第1章 预备知识

本章主要讲解有关汇编语言的几个基本概念以及学习汇编语言时必须掌握的微机硬件的基本知识。

1.1 基本概念

1.1.1 汇编语言

1.1.1.1 代码指令与代码程序

代码指令也称机器指令，它是计算机能识别的一组二进制代码，用于指出计算机所要进行的操作(如某种运算)以及操作的对象(如参与操作的数或操作数的地址)。例如：

1. 传送操作

把数“7”送到寄存器 AL 中。用下列代码指令实现此功能：

1011000000000111

机器执行该指令后，AL 的内容等于“7”，习惯上记为(AL)=7。

2. 加法操作

把数“3”与寄存器 AL 的内容相加，结果仍放在 AL 中，用下列代码指令实现：

0000010000000011

机器执行该指令后 AL 的内容等于“10”，即 (AL)=10。

3. 传送操作

把寄存器 AL 的内容送到地址为 5 的单元中，可用下列代码指令实现：

10100010010100000000000

机器执行该指令后单元 5 的内容为“10”，记为 (5)=10 。

不同的机器操作，由不同的代码指令实现。一个计算机能够执行的各种代码指令的集合，就是该计算机的指令系统。指令系统是计算机基本功能的体现，不同的计算机其指令系统也不相同。

计算机要完成一项有意义的功能，就需要一串代码指令来实现，通常把完成一定功能的代码指令串称为程序。

1.1.1.2 机器语言与机器语言程序

由于代码指令能够被机器识别并执行，因而把指令系统及书写代码程序的规则称为机器语言。代码程序又称为机器语言程序。

不同类型计算机的机器语言可不相同，因此计算机的机器语言是为特定的计算机或特

定的计算机系列设计的，它没有通用性。

虽然计算机能够直接接受机器语言程序，但是二进制代码指令组成的机器语言程序不便于记忆及阅读，而使用机器语言编写程序更为困难，故机器语言在实际中很少直接使用。除引导程序和检测程序外，微机中一般不直接用机器语言编写程序。

1.1.1.3 汇编语言

由于代码指令难于使用，人们采用符号代替二进制代码，于是产生了对代码指令符号化描述的汇编语句。每一种代码指令对应一种汇编语句。下面列出了前面提到的三条代码指令及其对应的汇编语句：

代码指令	汇编语句
1011000000000111	MOV AL,7
0000010000000011	ADD AL,3
101000100101000000000000	MOV VC,AL

其中 VC 代表地址 5，可看成是地址编号 5 的符号地址。从以上例子可以看出汇编语句比代码指令形象生动，便于记忆，从语句形式就能看出语句功能。

汇编语言是汇编语句和使用这些语句编写程序的一套语法规则的总称，它是描述机器指令系统的程序设计语言，其基本内容是机器语言的符号化描述。通常汇编语言的执行语句与代码指令一一对应，所以汇编语言是面向机器的初级的计算机程序设计语言。

与机器语言一样，汇编语言也是依赖于机器的，没有通用性。

1.1.1.4 高级语言

高级语言是面向算法过程的计算机程序设计语言，它的基本内容与具体的机器无关。例如，FORTRAN、BASIC、COBOL 及 PASCAL 语言等都是高级语言。这里仍以相同两个数的加法运算为例，用机器语言、汇编语言、高级语言分别列出了相应的语句：

机器语言	汇编语言	高级语言
1011000000000111	MOV AL,7	
0000010000000011	ADD AL,3	VC=7+3
101000100101000000000000	MOV VC,AL	

从以上例子可以看出，高级语言比汇编语言更好理解，它更接近数学形式，书写起来也很简便。

与汇编语言相比，高级语言有两个主要优点：

1. 使用方便。易读、易写、易调试，因而容易学习，编程速度也快。
2. 便于移植。程序很容易从一种计算机换到另一种计算机上运行。

与高级语言相比，汇编语言的优点是：

1. 效率高。其程序比相同功能的高级语言程序所占内存少(可成倍地缩短内存)，而且运行速度快(至少快 10 倍以上)。
2. 能将计算机的全部功能提供给用户使用。这是因为，汇编语言能最直接最充分地描述计算机语言，使用汇编语言就是直接使用机器语言。当然也就要求用户对机器性

能有较多的了解。通常在下列情况下必须使用汇编语言：

(1)扩充或修改系统软件；
(2)扩充或修改系统设备；
(3)对执行速度有严格要求；
(4)系统内存受到限制。

1.1.2 汇编程序

1.1.2.1 源程序与目标程序

用汇编语言编写的程序称为汇编语言源程序或汇编语言程序，本书中也简称为源程序。

前面已经指出：计算机只能识别和执行代码程序。汇编语言源程序或其他高级语言源程序都不能在计算机中运行，它们必须被翻译成对应机器的代码程序才能在计算机中运行。此相应的代码程序称为目标程序。例如可以把前面例子中的三条机器指令看成是三条汇编语句(一个小源程序)的目标程序。

