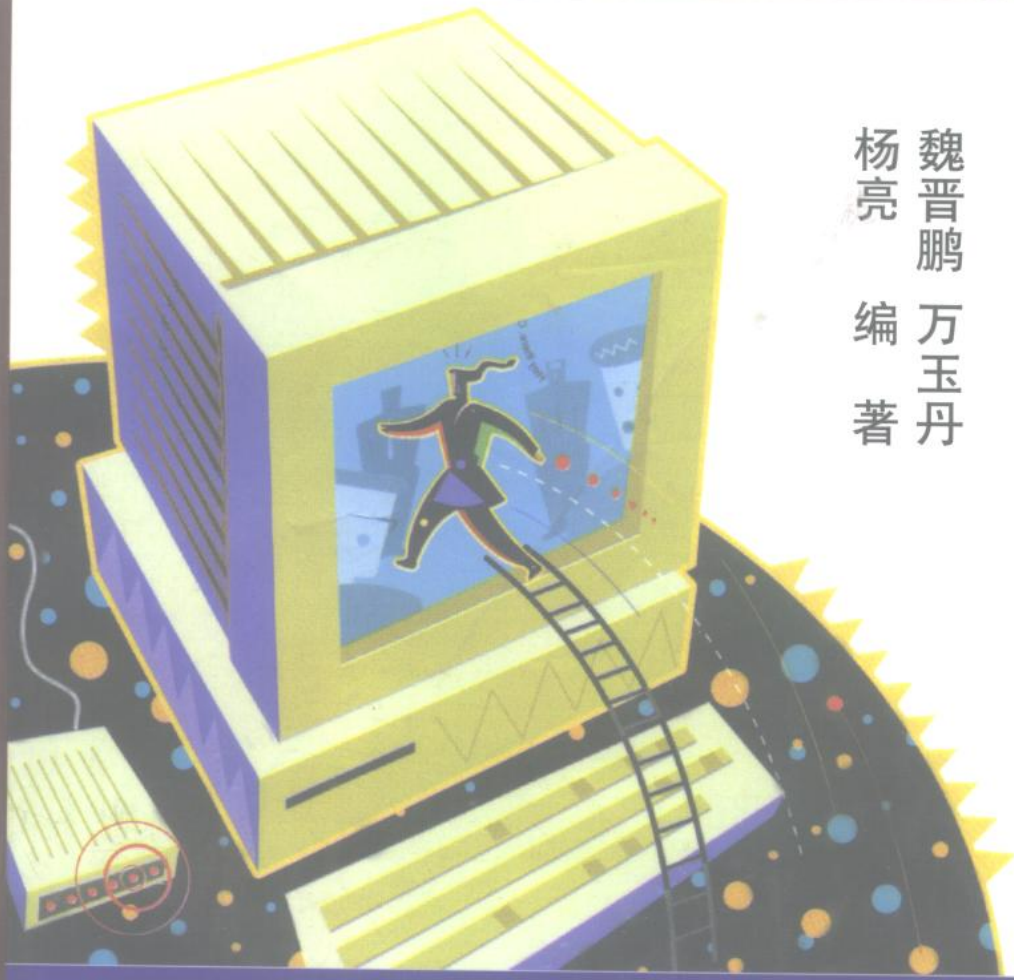
北京科海培训中心

Windows 深入剖析

初始化篇



魏晋鹏 万玉丹
杨亮 编著

清华大学出版社

北京科海培训中心

Windows 深入剖析——初始化篇

魏晋鹏　杨亮　万玉丹　编著

清华大学出版社

(京)新登字 158 号

内 容 提 要

本书适用于对 Windows 有一定基础，并希望通过了解 Windows 内核从而提高 Windows 的应用水平的读者。

全书详细分析了 Windows 的实模式初始化和虚拟机管理器的初始化过程，分析了虚拟设备驱动程序以及 Windows 的中断服务机制。本书不但对已有资料中简略提及的内容进行了详细的分析和解释，而且公布了许多新发现的数据结构。

本书的一大特色是对源代码详尽的注释，适合于计算机系统开发人员和高等院校师生阅读参考。

书　名：Windows 深入剖析——初始化篇

作　者：魏晋鹏　杨亮　万玉丹

出版者：清华大学出版社(北京清华大学校内，邮编 100084)

