第1章

基础知识

本章导读

汇编语言是计算机系统提供给用户的最快、最有效的语言，也是能对硬件直接编程的语言。因此，对空间和时间要求很高的程序或需要直接控制硬件的程序，必须使用汇编语言进行程序设计。要掌握汇编语言，必须对计算机中的数据表示及运算、8088/8086 系统结构有一定的了解，本章中将介绍这些知识。

数据表示与运算

1.1.1 进位计数制与不同基数制之间的转换

1. 二进制数、八进制数和十六进制数

(1) 二进制数

日常生活中一般采用十进制进行计数，但计算机只能识别 0,1 代码,也就是说,计算机采用二进制进行计数。二进制数只有 0,1 两个数码，其基数为 2,遵循逢二进一的原则，它的第 k 位权以 2^k 表示。

二进制数 $a_n a_{n-1}\cdots a_0 . b_{-1} b_{-2}\cdots b_{-m}$ 的值是

$$a_n \times 2^n + a_{n-1} \times 2^{n-1} + \cdots + a_0 \times 2^0 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \cdots + b_{-m} \times 2^{-m}$$

其中 a_i, b_j 为 0、1 两个数码中的一个。二进制的描述是在其尾部加注字母 B,例如：

$$10100101\text{B} = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 128 + 32 + 4 + 1 = 165$$

N 位二进制数可以表示 2^N 个数。例如,3 位二进制数可以表示 8 个数(见表 1-1)。

表 1-1 3 位二进制数与十进制数对应表

二进制数	000	001	010	011	100	101	110	111
相应的十进制数	0	1	2	3	4	5	6	7

4 位二进制数则表示十进制的 0～15 共 16 个数，如表 1－2 所示。

表 1－2 4 位二进制与十进制数的对应表

二进制数	0000	0001	0010	0011	0100	0101	0110	0111
相应的十进制数	0	1	2	3	4	5	6	7
二进制数	1000	1001	1010	1011	1100	1101	1110	1111
相应的十进制数	8	9	10	11	12	13	14	15

从上表可以看出，位数越多，二进制数越长，不便于人们阅读、书写和记忆，因此人们经常使用八进制数或十六进制数来表示二进制数。它们的基数和数码表示如表 1－3 所示。

(2) 八进制数

八进制数有 0、1、2、3、4、5、6、7、8 个数码，其基数为 8，遵循逢八进一的原则，它的第 k 位权以 8^k 表示。八进制的描述是在其尾部加注字母 O 或 Q。

八进制数 $a_n a_{n-1}\cdots a_0 . b_{-1} b_{-2}\cdots b_{-m}$ 的值是

$$a_n \times 8^n + a_{n-1} \times 8^{n-1} + \cdots + a_0 \times 8^0 + b_{-1} \times 8^{-1} + b_{-2} \times 8^{-2} + \cdots + b_{-m} \times 8^{-m}$$

例如：$534Q = 5 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 = 5 \times 64 + 3 \times 8 + 4 \times 1 = 348$

(3) 十六进制数

十六进制数有 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F 共 16 个数码，其中 A、B、C、D、E、F 表示 10～15 共 6 个数码，其基数为 16，遵循逢十六进一的原则，它的第 k 位权以 16^k 表示。十六进制的描述是在其尾部加注字母 H。

十六进制数 $a_n a_{n-1}\cdots a_0 . b_{-1} b_{-2}\cdots b_{-m}$ 的值是

$$a_n \times 16^n + a_{n-1} \times 16^{n-1} + \cdots + a_0 \times 16^0 + b^{-1} \times 16^{-1} + b_{-2} \times 16^{-2} + \cdots + b_{-m} \times 16^{-m}$$

例如：$2ACH = 2 \times 16^2 + 10 \times 16^1 + 12 \times 16^0 = 2 \times 256 + 10 \times 16 + 12 \times 1 = 684$

表 1－3 列出了几种常用的进位制的基数和数码。

表 1－3 几种常用的进位制的基数和数码

进位计数制	基数	数码
十六进制数	16	0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F
十进制数	10	0,1,2,3,4,5,6,7,8,9
八进制数	8	0,1,2,3,4,5,6,7
二进制数	2	0,1

2. 不同数制之间的转换

(1) 非十进制数转换为十进制数

各位非十进制数码乘以与其对应的权之和即为该数对应的十进制数。例如：

$1011100.1011B = 1 \times 2^6 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^{-1} + 1 \times 2^{-3} + 1 \times 2^{-4} = 92.6875\ D$

$A031H = 10 \times 16^3 + 3 \times 16^1 + 1 \times 16^0 = 41009$

$1001Q = 1 \times 8^3 + 1 \times 8^0 = 513$

(2) 十进制数转换为非十进制数

十进制数转换为非十进制数一般整数部分采用除基数取余法，小数部分采用乘基数取整法。除基数取余法的具体操作是把待转换的十进制数的整数部分不断除以要转换为的非十进制基数,并记下余数,直到商为 0 时为止；乘基数取整法的具体操作是把待转换的十进制数的小数部分不断乘以要转换为的非十进制基数，逐次记下乘积整数部分的值，直到小数部分为 0 时止。

现以十进制数转换为二进制数为例进行说明，十进制数转换为二进制数将“基数”换成 2 就成了除 2 取余法和乘 2 取整法，即对整数部分的处理是把待转换的十进制数的整数部分不断除以 2 ,并记下余数,直到商为 0 时为止；对小数部分的转换是把待转换的十进制数的小数部分不断乘以 2，逐次记下乘积整数部分的值，直到小数部分为 0 时止。

例 1.1 $N = 137.8125D$ 转换为二进制数。

整数部分 137D ,按除 2 取余法有：

$137/2 = 68$ ($a_0 = 1$)

$68/2 = 34$ ($a_1 = 0$)

$34/2 = 17$ ($a_2 = 0$)

$17/2 = 8$ ($a_3 = 1$)

$8/2 = 4$ ($a_4 = 0$)

