80386/387体系分析

H

中国科学院希望高级电脑技术公司

目　录

第一章 引 言

80386是不断发展的微处理器家族中的最新成员。为了最恰当地评价当代微处理器，我们必须回顾过去，看看它们都是从哪里起始，又是如何发展到现在这个程度的。本章包括这样的历史概述，以及计算机的某些基础知识，计算机中数的表示方法，和计算机中堆栈的使用。

本书的余下章节是这样组织的：第二章和第三章介绍386的基本体系结构。386的32位体系结构是从8086和286以来的最显著的变化；即使你已熟悉了8086和286，你也会想阅读第二章和第三章中的大部分内容。第四章介绍387的体系结构，大部分部是与8087和287体系结构相似，如果你熟悉浮点指令系统的话，你可以跳过这一部分。第五章涉及与8086和286保持兼容的386特性。第六章叙述386给予操作系统的支持；在这些方面386与286是相似的。如果你熟悉286，或者对操作系统软件不感兴趣的话，你可以跳过第六章。最后，第七章叙述由weitek公司设计的一个接口，它提供甚至比387还要快的浮点处理。

1.1 微型计算机的历史

现代微型计算机的发展可以分为两个阶段：缩小阶段和提高阶段。在缩小阶段（大约是1940～1970年），计算机由于制造包含的组件技术的提高而变得越来越小。随着不大于一个邮票的计算机的出现（尽管由1970年的标准还是原始的）宣告了这一阶段的结束。提高阶段（大约是1970年～现在）出现了与更大的计算机功能一样强大的邮票大小的计算机。

1.1.1 缩小阶段

在六十年代，从无线电和电视到计算机，所有的电子设备都是由庞大的真空管建造的。那个时代的计算机有时也称为第一代计算机，如IBM650和704。这些计算机安装在一个大的房间里，包括几排大的电子设备。在六十年代后期，晶体管和其它固体设备开始替代真空管。使用这种技术的计算机称为第二代计算机(如IBM7090和Burroughs B5500)。

在60年代，许多分立电子原件（晶体管、电阻等）联起来形成更复杂的称为集成电路的电子原件。一块集成电路是在比一枚邮票还小的硅片上制造的。它安装在一个可以扦入系统的象蜈蚣一样的结构上。这种可扦入的集成电路就是现在的芯片。由集成电路制造的计算机是第三代计算机（IBM 360，GE 635，Burroughs B6700）。但是集成电路技术在进一步发展，到了70年代初期，计算机的许多部件可以集成于一个单片上（Intel 4004和8008）。这导致了术语芯片上的计算机的产生。

芯片上的计算机称微型计算机或微处理器。尽管这两个术语有时可互换地使用，但是有一些差别。微处理器是单个芯片。它通常含有计算机的逻辑和算术部件，但是没有计算机的存储器和输入/输出设备。微计算机是由微处理器芯片、存储器芯片和输入/输出设备组成的整个计算机系统。有时这整个的计算机系统包含在一个芯片上（Intel 8048），这

种微型计算机称为单片微型计算机。

1.1.2 提高阶段

随着在1971年出现了Intel的4004和8008处理器，开始了微处理器时代。这是第一代微处理器。这两个芯片都是设计用于特殊用途的。4004用于一个计算器中，而8008用于一个计算机终端中。这些微处理器用于替代复杂的、常规设计的电路。现在有另一种设计复杂电子电路的方法，人们可以设计包含微处理器芯片的简单电路并为它写模拟复杂电路功能的程序。这一阶段的某些典型应用是电子取款机、智能打字机、交通安全灯控制器和微波炉等。这些应用称为嵌入应用，因为微处理器嵌入在不是计算机的其它设备中。

1974年，当8008发展成为8080（第二代微处理器）时，在微处理器领域发生了一场革命。8080并不是用于电子逻辑的替换物，而是出现于如果没有微处理器就是不可行的应用中。象字处理机、飞机惯性制导系统、和巡航导弹这样的应用需要易于修改（重编程）和适合于小型空间的能力。微处理器提供这些能力，8080成为使用的“标准”微处理器。

