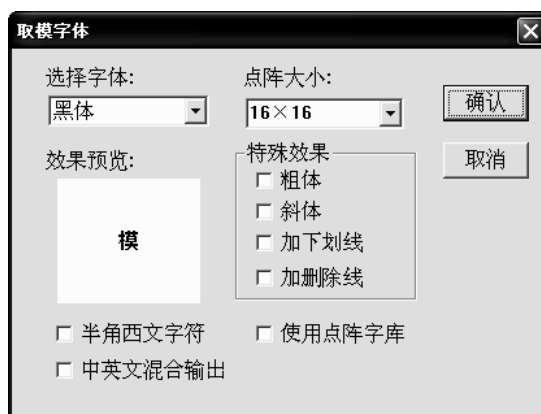


图 4.11.7 取模字体对话框

3. 在“设置”下拉菜单中选择“取模方式”选项，在本系统中选择如图所示，即以横向 8 个连续点构成一个字节，最左边的点为字节的最低位，即 BIT0，最右边的点为 BIT7。16×16

汉字按每行 2 字节，共 16 行取字模，每个汉字共 32 字节，点阵四个角取字顺序为左上角→右上角→左下角→右下角；

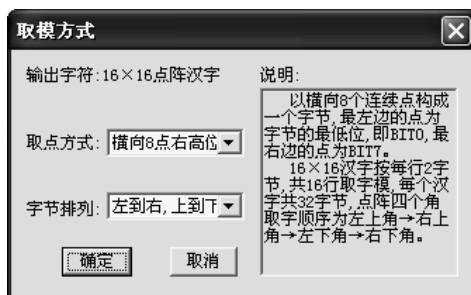


图 4.11.8 取模方式对话框

4. 在“设置”下拉菜单中选择“输出设置”选项，以设置输出格式，可以为汇编格式或 C 语言格式，根据实验程序语言而定，如图 4.11.9 所示；

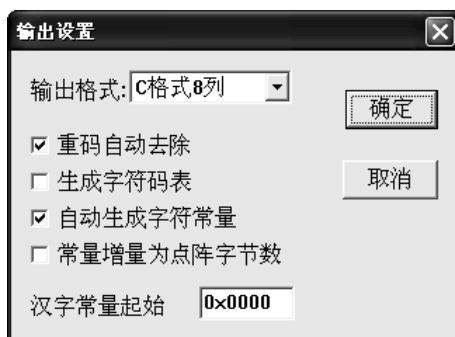


图 4.11.9 输出设置对话框

5. 点击 **字** 按钮，弹出字符输入对话框，输入“西安唐都科教仪器公司!”，如图 4.11.10 所示，然后点击输入按钮；

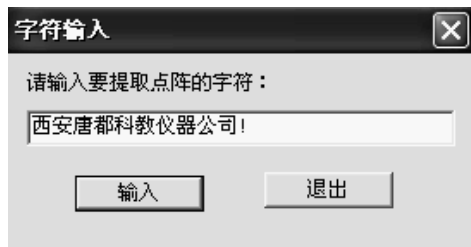


图 4.11.10 字符输入对话框

6. 字符输入后，可得到输入字符的点阵编码以及对应汉字的显示，如图 4.11.11 所示。此时可以对点阵进行编辑，方法是右键点击某一汉字，此时该汉字的编码反蓝，然后点击“编辑”下拉菜单中的“编辑点阵”选项来编辑该汉字，如图 4.11.12 所示。鼠标左键为点亮某点，鼠标右键为取消某点。若无需编辑，则进行保存，软件会将此点阵文件保存为 dot 格式；

7. 使用 Word 软件打开保存的文件，然后将字库复制到自己的程序中使用。

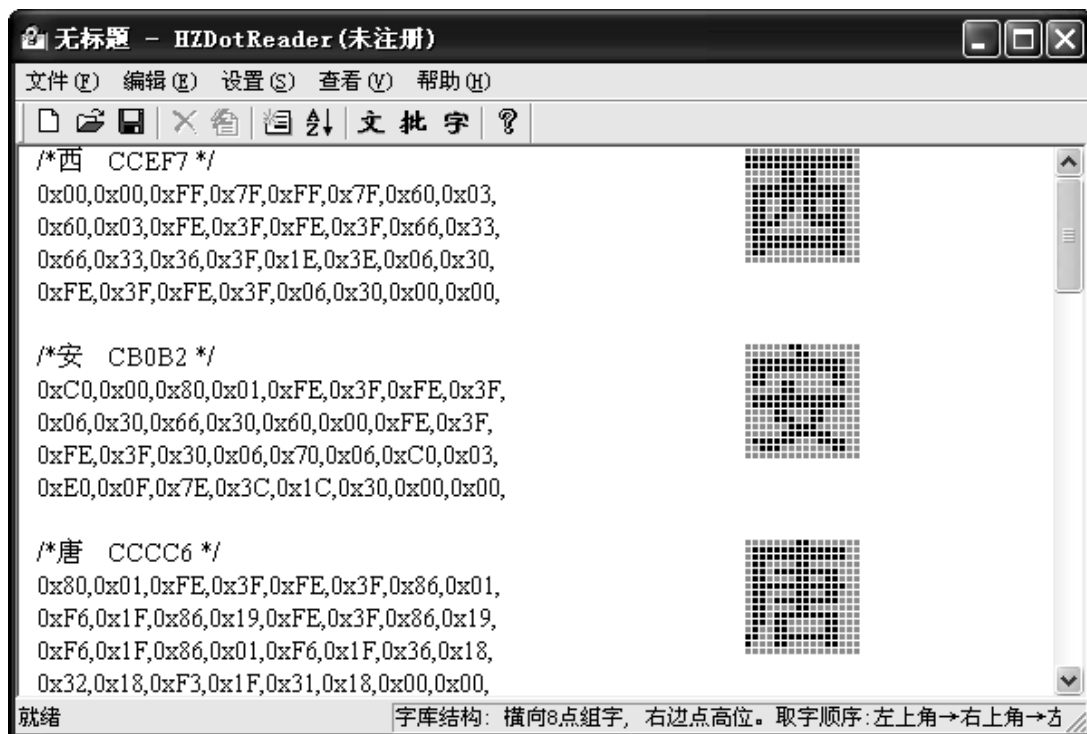


图 4.11.11 字模生成窗口

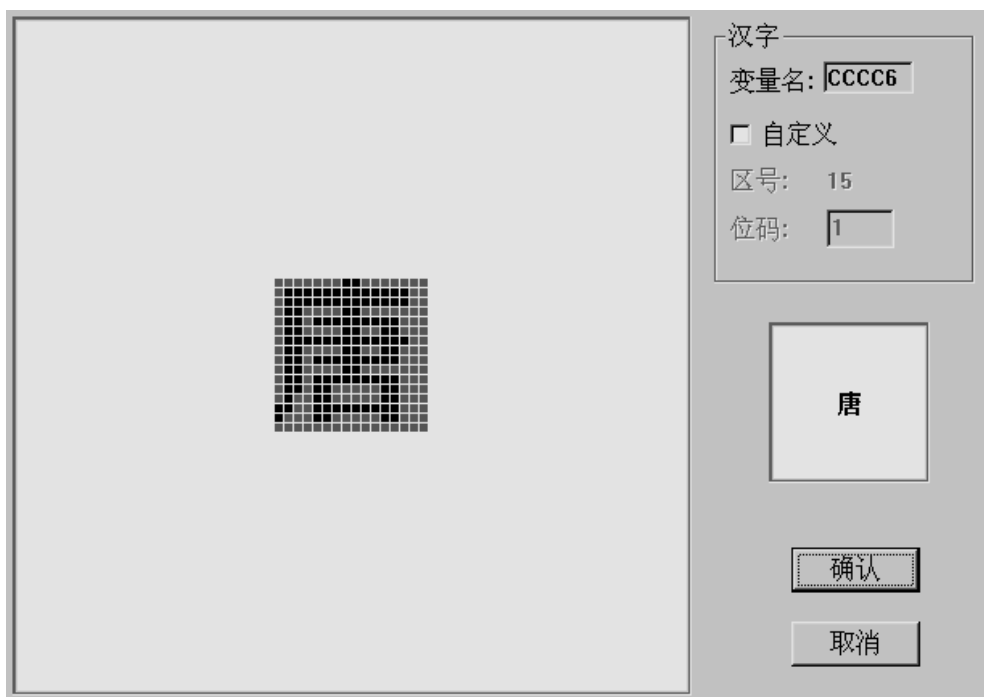


图 4.11.12 点阵编辑窗

## 4.12 图形 LCD 显示设计实验

### 4.12.1 实验目的

了解图形 LCD 的控制方法。

### 4.12.2 实验设备

PC 机一台，TD-PITE 实验装置一套，图形 LCD 液晶一块（选配）。

### 4.12.3 实验内容

本实验使用的是  $128 \times 64$  图形点阵液晶，编写实验程序，通过 8255 控制液晶，显示“唐都科教仪器公司欢迎你！”，并使该字串滚屏一周。

### 4.12.4 实验原理

#### 1. 液晶模块的接口信号及工作时序

该图形液晶内置有控制器，这使得液晶显示模块的硬件电路简单化，它与 CPU 连接的信号线如下：

表 4.12.1 时序参数说明

特性曲线	助记符	最小值	典型	最大值	单位
E 周期	tcyc	1000	-	-	ns
E 高电平宽度	twhE	450	-	-	ns
E 低电平宽度	twlE	450	-	-	ns
E 上升时间	tr	-	-	25	ns
E 下降时间	tf	-	-	25	ns
地址建立时间	tas	140	-	-	ns
地址保持时间	tah	10	-	-	ns
数据建立时间	tdsw	200	-	-	ns
数据延迟时间	tddr	-	-	320	ns
数据保持时间（写）	tdhw	10	-	-	ns
数据保持时间（读）	tdhr	20	-	-	ns

CS1、CS2：片选信号，低电平有效；

E：使能信号；

RS：数据和指令选择信号，RS=1 为 RAM 数据，RS=0 为指令数据；

R/W：读/写信号，R/W=1 为读操作，R/W=0 为写操作；

D7~D0：数据总线；

LT：背景灯控制信号，LT=1 时打开背景灯，LT=0 时关闭背景灯。

该液晶的时序参数说明如表 4.12.1 所列，读写时序图如图 4.12.1 和 4.12.2 所示。

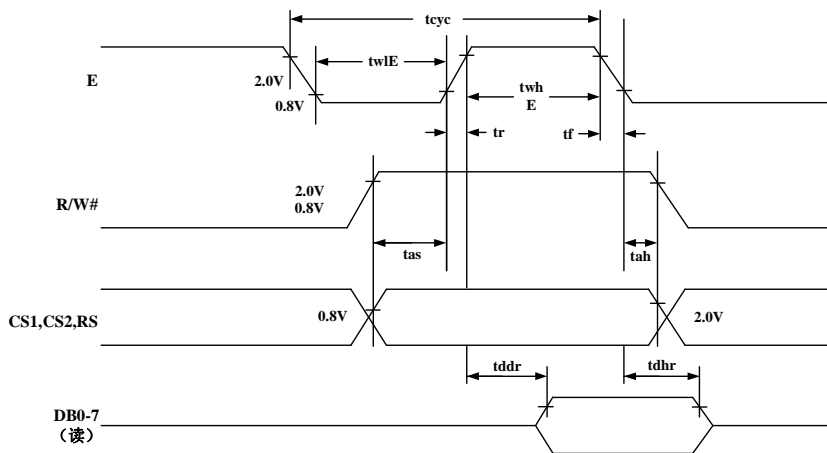


图 4.12.1 读操作时序图

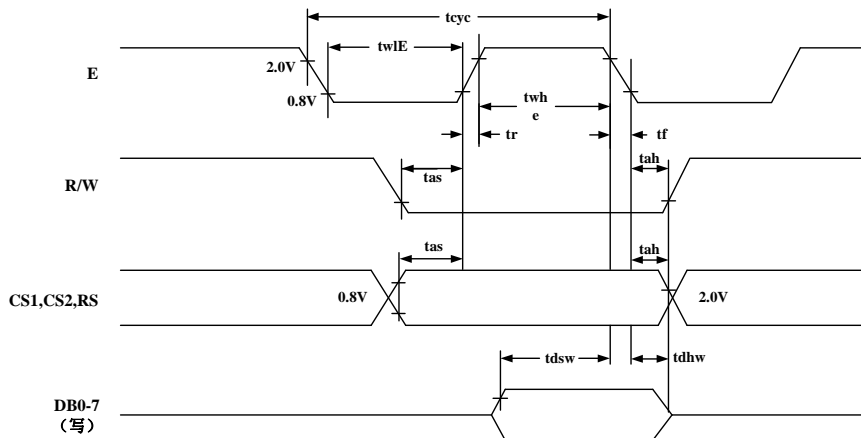


图 4.12.2 写操作时序图

## 2. 显示控制指令

显示控制指令控制着液晶控制器的内部状态，具体如表 4.12.2 所列。

表 4.12.2 显示控制命令列表

指令	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
显示 开/关	0	0	0	0	1	1	1	1	1	0/1
设置地址 (Y 地址)	0	0	0	1	Y 地址 (0~63)					
设置页 (X 地址)	0	0	1	0	1	1	1	页 (0~7)		
显示起始行 (Z 地址)	0	0	1	1	显示起始行 (0~63)					
状态读	0	1	忙	0	开/关	复位	0	0	0	0
写显示数据	1	0	写数据							
读显示数据	1	1	读数据							

显示开/关:

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

该指令设置显示开/关触发器的状态，当 D=1 为显示数据，当 D=0 为关闭显示设置。

设置地址 (Y 地址):

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0

该指令用以设置 Y 地址计数器的内容，AC5~AC0=0~63 代表某一页面上的某一单元地址，随后的一次读或写数据将在这个单元上进行。Y 地址计数器具有自动加一功能，在每次读或写数据后它将自动加一，所以在连续读写数据时，Y 地址计数器不必每次设置一次。

设置页 (X 地址):

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	0	0	1	0	1	1	1	AC2	AC1	AC0

该指令设置页面地址寄存器的内容。显示存储器共分 8 页，指令代码中 AC2~AC0 用于确定当前所要选择的页面地址，取值范围为 0~7，代表第 1~8 页。该指令指出以后的读写操作将在哪一个页面上进行。

显示起始行 (Z 地址):

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	0	0	1	1	L5	L4	L3	L2	L1	L0

该指令设置了显示起始行寄存器的内容。此液晶共有 64 行显示的管理能力，指令中的 L5~L0 为显示起始行的地址，取值为 0~63，规定了显示屏上最顶一行所对应的显示存储器的行地址。若等时间、等间距地修改显示起始行寄存器的内容，则显示屏将呈现显示内容向上或向下滚动的显示效果。

状态读:

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	0	1	忙	0	开/关	复位	0	0	0	0

状态字是 CPU 了解液晶当前状态的唯一信息渠道。共有 3 位有效位，说明如下。

忙: 表示当前液晶接口控制电路运行状态。当忙位为 1 表示正在处理指令或数据，此时接

口电路被封锁，不能接受除读状态字以外的任何操作。当忙位为 0 时，表明接口控制电路已准备好等待 CPU 的访问。

开/关：表示当前的显示状态。为 1 表示关显示状态，为 0 表示开显示状态。

复位：为 1 表示系统正处于复位状态，此时除状态读可被执行外，其它指令不可执行，此位为 0 表示处于正常工作状态。

在指令设置和数据读写时要注意状态字中的忙标志。只有在忙标志为 0 时，对液晶的操作才能有效。所以在每次对液晶操作前，都要读出状态字判断忙标志位，若不为 0 则需要等待，直到忙标志为 0 为止。

写显示数据：

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	1	0	D7	D6	D5	D4	D3	D2	D1	D0

该操作将 8 位数据写入先前确定的显示存储单元中。操作完成后列地址计数器自动加一。

读显示数据：

格式	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	1	1	D7	D6	D5	D4	D3	D2	D1	D0

该操作将读出显示数据 RAM 中的数据，然后列地址计数器自动加一。

### 3. 液晶显示单元电路图

如图 4.72 所示，调节 10K 微调可以改变液晶显示的对比度。

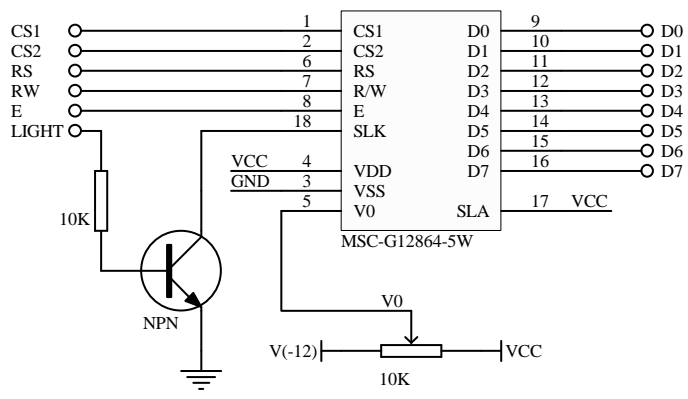


图 4.12.3 液晶显示电路图

## 4. 12. 5 实验步骤

1. 按照图 4.12.4 连接实验接线图。
2. 得到需显示汉字或图形的显示数据，这里需要得到“唐都科教仪器公司欢迎你!”的字模。
3. 编写实验程序（例程文件名为：CLCD.C），编译、链接无误后装入系统。
4. 运行实验程序，验证程序功能。

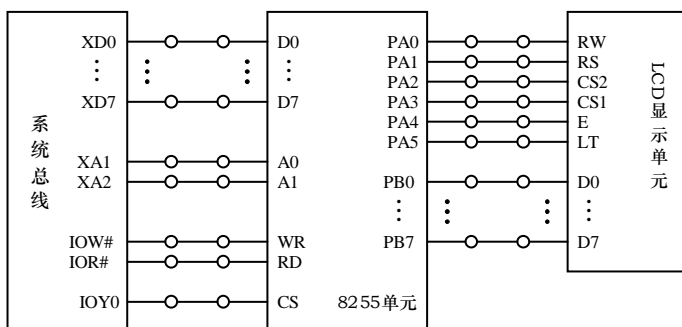


图 4.12.4 液晶实验线路图

## 实验程序清单

```

/*****

```

```

* 文件名: CLCD.C

```

```

* 功能描述: 液晶控制实验

```

```

*      显示"唐都科教仪器公司欢迎您!", 滚屏一周

```

```

*****/

```

```

#include <conio.h>

```

```

#include "lcd.h"

```

```

void delay(int time);

```

```

void query(char cmd);

```

```

void openlight(char cmd);

```

```

void closelight(char cmd);

```

```

void clear(char cmd,int x);

```

```

void write(char cmd,char data);

```

```

void writehz(char xadd,char yadd,char cmd,char* hz);

```

```

int MY8255_A    = 0x0600;

```

```

int MY8255_B    = 0x0602;

```

```

int MY8255_C    = 0x0604;

```

```

int MY8255_MODE = 0x0606;

```

```

char cmd1 = 0x04;

```

```

char cmd2 = 0x08;

```

```

char data;

```

```

char xadd;

```



```
char yadd;
```

```
////////////////////////////////////
```

```
void main()
```

```
{
```

```
    int x;
```

```
    outp(MY8255_MODE, 0x80);
```

```
    write(cmd1, 0x3f);
```

```
    //Display On 打开显示
```

```
    write(cmd2, 0x3f);
```

```
    write(cmd1, 0xc0);
```

```
    //设置起始行
```

```
    write(cmd2, 0xc0);
```

```
    for(x=0;x<8;x++)
```

```
    //清屏
```

```
    {
```

```
        clear(cmd1, x);
```

```
        clear(cmd2, x);
```

```
    }
```

```
    //写汉字
```

```
    writehz(0xba, 0x40, cmd1, tang);
```

```
    writehz(0xba, 0x50, cmd1, du);
```

```
    writehz(0xba, 0x60, cmd1, ke);
```

```
    writehz(0xba, 0x70, cmd1, jiao);
```

```
    writehz(0xba, 0x40, cmd2, yi);
```

```
    writehz(0xba, 0x50, cmd2, qi);
```

```
    writehz(0xba, 0x60, cmd2, gong);
```

```
    writehz(0xba, 0x70, cmd2, si);
```

```
    writehz(0xbc, 0x60, cmd1, huan);
```

```
    writehz(0xbc, 0x70, cmd1, ying);
```

```
    writehz(0xbc, 0x40, cmd2, nin);
```

```
    writehz(0xbc, 0x50, cmd2, gantan);
```

```
    delay(50);
```

```
    for(x=0xc1;x<=0xFF;x++)
```

```
    {
```

```
        write(cmd1, x);
```

```
        write(cmd2, x);
```

```

        delay(20);
    }
    while(1);
}
////////////////////////////////////

void write(char cmd,char data)          //写命令或数据子程序
{
    outp(MY8255_B, data);
    cmd = cmd | 0x10;
    outp(MY8255_A, cmd);
    cmd = cmd & 0xEF;
    outp(MY8255_A, cmd);
}

void writehz(char xadd,char yadd,char cmd,char* hz)
{
    int x,y;                                //写汉字子程序 16*16

    write(cmd,xadd);
    query(cmd+1);

    write(cmd,yadd);
    query(cmd+1);

    for(x=0;x<2;x++)
    {
        for(y=0;y<16;y++)
        {
            data = hz[y+(x*16)];
            write(cmd+2,data);
            query(cmd+1);
        }

        xadd++;
        write(cmd,xadd);
        query(cmd+1);

        write(cmd,yadd);

```

```
        query(cmd+1);
    }
}

void openlight(char cmd)                //打开背景灯
{
    cmd = cmd | 0x20;
    outp(MY8255_A, cmd);
}

void closelight(char cmd)              //关闭背景灯
{
    cmd = cmd & 0xdf;
    outp(MY8255_A, cmd);
}

void clear(char cmd,int x)              //清一页屏幕子程序
{
    int y;

    xadd = 0xb8+x;
    write(cmd,xadd);
    query(cmd+1);

    yadd = 0x40;
    write(cmd,yadd);
    query(cmd+1);

    for(y=0;y<64;y++)
    {
        data = 0x00;
        write(cmd+2,data);
        query(cmd+1);
    }
}

void query(char cmd)                   //查询子程序
{
    outp(MY8255_MODE, 0x82);
```

```
cmd = cmd | 0x10;
outp(MY8255_A, cmd);
cmd = cmd & 0xEF;
outp(MY8255_A, cmd);

while( inp(MY8255_B) & 0x80)
{
    cmd = cmd | 0x10;
    outp(MY8255_A, cmd);
    cmd = cmd & 0xEF;
    outp(MY8255_A, cmd);
}

outp(MY8255_MODE, 0x80);
}

void delay(int time)
{
    int i;
    int j;
    for(i=0;i<=time;i++)
    {
        for(j=0;j<=0x1000;j++);
    }
}
```

### 4.13 步进电机实验

#### 4.13.1 实验目的

掌握步进电机的控制方法。

#### 4.13.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

#### 4.13.3 实验内容

编写实验程序，利用 8255 的 B 口来控制步进电机的运转。

#### 4.13.4 实验原理

使用开环控制方式能对步进电机的转动方向、速度和角度进行调节。所谓步进，就是指每给步进电机一个递进脉冲，步进电机各绕组的通电顺序就改变一次，即电机转动一次。根据步进电机控制绕组的多少可以将电机分为三相、四相和五相。本实验系统所采用的步进电机为四相八拍电机。

励磁线圈如图 4.13.1 所示，励磁顺序如表 4.13.1 所列。

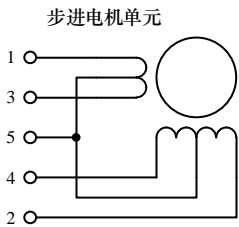


图 4.13.1 励磁线圈

表 4.13.1 励磁顺序

	步 序							
	1	2	3	4	5	6	7	8
5	+	+	+	+	+	+	+	+
4	-	-						-
3		-	-	-				
2				-	-	-		
1						-	-	-

**表 4.13.2 PB 端口各线的电平在各步中的情况**

步序	PB3	PB2	PB1	PB0	对应 B 口输出值
1	0	0	0	1	01H
2	0	0	1	1	03H
3	0	0	1	0	02H
4	0	1	1	0	06H
5	0	1	0	0	04H
6	1	1	0	0	0CH
7	1	0	0	0	08H
8	1	0	0	1	09H

The circuit diagram illustrates a 4-bit parallel-to-serial converter. It features a 4-bit parallel input with four 4.7K resistors (RMTN, RMT1, RMT2, RMT3, RMT4) connected to VCC. Each input line is connected to an LED (DMTN, DMTA, DMTB, DMTD) through a 4.7K resistor. The outputs of the LEDs are connected to the inputs of a ULN2803A (1C, 2C, 3C, 4C, 5C, 6C, 7C, 8C, 9C, 10C, 11C, 12C, 13C, 14C, 15C, 16C, 17C, 18C). The ULN2803A is connected to a +12V supply and ground. The output of the ULN2803A is connected to the input of a 74LS04 (A), which is also connected to ground. The output of the 74LS04 is connected to the output of the parallel-to-serial converter (N).

图 4.13.2 驱动电路原理图

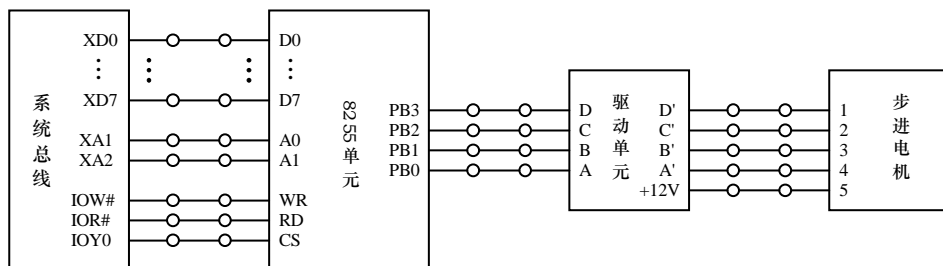


图 4.13.3 步进电机实验参考接线图

### 4.13.5 实验步骤

1. 按图 4.13.3 连接线路。
2. 编写实验程序（例程文件名为：BUJIN.ASM），编译、链接后装入系统。
3. 运行程序，观察实验现象。

注意：步进电机不使用时请断开连接器，以免误操作使电机过热损坏。

#### 实验程序清单

```

;=====
; 文件名: BUJIN.ASM
; 功能描述: 步进电机控制实验
;=====

IOY0      EQU    0600H          ;片选 IOY0 对应的端口始地址
MY8255_A   EQU    IOY0+00H*2    ;8255 的 A 口地址
MY8255_B   EQU    IOY0+01H*2    ;8255 的 B 口地址
MY8255_C   EQU    IOY0+02H*2    ;8255 的 C 口地址
MY8255_MODE EQU    IOY0+03H*2    ;8255 的控制寄存器地址

SSTACK    SEGMENT STACK
            DW 256 DUP(?)
SSTACK    ENDS
DATA      SEGMENT
TABDT     DB 01H,03H,02H,06H,04H,0CH,08H,09H
DATA      ENDS
CODE      SEGMENT
            ASSUME CS:CODE, DS:DATA
START:     MOV AX, DATA
            MOV DS, AX
MAIN:      MOV AL, 90H           ; 控制 B 口工作于方式 0，输出
            MOV DX, MY8255_MODE
            OUT DX, AL
A1:        MOV BX, OFFSET TABDT
            MOV CX, 0008H
A2:        MOV  AL,[BX]
            MOV DX, MY8255_B     ; 写 B 口
            OUT  DX, AL

```

```
        CALL DALLY          ; 控制步进电机的转速
        INC BX
        LOOP A2
        JMP     A1
DALLY:  PUSH CX
        MOV CX,8000H
A3:     PUSH AX
        POP     AX
        LOOP A3
        POP     CX
        RET
CODE   ENDS
        END START
```



## 4.14 直流电机闭环调速实验

### 4.14.1 实验目的

1. 了解直流电机闭环调速的方法。
2. 掌握 PID 控制规律及算法。
3. 了解计算机在控制系统中的应用。

### 4.14.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 4.14.3 实验内容

直流电机闭环调速实验原理如图 4.14.1 所示。

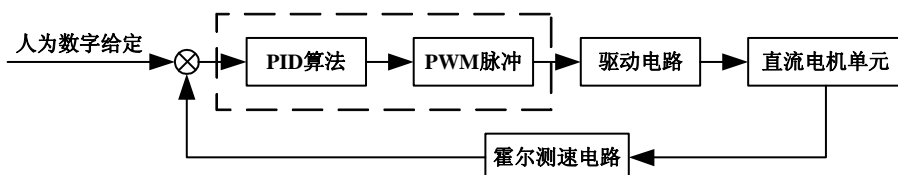


图 4.14.1 直流电机闭环调速实验原理图

如图 4.14.1 所示，人为数字给定直流电机转速，与霍尔测速得到的直流电机转速（反馈量）进行比较，其差值经过 PID 运算，将得到控制量并产生 PWM 脉冲，通过驱动电路控制直流电机的转动，构成直流电机闭环调速控制系统。

实验系统中直流电机电路原理图如图 4.14.2 所示。

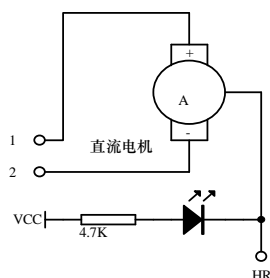


图 4.14.2 直流电机电路原理图

4.14.4 实验步骤

- 1. 根据图 4.14.4 连接实验线路图。
- 2. 参考图 4.14.3 的流程图编写实验程序（例程文件名为：ZHILIU.ASM），实验参数取值范围见表 4.14.1，检查无误后编译、链接并装入系统。

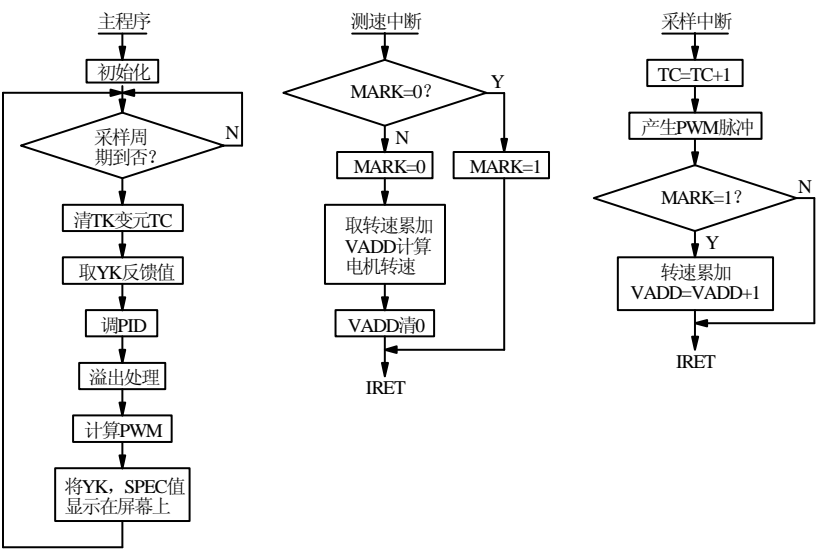


图 4.14.3 直流电机闭环调速实验流程图

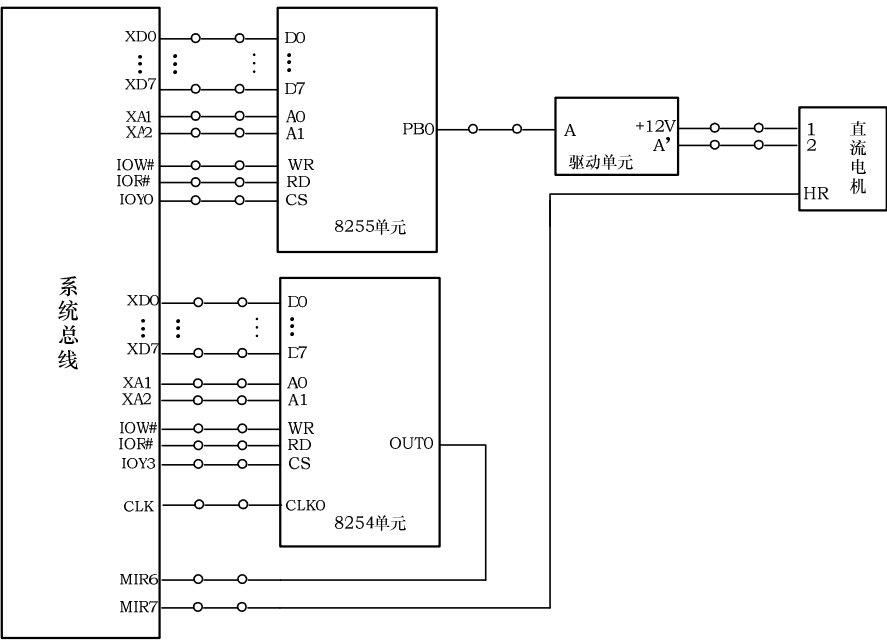





图 4.14.4 直流电机闭环调速实验参考接线图

3. 点击按钮，启动 86 专用图形界面。
4. 在专用图形界面中，点击，运行程序，观察电机转速及示波器上给定值与反馈值的波形。
5. 点击按钮，暂停程序运行，根据实验波形分析直流电机的响应特性。
6. 改变参数 IBAND、KPP、KII、KDD 的值后再观察其响应特性，选择一组较好的控制参数并填入下表。

参 数 项 目	IBAND	KPP	KII	KDD	超调	稳定时间<2%
1：例程中参数响应特性	0060H	1060H	0010H	0020H	15%	4.8 秒
2：去掉 IBAND	0000H	1060H	0010H	0020H		
3：自测一组较好参数						

注：实验中给定值、反馈值都为单极性，屏幕最底端对应值为 00H，最顶端对应值为 FFH，对于时间刻度值由于采样周期不同存在以下关系：

实际时间（秒）=  $n(\text{实际刻度值}) \times \text{采样周期}$

控制量具有双极性，00H~7FH 为负值，80H~FFH 为正值。

直流电机闭环调速实验中，电机转速范围为 6 转/秒~48 转/秒。即：给定值 SPEC 范围约在 06H~30H 之间。示例程序中给定 SPEC = 30H 为 48 转/秒。TS = 14H，由于 8253 OUT2 接 IRQ6 中断为 1ms，故采样周期=14H×1ms = 0.02s。如实际刻度值 n = 100，则实际响应时间 = 0.02×100 = 2s。

实验现象结果如图 4.14.5 所示：

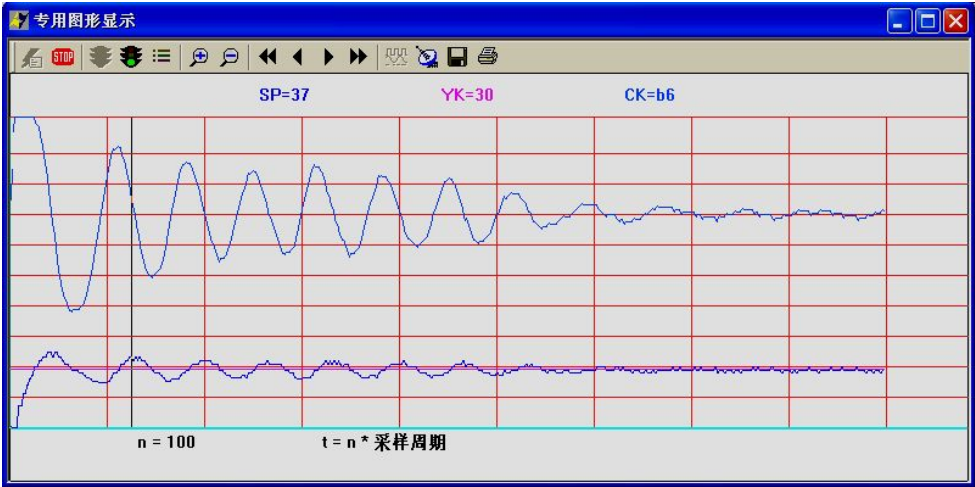


图 4.14.5 直流电机闭环调速实验结果图

## 实验程序清单

```

;=====
; 文件名: ZHILIU.ASM
; 功能描述: 直流电机闭环调速实验, 通过专用图形显示界面
;           观察实验现象。
;           IOY0--8255, IOY3--8254
;=====

```

```
SSTACK  SEGMENT STACK
```

```
        DW 64 DUP(?)
```

```
        TOP LABEL WORD
```

```
SSTACK  ENDS
```

```
DATA SEGMENT
```

```
TS      DB 14H
```

```
;采样周期
```

```
SPEC    DW 0030H
```

```
;给定电机转速 30H 为 48 转/秒
```

```
IBAND   DW 0060H
```

```
;PID 算法中积分分离值
```

```
KPP     DW 1060H
```

```
;PID 算法中比例项系数值
```

```
KII     DW 0010H
```

```
;PID 算法中积分项系数值
```

```
KDD     DW 0020H
```

```
;PID 算法中微分项系数值
```

```
CH1     DB ?
```

```
;专用图形显示的 CH1 通道
```

```
CH2     DB ?
```

```
;专用图形显示的 CH2 通道
```

```
CH3     DB ?
```

```
;专用图形显示的 CH3 通道
```

```
YK      DW ?
```

```
;电机转速反馈量
```

```
CK      DB ?
```

```
;电机转速控制量
```

```
VADD    DW ?
```

```
ZV      DB ?
```

```
ZVV     DB ?
```

```
TC      DB ?
```

```
FPWM    DB ?
```

```
CK_1    DB ?
```

```
EK_1    DW ?
```

```
AEK_1   DW ?
```

```
BEK     DW ?
```

```
AAAA    DB ?
```

```
VAA     DB ?
```

```
BBB     DB ?
```

```
VBB     DB ?
```

```
MARK    DB ?
```

```
R0      DW ?
```

```

R1      DW ?
R2      DW ?
R3      DW ?
R4      DW ?
R5      DW ?
R6      DW ?
R7      DB ?
R8      DW ?
DATA    ENDS

CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
MAIN:   CALL INIT           ;初始化
        STI
M1:     MOV AL, TS           ;判断 Ts=Tc ?
        SUB AL, TC
        JNC M1
        MOV TC, 00H         ;得到 Yk
        MOV AL, ZVV
        MOV AH, 00H
        MOV YK, AX
        CALL PID            ;调用 PID 计算控制量 CK
        MOV AL, CK          ;根据 CK 产生 PWM 脉冲
        SUB AL, 80H
        JC  IS0
        MOV AAAA, AL
        JMP COU
IS0:    MOV AL, 10H
        MOV AAAA, AL
COU:    MOV AL, 7FH
        SUB AL, AAAA
        MOV BBB, AL
        MOV AX, SPEC        ;将给定量 SPEC 存入 CH1
        MOV CH1, AL
        MOV AX, YK          ;将反馈量 YK 存入 CH2
        MOV CH2, AL
        MOV AL, CK          ;将控制量 CK 存入 CH3

```

```
        MOV CH3, AL
        CALL PUT_COM      ;调用 PUT_COM 显示给定、反馈与控制量的波形
        JMP M1
PUT_COM: MOV DX, 03FDH
WAIT1:  IN  AL, DX
        TEST AL, 20H
        JZ  WAIT1
        MOV DX, 03F8H
        MOV AL, CH2
        OUT DX, AL
        MOV DX, 03FDH
WAIT2:  IN  AL, DX
        TEST AL, 20H
        JZ  WAIT2
        MOV DX, 03F8H
        MOV  AL, CH1
        OUT  DX, AL
        MOV DX, 03FDH
WAIT3:  IN  AL, DX
        TEST AL, 20H
        JZ  WAIT3
        MOV DX, 03F8H
        MOV  AL, CH3
        OUT  DX, AL
        RET
INIT:   CLI
        PUSH DS
        XOR AX, AX
        MOV DS, AX
        MOV AX, OFFSET IRQ6      ;8259 IRQ6(T0:1ms)
        MOV SI, 0038H
        MOV [SI], AX
        MOV AX, CS
        MOV SI, 003AH
        MOV [SI], AX
        MOV AX, OFFSET IRQ7      ;8259 IRQ7(INT0:HR-OUT,COUNT-VVV)
        MOV SI, 003CH
        MOV [SI], AX
        MOV AX, CS
```

```
MOV SI, 003EH
MOV [SI], AX
POP DS
MOV AL, 2FH           ;允许 IRQ6,IRQ7
OUT 21H, AL
MOV VADD, 0000H       ;变量初始化
MOV ZV, 00H
MOV ZVV, 00H
MOV CK, 00H
MOV YK, 0000H
MOV CK_1, 00H
MOV EK_1, 0000H
MOV AEK_1, 0000H
MOV BEK, 0000H
MOV BBB, 00H
MOV VBB, 00H
MOV R0, 0000H
MOV R1, 0000H
MOV R2, 0000H
MOV R3, 0000H
MOV R4, 0000H
MOV R5, 0000H
MOV R6, 0000H
MOV R7, 00H
MOV R8, 0000H
MOV MARK, 00H
MOV FPWM, 01H
MOV AAAA, 7FH
MOV VAA, 7FH
MOV TC, 00H
MOV DX, 606H
MOV AL, 90H           ;初始化 8255-B 口
OUT DX, AL
MOV DX, 602H
MOV AL, 00H
OUT DX, AL
MOV DX, 6C6H
MOV AL, 36H           ;8254 计数器 0 的输出 OUT0
OUT DX, AL
```

```
        MOV DX, 6C0H
        MOV AL, 0E8H           ;定时 1ms
        OUT DX, AL
        MOV AL, 03H
        OUT DX, AL
        RET
IRQ7:   NOP                   ;7 号中断程序, 计算转速
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSHF
        MOV AL, MARK
        CMP AL, 01H
        JZ  IN1
        MOV MARK, 01H
IN2:    NOP
        MOV AL, 20H           ;中断返回, 关闭 IRQ7
        OUT 20H, AL
        POPF
        POP DX
        POP CX
        POP BX
        POP AX
        IRET
IN1:    MOV MARK, 00H
        CALL VV
        MOV AL, ZV
        MOV ZVV, AL
        JMP IN2
VV:     MOV DX, 0000H         ;计算电机转速
        MOV AX, 03E8H
        MOV CX, VADD
        CMP CX, 0000H
        JZ  MM1
        DIV CX
MM:     MOV ZV, AL
        MOV VADD, 0000H
MM1:    RET
```



```
IRQ6:  PUSH AX
        PUSH DX
        PUSHF
        INC TC
        CALL KJ
        CLC
        CMP MARK, 01H
        JC  TT1
        INC VADD
        CMP VADD, 0700H           ;转速值溢出, 赋极值
        JC  TT1
        MOV VADD, 0700H
        MOV MARK, 00H
TT1:    NOP
        MOV AL, 20H               ;中断返回, 关闭 IRQ6
        OUT 20H, AL
        POPF
        POP DX
        POP AX
        IRET
KJ:     NOP                       ;PWM 发生子程序
        PUSH AX
        CMP FPWM, 01H             ;FPWM 为 1, 产生 PWM 的高电平
        JNZ TEST2
        CMP VAA, 00H
        JNZ ANOT0
        MOV FPWM, 02H
        MOV AL, BBB
        CLC
        RCR AL, 01H
        MOV VBB, AL
        JMP TEST2
ANOT0:  DEC VAA
        MOV DX, 0602H             ;输出高电平
        MOV AL, 01H
        OUT DX, AL
TEST2:  CMP FPWM, 02H             ;FPWM 为 2, 产生 PWM 的低电平
        JNZ OUTT
        CMP VBB, 00H
```

```

    JNZ BNOT0
    MOV FPWM, 01H
    MOV AL, AAAA
    CLC
    RCR AL, 01H
    MOV VAA, AL
    JMP OUTT
BNOT0: DEC VBB
    MOV DX, 0602H           ;输出低电平
    MOV AL, 00H
    OUT DX, AL
OUTT:  POP AX
    RET
;=====
;PID 算法子程序
;根据 SPEC, KPP, KII, KDD 及 YK 计算对应控制量 CK
;=====
PID:   MOV AX, SPEC           ;求偏差 EK
    SUB AX, YK
    MOV R0, AX
    MOV R1, AX               ;求偏差的变化量 AEK
    SUB AX, EK_1
    MOV R2, AX
    SUB AX, AEK_1            ;求 BEK
    MOV BEK, AX
    MOV R8, AX
    MOV AX, R1
    MOV EK_1, AX
    MOV AX, R2
    MOV AEK_1, AX
    TEST R1, 8000H
    JZ EK1
    NEG R1
EK1:   MOV AX, R1             ;根据积分分离值, 判是否积分
    SUB AX, IBAND
    JC II
    MOV R3, 00H
    JMP DDD
II:    MOV AL, TS             ;计算积分项的值

```

```
MOV AH, 00H
MOV CX, R1
MUL CX
MOV CX, KII
DIV CX
MOV R3, AX
TEST R0, 8000H
JZ DDD
NEG R3
DDD: TEST BEK, 8000H           ;计算微分项的值
JZ DDD1
NEG BEK
DDD1: MOV AX, BEK
MOV CX, KDD
MUL CX
PUSH AX
PUSH DX
MOV AL, TS
MOV AH, 00H                   ;将微分项缩小 8 倍，防止溢出
MOV CX, 0008H
MUL CX
MOV CX, AX
POP DX
POP AX
DIV CX
MOV R4, AX
TEST R8, 8000H
JZ DD1
NEG R4
DD1: MOV AX, R3                ;积分项和微分项相加，判溢出
ADD AX, R4
MOV R5, AX
JO L9
L2: MOV AX, R5
ADD AX, R2
MOV R6, AX
JO L3
L5: MOV AX, R6                ;计算比例项
MOV CX, KPP
```

```

IMUL CX
MOV CX, 1000H
IDIV CX
MOV  CX, AX
RCL  AH, 01H
PUSHF
RCR  AL, 01H
POPF
JC  LLL1           ;判溢出，溢出赋极值
CMP  CH, 00H
JZ  LLL2
MOV  AL, 7FH
JMP  LLL2
LLL1: CMP  CH, 0FFH
JZ  LLL2
MOV  AL, 80H
LLL2: MOV  R7, AL           ;CK=CK+CK_1
      ADD AL, CK_1
      JO  L8
L18:  MOV CK_1, AL
      ADD AL, 80H
      MOV CK, AL
      RET                ;PID 子程序返回
L8:   TEST R7, 80H        ;溢出处理程序
      JNZ L17
      MOV AL, 7FH
      JMP L18
L17:  MOV AL, 80H
      JMP L18
L9:   TEST R3, 8000H
      JNZ L1
      MOV  R5, 7FFFH
      JMP L2
L1:   MOV R5, 8000H
      JMP L2
L3:   TEST R2, 8000H
      JNZ L4
      MOV R6, 7FFFH
      JMP L5

```

```

L4:    MOV R6, 8000H
      JMP L5
CODE  ENDS
      END START

```

表 4.14.1 实验程序参数表

符号	单位	取值范围	名 称 及 作 用
TS	MS	00H-FFH	采样周期:决定数据采集处理快慢程度
SPEC	N/s	06H-30H	给定:即要求电机达到的转速值
IBAND		0000H-007FH	积分分离值:PID 算法中积分分离值
KPP		0000H-1FFFH	比例系数:PID 算法中比例项系数值
KII		0000H-1FFFH	积分系数:PID 算法中积分项系数值
KDD		0000H-1FFFH	微分系数:PID 算法中微分项系数值
YK	N/s	0000H-0042H	反馈:通过霍尔元件反馈算出的电机转速反馈值
CK		00H-FFH	控制量:PID 算法产生用于控制的量
VADD		0000H-FFFFH	转速累加单元:记录霍尔输出脉冲用于转速计算
ZV		00H-FFH	转速计算变量
ZVV		00H-FFH	转速计算变量
TC		00H-FFH	采样周期变量
FPWM		00H-01H	PWM 脉冲中间标志位
CK_1		00H-FFH	控制量变量:记录上次控制量值
EK_1		0000H-FFFFH	PID 偏差: $E(K)=SPEC(K)-YK(K)$
AEK_1		0000H-FFFFH	$\Delta E(K)=E(K)-E(K-1)$
BEK		0000H-FFFFH	$\Delta E(K)=\Delta E(K)-\Delta E(K-1)$
AAAA		00H-FFH	用于 PWM 脉冲高电平时间计算
VAA		00H-FFH	AAAA 变量
BBB		00H-FFH	用于 PWM 脉低冲电平时间计算
VBB		00H-FFH	BBB 变量
MARK		00H-01H	
R0----R8			PID 计算用变量

## 4.15 温度闭环控制实验

### 4.15.1 实验目的

1. 了解温度调节闭环控制方法。
2. 掌握 PID 控制规律及算法。

### 4.15.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 4.15.3 实验内容

温度闭环控制原理如图 4.15.1 所示。人为数字给定一个温度值，与温度测量电路得到的温度值（反馈量）进行比较，其差值经过 PID 运算，将得到控制量并产生 PWM 脉冲，通过驱动电路控制温度单元是否加热，从而构成温度闭环控制系统。

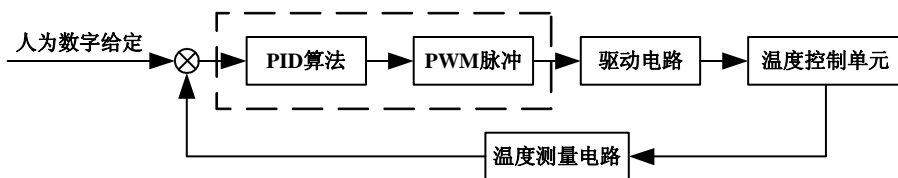
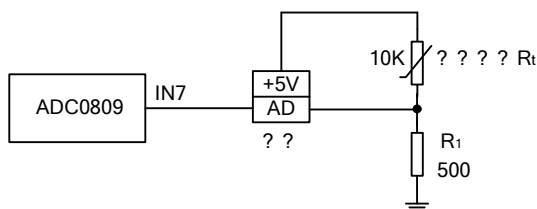


图 4.15.1 温度控制实验原理图

温度控制单元中由 7805 与一个  $24\Omega$  的电阻构成回路，回路电流较大使得 7805 芯片发热。用热敏电阻测量 7805 芯片的温度可以进行温度闭环控制实验。由于 7805 裸露在外，散热迅速。实验控制的温度最佳范围为  $50\sim 70^{\circ}\text{C}$ 。

### 4.15.4 实验原理

实验电路中采用的是 NTC MF58-103 型热敏电阻，实验电路连接如下：



温度值与对应 AD 值的计算方法如下：

25℃: $R_t = 10K$	$V_{AD} = 5 \times 500 / (10000 + 500) = 0.238(V)$	对应 AD 值: 0CH
30℃: $R_t = 5.6K$	$V_{AD} = 5 \times 500 / (5600 + 500) = 0.410(V)$	对应 AD 值: 15H
40℃: $R_t = 3.8K$	$V_{AD} = 5 \times 500 / (3800 + 500) = 0.581(V)$	对应 AD 值: 1EH
50℃: $R_t = 2.7K$	$V_{AD} = 5 \times 500 / (2700 + 500) = 0.781(V)$	对应 AD 值: 28H
60℃: $R_t = 2.1K$	$V_{AD} = 5 \times 500 / (2100 + 500) = 0.962(V)$	对应 AD 值: 32H
100℃: $R_t = 900$	$V_{AD} = 5 \times 500 / (900 + 500) = 1.786(V)$	对应 AD 值: 5AH
.....		

