IBMPC计算机
系统/应用软件开发

中国科学院希望高级电脑技术公司

一九八八·一

目　录

第一章

第二章

第三章

第四章

第五章

第六章

第一章 引 言

由于IBM pc计算机应用日益广泛,用于开发系统/应用软件的语言工具也越来愈多,以前只有中小型计算机上有的,现在pc机上也有了,表1—1列举了一些常见的语言工具及其程序例子。

在系统/应用软件开发时,根据软件的性质与特点不同,开发人员的具体条件不同,应对各种因素作出折衷的选择,挑选较为合适的语言工具。

一般说来,选择语言工具,必须考虑如下的因素:

1. 对内存的要求,是否是常驻内存或ROM软件,是否对内存的使用有苛刻的要求。
2. 软件的运行速度。
3. CPU资源的利用率
4. 是否经常需要对机器各种资源进行直接控制。
5. 程序可维护性和可读性
6. 可移植性
7. 软件生产率与开发时间

实际上,这些要求并不是一致的,特别是前三个因素与后四个因素更是相互矛盾的。要使软件具有良好的可移植性,就不能过多对具体资源进行直接控制;另外,使用超高级语言或高级语言编制程序的出错率比使用低级语言的出错率来得低,因而有较高的软件生产率,因而开发得也快,但内存要求较大、CPU资源利用率低,程序运行速度也慢。因此,在考虑语言工具选择时,只能满足一个或几个最重要的要求,而兼顾别的次要要求。

对于汇编语言,在内存使用省,运行速度高,CPU的利用率高和对机器资源能直接控制方面都优于其他语言工具。因此,当某些系统/应用软件在这几方面有苛刻的要求,例如程序要常驻内存,甚至要放在ROM中时,往往使用汇编语言作为开发工具。

有时为了兼顾其他的要求,例如,对可读性和可维护性有更高的要求,要求开发时间更短,则往往使用混合编程的方法,即大部份程序使用高级语言,如采用C、Pascal、FORTRAN来编写,而关键的部份,例如,对于影响内存要求、运行速度、CPU利用率以及直接控制计算机资源等部份,使用汇编语言编写。这样,可能比较全面地实现软件开发的全面要求。

下面,我们就从这些观点出发,首先在第二章先介绍汇编语言的基本概念;接着在第三章中介绍8088的体系结构指令系统;第四章将介绍BIOS、DOS的约定与接口以及宏汇编程序设计;第五章将介绍PC系统板上接口电路的程序设计;第六章将介绍图形和打印机适配器的程序设计;第七章将介绍串行通讯的程序设计;而在第八章、第九章中分别介绍磁盘I/O控制的程序设计和8087协处理器及其程序设计,可以看出从第三章到第九章都是描述PC机资源的直接控制的程序设计方法。而在第十章中,我们将介绍高级语言的汇编了程序

的调用，例如BASIC、FORTRAN、PASCAL、C以及Prolog等高级语言调用汇编子程序的方法。

表1. PC机上部份语言工具

语言	程序例子
Prolog	Preson("Lee", 40, "Beijing")。 Preson("YAN", 50, "Gungdong")。 findall(Age, person(—, Age, —), List)。
PaSCAL	While not(ok)do Begin : end
C	while(CC=*str)! =NUL) Str++;
FORTRAN	DO 10 I=1, 100 ARRAY(I)=FLOAT(I) 10 CONTINUE
BASIC	FOR I=1 TO N ⋮ NEXT I
汇编语言	PUSH DS MOV AX 0 PUSH AX MOV AX DATA

第二章　汇编程序的基本概念

如同前一章所述的那样，汇编程序具有内存容量省、执行速度快，可以直接使用计算机全部资源等优点。因此，在这种情况下，往往选用汇编语言作为程序设计语言。

在IBM PC计算机上配了好几种汇编语言，并且它们的功能和使用方法也大致相同。其中Microsoft公司的宏汇编语言MASM（1.25版），功能比较强，开发和调试手段也比较完善，使用比较普遍。与之相应还有小汇编语言ASM，虽然它没有宏处理功能，但却与MASM相兼容。为了以后章节的学习，本章主要介绍汇编程序的基本概念。

