第3章 8086/8088微处理器及其系统

学习目标

Intel系列CPU一直占据主导地位。尽管8086/8088后续的80286、80386、80486以及Pentium系列CPU结构与功能已经发生很大变化，但从基本概念与结构以及指令格式上来讲，它们仍然是经典的8086/8088 CPU的延续与提升。并且，其他系列流行的CPU(如AMD公司的6x86MX/MⅡ等)也与80x86 CPU兼容。因此，本章是80x86～Pentium系列微处理器的重要基础，也是整个课程的重点之一。

为了理解微处理器的工作原理和技术特征，本章在解析Intel 8086/8088微处理器的基础上，对8086/8088的存储器的组织、分段管理和物理地址与逻辑地址之间的相互关系及其变换等基本技术均做了详细的讨论，并着重介绍了8086/8088微处理器、系统组成及指令系统。

学习要求

- 8086/8088 CPU的内部组成结构是Intel 80x86系列微处理器体系结构的基础。对8086/8088的寄存器结构与总线周期等应透彻地理解和熟练掌握。
- 深入理解存储器的分段设计是一个关键技术。
- 正确理解与熟练掌握物理地址和逻辑地址的关系，它是存储器管理与地址变换的基础。
- 理解堆栈深度及操作。
- 理解“段加偏移”寻址机制为什么允许重定位。
- 掌握寻址方式。在掌握各种寻址方式的基础上，着重掌握存储器寻址的各种寻址方式。
- 掌握6大类指令系统的基本用法。其中，要求熟练地掌握4类数据传送指令。难点是XLAT、IN、OUT。算术运算类的难点是带符号乘、除指令与十进制指令。对逻辑运算和移位循环类指令要着重理解CL的设置以及进位位的处理。串操作类指令要着重理解重复前缀(REP)的使用。程序控制类指令要着重理解条件转移的条件及测试条件。

3.1 8086/8088 微处理器

8086 是 Intel 系列的 16 位微处理器。它有 16 条数据线，20 条地址线。8088 是准 16 位微处理器。8088 的内部寄存器、运算器以及内部数据总线都是 16 位，只是其外部数据总线为 8 位。

3.1.1 8086/8088 CPU 的内部结构

8086/8088 CPU 的内部功能结构可分为两个独立的部分：总线接口单元(bus interface unit，BIU)；执行单元(execution unit，EU)。

总线接口单元(BIU)的功能是负责完成 CPU 与存储器或 I/O 端口之间的信息传送，即负责从内存预取指令送到指令队列缓冲器；在 CPU 执行指令时，BIU 要配合执行单元(EU)对指定的内存单元或者 I/O 端口存取数据。

EU 的功能只是负责执行指令；执行的指令从 BIU 的指令队列缓冲器中取得，执行指令的结果或执行指令所需要的数据，都由 EU 向 BIU 发出请求，再由 BIU 对存储器或 I/O 端口进行存取。

BIU 和 EU 是相互配合工作的，其操作原则如下：

① 取指令时，每当指令队列缓冲器中存满 1 条指令后，EU 就立即开始执行。

② 指令队列缓冲器中只要空出两个(对 8086)或者空出一个(对 8088)字节时，BIU 就会自动执行取指令操作，直到填满指令队列缓冲器为止。

③ 在 EU 执行指令的过程中，如指令需要对存储器或 I/O 端口存取数据时，则 BIU 会在执行完现行取指令周期后的下一个存储器周期，对指定的内存单元或 I/O 端口进行存取操作，交换的数据经 BIU 由 EU 进行处理。

④ 当 EU 执行完转移、调用和返回指令时，则要清除指令队列缓冲器中按原序列存放的指令，并要求 BIU 从新的地址重新开始取指令，新取的第 1 条指令将直接经指令队列送到 EU 去执行，随后取来的指令将填入指令队列缓冲器。

由于 BIU 与 EU 分开并独立工作，所以 CPU 的指令预取与指令执行是并行重叠操作的。这是流水线操作设计思想的最初成功范例，它被广泛地用于后来各高档 CPU 的设计中。

3.1.2 8086/8088 的寄存器结构

8086/8088 的内部共有 14 个 16 位寄存器。寄存器按功能可分为 3 类：通用寄存器、段寄存器和标志寄存器。

1. 通用寄存器

8086/8088 的通用寄存器分为两组：数据寄存器以及指针寄存器和变址寄存器。

1) 数据寄存器

EU 中有 4 个 16 位数据寄存器 AX、BX、CX 和 DX。每个数据寄存器分为高字节 H 和低字节 L，它们均可作为 8 位数据寄存器独立寻址，独立使用。在多数情况下，这些数据寄存器是用在算术运算或逻辑运算指令中，用来进行算术逻辑运算。在有些指令中，它们则有特定的用途。

2) 指针寄存器和变址寄存器

指针寄存器是指堆栈指针寄存器(SP)和堆栈基址指针寄存器(BP)，简称为 P 组。变址寄存器是指源变址寄存器(SI)和目的变址寄存器(DI)，简称为 I 组。它们都是 16 位寄存器，一般用来存放地址的偏移量(简称为偏置)。

指针寄存器 SP 和 BP 都用来指示存取位于当前堆栈段中的数据所在的地址，但 SP 和 BP 在使用上有区别。入栈(PUSH)和出栈(POP)指令是由 SP 给出栈顶的偏移地址，故 SP 称为堆栈指针寄存器(简称堆栈指针)。而 BP 则是存放位于堆栈段中一个数据区基地址的偏移地址，故称为堆栈基址指针寄存器(简称基址指针)。两者意思不同，不可混淆。

2. 段寄存器

段寄存器是为实现“段加偏移”寻址机制而设置的，熟练应用非常重要。

8086/8088 CPU 内设置了 4 个 16 位段寄存器，用这些段寄存器的内容作为 16 位的段地址，再由段寄存器左移 4 位形成 20 位的段起始地址，这样就有可能寻址 1MB 存储空间并将其分成若干逻辑段，使每个逻辑段的长度为 64KB(它由 16 位的偏移地址限定)。要着重指出的是，这些逻辑段可以通过修改段寄存器的内容被任意设置在整个 1MB 存储空间上下浮动；逻辑段在存储器中定位以前，还不是可以真正寻址的实际内存地址，所以，通常人们就将尚未定位之前在程序中出现的地址称为逻辑地址。这个概念对于“重定位”十分有用。

8086/8088 的指令能直接访问这 4 个段寄存器，其中，代码段寄存器(CS)用来存放程序当前使用的代码段的段地址，CPU 执行的指令将从代码段取得；堆栈段寄存器(SS)用来存放程序当前所使用的堆栈段的段地址，堆栈操作的数据就在这个段中；数据段寄存器(DS)用来存放程序当前使用的数据段的段地址，一般来说，程序所用的数据就存放在数据段中；附加段寄存器(ES)用来存放程序当前使用的附加段的段地址，它通常也用来存放数据，但典型用法是用来存放处理以后的数据。