1.1.2.2 汇编程序的作用

汇编程序(assembler/assembly program)是把源程序转变为相应目标程序的翻译程序。这个翻译过程称为汇编。

显然，汇编程序是一个代码程序，它能在机器中直接运行。操作员用汇编命令调用汇编程序对指定的源程序进行汇编。汇编程序按命令逐条读入指定的源程序并将其翻译成相应的代码指令，这样就生成了目标程序（*.OBJ 文件)，同时还产生了列表文件(*.LST)及交叉参考文件。后两个文件用于对源程序进行查错及分析。顺便指出，对 MASM 汇编程序来说，汇编得到的目标程序虽然也是代码程序，但不能立即运行，它必须经过连接程序(LINK)进行连接才能得到可执行的代码程序(*.EXE)。这些在以后详细讲解。

1.1.2.3 汇编程序中的主要数据结构

为了把源程序翻译成目标程序，汇编程序使用了一些表格。通常软件中的表格有两种：静态表和动态表。汇编中的静态表是汇编程序在执行前就设计好并填写好的表格。汇编中的动态表是汇编程序在执行过程中根据用户编写的源程序形成的表格。这里仅简要介绍四种表格：

1. 操作符表

操作符表是一种静态表，它的基本内容如下所示：

操作符	特征	操作码
MOV	...	10110000
ADD	...	00000000
CALL	...	11101000
...	...	...

其中操作码取最简单的代码指令形式的前 8 位二进制数码；操作符是代码指令中操作码的符号化表示，通称(操作码的)助记符，汇编程序正是靠查阅此表把源程序中的操作符

(助记符)翻译成机器能识别的操作码;特征是把该操作符的各种特殊性质,如操作符的类型以及翻译该语句的子程序入口等用二进制数码表示出来。注意:操作符在表中也是用二进制表示,其中每一个字母用 8 位二进制(即 ASCII 码,见附录 D)表示。

2. 寄存器名表

寄存器名表也是一个静态表,它的形式和操作符表类似。下面为其简要形式:

寄存器名	寄存器代码	
	W	REG
AL	0	100
AX	1	000
...	...	...

寄存器名中的字母在表中用 ASCII 码表示。寄存器代码用二进制表示。汇编语句中使用的寄存器名,如"MOV AL,7"语句中的 AL,为一个 8 位寄存器名,计算机是不能识别的,汇编程序也是靠查阅此表把 AL 翻译成计算机能识别的寄存器代码"0100"。

3. 伪操作符表

汇编语言中还定义了许多伪操作符。伪操作符主要用于表示程序的结构,告诉汇编程序怎样处理源程序。伪操作符的功能由汇编程序调用相应的子程序来完成。除数据的伪操作符外,伪操作符不在目标程序中生成相应的代码,不占内存空间。伪操作符表的形式示意如下:

伪操作符	相应子程序入口
SEGMENT	100
ENDS	200
END	300
...	...

伪操作符中的字母在表中用 ASCII 码表示,相应子程序入口用十六进制数写出。

4. 名字表

名字表也称符号表,它是一个动态表。汇编程序在汇编源程序时,根据用户在源程序中定义的名字边查边建立该表。汇编语言中允许用户自己定义一些名字代替二进制的地址码。在汇编语句中使用的用户自定义的名字,翻译成相应代码指令时根据该表换成相应的二进制地址码。这些名字与地址码事先是无法规定的,只能根据源程序在汇编过程中逐步确定。名字表的形式示意如下:

名字	特性	地址
LAA	0111	400
LBB	0111	402
SUB1	0000	...
...	...	...

其中左边为名字，名字中的字母也是用 ASCII 码表示。特性表示该名字的种类，如是否已定义过等。这里假设用 4 位二进制表示特性，前两位表示该名字是标号还是变量，标号用“01”表示，变量用“10”表示；后两位表示该名字是否已定义过，已定义就是已分配了地址，否则为未定义。已定义用“11”表示，未定义用“00”表示。特征为“0000”时，表示名字无定义，并且也不能确定是标号还是变量。

1.1.2.4 汇编过程

汇编程序执行时，要对源程序字符逐个进行分析。每次从源程序的第一个字符取到最后一个字符，称为一次扫描。**汇编程序通常要两次扫描源程序才能实现把源程序翻译成代码程序。**

1. *第一次扫描过程*

汇编程序第一次扫描源程序，主要是建立名字表。建立名字表的过程主要是对名字进行语法检查和地址分配。为了分配名字的地址，汇编程序中设立了一个地址计数器（假设该地址计数器的名字为“$”）。下面简要说明第一次扫描的功能。

[**例**]：设有下列源程序语句，并设地址计数器 $ 的内容为 400（十六进制数），这说明下一个语句应存于地址为 400 开始的字节内。我们的讲解从列名字表的中途开始。

源程序：

```
        ...
LAA: ADD AL,7
LBB: ADD AL,3
     CALL SUB1 ;调用语句
     ...
SUB1: ......
     ...
```

当取出名字 LAA（取到分隔符“:”时暂停）时，用它去查名字表，如果表中有此名字，再检查特征，如果特征为已定义的，说明这个标号名用重了，是重名语法错；如果特征是未定义的，就将 $ 中的地址填入表中名字 LAA 的地址栏中。如果无此名字，那么在表中添加一行，填入名字 LAA，同时将 $ 中的地址也填于该表中 LAA 对应的地址栏中，并在特征栏中填已定义，然后继续扫描源程序。当取到回车符后，表示该语句已取完。根据查操作符表，知道该操作是否正确（查不到时，为错误），并知道了该语句对应的代码指令长度为两字节，于是（$）=（$）+2，即（$）=402。用同样的办法继续扫描其余源程序，将 LBB 也列入表中，此时（$）=404。直到取出名字 SUB1，此时 SUB1 在语句地址部分出现，前面又没见过，因而它是无定义的，也列入表中。至此，名字表中的内容如下：

```
        ...
LAA     0111        400
LBB     0111        402
SUB1    0000
```