汇编程序的基本构成

为了说明汇编程序的基本构成，下面我们列了一个简单的汇编程序。

```
; Simple Assembler Program        注介行
DATA  SEGMENT
A     DW  100
B     DW  50
C     DW  20
X     DW  ?
DATA  ENDS
CODE  SEGMEMT
      ASSUME  CS: CODE, DS: DATA
      MOV     AX, A   ; 取第一个数
      MOV     BX, B   ; 取第二个数
      ADD     AX, BX  ; 相加，AX=A+B
      MOV     BX, C   ; 取第三个数
      SUB     AX, BX  ; 相减，AX=A+B-C
      MOV     X, AX   ; 送出结果
CODE  ENDS
      END
```

在这个简单的汇编程序中，由二段程序构成，每段都是以SEGMET开始，而以ENDS结束。每段也有一个段名，第一段叫DATA，而第二段叫CODE。汇编程序一般有四种段，它们是代码段、数据段，过程段和附加段。前面的程序，DATA与DATA之间的段就是数据段，它指明数据的属性和初值，例如A定义为一个字，其初值为100；而X其属性也是字，？表示没有初值。在数据段中的程序中使用的都是指示性指令，汇编结果不产生机器指令，而起定义作用，因此有把指示性指令称为伪指令。IBM PC中的宏汇编MASM的伪指令如

表2—1所示。

表2—1 MASM的伪指令

分类	伪 指 令
符号定义伪指令	EQU, =, LABEL
数据定义伪指令	DB, DW, DD, DQ, DT, RECORD, STRUC
段定义伪指令	SEGMET, ENDS, GROUP, ASSUME, ORG
模块定义与通讯伪指令	EXTRN, PUBLIC, NAME, END
过程定义伪指令	PROC, ENDP
宏处理伪指令	MACRO, ENDM, EXITM, LOCAL, REPT, IRPC, IRP, PURGE
条件伪指令	IF, ENDIF, IF1, IF2, IFB, IFNB, ELSE, IFE, IFDIF, IFDEF, IFNDEF, IFIDN,
列表伪指令	PAGE, TITLE, SUBTTL, LIST, XLIST, %OUT
其他	COMMENT, RADIX, INCLUDE, EVEN

CODE之间的段是代码段、其中多数指令都汇编成机器相应的指令，所以这些语句都指令性语句。指令性语句的格式为：

〔标号：〕〔前缀〕指令助记符〔操作数〕〔；注释〕

其中：前面的标号：表示该语句的地址，为转移指令（JMP）、调用指令(CALL)或循环指令（LOOP）作为操作数使用。

前缀用于8088的一些特殊指令，如重复指令（REP等）、总线封锁指令(LOCK)等。这些8088，8087的指令，将在以后的章节中介绍。

使用汇编语言的环境

汇编语言使用的环境和工作流程如图2—1所示。在DOS操作系统之下，有各种编辑程序：如行编辑EDLIN和其他编辑程序Wordstar、PE等；汇编程序MASM或ASM，联结程

序LINK，动态调试程序DEBUG，库管理程序LIB和交叉参照列表程序CREF等。

首先使用各种编辑程序，编辑汇编语言源程序，建立源程序文件（扩展名为．ASM）；然后调用MASM或ASM对汇编源程序进行汇编，在没有语法错误的情况下，得到可重定位的目标代码文件（扩展名为．OBJ），同时得到列表文件（．LST）、交叉参照文件(.CRF)；利用LINK程序将OBJ文件，库文件相联结，便可得到可执行文件(扩展名为．EXE）。在DOS系统控制下运行可执行文件，如果有错，可用动态调试程序DEBUG进行调试，找到错误后，使用编辑程序修改源程序，并再次汇编、联结、调试直到正确为止。