3. 标志寄存器

8086/8088 的 16 位标志寄存器 FLAGS 只用了其中的 9 位作标志位，即 6 个状态标志位、3 个控制标志位。

状态标志位用来反映算术或逻辑运算结果的状态，以记录 CPU 的状态特征。这 6 位

是：CF(carry flag)进位标志；PF(parity flag)奇偶性标志；AF(auxiliary carry flag)辅助进位标志；ZF(zero flag)零标志；SF(sign flag)符号标志；OF(overflow flag)溢出标志：溢出标志用于判断在有符号数进行加法或减法时是否可能出现溢出。

控制标志有3个：DF(direction flag)方向标志；IF(interrupt enable flag)中断允许标志，它是控制可屏蔽中断的标志；TF(trap flag)跟踪(陷阱)标志。

需要指出的是，8086/8088所有上述标志位对Intel系列后续高型号微处理器的标志寄存器都是向上兼容的。

3.1.3 总线周期

一个最基本的总线周期由4个时钟周期组成，习惯上将4个时钟周期分别称为4个状态，即 T_1、T_2、T_3 与 T_4 这4个状态。

(1) 在 T_1 状态，CPU往多路复用总线上发送地址信息，以选中所要寻址的存储单元或外设端口的地址。

(2) 在 T_2 状态，CPU从总线上撤销地址，并使总线的低16位浮置成高阻状态，为传送数据做准备。

(3) 在 T_3 状态，多路总线的高4位继续提供状态信息，而其低16位(对8088 CPU则为低8位)上将出现由CPU写出的数据或者CPU从存储器或端口读入的数据。

(4) 在有些情况下，由于外设或存储器的速度较慢，不能及时地配合CPU传送数据。这时，外设或存储器就会通过READY的信号线在 T_3 状态启动前向CPU发一个“数据未准备好信号”，表示它们还来不及同CPU之间传送数据，于是，CPU会在 T_3 之后自动插入一个或多个附加的时钟周期 T_W，这个 T_W 就称为等待状态，它表示此时CPU在总线上的信息情况和 T_3 状态时的信息情况一样。只有在指定的存储器或外设已经完成数据传送时，它们又通过READY的信号线向CPU发出一个“准备好”信号，当CPU接收到这一信号后，才会自动脱离 T_3 状态而进入 T_4 状态。

(5) 在 T_4 状态，CPU完成本次读或写操作，总线周期结束。

3.1.4 8086/8088的引脚信号和功能

1. 地址/数据总线

地址/数据总线 AD_{15}～AD_0 是分时复用的存储器或端口的地址和数据总线。传送地址时为单向的三态输出，而传送数据时可双向三态输入输出。正是利用分时复用的方法才能使8086/8088用40条引脚实现20位地址、16位数据及众多的控制信号和状态信号的传输。

2. 地址/状态总线

地址/状态总线 A_{19}/S_6～A_{16}/S_3 为输出、三态总线，采用分时输出，即 T_1 状态输出地

址的最高 4 位，T_2～T_4 状态输出状态信息。当访问存储器时，T_1 状态时输出的 A_{19}～A_{16}送到锁存器(8282)锁存，与 AD_{15}～AD_0 一起组成 20 位的地址信号；而访问 I/O 端口时，不使用这 4 条引线，A_{19}～A_{16}＝0。

3. 控制总线

下面分别介绍各控制信号线。

(1) $\overline{BHE}/S_7$：高 8 位数据总线允许/状态复用引脚，三态、输出。$\overline{BHE}$在总线周期的 T_1 状态时输出，S_7 在 T_2～T_4 时输出。在 8086 中，当$\overline{BHE}/S_7$ 引脚上输出$\overline{BHE}$信号时，表示总线高 8 位 AD_{15}～AD_8 上的数据有效。在 8088 中，第 34 引脚不是$\overline{BHE}/S_7$，而是被赋予另外的信号：在最小模式时，它为$\overline{SS_0}$，和 DT/$\overline{R}$、$\overline{M}$/IO 一起决定了 8088 当前总线周期的读/写动作；在最大模式时，它恒为高电平。

(2) $\overline{RD}$：读控制信号，三态、输出。当$\overline{RD}$＝0 时，表示 CPU 将要执行一个对存储器或 I/O 端口的读操作。到底是对内存单元还是对 I/O 端口读取数据，取决于 M/$\overline{IO}$(8086)或 $\overline{M}$/IO(8088)信号。在一个读操作的总线周期中，$\overline{RD}$信号在 T_2、T_3 和 T_W 状态均为低电平，以保证 CPU 读有效。

(3) READY："准备好"信号，输入。它实际上是由所寻址的存储器或 I/O 端口发来的响应信号，高电平有效。当 READY＝1 时，表示所寻址的内存或 I/O 设备已准备就绪，马上就可进行一次数据传输。CPU 在每个总线周期的 T_3 状态开始对 READY 信号采样。如果检测到 READY 为低电平，表示存储器或 I/O 设备尚未准备就绪，则 CPU 在 T_3 状态之后自动插入一个或几个等待状态 T_W，直到 READY 变为高电平，才进入 T_4 状态，完成数据传送过程，从而结束当前总线周期。

(4) $\overline{TEST}$：等待测试信号，输入。它用于多处理器系统中且只有在执行 WAIT 指令时才使用。

(5) INTR：可屏蔽中断请求信号，输入，高电平有效。当 INTR＝1 时，表示外设提出了中断请求，8086/8088 在每个指令周期的最后一个 T 状态去采样此信号。若 IF＝1，则 CPU 响应中断，停止执行当前的指令序列，并转去执行中断服务程序。

(6) NMI：非屏蔽中断请求信号，输入，上升沿触发。此请求不受 IF 状态的影响，也不能用软件屏蔽，只要此信号一出现，就在现行指令结束后引起中断。

(7) RESET：复位信号，输入，高电平有效。它通常与 8284A(时钟发生/驱动器)的复位输出端相连，8086/8088 要求复位脉冲宽度不得小于 4 个时钟周期，而初次接通电源时所引起的复位，则要求维持的高电平不能小于 50μs；复位后，CPU 的主程序流程恢复到启动时的循环待命初始状态。

