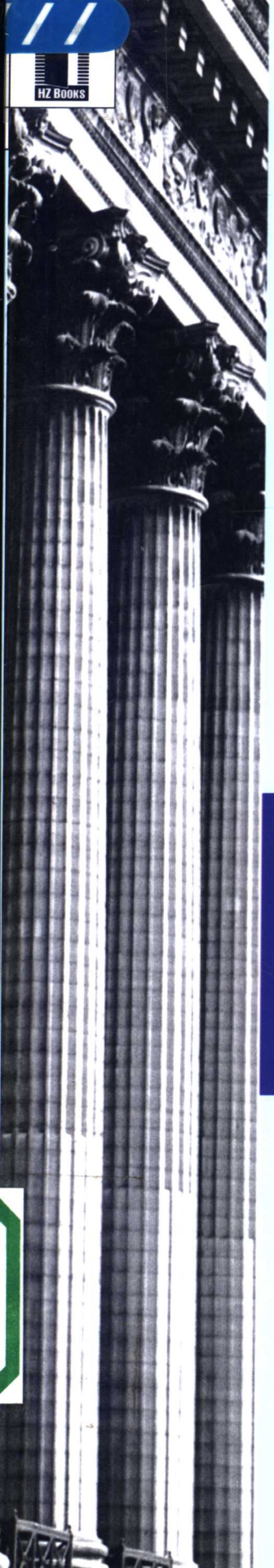

高等院校
计算机教材系列

严义　包健　周尉　编著

Win32汇编语言程序设计教程

机械工业出版社
China Machine Press

高等院校
计算机教材系列

严义　包健　周尉　编著

Win32汇编语言程序设计教程

机械工业出版社
China Machine Press

本书介绍Windows下的32位汇编语言程序设计方法。主要内容包括80x86处理器的基本组成、处理器的指令系统以及寻址方式、80x86处理器保护模式下的运行机制、Win32汇编程序的宏汇编的语法系统、Win32汇编语言的Windows应用程序的编程方法等。此外，本书用Win32汇编程序完成的基于WinSock的TCP/IP协议网络编程、VxD虚拟设备驱动程序设计、COM组件应用等范例都可作为实际应用的参考。

本书适合作为高等院校相关专业汇编语言课程的教材和参考书。

图书在版编目（CIP）数据

Win32汇编语言程序设计教程 / 严义等编著. -北京：机械工业出版社，2004.11
（高等院校计算机教材系列）
ISBN 7-111-15330-8

Ⅰ.W… Ⅱ.严… Ⅲ.汇编语言-程序设计-高等学校-教材 Ⅳ.TP313

中国版本图书馆CIP数据核字（2004）第100473号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）
责任编辑：李云静
北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行
2005年8月第1版第2次印刷
787mm × 1092mm 1/16 · 26.25印张
印数：5 001-7 000册
定价：35.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换
本社购书热线：（010）68326294

前　言

“汇编语言程序设计”是高校计算机、自动化、机电、电子等专业必须学的一门软硬件结合的语言课程，但由于近年来教材内容大大落后于当前计算机技术的发展，因此，这门课程是否还有必要设置，以及如何改革课程内容成为各高校计算机等专业紧迫的问题之一。更有甚者，某些高校计算机等专业教师认为该课程的教材及教学内容仍然是DOS下的8086汇编语言，这不适应当前计算机技术的发展，学生也缺乏学习积极性，因而已取消了该课程。

但是，汇编语言是一种与机器指令操作一一对应的计算机原始语言，是计算机等专业学生学习计算机基础知识必备的语言。这种语言编写出的程序代码效率最高，也是唯一能够利用计算机所有硬件特性并能直接进行控制的语言。在需要软硬件结合开发的计算机应用系统方面，尤其是计算机底层软件的设计方面，汇编语言有着其他高级语言所无法替代的作用；因此计算机等专业开设该课程是必要的。关键是汇编语言课程的教材及教学内容应跟上当前计算机技术的发展，使学生学以致用。基于这个目的，我们经过几年的调查研究，于2001年开始着手准备，对Windows下的32位汇编语言进行了3年的教学研究及试验。

汇编语言和处理器及操作系统都是紧密相关的。随着Windows、Linux以及UNIX等多任务操作系统的出现，DOS操作系统慢慢地退出历史舞台，建立在该操作系统之上的8086汇编语言也逐渐远离了人们的视线。但8086汇编语言和Win32汇编语言又有着相似之处，比如它们都有着相同的指令系统、相同的寻址方式，只不过操作数和地址的长度不同而已，因此本书在描述Win32汇编语言的过程中希望把握该语言的这种特点，并积极地把它和8086汇编语言进行比较，使有8086汇编语言教学经验的教师能够较快地转入Win32汇编语言的教学。由于大多数学生已习惯接受Windows操作系统的内容，本书在编写过程中尽量做到使没有接触过DOS操作系统的学生也可以方便地学习Win32汇编语言。

本书前四章着重介绍了Win32汇编语言的基础知识，第2章以80x86为代表，介绍该处理器的指令系统以及寻址方式；Windows、Linux和UNIX等多任务的操作系统都工作在80x86处理器的保护模式，而不是DOS所运行的实模式，因此在第3章介绍了80x86处理器保护模式下的运行机制，学习这些机制将大大有利于读者对Windows系统的工作机理的理解，同时这些内容也是编写Windows系统底层软件的必要知识；在第4章中，本书详细介绍了Win32汇编程序的宏汇编的语法系统。

