虚拟设备驱动程序开发起步与进阶

彭礼孝　编著　　雨人　策划

人民邮电出版社
www.pptph.com.cn

内 容 提 要

本书讲述如何开发虚拟设备驱动程序（VXD），共分为三部分。第一部分主要讲述 VXD 开发所需的基础知识。第二部分讲述如何使用软件 VTOOLSD 进行 VXD 的开发，包括对程序的调试方法和 VTOOLSD 类库的介绍，同时讲述一些简单实例的开发。第三部分主要讲述高级实例开发。

本书适合于熟悉 Windows 9x 操作系统，希望掌握开发 VXD 技术的读者学习使用。

虚拟设备驱动程序开发起步与进阶

◆ 编　　著　彭礼孝
策　　划　雨　人
责任编辑　李振广

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ pptph.com.cn
网址　http://www.pptph.com.cn
读者热线:010－67129212　010－67129211(传真)
北京汉魂图文设计有限公司制作
北京朝阳隆昌印刷厂印刷
新华书店总店北京发行所经销

◆ 开本:787×1092　1/16
印张:18
字数:446 千字　　2000 年 12 月第 1 版
印数:5 001－8 000 册　　2001 年 4 月北京第 2 次印刷
ISBN 7-115-09006-8/TP·1983

定价:27.00 元

前　言

CIH 病毒曾使很多人记忆深刻，就其本质来说，它利用了虚拟设备驱动程序（VXD）的技术。本书的目的就是带领读者进入虚拟驱动程序的世界，让读者一睹其奥妙和风采。

现在市面上关于 Windows 编程的书多如牛毛，但是关于设备驱动程序开发的书却少的可怜。驱动程序通常工作在与操作系统同级的核心级，因而具有同操作系统同级的权限，可以实现许多一般应用程序无法完成的功能，比如实现对整个 4GB 虚拟地址空间的访问。

本书的读者

本书假定读者熟悉 Windows9x 操作系统和具备一定的 C/C++开发知识，还要具备一定的计算机结构知识和汇编知识。如果读者希望能在最短的时间里学会控制物理硬件或者扩展操作系统的功能，那么本书将非常适合您学习使用。如果读者已经会开发驱动程序但还不是很精通，那么本书将有很好的参考价值。

工作平台

本书所讲述的所有内容均是基于运行在 Intel CPU 上的 Windows 9x 操作系统，所有程序均在 Windows 98 操作系统上调试通过。

本书的内容

本书分为三大部分，为了便于学习和掌握，各部分都循序渐进展开讲解。

第一部分主要讲述基础知识。介始学习开发虚拟驱动程序必须了解和掌握的 Windows 9x 的内核和结构以及虚拟设备驱动程序的基础知识，同时对 Intel CPU 的编程结构也进行了介绍，如果只想快速直接地开发 VXD，可以先阅读第二部分。

第二部分主要讲述如何使用软件 VTOOLSD 进行虚拟设备驱动程序的开发，包括对程序的调试方法和 VTOOLSD 类库的介绍。读者学习这部分内容，就可以进行一些简单的开发。

第三部分主要讲述实例开发，本部分将引导读者进行综合开发，详细具体的讲述一些实例和高级应用。

开发工具

本书使用软件 VTOOLSD 进行开发，该软件是由 VIREO 公司开发的，是用 C 和 C++进行开发虚拟驱动程序的强大工具，程序的调试采用由 NUMEGA 公司开发的软件 SOFTICE 和可以显示调试输出的 DBGVIEW 程序。虽然 SOFTICE 也显示调试输出，但是不如 DBGVIEW 使用方便。VTOOLSD 和 SOFTICE 均可以在中国超级工具库 (http://202.102.251.27/home/pctools/index.htm) 下载，而 DBGIEW 程序可以在 http://www.sysinternals.com 下载。

编著者

目　录

第1章

Intel CPU 编程结构

386 的体系结构
386 的寄存器结构

本章主要讲述 Intel CPU 设备驱动程序的编程结构，它包括体系结构和寄存器结构。这部分是进行系统核心开发所必须掌握的知识，通过对体系结构的学习，大家应该了解 CPU 的工作原理，而对寄存器结构的学习，将在用汇编语言进行系统开发时得到充分利用。