(8) CLK：系统时钟，输入。通常与 8284A 时钟发生器的时钟输出端 CLK 相连。

4. 电源线和地线

电源线 V_{CC}接入的电压为＋5V±10%，有两条地线 GND，均应接地。

5. 其他控制线

这些控制线(24～31 引脚)的功能将根据系统操作的模式控制线 MN/$\overline{\text{MX}}$所处的状态而确定。

3.2 8086/8088 系统的最小/最大工作方式

由 8086/8088 CPU 构成的微机系统,有最小方式和最大方式两种系统配置。

3.2.1 最小方式

当 MN/$\overline{\text{MX}}$接电源电压时,系统工作于最小方式,即单处理器系统方式,它适合于较小规模的应用。

在最小方式下,有关引脚信号的基本含义如下。

1. $\overline{\text{INTA}}$(interrupt acknowledge)中断响应信号输出

$\overline{\text{INTA}}$用于 CPU 对来自外设的中断请求做出响应。8086/8088 的$\overline{\text{INTA}}$信号实际上是两个连续的负脉冲,其第 1 个负脉冲是通知外设接口,它发出的中断请求已获允许;外设接口收到第 2 个负脉冲后,就往数据总线上发送一个中断类型码。

2. ALE(address latch enable)地址锁存信号输出

ALE 是 8086/8088 提供给地址锁存器 8282/8283 的控制信号,高电平有效。在任何一个总线周期的 T_1 状态,ALE 输出有效电平,以表示当前在地址/数据复用总线上输出的是地址信息,地址锁存器将 ALE 作为锁存信号,对地址进行锁存。

3. $\overline{\text{DEN}}$(data enable)数据允许信号

当用 8286/8287 作为数据总线收发器时,8086 CPU 的$\overline{\text{DEN}}$为收发器的$\overline{\text{OE}}$端提供一个控制信号,该信号决定是否允许数据通过数据总线收发器。

4. DT/$\overline{\text{R}}$(data transmit/receive)数据收发输出

在使用 8286/8287 作为数据总线收发器时,CPU 的 DT/$\overline{\text{R}}$信号用来控制 8286/8287 的数据传送方向。

5. M/$\overline{\text{IO}}$(memory/input and output)存储器/输入输出控制信号输出

M/$\overline{\text{IO}}$是作为区分 CPU 当前是进行存储器访问还是输入输出访问的控制信号。如果是高电平,则表示 CPU 是在和存储器之间进行数据传输;如果是低电平,则表示 CPU

是在和输入输出设备之间进行数据传输。

6. $\overline{WR}$(write)写信号输出

$\overline{WR}$有效时,表示 CPU 当前正在进行存储器或 I/O 写操作,到底是哪种写操作,则由 M/$\overline{IO}$信号决定。对任何写周期,$\overline{WR}$在 T_2、T_3、T_W 期间都有效。

7. HOLD(hold request)总线保持请求信号输入

HOLD 作为系统中由其他处理部件向 CPU 发出总线请求信号的输入引脚端。当系统中 CPU 之外的另一个处理主模块要求占用总线时,就要通过 HOLD 引脚向 CPU 发送一个高电平的请求信号。

8. HLDA(hold acknowledge)总线保持响应信号输出

当 HLDA 为有效电平时,表示 CPU 对其他主模块的总线请求正处于响应的状态,与此同时,所有与三态门相接的 CPU 的引脚都呈现高阻抗,从而让出了总线。

3.2.2 最大方式

当 MN/$\overline{MX}$线接地,则系统工作于最大方式。最大方式系统与最小方式系统的主要区别是外加有 8288 总线控制器,通过它对 CPU 发出的控制信号进行变换和组合,以得到对存储器和 I/O 端口的读/写信号和对锁存器 8282 及总线收发器 8286 的控制信号,使总线控制功能更加完善。通常,在最大方式系统中,包含两个或多个处理器,这样就要解决主处理器和协处理器之间的协调工作问题以及对总线的共享控制问题,为此,要从软件和硬件两方面去解决。8288 总线控制器就是因此需要而加在最大方式系统中的。

比较两种工作方式可以知道,在最小方式系统中,控制信号 M/$\overline{IO}$(或$\overline{M}$/IO)、$\overline{WR}$、$\overline{INTA}$、ALE、DT/$\overline{R}$和$\overline{DEN}$是直接从 CPU 第 24～29 脚送出的;而在最大方式系统中,状态信号$\overline{S}_2$、$\overline{S}_1$、$\overline{S}_0$ 隐含了上面这些信息,使用 8288 后,系统就可以从$\overline{S}_2$、$\overline{S}_1$、$\overline{S}_0$ 状态信息的组合中得到与这些控制信号功能相同的信息。

此外,还有几个在最大方式下使用的专用引脚,其含义简要解释如下。

1. QS_1、QS_0(instruction queue status)指令队列状态信号输出

QS_1 和 QS_0 两个信号组合起来提供了本总线周期的前一个时钟周期中指令队列的状态,以便外部对 8086/8088 内部指令队列的动作进行跟踪。

2. $\overline{LOCK}$(lock)总线封锁信号输出

当$\overline{LOCK}$为低电平时,系统中其他总线主部件就不能占有总线。

3. $\overline{RQ}/\overline{GT_1}$、$\overline{RQ}/\overline{GT_0}$(request/grant)总线请求信号输入/总线请求允许信号输出

$\overline{RQ}/\overline{GT_1}$和$\overline{RQ}/\overline{GT_0}$两个信号端可供CPU以外的两个处理器用来发出使用总线的请求信号和接收CPU对总线请求信号的回答信号。$\overline{RQ}/\overline{GT_1}$和$\overline{RQ}/\overline{GT_0}$都是双向的,总线请求信号和允许信号在同一引线上传输,但方向相反。其中,$\overline{RQ}/\overline{GT_0}$比$\overline{RQ}/\overline{GT_1}$的优先级要高。

在8288芯片上,还有几条控制信号线$\overline{MRDC}$(memory read command)、$\overline{MWTC}$(memory write command)、$\overline{IORC}$(I/O read command)、$\overline{IOWC}$(I/O write command)与$\overline{INTA}$等,它们分别是存储器与I/O的读/写命令以及中断响应信号。另外,还有$\overline{AMWC}$与$\overline{AIOWC}$两个输出信号,它们分别表示提前的写内存命令与提前的写I/O命令,其功能分别和$\overline{MWTC}$与$\overline{IOWC}$一样,只是它们由8288提前一个时钟周期发出信号,这样,一些较慢的存储器和外设将得到一个额外的时钟周期去执行写入操作。

3.3 8086/8088的存储器

3.3.1 存储器组织

8086/8088有20条地址线,可寻址1MB的存储空间。存储器仍按字节组织,每个字节只有唯一的一个地址。若存放的信息是8位的字节,将按顺序存放;若存放的数为一个字,则将字的低位字节放在低地址中,高位字节放在高地址中;若存放的是双字形式(这种数一般作为指针),其低位字是被寻址地址的偏移量,高位字是被寻址地址所在的段地址。

对存放的字,其低位字节可以在奇数地址中开始存放,也可以在偶数地址中开始存放;前者称为非规则存放,这样存放的字称为非规则字;后者称为规则存放,这样存放的字称为规则字。对规则字的存取可在一个总线周期完成,非规则字的存取则需两个总线周期。