印刷者：北京市朝阳区科普印刷厂

发　行：新华书店总店北京科技发行所

开　本：16　印张：30.125　字数：732 千字

版　次：1997 年 8 月第 1 版　　1997 年 8 月第 1 次印刷

印　数：0001～5000

书　号：ISBN 7-302-02687-4/TP・1391

定　价：52.00 元

绪 论

勿庸置疑，Microsoft 的 Windows 系列（包括 Windows 3. x 和高档的 Windows NT）是 PC 机上真正的主流操作系统，从现在直到将来，Windows 都将是我们的主要工作和开发平台。

Windows 编程的特点就在于对编程的支持非常丰富；在 DOS 编程的时代，从计算到界面，每次都需要从头设计，相比之下，Windows 程序员们的工作就显得轻松多了。

但是，正如在 DOS 下仅满足于 INT 21 调用是难以编写出精湛的程序来的一样，在 Windows 下也需要对其内部原理有深刻的认识，比如下面几个方面就是如此：

- 设计 Windows 下的设备驱动程序（有时候这并非仅仅是厂商的事）；
- 如同 DOS 下的 TSR 那样，有时候，利用未公开的 Windows 调用可以完成一些常规方法难以做到的工作（如实时多任务等）；
- 同样是编程，如果能了解 Windows 程序执行的"微观机制"，将使得程序员心中有数，从而能在很大程度上确保代码质量；
- Windows 程序的调试不是一件容易的事，对于 Windows 内部机制的深刻理解将有助于程序员定位错误。

还可以举出很多像这样的例子。实际上，只要我们长期植根于一个操作系统，几乎可以肯定会产生深入了解其原理的需求；在 UNIX 和 DOS 的时代我们就已经看到了这一规律，在今天 Windows 的时代也不例外。

由于这种需要，也由于操作系统的研究与国产化的心愿，我们以 Windows 3. 1 为蓝本进行了深入的、然而也是异常艰辛的剖析，本书及其姊妹篇（Windows 深入剖析——内核篇）可以说是我们长期工作的总结。

本书的研究对象是 Windows 3. 1 的系统文件 WIN386. EXE。在整个 Windows 系统中，这个文件构筑了操作系统层次，提供了一个采用页式内存分配的多任务操作系统内核。Windows 之所以具有多任务特性并且能够运行 DOS 下的应用程序，就是因为 WIN386. EXE为它创建了虚拟机器（Virtual Machine）的运行机制。本书从 WIN386. EXE 的实模式初始化开始讲起，一步一步地描述了这一机制的建立过程，然后介绍了虚拟机器运转当中的相关细节，例如虚拟机器上的内存管理、虚拟机器的调度、虚拟机器上资源（即设备）的管理，等等。可以说，本书所分析的 WIN386. EXE，其目标就是虚拟机器的实现，所以本书称作"初始化篇"。

虚拟化并不是 Windows 的发明，但 Windows 确实使用了许多虚拟技术，这从 Microsoft 文档中的用词就可以看出来，例如虚拟内存、虚拟设备、虚拟机器等。那么在 Windows 中这些是怎样具体实现的？弄清这一点，不论是对理解 Windows 操作系统，还是对 Windows 中系统级的编程（如编写虚拟设备驱动程序），都是十分有益的。

此外，Windows 作为一个优秀的系统软件，它实现中的许多技巧，都具有创新性，值得

软件工作者借鉴。对这些技巧，如实例数据(Instance)、回调(CallBack)、断点(Breakpoint)等，本书都有详细的解释和说明。另外，关于 Windows 对调试设备的支持，循着 Windows 的运行轨迹，本书也着意做了介绍。

当今硬件技术发展很快，Intel 公司从 80386 芯片起就引入了许多新的概念，例如保护虚地址方式、虚拟 8086 方式、分段机制、分页机制等，其中有不少是原来中、小型乃至大型计算机上才有的技术，现在则纷纷下移到小小的 80x86 系列芯片中。这一方面说明了技术的进步，另一方面也向广大计算机工作者提出了新的课题和研究方向。因为长期以来，人们已经习惯了单任务的 DOS 系统，对于保护模式及其相关技术则比较陌生。在这个方面，Windows 为我们提供了一个很好的例子。因此本书可作为一个保护模式操作系统的实验教材。