另外，利用库管理程序LIB对现有的库文和用户OBJ文件生成新的库文件。利用CREF对交叉参照文件进行处理，得到定义各种符号的源程序的行号以及引用各种符号的源程序行号的清单，供程序调试用。

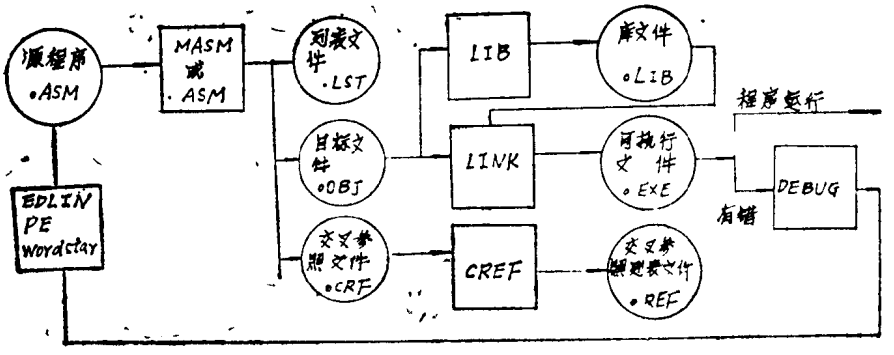

图2—1　汇编语言的工作环境与工作流程

汇编语言中的数与变量

在汇编源程序中，经常用到常数、它们可以表示成各种不同的数制。这就是：二进制、八进制、十进制、十六进制、字符串、十进制实数和十六进制实数等。它们如表2—2所示。

表2—2　常量的各种格式

数制	格式	X取值范围	例	注
二进制	××××B	0—1	010110B	
八进制	×××O ×××Q	0—F	F64O 327Q	
十进制	××× ×××D	0—9	6F94 48F9D	
十六进制	××××H	0—F	0FAEH	第一个符号一定为0
字符串	'××' "××"	ASCⅡ字符	'AB' "CD"	
十进制实数	××.××E±××	0—9	25.43E+5	
十六进制实数	×××R	0—F	74F76R	第一个数必须为0—9

在汇编语言中，变量是由伪指令DB、DW等定义，一般格式为：

〔〈变量名〉〕〈数据区定义伪指令〉〈表达式〉

在表2—3中，给出了由数据区伪指令所确定的变量类型，字长和说明性的例子。

表2—3　由伪指令确定的变量类型

伪指令	数据项数型	数据存取单位	例　　子
DB	BYTE	1字节	MSG DB 'MSG '5, 'T' 表示MSG共五字节，有初值
DW	WORD	2字节	SIZE DW 4米128 表示SIZE为一个字、值为512
DD	DWORD	4字节	PTR DD MSG 表示指针PTR初值为MSG的地址表达式
DQ	QWORD	8字节	REAL DQ 3.141597 REAL是长实数、初值为3.1415977
DT	TBYTE	10字节	DEC DT 273457891 表示 DEC存放紧凑十进数初值为273457891

除了这几种基本的数据类型外，还有二种复合的数据类型，这就是记录和结构。

伪指令RECORD可以用来设计一个记录，其格式为：

〈记录名称〉 RECORD 字段1，字段2，…

其中字段的格式为：

字段名称：宽度〔=初值表达式〕

例如：RED RECORD FL1：8='B'，FL2：4=3表示记录名称为RED，它有二个字段FL1和FL2，FL1宽度为8个二进制位，即8个bit，其初值为字符B；字段FL2 宽度为4个bit，初值为3。

伪指令STRUC可用来设计一个构造，其中各个字段不象记录那样，没有什么限制（记录中的字段只能是BYTE或Word类型）。构造的定义格式为：

〈结构名称〉STRUC

〈字段名称〉

⋮ 〈数据区定义伪指令〉〈初值表达式〉

〈字段名称〉

〈结构名称〉ENDS

例如，下面是名为STR的结构的定义：

```
STR    STRUC
FL1    DB    1，2，3
FL2    DB    〃XYZW〃
STR    ENDS
```

