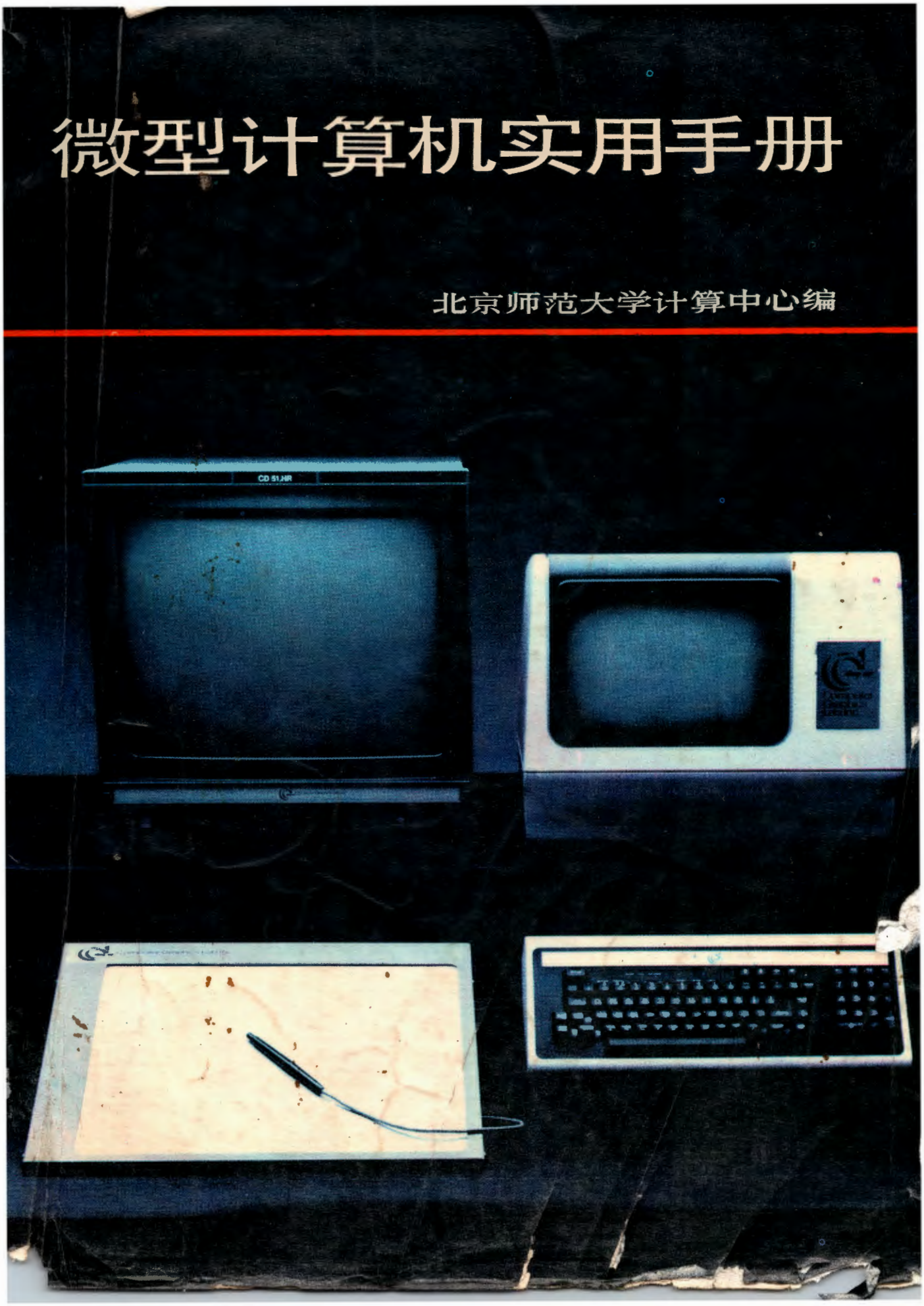
微型计算机实用手册

北京师范大学计算中心编



封面设计：田　苗

微型计算机实用手册
北京师范大学计算中心编
*
北京师范大学出版社出版
新华书店北京发行所发行
国营五二三厂印刷
*
787×1092　1/16　印张28.5　字数：704千
1985年10月第1版　1985年10月第1次印刷
印数1—147,000
科目：97—65
统一书号：13243·89　定价：5.25元

内 容 简 介

本手册包括三篇内容：第一篇概括介绍了计算机的基本知识、微型计算机的结构及常用器件的特性；第二篇归纳了APPLE-Ⅱ，CROMEMCO和IBM-PC上所配BASIC、PASCAL、FORTRAN和COBOL语言的语法规则和使用方法；第三篇详细介绍了APPLE-Ⅱ CROMEMCO和IBM-PC等微型计算机的使用方法，包括上机过程，如何在机器上建立和调试BASIC、PASCAL、FORTRAN、COBOL源程序，各种操作命令的使用方法。本手册资料先进、取材丰富、实用性强。

微型计算机实用手册

北京师范大学计算中心　编

北京师范大学出版社

微型计算机实用手册

北京师范大学计算中心　编

北京师范大学出版社

前　言

自七十年代初期微型计算机诞生以来，微型计算机以它的体积小，功耗低、工作可靠和价格便宜等优点，使计算机的应用发生了突破性变化。目前在我国，微型计算机技术已在工业交通、农业、国防、教育、科研等领域发挥越来越重要的作用。它已经成为新技术革命的重要手段。为了使更多的人掌握使用微型计算机的方法，更充分开拓微型机的应用领域，根据国内情况，我们编撰了这本“微型计算机实用手册”。

全手册分为三篇。

第一篇，微型计算机基础。这一篇主要介绍了计算机的基础知识：微型计算机结构，常用8位、16位微处理器内部结构和芯片引腿说明；存贮器的结构；输入输出接口以及中断系统和DMA传输等概念；几种常用微处理器的寻址方式和指令系统。

第二篇，程序设计语言。这一篇分四章，分别系统地介绍了APPLE II、IBM PC和CROMEMCO三种微型机上常用的BASIC、FORTRAN、PASCAL和COBOL四种程序设计语言，说明了这四种语言各语句的语法规则和使用方法。叙述中采取以某一种机型为主（PASCAL只结合IBM PC叙述），兼顾它种机器的相同和不同点。

第三篇，微型计算机的使用。这一篇根据实际需要，分四章详细说明了四种常用语言在APPLE II、IBM PC和CROMEMCO机上的使用方法，即如何在这些微型机上建立和运行四种语言编写的源程序。其中，第一章介绍了BASIC程序在这三种机器上建立，修改和运行的方法；第二、第三章分别按机型的不同，分别介绍了FORTRAN、PASCAL和COBOL三种语言编写的源程序的建立、编译以及连接目标代码形成可执行程序的过程。每章末尾附有编译、连接和运行时的错误信息表，供用户校正错误时参考。

本手册突出了实用性。第一篇介绍的是微型机的硬件构成，这一篇没有针对某一类型的机器作深入具体的讨论，只是作了概括的描述。第二篇和第三篇对四种语言和三种机器的使用作了系统而具体的说明，因为这三种微型机在国内使用的比较普遍，而这四种语言又是微型机的常用语言。因此，所编内容实为使用者所必需。

本书第一篇、第二篇第四章由师书恩同志执笔；第二篇第一章由王宝玲同志执笔；第二篇第二、三章由林钧礼同志执笔；第三篇由于久威同志执笔。袁淑君同志审阅了全书并提出了修改意见。

由于编写时间仓促，更限于编者水平，书中错误和不妥之处敬请读者不吝批评指正。

编者

1985年3月

目　录

第一篇　微型计算机基础

第二篇 程序设计语言

第三篇 微型计算机的使用

第一篇　微型计算机基础

第一章　基础知识

自从1946年美国研制成功第一台电子计算机以来，计算机的发展大体上经历了四个发展阶段：第一代，以电子管为主要逻辑元件（1946年～1957年）；第二代，以晶体管为主要逻辑元件（1958年～1964年）；第三代，以中、小规模集成电路为逻辑元件（1965年～1970年）；1970年以后，随着大规模集成电路的迅速发展，电子计算机进入了第四代。近几年来，工业发达的美、日等国，都组织人力和财力积极研制第五代电子计算机。

随着计算机应用的日益广泛，以及大规模集成电路技术的飞速发展，七十年代初诞生了一代新型的电子计算机——微型计算机。1971年，美国INTEL公司研制成功INTEL 4004 微处理器，这是一种4位并行中央处理单元（CPU），如果再配上相应的读/写存贮器(RAM)、只读存贮器（ROM）和输入输出（I/O）接口等芯片，即可构成一个微型计算机。同年，该公司还研制出了名为INTEL 8008 的8位并行微处理器。

从1971年到1980年，微型机共经历了三代，预计1985年可进入第四代，平均三年左右换代一次，其发展速度是前所未有的。

微型计算机的出现对计算机应用的推广和普及发生着深刻的影响。由于微型计算机利用了大规模和超大规模集成电路技术，因此它大大缩小了计算机的体积，提高了计算机的可靠性，同时也显著地降低了成本。再加上它的功耗低、适应性强等特点，使得计算机的应用深入到了工农业生产、国防、科技、文教，以及日常生活等各个领域。

1—1　计算机中的数

计算机的最基本功能是进行数值计算和数据加工处理。数在机器中是以器件的物理状态来表示的，为了方便和可靠，通常在计算中采用二进数制系统。即是说，把要机器处理和存贮的数、字母、符号等都用二进制数表示。计算机也只认得二进制数。

1-1-1　十进制数　十进制数是其数值用0、1、2、…8、9等这十个数字符号表示的数。我们把这些数字符号叫做数码，数码处于不同的位置（或数位）代表的量也不同，即在一串数字中间，每一个数字表示的量不仅决定于数字本身，还取决于它所在的位置。例如，整数3258，右起第一位是个位，表示它本身的数值；第二位是十位，表示5×10；第三位是百位，表示2×100；…。这个“个、十、百、千、万…”称为数的位权。

一般地，如果用α表示十个数码中的任意一个（由表示的数N决定），那么一个有n位整数和m位小数的十进制数N可表示成为：

$$N=\alpha_{n-1}\times10^{n-1}+\alpha_{n-2}\times10^{n-2}+\cdots+\alpha_0\times10^0+\alpha_{-1}\times10^{-1}+\cdots+\alpha_{-m}\times10^{-m}$$

或者缩写成

$$N=\sum_{i=n-1}^{-m}\alpha_i\times10^i$$

其中i表示数码的位置，10是基数。

1-1-2 二进制数 以2为基数的计数制叫做二进位计数制，简称二进制。它只用两个数码表示数，即0和1。进位是逢二进一。对于一个有n位整数和m位小数的二进制数，它的表示式是

$$N=\alpha_{n-1}\times 2^{n-1}+\alpha_{n-2}\times 2^{n-2}+\cdots+\alpha_0\times 2^0+\alpha_{-1}\times 2^{-1}+\cdots+\alpha_{-m}\times 2^{-m}$$

或缩写为

$$N=\sum_{i=n-1}^{-m}\alpha_i\times 2^i$$

其中α_i只能取1或0。

1-1-3 八进制数 八进制是包含0—7等八个数码的一种数制，计算时逢8进1。对于任意一个八进数N可表示为：

$$N=\alpha_{n-1}\times 8^{n-1}+\alpha_{n-2}\times 8^{n-2}+\cdots+\alpha_0\times 8^0+\alpha_{-1}\times 8^{-1}+\cdots+\alpha_{-m}\times 8^{-m}$$

$$=\sum_{i=n-1}^{-m}\alpha_i\times 8^i$$

其中，α_i由N值决定，可取0—7中的一个数码；n为小数点左边的位数，m为小数点右边的位数；8为基数。

1-1-4 十六进制数 十六进制是一种特殊的数制，它用数码0—9以及A、B、C、D、E、F表示数值。其基数为16，计数时逢16进1。一个任意的16进制数N可表示为

$$N=\alpha_{n-1}\times 16^{n-1}+\alpha_{n-2}\times 16^{n-2}+\cdots+\alpha_0\times 16^0+\alpha_{-1}\times 16^{-1}+\cdots+\alpha_{-m}\times 16^{-m}$$

$$=\sum_{i=n-1}^{-m}\alpha_i\times 16^i$$

其中α_i可取数码0—F之间的一个。

为了区别数制，可在数的右下角注明。例如1011_2、32_8、43_{10} 76_{16}的右下角的数字分别表示为二、八、十、十六进制数。有时在数的后面加H，以表示十六进制数。在数制明确的情况下，可不写这些标记，例如十进制数。各种数制的对照表如表1-1所列。

表1-1　　各种数制对照表

十进制	十六进制	八进制	二进制	十进制	十六进制	八进制	二进制
0	0	0	0000	8	8	10	1000
1	1	1	0001	9	9	11	1001
2	2	2	0010	10	A	12	1010
3	3	3	0011	11	B	13	1011
4	4	4	0100	12	C	14	1100
5	5	5	0101	13	D	15	1101
6	6	6	0110	14	E	16	1110
7	7	7	0111	15	F	17	1111

1-1-5 十进制数和二进制数的相互转换

一、十进制数转换为二进制数

对于既含有整数又含有小数的十进制数，需把整数部分和小数部分分别换算，然后把整数部分和小数部分加起来，即是转换后的二进制数。

整数部分的换算是采取连续除以2记录其余数的方法；分数部分的换算是采用连续乘以2记录其积中整数的方法。例如把十进制数59.625转换成二进制数。

整部部分　2|59　　余数

2|29　…$k_0=1$　最低位

2|14　…$k_1=1$

2|7　…$k_2=0$

2|3　…$k_3=1$

2|1　…$k_4=1$

0　…$k_5=1$　最高位

分数部分　　乘积的整数部分　0.625 × 2

最高位　$k_{-1}=1$←…　0.250 × 2

$k_{-2}=0$←…　0.500 × 2

最低位　$k_{-3}=1$←…　0.000

所以，$(59.625)_{10}=(111011.101)_2$

二、二进制数转换为十进制数

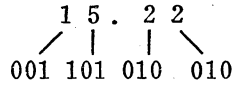
根据二进制数的定义，将它按权展开相加，即可得到等值的十进制数。例如：

$$(111.101)_2=1\times2^2+1\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-3}$$
$$=(7.625)_{10}$$

1-1-6　二进制数和八进制数的转换

一、八进制数转换为二进制数

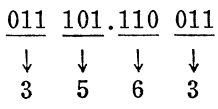
每位八进制数用三位二进制数表示即可。例如：



即　$(15.22)_8=(1101.01001)_2$

二、二进制数转换为八进制数

对于既有整数又有小数的二进制数，以小数点为界，整数部分从右向左，以三位为一组，不足三位时，在左边添0补足三位；小数部分从左向右，以三位为一组，不足三位时，在右边添0补足三位。然后把每组的三位二进制数用相应的八进制表示，即得八进制数。例如：



即　$(11101.110011)_2=(35.63)_8$

1-1-7　十迸制数和八进制数的转换

一、八进制数转换成十进制数

同把二进制数转换成十进制数的方法类似，将八进制数按权展开，然后相加，即可得到相应的十进制数。

二、十进制数转换为八进制数

将十进制数的整数部分和小数部分分别进行转换。整数部分不断用8去除，将每次所得的余数，按最后一次得到的余数为最高位，第一次得到的余数为最低位的顺序，依次把余数排列起来，即得到相应的八进制整数；小数部分可反复乘以基数8，把每次乘积的整数部分取出，最先得到的是八进制小数的最高位，最后得到的是最低位。

1-1-8 二进制数和十六进制数的转换

一、十六进制数转换为二进制数

不论是十六进制的整数或小数，只要把每一位十六进制数用相应的四位二进制数代替，即可转换为二进制数。

二、二进制数转换为十六进制数

二进制数的整数部分由小数点向左，每四位为一组，最后不足四位时，左面补0；小数部分由小数点向右，每四位一组，最后不足四位的后面补0。然后把每组二进制数用相应的十六进制数代替，即可转换为十六进制数。

1-2 带符号数的表示法

在计算机中，为了把减法运算变为加法运算，机器中的负数可用原码、反码和补码三种方法表示。正数的反码、补码与原码相同。

1-2-1 原码 用原码表示数时，符号位用0表示正数，用1表示负数，其余各位表示数本身。

表1—2 BCD编码表

十进制数	8421 BCD码
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000
11	0001 0001
12	0001 0010
13	0001 0011
14	0001 0100
15	0001 0101

1-2-2 反码 设二进制数$X=+0.X_1X_2\cdots X_n$或$X=-0.X_1X_2\cdots X_n$。并用$\bar{X}_i$表示X_i的求反，则X的反码定义为：

$$[X]_{反}=\begin{cases}X & \text{当}1>x\geqslant 0\\ 1.\bar{X}_1\bar{X}_2\cdots\bar{X}_n & \text{当}0\geqslant X>-1\end{cases}$$

1-2-3 补码 正数的补码与原码相同；负数的补码是该数的反码加1。

1-3 二—十进制码

用一组二进制数码表示一位十进制数，用若干组二进制数码表示一个十进制数的方法，称为十进制数的二进制编码，或称为二—十编码。一位十进制数可用四位二进制数码来表示，其方法极多，较常用的是8421 BCD码。表1-2列出了BCD码的编码关系。

1-4 ASCII码

在计算机中，数字、字母和各种符号都

按特定的规则用二进制编码表示，在微型计算机中最普遍的是采用 ASCII 码（American Standard Code for Information Interchange美国标准信息交换码）。编码表如表1-3所列。

表1-3　　ASCII码表

字符 \ 位654 / 位3210		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	、	*p*
1	0001	SOH	DC1	!	1	A	Q	*a*	*q*
2	0010	STX	DC2	"	2	B	R	*b*	*r*
3	0011	ETX	DC3	#	3	C	S	*c*	*s*
4	0100	EOT	DC4	$	4	D	T	*d*	*t*
5	0101	ENQ	NAK	%	5	E	U	*e*	*u*
6	0110	ACK	SYN	&	6	F	V	*f*	*v*
7	0111	BEL	ETB	,	7	G	W	*g*	*w*
8	1000	BS	CAN	(	8	H	X	*h*	*x*
9	1001	HT	EM	)	9	I	Y	*i*	*y*
A	1010	LF	SUB	*	:	J	Z	*j*	*z*
B	1011	VT	ESC	+	;	K	[	*k*	{
C	1100	FF	FS	,	<	L	\	*l*	¦
D	1101	CR	GS	-	=	M	]	*m*	}
E	1110	SO	RS	.	>	N	∧	*n*	~
F	1111	SI	US	/	?	O	—	*o*	DEL

其中：

NUL	空	VT	纵向列表
SOH	标题开始	FF	走纸控制
STX	正文开始	CR	回车
ETX	正文结束	SO	移位输出
EOT	传输结束	SI	移位输入
ENQ	询问	SP	空格
ACK	承认	DLE	数据链换码
BEL	报警	DC1	设备控制1
BS	退格	DC2	设备控制2
HT	横向列表	DC3	设备控制3
LF	换行	DC4	设备控制4
SYN	空转同步	NAK	否定

ETB	信息组传送结束	FS	文字分隔符
CAN	作废	GS	组分隔符
EM	纸尽	RS	记录分隔符
SUB	换置	US	单元分隔符
ESC	换码	DEL	删除

由表可以看出，ASCII码是用7位二进制数码来编码的，表示128个字符，这称为全 ASCII码。此外还有一种 6 位的ASCII码，它去掉了26个小写英文字母。如果用 8 位二进制 数，其中的7位表示ASCII码，多余的一位为奇偶校验位。

第二章　微型计算机的结构

微型计算机系统由硬件和软件两部分组成。硬件是指计算机设备本身；软件是事先编好的程序全体，以及相应的资料和说明书。软件不同于普通的解题程序，它存放在计算机里，是合理使用计算机系统的组成部分。它利用计算机本身的功能，合理地组织解题流程，简化或代替各个环节中人所承担的工作，便于用户熟悉和掌握计算机的使用，扩大和发挥其功能及效益。

2-1　微型计算机结构

图2-1所示是微型计算机硬件的结构框图。它由中央处理器 (CPU)、存贮器、输入输出 (I/O) 接口组成，通过I/O接口再与外部设备连接。相互之间通过三条总线 (BUS)——地址总线 (Address Bus)，控制总线 (Control Bus) 和双向数据总线 (Data Bus) 来连接。再加上必要的外设，即可构成一个微型计算机 (Micro-Computer)。

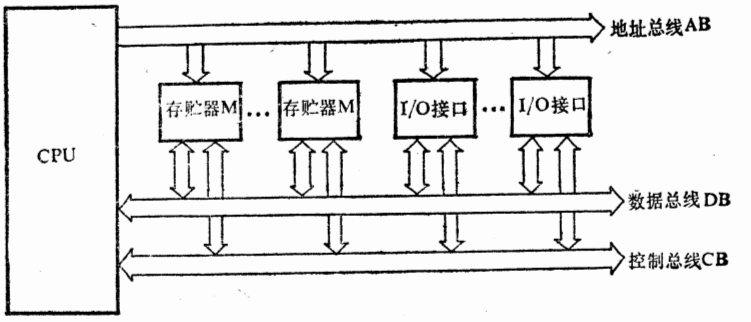


图2-1　微型机结构图

利用大规模集成电路技术把中央处理器 (CPU) 集成在一个芯片上，称为微处理器 (Microprocessor)。还可以把CPU，一定容量的RAM和ROM，以及输入输出接口电路，集成在一个芯片上，构成单片计算机 (Single Chip Computer)，或把CPU，RAM和ROM，输入输出接口装在一块印刷电路板上，即构成单板计算机 (Single Board Computer)。

2-1-1　CPU结构　中央处理器 (CPU-Central Processing Unit) 是计算机中运算器和控制器的总称，它是计算机的中枢，其作用是从存贮器取出指令和完成指定的操作。

图2-2　是CPU结构的示意图。

图中ALU是算术逻辑单元 (Arithmetic Logic Unit)，它是执行算术和逻辑运 算 的 装

置，通常以累加器A(Accumulator) 的内容作为一个操作数；另一个操作数由内部数据总线供给，它可以是寄存器 (Register) H中的内容，也可以是数据寄存器DR (Data Register) 供给的由内存读出的内容等；操作的结果通常放在累加器A中。

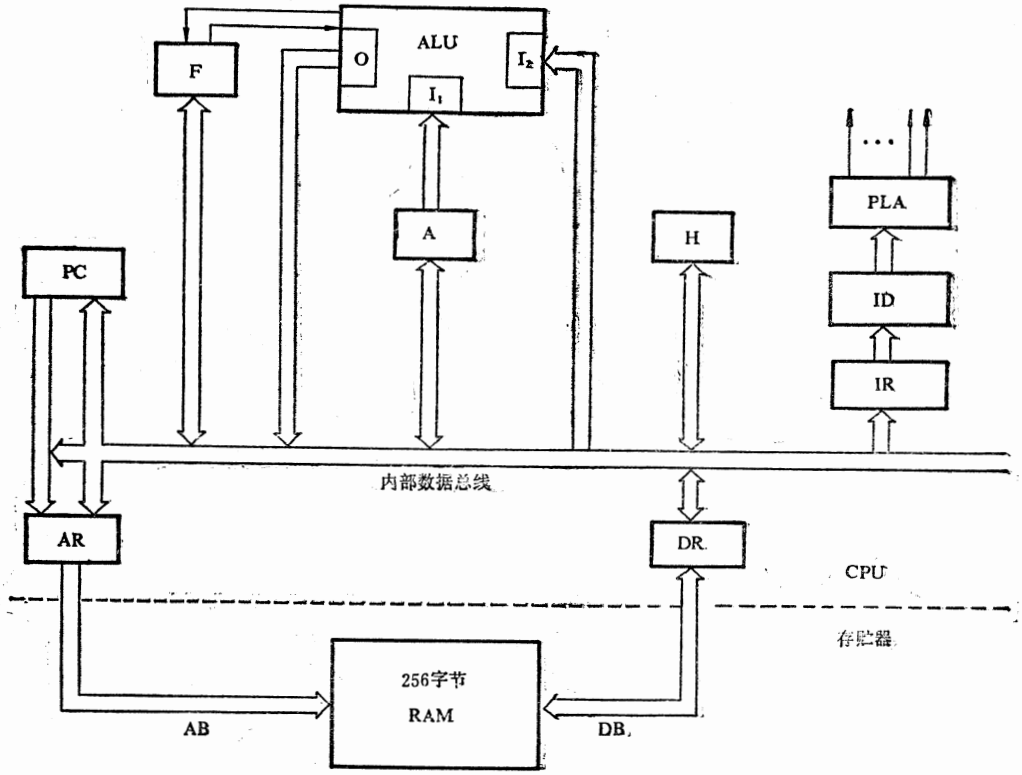


图2-2 CPU结构示意图

标志寄存器F(Flag)，由一些标志位组成，其内容是反映ALU操作的结果，例如是否有进位、辅助进位、运算结果是否为零等等。

程序计数器PC，它提供要执行的指令的地址，在微型机中通常是16位，AR(Address Register) 是地址寄存器，由它把要寻址的单元的地址通过地址总线，送至存贮器。从存贮器中取出的指令由数据寄存器送至指令寄存器IR (Instruction Register)，经过指令译码器ID (Instruction Decoder) 译码，通过控制电路，发出执行一条指令所需的各种控制信号。

在计算机中，把字 (word) 长规定为存贮器一个单元所包含的二进制信息的位数，把8位二进制位定义为一个字节 (Byte)。一般的微型机的字长为 8 位和16位，也有32 位的。在图2-2所示的结构图中，假设累加器A、寄存器H、数据寄存器DR都是8位的，因而双向数据总线也是8位的。PC一般为16位，它可寻址的内存空间为64k (1k为1024)。

在CPU内部各个寄存贮器之间及与ALU之间数据的传送也是采用内部总线结构，这样作可扩大数据传送的灵活性，减少内部连线，因此减少了这些连线所占的芯片面积。但是，采用总线结构，则在任一瞬时，总线上只能有一个信息在流动，因而使速度降低。

2-1-2 存贮器 存贮器是用来存贮程序和数据的。它由很多个单元(假定有256个单元)组成。为了能区分不同的单元，对每个单元分别给一个编号，这就是它们的地址。地址用16

进制数表示，如00、01、02、…FF等；每个单元可存放8位二进制数（通常也用16进制数表示），这就是它们的内容。

由地址总线上送来的地址，经过存贮器中的地址译码器来寻找指定的单元。然后，就可以对这个单元进行读或写操作。存贮器的结构如图2-3所示。

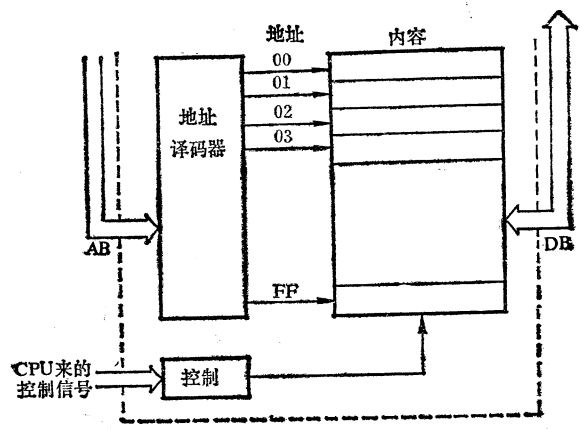


图2-3　存贮器结构图

一、读操作

假定在04号存贮单元中，存的内容为10000100（即84H），要想把它读到数据总线上，则要求CPU的地址寄存器先给出这个单元的地址（04），然后通过地址总线送至存贮器，地址译码器对它进行译码，找到04号单元；CPU发出读控制命令，于是04号单元的内容84H就出现在数据总线上，由它送至数据寄存器DR。如图2-4所示。

二、写操作

若要把数据寄存器中的内容26H写入存贮器的10号单元，则要求CPU的地址寄存器AR先给出地址10，通过地址总线AB送给存贮器，经译码后找到10号单元；然后把数据寄存器DR中的内容26H经数据总线（DB）送给存贮器；且CPU发出写控制命令，于是数据总线上的数据26H即可写入10号单元中，如图2-5所示。

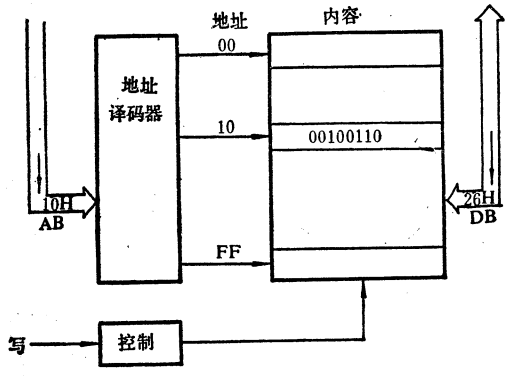


图2-5　存贮器写操作示意图

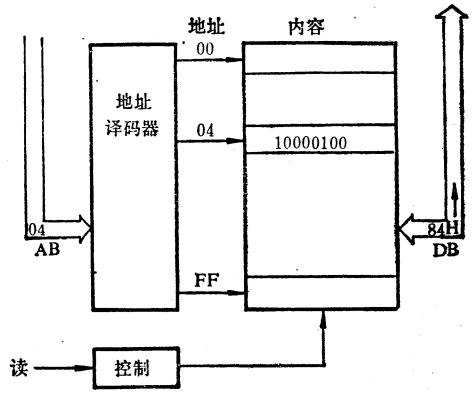


图2-4　存贮器读操作示意图

数据被写入存贮器的指定单元之后，在新的数据写入前一直被保留。对于非破坏性存贮器，读出指定单元的内容后，该单元的内容不变。

2-1-3 执行程序的简单过程 假定编写好的程序已通过输入设备（例如微型机的键盘）送入计算机，存在内存中，则机器执行程序的过程就是取指（取出指令）和执行指令这两个阶段的循环。

计算机从停机状态进入运行状态，要把第一条指令所在的地址赋给 PC，然后就 进入取指阶段。在取指阶段从内存中读出的内容必为指令，所以DR把它送至 IR，再由指令译码器译码，指出此指令要执行的操作。在取指阶段结束后就进入执行阶段。当一条指令执行完后，就进入下一条指令的取指阶段。在通常的情况下，指令是顺序执行的，因此当取出一条指令后，PC就自动增值，指出下一条要执行的指令的地址。当发生程序转移时，则把要转向的地址赋给PC，以取代原有地址。当遇到程序中的停机指令时，表示程序结束，停止运行，

2-1-4 中断 计算机在进行数据处理的过程中，常会遇到随机出现的异常情况和特殊请求，例如计算机内部发生硬件故障、程序出错以及外部设备（如键盘、打印机、磁带机）要求把信息传给计算机，或控制对象（如温度、压力）的参数发生超限报警要求计算机进行处理等。要求计算机能自动地转向处理随机出现的异常情况和请求。即提出了“中断”执行原数据处理程序的要求。为了能在恰当的时候响应中断请求，进行中断处理，并在处理完后准确无误地返回中断点，控制器中装有能够实现中断功能的装置，称为中断机构，也叫做中断系统。假如有一个发向CPU的请求信号，CPU接到这个请求信号之后，如允许暂时中止现行程序，则发出应答信号，并把程序计数器PC的内容和状态标志、累加器 A 及各寄存器的内容压入堆栈以后，就转而执行请求任务的程序。处理完毕后，把堆栈中保存的内容弹出，使PC、A、各寄存器等恢复原来的状态，再回到暂停的正常程序继续执行。

计算机中的中断系统具有如下功能

一、中断源的设置

不同的机器可以包含不同的中断源，一般有：

1.便于调机和调试程序的中断；

2.监控程序正确运行的中断；

3.便于人机通讯的中断；

4.提供机器检查用的中断；

5.用于自动调度外部设备的中断；

6.用于和外围设备或生产过程通讯连接的中断；

7.自中断。

二、中断优先级的安排

通常，在系统中有多个中断源（即能发出中断申请的来源），常会出现两个或更多个中断源同时提出中断请求的情况，根据轻重缓急，事先给每一个中断源确定一个中断级别—优先权。当多个中断源同时发出中断请求时，可利用硬件或软件的方法进行排队，使CPU首先响应优先权级别最高的中断源的中断请求（INT），处理完后，再响应级别较低的中断请求。

三、中断的响应

各种中断经排队后，还要在恰当的时候才能被响应，即需要满足一定的条件。如一条指令执行完后，要判断是否有中断请求信号输入，就是所谓的中断查询条件。若某个中断在排队中居先，并满足中断查询条件，就可被响应。此外，还与CPU内部的中断允许触发器INTE (Interrupt Enable)的状态有关，这个触发器可以用允许中断(EI) 指令把它置 1，这时CPU输出有效的允许中断信号(如INTE输出高电平)，表示允许中断。也可以利用不允许中断指令

(DI)把它置为0，使CPU不输出有效的允许中断信号（如 INTE为0），这时即使有中断请求信号，CPU也不响应中断。

四、中断的屏蔽

机器设有的某些中断在某些情况下不允许响应或不希望出现，就对这些中断实行屏蔽。屏蔽的方法有两种：一是软件方法；二是硬件方法。硬件方法是对每种需要屏蔽的中断源设置屏蔽状态触发器，比如当状态触发器为"1" 时，中断请求才会被响应。把各屏蔽状态触发器连在一起组成中断屏蔽寄存器。

五、中断处理和返回

1．断点及有关信息的保护

中断请求被响应，现行程序暂停执行时PC 中的指令地址称为断点（即下一条应执行的指令的地址）。运算器和控制器中的一些信息称为现场。断点和现场要在转向执行中断处理程序以前保护起来，即送进预定的内存单元（通常用堆栈），以便返回被中断的程序时取用。

2．中断处理及返回

不同的中断请求需要不同的中断处理程序，而这些程序是存放在内存的不同区域。所以当某一中断被响应时，要转向执行它的处理程序，就必须根据不同情况，采用一定的方法形成这个中断处理程序的首地址（入口地址）。中断被响应时，立即执行一条隐指令，完成中断断点和现场的保护，形成处理程序入口地址，接着转向执行相应的中断处理程序，对中断进行处理。在中断隐指令不能完成保护现场的情况下，可在处理程序首部继续完成现场保护。

在中断处理程序的末尾设置一定的指令，将保护的断点和现场送回处理机，实现正确的返回。一般机器的指令系统中都有相应的中断和返回指令。

六、多重中断

当CPU响应某一中断请求并正执行中断处理程序时，又出现了优先权级别更高的中断请求，则CPU能中断正在进行的中断处理程序，响应高级中断，在高级中断处理完以后，再继续进行被中断的中断处理程序。若发出新的中断请求的优先级与正在处理中断请求同级或更低时，则CPU先不响应这个中断请求，直至正在处理的中断处理程序执行完以后才去处理新的中断请求。

2-1-5 直接存贮器传送 在通常的情况下，外设要把数据送到内存，或者内存中的数据要送到外设，均要通过CPU 进行传送。如果外设（如磁盘）要和内存之间高速交换数据，最有效的方法是不经过 CPU，而使外部设备和内存之间直接传送数据，这种方法称为直接存贮器传送（Direct Memory Access，DMA）。

DMA传送的步骤如下：

一、外部设备发出DMA请求。（对8080A，这一信号接到HOLD端）。

二、CPU在执行完现行指令之后响应这一请求，发出保持应答信号（HLDA）信号。这时CPU暂停执行原来的程序，并把地址总线和数据总线让给DMA传送使用。这时CPU和这些总线之间处于高阻状态，CPU中各寄存器的状态保持原状。

三、进行DMA传送时，有专门的DMA控制器控制，例如Intel 8257可编程序DMA控制器，它的内部有地址寄存器，事先通过程序，由 CPU写入DMA传送时存贮器的起始地址，并把待传送的字节数写入计算字节数的寄存器中。

四、DMA控制器送出存贮器的起始地址，并发出控制信号，控制外设把数据传到存贮器的指定地址（或按相反方向传送）。传送一个字节后，地址自动加1，计数寄存器自动减1，不

断的传送，直到计数寄存器为0，全部传送完毕。

五、保持(HOLD)请求结束，保持应答HLDA 信号回到低电平。CPU 继续执行原来的程序。

2-2 几种常用的微处理器

自1971年美国Intel 公司制造出 4 位微处理器 4004 以来，微处理器无论在品种和数量上都有了飞速的发展。Intel 公司生产的8080A与Motorola公司的MC6800和 Zilog 公司的Z80是最通用的 8 位微处理器。1978年，Intel公司最先推出16位通用微处理器8086，与后来Motorola 公司的MC68000和Zilog 公司的Z8000，成为举世瞩目的三种16位机。

2-2-1 Intel 8080A

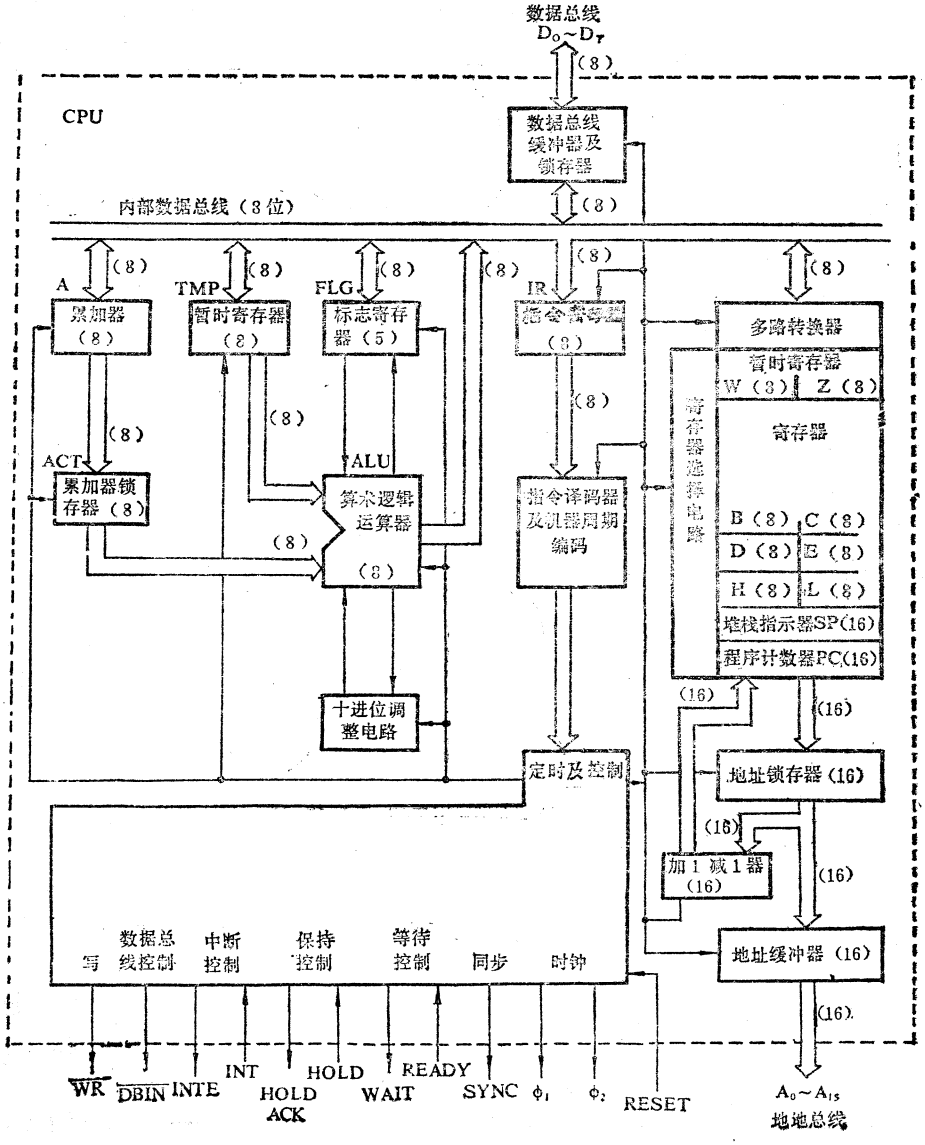


图2-6 Intel 8080A CPU 的内部结构框图

一、内部结构

Intel公司生产的8080A微处理器的内部结构如图2-6所示。

其中：

1．程序计数器(PC) 16位，其内容指出下一条要执行的指令的地址。

2．指令寄存器(IR)及控制部分

CPU从存贮器的指定单元取出指令后，由指令译码器译码，然后由定时和控制部分结合外部送来的信息，产生控制信号，用来控制CPU内部的各种寄存器、缓冲器、运算器等以及向外部的存贮器、输入输出端口等发出控制信号。

3．累加器(AC或A)

累加器是运算时的暂时寄存器。算术和逻辑运算部件的运算结果，一般存在累加器中。此外，在进行逻辑运算和移位等操作时，也用到累加器。

4.通用寄存器

通用寄存器用来暂时存放参加运算的操作数、中间结果或地址。图2-6中的寄存器B、C、D、E、H、L都是通用寄存器，他们可以单独使用，也可以组成BC、DE、HL三个16位的寄存器对使用。

5.堆栈指示器(SP)

16位，它的内容指出现栈顶的地址。堆栈是在内存(或寄存器)中，按照后进先出原则组织的一个存贮区域。SP保存现堆栈顶的地址。数据可通过执行压入(PUSH)和弹出(POP)指令，从指定的CPU寄存器压入堆栈，或从堆栈弹入CPU寄存器。利用堆栈可以实现多级中断、子程序嵌套和简化许多类型的数据操作。

6.多路转换器

8位，其作用是：使数据从内部数据总线传到某一寄存器或从某一寄存器传到内部数据总线。

7.地址锁存器和地址缓冲器

16位，它们可把程序计数器、堆栈指示器或寄存器对中存放的16位地址信息送到地址总线上。地址锁存器还可以把来自寄存器对中的16位信息或PC、SP中的16位地址码送到加1减1器中，实现加1或减1后再送回原处。

8.暂时寄存器W和Z

这两个8位的寄存器只在执行某些指令时供内部使用。

9.标志寄存器

8位，8080A只用了5位，每一位表示CPU内部的某种状态，通常叫做标志，或称为条件码。

10.算术逻辑运算部件(ALU)

算术逻辑运算部件，在指令译码后产生的控制信号控制下，完成算术加、减；逻辑与、或、异或、非运算；左移、右移操作；增1、减1操作；把二进制数修改成二—十进制数的操作等。

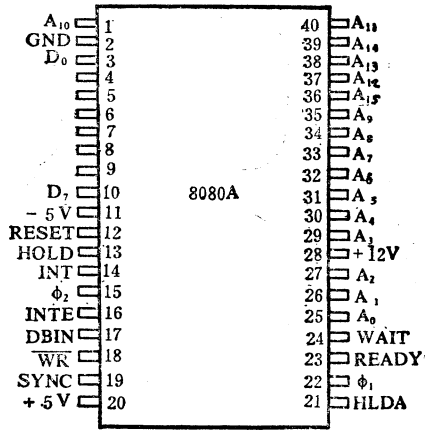


图2-7 Intel 8080A片脚

二、Intel 8080A的片脚及其功能

Intel 8080A是40只片脚的双列直插式组件，如图2-7所示。

各片脚的功能是：

1.A_0—A_{15}(地址总线)：输出，三态。

传送所要访问存贮器单元或输入输出接口的地址。

2.D_0—D_7(双向数据总线)：双向，三态。

这一数据总线称为CPU数据总线。它与系统控制器8228相连，既可以输入数据也可以输出数据，所以称为双向数据总线。系统控制器8228和存贮器、输入输出设备接口相连的数据总线称为系统数据总线。它们统称为数据总线。

3.DBIN(数据总线允许)：输出，高电平有效。

DBIN 信号控制数据总线上的数据传送方向。当它有效时，允许存贮器或输入接口的信息输入CPU。

4.$\overline{WR}$(写)：输出，低电平有效。

这一信号有效时，允许CPU把数据从数据总线写入存贮器或输出设备接口中去。

5.HOLD(保持请求)：输入，高电平有效。

这一信号是外部设备向CPU发出的保持请求信号，当CPU 响应这一信号后，CPU与地址总线、数据总线断开，即它们之间处于高阻状态。这时允许 DMA 控制器对地址总线和数据总线进行控制，执行DMA操作。

6.HLDA(保持响应)：输出，高电平有效。

HLDA是由CPU对HOLD信号的回答信号。

7.READY(就绪)：输入，高电平有效。

当这一信号有效时，表示存贮器或输入输出设备把存取的数据准备就绪。

8.WAIT(等待)：输出，高电平有效。

这一信号是CPU收到低电平的READY信号后的应答信号，表示CPU处于等待状态。

9.INT(中断请求)：输入，高电平有效。

这一信号是外围设备要求处理某一临时事件向CPU发出的请求信号。

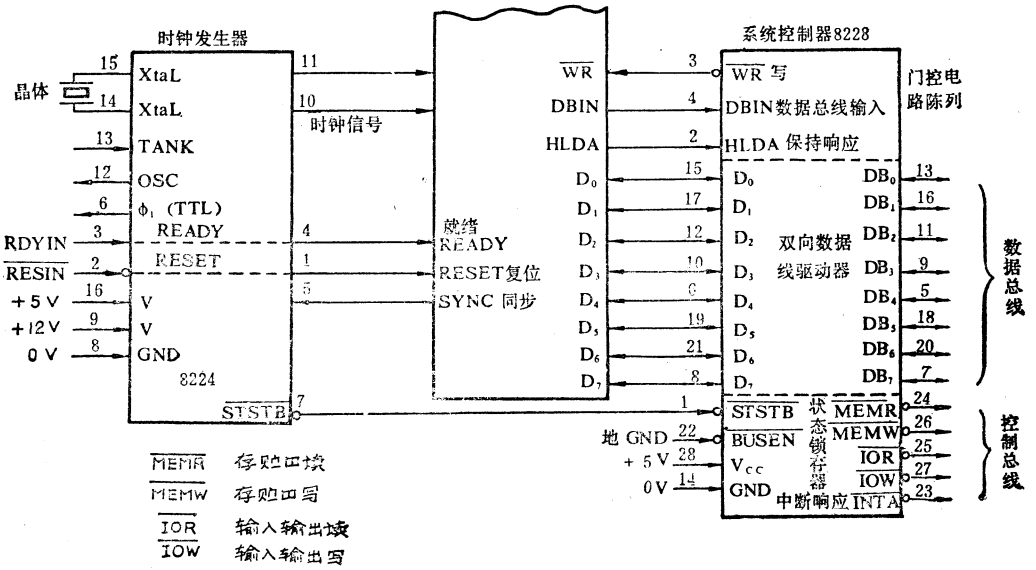


图2-8 CPU、时钟发生器、系统控制器的连接图

10. INTE(允许中断)：输出，高电平有效。

11. RESET（复位）：输入，高电平有效。

RESET信号是使 CPU 处于初始状态的信号。当这一信号有效时，程序计数器被置 0，指令寄存器置成空操作指令。但数据寄存器、累加器、堆栈指示器和标志寄存器都不清除。

12. SYNC（同步信号）：输出。

三、时钟发生器8224和系统控制器8228

8080A和时钟发生器8224以及系统控制器8228三部分合在一起构成了中央处理器。它们之间的连接关系如图2-8所示。

1. 时钟发生器8224

8224提供 Intel 8080A 各种操作所需要的定时信号以及其它定时信号。

8224产生的2.048兆赫的两相时钟信号 ϕ_1 和ϕ_2，送到8080A作为定时信号。从 CPU 来的 SYNC（同步）信号和ϕ_1取得同步后变成$\overline{\text{STSTB}}$（状态选通）信号送入系统控制器8228，用以锁存状态信息。通电时的$\overline{\text{RESIN}}$（复位输入）信号与ϕ_2同步后产生RESET信号送到CPU。存贮器或输入输出设备送来的RDYIN（就绪输入）信号与ϕ_2同步后成为READY 信号送到 CPU。OSC端的20兆赫的信号可作为串行传送数据时的定时信号。8224时钟发生器产生的ϕ_1的一个周期称为时钟周期或T周期，也称为一个拍节，它是CPU操作的最小单位。三到五个T周期合起来称为一个机器周期，或称M周期，一个M周期完成一个基本操作。在每一个机器周期的第一个时钟周期T_1时，CPU输出一个同步信号（SYNC），实现每一个机器周期开始工作的同步。根据指令的类型，完成一条指令要一到五个机器周期，完成一条指令的总时间称为指令周期。

2. 系统控制器8228

8228由 8 位双向数据线驱动器、状态锁存器以及门控电路陈列三部分组成。

双向数据线驱动器的作用是增加Intel 8080A数据总线的负载能力。当$\overline{\text{WR}}$(写)信号为低电平时，CPU送出的数据已在数据总线上稳定，可以写入存贮器或输出接口，当 DBIN 信号为高电平时，系统数据总线上的信息被CPU读入。

表2—1　　状 态 信 息 的 功 能

数据总线位	状态信息符号	功　　能
D_0	INTA(中断响应)	它是中断请求信号的应答信号，当CPU接受中断请求时，它为高电平
D_1	$\overline{\text{WO}}$(写入)	当本机器周期为写入(存贮器)或输出时，为低电平；反之当读入或输入时，为高电平
D_2	STACK(堆栈)	如地址线上已有从堆栈指示器来的堆栈地址时，它为高电平
D_3	HALT(暂停响应)	CPU执行暂停指令的响应信号
D_4	OUT(输出)	地址总线上有输出设备的地址，数据线上有输出数据时，它为高电平
D_5	M_1(第1机器周期)	在指令的第一个机器周期，即取指令操作码周期时为高电平
D_6	INP(输入)	在地址总线上有输入设备接口地址，数据总线上有输入数据时，它为高电平
D_7	MEMR(存贮器读)	数据总线用于存贮器读时为高电平

状态锁存器的作用是锁存 D_0—D_7 数据总线上的状态信息。当每一机器周期的第一个时钟周期T_1时，8080A把 8 位状态信息(其功能如表2-1所示)送到D_0—D_7数据总线上，当时钟发生器产生状态选通信号$\overline{STSTB}$时，锁存器把这 8 位状态信息锁存起来。8 位状态信息和 DBIN、$\overline{WR}$、HLDA 等信号通过门控电路形成$\overline{MEMR}$ (存贮器读)、$\overline{MEMW}$ (存贮器写)、$\overline{INTA}$ (中断响应)等控制信号，送到存贮器或输入输出接口进行读、写等控制。

$\overline{STSTB}$是由时钟发生器8224输出到系统控制器8228的状态选通信号。$\overline{INTA}$ 是中断应答信号。

8080A CPU 是 8080 的最初改进型。Intel 公司在 1976 年底又研制成功了将 8080 CPU、8228、8224三片LSI 做在一片芯片上的LSI 芯片，这就是8085 CPU 芯片。8080 CPU 和 8085 CPU芯片的片脚排列如图2-9和图2-10所示。

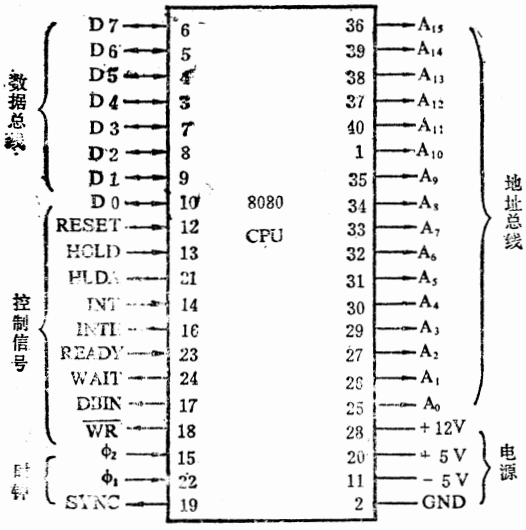


图2-9　8080的管脚排列

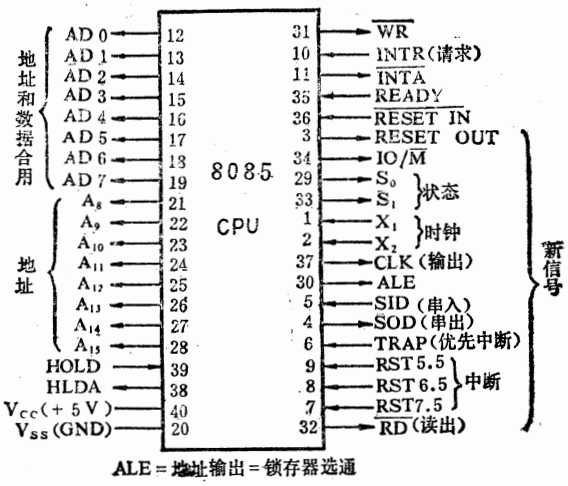


图2-10　8085 CPU 芯片的管脚排列

2-2-2 Intel 8086和8088 Intel 8086是美国Intel公司于1978年推出的16位机产品，目前已成为国际上最流行的16位机之一。

8086和8088是密切相关的第三代微处理机。8088CPU 与存贮器、输入/输出部分之间的外部数据通路是 8 位的，而8086则是16位。在其它方面，这两种处理机几乎完全一样。这两种CPU的突出特点是：它们构成的微型计算机系统应用范围广，适应性强。系统的规模可以小到只由少量器件构成的存贮容量很小的单处理机系统，也可以大到具有 1 兆字节存贮容量的多处理机系统。

一、8086和8088 CPU的结构

在 8086 和 8088 CPU 中，有两个独立的处理单元：执行单元 (EU) 和总线接口单元 (BIU)。执行单元负责执行指令，总线接口单元负责取指令、读出操作数和写入结果。这两个单元能相互独立地工作，因此，在大多数情况下，能使取指令和执行指令重迭进行，缩短了执行多条指令的时间。

图2-11　表示 8086和8088 CPU的结构框图

(一)执行单元 (EU)

8086和8088的EU是相同的。EU负责执行全部指令、给 BIU 提供数据和地址以及控制通用寄存器和标志寄存器。EU与系统总线并不相连，它从BIU中的一个指令队列中获得指令.当指令需要访问存贮器或外部设备时，EU就请求BIU存取数据。EU 处理的所有地址都是16位的。

1.通用寄存器

8086和8088这两种CPU都有 8 个16位的通用寄存器。这些通用寄存器又分成两组，每组四个寄存器。数据寄存器组，包括累加器 (AX)、基址寄存器 (BX)、计数寄存器 (CX)和数据寄存器 (DX)；指针和变址寄存器组 (也称P和I组寄存器)，包括堆栈指针 (SP)、基址指针 (BP)、源变址寄存器 (SI) 和目的变址寄存器 (DI)。

数据寄存器组有时也称为H和L组寄存器，H和L分别表示寄存器的“高 8 位”和 “低 8 位”，高 8 位和低 8 位能分别寻址，即是说每个数据寄存器既可以用作一个 16 位寄存器，又可用作两个 8 位寄存器。CPU中的其它寄存器则只能用作16位的寄存器。在大多数算术和逻辑操作中，可以随意地使用数据寄存器，也可以使用指针和变址寄存器。

2.暂存寄存器

寄存操作数。

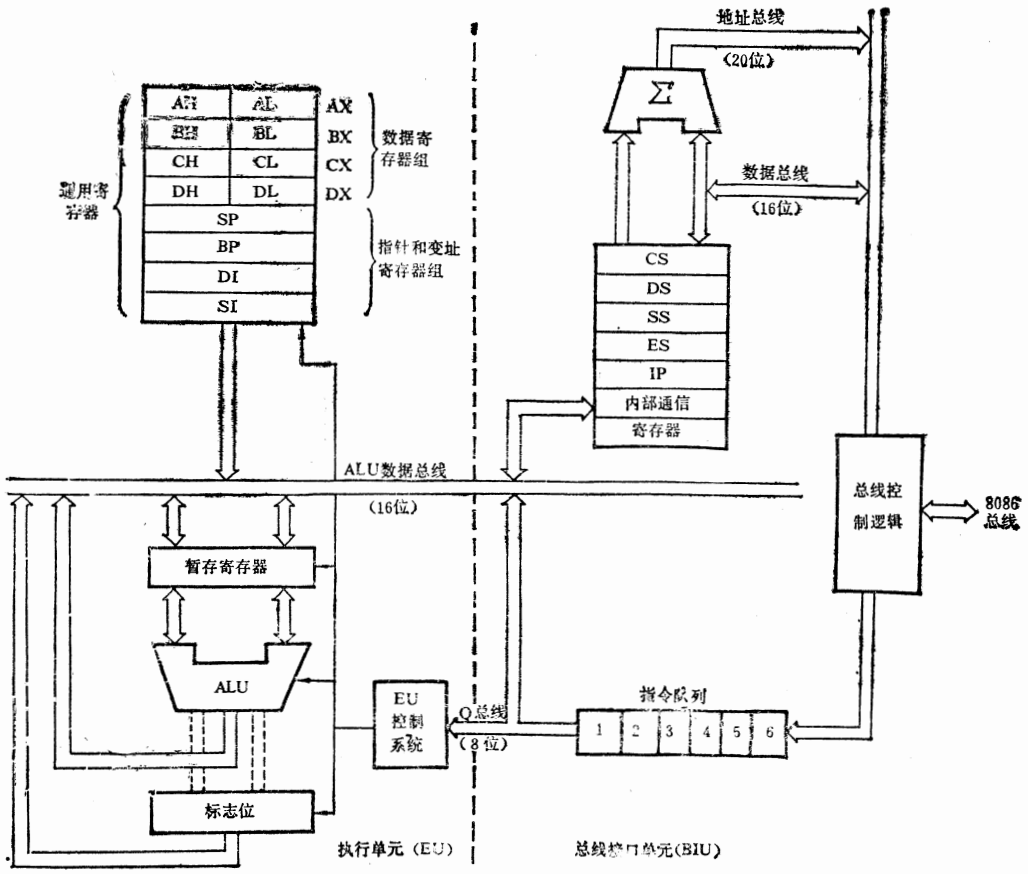


图2-11 (a) 8086的基本框图

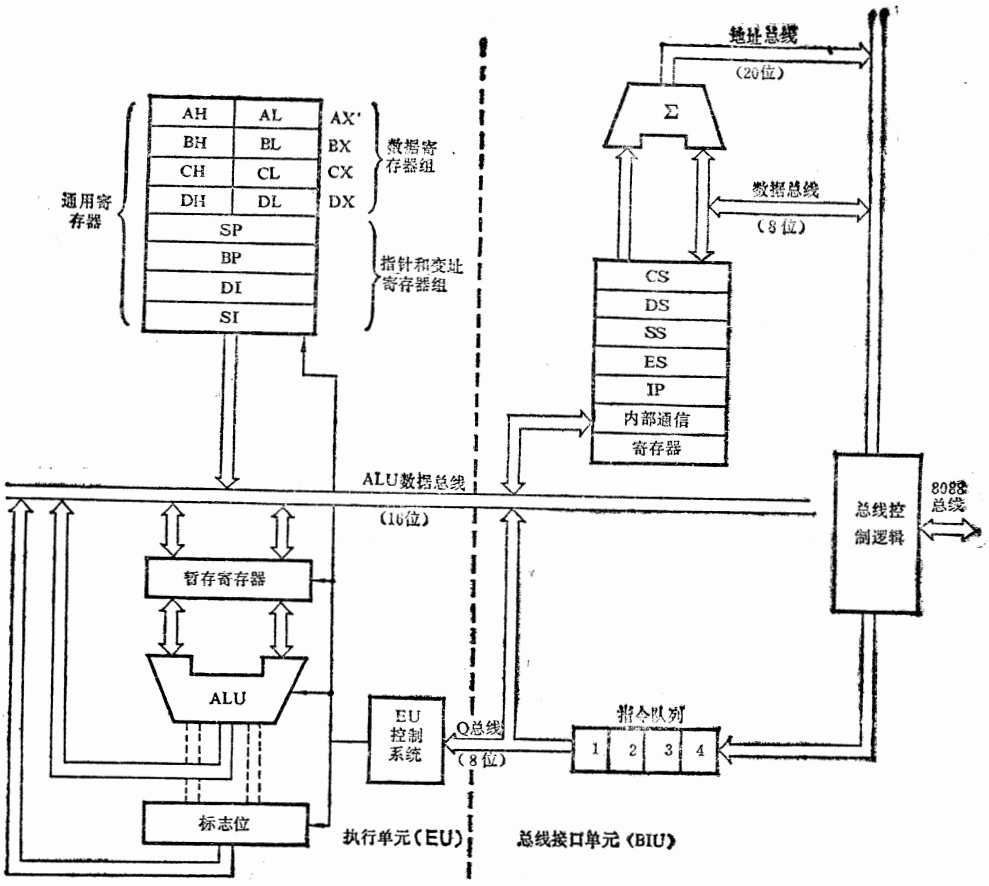


图2-11 (b) 8088的基本框图

3.标志

8086和8088均有6个1位的状态标志，这些标志由EU设置，以反映算术或逻辑运算结果的某些性质。有一组指令使程序能根据这些标志的状态（即前一次运算的结果）而改变其执行方向。

(二)总线接口单元 (BIU)

8086和8088的BIU原理上都是一样的,但实际电路有所不同,以适应各自的总线结构和性能特点。BIU为EU完成所有的总线操作，它包括段寄存器、通信寄存器、指令指针以及指令队列等。

1.段寄存器

8086和8088有4个16位的段寄存器：指令段寄存器 (CS)、数据段寄存器 (DS)、堆栈段寄存器 (SS) 和附加段寄存器 (ES)。

8086和8088可具有1兆字节的存贮空间，这些存贮空间被分成为许多逻辑段，每段最多可包含64K字节。CPU每次可以直接访问四个段。这四个段的基地址(起始单元地址) 存放在段寄存器中。CS寄存器指向当前的指令段，指令从此段取出。SS寄存器指向当前的堆栈段，堆栈操作所处理的正是这段中存贮单元的内容。DS寄 存器指向当前的数据段，它通常存的是程序变量。ES寄 存器指向当前的附加段，它通常也是用来存放数 据。程序通过改变段寄

存器的内容使之指向所需要的段，因而可存取那些段的指令和数据。

BIU用它的加法器把段寄存器的内容和偏移值组合，形成20位的地址。

2.指令指针（IP）

16位的指令指针由BIU 修改，使之始终存有下一条指令相对于当前指令段起点的偏移量（以字节为单位的距离）。

3.指令队列

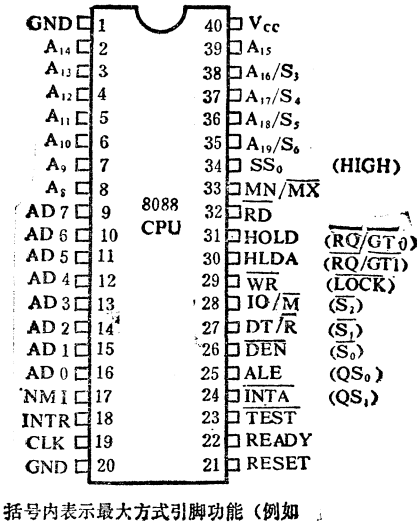
在EU忙于执行指令时，BIU“提前”从存贮器中取出更多的指令，这些指令存放在一个内部的RAM阵列中，该阵列称为指令队列。

8088 的指令队列最多能保存四个字节的指令流，而 8086 的队列最多能保存六个指令字节。这样长的队列使BIU能保证在大多数情况下向 EU 提供预取的指令，而又不会独占系统总线。在8086CPU中，当指令队列中空了 2 个或更多个字节且EU 不要求 BIU 执行总线周期时，BIU就执行取指令周期，以便填满队列。在 8088 CPU 中，当队列中空了 1 个字节时，BIU就执行取指令周期。由于8086 CPU 有 16 位数据总线，因此它能在 1 个总线周期内访问两个指令目标码字节。而8088 CPU 只有 8 位数据总线，因此每个总线周期只能访问 1 个指令目标码字节。如果 BIU 正在执行取指令总线周期时EU发出访问总线的请求，则 BIU必须在响应EU的请求之前结束该取指令周期。若EU执行的是一条控制转移指令，则 BIU 就使队列复位，从新的地址取指令，并立即传送给 EU，然后开始从新的单元取出指令来重新组成队列。

二、8088和8088 CPU的引脚

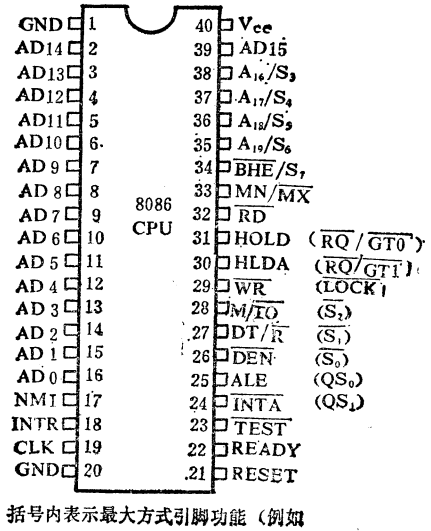
8088 CPU的引脚如图2-12所示。各引脚的功能列在表2-2内。

8086 CPU 的引脚如图2-13所示，其功能列在表 2-3内。



括号内表示最大方式引脚功能（例如 LOCK等）

图2-12 8088CPU的引脚



括号内表示最大方式引脚功能（例如 LOCK 等）

图2-13 8086 CPU 引脚

表2—2　8088 CPU 引脚的功能

公　用　信　号

名　称	功　能	类　型
AD7-AD0	地址/数据总线	双向、三态
A15-A8	地址总线	输出、三态
A19/S6-A16/S3	地址/状态	输出、三态
MN/$\overline{MX}$	最小/最大方式控制	输　入
$\overline{RD}$	读 控 制	输出、三态
$\overline{TEST}$	等待测试控制	输　入
READY	等待状态控制	输　入
RESET	系统复位	输　入
NMI	不可屏蔽中断请求	输　入
INTR	中断请求	输　入
CLK	系统时钟	输　入
Vcc	+5伏	输　入
GND	接　地	输　入

最小方式信号(MN/ $\overline{MX}$ = Vcc)

名　称	功　能	类　型
HOLD	保持请求	输　入
HLDA	保持响应	输　出
$\overline{WR}$	写 控 制	输出、三态
IO /$\overline{M}$	IO/存储器控制	输出、三态
DT/$\overline{R}$	数据发送/接收	输出、三态
$\overline{DEN}$	数据允许	输出、三态
ALE	地址锁存允许	输　出
$\overline{INTA}$	中断响应	输　出
SS0	S0状态	输出、三态

最大方式信号(MN /$\overline{MX}$ = GND)

名　称	功　能	类　型
$\overline{RQ}$ /$\overline{GT1,0}$	请求/允许总线访问控制	双　向
$\overline{LOCK}$	总线优先封锁控制	输出、三态
$\overline{S2}$ - $\overline{S0}$	总线周期状态	输出、三态
QS1,QS0	指令队列状态	输　出

表2—3　8086CPU引脚的功能

公　用　信　号

名　称	功　能	类　型
AD15-AD0	地址/数据总线	双向、三态
A19/S6-A16/S3	地址/状态	输出、三态
$\overline{BHE}$/S7	总线高允许/状态	输出、三态
MN/$\overline{MX}$	最小/最大方式控制	输　入
$\overline{RD}$	读 控 制	输出、三态
$\overline{TEST}$	等待测试控制	输　入
READY	等待状态控制	输　入
RESET	系统复位	输　入
NMI	不可屏蔽中断请求	输　入
INTR	中断请求	输　入
CLK	系统时钟	输　入
Vcc	+5伏	输　入
GND	接　地	

最小方式信号(MN/$\overline{MX}$ = Vcc)

名　称	功　能	类　型
HOLD	保持请求	输　入
HLDA	保持响应	输　出
$\overline{WR}$	写 控 制	输出、三态
M/$\overline{IO}$	存储器/IO控制	输出、三态
DT/$\overline{R}$	数据发送/接收	输出、三态
$\overline{DEN}$	数据允许	输出、三态
ALE	地址锁存允许	输　出
$\overline{INTA}$	中断响应	输　出

最大方式信号(MN/$\overline{MX}$ = GND)

名　称	功　能	类　型
$\overline{RQ}$/$\overline{GT1,0}$	请求/允许总线访问控制	双　向
$\overline{LOCK}$	总线优先权锁定控制	输出、三态
$\overline{S2}$—$\overline{S0}$	总线周期状态	输出、三态
QS1,QS0	指令队列状态	输　出

三、工作方式的选择

8086和8088这两种CPU都有一条“搭接”引脚（MN/$\overline{MX}$），这条引脚决定8086中8条CPU引脚和8088中9条引脚的功能（见表2-4）。将MN/$\overline{MX}$接+5伏电源，就使CPU处于最小方式，在这种工作方式下，CPU本身提供存贮器和外部设备所需要的总线控制信号。当MN/$\overline{MX}$引脚接地时，CPU处于最大方式，此时需要增添一个Intel 8288总线控制器来提供更为复杂的总线控制功能，并与Multibus结构兼容（8288与Intel 8289总线仲裁器配合使用），就使CPU能支持系统总线上的多个处理机。

表2-4　　最小/最大方式的引脚配置

8086			8088		
引脚	方式		引脚	方式	
	最小	最大		最小	最大
31	HOLD	RQ/$\overline{GT0}$	31	HOLD	RQ/$\overline{GT0}$
30	HLDA	RQ/$\overline{GT1}$	30	HLDA	RQ/$\overline{GT1}$
29	$\overline{WR}$	$\overline{LOCK}$	29	$\overline{WR}$	$\overline{LOCK}$
28	M/$\overline{IO}$	$\overline{S2}$	28	IO/$\overline{M}$	$\overline{S2}$
27	DT/$\overline{R}$	$\overline{S1}$	27	DT/$\overline{R}$	$\overline{S1}$
26	$\overline{DEN}$	$\overline{S0}$	26	$\overline{DEN}$	$\overline{S0}$
25	ALE	QS0	25	ALE	QS0
24	$\overline{INTA}$	QS1	24	$\overline{INTA}$	QS1
			34	SS0	高态

（一）最小方式

在最小方式时（MN/$\overline{MX}$引脚搭接到+5伏），CPU支持由少量设备组成的并使用系统总线的单处理机小型系统而不支持Multibus结构。在最小方式时，CPU本身产生全部总线控制信号（DT/$\overline{R}$、$\overline{DEN}$、ALE和M/$\overline{IO}$或IO/$\overline{M}$）和命令输出信号（$\overline{RD}$、$\overline{WR}$或$\overline{INTA}$），并提供请求访问总线的逻辑（HOLD/HLDA），该逻辑与总线主设备控制器（例如Intel 8237和8257 DMA控制器）兼容。

在最小方式时，当总线主设备请求访问总线时，它通过HOLD请求逻辑使输入到CPU的HOLD信号变为有效（高电平）。若CPU响应“HOLD”请求，则它把HLDA信号变为有效，以此作为对总线主设备请求的回答，同时使系统总线和有关的控制线浮动。因为总线

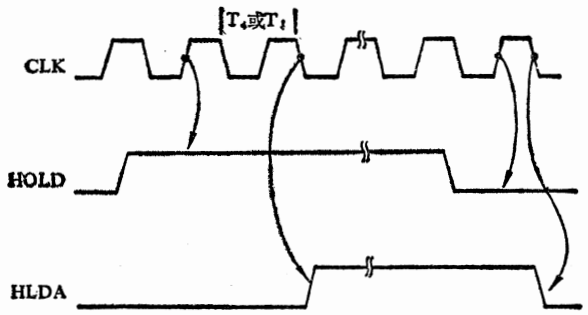


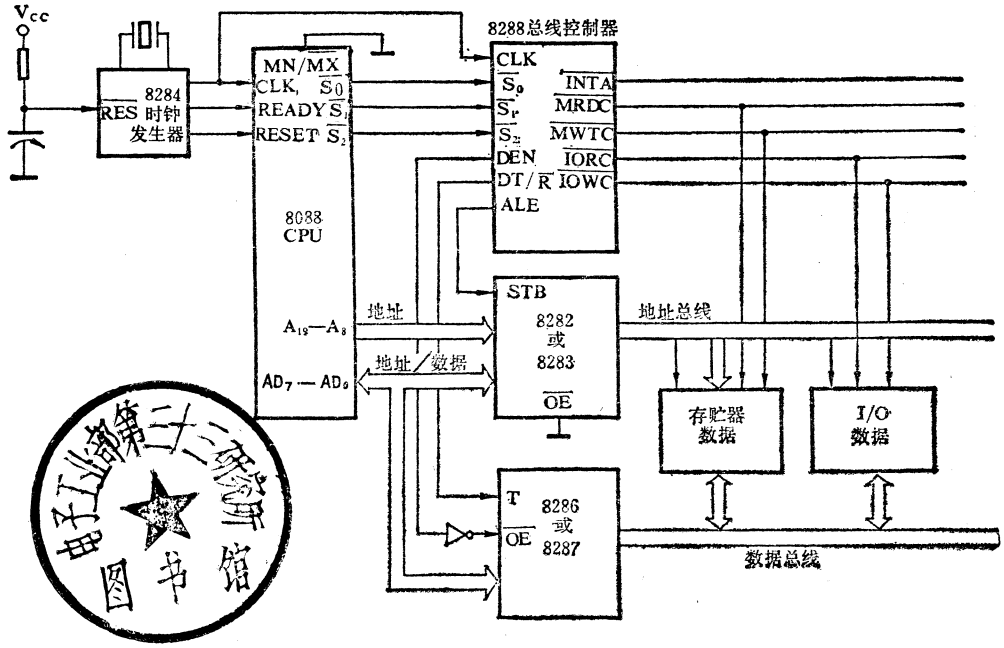
图2-14　HOLD/HLDA 定时图

请求是异步式输入的，所以 CPU 在每个 CLK 时钟信号的正向边沿采样 HOLD 输入信号，并且如图2-14所示，在现行总线周期结束（如果总线周期正在进行）或在空闲时钟周期时，CPU 把 HLDA 信号变为有效。CPU 保持 HOLD 状态，一直到总线主设备把 HOLD 信号变为无效为止。HOLD信号变为无效后，CPU 重新获得系统总线的控制权。

在最小方式时，8088 CPU 的I/O存储器控制线的极性与8086CPU的相应控制线的极性相反（8086是M/$\overline{IO}$，而8088是IO/$\overline{M}$）。由于8088是 8 位的器件，因此为它创造了与目前使用的 MCS-85系统及其系列器件（如 Intel 8155/56）兼容的条件。

（二）最大方式

在最大方式时（MN/$\overline{MX}$引脚搭接到地），增添一个 Intel 8288总线控制器来提供更复杂的总线控制功能，并与 Multibus 结构兼容（8288与 Intel 8289 总线仲裁器配合使用），就使 CPU能支持系统总线上的多个处理机。如图2-15所示，提供所有总线控制信号和命令信号的是总线控制器，而不是 CPU。总线控制器允许 CPU 的引脚消去原有的功能，而重新定义新的含义以支持多处理机工作方式。



基本的最大方式系统

图2-15　基本的最大方式系统

Intel 公司生产的8086（iAPX 86/10）分为三种时钟类型：8086为 5 兆赫型，8086-2为 8 兆赫型，8086-1为10兆赫型。8088（iAPX 88/10）可以直接与iPAX 86/10软件兼容，还可以直接与8080/8085硬件和外设兼容。

iAPX 86/10和一片数值处理扩充片（NPX）8087构成 iAPX 86/20，一个 iAPX 88/10 和一片8087可组成 iAPX88/20。这两种微处理器是数值数据处理器（NDP），它们都封装成两片40引脚双列直插式。这两种处理器在进行数值处理方面的性能要比单独使用 iAPX86/10 或iAPX 88/10优越100倍。

一片 iAPX 86/10 (或 iAPX 88/10)可与一片操作系统固件 (OSF) 80130组成操作系统处理器 iAPX 86/30 (或 iAPX 88/30)，这两种微处理器封装成两片 40 引脚双列直插式，由于它们具有 CPU 的指令系统与实时操作系统相结合的特点，因此适合于各种多程序与多任务应用场合。

1982年，Intel 公司新推出一种先进的高性能微处理器 iAPX 286/10，它采用 68 引脚四列直插封装。如图2-16所示。

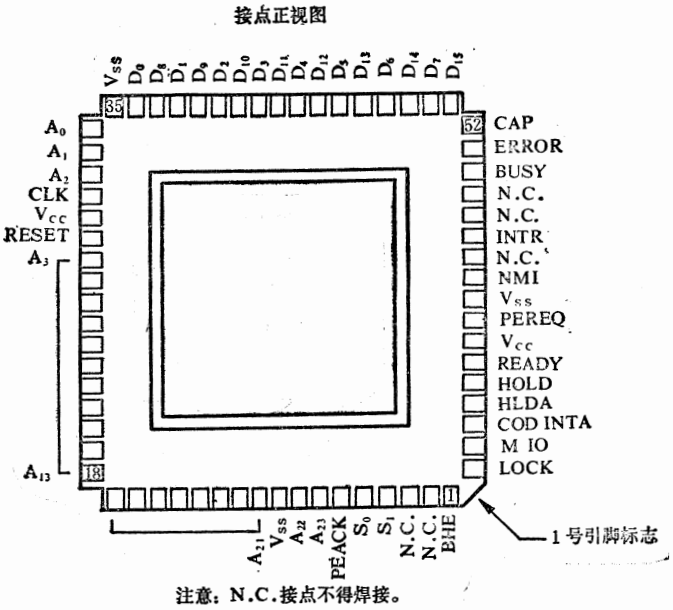


图2-16 80286引脚配置图

2-2-3 Z-80 CPU Z-80 微处理器是美国 Zilog 公司在 1974 年研制出来的。它是在 Intel 8080的基础上，吸收了 Motorola 6800 的优点研制成的一种八位的微处理器。

一、Z-80 CPU 的内部结构

Z-80 CPU的内部结构如图2-17所示。

(一)CPU 内部寄存器组

Z-80 的 CPU 内部寄存器组分为两类，一类是专用寄存器，另一类是通用寄存器。它们是：

1.程序计数器PC：16位，保存有待从存贮器中取出的现行指令的地址。

2.堆栈指针（或称堆栈指示器）SP

3.变址寄存器 IX 和IY

这是两个独立的16位寄存器，通常它们各自包含着一个16位的基地址，由它加上指令中给定的偏移量以形成操作数的有效地址。

4.中断页地址寄存器 I

这是一个 8 位的寄存器，它保存中断处理程序入口地址表的页地址。

5.存贮器刷新寄存器R

存贮器刷新寄存器也称存贮器再生寄存器。Z-80使用动态存贮器时，要求定期（一般为

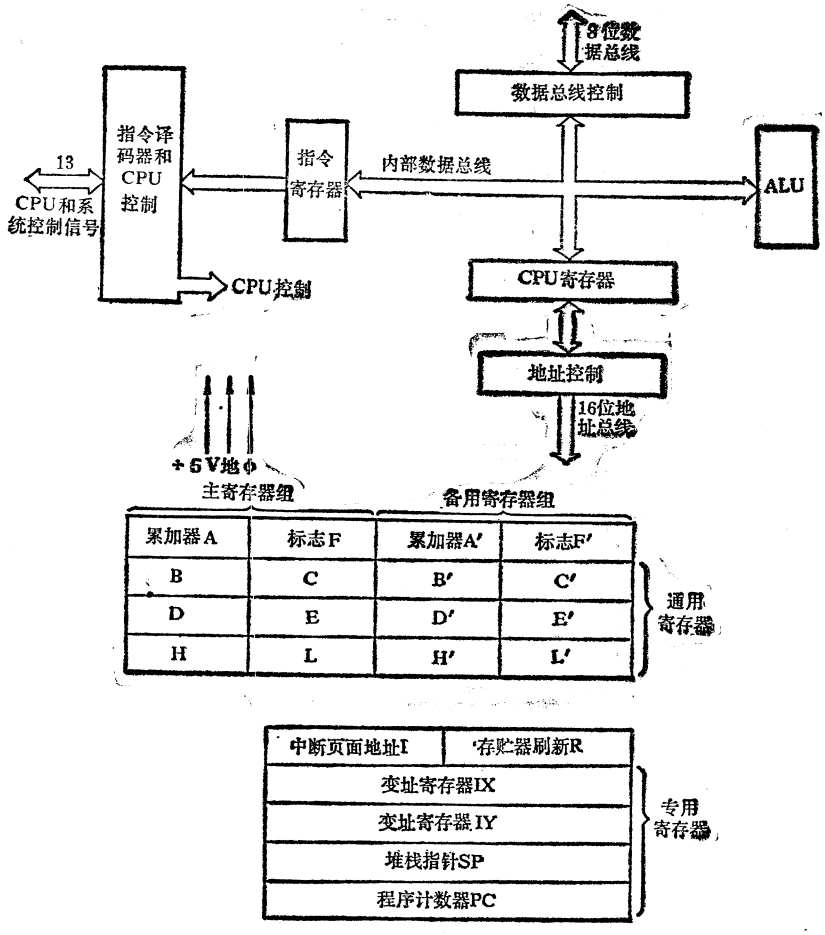


图2-17 Z-80 CPU 的内部结构

2ms）对动态存贮器进行刷新。Z-80 中 是利用 CPU 读出指令后，对指令进行译码和执行内部操作时不使用地址总线这段时间，按次序一行一行地刷新存贮器各单元。每次刷新的存贮器单元（一行）的地址由R寄存器（7 位）提供，而在每刷新一行后，R的内容自动加1，以指向下一行。

6.累加器和状态标志寄存器

Z-80中有两个累加器和与它相连的状态标志寄存器。在进行算术和逻辑操 作时，累加器A中的内容必为一个操作数，且操作的结果放在累加器中。另外，算术和逻辑操作结果的一些特征，例如操作结果是否为0，有没有进位等等，都寄存在标志寄存器中。

7.通用寄存器组

Z-80中有两组完全一样的通用寄存器，每组都有 6 个 8 位的寄存器，它们可以分别作为 6 个 8 位寄存器使用，也可以每两个连起来形成BC、DE、HL或B′C′、D′E′、H′L′三对16位的寄存器对。它们主要用于寄存参与运算的数据或操作数的地址。

（二）算术和逻辑部件ALU

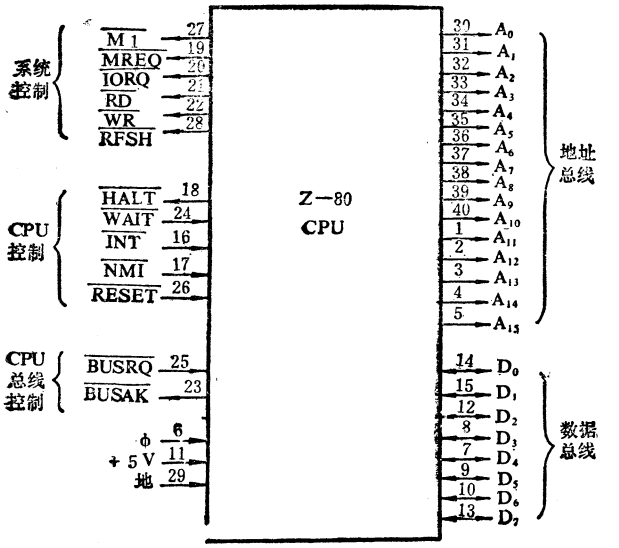
CPU的 8 位算术和逻辑指令在ALU中执行。

（三）指令寄存器和CPU控制

这部分完成控制器的功能

二、Z-80 CPU 引脚及其功能

Z-80 CPU用工业标准40脚双列直插式管壳封装。输入输出管脚如图2-18 所示。



Z—80 CPU引脚图

图2-18 Z-80 CPU 引脚图

在40条引脚中，有16条为地址总线 A_0—A_{15}，8条双向数据总线，有三条作为电源、地和时钟信号线，另外13条为控制信号线。这些引线的功能是：

A_0—A_{15}（地址总线）：三态输出，高电平有效。A_0—A_{15} 组成16位地址总线。

D_0—D_7(数据总线)：三态输入/输出，高电平有效。D_0—D_7组成双向数据总线，用于CPU、存贮器、I/O设备之间的数据交换。

$\overline{M}_1$(机器周期1)：输出，低电平有效。$\overline{M}_1$指明现行的机器周期是取操作码周期。当操作码是二字节操作码时，则每一取操作码周期，都发出$\overline{M}_1$信号，这样的双字节操作码的第一个字节常为十六进制数CB、DD、ED、或FD。此外，$\overline{M}_1$还与$\overline{IORQ}$信号一起指示一个中断响应周期。

$\overline{MREQ}$(存贮器请求)：三态输出，低电平有效。这个信号表明地址总线上保持有一个供存贮器读或写操作的有效地址。

$\overline{IORQ}$（输入/输出请求)：三态输出，低电平有效。这个信号指示地址总线的低八位中保持有一个供I/O读或写操作所需的有效I/O地址。当中断得到响应时，$\overline{IORQ}$和$\overline{M}_1$同时产生，通知外设把中断向量放到数据总线上。在M_1期间可以发生中断响应操作，但决不会发生I/O操作。

$\overline{RD}$（存贮器或I/O读)：三态输出，低电平有效。$\overline{RD}$指明CPU欲从存贮器或I/O设备读取数据。被寻址的I/O设备或存贮器应利用此信号作为门控信号，把数据放到CPU的数据总线上。

$\overline{WR}$（存贮器或I/O写)：三态输出，低电平有效。这个写信号指示CPU的数据总线上保持着待存入选定的存贮器或I/O设备的有效数据。

$\overline{RFSH}$(刷新)：输出，低电平有效，这个信号指明地址线的低7位(A_0—A_6，A_7是逻辑零)，保持有动态存贮器的刷新地址。这个信号应与$\overline{MREQ}$信号一起用于刷新动态存贮器。

$\overline{HALT}$(暂停)：输出，低电平有效。$\overline{HALT}$指明，CPU执行了HALT软件指令，并正在等待不可屏蔽或可屏蔽中断。在暂停期间，CPU执行空操作指令，保持存贮器的刷新操作。

$\overline{WAIT}$(等待)：输入，低电平有效。$\overline{WAIT}$告诉CPU所寻址的存贮器或I/O装置，尚未准备好数据传送。只要这个信号有效，CPU继续插入等待周期。这个信号能使慢速的存贮器或I/O设备与CPU同步。这个信号相当于8080的READY信号。

$\overline{INT}$ (中断请求)：输入，低电平有效。这个信号由I/O设备产生，如果由内部软件控制的中断开放触发器 (IFF) 处在允许状态，且在没有总线请求的情况下，则在现行指令结束时响应中断。当CPU接受中断，在下一个指令的开始，送出一个中断响应信号 (在$\overline{M_1}$周期有$\overline{IORQ}$信号)。

$\overline{NMI}$(不可屏蔽中断)：输入，负边沿触发。这个非屏蔽中断请求信号，具有比$\overline{INT}$更高的优先权，它不受IFF控制且总是在现行指令结束时被响应。$\overline{NMI}$自动地迫使Z-80 CPU转向0066H单元。PC的内容自动地保存在堆栈中，以保证用户能返回到中断前的程序。需注意：连续的等待周期会阻止现行指令的结束，另外，总线请求信号$\overline{BUSRQ}$优先于$\overline{NMI}$。

$\overline{RESET}$ (复位)：输入，低电平有效。复位信号迫使PC清零并使CPU置于初始状态，包括：

1.清除中断允许触发器；

2.置I寄存器为00H；

3.置R寄存器为00H；

4.置中断方式0。

在复位期间，地址总线和数据总线处于高阻状态，且所有控制信号处于无效状态。

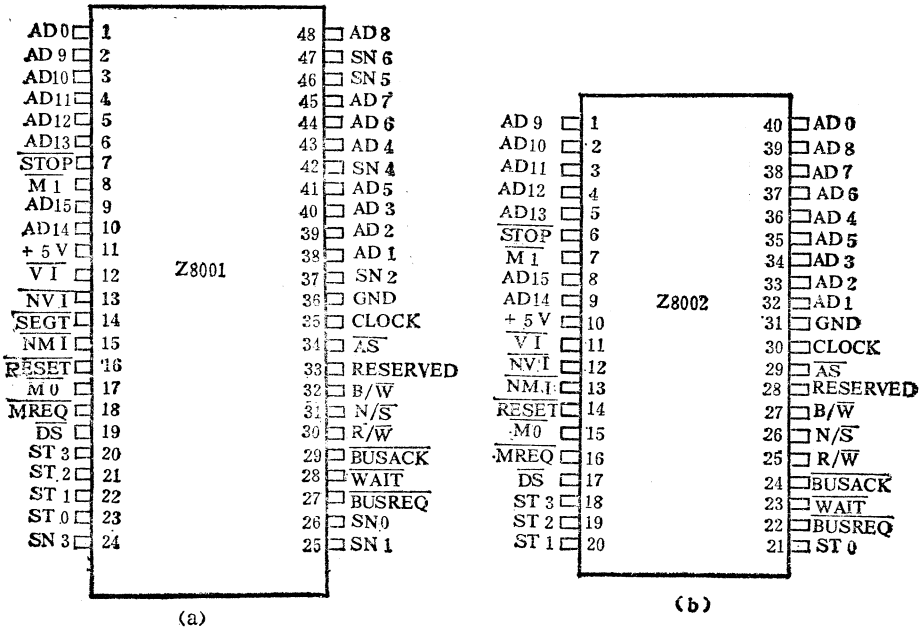


图2-19　　Z8000引脚图

$\overline{\text{BUSRQ}}$（总线请求）：输入，低电平有效。这个信号用于要求CPU把地址总线、数据总线和三态输出控制信号转为高阻状态，使其它设备能利用这些总线。当$\overline{\text{BUSRQ}}$有效时，CPU在现行的机器周期结束时立即响应，使这些总线处于高阻状态。

$\overline{\text{BUSAK}}$（总线响应）：输出，低电平有效。这个信号告诉总线请求设备，CPU的地址总线、数据总线和三态控制总线已处于高阻状态，外部设备现在可以使用这些总线。

ϕ(时钟)：单相系统时钟。Z-80时钟频率为2.5MHz。

2-2-4 Z8000 CPU Z8000是Zilog公司1979年推出的16位微处理器，它采用了n沟E/DMOS工艺，约有17500个管芯，单电源5V，功耗300mw，主频为6兆赫。Z8000有两种机型：40引脚的Z8002和48引脚的Z8001，如图2-19所示。

Z8000微处理器可以用于简单的单机独立系统，又可以用于复杂的多机并行处理系统。它的存贮管理部件可使用户有8兆字节的寻址空间，外围接口部件可以智能化地执行复杂任务，大大减轻了CPU的负担，提高了系统的吞吐能力。

Z8001有48条引脚(见图2-19(a))，是可分段的，直接寻址能力可达8兆字节；Z8002有40条引脚（见图2-19(b))，直接寻址能力为64K字节。

Z8000CPU有两种操作方式：系统方式和正常方式，从而可改进操作系统的执行能力。功能很强的指令系统和硬件结构特性相结合，可以支持多道程序的处理。

2-2-5 MC6800 微处理器 MC6800微处理器是Motorola公司在1974年研制成功的。它是用一片40个管脚、双列直插式封装内含有八位的并行处理机。该处理机具有变长堆栈、可屏蔽中断矢量、直接存贮器寻址的能力，6个内部寄存器及72条变长指令和七种寻址方式。微

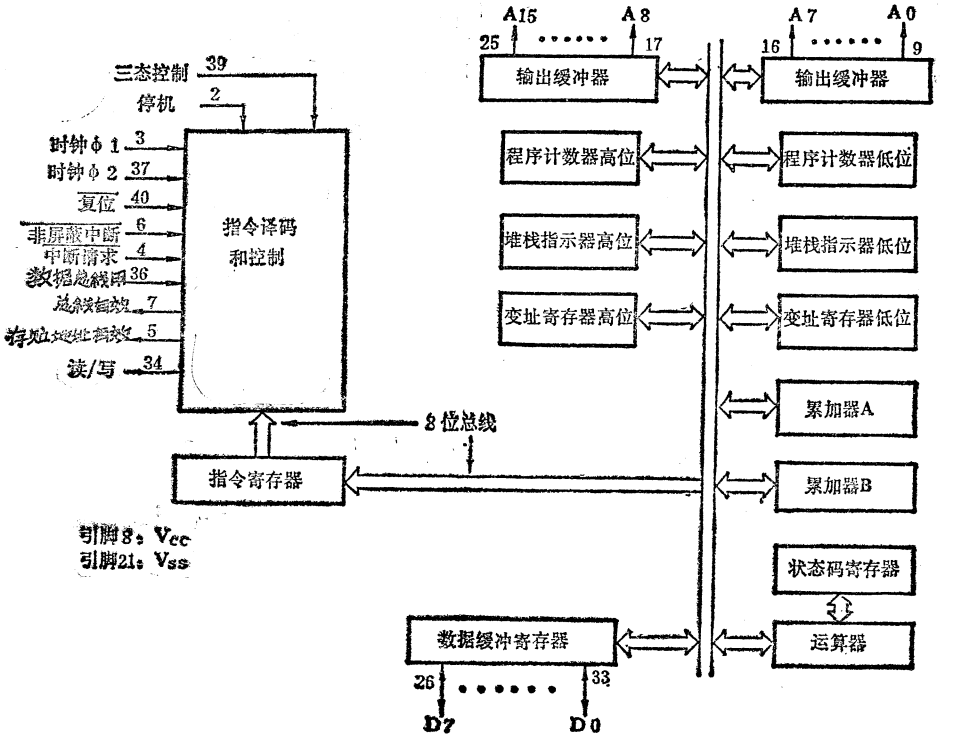


图2-20 MC6800结构框图

处理器的结构框图如图2-20所示。堆栈指示器、程序计数器和变址寄存器是三个16位的寄存器。状态码寄存器、累加器A和累加器B为8位。

MC6800的寻址能力为64K字节，在变址或扩展寻址时可存贮到74K字节。

图2-21是MC6800的片脚图。

IRQ　中断请求

VMA　有效存贮地址

NMI　非屏蔽中断

BA　总线有效

RES　复位

TSC　三态控制

DBE　数据总线可用

2-2-6　MC68000微处理器　MC68000是美国Motorola公司于1976年开始设计，而于1980年正式发表的16位微处理器。它是目前16位微处理器中功能最强的一种。

MC68000具有独自的多重微程序构造，它的所有处理器操作和指令都是用微码写的，这样便于改正设计错误。而且，在其指令码中，有1/8以上是为将来的扩展而保留的。

MC68000 可处理六种基本数据类型：位、字节、字（16位）、长字（32位）、BCD码及ASCII字符，还可以处理存贮器地址和状态数据等数据类型。

MC68000是用HMOS工艺制作的大面积集成芯片，采用双列直插式封装，64条引线，其编号如图2-22所示。

一、地址总线A_1～A_{23}

有23根地址线，没有A_0，不能直接寻址16M字节，但是用上总线控制部分的两个引线的输出信号UDS（高位数据选通）和LDS（低位数据选通）及AS（地址选通）信号组合起到A_0。

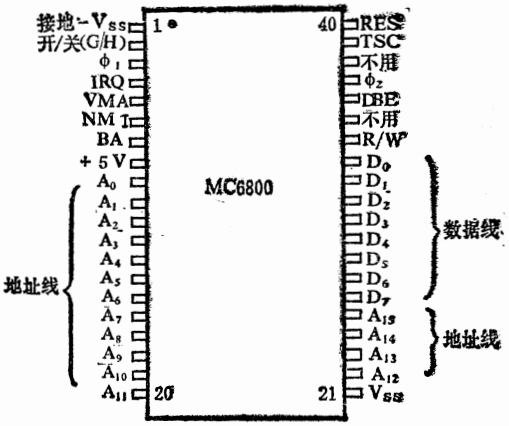


图2-21　MC6800片脚图

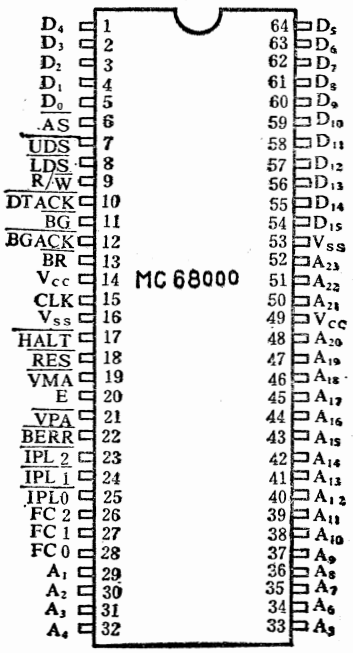


图2-22　MC68000片脚图

的作用，这样可直接寻址16M字节。

二、异步总线控制

以异步方式进行MC68000与16位的外围芯片和存贮器之间的数据传送。

$\overline{AS}$(地址选通)：表示地址总线上的地址有效。

$\overline{DTACK}$（数据传送响应)：数据传送结束的应答信号。

R/$\overline{W}$(读/写)：表示总线的周期是读或写周期。

$\overline{UDS}$（高位数据选通）和$\overline{LDS}$（低位数据选通)：这两个信号加上相应的 R/$\overline{W}$和$\overline{AS}$，表示处理字节。它们的组合，完成数据总线的控制。

三、总线仲裁控制

包括下述三个信号，以决定哪一个设备获得总线控制权。

$\overline{BR}$（总线请求)：要求使用总线的设备向处理器发出控制总线的信号。

$\overline{BG}$（总线准许)：许可总线仲裁，总线在当前周期结束时可以给请求总线的设备使用。

$\overline{BGACK}$（总线准许响应)：是从$\overline{BG}$来的确认信号，而且必须是总线空闲时才可发出，经仲裁表示选择好。

四、中断控制

由下述三个信号组成：

$\overline{IPL0}$、$\overline{IPL1}$、$\overline{IPL2}$（中断优先级别)。只有当外面中断优先级别比当前操作级别高时才能中断。

五、处理器状态

由FC0、FC1、FC2三个信号组成。MC68000有二种基本运行状态，即管理态和用户态(还有一种为调试用的跟踪态)。对这三个信号提供的功能码进行译码，可决定处理器进入哪一个基本状态。

六、外围控制

用于与8位的M6800的外围芯片连接

E(允许)：用于和M6800外围电路接口，其周期为MC68000的10倍。

$\overline{VPA}$（有效外围地址)：用于指定与M6800互换性的某个地址。

$\overline{VMA}$（有效存贮器地址)：表示M6800系统的外部设备中有效地址已在总线上。它是对$\overline{VPA}$的响应。

七、系统控制

包括下面三个信号：

$\overline{RESET}$（复位)：输入输出共用。“输入”是对MPU本身或系统全体的复位，“输出”是对外围芯片的复位。

$\overline{HALT}$（停止)：用来在当前机器周期结束时使处理器停止。它是双向的，“输入”是对MPU本身的停止，而“输出”是用于MPU辨认相对于外围芯片处理HALT状态。

$\overline{BERR}$（总线出错)：通知MPU发生了总线错误。

八、数据总线

D_0～D_{23}为双向数据通道，可以按高位字节、低位字节、字来访问总线。

九、时钟CLK。

在MC68000中，有16个32位的通用寄存器，一个24位的程序计数器，一个16位的状态寄

存器，运算器（ALU）也为16位。

2-2-7　6502 CPU　6502是用于APPLE等多种微型机中的微处理器芯片。

一、6502 CPU的内部结构

6502 CPU的内部结构如图2-23所示。

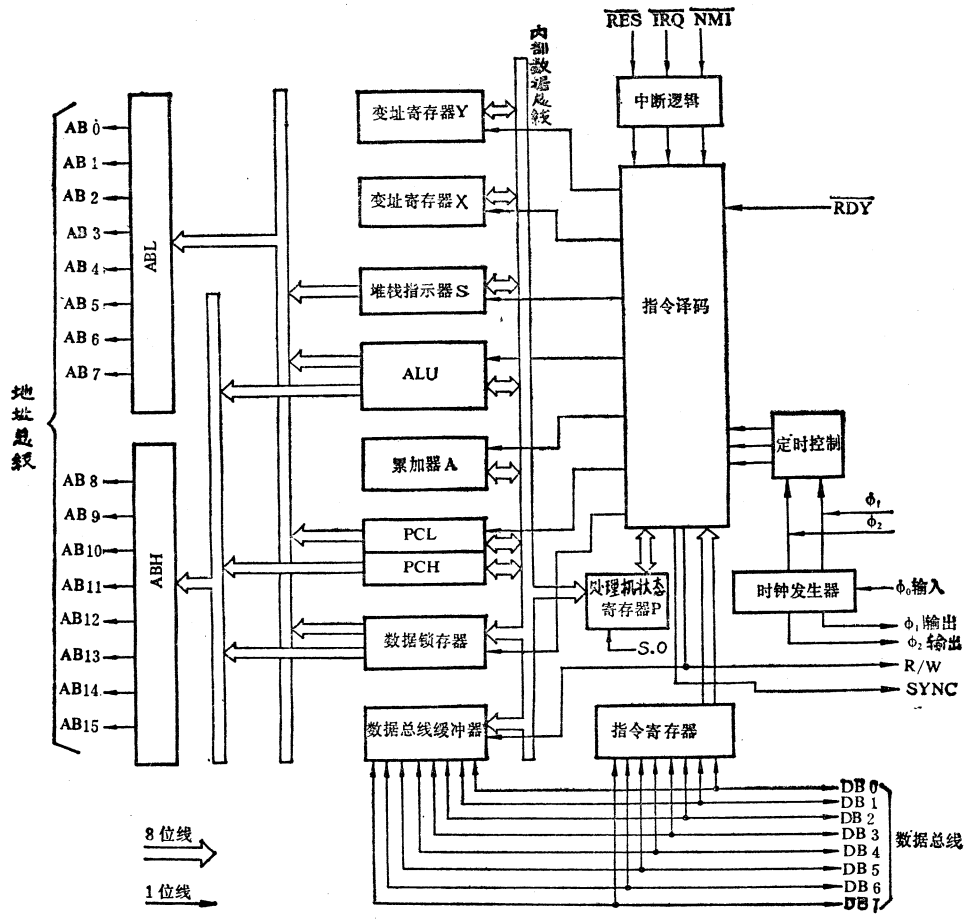


图2-23　6502内部结构框图

6502和其它微处理器有许多相同之处。从它的内部结构上看基本上分为两部分。第一部分为控制部分，包括指令寄存器、指令译码器、时钟发生器、定时控制器、中断逻辑部件等。另一部分为寄存器部分，有变址寄存器X和Y、堆栈指示器S、累加器A、程序计数器PC、标志寄存器P、数据总线缓存器和算术逻辑部件等。

1.程序计数器PC，16位。

2.堆栈指示器S，8位。由于6502规定了堆栈区域为0100～01FF，高位固定为01，因此，只要一个8位寄存器作为堆栈指示器，指示栈顶的地址。

3.变址寄存器X和Y，8位。用作存放变址寻址方式中的地址位移量，或作一般的寄存器用。

4.累加器A，8位。

5.标志寄存器P，8位，实际使用7位。

6.算术逻辑部件ALU。ALU执行算术运算、逻辑运算、移位等操作。与其一同工作的有

累加器A、标志寄存器P、内部数据锁存器和数据总线缓存器。

7.指令寄存器。存放从存贮器中取出的指令。

二、6502的片脚及其功能

6502的片脚如图2-24所示。

6502CPU是一个大规模集成电路芯片，有40个引出脚，各引脚的功用如下：

1.AB0～AB15（地址总线）：地址总线与存贮器ROM、RAM以及外部设备的地址线相联接。

2.DB0～DB7(数据总线)：数据总线的传送方向是双向的，它们用来实现6502与存贮器或与外设接口之间的信息传送。

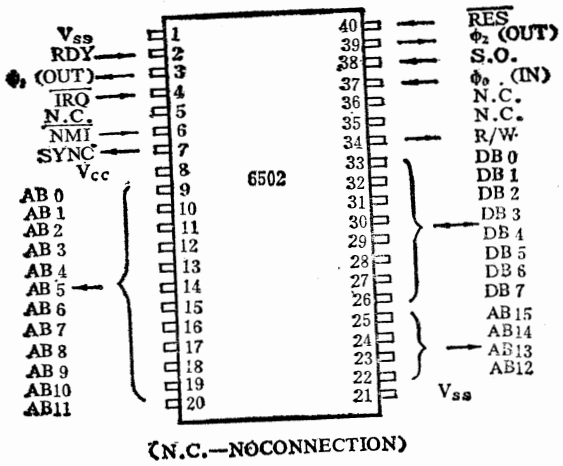


图2-24 6502片脚图

3. R/$\overline{W}$（读/写控制）：它控制数据总线上数据的流向。

4. SYNC（同步）：输出控制信号。

5. $\overline{RES}$（复位）：输入控制信号，低电平有效。无论何时，只要$\overline{RES}$信号有效，就使中央处理器初始化。

6. RDY（就绪）：输入控制信号，高电平有效。当这个信号有效时，表示存贮器或I/O设备已为传送数据作好了准备，允许进行数据交换。因此，它可使处理器与任意速度的存贮器或I/O设备相同步。

7. IRQ（中断请求）：输入控制信号，低电平有效。当6502的标志寄存器中的中断禁止位状态为0时，中断请求$\overline{IRQ}$被响应。

8. $\overline{NMI}$（不可屏蔽中断请求）：输入控制信号，低电平有效。中央处理器无法屏蔽这个中断请求，因此，一旦这个控制信号出现，中央处理器执行完当前的一条指令后立即响应这个中断请求。

9. Φ_0、Φ_1、Φ_2（定时控制信号）：其中Φ_0是输入脉冲信号，1MHz。Φ_1和Φ_2是输出脉冲信号，Φ_1与Φ_0反相，Φ_2与Φ_0同相。

10.S.O.(置溢出标志位)。

11.V_{CC}、V_{SS}（直流供电端）。

表2-5

公司名称	产品型号	公司名称	产品型号
DG公司	Microeagle	Motorola公司	Mc68020
DEC公司	Micro VAX1	National Semiconductor公司	NS32032
HP公司	Focus	NCR公司	NCR/32
Inmos公司	Transputer	Western Electric公司	WE32000
Intel公司	iAPX386	Zilog公司	Z80000

2-2-8 32位微处理器 随着大规模集成电路技术的迅速发展，各半导体厂生产的微处理器集成度不断提高，功能也不断增强。据1984年的资料，美国已有十家公司在生产32位微处理器，它们的名称和型号见表2-5

在微处理器技术方面处于领先地位的Intel公司已准备在1985年底推出利用CHMOS（互补高速金属一氧化物一半导体）工艺制成的32位微处理器80386。这种微处理器的性能完全可与32位小型相媲美。还有消息表明，Intel公司已开始准备研制功能更强的微处理器80486，这种微处理器的性能将相当于今天的大型机。

第三章 存贮器和接口

3-1 存贮器

存贮器是计算机极其重要的组成部分，有了它计算机才能有记忆功能，才能把要计算和处理的数据以及程序存入计算机，使计算机能自动地工作。

存贮器的容量和工作速度是影响计算机性能的重要因素。存贮器的容量越大，记忆的信息也越多，计算机的功能也越强。七十年代以来，随着大规模集成电路技术的发展，半导体存贮器的集成度大大提高，体积急剧减小，同时存取速度也有了极大的提高，使半导体存贮器逐步取代了磁芯存贮器。

3-1-1 半导体存贮器的分类 从使用功能上来分，半导体存贮器可分为随机存取存贮器RAM (Random Access Memory)，又称为读写存贮器；只读存贮器ROM (Read Only Memory)。RAM主要用来存放各种现场的输入、输出数据，中间计算结果，以及与外存交换信息和用作堆栈。它的存贮单元的内容按需要既可以读出，也可以写入或改写，而ROM的信息在使用时不能改变，即不可写入，只能读出，因此一般用来存放固定的程序，如微型机的管理、监控程序，汇编程序等。

一、RAM的种类

RAM可分为双极型 (Bipolar) 和MOS RAM 两类。双极型 RAM 的特点是存取速度高，但集成度较低，功耗大，成本也高。因此双极型 RAM 主要用在速度要求较高的微型机中。MOS RAM 又分为静态RAM和动态 RAM 两种。静态 RAM 的集成度高于双极型，但低于动态 RAM，功耗比双极型的低，它不需要刷新，故可省去刷新电路。动态的 RAM 有较高的集成度，功耗比静态 RAM 还低，价格也比静态 RAM 的便宜。因为动态 RAM 是靠电容来存贮信息，为防止因泄漏电流而导致信息消失，所以要求每隔一段时间（如2ms）刷新（再生）一次。

二、ROM的种类

ROM 分为三类：一类是掩模 ROM。这种 ROM 是按固定线路制造的，制造好后就只能读不能改变。它适用于批量生产，成本较低，但不适用于研究工作。另一类 ROM 是可编程序的只读存贮器PROM(Programmable ROM)，这种ROM允许用户对它进行编程，但只能写一次。还有一类 ROM 是可擦去的 PROM (Erasable PROM)，即EPROM，它允许多次改写，但它写的速度较慢，且需要一些额外条件。

3-1-2 RAM 的结构 存贮器包括存贮体和外围电路两部分。图3-1表示 RAM 存贮器的示意图。

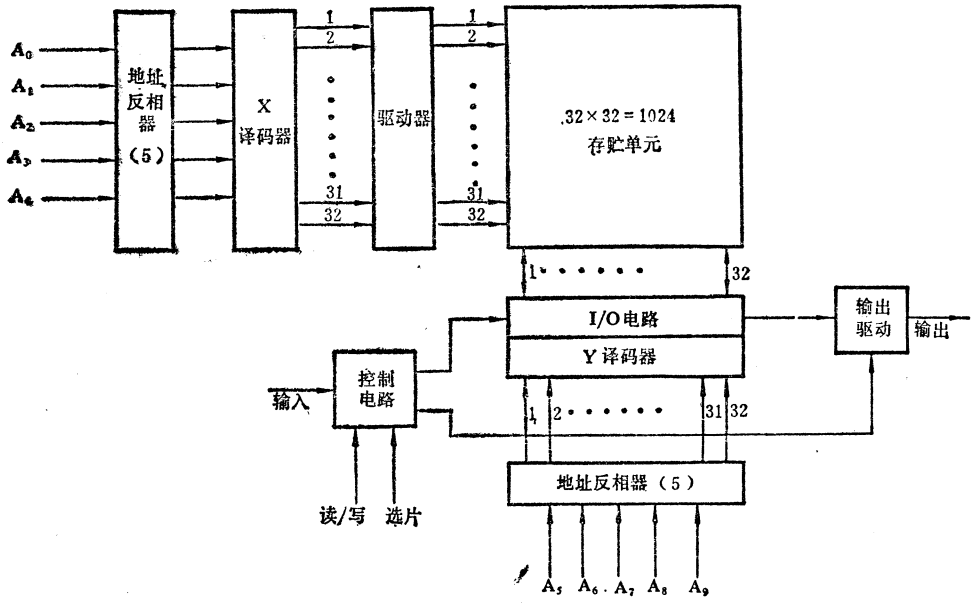


图3-1 典型的RAM示意图

一、存贮体

一个基本存贮电路可以表示一个二进制位，通常微型机要有32K或64K字节的存贮容量，所以需要32K×8或64K×8个基本存贮电路，显然存贮器中要有大量的存贮电路。这些存贮电路按一定规则组织起来，叫做存贮体，它是存贮信息的本体。为了很方便地访问每个存贮单元，在存贮器中，每个存贮单元都有一个编号——地址。所以可以用地址号来选择不同的存贮单元。

在容量较大的存贮器中，往往把各个字的同一位组织在一个片中，例如图3-1中的1024×1，它是1024个字的同一位。由这样的 8 个片子，可组成1024× 8 的存贮单元，不同字的同一位通常排成矩阵的形式，用X选择线—行线和Y选择线—列线的重叠来选择所需要的单元。

对于容量较小的存贮器，是把 RAM 芯片的单元阵列直接排成所需要位数的形式。这时每一条X选择线代表一个字，而每一条Y线代表字中的一位，所以常把X选择称为字线，而把Y选择线称为位线。

二、外围电路

外围电路通常有：

1.地址译码器　来自地址总线上的地址信号（对于64K的内存需16位的地址信号）经存贮器中地址译码器译码，以选择所要访问的内存单元。

地址译码方式有两种：一种是单译码方式或称字结构，字线选择某个字的所有位，这种方式适用于小容量的存贮器中；另一种是双译码结构，或称复合译码结构。在双译码结构中，地址译码器分成两个，X译码器和Y译码器，对于1024个字排成的32×32矩阵需要 10 根地址线A_0—A_9，A_0—A_4输至X译码器，它输出32条选择线分别选择1—32行；A_5—A_9输至Y译码器，它也输出32条选择线，分别选择1—32列，由行和列共同选中一个单元。在双译码结构中，一条X方向选择线要控制挂在它上面的所有存贮电路（如在1024×1中要控制32个电路），要带的负载很大，所以译码输出需要经过驱动器。

2.I/O电路　它处于数据总线和被选单元之间，用以控制被选中单元的读出或写入，并具有放大信号的作用。

3.片选控制CS(Chip Select)　一个存贮体由多个芯片组成。在进行地址选择时，首先要选片，用地址译码器输出和一些控制信号（如$\overline{\text{MREQ}}$）形成选片信号，只有当 CS 有效选中某一片时，此片所连的地址线才有效，才能对这一片上的存贮单元进行读或写的操作。

4.集电极开路或三状态输出缓冲器　连接多片 RAM 的数据线，或连接双向的数据总线。当连在数据总线上的某些存贮器或输入输出接口不用时，则与这些部分相连的三态缓冲器处于高阻状态，相当于和总线脱离了关系。

随着存贮器片子的容量和速度的增加、及体积的缩小，其内部结构也更加复杂。八十年代初期，动态 RAM 的最高集成度为64K(64K×1位)如TI4164，但到了1985年，予计美日等国将大量生产256K的DRAM。IBM公司已建立了大量生产256K DRAM 的体制，到85年年中将成为取代64K DRAM 的主流产品。Intel 公司已在84年开发成功CMOS型256K DRAM，日本的NEC、日立和富士通等公司都大量生产256K的 DRAM。 日本东芝公司公布了研制成功1兆位DRAM的科研成果，在 $4.78\times13.23mm^2$ 的硅片上集成 225 万个元件，存取时间为70ns。

3-1-3　RAM与CPU 的连接　在微型机中， CPU 对存贮器进行读写操作，首先要通过地址总线给出地址信号，然后发出读写控制信号，最后才能在数据总线上进行数据交换。显然，RAM与CPU的连接主要是地址线的连接、数据线的连接和控制线的连接。 在连接中要考虑以下几个问题：

1.CPU 总线的负载能力。现在的存贮器都为 MOS 电路，直流负载很小， 主要的负载是电容负载，故在小型系统中，CPU 可以直接与存贮器相连，而在较大的系统中， 就要考虑CPU能否带得动，需要时要加缓冲器，由缓冲器的输出再带负载。

2.CPU 的时序和存贮器的存取速度之间的配合问题。CPU 在取指和存贮器读、写操作时，是有固定时序的，由此来确定对存贮器的存取速度的要求。

3.存贮器的地址分配和选片问题。内存通常分为RAM和 ROM 两大部分，而 RAM 又分为系统区（即机器的监控程序或操作系统占用的区域）和用户区。所以内存的地址分配是个很重要的问题。另外，目前生产的存贮器，单片的容量仍然是有限的，所以总是要由许多片才能组成一个存贮器，对内存访问时需要解决选片问题。

4. 控制 信 号的连接。CPU 在与存贮器交换数据时，有以下几个控制信号 （对Z-80来说）：$\overline{\text{M}_1}$，$\overline{\text{MREQ}}$，$\overline{\text{RD}}$，$\overline{\text{WR}}$ 以及$\overline{\text{WAIT}}$信号，这些信号如何与存贮器要求的控制信号相连，以实现所需要的控制作用。

在实现RAM与CPU的连接时，根据RAM的不同结构可有不同的连接方式。例如1K位的存贮器芯片可具有1024×1位，256×4位和128×8位等不同结构，因此与八位微处理器相连时，可采用位并联或地址串联的方法来满足存贮体需要的容量和位数。

例如要组成1K×8位的存贮器，若采用1024×1位的片子， 地址线为10条， 有 8 片就可满足存贮体容量的要求。每一片相应于一位 （只有一条数据线），故只要把它们分别接到数据总线上的相应位即可，对片子没有选片要求。若片子有选片输入端 （$\overline{\text{CS}}$或 $\overline{\text{CE}}$），可把它们直接接至$\overline{\text{MREQ}}$。以这种方式联接时，每条地址总线有 8 个负载，而每一条数据总线只接有一个负载，每一片共有11个地址和数据引脚。

若选用256×4位的片子，每片上要 8 条地址线，共用 8 片。 总的存贮体容量 1K分成四

部分(或称为页)，所以，用地址总线上的A_0—A_7直接与各个片的地址输入端相连，可寻址256个单元，实现页内寻址；由A_8、A_9经过译码输出四条线，实现页的寻址。因为每一片上的数据为四位，用2片可组成一页，所以四条页寻址线，每条同时接两片。一页内两片的数据线，一个接到数据总线的D_0—D_3，另一个接到D_4—D_7。而各页的数据线并联接到数据总线上。

从负载角度和产品合格率等方面来看，后一种连接方式不如前一种，因此，在容量较大的存贮器中，通常采用一片一位的结构。

一、4KRAM的连接

当系统RAM的容量为4K或更大时，若用Intel2114 1K×4位的片子构成系统的存贮器，则需将片子分成4组(或更多)，A_{10}—A_{15}经过译码，可采用全译码方式，也可采用部分译码方式，如图3-2所示。

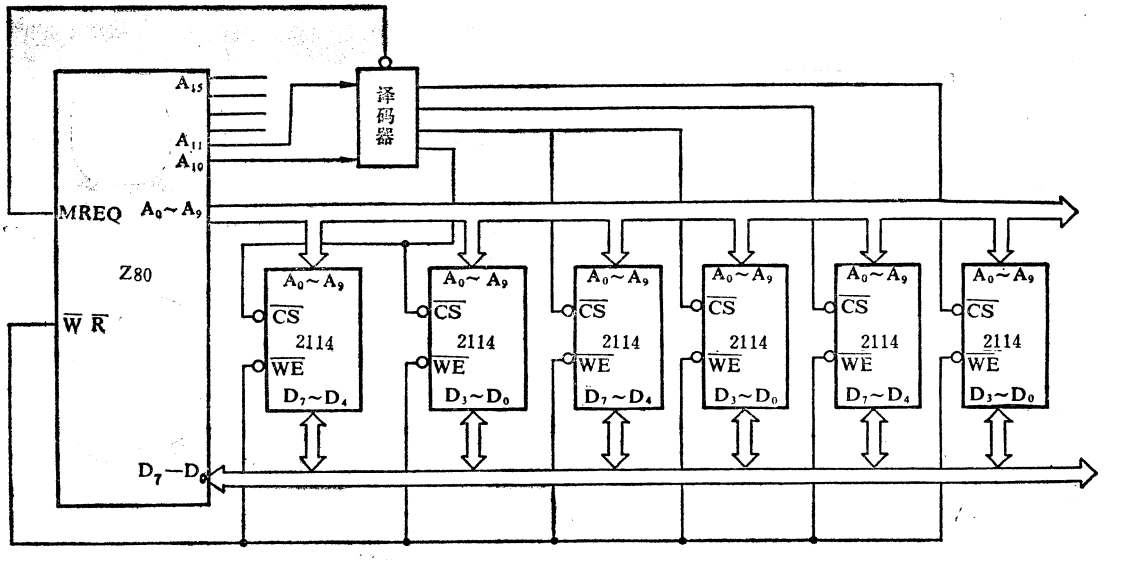


图3-2 4KRAM结构图

其中，A_0—A_9作为片内寻址，A_{10}、A_{11}经过译码作为组选择，其地址分布为：

第一组 0000—03FFH；

第二组 0400—07FFH；

第三组 0800—0BFFH；

第四组 0C00—0FFFH。

实际上A_{12}—A_{15}为任意值时仍可选中这几组，故每组有16K的地址重叠区(即每组占有16K地址，地址的最高位由0变到F都是重叠的范围)。显然，也可以用A_{10}—A_{15}中的任意两位经过译码作为组控制。这种用高位地址中的几位经过译码作为选片控制，这种方式称为部分译码方式。