继续扫描源程序，直到遇到后跟冒号的 SUB1 才能确定 SUB1 所对应的地址。此时就能回头填写表中 SUB1 的地址，并修改其特征为“0111”（标号，有定义）。因此只有第一次扫描完源程序，才能全部完成名字表的建立。最后，名字表中的名字都应是已定义的名字，否则肯定出现了名字无定义的语法错。

思考题：上面的名字表中 SUB1 对应的地址是否可填写成“404”？为什么？

2. 第二次扫描过程

汇编程序第二次扫描源程序时是一边扫描，一边查表，从而从表中得到各符号的二进制代码并进行代真（通常把汇编语句翻译成代码指令的过程称为代真）。在第二次扫描时，同时对第一次扫描中无法检查的语法错误进行检查，例如检查操作符后的地址是否合适，只能在代真时检查，而第一次扫描只检查这些地址是否为汇编语言所允许。但是并不是所有允许的地址都能在任意语句中出现。一个操作符允许使用哪些地址形式是一定的。

以上是对汇编过程的概略介绍，真正的汇编程序的功能要复杂得多。例如 MASM 5.0 版的宏汇编程序长达 256K 字节，MASM 6.0 版为 512K 字节。其中使用的表格也远非四个。本书所讲汇编语言，就是讲解 MASM 宏汇编程序所实现的功能。

1.2 系统结构

1.2.1 微处理器概况

1.2.1.1 iAPX86/88 微处理器系列

iAPX86/88 是以 Intel 8086 微处理器为基础发展起来的微处理器系列，中央处理器（CPU）可为 8086，8088，80186，80286 等中的一种，并配以适当数目的支持芯片（如 8087，80287，80130），从而构成具有各种扩展或增强功能的微处理器系统。表 1.1 给出了 iAPX86/88 系列的构成例子。

表 1.1 iAPX86/88 微处理器系列的一些构成例子

类别 / 器件 / 品名	中央处理器				支持芯片			
	8086	8088	80186	80286	8087	80287	8089	80130
iAPX86-10	1							
88-10		1						
86-11	1						1	
88-11		1					1	
86-20	1				1			
88-20		1				1		
86-21	1				1		1	
88-21		1			1		1	
86-30	1							1
88-30		1						1
186-10			1					
186-20			1		1			
186-30			1					1
286-10				1				
iAPX286-20				1		1		

8086 CPU 是一个 16 位微处理器，程序主要在该 CPU 上运行，它还控制其他处理器的

运行。

8088 CPU 是一个准 16 位微处理器，它采用 8 位数据总线，而使用 16 位内部结构。它与 8086 的区别仅在于存取 16 位数据时内部处理有所不同。对软件工作者来说，8088 与 8086 除了运行速度上的差别外，其他方面没有区别。因此，本书围绕 8086 CPU 讲解的内容都适合 8088 CPU。两者的指令系统相同，汇编语言一致。

8087 是高速度与高精度的浮点协处理器，它可处理 32 位及 64 位的浮点数和 16 位、32 位及 64 位的整数，还可计算 18 位十进制数和直接计算初等函数。

8089 是 I/O 处理器，即专门处理输入输出的微处理器。目前它的功能由 CPU 代替，很少使用。

80130 是操作系统的固件。

80186 是 8086 的增强型，增加了可靠性及速度，指令系统中只有 8 条指令增强了功能，还增加了 10 条新指令。

80286 是 16 位微处理器，它的主频在 10－30MHz 之间。除了具有 8086 的基本结构外，它还集成了类似于段式的存储管理，寻址空间可达 16MB(兆字节)，最大可提供给用户 1000MB 的虚存空间。它的指令系统包含了 8086 的全部指令，另外还增强了 9 种指令的功能，并增加了 25 种新的指令。

80287 也是浮点协处理器，它与 80286 CPU 配套使用，只比 8087 多 1 条指令。

80386 DX 是 32 位微处理器，其主频在 20－60MHz 之间。它具有部分 8086 的基本结构，并且集成了类似于页式的存储管理。最大寻址能力为 4GB(千兆字节)，最大虚存空间可达 64TB(兆兆字节)。它的指令系统除包含了 80286 的全部指令外，还对各种类型的指令、地址及功能进行了扩充，从而具有 32 位的指令集，并且增加了三组新指令。

80386 SX 是一个内部为 32 位数据总线，外部为 16 位数据总线的微处理器，它支持 80386 DX 的所有方式及操作。最大寻址空间为 16MB，主频为 16－20MHz。

80486 也是 32 位微处理器，其主频在 33－100MHz 之间。它与 80386 微处理器的主要差别如下：

1. 80486 芯片上配置了一个 8KB 字节的高速缓存(Cache)。
2. 80486 芯片上设置了一个浮点处理部件(FPU)，从而使得 80486 微处理器的性能比 80386 微处理器高数倍。

