

TD-PITD+

32 位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 2012

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，2012(C)，All Right Reserved.

32 位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

技术支持邮箱: tangdukejiao@126.com

唐都公司网址: <http://www.tangdu.com>

目 录

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.....	4
1.3 80X86 指令集.....	6
第 2 章 16 位微机原理及其程序设计实验.....	15
2.1 显示程序实验.....	15
2.2 数据传送实验.....	19
2.3 数码转换程序实验.....	23
2.4 运算类编程实验.....	31
2.5 分支程序设计实验.....	38
2.6 循环程序设计实验.....	41
2.7 子程序设计实验.....	43
第 3 章 32 位指令及其程序设计实验	48
3.1 80X86 指令及程序设计	48
3.2 32 位指令及寻址实验.....	50
第 4 章 80X86 微机接口技术及其应用实验.....	55
4.1 8/32 位I/O接口设计实验	55
4.2 地址译码电路设计实验.....	60
4.3 静态存储器扩展实验.....	63
4.4 8259 中断控制实验.....	68
4.5 8237 DMA控制实验	82
4.6 8254 定时/计数器应用实验.....	92
4.7 8255 并行接口应用实验.....	98
4.8 8251 串行接口应用实验.....	105
4.9 A/D转换实验.....	118
4.10 D/A转换实验.....	122
4.11 键盘扫描及显示设计实验.....	126
4.12 电子发声设计实验.....	131
4.13 点阵LED显示设计实验.....	136
4.14 图形LCD显示设计实验.....	142
4.15 步进电机控制实验.....	151
4.16 直流电机闭环调速实验.....	155
4.17 温度闭环控制实验.....	166

第 5 章 保护模式下的 80X86 机器组织	177
5.1 实模式和保护模式	177
5.2 寄存器组织	178
5.3 保护模式下的分段存储管理机制	180
5.4 任务管理的概念	187
5.5 任务内的控制转移	190
5.6 任务间的控制转移	193
5.7 中断/异常管理	195
5.8 基于Tddebug的保护模式程序设计	199
第 6 章 保护模式微机原理及其程序设计实验	203
6.1 描述符及描述符表实验	203
6.2 特权级变换实验	210
6.3 任务切换实验	219
6.4 中断与异常处理实验	230
第 7 章 80X86 虚拟存储器的组织及其管理	239
7.1 分段管理机制	239
7.2 分页管理机制	246
第 8 章 保护模式下的存储器扩展及其应用实验	252
8.1 保护模式下的存储器扩展实验	252
附录A Tddebug集成操作软件使用说明	255
附录B Tdpit集成操作软件使用说明	262
附录C 专用图形显示	271
附录D 示波器	273
附录E 系统实验程序清单	275

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。

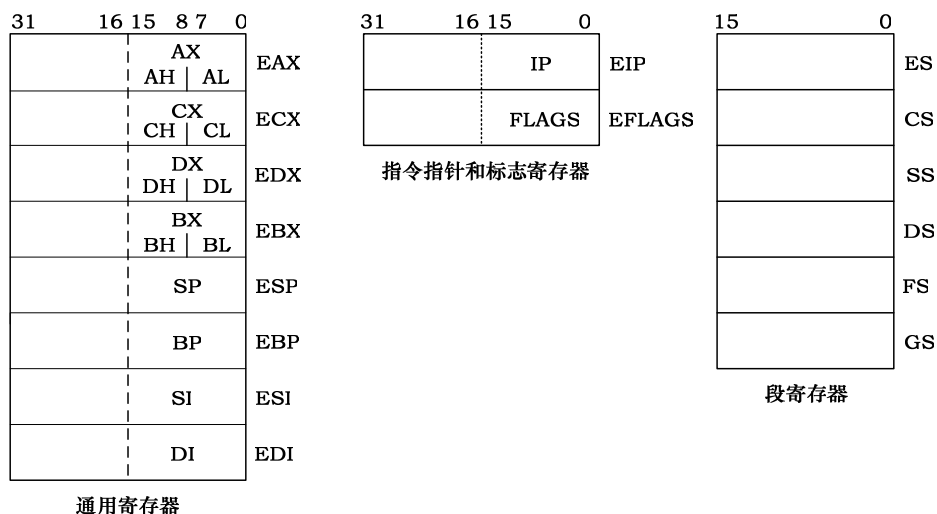


图1-1-1 80X86的部分寄存器

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 1-1-1。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 1-1-2 所示。

31	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
000000000000	I	V	V	A	V	R	0	N	IOPL		O	D	I	T	S	Z	0	A	0	P	I	C
	D	I	I	C	M	F		T			F	F	F	F	F	F		F		F	F	F

图 1-1-2 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。段寄存器指示段基址，各种寻址方式决定段内偏移。

在实模式下，段基址是 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“基地址+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 PUSHAX 和全出栈指令 POPAX

PUSHAX 指令和 POPAX 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：

PUSHAX

POPAX

PUSHAX 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 PUSHAX 执行之前的值。

POPAX 指令从堆栈弹出内容以 PUSHAX 相反的顺序送到这些通用寄存器，从而恢复 PUSHAX 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，目的操作数是 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 D 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 REP NZ/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:

IN SB ; 输入字节 BYTE
IN SW ; 输入字 WORD
IN SD ; 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:

OUT SB ; 输出字节 BYTE
OUT SW ; 输出字 WORD
OUT SD ; 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，同时掌握集成操作软件 Tdpit 的使用。

2.1 显示程序实验

2.1.1 实验目的

1. 掌握在 PC 机上以十六进制形式显示数据的方法。
2. 掌握部分 DOS 功能调用使用方法。
3. 熟悉 Windows 集成操作软件 Tdpit 的操作环境和操作方法。

2.1.2 实验设备

PC 微机一台、TD-PITD+实验系统一套。

2.1.3 实验内容及说明

一般来说，有很多程序需要显示输出提示运行的状况和结果，有的还需要将数据区中的内容显示在屏幕上。本实验要求将指定数据区的数据以十六进制数形式显示在屏幕上，并利用 DOS 功能调用完成一些提示信息的显示。通过本实验，初步掌握实验系统配套操作软件的使用。

实验中所使用 DOS 功能调用 (INT 21H) 说明如下。

(1) 显示单个字符输出

入口：AH=02H

调用参数：DL=输出字符

(2) 显示字符串

入口：AH=09H

调用参数：DS:DX=串地址，' '\$' 为结束字符

(3) 键盘输入并回显

入口：AH=01H

返回参数：AL=输出字符

(4) 返回 DOS 系统

入口: AH=4CH

调用参数: AL=返回码

程序流程图如图 2-1-1 所示。实验参考程序如下。

实验程序清单 (例程文件名: A1. ASM)

```

STACK1  SEGMENT STACK
            DW 256 DUP(?)
STACK1  ENDS
DATA SEGMENT  USE16
MES      DB  'Press any key to exit!', 0AH, 0DH, 0AH, 0DH, '$'
MES1     DB  'Show a as hex:', 0AH, 0DH, '$'
SD       DB  'a'
DATA ENDS
CODE SEGMENT  USE16
            ASSUME  CS:CODE, DS:DATA

START:     MOV  AX, DATA
            MOV  DS, AX
            MOV  DX, OFFSET MES           ;显示退出提示
            MOV  AH, 09H
            INT  21H
            MOV  DX, OFFSET MES1        ;显示字符串
            MOV  AH, 09H
            INT  21H
            MOV  SI, OFFSET SD
            MOV  AL, DS:[SI]
            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                ;显示字符
            MOV  AH, 02H
            INT  21H
            MOV  AL, DS:[SI]
            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
KEY:        MOV  AH, 1                  ;判断是否有按键按下?
            INT  16H                    ;(为观察运行结果, 使程序有控制的退出)
            JZ   KEY

```

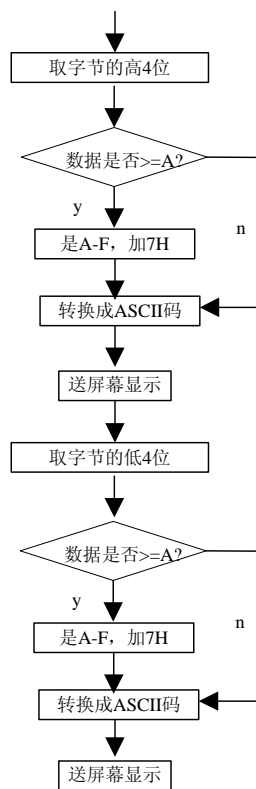


图 2-1-1 显示程序实验参考流程图

```

MOV  AX, 4C00H          ;结束程序退出
INT  21H
CODE  ENDS
      END  START

```

2.1.4 实验步骤

(1) 运行 Tdplt 集成操作软件，进入编辑调试集成环境。

(2) 根据程序设计使用语言不同，在“语言设置”菜单项中设置所使用的语言。如图 2-1-2 所示。该项一经设置，会再下次启动后仍保持不变。

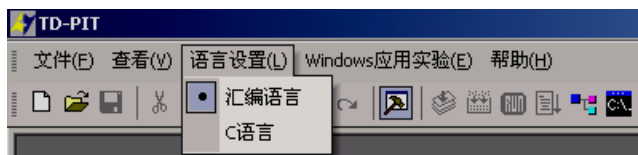


图 2-1-2 设置语言环境

(3) 开始新建文件进行编程。点击“文件”菜单项中的“新建”，可以新建一个空白文档。默认名为 Td-pit1。如图 2-1-3 所示。

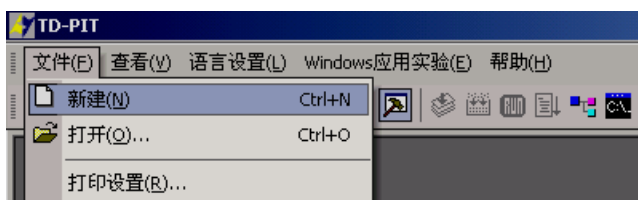


图 2-1-3 新建空白文档

(4) 编写程序，如图 2-1-4 所示，并保存，此时软件会提示输入新的文件名，输入文件名后点击保存。

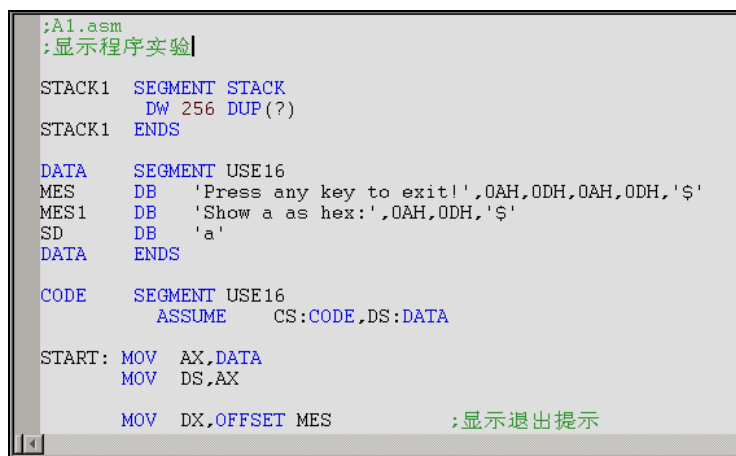




图 2-1-4 程序编辑界面

(5) 点击 ，编译文件，若程序编译无误，然后再点击 ，连接程序。编译连接成功会

在输出信息栏显示输出信息，如图 2-1-5 所示。

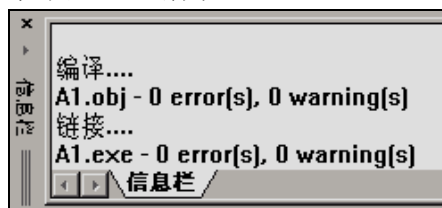


图 2-1-5 编译连接输出信息

(6) 编译连接成功后可以点击 ，运行程序，查看运行结果。

(7) 可以点击 ，调试程序，进入调试界面，进行程序的调试。

2.2 数据传送实验

2.2.1 实验目的

1. 掌握与数据有关的不同寻址方式。
2. 继续熟悉实验操作软件的环境及使用方法。

2.2.2 实验设备

PC 微机一台、TD-PITD+实验系统一套。

2.2.3 实验内容

本实验要求将数据段中的一个字符串传送到附加段中，并输出附加段中的目标字符串到屏幕上。参考实验程序如下。

实验程序清单（例程文件名：A2.ASM）


```
DDATA    SEGMENT                ;定义源数据段
    MSR    DB "HELLO, WORLD!$"
    LEN    EQU $- MSR
DDATA    ENDS
EXDA SEGMENT                ;定义附加数据段
    MSD    DB LEN DUP(?)
EXDA ENDS
MYSTACK SEGMENT STACK        ;定义堆栈段
    DW 20 DUP(?)
MYSTACK ENDS
CODE SEGMENT                ;定义代码段
ASSUME CS:CODE, DS:DDATA, ES:EXDA
START:    MOV AX, DDATA
          MOV DS, AX          ;装载数据段寄存器
          MOV AX, EXDA
          MOV ES, AX          ;装载附加数据段寄存器
          MOV SI, OFFSET MSR  ;设置 SI
          MOV DI, OFFSET MSD  ;设置 DI
          MOV CX, LEN
NEXT:     MOV AL, [SI]         ;开始传输数据
          MOV ES: [DI], AL
          INC SI
          INC DI
          DEC CX
          JNZ NEXT
```

```

        PUSH ES
        POP  DS                ;将附加段寄存器指向的段值赋给数据段寄存器
        MOV  DX,OFFSET MSD
        MOV  AH,9
        INT  21H
KEY:     MOV  AH,1              ;判断是否有按键按下?
        INT  16H              ;(为观察运行结果,使程序有控制的退出)
        JZ   KEY
        MOV  AX,4C00H         ;结束程序退出
        INT  21H
CODE     ENDS
        END      START
将程序主体部分的寄存器间接寻址方式改为相对寻址方式,则如下所示.
        MOV  BX,0
        MOV  CX,LEN
NEXT:    MOV  AL,MSR[BX]
        MOV  ES:MSD[BX],AL
        INC  BX
        LOOP NEXT

```

2.2.4 实验步骤

- (1) 运行 Tdпит 集成操作软件,编写实验程序。
- (2) 编译连接无误后,点击,进入调试环境,进行程序的调试。如图 2-2-1 所示。

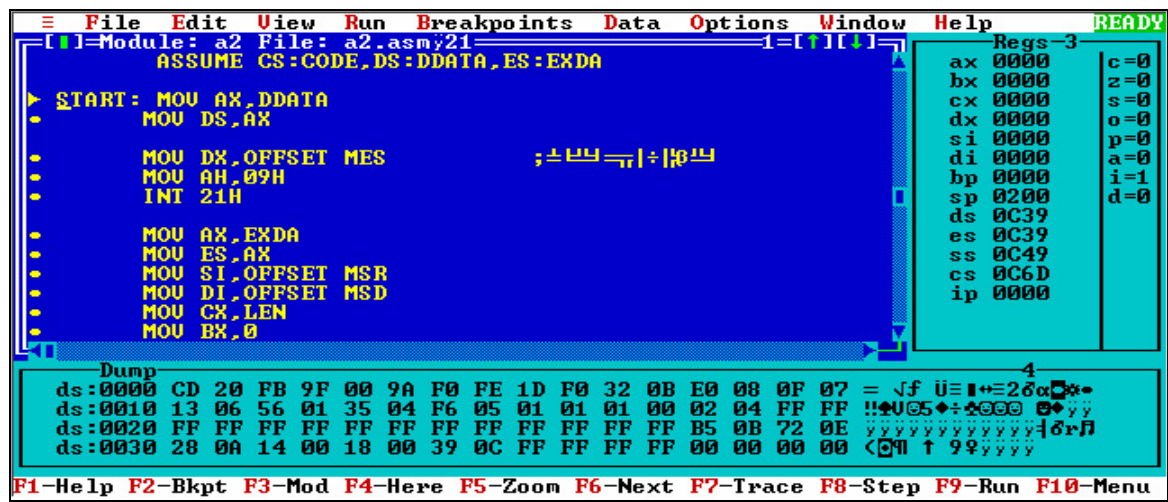


图 2-2-1 进入调试环境

(3) 按 F8 键单步运行程序,执行完 MOV DS,AX 语句后,观察 DS 寄存器中出现的段地址。激活 Dump 数据显示区,用 Ctrl+G 命令,输入要查看的数据区地址—“0C69:0000”。如图 2-2-2 所示。可以在 Dump 数据区看到 DS 数据段中 MSR 源数据串—“HELLO,WORLD!\$”。如图 2-2-3 所示。

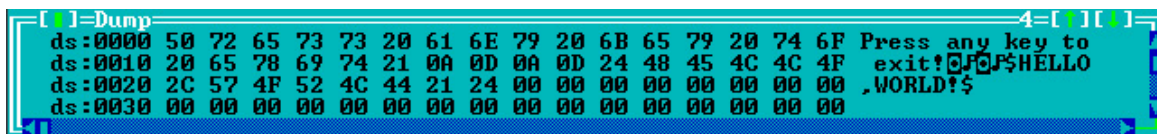


图 2-2-3 DS 源数据段数据

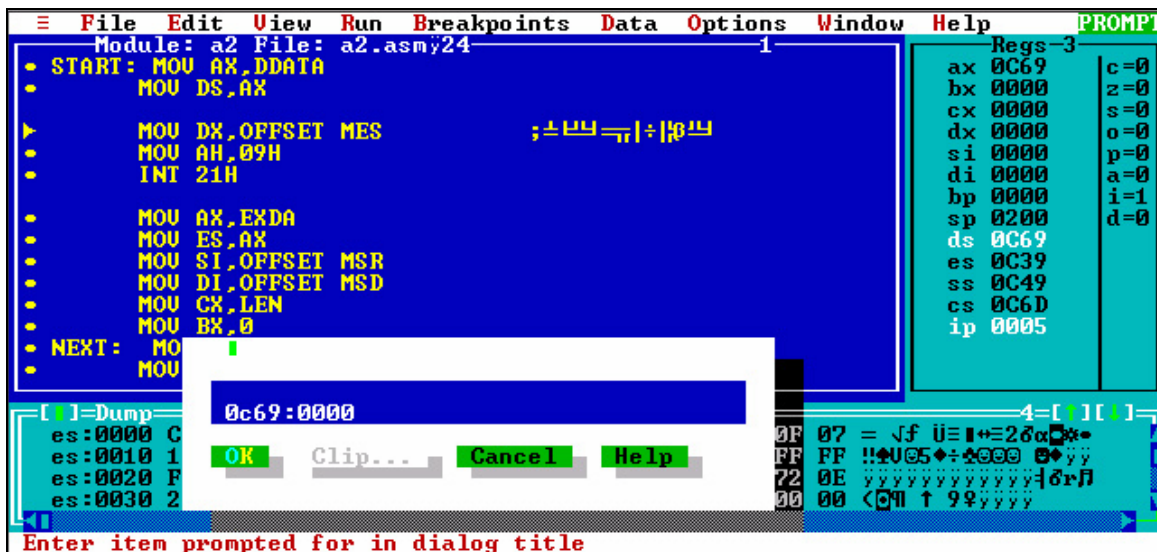


图 2-2-2 根据 DS 值查看数据段

(4) 继续单步运行程序，执行 MOV ES,AX 语句后，可以看到 ES 附加数据段出现的段地址，用同样的方法可以查看 ES:0000 的数据。如图 2-2-4 所示。



图 2-2-4 根据 ES 值查看附加数据段

(5) 数据传输还没开始进行，此时 ES 段的数据为空。继续单步执行程序，可以看到 ES 数据段逐渐被写入源数据段 DS 的数据。直到数据传输完毕，可以看到 ES 数据段中目的数据串 MSD 已经被写入了数据串—"HELLO, WORLD!\$". 如图 2-2-5 所示。

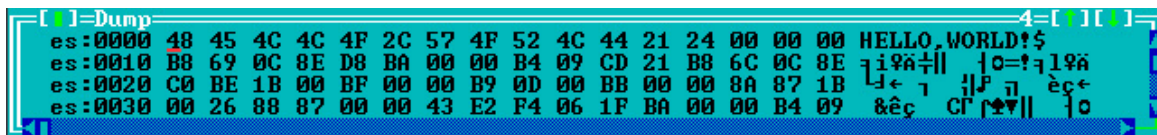


图 2-2-5 根据 ES 值查看附加数据段

(6) 可以更改程序中声明的源数据区数据，考察程序的正确性。

2.3 数码转换程序实验

2.3.1 实验目的

掌握不同进制数及编码相互转换的程序设计方法。

2.3.2 实验设备

PC 微机一台、TD-PITD+实验系统一套。

2.3.3 实验内容及说明

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他信息处理, 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, 各类数制的转换和代码的转换是必不可少的。计算机与外设间的数码对应关系如表 2-3-1 所示。数码转换关系如图 2-3-1 所示。

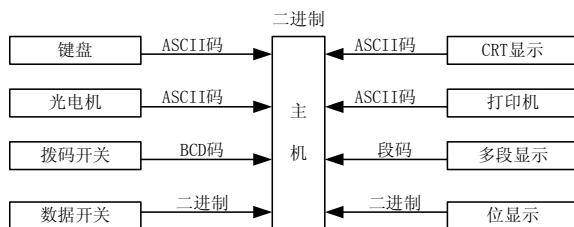


图 2-3-1 计算机与外设间的数码转换关系

表 2-3-1 数码转换对应关系

十六进制数	BCD 码	二进制机器码	ASC II 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

1. 将 ASCII 码表示的十进制数转换为二进制数

十进制数可以表示为： $D_n \times 10^n + D_{n-1} \times 10_{n-1} + \dots + D_0 \times 10^0 = D_i \times 10^i$ 其中 D_i 代表十进制数 1、2、3...9、0。

上式可以转换为： $\sum D_i \times 10^i = (((D_n \times 10 + D_{n-1}) \times 10) + D_{n-2}) \times 10 + \dots + D_1) \times 10 + D_0$

由上式可归纳十进制数转换为二进制的方法：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，则可求出二进制数结果。

本实验要求将缓冲区中的一个五位十进制数 00012 的 ASCII 码转换成二进制数，并将转换结果按位显示在屏幕上。转换过程的参考流程如图 2-3-2 所示。实验参考程序如下：

实验程序清单（例程文件名：A3-1.ASM）

```

STACK1  SEGMENT STACK
          DW 256 DUP(?)
STACK1  ENDS
DDATA   SEGMENT
MES1    DB 'The ascii code of decimal code are:$'
BUF     DB 30H, 30H, 30H, 31H, 32H
        DB 10H DUP(0)
DDATA   ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DDATA
START:  MOV  AX, DDATA
        MOV  DS, AX
        MOV  SI, OFFSET BUF
        MOV  BX, 000AH
        MOV  CX, 0004H
        MOV  AH, 00H
        MOV  AL, [SI]
        SUB  AL, 30H
A1:     IMUL  BX
        ADD  AL, [SI+01]
        SUB  AL, 30H
        INC  SI
        LOOP A1
        MOV  [SI], AX
        MOV  DX, OFFSET MES1
        MOV  AH, 09H
        INT  21H
        INC  SI
          ;显示高字节

```

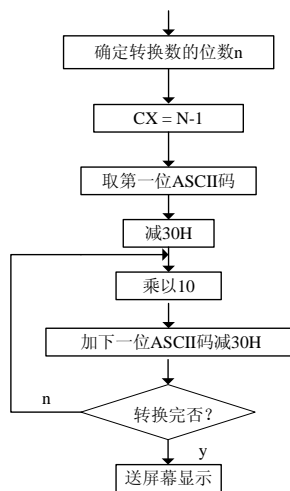


图 2-3-2 十进制 ASCII 转换为二进制数参考流程

```

        CALL SHOW
        DEC SI                      ;显示低字节
        CALL SHOW
WAIT1:  MOV AH, 1                    ;判断是否有按键按下
        INT 16H
        JZ WAIT1                    ;无按键则跳回继续等待，有则退出
        MOV AX, 4C00H
        INT 21H
SHOW    PROC NEAR
        MOV AL, DS:[SI]
        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                  ;show character
        MOV AH, 02H
        INT 21H
        MOV AL, DS:[SI]
        AND AL, 0FH                ;取低 4 位
        CMP AL, 0AH
        JB C3
        ADD AL, 07H
C3:     ADD AL, 30H
        MOV DL, AL                  ;show character
        MOV AH, 02H
        INT 21H
        RET
    ENDP
CODE ENDS
        END START

```

2. 将十进制数的 ASCII 码转换为 BCD 码

本实验要求将键盘输入的一个五位十进制数 54321 的 ASCII 码存放在数据区中，转换为 BCD 码后，并将转换结果按位分别显示于屏幕上。若输入的不是十进制数的 ASCII 码，则输出“FF”。提示：一字节 ASCII 码取其低四位即变为 BCD 码。转换部分的实验流程参见 2-3-3。实验参考程序如下：

实验程序清单（例程文件名：A3-2. ASM）

```

STACK1  SEGMENT STACK
        DW 256 DUP(?)
STACK1  ENDS
DDATA   SEGMENT
MES1 DB  ' The BCD code of decimal are:$'
BUF     DB  31H, 32H, 33H, 34H, 35H
        DB  10H DUP(0)
DDATA   ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS:DDATA

```

```

START: MOV     AX, DDATA
        MOV     DS, AX
        MOV     CX, 0005H
        MOV     DI, OFFSET BUF

```

```

A1:     MOV     BL, 0FFH
        MOV     AL, [DI]
        CMP     AL, 3AH           ;比较 AL 中的数是否是 0~9 的 ASCII 码
        JNB     A2
        SUB     AL, 30H
        JB      A2

```

```

A2:     MOV     BL, AL
        MOV     AL, BL
        MOV     [DI+05H], AL
        INC     DI
        LOOP    A1
        MOV     SI, DI
        MOV     CX, 05H
        MOV     DX, OFFSET MES1
        MOV     AH, 09H
        INT     21H

```

```

A3:     CALL    SHOW
        MOV     DL, 20H
        MOV     AH, 02H
        INT     21H
        INC     SI
        LOOP    A3

```

```

WAIT1:  MOV     AH, 1
        INT     16H
        JZ      WAIT1
        MOV     AX, 4C00H
        INT     21H

```

```

SHOW PROC NEAR
        MOV     AL, DS:[SI]
        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             ;show character
        MOV     AH, 02H
        INT     21H

```

```

        MOV     AL, DS:[SI]
        AND     AL, 0FH           ;取低 4 位
        CMP     AL, 0AH
        JB      C3
        ADD     AL, 07H
C3:     ADD     AL, 30H
        MOV     DL, AL             ;show character
        MOV     AH, 02H

```

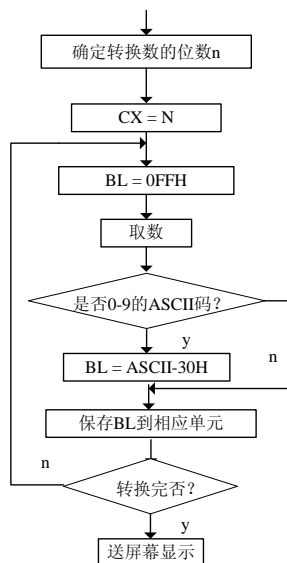


图 2-3-3 十进制 ASCII 转换为 BCD 码数参考流程

```

INT    21H
RET
ENDP
CODE ENDS
END START

```

3. 将十六进制数的 ASCII 码转换为十进制数

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： $N_D = D_4 \times 10^4 + D_3 \times 10^3 + D_2 \times 10^2 + D_1 \times 10 + D_0$ 因此，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 - D_4 ，并将它们转化为 ASCII 码。

本实验要求将缓冲区中存放的 000CH 的 ASCII 码转换成十进制数，并将转换结果显示在屏幕上。转换部分的实验流程参见 2-3-4。实验参考程序如下：

实验程序清单（例程文件名：A3-3. ASM）

```

STACK1 SEGMENT STACK
        DW 256 DUP(?)
STACK1 ENDS
DDATA SEGMENT
MES1 DB 'The ascii code of hex are:$'
BUF DB 0CH, 00H
      DB 10H DUP(0)
DDATA ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS:DDATA
START: MOV AX, DDATA
        MOV DS, AX
        MOV SI, OFFSET BUF
        MOV DX, [SI]
        MOV BX, SI
        ADD BX, 2
        ADD SI, 7
A1:     DEC SI
        MOV AX, DX
        MOV DX, 0000H
        MOV CX, 000AH
        DIV CX
        XCHG AX, DX
        ADD AL, 30H
        MOV [SI], AL
        CMP DX, 0000H
        JNE A1
A2:     CMP SI, BX
        JZ A3
        DEC SI
        MOV AL, 30H
        MOV [SI], AL
        JMP A2
A3:     MOV CX, 5
        MOV SI, OFFSET BUF

```

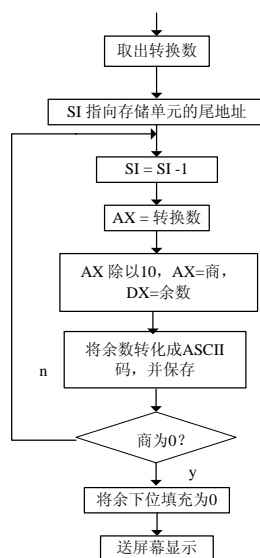


图 2-3-4 将十六进制数的 ASCII 码转换为十进制数参考流程

```

        ADD     SI, 2
        MOV     DX, OFFSET MES1
        MOV     AH, 09H
        INT     21H
A4:     CALL    SHOW
        MOV     DL, 20H
        MOV     AH, 02H
        INT     21H
        INC     SI
        LOOP    A4
WAIT1:  MOV     AH, 1           ;判断是否有按键按下
        INT     16H
        JZ      WAIT1         ;无按键则跳回继续等待，有则退出
        MOV     AX, 4C00H
        INT     21H
SHOW PROC NEAR
        MOV     AL, DS:[SI]
        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         ;show character
        MOV     AH, 02H
        INT     21H
        MOV     AL, DS:[SI]
        AND     AL, 0FH       ;取低 4 位
        CMP     AL, 0AH
        JB      C3
        ADD     AL, 07H
C3:     ADD     AL, 30H
        MOV     DL, AL         ;show character
        MOV     AH, 02H
        INT     21H
        RET
        ENDP
CODE ENDS
        END     START

```

4. BCD 码转换为二进制码

本实验要求将四个二位十进制数的 BCD 码存放在某一内存单元中，转换出的二进制数码存入其后的内存单元中，转换结束，送屏幕显示。转换部分的实验流程参见 2-3-5。实验参考程序如下：

实验程序清单（例程文件名：A3-4. ASM）

```

STACK1  SEGMENT STACK
        DW 256 DUP(?)
STACK1  ENDS
DDATA   SEGMENT

```



```

MES1      DB    'The BCD code of binary are:$'
BUF       DB    01H, 07H, 03H, 04H, 05H, 01H, 06H, 08H
          DB    10H DUP(0)
DDATA     ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DDATA
START:    MOV     AX, DDATA
          MOV     DS, AX
          MOV     CX, 0004H
          MOV     DI, OFFSET BUF
A1:       MOV     AL, [DI]
          ADD     AL, AL
          MOV     BL, AL
          ADD     AL, AL
          ADD     AL, AL
          ADD     AL, BL
          INC     DI
          MOV     AH, 00H
          ADD     AL, [DI]
          MOV     [DI+07H], AX
          INC     DI
          LOOP    A1
          MOV     DX, OFFSET MES1
          MOV     AH, 09H
          INT     21H
          MOV     CX, 04H
          MOV     DI, OFFSET BUF
          ADD     DI, 08H
A2:       MOV     AX, [DI]
          CALL    SHWORD
          MOV     DL, 20H
          MOV     AH, 02H
          INT     21H
          INC     DI
          INC     DI
          LOOP    A2
WAIT1:    MOV     AH, 1
          INT     16H
          JZ      WAIT1
          MOV     AX, 4C00H
          INT     21H
SHWORD    PROC NEAR
          MOV     BL, AH
          CALL    SHOW
          MOV     BL, AL
          CALL    SHOW
          RET
          ENDP

```

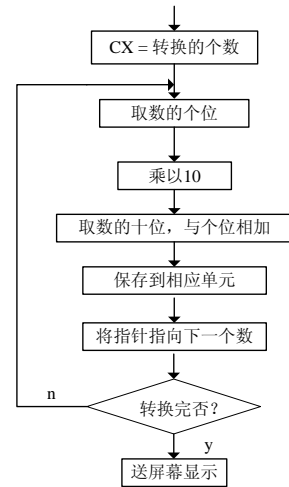


图 2-3-5 BCD 码转换为二进制码

```

SHOW PROC NEAR
    PUSH    AX
    PUSH    DX
    MOV     AL, BL
    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             ;show character
    MOV     AH, 02H
    INT     21H
    MOV     AL, BL
    AND     AL, 0FH           ;取低 4 位
    CMP     AL, 0AH
    JB      C3
    ADD     AL, 07H
C3:  ADD     AL, 30H
    MOV     DL, AL             ;show character
    MOV     AH, 02H
    INT     21H
    POP     DX
    POP     AX
    RET
    ENDP
CODE ENDS
    END     START

```

2.3.4 实验步骤

- (1) 运行 Tdptit 集成操作软件，按照各实验要求分别编写实验程序。
- (2) 对实验程序进行编译、链接。
- (3) 使用运行功能执行程序，观察运行结果。
- (4) 使用调试功能调试程序，观察在调试过程中，程序指令执行之后各寄存器及数据区的内容。
- (5) 更改数据区中的数据，反复测试，验证程序功能。

2.4 运算类编程实验

2.4.1 实验目的

1. 掌握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测试方法。

2.4.2 实验设备

PC 微机一台、TD-PITD+实验系统一套。

2.4.3 实验内容及说明

80x86 指令系统提供了实现加、减、乘、除运算的基本指令，可对表 2-4-1 所示的数据类型进行算术运算。

表 2-4-1 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

本实验要求计算 $X+Y=Z$ ，将结果 Z 输出到屏幕，其中 $X=001565A0H$ ， $Y=0021B79EH$ 。

实验利用累加器 AX ，先求低十六位和，并存入低址存储单元，后求高 16 位和，再存入高址存储单元。由于低位和可能向高位有进位，因而高位字相加语句需用 ADC 指令，则低位相加有进位时， $CF=1$ ，高位字相加时，同时加上 CF 中的 1。在 80386 以上微机中可以直接使用 32 位寄存器和 32 位加法指令完成本实验的功能。实验程序参考如下。

实验程序清单（例程文件名为：A4-1. ASM）

```
STACK1 SEGMENT STACK
        DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT
MES1 DB 'The result is:$'
XL     DW 65A0H
XH     DW 0015H
YL     DW 0B79EH
YH     DW 0021H
DATA ENDS
CODE SEGMENT
```

```

        ASSUME CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     DX, OFFSET MES1
        MOV     AH, 09H
        INT     21H
        MOV     AX, XL
        ADD     AX, YL
        MOV     BX, AX
        MOV     AX, XH
        ADC     AX, YH
        PUSH    BX
        CALL    SHWORD
        POP     BX
        MOV     AX, BX
        CALL    SHWORD
WAIT1:  MOV     AH, 1                ;判断是否有按键按下
        INT     16H
        JZ      WAIT1              ;无按键则跳回继续等待，有则退出
        MOV     AX, 4C00H
        INT     21H
SHWORD  PROC NEAR
        MOV     BL, AH
        CALL    SHOW
        MOV     BL, AL
        CALL    SHOW
        RET
        ENDP
SHOW PROC NEAR
        PUSH    AX
        PUSH    DX
        MOV     AL, BL
        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              ;show character
        MOV     AH, 02H
        INT     21H
        MOV     AL, BL
        AND     AL, 0FH            ;取低 4 位
        CMP     AL, 0AH
        JB      C3
        ADD     AL, 07H
C3:     ADD     AL, 30H
        MOV     DL, AL              ;show character
        MOV     AH, 02H
        INT     21H
        POP     DX
        POP     AX
        RET
        ENDP
CODE ENDS

```

END START

2. 十进制数的 BCD 码减法运算

本实验要求计算 $X-Y=Z$ ，其中，X、Y、Z 为 BCD 码，其中 $X=0400H$ ， $Y=0102H$ 。实验程序参考如下。

实验程序清单（例程文件名为：A4-2. ASM）

```
STACK1 SEGMENT STACK
        DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT
MES1 DB 'The result is:$'
X      DW 0400H
Y      DW 0102H
DATA ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     AH, 00H
        SAHF
        MOV     SI, OFFSET X
        MOV     AL, [SI]
        SBB     AL, [SI+02H]
        DAS
        PUSHF
        AND     AL, 0FH
        POPF
        MOV     BL, AL
        INC     SI
        MOV     AL, [SI]
        SBB     AL, [SI+02H]
        DAS
        PUSHF
        AND     AL, 0FH
        POPF
        MOV     BH, AL
        MOV     DX, OFFSET MES1
        MOV     AH, 09H
        INT     21H
        MOV     AX, BX
        CALL    SHWORD
WAIT1:  MOV     AH, 1                ;判断是否有按键按下
        INT     16H
        JZ     WAIT1                ;无按键则跳回继续等待，有则退出
        MOV     AX, 4C00H
        INT     21H
SHWORD PROC NEAR
        MOV     BL, AH
        CALL    SHOW
        MOV     BL, AL
        CALL    SHOW
        RET
        ENDP
SHOW PROC NEAR
```

```

        PUSH    AX
        PUSH    DX
        MOV     AL, BL
        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             ;show character
        MOV     AH, 02H
        INT     21H
        MOV     AL, BL
        AND     AL, 0FH           ;取低 4 位
        CMP     AL, 0AH
        JB      C3
        ADD     AL, 07H
C3:     ADD     AL, 30H
        MOV     DL, AL             ;show character
        MOV     AH, 02H
        INT     21H
        POP     DX
        POP     AX
        RET
    ENDP
CODE    ENDS
        END     START

```

3. 乘法运算

本实验要求实现十进制数的乘法，被乘数和乘数均以 BCD 码形式存放于内存中，被乘数为 54320H，乘数为 3H，运算结束后，将乘积在屏幕上显示。实验程序参考如下。

实验程序清单（例程文件名为：A4-3. ASM）

```

STACK1  SEGMENT STACK
        DW 256 DUP(?)
STACK1  ENDS
DATA     SEGMENT
MES1     DB 'The result is:$'
ERRMES   DB 'Error exist!$'
DATA1    DB 00H, 02H, 03H, 04H, 05H
DATA2    DB 03H
RESULT   DB 06H DUP(0)
DATA     ENDS
CODE     SEGMENT
        ASSUME CS:CODE, DS:DATA
START:   MOV     AX, DATA
        MOV     DS, AX
        MOV     SI, OFFSET DATA2
        MOV     BL, [SI]
        AND     BL, 0FH
        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
        ADD    AL, [DI+05H]
        AAA
        MOV    [DI+05H], AL
        DEC    DI
        MOV    [DI+05H], AH
        LOOP   A1
        MOV    DX, OFFSET MES1
        MOV    AH, 09H
        INT    21H
        MOV    CX, 06H
        MOV    SI, OFFSET RESULT
A2:    CALL   SHOW
        MOV    DL, 20H
        MOV    AH, 02H
        INT    21H
        INC    SI
        LOOP   A2
WAIT1: MOV    AH, 1                ;判断是否有按键按下
        INT    16H
        JZ     WAIT1              ;无按键则跳回继续等待，有则退出
        MOV    AX, 4C00H
        INT    21H
ERROR: MOV    DX, OFFSET ERRMES
        MOV    AH, 09H
        INT    21H
        MOV    AX, 4C00H
        INT    21H
SHOW PROC NEAR
        MOV    AL, DS:[SI]
        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              ;show character
        MOV    AH, 02H
        INT    21H
        MOV    AL, DS:[SI]
        AND    AL, 0FH            ;取低 4 位
        CMP    AL, 0AH
        JB     C3
        ADD    AL, 07H
C3:    ADD    AL, 30H
        MOV    DL, AL              ;show character
        MOV    AH, 02H
        INT    21H
        RET

```

```

        ENDP
CODE    ENDS
        END    START

```

4. 用减奇数开平方运算

80x86 指令系统中有乘法指令但没有开平方指令，因此，开平方运算是通过程序来实现的。用减奇数法可求得近似平方根，获得平方根的整数部分。我们知道，N个自然数中的奇数之和等于 N^2 ，即：

$$1+3+5=9=3^2$$

$$1+3+5+7=16=4^2$$

$$1+3+5+7+9+11+13+15=64=8^2$$

若要做 S 的开方运算，那么就可以从 S 中逐次减去自然数中的奇数 1, 3, 5, 7..., 一直进行到相减数为 0 或不够减下一个自然数的奇数为止，然后统计减去自然数的奇数个数，它就是 S 的近似平方根。本实验要求利用减奇法计算 0040H 的开平方值，并将运算结果显示在屏幕上。实验程序参考如下。

实验程序清单（例程文件名为：A4-4. ASM）

```

STACK1  SEGMENT STACK
        DW 256 DUP(?)
STACK1  ENDS
DDATA   SEGMENT
MES1 DB 'The square root of $'
MES2 DB ' is:$'
NUMB DW 0040H
DDATA   ENDS
CODE    SEGMENT
        ASSUME CS: CODE, DS: DDATA
START:  MOV     AX, DDATA
        MOV     DS, AX
        MOV     DX, OFFSET MES1
        MOV     AH, 09H
        INT     21H
        MOV     SI, OFFSET NUMB
        MOV     AX, [SI]
        CALL    SHWORD
        MOV     DX, OFFSET MES2
        MOV     AH, 09H
        INT     21H
        MOV     AX, [SI]
        MOV     CL, 00H
        MOV     DX, 0001H
A1:     SUB     AX, DX
        JB      A2
        INC     CL
        ADD     DX, 02H
        JMP     A1
A2:     MOV     BL, CL
        CALL    SHOW
WAIT1:  MOV     AH, 1                ;判断是否有按键按下
        INT     16H
        JZ      WAIT1              ;无按键则跳回继续等待，有则退出

```



```

        MOV     AX, 4C00H
        INT     21H
SHWORD  PROC NEAR
        MOV     BL, AH
        CALL    SHOW
        MOV     BL, AL
        CALL    SHOW
        RET
        ENDP
SHOW PROC NEAR
        PUSH    AX
        PUSH    DX
        MOV     AL, BL
        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             ;show character
        MOV     AH, 02H
        INT     21H
        MOV     AL, BL
        AND     AL, 0FH           ;取低 4 位
        CMP     AL, 0AH
        JB      C3
        ADD     AL, 07H
C3:      ADD     AL, 30H
        MOV     DL, AL             ;show character
        MOV     AH, 02H
        INT     21H
        POP     DX
        POP     AX
        RET
        ENDP
CODE    ENDS
        END     START

```

2.4.4 实验步骤

- (1) 运行 Tdplt 集成操作软件，按各实验要求编写实验程序。
- (2) 分别对实验程序进行编译、链接。
- (3) 使用运行功能运行程序，观察运行结果。
- (4) 使用调试功能调试程序，观察在调试过程中，各运算指令执行后，各寄存器、标志位及数据区内容的变化。
- (5) 更改数据区中的数据，反复测试，验证程序功能。

2.5 分支程序设计实验

2.5.1 实验目的

掌握分支程序的设计方法。

2.5.2 实验设备

PC 微机一台、TD-PITD+实验系统一套。

2.5.3 实验内容及说明

程序有顺序、循环、分支和子程序四种结构形式，分支结构的示意图如图 2-5-1 所示。本实验要求参考图 2-5-2 流程，通过求无符号字节序列中的最大值和最小值来反映分支程序的结构形

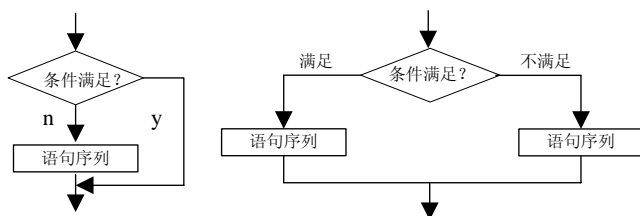


图 2-5-1 分支结构示意图

式。实验可以使用 BH, BL 作为暂存现行的最大值和最小值，且在程序的初始，将 BH 和 BL 初始化为首字节的内容，然后进入循环操作。在循环操作中，依次从字节序列中逐个取出一个字节的的内容与 BH, BL 进行比较，若取出的字节内容比 BH 的内容大或比 BL 中的内容小，则修改之。当循环结束操作时，将 BH, BL 分别送屏幕显示。参考实验程序如下。

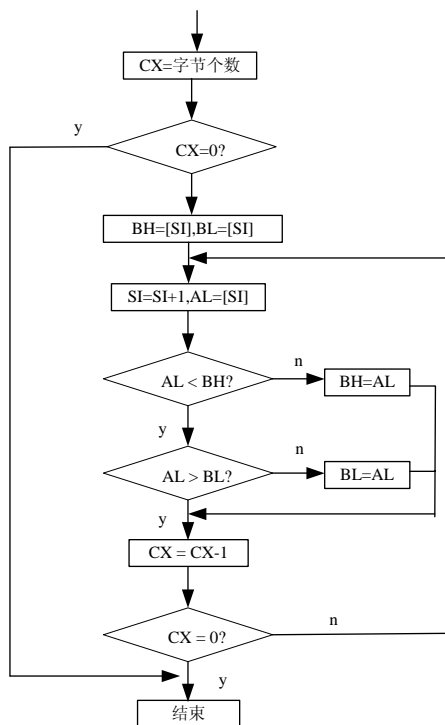


图 2-5-2 分支程序实验流程图

实验程序清单（例程文件名为 A5. ASM）

```

STACK1  SEGMENT STACK
        DW 256 DUP(?)
STACK1  ENDS
DDATA    SEGMENT
MES1 DB 'The least  number is:$'
MES2    DB 0AH, 0DH, 'The largest number is:$'
NUMB DB 0D9H, 07H, 8BH, 0C5H, 0EBH, 04H, 9DH, 0F9H
DDATA    ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS:DDATA
START: MOV     AX, DDATA
        MOV     DS, AX
        MOV     SI, OFFSET NUMB
        MOV     CX, 0008H
        JCXZ    A4
        MOV     BH, [SI]
        MOV     BL, BH
A1:     LODSB
        CMP     AL, BH
        JBE     A2
        MOV     BH, AL
        JMP     A3
A2:     CMP     AL, BL
        JAE     A3
        MOV     BL, AL
A3:     LOOP    A1
A4:     MOV     DX, OFFSET MES1
        MOV     AH, 09H
        INT     21H
        MOV     AL, BL
        AND     AL, 0F0H
        SHR     AL, 4
        CMP     AL, 0AH
        JB      C2
        ADD     AL, 07H
C2:     ADD     AL, 30H
        MOV     DL, AL
        MOV     AH, 02H
        INT     21H
        MOV     AL, BL
        AND     AL, 0FH
        CMP     AL, 0AH
        JB      C3
        ADD     AL, 07H
C3:     ADD     AL, 30H
        MOV     DL, AL
        MOV     AH, 02H
        INT     21H

```

```
        MOV     DX, OFFSET MES2
        MOV     AH, 09H
        INT     21H
        MOV     AL, BH
        AND     AL, 0F0H
        SHR     AL, 4
        CMP     AL, 0AH
        JB      C22
        ADD     AL, 07H
C22:    ADD     AL, 30H
        MOV     DL, AL
        MOV     AH, 02H
        INT     21H
        MOV     AL, BH
        AND     AL, 0FH
        CMP     AL, 0AH
        JB      C33
        ADD     AL, 07H
C33:    ADD     AL, 30H
        MOV     DL, AL
        MOV     AH, 02H
        INT     21H
WAIT1:  MOV     AH, 1
        INT     16H
        JZ      WAIT1
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

2.5.4 实验步骤

- (1) 运行 Tdpt 集成操作软件, 根据实验要求编写程序, 在数据段声明 8 个的数据: 0D9H, 07H, 8BH, 0C5H, 0EBH, 04H, 9DH, 0F9H。
- (2) 对实验程序进行编译、链接。
- (3) 使用运行命令运行程序, 观察运行结果。
- (4) 更改 8 个数据的值, 考察程序运行结果是否正确。

2.6 循环程序设计实验

2.6.1 实验目的

掌握循环程序的设计方法。

2.6.2 实验设备

PC 微机一台、TD-PITD+实验系统一套。

2.6.3 实验内容及说明

本实验要求通过求某数据区内负数的个数来表现循环程序的结构形式。要求实验程序在数据区中存放一组数据，为统计负数的个数，逐个判断区内的数据，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。循环程序的结构示意如图 2-6-1 所示。实验程序参考如下。

实验程序清单（例程文件名为 A6.ASM）

```
STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
DDATA SEGMENT
NUMB DB 12H, 88H, 82H, 89H, 33H, 90H, 01H, 10H, 0BDH, 01H
MES1 DB 'The number of negative is:$'
DDATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DDATA
START: MOV AX, DDATA
    MOV DS, AX
    MOV DI, OFFSET NUMB
    XOR BH, BH
    MOV CX, 10D
A1:    MOV AL, [DI]
    TEST AL, 80H
    JE A2
    INC BL
A2:    INC DI
    LOOP A1
    MOV DX, OFFSET MES1
    MOV AH, 09H
    INT 21H
```

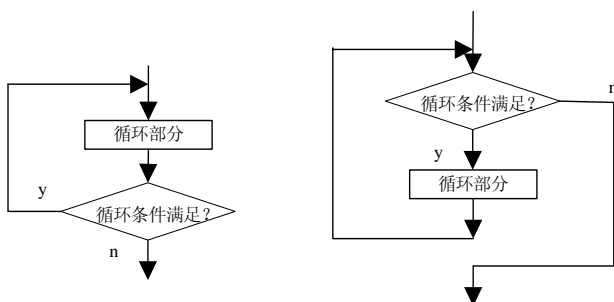


图 2-6-1 循环结构示意图

```
        MOV     AL, BL
        AND     AL, 0F0H
        SHR     AL, 4
        CMP     AL, 0AH
        JB      C2
        ADD     AL, 07H
C2:     ADD     AL, 30H
        MOV     DL, AL
        MOV     AH, 02H
        INT     21H
        MOV     AL, BL
        AND     AL, 0FH
        CMP     AL, 0AH
        JB      C3
        ADD     AL, 07H
C3:     ADD     AL, 30H
        MOV     DL, AL
        MOV     AH, 02H
        INT     21H
WAIT1:  MOV     AH, 1
        INT     16H
        JZ      WAIT1
        MOV     AX, 4C00H
        INT     21H
CODE    ENDS
        END     START
```

2.6.4 实验步骤

- (1) 运行 Tdplt 集成操作软件，根据实验要求编写程序。在数据段声明 10 个数据：12H，88H，82H，89H，33H，90H，01H，10H，0BDH，01H。
- (2) 对实验程序进行编译、链接。
- (3) 使用运行命令运行程序，观察运行结果。
- (4) 更改数据区中的数据，反复测试，验证程序功能。

2.7 子程序设计实验

2.7.1 实验目的

1. 掌握子程序的定义调用方法。
2. 掌握系统功能调用程序的使用和编写方法。

2.7.2 实验设备

PC 微机一台、TD-PITD+实验系统一套。

2.7.3 实验内容及步骤

在汇编程序设计中，用户通常会将常用的具有特定功能的程序段编制成子程序使用。一般过程定义伪操作的格式如下：

