

80X86 微机原理实验教程



西安唐都科教仪器公司

Copyright Reserved 2022

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，2022(C)，All Right Reserved.

80X86 微机原理实验教程

©版权所有 未经许可 严禁复制

唐都公司网址: <http://www.tangdu.com>

技术支持邮箱: tangdukejiao@126.com

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

目录

第一章 80x86 实模式微机原理	1
(一) 实模式下的 80X86 机器组织	1
1.1 80X86 寄存器	1
1.2 80X86 存储器寻址	3
1.3 80X86 指令集	4
(二) 16 位指令及其程序设计实验	13
2.1 系统认识实验	13
2.2 数制转换实验	18
2.3 运算类编程实验	23
2.4 分支程序设计实验	26
2.5 循环程序设计实验	28
2.6 排序程序设计实验	31
2.7 子程序设计实验	33
2.8 查表程序设计实验	35
2.9 输入输出程序设计实验	36
(三) 32 位指令及其程序设计实验	38
3.1 32 位寄存器和 32 位指令使用：双字排序并显示	40
3.2 32 位寄存器和 32 位指令使用：ASCII 转换 16 进制	44
(四) 总线地址译码及编程实验	45
4.1 总线地址译码及时序观测与分析实验	45
(五) 存储器管理及编程实验	48
5.1 静态存储器扩展及时序观测与分析实验	48
(六) PIC 中断管理及 8259 编程实验	53
6.1 8259 单一中断源控制实验	53
6.2 8259 优先级中断管理实验	58
6.3 8259 级连中断管理及时序观测与分析实验	60
(七) DMA 特性及 8237 编程实验	62
7.1 存储器到存储器 DMA 传送及时序观测与分析实验	62
7.2 存储器到 I/O 设备 DMA 传送及时序观测与分析实验	69
(八) 扩展接口电路及编程实验	72
8.1 基本 I/O 输入输出及时序观测与分析实验	72
8.2 8255 并行接口实验	78
8.3 8254 定时/计数器应用实验	82
8.4 8251 串行接口应用实验	87
8.5 A/D 转换实验	94
8.6 D/A 转换实验	97

第二章 80x86 保护模式微机原理	101
(一) 保护模式下的 80X86 机器组织	101
1.1 实模式和保护模式	101
1.2 寄存器组织	101
1.3 保护模式下的分段存储管理机制	103
1.4 任务管理的概念	109
1.5 任务内的控制转移	111
1.6 任务间的控制转移	114
1.7 中断/异常管理	115
1.8 80X86 保护模式程序设计	118
(二) 保护模式微机原理及其程序设计实验	126
2.1 描述符及描述符表实验	126
2.2 特权级变换实验	136
2.3 任务切换实验	139
2.4 中断与异常处理实验	143
(三) 80X86 虚拟存储器的组织及其管理	146
3.1 分段管理机制	146
3.2 分页管理机制	153
(四) 保护模式下的存储器扩展及其应用实验	159
4.1 无分页机制的存储器扩展实验	159
4.2 具有分页机制的存储器扩展实验	161
附录 1 TDX-PITE 联机软件使用说明	163
附 1.1 菜单功能	163
附 1.2 工具栏功能介绍	166
附 1.3 专用图形显示	168
附 1.4 示波器	171
附 1.5 时序观测窗	173
附 1.6 Debug 调试命令	174
附录 2 系统实验程序清单	177
附录 3 系统编程信息	180
附 3.1 地址分配情况	180
附 3.2 常用 BIOS 及 DOS 功能调用说明	182

第一章 80x86 实模式微机原理

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。

(一) 实模式下的 80X86 机器组织

1.1 80X86 寄存器

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

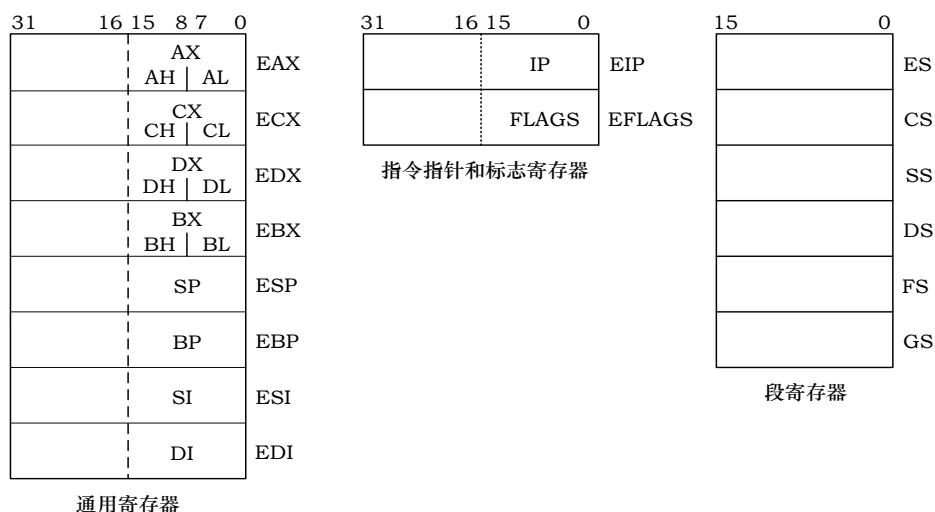


图1.1 80X86的部分寄存器

1.1.1 通用寄存器

80X86 有 8 个 32 位通用寄存器，这 8 个寄存器分别命名为 EAX、ECX、EDX、EBX、

ESP、EBP、ESI 和 EDI。它们是原先的 16 位通用寄存器的扩展，请参考图 1.1。这些通用寄存器的低 16 位可以作为 16 位的寄存器独立存取，并把它们命名为 AX、CX、DX、BX、SP、BP、SI 和 DI，它们也就是 X86 系列微处理器先前的 8 个 16 位通用寄存器。

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

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

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

在以前的微处理器中，只有 BX、BP、SI 和 DI 可以在基地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

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 PUSHA 和全出栈指令 POPA

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

PUSHA

POPA

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

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

(7) 32 位全进栈指令 PUSHAD 和全出栈指令 POPAD

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

PUSHAD

POPAD

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

栈的内容是 PUSHAD 执行之前的值。

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

这两条指令各不影响标志。

2. 地址传送指令组

(1) 装入有效地址指令 LEA

装入有效地址指令的格式和功能同 8086/8088。源操作数仍然必须是存储器操作数，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 默认的另一操作数是 EAX, 其功能是把 EAX 内容乘上源操作数内容所得积送入 EDX: EAX 中, 若结果的高 32 位为 0, 那么标志 CF 和 OF 被清 0, 否则被置 1; 除法指令 DIV 默认的被除数是 EDX: EAX, 其功能是把指令中给出的操作数作为除数, 所得的商送 EAX, 余数送 EDX。

(2) 有符号数乘法和除法指令

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

另外, 80X86 还提供了新形式的有符号数乘法指令。如:

```
IMUL    DST, SRC
IMUL    DST, SRC1, SRC2
```

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

3. 符号扩展指令组

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

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

CWDE

CDQ

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

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

4. 十进制调整指令组

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

1.3.3 逻辑运算和移位指令

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

1. 逻辑运算指令组

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

2. 一般移位指令组

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

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

3. 循环移位指令组

循环移位指令组有 4 条指令：ROL、ROR、RCL 和 RCR。

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

4. 双精度移位指令组

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

SHLD OPRD1, OPRD2, m

SHRD OPRD1, OPRD2, m

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

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

双精度右移指令 SHRD 的功能是把操作数 OPRD1 右移指定的 m 位，空出的位用操作数

OPRD2 低端的 m 位填补, 但操作数 OPRD2 的内容不变, 最后移出的位保留在进位标志 CF 中。当移位位数是 1 时, OF 标志受影响, 否则清 OF。

1.3.4 控制转移指令

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

1. 转移指令组

(1) 无条件转移指令

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

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

(2) 条件转移指令

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

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

JECXZ OK

它表示当 32 位寄存器 ECX 为 0 时, 转移到标号 OK 处。

2. 循环指令组

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

3. 过程调用和返回指令组

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

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

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

4. 中断调用和中断返回指令组

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

1.3.5 串操作指令

从 80X86 开始, 串操作的基本单位在字节和字的基础上增加了双字。

1. 基本串操作指令

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

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

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

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

2. 重复前缀

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

3. 串输入指令

串输入指令的格式如下：

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

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

INS DSTS, DX

4. 串输出指令

串输出指令的格式如下：

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

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

OUTS DX, SRCS

1.3.6 条件字节设置指令

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

条件字节设置指令的一般格式为：

SET** OPRD

共有以下 30 个指令：

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

1.3.7 位操作指令

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

位操作指令可分为位扫描指令和位测试及设置指令组。

1. 位扫描指令组

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

BSF OPRD1, OPRD2

BSR OPRD1, OPRD2

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

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

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

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

2. 位测试及设置指令组

位测试及设置指令含有 4 条指令，其格式如下：

BT OPRD1, OPRD2

BTC OPRD1, OPRD2

BTR OPRD1, OPRD2

BTS OPRD1, OPRD2

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

1.3.8 处理器控制指令

处理器控制指令用于设置标志、空操作和与外部事件同步等。

1. 设置标志指令组

设置进位标志 CF 的指令 CLC、STC 和 CMC 保持原先相同。

设置方向标志 DF 的指令 CLD 和 STD 保持原先相同。

设置中断允许标志 IF 的指令 CLI 和 STI 的功能在实模式下保持与原先相同。在保持模式

下它们是 I/O 敏感指令。

2. 空操作指令组

空操作指令 NOP 的一般格式如下：

NOP

空操作指令的功能是什么都不干，该指令就一个字节的操作码。

3. 外同步指令和前缀

(1) 等待指令 WAIT

等待指令 WAIT 的一般格式如下：

WAIT

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

(2) 封锁前缀 LOCK

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

(二) 16 位指令及其程序设计实验

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

2.1 系统认识实验

2.1.1 实验目的

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

2.1.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.1.3 实验内容

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

2.1.4 实验步骤

(1) 运行 TDX-PITE 软件，进入 TDX-PITE 集成开发环境。

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

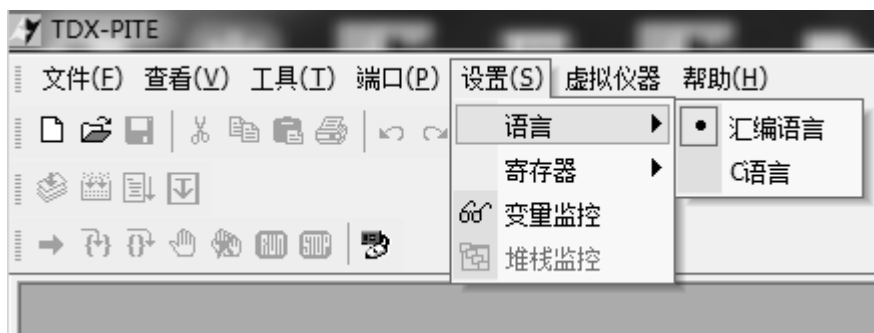


图 2.1 语言环境设置界面

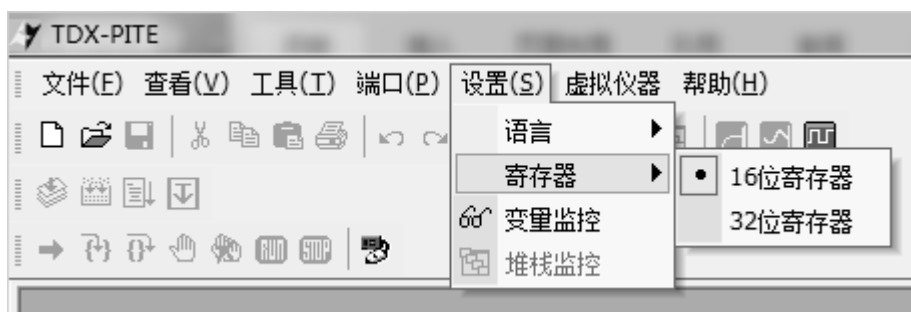


图 2.2 寄存器设置界面

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



图 2.3 新建文件界面

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



```

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

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

```

图 2.4 程序编辑界面

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

(6)



图 2.5 编译输出信息界面

(6) 连接 PC 与实验系统的通讯电缆，打开实验系统电源。



(7) 编译、链接都正确并且上下位机通讯成功后，就可以下载程序，联机调试了。可以通过端口列表中的“端口测试”来检查通讯是否正常。点击  下载程序。  为编译、链接、下载组合按钮，通过该按钮可以将编译、链接、下载一次完成。下载成功后，在输出区的结果窗中会显示“加载成功！”，表示程序已正确下载。起始运行语句下会有一条绿色的背景。如图 2.6 所示。




图 2.6 加载成功显示界面

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



图 2.7 内存地址单元数据显示

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

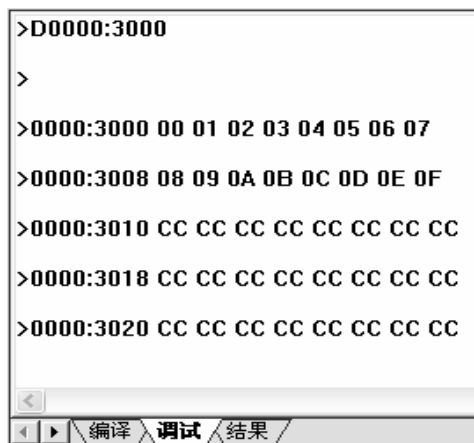


图 2.8 运行程序后数据变化显示

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

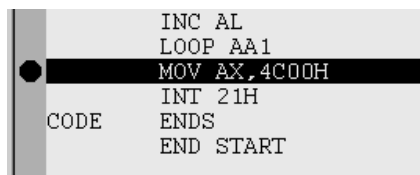


图 2.9 断点设置显示

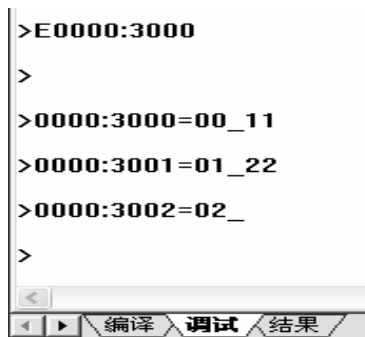


图 2.10 修改内存单元数据显示界面

实验例程文件名为 Wmd861.asm。

2.1.5 操作练习

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

2.2 数制转换实验

2.2.1 实验目的

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

2.2.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.2.3 实验内容及步骤

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

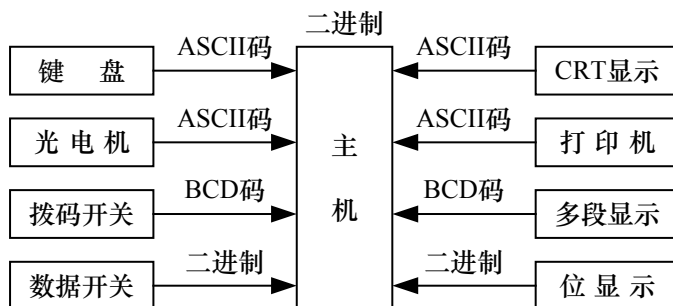


图 2.11 数制转换关系

1. 将 ASCII 码表示的十进制数转换为二进制数
十进制表示为：

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

D_i 代表十进制数 0, 1, 2, ..., 9;

上式转换为：

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

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

表 2.1 数制对应关系表

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

程序流程图如图 2.12 所示，实验程序参考例程。

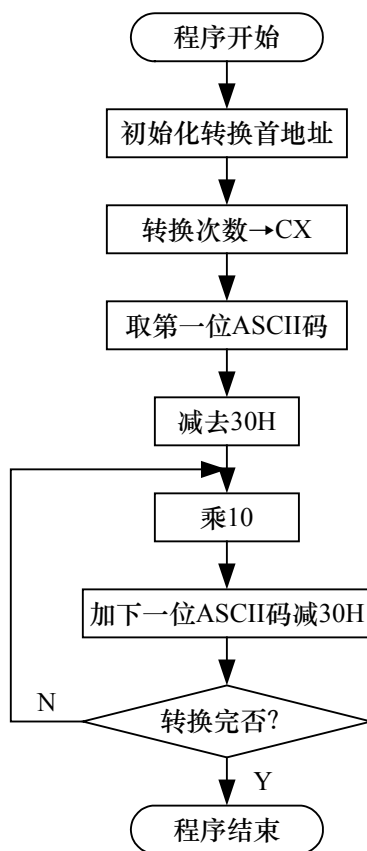


图 2.12 转换程序流程图

实验步骤

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

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

(3) 运行程序，然后停止程序。

(4) 查看 AX 寄存器，即为转换结果，应为：0100。

(5) 反复试几组数据，验证程序的正确性。

2. 将十进制数的 ASCII 码转换为 BCD 码

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

实验步骤

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

误后装入系统。

(2) 在 3500H~3504H 单元中存放五位十进制数的 ASCII 码, 即: 键入 E3500 后, 输入 31, 32, 33, 34, 35。

(3) 运行程序, 待程序运行停止。

(4) 在调试窗口键入 D350A, 显示运行结果, 应为:

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

(5) 反复测试几组数据, 验证程序功能。

3. 将十六位二进制数转换为 ASCII 码表示的十进制数

十六位二进制数的值为 0~65535, 最大可转换为五位十进制数。

五位十进制数可表示为:

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

D_i : 表示十进制数 0~9

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

实验步骤

(1) 编写程序 (例程文件名: A2-3.ASM), 经编译、链接无误后, 装入系统。

(2) 在 3500H、3501H 中存入 0C 00。

(3) 运行程序, 待程序运行停止。

(4) 检查运行结果, 键入 D3510, 结果应为: 30 30 30 31 32。

(5) 可反复测试几组数据, 验证程序的正确性。

4. 十六进制数转换为 ASCII 码

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

实验步骤

(1) 编写程序 (例程文件名为 A2-4.ASM), 经编译、链接无误后装入系统。

(2) 在 3500H、3501H 中存入四位十六进制数 203B, 即键入 E3500, 然后输入 3B 20。

(3) 先运行程序, 待程序运行停止。

(4) 键入 D350A, 显示结果为: 0000:350A 32 30 33 42 CC ...。

(5) 反复输入几组数据, 验证程序功能。

5. BCD 码转换为二进制数

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

实验步骤

(1) 编写程序（例程文件名为：A2-5.ASM），经编译、链接无误后装入系统。

(2) 将四个二位十进制数的 BCD 码存入 3500H~3507H 中，即：

先键入 E3500，然后输入 01 02 03 04 05 06 07 08。

(3) 先运行程序，待程序运行停止。

(4) 键入 D3510 显示转换结果，应为：0C 22 38 4E。

(5) 反复输入几组数据，验证程序功能。

2.2.4 思考题

1. 实验内容 1 中将一个五位十进制数转换为二进制数（十六位）时，这个十进制数最小可为多少，最大可为多少？为什么？

2. 将一个十六位二进制数转换为 ASCII 码十进制数时，如何确定 D_i 的值？

3. 在十六进制转换为 ASCII 码时，存转换结果后，为什么要把 DX 向右移四次？

4. 自编 ASCII 码转换十六进制、十六进制小数转换二进制、二进制转换 BCD 码的程序，并调试运行。

2.3 运算类编程实验

2.3.1 实验目的

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

2.3.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.3.3 实验内容及步骤

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

表 2.2 数据类型算术运算表

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

1. 二进制双精度加法运算

计算 $X+Y=Z$ ，将结果 Z 存入某存储单元。实验程序参考例程。

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

实验步骤

- (1) 编写程序（例程文件名为：A3-1.ASM），经编译、链接无误后装入系统。
- (2) 程序装载完成后，点击‘变量区’标签将观察窗切换到变量监视窗口。


(3) 点击 ，将变量 XH, XL, YH, YL, ZH, ZL 添加到变量监视窗中，然后修改 XH, XL, YH, YL 的值，如图 2.13 所示，修改 XH 为 0015，XL 为 65A0，YH 为 0021，YL 为 B79E。




图 2.13 变量监视窗口

- (4) 在 JMP START 语句行设置断点，然后运行程序。
- (5) 当程序遇到断点后停止运行，查看变量监视窗口，计算结果 ZH 为 0037，ZL 为 1D3E。
- (6) 修改 XH，XL，YH 和 YL 的值，再次运行程序，观察实验结果，反复测试几组数据，验证程序的功能。

2. 十进制的 BCD 码减法运算

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

实验步骤

- (1) 输入程序（例程文件名为 A3-2.ASM），编译、链接无误后装入系统。
- (2) 点击  将变量 X，Y，Z 添加到变量监视窗中，并为 X，Y 赋值，假定存入 40 与 12 的 BCD 码，即 X 为 0400，Y 为 0102。
- (3) 在 JMP START 语句行设置断点，然后运行程序。
- (4) 程序遇到断点后停止运行，观察变量监视窗，Z 应为 0208。
- (5) 重新修改 X 与 Y 的值，运行程序，观察结果，反复测试几次，验证程序正确性。

3. 乘法运算

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

实验步骤

- (1) 编写程序（例程文件名为 A3-3.ASM），编译、链接无误后装入系统。
- (2) 查看寄存器窗口获得 CS 的值，使用 U 命令可得到数据段地址 DS，然后通过 E 命令为被乘数及乘数赋值，如被乘数：01 02 03 04 05，乘数：01，方法同实验内容 1。
- (3) 运行程序，待程序运行停止。

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

(5) 反复测试几组数据，验证程序的正确性。

2.4 分支程序设计实验

2.4.1 实验目的

1. 掌握分支程序的结构。
2. 掌握分支程序的设计、调试方法。

2.4.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.4.3 实验内容

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

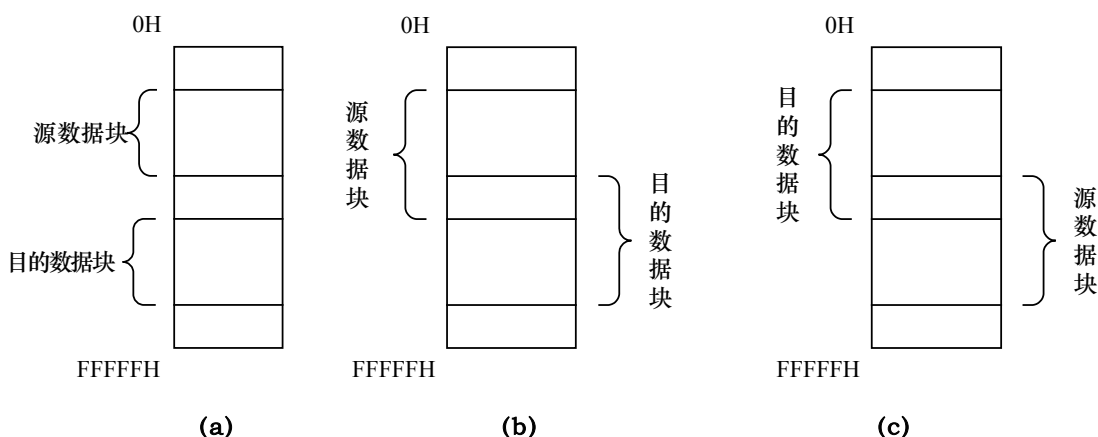


图 2.14 源数据块与目的数据块在存储中的位置情况

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

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

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

实验程序流程图如图 2.15 所示。

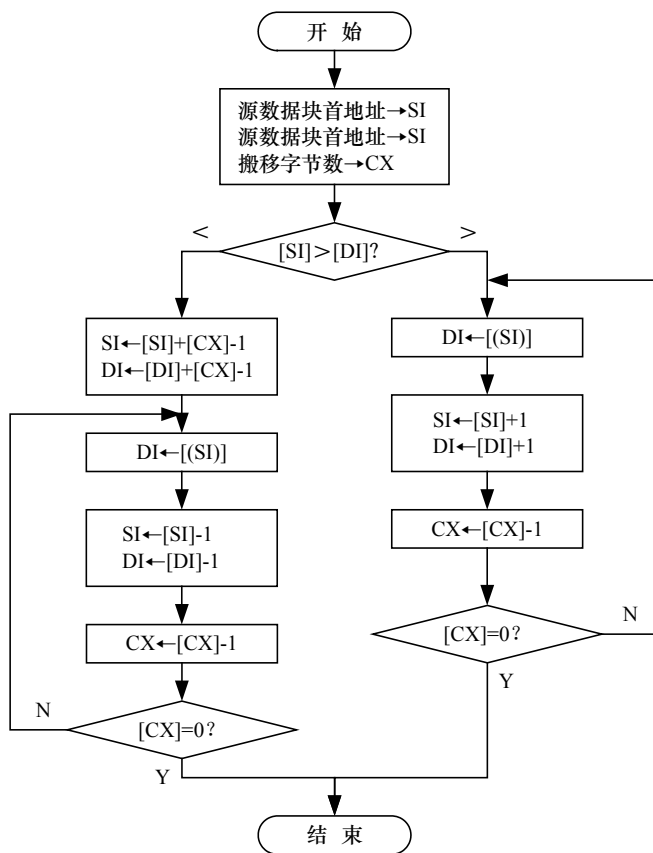


图 2.15 程序流程图

2.4.4 实验步骤

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

2.5 循环程序设计实验

2.5.1 实验目的

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

2.5.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.5.3 实验内容及步骤

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

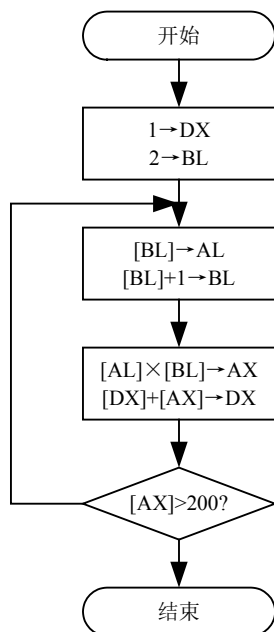


图 2.16 程序流程图

实验步骤

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

2. 求某数据区内负数的个数

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

实验程序流程图如图 2.17 所示。

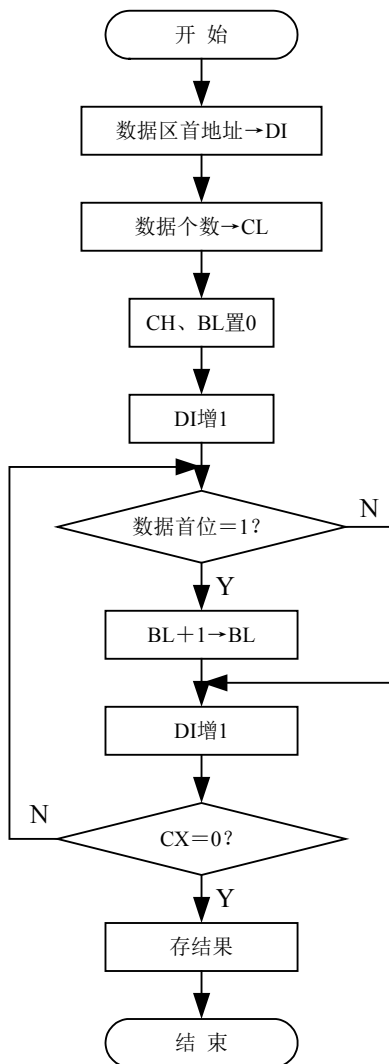


图 2.17 程序流程图

实验步骤

(1) 按实验流程编写实验程序（例程文件名为：A5-2.ASM）。

(2) 编译、链接无误后装入系统。

(3) 键入 E3000，输入数据如下：

3000=06 （数据个数）

3001=12

3002=88

3003=82

3004=90

3005=22

3006=33

(4) 先运行程序，待程序运行停止。

(5) 查看 3007 内存单元或寄存器 BL 中的内容，结果应为 03。

(6) 可以进行反复测试来验证程序的正确性。

2.6 排序程序设计实验

2.6.1 实验目的

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

2.6.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.6.3 实验内容及步骤

1. 气泡排序法

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

设计思想：

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

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

(3) 再重复 (2)，直到 N 个数全部排列好为止。

实验步骤

- (1) 分析参考程序（例程文件名为：A6-1.ASM），绘制流程图并编写实验程序。
- (2) 编译、链接无误后装入系统。
- (3) 键入 E3000 命令修改 3000H~3009H 单元中的数，任意存入 10 个无符号数。
- (4) 先运行程序，待程序运行停止。
- (5) 通过键入 D3000 命令查看程序运行的结果。
- (6) 可以反复测试几组数据，观察结果，验证程序的正确性。

2. 学生成绩名次表

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

实验步骤

- (1) 绘制流程图，并编写实验程序（例程文件名为：A6-2.ASM）。
- (2) 编译、链接无误后装入系统。

- (3) 将 10 个成绩存入首地址为 3000H 的内存单元中。
- (4) 调试并运行程序。
- (5) 检查 3100H 起始的内存单元中的名次表是否正确。

2.7 子程序设计实验

2.7.1 实验目的

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

2.7.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.7.3 实验内容及步骤

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

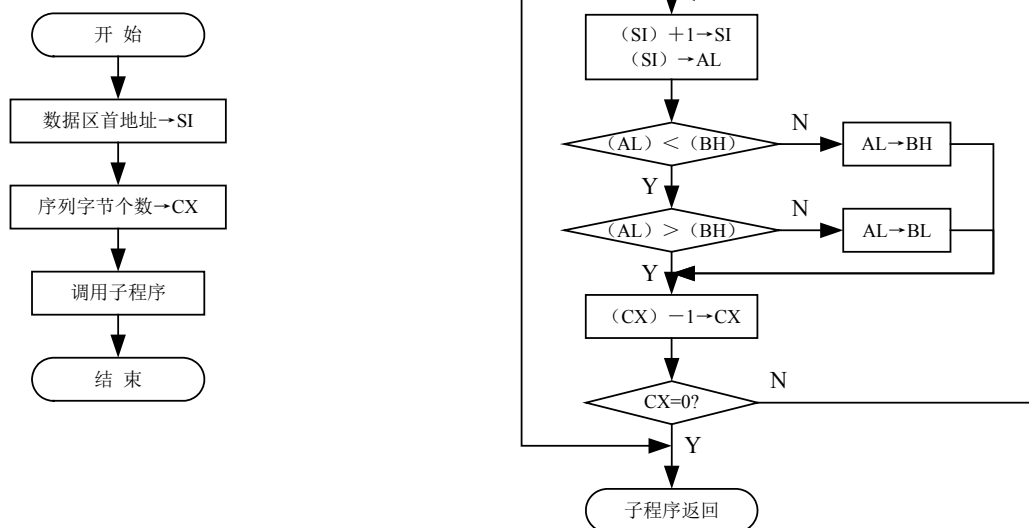


图 2.18 程序流程图

实验步骤

- (1) 根据程序流程图编写实验程序（例程文件名为：A7-1.ASM）。
- (2) 经编译、链接无误后装入系统。
- (3) 键入 E3000 命令，输入 8 个字节的数据，如：D9 07 8B C5 EB 04 9D F9。

(4) 运行实验程序。

(5) 点击停止按钮，停止程序运行，观察寄存器窗口中 AX 的值，AX 应为 F9 04，其中 AH 中为最大值，AL 中为最小值。

(6) 反复测试几组数据，检验程序的正确性。

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

2. 求 N!