在形式上，本书尽量做到全面、详尽，对源代码的注解细致到每一条指令、每一个内存变量；注意内容之间的横向联系和相互参考，方便读者阅读；总结性的文字说明力求保证一定的高度，既从操作系统的一般原理上加以诠释，又不脱离具体程序而流于空谈。然而限于作者的水平和客观条件，加之 Windows 的深入研究确是一项艰巨的任务，所以本书肯定会有一些不尽如人意之处，希望本书能起到抛砖引玉的作用，并能对我国的系统软件开发起到积极的推动作用。

在本书的写作过程中，得到了很多老师、朋友的关心和帮助，在此表示最诚挚的感谢！尤其是武汉大学计算机科学系的张焕国教授，在本书的写作过程中，他给予了极大的支持，谨在此致以深切的谢意！

目 录

第1章 基础知识

1.1 80386以上机器的寄存器

1. 通用寄存器

一共有8个，依次命名为EAX、ECX、EDX、EBX、ESP、EBP、ESI、EDI，它们的低16位可以作为16位的寄存器独立访问，并且依次命名为AX、CX、DX、BX、SP、BP、SI、DI，这些就是86系列处理器的以前的成员的8个16位通用寄存器。

2. 段寄存器

共有6个16位的寻址内存段的段寄存器，分别定名为ES、CS、SS、DS、FS和GS。FS和GS是从80386开始新增加的段寄存器。

3. 指令指针寄存器EIP

其低16位作为寄存器IP，提供用于执行8086和80286代码的16位的指令指针。

4. 处理器的状态和控制标志寄存器EFLAGS

32位的EFLAGS寄存器包含有若干个状态标志和控制标志位，图1.1展示了EFLAGS寄存器的各个位字段。

31 24	23 17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0
00000000	000000	VM	RF	0	NT	IOPL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

图1.1 EFLAGS寄存器详图

算术状态标志：CF、PF、AF、ZF、SF和OF位是传统的，不再赘述。

TF 是捕获允许标志。

IF 是中断允许标志，若IF=1，接收外部中断；如果IF=0，禁止外部中断。

DF 是方向标志。

IOPL 为特权级字段，是个两位宽的字段。它支持后面将谈到的保护方式。IOPL字段指定了要求执行I/O指令的特权级。如果当前的特权级在数值上小于或等于IOPL，I/O指令可以执行，否则发生一个保护异常。

NT 是嵌套任务位，控制IRET指令的运行，如果NT=0，则用栈中保存的值恢复EFLAGS，CS和EIP执行常规的从中断返回的动作。如果NT=1，中断返回用一任务转换代替上述过程。

RF 重启动标志控制着调试故障是接受(RF=0)，或者是被忽略(RF=1)。在成功地完成每一条指令以后处理器会把RF清成0，而当接收到一个非调试故障的故障信号时，处理器把RF置成1。

VM 是虚拟8086方式位。如果该位置为1，处理器将在虚拟的8086方式下工作，

如果清零，处理器将工作在一般的保护模式下。

5. 处理器控制寄存器

处理器的控制寄存器为 CR0～CR3，CR1 为保留寄存器，供今后开发的处理器使用。CR0～CR3 的各个位字段如图 1.2 所示。

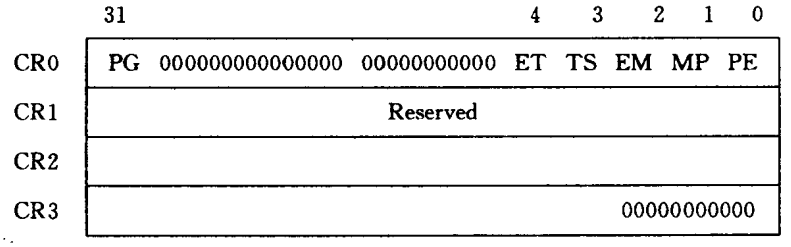


图 1.2 处理器控制寄存器

• CR0

ET，TS，MP，EM 是协处理器的控制位，用于控制 80387 浮点协处理器的操作，对本书不是很重要。

PE 这个位用于控制段机制，如 PE＝1，处理器工作于保护方式；PE＝0，处理器运行于实地址方式，且等同于 8086。

PG 此位用于控制分页机制，若 PG＝1，启用分页机制，进行线性-物理地址的转换，若 PG＝0，禁用分页机制，段机制产生的线性地址，直接地当作物理地址使用。

• CR2 及 CR3

CR2 及 CR3 由分页机制使用，CR3 中的内容是第一级页表即页目录表的起始物理地址，由于目录是页对齐的，所以寄存器中仅高 20 位有效，低 12 位保留未用。