```
procedure name PROC Attribute
    .....
procedure name ENDP
```

其中 Attribute 是指类型属性，可以是 NEAR 或 FAR，调用程序和过程在同一个代码段中使用 NEAR 属性，不在同一个代码段中，使用 FAR。

1. 数据移动实验

本实验要求将指定数据区的数据搬移到另一个数据区，并通过子程序调用的方法将搬移的数据显示在屏幕上。

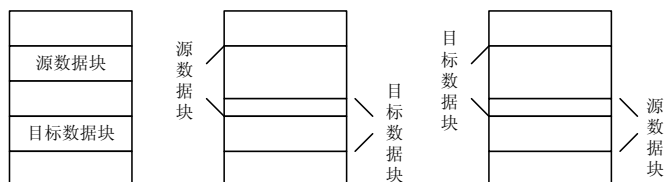


图 2-7-1 源数据块和目标数据块在存储器中的位置示意

源数据块和目标数据块在存储中的位置可能有三种情况，如图 2-7-1 所示。对于两个数据块分离的情况，数据的传送从数据块的首地址开始，或者从数据块的末地址开始均可。但对于有部分重叠的情况，则要加以分析，否则重叠部分会因搬移而遭到破坏。所以搬移过程可以通过以下两个方式完成：当源数据块首地址>目标块首址时，从数据块的首地址开始传送数据；当源数据块首地址<目标块首址时，从数据块的末地址开始传送数据。实验程序参考如下。

实验程序清单（例程文件名为 A7-1. ASM）

```

STACK1  SEGMENT STACK
        DW 256 DUP(?)
STACK1  ENDS
DDATA    SEGMENT
MES1     DB  'The data in buf2 are:', 0AH, 0DH, '$'
BUF1 DB  11H, 22H, 33H, 44H, 55H, 66H, 77H, 88H, 99H, 0AAH, 0BBH, 0CCH, 0DDH, 0EEH, 0FFH, 00H
BUF2 DB  20H DUP(0)
DDATA    ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS:DDATA
START:  MOV  AX, DDATA
        MOV  DS, AX
        MOV  CX, 0010H
        MOV  SI, OFFSET BUF1
        MOV  DI, OFFSET BUF2
        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  DX, OFFSET MES1
        MOV  AH, 09H
        INT  21H
        MOV  CX, 10H
        MOV  SI, OFFSET BUF2
A4:     CALL SHOW
        INC  SI
        MOV  DL, 20H
        MOV  AH, 02H
        INT  21h
        LOOP A4
WAIT1:  MOV  AH, 1                ;判断是否有按键按下
        INT  16H
        JZ   WAIT1                ;无按键则跳回继续等待，有则退出
        MOV  AX, 4C00H

```



```

        INT     21H
SHOW PROC NEAR
        MOV     AL, DS:[SI]
        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             ;show character
        MOV     AH, 02H
        INT     21H
        MOV     AL, DS:[SI]
        AND     AL, 0FH           ;取低 4 位
        CMP     AL, 0AH
        JB      C3
        ADD     AL, 07H
C3:     ADD     AL, 30H
        MOV     DL, AL             ;show character
        MOV     AH, 02H
        INT     21H
        RET
    ENDP
CODE ENDS
        END START

```

具体实验步骤如下所述。

(1) 运行 Tdpt 集成操作软件，根据实验要求编写程序。在数据段声明 16 字节的数据：11H, 22H, 33H, 44H, 55H, 66H, 77H, 88H, 99H, 0AAH, 0BBH, 0CCH, 0DDH, 0EEH, 0FFH, 00H。

(2) 对实验程序进行汇编、链接。

(3) 使用运行命令运行程序，观察运行结果。

(4) 使用调试命令调试程序，观察源数据区数据，并在程序运行结束后观察目的数据区数据，看传输是否正确。

(5) 更改数据区中的数据，考察程序的正确性。

2. 数码转换及显示实验

有时当系统运行或者程序运行期间在遇到某些特殊情况时，需要计算机自动执行一组专门的例行程序来进行中断处理。这段例程称为中断子程序。中断分为内部中断和外部中断两类。象除法错或者程序中为了作某些处理而设置的中断指令等属于内部中断。外部中断则主要用来处理 I/O 设备与 CPU 之间的通信。

在汇编语言程序设计中系统功能调用程序，只需要通过 MOV 指令，将中断参数装到与此有关的寄存器中，然后用 INT 指令调用所需中断。如果希望中断处理程序是用户自己编写的一段程序，则需要修改对应中断的中断处理程序入口。微机系统中可以使用 0-255 共 256 个中断。当 80x86 系统工作于实模式的时候，内存的 000H - 3FFH 被用于作为中断向量表，向量表中包含了 256 个中断的中断子程序入口（中断向量地址），向量表内容如图 2-7-2 示。

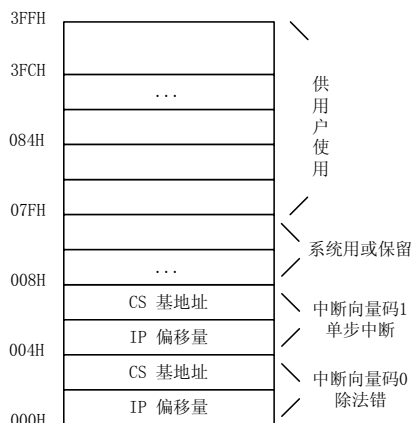


图 2-7-2 中断向量表

本实验要求利用 47H 号中断将一组字符转换成十六进制数码，并在屏幕上显示出来。实验程序参考如下。

实验程序清单（例程文件名为 A7-2. ASM）

```

STACK1  SEGMENT STACK
        DW 256 DUP(?)
STACK1  ENDS
DDATA   SEGMENT
CSBAK   DW  ?
IPBAK   DW  ?
MKBAK   DB  ?
SW       DW  ?
MES1    DB  'The data in buf1 are:', 0AH, 0DH, '$'
BUF1    DB  11H, 22H, 33H, 44H, 55H, 66H, 77H, 88H, 99H, 0AAH, 0BBH, 0CCH, 0DDH, 0EEH, 0FFH, 00H
DDATA   ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DDATA
START   PROC FAR
        MOV     AX, DDATA
        MOV     DS, AX
        MOV     AX, 0           ;修改 47H 号中断的中断矢量
        MOV     ES, AX
        MOV     DI, 4*47H
        MOV     AX, ES:[DI]
        MOV     IPBAK, AX      ;保存原有 IP
        MOV     AX, OFFSET MYINT ;修改为用户自定义中断入口
        CLD
        STOSW
        MOV     AX, ES:[DI]    ;保存原有 CS
        MOV     CSBAK, AX
        MOV     AX, SEG MYINT
        STOSW
        MOV     DX, OFFSET MES1 ;显示提示信息
        MOV     AH, 09H
        INT     21H
        MOV     SI, OFFSET BUF1 ;显示 BUF1 中的内容
        MOV     CX, 10H
        INT     47H

```

```

        MOV     AX, 0                ;恢复系统中断矢量
        MOV     ES, AX
        MOV     DI, 4*47H
        MOV     AX, IPBAK
        CLD
        STOSW
        MOV     AX, CSBAK
        STOSW
WAIT1:   MOV     AH, 1                ;判断是否有按键按下
        INT     16H
        JZ      WAIT1               ;无按键则跳回继续等待，有则退出
        MOV     AX, 4C00H           ; 返回 dos
        INT     21H
        RET
        ENDP
MYINT    PROC    FAR                ;自定义显示中断，入口参数为 BL
        PUSH    AX
        PUSH    DX
C1:       MOV     AL, [SI]
        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              ;显示字符
        MOV     AH, 02H
        INT     21H
        MOV     AL, [SI]
        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
        INC     SI
        LOOP    C1
        POP     DX
        POP     AX
        IRET                          ;中断返回
        ENDP
CODE     ENDS
        END     START

```

具体实验步骤如下。

- (1) 运行 Tdpit 集成操作软件，根据实验要求编写程序。
- (2) 对实验程序进行编译、链接。
- (3) 使用运行命令运行程序，观察运行结果。
- (4) 更改数据区中的数据，反复测试，验证程序功能。

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 微机一台、TD-PITD+实验系统一套。

3.2.3 实验内容及步骤

实验一：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. 实验程序清单（3-2-1.asm）

```
.386p
STACK1 SEGMENT STACK USE16
    DB 64 DUP(?)
STACK1 ENDS
DATA SEGMENT USE16
MES1 DB 'The array is:$'
MES2 DB 'After sort:$'
DATA1 DD 110015H, 111101D8H, 22110002H, 111a0004H, 1d110009H, 111f044H, 11d10203H, 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
        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
WAIT1:  MOV     AH, 1                      ;判断是否有按键按下
        INT     16H
        JZ      WAIT1                    ;无按键则跳回继续等待，有则退出
        MOV     AX, 4C00H
        INT     21H
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
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
C4:     LOOP    C1
        RET
SAHEX   ENDP
KENTER  PROC NEAR
        MOV     DL, 0AH
        MOV     AH, 02H
        INT     21H
        MOV     DL, 0DH
        MOV     AH, 02H
        INT     21H
        RET
KENTER  ENDP
CODE    ENDS
        END     START

```

2. 实验步骤

- (1) 运行 Tdplt 集成操作软件，根据实验要求编写程序。
- (2) 对实验程序进行编译、链接。
- (3) 使用运行命令运行程序，观察运行结果。
- (4) 使用调试命令调试程序，详细观察程序运行过程。
- (5) 更改数据区中的数据，考察程序的正确性。

实验二：本实验要求将一组 ASCII 字符转换成十六进制数码，并在屏幕上显示出来。要求使用 32 位寄存器、32 位的指令和寻址方式。如将字符串 “This is tangdu speaking!” 进行转换，应转换成：54H、68H、69H、73H、20H、69H、73H、20H、74H、61H、6EH、67H、64H、75H、20H、73H、70H、65H、61H、6BH、69H、6EH、67H、21H。实验程序参考如下。

1. 实验程序清单（3-2-2.asm）

```

.386
STACK1  SEGMENT STACK    USE16
        DB     64    DUP(?)
STACK1  ENDS

DATA SEGMENT    USE16
MES0     DB     'This is tangdu speaking!$'
MES1 DB     'Show this sentence as hex:$'
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
WAIT1:  MOV     AH, 1                     ;判断是否有按键按下
        INT     16H
        JZ      WAIT1                     ;无按键则跳回继续等待，有则退出
        MOV     AX, 4C00H
        INT     21H
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     BYTE PTR GS:[ESI], 24H    ;在串尾加上$符
        POPAD                                ;弹出所有寄存器值
        RET
SAHEX    ENDP
KENTER   PROC NEAR
        MOV     DL, 0AH
        MOV     AH, 02H
        INT     21H
        MOV     DL, 0DH
        MOV     AH, 02H
        INT     21H
        RET
KENTER   ENDP
CODE     ENDS
        END     START
```

2. 实验步骤

- (1) 运行 Tdplt 集成操作软件，根据实验要求编写程序。
- (2) 对实验程序进行编译、链接。
- (3) 使用运行命令运行程序，观察运行结果是否正确。
- (4) 使用调试命令调试程序，详细观察程序运行过程。
- (5) 更改数据区中的数据，反复测试，验证程序功能。

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 8/32 位 I/O 接口设计实验

4.1.1 实验目的

- (1) 掌握基本 I/O 接口电路的设计方法。
- (2) 熟悉 I/O 操作指令及 8 /32 位 I/O 端口的操作方法。

4.1.2 实验设备

PC 微机一台、TD-PITD+实验系统一套。

4.1.3 实验内容

(1) 利用一组三态缓冲器 245、锁存器 374 或 574 构成的 8 位 I/O 接口,实现微机对外部输入数据的读取和对输出数据的输出。用拨动开关和数据灯作为输入和输出显示设备,将读到开关的数据显示在数据灯上。

(2) 利用四组三态缓冲器 245、锁存器 374 或 574 构成的 32 位的 I/O 接口,按照 32 位的 I/O 操作方式,操作点阵 LED 显示单元的 16 行×16 列点阵。

4.1.4 实验原理

1. 输入接口设计

输入接口一般用三态缓冲器实现,外部设备输入数据通过三态缓冲器,通过数据总线传送给微机系统。74LS245 是一种 8 通道双向的三态缓冲器,其管脚结构如图 4-1-1 所示。DIR 引脚控制缓冲器数据方向,DIR 为 1 表示数据由 A[7:0]至 B[7:0],DIR 为 0 表示数据由 B[7:0]至 A[7:0]。G 引脚为缓冲器的片选信号,低电平有效。

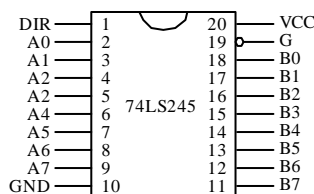


图 4-1-1 74LS245 双向三态缓冲器管脚图

2. 输出接口设计

输出接口一般用锁存器实现，从总线送出的数据可以暂存在锁存器中。74LS374/74LS574 是一种 8 通道上沿触发锁存器。74LS574 管脚结构如图 4-1-2 所示。D[7:0]为输入数据线，Q[7:0]为输出数据线。CLK 引脚为锁存控制信号，上升沿有效。当上升沿到时，输出数据线锁存输入数据线上的数据。OE 引脚为锁存器的片选信号，低电平有效。

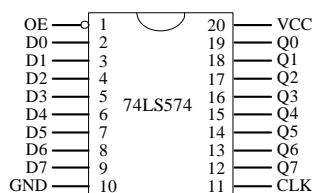


图 4-1-2 74LS574 上沿触发锁存器管脚图

3. 8 位 I/O 接口设计

用一组 74LS245 和 74LS374/574 可以构成一个 8 位的 I/O 接口电路，既实现数据的输入又实现数据的输出，输入输出可以占用同一个端口。是输入还是输出用总线读写信号来区分。总线读信号 IOR 和片选信号 CS 相“或”来控制输入接口 74LS245 的使能信号 G。总线写信号 IOW 和片选信号 CS 相“或”来控制输出接口 74LS574 的锁存信号 CLK。实验系统中基本 I/O 接口单元就实现了这种的电路，8 位 I/O 电路连接如图 4-1-3 所示。

IN AL, DX ; 将 IA[7:0]连接设备的 8 位数据通过数据总线 D[7:0]输入到 AL。
OUT DX, AL ; 将 AL 中的数据通过数据总线 D[7:0]输出到 OA[7:0]连接的设备。

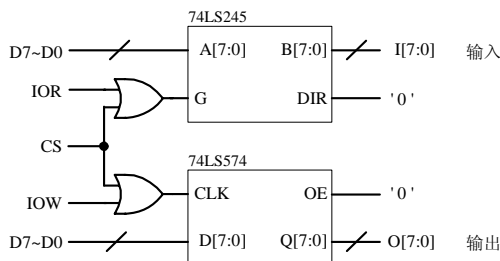


图 4-1-3 用 74LS245 和 74LS574 组成的 8 位 I/O 接口电路

4. 32 位 I/O 接口设计

用四组 8 位的 I/O 接口电路可以构成一个 32 位的 I/O 接口电路，可以一次进行 32 位数据宽度的 I/O 操作。I/O 读、写、片选信号对输入输出的控制基本和 8 位 I/O 接口电路相同，但是，对于 32 位数据总线，每个字节都对应着一位字节使能信号，共有 4 位字节使能信号 BE0~BE3，因此每个 8 位 I/O 接口电路是否有效要受 BE[3:0]的控制。32 位 I/O 电路连接如图 4-1-4 所示。

IN EAX, DX ;将 I[31:0]连接设备的 32 位数据通过数据总线 D[31:0]输入到 EAX。
 OUT DX, EAX ;将 EAX 中的数据通过数据总线 D[31:0]输出到 O[31:0]连接的设备。

4.1.4 实验说明及步骤

1. 8 位 I/O 操作实验

本实验实现的是将开关 K[7:0]的数据通过输入数据通道读入 CPU 的寄存器,然后再通过输出数据通道将该数据输出到数据灯显示,该程序循环运行,直到按动 PC 键盘上任意按键再退出程序。实验程序流程如图 4-1-5 所示。参考实验接线如图 4-1-6 所示。

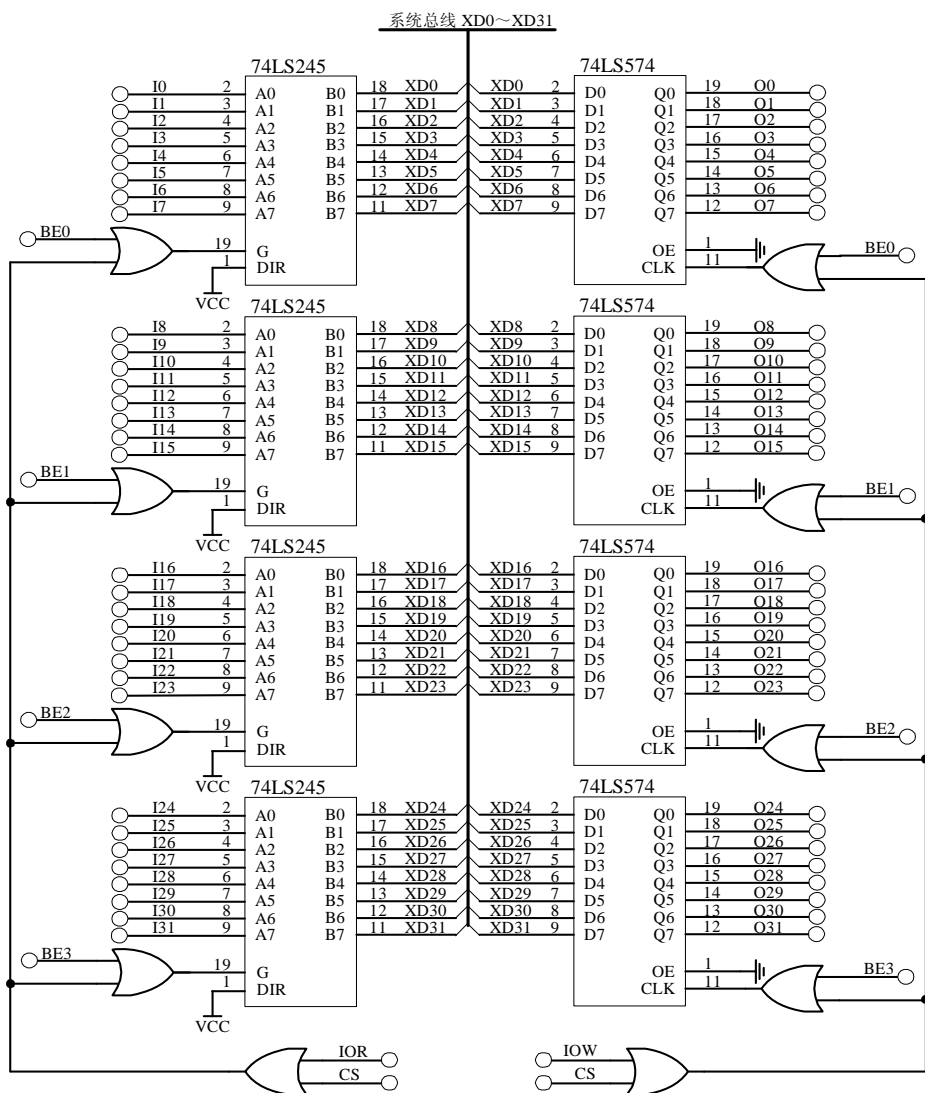


图 4-1-4 用 4 组 8 位 I/O 接口组成的 32 位 I/O 接口电路

实验步骤如下。

- (1) 实验接线图如图 4-1-6 所示，按图连接实验线路图。
- (2) 运行 Tdptit 集成操作软件，根据实验内容，编写实验程序，对实验程序进行编译、链接。
- (3) 运行程序，拨动开关，观看数据灯显示是否正确。

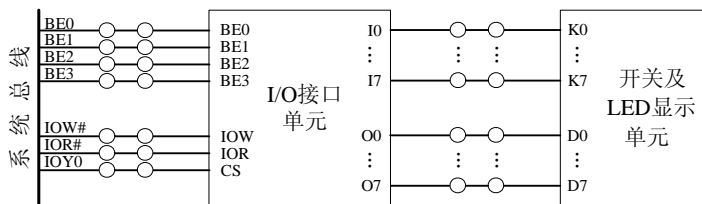
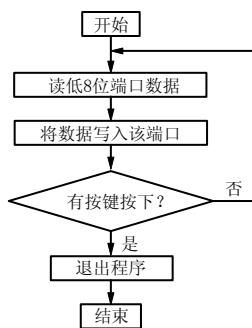


图 4-1-5 8 位 I/O 接口设计实验参考流程图

图 4-1-6 8 位 I/O

接口设计实验参考接线图

实验程序清单 (IO-8. ASM)

```

IOY0      EQU    3000H      ;片选 IOY0 对应的端口始地址
STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
CODE SEGMENT
    ASSUME CS:CODE
START: MOV  DX, IOY0        ;读写基本 I/O 单元低 8 位的端口
    IN  AL, DX
    OUT DX, AL
    MOV  AH, 1              ;判断是否有按键按下
    INT  16H
    JZ   START              ;无按键则跳回继续循环，有则退出
QUIT:  MOV  AX, 4C00H        ;结束程序退出
    INT  21H
CODE ENDS
    END  START
  
```

2. 32 位 I/O 操作实验

利用点阵 LED 显示单元的 16×16 点阵，将 16 行控制和 16 列控制合成一个 32 位端口来操作（列控制连接到发光管的阳极，行控制连接发光管的阴极，列为“1”，相应的行为“0”，则对应的一列发光管点亮）。用 32 位 I/O 接口单元中的 32 位输出 O[31:0] 的高 16 位控制 16 列，低 16 位控制 16 行，即一次 I/O 操作就可完成 LED 点阵的一次显示。实验要求控制点阵循环逐行显示，直到按动 PC 键盘上任意按键再停止程序退出。

实验步骤如下：

(1) 实验接线图如图 4-1-7 所示，按图连接实验线路图。

(2) 运行 Tdpt 集成

操作软件，根据实验内容，编写实验程序，对实验程序进行编译、链接。

(3) 运行程序，观看 LED 点阵显示是否正确。

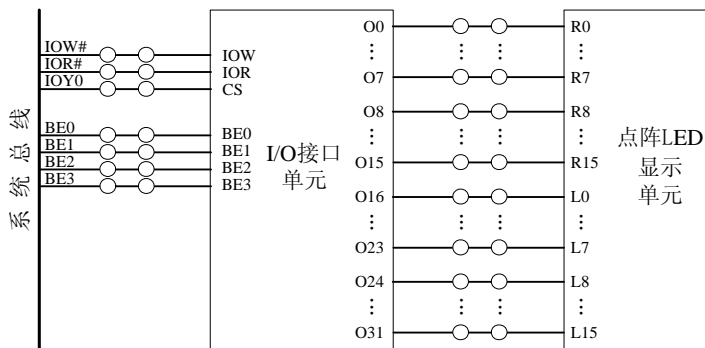


图 4-1-7 32 位 I/O 操作实验参考接线图

实验程序清单 (10-32. ASM)

```
.386P
IOY0      EQU  3000H           ;片选 IOY0 对应的端口始地址
STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
CODE SEGMENT USE16
    ASSUME CS:CODE
START: MOV  CX, 16
    MOV  EAX, 00000001H
LOOP1: MOV  DX, IOY0           ;依次点亮 16 行
    OUT  DX, EAX
    CALL DALLY
    ROL  EAX, 1
    LOOP LOOP1
CHECK: MOV  AH, 1              ;判断是否有按键按下?
    INT  16H
    JZ   START
QUIT:  MOV  EAX, 0
    MOV  DX, IOY0
    OUT  DX, EAX
    MOV  AX, 4C00H             ;结束程序退出
    INT  21H
DALLY PROC NEAR               ;软件延时子程序
    PUSH EAX
    MOV  EAX, 08FFFFFFH
D1:    DEC  EAX
    JNZ  D1
    POP  EAX
    RET
DALLY ENDP
CODE ENDS
    END  START
```

4.2 地址译码电路设计实验

4.2.1 实验目的

- (1) 学习 3-8 译码器在接口电路中的应用。
- (2) 掌握地址译码电路的一般设计方法。

4.2.2 实验设备

PC 微机一台、TD-PITD+实验系统一套。

4.2.3 实验内容

用 74LS138 译码器设计地址译码电路，并用其输出作为基本输入输出单元的片选信号，使用设计的端口地址编写程序，实现数据的输入输出。

4.2.4 实验原理

微机接口电路中，常采用 74LS138 译码器来实现 I/O 端口或存储器的地址译码。74LS138 有 3 个输入引脚、3 个控制引脚及 8 个输出引脚，其管脚信号如图 4-2-1 所示。当 3 个控制信号有效时，相应于输入信号 A、B、C 状态的那个输出端为低电平，该信号即可作为片选信号。

74LS138 输入输出对应关系如表 4-2-1 所示。

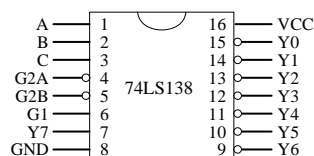


图 4-2-1 74LS138 译码器管脚

表 4-2-1 74LS138 输入输出对应关系

G1	G2A	G2B	A	B	C	Y0	Y1	Y2	Y2	Y4	Y5	Y6	Y7
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	1	0	0	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	1	1	0	1	1	1	1
1	0	0	0	0	1	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

0	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	1	X	X	X	1	1	1	1	1	1	1	1

32 位总线地址是由 XA2 开始，所以地址是以 4 字节边界对齐的。实验系统的 I/O 地址空间共有 256 字节，偏移地址一般从 00H~FFH。起始地址由 PC 机系统分配，可以查看端口资源得到起始地址。所以设计地址译码电路，主要是针对 XA7 以下低 8 位地址线译码，得到偏移在 00H~FFH 之间的端口。本实验要求不使用总线上的片选信号，自行设计端口偏移地址为 E0H~FFH 的译码电路，然后用译码输出作为 I/O 接口单元的片选。编写程序，完成 I/O 数据操作。实验参考线路如图 4-2-2 所示。

4.2.5 实验步骤

- (1) 实验接线图如图 4-2-2 所示，按图连接实验线路图。
- (2) 运行 Tdpit 集成操作软件，根据实验内容，编写实验程序，对实验程序进行编译、链接。
- (3) 运行程序，拨动开关，观看数据灯显示是否正确。

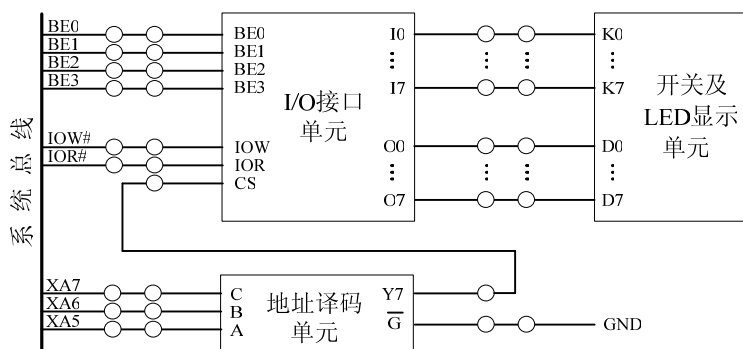


图 4-2-2 地址译码设计实验参考接线图

实验程序清单（T138.ASM）

```

IOY0      EQU    3000H           ;片选 IOY0 对应的端口始地址
Y7        EQU    IOY0+0E0H       ;译码电路输出 Y7 对应的端口地址
STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
CODE SEGMENT
    ASSUME CS:CODE
START: MOV  DX, Y7                ;读写片选接 Y7 的端口
    IN  AL, DX
    OUT DX, AL

```

```
        MOV  AH, 1                ;判断是否有按键按下
        INT  16H
        JZ   START                ;无按键则跳回继续循环，有则退出
QUIT:   MOV  AX, 4C00H            ;结束程序退出
        INT  21H
CODE    ENDS
        END  START
```

4.3 静态存储器扩展实验

4.3.1 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 32 位存储器的访问方法。

4.3.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.3.3 实验内容

编写实验程序，将 PC 机内存中的一段数据传送至扩展的存储器中，然后通过 Tdpit 软件中的“扩展存储区数据显示窗口”查看该存储空间，检测写入数据是否正确。

4.3.4 实验原理

1. SRAM 62256 介绍

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×8bits)，6264 (8K×8 bits) 和 62256 (32K×8 bits)。62256 SRAM 有 32768 个存储单元，每个单元为 8 位字长。62256 的引脚如图 3-3-1 所示。

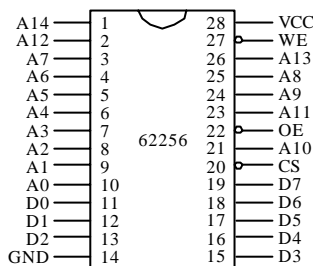


图 4-3-1 62256 引脚图

2. 32 位总线的存储器接口

32 位系统总线提供 XA2~XA31、BE0~BE3 信号为存储器提供物理地址。MY0 是系统为存储器扩展提供的片选信号，其地址空间为 D8000H~DFFFH(详见附录 B 的编程信息),XA2~XA31 用来确定一个 4 字节的存储单元，BE0~BE3 用来确定当前操作中所涉及到 4 字节存储单元中的那个字节。BE0 对应 D[7:0]，BE1 对应 D[15:8]，BE2 对应 D[23:16]，BE3 对应 D[31:24]。其对应关系如表 4-3-1 所示。

表 4-3-1 BE[3:0]指示和数据总线有效对照表

BE3	BE2	BE1	BE0	D[31:24]	D[23:16]	D[15:8]	D[7:0]
1	1	1	0	×	×	×	D[7:0]
1	1	0	1	×	×	D[15:8]	×
1	0	1	1	×	D[23:16]	×	×
0	1	1	1	D[31:24]	×	×	×
1	1	0	0	×	×	D[15:8]	D[7:0]
0	0	1	1	D[31:24]	D[23:16]	×	×
0	0	0	0	D[31:24]	D[23:16]	D[15:8]	D[7:0]

在 SRAM 实验单元中，使用了 4 片 62256 SRAM 构成 4×8bits 的 32 位存储器，存储体分为 0 体、1 体、2 体和 3 体，分别为字节使能线 BE0、BE1、BE2 和 BE3 选通。其电路结构如图 4-3-2 所示。

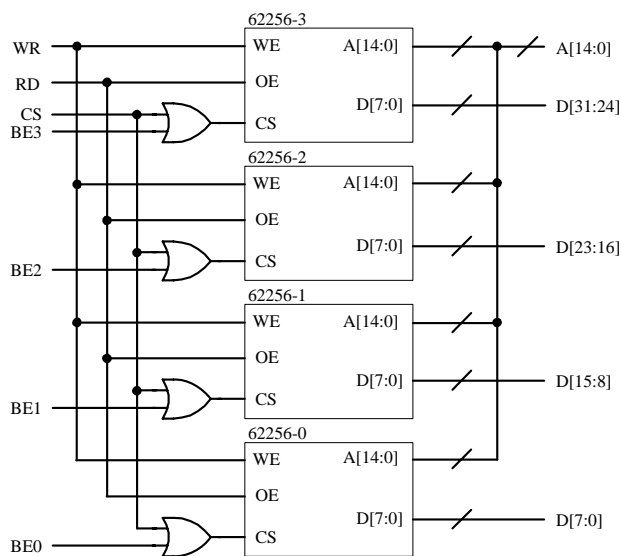


图 4-3-2 32 位存储器单元电路结构图

3. 32 位存储器操作

(1) 规则双字操作

在存储器中，从 4 的整数倍地址开始存放的双字称为规则双字。CPU 访问规则双字只需要一个总线周期，BE0、BE1、BE2 和 BE3 同时有效，从而同时选通 0、1、2 和 3 四个存储

体。两次规则双字操作对应的时序如图 4-3-3 所示。

```
MOV [0000], EAX    ;将 EAX 数据写入地址 0000H 中
MOV [0004], EAX    ;将 EAX 数据写入地址 0004H 中
```

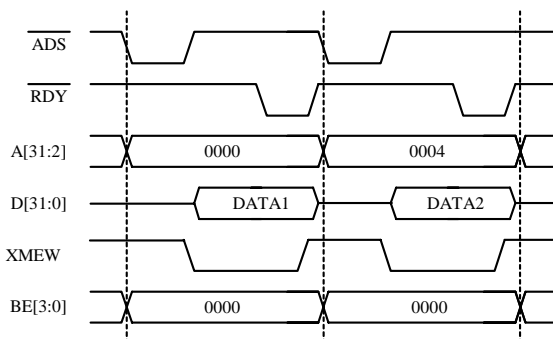


图 4-3-3 32 位存储器规则双字操作时序图

(2) 非规则双字操作

在存储器中，从 4 的非整数倍地址开始存放的双字称为非规则双字。CPU 访问非规则双字需要两个总线周期。通过 BE0、BE1、BE2 和 BE3 在两个周期中选通不同的字节。例如从 4 的整数倍地址加 1 的单元开始访问，第一个总线周期 BE1、BE2 和 BE3 有效，访问 3 个字节；第二个总线周期地址递增，BE0 有效，访问剩余的一个字节。然后自动将 4 个字节组合为一个双字。两次非规则双字操作对应的时序如图 3-3-4 所示。

```
MOV [0001], EAX    ;将 EAX 数据写入地址 0001H 中
MOV [0005], EAX    ;将 EAX 数据写入地址 0005H 中
```

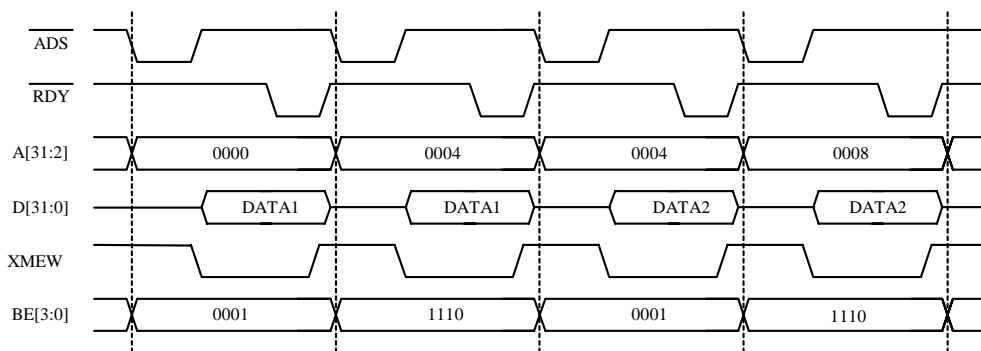


图 3-3-4 32 位存储器非规则双字操作时序图

实验程序清单 (MEM-32. ASM)

```
. 386P
STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT USE16
DD 11111111H, 22222222H, 33333333H, 44444444H ;定义原数据段数据
DD 55555555H, 66666666H, 77777777H, 88888888H
DD 11111111H, 22222222H, 33333333H, 44444444H ;定义原数据段数据
```

```

DD    55555555H, 66666666H, 77777777H, 88888888H
DD    11111111H, 22222222H, 33333333H, 44444444H ;定义原数据段数据
DD    55555555H, 66666666H, 77777777H, 88888888H
DATA ENDS
CODE SEGMENT USE16
        ASSUME CS:CODE, DS:DATA
START: MOV  AX, DATA
        MOV  DS, AX
        MOV  AX, 0D800H
        MOV  ES, AX
        XOR  SI, SI
        XOR  DI, DI
        MOV  CX, 18H
LOOP1:  MOV  EAX, DS:[SI]          ;将源数据段数据传输到目的数据段
        MOV  ES:[DI], EAX
        ADD  SI, 4
        ADD  DI, 4
        LOOP LOOP1
        MOV  AX, 4C00H
        INT  21H
CODE ENDS
        END    START

```

4. 8 位存储器操作

在 32 位总线上可以进行 8 位存储器的操作，允许 CPU 用字节指令进行访问。这样的操作相当于 32 位总线上其它 3 个字节不访问。如图 3-3-2 中只对 0 体存储器(62256-0)进行操作，其它 3 个存储器的数据线不用连接，字节使能始终是 BE0 有效，4 字节空间只访问最低的 1 个字节。

实验程序清单 (MEM-8. ASM)

```

STACK1 SEGMENT STACK
        DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT
DB    00H, 11H, 22H, 33H, 44H, 55H, 66H, 77H ;定义源数据段数据
DB    88H, 99H, 0AAH, 0BBH, 0CCH, 0DDH, 0EEH, 0FFH
DB    00H, 11H, 22H, 33H, 44H, 55H, 66H, 77H
DB    88H, 99H, 0AAH, 0BBH, 0CCH, 0DDH, 0EEH, 0FFH
DB    00H, 11H, 22H, 33H, 44H, 55H, 66H, 77H
DB    88H, 99H, 0AAH, 0BBH, 0CCH, 0DDH, 0EEH, 0FFH
DATA ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS:DATA
START: MOV  AX, DATA
        MOV  DS, AX
        MOV  AX, 0D800H
        MOV  ES, AX
        XOR  SI, SI
        XOR  DI, DI
        MOV  CX, 30H

```

```

LOOP1: MOV  AL, DS:[SI]          ;将源数据段数据传输到目的数据段
        MOV  ES:[DI], AL
        ADD  SI, 1
        ADD  DI, 4
        LOOP LOOP1
        MOV  AX, 4C00H
        INT  21H
CODE  ENDS
        END    START

```

4.3.5 实验步骤

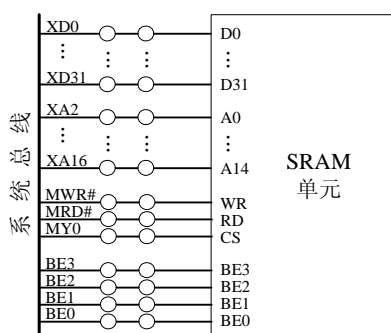


图 4-3-5 32 位存储器扩展实验参考接线图

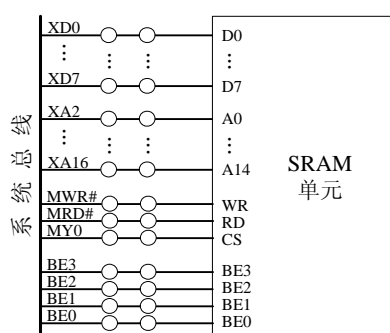


图 4-3-6 8 位存储器扩展实验参考接线图

- (1) 实验接线图如图 4-3-5 和 4-3-6 所示，按图接线。
- (2) 运行 Tdpt 集成操作软件，根据实验要求编写程序，编译、链接。
- (3) 使用运行命令运行程序，待程序运行停止后。
- (4) 通过软件中的“扩展存储区数据显示窗口”查看该存储空间，检测写入数据是否正确。
- (5) 改变实验程序，按非规则字写存储器，观察实验结果。
- (6) 改变实验程序，按字节方式写存储器，观察实验现象。

4.4 8259 中断控制实验

4.4.1 实验目的

1. 掌握 8259 中断控制器的工作原理。
2. 掌握系统总线上中断请求的应用编程方法。

4.4.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.4.3 实验内容

- (1) 利用系统总线上中断请求信号 INTR1，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 INTR1 和 INTR2，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) 利用实验平台上 8259 控制器，与 PC 内部主片 8259 进行级连。编写程序对级连 8259 控制器的 IR0 和 IR1 中断请求进行处理。

4.4.4 实验原理

1. 中断控制器 8259 简介

中断控制器 8259 是 Intel 公司专为控制优先级中断而设计开发的芯片。它将中断源优先级排队、辨别中断源以及提供中断矢量的电路集于一片中，因此无需附加任何电路，只需对 8259 进行编程，就可以管理 8 级中断，并选择优先模式和中断请求方式，即中断结构可以由用户编程来设定。同时，在不需增加其他电路的情况下，通过多片 8259 的级连，能构成多达 64 级的矢量中断系统。它的管理功能包括：1) 记录各级中断源请求，2) 判别优先级，确定是否响应和响应哪一级中断，3) 响应中断时，向 CPU 传送中断类型号。8259 的内部结构和引脚如图 4-4-1 所示。

8259 的命令共有 7 个，一类是初始化命令字，另一类是操作命令字。8259 的编程就是根据应用需要将初始化命令字 ICW1-ICW4 和操作命令字 OCW1- OCW3 分别写入初始化命令寄存器组和操作命令寄存器组。ICW1-ICW4 各命令字格式如图 4-4-2 所示，OCW1-OCW3 各命令字格式如图 4-4-3 所示，其中 OCW1 用于设置中断屏蔽操作字，OCW2 用于设置优先级循环方式和中断结束方式的操作命令字，OCW3 用于设置和撤销特殊屏蔽方式、设置中断查

询方式以及设置对 8259 内部寄存器的读出命令。

2. 8259 寄存器及命令的控制访问

在硬件系统中, 8259 仅占用两个外设接口地址, 在片选有效的情况下, 利用 A0 来寻址不同的寄存器和命令字。对寄存器和命令的访问控制如表 4-4-1 所示。

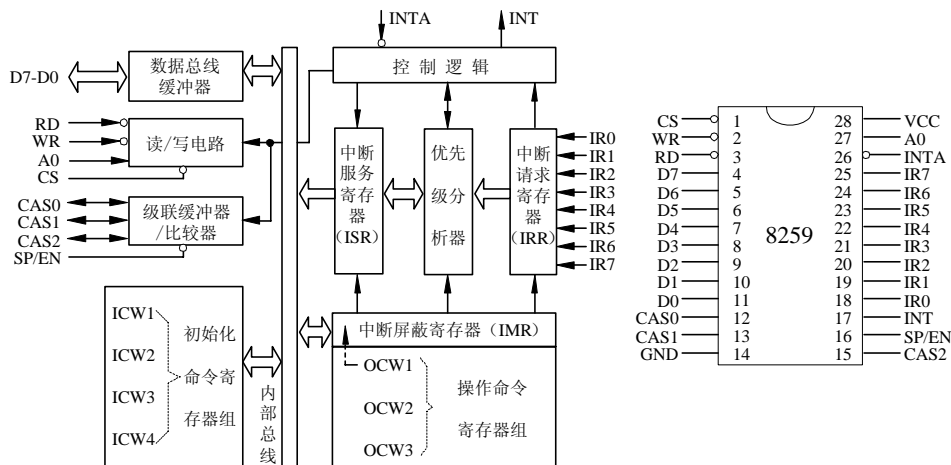


图 4-4-1 8259 内部结构和引脚图

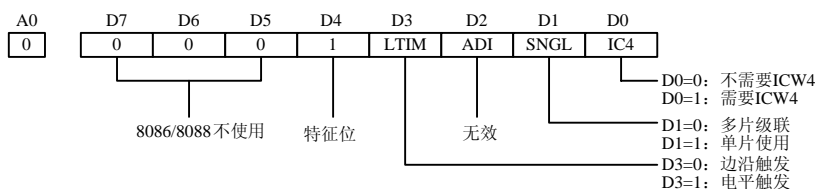


图 4-4-2 (a) ICW1 格式

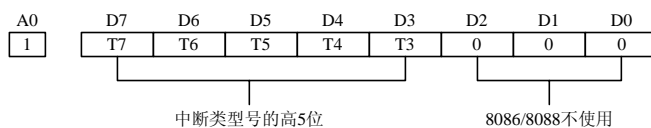


图 4-4-2 (b) ICW2 格式

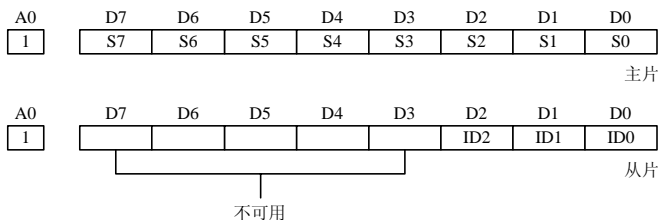


图 4-4-2 (c) ICW3 格式



图 4-4-2 (d) ICW4 格式



图 4-4-3 OCW 命令字格式

表 4-4-1 8259 寄存器及命令的访问控制

A0	D4	D3	读信号	写信号	片选	操作
0			0	1	0	读出 ISR,IRR 的内容
1			0	1	0	读出 IMR 的内容
0	0	0	1	0	0	写入 OCW2
0	0	1	1	0	0	写入 OCW3
0	1	×	1	0	0	写入 ICW1
1	×	×	1	0	0	写入 OCW1, ICW2, ICW3, ICW4

3. PC 微机系统中的 8259

在 80x86 系列 PC 微机系统中，系统中包含了两片 8259 中断控制器，通过级连可以管理 15 级硬件中断，但其中部分中断号已经被系统硬件占用，具体情况如表 4-4-2 示。两片 8259 的端口地址为：主片 8259 使用 20H 和 21H 两个端口；从片使用 A0H 和 A1H 两个端口。系统初始化两片 8259 的中断请求信号均采用上升沿触发，采用全嵌套方式，优先级的排列次序为 0 级最高，依次为 1 级、8 级~15 级，然后是 3 级~7 级。

在实验平台上系统总线单元的 INTR1 和 INTR2 两个信号对应的是两路中断请求线。在 Tdpit 集成操作环境中，INTR1 对应的是 PC 机内部主片 8259 中断的 IRQ7，INTR2 对应的是 IRQ6。PC 机内部 8259 已经在 PC 启动时初始化好，在使用时主要是将其中断屏蔽位打开，修改中断向量，程序结束时还原中断向量。

表 4-4-2 PC 微机系统中的硬件中断

中断号	功能	中断向量号	中断向量地址
主 8259 IRQ0	日时钟/计数器 0	08H	0020H~0023H
主 8259 IRQ1	键盘	09H	0024H~0027H
主 8259 IRQ2	接从片 8259	0AH	0028H~002BH
主 8259 IRQ3	串行口 2	0BH	002CH~002FH
主 8259 IRQ4	串行口 1	0CH	0030H~0033H
主 8259 IRQ5	并行口 2	0DH	0034H~0037H
主 8259 IRQ6	软盘	0EH	0038H~003BH
主 8259 IRQ7	并行口 1	0FH	003CH~003FH
从 8259 IRQ8	实时钟	70H	01C0H~01C3H
从 8259 IRQ9	保留	71H	01C4H~01C7H
从 8259 IRQ10	保留	72H	01C8H~01CBH
从 8259 IRQ11	保留	73H	01CCH~01CFH
从 8259 IRQ12	保留	74H	01D0H~01D3H
从 8259 IRQ13	协处理器中断	75H	01D4H~01D7H
从 8259 IRQ14	硬盘控制器	76H	01D8H~01DBH
从 8259 IRQ15	保留	77H	01DCH~01DFH

4.4.5 实验说明及步骤

1. 单中断应用实验

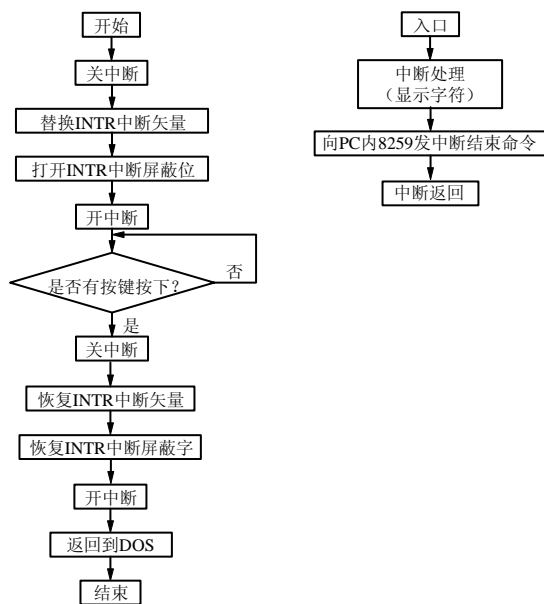
在前面已经介绍了实验平台上系统总线单元的 INTR1 中断请求信号已经是对应到 PC 机内部主片 8259 的 IRQ7。INTR1 产生一个上升沿的中断请求，PC 机内部相应的 IRQ7 中断处理就会得到响应。所以，使用 INTR1 中断请求信号，就相当在使用 PC 机内部 IRQ7 中断。

本实验要求使用总线上 INTR1 (IRQ7) 中断请求线完成一次单中断应用实验。用单次脉冲上升沿模拟中断源，中断处理程序完成在屏幕上的显示字符“7”。参考程序流程如图 4-4-5 所示。实验步骤如下。

(1) 实验接线图如图 4-4-4 所示，按图接线。

(2) 运行 Tdpit 集成操作软件，参考流程图 4-4-5 编写程序，编译、链接。

(3) 使用运行命令运行程序，重复按单次脉冲开关 KK1+，显示屏会显示字符“7”，说明响应了中断。



(a)主程序

(b)中断处理程序

图 4-4-5 8259 单中断应用实验参考程序流程图

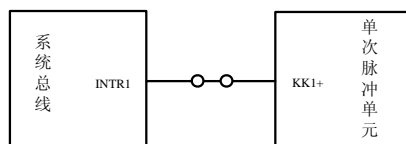


图 4-4-4 8259 单中断实验接线图

实验程序清单 (T8259-1. ASM)

```

INTR_IVADD EQU 003CH          ; INTR 对应的中断矢量地址
STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT
MES      DB  'Press any key to exit!', 0AH, 0DH, 0AH, 0DH, '$'
CS_BAK   DW  ?                ; 保存 INTR 原中断处理程序入口段地址的变量
IP_BAK   DW  ?                ; 保存 INTR 原中断处理程序入口偏移地址的变量
IM_BAK   DB  ?                ; 保存 INTR 原中断屏蔽字的变量
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
      MOV DS, AX
      MOV DX, OFFSET MES      ; 显示退出提示
      MOV AH, 09H
      INT 21H
      CLI
      MOV AX, 0000H           ; 替换 INTR 的中断矢量
      MOV ES, AX
      MOV DI, INTR_IVADD
      MOV AX, ES:[DI]
  
```