本书第5章开始从易到难介绍基于Win32汇编语言的Windows应用程序的编程方法，并选取了几个在程序结构上比较有特点的编程实例，比如最具Windows系统特色的窗口程序、Windows资源编程、控制台程序、动态链接库程序以及工作在零线程的虚拟设备驱动程序。Windows程序结构主要由这几种程序结构组成，其他变化只是使用不同的API函数及不同的逻辑过程而已。比如书中第5章的网络程序，仅仅在前面介绍的对话框资源编程的基础上，根据TCP/IP协议的通信规则使用了WinSock提供的API函数而已。因此，学会Win32汇编语言的这几种程序基本结构，便能编写出各种Windows应用程序。第6章中介绍用Win32汇编语言编写运行在Windows 9x系统下的VxD虚拟设备驱动程序，在实际教学中，第5章和第6章的内容可

根据实际情况增加或减少。

与本书配套的Yy－j01接口实验仪相结合可完成基于Win32汇编语言的微机原理及接口实验，通过这些实验可以进一步让学生了解汇编语言的特点，我们将另编写相应的实验指导书。另与本书配合的相关课件可以到www.hzbook.com上下载，所有书中例题程序我们都一一仔细校对，也均可在该网站上下载。为了便于教师备课及提高学生自学能力，本书不安排习题答案，需要习题答案的老师请与baijian@hziee.edu.cn联系，即可得到全部答案。

在编写本书的过程中，冯建文副教授给予了很大帮助，在此表示感谢。由于本书涉及到的计算机专业知识面很广，在编写过程中我们常常为必须割舍内容与保证内容完整而困惑。编者水平有限，书中一定存在一些不妥之处，诚请广大读者指正。

最后要说明的是，本书适合以下读者阅读:

- 没有任何汇编语言基础的读者。
- 想从DOS汇编语言转到Windows汇编语言的读者。
- 在Windows系统下从事高级语言编程，而又想了解高级语言所封装的一些特性的读者。
- 从事Windows驱动程序开发的读者。
- 从事加密解密、病毒分析以及逆向工程的读者。

目　录

前言
第1章　基础知识 ……1
1.1　微处理器、操作系统和汇编语言的发展史 ……1
1.2　微处理器的内部结构和工作方式 ……2
1.2.1　CPU ……2
1.2.2　80386的寄存器 ……3
1.3　80386的工作模式 ……8
1.3.1　实模式 ……8
1.3.2　保护模式 ……9
1.3.3　V86模式 ……9
1.3.4　三个模式之间的关系 ……10
习题一 ……10
第2章　80x86的指令系统和寻址方式 ……13
2.1　指令寻址方式 ……13
2.1.1　与数据相关的寻址方式 ……13
2.1.2　与转移地址相关的寻址方式 ……20
2.2　指令系统 ……21
2.2.1　数据传送指令 ……22
2.2.2　转换指令 ……26
2.2.3　算术指令 ……28
2.2.4　逻辑、移位、循环移位和位操作指令 ……38
2.2.5　条件测试指令 ……47
2.2.6　I/O指令 ……49
2.2.7　字符串指令 ……49
2.2.8　程序流控制指令 ……53
2.2.9　杂项指令 ……63
2.3　指令格式 ……65
2.3.1　操作码的二进制编码 ……66
2.3.2　寻址方式的二进制编码 ……67
习题二 ……71
第3章　80386的保护模式和Windows系统 …77
3.1　基本概念 ……77
3.1.1　特权级 ……77
3.1.2　虚拟地址空间 ……78
3.1.3　线性地址空间 ……78
3.1.4　物理地址空间 ……79
3.1.5　保护模式下段的概念 ……79
3.1.6　保护模式下页的概念 ……80
3.2　80386保护模式下的存储管理 ……80
3.2.1　分段管理机制 ……81
3.2.2　分页管理机制 ……94
3.2.3　虚拟存储器 ……101
3.3　Windows系统下对内存数据的访问 ……101
3.4　保护模式下转移指令的执行过程 ……102
3.4.1　和控制转移相关的基本概念 ……103
3.4.2　任务内无特权级变化的段间转移 …105
3.4.3　任务内特权级变化的段间转移 ……106
3.4.4　任务间的转移 ……108
3.5　80386保护模式的中断和异常 ……109
3.5.1　中断 ……110
3.5.2　异常 ……110
3.5.3　中断和异常的优先级 ……111
3.5.4　中断和异常的处理过程 ……111
3.6　Windows系统下的中断 ……114
3.7　80386保护模式的I/O保护 ……114
3.7.1　I/O特权级 ……114
3.7.2　I/O许可位图 ……115
3.7.3　I/O敏感指令 ……115
3.8　Windows系统下I/O指令的实现过程 …116
习题三 ……117
第4章　MASM下的Win32汇编语言语法 …119
4.1　程序框架 ……119
4.1.1　指令集选择 ……120
4.1.2　工作模式选择 ……120
4.1.3　区间选择 ……121
4.1.4　程序的起始点和结束点 ……122
4.1.5　注释与换行 ……122
4.2　数据组织 ……123
4.2.1　变量 ……123
4.2.2　数据类型 ……126