现在流行的微处理器都还是 32 位或准 64 位。如果说微处理器从 8 位到 16 位主要是总线的加宽，那么，32 位微处理器和 16 位微处理器相比，则是从体系结构设计上有了概念性的改变和革新。比如，32 位微处理器普遍采用了流水线和指令重叠执行技术，虚拟存储技术，片内存储管理技术以及存储器管理分段分页保护技术等。这些技术为在 32 位微机环境下实现多任务操作系统提供了有力的硬件支持。下面将在结合第一个 32 位微处理器 80386 介绍 32 位微处理器时，讲解这些重要的技术。

也许大家觉得现在都是 Pentium II 时代了，还在讲 80386，未免有些不适宜。其实不然。从 80386 开始，Intel 公司不断对其微处理器增加主频提高运行速度以及对其功能进行增强和扩展，但其微处理器在本质上，无论是从体系结构看还是从内部的寄存器结构看都具有很大的延续性和继承性。而且 80386 在 32 位微处理器系列中又是最简单的，因此学好 80386 是基础，也是关键。

1.1 386 的体系结构

80386 芯片内部可以被分为六个独立的处理部件：总线接口部件（BIU）、执行部件、分段部件、分页部件、指令预取部件和指令译码部件。而分段部件和分页部件统称为存储管理部件（MMU）。这六个部件可以并行工作，构成一个六级流水线体系结构，从而大大提高 80386 CPU 的工作效率。

80386DX 和 80386SX 在结构上的根本区别是它们的外部数据总线和地址总线尺寸不同；在内部，两者都采用 32 位数据总线。而 80386DX 的地址总线是 32 位宽，这使它可以访问 4GB（2^{32}）的地址空间；80386SX 的地址总线只有 24 位宽，只可以访问 16MB（2^{24}）的地址空间。我们将只讨论 80386DX。

1.1.1 总线接口部件

总线接口部件（BIU）是 CPU 与外部设备的连接。毫无例外，片上其他的处理部件访问总线的请求都传送给总线接口部件。由于各处理部件的并行性，有可能出现同一时刻总线接口部件同时收到多个访问总线的请求。除了作为总线接口外，总线接口部件还必须对这些请求进行优先排序。为了避免执行部件的延迟，执行部件进行数据传输的请求具有比指令预取部件进行预取指令的请求更高的优先级。由于总线接口部件只能处理物理（硬件）地址，所以，操作数的地址必须首先经分段部件处理，若需要，还需经分页部件处理。

1.1.2 指令预取部件

预取部件的工作相当简单。指令译码部件从一个 16 字节指令缓冲队列（对 80386 而言）中提取指令，然后指令预取部件试图填充指令缓冲队列。指令预取部件继续询问总线接

口部件以取出下一条指令。当预取部件接收到数据时，它把其放在队列中（如果队列未满），并且请求另一个 32 位存储器片。总线接口部件处理预取部件请求的重要性不如其他的部件。采用这种方法，请求操作数的当前执行指令接受最高优先级从而丝毫没有降低执行速度，但是预取部件仍然尽可能频繁地发生。预取部件在执行部件处理一个 CALL（调用）、一个 JMP（跳转）或一个中断时获得通知，从而它可以从新地址处取出指令。当一个 CALL、JMP 或中断时清空指令缓冲队列，通过预取部件重新取指令填充指令缓冲队列。

1.1.3 指令译码部件

指令译码部件的工作类似于预取部件。它从预取队列中提取单个字节组并且决定完成下条指令所需的字节数。在从预取队列中拿走整个指令以后，指令译码部件重新把操作码规定成一个内部指令格式，并且把译码的指令放入指令队列中。如果刚译码的指令将产生一个存储器引用，那么指令译码部件也向总线接口部件发出请求。这允许在执行指令前获取指令的操作数。

1.1.4 执行部件

执行部件是 CPU 完成运算的部件，能完成移位、加、除等指令所必须的操作。执行部件取出由译码部件译码的指令并执行它们。在执行一条指令时，执行部件有可能与其他部件进行通讯。80386 执行部件的功能可分为三个部件：控制部件，数据部件和保护测试部件。控制部件的功能是加速某种类型的操作，包括乘法、除法和有效地址的计算。加速是由微码和并行硬件结合起来完成的。实际上，控制部件包含了微码的设计，以便更快捷更有效地执行指令。数据部件包含了算术和逻辑部件及 80386 的 8 个 32 位通用寄存器。从 8086 开始，执行部件就与 ALU（算术逻辑单元）和通用寄存器联系在一起。保护测试部件监视存储器访问是否违反了保护规则。任何对存储器的访问，包括运算，在保护模式下都是严格监控的。这一部件用微码实现了保护违反检查。