这里，字段FL1是3个字节，初值为1，2，3而字段FL2是4个字节，初值为"XYZW"。

第三章　8080的体系结构与指令系统

在这一章，我们先看一看8088芯片的内部组分和存储器的寻址方法，然后学习8088的指令集并学习如何用它。这些概念构成了8088的体系结构。

从图3—1可看出8088CPU（中央处理器）的框图。CPU和主存之间的数据传送是经过存储器接口的，进来的数据，它是指定解释成指令的，是放到指令串字节队列中。执行部件控制系统从这个队列中取出指令并把它送到要解释和执行它的执行部件中。正在做这样事时，总线接口部件试图再装满指令串字节队列。这种设计被称为流水线，而且尽量保证每次CPU一旦结束正在执行的指令，就有另一条指令等待执行。大多数微处理器，执行一条指令之后必须等到从存储器读出下一条指令为止。因为8088有流水线，所以能更快地执行指令。

执行部件里包括了算术／逻辑部件，或叫ALU。这个组分担负着所有算术运算和比较。为了产生一个数字结果，ALU经常置各种标志来指示结果的各种状态。例如，一个计算结果为—5，则符号标志应被置成能指示该结果是负的。

8088的寄存器组

图3—2中给出了8088的寄存器组。从程序设计者的观点看，在微处理器里这些寄存器是最重要的组分。当我们开发自己的汇编语言程序时，我们会经常使用和引用8088的各种寄存器。

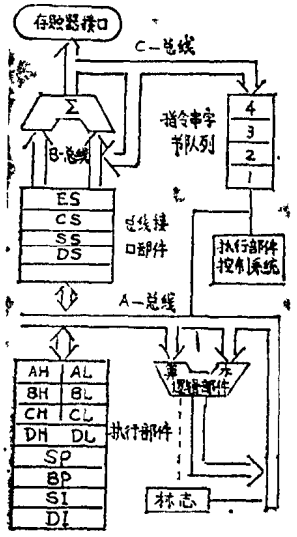

图3—1　8088的框图

有四个通用寄存器，每一个都是16比特宽，叫做AX，BX，CX，和DX。这四个中的每一个都能作为一对8比特（字节）寄存器被访问。这一对的左边（最高有效）字节为“高”字节，而右边（最低有效）字节为“低”节字。这样，寄存器AX可以被分为寄存器AH（“A高”）和AL（“A低”）；寄存器BX也可分为BH和BL；等等。

有两个串变址寄存器，都是16比特宽度。分别叫SI（“源变址”）和DI（“目的变址”）并通常用来指出存储器中的数据串。它们也能作为通用16比特寄存器。

SP是堆栈指针寄存器，用来实现存储电的硬件堆栈（下面有堆栈用法的简短说明）。SP由基址指针寄存器BP补充，而BP也可作为一个16比特通用寄存器。

四个段寄存器是专用寄存器。每当8088访问存储器时总是用到段寄存器。因此，每当执行任一指令时，微处理器总是隐含地访问一个或多个段寄存器。

另一个专用寄存器是指令指针寄存器IP，它指向要执行的下一条指令。最后是标志寄存器，它包括说明微处理器当前状态的若干个一位标志。

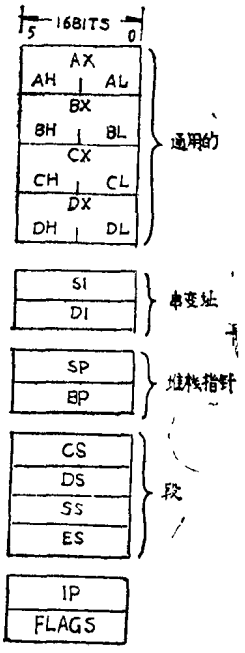

图3—2　8088号的存器组

存储器地址程序分段

传统上，微处理器总是用16比特地址字访问主存里的数据。因为地址是16比特长，最大地址数是65,536、或64K，为使微计算机变得更加完善、它们需要增加存贮器。对于8088，我们仍然用16比特地址字，但整个存贮器容量，以存贮器地址分段化方法，增加到1兆字节(1024K)。