不过两者的指令系统差别不大，80486 微处理器只比 80386 微处理器多出了 6 条指令，寻址方式与寻址空间两者也相同。

1.2.2 系统组成

iAPX86/88 处理器系列可组成许多微处理机系统。最简单的是由 iAPX86－10 或 iAPX88-10 组成的微处理机。IBM-PC 和 IBM-PC/XT 分别属于 iAPX88-10 和 iAPX88-20。它们的系统组成如图 1.1 所示。

以 8086 为 CPU 的微机中系统板内存容量为 640KB，这是系统的基本内存容量，加上扩充板的存储器容量后，整个系统内存可达到 1MB(1024×1024)。通常它的外存储器是一台 5.25 英寸软盘机和一台温彻斯特硬磁盘机。软盘存储容量为 320KB 或 360KB，硬盘存储

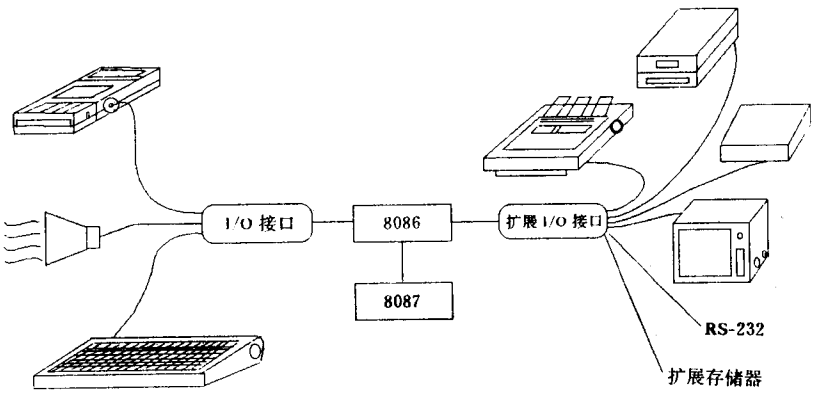


图 1.1 8086 的微机系统配置图

容量为 10MB 或 20MB。外部设备通常有键盘、彩色图形显示器及点阵打印机。

以 80286(80386)为 CPU 的微机简称 286(386)微机，其系统的配置形式上与以 8086 为 CPU 的微机配置基本一致，但是微机各外部设备的性能有了很大提高。286 微机的硬盘存储量为 40MB-100MB(80386 约为 120MB－300MB)。286 微机的软盘驱动器通常有两个：5.25 英寸高密驱动器，容量为 1.2MB；3.5 英寸高密驱动器，容量为 1.44MB。8086 微机的显示器为 CGA，最高分辨率为 320×200，最多可同时显示 4 种颜色。286 以上的微机其显示器可为 EGA 或 VGA，其最高分辨率为 640×350(EGA)或 640×480(VGA)，最多可同时显示 16 种(EGA)或 256 种(VGA 320×200 分辨率下)颜色。有的显示器如 TVGA，最高分辨率可达 1024×768(256 色)。打印机与键盘的指标也有提高。另外 286 以上的微机通常配有鼠标器。

能在这些微机上运行的软件很多，并且在不断地增加。目前运行的单任务操作系统是 MS-DOS(PC-DOS)，UCDOS，CCDOS，MS WINDOWS，OS2，多任务操作系统是 XENIX 和 UNIX，后者只能在 286 以上微机上运行。这些微机还能联入网络系统中运行。

这些微机系统能够提供使用的语言有很多种。通常的 BASIC，COBOL，FORTRAN，PASCAL，LISP 及 C 语言等都能使用，汇编语言有 MASM 和 TASM(Turbo Macro Assembler Language)等。还有 True BASIC，Turbo BASIC，Turbo C/C++，Visual BASIC 等。

从汇编语言的角度看，80x86 系列微机中，CPU 的基本指令是 8086 的指令集，协处理器的绝大部分指令是 8087 的指令集。下面我们进一步介绍这两个处理器。

1.2.3 8086 微处理器

8086 微处理器由两部分组成：一部分是指令执行单元 EU；另一部分是总线接口单元 BIU。图 1.2 为 8086 CPU 结构图。

EU(Execution Unit)主要由累加器 ALU、标志寄存器 FLAGS、通用寄存器 AX－DI 以及 EU 单元控制系统组成。EU 从指令队列中取出指令执行，要用到指令执行中存放在内存的操作数，则也由它形成操作数的地址送到 BIU，BIU 根据地址取到操作数后，送回 EU。

BIU(Bus Interface Unit)主要由指令队列、指令指针寄存器、段寄存器、地址加法器以及总线控制电路组成。它负责从存储器取指令和数据以及所有 EU 需要的总线操作，实现