CR2 用在发生页异常时报告出错信息。当发生页异常时，处理器把引起异常的线性地址保存于 CR2 中。操作系统中的页异常处理程序可以检查 CR2 的内容，从而查出线性地址空间中的哪一页引起本次异常。

6. 段表基地址寄存器

GDTR 是一个 48 位的寄存器，用来定义 GDT（全局描述符表）的基地址和界限，其中 32 位为基地址，16 位为界限。

IDTR 用来定义中断描述符表（IDT），也包含 32 位的基地址和 16 位的界限。

GDTR 和 IDTR 的结构如图 1.3 所示。

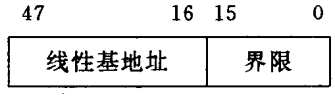


图 1.3 GDTR(IDTR)的结构

LDTR 这个 16 位的寄存器包含当前任务的 LDT 选择子。

TR 这个寄存器包含当前任务的任务状态段（TSS）的选择子，也是 16 位的。

1.2 80386以上机器的存储管理机制

在80386以上CPU中，提出了虚拟存储器(Virtual Memory)的概念，以便对内存进行分段管理。程序访问存储器时使用虚拟地址，虚拟地址再经过段和分页两种机制才能得到物理地址，从而达到了存储保护的目的。这个过程如图1.4所示。

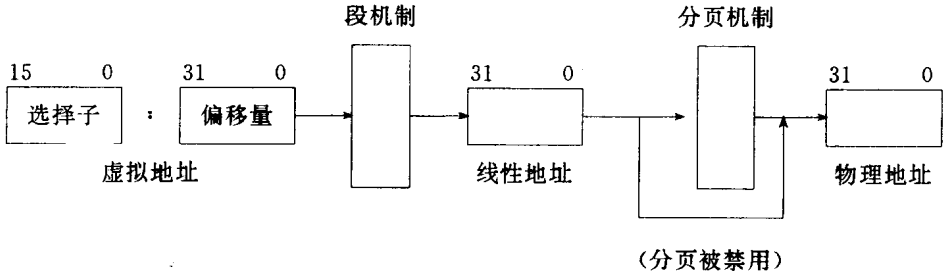


图1.4 虚-实地址转换

1. 段机制

段机制把虚拟存储器组织成大小可变的容量单位的集合，称之为段。每个段由它的基地址、界限及保护属性三个参数进行定义，它们存放在段的描述符中。

全局描述符表(GDT)及局部描述符表(LDT)是包含段描述符的两个特殊的段。每个段描述符占8个字节。

· 存储段描述符格式(如图1.5所示)。

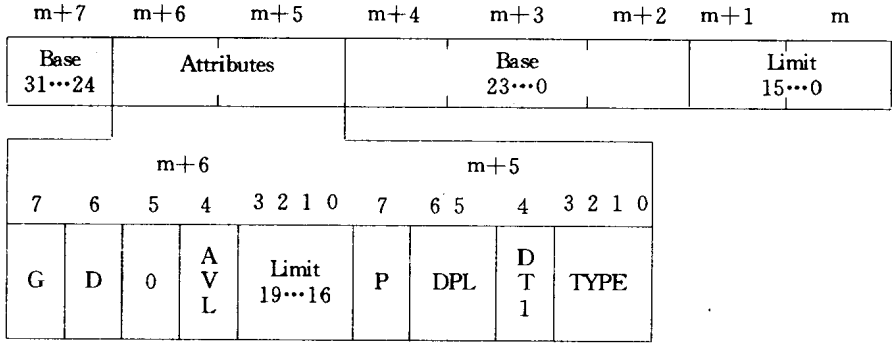


图1.5 存储段描述符格式

G　　G位是界限粒度属性位。G=0表示界限粒度为字节，G=1表示界限粒度为4K字节。应注意的是G位只对段界限的粒度起作用，而对段的基地址没有影响。即段基地址总以字节为粒度。