```

        MOV IP_BAK, AX                ;保存 INTR 原中断处理程序入口偏移地址
        MOV AX, OFFSET MYISR
        MOV ES: [DI], AX              ;设置当前中断处理程序入口偏移地址
        ADD DI, 2
        MOV AX, ES: [DI]
        MOV CS_BAK, AX                ;保存 INTR 原中断处理程序入口段地址
        MOV AX, SEG MYISR
        MOV ES: [DI], AX              ;设置当前中断处理程序入口段地址
        IN AL, 21H
        MOV IM_BAK, AL                ;保存 INTR 原中断屏蔽字
        AND AL, 7FH
        OUT 21H, AL
        STI
WAIT1:  MOV AH, 1                      ;判断是否有按键按下
        INT 16H
        JZ WAIT1                      ;无按键则跳回继续等待，有则退出
QUIT:   CLI
        MOV AX, 0000H                  ;恢复 INTR 原中断矢量
        MOV ES, AX
        MOV DI, INTR_IVADD
        MOV AX, IP_BAK                ;恢复 INTR 原中断处理程序入口偏移地址
        MOV ES: [DI], AX
        ADD DI, 2
        MOV AX, CS_BAK                ;恢复 INTR 原中断处理程序入口段地址
        MOV ES: [DI], AX
        MOV AL, IM_BAK                ;恢复 INTR 原中断屏蔽寄存器的屏蔽字
        OUT 21H, AL
        STI
        MOV AX, 4C00H                  ;返回到 DOS
        INT 21H
MYISR PROC NEAR                        ;中断处理程序 MYISR
        PUSH AX
        MOV AL, 37H
        MOV AH, 0EH
        INT 10H
        MOV AL, 20H
        INT 10H
OVER:   MOV AL, 20H
        OUT 20H, AL
        POP AX
        IRET
MYISR ENDP
CODE ENDS
        END START

```

2. 8259 中断优先级应用实验

8259 内部带有优先级排队电路，能对 8 个或级连中断源实现优先级控制。当一个中断请求 IR_x 被服务时，又可能有后续的中断请求到达，或者其它中断 IR_y 请求到达。发生这种情况时，8259 通过优先级控制电路来决定后续中断请求是否可以打断当前的服务。

本实验要求使用 INTR1 和 INTR2 实现一个双中断的应用实验，来观察 8259 的优先级控制功能。在实验程序中对 INTR1 和 INTR2 所对应的 IRQ7 和 IRQ6 分别设计两个中断服务，用单次脉冲 KK1 上升沿模拟 INTR1 (IRQ7) 中断源，并在所对应的中断服务中，在屏幕上显示字符“7”。用单次脉冲 KK2 上升沿模拟 INTR2 (IRQ6) 中断源，并在对应的中断服务中显示字符“6”。主程序延时显示“-”等待。重点观察两个中断服务对 KK1 和 KK2 两个开关先后次序的响应。参考程序流程如图 4-4-6 所示。

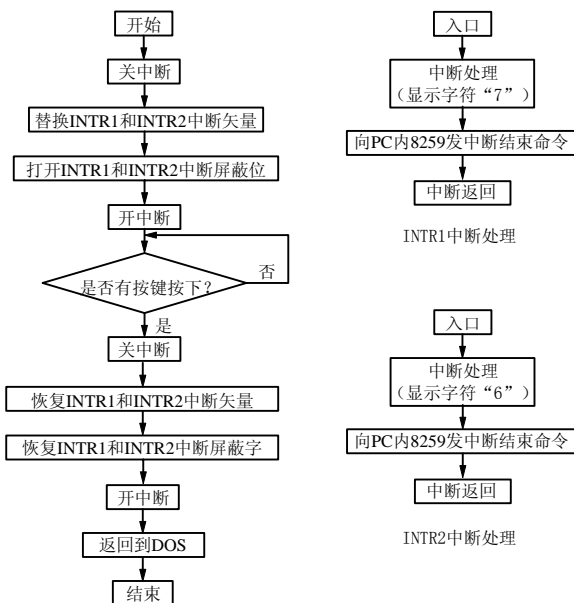


图 4-4-6 8259 中断优先级应用实验参考程序流程图

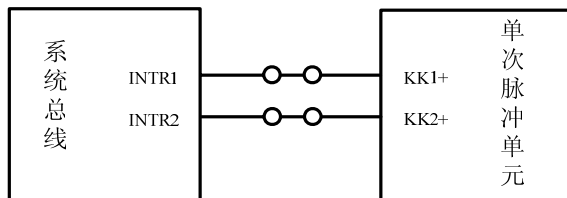


图 4-4-7 8259 中断优先级应用实验参考接线图

实验步骤如下。

- (1) 实验接线图如图 4-4-7 所示，按图接线。
- (2) 运行 Tdptit 集成操作软件，参考流程图 4-4-6 编写程序，编译、链接。
- (3) 运行程序，先后按动 KK1+、KK2+ 按键，观察中断优先级响应是否正确。

实验程序清单 (T8259-2. ASM)

```

INTR1_IVADD EQU 003CH      ; INTR1 对应的中断矢量地址
INTR2_IVADD EQU 0038H      ; INTR2 对应的中断矢量地址
STACK1 SEGMENT STACK
    DW 256 DUP(?)
  
```

```

STACK1 ENDS
DATA SEGMENT
MES      DB    'Press any key to exit!', 0AH, 0DH, 0AH, 0DH, '$'
CS_BAK1  DW    ?                ;保存 INTR1 原中断处理程序入口段地址的变量
IP_BAK1  DW    ?                ;保存 INTR1 原中断处理程序入口偏移地址的变量
CS_BAK2  DW    ?                ;保存 INTR2 原中断处理程序入口段地址的变量
IP_BAK2  DW    ?                ;保存 INTR2 原中断处理程序入口偏移地址的变量
IM_BAK   DB    ?                ;保存 INTR 原中断屏蔽字的变量
DATA ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS:DATA
START:  MOV AX, DATA
        MOV DS, AX
        MOV DX, OFFSET MES      ;显示退出提示
        MOV AH, 09H
        INT 21H
        CLI
        MOV AX, 0000H           ;替换 INTR1 的中断矢量
        MOV ES, AX
        MOV DI, INTR1_IVADD
        MOV AX, ES:[DI]
        MOV IP_BAK1, AX        ;保存 INTR1 原中断处理程序入口偏移地址
        MOV AX, OFFSET MYISR1
        MOV ES:[DI], AX        ;设置当前中断处理程序入口偏移地址
        ADD DI, 2
        MOV AX, ES:[DI]
        MOV CS_BAK1, AX        ;保存 INTR1 原中断处理程序入口段地址
        MOV AX, SEG MYISR1
        MOV ES:[DI], AX        ;设置当前中断处理程序入口段地址
        MOV DI, INTR2_IVADD
        MOV AX, ES:[DI]
        MOV IP_BAK2, AX        ;保存 INTR2 原中断处理程序入口偏移地址
        MOV AX, OFFSET MYISR2
        MOV ES:[DI], AX        ;设置当前中断处理程序入口偏移地址
        ADD DI, 2
        MOV AX, ES:[DI]
        MOV CS_BAK2, AX        ;保存 INTR2 原中断处理程序入口段地址
        MOV AX, SEG MYISR2
        MOV ES:[DI], AX        ;设置当前中断处理程序入口段地址
        IN  AL, 21H
        MOV IM_BAK, AL         ;保存 INTR 原中断屏蔽字
        AND AL, 7FH
        OUT 21H, AL
        STI
WAIT1:  MOV AH, 0EH
        MOV AL, 2DH
        INT 10H
        CALL DALLY
        MOV AH, 1              ;判断是否有按键按下
        INT 16H
        JZ  WAIT1              ;无按键则跳回继续等待，有则退出

```

```

QUIT:  CLI
        MOV AX, 0000H           ;恢复 INTR 原中断矢量
        MOV ES, AX
        MOV DI, INTR1_IVADD
        MOV AX, IP_BAK1         ;恢复 INTR1 原中断处理程序入口偏移地址
        MOV ES: [DI], AX
        ADD DI, 2
        MOV AX, CS_BAK1         ;恢复 INTR1 原中断处理程序入口段地址
        MOV ES: [DI], AX
        MOV DI, INTR2_IVADD
        MOV AX, IP_BAK2         ;恢复 INTR2 原中断处理程序入口偏移地址
        MOV ES: [DI], AX
        ADD DI, 2
        MOV AX, CS_BAK2         ;恢复 INTR2 原中断处理程序入口段地址
        MOV ES: [DI], AX
        MOV AL, IM_BAK          ;恢复 INTR 原中断屏蔽寄存器的屏蔽字
        OUT 21H, AL
        STI
        MOV AX, 4C00H           ;返回到 DOS
        INT 21H
MYISR1 PROC NEAR                ;中断处理程序 MYISR1
        PUSH AX
        STI
        CALL DALLY
        MOV AL, 37H
        MOV AH, 0EH
        INT 10H
        MOV AL, 20H
        INT 10H
        MOV AL, 20H
        OUT 20H, AL
        POP AX
        IRET
MYISR1 ENDP
MYISR2 PROC NEAR                ;中断处理程序 MYISR2
        PUSH AX
        STI
        CALL DALLY
        MOV AL, 36H
        MOV AH, 0EH
        INT 10H
        MOV AL, 20H
        INT 10H
        MOV AL, 20H
        OUT 20H, AL
        POP AX
        IRET
MYISR2 ENDP
DALLY PROC NEAR                 ;软件延时子程序
        PUSH CX
        PUSH AX

```



```

        MOV CX, 02000H
D1:     MOV AX, 0F000H
D2:     DEC AX
        JNZ D2
        LOOP D1
        POP AX
        POP CX
        RET
DALLY ENDP
CODE ENDS
END START

```

3. 实验平台 8259 与 PC 内部主片 8259 级连中断应用实验

利用实验单元中的 8259 控制器，可以与 PC 内部主片 8259 进行级连。在 PC 机内部，主片 8259 采用的是 IRQ2 请求线与从片 8259 进行的级连。理论上，主片 8259 的其它 7 根请求线还可以级连更多的从片 8259。在我们实验平台上，规定可以在 INTR1 (IRQ7) 请求线上进行级连 8259。Tdpit 集成操作环境为 INTR1 (IRQ7) 上级连的 8259 中断源分配了相应的中断矢量，其对应关系如表 4-4-3 所示。

表 4-4-3 实验平台 8259 中断源中断矢量分配表

中断号	中断向量号	中断向量地址
实验平台 8259 IR0	78H	01E0H~01E3H
实验平台 8259 IR1	79H	01E4H~01E7H
实验平台 8259 IR2	7AH	01E8H~01EBH
实验平台 8259 IR3	7BH	01ECH~01EFH
实验平台 8259 IR4	7CH	01F0H~01F3H
实验平台 8259 IR5	7DH	01F4H~01F7H
实验平台 8259 IR6	7EH	01F8H~01FBH
实验平台 8259 IR7	7FH	01FCH~01FFH

本实验要求利用实验平台 8259 单元，结合系统总线上的 INTR1 和 INTA，实现在主片 8259 IRQ7 上级连从片 8259 的应用。实验程序中针对 8259 单元 IR0、IR1 两路中断请求设计两个中断服务。用 KK1+和 KK2+模拟两个中断源，在 IR0 对应的服务程序中显示字符“0”，在 IR1 对应的服务程序中显示字符“1”。同时可以观察两个中断服务对 KK1 和 KK2 两个开关先后次序的响应。

实验步骤如下。

- (1) 实验接线图如图 4-4-8 所示，按图接线。
- (2) 运行 Tdpit 集成操作软件，参考流程图 4-4-9 编写程序，编译、链接。
- (3) 使用运行命令运行程序，按动 KK1+、KK2+按键，观察级连中断响应是否正确。

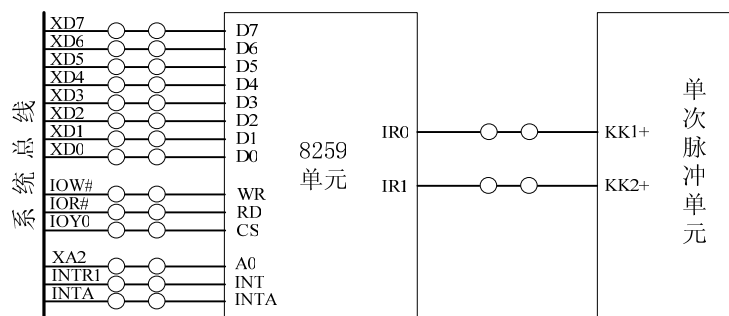


图 4-4-8 8259 级连中断应用实验参考接线图

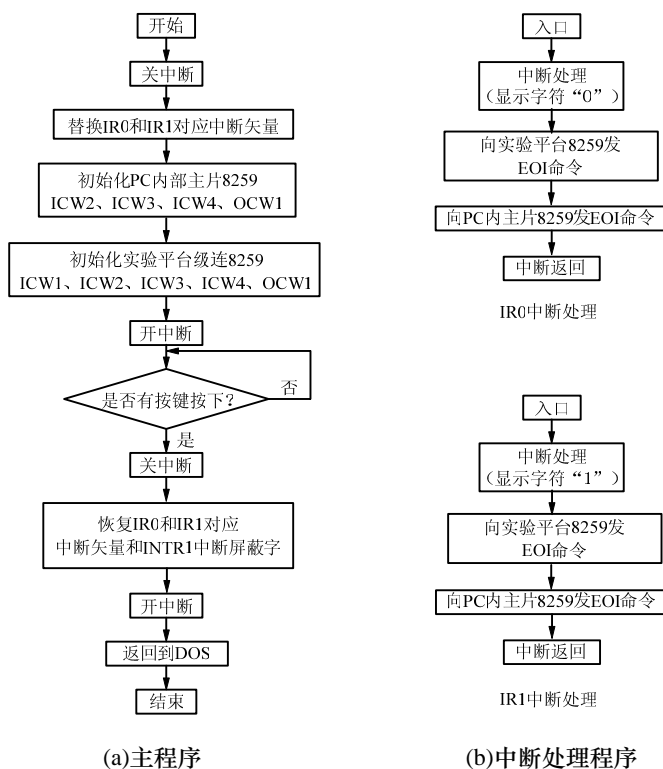


图 4-4-9 8259 级连中断应用实验参考程序流程图

实验程序清单 (T8259-3. ASM)

```

INTR_IVADD0 EQU 01E0H      ;INT78 中断矢量地址
INTR_IVADD1 EQU 01E4H      ;INT79 中断矢量地址
IOY0 EQU 3000H             ;片选 IOY0 对应的端口始地址
MY8259_ICW1 EQU IOY0+00H   ;实验系统中 8259 的 ICW1 端口地址
MY8259_ICW2 EQU IOY0+04H   ;实验系统中 8259 的 ICW2 端口地址
MY8259_ICW3 EQU IOY0+04H   ;实验系统中 8259 的 ICW3 端口地址
MY8259_ICW4 EQU IOY0+04H   ;实验系统中 8259 的 ICW4 端口地址
MY8259_OCW1 EQU IOY0+04H   ;实验系统中 8259 的 OCW1 端口地址
MY8259_OCW2 EQU IOY0+00H   ;实验系统中 8259 的 OCW2 端口地址
  
```

```

MY8259_OCW3 EQU IOY0+00H ;实验系统中 8259 的 OCW3 端口地址
STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT
MES      DB 'Press any key to exit!', 0AH, 0DH, 0AH, 0DH, '$'
CS_BAK0  DW ? ;保存 INTR 原中断处理程序入口段地址的变量
IP_BAK0  DW ? ;保存 INTR 原中断处理程序入口偏移地址的变量
CS_BAK1  DW ? ;保存 INTR 原中断处理程序入口段地址的变量
IP_BAK1  DW ? ;保存 INTR 原中断处理程序入口偏移地址的变量
IM_BAK   DB ? ;保存 INTR 原中断屏蔽字的变量
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
    MOV DS, AX
    MOV DX, OFFSET MES ;显示退出提示
    MOV AH, 09H
    INT 21H
    CLI
    MOV AX, 0000H ;替换 INTR 的中断矢量
    MOV ES, AX
    MOV DI, INTR_IVADD0
    MOV AX, ES:[DI]
    MOV IP_BAK0, AX ;保存 INTR 原中断处理程序入口偏移地址
    MOV AX, OFFSET MYISR0
    MOV ES:[DI], AX ;设置当前中断处理程序入口偏移地址
    ADD DI, 2
    MOV AX, ES:[DI]
    MOV CS_BAK0, AX ;保存 INTR 原中断处理程序入口段地址
    MOV AX, SEG MYISR0
    MOV ES:[DI], AX ;设置当前中断处理程序入口段地址
    MOV AX, 0000H ;替换 INTR 的中断矢量
    MOV ES, AX
    MOV DI, INTR_IVADD1
    MOV AX, ES:[DI]
    MOV IP_BAK1, AX ;保存 INTR 原中断处理程序入口偏移地址
    MOV AX, OFFSET MYISR1
    MOV ES:[DI], AX ;设置当前中断处理程序入口偏移地址
    ADD DI, 2
    MOV AX, ES:[DI]
    MOV CS_BAK1, AX ;保存 INTR 原中断处理程序入口段地址
    MOV AX, SEG MYISR1
    MOV ES:[DI], AX ;设置当前中断处理程序入口段地址
    MOV AL, 20H ;初始化主片 8259
    OUT 21H, AL
    MOV AL, 84H
    OUT 21H, AL
    MOV AL, 0DH
    OUT 21H, AL

```

```

    IN  AL, 021H
    MOV IM_BAK, AL           ;保存 INTR 原中断屏蔽字
    AND AL, 07fH
    OUT 021H, AL
    MOV DX, MY8259_ICW1      ;初始化实验系统中 8259 的 ICW1
    MOV AL, 11H              ;边沿触发、单片 8259、需要 ICW4
    OUT DX, AL
    MOV DX, MY8259_ICW2      ;初始化实验系统中 8259 的 ICW2
    MOV AL, 78H
    OUT DX, AL
    MOV DX, MY8259_ICW3
    MOV AL, 07H
    OUT DX, AL
    MOV DX, MY8259_ICW4      ;初始化实验系统中 8259 的 ICW4
    MOV AL, 09H              ;非自动结束 EOI
    OUT DX, AL
    MOV DX, MY8259_OCW1      ;初始化实验系统中 8259 的 OCW1
    MOV AL, 0FCH              ;打开 IR0 和 IR1 的屏蔽位
    OUT DX, AL
    STI
WAIT1: MOV AH, 0EH
    MOV AL, 2DH
    INT 10H
    CALL DALLY
    MOV AH, 1                 ;判断是否有按键按下
    INT 16H
    JZ  WAIT1                 ;无按键则跳回继续等待，有则退出
QUIT:  CLI
    MOV AX, 0000H             ;恢复 INTR 原中断矢量
    MOV ES, AX
    MOV DI, INTR_IVADD0
    MOV AX, IP_BAK0           ;恢复 INTR 原中断处理程序入口偏移地址
    MOV ES: [DI], AX
    ADD DI, 2
    MOV AX, CS_BAK0           ;恢复 INTR 原中断处理程序入口段地址
    MOV ES: [DI], AX
    MOV DI, INTR_IVADD1
    MOV AX, IP_BAK1           ;恢复 INTR 原中断处理程序入口偏移地址
    MOV ES: [DI], AX
    ADD DI, 2
    MOV AX, CS_BAK1           ;恢复 INTR 原中断处理程序入口段地址
    MOV ES: [DI], AX
    MOV AL, IM_BAK            ;恢复 INTR 原中断屏蔽寄存器的屏蔽字
    OUT 021H, AL
    STI
    MOV AX, 4C00H             ;返回到 DOS
    INT 21H
MYISR0 PROC NEAR             ;中断处理程序 MYISR
    PUSH AX
    STI
    CALL DALLY

```

```
        MOV AH, 0EH
        MOV AL, 30H
        INT 10H
        MOV AL, 20H
        INT 10H
        MOV DX, MY8259_OCW2
        MOV AL, 20H
        OUT DX, AL
        MOV AL, 20H
        OUT 20H, AL
        POP AX
        IRET
MYISR0 ENDP
MYISR1 PROC NEAR                ;中断处理程序 MYISR
        PUSH AX
        STI
        CALL DALLY
        MOV AH, 0EH
        MOV AL, 31H
        INT 10H
        MOV AL, 20H
        INT 10H
        MOV DX, MY8259_OCW2
        MOV AL, 20H
        OUT DX, AL
        MOV AL, 20H
        OUT 20H, AL
        POP AX
        IRET
MYISR1 ENDP
DALLY PROC NEAR                ;软件延时子程序
        PUSH CX
        PUSH AX
        MOV CX, 02000H
D1:     MOV AX, 0F000H
D2:     DEC AX
        JNZ D2
        LOOP D1
        POP AX
        POP CX
        RET
DALLY ENDP
CODE ENDS
        END START
```

4.5 8237 DMA 控制应用实验

4.5.1 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。

4.5.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.5.3 实验原理及内容

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、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。

1. 8237 芯片介绍

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 4-5-1 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)。

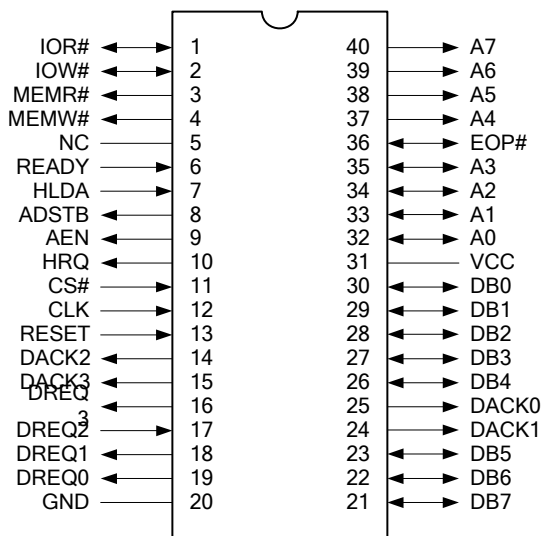
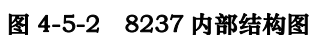


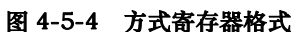
图 4-5-1 8237 外部引脚图

8237 的内部结构图如图 4-5-2 所示。



命令寄存器	D7	D6	D5	D4	D3	D2	D1	D0
0:DACK低电平有效								
1:DACK高电平有效								
0:DREQ高电平有效								
1:DREQ低电平有效								
0:不扩展写信号								
1:扩展写信号								
0:固定优先级								
1:循环优先级								

图 4-5-3 命令寄存器格式



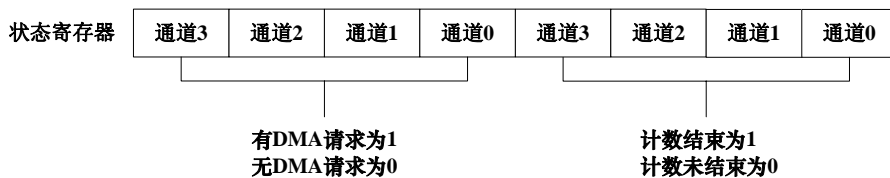


图 4-5-5 8237 状态寄存器

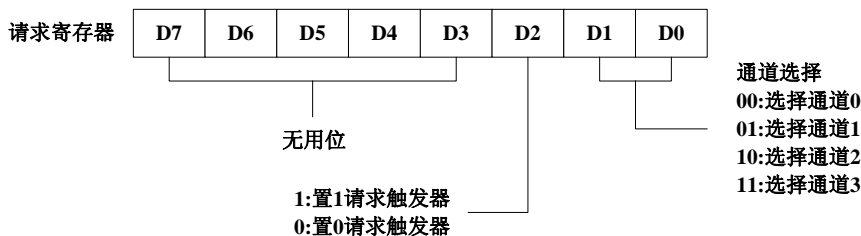
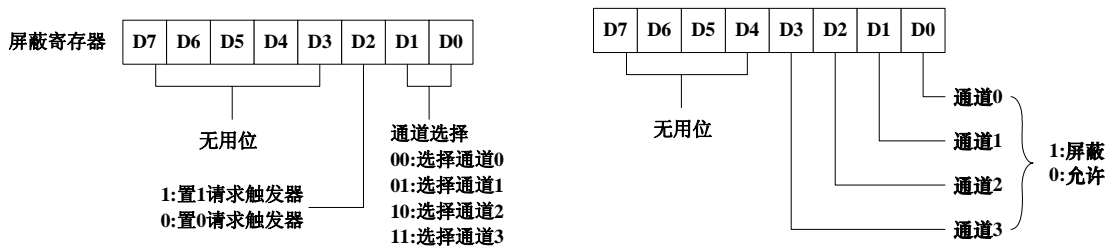


图 4-5-6 8237 请求寄存器格式



(a) 单个通道屏蔽寄存器格式

(b) 4 个通道屏蔽寄存器格式

图 4-5-7 通道屏蔽寄存器格式

表 4-5-1 列出了 8237 内部寄存器和软命令及其操作信息。

表 4-5-1 8237 内部寄存器和软命令及其读写操作一览表

[illegible]

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		写	1	0	1	1	0	0	CH		
主清除	-	写	1	0	1	1	1	0	EH		
清先/后触发器	-	写	1	0	1	1	1	0	FH		
清屏蔽寄存器	-	写	1	0	1	1	1	1			
写 4 通道屏蔽位寄存器	4	写	1	0	1	1	1	1			
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

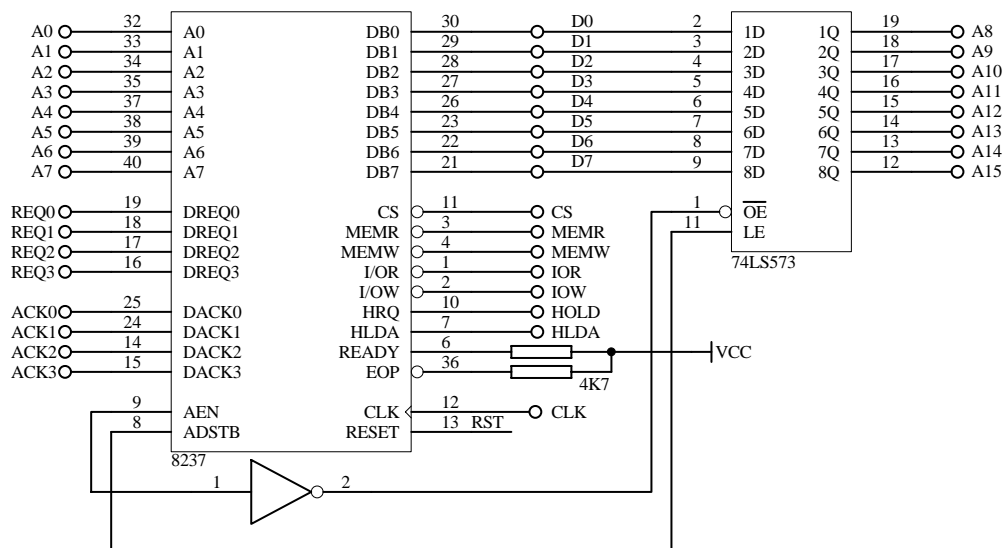


图 4-5-8 DMA 实验单元电路图

3. 实验内容

(1) 存储器到存储器 DMA 传输实验。利用 8237 两个传输通道，设计一个 DMA 传输。将扩展存储器单元中偏移 0000H~0007H 的连续 8 个字节传送到偏移 0008H~000FH 的连续 8 个单元中。实验参考线路图如图 4-5-9 所示。

(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。实验参考线路图如图 4-5-11 所示。

(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。实验参考线路图如图 4-5-11 所示。

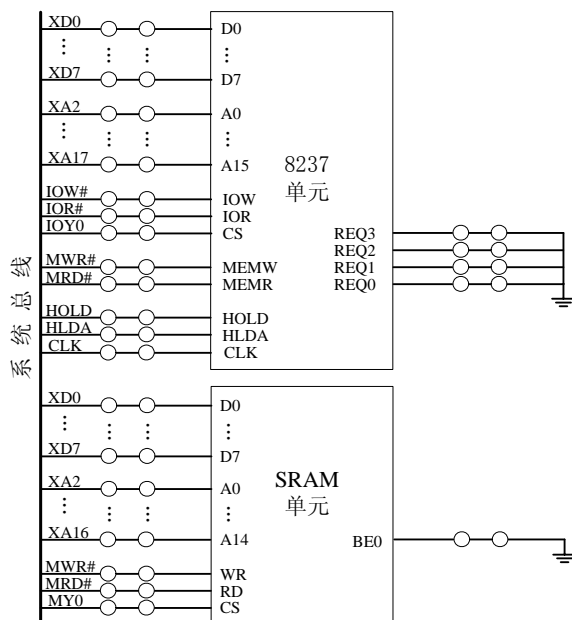
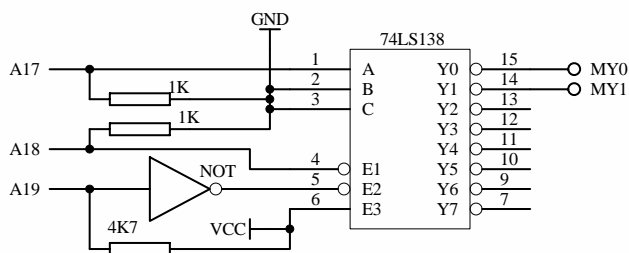


图 4-5-9 8237 实现存储器到存储器传输实验接线图

实验系统中提供了 MY0 这个存储器片选信号，译码空间为 D8000H~DFFFFH。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。



4.5.4 实验步骤

1. 存储器到存储器 DMA 传输实验

- (1) 实验接线图如图 4-5-9 所示，按图接线。
- (2) 运行 Tdpt 集成操作软件，参考流程图 4-5-10 编写程序，编译、链接。
- (3) 打开软件中的“扩展存储区数据显示窗口”，对存储器的前 8 个字节空间写数，即“00H×4、01H×4、02H×4、……、07H×4”单元写入 8 个数，偏移地址是从 00 开始。
- (4) 运行程序，待程序运行停止后。

(5) 在“扩展存储区数据显示窗口”中的偏移地址栏中输入 $08H \times 4$ ，即就是 20，并点击“读存储器”按钮，查看 DMA 传输结果，是否与前面写入的数据相同，可反复验证。

注：本实验中，DMA 传数只对存储器的低 8 位进行操作，所以，事先写入及运行后查看存储器的数据时，连续的地址空间都必须是 4 的倍数。想想看，是不是这样？

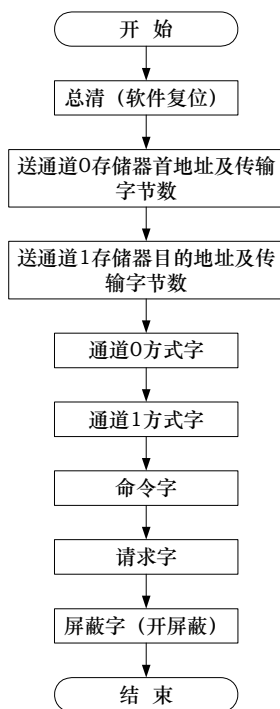


图 4-5-10 DMA 实验流程图

实验程序清单 (T8237-1. ASM)

```

IOY0      EQU 3000H      ;IOY0 起始地址
MY8237_0   EQU IOY0+00H*4 ;通道 0 当前地址寄存器
MY8237_1   EQU IOY0+01H*4 ;通道 0 当前字节计数寄存器
MY8237_2   EQU IOY0+02H*4 ;通道 1 当前地址寄存器
MY8237_3   EQU IOY0+03H*4 ;通道 1 当前字节计数寄存器
MY8237_8   EQU IOY0+08H*4 ;写命令寄存器/读状态寄存器
MY8237_9   EQU IOY0+09H*4 ;请求寄存器
MY8237_B   EQU IOY0+0BH*4 ;工作方式寄存器
MY8237_D   EQU IOY0+0DH*4 ;写总清命令/读暂存寄存器
MY8237_F   EQU IOY0+0FH*4 ;屏蔽位寄存器

STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
CODE SEGMENT
    ASSUME CS:CODE
START:  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, 00H
OUT DX, AL
MOV AL, 00H
OUT DX, AL
MOV DX, MY8237_2    ;写通道 1 当前地址寄存器
MOV AL, 08H
OUT DX, AL
MOV AL, 00H
OUT DX, AL
MOV DX, MY8237_1    ;写通道 0 当前字节计数寄存器
MOV AL, 07H
OUT DX, AL
MOV AL, 00H
OUT DX, AL
MOV DX, MY8237_3    ;写通道 1 当前字节计数寄存器
MOV AL, 07H
OUT DX, AL
MOV AL, 00H
OUT DX, AL
MOV DX, MY8237_B    ;写通道 0 工作方式寄存器
MOV AL, 88H
OUT DX, AL
MOV AL, 85H        ;写通道 1 工作方式寄存器
OUT DX, AL
MOV DX, MY8237_8    ;写命令寄存器
MOV AL, 81H
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

```

2. I/O 到存储器 DMA 传输实验

编写程序，在实验 1 基础上增加 8255 初始化为 B 口输入，A 口输出。B 口输入数据由波动开关模拟。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) 实验接线图如图 4-5-11 所示，按图接线。
- (2) 运行 Tdpt 集成操作软件，编写程序，编译、链接。
- (3) 拨动 B 口所连开关组，设置好一个数据。
- (4) 运行程序，待程序运行停止后。

(5) 在“扩展存储区数据显示窗口”中的偏移地址栏中输入“44”，并点击“读存储器”按钮，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 44H，所以，数据传输到扩展存储器偏移为 44H 地址单元。对于 8237 来讲，实际偏移地址为 11H。想想看，是不是这样？

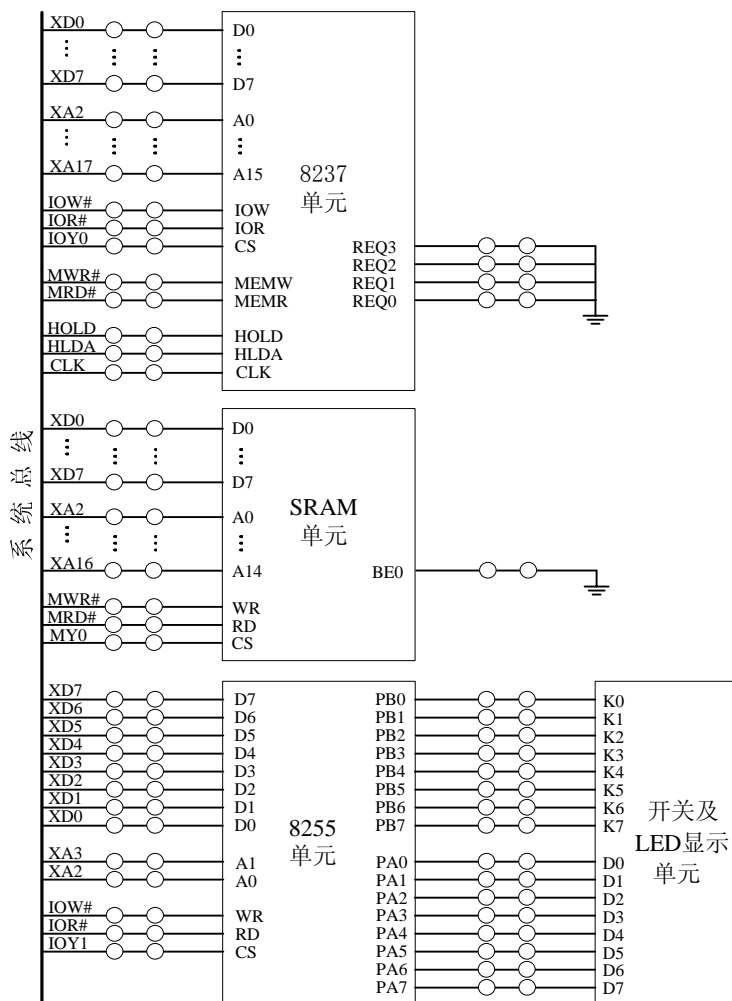


图 4-5-11 8237 实现存储器与 I/O 间 DMA 传输实验接线图

实验程序清单 (T8237-2. ASM)

```

IOY0      EQU 3000H      ;IOY0 起始地址
IOY1      EQU 3040H      ;IOY1 起始地址
MY8237_0  EQU IOY0+00H*4 ;通道 0 当前地址寄存器
MY8237_1  EQU IOY0+01H*4 ;通道 0 当前字节计数寄存器
MY8237_2  EQU IOY0+02H*4 ;通道 1 当前地址寄存器
MY8237_3  EQU IOY0+03H*4 ;通道 1 当前字节计数寄存器
MY8237_8  EQU IOY0+08H*4 ;写命令寄存器/读状态寄存器
MY8237_9  EQU IOY0+09H*4 ;请求寄存器
MY8237_B  EQU IOY0+0BH*4 ;工作方式寄存器
  
```

```

MY8237_D    EQU  IOY0+0DH*4    ;写总清命令/读暂存寄存器
MY8237_F    EQU  IOY0+0FH*4    ;屏蔽位寄存器
MY8255_A    EQU  IOY1+00H*4    ;8255 的 A 口地址
MY8255_B    EQU  IOY1+01H*4    ;8255 的 B 口地址
MY8255_C    EQU  IOY1+02H*4    ;8255 的 C 口地址
MY8255_MODE EQU  IOY1+03H*4    ;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, 11H
        OUT DX, AL
        MOV AL, 00H
        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

编写程序，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送到 A 口，由数码二极管组显示。

- (1) 实验接线图如图 4-5-11 所示，按图接线。
- (2) 运行 Tdpt 集成操作软件，编写程序，编译、链接。
- (3) 打开软件中的“扩展存储区数据显示窗口”，设置扩展存储单元偏移为 40H 中的数据。
- (4) 运行程序，待程序运行停止后。
- (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 40H，所以，我们设置是

扩展存储单元中偏移为 40H 的数据。对于 8237 来讲，实际偏移地址为 10H。想想看，是不是这样？

实验程序清单（T8237-3. ASM）

```
IOY0      EQU 3000H      ;IOY0 起始地址
IOY1      EQU 3040H      ;IOY1 起始地址
MY8237_0  EQU IOY0+00H*4 ;通道 0 当前地址寄存器
MY8237_1  EQU IOY0+01H*4 ;通道 0 当前字节计数寄存器
MY8237_2  EQU IOY0+02H*4 ;通道 1 当前地址寄存器
MY8237_3  EQU IOY0+03H*4 ;通道 1 当前字节计数寄存器
MY8237_8  EQU IOY0+08H*4 ;写命令寄存器/读状态寄存器
MY8237_9  EQU IOY0+09H*4 ;请求寄存器
MY8237_B  EQU IOY0+0BH*4 ;工作方式寄存器
MY8237_D  EQU IOY0+0DH*4 ;写总清命令/读暂存寄存器
MY8237_F  EQU IOY0+0FH*4 ;屏蔽位寄存器
MY8255_A  EQU IOY1+00H*4 ;8255 的 A 口地址
MY8255_B  EQU IOY1+01H*4 ;8255 的 B 口地址
MY8255_C  EQU IOY1+02H*4 ;8255 的 C 口地址
MY8255_MODE EQU IOY1+03H*4 ;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, 10H
        OUT DX, AL
        MOV AL, 00H
        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
```

4.6 8254 定时/计数器应用实验

4.6.1 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。

4.6.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.6.3 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‘5’。
2. 定时应用实验。编写程序，应用 8254 的定时功能，产生一个 1Hz 的方波。

4.6.4 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 4-6-1 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。

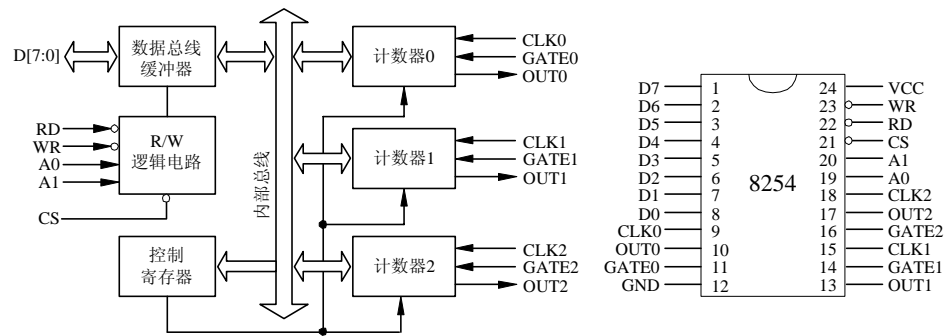


图 4-6-1 8254 的内部接口和引脚

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 4-6-1~4-6-3 所示。

表 4-6-1 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 4-6-2 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 4-6-3 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：

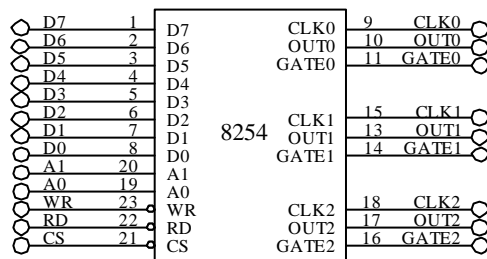


图 4-6-2 8254 实验电路原理图

4.6.5 实验步骤

1. 计数应用实验

编写程序，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 INTR1，每当 KK1+ 按动 5 次后产生中断请求，在屏幕上显示字符“5”。

实验步骤：

- (1) 实验接线如图 4-6-3 所示。
- (2) 运行 Tdpit 集成操作软件，根据实验内容，编写实验程序，编译、链接。
- (3) 运行程序，按动 KK1+ 产生单次脉冲，观察实验现象。
- (4) 改变计数值，验证 8254 的计数功能。

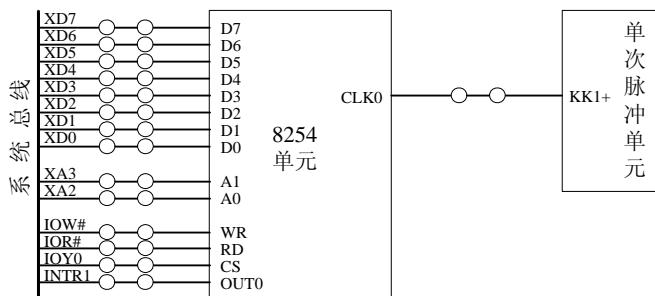


图 4-6-3 8254 计数应用实验接线图

实验程序清单（T8254-1. ASM）

```
INTR_IVADD    EQU    003CH        ;INTR 对应的中断矢量地址
IOY0          EQU    3000H        ;片选 IOY0 对应的端口始地址
MY8254_COUNT0 EQU    IOY0+00H*4   ;8254 计数器 0 端口地址
MY8254_COUNT1 EQU    IOY0+01H*4   ;8254 计数器 1 端口地址
MY8254_COUNT2 EQU    IOY0+02H*4   ;8254 计数器 2 端口地址
MY8254_MODE   EQU    IOY0+03H*4   ;8254 控制寄存器端口地址
STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT
CS_BAK    DW    ?                ;保存 INTR 原中断处理程序入口段地址的变量
```

```

IP_BAK    DW    ?                ;保存 INTR 原中断处理程序入口偏移地址的变量
IM_BAK    DB    ?                ;保存 INTR 原中断屏蔽字的变量
STR1      DB    'COUNT: $'      ;显示的字符串
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
      MOV DS, AX
      CLI
      MOV AX, 0000H                ;替换 INTR 的中断矢量
      MOV ES, AX
      MOV DI, INTR_IVADD
      MOV AX, ES:[DI]
      MOV IP_BAK, AX                ;保存 INTR 原中断处理程序入口偏移地址
      MOV AX, OFFSET MYISR
      MOV ES:[DI], AX                ;设置当前中断处理程序入口偏移地址
      ADD DI, 2
      MOV AX, ES:[DI]
      MOV CS_BAK, AX                ;保存 INTR 原中断处理程序入口段地址
      MOV AX, SEG MYISR
      MOV ES:[DI], AX                ;设置当前中断处理程序入口段地址
      IN AL, 21H
      MOV IM_BAK, AL                ;保存 INTR 原中断屏蔽字
      AND AL, 7FH
      OUT 21H, AL
      STI
      MOV DX, OFFSET STR1          ;显示字符串
      MOV AH, 9
      INT 21H
      MOV DX, MY8254_MODE          ;初始化 8254 工作方式
      MOV AL, 10H                  ;计数器 0, 方式 0
      OUT DX, AL
      MOV DX, MY8254_COUNT0        ;装入计数初值
      MOV AL, 4
      OUT DX, AL
WAIT1: MOV AH, 1                    ;判断是否有按键按下
      INT 16H
      JZ WAIT1                      ;无按键则跳回继续等待, 有则退出
QUIT:  CLI
      MOV AX, 0000H                ;恢复 INTR 原中断矢量
      MOV ES, AX
      MOV DI, INTR_IVADD
      MOV AX, IP_BAK                ;恢复 INTR 原中断处理程序入口偏移地址
      MOV ES:[DI], AX
      ADD DI, 2
      MOV AX, CS_BAK                ;恢复 INTR 原中断处理程序入口段地址
      MOV ES:[DI], AX
      MOV AL, IM_BAK                ;恢复 INTR 原中断屏蔽寄存器的屏蔽字
      OUT 21H, AL
      STI
      MOV AX, 4C00H                ;返回到 DOS

```

```

        INT 21H
MYISR PROC NEAR                ;中断处理程序 MYISR
        PUSH AX
        MOV AL, 35H
        MOV AH, 0EH
        INT 10H
        MOV AL, 20H
        INT 10H
        MOV DX, MY8254_COUNT0    ;重装计数初值
        MOV AL, 4
        OUT DX, AL
OVER:   MOV AL, 20H                ;向 PC 机内部 8259 发送中断结束命令
        OUT 20H, AL
        POP AX
        IRET
MYISR ENDP
CODE ENDS
        END START


```

2. 定时应用实验

编写程序，将 8254 的计数器 2 设置为方式 3，用信号源 1.8432MHz 作为 CLK2 时钟，计数初值为 100，相当对 CLK2 进行 100 分频。在 OUT2 输出频率为 18.432KHz 的时钟。将 OUT2 连接到计数器 0 的 CLK0，设置计数器 0 也工作在方式 3，计数初值为 18432，相当是进行 18432 分频。则在 OUT0 得到 1Hz 的输出。

实验步骤：

- (1) 接线图如图 4-6-4 所示，按图接线。
- (2) 运行 Tdpt 集成操作软件，根据实验内容，编写实验程序，编译、链接。
- (3) 单元中 GATE0 已经连接了一个上拉电阻，所以 GATE0 不用连接。
- (4) 运行实验程序，用示波器显示功能观察 OUT0 输出 1HZ 是否正确。

(5) 用软件所带示波器观测的方法：点击快捷工具栏上 “” 按钮，启动示波器显示窗口，即可观察波形显示。

实验程序清单 (T8254-2. ASM)

```

IOY0      EQU    3000H          ;片选 IOY0 对应的端口始地址
MY8254_COUNT0 EQU    IOY0+00H*4 ;8254 计数器 0 端口地址
MY8254_COUNT1 EQU    IOY0+01H*4 ;8254 计数器 1 端口地址
MY8254_COUNT2 EQU    IOY0+02H*4 ;8254 计数器 2 端口地址
MY8254_MODE EQU    IOY0+03H*4   ;8254 控制寄存器端口地址
STACK1 SEGMENT STACK
        DW 256 DUP(?)
STACK1 ENDS
CODE SEGMENT
        ASSUME CS:CODE
START:  MOV DX, MY8254_MODE      ;初始化 8254 工作方式
        MOV AL, 0B6H            ;计数器 2，方式 3
        OUT DX, AL

```

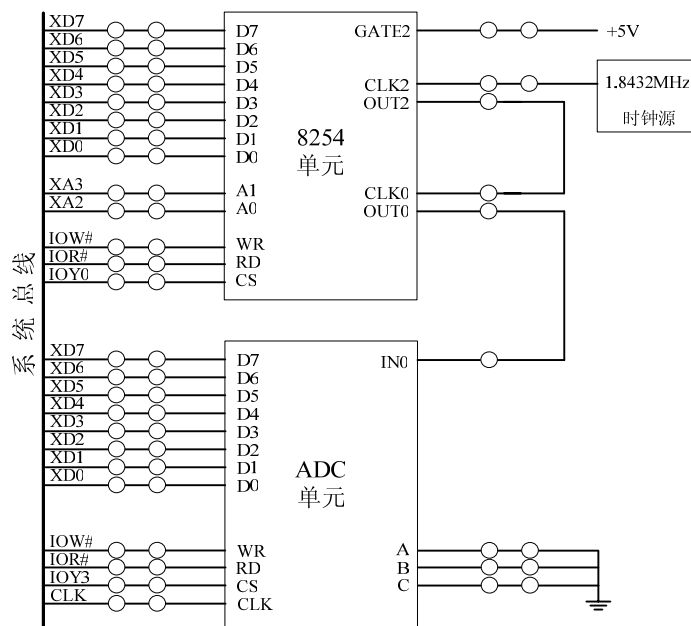


图 4-6-4 8254 定时应用实验接线图

```

MOV DX, MY8254_COUNT2      ;装入计数初值
MOV AL, 64H                 ;100 分频
OUT DX, AL
MOV AL, 00H
OUT DX, AL
MOV DX, MY8254_MODE         ;初始化 8254 工作方式
MOV AL, 36H                 ;计数器 0, 方式 3
OUT DX, AL
MOV DX, MY8254_COUNT0      ;装入计数初值
MOV AL, 00H                 ;18432 分频
OUT DX, AL
MOV AL, 48H
OUT DX, AL
QUIT: MOV AX, 4C00H          ;结束程序退出
      INT 21H
CODE ENDS
      END START

```

4.7 8255 并行接口实验

4.7.1 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。

4.7.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.7.3 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。

4.7.4 实验原理

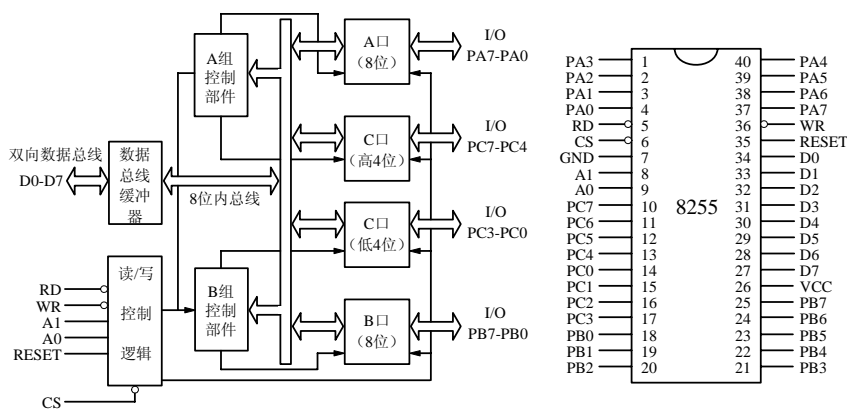


图 4-7-1 8255 内部结构及外部引脚图

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 4-7-1 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 4-7-2 所示。

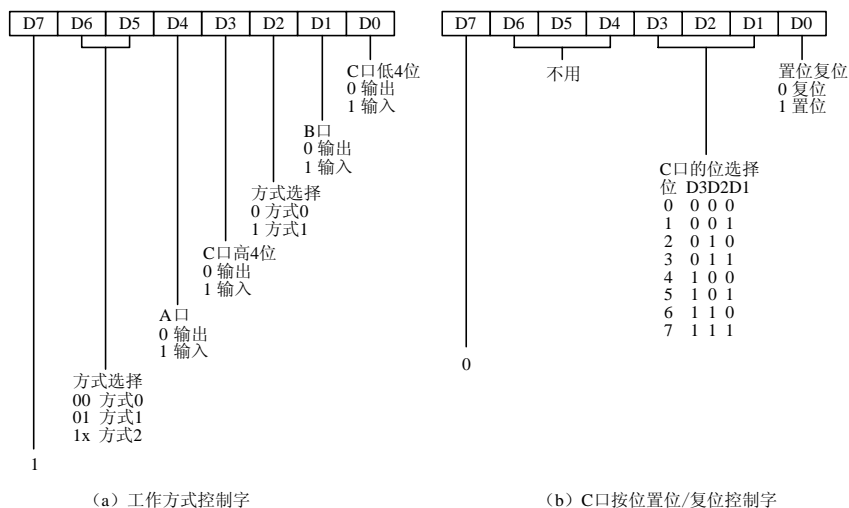


图 4-7-2 8255 控制字格式

8255 实验单元电路图如图 4-7-3 所示：

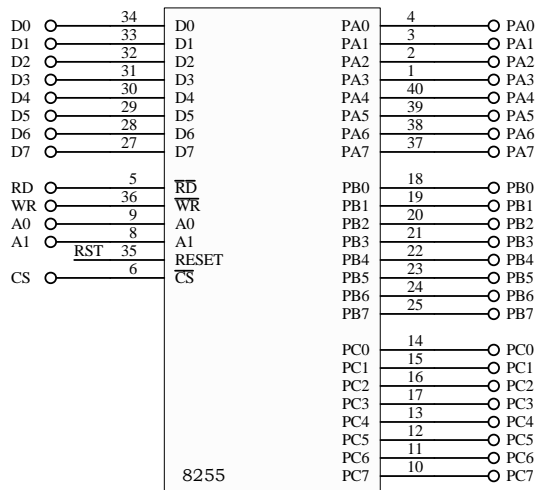


图 4-7-3 8255 实验单元电路图

4.7.5 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。具体实验步骤如下述：

- (1) 实验接线图如图 4-7-4 所示，按图连接实验线路图。
- (2) 运行 Tdpt 集成操作软件，根据实验内容，编写实验程序，编译、链接。
- (3) 运行程序，改变拨动开关，同时观察 LED 显示，验证程序功能。