测出的 AD 值是程序中数据表的相对偏移，利用这个值就可以找到相应的温度值。例如测出的 AD 值为 5AH=90，在数据表中第 90 个数为 64H，即温度值：100℃。

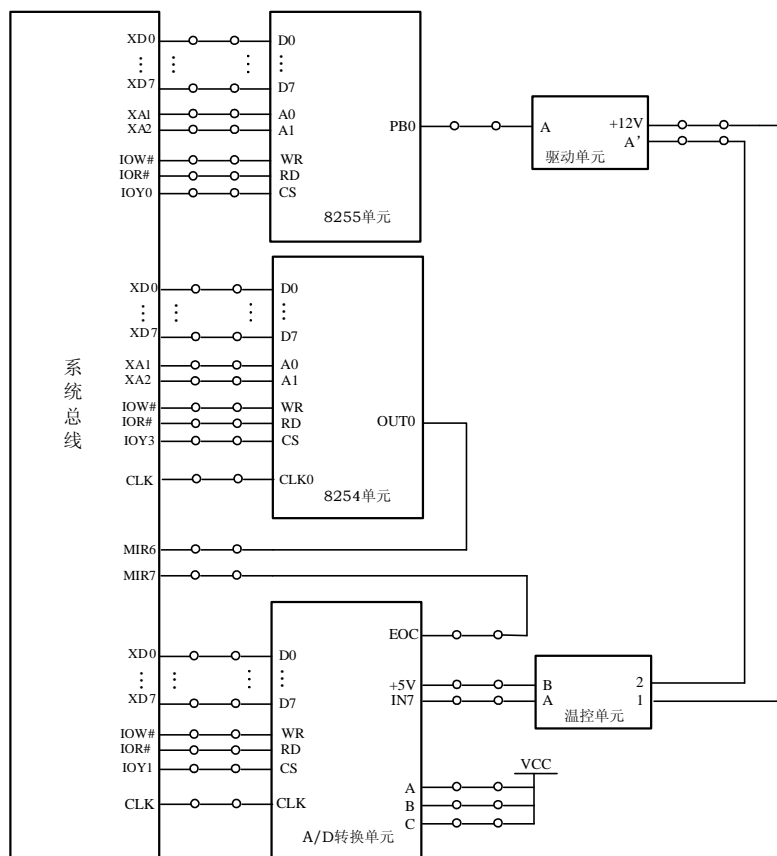


图 4.15.2 温度控制实验线路图

4.15.5 实验步骤

- 1. 实验接线图如图 4.15.2 所示，按图连接实验线路图。
- 2. 编写实验程序（例程文件名为：WENDU.ASM），实验参数取值范围见表 4.15.1，编译、链接后装入系统。
- 3. 下载程序完毕，打开专用图形界面，然后运行程序。
- 4. 观察响应曲线。
- 5. 改变 PID 参数 IBAND、KPP、KII、KDD，重复实验，观察实验现象，找出合适的参数并记录。

注：实验中给定值、反馈值都为单极性，屏幕最底端对应值为 00H，最顶端对应值为 FFH，对于时间刻度值由于采样周期不同存在以下关系：

实际时间（秒）= n(实际刻度值) × 采样周期

控制量具有双极性，00H~7FH 为负值，80H~FFH 为正。

温度闭环控制实验中，温度单元的 7805 控制范围的最佳温度范围为 50℃~70℃，不要过高。即给定值 SPEC 范围约在 14H(20℃)~46H(70℃)之间。示例程序中 SPEC = 32H 为 50℃。TS = 64H，由于 8253 OUT2 接 IRQ6 中断为 10ms，故采样周期 = 64H×10ms = 1s；如实际刻度值 n = 100，则实际响应时间(秒) = 1×100=100s。

实验现象结果如图 4.15.3 所示：

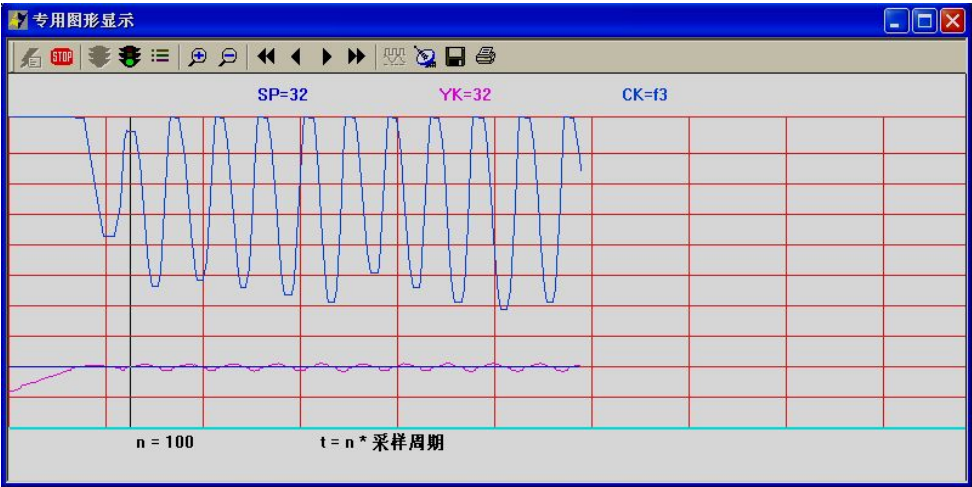


图 4.15.3 温度闭环控制实验结果图

表 4.15.1 实验参数取值范围

符号	单位	取值范围	名 称 及 作 用
TS	10mS	00H-FFH	采样周期:决定数据采集处理快慢程度
SPEC	℃	14H-46H	给定:要求达到的温度值
IBAND		0000H-007FH	积分分离值:PID 算法中积分分离值



KPP		0000H-1FFFH	比例系数:PID 算法中比例项系数值
KII		0000H-1FFFH	积分系数:PID 算法中积分项系数值
KDD		0000H-1FFFH	微分系数:PID 算法中微分项系数值
YK	°C	0014H-0046H	反馈:通过反馈算出的温度反馈值
CK		00H-FFH	控制量:PID 算法产生用于控制的量
TKMARK		00H-01H	采样标志位
ADMARK		00H-01H	A/D 转换结束标志位
ADVALUE		00H-FFH	A/D 转换结果寄存单元
TC		00H-FFH	采样周期变量
FPWM		00H-01H	PWM 脉冲中间标志位

## 实验程序清单

```

=====
; 文件名: WENDU.ASM
; 功能描述: 温度闭环控制实验, 通过专用图形显示界面观察
;           实验现象
;           IOY0--8255, IOY3--8254, IOY1--AD0809
=====

```

```

SSTACK    SEGMENT STACK
            DW 256 DUP(?)
TOP        LABEL WORD
SSTACK    ENDS
DATA      SEGMENT
TS        DB 64H                ;采样周期
SPEC      DW 0032H              ;给定温度值 32H 为 50°C
IBAND     DW 0060H              ;PID 算法中积分分离值
KPP       DW 1F60H              ;PID 算法中比例项系数值
KII       DW 0010H              ;PID 算法中积分项系数值
KDD       DW 0020H              ;PID 算法中微分项系数值
CH1       DB ?                  ;专用图形显示的 CH1 通道
CH2       DB ?                  ;专用图形显示的 CH2 通道
CH3       DB ?                  ;专用图形显示的 CH3 通道
YK        DW ?                  ;电机转速反馈量
CK        DB ?                  ;电机转速控制量
TC        DB ?
TKMARK    DB ?
ADMARK    DB ?
ADVALUE   DB ?
FPWM      DB ?
CK_1      DB ?

```

EK\_1 DW ?  
 AEK\_1 DW ?  
 BEK DW ?  
 AAAA DB ?  
 VAA DB ?  
 BBB DB ?  
 VBB DB ?  
 R0 DW ?  
 R1 DW ?  
 R2 DW ?  
 R3 DW ?  
 R4 DW ?  
 R5 DW ?  
 R6 DW ?  
 R7 DB ?  
 R8 DW ?

;热敏电阻温度表

TAB DB

14H,14H,14H,14H,14H,14H,14H,14H,14H,14H,15H,16H,17H,18H,19H,1AH  
DB

1BH,1CH,1DH,1EH,1EH,1FH,20H,21H,23H,24H,25H,26H,27H,28H,29H,2AH  
DB

2BH,2CH,2DH,2EH,2FH,31H,32H,32H,33H,34H,35H,36H,37H,38H,39H,3AH  
DB

3BH,3CH,3DH,3EH,3FH,40H,42H,43H,44H,45H,46H,47H,48H,49H,4AH,4BH  
DB

4CH,4DH,4EH,4FH,50H,4FH,50H,51H,52H,53H,54H,55H,56H,57H,58H,59H  
DB

5AH,5BH,5CH,5DH,5EH,5FH,60H,61H,62H,63H,64H,64H,65H,65H,66H,66H  
DB

67H,68H,69H,6AH,6BH,6CH,6DH,6EH,6EH,6FH,6FH,70H,71H,72H,73H,74H  
DB

75H,76H,77H,78H,79H,7AH,7BH,7CH,7DH,7EH,7FH,80H,81H,82H,83H,84H  
DB

84H,85H,86H,87H,88H,89H,8AH,8BH,8CH,8EH,8FH,90H,91H,92H,93H,94H  
DB

95H,96H,97H,98H,99H,9AH,9BH,9BH,9CH,9CH,9DH,9DH,9EH,9EH,9FH,9FH  
DB

0A0H,0A1H,0A2H,0A3H,0A4H,0A5H,0A6H,0A7H,0A8H,0A9H,0AAH,0ABH,0ACH

```

        DB
0ADH,0AEH,0AFH,0B0H,0B0H,0B1H,0B2H,0B3H,0B4H,0B4H,0B5H,0B6H,0B7H
        DB
0B8H,0B9H,0BAH,0BBH,0BDH,0BEH,0BEH,0C1H,0C2H,0C3H,0C4H,0C5H,0C6H
        DB
0C8H,0CAH,0CCH,0CEH,0CFH,0D0H,0D1H,0D2H,0D4H,0D5H,0D6H,0D7H,0D8H
        DB
0D9H,0DAH,0DBH,0DCH,0DDH,0DEH,0E3H,0E6H,0E9H,0ECH,0F0H,0F2H,0F6H
        DB
0FAH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH
        DB
OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH,OFFH
        DB OFFH,OFFH,OFFH,OFFH,OFFH
DATA    ENDS

CODE    SEGMENT
        ASSUME CS:CODE,DS:DATA
START:  MOV AX, DATA
        MOV DS, AX
MAIN:   CALL INIT                ;系统初始化
        STI
M1:     CMP    TKMARK, 01H        ;判采样周期到否
        JNZM1
        MOV    TKMARK, 00H
M2:     CMP    ADMARK, 01H
        JNZM2
        MOV    ADMARK, 00H
        MOV AX, 0000H            ;查温度表
        MOV    AL, ADVALUE
        MOV    BX, OFFSET TAB
        ADD    BX, AX
        MOV    AL, [BX]
        MOV YK, AX
        CALL PID                ;调用 PID 计算控制量 CK
        MOV AL, CK              ;根据 CK 产生 PWM 脉冲
        SUB AL, 80H
        JC    IS0
        MOV AAAA, AL
        JMP COU

```

```

IS0:MOV AL, 00H
      MOV AAAA, AL
COU:  MOV AL, 7FH
      SUB AL, AAAA
      MOV BBB, AL
      MOV AX, SPEC           ;将给定量 SPEC 存入 CH1
      MOV CH1, AL
      MOV AX, YK             ;将反馈量 YK 存入 CH2
      MOV CH2, AL
      MOV AL, CK             ;将控制量 CK 存入 CH3
      MOV CH3, AL
      CALL PUT_COM          ;调用 PUT_COM 显示给定、反馈与控制量的波

```

形

```

      JMP M1
PUT_COM:           ;显示子程序
      MOV DX, 03FBH
      IN  AL, DX
      AND AL, 7FH
      OUT DX, AL
      MOV DX, 03FDH
WAIT1: IN  AL, DX
      TEST AL, 20H
      JZ  WAIT1
      MOV DX, 03F8H
      MOV AL, CH1
      OUT DX, AL
      MOV DX, 03FDH
WAIT2: IN  AL, DX
      TEST AL, 20H
      JZ  WAIT2
      MOV DX, 03F8H
      MOV  AL, CH2
      OUT DX, AL
      MOV DX, 03FDH
WAIT3: IN  AL, DX
      TEST AL, 20H
      JZ  WAIT3
      MOV DX, 03F8H
      MOV  AL, CH3

```

```
        OUT    DX, AL
        RET

INIT:   NOP                                ;写中断入口地址
        PUSH DS
        XOR AX, AX
        MOV DS, AX
        MOV AX, OFFSET IRQ6                ;8259 IRQ10 定时中断
        MOV SI, 0038H
        MOV [SI], AX
        MOV AX, CS
        MOV SI, 003AH
        MOV [SI], AX
        MOV AX, OFFSET IRQ7                ;8259 IRQ7 中断
        MOV SI, 003CH
        MOV [SI], AX
        MOV AX, CS
        MOV SI, 003EH
        MOV [SI], AX
        CLI
        POP DS

        IN  AL, 21H
        AND AL, 3FH                        ;允许 IRQ6,IRQ7
        OUT 21H, AL

        MOV CK, 00H                        ;变量初始化
        MOV YK, 0000H
        MOV CK_1, 00H
        MOV EK_1, 0000H
        MOV AEK_1, 0000H
        MOV BEK, 0000H
        MOV BBB, 00H
        MOV VBB, 00H
        MOV R0, 0000H
        MOV R1, 0000H
        MOV R2, 0000H
        MOV R3, 0000H
        MOV R4, 0000H
```

```
MOV R5, 0000H
MOV R6, 0000H
MOV R7, 00H
MOV R8, 0000H
MOV TKMARK, 00H
MOV FPWM, 01H
MOV ADMARK, 00H
MOV ADVALUE, 00H
MOV AAAA, 7FH
MOV VAA, 7FH
MOV TC, 00H

MOV DX, 606H
MOV AL, 80H ;初始化 8255-B 口
OUT DX, AL
MOV DX, 640H ;启动 ADC0809
OUT DX, AL
MOV DX, 6C6H
MOV AL, 36H ;8254 计数器 0 输出 OUT0
OUT DX, AL
MOV DX, 6C0H
MOV AL, 10H ;定时 10ms 方波
OUT DX, AL
MOV AL, 27H
OUT DX, AL
RET
IRQ7: NOP
      PUSH AX
      PUSH DX
      MOV DX, 0640H
      IN AL, DX ;读 ADC0809 采样值
      MOV ADVALUE, AL
      MOV ADMARK, 01H
      MOV AL, 20H ;关闭 IRQ7
      OUT 20H, AL
      POP DX
      POP AX
      IRET
IRQ6: NOP
```

```
PUSH AX
PUSH DX
MOV  DX, 0640H
OUT  DX, AL           ;启动 ADC0809
MOV  AL, TC
CMP  AL, TS
JNC  TT2
INC  TC
TT1: CALL KJ
MOV  AL, 20H          ;关闭 IRQ6
OUT  20H, AL
POP  DX
POP  AX
IRET
TT2: MOV  TKMARK, 01H
MOV  TC, 00H
JMP  TT1
KJ:  NOP              ;PWM 子程序
PUSH AX
CMP  FPWM, 01H
JNZ  TEST2
CMP  VAA, 00H
JNZ  ANOT0
MOV  FPWM, 02H
MOV  AL, BBB
CLC
RCR  AL, 01H
MOV  VBB, AL
JMP  TEST2
ANOT0: DEC VAA
MOV  DX, 0602H        ;加温
MOV  AL, 01H
OUT  DX, AL
TEST2: CMP FPWM, 02H
JNZ  OUTT
CMP  VBB, 00H
JNZ  BNOT0
MOV  FPWM, 01H
MOV  AL, AAAA
```

```

        CLC
        RCR AL, 01H
        MOV VAA, AL
        JMP OUTT
BNOT0: DEC VBB
        MOV DX, 0602H           ;停止加温
        MOV AL, 00H
        OUT DX, AL
OUTT:   POP AX
        RET
;=====
;PID 算法子程序
;根据 SPEC, KPP, KII, KDD 及 YK 计算对应控制量 CK
;=====
PID:    MOV AX, SPEC           ;求偏差 EK
        SUB AX, YK
        MOV R0, AX
        MOV R1, AX             ;求偏差变化量 AEK
        SUB AX, EK_1
        MOV R2, AX             ;求 BEK
        SUB AX, AEK_1
        MOV BEK, AX
        MOV R8, AX
        MOV AX, R1
        MOV EK_1, AX
        MOV AX, R2
        MOV AEK_1, AX
        TEST R1, 8000H
        JZ EK1
        NEG R1
EK1:    MOV AX, R1              ;判积分分离值
        SUB AX, IBAND
        JC II
        MOV R3, 00H
        JMP DDD
II:     MOV AL, TS              ;计算积分项
        MOV AH, 00H
        MOV CX, R1
        MUL CX

```



```
MOV CX, KII
DIV CX
MOV R3, AX
TEST R0, 8000H
JZ DDD
NEG R3
DDD: TEST BEK, 8000H           ;计算微分项
JZ DDD1
NEG BEK
DDD1: MOV AX, BEK
MOV CX, KDD
MUL CX
PUSH AX
PUSH DX
MOV AL, TS
MOV AH, 00H
MOV CX, 0008H
MUL CX
MOV CX, AX
POP DX
POP AX
DIV CX
MOV R4, AX
TEST R8, 8000H
JZ DD1
NEG R4
DD1: MOV AX, R3               ;积分项和微分项相加，判溢出
ADD AX, R4
MOV R5, AX
JO L9
L2: MOV AX, R5
ADD AX, R2
MOV R6, AX
JO L3
L5: MOV AX, R6               ;计算比例项
MOV CX, KPP
IMUL CX
MOV CX, 1000H
IDIV CX
```

```

MOV    CX, AX
RCL    AH, 01H
PUSHF
RCR    AL, 01H
POPF
JC     LLL1           ;判溢出，溢出赋极值
CMP    CH, 00H
JZ     LLL2
MOV    AL, 7FH
JMP    LLL2
LLL1:  CMP    CH, 0FFH
JZ     LLL2
MOV    AL, 80H           ;CK=CK_1+CK
LLL2:  MOV    R7, AL
ADD    AL, CK_1
JO     L8
L18:   MOV    CK_1, AL
ADD    AL, 80H
MOV    CK, AL
RET                                ;PID 子程序返回
L8:    TEST   R7, 80H           ;溢出处理程序
JNZ    L17
MOV    AL, 7FH
JMP    L18
L17:   MOV    AL, 80H
JMP    L18
L9:    TEST   R3, 8000H
JNZ    L1
MOV    R5, 7FFFH
JMP    L2
L1:    MOV    R5, 8000H
JMP    L2
L3:    TEST   R2, 8000H
JNZ    L4
MOV    R6, 7FFFH
JMP    L5
L4:    MOV    R6, 8000H
JMP    L5
CODE   ENDS

```

END START

## 第 5 章 保护模式下的 80X86 机器组织

Intel 80x86 家族中的 32 位微处理器始于 80386，兼容了先前的 8086/8088、80186 和 80286。32 位微处理器全面地支持了 32 位数据类型、32 位操作和 32 位物理地址；支持实模式、保护模式的运行方式。在保护模式下的微处理器可以寻址 4GB 的物理地址空间，并且支持虚拟存储管理、多任务管理以及虚拟 86 运行模式。本章就 32 位微处理器在实模式和保护模式下的一些工作原理作一简要叙述。

### 5.1 实模式和保护模式

实模式和保护模式是 32 位微处理器的两种工作模式。在实模式下，32 位微处理器相当于一个可以进行 32 位快速处理的 8086。其最大的寻址空间为 1MB，每个段的最大长度为 64K，且段的起始地址必须是 16 的倍数。

而在保护模式下，全部的 32 条地址线有效，每个段可以寻址的物理空间达到 4GB。保护模式的存储管理，采用了扩充的分段管理机制和可选的分页管理机制，采用了 4 个特权级和完善的特权级检查机制，为存储器的共享和保护提供了硬件的支持。在保护模式下，引入了任务管理的概念，使得 CPU 从硬件上支持了多任务，任务切换提速，任务环境得以保护。

### 5.2 寄存器组织

32 位 80x86 的寄存器是 16 位 80x86 寄存器的超集，分为：通用寄存器、段寄存器、指令指针及标志寄存器、系统地址寄存器、控制寄存器、调试寄存器和测试寄存器，在 Pentium 中还定义了几种模型专用寄存器用于控制可测试性、执行跟踪、性能监测和机器检查错误的功能。其中通用寄存器、段寄存器、指令指针及标志寄存器被应用程序使用，其内容第 1 章已经讲过，本节主要对 32 位 80X86 新增寄存器功能作以讲解。

#### 5.2.1 系统地址寄存器

系统地址寄存器只在保护模式下使用，共有四个：GDTR，IDTR，LDTR 和 TR。如图 5.1 所示。系统地址寄存器是为了能够方便快捷地定位系统中的特殊段，其具体用途和说明在保护模式微机原理及程序设计实验一章中描述。

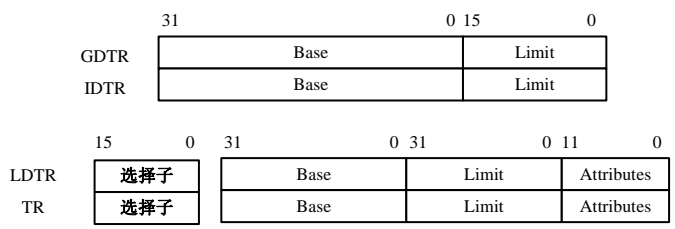


图 5.1 32 位处理器的系统地址寄存器

5.2.2 控制寄存器

在 32 位微处理器中，有 4 个 32 位控制寄存器，如图 5.2 所示，分别命名为 CR0，CR1，CR2 和 CR3。其中 CR0 用于指示处理器的工作方式，CR1 被保留，CR2 和 CR3 在分页管理机制启用的情况下使用。CR2 用于发生页面异常时报告出错信息，CR3 用于保存页目录表的起始物理地址，由于目录是页对齐的，所以仅高 20 位有效，低 12 位保留未用。

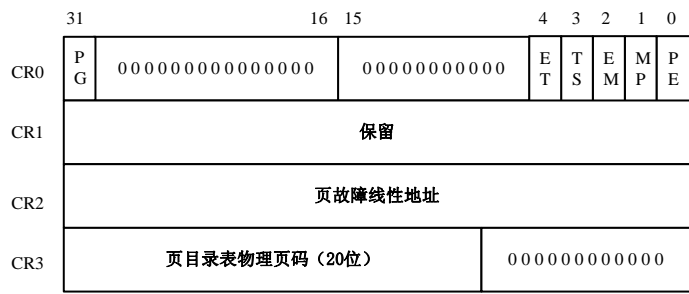


图 5.2 32 位处理器的控制寄存器

表 5.1 PG/PE 与处理器的工作方式

PG	PE	工作模式
0	0	实模式
0	1	保护模式，禁用分页机制
1	0	非法组合
1	1	保护模式，启用分页机制

CR0 中的位 0 用于标记 PE，31 位用于标记 PG，一起用于控制分段和分页管理机制，标记的具体含义如表 5.1 所示。CR0 的位 1~4 分别标记了 MP（算术存在位）、EM（模拟位）、TS（任务切换位）、ET（扩展类型位），它们用于控制浮点协处理器的操作。

5.2.3 调试寄存器和测试寄存器

32 位微处理器共支持 8 个 32 位的可编程调试寄存器，命名为 DRn。其中 DR0、DR1、

DR2、DR3 为断点地址寄存器，DR4，DR5 保留未用，DR6 为调试状态寄存器，DR7 调试控制寄存器。利用这些调试寄存器可以设置代码执行断点，数据访问断点。

32 位微处理器还含有 8 个测试寄存器。TR0 未定义，TR1，TR2 在 Pentium 中使用。TR3，TR4，TR5 用于测试片上高速缓存。TR6，TR7 用于支持转换旁视缓冲器 TLB 的测试。

## 5.3 保护模式下的分段存储管理机制

### 5.3.1 综述

为了对存储器中的程序及数据实现保护和共享，实现对虚拟存储器的管理，32 位微处理器在硬件上提供了支持。在保护模式下，32 位微处理器不仅采用了扩充的存储器分段管理机制，而且还提供了可选的存储器分页管理机制。

#### 1. 虚拟存储器

在实模式下，CPU 的可寻址范围只有 1M 的物理地址空间。而在保护模式下，CPU 可寻址的物理地址空间达到了 4GB。4GB 可谓很大，但实际的微机系统不可能安装如此大的物理内存。为了能够运行大型程序，并真正的实现多任务，必须采用虚拟存储器。虚拟存储器是一种软硬结合的技术，用于提供比计算机系统中实际可用的物理主存储器大得多的存储器空间，这样程序员在编写程序时就不用考虑计算机中物理存储器的实际容量。

#### 2. 地址转换

在保护模式下，虚拟存储器由大小可变的存储块构成，将这样的块称为段。每个段都由一个 8 字节长的数据来描述，描述的内容包括了段的位置，大小和使用情况等，这个 8 字节的数据称为描述符。在保护模式下，虚拟存储器的地址就是由指示描述符的选择子和段内偏移两部分构成，所有的虚拟存储器地址构成了虚拟地址空间，且虚拟地址空间可以达到 64TB。

由于程序只有在物理存储器中才能够运行，所以虚拟地址空间必须映射到物理地址空间，二维的虚拟地址必须转换成一维的物理地址。在保护模式下，每个任务都拥有一个虚拟地址空间，为了避免多个并行任务的多个虚拟地址空间映射到同一个物理地址空间，32 位处理器采用了线性地址空间来隔离虚拟地址空间和物理地址空间。各空间中地址的转换示意如图 5.3 所示。

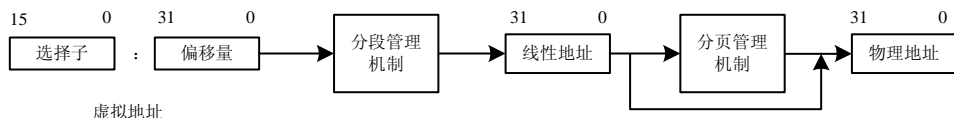


图 5.3 保护模式下的地址转换示意图

### 5.3.2 分段管理的概念

在保护模式下，分段管理机制实现了虚拟地址空间到线性地址空间的映射，也就是将二维

地址转换成一维地址。这一步总是存在的，且由 CPU 中的分段部件自动完成。本节首先介绍分段管理中的有关概念。分段管理的实现示意如图 5.4 所示。

1. 保护模式下段的定义

在保护模式下，每个段的描述符由三个参数构成：段基地址（Base Address）、段界限（Limit）和段属性（Attributes）。

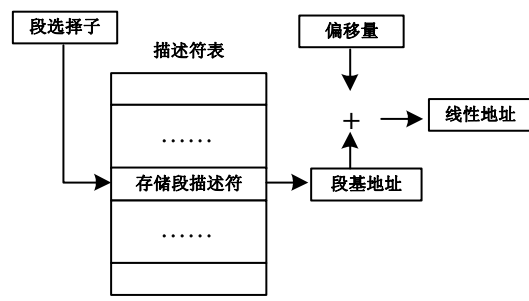


图 5.4 保护模式下的分段管理示意图

(1) 段基地址

段基地址规定了线性地址空间中段的开始，长 32 位。由于段基地址和寻址的长度相同，所以任意一个段都可以从 32 位线性地址空间中的任意一字节开始。

(2) 段界限

段界限规定了段的大小。在保护模式下，段界限用 20 位表示，其单位可以是 1 字节也可以是 4KB。当为字节时，段界限最大长度为  $2^{20}=1\text{MB}$ ，当为 4KB 时，段界限的最大长度为 4GB。

(3) 段属性

段属性规定了段的主要特性，在对段进行各种访问时，对访问的合法性检查主要依据段属性的定义。

2. 段描述符

在保护模式下，每个段都有相应的描述符来描述。根据描述的对象来划分，描述符可以划分成三种：存储段描述符、系统段描述符和门描述符。以下将分别介绍这三种描述符的定义。

(1) 存储段描述符的格式如图 5.5 所示。

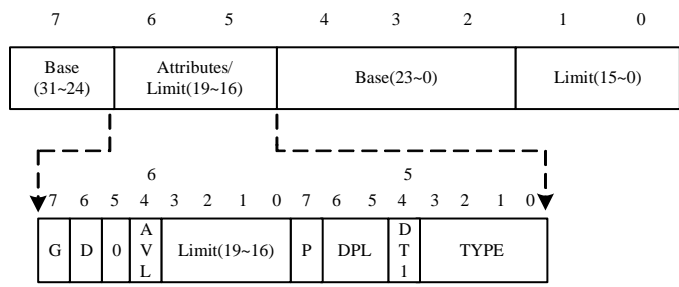


图 5.5 存储段描述符格式

- G: 段界限粒度位, 指示段界限的单位是 1 字节还是 4KB (0/1);
- D: 简单来说决定段是 32 位的还是 16 位的。(1/0);
- AVL: 软件可利用位 (未规定);
- P: 表示描述符对地址转换是有效还是无效 (1/0);
- DPL: 两位, 决定了描述符对应的特权级, 用于特权级检查, 以决定是否能对该段访问;
- DT: 表示段的类型 (描述对象), 为 1, 表示是存储段;
- TYPE: 决定了存储段的属性, 具体属性值见表 5.2。

表 5.2 存储段属性值

类型	说明	类型	说明
0	只读	8	只执行
1	只读, 已访问	9	只执行, 已访问
2	读/写	A	读/执行
3	读/写, 已访问	B	读/执行, 已访问
4	只读, 向低扩展	C	只执行, 一致码段
5	只读, 向低扩展, 已访问	D	只执行, 一致码段, 已访问
6	读/写, 向低扩展	E	读/执行, 一致码段
7	读/写, 向低扩展, 已访问	F	读/执行, 一致码段, 已访问

(2) 系统段描述符和门描述符

系统段是实现存储管理机制所使用的特殊段,用于描述系统段的描述符称为系统段描述符。32 位处理器中含有两种系统段: 任务状态段和局部描述符表段。系统段描述符的格式如图 5.6 所示。

门描述符并不是描述某种内存段, 而是描述控制转移的入口点。这种描述符好比一个通向另一个代码段的门, 通过这种门, 可以实现任务内特权级的变换和任务间的切换。门描述符可以分成: 任务门、调用门、中断门和陷阱门。门描述符的格式如图 5.7 所示。其中 Dword Count, 是双字计数字段, 该字段只在调用门描述符中有效。主程序通常通过堆栈把入口参数传递给子程序, 如果利用调用门调用子程序时, 将引起特权级的转换和堆栈的改变, 那么就需要将外层堆栈中的参数复制到内层堆栈, 双字计数字段决定了复制的双字参数的数量。

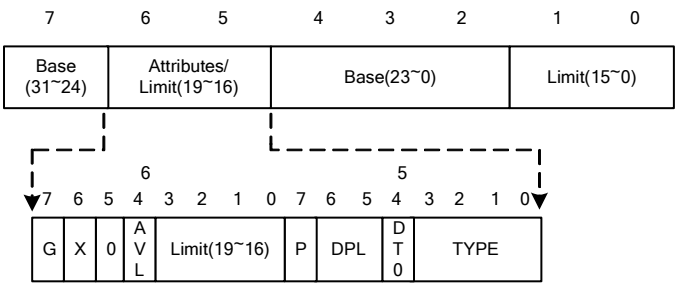


图 5.6 系统段描述符格式



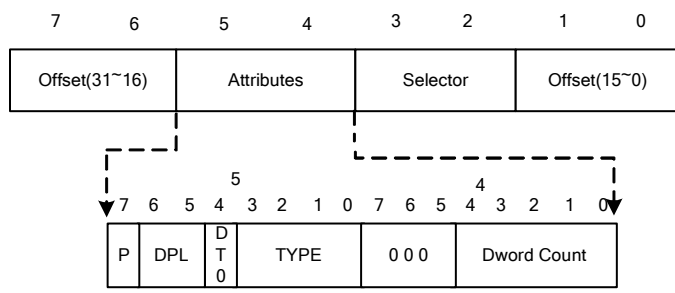


图 5.7 门描述符格式

存储段描述符、系统段描述符和门描述符的格式各不相同，但 DT 和 DPL，P，TYPE 的位置都相同。系统根据 DT 来区分存储段描述符和系统段描述符、门描述符，而当 DT 为 0 时通过 TYPE 值来区分系统段描述和门描述符。系统段描述符和门描述符的属性值如表 5.3 所示。

表 5.5 系统段和门描述符类型字段的编码和含义

类型	说明	类型	说明
0	未定义	8	未定义
1	可用 286 TSS	9	可用 32 位 TSS
2	LDT	A	未定义
3	忙的 286 TSS	B	忙的 32 位 TSS
4	286 调用门	C	32 位 调用门
5	任务门	D	未定义
6	286 中断门	E	32 位 中断门
7	286 陷阱门	F	32 位 陷阱门

### 3. 描述符表

为了方便管理，32 位处理器把描述符组织成线性表进行管理，这样的表称为描述符表。32 位处理器共支持三种描述符表：全局描述符表（GDT Global Descriptor Table）、局部描述符表（LDT Local Descriptor Table）和中断描述符表（IDT Interrupt Descriptor Table）。

#### (1) 全局描述符表（GDT）

GDT 中包含了每个任务都可能访问的段描述符，通常包含描述操作系统及各个任务公共使用的代码段、数据段、堆栈段描述符，还包含了各个任务的 TSS 段描述符、LDT 所在段描述符以及各种调用门和任务门描述符。一个 32 位微处理器系统中有且只有一张 GDT 表，且第一个描述符是一个空描述符（所有值为“0”），GDT 最多可以容纳 8192 个描述符，其在内存中的位置由 GDTR 来定位。

#### (2) 局部描述符表（LDT）

LDT 中包含了每个任务自己的代码段、数据段、堆栈段描述符以及任务内使用的门描述符。

#### (3) 中断描述符表（IDT）

在 32 位微处理器响应中断/异常处理时，需要通过查询 IDT 表定位处理程序所在的段及偏移。IDT 表中包含了中断门/陷阱门和任务门描述符，且最多包含 256 个。在整个 32 位处理器系统中，只允许有一张 IDT 表。

### 4. 段选择子

在实模式下，逻辑地址空间中存储单元的地址由段地址和段偏移构成。在保护模式下，虚拟地址空间中的存储器单元地址由段选择子和段偏移构成。段选择子用来在描述符表中查找相应的段描述符，其格式如图 5.8 所示。

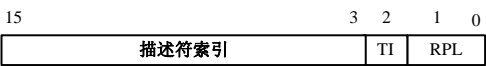


图 5.8 段选择子格式

段选择子长 16 位，其中高 13 位包含了一个指向描述符的索引值，该索引可以确定选择子对应的描述符在 GDT 或 LDT 表中的那一项。TI 位指示了选择子对应的描述符在 GDT 表还是 LDT 表，为 1，表示在 LDT 表中，为 0 表示在 GDT 表中。RPL 表明了选择子的请求特权级，用于特权级检查。

5. 段描述符高速缓冲寄存器

在保护模式下，段寄存器中装载的是段选择子，在访问存储器形成线性地址时，处理器需要使用选择子定位段的基地址等信息。为了避免在每次存储器访问时，都要访问描述符表从而取得段信息，32 位微处理器为每个段寄存器都配备了一个高速缓冲寄存器，这些寄存器称之为段描述符高速缓冲寄存器。每当把一个选择子装入某个段寄存器时，处理器自动从描述表中取出相应的描述符，把描述符中的信息保存到对应的高速缓冲寄存器中，且在以后对该段访问时，都从高速缓冲寄存器中取得段的信息。这部分内容对程序员不可见。段描述符高速缓冲寄存器的内容如图 5.9 所示。

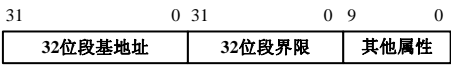


图 5.9 段描述符高速缓冲寄存器内容

6. 特权级

为了使操作系统的程序不受用户程序的破坏，各个任务的程序不相互干扰，32 位处理器提供了特权级检查机制用于程序间的保护。从级别来划分，特权级共分 4 级，取值为 0~3，0 级为最高特权级，3 级为最低特权级。对于一个操作系统来说，通常，操作系统的核心处于 0 级，而 1 级，2 级的程序通常为系统服务程序或操作系统的扩展程序，应用程序特权级最低为 3 级。从类型来划分，特权级可以表示成当前特权级 CPL，描述符特权级 DPL 和请求特权级 RPL。CPL 表示当前正在执行的代码段具有的访问特权级，其值在 CS 中的最低 2 位。DPL 表示了段的被访问权限，RPL 表示了选择子的特权级，指向同一个描述符的选择子可以拥有不同的特权级。

在程序的执行过程中，处理器要进行一系列的特权级检查工作，在检查过程中采用了如下的规则。其中，CPL、RPL、DPL 的值为数值（0、1、2、3）。

(1) 读/写数据段的特权级比较

① 操作堆栈：要求  $CPL=DPL$ ，其中 DPL 为堆栈段对应描述符的 DPL。

② 其他数据段： $CPL \leq DPL$ 。

如果访问中 CPL、DPL 不满足以上要求，将产生 13 号异常。

(2) 数据类段寄存器的装入规则

要访问某个段中的数据，首先需要将段的选择子装入一个段寄存器。在装段寄存器的过程中，必须遵循如下的特权级规定。

① 选择子不能为空。

② 指向的描述符必须是可读/可执行的代码段或数据段描述符。

③ 段必须在内存。

- ④ 装入堆栈段选择子：CPL=DPL，RPL=DPL，否则产生异常 12。
- ⑤ 装入其他数据段选择子：CPL<=DPL，RPL<=DPL，否则产生异常 13。

(3) 代码段寄存器的装入

装入代码段寄存器意味着控制将转移到另一个代码段，可能引起 CPL 的改变，此部分特权级比较规则在控制转移部分具体描述。

(4) IOPL 的使用规则

在 32 位处理器中，对 I/O 指令、STI、CLI 和带 LOCK 前缀的指令的使用有一定的限制。只有当 CPL<=IOPL，或者 TSS 的 I/O 位图允许访问特定的 I/O 地址时，上述指令才能使用，否则产生异常 13。

5.3.3 分段管理机制的实现过程

以下列举一实例说明保护模式下分段管理的过程。此例假设系统只采用分段机制，CPL=0，GDT 表的内容如图 5.10 所示。

1. 执行代码

```
MOV  AX, 08H
MOV  GS, AX
MOV  EBP, 2000H
MOV  AX, GS: [EBP]
```

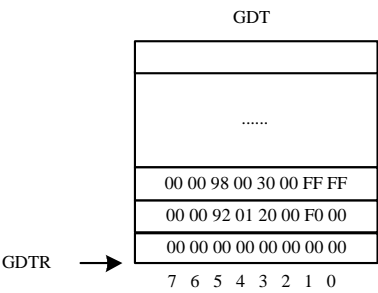


图 5.10 GDT 表内容

2. 执行过程

寻址过程如图 5.11 所示。

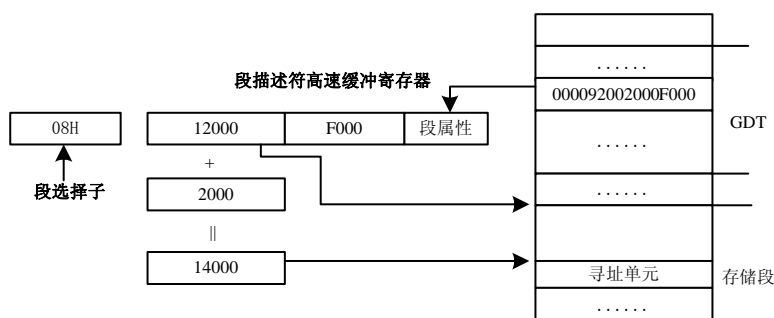


图 5.11 寻址过程示意图

(1) 执行 MOV AX,08H。

(2) 执行 MOV GS,AX。

① 选择子分解：查 GDT，索引=01H。

② 查找描述符：从 GDT 中取第 01H 个描述符，其内容为：0000 9201 2000 F000。

③ 特权级比较：分析描述符的内容，得到 RPL=0，CPL=0，DPL=0，满足  $CPL \leq DPL$ ， $RPL \leq DPL$ 。

④ 将 AX 的内容装入段寄存器，段描述符高速缓冲内容。

32 位基地址=012000H

32 位段界限=00F000H

段属性：G=0,D=0, P=1, DT=1,DPL=1,TYPE=2。

(3) 执行 MOV EBP,2000H。

(4) 执行 MOV AX,GS:[EBP]。

① 特权级比较：CPL=0，DPL=0，满足  $CPL \leq DPL$ 。

② 界限检查：2000H<0F000H。

③ 形成线性地址：线性地址=段基地址+32 位偏移量。

④ 从 GS:[EBP]指向的存储单元中取得数据装入 AX。

## 5.4 任务管理的概念

32 位微处理器从硬件上支持了多任务，这就意味着在系统中可以同时运行多个任务，任务之间可以进行相互切换。所谓的任务切换就是挂起一个任务，恢复另一个任务的执行。在挂起任务时处理器需要保存任务现场的各种寄存器状态的完整映像，而恢复任务时需要将任务现场各种寄存器状态的完整映像导入处理器。在 32 位微处理器中，这种保存和导入工作完全由处理器中的硬件完成。