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

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

$$0! = 1$$

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

定义两个变量 N 及 RESULT，RESULT 中存放 N! 的计算结果，N 在 00H~08H 之间取值。

实验步骤

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

表 2.3 阶乘表

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

2.8 查表程序设计实验

2.8.1 实验目的

学习查表程序的设计方法。

2.8.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.8.3 实验内容

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

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

2.8.4 实验步骤

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

2.9 输入输出程序设计实验

2.9.1 实验目的

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

2.9.2 实验设备

PC 机一台, TDX-PITE 实验装置一套。

2.9.3 实验内容及步骤

INT 21H 功能调用使用说明如下:

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

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

实验步骤

- (1) 编写实验程序 (例程文件名为: A9-1.ASM 或 CDISPLAY.C), 经编译、链接无误后装入系统。
- (2) 运行实验程序, 观察实验结果。
- (3) 修改实验程序, 在显示器上显示 'GOOD AFTERNOON', 要求使用 AH=09 功能 (显示一字符串功能块) 完成。

2. INT 21H 功能调用示例程序实验。

实验步骤

- (1) 参考附录中 INT 21H 功能调用使用说明，编写实验程序（例程文件名为：A9-2. ASM），经编译、链接无误后装入系统。
- (2) 运行实验程序，观察实验结果，可根据需要设断点观测实验现象。
- (3) 仔细分析实验内容，理解 INT 21H 各功能调用的用法。

(三) 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。

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。

3.1 32 位寄存器和 32 位指令使用：双字排序并显示

3.1.1 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。

3.1.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

3.1.3 实验内容

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

1. 实验步骤

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

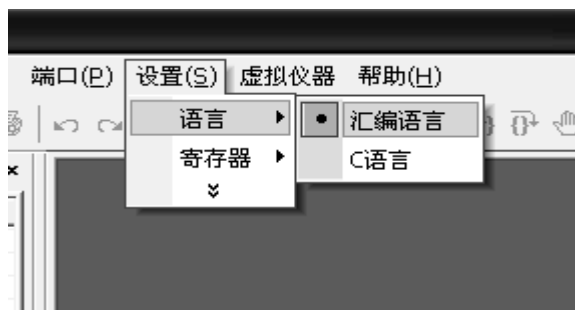


图 3.1 语言设置环境界面

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

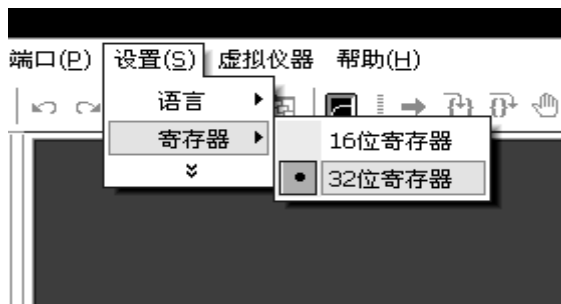


图 3.2 寄存器设置环境界面

设置好以后，下次再启动软件，设置栏将保持这次的修改不变。

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



图 3.3 新建文档界面

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

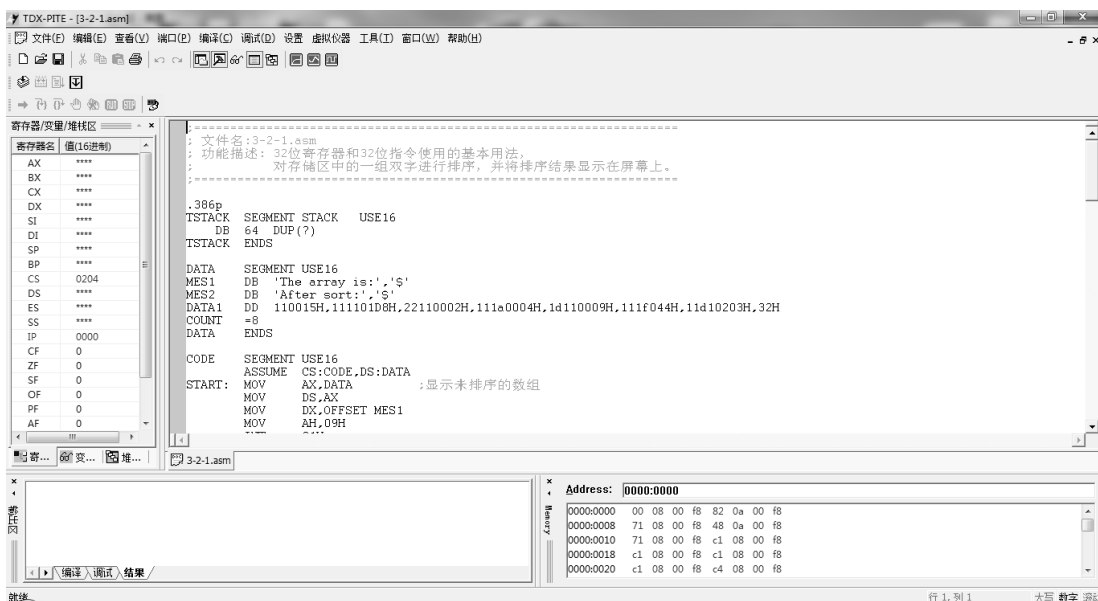




图 3.4 保存文件图

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

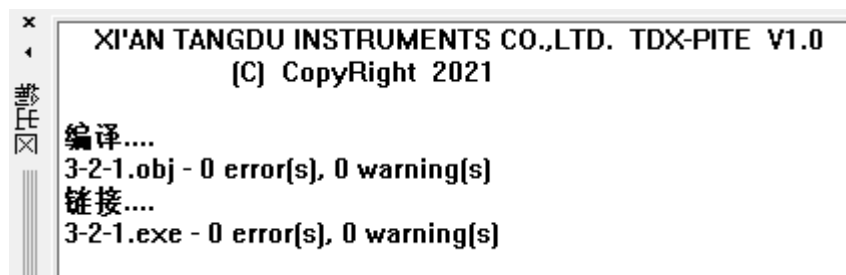




图 3.5 编译链接信息界面

(7) 连接 PC 与实验系统的通讯电缆，打开实验系统电源。

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

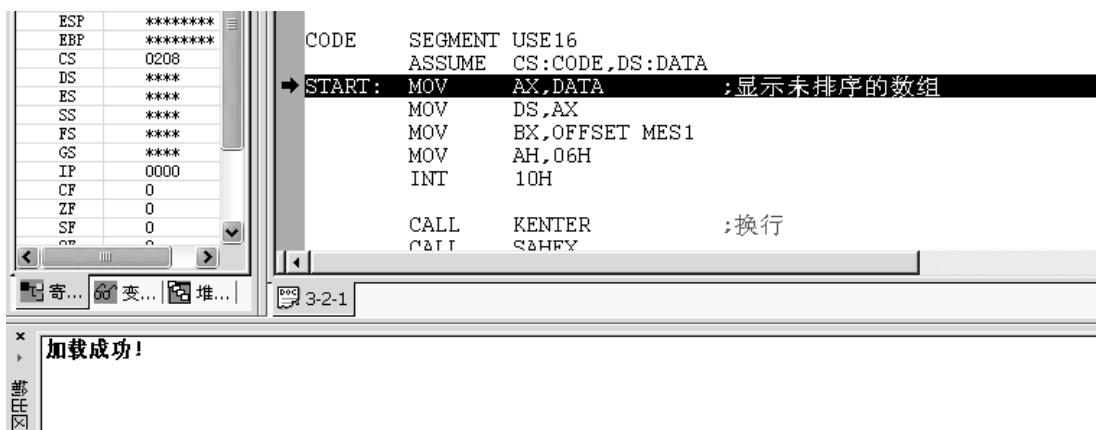



图 3.6 开始运行界面图

(9) 点击按钮运行程序，待程序运行停止后，观察运行结果。如图 3.7：

程序运行结果为：

The array is:

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

After sort:

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

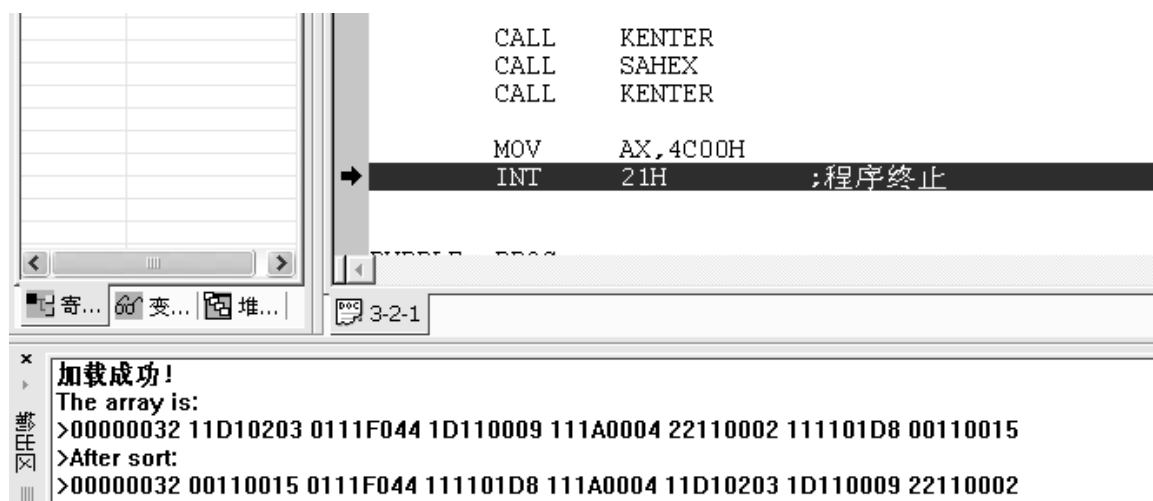


图 3.7 程序运行结果

3.2 32 位寄存器和 32 位指令使用：ASCII 转换 16 进制

3.2.1 实验目的

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

3.2.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

3.2.3 实验内容

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

1. 实验步骤

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

在输出区的结果栏将会显示：

This is tangdu speaking!

Show this sentence as hex:

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

(四) 总线地址译码及编程实验

4.1 总线地址译码及时序观测与分析实验

4.1.1 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。
3. 通过观测时序图，掌握地址总线、数据总线及控制总线的时序关系。

4.1.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

4.1.3 实验内容

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

4.1.4 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 4.1.1 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 4.1.1 所示。

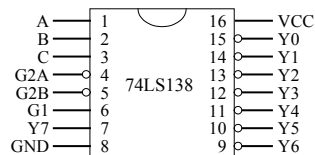


图 4.1.1 74LS138 译码器管脚

表 4.1.1 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

16 位总线的地址是由 XA1 开始, 所以地址是以 2 字节边界对齐的。实验平台上提供的每一个可扩展 I/O 片选 (IOY0~3) 其对应的 I/O 地址空间共有 256 字节, 偏移地址一般为 00H~FFH。设计地址译码电路, 主要是针对 XA7 以下低位地址线译码, 比如本实验要求通过译码得到偏移在 0638H~063FH 之间的端口, 然后用译码输出作为 I/O 接口单元的片选。因为 0638H~063FH 编址空间在 IOY0 范围内, 所以将 IOY0 作为地址译码单元的使能信号 G2B, 以保证在 0600H~063FH 空间内译码, 编写程序完成 I/O 读写数据操作。138 芯片译码原理图如图 4.1.2 所示。

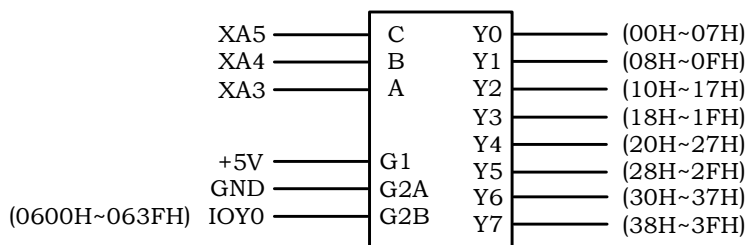


图 4.1.2 138 芯片译码原理图

4.1.5 实验步骤

(1) 按图 4.1.3 连接实验线路。

(2) 编写实验程序 (例程文件名为: 138.ASM), 单击“编译链接及加载”按钮将程序加载至单板机系统的 SRAM 中。

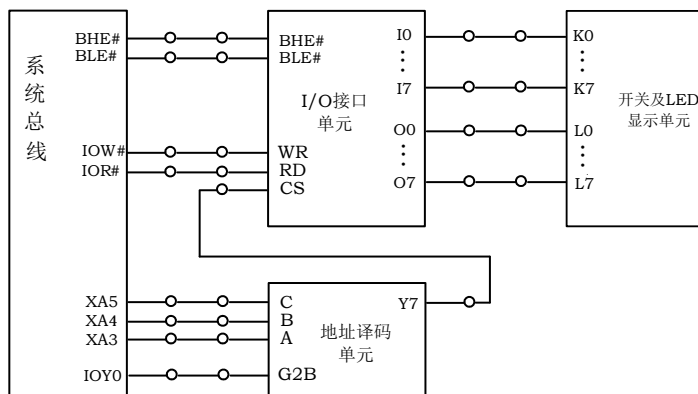




图 4.1.3 地址译码设计实验参考接线图

(3) 单击  按钮打开“时序观测窗”, 在窗内右键调出“选择观察信号”界面, 勾选要观测的信号“IOY0、IOW#、IOR#、XA19..0、XD7..0”, 点击确定。

(4) 拨动开关 K7..K0 为 12H, 即 0001 0010B。

(5) 单击  “单步”按钮两次，“时序观测窗”显示结果如图 4.1.4 所示，通过时序图可以看出 CPU 在 IOY0 片选有效期间，对 0638H 地址进行了 I/O 读操作，读取了上一步开关置的 12H 数据。

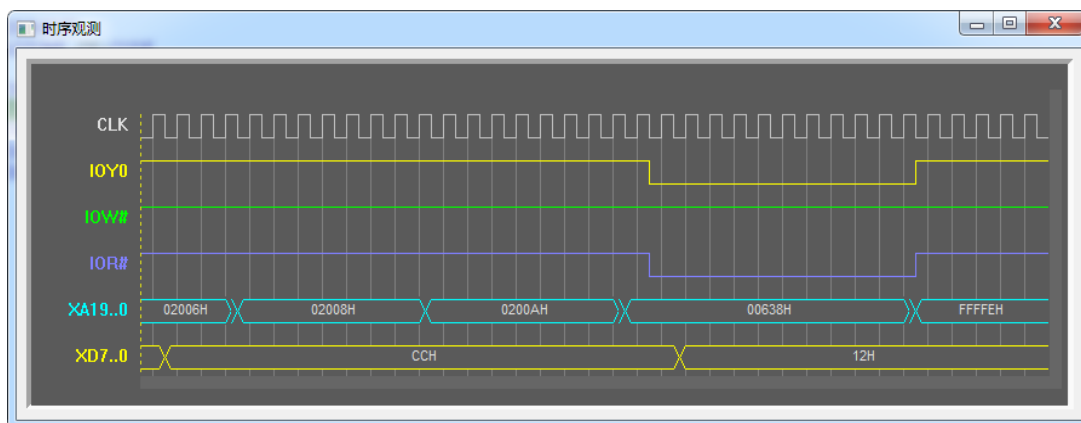



图 4.1.4 地址译码设计实验时序观测图

(6) 单击  “单步”按钮一次，“时序观测窗”显示结果如图 4.1.5 所示，通过时序图可以看出 CPU 在 IOY0 片选有效期间，对 0638H 地址进行了 I/O 写操作，写入了上一步读取的 12H 数据。

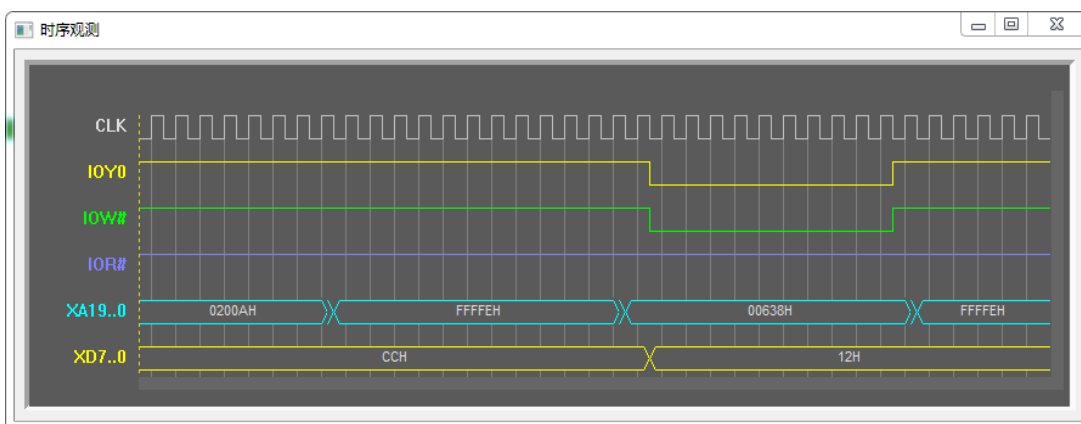


图 4.1.5 地址译码设计实验时序观测图

(7) 观看数据灯显示与开关是否一致。

（五）存储器管理及编程实验

5.1 静态存储器扩展及时序观测与分析实验

5.1.1 实验目的

1. 了解存储器扩展的方法和存储器的读/写。
2. 掌握 CPU 对 16 位存储器的访问方法。
3. 通过观测时序图，掌握存储器读写的时序关系。

5.1.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

5.1.3 实验内容

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

5.1.4 实验原理

存储器是用来存储信息的部件，是计算机的重要组成部分，静态 RAM 是由 MOS 管组成的触发器电路，每个触发器可以存放 1 位信息。只要不掉电，所储存的信息就不会丢失。因此，静态 RAM 工作稳定，不要外加刷新电路，使用方便。但一般 SRAM 的每一个触发器是由 6 个晶体管组成，SRAM 芯片的集成度不会太高，目前较常用的有 6116 (2K×8 位)，6264 (8K×8 位) 和 62256 (32K×8 位)。本实验平台上选用的是 6264，两片组成 8K×16 位的形式，共 16K 字节。

6264 的外部引脚图如图 5.1.1 所示。

本系统采用准 32 位 CPU，具有 16 位外部数据总线，即 D0、D1、…、D15，地址总线为 BHE #（# 表示该信号低电平有效）、BLE #、A1、A2、…、A19。存储器分为奇体和偶体，分别由字节允许线 BHE # 和 BLE # 选通。

存储器中，从偶地址开始存放的字称为规则字，从奇地址开始存放的字称为非规则字。处理器访问规则字只需要一个时钟周期，BHE # 和 BLE # 同时有效，从而同时选通存储器奇体和偶体。处理器访问非规则字却需要两个时钟周期，第一个时钟周期 BHE # 有效，访问奇字节；第二个时钟周期 BLE # 有效，访问偶字节。处理器访问字节只需要一个时钟周期，视其存放单元为奇或偶，而 BHE # 或 BLE # 有效，从而选通奇体或偶体。写规则字和非规则字的简单时序图如图 5.1.2 所示。

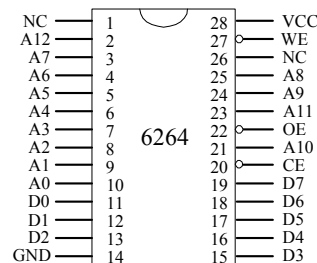


图 5.1.1 6264 引脚图

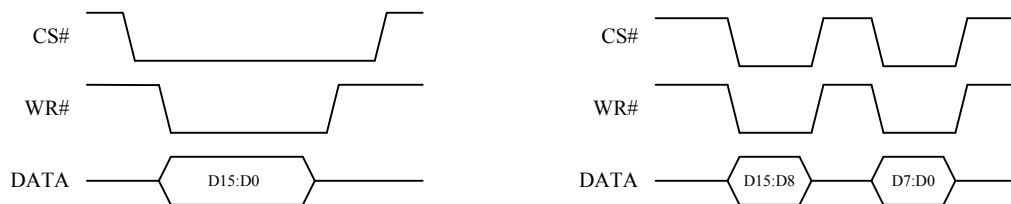


图 5.1.2 写规则字（左）和非规则字（右）简单时序图

实验单元电路如图 5.1.3 所示。

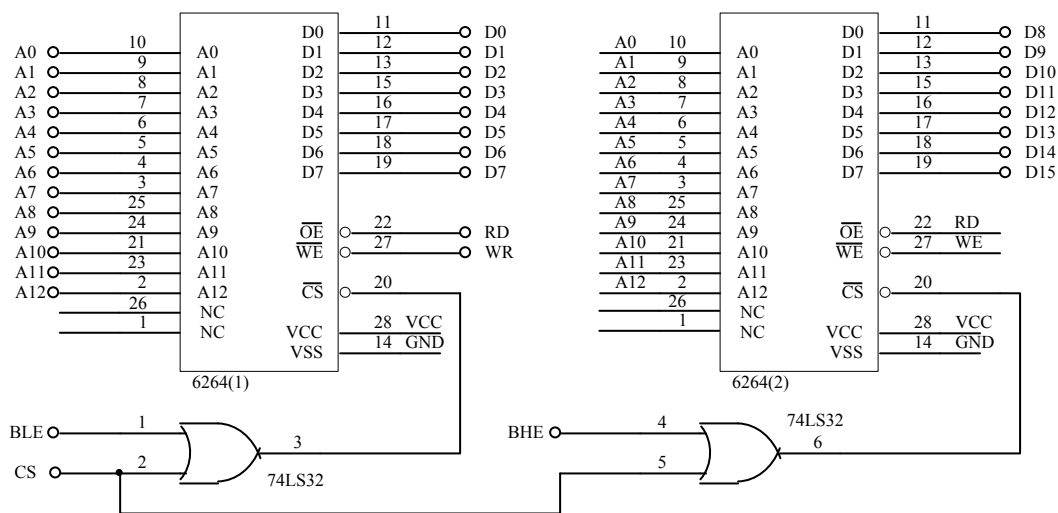


图 5.1.3 SRAM 单元电路图

5.1.5 实验步骤

(注：本章实验选择 16 位寄存器)

1. 实验接线图如图 5.1.4 所示，按图接线。

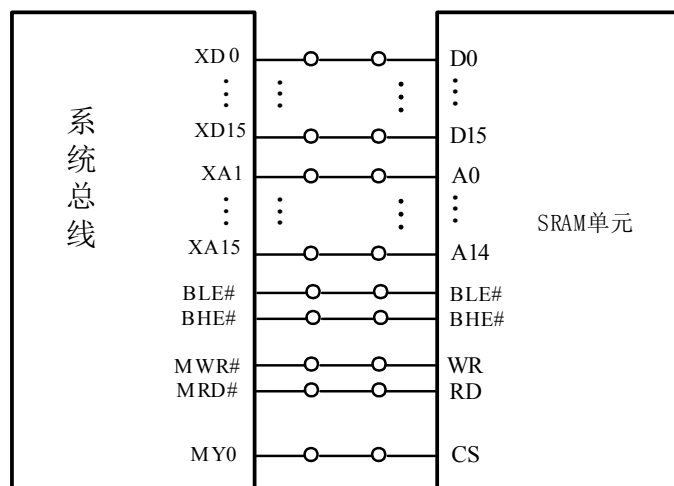






图 5.1.4 SRAM 实验接线图

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

3. 单击  按钮打开“时序观测窗”，在窗内右键调出“选择观察信号”界面，勾选要观测的信号“MY0、BHE#、BLE#、MWR#、MRD#、XA19..0、XD15..8、XD7..0”，点击确定。

4. 鼠标单击程序中“AA1: MOV [SI], AX”，单击 ，在此处设置断点。

5. 单击  程序运行至断点处停止，再单击  单步运行。

6. 点开“时序观测窗”，结果如图 5.1.5 所示。

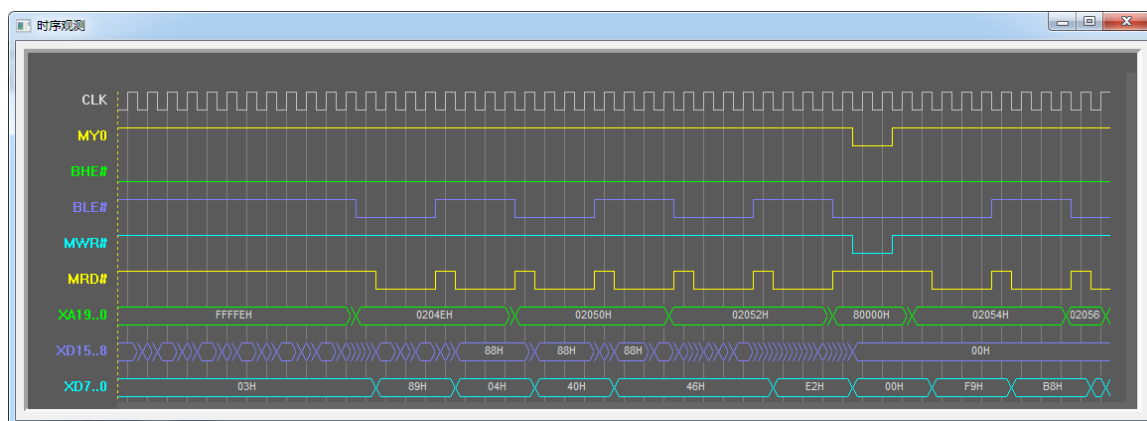




图 5.1.5 SRAM 实验写规则字时序观测图

7. 从图中可以看出存储器片选 MY0 有效期间，CPU 向“8000:0000H”地址进行存储器写操作，写入 00H。

8. 再单击  程序运行至断点处停止，单击  单步运行。可观察到存储器片选 MY0 有效

期间，CPU 向“8000:0002H”地址进行存储器写操作，写入 01H。结果如图 5.1.6 所示。

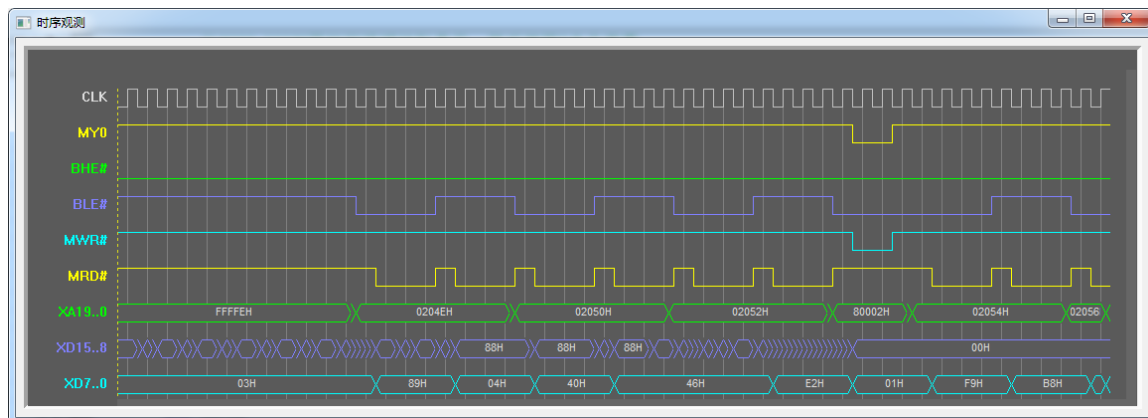




图 5.1.6 SRAM 实验写规则字时序观测图

9. 在“Memory”区查询“8000:0000H”地址和“8000:0002H”地址的数据，是否正确。

10. 单击  清除断点，点击  运行至程序结束。在“Memory”区查询“8000:0000H”地址至“8000:001FH”地址的数据，是否正确。

11. 改变实验程序，按非规则字写存储器，观察实验结果，如图 5.1.7。

给 SI 寄存器赋奇地址数，

MOV SI,0001H

即为非规则字写存储器。

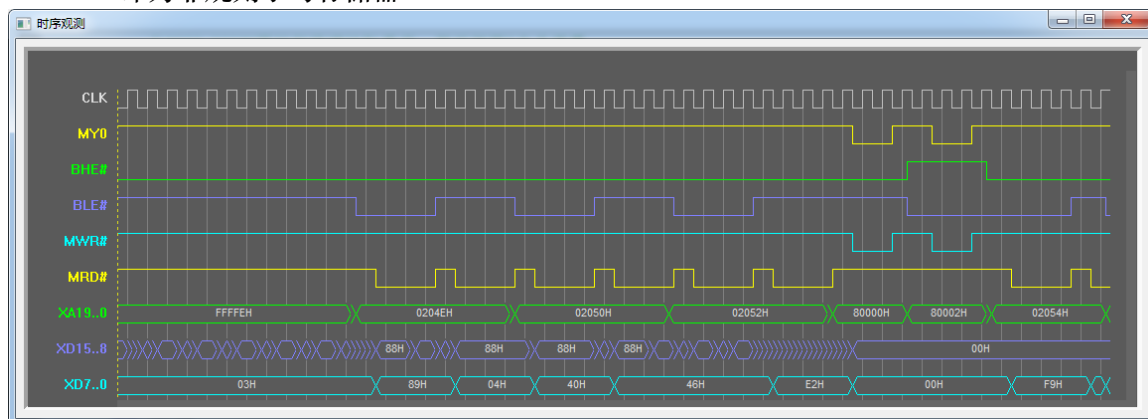


图 5.1.7 SRAM 实验写非规则字时序观测图

12. 改变实验程序，按字节方式写存储器，观察实验现象。

```
AA0:  MOV SI, 0000H
      MOV CX, 0010H
      MOV AL, 00H
```

```
AA1:  MOV [SI], AL
      INC AL
      INC SI
      LOOP AA1
```


(六) PIC 中断管理及 8259 编程实验

6.1 8259 单一中断源控制实验

6.1.1 实验目的

1. 掌握 8259 中断控制器的工作原理。
2. 学习 8259 的应用编程方法。

6.1.2 实验设备

PC 机一台, TDX-PITE 实验装置一套。

6.1.3 实验内容

利用系统总线上中断请求信号 MIR7, 设计一个单一中断请求实验。

6.1.4 实验原理

1. 中断控制器 8259 简介

在 Intel 386EX 芯片中集成有中断控制单元 (ICU), 该单元包含有两个级联中断控制器, 一个为主控制器, 一个为从控制器。该中断控制单元就功能而言与工业上标准的 82C59A 是一致的, 操作方法也相同。本书实验只用到 i386EX 芯片中的主片 8259 外部中断控制器。

主片 8259 的 IR4 供系统串口使用, 8259 的内部连接及外部管脚引出如图 6.1.1:

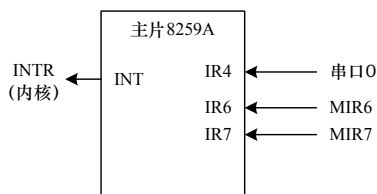


图 6.1.1 8259 内部连接及外部管脚引出图

表 6.1.1 列出了中断控制单元的寄存器相关信息。

表 6.1.1 ICU 寄存器列表

寄存器	口地址	功能描述
ICW1 (主) ICW1 (从) (只写)	0020H 00A0H	初始化命令字 1: 决定中断请求信号为电平触发还是边沿触发。
ICW2 (主) ICW2 (从) (只写)	0021H 00A1H	初始化命令字 2: 包含了 8259 的基址中断向量号, 基址中断向量是 IR0 的向量号, 基址加 1 就是 IR1 的向量号, 依此类推。
ICW3 (主) (只写)	0021H	初始化命令字 3: 用于识别从 8259 设备连接到主控制器的 IR 信号, 内部的从 8259 连接到主 8259

		的 IR2 信号上。
ICW3 (从) (只写)	00A1H	初始化命令字 3: 表明内部从控制器级联到主片的 IR2 信号上。
ICW4 (主) ICW4 (从) (只写)	0021H 00A1H	初始化命令字 4: 选择特殊全嵌套或全嵌套模式, 使能中断自动结束方式。
OCW1 (主) OCW1 (从) (读/写)	0021H 00A1H	操作命令字 1: 中断屏蔽操作寄存器, 可屏蔽相应的中断信号。
OCW2 (主) OCW2 (从) (只写)	0020H 00A0H	操作命令字 2: 改变中断优先级和发送中断结束命令。
OCW3 (主) OCW3 (从) (只写)	0020H 00A0H	操作命令字 3: 使能特殊屏蔽方式, 设置中断查询方式, 允许读出中断请求寄存器和当前中断服务寄存器。
IRR (主) IRR (从) (只读)	0020H 00A0H	中断请求: 指出挂起的中断请求。
ISR (主) ISR (从) (只读)	0020H 00A0H	当前中断服务: 指出当前正在被服务的中断请求。
POLL (主) POLL (从) (只读)	0020H 0021H 00A0H 00A1H	查询状态字: 表明连接到 8259 上的设备是否需要服务, 如果有中断请求, 该字表明当前优先级最高的中断请求。

初始化命令字 1 寄存器 (ICW1) 说明见图 6.1.2 所示。

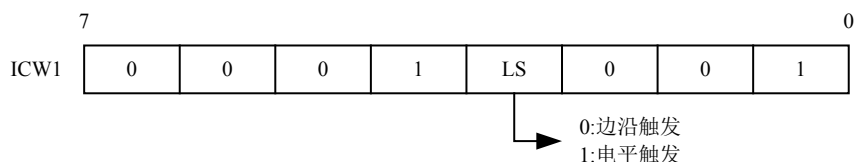


图 6.1.2 初始化命令字 1 寄存器

初始化命令字 2 寄存器 (ICW2) 说明见图 6.1.3 所示。

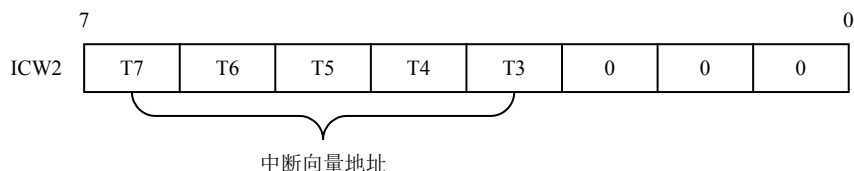


图 6.1.3 初始化命令字 2 寄存器

初始化命令字 3 寄存器 (ICW3) 说明, 主片见图 6.1.4, 从片见图 6.1.5。

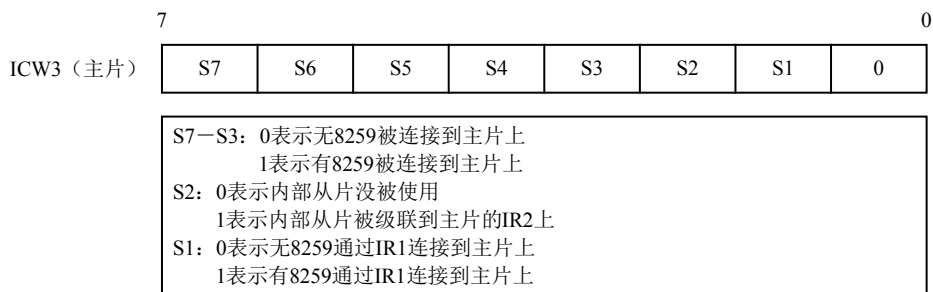


图 6.1.4 主片初始化命令字 3 寄存器

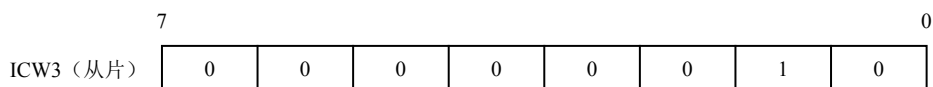


图 6.1.5 从片初始化命令字 3 寄存器

初始化命令字 4 寄存器 (ICW4) 说明见图 6.1.6。

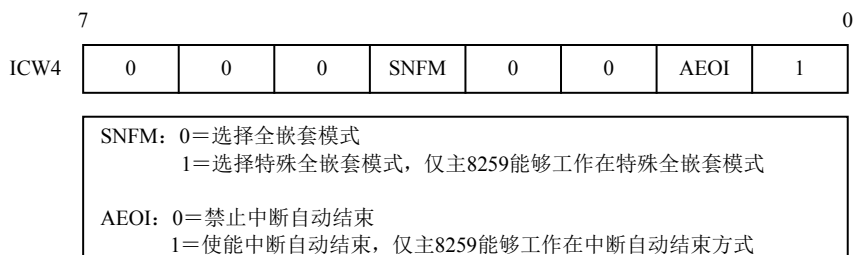


图 6.1.6 初始化命令字 4 寄存器

操作命令字 1 寄存器 (OCW1) 说明见图 6.1.7。

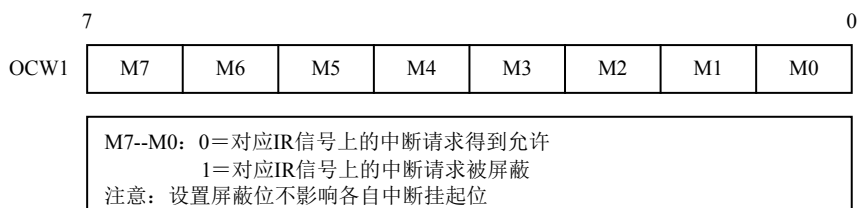


图 6.1.7 操作命令字 1 寄存器

操作命令字 2 寄存器 (OCW2) 说明如图 6.1.8 所示。

	7							0
OCW2	R	SL	EOI	0	0	L2	L1	L0

R SL EOI 命令

0 0 0 取消优先级自动循环 *

0 0 1 一般的中断结束命令

0 1 0 无操作

0 1 1 特殊的中断结束命令 **

1 0 0 中断优先级自动循环 *

1 0 1 在一般中断结束方式中优先级循环

1 1 0 优先级特殊循环方式 **

1 1 1 在特殊中断结束方式中优先级循环 **

* 当8259工作在中断自动结束方式下时，这些情况可以改变优先级结构。

** 在这些情况下优先级由L2:L0指定。

L2、L1、L0：在给出特殊的中断结束命令时，L2、L1、L0指出了具体要清除当前中断服务寄存器的哪一位；当给出特殊的优先级循环方式命令时，L2、L1、L0指出了循环开始时哪个中断的优先级最低。