$4/2 = 2$ ($a_5 = 0$)

$2/2 = 1$ ($a_6 = 0$)

$1/2 = 0$ ($a_7 = 1$)

故 $137D = 10001001B$

小数部分为 0.8125D,按乘 2 取整法有：

$0.8125 \times 2 = 1.625$ ($b_{-1} = 1$)

$0.625 \times 2 = 1.25$ ($b_{-2} = 1$)

$0.25 \times 2 = 0.5$ ($b_{-3} = 0$)

$0.5 \times 2 = 1.0$ ($b_{-4} = 1$)

故 $0.8125D = 0.1101B$

所以 $N = 137.8125D = 10001001.1101B$

十进制数转换为十六进制数和八进制数的方法与十进制数转换为二进制数的方法类同。

(3) 十六进制数与二进制数之间的转换

因为 $16 = 2^4$，所以一个十六进制数中的每一位可以用 4 位二进制数表示，便可形成相应的二进制数。

例 1.2

C	B	9	A
1100	1011	1001	1010

即 $CB9AH = 1100101110011010B$

反之，二进制数只要把它从低位到高位每 4 位组成一组，再用十六进制数来表示就可以了。

例 1.3

0111	0101	1011	1111
7	5	B	F

即　　0111010110111111B = 75BFH

1.1.2　二进制数和十六进制数运算

1. 二进制数的运算

加法规则：乘法规则：

$0+0=0$　　$0\times0=0$

$0+1=1$　　$0\times1=0$

$1+0=1$　　$1\times0=0$

$1+1=10$(1 为进位)　　$1\times1=1$

2. 十六进制数的运算

(1)十六进制加法

十六进制数的运算按照逢十六进一的规则进行，即当两个一位数之和 S 小于 16 时，与十进制数同样处理，如两个一位数之和 $S\geqslant16$ 时，则应该用 $S-16$ 及进位 1 来取代 S。

例 1.4

$$\begin{array}{r} 15C3H \\ +\ 3D45H \\ \hline 5308H \end{array}$$

(2) 十六进制数的减法

与十进制数类似，够减时可以直接相减，不够减时服从向高位借 1 为 16 的规则。

例 1.5

$$\begin{array}{r} 3DA6H \\ -\ 0FC3H \\ \hline 2DE3H \end{array}$$

1.1.3　数据表示

计算机内的数据有多种形式，其中最主要的是数值数据和字符数据。

1. 数值数据的表示

数值数据可以用不同的码制来表示，常用的有原码、补码和反码表示法。由于做加减运算时补码表示法的符号位可以参加运算，而且不影响运算结果的正确性，所以多数机器的有符号整数都采用补码表示法。这里只介绍补码表示法。

(1) 数的补码表示

①补码表示法中正数的表示

正数采用符号 - 绝对值表示，即数的最高有效位为 0 表示符号为正，数的其余部分则表示数的绝对值。

例如：假设机器字长为 8 位，则

$[+0]_{补}=00000000B$

$[+1]_{补}=00000001B$

$[+100]_{补}=01100100B$

② 补码表示法中负数的表示

用补码表示法来表示负数时可以采用“求反加 1”的方法来完成：先写出与该负数相对应的正数的补码表示（用符号－绝对值法），然后将其按位求反（即 0 变为 1，1 变为 0），最后在末位（最低位）加 1，就可以得到该负数的补码表示了。

例 1.6　机器字长为 8 位，写出 $N=-27$ 的补码表示。

+27D 可表示为　0001　1011

按位求反为　1110　0100

末位加 1 后为　1110　0101

用十六进制数表示为　E5H

即 $[-27]_{补}=E5H$

例 1.7　机器字长为 16 位，写出 $N=-32768$ 的补码表示。

32768D 可表示为　1000　0000　0000　0000

按位求反为　0111　1111　1111　1111

末位加 1 后为　1000　0000　0000　0000

用十六进制数表示为　8　0　0　0

即 $[-32768]_{补}=8000H$。

③数的表示范围

• 有符号数的表示范围

一般说来，n 位二进制补码的表示范围是

$$-2^{n-1}\leqslant N\leqslant 2^{n-1}-1$$

8 位二进制数可以表示 $2^8=256$ 个数。因为在补码表示法中 0 只有一种表示，即 00000000 对于 10000000 这个数，在补码表示法中被定义为 －128。这样，8 为补码能表示的范围为 －128～127。

$n=16$ 时的数的表示范围是

$$-32768\leqslant N\leqslant +32767$$

• 无符号整数的表示范围

在做无符号数处理时，把最高有效位作为数值处理。因此，16 位无符号数的表示范围是 $0\leqslant N\leqslant 65535$，8 位无符号数的表示范围是 $0\leqslant N\leqslant 255$。

（2）补码的运算

①求补运算

已知 $[X]$补，如何求 $[-X]$补？运算规则为：将 $[X]$补各位（含符号位）取反，末位加 1，即：

$$[-X]_{补}=\overline{[X]_{补}}+1$$

例 1.8　$[-117]_{补}=FF8BH$，求 $[+117]_{补}$。

对 $[-117]$补做求补运算：

$[-117]_{补}$为　1111　1111　1000　1011

按位求反后得　0000　0000　0111　0100

末位加 1 后得　0000　0000　0111　0101

此数正是 $[+117]_{补}=0075H$。

② 补码的加、减法运算

补码的加法规则是

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}$$

补码的减法规则是

$$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$$

其中的$[-Y]$补只要对$[Y]$补求补就可得到。

例 1.9　机器字长假定为 8 位，完成下列补码加法运算。

	十进制	二进制
①	23	00010111
	+36	+00100100
	59	00111011
②	36	00100100
	+(−23)	+11101001
	13	(进位 1)00001101
③	23	00010111
	+(−36)	+11011100
	−13	11110011
④	−23	11101001
	+(−36)	+11011100
	−59	(进位 1)11000101