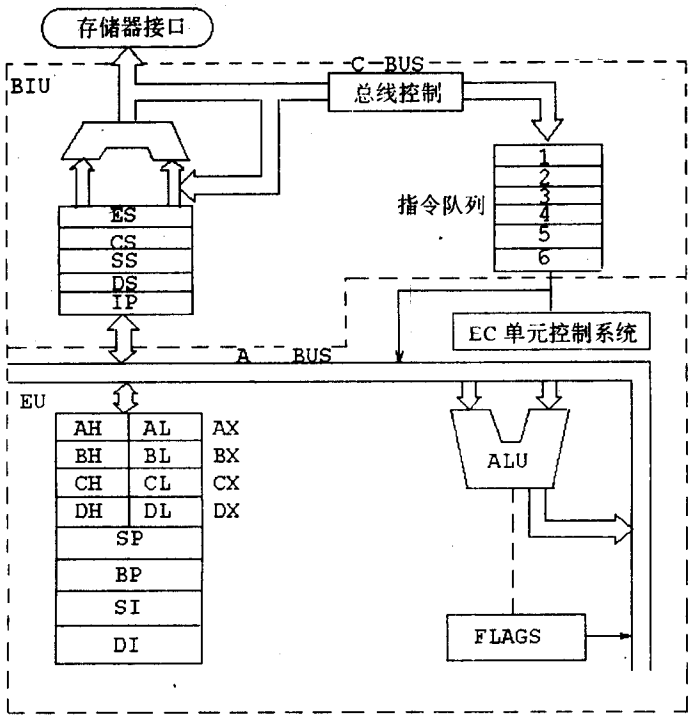


图 1.2 8086 CPU 结构图

CPU 与存储器或外部设备之间的信息传输。指令队列是能存 8 个字节指令的先进先出寄存器。EU 从队列的前面取出指令执行,BIU 往队列后面送入待执行的指令。

EU 和 BIU 能独立运行,并利用指令队列实现并行操作。只要指令队列有空,BIU 就可以从内存取出指令送入指令队列,同时 EU 不断地从指令队列中取出指令执行,从而减少了 EU 等待访问内存的时间,提高了 CPU 的速度。

8086 处理器中,可访问的内部寄存器如图 1.3 所示。

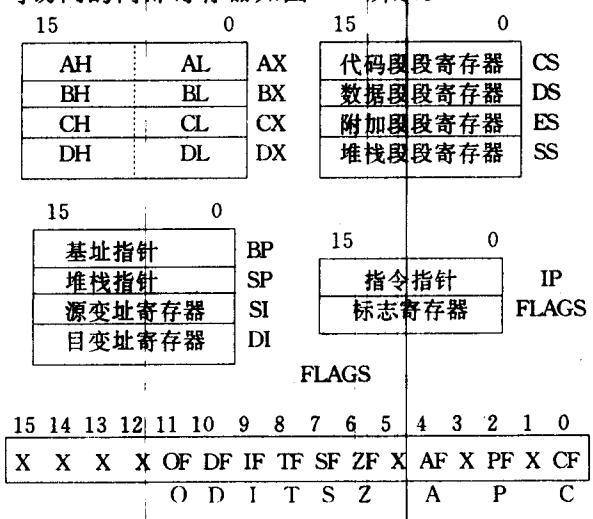


图 1.3 8086 CPU 内部寄存器

(1) 4 个 16 位通用寄存器——AX,BX,CX 及 DX。它们也可以作为 8 个 8 位的寄存器使用,分别以后缀 H 表示高 8 位,L 表示低 8 位,其寄存器名记作 AH,AL,BH,BL,CH,CL,DH 及 DL。这些寄存器主要用于存放数据和运算结果。AX 在乘除法运算中作累加器使用,它被指定存放被乘数与被除数;BX 还可存放地址,作指针寄存器使用。另外 CX 还可存放循环次数,CL 可存放移位次数,DX 可存放 I/O 端口号,这些将在具体指令中进一步讲解。

(2) 4 个 16 位指针寄存器——BP,SP,SI 及 DI。它们主要用于存放地址,其中 SP 主要作栈顶指针使用,SI 和 DI 有时作变址器使用,存放数据和运算结果。上述两类寄存器称为通用寄存器。

(3) 4 个段界寄存器(有时也称段寄存器)——CS,DS,ES 及 SS。它们专门用于存放段界地址,其中:

CS——代码段段界寄存器
DS——数据段段界寄存器
ES——附加数据段段界寄存器
SS——堆栈段段界寄存器

(4) 16 位指令指针寄存器 IP。它存放当前要执行的指令的地址,因而其内容将随指令的执行而变化。

(5) 16 位标志寄存器 FLAGS。它的内容标志着程序执行时 CPU 的状态。指令执行时由于标志寄存器的内容不同而产生不同的作用。标志寄存器的内容是随程序的执行而不断变化的。图 1.4 简要说明了标志寄存器各位的定义及作用,以后在涉及到的指令中我们将进一步讲解其作用。

O——OF 溢出标志位。运算结果超过系统所能表示的数的范围时为溢出(或上溢)。例如 1 个字节能表示的整数范围为数−128～127,1 个字能表示的整数范围为−32768～32767,超过此范围为溢出,溢出时(OF)=1。