1.1.5 分段部件

分段部件是执行地址转换的第一步。它负责把逻辑地址转换成线性地址。每个段寄存器（CS、DS、ES、SS、FS 和 GS）都有段描述符高速缓冲器加速地址转换。一旦转换，线性地址就传送给分页部件。

1.1.6 分页部件

分页部件接收从分段部件传送来的线性地址并将其转换成物理地址。如果分页机制不使能，那么线性地址和物理地址是相同的，并不需要再进行地址转换。如果分页机制是使能的，那么每次访存还需通过分页部件进行第二次的地址转换，即从线性地址到物理地址的转换。转换一旦完成，物理地址就传送给总线接口部件。

在 Intel 80386 中，为了加快 CPU 的速度，采用指令重叠执行技术。具体说就是将一条访存的指令和前一条指令的执行重叠起来，使两条指令并行执行，指令预取队列和指令译码队列为这种功能的实现提供了前提。

80386 允许使用虚拟存储器。所谓虚拟存储器就是系统中有一个速度较快的和容量较小的内存，还有一个速度较慢的外存，通过存储管理机制，使后者和前者有机地结合起来。如此，从程序员的角度来看，系统中似乎有一个容量巨大的，速度也比较快的主存，但是它并不是真正的物理上的存储器，故称虚拟存储器。80386 的虚拟存储器的容量可高达 64TB（64 兆兆字节），这样就可以运行要求存储器容量比实际主存容量大得多的程序。

在 Intel 80386 系统中，存储体按段划分，每个段的容量可变，最大可达 4GB。分段的作用是可以对容量可变的程序模块和数据模块提供模块性和保护性。80386 在运行时，可以同时运行多个任务。对每个任务来说，可以拥有多达 16384（2^{14}）个段。每个段又划分为多个页面，一个页面为 4KB 大小。分页的作用是便于实现虚拟存储管理，通常在内存和磁盘进行映像时，都以页为单位把 CPU 的地址空间映像到磁盘。

1.2 386 的寄存器结构

除了实现执行指令的逻辑以外，80386 和 80486 还拥有芯片上的存储单元，称之为寄存器。由于寄存器在 CPU 的内部，所以访问寄存器要比访问内存快得多。通用寄存器用于存储经常访问的数据，其他的寄存器含有控制处理器某个具体操作的信息。

80386 内部共有 30 多个寄存器。他们可以分为五类：通用寄存器（应用程序使用它来进行数据的存取和计算），段寄存器（将存储器分段寻址），系统地址寄存器（用来帮助操作系统运行在保护模式下），控制寄存器（控制处理器的行为）以及调试和测试寄存器（用于调试和测试）。

1.2.1 通用寄存器

80386 有 8 个 32 位的通用寄存器，如图 1.1 所示。它们都是 8086 中 16 位通用寄存器的扩展，故命名为 EAX、EBX、ECX、EDX、EBP、ESI、EDI 和 ESP，用来存放数据或地址。数据可以为 1 位、8 位、16 位、32 位或 64 位，地址则可为 16 位或 32 位。在 80386 及其后续系列中，用户可以寻址这些寄存器的各个部分。访问寄存器的尺寸可为 8 位、16 位和 32 位，并且不同的尺寸赋之不同的名称。比如，EAX 是一个 32 位寄存器的名称，而低 16 位寄存器则按 AX 访问，AX 的一半可以按 AL（低 8 位）或 AH（高 8 位）访问。