4.2.3 变量的访问 ……138
4.2.4 常量 ……144
4.2.5 汇编期的变量和常量 ……146
4.3 子程序 ……149
4.3.1 子程序原型定义 ……149
4.3.2 子程序定义 ……149
4.3.3 子程序调用 ……152
4.3.4 子程序举例 ……153
4.4 程序流控制 ……156
4.4.1 直接使用转移指令 ……156
4.4.2 使用高级语法伪指令 ……159
4.4.3 程序流控制举例 ……168
4.5 宏 ……171
4.5.1 宏的基础语法关键字 ……172
4.5.2 宏里面的循环指令关键字 ……173
4.5.3 宏里面的条件测试关键字 ……175
4.5.4 宏里面其他的关键字 ……176
4.5.5 用于字符串操作的预定义宏 ……178
4.5.6 宏与过程的比较 ……180
4.5.7 宏与TEXTEQU的比较 ……181
4.5.8 使用宏的好处与坏处 ……181
4.6 模块化程序设计 ……182
4.6.1 include ……183
4.6.2 includelib ……183
4.6.3 PUBLIC、EXTERN、EXTRN和EXTERNDEF伪指令 ……184
4.6.4 模块化编程的例子 ……184
4.7 条件汇编 ……187
4.7.1 [ELSE]IF 和 [ELSE]IFE ……187
4.7.2 [ELSE]IFB 和 [ELSE]IFNB ……187
4.7.3 [ELSE]IFDEF 和 [ELSE]IFNDEF …188
4.7.4 [ELSE]IFDIF[I] 和 [ELSE]IFIDN[I]…188
4.7.5 .ERR、.ERRB、.ERRDEF、.ERRDIF、.ERRE等 ……188
4.8 其他伪指令 ……189
4.8.1 列表文件伪指令（见附录A） ……189
4.8.2 选项伪指令（见附录A） ……189
4.8.3 COMMENT和ECHO ……189
4.9 宏汇编（MASM）下的Win32汇编语言的所有关键字 ……191
4.10 实验工具的介绍和编程环境的创建 …192
4.10.1 Windows环境下32位可执行文件的生成过程 ……192
4.10.2 编辑Win32汇编语言源文件 ……192
4.10.3 生成资源文件 ……193
4.10.4 汇编与链接 ……194
4.10.5 模块定义文件（.def） ……197
4.10.6 Win32汇编编程工具小结 ……200
4.10.7 Win32汇编程序的汇编和链接方法 ……200
4.10.8 调试Win32汇编程序 ……202
4.10.9 建立编写程序的环境 ……208
习题四 ……209
第5章 用Win32汇编语言进行应用程序开发 ……215
5.1 基本概念 ……215
5.1.1 Windows API函数 ……215
5.1.2 句柄 ……217
5.1.3 模块 ……217
5.1.4 动态链接库 ……217
5.2 编写消息框程序 ……218
5.3 如何创建Windows下的窗口程序 ……220
5.3.1 消息及传递过程 ……220
5.3.2 回调函数 ……222
5.3.3 编写窗口程序 ……222
5.4 资源 ……230
5.4.1 菜单和加速键 ……230
5.4.2 对话框程序设计 ……239
5.4.3 图标和光标 ……263
5.4.4 位图 ……268
5.4.5 字体 ……270
5.4.6 字符串资源 ……270
5.4.7 版本信息资源 ……272
5.4.8 原始数据资源和自定义资源 ……281
5.4.9 在Win32汇编语言中创建资源的常用方法 ……283
5.5 控制台程序设计 ……284
5.5.1 控制台程序 ……284
5.5.2 用Win32汇编语言实现控制台程序 286
5.6 动态链接库编程 ……289
5.6.1 动态链接库 ……289
5.6.2 如何用Win32汇编语言来创建动态链接库 ……290
5.6.3 使用动态链接库 ……293
5.7 COM编程 ……299

5.7.1 COM简介 ……300
5.7.2 用Win32汇编语言编写COM组件程序 ……305
5.7.3 生成COM组件 ……319
5.7.4 实例 ……324
5.7.5 使用COM ……339
5.8 网络编程 ……340
5.8.1 TCP/IP简介 ……340
5.8.2 Windows Sockets接口 ……343
5.8.3 用Windows Sockets接口编程 ……344
5.8.4 客户机/服务器模型 ……349
5.8.5 编程举例 ……350
习题五 ……370
第6章 用Win32汇编语言编写虚拟设备驱动程序 ……375
6.1 基本概念 ……375
6.1.1 Windows 9x系统的特点 ……375
6.1.2 虚拟机 ……375
6.1.3 VMM ……376
6.1.4 VxD ……378
6.2 虚拟设备驱动程序文件格式 ……382
6.3 VxD源程序的结构 ……384
6.4 用Win32汇编语言编写VxD ……387
6.4.1 VxD的编程语法规则 ……387
6.4.2 VxD程序编写 ……392
6.4.3 虚拟设备驱动程序的生成 ……400
习题六 ……401
附录A 列表伪指令和选项伪指令 ……403
附录B SoftICE快捷键的使用 ……408
附录C ASCII码表 ……409
参考文献 ……410

第1章 基础知识

汇编语言是一种与微处理器和操作系统联系非常紧密的计算机语言。通过编写某种类型的汇编语言，往往可以清楚地了解到该微处理器提供了哪些指令和寻址方式，并且还能清楚地了解到与之相配套的操作系统的工作原理和运行细节。因此，在介绍Windows环境下的32位汇编语言之前，需了解微处理器和操作系统发展历史及汇编语言的变化过程。

1.1 微处理器、操作系统和汇编语言的发展史

1978年6月英特尔（Intel）公司推出了16位的8086微处理器，该处理器采用16位的寄存器和16位数据总线，但是采用了20位的地址总线，最大的寻址空间达到2^{20}，即1MB地址空间。DOS操作系统是运行在该微处理器上的主要操作系统。MASM 4.0是当时的软硬件环境下被广泛使用的一种汇编编译器。

1985年英特尔公司推出了32位的80386微处理器，该处理器采用了32位的寄存器和32位的数据总线，同时采用了32位的地址总线，这使得处理器的寻址能力可以达到4GB地址空间。在设计该处理器时考虑了多用户和多任务的需求，在芯片中增加了保护模式、优先级、任务切换和片内存储单元管理等硬件单元。在以80386为CPU的PC机中出现了像Windows、UNIX这样的多任务操作系统。MASM 5.0汇编编译器就开始支持80386处理器的指令集了，但是还不能够编译工作在保护模式下的Windows程序。微处理器、操作系统和汇编编译器的详细历史，见表1-1、表1-2和表1-3。