D　　① 可执行段使用D位，为段中的指令访问的地址或操作数设置默认值。D=1表示指令使用的默认值是32位地址及32位或8位操作数，这是80386以上机器的正常使用情况；D=0表示使用的默认值是16位的地址及16位或8位的操作数，与286系统兼容。使用指令前缀，可把指令的地址或

操作数的长度改变为与默认值相反的值。

② 向下扩展的段使用 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 286 兼容。

③ 由 SS 寄存器寻址的段，使用 D 位以确定隐式的堆栈访问指令（如 PUSH 及 POP 指令）要使用 32 位的 ESP 寄存器（此时 D=1），还是使用 16 位的 SP 寄存器（此时 D=0）。后者是考虑与 80286 兼容。

AVL　AVL 位是软件可利用位。

P　P=1 表示描述符对地址的转换是有效的，P=0 表示描述符无效，且使用该描述符会引起异常。

DPL　描述符特权级，共 2 位，用于定义与段相联系的特权级。

DT　描述符类型位。区分描述符所指的段是存储段（DT=1），还是系统段及门（DT=0）。

Type　类型字段。4 位长，定义存储段描述符的类型，对应代码见表 1.1。

表 1.1　存储段描述符类型

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	10	执行/读
3	读/写，已访问	11	执行/读，已访问
4	只读，向下扩展	12	只执行，一致码段
5	只读，向下扩展，已访问	13	只执行，一致码段，已访问
6	读/写，向下扩展	14	执行/读，一致码段
7	读/写，向下扩展，已访问	15	执行/读，一致码段，已访问