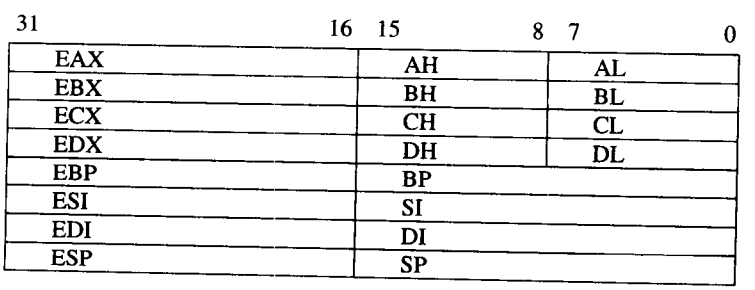

图 1.1 386 通用寄存器

31 16	15 0
EFLAGS	FLAGS
EIP	IP

图 1.2 386 标志寄存器和 IP 寄存器

两个附加的寄存器保留了当前指令流的状态信息。如图 1.2 所示。EIP 寄存器包含执行指令的下一条指令的偏移地址。EFLAGS 寄存器含有与不同的指令相关的一组字段。与其他寄存器一样，EIP 和 EFLAGS 寄存器拥有 16 位组成——IP 和 FLAGS 寄存器。

EIP 或 IP 寄存器与 CS 段寄存器共同指出了下一条被执行指令的存储器的地址。

EFLAGS 寄存器的组成如下，其中 EFLAGS 寄存器的 18~31 位未被使用。

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

AC——调准检查：此位仅用于 80486。当 AC 置为 1 时，80486 希望调准存储器访问，从而引用一个操作数仅需要最少的访存次数。由于硬件的接口方法，一个 32 位操作数必须以存储器地址除 4 作为开始。当 AC 为 0 时，80486 则简单地进行访存，而不管性能上的因素。

VM——虚拟 86 模式：当 VM 位为 1 时，则表示此时 386 工作在虚拟 86 工作模式。虚拟 86 模式的含义将在下一章讲述。应用程序不能修改 VM 位，并且修改 EFLAGS 寄存器的指令保留 VM 位不变。仅仅在任务切换操作，中断或中断返回操作时可以修改 VM 位。

RF——重新启动标志：此位控制一条指令执行期间是否可以生成一个调试故障。当程序执行期间发生一个异常时，CPU 把当前 CS、IP、EFLAGS 寄存器压入堆栈中并把控制权交给异常处理程序。EFLAGS 寄存器的堆栈映像中 RF 位被设置为 1。当异常处理程序返回到被中断的指令时，RF 位为 1，从而它将禁止生成一个重新启动异常，而其他任何故障都正常发生。调试异常拥有异常中最高的优先级。当控制权返回到被中断的指令时，设置 RF 位并且在指令执行完成前不再发生调试异常。而在 CPU 处理完被中断的指令时清除 RF 位。

NT——任务嵌套标志：当一个调用（CALL）、中断、陷阱或异常引起任务切换时，设置该位为 1，从而指出当前任务嵌套于另一个任务中。此位设置于新任务的 EFLAGS 寄存器中，并且指明一个反向任务切换有效（执行 IRET 指令反向切换任务）。

IOPL——I/O 特权级：这个双位字段在 0~3 之间取值，它指明完成 I/O 指令所需的级别和可以控制中断允许和禁止所需的级别。Intel CPU 分为四个优先级别：0 级的优先级最高，1 级次之，2 级更次之，3 级的优先级最低（在下一章中将具体讲述）。在保护模式下，IN、INS、OUT、OUTS、POPF、CLI 和 STI 等指令被分类为 IOPL 敏感的指令；而在虚拟 86 模式下，STI、CLI、PUSHF、POPF、LOCK、INT 和 IRET 等指令被分类为 IOPL 敏感的指令。虽然 INT 指令是 IOPL 敏感指令，但 INT3（CCh）、INT0 和 BOUND 指令却不是（在保护模式下它们也不是 IOPL 敏感的），值得注意的是 IN、INS、OUT 和 OUTS 不是 IOPL 敏感的。取而代之的是在虚拟 86 模式下 I/O 操作忽略 IOPL 域，I/O 请求会立即

与 I/O 允许位图（采用若干个字节的每个二进制位来表示每个 I/O 端口的可访问性：0 表示允许访问；1 表示禁止访问）对照检查。所谓 IOPL 敏感的指令是指某一任务在执行这些指令时，由 EFLAGS 寄存器中的 IOPL 域来控制。为了使用 IOPL 敏感指令，任务运行的特权级必须至少等于（数字的等于或小于）IOPL 域指定的当前特权值。一个低特权级的任务使用这些指令将会引起一个一般保护异常（GPF）。每个任务在它的任务状态段（TSS）保持了一个它自己的 EFLAGS 寄存器的备份，因此可为各个任务单独设置 IOPL 域的值。尽管 IOPL 位于 EFLAGS 寄存器中，但是只有运行在 0 特权级（最高特权级）的过程，并且仅能使用 POPF 或 POPFD 指令来修改 IOPL。当一个想要改变 IOPL 域值的非法企图不会导致一个异常，但是该企图将被忽略，当前 IOPL 域的值不会改变。因此，一个通常运行在 3 特权级的应用任务将不能改变它自己的 IOPL 域的值。