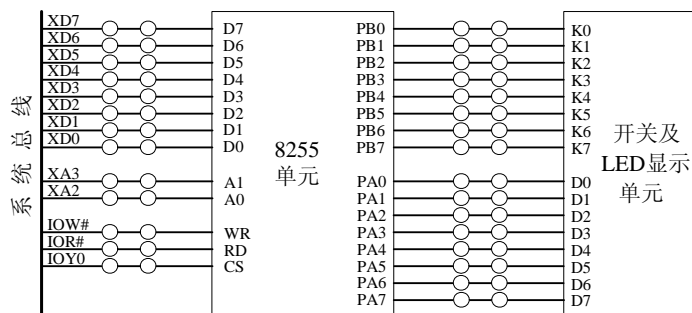


图 4-7-4 8255 基本输入输出实验接线图

实验程序清单（T8255-1. ASM）

```

IOY0      EQU    3000H           ;片选 IOY0 对应的端口始地址
MY8255_A  EQU    IOY0+00H*4      ;8255 的 A 口地址
MY8255_B  EQU    IOY0+01H*4      ;8255 的 B 口地址
MY8255_C  EQU    IOY0+02H*4      ;8255 的 C 口地址
MY8255_MODE EQU    IOY0+03H*4    ;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, 82H                 ;工作方式 0, A 口输出, B 口输入
      OUT DX, AL
LOOP1: MOV DX, MY8255_B          ;读 B 口
      IN AL, DX
      MOV DX, MY8255_A           ;写 A 口
      OUT DX, AL
      MOV AH, 1                  ;判断是否有按键按下
      INT 16H
      JZ LOOP1                  ;无按键则跳回继续循环, 有则退出
QUIT:  MOV AX, 4C00H             ;结束程序退出
      INT 21H
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 4-5-5 所示。实验步骤如下所述：

- (1) 实验接线图如图 4-7-5 所示，按图连接实验线路图。
- (2) 运行 Tdpt 集成操作软件，根据实验内容，编写实验程序，编译、链接。
- (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。

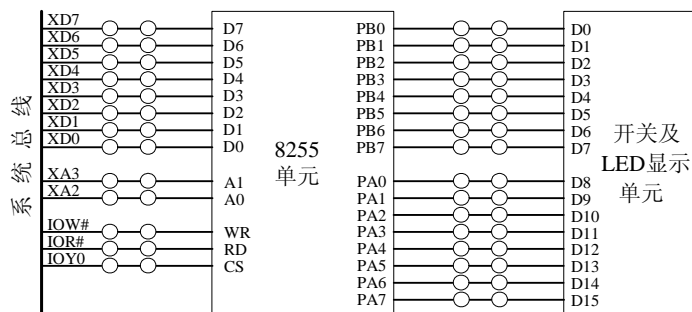


图 4-7-5 8255 流水灯实验接线图

实验程序清单 (T8255-2. ASM)

```

IOY0      EQU    3000H          ;片选 IOY0 对应的端口始地址
MY8255_A  EQU    IOY0+00H*4     ;8255 的 A 口地址
MY8255_B  EQU    IOY0+01H*4     ;8255 的 B 口地址
MY8255_C  EQU    IOY0+02H*4     ;8255 的 C 口地址
MY8255_MODE EQU    IOY0+03H*4   ;8255 的控制寄存器地址

STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT
LA      DB  ?                  ;定义数据变量
LB      DB  ?
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
    MOV DS, AX
    MOV DX, MY8255_MODE        ;定义 8255 工作方式
    MOV AL, 80H                ;工作方式 0, A 口和 B 口为输出
    OUT DX, AL
    MOV DX, MY8255_A           ;写 A 口发出的起始数据
    MOV AL, 80H
    OUT DX, AL
    MOV LA, AL
    MOV DX, MY8255_B           ;写 B 口发出的起始数据

```

```

MOV AL, 01H
OUT DX, AL
MOV LB, AL
LOOP1: CALL DALLY
MOV AL, LA           ;将 A 口起始数据右移再写入 A 口
ROR AL, 1
MOV LA, AL
MOV DX, MY8255_A
OUT DX, AL
MOV AL, LB           ;将 B 口起始数据左移再写入 B 口
ROL AL, 1
MOV LB, AL
MOV DX, MY8255_B
OUT DX, AL
MOV AH, 1           ;判断是否有按键按下
INT 16H
JZ LOOP1            ;无按键则跳回继续循环，有则退出
QUIT: MOV AX, 4C00H  ;结束程序退出
INT 21H
DALLY PROC NEAR     ;软件延时子程序
PUSH CX
PUSH AX
MOV CX, 0FFFFH
D1: MOV AX, 0FFFFH
D2: DEC AX
JNZ D2
LOOP D1
POP AX
POP CX
RET
DALLY ENDP
CODE ENDS
END START

```

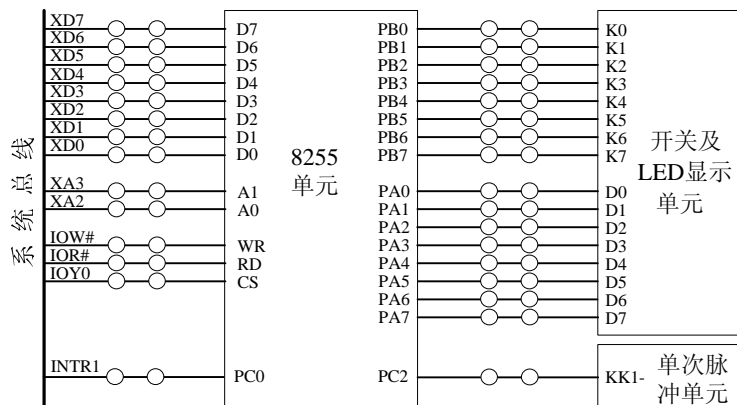


图 4-7-6 8255 方式 1 输入输出实验接线图

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。具体实验步骤如下述:

(1) 实验接线图如图 4-7-6 所示, 按图连接实验线路图。

(2) 运行 Tdptit 集成操作软件, 根据实验内容, 编写实验程序, 编译、链接。

(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 (T8255-3. ASM)

```
INTR_IVADD EQU 003CH ;INTR 对应的中断矢量地址
IOY0 EQU 3000H ;片选 IOY0 对应的端口始地址
MY8255_A EQU IOY0+00H*4 ;8255 的 A 口地址
MY8255_B EQU IOY0+01H*4 ;8255 的 B 口地址
MY8255_C EQU IOY0+02H*4 ;8255 的 C 口地址
MY8255_MODE EQU IOY0+03H*4 ;8255 的控制寄存器地址
STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT
MES DB 'Press any key to exit!', 0AH, 0DH, 0AH, 0DH, '$'
CS_BAK DW ? ;保存 INTR 原中断处理程序入口段地址的变量
IP_BAK DW ? ;保存 INTR 原中断处理程序入口偏移地址的变量
IM_BAK DB ? ;保存 INTR 原中断屏蔽字的变量
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
    MOV DS, AX
    MOV DX, OFFSET MES ;显示退出提示
    MOV AH, 09H
    INT 21H
    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
    CLI
    MOV AX, 0000H ;替换 INTR 的中断矢量
    MOV ES, AX
    MOV DI, INTR_IVADD
    MOV AX, ES:[DI]
    MOV IP_BAK, AX ;保存 INTR 原中断处理程序入口偏移地址
    MOV AX, OFFSET MYISR
    MOV ES:[DI], AX ;设置当前中断处理程序入口偏移地址
    ADD DI, 2
    MOV AX, ES:[DI]
    MOV CS_BAK, AX ;保存 INTR 原中断处理程序入口段地址
    MOV AX, SEG MYISR
    MOV ES:[DI], AX ;设置当前中断处理程序入口段地址
```

```

        IN  AL, 21H                ;设置中断屏蔽寄存器，打开 INTR 的屏蔽位
        MOV IM_BAK, AL            ;保存 INTR 原中断屏蔽字
        AND AL, 7FH
        OUT 21H, AL
        STI
WAIT1:  MOV AH, 1                  ;判断是否有按键按下
        INT 16H
        JZ  WAIT1                  ;无按键则跳回继续等待，有则退出
QUIT:   CLI
        MOV AX, 0000H              ;恢复 INTR 原中断矢量
        MOV ES, AX
        MOV DI, INTR_IVADD
        MOV AX, IP_BAK              ;恢复 INTR 原中断处理程序入口偏移地址
        MOV ES: [DI], AX
        ADD DI, 2
        MOV AX, CS_BAK              ;恢复 INTR 原中断处理程序入口段地址
        MOV ES: [DI], AX
        MOV AL, IM_BAK              ;恢复 INTR 原中断屏蔽寄存器的屏蔽字
        OUT 21H, AL
        STI
        MOV AX, 4C00H              ;返回到 DOS
        INT 21H
MYISR PROC NEAR                    ;中断处理程序 MYISR
        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
MYISR ENDP
CODE ENDS
        END START

```

4.8 8251 串行控制器应用实验

4.8.1 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。
3. 掌握使用 8251 实现双机通讯的软件编程和电路连接。

4.8.2 实验设备

PC 机两台，TD-PITD+实验装置两套。

4.8.3 实验内容

1. 串行通讯基础实验，连续向串口发送一个数，使用示波器测量串口输出的波形，分析串行数据格式。
2. 自收自发实验。将一串数据从发送口发送出去，然后自接收并在屏幕上显示出来。
3. 双机通讯应用实验。编写两个应用程序，一个给发送机使用，完成数据的发送，另一个给接收机使用，完成数据的接收。

4.8.4 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 4-8-1 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 4-8-2 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。

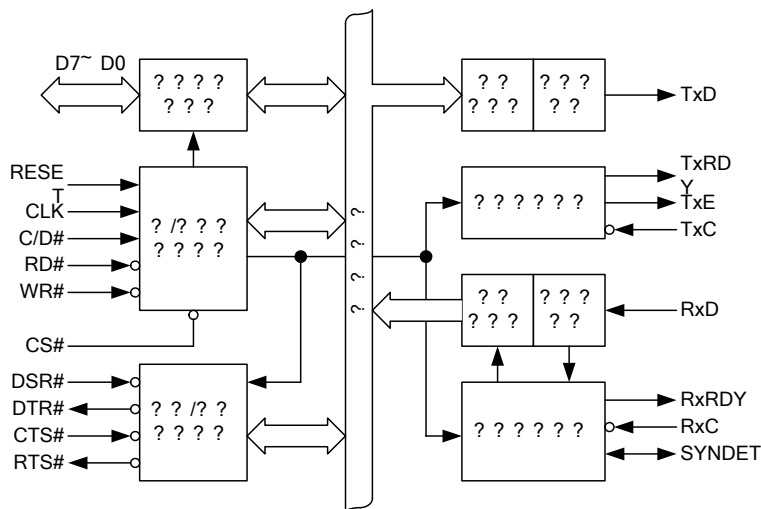


图 4-8-1 8251 内部结构图

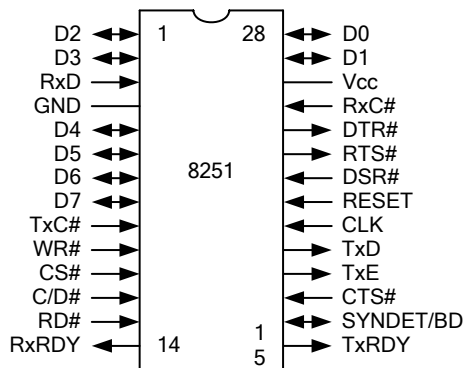


图 4-8-2 8251 外部引脚图

3. 8251 在异步方式下的 TXD 信号上的数据传输格式

图 4-8-3 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。

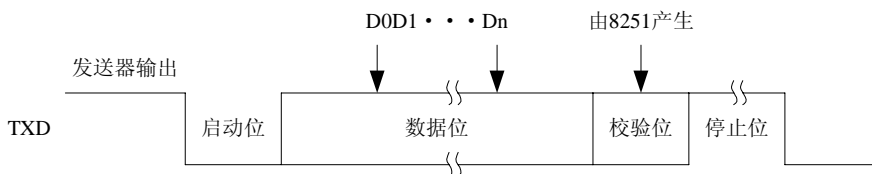


图 4-8-3 8251 工作在异步方式下 TXD 信号的数据传输格式

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 4-8-4 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≠00)	X0=无校验 01=奇校验 11=偶校验		00=5 位 01=6 位 10=7 位 11=8 位		异步 00=不用 01=01 10=16 11=64	同步 00=同步 方式标志
X0=内同步	00=不用						
X1=外同步	01=1 位						
0X=双同步	10=1.5 位						
1X=单同步	11=2 位						

图 4-8-4 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 4-8-5 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 1=使错误标志 PE、OE、FE 复位	发中止字符 1=使 TXD 为低电平 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 4-8-5 8251 命令控制字格式

(3) 状态字

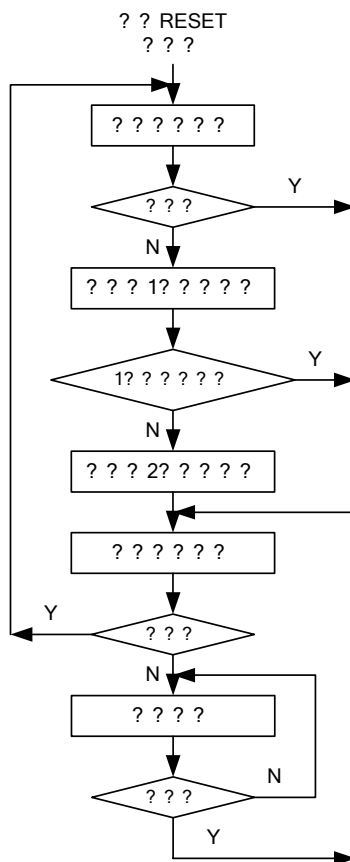
CPU 通过状态字来了解 8251 当前的工作状态，以决定下一步的操作，8251 的状态字如图 4-8-6 所示。

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 表明发送缓冲区空。

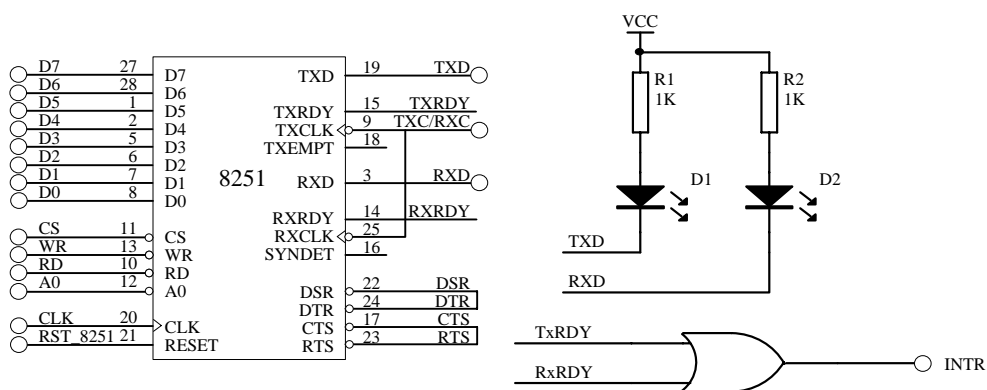
图 4-8-6 8251 状态字格式

(4) 系统初始化

8251 的初始化和操作流程如图 4-8-7 所示。



5. 8251 实验单元电路图



4.8.5 实验步骤

1. 串行通讯基础实验

对 8251 进行编程，不断向发送寄存器写数，用示波器观察 TXD 信号脉冲变化，仔细分析波形，理解波形原理。串行传输的数据格式可设定如下：RXC/TXC 频率为 32Hz，每个字节有一个逻辑“0”的起始位，8 位数据位，1 位逻辑“1”的停止位，如图 4-8-9 所示。

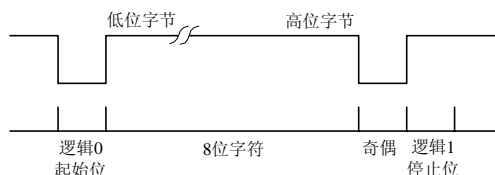


图 4-8-9 串行传输的数据格式

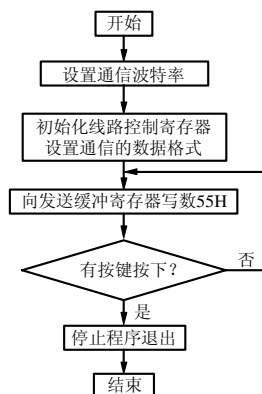



图 4-8-10 串行通讯基础实验程序流程图

具体实验步骤如下述。

- (1) 实验接线图如图 4-8-11 所示，按图连接实验线路图。
- (2) 运行 Tdpt 集成操作软件，参考图 4-8-10 所示流程图，编写实验程序，编译、链接。
- (3) 运行程序，用示波器观察 TXD 输出波形。

(4) 用软件所带示波器观测的方法：点击快捷工具栏上“”按钮，启动示波器显示窗口，即可观察波形显示。

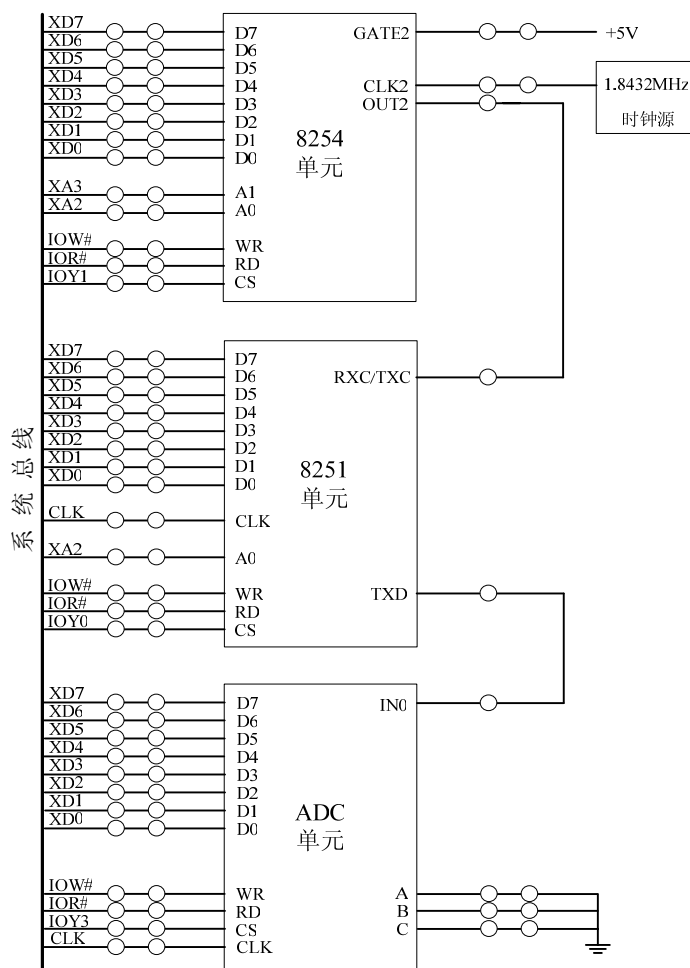


图 4-8-11 8251 数据串行传输实验线路图

实验参考例程 (T8251-1. ASM)

```

IOY0      EQU    3000H
IOY1      EQU    3040H
MY8251_DATA EQU    IOY0+00H*4    ;8251 数据寄存器
MY8251_MODE EQU    IOY0+01H*4    ;8251 方式控制寄存器
MY8254_COUNT2 EQU    IOY1+02H*4    ;8254 计数器 2 端口地址
MY8254_MODE EQU    IOY1+03H*4    ;8254 控制寄存器端口地址

SSTACK    SEGMENT STACK
          DW 64 DUP(?)
SSTACK    ENDS

CODE SEGMENT
          ASSUME CS:CODE

START:    CALL INIT
A1:       CALL SEND
          MOV CX, 0600H
A2:       MOV AX, 6000H

```

```

A3:      DEC AX
          JNZ A3
          LOOP A2
          MOV AH, 1                ;判断是否有按键按下
          INT 16H
          JZ A1                    ;无按键则跳回继续循环，有则退出
QUIT:    MOV AX, 4C00H             ;结束程序退出
          INT 21H
INIT:     MOV AL, 0B6H             ;初始化 8254，设置通讯时钟
          MOV DX, MY8254_MODE
          OUT DX, AL
          MOV AL, 00H
          MOV DX, MY8254_COUNT2
          OUT DX, AL
          MOV AL, 0E1H
          OUT DX, AL
          CALL RESET               ;对 8251 进行初始化
          CALL DALLY
          MOV AL, 7EH
          MOV DX, MY8251_MODE      ;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, MY8251_MODE      ;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, MY8251_MODE
          OUT DX, AL
          MOV AL, 55H
          MOV DX, MY8251_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。实验要求向发送端口发送一组字符串“Self-Communication by 8251!”，然后从接收端口进行接收，将接收到的数据显示在显示器屏幕上。

具体实验步骤如下：

- (1) 参考实验接线图如图 4-8-12 所示，按图连接实验线路。
- (2) 运行 Tdptit 集成操作软件，根据实验内容，编写实验程序，编译、链接。
- (3) 运行程序，观察屏幕数据显示，看接收是否正确。

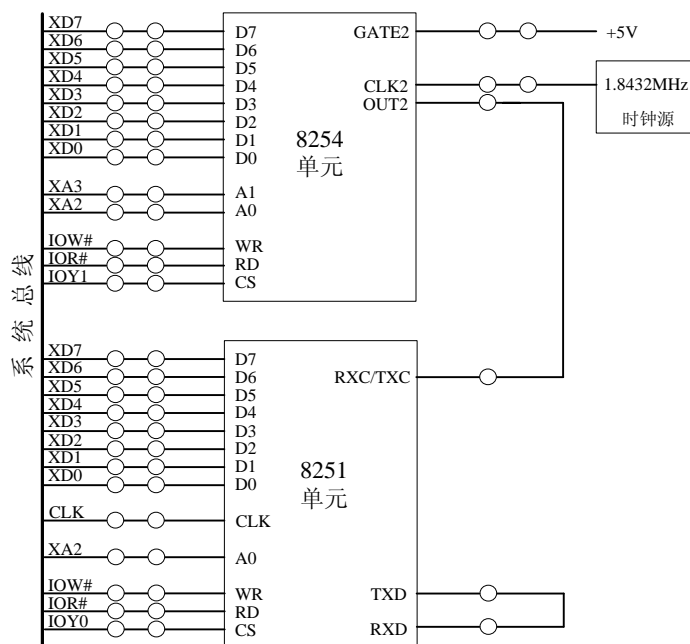


图 4-8-12 自收自发实验接线图

实验参考例程 (T8251-2. ASM)

```

IOY0      EQU    3000H
IOY1      EQU    3040H
MY8251_DATA EQU    IOY0+00H*4    ;8251 数据寄存器
MY8251_MODE EQU    IOY0+01H*4    ;8251 方式控制寄存器
MY8254_COUNT2 EQU    IOY1+02H*4    ;8254 计数器 2 端口地址
MY8254_MODE EQU    IOY1+03H*4    ;8254 控制寄存器端口地址
SSTACK    SEGMENT STACK
            DW 64 DUP(?)
SSTACK    ENDS
DATA SEGMENT

```

```

STR1      DB  'Self-Communication by 8251!$'      ;字符串
DATA ENDS
CODE SEGMENT
        ASSUME CS:CODE
START:   MOV AX, DATA
        MOV DS, AX
        MOV AL, 0B6H                ;初始化 8254, 得到收发时钟
        MOV DX, MY8254_MODE
        OUT DX, AL
        MOV AL, 0CH
        MOV DX, MY8254_COUNT2
        OUT DX, AL
        MOV AL, 00H
        OUT DX, AL
        CALL INIT                    ;初始化 8251
        CALL DALLY
        MOV AL, 7EH
        MOV DX, MY8251_MODE
        OUT DX, AL                  ;8251 方式字
        CALL DALLY
        MOV AL, 34H
        OUT DX, AL                  ;8251 控制字
        CALL DALLY
        MOV CX, 001BH                ;10 个数
        MOV BX, OFFSET STR1
A1:      MOV AL, 37H
        MOV DX, MY8251_MODE
        OUT DX, AL
        MOV AL, [BX]
        MOV DX, MY8251_DATA
        OUT DX, AL                  ;发送数据
        MOV DX, MY8251_MODE
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, MY8251_DATA
        IN AL, DX                    ;读取接收到的数据并显示
        MOV DL, AL
        MOV AH, 02H
        INT 21H
        INC BX
        LOOP A1
A4:      MOV AH, 1 ;判断是否有按键按下
        INT 16H
        JZ A4                        ;无按键则跳回继续循环, 有则退出
        MOV AX, 4C00H
        INT 21H
INIT:    MOV AL, 00H ;复位 8251 子程序
        MOV DX, MY8251_MODE

```

```

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：

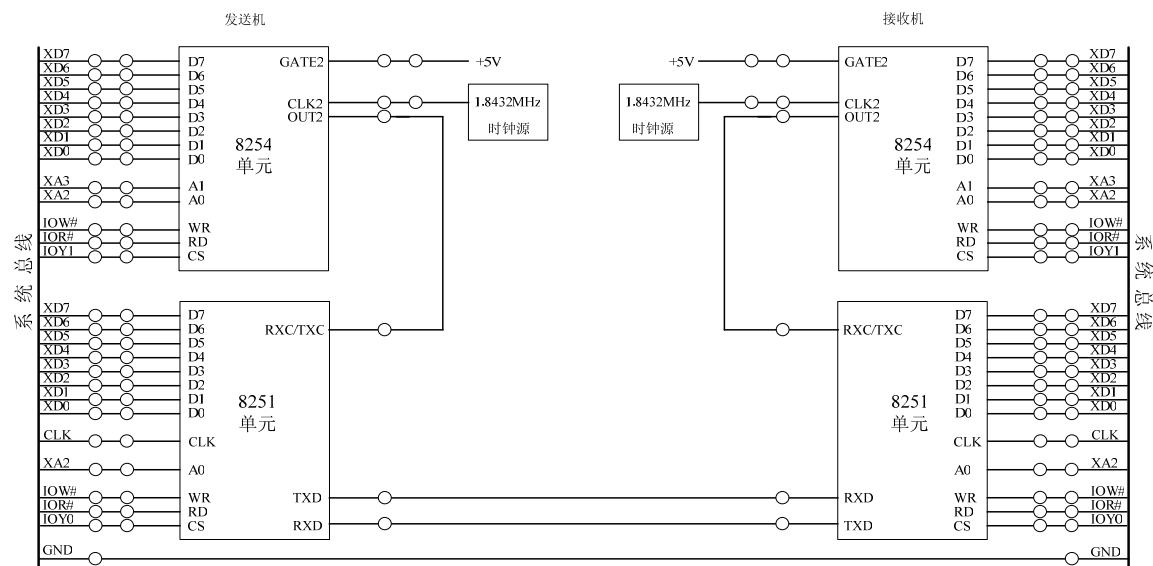


图 4-8-13 双机通讯实验接线图

- (1) 参考实验接线图如图 4-8-13 所示，按图连接实验线路。
- (2) 运行 Tdpit 集成操作软件，根据实验内容，为两台机器分别编写实验程序，编译、链接。
- (3) 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。

(4) 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。

实验参考例程 (接收机) (T8251-3. ASM)

```

IOY0      EQU    3000H
IOY1      EQU    3040H
MY8251_DATA EQU    IOY0+00H*4    ;8251 数据寄存器
MY8251_MODE EQU    IOY0+01H*4    ;8251 方式控制寄存器
MY8254_COUNT2 EQU    IOY1+02H*4    ;8254 计数器 2 端口地址
MY8254_MODE EQU    IOY1+03H*4    ;8254 控制寄存器端口地址
SSTACK    SEGMENT STACK
          DW 64 DUP(?)
SSTACK    ENDS
CODE SEGMENT
          ASSUME CS:CODE
START:    MOV AL, 0B6H              ;初始化 8254
          MOV DX, MY8254_MODE
          OUT DX, AL
          MOV AL, 0CH
          MOV DX, MY8254_COUNT2
          OUT DX, AL
          MOV AL, 00H
          OUT DX, AL
          CALL INIT                  ;初始化 8251
          CALL DALLY
          MOV AL, 7EH
          MOV DX, MY8251_MODE
          OUT DX, AL
          CALL DALLY
          MOV AL, 34H
          OUT DX, AL
          CALL DALLY
          MOV AX, 0E52H              ;输出显示字符 'R'
          INT 10H
          MOV AX, 0E3AH              ;输出显示字符 ':'
          INT 10H
A1:        MOV DX, MY8251_MODE        ;判断是否有数据接收
          IN AL, DX
          AND AL, 02H
          JNZ A11
          MOV AH, 1                  ;判断是否有按键按下
          INT 16H
          JZ A1                      ;无按键则跳回继续循环, 有则退出
QUIT:      MOV AX, 4C00H              ;结束程序退出
          INT 21H
A11:       MOV DX, MY8251_DATA
          IN AL, DX                  ;读取数据并显示
          MOV DL, AL
          MOV AH, 02H
          INT 21H
          JMP A1
INIT:      MOV AL, 00H              ;复位 8251 子程序

```

```

        MOV DX, MY8251_MODE
        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

```

实验参考程序（发送机）（T8251-4. ASM）

```

IOY0      EQU    3000H
IOY1      EQU    3040H
MY8251_DATA EQU    IOY0+00H*4    ;8251 数据寄存器
MY8251_MODE EQU    IOY0+01H*4    ;8251 方式控制寄存器
MY8254_COUNT2 EQU    IOY1+02H*4    ;8254 计数器 2 端口地址
MY8254_MODE EQU    IOY1+03H*4    ;8254 控制寄存器端口地址
SSTACK    SEGMENT STACK
          DW 64 DUP(?)
SSTACK    ENDS
DATA SEGMENT
AA    DB  2FH
DATA ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS:DATA
START:  MOV AX, DATA
        MOV DS, AX
        MOV AL, 0B6H            ;初始化 8254, 得到收发时钟
        MOV DX, MY8254_MODE
        OUT DX, AL
        MOV AL, 0CH
        MOV DX, MY8254_COUNT2
        OUT DX, AL
        MOV AL, 00H
        OUT DX, AL
        CALL INIT                ;初始化 8251
        CALL DALLY
        MOV AL, 7EH
        MOV DX, MY8251_MODE
        OUT DX, AL                ;8251 方式字

```



```

        CALL DALLY
        MOV AL, 34H
        OUT DX, AL                ;8251 控制字
        CALL DALLY
A1:      INC AA
        MOV AL, AA
        CALL SEND
        CALL DALLY
        CMP AA, 39H
        JNZ A1
        MOV AX, 4C00H
        INT 21H
INIT:    MOV AL, 00H              ;复位 8251 子程序
        MOV DX, MY8251_MODE
        OUT DX, AL
        CALL DALLY
        OUT DX, AL
        CALL DALLY
        OUT DX, AL
        CALL DALLY
        MOV AL, 40H
        OUT DX, AL
        RET
DALLY PROC NEAR                  ;软件延时子程序
        PUSH CX
        PUSH AX
        MOV CX, 0FFFFH
D1:      MOV AX, 0FFFFH
D2:      DEC AX
        JNZ D2
        LOOP D1
        POP AX
        POP CX
        RET
DALLY ENDP
SEND:    PUSH DX                  ;数据发送子程序
        PUSH AX
        MOV AL, 31H
        MOV DX, MY8251_MODE
        OUT DX, AL
        POP AX
        MOV DX, MY8251_DATA
        OUT DX, AL
        MOV DX, MY8251_MODE
A3:      IN AL, DX
        AND AL, 01H
        JZ A3
        POP DX
        RET
CODE ENDS
        END START

```

4.9 A/D 转换实验

4.9.1 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。

4.9.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.9.3 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。

4.9.4 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 4-9-1 所示，地址信号与选中通道的关系如表 4-9-1 所示。

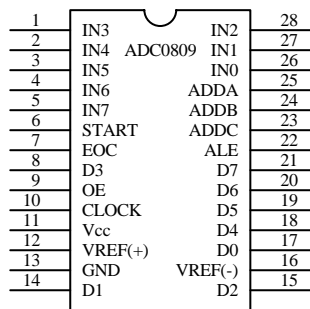


图 4-9-1 ADC0809 外部引脚图

表 4-9-1 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 4-9-2 所示：

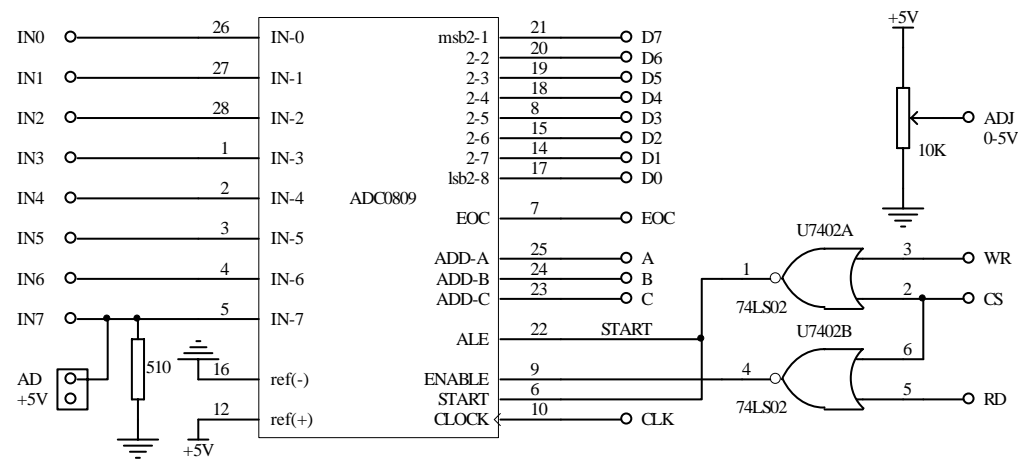


图 4-9-2 模/数转换电路图

4.9.5 实验步骤

- (1) 实验接线图如图 4-9-3 所示, 按图连接实验线路图。
 - (2) 运行 Tdpit 集成操作软件, 根据实验内容, 编写实验程序, 编译、链接。
 - (3) 运行程序, 调节电位器, 观察屏幕上显示的数字量输出。
 - (4) 用万用表测出电位器输出电压, 并记录显示屏上的相应数据。做出转换图, 及 VD-VA。
- 求 ADC0809 芯片的整量化误差。

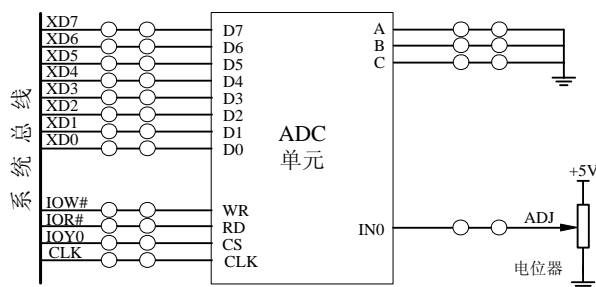


图 4-9-3 A/D 转换实验接线图

实验程序清单 (T0809.ASM)

```

IOY0      EQU    3000H           ;片选 IOY0 对应的端口始地址
AD0809    EQU    IOY0+00H       ;AD0809 的端口地址
STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT
STR1 DB 'AD0809: IN0 $'         ;定义显示的字符串
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
    MOV DS, AX
LOOP1: MOV DX, AD0809            ;启动 A/D 转换
    OUT DX, AL
    CALL DALLY
    MOV DX, OFFSET STR1         ;显示字符串 AD0809: IN0
    MOV AH, 9
    INT 21H
    MOV DX, AD0809              ;读出转换结果
    IN AL, DX
    MOV CH, AL                   ;分析结果进行显示
    AND AL, 0F0H
    MOV CL, 04H
    SHR AL, CL                   ;取出数据的十位
    CMP AL, 09H
    JG A1

```

```

        ADD AL, 30H
        JMP A2
A1:     ADD AL, 37H           ;对 A~F 的处理
A2:     MOV DL, AL           ;对 0~9 的处理
        MOV AH, 02H
        INT 21H
        MOV AL, CH
        AND AL, 0FH         ;取出数据的各位
        CMP AL, 09H
        JG A3
        ADD AL, 30H
        JMP A4
A3:     ADD AL, 37H           ;对 A~F 的处理
A4:     MOV DL, AL           ;对 0~9 的处理
        MOV AH, 02H
        INT 21H
        MOV DL, 0DH         ;回车, 置光标到行首
        MOV AH, 02H
        INT 21H
        MOV AH, 1           ;判断是否有按键按下
        INT 16H
        JZ LOOP1            ;无按键则跳回继续循环, 有则退出
QUIT:   MOV AX, 4C00H        ;结束程序退出
        INT 21H
DALLY PROC NEAR              ;软件延时子程序
        PUSH CX
        PUSH AX
        MOV CX, 4000H
D1:     MOV AX, 0600H
D2:     DEC AX
        JNZ D2
        LOOP D1
        POP AX
        POP CX
        RET
DALLY ENDP
CODE    ENDS
        END START

```

4.10 D/A 转换实验

4.10.1 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。

4.10.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.10.3 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，输入数字量由程序给出。要求产生方波和三角波，并用示波器观察输出模拟信号的波形。

4.10.4 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 I_{out1} 和 I_{out2} 输出，其主要性能参数如表 4-10-1 示，引脚如图 4-10-1 所示。

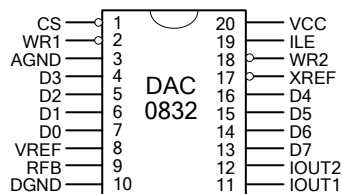


图 4-10-1 DAC0832 引脚图

表 4-10-1 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	$\pm 1\text{LSB}$
数据输入电平	与 TTL 电平兼容

D/A 转换单元实验电路图如图 4-10-2 所示：

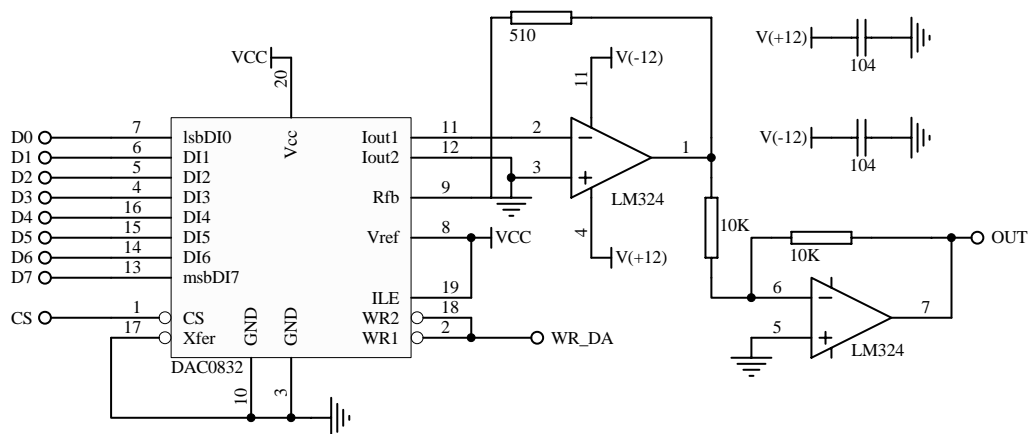


图 4-10-2 D/A 实验单元电路图

4.10.5 实验步骤

(1) 实验接线图如图 4-10-3 所示，按图连接实验线路图。

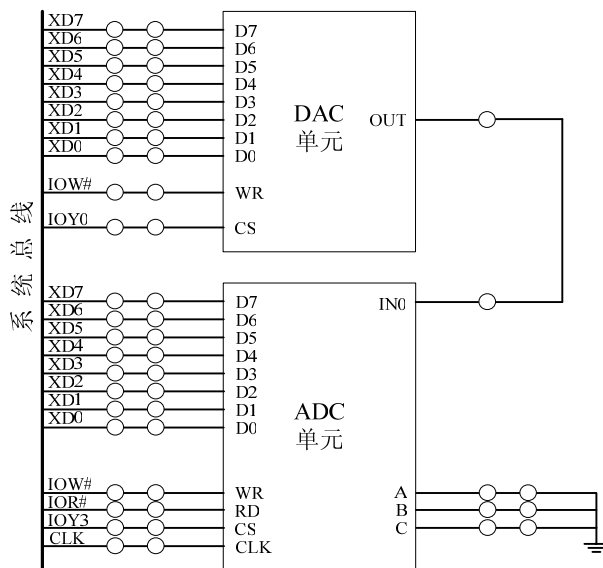



图 4-10-3 D/A 实验接线图

- (2) 运行 Tdpit 集成操作软件，根据实验内容，编写实验程序，编译、链接。
- (3) 运行程序，用示波器观察 DAC 单元的 OUT 输出，观察实验现象。
- (4) 自行编写实验程序，产生三角波形，使用示波器观察输出，验证程序功能。

(5) 用软件所带示波器观测的方法：点击快捷工具栏上“”按钮，启动示波器显示窗口，即可观察波形显示。

实验程序清单 1 (T0832-1. ASM) (产生方波)

```

IOY0      EQU    3000H           ;片选 IOY0 对应的端口始地址
DA0832     EQU    IOY0+00H*4     ;DA0832 的端口地址
STACK1     SEGMENT STACK
            DW 256 DUP(?)
STACK1     ENDS
DATA SEGMENT
STR1       DB    'DA0832: Square Wave $'           ;定义显示的字符串
DATA ENDS
CODE SEGMENT
            ASSUME CS:CODE, DS:DATA
START:     MOV AX, DATA
            MOV DS, AX
            MOV DX, OFFSET STR1           ;显示字符串
            MOV AH, 9
            INT 21H
LOOP1:     MOV DX, DA0832               ;写 00H, 输出低电平
            MOV AL, 00H
            OUT DX, AL
            CALL DALLY
            MOV DX, DA0832               ;写 0FH, 输出高电平
            MOV AL, 0FH
            OUT DX, AL
            CALL DALLY
            MOV AH, 1                     ;判断是否有按键按下
            INT 16H
            JZ  LOOP1                    ;无按键则跳回继续循环, 有则退出
QUIT:      MOV AX, 4C00H                 ;结束程序退出
            INT 21H
DALLY PROC NEAR                          ;软件延时子程序
            PUSH CX
            PUSH AX
            MOV CX, 05000H
D1:         MOV AX, 0F000H
D2:         DEC AX
            JNZ D2
            LOOP D1
            POP AX
            POP CX
            RET
DALLY ENDP
CODE ENDS
            END START

```

实验程序清单 2 (T0832-2. ASM) (产生三角波)

```

IOY0      EQU    3000H           ;片选 IOY0 对应的端口始地址
DA0832     EQU    IOY0+00H*4     ;DA0832 的端口地址
STACK1     SEGMENT STACK
            DW 256 DUP(?)
STACK1     ENDS

```



```

DATA SEGMENT
STR1  DB  'DA0832: Triangle Wave $'      ;定义显示的字符串
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START:  MOV AX, DATA
        MOV DS, AX
        MOV DX, OFFSET STR1      ;显示字符串
        MOV AH, 9
        INT 21H
LOOP1:  MOV AL, 00H              ;D/A 转换起始值
UP:     MOV DX, DA0832           ;启动 D/A 转换
        OUT DX, AL
        CALL DALLY
        INC AL
        CMP AL, 7FH
        JNE UP
DOWN:   MOV DX, DA0832
        OUT DX, AL
        CALL DALLY
        DEC AL
        CMP AL, 00H
        JNE DOWN
        MOV AH, 1                ;判断是否有按键按下
        INT 16H
        JZ  LOOP1                ;无按键则跳回继续循环，有则退出
QUIT:   MOV AX, 4C00H            ;结束程序退出
        INT 21H
DALLY PROC NEAR                  ;软件延时子程序
    PUSH CX
    PUSH AX
    MOV CX, 0F0H
D1:     MOV AX, 0F000H
D2:     DEC AX
        JNZ D2
        LOOP D1
        POP AX
        POP CX
        RET
DALLY ENDP
CODE ENDS
END START

```

4.11 键盘扫描及显示设计实验

4.11.1 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。

4.11.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.11.3 实验内容及原理

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 4×4 键盘，每个数码管显示值可为 $0 \sim F$ 共 16 个数。实验具体内容如下：将键盘进行编号，记作 $0 \sim F$ ，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 6 次按下的按键编号。

键盘及数码管显示单元电路图如图 4-11-1 和 4-11-2 所示。8255 键盘及显示实验参考接线图如图 4-11-3 所示。

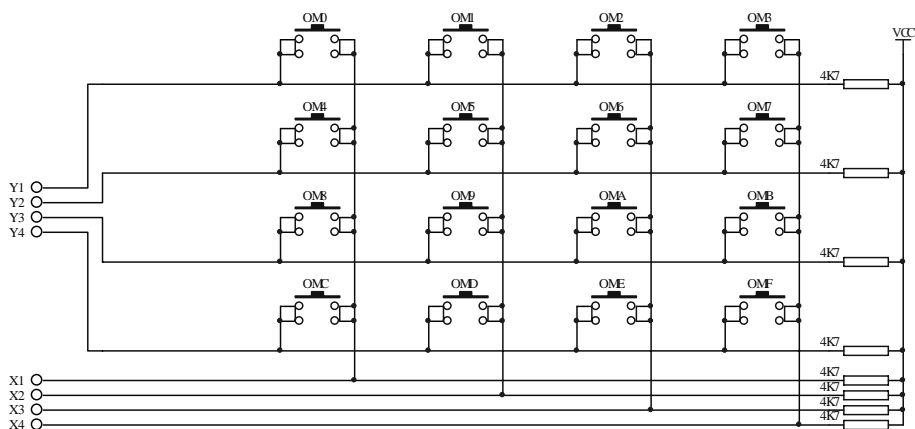


图 4-11-1 键盘及数码管显示单元 4×4 键盘矩阵电路图

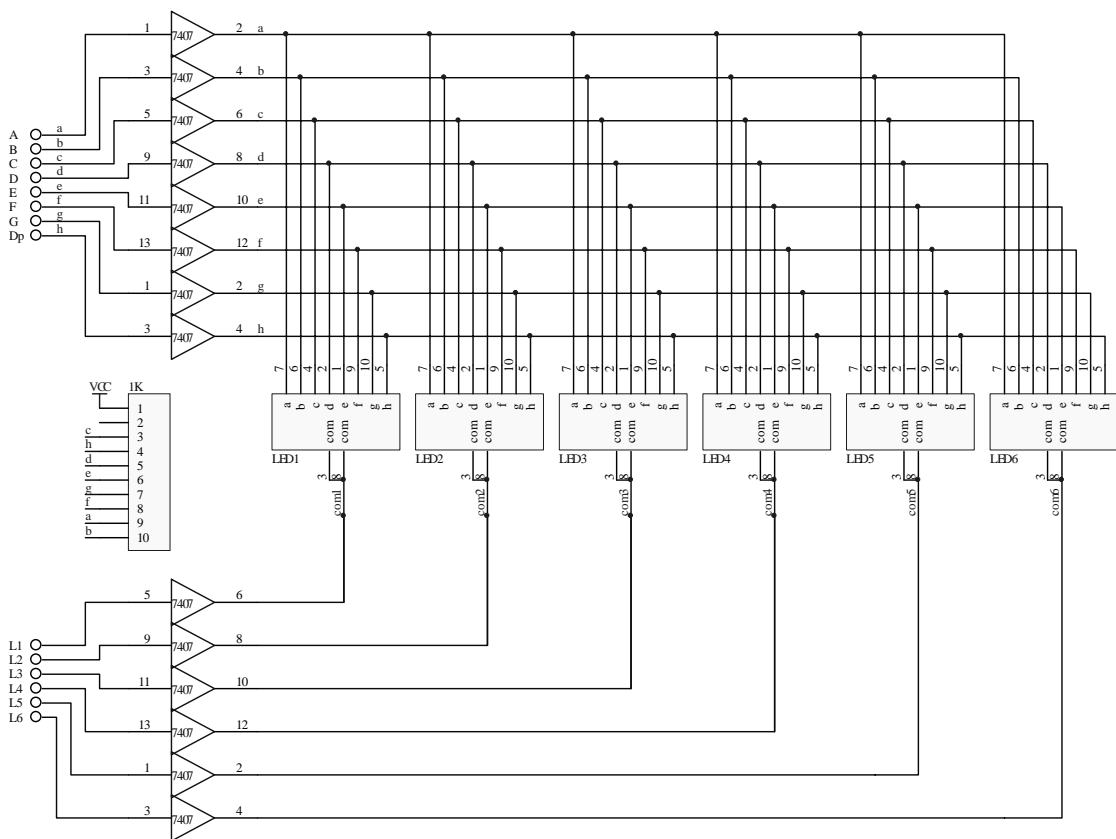


图 4-11-2 键盘及数码管显示单元 6 组数码管电路图

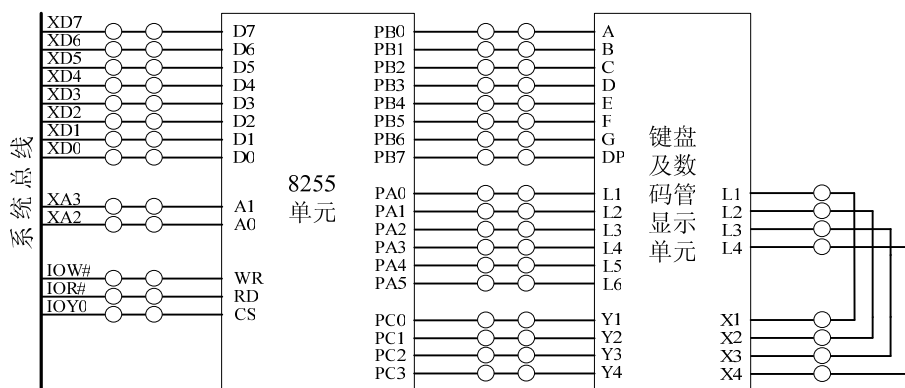


图 4-11-3 8255 键盘扫描及数码管显示实验线路图

4.11.4 实验步骤

(1) 实验接线图如图 4-11-3 所示，按图连接实验线路图。

(2) 运行 Tdpt 集成操作软件, 根据实验内容, 编写实验程序, 编译、链接。

(3) 运行程序, 按下按键, 观察数码管的显示, 验证程序功能。

实验程序清单 (Keyscan. ASM)

```
IOY0      EQU    3000H      ;片选 IOY0 对应的端口始地址
MY8255_A  EQU    IOY0+00H*4 ;8255 的 A 口地址
MY8255_B  EQU    IOY0+01H*4 ;8255 的 B 口地址
MY8255_C  EQU    IOY0+02H*4 ;8255 的 C 口地址
MY8255_MODE EQU    IOY0+03H*4 ;8255 的控制寄存器地址

STACK1 SEGMENT STACK
        DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT
DTABLE DB 3FH, 06H, 5BH, 4FH, 66H, 6DH, 7DH, 07H, 7FH, 6FH, 77H, 7CH, 39H, 5EH, 79H, 71H
DATA ENDS      ;键值表, 0~F 对应的 7 段数码管的段位值
CODE SEGMENT
        ASSUME CS:CODE, DS:DATA
START: MOV  AX, DATA
        MOV  DS, AX
        MOV  SI, 3000H      ;建立缓冲区, 存放要显示的键值
        MOV  AL, 00H        ;先初始化键值为 0
        MOV  [SI], AL
        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_MODE ;初始化 8255 工作方式
        MOV  AL, 81H        ;方式 0, A 口、B 口输出, C 口低 4 位输入
        OUT  DX, AL
BEGIN: CALL DIS              ;显示刷新
        CALL CLEAR          ;清屏
        CALL CCSCAN         ;扫描按键
        JNZ  GETKEY1        ;有键按下则跳置 GETKEY1
        MOV  AH, 1          ;判断 PC 键盘是否有按键按下
        INT  16H
        JZ   BEGIN         ;无按键则跳回继续循环, 有则退出
QUIT:  MOV  AX, 4C00H        ;返回到 DOS
        INT  21H
GETKEY1: CALL DIS           ;显示刷新
        CALL DALLY
        CALL DALLY
        CALL CLEAR          ;清屏
        CALL CCSCAN         ;再次扫描按键
        JNZ  GETKEY2        ;有键按下则跳置 GETKEY2
        JMP  BEGIN         ;否则跳回开始继续循环
GETKEY2: MOV  CH, 0FEH
        MOV  CL, 00H        ;设置当前检测的是第几列
COLUMN: MOV  AL, CH        ;选取一列, 将 X1~X4 中一个置 0
```