图 6.1.8 操作命令字 2 寄存器

操作命令字 3 寄存器（OCW3）说明如图 6.1.9 所示。

	7							0
OCW3	0	ESMM	SMM	0	1	P	RR	RIS

ESMM SMM

0 0 无影响

0 1 无影响

1 0 禁止特殊屏蔽模式

1 1 使能特殊屏蔽模式

P：设置该位使8259工作在中断查询方式

RR RIS

0 0 无影响

0 1 无影响

1 0 读中断请求寄存器IRR

1 1 读当前中断服务寄存器ISR

图 6.1.9 操作命令字 3 寄存器

查询状态字（POLL）说明如图 6.1.10 所示。

	7							0
POLL	INT	-	-	-	-	L2	L1	L0

INT：0=无请求

1=连接在8259上的设备请求服务

L2、L1、L0：当INT为1时，这些位指出了需要服务的最高优先级的IR；当INT为0时这些位不确定。

图 6.1.10 程序状态字寄存器

在对 8259 进行编程时，首先必须进行初始化。一般先使用 CLI 指令将所有的可屏蔽中断禁止，然后写入初始化命令字。8259 有一个状态机控制对寄存器的访问，不正确的初始化顺

序会造成异常初始化。在初始化主片 8259 时，写入初始化命令字的顺序是：ICW1、ICW2、ICW3、然后是 ICW4，初始化从片 8259 的顺序与初始化主片 8259 的顺序是相同的。

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

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

6.1.5 实验步骤

8259 单中断实验

实验接线图如图 6.1.11 所示，单次脉冲输出与主片 8259 的 IR7 相连，每按动一次单次脉冲，产生一次外部中断，在显示屏上输出一个字符“7”。

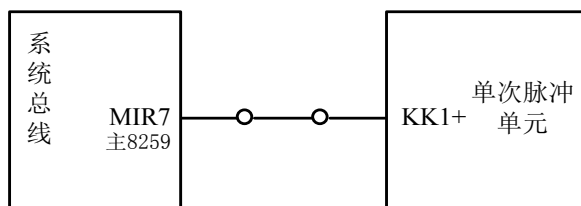



图 6.1.11 8259 单中断实验接线图

实验步骤

- (1) 按图 6.1.11 连接实验线路。
- (2) 编写实验程序（例程文件名为：A82591.ASM），经编译、链接无误后装入系统。
- (3) 单击  按钮，运行实验程序，重复按单次脉冲开关 KK1+，在界面的输出区会显示字符“7”，说明响应了中断。实验现象结果如图 6.1.12 所示。

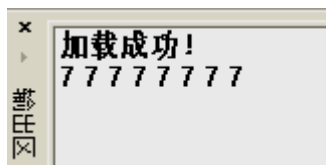


图 6.1.12 8259 单中断实验结果图

6.2 8259 优先级中断管理实验

6.2.1 实验目的

1. 掌握 8259 中断控制器的工作原理。
2. 学习 8259 的应用编程方法。

6.2.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

6.2.3 实验原理

8259 内部带有优先级排队电路，能对 8 个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 IRQ6 和 IRQ7 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 KK1+ 和 KK2+ 所对应的 IRQ6 和 IRQ7 分别设计两个中断服务，用单次脉冲 KK1 上升沿模拟 IRQ6 中断源，并在所对应的中断服务中，在屏幕上显示字符“6”。用单次脉冲 KK2 上升沿模拟 IRQ7 中断源，并在对应的中断服务中显示字符“7”。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。

实验接线图如图 6.2.1 所示，KK1+ 和 KK2+ 分别连接到主片 8259 的 IR6 和 IR7 上，当按一次 KK1+ 时，显示屏上显示字符“6”，按一次 KK2+ 时，显示字符“7”。编写程序。

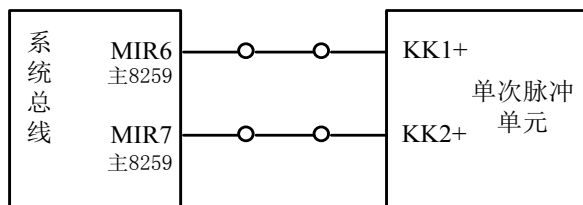



图 6.2.1 8259 优先级实验接线图

6.2.4 实验步骤

(1) 按图 6.2.1 连接实验线路。

(2) 编写实验程序（例程文件名为：A82592.ASM），经编译、链接无误后装入系统。

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

(4) 尝试先按 KK2+，再快速按 KK1+，观察 MIR6 和 MIR7 两个中断请求的优先级，分析实验结果。实验结果现象如图 6.2.2 所示。

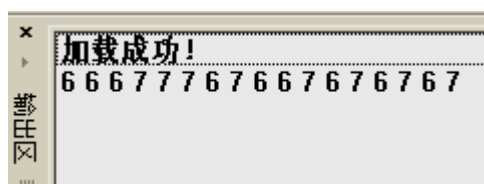


图 6.2.2 8259 双中断优先级实验结果图

6.3 8259 级连中断管理及时序观测与分析实验

6.3.1 实验目的

1. 掌握 8259 中断控制器的工作原理。
2. 学习 8259 的应用编程方法。
3. 掌握 8259 级联方式的使用方法。
4. 通过观测时序图，掌握中断请求、中断响应信号及中断矢量的时序关系。

6.3.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

6.3.3 实验原理

实验接线图如图 6.3.1 所示，实验平台上的 8259 作为从片，它通过系统总线 MIR7 请求线与 386 内部的主片 8259 进行级连，KK1+ 连接到从片 8259 的 IR0 上，KK2+ 连接到从片 8259 的 IR1 上，当按一次 KK1+ 时，显示屏上显示字符“S0”，按一次 KK2+ 时，显示字符“S1”。编写程序。中断的初始化及矢量地址可参考 6.1 章节。

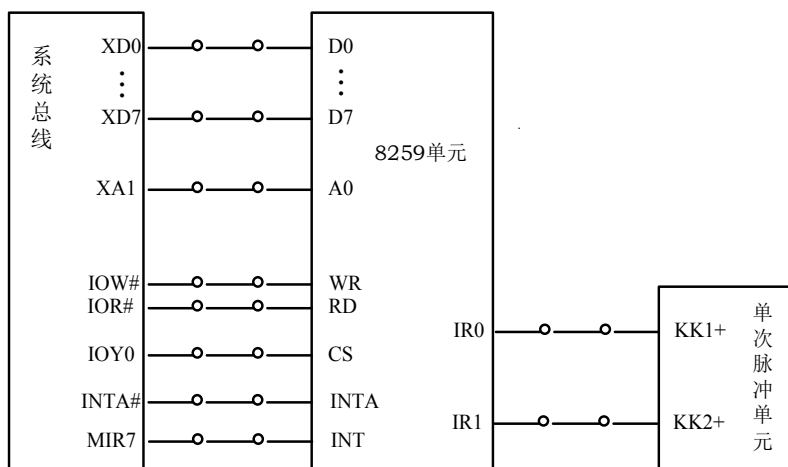





图 6.3.1 8259 级连实验接线图

6.3.4 实验步骤

- (1) 按图 6.3.1 连接实验线路。
- (2) 输入程序（例程文件名为：A82593.ASM），编译、链接无误后装入系统。

(3) 单击  按钮打开“时序观测窗”，在窗内右键调出“选择观察信号”界面，勾选要观测的信号“MIR7、INTA#、XD7..0”，点击确定。

(4) 单击  按钮，运行实验程序，按单次脉冲开关 KK1+，在界面的输出区会显示字符“S0”，说明 CPU 响应了从片 8259 的 IR0 中断。

(5) 单击  按钮，停止程序运行，观察时序信号如图 6.3.2 所示。从图中可知从片 8259 通过 MIR7 向 CPU 发送中断请求，然后 CPU 产生两个中断响应信号 INTA#，在第二个 INTA# 信号结束，从片 8259 通过低八位数据总线向 CPU 发送基址中断向量号 30H。

(6) 交替按单次脉冲开关 KK1+和 KK2+，验证实验程序的正确性，实验结果现象如图 6.3.3 所示。

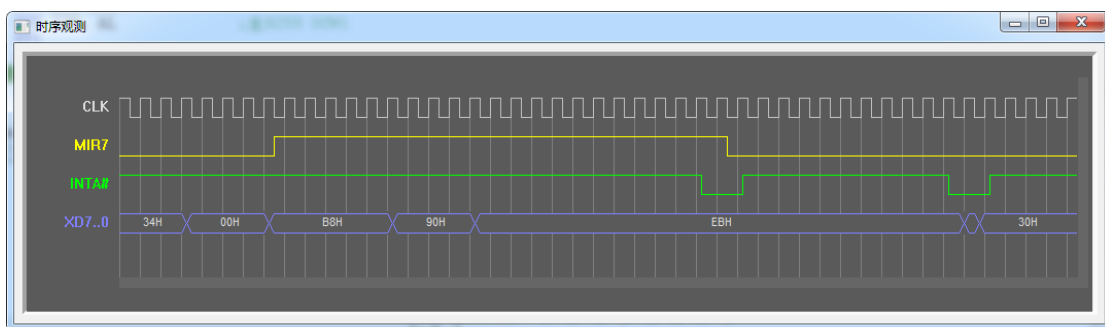


图 6.3.2 8259 级连时序观测图

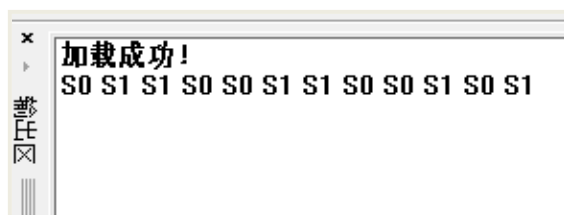


图 6.3.3 8259 级连实验结果图

(七) DMA 特性及 8237 编程实验

7.1 存储器到存储器 DMA 传送及时序观测与分析实验

7.1.1 实验目的

1. 掌握 8237DMA 控制器的工作原理。
2. 了解 DMA 特性及存储器到存储器的传输方式。
3. 掌握 8237 的应用编程。
4. 通过观测时序图，掌握 DMA 传送时各信号的时序关系。

7.1.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

7.1.3 实验内容

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

7.1.4 实验原理

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

1. 8237 芯片介绍

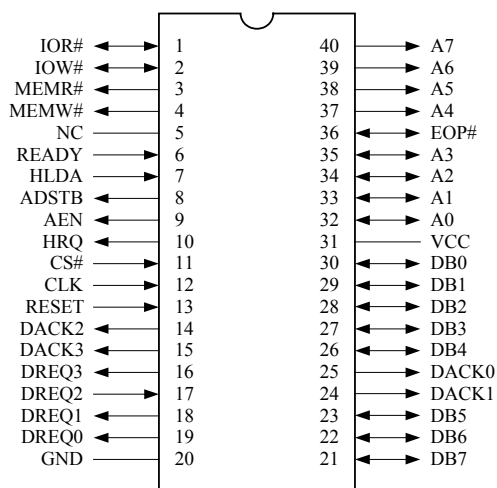


图 7.1.1 8237 外部引脚图

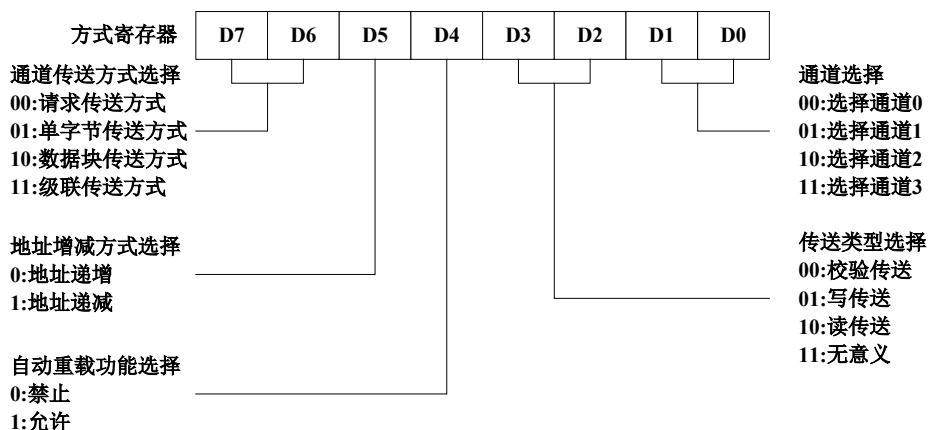


图 7.1.4 方式寄存器格式



图 7.1.5 8237 状态寄存器

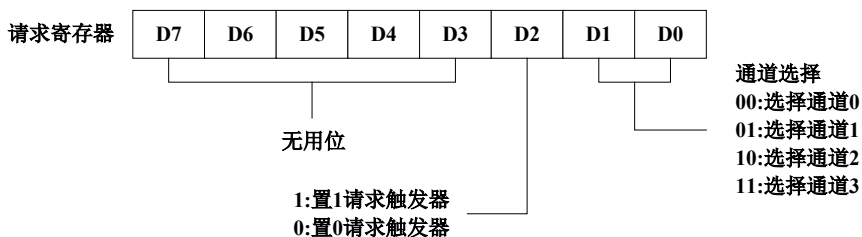
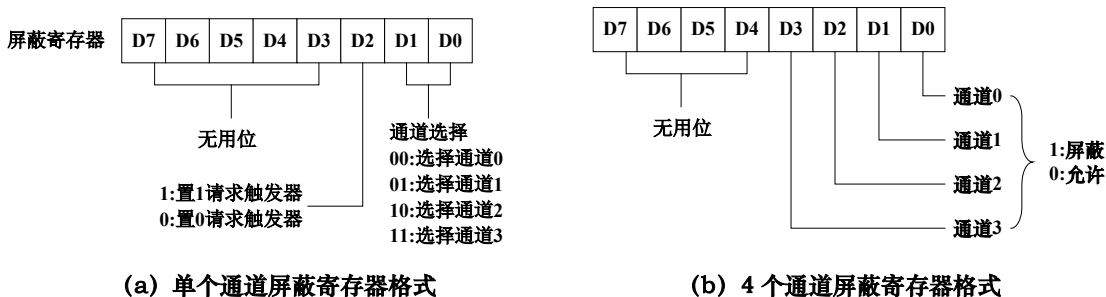


图 7.1.6 8237 请求寄存器格式



(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 7.1.7 通道屏蔽寄存器格式

表 7.1.1 列出了 8237 内部寄存器和软命令及其操作信息。

表 7.1.1 8237 内部寄存器和软命令及其读写操作一览表

寄存器名		位长	操作	片选逻辑 (CS # =0)				对 应 端口号	先/后 触发器	操作字节				
				IOR#	IOR#	A3	A2				A1	A0		
基地址寄存器 (4 个)		16	写	1	0	0	A2	A1	0					
当前地址寄存器 (4 个)		16	写			通道选择				0H	0	低 8 位		
			读	2H	1					高 8 位				
				0	4H					0	低 8 位			
					8H					1	高 8 位			
基字节数寄存器(4 个)		16	写	1	0	0	A2	A1	1					
当前字节数寄存器 (4 个)		16	写			通道选择				1H	0	低 8 位		
				3H	1					高 8 位				
			读	5H	0					低 8 位				
				7H	1					高 8 位				
命令寄存器		8	写	1	0	1	0	0	0	8H	-	-		
状态寄存器		8	读	0	1									
请求寄存器		4	写	1	0	1	0	0	1	9H				
写单个屏蔽位寄存器		4	写	1	0	1	0	1	0	AH				
方式寄存器 (4 个)		6	写	1	0	1	0	1	1	BH				
暂存寄存器		8	读	0	1	1	1	0	1	DH				
软 命 令	主清除	-	写	1	0					CH				
	清先/后触发器	-	写	1	0					EH				
	清屏蔽寄存器	-	写	1	0					FH				
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、存储器译码单元电路图

实验系统中提供的 8237 单元电路原理如图 7.1.8。

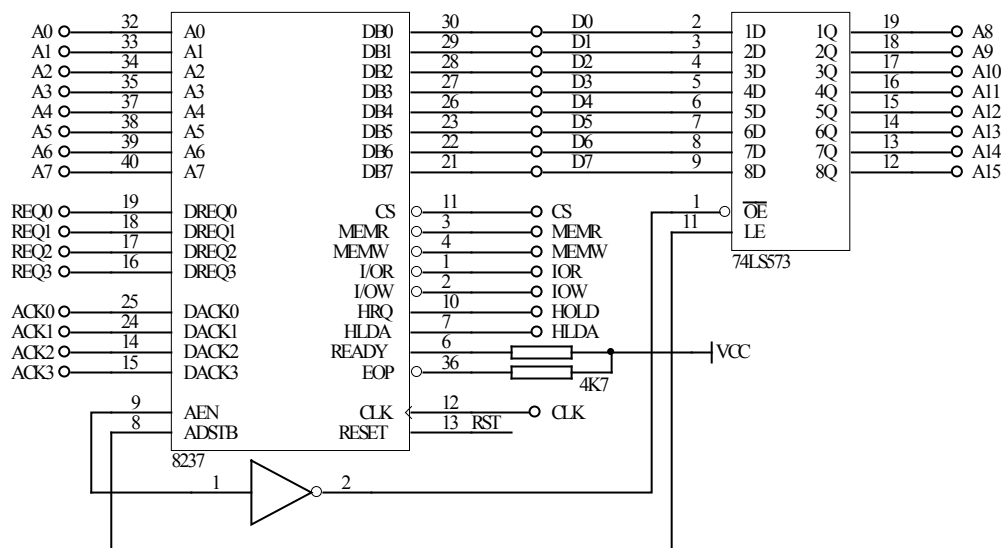


图 7.1.8 DMA 实验单元电路图

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

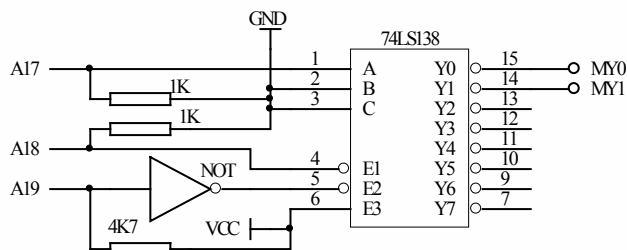


图 7.1.9 存储器译码单元电路图

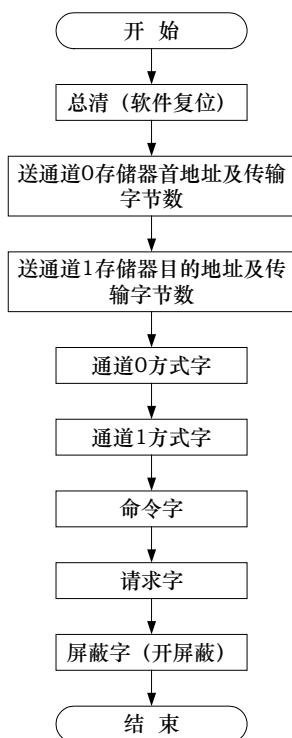


图 7.1.10 DMA 实验流程图

7.1.5 实验步骤

1. 存储器到存储器 DMA 传输实验

将存储器 1000H 单元开始的连续 10 个字节的数据复制到地址 0000H 开始的 10 个单元

中，实现 8237 的存储器到存储器传输。

(1) 根据实验要求，参考流程图 7.1.10 编写实验程序（例程文件名为：A82371.ASM）；实验接线如图 7.1.11 所示，按图连接实验线路。

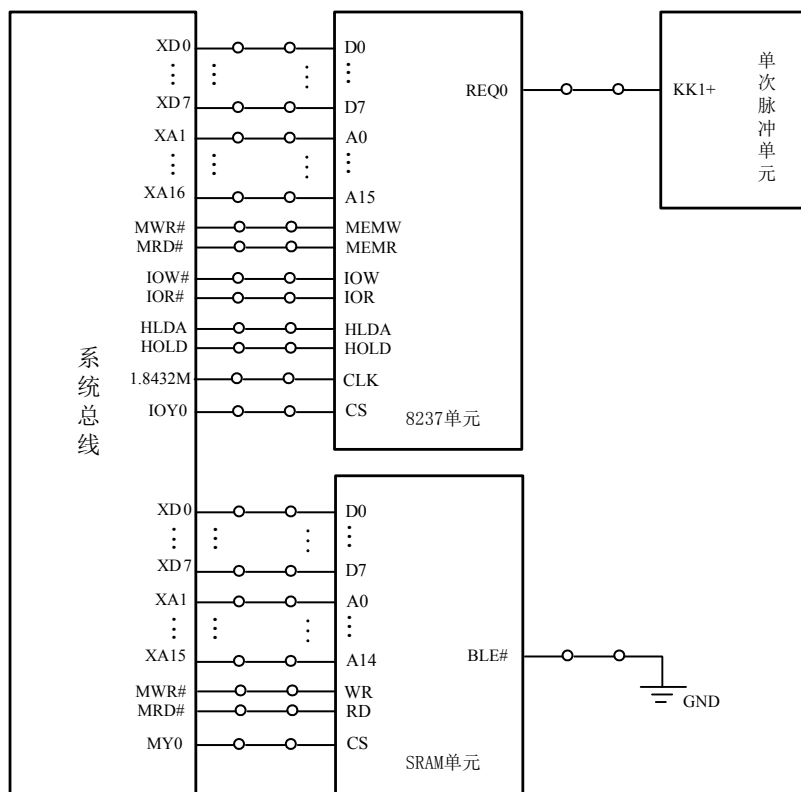


图 7.1.11 8237 实现存储器到存储器传输实验接线图

(2) 编译、链接程序无误后，将目标代码装入系统。

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

E8000:2000=11

E8000:2002=22

E8000:2004=33

E8000:2006=44

E8000:2008=55


E8000:200A=66


E8000:200C=77


E8000:200E=88

E8000:2010=99

E8000:2012=00

(4) 单击  按钮打开“时序观测窗”，在窗内右键调出“选择观察信号”界面，勾选要观测的信号“HOLD、HLDA、MY0、MWR#、MRD#、XA19..0、XD7..0”，点击确定。

(5) 单击  按钮，运行实验程序，按单次脉冲开关 KK1+ 产生 DMA 通道 0 的 REQ0 请求。

(6) 单击  按钮，停止程序运行，观察时序信号如图 7.1.12 所示。从图中可知当 DMA 请求信号 HOLD 有效后，CPU 响应 DMA 控制器 HLDA 有效，CPU 总线挂起，由 DMA 控制器完成从存储器到存储器的 DMA 传送。

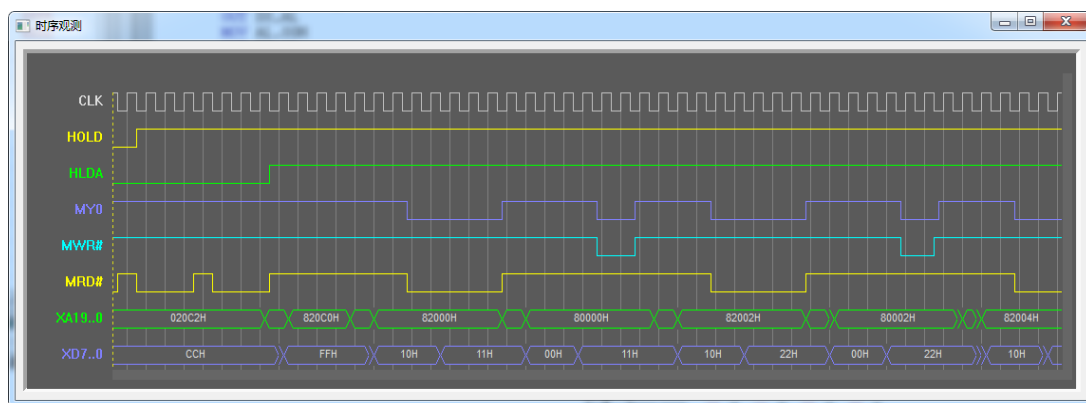


图 7.1.12 存储器到存储器 DMA 传送时序观测图

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

7.2 存储器到 I/O 设备 DMA 传送及时序观测与分析实验

7.2.1 实验目的

1. 掌握 8237DMA 控制器的工作原理。
2. 了解 DMA 特性及 I/O 到存储器、存储器到 I/O 的传输方式。
3. 掌握 8237 的应用编程。
4. 通过观测时序图，掌握 DMA 传送时各信号的时序关系。

7.2.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

7.2.3 实验内容

1. 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。
2. 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。

7.2.4 实验步骤

1. I/O 到存储器 DMA 传输实验

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

(1) 实验接线如图 7.2.1 所示，按图连接实验线路。

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

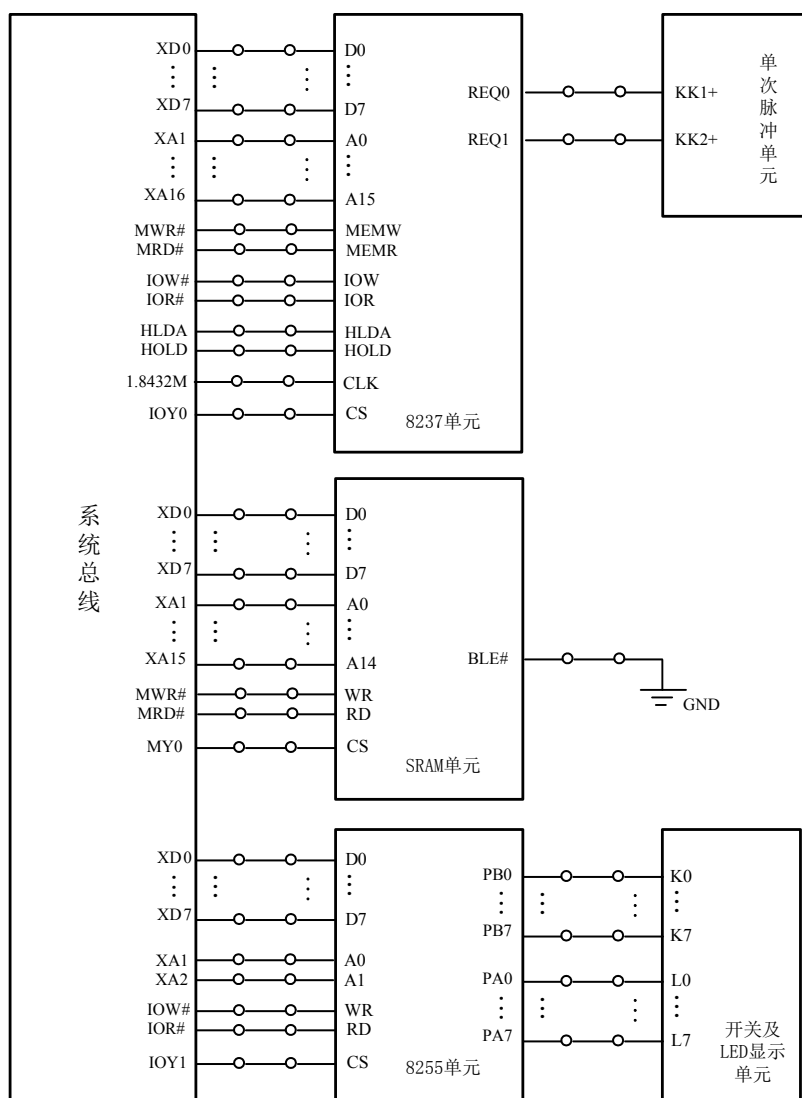





图 7.2.1 8237 实现存储器与 I/O 间 DMA 传输实验接线图

(3) 拨动 8255 的 B 口所连开关组，设置好一个数据。

(4) 单击  按钮打开“时序观测窗”，在窗内右键调出“选择观察信号”界面，勾选要观测的信号“HOLD、HLDA、IOY1、MY0、IOW#、IOR#、MWR#、MRD#、XA19..0、XD7..0”，点击确定。

(5) 单击  按钮，运行实验程序，按单次脉冲开关 KK2+ 产生 DMA 通道 1 的 REQ1 请求。

(6) 单击  按钮，停止程序运行，观察时序信号如图 7.2.2 所示。从图中可知当 DMA 请求信号 HOLD 有效后，CPU 响应 DMA 控制器 HLDA 有效，CPU 总线挂起，由 DMA 控制器完成从 I/O 到存储器的 DMA 传送。

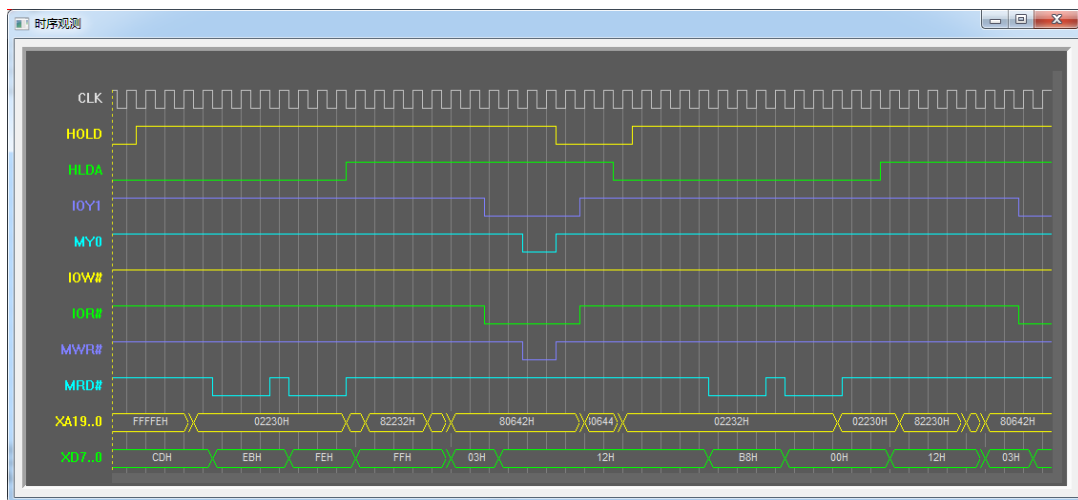


图 7.2.2 I/O 到存储器 DMA 传送时序观测图

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

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

2. 存储器到 I/O DMA 传输实验

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

(1) 实验接线如图 7.2.1 所示，按图连接实验线路。

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

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

(4) 运行程序，按动 KK1+产生 DMA 通道 0 的 REQ0 请求，停止程序。

(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。

(6) 可自行观察时序图，操作步骤同 I/O 到存储器 DMA 传输实验。

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

（八）扩展接口电路及编程实验

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。

8.1 基本 I/O 输入输出及时序观测与分析实验

8.1.1 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 / 16 位 I/O 端口的操作方法。
3. 通过观测时序图，掌握 I/O 操作时各信号的时序关系。

8.1.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

8.1.3 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. 利用两组三态缓冲器 245、锁存器 374 或 574 构成的 16 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。
3. 利用两组三态缓冲器 245、锁存器 374 或 574 构成的 16 位的 I/O 接口，按照 16 位的 I/O 操作方式，在开关及 LED 显示单元实现 16 位流水灯。

8.1.4 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 8.1.1 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。

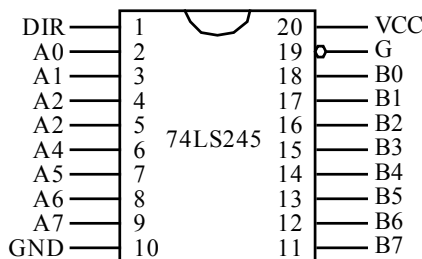


图 8.1.1 74LS245 双向三态缓冲器管脚图

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 8.1.2 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。

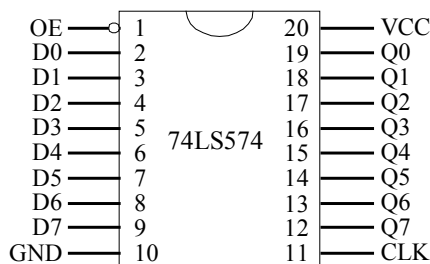


图 8.1.2 74LS574 上沿触发锁存器管脚图

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 8.1.3 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。

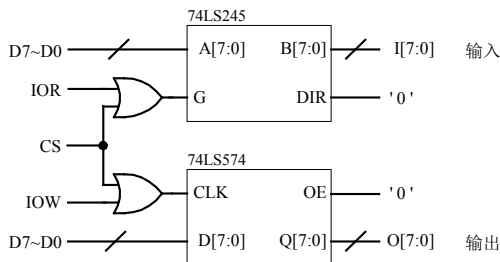


图 8.1.3 用 74LS245 和 74LS574 组成的 8 位 I/O 接口电路

4. 16 位 I/O 接口设计

用两组 8 位的 I/O 接口电路可以构成一个 16 位的 I/O 接口电路，可以一次进行 16 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 16 位数据总线，每个字节都对应着一位字节使能信号，共有两位字节使能信号 BHE、BLE，因此每个 8 位 I/O 接口电路是否有效要受 BHE、BLE 的控制。16 位 I/O 电路连接如图 8.1.4 所示。