可以看出，上述 4 个例子的计算结果都是正确的，在②和④中，从最高有效位向高位的进位由于机器字长的限制而自动丢失，但这并不会影响运算结果的正确性。

例 1.10　机器字长假定为 8 位，完成下列补码减法运算。

	十进制	补码	二进制
①	23	00010111	00010111
	−36	00100100	+11011100
	−13		11110011
②	36	00100100	00100100
	−(−23)	11101001	+00010111
	59		00111011
③	−23	11101001	11101001
	−(+36)	00100100	+11011100
	−59		(进位为 1) 11000101

④	-23	11101001	11101001
	-(-36)	11011100	+00100100
	13		(进位为1) 00001101

可以看出，补码减法是用对减数求补后把减法转换为加法进行的，它能自动地得到正确的结果。

2. 字符数据的表示

计算机内的常见字符包括：

字母：A、B、…、Z，a、b、…、z

数字：0～9

标点符号：!、,、?、;、:、等

算术运算符号：+、-、×、/

关系运算符号：>、<、=

专用字符：*、$、@、%、#、空格等

非打印字符：BEL（响铃）、LF（换行）、CR（回车）等

这些字符一般采用目前最常用的美国信息交换标准代码 ASCII码来表示。这种代码用一个字节（8位二进制码）来表示一个字符，其中低7位为字符的 ASCII 值，最高位一般用做校验位。常见字符的 ASCII 码表见附录 I。

1.1.4 定点数与浮点数

计算机中根据小数点的位置是否固定，将数的表示方法分为定点数表示法和浮点数表示法。

1. 定点数

定点数是指小数点位置固定不变的数。小数点的位置通常只有两种约定，小数点约定在最低数位右面的称为定点整数，可用来表示一个纯整数。小数点约定在符号位右面，最高数位左面的数称为定点小数，可用来表示一个纯小数。

无符号定点整数，即正整数，不需要设符号位，所有各数位都用来表示数值大小，并约定小数点在最低数的右面。

在定点整数或定点小数的表示法中，参加运算的数以及运算的结果必须在该定点数所能表示的数值范围之内，否则溢出。当发生溢出时，CPU 中的状态标志寄存器中的溢出标志 OF 置1。

2. 浮点数

定点数的表示比较单一，要么纯整数，要么纯小数，表示数的范围比较小，运算过程中也很容易发生溢出。计算机中也引入了类似与十进制的科学标识法（如 1.23456×10^4）来表示二进制实数，这种方法用来表示值很大或很小的数，也可以用它来表示既有整数又有小数的数。这种表示法标为浮点表示法，其小数点的实际位置随指数（阶）的大小而浮动。

浮点数由两部分组成　　阶码 E 和尾数 M。浮点数表示的数值为 $M\times R^E$。若尾数 M 为 m 位，阶码 E 为 e 位，则典型的浮点数格式如图 1-1 所示。

图 1-1 中，E 是阶码，即指数，为带符号定点整数，可用补码表示。E_s 阶符，表示阶的正负。阶为正，小数点实际位置向右浮动，阶为负，小数点实际位置向左浮动。

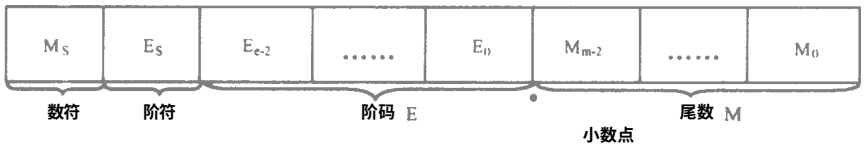

图 1－1 浮点数格式

M 是尾数，是带符号的定点小数，常用补码表示。M_s 是尾数的符号位，安排在最高位，表示该浮点数的正负。

小数点的位置约定在阶码最低位的右面，尾数最高数值位的左面，如图 1－1 所示。

R 是阶码的底，也就是尾数 M 的基(Radix)。一般基为 2,它是隐含约定的。

浮点数的表示范围主要由阶码的位数决定。精度则主要由尾数的位数决定。

1.2 8086/8088 系统结构

8086微处理器是美国 Intel 公司 1987 年推出的一种高性能的 16 位微处理器。它采用硅栅 HMOS 工艺制造,在 1.45 cm^2 单个硅片上集成了 29 000 个晶体管。8086 的内部结构规模较小，采用 40 引脚的双列直插式封装。8086 的一个突出特点是多重处理能力，用 8086 CPU 与 8087 协处理器以及 8089 I/O 处理器组成的多处理器系统，可大大提高其数据处理和输入／输出能力。与 8086 配套的各种外围接口芯片非常丰富，方便用户开发各种系统。

1.2.1 8086/8088 CPU 的内部结构

8086 CPU 内部结构如图 1－2 所示。按功能可分为两大部分：总线接口单元 BIU(Bus Interface Unit)和执行单元 EU(Execution Unit)。

1. 总线接口单元 BIU

总线接口单元 BIU 是 8086 CPU 同存储器和 I/O 设备之间的接口部件，负责对全部引脚的操作,即 8086 所有对存储器和 I/O 设备的操作都是由 BIU 完成的。其具体任务是：负责从内存单元中预取指令，并将它们送到指令队列缓冲器暂存。 CPU 执行指令时，总线接口单元要配合执行单元，从指定的内存单元或者 I/O 端口中取数据传送给执行单元，或者把执行单元的处理结果传送到指定的内存单元或 I/O 端口中。

总线接口单元 BIU 由 20位地址加法器、4 个段寄存器、16 位指令指针 IP、指令队列缓冲器和总线控制逻辑电路等组成。

(1) 地址加法器和段寄存器

8086 CPU 的 20 条地址线,可直接寻址 1 MB存储器物理空间。但 CPU 内部寄存器均为 16位的寄存器。那么,16 位的寄存器如何实现 20位地址寻址呢？它是由专门地址加法器将有关段寄存器内容(段的起始地址)左移 4 位后,与 16 位偏移地址相加，形成一个 20 位的物理地址，以对存储单元寻址。比如在取指令时，由 16 位指令指针 (IP)提供一个有效地址（逻辑地址或偏移地