总之，CPU的16条地址线可寻址64K，这么大容量的存贮器要由多个片子组成，则可由所选用的片子的字数分组。有一部分地址线（通常是用低位）连到所有片，实现片内寻址；另外一些地址线可组成译码器（部分译码或全译码），其输出控制片子的选片端（实际的选片

信号还要考虑 CPU 的控制信号，例如Z-80的$\overline{\text{MREQ}}$等），以实现组的寻址。在连接时要注意它们的地址分布和重叠区。

二、具有RAM和ROM的存贮器连接

通常的微型机系统的内存贮器中，总有相当容量的ROM，它们的地址必须与RAM一起分配，分别给它们一定的地址。图3-3是一个用8080和由8708（1024×8位）组成的 4KROM 和由Intel 2114组成的1K RAM 的方框图。它用A_0—A_9作为组内寻址，由A_{10}、A_{11}、A_{12}组成译码器(部分译码)，实现组寻址。ROM的地址为0000—0FFFH，共4K；RAM的地址为1000—13FFH，共1K。它们都有相当大的地址重叠区。

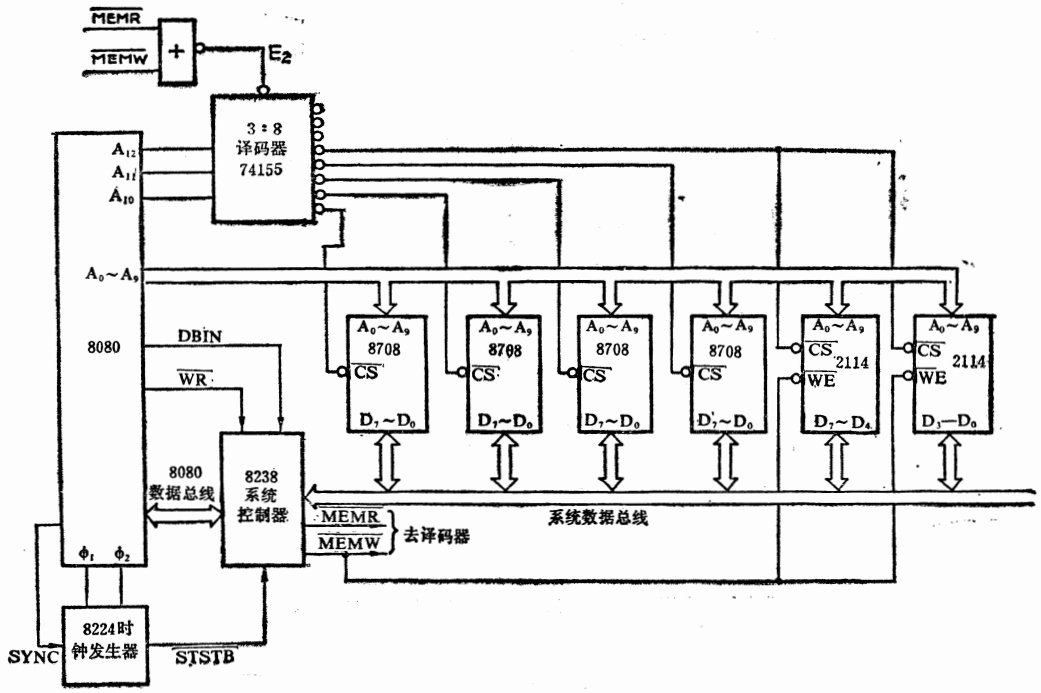


图3-3 具有RAM和ROM的系统方框图

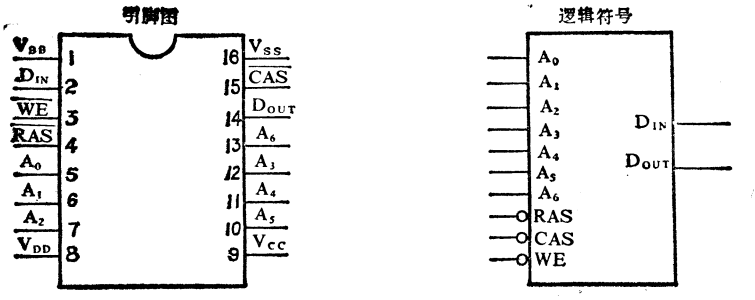
三、16K动态RAM存贮器片与CPU的连接

动态 RAM 的平均功耗小、集成度高、价格便宜，因此在大容量存贮器中得到广泛应用。这里仅以容量为16K×1位的Intel 2116 为例，扼要说明动态RAM与CPU的连接。

1．Intel 2116的结构

Intel 2116的逻辑图和引脚如图3-4所示。每一片的容量为16K×1位，是16脚封装，地址线分成两部分：行地址和列地址，存贮单元排成128×128的矩阵。地址引线只有七条，内部设有地址锁存器，由行地址选通信号$\overline{\text{RAS}}$ (Row Address Strobe)，把开始出现的 7 位地址送至行地址锁存器，由随后出现的列地址选通信号$\overline{\text{CAS}}$(Column Address Strobe) ，把后出现的 7 位列地址送至列地址锁存器。此外，7条地址线还用作刷新地址，刷新时地址计数，实现一行一行刷新。

7位行地址经过译码，产生128条选择线，分别选择128行；7位列地址经过译码，产生128条选择线，分别选择128列。由行选择和列选择确定被选中的存贮单元。



引 腿 名			
A_0～A_6	地址输入	$\overline{WE}$	写地址
$\overline{CAS}$	列地址选通	V_{BB}	电源(-5V)
D_{IN}	数据输入	V_{CC}	电源(+5V)
D_{OUT}	数据输出	V_{DD}	电源(+12V)
$\overline{RAS}$	行地址选通	V_{SS}	地

图3-4 Intel 2116逻辑图

2．动态RAM与Z-80的接口

由于16K动态 RAM用$\overline{RAS}$和$\overline{CAS}$来分别选通和锁存行地址和列地址，在行地址中，又要能接入刷新地址，所以必须有多路转换器来控制。接口电路的方框图如图3-5所示。

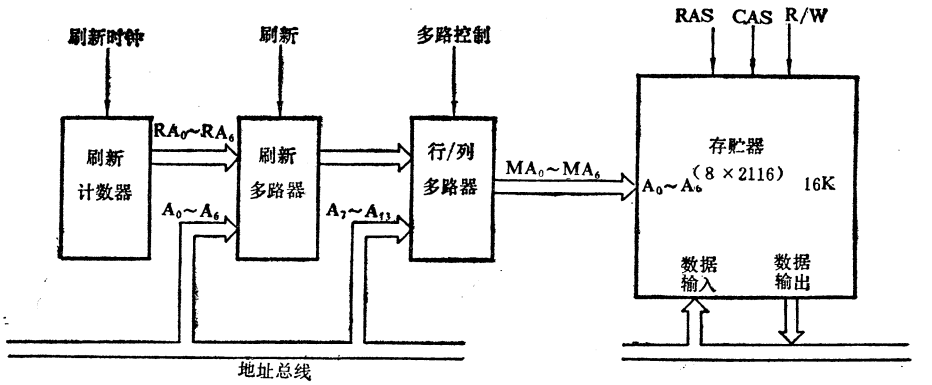


图3-5 动态RAM与Z-80的接口电路

行地址——地址总线上的 A_0—A_6 和刷新计数器（Z-80中的R寄存器）的7位地址通过一个多路开关输至行/列多路器，只有在刷新时才为刷新地址，否则为行地址。列地址—地址总线上的A_7—A_{13}也送到行/列多路器，由多路器控制输出的7位地址是行地址还是列地址（**按时间先后**）。然后送至存贮器的地址线，由选通信号$\overline{RAS}$和$\overline{CAS}$选通送至各自的地址锁存器，以实现读和写。

3-2 输入和输出

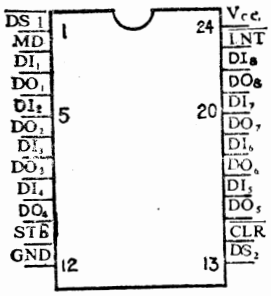
CPU 从外设输入信号或输出信号给外设，可以采用程序查询方式，中断的方式和DMA方式。但是，不论哪一种方式，CPU总是通过接口电路才能与外设连接。

3-2-1 通用I/O接口 Intel 8212 是一个 8 位的输入输出接口片子，是典型的通用I/O

接口。它由8位锁存器、带有三态输出的缓冲器以及控制和选择逻辑电路等组成，它还包括中断请求逻辑。其内部结构逻辑如图3-6所示。

数据锁存器是由8个D触发器组成，用以暂存数据。当时钟输入端(C)为高电平时，触发器的输出(Q)跟随数据输入（DI）变化，当时钟端（C）返回低电平时，保持锁存作用，即Q端的信号保持不变。数据锁存器可用复位输入端$\overline{CLR}$来清除。

数据锁存器的输出（Q端）接三态输出缓冲器。缓冲器有一条公共控制线(允许线EN)，当控制线为高电平时，打开输出缓冲器；当控制线为低电平时，缓冲器输出为第三态（高阻抗）。



引脚名

$DI_1 \sim DI_8$	数据输入
$DO_1 \sim DO_8$	数据输出
$\overline{DS_1}$, DS_2	装置选择
MD	模　式
STB	选　通
$\overline{INT}$	中断(低电平有效)
$\overline{CLR}$	清"零"(低电平有效)

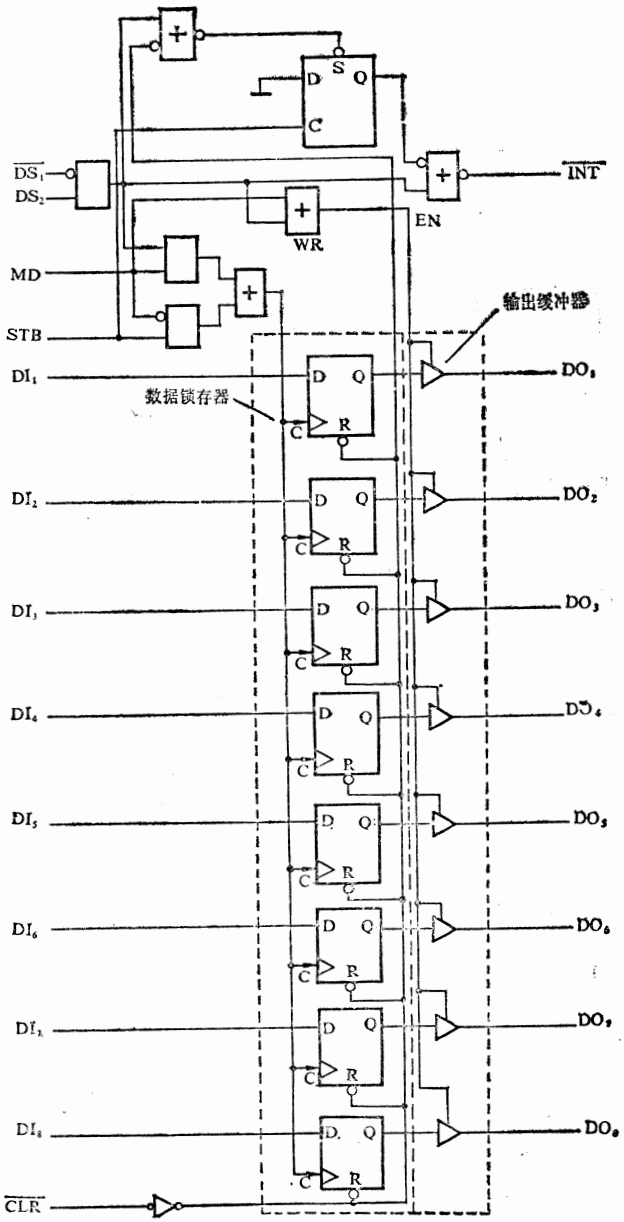


图3-6　8212内部结构逻辑图

控制逻辑包括控制输入$\overline{DS1}$、DS2、MD及STB。$\overline{DS1}$和DS2用于装置选择，MD（模式）用来控制输出缓冲器的状态，并决定那个信号通到数据锁存器的时钟输入端（C）。STB（选通），这个输入信号通向数据锁存器（MD＝0时）的时钟端以及中断请求触发器的时钟端C，使中断请求信号$\overline{INT}$＝0。

3-2-2 可编程序并联接口 8255A 可编程序接口是指其功能可由微处理机的指令来加以改变的接口设备。利用编程序的方法，可使一个接口片子执行多种不同的接口功能，因此使用十分灵活。Intel 8255A是一个为8080和8085微处理机系统（也可适用于Z-80系统）设计的通用并联I/O接口片子。

图3-7是Intel 8255A的结构框图。

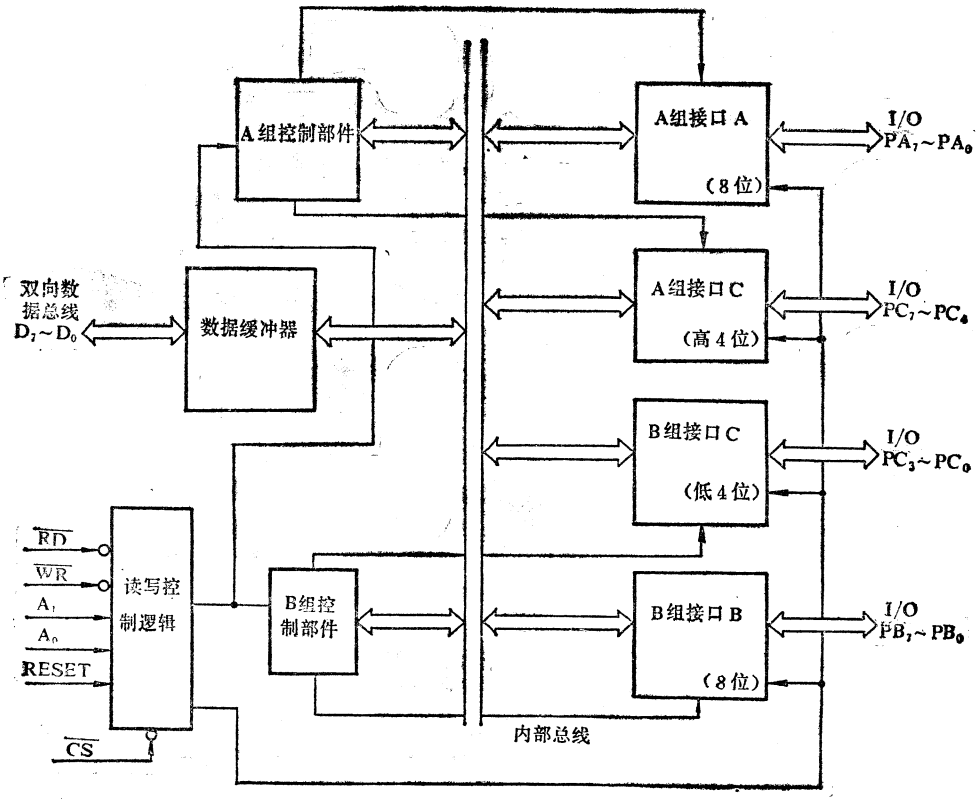


图3-7 8255A结构框图

8255A由以下几部分组成：

一、接口（或称端口）A、B、C

端口A、B、C各有8位，都可以选择作为输入或输出，它们通过内部总线与数据缓冲器、A和B接口的控制电路、读写控制逻辑相连，再由数据缓冲器及读写控制逻辑连到CPU总线上。

端口C除作数据端口外，还可分成为二个各有4位的独立端口，分别与端口A和B配合使用，以寄存数据传送所需要的状态和控制信息。

二、A组和B组控制部件

这是两组根据CPU的命令字，控制8255A工作方式的电路。它们有控制寄存器，接受CPU的命令字，然后分别决定接口A，B的工作方式，也可根据CPU的命令字对端口C的每一位实现按位“复位”或“置位”。

A组控制部件控制端口A和端口C的上半部(PC_7—PC_4)，B组控制部件控制端口B和C的下半部(PC_3—PC_0)。

三、数据总线缓冲器

这是一个三态双向8位缓冲器，它是8255A与系统数据总线的接口。

四、读写控制逻辑

它与接收来自CPU的地址总线中的A_1、A_0以及有关的控制信号($\overline{RD}$、$\overline{WR}$、$\overline{CS}$、RESET)控制接口A、B、C的操作。由A_0A_1的不同组合选择接口A、B或C，其编址方式列于表3-1。$A_1A_0=00$时，选择接口A；$A_1A_0=01$时，为接口B；$A_1A_0=10$时为C。$\overline{RD}=0$时为读，所选择接口的数据经数据缓冲器送入CPU。$\overline{WR}=0$时为写，数据从缓冲器写入所选择的接口。$\overline{CS}=0$时，表示该片被选中，可以工作。

五、8255A有三种基本的工作方式：

1.方式0：基本输入输出操作，它不需要在8255A和外部设备之间交换状态和控制信息，即可进行数据传送。

2.方式1：选通输入输出。它分为A、B两组：A组由数据口A和4位控制口C(高4位)组成；B组由数据口B和4位控制口C（低4位）组成。

3.方式2：选通双向传送。

8255A的工作方式，由CPU写入8255A的控制字寄存器的控制命令字来选择。控制命令字的格式如图3-8所示。可分别选择端口A和端口B的工作方式，端口C分成两部分，高位随端口A，低位随端口B。端口A有方式0、1和2三种，而端口B只能工作于方式0和1。

表3—1　　8255A端口选择表

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	操　作
0	0	0	1	0	读端口A
0	1	0	1	0	读端口B
1	0	0	1	0	读端口C
0	0	1	0	0	写入端口A
0	1	1	0	0	写入端口B
1	0	1	0	0	写入端口C
1	1	1	0	0	写入控制命令字
×	×	×	×	1	数据总线呈三态

3-2-3　**可编程序串行接口**　当CPU与外设进行串行通讯时，若用软件实现由串行到并行，以及由并行到串行的转换，则将大大降低CPU的利用率。所以，通常是用硬件UART电路来实现接口。随着大规模集成电路技术的发展，通用的可编程序的同步和异步的接口片子USART (Universal Synchronous Asynchrous Receiver/Transmitter)种类越来越多。典型的有Motorola的ACIA，Intel的8251和Zilog的SIO等等。但是，它们的基本功能是类似的。

Intel 8251A是一种通用可编程序串行接口，它能把CPU来的并行数据变为串行数据发送给外设，或者接收来自外设的串行数据变为并行数据输入到CPU中。接收或发送每一个数据字的时间间隔可以是相等的，即同步传送，也可以是不相等的，即异步传送。图3-9是Intel 8251A的结构框图。

图3-9中各符号的名称是：

CLK(Clock)

C/$\overline{D}$(Control/Data)

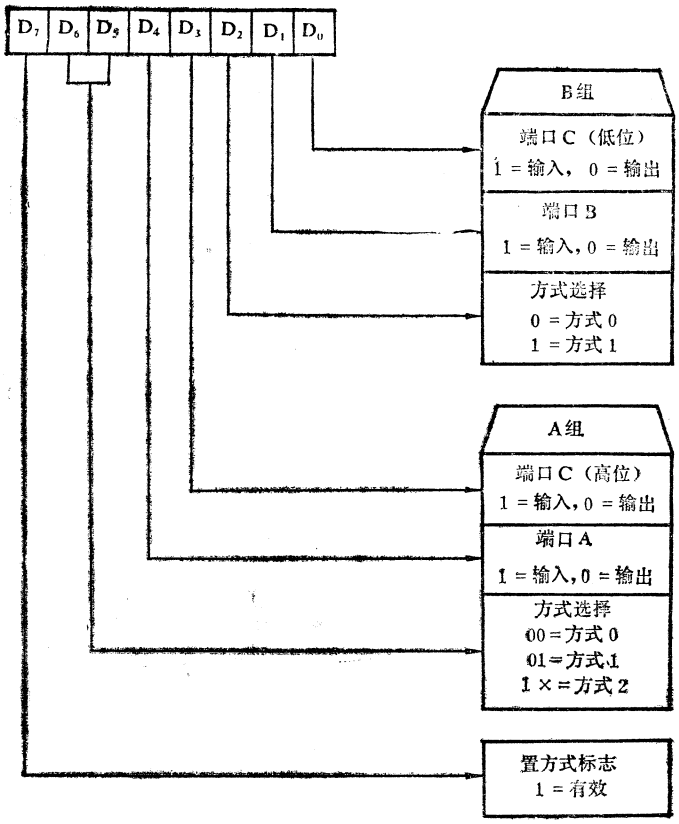


图3-8　8255A的控制字

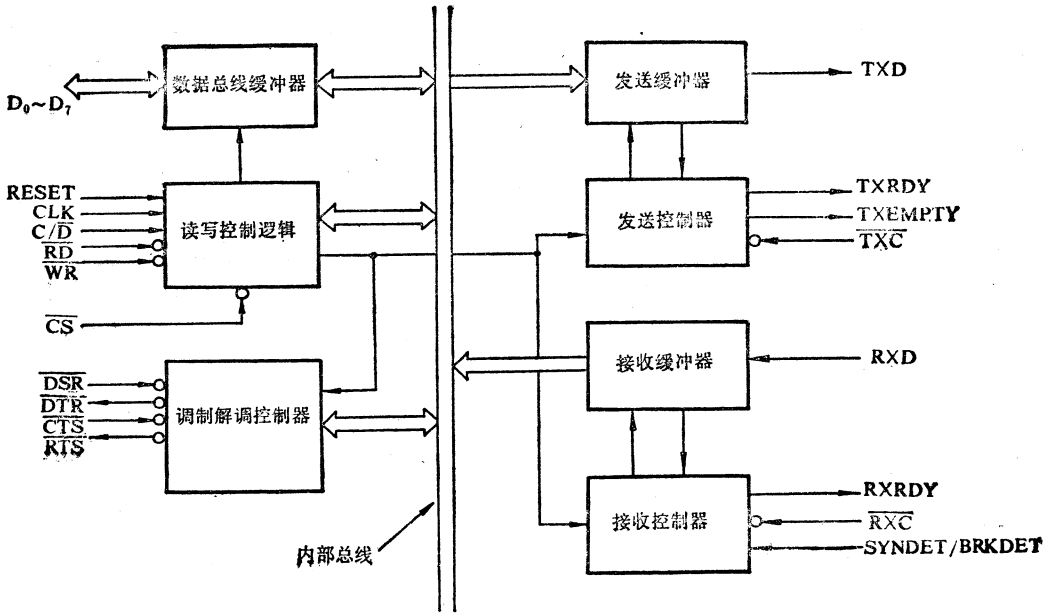


图3-9　8251A结构框图

$\overline{CS}$ (Chip Select)

$\overline{DSR}$(Data Set Ready)

$\overline{DTR}$(Data Terminal Ready)

$\overline{CTS}$ (Clear to Send)

$\overline{RTS}$ (Request to Send)

T×D(Transmitter Data)

T×RDY(Transmitter Ready)

T×EMPTY(Transmitter Empty)

$\overline{T\times C}$ (Transmitter Clock)

R×D(Receiver Data)

R×RDY(Receiver Ready)

$\overline{R\times C}$(Receiver Clock)

SYNDET(SYNC Detect)

BRKDET(Broak Detect)

Intel 8251A由五部分组成：

一、数据缓冲器

它是一个8位的双向缓冲器，三态输出。它是8251 A与系统数据总线的接口，数据、控制命令及状态信息均通过此接口传送。

二、发送缓冲器与发送控制器

发送缓冲器用来锁存待发送的数据，把数据由并行变成串行，从T×D输出端串行发送出去。发送速率由送入发送控制器时钟端$\overline{T\times C}$ 的时钟脉冲频率决定。对于同步传送，传输速率=$\overline{T\times C}$时钟频率。对于异步传送，传输速率可以等于$\overline{T\times C}$时钟频率或它的1/16或1/64，由编程来选取。发送缓冲器的工作还受发送控制器的两个状态信息控制，即发送器就绪(T×RDY) 和发送器空 (T×EMPTY)。这二个状态信息送到CPU，CPU 通过对这两个状态信息的检测来控制数据的发送。若T×RDY=1，表示发送器已准备好接收一个数据字，此时数据缓冲器已空，可以接收CPU送来的数据。T×EMPTY=1，表示发送缓冲器中的各位数据已全部发送完毕，即发送器已空。

三、接收缓冲器和接收控制器

接收缓冲器由接受端R×D接收外设送来的串行数据使之变成并行数据，再经数据缓冲器送给CPU。接收数据的速率取决于送入接受控制器时钟端$\overline{R\times C}$的时钟脉冲频率。此外它还能逐位或按字符进行校验。接收缓冲器的工作还受接收控制器中两个状态信息的控制，它们是接收器就绪 (R×RDY) 和同步检测/断缺检测 (SYNDET/BRKDET)。

R×RDY=1，表示8251A已准备好接收一个字符。

在作内同步操作时，SYNDET是输出端。SYNDET=1表示8251A已找到同步字符SYNC，从这时开始作内同步数据传送。在作外同步操作时，SYNDET是输入端，外同步信号经此输入端送入以实现同步传送。

作断缺检测时，BRKDET是输出端，它仅适合异步工作方式。BRKDET=1，表示在异步传送的数据中接收到一个全“0”字符，即没有数据可接收。BRKDET=1的状态将一直保持到一个新的数据字输入为止，即当R×D=1，开始一个新的数据字输入的启动信号将使其复

位。BRKDET信息送CPU作异步传送控制用。

四、读写控制逻辑

它接收来自CPU的控制信号及控制命令字（包括工作方式指令和控制方式指令），控制8251其余各组成部分的正常工作。它有6个输入端：

1.RESET是总清端。RESET=1，8251不工作，处于“闲置”状态。

2.CLK是时钟输入端，它提供8251内部控制所需的时钟信号。对于同步传送，要求CLK的频率比$\overline{T\times C}$和$\overline{R\times C}$大30倍；对于异步传送，要求大4.5倍。

3.$\overline{RD}$读信号输入端，$\overline{RD}=0$时为读。

4.$\overline{WR}$写信号输入端，$\overline{WR}=0$时为写。

5. C/$\overline{D}$用来表明读写的信息是数据还是控制信息或状态信息。C/$\overline{D}=1$时，是读状态信息或写控制命令，C/$\overline{D}=0$时，是读或写数据。

6.$\overline{CS}$选片端，连接到地址译码器上。$\overline{CS}=0$，表示此8251芯片被选中。CPU可通过“写”操作指令把待发送的数据或控制命令字写入8251A。通过“读”操作指令可以取出8251A接收的数据或读出各种状态控制信息。表2-6是8251A的编址真值表。表中ϕ表示可以是0或1。

表2—6　　8251A编址真值表

C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	操　　作
0	0	1	0	读数据字
0	1	0	0	写数据字
1	0	1	0	读状态信息
1	1	0	0	写控制命令字
ϕ	ϕ	ϕ	1	数据总线为高阻抗状态

五、调制解调控制器

当8251A作远距离传送通讯接口而与调制解调器相连时，用此控制器产生所需的控制信号：

1.$\overline{DSR}$ (Data Set Ready) 数据设置准备好(输入)。它是调制解调器送到调制解调控制器的信号，$\overline{DSR}=0$，表示调制解调器的数据已准备好。

2.$\overline{DTR}$ (Data Terminal Ready) 数据终端准备好（输出)。它是调制解调控制器输出到调制解调器的控制信号，$\overline{DTR}=0$，表示CPU要求终端设备进行数据通讯，因此可把$\overline{DSR}$看作是$\overline{DTR}$的回答信号。

3.$\overline{RTS}$ (Request to send) 请求发送(输出端)。$\overline{RTS}=0$，表示CPU已准备好发送。

4.$\overline{CTS}$ (Clear to Send) 清除发送请求信号(输入)。它是调制解调器或其它外设送到调制解调控制器的信号。$\overline{CTS}=0$，表示允许8251A向外发送数据。这个信号是对$\overline{RTS}$的应答信号。

8251A是可编程的多功能通讯接口，所以在具体使用时必须对它进行初始化编程，确定它的具体工作方式。这一点是通过CPU把工作方式指令写入8251A，用以规定8251A的工作方式，例如，是异步传送还是同步传送、传输速率、字符长度、奇偶性、同步符号个数等等。此外，为了使8251A能正确无误地实现数据传送，CPU还必须把控制指令字写入8251A，并通过读状态操作读取各种状态信息，用以控制8251A的数据传送。

3-2-4　Z-80 PIO　Z-80 PIO（并行I/O电路）是Z-80的并行接口芯片。它可以使外部设备如键盘、纸带读入机、打印机等，通过PIO和Z-80 CPU相连。

一、PIO的结构

PIO的结构如图3-10所示。

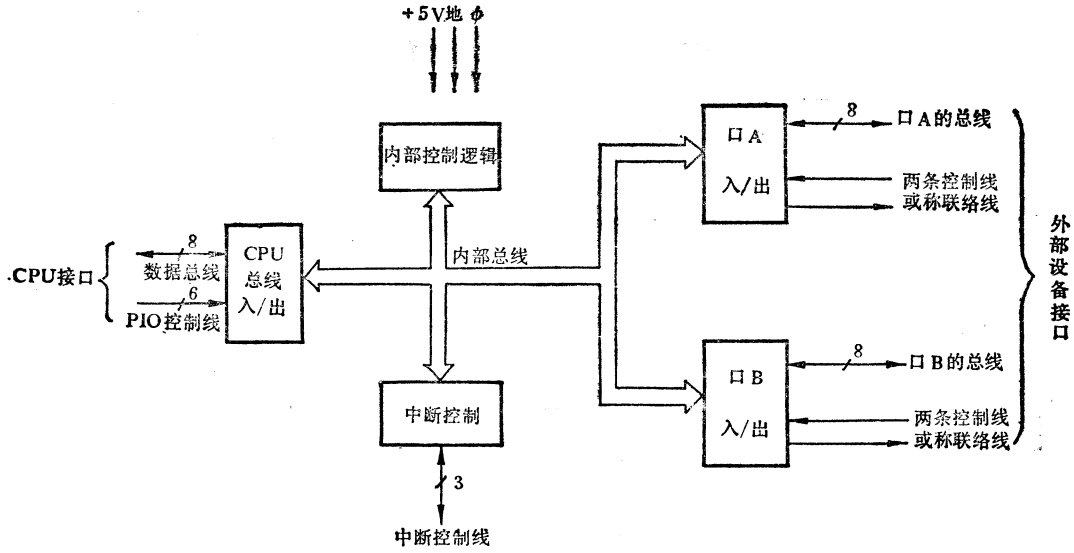


图3-10 PIO方框图

Z-80 CPU的8条数据总线（DB）和6条控制线接到PIO的“CPU总线入/出口”。口A和口B各有8条总线、2条控制线和外设相连。当CPU输出数据时，先送到口A或口B的8位数据输出寄存器中，再输出到外设。当外部设备的数据输入时，先送到口A或口B的8位输入寄存器中，再送到CPU去。每一个口有两条控制线，用来控制PIO与外部设备之间的数据传送。

PIO的口A或口B可选用下列四种不同工作方式之一。

1.CPU输出的字节通过PIO送给外部设备。

2.外部设备的字节通过PIO输入到CPU。

3.口A可用作双向入/出口。口B没有这一功能。

4.可以按“位控方式”工作。这时口A或口B的8条总线（8位）可由程序指定那几位作为输入口、那几位作为输出口。

利用PIO接口时，外部设备与CPU之间传送数据是在中断控制下进行的。PIO有下列处理中断的功能：当几个外部设备同时提出中断请求时，PIO能判断现在那一个外部设备的优先权最高。并能自动提供中断向量（地址）而不要外加逻辑电路。

二、PIO的片脚及其功能

Z-80 PIO采用双列直插式封装，有40个片脚（见图3-11），各片脚的功能是：

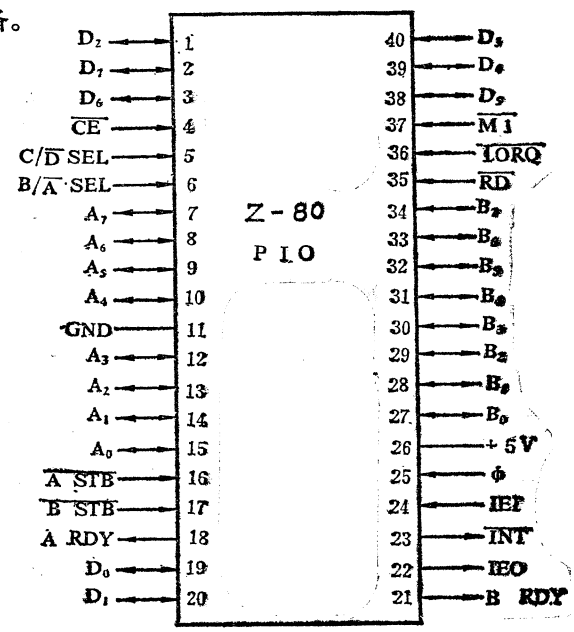


图3-11 Z-80 PIO 片脚图

1.D_0～D_7 数据总线（双向、三态）

D_0～D_7用来传送Z-80 CPU和PIO之间的数据或命令。

2. B/$\overline{A}$SEL 选择口B或口A(输入)

如这一片脚为高电平，表示选中口B。如为低电平，表示选中口A。通常用CPU的地址线A_0作为这一选择信号。

3. C/$\overline{D}$ SEL 传送的信息是控制命令还是数据的选择信号(输入)

如这一片脚为高电平，表示CPU正在送给PIO的信息是一个命令；如为低电平表示 CPU和PIO之间正在传送的信息是数据。

4.$\overline{CE}$ 选片信号（输入、低电平有效)

如这一片脚有低电平信号输入，则该PIO芯片被选中。

5.ϕ时钟信号（输入)

Z-80 PIO用标准的Z-80系统时钟信号来同步其内部的某些信号。

6.$\overline{M1}$ 从CPU来的第1个机器周期信号（输入、低电平有效)

7.$\overline{IORQ}$ Z-80 CPU来的输入/输出请求信号（输入、低电平有效)

8.$\overline{RD}$ Z-80 CPU来的读周期状态(输入、低电平有效)

信号$\overline{M1}$、$\overline{IORQ}$、$\overline{RD}$的不同组合，使PIO有不同的功能，如下表所列。

表3—2 PIO的功能与控制信号的关系

信号			PIO的 功 能
$\overline{M1}$	$\overline{IORQ}$	$\overline{RD}$	
0	0	0	没有功能
0	0	1	中断应答
0	1	0	检查中断服务程序的终止
0	1	1	复 位
1	0	0	CPU从PIO读入信息
1	0	1	CPU把信息写入PIO
1	1	0	没有功能
1	1	1	没有功能

9.IEI 中断允许入 (Interrupt enable in)（输入，高电平有效)

10. IEO 中断允许出 (Interrupt enable out)（输出，高电平有效)

当系统中有多个PIO时，优先级最高的PIO的IEI端接+5伏电源，它的 IEO端和下一个中断优先级的PIO的IEI端相连。根据中断优先级的次序按照这种方法把各PIO的IEI和IEO 端连成一个中断链，如有一个以上的PIO同时提出中断请求，CPU首先为其中优先级最高的PIO服务。

11.$\overline{INT}$ 中断请求(输出，低电平有效)。

PIO向CPU发出的中断请求信号。

12.A_0～A_7 端口A的总线(双向，三态)。

用来传送PIO的端口A和外设之间的数据或控制信息。

13. $\overline{\text{A STB}}$ 外设送给端口A的选通脉冲(输入，低电平有效)。

14. A RDY 寄存器A就绪（输出，高电平有效)。

15. $B_0 \sim B_7$ 端口B的总线(双向，三态)。

16. $\overline{\text{B STB}}$ 外设送给端口B的选通脉冲（输入，低电平有效)。

17. B RDY 寄存器B就绪（输出，高电平有效)。

三、PIO编程

(一)复位

接通电源，Z-80PIO自动进入复位状态。复位状态完成以下功能：

1. 两个端口（口A和口B）的屏蔽寄存器被置为高电位，使端口的全部数据线被屏蔽；

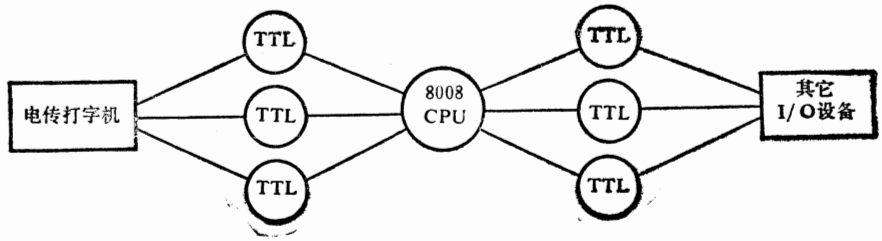
2. 端口的数据总线$A_0 \sim A_7$和$B_0 \sim B_7$被置成高阻状态，就绪信号A RDY 和 B RDY 不起作用(低电平)；

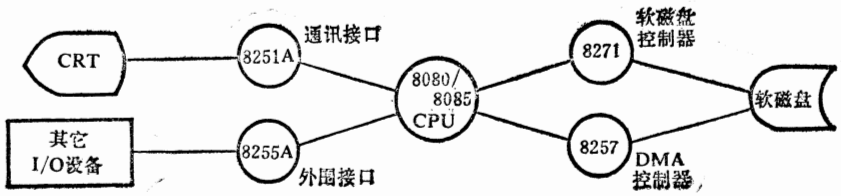
3.中断矢量地址寄存器未被复位；

4.两个端口的中断允许触发器被复位；

5.两个端口的输出寄存器被复位。

此外，在$\overline{M1}=0$，$\overline{RD}$和$\overline{IORD}$为1时，PIO也进入复位状态，执行上述功能。





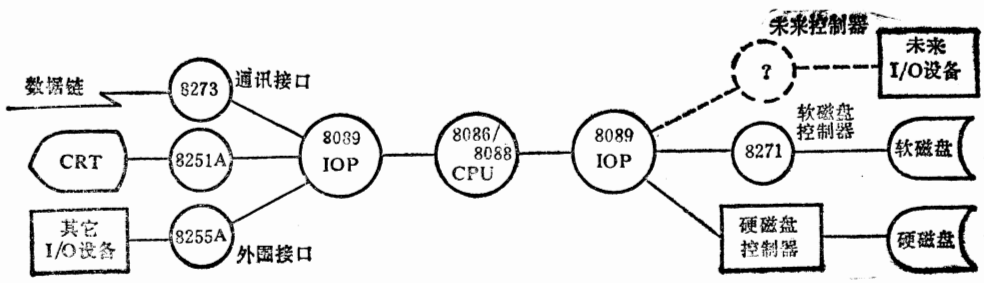


图3-12 IOP的演变

一旦PIO进入复位状态，就一直保持，直到收到CPU来的一个控制字为止。

(二)装入中断矢量

PIO按Mode2中断，在初始化程序时，应输入一个中断矢量，以便当中断响应时，能向CPU输出这个中断矢量。

(三)工作方式选择

PIO中口A和口B的工作方式，在PIO初始化编程时确定。口A可工作于下述四种工作方式：

方式0（输出方式）；

方式1（输入方式）；

方式2（双向方式）；

方式3（位控方式）。

口B除了无方式2（双向方式）外，具有上述的另外三种工作方式。

(四)设置中断控制字

PIO是以中断方式与CPU交换信息的，必须在初始化编程时，设置中断控制字。

3-2-5 8089输入/输出处理器 图3-12展示出最初三代微处理机中CPU和I/O设备的关系的发展趋势。第一代，CPU与大量的TTL电路打交道，常常执行逐位的传送。因此，仅能支持十分有限的相当慢的外设。

在第二代微处理机中，引入了一些单片接口控制器，这些器件使CPU摆脱了最低级的外设控制工作，并使CPU一次能传送整字节。随着DMA控制器的引入，可将高速外设加入系统，并能传送整个数据块而不需CPU的干预。I/O设备控制器和DMA控制器使处理机能够适应支持外设数量和传送速度方面达到中等水平的I/O的需要。但是，CPU对这些控制器还是进行相当多的管理工作。

8089引入了第三代输入/输出处理。它消除了CPU的另一级的I/O控制，进一步简化了CPU管理I/O设备的工作。

8089输入/输出处理器是高性能的单片通用I/O系统，它采用40条引脚的双列直插封装(见图3-13)。

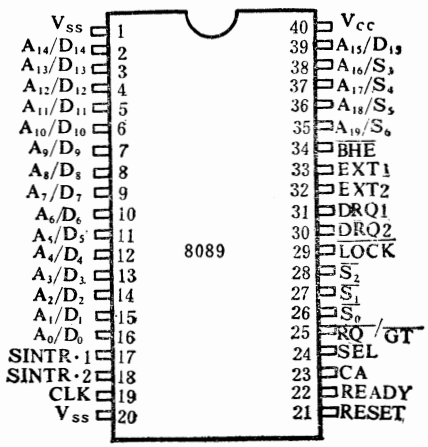


图3-13 8089输入/输出处理机引脚图

8089是8086系列的重要组成部分，在它的内部有两个独立的I/O通道，每一个通道都兼有CPU功能和非常灵活的DMA（直接访存）控制器的功能。例如，能像CPU一样执行程序，IOP的指令系统大约有50种为输入/输出处理而专门设计的指令；每个通道也能完成高速DMA传送等。

第四章 指令系统

指令系统是机器所具有的各种指令的集合，或称指令集。指令系统表征着计算机的基本功能，不同的计算机，它们的指令系统也不相同。指令系统所包括的指令类型和功能，在某

种程度上反映了机器的处理能力。下面分别列出几种微处理器的指令系统。

4-1 Intel 8080A指令系统

4-1-1 指令格式 对于一条指令，它应包括两部分的内容。一是操作码，它表示一条指令执行什么操作，如加或移位等；二是地址码，用来指出操作数或操作数地址。

微型计算机的指令有单字节、双字节、三字节等不同长度的格式。Intel 8080A的指令格式如下所述：

一、单字节指令

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	操作码

二、双字节指令

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
第一字节	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	操作码
第二字节	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	操作数

三、三字节指令

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
第一字节	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	操作码
第二字节	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	低位地址
第三字节	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	高位地址

4-1-2 寻址方式 指令的寻址方式是指出如何得到操作数地址。Intel 8080A 有如下几种寻址方式：

一、直接寻址

采用直接寻址时，指令的第二、第三字节指明操作数的地址。

二、立即寻址

立即寻址方式的指令，它的第一字节为操作码，第二字节(8位）或第二和第三字节（共16位）直接给出操作数。

三、寄存器直接寻址

这种寻址方式把操作数事先存放在某个寄存器中，在指令中给出这个寄存器的地址。

四、寄存器间接寻址

这种寻址方式中，指令提供一个寄存器对的地址，这个寄存器对的内容是操作数的16位地址。

五、堆栈指示器寻址

在8080A 中，堆栈指示器寻址只是在转子程序中或中断程序中，用于保存返回地址和寄存器的内容。

4-1-3 Intel 8080A指令系统 根据指令的功能，8080A的指令可分为以下几类：

一、数据传送类

数据传送类指令，使数据从一个寄存器传送到另一个寄存器，或在寄存器和存贮器之间进行传送，或者把一个数据直接传送到寄存器中。

二、算术运算类

这一类指令包括累加器中的数和寄存器中的数、或存贮器中的数、或立即数相加或相减；相加时可以不带进位也可以带进位；相减时可以带或不带进位减；寄存器或存贮单元中的内容加1或减1等等。

三、逻辑类

它包括累加器中的数和寄存器或存贮器中的数执行各种逻辑操作。

四、转移类

它包括条件和无条件的调用、转移和返回指令。

五、堆栈、输入/输出及计算机控制类

这类指令包括堆栈的压入和弹出指令、输入/输出指令、允许和不允许中断指令、暂停、空操作指令等。

表4-1列出了8080和8080A的指令系统。

4-2 Z-80指令系统

4-2-1 Z-80的寻址方式 Z-80的寻址方式有：

一、立即寻址

在这种寻址方式中，指令中跟在操作码（可以是一个或二个字节）后面的一个字节的内容就是实际操作数。

二、立即扩展寻址

在这种寻址方式中，指令的操作码可以是一个或二个字节，指令中的立即数为两个字节，形成一个16位的操作数或16位的地址。

三、寄存器寻址

这种寻址方式是指定CPU中的某一寄存器的内容为操作数。

四、扩展寻址

扩展寻址也称为直接寻址。在扩展寻址的指令中，操作码后面给出的二个字节组成的16位地址，可以是程序的转移地址，也可以是存放操作数的存贮单元地址。

五、寄存器间接寻址

指令指定CPU的某一寄存器对的内容作为操作数的地址。

六、变址寻址

变址寻址的指令中，由指令指定的变址寄存器（IX或IY）的内容和指令中给定的偏移量相加，所得的结果作为操作数的地址。

七、零页寻址

这是一个单字节指令，它指定零页（单元0000H～00FFH为零页）中的某些规定的单元作为子程序入口地址。例如重新启动指令RST P是一个单字指令，它的二进制形式为：

1	1	N	N	N	1	1	1

NNN构成0～7之间的整数，根据N的值，指令调用规定入口处的子程序。

八、相对寻址

相对寻址指令的第一个字节为操作码，第二个字节为位移量，它以程序计数器PC的内容作为基础，加上指令中给定的位移量作为转移地址。

记忆符	说　明	指令代码〔1〕 D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	时钟周期〔2〕
MOV r_1 ,r_2	寄存器送寄存器	0	1	D	D	D	S	S	S	5
MOV M, r	寄存器送存储器	0	1	1	1	0	S	S	S	7
MOV r, M	存储器送寄存器	0	1	D	D	D	1	1	0	7
HLT	暂　　停	0	1	1	1	0	1	1	0	7
MVI r	立即数送寄存器	0	0	D	D	D	1	1	0	7
MVI M	立即数送存储器	0	0	1	1	0	1	1	0	10
INR r	寄存器加 1	0	0	D	D	D	1	0	0	5
DCR r	寄存器减 1	0	0	D	D	D	1	0	1	5
INR M	存储器加 1	0	0	1	1	0	1	0	0	10
DCR M	存储器减 1	0	0	1	1	0	1	0	1	10
ADD r	加寄存器到累加器	1	0	0	0	0	S	S	S	4
ADC r	加寄存器到累加器(连进位)	1	0	0	0	1	S	S	S	4
SUB r	累加器减寄存器	1	0	0	1	0	S	S	S	4
SBB r	累加器减寄存器(连借位)	1	0	0	1	1	S	S	S	4
ANA r	寄存器“与”累加器	1	0	1	0	0	S	S	S	4
XRA r	寄存器“异或”累加器	1	0	1	0	1	S	S	S	4
ORA r	寄存器“或”累加器	1	0	1	1	0	S	S	S	4
CMP r	寄存器和累加器比较	1	0	1	1	1	S	S	S	4
ADD M	加存储器至累加器	1	0	0	0	0	1	1	0	7
ADC M	加存储器至累加器(连进位)	1	0	0	0	1	1	1	0	7
SUB M	累加器减存储器	1	0	0	1	0	1	1	0	7
SBB M	累加器减存储器(连借位)	1	0	0	1	1	1	1	0	7
ANA M	存储器“与”累加器	1	0	1	0	0	1	1	0	7
XRA M	存储器“异或”累加器	1	0	1	0	1	1	1	0	7
ORA M	存储器“或”累加器	1	0	1	1	0	1	1	0	7
CMP M	存储器和累加器比较	1	0	1	1	1	1	1	0	7
ADI data	立即数加至累加器	1	1	0	0	0	1	1	0	7
ACI data	立即数加至累加器(连进位)	1	1	0	0	1	1	1	0	7
SUI data	累加器减立即数	1	1	0	1	0	1	1	0	7
SBI data	累加器减立即数(连借位)	1	1	0	1	1	1	1	0	7
ANI data	立即数“与”累加器	1	1	1	0	0	1	1	0	7
XRI data	立即数“异或”累加器	1	1	1	0	1	1	1	0	7
ORI data	立即数“或”累加器	1	1	1	1	0	1	1	0	7
CPI data	立即数和累加器比较	1	1	1	1	1	1	1	0	7
RLC	累加器循环左移	0	0	0	0	0	1	1	1	4
RRC	累加器循环右移	0	0	0	0	1	1	1	1	4
RAL	累加器通过进位位循环左移	0	0	0	1	0	1	1	1	4
RAR	累加器通过进位位循环右移	0	0	0	1	1	1	1	1	4
JMP addr	无条件转移	1	1	0	0	0	0	1	1	10

续表1

记忆符	说明	指令代码[1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_6	时钟周期[2]
JC addr	有进位转移	1	1	0	1	1	0	1	0	10
JNC addr	无进位转移	1	1	0	1	0	0	1	0	10
JZ addr	“0”转移	1	1	0	0	1	0	1	0	10
JNZ addr	非“0”转移	1	1	0	0	0	0	1	0	10
JP addr	正数转移	1	1	1	1	0	0	1	0	10
JM addr	负数转移	1	1	1	1	1	0	1	0	10
JPE addr	按偶校验转移	1	1	1	0	1	0	1	0	10
JPO addr	按奇校验转移	1	1	1	0	0	0	1	0	10
CALL addr	无条件调用(转子)	1	1	0	0	1	1	0	1	17
CC addr	有进位调用	1	1	0	1	1	1	0	0	11/17
CNC addr	无进位调用	1	1	0	1	0	1	0	0	11/17
CZ addr	“0”调用	1	1	0	0	1	1	0	0	11/17
CNZ addr	非“0”调用	1	1	0	0	0	1	0	0	11/17
CP addr	正数调用	1	1	1	1	0	1	0	0	11/17
CM addr	负数调用	1	1	1	1	1	1	0	0	11/17
CPE addr	按偶校验调用	1	1	1	0	1	1	0	0	11/17
CPO addr	按奇校验调用	1	1	1	0	0	1	0	0	11/17
RET	返回	1	1	0	0	1	0	0	1	10
RC	有进位返回	1	1	0	1	1	0	0	0	5/11
RNC	无进位返回	1	1	0	1	0	0	0	0	5/11
RZ	“0”返回	1	1	0	0	1	0	0	0	5/11
RNZ	非“0”返回	1	1	0	0	0	0	0	0	5/11
RP	正数返回	1	1	1	1	0	0	0	0	5/11
RM	负数返回	1	1	1	1	1	0	0	0	5/11
RPE	按偶校验返回	1	1	1	0	1	0	0	0	5/11
RPO	按奇校验返回	1	1	1	0	0	0	0	0	5/11
RST	重新启动	1	1	A	A	A	1	1	1	11
IN port	输入	1	1	0	1	1	0	1	1	10
OUT port	输出	1	1	0	1	0	0	1	1	10
LXI B,data	立即数送寄存器对B及C	0	0	0	0	0	0	0	1	10
LXI D,data	立即数送寄存器对D及E	0	0	0	1	0	0	0	1	10
LXI H,data	立即数送寄存器对H及L	0	0	1	0	0	0	0	1	10
LXI SP	立即数送堆栈指示器	0	0	1	1	0	0	0	1	10
PUSH B	寄存器对B及C进栈	1	1	0	0	0	1	0	1	11
PUSH D	寄存器对D及E进栈	1	1	0	1	0	1	0	1	11
PUSH H	寄存器对H及L进栈	1	1	1	0	0	1	0	1	11
PUSH PSW	A和标志进栈	1	1	1	1	0	1	0	1	11
POP B	寄存器对B及C出栈	1	1	0	0	0	0	0	1	10
POP D	寄存器对D及E出栈	1	1	0	1	0	0	0	1	10

续表 2

记忆符	说明	指令代码〔1〕 D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	时钟周期〔2〕
POP H	寄存器对H及L出栈	1	1	1	0	0	0	0	1	10
POP PSW	A和标志出栈	1	1	1	1	0	0	0	1	10
STA addr	直接存累加器	0	0	1	1	0	0	1	0	13
LDA addr	直接送累加器	0	0	1	1	1	0	1	0	13
XCHG	寄存器H和L、寄存器D和E交换	1	1	1	0	1	0	1	1	4
XTHL	栈顶与寄存器H和L交换	1	1	1	0	0	0	1	1	18
SPHL	寄存器H及L送堆栈指示器	1	1	1	1	1	0	0	1	5
PCHL	寄存器H及L送程序计数器	1	1	1	0	1	0	0	1	5
DAD B	B及C加至H及L	0	0	0	0	1	0	0	1	10
DAD D	D及E加至H及L	0	0	0	1	1	0	0	1	10
DAD H	H及L加至H及L	0	0	1	0	1	0	0	1	10
DAD SP	栈指示器加至H及L	0	0	1	1	1	0	0	1	10
STAX B	间接存累加器	0	0	0	0	0	0	1	0	7
STAX D	间接存累加器	0	0	0	1	0	0	1	0	7
LDAX B	间接送累加器	0	0	0	0	1	0	1	0	7
LDAX D	间接送累加器	0	0	0	1	1	0	1	0	7
INX B	B及C寄存器加1	0	0	0	0	0	0	1	1	5
INX D	D及E寄存器加1	0	0	0	1	0	0	1	1	5
INX H	H及L寄存器加1	0	0	1	0	0	0	1	1	5
INX SP	堆栈指示器加1	0	0	1	1	0	0	1	1	5
DCX B	B及C寄存器减1	0	0	0	0	1	0	1	1	5
DCX D	D及E寄存器减1	0	0	0	1	1	0	1	1	5
DCX H	H及L寄存器减1	0	0	1	0	1	0	1	1	5
DCX SP	堆栈指示器减1	0	0	1	1	1	0	1	1	5
CMA	累加器取反	0	0	1	0	1	1	1	1	4
STC	置进位	0	0	1	1	0	1	1	1	4
CMC	进位取反	0	0	1	1	1	1	1	1	4
DAA	累加器十进制调整	0	0	1	0	0	1	1	1	4
SHLD addr	直接存H及L	0	0	1	0	0	0	1	0	16
LHLD addr	直接送H及L	0	0	1	0	1	0	1	0	16
EI	允许中断	1	1	1	1	1	0	1	1	4
DI	禁止中断	1	1	1	1	0	0	1	1	4
NOP	空操作	0	0	0	0	0	0	0	0	4

注：1.目标地址DDD和源地址SSS的意义为：

代码	相应的寄存器	代码	相应的寄存器	代码	相应的寄存器
000	B	011	E	110	存储器
001	C	100	H	111	累加器
010	D	101	L		

2.两种周期时间(如5/11)表示指令的周期取决于条件标志。

九、位寻址

位寻址指令能对任一内部寄存器的任一位，或任一存贮单元的任一位进行操作（如置1、置0、测试等）。

十、隐含寻址

在指令的操作码中，有一个操作数的地址没有明显指出。但在这类指令中，隐含了以某一寄存器（通常是指累加器A）中的内容作为一个操作数。Z-80中的算术和逻辑操作指令，就是这种寻址方式的指令。

4-2-2 Z-80指令系统 在8位微处理机中，Z-80的指令系统功能较强，类型较齐全，它包括了Intel 8080A的全部指令。Z-80的指令可分为下列几组：

一、取数和交换指令

这一类指令都要指出数据原来存在什么地方(源地址)，后来送到什么地方(目的地址)。它分为8位取数指令和16位取数指令。

二、数据块传送和搜索指令

在Z-80中为了便于数据块的成组传送，专门设置了数据块传送指令，它们又可分为单个传送和成组传送两类。数据块搜索指令是用以检查某一数据块中是否有规定的关键字，也可分为两种：即单条搜索指令和成组搜索指令。

三、算术和逻辑指令

这类指令分为8位数的算术和逻辑运算指令及16位数运算指令。

四、循环和移位指令

按指令的功能，这类指令分为：

1.逻辑和算术移位指令，其中包括算术左移指令、算术右移指令、逻辑右移指令等。

2.循环移位指令。

五、位操作指令

Z-80的位操作指令，允许对内部寄存器的任一位，对以HL间址和以IX或IY变址寻址的内存的任一单元的任一位进行检测，及令其置0或置1。

六、转移指令

这类指令包括无条件转移指令和条件转移指令。

七、子程序调用和返回指令

调用子程序指令有无条件调用指令和条件调用两种，返回指令也分为两种，无条件返回和条件返回。

八、输入/输出指令

九、CPU控制指令

这类指包括空操作指令、暂停指令、禁止中断指令、置中断方式指令等。

表4-2列出了Z-80的指令系统。

Z-80是在Intel 8080的基础上设计出来的，8080所有的指令，Z-80全有。8080的指令与Z-80指令的对照表如表4-3所示。

表4-2 Z-80 指 令 系 统

八 位 取 数 指 令 组

记忆符	用符号表示的操作	标志 C Z P/V S N H	操作码 76 543 210	字节数	M 周期数	T 周期数	说明
LD r,r′	r←r′	• • • • • •	01 r r′	1	1	4	r,r′,寄存器
LD r,n	r←n	• • • • • •	00 r 110	2	2	7	000 B
			← n →				001 C
LD r,(HL)	r←(HL)	• • • • • •	01 r 110	1	2	7	010 D
LD r,(IX+d)	r←(IX+d)	• • • • • •	11 011 101	3	5	19	011 E
			01 r 110				100 H
			← d →				101 L
LD r,(IY+d)	r←(IY+d)	• • • • • •	11 111 101	3	5	19	111 A
			01 r 110				
			← d →				
LD (HL),r	(HL)←r	• • • • • •	01 110 r	1	2	7	
LD (IX+d),r	(IX+d)←	• • • • • •	11 011 101	3	5	19	
			01 110 r				
			← d →				
LD (IY+d),r	(IY+d)←r	• • • • • •	11 111 101	3	5	19	
			01 110 r				
			← d →				
LD (HL),n	(HL)←n	• • • • • •	00 110 110	2	3	10	
			← n →				
LD (IX+d),n	(IX+d)←n	• • • • • •	11 011 101	4	5	19	
			00 110 110				
			← d →				
			← n →				
LD (IY+d),n	(IY+d)←n	• • • • • •	11 111 101	4	5	19	
			00 110 110				
			← d →				
			← n →				
LD A,(BC)	A←(BC)	• • • • • •	00 001 010	1	2	7	
LD A,(DE)	A←(DE)	• • • • • •	00 011 010	1	2	7	
LD A,(nn)	A←(nn)	• • • • • •	00 111 010	3	4	13	
			← n →				
			← n →				
LD (BC),A	(BC)←A	• • • • • •	00 000 010	1	2	7	
LD (DE),A	(DE)←A	• • • • • •	00 010 010	1	2	7	
LD (nn),A	(nn)←A	• • • • • •	00 110 010	3	4	13	
			← n →				
			← n →				
LD A,I	A←I	• ↕ IFF ↕ 0 0	11 101 101	2	2	9	
			01 010 111				
LD A,R	A←R	• ↕ IFF ↕ 0 0	11 101 101	2	2	9	
			01 011 111				
LD I,A	I←A	• • • • • •	11 101 101	2	2	9	
			10 000 111				
LD R,A	R←A	• • • • • •	11 101 101	2	2	9	
			01 001 111				

注：r,r′，表示A、B、C、D、E、H、L寄存器中的任何一个；
IFF表示中断允许触发器的内容复制到P/V标志内；
标志的表示法：•不改变标志，0标志复位，1标志置位，X标志状态不明，↕根据操作结果改变标志。

十 六 位 取 数 指 令

记忆符	用符号表示的操作	标志 C Z P/V S N H	76 543 210	字节数	M 周期数	T 周期数	说明
LD dd,nn	dd←nn	• • • • • •	00 dd0 001 ← n → ← n →	3	3	10	dd 寄存器对 00 BC 01 DE
LD IX,nn	IX←nn	• • • • • •	11 011 101 00 100 001 ← n → ← n →	4	4	14	10 HL 11 SP
LD IY,nn	IY←nn	• • • • • •	11 111 101 00 100 001 ← n → ← n →	4	4	14	
LD HL,(nn)	H←(nn+1) L←(nn)	• • • • • •	00 101 010 ← n → ← n →	3	5	16	
LD dd,(nn)	dd_H←(nn+1) dd_L←(nn)	• • • • • •	11 101 101 01 dd1 011 ← n → ← n →	4	6	20	
LD IX,(nn)	IX_H←(nn+1) IX_L←(nn)	• • • • • •	11 011 101 00 101 010 ← n → ← n →	4	6	20	
LD IY,(nn)	IY_H←(nn+1) IY_L←(nn)	• • • • • •	11 111 101 00 101 010 ← n → ← n →	4	6	20	
LD (nn),HL	(nn+1)←H (nn)←L	• • • • • •	00 100 010 ← n → ← n →	3	5	16	
LD (nn),dd	(nn+1)←dd_H (nn)←dd_L	• • • • • •	11 101 101 01 dd0 011 ← n → ← n →	4	6	20	
LD (nn),IX	(nn+1)←IX_H (nn)←LX_L	• • • • • •	11 011 101 00 100 010 ← n → ← n →	4	6	20	
LD (nn),IY	(nn+1)←IY_H (nn)←IY_L	• • • • • •	11 111 101 00 100 010 ← n → ← n →	4	6	20	
LD SP,HL	SP←HL	• • • • • •	11 111 001	1	1	6	
LD SP,IX	SP←IX	• • • • • •	11 011 101 11 111 001	2	2	10	
LD SP,IY	SP←IY	• • • • • •	11 111 101 11 111 001	2	2	10	qq 寄存器对
PUSH qq	(SP−2)←qq_L (SP−1)←qq_H	• • • • • •	11 qq0 101	1	3	11	00 BC 01 DE
PUSH IX	(SP−2)←IX_L (SP−1)←IX_H	• • • • • •	11 011 101 11 100 101	2	4	15	10 HL 11 AF
PUSH IY	(SP−2)←IY_L (SP−1)←IY_H	• • • • • •	11 111 101 11 100 101	2	4	15	
POP qq	qq_H←(SP+1) qq_L←(SP)	• • • • • •	11 qq0 001	1	3	10	
POP IX	IX_H←(SP+1) IX_L←(SP)	• • • • • •	11 011 101 11 100 001	2	4	14	
POP IY	IY_H←(SP+1) IY_L←(SP)	• • • • • •	11 110 101 11 100 001	2	4	14	

注：dd是BC、DE、HL和SP寄存器对中的任一对

qq是AF、BC、DE和HL寄存器对中的任一对

$(PAIR)_H$、$(PAIR)_L$分别表示寄存器对的高八位和低八位，

如BC_H=C，AF_H=A

交换指令和数据块转移和搜索指令

记忆符	用符号表示的操作	标志 C Z P/V S N H	操作码 76 543 210	字节数	M 周期数	T 周期数	说明
EX DE,HL	DE↔HL	• • • • • •	11 101 011	1	1	4	
EX AF,AF′	AF↔AF′	• • • • • •	00 001 000	1	1	4	
EXX	$\begin{pmatrix}BC\\DE\\HL\end{pmatrix}\leftrightarrow\begin{pmatrix}BC'\\DE'\\HL'\end{pmatrix}$	• • • • • •	11 011 001	1	1	4	寄存器组和辅助寄存器组交换
EX (SP),HL	H↔(SP+1) L↔(SP)	• • • • • •	11 100 011	1	5	19	
EX (SP),IX	$IX_H\leftrightarrow(SP+1)$ $IX_L\leftrightarrow(SP)$	• • • • • •	11 011 101 11 100 011	2	6	23	
EX (SP),IY	$IY_H\leftrightarrow(SP+1)$ $IY_L\leftrightarrow(SP)$	• • • • • •	11 111 101 11 100 011	2	6	23	
LDI	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1	• • ①↕ • 0 0	11 101 101 10 100 000	2	4	16	(HL)送(DE)，指针加1，字节计数器(BC)减1
LDIR	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1 重复直到 BC=0	• • 0 • 0 0	11 101 101 10 110 000	2 2	5 4	21 16	如BC≠0 如BC=0
LDD	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1	• • ①↕ • 0 0	11 101 101 10 101 000	2	4	16	
LDDR	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1 重复直到 BC=0	• • 0 • 0 0	11 101 101 10 111 000	2 2	5 4	21 16	如BC≠0 如BC=0
CPI	A−(HL) HL←HL+1 BC←BC−1	• ②↕ ①↕ ↕ 1 ↕	11 101 101 10 100 001	2	4	6	
CPIR	A−(HL) HL←HL+1 BC←BC−1 重复直到 A=(HL)或 BC=0	• ②↕ ①↕ ↕ 1 ↕	11 101 101 10 110 001	2 2	5 4	21 16	如BC≠0和A≠(HL) 如BC=0或A=(HL)
CPD	A−(HL) HL←HL−1 BC←BC−1	• ②↕ ①↕ ↕ 1 ↕	11 101 101 10 101 001	2	4	16	
CPDR	A−(HL) HL←HL−1 BC←BC−1 重复直到 A=(HL)或 BC=0	• ②↕ ①↕ ↕ 1 ↕	11 101 101 10 111 001	2 2	5 4	21 16	如BC≠0和A≠(HL) 如BC=0或A=(HL)

注：①如BC−1的结果为0，P/V为0，否则P/V为1；

②如A=(HL)，Z标志为1，否则Z=0。

八 位 算 逻 指 令

记忆符	用符号表示的操作	标志 C Z P/V S N H	操作码 76 543 210	字节数	M 周期数	T 周期数	说明
ADD A,r	A←A+r	↕ ↕ V ↕ 0 ↕	10 [000] r	1	1	4	r 寄存器 000 B 001 C
ADD A,n	A←A+n	↕ ↕ V ↕ 0 ↕	11 [000] 110 ← n →	2	2	7	010 D
ADD A,(HL)	A←A+(HL)	↕ ↕ V ↕ 0 ↕	10 [000] 110	1	2	7	011 E
ADD A,(IX+d)	A←A+(IX+d)	↕ ↕ V ↕ 0 ↕	11 011 101 10 [000] 110 ← d →	3	5	19	100 H 101 L 111 A
ADD A,(IY+d)	A←A+(IY+d)	↕ ↕ V ↕ 0 ↕	11 111 101 10 [000] 110 ← d →	3	5	19	
ADC A,s	A←A+S+CY	↕ ↕ V ↕ 0 ↕	[001]				S为r、n、(HL)、(IX+d)、(IY+d)之一，同ADD指令。所指的位代替ADD指令中的000
SUB s	A←A−S	↕ ↕ V ↕ 1 ↕	[010]				
SBC A,s	A←A−S−CY	↕ ↕ V ↕ 1 ↕	[011]				
AND s	A←A∧S	0 ↕ P ↕ 0 1	[100]				
OR s	A←A∨S	0 ↕ P ↕ 0 0	[110]				
XOR s	A←A⊕S	0 ↕ P ↕ 0 0	[101]				
CP s	A−S	↕ ↕ V ↕ 1 ↕	[111]				
INC r	r←r+1	• ↕ V ↕ 0 ↕	00 r [100]	1	1	4	
INC (HL)	(HL)←(HL)+1	• ↕ V ↕ 0 ↕	00 110 [100]	1	3	11	
INC (IX+d)	(IX+d)←(IX+d)+1	• ↕ V ↕ 0 ↕	11 011 101 00 110 [100] ← d →	3	6	23	
INC (IY+d)	(IY+d)←(IY+d)+1	• ↕ V ↕ 0 ↕	11 111 101 00 110 [100] ← d →	3	6	23	
DEC m	m←m−1	• ↕ V ↕ 1 ↕	[101]				m是r、(HL)、(IX+d)、(IY+d)之一，同INC。格式和状态同INC，操作码中的101代替100

注：P/V标志中，V表示运算结果是否溢出，V=1表示溢出，V=0表示不溢出。

P/V标志中，P表示运算后各位的和是奇数还是偶数，P=1表示结果为偶数，P=0表示结果为奇数。

CY为进位标志中的数。

通用算术和CPU控制指令

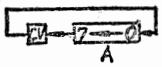
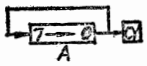
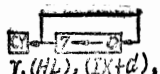
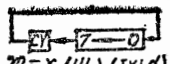
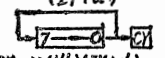
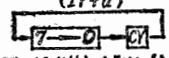
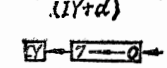
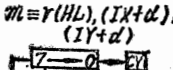
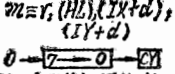
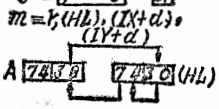
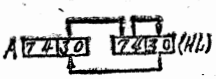
记忆符	用符号表示的操作	标志 C Z P/V S N H	操作码 76 543 210	字节数	M 周期数	T 周期数	说明
DAA	把累加器内容调整为二—十进制数	↕ ↕ P ↕ • ↕	00 100 111	1	1	4	累加器十进制调整
CPL	$A \leftarrow \overline{A}$	• • • • 1 1	00 101 111	1	1	4	累加器求反（对1补码）
NEG	A←0−A	↕ ↕ V ↕ 1 ↕	11 101 101 01 000 100	2	2	8	累加器求补（对2补码）
CFF	$CY \leftarrow \overline{CY}$	↕ • • • 0 X	00 111 111	1	1	4	进位标志求反
SCF	CY←1	1 • • • 0 0	00 110 111	1	1	4	进位标志置位
NOP	空操作	• • • • • •	00 000 000	1	1	4	
HALT	CPU暂停	• • • • • •	01 110 110	1	1	4	
DI	IFF←0	• • • • • •	11 110 011	1	1	4	禁止中断
EI	IFF←1	• • • • • •	11 111 011	1	1	4	允许中断
IM 0	置中断方式0	• • • • • •	11 101 101 01 000 110	2	2	8	
IM 1	置中断方式1	• • • • • •	11 101 101 01 010 110	2	2	8	
IM 2	置中断方式2	• • • • • •	11 101 101 01 011 110	2	2	8	

十六位算术指令

记忆符	用符号表示的操作	标志 C Z P/V S N H	操作码 76 543 210	字节数	M 周期数	T 周期数	说明
ADD HL, ss	HL←HL+ss	↕ • • • 0 X	00 ss1 001	1	3	11	ss 寄存器 00 BC 01 DE 10 HL 11 SP
ADC HL, ss	HL←HL+ss+CY	↕ ↕ V ↕ 0 X	11 101 101 01 ss1 010	2	4	15	
SBC HL, ss	HL←HL−ss−CY	↕ ↕ V ↕ 0 X	11 101 101 01 ss0 010	2	4	15	
ADD IX, pp	IX←IX+pp	↕ • • • 0 X	11 011 101 00 pp1 001	2	4	15	pp 寄存器 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY←IY+rr	↕ • • • 0 X	11 111 101 00 rr1 001	2	4	15	rr 寄存器 00 BC 01 DE 10 IY 11 SP
INC ss	ss←ss+1	• • • • • •	00 ss0 011	1	1	6	
INC IX	IX←IX+1	• • • • • •	11 011 101 00 100 011	2	2	10	
INC IY	IY←IY+1	• • • • • •	11 111 101 00 100 011	2	2	10	
DEC ss	ss←ss−1	• • • • • •	00 ss1 011	1	1	6	
DEC IX	IX←IX−1	• • • • • •	11 011 101 00 101 011	2	2	10	
DEC IY	IY←IY−1	• • • • • •	11 111 101 00 101 011	2	2	10	

注：ss是寄存器对BC、DE、HL和SP之一。

循 环 和 移 位 指 令

记忆符	符号表示的操作	标志 CZP/VSNH	操作码 76 543 210	字节数	M 周期数	T 状态数	说明
RLCA		↕ • • • 0 0	00 000 111	1	1	4	累加器向左循环移位
RLA		↕ • • • 0 0	00 010 111	1	1	4	累加器向左循环移位，带进位
RRCA		↕ • • • 0 0	00 001 111	1	1	4	累加器向右循环移位
RRA		↕ • • • 0 0	00 011 111	1	1	4	累加器向右循环移位，带进位
RLC r		↕ ↕ P ↕ 0 0	11 001 011 00 [000] r	2	2	8	寄存器 r 向左循环移位
RLC (HL)		↕ ↕ P ↕ 0 0	11 001 011 00 [000] 110	2	4	15	r寄存器 000 B
RLC(IX+d)		↕ ↕ P ↕ 0 0	11 011 101 11 001 011 ← d → 00 [000] 110	4	6	23	001 C 010 D 010 E 100 H 101 L 111 A
RLC(IY+d)		↕ ↕ P ↕ 0 0	11 111 101 11 001 011 ← d → 00 [000] 110	4	6	23	
RL m		↕ ↕ P ↕ 0 0	[010]				指令格式和状态同RLC m，为形成新的操作码，把新的码代替RLC m中的 [000]
RRC m		↕ ↕ P ↕ 0 0	[001]				
RR m		↕ ↕ P ↕ 0 0	[011]				
SLA m		↕ ↕ P ↕ 0 0	[100]				
SRA m		↕ ↕ P ↕ 0 0	[101]				
SRL m		↕ ↕ P ↕ 0 0	[111]				
RLD		• ↕ P ↕ 0 0	11 101 101 01 101 111	2	5	18	BCD 位在累加器和存储单元 (HL) 间向左或右循环，累加器高四位内容不受影响
RRD		• ↕ P ↕ 0 0	11 101 101 01 100 111	2	5	18	

位操作（置位、复位和测试）指令

记忆符	用符号表示的操作	标志 C Z P/V S N H	操作码 76 543 210	字节数	M 周期数	T 周期数	说明
BIT b, r	$Z \leftarrow \overline{r_b}$	• ↕ X X 0 1	11 001 011 01 b r	2	2	8	r 寄存器 000 B 001 C 010 D 011 E 100 H 101 L 111 A
BIT b (HL)	$Z \leftarrow \overline{(HL)_b}$	• ↕ X X 0 1	11 001 011 01 b 110	2	3	12	
BIT b, (IX+d)	$Z \leftarrow \overline{(IX+d)_b}$	• ↕ X X 0 1	11 011 011 11 001 011 ⟵ d ⟶ 01 b 110	4	5	20	
BIT b, (IY+d)	$Z \leftarrow \overline{(IY+d)_b}$	• ↕ X X 0 1	11 111 101 11 001 011 ⟵ d ⟶ 01 b 110	4	5	20	b 被测位 000 0 001 1 010 2 011 3 100 4 101 5 110 6 111 7
SET b,r	$r_b \leftarrow 1$	• • • • • •	11 001 011 [11] b r	2	2	8	
SET b,(HL)	$(HL)_b \leftarrow 1$	• • • • • •	11 001 011 [11] b 110	2	4	15	
SET b,(IX+d)	$(IX+d)_b \leftarrow 1$	• • • • • •	11 011 101 11 001 011 ⟵ d ⟶ [11] b 110	4	6	23	
SET b,(IY+d)	$(IY+d)_b \leftarrow 1$	• • • • • •	11 111 101 11 001 011 ⟵ d ⟶ [11] b 110	4	6	23	
RES b,m	$S_b \leftarrow 0$ m≡r,(HL), (IX+d), (IY+d)						为形成新的操作码，用[10]代替置位指令 SET b,m 中的[11]

注：符号S_b表示m指定的单元内容的第b位。

转 移 指 令

记忆符	用符号表示的操作	标志 C Z P/V S N H	操作码 11 543 011	字节数	M 周期数	T 周期数	说明
JP nn	PC←nn	• • • • • •	11 000 011 ← n → ← n →	3	3	10	cc 条件 000 NZ非零 001 Z零 010 NC无进位
JP CC, nn	若条件CC是真，PC←nn 否则继续	• • • • • •	11 cc 010 ← n → ← n →	3	3	10	011 C进位 100 PO奇偶(奇) 101 PE奇偶(偶) 110 P正号
JR e	PC←PC+e	• • • • • •	00 011 000 ← e-2 →	2	3	12	111 M负号
JR C, e	若C=0，则继续	• • • • • •	00 111 000 ← e-2 →	2	2	7	如条件不满足
	若C=1，PC←PC+e			2	3	12	如条件满足
JR NC, e	若C=1，则继续	• • • • • •	00 110 000 ← e-2 →	2	2	7	如条件不满足
	若C=0，PC←PC+e			2	3	12	如条件满足
JR Z, e	若Z=0，则继续	• • • • • •	00 101 000 ← e-2 →	2	2	7	如条件不满足
	若Z=1，PC←PC+e			2	3	12	如条件满足
JR NZ, e	若Z=1，则继续	• • • • • •	00 100 000 ← e-2 →	2	2	7	如条件不满足
	若Z=0，PC←PC+e			2	3	12	如条件满足
JP (HL)	PC←HL	• • • • • •	11 101 001	1	1	4	
JP (IX)	PC←IX	• • • • • •	11 011 101 11 101 001	2	2	8	
JP (IY)	PC←IY	• • • • • •	11 111 101 11 101 001	2	2	8	
DJNZ, e	B←B-1 若B=0，则继续	• • • • • •	00 010 000 ← e-2 →	2	2	8	若B=0
	若B≠0，PC←PC+e			2	3	13	若B≠0

注：e代表相对寻址方式中的位移量，e为用带符号的对2补码表示的数，其数值在〔-126，+129〕以内。操作码中的e-2提供有效地址PC+e，因为在加e之前PC的值已加2。

调 用 和 返 回 指 令

记忆符	用符号表示的操作	标志 C Z P/V S N H	操作码 76 543 210	字节数	M 周期数	T 周期数	说明
CALL nn	(SP-1)←PC_H (SP-2)←PC_L PC←nn	• • • • • •	11 001 101 ← n → ← n →	3	5	17	
CALL CC, nn	若条件CC是假，则继续，否则，同CALL nn	• • • • • •	11 CC 100 ← n → ← n →	3 3	3 5	10 17	若CC是假 若CC是真
RET	PC_L←(SP) PC_H←(SP+1)	• • • • • •	11 001 001	1	3	10	
RET CC	若条件CC是假，则继续；否则同RET	• • • • • •	11 CC 000	1 1	1 3	5 11	若CC是假 若CC是真
RETI	从中断返回	• • • • • •	11 101 101 01 001 101	2	4	14	CC条件 000 NZ非零 001 Z零 010 NC无进位 011 C进位 100 PO奇(偶奇) 101 PE奇(偶偶) 110 P正号 111 M负号
RETN	从不可屏蔽中断返回	• • • • • •	11 101 101 01 000 101	2	4	14	
RST_P	(SP-1)←PC_H (SP-2)←PC_L PC_H←0 PC_L←P	• • • • • •	11 t 111	1	8	11	t P 000 00H 001 08H 010 10H 011 18H 100 20H 101 28H 110 30H 111 38H

输 入 和 输 出 指 令

记忆符	用符号表示的操作	标志 C Z P/V S N H	操作码 76 543 210	字节数	M 周期数	T 周期数	说明
IN A,(n)	A←(n)	• • • • • •	11 011 011 ← n →	2	3	11	n至A_0～A_7
IN r,(C)	r←(C) 若r=110，仅标志受影响	• ↕ P ↕ 0 t	11 101 101 01 r 000	2	3	12	C至A_0～A_7
INI	(HL)←(C) B←B-1 HL←HL+1	• ①↕ X X 1 X	11 101 101 10 100 010	2	4	16	C至A_0～A_7
INIR	(HL)←(C) B←B-1 HL←HL+1 一直重复到B=0	• 1 X X 1 X	11 101 101 10 110 010	2 2	5（若B≠0） 4（若B=0）	21 16	C至A_0～A_7
IND	(HL)←(C) B←B-1 HL←HL-1	• ①↕ X X 1 X	11 101 101 10 101 010	2	4	16	C至A_0～A_7

续表

记忆符	用符号表示的操作	标志 C Z P/V S N H	操作码 76 543 210	字节数	M 周期数	T 周期数	说明
INDR	(HL)←(C) B←B−1 HL←HL−1 一直重复到 B=0	• 1 X X 1 X	11 101 101 10 111 010	2 2	5 (若B≠0) 4 (若B=0)	21 16	C至A_0～A_7
OUT(n),A	(n)←A	• • • • • •	11 010 011 ←— n —→	2	3	11	A_0～A_7至n
OUT(C),r	(C)←r	• • • • • •	11 101 101 01 r 001	2	3	12	A_0～A_7至C
OUTI	(C)←(HL) B←B−1 HL←HL+1	•①↕ X X 1 X	11 101 101 10 100 011	2	4	16	A_0～A_7至C
OTIR	(C)←(HL) B←B−1 HL←HL+1 一直重复到 B=0	• 1 X X 1 X	11 101 101 10 110 011	2 2	5 (若B≠0) 4 (若B=0)	21 16	A_0～A_7至C
OUTD	(C)←(HL) B←B−1 HL←HL−1	•①↕ X X 1 X	11 101 101 10 101 011	2	4	16	A_0～A_7至C
OTDR	(C)←(HL) B←B−1 HL←HL−1 一直重复到 B=0	• 1 X X 1 X	11 101 101 10 111 011	2 2	5 (若B≠0) 4 (若B=0)	21 16	A_0～A_7至C

注：①若B−1的结果为0，Z标志置1，否则置0。

表4-3　　intel8080和Z-80指令转换

INTEL 8080	Z-80	字节数	INTEL 8080	Z-80	字节数
ACI data	ADC A,n	2	CM addr	CALL M,nn	3
ADC r	ADC A,r	1	CMA	CPL	1
ADC M	ADC A,(HL)	1	CMC	CCF	1
ADD r	ADD A,r	1	CMP r	CP r	1
ADD M	ADD A,(HL)	1	CMP M	CP (HL)	1
ADI data	ADD A,n	2	CNC addr	CALL NC,nn	3
ANA r	AND r	1	CNZ addr	CALL NZ,nn	3
ANA M	AND (HL)	1	CP addr	CALL P,nn	3
ANI data	AND n	2	CPE addr	CALL PE,nn	3
CALL addr	CALL nn	3	CPI data	CP n	2
CC addr	CALL C,nn	3	CPO addr	CALL PO,nn	3

续表

INTEL 8080	Z-80	字节数	INTEL 8080	Z-80	字节数
CZ addr	CALL Z,nn	3	ORA r	OR r	1
DAA	DAA	1	ORA M	OR (HL)	1
DAD B	ADD HL,BC	1	ORI data	OR n	2
DAD D	ADD HL,DE	1	OUT port	OUT (n),A	2
DAD H	ADD HL,HL	1	PCHL	JP (HL)	1
DAD SP	ADD HL,SP	1	POP B	POP BC	1
DCR r	DEC r	1	POP D	POP DE	1
DCR M	DEC (HL)	1	POP H	POP HL	1
DCX B	DEC BC	1	POP PSW	POP AF	1
DCX D	DEC DE	1	PUSH B	PUSH BC	1
DCX H	DEC HL	1	PUSH D	PUSH DE	1
DCX SP	DEC SP	1	PUSH H	PUSH HL	1
DI	DI	1	PUSH PSW	PUSH AF	1
EI	EI	1	RAL	RLA	1
HLT	HALT	1	RAR	RRA	1
IN Port	IN A,(n)	2	RC	RET C	1
INR r	INC r	1	RET	RET	1
INR M	INC (HL)	1	RLC	RLCA	1
INX B	INC BC	1	RM	RET M	1
INX D	INC DE	1	RNC	RET NC	1
INX H	INC HL	1	RNZ	RET NZ	1
INX SP	INC SP	1	RP	RET P	1
JC addr	JP C,nn	3	RPE	RET PE	1
JM addr	JP M,nn	3	RPO	RET PO	1
JMP addr	JP nn	3	RRC	RRCA	1
JNC addr	JP NC,nn	3	RST	RST P	1
JNZ addr	JP NZ,nn	3	RZ	RET Z	1
JP addr	JP P,nn	3	SBB r	SBC A,r	1
JPE acdr	JP PE,nn	3	SBB M	SBC A,(HL)	1
JPO addr	JP PO,nn	3	SBI data	SBC A,n	2
JZ addr	JP Z,nn	3	SHLD addr	LD (nn),HL	3
LDA addr	LD A,(nn)	3	SPHL	LD SP,HL	1
LDAX B	LD A,(BC)	1	STA addr	LD (nn),A	3
LDAX D	LD A,(DE)	1	STAX B	LD (BC),A	1
LHLD addr	LD HL,(nn)	3	STAX D	LD (DE),A	1
LXI B,data	LD BC,nn	3	STC	SCF	1
LXI D,data	LD DE,nn	3	SUB r	SUB r	1
LXI H,data	LD HL,nn	3	SUB M	SUB(HL)	1
LXI SP,data	LD SP,nn	3	SUI data	SUB n	2
MOV r_1,r_2	LD r,r′	1	XCHG	EX DE,HL	1
MOV M,r	LD (HL),r	1	XRA r	XOR r	1
MOV r,M	LD r,(HL)	1	XRA M	XOR (HL)	1
MVI r,data	LD r,n	2	XRI data	XOR n	2
MVI M,data	LD (HL),n	2	XTHL	EX (SP),HL	1
NOP	NOP	1			

4-3 6502指令系统

6502微处理器具有56种指令。一条指令包括一个字节至三个字节，不论单字节指令或多字节指令，其第一个字节一律为操作码，多字节指令中第二或第三字节必定是操作数或操作数地址。单字节指令的操作数是隐含的，即约定操作数是某个寄存器中的内容，而这种约定是隐含在操作码的代码之中的。

4-3-1 6502寻址方式 6502有13种寻址方式，它们是：

一、隐含寻址

采用这种寻址方式的指令只用一个字节，操作数为指令所隐含。

二、累加器寻址

采用累加器寻址的指令也是单字节指令，操作数存在累加器A中。从广义上说，累加器寻址也是一种隐含寻址，是一种特殊的隐含寻址。

三、立即寻址

立即寻址的指令都是两字节指令，第一个字节为操作码，第二个字节为立即数。

四、绝对寻址

采用绝对寻址的指令皆为三字节指令。指令的第二第三字节指出操作数在存贮器中的有效地址。

五、绝对X变址

采用这种寻址方式的指令都是三字节指令。指令第二字节和第三字节所指示的地址加上变址寄存器X的内容，即是操作数的有效地址。

六、绝对Y变址

使用绝对Y变址的指令亦为三字节指令。它和绝对X变址方式的区别仅在于使用的是变址寄存器Y而不是X。即是说，操作数的地址是由指令的第二字节和第三字节所指示的地址加上变址寄存器Y的内容得到。

七、零页寻址

6502的寻址范围是64K，分为256页，其中第0页的地址为$0000～$00FF，用8位二进制数即可对零页进行寻址。因此采用零页寻址的指令皆为两字节指令，指令的第二字节是操作数在零页中的地址。

八、零页X变址

采用零页X变址方式的指令均为两字节指令，指令的第二字节指示的基本零页地址加上变址寄存器X的内容即为操作数的有效地址。

九、零页Y变址

零页Y变址方式的指令为两字节指令，指令的第二字节指示的基本零页地址加上变址寄存器Y的内容即为操作数的有效地址。

十、间接寻址

间接寻址方式在6502中仅用于无条件转移JMP指令。这是个三字节指令，指令的第二字节存放一个地址的低8位，第三字节存放这个地址的高8位，这个地址中的内容是要寻找的真正地址的低8位，而它的下一个地址中存放着真正地址的高8位。

十一、相对寻址

采用相对寻址方式的指令是两字节指令，在6502中只用于条件转移指令。第二字节为条

件转移指令的跳转步长，又称为偏移量D。本指令第一字节所在的地址和偏移量D相加，即得到有效地址。

十二、先变址(X)间接寻址

采用先变址(X)间接寻址方式(IND,X)的指令皆为两字节指令。这种寻址方式是指先以X作为变址寄存器和零页基地址IND相加，结果还是一个间接地址，还需再经过两次间接寻址，才能得到有效地址。

十三、后变址(Y)间接寻址

采用后变址(Y)间接寻址方式((IND),Y)的指令皆为两字节指令。这种寻址方式是指先对IND部分所指出的零页地址作一次间接寻址，得到一个低8位地址，再对IND+1作一次间接寻址，得到一个高8位地址，将这两部分合起来做为16位的基地址，再与变址寄存器Y的值相加即得到操作数的有效地址。

4-3-2 6502指令系统 指令系统是程序设计的基础，因此指令系统必须能满足程序设计的各种要求，它必须包括数据传送、程序控制及算术逻辑运算三类指令。下面就以这三种指令类型介绍6502的指令系统。

一、数据传送类指令

数据传送类指令主要包括将数据由内存贮器取到各种寄存器，由各种寄存器传送到内存单元及各种寄存器之间的数据传送三部分。

1.取数指令

指　令	寻址方式	十六进制代码	机器周期	操作	N Z C I D V
LDA #$nn	立　即	A9 nn	T=2	M→A	V V — — — —
LDA $nn	零　页	A5 nn	T=3		
LDA $nn,x	零页X变址	B5 nn	T=4		
LDA $nHnL	绝　对	AD nL nH	T=4		
LDA $nHnL,X	绝对X变址	BD nL nH	T-4 跨页T-5		
LDA $nHnL,Y	绝对Y变址	B9 nL nH	T-4 跨页T-5		
LDA ($nn,X)	先X变址后间址	A1 nn	T=6		
LDA ($nn),Y	先间址后Y变址	B1 nn	T-5 跨页T-6		
LDX #$nn	立　即	A2	T=2	M→X	V V — — — —
LDX $nn	零　页	A6 nn	T=3		
LDX $nn,Y	零页Y变址	B6 nn	T=4		
LDX $nHnL	绝　对	AE nL nH	T=4		
LDX $nHnL,Y	绝对Y变址	BE nL nH	T-4 跨页T-5		
LDY #$nn	立　即	A0 nn	T=2	M→Y	V V — — — —
LDY $nn	零　页	A4 nn	T=3		
LDY$nn,X	零页X变址	B4 nn	T=4		
LDY $nHnL	绝　对	AC nL nH	T=4		
LDY $nHnL,X	绝对X变址	BC nL nH	T-4 跨页T-5		
PLA	隐　含	68	T=4	A↑(弹出)	V V — — — —
PLP	隐　含	28	T=4	P↑	自堆栈弹出

$：表示其后面跟的是十六进制数。

#：表示其后面跟的是立即数。

nn：表示二位十六进制数，nHnL表示四位十六进制数，nH表示其高8位，nL表示其低位。

6502规定存贮器的第一页为堆栈区。PLA指令的功能是由堆栈取数到累加器。

标志位下面标的"V"号表示指令执行结果对该标志位有影响，"—"号表示对该标志位无影响(下同)。

2.存数指令

指　令	寻址方式	十六进制代码	机器周期	操　作	N Z C I D V
STA $nn	零　页	85 nn	T=3	A→M	— — — — — —
STA $nn,X	零页X变址	95 nn	T=4		
STA $nHnL	绝　对	8D nL nH	T=4		
STA $nHnL,X	绝对X变址	9D nL nH	T=5		
STA $nHnL,Y	绝对Y变址	99 nL nH	T=5		
STA($nn, X)	先X变址后间址	81 nn	T=6		
STA($nn), Y	先间址后Y变址	91 nn	T=6		
STX $nn	零　页	86 nn	T=3	X→M	— — — — — —
STX $nn, Y	零页Y变址	96 nn	T=4		
STX $nHnL	绝　对	8E nL nH	T=4		
STY $nn	零　页	84 nn	T=3	Y→M	— — — — — —
STY $nn,X	零页X变址	94 nn	T=4		
STY $nHnL	绝　对	8C nL nH	T=4		
PHA	隐　含	48	T=3	A↓	— — — — — —
PHP	隐　含	0 8	T=3	P	— — — — — —

3.寄存器取数至寄存器

指　令	寻址方式	十六进制代码	机器周期	操　作	N Z C I D V
TAX	隐　含	AA	T=2	A→X	V V — — — —
TAY	隐　含	A 8	T=2	A→Y	V V — — — —
TSX	隐　含	B A	T=2	S→X	V V — — — —
TXA	隐　含	8 A	T=2	X→A	V V — — — —
TXS	隐　含	9 A	T=2	X→S	— — — — — —
TYA	隐　含	9 8	T=2	Y→A	V V — — — —

二、程序控制类指令

程序控制指令包括程序转移指令、设置条件码指令、清除条件码指令、CPU控制指令四类，程序控制是程序设计中十分有用的操作，灵活使用程序控制指令体现了程序设计技巧。

1.程序转移指令

指　令	寻址方式	十六进制代码	机器周期	操　作	N Z C I D V
BCC $nn	相　对	90 nn	不转 T=2 转 T=3 跨页 T=4	C为0时转移	— — — — — —
BCS $nn	相　对	B0 nn	不转 T=2 转 T=3 跨页 T=4	C为1时转移	— — — — — —
BEQ $nn	相　对	F0 nn	不转 T=2 转 T=3 跨页 T=4	Z=1时转移	— — — — — —
BMI $nn	相　对	30 nn	不转 T=2 转 T=3 跨页 T=4	N=1时转移	— — — — — —
BNE $nn	相　对	D0 nn	不转 T=2 转 T=3 跨页 T=4	Z=0时转移	— — — — — —
BPL $nn	相　对	10 nn	不转 T=2 转 T=3 跨页 T=4	N=0时转移	— — — — — —
BVC $nn	相　对	50 nn	不转 T=2 转 T=3 跨页 T=4	V=0时转移	— — — — — —
BVS $nn	相　对	70 nn	不转 T=2 转 T=3 跨页 T=4	V=1时转移	— — — — — —
JMP $nHnL	绝　对	4C nL nH	T=3	无条件转移	— — — — — —
JMP($nHnL)	间　址	6C nL nH	T=5		
JSR$nHnL	绝　对	20 nL nH	T=6	PC+2↓ (PC+1)→PCL (PC+2)→PCH	— — — — — —
RTI	隐　含	40	T=6	P↑ PC↑	自堆栈弹出
RTS	隐　含	60	T=6	PC↑ PC+1→PC	— — — — — —

2.设置条件码

条件转移指令与状态寄存器某位的值密切相关。状态寄存器的各条件码一般由程序执行中的有关指令及操作情况自动产生。但它们也可以用指令设置或清除。

指　令	寻址方式	十六进制代码	机器周期	操　作	N Z C I D V
SEC	隐　含	38	T=2	1→C	— — 1 — — —
SED	隐　含	F8	T=2	1→D	— — — — 1 —
SEI	隐　含	78	T=2	1→I	— — — 1 — —

3.清除条件码

状态寄存器P中的某些标志也可以通过指令来清除，即不论该标志原来状态如何，通过一条指令可保证使它为0。

指　令	寻址方式	十六进制代码	机器周期	操　作	N Z C I D V
CLC	隐　含	18	T=2	0→C	— — 0 — — —
CLD	隐　含	D8	T=2	0→D	— — — — 0 —
CLI	隐　含	58	T=2	0→I	— — — 0 — —
CLV	隐　含	B8	T=2	0→V	— — — — — 0

4.CPU控制指令

CPU控制指令包括强迫中断指令和空操作指令。

指　令	寻址方式	十六进制代码	机器周期	操　作	N Z C I D V
BRK	隐　含	00	T=7	PC+2↓ P↓	— — — 1 — —
NOP	隐　含	EA	T=2	无	— — — — — —

三、算术逻辑运算类指令

算术逻辑运算类指令包括算术运算指令、逻辑运算指令、移位指令及比较指令四类指令。

1.算术运算指令

指令	寻址方式	十六进制代码	机器周期	操作	N Z C I D V
ADC *$nn	立即	6 9 nn	T=2	A+M+C→A	V V V — — V
ADC $nn	零页	6 5 nn	T=2		
ADC $nn,X	零页X变址	7 5 nn	T=4		
ADC $nHnL	绝对	6 D nL nH	T=4		
ADC $nHnL,X	绝对X变址	7 D nL nH	T=4 跨页T=5		
ADC $nHnL,Y	绝对Y变址	7 9 nL nH	T=4 跨页T=5		
ADC ($nn,X)	先X变址后间址	6 1 nn	T=6		
ADD ($nn),Y	先间址后Y变址	7 1 nn	T=5		
DEC $nn	零页	C 6 nn	T=5	M−1→M	V V — — — —
DEC $nn,X	零页X变址	D 6 nn	T=6		
DEC $nHnL	绝对	CE nL nH	T=6		
DEC $nHnL,X	绝对X变址	DE nL nH	T=7		
DEX	隐含	C A	T=2	X−1→X	V V — — — —
DEY	隐含	8 8	T=2	Y−1→Y	V V — — — —
INC $nn	零页	E6 nn	T=5	M+1→M	V V — — — —
INC $nn,X	零页X变址	F6 nn	T=6		
INC $nHnL	绝对	EE nL nH	T=6		
INC $nHnL,X	绝对X变址	FE nL nH	T=7		
INX	隐含	E 8	T=2	X+1→X	V V — — — —
INY	隐含	C 8	T=2	Y+1→Y	V V — — — —
SBC *$nn	立即	E9 nn	T=2	A—M—C→A	V V V — — V
SBC $nn	零页	E 5 nn	T=3		
SBC $nn,X	零页X变址	F 5 nn	T=4		
SBC $nHnL	绝对	ED nL nH	T=4		
SBC $nHnL,X	绝对X变址	FD nL nH	T=4 跨页T=5		
SBC $nHnL,Y	绝对Y变址	F 9 nL nH	T=4 跨页T=5		
SBC ($nn,X)	先X变址后间址	E 1 nn	T=6		
SBC ($nn),Y	先间址后Y变址	F 1 nn	T=5 跨页T=6		

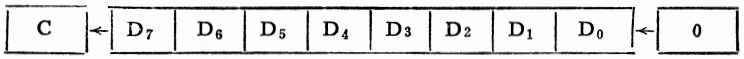
2.逻辑运算指令

指　　令	寻 址 方 式	十六进制代码	机 器 周 期	操　　作	N Z C I D V
AND　*$nn	立　即	2 9　nn	T = 2	A∧M→A	V V — — — —
AND　$nn	零　页	2 5　nn	T = 3		
AND　$nn,X	零页X变址	3 5　nn	T = 4		
AND　$nHnL	绝　对	2D　nL　nH	T = 4 跨页T = 5		
AND　$nHnL,X	绝对X变址	3D　nL　nH	T = 4 跨页T = 5		
AND　$nHnL,Y	绝对Y变址	3 9　nL nH	T = 4 跨页T = 5		
AND($nn,X)	先X变址后间址	2 1　nn	T = 6		
AND($nn),Y	先间址后Y变址	3 1　nn	T = 5		
BIT　$nn	零　页	2 4　nn	T = 3	A∧M M_7→N M_6→V	M_7 V — — — M_6
BIT　$nHnL	绝　对	2C　nL　nH	T = 4		
EOR　*$nn	立　即	4 9　nn	T = 2	A⊕M→A	V V — — — —
EOR　$nn	零　页	4 5　nn	T = 3		
EOR　$nn,X	零页X变址	5 5　nn	T = 4		
EOR　$nHnL	绝　对	4D　nL　nH	T = 4		
EOR　$nHnL,X	绝对X变址	5 D nL nH	T = 4 跨页T = 5		
EOR　$nHnL,Y	绝对Y变址	5 9　nL　nH	T = 4 跨页T = 5		
EOR　($nn,X)	先X变址后间址	4 1　nn	T = 6		
EOR　($nn),Y	先间址后Y变址	5 1　nn	T = 5 跨页T = 6		
ORA　*$nn	立　即	0 9　nn	T = 2	A∨M→A	V V — — — —
ORA　$nn	零　页	0 5　nn	T = 3		
ORA　$nn,X	零页X变址	1 5　nn	T = 4		
ORA　$nHnL	绝　对	0 D nL nH	T = 4		
ORA　$nHnL,X	绝对X变址	1 D　nL　nH	T = 4 跨页T = 5		
ORA　$nHnL,Y	绝对Y变址	1 9　nL　nH	T = 4 跨页T = 5		
ORA　($nn,X)	先X变址后间址	0 1　nn	T = 6		
ORA　($nn),Y	先间址后Y变址	1 1　nn	T = 5		

3.移位指令

①算术左移

操作：



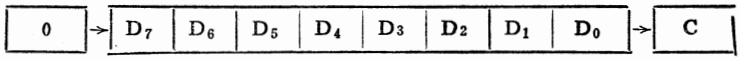
N	Z	C	I	D	V
V	V	V	—	—	—

指令	寻址方式	十六进制代码	机器周期
ASL A	累加器	0A	T=2
ASL $nn	零页	06 nn	T=5
ASL $nn,X	零页X变址	16 nn	T=6
ASL $nHnL	绝对	0E nL nH	T=6
ASL $nHnL,X	绝对X变址	1E nL nH	T=7

②算术右移

N	Z	C	I	D	V
0	V	V	—	—	—

操作：

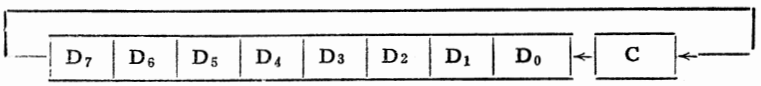


指令	寻址方式	十六进制代码	机器周期
LSR A	累加器	4A	T=2
LSR $nn	零页	46 nn	T=5
LSR $nn,X	零页X变址	56 nn	T=6
LSR $nHnL	绝对	4E nL nH	T=6
LSR $nHnL,X	绝对X变址	5E nL nH	T=7

③循环左移

N	Z	C	I	D	V
V	V	V	—	—	—

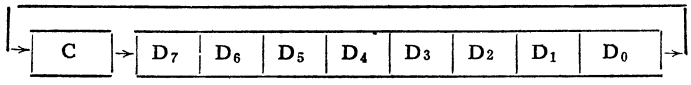
操作：



指令	寻址方式	十六进制代码	机器周期
ROL A	累加器	2A	T=2
ROL $nn	零页	26 nn	T=5
ROL $nn,X	零页X变址	36 nn	T=6
ROL $nHnL	绝对	2E nL nH	T=6
ROL $nHnL,X	绝对X变址	3E nL nH	T=7

④循环右移

操作：



N	Z	C	I	D	V
V	V	V	—	—	—

指　令	寻址方式	十六进制代码	机器周期
ROR A	累加器	6A	T=2
ROR $nn	零页	66 nn	T=5
RQR $nn,X	零页X变址	76 nn	T=6
ROR $nHnL	绝对	6E nL nH	T=6
ROR $nHnL,X	绝对X变址	7E nL nH	T=7

4.比较指令

比较指令实际执行减法运算，只是其结果不回送，执行结果仅对N、Z、C三个标志产生影响。

指　令	寻址方式	十六进制代码	机器周期	操　作	N Z C I D V
CMP #$nn	立即	C9 nn	T=2	A—M	V V V — — —
CMP $nn	零页	C5 nn	T=3		
CMP $nn,X	零页X变址	D5 nn	T=4		
CMP $nHnL	绝对	CD nL nH	T=4		
CMP $nHnL,X	绝对X变址	DD nL nH	T=4 跨页T=5		
CMP $nHnL,Y	绝对Y变址	D9 nL nH	T=4 跨页T=5		
CMP ($nn,X)	先X变址后间址	C1 nn	T=6		
CMP ($nn),Y	先间址后Y变址	D1 nn	T=5 跨页T=6		
CPX #$nn	立即	E0 nn	T=2	X—M	V V V — — —
CPX $nn	零页	E4 nn	T=3		
CPX $nHnL	绝对	EC nL nH	T=4		
CPY #$nn	立即	C0 nn	T=2	Y—M	V V V — — —
CPY $nn	零页	C4 nn	T=3		
CPY $nHnL	绝对	EC nL nH	T=4		

4-4 8086/8088指令系统

4-4-1 寻址方式 8086和8088存取指令操作数的方式有许多种。操作数可以存在寄存器中，也可以包含在指令内，或者存在存贮器或I/O口中。指令可以使用不同的方法来计算存贮器和I/O口中的操作数的地址。8086和8088的寻址方式简述如下：

一、寄存器操作数和立即操作数

采用寄存器操作数的指令，含有寄存器的地址，寄存器可存放源操作数，也可以存放目的操作数，或者二者皆存。

立即操作数是包含在指令内的常数，其长度可以是8位或16位。立即操作数只能作为源操作数。取寄存器操作数和立即操作数时都不必执行总线周期，因此取数速度很快。

二、存贮器寻址方式

当执行单元（EU）需要读或写一个存贮器操作数时，必须将一个偏移量传送给总线接口单元（BIU）。BIU将该偏移量加到移位后的段寄存器的内容上，产生一个20位的实际地址，然后执行存取该操作数所需的总线周期。

在8086/8088中，把操作数与它所在段的起点的距离(以字节为单位)，即偏移量，称为该操作数的有效地址。

1.直接寻址

直接寻址是最简单的存贮器寻址方式。其有效地址直接取自指令的位移量字段，不涉及任何寄存器。

2.寄存器间接寻址

存贮器操作数的有效地址直接取自一个基址寄存器或变址寄存器。但在JMP或CALL指令中，进行寄存器间接寻址时，可以使用任何16位的通用寄存器。

3.基址寻址

在基址寻址方式中，有效地址是位移量与寄存器BX或寄存器BP的内容之和。

4.变址寻址

在变址寻址中，有效地址是根据位移量与变址寄存器(SI或DI)的内容之和算出的。

5.基址变址寻址

在基址变址寻址方式中，有效地址是基址寄存器、变址寄存器和位移量之和。

6.字符串寻址

字符串指令不用普通的存贮器寻址方式来访问其操作数，而是隐含地使用变址寄存器。在执行字符串指令时，总是认为SI指向源字符串的第一个字节或字，而DI指向目的字符串的第一个字节或字。在重复执行的字符串指令中，CPU自动地调整SI或DI的值，以获得后续的其余字节或字。

三、I/O口的寻址

要访问位于I/O空间内的I/O口，可采用两种寻址方式。一种是用直接口寻址方式，口地址是一个8位的立即数，这就许可固定地访问地址为0～255内的口。二是间接口寻址方式，它类似于存贮器操作数的寄存器间接寻址。口地址取自寄存器DX，其范围为0～65535。通过预先调整寄存器DX的内容，就可用一条指令访问I/O空间内的任何一个口。

4-4-2 8086和8088的指令系统 8086和8088的指令系统完全一样。此指令系统中包含有以前的微处理机（例如8080/8085)常用的指令或与之相当的指令，此外还增加了一些新的操

作。例如：

一、带符号和不带符号的二进制数和拆开的十进制数的乘法和除法。

二、长度可达64K字节的字符串的传送、扫描和比较操作。

三、将一个字节从一种代码翻译成另一种代码。

四、由软件产生的中断。

五、一组能协调多处理机系统的活动的指令。

这些指令能同样方便地处理不同类型的操作数。几乎所有的指令都既能处理字节数据，又能处理字数据。大多数指令中都可互换地规定寄存器操作数、存贮器操作数和立即操作数。特别值得提及的是，存贮器变量能在原处实现加数、减数、移位、比较等操作，而不必将其移入和移出寄存器。

表4-4 详细地列出了8086/8088指令系统的有关信息。表中列出的对标志的影响所使用的标识符含意如下所述：

空格	不改变任何标志
0	置成0
1	置成1
X	根据结果置1或置0
U	不定——含有不可靠的值
R	恢复成原先保存起来的值

操作数类型中使用的标识符，说明如下：

（无操作数）	不写任何操作数
寄存器	8位或16位的通用寄存器
reg16	16位的通用寄存器
seg-reg	段寄存器
累加器	寄存器AX或AL
立即数	0-FFFFH 范围内的常数
immed8	0-FFH 范围内的常数
存贮器	8位或16位的存贮单元
mem8	8位的存贮单元
mem16	16位的存贮单元
源转换表	长度为256个字节的转换表的名称
源字符串	由寄存器SI寻址的字符串的名称
目的地字符串	由寄存器DI寻址的字符串的名称
DX	寄存器DX
短距离标号	与指令末尾相距－128～＋127字节的一个标号
近程标号	在当前指令段中的标号
远程标号	在另一个指令段中的标号
近程过程	在当前指令段内的过程
远程过程	在另一个指令段内的过程
memptr16	存有控制将转向当前指令段中的单元的偏移量的字
memptr32	存有控制将转向另一个指令段中的单元的段基地址和偏移量的双字长

字

regtr16　　存有控制将转向当前指令段中的单元的偏移量的16位通用寄存器

repeat　　字符串指令的重复前级

表4-4　　8086/8088 指 令 系 统

AAA	AAA(无操作数) 加法的ASCII调整			标志 O D I T S Z A P C U U U × U ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	4	—	1	AAA

AAD	AAD (无操作数) 除法的ASCII调整			标志 O D I T S Z A P C U × × U × U
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	60	—	2	AAD

AAM	AAM (无操作数) 乘法的ASCII调整			标志 O D I T S Z A P C U × × U × U
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	83	—	1	AAM

AAS	AAS (无操作数) 减法的ASCII调整			标志 O D I T S Z A P C U U U × U ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	4	—	1	AAS

ADC	ADC 目的地，源 带进位加			标志 O D I T S Z A P C × × × × × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，寄存器	3	—	2	ADC AX,SI
寄存器，存贮器	9+EA	1	2—4	ADC DX,BETA[SI]
存贮器，寄存器	16+EA	2	2—4	ADC ALPHA[BX][SI],DI
寄存器，立即数	4	—	3—4	ADC BX,256
存贮器，立即数	17+EA	2	3—6	ADC GAMMA,30H
累加器，立即数	4	—	2—3	ADC AL,5

ADD	ADD 目的地，源 加法			标志 O D I T S Z A P C × × × × × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，寄存器	3	—	2	ADD CX,DX
寄存器，存贮器	9+EA	1	2—4	ADD DI,[BX],ALPHA
存贮器，寄存器	16+EA	2	2—4	ADD TEMP,CL
寄存器，立即数	4	—	3—4	ADD CL,2
存贮器，立即数	17+EA	2	3—6	ADD ALPHA,2
累加器，立即数	4	—	2—3	ADD AX,200

续表1

AND	AND 目的地，源 逻辑“与”			标志 O D I T S Z A P C 0 × ×U × 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，寄存器	3	—	2	AND AL,BL
寄存器，存贮器	9+EA	1	2—4	AND CX,FLAG_WORD
存贮器，寄存器	16+EA	2	2—4	AND ASCII[DI],AL
寄存器，立即数	4	—	3—4	AND CX,0F0H
存贮器，立即数	17+EA	2	3—6	AND BETA,01H
累加器，立即数	4	—	2—3	AND AX,01010000B

CALL	CALL 目标 调用一个过程			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
近程过程	19	1	3	CALL NEAR_PROC
远程过程	28	2	5	CALL FAR_PROC
memptr 16	21+EA	2	2—4	CALL PROC_TABLE[SI]
regptr 16	16	1	2	CALL AX
memptr 32	37+EA	4	2—4	CALL [BX],TASK[SI]

CBW	CBW（无操作数） 字节转换成字			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	CBW

CLC	CLC（无操作数） 进位标志置0			标志 O D I T S Z A P C 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	CLC

CLD	CLD（无操作数） 方向标志置0			标志 O D I T S Z A P C 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	CLD

CLI	CLI（无操作数） 中断标志置0			标志 O D I T S Z A P C 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	CLI

CMC	CMC（无操作数） 进位标志取反			标志 O D I T S Z A P C ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	CMC

续表 2

CMP	CMP目的地，源 目的地与源比较			标志 O D I T S Z A P C × × × × × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，寄存器	3	—	2	CMP BX,CX
寄存器，存贮器	9+EA	1	2—4	CMP DH,ALPHA
存贮器，寄存器	9+EA	1	2—4	CMP [BP+2],SI
寄存器，立即数	4	—	3—4	CMP BL,02H
存贮器，立即数	10+EA	1	3—6	CMP [BX],RADAR[DI],3420H
累加器，立即数	4	—	2—3	CMP AL,00010000B

CMPS	CMPS 目的地字符串，源字符串 字符串比较			标志 O D I T S Z A P C × × × × × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
目的地字符串，源字符串	22	2	1	CMPS BUFF1,BUFF2
(重复前缀)目的地字符串，源字符串	9+22/rep*	2/rep	1	REPF CMPS ID,KEY

×：rep 表示重复，以下同。

CWD	CWD（无操作数） 单字转换成双字			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	5	—	1	CWD

DAA	DAA（无操作数） 加法的十进制调整			标志 O D I T S Z A P C × × × × × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	4	—	1	DAA

DAS	DAS（无操作数） 减法的十进制调整			标志 O D I T S Z A P C U × × × × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	4	—	1	DAS

DEC	DEC 目的地 减1			标志 O D I T S Z A P C × × × × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
reg16	2	—	1	DEC AX
reg8	3	—	2	DEC AL
存贮器	15+EA	2	2—4	DEC ARRAY[SI]

DIV	DIV 源 无符号除法			标志 O D I T S Z A P C U U U U U
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
reg8	80—90	—	2	DIV CL
reg16	144—162	—	2	DIV BX
mem8	(86—96)+EA	1	2—4	DIV ALPHA
mem16	(150—168) +EA	1	2—4	DIV TABLE[SI]

续表 3

ESC	ESC外部操作码，源交权			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
立即数，存贮器	8+EA	1	2—4	ESC 6,ARRAY[SI]
立即数，寄存器	2	—	2	ESC 20,AL

HLT	HLT（无操作数）停机			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	HLT

IDIV	IDIV 源 整数除法			标志 O D I T S Z A P C U U U U U U
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
reg8	101—112	—	2	IDIV BL
reg16	165—184	—	2	IDIV CX
mem8	(107—118)+EA	1	2—4	IDIV DIVISOR_BYTE[SI]
mem16	(171—190)+EA	1	2—4	IDIV [BX],DIVISOR WORD

IMUL	IMUL 源 整数乘法			标志 O D I T S Z A P C × U U U U ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
reg8	80—98	—	2	IMUL CL
reg16	128—154	—	2	IMUL BX
mem8	(86—104)+EA	1	2—4	IMUL RATE_BYTE
mem16	(134—160)+EA	1	2—4	IMUL RATE_WORD[BP][DI]

IN	IN 累加器，口地址 输入字节或字			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
累加器，immed8	10	1	2	IN AL,0FFEAH
累加器，DX	8	1	1	IN AX,DX

INC	INC 目的地 加1			标志 O D I T S Z A P C × × × × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
reg16	2	—	1	INC CX
reg8	3	—	2	INC BL
存贮器	15+EA	2	2—4	INC ALPHA[DI][BX]

INT	INT 中断类型 中断			标志 O D I T S Z A P C 0 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
immed8（类型码=3）	52	5	1	INT 3
immed8（类型码≠3）	51	5	2	INT 67

续表 4

INTR+	INTR 外部可屏蔽中断 若INTR和IF=1，则中断			标志 O D I T S Z A P C 0 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	61	7	N/A	N/A

INTO	INTO (无操作数) 溢出中断			标志 O D I T S Z A P C 0 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	53或4	5	1	INTO

IRET	IRET (无操作数) 中断返回			标志 O D I T S Z A P C R R R R R R R R R
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	24	3	1	IRET

JA/JNBE	JA/JNBE 短距离标号 高于转移/不低于，等于转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JA ABOVE

JAE/JNB	JAE/JNB短距离标号 高于或等于转移/不低于转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JAE ABOVE_EQUAL

JB/JNAE	JB/JNAE短距离标号 低于转移/不高于等于转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JB BELOW

JBE/JNA	JBE/JNA短距离标号 低于或等于转移/不高于转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JNA NOT_ABOVE

JC	JC 短距离标号 进位位为1转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JC CARRY_SET

续表5

JCXZ	JCXZ短距离标号 CX为0转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	18或6	—	2	JCXZ COUNT_DONE

JE/JZ	JE/JZ短距离标号 等于转移/为0转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JZ ZERO

JG/JNLE	JG/JNLE 短距离标号 大于转移/不小于、等于转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JG GREATER

JGE/JNL	JGE/JNL 短距离标号 大于或等于转移/不小于转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JGE GREATER_EQUAL

JL/JNGE	JL/JNGE 短距离标号 小于转移/不大于、等于转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JL LESS

JLE/JNG	JLE/JNG 短距离标号 小于或等于转移/不大于转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JNG NOT_GREATER

JMP	JMP 目标 转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	15	—	2	JMP SHORT
近 程 标 号	15	—	3	JMP WITHIN_SEGMENT
远 程 标 号	15	—	5	JMP FAR_LABEL
memptr16	18+EA	1	2—4	JMP [BX],TARGET
regptr16	11	—	2	JMP CX
memptr32	24+EA	2	2—4	JMP OTHER,SEG[SI]

JNC	JNC 短距离标号 进位位为0转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JNC NOT_CARRY

续表 6

JNE/JNZ	JNE/JNZ 短距离标号 不等于转移/不为0转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JNE HOT_EQUAL

JNO	JNO 短距离标号 无溢出转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JNO NO_OVERFLOW

JNP/JPO	JNP/JPO 短距离标号 P为0转移/奇状态转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JPO ODD_PARITY

JNS	JNS 短距离标号 S为0转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JNS POSITIVE

JO	JO 短距离标号 溢出转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JO SIGNED_OVRFLW

JP/JPE	JS/JPE短距离标号 P为1转移/偶状态转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JPE EVEN_PARITY

JS	JS 短距离标号 S为1转移			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	16或4	—	2	JS NEGATIVE

LAHF	LAHF（无操作数） 标志装入AH寄存器			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	4	—	1	LAHF

续表7

LDS	LDS 目的地，源 设定使用数据段的指针			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
reg16, mem32	16+EA	2	2—4	LDS SI,DATA,SEG[DI]

LEA	LEA 目的地，源 装入有效地址			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
reg16,mem16	2+EA	—	2—4	LEA BX,[BP][DI]

LES	LES 目的地，源 设定使用附加段的指针			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
reg16,mem32	16+EA	2	2—4	LES DI,[BX],TEXT_BUFF

LOCK	LOCK（无操作数） 总线封锁			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	LOCK XCHG FLAG,AL

LODS	LODS 源字符串 装入字符串			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
源字符串 （重复前缀）源字符串	12 9+13/rep	1 1/rep	1 1	LODS CUSTOMER_NAME REP LODS NAME

LOOP	LOOP 短距离标号 循环			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	17/5	—	2	LOOP AGAIN

LOOPE/LOOPZ	LOOPE/LOOPZ 短距离标号 等于循环/为0循环			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	18或6	—	2	LOOPE AGAIN

LOOPNE/LOOPNZ	LOOPNE/LOOPNZ 短距离 标号 不等于循环/不为0循环			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
短距离标号	19或5	—	2	LOOPNE AGAIN

续表 8

NMI*	NMI(外部非屏蔽中断) 若NMI=1，则中断			标志 O D I T S Z A P C 0 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	50	5	N/A	N/A

MOV	MOV 目的地，源 传送			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
存贮器，累加器	10	1	3	MOV ARRAY[SI],AL
累加器，存贮器	10	1	3	MOV AX,TEMP_RESULT
寄存器，寄存器	2	—	2	MOV AX,CX
寄存器，存贮器	8+EA	1	2—4	MOV BP,STACK_TOP
存贮器，寄存器	9+EA	1	2—4	MOV COUNT[DI],CX
寄存器，立即数	4	—	2—3	MOV CL,2
存贮器，立即数	10+EA	1	3—6	MOV MASK[BX][SI],2CH
seg-reg,reg16	2	—	2	MOV ES,CX
seg-reg,mem16	8+EA	1	2—4	MOV DS,SEGMENT_BASE
reg16,seg-reg	2	—	2	MOV BP,SS
存贮器,seg-reg	9+EA	1	2—4	MOV [BX],SEG_SAVE,CS

MOVS	MOVS 目的地字符串，源字符串 字符串传送			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
目的地字符串，源字符串 (重复前缀)目的地字符串，源字符串	18 9+17/rep	2 2/rep	1 1	MOVS LINE EDIT_DATA REP MOVS SCREEN,BUFFER

MOVSB/MOVSW	MOVSB/MOVSW(无操作数) 字符串(字节/字)传送			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数) (重复前缀)无操作数	18 9+17/rep	2 2/rep	1 1	MOVSB REP MOVSW