表1-1 英特尔微处理器的发展

微处理器型号	生产时间	性能特点（位长，地址线宽度，支持的工作模式）
4004	1971年	4位字长，曾用于计算器上
8008	1972年	8位字长，曾用于TV打字机上
8080	1974年	8位字长，主要应用于控制交通信号灯
8086	1976年	16位字长，时钟频率为4.77MHz，开始用于PC
8088	1978年	16位字长，时钟频率为4.77MHz
80286	1982年	16位字长，时钟频率为（6～20）MHz
80386	1985年	32位字长，时钟频率为（12.5～33）MHz
80486	1989年	32位字长，时钟频率为（25～100）MHz
Pentium Classic	1993年	32位字长，时钟频率为（60～166）MHz
Pentium Pro	1995年	32位字长，时钟频率为（150～200）MHz
Pentium II	1997年	32位字长，时钟频率为（233～333）MHz
Pentium III	2000年	32位字长，时钟频率为（500～700）MHz

表1-2 微软操作系统发展

操作系统名称	生产时间	性能特点
MS-DOS 1.0	1981年	支持16KB内存及160KB的5寸软盘/8086/8088
MS-DOS 2.0	1983年	增加了许多命令，并支持5MB硬盘

（续）

操作系统名称	生产时间	性能特点
MS-DOS 3.0	1984年	增加了对新的IBM AT硬件以及部分局域网功能的支持
MS-DOS 3.3	1987年	支持1.44M的3寸软盘,并支持其他语言的字符集
MS-DOS 5.0	1990年	新增了很好的内存管理和宏功能,增强了DOSSHELL
MS-DOS 6.22	90年代	MS-DOS的最后一个版本
Windows 1.0	1985年	第一版Windows
Windows 2.0	1987年	增强了键盘和鼠标接口
Windows/286	1987年	开始让80386微处理器运行于V86工作模式的操作系统
Windows 3.0	1990年	图形界面更加美观并支持虚拟内存，开始支持80286以上的处理器的保护模式，可访问16M内存
Windows 3.1	1992年	提供了更完善的多媒体功能
Windows NT 3.11	1993年	使用32位编程模式的第一个版本
Windows 95	1995年	可以独立运行而无需DOS支持
Windows 98	1998年	可靠性更强，支持通用串行总线 (USB) 标准
Windows 2000	2000年	功能更强大、稳定性和安全性更好、操作更简单
WindowsXP	2002年	总体性能显著提高，创建了到目前为止Windows操作系统中性能最为优异的一个

表1-3 微软汇编编译器的发展

编译器名称	开发年月	性能特点
MASM 4.00	80年代	这是最先广泛使用的一个MASM版，适用于DOS汇编编程
MASM 5.00	80年代	出现了类似于.code的伪指令，并开始支持80386指令集
MASM 5.10	80年代	有了@@标号，并开始支持OS/2 1.x
MASM 5.10B	1989年	传统DOS编译器中最完善的版本
MASM 6.00	1992年	开始支持.if/.while这样的高级语法，可以用invoke来调用函数
MASM 6.10	1992年	增加/SC选项，可以列出每条指令使用的时钟周期数
MASM 6.11	1993年	开始可以用来编译Win32汇编程序，支持WindowNT，支持Pentium指令
MASM 6.11C	1994年	增加了对Windows 95 VxD 的支持
MASM 6.12	1997年	增加了.686、.686p和.MMX的指令的支持
MASM 6.13	1997年	提供了K3D的声明，开始支持AMD处理器的3D指令
MASM 6.14	2000年	增加了SIMD指令集和16字节的变量类型(OWORD)
MAMS 6.15	2000年	一个很完善的版本

从表1-1到表1-3中我们可以看到，1985年支持32位并且同时支持实模式、V86模式和保护模式的Intel的80386处理器开始上市；1990年微软推出了支持保护模式的Windows 3.0操作系统；1993年微软开发出了可以支持Windows环境下的32位汇编语言程序设计的汇编编译器MASM6.11。

1.2 微处理器的内部结构和工作方式

在1.1节，已经清楚地看到了微处理器的整个发展过程，本节将根据Win32汇编语言的需要，重点介绍目前主流微处理器——80x86微处理器的中央处理器（CPU）、寄存器以及微处理器的工作方式的内容。

1.2.1 CPU

CPU（中央处理器）是英文Central Processing Unit的缩写，这个名词也是大家平时听到最

多的计算机术语之一，那么究竟什么是CPU？它和微处理器的关系又是怎样呢？

CPU是指计算机内部对数据进行处理并对处理过程进行控制的部件，20世纪70年代初，伴随着大规模集成电路技术的迅速发展，芯片集成密度越来越高，CPU可以集成在一个半导体芯片上，这种具有中央处理器功能的大规模集成电路器件，被统称为“微处理器”。图1-1为80386微处理器的结构图。