IN AX, DX ;将 I[15:0]连接设备的 16 位数据通过数据总线 D[15:0]输入到 AX。

OUT DX, AX ;将 AX 中的数据通过数据总线 D[15:0]输出到 O[15:0]连接的设备。

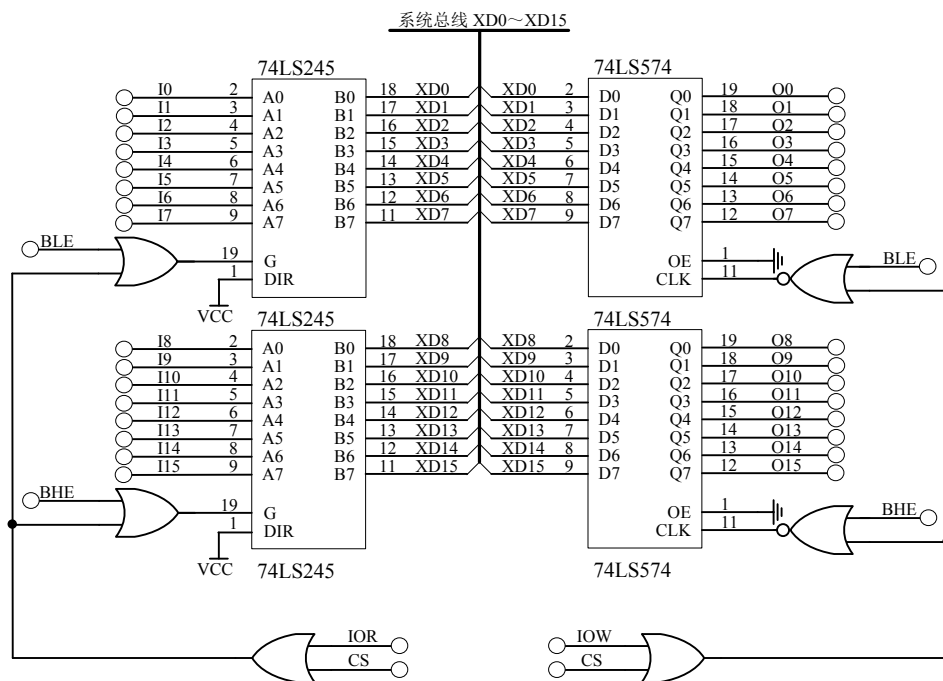


图 8.1.4 用两组 8 位 I/O 接口组成的 16 位 I/O 接口电路

8.1.5 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。

实验步骤如下。

(1) 实验接线图如图 8.1.5 所示，按图连接实验线路图。

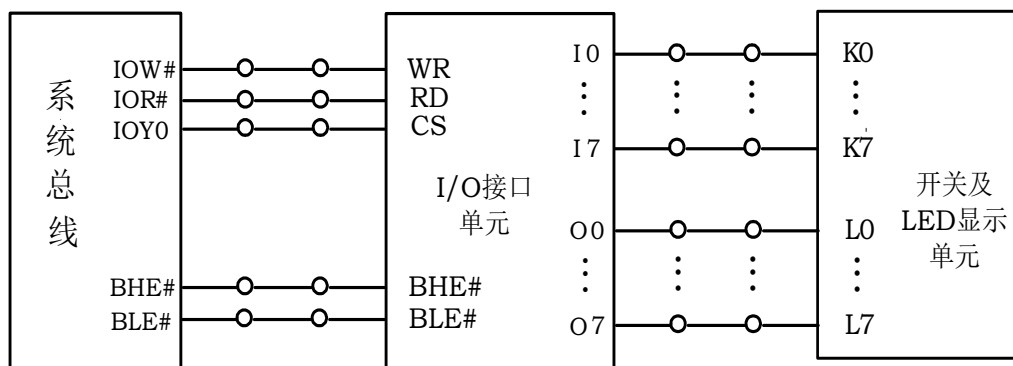




图 8.1.5 8 位 I/O 接口设计实验参考接线图

(2) 编写实验程序（例程文件名为：IO-8.ASM），经编译、连接无误后装入系统。

(3) 单击  按钮打开“时序观测窗”，在窗内右键调出“选择观察信号”界面，勾选要观测的信号“IOY0、IOW#、IOR#、XA19..0、XD7..0”，点击确定。

(4) 拨动开关 K7..K0 为 12H，即 0001 0010B。

(5) 单击  “单步”按钮两次，“时序观测窗”显示结果如图 8.1.6 所示，通过时序图可以看出 CPU 在 IOY0 片选有效期间，对 0600H 地址进行了 I/O 读操作，读取了上一步开关置的 12H 数据。

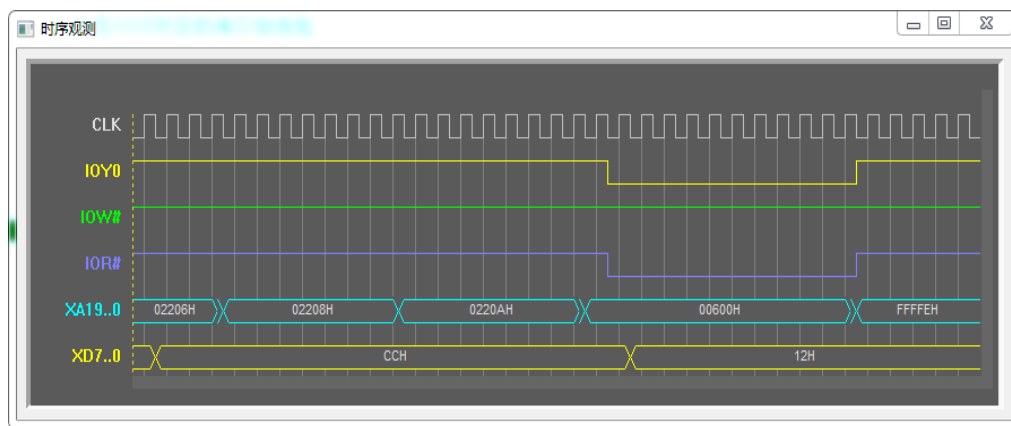



图 8.1.6 地址译码设计实验时序观测图

(6) 单击  “单步”按钮一次，“时序观测窗”显示结果如图 8.1.7 所示，通过时序图可以看出 CPU 在 IOY0 片选有效期间，对 0600H 地址进行了 I/O 写操作，写入了上一步读取的 12H 数据。

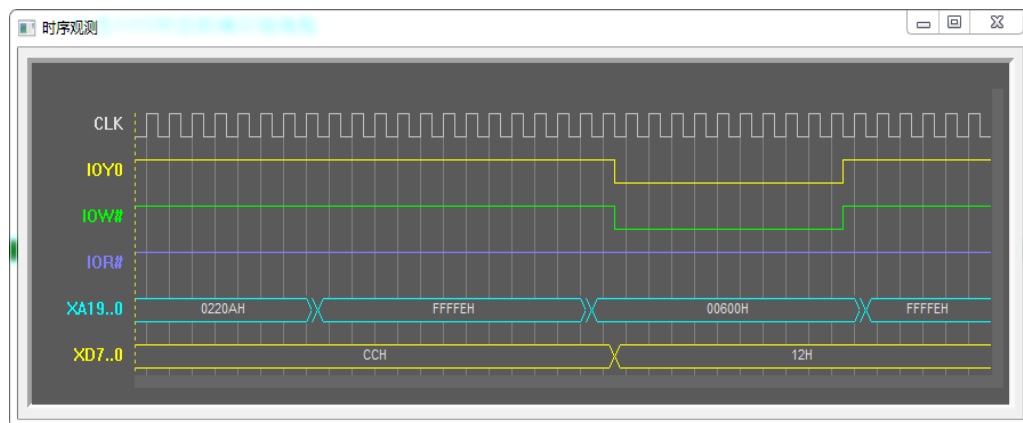


图 8.1.7 地址译码设计实验时序观测图

(7) 观看数据灯显示与开关是否一致。

2. 16 位 I/O 操作实验

本实验实现的是将开关 K[15:0]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。

实验步骤如下。

- (1) 实验接线图如图 8.1.8 所示,按图连接实验线路图。
- (2) 编写实验程序(例程文件名为:IO-16-1.ASM),经编译、连接无误后装入系统。
- (3) 运行程序,改变拨动开关,同时观察 LED 显示,验证程序功能。

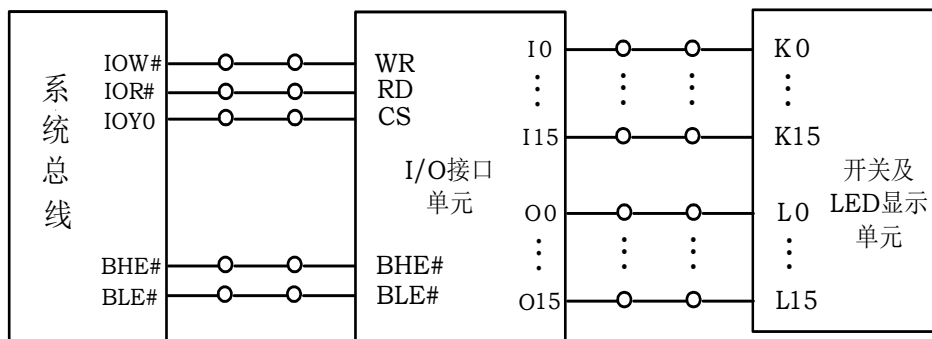


图 8.1.8 16 位 I/O 接口设计实验参考接线图

3. 16 位 I/O 流水灯实验

利用 16 位输出数据通道将数据总线的数输出到开关及 LED 显示单元显示。通过修改数据总线数据,实现流水灯输出显示。

实验步骤如下:

- (1) 实验接线图如图 8.1.9 所示，按图连接实验线路图。
- (2) 编写实验程序（例程文件名为：IO-16-2.ASM），经编译、连接无误后装入系统。
- (3) 运行程序，同时观察 LED 显示，验证程序功能。

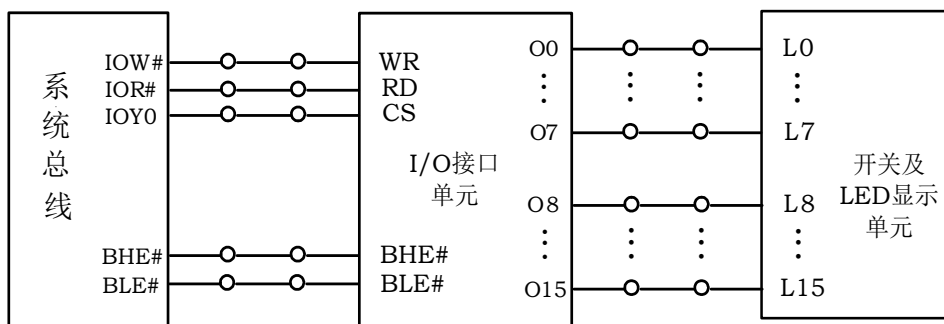


图 8.1.9 16 位 I/O 流水灯实验参考接线图

8.2 8255 并行接口实验

8.2.1 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。

8.2.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

8.2.3 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。

8.2.4 实验原理

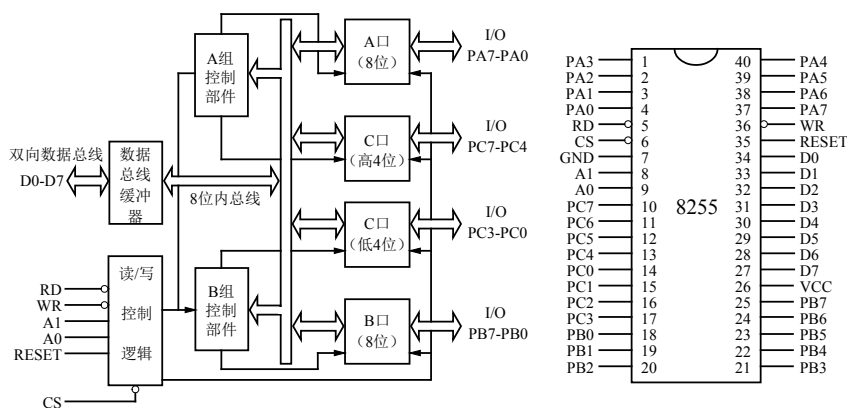


图 8.2.1 8255 内部结构及外部引脚图

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 8.2.1 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 8.2.2 所示。

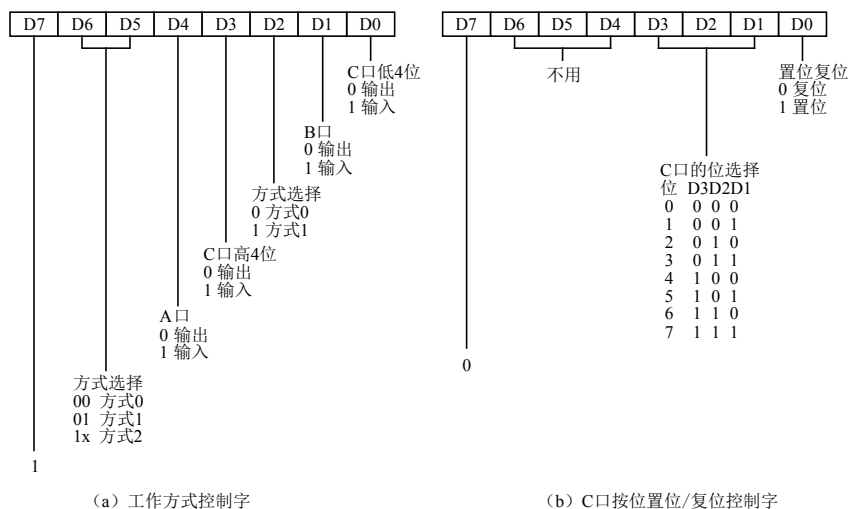


图 8.2.2 8255 控制字格式

8255 实验单元电路图如图 8.2.3 所示：

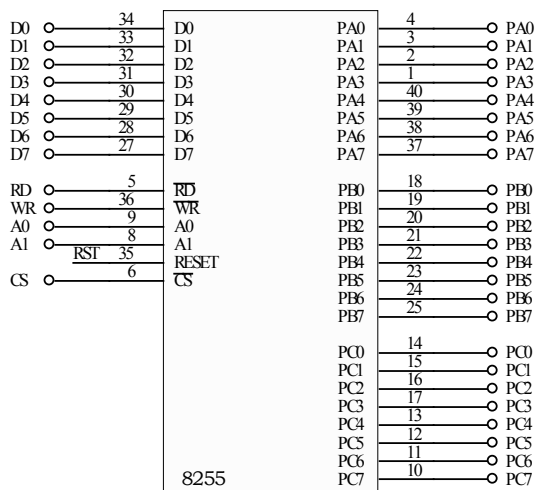


图 8.2.3 8255 实验单元电路图

8.2.5 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，端口 B 工作在方式 0 并作为输入。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：

- (1) 实验接线图如图 8.2.4 所示，按图连接实验线路图。
- (2) 编写实验程序（例程文件名为：A82551.ASM），经编译、连接无误后装入系统。
- (3) 运行程序，改变拨动开关，同时观察 LED 显示，验证程序功能。

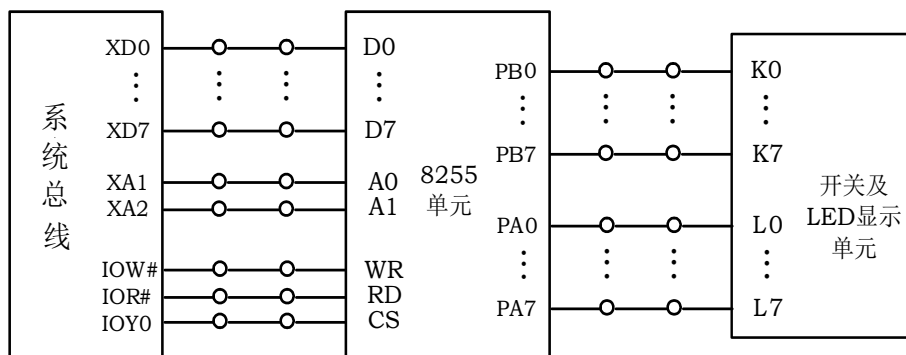


图 8.2.4 8255 基本输入输出实验接线图

2. 流水灯显示实验

使 8255 的 A 口和 B 口均为输出，数据灯 D7~D0 由左向右，每次仅亮一个灯，循环显示，D15~D8 与 D7~D0 正好相反，由右向左，每次仅点亮一个灯，循环显示。实验接线图如图 8.2.5 所示。实验步骤如下所述：

- (1) 按图 8.2.5 连接实验线路图。
- (2) 编写实验程序（例程文件名为：A82552.ASM），经编译、链接无误后装入系统。
- (3) 运行程序，观察 LED 灯的显示，验证程序功能。
- (4) 自己改变流水灯的方式，编写程序。

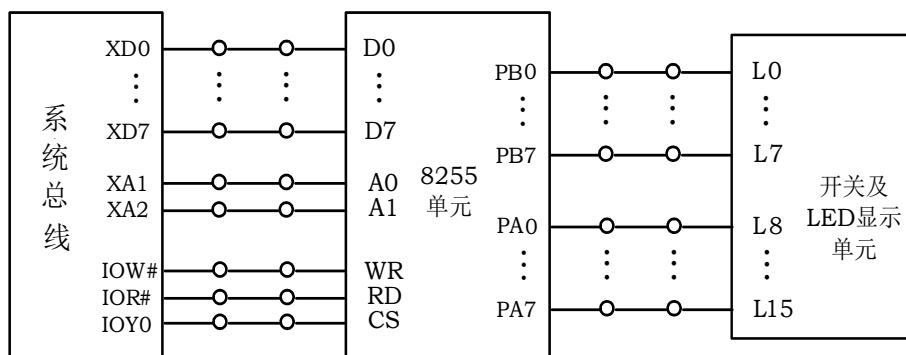


图 8.2.5 8255 流水灯实验接线图

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：

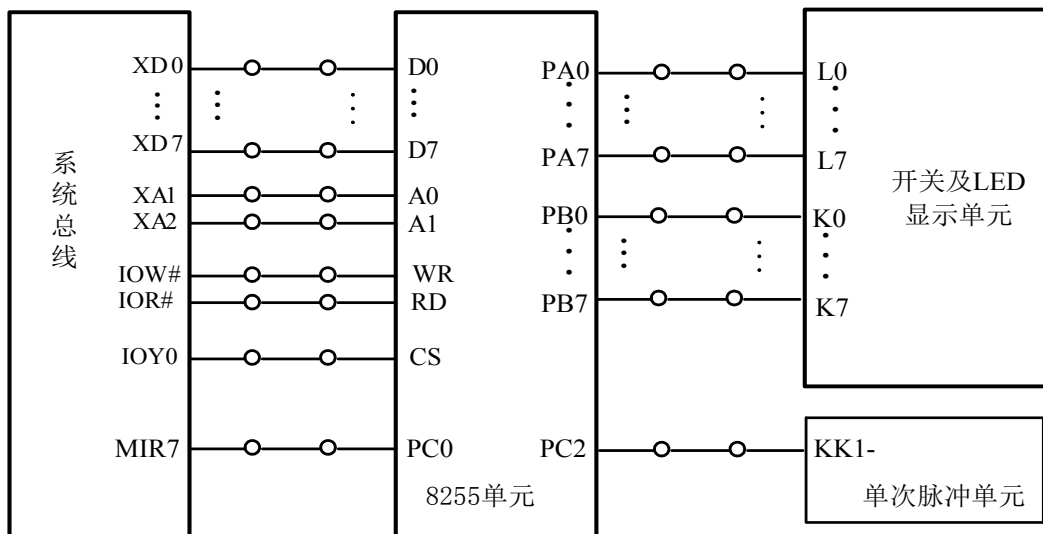


图 8.2.6 8255 方式 1 输入输出实验接线图

- (1) 按图 8.2.6 连接实验线路图。
- (2) 编写实验程序（例程文件名为：A82553.ASM），经编译、链接无误后装入系统。
- (3) 运行程序，然后改变拨动开关，准备好后，按动 KK1，同时观察数据灯显示，应与开关组信号一致。

8.3 8254 定时/计数器应用实验

8.3.1 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。

8.3.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

8.3.3 实验内容

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

8.3.4 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 8.3.1 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。

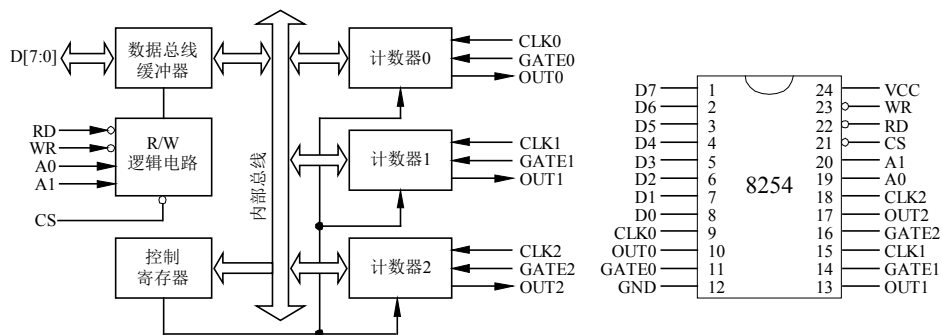


图 8.3.1 8254 的内部接口和引脚

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 8.3.1—8.3.3 所示。

表 8.3.1 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 8.3.2 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 8.3.3 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：

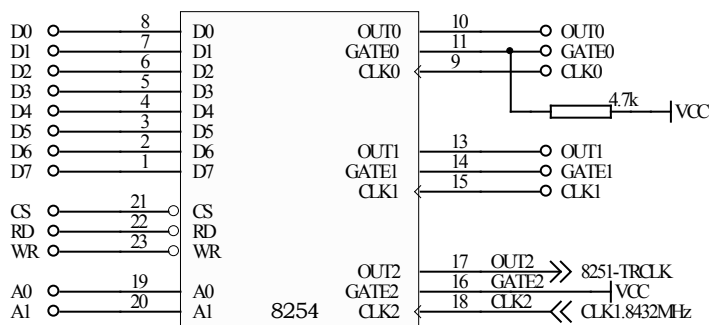


图 8.3.2 8254 实验电路原理图

8.3.5 实验步骤

1. 计数应用实验

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

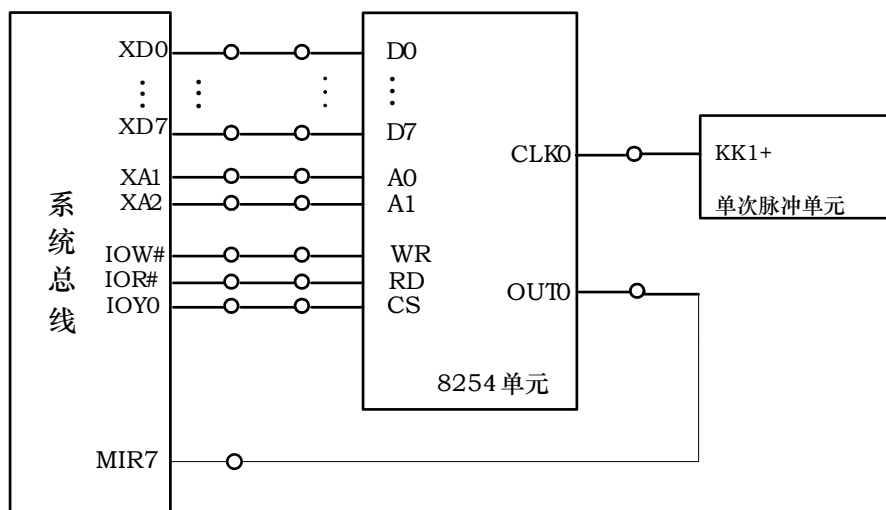



图 8.3.3 8254 计数应用实验接线图

实验步骤：

(1) 实验接线如图 8.3.3 所示(由于 8254 单元中 GATE0 信号已经上拉+5V, 所以 GATE0 不用接线)。

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

(3) 单击  按钮，运行实验程序，每连续按动 5 次 KK1+，在界面的输出区会显示字符“M”，观察实验现象。实验现象结果如图 8.3.4 所示。

(4) 改变计数值，验证 8254 的计数功能。



图 8.3.4 8254 计数实验结果图

2. 定时应用实验

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

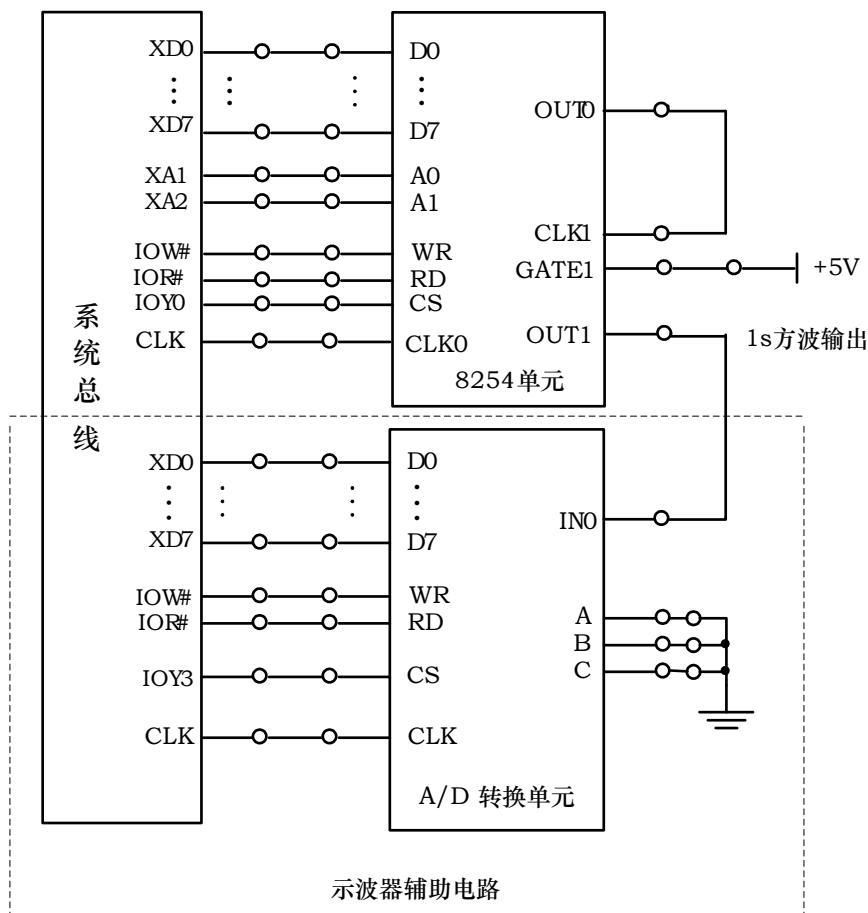






图 8.3.5 8254 定时应用实验接线图

实验步骤:

(1) 接线图如图 8.3.5 所示。

(2) 根据实验内容, 编写实验程序 (例程文件名为: A82542.ASM), 经编译、链接无误后装入系统。

(3) 单击  按钮，运行实验程序，8254 的 OUT1 会输出 1s 的方波，可用软件自带的示波器功能进行观察。

(4) 用示波器观察波形的方法：单击虚拟仪器菜单中的  示波器按钮或直接单击工具栏的  按钮，在新弹出的示波器界面上单击  按钮运行示波器，就可以观测出 OUT1 输出的波形。本实验现象结果如图 8.3.6 所示。

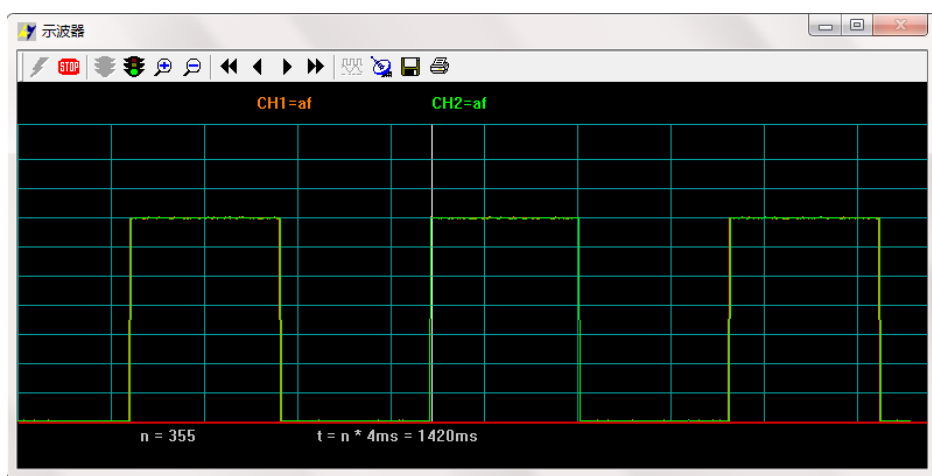


图 8.3.6 8254 定时应用实验结果图

8.4 8251 串行接口应用实验

8.4.1 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。

8.4.2 实验设备

PC 机一台, TDX-PITE 实验装置一套或两套。

8.4.3 实验内容

1. 数据信号的串行传输实验, 循环向串口发送一个数, 使用示波器测量 TXD 引脚上的波形, 以了解串行传输的数据格式。
2. 自收自发实验, 将 3000H 起始的 10 个单元中的初始数据发送到串口, 然后自接收并保存到 4000H 起始的内存单元中。

8.4.4 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 8.4.1 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 8.4.2 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。

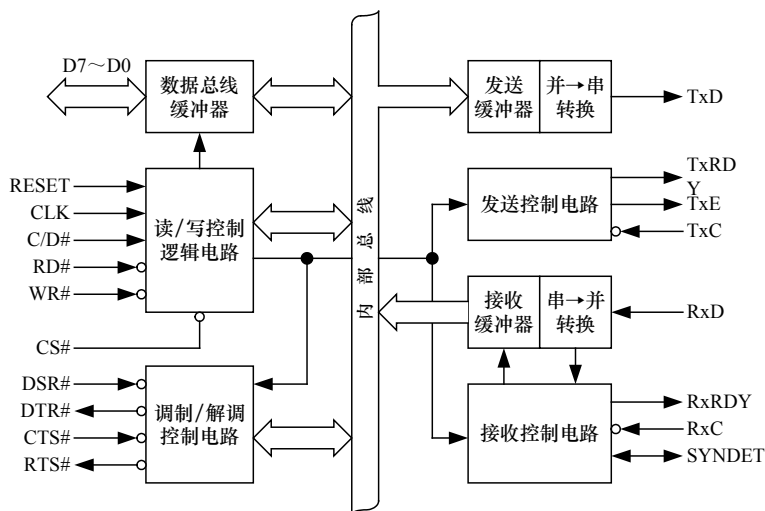


图 8.4.1 8251 内部结构图

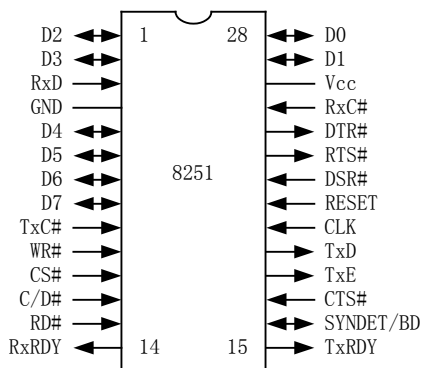


图 8.4.2 8251 外部引脚图

3. 8251 在异步方式下的 TXD 信号上的数据传输格式

图 8.4.3 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。

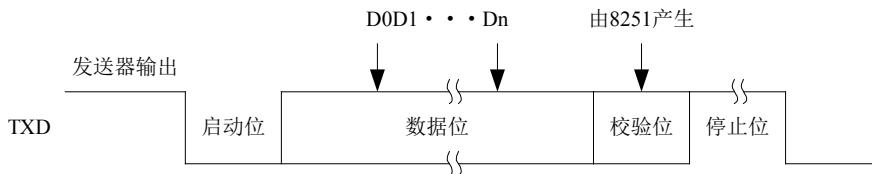


图 8.4.3 8251 工作在异步方式下 TXD 信号的数据传输格式

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 8.4.4 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) X0=内同步 X1=外同步 0X=双同步 1X=单同步	异步 (D1D0≠0) 00=不用 01=1 位 10=1.5 位 11=2 位	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

图 8.4.4 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 8.4.5 所示的是 8251 命令控制字各位的定义。