这仅是从专用的可重编程电路如字处理机走向基于微处理器的通用计算系统的一小步。早期这样的机器之一是1974年问世的Intel Micro-processor Development System（MDS）。尽管这可能是第一台个人计算机，由于种种原因Intel不想这样叫。首先，它们不想引起工业巨头（IBM和DEC）的注意，这些工业巨头都是Intel的用户；其次，Intel知道多数大公司的购货政策：一名工程师可以很容易地预订一台实验设备（如一台开发系统），但是需要一台计算机则必须通过数据处理部门。

现在微处理器能够完成老式的庞大的设备的计算任务，而且价格是这样的便宜，使得一般的业余爱好者都可以看到它。除了Intel公司外，许多公司开始制造8080芯片，有些公司（特别是Zilog）制造8080的高级版（Z80）。Intel公司自己在1976年引入了称为8085的高级版。但是在1978年Intel公布86系列微处理器的第一个成员8086之前，8080的基本特性都没有得到很大的改变。

1.1.3 86系列微处理器

直到这时，微处理器的发展还是跟随以前的巨型计算机的历史发展。但是现在Intel希望迈出把微处理器推为主导地位的一大步。为了达到这个目的，他们着手于一种先进的处理器开发。这种处理器，内部称为8816，包含了当时知道的最先进的操作系统和程序设计语言概念。遗憾的是，8816的开发比开始预想的时间要长得多。Intel承受着继续的压力，设计出比Zilog Z80更好的处理器。这提供了开发8080的后续者即8086的动力。在推出8816之前，8086是用于帮助Intel渡过难关。那时8816称为8800。

在1978年Intel设计出8086。这是第一个一次可以同时处理16位数据的微处理器（8080处理8位数据）。另外两个公司也很快宣布他们的16位处理器的计划，即Zilog公司的Z8000和Motorola公司的68000（当时流行大的数字）。这是第三代微处理器的开始。

1979年出现了8086的8位版8088。两年后IBM宣布进入个人计算机领域，并宣称8088将要作为它的第一台个人计算机IBM PC的处理器。由于具有这样的后盾作保证，8086

和8088开始成为最流行的微处理器。

随着IBM PC的出现，微处理器工业的性质发生了改变。生产的大量16位微处理器现在被用于个人计算机，而不是用于各种嵌入的应用（交通信号灯、微波炉等）。

1982年Intel推出8086和8088的增强型，即186和188。这些芯片除了包含 8086／8088外，而且还具有一些在微处理器系统中需要的支持芯片。这并不增加任何新的能力，而是减少了一个完整系统所需的芯片的数目，因此也减少了成本。

同在1982年，Intel又推出了286。这表现了Intel跳过8086的第一大步。关于286的计划在个人计算机的广泛使用之前的四年前就已开始。开发286的主要原因是8086 缺少存储器管理，这是Intel的竞争对手（即Motorola）在他们的16位处理器中提供的特性。Intel开始开发286的另一个原因是在70年代后期8800的继续忽略。事实上，当8800在1982年最终成为（又再次命名）iAP×432时，它一直为人们所遗忘。

这时个人计算机工业正在蓬勃发展。IBM为它的PC机和PC／XT机设置了标准，其它公司也在生产IBM机器的近似完全的复制品。1983年Apple公司推出了对IBM PC构成竞争威胁的机器Macintosh，它使用Motorola的68000系列中的一个微处理器。第二代IBM个人计算机是IBM PC／AT（AT表示先进技术（Advanced Technology）），它是在1984年面世的，使用的是286处理器。AT也成为被广泛复制的标准。

在Intel发展86系列微处理器的同时，Motorola正在发展它的 68000 系列微处理器。该系列的前两个成员即68000和68010有32位寄存器，但是仅容纳24 位地址。而且，与Intel的担心相反，68000系列的早期成员在支持存储器管理中有非常大的困难。1984 年推出的68020克服了这两个方面的缺陷。

最后，Intel在1985年正式公布了386微处理器，它是86系列微处理器中最大的进步。这不仅是Intel对下一代IBM兼容的个人计算机的奉献，它也是对Motorola的68020 32位微处理器的回答。386引入了32位通用寄存器组，一个非常大的扩充存储空间，一组更完全的存储器索引方式，存储器分页，以及一个提高的8086模拟的手段。1987年IBM推出个人系统／2（Ps／2）系列个人计算机，它把386选择为高档模型的处理器，早期86 系列微处理器作为低档模型的处理器。有些人推测，386是最终的微计算机体系结构，因为没有什么迫切的原因要求寄存器尺寸和存储器地址大于32位。我们感觉到把任何东西称为计算机世界中的终点是危险的：你可以得到保证，即使这本书正在写时，Intel正在设计 386 的后续者。