### 5.4.1 任务状态段 TSS

系统中的每个任务都有一个任务状态段 TSS，用以保存任务的信息。处理器主要是通过 TSS 实现任务的挂起和恢复。任务状态段是一个系统段，由任务状态段描述符来描述，在任务状态段寄存器 TR 中装入的是指向任务状态段描述符的选择子。在任务切换过程中，32 位微处理器首先将其内部各寄存器的当前值保存到 TR 指定的 TSS 中，然后将新任务的 TSS 的选择子装入 TR，并从 TR 指定的 TSS 中导入新的寄存器值。

TSS 主要由 104 字节组成，其基本格式如图图 5.12 所示。任务状态段可以分成如下几个区域。

31	0
0000000000000000	链接字段
ESP0	0H
0000000000000000	SS0
ESP1	4H
0000000000000000	SS1
ESP2	8H
0000000000000000	SS2
CR3	0CH
EIP	10H
EFLAGS	14H
EAX	18H
ECX	1CH
EDX	20H
EBX	24H
ESP	28H
EBP	2CH
ESI	30H
EDI	34H
0000000000000000	ES
0000000000000000	CS
0000000000000000	SS
0000000000000000	DS
0000000000000000	FS
0000000000000000	GS
0000000000000000	LDT
I/O许可位图	0000000000000000 T

图 5.12 任务状态段格式

(1) Link

存放着父任务的 TSS 选择子，即切换到该任务前处理器运行的任务的 TSS 选择子。

(2) 内层堆栈指针区

由于特权级检查规则中规定了不同级别的代码段必须使用相应级别的堆栈段，所以一个任务可能拥有 4 个级别的代码段，则其包含的堆栈段也应该有四个，对应的也包含了四个级别的堆栈指针。对任务来说只需要保存内层三个级别的堆栈指针即可。

(3) CR3

如果系统采用了分页机制，则该部分保存了分页机制中使用的页目录的物理地址。

#### (4) 通用寄存器、段寄存器、指令指针及指令标志寄存器区

在 TSS 对应的任务 A 运行的时候, 寄存器保存区域是没有定义的。当正在运行的任务 A 被切换时, 上述的各类寄存器, 就保存在该区域, 而当 A 任务被恢复运行的时候, 该区域中寄存器的值会被重新装入 CPU 中对应的寄存器中。对于 32 位的寄存器来说, 其保存区域为 32 位, 而段寄存器也分别对应了一个 32 位的双字, 但只用了低 16 位保存段选择子, 而高 16 位设置为 0。

#### (5) LDT 选择子

存放着任务自己的局部描述符表对应的选择子。

#### (6) T 位

调试陷阱位, 如果 T 位置 1, 则在进入该任务后, 会产生调试异常, 否则不会产生异常。

#### (7) I/O 许可位图

为实现输入/输出保护而设置。

### 5.4.2 门描述符

门描述符主要描述了控制转移的入口点, 可以实现任务内特权级的变换和任务间的切换。门描述符可以分成调用门、任务门、中断门/陷阱门三大类。

#### 1. 调用门

调用门描述了某个子程序的入口, 调用门描述符中描述的选择子必须指向一个代码段描述符, 而偏移则指示了子程序在对应代码段内的偏移。

#### 2. 任务门

任务门指示了转移的任务。在任务门描述符中描述的选择子必须指向一个 GDT 中的 TSS 段描述符。任务的入口在 TSS 中, 而描述符中的偏移无意义。

#### 3. 中断门/陷阱门

中断门/陷阱门主要用来描述中断/异常处理的入口点。类似于调用门描述符, 描述符中的选择子和偏移指示了中断/异常处理的入口。中断门/陷阱门描述符只有在 IDT 表中才有效。

### 5.4.3 保护模式下的控制转移

在保护模式下, 控制转移基本上可以分为任务内的转移和任务间的转移两大类。同一任务内的转移包括了段内转移, 段间转移, 而段间转移又有相同特权级和不同特权级之分。由于段内的转移和实模式下的转移相似, 也不涉及特权级变换问题, 所以在此不再重述。此处主要对同一任务中的段间转移和任务切换时的控制转移进行说明。

#### 1. 转移途径

在保护模式下, 段间转移可以通过 JMP、CALL、RET、INT 和 IRET 指令来实现, 也可以通过中断/异常处理来实现。

#### 2. 转移过程

段间转移/段间调用的目标地址由选择子和偏移表示,通常称为目标地址指针。在 32 位段中偏移用 32 位表示,形成了 48 位全指针,而 16 位段中用 16 位表示。在控制转移的过程中,完成转移一般需要经过如下的几个步骤。

- (1) 判别目标地址指示的描述符是否为空描述符。
- (2) 从 GDT 或 LDT 表中读出目标代码段描述符。
- (3) 检测描述的类型,比较 RPL, CPL, DPL。
- (4) 装载目标代码段描述符的内容到 CS 高速缓冲寄存器。
- (5) 判别偏移。
- (6) 装载 CS 和 EIP。

## 5.5 任务内的控制转移

### 5.5.1 任务内的控制转移过程

#### 1. 任务内控制转移的途径

任务内相同特权级的控制转移是指在段间转移过程中 CPL 不发生改变。其实现可以通过几种途径来完成。

- (1) 用 JMP 和 CALL 指令直接将控制转移到一个代码段中(直接转移)。
- (2) 用 RET 和 IRET 指令。
- (3) 用 INT 指令。
- (4) 用 JMP 和 CALL 指令通过调用门实现转移(间接转移)。

#### 2. 用 JMP 实现段间的直接转移

- (1) 判别目标地址指示的描述符是否为空描述符。
- (2) 从 GDT 或 LDT 表中读出目标代码段描述符。
- (3) 检测描述的类型,确定是代码段描述符。
- (4) 比较 RPL, CPL, DPL; 非一致代码段  $CPL=DPL$ ,  $RPL \leq DPL$ , 一致  $CPL \geq DPL$ 。
- (5) 装载目标代码段描述符的内容到 CS 高速缓冲寄存器。
- (6) 判别偏移。
- (7) 装载 CS 和 EIP。
- (8) 转移完成。

说明:一致的可执行段是一种特别的存储段,这种存储段为在多个特权级执行的程序,提供对子例程的共享支持,而不要求改变特权级。如将一段公用程序放在一致的可执行代码段中,则任何特权级的程序都可以使用段间调用指令调用该公用程序,且以调用者所具有的特权级执行该段公用程序。

#### 3. 用 CALL 指令实现的间接转移

- (1) 判别目标地址指示的描述符是否为空描述符。
- (2) 从 GDT 或 LDT 表中读出目标代码段描述符。



- (3) 检测描述的类型, 确定是代码段描述符。
- (4) 将返回地址指针压入堆栈。
- (5) 比较 RPL, CPL, DPL; 非一致代码段  $CPL=DPL$ ,  $RPL \leq DPL$ , 一致  $CPL > DPL$ 。
- (6) 装载目标代码段描述符的内容到 CS 高速缓冲寄存器。
- (7) 判别偏移。
- (8) 装载 CS 和 EIP。
- (9) 转移完成。

#### 4. 用段间返回指令 RET 实现控制转移

- (1) 从堆栈中弹出目标地址指针。
- (2) 判别目标地址指示的描述符是否为空描述符。
- (3) 判别目标地址指针中选择子的 RPL 是否等于 CPL。
- (4) 从 GDT 或 LDT 表中读出目标代码段描述符。
- (5) 检测描述的类型, 确定是代码段描述符。
- (6) 比较 RPL, CPL, DPL; 非一致代码段  $CPL=DPL$ ,  $RPL \leq DPL$ , 一致  $CPL > DPL$ 。
- (7) 装载目标代码段描述符的内容到 CS 高速缓冲寄存器。
- (8) 判别偏移。
- (9) 装载 CS 和 EIP。
- (10) 转移完成。

通常 RET 指令的使用和 CALL 指令的使用对应, 所以如果 CALL 指令能够正常的执行, 则保存在堆栈中的转移地址指针一定符合  $RPL=CPL$  的条件。

#### 5. 使用 JMP、通过调用门实现控制转移

在执行 JMP、CALL 指令时, 当指令中包含的地址指针中, 选择子指向的是一个调用门描述符, 则可以实现段间的间接转移。由于 CALL 指令通过调用门还可以实现任务内不同特权级的转移, 所以, 此处只介绍使用 JMP 指令, 通过调用门实现任务内相同特权级的转移过程。

##### (1) 调用门检查

- ① 分析指令, 取出选择子, 丢弃偏移。
- ② 判断是否  $CPL \leq DPL$ ,  $RPL \leq DPL$ 。
- ③ 门描述符是否存在。
- ④ 从门描述符中取出 48 位全指针。

##### (2) 内层代码段检查

- ① 判别目标地址指示的描述符是否为空描述符。
- ② 从 GDT 或 LDT 表中读出目标代码段描述符。
- ③ 检测描述的类型, 确定是代码段描述符; 调整  $RPL=0$ 。
- ④ 比较 RPL, CPL, DPL; 非一致代码段  $CPL=DPL$ ,  $RPL \leq DPL$ , 一致  $CPL > DPL$ 。
- ⑤ 装载目标代码段描述符的内容到 CS 高速缓冲寄存器。
- ⑥ 判别偏移。
- ⑦ 装载 CS 和 EIP。
- ⑧ 转移完成。

#### 6. 利用 INT 和 IRET 进行的控制转移



此部分在中断/异常处理的章节再进行详细描述。

## 5.5.2 任务内不同特权级的转移过程

任务内不同特权级间的控制转移,是指在段间转移过程中,CPL 发生改变,其中包括了特权级从内层到外层的变换和特权级从外层到内层的变换。通常用 CALL、INT 指令实现外层到内层的转移,用 RET、IRET 指令,实现内层到外层的变换。INT 和 IRET 指令都是在中断/异常中使用的,所以在中断/异常处理部分再详细讲解其用法,此处仅介绍利用 CALL 和 RET 指令实现任务内不同特权级的转移过程。

### 1. 用 CALL 指令实现外层到内层的转移

#### (1) 调用门检查

- ① 分析指令,取出选择子,丢弃偏移。
- ② 判断是否  $CPL \leq DPL$ ,  $RPL \leq DPL$ 。
- ③ 门描述符是否存在。
- ④ 从门描述符中取出 48 位全指针。

#### (2) 内层代码段检查判别目标地址指示的描述符是否为空描述符。

- ① 从 GDT 或 LDT 表中读出目标代码段描述符。
- ② 检测描述的类型,确定是代码段描述符,调整  $RPL=0$ 。
- ③ 比较  $RPL$ ,  $CPL$ ,  $DPL$  若非一致代码段要求  $CPL > DPL$ ,  $RPL \leq DPL$ 。
- ④ 改变  $CPL$ ,使其等于  $DPL$ 。

#### (3) 内层堆栈段检查

- ① 从 TSS 中取得内层堆栈的指针。
- ② 检测选择子是否为空。
- ③ 检测  $CPL$ ,  $RPL$ ,  $DPL$  是否符合  $CPL=DPL$ ,  $RPL=DPL$ 。
- ④ 堆栈指针是否符合段限约束。
- ⑤ 切换到内层堆栈,即将新的堆栈指针装入  $SS:ESP$ 。

#### (4) 其他操作

- ① 将外层代码段使用的堆栈段指针压入  $SS:ESP$  ( $SS$  扩展成 32 位,先压入,再压入  $ESP$ )。

- ② 将调用门中“Dword Count”指定的参数个数从旧的堆栈拷贝到新堆栈中。
- ③ 将外层代码段的  $CS$ ,  $EIP$  压入堆栈 ( $CS$  扩展成 32 位,先压入,再压入  $EIP$ )。
- ④ 装载  $CS$  和  $EIP$ 。
- ⑤ 转移完成。

### 2. 用 RET 指令实现从内层到外层的转移

RET 指令的使用,可以实现内层向外层的返回。RET 指令可以通过带立即数和不带立即数的形式被使用。下面就 RET 指令带立即数的形式介绍由内向外返回的过程。如果 RET 指令没有带立即数,则在转移过程中不包含第 3 步和第 4 步。

#### (1) 从堆栈中弹出返回地址。

- (2) 判别是否  $RPL > CPL$ 。
- (3) 跳过内层堆栈中的参数，参数个数由立即数决定。
- (4) 调整 ESP (跳过参数)。
- (5) 从内层堆栈中弹出指向外层的堆栈指针，并进行特权级检查，然后装入  $SS: ESP$ 。
- (6) 检查 DS、ES、FS、GS，保证寻址的段在外层是可以访问的，如果不可访问，则装入空选择子。
- (7) 判别装入 CS 和 EIP 的值，完成装载。
- (8) 转移完成。

## 5.6 任务间的控制转移

任务间的转移意味着任务的切换，可以使用 JMP 和 CALL 指令通过任务门或直接通过任务状态段来实现，也可以在进入或退出中断/异常处理时实现。此部分仅介绍通过 TSS 和任务门进行切换的过程。

### 5.6.1 通过 TSS 进行任务切换

当 JMP 或 CALL 指令中包含的选择子指向一个可用任务状态段描述符的时候，就可以发生从当前任务到 TSS 指向任务的转移。转移的目标地址由任务的 TSS 中的 CS 和 EIP 决定，而 JMP 和 CALL 指令中的偏移则被丢弃。通过 TSS 进行任务切换要求，选择子的  $RPL \leq TSS$  描述符的 DPL，且  $CPL \leq TSS$  描述符的 DPL。当条件满足时，就可以开始任务切换，具体任务切换的过程在后叙。

### 5.6.2 通过任务门进行切换

当 JMP 或 CALL 指令中包含的选择子指向一个任务门，就可以发生从当前任务到任务门指向的 TSS 所对应任务的转移。转移的目标地址由任务门指向的 TSS 中的 CS 和 EIP 决定，而 JMP 和 CALL 指令中的偏移则被丢弃，任务门中的偏移也无意义。通过任务门进行任务切换，要求任务门选择子的  $RPL \leq$  任务门描述符的 DPL，且  $CPL \leq$  任务门描述符的 DPL；任务门指向的 TSS 必须是 GDT 中可用的 TSS。当条件满足时，就可以开始任务切换，具体任务切换的过程在后面章节继续会讲到。

### 5.6.3 任务切换过程

不管是直接通过 TSS 进行任务切换，还是由任务门选择 TSS 实现任务切换，CPU 都需要对 TSS 中的信息进行一系列判别。以从 A 任务切换到 B 任务为例说明任务切换的过程。

- (1) 检测 B 任务的 TSS 长度要求  $\geq 103$ 。
- (2) 把当前的通用寄存器、段寄存器、EIP 及标志寄存器的内容保存到 A 任务的 TSS 中。
- (3) 将任务 B 的 TSS 段选择子装入 TR，将 B 任务的 TSS 段描述符中任务忙位置位。
- (4) 将保存在 B 的 TSS 中的通用寄存器、段寄存器、EIP 及标志寄存器的值装入 CPU 的各个寄存器（仅装载选择子，防止产生异常），同时也装入 B 任务的 CR3 和 LDT 选择子。
- (5) 链接处理。如果是 CALL 和中断/异常引起的任务切换，需要将 B 的 TSS 中的 LINK 置为 A，将 EFLAGS 中的 NT 位置位，表示是任务嵌套，且不修改任务 A 的 TSS 段描述符中任务忙位。如果是 JMP 指令，则不进行链接处理，只将任务 A 的 TSS 段描述符中任务忙位清零。
- (6) 解链处理。如果是 IRET 引起的任务切换，实施解链处理，要求 B 任务是忙任务，切换后将任务 A 的 TSS 段描述符中任务忙位清零，B 仍为忙任务。
- (7) 将 CR0 中的 TS 位置 1，表示发生了任务切换。
- (8) 将 B 任务 TSS 段中 CS 的 RPL 作为当前的 CPL（任务切换可以从 A 任务任何一个特权级的代码段切换到 B 任务的任意特权级代码段）。
- (9) 装入各个高速缓冲寄存器的值。

## 5.7 中断/异常管理

### 5.7.1 中断/异常的概念

为了支持多任务和虚拟存储功能，在保护模式下将外部中断称为“中断”，而把内部中断称为“异常”。在 32 位处理器系统中，最多支持 256 种中断或异常。

一般来说，中断是由于外部设备的异步事件引发的，通常中断被用来指示一次 I/O 操作的结束。而异常通常是在指令执行期间，由特权级检查时检测到的不正常或者非法的条件而引发的。当异常情况发生时，当前指令无法正确地结束执行，必须通过异常处理程序加以处理。中断调用指令 INT 和溢出中断指令 INTO 也属于异常而不是中断。

#### 1. 异常

CPU 可以识别多种异常，且给每种异常分配一个中断向量号，当异常发生时，系统会根据中断向量号调用相应的异常处理程序加以处理。根据引发异常的指令是否可恢复以及恢复点的不同，把异常分为故障、陷阱和中止三类，对应的异常处理程序被称为故障处理程序、陷阱处理程序和中止处理程序。

表 5.4 中列出了保护模式下异常的类型和说明。

表 5.4 保护模式下的异常说明

向量号	说明	出错码
00H	除法错	无
01H	调试异常	无
02H	NMI（未使用）	无
03H	断点	无
04H	溢出	无
05H	边界检查	无
06H	非法操作码	无
07H	协处理器不可用	无
08H	双重故障	有
09H	协处理段越界	无
0AH	无效 TSS	有
0BH	段不存在	有
0CH	堆栈异常	有
0DH	通用保护	有
0EH	页异常	有
10H	协处理器异常	无
剩余	软中断	无

有一些中断/异常在发生时会产生出错码，出错码可以指示错误的类型、引起错误的描述符所在区域以及引起错误的选择子，通过错误码可以快速、准确地定位错误源。异常出错码的格式如图 5.13 所示，其中 TI 位指示了选择子对应的描述符在 GDT 还是 LDT。如果在中断/异常处理时，从 IDT 重读表项时产生异常，IDT 位置位。如果在某一中断/异常正在处理时又产生了某种异常，则 EXT 位置位。

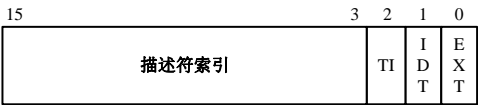


图 5.13 异常出错码格式

2. 中断门和陷阱门描述符

(1) 中断门/陷阱门描述符只在 IDT 中有效。

(2) 中断门/陷阱门描述符中的选择子指示了中断/异常处理程序所在的代码段描述符，偏移地址指示了处理程序在对应代码段内的偏移量。

(3) 使用中断门描述符进入中断处理程序时，CPU 标志寄存器的中断标志位 IF 自动清 0 来关中断。

(4) 使用陷阱门描述符进入异常处理程序时不关中断。

5.7.2 中断/异常处理过程

在实模式下，中断响应是根据中断向量号查找中断向量表，获得中断处理程序的入口地址并且转去执行相应的中断处理程序的一个控制转移过程。

而在保护模式下，中断/异常处理的控制转移是根据中断/异常向量号查找 IDT 中对应的中

断/异常处理门描述符。在 IDT 中存放的是中断门、陷阱门或者任务门描述符，由这些门描述符可以定位中断/异常处理程序所在段的段选择子和段内偏移即转移目标地址的 48 位全指针。在中断/异常的处理中，可以通过中断门/陷阱门，实现任务内的控制转移，让任务内的一个过程实现处理；也可以通过任务门实现任务间的控制转移，即由另一个任务实现处理。

### 1. 通过中断门/陷阱门实现的中断/异常处理

如果中断/异常向量号在 IDT 表中所指向的控制门描述符是中断门或陷阱门，那么控制转移到当前任务的一个处理过程，且在转移中可能引起任务内不同特权级的切换。和段间调用指令 CALL 通过调用门实现控制转移一样，系统从中断门或陷阱门描述符中获得指向中断/异常处理程序入口点的 48 位全指针。48 位全指针中 16 位的选择符是中断/异常处理程序所在代码段的选择符，它指向 GDT 或者 LDT 中的某个代码段描述符；32 位的偏移地址指示中断/异常处理程序入口点在代码段中的偏移量。通过中断/异常处理可将控制转移到同一特权级或者内层特权级。也就是说，可以通过中断/异常处理实现特权级变换。

若是通过中断门转移，则清除 IF、TF 和 NT 标志位；若是通过陷阱门转移，则只清除 TF 和 NT 标志位。将 TF 清 0 表示在中断/异常处理程序的执行过程中不允许单步执行。将 NT 清 0 表示中断/异常处理程序执行完毕按中断返回指令 IRET 返回时，需要返回同一个任务而不是嵌套任务。通过中断门和通过陷阱门转移的区别就在于对 IF 标志位的处理。对于中断门，在控制转移过程中清除 IF，使得在中断/异常处理程序执行期间不再响应来自 INTR 的中断处理请求。对于陷阱门，在控制转移过程中 IF 不发生任何变化，如果原来 IF=1 的话，那么通过陷阱门转移到中断/异常处理程序后仍然允许来自 INTR 的中断处理请求。因此，中断门适合处理中断，而陷阱门适合处理异常。

通过中断门/陷阱门实现的中断/异常处理可能引起任务内不同特权级的切换，则会引起堆栈的切换，在进入处理的整个过程中，堆栈及其中内容的变化如图 5.14 及图 5.15 示，图 5.14 表示了利用中断门/陷阱门进行中断/异常处理时不需要进行特权级的变换，则堆栈不需要切换，只需要将标志寄存器和返回地址压入堆栈即可，如果产生的异常有出错码，还将在堆栈中保留出错码的值。图 5.15 表示了利用中断门/陷阱门进行中断/异常处理时引起任务内特权级的变换，则需要将堆栈切换成内层堆栈，并将外层堆栈、标志寄存器、返回地址和出错码（如果有）的值压入堆栈。

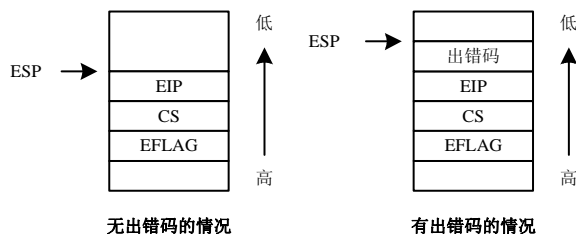


图 5.14 无特权级变换时的堆栈

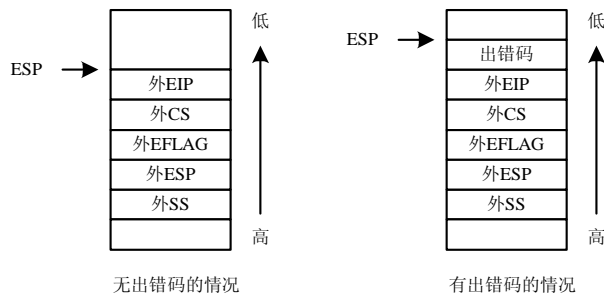


图 5.15 有特权级变换时的堆栈

2. 通过任务门实现的中断/异常处理

如果中断/异常向量号所指向的控制门描述符是任务门描述符，那么控制将转移到一个以独立任务方式出现的中断/异常处理程序。系统可从任务门描述符中获得一个 48 位的全指针，其中 16 位的选择符指向中断/异常处理程序任务的 TSS 段的描述符。通过任务门实现中断/异常控制转移，在进入中断/异常处理程序时，标志寄存器 EFLAG 中的 NT 位被置 1，表示是嵌套任务。

通过任务门的中断/异常控制转移与使用段间调用指令 CALL 通过任务门到一个 TSS 的转移过程很相似，主要的区别就是要在完成任务切换后把中断/异常处理的出错码压入新任务的堆栈中。

3. 中断/异常处理的返回

使用 IRET 指令，可以从中断/异常处理程序返回到异常点。该指令的执行根据中断/异常的进入方式有所不同。

当进入方式是通过任务门时，EFLAG 中的 NT 位被置位，则 IRET 的执行将引起一次任务切换，目标任务的 TSS 选择子就保存在中断/异常处理程序所在任务的 TSS 段中，即 LINK 字段指示的选择子。

当进入方式是通过中断门/陷阱门时，在堆栈中保存了返回地址的指针。处理器执行 IRET 时根据 CS 的 RPL 判定是否会引起任务内特权级的变换，如果不会引起变化，则直接弹出 EIP，CS 和 EFLAG，如果会引起内层到外层的变化，则还会弹出外层堆栈的堆栈指针。

5.8 80X86 保护模式程序设计

通过西安唐都科教仪器公司提供的联机软件 Wmd86 环境可实现 80X86 保护模式程序的编程和上机实验，使学生能够很好地理解和学习保护模式下的程序设计。

5.8.1 常用数据结构及标号定义

1. 存储段/系统段描述符数据结构定义



```

DESC          STRUC
LIMITL        DW    0    ; 段界限(BIT0-15)
BASEL          DW    0    ; 段基地址(BIT0-15)
BASEM          DB    0    ; 段基地址(BIT16-23)
ATTRIBUTES    DB    0    ; 段属性
LIMITH        DB    0    ; 段界限(BIT16-19)(含段属性的高 4 位)
BASEH          DB    0    ; 段基地址(BIT24-31)
DESC          ENDS

```

## 2. 常用标号定义

```

ATCE          =    98h      ; 存在的只执行代码段属性值
ATDR          =    90h      ; 存在的只读数据段类型值
ATDW          =    92h      ; 存在的可读写数据段属性值

```

## 5.8.2 程序设计的一些说明

### 1. 伪指令 “.386” 或 “.386P”

在编程最前头必须写上 “.386” 或 “.386P” 伪指令。这是为了让编译器识别 80386 处理器的新增指令或功能增强的指令。

如果缺省不写，MASM 和 TASM 只能识别 8086/8088 的指令。

### 2. 全局描述符表 GDT 编写

定义一个全局描述符表 GDT，其结构如下：

```

;
DSEG SEGMENT PARA USE16                ; 16 位数据段
GDT LABEL BYTE                          ; 全局描述符表
DUMMY DESC <>                           ; 空描述符
CODE DESC <0FFFFH,CODESEG,,ATCE,,>     ; 代码段描述符
CODE_SEL = CODE-GDT                     ; 代码段选择子
GDTLEN = $-GDT                           ; 全局描述符表长度
DSEG ENDS                               ; 数据段定义结束
;

```

说明：定义的 DSEG 是一个 16 位的数据段，在这个数据段内定义 GDT 表。

(1) “GDT LABEL BYTE” 表明全局描述符表的开始；

(2) 定义一个空描述表 “DUMMY DESC <>” 的结构；

(3) 定义代码段描述符 “CODE DESC <0FFFFH,CODESEG,,ATCE,,>”，它表示代码段的描述符是 “CODE”，代码段是 “CODESEG” 段，代码段的段界限是 “0FFFFH”，“ATCE” 表示代码段的属性是一个可执行段。

(4) “CODE\_SEL” 是代码段描述符的选择子，即代码段描述符 “CODE” 在全局描述表 GDT 中的位置。

(5) “GDTLEN” 是全局描述表的长度。

### 3. 定义全局描述符表寄存器 GDTR 的伪描述符 VGDR

可先定义伪描述符如下:

```
PDESC    STRUC
LIMIT    DW    0
BASE     DD    0
PDESC    ENDS
```

利用以上所定义的结构类型, 可定义 GDTR 的伪描述符:

```
VGDR     PDESC    <GDTLEN-1,>
```

其中: “GDTLEN-1” 表示 GDT 表的界限, “0” 表示 GDT 表的基址。

### 4. 实模式和保护模式的切换

80386 有四个控制寄存器, 分别命名为 CR0、CR1、CR2 和 CR3。其中控制寄存器 CR0 中的 0 位用 PE 标记, 它决定了 CPU 的工作模式。PE=0, 处理器运行于实模式; PE=1, 处理器运行于保护模式。

从实模式切换到保护模式, 可用如下 3 条指令:

```
MOV      EAX, CR0          ; 把 CR0 复制到 EAX
OR       EAX, 1            ; 把对应的 PE 位置 1
MOV      CR0, EAX          ; 使 CR0 的 PE 位为 1
```

执行完上述 3 条指令后, 紧接着要安排一条段间转移指令:

```
JUMP     <CODE_SEL>, <OFFSET_VIRTUAL>
```

这条段间转移指令是在实模式下被预取, 在保护模式下被执行。利用这条段间转移指令可把保护模式下代码段的选择子装入 CS, 同进刷新指令预取队列。从而真正进入保护模式。

从保护模式切换到实模式, 可用如下 3 条指令:

```
MOV      EAX, CR0          ; 把 CR0 复制到 EAX
AND      AL, 0FEH          ; 把对应的 PE 位置 0
MOV      CR0, EAX          ; 使 CR0 的 PE 位为 1
```

执行完上述 3 条指令后, 紧接着也要安排一条段间转移指令:

```
JUMP     <SEG_REAL>, <OFFSET_REAL>
```

这条段间转移指令, 一方面清指令预取队列, 另一方面把实模式下的代码段的段值送 CS。这条段间转移指令在保护模式下被预取, 在实模式下被执行。

## 5.8.3 一个实例

下面编写一个实例, 学习实模式和保护模式的切换, 理解保护模式的编程方法。具体实现步骤:

- (1) 作切换到保护模式的准备;
- (2) 切换到保护模式;
- (3) 把源数据段的内容传送到目的数据段;



(4) 切换回实模式;

(5) 显示源数据段和目的数据段的内容。

实例源程序如下:

```

;-----
;文件名:5-1.ASM
;功能:学习实模式和保护模式的切换
;-----
;存储段描述符结构类型定义
DESC          STRUC
LIMITL        DW 0    ;段界限(BIT0-15)
BASEL          DW 0    ;段基地址(BIT0-15)
BASEM          DB 0    ;段基地址(BIT16-23)
ATTRIBUTES     DB 0    ;段属性
LIMITH         DB 0    ;段界限(BIT16-19)(含段属性的高 4 位)
BASEH          DB 0    ;段基地址(BIT24-31)
DESC           ENDS
;-----
;常量定义
ATDW           EQU    92H    ;存在的可读写数据段属性值
ATCE           EQU    98H    ;存在的只执行代码段属性值
;-----
;伪描述符结构类型定义
PDESC          STRUC
LIMIT          DW 0    ;16 位界限
BASE           DD 0    ;32 位基地址
PDESC          ENDS
;-----
;16 位偏移的段间直接转移指令的宏定义
JUMP16         MACRO SELECTOR,OFFSET
DB    0EAH      ;操作码
DW    OFFSET     ;16 位偏移量
DW    SELECTOR   ;段值或段选择子
ENDM
;-----
.386P          ;.386P 伪指令
;-----
DSEG           SEGMENT PARA      USE16    ;定义 16 位数据段
GDT            LABEL BYTE        ;全局描述符表
DUMMY          DESC              <>       ;空描述符

```

```

CODEM    DESC    <0FFFFH,,,ATCE,,>    ;代码段描述符
DATAS    DESC    <0FFFFH,,,ATDW,,>    ;源数据段描述符
DATAD    DESC    <0FFFFH,,,ATDW,,>    ;目标数据段描述符
VGDTTR    PDESC <GDTLEN-1,>            ;伪描述符
GDTLEN    =      $-GDT                ;全局描述符表长度
CODEM_SEL    =      CODEM-GDT          ;代码段选择子
DATAS_SEL    =      DATAS-GDT          ;源数据段选择子
DATAD_SEL    =      DATAD-GDT          ;目标数据段选择子
DSEG      ENDS                        ;数据段定义结束
;-----
DATA1     SEGMENT PARA                USE16    ;定义 16 位源数据段
MES1      DB 'This is tangdu speak!',' $'    ;$ 为字符串结束标志
ML        =      $-MES1                ;字符串的长度
DATA1     ENDS                        ;源数据段定义结束
;-----
DATA2     SEGMENT PARA                USE16    ;定义 16 位目标数据段
BUF       DB 64 DUP(?)                ;定义 64 个缓冲区字节单元
DATA2     ENDS                        ;目标数据段定义结束
;-----
CSEG      SEGMENT USE16                ;16 位代码段
ASSUME    CS:CSEG,DS:DSEG
START PROC
    MOV    AX,DSEG
    MOV    DS,AX
    ;准备要加载到 GDTR 的伪描述符
    MOV    BX,16
    MUL    BX                        ;数据段地址左移 4 位
    ADD    AX,OFFSET GDT              ;加上 GDT 的偏移得到物理地
址
    ADC    DX,0
    MOV    WORD PTR VGDTTR.BASE,AX    ;将得到的物理地址填入
VGDTTR 描述符
    MOV    WORD PTR VGDTTR.BASE+2,DX
    ;设置代码段描述符
    MOV    AX,CS
    MUL    BX                        ;代码段地址左移 4 位
    MOV    WORD PTR CODEM.BASEL,AX    ;代码段开始偏移为 0
    MOV    BYTE PTR CODEM.BASEM,DL    ;将得到的物理地址填入
CODEM 描述符

```

```

MOV     BYTE PTR CODEM.BASEH,DH
;设置源代码段描述符
MOV     AX,DATA1
MOV     BX                                ;源数据段地址左移 4 位
MOV     WORD PTR DATAS.BASEL,AX          ;源数据段开始偏移为 0
MOV     BYTE PTR DATAS.BASEM,DL          ;将得到的物理地址填入
DATAS 描述符
MOV     BYTE PTR DATAS.BASEH,DH
;设置目标代码段描述符
MOV     AX,DATA2
MOV     BX                                ;目标数据段地址左移 4 位
MOV     WORD PTR DATAD.BASEL,AX          ;目标数据段开始偏移为
0
MOV     BYTE PTR DATAD.BASEM,DL          ;将得到的物理地址填入
DATAD 描述符
MOV     BYTE PTR DATAD.BASEH,DH
;加载 GDTR
LGDT    QWORD PTR VGDTR
CLI                                ;关中断
;切换到保护方式
MOV     EAX,CR0
OR      EAX,1
MOV     CR0,EAX
;清指令预取队列,并真正进入保护方式
JUMP16  <CODEM_SEL>,<OFFSET VIRTUAL>
VIRTUAL: ;现在开始在保护方式下运行
MOV     AX,DATAS_SEL
MOV     DS,AX                        ;加载源数据段描述符
MOV     AX,DATAD_SEL
MOV     ES,AX                        ;加载目标数据段描述符
XOR     DI,DI                        ;DI 清零
XOR     SI,SI                        ;SI 清零
MOV     CX,ML                        ;设置数据长度
REPZ    MOVSB                        ;通过串传送指令传数
;切换回实模式
MOV     EAX,CR0
AND     AL,11111110B
MOV     CR0,EAX
;清指令预取队列,进入实方式

```

```

        JUMP16    <SEG REAL>,<OFFSET REAL>
REAL:   ;现在又回到实方式
        STI                               ;开中断
        MOV      AX,DATA1
        MOV      DS,AX                    ;送源数据段
        MOV      DX,OFFSET MES1
        MOV      AH,09H
        INT      21H                      ;用 INT 21H 功能调用显示 MES1 数据
段的内容
        MOV      AH,02H
        MOV      DL,0DH
        INT      21H                      ;回车
        MOV      AH,02H
        MOV      DL,0AH
        INT      21H                      ;换行
        MOV      AX,DATA2
        MOV      DS,AX                    ;送目标数据段
        MOV      DX,OFFSET BUF
        MOV      AH,09H
        INT      21H                      ;用 INT 21H 功能调用显示 BUF 数据
段的内容
        MOV      AX,4C00H
        INT      21H                      ;程序终止
START   ENDP
;-----
CSEG    ENDS                               ;代码段定义结束
;-----
END      START

```

在上例中，程序刚开始定义了描述符结构类型及常量等。一般的，我们将这些结构类型及常量定义可放在一个头文件 386SCD.INC 里，在编程时列出头文件即可。如：

```
INCLUDE 386SCD.INC
```

参考上述例子，现给出保护模式编程格式大致如下：

```

;-----
INCLUDE 386SCD.INC
;-----
.386P
;-----
GDTSEG    SEGMENT PARA    USE16

```

```

        .....          ;定义全局描述符表 GDT 数据段
GDTSEG      ENDS
;-----
LDTSEG      SEGMENT PARA      USE16
        .....          ;定义局部描述符表 LDT 数据段
LDTSEG      ENDS
;-----
IDTSEG      SEGMENT PARA      USE16
        .....          ;定义中断描述符表 IDT 数据段
IDTSEG      ENDS
;-----
TSSSEG      SEGMENT PARA      USE16
        .....          ;定义任务表 TSS 数据段
TSSSEG      ENDS
;-----
DSTACKSEG   SEGMENT PARA      USE16
        .....          ;定义堆栈数据段
DSTACKSEG   ENDS
;-----
DDATASEG    SEGMENT PARA      USE16
        .....          ;定义数据段的数据段
DDATASEG    ENDS
;-----
CODEMSEG    SEGMENT PARA      USE16
        .....          ;定义主程序段的代码段
        MOV      EAX,CR0    ;切换到保护模式
        OR       EAX,1
        MOV      CR0,EAX
        JUMP16    <CODEM_SEL>,<OFFSET VIRTUAL>
VIRTUAL:    .....          ;现在开始在保护模式下运行
        .....
        MOV      EAX,CR0    ;切换到实模式
        AND      AL,11111110B
        MOV      CR0,EAX
        JUMP16    <SEG REAL>,<OFFSET REAL>
REAL:      .....          ;现在又回到实方式
CODEMSEG    ENDS
;-----

```

## 第 6 章 保护模式微机原理及其程序设计实验

本章列举了 80X86 工作在保护模式下的原理及其程序设计实验。其中包括四大类型的实验，即描述符及描述符表实验、特权级变换实验、任务切换实验和中断与异常处理实验。这些实验通过唐都科教仪器公司提供的含 80386EX 32 位单板微机系统的实验箱以及专为其配套开发的 Wmd86 联机软件来完成。

### 6.1 描述符及描述符表实验

#### 6.1.1 实验目的

1. 熟悉保护模式的编程格式。
2. 掌握全局描述符及局部描述符的声明方法。
3. 掌握使用选择子访问段的寻址方法。

#### 6.1.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

#### 6.1.3 全局描述表实验

本实验要求在一个 0 级代码段中将源数据段中的一段数据传输到目标数据段中。其中所有段的段描述符均放置在全局描述符表 GDT 中。

##### 1. 预备知识

在保护模式下，每一个段都有一个相应的描述符来描述，每个描述符长 8 个字节，可分为存储段描述符、系统段描述符和门描述符（控制描述符）。

存储段是存放可由程序直接进行访问的代码段和数据段。存储段描述符描述存储段，所以存储段描述符也被称为代码段和数据段描述符。

一个任务会涉及多个段，每个段需要一个描述符来描述，为了便于组织管理，80X86 把描述符组织成线性表。由描述符组成的线性表称为描述符表，在 80X86 中有三种类型的描述符表：全局描述表 GDT、局部描述符表 LDT 和中断描述符表 IDT。在整个系统中，全局描述表 GDT 和中断描述表 IDT 只有一张，局部描述符表 LDT 可以有若干张，每个任务可以有一张。

在实模式下，逻辑地址空间中存储单元的地址由段值和段内偏移两部分级成。在保护模式下，虚拟地址空间（相当于逻辑地址空间）中存储单元的地址由段选择子和段内偏移两部分组

成。与实模式相比，段选择子替代了段值。

段选择子长 16 位，高 13 位是描述符索引，即描述符在描述符表中的序号。段选择子第 2 位是引用描述符表指示位，标记为 TI，TI=0 指示从全局描述符表 GDT 中读取描述符；TI=1 指示从局部描述符表 LDT 中读取描述符。选择子最低两位是请求特权级 RPL，用于特权级检查（详见本书 5.3 节）。

选择子确定描述符，描述符确定段基地址，段基地址和偏移之和就是线性地址。所以虚拟地址空间中段选择子和偏移两部分构成的二维虚拟地址，就是这样确定了线性地址空间中的一维线性地址。

## 2. 实验分析

为了实现在 0 级代码段中完成数据传输，实验程序中需要安排一个 0 级代码段和两个 0 级数据段（可以是 0~3 级任一级别的数据段）。

在程序开始声明一个数据段“DSEG”，来描述这三个段的描述符，其中有代码段描述符 CODEM，源数据段描述符 DATAS 和目标数据段描述符 DATAD，将它们相应的选择子分别定义为 CODEM\_SEL，DATAS\_SEL，DATAD\_SEL。为这三个段分别定义描述符：

(1) 代码段描述符：CODE DESC <0FFFFH,,ATCE,,>

段属性说明：

G	:	0	；以字节为段界限粒度
D	:	0	；是 16 位的段
P	:	1	；描述符对地址转换有效/该描述符对应的段存在
DPL	:	0	；0 级段
DT	:	1	；描述符描述的是存储段
TYPE	:	0x8	；只执行段

(2) 源数据段描述符：DATAS DESC <0FFFFH,,ATDW,,>

段属性说明：

G	:	0	；以字节为段界限粒度
D	:	0	；是 16 位的段
P	:	1	；描述符对地址转换有效/该描述符对应的段存在
DPL	:	0	；0 级段
DT	:	1	；描述符描述的是存储段
TYPE	:	0x2	；可读写段

(3) 目标数据段描述符 DATAD DESC <0FFFFH,3000H,,ATDW,,>

目标数据段描述符的内容基本与源数据段的内容相同，只是我们给定了它的段基址是 00003000H。

在全局描述表 GDT 定义的过程中，首先需要定义一个空的描述符 DUMMY，作为定义的开始，然后再定义其它描述符。

本实验可实现将一个数据段中的数据搬运到另一个数据段所指定的地址空间中（指定地址给定为 00003000H）。源数据段 DSEG1 的内容为 1111H,2222H,3333H,4444H,5555H,6666H,7777H,8888H，在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

### 3. 实验步骤

- (1) 运行 Wmd86 集成操作软件，进入 Wmd86 集成开发环境。
- (2) 在菜单“设置\语言”栏，选择“汇编语言”，如图 6.1:

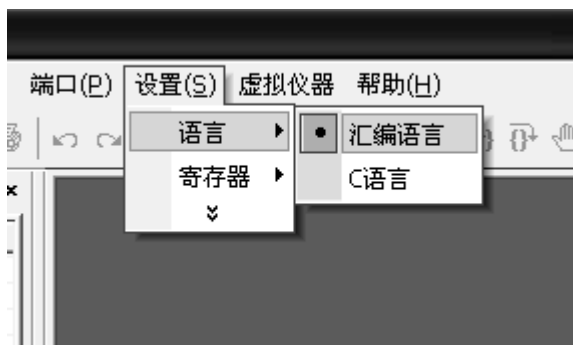


图 6.1 设置语言环境界面

- (3) 在菜单“设置\寄存器”栏，选择“32 位寄存器”，由于保护模式下的实验，都用到了 32 位寄存器，所以本章及本章向后的全部实验都必须在 32 位寄存器状态下工作，如图 6.2:

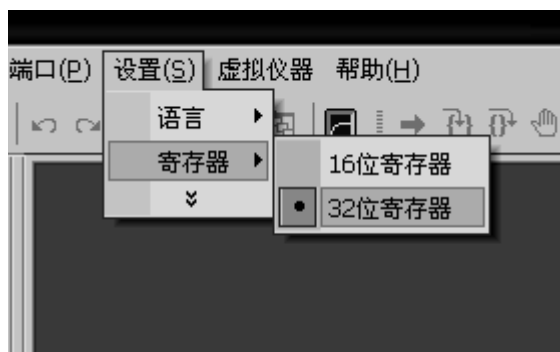


图 6.2 设置寄存器类型界面

设置好以后，下次再启动软件，设置栏将保持这次的修改不变。

- (4) 设置完毕后，点击新建或按 Ctrl+N 组合键来新建一个文档，如图 6.3 所示，默认文件名为 Wmd861。

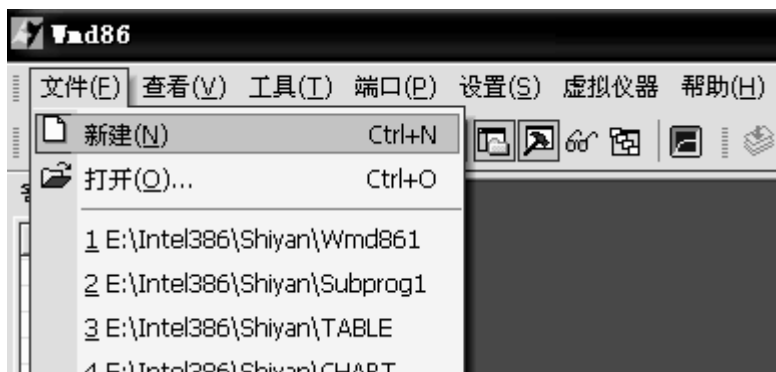


图 6.3 新建文件界面



(5) 编写实验程序(例程文件名为: 6-1-1.ASM), 如图 6.4 所示, 并保存, 此时系统会提示输入新的文件名, 输完后点击保存。



图 6.4 编辑环境界面





(6) 点击 , 编译文件, 若程序编译无误, 则可以继续点击  进行链接, 链接无误后方可加载程序。编译、链接后输出如下图 6.5 所示的输出信息。



图 6.5 编译链接信息界面

(7) 连接 PC 与实验系统的通讯电缆，打开实验系统电源。

(8) 编译、链接都正确并且上下位机通讯成功后，就可以下载程序，联机调试了。可以通过端口列表中的“端口测试”来检查通讯是否正常。点击下载程序。为编译、链接、下载组合按钮，通过该按钮可以将编译、链接、下载一次完成。下载成功后，在输出区的结果窗中会显示“加载成功！”，表示程序已正确下载。起始运行语句下会有一条绿色的背景。如图 6.6 所示：

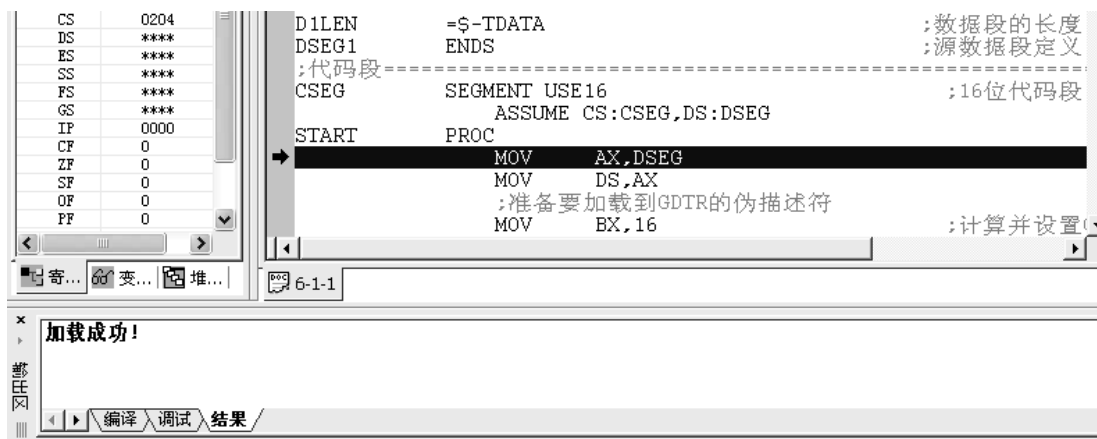



图 6.6 程序起始运行界面

(9) 将输出区切换到调试窗口，使用 D0000:3000 命令查看内存 00003000H 起始地址的数据，如图 6.7 所示。存储器在初始状态时，默认数据为 CC。