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

图 8.4.5 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 8.4.6 所示。

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

图 8.4.6 8251 状态字格式

(4) 系统初始化

8251 的初始化和操作流程如图 8.4.7 所示。

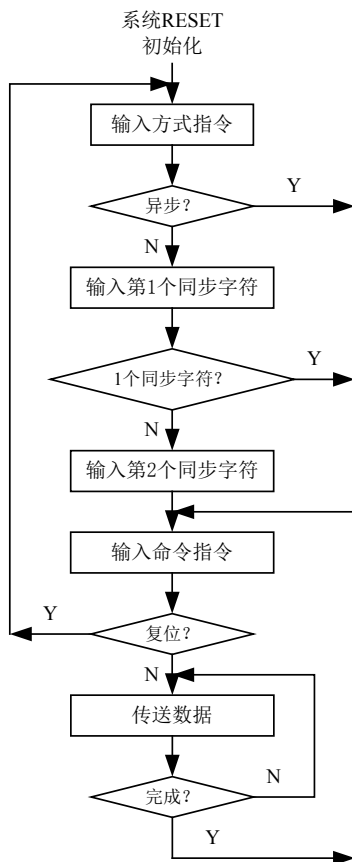


图 8.4.7 8251 初始化流程图

5. 8251 实验单元电路图

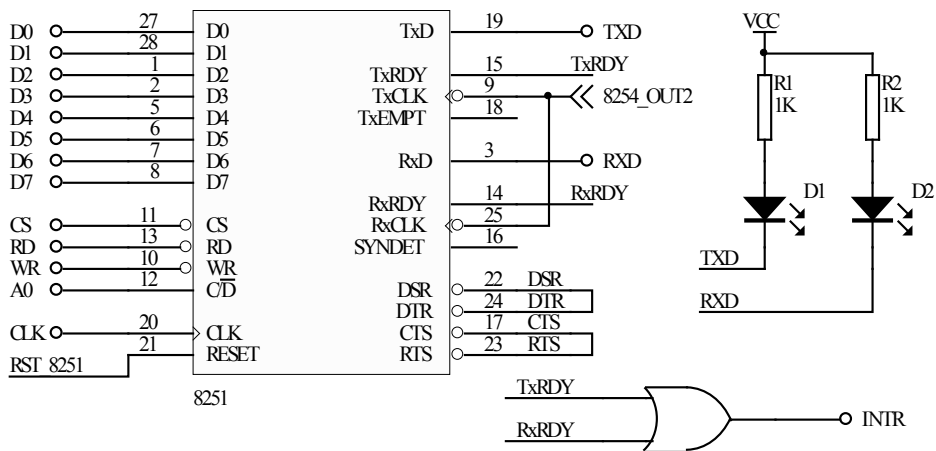


图 8.4.8 8251 实验单元电路图

8.4.5 实验步骤

1. 数据信号的串行传输

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

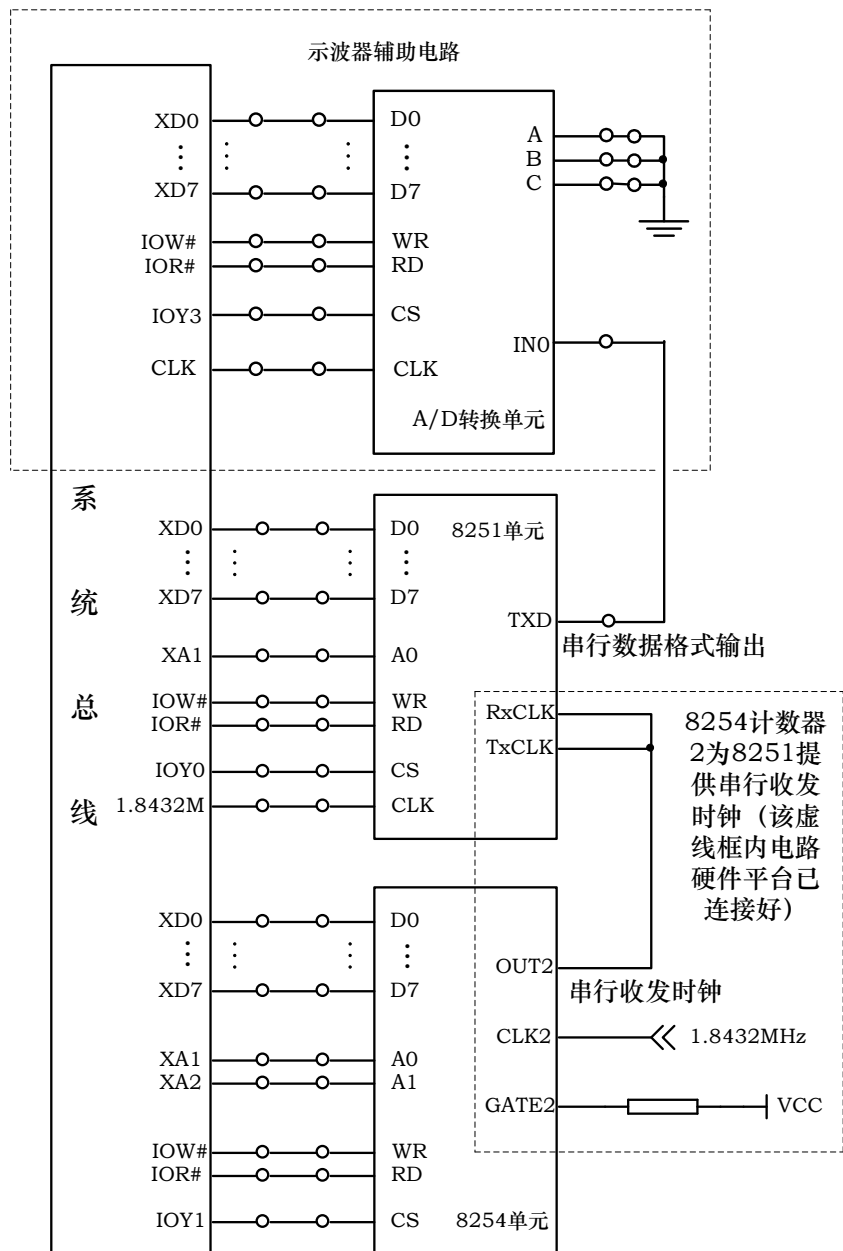






图 8.4.9 8251 数据串行传输实验线路图

- (1) 按图 8.4.9 连接实验接线。
- (2) 编写实验程序（例程文件名为：A82511.ASM），经编译、链接无误后装入系统。
- (3) 单击  按钮，运行实验程序，TXD 引脚上输出串行格式的数据波形。

(4) 用示波器观察波形的方法：单击虚拟仪器菜单中的  示波器 按钮或直接单击工具栏的  按钮，在新弹出的示波器界面上单击  按钮运行示波器，观测实验波形，分析串行数据传输格式。

2. 自收自发实验

通过自收自发实验，可以验证硬件及软件设计，常用于自测试。具体实验步骤如下：

- (1) 参考实验接线图如图 8.4.10 所示，按图连接实验线路。
- (2) 编写实验程序（例程文件名为：A82512.ASM），编译、链接无误后装入系统。
- (3) 使用 E 命令更改 4000H 起始的 10 个单元中的数据。
- (4) 运行实验程序，待程序运行停止。
- (5) 查看 3000H 起始的 10 个单元中的数据，与初始化的数据进行比较，验证程序功能。

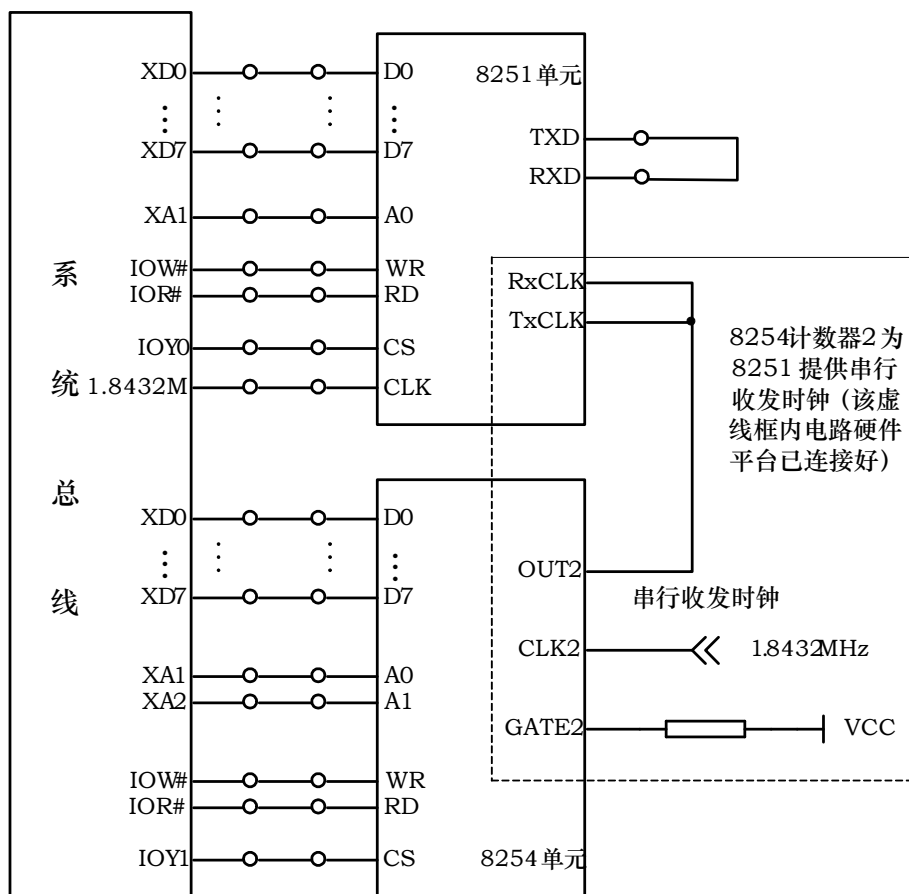


图 8.4.10 自收自发实验接线图

8.5 A/D 转换实验

8.5.1 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。

8.5.2 实验设备

PC 机一台，TDX-PITE 实验装置一套，万用表一个。

8.5.3 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。

8.5.4 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 8.5.1 所示，地址信号与选中通道的关系如表 8.5.1 所示。

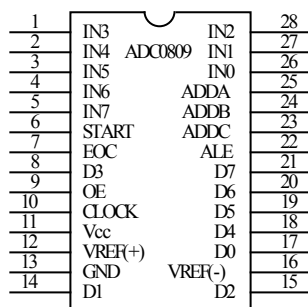


图 8.5.1 ADC0809 外部引脚图

地 址			选中通道
C	B	A	
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

[illegible]

图 8.5.2 模/数转换电路图

8.5.5 实验步骤

1. 按图 8.5.4 连接实验线路。
2. 编写实验程序（例程文件名为：AD.ASM），经编译、链接无误后装入系统。
3. 将变量 VALUE 添加到变量监视窗口中。

方法如下：打开设置\变量监控，出现如图 8.5.3 的界面，选中要监视的变量“VALUE”，单击“加入监视”后确定，就会在软件左侧栏的“变量区”出现该值。

8.6 D/A 转换实验

8.6.1 实验目的

1. 学习数/模转换的基本原理。
2. 掌握 DA 的使用方法。

8.6.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

8.6.3 实验内容

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

8.6.4 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。TLC7528 是 8 位、并行、两路电压型输出数字-模拟转换器。其主要性能参数如表 8.6 示，引脚如图 8.6.1 所示。

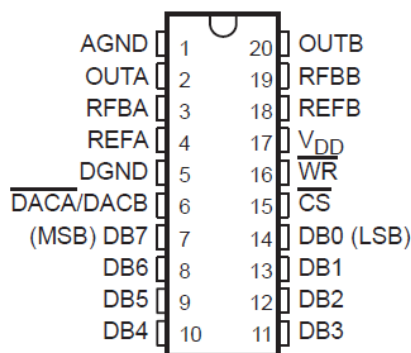


图 8.6.1 TLC7528 引脚图

表 8.6.1 TLC7528 性能参数

性能参数	参数值
分辨率	8 位
单电源	+5V~ +15V
线性误差	1/2LSB
功耗 (VDD=5V))	20mW
稳定时间 (VDD=5V))	100ns
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 8.6.2 所示：

数模转换单元

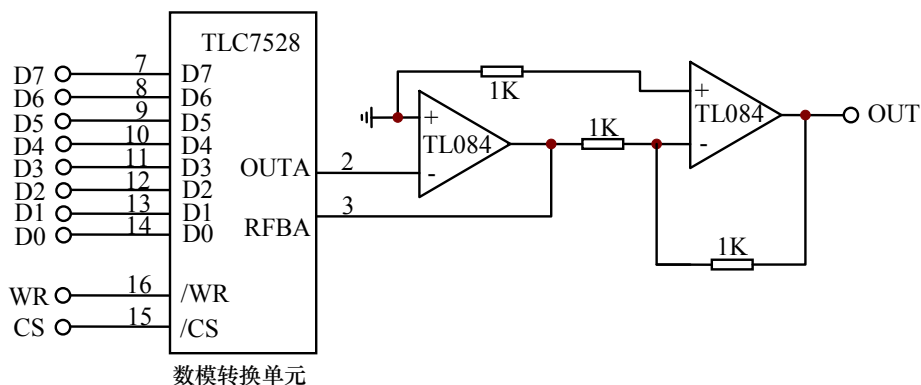


图 8.6.2 D/A 转换单元原理图

数模转换单元采用 TLC7528 芯片，它是 8 位、并行、两路、电压型输出数模转换器。其主要参数如下：转换时间 100ns，满量程误差 1/2 LSB，参考电压 -10V ~ +10V，供电电压 +5V ~ +15V，输入逻辑电平与 TTL 兼容。输入数字范围为 00H ~ FFH，80H 对应于 0V，输出电压为 -5V ~ +4.96V。

8.6.5 实验步骤

(1) 实验接线图如图 8.6.3 所示，按图接线。

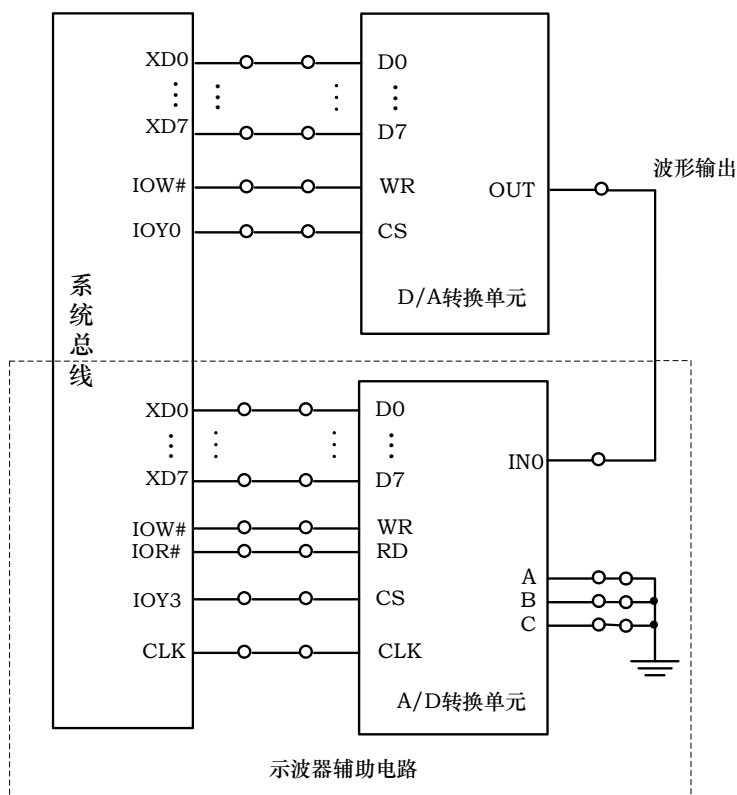






图 8.6.3 D/A 实验接线图

(2) 编写实验程序 (锯齿波例程文件名为: DA1.ASM, 方波例程文件名为: DA2.ASM), 经编译、链接无误后装入系统。

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

(4) 用示波器观察波形的方法: 单击虚拟仪器菜单中的  按钮或直接单击工具栏的  按钮, 在新弹出的示波器界面上单击  按钮运行示波器, 观测实验波形。

(5) 本实验现象结果如图 8.6.4 和图 8.6.5 所示。

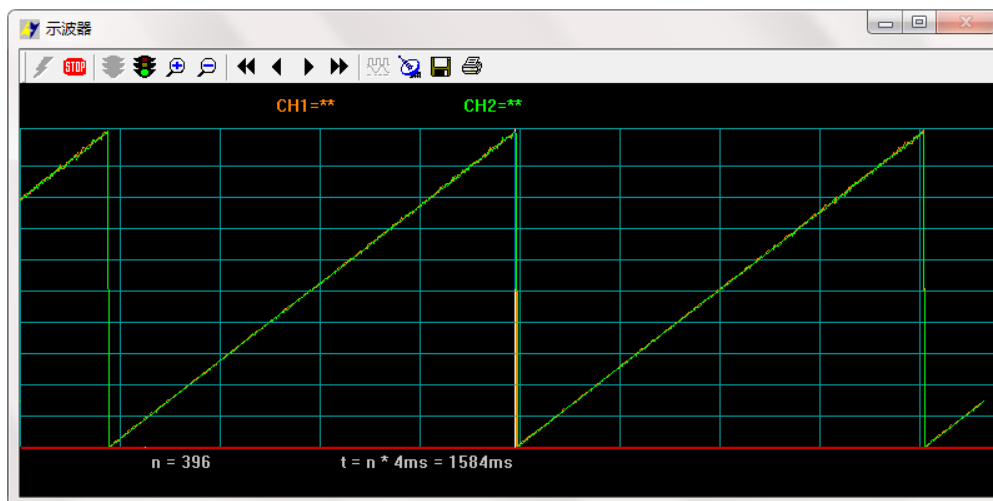


图 8.6.4 DA 产生锯齿波实验结果图

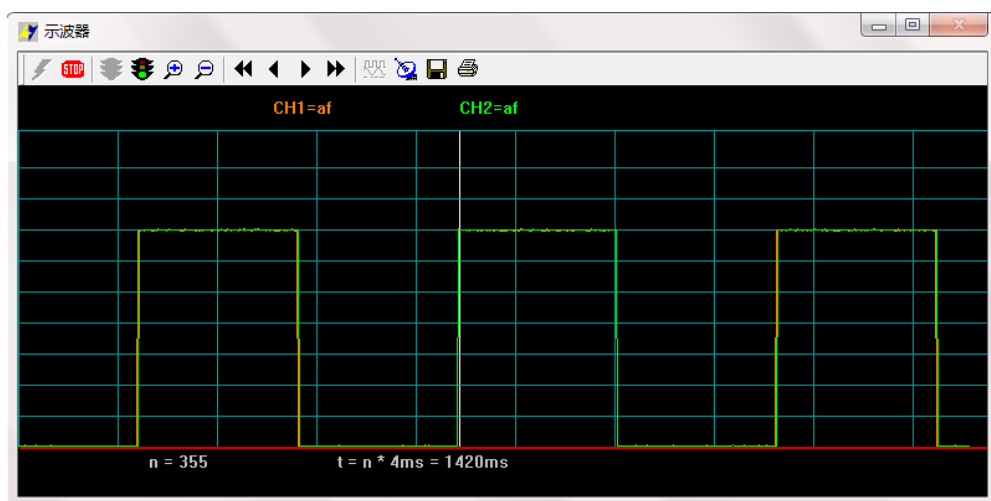


图 8.6.5 DA 产生方波实验结果图

第二章 80x86 保护模式微机原理

(一) 保护模式下的 80X86 机器组织

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

1.1 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。

1.2 寄存器组织

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

1.2.1 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 1.1 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。

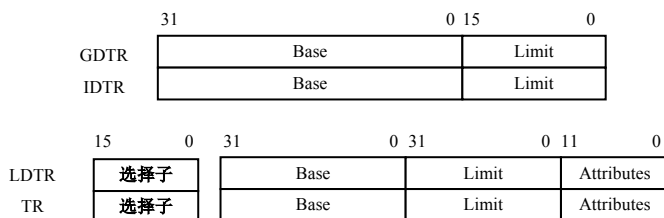


图 1.1 32 位处理器的系统地址寄存器

1.2.2 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 1.2 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。

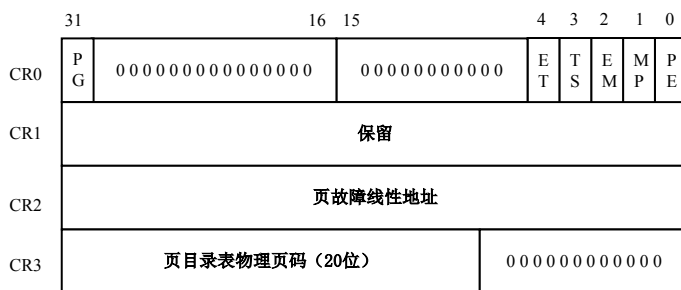


图 1.2 32 位处理器的控制寄存器

表 1.1 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 1.1 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。

1.2.3 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。

1.3 保护模式下的分段存储管理机制

1.3.1 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 1.3 所示。

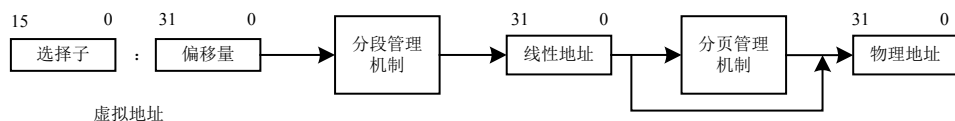


图 1.3 保护模式下的地址转换示意图

1.3.2 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 1.4 所示。

1. 保护模式下段的定义

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

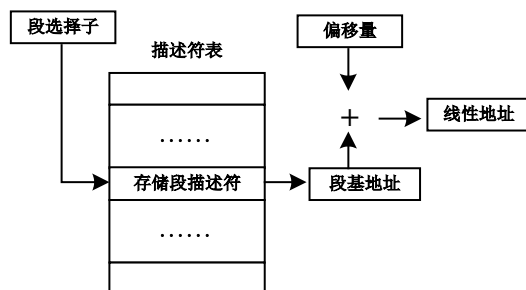


图 1.4 保护模式下的分段管理示意图

(1) 段基地址

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

(2) 段界限

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

(3) 段属性

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

2. 段描述符

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

(1) 存储段描述符的格式如图 1.5 所示。

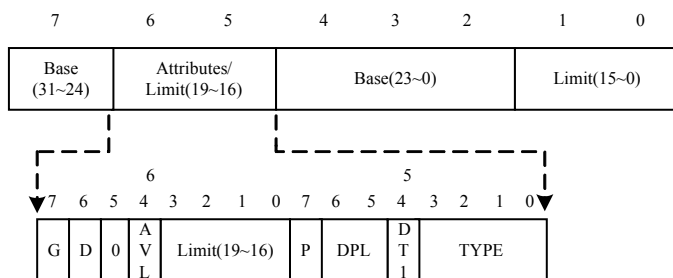


图 1.5 存储段描述符格式

G：段界限粒度位，指示段界限的单位是 1 字节还是 4KB（0/1）；

D：简单来说决定段是 32 位的还是 16 位的。（1/0）；

AVL：软件可利用位（未规定）；

P：表示描述符对地址转换是有效还是无效（1/0）；

DPL：两位，决定了描述符对应的特权级，用于特权级检查，以决定是否能对该段访问；

DT: 表示段的类型(描述对象), 为 1, 表示是存储段;

TYPE: 决定了存储段的属性, 具体属性值见表 1.2。

表 1.2 存储段属性值

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

(2) 系统段描述符和门描述符

系统段是实现存储管理机制所使用的特殊段, 用于描述系统段的描述符称为系统段描述符。32 位处理器中含有两种系统段: 任务状态段和局部描述符表段。系统段描述符的格式如图 1.6 所示。

门描述符并不是描述某种内存段, 而是描述控制转移的入口点。这种描述符好比一个通向另一个代码段的门, 通过这种门, 可以实现任务内特权级的变换和任务间的切换。门描述符可以分成: 任务门、调用门、中断门和陷阱门。门描述符的格式如图 1.7 所示。其中 Dword Count, 是双字计数字段, 该字段只在调用门描述符中有效。主程序通常通过堆栈把入口参数传递给子程序, 如果利用调用门调用子程序时, 将引起特权级的转换和堆栈的改变, 那么就需要将外层堆栈中的参数复制到内层堆栈, 双字计数字段决定了复制的双字参数的数量。

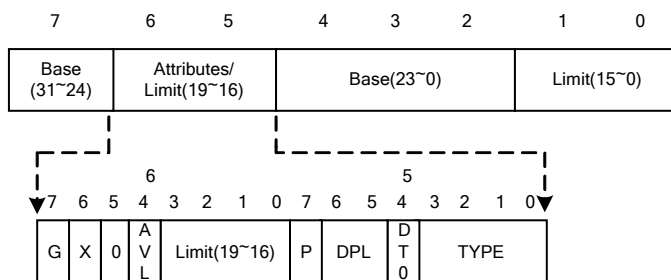


图 1.6 系统段描述符格式

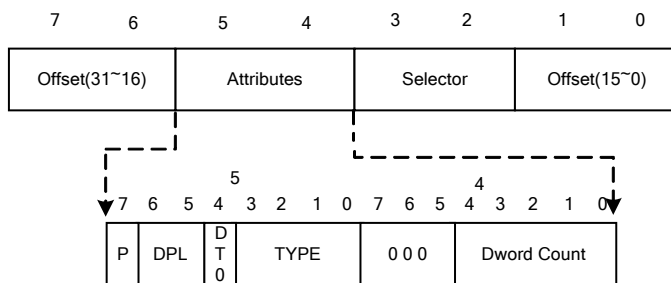


图 1.7 门描述符格式

存储段描述符、系统段描述符和门描述符的格式各不相同, 但 DT 和 DPL, P, TYPE 的位置都相同。系统根据 DT 来区分存储段描述符和系统段描述符、门描述符, 而当 DT 为 0 时

通过 TYPE 值来区分系统段描述符和门描述符。系统段描述符和门描述符的属性值如表 1.3 所示。

表 1.5 系统段和门描述符类型字段的编码和含义

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

3. 描述符表

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

(1) 全局描述符表 (GDT)

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

(2) 局部描述符表 (LDT)

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

(3) 中断描述符表 (IDT)

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

4. 段选择子

在实模式下, 逻辑地址空间中存储单元的地址由段地址和段偏移构成。在保护模式下, 虚拟地址空间中的存储器单元地址由段选择子和段偏移构成。段选择子用来在描述符表中查找相应的段描述符, 其格式如图 1.8 所示。

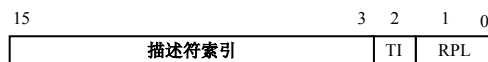


图 1.8 段选择子格式

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

1. 段描述符高速缓冲寄存器

在保护模式下, 段寄存器中装载的是段选择子, 在访问存储器形成线性地址时, 处理器需

要使用选择子定位段的基地址等信息。为了避免在每次存储器访问时，都要访问描述符表从而取得段信息，32 位微处理器为每个段寄存器都配备了一个高速缓冲寄存器，这些寄存器称之为段描述符高速缓冲寄存器。每当把一个选择子装入某个段寄存器时，处理器自动从描述表中取出相应的描述符，把描述符中的信息保存到对应的高速缓冲寄存器中，且在以后对该段访问时，都从高速缓冲寄存器中取得段的信息。这部分内容对程序员不可见。段描述符高速缓冲寄存器的内容如图 1.9 所示。

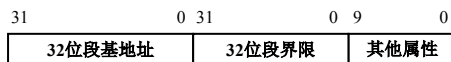


图 1.9 段描述符高速缓冲寄存器内容

6. 特权级

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

在程序的执行过程中，处理器要进行一系列的特权级检查工作，在检查过程中采用了如下规则。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。

1.3.3 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 1.10 所示。

1. 执行代码

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

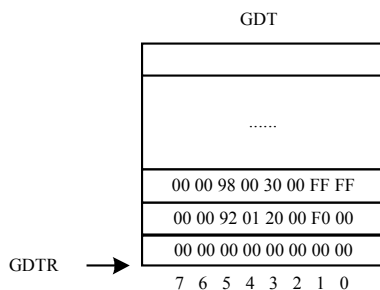


图 1.10 GDT 表内容

2. 执行过程

寻址过程如图 1.11 所示。

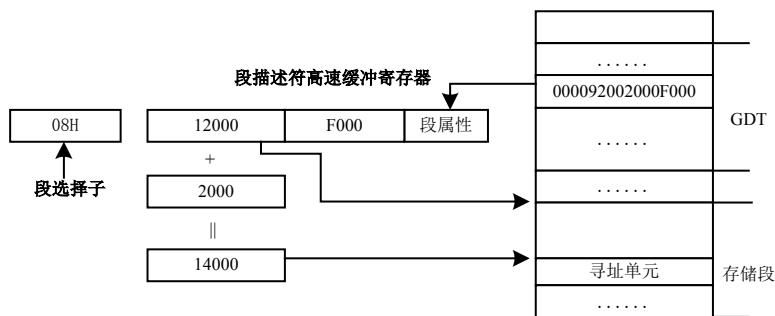


图 1.11 寻址过程示意图

(1) 执行 MOV AX,08H。

(2) 执行 MOV GS,AX。

① 选择子分解: 查 GDT, 索引=01H。

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

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

1.4 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。

1.4.1 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 1.12 所示。任务状态段可以分成如下几个区域。

31			0
0000000000000000	链接字段		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 1.12 任务状态段格式

(1) Link

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

(2) 内层堆栈指针区

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

(3) CR3

如果系统采用了分页机制，则该部分保存了分页机制中使用的页目录的物理地址。

(4) 通用寄存器、段寄存器、指令指针及指令标志寄存器区

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

(5) LDT 选择子

存放着任务自己的局部描述符表对应的选择子。

(6) T 位

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

(7) I/O 许可位图

为实现输入/输出保护而设置。

1.4.2 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。

1.4.3 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。

1.5 任务内的控制转移

1.5.1 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 $CPL \geq DPL$ 。
- (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。

1.5.2 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。

1.6 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。

1.6.1 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。

1.6.2 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 $RPL \leq \text{任务门描述符的 DPL}$, 且 $CPL \leq \text{任务门描述符的 DPL}$; 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。

1.6.3 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。

1.7 中断/异常管理

1.7.1 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 1.4 中列出了保护模式下异常的类型和说明。

表 1.4 保护模式下的异常说明

向量号	说明	出错码
00H	除法错	无
01H	调试异常	无
02H	NMI (未使用)	无
03H	断点	无
04H	溢出	无
05H	边界检查	无
06H	非法操作码	无
07H	协处理器不可用	无
08H	双重故障	有
09H	协处理段越界	无
0AH	无效 TSS	有
0BH	段不存在	有
0CH	堆栈异常	有
0DH	通用保护	有
0EH	页异常	有
10H	协处理器异常	无
剩余	软中断	无

有一些中断/异常在发生时会产生出错码，出错码可以指示错误的类型、引起错误的描述符所在区域以及引起错误的选择子，通过错误码可以快速、准确地定位错误源。异常出错码的格式如图 1.13 所示，其中 TI 位指示了选择子对应的描述符在 GDT 还是 LDT。如果在中断/异常处理时，从 IDT 重读表项时产生异常，IDT 位置位。如果在某一中断/异常正在处理时又产生了某种异常，则 EXT 位置位。

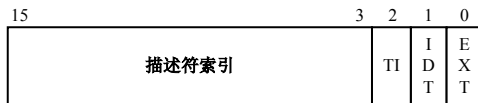


图 1.13 异常出错码格式

2. 中断门和陷阱门描述符

(1) 中断门/陷阱门描述符只在 IDT 中有效。

(2) 中断门/陷阱门描述符中的选择子指示了中断/异常处理程序所在的代码段描述符，偏移地址指示了处理程序在对应代码段内的偏移量。

(3) 使用中断门描述符进入中断处理程序时，CPU 标志寄存器的中断标志位 IF 自动清 0 来关中断。

(4) 使用陷阱门描述符进入异常处理程序时不关中断。

1.7.2 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 1.14 及图 1.15 示，图 1.14 表示了利用中断门/陷阱门进行中断/异常处理时不需要进行特权级的变换，则堆栈不需要切换，只需要将标志寄存器和返回地址压入堆栈即可，如果产生的异常有出错码，还将在堆栈中保留出错码的值。图 1.15 表示了利用中断门/陷阱门进行中断/异常处理时引起任务内特权级的变换，则需要将堆栈切换成内层堆栈，并将外层堆栈、标志寄存器、返回地址和出错码（如果有）的值压入堆栈。

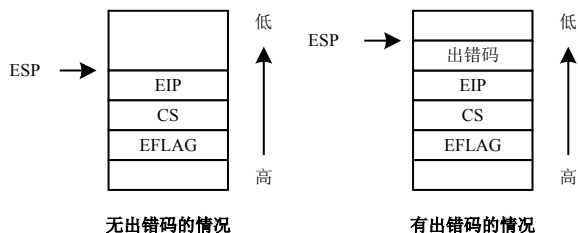


图 1.14 无特权级变换时的堆栈

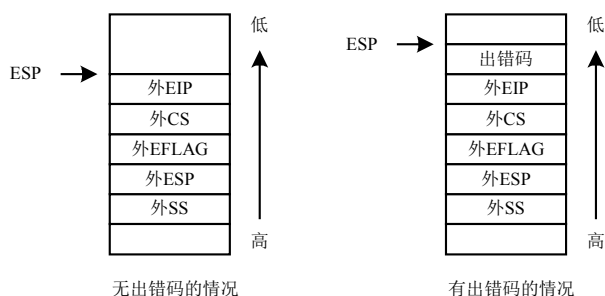


图 1.15 有特权级变换时的堆栈

2. 通过任务门实现的中断/异常处理

如果中断/异常向量号所指向的控制门描述符是任务门描述符，那么控制将转移到一个以独立任务方式出现的中断/异常处理程序。系统可从任务门描述符中获得一个 48 位的全指针，其中 16 位的选择符指向中断/异常处理程序任务的 TSS 段的描述符。通过任务门实现中断/异常控制转移，在进入中断/异常处理程序时，标志寄存器 EFLAG 中的 NT 位被置 1，表示是嵌套任务。

通过任务门的中断/异常控制转移与使用段间调用指令 CALL 通过任务门到一个 TSS 的转移过程很相似，主要的区别就是要在完成任务切换后把中断/异常处理的出错码压入新任务的堆栈中。

3. 中断/异常处理的返回

使用 IRET 指令，可以从中断/异常处理程序返回到异常点。该指令的执行根据中断/异常的进入方式有所不同。

当进入方式是通过任务门时，EFLAG 中的 NT 位被置位，则 IRET 的执行将引起一次任务切换，目标任务的 TSS 选择子就保存在中断/异常处理程序所在任务的 TSS 段中，即 LINK 字段指示的选择子。

当进入方式是通过中断门/陷阱门时，在堆栈中保存了返回地址的指针。处理器执行 IRET 时根据 CS 的 RPL 判定是否会引起任务内特权级的变换，如果不会引起变化，则直接弹出 EIP，CS 和 EFLAG，如果会引起内层到外层的变化，则还会弹出外层堆栈的堆栈指针。

1.8 80X86 保护模式程序设计

通过西安唐都科教仪器公司提供的联机软件 TDX-PITE 环境可实现 80X86 保护模式程序的编程和上机实验，使学生能够很好地理解和学习保护模式下的程序设计。

1.8.1 常用数据结构及标号定义

1. 存储段/系统段描述符数据结构定义

```
DESC          STRUC
LIMITL       DW      0    ; 段界限(BIT0-15)
BASEL        DW      0    ; 段基地址(BIT0-15)
```

```

BASEM      DB      0      ; 段基地址(BIT16-23)
ATTRIBUTES DB      0      ; 段属性
LIMITH     DB      0      ; 段界限(BIT16-19)(含段属性的高 4 位)
BASEH      DB      0      ; 段基地址(BIT24-31)
DESC        ENDS