在微处理器领域的发展过程中，微协处理器领域也正在进行并行的发展。协处理器是为通用处理器执行特定功能的附属处理器。第一个流行的协处理器是8087，它为 8088 和8086实现浮点运算。在开发8087的同时，IEEE正在研究一份关于微处理器浮点算术运算的标准。8087是第一个实现当时提出的标准的数学处理器。

8087除了作为8086的协处理器外，在80186和80188处理器问世时它也作为这些处理器的协处理器。由于80286中不同的协处理器接口，为了实现与286一起工作的数学协处理器，需要8087的更改版，即287。最后，386支持的32位总线给出另一个协处理器接口。到了这个时候，提出的IEEE标准已作了修改而且正式接纳为一个标准。既然不管怎样，对于386都需要一个新的协处理器，这个协处理器设计来实现新采纳的标准。因此这个新的

协处理器，即387，与287和8087有些不兼容。

这里顺便介绍一下术语。在一个时期术语曾经是清楚的：例如每一个都叫8080为“8080”。当引入8086时，每个人都叫它为“8086”。Intel为了使他们的产品听起来印象更深，用“iAPX”替代了前缀“80”（请不要问APX是什么意思）。他们反过来将8086称为iAP×86，而且将8086的后续者称为iAP×186，iAP×188，和iAP×286。但是在Intel之外，iAPX前缀决没有引起人们的注意。人们深深地习惯于型号的称谓：80186，80188，80286，80287。Intel最终屈服了；在Intel文献中没有出现iAPX×386的提法，80386和80387是官方的名称。对于8086和8087我们使用全型号来称谓，但是为了简单性，对于后续处理器的名字我们按照通常的省略“80”前缀的方法：186，188，286，287，386，387。我们将整个处理器系列称为“86系列”。

1.2 计算机基础

假定你对计算机的基本概念已经熟悉，因此我们在这里只是对这些概念简单地回顾一下。计算机从一个输入设备获取数据，并进行相应的处理，然后将最终结果传送给输出设备。计算机所完成的特定处理是由称为程序的一列指令来指定的。程序存放在计算机的内部存储器中。

计算机的操作是由中央处理单元（cpu）或简称为处理机来控制的。处理器从内存中提取指令，并给指令译码以确定要实现什么样的动作，最后通过实现这些动作执行指令。为了完成这些动作，处理器有时必须将一些控制信号发送给计算机内的其它设备。在指令执行过程中完成的操作包括移动数据和对数据实现相应的运算。计算机内部存储器用于为运算提供输入和保存运算的最终结果。

为了说明所有这些成份是怎样联系在一起的，我们分析一条加法指令的执行过程。处理器给存储器发送一个信号，请求下一条指令。存储器将一条指令传送给处理器来响应该信号。然后处理器对这条指令进行译码并发现它是一条加法指令。然它完成下列操作：（1）给存储器发出两个信号，告诉它传送两个值给处理器；（2）将接收的两个值相加；（3）给存储器发出信号，告诉它接收加法的结果。

存储器是顺序存储单元的集合，每个单元都有一个唯一的地址。每个单元含有一组位（简称二进制数字）。这些位是存储单元的内容。每一位的值只有两种可能：0或1。

寄存器象存储器一样也用于存放中间运算结果。寄存器在处理器的内部，因此访问寄存器中的值比访问存储中的值更容易更快。处理器内部的标志用于记录处理器的状态和运行情况。有两种标志——一种是记录关于前面执行的指令的影响的信息（状态标志），另一种是控制计算机的操作（控制标志）。状态标志的一个例子是指出一个结果对计算机来说是否太大而不能处理的标志。控制标志的一个例子是告诉计算机以一个更低的速率（如每小时一条指令）执行指令的标志。某一标志即为状态标志也为控制标志也是可能的。例如386的NT标志（在后面讨论）。