在8086/8088程序中,指令格式仅要求指出对某个字节或字进行访问,而对存储器访问的方式不必说明,无论执行哪种访问,都是由处理器自动识别的。

8086的1MB存储空间实际上分为两个512KB的存储体,又称存储库,分别称为高位库和低位库。低位库与数据总线$D_7 \sim D_0$相连,该库中每个地址为偶数地址;高位库与数据总线$D_{15} \sim D_8$相连,该库中每个地址为奇数地址。地址总线$A_{19} \sim A_1$可同时对高、低位库的存储单元寻址,A_0或$\overline{BHE}$则用于库的选择,分别接到库选择端$\overline{SEL}$上。当$A_0=0$时,选择偶数地址的低位库;当$\overline{BHE}=0$时,选择奇数地址的高位库。利用A_0或$\overline{BHE}$这两个控制信号可以实现对两个库进行读/写(即16位数据),也可单独对其中的一个库进行读/写。

在8088系统中,可直接寻址的存储空间同样也为1MB,但其存储器的结构与8086

有所不同，它的 1MB 存储空间同属一个单一的存储体，即存储体为 1M×8 位。它与总线之间的连接方式很简单，其 20 根地址线 A_{19}～A_0 与 8 条数据线分别同 8088 CPU 的对应地址线与数据线相连。8088 CPU 每访问 1 次存储器只读/写 1 字节信息，因此，在 8088 系统的存储器中不存在对准存放的概念，任何数据字都需要两次访问存储器才能完成读/写操作，故在 8088 系统中，程序运行速度比在 8086 系统中要慢些。

3.3.2 存储器的分段

存储器的分段是一个重要的概念，深入理解存储器的分段设计是掌握存储器管理技术的一个关键。

在 8 位微处理器中，由于存储器的存储空间受 16 条地址线的限制，它只有 64KB 的很小地址空间，所以存储器是没有分段的。而在 8086/8088 微处理器中，它有 20 条地址线，所以它允许寻址 1MB 大小的存储空间。但由于 8086/8088 CPU 的指令指针 IP 和堆栈指针 SP 都是 16 位，其直接寻址的最大空间只有 64KB。因此，为了能寻址 1MB 存储空间，就要对存储器实行分段，使每段的最大寻址空间为 64KB，16 个段的总寻址空间才可达到 1MB。

在 8086/8088 系统中，具体的分段设计为：将 1MB 存储空间分为若干逻辑段，每段的大小可以从 1 字节开始任意递增，直至最多可包含 64KB 长的连续存储单元。每个段的 20 位起始地址（即段基址）是一个能被 16 整除的数，即最后 4 位为 0。段和段之间可以是连续的、分开的、部分重叠的或完全重叠的。当然，每个存储单元的内容不允许重叠，否则会导致冲突。一个程序所用的具体存储空间可以为一个逻辑段，也可以为多个逻辑段。

存储器的分段设计和分段管理是这样实现的：将段地址存放在段寄存器 CS、DS、SS 和 ES 中，所以，程序可以从 4 个段寄存器给出的逻辑段中存取代码和数据。如果要从存储器另外的段而不是当前可寻址的段中存取信息，则程序必须首先改变相应的段寄存器的内容，将其设置成所要存取的新段的段地址，然后才可以从当前可寻址的段转到新段中去继续寻址。

一定要理解，段区的分配工作是由操作系统完成的，而系统允许程序员只有在必要时才能指定所需占用的内存区。

3.3.3 实际地址和逻辑地址

实际地址是指 CPU 在对内存进行访问而实际寻址时所能直接使用的地址。对 8086/8088 来说，实际地址是用 20 位二进制数或 5 位十六进制数表示的地址。通常，实际地址又称为物理地址。

逻辑地址是由程序和指令表示的一种地址，它包括两部分：段地址和偏移地址。对 8086/8088 来说，段地址和偏移地址都用无符号的 16 位二进制数或 4 位十六进制数来表示。

3.3.4 堆栈

8086/8088 系统中的堆栈是用段定义语句在存储器中定义的一个堆栈段，和其他逻辑段一样，它可在 1MB 的存储空间中浮动。系统中具有的堆栈数目不受限制，一个栈的深度最大为 64KB。

堆栈由 SS 和 SP 来寻址。SS 给定堆栈段的段地址，而 SP 给定当前栈顶，即指出从堆栈的段基址到栈顶的偏移量。为了加快堆栈操作的速度，堆栈操作均以字为单位进行。

3.3.5 “段加偏移”寻址机制允许重定位

“段加偏移”寻址机制允许重定位（或再定位）是一种重要的特性。必须理解，重定位是指一个完整的程序块或数据块可以在存储器所允许的空间内任意浮动并定位到一个新的可寻址的区域。

在 8 位微处理器中是没有这种特性的，而从 8086 引入分段概念之后，由于段寄存器中的段地址可以由程序来重新设置，所以，在偏移地址不变的情况下，就可以将整个存储器段移动到存储器的任何区域而不会改变程序块或数据块的结构。由于“段加偏移”的寻址机制允许程序在存储器内重定位，因此，原来为了在实模式下运行所编写的程序，在系统转换为保护模式时也可以运行。

由于“段加偏移”的寻址机制允许程序和数据不需要做任何修改就能使它们重定位，这就给各种通用计算机系统在运行同一软件和数据时能够保持兼容性带来很大的方便。

例如，有一条指令位于距存储器中某段首（即段基地址）8 字节的位置，它的偏移地址就是 8。当整个程序移到新的存储区时，这个偏移地址 8 仍然指向距存储器中新的段首 8 字节的位置。只是这时段寄存器的内容必须重新设置为程序所在的新存储段的起始地址。如果计算机系统没有重定位的特性，当一个程序在移动之前，就必须大范围地重写或更改，或者要为许多不同配置的计算机系统设计许多程序文本，这不仅需要花费大量的时间，还可能会引起程序出错。

3.4 8086/8088 指令系统

3.4.1 指令系统的特点及指令基本格式

8086 与 8088 的指令系统完全相同，它们是由 8 位的 8080/8085 指令系统扩展而来的，同时，它们又能在其后续的 80x86 系列的 CPU 上正确运行。因此，8086/8088 指令系统是 80x86 CPU 共同的基础。其主要特点如下：

（1）采用可变长指令，指令格式比较复杂。

（2）寻址方式多样灵活，处理数据的能力比较强，可处理字节或字、带符号或无符号

的二进制数据以及压缩型/非压缩型的十进制数据。

(3) 有重复指令和乘、除运算指令。扩充了条件转移、移位/循环指令。

(4) 为加强软件中断功能和支持多处理器系统工作,增设了相关的指令。

指令格式是按指令系统的规范与要求精心设计的,了解指令格式有助于深入掌握指令代码的组成原理。指令的基本组成包括两部分:操作码(Op-Code)与操作数(Oprand)。