MUL	MUL 源 无符号乘法			标志 O D I T S Z A P C × U U U U ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
reg8	70—77	—	2	MUL BL
reg16	118—133	—	2	MUL CX
mem8	(76—83)+EA	1	2—4	MUL MONTH[SI]
mem16	(124—139) +EA	1	2—4	MUL BAUD_RATE

NEG	NEG 目的地 求补			标志 O D I T S Z A P C × × × × ×1*
操 作 数	时钟周期数	传送次数*	字 节	编 码 例 子
寄存器 存贮器	3 16+EA	— 2	2 2—4	NEG AL NEG MULTIPLIER

* 若目的地=0，则本栏为0。

续表 9

NOP	NOP(无操作数) 空操作			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	3	—	1	NOP

NOT	NOT目的地 逻辑"非"			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器 存贮器	3 16+EA	— 2	2 2—4	NOT AX NOT CHARACTER

OR	OR 目的地，源 逻辑"或"			标志 O D I T S Z A P C 0 × × U × 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，寄存器	3	—	2	OR AL,BL
寄存器，存贮器	9+EA	1	2—4	OR DX,PORT_ID[DI]
存贮器，寄存器	16+EA	2	2—4	OR FLAG_BYTE,CL
累加器，立即数	4	—	2—3	OR AL,01101100B
寄存器，立即数	4	—	3—4	OR CX,01H
存贮器，立即数	17+EA	2	3—6	OR [BX],CMD_ WORD,0CFH

OUT	OUT 口地址，累加器 输出字节或字			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
immed8，累加器 DX，累加器	10 8	1 1	2 1	OUT 44,AX OUT DX,AL

POP	POP 目的地 将字从堆栈弹出			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器	8	1	1	POP DX
seg-reg(cs非法)	8	1	1	POP DS
存贮器	17+EA	2	2—4	POP PARAMETER

POPF	POPF (无操作数) 将标志从堆栈弹出			标志 O D I T S Z A P C R R R R R R R R R
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	8	1	1	POPF

续表10

PUSH	PUSH 源 将字压入堆栈			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器	11	1	1	PUSH SI
seg-reg(cs合法)	10	1	1	PUSH ES
存贮器	16+EA	2	2—4	PUSH RETURN—CODE[SI]

PUSHF	PUSHF（无操作数） 将标志压入堆栈			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	10	1	1	PUSHF

RCL	RCL 目的地，计数值 通过进位循环左移			标志 O D I T S Z A P C × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，1	2	—	2	RCL CX,1
寄存器，CL	8+4/位	—	2	RCL AL,CL
存贮器，1	15+EA	2	2—4	RCL ALPHA,1
存贮器，CL	20+EA+4/位	2	2—4	RCL [BP],PARM,CL

RCR	RCR 目的地，计数值 通过进位循环右移			标志 O D I T S Z A P C × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，1	2	—	2	RCR BX,1
寄存器，CL	8+4/位	—	2	RCR BL,CL
存贮器，1	15+EA	2	2—4	RCR [BX],STATUS,1
存贮器，CL	20+EA+4/位	2	2—4	RCR ARRAY[DI],CL

REP	REP （无操作数） 重复字符串操作			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	REP MOVS DEST,SRCE

SAR	SAR 目的地，计数值 算术右移			标志 O D I T S Z A P C × × × U × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，1	2	—	2	SAR DX,1
寄存器，CL	8+4/位	—	2	SAR DI,CL
存贮器，1	15+EA	2	2—4	SAR N_BLOCKS,1
存贮器，CL	20+EA+4/位	2	2—4	SAR N_BLOCKS,CL

续表11

REPE/REPZ	REPE/REPZ（无操作数） 等于/为0时重复字符串操作			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	REPE CMPS DATA KEY

REPNE/REPNZ	REPNE/REPNZ（无操作数） 不等于/不为0时重复字符串操作			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	REPNE SCAS INPUT_LINE

RET	REL 任选弹出值 从过程返回			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（段内，无弹出值）	8	1	1	RET
（段内，有弹出值）	12	1	3	RET 4
（段间，无弹出值）	18	2	1	RET
（段间，有弹出值）	17	2	3	RET 2

ROL	ROL 目的地，计数值 循环左移			标志 O D I T S Z A P C × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，1	2	—	2	ROL BX,1
寄存器，CL	8+4/位	—	2	ROL DI,CL
存贮器，1	15+EA	2	2—4	RCL FLAG_BYTE[DI],1
存贮器，CL	20+EA+4/位	2	2—4	ROL ALPHA,CL

ROR	ROR 目的地，计数值 循环右移			标志 O D I T S Z A P C × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，1	2	—	2	ROR AL,1
寄存器，CL	8+4/位	—	2	ROR BX,CL
存贮器，1	15+EA	2	2—4	ROR PORT_STATUS,1
存贮器，CL	20+EA+4/位	2	2—4	ROR CMD_WORD,CL

SAHF	SAHF（无操作数） 将AH内容存入标志寄存器			标志 O D I T S Z A P C R R R R R
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	4	—	1	SAHF

续表12

SAL/SHL	SAL/SHL 目的地，计数值 算术左移/逻辑左移			标志 O D I T S Z A P C × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，1	2	—	2	SAL AL, 1
寄存器，CL	8+4/位	—	2	SHL DI, CL
存贮器，1	15+EA	2	2—4	SHL [BX],OVERDRAW,1
存贮器，CL	20+EA+4/位	2	2—4	SAL STORE_COUNT,CL

SBB	SBB 目的地，源 带借位减			标志 O D I T S Z A P C × × × × × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，寄存器	3	—	2	SBB BX,CX
寄存器，存贮器	9+EA	1	2—4	SBB DI,[BX],PAYMENT
存贮器，寄存器	16+EA	2	2—4	SBB BALANCE,AX
累加器，立即数	4	—	2—3	SBB AX,2
寄存器，立即数	4	—	3—4	SBB CL,1
存贮器，立即数	17+EA	2	3—6	SBB COUNT[SI],10

SCAS	SCAS 目的地字符串 字符串扫描			标志 O D I T S Z A P C × × × × × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
目的地字符串 (重复前缀)目的地字符串	15 9+15/rep	1 1/rep	1 1	SCAS INPUT_LINE REPNE SCASBUFFER

SEGMENT++	SEGMENT 越界前缀 跨至规定的段			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	2	—	1	MOV SS:PARAMETER,AX

SHR	SHR 目的地，计数值 逻辑右移			标志 O D I T S Z A P C × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，1	2	—	2	SHR SI,1
寄存器，CL	3+4/位	—	2	SHR SI,CL
存贮器，1	15+EA	2	2—4	SHR ID_BYTE[SI][BX],1
存贮器，CL	20+EA+4/位	2	2—4	SHR INPUT_WORD,CL

SINGLE STEP+	SINGLE STEP (陷阱标志中断) 若TF=1则中断			标志 O D I T S Z A P C 0 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
(无操作数)	50	5	N/A	N/A

续表13

STC	STC（无操作数） 进位标志置 1			标志 O D I T S Z A P C 1
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	STC

STD	STD（无操作数） 方向标志置1			标志 O D I T S Z A P C 1
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	STD

STI	STI（无操作数） 中断开放标志置1			标志 O D I T S Z A P C 1
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	2	—	1	STI

STOS	STOS 目的地字符串 存储字节串或字串			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
目的地字符串 （重复前缀）目的地字符串	11 9+10/rep	1 1/rep	1 1	STOS PRINT_LINE REP STOSDISPLAY

SUB	SUB 目的地，源 减法			标志 O D I T S Z A P C × × × × × ×
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，寄存器	3	—	2	SUB CX,BX
寄存器，存贮器	9+EA	1	2—4	SUB DX,MATH_TOTAL[SI]
存贮器，寄存器	16+EA	2	2—4	SUB [BP+2],CL
累加器，立即数	4	—	2—3	SUB AL,10
寄存器，立即数	4	—	3—4	SUB SI,5280
存贮器，立即数	17+EA	2	3—6	SUB [BP],BALANCE,1000

TEST	TEST 目的地，源 测试或非破坏性逻辑“与”			标志 O D I T S Z A P C 0 × × U × 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，寄存器	3	—	2	TEST SI,DI
寄存器，存贮器	9+EA	1	2—4	TEST SI,END_COUNT
累加器，立即数	4	—	2—3	TEST AL,00100000B
寄存器，立即数	5	—	3—4	TEST BX,0CC4H
存贮器，立即数	11+EA	—	3—6	TEST RETURN_CODE,01H

WAIT	WAIT—（无操作数） 等待至TEST信号有效为止			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
（无操作数）	3+5n	—	1	WAIT

续表14

XCHG	XCHG 目作地，源 交换			标志 O D I T S Z A P C
作 操 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
累加器，reg16	3	—	1	XCHG AX,BX
存贮器，寄存器	17+EA	2	2—4	XCHG SEMAPHORE,AX
寄存器，寄存器	4	—	2	XCHG AL,BL

XLAT	XLAT 源转换表 转换			标志 O D I T S Z A P C
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
源转换表	11	1	1	XLAT ASCII_TAB

XOR	XOR 目的地，源 逻辑"异"			标志 O D I T S Z A P C 0 × × U × 0
操 作 数	时钟周期数	传送次数*	字 节 数	编 码 例 子
寄存器，寄存器	3	—	2	XOR CX,BX
寄存器，存贮器	9+EA	1	2—4	XOR CL,MASK_BYTE
存贮器，寄存器	16+EA	2	2—4	XOR ALPHA[SI],DX
累加器，立即数	4	—	2—3	XOR AL,01000010B
寄存器，立即数	4	—	3—4	XOR SI,00C2H
存贮器，立即数	17+EA	2	3—6	XOR RETURN_CODE,0D2H

* 对8086而言，以奇地址传送一个16位的字需增加四个时钟周期。对8088而言，每传送一个16位的字需增加四个时钟周期。

\+ INTR、NMI和SINGLE STEP不是指令；列在表中只是为了提供定时信息。

\++ ASM-86把段越界前缀归入操作数内，而不作为独立的指令。SEGMENT列在表中只是为了提供定时信息。

参 考 书 目

〔1〕《微型计算机硬件软件及其应用》 周明德 编著 清华大学出版社

〔2〕《微型计算机》 朱绍庐 主编 人民交通出版社

〔3〕《微型计算机—Z-80》 科学技术文献出版社重庆分社

〔4〕《Intel 8086 微型计算机》 科学技术文献出版社重庆分社

〔5〕《微型计算机的组装和使用》〔日〕 石田晴久著 周琴芳 王超平译 科学出版社

〔6〕《8086初阶体系结构、系统设计和程序设计入门》〔美〕 S.P.莫尔斯著 高志伟译 科学出版社

〔7〕《电子技术应用译文》 1976年第2期

〔8〕《计算机工程与应用》 1982年第6期，第11期

〔9〕《6502微处理机及其应用》 荣树熙 张开敬编 北京师范大学出版社

〔10〕《THE Z8000 MICROPROCESSOR A Design Handbook》
BRADLY K. FAWCETT

本 篇 附 录

附录一 美国二十家主要微型机制造商生产的部分产品一览表(1984年)

产品名称	地址空间	主存容量	操作系统	售价(美元)
IBM公司				
IBM PC	16位	64K	PC-DOS, CP/M 86	3,353
IBM XT	16位	128K	PC-DOS, CP/M	5,715
Commodore公司				
Pet 4032	8位	32K	嵌入式DOS	1,295
CBM商用系统	8位	32K	嵌入式DOS	4,285
Super Pet	8位和16位	96K	嵌入式DOS	1,995
数字设备公司				
Rainbow 100	8位和16位	64K～256K	CP/M 86/80, MS-DOS	2,695
Professional 300	16位	256K	P/OS, USCD Pascal	3,995
Osborne 计算机公司				
Osborne 1	8位	64K	CP/M	1,795
Osborne Executivne 1	8位	128K	P-System, CP/M Plus	2,495
Osborne Executive 11	8位和32位	256K	CP/M86,MS-DOS,P-SyStem	3,195
Xerox 公司				
820-11	8位	64K	CP/M 80	2,595
16/8	8位和16位	64K～256K	CP/M 80,CP/M86,MS-DOS	3,395
Apple 计算机公司				
Apple 11和Apple+	8位	16K～48K	DOS	1,530
Apple 111	8位	128～256K	SOS, CP/M	3,495
Macintosh	32位	128K～512K		2,495
Lisa 2		512K～1M		3,495
Lisa 2/5		512K～1M		4,495
Lisa 2/10		512K～1M		5,495
Intertec 数据系统公司				
Compuster	3位	64K	CP/M	1,995
Superbrain 11	8位	64K	CP/M 2.2	2.495
HP 公司				
HP 85	8位	16K～32K	组合到系统	2,750
HP86、HP87	8位	64K～640K	HP, CP/M	1,795
HP 125	8位	64K	CP/M	2,750
Tandy 公司的 Radio Shack				
TRS-80, 1型	8位	4K～48K	TRS-DOS	2,500
TRS-80, 2型	8位	380K～1.8M	TRS-DOS	3,499

续表

产品名称	地址空间	主存容量	操作系统	售价（美元）
Televideo系统公司				
TS 806	8位	64K～128K	CP/M 2.2	7,195
TS 816	8位	64K～128K	CP/M 2.2	12,995
Zenith无线电公司				
Zenith 89	8位	48K～64K	CP/M 2.2, USCD Pascal	2,895
Zenith 90	8位	48K～64K	CP/M 2.2, USCD Pascal	3,195
Z 100	8位和16位	128K～768K	Z-DOS, CP/M 86	3,249
Cromemco公司				
系统 1	8位	64K～512K	Cdos或Cromix	3,995
系统 2	8位	64K～512K	Cdos或Cromix	4,695
系统 3	8位	64K～512K	Cdos或Cromix	6,995
D系列系统 1	16位	256K～2M	Cdos或Cromix	4,995
D系列系统 2	16位	256K～2M	Cdos或Cromix	5,995
D系列系统 3	16位	256K～2M	Cdos或Cromix	7,995
TI公司				
TI-99/4A	8位	16K～48K	内部的	1,500
Franklin计算机公司				
ACE 1000	8位	64K	Apple Dos 3.3	1,595
Convergent技术公司				
1000	16位	128K～1M	Ctos	3,990
Altos计算机系统公司				
Altos 8000	8位	64K～204K	CP/M, MP/M, Oasis	1,200
Vector Graphic 公司				
Vector 4	8位和16位	128K～256K	CP/M, CP/M 86, MS-DOS	7,145
NEC信息系统公司				
PC-8000	8位	32K～64K	Dos, CP/M	1,205
APC	16位	128K～256M	CP/M 86	4,693
North Star计算机公司				
TSS	8位	128K～1.4M	TSS/A, TSS/B, TSS/C	9,241
Horizon	8位	32K～64K	Monitor	14,800
三洋商用系统公司				
FDS-1000	8位	64K	CP/M 2.2	1,995
MBS系列	8位	64K	CP/M 2.2	3,493

注：摘自〈计算机世界〉1984年 4 月

附录二　日本微型机主要生产厂家及机型一览表

生产厂家	机型	CPU	内存	
			RAM	ROM
爱电子测量公司	16位ai－m16＋VDU－140/CG	8086　8088　8089	512千字节	164千字节
	ai-MM16W	8086　8089	256千字节	16千字节
	ABC-26	Z-80A	64千字节	4千字节
	ABC-260W	Z-80A	320千字节	4千字节
日本Apple计算机公司	Apple IIe	6502	64千字节	16千字节
	Apple II-Jplus	6502	48千字节	16千字
	Lisa	68000	1兆字节	
冲电气工业公司	if800 30型	Z-80B	128～256千字节	
卡西欧计算机公司	FP-5700/8	Z-80	229千字节	16千字节
夏普公司	MZ-3500	Z-80AX2	128千字节	16千字节
	OA-8100	MC-6800	640千字节	16千字节
日立制作所	MB16001A	i-8088	320千字节	16千字节
	B-16系列	i-8086	256千字节	24千字节
富士通公司	FM-11	MBL6809EX2	128千字节最大1兆字节	
	9450-III	MN 1613	256千字节	8千字节
横河-HP公司	HP-87	HP专用微处理器	128～640千字节	48千字节
	HP-86	HP专用微处理器	64～576千字节	48千字节
	HP200系列165型	MC68000	128千字节～7.4兆字节	48千字节
	HP200系列26A型　36A型	MC6800	128千字节～7.4兆字节	48千字节
	HP200系列 36C	同上 MC6800	同上 同上	同上 同上
术德公司	M68	68000　Z-80A	256千字节～1兆字节	
	M343	8086	256～768千字节	
	M685	68000	512千字节或1兆字节	
精工舍公司	SEIKO 3300系列	i-8088-2i-8085A-2 i-8088	256～512千字节	8千字节
	SEIKO 9100系列	i-8086 i-8087 i-8088	384千字节最大896千字节	16千字节
	SEIKO 9500系列	i-8086 i-8087 i-8088	256～768千字节	32千字节
	SEIKO 9500K系列	i-8086 i-8087 i-8088	384～768千字节	8千字节
	SEIKO 5900系列	i-8085	128千字节	2千字节
三洋电机特机公司	MBC-200	Z-80AX,2	96千字节	8千字节
	MBC-5030	i-8086	128千字节	16千字节
	MBC-5130	i-8086	同上	同上
	MBC-100	Z-80	64千字节	2千字节
	MBC-55	i-8088	128千字节	8千字节
东芝公司	16	i-8088	256千字节	
日本电气公司	PC-8801	μPD780C-1	112千字节	72千字节
	PC-9801	μPD8086	128千字节～640千字节	96千字节
日本IBM公司	5550	i-8086	256千字节～512千字节	16千字节
日本DEC公司	100	Z-80 8088	64千字节～256千字节	16千字节
索尼公司	SMC-70	Z-80A	102千字节	48千字节
日本乐器制造公司	PU-1-20	YM-2002	64千字节	4千字节
	PU-1-20E	YM2002	128千字节	4千字节
	PU-10	YM2002	同上	同上
松下电器产业公司	3000	i-8088	128千字节	16千字节
	C-7000系列	MN1613X2	256～512千字节	8千字节

附录三　IBM　PC兼容机性能一览表

机　　型	标价	处　理　器	是否有8087插槽	RAM		操　作　系　统
				(最小)	(最大)	
IBM　PC	$3245	8088	是	64K	640K	PC与MS-DOS 1.10/2.00/2.10CP/M-86,CCP/M-86,USCDP,
台　　式　　机						
CAD Counsel Protean	$7300	8088	是	256K	900K	MS-DOS 1.10/2.00/2.10,CP/M-86
Canon AS-100	$3995	8088	否	128K	512K	MS-DOS 1.25,CP/M-86
Digigraphic EXTRA Model 70	$5995	8088	是	256K	640K	MS-DOS1.00/1.25/2.00/2.10, CP/M-86, UCSDP
Electro Design IMP-12	$4044	8088	是	64K	750K	MS-DOS 1.10/2.00,CP/M-86
Future Computers FX-20	$2995	8088	是	128K	1M	MS-DOS 1.10/2.00,CP/M-86, CCP/M-86
Intertec Head Start 512	$3495	8086　Z-80A	否	512K	1M	MS-DOS 1.10/2.00/2.10, CP/M-80/86
ITT XTRA	$3500	8088	是	128K	640K	MS-DOS 1.10/2.00/2.10, CP/M-86, UCSDp
Leading Edge PC	$2895	8088	是	128K	640K	MS-DOS 1.25/2.01,CP/M-86, CCP/M-86, UCSDP
Logical LX-T	$5985	8088	是	192K	1M	MS-DOS 2.00
MAD-1	$4195	80186	是	128K	704K	MS-DOS 2.00,CP/M-86,CCP/M-86
Micro Craft Dimension	$3995	MC68000, L8, 8086	是	128K	16.7M	MS-DOS 1.10/2.00,CP/M-80/86, MP/M-80,UCSDP,UNIX
NEC PC-8801A	$3092	8086 Z-80A	否	128K	128K	MS-DOS 2.00,CP/M2.2
Olivetti PC	$3295	8088	是	128K	512K	MS-DOS 1.25/2.00,CP/M-86,UCSDP
Olympia People	$3595	8086	是	128K	512K	MS-DOS 1.10/2.00/2.10, CP/M-86, CCP/M-86
OSM-PC	$2995	8088	否	128K	1M	MS-DOS 2.00
Polo	$3995	80188 Z-80A	否	128K	768K	MS-DOS 2.00/2.11,CP/M-80/86
Pronto	$3200	80186	否	256K	1M	MS-DOS 2.00/2.11
Radio Shack TRS-80 2000	$2999	80186	否	128K	768K	MS-DOS 1.10/2.00
Seattle Computer Gazellell	$6995	8086	是	256K	1M	MS-DOS 1.25/2.00/2.10, CP/M-86, MP/M-86, UCSDp
Sperry PC	$3119	8088	是	128K	640K	MS-DOS 1.25/2.00,CP/M-86,UCSDP
Stearns Desktop	$2945	8086	是	128K	896K	MS-DOS 1.25/2.00,CP/M-86, CCP/M-86,MP/M-86, ST-DOS
Tava PC	$1995	8088	是	128K	640K	MS-DOS 1.10/2.00,CP/M-86,UCSDP
Toshiba T-300	$2795	8088	是	192K	512K	MS-DOS 1.10/2.00/2.10,CP/M-86
Victor 9000	$3495	8088	是	128K	896K	MS-DOS 1.10/2.00/2.10,CP/M-86
Wyse WY-1000	$3090	80186	否	128K	768K	MS-DOS 2.10

续表

机　　型	标价	处　理　器	是否有8087插槽	RAM		操　作　系　统
				(最小)	(最大)	
IBM　PC	$3245	8088	是	64K	640K	PC与MS-DOS1.10/2.00/2.10CP/M-86,CCP/M-86,USCDP,
便　　携　　机						
ACT Apricot	$3190	8086, 8089	是	256K	768K	MS-DOS 2.00,CP/M-86,CCP/M-86, UCSDP
Colby PC3.2	$2795	8088	是	128K	640K	MS-DOS 1.10,CP/M-86,UCSDP 2.0/2.1
ColumbiaVP	$2995	8088	是	128K	512K	MS-DOS 1.25,CP/M-80/86, CCP/M-86,MP/M-86
COMPAQ-Plus	$4995	8088	是	128K	640K	MS-DOS 1.10/2.00/2.10,CP/M-86, UCSDp
Compucorp OmegaMite	$2995	80188 Z-80A	否	64K	512K	MS-DOS 2.00,CP/M-86,Omega DOS
Corona Portable PC	$2545	8088	是	128K	1M	MS-DOS 1.10/1.25,CP/M-86 UCSDP
Eagle PC Spirit	$3295	8088	是	128K	1M	MS-DOS 1.10/2.00/2.10,CPM-86, CCP/M-86,UCSDp
GRiD Tempest	$ 14,995	8006　8087	是	256K	256K	MS-DOS 2.00,GRiD OS
Hyperion	$3195	8088	是	256K	640K	MS-DOS 1.10/2.00,CP/M-86
JONOS 2150i	$3695	8088	否	64K	1M	MS-DOS 1.10/2.00/2.10/3.00
Otrona Attache 8:16	$3495	8086 Z-80A	是	256K	256K	MS-DOS 2.10,CP/M-86
Panasonic Sr. Partner	$2495	8088	是	128K	512K	MS-DOS 1.10/2.00/2.10
Seequa Chameleon	$1995	8088 Z-80A	是	128K	704K	MS-DOS 1.25/2.00,CP/M 2.1/2.2, CCP/M-86,CP/M-86,UCSDp
Sharp PC-5000	$1995	8088	否	128K	256K	MS-DOS 1.10/2.00
STM Pied Piper	$3000	80186	否	256K	512K	MS-DOS 1.10/2.00/2.10,UCSDP
Strategic Technologies PC Traveler	$4495	双 80186	否	128K	1M	MS-DOS2.00
TeleVideo TPC-11	$2995	8088	是	128K	640K	CP/M-86,TeleDOS 2.11,UCSDP
TI Portable Professional	$2395	8088	是	64K	768K	MS-DOS 1.10/2.00/2.10, CP/M-86, CCP/M-86,UCSDp
Visual Commuter	$1995	8088	是	128K	1M	MS-DOS 1.10/2.00/2.10, CP/M-86, CCP/M-86,UCSDp
Xerox 1810/1850	$2495	8088, Z-80,	否	128K	512K	MS-DOS 2.00/2.10,CP/M-80/86
		NSC-800				

附录四　部分国产八位微型机厂家一览表

生　产　厂　家	型　　号	CPU
沈阳辽河实验研究所	DJS-060系列	MC6800
四川固体电路研究所	DJS-060系列	MC6800
上海计算机厂	DJS-054	8080A
南京有线电厂	DJS-033	6502
	紫金——II	6502
北京有线电厂	DJS－044	Z80
	DJS-045	Z80A
北京工业大学电子厂	TP-803	Z80
北京计算机五厂	BCM-II	Z80
	BCM-III	Z80
天津无线电二厂	DJS-065A	6800
	LWJ-064	6800
	DJS-062T	6800
广州计算机厂	DJS－28	Z80
	PZ80	Z80
潍坊计算机厂	DJS-303	6502
广东韶关无线电厂		6502、Z80
	PJ-1	8080
北京计算技术研究所	BCMI、II、III	Z80
福州无线电三厂	FO-3	6502
南通计算机厂	MIC	Z80
华南师大微电子学研究所实验工厂	DJS-083A	6502、Z80
	DJS-033	6502

第二篇　程序设计语言

第一章　BASIC语言

1—1　基本概念

BASIC是英语 Beginner's All-purpose Symblic Instruction Code（初学者通用指令代码）的缩写。它是1964年由Dartmouth大学的J. G. Kemeny和T. E. Kurty两位教授创立的。以后在广泛应用中不断修改和扩充，增强了功能和适应性。BASIC可分为单用户BASIC、单用户扩充BASIC和多用户扩充BASIC等。

BASIC可用于各种数值计算、数据处理、辅助设计、自动控制，也可用于计算机仿真、情报检索和实时控制等方面。扩充BASIC还具有某些独到之处，如：字符串操作、字符串处理函数、矩阵运算函数、作图、演奏音乐和较强的文件处理系统。

本章包括APPLE II的整数BASIC、SOFT BASIC、MBASIC、GBASIC、IBM PC BASIC及CROMEMCO BASIC。其中以APPLE SOFT的小巧、灵活、功能比较完备，尤以具有游戏语句和高清晰度图形而受到广泛重视。所以本章的叙述将以 APPLE SOFT 为主，目录中列出的是 APPLE SOFT 除图形和文件外的所有语句。MBASIC、GBASIC 是 MicroSOFT BASIC的 CP/M 版本。MBASIC 具有格式输出语句、交换变量值语句、较强的字符串处理和循环能力。GBASIC 包含了 MBASIC 的全部功能，且具备高清晰度绘图功能。IBM PC BASIC和GBASIC用途非常接近。本章的前 7 节的BASIC内容中，先叙述 APPLE SOFT的基本成份或语句，其他版本与之不同之处按MBASIC、IBM PC BASIC、CROMEMCO BASIC的顺序一一列举；凡与之相近的语句，也按此顺序列在其后。1—8是本文中三种机器的其他语句。1-12是数据文件，因各版本差异较大，所以按版本分别列出。CROMEMCO机没有绘图能力。

1-1-1　语法定义符

语法定义符	含义
=	表示左边术语的定义
〔 〕	表示括号内的内容可有可无
|	表示分开可任择其一的各项
{ }	表示括号里的内容可以重复
␣	表示一个必要的空白
< >	表示括号内是语法成份
—*	必要的下横线，表示操作人员输入的命令。

*注：带有标号的程序行不使用下横线说明。无下横线的则表示是计算机输出的信息。在第三篇的叙述中也使用此规定。

以上符号是为了方便语句格式的描述，在写程序时并不使用它。

1-1-2 BASIC 语言的基本符号 每一种高级语言，都有一套特定的基本符号。BASIC应用ASCII（美国信息交换标准代码）码的字符集作为基本符号，它们基本上都反映在机器的键盘上。

各种BASIC语言几乎都包括下列符号：

数字	0…… 9	
字符	A……Z	
标点符号	.	（小数点或句号）
	,	（逗号）
	;	（分号）
	(	（左圆括号）
	)	（右圆括号）
	"	（引号，串定界符）
	:	（冒号）
特定符号	%	（百分号或整数说明符）
	$	（钱币符或字符串说明符）
	?	（问号）
	#	（数目符号或磅符号）
	!	（惊叹号）
	&	
	@	（At—记号）
比较符号	=	（等号或赋值号）
	<=	（小于或等于）
	>	（大于号）
	>=	（大于或等于）
	<>	（不等于）
	<	（小于号）
算术运算符	+	（加号）
	-	（减号）
	*	（乘号）
	/	（除号）
	\	（整除）
	∧	（乘方，也有用↑符号的）
	MOD	（求模，即求余数）
逻辑算符	AND	（逻辑乘）
	OR	（逻辑加）
	NOT	（逻辑非）
	XOR	（逻辑异或）
	EQV	（逻辑同或）
	IMP	（逻辑隐含）

1-1-3 BASIC 程序的结构 BASIC 程序是由一系列程序行构成的。一个程序行则由行

号和一个或多个语句组成，语句之间用“:”号分隔。程序和程序行用语法定义符表示如下：

程序:=｛程序行｝

程序行:=〈行号〉〈语句〉〔｛:语句｝〕↙

行号是顺序号，也是一行的标号。无论你按什么顺序打入一个程序，在通常情况下，计算机都会依照由小到大的行号顺序存贮和执行。但遇到控制语句时，可以按指定行号改变程序的执行顺序。

语句是由语句定义符和语句体组成的。BASIC语句一般分为非执行语句和可执行语句两类。

语句定义符规定了计算机执行某一特定的功能。语句体是跟在语句定义符后面的需要执行的具体内容。

符号‘↙’代表RETURN键，RETURN是语句终止符。

语句终止符是程序行结束的标志。因为每一行的末尾都要有一个结束符，即使不写出来，也是清楚的，所以本章以后将略去在每行语句末尾的终止符。

一个完整的BASIC程序应以END语句结束。

1-1-4 程序的注解 为了增强程序的易读性，可以使用注释语句对一段程序或某行程序予以说明或注释。

格式：REM｛字符｝

说明：

一、REM语句作为一个程序行可放在程序的任何地方。

二、在一个多语句的程序行中，REM必须放在此程序行的最后。否则REM后的其他语句将不被计算机执行，会因而产生错误。

1-2 常数、变量和表达式

1-2-1 常数

常数：在程序运行过程中保持不变的数，或直接写入程序的数。BASIC语言中的常数包括数值常数（整常数、实常数）和字符串常数。

一、整常数

整常数:=〔+|-〕｛数字｝

整常数可以是正数或负数，正数时‘+’号省略。数字是不多于五位的0～9之间的任意数字的组合。

数值范围：

1.APPLE II整数BASIC只能处理整型数。APPLE SOFT II则不论整数实数都能处理。其整数范围为：-32767～+32767之间。

2.MBASIC、IBM PC BASIC的取值范围为：-32768～+32767，在内存中占两个字节。整常数可分为十进制、十六进制和八进制三种。

其中

十六进制用前面冠以&H表示，如&HFF表示十六进制数的FF；

八进制数用前面冠以&O或&表示，如&77表示八进制数的77。

3.CROMEMCO BASIC的取值范围同APPLE SOFT。

二、实常数 即实型常数

实常数有下述二种表示方法：

1.小数点表示法即定点表示法　若一个实数的绝对值在0.01到999999999.2范围内，则计算机输出时是用小数点表示法。

2.指数表示法即浮点表示法　当实数的绝对值小于0.01或大于999999999.2时，会以指数形式输出。其格式为

〔+|-〕 n.nnnnnnnnE±e

其中n为有效数字，E表示底数10，e为阶码(指数)，用二位整数表示。数符为正时可以省略，但阶码符号不能省去。如

$2.718281E+13=2.718281\times10^{13}$

实常数的取值范围：

1.APPLE II中，以五个字节存贮一个实常数，但最多输出9个有效数字。SOFT的实数范围是－1＊10∧38～1＊10∧38，超出这个范围是错误的。不过在加法或减法时，所有的数字可以大到1.7×10∧38。若实数的绝对值比3*10∧(－39)还小的话，APPLE II SOFT会认为是0。

2.MBASIC的实数范围在10^{-38}～10^{38}之间，可分为单精度常数和双精度常数。

单精度常数　小于7位数的常数。(只印出6位有效数字)具体形式如下：

(1) 7位或位数更少的数字；如32767

(2) 带有E的指数形式的数；如－7.08E－08

(3) 常数尾带有惊叹号'!'的数。如37.88!

双精度常数

(1) 8位以上的数字；如327676767

(2) 带有D的指数形式的数；如－3.1415926D＋08

(3) 常数尾带有'#'号的数，如1.4142#或7715895#双精度常数最多可达16位有效数字。

3.IBM PC BASIC中的实常数同MBASIC。

4.CROMEMCO BASIC中的实常数：

短浮点数：有6位有效数字。数的范围在＋9.99E＋62～＋9.99E－65之间。

长浮点数：最多可以表示14位有效数字。数的范围同短浮点数。

此外，在CROMEMCO中还定义了十六进制数。

十六进制数：范围在0H到FFFFH之间，一律以%符号开头和结束。如%8642%

三、字符串常数

字符串常数是用'" '号括起来的字符串。其长度以字符个数（包括空格）计算。字符个数为0的字符串叫做空串，表示为""。

APPLE II BASIC、MBASIC、IBM PC BASIC的字符串中，允许的字符个数是0～255(包括空格)。

1-2-2　变量　在程序执行过程中，数值可以改变的量，称为变量。在程序中要给变量取名来表示它。变量名对机器来说标志一个内存单元的地址。

一、格式：变量名：=〈字母〉〔{字母|数字}〕

(一)、APPLE II BASIC（整数和SOFT）规定，变量名最多包含238个字符，但只有前两个有区别意义。

(二)、MBASIC语言中的变量以大写英语字母开头，其后为字母、数字或小数点，最长允

许有40个字符。如P、A1、MA·都是合法的变量名。

(三)、IBM PC BASIC语言变量名格式同APPLE SOFT 变量名，但最长只允许40个字符。

(四)、CROMEMCO BASIC 语言中的变量是用一个字母或一个字母跟一个数字来表示。如A、A1…Z9。最多许可286个变量。

说明：保留字不能包含在变量名字内。

二、同常数一样，变量可分为以下三种：整型变量，实型变量和字符串型变量。

(一)整型变量：

1.APPLE II BASIC整型变量：=〈变量名〉〈%〉

2.IBM PC BASIC整型变量同上，如AS1%就是整型变量。

3.CROMEMCO变量是用语句进行控制的。INTEGER 语句的功能是使被指定的变量成为整型变量；如果在IMODE语句之后执行RUN命令，则程序中的所有变量都取整数方式。

(二)实型变量：=〈字母字符〉|〈字母字符〉〈数字〉。

1.APPLE II BASIC实型变量为定义格式。

2.MBASIC实型变量分为单精度、双精度两种。

(1) 单精度变量：=〈变量〉|〈变量〉〈!〉

(2) 双精度变量：=〈变量〉〈#〉

3.IBM PC BASIC语言实型变量分类及表示法同MBASIC。

4.CROMEMCO BASIC实型变量分为：短浮点变量、长浮点变量。

(1) 用SHORT语句可使指定变量为短浮点变量。若在SFMODE之后执行 RUN 命令则使程序中所有变量都取短浮点形式。

(2) 要使某个变量取长浮点形式，可用LONG语句指定；欲使程序中所有变量取长浮点，只要在使用LFMODE初始语句后再执行RUN命令即可。

(三)字符串变量：=〈变量名〉〈$〉，如MA$。

字符串变量用来存贮字符串数据。串中最多可有255个字符。而CROMEMCO BASIC字符串的长度是用 DIM 语句说明的。如未加说明就只能含11个字符或更少些，如A$ B26$ RR4$为合法的串变量。

三、变量类型的限制语句

在MBASIC和IBM PC BASIC中，使用DEF语句可限定变量的类型是整型 (INT)、字符串型(STR)或是单精度(SNG)、双精度(DBL)。

格式：DEF〈类型〉〈字母范围〉

说明：

1.DEF INT I-N, W-Z

凡是以I、J、K、L、M、N、W、X、Y、Z字母为字首的变量，均为整型变量，但IAA#、JCD$、N!等仍分别为双精度、字符串和单精度变量

2.DEF STR A, S

凡以A或S为首的变量均为字符串变量。

3.DEF DBL D, W-Z

凡以D或W、X、Y、Z为字首的变量，一律视为双精度变量。

4.DEF SNG C, Q

凡是以C、Q为首的变量一律视为单精度变量。

上述变量均被称为简单变量。变量的另一种形式——下标变量，将在1-6-1说明。

1-2-3 数学函数 在BASIC语言中，允许使用以下几种标准函数。

一、APPLE SOFT有以下11种：

1. SIN(X) 求出X弧度的正弦值
2. COS(X) 求出X弧度的余弦值
3. TAN(X) 求出X弧度的正切值
4. ATN(X) 求出X的反正切值的弧度数
5. EXP(X) 求出指数函数e^x
6. LOG(X) 求出X的自然对数lnX
7. SQR(X) 求出X的算术平方根
8. ABS(X) 求出X的绝对值 ABS(X)=|X|
9. SGN(X) 求出X的符号：

若X>0 则SGN(X)=1；若X=0 则SGN(X)=0；若X<0 则SGN(X)=-1。

10. INT(X) 求出不大于X的最大整数，如INT(2.98)=2；INT(-2.322)=-3。
11. RND(X) 产生一个0～1之间的随机数，

若X>0，则每使用一次RND(X)都产生新的随机数；X<0，则产生相同的随机数；X=0，则产生最近刚产生过的随机数。

二、MBASCI除以上函数外，还具有：

FIX(X) 舍去X的小数部分

CDBL(X) 将单精度数值X转换成双精度

CSNG(X) 将X转换成单精度数值

CINT(X) 取X的四舍五入值

三、IBM PC BASIC语言中的函数同MBASIC

四、CROMEMCO BASIC中除1-11函数外，还有：

BINAND(X, Y) 二进制逻辑与

BINOR(X, Y) 二进制逻辑或

BINXOR(X, Y) 二进制逻辑异或

FRA(X) 取X的小数部分

IRN(X) 取0～32767之间的整随机数

MAX(X1, …, Xn) 测定在表达式列中具有最大值的数字表达式

MIN(X1, …, Xn) 测定在表达式列中具有最小值的数字表达式

RANDOMIZE和RND及IRN联用产生不同序列的随机数。

1-2-4 表达式 在BASIC语言中，把符合规定的，用BASIC运算符（包括算术运算符、关系运算符、逻辑运算符）和括号将常数、变量、函数连接而成的式子叫做表达式。其中包括：算术表达式、字符串相加表达式、逻辑表达式及关系表达式。

一、算术表达式：用算术运算符、圆括号按语法连接常数、变量和函数而成的式子。

在MBASIC和IBM PC BASIC中除加(+)、减(-)、乘(*)、除(/)、乘方(∧)外，还具有整除(\)和取模(MOD)两种运算符。

整除(\)：把运算元素四舍五入成-32768到32767之间的整数，再求得商；若商为小数时，则自动舍去小数部分而成为整数。

取模(MOD)：即求余数，在运算前先把运算元素四舍五入化为整数，再求得余数。如

10＼4＝2　求10除以4的商的整数部分；

5＼9＝0　求5除以9的商的整数部分；

25.68＼6.99＝3求26除以7的商的整数部分；

10 MOD 4＝2求10除以4的余数。

说明：

1.乘号不能省，如A*B不能写成AB。

2.只允许用圆括号，可使用多层。不允许用方括号、花括号。

3.运算对象与运算符号写在同一高度上。

4.一个变量、一个常数也可看作表达式。

二、字符串连接表达式：用连接符(+)连接字符串常数、字符串变量、字符串函数（取值为字符串者）所得的式子。

字符串相加运算就是按字符串的先后顺序串联起来。例如：

字符串连接表达式	运算结果
“1985”+“.”+“8”	“1985.8”
“GOOD”+“␣”+“MORNING”	“GOOD MORNING”

单个字符串常数或字符串变量可作字符串连接表达式的简单情况。

三、逻辑表达式：由逻辑运算符连接逻辑值构成的式子。

逻辑值有真值和假值。在BASIC中规定：真值用‘1’表示，假值用‘0’表示。

逻辑运算符有：

OR　逻辑和(或)

AND　逻辑乘(与)

NOT　逻辑非(非)

MBASIC和PC BASIC中还有：

XOR　逻辑异或

IMP　逻辑隐含

EQV　逻辑同或

设A、B为两个逻辑值，利用上述运算符，可得如下的真值表：

	A		1	1	0	0
	B		1	0	1	0
A	OR	B	1	1	1	0
A	AND	B	1	0	0	0
	NOT	A	0	0	1	1
A	XOR	B	0	1	1	0
A	EQV	B	1	0	0	1
A	IMP	B	1	0	1	1

在CORMEMCO BASIC没有后两种运算。

关系式可看作逻辑表达式的一种简单情况。

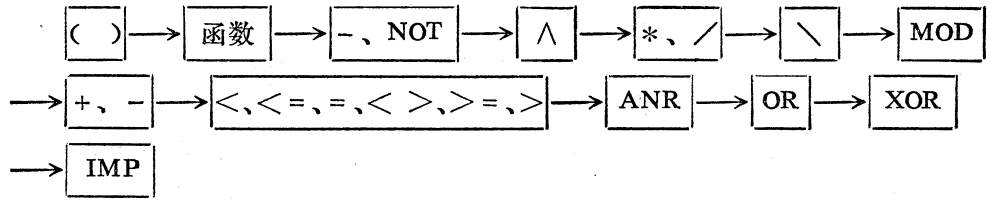
四、关系表达式：用关系运算符连接两个算术表达式（或两个字符串表达式）所成的式子叫做算术关系（或字符串关系）表达式。

关系运算符有六种，其符号和含义如下：

关系运算符	数关系意义	字符串关系意义
<	小于	居前
<=	小于等于	居前或相等
=	等于	相等
<>	不等于	不等
>=	大于等于	居后或等于
>	大于	居后

关系运算符用来比较两边结果值的关系。运算符两边必须是同类数据。如果数据是字符串时，则比较相应的 ASCII代码：依从左到右的顺序，以第一次出现不同的字符来确定运算结果，ASCII码小的居前，若比较结果和运算相符则结果为1，否则结果为0。

五、运算的优先顺序如下式箭号所示：



在同一方框内的优先级相同。

1-3 提供数据的语句

1-3-1 LET(赋值)语句

格式：〔LET〕<变量>=<表达式>

功能：计算机执行此语句时，先计算表达式（算术表达式或字符串连接表达式）的值，再将结果赋值给左边的变量。

说明：

一、算术表达式、逻辑表达式的值允许赋给数值型变量。字符串常量、字符串连接表达式的值只能赋给字符串变量。如LET A$="APPLE"

二、赋值语句的定义符可以省略。

三、表达式中的变量必须是赋过值的，否则变量的初值自动取零值。

四、不允许连续赋值，否则会给出错误信息

? TYPE MISMATCH ERROR

五、同一变量在不同执行时刻可以取不同值，当某个值赋给一个变量后，就“冲”掉了该变量的原有值。

六、赋值号(=)与数学中等号(=)含义不同，如赋值语句 S=S+1 表示把变量S 的现有值加上 1 再赋给S，而数学中S=S+1则不成立。

1-3-2 INPUT（键盘输入）语句

格式：INPUT〔“提示字符串”;〕<变量>〔{，变量}〕

功能：当计算机执行此语句时，屏幕上显示'?'号，并暂停工作，等待输入数据。在有提示字符串时，则屏幕不再出现问号，而显示提示字符串，等待输入数据。此时，计算机得到使用者从键盘输入的数据后，立即把各值传送给相应的变量，并继续运行程序。若输入的数据少于变量的个数，屏幕会显示'??'以表示还需输入数据，使用者应继续输入数据。

说明：INPUT语句接受的数据只能是数字或字符串，不接受表达式，且数据与变量要相符。

各机器中该语句的执行情况：

一、文中之叙述为APPLE SOFT的格式和功能。

二、MBASIC、PC BASIC的执行情况是：无提示符时同上执行，有提示符时，先显示提示字符串，再出现问号(?)。

三、CROMEMCO的执行情况同APPLE SOFT。

四、在CROMEMCO和MBASIC的格式中，提示字符串后面跟逗号(,)。

1-3-3 IN#语句

格式：IN#〈表达式〉

功能：计算机执行此语句时，依表达式的值指定的槽口作输入工作。槽口1到7表示不同的外部设备。

说明：

一、IN#0表示要从键盘输入，而不是从其他外部设备输入。

二、若指定的槽口没有相应的外部设备，则整个系统停下来，打入 RETURN 、RESET 或 CTRL C 才能返回。

三、若表达式的值小于0或大于255，会出现错误信息

? ILLEGAL QUANTITY ERROR

四、若表达式的值在8～255之间，则APPLE SOFT将有意想不到的改变。

五、此语句仅为APPLE SOFT所有。

1-3-4 READ和DATA语句

格式：READ 〈变量〉〔{，变量}〕

DATA 〈常量〉〔{，常量}〕

功能：DATA为非执行语句，它给程序形成一个数据区；READ从数据区中按顺序给变量取数据。可以认为数据区有一个指针，初始位置在第一个数据前，当READ语句读完第一数据时，指针就移到第二个数据前，读完第二个时，指针又移一位，依此执行。如

```
10 READ N, A$, C, D$
20 DATA 3, BAT, 9, MAY
```

说明：

一、READ与DATA两个语句必须联合使用，缺DATA语句时，会显示错误信息

? OUT OF DATA ERROR

二、DATA语句中的数据只能是常数，且个数、类型与顺序必须与READ语句的变量个数、类型及顺序完全一致。DATA语句可放在任何地方。

三、DATA语句中的字符串常数无前导空格（␣）或冒号（:），可不用双引号(" ")引括。

四、DATA语句中的常数个数不限，但总的字符个数不能超过一个程序行。

1-3-5 RESTORE（恢复数据区）语句

格式：RESTORE

功能：此语句能把数据指针重新移到数据区的起初位置，使得它后面的READ语句又从数据区中第一个数据读起。

说明：若使READ语句从特定位置读数，各种BASIC有各自的办法。

一、APPLE SOFT用虚设变量方法，使READ跳过虚设变量对应的数据，再往后读。如

```
10  READ  X1, X2, X3, X4, X5$
20  RESTORE
30  READ  Y1, Y2, Y3, Y4, Y5$
 ⋮
60  DATA  5, 4, 3, 2, ONE
```

若要把5～ONE输入给X1～X5$中，3～ONE输入给Y3～Y5$中，则行30的语句应为：

```
 ⋮
30  READ  T, T, Y3, Y4, Y5$
 ⋮
```

T就是虚设变量，其作用是使后面的变量跳过一个或几个数据。

二、CROMEMCO和MBASIC是在RESTORE后加行号。则下一个READ语句从指定行号的DATA语句中的第一个数据读起。如

```
10  READ  A, B, C, D
20  RESTORE  60
30  READ  W, X, Y, Z
40  DATA  1, 2, 3, 4
50  DATA  5, 6, 7, 8
60  DATA  9, 10, 11, 12
 ⋮
```

程序段中20行语句RESTORE 60使它后面的READ语句跳过50行，从60行的第一个数据开始读数。

在提供数值的语句里，MBASIC和IBM PC BASIC还有SWAP语句。

格式：SWAP 〈变量1〉,〈变量2〉

功能：语句执行后，会把变量2的原有值赋给变量1作为新值；变量1的原来的值赋给变量2，但两个变量类型必须相同。例如

```
10  A$ = "ONE␣"  :  B$ = "ALL␣"  :  C$ = "FOR"
20  PRINT  A$; C$; B$
30  SWAP  A$, B$
40  PRINT  A$; C$; B$
RUN
ONE  FOR  ALL
ALL  FOR  ONE
```

1-4 输出语句和输出格式

1-4-1 PRINT(输出)语句

格式：PRINT〔表达式〔{，|；〔表达式〕}〕〔，|；〕〕

功能：执行此语句时，计算机先计算表达式的值，并将结果输出在终端(屏幕或打印机)上。

说明：

一、表达式可以是常量、变量、函数或者表达式。

二、PRINT 可用问号(？)代替。

三、PRINT后面没有表达式时输出一个空行。

四、表达式之间用分号分隔时，输出项之间无空隙；表达式之间用逗号隔开时，就按打印区输出。APPLE II以16个字符为一区，IBM PC和CROMEMCO分别以16、20个字符为一区。

五、若表达式后面没有逗号或分号时，则印完最后一项自动换行。若语句最后以分隔符终止，则表示此行输出没有结束，下一个语句将在该行按要求继续输出。

六、字符串数据按照原样输出。

1-4-2 PR#语句

格式：PR#‹表达式›

功能：PR# 语句可将输出送至表达式的值指定的槽口。表达式的值必须在 0～7 之间。

说明：PR#0表示将输出送到屏幕。其他说明同IN#。此语句仅为APPLE II所有。

1-4-3 MBASIC 及PC BASIC的输出语句

一、MBASIC、PCBASIC(IBM PC BASIC)的 LPRINT语句

格式和功能同PRINT，只是将结果由打印机输出。

二、WRITE语句

格式：WRITE〔印出项表达式〕

功能：WRITE 同 PRINT，只是格式有所不同：

1．无打印项时，输出一空行；

2．打印项许可为数字或字串表达式，且必须以逗号分开。

当输出项被WRITE语句打印时，有下列情况：

(1) 每两个印出项之间自动印出一个逗号；

(2) 字符串被印出时，会自动冠以双引号；

(3) 每印完一串打印项后，自动换行；

(4) 输出项为数值时，同PRINT。

1-4-4 格式输出 为了使输出格式化，本文所述BASIC都具此功能。

一、APPLE II SOFT 的格式输出功能

(一) APPLE SOFT 的数据输出格式符

1．整数格式：In

I 表示以整数输出，n 表示输出位数。

2．定点数格式：Fn·m

F 表示按小数点形式的实数输出，n 表示全部数字位数，m表示小数点后的位数。如：

F6.3表示输出6位实数，保留小数点后三位。

3．指数格式：En

E表示实数用指数形式输出，n表示E之前的数位（不包括小数点）。

说明：以上三种格式中1≤n≤12，0≤m≤n

（二）APPLE SOFT的格式打印子程序

1．APPLE SOFT的格式打印功能由专用子程序完成。子程序首址与打印机接口板所在槽号有关，因此在使用格式打印子程序时，首先要把子程序首址送入变量WRITE单元中去。

LET WRITE＝49312＋256*SLOT

其中SLOT是槽号，标准连接打印机1*槽。

因此 WRITE＝49312＋256*1＝49568 或WRITE＝52480

2．调用格式打印子程序的格式：

CALL WRITE：〈打印项〉；〈格式符〉[，{打印项；格式符}][，CHR$(13)]：

打印项与格式符之间用‘；’分隔。多于一种打印项和格式符时，每组之间用‘，’分隔。一个调用语句打印完一行所有内容后给回车命令，命令与打印项用‘，’号隔开，即[，CHR$(13)]。若一个调用尚未完成完一行的所有内容，则用‘；’结束语句，而不给回车命令。下一个调用语句将在该行续继输出。例如

10 CALL WRITE：X%；I4：

表示整型变量X按四位输出。

说明：

(1) 当输出字符串常数时，可不用格式符而照原样输出；输出字符串变量同字符串常量；

(2) 若变量没有设置输出格式，则按其前面的格式打印；

(3) 调用格式打印子程序前，应设置PRINT CHR$(0)语句。

二、MBASIC 中的 PRINT USING 语句

格式：PRINT USING“格式字符串”；〈印出项〉

功能：将字串或数值以特定格式输出。

说明：

(一)格式字符串可为字符串常数、字符串变量，用来指定印出项的格式。

(二)印出项可为字符串或数，若印出项多于一个时，各项之间用分号(；)分开。

(三)印出格式对字符串和数值的规定如下：

1．字符串格式：

(1)“！”只印出该字串的第一个字母。

(2)“＼空n格＼”印出该字串的前 n＋2 个字母，有三种情况：

① 若两斜线“＼”间不留空格，则印出前两个字母。

② 若 n＋2 小于字符串长度，则输出前n＋2个字符，其余被忽略。

③ 若n＋2 大于字符串长度，则字符串靠左边输出，以右边留空格补充不足字数。

(3)“&”将字符串全部印出。其长度随字符串而改变。

2．数值格式：

(1)“#”数目符号，用来表示输出的位数。

“．”小数点符号，用于指定输出数值中小数点的位置。超出的位数，小数点以后的四舍五入。

“，”逗号，会在数值中标出“，”号。

“%”输出数据长度超出格式所定义位数时，在数值前面加印%号。但不许超过24位数。

例. 10 LET A=1985
20 LET B=43.765
30 LET C#=1 2 3 4 5 6 7 8 9#
40 PRLNT USING “THEN END##·##”; A
50 PRLNT USING “THEN END##·##”; B
60 PRINT USING “###, ###·##”; C#
OK
RUN
THEN END%1985.00
THEN END43.77
123,456,789.00
OK

(2) ‘+’号可写在输出格式‘#’号前或后，标明印出数值的正负。

‘-’号可放在输出格式‘#’号前或后。在前，则表示无论输出数值的正负，全冠以‘-’号，表示负数；在‘#’后，则输出数为负时，才加以标明。

(3) ‘**’将输出数值的不足数位，用*号补充。

(4) ‘$$’表示在数据前加上一个$。

说明：指数格式不能与$$一起使用，负号在格式前标出的也不能与$$一起使用。

(5) ‘**$’具有‘**’和‘$$’双重作用。

(6) ‘,’号的另外三种用法

① 若‘,’号在小数点前，则会自动在输出数的小数点前每三位加一个逗号。

② 若‘,’在定义数值格式之后，则被视为字符串的一部分，一起列出。

③ 在指数格式(^^^^)逗号将失效。

(7) ‘^^^^’为指数式，其指数为二位正数，输出形式为E+XX。若不另规定输出数的正负格式，则正数前面留空格，负数则印出负号。

(8) ‘-’使介于该符号之间的字符以文字、数字混合输出，若欲输出‘-’，则需定义两个‘-’。

三、IBM PC BASIC的PRINT USING语句

(一)PC BASIC与MBASIC的PRINT USING语句的用法基本相同，但格式稍有差别。

1.在多个印出项之间用逗号分开；

2.‘-’号同‘+’号的用法；

3.在打印格式中可使用一个$号，输出时，$号在最左边，而后数值则按‘#’号输出，称为固定的‘$’符号。如

10 PRINT USING “$####.##”; 34.78 的结果为：$␣␣34.78

(二)以上格式输出用LPRINT USING语句，结果可由打印机给出。

四、CROMEMCO BASIC的PRINT USING语句

格式：PRINT USING “表达式”, ‹X1, X2…, Xn›

功能：PRINT USING允许用户为打印输出指定格式。表达式为串常数或串变量格式定义，它可以包括一个以上的格式场和字符串，格式场的每一边以非特殊的字符(包括 # & * + , $ - .)为界。

说明：

1.表达式中包括文字符时，文字串的字符遵守任一专用格式场符号而在适当的位置上被打印。

2.如输出字符串中的字符数少于格式场中的符号数，则用空格填满不足位。如：

执行PRINT USING "****, **.*", "STRING" 的结果是STRING␣␣␣

如果输出项中的字符数多于格式场中的符号数，则多余的字符被删去。如

10 PRINT USING "&&&, &&&.&&", "STRING—LITERAL" 的输出结果为：STRING—LIT

3.如果输出项的数目多于拟定的格式场的数目，则拟定的格式场将被超出项重复使用。

例：10 PRINT USING "$$##␣␣$$&&.&", A,B,C的结果是A和C采用格式场的'$$##'格式，而B将用格式场的'$$&&.&'格式。

4.格式表达式中最多可有128个字符，

5.输出数据的数位多于格式定的位数时，会产生一个全星的出错信息。

6.输出数值时，下述的特殊符号可用来格式化数字：

"#" 用来表示印出数的位数：

"&" 不足位数在前边补0：

"*" 不足位数前面以*充满。

7. "," 号作用是把一个逗号放在格式格中逗号出现的位置上。

8. "." 用法同MBASIC。

9. 固定的正号和负号：若"+"或"-"在格式第一个符号位，则输出时将在说明符的位置上打印出输出项各自的符号。

10. 浮动的正负号（++）或（--）在格式场的开始使用：两个或两个以上的正号或负号时，将输出数各自的符号直接放在表达式数值前面。如果一个正的表达式用浮动的负号格式打印，则数前面的符号位上直接用空格来代替。浮动格式中的其他符号可用来代替数字。

11.固定的 "$" 号同PC BASIC的用法。

12.浮动的$符号"$$"同M BASIC 用法。但在"$$"前边允许有(-)号。

13.指数格式 "!!!!"用法同MBASIC的"^^^^"。

1-4-5 PRINT与TAB(X)函数

格式：PRINT{TAB(X)；〈打印项〉}

功能：TAB函数使输出项在规定的列开始。

说明：

一、TAB必须与PRINT连用；自变量X必须用括号括起来；X可为整数、变量或表达式，其值应介于1～255之间。

二、TAB函数只能使光标从1开始向右移动，所以X的值应越来越大。TAB(0)的显示位置是256列。

三、当表达式的值大于40而屏幕只有40列时，输出会自动换行。IBM PC 中每行为40或

80个字符。

四、若要显示TAB(X)的所在的列的位置，应加入：

```
PRINT  TAB(X); POS(0)+1
```

屏幕便显示X值所在的位置。POS 及SPC的位置从0算起。

1-4-6 PRINT 与SPC(X)函数

格式：PRINT{SPC(X)；〈打印项〉}

功能：SPC 表示一个长度为X的空白字符串，使光标由原来的位置向右移动 X格。SPC(0)表示没有空格。

说明：必须与PRINT连用。

1-5 控制语句

要改变、暂停或终止程序的运行，可采用如下控制语句。

1-5-1 GOTO(无条件转向)语句

格式：GOTO 〈行号〉

功能：使程序无条件地转到以语句指定的行号为启始的程序段去执行。

说明：

1. 语句无任何条件就可以使程序转到指定的行，非常活跃；
2. 在结构化程序中要慎重使用，以保证模块有单一出口。

1-5-2 IF…THEN(条件转移)语句

一、APPLE SOFT 的IF…THEN语句

格式：IF〈表达式〉THEN〈语句〉〔{:语句}〕

IF〈表达式〉THEN 〔GOTO〕〈行号〉

IF〈表达式〉〔THEN〕GOTO〈行号〉

功能：当表达式为逻辑表达式且其值为真时，执行 THET 后面的语句，反之，即逻辑表达式的值为假时，执行下一个后继语句。

当表达式为算术表达式且其值非零时，执行THEN后面的语句，其值为零时，则执行 IF 语句的后继语句。如

```
10  IF  X*Y< >0  THEN  160
10  IF  X*Y< >0 GOTO  160
10  IF  X*Y< > 0 THEN GOTO 160
10  IF  X*Y  THEN 160
```

说明：

1. 如果有“表达式< >0” 形式，则条件语句中的“< >0” 可以省略。
2. 在BASIC中，只有IF…THEN语句可用比较符。
3. 如果两个字符比较次序，则以ASCII 码为准。
4. THEN 之前有字母A时，会造成语法错误，出现错误信息 ? SYNTAX ERROR
5. IF…THEN可以按格式嵌套使用。

二 CROMEMCO BASIC 的IF 语句同上。

三 MBASIC 和 IBM PC BASIC 的条件语句

格式：IF〈表达式〉THEN〈语句〉|〈行号〉 ELSE〈语句〉|〈行号〉

IF〈表达式〉 GOTO〈行号〉 ELSE〈语句〉|〈行号〉

功能：以上格式可简写为IF A THEN C ELSE D,即：若A成立,则执行C,否则执行D。C、D可以是语句或行号。例如：

```
10  IF A>0 THEN IF B<0 THEN PRINT
    "A>0, B<0" ELSE PRINT "A>0, B=>0"
```

1-5-3 FOR…NEXT(循环)语句

微型机的功能之一是它能不断重复做某些事情。例如要计算S=1+2+3+…+99,用FOR…NEXT 语句就非常方便。

格式：FOR〈循环变量〉=〈算术表达式1〉 TO〈算术表达式2〉〔STEP〈算术表达式3〉〕
〈循环体〉
NEXT〔〈循环变量〉〕

功能：语句使以FOR…TO…STEP开始的而 NEXT 为返回点的一段程序多次重复执行。具体过程是：首先执行 FOR…TO…STEP 语句，求出作为初值、终值、增量值的三个对应的表达式的值，在执行循环体的过程中不再改变；

2.把循环初值赋给循环变量；

3.执行一次循环体；

4.遇到 NEXT语句时，把循环变量的当前值加上循环增量(或步长)值，并判断循环变量是否超过终值(增量为正时，指大于终值；增量为负时，指小于终值)。若超过终值，则结束循环，执行NEXT之后的语句，否则执行循环体。

例。
```
10  LET  S=0
20  FOR  N=1  TO  100  STEP  2
30  LET  S=S+N
40  NEXT  N
50  PRINT  "1+3+5+…+99=";S
60  END
```

说明：

1. FOR和NEXT中，循环变量必须是一致的实型变量；FOR、NEXT一定要配合使用；

2. 循环增量不许为零；

3. 初值、终值，增量值可为正、负及小数；

4. 当循环增量为小数时，应在循环终值上加1/2。因小数由十进制转换到二进制时，经常取近似值，经过若干次循环后，由于误差积累会造成丢失一次循环。

在MBASIC和 IBM PC BASIC 中还有一种形式的循环：WHILE WEND 循环，它类似于前面的FOR…NEXT循环

格式：WHILE〈表达式〉
⋮
WEND

说明：当WHILE 之后的表达式为真时，执行WHILE与WEND之间的语句；遇到WEND语句即重新判断WHILE语句后的表达式，若为真再执行一次 WHILE与WEND之间的语句；依此类推，直到表达式为假时,才跳出循环,执行WEND的后继语句；WHILE与 WEND 必须

成对出现。

例。
```
10  LET  S=0
20  LET  N=2
30  WHILE  N<=100
40  LET  S=S+N
50  LET  N=N+2
60  WEND
70  PRINT  "2+4+6+…+100=";S
80  END
RUN
2+4+6+…+100=2550
```

1-5-4 GOSUB和RETURN 语句

格式：GOSUB〈行号〉　（转子语句，在主程序中）
　　RETURN　（返回语句，用在子程序中）

功能：

一、GOSUB 语句把程序转移到以指定的行号为开始（入口）的子程序去执行。当遇到RETURN语句时，跳回使程序转入此子程序的GOSUB语句的下一个语句，继续运行；

二、RETURN是子程序的结束语句，使程序返回调用点。

说明：

一、子程序中只少有一个RETURN语句；

二、如果程序执行到RETURN语句的次数多于执行GOSUB的次数时，会出现错误信息：

? RETURN WITHOUT GOSUB ERROR

1-5-5 DEF 语句

格式：DEF〈函数名〉(虚拟变量)=〈表达式〉

功能：DEF语句定义一个程序中可引用的简单函数，这个函数名必须以FN 开头，后面跟简单变量名。

说明：

1.定义后的函数可在任何语句中引用，但括号中的虚拟变量必须换成已赋值的实际变量、常数或表达式。

2. 自定义函数必须在一个语句内写完。

3.表达式允许是虚拟变量、常量、变量、函数或有定义的自定义函数。

例。
```
10  LET X=3
20  DEF FNA(Z)=Z∧2-Z
30  PRINT  X+FNA(2*X)
40  END
RUN
33
```

4. MBASIC 和 PC BASIC 允许有数个虚拟变量；也可以定义字符串函数，但必须以 $ 结尾。如

```
10  DEF  FNA(X,Y,Z)=X∧2+Y∧2+Z∧2
```

```
10  DEF  FND$(A$,J)=LEFT$(A$, J)
```

1-5-6 ON…GOTO(开关)语句

格式：ON〈表达式〉GOTO〈行号〉{〔，行号〕}

功能：执行此语句时，首先计算表达式的值并取整，再依值的大小转到后面相应的行号开始的程序段去。

说明：

一、算术表达式的值必须在0～255之间，超出范围则屏幕出现错误信息

? ILLEGAL QUANTITY ERROR 并停机。

二、若算术表达式的值等于零或大于行号个数，本语句被忽略，程序继续往下执行。

三、使用此语句的关键在于建立一个适当的表达式。也可用变量。

四、MBASIC的执行方式是先计算表达式的值再四舍五入，然后依值的大小作流程转移工作。

例如：20 ON X+1 GOTO 70, 60, 60, 80

即 当 INT(X+1+0.5)=1 时转至70行执行；

INT(X+1+0.5)=2 时转至60行去；

INT(x+1+0.5)=3时转至60行去；

INT(X+1+0.5)=4 时转至80行执行。

1-5-7 ON…GOSUB(计算转子)语句

格式：ON〈算术表达式〉GOSUB〈行号〉{〔，行号〕}

功能：类似开关语句(ON GOTO)，但依表达式选择转入的是相应行号开始的子程序。

1-5-8 ONERR GOTO（错误转移）语句

一、APPLE SOFT 中

格式：ONERR GOTO 〈行号〉

功能：在本语句以后的程序发生错误时，程序转到指定行号开始的“处理错误”子程序，并把错误代码(见本章附录1) 放入地址为222的内存单元里。

说明：

1.可用命令PRINT PEEK (222) 使屏幕显示错误信息；

2.ONERR语句要用在错误发生之前，所以通常放在程序的开始；

3.必须与RESUME联合使用。

二、MBASIC中的格式为：

```
ON  ERROR  GOTO  N
```

功能：“错误处理”子程序发现错误种类后，可用ON ERROR GOTO 0(行号为0)结束执行，并显示错误信息；也可以依照指定行号继续执行。

三、CROMEMCO BASIC中的格式：

```
ON  ERROR  STOP
ON  ERROR  GOTO  N
ON  ERROR  GOSUB  N
```

功能：ON ERROR命令在程序发生非致命性错误时，使程序控制按所说的那样转移。

1-5-9 RESUME 语句

一、APPLE II SOFT中的RESUME语句

格式：RESUME

功能：使程序返回原来发生错误的语句去执行。

说明：

1.RESUME要用在错误发生之后的语句中，若用在这之前，会造成：

? SYNTAX ERROR N 65278或使程序中断；

2.若在"错误处理"子程序中发生错误，使用RESUME会使程序无限循环；这时可用 RESET、CTRL-C、RETURN使之回到APPLE SOFT。

二、MBASIC的RESUME语句

格式1. RESUME或RESUME 0

功能：该语句在"处理错误"子程序完成工作以后，指定程序继续执行的语句。具体如下：

格式1使程序回到转入"处理错误"子程序的语句去执行；

格式2：RESUME NEXT

转入"错误处理"子程序语句的下一个语句继续执行；

格式3：RESUME N 跳到行号N的语句去。

说明：RESUME只能用在"错误处理"子程序中。

三、CROMEMCO的ON ESC语句

格式：ON ESC STOP

ON ESC N

ON ESC GOSUB N

功能：在程序执行中按ESC键，ON ESC语句使程序控制按说明的那样转移。

说明：若ON ESC写在程序开始，则每按一次ESC键，用ON ESC说明的命令就能被执行一次。否则，只有在ON ESC语句执行后，按ESC键，ON ESC才能被执行。

1-5-10 STOP(暂停)语句

格式：STOP

功能：使程序中断运行，屏幕上显示信息：

BREAK IN 行号

行号表示执行STOP语句的行号。在程序中断运行以后，保留所有的断点，当给继续运行命令后，程序会从暂停语句后面下一个语句（不是下一行）继续运行。在没有破坏源程序的情况下，变量值都保持其正确性。

用途：用于调试程序。程序暂停后可以用PRINT语句把变量显示出来，以了解其正确性。

1-5-11 END(结束)语句

格式：END

功能：END结束程序的执行，准备接受新的命令。

1-6 数组

1-6-1 数组和下标变量

一、数组：

1.定义：数组是按一定顺序排列的一组变量，即变量的有序集合。

2.数组以一个BASIC变量名，作为它的名字。

3.数组的体积是指一个数组中元素的个数，它标明了数组的大小。

4.数组必须由同类元素组成，根据数组元素的类型，数组可分为整型数组、实型数组、字符串型数组。

二、数组里的单个元素，它们各自处于数组中的某一位置，即它们各自带有对应的下标，所以称它为下标变量。

下标变量用它所在的数组的名字后缀类型说明符，再加上一个含有对应下标的圆括号表示。下标规定从0开始。如

A%(3)是整型数组A中第4个下标变量；

B$(25，3)是字符串型数组B　中第26行第4列的下标变量。

1-6-2　DIM(数组说明)语句

格式：DIM〈数组名〉(下标表达式表)〔{，数组名(下标表达式表)}〕

功能：说明每个数组的名字、类型、维数及体积。

说明：

一、下标表达式表的形式如下：

(算术表达式1，算术表达式2，…，算术表达式n)，其中，算术表达式的个数是数组的维数。APPLE SOFT中n≤88而MBASIC及IBM PC BASIC中n≤255。

二、下标以0为下界，算术表达式n的值取整为上界。因此数组的体积=(算术表达式1的值取整+1)×(算术表达式2的值取整+1)×…×(算术表达式n的值取整+1)

例如：DIM　A%(20)，B(49，99)，C$(100)说明了三个数组，第一个是有21个下标变量的整型数组A；第二个是二维的实型数组B，它有50行、100列，共有50×100=5000个下标变量；最后一个是一维的字符串型数组C，它有101个元素。

三、标志数组维数的表达式都是常数时，叫做固定数组，表达式不全为常数时叫可调数组。

例如：DIM　D(M，L，N)

说明了一个名字为D的实型数组，有M+1行，L+1列，N+1页(层)。

四、当下标不超过10时，可省略DIM语句。

五、MBASIC 和IBM PC中，允许下标的最大值为32767。可以使用OPTION　BASE语句指定下标的最小值，其格式为

```
OPTION　BASE　n
```

其中n之值仅为0或1，使用该语句时，应放在DIM语句之前。

六、在CROMEMCO扩展BASIC中，下标最大值为16382。字符串变量的长度也用DIM语句定义。如：

```
10　DIM　A$(20)
```

说明了字符串变量A$的长度为21个字符。

DIM(数组说明)语句的应用，如：

```
10　DIM　A(3)
20　FOR　I=1 TO 3
30　READ　A(I)
40　PRINT　"␣␣␣A";I;"=";　A(I);
```

```
50  NEXT  I
60  DATA  10, 20, 70
RUN
␣␣␣ A1=10␣␣␣A2=20␣␣␣A3=70
```

1-7 字符串处理

1-7-1 LEN函数

格式：LEN(字符串表达式)

功能：测试字符串表达式的结果字符串的字符个数，即字符串长度。

例
```
10  A$="␣"  :  B$="TOM"
20  PRINT  A$, B$
30  PRINT  LEN(A$), LEN(B$)
RUN
                TOM
1                3
```

A$被赋值一个空格字符，所以得到的字符串长度为1。

说明：被测试长度的字符串的字符个数（包括空格字符）必须在0～255之间，若大于255会给出错误信息

```
? STRING  TOO  LONG  ERROR
```

1-7-2 LEFT$函数

格式：LEFT$(字符串表达式，N)

功能：函数从字符串表达式的结果字符串的最左边摘取N个字符组成子字符串。

若N大于或等于字符串的长度，则得到整个字符串。

1-7-3 RIGHT$函数

格式：RIGHT$(字符串表达式，N)

功能：函数从字符串表达式的结果字符串的最右边摘取子字符串，子字符串包含N个字符。

说明：

一、若N<1或N>255，则屏幕会给出错误信息

```
? ILLEGAL  QUANTITY  ERROR
```

二、若N>LEN(字符串表达式)，则RIGHT$摘取整个字符串。

1-7-4 MID$函数

格式：MID$(字符串表达式，M，N)

功能：函数从字符串表达式的结果字符串左起第M个字符开始，摘取N个字符，组成子字符串。

例
```
10  L$="ABCDEFG"
20  PRINT  MID$(L$, 3, 2)
```

的运行结果为CD

说明：

一、如果M>LEN(字符串表达式)则MID$为空串，

二、如果M或N超出1～255的范围，则会显示错误信息

? ILLEGAL QUANTITY ERROR

三、M＋N≥255时，多出的字符串长度值将被忽略。

1-7-5 STR$函数

格式：STR$(算术表达式)

功能：函数STR$能实现将括号内算术表达式的结果从数字转换成数字字符串。

例
```
10 LET A=531.28 : LET B=-531.28
20 PRINT STR$(A), STR$(B)
30 PRINT STR$(A+B), STR$(A)+STR$(B)
RUN
␣531.28          -531.28
0                ␣531.28-531.28
```

说明：

一、␣是符号的位置。

二、在30行第一个打印项是先做算术运算后转换；第二个打印项是先转换后作字符串连接。531.28和－531.28都是字符串，所以不能再相加，只能用字符串连接。当算术表达式的值超出实数范围时，将出现错误信息

? OVERFLOW ERROR

1-7-6 VAL函数

格式：VAL(字符串表达式)

功能：函数VAL能把数字字符串转换成一个实数或整数。在执行中从左边起，逐个字符检查看是否是可接受的数字字符（允许是空白、小数点、＋、－、E和数字）。直到出现非数字字符为止，将可接受的数字字符转换成数值，自第一个非数字字符以后的字符串全部忽略。

例
```
10 A$="42"
20 PRINT VAL(A$)
30 PRINT VAL(A$+"."+A$)
```

其输出结果为：42

42.42

如果作为自变量的字符串是由数字、字母等混合组成，那么VAL函数将依据字符串排列在前面的数字而确定结果。例如：

VAL("12"+"E"+"12")的结果为1.2E+13

VAL("100EEOC") 的结果为100

VAL("A400E5") 的结果为0

说明：

一、作为VAL的参数的字符串，若长度超过255个字符，会得到错误信息

? STRING TOO LONG ERROR

二、如果函数的绝对值大于1E＋38或数字包含了38位数字以上的数字，则计算机给出错误信息 ? OVERFLOW ERROR

1-7-7 CHR$函数

格式：CHR$(算术表达式)

功能：CHR $ 函数给出与算术表达式的值相对应的ASCII字符。

说明：算术表达式的值必须在0～255之间，否则给出错误信息

? ILLEGAL QUANTITY ERROR

例
```
10 A$ = CHR$(34)
20 PRINT "HE SAID:"A$; "HELLO"; A$
RUN
HE SAID: "HELLO"
```

注：引号做为字符串的专用符号，若直接书写于程序中，则被认为是字符串的定界符，不能作为字符串的内容；若使用CHR $(34)则执行PRINT语句时，会打印出'"'号。

1-7-8 ASC函数

格式：ASC(字符串表达式)

功能：这个函数给出字符串表达式的第一个字符的ASCII十进制代码（不一定是最低数字），在96～255范围内的ASCII代码产生的APPLE上的字符与0～95范围内的 ASCII 代码产生的字符重复。

说明：

一、虽然CHR $(65)的结果是A，CHR $(193)的结果也是A，但当使用逻辑运算符比较时APPLE SOFT不认为二者相等。

二、参数是字符串时，必须括在引号中。

三、参数为空串时，给出错误信息：

? ILLEGAL QUANTITY ERROR

1-7-9 MBASIC、PC BASIC所具有的函数

一、STRING $ 函数

功能：函数能产生某一长度同一个字符的字符串。

格式1：STRING $(I, J)

产生长度为I，字符的ASCII码皆为J的字符串。

格式2：STRING $(I, X $)

产生长度为I，字符皆为X $ 中第一个字符的字串。

例
```
10 A$ = STRING$(6, 42) : B$ = STRING$(6, "*")
20 PRINT "A$ = "; A$
30 PRINT "B$ = "; B$
RUN
A$ = ******
B$ = ******
```

二、INSTR函数

格式：INSTR(Y, A $, B $)

功能：函数从A $ 字串(母字串)中的第Y个字符开始，向后寻找与B $ 字串（子字串）相同的部分字符串，以首次出现B $ 字符串的位置为其函数值。Y的范围在0～255之间。

说明：

1.若B $ 字符不在A $ 字符串中，则其函数值为0。

2.若B $ 字符串为空字符串，则其值为1或为Y(若Y不省略)。

3.若Y>LEN(A$)或A$为空串，则其值为0。

4.Y可省略，省略时将自第一个字符找起。

例
```
10  A$="1234567890" : B$="456" : C$=""
20  PRINT "INSTR(3, A$, B$)="; INSTR(3, A$, B$)
30  PRINT "INSTR(5, A$, B$)="; INSTR(5, A$, B$)
40  PRINT "INSTR(1, A$, C$)="; INSTR(1, A$, C$)
RUN
INSTR(3, A$, B$)=4
INSTR(5, A$, B$)=0
INSTR(1, A$, C$)=1
```

三、SPACE$函数

格式：SPACE$(X)

其中，X可为数值、变量或表达式。

功能：运行时，先将X四舍五入为整数(允许0≤X≤255)，后给出长度为X的空白字符串。

说明：这个函数与SPC(X)的区别是：SPC(X)只能与PRINT或LPRINT联合使用，此函数则要单独使用。

四、OCT$/HEX$函数(IBM PC无此函数)

格式：OCT$(X)

HEX$(X)

功能：OCT$(X)函数将十进制数X转换为八进制数，并以字串表示；HEX$(X)函数将十进制数X转换成十六进制数，并以字符串表示。

执行时先将X四舍五入为整数，再进行转换。

例
```
10  PRINT "I          OCT$      HEX$"
20  PRINT STRING(16, "=")
30  FOR I=30 TO 32
40  PRINT I; TAB(7); OCT$(I); TAB(15); HEX$(I)
50  NEXT I
60  END
RUN
 I          OCT$       HEX$
= = = = = = = = = = = = = = = =
30          36          1E
31          37          1F
32          40          20
```

五、INPUT$函数

格式：INPUT$(字符串长度[,#文件号码])

功能：INPUT$自终端或文件中读一个某长度的字符串数据。例：

```
10  X$=INPUT$(1)
```

自终端机键盘读取一个字符，并指定为X$的内容。当执行此语句时，便停止运行，等

待打入任一个字符。得到某个字符后，不等使用者按RETURN键，就立即往下执行。又如

30 X$=INPUT$(4，#5)

自5号文件中每次读取4个字符，并指定为X$之内容。

六、INKEY$函数

格式：〈字符串变量〉=INKEY$

功能：用函数INKEY$自键盘读取数据。当BASIC解释程序执行INKEY$语句时，便开始检查键盘是否有数据输入。若此时打入一字符，则该字符立即成为字符串变量内容；若无字符打入，则字串变量成为空字串。

说明：1-6-1——1-6-8为APPLE SOFT、MBASIC、IBM PCBASIC所公有，CROMEMCO BASIC缺少LEFT$、RIGHT$和MID$三个函数。其功能由字符串的引用实现

1-7-10 字符串的引用和POS语句

一、在CROMEMCO中的字符串变量的引用有下面四种格式：

格式1：＜字符串变量＞

功能：在输入时(INPUT或GET)，如果引用一个不带下标的字符串变量，则整个字符串被引用；在输出时（PRINT或PUT)，如果引用一个不带下标的字符串，则引用一个从0号字符到最后一个有效字符的字符串。

格式2：＜字符串变量＞(表达式1)

功能：格式2是一个带下标的字符串，定义了一个子字符串：从表达式1指出的那个字符开始直到第n个字符(n是在输入时被定义的字符串长度)，或者到最后一个有效字符。这时要求：0＜表达式1≤定义值

如果表达式1小于或等于0，则引用整个字符串。

格式3：＜字符串变量＞(表达式1，表达式2) 表达式2≥0

功能：格式3定义一个有两个下标的子字符串，且第二个下标是正的。引用的字符串是从表达式1指定的位置的字符开始到表达式2指定位置的字符结束。这时要求：

0≤表达式1＜表达式2＜DIM(定义值)

格式4：＜字符串变量＞(表达式1，表达式2) 表达式2＜0

功能：若用两个下标（表达式1，表达式2)，第二个下标是负值，则所引用的子字符串从表达式1指出的字符开始，以表达式2的绝对值作为其长度。要求：

0≤表达式1＜定义值

表达式1＋|表达式2|≤定义值。

说明：表达式是算术表达式、变量或整数。如果字符串变量未被说明，则它被认为长度是10。

二、POS函数

格式：POS(字符串表达式1，字符串表达式2，表达式)

功能：用POS为字符串(X$)（表达式1）的子字符串(Y$)（表达式2）定位。在字符串X$中开始寻找的位置由算术表达式说明。则函数值等于在字符串的第一字符的位置。

1-8 其他功能的函数和语句

1-8-1 POS函数

一、APPLE SOFT的POS函数已在1-4-5叙述。

二、MBASIC的POS(X)/LPOS(X)函数

POS(X)与LPOS(X)都是位置函数，参数X为一个虚拟参数，可以是数字、字串或变量名称。

格式及功能：

POS(X)用来显示终端上光标目前所在的位置，最左边为1。

LPOS(X)用来获悉打印机缓冲区中打印机头所在位置。

三、PC BASIC的POS(X)/LPOS(X)的功能同MBASIC。其格式为：

POS(n)

LPOS(n)

1-8-2 FRE函数

一、APPLE SOFT中

格式：FRE(X)

功能：函数会显示内存中还有多少单元可用，以字节为单位计算。

说明：如果可用的内存超过32767个字节，FRE(X)会给出一个负数，加上65536就是实际所具有的可用存贮空间。

二、MBASIC

格式：FRE(X)显示出未被占用的内存单元，X为虚拟参数。

FRE(X$)作用同上。但在未显示之前先将内存中不用的数据集中在一起。

三、IBM PC BASIC的FRE(X$)用法同MBASIC。

四、CROMEMCO中的FRE(X)用法同APPLE SOFT。

1-8-3 PEEK(X)函数

一、APPLE SOFT中

格式：PEEK(X)

功能：函数PEEK(X)可以把地址X的内容用十进制显示出来。

二、M BASIC中

格式、功能同SOFT但0≤X≤65536，0≤PEEK≤255

三、PC BASIC的PEEK(X)同APPLE SOFT

四、CROMEMCO的PEEK

格式：PEEK(X)

功能：PEEK以十进制给出内存单元X的内容。

说明：X的范围在0≤X≤32767。X也可用16进制数，范围应在0≤X≤%FFFF%之间。

1-8-4 POKE语句

格式：POKE‹M，N›

功能：POKE语句会将N的值化为二进制数放在地址M里。

说明：M的值应在－65535～65535之间；

N的值应在0～255之间。

MBASIC的M值在0≤M≤65535；

PCBASIC中POKE语句的作用同MBASIC；

CROMEMCO中：若M为10进制则0≤M≤32767用16进制则0≤M≤%FFFF%。0≤N≤255。

1-8-5 USR函数

一、APPLE SOFT的USR函数

格式：USR(表达式)

功能：USR函数可以调用机器语言子程序。

二、MBASIC的USR函数

格式：USR〔n〕(X)

功能：调用机器语言子程序。

说明：n可有可无，且0≤n≤9。n为子程序编号，若无n时视为USR0X为参 数。

三、IBM PC中的使用格式为：USRn(X)

功能：调用机器语言子程序。

四、CROMEMCO的USR函数

格式：USR(m，P1…Pn)

其中：m为机器语言程序的地址。P1—Pn是参数，被转换为16位整数。

功能：调用机器语言子程序。

1-8-6 WAIT语句

一、APPLE SOFT中

格式：WAIT‹M，N›〔，P〕

功能：语句允许使用者在程序中插入有条件性的暂停。只有reset可以使WAIT失效。

说明：M的范围在－65535～65535；N，P是0到255的十进制整数。执行WAIT时将它们转换成二进制数，并进行测试，根据结果，决定是否连续执行该语句。

二、MBASIC中

格式：WAIT‹K，I›〔,J〕

其中：K为通道号，0≤K≤255。I，J是整数，J的省略值为0。

功能：同APPLE SOFT。

三、IBM PC中的WAIT 除M的范围为0～65535，其它的同APPLE SOFT。

1-8-7 CALL语句

一、APPLE SOFT中

格式：CALL M

功能：CALL会执行内存地址M中由机器语言写成的程序。M 的 范围在－65535～65535之间。

二、MBASIC的CALL可调用子程序。其格式为：

CALL‹变量名称›〔参数〕

功能：1.调用Z-80的汇编语言子程序。

2.用来调用一个FORTRAN子程序。如

```
100  CALL ROUT(I, J, K)
```

三、PC BASIC的CALL语句格式为：CALL

功能：调用代码子程序

1-8-8 HOME(清屏)语句

格式：HOME

功能：清除屏幕，并把光标移在屏幕左上角。

说明：语句HOME和“CALL-936”及“ esc@ RETURN ”效果一样。

1-8-9 CLEAR语句

格式：CLEAR

功能：不带任何参数，会清除任何变量、数组及字串，并将数据指针移到数据区的开始处，如

```
10 LET X=5 : PRINT X
20 CLEAR
30 PRINT X
RUN
 5
 0
```

1-8-10 FLASH、INVERSE、NORMAL语句

格式：FLASH
INVERSE
NORMAL

功能：三个语句用来设定屏幕输出型。

INVERSE会将屏幕设成相反型。即原来屏幕上的东西，黑的变为白的；白的变为黑的。

FLASH会将屏幕变成闪灭型，即屏幕变成一闪一闪的，也就是每隔一段时间黑白颠倒一次。

NORMAL将屏幕变为正常型，即黑底白字。

说明：前两个语句是对输出而言，第三个则输入、输出一样（INVERSE、NORMAL也为MBASIC所有）。

1-8-11 SPEED语句

格式：SPEED=n

功能：可以设定屏幕送出字符的速度，或其他输出/输入装置的速度。最低速度是0，最高速度为255，超出范围会出错。

1-8-12 TRACE、NOTRACE语句

格式：TRACE
NOTRACE

功能：TRACE可以指定机器进入除错状态，当程序执行时会把每行行号显示出来。

NOTRACE与TRACE相反，可以关闭追踪除错。

1-8-13 VTAB语句

格式：VTAB n

功能：执行VTAB语句时，光标移到第n行。屏幕上方为第1行，最下方为第24行，n的范围在1～24之间。此语句只会使光标上下移动。

1-8-14 HTAB语句

格式：HTAB n

功能：HTAB语句会使光标从显示行左边移动n个位置。当n>40而屏幕为40列时，则光标移动到第((n-1)MOD40)+1个位置上。n的范围在0～255之间，光标有1～255个位置。HTAB 0使光标移在256个位置上。此语句只能使光标左右移动。

说明：VTAB和HTAB及1-8-17的PDL函数也为MBASIC所有。

1-8-15 HIMEM语句

格式：HIMEM ：M

功能：可以设定BASIC程序中（包括变量）允许使用的最大内存地址。

1-8-16 LOMEM语句

格式：LOMEM

功能：用来设定内存中的最低地址

1-8-17 函数PDL

格式：PDL(n)

功能：函数PDL会得到游戏控制（摇捍Padel）的现在值。这个值在0～255之间。而函数值是由自变量决定的。n的范围在0～3之间。

说明：

一、游戏控制其实就是一个由0到150K欧姆的可变电阻器。

二、如果有连续两个PDL，则由第二个PDL读出的值，会受到第一个的影响。为了准确，最好在两个PDL函数中间插上几个语句，或插进几个停滞回路。像FOR I=1 To 10 : NEXT I

三、当n＜0或n＞255时会出现错误信息

? ILLEGAL QUANTITY ERROR

四、4≤n≤255时，PDL的值会是0～255之间的不可予期的数。可能会有些副作用，而干扰程序的运行。

1-8-8——1-8-17为APPLE SOFT独有的语句。

1-8-18 MBASIC的几个语句

一、RANDOMIZE用于产生一组新随机数。

格式1：RANDOMIZE 在执行时屏幕上显示Random number seed(－32768 to 32767)?当打入－32768到32767之间任意数后，程序继续运行。

格式2：RANDOMIZE M 由M中直接取出随机数。程序继续运行。

二、CHAIN语句用于调用磁盘中的程序。

格式1：CHAIN“＜程序存盘名称＞”

功能：执行被调用程序，原调用程序消失。

格式2：CHAIN“＜文件名称＞”,n

功能：使被调用程序从第n行执行。

格式3：CHAIN MERGE“＜文件名称＞”

功能：把原程序和被调用程序合起来执行。

格式4：COMMON〈变量〉,(〈数组〉)

功能：使原程序中定义的变量也为被调用程序所用。

格式5：CHAIN MERGE“〈文件名称〉”, DELETE n-m

功能：先删去原程序中n到m之间的程序段，再与调用程序联结。

三、KILL语句

格式：KILL〈文件名称〉

功能：删除磁盘中某一文件。使用时将文件全名称用双引号引括。

四、ERASE 语句

格式：ERASE〈数组〉

功能：可删去定义过的数组。

MBASIC 的其他函数如下表

函　　数	作　　用
INP(X)	自输入通道X输入一个字节。0≤X≤255
OUT X,Y	将指定字节Y送至指定的输出通道X。
VARPTR(X)	显示变量X在内存的第一个字节的地址
VARPTR(#文件号)	给出磁盘文件在内存缓冲区的开始地址
BUTTON(0)	用来指出按钮是否被按下
BEEP	可产生指定音高及持续时间的声音

1-8-19　IBM PC BASIC的其他语句

语　　句	功　　能	分　类
COM(n)ON	允许捕俘通讯中断	非输入输出语句
COM(n)OFF	禁止捕俘通讯中断	
COM(n)STOP	停止捕俘通讯中断	
COMMON	定义公共变量	
DATA$=X$	设置日期	
DEF SEG=地址	定义代码段地址	
DEF USRn	定义代码程序n的始址	
ERRORn	模拟错误号n	
KEY ON	显示功能键清单	
KEY OFF	停止显示功能键清单	
KEY LIST	列出功能键	
KEYn,X$	定义功能键	
KEY(n)ON	允许捕俘功能键中断	
KEY(n)OFF	禁止捕俘功能键中断	
KEY(n)STOP	停止捕俘功能键中断	
MID$(V$,n,m)=Y$	替换部分字符串	
MOTOR	接通磁带机电机	
ON COM(n) GOSUB	设置通讯陷井	
ON KEY(n) GOSUB	设置功能键陷井	
ON PEN GOSUB	设置光笔陷井	
ON STRIG(n) GOSUB	设置操纵杆陷井	
PEN ON	允许捕俘光笔中断	
PEN OFF	禁止捕俘光笔中断	
PEN STOP	停止捕俘光笔中断	
STRIG ON	允许使用操纵杆按钮	
STRIG OFF	禁止使用操纵杆按钮	

语　　句	功　　能	分　　类
STRIG(n) ON	允许捕俘操纵杆中断	非输入输出语句
STRIG(n) OFF	禁止捕俘操纵杆中断	
STRIG(n) STOP	停止捕俘操纵杆中断	
TIME$=V$	设置时间	
BEEP	扬声器警报	输入输出语句
CLS	清屏语句	
GET	读屏幕上的图形信息	
LOCATE	设置光笔位置	
OPEN"COMn:..."	打开异步通讯文件	
PLAY	按字符串演奏音乐	
PRINT*f,USING	按格式写数据给文件	
PUT	把图形显示到屏幕上	
SOUND	发出指定频率的声音	
WIDTH	设置屏幕宽度	
HEX$(n)	转换成16进制的字符串	字符串函数
CSRLIN	显示光标所在行的行号	输入输出函数和辅助函数
DATE$	系统日期	
ERL	出错所在行行号	
ERR	出错号	
INP(n)	从指定槽口输入一个字节	
LOC(f)	文件读写指针的位置	
PEN(n)	读光笔	
POINT(x,y)	显示指定点的颜色	
SCREEN(x,y,z)	读指定位置的ASCII码或色码	
STICK(n)	读操纵杆坐标	
STRIG(n)	读操纵杆按钮状态	
TIME$	系统时间	
VARPTR$(v)	变量的类型及地址串	

1-8-20　CROMEMCO 的其他语句

语　　句	功　　能
MAT　M=算术表达式	使数组中所有元素为表达式值
DEG	使三角函数运算以度为单位
RENAME字串-1,字串2	以新名字串2替换已存文件旧名
SET表达式1,表达式2	用来改变系统参数表达式1的值
RAD	使三角函数计算取弧度单位
SYS(算术表达式)	根据自变量提供系统信息
CON	使程序继续运行
ECHO	恢复在控制台终端显信息

语　　句	功　　能
NOECHO	不显示对INPUT的回答
ERASE	将从文件目录中删除文件
ESC	重新恢复ESC键的功能
NOESC	禁止控制台终端ESC键功能

1-9　低分辨率绘图语句

1-9-1　显示状态的选择　APPLE II 机有三种显示状态：文本状态、低分辨率图形状态和高分辨率图形状态。TEXT 语句使屏幕从低分辨率图形或高分辨率图形回到全屏幕文本状态(24行，每行40个字符)。

IBM PC BASIC 也有三种显示状态，即文本状态、中分辨率、高分辨率图形状态。使用SCREEN语句可选择所处的状态。

1-9-2　GR语句

格式：GR

功能：执行 GR 语句后，显示屏上除了最低部四行被保留用于显示文字外，其余的均被清为黑色，供图形使用。机器进入低分辨率图形和文本混合状态。这时图形部分可绘40×40个图形元素。而光标在四行文本的左上角。在使用GR后再使用POKE-16302，0或POKE 49234，0屏幕会全部用来绘图，这时整个屏幕为48行(0～47)，40列(0～39)，可绘1920个图形元素，称为全屏幕图形状态。只要再使用POKE-16301，0又能在底部留出四行写程序。

MBASIC与SOFT稍有差别，具体如下：

格式：GR〈屏幕号码〉，〈颜色号码〉

功能：屏幕号码为0或1；GR语句使机器进入低分辨率图形状态，并清除画面。

当屏幕号码为0或省略(机器认为是0) 时，进入混合状态。若这个号码为1，则进入全屏幕图形状态。

颜色号码用来选择画面(背景)的颜色。若被省略，则机器认为是0。

例：10　GR　　则与SOFT的GR相同。

　　20　GR　0，15　　背景白色，混合型

IBM PC BASIC 又异于以上两种：使用SCREEN语句使机器进入中分辨率图形状态，此时屏幕分为200行、320列可绘64000个图形元素。其格式为：

　　SCREEN 1，0　　有彩色

　　SCREEN 1，1　　没有彩色

1-9-3　COLOR(颜色选择)语句

一、APPLE II SOFT中：

格式：COLOR＝〈表达式〉

功能：使机器对表达式的值取整，整数范围在0～255，再以16的模选择绘图时的颜色。16种颜色编号如P128上表所示。

二、MBASIC 中：

格式：COLOR＝〈颜色号码〉

功能：由颜色号码选择低分辨率状态下绘图时的颜色。十六种颜色如P128上表所示：

编 号 (16进制)	颜 色	编 号 (16进制)	颜 色	编 号 (16进制)	颜 色
0	黑	6	中 蓝	C	绿
1	深 红	7	浅 蓝	D	黄 色
2	深 蓝	8	棕 色	E	水 蓝
3	紫	9	桔 红	F	白 色
4	深 绿	A	灰		
5	深 灰	B	粉 红		

说明：要找出在屏幕上某一色点的号码，可用SCRN命令。COLOR只能用在低分辨图形

三、IBM PC BASIC中：

格式：COLOR‹颜色号码›，‹颜色组码›

功能：颜色号码用来选择背景(画面)的颜色，共十六种颜色如下：

编 号	颜 色	编 号	颜 色	编 号	颜 色
0	黑	6	棕 色	12	浅 红
1	蓝	7	白 色	13	浅 紫 红
2	绿	8	灰	14	黄 色
3	青	9	浅 蓝	15	高 密 度
4	红	10	浅 绿		白 色
5	紫	11	浅 青		

颜色组码用来选择前景的颜色。前景的颜色分为两组：

第0组颜色(0)	第一组颜色(1)
1—绿	1—青
2—红	2—紫红
3—棕	3—白

例：100 COLOR 12, 0

这个语句使背景为浅红色，前景为0组中颜色，若没有使用新的COLOR语句，一直有效。从不同组中选择绘图颜色的工作由 PSET 语句来完成。

1-9-4 画点语句

一、APPLE II SOFT

格式：PLOT‹表达式1›，‹表达式2›

功能：它在以表达式1为横坐标X，表达式2为纵坐标Y 的位置画一个小色点。小色点颜色由COLOR语句确定。若缺此语句，则设定COLOR = 0。

说明：X的范围为0～39，Y的范围为0～47，否则给出错误信息：

? ILLEGAL QUANTITY ERROR

原点(0, 0)在屏幕左上角。

二、MBASIC的格式与功能与 APPLE SOFT 完全相同。当X、Y的值超出所允许的范围时，给出错误信息：

"ILLEGAL FUNCTION CALL"

三、IBM PC BASIC的PSET 语句与APPLE的PLOT语句的功效相同。

格式：PSET(X,Y)，N

功能：语句使X行、Y列处画上一个现在有效的色组中，N对应的色点。

例. 10 SCREEN 1,0

20 COLOR 12，0背景为浅红，前景0组

30 PSET(100,150)，1 在(100,150)处画绿点。

说明：

1.在PC BASIC 中，坐标系统以屏幕左上角为原点(0,0)横坐标刻度为0,1,2,…,319。纵坐标的刻度为0,1,2,…,199。这种坐标系叫屏幕坐标系统。

2.PSET(X,Y),N中，(X,Y)是欲画点的屏幕坐标，X,Y 应为整型常数或表达式。如果它们的值超出了屏幕坐标系统所允许的范围，则本语句不起作用。

3.当N被省略时，该点的颜色就选3或1。

4.点的位置也可用相对坐标形式STEP(DX，DY)来给出。假设画点之前光标的现行位置为(CX,CY)则该点的实际位置就是(CX＋DX，CY＋DY)。

1-9-4 擦点语句

PC BASIC的PRESET(擦点)语句

格式：PRESET(X,Y),C

功能：在坐标为(X,Y)的位置上以C指出的颜色画出一个点。

说明：当格式中C被省略时，表示用底色画这个点，从而擦掉原来(X,Y) 位置上的点。

1-9-5 画线语句

一、APPLE SOFT的画线语句

1.HLIN语句

格式：HLIN X1，X2 AT Y

功能：X1，X2，Y可以是表达式。HLIN语句在低分辨率状态下，画出一条从 (X1,Y)到(X2,Y)的一条水平色线，其颜色由前面最近的 COLOR 语句确定。若无 COLOR语句则认为COLOR＝0。

例. 10 GR

20 COLOR＝3 用紫色绘图

30 HLIN 14,20 AT 39 画一条从(14,39)到(20,39)的紫色水平线。

说明：X1、X2 必须在0～39之间，而Y必须在 0～47之间。否则，屏幕将显示错误信息

? ILLEGAL QUANTITY ERROR

2. VLIN语句

格式：VLIN Y1，Y2 AT X

功能：画出一条从(X,Y1)到(X,Y2)的铅直色线。颜色由最近执行过的COLOR语句决定。

例. 10 GR

20 COLOR＝1

30 COLUMN＝0

40 VLIN 0,39 AT COLUMN

50 COLUMN＝COLUMN＋1

60 IF COLUMN＜40 THEN GOTO 40

100 END

说明：X必须在0～39，Y1,Y2必须在0～47。否则会显示错误信息

? ILLEGAL QUANTITY ERROR

当屏幕为文本或混合型时，若Y2在0～47之间，会在应显示一行色点的地方出现一行字符。

HLIN、VLIN两个语句在高映像状态下无效。

二、MBASIC的画线语句的格式、功能、范围同APPLE SOFT，当X,Y的范围超出规定时，出现：ILLEGAL FUNCITON CALL 的错误信息。

使用HLIN 语句画水平线时．若两端点在绘图画面中，则连线由点组成；若Y为40～47的文字画面或混合画面中，两点的连线为字符所组成（一个字符占两个并排色点的位置）。

三、IBM PC的画线语句

IBM PC的LINE语句具有HLIN和VLIN两个语句的功能。其使用有如下三种形式：

格式1：LINE(X1,Y1)－(X2,Y2)，C

功能：分别以(X1,Y1)和(X2,Y2)为起点和终点画一条直线。起点和终点也可以用相对坐标形式给出。若缺省(X1，Y1)，则表示起点的位置就是游标的目前位置。参数C是直线的色码。若缺省C值，则认为C值为3。

例．10 LINE(0,0)－(319，199) 画一条屏幕的对角线。

20 LINE －(100,90) 画一条从游标目前位置到(100,90)的线段。

30 LINE (160,100)－STEP(159,0) 从中心点向右画一条水平线

40 LINE－STEP(400,－100) 自游标画一条至右方400格，上方100格的直线。

格式2：LINE(X1,Y1)－(X1,Y1),C，B

功能：画出分别以(X1，Y1)和(X2，Y2)为顶点的矩形。C用以指出线条的颜色号码，取值范围在0～3之间。B是一个表示画矩形的标志字符。是BOX的缩写。

格式3：LINE(X1,Y1)－(X2,Y2)，C,BF

功能：这一格式不但画出以(X1,Y1)和(X2,Y2)为顶点的矩形，而且要用C选择的彩色去把这个矩形填满，实际上是画出长方块。

例．10 SCREEN 1，0 进入彩色中分辨率状态

20 COLOR 12，1 背景浅红色，前景1组

30 LINE (20,50)－(80，199)，1 自(20,50)到(80,199)画一条青色线

50 LINE(0,0)－(319,199),,B 画一个最大的白色矩形方框

60 LINE(0，0)－STEP(160,100),1,BF 画一个填满彩色1的长方块

1-9-6 连续画线语句

一、APPLE II在低分辨率状态不能连续画线。

二、IBM PC的连续画线语句

格式：DRAW‹字符串›

其中字串由一系列的画图命令所组成，每个画图命令有两部分：一个命令字母及一个数值n,画图命令相互之间用分号隔开。可以使用的主要命令有：

Cn 选择颜色n。中分辨率图形时n的范围为0～3；高分辨率图形时n的范围为0,1。

Sn 设置比例因子。n的范围为1～255，n被4除是比例因子，其范围为1/4～64。

An 设置画图的角度，n可以是0、1、2和3，分别对应0度、90度、180度和270度。

Un　向上画线；

Dn　向下画线；

Ln　向左画线；

Rn　向右画线；

En　向右上画对角线；

Fn　向右下画对角线；

Gn　向左下画对角线；

Hn　向左上画对角线；

Mx,y从现行位置向(x，y)画线。若x前面冠以正负号，则表示以现行位置为起点，按相对坐标画线。

说明：

1.上面U、D、L、R、E、F、G、H、及M命令在画线时所移动的点数，是n倍的比例因子。这些命令字母的前面还可以放一个前缀字母B，表示只移动不画线。如果冠以前缀字母N，则表示在画线之后光标又返回原来的起始位置。

2.所有命令中的常数n也可以改用变量，这时命令中的n将由“=变量名”来代替。

3.命令字符串中又可以包括一个或几个子命令字符串，只要用X命令就行了。格式为

X　字符串

这条命令非常有用。因为它在DRAW语句所画出的一个“对象”中，独立地定义了其中的某些部分，从而可以单独对它们进行某些处理。

例. 10　DRAW　“R50；U50；L50；D50”画一方框

10　DRAW “R100；H50；G50”画一个三角形

10　DRAW　“M+100，0；M-50，-50；M-50,50”画一个三角形

10　DRAW “BM60，140R200G30L140H30”画梯形

10　DRAW “BM110，125；C2；R100；H50；G50”以(110,125)为起点画一个红色三角形。

1-9-7　SCRN(显示颜色代码)语句

APPLE II SOFT中，SCRN函数给出在低分辨状态下，点(x,y)的颜色代码。其格式是：

SCRN(x,y)

1-10　高分辨率绘图语句

1-10-1　HGR和HGR2语句

一、HGR语句

(一) APPLE II SOFT中

格式1：HGR

功能：使机器进入高分辨率图形(280×160)和文本混合工作状态。即屏幕下面留4行写程序，其余清除为黑色，以作绘图用。再使用POKE—16302，0可进入全屏幕图形状态(280×192)。

说明：HGR不能包含在变量名称里，否则会先执行HGR命令，然后给出

? SYNTAX ERROR 如

HGRIP=4

就会使机器进入高分辨率状态，清除原有程序。

格式2：HGR2

功能：使全屏幕进入高分辨率图形状态(280×192)。这时内存容量应大于24K。

(二) GBASIC

格式：HGR‹幕码›，‹色码›

其中幕码为0～3，色码为0～7。

功能：使屏幕进入高分辨率图形状态。

说明：幕码指定所采用的显示模式：

屏幕号	清除屏幕	屏幕模式	
0	清除	280×160	图形和4行文本
1	清除	280×192	全屏幕图形
2	不清除	280×160	图形和四行文本
3	不清除	280×192	全屏幕图形

色码指定屏幕的颜色。如无色码，则自动设为0。若色码为0或1时，所指定的颜色充满整个屏幕。参考HCOLOR中颜色名称及对应的色码。

例. 10 HGR 同 SOFTBASIC

10 HGR 1，2 设280×192模式，紫色屏幕

10 HGR 3 设280×192模式，不清除

(三) IBM PC中

格式：SCREEN 2

功能：屏幕进入高分辨率图形(640×200)状态。每一点的位置用(x,y)表示，即(列，行)。清除文本或画面。背景只有黑、白两色。原点在左上角。

1-10-2 HCOLOR(选择颜色)语句

一、APPLE II SOFT中

格式：HCOLOR = ‹色码›

说明：

1.在屏幕为高分辨率图形型时，用色码选定绘图时的颜色。颜色名称及代码如下

0	黑1	4	黑2
1	绿(依显示器而定)	5	橙(同1说明)
2	蓝(同上)	6	蓝(同1说明)
3	白1	7	白2

以上颜色依显示器略有差别，也因点所在位置而受影响。例如一个高分辨率的点的色码为3，而横座标值为偶数，则显蓝色；若横座标值为奇数，则会显示绿色。但如果横座标包括了偶数与邻近的奇数，则会显出白色。即若点(x,y)及点(x+1,y)都画出时，才呈白色。

2.HCOLOR不会被HGR、HGR2、RUN所改变。在第一个HCOLOR语句之前。关于高分辨率彩色图形的颜色是未定的。

二、GBASIC中

格式：HCOLOR = ‹色码›

其中色码为0～12的整数。

功能：HCOLOR用色码选定高分辨率绘图时的颜色。

说明：

1.可用的颜色及代码如下：

0黑　3白　6蓝　9白1　12反向

1绿　4黑　7白　10黑2

2紫　5橙　8黑1　11白2

表中各不同的黑色与白色有如下区别：

0，3，4，7画出极细线。

黑1，白1，黑2，白2,(8,9,10，11）画出一个较大的点，即与用绿、紫、橙、蓝画的点和线同样的粗细。若要在同位置画同宽度的点或线，黑1与白1应与绿或紫混用，而黑2与白2需与橙蓝混用。

2.如果装置是黑白萤幕显示器，则只用0,3，4，7色码。色码应在HGR 语句中定义。如果HGR没有定义。色码会自动设为0。直到HCOLOR语句中出现定义另一颜色为止。

3.HCOLOR只用在高分辨率图形。

4.由于显示器的不同，当使用HCOLOR＝3(白)或HCOLOR＝7(白)画高分辨率图形时同APPLE SOFT的HCOLOR说明。

1-10-3　连续画线语句

一、APPLE SOFT的HPLOT 语句

（一）格式

格式1：HPLOT　‹X1，Y1›

格式2：HPLOT　TO　‹X2，Y2›

格式3：HPLOT　‹X1，Y1›　TO　‹X2，Y2›[{TO　XN,YN}]

（二）功能

1.格式1在以(x1,y1）表示的位置上画一个色点。颜色由最近执行过的HCOLOR 语句确定。如果前面没有执行过HCOLOR语句，则颜色未定。

2.格式2的HPLOT语句由前面已画过的点到(x2,y2）画一条色线。颜色与前面的点的颜色一致。如果前面没有画过的点，这条语句画不出线来。

3.格式3从x1,y1画线到x2,y2，再由x2,y2画线到x3,y3，并继续按次序画线到xN,yN。

例如：30 HPLOT 0,0　TO　279,0　TO　279,159　TO　0,159　TO　0,0 这时若打入RUN命令，就会看到沿屏幕四周画出一个色方框。

（三）说明

1.x必须在0～279，y必须在0～191否则有信息？ILLEGAL QUANTITY ERROR出现。

2.语句中的x、y可为表达式。

例．x＝150＋INT(RND(1)•95)

　　y＝80＋INT(RND(1)•73)

3.在HPLOT语句之前，一定要有HGR或HGR2语句，以防止内存中的数据及程序被破坏。

二、GBASIC的连续画线语句

（一）格式

格式1：HPLOT[x1,y1][TO　x2，y2][{TO　xN,yN}]

格式2：HPLOT TO ‹x2，y2›

（二）功能

1.在高分辨率图形型，以(X1,y1)、(X2,y2)等座标定义画出点或线；

2.格式2的HPLOT由已画的前一点，画一条线到(X2,y2)。

（三）说明：语句定义内容不得超过屏幕限制和不得超过239个字符的限制。

例. 10 HGR　　进入高分辨率绘图

20 HCOLOR=2　　以绿色画图

30 HPLOT TO 24，125　由已画点到(24,125)画一条线。

三、IBM PC的连续画线已在中分辨率说明。

1-10-4 画圆语句

一、APPLE II的画圆是通过数学里的参数方程来实现的。圆的参数方程为：

$X=a+r\cos Z$

$Y=b+r\sin Z$

其中(a,b)是圆心的座标，r是半径，Z是点(X,Y)到(a,b)的连线和水平线间的夹角。使 Z从0°到360°之间，每1°计算并打印一个点(X,Y)就可以画出圆来。

二、IBM PC的CIRCLE(画圆)语句

格式：CIRCLE(XC,YC),R,C,AS,AE,ASP其中，(XC,YC)为圆心座标，R为半径，C为彩色码，AS和AE分别为圆弧的起始角和结束角（单位为弧度）。取值范围为$-2\pi\sim2\pi$，以逆时针方向进行度量。当角度为负值时，表示圆弧的端点应与圆心相连。若无AS或AE，则认定AS、AE为0。

CIRCLE语句中最后一个参数ASP 表示圆的“纵横比”，这是为了避免因纵横座标的单位不等而造成椭圆而设置的。若给出ASP的话，则Y方向半径应为 R*ASP。若缺此值时，则中分辨率时为5/6，而高分辨率时为5/12。

例. 10 CIRCLE(30,10)，10以(30,10) 为圆心，10为半径画圆。

20 CIRCLE(160,100),75,1.57080，4.71239 以(160,100)为圆心，75为半径的左半圆。

30 CIRCLE(160,100),75,,,,1 以(160,100)为圆心，短轴为75的椭圆。

三、IBM的着色语句

格式：PAINT(X,Y),CP,CB

其中(X,Y)为边界线内的任意一点（称为内点）的座标。它可以是绝对座标形式，也可以是相对座标形式。CP表示需要在此区域内着色，CB为边界线的颜色。如果参数CP省略，则认为CP值为3(中分辨率)或1(高分辨率)；如果参数CB省略或CB不是任何边界的颜色，则将对整个屏幕进行着色。若边界线上有“孔”，则彩色会从孔中漏出，并沿着边界进行着色工作。

1-10-5 查点语句

GBASIC中有HSCRN语句，能指明某点在不在图形上。

格式：HSCRN(X,Y)

功能：若点(X,Y)在图形上，就显示“-1”。

说明：HSCRN与SCRN不同，SCRN不分辨颜色。X范围在0～279，Y的范围在0～191

1-11 高分辨率造型

1-11-1 造型表及其生成

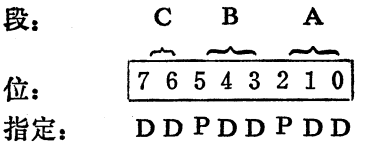
造型：将图形的轮廓线，分解成许多基本矢量的集合，称为造型或形状定义。

造型表：一个或多个造型及其索引，构成造型表。

造型表可使用键盘存入磁盘或卡式录音带中，是绘图的根据。

在造型中的每个字节可分为三段，每段可指定一个绘画矢量：要不要画图，及一个移动方向（向上、向下、向左或向右）。DRAW（画造型）语句、XDRAW（消除造型）语句将从造型里每个字节中每段顺序从右到左取出矢量。直到全0的字节结束。

下图显示的A、B、C三段的安排是造型的一个字节；



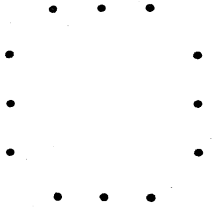
每两位(位对)DD，指定一个移动的方向，而P指定在移动之前要不要画图。说明如下：

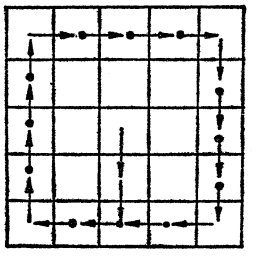
如果DD=00 向上移　　如果P=0 不画
=01 向右移　　=1 画
=10 向下移
=11 向左移

必须注意：C段只能移动而不能画图（没有P位），当它为0时，APPLE SOFT不理会它，将继续往下寻找另一个造型矢量。

每个字节可以表示到三个绘画矢量，一个在A段，一个在B段，而第三个在C段。

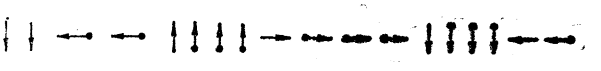
假设要画左下边这个图形；



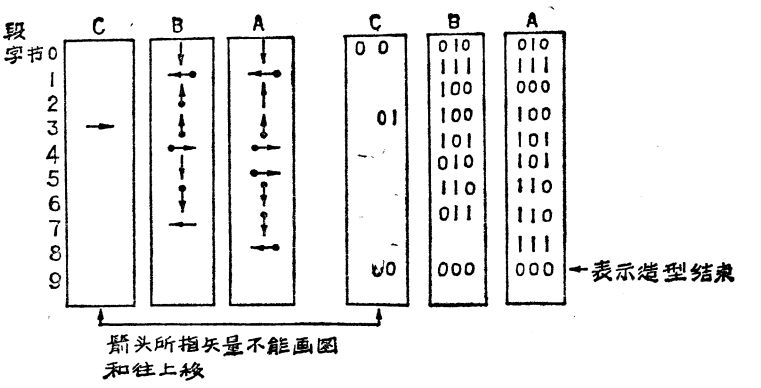


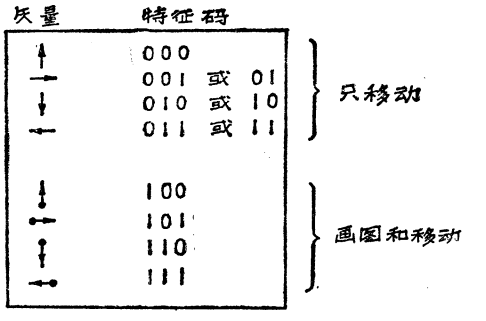
第一步工作是先在纸上绘图，可在每个方块里画一个点，再决定以那一点为起始，假设从中间开始。然后经过图形每个点按顺时针（或逆时针）方向画成矢量链（只能90度角来转弯）。如右上图。

第二步，重组这些绘画矢量，写成一条线：



第三步，制表



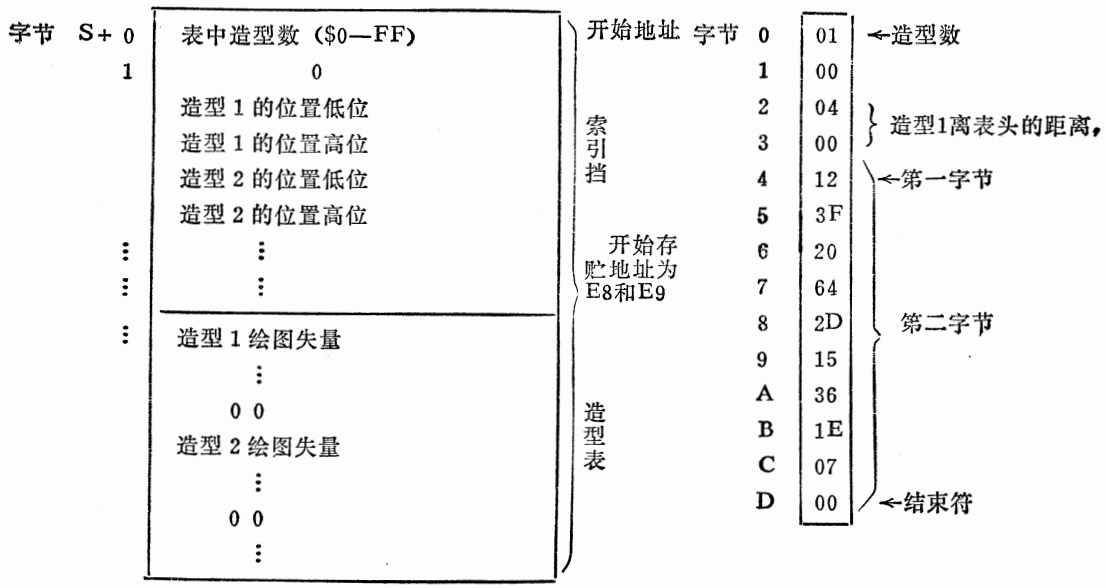


对于每个矢量，先决定特性码(其位的编码)，然后把这个特性码置在表的可用段里。如果这个位编码不适合（例如，在C段不能画点），或是00（或000），且在字节的最后，就跳过此段，到下一段去。结束矢量编码，看是否有错。

现在继续建立另一张表，如下图，重抄前面表中矢量码，再换算成一系列十六进制表示的字节。

段		C	B	A		字节	
字节	0	0 0	0 1	0 0 1 0	=	1 2	
	1	0 0	1 1	1 1 1 1	=	3 F	
	2	0 0	1 0	0 0 0 0	=	2 0	
	3	0 1	1 0	0 1 0 0	=	6 4	
	4	0 0	1 0	1 1 0 1	=	2 D	
	5	0 0	0 1	0 1 0 1	=	1 5	
	6	0 0	1 1	0 1 1 0	=	3 6	
	7	0 0	0 1	1 1 1 0	=	1 E	
	8	0 0	0 0	0 1 1 1	=	0 7	
	9	0 0	0 0	0 0 0 0	=	0 0	←造型结束
16进制		数位1		数位2			

有了一连串十六进制字节后，再加上索引就成了一个完整的造型表。如P137左图：



造型表

先参看右上图，因为只有一个造型，所以索引很简单。假设造型表启始地址为S，必须先存表中有几个造型，以十六进制数表示，范围在0～255之间（此例为1）。在此例中，形状定义是从S+4开始，所以造型#1对于S的相对位置是4（0004，以十六进制表示）所以索引字节S+2应存04，而索引字节S+3应存00然后将上面的一连串十六进制字节一一存进去。

1-11-2 造型表的存贮 有了造型表以后，就可以存入存贮器备用。但它不能存入图形缓冲区，因为在执行GR或HGR语句时，会使这些区域内容被清除，以存贮区的分配中我们知道，在低分辨率图形上部有一个区域

3072—8191（$C00—$1FFF）

在高分辨率图形区的上部也有一个区域

24576—49152（$6000—$C000）

可供使用。我们选择上例的起始地址为1DFC。

使用CALL—151使机器进入键控状态后，再送入图形地址，例如：1DFC

在回车之后APPLE会显示此地址及其内容。这时若按':'号，再打入两位十六进制数，然后按 RETURN，则此数存入指定地址中。

要存多个十六进制数于连续的地址中，只要先打入第一个地址。

1DFC:

然后打入要存进的各数，并以空格分开，如：

1DFC:01 00 04 00 12 3F 20 64 2D 15 36 1E 07 00 return。

于是整个造型表存贮在内存区了。还可以用回车键检查各地址的存贮内容。

然后，将造型表的起始地址存入E8(低二位)和E9(高二位)，浮点BASIC软件将从这里取得造型表的起始地址，此时应打入

E8:0060 ↵

1-11-3 DRAW语句

一、格式

格式1：DRAW n AT ‹X, Y›

格式2：DRAW n

二、功能：

1.格式1将造型表中第n个造型绘在以(X, Y)为启始的地方。

2.格式2将造型表内的第n个图形绘出，其位置由最近使用的HPLOT语句决定。如果没有设定坐标，则认为坐标为0。

三、说明：

1.n的范围在0～255之间。X的范围在0～279之间，Y必须在0～191之间。

2.关于形状的颜色，旋转角度及尺寸要先指明。

3.DRAW语句要在高分辨率图形型中使用。

1-11-4 XDRAW 语句

1.格式：XDRAW n〔AT X, Y〕

2.功能：含意和DRAW相同，不同的是它用以消除一个造型，实际上是用补色再给一次造型而达到抵消的目的。

3.说明：互补的颜色见下表。

颜色	补色
黑	白
白	黑
紫	绿
桔红	蓝
绿	紫
蓝	桔红

1-11-5 ROT语句

格式：ROT＝n

功能：用以使造型旋转。当SCALE＝1时，只有四种旋转

ROT＝0	0度
16	90度
32	180度
48	270度

当SCALE≥5时，可以有0到63，共64个旋转角度。

1-11-6 SCALE语句

1.格式：SCALE＝n

2.功能：用以设定矢量的大小。如

SCALE＝5

则每个矢量为5个点，最大矢量为SCALE＝0，此时，每个矢量给成256个点。

3.说明： 在ROT语句和SCALF语句中只有在下个非空白字符是(＝)取代符号时， 才能当成保留字。

1-11-7 SHLOAD语句

1.格式：*SHLOAD*

2.功能：用来将磁带上已存的造型表调入机器。造型表紧靠着HIMEM:的下面存放。其起始地址由机器专用程序自动给出。若调入第二个造型代替第一造型表，则HIMEM会自动更换。

3.说明：

(1) 以存贮器的分配中可以看出，对一个16K的系统来说，HGR将使8192-16384的图形存贮区清除。所以要在执行SHLOAD语句之前，置

HIMEM:8192

这样就强迫SHLOAD只能把图形造型放在此值下面。

(2) 对于一个24K的系统来说，不要使用HGR2语句，它会使16384-24575区域清除。或在执行SHLOAD之前，先设HIMEM:为16384，而不用HGR。若存贮容量很大，在24575上面有足够的地址用来存贮造型表，则就没有什么限制了。

只有 RESET 命令可以使SHLOAD中断。它无条件地停止任何命令的执行，并不破坏程序，但某些指针和堆栈被清除，停止后处于监控状态可用命令；

CTRL C

返回浮点BASIC状态，这个命令不破坏存贮的程序。

1-12 数据文件

1-12-1 数据文件的概念

数据文件：存于外部存贮器中的具有名字的数据集合。又可定义为：记录的集合。

记录：同样大小的单元集合。也称是数据项的集合。记录内的字节数，叫做"记录长度"。

数据项：字符的集合。

文件按其性质可分为程序文件、数据文件。

数据文件按存取方式又可分为顺序存取文件和随机存取文件。

顺序文件是按顺序从头至尾逐项存取的。

随机文件的存取是以记录为单位的，记录长度固定，存取时按记录号进行。

下面分别说明各种BASIC的文件处理

1-11-2 APPLE II SOFT的文件处理

一、文件的控制

1.打开文件

当需要对磁盘中某一文件进行存取时，必须首先使用命令，打开文件。

(1) 格式

OPEN f[, Ll][, Ss][, Dd][, Vv]

(2) 说明

f为文件名称，其命名规则是以字母开头和不含有','号的任意具有显示功能的字符组成的字符串，名称的长度不大于30个字符。

L表示记录长度，l以记录所占字节数计算。顺序文件没有此项，但对于随机文件必须用此项指出记录的长度。

d表示驱动器的编号，可为1或2。

s表示接口槽编号，可为1到7。

v表示软盘编号，可为1到254。

只有一个软盘时，系统指定其软盘编号为254，只有一个驱动器时，通常插入接口槽号6。

(3) 功能

① 首先检查是否已有此文件，若是新文件就建立文件目录；

② 若已建立过此文件，检查该文件是否打开，打开时先关闭再打开；

③ 给文件分配一个缓冲区，以便进行数据通讯。

例如：OPEN CLASS, D2

表示在第二个驱动器中建立一个名为CLASS的文件。

OPEN GEF, L91,D2

表示在第二个驱动器上建立一个随机存取文件，其记录长度为91个字节。

2.关闭文件

(1) 格式

CLOSE[f]

文件名可以选择，有该项时只关闭指定的文件。无此项时，关闭所有打开的文件。

(2) 功能

① 把缓冲区中的内容写入磁盘。

② 收回缓冲区另作它用。如果文件存取结束后不关闭，会使缓冲区内容被破坏，造成文件内容的错误。

3.读命令

(1) 格式：READ f[, Rr][, Bb]

(2) 说明：R是记录号，随机文件有此参数，省略时，从文件现行位置读起，刚打开的文件r值为0，表示第一个记录。b是字节数，表示从r记录b字节开始。

(3) 打开文件之后使用READ命令，以后遇到INPUT语句时，不是再向键盘要数据，而是从磁盘上要数据。

4.写命令

(1) 格式：WRITE f[, Rr][, Bb]

(2) 功能：打开文件后，使用WRITE命令不再是向屏幕或打印机输出，而是向磁盘写数据。

以上文件处理命令，在程序中作语句使用时，要用到PRINT语句的特殊功能。即用PRINT CTRL—D；“命令”形式。

二、顺序文件的存取

1.写文件

第一次把信息存到磁盘上去时，首先要定义一个文件名。顺序文件按项存放，项与项之间用RETURN分隔，一项最长可含有32767个字符，如果一行不是以RETURN结束时，以后的字符继续按同一项存放，系统不加入RETURN字符。下面这个程序建立名为GEF的顺序文件。

```
例.10  REM  MAKE  GEF
  20  D$=CHR$(4)  :  REM  CTRL-D
  30  PRINT  D$; "OPEN  GEF"
  40  PRINT  D$; "WRITE  GEF"
```

```
50  PRINT  "……"  (第一个记录)
60  PRINT  "……"  (第二个记录)
70  …
 ⋮
700  PRINT  D$;"CLOSE  GEF"
710  END
```

20行使用了CHR$函数，即把CTRL-D赋给D$， 使用PRINT D$， 磁盘操作系统即认为后面的是磁盘操作命令，30行打开文件， 40行使用了WRITE命令， 使其后面的PRINT语句将数据送入名为GEF的文件。700行是在文件输入结束后将文件关闭。

2.读文件

假设在建立GEF文件时，输入了9项，要想把所存的文件GEF读出来，可使用如下程序：

例.

```
10  REM  RETRIEVE  GEF
20  D$=CHR$(4)  :  REM  CTRL-D
30  PRINT  D$;"OPEN  GEF"
40  PRINT  D$;"READ  GEF"
50  FOR  I=1  TO  9
60  INPUT  A$(I)
70  NEXT  I
80  PRINT  D$;"CLOSE  GEF"
90  END
```

30行打开文件，40行指明以后的INPUT语句是从磁盘名为GEF的文件中读数据。INPUT语句是按项读入，共有9项，用循环语句读入字符串下标变量A$(1)，A$(2)，…，A$(9)中去。

GET语句也可以用于读文件，它与INPUT语句在文件系统中功能基本相同，只是一个字符一个字符读出。

3.增加文件内容

如果要求在磁盘上已有的文件中增加内容，可使用APPEND命令。

(1) 格式：APPENDf〔, Ss〕〔, Dd〕〔, Vv〕

(2) 功能：其作用是把文件指针调到文件尾，然后可以往文件中写内容。因此APPEND命令后面一定跟 WRITE 命令。 此外 APPEND 命令还可以完成 OPEN 命令的功能， 使用APPEND命令时，则可不用OPEN命令。

4.对指定项的读写

在APPLE SOFT中，可使用POSITION命令，对顺序文件的指定项进行读写。

(1) 格式 POSITION f〔, Pp〕

(2) 功能：P为从当前指针位置向后跳的项数，P=0时， 即从当前当指针所指的项开始操作；P=1时，表示从现行指针所指项的下一项开始操作；P=2时，表示从现行指针所指的项跳过一项开始操作，依此类推。

三、文件操作命令

使用下述命令可对文件整体进行操作。

1.删除文件命令

格式：DELETE‹文件名›

2.重新命名文件

格式 RENAME ‹旧文件名›，‹新文件名›

3.文件封锁和释放命令

为保护一个文件不被破坏，可使用文件封锁命令，对指定文件进行保护，不需要时可用释放封锁命令解除保护。

格式：LOCK ‹文件名›〔，Ss〕〔，Dd〕〔，Vv〕

UNLOCK ‹文件名›〔，Ss〕〔，Dd〕〔，Vv〕

四、随机存取文件的读写

随机存取文件的读写和顺序文件相同，必须先打开文件（OPEN），最后关闭文件（CLOSE）。只是OPEN命令中必须增加记录长度这个参数，

例如：OPEN GEF，L91

记录长度的允许范围是1到32767

此外，每次读写要有参数R，它指出文件中的第几个记录。

例如 WRITE GEF，R15

READ GEF，R16

有时，还要加上参数B，它指出记录的开始字节。记录字节编号自0开始。例如

WRITE GEF，R15，B16

READ GEF，R16，B16

但是APPEND和POSITION命令，只用于顺序文件。

1-12-3 MBASIC的文件处理

一、顺序文件

顺序文件中使用的语句及其格式如下：

1.OPEN语句

建立磁盘文件，必须首先是使用OPEN语句将文件打开。

(1) 格式：OPEN “方式”，#‹文件号›，“文件名”

(2) 说明

① 方式可为“I”和“O”，“I”方式是读文件，“O”方式是写文件。

② 文件号为1～15之间的整数。

③ 文件名的命名同程序名的命名，可有8个字母。例如：

10 OPEN “I”，#3，“TEST”

打开名为TEST的顺序文件，文件号为3。

2.PRINT#与PRINT # USING语句

(1) 格式PRINT #‹文件号›，‹变量›

PRINT #‹文件号›，USING‹格式›；‹变量›

(2) 说明：

① 这两个语句均可将数据写入顺序文件。

② PRINT#与PRINT语句的输入格式相同，但为节省空间，一般在输出项之间用分号（；）。例如：

10 PRINT #1，A$；B$；C$

将A$，B$，C$三个字符串存入文件号为1的顺序文件中。

为了使个别存入的数据能够分别被读出，各存入项之间需加“,”号。例如：

```
10 PRINT #1, A$; ","; B$; ","; C$
```

如果被存入的字符串数据中有逗号(，)、分号（;）和有效空格，则需将它们用双引号(“ ”)引起来，才能正确读出。如：

```
20 E$ = "GOOD MORNING, SIR."
30 PRINT #1, CHR$(34); E$; CHR$(34)
```

③ PRINT # USING使用与PRINT USING 相同的格式，规定将数据写入磁盘文件中。

3.WRITE# 语句

(1) 格式：WRITE #‹文件号›，‹变量›

(2) 说明：该语句也是用以将数据写入顺序文件，但与PRINT#语句不同之处是它将数据存入文件时，自动将数据用逗号(，)隔开，并将每一个字符串数据用双引号括起来。例如：

```
⋮
110 A$ = "TYPE"; B$ = "-B"
120 WRITE #1, A$, B$
```

则在磁盘上贮存形式为："TYPE", "-B"

右表是用PRINT#语句及WRITE#语句贮存数据的形式比较：

使 用 语 句	贮 存 形 式
PRINT #1, A$; B$	TYPE -B
PRINT #1, A$; "," B$	TYPE, -B
WRITE #1, A$; B$	"TYPE", "-B"

4.INPUT# 语句

(1) 格式：INPUT #‹文件号›,‹变量›

(2) 功能：该语句是从磁盘文件中读取数据。

(3) 说明：

① 语句中变量的类型要与磁盘文件中数据的类型一致。

② 读取数据时，前面的空格、回车及换行符均一一跳过去。

③ 磁盘文件中的数据须以逗号分隔。字符串数据中包含有有效空格、逗号等字符时，需以双引号(“ ”)引括。例：

```
100 INPUT #5, A$, B$, C
```

从文件号为5的顺序文件中，读取三个数据分别赋给变量A$，B$和C。

5. LINE INPUT #语句

格式：LINE INPUT #‹文件号›，‹字符串变量›

这个语句用于自顺序文件中读取一整行(最多254个字) 数据，存入字串变量中。

例 210 LINE INPUT #2, A$

从文件号为2的顺序文件中，读取一行数据赋给字符串变量A$。

6. CLOSE 语句

(1) 格式：CLOSE

CLOSE #‹文件号›

(2) 说明：

① 语句中不指出文件号时，则把所有打开的文件关闭；指出文件号时，则把指定的文

件关闭。

② 在顺序输出（"O"）方式中，CLOSE 语句是将缓冲区中的数据写入磁盘后再予关闭。

③ END语句与NEW命令能够自动关闭所有磁盘文件，但STOP语句则不能。

7．EOF语句

(1) 格式EOF（文件号）

(2) 说明：顺序文件读到最后一个数据时，EOF的值为－1。因此，可用这个语句测试读取数据时（INPUT＃），整个文件是否已被读完。

二、随机文件

1．OPEN 语句

格式：OPEN "R"，＃‹文件号›，"文件名"，记录长度

随机文件是以"R"方式打开。记录长度省略时，自动定为128字节。例如：

```
30 OPEN "R", #3, "PEN", 56
```

打开文件号为3的名为"PEN"的随机文件，记录长度为56个字节。

2．FIELD 语句

(1) 格式 FIELD ＃‹文件号›，{‹长度›AS‹变量名›}

(2) 说明：

① 该语句在随机文件缓冲区中规定各变量的长度。

② 在随机文件中，在使用GET ＃ 语句读取数据或用PUT ＃语句写入数据之前，均需用FIELD ＃语句规定读写数据的位置。例如

```
40 FIELD #3, 3 AS A$, 5 AS B$, 4 AS C$
```

规定文件号为3的随机文件中，A＄的长度为3，存放在前三个位置；B＄的长度为5，存于后面5个位置（4～8）；C＄长度为4，存于第9～12个位置。

3．LSET/RSET语句

(1) 格式 LSET‹字符串变量› = ‹字符串表达式›

RSET‹字符串变量› = ‹字符串表达式›

(2) 说明

① 在执行PUT ＃语句之前，需先使用LSET/RSET两个中的一个语句将数据从内存中依指定的形式置入随机文件缓冲区的变量中。

② 用这两个语句将数据置入缓冲区时，若超出了FIELD语句所规定的长度，则多出部分被截掉。LSET语句的含义是数据靠左存放，不足部分补以空格，超出长度时，则把右边的多出部分截掉。RSET和LSET正好相反。

③ 对于数值数据，需先用MKI＄，MKS＄或MKD＄函数将其转换成数字字符串后再用LSET或RSET放到缓冲区内。

4．MKI＄，MKS＄，MKD＄，函数

(1) 格式 MKI＄（整数常数或变量）

MKS＄（单精度常数或变量）

MKD＄（双精度常数或变量）

(2) 说明：

MKI＄将一个整数转换成一个占两个字节的字符串。

MKS$将一个单精度数转换成一个占4个字节的字串。

MKD$将一个双精度数转换成一个占8个字节的字串。

5. PUT语句

(1) 格式：PUT #〈文件号〉〔,〈记录编号〉〕

(2) 说明：

① PUT#语句用来将记录从随机文件缓冲区中写进磁盘文件。

② 记录编号指出写入位置，其值介于1～32767，省略时表示写入下一个有效位置。

6. GET#语句

(1) 格式：GET#〈文件号〉〔,记录编号〕

(2) 说明：GET#语句的作用与PUT语句正好相反，它是将记录从随机磁盘文件中取出而置于缓冲区中。

7. CVI, CVS, CVD函数

这三个函数的作用是将缓冲区内的字符串数据转换成各自原来的数值。

CVI (N$) 将占2个字节的字符串转换成整数。

CVS (N$) 将占4个字节的字符串转换成单精度数。

CVD (N$) 将占8个字节的字符串转换成双精度数。

8. LOC函数

(1) 格式：LOC (N)

(2) 说明：N为一文件号，在随机文件中，LOC给出下一个可存取的记录号；在顺序文件中，LOC在读写时，可得到文件占有磁盘段数。

9. CLOSE语句

用以关闭磁盘文件，其使用与顺序文件中的相同。

1-12-4 IBM PC BASIC的文件处理

一、文件的建立与打开

建立和打开文件都使用OPEN语句。

1. 格式：

格式1：OPEN〈文件说明〉〔FOR〈方式〉〕AS〔#〕〈文件号〉〔LEN=〈记录长〉〕

格式2：OPEN 〈方式〉,〔#〕〈文件号〉,〈文件说明〉〔,〈记录长度〉〕

2. 说明：

(1) 文件说明包括设备标识符和文件名。设备标识符是指出贮存设备的名字，如卡带录音机的名字为CASI:，第一台磁盘机为A:，第二台磁盘机为B:。文件名由1至8个字符组成。对磁盘文件，在文件名后可加文件类型，文件类型由句点(.)后跟一至三个字符组成的类型名。例如：

"B:GRADES.AUG"

(2) 格式1的方式有四种：

OUTPUT (顺序写)

例如：20 OPEN "B:GRADES.AUG" FOR OUTPUT AS #1

INPUT (顺序读)

例如：10 OPEN "CASI:PAYROLL" FOR INPUT AS #2

APPEND (顺序添加)

例如：20 OPEN "ADDRBK.DAT" FOR APPEND AS #1

缺省 （随机）

(3) 格式2的方式有三种：

O （顺序写）

如 20 OPEN "O", #1, "B:GRADES.AUG"

I （顺序读）

如 10 OPEN "I", #1, "CASI:PAYROLL"

R （随机）

(4) 两种格式的文件号均可为1～15。记录长度在随机文件中选用，缺省值为128个字节。

(5) 使用OPEN语句后，BASIC查找指定的文件，若找到则打开（如果使用 OUTPUT 方式打开则清除原文件），否则建立一个新文件。随后，BASIC为打开的文件分配一个缓冲区，并把文件号与文件名联系起来，在以后的读写等操作中，可使用文件号代替文件名。

(6) 用OUTPUT或APPEND方式打开的文件，在未关闭之前不允许再次打开。

二、文件的关闭

在文件使用完毕后，需要关闭，以释放所占的文件号并防止文件的意外损坏。关闭语句的格式为：

CLOSE〔<文件号>{, <文件号>}〕

三、顺序文件的读写

顺序文件在打开之后，关闭之前，可依照文件的打开方式进行读和写。

1. 写文件语句

WRITE #<文件号>, <表达式表>

PRINT #<文件号>, <表达式表>

2. 读文件语句

在文件建立之后，可使用读语句读出文件中的数据，常用的语句有：

INPUT #<文件号>, <变量>{, <变量>}

LINE INPUT #<文件号>, <字符串变量>

V$ = INPUT (n, #<文件号>)

后两种语句用于读入字符串数据，其中LINE INPUT #用于读入一行以<CR>结尾的字符串。而后一语句读文件中的n个字符。

四、随机文件的读写

1. FIELD语句

文件打开后，BASIC分配一个缓冲区，其长度为一个记录。接着用FIELD语句把这个缓冲区分配给名字段。其格式为：

FIELD〔#〕<文件号>{, <长度>AS<串变量>}

例如：10 OPEN "ADDRLIST.DAT"AS #1 LEN = 47

20 FIELD #1, 16 AS N$, 1 AS S$, 2 AS A$,4 AS P$,24 AS D$

20行的语句，指出了N$，S$，A$，P$和D$在缓冲区中的长度（以字节计）

2. 文件的读写

随机文件的读写语句有两个：

(1) PUT〔#〕<文件号>〔, <记录号>〕

这个语句把缓冲区的内容写到文件的指定记录中去。

(2) GET〔#〕<文件号>〔，<记录号>〕

把文件的指定记录读到缓冲区中。

这两个语句中的记录号如果缺省，则为当前记录的下一个记录。

3．LSET和RSET语句

格式为：LSET <字段变量> = <串变量>

RSET <字段变量> = <串变量>

4.转换函数

MKI$(I%) 把整型量转换成两个字节长的内码串，逆函数为CVI(A$)。

MKS$(S!) 把实型量转换成四个字节长的内码串，逆函数为CVS(A$)

MKD$(D#) 把长实型量转换成八个字节长的内码串，逆函数为CVD(A$)。

3、4所述语句和函数的使用同MBASIC的有关语句和函数。

1-12-5 CROMEMCO的文件处理

一、建立、打开和关闭文件

1．建立文件

在操作系统打开(OPEN)一个磁盘文件以前，必须用CREATE语句建立文件，其格式为：

CREATE<文件说明>

文件说明格式：d:filename.typ其中d:是驱动器的标符，filename是文件名，typ是文件类型。

CREATE语句把文件名放在指定磁盘的目录区中。

2．打开文件

(1) 格式：OPEN<文件号>，〔P1，P2〕<文件说明>

P1是可选项，对于磁盘文件，用于指定记录长度。

P2也是可选项，指定磁盘文件的存取方式

P2 = 1 只读

P2 = 2 只写

P2 = 3 读和写

缺 省 认为P2 = 3

3．关闭文件

CLOSE〔<文件号>〕

二、文件的输入和输出

1．ASCII码格式的输入和输出

(1) 输出语句

格式：PRINT <文件号>，〔P1，P2〕<表达式1,…表达式n>

(2) 输入语句

格式：INPUT <文件号>，〔P1，P2〕<变量1,…,变量n>

(3) 说明：

① 这两个语句中的P1和P2，对磁盘文件来说是可选项。P1是记录号，不给出时，是指从0号开始顺序存取。P2是字节号，省略时，指0号字节。

② PRINT语句是以ASCII码的格式把数据输出到磁盘文件。INPUT语句是把数据从磁盘文件按ASCII码格式输入内存。

2．机内代码格式的输出和输入

(1) 输出

格式：PUT‹文件号›，〔P1，P2〕‹表达式1，…，表达式n›

(2) 输入

格式：GET‹文件号›，〔P1，P2〕‹变量1，…变量n›

(3) 说明

① P1和P2的作用同1。

② PUT语句将数据以机内代码格式输出到指定的文件中去。

GET语句从文件号所指定的文件中取得数据并赋给各变量，数据不经译码，以机内代码输入主机。

本章参考书目

〔1〕《APPLESOFT》中文版

〔2〕《APPLE II MBASIC 使用手册》北方电脑公司

〔3〕《APPIE II CP/M 操作手册》(下册) 长沙市电子研究所翻印

〔4〕《苹果II微型计算机和结构化BASIC语言编程》王飞龙主编　湖北科学技术出版社

〔5〕《APPLESOFT BASIC 讲义》北京建筑工程学院

〔6〕《IBM PC BASIC与应用》天津鸿翔软件工厂

〔7〕《微型计算机IBM PC的原理与应用》张福炎 蒋新儿 李滨宇编著 南京大学出版社

〔8〕《CROMEMCO微型计算机软件资料汇编》(三) 清华大学计算中心编译　清华大学出版社

本章附录

附录一

代码	错误信息
0	NEXT循环中无FOR语句
16	语法错误
22	有RETURN，而无GOSUB语句
42	数据不够
53	非法数量
69	溢出
77	内存不够
90	无定义语句
107	下标不对
120	重定义数组
133	用0作除数
163	类型不匹配

176	字符串太长
191	公式太复杂
224	没有定义函数
254	给INPUT语句的回答不对
255	试图用Ctrl C 中断

附录二： APPLE SOFT中的保留字

&							
ABS	AND	ASC	AT	ATN			
CALL	CHRS	CLEAR	COLOR=	CONT	COS		
DATA	DFF	DEL	DIM	DRAW			
END	EXP						
FLASH	FN	FOR	FRE				
GET	GOSUB	GOTO	GR				
HCOLOR=	HGR	HGR2	HIMEM:	HLIN	HOME	HPLOT	HTAB
IF	IN#	INPUT	INT	INVERSE			
LEFTS	LEN	LET	LIST	LOAD	LOG	LOMEM:	
MIDS							
NEW	NEXT	NORMAL	NOT	NOTRACE			
ON	ONERR	OR					
PDL	PEEK	PLOT	POKE	POP	POS	PRINT	PR#
READ	RECALL	REM	RESTORE	RESUME	RETURN	RIGHTS	
	RND	ROT=	RUN				
SAYE	SCALE=	SCRN(	SGN	SHLOAD	SIN	SPC(	
	SPEED=	SOR	STEP	STOP	STORE	STRS	
TAB(	TAN	TEXT	THEN	TO	TRACE		
USR							
VAL	VLIN	VTAB					
WAIT							
XPLOT							

第二章 FORTRAN语言

2-1 概述

下面所编写的FORTRAN程序设计语言，适合于IBM PC、APPLE II、APPLE II-PLUS、CROMEMCO三种类型的微型机。CROMEMCO机使用的是FORTRANIV，除双精度和复型数据类型外，还包括所有的ANSI（美国国家标准）FORTRAN x3.9—1966（FORTRAN66）。IBM PC与APPLE II微型机使用了ANSIX 3.9—1978版本（即FORTRAN 77)的子集，而且IBM PC FORTRAN还具有ANSI X3.9-1978的全集水平的一些特性。

IBM PC与APPLE II 微型机都能将 MACRO 汇编和 PASCAL 例行子程序链接到FORTRAN程序上。

2-1-1 FORTRAN程序

一、FORTRAN程序结构

一个用FORTRAN语言设计的程序由一个或多个程序单位组成，其中只有一个是主程序，其余的为子程序。

1．程序单位

一个程序单位是一个语句序列，它描述一个算法过程，并以END语句结束。一个程序单位或者是一个主程序，或者是一个子程序。

2．主程序和子程序

主程序用PROGRAM语句或除SUBROUTINE和FUNCTION语句以外的任何其他语句开头，用END语句结束。

子程序用SUBROUTINE或FUNCTION语句开始，用END语句结尾。

3．语句

FORTRAN语句分为两类：可执行语句和非执行语句。可执行语句指定程序实现计算、输入、输出。非执行语句把变量类型、数据初值、输入输出格式等信息通知编译程序。

FORTRAN源程序的一个语句可占一行或若干行。一行有80列，即有80个字符位置。语句可从第7列起写到72列。如果一个语句在一行的7—72列写不下，或者不愿占满7～72列，可延续下一行或下几行，但所占的区域仍然为7～72列。IBM PC FORTRAN允许有九个续行。每个续行必须在第6列的位置上填写除0或空(null)以外的字符集中任何符号。

4．行

FORTRAN程序可分为注释行、起始行和续行。

(1) 注释行

注释行可在程序任何地方出现，不影响程序执行。注释行的标识为：

① 第一列上写上C(或c)；

② 第一列上写上 * 。

在大多数计算机上，程序行的第一列只要写字母C，FORTRAN编译就认为是注释行。在C字母后面可写要注明的内容，或者为了程序书写清晰起见，在C字母后全为空。

(2) 起始行

一个起始行的第1～5列必须是空格或者是标号。第6列是空格或者是0。起始行是语句的起点。

(3) 续行

续行的标记是在第6列上写上非0或非空的任意字符。为语句提供更多的空间（见本节3-语句）。

5．列

前面已讲到每行可有80列，每列写一个字符，共分四个区域：

(1) 语句标号区

1～5列为语句标号区，用于写标号、注释标识（前面提到第一列写上C），也可为空白。

(2) 续行指示区

第6列用于写续行指示符，或者为空白。

(3) 语句区

第7～72列为语句区。语句正文写入此区。

(4) 序号区

第73～80列为序号区。作为程序员标识程序编号用的区域。

由上面所述，在编写FORTRAN程序时，必须严格遵循行与列中的规定。

二、FORTRAN程序单元结构

FORTRAN程序的基本单元是行。一个单位中，语句的出现也有严格顺序。其顺序如下表：

<table>
<tr><td rowspan="6">注 释 行</td><td colspan="2">PROGRAM，FUNCTION或SUBROUTINE语句</td></tr>
<tr><td rowspan="5">FORMAT
语 句</td><td>IMPLICIT语句</td></tr>
<tr><td>其他说明语句</td></tr>
<tr><td>DATA语句</td></tr>
<tr><td>语句函数语句</td></tr>
<tr><td>可执行语句</td></tr>
<tr><td colspan="3">END语句</td></tr>
</table>

由上表说明，在一个程序单位中，无论是主程序还是子程序，语句的出现必须符合下面的规则：

1．如果使用PROGRAM，FUNCTION或SUBROUTINE语句中其一时，必须是一个程序单位的第一个语句。

2．FORMAT语句只允许出现在PROGRAM，FUNCTION或SUBROUTINE 语句之后的任何地方。

3．所有说明语句必须在DATA语句，函数语句和可执行语句之前。

4．IMPLICIT语句必须在其他说明语句之前。

三、FORTRAN字符集

FORTRAN字符集有：

字母：由A到Z和a到z共52个大小写字母。

数字：0到9。

专用字符：除字母与数字之外的可打印的ASCII字符。

对于FORTRAN程序，除了字符型常数和文字型字段外，把小写和大写字母看成相同的字符来解释。例如：用户用下面定义的四种名字，对FORTRAN系统完全一样。

BACED Baced bACEd bAceD

FORTRAN字符集的排列顺序就是ASCII码的排序。

四、一个简单的FORTRAN程序例

(1) 主程序

```
1 2 3 4 5 6 7
      PROGRAM  LAWCOS
C THE SIDE  OF  THE  TRIANGLE
      A1 = 1.0
      B1 = 2.0
5     ALFA1 = 0.2
      A2 = 2.0
      B2 = 4.0
      ALFA2 = 0.1
      C1 = C(A1,B1,ALFA1)
      C2 = C(A2,B2,ALFA2)
      WRITE(10,10)C1,C2
10    FORMAT(1X,2F10.4)
      STOP
      END
```

(2) 子程序：

```
1 2 3 4 5 6 7
C     FUNCTION  SUBPROGRAM
      FUNCTION  C(A,,B,ALFA)
      D = A*A + B*B - 2.0*A*B*cos (ALFA)
      C = SQRT (D)
      RETURN
      END
```

上边的程序由一个主程序和一个子程序两部分组成。主程序用PROGRAM语句后边接程序名作为起始。然后第一列写C的一行为注释行。从第3行到第8行是给变量赋值部分，依次把1.0，2.0，0.2，2.0，4.0，0.1六个数值赋给A1，B1，ALFA1，A2，B2，ALFA2六个变量。第9行和第10行是调用子程序赋值语句。在执行第9行时，将A1，B1，ALFA1的值传送给子程序C中，用以代替A，B和ALFA。并将子程序求出的C值带回主程序赋予C1。同样第10行的作用是通过子程序求出C2值。子程序中的RETURN语句表示执行完子程序后返回主程序。主程序中第11和12行为输出部分，它将C1和C2值按第12行的格式输出。最后停止运行。子程序是根据公式$C^2 = A^2 + B^2 - 2AB\cos\alpha$计算C值，子程序的第4行SQRT是求出平方根函数。

2-1-2 FORTRAN符号名 FORTRAN程序中的变量、数组、函数和子程序是用符号名表示的。符号名由1～5字母数字串组成，第一个必须是字母(A～Z)，空格可以出现在符号名中间，但没有什么意义。

FORTRAN符号名，尽管允许使用任何有效的字符序列，编译既承认语言中的关键字，又不限制用户定义名字使用。例如，用户可以在程序中用IF，READ，或GOTO命名变量、数组、函数和子程序。但是，我们不提倡这种做法。

一、FORTRAN符号名的作用域

一般符号名的作用域，可分为全程的和局部的作用域，但也有的例外，符号名可按某一定义范围使用，同一符号名在不同的作用域也可有不同的定义。

带有全程作用域的符号名可以用于一个以上的程序单位（一个子程序、函数或主程序），在同一程序单位中，只能作为单一相同类型的符号名使用。所有子程序、函数子程序和公用符号名，象程序的名字一样有一个全程作用域。因此，一个函数子程序不能具有和另一个子例子程序或一个公用区相同的名字。同样，在同一程序中不能有二个函数子程序具有相同的名字。

虽然 IBM FORTRAN 没有保留字、但使用某些符号名要和库程序名相同时，就有可能导致连接错误，因此下面这些符号名字不能做为全程符号名使用：

ABS	COS	ICLRER	MOVESL	SMULOK
ACOS	COSH	IDIM2	MOVESR	SQRT
AINT	DIM4	IDIM4	NINT2	TAN
ALOG	EXP	IGETER	NINT4	TANH
ALOG10	FILLC	ISIGN2	PUTCH	TNSR
AMOD4	FILLSC	ISIGN4	PUTSTR	VADDOK
ANINT	FTRANS	LOCKED	REAC	UM460K
ASIN	GETCH	MOD2	SADDOK	UMULOK
ATAN	GETSTR	MOD4	SIGN4	UNLOCK
ATAN2	IABS2	MOVEL	SIN	UTLR
CNVR	IABS4	MOVER	SINH	

注意：除了上面的符号之外，任何以QQ结束的6个字符的符号名也不能使用。

变量、数组、形参和语句函数名也都有其局部作用域。

在某一程序单位中，全程作用域的符号名是明确的，但局部作用域的符号名可在其他不同或相同意义的程序块中使用，其符号名不要求具有同一含义。

作用域的例外规定：所给的公用数据块名，可能在出现“局部”符号名的相同程序块中引用一个“全程”的公用区符号名，因为公用区符号名总要由斜线括起来，如/JobN1，编译程序可与一般符号名区别出来。

作用域还有另一种例外规定：语句函数参数的作用域完全限定在组成那个语句函数的单一语句中。

二、隐含FORTRAN符号名

在一个程序中，遇到可执行语句中有用户没有定义的符号名时，编译程序识别该符号名，并以它的第一个字母来判断其类型，判断规则是：变量名若以I、J、K、L、M或N字母为开头、则认为是整型变量，而其他所有的则认为是实型变量。当然，这些隐含的规则可以被 IMPLICIT 语句所否定。

如果一个隐含的符号名在一个函数调用中，或者遇到CALL语句对象的名时，其函数类型的判断如同判断一个变量一样。CALL语句对象的名，假如对象是一个子程序，或者是一个函数，编译程序都将在新建的符号名表中来协调确定。如果一但发生矛盾，如前后定义子程序名、函数名相同时，将输出一个错误信息。

2-1-3 数据类型 在 FORTRAN 程序中经常要用到常数。常数是一个在程序执行过程不变化的已知值。IBM FORTRAN 有四种数据类型，如下表所示：

类 型	说 明 语 句	要占存贮单元(字节)
逻辑型	LOGICAL	2或4
	LOGICAL*2	2
	LOGICAL*4	4
整 型	INTEGER	2或4
	INTEGER*2	2
	INTEGER*4	4
字符型	CHARACTER	1
	CHARACTER*n	n
实 型	REAL	4
	REAL*4	4

对于无格式文件，FORTRAN数据类型在存贮器中所占字节数如左表右侧部分。

一、整型

整型数据类型是包括零或正或负的整数。一个整型变量占用2个或4个字节的存贮单元。一个整型可以说明为INTEGER*2，INTEGE*4或INTEGER。第一种和第二种分别占用2个和4个字节整型值。第三种规定为2个或者为4个字节的整型值。具体将按照$STORAGE元命令来装配（缺省是4个字节）。

一个2字节整型量，可以是－32767到32767范围内的任何值。一个4字节整型量可以包含－2,147,483,647到2,147,483,647范围内的值。

整常数是一位或更多位的正、负的十进制数字，不允许小数点出现在整常数中。例如：

137　　+248　　－478　　0

00012　　3758　　－4726

不能用带小数点数12.或23.0表示整数

注意：APPLE II 和 CROMEMCO 微型机的 FORTRAN 整型变量只占二个字节的存贮单元，可以是－32767到32767之间的任一值。

二、实型

实型数据是由实数组成，实数在FORTRAN中为带有小数点或带有指数的数。若一个数据、变量、数组元素等具有实型时，在机内处理为实数的近似值。一个实型数据占用4个字节存贮单元，并用REAL或REAL*4说明。

实型常数有两种表示形式：

1.指数形式　例如：

+1.0000E－2　　1.E－2　　1E－2

+0.01　　100.0E－4　　.0001E+2

全部都表示同一实数：$\frac{1}{100}$

(1) IBM PC FORTRAN 中单精度实数的数值范围：

正数范围　3.0E－39到1.7E＋38

负数范围　－1.7E＋38到－3.0E－39

(2) APPLE的FORTRAN中单精度实数的数值范围：

正数范围　5.8E－39　到　1.7E＋38

负数范围　－1.7E＋38　到　－5.8E－39

(3) CROMEMCO机的 FORTRAN 单精度实数范围：

在10^{-38}～10^{38}之间，精确到7位有效数字。

2.小数形式　如：

-135.76	+135.76	135.76
-238.	+238.	238.0
-0.475	+.475	0.475

实型常数小数形式写法由一个整数部，一个小数点，一个小数尾数等三部分所组成。+.475与0.475含义是相同的。2.0与2.含义也是相同的。

三、逻辑型

逻辑型数据由真（TRUE）和假(FALSE)两个值所组成。一个逻辑型变量占用2个或4个字节存贮单元。

在IBM PC机上 FORTRAN 的逻辑变量用LOGICAL（占2个或4个字节)、LOGICAL*2（占2个字节)、LOGICAL*4（占4个字节）来说明。具体按照$STORAGE来装配（缺省的是4个字节)。

有两种逻辑常数.TRUE.和.FALSE.表示真与假两个值。.TRUE.内部表示为一个最低有效位是1的其余是0的数。而.FALSE.的内部表示为全0的数。除此之外，如果一个逻辑型常数包含任意其他值的数、它的逻辑值是无法定义的。

在APPLE II 和 CROMEMCO 微型机上 逻辑常数.TRUE.和.FALSE.与IBM PC机一样是真与假两个值。但其逻辑值作为一字节的带符号整数使用，范围在-128到+127。

四、字符型

字符型数据是 ASCII 字符集序列，一个字符型数据的长度等于字符序列中的字符的个数，字符常数或变量的长度是固定的，必须是1—127之间的数。

在序列中，每一个字符都占用一个字节，如果长度是奇数，就加一个字节。字符型变量总是定位在字边界上，其中空格是有效的。

一个字符常数是由一对撇号括起来的一个字符或字符串。字符常数中的撇号要求连续，中间不要有空格。字符常数的长度等于撇号中字符个数。中间多撇号只算一个单引号。例如：

```
'AH''''HOLP'
'A CHARACTER''''
```

AH的'''和CHRACTER的''''只表示一个单引号。被'和''''括起来的字符长度为11。

FORTRAN允许源程序由第七列写到72列共有66个字符位置。如字符常数在本行没有写满，又在续行继续写时，其空仍作为长度计算。如：

```
200     CHAR = 'ABC
       / XDEF'
```

被' '括起来的字符常数的长度为64即(66-6+4)。

五、Hollerith 常数

CROMEMCO、APPLE II 和 IBM PC 微型机所使用的FORTRAN还有 Hollerith 常数。它是字符集（ASCII 符号表）中任意数目的字符组成的串。所有字符，包括空格都有效。Hollerith 数据串中每一字符需要一个字节的存贮单元。

2-1-4 变量和数组

一、变量

变量是在程序执行过程中其值可以变化的量。在程序中出现一个变量，就在计算机的存贮器中相应地为它开辟一定的存贮单元，作为存放变量对应的数据之用。

每一个变量都需有一个在自己存在的程序块中唯一的符号名，即在同一程序块内变量不

能同名。不同类型的数据存放所占单元也不同，所以必为变量指明类型。在2-1-3节中已介绍数据有四种类型十种形式。其中有关实型及整型变量的类型指定，有两种说明方法：

1.隐式说明

(1) 以符号名的首字母确定类型

FORTRAN 规定，凡是没有附加说明的符号名，以I，J，K，L，M，N六个字母之一开头的符号名的变量为整型，以其它字母(A～H，O～Z) 开头的为实型变量。如：

I135，MAXC，LI3，N75为整型变量，而AVAR，XMAX，CRAD，F137 为实型变量。

(2) 使用 IMPLICIT 隐含说明语句

如果想否定(1) 点的隐式说明方法，即否定首字母所说明的类型，可用 IMPLICIT语句来说明。如：

IMPLICIT INTEGER(A，C，O，Z)，REAL(I,M)

这就表明以A，C，O，Z打头变量都定义为整型，以I，M打头变量都定义为实型，经此语句说明之后，在本程序块中就有：

A，CD12，ONL3，Z2均为整型变量，

而IN13，MX7为实型变量。

2.显式说明

变量类型是整型还是实型用类型说明语句来定义。如：

INTEGER*4 X7，A22

表示在本程序块中，变量X7，A22为整型。

REAL K，M1N7

表示在本程序块中，变量K，M1N7为实型。

至于变量的其他类型：逻辑类型和字符类型的定义还是用说明语句。请参考2-2-1部分。

二、数组

在许多应用中，常常会遇到要存贮和处理大量数据，FORTRAN 的数组为之提供了方便。数组是同类型数据的组合，这些数据存放在与一个符号名相关联的一组存贮单元中。这个符号名称之为数组名。数组中的成员称之为数组元素。

使用数组时，首先必须定义数组的维数和元素的个数。在 FORTRAN 中，可以用 DIMENSION 语句来定义，也可在类型说明中定义。数组的引用是用数组名引用整个数组，或者用数组名后面跟下标表达式引用某一特定的数组元素。其格式如下：

arr(sub[，sub]…)

arr：为数组名，sub 为一个下标表达式。

给一个数组第四个元素赋3.8值，可写成：

```
C ASSIGN THE VALUE 3.8 TO 4TH ELEMENT OF ARRAY
    A(4,1,)=3.8
```

下标表达式是在选择数组的一个特定元素时使用的整型表达式，数组说明符的下标表达式的个数必须和数组维数相附，其值是在1和对应于数组所给的上界之间。

2-1-5 表达式

FORTRAN 有四种表达式：算术表达式，字符表达式，关系表达式和逻辑表达式。

一、算术表达式

算术表达式能够产生一个整型或实型的值，最简单的算术表达式的形式有：无符号整数或实数，整型或实型变量名，整型或实型数组元素名，整型或实型函数名。

在算术表达式中出现一个变量名或数组元素名其值必须事先定义过。

一般的算术表达式是由括号和算术符与最简单的算术表达式形式所组成。

1.算符和运算优先级

算术表达式中使用五种算符及运算的优先级如左边表所示：

算 符	运 算	优 先 级
**	乘 方	最 高 级
/	除	中 间 级
*	乘	中 间 级
-	减或非	最 低 级
+	加	最 低 级

表中五种算符都具有两目算符功能，并位于表达式中间，其中＋和－也可以是单目算符，位于操作数之前。同级的算符，左边结合先算，然而乘方运算除外，右边先算，

计算数据类型	等级次序
REAL	最高级
INTEGER*4	中间级
INTEGER*2	最低级

如：

A/B*C和(A/B)*C是等同的

A**B**C和A**(B**C)是等同的

FORTRAN 禁止二个算符连续出现，如：

A**－C是不允许的，而A**(－C)则允许。

单一算符的负号是最低级优先，于是

－A*C可以解释为－(A*C)

在表达式中，括号可以用来控制求值运算结合次序。

2.整数除法

整数相除得值如是实数值的商，要舍去小数部分变为整数，如：9/4值为2，(－27)/13得值为－2，9/10值为0。

3.运算结果类型

当一个算术表达式里所有操作数据都是相同类型时，表达式产生的值也只有同一类型。当运算的数据类型不同时，表达式产生的值可由上右表中等级次序确定数据类型。

当表达式中有两种不同类型数值运算时，结果值的类型为最高级数据类型。如一个整型和实型运算结果为实型值。

表达式的数据类型是表达式的最后求值运算所得到的数据类型。

实型量与整型量运算时，整形量先转换成实型量，将小数部分换成等于 0 的实数，再求其结果。如：

B＝M/C

M先转换成一个实型，执行实型量与实型量相除。

当表达式含有混合的数据类型时，整型和实型运算按照算符的优先级顺序进行。如：

y＝X*(I＋M)

I和M是整型量相加，其和再转换成实型数，然后和X实型量相乘。

请注意：IBM FORTRAN 不校验整数溢出，如果发生整数越过所允许的范围，结果为随机数，是不确定的。

二、字符表达式

一个字符表达式操作结果是一个字符类型的值，字符表达式的形式是：

1.字符常数；

2.字符数组元素名；

3.任何用括号括起来的字符表达式。

字符表达式中没有运算符。

三、关系表达式

关系表达式是比较两个算术值或者比较两个字符值，但算术值与字符值不能相比较，表达式的结果为逻辑值。

关系表达式可用下表中的算符来进行运算：

关系符	所表示的运算
•LT•	小于
•LE•	小于等于
•EQ•	等于
•NE•	不等于
•GT•	大于
•GE•	大于等于

所有运算符都要在两个字符中间出现，例如：y•GT•2，M•EQ•7

上边的表达式是有效的，值得明确的是在关系表达式中没有相对的优先级和结合性，因此

A•LT•B•NE•C

是无效的。

带有字符型运算量的关系表达式，是按着 ASCII 码的排列顺序序号值来比较他们的运算量。出现早的就小于出现晚的字符；如果长度不同的字符进行关系运算时，短的运算量补空格直到和长的运算量相等长度后再作比较。

四、逻辑表达式

逻辑表达式只能产生逻辑型的值(真或假)，最简单形式的逻辑表达式有：

1.逻辑型常数；

2.逻辑型变量名；

3.逻辑型函数名；

4.关系表达式。

其他的逻辑表达式，是以上边列举的四种最简单形式，用括号和下边的左表中所给出的逻辑运算符组合起来：

左表

逻辑运算符	运　算	优先级
•NOT•	非	最高级
•AND•	与	中间级
•OR•	或	最低级

右表

算　符	优先级
算　术	最高级
关　系	中间级
逻　辑	最低级

表中表明三种逻辑运算符和运算的优先级•AND•和•OR•运算符是两目算符，应当出现在逻辑表达式中二个运算量之间，•NOT•算符是一目算符，它优先于其他逻辑算符。相同级运算从左开始，例如：

(1) MXC · AND · NXB · AND · H7相当于(MXC · AND · NXB) · AND · H7

(2) ·NOT · A · OR · B · AND · C相当于(·NOT · A) · OR · (B · AND · C)

值得注意的是两个 · NOT · 不能相邻，但A · AND · NOT · B是允许的。逻辑运算在数学上的标准定义是：·OR·表示不唯一性，如：

· TRUE··OR · A 不管A在逻辑型取什么值，其运算结果为·TRUE·。

五、表达式优选级

当算术、关系、逻辑的运算符在同一表达式里存在时，其相对优先级如158页右表：

六、表达式求值规律

表达式中要引用的变量、数组元素或函数必须在引用前定义。整型必须用一个算术值定义，不能用 ASSIGN 语句对整型变量赋值。

在实数范围内有一些运算是无意义的，例如：被0除，是禁止的，还有0的0次幂或0的负次幂、负数的实数次幂是不允许的。

2-2 语句

FORTRAN 语句分成可执行语句和非执行语句两大类，每类又可分若干种。IBM PC, APPLF II 和 CROMEMCO 的 FORTRAN 语言的语句及其分类如下：

- 可执行语句
  - 赋值语句
    - 计算型赋值语句
    - 标号赋值语句
  - 控制语句
    - 无条件GOTO 无条件转移语句
    - 计算GOTO 计算转移语句
    - 赋值GOTO 赋值转移语句
    - 算术IF 算术条件转移语句
    - 逻辑IF 逻辑条件转移语句
    - 块IF 块条件转移语句
    - DO 循环语句
    - CONTINUE 继续语句
    - CALL 调用子程序语句
    - RETURN 返回语句
    - PAUSE 暂停语句
    - STOP 停语句
    - END 结束语句
  - 输入/出语句
    - OPEN 打开文件语句
    - CLOSE 关闭文件语句
    - READ 读语句
    - WRITE 写语句
    - ENDFILE 文件结束语句
    - REWIND 反绕语句
    - BACKSPACE 回退语句
- 非执行语句
  - 说明语句
    - TYPE 类型说明语句
    - IMPLICIT 隐含语句
    - DIMENSION 数组定维数语句
    - COMMON 公用区说明语句
    - EQUIVALENCE 等价语句
    - EXTERNAL 外部语句
    - INTRINSIC 内部语句
    - DATA 数据初值语句
    - PROGRAM 程序语句
    - SAVE 保留语句
  - 子程序语句
    - SUBROUTINE 子程序定义语句
    - FUNCTION 函数定义语句
    - BLOCK DATA 数据块语句
  - 输入/出格式语句
    - FORMAT 格式语句

2-2-1 说明语句 用户可以用上面提到的十种说明语句来定义变量名，数组名，函数名的属性和特性。说明语句属于非执行语句。

在说明语句中，IMPLICIT 语句在一个程序单位中，必须在其余的说明语句之前，其余说明语句可以以任何顺序出现。说明语句又必须在所有的可执行语句之前。

2-2-2 TYPE(类型)语句 用途：类型语句是用来确定用户定义的变量、数值、外部函

数、语句函数的符号名的类型。类型语句可以重新定义或取代一个符号名的隐含类型；也可规定数组的维数。

格式：TYPE V[,V]

说明：TYPE是2-1-3节中的四种数据类型中的十个说明语句之一。V是变量、数组、函数、函数子程序或一组数组说明的符号名。在一个程序单位中，一个符号名只能由一个类型语句明确规定它的类型。

使用规则：

1.类型说明语句必须在所有可执行语句之前；

2.符号名的数据类型只能被说明一次；

3.类型说明语句禁止用标号；

4.可以在数组名上附加数组说明符来说明一个数组。

V被指定是字符类型时，字符类型符号名可紧跟一个*n来说明字符型的字符长度为n个。

例如：
```
CHARACTER*10 NAME
CHARACTER*80 CITY
```
或用CHARACTER NAME*10, CITY*80

又如：
```
INTEGER COUNT, MAX(4,4), SUM
REAL MA, IA
LOGICAL SW
```
经上面的说明之后，再在程序中出现的NAME, CITY 字符型的长度分别为10和80 个字符，而COUNT, MAX(4,4), SUM为整型。MA 和IA却为实型，以M与I字母打头隐含说明为整型就不起作用了。

2-2-3 IMPLICIT语句

用途：IMPLICIT 语句用来定义用户要否定以打头字母隐含说明类型所要说明的符号名。

格式：IMPLICIT TYPE(a[,a]…)[,TYPE(a[,a]…)]…

说明：TYPE 是2-1-3中的整数类型或实数类型。

a是一个单独的字母或字母串，字母串如是连续排序时，可简写成第一字母与最后字母用“—”隔开。例如：
```
IMPLICIT REAL(I—M), INTEGER(B,D,Z)
```
在程序单位中出现 I, J, K, L, M 打头的符号名变成了实型，B, D, Z字母打头的符号名反倒变成了整型。

IMPLICIT语句必须在其他说明语句之前，并在同一程序块中，IMPLICIT 语句不能对同一字母定义两次以上。

2-2-4 COMMON语句

用途：COMMON 语句用来向两个或两个以上的程序单位提供共享存贮空间，这些程序单位可共用同一数据而不需要变元来传递。

格式：COMMON[/Cname/]nlist[[,]/Cname/list]…

说明：Cname 是公用区名。如没有Cname时为无名公用区。在一个可执行程序中，无名公用区只能有一个。nlist 是由变量名、数组名、数组元素说明符组成的表列，各个元素由逗

点分开，哑元名和函数名不能出现在 COMMON 语句中。nlist 的所有变量和数组都要在公用区中说明。

公用区的大小等于其所有元素在存贮区所占用存贮单元的数目。若不同的程序单位引用同一个共用区时，其不同的程序单位指定的共用区的命名符个数必须相同。如：

在主程序中写：

```
PROGRAM MYN
COMMON I, J, X, K(10), Y(3)
I=1
CALL SUB1
⋮
END
```

而在子程中写：

```
SUBROUTINE SUB1
COMMON IOX, JOR, XO, K(10), y(3)
IF(IOX·NE·1)
⋮
END
```

这说明主程序的 I, J, X, K(10)与子程序的 IOX, JOR, XO, K(10)按先后顺序对应共享同一存贮单元。即每个单元存有相同数值。其关系如下表：

I	J	X	K(10)	Y(3)
IOX	JOR	XO	K(10)	Y(3)

若两个程序单位引用字符常数的同名公用区，则对应的字符变量的长度必须一致。如果两个变量引用同一存贮单元，则称两个变量结合。不同的程序单位中，无名公用区允许有不同的长度。

2-2-5 DIMENSION 语句

用途：维数语句指定有几个下标，即数组是几维的；由下标值和维数定数组元素个数。

格式：DIMENSION name(d[,d[,d]])[, name(d[, d[,d]])]…

说明：name 是用户定义的数组名，d 是数组下标。数组的维数就是下标 d 的个数，最大维数是三。维数说明符允许是：正整数，对应用户命名的非数组整型哑元名，星号。

维数说明符规定了维数的界限，最低是 1。d 值的大小与个数，决定数组的元素个数。

如果维数说明符是整变元，则确定维数的大小等于子程序运行时变元的初值。此时，数组称为可调数组。

若维数说明符是星号，则数组维数上界不定，是待定数组。

所有可调与待定数组的维数是某程序单位的哑元。同时星号只可为数组说明符的最后维数。下面举几个例子：

DIMENSION S(10), A(3,7), B(2,1,8)

DIMENSION 语句，定义了名为 S, A, B 三个数组，S 为 10 个元素的一维数组，A 为 3×7 个元素的二维数组，B 为 2×1×8 个元素的三维数组。

2-2-6 EQUIVALENCE 语句

用途：等价语句能够使同一程序块中的两个或更多个变量（数组或数组元素）共用同一存贮单元。

格式：EQUIVALENCE (nlist) [, (nlist)]…

说明：nlist是由至少两个元素组成的表。这些元素不能是哑元，元素可以是变量名、数组名或数组元素名。因为 EQUIVALENCE 语句规定了只能在同一程序块中起作用，所以主程序和子程序或子程序和子程序之间不同变量、数组或数组元素不能用等价来指定共用的存贮单元。

例：
```
REAL R, S(10), B, A(7)
EQUIVALENCE (R, S(4)), (B, A(5))
```

实型变量和实型数组元素的 R 和 S(4) 及 B 和 A(5) 分别共用同一存贮区，虽然 R 与 S(4) 及 B 和 A(5) 二者可能具有不同的数值，但因在程序出现有先有后，并不影响使用同一存贮单元。这是节省存贮区的一种途径。

等价语句还规定连续的数组只能顺序共享存贮单元。并且 EQUIVALENCE 语句规定只能在同一等价语句中符号名出现一次。因此下边的使用例子是错误的：

```
EQUIVALENCE (R, S(10)), R, S(7)
```

及

```
REAL R(10), S(10)
EQUIVALENCE (R(1), S(5)), (R(5), S(7))
```

2-2-7 EXTERNAL 语句

用途：外部语句用来识别用户定义的外部子程序或外部函数名。

格式：EXTERNAL name [, name]…

说明：name 是外部子程序或外部函数名。

在外部语句中出现的子程序或函数名，即定义为外部过程。语句函数名不能在外部语句中出现。若内部函数名在外部语句中出现时，则变成外部过程，且对应的内部函数不能再被本程序单位所调用。在给定的程序块中，用户命名符只能在外部语句中出现一次。如：

主程序：

```
EXTERNAL SIN, COS
READ (11, 1) A, B
CALL LT (A, SIN, U)
CALL LT (B, COS, V)
 ⋮
END
```

子程序：

```
SUBROUTINE LT (X, F, Y
Y=F(X)
IF(Y.GT.0.0) Y=Y**0.5
RETURN
END
```

2-2-8 INTRINSIC 语句

用途：内部语句用于指定内部函数名。

格式：INTRINSIC name [, name]…

说明：name 是内部函数名。每个函数名只允许在 INTRINSIC 语句出现一次，而且不

能在 EXTERNAL 语句中出现。因此在 INTRINSIC 语句中出现的所有函数名都必须是系统定义的内部函数。这些内部函数请参考2-5-2节的三。

内部函数名，如类型转换 (INT, IFIX, IDINT, FLOAT, REAL, ICHAR, CHAR)，字符顺序关系(LGE, LGT, LLE, LLT)，选取最大(小)值 (MAX0, AMAX1, AMAX0, MAX1, MIN0, AMIN1, MIN1) 及 EOF 等等，都不能作为实变元。

例：INTRINSIC SIN, COSIN

```
X = CALC2 (SIN, COSIN)
      ⋮
```

2-2-9 SAVE 语句

用途：在从定义过的一个公共块的过程返回后，SAVE 语句用于保留此公共块的定义。

格式：SAVE/name/[, /name/]…

说明：name 是公共块名。在子程序或函数内，SAVE 语句指定的公共块不能在子程序或函数的出口处，否则，会变得不确定。

在 IBM FORTRAN 中，由于公共块是静态分配自动保存的，因此，SAVE 语句是无效的。

2-2-10 DATA 语句

用途：DATA 语句是给变量、数组元素赋初值。它位于所有说明语句之后和在所有的语句函数或可执行语句之前。

格式：DATA nlist/clist/[, nlist/clist/]…

说明：nlist 是变量、数组元素表。clist 可为常数，或由整数作为重复因子加上星号开头的常数。

例：DATA B/1.5/, c/2.0/, D/4.7/

又如：DATA A, B, C/3*2.0/

再如：DATA S(5)/2.4/, LTST(4)/Heip/

DATA语句中不允许有形参、公用区变量、函数出现，只允许变量和数组元素出现。

clist 表中的值个数和类型必须与 nlist表中变量个数和类型保持一致。

若是字符类型变量或数组，对应字符的长度必须小于或等于变量或数组元素所定义的长度。若字符常数长度比定义长度短，则左对齐，右边增加空格补齐。

2-2-11 PROGRAM 语句

用途：PROGRAM 语句标识一个程序单位为主程序，并给程序命名。

格式：PROGRAM pname

说明：pname 是用户定义的主程序名。

pname是全程量，因此pname不能与任何局部量名相同。如

```
PROGRAM  CAL
INTEGER  X7, TAX
  ⋮
END
```

2-3 赋值语句

赋值语句是给某个变量或数组元素赋一个值。赋值语句可分为计算型和标号型两类。

2-3-1 计算型赋值语句

用途：计算型赋值语句是对表达式求值，并将所求得的值赋给赋值号左边的变量或数组元素。

格式：V=e

说明：V是变量名或数组元素名，=为赋值号，e是表达式。

变量、数组元素必须与表达式的类型相一致。它们必须是数值型、逻辑型或字符型。由以上三种情况下，计算型赋值语句可分为：

计算型赋值语句{算术赋值语句 / 逻辑赋值语句 / 字符赋值语句

若算术赋值语句中变量的类型与表达式值的类型不同，表达式的值的类型将会自动地转换成变量的类型。如下表所示：

变量V的类型	表达式e	
	整型	实型
整型	将E赋给V。	将E转换成整型(舍去小数)，并赋给V。
实型	将E转换成实型(加上小数点)，赋给V。	将E赋给V。

在一个字符赋值语句中，表达式的长度与变量的长度不一致时，则要调节一致。表达式的字符值短，则于值的右边填充足够的空格；若表达式的字符值长时，则右边的字符删去，与变量长取得一致。逻辑表达式不存在这样的问题。值得注意的是 INTEGER*4 类型的表达式不能赋值给 INTEGER*2 类型的变量。

2-3-2 ASSIGN 语句

用途：ASSIGN 语句是给格式语句或语句的标号赋给一个整型变量。

格式：ASSIGN Label TO Var

说明：Label 是语句标号和格式语句标号。Var 为整型变量。如：

```
      ASSIGN 10 TO INS
      GOTO INS
   10 READ A, B
```

把标号10赋给变量 INS，在执行GOTO语句转移到标号为10的 READ读语句。

2-4 控制语句

控制语句用来控制程序中语句执行的顺序，因此可以说是用于控制程序走向的。

在这一小节中，把控制语句中的 CALL 语句和 RETURN 语句留到后边叙述外，将其他十一个控制语句在此节中介绍。

由于控制语句功能上的原因，能否恰当运用控制语句，常常能够决定程序结构的优劣。如果能按着结构程序设计方法进行编制程序，在使用控制语句的程序块中具有一个入口，一个出口，程序块之间是由上至下的顺序结构，就能增加程序的正确性、易读性、易维护性。

2-4-1 无条件GO TO语句

用途：无条件GO TO语句使程序无条件转移到标号为S的语句。

格式：GO TO S

说明：S是可执行语句的语句标号，S标号语句与GO TO语句应处在同一个程序单位内。不允许从外部跳到循环DO语句、块IF语句内（在扩大DO循环范围的特殊功能下，可以允许跳到DO块内）。

例：

```
          COUNT = 0
          DO 10  I = 1, 10
          COUNT = COUNT + 1
          IF(COUNT.GE.10) GO TO 20
10        CONTINUE
20        WRITE (9, 100) COUNT
          STOP
100       FORMAT (1X, 'COUNT = ', I5)
          END
```

2-4-2 计算GO TO语句

用途：计算GO TO语句指定转移到标号表中的第I个标号的语句。

格式：GO TO(S[, S]…)[,]I

说明：S是本程序单位中的某个可执行语句的标号，I是在标号表中的标号所处的位置次序值。同一个标号可在标号表中重复出现。

若是标号表中有n个标号，又有1≥I≥n时，计算GO TO语句等价于 CONTINUE 语句，则下个执行语句由标号表中的第I标号指定。不允许从DO语句，块IF语句外，跳到这些语句块内(在特定的扩展的情况下，是允许从块外跳入DO块中)。如：

```
          I = 1
          GO TO(10, 20, 30),I
            ⋮
10        STOP'I = 1'
            ⋮
20        STOP'I = 2'
            ⋮
30        STOP'I = 3'
```

2-4-3 赋值GO TO语句

用途：赋值GO TO语句是指定转移到赋值I标号的可执行语句。

格式：GO TO I[[,](S[, S]…)]

说明：I是整型变量名。S是同一程序块中的转向可执行语句的标号。同一语句标号可以在标号表中重复多次出现。不允许从DO语句，块 IF 语句外，跳到这些语句块内（在特殊扩充循环语句的情况，允许从块外跳入DO块内）。如：

```
          ASSIGN  10  TO  I
          GO TO  I, (20, 30, 10)
            ⋮
10        WRITE (*, 100)
```

```
    ⋮
```

标号10赋值给I，再由标号表中寻找I代表的标号，然后转移到语句标号为10的可执行语句。

2-4-4 算术IF语句

用途：算术IF语句是根据表达式求得的值，去选择指定标号的可执行语句。

格式：IF(e) S_1，S_2，S_3

说明：e是整型或实型表达式。S_1，S_2，S_3是可执行语句的标号，它们与算术IF语句在同一程序单位中。三个标号，允许有相同标号存在。

当e＜0时，执行语句S_1

当e＝0时，执行语句S_2

当e＞0时，执行语句S_3

不允许从DO，块IF语句之外，转进块中（在扩展的DO循环的特殊情况下，允许转入DO块内）。如：

```
          IF (I-75) 10, 20, 30
10        WRITE (*, 500)
          STOP
20        WRITE (*, 600)
          STOP
30        WRITE (*, 700)
          STOP
500       FORMAT ('AB')
600       FORMAT ('CD')
700       FORMAT ('EF')
          END
```

2-4-5 逻辑IF语句

用途：逻辑IF语句是根据逻辑表达式的值，若其值是真，则执行语句st。

格式：IF(e) st

说明：e是逻辑表达式。st语句是除了DO块IF，ELSE IF，ELSE，END IF，END和另一个逻辑IF以外的任何可执行语句。表达式值为假，不执行语句st，就象碰到CONTINUE语句一样，继续向下执行程序。如：

```
IF (A.EQ.0) JM=27
DO  I=1, 10
  ⋮
END
```

A.EQ.0值为真，则执行赋值语句JM＝27，否则向下执行DO循环语句。

2-4-6 块IF语句 块IF语句只在IBM PC与APPLE II的FORTRAN 77上有，CROMEMCO的FORTRAN不具有块IF语句。在实际问题中，常常遇到选择结构不象上面的IF(e) st语句那么简单，只有一个语句要执行的，而是有一组语句或n组语句，要选择其中一组执行。对于这类问题可用块IF语句。块IF语句有如下四种格式：

格式1：IF(e) THEN

```
    Block
    END IF
```

说明：e是逻辑表达式。Block是一组可执行语句，称之为一个语句块。当表达式e的值为真时，执行Block，然后执行END IF语句；如果e表达式值为假时，不执行Block，而立即执行END IF语句。

格式2：

```
IF (e1) THEN
Block1
ELSE  IF (e2) THEN
        Block2
END IF
```

说明：e_1、e_2为逻辑表达式，Block1和Block2是可执行语句组。当e1的值为真时，执行Block 1，然后执行END IF语句；当e1的值为假时，执行ELSE IF语句，若e2为真时，执行Block 2，然后执行END IF语句；若e2为假时不执行Block2 而执行END IF语句。

格式3：

```
IF (e1) THEN
    Block 1
ELSE
    Block 2
END IF
```

说明：e和Block含义同格式1，其语句执行顺序为：当e1的值为真时，执行Block1，然后执行END IF语句。当e1的值为假时，执行Block2 然后执行END IF语句。

格式4：为块IF一般结构形式。

```
IF(e1) THEN
    Block1
ELSE IF (e2) THEN
    Block2
  ⋮
ELSE
    Blockn
END IF
```

块IF语句的格式1，格式2，格式3，是一般结构形式的格式4的某种特定组合。

注意：不允许从块IF外转移到块IF中。

例1。简单的块IF，运行程序时，若表达式的值不为真就将跳过这些语句：

```
IF (I.LT.10) THEN
      ⋮  这些语句仅当I<10时执行。
END IF
```

例2。带有一系列ELSE IF语句的块IF。

```
    IF (J.GT.1000) THEN
      ⋮  仅当J>1000时执行这些语句。
```

```
ELSE IF (J.GT.100) THEN
  ⋮  在100<J≤1000时执行这些语句。
ELSE IF (J.GT.10) THEN
  ⋮  仅当10<J≤1000且J≤100时执行这些语句。
ELSE
```

例3. 下例说明块IF结构可以嵌套，且可不用 ELSE IF 语句直接让 ELSE 语句跟在块IF后(这样作只是为了使程序更易读)。

```
IF (I.LT.100) THEN
  ⋮  这些语句仅当I<100时执行。
  IF (J.LT.10) THEN
    ⋮  这些语句仅当I<100且J<10时执行。
  END IF
    ⋮  这些语句仅当I<100 且J≥10时执行。
ELSE
    ⋮  这些语句仅当I≥100时执行。
  IF (J.LT.10) THEN
    ⋮  这些语句仅当I≥100且J<10时执行。
  END IF
    ⋮  这些语句仅当I≥100且J≥10时执行。
END IF
```

注意：CROMEMCO微型机的FORTRAN不具备块IF语句。

2-4-7 DO 语句

用途：DO语句是循环语句，在DO 之后的语句，包括结束语句重复执行。被重复执行的部分称之为循环体。

格式：DO *S* i=e1, e2[, e3]

说明：*S*是可执行语句的标号。i是循环控制变量，为整型变量。e1，e2，e3 是整型表达式。由标号*S*指定的语句必须位于DO语句之后，且同在一个程序块中。*S* 标 号语句为循环结束语句。e1、e2为循环控制变量的起始值和终止值，e3 为每循环一次使循环控制变量增加的增量值，称为步长值。当循环控制变量的值增加到等于或大于e2 时，循环结束。在 DO 语句中不含有e3时，则认为步长为1。现在举例说明之：

例1.

```
      DO 12  I=4, 16, 4
      J=I/2
      J=15-J
      ARRAY (J)=MY(I)
12    CONTINUE
```

此例中由I=4起循环执行标号12语句前几个语句，每循环一次I=I+4，至到I值为 16 时，循环结束。

例2.

```
   DO  200  I=1, 10
```

```
200   M=M+4
      ⋮
```

此例每循环一次，变量M增加4，共循环10次，因为步长为1。

DO 循环允许包括另一 DO 循环中，此种结构称之为 DO 循 环嵌套结构。但不允许从循环体外转到循环体内。

注意：对于 IBM PC 若 $DO 66有效，则循环至少执行一次。

2-4-8 CONTINUE语句

用途：带有标号的 CONTINUE 语句通常用于循环的终端语句。或使程序继续执行它的下一个语句。

格式：CONTINUE

说明：CONTINUE 语句对程序运行不产生什么操作。如：

```
      DO  10  I=1, 10
      IARRAY (I)=0
10    CONTINUE
```

2-4-9 STOP语句

用途：STOP 语句使程序停止运行。

格式：STOP[n]

说明：n是一个不多于 5 位的数字串或是一个字符常数。如果提供n时，则显示或打印 n。如：

```
      ⋮
10    STOP  100
20    STOP  AX
```

在标号为10的语句处停止，则打印或显示 “ STOP 100 ” 然后停止。在标号20的语句处停止，则打印或显示 “STOP AX” 然后停止运行。

2-4-10 PAUSE 语句

用途：PAUSE 语句是暂停语句，使程序暂停运行，直到在键盘上按下任意键。

格式：PAUSE[n]

说明：n是一个不多于 5 位的数字串或是一个字符常数。如有n 存在，则显示或打印n。如果n不出现，则显示或打印 “PAUSE” 和提问信息。按任一键程序继续执行。在初学者或程序规模较大时，用 PAUSE 语句分段调程序很方便。可以看每段运行状态。

2-4-11 END 语句

用途：END语句用于指定程序已达到单位结束位置。

格式：END

说明：END 必须位于主 程 序或子程序的末尾，并且必须是起始行，且不能有继续行。在主程序中 END 终止程序的执行。在子程序中执行END的作用与执 行 RETURN 语句一样。如：

```
      PROGRAM  MYP
        ⋮
      WRITE (*, 100) A
100   FORMAT (1H+, 2x, 'A=', F3.2)
```

```
END
```

2-5 子程序和函数

本节介绍如何定义和调用子程序和函数，以及所用到的语句。

在FORTRAN 语言的程序中，将过程分为函数过程和外部过程两类四种。如下所述：

过程 { 内部函数, 语句函数 } 函数过程
过程 { 外部函数, 子程序 } 外部过程

子程序和外部函数（也称函数子程序）都是独立的程序块，用于解决主程序中的一个子问题。子程序和外部函数分别以 SUBROUTINE 和 FUNCTION 语句开始，均以放在末尾的 RETURN 语句结束，或用 END 语句作为程序块的最后一个语句。

2-5-1 子程序 子程序由一个 SUBROUTINE 语句开头，由 END 语句结束。除了 PROGRAM，FUNCTION，SUBROUTINE 语句以外的任何语句在子程序块中都能使用。子程序是一个程序块，可以在其他程序块中用CALL 语句调用。被调时，子程序完成由可执行语句规定的一系列动作，然后控制返回调用语句的下一个语句。尽管调用子程序块时，可以用值代替参数或公共变元，但子程序本身不能直接用值代入。

一、SUBROUTINE 语句

用途：SUBROUTINE 语句是定义或者说是标识程序块为子程序。

格式：SUBROUTINE Sname[(fparm[, fparm]…)]

说明：Sname是用户给子程序定义的名字。它是全程名，又是子程序局部名。fparm 是用户定义形式参数名或称哑变元名，形式参数表定义个数。形式参数名不能出现在COMMON、DATA、EQUIVALENCE 和 INTRINSIC 语句中。子程序可以带回一个或更多的赋值形参值。如：

```
SUBROUTINE  CONVERT (RAD, DEG)
DEG=RAD*180.0/3.14159
RETURN
END
```

上例中，子程序是将RAD弧度转换成 DEG角度，名为CONVERT的子程序。

二、CALL 语句

用途：某个程序块，需要调用子程序时，只要用 CALL 语句就可调用子程序并使之运行。

格式：CALL Sname[(arg[, arg]…)]

说明：Sname 是用户定义的子程序名。arg 是实变元。实变元可以是表达式或数组名。CALL语句的实变元要与被调用的 SUBROUTINE 语句对应的哑变元，其类型与个数必须保持一致。若实变元是一个表达式时，先求其值，再与哑变元结合，执行被指定的子程序块。执行到RETURN语句或 END 语句后，控制返回到执行 CALL 语句之后的语句。子程序可被任何程序单位所调用，但不允许递归调用（在IBM FORTRAN中），即子程序不能直接调用自己。如：

```
C     EXAMPLE OF CALL STATEMENT
      IF (IEPP.NE.O) CALL ERROR (IERR)
      END
C
      SUBROUTINE ERROR (IERRNO)
      WRITE (*, 200) IERRNO
200   FORMAT (1X, 'ERROR', I5, 'DETECTED')
      END
```

三、RETURN语句

用途：RETURN 语句只能出现在外部函数或子程序内，其作用是使执行完外部函数或子程序后，导致控制返回到调用的程序块里。

格式：RETURN

说明：在一个函数或子程序内，执行 RETURN 语句等价于执行 END 语句。RETURN 语句的执行完成停止所执行的子程序或函数。使其返回到调用它的主程序或其他子程序、函数块内。如：

```
SUBROUTINE CAL
REAL A, B, C, H
H=A**B+(B+A)*C+C
RETURN
END
```

在主程序或其他子程序中用CALL语句调用子程序CAL时，将实变元A1，B1，C1，H代替子程序的哑变元A，B，C，H后，计算出H后，返回调用程序块的CALL后的第一个语句。

2-5-2 函数 在表达式中引用函数时，计算函数值并代入表达式。函数有三种：外部函数、语句函数和内部函数。

一、FUNCTION语句

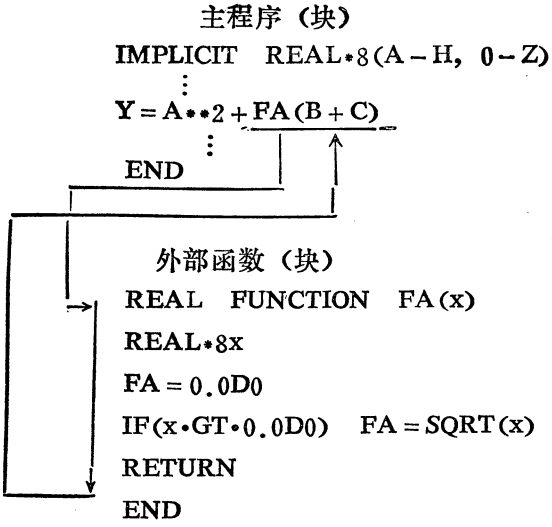
用途：FUNCTION 语句将一个过程作为函数，且规定了函数类型、函数名和形式参数(或称哑元)。

格式：TYPE FUNCTION fname ([fparm[, fparm…]])

说明：TYPE 是类型说明，可以是INTEGER，REAL，或LOGICAL 三种类型的八个类别中之一（见2—1—3节）。fname 为用户定义的函数名。fparm 是形参（或称哑元）名。

fname 是函数所在程序的全程名，又是定义的函数的局部名。TYPE 不出现时，函数类型由隐含或类型语句来定义（在函数说明部分进行）。若 TYPE 存在，则函数名不能出现在任何其他的类型说明语句中。外部函数不能用 CHARACTER（字符）类型。变量表定义了函数哑变元个数和类型。哑变元名或函数名都不允许在 COMMON，DATA，EQUIVALENCE 或INTRINSIC语句中出现。函数名必须作为变量，在调用它时出现。函数每调用一次就执行一次。调用时，是将调用程序中的实变元代入函数中的哑变元，计算函数最终值，返回引用的表达式。外部函数可带回函数值。在IBM FORTRAN 中不允许递归的函数调用，即函数不能直接调用自己。如：求

$$y=\begin{cases}a^2+\sqrt{b+c} & b+c>0\\ a^2 & b+c\leqslant 0\end{cases}$$



主程序用表达式调用以FA(x) 命名的外部函数，并执行FA(x)。执行到 FA＝SQRT(x) 时将B＋C实变元（此时已具备具体值，主程序在此语句前对B和C变元必须定义过，并提供了具体值）代入x计算出FA值。然后返回给主程序调用的表达式。

在定义子程序和函数子程序（外部函数)时，哑变元是可以去识别调用它的主程序或子程序或函数子程序中的实变元。实变元可以是具体变量、表达式、数组、数组元素。实变元通过哑变元传递给给子程序或函数子程序。实变元数目必须与哑变元相等，且对应的类型必须一致。这种相关性一直延续到子程序、函数子程序的可执行语句结束。

实变元可以是常数、函数名、表达式，但对应的哑变元不能在调用前赋值；一但赋值，将导致预想不到的结果。

若实变元是表达式，则在实、哑变元结合之前就有（算出）具体值（如例题的B＋C)。若实变元是数组元素，则它的下标表达式值在实、哑变元结合之前就必须算出。并且在程序运行过程中始终保持常数。若哑变元是变量，则实变元可以是变量、数组元素、表达式。若哑变元是数组，则实变元可以是数组或数组元素。相应的实、哑变元数组的维数大小，数目大小允许不同，但引用哑变元时，实变元必须限制在存贮记录的极限值内。当引用的元素超过这些范围时，运行程序不会检测出错误，其结果难以预料。

只有在 EXTERNAL外部语句或INTRINSTC内部语句中外部过程名（子程序和函数子程序)，内部函数名，形式参数（实、哑变元）才可以用这些名字。

类型转换内部函数名 (INT, IFIX, IDINT, FLOAT, REAL, ICHAR, CHAR)，字符顺序关系的内部函数名 (LGE, LGT, LLE, LLT)，取最大，最小值运算的内部函数名 (MAXO, AMAX1, AMAXO, MAX1, MIN1, MINO, AMIN1, MIN1) 和EOF名都不能作为实变元使用。

二、语句函数

用处：语句函数是一种形式上类似于赋值语句，用户自己定义的函数。同其他任何函数一样，通过函数名的出现来调用并执行。

格式：fname (fparm[, fparm]…) = expr

说明：fname 是用户定义的语句函数名，对所在的程序单位是局部名，否则不能使用。fparm 是形式参数名。expr 是表达式。

语句函数只能位于说明语句之后，本程序单位内任何可执行语句之前（请见2—1—1节二的表）。

表达式 expr 允许引用变量、形式参数、其他函数、数组元素及常量。其类型必须与语句函数名的类型名一致。语句函数的参数类型及个数在形式参数名表中定义。形参名的作用域就是语句函数。因此，形参名可在程序单位的其他部分，作为别的命名符。

一个语句函数引用另一语句函数，必须在引用前定义过。语句函数不能直接或间接地被递归调用。

语句函数名不能出现在任何说明句中。语句函数只能在它被定义的程序单位内引用。语句函数不允许是字符类型。如：

```
      DIMENSION  X(10)
      ADD(J, K) = J + K
      DO  1  I = 1, 10
      X(I) = ADD(I*10, I+2)
1     CONTINUE
```

在X(I) = ADD(I*10, I+2)语句中引用ADD(J, K) = J + K语句函数。计算I*10 + (I+2)值并赋给变量X(I)。

三、内部函数

在FORTRAN语言中，由 FORTRAN 编译程序定义了一套很有效的内部函数。IBM PC 和和APPLE II的FORTRAN 77 所用到的内部函数的名称、含义、变元类型、内部函数类型列入174页表中。

IMPLICIT 语句不改变内部函数的类型。对于内部函数允许有几种类型自变元。对于被引用的某一内部函数的类型与变元在类型上保持一致。

在174页表中所列举的内部函数名，可以在INTRINSIC语句中出现。若是内部函数的类型与类型说明语句中的类型相符合，这个内部函数名也可在此类型语句中出现。

一些内部函数的变元受算术函数的定义所限制，使用时要注意选择。如，负数的对数在数字上无意义，因此不能成立。

小结：

(1) 外部函数用FUNCTION语句定义的函数，格式、结构形式和作用都与子程序类似，故有函数子程序之称。

(2) 在程序设计中常常发现用户想用的函数在内部函数中没有，需要自己设计，其结构及其内容可用相当一个语句来描述，就可采用语句函数形式。

(3) 内部函数已有的就不必再费力编制了，可直接引用。

(4) 本节所列入表中的内部函数是 FORTRAN 77 所有的库函数。对于除FORTRAN 77 以外的其他FORTRAN版本的库函数还分为内部函数和外部函数。要注意区别，但是，不影响使用。

2-5-3 数据块子程序 数据块子程序是一种特殊的子程序，用它来给有名公共区变量（几个不同程序单位，需要相同初值的变量）赋值。其变量在各程序单位中都用 COMMON 语句定义同名的公用区中。数据块子程序是一个独立的程序单位，独立进行编译。

数据块子程序必须以BLOCK DATA作为起始语句，以END语句作为结束语句。

内部函数名	含义	变元类型	函数类型
INT(X)	转换成整型(1)	实型	整型
IFIX(X)	转换成整型(1)	实型	整型
REAL(I)	转换成实型(2)	整型	实型
FLOAT(I)	转换成实型(2)	整型	实型
ICHAR(C)	转换成整型(3)	字符型	整型
CHAR(I)	转换成字符型	整型	字符型
AINT(X)	截断为实型	实型	实型
ANINT(X)	四舍五入成实型(1)	实型	实型
NINT(X)	四舍五入成整型(1)	实型	整型
IABS(I)	整型绝对值	整型	整型
ABS(X)	实型绝对值	实型	实型
MOD(I,J)	整型余数(1)	整型	整型
AMOD(X,Y)	实型余数(1)	实型	实型
ISIGN(I,J)	整型传送(符号)	整型	整型
SIGN(X,Y)	实型传送(符号)	实型	实型
IDIM(I,J)	整型差分(4)	整型	整型
DIM(X,Y)	实型差分(4)	实型	实型
MAX_0(I,J,…)	整数的最大值	整型	整型
$AMAX_1$(X,Y,…)	实数的最大值	实型	实型
$AMAX_0$(I,J,…)	实数的最大值	整型	实型
MAX_1(X,Y,…)	整数的最大值	实型	整型
MIN_0(I,J,…)	整型最小值	整型	整型
$AMIN_1$(X,Y,…)	实型最小值	实型	实型
$AMIN_0$(I,J,…)	实型最小值	整型	实型
MIN_1(X.Y,…)	整型最小值	实型	整型
SQRT(X)	平方根	实型	实型
EXP(X)	e^x	实型	实型
ALOG(X)	实型变元的自然对数	实型	实型
$ALOG_{10}$(X)	实型变元的常用对数	实型	实型
SIN(X)	实型正弦	实型	实型
COS(X)	实型余弦	实型	实型
TAN(X)	实型正切	实型	实型
ASIN(X)	实型反正弦	实型	实型
ACOS(X)	实型反余弦	实型	实型
ATAN(X)	实型反正切	实型	实型
$ATAN_2$(X/Y)	X/Y的实型反正切	实型	实型
SINH(X)	实型双曲正弦	实型	实型
COSH(X)	实型双曲余弦	实型	实型
TANH(X)	实型双曲正切	实型	实型
LGE(C_1,C_2)	第一个变元大于或等于第二个变元(6)	字符型	逻辑型
LGT(C_1,C_2)	第一个变元大于第二个变元(5)	字符型	逻辑型
LLE(C_1C_2)	第一个变元小于或等于第二个变元(5)	字符型	逻辑型
LLT(C_1,C_2)	第一变元小于第二个变元(5)	字符型	逻辑型
EOF(I)	文件整型端(6)	整型	逻辑型

注释：

(1) 对于实型X，若X＞＝0，则INT(X)是不大于X的最大整数；若X＜0，则INT(X)是不小于X的最

负整数。IFIX(X)和INT(X)一样。

(2) 对于整型I，REAL(I)的有效部分和I的有效部分一样精确。FLOAT(I)类似于REAL(I)。

(3) ICHAR把字符值转化为整型值，字符的整数值是字符ASCII的内部表达式，它的范围0～255。对于任何两个字符C_1和C_2，(C_1.LE.C_2) 是真，当且仅当 ICHAR (C_1). LE.ICHAR (C_2)是真。

(4) 若数为正数且是 0 以外的数，IDIM和DIM即被定义为实际差分。

(5) 如$C_1 = C_2$或在ASCII码的序列中，C_1在C_2之后，则LGE(C_1, C_2) 带回值为真，否则为假。如果在ASCII码的序列中，C_1在C_2之后，那么LGT(C_1, C_2) 为真，否则为假。如果在ASCII 码的序列中，C_1在C_2之前或$C_1 = C_2$，则LLE(C_1,C_2)为真，否则为假。如果在ASCII码序列中，C_1在C_2之前，则LLT(C_1, C_2)为真，否则为假。

(6) 如果EOF(I)的变元所确定的单元在文件记录的末端或超过了，则返回值为真，否则返回值为假。I的值必须对应于一个打开文件或0。0是用来指显示器或键盘。

(7) 表中的X和Y是实型，I，J为整型，C_1，C_2是字符型。

格式：
```
BLOCK  DATA
  ⋮
END
```

例：
```
BLOCK  DATA
DIMENSION  A(10)
COMMON/L1/A/L2/X, Y, Z
INTEGER  X, Y, Z
DATA  A(3), A(4), A(7), X, Y/0.3, 0.4, 0.6, 24, 50/
END
```

2-6 输入/输出(I/O)系统

FORTRAN语言提供了输入/输出(I/O)语句，能使运行中的程序与接到CROMEMCO、APPLE II和IBM PC机上的任何设备(如：磁盘、键盘、显示器、打印机) 之间传送数据。本节将叙述FORTRAN的I/O系统，I/O基本概念，I/O语句和FORMAT格式语句。

2-6-1 概述 为了理解I/O语句功能与用法，首先需要熟悉有关FORTRAN的I/O系统的一些术语和概念。

一、文件和设备

文件是记录的集合。每个记录由一个或多个数据项组成。每个记录表明数据具有的某种属性，并且由用户给记录定义一个名字，称之为属性名，也称之为记录的域。

文件通常包括数据文件和程序文件。数据文件可做为程序运行时需要的输入数据，或是程序运行的结果。总之是数据的集合。程序文件为各类程序。在本节中只涉及数据文件。

文件的长度常常超过内存允许的空间，因此，无论是程序文件还是数据文件，通常都存放在外部设备上。这样，每一文件总是和某一特定设备相联系的。例如，建立在磁盘上的文件，称磁盘文件；在打印机上输出的文件，称之为打印机文件等。微型机上常用的设备有：

1.磁盘（多数为软盘，少数配硬盘）

2.键盘显示终端

3.打字机

4.盒式磁带机

二、文件名、设备名和逻辑设备号

在FORTRAN中，输入/输出(I/O)操作是独立于设备的，即输入/输出语句中不指定具体设备，只是在语句中引入一个逻辑设备号，这个设备号隐含地标识某一特定设备，被标识的设备正是执行输入/输出操作的设备。逻辑设备号和设备之间的联系是通过OPEN语句来实现的。

可选用的逻辑设备号及其约定随FORTRAN版本而异：

1.APPLE II和IBM PC FORTRAN逻辑设备号是：

(1) 逻辑设备号是一个整型表达式，取值为0～1X。

(2) 逻辑设备号可以用一个星号"*"表示设备号为0或控制台。

2.CROMEMCO FORTRAN逻辑设备号

(1) 逻辑设备号是一整常数或整型变量，其值在1～255范围内，包括1和255。

(2) CROMEMCO将逻辑设备1-19分配给有关的I/O设备，如下表所示：

逻辑设备号	外部设备
1	控制台
2	行式打印机
3～5	控制台
6～10	磁盘

3.任何一个或全部逻辑设备号都可以由用户重新分配，以便和特定的系统配置相适应。

4.FORTRAN也允许通过CALL OPEN语句将逻辑设备号分配给磁盘文件。当需要同时打开5个以上文件时，这种临时重新分配逻辑设备号的方法是很有用的。

每个文件都有一个文件名，它标识一个操作系统能够识别的符号名。

文件名包括两部分：文件名部分是由字母打头的字符串所组成；文件类型部分是由一个圆点"."后跟三个字符组成。

注意：文件名的长度随系统而异，在IBM PC上是1～8字符长；在APPLE II上是1～30字符；在CROMEMCO上是1～11个字符长。

三、文件缓冲区

在IBM PC中，文件缓冲区又称内部文件。内部文件是在内存建立一个缓冲区利用I/O系统的格式化能力，进行外部字符表示值与机内值之间的转换。即每读入一个字符变量，就使字符外观值转换成机内数值、逻辑值或字符值。输出（写出）一个字符变量，可把字符内部值转换成外部字符表示形式。

一个内部文件就是一个字符变量或字符数组元素。该文件有一个记录，其长度与字符变量或字符数组元素的长度相等。如果要写出的信息不够长，则用空格填满。在I/O语句执行之前，文件指针总是指向文件的始端。内部文件只有I/O的格式化文件和顺序文件。在I/O语句READ和WRITE中规定了内存单元。如：

```
C  EXAMPLE  OF  INTERNAL  FILE I/O
        CHARACTER*11  A, B
        DATA  A/'1.234—86.45'/
        READ(A, 2)  X, Y
        FORMAT  (F5.3, F6.2)
```

```
Z=X+Y
WRITE  (B, '(F6.2)')  Z
```

四、记录

一个记录是字符、数值的序列；记录是FORTRAN文件系统的最基本单位。记录有以下三种：

1.有格式记录

有格式记录是一个字符序列，以RETURN键为记录终止标志。FORTRAN对记录值的解释与操作系统和正文编辑程序对字符的解释是相同的

当读入源数据，或从另一个程序读入时，有格式记录是很有用的。有格式记录必须与有格式I/O语句配合使用，当我们想从显示器、键盘或打印机读入或写出信息时就需要有格式记录了。

2.无格式记录

无格式记录是值的序列存贮或恢复，不需要用户编辑或用户干于。没有系统转换或解释，也没有记录结束的物理表示。

3.文件结束记录

FORTRAN文件系统中，设置了一个结束文件记录。尽管没有一个真实的记录与它对应，仍然在一个文件的最末端记录后有一个假想结束文件记录。

五、文件指针

每一个文件都有初始点、终点，当前记录、先前记录和下一个记录。文件指针正是由I/O操作给定的，用来确定文件I/O操作时所指定的位置。在一个文件指针不能位于几个记录中间的情况下，记录是一个接一个的，不存在当前记录的问题。

打开一个顺序文件进行写时，指针指在文件的开头，文件中所有的旧文件数据被抹去，顺序写完，这时文件指针指在文件的末尾，而不是在结束文件的记录上。执行完 ENDFILE 语句后，指针超过了结束文件记录。同样，READ语句执行后，指针在文件的末尾，只有执行ENDFILE后，指针才超过结束文件。

六、文件结构

文件结构包括文件组织方式和存取方式。

1.文件组织方式

文件组织方式是指记录在文件中的排列方式。IBM PC，APPLE II，CROMEMCO 的FORTRAN支持两种文件组织：顺序组织文件和关系组织文件。

(1) 顺序组织文件

在顺序组织文件中，记录是按物理顺序出现的，其物理顺序总是与记录写入文件的原来次序完全一样。

(2) 关系组织文件

在关系组织文件中，文件是一系列固定长度的单元组成。从第一个单元到最后一个单元，依次编号为 1，2，3，……。每个单元号代表了该单元相对于文件起始点的相对位置。每个单元可包括一个记录，也可以是空的。对于文件中的一个特定记录是通过记录所在的单元号(或叫记录号)来引用的。

2.文件存取方式

文件存取方式是程序检索文件中的一个记录，或者把记录记入文件中的方法。为此在

读/写语句中要指明对文件的存取方式。IBM PC，APPLE II，CROMEMCO的FORTRAN支持两种存取方式：顺序的和直接的。

(1) 顺序存取

所谓顺序存取是指记录是接序，顺序存取的即对文件的记录是一个接一个地顺序读/写。而不是随机地读或写文件中某一个指定的记录。例如，要读或写文件中的第30个记录，必须顺序经过位于前面的29个记录，然后才能读或写第30个记录。如果刚读完第30个记录，还想读或写前30个记录的某一记录时，则必须先用退回语句，退到要读或写的记录上，才能读或写指定的记录。

对于顺序组织文件是按记录的物理顺序存取的；对于关系组织文件的顺序存取是按记录号递增的顺序存取的。

(2) 直接存取

直接存取是指直接读或写文件中任一个指定记录，而不是按顺序存取。存取方法灵活，可以随机地读或写文件中的某一个指定记录。由读/写语句来指直接存取的记录号。

注意：在IBM PC，APPLE II和CROMECO的FORTRAN中，按存取方式分成顺序文件和直接文件（或称随机文件），我们在下面将使用这两个术语。

七、直接文件和直接设备

与直接文件和顺序文件相对应的设备也有直接的和顺序的。顺序文件的特点是文件的字符流，它要求设备除了读和写外再没有什么明显的操作，能与此相适的有：键盘、显示器、打印机为顺序设备。直接设备还具有查找某一存贮单元的操作，它们既可顺序地也可随机地存取，如磁盘就是直接设备。因此，直接设备能够支持直接存取文件。FORTRAN I/O 系统不允许在顺序设备上有直接存取文件。

2-6-2 输入/输出(I/O)语句概述

输入/输出操作涉及两个方面：必须指出在哪个外部设备上执行I/O操作；有哪些变量的值是要输出的。要输入的数据又要送入给哪些变量。输出设备是由 OPEN（打开）语句中的逻辑设备号来指定的。变量是由READ(读)，WRITE（写）语句中的I/O表来确定的。

除了OPEN、READ、WRITE语句之外，输入/输出操作还用到其他一些语句，有：

(1) CLOSE关闭文件语句

(2) ENDFILE文件结束语句

(3) REWIND返转语句

(4) BACKSPACE回退语句

上述语句在本章所述的三种机型的FORTRAN中，I/O功能相同，其中大多数语句格式也相同。更主要的是对于I/O操作，其过程是一致的，即都是先打开文件，然后执行读/写操作，最后关闭文件。因此，本小节以文件操作过程顺序介绍上述语句，仅就不同的语句分别列出其各自的格式。

在说明I/O语句之前，先介绍几个概念。

1.I/O语句元素

各种I/O语句都有确定的元素，这些元素在IBM PC和 APPLE II 的FORTRAN中所用到的缩写符号U、f和iolist，说明如下：

(1) 设备说明符U

在I/O语句中可取以下形式之一：

① ＊——指的是键盘/显示器。

② 整型表达式——指的是外部文件的设备号等于表达式值。

③ 变量的符号名或数组元素符号名——指的是用变量值或数组元素值来确定的内部文件。

(2) 格式说明符*f*

在I/O语句中可取下面三种形式中的一种：

① 语句标号——由I/O语句中确定的语句标号，再由语句标号指定的格式 FORMAT 语句。

② 整型变量名——由ASSIGN 语句指定整型变量值作为语句标号引用 FORMAT 语句。

③ 字符表达式——字符表达式当前值用来作格式说明。

2.输入/输出实体

输入/输出实体是一个变量名，数组名，数组元素名。数组名意味着所有的数组元素以存贮序列的顺序输入/输出。输入、输出实体可分别在READ语句和WRITE语句的iolist(I/O表）中说明。

输出实体可以是不以"("左括号开头的任何表达式，以便与隐含的循环表中的表达式(见下面3.）区别开来。如：

表达式：(A＋B)＊(C＋D)

必须写成：＋(A＋B)＊(C＋D)

3.隐含循环表

在READ和WRITE语句中作为I/O操作指定隐含循环表。其形式为：

(iolist　i=e_1, e_2[, e_3])

iolist为隐含循环表标识符名，i、e_1、e_2、e_3 和DO循环语句定义一样，即i是整型变量，e_1, e_2, e_3是整型表达式。i的整型类型说明语句与e_1, e_2, e_3保持一致。

在读语句中，循环变量i或相关联的实体不能作为输入项在表 iolist中存在。但可以在同一读语句嵌套读入，嵌入iolist 的循环体，由i的循环迭代而有效地重复读入，被嵌入、最里层的循环体总是最先执行。举例如下：

```
        INTEGER  ARRAY(3, 2)
        READ  (*, 200)  ((ARRAY(I, J), J=1, 2), I=1, 3)
200     FORMAT(6I2)
        WRITE  (*, 300)  ((ARRAY(I, J), J=1, 2, I=1, 3)
300     FORMAT(' ', 6I2)
        END
```

2-6-3　打开文件语句　打开文件是把要进行I/O操作的设备与设备上的文件联系起来，这时才能对文件进行读写操作。打开文件有两种方式：

一、隐含打开文件(只限于CROMEMCO的FORTRAN)

在文件名与设备逻辑号已被指定的情况下，被指定的文件还没被逻辑号打开而准备执行从一个逻辑设备中读或向一个逻辑设备写时，为了实现输入或输出，就会分别自动地打开该文件。

对于CROMEMCO机，首先寻找和文件相关联的磁盘驱动器，隐含地选用当前所用的磁盘。对于文件将有一个隐含的文件名，它有逻辑设备号。如：

FORT06·DAT，或FORT07·DAT，……，FORT10·DAT

二、显式打开文件

1.CROMEMCO的FORTRAN的OPEN语句

格式：CALL OPEN (LUN, filename, drive)

说明：LUN是逻辑设备号，是一个和被指定的文件相联系的设备号。LUN必须是一个整型常数或整型变量，其值在1～10范围内，包括1和10。

filename是文件名，是一个ASCII字符串。文件名必须在11个字符以内，文件类型以圆点开始，后跟三个字符，若不是三个字符，可补以空格。

drive为设备，是文件存在或将要存在的磁盘所在的驱动器号码。驱动器号必须是一个整数，它所允许范围，由操作系统决定。

0——当前选用的驱动器

1——驱动器A

2——驱动器B

3——驱动器D

4——驱动器C

2.IBM PC和APPLE II的FORTRAN的OPEN语句

用途：OPEN 语句是把设备号指定的外设和外设上一个指定名的文件联系起来。

格式：OPEN (U, FILE=fname〔, *STATUS*=st〕
〔, ACCE*SS*=ac〕〔, FORM=fm〕〔, RECL=r1〕)

说明：U是设备标识符（参见本节开始叙述I/O语句的元素部分）。U必须是OPEN语句的第一个域，不能作内部单元的标识符。

fname是文件名，它必须是OPEN语句第二域如果省略fname时，文件名可在运行时确定。参见本节的后面的“运行时文件指定”一段。

fname后面所有的域都是任选的，并且可以按任何顺序出现。这些任选项是包括可选择空格（除RECL=外）字符常量。

st可以是'OLD'或'NEW'。如果此项省缺则认为是'OLD'。OLD用于读或写现有文件，而NEW则用于写新文件。

ac是SEQUENTIAL（顺序）或DIRECT（直接）。如果此项省缺，则认为是SEQUENTIAL。

fm是FORMATTED（有格式）或UNFORMATTED（无格式）或BINYRY(二进制)。

r1是整型表达式，它指定记录长度。OPEN语句中的r1（参数）仅用于直接存取文件。

把0作为设备逻辑号与一个文件相联是无效的，设备0永远是与键盘和显示器连接的。

例1. 显式打开的顺序有格式文件

从一个命名为OUT·TEXT的文件上，拷具一个带有3列，每列占7格域宽的整数。同时调换了第一列和第二列的位置。

```
      PROGRAM  COLSWP
      CHARACTER*64  FNAME
C  Prompt to the display by writing to *
      WRITE  (*, 900)
900   FORMAT  ('INPUT  FILE  NAME_')
```

```
C   Read the file name from the keyboard by reading
C   from*.
        READ  (*, 910)  FNAME
910     FORMAT  (A)
C   Use unit 3 for input; any unit number except
C    0  will do.
        OPEN  (3, FILE=FNAME)
C   Use unit 4 for output; any unit number except
C   Ouand 3 will do.
        OPEN  (4, FILE=OUT.TXT, STATUS='NEW')
C   Read and write until end of file.
100     READ  (3, 920, END=200)  I, J, K,
        WRITE  (4, 920)  J, I, K
920     FORMAT  (3I7)
        GOTO  100
200     WRITE  (*, 910)  'Done'
        END
```

例2．程序段1。

```
        WRITE  (*, '(A)')  'SPECIFY  OUTPUT  FILE  NAME_'
        READ  (*, '(A)')  fname
        OPEN  (7, FILE=fname, ACCESS='SEQUENTIAL'
```

程序段2。

```
C   OPEN  AN  EXISTING  FILE
C   CALLED  DATA3.TEXT  AS  UNIT  3.
        OPEN  (3, FILE='DATA3.TEXT')
```

3.运行文件名的指定

在有文件名的情况，通常用OPEN语句中的文件名代替程序中直接代码。在IBM PC的FORTRAN语言中，在没有文件名的情况下，可以通过在OPEN语句中的FILE =处，运用全空格作为指定的文件名。

例如：

```
            INTEGER  UNITNO
            CHARACTER*1  FNAME
            DATA  FNAME/' '/
C   Open  units 5 through 7.
            DO  10  I=5, 7
            OPEN  (I, FILE='     ')
10          CONTINUE
C   Obtain  the  unit  number  and  then
C   perform the  OPEN
            WRITE  (*, 20)
```

```
20      FORMAT  (1X, 'Enter      the Unit      number:')
        READ    (*, '(BN, 16)')      UNITNO
        OPEN    (UNITNO, FILE = FNAME)
        END
```

当运行FORTRAN程序，执行OPEN语句操作时，系统发现空的文件名，就从DOS操作系统的文件或打开的外设中得到实际文件名。

FORTRAN运行时，系统试图得到DOS文件说明：

①对执行的每一OPEN语句，就要找出运行文件的命令总线上空文件名出现的地方，从命令总线上取一个DOS文件说明。假如命令总线上出现的文件说明不只一个，则每个之间必须用空格分开。如：

```
B>files  user  prn  con  info•dat
Enter  the  unit  number:
          300
```

(2) 当命令总线上的文件说明用完时，则向用户发出提示信息：

```
Filename  missing  or invalid—try  again!
Unit  *****?
```

其中*****是需要提供文件名的单元号。用一个DOS文件说明回答提示后，OPEN语句就根据文件说明执行。

```
B>files
File    name    missing    or    empty—try    again!
UNIT    5?      user
UNIT    6?      prn
UNIT    7?      con
Enter   the     Unit     number:
 —12
UNIT—12?      info • dat
```

(3) 在命令总线上指定外部文件说明表是无效的。

```
B>files  user  prn  con  info • dat  extra1 • fil  extra2 • fil
Enter  the  Unit  number:
        30
```

注意：必须把所有文件说明正确地、有效地放在命令总线上，OPEN语句执行时，才能知道OPEN语句执行时它的顺序。

2-6-4 读/写语句 对文件进行读/写操作的读写语句，在IBM FORTRAN，APPLE FORTRAN 和 CROMEMCO FORTRAN 中形式和功能完全一样，只是所用的符号不同而已。因此，这里只对 IBM FORTRAN 语言中的读/写语句进行说明。

一、READ 语句

用途：假定没有结束文件或错误情况发生时，READ语句是用来建立输入输出表的项目的，并完成读入数据任务。

格式：READ(U[, f][, REC=rn][, END=S1][, ERR=S2])iolist

说明：U是前边“I/O语句的元素”中讲过的设备说明符。U必须为第一参数，对有格

式读，f必须为第二参数。在无格式情况不使用。只在直接存取情况下，才由rn指定，否则将导致出错。rn为正整数表达式，指出记录号为rn的当前值。对于一个直接存取文件，如果省略REC=rn，则在此文件当前位置顺序地继续下去。

*S*I是任选的语句标号。如果已提供，则遇到文件结束时，转移到标号为*S*I的可执行语句。如果不提供，则读文件末尾时导致运行出错。此语句必须与 READ 语句同处在一个程序块中。

*S*也是任选的语句标号。如果提供，则在I/O错时，使控制转移到标号为*S*的可执行语句，如果不提供则 I/O错，即导致运行出错。假如读是内部的，则指定的是字符变量为输入的数据源。否则外部设备是数据源。

例如：

```
C  NEED  A  TWO  DIMENSIONAL  ARRAY  FOR  THE
C  EXAMPLE
       DIMENSION  IA(10, 20)
C  READ  IN  THE  BOUNDS  FOR  THE  ARRAY  THESE
C  BOUNDS  SHOULD  BE  LESS  THAN  10  AND  20
C  RESPECTIVELY. THEN  READ  IN  THE  ARRAY  IN
C  NESTED  IMPLIED  DO  LISTS  WITH  INPUT  FORMAT  OF
C  8 COLUMNS  OF  WIDTH  5  EACH.
       READ (3,990) IL, JL, ((IA (I, J), J=1, JL), I=1, IL)
990    FORMAT (2I5/,  (200I5) )
```

为了一目了然，将注释行去掉，其程序简化如下：

```
       DIMENSION  IA(10, 20)
       READ(3, 990)IL, JL, ((IA(I, J), J=1, JL), I=1, IL)
900    FORMAT(2I5/, (200I5))
```

二、WRITE语句

用途：WRITE 语句建立输入输出表项目，并把表中指定应传输的数据传送给指定的设备或设备上的文件。

格式：WRITE(U[, f][, ERR=*S*][, REC=rn])iolist

说明：U是“I/O语句的元素”中提到的设备说明符。U必须写语句的第一参数。f对有格式写时，应为语句的第二参数，否则不使用。*S*是可任选的语句标号，如有，则在I/O错时，控制转移到标号为*S*的可执行语句；如没有，则在I/O错时，导致运行出错。rn仅指定直接存取文件，否则出错。rn是正整数表达式。其值使写操作定位在rn记录上。如是直接存取文件，REC=rn 被省略，则从此文件的当前位置继续写。如果内部写，则指定的字符变量或字符数组元素是输出的目标。否则外部单元是输出目标。

2-6-5 REWIND语句

用途：执行REWIND语句完成指定的设备的文件定位于起始点。

格式：REWIND U

说明：U是设备的说明符，如“I/O语句的元素”中所叙述的，U是必须有的。

2-6-6 ENDFILE 语句

用途：当文件的下一个记录与指定的设备相关联时，使用ENDFILE语句，表明“写”出

一个文件记录的结束。

执行ENDFILE U后，指针指向文件记录的末端。到此时，数据传递才终止，当BACKSPACE或REWIND执行时，再重新进行。

注意：BACKSPACE语句不能支持直接文件，如把BACKSPACE 语句用于直接文件时，此语句将被忽略而不起作用。

2-6-7 BACKSPACE 语句

用途：执行 BACKSPACE 语句，使指定的设备文件，回退到前一个记录的顶部。

格式：BACKSPACE U

说明：U是“I/O语句的元素”说明的设备标识符，U是必须有的。当BACKSPACE 与无格式/顺序文件一起使用时，只需要用一个字节支持文件，利用此优点，所使用二进制BINARY文件模块。

注意：若先前记录是文件记录的底部，那么指针将指向最后一个记录之前。若指针指向记录的中部，执行BACKSPACE语句后，将指针移到记录的开始。若没有先前记录，文件的位置不会改变。

一、BACKSPACE和顺序设备的联系

BACKSPACE（回退）语句对I/O系统，允许格式化顺序文件，即允许对顺序设备（如键盘、显示器或打印机）上的文件进行回退处理。

二、BACKSPACE和无格式顺序文件的联系

在无格式顺序存取文件中，没有指定记录的界线，因此，在无格式顺序存取文件中，用一个字节支持文件，BACKSPACE语句被文件定义为用一个字节代替。这与随机（直接）存取文件有很大区别，在直接存取文件中，每个记录已具有固定长度，因此，在直接无格式文件上是可以进行记录回退处理。

三、部分读和无格式顺序文件的联系

前面已提到，在无格式顺序存取文件中记录没有指定记录的界线，因此，读记录的一部分后，文件指针不会指在下一个记录起始点。如果采用这种限制的优点，就必须使用二进制文件。二进制文件定义BACKSPACE为替代一个字节。

2-6-8 CLOSE（关闭文件）语句

用途：CLOSE 语句用来关闭I/O序列的通道，只有再用同一个设备号重新打开，才有可能使不同文件或设备输入/输出。

格式：CLOSE(U[, STATUS=st])

说明：U是必要的设备说明符。应作第一参数出现。还必须与OPEN 语句中的设备说明符相同、它不能是内部单元说明符。st 是保存‘KEEP’和删去‘DELETE’两项中任选其一。若无此选项，则按‘KEEP’执行，这个选择项是一个字符常数。如果 STATUS = ‘DELETE’，则文件删除。按 FORTRAN 程序的标准定义，自动关闭所有打开的文件等价执行了相当于STATUS = ‘KEEP’时的CLOSE。对于设备标识符为0的CLOSE是无效的，因为CLOSE操作，对于设备0所对应的显示器、键盘来说，毫无意义。

例.

```
C  Close the file opened in OPEN example,
C  discarding the file
       CLOSE (7, STATUS= 'DELETE' )
```

```
      END
```

程序段例：

```
C  DECLARE  THE  DATA  AREAS
         CHARACTER*15  ITEM (10)
         REAL  COST (10)
         INTEGER  MIN, MAX, VOLUME (10)
C  OPEN  THE  FILES
         OPEN (6, FILE= 'PRN')
         OPEN (1, FILE= 'INVENTORY.DAT', STATUS= 'NEW',
        I FORM= ' UNFORMATTED')
C  READ  IN  THE  DATA  FORM  THE  KEYBOARD
         DO  10  I=1, 10
         WRITE (*, 110) I
110      FORMAT ( 'ENTER  ITEM', I2, ': ' /)
         READ (*, 100 ) ITEM (I)
100      FORMAT (A)
         WRITE (*, 210) ITEM (I)
210      FORMAT ( 'ENTER  COST  AND  VOLUME  FOR', A)
         READ (*, 200)  COST (I), VOLUME (I)
         IF (COST (I).EQ.0.0) GO  TO  20
200      FORMAT (BN, F10.2, I10)
C     WRITE  THE  DATA  TO  AN  UNFORMATTED  FILE
         WRITE (1) ITEM (I), COST (I), VOLUME (J)
10       CONTINUE
20       I=I-1
C  FIND  THE  MIN  AND  MAX  OF  THE  DATA
         MIN=VOLUME (I)
         MAX=VOLUME (I)
         DO  30  J=2, I
           IF (VOLUME (J).CT.MAX) MAX=VOLUME (J)
           IF (VOLUME (J).LT.MIN) MIN=VOLUME (I)
30       CONTINUE
C  REPORT  THE  RESULTS  TO  THE  PRINTER
         WRITE (6, 220)
220      FORMAT ( 'ITEM', 22X, 'COST', 9X, 'VOLUME', 6X,
        1 'INVENTORY')
         WRITE (6, 230)
230      FORMAT ( '+', '_______', 22X, '_______', 9X, '_____
        1_____', 6X, '________________')
           DO  40  J=1, I
```

```
          WRITE (6, 400) ITEM (J), COST (J), VOLUEM(J), COST
        1 (J)*VOLUME (J)
  4 0     CONTINUE
  4 0 0   FORMAT (1X, A, 5X, F10.2, 5X, I10, 5X, F10.2)
          WRITE (6, 600) MIN, MAX
  6 0 0   FORMAT ( 'VOLUME MAXIMUM = ', I10,
        1 /, 'VOLUME  MINIMUM = ', I10)
          END
```

上例中的FORMAT语句中的编辑符，请看下面部分。

2-6-9 格式化I/O和FORMAT语句

一、格式说明和FORMAT语句

有格式的I/O语句，就是在READ或WRITE语句中规定一定格式。象2-6-2 节中的“I/O语句的元素”所说的那样，有两种方式引用 FORMAT 格式语句或用字符表达式直接说明格式等三种格式指定格式。下面列举了同一格式用几种方式指定的程序段例：

1，用FORMAT语句指定格式

例1.

```
        WRITE(*, 990)I, J, K
9 9 0   FORMAT(1X, 2I5, I3)
```

例2.

```
        ASSIGN 990 TO IFMT
        WRITE(*, IFMT)I,J,K
9 9 0   FORMAT(' ', 2I5, I3)
```

2.用字符表达式指定格式

例3.

```
        WRITE((*, '(I6, I5, I3)')I, J, K
```

例4.

```
        CHARACTER*8 FMTCH
        FMTCH = '(1H, 2I5, I3)'
        WRITE(*, FMTCH)I, J, K
```

FORMAT语句前面必须有标号，且象其他非执行语句一样，不能有某分支操作项目。

格式说明本身必须用“(”左括号为起点，其后跟格式说明，最后以“)”右括号结束。“)”后的所有字符被忽略。格式说明是一个或多个格式说明符表，表中的每项要用逗号分开，每项为下列内容之一：

(1) 〔r〕ed 为可以重复编辑描述符；

(2) ned 为不可重复的编辑描述符；

(3) 〔r〕fs 为一个嵌套的格式说明，最多可有三层嵌套，在最外层里面允许有圆括号。

r是一个非零无符号的整常数，它说明可重复调用的次数。如果格式说明没有两重性，分隔两个表项的逗号可以省略。例如，在一个P 编辑描述符后或在/斜杠编辑描述符的前或后。

二、可重复编辑描述符

〔r〕ed为可重复编辑描述符，有如下几种：

IW	整型变量编辑描述符
FW•d	小数实型变量编辑描述符
EW•d	指数实型变量编辑描述符之一
EW•dEe	指数实型变量编辑描述符之二
LW	逻辑型变量编辑描述符
A	字符型编辑符之一
AW	字符型编辑符之二

I，F，E，L，A表示编辑方法，即以什么数据类型输出，是整型，还是浮点型……或是字符型。

W和e表示非零无符号整常数。d表示无符号的整常数。

三、不可重复的编辑描述符

ned为不可重复的编辑符，有如下几种：

'xxxx'	任意长的字符常量
nHxxxx	另一种规定字符常量的方法
nx	指针编辑符
KP	比例因子编辑符
／	斜杠编辑符
＼	反斜扛编辑符
BN	空格解释符之一
BZ	空格解释符之二

'，H，x，／，＼，P，BN，BZ均表示编辑方法。X是任意的ASCII字符。n是非零不带符号的整数，K是一可选择的有符号整数。

2-6-10 输入/输出表的相互作用与格式说明

一、输入/输出表

输入/输出表（iolist）实质是输入/输出实体项目表。在此表中如至少有一项的话，那么在格式说明中至少要有一个可重复的编辑描述符。在I/O语句执行期间，iolist 表中的每一项，都依次与一个可重复编辑描述符相联系。只有在iolist中，没有指定输入/输出的实体项即表空时，才能使用空编辑说明符(　　)。I/O语句在此情况下，只引起与此格式相联系的隐式跳行动作。除了可重复的编辑描述符和 iolist表外，其他格式控制项是与文件记录发生作用而不是与iolist表项联系着。

格式说明语句——FORMAT 语句中的各项是按从左到右的顺序解释。可重复编辑描述符的动作将出现r次，如r省略，重复因子按1处理。类似地，一个嵌套格式说明作为它的表项出现r次。

二、格式说明

I/O 语句中的格式说明是由“格式控制器”完成的，按自左向右的顺序扫描所有的格 式表项。当遇到可重复编辑符时，如果I/O iolist 表中有实体项，就把实体项与编辑描述符联系起来，在此编辑描述符控制下，处理 iolist 项的I/O。在iolist表中，不再出现实体项时，“格式控制器”停止I/O过程。如果“格式控制器”遇到了格式说明的最后一个终止符号“)”，而且在iolist中再没有项目了，“格式控制器”则终止I/O。但是，在iolist表中还有项目，则文件指针指向下一个记录的起始点，这时，“格式控制器”要用下一个终止符“)”，作为I/O终

止点。如果没有终止符“)”时，“格式控制器”就从头开始重新扫描格式说明。在被重新扫描的部分格式内，必须至少有一个可重复编辑的描述符。

如果“格式控制器”遇到重复嵌套格式说明开始，重复因子指出格式说重复的次数。

2-6-11 编辑描述符 前面已把编辑描述符分为两大部分，一是不可重复编辑描述符；二是可重复的编辑描述符。现在将这两部分分述如下：

一、不可重复的编辑描述符

1.'撇号编辑

撇号编辑描述符指定字符常量形式。如'A×CD'两撇号之间的字符A×CD为字符常量。字符常量中的空格是有意义的。两个以上的撇号，如'''可为只有一个。对于READ(输入)语句，若把被输出的字符常量送给输出设备，不能使用撇号编辑。参看“H(Hollerith编辑)”

例1.

```
          WRITE(*, 970)
970       FORMAT('ABC  DEF')
```

输出结果为 ABC DEF

或者用下面的例子：

例2.

```
      WRITE(*, '(" ABC"  "DEF" )')
```

输出与例1有同样的结果：ABC DEC

2.H(Hollerith) 编辑

nH编辑描述符的作用是把要传递的几个字符，包括空格（有效）送给输出设备。Hollerith不能用于输入(READ语句)。

例.

```
          WRITE(*, 900)
900       FORMAT(7HABC  DEF)
```

输出结果与前面例1、例2、有同样结果：ABC DEF。

H编辑符作为输出指定，其格式为 $nHh_1h_2\cdots h_n$象上例中的 7HABC DEF，输出时为 7 个字符，ABC DEF六个字符中间一空格符。

3.x定位编辑

指针编辑描述符的格式为nx。在输入（使用READ 语句）时，nx使文件指针前进n个字符，即越过n个字符。在输出（使用WRITE语句）时，nx 使输出有n个空格，然后再提供输出（记录）具体值。但是nx对其他不产生任何操作。

4./斜杠编辑

/斜杠编辑描述符表示在当前的记录传送终端数据。在输入时，文件位于下一个记 录 的起始处。在输出时，写上一记录终端标志，并把文件置于下一个记录的起始处。

5.＼反斜杠编辑

“＼”是IBM PC FORTRAN的编辑符，而在APPLE II中用 $ (Dollar)。一般说来，当“格式控制器”终止时，则是出现传送当前记录数据的末端场合。如果最后一个编辑符被“格式制控器”遇到，则删掉记录结束标志，这就允许子序列I/O 语句继续读或写超出这个记录。这种格式最常用于显示器提示并在同一行里读入回答信息。如：

```
WRITE(*, '(A＼)')'Input an integer→'
```

READ(*, '(BN, I6)')I

当从*设备读入时，＼编辑描述符就不删掉自动的记录结束标志。从键盘得到的输入总是要以Enter键作结束。这就允许backspace和Control-x键具有适当功能。

6.P比例因子编辑

比例因子编辑符的格式为nP，是为子序列F和E编辑描述符设置的，直到另一个nP编辑符出现。每一个I/O语句的起始，比例因子为0。比例因子作用于格式编辑是用下述方法：

(1) 在具有F和E编辑符的输入情况下，没有确定的指数关系。对于F编辑符用于输出编辑时，外部表示的数等于内部表示的数乘以10**K。

(2) 在具有F和E编辑符的输入情况下，输入域又没有确定的指数，那么比例因子就没有作用。

(3) 在有E编辑符的输出情况下，实际数为输出数乘以10**K，即数的小数点位左移K位，而将指数减去K，其值不变。

7.BN和BZ描述符

BN和BZ编辑描述符规定了对数值的输入域中空格的处理。BZ编辑描述符使得在数值输入域中遇到的所有嵌在中间或尾部的空格被处理为零。BN编辑描述符使得在一个数值输入域中遇到的所有嵌在中间或尾部的空格被忽略，即移去空格，使域的其余部分右对齐。

例如，下面READ语句接受了斜杠之间所表示的字符，如123：

```
          READ(*, 100)I
100       FORMAT(BN, I6)
          /123<cr>/,
          /123    <cr>/,
或者      /    123<cr>/
```

其中<cr>表示按下Enter键。在APPLE II为按return键

二、可重复的编辑

1. I、F和E的规则

对于I、F和E编辑描述符适用于整数和实数的I/O，其规则如下：

(1) 对于空格，前头的空格是无效的，其它空格的解释由BN或BZ指定，而空格域总是变为0值，“+”号是任选择的。

(2) 在输入的情况下，对于F和E编辑，在输入域中出现的小数点位置编辑符说明是不起作用。

(3) 在输出的情况下，所要输出的字符是右对齐，如必要可在前缀加一些空格。

(4) 对于输出，所要输出的字符超过格式指定的域的宽度（字符个数）或指数超过规定的宽度时，则整个域充满*号(即输出*号)。

2. 可重复的编辑符的含义

(1) I——整型编辑符

IW中的I是对整型变量的编辑，W是域宽，即整型的数值输入输出的位数。

(2) F——实型编辑符

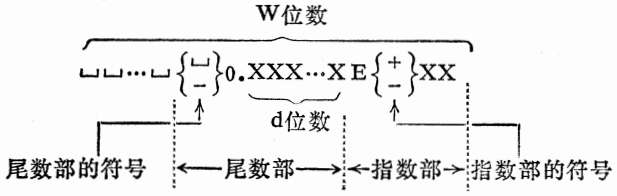
实型编辑符 FW.d 中的F是对小数实型数值编辑的指定，W是小数实型数值的域宽即全位数。d是小数点后的位数。在输出的情况下在满足全位数的情况下，多余位第一个数被四舍五入，而不是被截去。在输入情况下，有小数点出现输入实型数值，d不起作用。

计算全位数时，将数值的符号位正（+不出现）或负“-”计算在内。

3．E的实型编辑

E的实型编辑符用EW·d或EW·dEe形式。域宽度都用W值指定，对输入e没有影响。E的实型编辑符是指数形式的实型编辑说明符。输入域的形式依赖于比例因子 P。如果比例因子为0，则输入域是负号“-”，后跟小数点，再跟一串数最后为指数域。

EW·d的编辑使I/O结果形式如下：



其中␣：为空格

X：为数字

EW·dEe与EW·d的区别只在尾数部分，用e来表示±号后的位数，这种编辑符对中大型机，能表示较大数值范围。

请注意：在使用E编辑描述符时，一定要考虑到 I/O 过程的数值范围在单精度情况下，不超过机器允许的值(见2-1-3部分)。

4．L的逻辑编辑

L是逻辑编辑描述符，是对iolist表中的实体项逻辑类型，具有实现I/O 过程的格式编辑功能。W为域宽，即W个字符。对于输入，W域可为空格后跟一小数点后跟 T(表示真) 或F(表示假)组成，其它字符都被忽略。这样说来，只要在指W字符为1就能包括 TRUE或 FALSE头一个字母T或F，输入过程就当成真值 (TRUE) 或假值 (FALSE) 接受。输出时，会出现W－1个空格后跟T或者F。

5．A的字符编辑

字符编辑的格式为A或AW。对于字符编辑A来说，隐含一个W域宽。这个域宽为 iolist表中与格式指定符A相关联的项的字符个数。关联项必然是字符型。如果有W域宽说明符，并且是等于或大于 iolist 表中元素的字符个数时，则把要输入字符全作为输入字符，在不足W的个数，由左对齐，末尾由空格补全。对于输出，W大于 iolist 项所需要的个数时，则在字符前头补空格。否则取左边的W个字符作为输出。

三、走纸控制编辑

传送给打字机或显示器的每个记录的第一个字符，在 IBM FORTRAN I/O 中作为走纸控制编辑符。其字符与作用如下：

字　符	作　用
空格	前进一行
0	前进二行
1	前进到下一页的第一行
+	不前进(就打字机头所在位置打印)。

除上述字符外，其他任何字符在记录第一字符位置上，都作为空格而控制走纸或控制显示走一行。

注意：如果你偶尔忽略了走纸控制符，那么记录的第一个字符就不能被打印出来。

第三章　PASCAL语言

3-1　概述

PASCAL是1970年由瑞士计算机科学家Niklaus Wirth 设计的。其特点是适合于讲授程序设计系统；能够非常有效地实现结构程序设计。因此PASCAL语言既是一种通用的程序设计语言，又是一种系统程序设计的有力工具。

IBM PC的PASCAL语言与ISO 草案 (ISO/TC 97/SC 5N595) 相符合；但又具有自己的特征和扩展部分。因此，特别适用于编写编译程序，解释程序和操作系统。

由于PASCAL语言的各项扩充，减小了编译过程所需要的时间与空间，故生成程序效率高。

IBM PC的PASCAL能够象汇编语言一样，很容易完成某些操作。

目前，在IBM PC和APPLE II都配有PASCAL。在CROMEMCO 机上还没有PASCAL。又因篇幅限制，在这一章里只就IBM PC PASCAL语言予以简介。有关APPLE II的PASCAL资料较多，我们在这里就不再介绍了。

3-2　IBM PC扩展PASCAL

IBM PC的PASCAL 扩展有：编译命令、属性、超数组类型、字符串的并置处理、常量值、系统实现功能等。

3-2-1　编译程序命令　编译程序共有30条“元语言”命令可供使用，提供了产生出错校验码的控制，列表格式控制，条件编译构造和把其他源文件插入编译的手段。

3-2-2　程序单位　PASCAL对于用来编制大型的可靠程序（如编制系统软件）是一种很好的语言。IBM PC的PASCAL 以程序单位来进行编译的。程序单位是有关的数据类型、变量、常量、过程、函数以及初始化的各种模块（程序段）的集合。它有两部分：接口 (Interface) 和实现 (Implementation)。

接口：是一个供出口标识符及其说明部分（包括过程和函数首部）的表格。

实现：包括局部说明、过程和函数体，及初始化代码；一些可以用汇编程序而不用PASCAL实现的程序段的例行程序。

一个程序，或者是实现，或者是其他接口，使用时给出接口而不是实现的单位。这种方法提供了比用外部过程和变量的办法更能结构化，即程序能划分为几个模块的形式。

3-2-3　属性　变量、过程、函数的属性给用户提供了在链接翻译程序级上的控制。相应的属性包括：

1.PUBLIC和EXTERN是供全程标识符链接用；

2.STATIC和READONLY是供变量用的；

3.PURE是供过程和函数用的。

3-2-4　超数组　IBM PC PASCAL 提供了可以改变数组长度的超数组类型。使用时，超数组类型下界要给出，上界是不限定的。

超数组类型虽然不能直接用于变量，但可以用于以下两个重要方面：

1.用于引用形式参数类型；

2.用于引用指针类型。

另外，任何变量域可以从使用“命名符”的超数组那里推出类型。例如：

```
TYPE
  VECT = SUPER ARRAY[0..*]OF REAL;
VAR
  PVEC: AVECT; V10:VECT(10);
PROCEDURE SORT (VAR V:VECT);
BEGIN
  NEW (PVEC, UPPER(V));
    ⋮
END;
    ⋮
SORT(V10);
```

其中VECT是超数组类型，PVEC是指向实际类型的指针。NEW用来指定新的VECT，其上界用第二个参数表示，在本例中上界同参数V是一样的；V10是变量，VECT 的上界实际上为10。参数V可以从VECT那里导出任一类型变量。例如：V10或PVEC。

超数组类型能以简洁高效方法处理动态数组和共形数组。

3-2-5 字符串 在标准的PASCAL中，字符串的长度是固定的。IBM PC PASCAL 字符串类型的字符串的长度是可变的，加强了字符串变量的能力。而且常常使用紧缩的抽象数据类型。如：

字符串数型

SUPER PACKED ARRAY [0•••] OF CHAR

其中，数组元素0存放字符串长度；也包括字符过程和字符函数集。

IBM PC的PASCAL有一个字符串的并置CONCAT 过程，它需要第二个与第一个参数有关联的两组字符串参数。

这就使字符串的处理很方便。只要有：

1.说明字符串类型是定长的字符数组；

2.用超数组或其他共形数组再扩充一下，在PASCAL中用所变长度的字符串类型。就能用PASCAL来编写所有的任何过程和函数了。

在IBM PC的PASCAL中，字符串的赋值、比较和读/写操作都是自动完成的。

3-2-6 常量值 IBM PC PASCAL 常量值的绝大部分是在编译过程求表达式的值。所以，如果 WORDCOUNT 和 SYMBOLCOUNT 是常量的话，就可定义另一个常量：

TOTAL COUNT = WORDCOUNT + SYMBOLCOUNT。

常数可用二进制、八进制、十六进制表示。在READ和WRITE语句中也可使用这种表示法。

此外，字符串常量还可连接运算符，数组和记录常量。

一个程序可以包含一个或多个VALUE段，各变量均以初始常量值表示。

一个形式参数可位于关键字CONST的后面，其作用和VAR一样，只是形式参数可为一个常量，而不能在程序体运行时改变。

3-2-7 系统实现 系统实现功能的扩充，有如下几个方面：

1.WORD(字)类型

WORD类型是系统实现功能扩充最重要的方面。实际上，字类型是从0到最大65535(MAX WORD) 的一个子界。正如INTEGER是－32767到32767 (MAXINT) 的一个子界一样。

16位二进制值，可以带符号位，或不带符号位，因此数的实际范围是－32768到65535。在PASCAL程序中，具有WORD和INTEGER两种类型允许由－32767到65535范围，即使不使用这两种类型也允许有此范围。

不带符号的字在系统实现功能中，常用于作地址，或者具有最大值的整数，或者是位的集合，或者作为不知语义的存贮值。

当用到比较问题（#FFFF是否大于#FFFE）时，就要用整型INTEGER类型。

另外象逻辑运算符AND，OR，XOR允许使用WORO类型。

2.ADDRESS类型

ADDRESS是地址类型，是增加的新类型，它与指针类型相似，但又允许分段寻址。这种地址类型对于存取系统数据区是很有用的。

3.其他功能的扩充

系统扩充的其他功能还有：

(1) 输入/输出能力；

(2) 交互式的READ语句；

(3) 随机文件和文件方式；

(4) 内部过程与函数。

3-2-8 结束语 IBM PC PASCAL的许多特性有助于以结构方式建立和维持PASCAL特点，对系统程序设计工作又是完全必要的。特别显著的特性有：特殊的编译单位，可变长度的字符串，超数组类型和面向机器的结构等。因此，能够用IBM PC PASCAL编制系统、复杂的程序。IBM PC PASCAL不仅是系统软件的工具，而且还是一种生成程序。

注意：本章余下部分中，将IBM PC PASCAL简称为IBM PASCAL。

3-3 PASCAL语言的级别

IBM PASCAL可分为四个“级别”：

3-3-1 元语言 元语言用来设置编译程序的可选项和进行有选择条件的编译。

3-3-2 标准PASCAL 所有标准的ISO PASCAL程序，使用IBM PASCAL都能正确地进行编译和运行。

3-3-3 扩展PASCAL IBM PASCAL提供了相对“安全”的扩展功能，从而使IOS PASCAL功能有所加强，其扩展包括象BREAK语句，UNIT（单位）和结构常量那样的结构等，因此能使程序设计的更好，更容易。

3-3-4 系统PASCAL IBM PASCAL还提供了其他更完善的扩展，包括用于系统程序设计有用的或必要的功能，象地址类型和RETYPE函数。

3-4 语法和词汇

本节对PASCAL语法只作一般性介绍。语法的实例只是用来帮助理解，并不能形成完整的语法。

PASCAL的基本词汇是由字母、数字、保留字和专用字符组成的。小写和大写字母可以互换，但在字符串文字中小写和大写不能互换。

3-4-1 专用字符类别

一、元语言的前缀

用$字符作元语言的前缀；以此标识元语言。

二、标准PASCAL符号

+ − * / = < > ( ) [ ] . , : ; { } ‹ › ‹= ›=
:= ∧ 。在标识符中允许使用“—”下划线。

三、替换符号

标准PASCAL的符号，在IBM PC上不能实现的部分符号，用替代办法解决；用(*…*)代替{…}；用(*代替[，用*)代替].用？号或@代替指针符号∧。

四、更高级替换符号

用#字符表示整型常数(无小数的数)；

用！表示行的结束；

用[…]代替BEGIN…END。

五、无用字符

对于IBM PC PASCAL 没有用的符号有：% & “ | ～ 、 ＼等。

空白（空格）符、制表符和换页符也是标准字符集的一部分。制表符CHR(9) 看成空格符并可传送到文件清单上。换页符CHR(12)被转换成新页元命令($PAGE+).