图 6.7 调试界面

(10) 点击按钮运行程序，待程序运行停止，观察运行结果，使用命令 D0000:3000 回车，观察数据变化。如图 6.8：

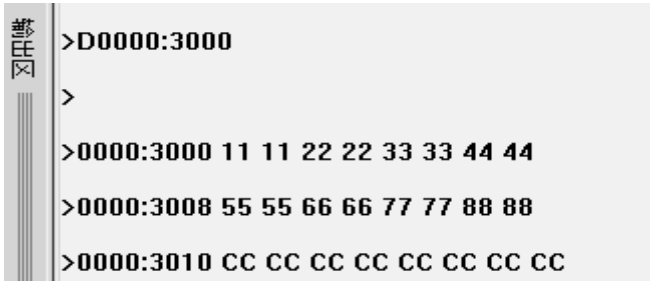


图 6.8 查看内存数据界面

(11) 可以使用 E0000:3000 来改变该地址单元的数据，如图 6.9 所示，输入 01 后，按“空格”键，可以接着输入第二个数，如 02，结束输入按“回车”键。

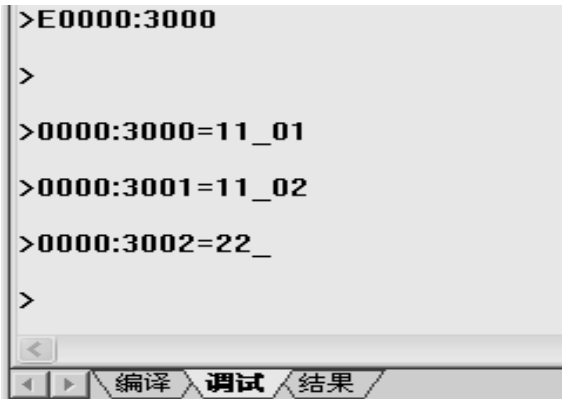


图 6.9 修改内存数据

实验程序清单

```

;-----
;文件名:6-1-1.ASM
;功能:全局描述符及全局描述符表实验
;-----
INCLUDE          386SCD.INC
;数据段=====
DSEG             SEGMENT USE16                ;定义 16 位数据段
GDT              LABEL  BYTE                  ;全局描述符表
DUMMY           DESC   <>                    ;空描述符
CODEM           DESC   <0FFFFH,,ATCE,,>      ;代码段描述符
DATAS           DESC   <0FFFFH,,ATDW,,>      ;源数据段描述符
DATAD           DESC   <0FFFFH,3000H,,ATDW,,> ;目标数据段描述符
VGDTTR          PDESC  <GDTLEN-1,>           ;伪描述符
GDTLEN          =      $-GDT                  ;全局描述符表长度
CODEM_SEL       =      CODEM-GDT              ;代码段选择子
DATAS_SEL       =      DATAS-GDT              ;源数据段选择子
    
```

```

DATAD_SEL      =      DATAD-GDT      ;目标数据段选择子
DSEG           ENDS                  ;数据段定义结束
;=====
DSEG1          SEGMENT PARA          USE16      ;定义 16 位源数据段
TDATA          DW      1111H, 2222H, 3333H, 4444H ;定义原数据段数据
               DW      5555H, 6666H, 7777H, 8888H
D1LEN          = $-TDATA              ;数据段的长度
DSEG1          ENDS                  ;源数据段定义结束
;代码段=====
CSEG           SEGMENT USE16          ;16 位代码段
               ASSUME CS:CSEG,DS:DSEG
START          PROC
               MOV     AX,DSEG
               MOV     DS,AX
               ;准备要加载到 GDTR 的伪描述符
               MOV     BX,16          ;计算并设置 GDT 基地址
               MUL     BX            ;数据段地址左移 4 位
               MOV     WORD PTR VGDTR.BASE,AX ;将得到的物理
地址填入 VGDTR 描述符
               MOV     WORD PTR VGDTR.BASE+2,DX
               ;设置代码段描述符
               MOV     AX,CS          ;计算并设置源数据段基
址
               MUL     BX            ;代码段地址左移 4 位
               MOV     WORD PTR CODEM.BASEL,AX ;代码段开始偏
移为 0
               MOV     BYTE PTR CODEM.BASEM,DL ;将得到的物理地址
填入 CODEM 描述符
               MOV     BYTE PTR CODEM.BASEH,DH
               ;设置源数据段描述符
               MOV     AX,DSEG1      ;计算并设置源数据段基址
               MUL     BX            ;;源数据段地址左移 4 位
               ADD     AX,OFFSET TDATA ;加上 TDATA 的偏移得到
物理地址
               ADC     DX,0
               MOV     WORD PTR DATAS.BASEL,AX ;将得到的物理地址
填入 DATAS 描述符
               MOV     BYTE PTR DATAS.BASEM,DL
               MOV     BYTE PTR DATAS.BASEH,DH

```

```

;加载 GDTR
LGDT    QWORD PTR VGDTR
CLI                                           ;关中断
;切换到保护方式
MOV     EAX,CR0
OR      EAX,1
MOV     CR0,EAX
;清指令预取队列,并真正进入保护方式
JUMP16  <CODEM_SEL>,<OFFSET VIRTUAL>
VIRTUAL:
;现在开始在保护方式下运行
MOV     AX,DATAS_SEL
MOV     DS,AX                               ;加载源数据段描述符
MOV     AX,DATAD_SEL
MOV     ES,AX                               ;加载目标数据段描述符
XOR     DI,DI                               ;DI 清零
XOR     SI,SI                               ;SI 清零
MOV     CX,D1LEN                            ;设置数据长度

M1:      MOVSB                               ;通过串传送指令传数
        LOOP M1

;切换回实模式
MOV     EAX,CR0
AND     AL,11111110B
MOV     CR0,EAX
;清指令预取队列,进入实方式
JUMP16  <SEG REAL>,<OFFSET REAL>
REAL:   ;现在又回到实方式
        STI                                 ;开中断

        MOV  AX,4C00H
        INT  21H                            ;程序终止

START    ENDP
CLEN     =$.-1
CSEG     ENDS
        END      START

```

#### 4. 保护模式下查看描述符表的方法

通过 Wmd86 软件输出区中集成的 Debug 调试功能,可以很方便的在做保护模式实验时,查看实验所涉及的描述符表及其变化。

在本章中,编程所定义的全局描述符表 GDT 中的各个段描述符、局部描述符表 LDT 中各个段描述符和中断/异常描述符表 IDT 中各个门描述符都可以用 Debug 命令来查看它。

现以本小节 6-1-1.asm 程序为例,讲述查看描述符表的方法。

(1) 编译链接加载该程序后,从变量区的寄存器栏可以读出在实模式下加载该程序的代码段和偏移值。分别为:

CS: 0204, IP: 0000

如图 6.10 所示。

(2) 用反汇编命令查看数据段装载的地址空间。

在输出区调试栏输入:

U0204: 0000 回车

如图 6.11 所示。

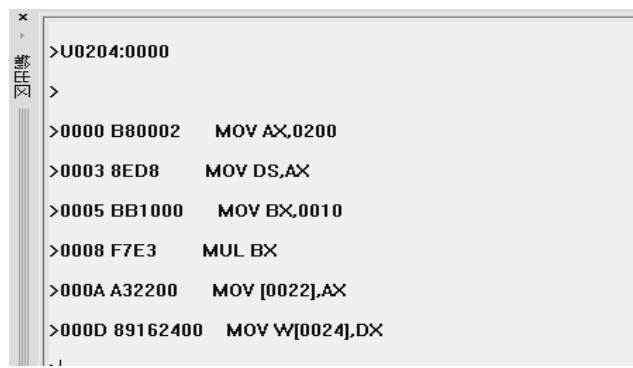


图 6.11 反汇编查看数据段

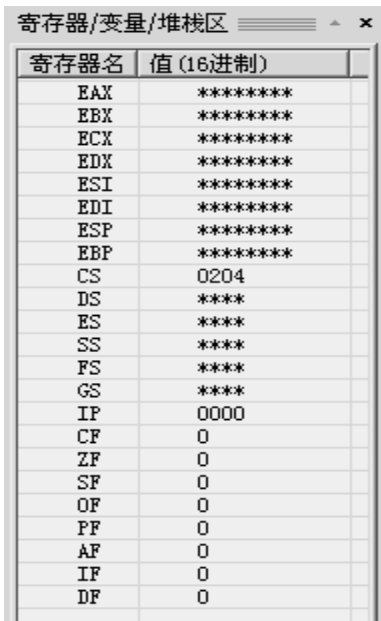


图 6.10 查看寄存器

(3) 从图 6.11 前两条指令可以看到数据段的段值 DS 为 0200,即就是原程序中 DSEG 数据段的段值,由于在原程序的 DSEG 段中,定义了全局描述符表 GDT,在该表中给出了 DUMMY、CODEM、DATAS、DATAD 等段的描述符,且每个描述符都是用 8 个字节来描述。所以就可以通过查看该 DSEG 段地址中的数据值来查看每个段的描述符。

下面是原程序中定义的 DSEG 段:

DSEG	SEGMENT	USE16		;定义 16 位数据段
GDT	LABEL	BYTE		;全局描述符表
DUMMY	DESC	<>		;空描述符
CODEM	DESC	<0FFFFH,,ATCE,,>		;代码段描述符
DATAS	DESC	<0FFFFH,,ATDW,,>		;源数据段描述符

DATAD	DESC	<0FFFFH, 3000H, , ATDW, , >	; 目标数据段描述符
VGDR	PDESC	<GDTLEN-1, >	; 伪描述符
GDTLEN	=	\$-GDT	; 全局描述符表长度
CODEM_SEL	=	CODEM-GDT	; 代码段选择子
DATAS_SEL	=	DATAS-GDT	; 源数据段选择子
DATAD_SEL	=	DATAD-GDT	; 目标数据段选择子
DSEG	ENDS		; 数据段定义结束

(4) 通过 D 命令来查看所定义的 GDT 描述符，即 DSEG 段，在输出区调试栏输入：  
DO200: 0000 回车  
如图 6.12 所示。

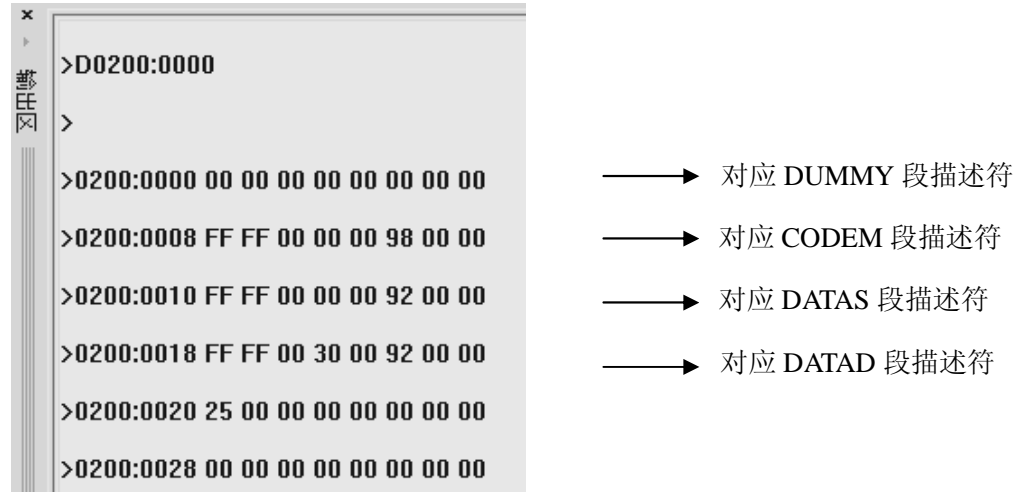


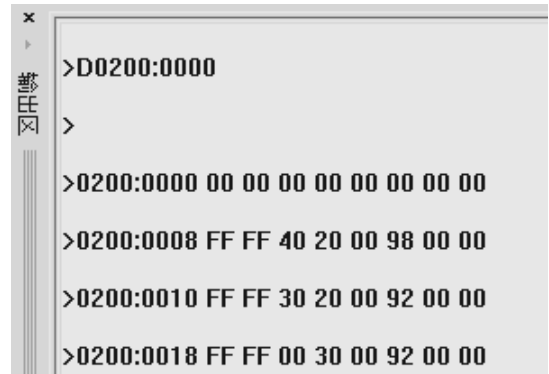
图 6.12

图 6.12 即就是所看到的 GDT 表中定义各段的描述符，这是程序加载完后各段描述符的值，其中：“0FFFFH”是段界限，“98”“92”等是段属性，各段的基地址目前全是“0”。

(5) 在程序进入保护模式之前设断点，然后运行至断点。如图 6.13 所示。



图 6.13



(6) 运行到进入保护模式之前停止，查看一下各描述符的值，如图 6.14 所示。可以看到各段的描述符中，段基地址已经有了值。

- > 对应 DUMMY 段描述符
- > 对应 CODEM 段描述符
- > 对应 DATAS 段描述符
- > 对应 DATAD 段描述符

图 6.14

(7) 继续运行程序，直到运行停止，再查看一下各描述符的值，如图 6.15 所示。可以看到各段的描述符中，段属性的值发生了变化，想一想这是为什么？

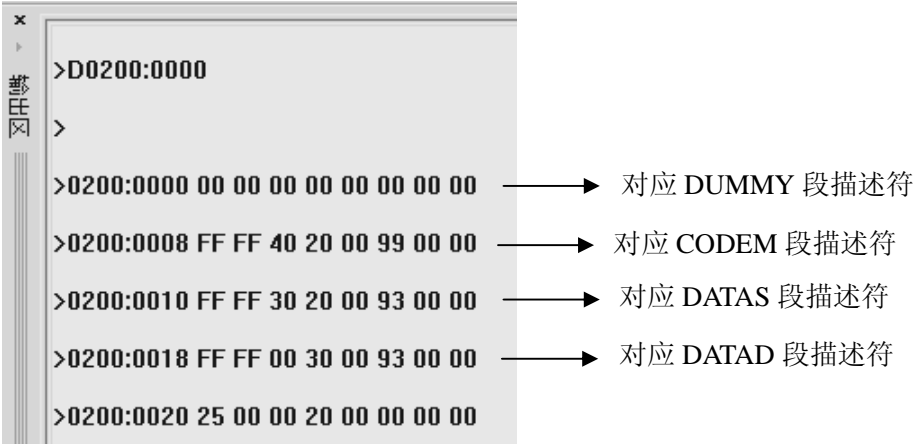


图 6.15

所以，通过以上方法，利用 Wmd86 软件提供的 Debug 调试功能，在做保护模式实验时，就可以查看描述符表内的各个段描述符的值。

6.1.4 局部描述表实验

本实验与上一实验所完成的功能相同，但要求将代码段安排在全局描述符表中，而将数据段安排在局部描述符表中。

1. 预备知识

全局描述符表 GDT 含有每一个任务都可能或可以访问的段的描述符，通常包含描述操作系统所使用的代码段、数据段和堆栈段的描述符，也包含多种特殊数据段描述符，如各个用于描述任务局部描述符表 LDT 的特殊数据段。在任务切换时，并不切换 GDT。

每个任务的局部描述符表 LDT 含有该任务自己的代码段、数据段和堆栈段的描述符，也



包含该任务所使用的一些门描述符，如任务门和调用门描述符等。随着任务的切换，系统当前的局部描述符表 LDT 也随之切换。

通过 LDT 可以使各任务私有的各个段与其他任务相隔离，从而达到受保护的目的。通过 GDT 可以使各任务都需要使用的段能够被共享。

## 2. 实验分析

本实验需要为代码段和数据段分别声明描述符，由于要求将数据段的描述符放入 LDT 表中，所以实验程序需要建立一张局部描述符表，并在 GDT 表中声明 LDT 表对应的描述符。描述符声明完成，还需要为它们定义相应的选择子。

实验程序在 DSEG 段中描述 GDT 表中的描述符，其中包括主代码段和 LDT 表的描述符。在 DSEG1 段中描述 LDT 表中的描述符，其中包括源数据段描述符目标数据段描述符。

由于主代码段需要访问的源数据段和目的数据段是在 LDT 表中声明的，所以在程序的初始需要执行装载 LDTR 的指令。装载 LDT 表使用的指令如下。

```
MOV     AX, LDT_SEL
LLDT    AX
```

(1) LDT 表对应段描述符: LDTABLE DESC <0FFFFH,,,ATLDT,,>

段属性说明:

```
G      :      0      ; 以字节为段界限粒度
D      :      0      ; 是 16 位的段
P      :      1      ; 描述符对地址转换有效/该描述符对应的段存在
DPL    :      0      ; 0 级段
DT     :      0      ; 描述符描述的是系统段或门描述符
TYPE   :      0x8     ; LDT 表
ATLDT EQU 82h        ;局部描述符表段类型值
```

(2) 数据段选择子。实验中的两个数据段均在 LDT 表中声明，则描述符对应的段选择子应该标记出来。

```
TIL          EQU    04h
DATAS_SEL    =      DATAS-LDT+TIL
DATAD_SEL    =      DATAD-LDT+TIL
```

(3) 目标数据段描述符的段基址给定为 00003000H。

在全局描述表 GDT 定义的过程中，首先需要定义一个空的描述符 DUMMY，作为定义的开始，然后再定义其它描述符。

本实验可实现将一个数据段中的数据搬运到另一个数据段所指定的地址空间中（指定地址给定为 00003000H）。源数据段 LDAT 值为 11H,22H,33H,44H,55H,66H,77H,88H，在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

## 3. 实验步骤

- (1) 编写实验程序（例程文件名为：6-1-2.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序，待程序运行停止。

## (4) 查看运行结果。

在输出区的调试栏输入：

D0000: 3000

然后回车，即可看到实验运行结果，将数据传输到了给定的地址空间中。

## 实验程序清单

```

;-----
;文件名:6-1-2.ASM
;功能:局部描述符及局部描述符表实验
;-----
INCLUDE          386SCD.INC
;数据段=====
DSEG             SEGMENT USE16                ;16 位数据段
GDT              LABEL    BYTE                ;全局描述符表
DUMMY            DESC     <>                  ;空描述符
CODEM            DESC     <0FFFFH,,,ATCE,,>   ;代码段描述符
LDTABLE          DESC     <0FFFFH,,,ATLDT,,>   ;局部描述符表段的描述
符
VGDT            PDESC    <GDTLEN-1,>          ;伪描述符

CODEM_SEL        =        CODEM-GDT           ;代码段选择子
LDT_SEL          =        LDTABLE-GDT         ;LDT 表选择子
GDTLEN           =        $-GDT               ;全局描述符表长度

DSEG             ENDS                        ;数据段定义结束
;-----
DSEG1            SEGMENT USE16                ;16 位数据段
LDT              LABEL    BYTE                ;局部描述符表
LDATAS           DESC     <0FFFFH,,,ATDW,,>   ;源数据段描述符
LDATAD           DESC     <0FFFFH,3000H,,ATDW,,> ;目标数据段描述符

LDTLEN           =        $-LDT               ;局部描述符表长度
LDATAS_SEL       =        LDATAS-LDT+TIL      ;源数据段选择子
LDATAD_SEL       =        LDATAD-LDT+TIL      ;目标数据段选择子
DSEG1            ENDS                        ;数据段定义结束
;-----
DATASSEG         SEGMENT PARA USE16           ;定义 16 位源数据段
TDATA            DB 11H,22H,33H,44H,55H,66H,77H,88H;定义源数据段数据
DATASLEN         =        $                   ;源数据段的长度

```

```

DATASSEG      ENDS                                ;源数据段定义结束
;代码段=====
CSEG          SEGMENT USE16                        ;16 位代码段
              ASSUME CS:CSEG
START         PROC
              ASSUME  DS:DSEG
              MOV     AX,DSEG
              MOV     DS,AX

              MOV     BX,16                        ;准备要加载到 GDTR 的
伪描述符
              MUL     BX                          ;计算并设置 GDT 基地址
              MOV     WORD PTR VGDTR.BASE,AX
              MOV     WORD PTR VGDTR.BASE+2,DX

              MOV     AX,CSEG
              MUL     BX
              MOV     WORD PTR CODEM.BASEL,AX      ;计算并设置代码段
基地址
              MOV     BYTE PTR CODEM.BASEM,DL
              MOV     BYTE PTR CODEM.BASEH,DH

              MOV     AX,DSEG1
              MUL     BX
              MOV     WORD PTR LDTABLE.BASEL,AX    ;计算并设置 LDT 表
段基地址
              MOV     BYTE PTR LDTABLE.BASEM,DL
              MOV     BYTE PTR LDTABLE.BASEH,DH

              ASSUME  DS:DSEG1
              MOV     AX,DSEG1
              MOV     DS,AX

              MOV     AX,DATASSEG
              MUL     BX
              MOV     WORD PTR LDATAS.BASEL,AX    ;计算并设置源数据段基
地址
              MOV     BYTE PTR LDATAS.BASEM,DL
              MOV     BYTE PTR LDATAS.BASEH,DH

```

```

        ASSUME    DS:DSEG
        MOV      AX,DSEG
        MOV      DS,AX

        LGDT     QWORD PTR VGDTR           ;装载 GDTR
        CLI                                           ;关中断
        ;切换到保护方式
        MOV      EAX,CR0
        OR       EAX,1
        MOV      CR0,EAX
        ;清指令预取队列,并真正进入保护方式
        JUMP16   <CODEM_SEL>,<OFFSET VIRTUAL>

VIRTUAL:    ;现在开始在保护方式下运行
        MOV      AX,LDT_SEL
        LLDT     AX                               ;装载 LDT 表寄存器 LDTR
        MOV      AX,LDATAS_SEL                   ;加载源数据段描述符
        MOV      DS,AX
        MOV      AX,LDATAD_SEL                   ;加载目标数据段描述符
        MOV      ES,AX
        XOR      DI,DI
        XOR      SI,SI
        MOV      CX,8                           ;设置数据长度
        REPZ     MOVSB                           ;传数
        ;切换回实模式
        MOV      EAX,CR0
        AND      AL,11111110B
        MOV      CR0,EAX
        ;清指令预取队列,进入实方式
        JUMP16   <SEG REAL>,<OFFSET REAL>

REAL:      ;现在又回到实方式
        STI                                           ;开中断

        MOV      AX,4C00H
        INT      21H                               ;程序终止

START      ENDP
;-----

```

```
CSEG          ENDS                                ;代码段定义结束
;-----
          END    START
```

## 6.2 特权级变换实验

### 6.2.1 实验目的

1. 掌握 JUMP、CALL、RETF 指令完成任务内变换转移的方法。
2. 熟悉保护模式下任务内特权级变换的方法。
3. 继续学习 GDT、LDT 表及选择子寻址的方法。

### 6.2.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 6.2.3 任务内无特权级变换的转移实验

#### 1. 预备知识

任务内无特权级变换的转移比较简单，在需要发生转移时，可用 JUMP、CALL、RETF 等指令进行相应段的转移。在用到 CALL 指令时，必须为之准备一个堆栈，以使得在调用 CALL 时，系统对 CALL 当前的程序运行点进行压栈保存，在遇到 RETF 返回指令时，进行弹栈返回。

任务内段间调用指令 CALL16 宏定义如下：

```
CALL16    MACRO    SELECTOR, OFFSET
           DB       9AH                ; 操作码
           DW       OFFSET            ; 16 位偏移量
           DW       SELECTOR          ; 段值或段选择子
           ENDM
```

JUMP16 宏定义如下：

```
JUMP16    MACRO    SELECTOR, OFFSET
           DB       0EAH              ; 操作码
           DW       OFFSET            ; 16 位偏移量
           DW       SELECTOR          ; 段值或段选择子
           ENDM
```

通常情况下，段间返回指令 RETF 与段间调用指令 CALL 对应。在利用段间调用指令 CALL 以任务内无特权级变换的方式转移到某个子程序后，在子程序内利用段间返回指令 RETF 以任务内无特权级变换的方式返回主程序。由于调用时无特权级变换，所以返回时也无特权级变换。

#### 2. 实验内容一

本实验需要在程序中安排 3 个相同特权级的段 (D1, D2, D3), 并在运行时能够实现从 D1 转移到 D2, 从 D2 转移到 D3, 在 D3 中将源数据段中的数据传输到目标数据段中, 利用 CALL、RETF 指令来实现。

实验程序在 DSEG 段中描述 GDT 中的描述符。先定义一个空的描述符表明 GDT 表的开始, 然后定义主程序段和 LDT 描述符。在 DSEG 段后, 用 DSEG1 段来描述 LDT 中的描述符, 其中包括源数据段描述符和目标数据段描述符。

为了体现任务的特性, 只将主程序段描述符和 LDT 段描述符安放在 GDT 中, 其余的所有代码段和数据段均放置在 LDT 中。由于实验内容中所有要求完成的都是任务内相同特权级段之间的转移, 所以不会涉及到堆栈的切换, 则在实验程序中只需要建立一张局部描述符表, 而不需要使用任务状态段。

本实验可实现将一个数据段中的数据搬运到另一个数据段所指定的地址空间中 (指定地址给定为 00003000H)。源数据段 TDATA 值为 11H,22H,33H,44H,55H,66H,77H,88H, 在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

### 3. 实验步骤

- (1) 编写实验程序 (例程文件名为: 6-2-1.ASM)。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序, 待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入:

D0000: 3000

然后回车, 即可看到实验运行结果, 将数据传输到了给定的地址空间中。

#### 实验程序清单

```

;-----
;文件名:6-2-1.ASM
;功能:通过 CALL, RET 指令任务内无特权级变换的转移
;-----
INCLUDE          386SCD.INC
;=====
DSEG             SEGMENT PARA USE16             ;16 位数据段
GDT              LABEL   BYTE                   ;全局描述符表
DUMMY            DESC    <>                     ;空描述符
CODEM            DESC    <0FFFFH,,,ATCE,,>      ;主代码段描述符
LDTABLE          DESC    <0FFFFH,,,ATLDT,,>      ;局部描述符表段的描述符
VGDT            PDESC    <GDTLEN-1,>            ;伪描述符

CODEM_SEL        =          0008H                ;主程序段选择子
LDT_SEL          =          0010H                ;局部描述符表段选择子

```

```

GDTLEN          =          $-GDT          ;全局描述符表长度

DSEG            ENDS                    ;数据段定义结束
;=====
=====
DSEG1           SEGMENT PARA USE16
LDT              LABEL  BYTE              ;局部描述符表
CODE1            DESC   <0FFFFH,,,ATCE,,> ;0 级代码段 1 描述符
CODE2            DESC   <0FFFFH,,,ATCE,,> ;0 级代码段 2 描述符
DSTACK           DESC   <0FFFFH,,,ATDW,,> ;0 级堆栈段描述符
DATAS            DESC   <0FFFFH,,,ATDW,,> ;源数据段描述符
DATAO            DESC   <0FFFFH,3000H,,ATDW,,> ;目的数据段描述符
LDTLEN           =          $-LDT          ;局部描述符表长度
CODE1_SEL        =          0000H+TIL      ;CODE1-LDT+TIL      ;代码段 1 的选
择子
CODE2_SEL        =          0008H+TIL      ;CODE2-LDT+TIL      ;代码段 2 的选
择子
DSTACK_SEL       =          0010H+TIL      ;DSTACK-LDT+TIL      ;0 级堆栈描述
符选择子
DATAS_SEL        =          0018H+TIL      ;DATAS-LDT+TIL      ;源数据段描述符选
择子
DATAO_SEL        =          0020H+TIL      ;DATAO-LDT+TIL      ;目的数据段描
述符选择子
DSEG1            ENDS                    ;数据段定义结束
;-----
STACKSEG        SEGMENT PARA USE16          ;定义 16 位的 0 级堆栈段
STACKLEN        =          128
                DB          128 DUP(?)      ;定义 128 个缓冲区字节单元
STACKSEG ENDS          ;0 级堆栈段结束
;-----
DATASSEG        SEGMENT PARA USE16          ;定义 16 位数据段
TDATA          DB 11H,22H,33H,44H,55H,66H,77H,88H ;定义源数据段数据
DATASLEN = $          ;数据段的长度
DATASSEG ENDS          ;源数据段定义结束
;=====
CODE1SEG        SEGMENT PARA USE16          ;任务代码段 1
                ASSUME  CS:CODE1SEG

                CALL16 CODE2_SEL,0          ;转向代码段 2

```



```

                                RETF                                ;段间返回

CODE1LEN =  $
CODE1SEG      ENDS
;-----
CODE2SEG      SEGMENT PARA USE16                                ;任务代码段 2
                ASSUME  CS:CODE2SEG
                MOV     AX, DATAS_SEL
                MOV     DS,AX                                    ;加载源数据段描述符
                MOV     AX, DATAO_SEL
                MOV     ES,AX                                    ;加载目标数据段描述符
                XOR     SI,SI
                XOR     DI,DI                                    ;设置指针初值
                MOV     CX, 8                                    ;设置传送长度

M1:            MOVSB                                            ;传送
                LOOP    M1

                RETF                                            ;段间返回,返回至代码段 1

CODE2LEN      =  $
CODE2SEG      ENDS                                            ;代码段 C 定义结束
;=====
CSEG          SEGMENT PARA USE16
                ASSUME CS:CSEG
START         PROC
                ASSUME  DS:DSEG
                MOV     AX,DSEG
                MOV     DS,AX

                MOV     BX,16                                    ;准备要加载到 GDTR
                MUL     BX                                       ;计算并设置 GDT 基址
                MOV     WORD PTR VGDTR.BASE,AX
                MOV     WORD PTR VGDTR.BASE+2,DX

                MOV     AX,CSEG

```

的伪描述符

址

代码段基地址	MUL	BX	
	MOV	WORD PTR CODEM.BASEL,AX	;计算并设置主
	MOV	BYTE PTR CODEM.BASEM,DL	
	MOV	BYTE PTR CODEM.BASEH,DH	
LDT 表段基地址	MOV	AX,DSEG1	
	MUL	BX	
	MOV	WORD PTR LDTABLE.BASEL,AX	;计算并设置
	MOV	BYTE PTR LDTABLE.BASEM,DL	
	MOV	BYTE PTR LDTABLE.BASEH,DH	
码段 1 基地址	ASSUME	DS:DSEG1	
	MOV	AX,DSEG1	
	MOV	DS,AX	
	MOV	AX,CODE1SEG	
	MUL	BX	
	MOV	WORD PTR CODE1.BASEL,AX	;计算并设置代
	MOV	BYTE PTR CODE1.BASEM,DL	
	MOV	BYTE PTR CODE1.BASEH,DH	
	MOV	AX,CODE2SEG	
	MUL	BX	
码段 2 基地址	MOV	WORD PTR CODE2.BASEL,AX	;计算并设置代
	MOV	BYTE PTR CODE2.BASEM,DL	
	MOV	BYTE PTR CODE2.BASEH,DH	
	MOV	AX,STACKSEG	
基地址	MUL	BX	
	MOV	WORD PTR DSTACK.BASEL,AX	;计算并设置堆栈段
	MOV	BYTE PTR DSTACK.BASEM,DL	
	MOV	BYTE PTR DSTACK.BASEH,DH	
	MOV	AX,DATASSEG	

```

        MUL        BX
        MOV        WORD PTR DATAS.BASEL,AX           ;计算并设置源
数据段基地址
        MOV        BYTE PTR DATAS.BASEM,DL
        MOV        BYTE PTR DATAS.BASEH,DH

        ASSUME     DS:DSEG
        MOV        AX,DSEG
        MOV        DS,AX

        LGDT       QWORD PTR VGDT                    ;装载 GDT 表寄存器
GDTR
        CLI                               ;关中断
        MOV        EAX,CR0                     ;切换到保护方式
        OR         EAX,1
        MOV        CR0,EAX
        JUMP16     <CODEM_SEL>,<OFFSET_VIRTUAL>      ;清指令预取队列,
并真正进入保护方式

        VIRTUAL:                                   ;现在开始在保护方式下运行
        MOV        AX,LDT_SEL
        LLDT       AX                             ;装载 LDT 表寄存器 LDTR

        CALL16     CODE1_SEL,0                    ;跳转到演示任务代码段
1
        REAL1:                                       ;切换回实模式
        MOV        EAX,CR0
        AND        AL,11111110B
        MOV        CR0,EAX
        JUMP16     <SEG_REAL>,<OFFSET_REAL>          ;清指令预取队列,进
入实方式

        REAL:                                       ;现在又回到实方式
        STI                               ;关中断

        MOV        AX,4C00H
        INT        21H                             ;程序终止

```

```

START          ENDP
;-----
CSEG           ENDS
               END      START

```

#### 4. 实验内容二

在保护模式下采用 JMP 和 RETF 指令来实现实验内容一的实验要求。

#### 5. 实验步骤

- (1) 编写实验程序（例程文件名为：6-2-2.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序，待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入：

D0000: 3000

然后回车，即可看到实验运行结果，将数据传输到了给定的地址空间中。

#### 实验程序清单

```

;-----
;文件名:6-2-2.ASM
;功能:通过 JMP, RET 指令任务内无特权级变换的转移
;-----
INCLUDE          386SCD.INC
;数据段=====
DSEG             SEGMENT PARA USE16             ;16 位数据段
GDT              LABEL    BYTE                  ;全局描述符表
DUMMY            DESC     <>                    ;空描述符
CODEM            DESC     <0FFFFH,,,ATCE,,>     ;主程序段描述符
LDTABLE          DESC     <0FFFFH,,,ATLDT,,>     ;局部描述符表段的描述符
VGDRTR          PDESC     <GDTLEN-1,>           ;伪描述符
GDTLEN           =        $-GDT                 ;全局描述符表长度
CODEM_SEL        =        0008H                 ;主程序段选择子
LDT_SEL          =        0010H                 ;局部描述符表段选择子
DSEG             ENDS                          ;数据段定义结束
;=====
=====
DSEG1            SEGMENT PARA USE16             ;16 位数据段
LDT              LABEL    BYTE                  ;局部描述符表
CODE1            DESC     <0FFFFH,,,ATCE,,>     ;0 级代码段 1
CODE2            DESC     <0FFFFH,,,ATCE,,>     ;0 级代码段 2

```

```

DSTACK      DESC    <0FFFFH,,,ATDW,,>      ;0 级堆栈段描述符
DATAS       DESC    <0FFFFH,,,ATDW,,>      ;源数据段描述符
DATAO       DESC    <0FFFFH,3000H,,,ATDW,,> ;目的数据段描述符
LDTLEN      =        $-LDT                  ;LDT 表长度
CODE1_SEL   =        CODE1-LDT+TIL          ;代码段 1 的选择子
CODE2_SEL   =        CODE2-LDT+TIL          ;代码段 2 的选择子
DSTACK_SEL  =        DSTACK-LDT+TIL         ;0 级堆栈描述符选择子
DATAS_SEL   =        DATAS-LDT+TIL          ;源数据段描述符选择子
DATAO_SEL   =        DATAO-LDT+TIL         ;目的数据段描述符选择子

子
DSEG1       ENDS                          ;数据段定义结束
;-----
STACKSEG    SEGMENT PARA USE16             ;定义 16 位的 0 级堆栈段
STACKLEN    =        128
            DB        128 DUP(?)           ;定义 128 个缓冲区字节单元
STACKSEG    ENDS                          ;0 级堆栈段结束
;-----
DATASSEG    SEGMENT PARA USE16             ;定义 16 位数据段
TDATA      DB 11H,22H,33H,44H,55H,66H,77H,88H ;定义源数据段数据
DATASLEN    =        $                     ;数据段的长度
DATASSEG    ENDS                          ;源数据段定义结束
;代码段=====
CODE1SEG     SEGMENT PARA USE16             ;任务代码段 1
            ASSUME    CS:CODE1SEG

            CALL16 CODE2_SEL,0              ;转向代码段 2

            JUMP16    <CODEM_SEL>,<OFFSET REAL1> ;跳转至主代码

段
CODE1LEN    =        $
CODE1SEG     ENDS                          ;代码段 1 定义结束
;-----
CODE2SEG     SEGMENT PARA USE16             ;任务代码段 2
            ASSUME    CS:CODE2SEG

            MOV       AX, DATAS_SEL
            MOV       DS,AX                  ;加载源数据段描述符
            MOV       AX, DATAO_SEL

```

```

                                MOV     ES,AX                ;加载目标数据段描述符
                                XOR     SI,SI
                                XOR     DI,DI                ;设置指针初值
                                MOV     CX, 8                ;设置传送长度
M1:    MOVSB                    ;传送
                                LOOP    M1

                                RETF                       ;段间返回

CODE2LEN      =      $
CODE2SEG      ENDS                    ;代码段 2 定义结束
;代码段=====
CSEG          SEGMENT PARA USE16      ;16 位主代码段
                ASSUME CS:CSEG
START         PROC
                ASSUME  DS:DSEG
                MOV     AX,DSEG
                MOV     DS,AX

                MOV     BX,16          ;准备要加载到 GDTR 的伪描
述符
                MUL     BX              ;计算并设置 GDT 基地址
                MOV     WORD PTR VGDTR.BASE,AX
                MOV     WORD PTR VGDTR.BASE+2,DX

                MOV     AX,CSEG
                MUL     BX
                MOV     WORD PTR CODEM.BASEL,AX      ;计算并设置主代码
段基地址
                MOV     BYTE PTR CODEM.BASEM,DL
                MOV     BYTE PTR CODEM.BASEH,DH

                MOV     AX,DSEG1
                MUL     BX
                MOV     WORD PTR LDTABLE.BASEL,AX    ;计算并设置 LDT 表
段基地址
                MOV     BYTE PTR LDTABLE.BASEM,DL
                MOV     BYTE PTR LDTABLE.BASEH,DH

                ASSUME  DS:DSEG1

```

	MOV	AX,DSEG1	
	MOV	DS,AX	
1 基地址	MOV	AX,CODE1SEG	
	MUL	BX	
	MOV	WORD PTR CODE1.BASEL,AX	;计算并设置代码段
	MOV	BYTE PTR CODE1.BASEM,DL	
	MOV	BYTE PTR CODE1.BASEH,DH	
2 基地址	MOV	AX,CODE2SEG	
	MUL	BX	
	MOV	WORD PTR CODE2.BASEL,AX	;计算并设置代码段
	MOV	BYTE PTR CODE2.BASEM,DL	
	MOV	BYTE PTR CODE2.BASEH,DH	
址	MOV	AX,STACKSEG	
	MUL	BX	
	MOV	WORD PTR DSTACK.BASEL,AX	;计算并设置堆栈段基址
	MOV	BYTE PTR DSTACK.BASEM,DL	
	MOV	BYTE PTR DSTACK.BASEH,DH	
段基地址	MOV	AX,DATASEG	
	MUL	BX	
	MOV	WORD PTR DATAS.BASEL,AX	;计算并设置源数据
	MOV	BYTE PTR DATAS.BASEM,DL	
	MOV	BYTE PTR DATAS.BASEH,DH	
GDTR	ASSUME	DS:DSEG	
	MOV	AX,DSEG	
	MOV	DS,AX	
	LGDT	QWORD PTR VGDTR	;装载 GDT 表寄存器
	CLI		;关中断

```

        MOV     EAX,CR0                      ;切换到保护方式
        OR      EAX,1
        MOV     CR0,EAX

        JUMP16  <CODEM_SEL>,<OFFSET VIRTUAL>;清指令预取队列,并真正进入保护方式

VIRTUAL:                                ;现在开始在保护方式下运行
        MOV     AX,LDT_SEL
        LLDT    AX                          ;装载 LDT 表寄存器 LDTR

        JUMP16  CODE1_SEL,0                 ;跳转到任务代码段 1

REAL1:                                ;切换回实模式
        MOV     EAX,CR0
        AND     AL,11111110B
        MOV     CR0,EAX
        ;清指令预取队列,进入实方式
        JUMP16  <SEG REAL>,<OFFSET REAL>

REAL:                                ;现在又回到实方式
        STI                                     ;关中断

        MOV     AX,4C00H
        INT     21H                          ;程序终止

START      ENDP
;-----
CSEG      ENDS
          END      START

```

## 6.2.4 任务内有特权级变换的转移实验

### 1. 预备知识

在一个任务之内，可以存在四种特权级，所以常常会发生不同特权级之间的变换。在同一个任务内，实现特权级从外层到内层变换的普通途径是：使用段间调用指令 CALL，通过调用门进行转移；实现特权级从内层到外层变换的途径是：使用段间返回指令 RETF。

当段间转移指令 JUMP 和段间调用指令 CALL 所含指针的选择子指示调用门描述符时，



就可实现通过调用门的转移。

调用门描述某个子程序的入口，调用门内的选择子必须指向代码段描述符，调用门内的偏移是对应代码段内的偏移。调用门描述符调用转移的入口点，包含目标地址的段及偏移量的 48 位全指针。在执行通过调用门的段间转移指令 JUMP 或段间调用指令 CALL 时，指令所含指针内的选择子用于确定调用门，而偏移被丢弃，把调用门内的 48 位全指针，作为目标地址指针进行转移。

调用门是门描述符的一种。所以，在取出调用门内的 48 位全指针，把它作为目标地址指针向目标代码段转移之前，要进行特权级检测。只有通过调用门描述符才可实现低特权级向高特权级的转移。

门描述符不描述某种内存段，而是描述控制转移的入口点。通过这种门，可实现任务内特权级的变换和任务间的切换。所以，这种门也称为控制门。门描述符结构定义如下：

```
GATE          STRUC
OFFSETL       DW          0          ; 32 位偏移的低 16 位
SELECTOR      DW          0          ; 选择子
DCOUNT        DB          0          ; 双字计数
GTYPE         DB          0          ; 类型
OFFSETH       DW          0          ; 32 位偏移的高 16 位
GATE          ENDS
```

利用门描述符结构类型 GATE 能方便地在程序中说明门描述符。

## 2. 实验内容

本实验需要在程序中安排 2 个 0 级的代码段 (D1, D2) 以及 1 个 3 级代码段 (D3)，并在运行时能够实现从 D1 转移到 D2，从 D2 转移到 D3，在 D3 中将源数据段中的数据传输到目标数据段中，利用 JUMP、CALL、RETF 指令来实现。

实验程序在 TDATASEG 段中描述 GDT 中的描述符。先定义一个空的描述符表明 GDT 表的开始，然后定义主程序段、LDT 描述符和任务状态段 TSS 描述符。在 TDATASEG 段后，用 LDTSEG 段来描述 LDT 中的描述符，其中包括源数据段描述符和目标数据段描述符。

为了体现任务的特性，只将主程序段描述符、LDT 段描述符和任务状态段 TSS 描述符安放在 GDT 中，其余的所有代码段和数据段均放置在 LDT 中。另外在 LDT 表中还定义了一个 TOCODE0 的任务调用门描述符，在由 3 级代码段向 0 级代码段转移时必须通过此调用门进行转移。

本实验可实现将一个数据段中的数据搬移到另一个数据段所指定的地址空间中（指定地址给定为 00003000H）。源数据段 DTEST 值为 11H,22H,33H,44H,55H,66H,77H,88H，在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

## 3. 实验步骤

- (1) 编写实验程序（例程文件名为：6-2-3.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序，待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入：

D0000: 3000

然后回车, 即可看到实验运行结果, 将数据传输到了给定的地址空间中。

### 实验程序清单

```

;-----
;文件名:6-2-3.ASM
;功能:任务内有特权级变换的转移
;-----
INCLUDE          386SCD.INC
;-----
TDATESEG          SEGMENT PARA USE16          ;全局描述符表数据段(16
位)
    GDT              LABEL    BYTE              ;全局描述符表
    DUMMY            DESC     <>                ;空描述符
    NORMAL            DESC     <0FFFFH,,,ATDW,,> ;规范段描述符
    CODEM             DESC     <0FFFFH,,,ATCE,,> ;主程序段描述符
    LDTABLE           DESC     <0FFFFH,,,ATLDT,,> ;局部描述符表段的描述符
    TSSTABLE          DESC     <0FFFFH,,,AT386TSS,,> ;任务状态段 TSS 描述符
;-----
    GDTLEN            =        $-GDT            ;全局描述符表长度
    NORMAL_SEL        =        NORMAL-GDT        ;规范段描述符选择子
    CODEM_SEL         =        CODEM-GDT         ;主程序段选择子
    LDT_SEL           =        LDTABLE-GDT        ;局部描述符表段选择子
    TSS_SEL           =        TSSTABLE-GDT        ;任务状态段选择子
;-----
    VGDR              PDESC     <GDTLEN-1,>      ;定义 GDT 的伪描述符
VGDR
    SPVAR             DW        ?
    SSVAR             DW        ?                ;定义实方式下的堆栈指针
;-----
TDATESEG          ENDS          ;全局描述符表段定义结束
;-----
LDTSEG            SEGMENT USE16          ;局部描述符表数据段
(16 位)
    LDT               LABEL    BYTE              ;局部描述符表
    DCODE0            DESC     <0FFFFH,,,ATCE,D32,> ;代码段 0 描述符(32 位
段,DPL=0)
    DCODE3            DESC     <0FFFFH,,,ATCE+DPL3,,> ;代码段 3 描述符(16 位
段,DPL=3)