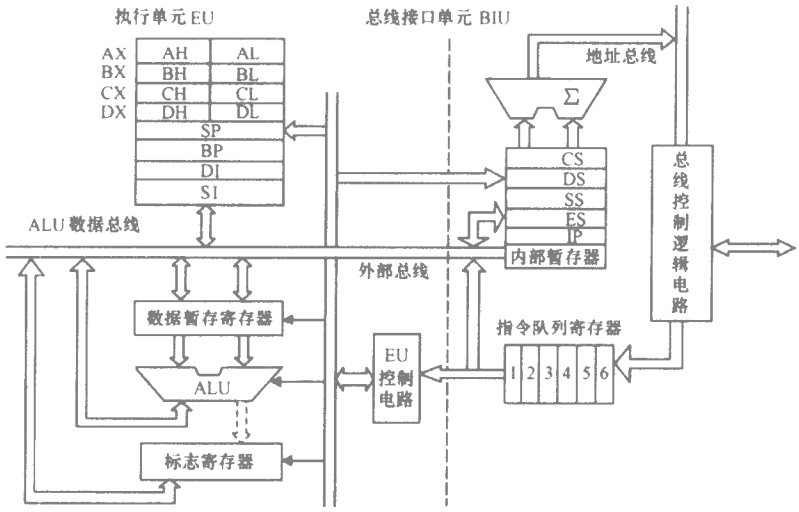

图 1-2　8086 CPU 内部结构示意图

址），在地址加法器中与代码段寄存器（CS)左移 4 位后的内容相加，形成实际的 20 位物理地址，送到总线上实现取指令的寻址。图 1-3 表示出了这一物理地址的形成过程。例如，假定代码段寄存器(CS) = 2000H，指令码单元的偏移地址(IP) = 1000H，则此指令的物理地址为 21000H。

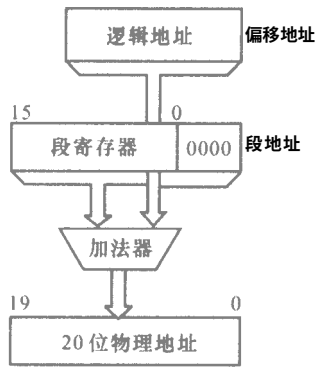

图 1-3　物理地址形成过程

（2）16 位指令指针 IP(Instruction Pointer)

指令指针 IP 用来存放下一条待执行指令在代码段中的偏移地址，它只有和 CS 相结合，才能形成指向指令存放单元的物理地址。在程序运行中，IP 的内容由 BIU 自动修改，使它总是指向下一条要取的指令在现行代码段中的偏移地址。用户不能直接访问 IP，但可以通过某些指令修改它的内容。例如，转移指令可将转移目标的偏移地址送入 IP 来实现程序的转移。

（3）指令队列缓冲器

当 EU 正在执行指令中，且不需占用总线时，BIU 会自动地进行预取指令操作，将所取得的指令按先后次序存入一个 6 字节的指令队列寄存器，该队列寄存器按“先进先出”的方式工作，并按顺序取到 EU 中执行。

（4）总线控制逻辑电路

总线控制逻辑电路将 8086 CPU 的内部总线和外部总线相连，是 8086 CPU 与内存单元或 I/O 端口进行数据交换的必经之路。它包括 16 条数据总线、20 条地址总线和若干条控制总线，CPU 通过这些总线与外部取得联系，从而构成各种规模的 8086 微型计算机系统。

2. 执行单元 EU

执行单元中包含一个 16 位的运算器 ALU，8 个 16 位的寄存器，一个 16 位标志寄存器 FLAGS，一个数据暂存寄存器和执行单元的控制电路，也就是说它已经包含了微处理机的 3 个基

本部件这个单元进行所有指令的解释和执行，同时管理上述有关的寄存器。

（1）算术逻辑运算单元（ALU）

它是一个 16 位的运算器，可用于 8/16 位二进制算术和逻辑运算，也可按指令的寻址方式计算寻址存储器所需的 16 位偏移量。

（2）标志寄存器(FLAGS)

它是一个 16 位的运算器，用来反映 CPU 运算的状态特征和存放某些控制标志。

（3）数据暂存寄存器

它协助 ALU 完成运算，暂存参加运算的数据。

（4）通用寄存器组

它包括 4 个 16 位的数据寄存器 AX、BX、CX、DX 和 4 个 16 位指针与变址寄存器 SP、BP、SI、DI。

（5）EU 控制电路

它负责从 BIU 的指令队列缓冲器中取指令，并对指令译码，根据指令要求向 EU 内部各部件发出控制命令，以完成各条指令规定的功能。

1.2.2 8086 CPU 寄存器组织

8086 微处理器内部共有 14 个 16 位寄存器，包括通用寄存器、地址指针和变址寄存器、段寄存器、指令指针和标志寄存器。8086 CPU 内部寄存器如图 1－4 所示。

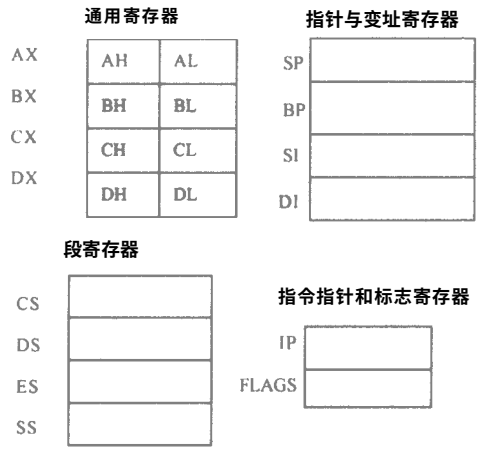

图 1－4 8086 CPU 内部寄存器

1. 通用寄存器

通用寄存器又称为数据寄存器，既可作为 16 位数据寄存器使用，也可作为两个 8 位数据寄存器使用。当用做 16 位时，称为 AX、BX、CX、DX 当用做 8 位时，AH、BH、CH、DH 存放高字节，AL、BL、CL、DL 存放低字节，并且可独立寻址，这样，4 个 16 位寄存器就可当做 8 个 8 位寄存器来使用。