除了CHR(9)和CHR(12)以外，CHR(0)到CHR(31)中的其他字符和从CHR(127)到CHR(255)字符，以及上面列出的无用字符(%，…，＼)等，如用到源文件上时，就会产生错误。但是允许在注释中或字符串中使用。

3-4-2 PASCAL保留字

一、标准PASCAL的保留字

AND	END	NIL	SET
ARRAY	FILE	NOT	THEN
BEGIN	FOR	OF	TO
CASE	FUNCTION	OR	TYPE
CONST	GOTO	PACKED	UNTIL
DIV	IF	PROCEDURE	VAR
DO	IN	PROGRAM	WHILE
DOWNTO	LABEL	RECORD	WITH
ELSE	MOD	REPEAT	

上面列出的35个保留字在IBM PASCAL仍然有效。

二、新增加的保留字

在IBM PASCAL中还增加一些特性保留字。有：

单位接口	IMPLEMENTATION
	INTERFACE
	UNIT
	USES
模块	MODULE
扩展CASE	OTHERWISE

超数组类型	SUPER
控制流	BREAK
	CYCLE
	RETURN
扩展运算符	XOR
地址类型	ADR
	ADS
	VARS
数值段	VALUE

三、属性保留字

属性（Attributes）是表示变量、过程或函数的特性的保留字。下列的一些为属 性保留字：

EXTERN	PUBLIC	READONLY
EXTERNAL	PURE	STATIC

四、命令保留字

命令是在过程和函数块中使用的保留字。如：

EXTERN EXTERNAL FORWARD

其中EXTERN既是属性又是命令。EXTERNAL是EXTERN的同义词，具有同其他几种PASCAL兼容的能力。

五、标准PASCAL中的标识符保留字

标准PASCAL具有的标识符保留字仍适用于IBM PASCAL。有：

ARCTAN	FALSE	OUTPUT	SIN
ABS	FLOAT	PACK	SQR
BOOLEAN	GET	PRED	SQRT
CHAR	INPUT	PUT	SUCC
CHR	INTEGER	READ	TEXT
COS	LN	READLN	TRUE
DISPOSE	MAXIN	REA	TRUNC
EOL	NEW	RESET	UNPACK
EOLN	ODD	REWRITE	WRITE
EXP	ORD	ROUND	WRITELN

六、扩展的内部特性保留字

ABORT	EVAL	RESULT
BYWORD	HIBYTE	SIZEOF
DFCODE	LOBYTE	UPPER
ENCODE	LOWER	

七、字符串内部特性保留字

CONCAT	POSITN	COPYLST
DELETE	SCANEQ	COPYSTR
INSERT	SCANNE	

八、系统内部特性保留字

FILLC　　MOVER　　FILLSC　　MOVESR
MOVEL　　RETYPE　　MOVESL

九、扩展I/O特性保留字

ASSIGN　　DISCARD　　READFN　　SEQUENTIAL
CLOSE　　FILEMODES　　READSET　　TERMINAL

十、系统I/O特性保留字

FCBFQQ

十一、字类型特性保留字

MAXWORD　　WORD　　WRD

十二、超数组类型特性保留字

LSTRING　　NULL　　STRING

3-4-3　注释符　{　}或(*　*)为注释定界符，括在符号中间的部分，为注释部分，可以延续多行。延续行用“!”开始，使用相同的定界符

3-4-4　分隔符　注释、空格或行边界只能出现在两个相邻的数或相邻的保留字和标识符之间。但是不能在它们中间嵌入。

在某些情况下，IBM PASCAL 可以接受没有分隔符的信息，且不给出 错误信息。如：

100mod＃50　　作为　100mod＃50 来接受
100mod127　　作为100后边跟标识符mod127接受

3-5　标识符和常量

3-5-1　标识符

一、定义

标识符是常量、类型变量、过程、函数、程序和记录中的字段和标记区段的命名的字符串名称。有些特性，如超数组类型、模块、单位和语句标号也有标识符。

语句标号使用不带符号的整数或标识符，并具有与标识符一样的规则：前缀的 0 是没有意义的。

标识符是由字母后边跟字母、数字、或横杠“—”所组成。这个横杠—(下划线)是有意义的。允许标识符字符串中间带有“—”是对IOS PASCAL的一点扩充。IBM PASCAL允许在一行之内有两个“—”或使用“—”结束标识符。

二、长度限制

标识符的长度是有限制的，一个标识符只有前31个字符有意义，大于31字符就会出现警告信息。每个标识符必须在一行上。

传送给链接程序的标识符包括用于传送给链接程序的程序，模块或程序单位，以及那些带有PUBLIC或EXTERN情况的标识符。其长度也可以是31个字符长。

如果标识符不是链接程序使用的话，在反汇编目标代码中，变量和过程标识符截成前 6 个字符有效。

使用小于 7 个字符的标识符在编译时可节省存贮空间。

运行时，I/O系统所用的所有外部标识符为 4 个字母后跟“QQ”形式。因此，用户在构成外部名时，应该避免使用这种格式。

三、作用域

1.标识符域

标识符的作用域为标识符所起作用的范围。标识符的作用域与IOS PASCAL的作用域一样。标识符在所起作用的域内必须是唯一的。

标识符的作用域，就是在这个作用域内定义的过程、函数、程序、模块、单位、实现、接口和随意嵌套的过程或函数。

一个嵌套的过程或函数可以重新定义--个还没有使用过的标识符。因此，如M是程序中的一个整型常数，那么，嵌套的过程就不能执行下述一段程序：

```
CONST  Q = M + 1;  M = 4;
```

2.过程或函数域

过程或函数的作用域，是在过程或函数标识符后就开始了。因此，作用域包括形式参数表。这意味着过程或函数标识符本身比它的参数、变量以及嵌套的过程和函数标识符"高"一级。

例如，允许用与函数相同的标识符来说明函数中的变量。

尽管预说明标识符，程序、模块和单位的标识符都比用户其他的标识符"高"一级。但使用已预说明标识符，象INTEGER或READ作为其他用场，仍会产生重复标识符的错误。

3.字段或标记区段

字段（或为标记区段）作为记录的标识符，它的作用域是字段命名符或使用记录类型的WITH语句。如果使用$LIST+，当前标识符层次可以在清单的I栏中找到，并把它看作为所提供的16个标识符层次之一。

除了用于程序参数和接口的结构成分，以及参考的指针类型外，标识符必须在使用之前定义。

当定义以指针类型^T为参考（或ADR OF T或ADS OF T）时，在引用的类型T过后，可以在同一TYPE段内出现，允许间接的递归。

4.FORWARD命令域

标识符在以FORWARD命令打头的过程或函数中是参数，在遇到实际的过程或函数块前不起作用。说明FORWARD命令之后，允许对过程和函数进行间接递归。

当指定过程或函数的参数时，所用到的参数标识符是无作用的。

用单位接口特性的USES子句，在接口或在USES子句本身处定义一组标识符，这些标识符是在接口中定义的"单位"的"组成部分"，而实际上是由实现引起的。

例如，USES (IVAR, COLOR, COLORPROC)，在表中定义三个标识符；他们可以是常量、变量、类型、超数组类型、过程或函数。

如果COLOR是枚举类型（RED, BLUE, GREEN），那么这些枚举类型的常量标识符不是自动定义的；它们也必须在接口中列出。组成记录成分的变量或类型的字段标识符可以用在字段命名（"."）中，或常用于WITH语句中。

3-5-2 常量

一、数值常量

通常的十进数可以表示实型（REAL）或整型（INTEGER）或字型（WORD）的常量。REAL型就是带有小数点或带指数的数，否则就是INTEGER或WORD型。

INTEGER整型常量可以在-MAXINT到MAXINT（-32767到32767）范围内。

WORD字型常量可在0到MAXWORD（0到65535）范围内。

整型、实型如超过允许的数值范围就会出错。

任意INTEGER类型的常量（其中包括由－1到－32767变化的常数和表达式值），INTEGER一旦是负值时，则加上65536其绝对值不变。能自动转变成WORD类型的常量。

WORD类型的常量子界可以说明成WRD(0)‥127；上界自动表示为类型WORD。反之不成立。类型WORD的常量不能自动地转变成类型INTEGER的常量。ORD函数和WRD函数也可由有序常量的类型转变成INTEGER或WORD常量。例如：

0	类型INTEGER，可变成WORD
32768	类型WORD
0‥20000	类型INTEGER的子界
0‥50000	类型WORD的子界
0‥80000	无效
－1‥50000	对INTEGER无效，可转变成65535‥50000

实型REAL常数的范围有7位精度，其最大值约为1.70141200E＋38。小数点的两边至少要有一位数字，否则会给出警告信息。

一个实数如果小数点打头或结尾可能会产生误解；如，（.1＋2.）会认为是〔1＋2〕，因为“(.”和“.)”分别代表“〔”和“〕”。

允许在实数中使用科学记数法。即用指数形式表示时，小数点和指数符号是随意的。用E或e描述。

数的前缀可为符号＋或－的，如ALPHA＝＋10是有效的，但ALPHA＝＋－10是无效的。不允许在符号后或指数部分插入空格。

编译程序截去任何多于31个字符的数的尾部字符（与标识符长度一样）并给出警告信息。

数的语法适用于文本文件以及程序；所有的数都由编译程序使用DECODE函数加以转换。数值常量如：

123	＋12.345	0.19	008
－26.0	26.0e7	－1.7E－12	1.2E＋3
17e＋3	1e1	1.5e＋4	3.75e－2

IBM PASCAL还能支持以十六进制，八进制，二进制为底数表示的数。十六进制允许由0～9和A～F（大写与小写都行）16个字符记法。如十六进制记法为16＃FF02，八进制为8＃776，二进制为2＃11100，十进制为10＃987。允许底数标识位前有零，如008＃147。如果没有底数标识位时，当作十进制数处理。非十进制记数法不适用实型REAL常量或语句的数字标号。

允许在常数出现的任何地方，都可直接使用常量表达式，但不能使用在无命令中。

常数可以出现在CONST段、表达式类型子句、常数集和其他结构常量之中，还可以在CASE语句的CASE常量和可变记录特征值中出现。

二、常量运算符和函数

1.实型或整型操作数的常量运算符

具有REAL或INTEGER类型的常数运符有：

一目运算符有＋和－。

2.整型或字类型操作数的常量运算符

具有 INTEGER 或 WORD 操作数的常量运算符有：

+	DIV	OR	函数有：HIBYTE ( )
−	MOD	NOT	LOBYTE ( )
*	AND	XOR	BYWORED ( )

具有有序操作数的常量运算符有：

<	>	函数有：ORD ( )		LOWER ( )
<=	=	CHR ( )		UPPER ( )
>=	<>	WRD ( )		

3．具有布尔型操作数的常量运算符

有三种：AND　　OR　　NOT

4．具有任意操作数类型的常量函数有：

SIZEOF ( )　　RETYPE ( )

三、字符串 (STRING)

在字符集内的有穷字符序列称之为字符串。在这里为了与字符串常量相区别，称字符串为字符串文字或 STRING 类型。

一个字符串常量可在 1 到 255 个字符范围以内。1 个字符串常量比他自己的紧缩类型 PACKED ARRAY[1..n] OF CHAR 长，也称为类型 STRING(n)。只含有一个字符的字符串常量是属于类型为 CHAR 的常量。如果需要进行赋值，传递参数等，可以把类型 CHAR 变为类型 PACKED ARRAY[1..1] OF CHAR，即是 STRING(1)。

空字符'　'是不允许的。如使用“不能用”的字符串文字，系统能够识别出，并会给出警告信息。

字符串常量表达式，允许包括字符串常量标识符和 CHR ( ) 函数，并置其他字符串常量组成新的字符串常量。常量可以跨越多行，但最多字符个数仍限定为 255 个。

符号 * 是并置运算符。允许在字符串常量表达式或字符串文件可以出现的地方存在。但不能在元命令中出现。

注意：并置运算符 * 只适用于常量，而不适用于字符串变量。

字符串表达式的实例是：

```
HEADER='NOW''S the time'
POLITC=HEADER*CHR(13)*'FOR all good men'
```

四、长字符串 (LSTRING)

LSTRING 是超数组类型或称长字符串。它类似于 PACKED ARRAY[0..n]OF CHAR。0元素放字符串的长度，可在 0 到 n 的最大值之间变化 (n≤255)。如果需要进行赋值等，类型为 STRING(n)或 CHAR 的常量可以自动地转变成类型为 LSTRING 的常量。

NULL 是给空字符串预先进行说明的常量，其元素 0 等于 CHAR(0)。因为 NULL 不是字符串 STRING 故不能串联。NULL 仅是 LSTRING 的常量。用户象不能说明结构常量一样，不能去说明 LSTRING 常量。

五、定义常量

定义常量含义就是定义代表引入一种常量的标识符。其格式为：

常量标识符 = 常量

这种定义是用在程序、过程、函数、模块、接口或实现的 CONST 段中。常量标识符在处

理定义语句之前是没有定义的，因此，N=N+1这样的常量是无效的。常量标识符也可用，$INCONST元命令定义。

六、结构常量

结构常量特性允许特定类型的数组或记录作为集合常量或无名的隐含类型的集合常量。

结构常量可以在允许结构值的地方，在CONST和VALUE段中使用。

数组或记录的常量是由类型标识符跟着括在方括号内，由逗点分开常量值表所组成。

集合常量是由随意的集合类型标识符跟着括在方括号内，由逗点分开的集合常量元素所组成。集合常量元素可以是一个或两个有序常量（用两个点分开以表示常量值范围）。例：

```
TYPE TVECT = ARRAY [-2..2] OF INTEGER
CONST VECT = TVECT(5, 4, 3, 2, 1);
TYPE TRECR = RECORD
                A, B :BYTE;
                C, D : CHAR
             END;
CONST RECR = TRECR (#20, 0, 'A', CHR(20));
TYPE TSETC = SET OF(RED, BLUE, WHITE, GREY, GOLD);
CONST SETC = TSETC [RED, WHITE..GOLD];
```

数组中的和记录结构中的常量，必须指定有一个能够对应的相关的类型。对应于标记区段的常量的值，即使是空的标记区段也一样，只是选择情况不同。

除了超数组类型的结构常量以外，常量元的个数必须等于结构中数元的个数

由于嵌套在另一个结构中的数组或记录常量必须有前置类型的标识符，超数组类型只能是一维的。

表示结构常量的长度必须为1到255个字节。结构常量的实例是：

```
TYPE R3 = ARRAY[1..3] OF REAL;
TYPE SAMPSLE = RECORD
                  I : INTEGER;
                  A : R3;
                  CASE BOOLEAN OF
                  TRUE :(S :SET OF'A'..'Z'; P :^SAMLE);
                  FALSE :(X :INTEGER)
               END;
CONST S1 = SAMPLE(27, R3(1.4, 1.4, 1.4);
TRUE, ['A', 'E', 'I'], NIL);
```

常量元可以用“DO n OF 常量”来重复，因此，上例中的“1.4, 1.4, 1.4”可以用DO 3 OF 1.4来代替。

字符串STRING(3)常量为'ABC'等价于结构常量STRING('A', 'B', 'C')。这里不允许LSTRING结构常量。

有两种集合常量，一种是具有显式类型， 如在CONST SETC = TSET[IRED, WHITE..GOLD]中的常量；另一种是带有未知类型常量，正如在 CONST RANGE = [20..40]。集

合常量可用在表达式中或者用以定义常量标识符的值。

有显式类型的那种常量也可作为引用（CONST）参数来传递。

七、关于常量的注意事项

定义有序类型的标识符常量，不能直接同数值或字符常量表达式联用。不过可以同ORD、WRD或CHR函数联用，如在12＋ORD(BLUE) 中的标识符。

布尔类型BOOLEAN 预先定义了常量为 TRUE 和 FALSE。MAXINT，MAXWORD 和 NULL也是预先定义的。由于NIL在ISO PASCAL 中是指针类型，是一个保留字，所以不能由程序员再定义。

数值语句标号与数值常量毫不相干，用户不能使用常量标识符或表达式作为标号。包括字符串表达式和结构常量在内的所有常量的内部表示都限制在1到255个字节。

3-6 数据类型

在程序中出现的每个变量，必须与一种数据类型相对应，并与类型的标识符相关联。有简单类型（整数、实数、字、字符、布尔、枚举、子界），和由简单类型构造起来的结构类型（数组、字符串、记录、集合和文件等）。对于IBM PASCAL的结构类型还有动态指针类型、地址类型和过程类型。

数据类型的定义是在程序、过程、函数、模块、接口、或者实现的TYPE 段中建立的。

定义类型的一般格式是：

类型标识符＝类型

类型标识符在处理后才定义，所以除了引用类型外，象T＝ARRAY〔0••9〕 OF T 类型定义是无效的。

3-6-1 简单类型

一、INTEGER(整数类型)

在必要时，INTEGER类型常量可转换成WORD类型常量；可把INTEGER的值（变量、常量或函数返回）转换为用表达式表示。

REAL（实数类型）的ORD函数将任何有序类型变成INTEGER类型。

二、WORD（字类型）

由于下述原因，导致WORD类型：

1.需要有32768到65535之间的数值；

2.需要用操作系统和机器结构作接口；

3.寻址操作的需要；

4.不需要用 INTEGER类型和陷入－32768值时，而能方便地获得机器的原语。

用汇编语言常用到－32768到65535之间的数，在使用INTEGER值能自动转换成 WORD类型（3-2-2节）。使用结构方法，既可是INTEGER类型又可具有WORD类型。但是这两 种类型不能在同一表达式中出现（整常数除外），赋值不具有兼容性。

一个有序类型的值可利用WRD函数变为WORD类型的值。

三、CHAR（字符类型）

CHRA类型的值为一字节ASCII码值，支持SET OF CHAR。在类型CHAR 中可以使用全部256字节值。关系比较要用到ASCII码排列顺序。

在文本文件TEXT中，使用“行标记（Line marker)”字符不属于 ISO—PASCAL 中的

CHAR类型部分。但是，在IBM PASCAL中，CHR(13)—回车〈CR〉——可以作为"行标记符"，CHR(26) 为文本文件结束标记。

CHR函数能将任何有序的类型转变成CHAR类型，取值与ORD的范围一样，即0—255。

四、BOOLEAN (布尔类型)

布尔类型值为真和假 (TRUE和FALSE)。BOOLEAN类型是枚举 (enumerated) 类型的一种特殊情况；而且ORD (FALSE) 为0而ORD(TRUE) 为1。在PASCAL 中有 FALSE＜TRUE。

五、Enumerated Types (枚举类型)

Enumerated类型是通过列举一些值的标识符来定义一组有序的值。这些值就是枚举类型的常量。例如：

```
TYPE
COLOR = (RED, BLUE, GREEN, YELLOW, PURPLE, ORANGE,
         GOLD);
SUITS = (OLUB, DIAMOND, HEART, SPADE);
```

所有枚举类型常量的标识符在其说明范围内必须是唯一的。所枚举标识符自左向右排序，由0开始，如上例中ORD(RED) = 0，ORD(GREEN) = 2等。

枚举类型对于表示操作名，事物种类名，命令名等抽象名字的集合特别有用。因为枚举类型是有序的，所以象RED＜GREEN比较之后会有TRUE结果。

LOWER和UPPER函数允许在FOR语句里访问枚举类型中的最低值和最高值。如：

```
VAR  TINT : COLOR
FOR TINT : LOWER (TINT) TO UPPER(TINT) DO PAINT(TINT)
```

六、Subrange Type(子界类型)

子界类型是定义一个有序类型的下界和上界。下界不得大于上界，但二者可相等。

子界类型的格式为：

常量··常量

如：100··200

'A'··'Z'

RED··YELLOW

'B'··'B'等等。

常量可以是整数、字符、字或枚举类型，在使用枚举类型作为上、下界时，必须在子界类型定义之前定义过。常量表达式可以作为子界的上、下界。

要预先说明的子界类型有两种：

1.BYTE = WRD(0)··255；{8位(二进制数的)WORD的子界}

2.SINT = －127··127；　　{8位(二进制数的)INTEGER子界}

七、REAL (实数类型)

实数具有一个24位二进制尾数和 8 位二进制指数，其精度约 7 位十进制数及最大值约为 1.70141200E＋38

3-6-2　结构类型　由简单类型构造出来的结构类型。可以以PACKED 紧缩，节省存贮空间。数组(ARRAY)、记录 (RECORD)、集合(SET)或文件(FILE) 等结构类型的最大长度为32766(MAXINT-1)个字节。

一、ARRAY（数组类型）

数组是由一个下标类型和一个基类型（也称分量类型或元素类型）所组成。下标可为多个，所以数组可为多维的。例如：

```
ARRAY〔1··10〕 OF INTEGER;
ARRAY〔0··9〕OF ARRAY〔0··99〕OF 0··999;
ARRAY 〔0··9, 0··99〕 OF 0··999;  (*same as above*)
ARRAY 〔COLOR〕 OF(PRETTY, TACKY) ;
```

所有 n 维 PACKED 数组可以紧缩为：

```
PACKED ARRAY 〔1··2, 3··4〕 OF C;
```

等效于

```
PACKED ARRAY 〔1··2〕
OF PACKED ARRAY 〔3··4〕 OF C;
```

用保留字 ARRAY 作前缀定义数组，数组下标所表示的元素类型可为简单类型或者结构类型。但是，在同一数组中，所有元素，必须是同一数据类型。

二、SUPER ARRAY（超数组类型）

超数组用保留字 SUPER 作数组前缀来定义，下标的下界已给定，而上界用 *，表示是可变的，下面定义的例说明超数组形式与用途。

```
TYPE
     VECTOR = SUPER ARRAY 〔1···〕 OF REAL;
     MATRIX = SUPER ARRAY 〔1···, 1···〕 OF REAL;
     VECT10 = VECTOR(10); MATDEC = MATRIX(100, 100);
  VAR
     ROW : VECT10; COL : VECTOR(10);
     ROWP : ∧VECTOR;
```

超数组类型的标识符只能作为引用参数而用在某一标识符之后，或者接在一个类型子句或一个命名符或一个结构常量的∧号之后。

三、超数组参数—共形数组

在超数组中的形式引用参数用超数组类型。

实在参数可以是：

1. 超数组变量本身（只能在过程与函数中出现）或指针引用；
2. 由超数组类型导出的类型变量或常量。

上述这种类型参数用法在 PASCAL 文献中称之“共形数组（conformant array）”。例如：

```
TYPE VECTOR = SUPER ARRAY 〔1···〕 OF REAL;
VAR  X : VECTOR(12);
     Y : VECTOR(24);
     Z : VECTOR(36);
     (*X, Y, and Z are derived from VECTOR,
     the super array*)
      ⋮
FUNCTION SUM (VAR V : VECTOR) : REAL;
```

```
    (*V, the formal reference Parameter, is the conformant array*)
VAR S : REAL; I : INTEGER;
    FUNCTION SUB(VAR V : VECTOR) : REAL;
    VAR S : REAL; I : INTEGER;
    BEGIN (*SUB*)
    S : = 0;
    FOR I : = 1 TO UPPER(V)DO
    S : = S - V[I];
    SUB : = S
    END; (*SUB*)
    BEGIN(*SUM*)
    S : = 0;
    FOR I : = 1 TO UPPER (V) DO
    S : = S + V[I];
    SUM : = S + SUB(V)
    (*V is a super arrag type being passed to the nested function SUB*)
    END; (*SUM*)
BEGIN (*MAIN*)
  :
 ZERO : = SUM(X) + SUM(Y) + SUM(Z);
  :
END. (*MAIN*)
```

在 FUCTION SUM 中允许语句 SUB(V)存在，因为在传递 V 之前，V 已经赋给了一个实际的数值。

四、Dynamic ARRAY(动态数组)

超数组类型的指针允许在堆上，分配特定容量的数组（要求的数组的实际上界传送给 NEW 过程），以 NEW 指定的超数组是不初始化的，所以当含有文件时，将发出警告信息。这种类型的堆变量在 PASCAL 文献中，常常称之为“动态数组”。例如：

```
    VAR PST : ∧LSTRING; (*PST is a pointer referece type*)
        :
    NEW(PST, 10); (*Upper bound of LSTRING will be 10*)
```

超数组类型的变量只能以形式参数或以指针引用类型的形式出现，不能在变量段 VAR 中说明或作为一个值参数。但是，允许用超数组类型来导出类型变量或常量。

即使其他变量具有同一超数组类型，具有超数组类型的变量也不能与其他类型相兼容或赋值兼容。

因为字符串 STRING 超数组类型的长度不变，所以不允许进行比较和赋值。

LSTRING 允许使用可变长度的字符串，在 LSTRING 中的字符可用字符表示法存取。长度放在第一个元素位置上，可以从 0 变化到 n。

LSTRING 的变量 T 的长度是 CHAR 类型，就可用 T(0)存取；或者变量是 BYTE 类型就可用 T. LEN 存取。例如：

```
VAR Y:INTEGER;
    X:BYTE;
    CH:CHAR;
CH:=T(0);
Y:=ORD(CH); (* Y will now contain the leng th *)
X:=T•LEN; (* X will also contain the length*)
```

类型CHAR或STRING(n)的字符串常量能自动变为LSTRING类型。

过程与函数同样可以使用STRING和LSTRING。

五、RECORD (记录类型)

记录类型是由若干不同类型的分量构成。定义记录类型就是指定称之为字段的每一个分量，分量的类型及类型的标识符。分量标识符的作用域就是记录本身。故分量标识符必须在定义中是唯一的。通过分量的标识符进行访问。为了能适应复杂对象对数据类型需要，可将记录变成，具有变体类型记录。变体记录分成固定部和变体部。如：

```
RECORD
  NAME:LSTRING(30);
  PHONE:RECORD
          AREA, PREFIX, EXTEN :INTEGER
        END;
  RIPE:BOOLEAN
END;
RECORD
   X, Y:REAL;
   CASE S:SHAPE OF
       SQUARE : (SIZE, ANGLE :REAL);
       CIRCLE : (DIAMETER : REAL)
END;
RECORD
  CASE BOOLEAN OF
    TRUE : (I, J :INTEGER);
    FALSE : (CASE COLOR OF
        BLUE: (X :REAL);
        RED : (Y: LONGINT))
END;
```

一个记录只能有一个变体部分，变体部分还可有变体，以此类推。变体部分必须在记录的末尾。

在变体子句中，可以支持 CASE 语句常量可选项；即，常量表可以定义一种情况。象 POINT :( )的空变体是允许的，并常常是有用的。

在最后的例子中，因IBM PASCAL不指定BOOLEAN的标记字段，对I不进行检查。

IBM PASCAL 对引用参数传递的记录，或由WITH 语句使用 DISPOSE 记录的错误不捕获。

变体记录以下面两种错误，编译是允许的，但会产生警告信息：

(1) 说明含有文件变体是不安全的，因变体的字段来改变该文件数据，即使是文件关闭，也会导致I/O错误。

(2) 对VALUE 段的某一个变量的几个重叠变体给出初始数据，可能会得到不可预测的结果。

对某些不安全操作，象重叠字段，字值在奇数节边界处等等，也允许。除非需要考虑到接口，否则，这些操作才是不可取的。例如：

```
TYPE ABC = RECORD
                NDRIVE [00] : BYTE;
                FILENM [01] : STRING (8);
                FILEXT [09] : STRING(3);
                EXTENT [12] : BYTE;
                ABCRES [13] : STRING (20);
                RECNUM [33] : WORD;
                RECOVF [35] : BYTE
           END;
     FOO = RECORD
                BYTEAR [00] : ARRAY [0..7] OF BYTE;
                WORDAR [00] : ARRAY [0..3] OF WORD;
                BITSAR [00] : PACKED ARRAY [0..63] OF 0..1
           END;
```

注意：任何大于一个字节长度的都整成偶字节长度。

例如：

```
EXM = RECORD
        A [1] : STRING( 3 );
        B [ 4 ] : WORD
      END;
```

abc赋给字段A，再写在B的第一字节上。

用位移量来表示任何字段，则所有字段都必须用位移量表示，即使没有给出位移量，编译程序也会采集位移量的。如使用了位移量就不该使用变体字段。

总之，除变体具有长格式的NEW，DISPOSE，SIZE OF 之外，具有明显位移量的字段重叠是完全可以控制的。

六、Sets(集合类型)

集合类型是对基本类型中的有序类型的一种多个成员的汇集。当集合中没有成员存在时，称为空集合，记为[ ]。由此可见，一个集合可由在方括号内的成员来表示。下面举一例来说明集合类型的定义：

```
TYPE
  COLOR = (RED, YELLOW, GREEN, BLUE)
  PALETTE = SET OF COLOR
```

```
VAR
  LETTER：SET OF 'A'..'Z';
  TINT：PALETTE;
```

集合的操作中，基本类型ORD的值在0到255范围以内。允许SET OF CHAR。但是不能在没有对年代定义的情况下，直接写出SET OF 1942..1984。

七、FILE（文件类型）

文件类型是由全为同一种类型的分量序列构成的。分量的数为文件长度，不靠文件类型说明来确定的。无分量的文件称为空文件。

文件说明的格式如下：

```
TYPE F1 = FILE OF COLOR;
     F2 = FILE OF INTEGER;
```

在PASCAL中，文件是一种结构数据类型，有点象数组，但文件没有界限的，并且一次只能访问一个分量。

在IBM PASCAL中，文件与操作系统的磁盘文件相对应，或者与显示器或打印机设备相对应。为此，FILE OF FILE 是无效的。但是 ARRAY OF FILE 或 FILE OF ARRAY 是允许的。

不能用变体记录构成超数组类型表示文件，否则会给出警告信息。文件变量不可赋值，比较或传递；只能作为参数进行说明与传递。

涉及文件还有两个问题需要说明：

1.文件控制块

文件变量是按着文件控制块（FCB）实现的。记录的字段可以用一般的记录号来访问。如，已知文件FILEVAR，文件方式为FILEVAR•MODE，出错捕获标志是FILEVAR•TRAP，出错状态为FILEVAR•ERRS

可以直接使用记录类型FCBFQQ；任一FILE类型可以传递给FCBFQQ类型的形式引用参数，反之亦然，并允许任何类型的文件进行过程和函数操作。

每一文件F都有与其他变量一样来引用的相关联的缓冲变量F↑。在某种情况下，存取文件的缓冲变量的值还会导致文件的物理输入。

文件缓冲变量可以作为引用参量来传递或用作WITH 语句的记录；然而，在过程，函数，或使用缓冲变量的WITH语句中，文件位置有所变化时，编译程序要发出警告信息。

IBM PASCAL能够支持位于栈中局部变量的文件和位于堆中作为指针引用的文件。

2.INPUT和OUTPUT（输入和输出）

INPUT与OUTPUT文件是作为TEXT文件预先定义过的。并且是自动打开的，和用户的键盘和显示屏幕相连的。还可看成其他文件。

文件有三种方式：

(1) SEQUENTIAL 顺序方式

(2) TERMINAL 终端方式

(3) DIRECT 直接方式（随机方式）

方式是预先说明的枚举类型FILEMODES的一个值。不加说明全都是顺序方式，INPUT和OUTPUT是终端方式。

在ISO PASCAL 中有标准的正文文件TEXT。又称文本文件，或行文文件。其分量为

CHAR类型，即由字符组成的。定义格式为：

```
TYPE TEXT = FILE OF CHAR;
```

字符顺序是构成行的，并有“行标记符”。在IBM PASCAL中，DOS文件没有引入别的格式，因而允许其他系统软件来生成和使用此文件。

3-6-3 引用类型 对变量或常量的引用就提供了间接访问它的方法。PASCAL提供了指针类型，用以建立、使用、删除称之为“堆”区域所指定的变量。主要用于作表格处理，树，图形等。

IBM PASCAL还提供了两种面向机器的地址类型，一种作单十六位（二进制位）地址，另一种作双十六位（二进制位）地址。主要提供给硬件和操作系统接口上用的。

一、指针类型

指针类型是“无界限”的数值集合。用以指向称之为“引用类型”的已知类型的变量。用NEW过程将变量完全动态地定位于称为“堆”的区域中；可是在一般的PASCAL中变量定位在栈中或固定单元中的。

指针操作只有赋值。用=和<>来测试相等和不相等，作为数值或引用参数进行传递。

指针值NIL属于每一种指针类型，指针常用于建立记录，记录的表结构。例如：

```
TYPE
  TREETIP = ^TREE;
  TREE = RECORD
           VALYU : INTEGER;
           LBRANCH, RBRANCH : ^ TREE
         END;
```

在同一个TYPE段中说明类型或类型分量，如TREE和TREETIP为向前指针说明，并允许递归和互相递归结构。

假如上例是在某一过程中，而过程又嵌套另一个也说明为TREE类型的过程中，那么引用类型TREETIP可以在外面定义或在同一TYPE段中接着定义。

指针可以象引用类型的超数组类型。实际的上界传递给NEW过程，以建立适当大小的堆变量。

两个不同类型说明的指针不能赋值或比较，即使它们指向同一个类型也不行。

在上例中，LBRANCH和RBRANCH的类型是TREETIP而不是^TREE，不能把TREETIP的变量赋给LBRANCH字段。

二、ADR和ADS（地址类型）

IBM PASCAL为适应系统工具语言的需要，扩充了关键字ADR和ADS地址类型。ADR和ADS是地址类型子句的前缀，又是前缀运算符。ADR是对应相对地址类型，ADS是对应分段地址类型。

地址类型用于建立、操作和引用实际机器地址的。

相对机器地址是一个16位二进制值，8088分段内存环境中的数据段的位移量。

分段机器地址是32位二进制值，由16位的相对位(bit)移值和16位(bit)8088段寄存器值构成。

ADR OF某类型的类型变量，可以两个分量即类型WORD的R(相对地址)和S(段地址)，一道使用。

类型ADS OF的类型变量，可以同一个分量一道使用，通常用R或S符号表示。以下面为例：

```
VAR
  P: ADS OF FOO; (*P is segmented address of type Foo*)
  Q: ADR OF FOO; (*Q is relative address of type Foo*)
  X: FOO; (*X is some variable of type Foo*)
BEGIN
  P:=ADS X; (*assign the address of X to P*)
  X:=P^; (*assign value whose address is in P to X*)
  P:=ADS P^; (*assign address of value whose address is in P to P; P
      is unchanged by this*)
  Q:=ADR X; (*assign the ralative address of X to Q*)
  P:=ADS Q^; (*assign address ot variable at Q to P*)
  Q:=ADR P^; (*ILLEGAL; cannot apply ADR to<ADS>^*)
  P.R:=16#8000; (*assign 32768 to P's offset field*)
  P.S:=16; (*assign 16 to P's segment field*)
  Q.R:=P.R+4 (*assign P's offset plus 4 to be value of Q*)
END;
```

供选择分量用的^符号是放在ADR或ADS一目运算符之前。如果将^放到任一地址变量之后，就会产生新的变量，所以它可用在赋值目标中，引用参数中和表达式中。

如果两种地址类型都是ADR或都是ADS时，就可把他们看成同一类型。例如，ADR OF WORD赋值ADR OF STRING(200)，由于把STRING(200)类型变量赋给WORD变量的起始地址从200个字节开始，所以可以方便地擦掉部分内存。

如果P_1是地址ADR OF STRING(200)，P_2是任一ADR OF类型，可进行赋值$P_1^{\wedge}:=P_2^{\wedge}$；这是系统软件必须有的一种能力。

关键字VARS可以传递变量的分段地址。在8088环境中，VAR参数是作为变量的默认数据段的位移量进行传递的，而VARS参数是作为段值和位移量进行传递的。DS和SS寄存器存放着这个默认段地址。

VARS参数和ADS变量都有一个用字表示的低地址的位移值，即一个用地址加2表示字段。

值得注意的是地址类型和指针类型是完全不相同的。指针类型只是从一个变量到另一个变量的未定义的映象(转换)。然而，地址类型是作为物理地址实现的。

三、引用类型的限制

在使用引用类型时，有如下限制：

(1) NEW和DISPOSE允许带指针；

(2) NIL不能带地址类型；

(3) 对于空地址，未初始化地址，或无效地址来说没有专用的地址值；

(4) 因为超数组类型只能用NEW过程动态地生成，并且，指向超数组的指针包含有上界，这与含有在地址中的段值相矛盾，所以不支持引用类型的“超数组类型的地址”，但是，允许用ADR和ADS来获得超数组类型变量。然而不象指针那样，地址不包含任何上界。下

面有两个预先说明的地址类型：

ADRMEM = ADR OF ARRAY〔0..32765〕OF BYTE;

ADSMEM = ADS OF ARRAY〔0..32765〕OF BYTE;

上面地址类型是字节类型数组的地址，故可用数组符号来获得字节位移量。例如：

```
VAR BX:ADRMEM; FOO:SOMETYPE;
BEGIN
    BX:=ADR FOO;
    BX^〔0〕:=BX^〔2〕
END;
```

3-6-4 过程类型

一、过程类型含义

过程类型与PASCAL的其他数据类型不同，不能在TYPE段中指定过程类型标识符，或定义过程类型的变量。然而，在过程或函数的首部能够定义过程参数类型和函数的结果类型。在IBM PASCAL中没有过程变量。

PROCEDURE ZEROPOINT (FUNCTION FUN (X:REAL):REAL);

上面的x有无是无关紧要的；作为过程类型参数的标识符的类型才是重要的。

二、类型的兼容性

IBM PASCAL利用标准ISO PASCAL类型的兼容性，在LSTRING类型，超数组类型和常量约定等方面增加了一些新规则，就可使用象ORD和RETYPE类型传递函数变量。

两种类型是等效的，相容的，或不相容的。表达式可以相同或不同是一个变量、数值参数，或数组下标是“赋值相容的”。

三、类型的等同和引用参数

两种类型是等同的话，其含义为两种类型具有相同的标识符，或标识符用同一类型如TYPE $T_1 = T_2$的类型定义说明是等效的。

对于引用参数还加有两个特别例外的情况：

(1) LSTRING 或 LSTRING(n)类型的实在参数，可以传递给超数组类型STRING 的形式参数。

(2) 任何FILE类型或FCBFQQ类型的实在参数，可以传递给任一FILE 类型或等同类型FCBFQQ的形式参数。

除了已注明是字符串外，两个记录或数组类型必须是等同才能赋值。

只有STRING(n)是PACKED ARRAY〔1…n〕OF CHAR的缩写符号，这两种所描述的类型是等同的。

然而，LSTRING(n) 不是PACKED ARRAY〔0…n〕OF CHAR 的缩写符号时，由于，在LSTRING类型的变量在赋值、比较、READ和WRITE情况下，进行特殊处理，为此二者不等同或不兼容或不赋值兼容。

四、类型兼容和表达式

两种简单类型或两种引用类型必须具备如下几种情况之一才是兼容的：

(1) 是等同的；

(2) 一个是另一个的子界；

(3) 都是兼容类型的子界；

(4) 都是ADR类型；

(5) 都是ADS类型。

两种结构类型若具备如下情况之一，才是兼容的：

(1) 两种结构类型既不是文件（或包含有文件）类型FILE，也不是超数组类型，并且是等同的；

(2) 都具备相兼容的简单类型的集合SET类型和都是PACKED的或都不是 PACKED的；

(3) 都是字符串STRING类型导出的类型，具有相等的上界；

(4) 都是LSTRING可变长字符类型导出的类型

两个值必须是类型相兼容，才能在表达式中用运算符组合。

一个CASE标号表达式类型必须与所有CASE常量值类型相兼容。

五、赋值兼容和赋值

如果T和TE是简单类型，那么：

```
VAR   X:T;
      XE:TE;
      X:=XE;
```

只要XE同X是赋值兼容就有：

(1) T和TE是等同类型；

(2) T和TE是相兼容的，并XE在T的范围内的；

(3) T是实数且TE与INTEGER相兼容的。

如果T和TE是结构类型，那么：

```
VAR   X:T;
      XE:TE;
      X:=XE;
```

只要具备XE同X是赋值兼容就有：

T和TE是相兼容的

(1) 对于集合，XE的每个成员是在T的简单类型之内。

(2) 假定T是LSTRING(m)，而TE是LSTRING(n)，并具有m>=TE•LEN。

集合和LSTRING的赋值要取决于赋值的表达式的值。

3-7 变量说明与使用

3-7-1 变量说明 变量说明是标识符表组成的，在表的后面接着是它们的类型。它们是放在程序，过程，模块，函数，接口中；或实现的VAR段中； 或函数， 过程类型的形参数表中。放在VAR段中的那些变量说明可以有任何类型，放在形参表中的那些变量说明只能有一个类型标识符。例如：

```
TYPE VECTOR=SUPER ARRAY [1..*] OF REAL;
     S8=STRING(8);
VAR XT, YT: REAL;
     PAINT: ARRAY [1..10] OF COLCR;
     VECTXX:VECTOR(10);
PROCEDURE NAME IT(VAR N:S8);
```

3-7-2 变量属性 属性是向编译程序给出关于变量或例行程序的专用信息。变量说明或过程或函数首部可具有一个或多个"属性"。

属性是在VAR段中或过程、函数首部之后给出，但不能在TYPE段或参数表中（除非是过程类型）出现。

一、STATIC(静态属性)

STATIC属性表明变量在内存的位置是固定的。变量具有STATIC属性就是变量有一个唯一的绝对地址（或数据段位移量地址)。其作用是节省内存空间和存取时间。

例如：

```
VAR BASEVECTOR〔STATIC〕:
      ARRAY〔0 .. MAXVECT〕OF INTEGER;
       :
VAR〔STATIC〕I, J, K:0 .. MAXVECT;
```

STATIC属性不能用于过程和函数。

二、PUBLIC和EXTERN

PUBLIC和EXTERN属性表示存取变量，而变量是被其他模块装入，或是驻留在其他模块中。EXTERN与变量联用时是属性，与过程或函数联用时是命令。属性是在放在括号里，并用逗点分开。命令要取代过程或函数块，所以要用分号分开。EXTERN过程函数不跟着程序块。

PUBLIC和EXTERN所指定的变量绝对是STATIC属性，其标识符是在生成代码目标文件时，传给链接程序的。

例：

```
VAR〔EXTERN〕GLOBIT, GLOBLIT:INTEGER;
VAR BASEPAGE〔PUBLIC〕:BYTE;
```

具有PUBLIC属性或EXTERN命令的任何过程或函数，必须直接地嵌套在程序或实现中。

例：

```
FUNCTION HPOWER (X, Y:REAL):REAL〔PUBLIC〕;
PROCEDURE ACCESS (KEY:KTYP); EXTERN;
```

三、READONLY

READONLY属性防止对变量进行赋值，同时也能防止作为VAR(或VARS)参数来传递变量。例如：

```
VAR INCAME〔READONLY〕: BYTE;
VAR〔READONLY〕I, J〔PUBLIC〕;
K〔EXTERN〕: INTEGER;
```

四、PURE

PURE属性是向优化程序表明，全局变量是不被过程或函数修改的。PURE只能用于函数，不能用于过程或变量。例如：

```
A:=VEC〔I*10+7〕;
B:=FOO;
C:=VEC〔I*10+9〕;
```

如果FOO为PURE，优化程序只生成一次计算I*10的代码。但FOO不是PURE，它可修改

I，每调用FOO之后I•10要重新计算。

3-7-3 组合属性的规则 在说明变量时，依据下面的规则来组合属性：

(1) STATIC和READONLY总是可用的；

(2) 不能同时使用EXTERN和PUBLIC；

在过程和函数说明中依据下面规则来组合属性和EXTERN命令：

(1) 任意函数可以给出PURE属性；

(2) 过程和函数必须直接地嵌套在一个使用程序、模块、或实现中，以便使用任意属性(PURE除外) 或EXTERN命令；

(3) 不能同时，只能单独使用EXTERN或PUBLIC。

3-7-4 VALUE段(值段) 只有静态地定位的变量，即：在程序，模块，或在实现上说明的变量，或者具有STATIC或PUBLIC属性的变量才可以在程序，模块，实现，过程或函数的VALUE段中给出初始值。

因为具有EXTERN属性的变量不是由编译程序定位，所以不能放在VALUE段中。

VALUE段存放着给整变量或分变量的常量（包括常量表达式和结构常量） 的赋值。在记录中只有一个变量必须赋初值。例如：

```
TYPE
VECTOR = SUPER ARRAY [1 .. *] OF REAL;
   VEC10 = VECTOR(10);
   YEAR = (FROSH, SOPH, JNR, SNP);
   PTR = ^IDREC;
   IDREC = RECORD
                NAME : LSTRING(10);
                CLASS : YEAR;
                NEXT : PTR
                END;
VAR
   VIVEC : ARRAY [1 .. 10] OF VEC10;
   IDRECORD : IDREC;
   I:INTEGER;
VALUE
   I := 10;
   IDRECORD := IDREC('ZEVS', JNR, NIL);
   VIVEC[4] := VEC 10 (DO 10 OF 0);
```

3-7-5 值 变量标识符既是指定整变量，分量变量，也可是指定指针引用的变量。在IBM PASCAL中，引进“值”的概念；值可以是：

(1) 一个变量；

(2) 一个常量；

(3) 一个函数命名符；

(4) 一个值分量；

(5) 一个由指针或地址值引用的变量。

在扩展函数特性中允许函数返回数组，记录或集合结构，在这些返回结构中的分量可以用变量所用的相同语法表示。

在扩展函数特性中还允许再引用由函数返回的引用类型，然而，函数命名符不能用作变量，只能用作数值；由指针引用的变量或由函数返回的地址变量不能直接地进行赋值。

结构常量特性允许用户说明结构类型常量，而结构常量标识符的分量可以用同样的语法表示。例子：

```
TYPE REAL3 = ARRAY [1..3] OF REAL;
CONST PIES = REAL3(3.14, 6.28, 9.42);
      •
      •
      •
X:=PIES[1] *PIES[3]; (* X:=3.14*9.42*)
Y:=REAL3(1.1, 2.2, 3.3)[2]; (* Y:=2.2 *)
```

下面我们分别叙述各种变量与值

1．转换函数变量和值

用标识符表示整变量，其值用文字常量或常量或变量标识符．函数命名符或结构常量表示的。

2．分量变量和值

分量变量或值的分量用定义分量的选择项符来表示。选择符的形式取决于数组、记录、文件或引用类型。例如：

```
VICTOR [20, I]
VICTOR [20, I] • COMPONE∧
VICTOR [20, I] • COMPONE∧ • INDY500∧
       [12] ['Q', RED] • PHONE
```

3．下标变量和值

数组的分量是数组变量或后接一个下标表达式的值来表示的。下标表达式的赋值必须与数组类型定义中给出的下标类型相兼容。

```
ARRICHR['C']
BETAMAX [12] [-3]
BETAMAX[12, -3] (*Same as above*)
ARYFUNV(A, B) [3, 7]; (*Selection made on a function returning an array*)
```

4．字段变量和值

记录的分量是用记录变量或值后接分量的字段标识符来表示的。在WITH 语句中，记录变量或值只能给出一次，在此语句中，可以直接使用它的字段标识符。例如：

```
PERSON • NAME
PEOPLE • DRIVERS • NAME
WITH PEOPLE • DRIVERS DO…
     NAME…(*Same as above*)
RECFUNX('Alpha')•BETA
  (*Selection(•BETA)on a function returning record 'Alpha' is being pass
```

as a parameter*)

记录字段的表示法适用于FCBFQQ字段文件，数字表达式的地址类型值，及表示当前长度的LSTRING。

3-7-6 引用变量 假如P是指针变量或指向类型T的类型值，或者，在地址类型的情况下，是ADR OF T 或 ADS OF T变量或值，则P就是表示此变量或值，P∧是被P引用类型T的变量。例如：

```
CURRENT PERSON∧
NEXT∧〔J〕∧
```

常数结构分量不能随意加上∧就可以引用，因为NIL是唯一的常量引用值，所以NIL∧是错误的。

对于指针或地址类型返回的函数的命名符，还可以加上一个∧表示返回值是引用变量。但变量不能赋给引用参数或作为引用参数传递。例如：

```
FUNK(I, J)∧
FUNK(K, L)∧•FOO〔2〕
```

对于地址类型如有：

```
VAR
  P:ADR OF 某类型，或有：
  P:ADS OF 某类型
```

那么P•R则表示类型WORD的地址值，或是WORD类型地址的位移部分。

3-8 表达式

3-8-1 表达式与运算符 表达式由运算符和操作数组合构成。操作数是值：常量，变量，或函数命名符。

标准的与扩展的运算符如右表：

名　　称	标　准　的	扩展的	优先级
非	NOT		1
乘，除…与	*, /, DIV, MOD, AND		2
加，减，或	+, −, OR	XOR	3
关　　系	=, <>, <=, >=, <, >, IN		4

优先级表明在表达式中运算的先后次序。NOT(非)运算符为最高级，是一目运算符。随后是"乘"而后是"加"，最低级是关系运算符。表达式如使用括号，可提高优选级，先运算括号内而后其他。

一、+，−，*

加、减、乘运算符是对整型(INTEGER)和实型 (REAL) 或二者混合及对字类型(WORD)进行操作的。其中+，−，是具有一目和两目性质的运算符。但减号的一目算符"−"不能在字类型前缀出现，否则会发出警告。如果两个操作数全为整型或全为实型，其结果也为整型或实型。整型与实型混合操作时，先将整型转换成实型之后再运算其结果为实型。操作数之间不能连续出现两个以上操作运算符。

二、/

/为除法算符。操作数可以是INTEGER或REAL类型，但不能是WORD类型，其结果总是为实型 REAL。

三、DIV，MOD

整数除(DIV)求商和取模(MOD)即求余数两种运算。参加的两个操作数都必须是INTEGER或都是WORD但不能是REAL。例如：运算结果为整型数据。

7 DIV 2=3

11 MOD 5=1

120 MOD 35=15

DIV是两整型数据相除，取其整数部分。MOD是两整型数据相除，取其余数。

四、AND，OR，XOR，NOT和关系运算符

AND，OR，XOR，NOT为布尔运算符，具有“逐位”的逻辑功能，其中只有NOT是单目运算符，其余为双目的。除了布尔运算符外，还有关系运算符。这些关系运算符有：

= 等于

<> 不等于

<= 小于等于

>= 大于等于

< 小于

> 大于

参加布尔运算的操作数必须都是INTEGER，或都是WORD(不能是REAL)；然而，关系运算允许整型、实型和字类型，其运算结果类型与操作数类型相同。其值为真(RTUE)和假(FALSE)，并有FALSE<TRUE。

五、IN

IN是检验集合成员资格的操作符。

3-8-2 简单表达式的几种形式

一、算术表达式

算术表达式就是与通常算术运算相似的一种表达式。例如：

3*X+17

二、布尔表达式

在标准的ISO PASCAL 的布尔(Boolean)运算符是NOT，AND和OR，IBM PASCAL又扩展了一个布尔运算符XOR。关系运算符=，<>，<=，>=，<和>也可与布尔运算符一起使用。例如：

P<=Q

表示P隐含Q的布尔运算。又如：

```
WHILE(I<=MAX)AND (V[I]<>T) DO
    I:=I+1;
```

上例中必须在I<=MAX和V[I] <> T两个条件成立的情况下，才执行I:=I+1语句。

关系运算符允许的操作数必须是类型兼容。

引用类型只能用=和<>进行比较。要使某种关系与地址类型一道使用，则必须采用·R或·S表示法。

文件，数组和记录，除了字符串类型STRING和LSTRING 外，不能作为整体进行比较。两个STRING类型比较必须有相同的上界；两个LSTRING可以具有不同的上界。超过当前长度的字符不予理睬，如一个 LSTRING 的当前长度大于另一个，则将短的一个看作是用 CHR(0)分量进行扩展的。

六种关系运算符=，<>，<=，>=，<和>对于数值，枚举或字符串操作数，具有相同含意。

三、集合表达式

有些运算符允许在集合运算下使用，其含意如下：

+　　为集合并

-　　为集合差

*　　为集合交

=和<>　检测集合相等性

<=和>=　检测集合的包含性

<和>　　检测本征集合的包含性

IN　　检测集合成员资格

其中的<和>是IBM PASCAL扩展运算符。在ISO PASCAL中，集合不能使用。

在使用IN运算符时，左边操作数与右边操作数的集合的简单类型要相同。如X为B集合成员，则X IN B表达式为真，否则为假。

例：如果X=1，B=SET OF 2..9，那么X IN B总为假(1是2..9类型相同，但赋值的1不在2..9之内)。

集合表达式可以用集合构造符[ ]中的集合元素。如集合的赋值用 Z:=['A'..'H'];

运用=，<>，>=，<=，>，<和IN七种关系运算符，进行集合类型关系运算的格式如下：

集合1　关系运算符　集合2

例如：

```
[A, B, C, D]=[B, D, C, A]   为TRUE;
[A, B, C]>=[A, B]           为TRUE;
A IN[B, C, D]               为FALSE;
```

3-8-3　其他表达式特性

一、EVAL过程

EVAL过程对有副作用的函数才有用，只计算参量值，而不调用它。

例如：

```
EVAL(NEXTLABEL(TRUE))
EVAL(SIDEFUNC(X, Y), INDEX(4), COUNT)
```

二、RESULT函数

RESULT函数用以访问函数的当前值，这比对一个函数值采用分离局部变量，并在返回前把这个局部变量赋给函数标识符更为有效。例如：

```
FUNCTION FACTORIAL (I:INTEGER) : INTEGER;
BEGIN
  FACTORIAL:=1;
    WHILE  I>1  DO
    BEGIN
    FACTORIAL:=I * RESULT (FACTORIAL);
    I:=I-1
```

```
    END;
END;
FUNCTION ABSVAL(I:INTEGER): INTEGER;
BEGIN
  ABSVAL:=I;
  IF I<0 THEN ABSVAL:=-RESULT(ABSVAL)
END;
```

三 RETYPE

IBM PASCAL为了适应于改变值的类型，采用RETYPE 函数。

使用RETYPE时，必须注意：RETYPE作用于存贮器字节数，而忽略两个倒数字节的数，是先出现还是后出现于内存中。例如：

```
TYPE COLOR=(RED, BLUE, GREEN, ORANGE);
VAR X, Y:INTEGER; C:COLOR;
X:=ORD(ORANGE);(*X will be 3 *)
C:=RETYPE(COLOR, Y); (* C will be ORANGE-acts as inverse of ORD *)
RETYPE(STRING2, I*J+K)[2]; (* effect may vary *)
```

3-9 语句

语句是表示算法的，也就是说是可执行的。本节中将介绍五部分内容：语句标号、基本语句、结构语句、重复语句和顺序控制运算符。

3-9-1 语句标号 语句前面可以冠以标号，其标号可以是一个或多个数字（前面的0是不理睬的），也可以是一般标识符、规则的标识符、常量标识符和表达式。但非十进制符号不能做为标号。所有标号必须在程序首部的LABEL段中加以说明。例如：

```
LABEL 10, 20, INNER, OUTER;
```

这些已被定义标号10，20，INNER，OUTER可以被GO TO语句引用；同时，BREAK或CYCLE语句也可以引用放在循环语句(WHILE，REPEAT，FOR语句)前面的标号。

在本节后面的“CASE语句”中，将对CASE常量进行介绍。现在要注意的是一个语句可以放在CASE常量表和GO TO标号之后，在这种场合下，它们是按下列顺序给出的：首先是CASE常量，尔后是GO TO 标号。例如：

```
321:139<语句>
{321是CASE值，而139是语句标号)
```

3-9-2 基本语句 基本语句是由具有单一算法的语句所构成的。有：赋值语句，过程语句，GOTO语句及空语句。具有控制程序走向特性的语句，还增加了RETURN语句，BREAK语句和 CYCLE语句。

一、赋值语句

用处：用表达式指定的值来代替变量（或函数）的当前值。

格式：V:=e

V为变量或函数，:=为赋值符号，e为表达式。

说明：表达式值的类型应同变量或函数类型是赋值相容的。

赋值语句的实例：

```
A:=B;
SUM:=X*(Y+2);
FUNX:=2*RESULT(FUNX);
```

二、过程语句

用处：用标识符指定要执行的过程。

格式：过程的标识符。

说明：过程是在程序的首部定义的。在程序体中要调用时，就写上过程的标识符后跟分号，就表明是一个过程语句。假如程序首部已利用PROCEDURE定义的 DOIT 为一个过程时，在程序中可有调用的过程序语句出现。例如：

```
  :
DOIT;
READ(INFILE, I, J, K);
  :
END.
```

三、GOTO语句

用处：使程序转向指定标号地方的语句继续处理。

格式：GOTO LABEL

GOTO是转向语句的保留字，LABEL是语句标号。

说明：执行GOTO语句之后，使程序由此转到LABEL指明的标号所在语句，继续执行。例如：

```
  GOTO 10;
    :
10:WRITE('here');
    :
  END.
```

要严格正确地使用GOTO语句，否则会破坏程序的结构，使程序混乱，不易读。GOTO语句基本上不能用于跳到较低层。这就是说不能用 GOTO 语句使程序控制转到 IF， CASE，WHILE REPEAT，FOR，或WITH语句块内，也不允许转入同层或较低层以及另一过程或函数内。但是可以从这些语句跳出或从体内跳到较高层上的过程或函数的一个语句。也允许从IF或CASE语句的一个分支跳到另一分支。

四、空语句

空语句表面看是不存的是一种隐含在END结束语句之前一条不执行语句。

五、BREAK，CYCLE和RETURN语句

这是具有控制程序走向特性的三种语句。

BREAK语句是转到重复语句后的第一个语句上的。

CYCLE语句是转到结束重复语句体的一个隐含空语句上的。由此开始下一个重复循环。

在WHILE或REPEAT循环语句中的CYCLE 在下一次执行前，将进行布尔测试；在 FOR 循环语句中的CYCLE转到变量的下一个值。

RETURN 语句是转到当前过程或函数，程序或实现的最后语句之后隐含的空语句上。

BREAK和CYCLE 语句有两种格式，一种是给出循环标号，另一种是无循环标号。如是

给出循环标号，则循环标号表示该循环要退出或重新启用。如果没有给出标号，则认为是最内层的循环。例如：

```
OUTER : FOR I:=1 TO N1 DO
    INNER : FOR J:=1 TO N2 DO
      IF A[I, J]=TARGET THEN BREAK OUTER;
```

实际上，使用IBM PASCAL的BREAK和CYCLE 语句的程序不需要使用GOTO语句。

循环标号放在 FOR, WHILE, REPEAT 语句之前，又因控制流语句允许使用整数或标识符标号。我们建议用整数作为转向的引用标号，用标识符作为循环标号。例如：

```
SEARCH : WHILE I<=ITOP DO
  IF PILE[I]=TARGET THEN BREAK
    SEARCH ELSE I:=I+1;
FOR I:=1 TO N DO
IF NEXT[I]=NIL THEN BREAK
CLIMB:WHILE NOT ITEMA·LEAF DO
  BEGIN
    IF ITEMA·LEFT <>NIL
      THEN [ITEM:=ITEMA·LEFT;
        CYCLE CLIMB];
    IF ITEM?·RIGHT<>NIL
      THEN[ITEM:=ITEMA·RIGHT; CYCLE CLIMB];
    WRITELN('Very strange node');
    BREAK CLIMB
END;
```

3-9-3 结构语句

结构语句是由具有两种语句以上或具有两种以上算法的语句所构成的。它们分为四种：复合语句，条件语句，重复语句和WITH语句。

一、复合语句

复合语句是由符号BEGIN 和 END 括起来的两个以上语句所组成。是按语句先后排列顺序去执行。语句之间的分号是语句的分隔符。

例如：

```
BEGIN
  TEMP:=A[I];
  A[I]:=A[J];
  A[J]:TEMP
END;
```

允许用[和]来代替BEGIN和 END。必须用于封闭程序、实现、过程、函数体。例如：

```
IF FLAG THEN X:=1 ELSE[X:=-1, Y:=0];
WHILE P·N<> NIL DO
[Q:=P; P:=P·N; DSPOSE(Q)];
FUNCTION R2(R:REAL):REAL;
```

〔R_2:=R*2〕;

二、条件语句

条件选择语句有IF语句和CASE 语句两种。用于选择执行某一分支的单一语句。

1.IF语句

格式：IF e THEN S_1 ELSE S_2

e为布尔表达式，S为语句。

说明：布尔表达式值为TRUE(真)时执行语句S_1，否则执行S_2。例如：

```
IF 1>0 THEN I:=I-1 ELSE I:=I+1;
IF(I<=TOP)AND(ARRI[I]<>TARGET)THEN
I:=I+1;
IF I<=TOP THEN IF ARRI[I]<>TARGET THEN
I:=I+1;
IF C1 THEN IF C2 THEN S1 ELSE S2;
(*an ELSE is paired with the closest previous IF;
here S2 is executed if C1 is true and C2 is false*)
IF C1 THEN [IF C2 THEN S1] ELSE S2;<*now S2  is
executed if C1 is false*)
```

2.CASE语句

用处：CASE 语句是允许几种情况执行一个成分语句，也允许一种情况执行许多语句。

格式：①标准格式：

```
CASE e OF C1,…,Cn：语句;
                ⋮
          W1,…,Wn：语句
END;
```

②还有另一种格式：

```
CASE e OF
  情况标号 ： S1;
  ⋮
  情况标号 ： Sn
END;
```

e为表达式，C为常量表，S为语句。

说明：表达式可为变量即整型、字符型或布尔型变量。常量或标号为整型、字符型或布尔型。变量与常量类型必须相容。例如：

```
CASE OPERATOR OF
  PLUS:X:=X+Y;
  MINUS:X:=X-Y;
  TIMES:X:=X*Y
END;
CASE NEXTCH OF
  'A'..'Z','_':IDENTIFIER
```

```
    '+','-','*','/':OPERATOR;
    OTHERWISE
      WRITE('Unknown cnaracter');
      BREAK SCANNER;
  END;
```

上例中一种情况(OPERATOR) 执行许多语句，这些语句必须由 END 括起来。CASE常量语句也能用在记录变量说明中。

三、重复语句 (WHILE, REPEAT, FOR)

重复语句又称循环语句，指定某些语句要循环重复执行。对于 PASCAL 有三种重复语句：WHILE, REPEAT 和 FOR语句。用于退出或重新启用循环的语句有BREAK和CYCLE(见3-9-2节)。

1.WHILE 语句

用处：WHILE 语句重复执行零次或多次指定的一些语句，直到条件不满足为止。

格式：WHILE e DO S;

或 WHILE e DO〔S_1;…;S_n〕;

其中 WHILE,DO 都是WHILE语句的保留字，e是表达式，用于指定的条件，S是指定要执行的语句。有时要执行的是多语句，可用〔和〕或BEGIN和END 括起来。

说明：是否执行指定语句或语句组,首先测验布尔关系表达式e是否满足为前提条件。如满足e为TRUE时，执行指定语句或语句组，不满足e为FALSE时，退出重复。例如：

```
WHILE P<> NIL DO P:=NEXT(P);
WHILE NOT MICKY DO〔NEXTMOUSE; MICE:=MICE+1〕;
```

2.REPEAT语句

用处：REPEAT 语 句完成重复执行只少一次或多次指定的一系列语句。直到布尔表达式为真时为止。

格式：REPEAT S;…;S_n UNTIL e;

其中REPEAT和 UNTIL 为直到语句的保留字,S_1;…;S_n为指定重复执行语句，e为测试条件。

说明：直到语句(REPEAT语句)与WHILE语句不同之处是：前者先执行指定语句，故有只少一次之说。后进行测试条件。WHILE 语句是先判断条件后执行，所以有可能执行零次之说。例如：

```
REPEAT
      READ(LINEBUFF);
      COUNT:=COUNT+1
UNTIL EOF;
```

3.FOR语句

用处：在给定控制量变化范围内，重复执行语句成分，重复的次数由语句中控制变量的初值和终值决定。

格式：FOR 语句两种格式。

其一：FOR I:=StV TO EV DO S

其二：FOR I:=StV TOWNTO EV DO S

其中FOR TO TOWNTO和DO是保留字，StV为控制变量初值，EV为终值，S是重复执行的语句组。

说明：格式一要求初值(StV)小于终值(EV)，格式二则相反。在满足此要求条件下，反复执行S语句成分。每重复执行一次S语句成分时控制变量I的值对格式一是递增值，对格式二是递减值，其值一般为1,或有序的相邻值。对于格式一具备初值＞终值，对格式二具备初值＜终值时退出循环。

对于ISO PASCAL控制变量为整数类型，递增值(对TO情形)或递减值(对DOWNTO情形)为整数1。对IBM PASCAL则允许控制变量为任何STATIC变量。

例如：FOR I:=1 TO 10 DO SUM:=SUM+VECTOR[I]

FOR CH:='Z' DOWNTO 'A' DO WRITE<CH>

四、开域语句(WITH)

用处：WITH语句用以打开一个语句的作用域，包括一个记录或一个记录字段，由此，可以直接引用被打开记录的字段。例如：

WITH PERSON DO WRITE(NAME,ADDRESS,PHONE)；与

WRITE(PERSON•NAME,PERSON•ADDRESS,PERSON•PHONE)；

是等价的。

给出记录可以是一个变量，常量标识符，结构常量或函数结果。

记录表可以在WITH之后给出，语句：

WITH记录1，记录2 DO语句；与

WITH记录1 DO WITH记录2 DO语句是相同的。

3-9-4 顺序控制运算符 在使用IF,WHILE及REPEAT语句常用到布尔表达的连续测试，提供顺序控制功能。

顺序控制特性提供两个“运算符”:AND THEN和OR ELSE。其运算法则为：

如果X为假，则X AND THEN Y就为假，并且不再对Y求值。

如果X为真，则X OR ELSE Y 就为真，且不再对Y求值。

如果连续用了两个以上顺序控制运算符时，必须严格地从左到右地进行求值。

这些运算符，只能用于IF, WHILE 或UNTIL子句中的布尔表达式里。例如：

```
IF SYMBOL<>NIL AND THEN SYMBOL↑.VAL<0
  THEN NEXT _ SYMBOL;
WHILE I<=MAX AND THEN VECT[I] <> KEY DO
  I:=I+1;
REPEAT GEN(VAL) UNTIL VAL=0
  OR ELSE (QU DIV VAL)=0;
WHILE W AND THEN X OR ELSE Y AND THEN Z DO
SOMETHING;
```

3-10 过程和函数

3-10-1 过程和函数说明 过程和函数说明是把标识符和程序段联系起来，因此在程序中可以分别用过程语句或者函数命名符来调用。过程与函数说明的格式与程序格式一样，包括首部、说明与体。但首部与程序首部写法不同。例如：

```
PROCEDURE MODEL(I:INTEGER; R:REAL);
LABEL 123;
CONST ATOP=199;
TYPE INDEX=0..ATOP;
VAR ARAY:ARRAY[INDEX] OF REAL; J:INDEX;
  FUNCTION FONE(RX:REAL):REAL;
  BEGIN FONE:=RX*I END;
  PROCEDURE FOUT(RY : REAL);
  BEGIN WRITE('Output is',RY)END
BEGIN(*MODEL*)
  FOR J:=O TO ATOP DO
    IF GLOBALVAR THEN
    FOUT(FONE(R+ARAY[J]))
    ELSE GOTO 123;
  123: WRITELN('Done')
END; (*MODEL*)
```

过程必须由PROCEDURE起头，函数必须由FUNCTION起始，而程序则用PROGRAM作为起点。构成过程、函数、程序结构。其结构由四部构成：1.首部；2.标号、常量、类型和变量说明；3.局部过程和函数；4.体，即BEGIN…END之间的语句。

在IBM PASCAL中，与标准PASCAL中不同，LABEL、CONST、TYPE、VAR、VALUE段以及过程与函数说明可任意顺序的。

3-10-2 过程和函数首部 过程或函数首部为定义过程或函数和形式参数而指定标识符。

为了在定义过程与函数之前允许间接递归（如A调用B，并B调用A），必须在调用前用FORWARD命令给出实际说明。

```
PROCEDURE ALPHA(Q,R:REAL) [PUBLIC]; FORWARD;
PROCEDURE BETA(S,T:REAL);
  BEGIN ALPHA(S,3.14)END;
PROCEDURE ALPHA;
  BEGIN BETA(Q,6.28)END;
```

属性，如例中的[PUBLIC]可赋给过程或函数，并用方括号括起来，之间用逗点分开。

PUBLIC、EXTERN和ORIGIN能把PASCAL例行程序用PASCAL或别的语言的例行程序链接在一起的低级手段。MODULE和UNIT提供PASCAL程序之间相互链接的高级手段。

3-10-3 函数说明 在PASCAL中的函数与过程类似，函数说明用以定义要计算值的那部分程序。函数是借助校对函数命名符进行赋值而起计算值作用的。

函数说明的格式与过程说明格式是一样的，所不同的是函数说明首部还要给出函数返回值的类型。

函数是用表达式而不象过程用语句引用，并将其值返回表达式。

函数返回值可为简单类型，结构类型或参考类型，超数组类型的导出类型，或含有文件结构，即任何可以赋值的类型。例如：

FUNCTION MAXIMUM(I, J:INTEGER):INTEGER〔PURE〕;

可以用RESULT函数，把函数标识符看作是参数，来调用标识符所代表的函数当前值。

3-10-4 数据参数 在过程和函数中，有三种类型参数：

一、数值参数

传递引用一个数值参数时，实在参数是一个表达式，要在调用程序的作用域内进行求值，并将数值赋给形式参数。

实在参数的表达式与形式参数类型必须是赋值兼容的。

允许用数值来传递结构类型。但因必须复制整个结构而效率极低。

文件变量或超数组变量不能作为数值参数来传递，因为不能对它赋值，然而，可以传递由超数组(SUPER ARRAY)导出类型的变量。

二、引用参数

传递引用参数时，由 VAR 关键字产生形式参数，变量必须是实在参数。在过程执行期间，形式参数所表示的是实在变量。

对形式参数作任何操作，实际上就是立刻对实在参数进行操作，通过实在变量的机器地址传递给过程来完成的。其地址就是默认数据段位移量。

如果变量的选择涉及到引用数组，或再引用地址或指针，其操作是在执行过程之前进行。

形式参数是超数组本身的引用参数。实在参数类型必须是超数组本身导出的类型，或者超数组本身（即，别的参数或被引用的指针）。例如：

TYPE REALS = ARRAY〔0…*〕OF REAL;

PROCEDURE SUMRS(UAR X:REALS; CONST X:REALS);

超数组的实际上界和下界作为变量，可以与UPPER和LOWER函数一起使用。

由于，LSTRING类型的实在参数能够传递给超数组类型STRING的引用参数，所以处理字符串的例行程序常常要用STRING引用参数。

保留字CONST定义的常量参数，作为实在参数是一个只读的引用参数。

三、过程参数

当传递引用一个过程或函数时，实在标识符就是过程或函数名，而后面是形式参数表，表中放有形式参数元素，每个元素，是用逗号分开。

PUBLIC或者EXTERN过程或者嵌套在任何层次上的任何局部过程，都可以传递给相同类型的形式参数。

过程参数的实例：

PROCEDURE ALPHA(FUNCTION FUN1(X,Y:REAL)):REAL〔PURE〕;

PROCEDURE PASS(PROCEDURE PARAMETER);

在IBM PASCAL中规定某些预先说明的过程和函数不能作为参数引用。

下列标准过程和函数不能传送：

CHR	PACK
ORD	UNPACK
SUCC	READ
PRED	READLN
NEW	WRITE

DISPOSE　　WRITELN
ABS
SQR
ODD

就特性来说，下列过程与函数是不能传送的：

LOWER	WRD
UPPER	BYWORD
LOBYTE	SIZEOF
HIBYTE	RESULT
ENCODE	EVAL
DECODE	RETYPE

使用形式过程的例子：

```
PROCEDURE ALPHA;
    VAR I:INTEGER;
    PROCEDURE DELTA;
        BEGIN END;
    PROCEDURE BETA(PROCEDURE XPR);
        VAR GLOB:INTEGER;
        PROCEDURE GAMMA;
            BEGIN GLOB:=GLOB+1 END;
        BEGIN(*BETA*)
            GLOB:=0;
            IF I=0
                THEN [I:=1; BETA(GAMMA)]
                ELSE [GLOB:=GLOB+1; XPR]
        END; (*BETA*)
    BEGIN(*ALPHA*)
        I:=0;
        BETA(DELTA)
    END; (*ALPHA*)
BEGIN(*main*)
    ALPHA;
        :
END.(*main*)
```

在上例中，当调用 ALPHA时，通过过程DELTA而去调用BETA。由于I 等于0， 所以放过GAMMA而递归地调用BETA。第二次调用BETA，GLOB由0变成1然后调用XPR，XPR非有GAMMA不可的。因此要执行GAMMA，GLOB = 1(否则放 过 GAMMA GLOB = 0 生 效)。GAMMA返回，第二次BETA调用返回，第一次BETA调用返回，并ALPHA返回。

过程参数通常有三种用途：

1.数值分析；

2.调用库某些例行程序；

3.特殊应用。

在数值分析中，可把函数传送给过程或函数，以求得界限间的积分以及求最大值与最小值等。

在某些程序库中，例行程序需要有出错处理程序或下一个数值函数。然而IBM PASCAL不必用此法。

所谓特殊应用是指一些有趣的算法在分析问题和人工智能等领域要使用的过程参数。

3-10-5 **内部调用约定** 每个过程或函数在栈中都有一个结构，并在栈顶用高地址列出如下数据：

一、frame pointe——程序结构指针(如有时)；

二、参数值或参数地址(如有时)；

三、调用程序的返回地址；

四、保存的调用程序结构指针；

五、局部数据，暂时数据，等等。

在BP寄存器中放结构指针，通常指向结构中的最高地址字，用于存取结构数据。一个过程和函数嵌套在另一个过程或函数中有一个高层结构指针，以便取得封闭的结构中的变量。

当调用一个过程或函数时，将发生如下处理过程：

一、调用程序需要的任意保存寄存器(结构指针除外)，按着源程序相同的顺序进栈的参数，并进行调用；

二、被调用程序进栈，旧的结构指针分配所需的任何别的栈位置，并建立本身新的结构指针，除DS，SS和BP之外，它还可改变任何寄存器内容；

三、返回时，被调用程序恢复调用程序的结构指针，释放整个结构，并返回。

由于被调用的程序必须撤消任何参数，所以一般要使用8088的"RET" n指令，"n"等于高层结构指针所占用的字节数。

函数总要在寄存器中返回它们的值，对于结构类型和指向超数组的指针来说，调用程序要分配结构暂存区供结果使用，作为参数传送地址给函数，并用寄存器接受返回地址。对简单类型和别的参考类型来说，返回是单字节的放AL中，是两字节放AX中，是四字节的存在ES对中。

$RUNTIME和$ENTRY/$LINE元命令控制着例行程序头或末的特定调用。

下面例子说明某些内部调用的约定，以及说明如何阅读和获得PAS2清单文件的信息。在实例程序后面的汇编程序上说明怎样用得到符号表示位移量。

```
1   PROGRAM EXAMPLE;
2     VAR I:INTEGER;
3
4     PROCEDURE PROC1;
5         VAR K:INTEGER;
6         PROCEDURE PROC2(VAR J:INTEGER);
7             PROCEDURE PROC3;
8                 BEGIN
9                   J:=J+1;
```

```
10                  K: = J
11                END;
offset Length       Variable—PROC3
—    2    6        Return Offset, Frame Length
12              BEGIN
13                  PROC3
14              END;
offest Length       variable—PROC2
—    4    8        Return Offset, Frame Length
—    2    2        J
15          BEGIN
16            PROC2(I)
17          END;
Offset Length    Variable—PROCI
—    0    6     Return Offset, Frame Length
—    4    2     K
18     BEGIN
19       I: = 1;
20       PROC1
21     END.
Offest Length    Variable
—    0    4     Return Offest, Frame Length
—    2    2     I
Procedure/Function: PROC3
0000001 PUSH BP
; Push the caller's frame Pointer.
0000002 MOV BP, SP
0000004 ADD BP, 04H
; Make the caller's SP(top of frame)
; the callee's frame pointer and add
; 4 bytes to allow for the upper frame
; pointer and caller's BP.
0000007 SUB  SP,0004H
; Set up the SP(top of frame)for the
; callee, Leave room for local variables
; and temporaries.
L9:
     ⋮
L10:
00001D  MOV  DI, [BP].00H
```

```
; Get the caller's frame pointer
000020  MOV  DI,FEH [DI]
; Get the address of J by adding the
; offest to the frame pointer.
000023  MOV  DX, [DI]
; Get the value of J.
000025  MOV  SI,[BP]+00H
; Get the caller's frame pointer.
000028  MOV  SI, [SI]
; Get the caller's caller's
; frame Pointer.
00002A  MOV  FCH [SI], DX
; Move the value of J to K
; through K's address.
00002D  LEA  SP,[BP]FCH
; Reset SP(top of frame)to
; point to caller's BP.
000030   POP   BP
; Pop the caller's BP.
000031   RET   0002H
; Throw away two bytes
; (the upper frame pointer)
; and return.
Procedure/Function:PROC2
000034  PUSH  BP
; Push the caller's frame pointer.
000035  MOV  BP,SP
000037  ADD  BP,06H
; Make the caller's SP
; (top of frame)the callee's
; frame pointer and add 6
; bytes to allow for the
; upper frame pointer.
; the VAR parameter, and the
; saved caller's frame pointer.
00003A  SUB  SP,0004H
; Set up the callee's SP
; (top of frame)and leave
; room for local
; variables and temporaries.
```

```
L13;
00003E  PUSH  BP
; Push the caller's BP(frame Pointer).
00003F  CALL  PROC3
; Call PROC3 and push the return address.
000042  LEA  SP, [BP] + FAH
000045  POP  BP
; Back up the SP to point to BP.
000046  RET  0004H
; Throw away four bytes
; (caller's frame pointer
; (upper frame pointer,
; and VAR parameter) ) and
; return.
Procedure/Function—PROC1
000049  PUSH  BP
; Push the caller's frame pointer.
00004A  MOV  BP,SP
00004C  ADD  BP,02H
; Make the caller's SP(top of frame)the callee's
; frame pointer and add 2 bytes to allow for the
; push of BP.
00004F  SUB  SP,0006H
; Set up the SP(top of frame)for the callee.
; Leave room for local variables and temporaries.
L16;
000053  PUSH  BP
; Push the caller's frame pointer(PROC1 is now
; the caller) because this is a nested procedure.
000054  MOV  DX,@@I
000057  PUSH  DX
; Push the address of I which is a global static
; variable
000058  CALL  PROC2
; Call PRCO2 and push the return address.
00005B  LEA  SP, [BP] + FEH
; Reset the SP(top of frame)to point to the
; caller's BP.
00005E  POP  BP
00005F  RET
```

```
; Pop the BP for the caller and return(pop
; the return address).
Procedure/Function:Example
    ⋮
L19:
00006F  MOV  1,0001H
L20:
000075  CALL  PROC1
; Call PROC1 and push the return address.
000078  LEA  SP,[BP]-.FCH
00007B  POP  BP
00007C  LRET
    ⋮
```

例2. 一个PASCAL可调用的汇编例行程序

```
; Unsigned Addition
; FUNCTION UADDOK(A, B:WORD; VAR C:WORD):
;                    BOOLEAN;
; set C:=A+B and return TRUE
; if no overflow
  FRAME : VALUE A          ; 10
;         VALUE B          ; 8
;         ADR C            ; 6
;         <RET/BP>         ; 0
UADDOK  PROC FAR
        PUSH BP            ; save frame pntr
        MOV BP,SP          ; to a ddress parms
        MOV AX,[BP+10]     ; get A
        ADD AX,[BP+08]     ; add B
        MOV BP,[BP+06]     ; adr C
        MOV [BP],AX        ; result
        MOV AX, 0          ; assume bad
        JB UADRET          ; yes it was
        INC AX             ; make good
UADRET : POP BP            ; get frame pntr
        RET 6              ; return pop 6
UADDOK  ENDP               ; bytes
```

3-11 可用的过程与函数

3-11-1 预先说明的过程和函数 在PASCAL中，标准的过程和标准函数是“预先说明的”因此可以在程序中直接使用。IBM PASCAL还提供了另外一些预先说明的过程和函数。如

果想把用IBM PASCAL编制的程序移植到其它机器上运行，则这些过程与函数是不能使用的。

IBM PASCAL中提供三大类过程与函数：

一、有些过程和函数是由编译程序转换成别的调用程序或专用的形成码。例如，ORD，RETYPE等。

二、预先说明的一些一般的过程和函数。例如，SIN，RESET等。

三、有些不是预先说明的过程和函数，然而，却是 IBM PASCAL 标准库的一部分，如，TIME，DATE。

3-11-2 动态分配过程

一、PROCEDURE NEW (VAR P:Pointer)

该过程分配一个新变量V并把变量V的指针赋给指针变量P(VAR参数)。如果V是超数组类型就必须使用长格式。如果V是有变体的记录类型，就认为变体要给出最大可能的长度，允许把任何变体赋给P^。

二、PROCEDURE NEW(VAR P:Pointer; T_1, T_2,…,T_n:tags)

用标记字段值T_1, T_2,…,T_n指定的变体来分配变量。

如果标记字段的值均是常量，则要分配的只是堆所需要的空间量，在字的范围内进行四舍五入。任何省略标记字段的值就认为要分配的是最大可能空间。

标记字段不是常量值，则认为要分配的是最大可能空间。

三、PROCEDURE DISPOSE(VAR P:Pointer)

P必须是有效的指针，不能是NiL，非初始化指针，也不能是指向已经解除了的堆项的指针，此过程释放中P指向的变量所使用的内存。使用这种格式可以安全地释放是超数组类型变量或是带有变体的记录。

四、PROCEDURE DISPOSE(VAR P:Pointer, T_1, T_2,…,T_n:tags)

T_1, T_2,…,T_n是NEW过程定义的那些上界值。参照标记字段或数组上界值 T_1, T_2,…, T_n所隐含的规格来检查该变量的规格。可以用此过程释放由P指向的内存。

3-11-3 过程和函数的数据传送

一、FUNCTION TRUNC(X:REAL):INTEGER

这是标准PASCAL提供的舍去小数部分的截尾函数。X是实型，结果是整型。例如

TRUNC(1.4) = 1;

TRUNC(−1.4) = −1;

二、FUNCTION ROUND(X:REAL):INTEGER

X是REAL，结果是INTEGER，为舍入函数。其作用是对小数部分，四舍五入，取整数。例如ROUND(1.6) = 2; ROUND(−1.6) = −2。

三、FUNCTION FLOAT(X:INTEGER):REAL

X是INTEGER，结果是实数类型REAL；为整型变实型函数。

四、FUNCTION ORD(X:ordinal) : INTEGER

X可以是任意有序类型，结果是INTEGER；

X是整型INTEGER，返回X；

X是WORD类型，若X<=MAXINT，返回X，否则返回X−2*(MAXINT+1)；

X是CHAR类型，返回X的ASCII码；如：ORD('L') = 76.

X是枚举类型，返回类型定义的从零算起的X位置序数；

X也可以是指针，返回值是INTEGER。

五、FUNCTION WRD(X:ordinal):WORD

X为有序任意类型，结果是WORD类型；

X为INTEGER类型，当X>=0时，返回X，否则返回X+MAXWORD+1；

X是WORD类型，返回X；

X为CHAR类型，返回X的ASCII码；

X是枚举类型，返回类型定义的从零算起的X位置；

X也可为指针，返回值为WORD。

六、FUNCTION CHR(X:ordinal):CHAR

X可为任意有序类型，结果为CHAR类型，即ASCII码表中ORD(X)字符：如果X>255或X<0时，则出错。例如：CHR(75)=K。

七、FUNCTION SUCC(X:ordinal):ordinal

X是任意有序类型，结果还是有序类型，返回Y，其值：ORD(Y)=ORD(X)+1；例如：SUCC(-8)=-7，SUCC(29)=30，超出范围或溢出则出错。如分别使用$RANGECK和$MATHCK，则能捕获错误。

八、FUNCTION ORD(X:ordinal):BOOLEAN

X可为任意有序类型，若ORD为奇数，则为TRUE，否则为ELSE

九、PROCEDURE PACK (CONST A:unpack—array;

I:index; VAR Z:Packed—array)

用PACK过程完成未紧缩数组的元素传送给已紧缩数组。A是未紧缩ARRAY〔M..N〕OF T，而Z是已紧缩数组PACKED ARRAY〔U..Y〕OF T。其作用相当于FOR J:=U TO V DO Z〔J〕:=A〔J-U+1〕。

十．PROCEDURE UNPACK (CONST Z:Packed—array;

VAR A:unpack—array; I:index)

以九的类似方式把紧缩数组传送给未紧缩数组。相当于：

FOR J:=U TO V DO A〔J-U+1〕:=Z〔J〕

在未紧缩数组中与紧缩数组中的元素数必须保持一致。

3-11-4 算术函数 所有的算术函数类型参数值为REAL实型，或与INTEGER相兼容的参数。ABS和SQR还可以是WORD类型。

对于INTEGER和WORD类型的ABS和SQR函数，若用了$MATHCK，出现错误状态会产生运行时错误。否则会产生不确定的错误结果。

算术函数有：

1. ABS(X)　　X(REAL,INTEGER,WORO,INTLONG)的绝对值。
2. SQR(X)　　X(REAL,INTEGER,WORD)的平方〔X*X〕。
3. SQRT(X)　　平方根；结果是REAL；如果X<0则出错。
4. SIN(X)　　X弧度的正弦，结果是REAL。
5. COS(X)　　X弧度的余弦结果REAL。
6. ARCTAN(X)　　X的反正切，返回弧度，结果是REAL。
7. EXP(X)　　指数〔e的X方〕；结果是REAL。
8. LN(X)　　X的对数〔底为e〕；结果是REAL，如果X<=0则出错。

在其他语言中的数字函数，而在PASCAL里没有的，如：

1. MAX(X, Y) = X + (Y - X)*ORD(X<Y)

2. MIN(X, Y) = X + (Y - X)*ORD(X>Y)

3. SIGN(X) = ORD(X>0) - ORD(X<0)

4. POWER(X, Y) = EXP(Y*LN(X)) {X的Y方，X>0}

上述诸类的函数作起来是十分简单的。在 PASCAL 补救办法是可以利用PURE 属性和$RUNTIME之命令自己写成用户函数。例如：

```
{$RUNTIME+}
FUNCTION POWER(A, B:REAL):REAL [PURE];
BEGIN
  IF A<=0 THEN ABORT('Nonplus real to Power ',24.0)
  POWER:=EXP(B*LN(A))
END;      (其中ABORT项见3—11—5节)
```

运行时，在库程序里，必须用EXTERN命名来指定的另外一些实型函数。有：

RSIRQQ (A:REAL, B:INTEGER) :返回A**B，整数幂。

RSRRQQ (A,B:REAL) :返回A**B，实数幂(A>=0)。

MINRQQ (A,B:REAL) :返回A和B的最小值。

MAXRQQ (A,B:REAL) :返回A和B的最大值。

AT2RQQ(A,B:REAL):返回(A/B)的反正切。

TANRQQ(A:REAL):返回A的正切。

ASNRQQ(A:REAL):返回A的反正弦。

ACSRQQ(A:REAL):返回A的反余弦。

TNHRQQ(A:REAL):返回A的双曲线正切。

SNHRQQ(A:REAL):返回A的双曲线正弦。

CHSRQQ(A:REAL):返回A的双曲线余弦。

LNDRQQ(A:REAL):返回以10为底的A的对数。

ANNRQQ(A:REAL):返回A的整数部分(类型REAL)。

AINRQQ(A:REAL):如同ANNRQQ但,返回A接近0。

DXPRQQ(A)：返回A的十进制指数。例如：如果E是 DXPRQQ(A) 时，则10的 E-1方<=ABS(A)<10的E方。

M10RQQ(A:REAL; I:INTEGER):返回A乘10给I。

MP2RQQ(A:REAL; I:INTEGER):返回A乘2给I。

3-11-5 扩展内部特性 扩展内部特性提供有下面一些过程和函数：

(1) PROCEDURE ABORT(CONST STRING, WORD, WORD)

用处：ABORT过程，能够在内部运行出错时，暂停程序执行,并给出错误信息。ABORT格式中的STRING或LSTRING是出错信息，第一个WORD是出错误信息码，第二个可为任何内容。

(2) FUNCTION LOWER(expression):Value

对于数组、集合、枚举、子界类型的一个参数的函数，对应上述类型分别返回数组的下界，集合允许的最低元素，枚举类型首值或子界的下界。

(3) FUNCTION UPPER(expression):Value

UPPER函数类似于LOWER，但返回的是参数类型的上界值。假如表达式是超数组类型表达式，UPPER函数返回超数组的实际上界。

(4) FUNCTION LOBYTE(integer—word):integer—word

(5) FUNCTION HIBYTE(integer—word):integer—word

(4)与(5)返回参数为低有效字节或高有效字节(值从0到255)，其类型与参数类型一样。

(6) FUNCTION BYWORD(one—byte, one byte):WORD

对于任何有序类型，需要两个参数放在一个字节中，返回WORD。

(7) PROCEDURE EVAL(expression, expression, …expression)

EVAL过程是对表达式求值，仅对参数本身求值，参数可以是任何类型，量不限。也能用来对函数求值。

(8) FUNCTION RESULT(function—identifier):Value

RESULT函数用来存或取某一函数当前值。

(9) FUNCTION SIZEOF(Variable):WORD

(10) FUNCTION SIZEOF(Variable, tag1, tag2,…,tagn):WORD

(9)(10) 返回变量长度，其值用WORD的字节数表示，标记值或数组的上界要按照NEW和DISPOSE函数中的方法进行设置，如果变量是有变体的记录，则要使用第(9)种格式并返回最大可能的长度；若变量是超数组就必须使用第(10)种格式来表示上界。

(11) ENCODE (VAR LSTRING, X:M:N):BOOLEAN

ENCODE是一种布尔函数，其求值规则是：把X转换成外部的ASCII表示形式，并把该字符放入LSTRING中。并返回真值。若LSTRING太小无法保存所产生的字符串，则返回值为假。X必须是INTEGER，WORD，枚举类型的一个子界，BOOLEAN，REAL，或指针类型(地址类型必须用.R或.S作后缀)。还包括参数M，N的使用，完全和WRITE过程一样（请见3-12文件一节)。

(12) DECOND(CONST STRING, VAR X:M:N):BOOLEAN

DECOND是布尔函数，能完成STRING或LSTRING中的字符串转换成为内部表示形式并把它赋给X。这种操作包括M与N参数的使用，完全与READ过程的功能一样。

(13) RETYPE (type-identifier, expression)

RETYPE为类型转义函数；函数可根据标识符转回表达式类型。

(14) PROCEDURE MDVEL(S, D :ADRMEM;L:WORD)

从源的SA开始传送L个字符给目的的DA，从字符串的左边开始延续到右边结束。

(15) PROCEDURE MOVER(S, D:ADRMEM; L:WORD)与MOVEL过程相类似，但是，由字符串右端开始传送。例如：

```
    TYPE S10=STRING(10);
VAR ST:S10;
BEGIN
  ST:='1234567890'
  MOVER(ADR ST[6], ADR ST&1], 5);
  (*result:'6789067890'*)
  MOVEL(ADR ST[1], ADR ST&lrb.3], 6);
```

(*result:'6767676790'*)

END.

(16) PROCEDURE FILLC(D:ADRMEM; L:WORD; C:CHAR)

用C字符串的L个字符填入D，值与MOVEL和MOVER的情况一样，不进行界限检验。还可调用MOVESL，MOVESR，FILLSC相应的复制例行程序分段地址；但必须用ADSMEM而不是ADRMEM参数来说明的。

3-11-6 字符串内部特性 字符串内部特性提供了一组过程和函数；有一些特性是供STRING使用的，而另一些特性供LSTRING使用。

IBM PASCAL用下述方式直接支持STRING和LSTRING。

一、LSTRING赋值

LSTRING的值均可赋给LSTRING变量，只要二者均不是超数组类型 LSTRING，只要目标变量的最大长度大于等于源值的当前长度就可以。若目标的最大长度小于源值的当前长度，那么仅赋给目标长度。如果使用 $RANGECK就产生运行时错误信息。

二、STRING赋值

STRING 的值可以赋给 STRING 变量，只要二者长度相同，并且均不是超数组类型STRING就可以。作为数值参数来传送两种字符串相似于赋值。

三、字符串比较

关系运算符<，<=，>，>=，<>和=等六种都可用于LSTRING和STRING。对于LSTRING类型，用于对字符串长度字节比较，并允许操作数为任何长度，比较操作数时，认为最短的操作数是由CHR(0)分量扩充而来。操作数可以是超数组类型LSTRING的。但对于STRING来说，一定要二者界限相同，且不能使用超数组的STRING。

四、READ LSTRING

设置当前长度为读得的字符数，读到填满 LSTRING 或行末为止。只要读入字符在已知 SET OF CHAR中，就可以用READSET读入。若该行小于STRING，读STRING操作就用空格来填充。

五、T.LEN

LSTRING变量T的当前长度，可用T·LEN及T(0)来存取。T·LEN的类型为 BYTE，除了用以取得当前长度外，还用于对新长度进行赋值，。

LSTRING最大长度以及STRING长度，能用UPPER函数求出，这对求得超数组的引用参数的上界特别有用。

混用LSTRING和STRING不能进行赋值或比较。

3-11-7 LSTRING专用特性

(1) PROCEDURE CONCAT(VAR D:LSTRING; CONST S:STRING)

把S加到D的末端。D的长度按S的长度增加。

(2) PROCEDURE DELETE(VAR D:LSTRING; I, L:INTEGER)

从D[I]处开始，删除D中的L个字符，D的长度按L减少。若长度(D)小于I+L-1，则出错。

(3) PROCEDURE INSERT(CONST S:STRING; VAR D:LSTRING; I:INTEGER)

在D[I]之前开始插入S个字符。D的长度按S的长度增加，如果上界(D)小于(S)上界加上(D)长度或长度(D)小于I，则出错。

(4) PROCEDURE COPYLST(CONST S:STRING; VAR D:LSTRING)

将S复制给D，D长度置成S长度。如果上界D小于上界S，则出错。

3-11-8 STRING或LSTRING特性

(1) FUNCTION POSITN(CONST P:STRING; CONST S:STRING; I:INTEGER):INEGER; 在S(I)处开始检索，返回S中P星花整数位置。若没有找到或I>上界S，则返回0；如果P是空字符串，则返回1，无出错状态。

(2) FUNCTION SCANEQ(L:INTEGER;P:CHAR;CONST S:STRING; I:INTEGER):INTEGER

在S(I)处开始扫描，并返回到过的字符数，找到一个等于星花P的字符，或者跳过L个字符时就停止扫描；如L<0，则反方向进行扫描并返回负数。如果没有找到等于星花P的字符就返回L参数。如果I大于上界(S)就返回0，没有出错状态。

(3) FUNCTION SCANNE(L:INTEGER; P:CHAR; CONST S:STRING; I:INTEGER):INTEGER

类似于SCANEQ，但当没有找到一个等于星花P的字符时就停止扫描。

(4) PROCEDURE COPYSTR(CONST S:STRING; VAR D:STRING)

把S制复给D。如果上界(D)<上界(S)，则出错，把D中的余下字符置成空格。

3-11-9 库过程和库函数 下面一些例行程序不是预先说明的，用户必须使用EXTERN命令来说明的。

(1) FUNCTION UADDOK (A, B:WORD; VAR C:WORD):BOOLEAN

(2) FUNCTION SADDOK (A, B:INTEGER; VAR C:INTEGER):BOOLEAN

(3) FUNCTION UMULOK (A, B:WORD; VAR C:WORD):BOOLEAN

(4) FUNCTION SMULOK (A, B:INTEGER; VAR C:INTEGER):BOOLEAN

这四种函数进行带符号或不带符号的16位算术运算，当溢出时不出现运行时错误。如果没有溢出全部返回真值，否则返回假值。

(5) FUNCTION ALLHQQ (SIZE:WORD):WORD

ALLHQQ是个堆分配例行程序；如果堆满，则返回0，若堆结构出错或是用所要求的长度来分配变量的指针值，则返回1。

(6) PROCEDURE TIME (VAR S:STRING)

(7) PROCEDURE DATE (VAR S:STRING)

这两个过程把当前的时间或日期赋给STRING或LSTRING变量，如“HH:MM:SS”或“DD:MM:YY”，这些都是用DOS进行设置的。

(8) FUNCTION TICS:WORD

返回DOS计时单元的值是从0%秒到99%秒，计时单元更新频率是每秒18.2次，因此，由TICS返回的值约按0.055进行递增。

(9) PROCEDURE BEGXQQ

此过程能全面初始化例行程序，栈复位、堆复位和初始化文件系统。调用BEGOQQ和调用程序体以及对突然出错而重新启动是有用的。并不关闭文件。

(10) PROCEDURE ENDXQQ;

能全面终止例行程序；调用ENDOQQ终止文件系统。关闭打开的文件并返回到目标操作系统上。

(11) FUNCTION DOSXQQ(COMMAND:BYTE; PARM:WORD):BYTE

DOSXQQ函数可以直接调用IBM PC机的DOS。

3-12 文件系统

3-12-1 文件 在PASCAL语言中，“文件”是一种抽象的结构的数据类型。当然，大多数的PASCAL实现是把这种类型变量连接到实际操作系统的数据文件上的。

IBM PASCAL文件是连接到一般种类文件上的以供IBM个人计算机磁盘操作系统使用。

一、文件有两种基本结构

文件结构是BINARY和ASCII

二、文件方式

文件有三种方式：TERMINAL，SEQUENTIAL和DIRECT：

(1) TERMINAL（终端）方式的文件，相当于人机会话的显示器/键盘或打字机。

(2) SEQUENTIAL（顺序）方式的文件，相当于磁盘文件或其他顺序存取设备。

(3) DIRECT（直接）方式的文件，相当于磁盘文件或其他随机存取设备。

TERMINAL和SEQUENTIAL方式的文件，是从打开的文件开头，并且按顺序进行读写(存取)记录直到文件结束为止。

预先说明的文件有INPUT和OUTPUT，在IBM PASCAL中，是与用户键盘显示器相联接的。对于ASCII结构（类型为TEXT）来说，允许行向编辑（退格，前移光标，删除等等）。然而，对于BINARY结构的TERMINAL方式（通常类型为FILE OF CHAR）来说，允许进行屏幕编辑，清单选择和依据键盘输入，人机对话进行程序设计。

3-12-2 文件系统原语(Primitives)

一、文件原语

1.PROCEDURE GET (VAR F)

如果文件参数F 中，有下一个分量，则把当前文件位置前进到下一个分量上，把这个分量值赋给缓冲变量FΛ，并把EOF(F)变成假；若没有下一个分量，则EOF(F)变成真，FΛ的值变成不确定。

如果FΛ是有变体的记录，则读有最大长度的变体。

2.PROCEDURE PUT (VAR F)

在PUT(F)之前，EOF(F)必须是真，在 PUT(F) 之后，EOF(F) 还是为真。如果预先操作REWRITE或PUT(或其他写过程)，在F上是将缓冲变量FΛ的值写到文件中。并将FΛ值变成不确定，否则会出现错。如果 FΛ是有变体的记录，就写最大长度的变体。

3.PROCEDURE RESET(VAR F)

把当前文件的位置复原到开始位置上，并且进行GET(F) 操作，如果文件不是空文件，就把F的第一分量赋给缓冲变量FΛ，EOF(F)变成假。如果文件是空文件，则EOF(F)变成真，FΛ的值是不确定的。

RESET操作将清除错误陷阱标志(置它为假)。

4.PROCEDURE REWRITE(F)

用REWRITE(F)在写文件之前，把当前文件的位置复原到它开始位置(这是必须进行初始化的操作)，可使EOF(F)变真，FΛ的值是不确定的。REWRITE将清除陷阱标志。

5.FUNCTION EOF:BOOLEAN或FUNCTION EOF(UAR F):BOOLEAN

EOF是判断SEQUENTIAL和TEMINAL方式的文件的缓冲变量Λ是否位于文件F末尾。因此，若EOF(F)是真时，要么开始写文件，要么GET操作已达文件末尾。EOF无参数时，相当于EOF(INPUT)。而它一般不为真。除非是按 Ctrl-2键打入由终止字符LHR(26)产生一结束文件状态，或把输入INPUT文件重新赋给另一个文件。

6.PROCEDURE PAGE或PROCEDURE PAGE(VAR F)

在打印文本文件时，跳到新的一页顶端上。无参数的PAGE等价于PAGE(OUTPUT)。

3-12-3 文本文件输入/输出 输入/输出是用类型为TEXT，结构为 ASCII 文本文件进行的。

为了简化对文本文件的处理除了过程GET和PUT之外，另外引进四个标准过程READ，READLN，WRITE和WRITELN。

文本文件是用“行标记符”排成行的。在读一文本文件时，文件位置向前移到行标记符上，则缓冲变量FΛ值变成空白，标准函数EOLN(F)为真

不止一次前移文件位置，要么EOF(F) 为真(如果到达文件末尾)，要么把下一行的第一个字符赋给FΛ，在下行不空的情况下，置EOLN(F)为假，或在下一行为空时，把空白赋给FΛ，并且次置EOLW(F)为真。

供文本文件进行READ，READLN，WRITE和WRITELN用的数据参数，总可以取下列一种形式：

P P:M P:M:N P::N

其中P为READ变量，而表达式是WRITE的变量，由下文描述。

M和N值是类型INTEGER的数值参数，它们决定输入/输出的格式。在扩展I/0特性中，对于READ和WRITE 允许M和N值，并允许不给出M，只给出N(如P::N)。

当省略M或N时，就将MAXINT值当成M或N，对于BINARY结构文件不接受M和N的值。

交互式提示和回答必须在同一行上进行输入来作为响应，为了指示，要用 WRITE，必须用READLN作为响应。

下面再介绍一下文本文件引入的四个过程：

一、过程READ

过程READ具有如下一些特性：

(1) READ(P_1, P_2,…, P_n)与READ(INPUT, P_1, P_2,…,P_n)是等价的。

(2) READ(F, P_1, P_2,…, P_n) 与 BEGIN READ(F,P_1); READ(F,P_2); …; READ(F, P_n)END是等价的。

(3) READ中的M和N值不予以管理。但是，N具有枚举类型除外。

(4) 若P是CHAR类型，则READ(F, P)与BEGIN P:=FΛ; GET(F)END等价。

(5) 如果P是INTEGER (或WORD) 或者是它的子界，则 READ(F, P)就从F中读出一字符序列，按照通常的PASCAL的语法排成数并赋给P。可为多种进制记数法 (16#C007, 8#74, 10#19, 2#101, #Face)作为INTEGER和WORD来接受的。若P是 INTEGER 类型的数，可接受前面的+或-符号，P若是 WORD 数，则可接受高达 MAXWORD 的十进制数(即从32768到65535)。

(6) 如果P是 REAL 类型的，则 READ(F, P) 是从F中读出字符序列，排成适当类型的数，并赋给P。非十进制记法是不接受的。

(7) 如果P是枚举类型的或BOOLEAN 的，则作为 WORD 的子界来读出数，并把某值赋

给P，于是这数便是ORD枚举类型的值。此外，如果P是类型BOOLEAN的，则对“TRUE“ 或“FALSE” 进行读操作，并将TRUE和FALSE赋给P，要读出的数必须在变量标识符的ORD值范围中。

(8) 若P是一个指针类型，则数值读成WORD并赋给P，在实现中先写入指针后读出，产生相同的指针值，必须利用·R或·S符号来把地址类型读成各个WORD。

(9) 如果P是STRING(n)的，则下面n个字符要顺序读入P，不跳过前面行的标记符，空格符，制表符和格式馈给符。

当读出没有空格而压缩在一起时，使用READ(STRING)是方便的。例如，在FORTRAN读一行格式是“(20I4)” 在PASCAL中可以通过反复读STRING(4)，并调用DECODE获得STRING整数值来实现。

(10) 如果P是LSTRING(n)，则把下面的n个字符顺序读入P中。并把LSTRING的长度置于n。不跳过前面的行的标记符，空格符，制表符，格式馈给符。

过程READ能够用来读出一个不是文本文件的文件(即，BINARY结构)。在这种情况下，READ(F, X)与BEGIN X:=FΛ; GET(F)END是等价的。

如上述，可以使用READ(F, P_1, P_2,…,P_n) 的格式，一般地说，在BINARY结构读操作中不能接受M和N值。

二、过程READLN。

READLN(P_1, P_2,…, P_n)与READLN(INPUT, P_1, P_2,…, P_n)是等价的。

READLN(F, P_1, P_2,…,P_n,)与BEGIN READ(F, P_1, P_2,…,P_n); READLN(F) END是等价的。

READLN(F)等价于BEGIN WHILE NOT EOLN (F) DO GET(F); GET(F)END

READLN只能用于文本文件(ASCII方式)，使之跳到下一行开始处。

三、过程WRITE

1.适用于WRITE规则

(1) F表示文本文件(类型TEXT)，P_1, P_2,…, P_n可为类型CHAR,INTEGER, REAL, BOOLEAN或STRING的表达式(带有任选的M和N值)。

(2) 在I/0扩展特性情况下，表达式可为类型WORD，枚举类型，指针类型或LSTRING的表达式。

2.过程WRITE特性

(1) WRITE(P_1, P_2,…, P_n)等价于WRITE(OUTPUT, P_1, P_2,…,P_n)

(2) WRITE(F, P_1, P_2,…,P_n) 与于 BEGIN WRITE(F, P_1); WRITE (F, P_2); … WRITE(F, P_n)END是等价的。

(3) 在WRITE过程中，M值为待写的字符数，称为字段宽度。M可以为负或零，如果M为负数则使用M的绝对值，字符按左对齐存放，右边补空格。如果表达式所表示的值，已超过ABS(M)个字符位置，则额外部分截去右边的值。

在ISO PASCAL中，M必须大于零，并要求表达式小于M个字符，字符以右对齐存放，则在左边填入空格。

如果省略M或M等于MAXINT，则要使用默认值8。如果P是类型REAL, INTEGER, WORD或指针(输出基数)，或枚举(选择数值或标识符的值)的，则只能用N值。

(4) 若P是CHAR类型，省缺M时，则默认M为1。并且WRITE(F, P)与BEGIN FΛ:=P;

PUT(F)END是等价的。

(5) 若P是类型INTEGER (或WORD) 或其子界，则要写入文件的是以十进制表示的。

如果P是正数则首先写入前面的空白，除非M值小于零，以表示要去掉左边的输出。

若P为负数，要在前面写上负号，当然WORD不能是负数，默认为8。如果ABS(M)比表示的数小，则需要使用附加的字符位置。

N可用十六进制，十进制，八进制，或二进制数。略掉N或N置成MAXINT就隐含着是十进制数，一般要写的WORD十进制数的范围是32768到65535。

(6) 如果P是类型REAL的数，则数P以十进制数表示，舍去小数部分，指定十进制整数位数相等的数写入文件

如果省略N或N等于MAXINT，则P以浮点形式写入文件，如果有N，其小数点之后有N位数。如果N为零，则把P写成一位整数，并带有一个小数点部分，P以此形式写入文件。

用作REAL的M省缺时，默认值为14，如果ABS(M)小于REAL值的表示式，则要写附加的字符位置。一些例子是：

```
WRITE(123.456)
'  1.2345600E+02'
WRITE(123.456:20)
'  1.2345600000000E+02'
WRITE(123.456:  :3)
'          123.456'
WRITE(123.456:2:3)
'  123.456'
WRITE(123.456:—20:3)
'123.456
```

(7) 若P是枚举类型，并且缺省N或等于MAXINT，则要写入文件的是ORD(P)，就好像把它看作是WORD似的。

如果P是类型BOOLEAN的数，则要写入文件的是一串'TRUE'或'FALSE'(看P而定)，如同STRING。对于布尔型省略M，其值是6。对于BOOLEAN来说决不写ORD值的，因为它是对枚举类型而言的。

(8) 如果P是指针类型的，则作为WORD (如上所述的) 来写它，在实现时，这样来定义写值的方法，使得以后读它时，会得到同样的指针值。作为WORD来写地址类型要使用.R或.S的表示法。

(9) 如果P是类型STRING(n)的，则要写入文件中的是P的值。省略M值是STRING的长度n。

若ABS(M)小于STRING的长度，则仅写入前ABS(M)个字符；如果M为零，则没有东西写入，STRING右边部分总是被截去，即使M是负数也是这样。

(10) 在扩展I/O特性的情况下，如果P是类型LSTRING(n)的，则要写入文件中的是P的值。省略M的值就是字符串的当前长度P•LEN.