```

```

    DSTACK0      DESC    <0FFFFH,,ATDW,,>          ;0 级堆栈段描述符
(DPL=0)

    DSTACK3      DESC    <0FFFFH,,ATDW+DPL3,,>      ;3 级堆栈段描述符
(DPL=3)

    DDATAS       DESC    <0FFFFH,,ATDW+DPL3,,>      ;源数据段描述符
(DPL=3)

    DDATAO       DESC    <0FFFFH,3000H,,ATDW+DPL3,,> ;目的数据段描述符
(DPL=3)

    TOCODE0      GATE     <OFFSET  PEND-  OFFSET
PSTART,DCODE0_SEL,,AT386CGATE+DPL3,>;指向代码段 0 的任务描述符
;-----
    DCODE0_SEL   =        DCODE0 -LDT+TIL          ;代码段 0 描述符选择子
(RPL=0)

    DCODE3_SEL   =        DCODE3 -LDT+TIL+RPL3      ;代码段 3 描述符选择子
(RPL=3)

    DSTACK0_SEL  =        DSTACK0-LDT+TIL          ;0 级堆栈描述符选择子
(RPL=0)

    DSTACK3_SEL  =        DSTACK3-LDT+TIL+RPL3      ;3 级堆栈描述符选择子
(RPL=3)

    DDATAS_SEL   =        DDATAS -LDT+TIL+RPL3      ;源数据段描述符选择子
(RPL=3)

    DDATAO_SEL   =        DDATAO -LDT+TIL+RPL3      ;目标数据段描述符选择子
(RPL=3)

    TOCODE0_SEL  =        (TOCODE0-LDT)+TIL          ;调用门描述符选择子
    LDTLEN       =        $-LDT
    LDTSEG       ENDS                                  ;局部描述符表段定义结束
;-----
    TSSSEG       SEGMENT PARA USE16                  ;任务状态段 TSS
                DD        0                          ;BACK
                DD        DSTACK0LEN                  ;0 级堆栈指针
                DD        DSTACK0_SEL                  ;初始化
                DD        0                          ;1 级堆栈指针
                DD        0                          ;初始化
                DD        0                          ;2 级堆栈指针
                DD        0                          ;未初始化
                DD        0                          ;CR3
                DD        0                          ;EIP
                DD        0                          ;EFLAGS
                DD        0                          ;EAX

```

```

        DD      0                      ;ECX
        DD      0                      ;EDX
        DD      0                      ;EBX
        DD      0                      ;ESP
        DD      0                      ;EBP
        DD      0                      ;ESI
        DD      0                      ;EDI
        DD      0                      ;ES
        DD      0                      ;CS
        DD      0                      ;SS
        DD      0                      ;DS
        DD      0                      ;FS
        DD      0                      ;GS
        DD      LDT_SEL                ;LDT
        DW      0                      ;调试陷阱标志
        DW      $+2                    ;指向 I/O 许可位图
        DW      0FFFFH                ;I/O 许可位图结束标志
TSSLEN      =      $
TSSSEG      ENDS                      ;任务状态段 TSS 结束
;-----
DSTACK0SEG  SEGMENT PARA USE16        ;定义 16 位的 0 级堆栈段
DSTACK0LEN  =      128
        DB      DSTACK0LEN DUP(?)    ;定义 128 个缓冲区字节单元
DSTACK0SEG  ENDS                      ;0 级堆栈段结束
;-----
DSTACK3SEG  SEGMENT PARA USE16        ;3 级堆栈段
DSTACK3LEN  =      128
        DB      DSTACK3LEN DUP(?)
DSTACK3SEG  ENDS
;-----
DDATASSEG   SEGMENT PARA USE16        ;定义 16 位数据段
DTEST      DB  11H,22H,33H,44H,55H,66H,77H,88H ;定义源数据段数据
DDATASLEN   =      $                  ;数据段的长度
DDATASSEG   ENDS                      ;源数据段定义结束
;-----
CODEMSEG    SEGMENT PARA USE16
        ASSUME  CS:CODEMSEG
START       PROC
        ASSUME  DS:TDATESEG

```

```

        MOV     AX,TDATASEG
        MOV     DS,AX

        MOV     BX,16                      ; 准 备 要 加 载 到
GDTR 的伪描述符
        MUL     BX                      ;计算并设置 GDT 基
地址
        MOV     WORD PTR VGDTR.BASE,AX
        MOV     WORD PTR VGDTR.BASE+2,DX

        MOV     AX,CODEMSEG
        MUL     BX
        MOV     WORD PTR CODEM.BASEL,AX      ; 计 算 并 设
置主程序段基地址
        MOV     BYTE PTR CODEM.BASEM,DL
        MOV     BYTE PTR CODEM.BASEH,DH

        MOV     AX,LDTSEG
        MUL     BX
        MOV     WORD PTR LDTABLE.BASEL,AX    ; 计 算 并 设
置 LDT 表段基地址
        MOV     BYTE PTR LDTABLE.BASEM,DL
        MOV     BYTE PTR LDTABLE.BASEH,DH

        MOV     AX,TSSSEG
        MUL     BX
        MOV     WORD PTR TSSTABLE.BASEL,AX   ; 计 算 并 设
置任务段 TSS 基地址
        MOV     BYTE PTR TSSTABLE.BASEM,DL
        MOV     BYTE PTR TSSTABLE.BASEM,DL
        MOV     BYTE PTR TSSTABLE.ATTRIBUTES,AT386TSS;任务段
TSS 的段属性
        ;-----
        ASSUME  DS:LDTSEG
        MOV     AX,LDTSEG
        MOV     DS,AX

        MOV     AX,DCODE0SEG
        MUL     BX

```

码段 0 基地址	MOV	WORD PTR DCODE0.BASEL,AX	;计算并设置代
	MOV	BYTE PTR DCODE0.BASEM,DL	
	MOV	BYTE PTR DCODE0.BASEH,DH	
	MOV	AX,DCODE3SEG	
	MUL	BX	
码段 3 基地址	MOV	WORD PTR DCODE3.BASEL,AX	;计算并设置代
	MOV	BYTE PTR DCODE3.BASEM,DL	
	MOV	BYTE PTR DCODE3.BASEH,DH	
	MOV	AX,DSTACK0SEG	
	MUL	BX	
置堆栈段 0 基地址	MOV	WORD PTR DSTACK0.BASEL,AX	;计算并设
	MOV	BYTE PTR DSTACK0.BASEM,DL	
	MOV	BYTE PTR DSTACK0.BASEH,DH	
	MOV	AX,DSTACK3SEG	
	MUL	BX	
置堆栈段 3 基地址	MOV	WORD PTR DSTACK3.BASEL,AX	;计算并设
	MOV	BYTE PTR DSTACK3.BASEM,DL	
	MOV	BYTE PTR DSTACK3.BASEH,DH	
	MOV	AX,DDATASSEG	
	MUL	BX	
数据段基地址	MOV	WORD PTR DDATAS.BASEL,AX	;计算并设置源
	MOV	BYTE PTR DDATAS.BASEM,DL	
	MOV	BYTE PTR DDATAS.BASEH,DH	
;-----			
	ASSUME	DS:TDATESEG	
	MOV	AX,TDATESEG	
	MOV	DS,AX	
	MOV	SSVAR,SS	
	MOV	SPVAR,SP	;保存实方式下的堆栈指针

```

                LGDT    QWORD PTR VGDT     ;装载 GDT 表寄存器
GDTR

                CLI                      ;关中断
                MOV     EAX,CR0             ;切换到保护方式
                OR      EAX,1
                MOV     CR0,EAX
                JUMP16  <CODEM_SEL>,<OFFSET_VIRTUAL> ;清指令预取队列,
并真正进入保护方式
                VIRTUAL:                   ;现在开始在保护方式下运行
                MOV     AX,TSS_SEL         ;装载 LDT 表
                LTR     AX
                MOV     AX,LDT_SEL         ;装载 TSS 表
                LLDT    AX

                JUMP16  DCODE0_SEL,0       ;转移到 0 级程序段
                REAL1:                      ;切换回实模式
                MOV     AX,NORMAL_SEL      ;把规范段描述符装
入各数据段寄存器
                MOV     DS,AX
                MOV     ES,AX
                MOV     FS,AX
                MOV     GS,AX
                MOV     SS,AX

                MOV     EAX,CR0             ;切换到实模式
                AND     AL,11111110B
                MOV     CR0,EAX

                JUMP16  <SEG_REAL>,<OFFSET_REAL> ;清指令预取队列,进
入实方式
                REAL:                       ;现在又回到实方式
                MOV     AX,TDATA_SEG
                MOV     DS,AX

                LSS     SP,DWORD PTR SPVAR ;恢复实方式堆栈指针

                STI                      ;开中断

```

```
MOV    AX,4C00H
```

```
INT     21H                                ;程序终止
```

```
CODEMLN    = $
```

```
START      ENDP
```

```
;-----
```

```
DCODE0SEG    SEGMENT PARA USE32                                ;0 级的 32 位代
```

码段

```
ASSUME  CS:DCODE0SEG
```

```
PSTART:      MOV    AX,DSTACK0_SEL                                ;建立 0 级堆栈
```

```
MOV          SS,AX
```

```
MOV          ESP,DSTACK0LEN-1
```

```
PUSH         DWORD PTR DSTACK3_SEL
```

```
PUSH         DWORD PTR DSTACK3LEN                                ;压入 3 级堆栈指针
```

```
PUSH         DWORD PTR DCODE3_SEL
```

```
PUSH         OFFSET 0                                            ;压入入口点
```

```
RETF                                                ;由 0 级程序段转移到 3 级程
```

序段

```
PEND:      JUMP32  <CODEM_SEL>,<OFFSET REAL1>  ;跳转到主程序
```

段

```
DCODE0LEN  =      $
```

```
;-----
```

```
DCODE0SEG  ENDS
```

```
;-----
```

```
DCODE3SEG  SEGMENT PARA USE16                                ;3 级的 16 位代码段
```

```
ASSUME  CS:DCODE3SEG
```

```
MOV      AX,DDATAS_SEL
```

```
MOV      DS,AX
```

;加载源数据段描述符

```
MOV      AX,DDATAO_SEL
```

```
MOV      ES,AX
```

;加载目标数据段描述符

```
XOR      SI,SI
```

```
XOR      DI,DI
```

;设置指针初值

```
MOV      CX,8
```

;设置传送长度

```
M1:      MOVS      B
```

;传送

```
LOOP     M1
```



CALL16 TOCODE0\_SEL,0 ;由 3 级程序段转移  
到 0 级程序段

DCODE3LEN = \$  
DCODE3SEG ENDS

;-----

CODMSEG ENDS  
END START

## 6.3 任务切换实验

### 6.3.1 实验目的

1. 掌握任务状态段 TSS 的建立及使用方法。
2. 学习并掌握保护模式下任务切换的方法。

### 6.3.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 6.3.3 使用 JMP 指令实现任务切换实验（无特权级）

#### 1. 预备知识

任务之间进行切换。每个任务都有一个任务状态段 TSS，用于保存任务的有关信息，在任务内变换特权级和任务切换时，要使用这些信息。为了控制任务内发生特权变换的转移，为了控制任务切换，一般要通过控制门进行这些转移。

任务状态段 TSS 是保存一个任务重要信息的特殊段。任务状态段描述符用于描述这样的系统段。任务状态段寄存器 TR 的可见部分含有当前任务状态段描述符的选择子，TR 的不可见部分含有当前任务状态段的段基地址和段界限等信息。

TSS 在任务切换过程中起着重要的作用，通过它实现任务的挂起与恢复。所谓任务切换是指，挂起当前正在执行的任务，恢复另一个任务的执行。在任务切换过程中，首先，处理器中各寄存器的当前值被自动地保存到 TR 所指定的 TSS 中；然后，下一任务的 TSS 的选择子被装入 TR；最后从 TR 所指定的 TSS 中取出各寄存器的值送到处理器的各寄存器中。由此可见，通过在 TSS 中保存任务现场各寄存器状态的完整映像，实现任务的切换。

任务状态段 TSS 的基本格式由 104 字节组成。这 104 字节的基本格式是不可改变的，但在此之外系统软件还可定义若干附加信息。基本的 104 字节可分为链接字段区域、内存堆栈指针区域、地址映射寄存器区域、寄存器保存区域和其它字段等五个区域。

用结构类型定义 TSS 如下：

TSS	STRUC		
TRLINK	DW	0	；链接字段
	DW	0	；不使用,置为 0
TRESP0	DD	0	；0 级堆栈指针
TRSS0	DW	0	；0 级堆栈段寄存器

	DW	0	; 不使用,置为 0
TRESP1	DD	0	; 1 级堆栈指针
TRSS1	DW	0	; 1 级堆栈段寄存器
	DW	0	; 不使用,置为 0
TRESP2	DD	0	; 2 级堆栈指针
TRSS2	DW	0	; 2 级堆栈段寄存器
	DW	0	; 不使用,置为 0
TRCR3	DD	0	; CR3
TREIP	DD	0	; EIP
TREFLAG	DD	0	; EFLAGS
TREAX	DD	0	; EAX
TRECX	DD	0	; ECX
TREDX	DD	0	; EDX
TREBX	DD	0	; EBX
TRESP	DD	0	; ESP
TREBP	DD	0	; EBP
TRESI	DD	0	; ESI
TREDI	DD	0	; EDI
TRES	DW	0	; ES
	DW	0	; 不使用,置为 0
TRCS	DW	0	; CS
	DW	0	; 不使用,置为 0
TRSS	DW	0	; SS
	DW	0	; 不使用,置为 0
TRDS	DW	0	; DS
	DW	0	; 不使用,置为 0
TRFS	DW	0	; FS
	DW	0	; 不使用,置为 0
TRGS	DW	0	; GS
	DW	0	; 不使用,置为 0
TRLDTR	DW	0	; LDTR
	DW	0	; 不使用,置为 0
TRTRIP	DW	0	; 调试陷阱标志(只用位 0)
TRIOMAP	DW	\$+2	; 指向 I/O 许可位图区的段内偏移
TSS	ENDS		

装载 TSS 表指令如下:

```
MOV     AX, TSS_SEL
LTR     AX
```

## 2. 实验内容

本实验要求使用 JUMP 指令实现任务切换。实验由主程序进入，先装入任务 0，再跳到任务 1 完成一段数据的传输。当任务 1 的工作完成后，进行任务切换，返回任务 0。

任务的切换可以通过 TSS 段和任务门完成，在实验中，从任务 0 切换到任务 1 时使用了 TSS 段，从任务 1 切换到任务 0 时使用了任务门。为了实现切换，程序需要为每个任务建立一个 TSS 段，并为每个任务建立自己的代码段、数据段和堆栈段等。

本实验可实现将一个数据段中的数据搬移到另一个数据段所指定的地址空间中（指定地址给定为 00003000H）。源数据段 DTEST 值为 11H,22H,33H,44H,55H,66H,77H,88H，在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

装载任务 TSS 到 TR 寄存器中可用下面两条指令：

```
MOV     AX, TSS0_SEL
LTR     AX
```

## 3. 实验步骤

- (1) 编写实验程序（例程文件名为：6-3-1.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序，待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入：

D0000: 3000

然后回车，即可看到实验运行结果，将数据传输到了给定的地址空间中。

### 实验程序清单

```
;-----
;文件名:6-3-1.ASM
;功能:使用 JMP 指令实现任务切换实验（不反映不同特权级的情况）
;-----
INCLUDE          386SCD.INC
;-----
DSEG              SEGMENT PARA USE16                      ;全局描述符表数据段(16
位)
GDT               LABEL  BYTE                               ;全局描述符表
DUMMY            DESC  <>                                   ;空描述符
NORMAL           DESC  <0FFFFH,,,ATDW,,>                 ;规范段描述符
CODEM            DESC  <0FFFFH,,,ATCE,,>                 ;主程序段描述符
LDTABLE1         DESC  <0FFFFH,,,ATLDT,,>                ;局部描述符表段的描述符
TSSTABLE0        DESC  <0FFFFH,,,,>                      ;任务 0 状态段 TSS 描述符
TSSTABLE1        DESC  <0FFFFH,,,,>                      ;任务 1 状态段 TSS 描述符
DSTACK0          DESC  <0FFFFH,,,ATDW,,>                 ;任务 0 的 0 级堆栈段描
```

述符

```

DDATA0          DESC  <0FFFFH,,,ATDW,,>      ;任务 0 的数据段描述符
;-----
NORMAL_SEL      =      NORMAL-GDT              ;规范段描述符选择子
CODEM_SEL       =      CODEM-GDT               ;主程序段选择子
LDT1_SEL        =      LDTABLE1-GDT            ;局部描述符表段选择子
TSS0_SEL        =      TSSTABLE0-GDT           ;任务 0 状态段选择子
TSS1_SEL        =      TSSTABLE1-GDT           ;任务 1 状态段选择子
DSTACK0_SEL     =      DSTACK0-GDT             ;任务 0 的堆栈段选择子
DDATA0_SEL      =      DDATA0-GDT              ;任务 0 的数据段描述符对

```

应选择子

```

GDTLEN          =      $-GDT                    ;全局描述符表长度
;-----
VGDTTR          PDESC  <GDTLEN-1,>              ;定义 GDT 的伪描述符

```

VGDTTR

```

SPVAR           DW  ?
SSVAR           DW  ?                          ;保存实模式下堆栈指针
;-----

```

```

DSEG            ENDS                          ;全局描述符表段定义结束
;-----

```

```

DSEG1           SEGMENT PARA USE16             ;局部描述符表数据

```

段(16 位)

```

LDT1            LABEL  BYTE                    ;局部描述符表
DCODE1          DESC  <0FFFFH,,,ATCE,,>        ;任务 1 的代码段描

```

述符

```

DSTACK1         DESC  <0FFFFH,,,ATDW,,>        ;任务 1 的堆栈段描述符
DDATAS          DESC  <0FFFFH,,,ATDW,,>        ;源数据段描述符
DDATAO          DESC  <0FFFFH,3000H,,,ATDW,,>   ;目的数据段描述符
TOTASK0         GATE  <0,TSS0_SEL,,ATTASKGATE,> ;指向任务 0 的调用

```

门描述符

```

;-----
DCODE1_SEL      =      DCODE1-LDT1+TIL         ;任务 1 的代码描述符选择子
DSTACK1_SEL     =      DSTACK1-LDT1+TIL        ;任务 1 的堆栈段描述符选择

```

子

```

DDATAS_SEL      =      DDATAS-LDT1+TIL         ;源数据段描述符选择子
DDATAO_SEL      =      DDATAO-LDT1+TIL         ;目的数据段描述符选择子
TOTASK0_SEL     =      TOTASK0-LDT1+TIL;任务门描述符选择子
LDT1LEN         =      $-LDT1
DSEG1           ENDS                          ;局部描述符表段定义结束

```

```

;-----
TSS0SEG      SEGMENT PARA USE16                      ;任务状态段 TSS0
TEMPTASK      TSS      <>
              DB      OFFH                          ;I/O 许可位图结束标志
TSSOLEN       =      $
TSS0SEG      ENDS                                    ;任务状态段 TSS0 结束
;-----
TSS1SEG      SEGMENT PARA USE16                      ;任务状态段 TSS1
;-----
              DD      0                              ;链接字
              DD      DSTACK1LEN                    ;0 级堆栈指针
              DD      DSTACK1_SEL                    ;0 级堆栈选择子
              DD      0                              ;1 级堆栈指针(实例不使用)
              DD      0                              ;1 级堆栈选择子(实例不使用)
              DD      0                              ;2 级堆栈指针
              DD      0                              ;2 级堆栈选择子
              DD      0                              ;CR3
              DD      0                              ;EIP
              DD      200H                          ;EFLAGS
              DD      0FFFH                        ;EAX
              DD      0                              ;ECX
              DD      0                              ;EDX
              DD      0                              ;EBX
              DD      DSTACK1LEN                    ;ESP
              DD      0                              ;EBP
              DD      0                              ;ESI
              DD      0                              ;EDI
              DW      DDATAS_SEL,0                  ;ES
              DW      DCODE1_SEL,0                  ;CS
              DW      DSTACK1_SEL,0                  ;SS
              DW      DDATAS_SEL,0                  ;DS
              DW      DDATAS_SEL,0                  ;FS
              DW      DDATAS_SEL,0                  ;GS
              DW      LDT1_SEL,0                     ;LDTR
              DW      0                              ;调试陷阱标志
              DW      $+2                            ;指向 I/O 许可位图
              DB      OFFH                          ;I/O 许可位图结束标志
TSS1LEN       =      $
;-----

```

```

TSS1SEG      ENDS                                ;任务状态段 TSS1 结束
;-----
DSTACK0SEG    SEGMENT PARA USE16                ;任务 0 的堆栈段
DSTACK0LEN    =      512
               DB      DSTACK0LEN DUP(?)        ;定义 512 个缓冲字节单元
DSTACK0SEG    ENDS                                ;任务 0 堆栈段结束
;-----
DSTACK1SEG    SEGMENT PARA USE16                ;任务 1 的堆栈段
DSTACK1LEN    =      512
               DB      DSTACK1LEN DUP(?)        ;定义 512 个缓冲字节单元
DSTACK1SEG    ENDS                                ;任务 1 堆栈段结束
;-----
DDATA0SEG     SEGMENT PARA USE16                ;任务 0 的数据段
DDATA0LEN     =      128
               DB DDATA0LEN DUP(?)
DDATA0SEG     ENDS
;-----
DDATASSEG     SEGMENT PARA USE16                ;源数据段
DTEST         DB 11H,22H,33H,44H,55H,66H,77H,88H ;定义源数据段数据
DDATASLEN     =      $                          ;数据段的长度
DDATASSEG     ENDS                                ;源数据段定义结束
;-----
CODEMSEG      SEGMENT PARA USE16                ;主程序段
               ASSUME  CS:CODEMSEG
START         PROC
               ASSUME  DS:DSEG
               MOV     AX,DSEG
               MOV     DS,AX

               MOV     BX,16                      ;准备要加载到 GDTR 的伪描述符
               MUL     BX
               ADD     AX,OFFSET GDT              ;计算并设置 GDT 基地址
               ADC     DX,0
               MOV     WORD PTR VGDTR.BASE,AX
               MOV     WORD PTR VGDTR.BASE+2,DX

               MOV     AX,CODEMSEG
               MUL     BX
               MOV     WORD PTR CODEM.BASE,AX     ;计算并设置主程序段基址

```

址

```
MOV    BYTE PTR CODEM.BASEM,DL
MOV    BYTE PTR CODEM.BASEH,DH
```

```
MOV    AX,DSEG1
```

```
MUL    BX
```

```
MOV    WORD PTR LDTABLE1.BASEL,AX ;计算并设置 LDT 表段
```

基地址

```
MOV    BYTE PTR LDTABLE1.BASEM,DL
```

```
MOV    BYTE PTR LDTABLE1.BASEH,DH
```

```
MOV    AX,TSS0SEG
```

```
MUL    BX
```

```
MOV    WORD PTR TSSTABLE0.BASEL,AX ;计算并设置任务段
```

TSS0 基地址

```
MOV    BYTE PTR TSSTABLE0.BASEM,DL
```

```
MOV    BYTE PTR TSSTABLE0.BASEH,DH
```

```
MOV    BYTE PTR TSSTABLE0.ATTRIBUTES,AT386TSS ;任 务
```

段 TSS0 的段属性

```
MOV    AX,TSS1SEG
```

```
MUL    BX
```

```
MOV    WORD PTR TSSTABLE1.BASEL,AX ;计算并设置任务段
```

TSS1 基地址

```
MOV    BYTE PTR TSSTABLE1.BASEM,DL
```

```
MOV    BYTE PTR TSSTABLE1.BASEH,DH
```

```
MOV    BYTE PTR TSSTABLE1.ATTRIBUTES,AT386TSS ;任 务
```

段 TSS1 的段属性

```
MOV    AX,DSTACK0SEG
```

```
MUL    BX
```

```
MOV    WORD PTR DSTACK0.BASEL,AX ;计算并设置堆栈段
```

基地址

```
MOV    BYTE PTR DSTACK0.BASEM,DL
```

```
MOV    BYTE PTR DSTACK0.BASEH,DH
```

```
MOV    AX,DDATA0SEG
```

```
MUL    BX
```

```
MOV    WORD PTR DDATA0.BASEL,AX ;计算并设置任务 0 数据
```



段基地址

```
MOV     BYTE PTR DDATA0.BASEM,DL
MOV     BYTE PTR DDATA0.BASEH,DH
```

;-----

```
ASSUME  DS:DSEG1
```

```
MOV     AX,DSEG1
```

```
MOV     DS,AX
```

```
MOV     AX,DCODE1SEG
```

```
MUL     BX
```

```
MOV     WORD PTR DCODE1.BASEL,AX ;计算并设置任务 1 代码
```

段基地址

```
MOV     BYTE PTR DCODE1.BASEM,DL
```

```
MOV     BYTE PTR DCODE1.BASEH,DH
```

```
MOV     AX,DSTACK1SEG
```

```
MUL     BX
```

```
MOV     WORD PTR DSTACK1.BASEL,AX ;计算并设置任务 1
```

堆栈段基地址

```
MOV     BYTE PTR DSTACK1.BASEM,DL
```

```
MOV     BYTE PTR DSTACK1.BASEH,DH
```

```
MOV     AX,DDATASSEG
```

```
MUL     BX
```

```
MOV     WORD PTR DDATAS.BASEL,AX ;计算并设置源数据段基
```

地址

```
MOV     BYTE PTR DDATAS.BASEM,DL
```

```
MOV     BYTE PTR DDATAS.BASEH,DH
```

;-----

```
ASSUME  DS:DSEG
```

```
MOV     AX,DSEG
```

```
MOV     DS,AX
```

```
MOV     SSVAR,SS
```

```
MOV     SPVAR,SP ;保存实模式下堆栈指针
```

```
LGDT    QWORD PTR VGDTR ;装载 GDT 表寄存器 GDTR
```

```

        CLI                                ;关中断

        MOV     EAX,CR0                    ;切换到保护方式
        OR      EAX,1
        MOV     CR0,EAX

        JUMP16  <CODEM_SEL>,<OFFSET VIRTUAL>    ;清指令预取队列,
并真正进入保护方式

VIRTUAL:                                ;现在开始在保护方式下运行
        MOV     AX,TSS0_SEL                ;装入任务 0
        LTR     AX

        MOV     AX,DDATA0_SEL              ;置数据段
        MOV     DS,AX
        MOV     ES,AX
        MOV     FS,AX
        MOV     GS,AX
        MOV     ESP,DSTACK0LEN-1
        MOV     AX,DSTACK0_SEL             ;置堆栈
        MOV     SS,AX

        JUMP16  TSS1_SEL,0                ;跳转到任务 1 的 0 级代码段

        MOV     AX,NORMAL_SEL              ;把规范段描述符装入各数据
段寄存器
        MOV     DS,AX
        MOV     ES,AX
        MOV     FS,AX
        MOV     GS,AX
        MOV     SS,AX

        MOV     EAX,CR0                    ;切换到实模式
        AND     AL,11111110B
        MOV     CR0,EAX
        JUMP16  <SEG REAL>,<OFFSET REAL>;清指令预取队列,进入实方式

REAL:                                ;现在又回到实方式

```

```

MOV     AX,DSEG
MOV     DS,AX
LSS     SP,DWORD PTRSPVAR ;恢复实模式下的堆栈

```

```

ASSUME  DS:TSS1SEG
MOV     AX,TSS1SEG
MOV     DS,AX

```

```

CALL    INIT_TSS1           ;恢复 TSS1 任务段表的初值,以备下次
继续运行

```

```

STI                      ;开中断

```

```

MOV     AX,4C00H
INT     21H               ;程序终止

```

```

START    ENDP
CODEMLN  = $

```

```

;-----

```

```

INIT_TSS1 PROC NEAR ;恢复 TSS1 任务段表的初值

```

```

    PUSH DS
    MOV     AX,TSS1SEG
    MOV     DS,AX

```

```

    MOV     SI,0 ;链接字
    MOV     AX,0
    MOV     [SI],AX

```

```

    ADD     SI,4 ;0 级堆栈指针
    MOV     AX,DSTACK1LEN
    MOV     [SI],AX

```

```

    ADD     SI,4 ;0 级堆栈选择子
    MOV     AX,DSTACK1_SEL
    MOV     [SI],AX

```

```

    ADD     SI,4 ;1 级堆栈指针(实例不使用)
    MOV     AX,0

```

```
MOV      [SI],AX

ADD      SI,4      ;1 级堆栈选择子(实例不使用)
MOV      AX,0
MOV      [SI],AX

ADD      SI,4      ;2 级堆栈指针
MOV      AX,0
MOV      [SI],AX

ADD      SI,4      ;2 级堆栈选择子
MOV      AX,0
MOV      [SI],AX

ADD      SI,4      ;CR3
MOV      AX,0
MOV      [SI],AX

ADD      SI,4      ;EIP
MOV      AX,0
MOV      [SI],AX

ADD      SI,4      ;EFLAGS
MOV      AX,200H
MOV      [SI],AX

ADD      SI,4      ;EAX
MOV      AX,0FFFH
MOV      [SI],AX

ADD      SI,4      ;ECX
MOV      AX,0
MOV      [SI],AX

ADD      SI,4      ;EDX
MOV      AX,0
MOV      [SI],AX

ADD      SI,4      ;EBX
```

```
MOV     AX,0
MOV     [SI],AX

ADD     SI,4    ;ESP
MOV     AX,DSTACK1LEN
MOV     [SI],AX

ADD     SI,4    ;EBP
MOV     AX,0
MOV     [SI],AX

ADD     SI,4    ;ESI
MOV     AX,0
MOV     [SI],AX

ADD     SI,4    ;EDI
MOV     AX,0
MOV     [SI],AX

ADD     SI,4    ;ES
MOV     AX,DDATAS_SEL
MOV     [SI],AX

ADD     SI,2
MOV     AX,0
MOV     [SI],AX

ADD     SI,2    ;CS
MOV     AX,DCODE1_SEL
MOV     [SI],AX

ADD     SI,2
MOV     AX,0
MOV     [SI],AX

ADD     SI,2    ;SS
MOV     AX,DSTACK1_SEL
MOV     [SI],AX
```

```
ADD     SI,2
MOV     AX,0
MOV     [SI],AX

ADD     SI,2    ;DS
MOV     AX,DDATAS_SEL
MOV     [SI],AX

ADD     SI,2
MOV     AX,0
MOV     [SI],AX

ADD     SI,2    ;FS
MOV     AX,DDATAS_SEL
MOV     [SI],AX

ADD     SI,2
MOV     AX,0
MOV     [SI],AX

ADD     SI,2    ;GS
MOV     AX,DDATAS_SEL
MOV     [SI],AX

ADD     SI,2
MOV     AX,0
MOV     [SI],AX

ADD     SI,2    ;LDTR
MOV     AX,LDT1_SEL
MOV     [SI],AX

ADD     SI,2
MOV     AX,0
MOV     [SI],AX

ADD     SI,2    ;调试陷阱标志
MOV     AX,0
MOV     [SI],AX
```

```

        ADD     SI,2      ;指向 I/O 许可位图
        MOV     AX,68H    ;TSS 有 104 个字节
        MOV     [SI],AX

        ADD     SI,2      ;I/O 许可位图结束标志
        MOV     AX,0FFH
        MOV     [SI],AX

        POP     DS

        RET

INIT_TSS1    ENDP

CODEMSEG    ENDS

;-----
DCODE1SEG    SEGMENT PARA USE16                ;任务 1 代码段
        ASSUME  CS:DCODE1SEG
        MOV     AX, DDATAS_SEL
        MOV     DS,AX                        ;加载源数据段描述符
        MOV     AX, DDATAO_SEL
        MOV     ES,AX                        ;加载目标数据段描述符
        XOR     SI,SI
        XOR     DI,DI                        ;设置指针初值
        MOV     CX, 8                        ;设置传送长度
M1:         MOVSB                            ;传送
        LOOP    M1

        JUMP16   TOTASK0_SEL,0                ;切换回任务 0
DCODE1LEN    = $
DCODE1SEG    ENDS
;=====

        END     START

```

### 6.3.4 使用 CALL 指令实现任务实验（有特权级）

#### 1. 实验内容

本实验要求使用 CALL 指令实现任务切换。实验安排了三个有特权级的任务段分别是：任务 1（特权级为 0 级）、任务 2（特权级为 0 级）、任务 3（特权级为 0 级）。

实验由主程序进入，装入任务 1，再跳到任务 2，由任务 2 再跳到任务 3 完成一段数据的传输。当任务 3 的工作完成后，再由任务 3 返回到任务 2，再由任务 2 返回到任务 1 到主程序。实验中涉及了 3 个任务，所以需要为这三个任务分别建立 3 个 TSS，并且为每个任务建立自己的代码段，数据段以及堆栈段。

在主程序段中应首先使用 LTR 指令将特权级为 0 级的任务 1 TSS 装入 TR 寄存器。在任务 1 内部执行过程中，使用 CALL16 宏直接指向任务 2 的 TSS 表进行任务切换。

在任务 2 内部执行过程中，使用 CALL16 宏通过任务门转移到任务 3，即通过任务门从一个 0 级代码段转移到了一个 3 级代码段。在转移过程中，处理器首先将各个寄存器的内容保存到任务 2 的任务状态段，再将任务 3 的任务状态段内容装入 CPU 的各寄存器。任务 3 中的 CS 指向的是一个 3 级程序段，则相应使用的 SS 也是 3 级。于是在任务切换的过程中也实现了代码段间有特权级的切换。

切换到任务 3 完成数据传输后，再通过 IRETD 段间返回指令，使得从任务 3 返回到任务 2 再返回到任务 1 直到主程序。

本实验可实现将一个数据段中的数据搬移到另一个数据段所指定的地址空间中（指定地址给定为 00003000H）。源数据段 DTEST 值为 11H,22H,33H,44H,55H,66H,77H,88H，在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

## 2. 实验步骤

- (1) 编写实验程序（例程文件名为：6-3-2.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序，待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入：

D0000: 3000

然后回车，即可看到实验运行结果，将数据传输到了给定的地址空间中。

### 实验程序清单

```

;-----
;文件名:6-3-2.ASM
;功能:使用 CALL 指令实现任务切换实验（反映从一个任务的 0 级程序段切换到另一个任务的 3 级程序段的状况）
;-----
INCLUDE          386SCD.INC
;-----
GDTSEG          SEGMENT PARA USE16                      ;全局描述符表数据段(16 位)
;-----

```



GDT	LABEL	BYTE	;全局描述符表
DUMMY	DESC	<>	;空描述符
NORMAL	DESC	<0FFFFH,,,ATDW,,>	;规范段描述符
CODEM	DESC	<0FFFFH,CODEMSEG,,ATCE,,>	;主程序段描述符
TSSTABLE1	DESC	<0FFFFH,TSS1SEG,,,,>	;任务 1 状态段 TSS 描述符
TSSTABLE2	DESC	<0FFFFH,TSS2SEG,,,,>	;任务 2 状态段 TSS 描述符
TSSTABLE3	DESC	<0FFFFH,TSS3SEG,,,,>	;任务 3 状态段 TSS 描述符
DCODE2	DESC	<0FFFFH,DCODE2SEG,,ATCE,,>	;0 级代码段 2 描述符
DCODE3	DESC	<0FFFFH,DCODE3SEG,,ATCE+DPL3,,>	;3 级代码段 3 描述符(DPL=3)
DSTACK1	DESC	<0FFFFH,DSTACK1SEG,,ATDW,,>	;任务 1 的 0 级堆栈段描述符
DSTACK2	DESC	<0FFFFH,DSTACK2SEG,,ATDW,,>	;任务 2 的 0 级堆栈段描述符
DSTACK3	DESC	<0FFFFH,DSTACK3SEG,,ATDW+DPL3,,>	;任务 3 的 3 级堆栈段描述符(DPL=3)
DDATA1	DESC	<0FFFFH,DDATA1SEG,,ATDW,,>	;任务 1 的数据段描述符
DDATA2	DESC	<0FFFFH,DDATA2SEG,,ATDW,,>	;任务 2 的数据段描述符
DDATAS	DESC	<0FFFFH,DDATASSEG,,ATDW+DPL3,,>	;3 级源数据段描述符(DPL=3)
DDATAO	DESC	<0FFFFH,3000H,,ATDW+DPL3,,>	;3 级目的数据段描述符(DPL=3)
TOTASK3	GATE	<0,TSS3_SEL,,ATTASKGATE,>	;指向任务 0 的调用门描述符
;-----			
NORMAL_SEL	=	NORMAL-GDT	;规范段描述符选择子

```

CODEM_SEL      =      CODEM-GDT              ;主程序段选择子
TSS1_SEL       =      TSSTABLE1-GDT          ;任务 1 状态段选择子
TSS2_SEL       =      TSSTABLE2-GDT          ;任务 2 状态段选择子
TSS3_SEL       =      TSSTABLE3-GDT          ;任务 3 状态段选择子
DCODE2_SEL     =      DCODE2-GDT              ;0 级代码段 2 描述符选择
子
DCODE3_SEL     =      DCODE3-GDT+RPL3        ;3 级代码段 3 描述符
选择子
DSTACK1_SEL    =      DSTACK1-GDT            ;0 级堆栈段 1 描述符选择
子
DSTACK2_SEL    =      DSTACK2-GDT            ;0 级堆栈段 2 描述符选择
子
DSTACK3_SEL    =      DSTACK3-GDT+RPL3        ;3 级堆栈段 3 描述符选择
子(RPL=3)
DDATA1_SEL     =      DDATA1-GDT              ;任务 1 的数据段描述符
对应选择子
DDATA2_SEL     =      DDATA2-GDT              ;任务 2 的数据段描述符
对应选择子
DDATAS_SEL     =      DDATAS-GDT+RPL3        ;源数据段选择子
(RPL=3)
DDATAO_SEL     =      DDATAO-GDT+RPL3        ;目的数据段选择子
(RPL=3)
TOTASK3_SEL    =      TOTASK3-GDT              ;任务门描述符选择子
GDTLEN         =      $-GDT                  ;全局描述符表长度
VGDTTR         PDESC    <GDTLEN-1,>          ;定义 GDT 的伪描述符
VGDTTR
SPVAR          DW      ?
SSVAR          DW      ?                    ;保存实模式下堆栈指针
GDTSEG         ENDS                        ;全局描述符表段定义结束
;-----
TSS1SEG        SEGMENT PARA USE16            ;任务 1 状态段 TSS
TEMPTASK       TSS      <>
               DB      OFFH                  ;I/O 许可位图结束标志
TSS1LEN        =      $
TSS1SEG        ENDS                        ;任务 1 状态段 TSS 结束
;-----
TSS2SEG        SEGMENT PARA USE16            ;任务 2 状态段 TSS
               DD      0                      ;链接字
               DD      DSTACK2LEN            ;0 级堆栈指针

```

```

        DD    DSTACK2_SEL        ;0 级堆栈选择子
        DD    0                   ;1 级堆栈指针(实例不使用)
        DD    0                   ;1 级堆栈选择子(实例不使用)
        DD    0                   ;2 级堆栈指针
        DD    0                   ;2 级堆栈选择子
        DD    0                   ;CR3
        DD    0                   ;EIP
        DD    200H                ;EFLAGS
        DD    0FFFH               ;EAX
        DD    0                   ;ECX
        DD    0                   ;EDX
        DD    0                   ;EBX
        DD    DSTACK2LEN          ;ESP
        DD    0                   ;EBP
        DD    0                   ;ESI
        DD    0                   ;EDI
        DD    DDATA2_SEL          ;ES
        DD    DCODE2_SEL          ;CS
        DD    DSTACK2_SEL         ;SS
        DD    DDATA2_SEL          ;DS
        DD    DDATA2_SEL          ;FS
        DD    DDATA2_SEL          ;GS
        DD    0                   ;LDTR
        DW    0                   ;调试陷阱标志
        DW    $+2                 ;指向 I/O 许可位图
        DB    0FFH               ;I/O 许可位图结束标志

TSS2LEN =    $
TSS2SEG ENDS                    ;任务 2 状态段 TSS 结束
;-----
TSS3SEG    SEGMENT PARA USE16    ;任务 3 状态段 TSS
        DD    0                   ;链接字
        DD    0                   ;0 级堆栈指针
        DD    0                   ;0 级堆栈选择子
        DD    0                   ;1 级堆栈指针(实例不使用)
        DD    0                   ;1 级堆栈选择子(实例不使用)
        DD    0                   ;2 级堆栈指针
        DD    0                   ;2 级堆栈选择子
        DD    0                   ;CR3
        DD    0                   ;EIP

```

```

        DD      200H                      ;EFLAGS
        DD      0FFFH                     ;EAX
        DD      0                         ;ECX
        DD      0                         ;EDX
        DD      0                         ;EBX
        DD      DSTACK3LEN                ;ESP
        DD      0                         ;EBP
        DD      0                         ;ESI
        DD      0                         ;EDI
        DD      DDATAO_SEL                 ;ES
        DD      DCODE3_SEL                 ;CS
        DD      DSTACK3_SEL                ;SS
        DD      DDATAS_SEL                 ;DS
        DD      DDATAS_SEL                 ;FS
        DD      DDATAS_SEL                 ;GS
        DD      0                         ;LDTR
        DW      0                         ;调试陷阱标志
        DW      $+2                       ;指向 I/O 许可位图
        DB      OFFH                      ;I/O 许可位图结束标志

TSS3LEN  =      $
TSS3SEG  ENDS                             ;任务 3 状态段 TSS 结束
;-----
DSTACK1SEG  SEGMENT PARA USE16            ;任务 1 堆栈段
DSTACK1LEN  =      512
        DB      DSTACK1LEN DUP(?)        ;定义 512 个缓冲字节单元
DSTACK1SEG  ENDS                          ;任务 1 堆栈段结束
;-----
DSTACK2SEG  SEGMENT PARA USE16            ;任务 2 堆栈段
DSTACK2LEN  =      512
        DB      DSTACK2LEN DUP(?)
DSTACK2SEG  ENDS
;-----
DSTACK3SEG  SEGMENT PARA USE16            ;任务 3 堆栈段
DSTACK3LEN  =      512
        DB      DSTACK3LEN DUP(?)
DSTACK3SEG  ENDS
;-----
DDATA1SEG  SEGMENT PARA USE16            ;任务 1 的数据段
DDATA1LEN  =      128

```

```

        DB DDATA1LEN DUP(?)
DDATA1SEG  ENDS
;-----
DDATA2SEG  SEGMENT PARA USE16      ;任务 2 的数据段
DDATA2LEN  = 128
        DB DDATA2LEN DUP(?)
DDATA2SEG  ENDS
;-----
DDATASSEG  SEGMENT PARA USE16      ;源数据段
DTEST      DB 11H,22H,33H,44H,55H,66H,77H,88H ;定义源数据段数据
DDATASLEN  = $                      ;数据段的长度
DDATASSEG  ENDS                    ;源数据段定义结束
;-----
CODEMSEG   SEGMENT PARA USE16      ;主程序段
        ASSUME  CS:CODEMSEG
START      PROC
        ASSUME  DS:GDTSEG
        MOV     AX,GDTSEG
        MOV     DS,AX

        MOV     BX,16              ;准备要加载到 GDTR 的伪描述符
        MUL     BX
        ADD     AX,OFFSET GDT      ;计算并设置 GDT 基地址
        ADC     DX,0
        MOV     WORD PTR VGDTR.BASE,AX
        MOV     WORD PTR VGDTR.BASE+2,DX

        MOV     AX,CODEMSEG
        MUL     BX
        MOV     WORD PTR CODEM.BASEL,AX ;计算并设置主程序段基址

        MOV     BYTE PTR CODEM.BASEM,DL
        MOV     BYTE PTR CODEM.BASEH,DH

        MOV     AX,TSS1SEG
        MUL     BX
        MOV     WORD PTR TSSTABLE1.BASEL,AX ;计算并设置任务段 1 基址
        MOV     BYTE PTR TSSTABLE1.BASEM,DL

```

	MOV	BYTE PTR TSSTABLE1.BASEH,DH
	MOV	BYTE PTR TSSTABLE1.ATTRIBUTES,AT386TSS ; 任 务
段 TSS1 的段属性		
	MOV	AX,TSS2SEG
	MUL	BX
	MOV	WORD PTR TSSTABLE2.BASEL,AX ;计算并设置任务段 2
基地址		
	MOV	BYTE PTR TSSTABLE2.BASEM,DL
	MOV	BYTE PTR TSSTABLE2.BASEH,DH
	MOV	BYTE PTR TSSTABLE2.ATTRIBUTES,AT386TSS ; 任 务
段 TSS2 的段属性		
	MOV	AX,TSS3SEG
	MUL	BX
	MOV	WORD PTR TSSTABLE3.BASEL,AX ;计算并设置任务段 3
基地址		
	MOV	BYTE PTR TSSTABLE3.BASEM,DL
	MOV	BYTE PTR TSSTABLE3.BASEH,DH
	MOV	BYTE PTR TSSTABLE3.ATTRIBUTES,AT386TSS ; 任 务
段 TSS3 的段属性		
	MOV	AX,DCODE2SEG
	MUL	BX
	MOV	WORD PTR DCODE2.BASEL,AX ;计算并设置代码段 2 基地址
址		
	MOV	BYTE PTR DCODE2.BASEM,DL
	MOV	BYTE PTR DCODE2.BASEH,DH
	MOV	AX,DCODE3SEG
	MUL	BX
	MOV	WORD PTR DCODE3.BASEL,AX ;计算并设置代码段 3 基地址
址		
	MOV	BYTE PTR DCODE3.BASEM,DL
	MOV	BYTE PTR DCODE3.BASEH,DH
	MOV	AX,DSTACK1SEG
	MUL	BX
	MOV	WORD PTR DSTACK1.BASEL,AX ;计算并设置堆栈段 1 基地址

地址

```
MOV    BYTE PTR DSTACK1.BASEM,DL
MOV    BYTE PTR DSTACK1.BASEH,DH
```

```
MOV    AX,DSTACK2SEG
MUL    BX
MOV    WORD PTR DSTACK2.BASEL,AX    ;计算并设置堆栈段 2 基
```

地址

```
MOV    BYTE PTR DSTACK2.BASEM,DL
MOV    BYTE PTR DSTACK2.BASEH,DH
```

```
MOV    AX,DSTACK3SEG
MUL    BX
MOV    WORD PTR DSTACK3.BASEL,AX    ;计算并设置堆栈段 3 基
```

地址

```
MOV    BYTE PTR DSTACK3.BASEM,DL
MOV    BYTE PTR DSTACK3.BASEH,DH
```

```
MOV    AX,DDATA1SEG
MUL    BX
MOV    WORD PTR DDATA1.BASEL,AX    ;计算并设置数据段 1 基地
```

址

```
MOV    BYTE PTR DDATA1.BASEM,DL
MOV    BYTE PTR DDATA1.BASEH,DH
```

```
MOV    AX,DDATA2SEG
MUL    BX
MOV    WORD PTR DDATA2.BASEL,AX    ;计算并设置数据段 2 基地
```

址

```
MOV    BYTE PTR DDATA2.BASEM,DL
MOV    BYTE PTR DDATA2.BASEH,DH
```

```
MOV    AX,DDATASSEG
MUL    BX
MOV    WORD PTR DDATAS.BASEL,AX    ;计算并设置源数据段基地
```