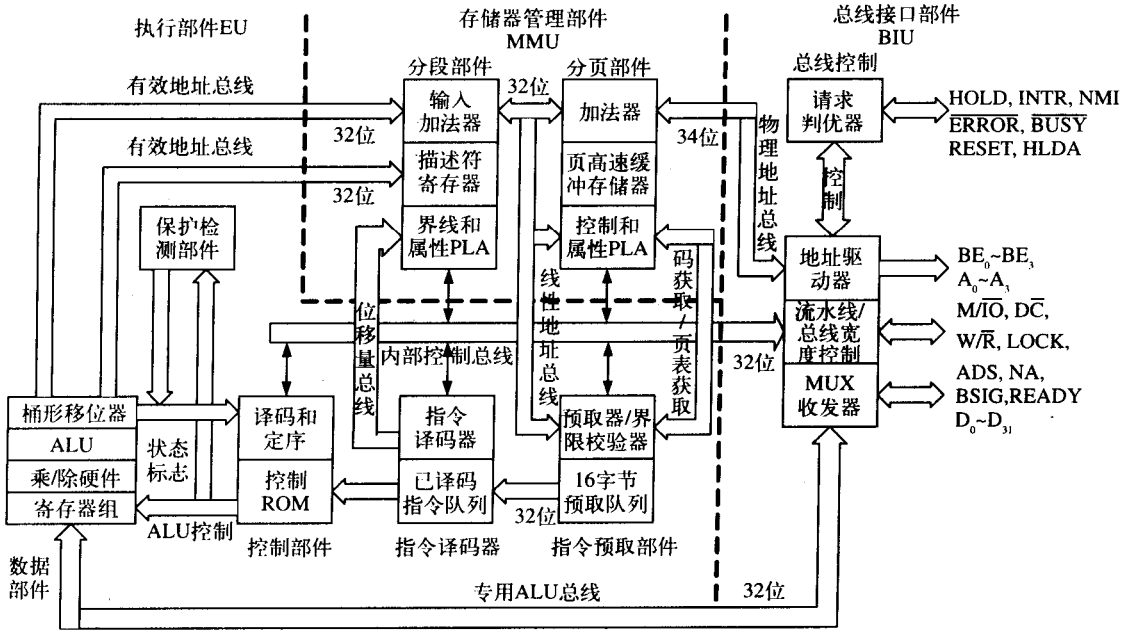

图1-1 80386内部结构框图

执行部件EU由指令预取部件、指令译码部件、控制部件、数据部件及保护检测部件组成。其中数据部件的ALU用来进行算术/逻辑运算，乘、除硬件和桶形移位器主要进行快速乘/除法运算。保护检测部件在程序执行过程中由微指令代码控制，检测静态且与段有关的错误，即检测是否段越界。

存储器管理部件包括分段部件、分页部件。这些部件实现了对存储器的段页式管理，并把指令中给出的逻辑地址转换成存储单元的物理地址。为了提高存储器访问速度，在构成微机系统时可配置高速缓存（cache），这样可构成完整的缓存-主存-外存三级存储体系。

总线接口部件包括地址驱动器、流水线/总线宽度控制逻辑电路、MUX收发器。其作用是与外部联系，访问存储器，预取指令，读/写数据，访问I/O设备等。另外，在总线接口部件中还设有总线请求判优器，用以控制与其他主设备联接，构成多机系统。

图1-1右侧的信号为80386 CPU的外部总线信号，这些信号经过隔离、驱动及有机的组合，构成了系统总线信号。

1.2.2 80386的寄存器

CPU的寄存器是可以用于存放数据的特殊存储空间。但寄存器和内存不同，它位于CPU芯片的内部。因此，CPU访问寄存器里面的数据要比访问内存中的数据快得多。

CPU中的大多数寄存器都有它们自己特定的作用，因此编程时，要谨慎地使用寄存器，不能够随意地把所有的寄存器都用来存放任意的临时数据。

因为x86系列的处理器中的寄存器比较少，因此没有给CPU中的寄存器进行编址，而是直

接对这些寄存器进行了命名。在访问寄存器的时候只要直接使用寄存器名就可以了。

80386处理器中的绝大多数寄存器都是通过8086中的寄存器扩展而来，另外和8086相比，80386还增加了一些新的寄存器。8086中AX、BX、CX、DX、SI、DI、BP、SP、IP和Flags这几个16位寄存器，在80386中都被扩展成32位，并且分别把它们命名为EAX、EBX、ECX、EDX、ESI、EDI、EBP、ESP、EIP和EFlags。除了这几个被扩展的32位寄存器外，80386还增加了两个16位的段寄存器FS和GS，加上8086本身具有的四个16位段寄存器，80386共有六个16位的段寄存器，这使得程序可以在不重新加载段寄存器内容的情况下同时访问六个不同的段。另外，这些段寄存器不像8086中的其他通用寄存器，它们在80386中没有被扩展成32位。

在80386中，虽然Flags寄存器被扩展成32位，并命名为EFlags，但EFlags寄存器并没有改变Flags寄存器中对应位的含义，如在Flags中最低位为进位标志位（CF），在EFlags寄存器中的最低位同样也为进位标志位（CF）。在EFlags寄存器新扩展出来的16位中的最低两位，即EFlags中的第16和第17位，在80386中进行了定义。关于EFlags寄存器各位的具体含义将在后面的说明中给出。

下面将分类介绍80386微处理器中寄存器的具体含义。根据寄存器的用途，可以把所有的寄存器分成以下四大类:

1) 基本结构寄存器组。

2) 系统级寄存器组。

3) 调试寄存器组。

4) 测试寄存器组。

1.2.2.1 基本结构寄存器组

1. 通用寄存器

在80386处理器里，共有8个由8086的16位寄存器扩展而来的32位通用寄存器，用于保存数据和地址，它们命名为EAX、EBX、ECX、EDX、ESI、EDI、EBP和ESP。这些寄存器的低16位可作为16位寄存器单独使用，它们是AX、BX、CX、DX、SI、DI、BP和SP，使用时不影响高16位的值。AX、BX、CX、DX这4个寄存器的高8位和低8位还可以单独用来保存8位数据，它们是AH、AL、BH、BL、CH、CL、DH、DL。上面这些寄存器可以用来保存多种长度的数据，且由程序来指定用于各种用途。在通常情况下通过寄存器的名字就可以知道其用途。

表1-4列出了80386的通用寄存器，图1-2表明了这些通用寄存器的组合关系。

表1-4 80386的通用寄存器

寄存器名	长 度	通常用途	处理器的工作模式
EAX	32	累加器	实模式和保护模式
EBX	32	基址和变址寄存器	同上
ECX	32	计数器	同上
EDX	32	数据寄存器	同上
AX	16	累加器	同上
BX	16	基址和变址寄存器	同上
CX	16	计数器	同上
DX	16	数据寄存器	同上
AL/AH/BL/BH CL/CH/DL/DH	8	8位数据寄存器	同上