如果ABS(M)小于当前长度，则用空格填入剩余位置(超过LSRING长度的部分无字符)。注意，可以用NULL:M来写入M个空白的字符串。

注意，第5点到第8点也适用于函数ENCODE。

过程WRITE还可以用来写非文本文件的文件（即是BINARY 的文件），在这种情况下：

WRITE(F,X)；等价于BEGIN FΛ:=X; PUT(F) END;

如上所述可以使用WRITE(F, P_1, P_2,…,P_n)的形式，通常在BINARY写操作时，M和N值是不能接受的，可以用于改进型版本的编译程序中。

4.过程WRITELN

WRITELN(P_1, P_2,…,P_n)与WRITELN(OUTPUT, P_1, P_2,…,P_n)等价

WRITELN(F,P_1,P_2,…,P_n) 与 BEGIN WRITE(P_1, P_2,…,P_n); WRITELN(F) END等价

WRITELN(F)加行标记符给文件F。

使用 WRITELN 是为了写好标记符（结束一行）。只适用于文本文件(ASCII方式)。

3-12-4 扩展I/O特性

一、PROCEDURE ASSIGN(VAR F; CONST N:STRING)

本过程把STRING(或LSTRING)中DOS 文件名赋给文件F。ASSIGN 总是截掉任何后面的空白部分并取消以前所设定的任何文件名。

文件名取下面的格式：

〔drive:〕filename〔.ext〕

filename 可以是1～8字符，任选的扩展名 ext 可以是1～3字符， 还可以给出驱动器说明（如A:）。

filename 可以是CHR(0)——暂存文件

对于非缓冲I/O来说，文件名可以是如下一些专用的DOS名：

(1) PRN和LPT1表示行式打印机

(2) CON，表示控制台

(3) NUL 表示空驱动器

(4) COM1或AUX表示R232端口。

对于非缓冲I/O来说，文件名可以是下面两个专用的IBM PASCAL名之一：

(1) USER表示控制台

(2) LINE表示RS232端口

注意，指定CON作为输入是不提供的，因为512个字符在可供 IBM PASCAL 使用之前，DOS要对它们进行缓冲。USER 应用作非DOS缓冲的控制台I/O。

二、PROCEDURE CLOSE(VAR F)

CLOSE 让 DOS 对一个文件进行关闭的操作，以确保能正确地结束文件的存取。

对于在栈或堆上分配文件变量是特别重要的， 这些文件必须在 RETURN 或 DISPOSE 释放文件控制块之前关闭掉。并且是在 RETRNU 或 DISPOSE 释放栈或堆的文件变量时自动关闭掉的。当程序结束时，静态分配（在固定存贮器中）的文件是自动关闭的。

注意，某些运行时错误可能是由于 PASCAL 运行时系统的失控而引起的。在这种情况下，已经写好的文件可能不被关闭，因而这些文件中的信息就可能丢失。

如有必要，任何与写文件相当的 DOS 缓冲区要空出来(然而，PASCAL 的缓冲变量不是 PUT)。

如果要写入的文件具有方式 SEQUENTIAL，则写到文件末为止。对已关闭了的文件或从没打开过的文件（不是RESET 或 不是REWRITE 操作）进行 CLOSE 操作是允许的。

对文件进行错误捕获，如果没有发现错误就清除错误状态。

三、PROCEDURE DISCARD(VAR F)

过程DISCARD（作废）完成关闭并删除一个打开的文件。该过程极像 CLOSE 过程，所不同的是该过程要删除文件。

四、暂时文件

某些时候一道程序需要一个只供中间数据用的“中间结果”文件。对于使用该程序来说，没有必要给出以正常格式命名的中间结果文件名，IBM PASCAL 提供有一个称为暂时文件的特性，为了建立暂时文件，要 ASSIGN 名CHR(0)；例如，ASSIGN(F, CHR(0))。文件系统将为该文件建立唯一的名字。

当 CLOSE是暂时文件时，无论是程序员显式地还是隐式关闭文件，文件撤除分配地址，脱离内存，暂时文件均被删掉。

3-12-5 其他文件过程和文件变量

一、PROCEDURE READSET (VAR F, VAR L:LSTRING, CONST S:SETOFCHAR

只要字符在集合S中，并在L中有空间，READSET 读出字符，并置入L。

如果没有给出文件参数，就认为是 INPUPT 如同 READ 和 WRITE 一样。

读字符时，总要跳过前面的空格符、制表符，格式馈给符和行标记符。由于行标记符决不在 CHAR 类型中的。所以当遇到行标记符时才停止读操作。

本过程和 ENCODE 同时使用，借助于运行时系统进行格式化的 READ 过程，以及用 READFN 来读文件名(见下文)。

这就为简单命令扫描程序读和分析输入行提供方便。

二、PROCEDURE READFN(VAR F, P_1, P_2,…,P_n)

READFN与READLN一样，但有两点例外：(1) 必须有文件参数F(认为是进行INPUT，但会给出警告信息)；(2) 如果P是类型 FILE 的参数，则从F中读出组成有效文件名的字符系列，并用 ASSIGN 同样的方式把这个字符序列赋给P。别的类型参数是用与READ过程一样的方式读的。

READFN 在内部是用来读程序参数的，当读文件名（也许来自用户）和把文件名赋给某些文件时，这种操作是方便的。

三、文件字段值

文件变量实际上是一种称为类型 FCBFQQ 的文件控制块的记录。在这种记录中有几个标准字段可以用来处理文件方式和错误陷阱。其它字段和记录类型 FCBFQQ 本身可以按照下文描述的方法来使用。

一般的记录字段语法是用来存取文件的FCB 字段的，对于文件F来说，字段是F.MODE，F•TRAP和F•ERRS，这些字段随时可检验或随时改变。F•MODE字段存放文件方式，这些值是预先说明的枚举类型 FILEMODES 的常量。MODE字段仅在 RESET 和 REWRITE期间为文件系统所用，因此改变打开文件的 MODE 字段是无效的（但不提倡这些做法）。

默认的文件方式是 SEQUENTIAL，除非是 INPUT 和 OUTPUT(INPUT 和 OUTPUT 的方式是TERMINAL)。

1.F•TRAP:BOOLEAN

F•TRAP:BOOLEAN字段的初值是假；如果置为真，则它就给文件F打开错误陷阱，以便如果输入/输出出错，不结束程序，并可以检验出错码；如果该字段为假并且I/O出错，则结

束程序，关闭文件将置陷阱成假。

注意，RESET 和REWRITE 关闭文件。

2.F•ERRS:0…15

F•ERRS:0 ··15字段存放着文件F的出错码；零意味着无错，1到15的值表示有一错误状态(见下表)。如果企图对文件F进行操作(除(LOSE 和DICARD之外),并且F•ERRS非零，如果 F•TRAP 是真，则不理这种操作 (程序没结束)。如果 F•TRAR 是假， 就立即结束程序。

3.文件F的出错码：

码	错误
0	无错误
1	备用的
2	备用的
3	OPERATION—无效的 GET 操作，如果EOF、RESET 一台打印机，等等。
4	备用的
5	备用的
6	备用的
7	FILE NAME—无效的语法，名字太长，无暂时名字等。
8	DEVICE FULL—磁盘满，目录满，全部通道已分配完。
9	备用的
10	FILE NOT FOUND—没有找到文体本身
11	备用的
12	备用的
13	FILE NOT OPEN——文件被关闭，对没打开的FCB进行I/O。
14	DATA FORMAT——数据格式错。译码错，范围错。
15	LINE TOO LONG——缓冲区溢出，行太长。

四、首部中的文件变量

文件变量可以作程序参数放在 PASCAL 程序的首部中，并且可以在 VAR 段中作为一个文件而加以说明。运行时，运行时系统将从用户的键盘上获得文件名。

在这里，文件名的格式与 ASSIGN函数中的文件名格式一样。

例如：

```
PROGRAM FILES(FILE1, FILE2);
VAR FILE1:FILE OF REAL;
    FILE2:TEXT;
BEGIN
  ⋮
END.
```

当起动FILES 时，该程序将设法从命令行中读得程序参数， 如果程序没有找到它们，则程序将提示要给出它们。因此，与文件NUMBERS•DAT和文件 LINES•DAT 一道来用程序FILES可以看作是下面样式：

```
A>files numbers.dat lines.dat
```

或

A>files numbers. dat

FILE2:lines. dat

或

A>files

FILE1:numbers. dat

FILE2:lines. dat

(记住，用 Enter键结束各行).

3-12-6 系统I/O特性

系统 I/O 特性允许带有任何文件类型或类型 FCBFQQ的形式参数的过程和函数，要用等同 FILE 类型或等同 FCBFQQ 类型的实在参数去调用。

FCBFQQ 是用来实现文件类型的基本记录类型。

一般地说，FCBFQQ 类型 (和所需的任何别的类型)的接口是由 $INCLUDE 文件和子句 "USES FILKQQ" 而引入的 (FILKQQ 是包含有 FCBFQQ 类型的单位名)。

在PAS_1盘的文件 FILKQQ.INC 中包含有FCBFQQ 的说明。

带有 FCBFQQ 引用参数的过程和函数可以用任何文件类型来调用。包含像 CLOSE 和 READ 那样预先说明的过程和函数。可以把 FCBFQQ 类型变量传递给像要求有文本文件这样的过程 READLN和WRITELN。

这就允许，例如，直接调用文件系统接口例行程序和其他文件系统机构。

为此必须对文件系统要有全面了解。附录B(见英文原版)将详细介绍文件系统接口和文件控制块。使用这方面的资料，我们就可以为其他操作系统写出文件系统接口。

3-12-7 DIRECT文件(直接或随机文件) 为了使用 DIRECT 文件，必须在用 RESET 或REWRITE 打开文件之前，把方式字段设置成DIRECT。

如果文件实际上是按顺序读或写的，可用通常的 READ 和 WRITE 过程，而且在文本文件 (ASCII 结构文件)中,一般也可以使用READ和 WRITE。然而由于 PASCAL 文件是定义的，所以，BINARY 结构的文件一般必须进行GET和PUT而不进行READ和WRITE。

DIRECT 方式的文件适用于下面情况：

1. 读和写时总要打开文件；

2. 可以用记录号而随机地存取记录；

3. ASCII 结构文件的行长度设置着记录的长度，是在TEXT之后,放在括号中作为常量给出的。例如：

```
VAR SMALLBUF:TEXT( 2 );
    DEFAULTX:TEXT;
    PANDOMTXT:TEXT(132);
```

一般地说，行长度是为 DIRECT 方式的文件而说明的，但可以用于其他TEXT文件。

对于DIRECT ASCII 文件来说，行长度，不管是显式设置或是隐式设置的，就是读或写时所用的记录长度。

下面描述其有关过程

一、PROCEDURE GET(VAR F)

当使用GET(F)时，在GET(F)之前EOF(F)的值可能是真也可能是假，因为 DIRECT方

式允许在文件末有重复的GET。

二、PROCEDURE PUT(VAR F)

当使用PUT(F)时，在PUT(F)之前EOF(F)的值可能是真也可能是假。

三、PROCEDURE REWRITE(VAR F)

REWRITE 既允许读也允许写文件，并且，当文件不存在时，则建立之；如果文件存在，则旧值保留。

四、PRCEDURE RESET(VAR F)

RESET 既允许读也允许写，但不自动地建立文件，起始的 GET 读出记录号数。

五、PROCEDURE EOF(VAR F)

如果EOF(F)为真，则最后操作就是写或最后的GET已经达到文件末尾（见3-12-2节）。

六、PROCEDURE SEEK(VAR F; N:WOKD)

SEEK修改F中的一个字段，以便下一个GET或PUT可作用于记录号数N上。记录号参数N可以是类型INTEGER或WORD，其值从1开始。F必须具有DIRECT方式。

如果F是ASCII文件，则 SEEK 就把缓慢求值状态置为“空”。

在 RESET 或 REWRITE 之前必须显式地设置DIRECT方式，以便可以在文件中使用SEEK过程。

1.SEEK后面是GET

如果在文件F中有分量N，则当前文件的位置就前移到这个分量上，这分量的值赋给缓冲变量F∧，并且EOFCF)变成假。

2.SEEK后面是READ，BINARY文件

因为READ(F,B)是“B:=F∧；GET(F)”，所以使用 READ 将导致把当前缓冲变量(不管情况如何)赋给B并读出记录号N后的记录。因为这个作用与记录N的值毫不相干，所以应避免使用。

3.SEEK后面是READ/READLN，ASCII 文件

由于缓慢求值的原因，这将按我们所期望的那样进行工作，SEEK(F.N) 把状态置为“空”，使得由 READ 隐含缓冲变量的存取，实际上在 READ 处理任何字符之前把记录N读到内部行缓冲区中。

4.SEEK后面是PUT

把缓冲变量F∧的变量写到记录号N处的文件F中，F∧值变成不确定，EOF(F)置为真，SEEK 后接 PUT 通常要以设置缓冲变量的值为先导。

5.SEEK 后面是WRITE，BINARY文件

因为WRITE(F,B)是“F∧:=B；PUT(F)”，所以 SEEK 调用之后使用 WRITE 将导致把B写到记录N中，(如同我们所期望的那样)。然而，对于 BINARY 文件来说在 SEEK 之后不能进行READ。

6.SEEK后面是WRITE/WRITELN，ASCII文件

一般说来，ASCII DIRECT 文件的I/O将按我们所期望的那样进行工作，把字符写到记录N中，但BINARY DIRECT 文件的I/O应使用GET和PUT而不是使用READ和WRITE。

7.DIRECT 方式中的EOF

一旦修改（或建立）DIRECT 方式中的文件，文件末尾就不准确了。因此，EOF 不能用来检测 DIRECT 方式中真的文件末尾。

此外，在顺序方式中，文件的准确结束（EOF）不再是可检测的。

在ASCII结构的文件中，如果使用DIRECT方式来修改文件，并且使用SEQUENTIAL重新存取文件，则使用SEQUENTIAL方式就有可能不能存取某些数据。

3-13 编译对象

所谓“编译对象”就是由编译程序编译一个程序的源文件。IBM PASCAL 能够编译三种对象，就是程序、模块以及单位。这些编译的每一个对象又可以含有单位的接口。

3-13-1 程序 用PASCAL 语言所编写的程序，除了它的首部和末尾的句号之外，还有一个过程说明结构。在BEGIN和END之间的语句称之为程序体。

例如：

```
PROGRAM ALPHA(OUTPUT, AFILE:PARAMETER);
VAR AFILE:TEXT; PARAMETER:STRING(10);
BEGIN
    REWRITE(AFILE); WRITELN(AFILE, PARAMETER);
END.
```

在编译和链接之后，程序可按以下所说明任一种方式运行，假定程序ALPHA在文件ALPHA·EXE中存在的，则：

```
A>ALPHA
AFILE:DATA. FIL
PARAMETER:ABCDEFGHIJ
```

或者

```
A>ALPHA DATA·FIL
PARAMETER:ABCDEFGHIJ
```

或者

```
A>ALPHA DATA·FIL ABCDEFGHIJ
```

（要记住，在每一行的后面用户必须按INTER键）

ALPHA是程序标识符或称程序名；是用户自己定义的。因此，程序标识符可以在程序中重新说明，并且可以使用通常的作用域规则。

程序的参数是程序与其环境进行通信的手段，在标准 PASCAL中FILE 类型的变量必须是参数。因为，除此之外再没有任何方法可以把变量赋给操作系统的文件。

程序参数与过程参数完全不一样，程序参数不能传递给程序体的，而且必在构成程序的变量说明块部分中说明

ASSIGN 和 READFN 可用来指定文件名，而不必把它们看作程序参数。

如果没有程序参数，且也没使用INPUT和OUTRUT文件，那么可以使用“PROGRAM标识符;”的形式。

两个预先说明文件，INPUT和OUTPUT，都必须作为程序参数来提供，如果使用了它们而又作为程序参数来提供，那么就可以抑制编译程序的警告信息。

每一参数所具有的类型必须是 READFN 可接受的，也就是，必须是简单类型（INTEGER，WORD，枚举，子界，REAL），指针类型，STRING，LSTRING，或者FILE类型。程序参数必须是整体变量；不允许是选择分量（包括·R和·S地址类型）。

当作为程序参数来指定LSTRING时，就会存在有模棱两可的情况。假设存在有下列的程序：

```
PROGRAM PARMS(In);
VAR In:LSTRING(10);
  •
  •
  •
END;
```

如果，在编译和链接之后，用户按如下来启用程序：

A＞Parms

到底是用户要输入长度为0的 LSTRING 还是想提示 LSTRING。这就不清楚了，IBM PASCAL 则认为是后者，如果省去LSTRING，那么总要出现提示。

对于想使用高级命令行处理的应用程序员来说，单位U的过程PPM具有这种功能。使用PPM，在程序语句中不指定任何参数。

3-13-2 模块 使用模块是一种把几个编译对象合并成一个程序的快速方法，模块是没有体的程序。

模块首部要给出模块标识符，它与程序标识符具有同样的作用域，模块没有体和参数，该编译对象以“END.”结尾、下面是模块的一个实例：

```
MODULE BETA〔PUBLIC〕;
VAR X:INTEGER;
    PROCEDURE GAMMA;
    BEGIN
      X:=1
    END;
    FUNCTION DELTA:WORD;
    BEGIN
      DELTA:=123
    END
END.
```

若没给出显式属性，则认为是〔PUBLIC〕的，因此不需要属性时一定要用〔　　〕。

虽然模块没有体，但其它编译对象可以使用模块标识符来调用该模块，因为模块是一个没有参数的过程。

在某些情况下，编译程序生成必须由模块过程执行的模块初始化码，如果模块说明了某一FILE变量或模块USES某个要初始化接口，则必须完成这次工作，在编译模块时，若发现有这两种情况就给出“Intialize Module”的警告信息。所以，若模块说明了任何模块变量(如上面的X变量)，且在模块中使用这些变量，则该模块也必须进行初始化。

在此情况下，程序编译对象中的模块应该作为一个无参数的EXTERN过程来说明，且在使用模块过程之前，该过程只能调用一次。

例如：

```
PROCEDURE MODULNAME; EXTERN;
```

模块变量是不会自动地给出属性的，并且与程序的变量一样，除非按所说明的那样对FILE变量进行初始化。

如果上述的情况不适用，那么可作为EXTERN过程来调用模块过程，而不必进行模块的初始化调用。

3-13-3 单位 一个单位包括常数，类型，超类型，变量，过程以及在单位接口中说明的函数，任一个编译对象或其他接口都可使用单位接口。

在单位中，实现编译对象存放着过程和函数的体，通过把实现与接口相分开的方法，可以在写实现之前写好程序并进行编译，或者与几个实现一道装入（例如，一个是用PASCAL写的，一个是用汇编语言写的等等）。

一个大的 PASCAL程序最好要按一个主程序和许多个单位来组织，然而，只有程序、模块、接口或实现才可以使用单位，不是个别的过程或函数。

使用接口的编译对象首先必须要为接口提供源码，这一般要用 $INCLUDE 元命令引入个别的文件，这种形式要比把接口实际放在使用它的编译对象里要来的容易可靠，因为只有一个主程序接口复制。

在单位子句中需要有一张单位中输出的所有标识符表（在USES子句中是随意的），以避免标识符发生冲突，系统中有初始化单位的专用内部码和控制接口的翻译程序。

一个PASCAL源文件是由0个或多个单位、接口组成的，其中用分号“；”分开，后面跟着程序，模块或实现，接着一个句号，每部分叫做“分部”。

单位是由单位的标识符组成的。紧跟着是放在括号中的标识符表，它们称为单位的“成分”。放在括号中的标识符是由单位提供的，或是程序、接口，以及实现要求的。

单位要以关键字UNIT打头，这是给单位所提供的，而USES是单位所要求的，在源文件中，所有的单位标识符必须是唯一的，括号中的标识符，在提供的与要求的部分中可以不相同。

提供和要求的标识符是根据表的位置互相对应（类似于过程中的形式参数和实在参数）。放在USES子句中的表是随意的，若没有给出，那么就用UNIT子句中的表。在 USES 子句中命名不同的标识符能消除接口与接口间的标识符冲突。复合USES子句“USES A;USES B;USES C;”，可以合并成为“USES A，B，C；”的形式。

例如：

程序文件：

```
(*$INCLUDE:'GRAPHI'*)
PROGRAM PLOTBOX(INPUT, OUTPUT);
USES GRAPHICS(MOVE, PLOT);
BEGIN(*GRAPHICS*)
    MOVE(0, 0);
    PLOT(10,0); PLOT(10, 10);
    PLOT(0, 10); PLOT(0,0)
END.
```

接口文件GRAPHI：

```
INTERFACE;
    UNIT GRAPHICS(BJUMP, WJUMP);
```

```
PROCEDURE BJUMP(X, Y : INTEGER);
PROCEDURE WJUMP(X, Y:INTEGER);
BEGIN
END;
```

实现文件：

```
(*$INCLUDE:'GRAPHI'*)
(*$INCLUDE:'BASEPL'*)
IMPLEMENTATION OF GRAPHICS;
    USES BASEPLOT; (*identifiers below*)
    PROCEDURE BJUMP;
        BEGIN DRAWLINE(BLACK, X, Y)END;
    PROCEDURE WJUMP;
        BEGIN DRAWLINE (WHITE, X, Y)END;
BEGIN
    DRAWLINE(BLACK, 0, 0)
END.
```

接口文件BASEPL：

```
INTERFACE;
    UNIT BASEPLOT(BLACK, WHITE, DRAWLINE);
    TYPE RAINBOW = (BLACK, WHITE, RED, BLUE, GREEN);
    PROCEDURE DRAWLINE(C:RAINBOW; H, V:INTEGER)
END;
```

实现文件：

```
(*$INCLUDE:'BASEPL'*)
IMPLEMENTATION OF BASEPLOT
```

USES子句只能直接跟在程序，模块，接口或实现首部的后面。当遇到一个USES子句时，所有的分标识符（来自UNIT子句的和USES子句本身）都是放在符号表中输入的，这些与变量，过程，以及函数有关的标识符和接口中的标识符是对应的，从而成为装入程序的外部引用。

上面例子中，当编译好程序时，每引用PLOT过程就会生成一个WUMF的外部引用，然而对于DRAWLINE的引用，要使用外部引用的同样标识符。

接口中的常量和类型（包括任一超数组类型）仅仅是放在程序的符号表中的（若需要，就用新的标识符），所以接口的类型与USES子句中的对应类型是等同的。

使用UNIT的方法要比用$INCLUDE优越的多：

(1) 可改变标识符以避免冲突。

(2) 常量、类型和过程，以及函数的参数只是在接口中给出的，就避免了检测不到的失配。

(3) 接口一般是个清除程序。

在程序或实现前面把一个或多个接口组合起来，$INCLUDE是非常有用的。

一、接口部分

接口部分是以关键字 INTERFACE 打头的，放在括号中的任选项号，并有一个分号，接着是关键字UNIT，设备标识符，放在括号中的输出(分)标识符表，后跟一个分号。再接下去是放USES子句中的该接口所要求的别的任何单位，然后是接口表所给出的所有标识符的实际说明，通常要用CONST，TYPE，VAR部分，过程和函数首部，其次序是任意的，允许没有LABEL或VALUE部分。

对变量，过程和函数可以给出属性。但是PUBLIC或ENTERN属性，或 ENTERN命令是自动给出的，因此务必不要使用这些属性以免发生冲突。

一般地说，只有已经说明过的标识符才是分量的，但是允许说明其他的标识符。如果接口需要一个调用来初始化单位，则关键字BEGIN将生成它，接口用保留字 END 和一个分号来结束。

例如：

```
INTERFACE(3);
UNIT KEYFILE (FINDKEY, INSKEY, DELKEY, KEYREC);
    USES KEYPRIM (KFCB, KEYREC);
    PROCEDURE FINDKEY (CONST NAME : LSTRING;
      VAR KEY : KEYREC; VAR REC : LSTRING);
    PROCEDURE INSKEY (CONST REC : LSTRING; VAR KEY : KEYREC);
    PROCEDURE DELKEY (CONST KEY : KEYREC);
    PROCEDURE NEWKEY (CONST KEY : KEYREC);
    BEGIN
    EDN;
```

在这个例子中，KFCB是KEYPRIM单位的一部分，但不是由KEYFILE 单位输出的，这是允许的。NEWKEY是在接口中定义的，但不是由KEYFILE单位输出的，这也是允许的。但这是毫无意义，因为在单位的任何USES子句和单位的IMPLEMENTATION中，NEWKEY是未知的。

二、实现部分

一个接口的实现是用保留字 IMPLEMENTAION OF，单位标识符和一个分号作为开头的。接下去是单位自己需用的USES子句；下面是一般LABEL，CONSTANT，TYPE，VAR，VALUE 部分，作为分量(这些分量是在外块)说明的或是由内部使用的过程和函数，这些顺序是任意的。

VALUE和LABEL部分可以用在实现中，但不可用在接口中。

只有GOTO(若有的话)是来自实现体本身而不是来自过程和函数，才必须有标号。

常量，变量，和类型在接口中说明后，就不能在实现中再说明，但可以说明别的“专用”项目。

构成单位的过程和函数不能有它们的参数表(它是由接口指定的)或任何属性。

PUBLIC的属性是自动地给出的，除非显式给出EXTERN命令。

INTERFACE中的所有过程和函数必须在 IMPLEMENTATION (或者一个IMPLEMENTATION和汇编码等等)可以实现单一接口。

使用EXTERN命令的所有过程和函数必须放在前头；编译程序要对它进行检查。

单位也可以用其他语言来实现，比如说是 FORTRAN 或各种语言混用，若接口不是用

PASCAL语言来实现的，则接口中必须使用适当的调用顺序属性，当然用户必须熟悉调用顺序和参数的内部表示。

有些运行时的单位，一部分是用PASCAL实现的，一部分是汇编语言实现的，如同所提到的那样，任何不能实现全部接口的过程和函数的实现部分，必须在实现部分开头说明那些不能用EXTERN命令来实现的过程和函数。

一个实现象程序一样可以有一个体；当用单位来引用一道程序时，就执行该体，所以可以进行单位所必须的初始化工作。

这包括内部初始化，象文件变量初始化及用户初始化代码一样，如果源文件含有许多单位，则要按在源文件中找到的USES子句顺序进行调用。所谓体，如同程序中的体一样是一个BEGIN…END的语句。

在初始化时，用以编译的实现接口项号将与用以编译的程序接口项号进行比较，这两个项号必须是相同的，这就可以防止企图一道带有过时单位实现的程序。如果没给出项号，就认为项号为零。

例如：

```
IMPLEMENTATION OF KEYFILE;
    USES KEYPRIM (KBLOCK, KEYREC);
    VAR KEYTEMP: KEYREC;
    PROCEDURE FINDKEY;
        BEGIN (*code for FINDKEV*) END;
    PROCEDURE INSKEY;
        BEGIN (*code for INSKEY*) END;
    PROCEDURE DELKEY;
        BEGIN (*code for DELKEY*) END;
    PROCEDURE NEWKEY (CONST KEY: KEYREC);
        BEGIN (*code for NEWKY*) END
    END.
```

实现部分用句号来结束。

参考书目：Pascal Compiler by Microsoft

第四章　COBOL语言

COBOL (Common Business Oriented Language) 是计算机应用中广泛使用的面向一般企业的通用语言，主要用来解决商业和企业管理以及图书资料检索等方面的大量数据处理问题。

1960年美国公布了第一个COBOL版本，称为COBOL-60。1974年美国发表了ANSI COBOL X3.23-1974文本，1978年ISO宣布ANSI COBOL X3.23-1974作为国际标准文本，即ISO COBOL-78。这一标准文本，COBOL语言已扩充到了12个功能模块。如表4-1所列。

最小的COBOL至少要包含核心一级和表处理、顺序存取一级。各模块的一级是二级的一个子集。IBM PC的COBOL含有ANSI-X3.23-1974中的九个模块的一级的功能，在多数情况下还包括许多二级功能；CROMEMCO COBOL包含了除报表编制、排序和通讯之外的8个功能模块的全部一级功能和部分二级的功能。

表4—1

核心	表处理	顺序存取	相关存取	索引存取	报表编辑	排序	分段	库	调试	间通内部程序	通讯
二级	二级	二级	二级	二级	一级	二级	二级	二级	二级	二级	二级
			一级	一级		一级	一级	一级	一级	一级	一级
一级	一级	一级	空	空	空	空	空	空	空	空	空

4-1 COBOL程序

4-1-1 COBOL源程序结构 COBOL是一种高度结构化的语言，用户编写COBOL程序时必须遵照有关程序组织和格式的特殊规则。每一个 COBOL 源程序都必须顺序地由下述四部分(区)组成，且每部分(区)以标题开始。

标识部分 IDENTIFICATION DIVISION.

设备部分 ENVIRONMENT DIVISION.

数据部分 DATA DIVISION.

过程部分 PROCEDURE DIVISION.

各部分标题在程序中独占一行。标识部分用来注明程序的名称和有关程序的说明；设备部分指出本程序所用的计算机设备和它们的特点；数据部分用来定义将要处理的数据名及特性；过程部分是程序的执行部分，它指出对数据的各种处理。

下面列出的是一个完整的COBOL源程序的例子。它说明程序各部分的基本构成和先后次序。这个程序将一个建立在磁盘上的库存文件的每一个记录打印在宽行打印机上，记录中包括某种货物的货号、单价和数量，此外还计算和打印每种货物的库存金额。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. STORE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MAIN-FILE ASSIGN DISC
    SELECT PRINT-FILE ASSIGN SYSOUT
DATA DIVISION.
FILE SECTION.
FD MAIN-FILE LABEL RECORD
    STANDARD.
01 STORAGE.
    02 GOODS-NUMBER PICTURE X(8).
    02 UNIT-PRICE PICTURE
       9(5)V999.
    02 QUANTITY PICTURE 9(6).
```

```
FD  PRINT-FILE  LABEL    RECORD
    OMITTED.
01  ONE-LINE.
    02  GOODS-NUMBER  PICTURE    X(8)B(8).
    02  UNIT-PRICE    PICTURE    $(5)9.999B(8).
    02  QUANTITY      PICTURE    Z(6)B(12).
    02  GOLDSUMMER    PICTURE    $(7)9.999.
PROCEDURE  DIVISION.
BEGIN. OPEN  INPUT  MAIN-FILE  OUTPUT
       PRINT-FILE.
MAIN. READ  MAIN-FILE  AT  END  GO  TO    EOJ.
      MOVE  STORAGE  TO  ONE-LINE.
      MULTIPLY UNIT-PRICE BY QUANTITY GIVING GOLDSUMMER.
      WRITE  ONE-LINE  AFTER 2 LINES.
      GO  TO  MALN.
EOJ.  CLOSE  MAIN-FILE PRINT-FILE.
      STOP  RUN.
```

4-1-2 COBOL 语言元素

一、字符集 COBOL语言所使用的全部字符叫COBOL字符集，它由以下字符组成：

字母 A～Z
数字 0～9
连接符—(同减号)
} 用来组成COBOL字

句号 .(同小数点)
逗号 ,
分号 ;
单引号 ’(撇号)
双引号 ”
左括号 (
右括号)
空格(空白)(为叙述方便，用␣表示)
} 用作标点

大于 >
小于 <
等于 =
} 用于比较(关系运算符)

加号 +
减号 −
乘号(星号) *
除号(斜杠) /
} 用于算术运算(算术运算符)

在IBM PC中还允许使用小写英文字母(a～z)。

二、COBOL字 COBOL字是为表示一定的意思，由COBOL 字符组合而成的最小单位。其长度可为1到30个字符，组成规则是：

1.由字母A～Z (IBM PC还包括a～z)、数字0～9和连接符"—"构成，且以字母开头(过程名除外)。

2.连接符只能出现在字的内部，不允许出现在字的开头和结尾。在一个字内可以包含一个以上的连接符，多个连接符可以邻接，也可不邻接。

3.字由分隔符定界，如空格、标点等。

4.所有的COBOL字可分为两种：保留字和用户字。

保留字：保留字是COBOL语言的专用字，它们具有固定的组成和特定含义，这些字不能由程序员使用。COBOL的保留字见本章附录。

用户字：由程序员指定的COBOL字称为用户字，按其用途可分为：

(1) 数据名：用来对数据项命名。

(2) 文件名：用来对程序中所使用的文件命名。

(3) 条件名：条件名赋给数据项可能采用的一个特定值、一些值的集合或值的范围的名字。

(4) 过程名：COBOL源程序中的过程部分由一系列可执行语句组成，若干条相继语句构成一个过程段，以段名标识；若干个相继的过程段构成过程节，以节名标识。通常把节名和段名统称为过程名。过程名可以用数字开头。

(5) 助忆名(记忆名)：助忆名是与某种外部设备或外部设备的状态相联系的名字。这种联系是在设备部分(或称环境部分)建立的，以后在过程部分中就可以直接用助忆名来表示对应的设备。

三、常字(也叫直接量，也有称它为常量或文字)：常字是一个常数，在程序运行过程中保持不变。COBOL中的常字有三种：

1.非数值常字：由引号(单引号或双引号)括起来的任意ASCII 字符串，但不包括定界符引号。常字的长度是字符串的字符个数，COBOL规定长度应为1～120。

2.数值常字：数值常字是由数字0～9、符号 +、 − 和小数点等字符按下述规则组成：

(1) 至少要有一位数字，最多不超过十八位数字。

(2) 只能包括一个符号，如果有符号，则它必须是数字常字的最左字符，如果一个数值常字不带符号，则是正的。

(3) 只能包含一个小数点，它可出现在数值常字中除最右位置外的任何位置上，不含有小数点的常字被认为是整数。

3.象征常数(又称表意常量或赋形常数)：象征常数是用具有特殊意义的保留字来表达的常数，这些保留字具有明确表示的值。COBOL中使用的象征常数有：

ZERO(或ZEROS、ZEROES) 表示一个或一列数值0。

SPACE(或SPACES) 表示一个或一列空格字符(代码为十六进制20H)。

HIGH-VALUE(或HIGH-VALUES)　表示一个或一列以十六进制"FF"表示的具有最大值的字符。

LOW-VALUE(或LOW-VALUES)　表示一个或多个以十六进制"00"表示的具有最低值的字符。

QUOTE(QUOTES)　表示一个或多个引号字符，但不能用作非数值常字的定界符。

ALL常字　表示一个或多个指定的非数值常字或ALL之外的象征常数。

4-1-3　COBOL语言描述中的一些规定和记号　为了正确使用COBOL语句，应了解COBOL语言中每一种语句和子句的一般格式，在这些格式的描述中使用了统一的规定和记号：

一、所有写成大写字母的字为关键字或保留字。其中划底线的字是必写的；未划底线的是可选择的。

二、凡用中文或用英文小写字母写的字，表示是要由程序员填入的信息，同一格式中出现具有相同作用的不同项时，用中文字后跟一个连接符和一个数字或字母以资区别，中文字的语法含意不变。

三、方括号〔　〕、花括号{　}和省略号…是描述记号，它们不出现在源程序中，它们的作用分别是：

〔　〕　表示其中的语法成分是任选的。

{　}　表示所列出的各语法成分，在程序中使用时必选其一。

…　一般跟在方括号或花括号的后面，表示括号中的语法成分可以出现多次。

4-1-4　COBOL源程序的书写规则

一、程序行的格式

微型机COBOL是ANSI COBOL的一个子集，因此，源程序可以填写在标准的COBOL程序纸上。程序纸的每行有80列，分为四个区：序号区(1～6列)；指示区(第7列)；正文区（8～72列)；注释区(73～80列)。正文区又分为A区(8～11列)和B区(12～72列)。如图4—2所示。

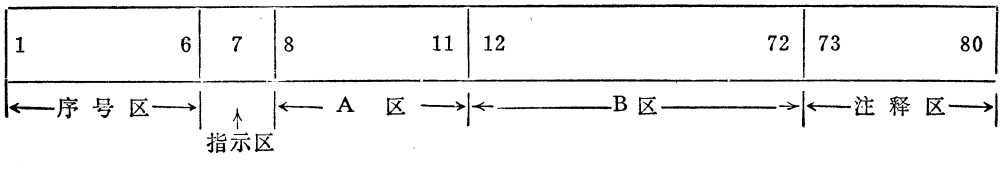


图4-2

1.序号区：序号是一个6位的十进制数，写在1～6列上，用来表示程序的行。序号不是必须的，可以不写。如果使用，则必须按升序排列。

2.指示区：指示区可出现字符—、＊、/和D，它们的作用是：

—：表示该行为继续行；

＊：表示该行为注释行；

/：也表示该行为注释行，但在打印源程序时，这一行打在新的一页的顶部；

D：表示该行为调试行。

3.正文区：A区和B区用于书写源程序。

部分、节和段的标题以及文件描述项指示符"FD"、层号01和层号77必须从A 区的第8列开始书写，其它层是可以出现在A区和B区的任意位置。程序中的所有其它元素都必须写在B

区内。

4.注释区：同注释行类似，用作注释，编译程序对它不作处理。

二、分隔符用法

1.两个相邻的字符串（COBOL 字或常字）之间至少要有一个空格。计算机把连续的多个空格只当作一个空格。

2.逗号、分号或空格可以分隔一串语句或子句，句点必须跟在句子或描述项之后。

3.逗号、分号、句号作分隔符使用时，它们的左边不能留空格，但右边必须留一个空格。

4.逗号或空格可用来作为两个运算量之间的分隔符，或两个下标之间的分隔符。

5.括号的凹侧不得留有空格，凸侧则必须留一空格。

6.算术运算符和关系运算符的左右两边必须有一个空格，但+、-号若出现在数值常字之前时，其后边不能留有空格。

4-2 标识部分和设备部分

4-2-1 标识部分 每一个COBOL 程序都以标识部分开始，它用来说明程序的名字、作者、用途和其它特点。

一、标识部分的一般格式为：

IDENTIFICATION DIVISION.
PROGRAM-ID. 程序名。
〔AUTHOR. 作者注释项〕
〔INSTALLATION. 所属系统或应用范围注释项〕
〔DATE-WRITTEN. 程序编写日期注释项〕
〔DATE-COMPILED. 程序编译日期注释项〕
〔SECURITY. 程序的保密限制注释项〕
〔REMARKS. 其它说明注释项〕

二、说明：

1.标识部分不包含节，只有七个具有固定段名的段。段名本身就隐含了该段的内容。

2.在这些段中只有程序名段是必写的，且必须是第一段。其它各段可有可无，次序也可任意。初学者完全可以不写它们。如4-1-1的例中，只写了程序段，STORE 是程序员给程序起的名子。

3.IBM PC的COBOL不含有REMARKS 段，当用户希望要其它注释时，可在程序中任何地方使用第 7 列加写星号(*)来说明。

4.标识部分是非执行部分。

4-2-2 设备部分 （又称环境部分） 用来说明 COBOL程序编译和运行时所使用的计算机和外部设备，并定义各设备上的文件，描述文件的物理特性。它含有两个节：配置节和输入输出节。

一、一般格式：

ENVIRONMENT DIVISION. （设备部分）
〔CONFIGURATION SECTION.〕 （配置节）

〔SOURCE-COMPUTER. 源计算机名.〕（源计算机段）

〔OBJECT-COMPUTER. 目标计算机名.〕（目标计算机段）

〔SPECIAL-NAMES. 专用名描述项〕（专用名段）

〔INPUT-OUTPUT SECTION. 〕（输入输出节）

〔FILE-CONTROL. 〔文件控制项〕…〕（文件控制段）

〔I-O-CONTROL. 〔输入输出控制描述项〕〕（输入输出控制段）

二、说明：

1.配置节：该节中指定用户所使用的计算机类型、特性及名称。配置节是任选节，它有三个可能的段：

(1) 源计算机段：说明源程序在哪种计算机上进行编译。其格式为：

SOURCE-COMPUTER. 计算机名

〔WITH DEBUGGING MODE〕.

本段的内容作为说明，但如果出现 WITH DEBUGGING MODE 子句时则例外。这个子句使源程序的调试行成为有效，即如果用户在源程序里插入跟踪程序错误的行，例如 D EXHIBIT，且在源计算段写上 WITH DEBUGGING MODE 子句，则该行被编译为程序的一部分；如果没有写这个子句，则该行被忽略。

(2) 目标计算机段：这一段说明运行目标程序的计算机，并提供程序中过程部分分块的有关信息。

IBM PC中，这一段的格式是：

OBJECT-COMPUTER. 目标计算机名

[MEMORY SIZE 整数 { WORDS / CHARACTERS / MODULES }]

〔PROGRAM COLLATING SEQUENCE IS ASCII〕.

在CROMEMCO中的格式是：

OBJECT-COMPUTER. 目标计算机名

[MEMORY SIZE 整数 { WORDS / CHARACTERS / MODULES }]

〔SEGMENT-LIMIT IS 块号〕.

本段的内容是作为说明用。

(3) 专用名段：本段把设备名与用户指定的名字联系起来，亦可改变字符的意义，如小数点和逗号。其格式为：

SPECIAL-NAMES. 〔PRINTER IS 助忆名〕

[ASCII IS { STANDARD-1 / NATIVE }]

〔CURRENCY SIGN IS 常字〕

〔DECIMAL-POINT IS COMMA〕

[SWITCH-n IS comment-ID {OFF / ON} IS 条件名].

其中：① PRINTER子句，让用户对打印机定义一个助忆名，这个助忆名在过程部分的DISPAY语句中使用。

② ASCII子句起注释作用，说明数据均以 ASCII 码表示，如不写此句，也隐含这个说明。

③ CURRENCY子句让用户选择一个货币符号(例如 L) 而不用美元号。用户选择的符号必须是一个非数值常字，但不能是引号、数字或 PICTURE 表示法中使用的任何字符。不使用该子句时，以美元符号 $ 作为货币符号。

④ DECIMAL-POINT 子句让用户采用欧洲传统方式的表示法，即用逗号代替小数点。

⑤ SWITCH-n 子句允许用户在运行时设置开关，范围为 SWITCH-1～SWITCH-8。若使用该子句，则在运行时，计算机会提示用户输入开关的设置。开关的设置为："空格" 代表OFF，"非空格"代表ON。CROMEMCO无此子句。

⑥ 在IBM PC的COBOL中，除ASCII子句外，其它子句中均需要保留字IS。

2.输入—输出节　输入输出节用于建立文件名和外部设备之间的对应关系，它包括两个段：文件控制段和输入—输出控制段。

(1) 文件控制段　主要功能是定义文件，给文件分配它所允许的输入/输出类型。其一般格式为：

FILE-CONTROL.

SELECT 文件名 ASSIGN TO {DISK / PRINTER}

[ORGANIZATION IS { SEQUENTIAL / INDEXED / RELATIVE / LINE SEQUENTIAL }]

[ACCESS MODE IS { SEQUENTIAL / RANDOM / DYNAMIC }]

〔RECORD KEY IS 数据名-1〕

〔RELATIVE KEY IS 数据名-2〕

[RESERVE 整数 {AREA / AREAS}]

〔FILE STATUS IS 数据名-3〕。

其中：① SELECT 子句总处在文件控制段的开头，用来对程序中使用的文件命名，程序中用到的每一个文件都必须用 SELECT 子句命名一次。书写时，SELECT 应从B区开始。跟在

SELECT子句之后的ASSIGN子句，是分配子句，它为文件分配外部设备。例如4-1-1的例子中，在文件控制段中用两个SELECT子句分别定义两个文件。

② ACCESS子句指出文件的存取方式，如果省略，则按顺序方式存取。ORGANIZATION子句指出文件的组织形式，顺序的磁盘文件可以使用两种形式，一种是约定形式ORGANIZATION IS SEQUENTIAL，另一种是ORGANIZATION IS LINE SEQUENTIAL。(CROMEMCO机的COBOL无选用短语LINE SEQUENTIAL)。行顺序结构(LINE SEQUENTIAL)文件是在用户不使用COBOL程序的情况下建立数据文件时产生的。

③ 对索引组织文件要用RECORD KEY子句说明记录键，数据名-1是指定的记录键名。

④ 相关组织文件要用RELATIVE子句说明相关键。数据名-2必须用无符号整数项描述，它不能出现在文件本身的任何记录描述中，其值必须是正的和非零的。

⑤ RESERVE子句用来确定文件缓冲区的个数，在IBM COBOL中这个子句没有作用。

⑥ STATUS子句为文件定义一个状态字，当执行该文件的输入/输出语句时，文件管理系统将自动对状态字赋值，以反映该输入/输出操作执行的情况。

(2) 输入—输出控制段 这一段允许两个或几个文件间共用缓冲区。其一般格式为：

〔I—O—CONTROL.

〔SAME AREA FOR 文件名-1 〔文件名-2〕…〕…〕.

IBM PC COBOL的格式稍有不同，其格式是：

〔I—O—CONTROL.

〔SAME RECORD ARE FOR 文件名-1 〔文件名-2〕…〕…〕.

4-3 数据部分

4-3-1 数据及数据组织

一、数据：数据是COBOL程序处理的对象，它包括数字数据和文字数据。数字数据由数字0～9、小数点以及＋、－号按一定法则组成；文字数据由字母或任何有意义的字符组成。

二、组合项和初等项：在数据的层次结构中，把可进一步分解的数据项称为组合项，把不可分的数据项称为初等项，也叫基本项。组合项的层号小于相邻的后继项的层号。

三、数值项和非数值项：数值项是只含有数值数据的初等项，也叫数字项。根据它在机内的存贮方式可进一步分为：外部十进制项，内部十进制项，二进制项和指标数据项（索引数据项）。非数值项是指由字母、数字及其它符号组成的“字符串”数据项，按照组成它的字符形式可进一步分为：字母项，字符项，字符编辑项和数值编辑项。

四、层号：为了描述数据的层次结构(数据之间的隶属关系)，在COBOL语言中，对不同的层次规定了不同的层号。层号是两位整数。记录的层号固定为01，从属于记录的数据项的层号可在02～49之间。下属的数据项必须比它所属的数据项有较大的层号。

五、记录：指由若干个组合项和基本项组成的具有独立逻辑含义的最大可存取项。能表达一个完整记录的数据名叫做记录名。

六、文件：文件是COBOL程序处理中数据的最大单位，它是一类或几类记录的集合，这些记录按一定方式存放在存贮介质上构成一个文件。

在一个文件中，若所包含的记录具有固定的长度，这种文件叫定长记录文件；若含有的记录长度不固定，这种文件叫变长记录文件。

七、文件的组织方式：指文件中记录的排列方式，它在文件建立时被确定，软件使用过程中其组织方式不变。文件的组织方式一般有：顺序组织、索引组织和相关组织三种。

八、文件的存取方式：文件的存取方式有：

1.顺序存取：按记录在文件中的排列次序进行存取。对顺序组织的文件只能顺序存取，对索引组织的文件可按记录键值的递增次序顺序存取，对相关组织的文件，可按相关键递增次序顺序存取。

2.随机存取：按指定的位置存取文件中的记录。只有按索引组织或相关组织建立的磁盘文件才能按随机方式存取。

3.动态存取：指对同一文件的同一处理过程中，可交替地或随意地采用顺序存取和随机存取两种方式。

九、文件与存贮介质：文件必须存放在计算机外部设备的物理介质上。按介质分，文件可有：打印文件，即打印在纸上的文件；卡片文件，即穿孔在卡片上的文件；磁带文件，即建立在磁带上的文件；磁盘文件，即建立在磁盘上的文件。

十、块：块是文件在外部介质上的信息单位，即是说，文件在外部介质上是分成为块的，而不是直接分成为记录。通常把块叫做物理记录或物理块，而把记录叫做逻辑记录。

4-3-2 文件描述项 数据部分的作用是对输入的初始数据和输出的结果数据进行描述。它分为四个节：文件节、工作存贮节、连接节和屏幕节。数据部分的一般格式如下：

```
DATA DIVISIDN.
〔FILE SECTION.〔文件描述项〔记录描述项〕…〕…〕（文件节）
〔WORKING-STORAGE SECTION. 〔数据描述项〕…〕（工作存贮节）
〔LINKING SECTION.〔数据项描述项〕…〕（连接节）
〔SCREEN SECTION.〔屏幕描述项〕…〕（屏幕节）
```

一、一般格式

数据部分文件节中的文件描述项(也叫文件描述体)，是用来指定有关文件标号、文件物理块的大小、记录名和记录的大小等信息。其一般格式为：

```
FD 文件名 LABLE {RECORD  IS}
                {RECORDS ARE}
                {OMITTED  }
                {STANDARD }
〔VALUE OF FILF-ID IS 常字〕
[DATA {RECORD IS} 数据名-1〔数据名-2〕…]
     {RECORDS ARE}
[BLOCK CONTAINS 整数-2 {CHARACTERS}]
                       {RECORDS   }
〔RECCRD CCNTAINS 〔整数-1 TO〕 整数-2 CHARACTER〕
〔CODE-SET IS 字母名〕
```

$$\left[\underline{\text{LINAGE}}\ \text{IS}\ \left\{\begin{matrix}\text{数据名-3}\\ \text{整数-3}\end{matrix}\right\}\ \text{LINES}\right]$$

$$\left[\text{WITH}\ \underline{\text{FOOTING}}\ \text{AT}\ \left\{\begin{matrix}\text{数据名-4}\\ \text{整数-4}\end{matrix}\right\}\right]$$

$$\left[\text{LINES}\ \text{AT}\ \underline{\text{TOP}}\ \left\{\begin{matrix}\text{数据名-5}\\ \text{整数-5}\end{matrix}\right\}\right]$$

$$\left[\text{LINES}\ \text{AT}\ \underline{\text{BOTTOM}}\ \left\{\begin{matrix}\text{数据名-6}\\ \text{整数-6}\end{matrix}\right\}\right]\right]$$

二、说明：

1.FD 是文件描述项指示符，也叫层指示符，它后面的文件名必须与文件控制项中定义的一致。

2.LABLE子句用来说明文件标号。OMITTED 选择指出某个文件没有标号，对于打印文件必须选择此项。STANDARD 选择指出一个文件有标号且符合系统的规定，对于磁盘文件必须有此项。

3.VALUE OF 子句用于指定磁盘文件标号记录的具体值。其中常字用来表示这个具体值，它应是非数值常字（IBM PC 还允许它是数据名）。在微机中，这个常字是根据操作系统规则来规定的文件名。

如果一个文件指定为“PRINTER”，这个文件不用标号，因而VALUE子句就不能包括在相应的FD中。如果一个文件指定给磁盘，那么在相应的FD中，既要有 LABEL RECORDS STANDARD子句，也要有VALUE子句。

4.DATA RECORD(S)（数据记录）子句用来描述文件记录格式的种类。当一个文件仅由一种形式的记录组成时，则用“RECORD IS”，且后面跟一个记录名；若文件由多种形式的记录组成时，则用“RECORDS ARE”，其后列出所有的记录名。子句中的数据名，即是属于本文件的所有种类的记录的名字。记录名的次序是任意的，但是子句中用到的记录名必须与记录描述中定义的记录名一致。

5.BLOCK（物理块）子句，用于说明文件的物理块的大小。选择CHARACTERS或RECORDS，表示计算块大小的单位。（这个子句在IBM COBOL中无效）。

6.RECORD CONTAINS子句用来指定数据记录的大小，整数-1表示最小记录的长度，整数-2表示最大记录的长度。仅当文件中的记录是可变的，本子句才有必要选用。

7.CODE-SET（代码设置）子句。用来规定非磁盘文件。这个子句仅在 IBM COBOL 的文件中使用。

8.LINAGE（行）子句。对于指定给打印机的文件来说，这个子句提供了规定一页可打印多少行的方法。CROMEMCO COBOL没有这个子句。

4-3-3 记录描述项 对于组成一个文件的所有记录必须紧接在文件描述之后进行描述。记录描述的主要任务是指出一个记录的层次结构及构成这个记录的所有数据的特征。“记录描述”由一组“数据描述”组成，这一组数据描述刻划了记录中所有数据项的层次关系；数据项所代表的数据是数值的还是非数值的；在内存中占有多大的存贮区域；等等。因此，记录描述的一般格式是：

01 记录名

$$\left\{层号\left\{\begin{matrix}数据名\\FILLER\end{matrix}\right\}(形象子句)\right\}\cdots$$

一、数据项描述的一般格式

按照数据项在记录中出现的先后次序逐个加以描述。每个数据项描述项包括层号、数据名和若干相对独立的子句，其一般格式为：

$$层号\left\{\begin{matrix}数据名\\FILLER\end{matrix}\right\}〔REDEFINES子句〕〔PICTURE子句〕$$
$$〔USAGE子句〕〔VALUE子句〕$$
$$〔OCCURS子句〕〔JUSTIFIED子句〕$$
$$〔SIGN子句〕〔BLANK\quad WHEN\quad ZERO子句〕$$

二、说明：除了层号、数据名和REDEFINES 子句的位置是固定的以外，其它子句出现的次序是任意的。数据名用来标识要描述的数据项，过程部分的语句可通过这里定义的数据名来访问具体的数据项。FILLER 是一个专用数据名，它作为初等项，可以是记录的组成项或某个组合项的组成项。

4-3-4 PICTURE子句

一、子句格式

PICTURE子句用一种形象的方式描述基本数据项，其一般格式为：

$$\left\{\begin{matrix}PIC\\PICTURE\end{matrix}\right\}\ IS\quad 形象字符串$$

式中的形象字符串是用来描述该数据项的类型、长度和编辑要求的。

二、说明：

形象字符串由以下三种字符按一定的规则组成。

1.数据字符(A、X、9)

A代表一个字母或空格；X代表ASCII字符集中的任一个字符；9代表一个数字，在PIC子句中最多可用18个 9 。

2.运算字符(S、V、P)

S表示所描述的数据可以带有一个正负号。它在形象串只能出现一次，且是在左起第一个字符的位置上。当数据为正时，S可以不写，当数据为负时，必须用S描述。

V指出数值项中一个隐含的小数点位置。它不占存贮单元，也不计入数据项的长度。一个形象串中只允许出现一个V。

P用来指定隐含小数点的位置，不占有内存空间。如果P是该子句中最左边的字符，则隐含的小数点就在P字符串的左边，而每一个P暗指小数点后的一个零；如果P是该子句中最右边的字符，则隐含的小数点在P字符串的右边，每一个P也暗指一个零。

3.编辑字符

编辑字符指示具体的编辑方法，根据编辑功能的不同，可分为替换字符、插入字符和浮动插入字符三类：

(1) 替换字符 (Z，＊) 每一个字符表示一个数字位置，如果该位置是无效左零，则编辑时，每一个由Z或＊表示的无效左零被一个空格或＊代替。

(2) 插入字符

B，0，，，/为简单插入字符，它们指示编辑时在该字符所在的位置上分别插入空格、

数字 0 、逗号(,)和斜杠(/)。除逗号不能是形象串的最后一个字符外，简单插入字符可出现在形象串的任何位置上。

· 专用插入字符，它表示一个实际小数点应插入的位置。形象串中实际小数点左边只能出现相同的形象字符。在一个形象串中只能使用一个 "·"，且不能出现在最后，也不能与P及V同时使用。

+，－，$，CR，DB 为固定插入字符。正、负号只能固定地出现在形象串的左端或右端，它们指示编辑时在对应字符位置插入＋号或－号。用"＋"时，数据项的值为正时插入＋，为负时插入－；用"－"时，数据项的值为正插入空格，为负插入－号。

$ 固定出现在形象串的左端，表示在该位置上要插入一个 $ 。

CR，DB 贷方和借方符号。它们只能固定地放在形象串的最后，当对应的数据项的值为正时，在CR或DB的位置出现两个空格；当对应的数据项的值为负时，则在该位置上打印CR或DB。＋、－、CR、DB是互相排斥的。

(3) 浮动插入字符串

浮动插入字符串是由 $ 、＋或－中的任一个字符组成的连续字符串。当它出现在 PIC子句的形象串中时，表示要指定浮动插入编辑，其基本功能是：取消无效左零，代之以空格，而且在有效数字左边一个位置上插入某个浮动字符($ 、＋或－)。在数值编辑项中插入浮动插入字符时，浮动字符从左至右移动，当遇到下述情况之一时，浮动字符被插入：

① 遇到第一个非零数字，浮动字符插在这个非零数字的前一位。

② 遇到实际小数点，浮动字符插在小数点前的一个数位。

③ 遇到浮动串中最右一个浮动字符所在的数位时，浮动字符插在这个数位上。

在使用浮动插入时需注意：

第一，不允许只在小数部分写浮动字符。若在全部数位上都写有浮动字符，对于数值为 0 的数据项，编辑结果是全部置为空格（若用替换字符＊，则结果是除实际小数点外，全部置为＊)；对于数值不为0的数据项，编辑结果是：小数点左边的插入规则与上述的①、②、③相同，小数点右边的形象字符视作形象字符 9 。

第二、浮动插入字符的个数应足够多，以免截去数据的高位数。

第三，在同一形象串中只能使用一种浮动插入字符串，也不能与替换字符 (Z，＊)同时使用。

第四，简单插入字符出现在浮动插入字符串或替换插入字符串中间时，一起参加浮动。若在它的位置上出现有效数字，则它执行正常的简单插入功能，否则它被看作一个浮动字符(或替换字符)。

4-3-5 USAGE(用法)子句

一、一般格式：

$$\text{USAGE IS} \left\{ \begin{array}{l} \underline{\text{COMPUTATIONAL}} \\ \underline{\text{COMP}} \\ \underline{\text{COMPUTATIONAL-3}} \\ \underline{\text{COMP-3}} \\ \underline{\text{DISPLAY}} \\ \underline{\text{INDEX}} \end{array} \right\}$$

二、说明：

1.USAGE子句指定数据项中的数据在计算机内存中的存贮方式。它可用来描述任何一层的数据项，当描述组合项时，那它就同时描述了从属于该项每一个组成项。一个数据项的USAGE子句不能与它所属项的USAGE子句相矛盾，也不能与它的PIC子句相冲突。

2.COMPUTATIONAL（COMP）表示数据按二进制项方式存贮，其形象字符串可由1～4个字符9组成。

3.COMPUTATPONAL-3（COMP-3）表示数据按内部十进制项方式存贮。

4.DISPLAY表示数据是按非数值项或外部十进制项方式存贮。

5.INDEX表示数据项是指标数据项，这种数据项不能用PIC子句描述。

6.不用USAGE子句时，则认为该项数据是按DISPLAY方式存贮。

4-3-6 JUSTIEIED（右对齐）子句

一、一般格式：

$$\left\{\begin{matrix}\text{JUSTIFIED}\\ \text{JUST}\end{matrix}\right\}\ \text{RIGHT}$$

二、说明：

1.JUST子句用于接收项为非数值类型的初等项时，实现右对齐，即从最右字节开始，从右至左逐个安置送入的数据，不足部分填以空格，超出部分把左边多出的字符截去。

2.当接收ALL〈常字〉这种非数值项时，右对齐子句不起作用。

4-3-7 VALUE（赋初值）子句

一、格式：

VALUE IS 常字

二、说明：

1. VALUE子句仅用于工作存贮节数据项的描述，在子句中的常字的类型必须与它所要赋值的数据项类型一致。

2. VALUE子句给出的值应适合PIC子句描述的范围。

3. 用于条件名描述体中的VALUE子句的格式是：

$$88\quad \text{条件名}\quad \left\{\begin{matrix}\text{VALUE IS}\\ \text{VALUE ARE}\end{matrix}\right\}\left\{\begin{matrix}\text{常字-1}[\text{常字-2}]\cdots\\ \text{字常 -3 THRU 常字-4}\end{matrix}\right\}$$

88是专用层号，它用来引导条件名描述体。88层前面第一个非88层描述体描述的数据项必须是一个初等项，称为条件变量。条件名是条件变量的值（由常字-1，常字-2等指出）或值的范围（由常字-3 THRU常字-4指出）的名称。

常字的类型必须与条件变量的类型一致。

4-3-8 OCCURS（重现）子句

一、格式：

OCCURS 整数 TIMES

二、说明：

1. OCCURS 子句说明所描述数据项以相同形式可重复的次数，因此，它可定义一个表，即数组。

2. OCCURS 子句不能用在01层、77层和88层的描述项中。

3．对于任一数据项，最多可用三个OCCURS子句进行描述。数据项的每次重复出现，构成一个表元素，因而重现序号可用来确定表元素在表中的位置。

4-3-9 REDEFINES（重定义）子句

一、格式：

层号 数据名-1 REDEFINES 数据名-2

二、说明：

1．REDEFINES 子句用来重新定义一个已被定义过的存贮区，使它可以存放具有不同名称和类型的数据项。数据名-1是数据项存贮区的新名称，而数据名-2是数据项存贮区已有的名字。

2．数据名-1和数据名-2标识的数据项应具有相同的层号，且它们之间不允许有其它相同层号的数据项出现。

3．重定义子句不能放在文件节的01层使用，66层及88层的数据项不能重新定义。

4．在重定义子句的数据项描述中以及下属项的描述中不能使用VALUE子句，也不能包含OCCURS子句。

5．数据名-2的描述项不能包含OCCURS子句，也不能包含有OCCURS子句的描述项。

4-3-10 SIGN（符号）子句

一、格式：

$$\text{SIGN IS } \left\{\begin{matrix}\underline{\text{TRAILING}}\\ \underline{\text{LEADING}}\end{matrix}\right\} [\underline{\text{SEPARATE}} \text{ CHARACTER}]$$

二、说明：

1．当只选TRAILING时，表示符号嵌在最右字节中；若只选LEADING，则表示符号嵌在最左字节中。在这两种选择中，符号均不单独占用存贮单元。若选用 TRAILING SEPARATE 时，表示符号单独放在最右字节；若选用 LEADING SEPARATE 时，表示符号单独放在最左字节。这后两种选择，符号单独占一个字节。

2.若形象字符串中没有S字符，则不能使用SIGN子句；若形象串中有S字符，而SIGN子句省略，则认为隐含地选择SIGN IS TRAILING形式。

3.当 SIGN 子句说明组合项时，则它所说明的符号格式适用于任一个带符号的从属的外部十进制项。

4.SIGN子句只用来说明外部十进制项的符号表示方法。但对 IBM PC COBOL，只要用 CODE-SET 子句描述一个文件，则该文件的所有带符号的数字数据项都必须用 SIGN IS SEPARATE子句来加以描述。

4-3-11 BLANK（遇零置空）子句

一、格式：

BLANK WHEN ZERO

二、说明：

BLANK子句用于数值项时，使数值为零的数值项的内容置成空格。用于数值编辑项时，作用也相同。

4-3-12 SYNCHRONIZED（同步安置）子句

一、格式：

$$\left\{\begin{matrix}\text{SYNCHRONIZED}\\ \text{SYNC}\end{matrix}\right\}\left[\left\{\begin{matrix}\text{LEFT}\\ \text{RIGHT}\end{matrix}\right\}\right]$$

二、说明：

在IBM COBOL 中使用，它使内存以一种更有效的方式为数据分配内存空间。

4-3-13 工作存贮节

数据部分的第二节是以WORKING-STORAGE SECTION 为标题的工作存贮节。它用于非输入输出数据的描述，即对运算过程的中间结果，或用作累加的数据项的描述。描述时所使用的子句有：

OCCURS
PICTURE
REDEFLNES
USAGE
VALUE

本节的数据描述体可使用的层号为01～49（组合项），此外还可以使用77层号(孤立的数据项)。

4-3-14 连接节

连接节是数据部分的第三个节，它只出现在被调程序中，用来描述那些在主调程序中确定存贮区，而为被调程序使用的数据项。其格式为：

LINKAGE SECTION.
〔独立数据描述体〕…
〔记录描述体〕…

在这一节中，亦用数据名和已介绍过的各种属性子句来描述各种数据项，但这些数据项只是形式上存在，并不给它们分配实际的存贮单元，因而数据描述体中不能使用 VALUE 子句（88层除外）。

4-3-15 屏幕节（CROMEMCO COBOL未用）

屏幕节用于定义有关显示的屏幕格式。它由屏幕数据描述体组成。屏幕数据包括初等项和组合项，初等屏幕项定义在屏幕范围内的个别显示和/或数据项字段。组合屏幕项用来命名被接收的或与一过程区语句一起显示的任一组初等屏幕项。

一、组合屏幕描述体的格式：

层号 屏幕名〔AUTO〕〔SECURE〕〔REQUIRED〕〔FULL〕.

二、初等屏幕项的格式：

层号 〔屏幕名〕〔BLANK SCREEN〕
〔LINE NUMBER IS 〔PLUS〕整数-1〕
〔COLUMN NUMEER IS 〔PLUS〕整数-2〕
〔BLANK LINE〕〔BELL〕〔UNDERLINE〕
〔REVERSE-VIDEO〕〔HIGHLIGHT〕〔BLINK〕
〔FOREGROUND-COLOR 整数-3〕
〔BACKGROUND-COLOR 整数-4〕
〔VALUE IS 常字-1〕

```
[{PICTURE} IS 字形字符串  {FROM 常字-2}]
[{PIC    }                {     标识符-1}]
[TO {标识符-2}]]
   {USING 标识符-3}
[BLANK WHEN ZERO]
[{JUSTIFIED} RIGHT ][AUTO]
[{JUST     }       ]
[SECURE][REQUIRED][FULL]
```

这两种格式中的层号均为01～49内的整数。

4-4 过程部分

过程部分是COBOL程序的执行部分，它指明了对数据的各种处理，其基本格式是：

```
PROCEDURE DIVISION.
[节名 SECTION.
[段名、[句子]…]…]…
```

句子由一个语句或若干相继的语句组成，语句之间可用分号或空格分开。句子以句号(.)和一个空格结束。

语句由动词开头，后跟相应的操作数和短语。语句分为两类：强制性语句（或称命令语句）和条件语句。一个强制性语句规定无条件地执行某一个动作，它由一个动词及操作对象组成。条件语句规定了一个条件，根据这个条件是否满足而决定程序流程应走那条路径。

按照语句的功能，大体上可分为六类：文件处理；算术运算；数据传送；控制转移；字符处理；表处理。这一节先对文件处理语句加以说明。

4-4-1 OPEN（打开文件）语句

一、格式：

```
OPEN {[{INPUT  }        ]
     {[{OUTPUT }文件名…]
     {[{I-O    }        ]…
     {[{EXTEND }        ]
```

二、说明：

1.在执行文件操作之前，必须先用OPEN语句把文件打开，并且在设备部分的 SELECT 和数据部分的文件节中对文件已有说明。

2.选用OUTPUT时，文件以输出方式打开，以后只能用WRITE语句向文件写入记录，以建立一个新文件。

3.选用INPUT时，文件以输入方式打开，以后只能用READ语句读取它的记录。

4.选用I-O时，按I-O方式将文件打开，然后可用READ、WRITE、REWRITE和DELETE等语句来读取、追加、更改和删除文件的记录。这种方式只适用于已建立的磁盘文件。

5.选用EXTEND时，是以扩充方式打开已建立的顺序文件，此后可用 WRITE 语句把要补充的记录安置在该文件的末尾。这一方式也适用于IBM COBOL中的行顺序文件。

4-4-2 READ（读）语句

READ语句从文件中读取数据。根据文件组织和读取方式的不同，这一语句可用三种格式。

一、READ 文件名 RECORD〔INTO 数据名〕〔AT END 强制语句〕

对于顺序组织文件，使用这一格式的读语句可按读写指针的当前位置将指定文件的一个记录读入内存中该文件的缓冲区，并使指针指向下一个记录。

若选用 INTO短语，则表示记录不仅读入文件缓冲区，同时还送入数据名所标识的存贮区。若选用AT END短语，表示当读到文件结束时，将执行强制语句。

二、READ 文件名 NEXT RECORD〔INTO 数据名〕
〔AT END 强制语句〕

对于非顺序组织文件，使用这种格式的读语句，可按顺序方式读取文件中的记录。

三、READ 文件名 RECORD〔INTO 数据名〕
〔INVALID KEY 强制语句〕

对非顺序组织文件，使用这种格式的读语句，可随机地或动态地从文件中读取记录。在读语句执行之前，若是索引组织文件，需对记录键赋值；若是相关组织文件，需对相关键赋值。然后，读语句根据键值从文件中读取指定的记录。若选用 INVALID 短语，则在找不到所需记录时，执行强制语句。

4-4-3 START（起动）语句

一、格式：

START 文件名〔KEY IS {GREATER THAN / NOT LESS THAN / EQUAL TO} 数据名〕

〔INVALID KEY 强制语句〕

二、说明：

1.本语句用来说明对非顺序组织文件进行读操作的开始位置。

2.数据名是指记录键或相关键的名字，在执行START 语句之前，必须先对此键名赋值。

3.在执行START语句之前，必须用INPUT方式或I-O方式将文件打开。

4.执行 START 语句时，将文件中各记录对应的键值与指定的键值逐个进行比较，找出第一个其值大于（当选择GREATER项时）、或不小于（当选择NOT LESS 项时）或等于(当选择 EQUAL项时）指定键值的记录就作为随后读取的第一个记录，若找不到满足给定条件的记录，则发生无效键条件，若选用了INVALID短语，就将执行强制语句。

4-4-4 WRITE(写)语句

WRITE 语句将数据记录从文件缓冲区或某个内存区写到文件所在的外部设备上。写语句执行之前，必须用OUTPUT方式或I-O方式或 EXTEND 方式把文件打开。写语句有三种格式：

一、格式I：

WRITE 记录名〔FROM 数据名〕

说明：

1.记录名必须在数据部分文件节的一个文件中用01层描述项描述过，执行写语句时，将

把一个逻辑记录输出给对应的文件。

2.若选择FROM短语，则在执行写语句时，先将短语中数据名所标识的存贮区中的内容传送到以记录名标识的文件缓冲区内，然后再输出到指定的外部设备上。

3.应用格式I，可对磁盘文件进行顺序写。

二、格式II：

WRITE 记录名〔FROM 数据名〕〔{AFTER / BEFORE} ADVANCING

{运算量 LINES / PAGE}〕

〔AT {END-OF-PAGE / EOP} 强制短语〕

说明：

1.这一格式的前半部分与格式 I 相同，选用后半部分时，只限用于行式打印机的输出文件。

2.ADVANCING 短语控制打印机输出时的走纸功能。其中运算量是整数或数据名，取值范围为0～60（IBM COBOL规定为0～120），为0时，表示不换行；为 1时，表示正常换行；为3时表示走3行（即空2行）；等等。

3.选择 AFTER，表示先走纸后打印；选择 BEFORE，则表示先打印后走纸。若这两项均不选择，则隐含选择“AFTER ADVANCING 1 LINE”，即走纸一行后打印。对同一个打印文件，执行写语句时必须作同样的选择。

4.选用PAGE，表示打印时需要换页。若前面选用 AFTER，则表示从下一页开始打印；若前面是BEFORE，则表示打完本记录后跳到下一页开头。

5.如果指定END-OF-PAGE短语，则必须在有关文件的文件描述中指定LINAGE子句。带有EOP短语的 WRITE 语句在一页的最下部分打印或换行时，就产生页结束条件，在执行完WRITE语句后，执行短语中的强制语句。(CROMEMCO COBOL无此短语)。

三、格式III：

WRITE 记录名〔FROM 数据名〕〔INVALID KEY 强制语句〕

这种格式的写语句可对非顺序组织文件进行随机写。

对相关组织文件可随机地建立和追加，当随机地写入一个记录时应先对相关键赋值。对索引组织文件可随机地写入一个记录，此时文件必须已经作为 I-O 方式打开，即文件已经建立，并且应该先对记录键赋值。在这两种情况下，如果键所对应的记录位置上已有记录，或者磁盘可供分配的空间已满，则发生无效键条件。当无效键条件成立时，若选择了INVALID短语，则执行短语中的强制语句。

4-4-5 REWRITE（重写）语句

重写语句可对用I-O 方式打开的文件中的记录进行更改。根据存取方式的不同，重写语句有两种格式：

一、格式I：REWRITE 记录名〔FROM 数据名〕

这一格式的重写语句用于顺序存取方式的文件。记录名标识出被修改的逻辑记录，它必

须在文件节中定义过。

REWRITE语句只能修改刚用READ语句读出的当前记录。

二、格式II：REWRITE 记录名〔 FROM 数据名〕

〔INVALID KEY 强制语句〕

这一格式的重写语句用于随机方式存取文件。在执行REWRITE 语句之前，对索引文件要先对记录键赋值；对相关文件要先对相关键赋值，以便指出需要修改的记录。如果在文件中找不到该记录，则无效键条件成立。

4-4-6 DELETE（删除）语句 删除语句用来删除一个文件的记录。它有两种格式：

一、格式I：DELETE 文件名 RECORD

删除按顺序方式存取的文件记录，文件名指出欲删除记录所属的文件，并先按I-O方式打开。删除的记录是刚用READ语句读出的当前记录。

二、格式 II：DELETE 文件名 RECORD

〔INVALID KEY 强制语句〕

这一格式的删除语句用于删除按随机方式存取的文件记录，不要求事先执行READ语句，但要求先对键名赋值。若按给定的键值在文件中找不到相应的记录，则无效键条件成立。

4-4-7 CLOSE（关闭）语句

格式：CLOSE {文件名〔WITH LOCK〕}…

一个文件处理完后，必须用CLOSE语句将该文件关闭。关闭后，未重新打开之前，不能执行任何有关文件操作的语句。

文件名标识一个在程序中按某种方式被打开的文件，若使用 LOCK，表明在现行程序的后继语句不能再打开这个文件；若无 LOCK，则表明在程序中稍后的语句仍能重新打开这个文件。如果试图关闭一个未打开的文件，则会产生一个运行错误。

4-4-8 DECLARATIVES和USE （说明节和使用语句）

一、格式：

PROCEDURE DIVISION.

DECLARATIVES.

{节名 SECTION. USE 语句.

{段名.{语句}…}…}…

END DECLARATIVES.

二、说明：

1.说明节是过程部分的一个任选部分，如果选用，就必须放在过程部分的开头，其作用是说明一个过程体——“说明过程体”，一个在程序员预计不到的非正常情况出现时才被执行的过程体。

2.关键字DECLARATIVES和 END DECLARATIVES 书写时必须从正文区的A区开始，并且后面跟句点和空格。

3.一个说明过程体是从节名开始到遇到另一个带USE语句的节标题为止或到遇到 END DECLARATIVES时为止。

4.USE语句的格式为：

USE AFTER STANDARD {EXCEPTION / ERROR} PROCEDURE ON

{文件名-1〔文件名-2〕… / INPUT / OUTPUT / I-O / EXTEND}

USE语句的作用是用来确定该说明过程体的应用范围。

4-5 算术运算语句

4-5-1 与运算有关的选择性短语 算术运算语句中可包括任选的短语，以对运算结果作相应的处理。选择性短语有三个，它们是：

一、ROUNDED（舍入）选择

在运算过程中，若将小数点对齐后，运算结果的小数点后的位数多于存放计算结果的"接收项"所描述的小数点后的位数，ROUNDED选择将结果进行四舍五入。无此选择时，会将多余的位数截去。

计算结果为负时，先对其绝对值进行四舍五入处理，然后再把它变为负值。

如果接收项的 PIC 描述中，在最低整数位上出现有P时，则四舍五入和简单截去都将以实际分配内存的最右位置为基准进行。

二、SIZE ERROR（高位溢出处理）选择

SIZE ERROR 选择的格式为：

ON SIZE ERROR 强制语句

如果在算术运算语句中含有SIZE ERROR选择，当出现高位溢出（截去高位）时，则作为结果的数据名的值不变，转而执行高位溢出处理选择中的强制语句。

三、GIVING（接受项）选择

GIVING选择表明跟在GIVING后面的数据名是运算结果的"接受项"。

4-5-2 ADD（加法）语句

一、格式：

ADD {数值常字－1 / 数据名－1}…{TO / GIVING}数据名－n

〔ROUNDED〕〔ON SIZE ERROR 强制语句〕

二、说明：

1.加法语句是将两个或多个数值相加，并把它们的结果存贮起来。

2.当选用TO时，把前面所有运算量的值相加，并将和加到由数据名－n所标识的数据项中去，再将结果作为数据名－n的新值。当选用GIVING时，将GIVING 前的所有运算量的值相加，其和直接送入数据名－n中。

4-5-3 SUBTRACT(减法)语句

一、格式：

SUBTRACT {数值常字－1 / 数据名－1} …FROM

{数值常字－m GIVING数据名－n / 数据名－m 〔GIVING数据名－n〕}

〔ROUNDED〕〔ON SIZE ERROR 强制语句〕

二、说明：

1.减法语句把FROM之前的所有运算对象的值相加，并从FROM 后面那个项的值中把上述的“和”减去。

2.如果有GIVING选择，则结果（差）存放在数据名－n中，否则结果存放在数据名－m中。

4-5-4 MULTIPLY(乘法)语句 乘法语句把两个数值项相乘，并存贮其积。

一、格式：

MULTIPLY {数值常字－1 / 数据名－1} BY

{数值常字-2 GIVING 数据名-3 / 数据名-2〔GIVING 数据名-3〕}

〔ROUNDED〕〔SIZE ERROR 强制语句〕

二、说明：

1.其作用是将{数值常字-2 / 数据名-2}与{数值常字-1 / 数据名-1}

相乘，然后将积存放在数据名-2或数据名-3标识的数据项中（当有GIVING选择时）。

2.如果省略GIVING，则第二个运算量必须是数据名，它同时用作乘积的接收项。

4-5-5 DIVIDE(除法)语句 除法语句实现两个数值项相除，并存贮其商。

一、格式：

DIVIDE {数值常字-1 / 数据名-1} {BY / INTO} {数值常字-2 / 数据名-2}

〔GIVING 数据名-3〕〔ROUNDED〕

〔SIZE-ERROR强制语句〕

二、说明：

1.DIVIDE…BY把第一个运算量作为被除数，第二个运算量作为除数。在这种情况下，若无GIVING选择，则第一个运算量必须是一个数据名，除商存贮在数据名-1中。

2.DIVIDE…INTO把第一个运算量作为除数，第二个运算量作为被除数。在这种情况下，若无GIVING选择，则第二个运算量必须是一个数据名，除商存放在数据名-2内。

3.在除法运算中，无论何时被零除都导致高位溢出。

4-5-6 COMPUTE（计算）语句 计算语句用来求一个算术表达式的值，并存贮计算结果。

一、格式：

COMPUTE {数据名-1 〔ROUNDED〕}…

$$= \left\{\begin{array}{l}\text{数值常字}\\ \text{数据名-2}\\ \text{算术表达式}\end{array}\right\} 〔\text{SIZE ERROR 强制语句}〕$$

二、说明：

1.数据名-1存放计算结果，它可以是数值项也可以是数值编辑项。一个计算语句可有多个接收项。

2.算术表达式是指由+(加)、-(减)、*(乘)、/(除)、**(乘方)等五种算术运算符和圆括号等将数据名或数值常字连结起来的式子。其运算次序是：有括号时，先括号内后括号外；无括号或在同一括号内，则运算次序为：单目运算（只连接一个运算量)、乘方、乘与除、加与减。同级运算符（例如乘除和加减)，则按书写顺序从左至右依次运算。

3.表达式的书写格式是：运算符及等号两边必须留空格，但+、-号作为单目运算符时，若它后面是常字则不必留空格。圆括号的凹侧不留空格，凸侧要留空格。

4-6 数据传送语句

数据传送语句，用于实现少量数据的输入、输出和内存中不同存贮区域之间的数据交换。

4-6-1 ACCEPT（接收）语句

ACCEPT语句用来从控制台、终端设备或从专用数据区接收少量数据。这一语句可有四种有效格式：

一、格式I：

$$\text{ACCEPT}\quad \text{数据名}\quad \text{FROM}\left\{\begin{array}{l}\text{DATE}\\ \text{DAY}\\ \text{TIME}\\ \text{ESCAPE KEY}\end{array}\right\}$$

利用格式I语句可得到若干标准值中的任一种。这些标准值的格式是：

1.DATE(日期)：六位十进制数，格式为YYMMDD(年、月、日)。例July4，1976表示为760704

2.DAY(天数)：五位十进制数，格式为YYNNN，这里前两位是年号的低2位，NNN是一年中的天数(1～366)。例如January30，1981表示为81030。

3.TIME(时间)：八位十进制数，格式为HHMMSSFF，这里HH表示小时（从00到23)，MM表示分钟(从00到59)，SS表示秒(从00到59)，FF表示百分之一秒(从00到99)。

例如9:30:53.72表示为09305372。

4.ESCAPE KEY(退出ESC键)：由退出键来产生二位代码，以便结束当前被处理的格式III或格式IV ACCEPT语句。

ESCAPE短语在CROMEMCO机上没有使用。

二、格式II：

ACCEPT 数据名

执行此语句时，程序暂停，将从系统隐含的设备上（如控制台打字机）输入数据。在打入回车键后，表示数据输入完毕，程序将把输入的数据按标准定位规则送入数据名所标识的

内存区内。如果输入的字符比接收域（即接收项的长度）短，则多余部分以空格或0(视接收项是非数值型或数值型而定）填充，反之将把多余部分截去。

三、格式III：

ACCEPT 定位说明 标识符[WITH {[{SPACE-FILL / ZERO-FILL}] / [{LEFT-JUSTIFY / RIGHT-JUSTIFY}] / TRAILING-SIGN / PROMPT / UPDATE / LENGTH-CHECK / AUTO-SKIP / BEEP / EMPTY-CHECK / NO-ECHO} …]

说明：

1.这一格式的ACCEPT语句把数据接收进一个显示屏幕上规定位置的区域内。位置说明和接收体（标识符）说明用来定义屏幕上数据输入区的位置和特征。

2.使用ACCEPT的这一格式时，必须遵守如下的语法规则：

(1) 在同一个ACCEPT语句中不可同时规定SPACE-FILL和ZERO两个选择项。

(2) 在同一个ACCEPT语句内不可以同时规定LEFT-JUSTIFY和 RIGHT-JUSTIFY 两个选择项。

(3) 若标识符是一个数字编辑项，则不应规定UPDATE选择。

(4) 仅设标识符是一个初等数字数据项时，可以规定TRAILING-SIGN选择。若标识符是不带符号的项，则略去TRAILING-SIGN选择。

(5) 对于一个字母数字或字母数字编辑的标识符来说，若不规定ZERO-FILL选择时采用SPACE-FILL选择，而若不规定RIGHT-JUSTIFY选择时采用LEFT-JUSTIFY选择。

(6) 对于一个数字或数字编辑的标识符来说，若不规定SPACE-FILL选择，则采用ZERO-FILL选择。

四、格式IV：

ACCEPT 屏幕名 〔ON ESCAPE 强制语句〕

ACCEPT语句的格式IV，用来接收完全格式化屏幕的数据。若在屏幕节中的屏幕名标题下有格式化的注释并要在屏幕上显示该注释，那么在 ACCEPT 屏幕名语句之前必须执行 DISPLAY 屏幕名语句。

ACCEPT语句的格式 IV 把显示器的信息传送给由屏幕节屏幕名规定的所有 TO 和（或）USING字段，或从属于屏幕名的任一屏幕项。

格式III和格式IV只在IBM PC中使用，更具体的说明可查阅专用手册。

4-6-2 DISPLAY(显示)语句 显示语可在指定的设备上输出少量的数据。它可有两种格式：

一、格式I：

$$\underline{\text{DISPLAY}}\ \left\{\begin{matrix}\text{数据名}\\ \text{常字}\end{matrix}\right\}\cdots[\underline{\text{UPON}}\ \text{助忆名}]$$

说明：

1.若不用UPON选项，则在系统隐含指定的输出设备上输出数据。若有UPON选项，则在助忆名所指定的设备上输出数据。其中助忆名需在设备部分配置节的专用名段赋给某个设备。

2.若要输出较多的数据，可用多个DISPLAY语句，每执行一个DISPLAY语句，就从新的一行开始输出。（这一格式适用于CROMEMCO COBOL）。

二、格式II：

$$\underline{\text{DISPLAY}}\ \left\{[\text{位置说明}]\left\{\begin{matrix}\text{数据名}\\ \text{常　字}\\ \underline{\text{ERASE}}\end{matrix}\right\}\cdots\begin{matrix}[\text{UPON助忆名}]\\ [\text{屏幕名}]\end{matrix}\right\}$$

说明：

1.这一格式的DISPLAY语句可把数据送到屏幕或打印机上。当指定UPON助忆名时，数据送到打印机，而助忆名在专用名段定义。若指定屏幕名，数据送到屏幕上，屏幕名要在屏幕节加以定义。

2.位置说明指定了打印头的位置，若不写此项，则把回车和换行送给打印机；对于每个显示项来说，若指定位置说明，则在传送数据之前定位光标的位置。若不用此项，则说明光标在当前的位置上。

3.若指定数据名或常字为输出项，则将数据名的内容或常字的值送到接收设备。

若规定ERASE并为一个原来的显示项编写位置说明，则从当前的光标位置到屏幕的末端清除该屏幕。对于下一个显示项的光标初始位置是由在ERASE显示项（若存在的话）中编写的位置说明来规定的，或是由留在原来的显示项光标的位置来规定的。若规定ERASE并且在DISPLAY语句中没有位置说明，则其作用无效。

这一格式适用于IBM PC COBOL。

4-6-3 MOVE(传送)语句 传送语句将一个数据从一个内存区传送到另一个内存区，并对传送的数据进行必要的转换和编辑。

一、格式：

$$\underline{\text{MOVE}}\ \left\{\begin{matrix}\text{常字}\\ \text{数据名}-1\end{matrix}\right\}\ \underline{\text{TO}}\ \ \text{数据名}-2[,\ \text{数据名}-3]\cdots$$

二、说明：

由数据名-1表示的数据或规定的文字传送到由数据名-2指出的区域。还可以指定另外的接收项(数据名-3，数据名-4等)。

当接收项是一个组合项时，在传送字符时，不考虑这个组合的层次结构，也不进行编辑。

如果数据名-2使用了下标或索引，则应在数据传送到接收域之前求出该下标或索引的值。对其它的接收域（如数据名-3等）也是一样。

对传送的项应考虑下面一些问题：

1.数字型数据（外部的或内部的十进制数、二进制数、数字文字、或零）或字母数字型数据传送给数字数据或报表型数据。

(1) 按小数点位置把发送域（源域）和接收域对齐，若发送域的长度超过接收域，则产生截断。若发送域长度小于接收域，其不足部分用零字符补足。如果发送域是字母数字型数据，则按无符号整数那样处理，它的长度不能超过31个字符。

(2) 发送域（源域）的类型和接收域类型不同时，则转换成接收域的类型。字母数字型的源项按一个USAGE IS DISPLAY子句的无符号整数处理。

(3) 送到接收域的各项都要按照接收项给定的类型进行编辑，例如在相应的位置上加零，插入货币符号等，以及进行小数点的对齐。

(4) 在 PICTURE 描述中已说明为字母型或字母数字编辑的项是不能传送给一个数字项或报表项的。也不能把任一种数字数据传送给一个字母型数据，虽然数字整数数据和数字报表数据可以传送给字母数字数据(编辑的或不编辑的)，但是在这种情况下即使给出了"SIGN IS SEPARATE"运算符号仍然是不传送的。

2.非数字型的源和目的：

(1) 在接收域中字符的存放是按从左到右次序进行的，除非用了 JUSTIFIED RIGHT子句。

(2) 如果传送给接收项的数据没有放满接收域(即有剩余位置)，则这些位置上用空格填满。

(3) 如果发送域的长度比接收域长度长，则在接收域放满后就不再传送了。

(4) 当发送域和接收域出现矛盾并出现复盖字段时，有时结果是难以预料的。

(5) 带有USAGE IS INDEX的数据项不能作为一个运算量在MOVE语句中出现。

4-7 控制转移语句

过程部分中的节、段和操作语句，通常是按出现的顺序被执行。但也可以按实际处理的要求，使用过程转移语句改变程序正常的执行顺序。

4-7-1 GO TO(转移或转向)语句 GO TO 语句使程序跳开当前执行的语句序列，直接转移到另一过程段（或节）去执行。

一、格式：

GO TO 〔过程名-1〔〔过程名-2〕… DEPENDING ON 数据名〕〕

二、说明：

1.用GO TO过程名这一简单形式，可以改变走向，转到所指定的段或节去。如果GO语句没有过程名，则GO语句必须是一个段中唯一的语句，并且执行它之前必须用ALTER 语句对它进行修改。

ALTER语句的格式为：

ALTER 段 TO 〔PROCEED TO〕 过程名

这里"段"（它是第一个操作数)必须是COBOL的段，该段仅由一个简单的GO TO语句组成。ALTER语句的作用是通过过程名取代那个GO TO语句的原先的操作数。

2.更为一般的形式是，给出N个过程名，当数据名的值为 1 到N时，选择转移到相应的过程名去。如果数据名的值在 1 到N的范围之外，则不执行控制转移，并按正常的顺序执行下去。数据名必须是一个数字型的初等项，并且应为整数。

如果在一个命令语句序列中使用GO语句(不带DEPENDING)，则它必须是该序列中的最后一个语句。

4-7-2 PERFORM(执行)语句 使用PERFORM语句，可跳开当前执行的语句序列，转到指定的过程体去，连续执行若干次后，再返回到PERFORM 语句的下一条语句。它的两种常用格式为：

一、格式I：

PERFORM 范围 [{整数 / 数据名} TIMES]

在格式I 中，整数或其值为整数的数据名决定指定的范围（range）应执行几次。如果不写"TIMES"短句，只执行一次。PERFORM执行完后，就接着执行PERFORM下面一个语句。如果数据名＜0，则不执行该范围。

二、格式II：

PERFORM 范围 VARYING {索引名 / 数据名} FROM 参量-1 BY 参量-2 UNTIL 条件。

其中范围是：过程名-1 [THROUGH / THRU 过程名-2]

过程名-1可以是一段名或一节名。

如果指定的只是一个段名，则在执行完该段的最后一个语句后返回。如果指定的只是一个节名，则在执行完该节中最后一段的最后一个句子后返回。如果指定了范围，则只有在适当地执行完一个段或一个节的最后一个句子后才返回。只有在执行完PERFORM 指出的"过程"后的返回才是正常的返回。在其它一些情况下控制将跳过该"过程"，即跳过"返回点"。

一般运算量（参量-1和参量-2）可以是一个数值常数，索引名，或数据名。实际上，它们通常是整数，或者其内容为整数的数据名。它们好比是数据名的下标一样用来说明数据名的变化范围。

在格式II中，要执行"范围"（range）的次数是一个变化的数值，从数据名的初始值等于参量-1开始，每次按步长值（参量-2）增量，直到满足给定的条件为止。此后，就转为执行PERFORM后的下一个语句。

注意：

在格式II的 PERFORM 语句中，先进行条件的求值和判断，然后再决定是否执行该范围(range)。一般来说，如果该条件在开始时就不满足，则一次也不执行该范围。

在运行时，如果几个同时起作用的PERFORM 语句规定的范围具有同一个终点的话，则认为是不合法的。

格式II有一种扩展的方案，允许同时变化2个或3个项。

4-7-3 EXIT(出口)语句 EXIT语句用来作为一个过程的出口。其格式为：

段名．EXIT．

源程序中EXIT语句只在段名后写上 "EXIT"一个字即可。出口段提供了一个出口，如果需要跳过一段或一节中某些部分的话，则可以从EXIT语句前面的某个语句控制转移到这个出口。

4-7-4 IF(条件)语句 IF语句允许在给定的条件为"真"时执行一系列的过程类的语句。如果条件不成立（为"假"时）则执行另外选择的语句。

一、格式：

IF 条件 {NEXT SENTENCE / 语句序列-1} [ELSE {语句序列-2 / NEXT SENTENCE}].

二、说明：

1.条件

“条件”可以是一个简单的条件或是一个复合的条件。简单的条件有四种：关系的、类别的、条件名的和符号的条件测试。简单的关系条件格式如下：

〈运算量-1〉 〈关系符〉 〈运算量-2〉

这里的运算量可以是数据名、常字或赋形常数。

(1) 最简单的“简单关系符”有三种基本形式，以等于 (egual to)、小于 (less than) 或大于 (greater than) 表示(即=，< 或 >)。

另外一种简单的关系符，是在上面三个关系符前面加上保留字NOT。这样，就共有六种简单的关系符，它们是：

关系符	意义
=	等于
<	小于
>	大于
NOT=	不等于
NOT<	大于或等于
NOT>	小于或等于

关系算符 >、<、= 分别与 COBOL 中的保留字 GREATER THAN、LESS THAN、EQUAL TO等价。

两个数据项的比较分为数值项的比较和非数值项的比较。

数值项的比较是在它们的小数点位置对准以后才进行比较的。它的结果也是以数值表示的。任何负数都小于零，而零又小于任何的正数。在一个比较式中可以出现索引名或索引项。比较两个数值型运算量可以不必考虑它们在USAGE子句中所指定的格式，也不必考虑它们的长度。

字符的比较是按ASCII代码进行的，字母A－Z的代码是按增加的顺序变化的，而数字的代码是小于字母的。组合项在比较时是简单地按字符来对待的。如果一个运算量是数值型，而另一个是非数值型的，那么该数值项必须是整数，而且它必须用隐含的或明显的“USAGE IS DISPLAY”形式。

(2) 类别测试条件

类别测试的格式为：

数据名 IS [NOT] {NUMERIC / ALPHABETIC}

这个条件检查该数据项的内容，以确定它是数字的还是字母的。当它对数字测试时，若规定SIGN IS SEPARATE，则所有字符必须是带运算符的适当的数字表达式 (0…9)。当对字母测试时，必须是仅出现大写字母 (A…Z) 或空白字符。NUMERIC测试仅适用于组合项、十进制数项或字符项(不能有字母型PICTURE)。ALPHABETIC 测试只能用于组合项或字符项(都是一种PICTURE格式)。

(3) 符号测试。其格式为：

数据名 IS [NOT] { POSITIVE / NEGATIVE / ZERO }

这个测试相当于把数据名和零比较，以便确定此条件的真值（即判断此数据名的内容是正数？零？还是负数？）。

（4）条件名测试

在一个条件名测试中，测试一个条件变量的值，以确定它和与它相关联的条件名所确定的值之中的一个是否相等。条件名测试可以用下面的语句（格式）表示：

条件名

这里，条件名必须由数据部分层号88来确定。

用保留字AND或OR可以构成复合的条件，如下：

由AND 联接几个单独的关系式来确定复合的条件，只有所有的单独关系式的值为真时，这个复合的关系条件的值为真。

由OR联接单独的关系式来确定复合条件，只要有任何一个单独的关系式的值为真，则此复合的条件的值为真。

2.语句序列-1仅在给定的条件为真时执行。

语句序列-2(ELSE部分) 仅在给定的条件为假时执行。

不论条件是否成立，在执行完相应的语句序列之后应当执行下一个句子，除非在所执行的语句中包括GO TO语句，或由于执行一个PERFORM语句而使程序流程改变。

3.IF语句的嵌套

当在一个句子中出现一个以上的IF时，则存在“嵌套的IF”。

4-7-5 STOP(停止)语句 STOP语句用来终止或暂停程序的运行。

一、格式：

STOP { RUN / 常字 }

二、说明：

1.“STOP RUN”是程序的逻辑终点，它的执行将结束程序的运行，使控制返回到操作系统。

2.“STOP 常字” 执行时会在控制台上显示出指定的常字，并使程序暂停。在继续运行程序之前，操作员可根据需要作相应的操作，使程序从STOP语句的下一个语句开始继续执行。

4-8 字符处理语句

COBOL中有三条专门用来对字符进行计数、替换、链接和分解等操作的语句，统称为字符处理语句。

4-8-1 INSPECT(检测)语句 INSPECT 语句能够帮助程序员检测一个字符串项。不同的选择可以具有下面不同的作用：

1.累计一个指定的字符出现的次数。

2.用一个指定的字符去代替另一个指定的字符。

3.用要求其它指定的字符出现的次数来限制上述作用。

一、格式：INSPECT 数据名-1〔TALLYING子句〕〔REPLACING子句〕

此处TALLYING子句有如下格式：

$$\text{TALLYING}\ \text{数据名-2}\ \text{FOR}\left\{\left\{\begin{matrix}\text{ALL}\\\text{LEADING}\end{matrix}\right\}\ \text{运算量-1}\right\}$$

$$\left[\left\{\begin{matrix}\text{BEFORE}\\\text{AFTER}\end{matrix}\right\}\text{INITIAL}\ \ \text{运算量-2}\right]$$

在数据名-1的字符串中按条件查找运算量-1，匹配数存入数据名-2

REPLACING子句的格式如下：

$$\text{REPLACING}\left\{\begin{matrix}\left\{\begin{matrix}\text{ALL}\\\text{LEADING}\\\text{FIRST}\end{matrix}\right\}\ \text{运算量-3}\\\text{CHARACTERS}\end{matrix}\right\}\ \text{BY运算量-4}$$

$$\left[\left\{\begin{matrix}\text{BEFORE}\\\text{AFTER}\end{matrix}\right\}\ \ \text{INITIAL}\ \text{运算量-5}\right]$$

二、说明：

1.因为数据名-1 是 INSPECT 的检测对象，其内容被看作一个字符串来处 理，它 应 由 USAGE IS DISPLAY来描述(隐含地或明显地)，(若不指定USAGE子句，则采用 DISPLAY 语句) 它不能用 USAGE IS INDEX来描述。数据名-2必须是一个数字数据项。作为计数单元，用来存放计数子句产生的计数结果。

2.在上面的格式中，运算量-n可以是一个以引号括起来的长度为 1 的常字，或代表一个简单字符的赋形常数，或一个其长度为 1 的数据名的项。

3.不能同时省略TALLYING子句和REPLACING子句。如果同时用这二个子句，则TALLYING子句必须在前面。

4.TALLYING子句按该子句给出的要求把数据名-1中的字符从左到右地一个一个地和给定的条件比较，比较后每找到一个配对就给数据名-2累计加1。BEFORE和AFTER 短语用来限制检测范围。若有BEFORE INITIAL运算量-2，只有在数据名-1中发现与运算量-2相同的一个字符时，累计过程才停止。若有AFTER INITIAL运算量-2，只有在数据名-1中发现与运算量-2相同的一个字符后，累计过程才开始。

5. REPLACING 子句的作用是按这个子句所给的条件自左到右地取代字符。如果有 BEFORE INITIAL 运算量-5，则在发现数据名-1中一个字符与运算量 -5 相同时就不再进行取代了。如果用AFTER INITIAL运算量-5，则在发现数据名-1的一个字符与运算量 -5 相同之后开始取代工作。

6.对数据名-1的取代和累计决定于TALLYING和REPLACING 子句中对字符的处理，说明如下：

(1) “CHARACTERS” 隐含地指出在数据名-1中的每一个字符都要进行累计或取代。

(2) “ALL” 运算量意思是：在数据名-1中的和 “运算量” 所规定的字符相同的所 有 字符，都参与累计和取代。

(3)“LEADING运算量”表示：把数据名-1中的字符从最左的一个开始和“运算量”所确定的字符相比较，如果连续地有几个是和运算量的字符相同的，则这几个字符（例如连续有几个零）都参加累计和取代。

(4)“FIRST 运算量”表示：只是和运算量所确定的字符相同的第一个字符被取代(REPLACING)。(这个FIRST选择在TALLYING中不用。)

当同时用TALLYING和REPLACING子句时，这两个子句的作用和分别写两个INSPECT语句时一样，其中第一个INSPECT语句只包括一个TALLYING子句，第二个只包括一个 REPLACING子句。

当累计的值不断增加，最后在数据名-2中的结果是累计的值加上数据-2的初值。

4-8-2 STRING(串连或字符串链接)语句 STRING语句用于从若干个发送数据项中取出部分或全部内容，依次送到一个接收数据项中，使之链接起来。

一、格式：

```
STRING {{运算量-1}…DELIMITED BY {运算量-2 }}…
                                 {SIZE     }
       INTO 数据名-1 〔WITH POINTER 数据名-2〕
       〔ON OVERFLOW 强制语句〕
```

二、说明：

1.在这个格式中，运算量这一项应该是一个非数字文字，或一个字符的赋形常数，或数据名。

2.数据名-1标识一个接收字符串的数据项，它必须是不含编辑字符和右对齐（JUST）子句的字母数字型字符。

3.数据名-2标识的数据项是一个计数器，它必须是一个整型的基本数据项，这个数据项的值用来指示数据名-1中的指针位置(不断加1)。

如果没有 POINTER 短句，则逻辑指针的约定值是 1 。逻辑指针值指出接收域的起始位置，即数据从这里开始存放。在向接收域传送时，“DELIMITED BY”短语控制该源应到那里为止。如：

DELIMITED BY SIZE 传送整个源的域。(除非接收域放不下)

DELIMITED BY运算量-2：由运算量-2规定的字符串是一个检索模型(Search pattern)，如果在发送的字符中遇到和这个“检索模型”相同的字符序列，则停止这个运算量的传送，并自动转到执行下一个发送的运算量(如果有下一个运算量的话)。

如果逻辑指针（它是每存入一个字符就向数据名-1中加 1 而得的值）小于 1 或大于数据名-1的大小，则不再发生数据的传送，而去执行OVERFLOW短句（如果有此短句)中给出的强制语句序列。如果没有OVERFLOW短句，则转移到下一个可执行的语句。

4.在完成STRING语句时，接收项中的未被送入字符的位置上原有内容不变。

5.在STRING语句中，如果有一个POINTER短句，则在执行完此语句时，数据名-2结果值等于它的的起始值加上在执行STRING语句过程中向数据名-1所传送的字符总数。

4-8-3 UNSTRING（串分解)语句 UNSTRING 语句的作用是将发送项的数据进行分解，并把分解得到的子串分别传送到多个接收项中去。

一、格式：

UNSTRING 数据名-1

〔DELIMITED BY 〔ALL〕 运算量-1〔OR〔ALL〕运算量-2〕…〕

INTO {数据名-2 〔DELIMITER IN 数据名-3〕

〔COUNT IN 数据名-4〕}…

〔WITH POINTER 数据名-5〕

〔TALLYING IN 数据名-6〕

〔ON OVERFLOW 强制语句〕

二、说明：

1.数据名-1必须是一个组合项或字符串（字母数字型）项。如果用一个数据项来作任一个运算量i，则该运算量也必须是一个组合项或字符串项。

2.分界标志DELIMITERS用来分开在DELIMITER短句中输入的子域。一个连续的字符每次遇到和当前字符相同的一个非数字常字、或一个字符的赋形常数，或名为运算量i的数据项的值，则字符集结就停止，并把这批字符送到INTO子句中所规定的下一个接收域中。当使用ALL短语时，如果数据名-1中的运算量i连续出现多次，也按一次处理。

当有两个或多个分界标志时，它的意思相当于是一个“OR”（或）条件存在。每一个分界标志按UNSTRING语句所规定的次序和发送域进行比较。

3.接收域数据名-2可以是以下任一种形式：

（1）一个非编辑的字母项；

（2）一个字符串（字母数字）项；

（3）一个组合项；

（4）一个外部的十进制项(数字型的，用usage DISPLAY)，在它的PICTURE描述中不能包括P字符。

如果发现有二个连续的分界标志，则当前的接收区中就以空格或填入零（依其类型决定是空格还是零）。如果在UNSTRING语句中有一个“DELIMTED BY”短语，在“INTO”子句中又包括“DELIMITER IN”短语(它是跟在任何一个接收项数据名-2后面的)，在这情况下，使传送给数据名-2的数据产生分界的字符（一个或多个字符）存放在数据名-3中，它应该是一个字母数字项。

4.如果有一个“COUNT IN”短语，则传送给数据名-2中的字符总数就送到数据名-4中去，数据名-4必须是一个整型的基本数值项(即初等数值项)。

5.如果有一个“POINTER”短语，则数据名-5必须是一个整型的数值项，它的初始值变成逻辑指针的初始值。如果无此短语，则认为逻辑指针的初始值等于1。

对数据名-1中的源字符（Source characters）的检查是从这个逻辑指针所指定的位置开始的，在UNSTRING语句完成时，逻辑指针的最终值放回到数据名-5中。

6.任何时候，如果逻辑指针的值小于1或超过数据名-1的范围，则发生“溢出”，并转而执行“ON OVERFLOW”子句中给出的强制语句序列（如果有“QN OVERFLOW”子句存在的话)。

当全部的接收域放不下源域传送的内容时也会出现溢出。

在对源域进行扫描过程中(即寻找相应的分界符的序列),不断累积字符(字符串的长度不断地改变),当源字符被分界标志辨别出来(即遇到相应的分界标志)时,或者虽然未遇到相应的分界标志但是已达到了当前接收项的范围(长度),则把这些累积起来的字符传送到当前接收域中,如同MOVE语句的功能一样。

如果有一个"TALLYING IN"短语,数据名-6必须是一个整型数值项。在做完UNSTRING语句后,接收域的个数加上数据名-6的初始值放在数据名-6中。

和数据名-1,5或6有关系的下标或索引只在UNSTRING语句开始时求其值一次。与运算量i或数据名-2,3,4有关系的下标值是在存取数据项之前立即求值的。

4-9 表处理语句

4-9-1 索引名和索引数据项 表必须先在数据部分定义,才能在过程部分引用。对表元素的引用有下标法和索引法两种。下标法是通过表名及放在表后面圆括号内的下标值来实现对表中任一元素的引用。

一、索引名

索引名不用通常的层号、名字和数据描述子句的方法来说明,而是在OCCURS子句的"INDEXED BY 索引名"部分隐含地出现。索引名必须是唯一的。其定义格式是:

OCCURS 整数 TIMES INDEXED BY 索引名

索引名的内容,即索引值,是与OCCURS子句中所定义表元素的重现序号相对应的,它表示所指定的元素的第一字节相对表的起始位置的位移量。在用索引名引用表元素之前,先要对索引名赋值,实现重现序号与索引值之间的转换,在使用索引名之前,必须用SET,SEARCH或PERFORM来给它置入初值。

二、索引数据项

索引数据项是由USAGE IS INDEX短句所定义的项。索引数据项用作索引值的暂存区,可以不经转换直接取用。索引数据项的描述中不能使用PICTURE、JUSTIFIED、BLANK WHEN ZERO和VALUE等子句。

当USAGE IS INDEX用于组项时,说明组项中的初等项都是索引数据项,但组项本身不是索引数据项。

一个索引名或索引数据项只能用下列方法来修改:

1.SET语句

2.SEARCH语句

3.与子程序中PROCEDURE HEADER USING表相应的CALL语句的USING表。

在下列情况中可以使用索引名或索引数据项:

1.在一个关系条件中

2.PERFORM VARYING语句中的变化项

3.在下标的位置中。

在这些情况中,它等价于处理一个二进制字的整数下标。

4-9-2 SET(设置)语句 为了进行表处理,可使用SET语句改变索引名、索引项或二进制的下标。其格式有二:

一、格式I:

$$\underline{\text{SET}}\ \left\{\begin{array}{l}\text{索引名-1}\\ \text{索引数据项-1}\\ \text{数据名-1}\end{array}\right\}\cdots\underline{\text{TO}}\ \left\{\begin{array}{l}\text{索引名-2}\\ \text{索引数据项-2}\\ \text{数据名-2}\\ \text{整数-2}\end{array}\right\}$$

格式I的作用是把“TO”后面的值（如整数-2）传送到紧跟 SET 之后的若干个接收域中。

二、格式II：

$$\underline{\text{SET}}\quad \text{索引名-3}\cdots\left\{\begin{array}{l}\underline{\text{UP}}\ \underline{\text{BY}}\\ \underline{\text{DOWN}}\ \underline{\text{BY}}\end{array}\right\}\left\{\begin{array}{l}\text{数据名-4}\\ \text{整数-4}\end{array}\right\}$$

格式II 的作用是：SET 后面的量（索引名-3 的值）减少（DOWN）或增加（UP）一个值。减少或增加的值由BY后面的名字或值来确定。

在这两种格式中，数据名只限于整数项。

4-9-3 SEARCH（检索）语句 在 COBOL 中，SEARCH语句专门用于表元素的检索，它有两种格式，分别对应于顺序检索和非顺序检索。

一、格式I：

1.格式：

$$\underline{\text{SEARCH}}\quad \text{表}\quad \left[\underline{\text{VARYING}}\left\{\begin{array}{l}\text{索引名}\\ \text{标识符}\end{array}\right\}\right]$$

$$[\text{AT}\quad \underline{\text{NED}}\quad \text{强制语句-1}]$$

$$\left\{\text{WHEN}\quad \text{条件-1}\ \left\{\begin{array}{l}\text{强制语句-2}\\ \underline{\text{NEXT SENTENCE}}\end{array}\right\}\right\}\cdots$$

2.说明：

(1)“表”是一个带有 OCCURS 子句的数据项的名字，而 这个 OCCURS 子句包括了一个INDEXED—BY 列表。因为 SEARCH 语句的性质能使与特定表有关的“索引名”自动变化。因此，“表”应写成不带下标或索引。

(2) VARYING有四种可能的用法：

① 没有VARYING 短语。则规定只使用这个表中的第一个索引名来查表。

② 有 VARYING 短 语，且索引名属于被检索的那个表，则用索引名来检索，检索时只对它进行调整，其它索引名的值不变。

③ VARYING后面是索引名，但它不属于 SEARCH 后面指出的那个表，而属于另外一个表。此时，使用前一个表中的第一个索引名来进行查表。当这个索引名在语句执行中增值时，VARYING后面的索引名也同时隐含地增值，并指向与它相关的那个表的下一个表元素。

④ 若 VARYING 短语中选择标识符，这时按 OCCURS 子 句 中定义的第一个索引名进行查表。

(3) 有关索引名按以下规定：

① 假设初值已由前面的某一语句（如SET）所设置了。

② 如果初始值超过了所用的 OCCURS 子句所指定的最大值，立即停止SEARCH操作；如果AT END 短语存在，则去执行强制语句序列-1。

③ 如果索引值在有效索引的范围内（1，2，…直到并且包括出现的最大号），则对每一个WHEN条件求值，直到有一个条件为真或所有条件都为假。如果有一个条件为真，则执行与它相应的强制语句，并且停止SEARCH操作。如果没有条件为真，则把索引加 1 并重复检

索过程，注意：索引的增量无论用于项或是索引名都必须根据上述VARYING状态的四种用法来选定。

如果这个表是另一个表的子表，则每一个索引名必须和整个表的每一维相对应，即表的每一维都应至少有一个自己的索引名。它们是通过 OCCURS 子句的 “INDEXED BY” 短语来规定的。只有 SEARCH 语句中的索引名是变化的（“VARYING” 的索引名或数据项和它一起变化）。为了检索一个二维的或三维的表，必须多次执行 SEARCH 语句，每次都要单独设一个索引名和定初值，也可以用 PERFORM VARYING 语句来实现。

二、格式II：

1.格式：SEARCH ALL 表 〔AT END 强制语句-1…〕

WHEN 条件 {强制语句-2… / NEXT SENTENCE}

2.说明：

(1) 这一格式用来检索已排序好的表，表的定义同格式I。

(2) 这里只允许有一个WHEN子句，关于条件有如下规则：

① 只可使用简单的关系条件或条件名，条件的主体必须是用与“表”相联系的第一个索引名构成的索引下标数据名（如果多次使用 OCCURS 子句，则每一维都应满足这个规则）。而且，在一个条件中的每一个主体的数据名（或与条件名相联系的数据名）必须在该表的 KEY 子句中加以说明。KEY子句是OCCURS子句的一个附属句，其格式为：

{ASCENDING / DESCENDING} KEY IS 数据名…

这里，数据名是在数据描述体中定义的名字或者是从属的数据名中的一个。如果给出多于一个的数据名，那么它们全部必须是附属于这个组合项的各项的名字。

当选用 “ASCENDING KEY IS 数据名”时，表示数据名的值按由小到大来排列表元素；若选用 “DESCENDING KEY IS 数据名”，表示根据数据名的值按递减次序排列表元素。可以使用多个KEY短语。

② 在简单的关系条件中，只允许用 “等于” 的测试（即用关系符“=”或者“IS EQUAL TO”）。

③ 任何一个条件名变量（即88级的项），必须定义为只有一个单值。

④ 条件可以用逻辑连接符 AND 来组合，但不能用 OR来组合。

⑤ 在简单的关系条件中，客体（即等号右边的内容）可以是常字或标识符。这些标识符不能被表中的 KEY 子句来引用，也不能被与表相应的第一个索引名所检索。（“标识符”表示包括任何形式的定义符下标或索引的数据名）。

如果不遵守以上规则，就会得到不可预料的结果。如果表数据不按照 KEY 子句说明的方式安排，或者如果 WHEN —条件中引用的关键字不能完全识别一个唯一的表元素，也会产生不可预料的结果。

(3) 在格式II的 SEARCH 中，可以进行不连续的（非顺序）检索操作，这是依靠数据规定的顺序来完成的。表的索引名的初始设置被忽略，在检索期间，它的设置会自动改变，且总是在检索数据的最大值范围内。

(4) 如果条件 (WHEN) 不能满足任何一个有效的索引值，且如果 AT END 子句存在，

则执行“强制语句-1”；而如果AT END 子句不存在，则执行下一个句子。

如果在一个 WHEN 条件中，所有的简单条件都得到满足，则作为结果的索引值指向已经找到的那个表元素，并转向强制语句-2。否则，索引值的最后结果是不可预料的。

4-10 用于程序间通讯的语句

4-10-1 COPY 语句

一、格式：COPY 文本标记

二说明：

1.COPY 语句可出现在程序中的任何地方，并在程序中独占一个程序行。

2.COPY 语句的作用是将在磁盘中保存的以文本标记所标识的标准COBOL编码实体，插到源程序中 COPY 语句出现的地方。

4-10-2 CALL（调用）语句

一、格式： CALL 程序名〔USING 数据名…〕

二、说明：

1.程序名是指在被调用程序标识部分的 PROGRAM-ID 段 中所定义的程序名。

2.USING 表中的数据名通过与子程序的相应的地址发生联系从而使子程序能使 用这些数据。连接节中的项可以分配到相应的地址，这些项也在子程序的 USING 表 中加以说明。因此，由 CALL 语句指定的数据名的个数应与过程部分的 USING 表指定的数据名个数相等。

3.调用程序与被调用程序的 USING 表的对应是按它们的先后次序（位 置）来进行的，而不是按它们的名字是否相同。

4-10-3 EXIT PROGRAM语句

一、格式： EXIT PROGRAM.

二、说明：

这一语句在被调用的子程序中使用，表示子程序执行到此为止，而使控制从这里返回到主调程序中调用该子程序的 CALL 语句的下一个语句。

4-10-4 被调子程序的过程部分标题 一个可以被调用的子程序的过程部分的开头可表示为：

PROCEDURE DIVISION 〔USING 数据名…〕

其中的 USING 短语定义一个数据名表，每个数据名标识的数据项必须在本程序的连接节中作为77层或01层数据项被描述，而在本程序的过程部分中使用。在 USING 短语中，一个数据名不能出现二次或二次以上。

本章参考书目

〔1〕《微型计算机 COBOL 程序设计》 何克抗 裴广生编 北京师范大学出版社

〔2〕《CROMEMCO 微型计算机软件资料汇编》（二）清华大学计算中心译编 清华大学出版社

〔3〕《TM个人计算机 COBOL 编译程序》 福建省福州市开大电脑服务公司

〔4〕《COBOL 程序设计》（讲义）南京大学 计算机科学系软件教研室编

〔5〕《COBOL Compiler by Microsoft》

附　录

COBCL　保留字

正号（+）指出是 IBM-PC COBOL 互交式屏幕，调试扩展和缩合十进制格式所需的附加字。

实方块（■）指出是标准COBOL中的保留字，在IBM和CROMEMCO COBOL 不保留它，为了兼容，应该在程序中尽量避免使用这些字作为文件名，数据名和变量。

*指出 CROMEMCO-COBOL 不用的保留字

** 指出 CROMEMCO-COBOL 附加的保留字

ACCEPT	ASCENDING
ACCESS	+ASCII
ADD	ASSIGN
ADVANCING	AT
AFTER	AUTHOR
ALL	AUTO *
ALPHABETIC	+AUTO-SKIP *
■ALSO	
ALTER	+BACKGROUND-COLOR *
■ALTERNATE	+BEEP
AND	BEFORE
ARE	BELL *
AREA(S)	BLANK
BLINK*	COLUMN *
BLOCK	COMMA
BOTTOM*	■COMMUNICATION
BY	COMP
	COMPUTATIONAL
CALL	COMPUTATIONAL—O *
■CANCEL	+COMPUTATIONAL—3
■CD	COMPUTE
■CF	COMP—O *
■CH	+COMP—3
CHAIN *	CONFIGURATION
CHAINING *	CONTAINS
CHARACTER(S)	■CONTROL(S)
■CLOCK—UNITS	COPY
CLOSE	■CORR (ESPONDING)
■COBOL	COUNT

■CODE
CODE–SET
+COL *
COLLATING
DATE–COMPILED
DATE–WRITTEN
DAY
■DE (TAIL)
DEBUGGING
■DEBUG—CONTENTS
■DEBUG—ITEM
■DEBUG–NAME
■DEBUG–SUB–1
■DEBUG–SUB–2
■DEBUG–SUB–3
DECIMAL–POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
■DESTINATION
■IDSABLE
ERROR
ESCAPE *
■ESI
■EVERY
EXCEPTION
+EXHIBIT
EXIT
EXTEND

FD
FILE
FILE–CONTROL
+FLLE–ID *
FILLER
■FINAL
FIRST

CURRENCY
CONSOLE **
DATA
DATE
+DISK
DISPLAY
DIVIDE
DIVISION
DOWN
■DUPLICATES
DYNAMIC

■EGI
ELSE *
■EMI
+EMPTY–CHECK *
■ENABLE
END
END–OF–PAGE *
■ENTER
ENVIRONMENT
EOP *
EQUAL
+ERASE *
+FULL *

■GENERATE
GIVING
GO
GREATER
■GROUP

■HEADING
+HIGHLIGHT *
HIGH–VALUE (S)

IDENTIFICATION
IF
IN
INDEX

FOOTING *
FOR
+FOREGROUND-COLOR *
FROM
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS
I—O
I—O—CONTROL

JUST (IFIED)
KEY

LABEL
LAST
LEADING
LEFT
+LEFT-JUSTIFY *
LENGTH
NAMED **
+NAMES *
NATIVE *
NEGATIVE
NEXT
NO
+NO-ECHO *
NOT
NUMBER *
NUMERIC *

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON

INDEXED
INDICATE
INITIAL
INITIATE
+LENGTH-CHECK *
LESS
LIMIT (S)
+LIN *
LINAGE *
LINAGE-COUNTER *
LINE (S)
LINE-COUNTER
LINKAGE
LOCK
LOW-VALUE (S)
MEMORY
MERGE *
MESSAGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY
ORGANIZATION
OUTPUT
OVERFLOW

PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC (TURE)
PLUS *
POINTER
POSITION
POSITIVE
+PRINTER
PROCEDURE (S)
PROCEED

OPEN
OPTIONAL
OR

QUEUE *
QUOTE

RANDOM
RD
READ
+READY
RECEIVE
RECORD (S)
REDEFINES
REEL
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT (S)
SEARCH
SECTION
SECURE *
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT *
SORT-MERGE *

PROGRAM
PROGRAM-ID
+PROMPT
REPORTING
+REQUIRED *
RERUN *
RESERVE
RESET
RETURN *
REVERSED
+REVERSE-VIDEO *
REWIND
REWRITE
RF
RH
RIGHT
+RIGHT-JUSTIFY *
ROUNDED
RUN

SAME
SCREEN *
SD
+SPACE-FILL *
SPECIAL-NAMES
STANDARD
STANDARD-1
START
STATUS
STOP
STRING
SUB-QUEUE-1, 2, 3
SUBTRACT
SUM
SUPPRESS
+SWITCH-1 *
+SWITCH-2 *
+SWITCH-3 *
+SWITCH-4 *
+SWITCH-5 *

■SOURCE
SOURCE – COMPUTER
SPACE (S)
■SYMBOLIC
SYNC (HRONIZED)

■TABLE
TALLYING
■TAPE
■TERMINAL
■TERMINATE
■TEXT
THAN
THROUGH
THRU
TIME
TIMES
TO
TOP *
+TRACE
TRAILING
+TRAILING – SIGN *
■TYPE
WRITE
ZERO ((E) S)
+ZERO – FILL *

+SWITCH – 6 *
+SWITCH – 7 *
+SWITCH – 8 *

+UNDERLINE *
■UNTI
UNSTRING
UNTIL
UP
+UPDATE *
UPON
USAGE
USE
+USER *
USING

VALUE (S)
VARYING

WHEN
WITH
WORDS
WORKING – STORAGE

第三篇　微型计算机的使用

如第二篇所述，高级语言的出现使计算机易于为更多的人使用，这是因为高级语言是独立于计算机系统的，在使用上存在许多共性。例如，用各种高级语言编写的程序（称为源程序），都需要通过“翻译”，成为相应的机器语言程序，才能执行。这种“翻译”工作是由各计算机系统完成的。由于各计算机系统对同一种语言的“翻译”能力存在差异，同一种语言在不同计算机系统上，确切地说在不同的操作系统下，功能也略有差异。其次各系统有各自的系统管理方法。因此，各种语言即使发展语言程序的过程一致，所使用的命令或命令格式也不尽相同。本篇所述的语言使用，实际上就是讲述在各种操作系统支持下，建立、运行及保存源程序的一般方法及其所使用的各种命令，同时也介绍了相应的操作系统命令。

通常“翻译”有两种方式：解释方式和编译方式。大部分微机上使用 的 BASIC 语言都属于解释方式（CBASIC 除外），它的程序发展过程一般是在系统启动后：

1.建立和运行源程序；

2.必要时修改源程序；

3.源程序的保存与调用。

对于使用编译方式的高级语言（如本书介绍的 FORTRAN，PASCAL，COBOL 语言），也都有大致相同的发展过程，即在系统启动以后：

1.用编辑系统建立源程序；

2.用编译程序对源程序进行编译，形成代码组成的目标文件；

3.连接目标文件，形成可执行的机器码文件；并且运行它；

4.必要时对源程序进行编辑（修改）。

本篇在叙述上，采用了上述发展程序的过程。

最后提醒注意的是，即使同一种机型，同一种语言，也会因版本的不断更新，在操作上产生差异。因此，这里所叙述的，如与实际过程(如屏幕显示)不同，是允许的，且以实际为准。

下面附上APPLE机、Cromemco 和 IBM PC 机的键盘示意图，供读者在阅读本篇时参考。

图3-1　APPLE机的键盘

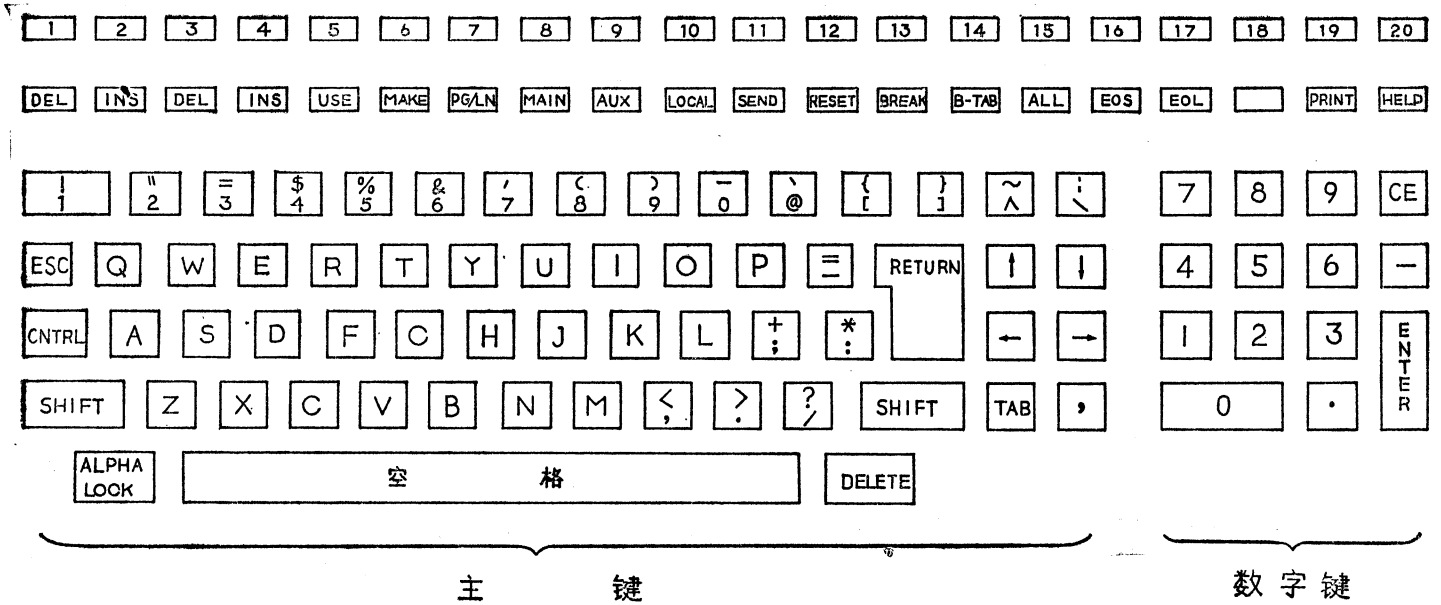


图3-2 Cromemco 机的键盘

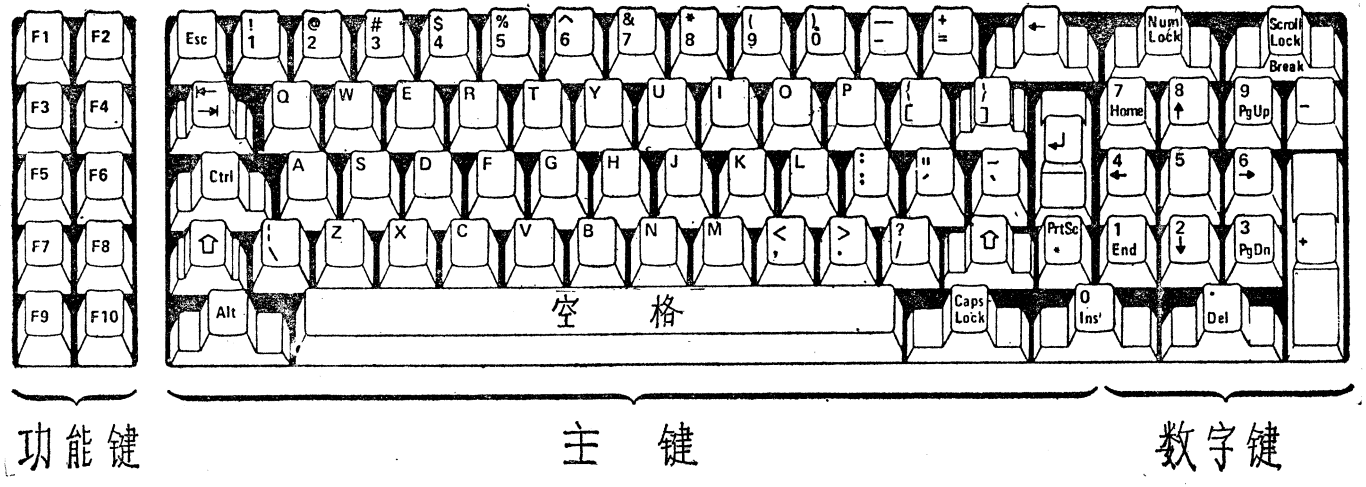

F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
Esc
Ctrl
Alt
空 格
Caps Lock
Num Lock
Scroll Lock
Break
PrtSc
Home
PgUp
End
PgDn
Ins
Del
功能键
主 键
数字键

图3-3 IBM 机的键盘

第一章　BASIC 语言在微机上的使用

BASIC语言在微型机上，一般有两种工作方式：立即方式（命令方式）和程序方式。

在立即方式中，BASIC的语句前不加行号，只要在输入完毕按 RETURN 键，即可立即执行，结果马上显示在显示器上或存入内存备用，但命令本身在执行后就消失了。此种工作方式常用来检验错误或把BASIC命令当做计算器来使用。例如：