址

```
MOV    BYTE PTR DDATAS.BASEM,DL
MOV    BYTE PTR DDATAS.BASEH,DH
```

;-----

```

        ASSUME  DS:GDTSEG
        MOV     AX,GDTSEG
        MOV     DS,AX

        MOV     SSVAR,SS
        MOV     SPVAR,SP                ;保存实模式下的堆栈指针

        LGDT    QWORD PTR VGDTR        ;装载 GDT 表寄存器 GDTR
        CLI                                ;关中断
        MOV     EAX,CR0                ;切换到保护方式
        OR      EAX,1
        MOV     CR0,EAX
        JUMP16  <CODEM_SEL>,<OFFSET_VIRTUAL>;清指令预取队列,并真正进入保护方式

VIRTUAL:    ;现在开始在保护方式下运行
        MOV     AX,TSS1_SEL            ;装入任务 1
        LTR     AX
        MOV     AX,DDATA1_SEL          ;置数据段 1
        MOV     DS,AX
        MOV     ES,AX
        MOV     FS,AX
        MOV     GS,AX
        MOV     ESP,DSTACK1LEN-1
        MOV     AX,DSTACK1_SEL         ;置堆栈段 1
        MOV     SS,AX

        CALL16  TSS2_SEL,0             ;跳转到任务 2 的 0 级代码段

        MOV     AX,NORMAL_SEL           ;把规范段描述符装入各
数据段寄存器
        MOV     DS,AX
        MOV     ES,AX
        MOV     FS,AX
        MOV     GS,AX
        MOV     SS,AX

        MOV     EAX,CR0                ;切换到实模式
        AND     AL,11111110B

```



MOV CR0,EAX  
JUMP16 <SEG REAL>,<OFFSET REAL> ;清指令预取队列,进入实方式

REAL: ;现在又回到实方式

MOV AX,GDTSEG  
MOV DS,AX  
LSS SP,DWORD PTRSPVAR ;恢复实模式下的堆栈

ASSUME DS:TSS2SEG  
MOV AX,TSS2SEG  
MOV DS,AX  
CALL INIT\_TSS2 ;恢复 TSS2 任务段表的初值,以备下次  
继续运行

ASSUME DS:TSS3SEG  
MOV AX,TSS3SEG  
MOV DS,AX  
CALL INIT\_TSS3 ;恢复 TSS3 任务段表的初值,以备下次  
继续运行

STI ;开中断

MOV AX,4C00H  
INT 21H ;程序终止

START ENDP  
CODEMLN = \$

;-----

INIT\_TSS2 PROC NEAR ;恢复 TSS2 任务表的初值

PUSH DS  
MOV AX,TSS2SEG  
MOV DS,AX

MOV SI,0 ;链接字  
MOV AX,0  
MOV [SI],AX

```
ADD     SI,4      ;0 级堆栈指针
MOV     AX,DSTACK2LEN
MOV     [SI],AX

ADD     SI,4      ;0 级堆栈选择子
MOV     AX,DSTACK2_SEL
MOV     [SI],AX

ADD     SI,4      ;1 级堆栈指针
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;1 级堆栈选择子
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;2 级堆栈指针
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;2 级堆栈选择子
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;CR3
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;EIP
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;EFLAGS
MOV     AX,200H
MOV     [SI],AX

ADD     SI,4      ;EAX
MOV     AX,0FFFH
MOV     [SI],AX
```

```
ADD     SI,4    ;ECX
MOV     AX,0
MOV     [SI],AX

ADD     SI,4    ;EDX
MOV     AX,0
MOV     [SI],AX

ADD     SI,4    ;EBX
MOV     AX,0
MOV     [SI],AX

ADD     SI,4    ;ESP
MOV     AX,DSTACK2LEN
MOV     [SI],AX

ADD     SI,4    ;EBP
MOV     AX,0
MOV     [SI],AX

ADD     SI,4    ;ESI
MOV     AX,0
MOV     [SI],AX

ADD     SI,4    ;EDI
MOV     AX,0
MOV     [SI],AX

ADD     SI,4    ;ES
MOV     AX,DDATA2_SEL
MOV     [SI],AX

ADD     SI,4    ;CS
MOV     AX,DSTACK2_SEL
MOV     [SI],AX

ADD     SI,4    ;SS
MOV     AX,DSTACK2_SEL
```

```

        MOV     [SI],AX

        ADD     SI,4      ;DS
        MOV     AX,DDATA2_SEL
        MOV     [SI],AX

        ADD     SI,4      ;FS
        MOV     AX,DDATA2_SEL
        MOV     [SI],AX

        ADD     SI,4      ;GS
        MOV     AX,DDATA2_SEL
        MOV     [SI],AX

        ADD     SI,4      ;LDTR
        MOV     AX,0
        MOV     [SI],AX

        ADD     SI,4      ;调试陷阱标志
        MOV     AX,0
        MOV     [SI],AX

        ADD     SI,2      ;指向 I/O 许可位图
        MOV     AX,68H    ;TSS 有 104 个字节
        MOV     [SI],AX

        ADD     SI,2      ;I/O 许可位图结束标志
        MOV     AX,0FFH
        MOV     [SI],AX

        POP     DS

        RET

INIT_TSS2      ENDP
;-----
INIT_TSS3 PROC NEAR ;恢复 TSS3 任务表的初值
        PUSH   DS
        MOV     AX,TSS3SEG
        MOV     DS,AX

```

```
MOV     SI,0      ;链接字
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;0 级堆栈指针
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;0 级堆栈选择子
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;1 级堆栈指针(实例不使用)
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;1 级堆栈选择子(实例不使用)
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;2 级堆栈指针
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;2 级堆栈选择子
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;CR3
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;EIP
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;EFLAGS
MOV     AX,200H
```

```
MOV     [SI],AX

ADD     SI,4     ;EAX
MOV     AX,0FFFH
MOV     [SI],AX

ADD     SI,4     ;ECX
MOV     AX,0
MOV     [SI],AX

ADD     SI,4     ;EDX
MOV     AX,0
MOV     [SI],AX

ADD     SI,4     ;EBX
MOV     AX,0
MOV     [SI],AX

ADD     SI,4     ;ESP
MOV     AX,DSTACK3LEN
MOV     [SI],AX

ADD     SI,4     ;EBP
MOV     AX,0
MOV     [SI],AX

ADD     SI,4     ;ESI
MOV     AX,0
MOV     [SI],AX

ADD     SI,4     ;EDI
MOV     AX,0
MOV     [SI],AX

ADD     SI,4     ;ES
MOV     AX,DDATAO_SEL
MOV     [SI],AX

ADD     SI,4     ;CS
```

```
MOV     AX,DCODE3_SEL
MOV     [SI],AX

ADD     SI,4      ;SS
MOV     AX,DSTACK3_SEL
MOV     [SI],AX

ADD     SI,4      ;DS
MOV     AX,DDATAS_SEL
MOV     [SI],AX

ADD     SI,4      ;FS
MOV     AX,DDATAS_SEL
MOV     [SI],AX

ADD     SI,4      ;GS
MOV     AX,DDATAS_SEL
MOV     [SI],AX

ADD     SI,4      ;LDTR
MOV     AX,0
MOV     [SI],AX

ADD     SI,4      ;调试陷阱标志
MOV     AX,0
MOV     [SI],AX

ADD     SI,2      ;指向 I/O 许可位图
MOV     AX,68H    ;TSS 有 104 个字节
MOV     [SI],AX

ADD     SI,2      ;I/O 许可位图结束标志
MOV     AX,0FFH
MOV     [SI],AX

POP     DS

RET

INIT_TSS3      ENDP
```

```

;-----
DCODE2SEG  SEGMENT PARA USE16                      ;任务 2 的 0 级代码段
            ASSUME  CS:DCODE2SEG

            CALL16  TOTASK3_SEL,0                  ;调用任务门描述符到任务 3

            IRETD                                   ;返回指令

DCODE2LEN   = $
DCODE2SEG   ENDS

;-----
DCODE3SEG   SEGMENT PARA USE16                      ;任务 3 的 3 级代码段
            ASSUME  CS:DCODE3SEG

            MOV     AX, DDATAS_SEL
            MOV     DS,AX                          ;加载源数据段描述符
            MOV     AX, DDATAO_SEL
            MOV     ES,AX                          ;加载目标数据段描述符
            XOR     SI,SI
            XOR     DI,DI                          ;设置指针初值
            MOV     CX, 8                          ;设置传送长度
M1:         MOVSB                                   ;传送
            LOOP    M1

            IRETD                                   ;段间返回

DCODE3LEN   = $

DCODE3SEG   ENDS

;-----
CODEMSEG    ENDS
;=====
            END      START

```



## 6.4 中断与异常处理实验

### 6.4.1 实验目的

1. 掌握编写保护模式下中断/异常处理程序的方法。
2. 掌握通过 IDT 表实现中断/异常处理的方法。
3. 掌握通过任务门、中断门、陷阱门实现中断/异常处理的方法。
4. 了解通过 INT 及 IRETD 实现任务内无/有特权级变换以及任务切换的过程。

### 6.4.2 实验设备

PC 机一台, TD-PITE 实验装置一套。

### 6.4.3 用中断门、陷阱门实现中断/异常处理实验

80X86 把中断分为外部中断和内部中断两大类。把外部中断称为“中断”, 把内部中断称为“异常”。

#### 1. 预备知识

##### (1) 中断描述表 IDT

与 8086/8088 一样, 在响应中断或者处理异常时, 80X86 根据中断向量号转向对应的处理程序。但是, 在保护模式下 80X86 不再使用实模式下的中断向量表, 而是使用中断描述符表 IDT。在保护模式下, 80X86 把中断向量号作为中断描述符表 IDT 中描述符的索引, 而不再是中断向量表中的中断向量的索引。

像全局描述符表 GDT 一样, 在整个系统中, 中断描述符表 IDT 只有一个。中断描述符表寄存器 IDTR 指示 IDT 在内存中的位置。由于 80X86 只识别 256 个中断向量号, 所以 IDT 最大长度是 2K。

中断描述符表 IDT 所含的描述符只能是中断门、陷阱门和任务门。

##### (2) 通过中断门、陷阱门的转移

如果中断向量号所指示的门描述符是 386 中断门或 386 陷阱门, 那么控制转移到当前任务的一个处理程序过程, 并且可以变换特权级。与通过调用门的 CALL 指令一样, 从中断门或陷阱门中获取指向处理程序的 48 位全指针。其中, 16 位选择子是对应处理程序代码段的选择子, 它指示 GDT 或 LDT 中的描述符; 32 位偏移指示处理程序入口点在代码段内的偏移。

通过中断门、陷阱门的转移, 由处理器硬件自动进行。

LIDT    QWORD    PTRVIDTR    表示装载中断描述符表寄存器 IDTR。

## 2. 实验内容

本实验要求通过 INT 20H 调用 20H 号中断/异常处理程序实现数据传输，其中调用 INT 指令的代码段为 0 级代码段，IDT 中 20H 号门描述符门为中断门。

下面定义一个 IDTSEG 表如下：

```

IDTSEG    SEGMENT      PARA    USE16
IDT        LABEL      BYTE
            REPT        32          ; 从 00H---1FH 的 32 个中断门描述符
            GATE        <,,,AT386IGate,>
            ENDM
INT20      Gate        <0,ICode_Sel,,AT386IGate,>
            ; 对应 20H 号异常/中断处理程序段的中断门描述符
            REPT        256-33      ; 从 21H---FFH 的 223 个中断门描述符
            GATE        <,,,AT386IGate,>
            ENDM
IDTLEN     =            $-IDT
IDTSEG     ENDS

```

本实验可实现将一个数据段中的数据搬移到另一个数据段所指定的地址空间中（指定地址给定为 00003000H）。源数据段 DTEST 值为 11H,22H,33H,44H,55H,66H,77H,88H，在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

## 3. 实验步骤

- (1) 编写实验程序（例程文件名为：6-4-1.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序，待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入：

D0000: 3000

然后回车，即可看到实验运行结果，将数据传输到了给定的地址空间中。

### 实验程序清单

```

;-----
;文件名:6-4-1.ASM
;功能:通过中断门或陷阱门转移的中断/异常处理
;-----
INCLUDE          386SCD.INC
;-----
GDTSEG           SEGMENT PARA USE16                                ;全局描述符表数据

```

段(16 位)

```

;-----
GDT          LABEL  BYTE          ;全局描述符表
DUMMY        DESC   <>            ;空描述符
NORMAL       DESC   <0FFFFH,,,ATDW,,> ;规范段描述符
CODEM        DESC   <0FFFFH,CODEMSEG,,ATCE,,> ;主程序段
描述符
DCODE        DESC   <0FFFFH,DCODESEG,,ATCE,,> ;DCODESEG
代码段描述符
ICODE        DESC   <0FFFFH,ICODESEG,,ATCE,,> ;异常/中断处理
代码段描述符
DSTACK0      DESC   <0FFFFH,DSTACK0SEG,,ATDW,,> ;0 级堆栈
段描述符
DDATAS       DESC   <0FFFFH,DDATASSEG,,ATDW,,> ;源数据段
描述符
DDATAO       DESC   <0FFFFH,3000H,,ATDW,,> ;目的数据段描
述符
;-----

```

```

;-----
NORMAL_SEL   =          NORMAL-GDT          ;规范段描述符选择
子
CODEM_SEL    =          CODEM-GDT           ;主程序段选择子
DDCODE_SEL   =          DCODE-GDT           ;DCODESEG 代码
段选择子
ICODE_SEL    =          ICODE-GDT           ;异常/中断处理代码段
选择子
DSTACK0_SEL  =          DSTACK0-GDT          ;0 级堆栈段选择子
DDATAS_SEL   =          DDATAS-GDT          ;源数据段选择子
DDATAO_SEL   =          DDATAO-GDT          ;目的数据段选择子
GDTLEN       =          $-GDT               ;全局描述符表长度
GDTSEG       ENDS                          ;全局描述符表段定义结
束
;-----

```

```

;-----
IDTSEG       SEGMENT      PARA  USE16          ;16 位中断/异常处理程序段
IDT          LABEL  BYTE          ;定义中断描述符表 IDT
            REPT  32              ;从 00H---1FH 的 32 个空中断门
描述符
            GATE  <,,,AT386IGATE,>
            ENDM

```

INT20        GATE     <0,ICODE\_SEL,,AT386IGATE,>     ;对应 20H 号异常/中断  
处理程序段的中断门描述符

描述符                REPT    256-33                                ;从 21H---FFH 的 223 个空中断门

                      GATE   <,,,AT386IGATE,>  
                      ENDM

IDTLEN        =    \$-IDT                                ;中断描述符表长度  
IDTSEG        ENDS                                        ;16 位中断/异常处理程序段结束

;-----

段)        DSTACK0SEG        SEGMENT PARA    USE16                                ;0 级堆栈段(16 位

                      DSTACK0LEN    =        1024  
                                      DB        DSTACK0LEN DUP(0)                ;定义 1024 个缓冲字节单

元

DSTACK0SEG    ENDS                                        ;0 级堆栈段结束

;-----

DDATASSEG    SEGMENT    PARA USE16                                ;源数据段  
DTEST        DB 11H,22H,33H,44H,55H,66H,77H,88H     ;定义源数据段数据  
DDATASLEN    =    \$                                        ;数据段的长度  
DDATASSEG    ENDS                                        ;源数据段定义结束

;=====

=====

RDATASEG        SEGMENT        PARA    USE16                                ;实模式下的数据段  
VGDT            PDESC    <GDTLEN-1,>                                ;GDT 伪描述符  
VIDTR           PDESC <IDTLEN-1,>                                ;IDT 伪描述符  
NORVIDTR       PDESC <3FFH,0>                                ;用于保存原 IDTR 值  
SPVAR           DW    ?  
SSVAR           DW    ?                                        ;保存实模式下的堆栈指针  
REGV            DB    ?                                        ;用于保存原中断屏蔽寄存器值  
RDATASEG        ENDS

;-----

CODEMSEG        SEGMENT PARA USE16  
                      ASSUME    CS:CODEMSEG  
START            PROC  
                      ASSUME    DS:GDTSEG  
                      MOV        AX,GDTSEG  
                      MOV        DS,AX

```

MOV     BX,16
MOV     AX,CODEMSEG
MOV     BX
MOV     WORD PTR CODEM.BASEL,AX ;计算并设置主程序段基地址
址

MOV     BYTE PTR CODEM.BASEM,DL
MOV     BYTE PTR CODEM.BASEH,DH

MOV     AX,DCODESEG
MOV     BX
MOV     WORD PTR DCODE.BASEL,AX ;计算并设置 DCODESEG
代码段基地址

MOV     BYTE PTR DCODE.BASEM,DL
MOV     BYTE PTR DCODE.BASEH,DH

MOV     AX,ICODESEG
MOV     BX
MOV     WORD PTR ICODE.BASEL,AX ;计算并设置中断代码段基地址
址

MOV     BYTE PTR ICODE.BASEM,DL
MOV     BYTE PTR ICODE.BASEH,DH

MOV     AX,DSTACK0SEG
MOV     BX
MOV     WORD PTR DSTACK0.BASEL,AX ;计算并设置堆栈段 0 基
地址

MOV     BYTE PTR DSTACK0.BASEM,DL
MOV     BYTE PTR DSTACK0.BASEH,DH

MOV     AX,DDATASEG
MOV     BX
MOV     WORD PTR DDATAS.BASEL,AX ;计算并设置源数据基地址
MOV     BYTE PTR DDATAS.BASEM,DL
MOV     BYTE PTR DDATAS.BASEH,DH
;-----
ASSUME  DS:RDATASEG
MOV     AX,RDATASEG
MOV     DS,AX

```

```

MOV     AX,GDTSEG
MOV     BX,16
MUL     BX
ADD     AX,OFFSET GDT           ;计算并设置 GDT 基地址
ADC     DX,0
MOV     WORD PTR VGDTR.BASE,AX
MOV     WORD PTR VGDTR.BASE+2,DX

```

```

MOV     AX,IDTSEG
MUL     BX
ADD     AX,OFFSET IDT          ;计算并设置 IDT 基地址
ADC     DX,0
MOV     WORD PTR VIDTR.BASE,AX
MOV     WORD PTR VIDTR.BASE+2,DX

```

;-----

```

MOV     SSVAR,SS
MOV     SPVAR,SP              ;保存实模式下的堆栈指针

IN      AL,21H                ;中断屏蔽字寄存器端口地址
MOV     REGV,AL               ;保存中断屏蔽字节

```

```

LGDT    QWORD PTR VGDTR      ;装载 GDTR
SIDT    NORVIDTR              ;保存 IDTR 值
CLI                                           ;关中断

```

```

LIDT    QWORD    PTRVIDTR ;置 IDTR

```

```

MOV     EAX,CR0                ;切换到保护方式
OR      EAX,1
MOV     CR0,EAX

```

JUMP16 <CODEM\_SEL>,<OFFSET VIRTUAL> ;清指令预取队列,  
并真正进入保护方式

;-----

```

VIRTUAL:                               ;现在开始在保护方式下运行
CALL16 DCODE_SEL,0                    ;跳转到到 DCODESEG 代码段

```

```

MOV     AX,NORMAL_SEL    ;准备返回实模式
MOV     DS,AX             ;把规范段描述符装入各数据段寄存器
MOV     ES,AX
MOV     FS,AX
MOV     GS,AX
MOV     SS,AX

MOV     EAX,CRO           ;切换到实模式
AND     AL,11111110B
MOV     CRO,EAX
JUMP16  <SEG REAL>,<OFFSET REAL>    ;清指令预取队列,进入实

```

方式

```

REAL:                                     ;现在又回到实方式

MOV     AX,RDATASEG
MOV     DS,AX
LSS     SP,DWORD PTRSPVAR    ;恢复实模式下的堆栈指针
LIDT    NORVIDTR             ;恢复 IDTR

MOV     AL,REGV              ;恢复中断屏蔽字节
OUT     21H,AL

STI                                     ;开中断

MOV     AX,4C00H
INT     21H                   ;程序终止

```

```

START   ENDP
CODEMLN = $
CODEMSEG ENDS

```

;-----

```

DCODESEG SEGMENT PARA USE16    ;DCODESEG 的 16 位代码段
ASSUME CS:DCODESEG
MOV     AX,DDATAS_SEL
MOV     DS,AX                  ;加载源数据段描述符
MOV     AX,DDATAO_SEL
MOV     ES,AX                  ;加载目标数据段描述符

```

```

        XOR     SI,SI
        XOR     DI,DI           ;设置指针初值
        MOV     CX, 8           ;设置传送长度
        INT     20H             ;用 INT 指令调用一个中断/异常处理程序来
传送数据
        RETF                    ;中断返回
DCODELEN= $
DCODESEG     ENDS              ;DCODESEG 的 16 位代码段结束
;-----
ICODESEG     SEGMENT PARA USE16 ;异常/中断处理代码段
        ASSUME  CS:ICODESEG

        PUSHAD                    ;保护现场

M1:        MOVSB                    ;传送
        LOOP   M1

        POPAD                     ;恢复现场

        IRETD                     ;段间返回

ICODELEN = $

ICODESEG     ENDS
;=====
=====

                END      START

```

#### 6.4.4 用任务门实现中断/异常处理实验

如果中断向量号所指示的门描述符是任务门描述符，那么控制转移到一个作为独立的任务方式出现的处理程序。任务门中含有 48 位全指针。16 位选择子是指向描述符对应处理程序任务的 TSS 段的选择子，也即该选择子指示一个可用的 386TSS。通过任务门的转移与通过任务门到一个可用的 386TSS 的 CALL 指令的转移很相似。

在响应中断或处理异常时，使用任务门可提供一个处理程序任务的自动调度。这种任务调度由硬件直接执行。

##### 1. 实验内容



本实验是通过调用任务门实现中断/异常的转移,也是通过 INT 20H 调用 20H 号异常处理程序实现数据传输,其中调用 INT 指令的代码段为 0 级代码段, IDT 中 20H 号门描述符为任务门描述符。

```
INT20      Gate    <0,TSS3_Sel,,ATTASKGate,>
```

；对应 20H 号异常/中断处理程序段的任务门描述符

本实验可实现将一个数据段中的数据搬移到另一个数据段所指定的地址空间中(指定地址给定为 00004000H)。源数据段 DTEST 值为 11H,22H,33H,44H,55H,66H,77H,88H,在程序运行结束后会将此内容传输到地址为 00004000H 开始的地址空间中。

## 2. 实验步骤

(1) 编写实验程序(例程文件名为: 6-4-2.ASM)。

(2) 编译、链接无误后装入系统。

(3) 点击 RUN 运行程序,待程序运行停止。

(4) 查看运行结果。

在输出区的调试栏输入:

D0000: 4000

然后回车,即可看到实验运行结果,将数据传输到了给定的地址空间中。

### 实验程序清单

```

;-----
;文件名:6-4-2.ASM
;功能:通过任务门转移的中断/异常处理
;-----
INCLUDE      386SCD. INC
;-----
GDTSEG      SEGMENT PARA USE16      ;全局描述符表数据段(16 位)
;-----
GDT          LABEL    BYTE          ;全局描述符表
DUMMY        DESC     <>            ;空描述符
NORMAL       DESC     <0FFFFH,,ATDW,,> ;规范段描述符

CODEM        DESC     <0FFFFH,CODEMSEG,,ATCE,,> ;主程序段描述符

TSSTABLE1    DESC     <0FFFFH,TSS1SEG,,,>      ;任务 1 状态段 TSS 描述符
TSSTABLE2    DESC     <0FFFFH,TSS2SEG,,,>      ;任务 2 状态段 TSS 描述符
TSSTABLE3    DESC     <0FFFFH,TSS3SEG,,,>      ;任务 3 状态段 TSS 描述符

DCODE2       DESC     <0FFFFH,DCODE2SEG,,ATCE,,> ;任务 2 的 0 级代码段描述符

```

符

```

        DCODE3          DESC      <0FFFFH, DCODE3SEG, , ATCE+DPL3, , > ;任务 3 的 3 级代码段描述符 (DPL=3)

        DSTACK1         DESC      <0FFFFH, DSTACK1SEG, , ATDW, , > ;任务 1 的 0 级堆栈段描述符
        DSTACK2         DESC      <0FFFFH, DSTACK2SEG, , ATDW, , > ;任务 2 的 0 级堆栈段描述符
        DSTACK3         DESC      <0FFFFH, DSTACK3SEG, , ATDW+DPL3, , > ;任务 3 的 3 级堆栈段描述符 (DPL=3)

        DDATA1          DESC      <0FFFFH, DDATA1SEG, , ATDW, , > ;任务 1 的数据段描述符
        DDATA2          DESC      <0FFFFH, DDATA2SEG, , ATDW, , > ;任务 2 的数据段描述符
        DDATAS          DESC      <0FFFFH, DDATASSEG, , ATDW+DPL3, , > ;源数据段描述符 (DPL=3)
        DDATA0          DESC      <0FFFFH, 4000H, , ATDW+DPL3, , > ;目的数据段描述符 (DPL=3)
;-----
NORMAL_SEL      =      NORMAL-GDT      ;规范段描述符选择子
CODEM_SEL      =      CODEM-GDT      ;主程序段选择子
TSS1_SEL       =      TSSTABLE1-GDT    ;任务 1 状态段选择子
TSS2_SEL       =      TSSTABLE2-GDT    ;任务 2 状态段选择子
TSS3_SEL       =      TSSTABLE3-GDT    ;任务 3 状态段选择子
DCODE2_SEL     =      DCODE2-GDT      ;任务 2 代码段描述符选择子
DCODE3_SEL     =      DCODE3-GDT+RPL3  ;任务 3 代码段描述符选择子 (RPL=3)
DSTACK1_SEL    =      DSTACK1-GDT     ;0 级堆栈段 1 描述符选择子
DSTACK2_SEL    =      DSTACK2-GDT     ;0 级堆栈段 2 描述符选择子
DSTACK3_SEL    =      DSTACK3-GDT+RPL3 ;3 级堆栈段 3 描述符选择子 (RPL=3)
DDATA1_SEL     =      DDATA1-GDT      ;任务 1 的数据段描述符对应选择子
DDATA2_SEL     =      DDATA2-GDT      ;任务 2 的数据段描述符对应选择子
DDATAS_SEL     =      DDATAS-GDT+RPL3 ;源数据段选择子 (RPL=3)
DDATA0_SEL     =      DDATA0-GDT+RPL3 ;目的数据段选择子 (RPL=3)
GDTLEN         =      $-GDT           ;全局描述符表长度
GDTSEG         ENDS                  ;全局描述符表段定义结束
;-----
IDTSEG         SEGMENT      PARA      USE16      ;16 位中断/异常处理程序段
IDT            LABEL      BYTE      ;定义中断描述符表 IDT
              REPT      32      ;从 00H---1FH 的 32 个中断门描述符
              GATE      <, , , AT386IGATE, >

```

```

                                ENDM

                                INT20      GATE      <0, TSS3_SEL, , ATTASKGATE, > ;对应 20H 号异常/中断处理程序段
                                的任务门描述符

                                REPT      256-33                                ;从 21H---FFH 的 223 个中断门描述符

                                GATE      <, , , AT386IGATE, >
                                ENDM

                                IDTLEN      =      $-IDT                        ;中断描述符表长度
                                IDTSEG      ENDS                                ;16 位中断/异常处理程序段结束
                                ;-----

                                TSS1SEG      SEGMENT PARA USE16                ;任务 1 的任务状态段 TSS
                                TEMPTASK      TSS      <>
                                DB      OFFH                                ;I/O 许可位图结束标志

                                TSS1LEN      =      $
                                TSS1SEG      ENDS                                ;任务 1 状态段 TSS 结束
                                ;-----

                                TSS2SEG      SEGMENT PARA USE16                ;任务状态段 TSS2
                                DD      0                                    ;链接字
                                DD      DSTACK2LEN                        ;0 级堆栈指针
                                DD      DSTACK2_SEL                      ;0 级堆栈选择子
                                DD      0                                    ;1 级堆栈指针(实例不使用)
                                DD      0                                    ;1 级堆栈选择子(实例不使用)
                                DD      0                                    ;2 级堆栈指针
                                DD      0                                    ;2 级堆栈选择子
                                DD      0                                    ;CR3
                                DD      0                                    ;EIP
                                DD      200H                              ;EFLAGS
                                DD      OFFFH                              ;EAX
                                DD      0                                    ;ECX
                                DD      0                                    ;EDX
                                DD      0                                    ;EBX
                                DD      DSTACK2LEN                        ;ESP
                                DD      0                                    ;EBP
                                DD      0                                    ;ESI
                                DD      0                                    ;EDI
                                DD      DDATA2_SEL                        ;ES

```

```

        DD    DCODE2_SEL        ;CS
        DD    DSTACK2_SEL       ;SS
        DD    DDATA2_SEL        ;DS
        DD    DDATA2_SEL        ;FS
        DD    DDATA2_SEL        ;GS
        DD    0                  ;LDTR
        DW    0                  ;调试陷阱标志
        DW    $+2                ;指向 I/O 许可位图
        DB    OFFH               ;I/O 许可位图结束标志

TSS2LEN    =    $
TSS2SEG    ENDS                ;任务状态段 TSS 结束
;-----

```

```

TSS3SEG    SEGMENT PARA USE16   ;任务状态段 TSS3
        DD    0                  ;链接字
        DD    0                  ;0 级堆栈指针
        DD    0                  ;0 级堆栈选择子
        DD    0                  ;1 级堆栈指针(实例不使用)
        DD    0                  ;1 级堆栈选择子(实例不使用)
        DD    0                  ;2 级堆栈指针
        DD    0                  ;2 级堆栈选择子
        DD    0                  ;CR3
        DD    0                  ;EIP
        DD    200H               ;EFLAGS
        DD    OFFFH              ;EAX
        DD    0                  ;ECX
        DD    0                  ;EDX
        DD    0                  ;EBX
        DD    DSTACK3LEN         ;ESP
        DD    0                  ;EBP
        DD    0                  ;ESI
        DD    0                  ;EDI
        DD    DDATA0_SEL         ;ES
        DD    DCODE3_SEL        ;CS
        DD    DSTACK3_SEL       ;SS
        DD    DDATAS_SEL        ;DS
        DD    DDATAS_SEL        ;FS
        DD    DDATAS_SEL        ;GS
        DD    0                  ;LDTR
        DW    0                  ;调试陷阱标志

```

```

        DW      $+2                ;指向 I/O 许可位图
        DB      OFFH              ;I/O 许可位图结束标志
TSS3LEN    =      $
TSS3SEG     ENDS                  ;任务状态段 TSS 结束
;-----
DSTACK1SEG  SEGMENT PARA USE16    ;任务 1 的 0 级堆栈段
DSTACK1LEN  =      512
        DB      DSTACK1LEN DUP(0) ;定义 512 个缓冲字节单元
DSTACK1SEG  ENDS                  ;任务 1 堆栈段结束
;-----
DSTACK2SEG  SEGMENT PARA USE16    ;任务 2 的 0 级堆栈段
DSTACK2LEN  =      512
        DB      DSTACK2LEN DUP(?)
DSTACK2SEG  ENDS
;-----
DSTACK3SEG  SEGMENT PARA USE16    ;任务 3 的 3 级堆栈段
DSTACK3LEN  =      512
        DB      DSTACK3LEN DUP(?)
DSTACK3SEG  ENDS
;-----
DDATA1SEG   SEGMENT PARA USE16    ;任务 1 的数据段
DDATA1LEN   =      128
        DB      DDATA1LEN DUP(?)
DDATA1SEG   ENDS
;-----
DDATA2SEG   SEGMENT PARA USE16    ;任务 2 的数据段
DDATA2LEN   =      128
        DB      DDATA2LEN DUP(?)
DDATA2SEG   ENDS
;-----
DDATASSEG   SEGMENT PARA USE16    ;源数据段
DTEST       DB      11H, 22H, 33H, 44H, 55H, 66H, 77H, 88H ;定义源数据段数据
DDATASLEN   =      $              ;数据段的长度
DDATASSEG   ENDS                  ;源数据段定义结束
;-----
RDATASEG    SEGMENT      PARA USE16 ;实模式下的数据段
VGDR        PDESC      <GDTLEN-1,> ;GDT 伪描述符
VIDTR       PDESC      <IDTLEN-1,>  ;IDT 伪描述符
NORVIDTR    PDESC      <3FFH, 0>    ;用于保存原 IDTR 值

```

```

SPVAR          DW  ?
SSVAR          DW  ?                      ;用于保存实模式下的堆栈指针
REGV           DB  ?                      ;用于保存原中断屏蔽寄存器值
RDATASEG       ENDS

;-----
CODEMSEG       SEGMENT PARA USE16        ;主程序段
                ASSUME  CS:CODEMSEG
START          PROC

                ASSUME  DS:GDTSEG
                MOV     AX, GDTSEG
                MOV     DS, AX

                MOV     BX, 16
                MOV     AX, CODEMSEG
                MUL     BX
                MOV     WORD PTR CODEM. BASEL, AX      ;计算并设置主程序段基地址
                MOV     BYTE PTR CODEM. BASEM, DL
                MOV     BYTE PTR CODEM. BASEH, DH

                MOV     AX, TSS1SEG
                MUL     BX
                MOV     WORD PTR TSSTABLE1. BASEL, AX  ;计算并设置任务 1 基地址
                MOV     BYTE PTR TSSTABLE1. BASEM, DL
                MOV     BYTE PTR TSSTABLE1. BASEH, DH
                MOV     BYTE PTR TSSTABLE1. ATTRIBUTES, AT386TSS ;任务段 TSS1 的段属

                MOV     AX, TSS2SEG
                MUL     BX
                MOV     WORD PTR TSSTABLE2. BASEL, AX  ;计算并设置任务 2 基地址
                MOV     BYTE PTR TSSTABLE2. BASEM, DL
                MOV     BYTE PTR TSSTABLE2. BASEH, DH
                MOV     BYTE PTR TSSTABLE2. ATTRIBUTES, AT386TSS ;任务段 TSS2 的段属

                MOV     AX, TSS3SEG
                MUL     BX
                MOV     WORD PTR TSSTABLE3. BASEL, AX  ;计算并设置任务 3 基地址

```

性

性

性

```

MOV    BYTE PTR TSSTABLE3. BASEM, DL
MOV    BYTE PTR TSSTABLE3. BASEH, DH
MOV    BYTE PTR TSSTABLE3. ATTRIBUTES, AT386TSS ;任务段 TSS3 的段属

```

```

MOV    AX, DCODE2SEG
MUL    BX
MOV    WORD PTR DCODE2. BASEL, AX ;计算并设置代码段 2 基地址
MOV    BYTE PTR DCODE2. BASEM, DL
MOV    BYTE PTR DCODE2. BASEH, DH

```

```

MOV    AX, DCODE3SEG
MUL    BX
MOV    WORD PTR DCODE3. BASEL, AX ;计算并设置代码段 3 基地址
MOV    BYTE PTR DCODE3. BASEM, DL
MOV    BYTE PTR DCODE3. BASEH, DH

```

```

MOV    AX, DSTACK1SEG
MUL    BX
MOV    WORD PTR DSTACK1. BASEL, AX ;计算并设置堆栈段 1 基地址
MOV    BYTE PTR DSTACK1. BASEM, DL
MOV    BYTE PTR DSTACK1. BASEH, DH

```

```

MOV    AX, DSTACK2SEG
MUL    BX
MOV    WORD PTR DSTACK2. BASEL, AX ;计算并设置堆栈段 2 基地址
MOV    BYTE PTR DSTACK2. BASEM, DL
MOV    BYTE PTR DSTACK2. BASEH, DH

```

```

MOV    AX, DSTACK3SEG
MUL    BX
MOV    WORD PTR DSTACK3. BASEL, AX ;计算并设置堆栈段 3 基地址
MOV    BYTE PTR DSTACK3. BASEM, DL
MOV    BYTE PTR DSTACK3. BASEH, DH

```

```

MOV    AX, DDATA1SEG
MUL    BX
MOV    WORD PTR DDATA1. BASEL, AX ;计算并设置数据段 1 基地址
MOV    BYTE PTR DDATA1. BASEM, DL

```

```

MOV     BYTE PTR DDATA1. BASEH, DH

MOV     AX, DDATA2SEG
MUL     BX
MOV     WORD PTR DDATA2. BASEL, AX    ;计算并设置数据段 2 基地址
MOV     BYTE PTR DDATA2. BASEM, DL
MOV     BYTE PTR DDATA2. BASEH, DH

MOV     AX, DDATASSEG
MUL     BX
MOV     WORD PTR DDATAS. BASEL, AX    ;计算并设置数据段 3 基地址
MOV     BYTE PTR DDATAS. BASEM, DL
MOV     BYTE PTR DDATAS. BASEH, DH

```

```

;-----
ASSUME  DS:RDATASEG
MOV     AX, RDATASEG
MOV     DS, AX

MOV     AX, GDTSEG
MOV     BX, 16                      ;准备要加载到 GDTR 的伪描述符
MUL     BX
ADD     AX, OFFSET GDT              ;计算并设置 GDT 基地址
ADC     DX, 0
MOV     WORD PTR VGDTR. BASE, AX
MOV     WORD PTR VGDTR. BASE+2, DX

MOV     AX, IDTSEG
MUL     BX
ADD     AX, OFFSET IDT              ;计算并设置 IDT 基地址
ADC     DX, 0
MOV     WORD PTR VIDTR. BASE, AX
MOV     WORD PTR VIDTR. BASE+2, DX

MOV     SSVAR, SS
MOV     SPVAR, SP                  ;保存实模式下堆栈指针

IN      AL, 21H                    ;中断屏蔽字寄存器端口地址
MOV     REGV, AL                   ;保存中断屏蔽字节

```



```

LGDT    QWORD PTR VGDT    ;装载 GDTR
SIDT    NORVIDTR           ;保存 IDTR 值
CLI      ;关中断

LIDT    QWORD PTR VIDTR    ;置 IDTR

MOV      EAX, CRO           ;切换到保护方式
OR       EAX, 1
MOV      CRO, EAX
JUMP16   <CODEM_SEL>, <OFFSET VIRTUAL>;清指令预取队列, 并真正进入保护

```

方式

VIRTUAL: ;现在开始在保护方式下运行

```

MOV      AX, TSS1_SEL       ;装入任务 1
LTR      AX
MOV      AX, DDATA1_SEL     ;置数据段
MOV      DS, AX
MOV      ES, AX
MOV      FS, AX
MOV      GS, AX
MOV      ESP, DSTACK1LEN-1
MOV      AX, DSTACK1_SEL    ;置堆栈
MOV      SS, AX

CALL16   TSS2_SEL, 0        ;跳转到任务 2 的 0 级代码段

MOV      AX, NORMAL_SEL     ;把规范段描述符装入各数据段寄存

MOV      DS, AX
MOV      ES, AX
MOV      FS, AX
MOV      GS, AX
MOV      SS, AX

MOV      EAX, CRO           ;切换到实模式
AND      AL, 11111110B
MOV      CRO, EAX
JUMP16   <SEG REAL>, <OFFSET REAL> ;清指令预取队列, 进入实方式

```

器

REAL: ;现在又回到实方式

```
MOV    AX, RDATA SEG
MOV    DS, AX
```

```
LSS    SP, DWORD PTR SPVAR ;恢复实模式下的堆栈
LIDT   NORVIDTR             ;恢复 IDTR
```

```
MOV    AL, REGV             ;恢复中断屏蔽字节
OUT    21H, AL
```

```
ASSUME DS:TSS2SEG
MOV    AX, TSS2SEG
MOV    DS, AX
CALL   INIT_TSS2           ;恢复 TSS2 任务段表的初值, 以备下次
```

继续运行

```
ASSUME DS:TSS3SEG
MOV    AX, TSS3SEG
MOV    DS, AX
CALL   INIT_TSS3           ;恢复 TSS3 任务段表的初值, 以备下次
```

继续运行

```
STI                                         ;开中断
```

```
MOV    AX, 4C00H
INT     21H                               ;程序终止
```

```
START    ENDP
CODEMLN  = $
```

-----

```
INIT_TSS2 PROC NEAR ;恢复 TSS2 任务表的初值
```

```
    PUSH    DS
    MOV     AX, TSS2SEG
    MOV     DS, AX
```

```
    MOV     SI, 0 ;链接字
    MOV     AX, 0
```

```
MOV    [SI], AX

ADD    SI, 4        ;0 级堆栈指针
MOV    AX, DSTACK2LEN
MOV    [SI], AX

ADD    SI, 4        ;0 级堆栈选择子
MOV    AX, DSTACK2_SEL
MOV    [SI], AX

ADD    SI, 4        ;1 级堆栈指针
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;1 级堆栈选择子
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;2 级堆栈指针
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;2 级堆栈选择子
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;CR3
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;EIP
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;EFLAGS
MOV    AX, 200H
MOV    [SI], AX

ADD    SI, 4        ;EAX
```

```
MOV     AX, 0FFFH
MOV     [SI], AX

ADD     SI, 4           ;ECX
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;EDX
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;EBX
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;ESP
MOV     AX, DSTACK2LEN
MOV     [SI], AX

ADD     SI, 4           ;EBP
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;ESI
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;EDI
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;ES
MOV     AX, DDATA2_SEL
MOV     [SI], AX

ADD     SI, 4           ;CS
MOV     AX, DCODE2_SEL
MOV     [SI], AX
```

```

        ADD     SI, 4           ;SS
        MOV     AX, DSTACK2_SEL
        MOV     [SI], AX

        ADD     SI, 4           ;DS
        MOV     AX, DDATA2_SEL
        MOV     [SI], AX

        ADD     SI, 4           ;FS
        MOV     AX, DDATA2_SEL
        MOV     [SI], AX

        ADD     SI, 4           ;GS
        MOV     AX, DDATA2_SEL
        MOV     [SI], AX

        ADD     SI, 4           ;LDTR
        MOV     AX, 0
        MOV     [SI], AX

        ADD     SI, 4           ;调试陷阱标志
        MOV     AX, 0
        MOV     [SI], AX

        ADD     SI, 2           ;指向 I/O 许可位图
        MOV     AX, 68H         ;TSS 有 104 个字节
        MOV     [SI], AX

        ADD     SI, 2           ;I/O 许可位图结束标志
        MOV     AX, 0FFH
        MOV     [SI], AX

        POP     DS

        RET

INIT_TSS2      ENDP
;-----
INIT_TSS3      PROC      NEAR    ;恢复 TSS3 任务表的初值
        PUSH    DS

```

```
MOV    AX, TSS3SEG
MOV    DS, AX

MOV    SI, 0        ;链接字
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;0 级堆栈指针
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;0 级堆栈选择子
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;1 级堆栈指针 (实例不使用)
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;1 级堆栈选择子 (实例不使用)
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;2 级堆栈指针
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;2 级堆栈选择子
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;CR3
MOV    AX, 0
MOV    [SI], AX

ADD    SI, 4        ;EIP
MOV    AX, 0
MOV    [SI], AX
```

```
ADD     SI, 4           ;EFLAGS
MOV     AX, 200H
MOV     [SI], AX

ADD     SI, 4           ;EAX
MOV     AX, 0FFFH
MOV     [SI], AX

ADD     SI, 4           ;ECX
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;EDX
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;EBX
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;ESP
MOV     AX, DSTACK3LEN
MOV     [SI], AX

ADD     SI, 4           ;EBP
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;ESI
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;EDI
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;ESI
MOV     AX, DDATA0_SEL
MOV     [SI], AX
```

```
ADD     SI, 4           ;CS
MOV     AX, DCODE3_SEL
MOV     [SI], AX

ADD     SI, 4           ;SS
MOV     AX, DSTACK3_SEL
MOV     [SI], AX

ADD     SI, 4           ;DS
MOV     AX, DDATAS_SEL
MOV     [SI], AX

ADD     SI, 4           ;FS
MOV     AX, DDATAS_SEL
MOV     [SI], AX

ADD     SI, 4           ;GS
MOV     AX, DDATAS_SEL
MOV     [SI], AX

ADD     SI, 4           ;LDTR
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 4           ;调试陷阱标志
MOV     AX, 0
MOV     [SI], AX

ADD     SI, 2           ;指向 I/O 许可位图
MOV     AX, 68H         ;TSS 有 104 个字节
MOV     [SI], AX

ADD     SI, 2           ;I/O 许可位图结束标志
MOV     AX, 0FFH
MOV     [SI], AX

POP     DS
```



```

                RET
INIT_TSS3      ENDP
;-----
DCODE2SEG      SEGMENT PARA USE16                ;任务 2 的 0 级代码段
                ASSUME  CS:DCODE2SEG

                INT     20H                        ;用 INT 指令调用一个中断/异常处理
程序来传送数据

                IRET                                ;中断返回

DCODE2LEN      = $
DCODE2SEG      ENDS
;-----
DCODE3SEG      SEGMENT PARA USE16                ;任务 3 的 3 级代码段
                ASSUME  CS:DCODE3SEG

                MOV     AX, DDATAS_SEL
                MOV     DS, AX                      ;加载源数据段描述符
                MOV     AX, DDATAO_SEL
                MOV     ES, AX                      ;加载目标数据段描述符
                XOR     SI, SI
                XOR     DI, DI                      ;设置指针初值
                MOV     CX, 8                      ;设置传送长度
M1:            MOVSB                                ;传送
                LOOP    M1

                IRET                                ;段间返回

DCODE3LEN      = $
DCODE3SEG      ENDS
;-----
CODEMSEG      ENDS
;=====
                END      START

```

## 第 7 章 80X86 虚拟存储器的组织及其管理

80X86 虚拟存储器管理机制可分为分段管理机制和分页管理机制。段管理机制实现虚拟地址（由段和偏移构成的逻辑地址）到线性地址的转换，分页管理机制实现线性地址到物理地址的转换。如果不启用分页机制，则线性地址就作为物理地址。

### 7.1 分段管理机制

本节介绍保护模式下的段定义以及由段选择子及段内偏移构成的二维虚拟地址如何被转换为一维线性地址。

#### 7.1.1 段定义和虚拟地址到线性地址的转换

段是实现虚拟地址到线性地址转换机制的基础。在保护模式下，每个段由如下三个参数进行定义：段基地址(Base Address)、段界限(Limit)和段属性(Attributes)。

段基地址规定线性地址空间中段的开始地址。在 80X86 保护模式下，段基地址长 32 位。因为基地址长度与寻址地址的长度相同，所以任何一个段都可以从 32 位线性地址空间中的任何一个字节开始，而不像实模式下规定的边界必须被 16 整除。