（续）

寄存器名	长　度	通常用途	处理器的工作模式
ESI	32	32位源变址寄存器	实模式和保护模式
EDI	32	32位目的变址寄存器	同上
EBP	32	32位基址指针	同上
ESP	32	32位堆栈指针	同上
SI	16	16位源变址寄存器	同上
DI	16	16位目的变址寄存器	同上
BP	16	16位基址指针	同上
SP	16	16位堆栈指针	同上

其中EAX和AX、EBX和BX、ECX和CX、EDX和DX这四组寄存器可以用于各自专用目的。

EAX和AX（来自英文accumulator）作为累加器用，所以它是算术运算的主要寄存器；在乘除等指令中指定用来存放操作数；在所有的I/O指令中都使用这组寄存器与外部设备传送信息。

EBX和BX（base）在计算存储单元地址时常用作基地址寄存器。

ECX和CX（count）通常用来保存计数值，如在移位指令、循环指令和串处理指令中用作隐含的计数器。

EDX和DX（data）在作四字（或双字）运算的时候可以把EDX（或DX）和EAX（或AX）组合在一起用来存放一个四字长（或双字长）的数据；在对某些I/O操作的时候，DX可以用来存放I/O的端口地址。

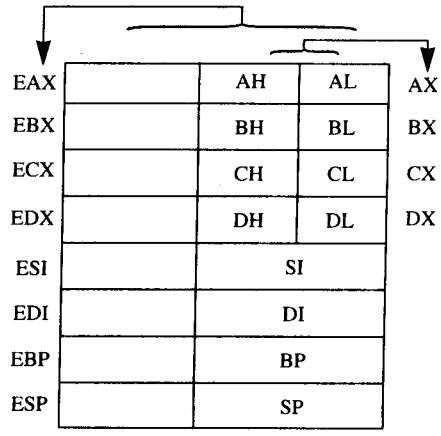

图1-2　通用寄存器的结构

对于另外四组寄存器（ESI和SI、EDI和DI、EBP和BP、ESP和SP），每组中的两个寄存器的功能是相同的，只不过一个用在32位的程序中，另一个用在16位的程序中。这四组寄存器同样也可以用来存放操作数，但它们更经常的用途是在存储器寻址时提供偏移量，因此可以把它们称为指针寄存器或变址寄存器。

ESP和SP（stack pointer）这组寄存器称为堆栈指针寄存器。EBP和BP（base pointer）这组寄存器称为基址指针寄存器。这两组寄存器经常被用来指示堆栈段中某一存储单元的位置。通常，用ESP和SP这组寄存器来指示堆栈栈顶的偏移地址，用EBP和BP这组寄存器来指示堆栈中的某一地址，以便程序能够在不改变ESP指针的情况下方便地访问。如，在程序中调用子程序，在进入子程序后会自动执行如下一段代码：

```
PUSH   EBP
MOV    EBP, ESP
PUSH   子程序参数
......
```

这样，在子程序中我们就可以通过EBP寄存器来访问主调函数传递过来的参数，而不是通过ESP寄存器来对数据进行访问，因为访问参数的时候ESP寄存器正指示堆栈的栈顶，程序不能够通过ESP寄存器的内容来知道参数在堆栈中的位置。

ESI和SI（source index）这组寄存器称为源变址寄存器。EDI和DI（destination index）这组寄存器称为目的变址寄存器。这两个寄存器一般用来确定数据段中某一存储单元的偏移地址。有专用的指令可以实现对这两组寄存器进行自动增量和自动减量，这样可以很方便地用于变址

寻址。另外，在串处理指令中，这两组寄存器作为隐含的源变址寄存器和目的变址寄存器。

2. 段寄存器

80386微处理器有6个段寄存器，每个都是16位长，CS是代码段寄存器，DS、ES、FS、GS是数据段寄存器，SS是堆栈段寄存器。表1-5给出了这些段寄存器在微处理器相应工作模式下的通常用途。

表1-5 段寄存器的用途

寄存器名	长 度	通常用途	微处理器的工作模式
CS	16	存放代码段的段基址	实模式
DS	16	存放数据段的段基址	同上
ES	16	存放附加数据段的段基址	同上
FS	16	存放附加数据段的段基址	同上
GS	16	存放附加数据段的段基址	同上
SS	16	存放堆栈段的段基址	同上
CS/DS/ES FS/GS/SS	16	存放选择子①	保护模式

① 关于段寄存器作为选择子在保护模式下的工作过程将在第3章具体介绍。

3. 程序指针寄存器

当微处理器工作在保护模式时，程序指针寄存器是一个32位的寄存器，名为EIP。它保存下一条待取出指令的代码段的段内偏移地址。EIP的低16位名为IP，是16位的程序指针寄存器，供16位寻址时使用，当微处理器工作在实模式的时候就是采用16位寻址的。在程序运行的过程中，程序指针寄存器始终指向下一条要被执行的指令的偏移地址。该寄存器与CS段寄存器配合来确定下一条指令的物理地址。

当该寄存器中的地址送到存储器后，控制器就可以取得下一条要执行的指令，而控制器一旦取得了这条指令就马上改变该寄存器的值，使它指向下一条要执行的指令。

4. 标志寄存器

标志寄存器EFlags是一个32位寄存器，低16位名为Flags，也是在16位寻址方式时使用。EFlags的第1、3、5和22～31位，英特尔公司未定义，其余的19位通常都是用来反映操作结果的状态（S），但也有少数位起着控制作用（C），其余为系统标志（X）。图1-3展示了标志寄存器里面各个位的内容。