有关 8086/8088 指令的详细格式可参见主教材的附录 A。

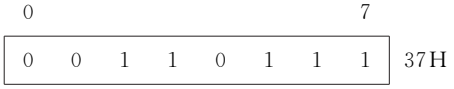
3.4.2 寻址方式

CPU 的寻址方式就是根据指令功能所规定的操作码自动寻找相应的操作数所在地址的方式。8086/8088 的操作数可位于寄存器、存储器或 I/O 端口中。下面简要介绍 8086/8088 的寻址方式。

1. 固定寻址

有些单字节指令其操作是规定 CPU 对某个固定的寄存器进行的,如加法的 ASCII 调整指令 AAA,规定被调整的数总是位于 AL 中。

该指令用来调整 AL 中的结果,此结果是把两个 ASCII 字符当作操作数相加后形成的。其指令编码如下:



2. 立即数寻址

操作数就在指令中,当执行指令时,CPU 直接从指令队列中取得该立即数,而不必执行总线周期。立即数可以是 8 位,也可以是 16 位;并规定只能是整数类型的源操作数。这种寻址主要用来给寄存器赋初值,指令执行速度快。例如:

```
MOV AX,1680H;  将 1680H 送入 AX,AH 中为 16H,AL 中为 80H
```

3. 寄存器寻址

操作数就放在 CPU 的寄存器中,而寄存器名在指令中指出。例如:

```
INC reg
```

表示将指令的寄存器的内容加 1。

对 16 位操作数来说,寄存器可以是 8 个 16 位通用寄存器,而对 8 位操作数来说,寄存器只能是 AH、AL、BH、BL、CH、CL、DH、DL。在一条指令中,源操作数或/和目的操作数都可以采用寄存器寻址方式。

4. 存储器寻址

CPU 寻找存储器操作数，必须经总线控制逻辑电路进行存取。当执行单元 EU 需要读/写位于存储器的操作数时，应根据寻址方式由 EU 先计算出操作数地址的偏移量（即有效地址 EA），它是一个不带符号的 16 位地址码，表示操作数所在段的首地址与操作数地址之间的字节距离。所以，它实际上是一个相对地址。EA 的值由汇编程序根据指令所采用的寻址方式自动计算得出。计算 EA 的通式如下：

EA＝基址值（BX 或 BP）＋变址值（SI 或 DI）＋位移量 D（0 或 8 或 16）

存储器寻址细分为以下几种。

1）直接寻址方式

直接寻址方式是指指令中以位移量方式直接给出操作数的有效地址 EA，即 EA＝DISP。这种寻址方式的指令执行速度快，主要用于存取位于存储器中的简单变量。例如：

```
MOV AX,[1680H]  ;将 1680H 和 1681H 两单元的字内容取入 AX 中
```

2）间接寻址方式

间接寻址方式是指寄存器间接寻址方式，其操作数一定是在存储器中，而存储单元的有效地址 EA 则由寄存器指出，这些寄存器是基址寄存器 BX、基址指针寄存器 BP、变址寄存器 SI 和 DI 之一或它们的某种组合。书写指令时，这些寄存器带有方括号“[ ]”。根据所采用寄存器的不同，间接寻址方式又可分为以下 3 种。

（1）基址寻址方式。

基址寻址是指操作数的有效地址由基址寄存器（BX 或 BP）的内容和指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。

（2）变址寻址方式。

变址寻址是指操作数的有效地址由变址寄存器（SI 或 DI）的内容与指令中给出的地址位移量（0 位、8 位或 16 位）之和来确定。

（3）基址加变址寻址方式。

基址加变址寻址是指操作数的有效地址 EA 由基址寄存器（BX 或 BP）的内容与变址寄存器（SI 或 DI）的内容以及指令中的地址位移量（0 位、8 位或 16 位）三者之和来确定。

注意：由于指令中的位移量也可以看作一个相对值，因此，有时又把带位移量的寄存器间接寻址称为寄存器相对间接寻址。

5. 其他寻址方式

1）串操作指令寻址方式

数据串（或称字符串）指令不能使用正常的存储器寻址方式来存取数据串指令中使用的操作数。执行数据串指令时，源串操作数第 1 个字节或字的有效地址应存放在源变址寄存器 SI 中（不允许修改），目标串操作数第 1 个字节或字的有效地址应存放在目标变址寄存器 DI 中（不允许修改）。在重复串操作时，8086/8088 能自动修改 SI 和 DI 的内容，

以使它们能指向后面的字节或字。因指令中不必给出 SI 或 DI 的编码，故串操作指令采用的是隐含寻址方式。

2）I/O 端口寻址方式

在 8086/8088 指令系统中，输入输出指令对 I/O 端口的寻址可采用直接的或间接的两种方式。

（1）直接端口寻址：I/O 端口地址以 8 位立即数方式在指令中直接给出。例如，IN AL，n。所寻址的端口号只能在 0～255 范围内。

（2）间接端口寻址：这类似于寄存器间接寻址，16 位的 I/O 端口地址在 DX 寄存器中，即通过 DX 间接寻址，故可寻址的端口号为 0～65 535。例如，OUT DX，AL。它是将 AL 的内容输出到由(DX)指出的端口中去。

3）转移类指令的寻址方式

在 8086/8088 系统中，由于存储器采用分段结构，所以转移类指令有段内转移和段间转移之分。所有的条件转移指令只允许实现段内转移，而且是段内短转移，即只允许转移的地址范围在－128～＋127 字节内，由指令中直接给出 8 位地址位移量。

对于无条件转移和调用指令又可分为段内短转移、段内直接转移、段内间接转移、段间直接转移和段间间接转移 5 种不同的寻址方式。有关这类寻址的详细情况，将在转移指令中讨论。

3.4.3 指令的分类

8086/8088 的指令按功能可分为 6 类：数据传送、算术运算、逻辑运算和移位循环、串操作、程序控制和处理器控制。

1. 数据传送类指令

数据传送类指令可完成寄存器与寄存器之间、寄存器与存储器之间以及寄存器与 I/O 端口之间的字节或字传送，它们所具有的共同特点是不影响标志寄存器的内容。

1）通用数据传送指令

（1）MOV d，s。

这条指令将由源 s 指定的源操作数送到目标 d。其中，s 表示源，d 表示目标。由 s 与 d 可分别指定源操作数与目标操作数。源操作数可以是 8/16 位寄存器、存储器中的某个字节/字或者是 8/16 位立即数；目标操作数不允许为立即数，其他同源操作数。并且源操作数和目标操作数不能同时为存储器操作数。