D——DF 方向标志位。它标志字符串指令处理的方向。(DF)=0 是递增方向,(DF)=1 是递减方向。

I——IF 中断标志位。它表示系统可否接受中断(除不可屏蔽中断之外)。(IF)=0 屏蔽中断,(IF)=1 允许中断。

T——TF 单步中断标志位。当(TF)=0 时,指令连续执行,当(TF)=1 时,执行一条指令后停止,即一步一停。

S——SF 符号标志位。它表示运行结果最高位(符号位)的值。(SF)=1 结果为负,(SF)=0 结果为正。

Z——ZF 全零标志位。运算结果为全零时,(ZF)=1,否则(ZF)=0。

A——AF 辅助进位标志位。在运算中,数的低一半产生进位或借位时(AF)=1,否则(AF)=0。

P——PF 奇偶校验标志位。它用于核对 ASCII 码的奇偶性,(PF)=1 时表示运算结果中有偶数个“1”,否则(PF)=0(奇数个“1”)。

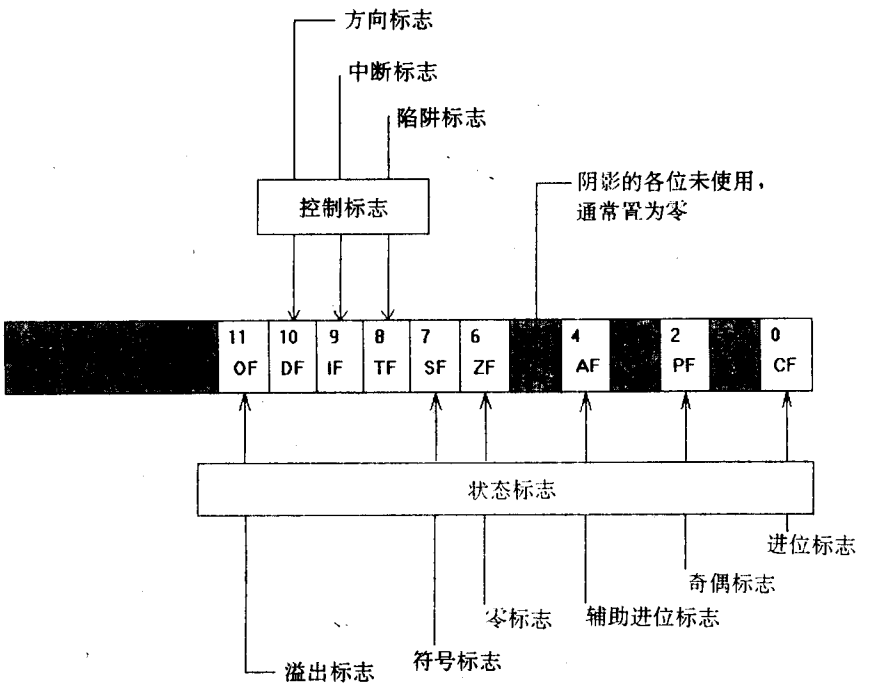


图 1.4 8086 的 FLAGS 寄存器

C——CF 进位标志位。运算结果在最高位产生进位或借位时(CF)=1(字节的最高位为 D_7，字的最高位为 D_{15})，否则(CF)=0。

DF、IF 及 TF 是控制标志位，它的内容起控制 CPU 状态的作用，其他标志位均表示指令运行结果的状态。

1.2.4 存储器及其分配方式

8086 的内存可达 1048576 字节，内存按字节编址，每个内存单元的地址包含 20 个二进位。地址以十六进制表示，从 0000 到 FFFF。字长为 16 位，字的地址最好从偶数地址开始。

1.2.4.1 物理地址与逻辑地址

物理地址是指内存的真实的地址，它有 20 位。存取数据时必须使用物理地址。但是在多道系统中，物理地址不能由用户直接指定，因为用户使用内存的情况是不可预测的。为此，引入了逻辑地址的概念。逻辑地址是用户定义的地址，一般是指系统分配给用户的空间的相对地址。

1.2.4.2 主存储器的分段结构

8086 的主存储器基本上是界地址分配方式，但没有越界检查。系统设立的 4 个段界寄存器专门用于存放界地址。

内存以块为分配单位，每段可以分配若干个连续块，块的大小可按下列公式推算：

块长 $=2^N$

其中 N 为内存地址位数减界地址寄存器位数。因 8086/88 内存地址为 20 位，而所有的段界寄存器都是 16 位的，故 N=4，所以块长为 16 字节。1MB 内存可分为 64K 块，块号从 0000 编到 FFFF。

系统根据用户的需要，把连续的若干块内存组成的段分配给用户，各段的首块号即为该段的界地址，段界寄存器中存放的就是当前段的界地址。由界地址变为相对应的该段的物理首地址，只需在其尾部加上 1 个十六进制的“0”。下面列出了 3 个界地址及与其相对应的物理地址：

界地址	00FF	0001	0006
对应物理地址	00FF0	00010	00060

我们不妨把段的界地址相对应的物理地址称为段始址。8086 中的逻辑地址就是段内相对于段始址的地址，也称为偏移地址。由于指令中使用的偏移地址最多只能是 16 位，所以段的最大长度也只能是 64KB。

1.2.4.3 逻辑地址与物理地址的转换

逻辑地址只有转换成物理地址才能访问内存，这个转换由硬件来实现，转换遵循下列公式：

逻辑地址相对应的物理地址＝当前段始址＋该逻辑地址

例如，设某程序段的段界地址为 AAAA（意味着在执行程序时 CS=AAAA），并设该段中某逻辑地址为 BBBB。要访问 BBBB，就一定要求出 BBBB 对应的物理地址。由于段始址是段界地址（CS 的内容）对应的物理地址，这里 CS=AAAA，故段始址为 AAAA0，于是：