要编址1兆字节，我们必须能表示0～1,048，576范围的数，这需要20个比特。8088是用20比特地址字来编制实际存贮器的。通常把完整的一组1，048，576个不同地址称为1兆字节地址空间。然而，从我们的角度，存储器是按块编址的，这个块叫做段。每一段能包括多至64K的数据。这允许我们在一个段内用标准的16比特编址，正如图3—3中所示

一个段可以在1兆字节地址空间内的每一16字节边界开始（叫一个节）。图3—4中可以看到这一点。注意：有64K个不同的段开始地址；在每一个段开始地址中低4个比特是零。考虑到这些段不一定包这括64K的数据，这只是最大值。一个段可以包括一个，一百个，或者一千个字节。另外，段是可以重迭的；就是说，一个给定的数据字节可从一个以上的段开始地址访问它。

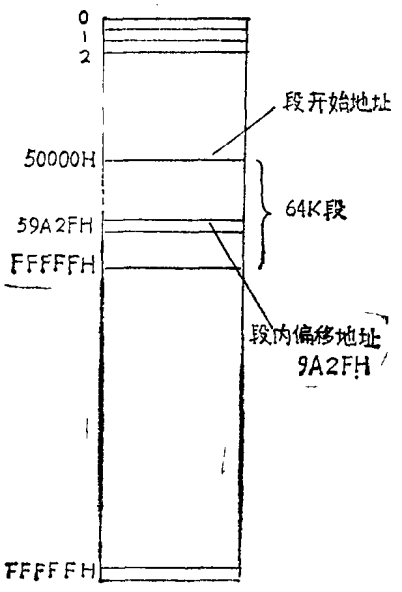

图3—3　在1兆地址空间中的一个64K段

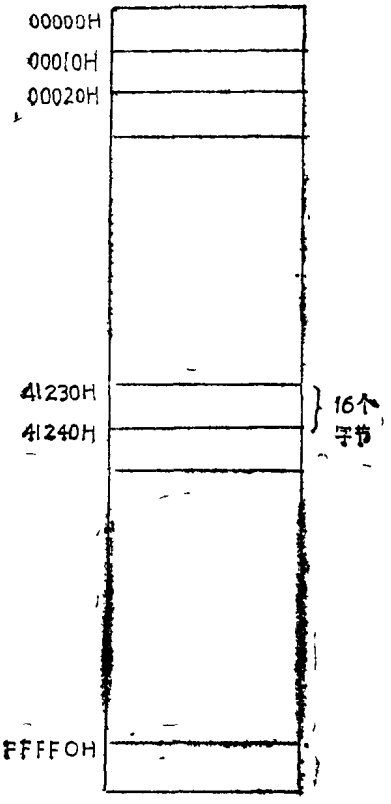

图3--4　可能的段开始地址

8088如何使用这分段方法？见图3—5。8088每当访问存储器时总得选四个段寄存器中的一个，用作段开始地址。这个值被左移4比特，然后加上偏移地址，以构成20个比特的物理地址，这是个实际用来访问存贮器的完整的物理地址。

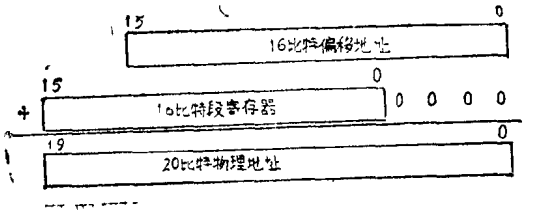

图3—5　物理地址的计算

	15 0	
CS	CODE SEGMENT	程序指令
DS	DATA SEGMENT	公用数据和源串
SS	STACKSEGMENT	栈操作用BP当作基址寄存器
ES	EXTRASEGMENT	目的串