MOV 指令可实现的数据传送类型有以下 7 种。

① MOV mem/reg1，mem/reg2：由 mem/reg2 所指定的存储单元或寄存器中的 8 位数据或 16 位数据传送到由 mem/reg1 所指定的存储单元或寄存器中，但不允许从存储器传送到存储器。

② MOV mem/reg，data：将 8 位或 16 位立即数 data 传送到由 mem/reg 所指定的存储单元或寄存器中。

③ MOV reg,data：将 8 位或 16 位立即数 data 传送到由 reg 所指定的寄存器中。

④ MOV ac,mem：将存储单元中的 8 位或 16 位数据传送到累加器 ac 中。

⑤ MOV mem,ac：将累加器 AL(8 位)或 AX(16 位)中的数据传送到由 mem 所指定的存储单元中。

⑥ MOV mem/reg,segreg：将由 segreg 所指定的段寄存器(CS、DS、SS、ES 之一)的内容传送到由 mem/reg 所指定的存储单元或寄存器中。

⑦ MOV segreg,mem/reg：允许将由 mem/reg 指定的存储单元或寄存器中的 16 位数据传送到由 segreg 所指定的段寄存器中(但代码段寄存器 CS 除外)。

注意：MOV 指令不能直接实现从存储器到存储器之间的数据传送，但可以通过寄存器作为中转站来完成这样的传送。

(2) PUSH 和 POP。

```
PUSH s      ;将源操作数(16 位)压入堆栈
POP d       ;将堆栈中当前栈顶两相邻单元的数据字弹出到 d
```

这两条指令分别是进栈指令与出栈指令。其中，s 和 d 可以是 16 位寄存器或存储器的两相邻单元，以保证堆栈按字操作。例如：

```
PUSH BX
```

设当前 CS=1000H，IP=0030H，SS=2000H，SP=0040H，BX=2340H，则该指令执行时，堆栈指针被修改为 SP−2→SP，使之指向新栈顶 2003EH，同时将 BX 中的数据字 2340H 压入栈内 2003FH 与 2003EH 两单元中。又如：

```
POP CX
```

设当前 CS=1000H，IP=0020H，SS=1600H，SP=004CH，则该指令执行时，将当前栈顶两相邻单元 1604CH 与 1604DH 中的数据字弹出并传送到 CX 中，同时修改堆栈指针，SP+2→SP，使之指向新栈顶 1604EH。

PUSH 和 POP 是两条很重要的指令，它们可用来保存并恢复来自堆栈存储器的数据。例如，在子程序调用或中断处理过程时，分别要保存返回地址或断点地址，在进入子程序或中断处理后，还需要保留通用寄存器的值；而在由子程序返回或由中断处理返回时，则要恢复通用寄存器的值，并分别将返回地址或中断地址恢复到指令指针寄存器中。

(3) XCHG d,s。

数据交换指令 XCHG d,s 的功能是将源操作数与目标操作数(字节或字)相互对应交换位置。

交换可以在通用寄存器与累加器之间、通用寄存器之间、通用寄存器与存储器之间进行。但是，不能在两个存储单元之间交换，段寄存器与 IP 也不能作为源或目标操作数。例如：

```
XCHG AX,[SI+0400H]
```

设当前 CS=1000H，IP=0064H，DS=2000H，SI=3000H，AX=1234H，则在该指令

执行后，将把 AX 寄存器中的 1234H 与物理地址 23400H(＝DS×16＋SI＋0400H＝20000H＋3000H＋0400H)单元开始的数据字(设为 ABCDH)相互交换位置，即 AX＝ABCDH；(23400H)＝34H，(23401H)＝12H。

(4) XLAT。

这是一条用于实现字节翻译功能的指令，也称为代码转换指令。具体地说，它可以将 AL 寄存器中设定的一个字节数值变换为内存一段连续表格中的另一个相应的代码，以实现编码制的转换。

该指令是通过查表方式来完成代码转换功能的。例如，通过查七段显示码表，可将 AL 中任一个设定的十进制数转换为内存表格中某个同该数对应的七段显示码。

设有一个代码转换表(如十进制数 0～9 的七段显示码表)被定位在当前数据段中，其起始地址的偏移地址值为 0030H。假定当前 CS＝2000H，IP＝007AH，DS＝4000H。若欲将 AL 中待转换的十进制数 5 转换成对应的七段码 12H，试分析执行 XLAT 指令的操作过程。

首先，将数据段中该转换表的首地址的偏移地址 0030H 置入 BX；再将待转换的十进制数在表中的序号 05H 送入 AL；然后，执行 XLAT 指令。这时，放在代码段物理地址 2007AH 单元中的指令代码 11010111 即 D7H 被取入 8086 BIU 中的指令队列缓冲器。该指令经 EU 控制电路译码与执行后，将数据段的转换表中物理地址为 DS×16＋BX＋AL＝40035H 单元中的七段码 12H 传送至 AL。于是，完成代码转换过程。

假设 0～9 的七段显示码表存放在偏移地址为 0030H 开始的内存中，则取出 5 所对应的七段码可以用如下 3 条指令助记符完成。

```
MOV BX,0030H
MOV AL,5
XLAT
```

2) 目标地址传送指令

这类指令专用于 8086/8088 中传送地址码，可传送存储器的逻辑地址(即存储器操作数的段地址或偏移地址)至指定寄存器中。

(1) LEA d,s。

这是取有效地址指令，其功能是把用于指定源操作数(它必须是存储器操作数)的 16 位偏移地址(即有效地址)，传送到一个指定的 16 位通用寄存器中。例如：

```
LEA BX,[SI+100AH]
```

设当前 CS＝1500H，IP＝0200H，DS＝2000H，SI＝0030H，源操作数 1234H 存放在[SI＋100AH]开始的存储器内存单元中，则该指令执行的结果是将源操作数 1234H 的有效地址 103AH 传送到 BX 寄存器中。

请注意比较 LEA 指令和 MOV 指令的不同功能。例如，LEA BX,[SI]指令是将 SI 指示的偏移地址(SI 的内容)装入 BX；而 MOV BX,[SI]指令则是将由 SI 寻址的存储单元中的数据装入 BX。

(2) LDS d,s。

这是取某变量的32位地址指针的指令,其功能是从由指令的源s所指定的存储单元开始,由4个连续存储单元中取出某变量的地址指针(共4字节),将其前两个字节(即变量的偏移地址)传送到由指令的目标d所指定的某16位通用寄存器,后两个字节(即变量的段地址)传送到DS段寄存器中。例如:

```
LDS SI,[DI+100AH]
```

设当前CS=1000H,IP=0604H,DS=2000H,DI=2400H,待传送的某变量的地址指针其偏移地址为0180H,段地址为2230H,则该指令执行后,将物理地址2340AH单元开始的4字节中前两个字节(偏移地址值)0180H传送到SI寄存器中,后两个字节(段地址)2230H传送到DS段寄存器中,并取代它的原值2000H。