1.3 数的表示

我们习惯于用一系列十进制数字表示整数，如365。它被解释为：3个百，6个十，

5个1，即三百六十五。它有时称为以10为基的表示法。计算机的整数通常是用一系列二进制数字（位）来表示，如11010。这是26的以2为基的表示法：1个16，1个8，0个4，1个2，0个1。只要我们记住1加1等于10（1个2和0个1）而不等于2，我们就可以直接对二进制数进行加、减、乘、除。例如：

```
  1001     9的二进制表示
+ 0101     5的二进制表示
  1110     14的二进制表示
```

尽管计算机对最小的位不感到困扰，但是我们对长的二进制数字系列还会感到烦恼。例如，10110101是181的二进制表示。为了使事情更加简单，我们通过将长的二进制数字序列每4位为一组进行划分，设计出压缩长的二进制数字序列的方案。每个4位组（有时称为半字节）由如表1.1中所示的单个字符来表示。因此10110101压缩为B5。这个数称为十六进制数，如果我们出生时有16个手指的话，这就会成为我们使用的数制。

对于描述正数和零，二进制表示是完善的。但是当我们要表达负数时，我们需要另外的机制来指出数的符号。最简单的方法是使用数的最高有效位（最左边的位）来指明符号。例如：

```
0000  0100    是+4
1000  0100    是-4
0111  1111    是+127
1111  1111    是-127
```

表1.1　十六进制表示

4位组	十六进制数字	值
0000	10	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

这种表示法称为符号大小表示法，而且有一个严重的缺点：它需要一组新的算术运算规则。当我们试图使用二进制算术从0减去+1并期望得到-1时，这是十分显然的。

```
  0000  000          符号大小表示的零
 -0000  0001         符号大小表示的+1
  1111  1111         符号大小表示的-127
```

如果我们希望对有符号数使用与对无符数使用的相同的二进制算术运算，我们需要一个1111 1111表示-1而不表示-127的带符号数的表示方法。而且-1减去1应是-2。我们完成这样的减法看看-2的表示是什么。

```
  1111  1111
 -0000  0001
  1111  1110
```

这种表示法称为2的补码表示法。它有这样的特性：二进制加法和减法会给出正确的2的补码结果。例如：

```
  0000  0011        +3的2的补码
 +1111  1110        -2的2的补码
  0000  0001        +1的2的补码
```

它也有这样的属性：每个非负数（正数或零）的最高效位是0，每个责数的最高有效位是1。因此，与符号大小表示法一样，这一位也是一个符号位。

2的补码数的符号可以通过改变每一位的值并加1来改变。例如，我们可以从+5的2的补码表示得到-5的2的补码表示。

```
  0000  0101        +5的2的补码
  1111  1010        改变每一位的值
 +0000  0001        +1的2的补码
  1111  1011        -5的2的补码
```

当加长2的补码数时，我们必须小心。如果一个8位2的补码数扩展到16位（因此它可以加到一个16位2的补码数），我们必须考虑附加的8位。

假设我们将0000 0001（+1的2的补码）加到0000 0011（+3的2的补码）。在这种情况下，毫无疑问我们简单地在+1的左边附加8个零，然后进行相加：

```
  0000  0000  0000  0011    +3的2的补码
 +0000  0000  0000  0001    +1的2的补码
  0000  0000  0000  0100    +4的2的补码
```

但是，如果我们将1111 1111（-1的2的补码）加到0000 0011（+3的2的补码），我们必须在-1的左边附加8个1（附加零会使它成为一个正数）。则加法是：

```
  0000  0000  0000  0011    +3的2的补码
 +1111  1111  1111  1111    -1的2的补码
  0000  0000  0000  0010    +2的2的补码
```

因此8位数对16位数的扩展看起来是：

值	8位表示	16位表示
+1	0000 0001	0000 0000 0000 0001
-1	1111 1111	1111 1111 1111 1111

扩展2的补码数的规则是：将数的左边每一位扩展为与原始符号位相同的值。这个过程称为符号扩展。

1.4 堆栈

堆栈是在微处理器及更大的机器中经常发现的概念。堆栈的其它名称是“下压列表”或“后进先出队列”。这些名称是根据存放餐馆盘碟的设备想象得来的。当一个新的盘子放到盘架的顶端时，它将它下面的所有盘子向下压一格。当从架的顶端取走一个盘子时，所有的盘子都向上弹一格。最后放到架子上的盘子是第一个取走的盘子。