```

2. 常用标号定义

```

ATCE       =   98h        ; 存在的只执行代码段属性值
ATDR       =   90h        ; 存在的只读数据段类型值
ATDW       =   92h        ; 存在的可读写数据段属性值

```

1.8.2 程序设计的一些说明

1. 伪指令 “.386” 或 “.386P”

在编程最前头必须写上 “.386” 或 “.386P” 伪指令。这是为了让编译器识别 80386 处理器的新增指令或功能增强的指令。

如果缺省不写，MASM 和 TASM 只能识别 8086/8088 的指令。

2. 全局描述符表 GDT 编写

定义一个全局描述符表 GDT，其结构如下：

```

;
DSEG SEGMENT PARA USE16                ; 16 位数据段
GDT   LABEL   BYTE                      ; 全局描述符表
DUMMY DESC    <>                        ; 空描述符
CODE  DESC    <0FFFFH, CODESEG,, ATCE,, > ; 代码段描述符
CODE_SEL =    CODE-GDT                  ; 代码段选择子
GDTLEN  =    $-GDT                      ; 全局描述符表长度
DSEG    ENDS                            ; 数据段定义结束
;

```

说明：定义的 DSEG 是一个 16 位的数据段，在这个数据段内定义 GDT 表。

(1) “GDT LABEL BYTE” 表明全局描述符表的开始；

(2) 定义一个空描述表 “DUMMY DESC <>” 的结构；

(3) 定义代码段描述符 “CODE DESC <0FFFFH, CODESEG,, ATCE,, >”，它表示代码段的描述符是 “CODE”，代码段是 “CODESEG” 段，代码段的段界限是 “0FFFFH”，“ATCE” 表示代码段的属性是一个可执行段。

(4) “CODE_SEL” 是代码段描述符的选择子，即代码段描述符 “CODE” 在全局描述表 GDT 中的位置。

(5) “GDTLEN” 是全局描述表的长度。

3. 定义全局描述符表寄存器 GDTR 的伪描述符 VGDR

可先定义伪描述符如下：

```

PDESC      STRUC
LIMIT      DW      0

```

```
BASE      DD      0
PDESC     ENDS
```

利用以上所定义的结构类型，可定义 GDTR 的伪描述符：

```
VGDTR     PDESC     <GDTLEN-1,>
```

其中：“GDTLEN-1”表示 GDT 表的界限，“0”表示 GDT 表的基址。

4. 实模式和保护模式的切换

80386 有四个控制寄存器，分别命名为 CR0、CR1、CR2 和 CR3。其中控制寄存器 CR0 中的 0 位用 PE 标记，它决定了 CPU 的工作模式。PE=0，处理器运行于实模式；PE=1，处理器运行于保护模式。

从实模式切换到保护模式，可用如下 3 条指令：

```
MOV       EAX, CR0           ; 把 CR0 复制到 EAX
OR        EAX, 1             ; 把对应的 PE 位置 1
MOV       CR0, EAX           ; 使 CR0 的 PE 位为 1
```

执行完上述 3 条指令后，紧接着要安排一条段间转移指令：

```
JUMP      <CODE_SEL>, <OFFSET_VIRTUAL>
```

这条段间转移指令是在实模式下被预取，在保护模式下被执行。利用这条段间转移指令可把保护模式下代码段的选择子装入 CS，同进刷新指令预取队列。从而真正进入保护模式。

从保护模式切换到实模式，可用如下 3 条指令：

```
MOV       EAX, CR0           ; 把 CR0 复制到 EAX
AND       AL, 0FEH           ; 把对应的 PE 位置 0
MOV       CR0, EAX           ; 使 CR0 的 PE 位为 1
```

执行完上述 3 条指令后，紧接着也要安排一条段间转移指令：

```
JUMP      <SEG_REAL>, <OFFSET_REAL>
```

这条段间转移指令，一方面清指令预取队列，另一方面把实模式下的代码段的段值送 CS。这条段间转移指令在保护模式下被预取，在实模式下被执行。

1.8.3 一个实例

下面编写一个实例，学习实模式和保护模式的切换，理解保护模式的编程方法。具体实现步骤：

- (1) 作切换到保护模式的准备；
- (2) 切换到保护模式；
- (3) 把源数据段的内容传送到目的数据段；
- (4) 切换回实模式；
- (5) 显示源数据段和目的数据段的内容。

实例源程序如下：

```
;-----
;文件名:PM.ASM
;功能:学习实模式和保护模式的切换
;-----
```

;存储段描述符结构类型定义

```
DESC          STRUC
LIMITL       DW 0    ;段界限(BIT0-15)
BASEL        DW 0    ;段基地址(BIT0-15)
BASEM        DB 0    ;段基地址(BIT16-23)
ATTRIBUTES   DB 0    ;段属性
LIMITH       DB 0    ;段界限(BIT16-19)(含段属性的高4位)
BASEH        DB 0    ;段基地址(BIT24-31)
DESC          ENDS
```

;-----

;常量定义

```
ATDW          EQU    92H    ;存在的可读写数据段属性值
ATCE          EQU    98H    ;存在的只执行代码段属性值
```

;-----

;伪描述符结构类型定义

```
PDESC        STRUC
LIMIT        DW 0    ;16 位界限
BASE         DD 0    ;32 位基地址
PDESC        ENDS
```

;-----

;16 位偏移的段间直接转移指令的宏定义

```
JUMP16        MACRO SELECTOR, OFFSET
DB    0EAH    ;操作码
DW    OFFSET  ;16 位偏移量
DW    SELECTOR ;段值或段选择子
ENDM
```

;-----

.386P ;.386P 伪指令

;-----

```
DSEG          SEGMENT PARA      USE16    ;定义 16 位数据段
GDT           LABEL BYTE                ;全局描述符表
DUMMY         DESC      <>              ;空描述符
CODEM         DESC      <0FFFFH,,,ATCE,,> ;代码段描述符
DATAS         DESC      <0FFFFH,,,ATDW,,> ;源数据段描述符
DATAD         DESC      <0FFFFH,,,ATDW,,> ;目标数据段描述符
VGDTR         PDESC <GDTLEN-1,>          ;伪描述符
GDTLEN        =        $-GDT            ;全局描述符表长度
CODEM_SEL     =        CODEM-GDT         ;代码段选择子
DATAS_SEL     =        DATAS-GDT         ;源数据段选择子
```

```

DATAD_SEL      =      DATAD-GDT      ;目标数据段选择子
DSEG           ENDS                    ;数据段定义结束
;-----
DATA1          SEGMENT PARA           USE16      ;定义 16 位源数据段
MES1           DB 'This is tangdu speak!',' $' ;$ 为字符串结束标志
ML             =   $-MES1              ;字符串的长度
DATA1          ENDS                    ;源数据段定义结束
;-----
DATA2          SEGMENT PARA           USE16      ;定义 16 位目标数据段
BUF            DB 64 DUP(?)            ;定义 64 个缓冲区字节单元
DATA2          ENDS                    ;目标数据段定义结束
;-----
CSEG           SEGMENT USE16           ;16 位代码段
ASSUME CS:CSEG,DS:DSEG
START PROC
    MOV        AX,DSEG
    MOV        DS,AX
    ;准备要加载到 GDTR 的伪描述符
    MOV        BX,16
    MUL        BX                      ;数据段地址左移 4 位
    ADD        AX,OFFSET GDT          ;加上 GDT 的偏移得到物理地
址
    ADC        DX,0
    MOV        WORD PTR VGDTR.BASE,AX ;将得到的物理地址填入
VGDTR 描述符
    MOV        WORD PTR VGDTR.BASE+2,DX
    ;设置代码段描述符
    MOV        AX,CS
    MUL        BX                      ;代码段地址左移 4 位
    MOV        WORD PTR CODEM.BASEL,AX ;代码段开始偏移为 0
    MOV        BYTE PTR CODEM.BASEM,DL ;将得到的物理地址填入
CODEM 描述符
    MOV        BYTE PTR CODEM.BASEH,DH
    ;设置源代码段描述符
    MOV        AX,DATAS
    MUL        BX                      ;源数据段地址左移 4 位
    MOV        WORD PTR DATAS.BASEL,AX ;源数据段开始偏移为 0
    MOV        BYTE PTR DATAS.BASEM,DL ;将得到的物理地址填入
DATAS 描述符

```

```

MOV     BYTE PTR DATAS.BASEH,DH
;设置目标代码段描述符
MOV     AX,DATA2
MUL     BX                      ;目标数据段地址左移 4 位
MOV     WORD PTR DATAD.BASEL,AX      ;目标数据段开始偏移为
0
MOV     BYTE PTR DATAD.BASEM,DL      ;将得到的物理地址填入
DATAD 描述符
MOV     BYTE PTR DATAD.BASEH,DH
;加载 GDTR
LGDT    QWORD PTR VGDTR
CLI                      ;关中断
;切换到保护方式
MOV     EAX,CRO
OR      EAX,1
MOV     CRO,EAX
;清指令预取队列,并真正进入保护方式
JUMP16  <CODEM_SEL>,<OFFSET_VIRTUAL>
VIRTUAL: ;现在开始在保护方式下运行
MOV     AX,DATAS_SEL
MOV     DS,AX                ;加载源数据段描述符
MOV     AX,DATAD_SEL
MOV     ES,AX                ;加载目标数据段描述符
XOR     DI,DI                ;DI 清零
XOR     SI,SI                ;SI 清零
MOV     CX,ML                ;设置数据长度
REPZ    MOVSB                ;通过串传送指令传数
;切换回实模式
MOV     EAX,CRO
AND     AL,11111110B
MOV     CRO,EAX
;清指令预取队列,进入实方式
JUMP16  <SEG_REAL>,<OFFSET_REAL>
REAL:   ;现在又回到实方式
STI                      ;开中断
MOV     AX,DATA1
MOV     DS,AX                ;送源数据段
MOV     DX,OFFSET MES1
MOV     AH,09H

```



```

                INT      21H                      ;用 INT 21H 功能调用显示 MES1 数据
段的内容
                MOV      AH,02H
                MOV      DL,0DH
                INT      21H                      ;回车
                MOV      AH,02H
                MOV      DL,0AH
                INT      21H                      ;换行
                MOV      AX,DATA2
                MOV      DS,AX                    ;送目标数据段
                MOV      DX,OFFSET BUF
                MOV      AH,09H
                INT      21H                      ;用 INT 21H 功能调用显示 BUF 数据
段的内容
                MOV      AX,4C00H
                INT      21H                      ;程序终止
START          ENDP
;-----
CSEG           ENDS                      ;代码段定义结束
;-----
END            START

```

在上例中，程序刚开始定义了描述符结构类型及常量等。一般的，我们将这些结构类型及常量定义可放在一个头文件 386SCD.INC 里，在编程时列出头文件即可。如：

```
INCLUDE 386SCD.INC
```

参考上述例子，现给出保护模式编程格式大致如下：

```

;-----
INCLUDE      386SCD.INC
;-----
.386P
;-----
GDTSEG      SEGMENT PARA      USE16
            .....            ;定义全局描述符表 GDT 数据段
GDTSEG      ENDS
;-----
LDTSEG      SEGMENT PARA      USE16
            .....            ;定义局部描述符表 LDT 数据段
LDTSEG      ENDS
;-----

```



```

IDTSEG      SEGMENT PARA USE16
    .....      ;定义中断描述符表 IDT 数据段
IDTSEG      ENDS
;-----
TSSSEG      SEGMENT PARA USE16
    .....      ;定义任务表 TSS 数据段
TSSSEG      ENDS
;-----
DSTACKSEG   SEGMENT PARA USE16
    .....      ;定义堆栈数据段
DSTACKSEG   ENDS
;-----
DDATASEG    SEGMENT PARA USE16
    .....      ;定义数据段的数据段
DDATASEG    ENDS
;-----
CODEMSEG    SEGMENT PARA USE16
    .....      ;定义主程序段的代码段
    MOV      EAX,CR0 ;切换到保护模式
    OR       EAX,1
    MOV      CR0,EAX
    JUMP16    <CODEM_SEL>,<OFFSET VIRTUAL>
VIRTUAL:    .....      ;现在开始在保护模式下运行
    .....
    MOV      EAX,CR0 ;切换到实模式
    AND      AL,11111110B
    MOV      CR0,EAX
    JUMP16    <SEG REAL>,<OFFSET REAL>
REAL:      .....      ;现在又回到实方式
CODEMSEG    ENDS
;-----

```

(二) 保护模式微机原理及其程序设计实验

本章列举了 80X86 工作在保护模式下的原理及其程序设计实验。其中包括四大类型的实验，即描述符及描述符表实验、特权级变换实验、任务切换实验和中断与异常处理实验。这些实验通过唐都科教仪器公司提供的含 80386EX 32 位单板微机系统的实验箱以及专为其配套开发的 TDX-PITE 联机软件来完成。

2.1 描述符及描述符表实验

2.1.1 实验目的

1. 熟悉保护模式的编程格式。
2. 掌握全局描述符及局部描述符的声明方法。
3. 掌握使用选择子访问段的寻址方法。

2.1.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.1.3 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 1.3 节）。

选择子确定描述符，描述符确定段基地址，段基地址和偏移之和就是线性地址。所以虚拟地址空间中段选择子和偏移两部分构成的二维虚拟地址，就是这样确定了线性地址空间中的一

维线性地址。

2. 实验分析

为了实现在 0 级代码段中完成数据传输,实验程序中需要安排一个 0 级代码段和两个 0 级数据段(可以是 0~3 级任一级别的数据段)。

在程序开始声明一个数据段“DSEG”,来描述这三个段的描述符,其中有代码段描述符 CODEM,源数据段描述符 DATAS 和目标数据段描述符 DATAD,将它们相应的选择子分别定义为 CODEM_SEL, DATAS_SEL, DATAD_SEL。为这三个段分别定义描述符:

(1) 代码段描述符: CODE DESC <0FFFFH,,,ATCE,,>

段属性说明:

```
G      :      0      ; 以字节为段界限粒度
D      :      0      ; 是 16 位的段
P      :      1      ; 描述符对地址转换有效/该描述符对应的段存在
DPL    :      0      ; 0 级段
DT     :      1      ; 描述符描述的是存储段
TYPE   :      0x8     ; 只执行段
```

(2) 源数据段描述符: DATAS DESC <0FFFFH,,,ATDW,,>

段属性说明:

```
G      :      0      ; 以字节为段界限粒度
D      :      0      ; 是 16 位的段
P      :      1      ; 描述符对地址转换有效/该描述符对应的段存在
DPL    :      0      ; 0 级段
DT     :      1      ; 描述符描述的是存储段
TYPE   :      0x2     ; 可读写段
```

(3) 目标数据段描述符 DATAD DESC <0FFFFH,3000H,,,ATDW,,>

目标数据段描述符的内容基本与源数据段的内容相同,只是我们给定了它的段基址是 00003000H。

在全局描述表 GDT 定义的过程中,首先需要定义一个空的描述符 DUMMY,作为定义的开始,然后再定义其它描述符。

本实验可实现将一个数据段中的数据搬运到另一个数据段所指定的地址空间中(指定地址给定为 00003000H)。源数据段 DSEG1 的内容为 1111H, 2222H, 3333H, 4444H, 5555H, 6666H, 7777H, 8888H, 在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

3. 实验步骤

(1) 运行 TDX-PITE 集成操作软件,进入 TDX-PITE 集成开发环境。

(2) 在菜单“设置\语言”栏,选择“汇编语言”,如图 2.1:

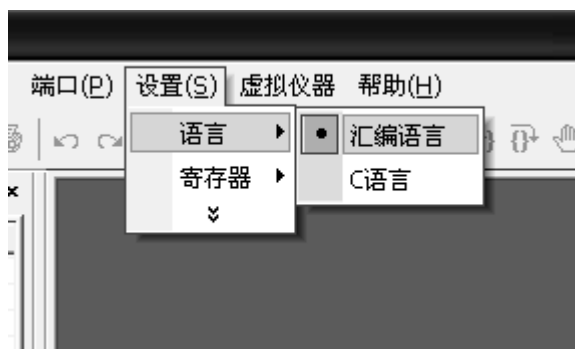


图 2.1 设置语言环境界面

(3) 在菜单“设置\寄存器”栏，选择“32 位寄存器”，由于保护模式下的实验，都用到了 32 位寄存器，所以本章及本章向后的全部实验都必须在 32 位寄存器状态下工作，如图 2.2:

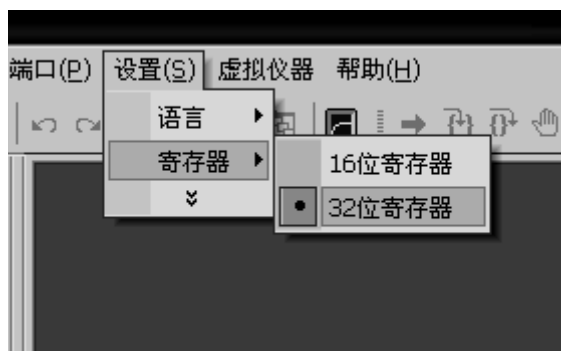


图 2.2 设置寄存器类型界面

设置好以后，下次再启动软件，设置栏将保持这次的修改不变。

(4) 设置完毕后，点击新建或按 Ctrl+N 组合键来新建一个文档，如图 2.3 所示，默认文件名为 Wmd861。



图 2.3 新建文件界面

(5) 编写实验程序（例程文件名为：P1-1.ASM），如图 2.4 所示，并保存，此时系统会提示输入新的文件名，输完后点击保存。



图 2.4 编辑环境界面





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



图 2.5 编译链接信息界面

(7) 连接 PC 与实验系统的通讯电缆，打开实验系统电源。

(8) 编译、链接都正确并且上下位机通讯成功后，就可以下载程序，联机调试了。可以通过端口列表中的“端口测试”来检查通讯是否正常。点击  下载程序。  为编译、链接、下载组合按钮，通过该按钮可以将编译、链接、下载一次完成。下载成功后，在输出区的结果窗中会显示“加载成功！”，表示程序已正确下载。起始运行语句下会有一条绿色的背景。如图 2.6 所示：

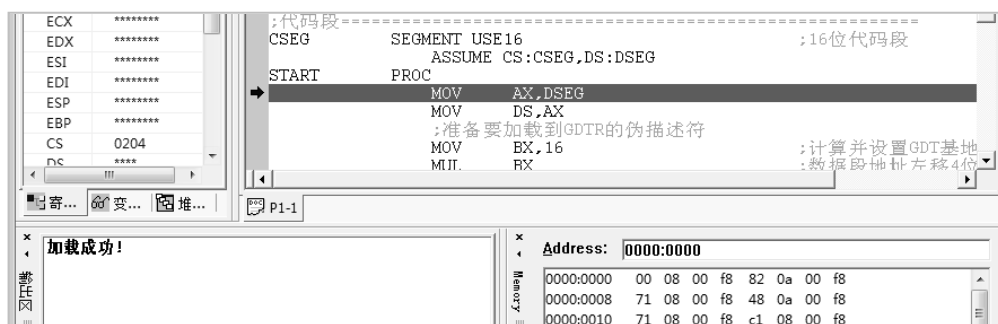


图 2.6 程序起始运行界面

(9) 将输出区切换到调试窗口，使用 D0000:3000 命令查看内存 0000: 3000H 起始地址的数据，如图 2.7 所示。存储器在初始状态时，默认数据为 CC。

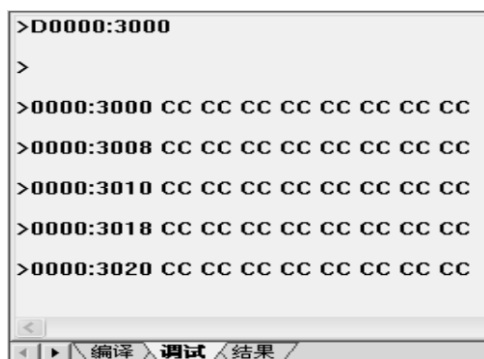



图 2.7 调试界面

(10) 点击按钮  运行程序，待程序运行停止，观察运行结果，使用命令 D0000:3000 回车，观察数据变化。如图 2.8:

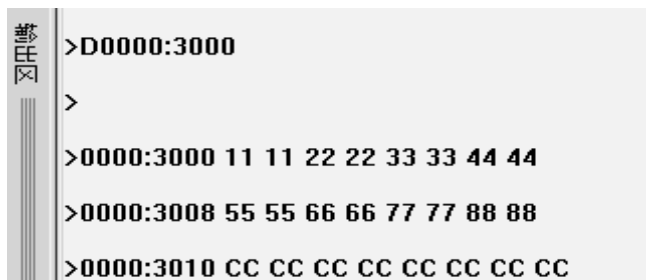


图 2.8 查看内存数据界面

(11) 可以使用 E0000:3000 来改变该地址单元的数据，如图 2.9 所示，输入 01 后，按“空格”键，可以接着输入第二个数，如 02，结束输入按“回车”键。

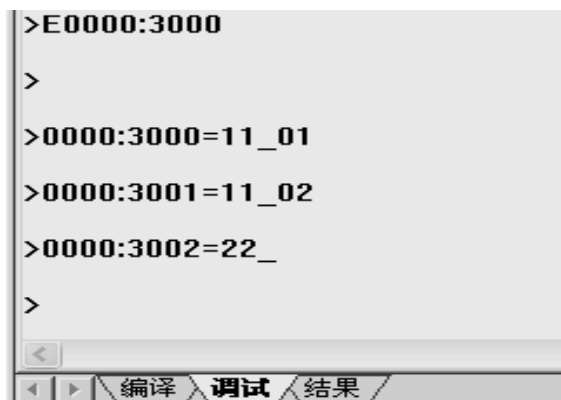


图 2.9 修改内存数据

4. 保护模式下查看描述符表的方法

通过 TDX-PITE 软件输出区中集成的 Debug 调试功能，可以很方便的在做保护模式实验时，查看实验所涉及的描述符表及其变化。

在本章中，编程所定义的全局描述符表 GDT 中的各个段描述符、局部描述符表 LDT 中各个段描述符和中断/异常描述符表 IDT 中各个门描述符都可以用 Debug 命令来查看它。

现以本小节 P1-1.asm 程序为例，讲述查看描述符表的方法。

(1) 编译链接加载该程序后，从变量区的寄存器栏可以读出在实模式下加载该程序的代码段和偏移值。分别为：

CS: 0204, IP: 0000

如图 2.10 所示。

(2) 用反汇编命令查看数据段装载的地址空间。

在输出区调试栏输入：

U0204: 0000 回车

如图 2.11 所示。



图 2.11 反汇编查看数据段

寄存器/变量/堆栈区		
寄存器名	值 (16进制)	
EAX	*****	
EBX	*****	
ECX	*****	
EDX	*****	
ESI	*****	
EDI	*****	
ESP	*****	
EBP	*****	
CS	0204	
DS	****	
ES	****	
SS	****	
FS	****	
GS	****	
IP	0000	
CF	0	
ZF	0	
SF	0	
OF	0	
PF	0	
AF	0	
IF	0	
DF	0	

图 2.10 查看寄存器

(3) 从图 2.11 前两条指令可以看到数据段的段值 DS 为 0200，即就是原程序中 DSEG 数据段的段值，由于在原程序的 DSEG 段中，定义了全局描述符表 GDT，在该表中给出了 DUMMY、CODEM、DATAS、DATAD 等段的描述符，且每个描述符都是用 8 个字节来描述。所以就可以通过查看该 DSEG 段地址中的数据值来查看每个段的描述符。

下面是原程序中定义的 DSEG 段：

DSEG	SEGMENT USE16	;定义 16 位数据段
GDT	LABEL BYTE	;全局描述符表
DUMMY	DESC <>	;空描述符
CODEM	DESC <OFFFFH,,ATCE,,>	;代码段描述符
DATAS	DESC <OFFFFH,,ATDW,,>	;源数据段描述符
DATAD	DESC <OFFFFH,3000H,,ATDW,,>	;目标数据段描述符
VGDT	PDESC <GDTLEN-1,>	;伪描述符
GDTLEN	= \$-GDT	;全局描述符表长度
CODEM_SEL	= CODEM-GDT	;代码段选择子
DATAS_SEL	= DATAS-GDT	;源数据段选择子
DATAD_SEL	= DATAD-GDT	;目标数据段选择子
DSEG	ENDS	;数据段定义结束

(4) 通过 D 命令来查看所定义的 GDT 描述符，即 DSEG 段，在输出区调试栏输入：
D0200: 0000 回车
如图 2.12 所示。

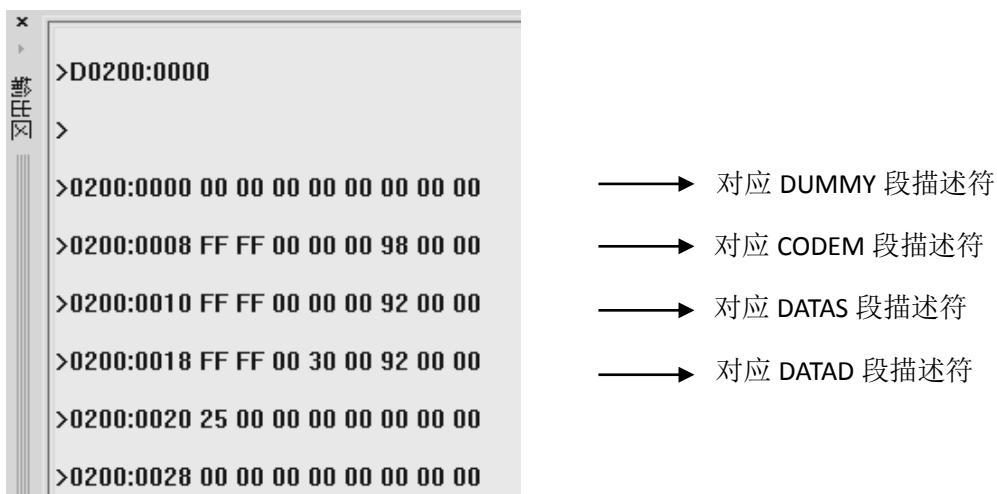


图 2.12

图 2.12 即就是所看到的 GDT 表中定义各段的描述符,这是程序加载完后各段描述符的值,其中:“0FFFFH”是段界限,“98”“92”等是段属性,各段的基地址目前全是“0”。

(5) 在程序进入保护模式之前设断点,然后运行至断点。如图 2.13 所示。



图 2.13

(6) 运行到进入保护模式之前停止,查看一下各描述符的值,如图 2.14 所示。可以看到各段的描述符中,段基地址已经有了值。

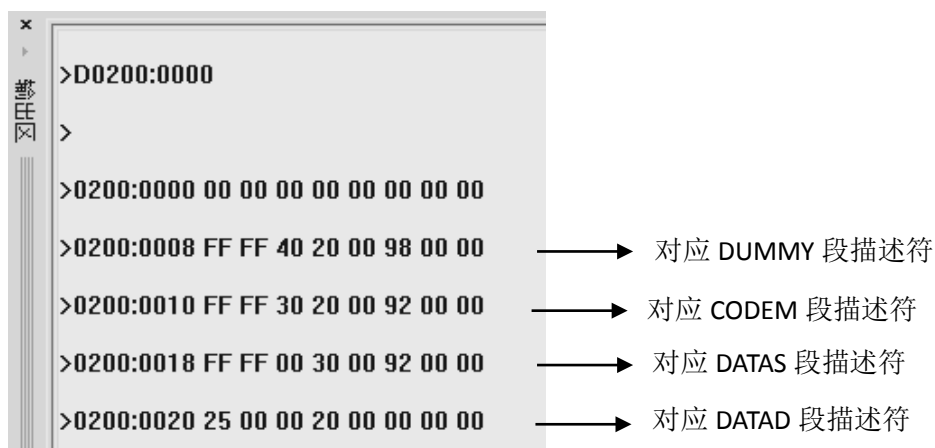


图 2.14

(7) 继续运行程序,直到运行停止,再查看一下各描述符的值,如图 2.15 所示。可以看到各段的描述符中,段属性的值发生了变化,想一想这是为什么?

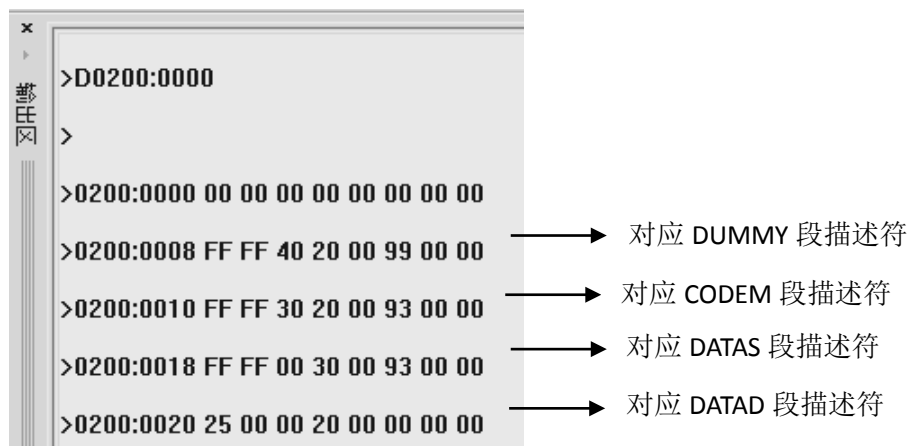


图 2.15

所以,通过以上方法,利用 TDX-PITE 软件提供的 Debug 调试功能,在做保护模式实验时,就可以查看描述符表内的各个段描述符的值。

2.1.4 局部描述表实验

本实验与上一实验所完成的功能相同,但要求将代码段安排在全局描述符表中,而将数据段安排在局部描述符表中。

1. 预备知识

全局描述符表 GDT 含有每一个任务都可能或可以访问的段的描述符,通常包含描述操作系统所使用的代码段、数据段和堆栈段的描述符,也包含多种特殊数据段描述符,如各个用于描述任务局部描述符表 LDT 的特殊数据段。在任务切换时,并不切换 GDT。

每个任务的局部描述符表 LDT 含有该任务自己的代码段、数据段和堆栈段的描述符,也包含该任务所使用的一些门描述符,如任务门和调用门描述符等。随着任务的切换,系统当前的局部描述符表 LDT 也随之切换。

通过 LDT 可以使各任务私有的各个段与其他任务相隔离,从而达到受保护的目的。通过 GDT 可以使各任务都需要使用的段能够被共享。

2. 实验分析

本实验需要为代码段和数据段分别声明描述符,由于要求将数据段的描述符放入 LDT 表中,所以实验程序需要建立一张局部描述符表,并在 GDT 表中声明 LDT 表对应的描述符。描述符声明完成,还需要为它们定义相应的选择子。

实验程序在 DSEG 段中描述 GDT 表中的描述符,其中包括主代码段和 LDT 表的描述符。在 DSEG1 段中描述 LDT 表中的描述符,其中包括源数据段描述符目标数据段描述符。

由于主代码段需要访问的源数据段和目的数据段是在 LDT 表中声明的,所以在程序的初始需要执行装载 LDTR 的指令。装载 LDT 表使用的指令如下。

```
MOV     AX, LDT_SEL
```

LLDT AX

(1) LDT 表对应段描述符: LDTABLE DESC <0FFFFH,,,ATLDT,,>

段属性说明:

G : 0 ; 以字节为段界限粒度
D : 0 ; 是 16 位的段
P : 1 ; 描述符对地址转换有效/该描述符对应的段存在
DPL : 0 ; 0 级段
DT : 0 ; 描述符描述的是系统段或门描述符
TYPE : 0x8 ; LDT 表
ATLDT EQU 82h ;局部描述符表段类型值

(2) 数据段选择子。实验中的两个数据段均在 LDT 表中声明,则描述符对应的段选择子应该标记出来。

TIL EQU 04h
DATAS_SEL = DATAS-LDT+TIL
DATAD_SEL = DATAD-LDT+TIL

(3) 目标数据段描述符的段基地址给定为 00003000H。

在全局描述表 GDT 定义的过程中,首先需要定义一个空的描述符 DUMMY,作为定义的开始,然后再定义其它描述符。

本实验可实现将一个数据段中的数据搬移到另一个数据段所指定的地址空间中(指定地址给定为 00003000H)。源数据段 LDATA 值为 11H,22H,33H,44H,55H,66H,77H,88H,在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

3. 实验步骤

- (1) 编写实验程序(例程文件名为: P1-2.ASM)。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序,待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入:

D0000: 3000

然后回车,即可看到实验运行结果,将数据传输到了给定的地址空间中。

2.2 特权级变换实验

2.2.1 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。

2.2.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.2.3 任务内无特权级变换的转移实验

1. 预备知识

任务内无特权级变换的转移比较简单，在需要发生转移时，可用 JUMP、CALL、RETF 等指令进行相应段的转移。在用到 CALL 指令时，必须为之准备一个堆栈，以使得在调用 CALL 时，系统对 CALL 当前的程序运行点进行压栈保存，在遇到 RETF 返回指令时，进行弹栈返回。

任务内段间调用指令 CALL16 宏定义如下：

```
CALL16    MACRO    SELECTOR, OFFSET
           DB        9AH                ; 操作码
           DW        OFFSET            ; 16 位偏移量
           DW        SELECTOR          ; 段值或段选择子
           ENDM
```

JUMP16 宏定义如下：

```
JUMP16    MACRO    SELECTOR, OFFSET
           DB        0EAH              ; 操作码
           DW        OFFSET            ; 16 位偏移量
           DW        SELECTOR          ; 段值或段选择子
           ENDM
```

通常情况下，段间返回指令 RETF 与段间调用指令 CALL 对应。在利用段间调用指令 CALL 以任务内无特权级变换的方式转移到某个子程序后，在子程序内利用段间返回指令 RETF 以任务内无特权级变换的方式返回主程序。由于调用时无特权级变换，所以返回时也无特权级变换。