BBBB 的物理地址＝AAAA0＋BBBB＝B665B

因而在程序中访问 BBBB（逻辑地址）时，实际上访问的是 B665B。上例中只讲了在代码段中访问指令时逻辑地址到物理地址的转换。访问数据时也有类似的转换，只是使用的段界寄存器不同，可能是 DS，ES 或 SS。

程序也须按段使用内存，程序中出现的地址（除直接段间转移指令外）一般都是逻辑地址，并且在执行时逻辑地址应与哪个段界寄存器相对应是明确的，段界寄存器的值也是确定的。

系统虽然只有 4 个段界寄存器，但程序中可以有任意个段，并且各段可以互相独立，可以不邻接，还可以互相重叠，只是当前参与运行的最多只能有 4 个段。段界寄存器的内容软件可设置，用户也可设置。

这种内存分配方式是很灵活的，同时也扩展了使用空间。但是由于内存缺乏必要的保护，用户必须正确设置段界寄存器的内容，否则后果严重。

对于内存分段结构，我们可以总结出如下几点：

1. 内存分段使用，段长≤65KB。
2. 段的类型有代码段、数据段、堆栈段和附加数据段。每类段在程序中可以有多个。
3. 系统给每个段分配一个段界地址，当前使用段的段界地址存放在相应的段界寄存器中。段始址＝段界地址尾＋0。
4. 段内使用相对于段始址的偏移地址（相对地址）。

5. 内存以块(16字节)为单位分配,不足1块的也分配1块。段始址的尾部必然为0。例如,若设段长度为124H,则系统分配的段空间为130H,占13H块。

1.2.4.4 堆栈

堆栈是一种后进先出的存储区,它常用于存放调用程序的现场及参数。有了堆栈,多重调用的层次关系(包括递归调用的层次关系)的处理就变得又明确又方便。

8086的堆栈设在内存中,并且由堆栈段界寄存器SS指示堆栈段的始址。栈指针寄存器SP指示栈顶(当前使用位置的偏移地址),栈段中的信息按从高地址到低地址的方向存放。栈中每个元素为1个字长。由于按后进先出的原则退栈,故进栈时(SP)－2→SP,退栈时(SP)＋2→SP。SP的初值指向栈最大地址元素的下一个字节的偏移地址,即初值(SP)＝堆栈长度。一个栈段最长可达64KB。超过64KB可以再定义一个栈段。程序中可以有若干个栈段。只有当前栈段的段界地址及栈顶相对位置分别由寄存器SS和SP指示。图1.5为堆栈操作示意图。

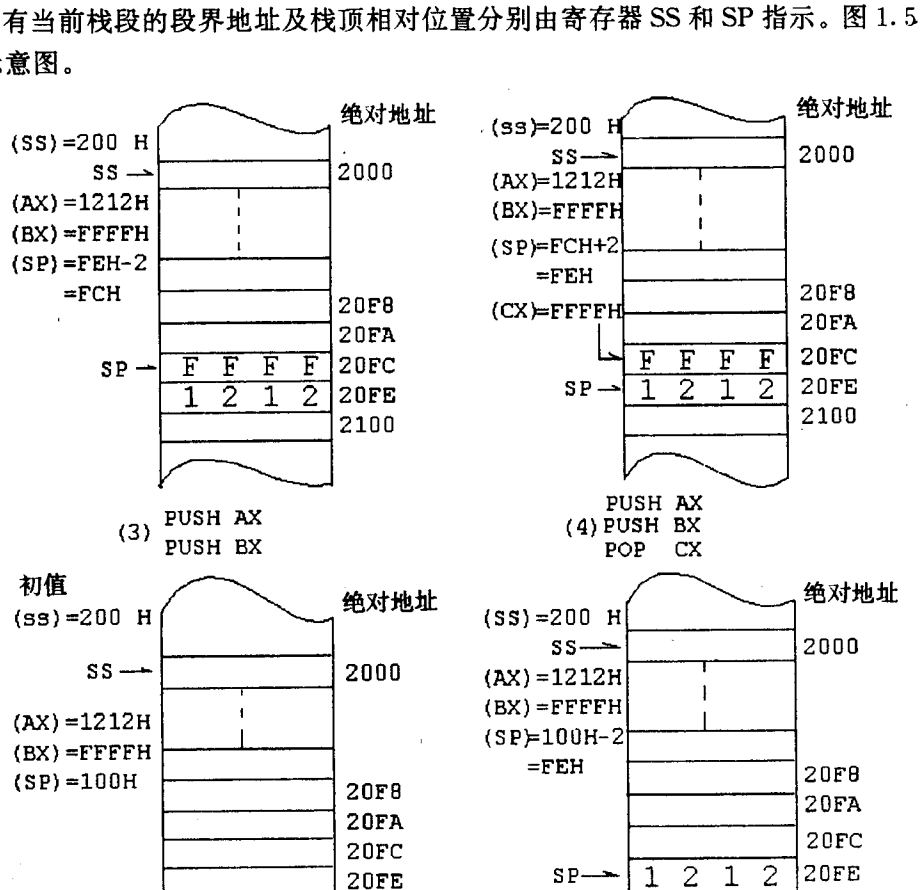


图1.5 堆栈操作示意图

1.2.5 8087 处理器

8087 是高速运算处理器，它专门用于计算浮点数和长整数，既可以计算 32 位到 64 位的浮点数及整数，也可以计算 80 位的压缩十进制数及直接计算初等函数。

8087 有 8 个浮点栈，这 8 个浮点栈由 80 位长的寄存器组成，记为 ST(N)，N＝0，1，2，...，7。它有 62 种指令，相应有 77 种汇编语句，运算绝大部分在浮点栈上进行。系统中 8087 的指令和 8086 的指令可以在同一个程序中使用。8087 的指令是不完备的，不能构成循环程序，它一定要和 8086 的指令配合使用。但 8087 的浮点运算及整数运算速度快且精度高。