为了理解这与计算机有什么关系，我们必须看看子程序：子程序（有时称为过程）是被调用执行特定任务的程序的部分。这提供了将要解决的整个问题划分成更小或更简单的部分的方法。一个子程序本身会调用另外的子程序的进一步划分要做的工作。一个子程序完成它的工作以后，它将控制返回到调用它的程序。结果是一串子程序，每一个调用另外的子程序，直到最后一个调用的子程序决定返回。换句话说，最后一个被调用的子程序是第一个返回的子程序。

当一个子程序被调用时，有一些信息要被保存起来。这会包括一些寄存器的当前内容和标志的当前设置。它当然包括子程序最终要将控制返回到的调用程序的地址。当子程序完成它的任务时，它会提取保存的信息，使得它能恢复影响的寄存器的内容，将标志设置成原来的设置，并使用“返回地址”将控制返回到合适的指令。但是由于调用的最后一个子程序是第一个返回的子程序，保存的最后一些信息必须首先提取。因此信息必须象餐馆放盘子那样存放。

到目前为止，我们已经描述一个堆栈怎样动作，以及为什么堆栈在计算机中是十分有用的。现在我们讨论怎样实现计算机中的堆栈。因为堆栈必须存放信息，它必须是某种存储器。实际上，可用的存储器的任何部分（只要它可以被读写）可以作为堆栈使用。所需要的是指向存放在存储器的堆栈部分的最后一批信息的指针。这个指针通常称为堆栈指针，指针所指向的信息通常称为栈顶。当一新批的信息存放到堆栈时(称为压栈的过程)，堆栈指针的改变成指向下一内存位置，且新的信息存放在该位置。当从堆栈取走信息时（称为弹栈的过程），该信息是从堆栈指针指向的内存位置提取的，而且堆栈指针再次更改，但是这次是向相反的方向进行的。

1.5 迈向386

我们已经介绍了计算机以及一些相关的一般概念，我们可以学习一个特定的处理器，即386。下一章叙述该设备的机器组织。

第二章 机器组织

2.1 概述

描述一台计算机的方法之一是描述组成该计算机的功能部件。这些功能部件和它们之间相互作用的描述有时也称为计算机的体系结构，包括计算机中有多少寄存器，寄存器的功能是什么，可连接多少内存，可寻址多大内存，以及可以使用什么样的输入／输出设备。

386是一个含有组成一个计算机的大部分部件的单个集成电路芯片。386芯片上含有控制计算机的所有功能的电路以及所有寄存器和标志。芯片中不含有内存和输入／输出设备，但是它们可以容易地连接到芯片上以组成一台计算机。芯片上所有这些部件的集合有时也称为处理器。

一块运行正常的386芯片会装在一块电路板上，该电路板通常配置于一个盒子内。（一块坏了的386芯片常放在有机玻璃盒内并作为Intel公司雇员的一个纪念品）。该盒子就是计算机，它还包括电源、磁盘驱动器以及连接到显示屏幕、键盘、打印机以及其它计算机的电缆连接器。当计算机接通电源时，386开始运行一个管理计算机的所有主要功能的程序，这个程序就是我们通常所说的操作系统。在计算机运行的整个过程中，操作系统保留在计算机内存的某个部分，在计算机的管理中准备为任何其它程序提供服务。

在继续讨论之前，让我们粗略地划一条贯穿386处理器的线，把它分成两半。靠近线的一半是对所有运行于386上的程序可见的控制功能、寄存器和标志。远离线的一半是仅对操作系统可见的控制功能、寄存器和标志。如果操作系统设计得好，你无须考虑386处理器的那远的一半，除非你非要修改操作系统不可。我们在第五章和第六章讨论处理器的第二部分（远离的部分），在这之前，我们假定386处理器的第一部分（近的部分）就是整个处理器。

我们可以把386的体系结构概括的描述为：386有8个通用寄存器，每个寄存器都是32位。这些寄存器可以用于存放中间结果，也可以用于作为在某一特定存储器段内定位信息的指针和索引寄存器。在32位通用寄存器中还包含有8个16位寄存器和8个8位寄存器。386上还有一个32位指令指针和10个标志。这些标志用于记录处理器的状态和控制处理器的动作。386大约可以访问340亿位的内存和65,000输入／输出端口。本章的前半部分详尽地阐述这些特性。