· 系统段描述符的格式（如图 1.6 所示）。

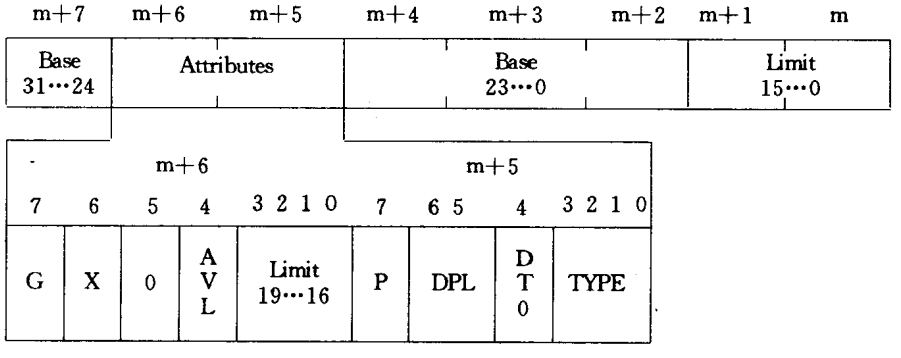


图 1.6　系统段描述符格式

· 门描述符的格式（如图 1.7 所示）。

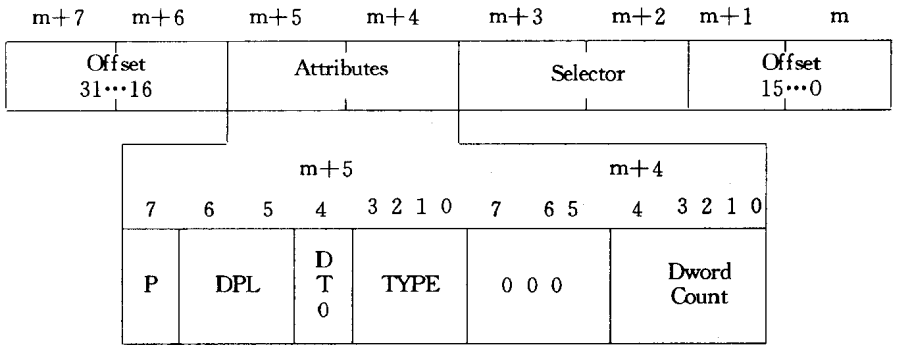


图 1.7 门描述符格式

在门描述符中，属性位存放在 m+4 及 m+5 字节。各属性位的详细说明如下：

P 存在位。P=1 表示门存在，即门有效；P=0 表示门不存在，即门无效。使用无效门将产生异常。

DPL 描述符特权级。定义门的特权级。

DT 描述符类型位。这里为 0，图中以 DT0 表示。

Type 门的类型。用四位编码定义，详见表 1.2。

DwordCount 双字计数。在通过调用门调用另一过程时，通常要传递一些参数到被调用的过程，这些参数一般放在堆栈内。如果由于门的使用引起特权级的转换及堆栈的改变，则需将一个堆栈的参数复制到另一个堆栈。DwordCount 字段给出的值，即是要复制的双字参数的数量。

表 1.2 系统段和门的类型字段的编码

类型编码	定义	类型编码	定义
0	未定义	8	未定义
1	可用 286TSS	9	可用 386TSS
2	LDT	10	未定义
3	忙的 286TSS	11	忙的 386TSS
4	286 调用门	12	386 调用门
5	任务门	13	未定义
6	286 中断门	14	386 中断门
7	286 陷阱门	15	386 陷阱门

• 段选择子

段选择子标识一个段，也可把段选择子视为段的名字，如图 1.8 所示。

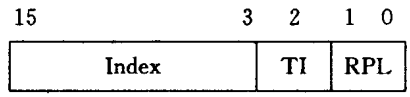


图 1.8 选择子格式

RPL 是请求特权级字段，在段保护模式中起着关键作用。

TI 此位指定包含段描述符的描述符表，TI=0 表示段描述符在 GDT 中，TI=1 则表示段描述符在 LDT 中。

Index 索引字段给出段描述符在 GDT 或 LDT 中的索引。

总之，一个选择子通过把段描述符定位到 GDT 或 LDT 中的某个位置而命名了一个段。

2. 分页机制

分页机制把线性地址转换为物理地址，分页机制由 CR0 中的 PG 位启用。

- 两级页表结构(如图 1.9 所示)。

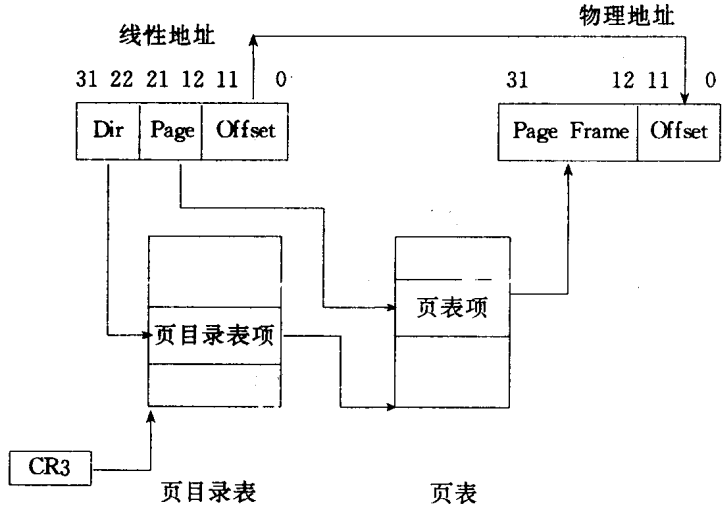


图 1.9 两级页表结构

- 页表项格式(如图 1.10 所示)。

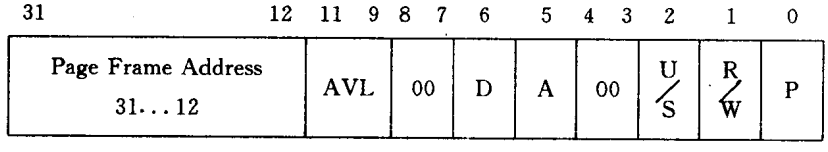


图 1.10 页目录表/页表的表项格式

P 位 0 为存在位，该位表示表项对地址转换有效(P=1)，或无效(P=0)。在页转换期间，无论在目录表或页表中遇到无效表项，都会产生异常。

R/W 位 1 为 R/W 位，即读/写位。如该位为 1，对页表指定的页可进行读，写或执行；如果该位为 0，该页可读或执行，但不能对该页进行写操作。然而读写位并不总是