作为通用寄存器，多数情况下，通用寄存器是用在算术和逻辑运算指令中，用来存放算术逻辑运算的源／目的操作数。根据各自特殊的使用场合，CX 又叫计数寄存器，AX 又叫累加器，BX 叫基址寄存器，而 DX 叫做数据寄存器。

2. 段寄存器

8086 CPU 有 20 条地址线，它可寻址的存储空间为 1 MB。而 8086 指令给出的地址编码只有 16 位，指令指针和变址寄存器也都是 16 位的，所以 CPU 不能直接寻址 1 MB 空间。为此采用分段寄存器，即 8086 用一组段寄存器将这 1 MB 存储空间分成若干个逻辑段，每个逻辑段长度 $\leqslant$64 KB，用 4 个 16 位的段寄存器分别存放各个段的起始地址（又称段基址），8086 的指令能直接访问这 4 个段寄存器。寻址根据给出一个相对于分段寄存器值所指定的起始地址的偏移值（也称为有效地址），以确定段内的确切地址。对物理地址的计算是在 BIU 中进行的，它先将段地址左移

4 位，然后与 16 位的偏移值相加。

段寄存器共有 4 个。代码段寄存器 CS 表示当前使用的指令代码可以从该段寄存器指定的存储器段中取得，相应的偏移值则由 IP 提供。堆栈段寄存器 SS 指定当前堆栈的底部地址。数据段寄存器 DS指示当前程序使用的数据所存区段的最低地址。而附加段寄存器 ES 则指出当前程序使用附加段地址的位置，该段一般用来存放原始数据或运算结果。

3. 地址指针和变址寄存器

这 4 个寄存器均是 16 位寄存器。堆栈指针 SP 用以指出在堆栈段中当前栈顶的地址，入栈(PUSH)和出栈(POP)指令是由 SP 给出栈顶的偏移地址。基址指针 BP 指出要处理的数据在堆栈段中的基地址，故称为基址指针寄存器。这里要注意两点： BP 并非确切的偏移量，它只是全部偏移量中的一个基本值；BP 所指的物理地址必须用堆栈寄存器 SS 来计算变址寄存器 SI 和 DI 用来存放当前数据段中某个单元的偏移量。在字符串处理中，被处理的数据称为源操作数据，它们存放在源变址寄存器 SI 给出的偏移地址上。而处理后的字符串则放在由目的变址寄存器 DI 给出的偏移地址上。这时 SI 和 DI 不能倒过来用。顺便指出， BX 做基地址用时也可以指定数据段的偏移基本量。

4. 指令指针

正常运行时，IP 中存放的是 BIU 要取的下一条指令的偏移地址。它具有自动加 1 功能，每当执行一次取指操作，它将自动加 1，使它指向要取的下一内存单元。每取一个字节后， IP 内容加 1；若取一个字后，IP 内容加 2。某些指令可使 IP 值改变，某些指令还可使 IP 值压入堆栈，或从堆栈中弹出。

5. 标志寄存器

标志寄存器 FLAGS 是一个 16 位的寄存器，8086 共使用了 9 个有效位，标志寄存器格式如图 1－5 所示。其中的 6 位是状态标志位，3 位为控制标志位。状态标志位是当一些指令执行后，所产生数据的一些特征的表征。而控制标志位则是可以由程序写入，以达到控制处理机状态或程序执行方式的表征。

D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

图 1－5 标志寄存器格式

下面指出 6 个状态标志位的功能。

CF (Carry Flag)进位标志位 当执行一个加法(或减法)运算使最高位产生进位(或借位)时，CF 为 1；否则为 0。

PF (Prity Flag) 奇偶标志位：该标志位反映运算结果中 1 的个数是偶数还是奇数。当指令执行结果的低 8 位中含有偶数个 1 时，PF 为 1；否则为 0。

AF (Auxiliary carry Flag) 辅助进位标志位：当执行一个加法（或减法）运算使结果的低 4 位向高 4 位有进位（或借位）时，AF 为 1 否则为 0。

ZF (Zero Flag) 零标志位：若当前的运算结果为零， ZF 为 1；否则为 0。

SF(Sign Flag)符号标志位：它和运算结果的最高位相同。

OF(Overflow Flag)溢出标志位：当补码运算有溢出时，OF 为 1；否则为 0.

以下 3 个控制标志位用来控制 CPU 的操作，由指令进行置位和复位。

DF(Direction Flag)方向标志位：用以指定字符串处理时的方向，当该位置"1"时，字符串以递减顺序处理，即地址以从高到低顺序递减；反之，则以递增顺序处理。

IF(Interrupt enable Flag)中断允许标志位 用来控制 8086 是否允许接收外部中断请求。若 IF = 1，8086 能响应外部中断；反之，则不响应外部中断。注意，IF 的状态不影响非屏蔽中断请求(NMI)和 CPU 内部中断请求。

TF(Trap Flag)跟踪标志位：是为调试程序而设定的陷阱控制位。当该位置"1"时，8086 CPU 处于单步状态，此时 CPU 每执行完一条指令就自动产生一次内部中断；当该位复位后，CPU 恢复正常工作。

1.2.3 8086 CPU 引脚功能

8086 CPU 具有 40 条引脚，采用双列直插式封装形式，如图 1-6 所示。为了减少芯片上的引脚数目，8086 CPU 采用了分时复用的地址／数据总线。为了适应各种使用场合，8086 CPU 可在两种模式下工作（最小模式和最大模式）。所谓最小模式，指系统中只有一个 8086 CPU，在这种系统中，8086 CPU 直接产生所有的总线控制信号，系统所需的外加其他总线控制逻辑部件最少；所谓最大模式，指系统中常含有两个或多个微处理器，其中一个为主处理器 8086 CPU，其他的处理器称为协处理器，它们是协助主处理器工作的。在最大模式工作时，控制信号是通过 8288 总线控制器提供的。因此，在不同方式下工作时，部分引脚（第 24～31 引脚）会具有不同的功能。图 1-6 括号中为最大模式时引脚名称。本节主要讨论最小模式下的引脚功能。