典型的计算机指令包括：确定指定操作数的位置（待处理的数据），完成对这些操作数的值的一次操作，以及将运算结果存放回指定的结果单元。操作数和结果的存放位置可以是在存储器中，也可以是在寄存器中，这由指令来指定。用于指定这些位置的方法称为计算机的操作数的寻址方式。386的操作数寻址方式在本章的后半部分描述。对指定操作数操作的实际指令在后面的章节中描述。

2.2 存储器结构

在386系统中的存储器是一个最多为 2^{32}（大约43亿）个称为字节的8位量序列。每

一字节都赋予一个唯一地址（无符号数），地址范围为 $0 \sim 2^{32}-1$（十六进制为00000000～FFFFFFFF）。如图2·1所示。

存储器中任意两个连续字节定义为一个字。字中的每个字节都有一个字节地址，这两个字节地址的更小的一个用于作为这个字的地址。字的示例如图2.2所示。

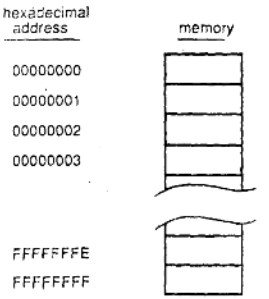


图2.1 存储器地址

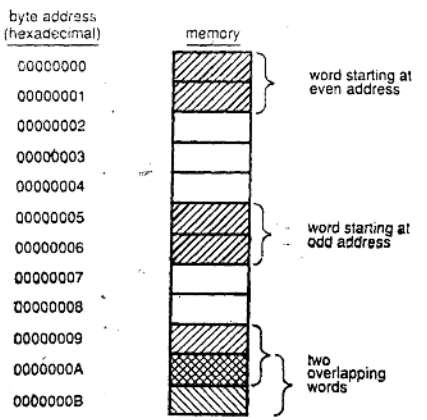


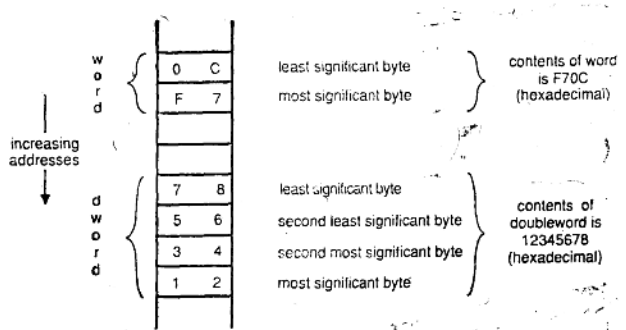
Figure 2.2 Examples of words in memory.

图2.2 内存中的字

一个字含有16位。具有更高内存地址的字节含有这个字的8个最高有效位，具有低的

存储器地址的字节含有字的8个最低有效位。首次阅读时，这似乎是十分自然的，最高有效字节当然应该有高的内存地址。但是，当你考虑到内存是从最低地址开始并向最高地址扩展的字节序列时，很显然386是由前向后存放字的。

最后一点是，内存中任意4个连续字节定义为一个双字。一个双字含有32位。在双字中，字节也是由前向后存放的：具有最高地址的字节含有最高有效位，具有最低地址的字节含有最低有效位。字和双字的由前向后存储如图2·3所示。



igure 2.3 Example of "backwords" storage in memory.

图2.3 内存中由前向后存储示例

2.3 输入／输出结构

将386系统连接到其它部件或设备的事物叫做端口。386正是通过这些端口接收有关外部事件的信息并向外发送控制其它事件的信号。

386能访问多达2^{16}（大约65,000）个8位端口，这些端口与内存字节相似。每个8位端口都赋予一个范围在$0 \sim 2^{16}-1$内的唯一地址。任意两个连续的8位端口都可以作为一个与内存字相似的16位端口对待；任意四个连续的8位端口都可以相似地作为一个32位端口对待。

2.4 寄存器结构

386处理器含有9个32位的寄存器和10个1位的标志。这9个寄存器中有8个是通用寄存器。它们都可以互换地用于80386的大多数算术、逻辑和内存索引功能。第九个寄存器是指令指示器，它不是程序员可直接访问的。386寄存器和标志如图2·4所示。