```
PRINT 3+4

 7
```

程序方式则是先输入程序，输入的程序存放在内存中，打入 RUN 命令，就可以执行内存中程序。

下面以MBASIC 为例，说明两种工作方式的区别。（注意：OK是提示符）

立即方式：

```
OK
T=SQR(2*5/9.81)                          ; 立即命令
OK
PRINT "T=";T                             ; 立即命令
T=1.00964                                ; 显示结果
OK
```

程序方式：

```
OK
110 H=5
120 G=9.81
130 T=SQR(2*H/G)
140 PRINT "T=";T
150 END
OK
RUN                                      ; 运行命令

T=1.00964                                ; 显示结果
```

1-1　APPLE机DOS3.3操作系统下的使用

DOS3.3系统支持两种BASIC，即APPLESOFT和INTBASIC。目前在国内使用的 APPLE 机种繁多，功能也不尽相同。这里我们以一般带固化的APPLESOFT解释程序的APPLE 机为例，说明其用法。一般应具有：

显示器一台（彩色或单色）

磁盘驱动器两台(或一台)

其中：1*驱动器—主驱动器，代号D1

2*驱动器—辅驱动器，代号D2

打印机一台

至少48K RAM

1-1-1 系统启动和关闭

一、开机准备

按一般习惯，驱动器接在6*槽口的驱动器板上，打印机接上1*槽口的打印机板上。

主磁盘一块，即包括DOS引导程序在内的系统文件的磁盘。

用户盘一块。如果是新盘或旧盘重用，则在开机后首先进行初始化工作。简单的初始化操作是先用LOAD命令将主盘上HELLO程序调入内存，然后将待初始化的磁盘插2*驱动器，打入下述命令行即可。

```
INIT HELLO,D2
```

二、开机

把主磁盘放入1*驱动器，用户盘放入2*驱动器，关好小门，接通显示器电源、主机电源。随着驱动器指示亮，且有噼啪的正常声音，大约10秒左右将显示出类似于下面的有关 DOS3.3 的信息：

```
DOS VERSION 3.3
APPLE II STANDARD
]
```

符号“]”是APPLESOFT 的提示符，它的出现表明系统引导或说冷启动完成，用户可以使用BASIC执行各项任务。

DOS 3.3磁盘操作系统支持两种BASIC，整型BASIC–INTBASIC和浮点BASIC–APPLESOFT。前者适用于整数BASIC编写的程序和游戏程序，它的主要特点是比APPLESOFT运行速度快。要想进入整数BASIC，打入命令：

```
INT
```

显示器出现提示符“>”。如果想返回 APPLESOFT，则打入命令：

```
FP
```

其转换过程如下：

```
] INT

>

> FP

]
```

三、关机

关机前，一般先要从磁盘驱动器拿出磁盘（以免破坏磁盘中的信息），再依次关闭主机和显示器电源。

1-1-2 建立和运行 BASIC 程序

一、建立源程序——NEW命令

建立新程序前，首先打入NEW 命令，以清除内存，然后按BASIC 语言程序格式逐行打

入程序。例如：

```
] NEW

] 10 PI = 3.141592
] 20 INPUT "R = "; R
] 30 S = PI * R ∧ 2
] 40 ? "S = "; S
] 50 END
```

在打入程序过程中，如果发现打错了字符，但还未按 RETURN 键，可以按 ← 键返回错误位置，打入正确的字符，用⊡键退回后，再继续输入新的内容。如果按了 RETURN 键后，才发现打错了字符，可以重新打入有错误的行号及正确的内容，或使用编辑功能键进行修改。（参阅修改BASIC程序一节。）

二　程序的显示——LIST命令

打入：

LIST

命令，可看到系统已接收并经过整理的程序（可能与你打入的形式不同），供检查修改。例如：

```
]LIST

10 PI = 3.141592
20 INPUT "R = "; R
30 S = PI * R ∧ 2
40 PRINT "S = " ; S
50 END
```

如果程序较大，可用下述LIST命令格式分段显示。

(1) LIST n_1—n_2

列出n_1～n_2的各程序行。

(2) LIST —n_2

列出从开始到n_2的各程序行。

(3) LIST n_1—

列出从n_1开始到结束的各程序行。

(4) LIST n

列出行号为n的程序行。

三、运行源程序—RUN命令

打入命令：

RUN

可运行内存中的程序，上述程序会有如下结果：

] RUN

R= 2

S=12.566368

也可以使用“GOTO行号n”的立即命令，使程序从指定行号开始运行。

如果程序已存入磁盘，可使用如下的命令格式运行：

RUN 程序名(或文件名)

(有关文件请看1-1-7内容)

中断程序的运行（例如造成死循环),可打CTRL-C。(CTRL-C即压下 CTRL 键后再按 C 键。下同)

暂停程序的运行，可打CTRL-S。再按任一键，程序可继续执行。

1-1-3 修改BASIC 程序—编辑功能键 一个程序常常会发生错误，有的在运行前被检查出来，有的在运行时由系统检查到，并打出错误信息（错误信息请看1-1-8节错误信息)。

程序的修改除用前述的简单方法外，APPLE 有它特有的编辑功能，供程序修改用。

一、APPLE 的编辑功能

(1) 可用来修改程序，如对程序行进行改字，删字或插入新内容。

(2) 可用来复制程序行或命令行，减少输入的错误，节省时间。

二、编辑功能键

要使用APPLE的编辑功能，必须先熟悉APPLE机键盘上的四个特殊键的功能。它们是：ESC 键；REPT 键；← 和 → 键。

1. ESC

ESC 总是和其它键一同使用，完成纯光标移动或清除屏幕等功能。

操作顺序是：按下 ESC 键，使系统脱离当前状态，进入 ESCAPE 工作方式。再按一下其它指定的键，完成一次纯光标的移动。

例如再按一下A(称为ESCAPE A)。光标向右移动一格。

使用 ESC 和其它需要的键，光标可以移动到屏幕中的任意位置，而不影响现行的程序行和内存。

ESC 键用于纯光标移动有：

ESCAPE A 光标右移一位

ESCAPE B 光标左移一位

ESCAPE C 光标下移一位

ESCAPE D 光标上移一位

以上四种功能作用一次即退出ESCAPE方式。如要多次移动光标，则需多次进入ESCAPE

方式。

ESCAPE K 光标右移一位
ESCAPE J 光标左移一位
ESCAPE M 光标下移一位
ESCAPE I 光标上移一位

以上四种功能，光标移动后并不退出ESCAFE方式，只要连续按动I,J,M,K任意一键，光标可按要求移动，直到按任一个非ESCAPE功能键（如可再按一次 ESC 键),才能退出ESCAPE方式，结束纯光标的移动。

除移动光标功能外，还有：

ESCAPE @ 清除屏幕画面
ESCAPE E 清除从光标开始至行末的所有字符
ESCAPE F 清除光标右方及下方的所有字符

2. →

此键把光标右移过程中所扫过的字符和符号（包括空格),存入内存。这和从键盘打入这些字符和符号的结果一样。

使用 → 键时，不改变显示器(TV)上的显示内容。

3. ←

此键把光标左移，光标移动的同时。从现行程序行中抹去光标扫过的字符和符号。

使用 ← 键时，不改变显示器(TV)上的显示内容。

4. REPT

键盘上任一个键和 REPT 同时按下时。这个键的内容将被重复。直至释放 REPT 键。

三、使用编辑功能的简单例子：

例1. 改字符

当你输入下述程序后，打入 LIST ，并按 RETURN ，屏幕显示出：

```
] LIST

10  PRIMT "THIS IS A PREGRAM"
20  GOTO 10
]
```

注意:"PRIMT" 和 "PREGRAM" 是有意拼写错的。

■表示光标。

操作如下：

按 ESC ，再按2次 I ，将光标移到第10行。然后再按1次 J ，将光标移到本行

行号的第一个数字上。注意：把光标放在行号的第一个数字上是非常重要的。此时 TV 显示为：

```
] LIST
■0 PRIMT "THIS IS A PREGRAM"
20 GOTO 10
]
```

接着按一下 [ESC] 退出ESCAPE 方式。再按 [→] 6次，将光标移动到"PRIMT" 中的字母M上，按字母 [N] ，把M 改成N；再同时按 [→] 和 [REPT] 将光标移到"PREGRAM" 的字母E上。(如超过E可用 [←] 键使光标返回到字母 E上。)按字母 [O] ，把E改成 O。然后用 [→] 把光标移到本行末尾。最后按 [RETURN] 键，在内存中存入新的程序行。这时再打入LIST。屏幕显示出正确的程序：

```
] LIST
10 PRINT "THIS IS A PROGRAM"
20 GOTO 10
]■
```

例2. 插入新内容

在上一例中，如果想在第10句PRINT后插入TAB(10)，可进行如下操作：

首先打入LIST 10。屏幕显示出：

```
] LIST 10
10 PRINT "THIS IS A PROGRAM"
] ■
```

接着按 [ESC] 和 [I] 、 [J] 将光标移到此句行号的第一个数字上，再按 [ESC] 退出ESCAPE 方式，连续使用 [→] 将光标移到第一个引号上，TV 显示如下：

```
] LIST 10
10 PRINT ■THIS IS A PROGRAM"
]
```

此时再进入ESCAPE方式，即再按 [ESC] ，按 [I] 将光标上移到改动语句行上的空行。(注意此时引号并没有被存入内存。)按 [ESC] 退出ESCAPE方式，接着打TAB(10)；屏幕显示为：

```
] LIST 10
      TAB(10); ■
10 PRINT "THIS IS A PROGRAM"
```

]

接着按 ESC 键，按 M 和若干次 J 。使光标返回改动行的原插入位置，即第一个引号处。再按 ESC 退出ESCAPE方式，同时使用 → 和 REPT 将光标移动本行末尾。最后按 RETURN ，将改动后的程序行存入内存。打入 LIST 10，显示改动行如下：

```
] LIST 10
10 PRINT TAB(10); "THIS IS A PROGRAM"
] ■
```

1-1-4 保存源程序和调用、运行原有程序

一、保存源程序——SAVE命令

存放在内存中的程序会因关机、停电或其它错误的操作而丢失，因此常常需要把程序保存起来，常用的存贮介质是磁盘或磁带。存入磁盘或磁带的程序可称为程序文件，或称文件。SAVE命令可完成此功能，其命令格式如下：

SAVE 文件名，磁盘驱动器号

其中：

文件名是1至30个字符且以字母开头的字符串，中间不允许使用逗号或 CTRL-M。超过30个字符的多余字符被系统忽略掉。

磁盘驱动器号为 D1或 D2，如果是当前正使用的驱动器，可以省略驱动器号。

注意：如果在SAVE 之后没有文件名，则表示程序将要存入的是盒式磁带而不是磁盘。当然事先应该将盒式录音机接在录音机接口上，并做好录音准备。

SAVE 命令执行之后，并不影响内存中原有的程序。

二、显示磁盘目录——CATALOG 命令

打入CATALOG 命令可以显示磁盘上的文件目录清单，其格式为：

CATALOG，驱动器号(D1或D2)

如果是当前正使用的驱动器，可省略D1或D2。

文件目录包括文件类别、文件长度、文件名或文件是否被保护等信息。

例如，显示出：

```
*A  006  HELLO
```

*表示此文件被锁住(保护)，如不进行解锁，文件不能删除或重写。没有*表示文件没被锁住。

A表示文件类别是 APPLESOFT 源程序。它是由系统自动产生的。

006表示文件长度。文件长度以256字节为单位，所以本例为6×256字节。

HELLO是文件名

三、调用和运行原有程序——LOAD 命令

如果程序已保存，可用LOAD 命令调入内存，命令格式如下：

LOAD 文件名，驱动器号

如果使用盒式磁带，只要打入LOAD，但事先要打开盒式录音机，并做好放音准备。

注意：当使用LOAD 命令时，内存原有程序将要被清除。

程序调入内存后，可用LIST 命令去显示，或用RUN 命令运行。如需修改可按上节描述的方法进行修改。

1-1-5 文件操作

一、文件保护——LOCK(UNLOCK) 命令

对已存入磁盘的程序，可用LOCK 命令加以保护，其格式如下：

LOCK 文件名，驱动器号

经过保护的文件，可防止由于误操作而被修改或删除。如需修改或删除可打入命令行：

UNLOCK 文件名，驱动器号

解除保护。

二、文件更名——RENAME命令

RENAME命令可给程序重新命名，命令格式如下：

RENAME 老文件名，新文件名，驱动器号

注意：应是老文件名在先，新文件名在后。

三、文件的删除——DELETE命令

对于不再需要的文件可用DELETE命令去删除。命令格式如下：

DELETE 文件名，驱动器号

注意：如果文件被保护，一定要先解除保护。

四、文件的复制

简单的文件复制，可用LOAD命令，将程序调入内存，再用SAVE 命令重新以指定文件名存入磁盘。

复制大量文件，可使用系统盘中的实用程序FID。打入命令行：

BRUN FID

接下按提示操作，即可完成复制工作。

1-1-6 打印机使用 通常打印机接在1# 槽口板上，因此将输出传送至打印机上可打入命令：

PR#1

此后屏幕上显示的内容都被传送到打印机上。

打印机的行宽通常为40列与屏幕一致。而打印机的行宽最多可设置为130列。设置行宽的命令是：

POKE 1657，n

其中n是一行所要打印的最多字数。

注意：当行宽超过40时，仅打印机有输出，而屏幕上不再有显示。

一般打印机最大行宽为80列，所以要想设置大于80列的行宽，需要首先缩小字型，可使用下述命令：

PRINT CHR$(15)

而打入命令：

PRINT CHR$(18)

可恢复原字型。

打印机的输出控制命令还有以下几个：

换行—PRINT CHR $(10)

换页—PRINT CHR $(12)

1-1-7 DOS3.3操作系统命令和实用程序简介

一、DOS命令格式和说明

命令格式：

关键字 f〔,Vv〕〔,Ss〕〔,Dd〕

其中：

关键字指DOS命令中的任意一个命令，关键字必须大写。

f——文件名，最长为30个字符，但必须以字母开始，文件名中不能有逗号或CTRL-M，超过30个字符的多余字符将被忽略。

〔 〕——方括号中的项是可选择的（这些项有时被称为参数），如果不采用此项，则被认为是约定值。V，S，D是关键字，分别代表卷，槽和驱动器。

s——槽号。约定值是6。

v——磁盘卷号。如不指定，一般系统约定的值为254。

d——驱动器号（取值为1或2）。约定值是现行驱动器。

DOS命令中任何数字常量能用十六进制数写入，但十六进制数前必须冠有$符号。

二、内务命令：

命令格式	说明
INIT f〔, Vv〕〔, Ss〕〔, Dd〕	初始化磁盘命令
CATALOG〔 Vv〕〔, Ss〕〔, Dd〕	显示磁盘上文件目录
SAVE f〔, Vv〕〔, Ss〕〔, Dd〕	建立APPLESOFT和INTBASIC源程序文件
LOAD f〔, Vv〕〔, Ss〕〔, Dd〕	清除内存，装入新程序
RUN f〔, Vv〕〔, Ss〕〔, Dd〕	运行程序
RENAME f_1, f_2〔, Vv〕〔, Ss〕〔, Dd〕	查找文件f_1后改为f_2
DELETE f〔, Vv〕〔, Ss〕〔, Dd〕	删除以f命名的文件
LOCK f〔, Vv〕〔, Ss〕〔, Dd〕	建立文件的保护
UNLOCK f〔, Vv〕〔, Ss〕〔, Dd〕	解除文件的保护
VERIFY f〔, Vv〕〔, Ss〕〔, Dd〕	检查文件
MON 〔C〕〔, I〕〔, O〕	显示主机与磁盘之间传送的信息 C—显示磁盘命令 I—显示磁盘到主机的信息 O—显示主机到磁盘的信息
NOMON 〔C〕〔, I〕〔, O〕	使系统恢复到正常状态
MAXFILES n (n:1～16整数)	确定能同时使用的文件数目，省略后约定值为3。

三、选取命令：

命令	说明

PP	将系统转换到APPLESOFT。
INT	将系统转换到INTBASIC。
CTRL-D (CHR $(4))	DOS 命令标志，通知后面的字符是DOS命令。
PR # S	把输出送到S槽口所联接的外部设备，代替屏幕显示。
IN # S	从S 槽口控制的设备输入，代替键盘输入。
CHAIN f[, Ss][, Vv][, Dd]	用于INTBASIC 从磁盘调入和运行名为f的整数 BASIC 程序，但不清除任何变量。

四、机器语言文件命令

命令	说明
BSAVE f,Aa,Ll [,Ss][,Vv][,Dd]	建立名为f的机器码文件 a—内存中起始地址 l—字节数(文件长度)
BLOAD f [,Aa][,Ss][,Vv][,Dd]	把机器码文件装入 以a为起始地址的内存中
BRUN f[,Ss][,Vv][Dd]	运行机器码文件

五、DOS信息

LANGUAGE NOT AVAILABLE (无效语言)磁盘上没有 APPLESOFT 解释程序
RANGE ERROR (范围错) 命令参数太大
WRITE PROTECTED (写保护) 磁盘有写保护标签
END OF DATA (数据完) 读数据时超过了文本文件结尾
FILE NOT FOUND (没找到文件) 文件名拼写错或文件不在磁盘上
VOLUME MISMATCH (卷号配合错) 卷号参数错误
I/O ERROR (输入/输出错) 驱动器小门没关上或磁盘没初始化
DISK FULL (磁盘已满) 磁盘上文件太多
FILE LOCKED (文件被锁住) 企图改写或删除已锁住的文件
SYNTAX ERROR (句法错) 错误的文件名、参数或逗号
NO BUFFERS AVAILABLE (无有效的缓冲区) 打开的文件太多
FILE TYPE MISMATCH (文件类型配合错) 磁盘的文件类型与命令不符
PROGRAM TOO LARGE (程序太大) 有效内存不够
NOT DIRECT COMMAND (不是直接命令) 此命令必须在程序中使用

六、实用程序简介

随机来的系统盘中往往带有许多实用程序，它能帮助你更好地使用计算机。主要有：

类型	文件名	作用
A	HELLO	DOS引导程序
B	BOOT13	引导13扇区磁盘
B	CHAIN	链接程序

I	COLOR DEMO	INTBASIC 调整彩色程序
A	COLOR DEMOSOFT	APPLESOFT 调整彩色程序
I	COPY	按磁道复制磁盘，适用 INTBASIC
A	COPYA	按磁道复制磁盘，适用 APPLESOFT
B	COPY.OBJ0	COPY 调用的机器码子程序
B	FID	按文件复制文件
B	INTBASIC	INTBASIC解释程序
B	FPBASIC	浮点BASIC解释程序
A	RENUMBER	修改程序行号及合并程序
A	RENUMBER INSTRUCTIONS	RENUMBER 的说明文件
A	MAKE TEXT	产生正文文件
A	RETRIEVE TEXT	读正文文件
B	MASTER CREATE	SLAVE盘转变为MASTER盘
B	MUFFIN	13扇区转变为16 扇区文件

上述文件执行过程中都有详细的提示，请按提示去操作。

1-1-8 APPLE SOFT 错误信息

一、APPLE SOFT 错误信息格式：

(1) 立即执行方式？XX ERROR

(2) 程序执行方式？XX ERROR IN YY

其中：XX—错误名称；YY—发生错误的语句行号。

二、错误名称及其发生的原因：

CON'T CONTINUE（不能继续运行）

试图运行一个不存在的程序或试图在发生错误后继续执行程序。

DIVISION BY ZERO（被零除）除数不能为零，常发生于除数事先没被赋值。

ILLEGAL DIRECT（非法的直接命令）

不能将INPUT，DEF FN，GET或DATA语句做为立即执行语句。

ILLEGAL QUANTITY（非法数值）

主要包括：(1) 负的数组下标

(2) LOG的自变量X≤0

(3) SQR的自变量X＜0

(4) A∧B时，A为负数同时B为非整数

(5) 其它语句出现非法自变量。

NEXT WITHOUT FOR（有NEXT语句而无FOR 语句）NEXT 与FOR变量不相称。

OUT OF DATA（数据不够）程序所要读的数据多于程序中所给的数据，常发生于所有

DATA语句的数据被读完后，又要执行新的READ语句。

OUT OF MEMORY（内存不够）

程序太长、变量太多、循环或子程序嵌套太深、表达式括号重叠太多等等。

FORMULA TOO COMPLEX（公式太复杂）在一句中执行两个以上的IF句。

OVERFLOW（上溢出）计算结果太大

REDIM'D ARRAY（重复说明数组）同一数组，只需定义一次。

RETURN WITHOUT GOSUB（无转子返回）

遇到的RETURN，找不到调用的GOSUB 语句。

STRING TOO LONG（字符串太长）字符串长度不得超过255个字符

BAD SUBSCRIRT（错误下标）试图引用一个超出数组定义范围的数组元素。

SYNTAX ERROR（句法错误）表达式中漏掉括号，出现非法字符，错误标点等。

TYPE MISMATCH（类型配合错误）赋于数字型变量字符串，或反之。

UNDEF'D STATEMENT（未定义语句）

试图用GOTO，GOSUB或IF THEN指令转向某一不存在的行号。

UNDEF'D FUNCTION（未定义的函数）试图引用一个未定义的自定义函数。

1-2 APPLE-II机CP/M操作系统下的使用

在CP/M系统支持下的BASIC语言有 MBASIC 和 GBASIC 两种。MBASIC 是 Microsoft BASIC的CP/M版本。它包括了Applesoft标准扩充功能（高清晰度绘图功能除外），系统文件名为MBASIC. COM。GBASIC是在MBASIC的功能外，增加了高清晰度绘图能力，系统文件名为GBASIC. COM。

下面的叙述中除特别指明外，其它适用于两种BASIC语言。

系统配置如下：

显示器一台

磁盘驱动器两台（或一台）

其中：一号驱动器-主驱动器，代号A：

二号驱动器-辅驱动器，代号B：

打印机一台

主机上应安装z-80板和语言扩充板，z-80板一般插在4*槽口，语言板插在0*槽口。

1-2-1 系统的启动

一、开机准备

开机前应将驱动器接在6*槽口的驱动器板上，打印机接到1*槽口打印机板上。并检查主机上是否已配置了z-80板。

主磁盘（系统盘）一块，即至少包括系统引导程序，MBASIC和 GBASIC 等系统文件的磁盘。

用户盘一块。如果是新磁盘或旧盘重用，应在开机后进行初始化工作。请参照 2-4-1 有关初始化操作进行。

二、开机

将主磁盘放入一号驱动器，用户盘放入二号驱动器，关好驱动器小门，接通显示器、打印机和主机电源，屏幕显示出CP/M操作系统的信息：

APPLE II CP/M
56K VER, 2.2X
(C) 1980 Microsoft
A>

符号A>是CP/M操作系统系统命令级的提示符，它的出现表示系统引导（冷启动）成功。

提示符中的A表示开机后现行的驱动器是一号驱动器(或称A驱动器)。

以上启动过程称为冷启动，打入CTRL-C（即压下 CTRL 键后再按 C 键，下同）可进行系统的热启动。

三、装入MBASIC或GBASIC

由于MBASIC和GBASIC解释程序存放在磁盘上，使用前应将它们之一装入内存中。

在系统命令提示符A>出现后打入：

MBASIC(或GBASIC)

几秒钟后有关MBASIC（或GBASIC）的信息会显示在屏幕上。如下所示：

BASIC-80 REV 5.2
[APPLE CP/M VERSION]
COPYRIGHT(C) 1980 BY MICROSOFT
CREATEO: 12-NOV-80
26483 BYTES FREE
OK

"OK"是MBASIC和GBASIC的提示符。表示系统处于BASIC命令级，等待输入BASIC命令。

请读者注意，下面叙述的各命令都是在提示符"OK"下输入的，命令的结尾压 RETURN 键。在叙述中省略了提示符OK及 RETURN 键。

四、退出BASIC和关机

退出BASIC命令级，返回操作系统使用SYSTEM命令，其过程如下：

OK
SYSTEM

A>

现在可使用系统操作命令进行磁盘和文件的管理工作。

如果想关闭系统，最好先取出磁盘，再关闭电源，以保护磁盘信息不被破坏。

1-2-2 建立和运行BASIC程序

一、建立源程序—NEW，AUTO命令

建立新程序时，首先打入NEW命令，以清除内存中原来的程序和变量，然后按BASIC语言格式逐行打入程序。打入程序时，要注意在一个程序行中，行号、语句动词（关键字，保留字）和语句体之间要留有空格，否则会发生语法错误，除非它们之间有语法要求的引号、

分号、逗号、冒号等分隔符。

请看本章前言中的例题。

BASIC系统还提供自动产生行号的命令—AUTO命令。命令格式如下：

AUTO〔〈行号〉〔，〈增量〉〕〕

行号为起编行号；增量为相邻行号间的间隔。若省略行号或增量，其约定值都为10。例如：

命令格式	产生的行号序列
AUTO	10，20，30，……
AUTO 100	100，110，120，……
AUTO 50，20	50，70，90，……

打入CTRL-C可终止AUTO命令的执行。

输入程序时发现打错了字符，当还没按 RETURN 键时，可以用 ← 键或 CTRL-H，向左移动光标，移动的同时删去扫过的字符。错误字符删除后，可继续输入新的字符。如果已按了 RETURN 键，可以在退出AUTO命令后，重新打入有错误的行号及正确的内容，或使用EDIT命令进行修改。（参阅修改BASIC程序一节）

二、显示程序——LIST命令

程序输入后，可用LIST命令列出全部或部分程序，供检查修改。命令格式如下：

命令格式	功能
LIST	列出全部程序行
LIST n	列出行号为n的程序行
LIST n_1，	列出由n_1到末尾的所有程序行
LIST，n_2	列出由开始到n_2的所有程序行
LIST n_1，n_2	列出由n_1到n_2的所有程序行

如想中断列表可打CTRL-C；暂停列表可打CTRL-S，恢复列表打CTRL-Q。

三、运行程序——RUN命令

程序输入后可打RUN命令运行程序，运行后返回BASIC命令级。命令格式如下：

命令格式	作用
RUN	从起始行号运行程序。
RUN 行号	从指定行号起运行程序。
RUN 文件名	运行已存入磁盘中的程序。

中断或暂停运行和列表相同。

1-2-3 修改BASIC程序——EDIT命令 程序往往会发生错误，有的可以在运行前被检查出来，有的在运行时由系统检查出来，并且打印出错误信息。错误信息请查看 1-2-7错误信息。

程序发生错误可用上述方法修改，也可用EDIT命令修改。系统的EDIT编辑命令提供了更灵活的方法。EDIT命令格式如下：

EDIT 行号

打入上述命令，进入编辑状态，系统显示出要修改行的行号，然后跳过一个空格，等待输入

编辑子命令。熟练地使用编辑子命令，可以很快地修改好程序。

下面介绍编辑命令的使用：

格式：EDIT〈行号〉

目的：使被设定的那一行进入准备编辑状态。

一、编辑子命令

功能：用来移动游标或在某一行中进行插入、消除、替代、寻找等修改工作。大多数编辑子命令前面可冠以一整数i，如此可使命令执行多次。如不设定，则视为1。

1.移动游标

空格键：利用键盘上的空格键按杆可使游标向右移动。

[←]：使游标向左移动，被游标经过的字符在屏幕上消失。

2.插入新的文字——I命令，X命令

格式：I〈插入的字符〉

功能：在目前游标处插入该字符，[ESC]或[RETURN]可以终止此插入。

格式：X

功能：它将光标移到该行末端，以便插入新的字符，[ESC]或[RETURN]退出此状态。

3.消除不要的字符——D命令，H命令

格式：〔i〕D

功能：消除游标右边 i 个字符。消除的字符被印在两反斜线之间，游标停留在最后一个被消去的字符的右边。

格式：H

功能：消去游标右边的所有字符，然后进入插入方式。

4.找出需要的文字——S命令，K命令

格式：〔i〕S〈字符串〉

功能：找出第i个指定的字符串，然后将光标移到此字符串的前面。假如没找到，游标停在该行末端，所有被找过的字符串都会出现在屏幕上。

格式：〔i〕K〈字符串〉

功能：除了在寻找期间会将所有这种字符串消除外，其余与〔i〕S〈字符串〉作用类似。

5.取代旧的文字——C命令

格式(1)：C〈字符〉

功能：以该字符替代下一个字符。

格式(2)：iC〈字符串〉

功能：字符串由i个字符组成，执行后显示出新的i个字符，并返回编辑状态。

6.结束或重新进入编辑状态——E命令，Q命令，L命令，A命令

按[RETURN]会印出该行其余部分，存下修改过的部分，且跳出编辑状态。

格式：E

功能：除了不印出一行其余的部分外，其它与[RETURN]相同。

格式：Q

功能：结束编辑，返回等待命令状态，且不储存编辑状态中已更改的内容。

格式：L

功能：印出该行其余部分且存下已修改过的内容，并使游标回到该行之开头。

格式：A

功能：使游标回到该行开头，重现该行内容，使你可进行再修改。

注意：在编辑状态下，输入不可识别命令，系统会发出声响，表示不接受此命令。

二、当程序运行时，如发现语法错误，系统将自动在产生错误的那一行进入编辑状态。例如：

```
10  K=2(4)

RUN

Syntax error in 10
10
```

现在可打入编辑子命令，进行修改操作。

1-2-4 保存源程序和调用、运行原有程序

一、保存源程序——SAVE命令

为使内存中的程序不被丢失以备再用，常常把程序保存在磁盘上。SAVE命令可完成此功能，其命令格式如下：

SAVE<程序名>〔,A〕

程序名是加引号的字符串，由1-8个字符（< > . , ; : = ? * ( ) 等除外）组成。A是任选项，在命令格式中包括选项A，系统将源程序以ASCII码形式存入磁盘，否则系统以压缩的二进制码形式存入磁盘。存入磁盘的程序被称为文件。所以程序名也叫做文件名。

SAVE命令执行后，不影响内存中原有的程序。

二、显示磁盘目录——FILES命令

使用FILES命令可以显示出现行磁盘上的文件目录。命令格式如下：

命令格式	作用
FILES	显示现行磁盘上的文件目录。
FILES 程序名	显示现行盘上指定的程序文件。

三、调用和运行原有程序—LOAD命令，MERGE命令

从磁盘中调出程序，可以使用命令LOAD或MERGE。其命令格式如下：

命令格式	功能
LOAD 文件名 .BAS	首先清除内存中原有的程序，再将磁盘中的程序文件装入内存。
LOAD 文件名 .BAS, R	除执行上述操作外，程序装入内存后，立即运行此程序。
MERGE 文件名 .BAS	不清除内存原有的程序，新装入的程序和原内存的程序进行合并(或复盖)，相同行号的行只保留新程序行。

注意：MERGE只允许调用ASCII形式的文件。

1-2-5 文件操作

一、文件改名——NAME命令

NAME命令可给程序改名，命令格式如下：

NAME‹老文件名．文件类型›AS‹新文件名．文件类型›

二、文件的删除——KILL命令

KILL命令可删除现行磁盘上的指定文件。命令格式如下：

KILL‹文件名．文件类型›

其它文件操作可在系统级命令进行，请看2-4-1节内容。

1-2-6 打印机的使用 把LIST，PRINT和PRINT USING前面加上字母L，即改成LLIST，LPRINT和LPRINT USING，即可把应在屏幕上显示的内容传到打印机上输出。

如果在系统级想使用打印机，只要打入CTRL-P就可以实现联机，屏幕上的显示同时传送给打印机。

1-2-7 错误信息

错误码	信息和可能发生的原因
1	NEXT without FOR　NEXT语句没有对应的FOR语句或FOR语句的变量与NEXT语句变量不一致。
2	Syntax error 语法错误如表达式中括号不匹配，出现非法字符，错误标点等。
3	Return without GOSUB　遇到RETURN，找不到调用的GOSUB语句。
4	Out of data　DATA语句中的数据用完，即REDA语句中的变量数目多余数据数目。
5	Illegal function call　非法函数调用。调用函数时传送的参数超出限定范围。下列情况也可引起此错误： (1) 数组下标值为负或太大 (2) LOG函数自变量为0或负数 (3) 使用SQR函数时自变量为负数 (4) 带有非整数指数的负指数 (5) 调用一个未给定起始地址的USR函数 (6) 其它函数使用非法自变量
6	Overflow　计算结果太大，超出BASIC允许的表示范围。
7	Out of memory 程序太大；变量太多；FOR循环嵌套或GOSUB嵌套太深；表达式太复杂。
8	Undefined line　GOTO、GOSUB、IF … THEN … ELSE或DELETE中指定的行号并不存在。
9	Subscript out of range　数组元素的下标超出数组定义的维数或大小。
10	Redimensioned array　一个数组只可定义一次，不可重复定义，或已被自动定为11维后，又出现一个DIM语句定义此数组。
11	Division by zero　在表达式出现以零为除数或幂运算中发生零的负方幂。
12	Illegal direct　把非立即命令用于立即方式。
13	Type mismatch　把数值量存于字符串变量，或反之；调用函数的参数和

函数变元类型不一致，或反之。

14 Out of string space 字符串变量超出BASIC允许的长度。

15 string too long 试图建立的字符串的长度超过255个字符。

16 String formula too complex

字符串表达式太长或太复杂，此表达式必须拆成较短的表达式。

17 Can't continue 在下述情况下，企图继续执行程序：

(1) 已遇到错误，程序被停止；

(2) 执行断点时，程序被破坏；

(3) 程序不存在。

18 Undefinded user function 调用未被定义的USR函数。

19 NO RESUME 进入错误处理子程序，但子程序内不具有RESUME语句。

20 RESUME without error

在进入错误处理子程序前已遇到了RESUME语句。

21 Unprintable error (没有合用的错误信息，用来描述此错误状况)

22 Missing operand 表达式中运算符后无操作数

23 Line buffer overflow 试图在一行上输入太多的字符。

26 FOR without NEXT FOR之后没有与其配合的NEXT。

27 WHILE without WEND WHILE之后没有与之对应的WEND。

30 WEND without WHILE WEND之前没有与之配合的WHILE。

31 Reset error Apple键盘上的RESET被按下。

32 Graphic statement not implemented 图形语句没被设置

(此信息只用于GBASIC)。

1-3 Cromemco机上的使用

在Cromemco机上使用的BASIC是在CDOS操作系统支持下的16K扩展 BASIC。它具有许多扩展BASIC版本的特点。系统文件为BASIC. COM。

一般系统配置2～4个磁盘驱动器，它们的代号分别为A，B，C，D。A 驱动器为主驱动器。

1-3-1 系统的起动和关闭

一、开机准备

准备主磁盘（系统盘）一块，即包括CDOS引导程序和BASIC. COM 等系统文件在内的磁盘；准备用户盘一块，若是没初始化的新盘，开机后参照3-1节INIT 命令进行新盘初始化工作。

二、开机——引导CDOS

接通终端和主机电源，屏幕应显示出：

CROMEMCO RDOS1

如果接着出现的是"; "，需要打入：

B

再将主磁盘插入A驱动器，关好驱动器小门，驱动器在发出正常声响后，屏幕上显示出如下信息：

CDOS Version 02.17
Cromemco Disk Operating System
Copyright (c) 1978, 1979 Cromemco, Inc
A.

符号“A.”是CDOS操作系统提示符，它的出现表明系统引导成功，同时表示A驱动器是现行驱动器。若希望把现行驱动器改为B驱动器，打入：

B:

此时提示符变成B.。同样可打入A:，又可换回A驱动器。

上述转换过程如下：

A. B:

B.

B. A:

A.

现在用户可使用CDOS操作命令进行磁盘和文件操作。

三、进入BASIC系统

在提示符A.下打入：

BASIC

几秒钟后，屏幕显示出BASIC的版权信息和提示符“»”。提示符“»”的出现表明BASIC解释程序已装入内存，等待用户打入的BASIC命令。

上述过程如下：

A. BASIC

CDOS 16K BASIC, VERSION 5.4

»

四、退出BASIC与关机——BYE命令

退出BASIC使用BYE命令，其过程如下：

»BYE

A.

注意：退出BASIC将丢掉内存中的程序，所以退出前要先将程序保存好。

关机前一定要先从驱动器取出磁盘，再关掉主机和终端电源，以防破坏磁盘信息。

1-3-2 建立和运行BASIC程序

一、建立源程序——SCR命令，AUTOL命令

建立新程序前，首先打入SCR命令，以清除内存原来的程序，然后按BASIC语言程序格式逐行打入程序。

BASIC系统提供AUTOL命令，可自动提供语句行号，使你注意力集中在程序上，而不必考虑对行号的输入。AUTOL命令格式如下：

AUTOL n, m

n——起始行号

m——行号增量

例如：

AUTOL 100，10

起始行号100，行号增量10，可能取值为：100，110，120，……

连续两次按 RETURN 键退出AUTOL命令。

上述建立源程序的过程，举例如下：

≫SCR

≫AUTOL 10，10

≫10 P=3.14159

≫20 INPUT "R=", R

≫30 S=P*R**2

≫40 PRINT "S="; S

≫50 END

≫60

输入程序时，如果发现错误，若未按 RETURN 键，可按 DELETE 删除，再打入正确的内容。如果已经按过 RETURN ，则可重新打入发生错误的行号及正确内容。

二、程序的显示——LIST命令

LIST命令可将程序列表以备检查，其命令格式为：

LIST ；列出全部程序。

LIST n ；列出行号n到末尾的全部程序行。

LIST n，m (m>n) ；列出从行号n到行号m之间的全部程序行。

例如：

≫LIST

```
 10  P=3.14159
 20  INPUT "R=" , R
 30  S=R*R**2
 40  PRINT "S="; S
 50  END
```

三、运行源程序——RUN命令

程序运行使用RUN命令，其命令格式为：

RUN ；从最小行号起，运行内存中当前的程序。

RUN "程序名" ；运行已存入磁盘中的程序。

例如：

≫RUN

R = 2

S=12.566359999997

＊＊＊50 END＊＊＊

在程序运行过程中，若试图中断运行，可使用CTRL-C；若暂停运行，可用 CTRL-S（它们是在压住 CTRL 键后，再按相应的字符）。按任意键，可重行运行。

1-3-3 程序的修改——DELETE 命令 程序经常发生错误，有些语法错误在输入时就能被系统发现；例如：

```
AUTOL 10,10

≫ 10 PI=3.14159
        $
ERROR 1 — SYNTHX
```

系统指出发生错误的位置和错误性质，供修改使用。有些错误在运行时被系统发现，并打出错误信息（错误信息与说明见1-3-6）。

Cromemco的 BASIC 没有提供编辑功能，因此只能用重新打入发生错误的行号及新内容来修改。

程序提供了DELETE 的命令，用来删去不需要的程序行，命令格式为：

DELETE n ；指出删除行号为n的程序行。

DELETE n, m ；指定删除从行号n到m的程序行。

1-3-4 保存源程序和调用运行原有程序

一、保存源程序——SAVE命令，LIST命令

把程序存在磁盘上，可以防止因误打入SCR命令或停电、关机而丢失。存入磁盘的程序叫程序文件或文件。

Cromemco BASIC 提供了两种命令：

SAVE “文件名.文件类型” ；以机器码形式存入磁盘

LIST “文件名.文件类型” ；以ASCII码形式存入磁盘

二、磁盘目录的显示——DIR命令

打入DIR命令，可以显示磁盘上文件目录清单，命令格式为：

DIR “＊.＊”

三、调用运行老程序—LOAD命令，ENTER命令

如果程序是用SAVE命令，以机器码形式存入的，则用LOAD命令调用，或用RUN命令直接运行；用LIST命令以ASCII码形式存入的，应用ENTER命令调用。命令格式如下：

LOAD “文件名．文件类型”

RUN “文件名．文件类型”

ENTER “文件名．文件类型”

在使用上，LOAD命令和RUN命令总是先清内存，再调入老程序，而 ENTER 命令不清除内存，这为用户提供了程序连接和覆盖的功能，扩大了内存的使用效率。

1-3-5 启动打印机 如果打印机电源接通，任何时候打入 CTRL-P 控制命令，都可以启动打印机，将屏幕显示的内容输送到打印机上。

打印机使用完毕，再按CTRL-P可以停止使用。

1-3-6 错误信息 Cromemco BASIC的错误分为致命性错误和非致命性错误两种。

一、致命性错误

编　号	信　息	含　义
1	Syntax (语法错)	源程序中的语句不符合BASIC语法规则例如： 括号不匹配：A=(B*(C) 单词拼写错：PIRNT A 数据类型错：A$=3*A 标点符号错：PRINT A(7；2) 因为这么多错误只用一个出错信息表示，所以在发生错误的语句行下面，打印一个“$”符号，大致指出发生错误的地方。
2	Using Syntax (使用格式打印语句句法错)	PRINT USING 语句的字符串格式出错。例如： PRINT USING “**##!!!”，3.2E9 (仅有三个感叹号，实际要求有4个)
3	Number of Arguments (变元个数错)	调用函数时所给的自变量个数与原来赋于此函数的自变量个数不同。例如： DEF FNA (X,Y)=X+Y PRINT FNA(J)
4	No Disk in System (系统中没有磁盘)	在独立系统中，产生存取磁盘的要求，而系统中没有磁盘。如： OPEN\1\“LP” (用户原来想指“$LP”，但错写成“LP”，产生存取磁盘要求)
5	Illegal statement (非法语句)	1) 敲入语法错误的程序行，不经修改就运行，将导致这种错误。 2) 某些语句在一些系统中可能被宣布为无效。例如：POKE语句在分时BASIC系统中是非法的。
6	Print Item Size (打印项大小错)	企图打印一条字符数比现行页宽还要多的数据项。例如： SET 0，10 PRINT “LOTS OF CHARACTERS”
7	Too Mamy GOSUBS (转子程序太多)	子程序嵌套的层次超过BASIC允许的深度。
8	EXP Too Complex (表达式太复杂)	表达式层次太多，括弧或函数引用太多。
9	Return,No GOSUB Active (有RETURN指令无GOSUB指令)	程序没有可返回的地方，这可能是因删除一行GOSUB语句，以后又遇到它的RETURN语句所引起。
10	NEXT Without FOR	FOR和NEXT语句必须配对。如果包含FOR语句的程序行被删去，就要产生这种错误。

续表1

编　号	信　息	含　义
12	Function not Defined （函数未定义）	程序引用未定义过的用户函数。如果含有FNS（X）的程序行被删除，函数不再有定义。
13	DIMension Statement Error （维数语句错）	DIM 语句中有不符合规定的自变量。例如负数：DIM A(－20)；以及下标太多：DIM B(5、5、5、5)；整数太大，大于16382。
14	GOTO/GOSUB undefined Line Number （未规定行编号）	GOTO 或 GOSOB 语句涉及的行号不存在。
15	Subscript Values （下标值错）	使用的下标值必须比DIM语句中所规定的小。否则将产生此错误。
16	Number of Subscripts （下标个数值）	用变量表示的下标个数 与DIM 语句中的下标个数不匹配。
101	End of Statement/End of Line （语句结束或行结束错）	这是一个内部的 BASIC 错误，请写出一个记录并邮寄 Cromemco Customer Service Dept.（克罗冯柯公司用户服务部）。
102	Array or String Space Overflow （数组或字符串区溢出）	没有足够的内存来存贮数组或字符串

二、非致命性错误

编　号	信　息	含　义
128	File Not Fnd （文件找不到）	在磁盘里找不到文件（文件不在目录中）或在设备目录中没有该设备名。
129	File name （文件名错）	在独立 BASIC 系统中用了磁盘文件或传送了非法文件名。
130	Invalid Cmd for Device （无效设备命令）	给设备一个不能执行的命令。例如：给行式打印机一个读命令。
131	File Already open （文件已打开）	把 OPEN 命令加到已经打开的文件上。
132	File not open （文件未打开）	企图读写未打开的文件。
133	File Number （文件号错）	文件号超出允许范围。文件号必须大于0，小于或等于最大通道数。

续表 1

编　号	信　　息	含　　义
134	Cannot Open File (打不开文件)	来自设备驱动程序（或CDOS）的信息。
135	No Fils Space (没有文件空间了)	全部文件都在使用中。系统必须有一个不用的通道以便LIST、ENTER、SAVE 或 LOAD使用。 在 CDOS 运行时，此信息含义为盘上无多余空间（或者目录已满64项）。
136	File Mode Error (文件读写方式错)	企图读“只写文件”或写“只读文件”。
137	File Already Exists (文件已存在)	企图建立（CREATE）已存在的文件。
138	File Read; NO Data (读不到数据了)	读到文件结尾，或在随机存取时想去读未曾写入的文件部分。
139	File Write (写文件出错)	从 CDOS 来的信息。企图写一个有写保护的盘或者正在写盘时出错。
140	File Position 文件位置错	企图读记录号为负的记录或记录大于240K字节。
141	No Channel Available (无通道可用)	所有入/出通道都在使用。（最大通道数由系统决定）
200	Invalid Hex Number (不合理的16进制数)	无效的16进制数。16进制数字符必须在0～9和 A～F 之间。
201	Integer Overflow (整数溢出)	赋于整数变量的值大于32767。
202	Function Arg Value (函数变元值错)	用非法自变量调用函数，例如：SQR(－2)。
203	Invalid Input (非法输入)	企图输入非数值的数据给数值变量。
204	Input (输入项太多)	企图输入比 INPUT 语句中规定的还要多的数据项。
205	Not Dimensioned	引用未曾定义维数的下标变量。
206	No Data Statement	在 Data 语句供应的数据被读完之后还想再读。在有READ语句而无DATA语句、或者在DATA语句中没有象READ 语句中所指定的那么多变量等情况下，都会产生这种错误。

续表2

编　号	信　　息	含　　义
207	DATA TYPE Mismatch (数据类型不匹配)	企图读数值给字符串变量或读字符给数值变量。例如： 10 DATA 5 20 READ A$
208	Number Size (数值大小错)	企图把 9.99E+62 到 9.99E-65范围之外的值赋于变量。
209	Line Length (行宽错)	一行的字符个数超过132字符
210	Input timeout (输入时间到)	详见第二篇有关内容
250	Overflow /underflow (上溢或下溢)	浮点运算产生的数值超出9.99E+62到9.99E-65 的范围。例如：A=1/0。或者，整数运算得出的结果在-32768到32767的范围之外。

1-4 IBM PC机上的使用

IBM PC所使的BASIC有三种版本。第一种称为 Cassette BASIC，它是三种中能力最差的一种，适用所有IBM个人计算机。若所使用的系统配置有磁盘驱动器，可以使用另一种能力较强的Diskette BASIC，这种BASIC除具有Cassette BASIC的所有指令外，还增加了磁盘操作指令。第三种BASIC称为Advanced BASIC，它除具有上述功能外，还可以绘图，演奏音乐或控制其它外部设备（如游戏控制器等）。但此种BASIC的使用，必须装有彩色/绘图接口 (Color/Graphics Interface)。

在系统配置上，值得注意的是显示器的选择，应选用一行为80字符的单色或黑白显示器，IBM彩色显示器，不能满足要求，而需要在系统启动以后；打入下述指令，转换成一行80字符：

WIDTH 80

1-4-1 系统的启动 首先叙述使用磁盘系统的启动过程。

1.开机

标有DOS Diskette的磁盘是系统盘，它除带有磁盘操作系统外，还包括Disktte BASIC、Advanced BASIC的解释程序以及一些DOS的外部命令。

将系统盘插入左边的驱动器中（若是只有一部磁盘驱动器的系统，则直接插入驱动器中）。关好驱动器小门，依次接通显示器、打印机和主机电源，此时显示器显示如下信息：

Current date is Tue 1-01-1980

Enter new date: _

("_" 是IBM PC机的游标)

提示应输入新的日期，日期的格式如下：

mm-dd-yy

其中：

mm是表示月份的1～12中的1位或2位数字。

dd是表示日的1～31之中的1位或2位数字。

yy是表示年的80—99之中的2位数或1980～2099之中的4位数。

数字之间的分隔符可用连字符“-”或斜杠“/”。

当输入无效日期，系统会重复提示；当输入正确日期，屏幕会显示如下的信息。

```
Current time is 0:00:07.96
Enter new time:_
```

提示应输入当前的时间，时间的格式如下：

```
hh:mm:ss.xx
```

其中：

hh是表示小时的0～23之间的1位或2位数字。

mm是表示分的0～59之间的1位或2位数字。

ss是表示秒的0～59之间的1位或2位数字。

xx是表示1%秒的0～99之间的1位或2位数字。

正确的分隔符是冒号和小数点。

当输入无效的时间，系统会重复提示；当输入正确的时间，屏幕上显示如下信息：

```
The IBM Personal Computer DOS
Version 1.10 (C) Copyright IBM Corp 1981, 1982
A>
```

这里A>是DOS 系统提示符，它的出现表明已完成引导DOS操作系统。提示符中的A表示现行驱动器是A驱动器。

现行磁盘驱动器是可以改变的。只要打入新的驱动器标志符（如B:），就可以改变现行驱动器。过程如下所示：

```
A>                  ; 原提示符
A> B:               ; 打入新的驱动器标志符
B>                  ; 新的提示符
```

每当看到A>或B>时，系统总是处在等待输入DOS命令状态。

2．进入BASIC系统

在系统提示符A>下打入：

```
BASIC
```

并按↵键，屏幕上显示出BASIC的信息：

```
The IBM Personal Computer Basic
Version D1.10 Copyright IBM Corp. 1981, 1982
61371 Bytes free
OK
```

OK是BASIC提示符，表示系统已进入BASIC命令状态。

对于不使用磁盘的系统，或者你不想用磁盘，在接通主机电源后，系统会自动进入Cassette BASIC，出现BASIC提示符“OK”，等待输入命令。

3．退出BASIC系统

由于 Diskette BASIC是受DOS 磁盘操作系统支持的，必要时要直接用DOS系统管理文件。这首先要退出BASIC命令级，返回DOS系统级。

SYSTEM命令可使系统返回DOS系统级。过程如下：

```
OK                          ；BASIC提示符
SYSTEM↵                     ；打入SYSTEM命令
A>                          ；系统提示符
```

↵符号表示Enter键。注意：BASIC的命令和程序行都是在“OK”提示符下打入的，并以↵键结束。只有按↵键，系统才会接收这条指令，并相应作出处理。在下面叙述中，我们省略了↵键的符号。

1-4-2 建立和运行BASIC程序

一、建立源程序——NEW命令，AUTO命令

在输入新的程序前，应打入NEW命令，以清除内存中原有的程序，然后按 BASIC 语言的程序格式逐行打入程序。

在输入源程序的过程中，可以使用AUTO命令自动产生行号。AUTO 命令可在建立程序的任何时候使用。它的命令格式为：

AUTO 〔起始行号〔，行号增量〕〕

省略起始行号或行号增量，它们的约定值是10。例如：

命令	可能产生的行号序列
AUTO	10，20，30，……
AUTO 100	100，110，120，……
AUTO ,5	10，15，20，……
AUTO 50，20	50，70，90……

若想终止AUTO 命令，可在压下 CTRL 键时按 BREAK 键或 C 键。（读作control-break或control-c）

上述过程举例如下：

```
OK
NEW
OK
AUTO
10  PI=3.14159
20  INPUT "R=" ,R
30  S=PI*R*R
40  PRINT "S=";  S
50  END
60  CTRL-BREAK
```

OK

打入程序时，如果在按⌫键前发现错误，可以将标有⌫的键（退后键），按若干次，把光标退回到发生错误的位置，再打入正确的内容，进行修改。一旦按了⌫键，只能用重新打入程序行的方法，进行修改；也可以暂时不去处理，待全部程序输入完，用编辑命令去修改。编辑命令将在程序修改一节介绍。

二、程序的列表显示——LIST命令

在程序建立过程中，由于行号可以不连续，允许随时增加或删除程序行。所以屏幕上所显示的内容，和内存中实际存贮的内容常常不一致。若想检查当前内存中的程序，可使用LIST命令。命令格式如下：

LIST n, m

n: 起始行号

m: 终止行号

IBM PC 的屏幕一次只可显示25行信息，若要列出 1～25 行的程序，可使下面的命令行：

LIST 1—25

LIST命令格式还有几种变化，其格式和说明如下：

```
LIST —m          ; 从开始行列到m行
LIST n—          ; 从n行列到末尾
LIST n           ; 只列出第n行
```

例如：

```
OK
LIST
10  PI=3.14159
22  INPUT "R=";R
30  S=PI*R*R
40  PRINT "S=";S
50  END
OK
```

三、运行程序——RUN命令

RUN命令可使系统进入执行状态，从最小行号开始执行内存中的程序，程序运行后，返回命令状态。

命令格式：

RUN

例如：

```
OK
RUN
R = 2
S=12.56636
```

若想中断程序运行，可打CTRL-BREAK 或 CTRL-C，暂停程序运行，返回命令状态，等待接受来自键盘的新命令。此时若打CONT可恢复程序运行。

1-4-3 修改BASIC 程序——编辑功能 程序发生错误时，可以用重新打入程序行的方法进行修改，也可以使用系统提供的编辑功能，进行修改。

一、编辑的一般步骤：

编辑是指把打好的程序修改更正的过程，一般分三个步骤：

1.找出需要修改的部分，把光标移到此处。

2.打入编辑命令，更改错误内容。

3.把更改后的内容按 [←┘] 存入内存。

二、编辑功能键

IBM PC的大部分编辑功能键位于键盘右侧的数字键上，它们是：

[↑]：游标上移一行。

[↓]：游标下移一行。

[←]：游标左移一格。

[→]：游标右移一格。

[Home]：游标移到屏幕最左上方。

[CTRL]－[Home]：清除屏幕，游标移至最左上方。

[CTRL]－[→]：游标移至下个字(Word)的开头。

[CTRL]－[←]：游标移至上个字的开头。

[DEL]：删除字符。

[Ins]：进入插入状态，在游标处插入新的内容，将原内容右移。

[END]：把游标移到所在行的终点。

三、编辑功能键的使用

假定有如下程序：

```
10  PRIMT X, Y, Z
20IF A=5 THN 50 ELSE 30
_
```

程序中有两处拼写错误：PRIMT和THN，并且希望第10行的X，Y，Z改成A，X，Y，Z，第20行的ELSE30去掉。

修改过程如下：

1.修改第10行

按↑键两次，再按→键六次，使游标移到字符M的下方，屏幕显示变成：

```
10  PRIMT  X, Y, Z
```

20 IF A=5 THN 50 ELSE 30

打字母N，M将被取代，屏幕显示变成：

10 PRINT X, Y, Z

20 IF A=5 THN 50 ELSE 30

依要求，需要在X前插入一个字母A，所以再按三次⇦键，将游标移到 X 的下方，然后 按 Ins 键，进入插入方式，打入“A,”屏幕显示变成：

10 PRINT A, X, Y, Z

20 IF A=5 THN 50 ELSE 30

按⇦键退出插入方式。到此第10行的修改工作结束。

2.修改第20行

现在屏幕显示为：

10 PRINT A, X, Y, Z

20 IF A=5 THN 50 ELSE 30

先按⇦将游标移到“N”处，按 Ins 键，再打入字母E，屏幕显示变成：.

10 PRINT A, X, Y, Z

20 IF A=5 THEN 50 ELSE 30

再按 Ins 退出插入。

按要求应删去“ELSE 30”。将游标移到“E”处，再按 DEL 键七次，删除“ELSE 30”包括空格在内的七个字符。最后屏幕的显示是：

10 PRINT A, X, Y, Z

20 IF A=5 THEN 50

现在已全部完成修改工作，按↵键把修改好的内容存入内存。

注意：若欲修改的那一行目前没有显示在屏幕上，可以先用LIST命令列出那行内容，再行修改。

四、DELETE命令的使用

如需把程序中的某些程序行删掉，可以使用DELETE命令。命令格式如下：

命令格式	说明
DELETE n	删除指定的第n行。
DELETE n1－n2	删除n1到n2的所有程序行。
DELETE －n2	从开头删到第n2行。

1-4-4 保存源程序和调用原有程序 一旦程序输入内存，就可以把该程序保存在磁盘或磁带中，以免丢失，也便于将来调出来运行或修改。存入磁盘的程序称为文件，程序名也是文件名。

一、源程序的保存——SAVE命令

SAVE命令将源程序以指定的名称存入指定的盘。命令格式是：

SAVE “d:程序名”

d:—是驱动器标识符，左驱动器为A:，右驱动器为B:

程序名——是由1～8个字符组成字符串

如果存入磁带，命令格式为：

SAVE "CASI:程序名"

二、调用老程序——LOAD命令

LOAD 命令把程序从磁盘上调入内存，使用的命令格式如下：

LOAD "d:程序名"

LOAD "CAST:程序名"

为确保程序调入内存，可用LIST命令列表检查。

三、如果使用的是 Casstte BASIC，因为没有DOS 磁盘操作系统的支持，程序只能存入磁带，上述命令格式中的 "CASI:"可以省略。命令格式成为：

SAVE "程序名"

LOAD "程序名"

1-4-5 磁盘的初始化和复制

一、磁盘的初始化

为了在磁盘上记录IBM DOS所能接受的格式，应对新的空盘或内容可废弃的旧磁盘进行初始化。

具体过程如下：

```
A> FORMAT B:                          ; 打入初始化命令
Insert new diskette for drive B:      ; 系统响应
and strike any key when ready
 (在B驱动器插入待初始化的磁盘)
↵                                     ; 按任一键（如Enter键）开始初始化
Formatting…Format Complete            ; 初始化结束
Format another  (Y/N)?
N                                     ; 表示不再继续初始化
```

2 磁盘的复制

为防止误操作而把磁盘上的文件破坏掉，应对系统盘进行复制。复制的操作过程如下：

将系统盘插入A驱动器，初始化的磁盘放入B驱动器。打入：

```
A> DISKCOPY A:B:                      ; 打入复制命令
Insert Source Diskette in Drive A:    ; 系统提示
Strike Any key when Ready
↵                                     ; 如果是复制其它盘，需要更换系统盘
Insert target DiSkette in Drive B:
Strike any key when ready             ; 系统提示
↵                                     ; 开始复制
……
Copy Complete                         ; 复制结束
```

Copy Another (Y/N)?

N　　；表示不再复制

1-4-6　文件操作　Diskette BASIC是受DOS 磁盘操作系统支持的，大部分的文件操作在DOS命令下进行。首先打入：

SYSTEM

从BASIC 命令级退回DOS系统命令级，在提示符“A>”或“B>”下；进行文件操作。

常用的文件操作命令如下：

DIR

格式：DIR〔A:〕〔B:〕

功能：检查指定磁盘中的文件名称和类别。

ERASE

格式：ERASE　d:文件名

功能：删除指定的文件

RENAME

格式：RENAME 旧文件名，新文件名

功能：更改文件名称

DISKCOMP

格式：DISKCOMP A: B:

功能：复制磁盘后，可用此命令对两磁盘进行比较

其它的操作命令请看第四章 IBM DOS 命令的内容。

1-4-7　打印机的使用　为把 BASIC 程序或运行结果打印出来，可以分别用 LLIST 和 LPRINT命令。

同时按CTRL和Prtsc 两键，实现打印机联机，使在屏幕上的每一个输出显示，也都输出到打印机上，直到再按这两键才能结束。

如果需要复制整个屏幕上的信息，可同时按⇧和Prtsc键，将会把信息一次输出到打印机上。

下面讨论打印格式的安排。

一、设定每行长度

打印机每行长度可以不一样，根据纸张的宽度和实际需要进行设定。例如：

WIDTH “LPT1:”, 80

可设定每行80个字符。

二、选择字的大小

IBM点列式打印机一般可印出三种字型：

标准字：每吋10个字符

宽体字：每吋 5 个字符

缩小字：每 8 吋132个字符

改用宽体字，打入下面命令行：

LPRINT CHR$(1)

改用缩小字，打入下面命令行：

```
LPRINT CHR$(15)
```

改回标准字，则打入下面命令行：

```
LPRINT CHR$(27); CHR$(69)
```

三、选择行距

IBM点列式打印机有三种行距选择：

每吋 6 行

每吋 8 行

每吋72/7行

前两者适用于标准字或宽体字，后一种适用于缩小字。

当打印机被打开时，自动设定成行距每吋6行，如需每吋8行，请打入下面命令行：

```
LPRINT CHR$(27); "0"
```

如需选用每吋72/7行，请打入下面命令行：

```
LPRINT CHR$(27); "1"
```

返回每吋6行设定，请打入下面命令行：

```
LPRINT CHR$(27); "2"
```

四、设定每页长度

每页长度由每页设定的行数来决定。开机时系统设定每吋6行，每页66行，即11吋长。

打入下面命令行可改变每页长度：

```
LPRINT CHR$(27); "C"; "33"
```

这条命令使每页长度改为33行。

如果希望打印从下一页起始行开始，可打入下面命令行：

```
LPRINT CHR$(12)
```

以上这些命令也可以用语句中。

1-4-8 错误信息

Syntax Error　程序中含有不明确的指令（拼字错误?）、未成对的括弧、不合法的标点符号、不合法的字符或系数名称。

Undefined line number

程序中用到一些不存在的程序行号。这种错误常发生在不自觉地把一些仍需用到的行删掉的情况。也可能在调试某段程序时，该段调用了还没有写的行时发生。

Overflow　用到的数字太大了，发生溢出。

Division by zero　试图除以 0，这种错误很不易发现，因某些数字太小时，把其当0处理，如程序恰巧用到这个数时，就可能发生此种错误。

Illegal function call　当试图要一函数求出超出其定义范围的解时所发出的信息。

Missing Operand　试图执行一个欠缺必要数据的指令。

Subscript Out of Range　使用一个或多个超出DIM语句定义范围的数组。

String TOO Long　试图使用一个超过255字符的字符串。

Out of Memory

程序不合乎内存空间要求。可能因使用的数组过大或程序的步骤太多或两者混合。

String Formula Too Complex　字符串公式太长或太复杂，可以把此字符串拆成数个

连续的小字符串来改正这个错误。

Type Mismatch

试图为一个字符串常数指定一个数值变量，或为数值常数指定一个字符串变量。

Duplicate Definition　试图为一个已定义大小的阵列再下一次定义。注意，一旦在程序中用到某一数组，纵使您没定义其大小。系统自动把其定义成10。

Next without FOR　有“Next”语句却没有对应的“FOR”。

RETURN without GOSUB　在不是子程序中遇到“RETURN”语句。

Out of Data　试图读取不存在的数据。这项错误可能发生在DATA语句，磁带、磁盘中读取数据时发生。

Can't Continue　在程序终止后（ENDed）试图给一个继续执行的指示（Cont），或程序没运行前就给予继续执行的指示（如于编辑行动后）。

第二章　FORTRAN 和 PASCAL 在 APPLE II 上的使用

本章介绍在Apple II 上如何建立和运行FORTRAN和PASCAL 程序。Apple II 提供了两种FORTRAN语言文本：FORTRAN-80和Apple FORTRAN。两种FORTRAN语言在各自的环境下使用。FORTRAN-80需要有CP/M操作系统的支持，而Apple FORTRAN是在Apple PASCAL 操作系统下工作。因此本章第一节首先描述PASCAL 操作系统，第二节、第三节分别描述 PASCAL 语言和 Apple FORTRAN 语言的使用，第四节描述 CP/M 操作系统和FORTRAN-80的使用。

2-1　Apple PASCAL 操作系统简介

2-1-1　系统概述　Apple PASCAL 是ULSD PASCAL 1.1 版本的翻版。该系统是美国加州大学圣地亚哥分校为小型机设计的。后来许多微型机厂家移植了这个系统，Apple PASCAL就是其中之一。此系统为PASCAL程序提供了良好的环境。

一、基本系统配置

除主机与显示器外，基本系统一般包括两台磁盘驱动器和一台打印机。系统内每一设备都有特定的设备号，其分配如下表所示：

设　备　号	设　备　名	备　　注
#1:	CONSOLE	控制台、键盘与显示屏
#4:	盘片名1:	一号驱动器，常放入系统盘
#5:	盘片名2:	二号驱动器
#6:	PRINTER:	打印机

注：PASCAL系统也可以配置一台磁盘驱动器。

插入磁盘驱动器中的磁盘，在初始化后，都有唯一的盘片名。盘片名由1～7 个 ASCII 可打印字符（?、=、$等专用符除外）组成，并以冒号（:）结尾。例如 APPLE1: 和 FOR1:都是合法的盘片名。

在正常情况下，插入两台驱动器内的磁盘，不能同时使用相同的盘片名。

二、系统结构

操作系统是分层次的，系统级命令属于第一层，通常称最外层命令；编辑子系统、文件管理子系统属于第二层。此外还有一些各层共用的控制命令。常用命令如下：

1.通用控制命令，这是用控制键实现的。

CTRL-A　执行左右换卷，用于控制屏幕上的显示。

CTRL-Z　进入使游标总处于屏幕上的跟踪显示方式。

监视器屏幕显示每行40个字符，但程序行和打印行都可能有80个字符。为了在行宽仅有40个字符的显示器上显示多于 40 个字符的程序行或系统提示行，系统设置了 CTRL-A 与CTRL-Z两个命令。

CTRL-@　中断当前的程序，再压空格键，则系统初始化。

CTRL-F　暂停大程序的输出，但不中断程序运行，直到再打入CTRL-F。

CTRL-S　暂停系统或程序的运行，直到打入下一个CTRL-S。

RESET 或 CTRL-RESET　热启动（主机关机后再开机，则实现冷启动）。

几点说明：

(1) CTRL-A读作control/A，指在压下CTRL键的同时压下A键。同 样，CTRL- 字符等代表在压下CTRL键同时压某个字符键。

(2) 系统用‹ETX›表示CTRL-C

‹RET›表示RETURN（回车键）

‹ESC›表示ESCAPE（换码键）

‹DEL›表示CTRL-X

2.常用系统命令

E(EDIT)　进入编辑子系统。

F(FILE)　进入文件管理子系统。

C(COMPILE)　编译PASCAL源文件，生成P代码的目标文件。

L(LINK)　对编译后的代码文件与程序库中的子程序进行连接，以生成可执行的代码文件。

X(EXECUTE)　运行可执行的文件。

R(RUN)　自动编译工作文件，如成功则接着运行它，顺利运行之后，再把工作文件中的P代码文件记入磁盘。

2-1-2　文件处理子系统

一、文件结构

程序和数据都可以文件形式存于磁盘上，存贮在磁盘上的文件必须有唯一的文件说明来标识。文件说明的格式如下：

盘片名:文件名.文件类型

盘片名由1～7个字母、数字和字符组成，并以冒号“：”结尾。盘片名也可以用设备号表示。当用“＊”作盘片名时，则表示系统盘，也称“根盘”。若省略盘片名，则系统认为是用户事先已指定的盘片名。

文件名由1～20个字符（$、[、=、,、? 以及各种 CTRL 字符除外）组成，通常文件名的选择，以能表明文件特征为好。

文件类型表明文件的属性，该系统约定为三类：.TEXT、.CODE和.DATA。

.TEXT文件是由键盘输入的字符组成的文件，凡进入编辑子系统后形成的文件，存贮后都形成.TEXT文件。任一磁盘文件，若文件类型不是.TEXT就不能用编辑命令去修改它。

.CODE文件是编译之后得到的目标文件。

.DATA文件是数据文件。

二、常用的文件处理子系统命令

文件处理子系统可执行文件的传送、改名、删除以及各种文件管理的操作，文件处理子系统共有18条命令。常用的命令如下：

E(Extended-directory-list)

显示磁盘上的文件目录，通常包括文件名，文件所占用的存贮器长度，上次修改文件日期，文件的起始存贮地址，文件类型等信息，最后显示已经用了多少存贮块，那些块没有用完，其中最大的未用空间占多少存贮块。

L(List-directory)　显示磁盘文件目录摘要（不如E命令详细）。

D(Date)　用来修改日期。

C(Change)　用来给磁盘或磁盘文件重新命名。

R(REMOVE)　清除所指定的磁盘文件。

N(NEW)　专门清除原有的编辑工作文件，以便写入新的工作文件。

S(SAVE)　专门用来以指定的文件名存贮工作文件。

T(Transfer)　用来传送磁盘文件，复制整个磁盘或把文件送到打印机或其它设备。

G(Get)　用来取一文件做为工作文件，供编辑、编译和运行。

K(Krunch)　移动磁盘文件，把分散的空存贮块集中为一个大存贮块，

V(Volumes)　查看该系统中可用设备的现状。

Q(Quit)　退出文件处理子系统，回到系统命令级。

三、进入和退出文件处理子系统

要想进入文件处理子系统，系统必须处于系统命令级，此时屏幕上应显示命令提示行：

COMMAND:E(DIT, R(UN, F(ILE, C(OMP, L(IN

打入字符F，则进入文件处理子系统，屏幕上显示出该子系统命令提示行：

FILER:G,S, N, L, R, C, T, D, Q〔1, 1〕

用户可以按需要打入字符，选择执行某一功能，也可打入“？”，查看提示行的余下部分：

FILER:W, B, E, K, M, P, V, X, Z〔1, 1〕

例如要想更改日期，则打入D，屏幕显示为：

DATE SET:‹1…31› – ‹JAN…DEC› – ‹00 – 99›

TODAY IS XX-XXX-XX

NEW DATE?

XX-XXX-XX是上次使用的日期。接着按规定格式打入正确的日期，进行修改日期的工作（详细操作请看系统启动与关闭）。

要想从文件处理子程序退出，在操作完毕后，可打入Q，即可返回到系统命令级。

2-1-3　编辑子系统简介　编辑子系统的功能是建立和修改.TEXT文件。PASCAL语言程序必须在编辑子系统下建立并保存，熟悉它的使用是发展程序的必经途径。这里给大家介绍最基本的编辑功能。

一、常用的命令和说明如下：

I(INSERT)

I 命令在光标所在位置插入字符或插入一些新的程序行，输入源程序也是在插入状态进行的。

在使用此命令之前，要把光标移动到待插入字符的地方，然后按 I，进入插入状态，再打入要插入的字符，插入完后按下CTRL-C，则刚插入的内容被接受，若按ESC键则刚插入的内容无效。这两种操作后都返回EDIT。

D(DELETE)

用来删除一些错误的字符。使用时先把光标移到要删除的字符上，再按D，然后按→键，则该字符被删除。实际使用时，可利用光标移动命令使光标移动，最后按CTRL-C，则光标移动后的位置与进入D命令状态时光标的位置之间的字符都被删除，若按ESC 键，则刚才的删除无效，并返回EDIT。

X(EXCHANGE)

用来修改字符。使用时先把光标移动到要修改的字符上，再按X，以后打入的字符就取代了光标所在位置上的字符。同样，按CTRL-C则接受修改，按ESC则修改无效，并返回EDIT。

A(ADJUST)

按下A后，可用→键或←键使光标所在的一行字符同时右移或左移。按CTRL-C移动被接受，并返回EDIT。

C(COPY)

COPY 命令 把另一个磁盘文件或拷贝缓冲器中的内容(即上一次插入或删除的内容)抄送到光标与光标左边的字符之间。

Q(QUIT)

Q子命令可退出EDIT，有四种可供选择的方式：

U(UPDATE) 若按 下U，则这次编辑的工作文件将存入系统盘，然后返回到系统命令级。

E(EXIT) 按E则直接返回到系统命令级，刚才的编辑工作无效。

R(RETURN) 按R则又回到EDIT，当用户正在编辑修改程序时，不小心错按了Q键，可用R命令返回EDIT编辑状态。

W(WRITE) 按W可把刚编辑的工作文件，用指定的文件名存在指定的磁盘上，然后允许用户选择是退出还是重返EDIT。

二、光标的移动

上述命令中有些命令的执行与光标所在位置有极大的关系。如I、D、X等命令，都要先把光标移到指定的位置，才能按键以执行相应的命令。

提示行中的第一个字符“>”表示顺着书写的方向编辑，此时按：

空格键 ；光标右移一格

RETURN键 ；光标移到下行第一个字符上

[P]键　　；光标向下移动一页

提示行中的第一个字符“<”表示逆书写方向编辑，此时压：

空格键　　；光标左移一格

[RETURN]键　　；光标移动上行第一个字符上

[P]键　　；光标向上移动一页

设定编辑工作方向方法为：

顺向——压“>”，“.”或“+”键

逆向——压“<”，“,”或“—”键

下面四个键也可以控制光标移动，而不受书写方向的限制：

[←]——光标左移

[→]——光标右移

CTRL-O——光标上移

CTRL-L——光标下移

以上所描述的操作都可以加上重复因子，即先按一个数字键，则该操作可重复执行多次。斜杠“/”代表无穷大重复因子，所以先打“/”后打[←]，则光标返回到文件头；先打“/”再打[→]，则光标移动到文件尾部。

三、进入和退出编辑系统

当系统处于系统命令级，在屏幕上有命令提示行显示时，打入字符E，则进入文件处理子系统，屏幕上会显示出该子系统的命令提示行：

```
>EDIT:A(DJST, C(PY, D(LETE, F(IND, I(NSRT, J
(MP, R(PLACE, Q(UIT, X(CHNG, Z(AP[1, 1]
```

此时，如果系统盘中已经存在名为SYSTEM.WRK.TEXT的工作文件，系统将自动读入内存工作区，并将其首页内容显示在屏幕上，以供编辑。

如果没有工作文件，则显示下述提示行：

```
NO WORKFILE IS PRESENT FILE? (<RET>FO
R NO FILE <ESC-REC>TO EXIT)
:
```

对此提示，有三种回答方式：

1.如果要编辑某盘上的已存在文件，则可打入该文件的文件说明，例如：

```
#5:MYFILE.TEXT
```

2.如果要编辑一个新文件，只要按[RETURN]键，即可将新的工作文件打开。屏幕显示出编辑子系统提示行，再打入I，可开始插入新程序。

3.如果按[ESC]和[RETURN]键，可返回系统命令级。

编辑工作结束后，可打入Q，并根据需要打入字符U、E、R、W退出编辑子系统。(详见编辑命令和说明)

2-1-4 系统启动与关闭

一、启动系统

首先接通打印机、监视器的电源，将系统盘——APPLE1：盘插入一号驱动器，轻轻关好驱动器小门，接通主机电源，即开始PASCAL操作系统的引导。屏幕上将显示出如下信息，表明进入PASCAL操作系统。

```
APPLE II

WELCOME APPLE1, TO APPLE II PASCAL 1.1
BASED ON UCSD PASCAL II.1
CURRENT DATE IS XX-XXX-XX

(C) APPLE COMPUTER INC. 1979,1980
(C) U.C. REGENTS 1979
```

接着在屏幕顶端显示系统命令提示行的一半：

```
COMMAND:E(DIT, R(UN, F(ILE, C(OMP, L(IN
```

这是由于APPLE屏幕显示为40字符，而PASCAL系统在屏幕上显示的信息为80个字符，为了看到命令提示行另一半，可打入CTRL-A，这时可以看命令提示行的另一半：

```
K, X(ECUTE, A(SSEM, D(EBUG, ? [II.1]
```

完整的命令提示行如下：

```
COMMAND:E(DIT, R(UN, F(ILE, C(OMP, L(IN
K, X(ECUTE, A(SSEM, D(EBUG, ? [II.1]
```

提示行中的每一个命令的第一个字符以及下面叙述的子命令的字符，表示了要做的一项工作，使用时只需按单个字母键，不用按RETURN键，即可执行。

2.更改日期

通常磁盘上的日期不是现行日期，最好先更改日期，再做其它工作。

更改日期工作在文件处理子系统中进行。首先按 F，屏幕全清后，在顶部显示出文件处理子程序的提示行：

```
FILER:G , S, N, L, R, C, T, D, Q[1.1]
```

接着按D键后屏幕显示出：

```
DATE SET:<1…31>—<JAN…DEC>—<00…99>
TODAY IS XX—XXX—XX
NEW DATE?
```

XX—XXX—XX——是上次使用的日、月、年

系统所承认的日期格式是：

日－月－年

日、月、年之间用短横线(减号)连接。

日——1至31的数字

月——英文月名的前三个字母

年——年的后两位数字

例如：20—FEB—85，表示1985年2月20日。

回答询问时，若年月不变，可只打入新的日期；若年不变，也可只打入日－月；若直接按[RETURN]键，表示保持原来日期。新的日期打入后，系统将把新日期保存在系统盘上。

现在打入Q命令，返回系统命令级。

3.关闭系统

首先取出驱动器内的磁盘，以免破坏盘上的信息，再依次断开打印机，监视器和主机的电源。

2-2 PASCAL 语言的使用

本节与下一节分别讲述PASCAL语言和APPLE FORTRAN 语言的使用。讲述时都假定已经启动系统，即用户已处在PASCAL操作系统环境下。

应当明确，通常在屏幕顶端，有一提示行，提供可选择的操作，当打入某个选中的字母时，将会看到另一个进一步供选择的提示行或系统执行的指定操作。如果打入错误的字母，则提示行闪烁，不产生动作。对选定的操作只需打入单个字母，并不需要按[RETURN]键，而对系统询问的回答，都要用[RETURN]键结束，表明回答已经输入完毕。

下面叙述发展PASCAL语言程序的过程。

发展PASCAL语言程序所需的全部系统文件分别存于盘片名为 APPLE1:、APPLE2:、APPLE3:的磁盘上，它们所含文件如下：

APPLE1:包含有：

SYSTEM.APPLE
SYSTEM.PASCAL
SYSTEM.MISCINFO
SYSTEM.EDITOR
SYSTEM.FILER
SYSTEM.SYNTAX
SYSTEM.LIBRARY

APPLE2:包含有：

SYSTEM.COMPILER
SYSTEM.LINKER
SYSTEM.ASSMBLER
6500.OPCODES
6500.ERRORS

APPLE3:包含有：

SYSTEM.APPLE
FORMATTER.CODE
FORMATTER.DATA
LIBRARY.CODE

LIBMAP.CODE
SETUP.CODE
BINDER.CODE
CALC.CODE
LINEFEED.TEXT
LINEFEED.CODE
SOROCGOTO.TEXT
SOROCGOTO.CODE
SOROC.MISCINFO
HAZELGOTO.TEXT
HAZELGOTO.CODE
HAZEL.MISCINFO
CROSSREF.TEXT
CROSSREF.CODE
SPIRODEMO.TEXT
SPIRODEMO.CODE
HILBERT.TEXT
HILBERT.CODE
GRAFDEMO.TEXT
GRAFDEMO.CODE
GRAFCHARS.CODE
GRAFCHARS.TEXT
TREE.TEXT
TREE.CODE
BALANCED.TEXT
BALANCED.CODE
DISKIO.TEXT
DISKIO.CODE

2-2-1 建立源程序 建立源程序的过程如下：

一、处理原有的工作文件SYSTEM.WRK，以备存放新的工作文件。

这是调用文件处理子系统，对系统盘上的原工作文件进行适当处理，或废弃，或保留。

(1) 废弃原工作文件

如果原工作文件已经通过编译运行，转存于用户盘的指定文件中，或原工作文件保存的程序不再是必要的，可选择此操作。

在文件处理子系统提示行下，打入子命令N(NEW)，并对询问回答 Y(YES)，即可完成废弃原工作文件的操作，如果还要对原工作文件进行其它处理，可压 RETURN 键，返回文件处理子系统。

其操作过程如下：

在系统命令提示行下打入：

F

屏幕显示：

FILER:G, S, N, L, R, C, T, D, Q [1,1]

再打入：

N

屏幕显示：

THROW AWAY CURRENT WORKFILE?

回答询问，打入：

Y（表示废弃）

或者按 RETURN 键（表示暂不废弃）。

最后打入：

Q(返回系统命令级)

(2) 保留原工作文件内容后，再废弃工作文件

如果原工作文件内容需要保留，可在文件处理子系统提示行下打入子命令S(SAVE)，将工作文件按指定的文件说明转存。转存后再进行废弃的操作。

其操作过程如下：

在系统命令提示行下打入：

F

进入文件处理子系统，显示出：

FILER:G, S, N, L, R, C, T, D, Q [1,1]

打入：

S

屏幕显示出：

SAVE AS?

回答询问，打入指定的文件说明，例如打入：

#5:XX.TEXT

在二号驱动器的盘上产生新文件XX.TEXT。此时再按（1）的操作废弃原工作文件。

二、调用编辑子系统输入源程序

在系统命令提示行下打入命令E，进入编辑子系统，压 RETURN 键回答关于FILE? 的提问，待显示出子系统命令提示行后，打入子命命 I(INSRT)，便进入插入方式，此后可按PASCAL语言程序的格式逐行打入。待输入完毕，打入 CTRL-C，退出插入状态，再打入子命令Q，准备退出编辑。用U回答子命令 Q 的提问，将输入的程序保存 在 SYSTEM.WRK.TEXT工作文件中，并退出编辑子系统，回到系统命令级。

其过程如下：

在系统命令提示行下打入：

E

屏幕显示出：

```
>EDIT:
NO WORKFILE IS PRESENT.FILE? (<RET>FO
R NO FILE <ESC—REC> TO EXIT
:
```

直接压|RETURN|表示要编写一个新程序，接着屏幕显示出编辑命令提示行：

```
>EDIT:A(DJST, C(PY, D(LETE, F(IND, I(NSRT, J
```

打入：

```
I
```

屏幕显示出：

```
>INSERT:TEXT〔‹BS›A CHAR, ‹DEL› A LINE〕
          〔ETX ACCEPTS, ‹ESC›ESCAPES〕
--------
(插入程序正文)
--------
```

插入完毕，打入命令：

```
CTRL-C
```

再打入：

```
Q
```

屏幕显示出：

```
>QUIT:U(PDATE THE WORKFILE AND LEAVE
       E(XLT WITHOUT UPDATING
       R(ETURN TO THE EDITOR WITHOUT UPDAT
       W(RITE TO A FILE NAME AND RETURN
       S(AVE WITH SAME NAME AND RETURN
```

打入：

```
U
```

屏幕显示出：

```
WRITING  ...
YOUR FILE IS xxx BYTES LONG
```

最后，清除屏幕后，重新显示出系统命令级提示行，等待新的命令。

2-2-2 编译、连接、运行 PASCAL 程序 新建立的源程序总是存放在系统盘 APPLE1:上，其文件名为 SYSTEM. WRK. TEXT 的工作文件中。因此编译 PASCAL 源程序，即对工作文件进行编译。PASCAL 系统提供了两种工作方式，操作过程随工作方式而异，下面分两种情况叙述。

无论选用那种工作方式，首先要将盘片名为APPLE2:的磁盘插入 2 号驱动口内，因为编译连接所需要的系统文件在APPLE2:上。

一、使用 R(RUN)命令方式

当要运行的 PASCAL 程序不需要从其它磁盘上读/写数据时，可以使用R命令运行之。

其执行过程是：先调用 APPLE2：上的SYSTEM．COMPILER程序对源程序(SYSTEM．WRK．TEXT)进行编译。若编译时未发现错误，则自动调用 SYSTEM．LINKER，把编译成功生成的目标代码与相应的库程序进接，而形成一个可执行程序。若连接未产生错误，则自动运行所生成的可执行程序。

其过程如下：

在系统命令提示行下打入：

R

屏幕显示出：

```
COMPILING …      (开始编译工作文件)
PASCAL COMPILER 〔1.1〕
<0>…
TURTLEGR 〔2307  WORDS〕
<2>……
STAR 〔1156 WORDS〕                    (STAR 是编译的程序名)
<30>……
xxx  LINES
SMALLEST AVAILABLE SPACE = yyyy WORDS
```

xxx：行数

yyyy：程序所占内存的字数

```
RUNNING  …
```

最后出现运行的结果。

二、使用C、L、X命令方式：

准备运行的程序要用到其它盘上的数据时，则需分别用 C、L、X 命令进行编译、连接和运行。

其过程如下：

首先在系统命令提示行下打入：

C

调用 SYSTEM．COMPILER 程序对源程序进行编译，屏幕显示：

```
COMPILING  …
```

以表示开始编译操作。当编译正在进行时，屏幕上将显示出表示编译过程的信息。

若编译成功，则屏幕显示出：

```
xxx  LINES
SMALLEST AVAILABLE SPACE =  yyyy WORDS
```

同时退出编译回到系统命令级。

然后，在系统命令提示行下打入：

L

调用 SYSTEM．LINKER 程序执行连接操作。屏幕显示：

```
LINKING  …
APPLE PASCAL LINKER 〔1.1〕
```

HOST FILE?

询问主程序名，在问号之后可打入 SYSTEM. WRK或压 RETURN 键，表示主程序是工作文件。接着显示出：

OPENING *SYSTEM.WRK.CODE

LIB FILE?

询问需连接的子程序文件名。在问号后打入子程序的代码文件名，此文件找到后，系统出现新的提示：

LIB FILE?

此时可打入新的要连接的代码文件名。PASCAL 系统允许最多连接 5 个文件。若没有要连接的文件，则在 LIB FILE? 之后，压 RETURN 键，这时屏幕显示：

MAP NAME?

这是询问是否需要把连接过程中的信息保存到一个印象文件中。若需要，则回答一个文件说明；否则压 RETURN 键表示不需要印象文件。

此后系统开始执行连接操作。并显示出连接过程中的相应信息：

READING ……

READING ……

……

OUTPUT FILE?

用压 RETURN 键回答，表示连接后的可执行文件仍存在 SYSTEM. WRK. CODE 中，接下显示：

LINKING ……

COPYING ……

……

最后连接成功，屏幕显示出：

ALL SEGS LINKED

并返回系统命令级，出现系统命令提示行。

最后在系统命令提示行下，打入X命令，运行连接程序后形成的可执行程序。其过程如下：

打入：

X

屏幕显示：

EXECUTE WHAT FILE?

打入：

SYSTEM.WRK

接下屏幕显示出运行结果。

上述过程中，当使用C命令完成编译后，也可直接打入R命令进行自动连接和运行。

三、编译错误信息及处理

无论上述的那种工作方式，在编译时发现错误，都会使编译过程暂停，并打出错误信息。其格式如下：

xxx ‹‹‹‹

LINE n, ERROR yyy:‹SP›(CONTINE, ‹ESC›

(TERMINATE, E(DIT

其中：xxx　发生错误的位置

n　出错的行号

yyy　错误性质代码

操作选择：

1.压空格键，可继续编译，最后和其它错误一起改正。

2.按 ESC 键，可停止编译，返回系统命令级。

3.按 E 键，重返编辑状态，此时屏幕上显示出错误信息，再按空格键启动编辑子系统，把程序的第一页显示在屏幕上，此时光标自动停在出错处附近。现在应使用编辑子命令进行修改。有关程序的修改见2-2-4。

2-2-3 保存工作文件 在建立、运行 PASCAL 程序之后，APPLE1：盘上应有两个工作文件：

SYSTEM.WRK.TEXT　(源程序)

SYSTEM.WRK.CODE　(可执行程序)

如果想保存它们以备后用，可以用命令 S(SAVE) 将它们保存在系统盘以外的盘上。首先将 APPLE2：或其它用户盘放入二号驱动器，然后在系统命令提示行下打入 S 命令，用盘片名：文件名或 #5:文件名回答系统提出的询问，系统则把类型为.TEXT 的工作文件，用指定文件名，其类型为.TEXT 存入指定磁盘，同时将类型为.CODE 的工作文件，用指定的文件名，其类型为.CODE 存入指定磁盘。其过程如下：

在系统提示行下打入：

S

屏幕显示：

SAVE　AS?

回答：

盘片名:文件名 (或 #5:文件名)

屏幕显示：

APPLE1: SYSTEM. WRK. TEXT

→#5:文件名. TEXT

APPLE2: SYSTEM.WRK.CODE

→#5:文件名.CODE

完成此操作后返回系统命令级。

若指定的文件名已存在，则有两种可能：一是取代原有文件，则在提示之后用 Y(YES) 回答；反之，则回答N(NO)，系统再次显示：

SAVE　AS?

此时应打入新的文件说明。

注意不要打入文件类型。

2-2-4 修改PASCAL 程序 程序设计是很难一次成功的，经常发生语法或逻辑错误。前者由于编码错误，或输入时打错字符造成，这类错误在编译或连接过程中被发现，根据屏幕上显示的信息，用户可以知道错误所在。出现逻辑错误，程序可以通过编译，连接，但运行后不能得到预期的结果。无论发生那类错误，都必须对源程序进行修改。

此外，如果用户想改变或扩充原有程序的功能，也需对源程序进行修改。

上述对源程序的修改，都是对现存文件的修改。如果现存文件就是系统盘上的工作文件，则可重新进入编辑子系统，进行各种修改（详细修改命令请看第一节有关编辑子系统的内容）。若修改的文件不是工作文件，首先把含有该文件的磁盘插入二号驱动器内，然后调用文件处理子系统，再用G 子命令清除原工作文件，选定欲修改文件。再进入编辑子系统，进行各种修改。其过程如下：

在系统命令提示行下打入：

F

屏幕显示：

FILER :G, S, N, L, R, C, T, D, Q 〔1.1〕

接着打入：

G

显示提示：

THROW AWAY CURRENT WORFILE?

回答：

Y　　（把原工作文件清除）

又显示：

GET?

回答：

#5:文件名　　（注意不可打入. TEXT）

又显示出：

TEXT & CODE FILE LOADED

打入：

Q　　（退出文件处理子系统）

显示出系统命令提示行，接着打入：

E

显示出编辑子系统的命令提示行。

现在可进行预期的修改，其过程同前，不再赘述。

2-2-5 APPLE PASCAL 文件管理 我们在这里主要介绍常用的查看文件目录、文件的传输及修改盘片名和程序名等操作。这些操作都在文件处理子系统内进行，下面假定已进入文件处理子系统。

一、查看文件目录

查看文件目录使用文件处理子系统的L 命令。操作过程如下：

在文件处理子系统命令提示行下打入：

L

屏幕显示出：

DIR LISTING OF?

回答询问打入：

#5：（或#4：）

屏幕上将显示出该驱动器内磁盘的文件目录。

若打入：

#4：，#6：

则会把文件目录传输到打印机上。

二、文件传输

文件的传输，使用文件处理子系统的T命令。一般有三种操作：

1．将指定的文件传送到指定的目的文件上。

在文件处理子系统提示行下打入：

T

显示出：

TRANSFER?

回答询问打入源文件说明，例如：

#5:XX.TEXT

显示出：

TO WHERE?

回答询问打入目的文件说明，例如：

#4:YY.TEXT

显示出：

APPLE2:XX.TEXT

→APPLE1:YY.TEXT

便将XX.TEXT传送到系统盘的YY.TEXT文件上。

2．文件传送到打印机

首先应将打印机接通电源。

打入：

T

显示出：

TRANSFER?

回答询问打入源文件说明，例如：

#5:XX.TEXT

显示出：

TO WHERE?

此时打入：

#6:（或 PRINTER:）

便将 XX.TEXT 文件传送到打印机打印出来。

3．整盘文件的拷贝

打入：

T

显示出：

TRANSFER?

回答询问打入盘片名或驱动器的设备号，例如：

#4:

显示出：

TO WHERE?

回答：

#5:

显示出：

TRANSFER 280 BLOCKS? (Y/N)

打入：

Y

显示出：

DESTROY *BLANK:?

打入：

Y

接着屏幕显示出传送过程，最后显示出：

APPLE1:→BLANK:?

此时两盘具有相同的盘片名APPLE1:和相同的文件。

三、修改盘片名或文件名

修改盘片名或文件名使用文件处理子系统的C命令。

1．修改盘片名

在文件处理子系统提示行下打入：

C

屏幕显示出：

CHANGE?

回答询问打入新老盘片名，例如：

BLANK:, APPLE2:

便将盘片名由 BLANK: 改为 APPLE2:

2．修改文件名：

在文件处理子系统提示行下打入：

C

屏幕显示出：

CHANGE?

回答询问打入新老文件说明，例如：

APPLE2:OLD. TEXT, APPLE2:NEW. TEXT

便将APPLE2：盘上的OLD. TEXT文件名改为NEW. TEXT。

2-2-6 磁盘的初始化 初始化磁盘的操作是通过调用 APPLE3：中的初始化程序完成的。因此将APPLE3:插入二号驱动器是必不可少的。

其操作过程如下：

将APPLE3:插入二号驱动器中，在系统命令提示行下打入：

X

屏幕显示出：

EXECUTE WHAT FILE?

回答询问打入：

#5:FORMATTER

屏幕接着显示出：

APPLE DISK FORMATTER PROGRAM

FORMAT WHICH DISK (4, 5, 9, …12)?

此时将APPLE3:取出，换上待初始化磁盘，然后打入：

5

显示出：

NOW FORMATTING DISKETTE IN DRIVE 5

表示 #5：中的磁盘正在进行初始化处理。初始化结束，屏幕又会显示出：

APPLE DISK FORMATTER PROGRAM

FORMAT WHICH DISK (4, 5, 9, ……12)?

从#5:中取出已格式化的磁盘。如果还有待初始化的磁盘,可插入驱动器重复上述的操作。如不需要初始化操作，按 RETURN 返回系统命令级。

注意：如果错把已格式化的盘插入#5:中，则系统显示出警告性信息，例如：

DESTROY DIRECTORY OF APPLE3:

因为APPLE3:上的信息不能去掉，应立即打入：

N (NO)

使磁盘信息保存下来。

2-3 APPLE FORTRAN的使用

APPLE FORTRAN 是在APPLE PASCAL 操作系统上运行的 FORTRAN 版本。把FORTRAN语言置于APPLE PASCAL操作系统控制之下，有如下特点：

1.完整的FORTRAN开发环境（包括编辑系统，文件管理系统，代码库和库管理程序，汇编语言的编译程序，以及其它各实用程序）与提供给PASCAL语言的相同。

2.APPLE II机在不增加硬件的前提下，只要用户能理解PASCAL操作系统，就可以在APPLE II机上使用FORTRAN语言。

3.由于 APPLE FORTRAN 和 PASCAL 语言同在一个操作系统支持之下，因此发展FORTRAN或PASCAL程序，只需掌握其中一种操作系统。

4.PASCAL例行子程序，可以连接到FORTRAN程序上，反之亦然。

5.汇编语言子程序既可连接到FORTRAN上，也可连接到PASCAL上，或两者同时连接。

PASCAL操作系统提供了较理想的操作功能，借助于此操作系统，使得发展FORTRAN程序更加简单。

由于APPLE FORTRAN和PASCAL在同一环境下运行，所以建立和运行一个FORTRAN程序的过程和一个PASCAL程序的过程基本相同。其差别仅在于某些显示信息和一些内部过程不同。需要注意的是：在使用APPLE FORTRAN语言时，盘片名与对应的 PASCAL 语言不同，各盘上的文件也略有差异。具体情况如下：

盘片名：FORT1：

文件名：SYSTEM.APPLE
SYSTEM.FILER
SYSTEM.LINKER
SYSTEM.EDITOR
FORTLIB.CODE

盘片名：FORT2：　（做为系统盘）

文件名：SYSTEM.APPLE
SYSTEM.PASCAL
SYSTEM.MISCINFO
SYSTEM.CHARSET
SYSTEM.COMPILER
SYSTEM.LIBRARY

其中：SYSTEM.COMPILER是和PASCAL语言不同的编译程序。

盘片名：APPLE3：

包含的文件和PASCAL的相同。

系统启动时，把盘片FORT2：放入驱动器＃4：中（一号驱动器），把盘片FORT1：放入驱动器＃5：中(二号驱动器)，关好驱动器小门，接通主机电源。启动系统的详细操作，请看2-1-4有关内容。FORTRAN源程序的建立，请参照2-2-1内容进行。

2-3-1　编译FORTRAN程序　编译FORTRAN程序需要以下文件：

1.要编译的正文文件

如不指定磁盘或驱动器，则认为要编译的是系统盘上的工作文件SYSTEM.WRK.TEXT。

2.SYSTEM.COMPILER

此文件一般已装在系统盘上。

有关编译的操作请参照本章2-2-2。不同的地方在于，无论是R命令方式还是C，L，X命令方式，系统总要显示：

LISTING FILE?

询问要不要列表文件。如不需要列表文件，可直接压 RETURN 键，否则打入：

CONSOLE:(或 PRINTER:)

把列表文件传送到控制台或打印机。

编译过程中发现错误，系统显示如下的错误信息：

＊＊＊＊＊ ERROR NUMBER:xxx IN LINE:yy

其中：xxx　　错误性质代码

　　yy　　出错的行号

（有关错误信息请看2-3-3）

接下按E键，可直接进入编辑子程序，进行修改。

2-3-2 连接运行FORTRAN程序 FORTRAN和PASCAL都使用同一连接程序，但用法稍有不同。因为即使一个已编译的 FORTRAN 程序包括在一个独立的代码文件中，仍需对FORTRAN的SYSTEM.LIBRARY中的代码进行连接。为了执行所有的FORTRAN程序，系统盘上必须有SYSTEM.LIBRARY文件。

相应在操作过程中，系统第一次显示：

LIB FILE?

询问子程序文件名时，应打入：

＊　　（或SYSTEM.LIBRARY）

屏幕上显示出：

OPENING ＊SYSTEM.LIBRARY

以表示首先引用建立在系统盘上的SYSTEM.LIBRARY文件。

如果文件SYSTEM.LIBRARY不在系统盘上，会显示下列信息：

NO FILE ＊SYSTEM.LIBRARY

TYPE 〈SP〉 (CONTINUE) 〈ESC〉 (TERMINATE)

现在只有按 ESC 键，返回系统命令级，把SYSTEM.LIBRARY重新安置到系统盘上，再重新进行连接。

有关APPLE FORTRAN的其它操作，请参照本章第二节有关内容。

2-3-3 错误信息 在下述的错误信息中，左边的数字为相应的错误代码，右边的文字为错误说明。

一、编译时错误信息

错误代码	错误说明
1	读源程序块的致命错误
2	标号域中出现非数值字符
3	后续行太多
4	遇到文件的异常结束
5	后续行出现标号
6	在$编译程序控制指令中遗失了域
7	无法打开由$指令所指明的列表文件
8	无法识别的$指令
9	输入的源程序文件不符合正文文件格式
10	所含文件的最大嵌套层数超过了规定值
11	整型常量溢出
12	实型常量溢出(出错)
13	常量中的数字太多

14 标识符太长
15 字符常量扩展到行未端
16 字符常量的长度为零
17 输入中出现非法字符
18 缺整型常量
19 缺标号
20 标号错误
21 缺类型名（INTGER，LOGICAL，或CHARACTER〔*n〕）
22 缺整型常量
23 语句结束处有多余字符
24 缺‘(’（缺左括号）
25 字母被IMPLICIT语句多次说明
26 缺‘)’（缺右括号）
27 缺字母
28 缺标识符
29 DIMENSION语句中缺少维数
30 多次指定数组维数
31 数组维数超过三维
32 EQUIVALENCE语句中出现不相容的自变量
33 在一个类型说明语句中同一变量出现多次
34 该标识符已被说明过
35 该内部函数不能作为自变量传送
36 标识符必须是变量
37 标识符必须是变量或是当前函数
38 缺‘/’ （缺斜杠）
39 被取名的公共块已保存起来
40 变量已出现在公共块中
41 在两个不同公共块中的变量不可等价
42 EQUIVALENCE语句中下标数与该变量的说明不符
43 EQUIVALENCE语句中下标越界
44 两个不同单元等价一个公共块中的同一存贮单元
45 EQUIVALENCE语句向负方向扩展公共块
46 EQUIVALENCE语句使一变量成为两个不同的存贮单元（不在公共块中的）
47 缺语句数
48 在一个共公块中，不允许字符和数字的混和项
49 字符项不能等价于非字符项
50 表达式中出现了非法符号
51 表达式中不能使用子程序名
52 自变量必须是整型或实型

53 自变量必须是整型、实型或字符型
54 相比较的类型必须相容
55 表达式的类型必须是逻辑型
56 下标太多
57 下标太小
58 缺变量
59 缺‘=’（缺等号）
60 等价字符的大小必须相同
61 非法赋值——类型不匹配
62 只能调子程序
63 形参不能出现在公共块中
64 形参不能出现在等价语句中
65 不定大小的数组说明仅能用于虚拟数组
66 可变数组说明只能用于虚拟数组
67 不定数组的维数说明符必须是最后一维
68 可调界要么是参数，要么在它出现之前的公共块中
69 可调界必须是简单的整型变量
70 主程序不能多于一个
71 有名公共区的大小在所有过程中必须相同
72 伪自变量不可出现在DATA语句中
73 公用变量不可出现在DATA语句中
74 子程序名，函数名，内部函数名等不可出现在DATA语句中
75 DATA 语句中下标越界
76 重复计数必须≥1
77 缺常数
78 DATA语句中的类型有矛盾
79 变量的数目与DATA语句表中值的数目不匹配
80 语句不能有标号
81 没有这样的内部函数
82 内部函数的类型说明与内部函数的实际类型不匹配
83 缺字母
84 函数的类型与先前调用不符
85 在本编译过程中，这个过程已出现过（同一过程说明1次以上）
86 在其它单位中已存在的这个过程，已通过 $ USES语句被定义
87 内部函数的自变量类型出错
88 “子程序/函数”先前已作为“函数/子程序”使用过
89 无法识别的语句
90 函数不可是字符型
91 缺END语句
92 一个程序单位不能出现在 $ SEPARATE编译中

93	在函数或子程序调用中实参少于形参
94	在函数或子程序调用中实参多于形参
95	实参类型与形参类型不符
96	被调用的过程还未定义
97	该过程已由 $EXT指令定义
98	字符类型的最大长度为225，最短为1
100	语法顺序不对
101	无法识别的语句
102	非法转入块
103	标号已用于FORMAT
104	标号已被定义
105	转向格式说明标号
106	在此处禁止出现DO语句
107	DO标号必须跟在DO语句之后
108	此处禁止用ENDIF
109	没有与这个ENDIF匹配的IF
110	IF块中，DO块嵌套不当
111	此处禁用ELSEIF
112	没有与ELSEIF匹配的IF
113	DO或ELSE块的嵌套不当
114	缺‘(’
115	缺‘)’
116	缺THEN
117	缺逻辑表达式
118	此处禁用逻辑ELSE语句
119	没有与ELSE匹配的IF
120	此处禁用无条件GOTO语句
121	此处禁用赋值GOTO语句
122	此处禁用块IF语句
123	此处禁用逻辑IF语句
124	此处禁用算术IF语句
125	缺‘, ’(缺逗号)
126	表达式类型错误
127	此处禁用RETURN语句
128	此处禁用STOP语句
129	此处禁用END语句
131	引用的标号未定义
132	DO或IF块没有结尾
133	此处不允许用FORMAT语句
134	FORMAT标号已经引用过

135 FORMAT语句必须给予标号
136 缺标识符
137 缺整型变量
138 缺'TO'
139 缺整型表达式
140 赋值GOTO而非赋值语句
141 无法识别作为选用项的字符常量
142 缺作为选用项的字符常量
143 缺用于单位说明的整型表达式
144 在CLOSE语句中，'，'后缺STATUS选用项
145 OPEN中字符表达式作为'文件名'
146 在OPEN语句中必须出现'FILE = '选用项
147 在OPEN语句中，'RECL = '选用项定义了两次
148 在OPEN语句中，'RECL = '选用项缺整型表达式
149 无法识别在OPEN语句中的选用项
150 直接存取文件在OPEN语句中，必须说明'RECL = '
151 可调数组不允许作为I/O表元素
152 在隐含的DO中遇到语句结束，以'('开头的表达式不允许作I/O表的元素
153 需要作为隐含DO的控制的变量
154 表达式不允许作为读入的I/O表元素
155 语句中的'REC = '出现了两次
156 'REC = '缺整数表达式
157 'END = '选用项只能出现在READ语句中
158 'END = '选用项在语句中出现了两次
159 无法识别的I/O设备
160 I/O语句中无法识别的格式
161 在I/O语句中'，'后缺任选项
162 无法识别的I/O表元素
163 用于格式中的标号还未在格式语句中定义
164 整型变量用作格式赋值，而非ASSIGN语句
165 可执行语句的标号作为一个格式的标号
166 缺供格式赋值的整型变量
167 格式标号定义了一次以上
169 函数调用需要'(　)'

200 在读$USES文件中出错
201 $USES文件中语法出错
202 $USES文件中的子程序或函数名已说明过
203 函数不能送回字符型值
204 不能打开$USES文件

205	$USES语句太少
206	$USES文件中，没有这个单位的·TEXT信息
207	$USES文件中有非法段
208	$USES文件中没有这样的单位
209	$USES语句中遗失了单位名
210	$USES命令结束处有多余字符
211	内部单位不可被复盖
212	$EXT指令中语法出错
213	子程序不能有类型
214	在$EXT指令中子程序或函数名已经被定义过
400	代码文件书写错误
401	JTAB中的实体太多
402	段中的子程序或函数太多
403	过程太长（代码缓冲太小）
404	在系统盘上的高速暂存存储器文件空间不够
405	高速暂存存储器文件上的读错误

二、运行时错误信息

错误代码	错误说明
600	格式遗失最后的')'
601	输入中不需要符号
602	输入中符号后未跟数字
603	输入中缺数字
604	格式中在B后面遗失了N或Z
605	格式中的异常字符
606	格式中不可有零重复因子
607	格式中w域缺整数
608	格式中w域需要正数
609	格式中'·'缺
610	格式中的d域缺整数
611	格式中的e域缺整数
612	格式中的e域缺正整数
613	A格式中的w域需要正整数
614	格式中的何勒内斯域不可在读的过程中出现
615	格式中的何勒内斯域需要重复因子
616	格式中的x域需要重复因子
617	格式中的p域需要重复因子
618	格式中整数出现在'+'或'-'的前面
619	格式中缺少'+'或'-'后的整数
620	格式中带符号的重复因子之后缺p格式
621	格式的最大嵌套层越界

622 格式中‘)’有了重复因子
623 格式中整数后接了一非法的‘，’
624 ‘·’是非法的格式控制符
625 字符常量不可以出现在供读入使用的格式说明中
626 格式中的字符常量不可重复
627 格式中的‘/’不可被重复
628 格式中的 $ 不可被重复
629 BN或BZ格式控制不可重复
630 试图在不知道的设备上完成I/O
631 试图在作为非格式化打开的文件上进行格式化的I/O
632 开始于‘(’的格式错误
633 对于读整数缺‘I’格式
634 对于读实数缺F或E格式
635 在读格式化的实数中出现两个‘·’字符
636 读格式化实数中缺数字
637 读逻辑数中缺L格式
639 读逻辑数中缺T或F
640 读字符中缺A格式说明
641 写整数中缺I格式
642 F格式中的w域不大于d+1
643 E格式中比例因子超出了d域的范围
644 写实数中缺E或F格式
645 逻辑写缺L格式
646 写字符中缺A格式
647 试图对一作为格式化打开单位进行非格式的I/O
648 不能书写分块的输出，文件所在的设备的空间不够
649 不能读分块输入
650 格式化正文文件中的错误，而在最后512个字节中没有(CR)
651 输入中整数溢出
652 读入的字节数太多，超出直接存取单位记录
653 以直接存取单位记录读出的字节数目不正确
654 试图在非块式设备上打开直接存取单位
655 试图对文件记录结束之后的单位进行多余的I/O
656 试图把直接存取的单位定位在记录号为负的记录上
657 试图把直接存取作为顺序文件打开的单位
658 试图把直接存取单位定位在非块式单位上
659 试图定位直接存取单位在所读文件结束之后
660 试图回退连接在非块式设备上的单位
661 试图回退顺序的非格式化单位
662 ASIN或ACOS的自变量越界 (ABS(X).GT.1.0)

663 SIN或COS自变量太大 (ABS(X).GT.10E6)
664 试图对内部单位进行非格式的I/O
665 试图把一个以上的记录放入一个内部单位
666 试图把比内部长度还要长的字符写入该内部单位
667 在无名的单位上调用了EOF

697 整数变量当前未赋予格式标号
698 读入中遇到的文件结束没有'END='任选项
699 整数变量不可赋给一个用于赋值GOTO中的标号
1000 + 编译程序调试出错信息 - 在程序正确时决不会出现

2-4 CP/M操作系统和FORTRAN-80的使用

在APPLE II机上发展FORTRAN程序，还可以在CP/M 操作系统支持下进行。这一节首先讲述CP/M操作系统，然后讲述FORTRAN-80的使用。

2-4-1 CP/M操作系统简介 CP/M (即 Control program/Microprocessors) 原是为8080和Z-80微机的使用而设计的操作系统。由于Z-80板 (Soft Card) 的出现，APPLE II机也使用了CP/M操作系统，当然标准的CP/M和APPLE 的 CP/M 之间，是有所区别的。下面介绍的是APPLE的CP/M操作系统。

一、基本系统配置

要在APPLE II上配置CP/M操作系统，硬件必须有Z-80板，内存至少 56K。所以按一般习惯，主机的4号槽口上插Z-80板；0号槽口插16K语言板。

除上述主机和显示口外，基本系统还应包括两台磁盘驱动器和一台打印机。设备名规定如右表所示。

设　　备	设 备 名
一号磁盘驱动器	A:
二号磁盘驱动器	B:
打印机	LST:
控制台、键盘、显示屏	TTY:

二、文件说明

CP/M系统上的文件必须有唯一的文件说明来标识，文件说明的一般格式为：

d:filename.typ

其中：

d:——磁盘驱动器名A:或B:。若省略d:域系统认为是现行驱动器。

filename——文件名。是由1-8个字符（< > . , ; : = ? * ( )除外）组成。

.typ——文件类型。由"."和三个字符组成，说明文件属性。

文件名和文件类型都可以使用代用字符"*"和"?"。

? :可代替文件名或文件类型中任一字符

* :可代替文件名和文件类型中任意一项

例如：B:XX.C? M

则.COM，.CDM，.CXM都符合.C? M

*.COM

则F80.COM，L80.COM，ED.COM都符合*.COM

.

则代表磁盘上所有文件

三、内部命令的格式和说明

1.ERA命令

格式：ERA文件说明

功能：删除指定磁盘上的文件

例如：ERA XX.FOR ；删去现行磁盘驱动器内磁盘上名为XX.FOR的文件

ERA B:*.PRN ；删除B:磁盘上所有文件类型为.PRN的文件

ERA *.* ；删除现行盘上所有文件

2.DIR命令

格式：DIR文件说明

功能：在屏幕上显示出满足于文件说明的文件名

例如：DIR ；列出现行盘上所有文件名

DIR B:*.COM ；列出B:磁盘上的所有文件类型为.COM的文件

3.REN命令

格式：REN 新文件说明=老文件说明

功能：改变磁盘上的文件名

例如：REN A:X.Y=Q.R ；把A:磁盘上的Q.R文件改名为X.Y

4.SAVE命令

格式：SAVE n文件说明

功能：可把n页（每页256字节）信息，按指定的文件说明存贮到磁盘上。

例如：SAVE 3 X.FOR ；把3页信息存到现行盘上的X.FOR文件上去。

5.TYPE命令

格式：TYPE 文件说明

功能：在屏幕上显示指定磁盘上ASCII源文件内容

例如：TYPE B:X.PRN ；显示B:磁盘上的X.PRN文件内容

四、外部命令

外部命令是由磁盘输入到内存，然后再执行的，所以称外部命令。

1.STAT命令——显示磁盘空间，设置文件属性等。常用格式有：

STAT ；列出所有磁盘的剩余空间

STAT d: ；列出指定驱动器盘上的剩余空间

STAT 文件说明 ；列出该文件的存贮空间

STAT 文件说明$R/O；设置文件为只读状态

STAT 文件说明$R/W；设置文件为读写状态

2.PIP命令——PIP是CP/M的外部设备信息交换程序，它可以完成复制，连接，打印等项操作。

常用的命令格式有：

PIP X=Y ；把文件Y复制到文件X上，文件Y保持不变。

PIP X=Y,Z ；把文件Y和Z连接起来，并复制到X上，Y和Z保持不变。

PIP A:=B:　　；把B:磁盘上文件复制到A:磁盘上。

PIP LST:=X.PRN　；把文件X.PRN复制到打印机上。

3.ED命令——ED是CP/M系统的文本编辑程序，有关ED程序请看下节有关内容。

五、实用程序简介

1.FORMAT程序：FORMAT程序可将磁盘初始化。

命令格式：FORMAT　　；初始化现行驱动器盘。

　　或FORMAT d:　；初始化指定驱动器盘。

2.COPY程序：COPY程序可以复制APPLE CP/M磁盘。

命令格式：COPY d1:=d2:　；把d2:复制到d1:。

　　COPY d1:=d2:/S；只允许把d2:的0～2磁道复制到d1:上。

3.APDOS程序：APDOS程序可将APPLE DOS的文本文件和二进制文件转换成CP/M文件。

命令格式：

APDOS d1:文件名.类型=d2:文件名.类型

将d2:中的APPLE DOS文件转换成d1:的CP/M文件。

2-4-2　系统的启动与关闭

一、系统的启动

把标有CP/M2.2的系统盘放入一号驱动器（A:驱动器），关闭驱动器小门。依次接通打印机，显示器和主机电源，屏幕显示出如下信息：

APPLE-II CP/M

56K Vers 2.2X

(C) 1980 MICROSOFT

A>

其中："A>"为CP/M提示符，它表明系统引导成功。

二、现行驱动器的转换

提示符中的"A"，表示现行驱动器是驱动器A:，若要转换成驱动器B:可打入：

B:

B>

同理若打入：

A:

A>

则又返回A:驱动器。

以上是系统冷启动过程。运行中若更换磁盘，需打入CTRL-C（即按 CTRL 键同时，按 C 键），完成系统的热启动。

三、系统的关闭

关闭系统首先从驱动器取出磁盘，以免破坏磁盘信息，再依次关闭打印机，主机与显示器电源。

2-4-3　CP/M文本编辑程序(ED程序)

一、启动ED编辑程序

ED程序是CP/M的文本编辑（Editor）程序，用来建立或修改CP/M源文件。在CP/M命令级中，打如下命令：

ED{‹文件名›
‹文件名›.‹文件类型›}

即可启动ED程序。

通常ED把由‹文件名›或‹文件名›.‹文件类型›给出的源文件信息，用#A命令读入内存缓冲区，对文件进行处理，修改完成以后，再写回磁盘上。如果所给出的源文件不存在，则ED建立新的文件，并显示出：

NEW FILE

的提示，等待输入源程序。

二、控制字符和编辑命令摘要

1.控制字符

控制字符	功能
CTRL-C	重新启动操作系统(热启动)。
CTRL-E	执行物理回车换行，但输入的字符并不送回主机。
CTRL-I	跳到第n×8列。n为打CTRL-I的次数。
CTRL-L	在检索和替换字符串中逻辑回车换行，不马上执行。
CTRL-V	删除正在输入的行。
CTRL-Z	结束插入（与编辑命令"I"对应)。
RUBOUT	删除刚打入的字符。
BREAK	中断。
CTRL-P	接通打印机。

2.ED命令

命令	功能
nA	从盘上原始文件中读取n行，填写到内存缓冲区内。
±B	把字符指针移到缓冲区的顶部(B)或底部(-B)。
±nC	把字符指针左移n格(-nC)或右移n格(nC)。
±nD	删除字符左边的(-nD)或右边的(nD)n个字符。
E	结束编辑。把编辑好的文件存入磁盘并关闭文件。
nF	寻找第n个F后打入的字符串。
H	结束编辑。关闭并重新打开文件。 (注：把编好的文件存盘，还可继续编辑)
I	插入文本和程序。
±nK	删去字符指针前(-nK)或后(nK)的n行。
±nL	把行指针上移(-nL)或下移(nL)n行。
nM	宏命令定义（把M后的命令串执行n次。如果不写n则执行到出错为止)
nN	寻找第n个在N后打入的字符串。

O　　恢复到原始文件，修改作废。

±nP　　移动并打印若干页。

Q　　退出编辑，关闭文件。但不改变原始文件的内容。

R　　读文件库。

nS　　寻找并替换所寻找的字符串。

±nT　　打印行指针前(－nT)或后(nT)n行。

±U　　把文本中的小写变大写(＋U)。如为－U则不转换。

nW　　写入若干行（在存贮缓冲区满时用）到磁盘文件中，空出缓冲区。

nZ　　停止编辑。

±n　　移动并打印指针前(－n)或后(＋n)的若干行。

n　　把字符指针移到第n行。

注意：字符指针是不可见的。可用0T和T查看它所在的位置。字符指针所指行可由“＊”前的行号得知。

2-4-4 FORTRAN-80的使用 FORTRAN-80语言在APPLE-II机上的使用，是在CP/M操作系统控制下进行的。系统盘除有CP/M操作系统文件外，还应有下述文件：

F80.COM　　FORTRAN的编译程序

L80.COM　　FORTRAN的连接装配程序

FORLIB.REL　　FORTRAN子程序库

下面讲述的内容都假定用户已看过CP/M 操作系统的内容；同时假定系统已启动成功，处于CP/M命令级，提示符是A＞或B＞。

一、建立源程序

建立源程序须调用ED文本编辑程序。

其过程是：

打入ED和准备编辑的文件名（文件类型应该是.FOR），启动编辑程序。当出现编辑命令提示符“*”时，打入插入命令I，进入插入状态，现在应按FORTRAN语言程序格式逐行输入源程序。

编辑系统自动显示出行号，等待插入程序。

输入完毕，按CTRL-Z结束插入，返回编辑状态。

退出插入后，应仔细检查程序有无错误，有错要进行修改；无错可用E(Exist) 命令，退出编辑程序。输入的程序保存在指定的文件中。

假如要建立的文件名为MYFILE.FOR，上述过程如下：