OF——溢出标志：当执行一个整数运算指令时，如果结果过大或过小而不适合目的寄存器或存储器的地址时，设置 OF 为 1。由于 OF 的设置与整数指令有关，所以 CPU 假设目的寄存器比符号位允许的尺寸小一位。比如：MOV AL，127；AL=7FH，对符号整数 OF=0 ADD AL， 2； AL=81H （–127），对符号整数 OF=1。（注意：若想使用无符号运算，那么忽略 OF 位）。在上例中，寄存器 AL 中 127 加 2 得到有效的无符号结果 129。

DF——方向标志：方向标志位控制字符串指令的行为：MOVS，STOS，LODS，CMPS，SCAS，INS 和 OUTS。当 DF=0 时，字符串指令从低地址向高地址操作；当 DF=1 时，字符串指令从高地址向低地址操作。STD 指令置 DF 为 1，CLD 指令清除 DF。

IF——中断允许标志：当设置此位时，CPU 能响应外部硬件中断。当复位此位时，将禁止 CPU 响应外部硬件中断。但是，此位并不影响不可屏蔽中断（NMI）、故障（异常）和软件中断，CPU 将总是响应 NMI、异常和软件中断。STI 指令设置此位，CLI 指令清除此位。

TF——陷阱标志：陷阱标志帮助调试应用程序。当置 TF=1 时，下一条指令执行完后立刻触发中断 INT 1。

SF——符号标志：当执行算术或逻辑运算指令时符号标志位变化。符号标志位接收结果的高端位值，当 SF=1 时表示此时结果为负数。

ZF——零标志：当 ZF=1 时表示运算结果为零；ZF=0 时表示运算结果为非零。

AF——辅助进位标志：辅助进位标志表明在一个算术运算中发生的 AL 寄存器低半个字节进位现象。此位由 ASCII 和 BCD 指令使用。它实现了多个数字精度十进制运算。

PF——奇偶标志：当一个算术运算指令生成一个有偶数个 1 的值时，奇偶标志 PF 置 1。例如，MOV AH，01000111B；PF 不变

ADD AH，10001000B；AH=11001111B，PF 被置 1。

CF——进位标志：当一个算术运算操作的结果对应的目的寄存器过小时，设置进位标志。它与 OF 位操作相似，但指明了目的单元的一个无符号溢出。

1.2.2 段寄存器

与 8086 类似，80386 中存储器的地址也是由段基地址和段内偏移量构成的。为此，386 中内部设置了 6 个 16 位的段寄存器 CS、DS、ES、SS、FS 和 GS。如图 1.3 所示。CPU 在不同的工作模式下段寄存器给出的值的含义不同。

15	0
CS	
DS	
ES	
SS	
FS	
GS	

图 1.3　段寄存器

考虑到性能上的因素，每个段寄存器都配置了一个描述符高速缓存，它保存段存储器的起始地址、段的大小界限和访问权限。段寄存器的描述符高速缓存对于程序员来说是不能访问的；仅仅 16 位的段寄存器可以直接访问。图 1.4 说明了段寄存器和内部描述符高速缓存的关系。

15	0	基址	段界限	访问权限
可访问部分	不可访问部分			
CS				
DS				
ES				
SS				
FS				
GS				

图 1.4　段寄存器的实际构成

1.2.3 系统地址寄存器

系统地址寄存器用来管理在保护模式下使用的系统表。系统表有全局描述符表（GDT）、中断描述符表（IDT）、局部描述符表（LDT）和用于存储任务内容的任务状态段（TSS）。系统地址寄存器有 4 个：全局描述符表寄存器（GDTR），中断描述符表寄存器（IDTR），任务状态寄存器（TR），局部描述符表寄存器（LDTR）。