2. 实验内容一

本实验需要在程序中安排 3 个相同特权级的段 (D1, D2, D3)，并在运行时能够实现从 D1 转移到 D2，从 D2 转移到 D3，在 D3 中将源数据段中的数据传输到目标数据段中，利用 CALL、RETF 指令来实现。

实验程序在 DSEG 段中描述 GDT 中的描述符。先定义一个空的描述符表明 GDT 表的开始，然后定义主程序段和 LDT 描述符。在 DSEG 段后，用 DSEG1 段来描述 LDT 中的描述符，其中包括源数据段描述符和目标数据段描述符。

为了体现任务的特性，只将主程序段描述符和 LDT 段描述符安放在 GDT 中，其余的所有代码段和数据段均放置在 LDT 中。由于实验内容中所有要求完成的都是任务内相同特权级段之间的转移，所以不会涉及到堆栈的切换，则在实验程序中只需要建立一张局部描述符表，而不需要使用任务状态段。

本实验可实现将一个数据段中的数据搬移到另一个数据段所指定的地址空间中（指定地址给定为 00003000H）。源数据段 TDATA 值为 11H,22H,33H,44H,55H,66H,77H,88H，在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

3. 实验步骤

- (1) 编写实验程序（例程文件名为：P2-1.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序，待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入：

D0000: 3000

然后回车，即可看到实验运行结果，将数据传输到了给定的地址空间中。

4. 实验内容二

在保护模式下采用 JMP 和 RETF 指令来实现实验内容一的实验要求。

5. 实验步骤

- (1) 编写实验程序（例程文件名为：P2-2.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序，待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入：

D0000: 3000

然后回车，即可看到实验运行结果，将数据传输到了给定的地址空间中。

2.2.4 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送到目标数据段中，利用 JUMP、CALL、RETF 指令来实现。

实验程序在 TDATASEG 段中描述 GDT 中的描述符。先定义一个空的描述符表明 GDT 表的开始，然后定义主程序段、LDT 描述符和任务状态段 TSS 描述符。在 TDATASEG 段后，用 LDTSEG 段来描述 LDT 中的描述符，其中包括源数据段描述符和目标数据段描述符。

为了体现任务的特性，只将主程序段描述符、LDT 段描述符和任务状态段 TSS 描述符安放在 GDT 中，其余的所有代码段和数据段均放置在 LDT 中。另外在 LDT 表中还定义了一个 TOCODE0 的任务调用门描述符，在由 3 级代码段向 0 级代码段转移时必须通过此调用门进行转移。

本实验可实现将一个数据段中的数据搬运到另一个数据段所指定的地址空间中（指定地址给定为 00003000H）。源数据段 DTEST 值为 11H,22H,33H,44H,55H,66H,77H,88H，在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

3. 实验步骤

(1) 编写实验程序（例程文件名为：P2-3.ASM）。

(2) 编译、链接无误后装入系统。

(3) 点击 RUN 运行程序，待程序运行停止。

(4) 查看运行结果。

在输出区的调试栏输入：

D0000: 3000

然后回车，即可看到实验运行结果，将数据传输到了给定的地址空间中。

2.3 任务切换实验

2.3.1 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。

2.3.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.3.3 使用 JMP 指令实现任务切换实验（无特权级）

1. 预备知识

任务之间进行切换。每个任务都有一个任务状态段 TSS，用于保存任务的有关信息，在任务内变换特权级和任务切换时，要使用这些信息。为了控制任务内发生特权变换的转移，为了控制任务切换，一般要通过控制门进行这些转移。

任务状态段 TSS 是保存一个任务重要信息的特殊段。任务状态段描述符用于描述这样的系统段。任务状态段寄存器 TR 的可见部分含有当前任务状态段描述符的选择子，TR 的不可见部分含有当前任务状态段的段基地址和段界限等信息。

TSS 在任务切换过程中起着重要的作用，通过它实现任务的挂起与恢复。所谓任务切换是指，挂起当前正在执行的任务，恢复另一个任务的执行。在任务切换过程中，首先，处理器中各寄存器的当前值被自动地保存到 TR 所指定的 TSS 中；然后，下一任务的 TSS 的选择子被装入 TR；最后从 TR 所指定的 TSS 中取出各寄存器的值送到处理器的各寄存器中。由此可见，通过在 TSS 中保存任务现场各寄存器状态的完整映像，实现任务的切换。

任务状态段 TSS 的基本格式由 104 字节组成。这 104 字节的基本格式是不可改变的，但在此之外系统软件还可定义若干附加信息。基本的 104 字节可分为链接字段区域、内存堆栈指针区域、地址映射寄存器区域、寄存器保存区域和其它字段等五个区域。

用结构类型定义 TSS 如下：

```
TSS          STRUC
    TRLINK    DW          0          ; 链接字段
              DW          0          ; 不使用,置为 0
    TRESP0    DD          0          ; 0 级堆栈指针
    TRSS0     DW          0          ; 0 级堆栈段寄存器
              DW          0          ; 不使用,置为 0
    TRESP1    DD          0          ; 1 级堆栈指针
    TRSS1     DW          0          ; 1 级堆栈段寄存器
              DW          0          ; 不使用,置为 0
    TRESP2    DD          0          ; 2 级堆栈指针
    TRSS2     DW          0          ; 2 级堆栈段寄存器
```

	DW	0	; 不使用,置为 0
TRCR3	DD	0	; CR3
TREIP	DD	0	; EIP
TREFLAG	DD	0	; EFLAGS
TREAX	DD	0	; EAX
TRECX	DD	0	; ECX
TREDX	DD	0	; EDX
TREBX	DD	0	; EBX
TRESP	DD	0	; ESP
TREBP	DD	0	; EBP
TRESI	DD	0	; ESI
TREDI	DD	0	; EDI
TRES	DW	0	; ES
	DW	0	; 不使用,置为 0
TRCS	DW	0	; CS
	DW	0	; 不使用,置为 0
TRSS	DW	0	; SS
	DW	0	; 不使用,置为 0
TRDS	DW	0	; DS
	DW	0	; 不使用,置为 0
TRFS	DW	0	; FS
	DW	0	; 不使用,置为 0
TRGS	DW	0	; GS
	DW	0	; 不使用,置为 0
TRLDTR	DW	0	; LDTR
	DW	0	; 不使用,置为 0
TRTRIP	DW	0	; 调试陷阱标志(只用位 0)
TRIOMAP	DW	\$+2	; 指向 I/O 许可位图区的段内偏移
TSS	ENDS		

装载 TSS 表指令如下:

```
MOV     AX, TSS_SEL
LTR     AX
```

2. 实验内容

本实验要求使用 JUMP 指令实现任务切换。实验由主程序进入,先装入任务 0,再跳到任务 1 完成一段数据的传输。当任务 1 的工作完成后,进行任务切换,返回任务 0。

任务的切换可以通过 TSS 段和任务门完成,在实验中,从任务 0 切换到任务 1 时使用了 TSS 段,从任务 1 切换到任务 0 时使用了任务门。为了实现切换,程序需要为每个任务建立一个 TSS 段,并为每个任务建立自己的代码段、数据段和堆栈段等。

本实验可实现将一个数据段中的数据搬运到另一个数据段所指定的地址空间中(指定地址

给定为 00003000H)。源数据段 DTEST 值为 11H,22H,33H,44H,55H,66H,77H,88H, 在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

装载任务 TSS 到 TR 寄存器中可用下面两条指令:

```
MOV     AX, TSS0_SEL
LTR     AX
```

3. 实验步骤

- (1) 编写实验程序(例程文件名为: P3-1.ASM)。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序,待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入:

D0000: 3000

然后回车,即可看到实验运行结果,将数据传输到了给定的地址空间中。

2.3.4 使用 CALL 指令实现任务实验(有特权级)

1. 实验内容

本实验要求使用 CALL 指令实现任务切换。实验安排了三个有特权级的任务段分别是:任务 1(特权级为 0 级)、任务 2(特权级为 0 级)、任务 3(特权级为 0 级)。

实验由主程序进入,装入任务 1,再跳到任务 2,由任务 2 再跳到任务 3 完成一段数据的传输。当任务 3 的工作完成后,再由任务 3 返回到任务 2,再由任务 2 返回到任务 1 到主程序。实验中涉及了 3 个任务,所以需要为这三个任务分别建立 3 个 TSS,并且为每个任务建立自己的代码段,数据段以及堆栈段。

在主程序段中应首先使用 LTR 指令将特权级为 0 级的任务 1 TSS 装入 TR 寄存器。在任务 1 内部执行过程中,使用 CALL16 宏直接指向任务 2 的 TSS 表进行任务切换。

在任务 2 内部执行过程中,使用 CALL16 宏通过任务门转移到任务 3,即通过任务门从一个 0 级代码段转移到了一个 3 级代码段。在转移过程中,处理器首先将各个寄存器的内容保存到任务 2 的任务状态段,再将任务 3 的任务状态段内容装入 CPU 的各寄存器。任务 3 中的 CS 指向的是一个 3 级程序段,则相应使用的 SS 也是 3 级。于是在任务切换的过程中也实现了代码段间有特权级的切换。

切换到任务 3 完成数据传输后,再通过 IRETD 段间返回指令,使得从当任务 3 返回到任务 2 再返回到任务 1 直到主程序。

本实验可实现将一个数据段中的数据搬运到另一个数据段所指定的地址空间中(指定地址给定为 00003000H)。源数据段 DTEST 值为 11H,22H,33H,44H,55H,66H,77H,88H, 在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

2. 实验步骤

- (1) 编写实验程序(例程文件名为: P3-2.ASM)。

- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序，待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入：

D0000: 3000

然后回车，即可看到实验运行结果，将数据传输到了给定的地址空间中。

2.4 中断与异常处理实验

2.4.1 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。

2.4.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

2.4.3 用中断门、陷阱门实现中断/异常处理实验

80X86 把中断分为外部中断和内部中断两大类。把外部中断称为“中断”，把内部中断称为“异常”。

1. 预备知识

(1) 中断描述表 IDT

与 8086/8088 一样，在响应中断或者处理异常时，80X86 根据中断向量号转向对应的处理程序。但是，在保护模式下 80X86 不再使用实模式下的中断向量表，而是使用中断描述符表 IDT。在保护模式下，80X86 把中断向量号作为中断描述符表 IDT 中描述符的索引，而不再是中断向量表中的中断向量的索引。

像全局描述符表 GDT 一样，在整个系统中，中断描述符表 IDT 只有一个。中断描述符表寄存器 IDTR 指示 IDT 在内存中的位置。由于 80X86 只识别 256 个中断向量号，所以 IDT 最大长度是 2K。

中断描述符表 IDT 所含的描述符只能是中断门、陷阱门和任务门。

(2) 通过中断门、陷阱门的转移

如果中断向量号所指示的门描述符是 386 中断门或 386 陷阱门，那么控制转移到当前任务的一个处理程序过程，并且可以变换特权级。与通过调用门的 CALL 指令一样，从中断门或陷阱门中获取指向处理程序的 48 位全指针。其中，16 位选择子是对应处理程序代码段的选择子，它指示 GDT 或 LDT 中的描述符；32 位偏移指示处理程序入口点在代码段内的偏移。

通过中断门、陷阱门的转移，由处理器硬件自动进行。

LIDT QWORD PTRVIDTR 表示装载中断描述符表寄存器 IDTR。

2. 实验内容

本实验要求通过 INT 20H 调用 20H 号中断/异常处理程序实现数据传输，其中调用 INT 指令的代码段为 0 级代码段，IDT 中 20H 号门描述符门为中断门。

下面定义一个 IDTSEG 表如下：

```
IDTSEG    SEGMENT    PARA    USE16
```

```

IDT      LABEL    BYTE
          REPT     32          ; 从 00H---1FH 的 32 个中断门描述符
          GATE     <,,,AT386IGate,>
          ENDM
INT20    Gate     <0,ICode_Sel,,AT386IGate,>
          ; 对应 20H 号异常/中断处理程序段的中断门描述符
          REPT     256-33      ; 从 21H---FFH 的 223 个中断门描述符
          GATE     <,,,AT386IGate,>
          ENDM
IDTLEN   =        $-IDT
IDTSEG   ENDS

```

本实验可实现将一个数据段中的数据搬移到另一个数据段所指定的地址空间中（指定地址给定为 00003000H）。源数据段 DTEST 值为 11H,22H,33H,44H,55H,66H,77H,88H, 在程序运行结束后会将此内容传输到地址为 00003000H 开始的地址空间中。

3. 实验步骤

- (1) 编写实验程序（例程文件名为：P4-1.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序，待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入：

D0000: 3000

然后回车，即可看到实验运行结果，将数据传输到了给定的地址空间中。

2.4.4 用任务门实现中断/异常处理实验

如果中断向量号所指示的门描述符是任务门描述符，那么控制转移到一个作为独立的任务方式出现的处理程序。任务门中含有 48 位全指针。16 位选择子是指向描述符对应处理程序任务的 TSS 段的选择子，也即该选择子指示一个可用的 386TSS。通过任务门的转移与通过任务门到一个可用的 386TSS 的 CALL 指令的转移很相似。

在响应中断或处理异常时，使用任务门可提供一个处理程序任务的自动调度。这种任务调度由硬件直接执行。

1. 实验内容

本实验是通过调用任务门实现中断/异常的转移，也是通过 INT 20H 调用 20H 号异常处理程序实现数据传输，其中调用 INT 指令的代码段为 0 级代码段，IDT 中 20H 号门描述符为任务门描述符。

```

INT20    Gate     <0,TSS3_Sel,,ATTASKGate,>
          ; 对应 20H 号异常/中断处理程序段的任务门描述符

```

本实验可实现将一个数据段中的数据搬运到另一个数据段所指定的地址空间中（指定地址给定为 00004000H）。源数据段 DTEST 值为 11H,22H,33H,44H,55H,66H,77H,88H, 在程序运行结束后会将此内容传输到地址为 00004000H 开始的地址空间中。

2. 实验步骤

- (1) 编写实验程序（例程文件名为：P4-2.ASM）。
- (2) 编译、链接无误后装入系统。
- (3) 点击 RUN 运行程序，待程序运行停止。
- (4) 查看运行结果。

在输出区的调试栏输入：

D0000: 4000

然后回车，即可看到实验运行结果，将数据传输到了给定的地址空间中。

(三) 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。

3.1 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二维线性地址。

3.1.1 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：

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

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。

3.1.2 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 3.1 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 3.1 存储段描述符格式

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 3.2:

(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 3.2 存储段描述符属性

数据 段 类 型	类型值	说 明	代 码 段 类 型	类型值	说 明
	0	只读		8	只执行
	1	只读, 已访问		9	只执行, 已访问
	2	读/写		A	读/执行
	3	读/写, 已访问		B	读/执行, 已访问
	4	只读, 向低扩展		C	只执行, 一致码段

5	只读, 向低扩展, 已访问	D	只执行, 一致码段, 已访问
6	读/写, 向低扩展	E	读/执行, 一致码段
7	读/写, 向低扩展, 已访问	F	读/执行, 一致码段, 已访问

(6) D 位是一个很特殊的位, 在描述可执行段、向下扩展数据段或由 SS 寄存器寻址的段(通常是堆栈段)的三种描述符中的意义各不相同。

在描述可执行段的描述符中, D 位决定了指令使用的地址及操作数所默认的大小。D=1 表示默认情况下指令使用 32 位地址及 32 位或 8 位操作数, 这样的代码段也称为 32 位代码段; D=0 表示默认情况下, 使用 16 位地址及 16 位或 8 位操作数, 这样的代码段也称为 16 位代码段, 它与 80286 兼容。可以使用地址大小前缀和操作数大小前缀分别改变默认的地址或操作数的大小。

在向下扩展数据段的描述符中, D 位决定段的上部边界。D=1 表示段的上部界限为 4G; D=0 表示段的上部界限为 64K, 这是为了与 80286 兼容。

在描述由 SS 寄存器寻址的段描述符中, D 位决定隐式的堆栈访问指令(如 PUSH 和 POP 指令)使用何种堆栈指针寄存器。D=1 表示使用 32 位堆栈指针寄存器 ESP; D=0 表示使用 16 位堆栈指针寄存器 SP, 这与 80286 兼容。

(7) AVL 位是软件可利用位。80386 对该位的使用未做规定, Intel 公司也保证今后开发生产的处理器只要与 80386 兼容, 就不会对该位的使用做任何定义或规定。

此外, 描述符内第 6 字节中的位 5 必须置为 0, 可以理解成是为以后的处理器保留的。

2. 存储段描述符的结构类型表示

根据存储段描述符的结构, 可定义如下的汇编语言描述符结构类型:

```
DESC      STRUC
LIMITL    DW      0      ;段界限低 16 位
BASEL     DW      0      ;基地址低 16 位
BASEM     DB      0      ;基地址中间 8 位
ATTRIB    DB      0      ;段属性
LIMITH    DB      0      ;段界限的高 4 位(包括段属性的高 4 位)
BASEH     DB      0      ;基地址的高 8 位
DESC ENDS
```

利用结构类型 DESC 能方便地在程序中说明存储段描述符。例如: 下面的描述符 DATAS 描述一个可读写的有效(存在的)数据段, 基地址是 100000H, 以字节为单位的界限是 0FFFFH, 描述符特权级 DPL=3。

```
DATAS      DESC      <0FFFFH,,10H,0F2H,,>
```

再如: 下述描述符 CODEA 描述一个只可执行的有效的 32 位代码段, 基地址是 12345678H, 以 4K 字节为单位的段界限值是 10H(以字节为单位的界限是 10FFFH), 描述符特权级 DPL=0。

```
CODEA      DESC      <10H,5678H,34H,98H,0C0H,12H>
```

3.1.3 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：

DESCTAB	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：

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

3.1.4 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 3.1 所示。

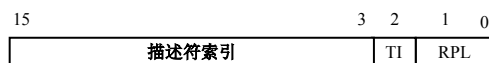


图 3.1 选择子格式

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。

3.1.5 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 3.3 所示。其中，32 位段基地址直接取自描述符，32 位的段界限取自描述符中 20 位的段界限，并根据描述符属性中的粒度位转换成以字节为单位。其它十个特性根据描述符中的属性而定，“Y”表示“是”，“N”表示“否”，“R”表示必须可读，“W”表示必须可写，“P”表示必须存在，“D”表示根据描述符中属性而定。

图 3.3 高速缓冲寄存器内容

段描述符	段寄存器	段基址	段界限	段属性							
				存	特	己	粒	扩	可	可	可

高速缓冲寄存器的内容		地址	限	在性	权级	取取	度	展方向	读性	写性	执行	栈大小	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。

3.2 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。

3.2.1 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。

3.2.2 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 3.2 显示了由页目录表和页表构成的页映射表结构。从图 3.2 中可见，控制寄存器 CR3 指定页目录表；页目录表可以指定 1K 个页表，这些页表可以分散存放在任意的物理页中，而不需要连续存放；每张页表可以指定 1K 个物理地址空间的页，这些物理地址空间的页可以任意地分散在物理地址空间中。需要注意的是，存储页目录表和页表的基地址是对齐在 4K 字节边界上的。

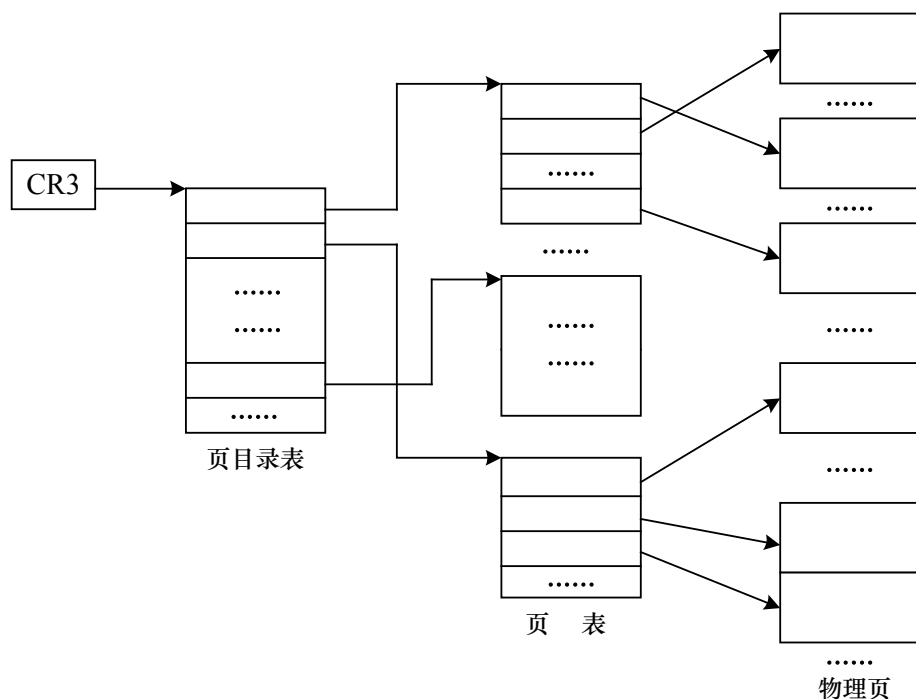


图 3.2 页映射表结构

2. 表项格式

页目录表和页表中的表项都采用如图 3.3 所示的格式。从图中可见，最高 20 位(位 12—位 31)包含物理地址空间页的页码，也就是物理地址的高 20 位。低 12 位包含页的属性。图 3.3 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 3.3 页目录表或页表的表项格式

3. 线性地址到物理地址的转换

分页管理机制通过上述页目录表和页表实现 32 位线性地址到 32 位物理地址的转换。控制寄存器 CR3 的高 20 位作为页目录表所在物理页的页码。首先把线性地址的最高 10 位(即位 22 至位 31)作为页目录表的索引, 对应表项所包含的页码指定页表; 然后, 再把线性地址的中间 10 位(即位 12 至位 21)作为所指定的页目录表中的页表项的索引, 对应表项所包含的页码指定物理地址空间中的一页; 最后, 把所指定的物理页的页码作为高 20 位, 把线性地址的低 12 位不加改变地作为 32 位物理地址的低 12 位。

为了避免在每次存储器访问时都要访问内存中的页表, 以便提高访问内存的速度, 80386 处理器的硬件把最近使用的线性—物理地址转换函数存储在处理器内部的页转换高速缓存中。在访问存储器页表之前总是先查阅高速缓存, 仅当必须的转换不在高速缓存中时, 才访问存储器中的两级页表。页转换高速缓存也称为页转换查找缓存, 记为 TLB。

在分页机制转换高速缓存中的数据与页表中数据的相关性, 不是由 80386 处理器进行维护的, 而必须由操作系统软件保存, 也就是说, 处理器不知道软件什么时候会修改页表, 在一个合理的系统中, 页表只能由操作系统修改, 操作系统可以直接地在软件修改页表后通过刷新高速缓存来保证相关性。高速缓存的刷新通过装入处理器控制寄存器 CR3 完成, 实际过程可能用如下的两条指令实现:

```
MOV    EAX, CR3
MOV    CR3, EAX
```

一个重要的修改页表项的特殊情况不需要对页转换高速缓存刷新, 这种情况是指修改不存在表项的任一部分, 即使 P 位本身从 P=0 改变为 P=1 时也一样, 因为无效的表项不会存入高速缓存。因此, 当无效的表项被改变时, 不需要刷新高速缓存。这表明在从磁盘上读入一页使其存在时, 不必刷新高速缓存。

在一个多处理器系统中, 必须特别注意是否在一个处理器中执行的程序, 会改变可能由另外的处理器同时访问的页表。在 80X86 处理器中, 每当要更新页表项并设置 D 位和 A 位时, 通过使用不可分的读/修改/写周期支持多处理器的配置。对于页表项的软件更新需要借助于使用 LOCK 前缀, 从而保证修改页表的指令工作在不可分的读/修改/写周期中。在改变一个可能由另外的处理器使用的页表之前, 最好使用一条加锁的 AND 指令在一个不可分的操作中将 P 位清除为 0, 然后, 该表项可根据要求进行修改, 并随后把 P 位置成 1 而使表项成为可用。当修改页表项时必须及时通知(通常使用中断方式)系统中该表项已被高速缓存的所有处理器刷新各自的页转换高速缓存, 以撤消该表项的旧拷贝。在表项的旧拷贝被刷新之前, 各处理器仍可继续访问旧的页, 并可以设置正被修改的表项的 D 位。如果这样做引起表项修改失败, 则分页机制高速缓存最好在标记为不存在之后, 并在对表项进行另外的修改之前进行刷新。

4. 不存在的页表

采用上述页映射表结构, 存储全部 1K 张页表需要 4M 字节, 此外还需要 4K 字节用于存储页目录表。这样的两级页映射表似乎反而比单一的整张页映射表多占用 4K 字节。其实不然, 事实上不需要在内存中存储完整的两级页映射表。两级页映射表结构中对于线性地址空间中不存在的或未使用的部分不必分配页表。除必须给页目录表分配物理页外, 仅当在需要时才给页表分配物理页, 于是页映射表的大小就对应于实际使用的线性地址空间大小。因为任何一个实际运行的程序使用的线性地址空间都远小于 4G 字节, 所以用于分配给页表的物理页也远小于 4M 字节。

页目录表项中的存在位 P 表明对应页表是否有效。如果 $P=1$, 表明对应页表有效, 可利用它进行地址转换; 如果 $P=0$, 表明对应页表无效。如果试图通过无效的页表进行线性地址到物理地址的转换, 那么将引起页故障。因此, 页目录表项中的属性位 P 使得操作系统只需给覆盖实际使用的线性地址范围的页表分配物理页。

页目录表项中的属性位 P 可用于把页表存储在虚拟存储器中。当发生由于所需页表无效而引起的页故障时, 页故障处理程序再申请物理页, 从磁盘上把对应的页表读入, 并把对应页目录表项中的 P 位置 1。换言之, 可以当需要时才为所要的页表分配物理页。这样页表占用的物理页数量可降到最小。

5. 页的共享

由上述页映射表结构可见, 分页机制没有全局页和局部页的规定。每一个任务可使用自己的页映射表独立地实现线性地址到物理地址的转换。但是, 如果使每一个任务所用的页映射表具有部分相同的映射, 那么也就可以实现部分页的共享。

常用的实现页共享的方法是线性地址空间的共享, 也就是不同任务的部分相同的线性地址空间的映射信息相同, 具体表现为部分页表相同或页表内的部分表项的页码相同。例如, 如果任务 A 和任务 B 分别使用的页目录表 A 和页目录表 B 内的第 0 项中的页码相同, 也就是页表 0 相同, 那么任务 A 和任务 B 的 00000000H 至 003FFFFFFH 线性地址空间就映射到相同的物理页。再如, 任务 A 和任务 B 使用的页表 0 不同, 但这两张页表内第 0 至第 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。

3.2.3 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 3.4 列出了上述属性位 R/W 和 U/S 所确定的页级保护下，用户级程序和系统级程序分别具有的对用户级页和系统级页进行操作的权限。

表 3.4 页级保护属性

U/S	R/W	用户级访问权限	系统访问权限
0	0	无	读/写/执行
0	1	无	读/写/执行
1	0	读/执行	读/写/执行
1	1	读/写/执行	读/写/执行

由表 3.4 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得。表 3.5 列出了组合计算前后的保护属性位的值，组合计算是“与”操作。

表 3.5 组合页保护属性

目录表项 U/S	页表项 U/S	组合 U/S	目录表项 R/W	页表项 R/W	组合 R/W
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	1	1	1

正如在 80386 地址转换机制中分页机制在分段机制之后起作用一样，由分页机制支持的页级保护也在由分段机制支持的段级保护之后起作用。先测试有关的段级保护，如果启用分页机制，那么在检查通过后，再测试页级保护。如果段的类型为读/写，而页规定为只允许读/执行，那么不允许写；如果段的类型为只读/执行，那么不论页保护如何，也不允许写。

页级保护的检查是在线性地址转换为物理地址的过程中进行的，如果违反页保护属性的规定，对页进行访问(读/写/执行)，那么将引起页异常。

2. 对虚拟存储器的支持

页表项中的 P 位是支持采用分页机制虚拟存储器的关键。P=1，表示表项指定的页存在于物理存储器中，并且表项的高 20 位是物理页的页码；P=0，表示该线性地址空间中的页所对应的物理地址空空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。

3.2.4 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 3.4 所示。

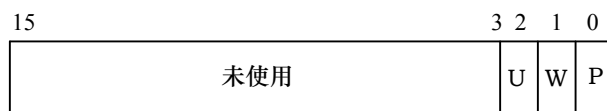


图 3.4 页故障出错格式

其中, U 位表示引起故障程序的特权级, U=1 表示用户特权级(特权级 3), U=0 表示系统特权级(特权级 0、1 或 2); W 位表示访问类型, W=0 表示读/执行, W=1 表示写; P 位表示异常类型, P=0 表示页不存在故障, P=1 表示保护故障。页故障的响应处理模式同其它故障一样。

(四) 保护模式下的存储器扩展及其应用实验

4.1 无分页机制的存储器扩展实验

4.1.1 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。

4.1.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

4.1.3 实验内容

在保护模式下，存储器的寻址为：段选择子加偏移，得到线性地址，由于本实验不启动分页机制，所以线性地址就等于实际内存的物理地址。

1. 利用保护模式机制，实现对外部存储器的扩展（本实验所定义的外部存储器空间为：00080000H~0008FFFFH）。

2. 利用保护模式的段管理机制，编程实现从地址空间 00080000H 开始将一段数据传送至地址空间从 00080028H 开始的存储器区域中，然后将它的值在屏幕上显出来。

本实验需要接线：将存储器的地址、数据、读写、字节使能及片选分别接到 80X86 系统扩展应用总线相对应的信号引脚上。如图 4.1：

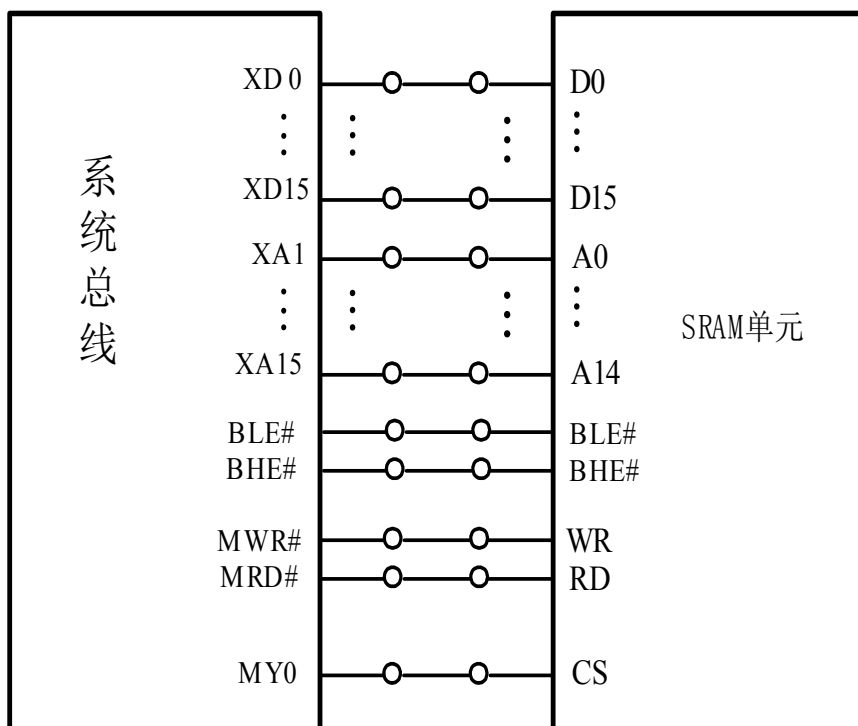


图 4.1 存储器扩展实验接线图

4.1.4 实验步骤

1. 编写实验程序（例程文件名为：PMEM1.ASM），按如图 4.1 进行实验接线。
2. 编译、链接无误后装入系统。
3. 点击 RUN 运行程序，待程序运行停止。
4. 查看运行结果。

在输出区的显示栏显示如下信息：

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ

5. 分析实验结果，理解分段的存储器管理机制。

4.2 具有分页机制的存储器扩展实验

4.2.1 实验目的

1. 掌握 80X86 微机保护模式下分页机制的存储器扩展方法。
2. 掌握 80X86 微机保护模式下分页机制对扩展存储器的操作方法。

4.2.2 实验设备

PC 机一台，TDX-PITE 实验装置一套。

4.2.3 实验内容

在保护模式下，启动分页机制的存储器管理后，线性地址不再等于物理地址，线性地址要经过分页机制转换才能成为物理地址。要建立分页机制，必须先创建页目录表及页表并对其在保护模式下进行初始化，然后启动分页机制，启动分页机制很简单，只要把控制寄存器 CR0 中的最高位 PG 位置 1 即可，具体指令如下：

```
MOV    EAX, CR0
OR      EAX, 80000000H
MOV     CR0, EAX
JMP     SHORT  PAGE
PAGE:
```

.....