```

        MOV DX, MY8255_A
        OUT DX, AL
        MOV DX, MY8255_C           ;读 Y1~Y4, 用于判断是哪一行按键闭合
        IN AL, DX
L1:     TEST AL, 01H                ;是否为第 1 行
        JNZ L2                     ;不是则继续判断
        MOV AL, 00H                ;设置第 1 行第 1 列的对应的键值
        JMP KCODE
L2:     TEST AL, 02H                ;是否为第 2 行
        JNZ L3                     ;不是则继续判断
        MOV AL, 04H                ;设置第 2 行第 1 列的对应的键值
        JMP KCODE
L3:     TEST AL, 04H                ;是否为第 3 行
        JNZ L4                     ;不是则继续判断
        MOV AL, 08H                ;设置第 3 行第 1 列的对应的键值
        JMP KCODE
L4:     TEST AL, 08H                ;是否为第 4 行
        JNZ NEXT                   ;不是则继续判断
        MOV AL, 0CH                ;设置第 4 行第 1 列的对应的键值
KCODE:  ADD AL, CL                 ;将第 1 列的值加上当前列数, 确定按键值
        CALL PUTBUF                ;保存按键值
        PUSH AX
KON:    CALL DIS                   ;显示刷新
        CALL CLEAR                 ;清屏
        CALL CCSCAN                ;扫描按键, 判断按键是否弹起
        JNZ KON                    ;未弹起则继续循环等待弹起
        POP AX
NEXT:   INC CL                     ;当前检测的列数递增
        MOV AL, CH
        TEST AL, 08H               ;检测是否扫描到第 4 列
        JZ KERR                    ;是则跳回到开始处
        ROL AL, 1                  ;没检测到第 4 列则准备检测下一列
        MOV CH, AL
        JMP COLUM
KERR:   JMP BEGIN
CCSCAN PROC NEAR                  ;扫描是否有按键闭合子程序
        MOV AL, 00H
        MOV DX, MY8255_A           ;将 4 列全选通, X1~X4 置 0
        OUT DX, AL
        MOV DX, MY8255_C
        IN AL, DX                  ;读 Y1~Y4
        NOT AL
        AND AL, 0FH                ;取出 Y1~Y4 的反值
        RET
CCSCAN ENDP
CLEAR PROC NEAR                  ;清除数码管显示子程序
        MOV DX, MY8255_B           ;段位置 0 即可清除数码管显示
        MOV AL, 00H
        OUT DX, AL
        RET
CLEAR ENDP
DIS PROC NEAR                    ;显示键值子程序
        PUSH AX
        MOV SI, 3000H
        MOV DL, 0DFH

```

```

        MOV AL, DL
AGAIN:  PUSH DX
        MOV DX, MY8255_A
        OUT DX, AL           ;设置 X1~X4, 选通一个数码管
        MOV AL, [SI]         ;取出缓冲区中存放键值
        MOV BX, OFFSET DTABLE
        AND AX, 00FFH
        ADD BX, AX
        MOV AL, [BX]         ;将键值作为偏移和键值基地址相加得到相应的键值
        MOV DX, MY8255_B
        OUT DX, AL           ;写入数码管 A~Dp
        CALL DALLY
        INC SI               ;取下一个键值
        POP DX
        MOV AL, DL
        TEST AL, 01H         ;判断是否显示完?
        JZ OUT1              ;显示完, 返回
        ROR AL, 1
        MOV DL, AL
        JMP AGAIN            ;未显示完, 跳回继续
OUT1:   POP AX
        RET
DIS ENDP
PUTBUF PROC NEAR            ;保存键值子程序
        MOV SI, DI
        MOV [SI], AL
        DEC DI
        CMP DI, 2FFFH
        JNZ GOBACK
        MOV DI, 3005H
GOBACK: RET
PUTBUF ENDP
DALLY PROC NEAR             ;软件延时子程序
        PUSH CX
        MOV CX, 00FFH
D1:     MOV AX, 00FFH
D2:     DEC AX
        JNZ D2
        LOOP D1
        POP CX
        RET
DALLY ENDP
CODE    ENDS
        END START

```

4.12 电子发声设计实验

4.12.1 实验目的

学习用 8254 定时/计数器使蜂鸣器发声的编程方法。

4.12.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.12.3 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。

4.12.4 实验原理及步骤

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 4-12-1 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,200H

D1: MOV AX,0FFFFH

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,200H

D1: MOV AX,0FFFFH

D2: DEC AX

JNZ D2

LOOP D1

DEC DL

JNZ D0

RET

DALLY ENDP

表 4-12-1 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)。

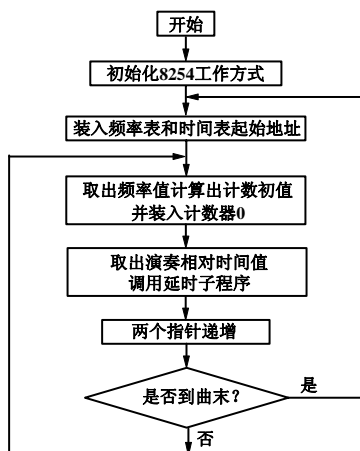


图 4-12-1 实验参考流程图

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 4-12-1 所示。

电子发声电路图如图 4-12-2 所示。

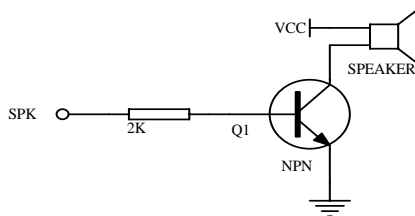


图 4-12-2 电子发声单元电路图

实验步骤如下：

- (1) 实验接线图如图 4-12-3 所示，按图接线。
- (2) 运行 Tdpt 集成操作软件，根据实验要求编写实验程序，编译、链接。
- (3) 运行程序，听扬声器发出的音乐是否正确。

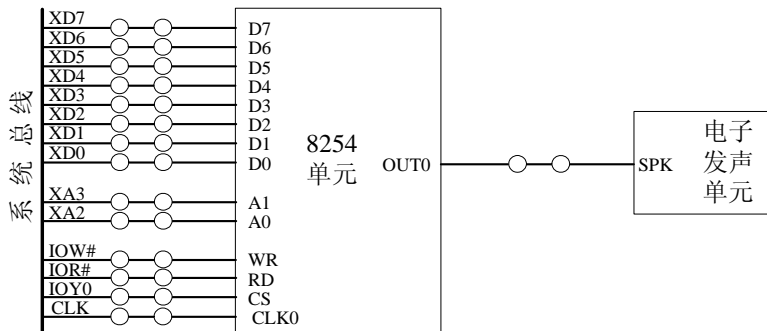


图 4-12-3 8254 电子发声实验接线图

实验参考例程 (SOUND. ASM)

```

IOY0      EQU    3000H      ;片选 IOY0 对应的端口始地址
MY8254_COUNT0 EQU    IOY0+00H*4 ;8254 计数器 0 端口地址
MY8254_COUNT1 EQU    IOY0+01H*4 ;8254 计数器 1 端口地址
MY8254_COUNT2 EQU    IOY0+02H*4 ;8254 计数器 2 端口地址
MY8254_MODE EQU    IOY0+03H*4 ;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 1.0416667MHz, 1.0416667M = 0FE502H
        MOV AX, 0E502H
        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
        MOV AH, 1                 ;判断是否有按键按下?
        INT 16H
        JZ PLAY
QUIT:  MOV DX, MY8254_MODE        ;退出时设置 8254 为方式 2, OUT0 置 0
        MOV AL, 10H

```



```
        OUT DX, AL
        MOV AX, 4C00H          ;结束程序退出
        INT 21H
DALLY PROC          ;延时子程序
D0:      MOV CX, 0F00H
D1:      MOV AX, 0FFFFH
D2:      DEC AX
        JNZ D2
        LOOP D1
        DEC DL
        JNZ D0
        RET
DALLY ENDP
CODE    ENDS
        END START
```

4.13 点阵 LED 显示设计实验

4.13.1 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。

4.13.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.13.3 实验内容及原理

使用 32 位 I/O 接口单元的 32 位输出 O0~O31 控制点阵 LED 单元 R0~R15 和 L0~L15。编写程序，在 16×16 点阵上循环显示汉字。

8×8 点阵 LED 相当于 8×8 个发光管组成的阵列，对于共阳极 LED 来说，其中每一行共用一个阳极（行控制），每一列共用一个阴极（列控制）。行控制和列控制满足正确的电平就可使相应行列的发光管点亮。实验平台上点阵 LED 的管脚及相应的行、列控制位如图 4-13-1 所示。

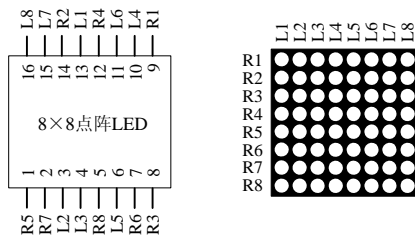


图 4-13-1 点阵 LED 管脚图

共阳极和共阴极 LED 的内部结构分别如图 4-13-2 和 4-13-3 所示。

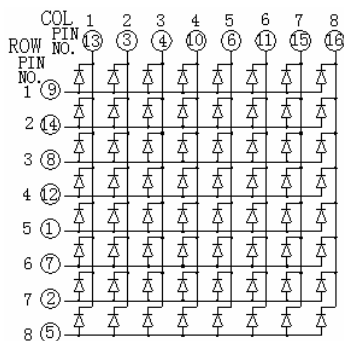


图 4-13-2 共阳极 LED 内部结构图

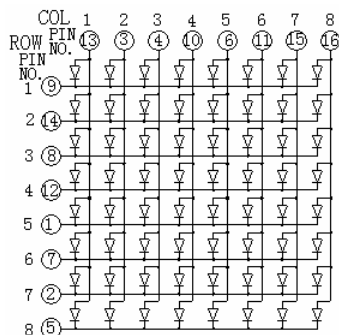
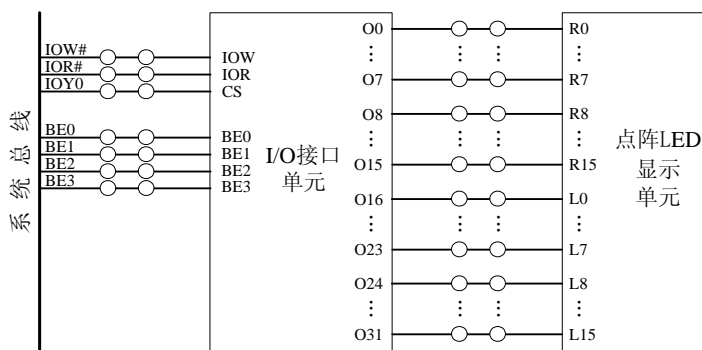


图 4-13-3 共阴极 LED 内部结构图

在 TD-PITD+实验系统上的 LED 点阵单元采用了 4 片 $\Phi 1.9$ 的共阴极 LED 点阵组成 16×16 的点阵。利用取字模软件得到汉字字符数组，设计程序，在点阵上滚动显示“西安唐都科教仪器公司”。实验参考接线如图 4-13-4 所示。

图 4-13-4 16×16 点阵汉字显示实验参考接线图

4.13.4 实验步骤

- (1) 实验接线图如图 4-13-4 所示，按图接线。
- (2) 运行 Tdptit 集成操作软件，根据实验要求编写实验程序，编译、链接。
- (3) 运行程序，观察点阵的显示，验证程序功能。

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 4.13.5 节所述。

实验程序清单（LED-HZ.asm）

```
;Led-HZ.asm, 32 位 LED 点阵汉字显示实验
INCLUDE LED-HZ.inc
.386P
IOY0      EQU    3000H      ;片选 IOY0 对应的端口始地址
```

```

STACK1 SEGMENT STACK
        DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT USE16
ADDR DW ?
DATA ENDS
CODE SEGMENT USE16
        ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
        MOV DS, AX
A2:    MOV ADDR, OFFSET HZDOT      ;取汉字数组始地址
        MOV SI, ADDR
A1:    MOV CX, 20H                  ;控制 1 屏显示时间
LOOP2: CALL DISPHZ
        SUB SI, 32
        LOOP LOOP2
KEY:   MOV AH, 1                    ;判断是否有按键按下?
        INT 16H
        JNZ QUIT
        ADD SI, 2
        MOV AX, SI
        SUB AX, ADDR
        CMP AX, 352                 ;比较文字是否显示完毕
        JNB A2
        JMP A1
QUIT:  MOV EAX, 0                    ;灭灯
        MOV DX, IOY0
        OUT DX, EAX
        MOV AX, 4C00H                ;结束程序退出
        INT 21H
DISPHZ PROC NEAR                    ;显示 1 屏汉字子程序
        PUSH CX
        MOV CX, 16
        MOV BX, 0FFFFH
LOOP1: MOV AL, BYTE PTR[SI]
        MOV AH, BYTE PTR[SI+1]
        ROL EAX, 16
        MOV AX, BX
        ADD SI, 2
        ROL BX, 1
        NOT EAX
        MOV DX, IOY0
        OUT DX, EAX
        CALL DALLY
        LOOP LOOP1
        POP CX
        RET
DISPHZ ENDP
DALLY PROC NEAR                    ;软件延时子程序
        PUSH CX
        PUSH AX


```

```

MOV CX, 09H
D1: MOV AX, 0F000H
D2: DEC AX
JNZ D2
LOOP D1
POP AX
POP CX
RET
DALLY ENDP CODE ENDS
END START

```

4.13.5 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；

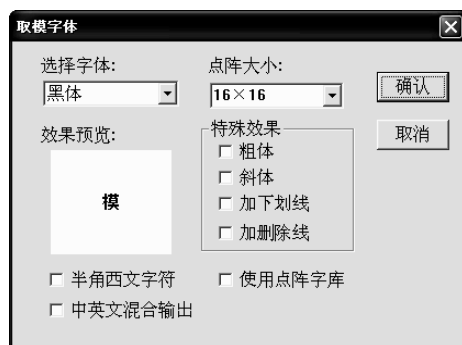


图 4-13-5 取模字体对话框

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；

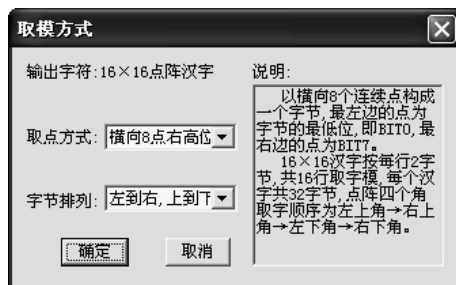


图 4-13-6 取模方式对话框

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 4-13-7 所示；

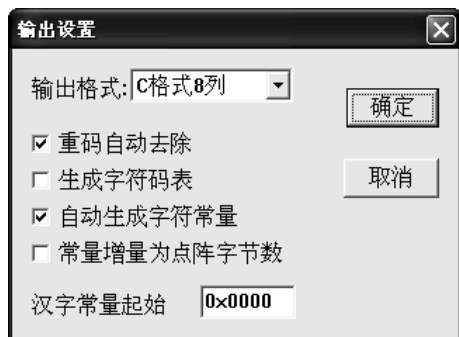


图 4-13-7 输出设置对话框

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 4-13-8 所示，然后点击输入按钮；

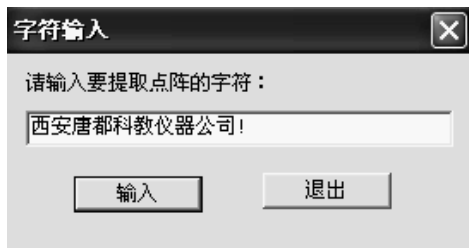


图 4-13-8 字符输入对话框

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 4-13-9 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 4-13-10 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。

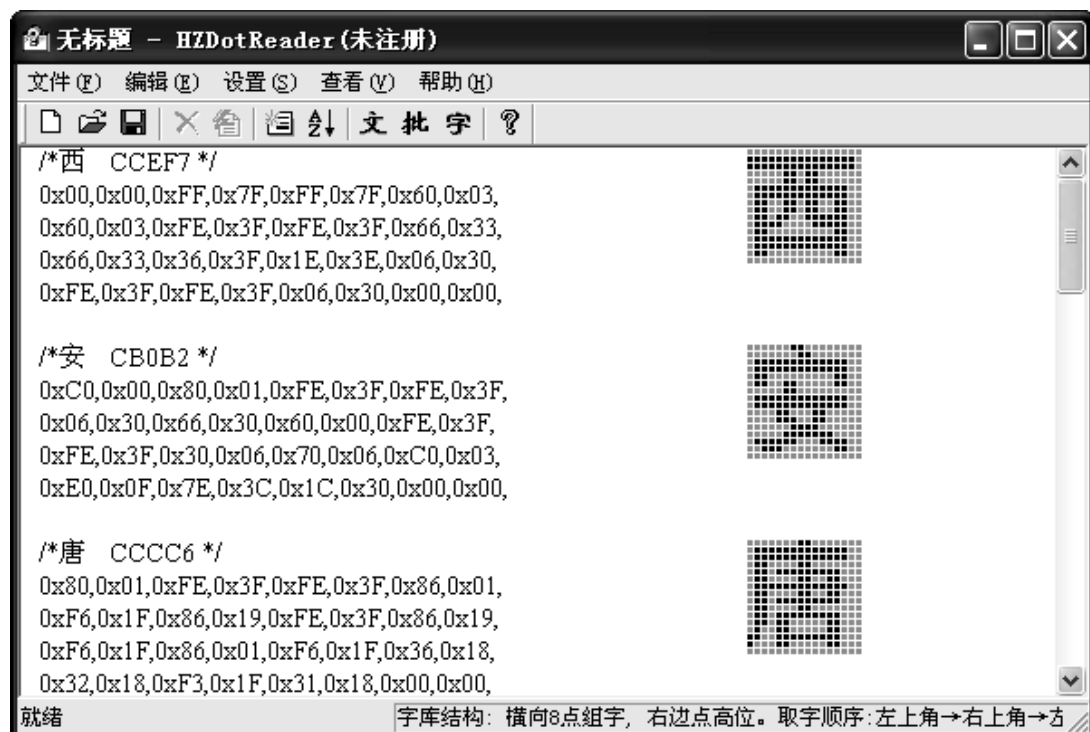


图 4-13-9 字模生成窗口

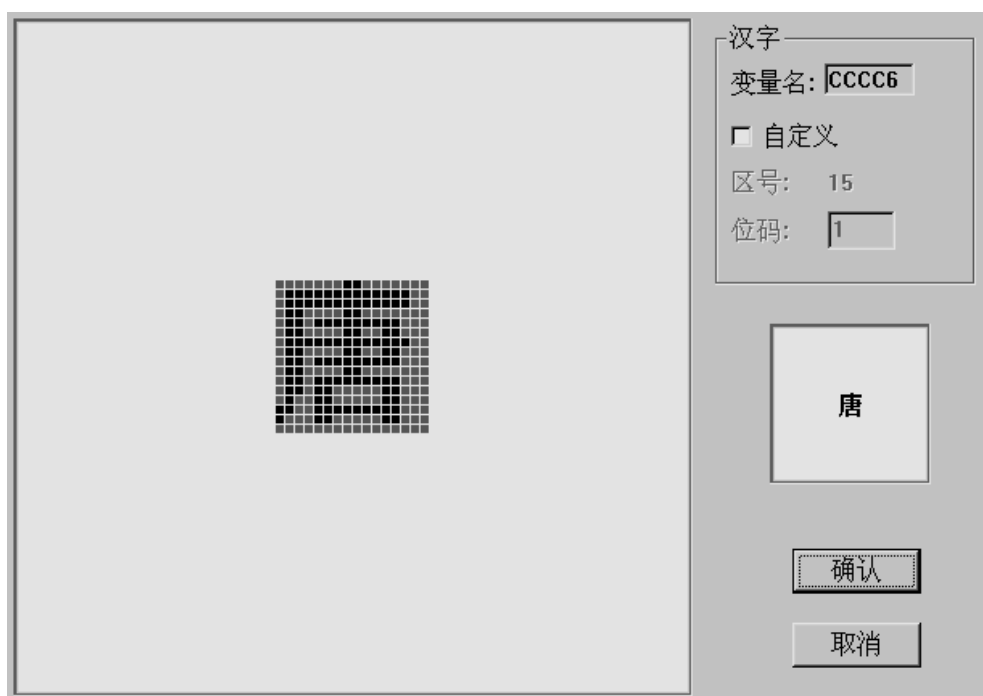


图 4-13-10 点阵编辑窗

4.14 图形 LCD 显示设计实验

4.14.1 实验目的

了解图形 LCD 的控制方法。

4.14.2 实验设备

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

4.14.3 实验内容

本实验使用的是 128×64 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。

4.14.4 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 4-14-1 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 4-14-1 所列，读写时序图如图 4-14-1 和 4-14-2 所示。

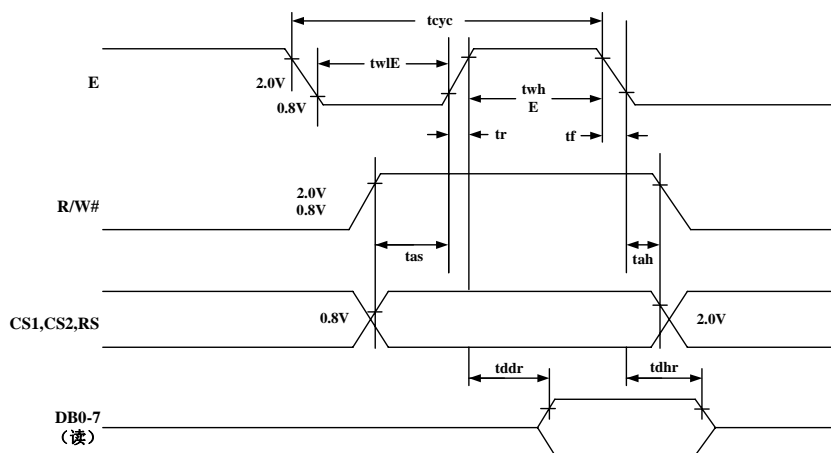


图 4-14-1 读操作时序图

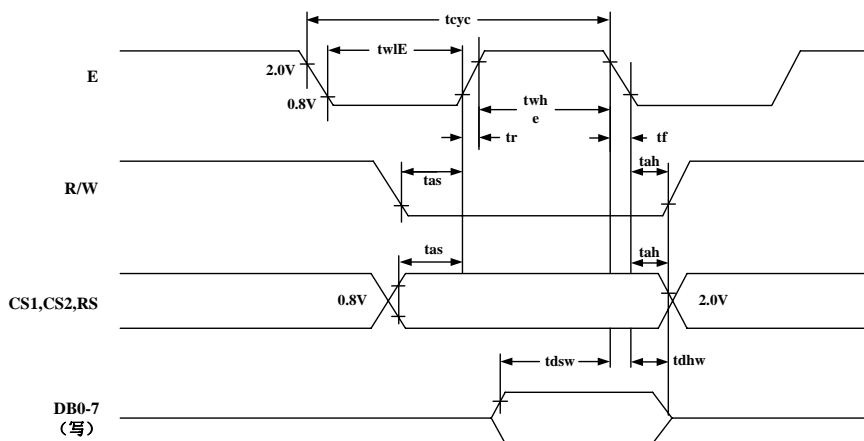


图 4-14-2 写操作时序图

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 4-14-2 所列。

表 4-14-2 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 4-14-3 所示，调节 10K 微调可以改变液晶显示的对比度。

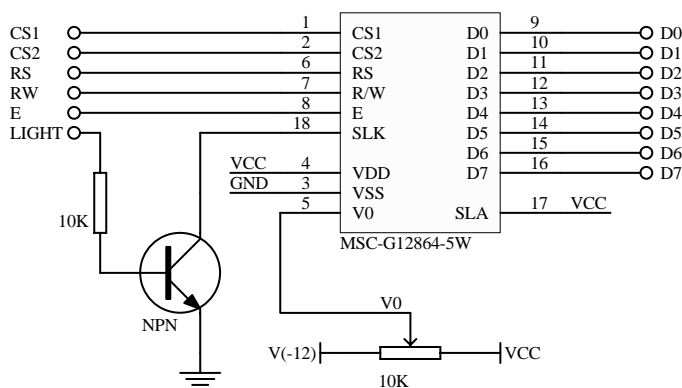


图 4-14-3 液晶显示电路图

4. 14.5 实验步骤

- (1) 实验接线图如图 4-14-4 所示，按图接线。
- (2) 运行 Tdptit 集成操作软件，根据实验要求编写实验程序，编译、链接。
- (3) 运行程序，观察 LCD 屏幕上显示是否正确。

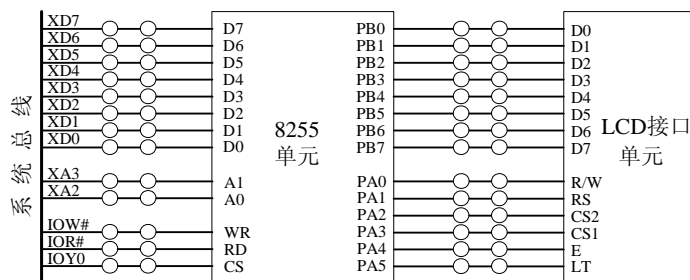


图 4-14-4 液晶实验参考接线图

实验程序清单 (Lcd. asm)

```

INCLUDE LCD. INC
IOY0      EQU    3000H          ;片选 IOY0 对应的端口始地址
MY8255_A   EQU    IOY0+00H*4    ;8255 的 A 口地址
MY8255_B   EQU    IOY0+01H*4    ;8255 的 B 口地址
MY8255_C   EQU    IOY0+02H*4    ;8255 的 C 口地址
MY8255_MODE EQU    IOY0+03H*4    ;8255 的控制寄存器地址
STACK1 SEGMENT STACK
        DW 256 DUP(?)
STACK1 ENDS
DATA SEGMENT
CMD      DB  ?                ;定义操作 LCD 命令变量
DAT      DB  ?                ;定义操作 LCD 数据变量
XAD      DB  ?                ;定义 X 地址变量
YAD      DB  ?                ;定义 Y 地址变量
DATA ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS:DATA
START:  MOV AX, DATA
        MOV DS, AX
        MOV DX, MY8255_MODE    ;定义 8255 工作方式
        MOV AL, 80H            ;工作方式 0, A 口和 B 口为输出
        OUT DX, AL
        MOV CMD, 04H           ;设置第一块显示打开
        MOV DAT, 3FH
        CALL WRITE
        MOV CMD, 08H           ;设置第二块显示打开
        MOV DAT, 3FH
        CALL WRITE
        MOV CMD, 04H           ;设置第一块起始行
        MOV DAT, 0COH
        CALL WRITE
        MOV CMD, 08H           ;设置第二块起始行
        MOV DAT, 0COH
        CALL WRITE
        MOV AL, 0
CLRALL: MOV CMD, 04H           ;清屏
        CALL CLEAR
        MOV CMD, 08H
        CALL CLEAR

```

```

INC AL
CMP AL, 8
JNZ CLRALL
MOV XAD, 0BAH          ;在第一块、以 X 地址 BAH
MOV YAD, 40H           ;Y 地址 40H 为起始
MOV CMD, 04H
MOV SI, OFFSET TANG    ;显示汉字“唐”
CALL WRITEHZ
MOV XAD, 0BAH          ;在第一块、以 X 地址 BAH
MOV YAD, 50H           ;Y 地址 50H 为起始
MOV CMD, 04H
MOV SI, OFFSET DU      ;显示汉字“都”
CALL WRITEHZ
MOV XAD, 0BAH          ;在第一块、以 X 地址 BAH
MOV YAD, 60H           ;Y 地址 60H 为起始
MOV CMD, 04H
MOV SI, OFFSET KE      ;显示汉字“科”
CALL WRITEHZ
MOV XAD, 0BAH          ;在第一块、以 X 地址 BAH
MOV YAD, 70H           ;Y 地址 70H 为起始
MOV CMD, 04H
MOV SI, OFFSET JIAO    ;显示汉字“教”
CALL WRITEHZ
MOV XAD, 0BAH          ;在第二块、以 X 地址 BAH
MOV YAD, 40H           ;Y 地址 40H 为起始
MOV CMD, 08H
MOV SI, OFFSET YI      ;显示汉字“仪”
CALL WRITEHZ
MOV XAD, 0BAH          ;在第二块、以 X 地址 BAH
MOV YAD, 50H           ;Y 地址 50H 为起始
MOV CMD, 08H
MOV SI, OFFSET QI      ;显示汉字“器”
CALL WRITEHZ
MOV XAD, 0BAH          ;在第二块、以 X 地址 BAH
MOV YAD, 60H           ;Y 地址 60H 为起始
MOV CMD, 08H
MOV SI, OFFSET GONG    ;显示汉字“公”
CALL WRITEHZ
MOV XAD, 0BAH          ;在第二块、以 X 地址 BAH
MOV YAD, 70H           ;Y 地址 70H 为起始
MOV CMD, 08H
MOV SI, OFFSET SI1     ;显示汉字“司”
CALL WRITEHZ
MOV XAD, 0BCH          ;在第一块、以 X 地址 BCH
MOV YAD, 60H           ;Y 地址 60H 为起始
MOV CMD, 04H
MOV SI, OFFSET HUAN    ;显示汉字“欢”
CALL WRITEHZ
MOV XAD, 0BCH          ;在第一块、以 X 地址 BCH
MOV YAD, 70H           ;Y 地址 70H 为起始
MOV CMD, 04H

```



```
MOV SI, OFFSET YING          ;显示汉字“迎”
CALL WRITEHZ
MOV XAD, 0BCH                ;在第二块、以 X 地址 BCH
MOV YAD, 40H                 ;Y 地址 40H 为起始
MOV CMD, 08H
MOV SI, OFFSET NIN          ;显示汉字“您”
CALL WRITEHZ
MOV XAD, 0BCH                ;在第二块、以 X 地址 BCH
MOV YAD, 50H                 ;Y 地址 50H 为起始
MOV CMD, 08H
MOV SI, OFFSET GANTAN       ;显示标点“!”
CALL WRITEHZ
MOVE1: MOV CX, 0C0H          ;设置起始行从 C0H 到 FFH
MOVE2: MOV DAT, CL           ;达到显示向上移动的效果
MOV CMD, 04H
CALL WRITE
MOV CMD, 08H
CALL WRITE
CALL DALLY
MOV AH, 1                    ;判断是否有按键按下
INT 16H
JNZ QUIT                     ;无按键则继续循环，有则退出
INC CX
CMP CX, 100H
JNZ MOVE2
JMP MOVE1
QUIT: CALL LEDOFF
MOV AX, 4C00H                ;结束程序退出
INT 21H
WRITE PROC NEAR              ;写命令和数据子程序
MOV DX, MY8255_B             ;送出命令或数据
MOV AL, DAT
OUT DX, AL
OR CMD, 30H                  ;使 E 信号产生高脉冲，将命令或数据写入
MOV AL, CMD
MOV DX, MY8255_A
OUT DX, AL
AND CMD, 0EFH
MOV AL, CMD
OUT DX, AL
RET
WRITE ENDP
CLEAR PROC NEAR              ;清 X 地址为 B8H~BFH 中的一页屏幕子程序
PUSH AX
ADD AL, 0B8H
MOV DAT, AL                  ;设置 X 地址
CALL WRITE
CALL QUERY
MOV DAT, 40H                 ;设置 Y 地址
CALL WRITE
CALL QUERY
```



```

        MOV CX, 64                ;循环 64 次，清除整页
LC:     MOV DAT, 00H              ;向数据单元中写 00H，达到清屏
        ADD CMD, 2
        CALL WRITE
        SUB CMD, 2
        CALL QUERY
        LOOP LC
        POP AX
        RET
CLEAR ENDP
QUERY PROC NEAR                  ;查询 LCD 控制器是否空闲
        ADD CMD, 1
        MOV DX, MY8255_MODE       ;设置 8255 的 B 口为输入，需要读数据
        MOV AL, 82H
        OUT DX, AL
Q1:     OR  CMD, 10H               ;将命令送入
        MOV AL, CMD
        MOV DX, MY8255_A
        OUT DX, AL
        AND CMD, 0EFH
        MOV AL, CMD
        OUT DX, AL
        MOV DX, MY8255_B          ;读出查询字，进行判断
        IN  AL, DX
        TEST AL, 80H
        JZ  Q2                    ;空闲则退出，否则继续查询
        JMP Q1
Q2:     MOV DX, MY8255_MODE       ;恢复 8255 控制字，A、B 口均为输出
        MOV AL, 80H
        OUT DX, AL
        SUB CMD, 1
        RET
QUERY ENDP
WRITEHZ PROC NEAR                ;从某一坐标为起始写汉字子程序
        MOV BL, 0                 ;将 16*16 分成两个 16*8 完成写入
WRHZ1:  MOV AL, XAD                ;设置 X 坐标
        MOV DAT, AL
        CALL WRITE
        CALL QUERY
        MOV AL, YAD               ;设置 Y 坐标
        MOV DAT, AL
        CALL WRITE
        CALL QUERY
        MOV CX, 0
WRHZ2:  MOV DI, SI                 ;装入汉字点阵数据起始地址
        MOV AL, BL                ;计算偏移[CX+(BL*16)]
        MOV DL, 16
        MUL DL
        ADD AX, CX
        ADD DI, AX                ;将结果与起始地址相加
        MOV AL, BYTE PTR[DI]      ;取出数据并写入 LCD

```

```

MOV DAT, AL
ADD CMD, 2
CALL WRITE
SUB CMD, 2
CALL QUERY
INC CX
CMP CX, 16
JNZ WRHZ2           ;未写完则跳回继续
ADD XAD, 1          ;X 地址加 1, 准备写下页
INC BL
CMP BL, 2
JNZ WRHZ1
RET
WRITEHZ ENDP
LEDON PROC NEAR     ;打开背景灯
    OR  CMD, 20H
    MOV DX, MY8255_A
    MOV AL, CMD
    OUT DX, AL
    RET
LEDON ENDP
LEDOFF PROC NEAR    ;关闭背景灯
    AND CMD, 0DFH
    MOV DX, MY8255_A
    MOV AL, CMD
    OUT DX, AL
    RET
LEDOFF ENDP
DALLY PROC NEAR     ;软件延时子程序
    PUSH CX
    PUSH AX
    MOV CX, 03FFH
D1:  MOV AX, 0FFFFH
D2:  DEC AX
    JNZ D2
    LOOP D1
    POP AX
    POP CX
    RET
DALLY ENDP
CODE ENDS
END START

```

4.15 步进电机实验

4.15.1 实验目的

1. 学习步进电机的控制方法。
2. 学会用 8255 控制步进电机。

4.15.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.15.3 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。

4.15.4 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 4-15-1 所示，励磁顺序如表 4-15-1 所列。

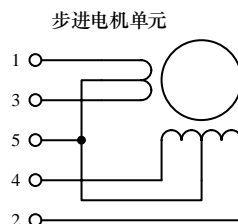


图 4-15-1 励磁线圈

表 4-15-1 励磁顺序

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

表 4-15-2 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

图 4-15-2 驱动电路原理图

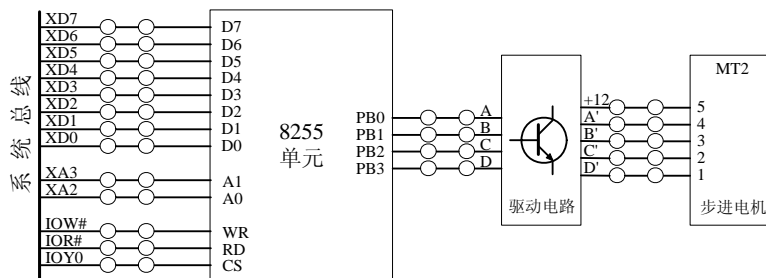


图 4-15-3 步进电机实验参考接线图

4.15.5 实验步骤

- (1) 实验接线图如图 4-15-3 所示，按图接线。
- (2) 运行 Tdptit 集成操作软件，根据实验要求编写实验程序，编译、链接。
- (3) 运行程序，观察电机运转情况。

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 (BUJIN.ASM)

```
IOY0      EQU    3000H          ;片选 IOY0 对应的端口始地址
MY8255_A  EQU    IOY0+00H*4     ;8255 的 A 口地址
MY8255_B  EQU    IOY0+01H*4     ;8255 的 B 口地址
MY8255_C  EQU    IOY0+02H*4     ;8255 的 C 口地址
MY8255_MODE EQU    IOY0+03H*4   ;8255 的控制寄存器地址

STACK1 SEGMENT STACK
    DW 256 DUP(?)
STACK1 ENDS

DATA SEGMENT
TTABLE 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 DX, MY8255_MODE      ;定义 8255 工作方式
        MOV AL, 80H             ;工作方式 0, A 口和 B 口为输出
        OUT DX, AL
A1:     MOV BX, OFFSET TTABLE
        MOV CX, 0008H
A2:     MOV AL, [BX]
        MOV DX, MY8255_B
        OUT DX, AL
        CALL DALLY
        INC BX
        LOOP A2
        MOV AH, 1               ;判断是否有按键按下
        INT 16H
        JZ A1                   ;无按键则跳回继续循环，有则退出
QUIT:   MOV AX, 4C00H           ;结束程序退出
        INT 21H
DALLY PROC NEAR                ;软件延时子程序
    PUSH CX
    PUSH AX
    MOV CX, 5000H
D1:     MOV AX, 5000H
D2:     DEC AX
        JNZ D2
        LOOP D1
```

```
        POP  AX
        POP  CX
    RET
DALLY ENDP
CODE  ENDS
END  START
```

4.16 直流电机闭环调速实验

4.16.1 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。

4.16.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.16.3 实验内容及原理

直流电机闭环调速实验原理如图 4-14-1 所示。

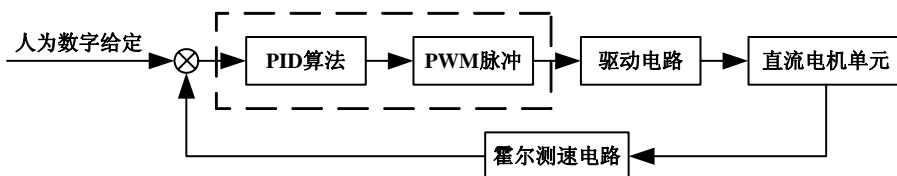


图 4-16-1 直流电机闭环调速实验原理图

如图 4-16-1 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 4-16-2 所示。

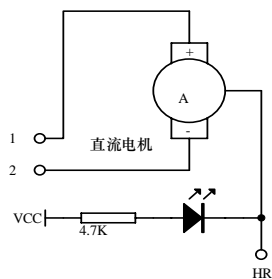



图 4-16-2 直流电机电路原理图

4. 16. 4 实验步骤

- (1) 实验接线图如图 4-16-4 所示，按图接线。
- (2) 运行 Tdpit 集成操作软件，根据实验要求和流程图 4-16-3 编写实验程序，实验参数取值范围见表 4-16-1，编译、链接。
- (3) 运行程序，观察电机运转情况以及显示在屏幕上的结果。
- (4) 用软件所带专用图形界面观测的方法：点击快捷工具栏上“”按钮，启动专用图形界面显示窗口，即可观察波形显示。

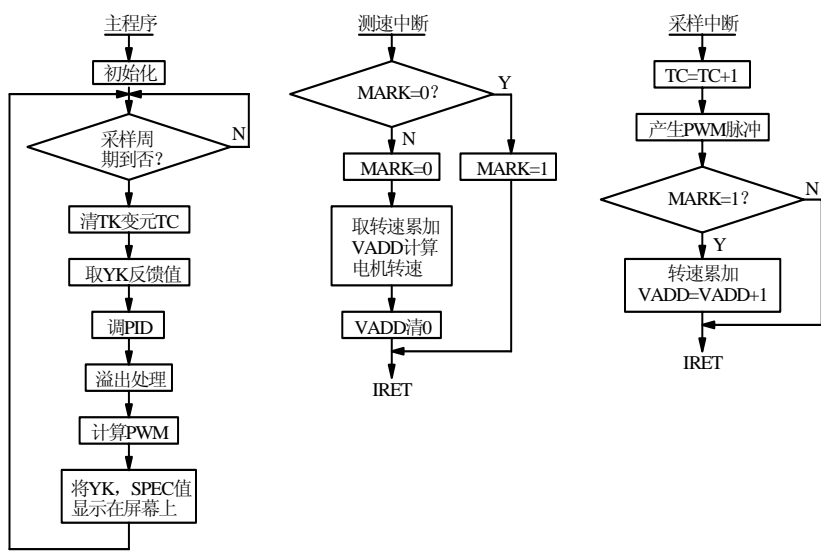


图 4-16-3 直流电机闭环调速实验流程图

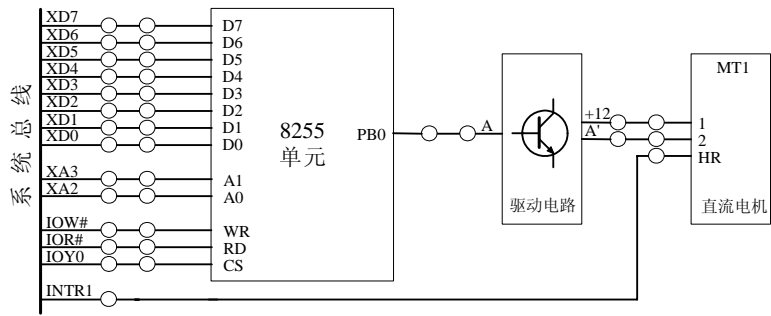


图 4-16-4 直流电机闭环调速实验参考接线图

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。

项目 \ 参数	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：直流电机闭环调速控制实验以及后面的温度控制实验中给定与反馈都为单极性，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）=（采样周期×实际刻度值）÷4

电机转速范围在 6 转/秒~48 转/秒之间，速度过低受阻力影响较大不稳定。最大转速不超过 4000 转/分。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 8254 的 OUT0 接中断 MIR6，间隔为 1ms，故采样周期=14H×1=0.02 秒。如实际刻度值=900，则实际响应时间（秒）=（0.02×900）÷4=4.5 秒。

表 4-16-1 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	

实验程序清单（Zhiliu.asm）

```

INTR_IVADD    EQU    003CH        ;INTR 对应的中断矢量地址
IOY0          EQU    3000H        ;片选 IOY0 对应的端口始地址
MY8255_A      EQU    IOY0+00H*4   ;8255 的 A 口地址

```

```

MY8255_B      EQU    IOY0+01H*4      ;8255 的 B 口地址
MY8255_C      EQU    IOY0+02H*4      ;8255 的 C 口地址
MY8255_MODE    EQU    IOY0+03H*4      ;8255 的控制寄存器地址
PC8254_COUNT0 EQU    40H              ;PC 机内 8254 定时器 0 端口地址
PC8254_MODE    EQU    43H              ;PC 机内 8254 控制寄存器端口地址

STACK1 SEGMENT STACK
        DW 64 DUP(?)
TOP     LABEL WORD
STACK1 ENDS

DATA SEGMENT
TABLE1  DB  'Assumed Fan Speed: (/s)', 0AH, 0DH, '$'      ;字符串变量
TABLE2  DB  'Current Fan Speed: (/s)', 0AH, 0DH, '$'      ;字符串变量
ENT      DB  0AH, 0DH, '$'          ;换行, 回车
CS_BAK  DW  ?                      ;保存 INTR 原中断处理程序入口段地址的变量
IP_BAK  DW  ?                      ;保存 INTR 原中断处理程序入口偏移地址的变量
IM_BAK  DB  ?                      ;保存 INTR 原中断屏蔽字的变量
CS_BAK1 DW  ?                      ;保存定时器 0 中断处理程序入口段地址的变量
IP_BAK1 DW  ?                      ;保存定时器 0 中断处理程序入口偏移地址的变量
IM_BAK1 DB  ?                      ;保存定时器 0 中断屏蔽字的变量
TS       DB  20                    ;采样周期
SPEC     DW  55                    ;转速给定值
IBAND    DW  0060H                 ;积分分离值
KPP       DW  1060H                 ;比例系数
KII       DW  0010H                 ;积分系数
KDD       DW  0020H                 ;微分系数
YK        DW  ?
CK        DB  ?
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
        MOV  AX, SPEC                ;将给定值输出给专用界面显示
        MOV  DI, 0D80H
        CALL SEND
        MOV  DX, OFFSET TABLE1      ;显示字符串 1
        MOV  AH, 09H
        INT  21H
        MOV  AX, SPEC                ;显示给定值
        CALL DECSHOW
        MOV  DX, OFFSET ENT          ;回车, 换行
        MOV  AH, 09H
        INT  21H
        MOV  DX, OFFSET TABLE2     ;显示字符串 2
        MOV  AH, 09H
        INT  21H
        CLI
        MOV  AX, 0000H
        MOV  ES, AX
        MOV  DI, 0020H
        MOV  AX, ES: [DI]
        MOV  IP_BAK1, AX             ;保存定时器 0 中断处理程序入口偏移地址
        MOV  AX, OFFSET TIMERISR
        MOV  ES: [DI], AX            ;设置实验定时中断处理程序入口偏移地址
        ADD  DI, 2
        MOV  AX, ES: [DI]
        MOV  CS_BAK1, AX             ;保存定时器 0 中断处理程序入口段地址
        MOV  AX, SEG TIMERISR
        MOV  ES: [DI], AX            ;设置实验定时中断处理程序入口段地址
        IN   AL, 21H
        MOV  IM_BAK1, AL             ;保存 INTR 原中断屏蔽字
        AND  AL, 0F7H
        OUT  21H, AL                 ;打开定时器 0 中断屏蔽位
        MOV  DI, INTR_IVADD
        MOV  AX, ES: [DI]
        MOV  IP_BAK, AX              ;保存 INTR 原中断处理程序入口偏移地址
        MOV  AX, OFFSET MYISR
        MOV  ES: [DI], AX            ;设置当前中断处理程序入口偏移地址
        ADD  DI, 2
        MOV  AX, ES: [DI]
        MOV  CS_BAK, AX              ;保存 INTR 原中断处理程序入口段地址
        MOV  AX, SEG MYISR
        MOV  ES: [DI], AX            ;设置当前中断处理程序入口段地址

        IN   AL, 21H
        MOV  IM_BAK, AL              ;保存 INTR 原中断屏蔽字
        AND  AL, 7FH
        OUT  21H, AL                 ;打开 INTR 的中断屏蔽位
        MOV  VADD, 0000H             ;变量的初始化

```

```

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  AL, 80H                ;初始化 8255
MOV  DX, MY8255_MODE
OUT  DX, AL
MOV  AL, 00H
MOV  DX, MY8255_B
OUT  DX, AL
MOV  DX, PC8254_MODE      ;初始化 PC 机定时器 0，定时 1ms
MOV  AL, 36H
OUT  DX, AL
MOV  DX, PC8254_COUNT0
MOV  AL, 8FH
OUT  DX, AL
MOV  AL, 04H
OUT  DX, AL
STI
M1:  MOV  AL, TS            ;判断采样周期到否?
      SUB  AL, TC
      JNC  M1              ;没到则继续等待
      MOV  TC, 00H        ;采样周期到，将采样周期变量清 0
      MOV  AL, ZVV
      MOV  AH, 00H
      MOV  YK, AX          ;得到反馈量 YK
      CALL PID             ;调用 PID 子程序，得到控制量 CK
      MOV  AL, CK          ;把控制量转化成 PWM 输出
      SUB  AL, 80H
      JC   ISO
      MOV  AAAA, AL

```

```

        JMP    COU
ISO:    MOV    AL, 10H                ;电机的启动值不能低于 10H
        MOV    AAAA, AL
COU:    MOV    AL, 7FH
        SUB    AL, AAAA
        MOV    BBB, AL
        MOV    AX, YK                ;将反馈值 YK 送到屏幕显示
        CALL   DECSHOW
        MOV    AX, YK                ;将反馈值 YK 输出给专用界面显示
        MOV    DI, 0D81H
        CALL   SEND
        MOV    AL, CK                ;将控制量 CK 输出给专用界面显示
        MOV    DI, 0D82H
        CALL   SEND
        MOV    DL, 0DH                ;回车
        MOV    AH, 02H
        INT    21H
        MOV    AH, 1                ;判断是否有按键按下
        INT    16H
        JZ     M1                    ;无按键则跳回继续等待，有则退出
EXIT:    CLI
        MOV    AL, 00H                ;退出时停止电机运转
        MOV    DX, MY8255_B
        OUT    DX, AL
        MOV    DX, PC8254_MODE        ;恢复 PC 机定时器 0 状态
        MOV    AL, 36H
        OUT    DX, AL
        MOV    DX, PC8254_COUNT0
        MOV    AL, 00H
        OUT    DX, AL
        MOV    AL, 00H
        OUT    DX, AL
        MOV    AX, 0000H                ;恢复 INTR 原中断矢量
        MOV    ES, AX
        MOV    DI, INTR_IVADD
        MOV    AX, IP_BAK                ;恢复 INTR 原中断处理程序入口偏移地址
        MOV    ES: [DI], AX
        ADD    DI, 2
        MOV    AX, CS_BAK                ;恢复 INTR 原中断处理程序入口段地址
        MOV    ES: [DI], AX
        MOV    AL, IM_BAK                ;恢复 INTR 原中断屏蔽寄存器的屏蔽字
        OUT    21H, AL
        MOV    DI, 0020H
        MOV    AX, IP_BAK1                ;恢复定时器 0 中断处理程序入口偏移地址
        MOV    ES: [DI], AX
        ADD    DI, 2
        MOV    AX, CS_BAK1                ;恢复定时器 0 中断处理程序入口段地址
        MOV    ES: [DI], AX
        MOV    AL, IM_BAK1
        OUT    21H, AL                ;恢复屏蔽字
        STI

```

```

        MOV  AX, 4C00H
        INT  21H
MYISR PROC NEAR                                ;系统总线 INTR 中断处理程序
        PUSH AX
        PUSH CX
        PUSH DX
        MOV  AX, DATA
        MOV  DS, AX
        MOV  AL, MARK
        CMP  AL, 01H
        JZ   IN1
        MOV  MARK, 01H
        JMP  IN2
IN1:    MOV  MARK, 00H                        ;计算转速
VV:     MOV  DX, 0000H
        MOV  AX, 03E8H
        MOV  CX, VADD
        CMP  CX, 0000H
        JZ   MM1
        DIV  CX
MM:     MOV  ZV, AL
        MOV  VADD, 0000H
MM1:    MOV  AL, ZV
        MOV  ZVV, AL
IN2:    MOV  AL, 20H                        ;向 PC 机内部 8259 发送中断结束命令
        OUT  20H, AL
        POP  DX
        POP  CX
        POP  AX
        IRET
MYISR ENDP
TIMERISR PROC NEAR                            ;PC 机定时器 0 中断处理程序
        PUSH AX
        PUSH CX
        PUSH DX
        MOV  AX, DATA
        MOV  DS, AX
        INC  TC                            ;采样周期变量加 1
        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:    MOV  AL, 20H                        ;中断结束, 发 EOI 命令
        OUT  20H, AL
        POP  DX
        POP  CX

```