起作用的。当处理器正在超级特权之一(即0、1或2级)执行时,R/W位被忽略。在目录表项中的R/W位,应用于该目录表项映射的所有各页。

U/S 位2为U/S位,即用户/系统位。如该位为1,页表指定的页可由在任何特权级(包括用户级即3级)下执行的程序访问;如该位为0,则该页只能由在超级特权级(即0,1或2级)执行的程序访问。在目录表项中的U/S位,应用于该目录表项映射的所有各页。

A 位5为A位,即访问位。页表项中的A位在对页表项映射的页进行任何访问之前,由处理器设置为1。目录表项中的A,在对目录表项映射的各页中的任何一页进行任何访问之前,由处理器设置为1。处理器永不清除A位,但操作系统软件为获取页使用情况的统计信息而周期地清除A位。

D 位6为D位,即已写标志位。第二级页表项中的D位,由处理器对页表项映射的页进行写访问之前将其设置为1。处理器不修改页目录中表项的D位。

AVL AVL字段在位9～位11,供软件使用。处理器不能对其进行修改。

1.3 处理器的三种工作方式

1. 实地址方式

这是8086上采用的传统的方式,地址形式为

段址:偏移

物理地址=段址*16+偏移。这种方式只能寻址1MB的地址空间。

2. 虚拟8086方式(V86)

在这种方式下,CR0的PG=1,PE=0,即不支持分段,但支持分页。地址形式仍为

段址:偏移

但是用"段址*16+偏移"算得的只是线性地址,还需经过分页机制才能得到物理地址。

3. 保护方式

CR0的PG=1,PE=1,段机制和分页机制都启用,地址形式为

选择子:偏移

虚拟地址到物理地址的转换过程如图1.4所示。

1.4 中断及异常机制

中断和异常是特殊的控制转移方法,这种控制转移在正常编程的指令流之外完成。每一种中断及异常,都有一个与之相联系的中断向量,用8位二进制数字表示。向量号用来从中断描述符表(IDT)中为给定的中断及异常选择处理程序。

分配给异常的向量是前32个向量编号,即0～31. 中断的向量号可以分配给0～255范围内的任何值,但为了避免与异常向量号发生冲突,最好分配在32～255范围内。

在发生中断及异常时要使用中断描述符表(IDT),该表由IDTR寄存器指向。向量号作

为 IDT 表的索引，从 IDT 表中获取一个 8 字节的门描述符。

下面讨论作为中断描述符的门描述符，图 1.11 显示了门描述符的格式：

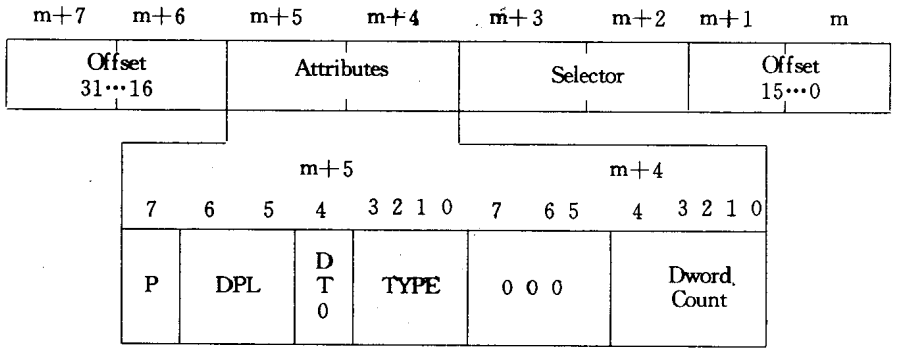


图 1.11 门描述符格式

在门描述符中，属性位存放在 m+4 及 m+5 字节。各属性位的详细说明如下：

P 存在位。P=1 表示门存在，即门有效；P=0 表示门不存在，即门无效。使用无效门将产生异常。

DPL 描述符特权级。定义与门相联系的特权级。门的 DPL 只在 INTn 及 INT0 时进行检查，以避免应用程序执行 INTn 时，使用了各种设备在中断表中使用的向量号。对于所有其他的异常或中断，忽略门 DPL。