GDTR 是 48 位的寄存器，存放全局描述符表 GDT 的 32 位线性地址和 16 位的全局描述符表的界限值。

IDTR 是 48 位的寄存器，存放中断描述符表 IDT 的 32 位线性地址和 16 位的中断描述符表的界限值。

TR 是 16 位的寄存器，存放任务状态段 TSS 的 16 位选择字。

LDTR 是 16 位的寄存器，存放局部描述符表 LDT 的 16 位选择字。

GDTR 和 IDTR 必须在转入保护模式之前进行初始值设定，这两个寄存器在实地址模式下可以访问。但是，LDTR 和 TR 仅可以在保护模式下使用，程序仅可以访问的是段选择寄存器，其他的缓冲部分是在任务切换时由 LDT 描述符和 TSS 描述符中自动装入的。

从 CPU 的角度看，系统地址寄存器都还有不可访问的缓冲部分。如图 1.5 所示。

31	16	15	0
可访问部分		不可访问部分	
基地址		段界限	访问权限
GDTR			
IDTR			

15 0

可访问部分	不可访问部分		
选择寄存器	访问权限	基地址	段界限
LDTR			
TR			

图 1.5 系统地址寄存器构成

关于各种工作模式，描述符和任务状态段等概念将在下一章详细讲述。

1.2.4 控制寄存器

80386 内部有 3 个 32 位的控制寄存器 CR0、CR2 和 CR3，用来控制分页和数学协处理器的操作，保存机器的各种全局性状态，这些状态影响系统中所有任务的运行。它们主要是供操作系统使用。

CR0 的低 16 位称为机器状态字 MSW（Machine Status Word），如图 1.6 所示。

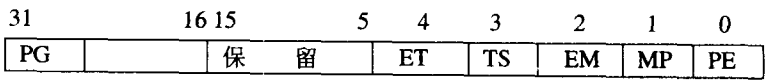

图 1.6 CR0 的含义

CR0 各位的含义如下：

- PE（Protection Enable）

保护模式允许位。设置 PE 位为 1 时使 CPU 进入保护模式。通常，这个操作仅在初始化时执行一次。复位 PE 位即可以使 CPU 从保护模式切换回实地址模式。

- MP（Monitor Coprocessor）和 TS（Task Switched）

协处理器监控位和任务切换位。MP 位表示协处理器是否存在，它影响 WAIT 指令的执行。在多任务系统中，任务切换时，系统硬件总是使 TS 置 1，如此时 MP=1，则 CPU 在执行 WAIT 指令时，会产生一个协处理器无效异常。在异常处理程序中，CPU 把协处理器中前一任务的状态保存起来，再重新装入现行任务的协处理器的状态，并清除 TS 位。这个功能允许操作系统在实现多任务时，不需要在每次任务切换时都保存协处理器的状态，从而节省了 CPU 的宝贵时间。

- EM（Emulated Coprocessor）

协处理器模拟位。当此位设置时，则将使所有的协处理器指令都产生一个协处理器无效异常，在异常处理程序中，使用软件模拟指令来实现浮点指令。当此位被清除时，协处理器指令才会在真正实际的协处理器上执行。

- ET（Processor Extension Type）

处理器扩展类型控制位。在 486 及其后续系列中，由于协处理器总是存在，故 ET 为 1。若 80386 协处理器存在，则 ET 为 1，80386 使用 32 位数据类型。否则，要么存在 80286 或没有协处理器，ET 为 0，80386 使用 16 位数据类型。

- PG（Paging Enable）

分页允许控制位。当 PG=1 时启用分页功能，即启动 80386 片内的分页部件工作。当 PG=0 时禁止分页功能。通常，操作系统在初始化时执行它一次。

PG 和 PE 组合起来为 386 提供了三种工作环境，如表 1.1 所示。

表 1.1 PG 和 PE 的组合

PG	PE	工作环境
0	0	实地址模式
0	1	不分页保护模式。有分段功能无分页功能
1	0	无定义
1	1	分页保护模式。有分段功能和分页功能

CR1 是未定义的控制寄存器。

CR2 是页面故障线性地址寄存器。当发生一个页故障时，CR2 保存线性故障地址。

CR3 是页目录基地址寄存器。CR3 中存放一个表（页目录）的基地址。分页硬件使用这个寄存器，CR3 含有页目录起始点线性地址的高 20 位，低 12 位地址总为 0。在 80386 中 CR3 的低 12 位寄存器应该为 0。