段界限规定段的大小。在 80X86 保护模式下，段界限用 20 位表示，而且段界限可以是以字节为单位或以 4K 字节为单位。段属性中有一位对此进行定义，把该位称为粒度位，用符号 G 标记。G=0 表示段界限以字节为单位，于是 20 位的界限可表示的范围是 1 字节至 1M 字节，增量为 1 字节；G=1 表示段界限以 4K 字节为单位，于是 20 位的界限可表示的范围是 4K 字节至 4G 字节，增量为 4K 字节。当段界限以 4K 字节为单位时，实际的段界限 LIMIT 可通过下面的公式从 20 位段界限 Limit 计算出来：

$$\text{LIMIT} = \text{limit} * 4\text{K} + 0\text{FFFFH} = (\text{Limit SHL } 12) + 0\text{FFFFH}$$

所以当粒度为 1 时，段的界限实际上就扩展成 32 位。由此可见，在 80X86 保护模式下，段的长度可大大超过 64K 字节。

基地址和界限定义了段所映射的线性地址的范围。基地址 Base 是线性地址对应于段内偏移为 0 的虚拟地址，段内偏移为 X 的虚拟地址对应 Base+X 的线性地址。段内从偏移 0 到 Limit 范围内的虚拟地址对应于从 Base 到 Base+Limit 范围内的线性地址。

通过增加段界限，可以使段的容量得到扩展。这对于那些要在内存中扩展容量的普通数据段很有效，但对堆栈段情况就不是这样。因为堆栈底在高地址端，随着压栈操作的进行，堆栈向低地址方向扩展。为了适应普通数据段和堆栈数据段在两个相反方向上的扩展，数据段的段属性中安排了一个扩展方向位，标记为 ED。ED=0 表示向高端扩展，ED=1 表示向低端扩展。一般只有堆栈数据段才使用向低端扩展的属性(堆栈段也可使用向上扩展的段)，这是因为，向下扩展的段是为以下两个目的而设计的：

第一，堆栈段被定义为独特段，即 DS 和 SS 包含不同的选择器。

第二，一个堆栈段是靠将它复制到一个更大的段来扩充自己(而不是靠将现存的页增加到它的段上)。不打算用这种方法实现堆栈的设计者不需要定义向下扩展的段。

需要注意的是，只有数据段的段属性中才有扩展方向属性位 ED，也就是说只有数据段(堆栈段作为特殊的数据段)才有向上扩展和向下扩展之分，其它段都是自然的向上扩展。

数据段的扩展方向和段界限一起决定了数据段内偏移的有效范围。当段最大为 1M 字节时，在向高端扩展的段内，从 0 到 Limit 的偏移是合法有效的偏移，而从 Limit+1 到 1M-1 的偏移是非法无效的偏移；在向低端扩展的段内，情形刚好相反，从 0 到 Limit 的偏移是非法无效的偏移，而从 Limit+1 到 1M-1 的偏移是合法有效的偏移，注意边界值 Limit 对应地址的有效性。段最大为 4G 时，情形类似。由此可见，如果一个段是向下扩展的，则所有的偏移必须大于限长，因为其限长是指下限，其基地址从高地址出开始。反之，若一个段是向上扩展的，则所有偏移必须小于等于限长，因为其限长是指上限，基地址从低地址处开始。通过使用段环绕，可以把向下扩展段定义到任何线性地址且可定义为任何大小。

在每次把虚拟地址转换为线性地址的过程中，要对偏移进行检查。如果偏移不在有效的范围内，那么就引起异常。

段属性规定段的主要特性。例如上面已经提到的段粒度 G 就是段属性的一部分。在对段进行各种访问时，将对访问是否合法进行检查，主要依据是段属性。例如：如果向一个只读段进行写入操作，那么不仅不能写入，而且会引起异常。在下面会详细说明各个段属性位的定义和作用。

## 7.1.2 存储段描述符

用于表示上述定义段的三个参数的数据结构称为描述符。每个描述符长 8 个字节。在保护模式下，每一个段都有一个相应的描述符来描述。按描述符所描述的对象来划分，描述符可分为如下三类：存储段描述符、系统段描述符、门描述符(控制描述符)。下面先介绍存储段描述符。

### 1. 存储段描述符的格式

存储段是存放可由程序直接进行访问的代码和数据的段。存储段描述符描述存储段，所以存储段描述符也被称为代码和数据段描述符。80386 存储段描述符的格式如下表 7.1 所示。表中上面一排是对描述符 8 个字节的使用的说明，最低地址字节(假设地址为 m)在最右边，其余字节依次向左，直到最高字节(地址为 m+7)。下一排是对属性域各位的说明。

表 7.1 存储段描述格式

存储段描述符	m+7	m+6	m+5	m+4	m+3	m+2	m+1	m+0
	Base(31..24)		Attributes		Segment Base(23..0)		Segment Limit(15..0)	

存储段描述符属性	Byte m+6					Byte m+5				
	BIT7	BIT6	BIT5	BIT4	BIT3..BIT0	BIT7	BIT6..BIT5	BIT4	BIT3..BIT0	
	G	D	0	AVL	Limit(19..16)	P	DPL	DT1	TYPE	

从上表可知,长 32 位的段基地址(段开始地址)被安排在描述符的两个域中,其位 0—位 23 安排在描述符内的第 2—第 4 字节中,其位 24—位 31 被安排在描述符内的第 7 字节中。长 20 位的段界限也被安排在描述符的两个域中,其位 0—位 15 被安排在描述符内的第 0—第 1 字节中,其位 16—位 19 被安排在描述符内的第 6 字节的低 4 位中。

使用两个域存放段基地址和段界限的原因与 80286 有关。在 80286 保护模式下,段基地址只有 24 位长,而段界限只有 16 位长。80286 存储段描述符尽管也是 8 字节长,但实际只使用低 6 字节,高 2 字节必须置为 0。80386 存储段描述符这样的安排,可使得 80286 的存储段描述符的格式在 80386 下继续有效。

80386 描述符中的段属性也被安排在两个域中。下面对其定义及意义作说明。

(1) P 位称为存在(Present)位。P=1 表示描述符对地址转换是有效的,或者说该描述符所描述的段存在,即在内存中;P=0 表示描述符对地址转换无效,即该段不存在。使用该描述符进行内存访问时会引起异常。

(2) DPL 表示描述符特权级(Descriptor Privilege level),共 2 位。它规定了所描述段的特权级,用于特权级检查,以决定对该段能否访问。

(3) DT 位说明描述符的类型。对于存储段描述符而言位 DT=1,以区别与系统段描述符和门描述符(DT=0)。

(4) TYPE 说明存储段描述符所描述的存储段的具体属性。

其中的位 0 指示描述符是否被访问过,用符号 A 标记。A=0 表示尚未被访问,A=1 表示段已被访问。当把描述符的相应选择子装入到段寄存器时,80386 把该位置为 1,表明描述符已被访问。操作系统可测试访问位,已确定描述符是否被访问过。

其中的位 3 指示所描述的段是代码段还是数据段,用符号 E 标记。E=0 表示段为数据段,相应的描述符也就是数据段(包括堆栈段)描述符。数据段是不可执行的,但总是可读的。E=1 表示段是可执行段,即代码段,相应的描述符就是代码段描述符。代码段总是不可写的,若需要对代码段进行写入操作,则必须使用别名技术,即用一个可写的数据段描述符来描述该代码段,然后对此数据段进行写入。

在数据段描述符中(E=0 的情况),TYPE 中的位 1 指示所描述的数据段是否可写,用 W 标记。W=0 表示对应的数据段不可写。反之,W=1 表示数据段是可写的。注意,数据段总是可读的。TYPE 中的位 2 是 ED 位,指示所描述的数据段的扩展方向。ED=0 表示数据段向高端扩展,也即段内偏移必须小于等于段界限。ED=1 表示数据段向低扩展,段内偏移必须大于段界限。

在代码段描述符中(E=1 的情况),TYPE 中的位 1 指示所描述的代码段是否可读,用符号 R 标记。R=0 表示对应的代码段不可读,只能执行。R=1 表示对应的代码段可读可执行。在代码段中,TYPE 中的位 2 指示所描述的代码段是否是一致代码段,用 C 标记。C=0 表示对应的代码段不是一致代码段(普通代码段),C=1 表示对应的代码段是一致代码段。

存储段描述符中的 TYPE 字段所说明的属性可归纳为如表 7.2:

(5) G 为段界限粒度(Granularity)位。G=0 表示界限粒度为 1 字节;G=1 表示界限粒度为 4K 字节。注意,界限粒度只对段界限有效,对段基地址无效,段基地址总是以字节为单位。

表 7.2 存储段描述符属性

数据 段 类 型	类型值	说 明	代 码 段 类 型	类型值	说 明
	0	只读		8	只执行
	1	只读, 已访问		9	只执行, 已访问
	2	读/写		A	读/执行
	3	读/写, 已访问		B	读/执行, 已访问
	4	只读, 向低扩展		C	只执行, 一致码段
	5	只读, 向低扩展, 已访问		D	只执行, 一致码段, 已访问
	6	读/写, 向低扩展		E	读/执行, 一致码段
	7	读/写, 向低扩展, 已访问		F	读/执行, 一致码段, 已访问

(6) D 位是一个很特殊的位, 在描述可执行段、向下扩展数据段或由 SS 寄存器寻址的段(通常是堆栈段)的三种描述符中的意义各不相同。

在描述可执行段的描述符中, D 位决定了指令使用的地址及操作数所默认的大小。D=1 表示默认情况下指令使用 32 位地址及 32 位或 8 位操作数, 这样的代码段也称为 32 位代码段; D=0 表示默认情况下, 使用 16 位地址及 16 位或 8 位操作数, 这样的代码段也称为 16 位代码段, 它与 80286 兼容。可以使用地址大小前缀和操作数大小前缀分别改变默认的地址或操作数的大小。

在向下扩展数据段的描述符中, D 位决定段的上部边界。D=1 表示段的上部界限为 4G; D=0 表示段的上部界限为 64K, 这是为了与 80286 兼容。

在描述由 SS 寄存器寻址的段描述符中, D 位决定隐式的堆栈访问指令(如 PUSH 和 POP 指令)使用何种堆栈指针寄存器。D=1 表示使用 32 位堆栈指针寄存器 ESP; D=0 表示使用 16 位堆栈指针寄存器 SP, 这与 80286 兼容。

(7) AVL 位是软件可利用位。80386 对该位的使用未做规定, Intel 公司也保证今后开发生产的处理器只要与 80386 兼容, 就不会对该位的使用做任何定义或规定。

此外, 描述符内第 6 字节中的位 5 必须置为 0, 可以理解成是为以后的处理器保留的。

## 2. 存储段描述符的结构类型表示

根据存储段描述符的结构, 可定义如下的汇编语言描述符结构类型:

```

DESC      STRUC
LIMITL    DW      0      ;段界限低 16 位
BASEL     DW      0      ;基地址低 16 位
BASEM     DB      0      ;基地址中间 8 位
ATTRIB    DB      0      ;段属性
LIMITH    DB      0      ;段界限的高 4 位(包括段属性的高 4 位)
BASEH     DB      0      ;基地址的高 8 位
DESC ENDS
    
```

利用结构类型 DESC 能方便地在程序中说明存储段描述符。例如: 下面的描述符 DATAS 描述一个可读写的有效(存在的)数据段, 基地址是 100000H, 以字节为单位的界限是 0FFFFH, 描述符特权级 DPL=3。

```
DATAS     DESC      <0FFFFH,,10H,0F2H,,>
```

再如: 下述描述符 CODEA 描述一个只可执行的有效的 32 位代码段, 基地址是



12345678H, 以 4K 字节为单位的段界限值是 10H(以字节为单位的界限是 10FFFFH), 描述符特权级 DPL=0。

CODEA      DESC      <10H,5678H,34H,98H,0C0H,12H>

### 7.1.3 全局描述符和全局描述符表

一个任务会涉及多个段, 每个任务需要一个描述符来描述, 为了便于组织管理, 80X86 把描述符组织成线性表。由描述符组成的线性表称为描述符表。在 80X86 中有三种类型的描述符表: 全局描述符表 GDT(Global Descriptor Table)、局部描述符表 LDT(Local Descriptor Table)和中断描述符表 IDT(Interrupt Descriptor Table)。在整个系统中, 全局描述符表 GDT 和中断描述符表 IDT 只有一张, 局部描述符表 LDT 可以有若干张, 每个任务可以有一张。

例如, 下列描述符表有 6 个描述符构成:

DESCRIPTOR	TYPE	BYTES
DESC1	CODE	<1234H,5678H,34H,92H,,>
DESC2	CODE	<1234H,5678H,34H,93H,,>
DESC3	CODE	<5678H,1234H,56H,98H,,>
DESC4	CODE	<5678H,1234H,56H,99H,,>
DESC5	CODE	<0FFFFH,,10H,16H,,>
DESC6	CODE	<0FFFFH,,10H,90H,,>

每个描述符表本身形成一个特殊的数据段。这样的特殊数据段最多可包含有 8K(8192)个描述符。

关于中断描述符表 IDT 在本节不予介绍。

每个任务的局部描述符表 LDT 含有该任务自己的代码段、数据段和堆栈段的描述符, 也包含该任务所使用的一些门描述符, 如任务门和调用门描述符等。随着任务的切换, 系统当前的局部描述符表 LDT 也随之切换。

全局描述符表 GDT 含有每一个任务都可能或可以访问的段的描述符, 通常包含描述操作系统所使用的代码段、数据段和堆栈段的描述符, 也包含多种特殊数据段描述符, 如各个用于描述任务 LDT 的特殊数据段等。在任务切换时, 并不切换 GDT。

通过 LDT 可以使各个任务私有的各个段与其它任务相隔离, 从而达到受保护的目的。通过 GDT 可以使各任务都需要使用的段能够被共享。

一个任务可使用的整个虚拟地址空间分为相等的两半, 一半空间的描述符在全局描述符表中, 另一半空间的描述符在局部描述符表中。由于全局和局部描述符表都可以包含多达 8192 个描述符, 而每个描述符所描述的段的最大值可达 4G 字节, 因此最大的虚拟地址空间可为:

$$4GB \times 8192 \times 2 = 64\text{MMB} = 64\text{TB}$$

### 7.1.4 段选择子

在实模式下，逻辑地址空间中存储单元的地址由段值和段内偏移两部分组成。在保护模式下，虚拟地址空间(相当于逻辑地址空间)中存储单元的地址由段选择子和段内偏移两部分组成。与实模式相比，段选择子代替了段值。

段选择子长 16 位，其格式如下图 7.1 所示。

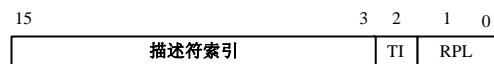


图 7.1 选择子格式

从表中可见，段选择子的高 13 位是描述符索引(Index)。所谓描述符索引是指描述符在描述符表中的序号。段选择子的第 2 位是引用描述符表指示位，标记为 TI(Table Indicator)，TI=0 指示从全局描述符表 GDT 中读取描述符；TI=1 指示从局部描述符表 LDT 中读取描述符。

选择子确定描述符，描述符确定段基地址，段基地址与偏移之和就是线性地址。所以，虚拟地址空间中的由选择子和偏移两部分构成的二维虚拟地址，就是这样确定了线性地址空间中的一维线性地址。

选择子的最低两位是请求特权级 RPL(Requested Privilege Level)，用于特权级检查。RPL 字段的用法如下：

每当程序试图访问一个段时，要把当前特权级与所访问段的特权级进行比较，以确定是否允许程序对该段的访问。使用选择子的 RPL 字段，将改变特权级的测试规则。在这种情况下，与所访问段的特权级比较的特权级不是 CPL，而是 CPU 与 RPL 中更外层的特权级。CPL 存放在 CS 寄存器的 RPL 字段内，每当一个代码段选择子装入 CS 寄存器中时，处理器自动地把 CPL 存放到 CS 的 RPL 字段。

由于选择子中的描述符索引字段用 13 位表示，所以可区分 8192 个描述符。这也就是描述符表最多包含 8192 个描述符的原因。由于每个描述符长 8 字节，根据上表所示选择子的格式，屏蔽选择子低 3 位后所得的值就是选择子所指定的描述符在描述符表中的偏移，这可认为是安排选择子高 13 位作为描述符索引的原因。

有一个特殊的选择子称为空(Null)选择子，它的 Index=0，TI=0，而 RPL 字段可以为任意值。空选择子有特定的用途，当用空选择子进行存储访问时会引起异常。空选择子是特别定义的，它不对应于全局描述符表 GDT 中的第 0 个描述符，因此处理器中的第 0 个描述符总不被处理器访问，一般把它置成全 0。但当 TI=1 时，Index 为 0 的选择子不是空选择子，它指定了当前任务局部描述符表 LDT 中的第 0 个描述符。

### 7.1.5 段描述符高速缓冲寄存器

在实模式下，段寄存器含有段值，为访问存储器形成物理地址时，处理器引用相应的某个段寄存器并将其值乘以 16，形成 20 位的段基地址。在保护模式下，段寄存器含有段选择子，如上所述，为了访问存储器形成线性地址时，处理器要使用选择子所指定的描述符中的基地址

等信息。为了避免在每次存储器访问时，都要访问描述符表而获得对应的段描述符，从 80286 开始每个段寄存器都配有一个高速缓冲寄存器，称之为段描述符高速缓冲寄存器或描述符投影寄存器，对程序员而言它是不可见的。每当把一个选择子装入到某个段寄存器时，处理器自动从描述符表中取出相应的描述符，把描述符中的信息保存到对应的高速缓冲寄存器中。此后对该段访问时，处理器都使用对应高速缓冲寄存器中的描述符信息，而不用再从描述符表中取描述符。

各段描述符高速缓冲寄存器之内容如表 7.3 所示。其中，32 位段基地址直接取自描述符，32 位的段界限取自描述符中 20 位的段界限，并根据描述符属性中的粒度位转换成以字节为单位。其它十个特性根据描述符中的属性而定，“Y”表示“是”，“N”表示“否”，“R”表示必须可读，“W”表示必须可写，“P”表示必须存在，“D”表示根据描述符中属性而定。

图 7.3 高速缓冲寄存器内容

段描述符高速缓冲寄存器的内容	段寄存器	段基地址	段界限	段 属 性									
				存在性	特权级	已取	粒度	扩展方向	可读性	可写性	可执行	堆栈大小	一致权
	CS	32 位基地址	32 位段界限	P	D	D	D	D	D	N	Y	—	D
	SS			P	D	D	D	D	R	W	N	D	—
	DS			P	D	D	D	D	D	D	N	—	—
	ES			P	D	D	D	D	D	D	N	—	—
	FS			P	D	D	D	D	D	D	N	—	—
	GS			P	D	D	D	D	D	D	N	—	—

段描述符高速缓冲寄存器再处理器内，所以可对其进行快速访问。绝大多数情况下，对存储器的访问是在对应选择子装入到段寄存器之后进行的，所以，使用段描述符高速缓冲寄存器可以得到很好的执行性能。

段描述符高速缓冲寄存器内部保存的描述符信息将一直保存到重新把选择子装载到段寄存器时再更新。程序员尽管不可见段描述符高速缓冲寄存器，但必须注意到它的存在和它的上述更新时机。例如，在改变了描述符表中的某个当前段的描述符后，也要更新对应的段描述符高速缓冲寄存器的内容，即使段选择子未作改变，这可通过重新装载段寄存器来实现。



## 7.2 分页管理机制

本文将介绍 80X86 的存储器分页管理机制以及线性地址如何转换为物理地址。

### 7.2.1 存储器分页管理机制

在保护模式下，控制寄存器 CR0 中的最高位 PG 位控制分页管理机制是否生效。如果 PG=1，分页机制生效，则线性地址需经过分页管理机制才能转换为物理地址；如果 PG=0，分页机制无效，线性地址就直接作为物理地址。必须注意，只有在保护模式下分页机制才可能生效。即只有在保证使 PE 位为 1 的前提下，才能够使 PG 位为 1，否则将引起通用保护故障。

分页机制把线性地址空间和物理地址空间分别划分为大小相同的块。这样的块称之为页。通过在线性地址空间的页与物理地址空间的页之间建立的映射，分页机制实现线性地址到物理地址的转换。线性地址空间的页与物理地址空间的页之间的映射可根据需要而确定，可根据需要而改变。线性地址空间的任何一页，可以映射到物理地址空间中的任何一页。

采用分页管理机制实现线性地址到物理地址转换映射的主要目的是便于实现虚拟存储器。不象段的大小可变，页的大小是相等并固定的。我们可以根据程序的逻辑来划分段，而根据实现虚拟存储器的方便来划分页。

在 80X86 中，页的大小固定为 4K 字节，每一页的边界地址必须是 4K 的倍数。因此，4G 大小的地址空间被划分为 1M 个页，页的开始地址具有“XXXXX000H”的形式。为此，我们把页开始地址的高 20 位 XXXXXH 称为页码。线性地址空间页的页码也就是页开始边界线性地址的高 20 位；物理地址空间页的页码也就是页开始边界物理地址的高 20 位。可见，页码左移 12 位就是页的开始地址，所以页码规定了页。

由于页的大小固定为 4K 字节，且页的边界是 4K 的倍数，所以在把 32 位线性地址转换成 32 位物理地址的过程中，低 12 位地址保持不变。也就是说，线性地址的低 12 位就是物理地址的低 12 位。假设分页机制采用的转换映射把线性地址空间的 XXXXXH 页映射到物理地址空间的 YYYYYH 页，那么线性地址 XXXXXxxxH 被转换为 YYYYYxxxH。因此，线性地址到物理地址的转换要解决的是线性地址空间的页到物理地址空间的页的映射，也就是线性地址高 20 位到物理地址高 20 位的转换。

### 7.2.2 线性地址到物理地址的转换

#### 1. 映射表结构

线性地址空间的页到物理地址空间的页之间的映射用表来描述。由于 4G 的地址空间划分为 1M 个页，因此，如果用一张表来描述这种映射，那么该映射表就要有 1M 个表项，若每个表项占用 4 个字节，那么该映射表就要占用 4M 字节。为避免映射表占用如此巨大的存储器资

源，所以 80386 把页映射表分为两级。

页映射表的第一级称为页目录表，存储在一个 4K 字节的物理页中。页目录表共有 1K 个表项，其中，每个表项为 4 字节长，包含对应第二级表所在物理地址空间页的页码。页映射表的第二级称为页表，每张页表也安排在一个 4K 字节的页中。每张页表都有 1K 个表项，每个表项为 4 字节长，包含对应物理地址空间页的页码。由于页目录表和页表均由 1K 个表项组成，所以使用 10 位的索引就能指定表项，即用 10 位的索引值乘以 4 加基地址就得到了表项的物理地址。

图 7.2 显示了由页目录表和页表构成的页映射表结构。从图 7.2 中可见，控制寄存器 CR3 指定页目录表；页目录表可以指定 1K 个页表，这些页表可以分散存放在任意的物理页中，而不需要连续存放；每张页表可以指定 1K 个物理地址空间的页，这些物理地址空间的页可以任意地分散在物理地址空间中。需要注意的是，存储页目录表和页表的基地址是对齐在 4K 字节边界上的。

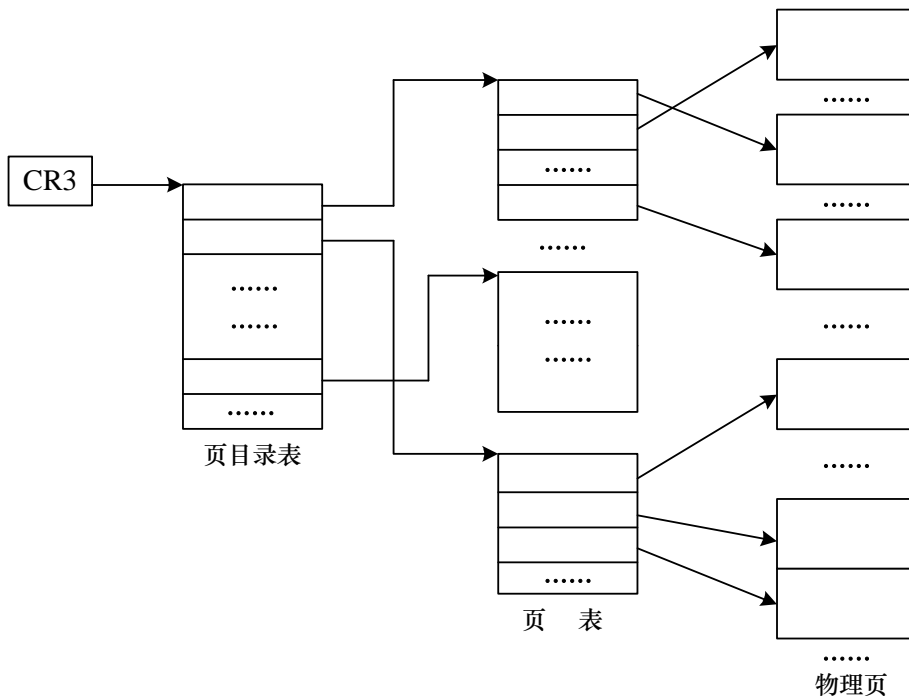


图 7.2 页映射表结构

## 2. 表项格式

页目录表和页表中的表项都采用如图 7.3 所示的格式。从图中可见，最高 20 位(位 12—位 31)包含物理地址空间页的页码，也就是物理地址的高 20 位。低 12 位包含页的属性。图 7.3 所示的属性中内容为 0 的位是 Intel 公司为 80486 等处理器所保留的位，在为 80386 编程使用到它们时必须设置为 0。在位 9 至位 11 的 AVL 字段供软件使用。表项的最低位是存在属性位，记作 P。P 位表示该表项是否有效。P=1 表项有效；P=0 表项无效，此时表项中的其余各位均可供软件使用，80386 不解释 P=0 的表项中的任何其它的位。在通过页目录表和页

表进行的线性地址到物理地址的转换过程中，无论在页目录表还是在页表中遇到无效表项，都会引起页故障。

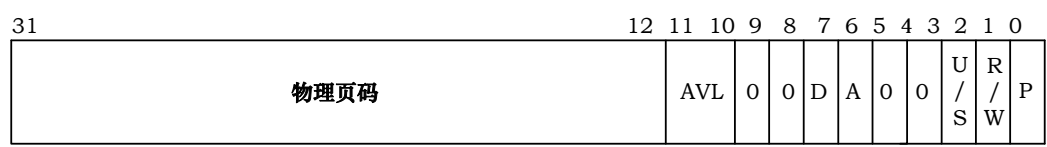


图 7.3 页目录表或页表的表项格式

3. 线性地址到物理地址的转换

分页管理机制通过上述页目录表和页表实现 32 位线性地址到 32 位物理地址的转换。控制寄存器 CR3 的高 20 位作为页目录表所在物理页的页码。首先把线性地址的最高 10 位(即位 22 至位 31)作为页目录表的索引，对应表项所包含的页码指定页表；然后，再把线性地址的中间 10 位(即位 12 至位 21)作为所指定的页目录表中的页表项的索引，对应表项所包含的页码指定物理地址空间中的一页；最后，把所指定的物理页的页码作为高 20 位，把线性地址的低 12 位不加改变地作为 32 位物理地址的低 12 位。

为了避免在每次存储器访问时都要访问内存中的页表，以便提高访问内存的速度，80386 处理器的硬件把最近使用的线性—物理地址转换函数存储在处理器内部的页转换高速缓存中。在访问存储器页表之前总是先查阅高速缓存，仅当必须的转换不在高速缓存中时，才访问存储器中的两级页表。页转换高速缓存也称为页转换查找缓存，记为 TLB。

在分页机制转换高速缓存中的数据与页表中数据的相关性，不是由 80386 处理器进行维护的，而必须由操作系统软件保存，也就是说，处理器不知道软件什么时候会修改页表，在一个合理的系统中，页表只能由操作系统修改，操作系统可以直接地在软件修改页表后通过刷新高速缓存来保证相关性。高速缓存的刷新通过装入处理器控制寄存器 CR3 完成，实际过程可能用如下的两条指令实现：

```
MOV    EAX, CR3
MOV    CR3, EAX
```

一个重要的修改页表项的特殊情况不需要对页转换高速缓存刷新，这种情况是指修改不存在表项的任一部分，即使 P 位本身从 P=0 改变为 P=1 时也一样，因为无效的表项不会存入高速缓存。因此，当无效的表项被改变时，不需要刷新高速缓存。这表明在从磁盘上读入一页使其存在时，不必刷新高速缓存。

在一个多处理器系统中，必须特别注意是否在一个处理器中执行的程序，会改变可能由另外的处理器同时访问的页表。在 80X86 处理器中，每当要更新页表项并设置 D 位和 A 位时，通过使用不可分的读/修改/写周期支持多处理器的配置。对于页表项的软件更新需要借助于使用 LOCK 前缀，从而保证修改页表的指令工作在不可分的读/修改/写周期中。在改变一个可能由另外的处理器使用的页表之前，最好使用一条加锁的 AND 指令在一个不可分的操作中将 P 位清除为 0，然后，该表项可根据要求进行修改，并随后把 P 位置成 1 而使表项成为可用。当修改页表项时必须及时通知(通常使用中断方式)系统中该表项已被高速缓存的所有处理器刷新各自的页转换高速缓存，以撤消该表项的旧拷贝。在表项的旧拷贝被刷新之前，各处理器仍可继续访问旧的页，并可以设置正被修改的表项的 D 位。如果这样做引起表项修改失败，则分页

机制高速缓存最好在标记为不存在之后，并在对表项进行另外的修改之前进行刷新。

#### 4. 不存在的页表

采用上述页映射表结构，存储全部 1K 张页表需要 4M 字节，此外还需要 4K 字节用于存储页目录表。这样的两级页映射表似乎反而比单一的整张页映射表多占用 4K 字节。其实不然，事实上不需要在内存中存储完整的两级页映射表。两级页映射表结构中对于线性地址空间中不存在的或未使用的部分不必分配页表。除必须给页目录表分配物理页外，仅当在需要时才给页表分配物理页，于是页映射表的大小就对应于实际使用的线性地址空间大小。因为任何一个实际运行的程序使用的线性地址空间都远小于 4G 字节，所以用于分配给页表的物理页也远小于 4M 字节。

页目录表项中的存在位 P 表明对应页表是否有效。如果 P=1，表明对应页表有效，可利用它进行地址转换；如果 P=0，表明对应页表无效。如果试图通过无效的页表进行线性地址到物理地址的转换，那么将引起页故障。因此，页目录表项中的属性位 P 使得操作系统只需给覆盖实际使用的线性地址范围的页表分配物理页。

页目录表项中的属性位 P 可用于把页表存储在虚拟存储器中。当发生由于所需页表无效而引起的页故障时，页故障处理程序再申请物理页，从磁盘上把对应的页表读入，并把对应页目录表项中的 P 位置 1。换言之，可以当需要时才为所要的页表分配物理页。这样页表占用的物理页数量可降到最小。

#### 5. 页的共享

由上述页映射表结构可见，分页机制没有全局页和局部页的规定。每一个任务可使用自己的页映射表独立地实现线性地址到物理地址的转换。但是，如果使每一个任务所用的页映射表具有部分相同的映射，那么也就可以实现部分页的共享。

常用的实现页共享的方法是线性地址空间的共享，也就是不同任务的部分相同的线性地址空间的映射信息相同，具体表现为部分页表相同或页表内的部分表项的页码相同。例如，如果任务 A 和任务 B 分别使用的页目录表 A 和页目录表 B 内的第 0 项中的页码相同，也就是页表 0 相同，那么任务 A 和任务 B 的 00000000H 至 003FFFFFFH 线性地址空间就映射到相同的物理页。再如，任务 A 和任务 B 使用的页表 0 不同，但这两张页表内第 0 至第 0FFFH 项的页码对应相同，那么任务 A 和任务 B 的 00000000H 至 000FFFFFFH 线性地址空间就映射到相同的物理页。需要注意的是，共享的页表最好由两个页目录中同样的目录项所指定。这一点很重要，因为它保证了在两个任务中同样的线性地址范围将映射到该全局区域。

## 7.2.3 页级保护和虚拟存储器支持

#### 1. 页级保护

80X86 不仅提供段级保护，也提供页级保护。分页机制只区分两种特权级。特权级 0、1 和 2 统称为系统特权级，特权级 3 称为用户特权级。页目录表和页表的表项中的保护属性位 R/W 和 U/S 就是用于对页进行保护。

页表项的位 1 是读写属性位，记作 R/W。R/W 位指示该表项所指定的页是否可读、写或执行。若 R/W=1，对表项所指定的页可进行读、写或执行；若 R/W=0，对表项所指定的页可



读或执行，但不能对该指定的页写入。但是，R/W 位对页的写保护只在处理器处于用户特权级时发挥作用；当处理器处于系统特权级时，R/W 位被忽略，即总可以读、写或执行。

表项的位 2 是用户/系统属性位，记作 U/S。U/S 位指示该表项所指定的页是否是用户级页。若 U/S=1，表项所指定的页是用户级页，可由任何特权级下执行的程序访问；如果 U/S=0，表项所指定的页是系统级页，只能由系统特权级下执行的程序访问。下述表 7.4 列出了上述属性位 R/W 和 U/S 所确定的页级保护下，用户级程序和系统级程序分别具有的对用户级页和系统级页进行操作的权限。

表 7.4 页级保护属性

U/S	R/W	用户级访问权限	系统访问权限
0	0	无	读/写/执行
0	1	无	读/写/执行
1	0	读/执行	读/写/执行
1	1	读/写/执行	读/写/执行

由表 7.4 可见，用户级页可以规定为只允许读/执行或规定为读/写/执行。系统级页对于系统级程序总是可读/写/执行，而对用户级程序总是不可访问的。与分段机制一样，外层用户级执行的程序只能访问用户级的页，而内层系统级执行的程序，既可访问系统级页，也可访问用户级页。与分段机制不同的是，在内层系统级执行的程序，对任何页都有读/写/执行访问权，即使规定为只允许读/执行的用户页，内层系统级程序也对该页有写访问权。

页目录表项中的保护属性位 R/W 和 U/S 对由该表项指定页表所指定的全部 1K 各页起到保护作用。所以，对页访问时引用的保护属性位 R/W 和 U/S 的值是组合计算页目录表项和页表项中的保护属性位的值所得。表 7.5 列出了组合计算前后的保护属性位的值，组合计算是“与”操作。

表 7.5 组合页保护属性

目录表项 U/S	页表项 U/S	组合 U/S	目录表项 R/W	页表项 R/W	组合 R/W
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	1	1	1

正如在 80386 地址转换机制中分页机制在分段机制之后起作用一样，由分页机制支持的页级保护也在由分段机制支持的段级保护之后起作用。先测试有关的段级保护，如果启用分页机制，那么在检查通过后，再测试页级保护。如果段的类型为读/写，而页规定为只允许读/执行，那么不允许写；如果段的类型为只读/执行，那么不论页保护如何，也不允许写。

页级保护的检查是在线性地址转换为物理地址的过程中进行的，如果违反页保护属性的规定，对页进行访问(读/写/执行)，那么将引起页异常。

## 2. 对虚拟存储器的支持

页表项中的 P 位是支持采用分页机制虚拟存储器的关键。P=1，表示表项指定的页存在于物理存储器中，并且表项的高 20 位是物理页的页码；P=0，表示该线性地址空间中的页所对应的物理地址空中的页不在物理存储器中。如果程序访问不存在的页，会引起页异常，这样操作系统可把该不存在的页从磁盘上读入，把所在物理页的页码填入对应表项并把表项中的 P 位置为 1，然后使引起异常的程序恢复运行。

此外，表项中的访问位 A 和写标志位 D 也用于支持有效地实现虚拟存储器。

表项的位 5 是访问属性位，记作 A。在为了访问某存储单元而进行线性地址到物理地址的转换过程中，处理器总是把页目录表内的对应表项和其所指定页表内的对应表项中的 A 位置 1，除非页表或页不存在，或者访问违反保护属性规定。所以，A=1 表示已访问过对应的物理页。处理器永不清除 A 位。通过周期性地检测及清除 A 位，操作系统就可确定哪些页在最近一段时间未被访问过。当存储器资源紧缺时，这些最近未被访问的页很可能就被选择出来，将它们从内存换出到磁盘上去。

表项的位 6 是写标志位，记作 D。在为了访问某存储单元而进行线性地址到物理地址的转换过程中，如果是写访问并且可以写访问，处理器就把页表内对应表项中的 D 位置 1，但并不把页目录表内对应表项中的 D 置 1。当某页从磁盘上读入内存时，页表中对应表项的 D 位被清 0。所以，D=1 表示已写过对应的物理页。当某页需要从内存换出到磁盘上时，如果该页的 D 位为 1，那么必须进行写操作(把内存中的页写入磁盘时，处理器并不清除对应页表项的 D 位)。但是，如果要写到磁盘上的页的 D 位为 0，那么不需要实际的磁盘写操作，而只要简单地放弃内存中该页即可。因为内存中的页与磁盘中的页具有完全相同的内容。

7.2.4 页异常

启用分页机制后，线性地址不再直接等于物理地址，线性地址要经过分页机制转换才成为物理地址。在转换过程中，如果出现下列情况之一就会引起页异常：

- (1) 涉及的页目录表内的表项或页表内的表项中的 P=0，即涉及到页不在内存；
- (2) 发现试图违反页保护属性的规定而对页进行访问。

报告页异常的中断向量号是 14(0EH)。页异常属于故障类异常。在进入故障处理程序时，保存的指令指针 CS 及 EIP 指向发生故障的指令。一旦引起页故障的原因被排除后，即可从页故障处理程序通过一条 IRET 指令，直接地重新执行产生故障的指令。

当页故障发生时，处理器把引起页故障的线性地址装入 CR2。页故障处理程序可以利用该线性地址确定对应的页目录项和页表项。页故障还在堆栈中提供一个出错码，出错码的格式如图 7.4 所示。

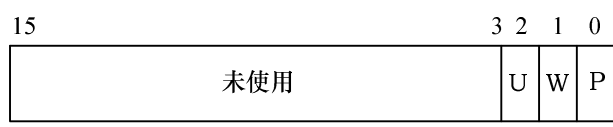


图 7.4 页故障出错格式

其中，U 位表示引起故障程序的特权级，U=1 表示用户特权级(特权级 3)，U=0 表示系统特权级(特权级 0、1 或 2)；W 位表示访问类型，W=0 表示读/执行，W=1 表示写；P 位表示异常类型，P=0 表示页不存在故障，P=1 表示保护故障。页故障的响应处理模式同其它故障一样。

## 第 8 章 保护模式下的存储器扩展及其应用实验

### 8.1 无分页机制的存储器扩展实验

#### 8.1.1 实验目的

1. 掌握 80X86 微机保护模式下存储器扩展的方法。
2. 掌握 80X86 微机保护模式下对扩展存储器的操作方法。

#### 8.1.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

#### 8.1.3 实验内容

在保护模式下，存储器的寻址为：段选择子加偏移，得到线性地址，由于本实验不启动分页机制，所以线性地址就等于实际内存的物理地址。

1. 利用保护模式机制，实现对外部存储器的扩展（本实验所定义的外部存储器空间为：00080000H~0008FFFFH）。

2. 利用保护模式的段管理机制，编程实现从地址空间 00080000H 开始将一段数据传送至地址空间从 00080028H 开始的存储器区域中，然后将它的值在屏幕上显出来。

本实验需要接线：将存储器的地址、数据、读写、字节使能及片选分别接到 80X86 系统扩展应用总线相对应的信号引脚上。如图 8.1:

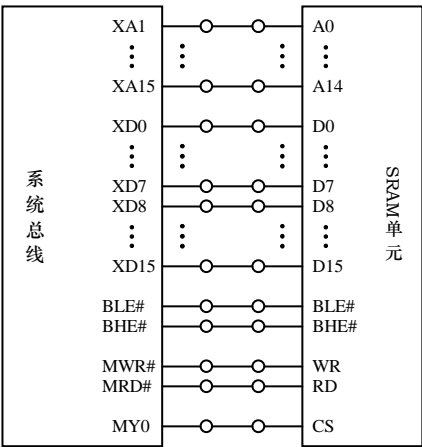


图 8.1 存储器扩展实验接线图

8.1.4 实验步骤

- 1. 编写实验程序（例程文件名为：PMEM1.ASM），按如图 8.1 进行实验接线。
  - 2. 编译、链接无误后装入系统。
  - 3. 点击 RUN 运行程序，待程序运行停止。
  - 4. 查看运行结果。
- 在输出区的显示栏显示如下信息：
- 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
- 5. 分析实验结果，理解分段的存储器管理机制。

实验程序清单

```
-----  
; 文件名:PMEM1.ASM  
; 实验内容：利用保护模式机制,实现外部存贮器扩展实验.  
; 此实验需要接线,把存储器的地址、数据、读写、字节使能及片选分别接到总线上。  
; 从 00080000H 的地址空间开始将一段数传至从 00080028H 开始的地址空间中,  
; 数据为 30-39, 41-5A  并将它的 ASCII 值在屏幕上显出来。  
-----  
XIANXIN_WULI_AD1      =    00080000H    ;线性地址 1 (不分页时就是物理地址)  
XIANXIN_WULI_AD2      =    00080028H    ;线性地址 2 (不分页时就是物理地址)  
-----  
INCLUDE                386SCD.INC  
-----  
DSEG                  SEGMENT PARA          USE16                                ;定义 16 位数据段
```



```

GDT      LABEL    BYTE                                ;全局描述符表
DUMMY    DESC     <>                                ;空描述符
CODEM    DESC     <OFFFFH, , , ATCE, , >            ;代码段描述符
DATAS    DESC     <OFFFFH, XIANXIN_WULI_AD1 AND OFFFFH, XIANXIN_WULI_AD1
SHR 16, ATDW, , >;源数据段描述符
DATA0    DESC     <OFFFFH, XIANXIN_WULI_AD2 AND OFFFFH, XIANXIN_WULI_AD2
SHR 16, ATDW, , >;目标数据段描述符
VGDTTR    PDESC    <GDTLEN-1, >                    ;伪描述符
GDTLEN    =        $-GDT                            ;全局描述符表长度
CODEM_SEL =        CODEM-GDT                        ;代码段选择子
DATAS_SEL =        DATAS-GDT                        ;源数据段选择子
DATA0_SEL =        DATA0-GDT                       ;目标数据段选择子
DSEG      ENDS                                        ;数据段定义结束
;-----
CSEG      SEGMENT USE16                                ;16 位代码段
          ASSUME  CS:CSEG, DS:DSEG
START     PROC
          MOV     AX, DSEG
          MOV     DS, AX
          ;准备要加载到 GDTR 的伪描述符
          MOV     BX, 16
          MUL     BX                                ;数据段地址左移 4 位
          ADD     AX, OFFSET GDT                    ;加上 GDT 的偏移得到物理地址
          ADC     DX, 0
          MOV     WORD PTR VGDTTR. BASE, AX        ;将得到的物理地址填入 VGDTTR 描述符
          MOV     WORD PTR VGDTTR. BASE+2, DX;
          ;设置代码段描述符
          MOV     AX, CS
          MUL     BX                                ;代码段地址左移 4 位
          MOV     WORD PTR CODEM. BASEL, AX        ;代码段开始偏移为 0
          MOV     BYTE PTR CODEM. BASEM, DL        ;将得到的物理地址填入 CODEM 描述符
          MOV     BYTE PTR CODEM. BASEH, DH
          ;-----
          MOV     AX, (XIANXIN_WULI_AD1 SHR 16) SHL 12 ;给地址从 00080000H 开始写
数 30-39
          MOV     DS, AX
          MOV     SI, (XIANXIN_WULI_AD1 AND OFFFFH)
          MOV     CX, 10                            ;送 10 个数据
          MOV     AX, 30H                            ;从 30H 开始

```

```

MEM1:  MOV     [SI], AX
        INC     SI
        ADD     AX, 1
        LOOP    MEM1

        MOV     CX, 26                ;给地址从 0008000AH 开始写数 41-5A
        MOV     AX, 41H              ;从 A 到 Z 总数 26 个数
MEM2:  MOV     [SI], AX
        INC     SI
        ADD     AX, 1
        LOOP    MEM2

;-----
        MOV     AX, DSEG
        MOV     DS, AX
        ;加载 GDTR
        LGDT     QWORD PTR VGDTR
        CLI                      ;关中断
        ;切换到保护方式
        MOV     EAX, CR0
        OR      EAX, 1
        MOV     CR0, EAX
        ;清指令预取队列, 并真正进入保护方式
        JUMP16  <CODEM_SEL>, <OFFSET_VIRTUAL>
VIRTUAL: ;现在开始在保护方式下运行
        MOV     AX, DATAS_SEL
        MOV     DS, AX                ;加载源数据段描述符
        MOV     AX, DATA0_SEL
        MOV     ES, AX                ;加载目标数据段描述符
        XOR     DI, DI                ;DI 清零
        XOR     SI, SI                ;SI 清零
        MOV     CX, 36                ;设置数据长度
        REPZ    MOVSB                ;通过串传送指令传数
        ;切换回实模式
        MOV     EAX, CR0
        AND     AL, 11111110B
        MOV     CR0, EAX
        ;清指令预取队列, 进入实方式
        JUMP16  <SEG_REAL>, <OFFSET_REAL>
REAL:   ;现在又回到实方式

```

```

;-----
MOV     AX, (XIANXIN_WULI_AD2 SHR 16) SHL 12 ;从地址 00080028H 开始输出
数 0-9
MOV     DS, AX
MOV     CX, 0AH
MOV     SI, (XIANXIN_WULI_AD2 AND 0FFFFH)
MEMO1: MOV     DL, [SI]
MOV     AH, 02H
INT     21H                                ;用 INT 21H 功能调用输出字符
ADD     SI, 1
LOOP    MEMO1

MOV     CX, 1AH                            ;从地址 00080032H 开始输出字母 A-Z
MEMO2: MOV     DL, [SI]
MOV     AH, 02H
INT     21H                                ;用 INT 21H 功能调用输出字符
ADD     SI, 1
LOOP    MEMO2
STI                                           ;开中断

MOV     AX, 4C00H
INT     21H                                ;程序终止

START      ENDP
;-----
CSEG      ENDS                                ;代码段定义结束
;-----
END      START

```

## 8.2 具有分页机制的存储器扩展实验

### 8.2.1 实验目的

1. 掌握 80X86 微机保护模式下分页机制的存储器扩展方法。
2. 掌握 80X86 微机保护模式下分页机制对扩展存储器的操作方法。

### 8.2.2 实验设备

PC 机一台，TD-PITE 实验装置一套。

### 8.2.3 实验内容

在保护模式下，启动分页机制的存储器管理后，线性地址不再等于物理地址，线性地址要经过分页机制转换才能成为物理地址。要建立分页机制，必须先创建页目录表及页表并对其在保护模式下进行初始化，然后启动分页机制，启动分页机制很简单，只要把控制寄存器 CR0 中的最高位 PG 位置 1 即可，具体指令如下：

```
MOV    EAX, CR0
OR      EAX, 80000000H
MOV     CR0, EAX
JMP     SHORT  PAGE
PAGE:
```

.....