图3—6　8088的段寄存器

图3—6 给出了四个段寄存器并指出处理器何时选用它们来做地址计算。CS寄存器总是定义当前代码段，这一段包括处理器要执行的那些指令。前面，我们已提到16比特IP寄存器是指向要执行的那条指令。实际上，处理器取指令时的做法是，从IP里找出的偏移地址和从CS找出的段开始地址相结合。

DS寄存器定义数据段的开始地址；将用于通用数据访问和源操作数的串操作指令。SS寄存器定义堆栈段的开始地址，用于所有的堆栈操作。最后，ES寄存器定义附加段.可用于第二个通用数据区。附加段也可为目的操作数的串操作指令提供指针。

这些段寄存器允许8088的程序设计员能同时访问多至四个的不同地址空间，其中的每一个都能包括多至64K字节的数据。图3—7给出如何配置段寄存器的例子。在图3—7A中，把段寄存器置得能同时地访问最大的存储器量。按着这个配置，我们能有64K字节的指令，64K字节的堆栈空间，以及两个64K的数据空间。图3—7 B给出一个更实际一些的配置。这里，我们在代码段中有一个8K（2000H）的程序，这个程序涉及到数据段里的2K（800H）数据并用256个字节的堆栈。由于数据存储的需要是很小的，所以另一个附加段就没有必要了。因此ES寄存器置得使附加段重迭，与数据段一致。

由于段寄存器在程序控制下能读和写，设计出更精益求精的配置是完全可能的，其中的段位置及其大小可以动态地变化。

8088的指令集

我们早已提到，把微处理器能实现的一套完整的原始功能称为它的指令集。8088指令集由六类指令组成，图3—3里概括了它。成功的8088汇编语言程序设计要求要懂得所有这些指令类型，而这就是我们下一步的目的。关于每一条特定指令的更详细的资料请参看附录A。

数据传输指令允许我们把数据从一个点传输到另一点，一般地，数据每一次可以传送一个字节或一个字。传送数据，看起来是个很简单的概念，但情况是复杂的。因为为寻址要传送的数据可以用很多不同的方式。这些不同的数据寻址方式也用于算术的和逻辑的的指令类型，因此现在我们要研究它们。