1.2.5 调试和测试寄存器

80386 大大增强了对调试的支持。在 386 CPU 中加入了 8 个 32 位调试寄存器。这些调试寄存器支持指令断点和数据断点。调试寄存器 DR0 到 DR3 用来保存与定义在 DR7 中的四个断点条件之一的相关的线性地址。大家熟悉的 CIH 病毒就采用了 DR0 调试寄存器作为一个病毒发作的判断条件。DR4 和 DR5 在 386 中没有使用，它们保留给 Intel 今后的处理器。

DR6 是调试状态寄存器。如图 1.7 所示。它允许控制调试处理程序决定引起其获得控制权的条件。当 CPU 检测到允许调试异常即 EFLAGS 寄存器中 RF 位为 0 时，则它设置该寄存器的低 4 位。通过解释这些位，控制程序能决定引起异常的原因。例如，条件是在 DR0 指定的线性地址执行了某个操作，操作类型由 R/W0 和 LEN0 确定，即在指定的地址读、写或执行 LEN0 个字节，则 DR6 中的 B0 位被置位。每个 Bn 位可被置位而不管 DR7 中 Gn 位和 Ln 位的状态。

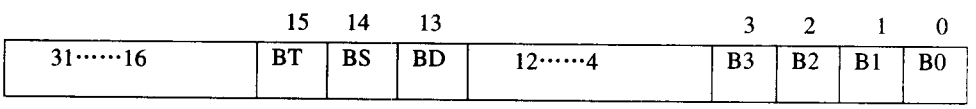

图 1.7 DR6 调试寄存器

若产生任务切换并且新任务的调试陷阱位为允许状态，则在把控制权传递给调试处理程序之前，BT 位被置位。若调试处理程序是由产生单步中断（中断 1）引起的，则 BS 位为 1。如果 EFLAGS 寄存器的陷阱标志 TF 置位，将产生这种情况。若下一条要执行的指令将读或写八个调试寄存器之一，则 BD 位被置位，这一标志当多个调试程序使用调试寄存器时，协调调试寄存器的使用。CPU 不能自动清除 DR6 寄存器。在每次异常之后，程序员必须清除 DR6 寄存器以避免含糊的结果。

DR7 是调试控制寄存器，它定义和分别地使能每个断点寄存器。对于寄存器 DR0 到 DR3 中的每一地址，在 DR7 中与之相对应的 LEN 域和 R/W 域确定了引起中断的动作类型。表 1.2 给出了 LEN 和 R/W 各位的解释。

表 1.2 DR7 的 R/W 域和 LEN 域

R/W 域位	动　　作
00	仅在执行指令时中断（必须 LEN=00）
01	仅在写数据时中断
10	未定义——不使用
11	读或写数据时中断，但不是取指
LEN 域位	动作
00	一字节长
01	两字节长
10	未定义——不使用
11	四字节长

DR7 寄存器的各位的布局如下所示：

LEN3	R/W3	LEN2	R/W2	LEN1	R/W1	LEN0	R/W0	14 …… …… 13	GD	11 …… …… 0	GE	LE	G3	L3	G2	L2	G1	L1	G0	L0

DR7 寄存器的低 8 位可被分别地置位来使能四种断点条件的任何一种组合。L0 到 L3 位使断点在局部的或任务级。这些位在每次任务转换时自动复位以避免在新任务中不需要调试。全局级的 G0 到 G3 不由任务转换复位。LE 和 GE 控制处理器是否减慢其执行速度以便能够报告引起数据断点的正确的指令。若这些位被清除，则会存在处理器的运行超前于断点的可能性。Intel 建议无论数据断点是否有效，这些位都置位。LE 位控制局部任务的这一行为，而 GE 控制全局的这一行为。

80386 还提供了两个 32 位的测试寄存器：TR6 和 TR7。它们是程序员用来证实在芯片加电后转换监视缓冲器 TLB（Translation Lookaside Buffer）操作的正确性的一个机制。TLB 是 386 内使用的把线性地址转换成物理地址的一个高速缓冲存储器。TR6 为测试命令寄存器，其中存放测试控制命令；TR7 为数据寄存器，其中存放测试所得的数据。

第2章

Intel CPU 工作模式

实地址模式
保护模式
虚拟 86 模式