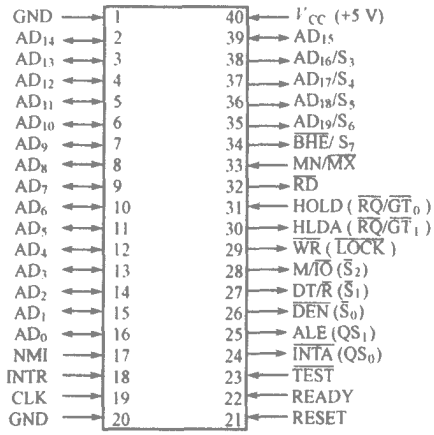

图 1-6 8086 CPU 引脚图

1. 地址／数据复用总线 $AD_0 \sim AD_{15}$

分时复用的地址／数据总线，具有双向、三态功能。用于输出低 16 位地址 $A_0 \sim A_{15}$ 和输入／输出数据 $D_0 \sim D_{15}$。在总线周期的第一个时钟周期 T_1 用来输出要访问的存储器单元或 I/O 的低 16 位地址 $A_0 \sim A_{15}$；而在总线周期的其他（$T_2 \sim T_3$）时钟周期，对于读周期来说是处于悬浮（高阻）状态；对于写周期来说则是传送数据。在此顺便指出，8088 CPU 的地址线同 8086 CPU 一样，内部数据总线也为 16 位，而外部数据总线宽度只有 8 位，8088 CPU 是一种准 16 位微处理器。

2. 地址／状态复用总线 $AD_{19}/S_6 \sim AD_{16}/S_3$

分时复用的地址／状态线，输出，三态。在总线周期的第一个时钟周期 T_1 用来输出访问存储器的 20 位物理地址的最高 4 位地址（$AD_{19} \sim AD_{16}$），与 $AD_{15} \sim AD_0$ 一起构成访问存储器的 20 位物理地址。当 CPU 访问 I/O 端口时，$AD_{19} \sim AD_{16}$ 保持为"0"。

而在其他时钟周期，则用来输出状态信息。其中 S_6 为 0，用来指示 8086 CPU 当前正与总线相连 S_5 状态用来指示中断允许标志位 IF 的当前设置：若 IF = 1，表明当前允许可屏蔽中断请求 若 IF = 0，则禁止可屏蔽中断请求。S_4、S_3 组合起来用来指示 CPU 当前正在使用哪个段寄存器 。S_4、S_3 的代码组合与对应的状态如表 1-4 所示。

表 1-4 S_4、S_3 状态编码表

S_4	S_3	当前使用的段寄存器
0	0	当前正在使用 ES
0	1	当前正在使用 SS
1	0	当前正在使用 CS(访问 I/O 端口时，不使用任何段寄存器)
1	1	当前正在使用 DS

3. 控制总线

(1) BHE/S_7(Bus High Enable/Status)：高 8 位数据总线允许／状态复用引脚，三态输出，低电平有效，在总线周期的第一个时钟周期 T_1 用来表示总线高 8 位 AD_{15} ~ AD_8 上的数据有效。若 $\overline{BHE}$ = 1，表示仅在数据总线 AD_7 ~ AD_0 上传送数据。读／写存储器或 I/O 端口以及中断响应时，$\overline{BHE}$ 用做选体信号，与最低位地址码 AD_0 配合，表示当前总线使用情况，如表 1-5 所示。

S_7 用来输出状态信息，在当前的 8086 芯片设计中未被赋予定义，暂作备用。

表 1-5 $\overline{BHE}$ 和 AD_0 编码对数据访问的影响

$\overline{BHE}$	AD_0	总线使用情况
0	0	16 位数据总线上进行字传送
0	1	高 8 位数据总线上进行字节传送(访问奇地址存储单元)
1	0	低 8 位数据总线上进行字节传送(访问偶地址存储单元)
1	1	无效

(2) $\overline{RD}$(Read)：读信号，三态、输出。当 $\overline{RD}$ = 0 低电平有效时，表示当前 CPU 正在对存储器或 I/O 端口进行读操作。$\overline{RD}$ = 0 与 M/$\overline{IO}$ 信号高电平配合，表示读存储器操作；$\overline{RD}$ = 0 与 M/$\overline{IO}$ 信号低电平配合，表示读 I/O 端口操作。

(3) WR(Write)：写信号，三态、输出。当 $\overline{WR}$ = 0 低电平有效时，表示当前 CPU 正在对存储器或 I/O 端口进行写操作。跟 $\overline{RD}$ 信号一样由 M/$\overline{IO}$ 信号区分对存储器或 I/O 端口的访问。

(4) M/IO(Memory/Input Ouput)　存储器或 I/O 端口选择控制信号，三态输出。M/$\overline{IO}$ = 1，表示当前 CPU 正在访问存储器；M/$\overline{IO}$ = 0，表示 CPU 当前正在访问 I/O 端口。一般在前一个总线周期的 T_4 时钟周期，就使 M/IO 端产生有效电平，然后开始一个新的总线周期。在此新的总线周期中，M/$\overline{IO}$ 一直保持有效电平，直至本总线周期的 T_4 时钟周期为止。在 DMA 方式时，M/$\overline{IO}$ 被悬空为高阻状态。

(5) READY：准备就绪信号，输入，高电平有效。READY = 1 时，表示 CPU 访问的存储器或 I/O 端口已准备好传送数据，马上可以进行读／写操作。若 CPU 在总线周期 T_3 状态检测到