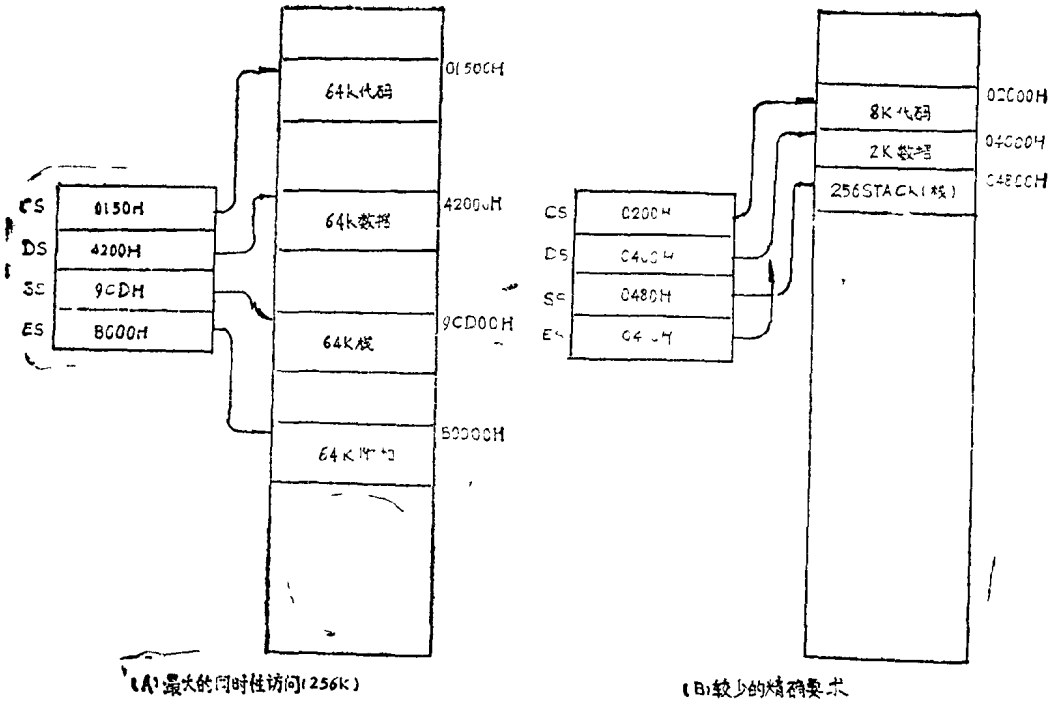

图3—7　典型的段寄存器配置

1.	数据传输指令	
	MOV	传送
	PUSH, POP	栈操作
	XCHG	交换
	IN, OUT	I/O口
	XLATB	转换
2.	算术运算指令	
	ADD	加
	INC	递增
	SUB	减
	DEC	递减
	NEG	取补
	CMP	比较
	MUL	乘
	DIV	除
3.	逻辑指令	
	NOT	取反
	AND, OR	逻辑乘，逻辑加

	XOR	异或
	TEST	位测试
	SIIL, SHR	左/右移
	ROL, ROR	左/右环移
4、	串操作指令	
	MOVS	串传送
	CMPS	串比较
	SCAS	串扫描
	LODS	串装入
	STOS	串存储
5、	控制转移指令	
	CALL	连到子程序（调用子程序）
	RET	从子程序返回
	JMP	无条件转移
	JZ、JNZ	条件转移
	LOOP	迭代（循环）
	LOOPNE	条件迭代（条件循环）
	INT	中断
	IRET	从中断返回
6、	处理器控制指令	
	CLC STC	清/置标志
	HLT	停处理器

图3—8 8088指令集

数据寻址方式

图3—9 给出了一些指令，其中的每一条都是选一个字的数据并把它传送给AX寄存器。这些指令说明了8088的各种寻址方法。第一个例子是个简单的寄存器对寄存器的传输。把BX寄存器的内容传送到AX寄存器。第二个例子给出立即寻址方式，数据传送到AX寄存器，在指令本身里面指定数据5作为常数值。第三个例子是一个直接存储器寻址，以符号"VALUE1"指的内存地址的内容送到AX寄存器。

其余方式的功能更强，如允许间接存储器寻址。在第四个例子，把BX寄存器的内容解释成存储器地址，并把该地址的内容送到AX寄存器。在第五个例中，BX寄存器保持的也是存储器地址，但在这里还要加上位移量才能定出要找的存储器地址。在间接方式中可以用寄存器器BX、BP、SI，或DI中的任一个。最后一个例子是最复杂的：BX的内容、SI的内容和位移量5都加到一起才确定包含所要数据的那个存储器地址。