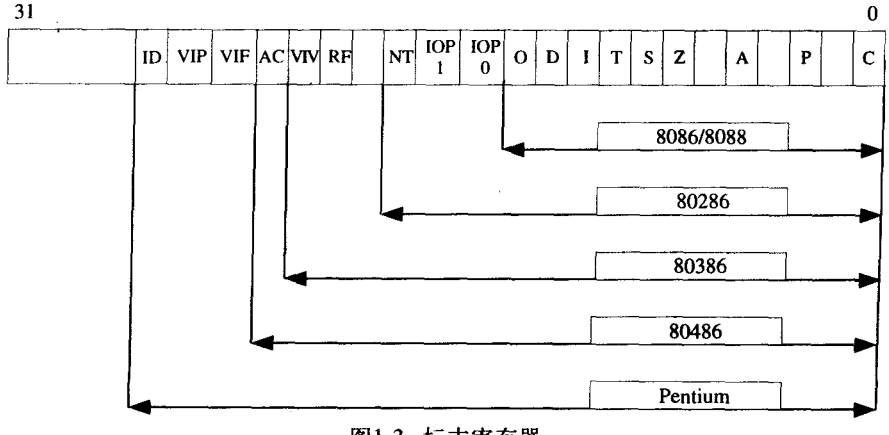

图1-3 标志寄存器

下面我们将用表格来概括标志寄存器各位的含义。表1-6中“位”一项中我们将用D31~D0来表示标志寄存器里面的各个位。

表1-6 标志寄存器各位的含义

类 别	符 号	位	含 义	说 明
A	CF	D0	进位标志	记录运算时，最高有效位产生的进位值
A	PF	D2	奇偶校验标志	用来为机器中传送信息时可能产生的代码出错情况提供校验条件。当结果操作数中1的个数为偶数时该标志置1，否则置0
A	AF	D4	辅助进位标志	用于记录运算时第三位（半个字节）产生的进位值
A	ZF	D6	零标志	运算结果为0时，该标志位置1，否则置0
A	SF	D7	符号标志	运算结果为负时，该标志位置1，否则置0
C	TF	D8	陷阱标志	用于调试时的单步控制操作。TF为1，表示每条指令执行完后产生陷阱，由系统控制计算机；否则，不产生陷阱，CPU正常工作
C	IF	D9	中断允许标志	当该标志位为1时，允许CPU响应可屏蔽中断请求，否则关闭中断
B	DF	D10	方向标志	为1时，每次操作后使变址寄存器SI和DI减小；为0时，则使SI和DI增大
A	OF	D11	溢出标志	在运算过程中，如果操作数超出了机器所能表示的范围，称为溢出，此时OF位置1，否则置0
C	IOPL	D12,D13	I/O特权值	在保护模式下用于控制对I/O地址空间的访问
C	NT	D14	嵌套任务标志	该标志位用于指示当前任务嵌套于另一任务之内。CPU用此位来控制被中断和被调用任务的连接
C	RF	D16	恢复标志	该位由调试器设置，它可以暂时关闭调试故障，以便在一次调试故障之后，指令可以重新启动工作，以免立即引起另一次调试故障
C	VM	D17	V86模式标志	该位置位时表明当前工作在虚拟86（V86）模式下
C	AC	D18	对准检查标志	当AC位0时，对访问内存单元的地址不加以限制，只是访问时间较长。否则，如果访问内存单元的地址不符合要求，则系统自动实现地址对准，以提高访问速度。该位适用于特权级为3的用户工作模式下访问
C	VIF	D19	虚拟中断标志	这两个标志位组合在一起，允许多任务的环境下的应用程序有虚拟的系统IF标志，VIF即为IF位的虚拟映像
C	VIP	D20	虚拟中断挂起标志	
C	ID	D21	识别标志	若这位能被置位和复位，则指明这个处理器能支持CPUID指令。该指令能提供处理器的厂商、系列和模式等信息

表1-6把标志寄存器中的各位进行分类介绍。其中类别A表示条件码标志，这类标志用来记录程序中运行结果的状态信息，它们是根据有关指令的运行结果由CPU自动进行设置的。这些状态信息往往作为条件转移指令的转移判断条件。

类别B为方向标志位，这类标志用来在串处理指令中控制处理信息的方向。

类别C为系统标志位，这类标志可用于I/O、可屏蔽中断、程序调试、任务切换和系统工作方式等的控制。一般应用程序不必关心或修改这些标志位，只有系统程序员或编写I/O设备控制驱动程序时才需要访问其中的有关位。

1.2.2.2 系统级寄存器组

系统级寄存器组是指，这些寄存器只能由特权级最高的程序访问。它们由操作系统核心维

护和使用。因此，这些系统级寄存器只在保护模式提供服务。系统级寄存器组包括4个表（或段）的基地址寄存器GDTR、IDTR、LDTR、TR和4个控制寄存器CR0、CR1、CR2、CR3。

1. 基地址寄存器

基地址寄存器主要为了支持处理器实现保护模式下的存储管理和多任务机制而设置的，其实这几个在80286处理器中就已经出现了。这些寄存器分别是：全局描述符表寄存器GDTR、中断描述符表寄存器IDTR、局部描述符表寄存器LDTR和任务状态段寄存器TR。

2. 控制寄存器

和8086处理器相比，80386新增加了四个控制寄存器，分别命名为CR0、CR1、CR2和CR3。在80386中，这些寄存器主要用来控制存储单元的分页机制等，服务于80386处理器在保护模式的工作。在80286 CPU中只有机器状态字MSW，相当于CR0中的低16位。另外，到Pentium又增加了1个控制寄存器CR4，并对CR0的CD和NW位进行重新定义。