关闭分页机制也很简单，把控制寄存器 CR0 中的 PG 位清 0 即可。

本实验为了简单，只有一个任务，并且没有局部描述符和中断描述符，不允许中断，不考虑发生异常。旨在体现分页机制的工作原理及过程。

实验建立了一张页目录表和一张页表，其所在物理页的地址为：

PDT_AD=00010000H ;页目录表所在物理页的地址

PT_AD =00012000H ;页表所在物理页的地址

另外，还给定了两个地址，实验中所要用到的线性地址和物理地址：

XIANXIN_AD=00301028H ;线性地址 XIANXIN_AD

WULI_AD =00080028H ;线性地址 XIANXIN_AD 对应的物理地址

实验内容是在创建并启动分页机制后，将源数据段中的字符串“Page Is Successful!”传送到线性地址 XIANXIN_AD 中，由于是在分页机制下工作，硬件系统会根据用户初始化的要求，通过控制寄存器 CR3 和线性地址 XIANXIN_AD 中的内容，将线性地址 XIANXIN_AD 映射到物理地址 WULI_AD 中，然后将物理地址 WULI_AD 中的内容在屏幕上输出显示。可参考图 4.2：

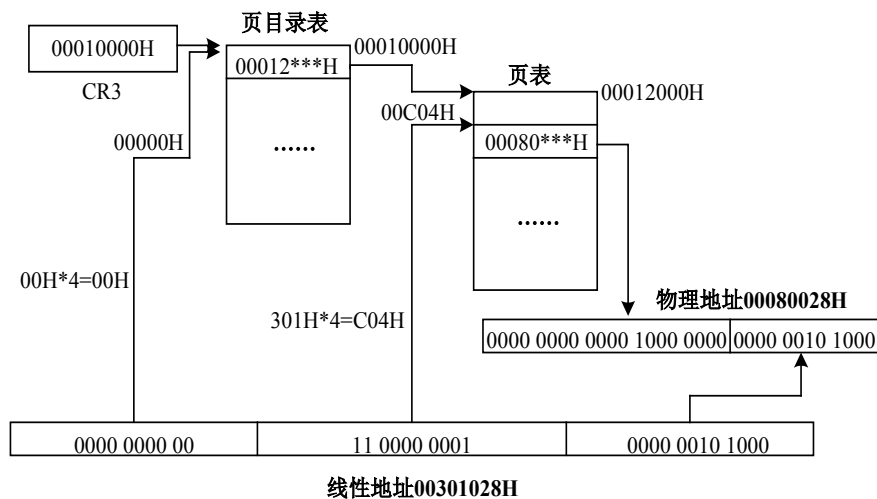


图 4.2 线性地址到物理地址的转换过程

其中，图 4.2 中的 “***” 表示页表的属性。

本实验需要接线：将存储器的地址、数据、读写、字节使能及片选分别接到 80X86 系统扩展应用总线相对应的信号引脚上。如图 4.1：

4.2.4 实验步骤

1. 编写实验程序（例程文件名为：PMEM2.ASM），按如图 4.1 进行实验接线。
2. 编译、链接无误后装入系统。
3. 点击 RUN 运行程序，待程序运行停止。
4. 查看运行结果。

在输出区的显示栏显示如下信息：

Page Is Successful!

5. 分析实验结果，理解分页的存储器管理机制。

附录 1 TDX-PITE 联机软件使用说明

附 1.1 菜单功能

1. 文件菜单项

文件菜单如附图 1-1 所示。

(1) 新建 (N) : 用此命令在 TDX-PITE 中建立一个新文档。

(2) 打开 (O) : 用此命令在窗口中打开一个现存的文档。

(3) 关闭 (C) : 用此命令来关闭当前活动文档。

(4) 保存 (S) : 用此命令将当前活动文档保存到它的当前文件名和目录下。当您第一次保存文档时, TDX-PITE 显示另存为对话框以便您命名您的文档。

(5) 另存为 (A) : 用此命令来保存并命名活动文档。

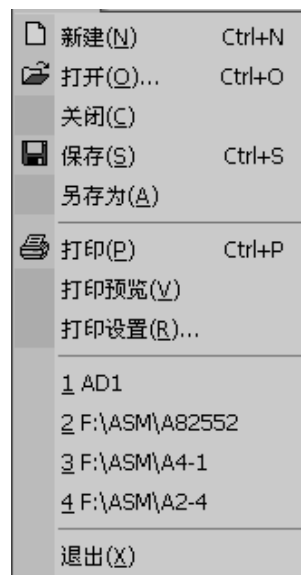
(6) 打印 (P) : 用此命令来打印一个文档。

(7) 打印预览 (V) : 用此命令来打印当前显示活动文档。

(8) 打印设置 (R) : 用此命令来选择连接的打印机及其设置。

(9) 最近浏览文件: 通过此列表, 直接打开最近打开过的文件。

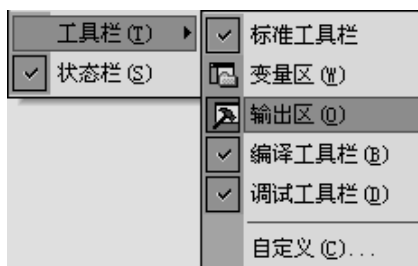
(10) 退出 (X) : 用此命令来结束 TDX-PITE 的运行阶段。TDX-PITE 会提示您保存尚未保存的改动。



附图 1-1 文件菜单

2. 查看菜单项

查看菜单如附图 1-2 所示。



附图 1-2 查看菜单

(1) 工具栏 (T) : 显示或隐藏工具栏

(2) 状态栏 (S) : 显示或隐藏状态栏

(3) 工具栏

a、标准工具栏：用此命令可显示和隐藏标准工具栏。标准工具栏包括了 TDX-PITE 中一些最普通命令的按钮，如文件打开。在工具栏被显示时，一个打勾记号出现在该菜单项目的旁边。

b、变量区 (W)：用此命令可显示和隐藏寄存器/变量/堆栈区。

c、输出区 (O)：用此命令可显示和隐藏输出区。

d、编译工具栏 (B)：用此命令可显示和隐藏编译工具栏。

e、调试工具栏 (D)：用此命令可显示和隐藏调试工具栏。

f、自定义 (C)：见自定义功能。

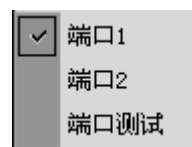
3. 端口菜单项

端口菜单如附图 1-3 所示。

(1) 端口 1：此命令用来选择串口 1 进行联机通讯，该命令会对串口 1 进行初始化操作，并进行联机测试，报告测试结果。

(2) 端口 2：此命令用来选择串口 2 进行联机通讯，该命令会对串口 2 进行初始化操作，并进行联机测试，报告测试结果。

(3) 端口测试：此命令用来对当前选择的串口进行联机通讯测试，并报告测试结果。



附图 1-3 端口菜单

4. 编译菜单项

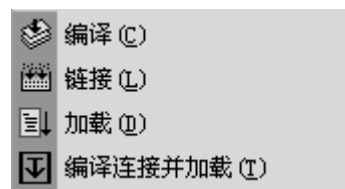
编译菜单如附图 1-4 所示。

(1) 编译 (C)：编译当前活动文档中的源程序，在源文件目录下生成目标文件。

(2) 链接 (L)：链接编译生成的目标文件，在源文件目录下生成可执行文件。

(3) 加载 (D)：把链接生成的可执行文件加载到下位机。加载成功，输出区显示“加载成功！”。

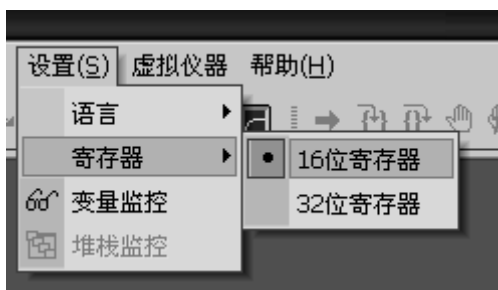
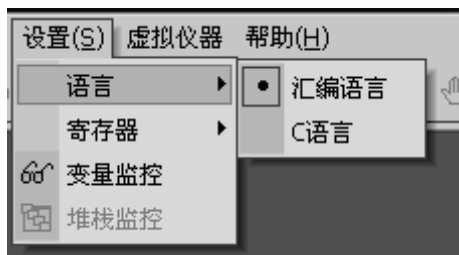
(4) 编译链接并加载 (T)：依次执行编译、链接和加载。



附图 1-4 编译菜单

5. 设置菜单

设置菜单如附图 1-5 所示。



附图 1-5 设置菜单

(1) 语言：设置语言环境

汇编语言：设置编译环境为汇编语言环境。此时可编辑、编译和链接 IBM-PC 汇编语言源

程序。

C 语言：设置编译环境为 C 语言环境。此时可编辑、编译和链接 C 语言源程序。由于监控目前不支持浮点运算，故 C 语言程序中不应该出现浮点运算，如果 C 语言程序中出现浮点运算，链接时会出现错误。

(2) 寄存器：设置寄存器格式

16 位寄存器：设置成 16 位寄存器，可观察到 16 位寄存器的变化。

32 位寄存器：设置成 32 位寄存器，可观察到 32 位寄存器的变化。

(3) 变量监控：加载成功后才可用此按钮。系统只能监视全局变量。在汇编语言源文件中，数据段定义的变量并不是全局变量，因此数据段定义的变量并不出现在上图所示的对话框的左边列表，要想监视这些变量，必须使它们成为全局变量，使一个变量成为全局变量的方法是用关键字 PUBLIC 在源程序的最前面声明之。

(4) 堆栈监控：用于选择是否监控堆栈。

6. 调试菜单项

调试菜单如附图 1-6 所示。



附图 1-6 调试菜单

(1) 设置断点/删除断点 (B)：当前光标所在的行为当前行，如果当前行无断点则在当前行设置断点，如果当前行有断点则删除当前行的断点。源程序设置的断点数不能超过 8 个。

(2) 清除所有断点 (D)：清除源程序中设置的所有断点。

(3) 设置起点 (J)：当前光标所在的行为当前行，此命令把当前行设置为程序的起点。

(4) 单步 (T)：点击此命令使程序执行一条语句，如果是函数则进入函数内部。

(5) 跳过 (O)：点击此命令使程序执行一个函数，执行后刷新所有变量和寄存器的值。

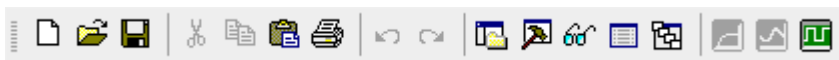
(6) 运行/运行到断点：从当前执行行开始向后运行，如果没有断点，则运行直到程序结束。如果有断点，则运行到断点后停止。


















(7) 停止：发送此命令使程序停止运行，程序停止后刷新所有寄存器和变量。

附 1.2 工具栏功能介绍

1. 标准工具栏

标准工具栏共有十七个按钮，如下图所示。







- (1)  按钮：用此按钮在 TDX-PITE 中建立一个新文档。
- (2)  按钮：用此命令在一个新的窗口中打开一个现存的文档。
- (3)  按钮：用此命令将当前活动文档保存到其当前的文件名和目录下。
- (4)  按钮：用此命令将当前被选取的数据从文档中删除并放置于剪贴板上。
- (5)  按钮：用此命令将被选取的数据复制到剪切板上。
- (6)  按钮：用此命令将剪贴板上内容的一个副本插入到插入点处。。
- (7)  按钮：用此命令来打印一个文档。
- (8)  按钮：用此命令来撤消上一步编辑操作。
- (9)  按钮：用此命令来恢复撤消的编辑操作。
- (10)  按钮：用此按钮可显示和隐藏变量和寄存器区。
- (11)  按钮：用此按钮可显示和隐藏输出区。
- (12)  按钮：加载成功后才可用此按钮。点击此按钮，可进行全局变量监视。
- (13)  按钮：用此按钮可显示和隐藏 Memory 区。
- (14)  按钮：堆栈监控按钮，点击此按钮将弹出堆栈监控对话框。
- (15)  按钮：用来启动专用图形显示。
- (16)  按钮：用来启动示波器功能。
- (17)  按钮：用来启动时序观测窗功能。

2. 编译工具栏

编译工具栏共有四个按钮，其图如下：










- (1)  编译：编译当前活动文档中的源程序，在源文件目录下生成目标文件。
- (2)  链接：链接编译生成的目标文件，在源文件目录下生成可执行文件。
- (3)  加载：把链接生成的可执行文件加载到下位机。
- (4)  编译链接并加载：依次执行编译、链接和加载。

3. 调试工具栏

调试工具栏共有七个按钮，其图如下：



- (1)  设置起点：当前光标所在的行为当前行，此命令把当前行设置为程序的起点，即程序从此行开始运行，寄存器区的 CS 和 IP 的值刷新后指向此行。
- (2)  单步：点击此命令使程序执行一条语句。
- (3)  跳过：点击此命令使程序执行一个函数，执行后刷新所有变量和寄存器的值。
- (4)  设置断点/删除断点：为光标所在行设置断点或删除当前行的已有断点。源程序设置的断点数不能超过 8 个。
- (5)  清除所有断点：清除源程序中设置的所有断点。
- (6)  运行到断点/运行：从当前执行行开始向后运行，如果没有断点，则运行直到程序结束。如果有断点，则运行到断点后停止，运行到断点后再次点击此按钮，则程序从当前断点位置继续执行，直到再次遇到断点或程序结束。
- (7)  停止：发送此命令使程序停止运行，程序停止后刷新所有寄存器和变量的值。

附 1.3 专用图形显示

主要用于观察“直流电机闭环调速实验”及“温度单元或电烤箱闭环温度控制实验”的响应曲线，本界面可以观察系统的给定值、反馈值及控制量之间的响应曲线关系。

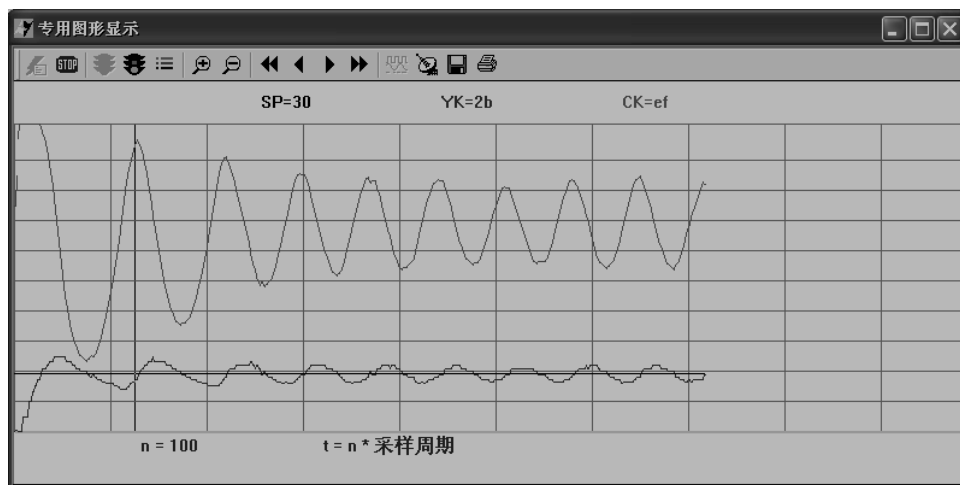
实验中给定值、反馈值都为单极性，屏幕最底端对应值为 00H，最顶端对应值为 FFH，对于时间刻度值由于采样周期不同存在以下关系：

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

控制量具有双极性，00H~7FH 为负值，80H~FFH 为正值。

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

温度闭环控制实验中，温度单元的 7805 控制范围的最佳温度范围为 50℃~70℃，不要过高。即给定值 SPEC 范围约在 14H(20℃)~46H(70℃)之间。示例程序中 SPEC = 30H 为 48℃。TS = 64H，由于 8253 OUT2 接 IRQ6 中断为 10ms，故采样周期 = $64\text{H} \times 10\text{ms} = 1\text{s}$ ；如实际刻度值 $n = 100$ ，则实际响应时间(秒) = $1 \times 100 = 100\text{s}$ 。界面如下：




1) 显示说明


(1) SP=30H：要求电机达到的转速值 48 转/秒（或要求达到的温度值 48℃）。


(2) YK=2bH：运行状态下表示电机当前的转速值 43 转/秒（或当前的温度值 43℃），暂停状态下表示指定时刻电机的转速值（或指定时刻的温度值）。


(3) CK=efH：运行状态下表示当前控制量的输出值，暂停状态下表示指定时刻控制量的输出值。控制量 CK 在正数值时转化成 PWM 输出占空比为：错误！未定义书签。100%=87.4%。


2) 工具栏功能简介


(1)  按钮：启动并运行程序。运行加载在下位机中的程序。“SP=”后显示的值是当前时刻系统的给定值，“YK=”后显示的值是当前系统的反馈值，“CK=”后显示的值是当前时刻系统控制量的值。


(2)  按钮：停止程序运行。使下位机中运行的程序停止。


(3)  按钮：暂停程序运行。在运行状态下使波形暂停显示并出现游标。


(4)  按钮：退出暂停状态，使波形继续显示，游标消失。


(5)  按钮：显示选择按钮，可选择性的选择要显示的波形。


(6)  按钮：放大波形。


(7)  按钮：缩小波形。

(8)  按钮：快速左移游标。在暂停状态下，使游标快速向左移动。“SP=”后显示的值是游标所在时刻系统的给定值，“YK=”后显示的值是游标所在时刻系统的反馈值，“CK=”后显示的值是游标所在时刻系统控制量的值。

(9)  按钮：左移游标。在暂停状态下，使游标向左移动。“SP=”后显示的值是游标所在时刻系统的给定值，“YK=”后显示的值是游标所在时刻系统的反馈值，“CK=”后显示的值是游标所在时刻系统控制量的值。


(10)  按钮：右移游标。在暂停状态下，使游标向右移动。“SP=”后显示的值是游标所在时刻系统的给定值，“YK=”后显示的值是游标所在时刻系统的反馈值，“CK=”后显示的值是游标所在时刻系统控制量的值。

(11)  按钮：快速右移游标。在暂停状态下，使游标快速向右移动。“SP=”后显示的值是游标所在时刻系统的给定值，“YK=”后显示的值是游标所在时刻系统的反馈值，“CK=”后显示的值是游标所在时刻系统控制量的值。

(12)  按钮：记录波形。点击此按钮，出现如下对话框：



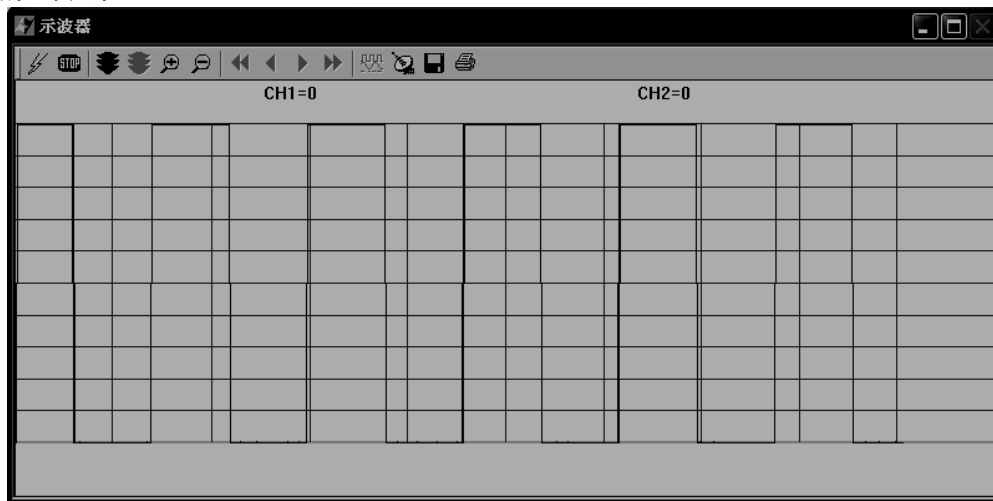
选中“图一”单选按钮，点击确定，系统会把当前时刻的波形保存到图一中，选中“图二”单选按钮，点击确定，系统会把当前时刻的波形保存到图二中，选中“图三”单选按钮，点击确定，系统会把当前时刻的波形保存到图三中。


(13)  按钮：显示保存到图一，图二和图三中的波形。


(14)  按钮：打印当前屏幕上的波形。


附 1.4 示波器


主要用于“8254 定时/计数器实验”、“D/A 转换实验”及“8251 串行接口实验”中波形的观察。





(1)  按钮：启动示波器。“CH1=”、“CH2=”后分别显示光标当前位置的采样值。

(2)  按钮：停止使下位机中运行的程序停止。

(3)  按钮：在运行状态下使能，使波形暂停显示并出现光标。

(4)  按钮：在暂停状态下使能，使波形继续显示，光标消失。

(5)  按钮：放大波形。

(6)  按钮：缩小波形。

(7)  按钮：在暂停状态下，使光标快速向左移动。


“CH1=”、“CH2=”后分别显示光标当前位置的采样值。“T = xxx”表示光标所在位置的时刻与图形最左端时刻的差值。

(8)  按钮：在暂停状态下，使光标向左缓慢移动。


(9)  按钮：在暂停状态下，使光标向右缓慢移动。





(10)  按钮：在暂停状态下，使游标快速向右移动。

(11)  按钮：点击此按钮，出现如右图所示对话框：

选中“图一”单选按钮，点击确定，系统会把当前时刻的波形保存到图一中，共可保存三副图。图形只是保存于数据缓冲中，供图形比较时使用。

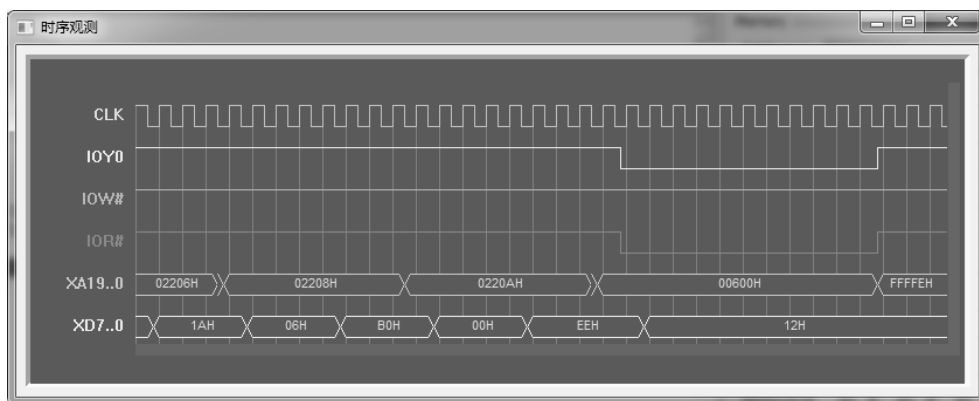
(12)  按钮：显示保存到图一，图二和图三中的波形，此时可以对几副图进行比较。

(13)  按钮：以.bmp 格式保存当前屏幕上的波形到指定文件。

(14)  按钮：打印当前屏幕上的波形。

附 1.5 时序观测窗

可对总线信号及各个指令操作的时序状态和时序关系进行准确的观测和分析，包含 I/O 操作、中断、DMA 等多种类型的观测。



在时序观测界面点击“右键”，弹出“选择观察信号”界面，可根据需要选择想要观测的信号或数据，点击确定。运行实验程序直至停止，观察各信号的时序图。

附 1.6 Debug 调试命令

TDX-PITE 软件输出区集成有 Debug 调试，点击调试标签，进入 Debug 状态，会出现命令提示符“>”，主要命令叙述如下：

A 进入小汇编

格式：A[段址:] [偏移量] ↵

A 段址:偏移量 ↵——从段址:偏移量构成的实际地址单元起填充汇编程序的目标代码；

A 偏移量 ↵——从默认的段址与给定的偏移量构成的实际地址单元起填充汇编程序目标代码；

A ↵——从默认段址:默认偏移量构成的实际地址单元起填充汇编程序的目标代码；

输入上述命令后，屏幕显示地址信息，即可输入源程序。若直接回车，则退出命令。汇编程序输入时，数据一律为十六进制数，且省略 H 后缀。[m]类操作一定要在[]之前标注 W（字）或 B（字节）。如：MOV B[2010], AX, MOV W[2010], AX。

例：在“>”提示符下键入 A2000 ↵，此时默认的段址 CS 为 0000，规定偏移量 IP 为 2000，屏幕显示与操作为：

表 4-2-1 小汇编操作示例

显示内容	键入内容
0000:2000	MOV AX, 1234 ↵
0000:2003	INC AX ↵
0000:2004	DEC AX ↵
0000:2005	JMP 2000 ↵
0000:2007	↵

B 断点设置

在系统提示符下，键入 B ↵，系统提示[i]:，等待输入断点地址。输入断点地址后回车，系统继续提示[i+1]:。若直接键入回车，则结束该命令。系统允许设置最多 10 个断点，断点的清除只能是通过系统复位或重新上电来实现。例：

表 4-2-2 B 命令示例

显示内容	键入内容
>	B ↵
[0]:	2009 ↵
[1]:	↵

D 显示一段地址单元中的数据

格式：D[[段址:]起始地址, [尾地址]] ↵

D 命令执行后屏幕上显示一段地址单元中的数据, 在显示过程中, 可用 Ctrl+S 来暂停显示, 用任意键继续; 也可用 Ctrl+C 终止数据显示, 返回监控状态。

E 编辑指定地址单元中的数据

格式：E[[段址:]偏移量] ↵

该命令执行后, 则按字节显示或修改数据, 可通过“空格”键进入下一高地址单元数据的修改, 使用“-”键则进入下一低地址单元进行数据的修改, 并可填入新的数据来修改地址单元的内容。若输入回车, 则结束 E 命令。例:

表 4-2-3 E 命令示例

显示内容	键入内容
>	E3500 ↵
0000:3500 00_	05 空格
0000:3501 01_	空格
0000:3502 02_	—
0000:3501 01_	↵

G 运行程序

格式：G=[段址:]偏移量 ↵

GB=[段址:]偏移量 ↵

其中 G 格式表示无断点连续运行程序, GB 格式表示带断点连续运行程序, 连续运行过程中, 当遇到断点或按下 Ctrl+C 键时, 终止程序运行。

I 从 I/O 端口读入数据并显示

格式：I[I/O 接口地址] ↵

如：>I 0042 显示地址为 0042 接口单元的内容。

M 数据块搬移

格式：M[段地址:]源起始地址, 尾地址[目标地址:]目标起始地址 ↵

O 数据送存指定从 I/O 接口地址单元

格式：O I/O 接口地址 数据 ↵

如：>O 0600 80 就完成送 80 到地址为 0600 的 I/O 端口去。

R 寄存器或片内 RAM 区显示与修改

格式：R ↵ 或 R 寄存器名 ↵

R ↵ 操作后, 屏幕显示：CS=XXXX DS=XXXX IP=XXXX AX=XXXX F=XXXX

若需要显示并修改特定寄存器内容, 则选择 R 寄存器名 ↵ 操作。如 RAX ↵, 则显示：AX=

XXXX，键入回车键，结束该命令。若输入四位十六进制数并回车，则将该数填入寄存器 AX 中，并结束该命令。

T 单步运行指定的程序

格式：T=[段址:]偏移量]↵

每次按照指定的地址或 IP/PC 指示的地址，单步执行一条指令后则显示运行后的 CPU 寄存器情况。

U 反汇编

格式：U[[段址:]起始地址[, 尾地址]]↵或者 U↵

系统提供反汇编程序能力，上面第一格式可实现连续显示从某地址到另一高端地址间的代码反汇编，而后一种格式每次只能显示当前行。

附录 2 系统实验程序清单

80X86 实模式及其程序设计实验程序清单	
文件名	实验项目
Wmd861.ASM	2.1 系统认识实验
A2-1.ASM A2-2.ASM A2-3.ASM A2-4.ASM A2-5.ASM	2.2 数制转换实验 1. 将 ASCII 码表示的十进制转换为二进制数 2. 将十进制数的 ASCII 码转换为 BCD 码 3. 将十六位二进制转换为 ASCII 码表示的十进制数 4. 十六进制数转换为 ASCII 码 5. BCD 码转换为二进制数
A3-1.ASM A3-2.ASM A3-3.ASM	2.3 运算类编程实验 1. 二进制双精度加法运算 2. 十进制的 BCD 码减法运算 3. 乘法运算
A4-1.ASM	2.4 分支程序设计实验
A5-1.ASM A5-2.ASM	2.5 循环程序设计实验 1. 计算 $S=1+2\times 3+3\times 4+\cdots+N(N+1)$ 2. 求某数据区内负数的个数
A6-1.ASM A6-2.ASM	2.6 排序程序设计实验 1. 气泡排序法 2. 学生成绩名次表
A7-1.ASM A7-2.ASM	2.7 子程序设计实验 1. 求无符号字节序列中的最大值和最小值 2. 求 $N!$
A8-1.ASM	2.8 查表程序设计实验
A9-1.ASM A9-2.ASM CDISPLAY.C	2.9 输入输出程序设计实验 1. 显示程序实验 2. INT 21H 功能调用示例程序 3. C 语言显示输出实验
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
138.ASM	4.1 总线地址译码及时序观测与分析实验
MEM.ASM	5.1 静态存储器扩展及时序观测与分析实验

A82591.ASM	6.1. 8259 单一中断源控制实验
A82592.ASM	6.2. 8259 优先级中断管理实验
A82593.ASM	6.3. 8259 级连中断管理及时序观测与分析实验
A82371.ASM	7.1 存储器到存储器 DMA 传送及时序观测与分析实验
A82372.ASM	7.2 存储器到 I/O 设备 DMA 传送及时序观测与分析实验
A82373.ASM	7.2 I/O 设备到存储器 DMA 传送及时序观测与分析实验
IO-8.ASM	8.1 基本 I/O 输入输出及时序观测与分析实验
IO-16-1.ASM	1. 8 位 I/O 操作实验
IO-16-2.ASM	2. 16 位 I/O 操作实验
	3. 16 位 I/O 流水灯实验
A82551.ASM	8.2 8255 并行接口实验
A82552.ASM	1. 基本输入输出实验
A82553.ASM	2. 流水灯显示实验
	3. 8255 方式一输入输出实验
A82541.ASM	8.3 8254 定时/计数器应用实验
A82542.ASM	1. 计数应用实验
	2. 定时应用实验
A82511.ASM	8.4 8251 串行接口应用实验
A82512.ASM	1. 数据信号的串行传输
	2. 自收自发实验
AD.ASM	8.5 A/D 转换实验
DA1.ASM	8.6 D/A 转换实验
DA2.ASM	1. 锯齿波实验
	2. 方波实验
80X86 保护模式微机原理及其程序设计实验程序清单	
文件名	实验项目
PM.ASM	1.8 保护模式和实模式切换实验
P1-1.ASM	2.1 全局描述符及全局描述表实验
P1-2.ASM	2.1 局部描述符及局部描述表实验
P2-1.ASM	2.2 任务内无特权级变换的转移实验
P2-2.ASM	2.2 任务内无特权级变换的转移实验
P2-3.ASM	2.2 任务内有特权级变换的转移实验
P3-1.ASM	2.3 任务间无特权级切换实验
P3-2.ASM	2.3 任务间有特权级切换实验
P4-1.ASM	2.4 中断门、陷阱门实现中断/异常实验
P4-2.ASM	2.4 任务门实现中断/异常实验
PMEM1.ASM	4.1 无分页机制的存储器扩展实验
PMEM2.ASM	4.2 具有分页机制的存储器扩展实验

微机接口应用技术及其程序设计实验程序清单

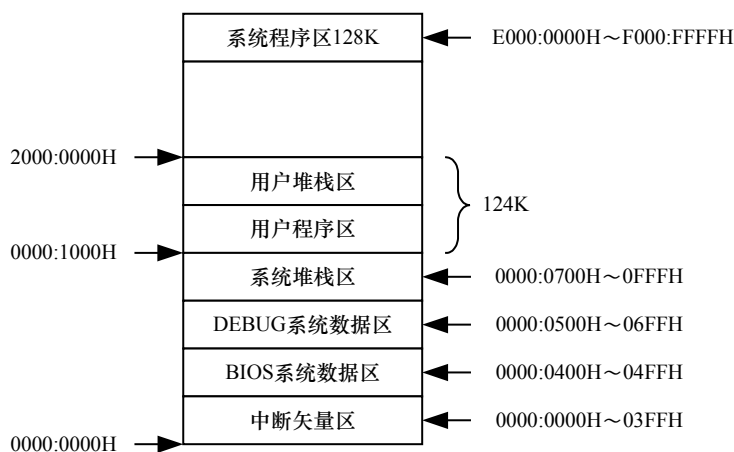
CKeyScan.C	2. 键盘扫描及显示设计实验
CSOUND.C	3. 电子发声设计实验
CLED16.C CHZDOT.H	4. 点阵 LED 显示设计实验
C82513.C C82514.C	5. 双机通讯实验接收程序 双机通讯实验发送程序
CBUJIN.C	6. 步进电机实验
CWENDU.C	7. 温度闭环控制实验
CZHILIU.C	8. 直流电机闭环调速实验

附录 3 系统编程信息

附 3.1 地址分配情况

1. 系统内存分配

系统内存分配情况如附图 3-1 所示。系统内存分为程序存储器和数据存储器，程序存储器为一片 128KB 的 FLASH ROM，数据存储器为一片 128KB 的 SRAM。



附图 3-1 系统内存分配

2. 系统编址

采用内存与 IO 独立编址形式，内存地址空间和外设地址空间是相对独立的。内存地址是连续的 1M 字节，从 00000H~FFFFFFH。外设的地址范围从 0000H~FFFFH，总共 64K 字节。

(1) 存储器编制

存储器编址情况见下表。

附表 3-1 存储器编址

	信号线	编址空间
系统程序存储器		E0000H~FFFFFFH
系统数据存储器		00000H~1FFFFFFH
扩展存储器	MY0	80000H~9FFFFFFH
	MY1	A0000H~BFFFFFFH

即 SRAM 空间: 00000H~1FFFFH 共 128K

其中: 00000H~00FFFH 为 4K 系统区

01000H~1FFFFH 为 124K 用户使用区

FALSH 空间: 0E0000H~0FFFFFFH 共 128K

其中: 0E0000H~0EFFFFH 为 64K 供用户使用区

0F0000H~0FFFFFFH 为 64K 系统监控区

(2) 输入/输出接口编址

输入/输出接口编址见下表。

附表 3-2 输入/输出接口编址

	信号线	编址空间
主片 8259		20H、21H
扩展 I/O 接口	IOY0	0600H~063FH
	IOY1	0640H~067FH
	IOY2	0680H~06BFH
	IOY3	06C0H~06FFH

附 3.2 常用 BIOS 及 DOS 功能调用说明

附表 3-3 INT 03H 使用说明

入口：无
功能：程序终止

附表 3-4 INT 10H 使用说明

入口：AH=01H, AL=数据
功能：写 AL 中的数据到屏上
入口：AH=06H, DS: BX=字串首址, 且字串尾用 00H 填充
功能：显示一字串, 直到遇到 00H 为止

附表 3-5 INT 16H 使用说明

入口：AH=00H
功能：读键盘缓冲到 AL 中, 读指针移动, ZF=1 无键值, ZF=0 有键值
入口：AH=01H
功能：检测键盘缓冲, 并送到 AL 中, 读指针不动, ZF=1 无键值, ZF=0 有键值

附表 3-6 INT 21H 使用说明

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