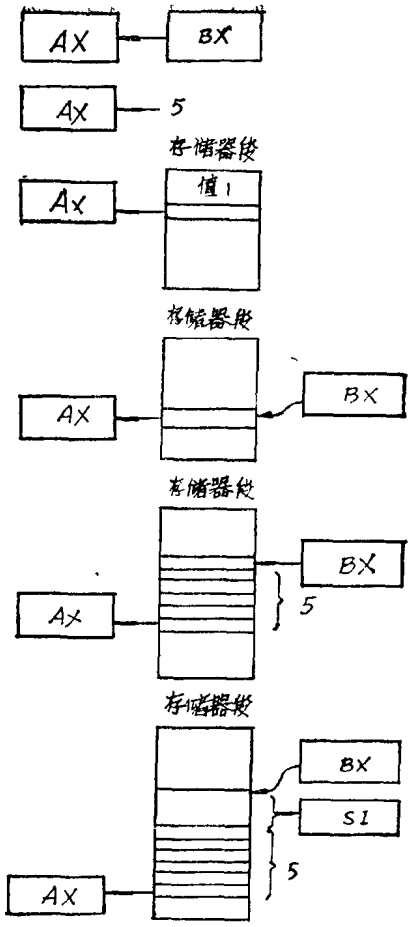

图3—9　数据寻址方式

除了立即方式，其余的每一个数据寻址方式都可用于数据传输、算术和逻辑类型指令的目的操作数。例如指令：

ADD　(BX)，DX

将DX寄存器的内容加到包括在BX寄存器里的地址的内容，并把结果返回放到同一个存储单元中。

对这些情况，几乎都是：当寻址方式指定一个存储器的操作数时，处理器就已经假定它是在数据段里并选DS寄存器来计算其物理地址（图3—5）。只是当BP寄存器用作基址寄存器时是例外；在这种场合，访问堆栈段要经过SS寄存器。假定我们要访问的数据是在一个不同的段里，该怎么办？这可对指令加一个段代用码前缀来解决。我们能指示汇编语言，要用一个段代用码，它表示所想要的段寄存器，后面跟着个冒号（：）。这样，指令

MOV AX, ES:〔BX〕

将产生的物理地址是用ES段寄存器和作为位移地址的BX寄存器的内容算出来的。把这个地址的一个字送到AX寄存器，当然这地址是在附加段内。

堆 栈 操 作

压进和弹出指令是个特殊的数据传输指令，它实现称为堆栈的“后进，先出”存储器结构。在图3—10里给出了这些指令的操作。堆栈总是驻留在堆栈段里，因此访问堆栈期间总用SS寄存器，因此不会有段代用码。堆栈指针寄存器SP是作为一个存储器地址隐含地用在所有堆栈操作中。压进（PUSH）指令把SP内容减2，然后把它的操作数存放在SP指定的存储器地址中。弹出（POP）指令从SP指定的存储器地址中取数据并作为它的操作数，然后SP的内容加2。注意PUSH和POP指令总是传输一个字的数据。这与一般MOV指令大不一样。后者能传送一个字节或一个字。

PUSH和POP指令提供一种存储机构，它能用于保存若干个不同的数据值，然后以与保存数据值时相反的顺序把它们收回。弹出该堆栈的第一个可用的值总是那个最后压进堆栈的值。因为这个特点，当我们不希望给作业指定一个特定存储单元时，要暂时保存数据值堆栈是理想的。例如，假定我们正在写一程序，而它的部分中使用和改变CX寄存器的值。如果在该程序的后面我们需要再用这个值，我们可以在堆栈里保存它，如：

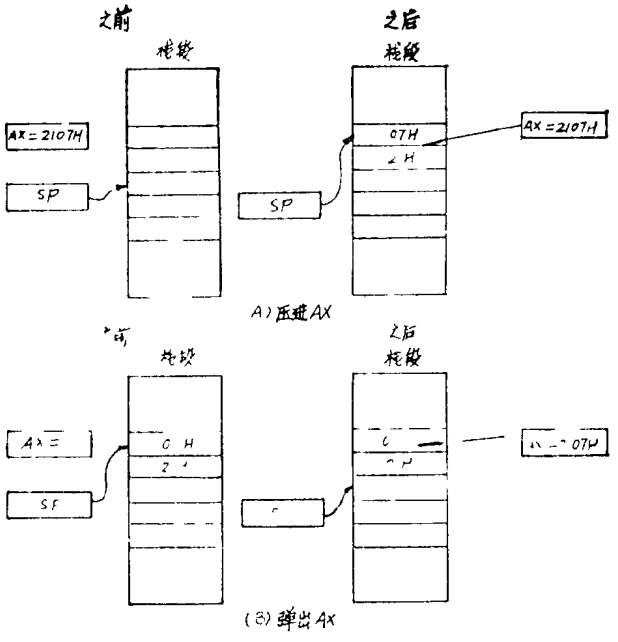

图3－10 堆栈操作