寄存器和算术运算：在没有通用寄存器的处理器中，每条指令都要从内存中提取操作数并将结果送回内存中。但是内存访问需要时间。这个时间可以通过将接着要使用的操作数和结果暂时保存在一个可以快速存取的地方来减少。386处理器中的通用寄存器组就是这样的快速存取的地方。

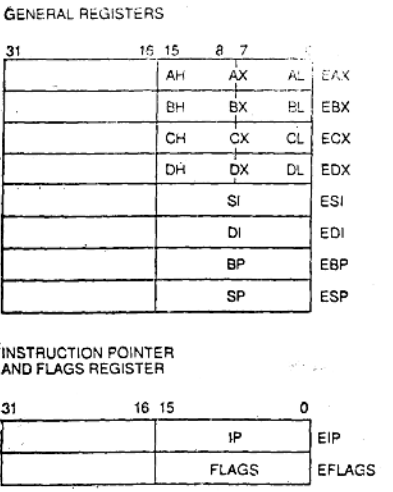


Figure 2.4 80386 basic registers and flags.

图2.4 80386基本寄存器和标志

386的通用寄存器是32位寄存器EAX、EBX、ECX、EDX、ESI、EDI、ESP和EBP。每个寄存器名开头的E代表扩展的（Extended），因为每个32位寄存器都是16位寄存器的扩展，这个16位寄存器是对应的32位寄存器的低半部分（低有效的16位）。这些16位寄存器分别叫做AX、BX、CX、DX、SI、DI、SP和BP。注意这些16位寄存器都是386的先祖86和286中寄存器。前4个16位寄存器又进一步划分成8位寄存器。这时低半部分和高半部分都使用了。低有效部分命名为AL、BL、CL和DL，高有效部分命名为AH、BH、CH和DH。386寻址32位通用寄存器中的8位和16位子集的能力允许386同样容易地处理字节、字和双字量。

对于大多数的部分，通用寄存器的内容可以互换地参与386的算术和逻辑运算。例如，ADD指令可以将任意的8位、16位或32位通用寄存器的内容加到任意的另一具有相同尺寸的通用寄存器中，并将结果存放到其中的某一个寄存器中。但是，有几条指令专门使用某些通用寄存器。例如，串指令要求ECX寄存器含有串中元素个数的计数。任何其它的寄存器都不能用于这个目的。ECX的这种专用性导致了该寄存器的描述名COUNT，EAX、EBX和EDX寄存器的专用性（在后面描述）导致了描述名ACCU-MULATOR、BASE、和DATA。

通用寄存器的这些专门用途有使处理器更难于学习的缺点，因为要记住更多的专门规则。而且很显然程序会是更长，因为在执行某些指令前需要将数据从一个通用寄存器移到

另一个通用寄存器。但是，让我们考虑一下我们怎样为一个将所有通用寄存器都看为相同寄存器的处理器编写程序。为了记录事物的所在地，我们可能这样地组织程序，使得特定种类的数据总是保持在特定的寄存器中。我们或许选择总是使用ECX寄存器来记录串中的元素个数。我们无需将串的大小移入ECX，它会总是在那儿。但是，因为在我们的假想的处理器中串指令可以从任何通用寄存器中得到串的大小，因此每条串指令必须指明在哪儿找到它的串大小。这样就使得每条串指令更长（2字节而不是1字节）或者有更多的一字节指令。第一个结论对使程序更长有直接的影响。第二个结论也使程序变得更长，因为仅有少量的单字节指令（精确地说256个），有更多的单字节串指令意味着某些其它的单字节指令必须使用两字节代之。因此，因为386对某些指令有专用寄存器，因此386体系结构实际上导致了程序长度的减小。

寄存器和存储器寻址：访问存储器中某一单元的指令可以直接指明该单元的地址。这个地址占据了指令的空间，因此增加了代码的长度。如果频繁使用的单元的地址可以存放于寄存器中，访问这些单元的指令就不再需要含有地址，而只需指明含有这些地址的寄存器即可。

寄存器的这个用途与缩略的电话拨号差不多。你可以通过拨你的朋友的7位电话号话给你的朋友去电话。如果你的电话公司提供这个服务，你可以将某些频繁使用的电话号码输入到一组“寄存器”中。然后你可以只拨说明这些寄存器的某一位或两位数字就可给这些选定的人打电话。