(3) LES d,s。

这条指令与LDS d,s指令的操作基本相同,其区别仅在于将把由源所指定的某变量的地址指针中后两个字节(段地址)传送到ES段寄存器,而不是DS段寄存器。例如:

```
LES DI,[BX]
```

设当前DS=B000H,BX=080AH,B080AH单元指定的存储字为05A2H,B080CH单元指定的存储字为4000H,该指令执行后,则将某变量地址指针的前两个字节(即偏移地址)05A2H装入DI,而将地址指针的后两个字节(即段地址)4000H装入ES,于是,DI=05A2H,ES=4000H。

3) 标志位传送指令

(1) LAHF。

指令功能:将标志寄存器F的低字节(共包含5个状态标志位)传送到AH寄存器中。

LSHF指令执行后,AH的D_7、D_6、D_4、D_2与D_0这5位将分别被设置成SF(符号标志)、ZF(零标志)、AF(辅助进位标志)、PF(奇偶标志)与CF(进位标志)5位,而AH的D_5、D_3、D_1这3位没有意义。

(2) SAHF。

指令功能:将AH寄存器内容传送到标志寄存器F的低字节。

SAHF与LAHF的功能相反,它常用来通过AH对标志寄存器的SF、ZF、AF、PF与CF标志位分别置位或复位。

(3) PUSHF。

指令功能:将16位标志寄存器F的内容入栈保护。其操作过程与前述的PUSH指令类似。

(4) POPF。

指令功能:将当前栈顶和次栈顶中的数据字弹出并送回到标志寄存器F中。

4) I/O数据传送指令

(1) IN 累加器,端口号。

IN指令是将指定端口中的内容输入到累加器AL/AX中。端口号可以用8位立即

数直接给出;也可以将端口号事先安排在 DX 寄存器中,间接寻址 16 位长端口号(可寻址的端口号为 0～65 535)。其指令如下:

```
IN AL,PORT  ;AL←(端口 PORT),即将端口 PORT 中的字节内容读入 AL
IN AX,PORT  ;AX←(端口 PORT),即将由 PORT 两相邻端口中的字内容读入 AX
IN AL,DX    ;AL←(端口(DX)),即从 DX 所指的端口中读取 1 字节内容送往 AL
IN AX,DX    ;AX←(端口(DX)),即从 DX 和 DX+1 所指的两个端口中读取 1 个字内容送往 AX
```

例如:

```
IN AL, 40H
```

设当前 CS=1000H,IP=0050H;8 位端口 40H 中的内容为 55H,则该指令执行后,将 40H 端口中输入的数据字节 55H 传送到累加器 AL 中。

(2) OUT 端口号,累加器。

OUT 指令是将累加器 AL/AX 中的内容输出到指定的端口。与 IN 指令相同,OUT 指令的端口号可以由 8 位立即数给出,也可由 DX 寄存器间接给出。其指令如下:

```
OUT PORT,AL ;端口 PORT←AL,即将 AL 中的字节内容输出到由 PORT 直接指定的端口
OUT PORT,AX ;端口 PORT←AX,即将 AX 中的字内容输出到由 PORT 直接指定的端口
OUT DX,A    ;端口(DX)←AL,即将 AL 中的字节内容输出到由 DX 所指定的端口
OUT DX,AX   ;端口(DX)←AX,即将 AX 中的字内容输出到由 DX 所指定的端口
```

例如:

```
OUT DX, AL
```

设当前 CS=4000H,IP=0020H,DX=6A10H,AL=66H,则该指令执行后,将累加器 AL 中的数据字节 66H 输出到 DX 指定的端口 6A10H 中。

注意:I/O 指令只能用累加器作为执行 I/O 数据传送的机构,而不能用其他寄存器代替。另外,当用直接 I/O 指令时,寻址范围仅为 0～255,这适用于较小规模的微机系统;当需要寻址大于 255 的端口地址时,则必须用间接寻址的 I/O 指令。

2. 算术运算类指令

算术运算类指令能对无符号或有符号的 8/16 位二进制数以及无符号的压缩型/非压缩型(又称装配型/拆开型,或称组合型/未组合型)十进制数进行运算,有加、减、乘、除以及十进制调整 5 类指令。

1) 加法指令

(1) ADD d,s ;d←d+s。

指令功能:将源操作数与目标操作数相加,结果保留在目标中。并根据结果置标志位。

源操作数可以是 8/16 位通用寄存器、存储器操作数或立即数;目标操作数不允许是立即数,其他同源操作数。且不允许两者同时为存储器操作数。

例如:

```
ADD WORD PTR[BX+106BH],1234H
```

设当前 CS=1000H,IP=0300H,DS=2000H,BX=1200H,则该指令执行后,将立即数 1234H 与物理地址为 2226BH 和 2226CH 中的存储器字 3344H 相加,结果 4578H 保留在目标地址 2226BH 和 2226CH 单元中。

(2) ADC d,s ;d←d+s+CF。

带进位加法(ADC)指令的操作过程与 ADD 指令基本相同,唯一不同的是进位标志位 CF 的原状态也将一起参与加法运算,待运算结束,CF 将重新根据结果置成新的状态。

例如:

```
ADC AX,BX          ;AX=AX+BX+C(进位位)
ADC BX,[BP+2]      ;由 BX+2 寻址的堆栈段存储单元的字内容加 BX 和进位位,结果存入 BX
```

ADC 指令一般用于 16 位以上的多字节数字相加的软件中。

(3) INC d ;d←d+1。

指令功能:将目标操作数当作无符号数,完成加 1 操作后,结果仍保留在目标中。

目标操作数可以是 8/16 位通用寄存器或存储器操作数,但不允许是立即数。

例如:

```
INC SP                     ;SP=SP+1
INC BYTE PTR[BX+1000H]     ;将数据段中由 BX+1000H 寻址的存储单元的字节内容加 1
INC WORD PTR[SI]           ;将数据段中由 SI 寻址的存储单元的字内容加 1
INC DATA1                  ;将数据段中 DATA1 存储单元的内容加 1
```

注意:对于间接寻址的存储单元加 1 指令,数据的长度必须用 TYPE PTR、WORD PTR 或 DWORD PTR 类型伪指令加以说明;否则,汇编程序不能确定是对字节、字还是双字加 1。另外,INC 指令只影响 OF、SF、ZF、AF、PF 这 5 个标志,而不影响进位标志 CF,故不能利用 INC 指令来设置进位位,否则程序会出错。

2) 减法指令

(1) SUB d,s ;d←d−s。

指令功能:将目标操作数减去源操作数,其结果送回目标,并根据运算结果置标志位。源操作数可以是 8/16 位通用寄存器、存储器操作数或立即数;目标操作数只允许是通用寄存器或存储器操作数。并且不允许源操作数和目标操作数同时为存储器操作数,也不允许做段寄存器的减法。例如:

```
SUB AX,[BX]
```