READY 信号为低电平，表示存储器或 I/O 设备尚未准备就绪，CPU 自动插入一个或多个等待状态 T_W，直到 READY 信号变为高电平为止。

(6) INTR(Interrupt Request) 可屏蔽中断请求信号，输入，电平触发，高电平有效。当 INTR = 1 时，表示外设向 CPU 发出中断请求，CPU 在每个指令周期的最后一个 T 状态去采样该信号，若 INTR = 1 且 IF = 1 时，则 CPU 就会在结束当前指令后去响应中断，转去执行中断服务程序。

(7) INTA(Interrupt Acknowledge) ：中断响应信号，输出，低电平有效。表示 CPU 响应了外设发来的 INTR 信号。在中断响应周期的 T_2、T_3、T_W 时钟周期内使 INTA引脚变为低电平，通知外设端口可向数据总线上放置中断类型号，以便获取相应中断服务程序的入口地址。

(8) NMI (NO - Maskable Interrupt) ：不可屏蔽中断请求信号，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，一旦该信号有效，就在现行指令结束后引起中断。

(9) TEST: 测试信号，输入，低电平有效。当 CPU 执行 WAIT 指令时，每隔 5 个时钟周期对 $\overline{TEST}$进行一次测试，若测试到 TEST 为高电平状态，则 CPU 处于空闲等待状态，直到 TEST低电平有效，CPU 才结束等待状态继续执行后续指令。 TEST引脚信号用于多处理器系统中，实现 8086 CPU 与协处理器间的同步协调功能。

(10) RESET ：复位信号，输入，高电平有效。RESET 信号至少要保持 4 个时钟周期。CPU 检测到 RESET 为高电平信号后，停止进行操作，并将标志寄存器、段寄存器、指令指针 IP 和指令队列等复位到初始状态。 CPU 复位后，从 FFFF0H 单元开始读取指令。

(11) ALE(Address Latch Enable) 地址锁存允许信号，输出，高电平有效。由于 8086 CPU AD_0 ~ AD_{15}是地址／数据复用的总线，CPU 与内存、I/O 电路交换信息时，先利用此总线传送地址信息，后传送数据信息。为此，在任何一个总线周期的 T_1 时钟 ALE 端产生正脉冲，利用它的下降沿将地址信息锁存，达到地址信息与数据信息复用分时传送的目的。

(12) DT/R(Data Transmit/Receive) ：数据发送／接收控制信号，三态输出。在最小模式系统中使用 8286/8287 作为数据总线收发器时，DT/$\overline{R}$信号用来控制 8286/8287 的数据传送方向。当 DT/$\overline{R}$ = 1 时，则进行数据发送，即完成写操作 当 DT/$\overline{R}$ = 0 时，则进行数据接收，即完成读操作。

(13) DEN(Data ENable)：数据允许信号，三态输出，低电平有效。在最小模式系统中，用做数据收发器 8286/8287 的选通控制信号。在 DMA 方式时，$\overline{DEN}$为悬空状态。

(14) HOLD(Hold Request) ：总线请求响应信号，输出，高电平有效。通常把具有对总线控制能力的部件称为主控设备，显然 CPU 是一种主控设备。如果在一个总线上有两个主控设备时，它们对总线的控制就需要进行协调，即同一时间中只能由一个主控设备起作用。在较简单的系统中通常以 CPU 的控制为主，平时对总线的控制权总是在 CPU 的手上。当另一个主控设备需要使用总线(即获得总线控制权)时，就向 CPU 的 HOLD 引脚送出一个高电平的请求信号。

(15) HLDA(Hold Acknowledge) 总线请求响应信号，输出，高电平有效。HLDA 输出高电平有效时，表示 CPU 已响应其他部件的总线请求，通知提出请求的设备可以使用总线。与此同时，CPU 的有关引脚呈现高阻状态，从而让出系统总线，这种状态将一直延续到 HOLD 端的请求撤销，即输入电平降为低电平为止，CPU 恢复对总线的控制权。

(16) MN/MX(Minimun/Maximun): 工作方式选择信号，输入。MN/MX = 1，表示 CPU 工作在最小方式系统；MN/MX = 0，表示 CPU 工作在最大方式系统。

(17) CLK(Clock) 主时钟信号，输入。CLK 时钟输入端为微处理器提供基本的定时脉冲，通

常与 8284 时钟发生器的时钟输出端 CLK 相连。时钟引脚 CLK 要求输入一个符合处理机芯片工作频率要求的时钟，这个时钟表最好具有 33% 的占空度，使处理器内获得一个最佳的工作定时。8086 CPU 可使用的时钟频率随芯片型号不同而异， 8086 为 5 MHz,8086 - 2 为 8 MHz,8086 - 1 为 10 MHz

4. 最大模式下的有关引脚功能

下面对 8086 CPU 工作在最大模式系统中几个重新定义的引脚做简要说明。

(1) S_2、S_1、S_0(Bus Cycle Status)：总线周期状态信号，三态输出。在最大方式系统中，它用来作为总线控制器 8288 的输入，经译码后产生如表 1 - 6 所示的 7 个控制信号。状态线的组合情况如下。

表 1 - 6 S_2、S_1、S_0 编码的功能与 8288 控制信号表

状态			CPU 总线周期	8288 控制信号
S_2	S_1	S_0		
0	0	0	中断响应	$\overline{INTA}$
0	0	1	读 I/O 端口	$\overline{IORC}$
0	1	0	写 I/O 端口	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	暂停	无
1	0	0	访问代码	$\overline{MRDC}$
1	0	1	读存储器	$\overline{MRC}$
1	1	0	写存储器	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	无效	无

此外，最大模式时锁存地址所需的 ALE、控制数据接收器用的 $\overline{DEN}$ 和 DT/$\overline{R}$ 信号也由 8288 提供。

(2) RQ/GT_0 和 RQ/GT_1：总线请求信号输入／总线请求允许信号输出，双向、低电平有效。它可以用来协调 8086 CPU 与外部处理机对局部总线使用权的，且总是与协处理机 8087 和 I/O 处理机 8089 的相应端 $\overline{RQ}/\overline{GT}$ 连接在一起。当某个外部处理机要占用总线时，就从 $\overline{RQ}/\overline{GT}$ 引脚向 8086 输出一个负脉冲，提出使用总线的申请。如果 8086 正好完成一个总线周期，就会让出总线控制权，并从同一条引脚向该处理机送出一个负脉冲，以示对方可以使用总线。该处理机用完总线后,再以一个负脉冲向 8086 CPU 报告。两个信号分别在同一引线上传输，但方向相反，其中 $\overline{RQ}/GT_0$ 优先级高。