寄存器用于存储器寻址除了减少指令的长度外还有另一个（或许是更重要的）功能，它允许指令访问地址为程序运行中前面所执行的运算的结果的单元。为了建立变量的位置常常需要这样的计算，特别是在高级语言程序中。386通过为存储器索引和算术运算提供相同的通用寄存器组实现这种机制。不需要将一个计算所得的地址从一个算术运算寄存器移到一个索引寄存器，因为所有的通用寄存器都有算术运算和索引能力。

最后四个通用寄存器ESI、EDI、EBP和ESP主要是要用于存储器寻址。这些寄存器中有些区别，它们被划分成指针寄存器ESP和EBP及索引寄存器ESI和EDI。指针寄存器提供对堆栈数据的方便存取，而不是用于访问一般数据区。堆栈作为辅助数据区使用对实现高级语言有一些的优点（这些优点要在本章的尾部讨论）。

有区分这两个指针寄存器ESP和EBP的指令。PUSH和POP指令从ESP寄存器获得堆栈顶位置的地址，因此为这个寄存器建议描述名为STACK POINTER。ENTER和LEAVE指令将EBP设立为堆栈数据区的“基”，因此为EBP建议描述名为BASE POINTER。

最后，串指令区别两个索引寄存器ESI和EDI。要求原操作数的那些串指令从ESI获得原操作数的地址；类似地，EDI含有目的操作数的地址。这样分别为这两个寄存器建议描述名SOURCE INDEX和DESTINATION INDEX。对于那些串指令，ESI和EDI的作用是不可互换的。例如，串移动指令将由ESI指向的串移到由EDI指定的位置，串移动指令不显式地提及ESI和EDI寄存器。

标志：386有10个标志，它们用于记录状态信息（状态标志）或用于控制处理器的运行（控制标志）。状态标志通常在算术或逻辑指令的执行后设置，以反映这些运算的结果

的某些属性。这些标志是：进位标志（CF），指示指令是否产生从最高有效位的进位；辅助进位标志（AF），指示指令是否产生从4个最低有效位的进位；溢出标志（OF），指示指令是否产生越界的带符号结果；零标志（ZF），指示指令是否产生一个零结果；符号标志（SF），指示指令是否产生一个负结果；以及奇偶标志，指示指令是否产生一个具有偶数个“1”的结果。

控制标志是：方向标志(DF)，控制寻址操作指令的方向；中断使能标志（IF），允许或禁止外部中断；自陷标志（TF），使处理器进入程序调试的单步方式；以及重执标志（RF），为下一条指令抑止处理器的内部调试器断点。

在整个第三章中给出每个标志的细节，并在第三章的最后一节概述标志的动作。

2.5 指令操作数和操作数寻址方式

386中的指令通常执行对一个或两个操作数的运算。例如，ADD指令将含在一个操作数中的值加到含在第二个操作数的值上，并将结果放回到其中的一个操作数中。INC指令将含在操作数中的值增1并将结果放回操作数中。386允许多种由指令使用的操作数类型。现在该指出一条指令怎样说明它的操作数以及它能够指明什么类型的操作数。这更正规地称之为处理器的操作数寻址方式。

在我们进一步讨论之前，让我们认识一下对任何给定的处理器的指令系统都有两种观点。第一种观点是为处理器编写汇编语言程序的人——程序员的观点。第二种观点是处理器本身——机器的观点。程序员将指令系统看成是一组不同的汇编源程序行。机器将指令系统看成是一组不同位编码（操作码）。对于许多处理器，这两种观点几乎是相同的：每个操作码有一个不同的汇编语言助记符，每一种操作数寻址方式在一个操作数位域中有一个直接的编码。对于386，这两种观点是非常不同的。首先，对于不同的操作码386汇编语言使用相同的指令助记符。例如，MOV助记符至少有17种操作码。其次，386对操作数的位编码比汇编语言指定这些操作码的规则要复杂得多。

在本书中，我们注意力集中在程序员的观点，因为它使得386看起来更简单。我们讨

表2.1 操作数寻址方式

立即数寻址
寄存器寻址
直接存储器寻址
间接存储器寻址
基址
（放大的）索引
基址+（放大的）索引
基址+位移
（放大的）索引+位移
基址+（放大的）索引+位移