设当前 CS=1000H,IP=60C0H,DS=2000H,BX=970EH,则该指令执行后,将 AX 寄存器中的目标操作数 8811H 减去物理地址 2970EH 和 2970FH 单元中的源操作数 00FFH,并把结果 8712H 送回 AX 中。各标志位的改变为:O=0(没有溢出),S=1(结果为负),Z=0(结果不为 0),A=1(有半进位),P=1(奇偶性为偶),C=0(没有借位)。SUB 指令的寻址方式和汇编语句形式也很多,例如:

```
SUB CL,BL                  ;CL=CL-BL
```

```
SUB AX,SP                ;AX=AX-SP
SUB BH,6AH               ;BH=BH-6AH
SUB AX,0AAAAH            ;AX=AX-0AAAAH
SUB DI,TEMP[SI]          ;从 DI 中减去由 TEMP+SI 寻址的数据段存储单元的字内容
```

(2) SBB d,s;d←d−s−CF。

这条指令与 SUB 指令的功能、执行过程基本相同,唯一不同的是完成减法运算时还要再减去进位标志 CF 的原状态。运算结束时,CF 将被置成新状态。这条指令通常用于比 16 位数宽的多字节减法,在多字节减法中,如同多字节加法操作时传递进位一样,它需要传递借位。

(3) DEC d;d←d−1。

指令功能:将目标操作数的内容减 1 后送回目标。该指令也称为减 1 指令。

目标操作数可以是 8/16 位通用寄存器和存储器操作数,但不允许是立即数。例如:

```
DEC BL                   ;BL=BL-1
DEC CX                   ;CX=CX-1
DEC BYTE PTR[DI]         ;由 DI 寻址的数据段字节存储单元的内容减 1
DEC WORD PTR[BP]         ;由 BP 寻址的数据段字存储单元的内容减 1
```

从以上指令汇编语句的形式可以看出,对于间接寻址存储器数据减 1 指令,要求用 TYPE PTR 类型伪指令来标识数据长度。

(4) NEG d;d←$\bar{d}$+1。

NEG 是一条求补码的指令,简称求补指令。

指令功能:将目标操作数取负后送回目标。

目标操作数可以是 8/16 位通用寄存器或存储器操作数。

NEG 指令是把目标操作数当成一个带符号数,如果原操作数是正数,则 NEG 指令执行后将其变成绝对值相等的负数(用补码表示);如果原操作数是负数(用补码表示),则 NEG 指令执行后将其变成绝对值相等的正数。

若 AL=00000100=+4,执行 NEG AL 指令后将各位变反,末位加 1 成为 11111100=$[-4]_{补}$;若 AL=11101110=$[-18]_{补}$,执行 NEG AL 指令后将变成00010010=+18。

例如:

```
NEG BYTE PTR[BX]
```

设当前 CS=1000H,IP=200AH,DS=2000H,BX=3000H,且由目标[BX]所指向的存储单元(=DS×16+BX=23000H)已定义为字节变量(假定为 FDH),则该指令执行后,将物理地址 23000H 中的目标操作数 FDH=$[-3]_{补}$,变成+3 后送回物理地址 23000H 单元中。

(5) CMP d,s;d−s,只置标志位。

指令功能:将目标操作数与源操作数相减但不送回结果,只根据运算结果置标志位。源操作数可以是 8/16 位通用寄存器、存储器操作数或立即数;目标操作数只可以是 8/16

位通用寄存器或存储器操作数。但不允许两个操作数同时为存储器操作数，也不允许做段寄存器比较。

比较指令使用的寻址方式与前面介绍过的加法和减法指令相同。

注意：执行比较指令时，会影响标志位 OF、SF、ZF、AF、PF、CF。当判断两比较数的大小时，应区分无符号数与有符号数的不同判断条件：对于两无符号数比较，只需根据借位标志 CF 即可判断；而对于两有符号数比较，则要根据溢出标志 OF 和符号标志 SF 两者的异或运算结果来判断。具体判断方法是：若为两无符号数比较，当 ZF＝1 时，则表示 d＝s；当 ZF＝0 时，则表示 d≠s。当 CF＝0 时，表示无借位或够减，即 d≥s；当 CF＝1 时，表示有借位或不够减，即 d＜s。

若为两有符号数比较，当 OF ⊕ SF＝0 时，则 d≥s；当 OF ⊕ SF＝1 时，则 d＜s。通常，比较指令后面跟一条条件转移指令，检查标志位的状态以决定程序的转向。

3) 乘法指令

乘法指令包括无符号数乘法指令 MUL 和有符号数乘法指令 IMUL。

(1) MUL s。

MUL s 是无符号数乘法指令，它完成两个无符号的 8/16 位二进制数相乘的功能。被乘数隐含在累加器 AL/AX 中；指令中由 s 指定的源操作数作为乘数，它可以是 8/16 位通用寄存器或存储器操作数。相乘所得双倍位长的积，按其高 8/16 位与低 8/16 位两部分分别存放到 AH 与 AL 或 DX 与 AX 中去，即对 8 位二进制数乘法，其 16 位积的高 8 位存于 AH，低 8 位存于 AL 中；而对 16 位二进制数乘法，其 32 位积的高 16 位存于 DX，低 16 位存于 AX 中。

若运算结果的高位字节或高位字有效，即 AH≠0 或 DX≠0，则将 CF 和 OF 两标志位同时置 1；否则，CF＝OF＝0。据此，利用 CF 和 OF 标志可判断相乘结果的高位字节或高位字是否为有效数值。例如：

```
MUL BYTE PTR[BX+2AH]
```

设当前 CS＝3000H，IP＝0250H，AL＝12H，DS＝2000H，BX＝0234H，且源操作数已被定义为字节变量(66H)，则该指令执行后，乘积 072CH 存放于 AX 中。根据机器的约定，因 AH≠0，故 CF 与 OF 两位置 1，其余标志位为任意状态，是不可预测的。

(2) IMUL s。

IMUL s 是有符号数乘法指令，它完成两个带符号的 8/16 位二进制数相乘的功能。

对于两个带符号的数相乘，如果简单采用与无符号数乘法相同的操作过程，那么会产生完全错误的结果。为此，专门设置了 IMUL 指令。

IMUL 指令除计算对象是带符号二进制数以外，其他都与 MUL 是一样的，但结果不同。

IMUL 指令对 OF 和 CF 的影响是：若乘积的高一半是低一半的符号扩展，则 OF＝CF＝0；否则，OF 和 CF 均为 1。它仍然可用来判断相乘的结果中高一半是否含有有效数值。另外，IMUL 指令对其他标志位没有定义。

有关 IMUL 指令的其他约定都与 MUL 指令相同。