关闭分页机制也很简单，把控制寄存器 CR0 中的 PG 位清 0 即可。

本实验为了简单，只有一个任务，并且没有局部描述符和中断描述符，不允许中断，不考虑发生异常。旨在体现分页机制的工作原理及过程。

实验建立了一张页目录表和一张页表，其所在物理页的地址为：

PDT\_AD=00010000H ;页目录表所在物理页的地址

PT\_AD =00012000H ;页表所在物理页的地址

另外，还给定了两个地址，实验中所要用到的线性地址和物理地址：

XIANXIN\_AD=00301028H ;线性地址 XIANXIN\_AD

WULI\_AD =00080028H ;线性地址 XIANXIN\_AD 对应的物理地址

实验内容是在创建并启动分页机制后，将源数据段中的字符串“Page Is Successful!”传送到线性地址 XIANXIN\_AD 中，由于是在分页机制下工作，硬件系统会根据用户初始化的要

求，通过控制寄存器 CR3 和线性地址 XIANXIN\_AD 中的内容，将线性地址 XIANXIN\_AD 映射到物理地址 WULI\_AD 中，然后将物理地址 WULI\_AD 中的内容在屏幕上输出显示。可参考图 8.2:

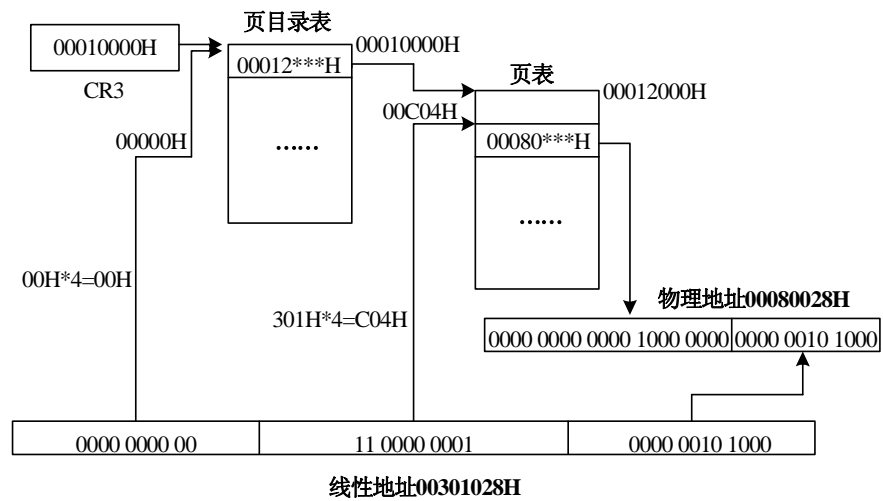


图 8.2 线性地址到物理地址的转换过程

其中，图 8.2 中的 “\*\*\*” 表示页表的属性。

本实验需要接线：将存储器的地址、数据、读写、字节使能及片选分别接到 80X86 系统扩展应用总线相对应的信号引脚上。如图 8.1:

### 8.2.4 实验步骤

1. 编写实验程序（例程文件名为：PMEM2.ASM），按如图 8.1 进行实验接线。
2. 编译、链接无误后装入系统。
3. 点击 RUN 运行程序，待程序运行停止。
4. 查看运行结果。

在输出区的显示栏显示如下信息：

Page Is Successful!

5. 分析实验结果，理解分页的存储器管理机制。

实验程序清单

```
;-----  
;文件名:PMEM2.ASM  
;功能: 分页机制  
; 为了简单化,只有一个任务,并且没有 LDT 和 IDT,不允许中断,  
; 不考虑发生异常,不使用堆栈.SRAM:00000000H--0001FFFFH
```

; 外部存储器 MY0 地址空间: 00080000H--0009FFFFH

-----

INCLUDE 386SCD.INC

-----

;PL = 1 ;页存在属性位 P 值

;RWR = 0 ;R/W 属性位值,读/执行

;RWW = 2 ;R/W 属性位值,读/写/执行

;USS = 0 ;U/S 属性值,系统级

;USU = 4 ;U/S 属性值,用户级

PDT\_AD = 00010000H ;页目录表所在所理页的地址

PT\_AD = 00012000H ;页表所在所理页的地址

XIANXIN\_AD = 00301028H ;线性地址 XIANXIN\_AD

WULI\_AD = 00080028H ;线性地址 XIANXIN\_AD 对应的物理地址

-----

DSEG SEGMENT PARA USE16 ;定义 16 位数据段

GDT LABEL BYTE ;全局描述符表

DUMMY DESC <> ;空描述符

NORMAL DESC <0FFFFH,,,ATDW,,> ;规范段描述符

PDTABLE DESC <0FFFFH,PDT\_AD AND 0FFFFH,PDT\_AD SHR

16,ATDW,,>;页目录表所在段描述符

PTABLE DESC <0FFFFH,PT\_AD AND 0FFFFH,PT\_AD SHR 16,ATDW,,> ;

页表所在段描述符

CODEM DESC <0FFFFH,,,ATCE,,> ;主代码段描述符

PCODE DESC <0FFFFH,,,ATCE,,> ;分页机制代码段描述符

DDATA DESC <0FFFFH,,,ATDW,,> ;源数据段描述符

DATAO DESC <0FFFFH,XIANXIN\_AD AND 0FFFFH,XIANXIN\_AD SHR

16,ATDW,,>;目标数据段描述符

NORMAL\_SEL = NORMAL-GDT ;规范段描述符选择

子

PDT\_SEL = PDTABLE-GDT ;页目录表所在段描

述符选择子

PT\_SEL = PTABLE-GDT ;页表所在段描述符选择

子

CODEM\_SEL = CODEM-GDT ;主代码段选择子

PCODE\_SEL = PCODE-GDT ;分页机制代码段描

述符选择子

DDATA\_SEL = DDATA-GDT ;源数据段描述符选

择子

DATAO\_SEL = DATAO-GDT ;目标数据段选择子

```

GDTLEN      =      $-GDT                      ;全局描述符表长度

VGDTTR      PDESC <GDTLEN-1,>                ;伪描述符 VGDTTR
SPVAR        DW  ?
SSVAR        DW  ?                            ;保存实模式下堆栈指针

DSEG         ENDS                            ;数据段定义结束
;-----
DDATASEG     SEGMENT PARA USE16                ;定义 16 位源数据段
MESS         DB 'Page Is Successful! ','$'      ;字符$表示字符串结束
ML           =  $-MESS                          ;字符串的长度
DDATASEG     ENDS                            ;源数据段定义结束
;-----
CSEG         SEGMENT USE16                      ;16 位主代码段
              ASSUME  CS:CSEG,DS:DSEG
START        PROC
              MOV     AX,DSEG
              MOV     DS,AX
              ;准备要加载到 GDTR 的伪描述符
              MOV     BX,16
              MUL     BX                        ;数据段地址左移 4 位
              ADD     AX,OFFSET GDT              ;加上 GDT 的偏移得到物理地
址
              ADC     DX,0
              MOV     WORD PTR VGDTTR.BASE,AX    ;将得到的物理地址填入
VGDTTR 描述符
              MOV     WORD PTR VGDTTR.BASE+2,DX
              ;设置主代码段描述符
              MOV     AX,CS
              MUL     BX                        ;代码段地址左移 4 位
              MOV     WORD PTR CODEM.BASEL,AX    ;代码段开始偏移为 0
              MOV     BYTE PTR CODEM.BASEM,DL    ;将得到的物理地址填入
CODEM 描述符
              MOV     BYTE PTR CODEM.BASEH,DH
              ;设置分页机制代码段描述符
              MOV     AX,PCODESEG
              MUL     BX                        ;分页机制代码段地址左移 4 位
              MOV     WORD PTR PCODE.BASEL,AX    ;代码段开始偏移为 0

```

```

MOV      BYTE PTR PCODE.BASEM,DL      ;将得到的物理地址填入
CODEM 描述符
MOV      BYTE PTR PCODE.BASEH,DH
;设置源代码段描述符
MOV      AX,DDATASEG
MUL      BX                          ;源数据段地址左移 4 位
MOV      WORD PTR DDATA.BASEL,AX      ;源数据段开始偏移为 0
MOV      BYTE PTR DDATA.BASEM,DL      ;将得到的物理地址填入
DATAS 描述符
MOV      BYTE PTR DDATA.BASEH,DH
;-----
MOV      SSVAR,SS
MOV      SPVAR,SP                    ;保存实模式下堆栈指针
;加载 GDTR
LGDT     QWORD PTR VGDTR
CLI                               ;关中断
;切换到保护方式
MOV      EAX,CR0
OR       EAX,1
MOV      CR0,EAX
;清指令预取队列,并真正进入保护方式
JUMP16   <CODEM_SEL>,<OFFSET VIRTUAL>
VIRTUAL: ;现在开始在保护方式下运行

JUMP16   <PCODE_SEL>,INIT_PDT

TOREAL:
MOV      AX,NORMAL_SEL              ;把规范段描述符装入各数据
据段寄存器
MOV      DS,AX
MOV      ES,AX
MOV      FS,AX
MOV      GS,AX
MOV      SS,AX
;切换回实模式
MOV      EAX,CR0
AND      AL,11111110B
MOV      CR0,EAX
;清指令预取队列,进入实方式

```



```

        JUMP16    <SEG REAL>,<OFFSET REAL>
REAL:   ;现在又回到实方式
        MOV      AX,DSEG
        MOV      DS,AX
        LSS      SP,DWORD PTRSPVAR      ;恢复实模式下的堆栈

        MOV      AX,(WULI_AD SHR 16) SHL 12
        MOV      DS,AX                  ;送目标数据段
        MOV      DX,OFFSET (WULI_AD AND 0FFFFH)
        MOV      AH,09H
        INT      21H                    ;用 INT 21H 功能调用显示 BUF
数据段的内容

        STI                                ;开中断

        MOV      AX,4C00H
        INT      21H                    ;程序终止

START   ENDP
;-----
PCODESEG SEGMENT PARA USE16 ;初始化页描述符,启动分页机制,16 位代码段
        ASSUME   CS:PCODESEG
INIT_PDT:                                ;开始初始化页目录
        MOV      AX,PDT_SEL
        MOV      ES,AX
        XOR      DI,DI
        MOV      CX,1024
        XOR      EAX,EAX                ;把页目录表 PDT 中全部表项
置成无效
        REP      STOSD
                                           ;置页目录表中表项 PT
        MOV      DWORD PTR ES:[0],PT_AD OR (USS+RWW+PL);将页表的地
址送到 ES 段的 0 偏移 32 位
        MOV      AX,PT_SEL              ;初始化页表
        MOV      ES,AX
        XOR      DI,DI
        MOV      CX,1024
        XOR      EAX,EAX

```

```

        OR      EAX,US$+RWW+PL
PTLOOP: STOSD
        ADD     EAX,1000H           ;先全部置成直接对应等地址的物
理页
        LOOP   PTLOOP             ;再特别设置一项
        MOV     DI,(XIANXIN_AD SHR 12)*4 ;将线性地址的中间 10 位(12-21)
作为已指定页表的索引
        MOV     DWORD PTR ES:[DI],WULI_AD+USU+RWW+PL;将物理地址
送到 ES 段的 DI 偏移 32 位

        MOV     EAX,PDT_AD         ; 页 目 录 表 的 地 址
00010000H 送给 CR3
        MOV     CR3,EAX
        MOV     EAX,CR0            ;控制寄存器 CR0 最高位置为
1,使分页机制有效
        OR      EAX,80000000H
        MOV     CR0,EAX            ;启动分页机制
        JMP     SHORT PAGEE
PAGEE:

;-----
        MOV     AX,DDATA_SEL       ;送源数据段选择子
        MOV     DS,AX
        MOV     AX,DATA0_SEL       ;送目标数据段选择子
        MOV     ES,AX
        XOR     DI,DI              ;DI 清零
        XOR     SI,SI              ;SI 清零
        MOV     CX,ML              ;设置数据长度
        REPZ    MOVSB              ;通过串传送指令传数
;-----
PTT:
        MOV     EAX,CR0            ;关闭分页机制
        AND     EAX,7FFFFFFFH
        MOV     CR0,EAX

        JUMP16  <CODEM_SEL>,TOREAL ;转向主代码段,准备退出
保护模式

```

PCODELEN = \$

```
PCODESEG      ENDS
;-----
CSEG           ENDS           ;代码段定义结束
;-----
                END      START
```

# 附录 1 Wmd86 联机软件使用说明

## 附 1.1 菜单功能

### 1. 文件菜单项

文件菜单如附图 1-1 所示。

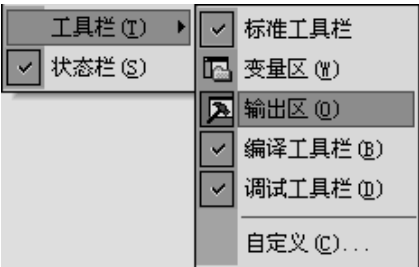
- (1) 新建 (N)：用此命令在 Wmd86 中建立一个新文档。
- (2) 打开 (O)：用此命令在窗口中打开一个现存的文档。
- (3) 关闭 (C)：用此命令来关闭当前活动文档。
- (4) 保存 (S)：用此命令将当前活动文档保存到它的当前文件名和目录下。当您第一次保存文档时，Wmd86 显示另存为对话框以便您命名您的文档。
- (5) 另存为 (A)：用此命令来保存并命名活动文档。
- (6) 打印 (P)：用此命令来打印一个文档。
- (7) 打印预览 (V)：用此命令来打印当前显示活动文档。
- (8) 打印设置 (R)：用此命令来选择连接的打印机及其设置。
- (9) 最近浏览文件：通过此列表，直接打开最近打开过的文件。
- (10) 退出 (X)：用此命令来结束 Wmd86 的运行阶段。Wmd86 会提示您保存尚未保存的改动。



附图 1-1 文件菜单

### 2. 查看菜单项

查看菜单如附图 1-2 所示。



附图 1-2 查看菜单

- (1) 工具栏 (T)：显示或隐藏工具栏
- (2) 状态栏 (S)：显示或隐藏状态栏
- (3) 工具栏
  - a、标准工具栏：用此命令可显示和隐藏标准工具栏。标准工具栏包括了 Wmd86 中一些

最普通命令的按钮，如文件打开。在工具栏被显示时，一个打勾记号出现在该菜单项目的旁边。

- b、变量区 (W)：用此命令可显示和隐藏寄存器/变量/堆栈区。
- c、输出区 (O)：用此命令可显示和隐藏输出区。
- d、编译工具栏 (B)：用此命令可显示和隐藏编译工具栏。
- e、调试工具栏 (D)：用此命令可显示和隐藏调试工具栏。
- f、自定义 (C)：见自定义功能。

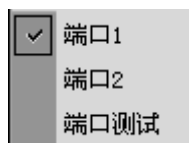
### 3. 端口菜单项

端口菜单如附图 1-3 所示。

(1) 端口 1：此命令用来选择串口 1 进行联机通讯，该命令会对串口 1 进行初始化操作，并进行联机测试，报告测试结果。

(2) 端口 2：此命令用来选择串口 2 进行联机通讯，该命令会对串口 2 进行初始化操作，并进行联机测试，报告测试结果。

(3) 端口测试：此命令用来对当前选择的串口进行联机通讯测试，并报告测试结果。



附图 1-3 端口菜单

### 4. 编译菜单项

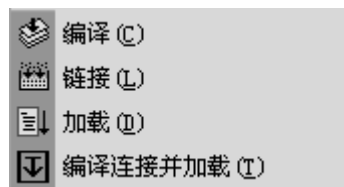
编译菜单如附图 1-4 所示。

(1) 编译 (C)：编译当前活动文档中的源程序，在源文件目录下生成目标文件。

(2) 链接 (L)：链接编译生成的目标文件，在源文件目录下生成可执行文件。

(3) 加载 (D)：把链接生成的可执行文件加载到下位机。加载成功，输出区显示“加载成功!”。

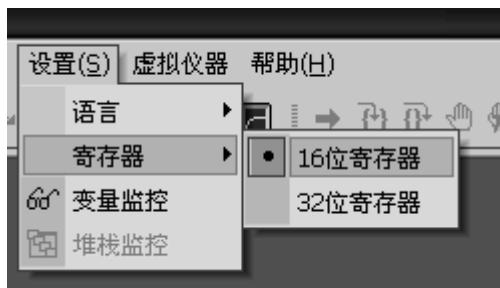
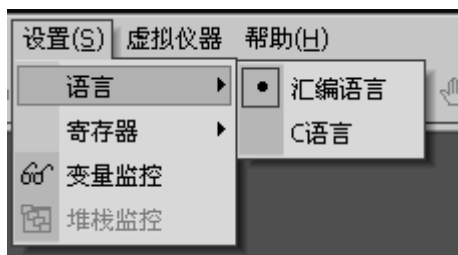
(4) 编译链接并加载 (T)：依次执行编译、链接和加载。



附图 1-4 编译菜单

### 5. 设置菜单

设置菜单如附图 1-5 所示。



附图 1-5 设置菜单

(1) 语言：设置语言环境

汇编语言：设置编译环境为汇编语言环境。此时可编辑、编译和链接 IBM-PC 汇编语言源程序。

C 语言：设置编译环境为 C 语言环境。此时可编辑、编译和链接 C 语言源程序。由于监控目前不支持浮点运算，故 C 语言程序中不应该出现浮点运算，如果 C 语言程序中出现浮点运算，

链接时会出现错误。

(2) 寄存器：设置寄存器格式

16 位寄存器：设置成 16 位寄存器，可观察到 16 位寄存器的变化。

32 位寄存器：设置成 32 位寄存器，可观察到 32 位寄存器的变化。

(3) 变量监控：加载成功后才可用此按钮。系统只能监视全局变量。在汇编语言源文件中，数据段定义的变量并不是全局变量，因此数据段定义的变量并不出现在上图所示的对话框的左边列表，要想监视这些变量，必须使它们成为全局变量，使一个变量成为全局变量的方法是用关键字 PUBLIC 在源程序的最前面声明之。

(4) 堆栈监控：用于选择是否监控堆栈。

## 6. 调试菜单项

调试菜单如附图 1-6 所示。



附图 1-6 调试菜单

(1) 设置断点/删除断点 (B)：当前光标所在的行为当前行，如果当前行无断点则在当前行设置断点，如果当前行有断点则删除当前行的断点。源程序设置的断点数不能超过 8 个。

(2) 清除所有断点 (D)：清除源程序中设置的所有断点。

(3) 设置起点 (J)：当前光标所在的行为当前行，此命令把当前行设置为程序的起点。

(4) 单步 (T)：点击此命令使程序执行一条语句，如果是函数则进入函数内部。

(5) 跳过 (O)：点击此命令使程序执行一个函数，执行后刷新所有变量和寄存器的值。

(6) 运行/运行到断点：从当前执行行开始向后运行，如果没有断点，则运行直到程序结束。如果有断点，则运行到断点后停止。

(7) 停止：发送此命令使程序停止运行，程序停止后刷新所有寄存器和变量。














(8) 固化程序：将实验程序固化到系统存储器 FLASH 中，以实现程序的脱机运行。固化程序之前，必须先将程序加到静态存储器中，然后才能进行固化程序的操作。

## 附 1.2 工具栏功能介绍

### 1. 标准工具栏

标准工具栏共有十二个按钮，如下图所示。







- (1)  按钮：用此按钮在 Wmd86 中建立一个新文档。
- (2)  按钮：用此命令在一个新的窗口中打开一个现存的文档。
- (3)  按钮：用此命令将当前活动文档保存到你当前的文件名和目录下。
- (4)  按钮：用此命令将当前被选取的数据从文档中删除并放置于剪贴板上。
- (5)  按钮：用此命令将被选取的数据复制到剪贴板上。
- (6)  按钮：用此命令将剪贴板上内容的一个副本插入到插入点处。。
- (7)  按钮：用此命令来打印一个文档。
- (8)  按钮：用此命令来撤消上一步编辑操作。
- (9)  按钮：用此命令来恢复撤消的编辑操作。
- (10)  按钮：用此按钮可显示和隐藏变量和寄存器区。
- (11)  按钮：用此按钮可显示和隐藏输出区。
- (12)  按钮：加载成功后才可用此按钮。点击此按钮，可进行全局变量监视。
- (13)  按钮：堆栈监控按钮，点击此按钮将弹出堆栈监控对话框。

### 2. 编译工具栏

编译工具栏共有五个按钮，其图如下：










- (1)  编译：编译当前活动文档中的源程序，在源文件目录下生成目标文件。
- (2)  链接：链接编译生成的目标文件，在源文件目录下生成可执行文件。
- (3)  加载：把链接生成的可执行文件加载到下位机。
- (4)  编译链接并加载：依次执行编译、链接和加载。

### 3. 调试工具栏

调试工具栏共有八个按钮，其图如下：



- (1)  设置起点：当前光标所在的行为当前行，此命令把当前行设置为程序的起点，即程序从此行开始运行，寄存器区的 CS 和 IP 的值刷新后指向此行。
- (2)  单步：点击此命令使程序执行一条语句。
- (3)  跳过：点击此命令使程序执行一个函数，执行后刷新所有变量和寄存器的值。
- (4)  设置断点/删除断点：为光标所在行设置断点或删除当前行的已有断点。源程序设置的断点数不能超过 8 个。
- (5)  清除所有断点：清除源程序中设置的所有断点。
- (6)  运行到断点/运行：从当前执行行开始向后运行，如果没有断点，则运行直到程序结束。如果有断点，则运行到断点后停止，运行到断点后再次点击此按钮，则程序从当前断点位置继续执行，直到再次遇到断点或程序结束。
- (7)  停止：发送此命令使程序停止运行，程序停止后刷新所有寄存器和变量的值。
- (8) 固化程序：程序加载到存储器后，可用固化程序功能将该程序烧写到 FLASH 存储器里面。



## 附 1.3 专用图形显示

主要用于观察“直流电机闭环调速实验”及“温度单元或电烤箱闭环温度控制实验”的响应曲线，本界面可以观察系统的给定值、反馈值及控制量之间的响应曲线关系。

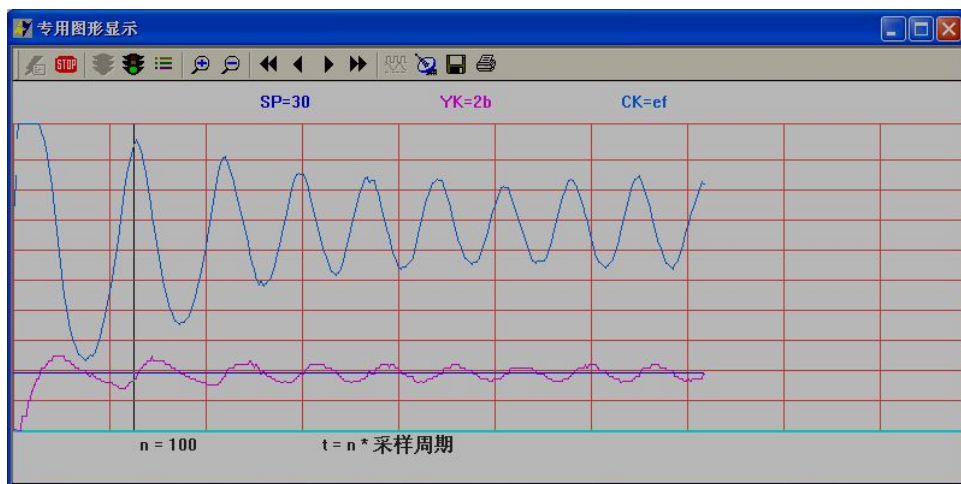
实验中给定值、反馈值都为单极性，屏幕最底端对应值为 00H，最顶端对应值为 FFH，对于时间刻度值由于采样周期不同存在以下关系：

实际时间（秒）=  $n(\text{实际刻度值}) \times \text{采样周期}$

控制量具有双极性，00H~7FH 为负值，80H~FFH 为正值。

直流电机闭环调速实验中，电机转速范围为 6 转/秒~48 转/秒。即：给定值 SPEC 范围约在 06H~30H 之间。示例程序中给定 SPEC = 30H 为 48 转/秒。TS = 14H，由于 8253 OUT2 接 IRQ6 中断为 1ms，故采样周期=14H×1ms = 0.02s。如实际刻度值  $n = 100$ ，则实际响应时间 =  $0.02 \times 100 = 2\text{s}$ 。


温度闭环控制实验中，温度单元的 7805 控制范围的最佳温度范围为 50℃~70℃，不要过高。即给定值 SPEC 范围约在 14H(20℃)~46H(70℃)之间。示例程序中 SPEC = 30H 为 48℃。TS = 64H，由于 8253 OUT2 接 IRQ6 中断为 10ms，故采样周期 =  $64\text{H} \times 10\text{ms} = 1\text{s}$ ；如实际刻度值  $n = 100$ ，则实际响应时间(秒) =  $1 \times 100 = 100\text{s}$ 。界面如下：





### 1) 显示说明


- (1) SP=30H：要求电机达到的转速值 48 转/秒（或要求达到的温度值 48℃）。
- (2) YK=2bH：运行状态下表示电机当前的转速值 43 转/秒（或当前的温度值 43℃），暂停状态下表示指定时刻电机的转速值（或指定时刻的温度值）。
- (3) CK=efH：运行状态下表示当前控制量的输出值，暂停状态下表示指定时刻控制量的输出值。控制量 CK 在正数值时转化成 PWM 输出占空比为： $\frac{\text{EFH}-80\text{H}}{\text{FFH}-80\text{H}} \times 100\% = 87.4\%$ 。


## 2) 工具栏功能简介


(1)  按钮：启动并运行程序。运行加载在下位机中的程序。“SP=”后显示的值是当前时刻系统的给定值，“YK=”后显示的值是当前系统的反馈值，“CK=”后显示的值是当前时刻系统控制量的值。


(2)  按钮：停止程序运行。使下位机中运行的程序停止。


(3)  按钮：暂停程序运行。在运行状态下使波形暂停显示并出现光标。


(4)  按钮：退出暂停状态，使波形继续显示，光标消失。


(5)  按钮：显示选择按钮，可选择性的选择要显示的波形。


(6)  按钮：放大波形。


(7)  按钮：缩小波形。

(8)  按钮：快速左移光标。在暂停状态下，使光标快速向左移动。“SP=”后显示的值是光标所在时刻系统的给定值，“YK=”后显示的值是光标所在时刻系统的反馈值，“CK=”后显示的值是光标所在时刻系统控制量的值。

(9)  按钮：左移光标。在暂停状态下，使光标向左移动。“SP=”后显示的值是光标所在时刻系统的给定值，“YK=”后显示的值是光标所在时刻系统的反馈值，“CK=”后显示的值是光标所在时刻系统控制量的值。


(10)  按钮：右移光标。在暂停状态下，使光标向右移动。“SP=”后显示的值是光标所在时刻系统的给定值，“YK=”后显示的值是光标所在时刻系统的反馈值，“CK=”后显示的值是光标所在时刻系统控制量的值。

(11)  按钮：快速右移光标。在暂停状态下，使光标快速向右移动。“SP=”后显示的值是光标所在时刻系统的给定值，“YK=”后显示的值是光标所在时刻系统的反馈值，“CK=”后显示的值是光标所在时刻系统控制量的值。

(12)  按钮：记录波形。点击此按钮，出现如下对话框：



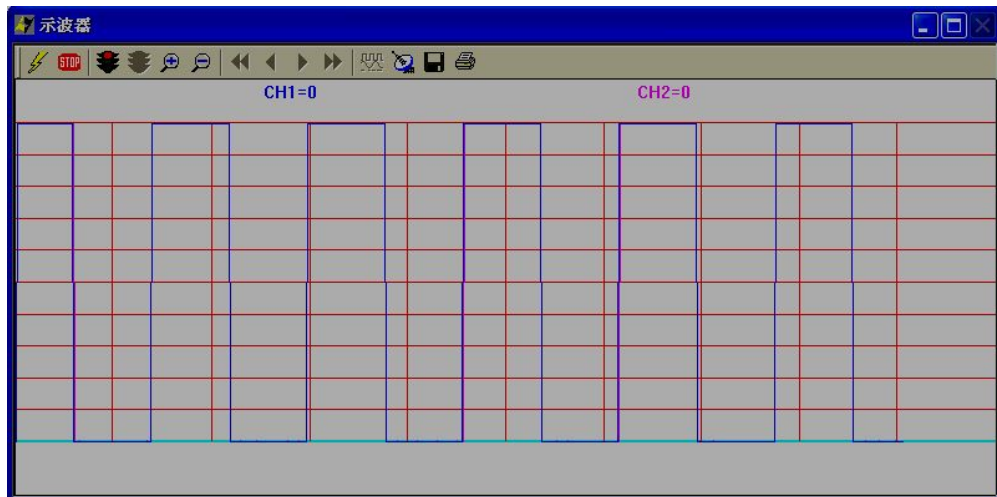
选中“图一”单选按钮，点击确定，系统会把当前时刻的波形保存到图一中，选中“图二”单选按钮，点击确定，系统会把当前时刻的波形保存到图二中，选中“图三”单选按钮，点击确定，系统会把当前时刻的波形保存到图三中。











(13)  按钮：显示保存到图一，图二和图三中的波形。

(14)  按钮：打印当前屏幕上的波形。


## 附 1.4 示波器

主要用于“8254 定时/计数器实验”、“D/A 转换实验”及“8251 串行接口实验”中波形的观察。





- (1)  按钮：启动示波器。“CH1=”、“CH2=”后分别显示游标当前位置的采样值。
- (2)  按钮：停止使下位机中运行的程序停止。
- (3)  按钮：在运行状态下使能，使波形暂停显示并出现游标。
- (4)  按钮：在暂停状态下使能，使波形继续显示，游标消失。
- (5)  按钮：放大波形。
- (6)  按钮：缩小波形。
- (7)  按钮：在暂停状态下，使游标快速向左移动。“CH1=”、“CH2=”后分别显示游标当前位置的采样值。“T = xxx”表示游标所在位置的时刻与图形最左端时刻的差值。
- (8)  按钮：在暂停状态下，使游标向左缓慢移动。
- (9)  按钮：在暂停状态下，使游标向右缓慢移动。
- (10)  按钮：在暂停状态下，使游标快速向右移动。



- (11)  按钮：点击此按钮，出现如右图所示对话框：

选中“图一”单选按钮，点击确定，系统会把当前时刻的波形保存到图一中，共可保存三副图。图形只是保存于数据缓冲中，供图形比较时使用。

- (12)  按钮：显示保存到图一，图二和图三中的波形，此时可以对几副图进行比较。

- (13)  按钮：以.bmp 格式保存当前屏幕上的波形到指定文件。

- (14)  按钮：打印当前屏幕上的波形。

附 1.5 Debug 调试命令

Wmd86 软件输出区集成有 Debug 调试，点击调试标签，进入 Debug 状态，会出现命令提示符 “>”，主要命令叙述如下：

A 进入小汇编

格式：A[段址:][偏移量]↵

A 段址:偏移量↵——从段址:偏移量构成的实际地址单元起填充汇编程序的目标代码；

A 偏移量↵——从默认的段址与给定的偏移量构成的实际地址单元起填充汇编程序目标代码；

A↵——从默认段址:默认偏移量构成的实际地址单元起填充汇编程序的目标代码；

输入上述命令后，屏幕显示地址信息，即可输入源程序。若直接回车，则退出命令。汇编程序输入时，数据一律为十六进制数，且省略 H 后缀。[m]类操作一定要在[ ]之前标注 W（字）或 B（字节）。如：MOV B[2010], AX, MOV W[2010], AX。

例：在“>”提示符下键入 A2000↵，此时默认的段址 CS 为 0000，规定偏移量 IP 为 2000，屏幕显示与操作为：

附表 1-1 小汇编操作示例

显示内容	键入内容
0000:2000	MOV AX, 1234↵
0000:2003	INC AX↵
0000:2004	DEC AX↵
0000:2005	JMP 2000↵
0000:2007	↵

B 断点设置

在系统提示符下，键入 B↵，系统提示[i]:，等待输入断点地址。输入断点地址后回车，系统继续提示[i+1]:。若直接键入回车，则结束该命令。系统允许设置最多 10 个断点，断点的清除只能是通过系统复位或重新上电来实现。例：

附表 1-2 B 命令示例

显示内容	键入内容
>	B↵
[0]:	2009↵
[1]:	↵

D 显示一段地址单元中的数据

格式：D[[段址:]起始地址, [尾地址]]↵

D 命令执行后屏幕上显示一段地址单元中的数据，在显示过程中，可用 Ctrl+S 来暂停显

示，用任意键继续；也可用 Ctrl+C 终止数据显示，返回监控状态。

E 编辑指定地址单元中的数据

格式：E[[段址:]偏移量]↵

该命令执行后，则按字节显示或修改数据，可通过“空格”键进入下一高地址单元数据的修改，使用“—”键则进入下一低地址单元进行数据的修改，并可填入新的数据来修改地址单元的内容。若输入回车，则结束 E 命令。例：

附表 1-3 E 命令示例

显示内容	键入内容
>	E3500↵
0000:3500 00_	05 空格
0000:3501 01_	空格
0000:3502 02_	—
0000:3501 01_	↵

G 运行程序

格式：G=[段址:]偏移量↵

G=[段址:]偏移量↵

其中 G 格式表示无断点连续运行程序，GB 格式表示带断点连续运行程序，连续运行过程中，当遇到断点或按下 Ctrl+C 键时，终止程序运行。

M 数据块搬移

格式：M 源地址，尾地址 目标地址↵

R 寄存器或片内 RAM 区显示与修改

格式：R↵或 R 寄存器名↵

R↵操作后，屏幕显示：CS=XXXX DS=XXXX IP=XXXX AX=XXXX F=XXXX

若需要显示并修改特定寄存器内容，则选择 R 寄存器名↵操作。如 RAX↵，则显示：AX=XXXX，键入回车键，结束该命令。若输入四位十六进制数并回车，则将该数填入寄存器 AX 中，并结束该命令。

T 单步运行指定的程序

格式：T=[段址:]偏移量]↵

每次按照指定的地址或 IP/PC 指示的地址，单步执行一条指令后则显示运行后的 CPU 寄存器情况。

U 反汇编

格式：U[[段址:]起始地址[, 尾地址]]

## 附录 2 系统实验程序清单

16 位微机原理及其程序设计实验程序清单	
文件名 (汇编语言)	实验项目
Wmd861.ASM	2.1 系统认识实验
A2-1.ASM A2-2.ASM A2-3.ASM A2-4.ASM A2-5.ASM	2.2 数制转换实验 1. 将 ASCII 码表示的十进制转换为二进制数 2. 将十进制数的 ASCII 码转换为 BCD 码 3. 将十六位二进制转换为 ASCII 码表示的十进制数 4. 十六进制数转换为 ASCII 码 5. BCD 码转换为二进制数
A3-1.ASM A3-2.ASM A3-3.ASM	2.3 运算类编程实验 1. 二进制双精度加法运算 2. 十进制的 BCD 码减法运算 3. 乘法运算
A4-1.ASM	2.4 分支程序设计实验
A5-1.ASM A5-2.ASM	2.5 循环程序设计实验 1. 计算 $S = 1 + 2 \times 3 + 3 \times 4 + \dots + N(N+1)$ 2. 求某数据区内负数的个数
A6-1.ASM A6-2.ASM	2.6 排序程序设计实验 1. 气泡排序法 2. 学生成绩名次表
A7-1.ASM A7-2.ASM	2.7 子程序设计实验 1. 求无符号字节序列中的最大值和最小值 2. 求 $N!$
A8-1.ASM	2.8 查表程序设计实验
A9-1.ASM A9-2.ASM CDISPLAY.C	2.9 输入输出程序设计实验 1. 显示程序实验 2. INT 21H 功能调用示例程序 3. C 语言显示输出实验

32 位指令及其程序设计实验程序清单	
3-2-1.ASM	3.2 32 位寄存器和 32 位指令使用基本方法, 双字排序并显示
3-2-2.ASM	3.2 32 位寄存器和 32 位指令使用基本方法, ASCII 转换 16 进制



80X86 微机接口技术及其应用实验程序清单		
文件名		实验项目
汇编语言	C 语言	
MEM.ASM	CMEM.C	4.1 静态存储器扩展实验
A82591.ASM A82592.ASM A82593.ASM	C82591.C C82592.C C82593.C	4.2 8259 中断控制实验 1. 8259 单中断实验 2. 8259 优先级实验 3. 8259 级联实验
A82371.ASM A82372.ASM A82373.ASM	C82371.C C82372.C C82373.C	4.3 DMA 特性及 8237 应用实验 1. 存储器到存储器实验 2. IO 到存储器实验 3. 存储器到 IO 实验
A82541.ASM A82542.ASM	C82541.C C82542.C	4.4 8254 定时/计数器应用实验 1. 计数应用实验 2. 定时应用实验
A82551.ASM A82552.ASM A82553.ASM	C82551.C C82552.C C82553.C	4.5 8255 并行接口实验 1. 基本输入输出实验 2. 流水灯显示实验 3. 8255 方式一输入输出实验
A82511.ASM A82512.ASM A82513.ASM A82514.ASM	C82511.C C82512.C C82513.C C82514.C	4.6 8251 串行接口应用实验 1. 数据信号的串行传输 2. 自收自发实验 3. 双机通讯实验接收程序 4. 双机通讯实验发送程序
AD0809.ASM	CADC.C	4.7 A/D 转换实验
DA08321.ASM DA08322.ASM	CDAC1.C CDAC2.C	4.8 D/A 转换实验 1. 锯齿波实验 2. 方波实验
KEYLED.ASM	CKEYLED.C	4.9 键盘扫描及显示设计实验
SOUND.ASM	CSOUND.C	4.10 电子发声设计实验
LED16.ASM	CLED16.C CHZDOT.H	4.11 点阵 LED 显示设计实验
	CLCD.C LCD.H	4.12 图形 LCD 显示设计实验
BUJIN.ASM	CBUJIN.C	4.13 步进电机实验
ZHILIU.ASM	CZHILIU.C	4.14 直流电机闭环调速实验
WENDU.ASM	CWENDU.C	4.15 温度闭环控制实验

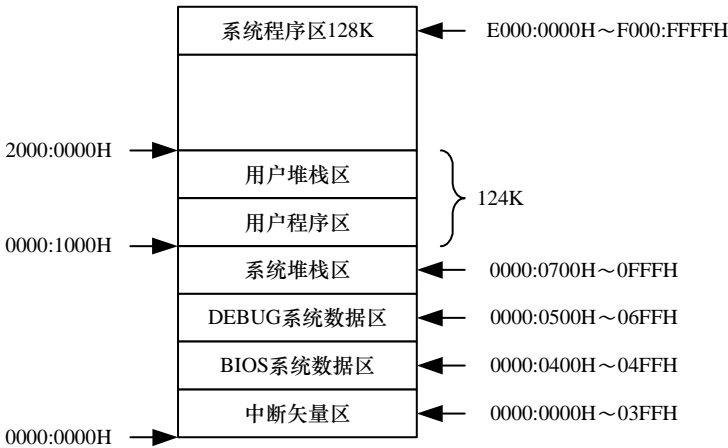
保护模式原理及其程序设计实验程序清单	
5-1.ASM	5.8 保护模式和实模式切换实验
6-1-1.ASM	6.1 全局描述符及全局描述表实验
6-1-2.ASM	6.1 局部描述符及局部描述表实验
6-2-1.ASM	6.2 任务内无特权级变换的转移实验
6-2-2.ASM	6.2 任务内无特权级变换的转移实验
6-2-3.ASM	6.2 任务内有特权级变换的转移实验
6-3-1.ASM	6.3 任务间无特权级切换实验
6-3-2.ASM	6.3 任务间有特权级切换实验
6-4-1.ASM	6.4 中断门、陷阱门实现中断/异常实验
6-4-2.ASM	6.4 任务门实现中断/异常实验
PMEM1.ASM	8.1 无分页机制的存储器扩展实验
PMEM2.ASM	8.2 具有分页机制的存储器扩展实验

附录 3 系统编程信息

附 3.1 地址分配情况

1. 系统内存分配

系统内存分配情况如附图 3-1 所示。系统内存分为程序存储器和数据存储器，程序存储器为一片 128KB 的 FLASH ROM，数据存储器为一片 128KB 的 SRAM。



附图 3-1 系统内存分配

2. 系统编址

采用内存与 IO 独立编址形式，内存地址空间和外设地址空间是相对独立的。内存地址是连续的 1M 字节，从 00000H~FFFFFH。外设的地址范围从 0000H~FFFFH，总共 64K 字节。

(1) 存储器编制

存储器编址情况见下表。

附表 3-1 存储器编址

	信号线	编址空间
系统程序存储器		E0000H~FFFFFH
系统数据存储器		00000H~1FFFFH
扩展存储器	MY0	80000H~9FFFFH
	MY1	A0000H~BFFFFH

即 SRAM 空间: 00000H~1FFFFH 共 128K  
其中: 00000H~00FFFH 为 4K 系统区  
01000H~1FFFFH 为 124K 用户使用区  
FALSH 空间: 0E0000H~0FFFFFFH 共 128K  
其中: 0E0000H~0EFFFFH 为 64K 供用户使用区  
0F0000H~0FFFFFFH 为 64K 系统监控区

(2) 输入/输出接口编址  
输入/输出接口编址见下表。

附表 3-2 输入/输出接口编址

	信号线	编址空间
主片 8259		20H、21H
从片 8259		A0H、A1H
扩展 I/O 接口	IOY0	0600H~063FH
	IOY1	0640H~067FH
	IOY2	0680H~06BFH
	IOY3	06C0H~06FFH

附 3.2 常用 BIOS 及 DOS 功能调用说明

附表 3-3 INT 03H 使用说明

入口: 无
功能: 程序终止

附表 3-4 INT 10H 使用说明

入口: AH=01H, AL=数据
功能: 写 AL 中的数据到屏上
入口: AH=06H, DS: BX=字串首址, 且字串尾用 00H 填充
功能: 显示一字串, 直到遇到 00H 为止

附表 3-5 INT 16H 使用说明

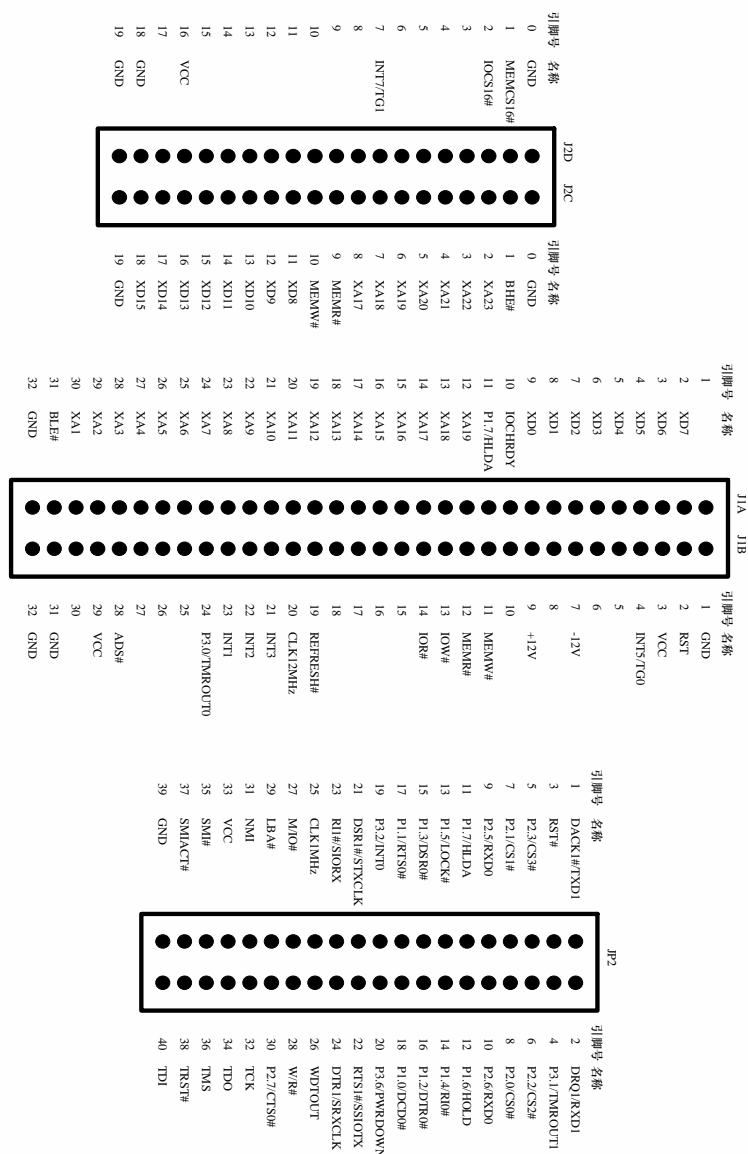
入口: AH=00H
功能: 读键盘缓冲到 AL 中, 读指针移动, ZF=1 无键值, ZF=0 有键值
入口: AH=01H
功能: 检测键盘缓冲, 并送到 AL 中, 读指针不动, ZF=1 无键值, ZF=0 有键值

附表 3-6 INT 21H 使用说明

入口：AH=00H 或 AH=4CH 功能：程序终止
入口：AH=01H 功能：读键盘输入到 AL 中并回显
入口：AH=02H，DL=数据 功能：写 DL 中的数据到显示屏
入口：AH=08H 功能：读键盘输入到 AL 中无回显
入口：AH=09H，DS:DX=字符串首地址，字符串以 '\$' 结束 功能：显示字符串，直到遇到 '\$' 为止
入口：AH=0AH，DS:DX=缓冲区首地址，(DS:DX)=缓冲区最大字符数， (DS:DX+1)=实际输入字符数，(DS:DX+2)=输入字符串起始地址 功能：读键盘输入的字符串到 DS:DX 指定缓冲区中并以回车结束

## 附录 4 I386EX 系统板引出管脚排列及名称

如图附图 4-1 所示,该图给出了 I386EX 系统板引出的管脚的排列顺序以及对应的管脚名称。其中 J1A、J1B、J2C、J2D 这四排针上的信号与 PC-104 总线标准兼容,但并未将所有的 PC-104 信号都提供,引脚名称空处为未提供的信号。JP2 这两排针上提供了 I386EX 芯片的一些功能引脚,具体引脚说明请参考器件手册,引脚 20 (P3.6/PWRDOWN) 接 GND,系统则脱离调试模式进入脱机运行状态。



附图 4-1 I386EX 系统板引出管脚排列及名称