```

        POP  AX
        IRET
TIMERISR ENDP
KJ PROC NEAR                                ;PWM 子程序
        PUSH AX
        CMP  FPWM, 01H                      ;PWM 为 1，产生 PWM 的高电平
        JNZ  TEST2
        CMP  VAA, 00H
        JNZ  ANOTO
        MOV  FPWM, 02H
        MOV  AL, BBB
        CLC
        RCR  AL, 01H
        MOV  VBB, AL
        JMP  TEST2
ANOTO:  DEC  VAA
        MOV  AL, 01H                        ;PB0=1 电机转动
        MOV  DX, MY8255_B
        OUT  DX, AL
TEST2:  CMP  FPWM, 02H                      ;PWM 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  AL, 00H                        ;PB0=0 电机停止
        MOV  DX, MY8255_B
        OUT  DX, AL
OUTT:   POP  AX
        RET
KJ ENDP
PID:    MOV  AX, SPEC                      ;PID 子程序
        SUB  AX, YK                        ;求偏差 EK
        MOV  R0, AX
        MOV  R1, AX
        SUB  AX, EK_1
        MOV  R2, AX
        SUB  AX, AEK_1                      ;求 BEK
        MOV  BEK, AX
        MOV  R8, AX
        MOV  AX, R1                        ;求偏差变化量 AEK
        MOV  EK_1, AX
        MOV  AX, R2
        MOV  AEK_1, AX
        TEST R1, 8000H
        JZ   EK1                          ;若偏差 EK 为正数，则不要求补码

```

```

      NEG R1 ;若偏差 EK 为负数，则求偏差 EK 的补码
EK1:  MOV AX, R1 ;判断偏差 EK 是否在积分分离值的范围内
      SUB AX, IBAND
      JC II ;在积分分离值范围内，则跳转到 II，计算积分项
      MOV R3, 00H ;若不在积分分离值范围内，则将积分项清 0
      JMP DDD ;计算微分项
II:   MOV AL, TS ;计算积分项，结果放在 R3 变量中 (R3=EK*TS/KII)
      MOV AH, 00H ;其中 TS 和 KII 均为正数，所以 R3 的正负由 EK 决定
      MOV CX, R1
      MUL CX
      MOV CX, KII
      DIV CX
      MOV R3, AX
      TEST R0, 8000H ;判断积分项的正负
      JZ DDD ;为正数，则跳转去计算微分项
      NEG R3 ;为负数，则将积分项的结果求补码
DDD:  TEST BEK, 8000H ;判断 BEK 的正负
      JZ DDD1 ;为正数，则 BEK 不变
      NEG BEK ;为负数，则求 BEK 的补码
DDD1: MOV AX, BEK ;计算微分项 (R4=KDD*BEK/8TS)
      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 ;判断微分项的正负
      JZ DD1 ;为正数，则结果不要求补码
      NEG R4 ;为负数，则微分项结果 R4 求补码
DD1:  MOV AX, R3 ;积分项和微分项相加，结果放在 R5 变量中
      ADD AX, R4
      MOV R5, AX
      JO L9 ;判断溢出
L2:   MOV AX, R5
      ADD AX, R2
      MOV R6, AX ;R6=R5+R2=积分项+微分项+AEK
      JO L3
L5:   MOV AX, R6 ;计算 KPP*R6
      MOV CX, KPP
      IMUL CX
      MOV CX, 1000H
      IDIV CX
      MOV CX, AX
      RCL AH, 01H ;判断溢出，溢出赋极值
      PUSHF
      RCR AL, 01H
      POPF
      JC LLL1

```



```

        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_1+CK
        ADD  AL, CK_1
        JO   L8
L18:    MOV  CK_1, AL
        ADD  AL, 80H
        MOV  CK, AL
        RET
L8:     TEST R7, 80H                ;CK 溢出处理程序
        JNZ  L17
        MOV  AL, 7FH                ;若为正溢出，则赋给正极值 7FH
        JMP  L18
L17:    MOV  AL, 80H                ;若为负溢出，则赋给赋极值 80H
        JMP  L18
L9:     TEST R3, 8000H
        JNZ  L1
        MOV  R5, 7FFFH                ;若为正溢出，则赋给正极值 7FFFH
        JMP  L2
L1:     MOV  R5, 8000H                ;若为负溢出，则赋给负极值 8000H
        JMP  L2
L3:     TEST R2, 8000H
        JNZ  L4
        MOV  R6, 7FFFH
        JMP  L5
L4:     MOV  R6, 8000H
        JMP  L5
DECSHOW PROC NEAR                ;完成两位十进制数显示子程序
        MOV  DX, 0
        MOV  BX, 10                ;计算 AX/10
        DIV  BX
        ADD  AL, 30H                ;商+30H，即为十位数 ASCII 码
        MOV  AH, 0EH
        INT  10H
        ADD  DL, 30H                ;余+30H，即为个位数 ASCII 码
        MOV  AH, 2
        INT  21H
        RET
DECSHOW ENDP
SEND PROC NEAR                ;完成变量输出到专用图形界面显示
        MOV  BX, 0D800H
        MOV  ES, BX
        MOV  ES: [DI], AX
        RET
SEND ENDP
CODE ENDS
        END  START

```

4.17 温度闭环控制实验

4.17.1 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。

4.17.2 实验设备

PC 机一台，TD-PITD+实验装置一套。

4.17.3 实验内容

温度闭环控制原理如图 4-17-1 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。

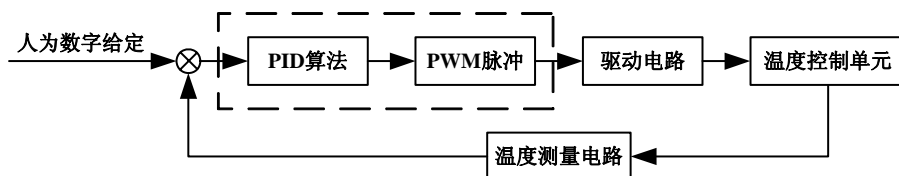
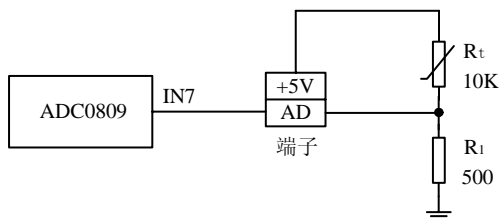


图 4-17-1 温度控制实验原理图

温度控制单元中由 7805 与一个 24Ω 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。实验控制的温度范围为 $50\sim 70^{\circ}\text{C}$ 。

4.17.4 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℃。

4. 17.5 实验步骤


- (1) 实验接线图如图 4-17-2 所示, 按图接线。
- (2) 运行 Tdptit 集成操作软件, 根据实验要求编写实验程序, 实验参数取值范围见表 4-17-1, 编译、链接。
- (3) 运行程序, 观察屏幕上给定温度值与反馈值的输出显示。
- (4) 用软件所带专用图形界面观测的方法: 点击快捷工具栏上 “” 按钮, 启动专用图形界面显示窗口, 即可观察波形显示。

表 4-17-1 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	℃	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

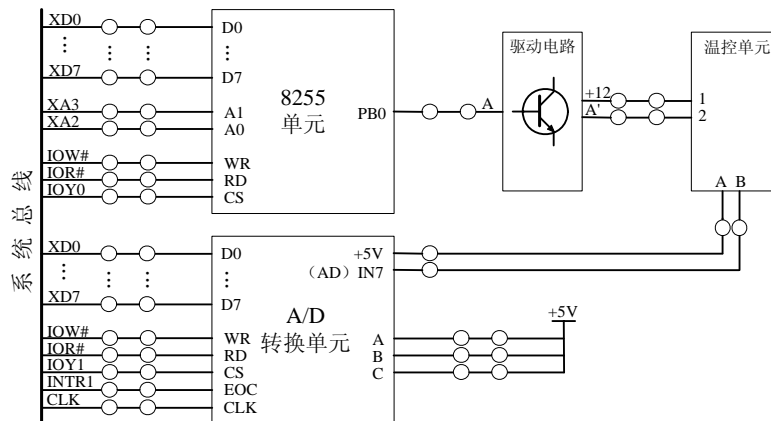


图 4-17-2 温度控制实验线路图

实验程序清单 (Tem. asm)

```

INTR_IVADD    EQU    003CH           ; INTR 对应的中断矢量地址
IOY0          EQU    3000H           ; 片选 IOY0 对应的端口始地址
IOY1          EQU    3040H           ; 片选 IOY1 对应的端口始地址
MY8255_A      EQU    IOY0+00H*4      ; 8255 的 A 口地址
MY8255_B      EQU    IOY0+01H*4      ; 8255 的 B 口地址
MY8255_C      EQU    IOY0+02H*4      ; 8255 的 C 口地址
MY8255_MODE   EQU    IOY0+03H*4      ; 8255 的控制寄存器地址
AD0809        EQU    IOY1+00H*4      ; AD0809 端口地址
PC8254_COUNT0 EQU    40H             ; PC 机内 8254 定时器 0 端口地址
PC8254_MODE   EQU    43H             ; PC 机内 8254 控制寄存器端口地址
STACK1 SEGMENT STACK
    DW 64 DUP(?)
TOP LABEL WORD
STACK1 ENDS
DATA SEGMENT
TABLE1 DB 'Assumed Temperature:', 0AH, 0DH, '$' ; 字符串变量
TABLE2 DB 'Current Temperature:', 0AH, 0DH, '$' ; 字符串变量
ENT DB 0AH, 0DH, '$' ; 换行, 回车
CS_BAK DW ? ; 保存 INTR 原中断处理程序入口段地址的变量
IP_BAK DW ? ; 保存 INTR 原中断处理程序入口偏移地址的变量
IM_BAK DB ? ; 保存 INTR 原中断屏蔽字的变量
CS_BAK1 DW ? ; 保存定时器 0 中断处理程序入口段地址的变量
IP_BAK1 DW ? ; 保存定时器 0 中断处理程序入口偏移地址的变量
IM_BAK1 DB ? ; 保存定时器 0 中断屏蔽字的变量
TS DB 100 ; 采样周期
SPEC DW 40 ; 温度给定值
IBAND DW 0060H ; 积分分离值
KPP DW 1F60H ; 比例系数
KII DW 0010H ; 积分系数
KDD DW 0020H ; 微分系数
YK DW ?
CK DB ?
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 ?
TEMTABLE  DB 14H, 14H, 14H, 14H, 14H, 14H, 14H, 14H, 14H, 14H
           DB 15H, 16H, 17H, 18H, 19H, 1AH, 1BH, 1CH, 1DH, 1EH
           DB 1EH, 1FH, 20H, 21H, 23H, 24H, 25H, 26H, 27H, 28H
           DB 29H, 2AH, 2BH, 2CH, 2DH, 2EH, 2FH, 31H, 32H, 32H
           DB 33H, 34H, 35H, 36H, 37H, 38H, 39H, 3AH, 3BH, 3CH
           DB 3DH, 3EH, 3FH, 40H, 42H, 43H, 44H, 45H, 46H, 47H
           DB 48H, 49H, 4AH, 4BH, 4CH, 4DH, 4EH, 4FH, 50H, 4FH
           DB 50H, 51H, 52H, 53H, 54H, 55H, 56H, 57H, 58H, 59H
           DB 5AH, 5BH, 5CH, 5DH, 5EH, 5FH, 60H, 61H, 62H, 63H
           DB 64H, 64H, 65H, 65H, 66H, 66H, 67H, 68H, 69H, 6AH
           DB 6BH, 6CH, 6DH, 6EH, 6EH, 6FH, 6FH, 70H, 71H, 72H
           DB 73H, 74H, 75H, 76H, 77H, 78H, 79H, 7AH, 7BH, 7CH
           DB 7DH, 7EH, 7FH, 80H, 81H, 82H, 83H, 84H, 84H, 85H
           DB 86H, 87H, 88H, 89H, 8AH, 8BH, 8CH, 8EH, 8FH, 90H
           DB 91H, 92H, 93H, 94H, 95H, 96H, 97H, 98H, 99H, 9AH
           DB 9BH, 9BH, 9CH, 9CH, 9DH, 9DH, 9EH, 9EH, 9FH, 9FH
           DB 0A0H, 0A1H, 0A2H, 0A3H, 0A4H, 0A5H, 0A6H, 0A7H, 0A8H, 0A9H
           DB 0AAH, 0ABH, 0ACH, 0ADH, 0AEH, 0AFH, 0B0H, 0B0H, 0B1H, 0B2H
           DB 0B3H, 0B4H, 0B4H, 0B5H, 0B6H, 0B7H, 0B8H, 0B9H, 0BAH, 0BBH
           DB 0BDH, 0BEH, 0BEH, 0C1H, 0C2H, 0C3H, 0C4H, 0C5H, 0C6H, 0C8H
           DB 0CAH, 0CCH, 0CEH, 0CFH, 0D0H, 0D1H, 0D2H, 0D4H, 0D5H, 0D6H
           DB 0D7H, 0D8H, 0D9H, 0DAH, 0DBH, 0DCH, 0DDH, 0DEH, 0E3H, 0E6H
           DB 0E9H, 0ECH, 0F0H, 0F2H, 0F6H, 0FAH, 0FFH, 0FFH, 0FFH, 0FFH
           DB 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH
           DB 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH
           DB 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH, 0FFH
DATA      ENDS
CODE      SEGMENT
          ASSUME CS:CODE, DS:DATA
START:    MOV  AX, DATA
          MOV  DS, AX

```



```
MOV AX, SPEC ;将给定值输出给专用界面显示
MOV DI, 0D80H
CALL SEND
MOV DX, OFFSET TABLE1 ;显示字符串 1
MOV AH, 09H
INT 21H
MOV AX, SPEC ;显示给定值
CALL DECSHOW
MOV DX, OFFSET ENT ;回车, 换行
MOV AH, 09H
INT 21H
MOV DX, OFFSET TABLE2 ;显示字符串 2
MOV AH, 09H
INT 21H
CLI
MOV AX, 0000H
MOV ES, AX
MOV DI, 0020H
MOV AX, ES: [DI]
MOV IP_BAK1, AX ;保存定时器 0 中断处理程序入口偏移地址
MOV AX, OFFSET TIMERISR
MOV ES: [DI], AX ;设置实验定时中断处理程序入口偏移地址
ADD DI, 2
MOV AX, ES: [DI]
MOV CS_BAK1, AX ;保存定时器 0 中断处理程序入口段地址
MOV AX, SEG TIMERISR
MOV ES: [DI], AX ;设置实验定时中断处理程序入口段地址
IN AL, 21H
MOV IM_BAK1, AL ;保存 INTR 原中断屏蔽字
AND AL, 0F7H
OUT 21H, AL ;打开定时器 0 中断屏蔽位
MOV DI, INTR_IVADD
MOV AX, ES: [DI]
MOV IP_BAK, AX ;保存 INTR 原中断处理程序入口偏移地址
MOV AX, OFFSET MYISR
MOV ES: [DI], AX ;设置当前中断处理程序入口偏移地址
ADD DI, 2
MOV AX, ES: [DI]
MOV CS_BAK, AX ;保存 INTR 原中断处理程序入口段地址
MOV AX, SEG MYISR
MOV ES: [DI], AX ;设置当前中断处理程序入口段地址
IN AL, 21H ;设置中断屏蔽寄存器, 打开 INTR 的屏蔽位
MOV IM_BAK, AL ;保存 INTR 原中断屏蔽字
AND AL, 7FH
OUT 21H, AL
MOV CK, 00H ;变量的初始化
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  AL, 80H                ;初始化 8255
MOV  DX, MY8255_MODE
OUT  DX, AL
MOV  DX, AD0809            ;启动 A/D 采样
OUT  DX, AL
MOV  DX, PC8254_MODE      ;初始化 PC 机定时器 0，定时 10ms
MOV  AL, 36H
OUT  DX, AL
MOV  DX, PC8254_COUNT0
MOV  AL, 69H
OUT  DX, AL
MOV  AL, 2DH
OUT  DX, AL
STI
M1:  CMP  TKMARK, 01H      ;判断采样周期到否？
      JNZ  M1              ;没到则继续等待
      MOV  TKMARK, 00H
M2:  CMP  ADMARK, 01H      ;判断 A/D 采样标志
      JNZ  M2
      MOV  ADMARK, 00H
      MOV  AX, 0000H
      MOV  AL, ADVALUE     ;查温度表
      MOV  BX, OFFSET TEMTABLE
      ADD  BX, AX
      MOV  AL, [BX]
      MOV  YK, AX
      CALL PID              ;调用 PID 子程序，计算控制量 CK
      MOV  AL, CK          ;把控制量 CK 转化成 PWM 输出
      SUB  AL, 80H
      JC   ISO
      MOV  AAAA, AL
      JMP  COU
ISO:  MOV  AL, 00H

```

```

MOV  AAAA, AL
COU:  MOV  AL, 7FH
      SUB  AL, AAAA
      MOV  BBB, AL
      MOV  AX, YK                ;将反馈值 YK 送到屏幕显示
      CALL DECSHOW
      MOV  AX, YK                ;将反馈值 YK 输出给专用界面显示
      MOV  DI, 0D81H
      CALL SEND
      MOV  AL, CK                ;将控制量 CK 输出给专用界面显示
      MOV  DI, 0D82H
      CALL SEND
      MOV  DL, 0DH              ;回车
      MOV  AH, 02H
      INT  21H
      MOV  AH, 1                ;判断是否有按键按下
      INT  16H
      JZ   M1                   ;无按键则跳回继续等待，有则退出
EXIT:  CLI
      MOV  AL, 00H              ;退出时停止驱动信号
      MOV  DX, MY8255_B
      OUT  DX, AL
      MOV  DX, PC8254_MODE      ;恢复 PC 机定时器 0 状态
      MOV  AL, 36H
      OUT  DX, AL
      MOV  DX, PC8254_COUNT0
      MOV  AL, 00H
      OUT  DX, AL
      MOV  AL, 00H
      OUT  DX, AL
      MOV  AX, 0000H            ;恢复 INTR 原中断矢量
      MOV  ES, AX
      MOV  DI, INTR_IVADD
      MOV  AX, IP_BAK           ;恢复 INTR 原中断处理程序入口偏移地址
      MOV  ES: [DI], AX
      ADD  DI, 2
      MOV  AX, CS_BAK           ;恢复 INTR 原中断处理程序入口段地址
      MOV  ES: [DI], AX
      MOV  AL, IM_BAK           ;恢复 INTR 原中断屏蔽寄存器的屏蔽字
      OUT  21H, AL
      MOV  DI, 0020H
      MOV  AX, IP_BAK1          ;恢复定时器 0 中断处理程序入口偏移地址
      MOV  ES: [DI], AX
      ADD  DI, 2
      MOV  AX, CS_BAK1          ;恢复定时器 0 中断处理程序入口段地址
      MOV  ES: [DI], AX
      MOV  AL, IM_BAK1
      OUT  21H, AL             ;恢复屏蔽字
      STI
      MOV  AX, 4C00H
      INT  21H

```



```

MYISR PROC NEAR
    PUSH AX
    PUSH CX
    PUSH DX
    MOV AX, DATA
    MOV DS, AX
    MOV DX, AD0809
    IN AL, DX                ;读取 A/D 转换值
    MOV ADVALUE, AL
    MOV ADMARK, 01H
    MOV AL, 20H              ;向 PC 机内部 8259 发送中断结束命令
    OUT DX, AL
    POP DX
    POP CX
    POP AX
    IRET
MYISR ENDP
TIMERISR PROC NEAR
    PUSH AX
    PUSH CX
    PUSH DX
    MOV AX, DATA
    MOV DS, AX
    MOV DX, AD0809           ;启动 A/D 转换
    OUT DX, AL
    MOV AL, TC                ;采样周期变量递增
    CMP AL, TS
    JNC TT2
    INC TC
    JMP TT1
TT2:  MOV TKMARK, 01H
    MOV TC, 00H
TT1:  CALL KJ                 ;调用 PWM 子程序
    MOV AL, 20H              ;中断结束，发 EOI 命令
    OUT 20H, AL
    POP DX
    POP CX
    POP AX
    IRET
TIMERISR ENDP
KJ PROC NEAR                 ;PWM 子程序
    PUSH AX
    CMP FPWM, 01H            ;PWM 为 1，产生 PWM 的高电平
    JNZ TEST2
    CMP VAA, 00H
    JNZ ANOTO
    MOV FPWM, 02H
    MOV AL, BBB
    CLC
    RCR AL, 01H
    MOV VBB, AL

```

```

        JMP TEST2
ANOTO: DEC VAA
        MOV DX, MY8255_B           ;PB0=1 开始加温
        MOV AL, 01H
        OUT DX, AL
TEST2:  CMP FPWM, 02H              ;PWM 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, MY8255_B           ;PB0=0 停止加温
        MOV AL, 00H
        OUT DX, AL
OUTT:   POP AX
        RET
KJ ENDP
PID:    MOV AX, SPEC               ;PID 子程序
        SUB AX, YK                 ;求偏差 EK
        MOV R0, AX
        MOV R1, AX
        SUB AX, EK_1
        MOV R2, AX
        SUB AX, AEK_1              ;求 BEK
        MOV BEK, AX
        MOV R8, AX
        MOV AX, R1                 ;求偏差变化量 AEK
        MOV EK_1, AX
        MOV AX, R2
        MOV AEK_1, AX
        TEST R1, 8000H
        JZ EK1                    ;若偏差 EK 为正数, 则不要求补码
        NEG R1                     ;若偏差 EK 为负数, 则求偏差 EK 的补码
EK1:    MOV AX, R1                  ;判断偏差 EK 是否在积分分离值的范围内
        SUB AX, IBAND
        JC II                      ;在积分分离值范围内, 则跳转到 II, 计算积分项
        MOV R3, 00H               ;若不在积分分离值范围内, 则将积分项清 0
        JMP DDD                   ;计算微分项
II:     MOV AL, TS                 ;计算积分项, 结果放在 R3 变量中 (R3=EK*TS/KII)
        MOV AH, 00H               ;其中 TS 和 KII 均为正数, 所以 R3 的正负由 EK 决定
        MOV CX, R1
        MUL CX
        MOV CX, KII
        DIV CX
        MOV R3, AX
        TEST R0, 8000H             ;判断积分项的正负

```

```

        JZ   DDD                ;为正数，则跳转去计算微分项
        NEG  R3                ;为负数，则将积分项的结果求补码
DDD:    TEST BEK, 8000H        ;判断 BEK 的正负
        JZ   DDD1             ;为正数，则 BEK 不变
        NEG  BEK              ;为负数，则求 BEK 的补码
DDD1:   MOV  AX, BEK           ;计算微分项 (R4=KDD*BEK/8TS)
        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        ;判断微分项的正负
        JZ   DD1              ;为正数，则结果不要求补码
        NEG  R4                ;为负数，则微分项结果 R4 求补码
DD1:    MOV  AX, R3            ;积分项和微分项相加，结果放在 R5 变量中
        ADD  AX, R4
        MOV  R5, AX
        JO   L9                ;判断溢出
L2:     MOV  AX, R5
        ADD  AX, R2
        MOV  R6, AX            ;R6=R5+R2=积分项+微分项+AEK
        JO   L3
L5:     MOV  AX, R6            ;计算 KPP*R6
        MOV  CX, KPP
        IMUL CX
        MOV  CX, 1000H
        IDIV CX
        MOV  CX, AX
        RCL  AH, 01H           ;判断溢出，溢出赋极值
        PUSHF
        RCR  AL, 01H
        POPF
        JC   LLL1
        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_1+CK
        ADD  AL, CK_1
        JO   L8

```

```

L18:  MOV  CK_1, AL
      ADD  AL, 80H
      MOV  CK, AL
      RET

L8:   TEST R7, 80H                ;CK 溢出处理程序
      JNZ  L17
      MOV  AL, 7FFH              ;若为正溢出, 则赋给正极值 7FFH
      JMP  L18
L17:  MOV  AL, 80H                ;若为负溢出, 则赋给赋极值 80H
      JMP  L18
L9:   TEST R3, 8000H
      JNZ  L1
      MOV  R5, 7FFFH             ;若为正溢出, 则赋给正极值 7FFFH
      JMP  L2
L1:   MOV  R5, 8000H             ;若为负溢出, 则赋给负极值 8000H
      JMP  L2
L3:   TEST R2, 8000H
      JNZ  L4
      MOV  R6, 7FFFH
      JMP  L5
L4:   MOV  R6, 8000H
      JMP  L5

DECSHOW PROC NEAR                ;完成两位十进制数显示子程序
      MOV  DX, 0
      MOV  BX, 10                ;计算 AX/10
      DIV  BX
      ADD  AL, 30H                ;商+30H, 即为十位数 ASCII 码
      MOV  AH, 0EH
      INT  10H
      ADD  DL, 30H                ;余+30H, 即为个位数 ASCII 码
      MOV  AH, 2
      INT  21H
      RET

DECSHOW ENDP
SEND PROC NEAR                  ;完成变量输出到专用图形界面显示
      MOV  BX, 0D800H
      MOV  ES, BX
      MOV  ES: [DI], AX
      RET
SEND ENDP
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的情况已经在第 1 章讲过，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 5-2-1 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。

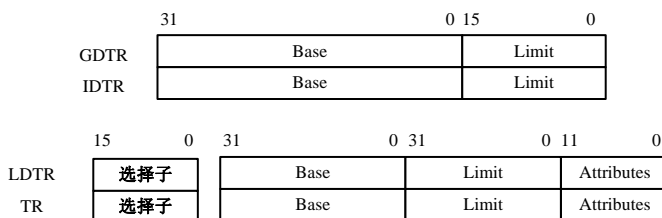


图 5-2-1 32 位处理器的系统地址寄存器

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 5-2-2 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。

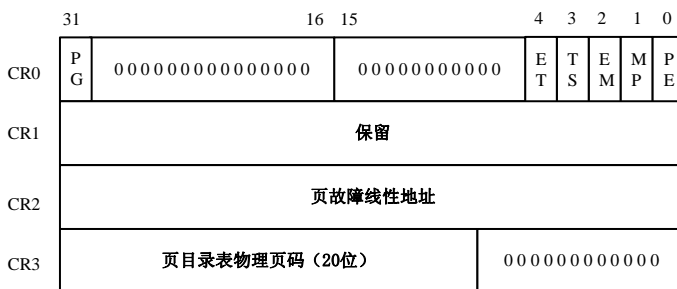


图 5-2-2 32 位处理器的控制寄存器

表 5-2-1 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 5-2-1 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 5-3-1 所示。

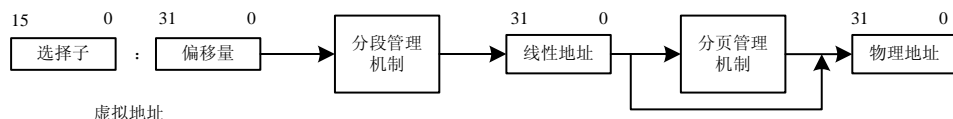


图 5-3-1 保护模式下的地址转换示意图

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 5-3-2 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）。

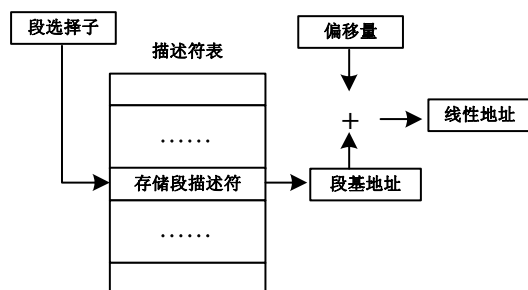


图 5-3-2 保护模式下的分段管理示意图

(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 5-3-3 所示。

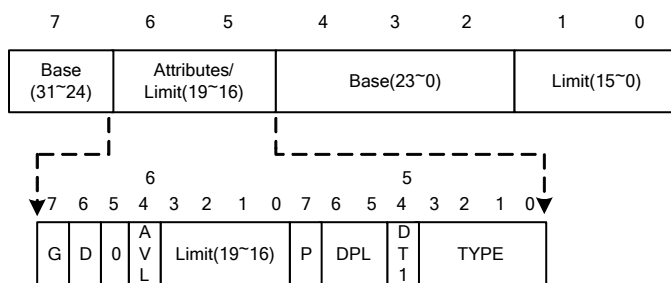


图 5-3-3 存储段描述符格式

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）；

TYPE: 决定了存储段的属性, 具体属性值见表 5-3-1。

表 5-3-1 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

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。

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 5-3-5 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。

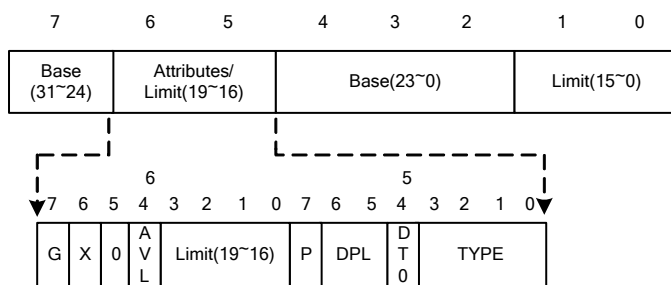


图 5-3-4 系统段描述符格式

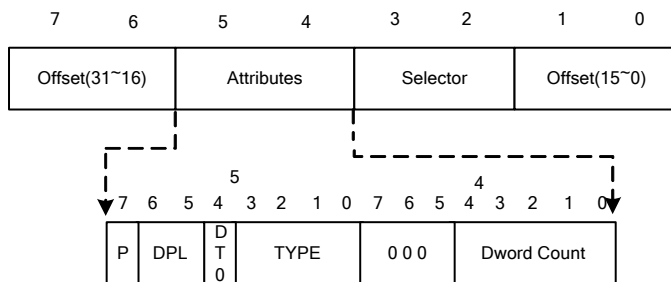


图 5-3-5 门描述符格式

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符和门描述符。系统段描述符和门描述符的属性值如表 5-3-2 所示。

表 5-3-2 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 5-3-6 所示。

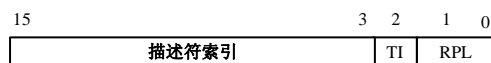


图 5-3-6 段选择子格式

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，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 5-3-7 所示。

31	0 31	0 9	0
32位段基地址		32位段界限	其他属性

图 5-3-7 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权限级检查工作，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： $CPL \leq DPL$ ， $RPL \leq DPL$ ，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 $CPL \leq IOPL$ ，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 5-3-8 所示。

1. 执行代码

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

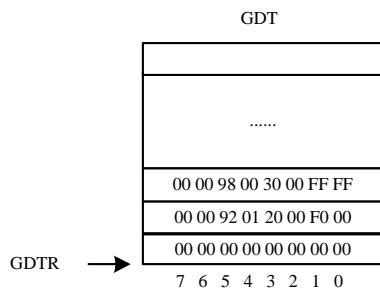


图 5-3-8 GDT 表内容

2. 执行过程

寻址过程如图 5-3-9 所示。

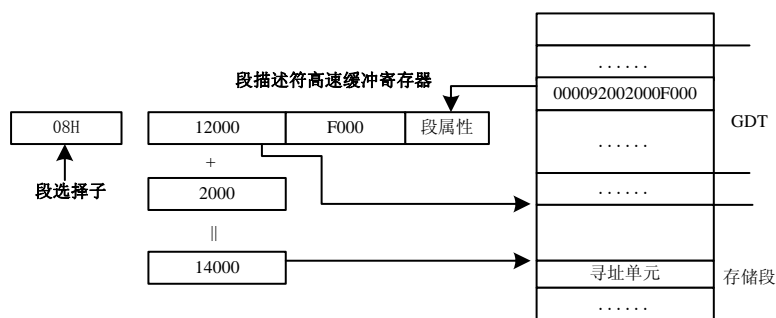


图 5-3-9 寻址过程示意图

(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 5-4-1 所示。任务状态段可以分成以下几个区域。

31			0
0000000000000000		链接字段	0H
ESP0			4H
0000000000000000		SS0	8H
ESP1			0CH
0000000000000000		SS1	10H
ESP2			14H
0000000000000000		SS2	18H
CR3			1CH
EIP			20H
EFLAGS			24H
EAX			28H
ECX			2CH
EDX			30H
EBX			34H
ESP			38H
EBP			3CH
ESI			40H
EDI			44H
0000000000000000		ES	48H
0000000000000000		CS	4CH
0000000000000000		SS	50H
0000000000000000		DS	54H
0000000000000000		FS	58H
0000000000000000		GS	5CH
0000000000000000		LDT	60H
I/O许可位图		0000000000000000	T 64H

图 5-4-1 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 $CPL \geq DPL$ 。
- (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 ≥ 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 5-7-1 中列出了保护模式下异常的类型和说明。

表 5-7-1 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 5-7-1 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。

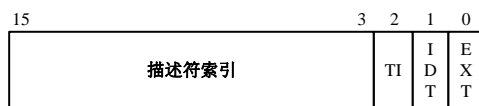


图 5-7-1 异常出错码格式

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 5-7-2 及图 5-7-3 示，图 5-7-2 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 5-7-3 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。

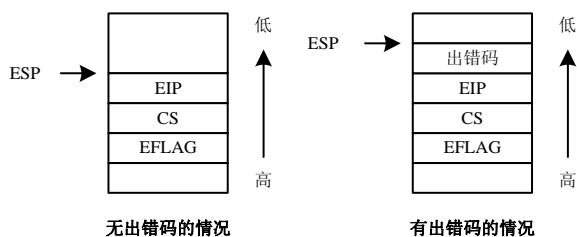


图 5-7-2 无特权级变换时的堆栈

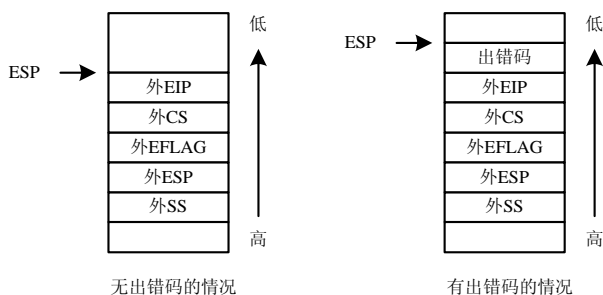


图 5-7-3 有特权级变换时的堆栈

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 基于 Tddebug 的保护模式程序设计

调试系统在装入实验程序时，要对程序进行重新定位，并将用户定义的部分描述符填入系统描述符表。而现有的编译、连接器如 Tasm 3.1、Tlink 5.1，Masm 5.0、Link 3.6 支持的是 DOS 格式，MASM 6.0 以上版本支持的是 Windows 格式，都无法提供调试工具所需的重新定位信息。为了从程序中获取必要的重新定位信息，要求用户在编写实验程序时必须按照特定的格式来编写。

5.8.1 指令集选择

由于在缺省情况下，MASM 和 TASM 只能识别 8086/8088 的指令，为了让编译器可以识别 80386、80486 等 CPU 的新增指令或功能增强的指令，必须在程序中使用提示处理器类型的伪指令。则在实验程序中，必须使用 .386，.386p 等伪指令。

5.8.2 基本约定

1. 描述符的声明顺序

实验程序中需要将在程序中使用的所有描述符定义在程序的最前端。首先使用一个全“F”的描述符作为定义的开始，其后要声明在全局描述符表 GDT 中出现的描述符。调试系统要求实验程序中至少应有一个起始代码段描述符在 GDT 表中。声明完 GDT 表中的描述符，需要再使用一个全“F”的描述符作为区分于局部描述符表 LDT 的界限。用户可以定义多个 LDT 表，LDT 表的定义必须连续且在定义结束后，再使用一个全“F”描述符作为结尾。即实验程序声明描述符时需要定义三个全“F”的描述符作为分界线，标识 GDT 和 LDT 的开始及结尾。

每个段描述符中包含了所描述的段的线性地址，该线性地址是在调试软件装入实验程序时进行重新定位得来的。重新定位的信息是在编写实验程序时给定的：在声明描述符时，描述符线性地址的低 16 位中要写入对应段的标号，做为重新定位信息。

2. 选择子定义

实验程序中描述符对应的选择子均由用户自己定义。调试系统在 GDT 表的最前端预留了 128 个描述符的空间给实验程序使用，实验程序中段的的选择子可以由用户在 128 个描述符的选择子范围内选定。

3. 程序运行的起始

每个 32 位保护模式下的实验程序都需要安排一个 0 级的代码段作为程序运行的开始。编程时需要在用户可以使用的第一个描述符处描述该代码段，即规定程序起始运行段对应的选择子为 08H。

4. 结束程序运行

调试系统为用户专门提供了 OFFH 号软件中断，作为程序运行结束返回调试系统的出口。用户可以在实验程序的最后使用“INT OFFH”指令，正常结束程序运行。

5.8.3 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
```

5.8.4 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：

```
-----
;
;P-example.asm
;演示实方式和保护方式切换
;
-----
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
Code            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
```

```

GDTLen      =      $-GDT                ;全局描述符表长度
VGDTTR      PDesc  <GDTLen-1,>         ;伪描述符
Code_Sel     =      Code-GDT            ;代码段选择子
DataS_Sel    =      DataS-GDT           ;源数据段选择子
DataD_Sel    =      DataD-GDT           ;目标数据段选择子
DataLen      =      960                 ;缓冲区字节长度
DSEG         ENDS                       ;数据段定义结束
;-----
Data1        SEGMENT PARA USE16
MES1        DB 'This tangdu speaking!',13,10,'$'
ML          =  $-MES1-1
Data1        ENDS
;-----
Data2        SEGMENT PARA USE16
BUF         DB      64 DUP(?)
Data2        ENDS
;-----
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
    add     ax,OFFSET GDT              ;计算并设置基地址
    adc     dx,0                      ;界限已在定义时设置好
    mov     WORD PTR VGDTTR.Base,ax
    mov     WORD PTR VGDTTR.Base+2,dx
    ;设置代码段描述符
    mov     ax,cs
    mul     bx
    mov     WORD PTR Code.BaseL,ax    ;代码段开始偏移为 0
    mov     BYTE PTR Code.BaseM,d1    ;代码段界限已在定义时设置好
    mov     BYTE PTR Code.BaseH,dh
    ;设置源代码段描述符
    mov     ax,Data1
    mul     bx
    mov     WORD PTR DataS.BaseL,ax
    mov     BYTE PTR DataS.BaseM,d1
    mov     BYTE PTR DataS.BaseH,dh
    ;设置目标代码段描述符
    mov     ax,Data2
    mul     bx
    mov     WORD PTR DataD.BaseL,ax
    mov     BYTE PTR DataD.BaseM,d1
    mov     BYTE PTR DataD.BaseH,dh
    ;加载 GDTR
    lgdt    QWORD PTR VGDTTR
    cli                                ;关中断
    EnableA20                          ;打开地址线 A20

```

```

;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  Code_Sel, <OFFSET Virtual>
Virtual:
;现在开始在保护方式下运行
mov     ax, DataS_Sel
mov     ds, ax                ;加载显示缓冲区描述符
mov     ax, DataD_Sel
mov     es, ax
cld
xor     di, di
xor     si, si
mov     cx, ML                ;设置数据长度
repz    movsb
mov     al, '$'
mov     es:[si], al
;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
DisableA20
sti
mov     ax, Data1
mov     ds, ax
mov     dx, offset Mes1
mov     ah, 09h
int     21h
mov     ax, Data2
mov     ds, ax
mov     dx, offset BUF
mov     ah, 09h
int     21h
mov     ax, 4c00h
int     21h
Start   ENDP
;-----
CSEG    ENDS                ;代码段定义结束
END     Start

```

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。这些实验需在保护模式集成操作软件 Tddebug 下完成。

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. 掌握使用选择符访问段的寻址方法。

6.1.2 实验设备

PC 机一台，TD-PITD+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 SCODE，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 SCODE_SEL，DATAS_SEL，DATAD_SEL。为这三个段分别定义描述符：

(1) 代码段描述符：SCODE DESC <Clen,CSEG,,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

段基地址说明：定义代码段的标号为 CSEG，则在段基地址处填写 CSEG，为调试器提供重定位信息。

段界限说明：段界限定义为 Clen。

(2) 源数据段描述符：DATAS DESC <DLEN,DSEG1,,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

段基地址说明：定义源数据段标号为 DSEG1，则在段基地址处填写 CSEG，为调试器提供重定位信息。

段界限说明：定义段界限为 DLEN。

(3) 目标数据段描述符 DATAD DESC <BUFLen,DSEG2,,ATDW,,>

目标数据段描述符的内容基本与源数据段的内容相同，只要修改段基地址和段界限的定义即可。

为了给装入程序提供重定位信息，三个存储段描述符中地址的低 16 位用每个描述符对应

段的标号来填写。在程序装入内存时，调试系统会根据地址的低 16 位重定位该段对应的真实物理地址，并将该地址写入描述符中（系统没有使用分页机制，线性地址等价于物理地址）。在实验中可查询 GDT 表来确定每个段的真实物理地址。

在程序定义过程中，首先使用一个全“F”的描述符作为定义的开始，然后定义代码段描述符 Scode、源数据段的描述符 DSEG1 和目标数据段描述符 DSEG2。为了区分 LDT 表和 GDT 表的定义，再使用一个全“F”的描述符作为界限。由于本实验中不使用 LDT 表，则再使用一个全“F”的描述符结束描述符的声明。

本程序可实现将一个数据段中数据搬移到另一个数据段的过程。传输过程中可使用 DS，ES 两个段寄存器，其中 DS 装入源数据段的选择符 DATAS_SEL，ES 装入目标数据段的选择符 DATAD_SEL。在实验程序的最后使用“INT 0FFH”指令，正常结束程序运行。

3. 实验程序清单（P1-1.asm）

```
. 386P
DESC      STRUC
LIMITL    DW    0
BASEL     DW    0
BASEM     DB    0
ATTR      DB    0
LIMITH    DB    0
BASEH     DB    0
DESC      ENDS
ATCE      =    98H
ATDR      =    90H
ATDW      =    92H
;-----
DSEG      SEGMENT USE16
GDT       LABEL  BYTE
ID1       DESC  <OFFFFH, OFFFFH, OFFH, OFFH, OFFH, OFFH>
SCODE     DESC  <OFFFFH, CSEG, , ATCE, , >
DATAS     DESC  <D1LEN-1, DSEG1, , ATDW, , >
DATAD     DESC  <BUFLen-1, DSEG2, , ATDW, , >
GDTLEN    =    $-GDT
SCODE_SEL =    SCODE-GDT
DATAS_SEL =    DATAS-GDT
DATAD_SEL =    DATAD-GDT
ID2       DESC  <OFFFFH, OFFFFH, OFFH, OFFH, OFFH, OFFH >
ID3       DESC  <OFFFFH, OFFFFH, OFFH, OFFH, OFFH, OFFH >
DSEG      ENDS
;-----
DSEG1     SEGMENT USE16
BUF       DB    00H, 11H, 22H, 33H, 44H, 55H, 66H, 77H
          DB    88H, 99H, 0AAH, 0BBH, 0CCH, 0DDH, 0EEH, 0FFH
D1LEN     =    $
DSEG1     ENDS
;-----
DSEG2     SEGMENT USE16
BUFLen    =    256
BUFFER    DB  BUFLen DUP(0)
```

```

DSEG2      ENDS
;-----
CSEG      SEGMENT USE16
            ASSUME CS:CSEG, DS:DSEG
START      PROC
            MOV AX, DATAS_SEL
            MOV DS, AX
            MOV AX, DATAD_SEL
            MOV ES, AX
            CLD
            XOR SI, SI
            XOR DI, DI
            MOV CX, 16
M1:         MOVSB
            LOOP M1
            INT 0FFH
START      ENDP
CLEN       = $-1
CSEG      ENDS
END        START

```

4. 实验步骤

(1) 进入纯 DOS 环境，运行 Tddebug 集成操作软件。

(2) 运行 Tddebug 软件，使用 Alt+E 选择 Edit 菜单项进入程序编辑环境。按实验要求编写程序。实验参考流程图如图 6-1-1。

(3) 程序编写完后保存退出，使用 Alt+C 选择 Compile 菜单中的 Compile 和 Link 命令对实验程序进行编译、链接。

(4) 编译输出信息表示无误后，使用 Alt+P 选择 Pmrun 命令装入实验程序，如果装入成功，屏幕上会显示“Load OK!”，否则，会给出相应的错误提示信息。

(5) 若程序装入成功，则进入保护模式调试环境。在寄存器显示区可以查看寄存器的初始状态。在数据显示区，将显示 0458:00000000 开始的内容。在程序显示区中将显示从 0008:00000000 开始的代码段内容。

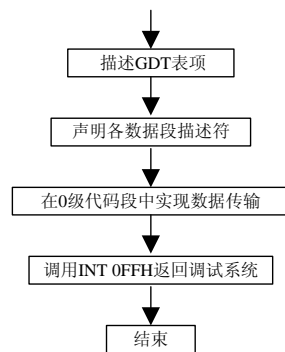


图 6-1-1 全局描述符表实验参考流程图

(6) 在命令输入行使用 GDT 命令查询系统的 GDT 表，并且查看实验程序中声明的代码段、数据段描述符在 GDT 表中的位置以及对应段的物理地址、段属性、段界限等。

(7) 使用 D 命令查看源数据区的数据和目的数据区数据内容。本实验中，源数据段的段地址为 0010，目的数据段的段地址为 0018。命令格式为 D0010:00000000。

(8) 按 F9 运行程序，如果程序正常运行结束，命令显示区中将显示“Correct Running”。

(9) 运行结束后，再次查看目的数据区内容，观察数据传输是否正确。

(10) 还可以使用 F7 单步执行程序，在单步过程中使用 D 命令查询源数据区及目标数据区中的数据，查看各寄存器的状态变化，详细观察程序执行过程。

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 < LDTLen-1,DSEG1,,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

段基址说明: 需要在重定位后确定，但可以知道，该描述符对应的数据段是 DSEG1

段界限说明: 段界限为 LDTLen-1

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. 实验程序清单 (P1-2. asm)

```

INCLUDE      386SCD. INC
;-----
DSEG          SEGMENT use16
;-----
GDT           LABEL   BYTE           ;全局描述符表
ID1           Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh>
MCode         Desc    <0ffffh, CSEG, , ATCE, , >      ;代码段描述符
LDTable       Desc    <LDTLen-1, DSEG1, , ATLD, , >    ;局部描述符表段的描述符
GDTLen        =       $-GDT          ;全局描述符表长度
MCode_Sel     =       MCode-GDT
LDT_Sel       =       LDTable-GDT    ;局部描述符表段的选择子
ID2           Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh>
DSEG          ENDS                  ;数据段定义结束
;-----
DSEG1         SEGMENT use16
LDT           LABEL   BYTE           ;局部描述符表
DataS         Desc    <0ffffh, 0, 11h, ATDW, , >      ;源数据段描述符
DataD         Desc    <0ffffh, Dseg2, , ATDW, , >      ;目标数据段描述符
DataS_Sel     =       DataS-LDT+TIL
DataD_Sel     =       DataD-LDT+TIL
LDTLen        =       $-LDT          ;局部描述符表长度
ID3           Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh>
DSEG1         ENDS
;-----
Dseg2         Segment use16
BufLen        =       256            ;缓冲区字节长度
Buffer        DB      BufLen DUP(0) ;缓冲区
Dseg2         ends
;-----
CSEG          SEGMENT use16          ;16 位代码段
;-----
Start         PROC
mov           ax, LDT_Sel
lldt          ax
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
cld
xor           si, si
xor           di, di                  ;设置指针初值
mov           cx, BufLen/4           ;设置 4 字节为单位的缓冲区长度
repz         movsd                    ;传送
int           0ffh
Start         ENDP
CSEG          ENDS                  ;代码段定义结束
;-----
END           Start

```

4. 实验步骤

(1) 进入纯 DOS 环境, 运行 Tddebug 集成操作软件。

(2) 运行 Tddebug 软件, 使用 Alt+E 选择 Edit 菜单项进入程序编辑环境。按实验要求编写程序。实验参考流程图如图 6-1-2。

(3) 程序编写完后保存退出, 使用 Alt+C 选择 Compile 菜单中的 Compile 和 Link 命令对实验程序进行编译、链接。

(4) 编译输出信息表示无误后, 使用 Alt+P 选择 Pmrun 命令装入实验程序, 如果装入成功, 屏幕上会显示 “Load OK!”, 否则, 会给出相应的错误提示信息。

(5) 若程序装入成功, 则进入保护模式调试环境。在命令输入行使用 GDT 命令查询系统的 GDT 表, 并且查看实验程序中声明的代码段、数据段描述符在 GDT 表中的位置以及对应段的物理地址、段属性、段界限等。

(6) 使用 F7 单步执行程序, 执行 LLDT AX 语句后, 使用 LDT 命令查看 LDT 局部描述符表的内容。

(7) 根据 LDT 内容使用 D 命令查看源数据区的数据和目的数据区数据内容。

(8) 按 F9 运行程序, 如果程序正常运行结束, 命令显示区中将显示 “Correct Running”。

(9) 运行结束后, 再次查看目的数据区内容, 观察数据传输是否正确。

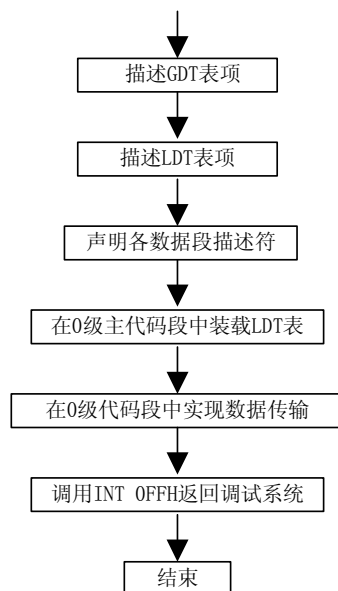


图 6-1-2 局部描述符表实验参考流程图

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-PITD+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

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。实验中各个段的描述符定义如下。

ATLDT	EQU	82H	;局部描述符表段类型值
Codem	Desc	<CodemLen-1,CodemSeg,,ATCE,,>	;主程序段描述符
LDTTable	Desc	<LDTLen-1,LDTSeg,,ATLDT,,>	;局部描述符表段的描述符
Code1	Desc	<Code1Len-1,Code1Seg,,ATCE,,>	;0 级代码段 1
Code2	Desc	<Code2Len-1,Code2Seg,,ATCE,,>	;0 级代码段 2
DStack0	Desc	<DStack0Len-1,DStack0Seg,,ATDW,,>	;0 级堆栈段描述符
DDataS	Desc	<DDataSLen-1,DDataSSeg,,ATDW,,>	;源数据段描述符
DDataO	Desc	<DDataOLen-1,DDataOSeg,,ATDW,,>	;目的数据段描述符

3. 实验程序清单 (P2-1.asm)

```

INCLUDE      386SCD. INC
;
TDataSeg     SEGMENT PARA USE16                      ;全局描述符表数据段(16 位)
GDT          LABEL   BYTE                            ;全局描述符表
ID1          Desc   <0ffffh,0ffffh,0ffh,0ffh,0ffh,0ffh> ;标记描述符 1
Codem        Desc   <CodemLen-1,CodemSeg,,ATCE,,>      ;主程序段描述符
LDTTable     Desc   <LDTLen-1,LDTSeg,,ATLDT,,>        ;局部描述符表段的描述符
Codem_Sel    =      Codem-GDT                        ;主程序段选择子
LDT_Sel      =      LDTTable-GDT                    ;局部描述符表段选择子
ID2          Desc   <0ffffh,0ffffh,0ffh,0ffh,0ffh,0ffh> ;标记描述符 2
TDataSeg     ENDS                                    ;全局描述符表段定义结束
;
LDTSeg       SEGMENT PARA USE16                      ;局部描述符表数据段(16 位)
LDT          LABEL   BYTE                            ;局部描述符表
Code1        Desc   <Code1Len-1,Code1Seg,,ATCE,,>      ;0 级程序段 1
Code2        Desc   <Code2Len-1,Code2Seg,,ATCE,,>      ;0 级程序段 2
DStack0      Desc   <DStack0Len-1,DStack0Seg,,ATDW,,>  ;0 级堆栈段描述符
DDataS       Desc   <DDataSLen-1,DDataSSeg,,ATDW,,>    ;源数据段描述符
DDataO       Desc   <DDataOLen-1,DDataOSeg,,ATDW,,>    ;目的数据段描述符
LDTLen       =      $-LDT                            ;LDT 所占字节数
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
DStack0_Sel  =      DStack0-LDT+TIL                  ;0 级堆栈描述符选择子
DDataS_Sel   =      DDataS-LDT+TIL                   ;0 级堆栈描述符选择子
DDataO_Sel   =      DDataO-LDT+TIL                   ;0 级堆栈描述符选择子
ID3          Desc   <0ffffh,0ffffh,0ffh,0ffh,0ffh,0ffh> ;局部描述符表段定义结束
LDTSeg       ENDS

```