练习一

1. 何谓代码指令？何谓代码程序？它们的最大优点和缺点是什么？
2. 何谓机器语言？何谓汇编语言？它们主要有何相同和不同之处？
3. 汇编程序有何作用？它和汇编语言有何关系？
4. 选择填空，将正确的答案填于下列叙述中：

   计算机只能识别和执行(A)，汇编语言源程序(B)在计算机中运行，它们必须被(C)成(D)才能在计算机中运行。

   可供选择的答案为：(1)能 (2)汇编 (3)不能 (4)机器语言
5. 简述汇编过程中两次扫描的主要功能。
6. 在 8086 微机系统中，设(DS)＝1100H，某语句(对应的代码指令)的偏移地址为 ABCDH，求该语句的物理地址。
7. 在 8086 微机中，设(CS)＝1100H，程序段的长度为 423 字节，问(1)系统分配给该程序段的空间为多少个字节？(2)绝对地址“111F”是否在分配给程序段的空间中？

1.3 数 据 表 示

1.3.1 十六进制数及 INTEL 惯例

1.3.1.1 十六进制数

十六进制数是逢 16 进 1 位，每一位有 16 种状态，用 0，1，2，...，9，A，B，C，D，E，F 表示。4 位二进制数恰好代表 1 位十六进制数。表 1.2 是十进制、二进制与十六进制数码对照表。

表 1.2 十进制、二进制与十六进制数码对照表

十进制	二进制	十六进制
0	0000	0
1	0001	1
2	0010	2
3	0011	3

（续）

十进制	二进制	十六进制
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

计算机中实际操作的是二进制数，但二进制数书写起来太长，不方便，故在打印、显示、书写时常用十六进制来表示。为了区别起见，十六进制数码后用 H 或用脚标 16 标出。如：

AB11H，　$(167)_{16}$

十六进制数和八进制数都能很方便地转化为二进制数。微处理机汇编语言调试系统的数全部用十六进制数。汇编列表文件中的代码指令及内存地址也用十六进制数表示。故十六进制数是汇编语言书写的工具，应当熟悉它。

1.3.1.2 INTEL 存数的惯例

INTEL 芯片存储数据是以一个数据长度为单位，按字节倒着存放，通称倒置，即数据的低位字节放在地址低(小)的字节中，数据的高位字节放在地址高(大)的字节中。也就是说，“低位放低字节，高位放高字节”。

例如，若寄存器 AX 的内容为“4A84H”，即(AX)＝4A84H，则用指令把 AX 的内容放到内存 NN 中将有下列结果：

内存地址	内容
NN	84
NN＋1	4A

而当指令取回 NN 的内容到 AX 中时，由于是自动倒着取回，故寄存器恢复原状，即：

(AX)＝4A84H

因而对该惯例有下列结论：

1. 如果数据的定义与使用的长度均为字节，则没有倒置问题，因为在字节内部，高低位之间不倒置。
2. 如果数据是由指令存放在内存或者是由数据语句定义在内存的，并且按原来的长度用指令读取，则倒置问题由硬件自动处理，用户不必考虑。

3. 如果用户存放与读取数据的长度不一致，则用户必须考虑倒置问题，否则数据可能会用错。

[例] 考虑下列语句的执行结果：

```
MOV AX,0102H            ;(AX)=0102H
MOV N ,AX               ;数据送内存地址 N 开始的位置
MOV AL,BYTE PTR N       ;从内存地址 N 处取回 1 个字节
```

思考题：此时(AL)=01 还是(AL)=02？如果再执行语句 MOV AX,N ;(AX)=？

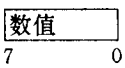
1.3.2 8086 处理的数据类型

8086 能够处理 4 种类型的数据。

1.3.2.1 无符号二进制数

1. 8 位无符号二进制数

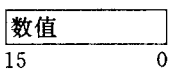
形式：



其数值范围为 0－255。

2. 16 位无符号二进制数

形式：



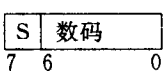
其数值范围是 0－65535。

1.3.2.2 有符号二进制数

有符号二进制数通常称为整数，其最左边的一位(最高位)为符号位，“0”表示正，“1”表示负。整数用标准的补码表示法来表示，它也有两种长度。

1. 8 位有符号二进制数

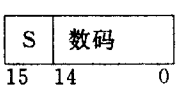
形式：



其中 S 为符号位。数值范围为－128～＋127。

2. 16 位有符号二进制数

形式：



其中 S 为符号位。数值范围为－32768～＋32767。

1.3.2.3 无符号十进制数(BCD 码)

无符号十进制数每位数用 4 位二进制数表示。显然，4 位二进制数中只有 0～9 有效。这种数又称为 BCD 码，它的存放有两种形式。

1. 无符号压缩十进制数