由于这些系统级寄存器主要服务于处理器的保护模式，因此我们将在第3章再对这些寄存器进行详细介绍。

1.2.2.3 调试寄存器组

在80386处理器中还增加了8个调试寄存器。这些调试寄存器主要为调试程序提供服务，如支持微软的CodeView调试程序在被调试的程序中设置断点。因此，像CodeView这样的调试工具只能用于80386或80386以后的处理器。当然，我们不能在编写应用程序的时候来使用这些调试寄存器。

1.2.2.4 测试寄存器组

80386处理器中还增加了一组测试寄存器，这组测试寄存器主要用来供系统设计人员测试机器上电后处理器是否正常工作。和调试寄存器一样，这组寄存器同样也不能供一般程序员来编写应用程序使用。

1.3 80386的工作模式

80386有三种工作方式，分别是实模式、保护模式和V86模式。常用的Windows和Linux这些多用户多任务的操作系统都是工作在微处理器的保护模式下，而这些操作系统相对于以前DOS单任务操作系统的优势是不言而喻的。因此80386的保护模式也是本书中重点介绍的内容之一。从前述的微处理器的发展史可以知道，在80386之前的微处理器还不具有保护模式，因此在80386之前的操作系统和所有的应用软件都是工作在与8086同类型的微处理器上面的，为了不浪费这些宝贵的软件资源，80386处理器还需要具有向上兼容的功能，即80386的实模式。因此，当80386工作在实模式的时候，它就像一个快速的8086处理器，但是在实模式下80386不能实现多用户多任务的目的。V86模式就是为了解决这个问题而产生的。下面将详细来讨论80386的这几种工作模式。

1.3.1 实模式

80386处理器工作在实模式时，A_0 ~ A_{19}的20根地址线可用，可寻址空间为1MB。在这种情况下，80386与8086、8088的寻址方式一样，把段寄存器内容乘以16作为段基址，再加上16位的段内偏移地址形成最终的20位内存物理地址。由于段内偏移地址是16位的，所以每个段的最大长度为2^{16}即64KB，这就是所谓的64KB段长限制。在实模式下最高物理地址为0FFFFFH，若段基址加上偏移地址计算出的物理地址超过20位，则超出位被丢弃。例如段地址和偏移量都为0FFFFH，那么计算出的地址为10FFEFH，实际送出的物理地址为0FFEFH。

在实模式下，80386不支持优先级，所有程序都可以执行特权指令，包括读写系统的一些特殊寄存器，如：GDTR、LDTR、IDTR和TR等。实模式下80386不支持硬件上的多任务切换，DOS操作系统就是运行在实模式中，它是单任务的操作系统。

和8086处理器一样，实模式下80386也用中断向量表来定位中断服务子程序地址，中断向量为4个字节，由两个字节的段地址和两个字节的偏移地址组成。把中断向量号乘以4作为访问中断向量表的偏移地址，这样就可以得到该中断对应的中断向量，从而直接进入到中断服务子程序。

工作在80386的实模式下，可以用in/out等输入/输出指令对I/O地址空间直接进行读写操作。

实模式的80386处理器和8086处理器的工作过程非常相似，除了80386新增了一些寄存器并加快了处理速度之外，它们最大的区别就是实模式的80386处理器具有32位的数据线和32位的通用寄存器。使用32位通用寄存器可以方便对数据的处理，而且还可以用PUSHAD和POPAD指令来对80386的8个32位通用寄存器进行压栈和出栈操作，但是使用了32位寄存器及其相应指令的程序是不能在8086处理器上运行的。

1.3.2 保护模式

80386处理器提供了实模式来兼容先前的8086处理器，但这时8086程序实际上只是运行得快了一点，对CPU的资源还是独占的，80386有许多优越的性能没有得到应用，只有在保护模式下，80386才能真正发挥最大的作用。

在保护模式存储器中，仍然由程序员在程序中指定逻辑地址，只是机器采用分段、分页的虚拟内存管理机制的方法来求得相应的物理地址。因此对程序员编程来说，并未增加复杂性。在保护模式下逻辑地址由选择器和偏移地址两部分组成，选择器为段寄存器，但它不能直接表示段基地址，而由操作系统通过一定的方法取得段基地址，段基地址和偏移地址相加成为线性地址。

在保护模式下，全部32条地址线有效，可寻址高达4GB的地址空间。通过分段机制可以把程序中由段寄存器（选择器）和偏移量构成的二维逻辑地址空间转换成一维的4GB线性地址空间，再由分页机制将线性地址转换成物理地址。通过虚拟内存可以让比实际物理内存空间大的程序很好地运行。而且分段管理机制和可选的分页管理机制，还为存储器的共享和保护提供了硬件支持。

保护模式还支持多任务的工作环境，依靠80386的硬件系统，可以使用一条指令或一个中断就能够快速地进行任务内的切换和任务间的切换，并保护任务环境。

保护模式下的80386提供0～3共4个特权级，让操作系统运行在优先级最高的0级上，让应用程序运行在优先级较低的级别上，再配合完善的特权检查机制后，既能够实现资源共享又能保证代码和数据的安全保密及任务的隔离。

在支持多任务的保护模式环境下，80386不再简单地通过中断向量表来得到中断服务子程序的入口地址，而是通过各种门描述符并配合完善的特权级检查机制来支持多任务环境的实现。

同样为了支持多任务工作环境的实现，那些用于I/O操作的指令就不能够直接对I/O端口进行读写操作了。80386通过优先级和I/O许可位图来对这些I/O指令进行安全检查，从而很好地实现多任务工作环境下的I/O操作。

第3章将会给出保护模式下多任务环境的具体实现过程。

1.3.3 V86模式

前面已介绍过，80386的实模式与8086兼容，可以运行DOS及其平台的几乎所有软件，但在实模式下，处理器不能发挥自身的优越性能：不能支持多用户、多任务操作系统的运行，并