```

;-----
DStack0Seg    SEGMENT  PARA USE16                                ;0 级堆栈段
DStack0Len    =        512
               db      512 dup(2)
DStack0Seg    ENDS                                              ;0 级堆栈段结束
;-----
DDataSSeg     SEGMENT  PARA USE16
DTest         DB      11h, 22h, 33h, 44h, 55h, 66h, 77h, 88h
DDataSL en    =      $
DDataSSeg     ENDS
;-----
DData0Seg     SEGMENT  PARA USE16
DData0Len     =      128
               DB      DData0Len DUP(?)
DData0Seg     ENDS
;-----
Code2Seg       SEGMENT  PARA USE16                                ;任务代码段
               ASSUME  CS:Code2Seg
               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
               cld
               xor     si, si
               xor     di, di                                    ;设置指针初值
               mov     cx, 8                                    ;设置传送长度
               ml:     movsb                                     ;传送
                       loop ml
               retf
Code2Len       =      $
Code2Seg       ENDS                                              ;代码段定义结束
;-----
Code1Seg       SEGMENT  PARA USE16
               ASSUME  CS:Code1Seg
               mov     esp, DStack0Len-1                        ;为任务准备堆栈
               mov     ax, DStack0_Sel
               mov     ss, ax
               Call16  Code2_Sel, 0
               int     0ffh                                     ;返回调试系统
Code1Len       =      $
Code1Seg       ENDS
;-----
CodeMSeg       SEGMENT  PARA USE16
               ASSUME  CS:CodeMSeg
;-----
start          PROC
               mov     ax, LDT_Sel
               LLDT    ax                                       ;加载局部描述符表寄存器 LDTR
               JUMP16  Code1_Sel, 0                             ;跳转到任务 1 代码段
CodeMLen       =      $
start          ENDP

```



```

;-----
CodeMSeg      ENDS
               END      Start

```

(1) LLDT AX

由于实验要从主程序段转移到任务的 CODE1 代码段中, 而 CODE1 的段描述符在 LDT 表中, 所以主程序运行 JMP 指令之前首先要加载任务的局部描述符表。任务的局部描述符表的选择子为 LDT_SEL, 所以只要将 LDT_SEL 装入 LDTR 即可。LLDT 指令为加载局部描述符表的指令。

(2) JUMP16 CODE1_SEL, 0

CODE1_SEL 是指向代码段 1 的选择子, 因为是在 LDT 表中描述的, 所以 TI 位为 1。使用 JUMP16 时, CPU 的 CPL 为 0, CODE1_SEL 中的 RPL 为 0, 而 CODE1_SEL 中的 DPL 为 0, 满足转移的条件 ($CPL=DPL$, $RPL \leq DPL$), 则 CODE1_SEL 被装入 CS, CS 对应描述符的内容被装入高速缓冲寄存器, EIP 被置为零, 成功转移到 CODE1。

(3) CALL16 CODE2_SEL, 0

CODE2_SEL 是指向代码段 2 的选择子, 因为是在 LDT 表中描述的, 所以 TI 位为 1。使用 CALL16 时, CPU 的 CPL 为 0, CODE2_SEL 中的 RPL 为 0, 而 CODE2_SEL 对应的描述符的 DPL 为 0, 满足转移的条件 ($CPL=DPL$, $RPL \leq DPL$), 则当前 CS, EIP (CS 为 CODE1_SEL) 被保存在堆栈中, 而 CODE2_SEL 被装入 CS, CS 对应描述符的内容被装入高速缓冲寄存器, EIP 被置为零, 成功转移到 CODE2。

(4) RETF

从堆栈中弹出返回地址的选择子, 偏移, 判断 RPL, CPL, DPL 的值 (均为 0 级) 符合转移的条件成功切换回 CODE1。

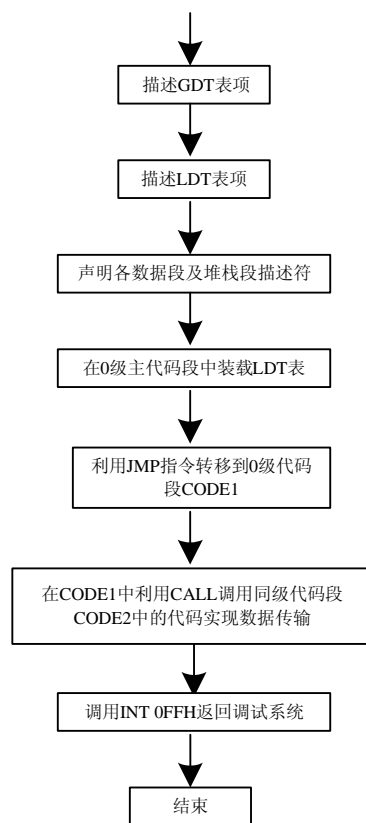


图 6-2-1 任务内无特权级变换的控制转移实验参考流程图

4. 实验步骤

- (1) 进入纯 DOS 环境, 运行 Tddebug 集成操作软件。
- (2) 按实验要求编写实验程序, 实验程序参考流程如图 6-2-1 所示。
- (3) 对实验程序进行编译、链接无误后使用 Pmrun 装入程序。
- (4) 使用 GDT 命令查询系统的 GDT 表, 并且查看实验程序中声明的代码段、LDT 表的描述符在 GDT 表中的位置以及对应段的物理地址、段属性、段界限等。
- (6) 按 F7 单步执行程序, 在执行到 LLDT AX 后, 用 LDT 命令查看 LDT 表的内容, 可以

看到 LDT 表中的代码段、数据段及堆栈段描述符。

(7) 继续单步执行或在 CODE1 中设置断点,可以看到在执行了 JUMP16 CODE1_SEL,0 (JMP 0004:00000000)指令后,CS:EIP 指向了 CODE1 的入口处(0004:00000000)。

(8) 继续单步,执行 CALL CODE2_SEL,0 (000C:00000000)后,可以看到 CS:EIP 指向了 CODE2 的入口处。

(9) 继续单步到 RET,可以看到执行 RET 后,CS:EIP 指向了 CODE1 中的 INT FFH。

(10) 使用 D 命令验证源、目的数据段中的数据是否一致。

(11) 继续单步执行到 INT FFH,程序运行结束。

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。

实验从主程序段开始，应首先加载任务的 TSS 和 LDT 表，使这些表对系统可用。在从 0 级到三级的转换开始之前，应将 3 级代码段入口对应的堆栈段选择符、堆栈指针、代码段选择符、偏移地址依次压入 0 级堆栈段，然后执行 RETF 指令将堆栈中的内容弹出。由于在保护模式下，过程的返回是从堆栈中弹出的地址是 48 位地址全指针的 2 个双字，所以 0 级代码段是一个 32 位的代码段的时候，执行 RETF 才可以弹出 32 位的偏移和扩展的选择符。

3. 实验程序清单 (P2-2. asm)

```

INCLUDE      386SCD. INC
;-----
TDataSeg      SEGMENT PARA USE16                      ;全局描述符表数据段(16 位)
GDT            LABEL   BYTE                            ;全局描述符表
ID1            Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 1
Codem          Desc    <CodemLen-1, CodemSeg, , ATCE, , >      ;主程序段描述符
LDTable        Desc    <LDTLen-1, LDTSeg, , ATLD, , >          ;局部描述符表段的描述符
TSSTable       Desc    <TssLen-1, TSSSeg, , AT386TSS, , >      ;任务状态段 TSS 描述符
ID2            Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 2
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
TDataSeg       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    <DCode0Len-1, DCode0Seg, , ATCE, D32, > ;代码段描述符(32 位段)
DCode3         Desc    <DCode3Len-1, DCode3Seg, , ATCE+DPL3, , > ;代码段描述符(16 位段, DPL=3)
DStack0        Desc    <DStack0Len-1, DStack0Seg, , ATDW, , >   ;0 级堆栈段描述符
DStack3        Desc    <DStack3Len-1, DStack3Seg, , ATDW+DPL3, , > ;3 级堆栈段描述符
DDataS         Desc    <DDataSLen-1, DDataSSeg, , ATDW+DPL3, , > ;源数据段描述符
DData0         Desc    <DData0Len-1, DData0Seg, , ATDW+DPL3, , > ;目的数据段描述符
ToCode0        Gate    <offset pend- offset pstart, DCode0_Sel, , AT386CGate+DPL3, >
;指向 0 级程序段的调用门描述符
DCode0_Sel     =       DCode0-LDT+TIL                  ;代码段描述符选择子
DCode3_Sel     =       DCode3-LDT+TIL+RPL3             ;过渡代码段描述符选择子(RPL=3)
DStack0_Sel    =       DStack0-LDT+TIL                 ;0 级堆栈描述符选择子
DStack3_Sel    =       DStack3-LDT+TIL+RPL3            ;3 级堆栈描述符选择子(RPL=3)
DDataS_Sel     =       DDataS-LDT+TIL                  ;源数据段描述符选择子
DData0_Sel     =       DData0-LDT+TIL                  ;目的数据段描述符选择子
ToCode0_Sel    =       ToCode0-LDT+TIL                 ;调用门描述选择子
LDTLen         =       $-LDT
ID3            Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 3
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          ;0 级堆栈段
DStack0Len =      512
        DB      DStack0Len DUP(?)
DStack0Seg ENDS
;-----
DStack3Seg SEGMENT PARA USE16          ;3 级堆栈段
DStack3Len =      512
        DB      DStack3Len DUP(?)
DStack3Seg ENDS
;-----
DDataSSeg SEGMENT PARA USE16
DTest     DB      11h, 22h, 33h, 44h, 55h, 66h, 77h, 88h
DDataSLen =      $
DDataSSeg ENDS
;-----
DData0Seg SEGMENT PARA USE16
DData0Len =      128
        DB      DData0Len DUP(?)
DData0Seg ENDS
;-----
DCode0Seg SEGMENT PARA USE32
ASSUME    CS:DCode0Seg
pstart:   mov     esp, DStack0Len-1
          mov     ax, DStack0_Sel      ;准备切换到 3 级程序段
          mov     ss, ax

```

```

        push    dword ptr DStack3_Sel
        push    dword ptr DStack3Len
        push    dword ptr DCode3_Sel
        push    dword ptr 0
        retf                                ;由 0 级程序段转移到 3 级程序段
pend:    int     0ffh
DCode0Len =     $
DCode0Seg ENDS
;-----
DCode3Seg SEGMENT PARA USE16
        ASSUME  CS:DCode3Seg
        mov     ax, DDataS_Sel
        mov     ds, ax                    ;加载源数据段描述符
        mov     ax, DData0_Sel
        mov     es, ax                    ;加载目标数据段描述符
        cld
        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
        Call16  ToCode0_Sel, 0           ;由 3 级程序段转移到 0 级程序段
DCode3Len =     $
DCode3Seg ENDS
;-----
CodeMSeg SEGMENT PARA USE16
        ASSUME  CS:CodeMSeg
Start    Proc
        mov     ax, TSS_Sel              ;装载 ldt, tss 表
        ltr     ax
        mov     ax, LDT_Sel
        lldt    ax
        JUMP16  DCode0_Sel, 0            ;转移到 0 级程序段
CodeMLen =     $
Start    ENDP
CodeMSeg ENDS
        END     Start

```

4. 实验步骤

- (1) 进入纯 DOS 环境，运行 Tddebug 集成操作软件。
- (2) 按实验要求编写实验程序，实验程序参考流程如图 6-2-2 所示。
- (3) 对实验程序进行编译、链接无误后使用 Pmrun 装入程序。
- (4) 使用 GDT 命令查询系统的 GDT 表，并且查看实验程序中声明的代码段、LDT 表的描述符在 GDT 表中的位置以及对应段的物理地址、段属性、段界限等。
- (5) 按 F7 单步执行程序，在执行到 LTR AX 后，用 TSS 命令查看 TSS 段的内容。
- (6) 在执行到 LLDT AX 后，用 LDT 命令查看 LDT 表的内容。

(7) 单步到 JUMP16 DCODE0_SEL,0 指令后,可以看到控制转移到 DCODE0_SEL 所指的代码段,偏移为 0。

(8) 单步到 RETF 指令后,可以看到控制由 0 级代码段转移到 3 级代码段。

(9) 使用 D 命令查看目的数据区的数据是否正确。

(10) 继续单步到 CALL16 TOCODE0_SEL,0 语句后,控制从 3 级代码段转移到 0 级代码段,入口为 PEND 处(0004:00000017)。

(11) 继续单步到程序结束。

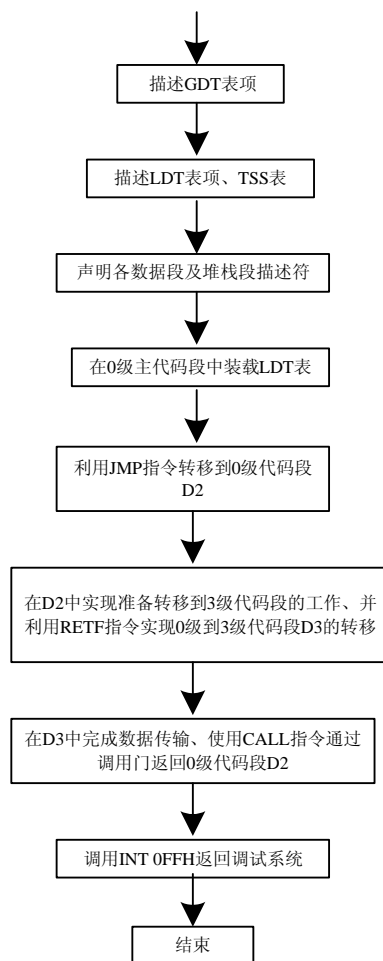


图 6-2-2 任务内有特权级变换的转移实验参考流程图

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-PITD+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。

在主代码段中首先应使用 LTR 指令将监控任务的 TSS 装入 TR 寄存器，但这里并不是任务的切换，所以没有引用监控任务 TSS 中的内容。从监控任务利用 TSS 直接切换到任务 1，可以使用指令“JUMP16 TSS1_SEL,0”，其中 TSS1_SEL 指向任务 1 的 TSS 段。在任务 1 中，使用 JMP 指令，通过任务门 TOTASK0_SEL 可以切换回监控任务。TOTASK0_SEL 指向监控任务 TSS 段的选择子 TSS0_SEL。

3. 实验程序清单 (P3-1.asm)

```
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
ID1         Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 1
Codem       Desc    <CodemLen-1, CodemSeg, , ATCE, , >      ;主程序段描述符
LDTable1    Desc    <LDT1Len-1, LDT1Seg, , ATLD, , >        ;局部描述符表段的描述符
TSSTable0   Desc    <Tss0Len-1, TSS0Seg, , AT386TSS, , >    ;任务 0 状态段 TSS 描述符
TSSTable1   Desc    <Tss1Len-1, TSS1Seg, , AT386TSS, , >    ;任务 1 状态段 TSS 描述符
DStack0     Desc    <DStack0Len-1, DStack0Seg, , ATDW, , >  ;任务 0 的 0 级堆栈段描述符
DData0      Desc    <DData0Len-1, DData0Seg, , ATDW, , >    ;任务 0 的数据段描述符
ID2         Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 2
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                     ;任务状态段选择子
TSS1_Sel    =       TSSTable1-GDT                     ;任务状态段选择子
DStack0_Sel =       DStack0-GDT                       ;0 级堆栈描述符选择子
DData0_Sel  =       DData0-GDT                        ;数据段描述符对应选择子
GDTSeg      ENDS                                     ;全局描述符表段定义结束
;-----
LDT1Seg     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    <DCode1Len-1, DCode1Seg, , ATCE, , >    ;任务 1 的 0 级代码段描述符
DStack1     Desc    <DStack1Len-1, DStack1Seg, , ATDW, , >  ;任务 1 的 0 级堆栈段描述符
DDataS      Desc    <DDataSLen-1, DDataSSeg, , ATDW+DPL3, , > ;源数据段描述符
DData0      Desc    <DData0Len-1, DData0Seg, , ATDW+DPL3, , > ;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                  ;代码段描述符选择子
DStack1_Sel =       DStack1-LDT1+TIL                  ;堆栈描述符选择子
DDataS_Sel  =       DDataS-LDT1+TIL                  ;源数据段描述符选择子
DData0_Sel  =       DData0-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
ID3         Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 3
LDT1Seg     ENDS                                     ;局部描述符表段定义结束
```

```

;-----
TSS0Seg    SEGMENT PARA USE16                ;临时任务的任务状态段 TSS0
TempTask   TSS    <>
           DB      0ffh                      ;I/O 许可位图结束标志
TSS0Len    =      $
TSS0Seg    ENDS
;-----
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-1          ;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-1            ;ESP
           DD      0                        ;EBP
           DD      0                        ;ESI
           DD      0                        ;EDI
           DW      DData0_Sel, 0           ;ES
           DW      DCode1_Sel, 0          ;CS
           DW      DStack1_Sel, 0         ;SS
           DW      DData0_Sel, 0          ;DS
           DW      DData0_Sel, 0          ;FS
           DW      DData0_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      0ffh                  ;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(0)
DStack0Seg ENDS
;-----
DStack1Seg SEGMENT PARA USE16                ;任务 1 的堆栈段
DStack1Len =      512
           DB      DStack1Len DUP(0)
DStack1Seg ENDS
;-----
DData0Seg  SEGMENT PARA USE16
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
DDataSLen  =  $
DDataSSeg  ENDS
;-----
DData0Seg  SEGMENT  PARA USE16          ;目标数据段
DData0Len  =  128
          DB  DData0Len DUP (?)
DData0Seg  ENDS
;-----
CodemSeg  SEGMENT  PARA USE16          ;主程序段
          ASSUME  CS:CodemSeg, DS:DDataSseg
start      PROC
          mov  ax, TSS0_Sel            ;装入任务 0
          ltr  ax
          mov  ax, DData0_Sel
          mov  ds, ax
          mov  es, ax
          mov  fs, ax
          mov  gs, ax
          mov  esp, DStack0Len-1
          mov  ax, DStack0_Sel
          mov  ss, ax
          JUMP16  TSS1_Sel, 0          ;跳转到任务 1 的 0 级代码段
          int  0ffh                    ;返回调试系统
start      ENDP
CodemLen  =  $
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, DData0_Sel
          mov  es, ax                  ;加载目标数据段描述符
          cld
          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       ;切换回监控任务
DCode1Len  =  $
DCode1Seg  ENDS
          END      Start

```

4. 实验步骤

(1) 进入纯 DOS 环境, 运行 Tddebug 集成操作软件。
(2) 按实验要求编写实验程序, 实验程序参考流程如图 6-3-1 所示。

(3) 对实验程序进行编译、链接无误后使用 Pmrun 装入程序。

(4) 使用 GDT 命令查询系统的 GDT 表, 并且查看实验程序中声明的代码段、LDT 表的描述符在 GDT 表中的位置以及对应段的物理地址、段属性、段界限等。

(5) 按 F7 单步执行程序, 在执行到 LTR AX 后, 用 TSS 命令查看 TSS 段的内容。

(6) 执行到 JUMP16 TSS1_SEL,0 语句后, 任务切换到任务 1, 程序在 DCODE1SEG 的第 1 条语句停止。观察各寄存器的内容, 可以看到任务成功的切换。

(7) 继续单步执行到 JUMP16 TOTASK0_SEL,0 语句前, 用 D 命令查看目的数据区数据是否传输正确。

(8) 执行 JUMP16 TOTASK0_SEL,0 语句后, 任务又切换到监控任务, 程序停在 int 0ffh 语句。观察寄存器的内容, 可以看到任务成功的切换。

(9) 继续单步执行直到程序结束。

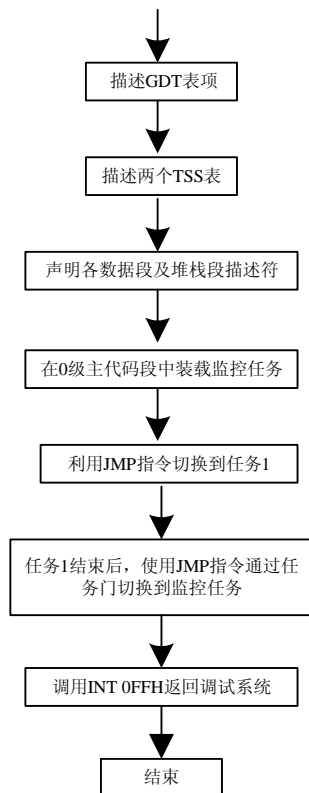


图 6-3-1 任务切换实验 (1) 参考流程图

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。实验由主程序进入, 装入监控任务, 从监控任务通过 TSS 段直接切换到任务 1 的 0 级代码段, 然后从任务 1 通过任务门切换到任务 2 的一个 3 级代码段, 完成一段数据的传输。当任务 2 的工作结束后, 利用 IRETD 指令切换回任务 1。

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 指令将监控任务的 TSS 装入 TR 寄存器。在监控任务中, 使用 Call16 宏直接指向任务 1 的 TSS 表进行任务切换。

在任务 1 中, 使用 Call16 宏通过任务门转移到任务 2 的 3 级程序段。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 1 的任务状态段, 再将任务 2 的状态段中的内容装入 CPU 的各寄存器。任务 2 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不同特权级的切换。

2. 实验程序清单 (P3-2.asm)

```
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
ID1              Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 1
Codem            Desc    <CodemLen-1, CodemSeg, , ATCE, , >        ;主程序段描述符
TSSTable1        Desc    <Tss1Len-1, TSS1Seg, , AT386TSS, , >     ;任务 1 状态段 TSS 描述符
TSSTable2        Desc    <Tss2Len-1, TSS2Seg, , AT386TSS, , >     ;任务 2 状态段 TSS 描述符
TSSTable3        Desc    <Tss3Len-1, TSS3Seg, , AT386TSS, , >     ;任务 3 状态段 TSS 描述符
DCode2           Desc    <DCode2Len-1, DCode2Seg, , ATCE, , >     ;任务 2 的 0 级代码段描述符
DCode3           Desc    <DCode3Len-1, DCode3Seg, , ATCE+DPL3, , > ;任务 3 的 3 级代码段描述符
DStack1          Desc    <DStack1Len-1, DStack1Seg, , ATDW, , >   ;任务 1 的 0 级堆栈段描述符
DStack2          Desc    <DStack2Len-1, DStack2Seg, , ATDW, , >   ;任务 2 的 0 级堆栈段描述符
DStack3          Desc    <DStack3Len-1, DStack3Seg, , ATDW+DPL3, , > ;任务 3 的 3 级堆栈段描述符
DData1           Desc    <DData1Len-1, DData1Seg, , ATDW, , >     ;任务 1 的数据段描述符
DData2           Desc    <DData2Len-1, DData2Seg, , ATDW, , >     ;任务 2 的数据段描述符
DDataS           Desc    <DDataSLen-1, DDataSSeg, , ATDW+DPL3, , > ;源数据段描述符
DData0           Desc    <DData0Len-1, DData0Seg, , ATDW+DPL3, , > ;目的数据段描述符
DStack0          Desc    <DStack0Len-1, DStack0Seg, , ATDW, , >   ;
ToTask3          Gate    <0, TSS3_Sel, , ATTASKGate, >           ;指向任务 0 的调用门描述符
ID2              Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 2
ID3              Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 3
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 级代码段描述符选择子
DCode3_Sel       =      DCode3-GDT+RPL3                           ;3 级代码段描述符选择子
DStack1_Sel      =      DStack1-GDT                                ;0 级堆栈描述符选择子
DStack2_Sel      =      DStack2-GDT                                ;0 级堆栈描述符选择子
DStack3_Sel      =      DStack3-GDT+RPL3                           ;3 级堆栈描述符选择子
DData1_Sel       =      DData1-GDT                                 ;任务 1 数据段描述符对应选择子
DData2_Sel       =      DData2-GDT                                 ;任务 2 数据段描述符对应选择子
DDataS_Sel       =      DDataS-GDT+RPL3                           ;3 级源数据段描述符选择子
DData0_Sel       =      DData0-GDT+RPL3                           ;3 级目的数据段描述符选择子
DStack0_Sel      =      DStack0-GDT                                ;
ToTask3_Sel      =      ToTask3-GDT                                ;任务门描述符选择子
GDTSeg           ENDS                                             ;全局描述符表段定义结束
;-----
TSS1Seg          SEGMENT PARA USE16                                ;临时任务的任务状态段 TSS
TempTask         TSS      <>
                 DB      0ffh                                     ;I/O 许可位图结束标志
TSS1Len          =      $
TSS1Seg          ENDS
;-----
TSS2Seg          SEGMENT PARA USE16                                ;任务状态段 TSS
                 DD      0                                         ;链接字
                 DD      DStack2Len-1                             ;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      0ffffh           ;EAX
        DD      0                ;ECX
        DD      0                ;EDX
        DD      0                ;EBX
        DD      DStack2Len-1     ;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                ;任务状态段 TSS 结束
;-----
TSS3Seg  SEGMENT PARA USE16    ;任务状态段 TSS
        DD      0                ;链接字
        DD      DStack0Len-1     ;0 级堆栈指针
        DD      DStack0_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      0ffffh           ;EAX
        DD      0                ;ECX
        DD      0                ;EDX
        DD      0                ;EBX
        DD      DStack3Len-1     ;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      0ffh                  ;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 堆栈段
DStack1Len =      512
        DB      DStack1Len DUP(0)
DStack1Seg ENDS
;-----
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          ;源数据段
DData1Len =      128
        db      DData1Len DUP(?)
DData1Seg ENDS
;-----
DData2Seg SEGMENT PARA USE16          ;源数据段
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
DDataSLen =      $
DDataSSeg ENDS
;-----
DData0Seg SEGMENT PARA USE16          ;目标数据段
DData0Len =      128
        DB      DData0Len DUP(?)
DData0Seg ENDS
;-----
DStack0seg SEGMENT PARA USE16
DStack0Len =      256
        DB      DStack0Len DUP(?)
DStack0Seg ENDS
;-----
CodemSeg SEGMENT PARA USE16          ;主程序段
        ASSUME  CS:CodemSeg, DS:DDataSseg
start     PROC
        mov     ax, TSS1_Sel          ;装入任务 1

```

```

        ltr ax
        mov ax, DData1_Sel
        mov ds, ax
        mov es, ax
        mov fs, ax
        mov gs, ax
        mov esp, DStack1Len-1
        mov ax, DStack1_Sel
        mov ss, ax
        Call16 TSS2_Sel, 0          ;跳转到任务 2
start    ENDP
CodemLen = $
CodemSeg ENDS
;-----
DCode2Seg SEGMENT PARA USE16      ;任务 2 代码段
        ASSUME CS:DCode2Seg
        mov ax, 1234h
        mov bx, 5678h
        Call16 ToTask3_Sel, 0
        int 0ffh
DCode2Len = $
DCode2Seg ENDS
;-----
DCode3Seg SEGMENT PARA USE16      ;任务 3 代码段
        ASSUME CS:DCode3Seg
        mov ax, DDataS_Sel
        mov ds, ax                ;加载源数据段描述符
        mov ax, DData0_Sel
        mov es, ax                ;加载目的数据段描述符
        cld
        xor si, si
        xor di, di                ;设置指针初值
        mov cx, 8                 ;设置传送长度
ml:      movsb                    ;传送
        loop ml
        iretd                    ;切换回监控任务
DCode3Len = $
DCode3Seg ENDS
        END Start

```

3. 实验步骤

- (1) 进入纯 DOS 环境，运行 Tddebug 集成操作软件。
- (2) 按实验要求编写实验程序，实验程序参考流程如图 6-3-2 所示。
- (3) 对实验程序进行编译、链接无误后使用 Pmrun 装入程序。
- (4) 使用 GDT 命令查询系统的 GDT 表，并且查看实验程序中声明的代码段、LDT 表的描述符在 GDT 表中的位置以及对应段的物理地址、段属性、段界限等。

(5) 按 F7 单步执行程序, 在执行到 LTR AX 后, 用 TSS 命令查看 TSS 段的内容。

(6) 单步执行到 CALL16 TSS2_SEL,0 指令后, 任务切换到任务 1, 程序在 DCODE2SEG 的第 1 条语句停止。观察寄存器的内容, 可以观察到任务成功的切换。

(7) 继续单步执行 CALL16 TOTASK3_SEL,0 指令后, 任务切换到任务 3, 程序在 DCODE3SEG 的第 1 条语句停止。观察寄存器内容, 可以看到任务成功的切换。

(8) 继续单步执行到 IRET 指令时, 用 D 命令查看目的数据区数据是否正确。

(9) 单步执行 IRET 指令, 任务将切换回任务 2, 程序停在 DCODE2SEG 的 INT 0FFh。观察寄存器的内容, 可以看到任务成功的切换。

(10) 继续单步执行直到程序结束。

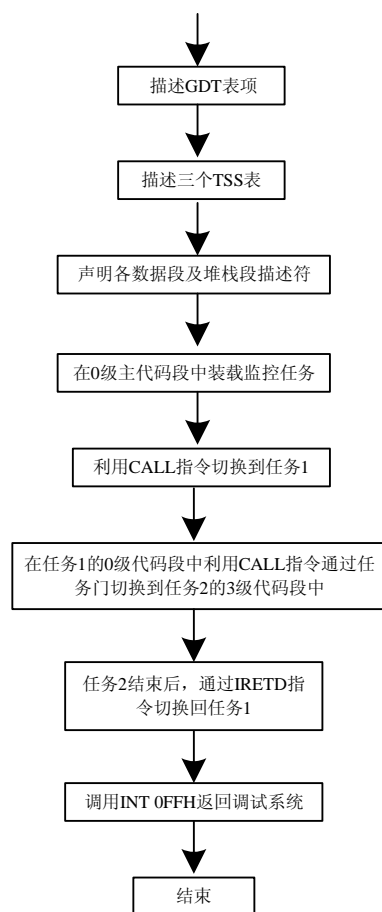


图 6-3-2 任务切换实验 (2) 参考流程图

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-PITD+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 号门描述符为中断门。

由于实验中调用 INT 指令的代码段为 0 级代码段, 而 20H 号异常对应的门描述符是中断门, 所以 INT 调用引起的控制转移是任务内无特权级变换的转移, 所以在访问中断/异常处理的过程中不会引起堆栈切换, 且在控制转移时堆栈中压入了中断/异常处理返回时需要的 CS, EIP, EFLAG。为了简便起见, 实验中可以不使用 TSS 段。

为了在保护模式下将某个中断/异常处理的入口指向用户自己编写的处理过程, 需要修改 IDT 表中相应的门描述符。因为 IDT 表是系统段, 不可以直接读写, 所以必须用一个可读写的存储段描述符来描述 IDT 表。在执行了 INT 指令后, CPU 将进行一系列检测, 并将引起中断/异常的指令的 (或下条指令的) CS, EIP, EFLAG 保存到堆栈中, 然后将 EFLAG 中的 TF 位置 0。到执行 IRETD 返回调用处时, CPU 自动的将堆栈中的 CS, EIP, EFLAG 弹出, 返回到调用处。

3. 实验程序清单 (P4-1.asm)

```
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
ID1         Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 1
Codem       Desc    <CodemLen-1, CodemSeg, , ATCE, , >      ;主程序段描述符
LDTable     Desc    <LDTLen-1, LDTSeg, , ATLD, , >          ;局部描述符表段的描述符
RLDT        Desc    <LDTLen-1, LDTSeg, , ATDW, , >          ;将局部描述符表所在段当成一个可读段
ID2         Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 2
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
RLDT_Sel    =       RLDT-GDT                      ;可读
GDTSeg      ENDS                                  ;全局描述符表段定义结束
;-----
LDTSeg      SEGMENT PARA USE16                      ;局部描述符表数据段(16 位)
LDT         LABEL   BYTE                          ;局部描述符表
DCode       Desc    <DCodeLen-1, DCodeSeg, , ATCE, , >      ;代码段描述符
ICode       Desc    <ICodeLen-1, ICodeSeg, , ATCE, , >      ;异常/中断处理代码段描述符
DStack0     Desc    <DStack0Len-1, DStack0Seg, , ATDW, , >  ;0 级堆栈段描述符
DDataS      Desc    <DDataSLen-1, DDataSSeg, , ATDW, , >    ;源数据段描述符
DData0      Desc    <DData0Len-1, DData0Seg, , ATDW, , >    ;目的数据段描述符
WIDT        Desc    <7ffh, , , ATDW, , >                  ;IDT 表对应的可读写段描述符
IGate       Gate    <, ICode_Sel, , AT386TGate+DPL3, >      ;异常/中断处理程序段的陷阱门描述符
DCode_Sel   =       DCode-LDT+TIL                  ;代码段描述符选择子
ICode_Sel   =       ICode-LDT+TIL                  ;异常/中断处理描述符选择子
DStack0_Sel =       DStack0-LDT+TIL                 ;0 级堆栈描述符选择子
DDataS_Sel  =       DDataS-LDT+TIL                 ;源数据段描述符对应选择子
DData0_Sel  =       DData0-LDT+TIL                 ;目标数据段描述符对应选择子
WIDT_Sel    =       WIDT-LDT+TIL                   ;可读写 IDT 段描述符选择子
IGate_Sel   =       IGate-LDT+TIL                  ;调用门描述符选择子
LDTLen      =       $-LDT
ID3         Desc    <0ffffh, 0ffffh, 0ffh, 0ffh, 0ffh, 0ffh> ;标记描述符 3
```

```

LDTSeg      ENDS                                ;局部描述符表段定义结束
;-----
DStack0Seg  SEGMENT PARA use16                  ;任务 0 级堆栈段
DStack0Len  = 1024
            DB DStack0Len DUP(0)
DStack0Seg  ENDS                                ;任务 0 级堆栈段结束
;-----
DDataSSeg   SEGMENT PARA USE16
DTest       DB 11h, 22h, 33h, 44h, 55h, 66h, 77h, 88h
IDTR_V      PDESC <>
DDataSLen   = $
DDataSSeg   ENDS
;-----
DData0Seg   SEGMENT PARA USE16
DData0Len   = 128
            DB DData0Len DUP(?)
DData0Seg   ENDS
;-----
CodemSeg    SEGMENT PARA USE16
            ASSUME CS:CodemSeg, DS:DDataSseg
start       PROC
            mov ax, LDT_Sel
            lldt ax
            mov ax, DDataS_Sel                  ;填写中断描述符表
            mov ds, ax
            sidt IDTR_V                        ;取得 IDT 所在的物理地址
            mov si, offset IDTR_V
            mov ax, RLDT_Sel
            mov es, ax
            mov ax, WIDT_Sel
            and ax, 0f8h
            mov di, ax
            mov ax, ds:[si+2]                  ;将 IDT 的物理地址装入 WIDT_Sel 对应的描述符
            mov es:[di+2], ax
            mov ah, ds:[si+4]
            mov es:[di+4], ah
            mov ah, ds:[si+5]
            mov es:[di+7], ah
            mov ax, WIDT_Sel
            mov ds, ax
            mov bx, 8
            mov ax, 20h                        ;n 指向中断/异常向量
            mul bx
            mov si, ax
            mov ax, IGate_Sel                  ;将门描述符填入中断描述符表
            and ax, 0f8h
            mov di, ax
            mov eax, es:[di]
            mov ds:[si], eax
            mov eax, es:[di+4]
            mov ds:[si+4], eax

```

```

        JUMP16 DCode_Sel,0          ;跳转到任务 0 级代码段
start    ENDP
CodemLen = $
CodemSeg ENDS
;-----
DCodeSeg SEGMENT PARA USE16        ;任务代码段
        ASSUME CS:DCodeSeg
        mov ax, DDataS_Sel
        mov ds, ax                  ;加载源数据段描述符
        mov ax, DData0_Sel
        mov es, ax                  ;加载目标数据段描述符
        cld
        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
        int 0ffh
DCodeLen = $
DCodeSeg ENDS                      ;任务位代码段结束
;-----
ICodeSeg SEGMENT PARA USE16        ;临时任务的代码段
        ASSUME CS:ICodeSeg
ml:      movsb                       ;传送
        loop ml
        iretd
ICodeLen = $
ICodeSeg ENDS
        END      Start

```

4. 实验步骤

- (1) 进入纯 DOS 环境，运行 Tddebug 集成操作软件。
- (2) 按实验要求编写实验程序，实验程序参考流程如图 6-4-1 所示。
- (3) 对实验程序进行编译、链接无误后使用 Pmrund 装入程序。
- (4) 使用 GDT 命令查询系统的 GDT 表，并且查看实验程序中声明的代码段、LDT 表的描述符在 GDT 表中的位置以及对应段的物理地址、段属性、段界限等。
- (5) 按 F7 单步执行程序，在执行到 LTR AX 后，用 TSS 命令查看 TSS 段的内容。
- (6) 单步到 JUMP16 DCode_Sel,0 语句后，可以看到控制转移到 DCode_Sel 所指的代码段，偏移为 0。
- (7) 单步到 int 20h 前停止，用 IDT 命令查看 IDT 表中 20H 项，可以看到该项指向用户安排的异常处理陷阱门。
- (8) 单步到 int 20h 后，程序进入中断处理过程，继续单步到 IRETD 处停止。使用 D 命令查看数据传输是否正确。
- (9) 继续单步到 IRETD 后，程序返回 DcodeSeg 代码段的 MOV AX,0FFFFH 处，表示中断/异常处理结束。

(10) 继续单步执行直到程序结束。

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

本实验中要求使用 INT 20H 完成一次中断/异常调用，且此处的中断/异常处理以一个独立任务的方式出现。

为了反映通过任务门实现中断响应/异常处理，程序中安排两个任务，一个作为正常工作的任务 (T1)，另一个任务 (T2) 专门进行中断/异常处理，该任务中的中断/异常处理代码段为 0 级。程序中需要完成 IDT 表 20H 号中断/异常处理向量的重定位。

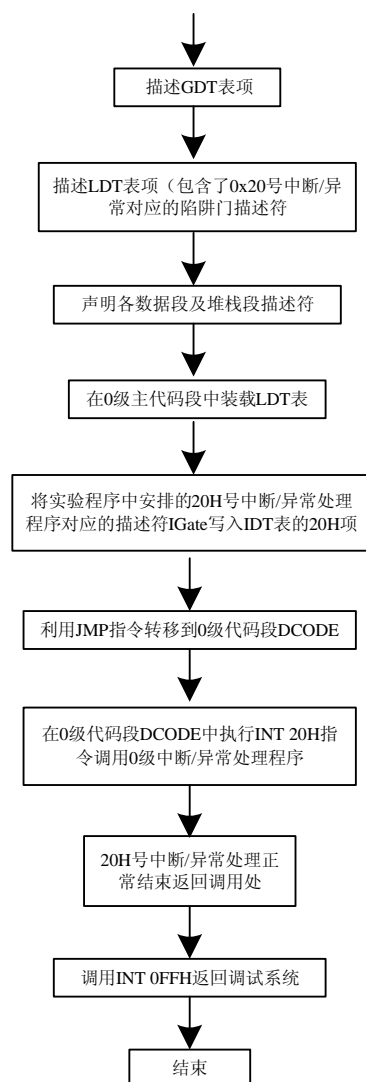


图 6-4-1 用中断门/陷阱门实现中断/异常处理 实验参考流程图

2. 实验程序清单 (P4-2.asm)

```

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
ID1         Desc    <0ffffh,0ffffh,0ffh,0ffh,0ffh,0ffh>
Codem       Desc    <CodemLen-1,CodemSeg,,ATCE,,>;主程序段描述符
LDTable     Desc    <LDTLen-1,LDTSeg,,ATLDT,,> ;局部描述符表段的描述符
ILDTTable   Desc    <ILDLen-1,ILDTSeg,,ATLDT,,> ;中断/异常处理任务局部描述符表段的描述符
TSSTable    Desc    <TssLen-1,TSSSeg,,AT386TSS,,>;任务状态段 TSS 描述符
ITSSTable   Desc    <ITssLen-1,ITSSSeg,,AT386TSS,,>;任务状态段 TSS 描述符
RLDT        Desc    <LDTLen-1,LDTSeg,,ATDW,,> ;将局部描述符表所在段当成一个可读的数据段
ID2         Desc    <0ffffh,0ffffh,0ffh,0ffh,0ffh,0ffh>

```

```

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
ILDT_Sel       =      ILDTable-GDT                  ;中断/异常任务状态段选择子
TSS_Sel        =      TSSTable-GDT                  ;任务状态段选择子
ITSS_Sel       =      ITSSTable-GDT                 ;任务状态段选择子
RLDT_Sel       =      RLDT-GDT                      ;可读写
GDTSeg         ENDS                                  ;全局描述符表段定义结束
;
LDTSeg         SEGMENT PARA USE16                    ;局部描述符表数据段(16 位)
LDT            LABEL    BYTE                          ;局部描述符表
DCode          Desc    <DCodeLen-1,DCodeSeg,,ATCE,,> ;代码段描述符
DStack         Desc    <DStackLen-1,DStackSeg,,ATDW,,> ;0 级堆栈段描述符
WTSS           Desc    <TssLen-1,TSSSeg,,ATDW,,>      ;把 TSS 当成可读写的 数据段
WIDT           Desc    <7ffh,,ATDW,,>                ;IDT 表对应的可读写段描述符
IGate          Gate    <,ITSS_Sel,,ATTaskGate,>       ;异常/中断处理程序段的陷阱门描述符
DCode_Sel      =      DCode-LDT+TIL                  ;代码段描述符选择子
DStack_Sel     =      DStack-LDT+TIL                 ;0 级堆栈描述符选择子
WTSS_Sel       =      WTSS-LDT+TIL                   ;可读写 TSS 段对应选择子
WIDT_Sel       =      WIDT-LDT+TIL                   ;可读写 IDT 段选择子
IGate_Sel      =      IGate-LDT+TIL                   ;调用门描述符对应选择子
LDTLen         =      $-LDT
LDTSeg         ENDS                                  ;局部描述符表段定义结束
;
ILDTSeg        SEGMENT PARA USE16                    ;局部描述符表数据段(16 位)
ILDT           LABEL    BYTE                          ;局部描述符表
IStack         Desc    <IStackLen-1,IStackSeg,,ATDW,,> ;0 级堆栈段描述符
ICode          Desc    <ICodeLen-1,ICodeSeg,,ATCE,,>   ;异常/中断处理代码段描述符
IDDataS        Desc    <DDataSLen-1,DDataSSeg,,ATDW,,> ;源数据段描述符
IDData0        Desc    <DData0Len-1,DData0Seg,,ATDW,,> ;目的数据段描述符
IData          Desc    <IDataLen-1,IDataSeg,,ATDW,,>   ;数据段描述符
IStack_Sel     =      IStack-ILDT+TIL                 ;0 级堆栈描述符选择子
ICode_Sel      =      ICode-ILDT+TIL                  ;异常/中断处理描述符选择子
IDDataS_Sel    =      IDDataS-ILDT+TIL                ;源数据段描述符对应选择子
IDData0_Sel    =      IDData0-ILDT+TIL                ;目标数据段描述符对应选择子
IData_Sel      =      IData-ILDT+TIL                  ;数据段描述符对应选择子
ILDTLen        =      $-ILDT
ID3            Desc    <0ffffh,0ffffh,0ffh,0ffh,0ffh,0ffh>
ILDTSeg        ENDS                                  ;局部描述符表段定义结束
;
TSSSeg         SEGMENT PARA USE16                    ;临时任务的任务状态段 TSS
TempTask       TSS    <>
               DB      0ffh                          ;I/O 许可位图结束标志
TSSLen         =      $
TSSSeg         ENDS
;
DStackSeg      SEGMENT PARA use16                    ;任务 0 级堆栈段(16 位段)
DStackLen      =      1024
               DB      DStackLen DUP(0)
DStackSeg      ENDS                                  ;任务 0 级堆栈段结束
;
DDataSSeg      SEGMENT PARA USE16
DTest          DB      11h,22h,33h,44h,55h,66h,77h,88h
IDTR_V         PDESC    <>
DDataSLen      =      $
DDataSSeg      ENDS

```

```

;-----
DData0Seg    SEGMENT PARA USE16
DData0Len    =      128
              DB    DData0Len DUP(?)
DData0Seg    ENDS
;-----
ITSSSeg      SEGMENT PARA USE16                ;任务状态段 TSS
              DD      0                        ;链接字
              DD      IStackLen-1              ;0 级堆栈指针
              DD      IStack_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      1234h                    ;EBX
              DD      IStackLen-1              ;ESP
              DD      0                        ;EBP
              DD      0                        ;ESI
              DD      0                        ;EDI
              DD      0                        ;ES
              DD      ICode_Sel                ;CS
              DD      IStack_Sel              ;SS
              DD      0                        ;DS
              DD      IData_Sel                ;FS
              DD      IData_Sel                ;GS
              DD      ILDT_Sel                ;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
ITSSLen      =      $
ITSSSeg      ENDS                                ;任务状态段 TSS 结束
;-----
IStackSeg    SEGMENT PARA use16                ;任务堆栈段
IStackLen    =      1024
              DB      IStackLen DUP(0)
IStackSeg    ENDS                                ;任务堆栈段结束
;-----
IDataSeg     SEGMENT PARA USE16
IDataLen     =      128
              DB      IDataLen DUP(?)
IDataSeg     ENDS
;-----
CodemSeg     SEGMENT PARA USE16
              ASSUME  CS:CodemSeg, DS:DDataSseg
start        PROC
              mov  ax, LDT_Sel
              lldt ax
              mov  ax, WTSS_Sel                ;把 LDT_Sel 填入 TSS 的 LDTR 项
              mov  ds, ax

```



```

        mov si, 60h
        mov ax, LDT_Sel
        mov ds:[si], ax
        mov ax, TSS_Sel
        ltr ax
        mov ax, IDDataS_Sel           ;填写中断描述符表
        mov ds, ax
        sidt IDTR_V                   ;取得 IDT 所在的物理地址
        mov si, offset IDTR_V
        mov ax, RLDT_Sel
        mov es, ax
        mov ax, WIDT_Sel
        and ax, 0f8h
        mov di, ax
        mov ax, ds:[si+2]             ;将 IDT 的物理地址装入 WIDT_Sel 对应的描述符
        mov es:[di+2], ax
        mov ah, ds:[si+4]
        mov es:[di+4], ah
        mov ah, ds:[si+5]
        mov es:[di+7], ah
        mov ax, WIDT_Sel
        mov ds, ax
        mov bx, 8
        mov ax, 20h                   ;n 指向中断/异常向量
        mul bx
        mov si, ax
        mov ax, IGate_Sel             ;将门描述符填入中断描述符表
        and ax, 0f8h
        mov di, ax
        mov eax, es:[di]
        mov ds:[si], eax
        mov eax, es:[di+4]
        mov ds:[si+4], eax
        JUMP16 DCode_Sel, 0           ;跳转到任务 0 级代码段
start    ENDP
CodemLen = $
CodemSeg ENDS
;-----
DCodeSeg    SEGMENT PARA USE16       ;任务代码段
            ASSUME CS:DCodeSeg
            int 20h                   ;调用 20h 中断/异常处理程序来传送数据
            mov ax, ILdt_Sel
            lldt ax
            int 0ffh
DCodeLen = $
DCodeSeg    ENDS                     ;任务代码段结束
;-----
ICodeSeg    SEGMENT PARA USE16       ;临时任务的代码段
            ASSUME CS:ICodeSeg
            mov ax, IDDataS_Sel
            mov ds, ax                 ;加载源数据段描述符
            mov ax, IDData0_Sel
            mov es, ax                 ;加载目标数据段描述符
            cld
            xor si, si

```

```

        xor    di, di                ;设置指针初值
        mov    cx, 8                ;设置传送长度
ml:     movsb                        ;传送
        loop   ml
        iretd
ICodeLen =    $
ICodeSeg     ENDS
END          Start

```

3. 实验步骤

(1) 进入纯 DOS 环境,运行 Tddebug 集成操作软件。

(2) 按实验要求编写实验程序,实验程序参考流程图 6-4-2 所示。

(3) 对实验程序进行编译、链接无误后使用 Pmrund 装入程序。

(4) 使用 GDT 命令查询系统的 GDT 表,并且查看实验程序中声明的代码段、LDT 表的描述符在 GDT 表中的位置以及对对应段的物理地址、段属性、段界限等。

(5) 按 F7 单步执行程序,在执行到 LLDT AX 后,用 LDT 命令查看 LDT 表的内容。

(6) 单步到 JUMP16 DCODE_SEL,0 语句后,可以看到控制转移到 DCODE_SEL 所指的代码段,偏移为 0。

(7) 单步到 int 20h 前停止,用 IDT 命令查看 IDT 表中 20H 项,可以看到该项指向用户安排的异常处理任务门。

(8) 单步到 int 20h 后,程序进入中断处理过程,继续单步到 IRETD 处停止。使用 D 命令查看数据传输是否正确。

(9) 继续单步到 IRETD 后,程序停在 mov ax,0fffh 处,表示中断/异常处理结束。

(10) 继续单步执行直到程序结束。

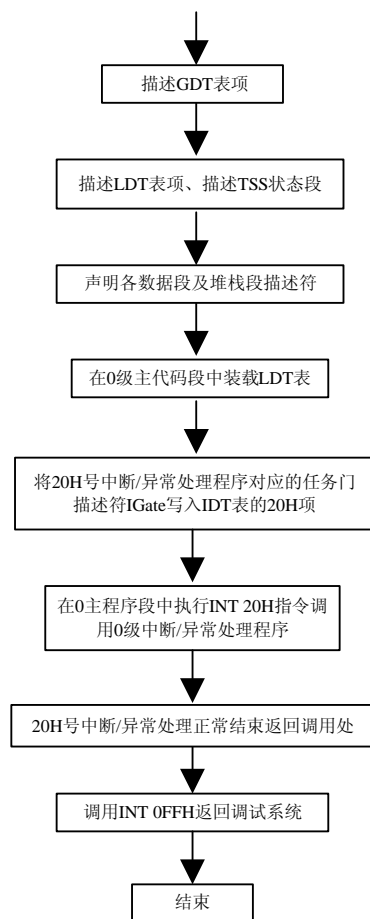


图 6-4-2 调用任务门实现中断/异常处理实验参考流程图

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：

$$\text{LIMIT} = \text{limit} * 4\text{K} + 0\text{FFFFH} = (\text{Limit} \text{ SHL } 12) + 0\text{FFFFH}$$

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 7-1-1 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 7-1-1 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 将该位置为 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 7-1-2:

(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 7-1-2 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 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:

DESCRIPTAB	LABEL	BYTE
DESC1	DESC	<1234H,5678H,34H,92H,,>
DESC2	DESC	<1234H,5678H,34H,93H,,>
DESC3	DESC	<5678H,1234H,56H,98H,,>
DESC4	DESC	<5678H,1234H,56H,99H,,>
DESC5	DESC	<0FFFFH,,10H,16H,,>
DESC6	DESC	<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:

$4GB \times 8192 \times 2 = 64MMB = 64TB$

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 7-1-1 所示。

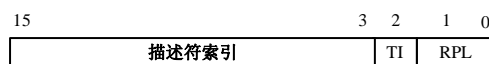


图 7-1-1 选择子格式

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 7-1-2 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 7-1-2 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 7-2-1 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。

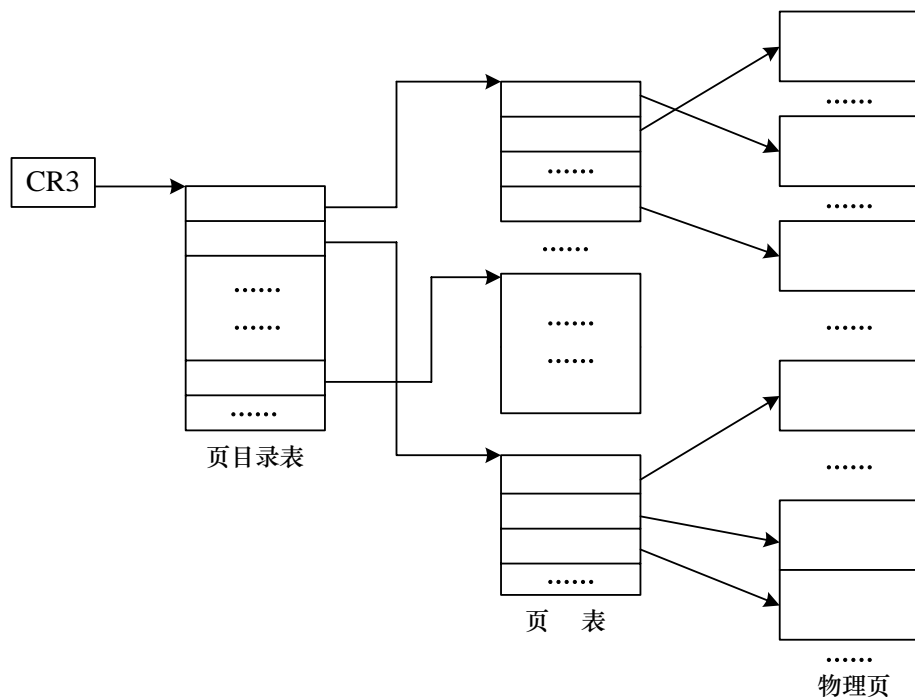


图 7-2-1 页映射表结构

2. 表项格式

页目录表和页表中的表项都采用如图 7-2-2 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。

31	12	11	10	9	8	7	6	5	4	3	2	1	0
物理页码			AVL	0	0	D	A	0	0		U / S	R / W	P

图 7-2-2 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 0FFH 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 7-2-1 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 7-2-1 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 7-2-2 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 7-2-2 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(OEH)。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 7-2-3 所示。

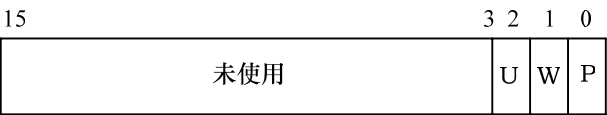


图 7-2-3 页故障出错格式

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 保护模式下的存储器扩展实验

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-PITD+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。

实验系统的存储器空间共有 16MB，偏移为 000000H~FFFFFFH。起始地址 MY0 由 PC 机系统动态分配（如 D9000000H~D97FFFFFFH），可以通过运行 CHECK 程序查看到，如图 8-1 所示。