(3) LOCK 总线封锁信号,三态输出,低电平有效。LOCK有效时,表示 CPU 不允许其他总线控制器占用总线。 $\overline{LOCK}$ 信号是由软件设置的，目的是保证 8086 CPU 在一条指令的执行中，总线使用权不会为其他主设备所打断。如果在某一条指令的前面加一个 LOCK 前缀，这条指令执行时,就会使 CPU 产生一个 LOCK 信号，直到这条指令结束为止，即它只在一条指令执行的周期内有效。

(4) QS_1、QS_0(Instruction Queue Status)：指令队列状态，输出。作为指令队列状态的标志，当 8086 的 EU 在指令队列中取指令时，队列中的变化情况就以这两个输出位的状态编码表示出来，

以便于外部其他处理机对 8086 内部指令队列进行跟踪。QS_1、QS_0 的编码含义如表 1-7 所示。

表 1-7 QS_1、QS_0 编码含义

QS_1	QS_0	指令队列状态
0	0	无操作
0	1	从队列中取指令第一字节
1	0	队列为空
1	1	从队列中取指令后续字节

5. 电源线 V_{CC}和地线 GND

8086 只需单一的 +5 V 电源，由 V_{CC}端输入，GND 是接地端。

1.3 计算机语言基本概念

计算机语言的种类非常多，总的来说，可以分成机器语言、汇编语言、高级语言三大类。

1.3.1 机器语言

1. 机器指令

机器指令是指挥计算机完成某一基本操作的命令，是由硬件电路设计决定的，因而也叫做硬指令。机器指令是由一组能为计算机所接受的 0 和 1 组成的二进制代码。机器指令由操作码和地址码组成，规定了要求计算机完成的操作及其操作的对象（数据或存储单元地址）。

2. 指令系统

每台计算机所特有的、全部指令的集合构成该 CPU 的指令系统。不同的 CPU 具有不同的指令系统。

3. 机器语言程序

机器指令的集合构成了机器语言，用机器语言编写的程序就是机器语言程序。

计算机所能识别的语言只有机器语言，但机器语言非常难于记忆和识别。

通常人们编程时，不采用机器语言，而采用汇编语言和高级语言。

1.3.2 汇编语言

1. 汇编指令

汇编指令是用助记符号表示的机器指令，它与机器指令一一对应。

2. 汇编程序

计算机不能直接识别汇编指令，要让机器接受汇编指令还需要有一个将汇编指令翻译为机器指令的过程，这个过程称为汇编。汇编程序就是把汇编语言源程序翻译成机器语言程序的一

种系统软件。

IBM PC 机中的汇编程序有 ASM 和 MASM 两种，ASM 称为小汇编程序，它只需要较小的存储区。MASM 称为宏汇编程序，它需要的存储区较大，但功能较强，且具有宏汇编能力，ASM 则不具备这种能力。

3. 伪指令

伪指令就是向汇编程序提供如何进行汇编工作的命令，也叫汇编控制命令。

伪指令没有对应的机器指令，汇编时不产生机器码。

4. 汇编语言

汇编指令、伪指令、宏指令和汇编程序一起组成了汇编语言。汇编语言直接面向机器，用汇编语言编制的程序简洁、快速，常用于对运行速度要求较高的实时控制等场合。

用汇编语言编制的用户程序称为汇编语言源程序。

汇编语言的实质和机器语言是相同的，都是直接对硬件操作，但指令采用了英文缩写的标识符，更容易识别和记忆而其所占用的存储空间和执行速度与机器语言相仿。

1.3.3 高级语言

高级语言主要是相对于汇编语言而言，它是较接近自然语言和数学公式形式的编程，基本脱离了机器的硬件系统，用人们更易理解的方式编写程序。高级语言并不是特指某一种具体的语言，而是包括很多编程语言，如目前流行的 VB、VC、FoxPro、Delphi 等，这些语言的语法、命令格式都各不相同。

高级语言所编制的程序不能直接被计算机识别，必须经过转换才能被执行，按转换方式可将它们分为两类：

解释(Interpret)类：执行方式类似日常生活中的“同声翻译”，应用程序源代码一边由相应语言的解释器‘翻译’成目标代码(机器语言),一边执行,因此效率比较低,而且不能生成可独立执行的可执行文件，应用程序不能脱离其解释器，但这种方式比较灵活，可以动态地调整、修改应用程序，BASIC 语言 dBASE 数据库管理语言即采用这种方式。

编译类：编译(Compile)是指在应用源程序执行之前，就将程序源代码一次性“翻译”成目标代码（机器语言），然后再在机器上运行目标程序，因此其目标程序可以脱离其语言环境独立执行，使用比较方便，效率较高。但应用程序一旦需要修改，必须先修改源代码，再重新编译生成新的目标文件(*.OBJ 才能执行，只有目标文件而没有源代码，修改很不方便，现在大多数的编程语言都是编译型的，例如 Visual C++、Visual FoxPro、Delphi 等。

1.3.4 汇编语言与高级语言的比较

和汇编语言相比，高级语言不但将许多相关的机器指令合成为单条指令，并且去掉了与具体操作有关但与完成工作无关的细节，例如使用堆栈、寄存器等，这样就大大简化了程序中的指令。但用高级语言编写如过程控制、接口控制、设备通信等方面的程序翻译成机器语言后，程序代码冗长，占用存储空间大，执行速度慢。相反，这样的情况下汇编语言能直接控制计算机的内存和外设，产生的目标程序简短，占用存储空间小，执行速度快。