DT 描述符类型位。这里为 0，图中以 DT0 表示。

Type 门的类型。用四位编码定义，详见表 1.3；它与表 1.2 完全相同，为了阅读方便，该表也在此处列出。IDT 中的描述符必须具有下列类型之一：

- 任务门
- 286 中断门
- 286 陷阱门
- 386 中断门
- 386 陷阱门

DwordCount 该字段在此处的门中没有使用。

表 1.3 系统段和门的类型字段的编码

类型编码	定义	类型编码	定义
0	未定义	8	未定义
1	可用 286TSS	9	可用 386TSS
2	LDT	10	未定义
3	忙的 286TSS	11	忙的 386TSS
4	286 调用门	12	386 调用门
5	任务门	13	未定义
6	286 中断门	14	386 中断门
7	286 陷阱门	15	386 陷阱门

在保护模式下,各中断向量的含义和作用也发生了根本的变化,比如我们熟悉的 DOS 下的屏幕打印中断(INT 5),在保护模式下用于"边界检查"。详见表 1.4 所示。

表 1.4 异常一览表

向量号	异常名	类别	出错码	产生异常的指令
0	除法出错	故障	/	DIV,IDIV
1	排错	故障/陷阱	/	任何指令
3	单字节 INT3	陷阱	/	INT3
4	溢出	陷阱	/	INTO
5	边界检查	故障	/	BOUND
6	非法操作码	故障	/	非法指令编码或操作数
7	设备不可用	故障	/	浮点指令或 WAIT
8	双重故障	终止	有	任何指令
9	协处理器段越界	终止	/	访问存储器的浮点指令
10	无效 TSS	故障	有	JMP,CALL,IRET,中断
11	段不存在	故障	有	装入段寄存器的任何指令
12	栈段异常	故障	有	·装入 SS 寄存器的任何指令 ·对以 SS 寄存器寻址的段中进行存储访问的任何指令
13	通用保护	故障	有	·任何特权指令 ·任何访问存储器的指令
14	页异常	故障	有	任何访问存储器的指令
15	协处理器出错	故障	/	浮点指令或 WAIT
0～255	软中断	陷阱	/	INT n

1.5 任务状态段

任务状态段 TSS(Task State Segment)是包含关于任务的重要信息的特殊段。活动任务的 TSS 通过 TR 寄存器来寻址。不活动任务的 TSS 中包含着暂时固定的任务的各种信息,通过在 TSS 中保存任务的寄存器状态的完整映象,TSS 可以保证任务的挂起与恢复。当任务挂起时,CPU 各寄存器的值被保存到 TSS 中;当任务重新激活时,从 TSS 中装入各寄存器的值,CPU 从原来被打断的地方继续执行,就好象该任务从未被挂起过一样。任务状态段的格式见表 1.5。

表 1.5 任务状态段的格式

31	0	
I/O 许可位图的偏移	0000000000000000T	64h
0000000000000000	LDT	60h
0000000000000000	GS	5Ch
0000000000000000	FS	58h
0000000000000000	DS	54h
0000000000000000	SS	50h
0000000000000000	CS	4Ch
0000000000000000	ES	48h
EDI		44h
ESI		40h
EBP		3Ch
ESP		38h
EBX		34h
EDX		30h
ECX		2Ch
EAX		28h
EFLAGS		24h
EIP		20h
CR3		1Ch
0000000000000000	SS2	18h
ESP2		14h
0000000000000000	SS1	10h
ESP1		0Ch
0000000000000000	SS0	08h
ESP0		04h
0000000000000000	LINK	00h

1. 链接字段 LINK，指向下一个任务状态段，其低 16 位为选择子，高 16 位为 0000。

2. 内层栈指针

TSS 中有三个堆栈指针 SS0:ESP0、SS1:ESP1、SS2:ESP2，分别指向 0 级、1 级及 2 级堆栈的栈顶。当发生向内层特权级的转移时，就将对应的堆栈指针取出来装入到 SS 及 ESP 寄存器，从而避免由于共享栈而引起的保护问题。在本书中，最重要的是 SS0:ESP0，在当前任务执行过程中发生中断时，如果要陷入特权级 0 的 VMM，则通过使用 SS0:ESP0 进行堆