```
***** Configuration Information *****  
  
I/O Address: IOY0: 9C00 -- 9C3F   (64B)  
              IOY1: 9C40 -- 9C7F   (64B)  
              IOY2: 9C80 -- 9CBF   (64B)  
              IOY3: 9CC0 -- 9CFF   (64B)  
  
MEM Address:  MY0: D9000000 -- D97FFFFFF (8MB)  
              MY1: D9800000 -- D9FFFFFFF (8MB)  
  
INTR Number: 11   INTR_IVADD : 01CCH  
                  INTR_OCW1  : A1H  
                  INTR_OCW2  : A0H  
                  INTR_IM    : F7H
```

图 8-1 CHECK 程序显示

分配的存储器寻址空间远超出 1MB，在实地址模式下无法操作，需要在 CPU 的保护模式下操作。所以实验程序要按照保护模式程序结构编写，在保护模式调试集成软件 Tddebug 上运

行。

8.1.4 实验步骤

(1) 确认从 PC 机引出的扁平电缆已经连接在实验平台上。

(2) 启动纯 DOS 环境, 进入 Tddebug 软件所在的安装目录。执行 CHECK 程序, 查看存储器空间始地址并记录。

(3) 运行 Tddebug 集成操作软件。操作 Alt+E 进入程序编辑环境。利用查出的地址编写 16 位存储器操作的程序, 然后编译链接。

(4) 参考图 8-2 所示连接实验线路。

(5) 操作 Alt+P 进入保护模式调试环境, 根据程序中的设计用 D 命令分别查看源数据区和实验单元存储器中的数据。

(6) 按 F9 运行程序或按 F7 单步执行到结束, 再用 D 命令查看存储器单元中的数据, 观察数据写入是否正确。

(7) 将程序改为非规则字写入操作, 再调试程序, 观察存储器中数据的变化, 分析写入字的排列规则以及总线操作时序的原理。

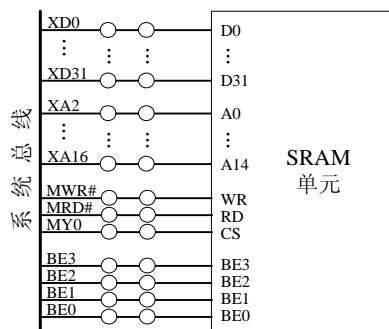


图 8-2 存储器扩展实验接线图

实验程序清单 (PMEM-32.asm)

```

;*****根据 CHECK 配置信息修改下列符号值*****
MY0_H      EQU    0D9H          ;片选 MY0 起始地址的最高位字节
MY0_M      EQU    00H          ;片选 MY0 起始地址的次高位字节
MY0_L      EQU    0000H        ;片选 MY0 起始地址的低两位字节
;*****

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

Desc        STRUC
LimitL      DW          0000H ;段界限 (BIT0-15)
BaseL       DW          0000H ;段基地址 (BIT0-15)
BaseM       DB          00H   ;段基地址 (BIT16-23)
Attributes  DB          00H   ;段属性

```

```

LimitH      DB      00H ;段界限(BIT16-19) (含段属性的高4位)
BaseH       DB      00H ;段基址(BIT24-31)
Desc        ENDS

```

```

DSEG        SEGMENT USE16
GDT         LABEL   BYTE
ID1         DESC    <OFFFFH, OFFFFH, OFFH, OFFH, OFFH, OFFH>
SCODE       DESC    <OFFFFH, CSEG, , ATCE, , >
DATAS       DESC    <D1LEN, DSEG1, , ATDW, , > ;源数据段描述符
DATAD       DESC    <2000H, MYO_L, MYO_M, ATDW, , MYO_H> ;目的数据段描述符
GDTLEN      =      $-GDT
SCODE_SEL   =      SCODE-GDT
DATAS_SEL   =      DATAS-GDT
DATAD_SEL   =      DATAD-GDT
ID2         DESC    <OFFFFH, OFFFFH, OFFH, OFFH, OFFH, OFFH >
ID3         DESC    <OFFFFH, OFFFFH, OFFH, OFFH, OFFH, OFFH >
DSEG        ENDS

```

```

DSEG1       SEGMENT USE16
TDATA       DD      11111111H, 22222222H, 33333333H, 44444444H ;定义原数据段数据
            DD      55555555H, 66666666H, 77777777H, 88888888H
            DD      11111111H, 22222222H, 33333333H, 44444444H ;定义原数据段数据
            DD      55555555H, 66666666H, 77777777H, 88888888H
            DD      11111111H, 22222222H, 33333333H, 44444444H ;定义原数据段数据
            DD      55555555H, 66666666H, 77777777H, 88888888H
D1LEN       =      $-1
DSEG1       ENDS

```

```

CSEG SEGMENT USE16
      ASSUME CS:CSEG

```

```

START      PROC
            MOV AX, DATAS_SEL ;装入数据段地址
            MOV DS, AX
            MOV AX, DATAD_SEL
            MOV ES, AX
            XOR SI, SI
            XOR DI, DI
            MOV CX, 18H
LOOP1:     MOV EAX, DS:[SI] ;将源数据段数据传输到目的数据段
            MOV ES:[DI], EAX
            ADD SI, 4
            ADD DI, 4
            LOOP LOOP1
            INT 0FFH
START      ENDP
CLEN       =      $-1
CSEG       ENDS
END        START

```

附录 A Tddebug 集成操作软件使用说明

1. 软件运行环境

操作系统：纯 DOS 系统环境

CPU：80x86 及兼容 CPU

内存：32MB 以上

显示器：标准 VGA

硬盘：64MB 以上，FAT32 格式

该集成操作软件要求微机系统具有实 DOS 操作系统方可运行，其原因是：集成操作软件要将微处理器从实地址模式下切换到保护模式，完全操作、调试 CPU 的运行。若微机系统已启动在如 Windows 之类的保护模式系统下，则无法运行。另外，存储器操作和使用到 PCI 中断的接口实验也需在该软件下完成。

在 Tdpit 软件安装后，Tddebug 也已经拷贝到安装目录下。默认的目录为：C:\TangDu\PitDP\Tddebug，在该目录下，存放着 Tddebug 软件及所有例程（包括保护模式原理实验程序及保护模式存储器扩展实验程序）。

2. 主菜单说明

Tddebug 集成操作软件集编辑、编译、链接、调试等多个功能于一体，为用户提供了一个学习 32 位微机保护模式汇编程序设计的实验平台。该软件主界面包含了 6 个菜单，分别为：Edit，Compile，Pmrun，Rmrun，Help 和 Quit。部分菜单还包含了子菜单。下面将对每个菜单可以实现的功能作一一介绍。

Edit		编辑源文件
Compile-----	Compile	编译源文件
-----	Link	连接目标文件
-----	Build All	编译和连接
Pmrun		进入保护模式调试状态
Rmrun -----	Run	运行实模式程序
-----	Debug	进入实模式调试状态
Help		版本信息
Quit		退出 Tddebug

(1) 选择主菜单：ALT+KEY (E,C,P,R,H,Q)

ALT+E	选择 Edit 菜单
ALT+C	选择 Compile 菜单
ALT+P	选择 Pmrun 菜单
ALT+R	选择 Rmrun 菜单

ALT+H 选择 Help 菜单

ALT+Q 选择 Quit 菜单

(2) 菜单切换

可以通过小键盘上的左右键或直接使用快捷键在主菜单之间进行切换。使用小键盘上的上下键可以选择子菜单中的菜单项。

(3) 执行菜单项

选中要执行的菜单项，键入 Enter 键即可。

(4) 说明

在执行编辑、编译、链接、运行、调试前，系统会弹出对话框，要求用户键入操作的文件名称。结束键入则以 Enter 键作为结尾，取消操作可以按 ESC。

3. 保护模式调试窗口说明

在 Tddebug 主菜单中执行 ALT+P (Pmrun)，就进入了保护模式调试窗口。保护模式原理实验均在这个环境中完成。

(1) 窗口划分

保护模式调试窗口共分为 4 个区域 Data,Code,Command,Register，分别指示数据区、代码区、命令区和寄存器显示区，如附图 A-1 所示。默认状态下，光标停留在 Command 窗口，用户可以在此键入操作命令。通过 TAB 键可以在四个窗口间进行切换。当切换到 Data 窗口中时，可以通过上下键浏览存储段中的内容。当切换到 Code 窗口中时，可以通过上下键反汇编存储段中的程序。

(2) 快捷键

F1 弹出帮助对话框

F7 单步执行程序

F8 单句执行程序

F9 运行程序



附图 A-1 保护模式调试界面窗口划分图

(3) 命令说明

Tddebug 调试环境提供的命令如附表 A-1 所示。系统除支持保护模式下汇编语言程序的调

试外，还支持 32 位寄存器显示等。下面对每个命令给出相应的命令格式及说明。

附表 A-1 Tddebug 调试命令表

命令内容	格式	命令说明
Load	l filename	装载可执行程序
Reload	Reload	重新装载当前调试程序
Trace	t [[seg:]offset]	单步执行一条指令
Step	p [=seg:]offset]	单句执行一条指令
Go	g=[seg:]offset	执行程序
Go break	gb=[seg:]offset]	断点执行程序
Set breakpoints	B	设置断点
List breakpoints	Bl	列断点表
Clear breakpoints	bc number(0,1,2,3)	清除断点
Unassemble	U[[seg:]offset]	反汇编
Dump	D[[seg:]offset]	显示存储单元内容
Enter	e[seg:]offset	修改存储单元
Register	r[regname]	显示/修改寄存器内容
Peek	peek type(b,w,d) phys_add	从物理地址取数据 (字节、字、双字)
Poke	Poke type(b,w,d) phys_add value	向物理地址写数据 (字节、字、双字)
Cpu	Cpu	显示系统寄存器
Gdt	Gdt	显示全局描述符表
Idt	Idt	显示中断描述符表
Ldt	Ldt	显示局部描述符表
Tss	Tss	显示任务状态段
Quit	Q	退出调试状态

命令名: Load

格式: L filename(full path)

说明: load 命令用来装入实验程序。系统按照实验程序内部各个段的排列顺序，将程序装入从 120000h 的内存地址开始的一个连续空间内。程序装入后，系统将为用户分配起始运行时的 ds,fs,gs,es (对应实验程序装入段选择子)，系统将除 esp 以外的通用寄存器清零，并为用户分配一个 0 级的 1024 字节大的堆栈，置 cs, eip 及标志寄存器。

Eg: l d:\masm\ldt.exe

屏幕显示: loading.....please wait for a minute!

Load OK!

命令名: Reload

格式: Reload

说明: 清零命令，将通用寄存器，段寄存器，标志寄存器设置成 load 后的状态，并将

用户设置断点清空，调用 reload 指令后，各窗口显示内容将刷新到初始状态。

Eg: reload

命令名: Trace

格 式: T

T offset

T seg:offset

说 明: 单步命令有三种格式:

- (1) 单独的 t 命令，系统会从当前 cs, eip 所指地址开始执行程序。
- (2) 系统从当前 cs 和 offset 所指地址开始执行。
- (3) 系统从 seg: offset 指示的地址开始执行一条指令。

Eg: T / T 0000 / T 50:0000

正常结束: 显示通用寄存器，段寄存器，标志寄存器内容，显示下一条指令。

- 错误提示:**
- (1) Selector Error!选择子错误。
 - (2) Format Error!地址格式错误。
 - (3) Limit Error!偏移超出段限。
 - (4) 其他错误提示，可能对应了一个异常提示信息。

命令名: Step

格 式: P

P=offset

P=seg:offset

说 明: 执行过程命令，从给定地址处开始执行一个过程，如未给定地址，将从系统当前指示地址处开始执行若地址仅包含偏移部分，则取系统当前 cs 作为地址的选择子，执行一个过程。

Eg: P

P=50:0000

- 错误提示:**
- (1) Selector Error!选择子错误。
 - (2) Format Error!地址格式错误。
 - (3) Limit Error!偏移超出段限。
 - (4) 其他错误提示，可能对应了一个异常提示信息。

命令名: Go

格 式: G=offset

G=seg:offset

说 明: go 命令从给定地址处开始执行指令，地址应当包含段选择子和偏移两部分，若给定地址仅包含 offset，则取当前指令所指的 cs 作为地址的段选择子。(忽略断点)

Eg: g=50:0000

- 错误提示:**
- (1) Selector Error!选择子错误。
 - (2) Format Error!地址格式错误。
 - (3) Limit Error!偏移超出段限。
 - (4) 其他错误提示，可能对应了一个异常提示信息。

命令名: Go break

格 式: GB

GB=offset

GB=seg:offset

说 明: GB 命令从给定地址处开始执行指令, 给定地址应当包含选择子和偏移两部分, 若键入的命令没有带地址参数, 则从系统当前 cs,eip 处开始执行指令, 若地址仅包含偏移部分, 则取系统当前 cs 作为地址的选择子, 执行过程中如果遇到断点地址, 则程序的执行将会停止。

Eg: GB

GB=50:0000

错误提示:

- (1) Selector Error!选择子错误。
- (2) Format Error!地址格式错误。
- (3) Limit Error!偏移超出段限。
- (4) 其他错误提示, 可能对应了一个异常提示信息。

命令名: Set breakpoints

格 式: B

0: [seg:]offset

1: [seg:]offset

2: [seg:]offset

3: [seg:]offset

说 明: 系统最多可支持 4 个断点的调试, 当用户键入了命令 b 后, 系统会要求用户分别键入 4 个断点的地址, 并判断用户键入地址格式的正确性, 如果地址正确, 系统将会设置对应编号的调试寄存器, 并允许 cpu 监视该编号的断点。目前系统仅支持可执行断点的设置。

Eg: B

0: 50: 0000

1:

备 注: 调试系统在设置断点时使用的是 cpu 提供的调试寄存器, 所以目前最多支持 4 个断点, 而且目前仅支持代码断点, 而且此处 b 命令并未对地址的有效性进行判定, 如果用户设置的断点指示的是数据断点, 则无法到达断点。而且用户如果将断点设置在系统程序段中, 可能引起系统崩溃!

错误提示:

- (1) Format Error!地址格式错误。
- (2) Address Error!地址不合法。

命令名: List breakpoints

格 式: bl

说 明: 如果用户在系统中设置了断点, 该命令会将用户设置的所有断点号及断点地址显示出来。如果没有设置断点, 系统会给予提示。

命令名: Clear breakpoints

格 式: bc

bc n (n 为 0, 1, 2, 3 中的任意一个)

说 明: 使用 bc 命令可以清除系统内的所有断点, 也可清除指定断点。

命 令 名: **Uassemble**

格 式: U

U [seg:]offset

说 明: 反汇编命令。有两种格式, 第一种, 单独的 U 命令, 系统将从当前 CS, EIP 指向的地址开始读取最多 128 个字节长的代码 (读取字节数视 CS 的段限和 EIP 所指位置而定), 并显示。第二种, 用户给定读取的起始地址, 系统对地址进行合法性判别, 若是合法的, 则从给定地址处读取最多 128 个字节的内容并显示。

Eg: U

U 0000

U 10:0000

错误提示: (1) Selector Error!选择子错误。
(2) Format Error!地址格式错误。
(3) Limit Error!偏移超出段限。

命 令 名: **Enter**

格 式: E [seg:]offset

E [seg:]offset 清单

说 明: 修改存储单元命令, 两种格式。第一种, 仅输入要修改的存储单元地址, 系统首先分析地址的正确性, 正确则从内存中取得该存储单元的内容并显示, 等待用户的修改, 在对修改内容进行合法性判别后, 回写到该单元。第二种, 输入要修改的一组存储单元起始地址和修改清单, 系统将对地址及清单进行合法性判别, 通过判别之后, 把清单中列出的值依次写入从起始地址开始的最多 16 个单元中。

Eg: 1. E 10:20 显示: 0010:00000020 ff.

2. E 10:20 ff 12 44 23

错误提示: (1) Selector Error!选择子错误。
(2) Format Error!地址格式错误。
(3) Limit Error!偏移超出段限。

命 令 名: **Dump**

格 式: D

D [seg:]offset

D [seg:] offset1 offset2

说 明: 显示存储单元命令, 有三种格式。第一种, 单独的 d 命令, 系统将从当前 ds,esi 所指的单元中读取最多 128 个字节的内容 (读取字节数视 ds 的段限和 esi 所指位置而定), 并显示。第二种, 用户给定读取的起始地址, 系统对地址进行合法性判别, 合法的, 则从给定地址处读取最多 128 个字节的内容并显示。第三种, 用户给定读取的起始和终止地址, 系统对地址进行合法性判别, 合法, 则从内存读出给定范围内的数据并显示。

Eg: D

- D 0000
D 10:0000
- 显示: 0010:00000000 ff 11 ff 22 ff ff ff ff-00 00 00 00 00 00 00 00
- 错误提示: (1) Selector Error!选择子错误。
(2) Format Error!地址格式错误。
(3) Limit Error!偏移超出段限。
- 命令名: Register**
格式: R
R regname
- 说明: 显示寄存器命令。显示的寄存器包括 6 个段寄存器, 8 个 32 位通用寄存器和标志寄存器, 单独的 r 命令将显示所有上述的寄存器, r 命令后如果带了寄存器的名称, 将只显示指定寄存器的内容, 并允许用户修改该寄存器的值。
- 命令名: Peek**
格式: peek type (b,w,d) physical_address
- 说明: 从 physical_address 指定的物理地址取一个字节、字或者双字的内容, 并显示, type 标识了取的长度, b 为字节, w 为字, d 为双字。
- 命令名: Poke**
格式: poke type (b,w,d) physical_address value
- 说明: 向 physical_address 指定的物理地址写一个字节、字或者双字的内容, type 标识了取的长度, b 为字节, w 为字, d 为双字; value 标识了写的内容。
- 命令名: Cpu**
格式: CPU
- 说明: 显示调试寄存器 DR0~3, TR6~7 的内容, CR0 的内容, 以后可扩展显示其余的 CR 寄存器, 和测试寄存器的系统寄存器内容。
- 命令名: Gdt**
格式: GDT
- 说明: 显示系统描述符表命令。
- 命令名: Idt**
格式: IDT
- 说明: 显示中断描述符表命令。
- 命令名: Ldt**
格式: LDT
- 说明: 显示局部描述符表命令, 如果实验程序中没有使用局部描述符表, 该命令无效。
- 命令名: Tss**
格式: TSS
- 说明: 显示任务状态段命令, 如果实验程序中没有使用任务, 该命令无效。
- 命令名: Quit**
格式: Q
- 说明: 退出调试命令。

附录 B Tdpit 集成操作软件使用说明

1. 软件运行环境

操作系统: Windows2000/XP/Windows7

CPU: 1GHz 以上, 具有多线程处理能力

内存: 512MB 以上

显示器: 标准 VGA

安装空间: 50MB

软件安装完成后默认的目录结构如下。

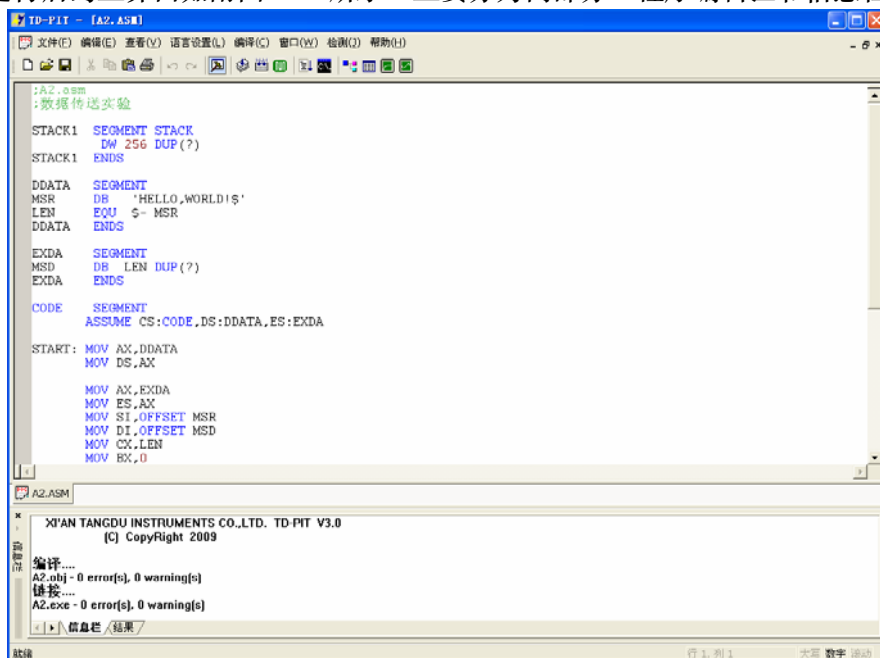
C:\Tangdu\PITDP\Asm 汇编参考实验程序

C:\Tangdu\PITDP\C C 语言参考实验程序

2. 软件概述

Tdpit 集成操作软件是在 Windows2000/XP/Windows7 系列操作系统下进行汇编和 C 语言接口实验的集成编辑调试环境。它允许用户在该环境下编辑、编译、运行及源语言级调试汇编和 C 语言程序。并且提供了详细的软件操作及实验操作帮助系统, 用户可以在实验过程中随时查看实验操作步骤、方法等帮助说明。该环境还集成大量的 Windows 接口应用实验例程, 使用户可以很方便地完成 Windows 下接口应用实验。

软件运行后的主界面如附图 B-1 所示。主要分为两部分: 程序编辑区和信息栏。



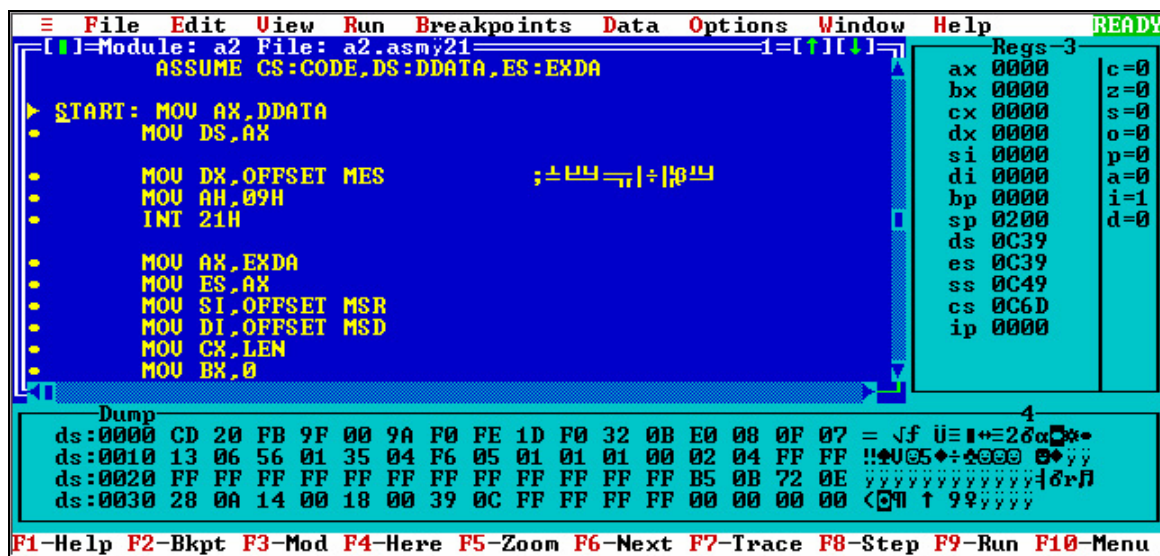
附图 B-1 Tdpit 集成操作软件运行界面

(1) 程序编辑区

位于界面上部区域，用户可在程序编辑区用“新建”命令打开一个新文档或用“打开”命令打开一个已存在的文档，在文档中用户可编辑程序。用户可在程序编辑区打开多个文档，点击文档标签可激活任一文档。编译、链接、加载以及调试命令只针对当前活动文档。用户调试程序时，将切换到调试界面中，调试界面如附图 B-2 所示。

(2) 信息栏

位于界面下部，主要显示编译和链接的结果，如果编译时有错误或警告，双击错误或警告信息，错误标识符会指示到相应的有错误或警告的行。



附图 B-2 调试界面窗口

3. 软件操作说明

1. 菜单功能介绍

(1) 文件菜单项

文件菜单如附图 B-3 所示，提供了以下命令。

新建 (N)：用此命令在 Tdtpit 中建立一个新文档。

打开 (O)：用此命令在窗口中打开一个现存的文档。您可同时打开多个文档，点击某文档的标签可激活此文档。您可用窗口菜单在多个打开的文档中切换。

关闭 (C)：用此命令来关闭当前活动文档。Tdtpit 会建议您在关闭文档之前保存对您的文档所做的改动。如果您没有保存而关闭了一个文档，您将会失去自从您最后一次保存以来所做的所有改动。在关闭一无标题的文档之前，Tdtpit 会显示另存为对话框，建议您命名和保存文档。

保存 (S)：用此命令将当前活动文档保存到它的当前的文件名和目录下。当您第一次保存文档时，Tdtpit 显示另存为对话框以便您命名您的文档。如果在保存之前，您想改变当前文档的文件名和目录，您可选用另存为命令。

另存为 (A) …：用此命令来保存并命名活动文档。Tdtpit 会显示另存为对话框以便您命名

您的文档。



附图 B-3 文件菜单项

打印 (P) …: 用此命令来打印一个文档。在此命令提供的打印对话框中, 您可以指明要打印的页数范围、副本数、目标打印机, 以及其它打印机设置选项。

打印预览 (V): 用此命令按要打印的格式显示活动文档。当您选择此命令时, 主窗口就会被一个打印预览窗口所取代。这个窗口可以按它们被打印时的格式显示一页或两页。打印预览工具栏提供选项使您可选择一次查看一页或两页, 在文档中前后移动, 放大和缩小页面, 以及开始一个打印作业。

打印设置 (R) …: 用此命令来选择一台打印机和一个打印机连接。在此命令提供的打印设置对话框中, 您可以指定打印机及其连接。

最近使用文件: 您可以通过此列表, 直接打开最近打开过的文件, 共四个。

退出 (X): 用此命令来结束您 Tdpit 的运行阶段。您也可使用在应用程序控制菜单上的关闭命令。Tdpit 会提示您保存尚未保存的改动。

(2) 编辑菜单项

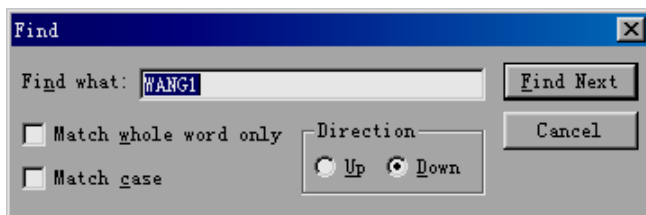
编辑菜单如附图 B-4 所示, 提供了以下命令。



附图 B-4 编辑菜单项

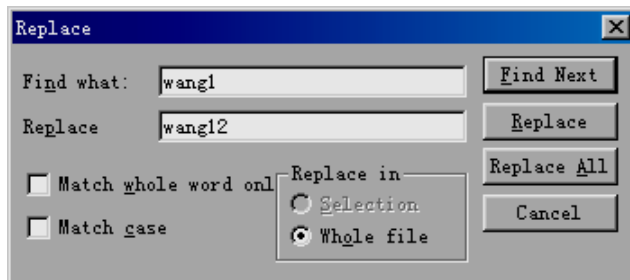
- 撤销: 可用此命令来撤销上一步操作。如果无法撤销上一步操作, 菜单的撤销命令会变灰。

- 重复：可用此命令来恢复撤消的编辑操作。如果无法恢复撤消的编辑操作，菜单上的重复命令会变灰。
- 剪切 (T)：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。如当前没有数据被选取时，此命令则不可用。把数据剪切到剪贴板上将取代原先存放在那里的内容。
- 复制 (C)：用此命令将被选取的数据复制到剪贴板上。如当前无数据被选取时，此命令则不可用。把数据复制到剪贴板上将取代以前存在那里的内容。
- 粘贴 (P)：用此命令将剪贴板上内容的一个副本插入到插入点处。如剪贴板是空的，此命令则不可用。
- 查找：点击此命令将弹出查找对话框，如附图 B-5 所示，用于查找指定字符串。
- “Find what:” 编辑框：写入你想要查找的字符串。
- “Match whole word only” 复选框：是否全字匹配。如果不选中此复选框，找到的字符串的长度有可能大于想要查找的字符串，如：我们想要查找字符串 ‘WANG1’，可能会找到字符串 ‘WANG10’，这是因为我们没有选中全字匹配复选框。
- “Match case” 复选框：是否辨认大小写。如果不选中此复选框，找到的字符串中字符的大小写可能与我们想要查找的字符串有差别，如：我们想要查找字符串 “WANG1”，可能会找到字符串 “Wang1”。
- “Up” 单选按钮：从下向上查找
- “Down” 单选按钮：从上向下查找
- “Find Next” 按钮：查找下一个字符串，如果是第一次查找则从当前光标处开始向下或向上开始查找，如果不是第一次查找，则从上一次找到的位置向下或向上开始查找。
- “Cancel” 按钮：取消查找对话框。



附图 B-5 查找对话框

- 替换：点击此命令将弹出替换对话框，如附图 B-6 所示，找到某一字符串，并用指定字符串替换之。



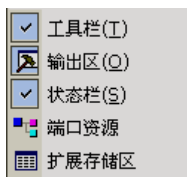
附图 B-6 替换对话框

- “Replace” 编辑框：替换后的字符串。

- “Selection” 单选按钮：如果文档中有选中部分，此按钮使能，选中此按钮则从选中部分查找和替换。
- “Whole file” 单选按钮：从整个文档中查找和替换。
- “Find Next” 按钮：查找下一个字符串。如果是第一次查找，从当前光标位置开始查找，如果不是第一次查找，则从上一次找到的位置开始查找。
- “Replace” 按钮：替换一个字符串。如果当前已经找到某一字符串，用指定字符串替换它，并找到下一个字符串，如果还没有找到某一字符串，不进行替换并找到字符串。
- “Replace All” 按钮：用指定字符串替换全部能够找到的字符串。
- “Cancel” 按钮：取消替换对话框。

(3) 查看菜单项

查看菜单如附图 B-7 所示，提供了以下命令。



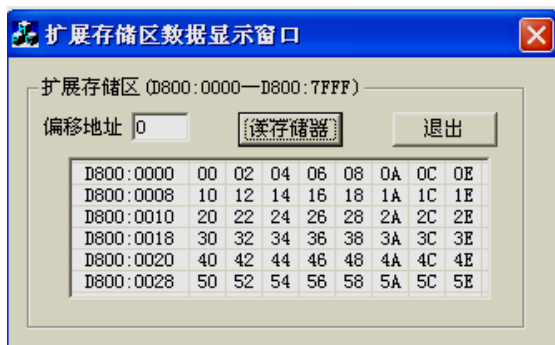
附图 B-7 查看菜单项

- 工具栏 (T)：用此命令可显示和隐藏工具栏。工具栏包括了 Tdpit 中一些主要命令的按钮，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。
- 输出区 (O)：用此命令可显示和隐藏输出区（信息栏）。
- 状态栏 (S)：此命令可用来显示和隐藏状态栏。状态栏描述了被选取的菜单项目或被按下的工具栏按钮，以及键盘的锁定状态将要执行的操作。当状态栏被显示时，在菜单项目的旁边会出现一个打勾记号。
- 端口资源：此命令可用来查看实验系统被分配的端口资源，在系统总线上共有 4 个 I/O 片选 IOY0~IOY3，每个 I/O 空间所对应的地址范围在弹出的窗口中给出。如附图 B-8 所示。



附图 B-8 查看端口资源

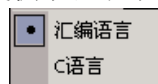
- 扩展存储区：此命令可用来查看并修改实验系统 SRAM 单元中的存储器中的内容。如图 B-9 所示。



附图 B-9 扩展存储区

(4) 语言设置菜单项

语言设置菜单如附图 B-10 所示，提供了以下命令。



附图 B-10 语言设置菜单项

汇编语言：此命令用来设置软件的当前语言环境为汇编语言环境，此时可以编辑、编译、链接和调试汇编语言程序。

C 语言：此命令用来设置软件的当前语言环境为 C 语言环境，此时可以编辑、编译、链接和调试 C 语言程序。

(5) 编译菜单项

编译菜单如附图 B-11 所示，提供了以下命令。



附图 B-11 编译菜单项

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。如果有错误或警告生成，则在输出区显示错误或警告信息，双击错误或警告信息，可定位到有错误或警告的行，修改有错误或警告的行后应重新“编译”。编译时自动保存源文件中所做的修改。

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。如果有错误或警告生成，则在输出区显示错误或警告信息，查看错误或警告信息修改源程序，修改后应重新“编译”和“链接”。

运行 (R)：执行当前连接成功的可执行程序。当前激活的程序编译连接成功或者该程序已经编译过，可执行程序已经存在，这时就可运行该程序。如果该程序没有连接成功，或者没有被连接过，可执行程序不存在，则不可以执行“运行”。所有实验例程均设计为按任意键退出运行状态。

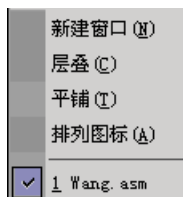
调试 (D)：打开调试环境进行当前程序的调试。每次打开或者新建一个新的程序，都必须

先进行编译连接，然后才可以执行该操作，进入调试环境。调试完毕后按“Alt + X”键退出调试环境。调试环境的操作详见调试环境的帮助。

显示模式：可以将程序调试或者运行环境调整到全屏或者窗口模式。

(6) 窗口菜单项

窗口菜单如附图 B-12 所示，提供了以下命令。



附图 B-12 窗口菜单项

新建窗口 (N)：用此命令来打开一个具有与活动的窗口相同内容的新窗口。您可同时打开数个文档窗口以显示文档的不同部分或视图。如果您对一个窗口的内容做了改动，所有其它包含同一文档的窗口也会反映出这些改动。

层叠 (C)：用此命令按相互重叠形式来安排多个打开的窗口。

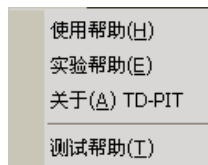
平铺 (T)：用此命令按互不重叠形式来安排多个打开的窗口。

排列图标 (A)：用此命令在主窗口的底部安排被最小化的窗口的图标。如果在主窗口的底部有一个打开的窗口，则有可能会看不见某些或全部图标，因为它们在这个文档窗口的下面。

窗口选择：Tdpit 在窗口菜单的底部显示出当前打开的文档窗口的清单。有一个打勾记号出现在活动的窗口的文档名前。从该清单中挑选一个文档可使其窗口成为活动窗口。

(8) 帮助菜单项

帮助菜单如附图 B-13 所示，提供了以下命令。



附图 B-13 帮助菜单项

使用帮助 (H)：用此命令来显示帮助的开场屏幕。从此开场屏幕，您可跳到关于使用 Tdpit 的一些指令以及各种不同类型参考资料。

实验帮助 (E)：用此命令来显示帮助的开场屏幕。从此开场屏幕，您可跳到关于使用 Tdpit 系列实验系统做实验时的一些相关信息。其中为每个实验提供了详细的实验说明及相关的实验原理，参考设计流程及实验步骤和实验接线等。

关于 (A) TD-PIT：用此命令来显示您的 Tdpit 版本的版权通告和版本号码。

测试帮助 (T)：用此命令来显示基于 PCI 桥接卡扩展到设备上的 PC 机资源分配是否正确。











2. 工具栏功能介绍

(1) 标准工具栏

标准工具栏共有十个按钮，如附图 B-14 所示。



附图 B-14 标准工具栏










-  新建按钮：用此按钮在 Tdpit 中建立一个新文档。
-  打开按钮：用此命令在一个新的窗口中打开一个现存的文档。
-  保存按钮：用此命令来关闭当前活动文档。
-  剪切按钮：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。
-  复制按钮：用此命令将被选取的数据复制到剪贴板上。
-  粘贴按钮：用此命令将剪贴板上内容的一个副本插入到插入点处。
-  打印按钮：用此命令来打印一个文档。
-  撤消按钮：如果可能的话，可用此命令来撤消上一步编辑操作。
-  重复按钮：如果可能的话，可用此命令来恢复撤消的编辑操作。
-  显隐输出区按钮：用此按钮可显示和隐藏输出区。

(2) 编译工具栏

编译工具栏共有 9 个按钮，如附图 B-15 所示。



附图 B-15 编译工具栏

-  编译按钮：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。
-  链接按钮：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。
-  运行按钮：执行当前连接成功的可执行程序。
-  调试按钮：打开调试环境进行当前程序的调试。
-  进入 DOS 环境按钮：此按钮提供一个进入 DOS 环境的快捷工具，若想进入 DOS 环境进行命令行操作，可以按此按钮。
-  查看端口资源按钮：此按钮功能可用来查看实验系统被分配的端口资源。
-  扩展存储区显示按钮：此按钮可以查看并修改设备上的 SRAM 单元中的存储器内容。
-  专用图形界面按钮：启动专用图形显示界面，相关说明详见附录 C。
-  示波器界面按钮：启动示波器显示界面，相关说明详见附录 D。

4. 编程信息

当 PC 系统开电时,自动为实验系统分配了相应的资源配置。但打开 Td-Pit 软件后,Td-Pit 把 Windows 的资源配置进行了重新配置,重新分配的 I/O 地址、中断信号如表 B-1 所示。

附表 B-1 端口分配范围

IOY0	3000H~303FH
IOY1	3040H~307FH
IOY2	3080H~30BFH
IOY3	30C0H~30FFH
INTR1	IRQ7
INTR2	IRQ6

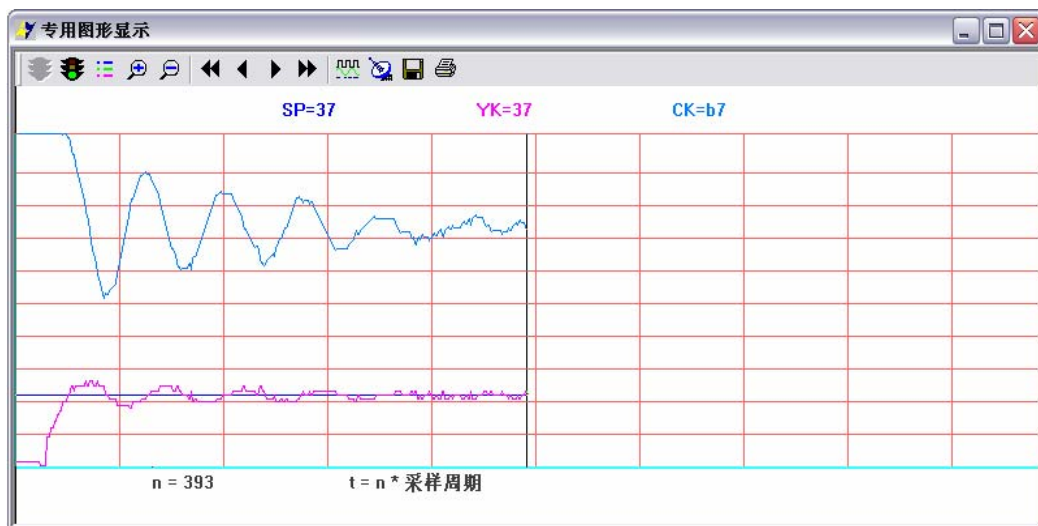
为了在 Windows 下可以完成存储器实验,Td-Pit 把存储空间 MY0 映射到了 D8000H~DFFFFH。

实验程序中资源设置举例


```
;*****
IOY0          EQU      3000H          ;片选 IOY0 对应的端口始地址
IOY1          EQU      3040H          ;片选 IOY1 对应的端口始地址
IOY2          EQU      3080H          ;片选 IOY2 对应的端口始地址
IOY3          EQU      30C0H          ;片选 IOY3 对应的端口始地址
INTR_IVADD1   EQU      003CH          ;INTR1 对应的中断矢量地址
INTR_IVADD2   EQU      0038H          ;INTR2 对应的中断矢量地址
;*****
```


附录 C 专用图形显示


专用图形显示功能主要用于观察“直流电机闭环调速实验”跟踪曲线（实时显示给定转速、反馈转速的曲线响应关系）及“温度闭环控制实验”跟踪曲线（实时显示给定温度和反馈温度的曲线响应关系）。该功能界面如下图所示。

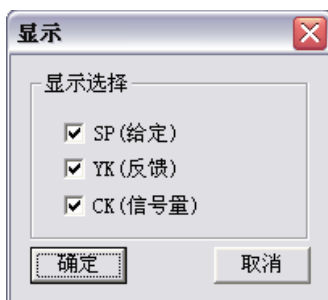



1. 工具栏功能简介


(1)  按钮：在运行状态下使能，使波形暂停显示并出现游标。


(2)  按钮：在暂停状态下使能，使波形继续显示，游标消失。

(3)  按钮：点击此按钮，出现如下图所示对话框。可以分别选择专用图形显示窗口中 3 路通道波形的显示和关闭。



(4)  按钮：放大波形。


(5)  按钮：缩小波形。

(6)  按钮：在暂停状态下，使游标快速向左移动。“SP=”后显示的是游标所在时刻要求电机达到的转速值，“YK=”后显示的是游标所在时刻电机实际达到的转速值。“CK=”后显示的是游标所在时刻控制量的数值。“ $t=n \times \text{采样周期}$ ”表示游标所在位置的时刻与图形最左端时刻的差值。


(7)  按钮：在暂停状态下，使游标向左缓慢移动。


(8)  按钮：在暂停状态下，使游标向右缓慢移动。


(9)  按钮：在暂停状态下，使游标快速向右移动。

(10)  按钮：点击此按钮，出现如下图所示对话框。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。



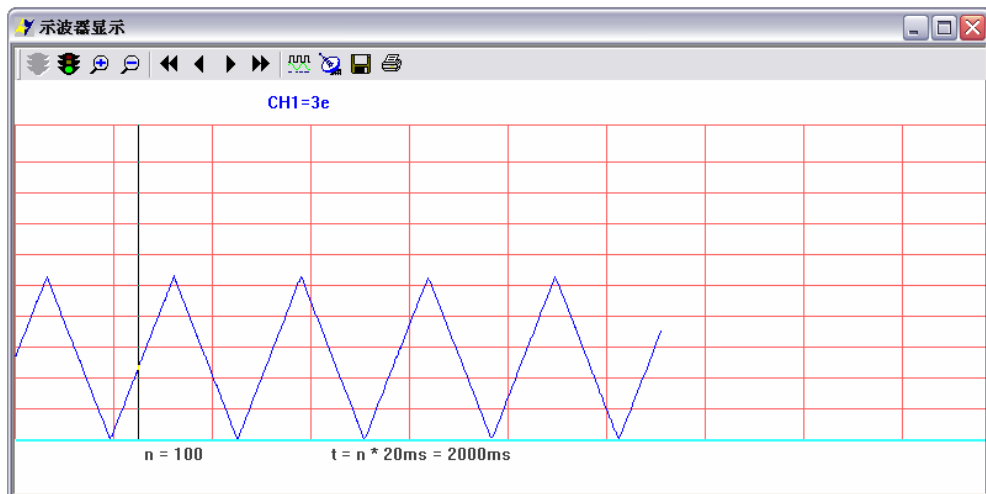
(11)  按钮：显示保存到图一，图二和图三中的波形，此时可以对几副图进行比较。

(12)  按钮：以.bmp 格式保存当前屏幕上的波形到指定文件。

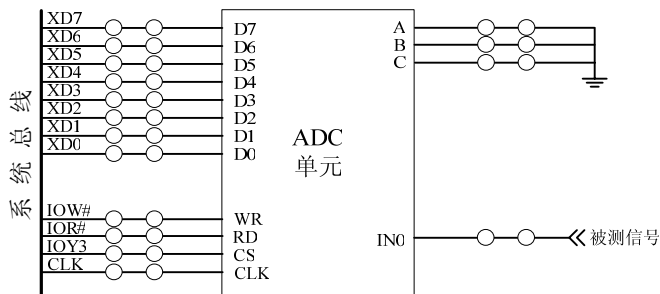
(13)  按钮：打印当前屏幕上的波形。





附录 D 示波器






示波器功能主要是利用实验箱上 A/D 转换单元，实现了一个简易示波器的测量功能。可用于“8254 定时/计数器应用实验”、“D/A 转换实验”及“8251 串行接口应用实验”中信号波形的观察。该功能界面如下图所示。






使用该功能时，需要在各实验接线基础上增加以下接线。



- (1)  按钮：在运行状态下使能，使波形暂停显示并出现游标。
- (2)  按钮：在暂停状态下使能，使波形继续显示，游标消失。
- (3)  按钮：放大波形。
- (4)  按钮：缩小波形。

- (5)  按钮：在暂停状态下，使游标快速向左移动。“CH1=”后显示游标当前位置的采样值。“ $t = n \times 20\text{ms} = x \text{ ms}$ ”表示游标所在位置的时刻与图形最左端时刻的差值。
- (6)  按钮：在暂停状态下，使游标向左缓慢移动。
- (7)  按钮：在暂停状态下，使游标向右缓慢移动。
- (8)  按钮：在暂停状态下，使游标快速向右移动。
- (9)  按钮：点击此按钮，出现如下图所示对话框。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。



- (10)  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。
- (11)  按钮：以.bmp 格式保存当前屏幕上的波形到指定文件。
- (12)  按钮：打印当前屏幕上的波形。

附录 E 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
A1.ASM	2.1 显示程序实验
A2.ASM	2.2 数据传送实验
A3-1.ASM A3-2.ASM A3-3.ASM A3-4.ASM	2.3 数码转换实验 <ol style="list-style-type: none"> 1. 将 ASCII 码十进制数转换为二进制数 2. 将十进制数的 ASCII 码转换为 BCD 码 3. 将十六进制的 ASCII 码转换为十进制数 4. BCD 码转换为二进制码
A4-1.ASM A4-2.ASM A4-3.ASM A4-4.ASM	2.4 运算类程序实验 <ol style="list-style-type: none"> 1. 二进制双精度加法运算 2. 十进制数的 BCD 码减法运算 3. 乘法运算 4. 用减奇法开平方运算
A5.ASM	2.5 分支程序设计实验
A6.ASM	2.6 循环程序设计实验
A7-1.ASM A7-2.ASM	2.7 子程序设计实验 <ol style="list-style-type: none"> 1. 数据移动实验 2. 数码转换及显示实验

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	
IO-8.ASM IO-32.ASM	IO-8.C	4.1 8/32 位 I/O 接口设计实验 1. 8 位 I/O 接口设计实验 2. 32 位 I/O 接口设计实验
T138.ASM	T138.C	4.2 地址译码电路设计实验
Mem-8.ASM Mem-32.ASM		4.3 静态存储器扩展实验 1. 8 位存储器读写 2. 32 位存储器读写
T8259-1.ASM T8259-2.ASM T8259-3.ASM		4.2 8259 中断控制实验 1. 8259 单中断应用实验 2. 8259 中断优先级应用实验 3. 8259 级连中断应用实验
T8237-1.ASM T8237-2.ASM T8237-3.ASM	T8237.C	4.3 8237 DMA 控制实验 1. 存储器到存储器 DMA 传输实验 2. I/O 到存储器 DMA 传输实验 3. 存储器到 I/ODMA 传输实验
T8254-1.ASM T8254-2.ASM	T8254-2.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
T8255-1.ASM T8255-2.ASM T8255-3.ASM	T8255-1.C T8255-2.C	4.5 8255 并行接口应用实验 1. 基本输入输出实验 2. 流水灯显示实验 3. 方式一输入输出实验
T8251-1.ASM T8251-2.ASM T8251-3.ASM T8251-4.ASM	T8251-1.C T8251-2.C T8251-3.C T8251-4.C	4.6 8251 串行接口应用实验 1. 串行通讯基础实验 2. 自发自收通讯应用实验 3. 双机通讯实验接收程序 4. 双机通讯实验发送程序
T0809.ASM	T0809.C	4.7 A/D 转换实验
T0832-1.ASM T0832-2.ASM	T0832-1.C T0832-2.C	4.8 D/A 转换实验 1. 产生方波 2. 产生三角波
Keyscan.ASM	Keyscan.C	4.9 键盘扫描及显示设计实验
SOUND.ASM	SOUND.C	4.10 电子发声设计实验
LED-HZ.ASM LED-HZ.INC	Led-HZ.C Led-HZ.h	4.11 点阵 LED 显示设计实验
LCD.ASM Lcd.inc	Lcd.C Lcd.h	4.12 图形 LCD 显示设计实验
BUJIN.ASM	Bujin.C	4.13 步进电机实验
ZHILIU.ASM		4.14 直流电机闭环调速实验
TEM.ASM		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	
P-EXAM.ASM	5.8 保护模式与实模式切换实例
P1-1.ASM	6.1 全局描述符及全局描述表实验
P1-2.ASM	6.1 局部描述符及局部描述表实验
P2-1.ASM	6.2 任务内无特权级变换的转移实验
P2-2.ASM	6.2 任务内有特权级变换的转移实验
P3-1.ASM	6.3 任务间无特权级切换实验
P3-2.ASM	6.3 任务间有特权级切换实验
P4-1.ASM	6.4 中断门、陷阱门实现中断/异常实验
P4-2.ASM	6.4 任务门实现中断/异常实验
PMEM-32.ASM	8.1 保护模式下的存储器扩展实验