```
A>ED MYFILE.FOR          ; 调用编辑程序
NEW FILE                 ; 表示是新文件
*I                       ; 选择插入方式
1:    PROGRAM MYFILE     ; 出现行号1，等待插入
2:                       ; 以下是插入的程序
3:
:
:
10:   END
```

11：CTRL-Z　　；退出插入状态。

*E　　；退出编辑，返回系统命令级。

A>

二、编译FORTRAN程序

编译FORTRAN源程序用F80.COM文件。编译命令的格式为：

d：目标文件，d：列表文件=d：源文件

如果不指定d：，约定的设备号为现行驱动器；如果以源文件的名作为目标文件名，也可以不写目标文件名。例如：

=X

编译X.FOR程序，并把目标文件放在X.REL中。

Y=X

编译X.FOR程序，并把目标文件放在Y.REL中。

B:=A:X

编译A驱动器盘上的X.FOR程序，把目标文件放在B:X.REL中。

编译操作首先打入F80调用编译程序，出现提示符“*”后，打入编译命令，开始编译。编译中发现错误，打印出错误信息，否则在最后打出提示符，表示编译完成。编辑完成后，可打CTRL-C，退出编译，返回系统命令级。

其过程如下：　　；调用F80编译程序

A>F80

=MYFILE　　；“”为F80提示符，打入=源文件名

FORTRAN-80 Ver.3.3 Copright 1979(C)

By Microsoft-Bytes:4524

（显示编译过程）

*　　；编译结束

*CTRL-C　　；退出编译程序

A>

三、连接和运行程序

连接文件使用L80.COM文件。

1.连接命令格式

连接命令格式为：

d 1:目标文件1，d 2:目标文件2，……，d n:目标文件n

目标文件是经编译而产生的文件类型为.REL文件，在命令格式中.REL可省略。打入命令后L80对指定文件进行连接装配。连接完成后，它将列出其余全部未定义符号，并在其后加一个星号“*”。

2.开关项的使用

开关项用以详细说明程序在装配或执行中所产生的影响。开关项前须带有符号斜杠“/”，常用的开关项有以下几个：

/G：命令L80执行已装入的程序

/E：退出L80，并返回系统命令级

/N：当使用/G或/E开关项时，如指定〈文件名〉/N，可按指定的文件名把可执行的程序存贮在磁盘上。系统赋予的文件类型是.COM。

3.连接操作

首先调用L80，见到提示符“*”后，打入上述的命令格式和附加开关项。如用/G作开关项，连接后自动转入运行。如用〈文件名〉/N/E，连接后产生可执行文件。以后只要打入此文件名，即可运行程序。

连接时发生错误可打CTRL-C退出L80，返回系统命令级。

4.连接举例

(1) 连接并运行

```
A>L80                          ；调用连接程序
*EXAMPL, EXAMPL1/G;            打入连接命令
DATA    3000    30AC
〔304F    30AC    49〕
〔BEGIN EXECUTION〕             ；开始运行

  (显示运行结果)

A>                             ；结束运行
```

(2) 连接并存盘，然后运行

```
A>L80                          ；调用连接程序
*EXAMPL,EXAMPL1,EXAM/N/E;      打入连接命令
DATA        3000        30AC
〔304F        30AC        49〕
A>                             ；连接结束，生成EXAM.COM可执行文件。
A>EXAM                         ；运行程序

  (显示运行结果)

A>                             ；结束运行
```

四、程序的修改

无论在哪一过程发现错误，都必须重新进入编辑，进行修改。修改后再编译和连接运行。经常需要多次重复上述过程，才能得到正确的结果。

1.修改文件最常用的命令是：

K　可删除某些程序行

I　可以插入新的内容

S　可用新字符代替老字符串

进行上述操作，要注意“字符指针”的位置，一般指针位置在“*”前的行号处。

(详细命令请看2-4-3内容)

2.修改操作过程：

```
A>ED <文件名.FOR>          ; 调用编辑程序
*#A                        ; 程序装入缓冲区
*#T                        ; 显示程序

(显示的程序)

*(使用K, I, S等命令)       ; 检查错误并进行修改

*E                         ; 退出编辑
A>
```

2-4-5 FORTRAN-80 错误信息

一、编译错误信息

FORTRAN-80 的编译程序可以检测二种错误：警告性错误和致命性错误。警告性错误信息前有一百分号(%)，致命性错误前加一个问号(?)。如果有行号，随后打印编辑行号或实际行号。行号的后面跟着代码或信息，最后列出检测到错误时扫描的20个字符。

例如：

```
? Line 25:Mismatched Parenthesis:I=(I+J))
    [在25行致命性错误：括号不配对：I=(I+J))]
%Line 16:Missing Integer Variable:I(2 *2,
    [在16行警告性错误：：整型变量名错：I(2 *2, ]
```

当产生任何一种错误时，都要修改程序，编译时错误的程序不能正确运行。下面列出编译时的误错信息。

1.致命性错误

代码	错误信息
100	Illegal Statement Number (非法语句标号)
101	Statement Unrecognizable or Misspelled (语句不可识别或拼写错误)
102	Illegal Statement Completion (非法结束语句)
103	Illegal Do Nesting (非法循环嵌套)
104	Illegal Data Constant (非法数据常数)
105	Missing Name (丢失名字)
106	Illegal Procedure Name (非法过程名)
107	Invalid Data Constant or Repeat Factor (无效数据常数或重复因子)
108	Incorrect Number of DATA Constants (数据常数的数量不正确)
109	Incorrect Integer Constant (整型常数不正确)
110	Invalid Statement Number (无效语句号)
111	Not a Variable Name (不是变量名)
112	Illegal Logical Form Operator (非法逻辑式运算符)
113	Out of Memory (内存不够)
114	Literal String Too Larae (字符串太长)
115	Invalid Data List Element in I/O (在I/O表中，有无效数据表元素)
116	Unbalanced DO Nest (DO循环嵌套失配)
117	Identifier Too Long (标识符太长)

118 Illegal Operator (非法运算符)
119 Mismatched Parenthesis (括号不配对)
120 Consecutive Operators (连续的运算符)
121 Improper Subscript Syntax (下标错)
122 Illegal Integer Quantity (非法整型量)
123 Illegal Hollerith Construction (非法Hollerith结构)
124 Backwards Do reference (Do循环参数反向)
125 Illegal Statement Function Name (非法语句函数名)
126 Illegal Character for Syntax (非法语法字符)
127 Statement Out of Sequence (语句顺序不对)
128 Missing Integer Ouantity (缺少整型量)
129 Invalid Logical Operator (无效逻辑运算符)
130 Ille3al Item Following INTEGER or REAL or LOGICAL (在INTEGER或REAL或LOGICAL的后面出现了非法项)
131 Premature End Of File on Input Device (在输入设备上的文件结束过早)
132 Illegal Mixed Mode Operation (非法的混合运算)
133 Function Call with No Parameters (函数调用无参数)
134 Stack Overflow (栈溢出)
135 Illegal Statement Following Logical IF (逻辑IF后跟有非法语句)

2.警告性错误

代码	错误信息
0	Duplicate Statement Label (语句标号重复)
1	Illegal DO Termination (非法DO循环结束)
2	Block Name = Procedure Name (程序块名 = 过程名)
3	Array Name Misuse (使用的数组名不对)
4	COMMON Name Usage (使用了公用块名)
5	Wrong Numder of Subscrpts (下标的个数错)
6	Array Multiply EQUIVALENCEd Within a Group (在一组元素内，数组被多次等价)
7	Multiply EQUIVALENCE of COMMON (重复等价一个公用块)
8	COMMON Base Lowered (公用块的起始单元被降低。即打算反向扩展公用区)
9	Non-COMMON Variable in BLOCK DATA (在数据块子程序中没有公共变量)
10	Empty List for Unformatted WRITE (无格式写语句没有变量表)
11	Non-Integer Exprssion (非整型表达式)
12	Operand Mode Not Compatidle with Operator (操作方式和运算符不一致)
13	Mixing of Operand Modes Not Allowed (不允许的运算数混合方式)
14	Missing Integer Variable (缺少整型变量)
15	Missing Statement Number on FORMAT (FORMAT语句缺少语句标号)
16	Zero Repeat Factor (重复因子为0)
17	Zero Format Value (格式值为0)
18	Format Nest Too Deep (格式嵌套太深)
19	Statement Number Not FORMAT Associated (语句标号所指的不是格式(FORMAT)语句)
20	Invalid Statement Number Usage (使用了无效语句标号)
21	No path to this Statement (程序中无路径通向此语句)
22	Missing Do Termination (DO循环缺少终止语句)

23 Code Output in BLOCK DATA（在数据块子程序中有代码输出）
24 Undefined Ladels Have Occurred（出现了未定义标号）
25 RETURN in a Main Program（在主程序中有RETURN语句）
26 STATUS Error on READ（在输入中有状态错误）
27 Invalid Operand Usage（使用了无效的操作数）
28 Function with no Parameter（函数无参数）
29 Hex Constant Overflow（十六进制常数溢出）
30 Division by Zero（被零除）
32 Array Name Expected（缺少数组名）
33 Illegal Argument to ENCODE/DECODE（在编码或译码语句中有非法自变量）

二、运行错误信息

运行时，程序出错会在控制终端上显示错误代码。这种代码的前后都有两个星号。例如：

FW

致命性错误会中断程序的执行并返回到操作系统。警告性错误则被忽略（虽然可能导致运算错误），直到发生20次警告性错误，才停止程序的执行。

1.警告性错误

代码	含义
FW	域的宽度太小。
EX	非法幂运算。
IB	超出了输入缓冲区的界限。
TL	FORMAT语句中的左括号太多。
OB	超出了输出缓冲区的界限。
DE	十进制指数溢出(指数大于99)。
IS	整数值太大。
BE	二进制指数溢出。
IN	输入记录太长。
OV	运算溢出。
CN	转换溢出(实型转换成整型时)。
GL	计算GOTO语句数太大。
GS	计算GOTO语句数太小。
SN	正弦函数的自变量太大。
A2	反正切函数(ATAN2)的两个自变量都是零。
BI	在二进制输入、输出期间超出了输入、输出缓冲区的长度。
RC	在FORMAT语句中有负的重复因子。

2.致命性错误

代码	含义
ID	非法格式(FORMAT)说明符。
F0	格式(FORMAT)域宽为零。
MP	在FORMAT语句的格式字符中没有逗号。
IR	企图在整型域中放入实数。
IT	I/O传送出错。
DO	在DO循环中有非法增量或终值。
ML	在FORMAT语句中缺少左括号。

DZ　实数或整数被零除。

LG　在LOG函数中有非法自变量(自变量为负或零)。

SQ　在SQRT函数中有非法自变量(自变量小于零)。

IO　非法I/O操作。

DT　数据类型和FORMAT语句说明不符。

EF　在输入(READ)时遇到文件的结尾(EOF)。

FN　未找到要打开的文件。

DF　在写磁盘时遇到磁盘满。

UN　逻辑设备号太小。

OM　内存用完。

三、连接的错误信息和程序中断信息

? NO Start Address (无起始地址)

已发出了/G开关，但还未装入主程序。

? Loading Error (装配出错)

最后输入的文件不符合LINK-80目标文件的格式。

? Out of Memery (内存用完)

没有足够的内存空间装入程序。

? Command Error(命令错)

不可识别的LINK—80命令。

? ‹File› Not Found(‹文件›未找到)

在盘上没有命令串中给定的文件。

%2nd COMMON Larger/xxxxxx/

公用块/xxxxxx/的第一次定界不是最大的定界。重新排列模块装配顺序或改变公用块定界。会产生这种情况。

% Mult.Def.Globed yyyyyy (重复定义总体符号)

在装配期间，遇到总体（内部的）符号yyyyyy有多个定义。

% Overlaying Program Area, Start = xxxx
,Public = <Symbol name>(xxxx)
,External = ‹Symbol name› (xxxx)

或

% Overlaying Data Area ,Start = xxxx
,Public = <symbol name>(xxxx)
,External = <symbol name>(xxxx)

(复盖的程序区，起点 = xxxx
，公用区 = ‹符号名›(xxxx)
，外部过程 = ‹符号名›(xxxx)

或

复盖的数据区，起点 = xxxx
，公用区 = ‹符号名› (xxxx)
，外部过程 = ‹符号名›(xxxx))

/D和/P开关会使装入的数据被冲掉。

? Intersecting Program Area或

? Intersecting Data Area (程序区交叉或数据区交叉)

程序和数据区交叉，且地址或外部链接入口在这个交叉点中。因为最终的数据在这个交叉点上，所以

不能将它转换成当前值。

？ Start Symbol—‹name›—Undefined(开始符号—‹名›—未定义)

给定/E：‹名›或G：‹名›以后，没有定义已指定的符号。

Origin Above Loader Memory, Move Anyway (Y or N)？或

Origiu Below Loader Memory, Move Anyway (Y or N)？

(用某种方法向上移动装配程序存贮区的起点吗？，或用某种方法向下移动装配程序存贮区的起点吗?)

在给定/E或/G以后，程序区或数据区都有一个位于装配程序存储区以外的起点或顶点（即装配程序的起点在内存的顶点)。如果打入Y，LINK—80 将移动这一区域并使其连续。如果什么都不给，则退出LINK—80。在上面两种的任一种情况下，如果给定了/N开关，则把内存映象存贮起来。

？ Nothing Loaded（未装入程序)

给定了‹文件名›/S或/E或/G，但未装入目标程序。即，在实际未装入程序的情况下，企图检索库程序、退出连接程序或打算执行程序。例：

TEST/N/E 导致'？Nothing Loaded'，因为TEST/N 命令只告诉机器把链接好的程序命令为 TEST.COM，但未装入程序。

？ Can't Save Object File（不能存贮目标文件)

在存贮文件时，发生磁盘出错。

第三章 FORTRAN和COBOL在Cromemco机上的使用

在Cromemco机上使用FORTRAN和COBOL都是在CDOS操作系统支持下进行的。它们使用同一个文本编辑程序（EDIT）和连接程序(LINK)。因此，本章第一节描述CDOS操作系统，第二节较详细的叙述FORTRAN的操作过程，第三节着重于 COBOL与FORTRAN操作中不同之处，关于 COBOL 的操作过程及各步的屏幕显示，请参看第二节内容。

3-1 CDOS 磁盘操作系统简介

3-1-1 系统概述 CDOS 是 Cromemco 磁盘操作系统的缩写(Cromemco Disk Operating System)。

一、基本系统配量

基本系统配量应具有：带有2—4驱动器的主机，终端和打印机。驱动器代号为A：，B：，C：，D：，主驱动器为驱动器A：。插入驱动器的磁盘，一般随驱动器被称为A盘，B盘……。A盘一般是系统盘，它应包括CDOS引导程序在内的系统文件。

二、系统操作命令简介

CDOS 操作命令是分层次的，系统操作命令属于第一层；编辑程序和各管理程序属于第二层。

下面描述的是系统级操作命令。

1.CDOS 文件说明

CDOS 系统上所使用的文件，必须有唯一的文件说明来标识。统一的格式如下：

〔X:〕文件名〔.文件类型〕

其中：

X:——是驱动器标识符，它指明文件所在的位置。

文件名——由1～8个ASCII码字符组成。

文件类型——由"."和1～3个ASCII码字符组成。用来表明文件的属性。

注意：

(1) 下列符号 $ • ? = / • :，及空格都不能用于文件名和文件类型

(2) CDOS系统约定的文件类型：

.BAK	；后备文件
.BAC	；BASIC源文件
.COB	；COBOL源文件
.COM	；可执行的命令程序
.FOR	；FORTRAN源文件
.PRN	；列表文件
.REL	；浮动模块(目标文件)

(3) 当访问的文件没有指明磁盘驱动器，被认为是现行驱动器。

(4) 文件名和文件类型都可以使用代用字符星号(•)和问号(?)。

? 可代替任一个字符

• 可代替余下部分或全体。

例如：

使用代用字符的文件说明	可能涉及的文件
PROGRAM?.REL	PROGRAM1.REL PROGRAM2.REL
PROG•.FOR	PROG.FOR PROGRAM.FOR PROGRAM1.FOR
PROGRAM.•	PROGRAM.FOR PROGRAM.BAC PROGRAM.COB
•.BAC	X.BAC PROG.BAC PROGRAM1.BAC
•.•	所有磁盘文件

2.内部命令：

ATRIB——文件属性命令

格式：ATRIB 文件说明[+][P]

说明：P——属性参数，可选用参数是E，R，W。

E：删除保护

R：读保护

W：写保护

+——表示把后面属性加到文件上不用+号表示删去后面的属性。

DIR——文件目录列表命令

格式：

$$DIR\left[\left\{\begin{matrix}X:\\ 文件说明\end{matrix}\right\}\right]$$

说明："DIR X:"列出指定驱动器盘上的目录清单。

"DIR文件说明" 列出指定文件清单。

ERA——删除文件命令

格式：ERA文件说明

REN——文件改名命令

格式：REN新文件说明 = 老文件说明

说明：用来更改现有的文件名或文件类型。

SAVE——存贮命令

格式：SAVE 文件说明，n

说明：n为10进制的页数(一页256字节)。

TYPE——打印命令

格式：TYPE 文件说明

说明：将ASCII码文件输送到终端或打印机。

2.外部命令(实用程序)

这些命令作为命令文件，存放在磁盘上，需要时可调进内存。

INIT——初始化命令

格式：〔X:〕INIT

功能：用于磁盘初始化操作

说明：运行后按提示，打入磁盘所在驱动器代号。

STAT——检查磁盘状态命令

格式：〔X:〕STAT〔Y:〕

功能：检查以下项目：

用户可占用的磁盘空间总量；

用户区的大小；

用户可使用的磁盘空间大小；

目录总数；

空文件名；

任何磁盘错误。

说明：X:是含有STAT.COM文件所在盘的驱动器代号。

Y:被检查的磁盘所在驱动器代号。

XFER——传送命令

格式：〔X:〕XFER

$$〔X:〕XFER\ 〔/S1/S2\cdots\cdots〕\left\{\begin{matrix}Y:\\文件说明1\end{matrix}\right\}=文件说明2〔，文件说明3\cdots〕$$

功能：用于从磁盘或其它外设上传送文件到另一个磁盘或另一外设上。

说明：S1，S2是下列开关项之一：

A——传送ASCII码文件

C——比较文件，但不传送

R——传送读保护文件

V——传送后校验文件

X:——XEFR文件所在盘驱动器

Y:——目的文件所在盘驱动器

文件说明 1——目的文件说明

文件说明2,3——源文件说明

3-1-2 系统的启动和关闭

一、启动过程

依次接通打印机，终端和主机电源，屏幕显示出如下信息：

CROMEMCO RDOS1

将系统盘插A驱动器中，如果出现的不是“A.”,而是“; ”，请打[B]键，屏幕显示出CDOS的信息和提示符A.:

CDOS version xx.xx
Cromemco Disk Operating System
Copyright (C)1978, 1979 Cromemco Inc.
A.

提示符A.的出现，表示CDOS磁盘操作系统引导成功。现在系统处于CDOS系统命令级。

二、驱动器的转换

提示符A.还表示了现行驱动器是A驱动器，若想把现行驱动器改为B驱动器，打入：

B:

其转换过程如下：

A.B:
B.

同理：

B.A:
A.

三、关闭系统

打开驱动器小门，取出磁盘（以防破坏磁盘信息），依次关闭打印机，主机和终端电源。

3-1-3 文本编辑程序——EDIT 文本编辑程序可以用来建立和修改 FORTRAN和COBOL语言源程序。这里介绍的是最基本的编辑功能。

一、文本编辑控制字符

下面是文本编辑控制字符表。

控制字符	功　能
CTRL-A	打印字符指针前一页
CTRL-C	破坏已编辑文本并重新启动CDOS
CTRL-E	实际的回车-换行，不送入文本缓冲区
CTRL-H {DEL}	没有字符响应的退格
CTRL-I	产生空格或产生相当的间格数
CTRL-L	逻辑的回车-换行，由编辑程序产生并插入文本
CTRL-P	接通或断开打印机
CTRL-R	重新打印当前行
CTRL-S	在控制台上暂停或重新启动输出
CTRL-U	删去当前行
CTRL-X	以带响应符的方式删去字符
CTRL-Z {ESC}	在命令串中F，N，R，S和其它命令的定界符

•：CTRL-字符，　即压住 CTRL 键后，按字符键。

二、文本编辑命令

以下是文本编辑命令汇总表。

命　令	功　　能

nA	把输入文件的n行附加到文本缓冲区的末端。(Append)
±B	将字符指针移到文本缓冲区始端或底部(Beginning or Bottom)
±nC	将指针移过一个或多个字符(Characters)
±nD	从文本缓冲区删去若干字符(Delete)
E	结束编辑，关闭文件，返回CDOS(End)
±nF	在文本缓冲区中寻找字符串(Find)
G	获得保存缓冲区内容并将它们插入文本(Get)
H	关闭文件并从头重新打开文件(Head)
I	在字符指针前插入文本(Insert)
±nJ	字符指针向前或向后跳过n个字(Jump)
±nK	去掉文本n行(Kill)
±nL	字符指针向前或向后移过的n行(Lines)
±nN	在输入文件中寻找一个字符串的下一个出现位置(Next)
O	清除文本缓冲区内容并从头重新打开输入文件(Obliterate)
±nP	移动和打印规定页数(Print • Pages)
Q	不改变文件退出编辑并返回CDOS(Quit)
R	将文件从磁盘读入文本缓冲区。(Read)
±nS	一个字符串替代另一个字符串(Substitute • String)
±nT	在控制台打印规定行数(Type)
±U	大写转换(Upper)
V	检查文本缓冲区剩余空间(Verify)
W	将指针前的行写为输出文件(Write)
±nX	删去文本字
nY	将文本行放入保存缓冲区
Z	设置间格区
±n	移动字符指针n行并打印一行(相当于±nLT)

上表中：n表示命令重复执行次数

±表示命令执行的方向

三、文本编辑信息

信　息	含　义
All text now on Output File	从输入文件和从文本缓冲区得到的所有文本是现行磁盘上的输出文件。这些文本存取前必须发出“H”或“E”命令。
Cannot edit XXX file	XXX 是文件类型的三个字母。指出文件不是预期中被编辑的，需要重定文件类型。
Cannot find“textstring”	见“F”和“S”命令。
CDOS EDITOR version XX.YY	XX.YY是文本编辑程序的版本号。
Discard Text Buffer, restore Input File, and quit EDITOR	如果发出的“Q”命令有错，这个信息给出更正的机会。

(Y/N)?	
End of Input File	当输入文件读入文本缓冲区时，遇到文件的末尾。如果没此信息，表示文件没全部读入缓冲区。
File not found	在"R"命令中，指定的文件未被找出。应该检查驱动器或文件名是否正确。
Goodbye	这是停止编辑工作的信息。是在发出"E"命令后，屏幕上的显示。
Illegal Filename	调用编辑程序时，文件名是非法的。
New File	调用编辑程序时，文件不存在。若不是建立新文件，应检查所用的盘或文件名是否正确。
No Directory Space	磁盘目录区已满，文件名已达到最大数量（无存贮空间）。
Obliterate Text Buffer and rewind Input File(Y/N)?	发出的"O"命令有错，给予用户更正的机会。
Output File Write Error	磁盘空间不够。建议用户在编辑前，检查磁盘的空间。
Rewinding file	这是"H"命令的响应。
ROOM for XXXXX more characters in buffer	XXXXX 是字符(字节)数。是响应"V"命令时的显示。指出在文本缓冲区中剩下的可用有效空间数。
Save Buffer Full	试图读多于2048个字符进入保存缓冲区。(见"Y"命令)。
"X"not legal here	"X"代表不允许的命令或当前命令行中的非法字符或控制符。
"."	文本编辑的提示符。它的显示，表示编辑程序等待命令。

3-2 FORTRAN语言的使用

在Cromemco 机上发展FORTRAN程序，除必须具有CDOS操作系统文件外，还应备有下述文件：

```
EDIT.COM
FOR.COM          ; 编译程序
LINK.COM         ; 连接程序
FORLIB.REL       ; FORTRAN IV 子程序库
```

这些文件最好复制到系统盘上，可以简化操作过程。下面的叙述我们假定系统盘上有上述文件。还应准备一个初始化的用户盘。有关复制和初始化的操作请看上节的内容。

本节和下一节COBOL的使用，都假定系统启动成功，即已处于CDOS操作系统环境下。

3-2-1 建立FORTRAN源程序 源程序的建立是在文本编辑程序EDIT.COM下进行，其建立过程是：

首先调用文本编辑程序。在系统命令级提示符"A."下打入命令行：

EDIT 文件说明

文件说明指定要建立的文件的文件名和存放的磁盘，文件类型应该用.FOR，这会给以后的编译连接带来很多方便。

然后在EDIT提示符"*"下，打入命令：

I

进入插入状态。此时可按FORTRAN IV语言程序格式逐行打入程序。

插入程序过程中，可利用 TAB 键，跳过标号区和继行区，进入语言区。发现错误，可用 DELETE 键删除字符，重新打入新的内容。全部程序输入完毕，按 ESC 键，退出插入状态。

退出插入后，可打入命令：

B#T（或－P）

显示刚插入的内容，以便检查错误。

最后打入命令：

E

退出EDIT文本编辑，并将程序以事先指定的文件说明存入磁盘。

假定要建立的文件为XX.FOR，上述操作过程和系统显示如下：

```
A. EDIT XX.FOR                          ；调用编辑程序

CROMEMCO TEXT EDITOR
VERSION 00.10
NEW FILE                                ；表示是新文件
* I                                     ；打入插入命令
.....
(插入的程序)
.....
ESC                                     ；退出插入
* B#T                                   ；显示插入内容
.....
(显示的程序)
.....
* E                                     ；退出编辑
GOOD BYE
END OF INPUT FILE
A.                                      ；返回系统命令级。
```

3-2-2 编译源程序 编译FORTRAN程序使用FOR.COM文件。

首先调用编译程序，打入：

FOR

屏幕显示出提示符"*"后，打入编译命令。其格式如下：

d1：文件名1.REL，d2：文件名2.PRN＝d3：文件名3.FOR

其中：

d1，d2，d3分别是存放目标文件，清单文件和源程序文件的驱动器标识符，通常为A，B，C，D。若省略指存放在现行驱动器。

文件名1，文件名2，文件名3分别是目标文件名，清单文件名和源程序文件名。

.REL，.PRN，.FOR分别为上述文件的文件类型。命令行中文件类型也可以省略。

简单而实用的命令格式是：

=源文件名

它编译源文件，并生成与源文件同名的目标文件。

编译过程中，逐渐显示出编译的程序名，（缺少程序名，系统会以 MAIN 补缺）及编译时所检查出的错误。无错误显示表示编译成功。最后打入CTRL-C退出编译，返回CDOS。

上述过程如下：

```
A.FOR                          ; 调用编译程序

VERS. 03.21
* =XX                          ; 打入源文件名
$MAIN                          ; 开始编译

* CTRL-C                       ; 退出编译
A.                             ; 返回CDOS
```

上述过程也可以用：

FOR=源文件名

的命令行方式打入，编译后无论成功与否，都自动返回CDOS。

其过程如下：

```
A.FOR␣=XX                      ; 打入命令行
$MAIN

A.                             ; 返回CDOS
```

3-2-3 连接运行 程序经编译成为目标文件，还要经连接装配才能形成可执行的程序。

首先调用连接程序LINK.COM，打入：

LINK

出现提示符“*”后，打入连接命令，其命令格式如下：

d:文件名1.REL，d:文件名2.REL，…/开关项

其中：

d:为目标文件所在的驱动器标识符，若省略则系统约定为现行驱动器。

文件名为目标文件名。

.REL为目标文件的文件类型，可省略。

开关项用来指定装配过程结束时的操作，常用的开关项有：

E——完成连接后退出LINK，返回CDOS。退出前显示出：

〔xxxx yyyy zz〕

xxxx：执行程序的内存起始地址。

yyyy：指出程序的结束地址。

zz：程序占有的页数(256个字节为1页)。

G——连接完成后打出如下信息：

〔xxxx yyyy zz〕

〔BEGIN EXECUTION〕

并立即执行程序，程序运行完才返回CDOS。

使用开关项E连接后，接着打入：

SAVE d:文件名.COM zz

则建立一个可执行的命令文件。以后只要打入此文件名，就可运行此程序。

上述过程如下：

```
A.LINK                          ; 调用连接程序

LINK VERSION 03.21
* XX/G                          ; 打入连接命令

〔xxxx yyyy zz〕                 ; 结束连接
〔BEGIN EXECUTION〕              ; 开始运行程序

(显示运行结果)
A.                              ; 返回CDOS
或者：
A.LINK                          ; 调用连接程序
LINK VERSION 03.21
* XX/E                          ; 打入连接命令

〔xxxx yyyy zz〕                 ; 结束连接
A.SAVE B:XX.COM zz              ; 存入磁盘

A.                              ; 返回CDOS
```

上述过程也可以使用命令行形式：

LINK 目标文件1，目标文件2，……/G

LINK 目标文件1，目标文件2，……/E

3-2-4 修改程序 无论在编译或连接运行中出现错误，都必须重新进入编辑状态。修改后的程序需重新进行编译，连接操作。

首先调用文本编辑程序EDIT，进入编辑状态后，可用各编辑命令进行显示，修改，删除，插入等操作。进行各种编辑操作时，要时刻记住“字符指针”的位置，尽管它不出现在屏幕上。

下面介绍几个常用的编辑命令。

显示：

B#T　　显示全部程序，指针停留在程序的第一行。

±nP　显示由当前页向后(+)或向前(−) n页的整页 (23行) 内容。0P显示当前页。显示后指针停留在本页第一行。

±n　显示从当前行向后或向前的第n行的内容，指针同时移动n行。0显示当前行。省略值为1，即直接压 RETURN 键，可以一行一行的显示，同时指针一行一行的下移。

±nT　显示出当前行向后或向前n行的全部内容，而指针不动。

删除：

±nD　从指针位置向后或向前删除n个字符。

±nX　从指针位置向后或向前删除n个字。

±nK　从指针位置向后或向前删除n行。

n：表示删除数量，省略值为1

替代：

一个新字符串替代另一个老字符串用S命令。

命令格式：

±nS〈老字符串〉 ESC 〈新字符串〉

命令首先从指针当前位置向后或向前查找第n次相遇的老字符串，删去后，插入新的字符串。

如果新字符串省略，则相当于一个字符串的删除作用。

插入：

插入新的内容用I命令。

注意：插入前要清楚指针所在的位置。新插入的内容总是插在指针的前面。

程序修改后，用E命令退出EDIT，系统将修改后的程序存入原文件说明的文件，而把未经修改的程序存入文件类型为.BAK的后备文件。

假如待修改的文件名为XX.FOR，其修改过程如下：

```
A.EDIT XX.FOR                    ; 调用编辑程序

CROMEMCO TEXT EDITOR
VERSION 00.10
* N                               ; 用N命令把老文件
END OF INPUT FILE                   调入文本缓冲区
*
(进行修改操作)

* E                               ; 退出编辑
GOOD BYE
END OF INPUT FILE

A.                               ; 返回CDOS
```

（详细的文本编辑命令请看3-1-1内容）

3-2-5 错误信息

一、FORTRAN 编译错误信息

FORTRAN IV 编译程序查出两种类型的错误：警告性的和致命性的错误。

警告性错误以百分号(%)打头，致命性错误以问号(？)打头。

对两种类型的错误，都应修改，不能运行编译有错的程序。

在"%"或"？"后有编译行号就打印编译行号，否则打印物理行号，接着打印错误信息文本。

例子：

？ Line 25: Mismatched Parentheses (括号不配对)

% Line 16: Missing Integer Varible (缺整型变量)

1.致命性错误：

代码	错误信息
100～112	同2-4-5
113	Data Pool Overflow (数据库上溢)
114～116	同2-4-5
117	* * * Reserved for Futtre Use * * * (保留给将来使用)
118～135	同2-4-5

2.警告性错误：

代码	错误信息
0～30	同2-4-5
31	Missing RETURN in Subprogram (在子程序中缺返回语句)

二、连接中的错误信息

遇到非法操作时，连接程序给出两种类型的错误信息：致命性的和警告性的。致命性的错误用问号(？)打头，警告性的错误用百分号(%)打头。对两种错误都应更正，再做连接。

1.致命性错误：

错误信息	含义
？No Start Address (无启动地址)	使用/G开关项，但没有装配主程序模块。
？Loading Error (装配错误)	连接和装配的目的文件格式不正确。
？Fatal Table Collision (致命的表冲突)	没有足够的内存来装配给定的程序
？Command Error (命令错误)	LINK命令不正确。再打正确的命令重新连接。
？File Not Found (文件没有找到)	命令串中的某文件不存在。此错误常常是由于用户忘记指定驱动器代号，而LINK只在当前驱动器寻找而造成的。

2.警告性错误：

错误信息	含义
%2nd COMMON Larger/XXXXXX/	第一个定义的公用块 XXXXXX 不是最大的。包含大的公用块的模块需要首先连接，这样LINK 就可为数据存贮区分配适当数目的字节 。可

(第二个公用块较大)　　重新安排模块的装配次序或改变公用块的定义。

% Mult.Def.Global　　在连接过程中（内部）符号YYYYYY遇到一次以上的定义。
YYYYYY

(全局量YYYYYY多重定义)

三、运行时错误信息

1.致命性错误：

代码	含义
ID	非法的FORMAT描述符
F0	FORMAT的场宽是0
MP	在FORMAT中缺句点
FW	FORMAT场宽太小
IT	输入/输出传送错误
ML	在FORMAT中缺左括号
DZ	实数或整数被0除
LG	LOG函数自变量非法（负的或0）
SQ	SQRT函数自变量非法(负的)
DT	数据类型与FORMAT说明中不一致
EF	在读时遇文件结束标志EOF

2.警告性错误：

代码	含义
IB	超过输入缓冲区极限值
TL	在FORMAT里左括号太多
OB	超过输出缓冲区极限值
DE	十进制指数溢出。在输入流中数的指数部分大于99
IS	整数太大
BE	二进制指数溢出
IN	输入记录太长
OV	算术溢出
CN	转换溢出。在实型向整型转换时
GL	计算GOTO数太大
GS	计算GOTO数太小
SN	SIN的变量太大
A2	ATAN2的二个变量都是0
IO	非法的输入输出操作
BI	在二进制输入/输出时，超过了缓冲区长度
RC	在FORMAT中有负的重复系数

3-3　COBOL语言的使用简介

在Cromemco机上发展COBOL程序，有和FORTRAN 相同的操作过程。这一节只是讲述COBOL使用注意事项，具体的操作请参阅3-2节的内容进行。

一、发展COBOL程序至少应具备下述文件：

CDOS. COM

EDIT. COM

COBOL. COM
OVRLY1. OVR
OVRLY2. OVR
OVRLY3. OVR
OVRLY4. OVR
LINK. COM
COBLIB. REL

二、建立源程序

在调用文本编辑程序时，打入的命令格式：EDIT　文件说明

其中，文件类型用约定的COBOL文件类型. COB。

三、编译源程序

编译源程序使用COBOL. COM文件。所以应使用下述命令格式：

COBOL ~ =源文件名

四、连接运用

连接运用的过程与FORTRAN相同，请参照上节所述操作。

五、错误信息

1.编译错误信息

编译过程中的语法检查产生两种诊断错误信息。当简单的语法错误出现时，出错标记直接安置在列表文件中出错程序行的下面，并对每种出错情况假定了纠正错误的作法。编译继续进行。

复杂的诊断错误信息由二或三部分组成：

(1) 有关的程序行标号——4位数字，后跟一个冒号。

(2) "错误"的英文解释，如果这个解释是用开关项/W/开始，那么它仅仅是一个警告，如果不是/W/开始，则是一个相当严重的错误，它禁止目标程序的装配，连接和执行。

(3) 任选地列出出错处程序的内容。

简单错误信息见下表。

2.连接错误信息　由于COBOL和FORTRAN使用同一个连接程序，所以它们的连接错误信息相同，这里不再叙述。请看3-2-5内容。

3.运行时的错误信息　若编译时未发现语法错误（清单文件中不再给出错误信息），即可将程序进行连接并投入运行(通过COM命令文件形式或是通过连接开关/G的形式使程序运行)，这时还可能因程序中的逻辑设计错误而使运行中止。(逻辑错误不能通过语法检查发现)。

当运行中出现错误时，在控制台上将显示出四行信息：

＊＊RUN—TIME ERR:
错误原因（见下表所列）
程序名(PROGRAM—ID)
出错处的行号

下面列出终止运行的原因及附加说明：

REDUNDANT OPEN(多余的打开)　企图打开一个已经打开的文件。

DATA UNAVAILABLE(不可应用的数据)　企图引用一个没有打开的文件记录中的数据或已到达 " AT END" 状态的文件记录中的数据。

标　　记	出　　错　　原　　因	纠 正 错 误 作 法
"QLIT"?	非数值常字出错：①长度为0②非法继续。	假设可以接受，忽视错误继续编译
LENGTH?	非数值常字超过120个字符，或数值常字超过18个数字或COBOL字超过30个字符。	超过部分的字符被忽略
CHRCTR?	出现非法字符。	忽略错误继续编译
PUNCT?	标点用法错(例如，逗号后没有跟着空格)。	假设可以接受。
BADWORD	现行字被变形，例如，用连字符结束，或在数值常字中有多余一个的十进小数点。	忽略错误继续编译
SEQ*	序列标号出错（序列不按上升次序排列)。	接受该数，继续编译
NAME?	名字没有按字母开始。	接受该名字继续编译
PIC=X	形象字符串出错。	假设为PICX
COL.7?	一个错误字符出现在程序行的第7列中(除*-/D字符之外的字符)	假设第7列中安置空格
AREA A?	继续行的A区(8—12列)中，不是空格	忽略A区的内容

SUBSCRIPT FAULT(下标错)　下标具有非法值(例如，具有小于1的值)。

INPUT/OUTPUT (输入/输出错)　不可恢复的输入/输出错，用户程序中没有准备"AT END"短语、"INVALID KEY"短语或声明过程体来作出错处理。

NON—NUMERIC DATA(无数字的数据)

数值项的内容和关于该项的形象描述不一致。

PERFORM OVERLAP(非法嵌套　两个PERFORM语句的执行体非法嵌套。

CALL PARAMETERS(调用参数错)

主调用程序和被调用程序的数据名表中数据个数不一致。

ILLEGAL READ(非法读)　企图读一个设有按INPUT或I-O方式打开的文件。

ILLEGAL WRITE(非法写)　对顺序存取文件，企图把内存缓冲区当前内容，写到一个未按OUTPUT方式打开的文件中；对随机或动态存取文件，企图把内存缓冲区当前内容，写到一个未按OUTPUT或I-O方式打开的文件中。

ILLEGAL REWRITE(非法重写)　企图重写一个未按I-O方式打开的文件的记录。

REWRITE; NO READ(重写,未读)　最后一个输入/输出操作不是成功的读(READ)，就企图重写一个顺序存取文件的记录。

REDUNDANT CLOSE(多余的关闭)　企图关闭一个没有打开的文件。

GO TO (NOT SET)（转移，未设置）

企图执行一个仅含一个未被初始化的GO TO语句的程序段。

FILE LOCKED（文件锁闭）　企图打开一个以LOCK方式锁闭的文件。

READ BEYOND EOF(文件结束后读)　在遇到文件结尾后，还企图读下一个记录。

DELETE; NO READ（删除，未读）

最后一个输入/输出操作不是成功的读，就企图删除一个顺序存取的文件记录。

ILLEGAL DELETE(非法删除) 未以I-O方式打开相关文件，就企图删除文件记录。
ILLEGAL START(非法启动)
未以INPUT或I-O方式打开文件，就企图执行START语句。

第四章 FORTRAN、PASCAL、COBOL语言在IBM PC上的使用

如本篇前言所述，除BASIC语言以外的高级语言，其发展过程是一致的。本章所述的三种语言，都是在同一操作系统——IBM DOS磁盘操作系统支持下使用的，其过程更为相似。例如，它们都使用同一个编辑程序——EDLIN程序，建立或修改源程序；使用相同的连接程序——LINK程序，形成可执行程序。区别只是使用的编译程序不同，这是显而易见的。因此本章首先介绍常用的IBM DOS操作系统及EDLIN编辑程序。然后重点介绍FORTRAN语言的建立，编译，连接和运行。在此基础上重点指出PASCAL，COBOL与FORTRAN不同之处和使用注意事项。

4-1 IBM DOS操作系统简介

IBM DOS是IBM个人用磁盘操作系统，目前已有很多版本，例如：DOS 1.00，DOS 1.10，DOS 2.00其功能是向上兼容的，后出的版本总要扩充一些功能，但对内存的需求也相应增加，因此使用时必须注意硬件的配置。

DOS的核心部分驻留在DOS盘上，由以下四个程序组成：

1.boot record(引导记录)。启动DOS时，引导记录自动地被装入内存，以后由它将DOS的其余部分装入内存。

2.IBMBIO. COM程序：输入/输出设备管理程序。

3.IBMDOS. COM程序：包括文件管理程序和用户可以使用的一系列服务功能程序。

(以上三部分在列磁盘目录时，不被列出)

4.COMMAND. COM 程序：命令处理程序，用于接受输入的命令，并运行相应的程序。

DOS盘上的所有程序均在DOS控制下运行，用户通过适当的命令来使用系统的各种功能。

4-1-1 DOS命令及命令格式

一、DOS命令分类

DOS命令按其功能分为：

1.文件的比较、复制、显示、删除和文件的更名；

2.磁盘的格式化和磁盘复制；

3.执行系统程序，例如：EDIT编辑程序；

4.输入时间，日期与注释；

5.置各种各样打印机和显示屏幕的选择项；

6.磁盘文件的后备和恢复；

7.其它系统程序。

按其驻留方式分为两种类型：

内部命令

外部命令

任何具有文件类型.COM和.EXE 的文件，被称为外部命令。外部命令驻留在磁盘上，因此在使用时，相应的磁盘应插在驱动器内。

内部命令常驻于内存。

二、DOS命令格式说明

一般DOS命令由命令关键字，命令域，任选项和参数构成。其一般格式：

命令关键字〔/任选项〕〔filespec/任选项……〕〔参数〕

其中：

命令关键字是DOS所提供的命令，即指定要执行的某一种操作。

filespec——文件说明，指明命令的处理对象，它的一般格式如下：

d:filename.ext

其中：

d:_设备标识符。如A:代表系统上第一个驱动器，B:代表第二个 驱动器。如果省略，那么DOS指定为当前的（或约定的）驱动器。系统设备名指派如下：

保　留　名	设　　备
CON：	控制台的键盘/屏幕显示。如果当作输入设备使用时，可以按F6键，按 Enter键产生文件结束指示，结束 CON:作为输入设备
AUX： or COM1：	第一个异步通信适配器端口。
LPT1： or PRN：	并行打印机（只当作输出设备）。
NUL：	不存在的（假的）设备。当作输入设备时，立即产生文件结束。当作输出设备时，模拟写操作，但实际上没有写任何数据。

注：1.可以用保留设备名代替文件名。

2.忽略用这些设备名输入的任一驱动器的区分符或文件名的扩展名。

3.在保留设备字后的冒号是任选项。

filename——文件名。文件名是1～8个字符的字符串，可使用的字符有：A—Z，0—9，$，&，#，@，!，%，'，(，)，-，<，>，{，}，—，/，∧，～，｜，‘，除此之外任何其它字符都无效。

.ext——文件类型。可选的文件类型是由一个“.”（小数点）和1～3个字符的字符串组成。可使用的字符与文件名filename一样。

任选项和参数限定DOS命令的某些功能，具体请看有关命令的描述。

有关DOS命令的使用规定

使用DOS命令中，必须遵循如下的约定：

1.关键字必须用大写字母表示。

2.方括号(〔 〕)中的项目是选择项。命令中的括号键不需要打入，只需打入括号内的信息。

3.省略号（……）表示把某项按要求重复多次。

4.除方括号外，命令中的各种标点，如逗号，等号，问号，冒号或正斜线都是必要的，

不允许忽略。

三、DOS命令的格式和功能

各种命令的格式及其功能说明如下：

批处理(Batch)

批处理是执行指明的或约定的驱动器盘上，所指定的批文件所包含的命令。

批文件是包括一个或多个的命令文件，其文件类型必为. BAT。批文件可以使用行编辑(EDLIN)或COPY命令建立。

格式：[d:]filename[parameters]

类型：内部

注：AUTOEXEC. BAT文件

AUTOEXEC. BAT 文件是一个特殊的批文件，当启动 DOS 时，命令处理程序检索AUTOEXEC. BAT文件，如果该文件在DOS盘上，那么DOS总是自动地执行该文件。

CHKDSK命令

CHKDSK命令分析指定的或约定的驱动器盘上的目录和文件分配表，并产生盘和存贮器状态报告清单。

格式：CHKDSK[d:]

类型：外部

CHKDSK暂时用d:指定的驱动器作为约定的驱动器。如果CHKDSK 过早地结束，那么约定的驱动器变成CHKDSK曾检查过的驱动器。如果用A 回答盘错误信息，那么可能过早地结束CHKDSK。

盘检查后，显示某错误信息，后面是状态报告清单。

COMP命令

COMP命令将一个文件的内容同另一个文件的内容进行比较。

注：该命令比较两个文件的内容，而DISKCOMP命令是比较两个整盘文件的内容。

格式：COMP[filespec] [d:] [filename] [.ext]

类型：外部

COMP 一个接一个字节地比较两个文件。不相等的字节会导致错误信息，这些错误信息给出不相等的16进制偏移值以及不相等的字节。

COPY

COPY 命令执行设备之间的文件传送，它可以把一个或一个以上的文件有选择地复制到另外的盘上，如果需要，还可以给复制文件赋以需要的名字。

将文件复制到同一盘上时，必须给复制文件以不同的文件名，否则不准使用COPY命令。

在复制过程中，还可以完成文件的连接和在某些系统设备之间传送数据。

格式：Copy [/A] [/B] filespec [/A] [/B]
[d:] [fiename [. ext]] [/A] [/B] [/V]
或
Copy [A/] [/B] filespec [/A] [/B] [+filespec [/A] [/B]]
[d:] [fileenam [. ext]] [/A] [/B] [/V]

类型：内部

把filespec标识的文件，传送到[d:] 所标识的设备上，其文件名为：[filename[. ext]]

/V参数使DOS检验写在目标d:盘上的扇区记录是否正确。

/A和/B参数指定由 COPY 命令处理的数据量，/A和/B的每一个既适用于它前面的那个filespec也适用于它后面所有filespec，直到遇到另外的/A或/B为止。这些参数具有下列的含义：

当同要传送的filespec一起使用时：

/A使该文件象ASCII（文本）文件一样对待。复制文件的数据一直到文件中找到了文件的第一个结束字符（X '1A'或Ctrl-Z码）为止，但不包含该字符。文件中的其余部分不复制。

/B复制整个文件。（以目录文件大小为准）。

当同复制的目标filespec一起使用时：

/A加上Ctrl-Z字符作为文件的最后结束字符。

/B不加文件结束字符（Ctrl-Z）。

当完成连结时，未指明的值是/A（参看下面的格式3），而当不完成连结时是/B(参看格式1和2)

可以在源文件和复制文件的文件名和类型参数中，使用替代字符？和＊。

COPY命令有三种格式选择：

格式1：复制的文件同被复制的文件具有相同的文件名和文件类型。

COPY filespec

或

COPY filespec d:

格式2：复制的文件与被复制的文件具有不同的名字。

COPY filespec filename[. ext]

或

COPY filespec d:filename [. ext]

格式3：复制连结文件，即可以将另一些文件加到第一个文件的末尾，将两个或两个以上的文件组合成一个文件。在结果文件目录中记录的日期和时间是当前的日期和时间。显示的复制文件数指的是建立结果文件个数。

为了连结文件，在COPY命令中列出由加(＋)号分开的源文件。使用如下格式：

COPY [/A] [/B] filespec [/A] [/B] [＋filespec [/A] [/B]]

[d:] [filename [.ext]] [/A] [/B] [/V]

例如：

```
COPY A. XYZ＋B. COM＋B:C. TXT BIGFILE. TXT
COPY A. COM＋B. COM
```

DATE命令

用于输入日期或修改日期，在建立或改动某些文件时，日期会自动记录在目录项目中。

格式：DATE [mm-dd-yy]

类型：内部

如果和DATE 命令一起，输入了一个有效日期，那么系统就接受这个新日期并出现系统提示符，否则DATE命令发出下列的提示符：

```
Current date is day mm-dd-yy
```

Enter new date: _

按mm-dd-yy或mm/dd/yy形式输入新日期。此处：

mm是1—12之中的1位或2位数。　　（表示月份）

dd是1—31之中的1位或2位数。　　（表示日）

yy是80—99（假定是19）之中的两位数。

或者是1980—2099之中的4位数。　　（表示年）

DEL命令

查看本节中“ERASE命令”

DIR命令

DIR命令列出所有目录项，或只列出指定文件的目录项。在显示中对每个文件提供的信息包含文件的10进制字节大小，以及文件最后写入的日期。

注：对于系统文件IBMBIO.COM和IBMDOS.COM即使指出来也不列出目录项。

格式：DIR [d：] [filename [. ext]] [/P] [/W]

类型：内部

/P：当屏幕显示满一幕时，/P参数使显示暂停。当需要继续列目录时，则须按任一键。

/W参数产生宽的目录显示，它只列出文件名，每个显示行包含5个文件名。该参数只用于80列的显示上）。

在文件名和文件类型中，可以使用替代字符？和＊。

DIR命令有两种格式选择（/P和/W参数可同两种格式中的任何一个一起使用）：

格式1

使用本格式列出目录中的所有文件。例如：

DIR 或 DIR d：

格式2

使用本格式是为了从目录中列出被选择的文件。例如：

DIR filename. ext 或 DIR d：filename. ext

DISKCOMP 命令

用于将第一个指定的驱动器盘上的内容同第二个指定的驱动器盘上的内容进行比较。

注：该命令比较两个整盘的内容，而COMP命令只比较两个文件的内容。

格式：DISKCOMP [d：] [d：] [/1]

类型：外部

/1参数强迫DISKCOMP命令只比较盘的第一面，即使盘和驱动器是双面的也是一样。

DISKCOMP命令按一道接一道方法比较全部40个磁道，如果磁道内容不等，则发出信息。该信息包含被找到内容不相等的磁道号(0—39)和面号(0面或1面)。

DISKCOPY 命令

用于将源驱动器盘上的内容复制到目的驱动器盘上。

格式：DISKCOPY [d：] [d：] [/1]

类型：外部

格式中第一个d：是源驱动器，第二个d：是目的驱动器

/1参数使DISKCOPY命令不管盘或驱动器的类型如何而只复制盘的第一面。

ERASE命令

用于从指定的驱动器中，删除具有指定文件名的文件。如果不指定任何驱动器，则从约定的驱动器中删除指定的文件。

格式：ERASE filespec

或

DEL filespec

EXE2BIN 命令

将. EXE文件转换成. COM文件。可节省盘空间，并较快地装入程序。

格式：EXE2BIN filespec [d:] [filename [. ext]]

类型：外部

FORMAT 命令

用于初始化磁盘，以便记录为DOS可接受的格式；也用检查分析磁盘，以便找出有缺陷的磁道。

格式：FORMAT [d :] [/S] [/1]

类型：外部

如果在 FORMAT 命令中，指定/S参数，那么操作系统的文件也从约定的驱动器盘上按下列顺序复制到新盘上：

IBMBIO. COM

IBMDOS. COM

COMMAND. COM

如果指定/1，那么不管驱动器的类型如何，格式化的目标盘只作为单面盘用。

注：格式化破坏盘上原先的数据。

MODE命令

用于打印机或为连到彩色/图形监督适配器上的显示器设置工作方式，为异步通信适配器设置选择参数，或使打印的输出规定发送到异步通信适配器上。

格式：MODE [LPT#:] [n] [,m] [,T]

或

MODE COMn: baud [,parity [,databits [,stop bits [,p]]]]

或

MODE LPT#: = COMn

类型：外部

注：错误的或无效的n或m参数不改变工作方式。

MODE命令有4种格式选择：

格式1（适用于打印机）

MODE LPT#: [n] [,m]

此处：

#号为1，2或3（打印机号）

n为80或132（每行字符数）

m为6或8（每英寸纵向空间行数）

例如：

MODE LPT1：132，8

格式2（适用于连到彩色/图形监督适配器的显示器）

注：本命令不影响连到IBM单色显示器和打印机适配器的显示器。

MODE [n] [,m] [,T]

此处：

n为40或80（每行的字符数）。

m为R或L（把显示向右或向左移动）。

T请求一个测试图案来调整显示。

格式3（适用于异步通信适配器）

MODE COMn：baud [,parity [,databits [,stopbits [,p]]]]

此处：

n　　1或2（异步通信适配器号）

baud　　110，150，300，600，1200，2400，4800，或9600

格式4（把并行打印机的输出改道到异步通信适配器上）。

MODE LPT#：=COMn

此处：

1，2或3（打印机号）

n 1或2（异步通信适配器号）

PAUSE命令

可以使系统暂停运行，并发出Strik a key when ready……信息。

如果结束处理，则按Ctrl-Break键。如果继续处理，则按任一其他键。

格式：PAUSE [remark]

类型：内部

REM命令

用于显示批文件内的附注。（附注可以是长达123个字节的任何字符串。）

格式：REM [remark]

类型：内部

RENAME（或REN）命令

更改磁盘文件的文件名和类型。

格式：REN [AME] filespec filename [. ext]

类型：内部

将第一个参数中指定的文件名改变成第二个参数中给出的文件名和扩展名。可以在参数中使用替代字符？和＊。

SYS（系统）命令

用于将操作系统文件，按下述顺序从约定的驱动器上，传送到指定的驱动器上：

IBMBIO. COM

IBMDOS. COM

格式：SYS d：

类型：外部

被指定的驱动器盘，必须已经用FORMAT d:/s命令格式化，以得到分配给系统文件所

需要的特定的盘位置。

TIME命令

使用TIME命令可为系统输入或修改时间。

格式：TIME [hh：mm：ss. xx]

类型：内部

注：如果用TIME命令输入一个有效时间，那么该时间就被接收，并且出现系统提示符，否则TIME命令发出下列提示符：

Current time is hh：mm:ss. xx

Enter new time:

此处：

hh　是0—23中的1或2位数（代表时）。

mm　是0—59中的1或2位数（代表分）。

ss　是0—59中的1或2位数（代表秒）。

xx　是0—99中的1或2位数（代表1秒%）。

TYPE命令

可以将指定文件内容显示在屏幕上。

格式：TYPE filespec

类型：内部

四、DOS命令一览表

为便于查阅DOS命令，将上述命令列于下表：

注：I代替内部，而E代替外部。

命　令	类　型	用　途	格　式
(Batch)	I	执行批文件	[d：] filename [parameters]
CHKDSK	E	检查盘并报告状态	CHKDSK [d：]
COMP	E	比较文件	COMP [filespec] [d：] [filename [.ext]]
COPY	I	复制文件	COPY [/A] [/B] fileepec [/A] [/B] [d：] [filename [.ext]] [/A] [/B] [/V] 或 COPY [/A] [/B] filespec [/A] [/B] [+filespec [/A] [/B] [d：] [filename [.ext]] [/A] [/B] [/V]
DATE	I	输入日期	DATE [mm—dd—yy]

续表

命　　令	类　型	用　　途	格　　式
DIR	I	列出文件名	DIR [d：] [filename [.ext]] [/P] [/W]
DISKCOMP	E	比　较　盘	DISKCOMP [d：] [d：] [/1]
DISKCOPY	E	复　制　盘	DISKCOPY [d：][d:] [/1]
ERASE	I	删除文件	ERASE filespec 或 DEL filespec
EXE2BIN	E	将•EXE 文件转换成•COM 格式文件	EXE2BIN filespec [d：] [filename [.ext]
FORMAT	E	格　式　化　盘	FORMAT [d：] [/s] [/1]
MODE	E	对打印机/显示器设置工作方式	MODE [LPT# ：] [n] [, m] [, T] 或 MODE COMn：baud [, parity] [, databits [stopbits [, p]]] 或 MODE LPT#：=COMn
PAUSE	I	提供系统等待	PAUSE [remark]
REM	I	显　示　附　注	REM [remark]
RENAME	I	改　文　件　名	REN [AME] filespec filename [.ext]
SYS	E	传送DOS文件	SYS d：
TIME	I	输　入　时　间	TIME [hh：mm：ss.xx]
TYPE	I	显示文件的内容	TYPE filespec

4-1-2　EDLIN编辑程序

一、功能

EDLIN行编辑程序用来建立、修改和显示源程序或文本文件。

EDLIN是一个行文本编辑程序，该程序可用作：

1.建立新的源程序，并且保存它们。

2.修改已存在的老文件，并将修改过的以及原始的文件保存起来。

3.删除、编辑、插入和显示程序行。

4.检索、或在一个和一个以上的行内替换文本。

由EDLIN建立的文本行的长度是可变的，每行最长允许253个字符。

在编辑的过程中，由EDLIN动态地产生和显示行号，但在被保存的文件中，实际上并不存在行号。

当插入一些行时，被插入文本后面的所有行号自动地增加被插入的行数。当删除一些行时，被删除文本后面的所有行号，自动地减少被删除的行数。因而行号从1到最后总是连续的。

二、常用的编辑命令

所有的EDLIN命令都只能在EDLIN提示符“*”下打入，也只有按过ENTER 键之后，命令才能有效。(ENTER键或称为进入键，在IBM PC 机为↵键。) 命令的结束 按 Ctrl-Break，暂停显示可按Ctrl-Numlock键，再按任一键后可显示新的内容。

为了更好地理解编辑命令格式，现将编辑命令参数列表如下：

参　数	定　义
line	表示必须指定行号。 该参数有三种输入的方法： 1. 输入1～65529之中的十进制整数。如果指定的行号大于内存中已有的行号，那么就将这行加到已存在的最后行后面。 2. 输入符号(*)来指定内存中最后行的下一行。输入*号同指定的行号大于内存中已有行号具有相同的作用。 3. 输入一个句号(.)来指定当前行。 当前行表示最后修改文件的位置，但不显示最后行。行号和该行文本的第一个字符之间是由星号(*)标志当前行。例如： 10：*FIRST CHARACTER OF TEXT
n	表示必须指定行数。 输入写到盘上的行数或从盘上读出的行数。 只有用写行(Write lines)和增补(Append Lines)行命令时，要编辑的文件超出内存的能力，才用这些命令。
string	表示必须输入一个或一个以上的字符来代表要找到的、替换的、删除的文本，或者被替换的文本。 只有用检索(Search Text)和替换(Replace Text)命令时，才使用这个参数。

常用的编辑命令格式和用途如下：

增补行命令 (Append Lines Command)

用途：将盘上指定的行数加到内存中正编辑的文件上。这些行被加在内存中当前行的末尾。

格式：[n]A

删除行命令 (Delete Lines Command)

用途：将指定范围内的所有行删除。

格式：[line] [,line] D

编辑行命令 (Edit Line Command)

用途：允许编辑文本行，必须输入要编辑行的行号，或输入一个句号（.）来表示当前行。

格式：[line]

结束编辑命令 (End Edit command)

用途：结束EDLIN，并保存被编辑过的文件。

格式：E

插入行命令 (Insert Lines Command)

用途：将待插入的文本行插在指定行的前面。

格式：[line] I

列行命令 (List Lines command)

用途：显示指定范围内的所有行。当前行保持不变。

格式：[line] [, line] L

退出编辑命令 (Quit Edit Command)

用途：在不保存任何修改内容的情况下，退出编辑。

格式：Q

替换文本命令 (Replace Text Command)

用途：用第二个字符串替换指定行范围内出现的所有第一个字符串。

注：如果省略了第二个字符串，那么替换文本命令就删除指定行范围内出现的所有第一个字符串。每次修改行时，就将被修改的行显示出来，最后被修改的行变成当前行。

格式：[line] [, line] [?] Rstring [<F6>string]

(F6是键盘上的一个功能键)

检索文本命令 (Search Text Command)

用途：为了定位某个指定的字符串，对指定的行范围进行检索。

格式：[line] [, line] [?] S string

写行命令 (Write Lines Command)

用途：从存贮器被编辑过的行中，将指定的行数写到盘上，被写入的行从行号1开始。

格式：[n] W

为了查阅，提供下列表格。

EDLIN 命 令 表

命　　令	格　　式
Append Lines	[n] A
Delete Lines	[Line] [,Line] D
Edit Line	[Line]
End Edit	E
Insert Lines	[Line] I
List Lines	[Line] [, Line] L
Quit Edit	Q
Replace Text	[Line] [, Line] [?] Rstring [<F6>String]
Search Text	[Line] [,Line] [?] S string
Write Lines	[n] W

4-1-3 常用的DOS控制键和编辑键

一、控制键

当输入命令或编辑某程序时，须使用一些控制键。表中指定两个键的地方，例如Ctrl-Break键，必须按住第一个键，然后再按第二个键。

控 制 键 一 览 表

控　制　键	功　　能
[←┘]	这是Enter(进入)键，一旦按了Enter 键，被显示的行就发送给请求的程序。
[Ctrl] + [Break]	结束(删除)当前的操作。
[Ctrl] + [Enter]	允许您进到屏幕上的下一个显示行，以便继续打入某行。
[Ctrl] + [NumLock]	暂停系统操作。再按任一字符键来恢复操作。这对产生大量的屏幕输出时非常有用。可以按Ctrl-NumLock来暂停输出显示，以便有足够的时间观察它。
[Ctrl] + [Prtsc]	这些键在给打印机以及屏幕发送显示输出时，作为接通/断开开关用。可以按这些键在打印机上打印显示输出，然后再按它们停止在打印机上打印显示输出。注：当运行盘和高级BASIC时，禁止使用这个功能。
[ESC]	删除当前行，并移到下个显示行，显示反斜线(\)表示被删除的行。
[⇧] + [Prtsc]	将屏幕上当前显示的内容复制到打印机上。事实上这是屏幕的“抽点打印”(Snapshot)。
[←—]	光标从屏幕上左移并删除一个字符。(该键在Numlock键的左边，而不是数字Key的键4。)

二、编辑键

使用DOS编辑键是用来纠正输入的命令以及输入的文本行。

DOS编辑键用于行内编辑，而行编辑(EDLIN)程序则用于对某个文件或文本内的行进行操作。当调用EDLIN程序工作时，想在某一行内进行编辑，就可以使用DOS编辑键。

当按Enter键时，从键盘上输入的某行被保留在输入缓冲区中，该行可供程序使用，同时也可作为编辑的样板行。DOS编辑键根据该行的副本进行操作。利用DOS编辑键，可以复制或修改该行，也可以输入一个完整的新行。

DOS 编 辑 键 及 功 能

DOS 编 辑 键	功 能
Del	在样板行中跳过一个字符，但光标不移动。
Esc	删除当前显示的行，但样板行保持不变。
F1 或 →	从样板行中复制一个字符，并显示它。
F2	复制指定字符之前的所有的字符。
F3	将样板行中的所有剩余字符复制到屏幕上。
F4	跳过所有的字符，直到指定的字符。(F4与F2相反)。
F5	为了继续编辑而接受某个编辑行——当前显示的行变成样板行，但它不发送给请求的程序。
Ins	允许在某行内插入一些字符。

4-2 FORTRAN语言的使用

4-2-1 FORTRAN程序开发的环境 为在IBM PC上成功地运行程序，一般需要：

二个磁盘驱动器

一台打印机

一台显示器

内存容量：

FORTRAN—至少128K内存

根据需要配备$5\frac{1}{4}$英时软盘，作为用户盘。

DOS系统盘一块。

此外，还需备有自己的主盘，供编译，连接使用。

FORTRAN有三个主盘，分别叫FOR1，FOR2，LIBRARY，各主盘包括：

FOR1—FOR1.EXE

FOR2—FOR2.EXE

LIBRARY—FORTRAN．LIB

FORTRAN．ARF

LINK．EXE

建议尽快复制上述各种磁盘，作为主盘的备用盘片，日常操作应使用复制盘。

为了使用方便，希望把COMMAND．COM 从DOS磁盘中复制到下列主盘上：

FOR1

FOR2

以便使用上述主盘后，可以重新装入COMMAND·COM程序。（系统每次启动时，此程序是自动装入的）

4-2-2 系统的启动与关闭

一、系统的启动

将DOS系统盘插入A驱动器，依次接通显示器、打印机及主机电源。按提示要求输入正确日期和时间。系统显示下述信息和系统提示符（详细过程可参阅本篇1-4节）：

The IBM Personal Computer DOS

Version 1.10(c) Copyright IBM Corp 1981, 1982

A>

A>是DOS提示符，表示已经完成了起动DOS的步骤。每当A>的出现，系统总是等待输入DOS命令。

二、驱动器的转换

提示符中的A是当前使用的(default)驱动器。为了寻找输入的文件名，DOS总是检索装在当前的驱动器中的盘，除非指定了另一个驱动器。

通过打入新的标志字母后跟冒号，可以改变当前使用的驱动器。

例如：

A>　（原先的提示符）

A>B:　（新驱动器标志符）

B>　（新的提示符）

现在，B是当前的驱动器。为了寻找输入的某些文件，DOS检索装在驱动器B 中 的 盘，除非指定另一个驱动器。

三、新盘格式化和复制主盘

如果是第一次使用IBM PC，现在可进行新盘初始化，复制主盘或COPY文件的工作（具体操作见4-1节DOS操作命令一节）。

注意：DOS支持单面和双面盘驱动器两者的任何组合形式——驱动器不一定 是 同 一 类型。

单面格式化过的盘可以用于单面或双面盘驱动器中。然而，双面格式化过的盘只可用于双面驱动器中，因为两面都记录了数据并且分配空间的方法不同。因此，决不能把双面盘用于单面驱动器中。

四、关闭系统

关闭系统最好是取出磁盘后，再关闭打印机、主机及显示器电源，以防破坏磁盘信息。

4-2-3 建立源程序

建立源程序可以使用行编辑(EDLIN)程序(有关EDLIN命令见4-1-2节)。其步骤如下：

首先把已初始化的用户盘插入B驱动器，打入命令：

EDLIN B:文件名.FOR

进入行编辑。如果指定的文件不存在，显示出下述的信息和提示符：

New file

*

“*”是EDLIN的提示符。

在“*”之后，打入命令：

I

系统进入插入状态，现在可按照第二篇所述的语法规定，输入要建立的程序。

注意，现在打入的每一字符(有用的或误打的)都被视为插入的内容。在压ENTER键后，变成程序的一部分。因此在按ENTER键之前发现错误，可以用←退回，重新打入正确的内容。已经按了ENTER键，则必须退出插入状态后，才能进行修改。

程序输入完毕，可按Ctrl + Break控制键退出插入，返回编辑。

使用列表命令L显示程序，检查是否有错，如有错要进行修改(有关修改操作，请看程序修改一节)。

使用命令E退出编辑。退出编辑时，将源程序按照进入编辑时打入的文件说明，用指定的文件名和文件类型，存入指定的磁盘中，最后显示出带有当前驱动器标志的提示符A>。

假如准备建立的源程序名为myfile .FOR编辑源程序的过程如下：

```
A>EDLIN  B:MYFILE . FOR        ; 调入EDIT
New file
*I                             ; 进入插入状态
1:  (插入程序正文)
2:
:
:
:Ctrl + Break                  ; 结束插入
*L                             ; 列出程序

      (显示程序正文)

*E                             ; 退出编辑
A>                             ; 返回DOS
```

现在打入命令行：

DIR B:

以便查看在B驱动器中的用户盘上是否已建立了所指定的MYFILE.FOR文件。

4-2-4 编译FORTRAN源程序 对不同语言的源程序，编译的内部过程是不同的，在

IBM DOS系统上FORTRAN需要经过两次编译。

一、FORTRAN的第一次编译

将驱动器B:转换为当前驱动器。并把包括源程序在内的用户盘插入B:驱动器，而把FOR1主盘插入A驱动器。打入：

A:FOR1

调用编译程序。片刻，编译程序显示以下的信息和提示：

Source filename〔.FOR〕:

注意：

1.如果用户没有选择文件类型，则括弧内所示的文件类型为约定文件类型。

2.当用户没有提供文件类型时，IBM将提供一个约定文件类型，当用户明确地指定一个新的文件类型时，则取代约定文件类型。

3.约定的磁盘驱动器为当前使用的驱动器（在这种情况下为驱动器B:），如果文件说明中包括驱动器，则指定的驱动器代替约定驱动器。

Source filename 是用户已经存贮的源程序文件名。例如，用文件名 Myfile 来回答提示，则屏幕显示出：

Sourse filename〔.FOR〕:myfile

源程序文件名输入之后，会出现新的提示：

Object filename〔MYFILE .OBJ〕:

Object filename是将要建立的目标（机器能读的）文件名。如果准备用MYFILE.OBJ来存贮目标文件，只需按Enter键；如果想给目标文件取另外的名，一定要加上文件类型.OBJ。

例如，用按Enter键来回答，则显示为：

Object filename〔MYFILE .OBJ〕:

屏幕接着显示出：

Source listing〔NUL.LST〕:

Source listing是编译后产生的源列表文件。如果不要列表文件，则按Enter键。这样使用约定文件名NUL.LST，实际是告诉编译程序，不要建立源列表文件。

注意：当编译及调试程序时，无须每次都要列表文件。错误信息通常会显示在屏幕上(同时也会出现在列表文件中)。使用Ctrl-Prtsc或Ctrl-P键，可以打印出一份包含错误信息的清单。

例如，假定我们需要一个列表文件，打入文件名myfile，显示如下：

Source listing〔NUL. LST〕:myfile

编译程序将产生列表文件MYFILE.LST。

最后的提示是：

Object Listing〔NUL.COD〕:

Object Listing是编译后产生的目标列表文件。如不需要，则按Enter键，使用约定的文件名NUL.COD，实际上告诉编译程序不要建立目标列表文件。

假如，打入myfile，表示需要列表文件：

Object Listing〔NUL.COD〕:Myfile

则编译程序产生列表文件：MYFILE.COD

对于上述例子，若使用文件名myfile，则整个过程如下：

Source filcname 〔. FOR〕: myfile

Object filename 〔MYFILE.OBJ〕:

Source Listing 〔NOL.LST〕: myfile

Object Listing 〔NOL.COD〕: myfile

当回答了最后提示，按了Enter键，编译程序便开始执行。如果源程序中包含任何语法错误，编译程序将错误信息显示在屏幕上，同时也出现在列表文件中。

上述的编辑命令，也可以使用命令行，其格式如下：

FOR1 Source File, Object File, Source List, Object List;

使用命令时若以“；”号结束，则不再显示上述的编译程序提示。如果用户输入一个不完整的命令行且没有分号，则编译程序为剩余的未指定文件作出提示。如果输入一个不完整的命令行且又在最后加了分号，则未指定文件取约定的文件名，而且不再作出提示。

例如：

命令行	作用
MYFILE;	根据MYFILE.FOR文件编译源程序。虽然没有逗号，该命令产生文件MYFILE.OBJ。
MYFILE,;	该命令与上一例的命令作用相同。
MYFILE,,;	根据MYFILE.FOR文件编译源程序并产生文件MYFILE.OBJ和文件MYFILE.LST。
MYFILE,,CON;	根据MYFILE.FOR文件编译源程序并显示出程序列表。目标程序为MYFILE.OBJ。
MYFILE,MYOBJ,PRN;	根据MYFILE.FOR文件编译源程序，打印出程序列表，并把目标码存于MYOBJ.OBJ。
A:MYFILE,MYOBJ,A:;	编译磁盘A上的MYFILE.FOR文件并把目标码存于MYOBJ.OBJ，把列表存于驱动器A中的MYFILE.LST。

从上例可以看出，最简单可行的命令行是：

A:FOR1 文件名;

当编译结束时，屏幕显示出以下信息。

无错显示为：

Pass one No Errors Detected

10 Source Lines

若有错，将显示错误信息及以下的一个或二个信息：

Pass one 3 Errors Detected

10 Source Lines

如果编译程序发现到错误，必须在第二次编译之前在源程序中找出并解决这些问题。(错误信息在4-2-7给出)

二、FORTRA的第二次编译

在通过第一次编译以后，可以开始准备第二次编译。第二次编译使用的主盘是FOR2。

首先从A驱动器上取出FOR1主盘，插入FOR2主盘。然后打入：

A:FOR2

当FOR2文件调入后，便自动开始运行。它会产生目标文件和列表文件清单，编译完成以后，FOR2给出以下类似信息：

Code Area Size = ＃0116(278)

Cone Area Size = ＃005E(94)

Data Area Size = ＃000E(14)

Pass two No Errors Detected

Code Area Size是程序所占字节总数，Cons Area Size是程序中常数所占的字节数，Data Area Size表示静态分配的数据。三种数据的大小均以十六进制和十进制形式给出。

若第二次扫描出错，应检查这些错误，如果必要的话，则重新运行FOR1和FOR2。

如不发生错误，则FORTRAN的编译全部完成，可进行连接和运行。

4-2-5 连接运行 连接程序使用LIBRARY主盘。

首先从A驱动器取出FOR2主盘，插入LIBRARY主盘，然后打入：

A:LINK

命令启动连接程序，并给出以下的提示：

object Modules〔.OBJ〕:_

要求输入目标文件名。目标文件名可以不用文件类型.OBJ。例如：

Object Modules〔.OBJ〕: myfile

下一个提示是：

Run File〔MYFILE.EXE〕:

提示输入可执行文件的文件名，此文件将保存连接后形成的代码，文件类型约定为.EXE。按Enter键表示使用约定的文件名和类型，本例为MYFILE.EXE。

下一个提示为：

List File〔NUL.MAP〕:_

按Enter键使连接程序选用约定的文件名NUL.MAP。实际上是不必建立MAP文件。

假如我们要建立一个MAP文件，并键入myfile，显示如下：

List File〔NUL.MAP〕: myfile

连接程序自动增加文件类型。MAP。

下一个提示是：

Libraries〔. LIB〕:

Libraries里含有IBM FORTRAN程序运行时所需要的子程序。所有子程序都包括在FORTRAN。LIB中。

此时，可以按Enter键来响应这个提示：

Libraries〔. LIB〕:

当连接一个FORTRAN程序时，FORTRAN库就自动引入。

如果使用上面的文件名，所有显示的提示如下：

B>A：LINK

IBM Pesonal Computer Linker

Version 1.10(C)Copyright IBM Corp 1982
object Modules〔.OBJ〕: myfile

Run File〔MYFILE.EXE〕:
List File〔NUL.MAP〕: myfile

Libraries〔.LIB〕:

现在开始连接，连接之后在B驱动器盘上生成名为MYFILE.EXE的可执行文件。

连接完成之后，只需打入可执行的文件名（不必带文件类型.EXE），就可运行程序。例如打入：

myfile

即可运行上例的程序。

4-2-6 修改程序 当编译或连接运行中发生错误，首先要根据系统显示的错误信息，认真的检查出错原因，必要时启动打印机，将源程序文件或列表文件打印出来。

操作过程如下：

一、从A驱动器中取出编译或连接主盘，插入DOS系统盘。

二、调用编辑程序，打入命令行：

A:EDLIN filespec

按上面的叙述，源程序应存于B驱动器中用户盘上，所以命令行是：

A:EDLIN B:MYFILE.FOR

三、在出现编辑程序提示符"*"后，使用各种编辑命令，如显示，删除，替代和插入的命令进行修改。

使用编辑命令时，要注意当前行所在的位置，以免发生误改。

下面介绍几种常用的编辑命令：

显示：用L命令

n1，n2L	显示n_1—n_2范围内的所有行
nL	显示从n行开始的23行内容
L	显示以当前行为中心的23行内容。

L命令不改变当前行位置。

删除：用D命令

n1，n2D	删除n1—n2范围内的所有行
nD	删除第n行（不是n个行）
D	删除当前行

每删除一次，EDLIN会重新安排一次行号。

替代：用R命令

n1，n2R老字符串<F6>新字符串　在指定的n1—n2范围内，用新字符串替代老字符串。

最后发生替代的行变成当前行。

如省略新字符串，实际上完成的是删除老字符串的操作。

插入：用I命令

nI	在指定的第n行前插入

.I或I　　在当前行前插入

#I　　在最后行之后插入

四、修改也可以使用行编辑命令。

首先输入要编辑行的行号。假定要修改的是第6行，打入：

6

屏幕显出该行的内容，例如：

6: This is sample unedited line

6:*_

头一行是准备编辑行的内容，称为样板行，后一行表示等待行编辑命令。使用编辑功能键进行删除，复制，插入等项操作。有关编辑功能键的使用见4-1-3节。

修改后，按Enter键保存被修改的内容。

五、退出EDLIN编辑程序

退出EDLIN编辑有两种操作：

1.结束编辑，用结束编辑命令E。

E命令结束编辑，并将修改后的程序按照调用编辑程序时的文件说明，用指定的文件名和文件类型，存入指定的磁盘中。同时将原始文件（未修改的文件）用.BAK的文件类型，重新命名，成为后备文件。

2.在不保存任何修改的情况下，退出编辑用Q命令。

上述过程全部显示如下：

```
B> A:EDLIN B:MYFILE.FOR

*L

(显示的程序行)

*(用编辑命令修改程序)

*E

B>
```

4-2-7 FORTRAN的错误信息 错误是由编译程序或运行时系统发现的。错误信息包括出错信息和代码。编译程序发现的错误分成FOR1前端错误和FOR2后端错误。运行时系统发现的错误分成文件系统错误和其它错误。

一、编译错误信息

1.前端错误

前端错误信息含有一个标记。在前端错误信息中，有一些被称为“模糊”错误，它给出这样出错误信息：

? ERROR:error message（错误信息）

这类错误可能发生的原因是：

(1) 指定的文件名出错——文件名语法错误

(2) 装配溢出错误——磁盘已满

(3) 指定的文件找不到——没有发现文件
(4) 堆栈溢出——编译程序超出存储区
(5) 堆栈已满——编译程序超出存储区
(6) 超出内存范围
(7) 实型运算溢出
上述情况下，由于没有给出错误行号，因而很难确定出错的位置，以便修改错误。
下面列出前端错误的出错信息和代码。

1 Fatal error reading source block. 读到源程序块的致命错误。
2 Nonnumeric characters in label field. 在标号区里出现非数字字符。
3 Too many continuation lines. 续行太多。对于每个起始行的续行不得超过9行。
4 Fatal end of file encountered.
遇到非正常的文件末端。例如：缺END语句或END语句后多出一行或多行。
5 Labeled continuation line. 续行带标号
6 Missing field on $ compiler metacommand. 在$编译元命令行上缺域。
7 Cannot open file. 不能打开文件。
8 Unrecognizable metacommand. 不能识别的元命令。
9 Input file invalid format. 输入文件格式无效。
10 Too many nested include files. 文件含嵌套太多。
11 Integer constant overflow. 整常数溢出。
12 Real constant error. 实常数错误。
Incorrect representation of real constant. 实常数表示法不对。
13 Too many digits in constant. 常数的位数太多。
14 Identifier too long. 标识符太长。
15 Character constant not closed. 字符常数没有括起来。
例如：缺少撇号或字符常数超过了72列。
16 Zero length character constant. Zero length character constants not allowed.
零长度字符常数。零长度字符常数不可接受。
17 Invalid character in input. 输入中含有无效字符。
18 Intger constant expected. 要求整常数。
19 Label expected. 要求标号。
20 Label error. 标号错误
21 Type expected, for example: in IMPLICIT statement.
要求类型，例如：在IMPLICIT语句中。
22 Integer constant expected. 要求是整常数。
23 Extra characters at end of statement. Characters encuntered after expected end of line. 在语句末端有多余的字符。在所希望的行末端之后出现字符。
24 " ("expected. 要求是"("。
25 Letter already used in IMPLICIT. 在IMPLICIT中字母已经用过。
26 ") "expected. 要求是")"。
27 Letter expected. 要求是字母。

28 Identifier expected. 要求是标识符。
29 Dimension(s) expected. 要求是维数。
30 Array already dimensioned. 数组已定维。
31 Too many dimensions. 维数太多。
32 Incompatible arguments. 不相容的变量。
33 Identifier already has type. 标识符已赋于类型。
34 Identifier already declared. 标识符已说明。
35 INTRINSIC FUNCTION not allowed here.
这里不允许INTRINSIC FUNCTION.
This INTRINSIC FUNCTION not allowed as an argument.
这个INTRINSIC FUNCTION 不允许作为变量。
36 Identifier must be a variable. 标识符必须是变量。
37 Identifier must be a variable or the current FUNCTION.
标识符必须是变量或现行FUNCTION。
38 "/"expected. 要求是"/"。
39 Named COMMON block already saved. 指定的COMMON块已保存。
40 Variable already appears in a COMMON.
变量已经在COMMON语句中出现过。
41 variables in two different COMMON blocks.
变量出现在两个不同的COMMON块中。
42 Number of subscripts conflicts with declaration. 下标与数组说明相矛盾。
43 Subscript out of range. 下标超出范围。
44 Forces two calls to the same location. 迫使两次调用同一存贮单元。
45 Forces location in negative direction. 迫使存贮单元向着相反方向。
EQUIVALENCE以相反方向扩展COMMON块。
46 Forces location conflict. 迫使存贮单元冲突。
47 Statement number expected. 希望是语句标号。
48 CHARACTER and numeric items in same COMMON block.
CHARACTER与数字项在同一个COMMON块中。
49 CHARACTER and noncharacter item conflict.
CHARACTER与非字符项冲突。
50 Invalid Symbol in expression. 表达式里出现无效符号。
51 SUBROUTINE name in expression. 在表达式中出现SUBROUTINE名。
52 INTEGER or REAL expected. 希望是INTEGER或REAL。
53 INTEGER REAL or CHARACTER expected.
希望是INTEGER, REAL或CHARACTER。
54 Types not compatible. 类型不一致
55 LOGICAL expression expected. 要求是逻辑表达式。
56 Too many subscripts. 下标太多。
57 Too few subscripts. 下标太少。

58 Variable expected. 要求是变量。

59 "="expected. 要求是"="。

60 Size of CHARACTER items must agree. CHARACTER项的大小要一致。

61 Assignment types do not match. 赋值类型不匹配。

62 SUBROUTINE name expected. 希望是SUBROUTINE名。

63 Dummy parameter not allownd. Formal parameter not allowed in COMMON statement. 不允许哑元。在COMMON语句中不允许有形参。

64 Dummy parameter not allowed. Formal parameter not allowed in EQUIVALENCE statement.
不允许哑元，在EQUIVALENCE语句中不允许出现形式参数。

65 Assumed size declarations only for dummy arrays.
假定的大小说明只对哑数组。

66 Adjustable size declarations only for dummy arrays.
可调大小的说明只能对哑数组。

67 Assumed size must be last dimension. 假定的大小应是最后一个维数。

68 Adjustable bound must be parameter or in COMMON.
可调的界必须是参数或在COMMON中。

69 Adjustable bound must be simple integer variable.
可调的界必须是简单整型变量。

70 More than one main program. 主程序多于一个。

71 Size of named COMMON must agree. 被命名的COMMON的大小必须一致。

72 Dummy arguments not allowed. 不允许哑变元。

73 COMMON variables not allowed. 不允许COMMON变量。
COMMON variables not allowed in DATA statement.
在DATA语句里不允许有COMMON变量。

74 SUBROUTINE, FUNCTION, or INTRINSIC names not allowed.
不允许SUBROUTINE, FUNCTION或INTRINSIC名。

75 Subscript out of range. 下标超过范围。

76 Repeat count must be>=1. 重复计数必须>=1。

77 Constant expected. 要求常数。

78 Type conflict. 类型冲突。

79 Number of variables does not match. 变量个数不匹配。

80 Label not allowed. 不允许标号。

81 No such INTRINSIC FUNCTION. 没有这样的INTRINSIC FUNCTION。

82 INTRINSIC FUNCTION type conflict. INTRINSIC FUNCTION类型冲突。

83 Letter expected. 希望是字母。

84 FUNCTION type conflict with previous call.
FUNCTION类型与前面调用的冲突。

85 SUBROUTINE/FUNCTION already defined.
SUBROUTINE/FUNCTION已经定义过。

87 Argument type conflict. 自变量类型冲突。
88 SUBROUTINE/FUNCTION conflict with previous use.
SUBROUTNE/FUNCTION与前面使用相冲突。
89 Unrecognizable statement. 不可识别的语句。
90 CHARACTER FUNCTION not allowed.
不允许CHARACTER FUNCTION。
91 Missing END statement. 缺少END语句。
93 Fewer actual arguments than dummy arguments in call.
在调用中实元的个数比哑元少。
94 More actual arguments than dummy arguments in call.
在调用中实元的个数比哑元多。
95 Argument type conflict. 变元类型冲突。
96 SUBROUTINE/FUNCTION not defined. SUBROUTINE/FUNCTION没定义。
98 CHARACTER size invalid. 无效的CHARACTER长度。
100 Statement order. 语句顺序。
Statement out of order. 语句混乱。
101 Unrecognizable statement. 不可识别语句。
102 Jump into block not allowed. 不允许跳入模块。
103 Label already used for FORMAT. 标号已经用于FORMAT语句。
104 Label already defined. 标号已定义过。
105 Jump to format not allowed. 不允许跳到格式语句。
106 DO statement not allowed here. 这里不允许有DO语句。
107 DO label must follow DO statement. DO标号必须跟在DO语句后面。
108 ENDIF not allowed here. 这里不允许ENDIF。
109 Matching IF missing. 缺少匹配的IF。
110 Improperly nested DO block in IF block.
在IF程序块中不恰当的嵌套DO程序块。
111 ELSEIF not allowed here. 在这里不允许ELSEIF。
112 Matching IF missing. 缺少匹配的IF。
113 Improperly nested DO or ELSE block. 不恰当嵌套DO或ELSE程序块。
114, 115 "("expected. 希望是"("。
116 THEN expected. 希望是THEN。
117 Logical expression expected. 要求是逻辑表达式。
118 ELSE not allowed here. 在这里不允许ELSE。
119 Matching IF missing. 缺少匹配的IF。
120 GOTO not allowed here. 这里不允许GOTO。
121 GOTO not allowed here. 这里不允许GOTO。
122 Block IF not allowed here. 这里不允许IF块程序。
123 Logical IF not allowed here. 这里不允许逻辑IF。
124 Arithmetic IF not allowed here. 这里不允许算术IF。

125 “, ”expected.　　希望是“,”。
126 Expression of wrong type.　　错误类型的表达式。
127 RETURN not allowed here.　　这里不允许RETRUN。
128 STOP not allowed here.　　这里不允许STOP。
129 END not allowed here.　　这里不许END。
131 Label not defined.　　标号没定义。
132 DO or IF block not terminated.　　DO或IF块没有终止。
133 FORMAT not allowed here.　　这里不允许FORMAT。
134 FORMAT label already referenced.　　FORMAT标号已经引用过。
135 FORMAT label missing.　　缺FORMAT标号。
136 Identifier expected.　　希望是标识符。
137, 166 Integer variable expected.　　希望是整型变量。
138 TO expected.　　希望是TO。
139 , 143, 148. 156 Integer expression expected.　　希望是整型表达式。
140 ASSIGN statement missing.　　缺少ASSIGN语句。
141 Unrecognizablc character constant.　　不可识别的字符常数。
142 Character constant expected.　　希望是字符常数。
144 STATUS option expected.　　希望是STATUS选择。
145 Characrer expression not allowed.　　不允许字符表达式。
预期的字符表达式是错误类型的表达式。
146 FILE=missing.　　缺少FILE=。
147 RECL=already defined.　　RECL=已经定义过。
149 Unrecognizable option.　　不可识别的选择。
150 RECL=missing.　　缺少RECL=。
151 Adjustable arrays not allowed here.　　这里不允许可调的数组。
152 End of statement encountered in implied DO, expressions begining with “(”not allowed as I/O list elements.
在隐含的DO中没遇到语句结束，用“(”开始的表达式不允许作为I/O表元素。
153 Variable required as control for implied DO.
需要变量作为隐含DO的控制。
154 Expressions not allowed in I/O list.
在I/O列表中不允许有表达式。
155 REC=option already defined.　　REC=选择项已定义过。
157 END=not allowed here.　　这里不允许END=。
158 END=alraedy defined.　　END=已经定义过。
159 Unrecognizable I/O unit.　　不可识别的I/O设备。
160 Unrecognizable format in I/O.　　不可识别的I/O格式。
161 options expected after”, ”.　　在“, ”后希望是选择。
162 Unrecognizable I/O list element.　　不可识别的I/O表元素。
163 FORMAT not found.　　找不到FORMAT。

164 ASSIGN missing. 缺少ASSIGN.

165 Label already used as FORMAT. 标号已用于FORMAT。

167 Label defined more than once as FORMAT.
FORMAT的标号的定义多于一次。

203 CHARACTER FUNCTION not allowed. 不允许CHARACTER FUNCTION。

406 Unit zero must be formatted and sequential.
单元零必须是被格式化的，序列的。

407 ERR=arleady defineed. ERR=已经定义。

408 Too many labels. 标号太多。

409 Invalid size for this type. 对于这种类型大小无效。

411 Integer type conflict. 整型类型冲突。

415 DIMENSION too big. DIMENSION太大。

420 Invalid FUNCTION call. 无效FUNCTION调用。

421 Invalid INTRINSIC FUNCTION. 无效的INTRINSIC FUNCTION。
类型转换，词汇关系和选择的最大最小值的内部函数名，不能作为实元。

501 Unrecogeizable character. 不可识别的字符。

502 Blank not allowed in metacommand. 在元命令中不允许有空格。

503 Metacommand not allowed here. 这里不允许元命令。

504 Size already defined. 大小已定义。

601 Out of range. 超出范围。

701 CHARACTER type expected. 希望是CHARACTER类型。内部错误。

703～708，710，711 Internal error. 内部错误。

709 CHARACTER type not expected. 不期望的CHARACTER类型。

713 Long integer conversion error. 长整数转换错误。

2.后端错误

后端给出两种类型的错误：用户错误和内部错误。

发现用户和内部后端错误，会导致编译立即停止，并给出一个错误标号。

(1) 后端用户错误

1.企图用0作除数。例如：A DIV 0。

2.当整常数折合时溢出。例如：

Maximun + A + Integer

3.表达式太复杂/太多的内部标号。试图用赋于中间值来分解表达式。

(2) 后端内部错误

这些错误的格式如下：

***Internal Error NNN

NNN是内部错误标号，它的范围在1～999之间。当出现一个内部错误时，程序还会继续执行一段。

二、运行出错信息

1.文件系统错误

在运行时，错误可以分成文件系统错误和其他错误。首先描述文件系统错误。文件系统

错误范围从1000到1299。

文件系统错误有如下格式：

error type error in file filename

错误代码

其中error type是下述类型之一。

Operation	操作
Filename	文件名
Device full	设备溢出
File not found	文件找不到
File not open	文件没打开
Line too long	行太长
Data format	数据格式

文件系统错误代码和信息

1000 Write error when writing end of file. 当写文件末端时发生写错误。

1002 Filename extension with more than 3 characters.
文件类型多于3个字符。

1003 Error during creation of new file (disk/directory full).
当建立新文件时出错(磁盘/目录已满)。

1004 Error during open of existing file (file not found).
当打开存在文件时错(文件找不到)。

1005 Filename with zero or more than 8 characters.
文件名用0个或多个8个字符。

1007 Total filename length over 21 characters.
总的文件名长度超过21个字符。

1008 Write error when advancing to next record.
当推进到下一个记录时写错误。

1009 File too big (over 65,535 logical sectors).
文件太长(超过65,535逻辑段)。

1010 Write error when seeking to direct record. 当查找直接记录时写错误，

1011 Attempt to open a random file to a non—disk device.
企图在非磁盘设备上打开随机文件，

1012 Forward space or backspace on a non—disk device.
在非磁盘设备上前进或退回，

1013 Disk or directory full error during forward space or back space.
前进或退回时磁盘或目录已满，

1200 Format missing final")". 格式缺少最后的")"。

1201 Sign not expected in input. 在输入中不希望的符号。

1202 Sign not followed by digit in input. 在输入中数字后面不能跟符号。

1203 Digit expected in input. 在输入中希望是数字。

1204 Missing N or Z after B in format. 在格式里，B后面缺少N或Z。

1205 Uuexpected character in format. 在格式里，出现不希望的字符。

1206 Zero repetition factor in format not allowed.
在格式里，不允许有零重复因子。

1207 Integer expected for w fieid in format. 在格式里，w域希望是整型。

1208 Positive integer required for w field in format.
在格式里，w域要求是正整数。

1209 "·"expected in format. 在格式里要求"·"。

1210 Integer expected for d field in format. 在格式里，d域希望是整数。

1211 Integer expected for e field in format. 在格式里，e域希望是整数。

1212 Positive integer erquired for e field in format.
在格式里，e域要求是正整数。

1213 Positive intger required for w field in A format.
在A格式里，w域要求是正整数。

1214 Hollerith field in format must not appear for reading.
在读入格式里，不应该出现Hollerith域。

1215 Hollerith field in format requires repetition factor.
在格式里，Hollerith域需要重复因子。

1216 X field in format requires repetition factor. 在格式里，X域要求重复因子。

1217 P field in format requires repetition factor.
在格式里，P域需要重复因子。

1218 Integer appears before +or - in format. 在格式里，整数出现在 +或 - 之前

1219 Integer expected after +or - in format.
在格式里，要求整数在 +或 - 之后。

1220 P format expected after signed repetition factor in format.
在格式里，要求P格式在符号重复因子之后。

1221 Maximum nesting level for formats excecded. 超过格式最大嵌套层数。

1222 ") " has repetition factor in format. 在格式里，") "有重复因子。

1223 Integer followed by ", " invalid in format.
在格式里在整数后面有无效的", "。

1224 ". " is invalid format control character. ". "是无效的格式控制字符。

1225 Character constant must not appear in format for reading.
在输入格式里，不应该出现字符常量。

1226 Character constant in format must not be repeated.
在格式里的字符常数不能重复。

1227 "/" in format must not be repeated. 在格式里"/"不能重复。

1228 "? " in format must not be repeated. 在格式里"? "不能重复。

1229 BN or BZ format coutrol must not be repeated.
BN或BZ格式控制不能重复。

1230 Attempt to perform I/O on unknown unit number, for example,
OPEN statement missing or never executed.

企图在未知设备号上执行I/O。例如，缺少OPEN语句或者从未执行OPEN 语句。

1231 Formatted I/O attempted on file opened as unformatted.
企图在按非格式化打开的文件上，执行格式化I/O。

1232 Format fails to begin with"(". 格式没有以"("打头。

1233 I format expected for integer read. 整数输入希望是I格式。

1234 F or E format expected for real read. 对实型输入，要求是F格式或E格式。

1235 Two ". " characters in formatted real read.
在输入的格式化实数中含有两个". "。

1236 Digit expected in formatted real read. 在输入的格式化数中希望是数字。

1237 L format expected for legical read. 在读逻辑量中希望是L格式。

1238 Blank logical field. 空逻辑域

1239 T or F expected in logical read. 在读逻辑量中希望是T或F。

1240 A format expected for character read. 读字符量希望是A格式。

1241 I format expected for integer write. 写整型量中希望是I格式。

1242 w field in F format not greater than d field +1
在F格式里的w域不大于d +1。

1243 Scale factor out of range of d field in E format.
在E格式里，比例因子超出d的范围。

1244 E or F format expected for real write. 写实型量希望是E或F格式。

1245 L format expected for logical write. 写逻辑量希望是L格式。

1246 A format expected for character write. 写字符希望是A格式。

1247 Attempt to do unformatted I/O to a unit opened as formatted.
企图在按格式化打开的设备上执行非格式化I/O。

1251 Integer overflow on input. 在输入上整数溢出。

1252 Not enough input to satisfy I/O list/format, for example, specifying an il0 format and only inputting 5characters.
没有足够的输入数据以满足I/O表的格式，例如，一个I10格式却只输入5个字符。

1253 Too many bytes written to direct access unit record.
将太多的字节写入直接存取单元记录。

1255 Attempt to do external I/O on a unit beyond end of file record.
企图在文件记录末端之外的单元上执行外部I/O。

1256 Attempt to position a unit for direct access on a nonpositive record number.
企图在非正记录号上，置一个直接存取单元。

1257 Attempt to do direct access to a unit opened as sequential.
企图对按顺序方式打开的设备执行直接存取。

1258 Unable to seek to file position. 禁止查找文件位置。

1260 Attempt to backspace unit connected to printer or keyboard/display.
企图回退连接到打印机或键盘/显示器的单元。

1264 Attempt to do unformatted I/O to internal unit.
企图对内部单元执行非格式化I/O。

1265 Attempt to put more than one record into internal unit.
企图写多于一个的记录到内部单元。

1266 Attempt to write more characters to internal unit than its length.
企图写到内部单元的字符多于其长度。

1267 EOF called on unknown unit. 在未知单元上调用EOF。

1268 Dynamic file allocation limit exceeded. 动态文件超过存贮单元量。

1270 Console I/O error. 控制台I/O错。

1271 EOF function called on printer or keyboard/ display.
在打印机或键盘/显示器上调用EOF函数。

1272 File operation attempted after error encountere on previous operation.
在前面的操作错误后企图做文件操作。

1273 Keyboard buffer overflow:too many bytes
written to keyboard input record (must be less than 132).
键盘缓冲器溢出：太多的字节写到键盘输入记录(必须小于132)。

1274 Error while reading long integer. 当读入长整数时错误。

1275 Error while wrting long integer. 当写长整数时错误。

1297 Integer variable not currently assigned a format label.
整型变量当前没有指定格式标号。

1298 End of file encountered on read with no END= option.
在未带有END=选择的输入中，迂到文件末端。

1299 Integer variable not ASSIGNed a label used in assigned GOTO.
在赋值GOTO中，不是用整型变量作标号。

2.其它运行错误

从2000到2999是非文件系统错误代码的范围。在一般情况下，可发现元命令是否错误。下面给出元命令控制下的检测：

2000—2049存贮错误

堆是存贮区城。IBM Fortran系统用它来动态地分配存贮区。当栈和堆相互向上建立时，这些错误也伴随着它们。例如：栈溢出就引起"堆无效"错误。

2000 Overflow. 溢出。当调用一个过程或函数时，栈超过存贮空间。

2001 No Room In Heap. 在堆中无空间。

2002 Heap Is Invalid. 堆无效。

2050—2099整形运算

2052 Signed Divide By Zero. 有符号的数用零除。
整数值用零除；如果有$DEBUG则检查。

2054 Signed Math Overflow. 带符号算术溢出。
INTEGER结果超出最大范围；如有$DEBUG则检查。

2084 Integer zero to Negative Power; always checked.
整数零用负幂；总被校验。

2100—2149类型REAL运算

2100 REAL Divide By Zero. 实数用零除。

2101 REAL Math Overflow. 实数运算溢出。

2102 SIN Or COS Argument Range. SIN 或 COS 变量范围。

2103 EXP Argument Range. EXP 变元范围。EXP 内变元太大，结果超出表示范围。

2104 SQRT of Negative Argument. 负数的 SQRT。
平方根函数的变元小于零；总被死锁。

2105 LN of Non—Positive Argument. 非正变元的 LN；死锁。

2106 TRUNC/ROUND Argument Range. TRUNC/ROUND 变元范围。
转化的实型超过整型范围。总被死锁。

2131 Tangent Argument Too Small. 正切函数的变元太小；死锁。

2132 ARCSIN Or ARCCOS of REAL>1.0 REAL 的 ARCSIN 或 ARCCOS>1.0。

2133 Negative Real To Real Power. 负实数的实数幂；死锁。

2134 Real Zero to Negative Power; always checked. 实数零用负幂；死锁。

2200—2249 长整数运算

2200 Long Integer Divided by Zero; always checked. 长整数用零除；总被校验。

2201 Long Integer Math Overflow; always checked. 长整数运算溢出；总被校验。

2234 Long Integer Zero to Negative Power; always checked.
长整数零用负幂；总被校验。

2250—2999 其他错误

2451 Assigend GOTO Label Not In List. 在表中无赋值 GOTO 标号。

4-3 PASCAL 语言的使用

在 IBM PC 上发展 PASCAL 程序，和发展 FORTRAN 程序有相同的环境，基本相同的操作过程。这节主要指出PASCAL在使用上与FORTRAN 不同之处，详细的操作见上一节内容。

4-3-1 主盘与文件 供PASCAL编译，连接使用的主盘是PAS1，PAS2，PASCAL.LIB。各主盘包括下述文件：

PAS1——PAS1. EXE
PASKEY
FILKQQ. INC
FILUQQ. INC
ENT6XS. ASM

PAS2——PAS2. EXE

PASCAL. LIB——PASCAL. LIB
——PASCAL
——LINK. EXE

为了使用方便建议将系统盘上的 COMMAND. COM 文件复制到：

PAS1
PAS2

4-3-2 建立和编译源程序 建立和编译源程序的过程和FORTRAN 完全相同。注意事项有以下两点：

1. 建立源程序时，打入的命令格式是：

EDLIN B：文件名. PAS

.PAS是PASCAL源程序的约定文件类型。

2.两次编译分别使用PAS1和PAS2主盘，编译命令格式分别是：

A:PAS1（或A:PAS1文件名；）

A:PAS2

4-3-3 连接运行

在打入命令 A:LINK以后，屏幕连续显示：

Objecst Modules:

Run File:

List File〔文件名.MAP〕:

在回答上述提示后，屏幕接下显示：

libraries〔 〕:

libraries（程序库）指PASCAL运行程序所需的运行例行程序。所有这些程序都在PASCAL. LIB中。可用按Enter键或打入：

A:PASCAL. LIB

来回答，两者作用相同。

接下的五个提示是：

Publies〔NO〕:

Line Numbers〔NO〕:

stack size〔Obzect file stack〕:

Load Low〔Yes〕:

DSALLocation〔NO〕:

方括号内是系统的约定值。我们建议全部用按Enter键来回答。

如果使用FORTRAN中的文件名MYFILE，所有屏幕显示如下：

B>A:LINK

IBM Personal Computer Linker

Version 1.00(c) Copyright IBM Corp 1981

Object Modules:myfile

Run File:myfile

List File〔MYFILE. MAP〕:

Libraries〔 〕:

Publies Numbers〔NO〕:

Line Numbers〔NO〕:

Stack Size〔Obzect file stack〕:

Load Low〔Yes〕:

DSALLocation〔NO〕:

现在开始连接用户程序，连接后生成一个PASCAL的可执行文件。本例中生成MYFILE.EXE文件。

连接完成后，只需打入可执行的文件名（文件类别可省略）即可运行程序。例如打：myfile。

其它操作请看DOS和FORTRAN的使用。

4-3-4 错误信息 错误可被编译程序发现，也可能在运行时由系统发现。编译时的错误分成前端（第一次编译）错误和后端（第二次编译）错误；运行时的错误分成文件系统错误和其它错误。

一、编译错误信息

1.前端错误

前端错误包含一个代码和一个信息。多数的前端错误含有一串破折号和一个表示出错位置的箭头，但有三个前端错误（128，129，130）是出现在$SYMTAB列表区的体后。前端能够修复大多数的错误，而发出一个警告信息，但少数称为"危险的"错误是不可修复的。这些危险错误给出如下信息：

Compiler Cannot Continue

危险错误是在下列条件下产生的：

(1) 已经超出$ERRORS所设置的错误计数。

(2) 在不希望地方出现文件结束。

(3) 标识符作用域嵌套太深。

(4) 无法找到关键字PROGRAM，MODULE，或者IMPLEMENTATION。

(5) 无法找到标识符PROGRAM，MODULE，IMPLEMENTATION。

信息前面的"Warning"表示前端所生成的中间代码文件是正确的，但状况不佳或视为不安全的，其它信息表明实际的错误。

错误信息"Compiler"指内部的一致性检验失败，无论编译的是什么源程序，都不应有这些信息出现。

下面列出前端错误的出错信息和代码。

101 Invalid Line Number（无效行号） 行号大于32767；源程序的行太多。

102 Line Too Long Truncated（行太长，截尾）

103 Identifier Too Long Truncated（标识符太长，截尾）

104 Number Too Long Truncated（数字太长，截尾）

105 END Of String Not Found（没有找到字符串尾）
在找到闭合引号之前，该行已经结束。

106 Assumed String（假定的字符串）
采用双引号来装字符串；使用单引号。

107 Unexpected End Of File（不希望的文件末端）
用数、元命令等来结束文件（扫描时）。

108 Meta Command Expected Command Ignored（忽略元命令所希望的命令）以$开头的注释后没有标识符。

109 Unknown Meta Command Ignored（忽略未知的元命令）
元命令的标识符是未知的，或是无效的。

110 Constant Identifier Unknown Or Invalid Assumed Zero(未知的或无效的常量标识符，假设为零)
把元命令设定给一个常量标识符（如$DEBUB:A），标识符是未知的或不是正确类型的常量。

112 Invalid Numeric Constant Assumed Zeto（无效的数值常量认为是零）

把元命令设定给数值常量，常量的格式错误或超出范围。

113 Invalid Meta Value Assumed Zero（无效的元值，认为是零）

对常量或标识符都不设置元命令。

114 Invalid Meta Command（无效的元命令）

元命令后所希望的是+，－或：。

115 Wrong Type Value For Meta Command Skipped（给元命令的值类型错误，跳过此命令）

116 Meta Value Out of Range Skipped（元值超出范围，跳过此命令）

$LINESIZE 整数值小于 16 或大于 160。

117 File Identifier Too Long Skipped（文件标识符太长，跳过）

118 Too Many File Levels（文件层次太多）

119 Invalid Initialize Meta command（无效的初始化元）

$POP 元命令没有对应的 $PUSH 元命令。

120 CONST Identifier Expected（希望有 CONST 标识符）

$ INCONST 元命令后没有标识符。

121 Invalid INPUT Number Assumed Zero（无效的 INPUT 值，认为零）

122 Invalid Meta Command Skipped（无效的元命令，跳过）

$IF 和它的值后面没跟有 $THEN 或 $ELSE。

123 Unexpected Meta Command Skipped（不希望的元命令，跳过）

$THEN 没有对应的 $IF 元命令。

124 Unexpected Meta Command（不希望的元命令）

元命令不在注释中；被随便处理。

125 Resered（保留）

126 Invalid Real Constant（无效的实型常量）

REAL 类型的常量是前面有小数点或后面有小数点的常量。

127 Invalid Character Skippped（跳过无效字符）

源文件的字符在程序文本中是不可接受的。

128 Forward Proc Missing（缺前向过程）

用信息表示的过程和函数已被说明，但FORWARD没找到(信息出现在$SYMTAB区中）。

129 Label Not Encountered（没遇到标号）

用信息表示的标号已被说明或已在 GOTO，BREAK 或者 CYCLE 中被使用，但没有找到（信息出现在 $SYMTAB 区中）。

130 Program Parameter Bad（不良的程序参数）

用信息表示的程序参数从没说明过或者给 READFN 的类型是错误的。（信息出现在$SYMTAB区。）

有几种原因会出现 Overflow Errors（溢出错误）：

133 Type Size Overfiow（类型长度溢出）

数据类型稳含着有多于 32766 个字节的结构。

134 Constant Memory Overflow（常量存贮溢出）

常量存贮分配超过65534个字节。

135 Static Memory Overflow（静态存贮溢出）

静态存贮分配超过65534个单元。

136 Stack Memory Overflow（堆栈存贮溢出）

静态结构存贮分配超过65534个字节。

137 Integer Constant Overflow（整型常量溢出）

INTEGER类型的带符号的常量表达式超出范围。

138 Word Constant Overflow（字常量溢出）

WORD类型或其它不带符号的常量超出范围。

139 Value Not In Range For Record（记录的值不在范围内）

记录的标记（特征）值不在变体范围内、结构常量内、长格式的NEW/DISPOSE/SIZEOF、或其它应用中。

140 Too Many Compiler Labels（编译程序的标号太多）

编译程序要使用内部标号；程序太大。

141 Compiler（编译程序）

142 Too Many identifier Levels（标识符层次太多）

143,144 Compiler（编译程序）

145 Identifier Already Declared（标识符已经说明过）

146 Unexpected End Of File（不是预期的文件末端）

在语句，说明等中结束文件（在语法分析时）。

Common Substitution Mistakes（公共替换错误）：得到自己专用信息，错误被校正，只给予警告信息。

147：Assumed =

148 = Assumed：

149：= Assumed =

150 = Assumed：=

151 〔Assumed(

152 (Assumed〔

153)Assumed〕

154 〕Assumed)

155 ;Assumed,

156 ,Assumed;

Missing Symbol（缺符号）：如果希望有特定的符号，但找不到，它可以被插入并给出下列信息：

162 Insert Symbol

163 Insert,

164 Insert;

165 Insert =

166 Insert: =
167 Insert OF
168 Insert]
169 Insert)
170 Insert[
171 Insert(
172 Insert DO
173 Insert:
174 Insert.
175 Insert. .
176 Insert END
177 Insert TO
178 Insert THEN
179 Insert *

如果希望的特定符号在某些无效的符号后面找到，那么无效的符号被删掉，并给出下列信息：

185 Invalid Symbol Begin Skip（开始跳过无效的符号）

186、187 End Skip（结束跳越）

前一错误信息用短语“Begin Skip”来结束的；该信息标志着结束源文本的跳越。

188 Section Or Expression Too Long（部分或表达式太长）

189 Invalid Set Operator or Function（无效集合运算符或函数）

190 Invalid Real Operator or Function（无效的实型运算符或函数）

191 Iuvalid Value Type For Operator or Function（给运算符和函数的数值类型是无效的）

194 Type Too Long（类型太长）

使用大于32766个字节的变量或类型。

195 Compiler（编译程序）

196 Zero Size Value（零长度值）

使用空记录“RECORD END”时认为它为长度。

197 Compiler（编译程序）

198 Constant Expression Value Out Of Range（常量表达式的值超出范围）

199 Integer Type Not Compatible with Word Type（整型类型与字类型不相容）

公共错误表示这是混淆了带符号和不带符号的算术运算：要么用WORD（ ）把带正符号的值变为不带符号的值，要么用ORD（ ）把不带符号的值（<MAXINT）改变为带符号的值。

201 TypesNot Assignment Compatible（非赋值相容的类型）

202 Types Not Compatible in Expression（表达式中类型不相容）

203 Not Array Begin Skip（非数组开始跳过）

变量后是一个左括号（或圆括号），不是数组。

204 Invalid Ordinal Expression Assumed Integer Zero (无效的有序表达式，认为是整型为零)

表达式的类型错误或是非有序类型。

205 Invalid Use of PACKED Components (无效的使用PACKED分量)

PACKED 结构的分量没有地址(它可能不在字节范围内)；它是不能靠引用来传递。

206 Not Record Field Ignored (非记录字段，不予理采)

变量后跟的是点，不是记录、地址、或文件。

207 Invalid Field (无效的字段)

记录变量和点不跟有有效的字段。

208 File Dereference Considered Harmful (文件被引用看作为有危险的)

当计算文件缓冲区变量的地址时，通常对缓冲区变量做的特殊处理（就是文本文件的缓慢求值或二进制文件的并行操作）不能进行；该地址的缓冲变量可能是无效的。

209 Cannot Dereference Value (不能被引用的值)

变量后的箭头不是指针、地址或文件。

210 Invalid Segment Address (无效的段被引用)

变量驻留在段地址中，但需要一个约定的段地址，可能需要进变量的局部复制。

211 Ordinal Expression Invalid Or Not Constant (有序表达式无效或非常量)

214 Out of Range For Set 255 Assumed (超出假设为255的集合范围)

集合常量的元素必须有小于等于255的有序值。

215 Type Too Long Or Contains File Begin Skip (类型太长或包含文件开始跳过)

结构常量必须有255或小于255字节；同时，它还不能含有文件类型 或LSTRING类型。

216 Extra Array Components Ignored (忽略额外的数组元素)

数组常量含有太多的数组类型的分量。

217 Extra Record Components Ignored (忽略额外的记录元素)

记录常量含有太多的记录类型的分量。

218 Constant Value Expected Zero Assumed (希望常量值，假定为0)

结构常量中的值不是常量。

220 Compiler (编译程序)

221 Components Expected For Type (类型要求多分量)

222 Overflow 255 Components In String Constant (字符串常量的分量溢出255)

字符串常量只能占255个或小于255个字节。

223 Use NULL (使用NULL)

必需使用的是预先说明的常量NULL，而不是两个引号。

224 Cannot Assign With Supertype LSTRING (不能用超数组类型LSTRING来赋值)

225 String Expression Not Constant (字符串表达式非常量)

用星号串接的字符串只适用常量。

226 String Expected Character 225 Assumed (字符串所希望的字符为约定值255)

字符串常量没有字符，也许使用了NULL。

227 Cannot Assign To Function (不能对函数赋值)

228 Cannot Assign To Varible (不能对变量赋值)

229 Cannot Use As CONST Parameter Or Address Assumed Zero (不能按照CONST参数或零地址来使用)

表达式没有地址和不能是参考参数。

230 Unknown Identifier Assumed Integer Begin Skip (未知的标识符假设为整型，开始跳过)

231 VAR Parameter Or WITH Record Assumed Integer Begin Skip (VAR参数或WITH记录，假设为整型，开始跳过)

无效的标识符，需要有地址。

232 Cannot Assign To Type (不能对类型进行赋值)

233 Invalid Procedure Or Function Parameter Begin Skip (无效的过程或函数参数，开始跳过)

在使用内部过程和函数中出错，诸如：

NEW或DISPOSE第一个参数不是指针变量。

长格式的NEW/DISPOSE/SIZEOF记录特征值没找到。

长格式的NEW/DISPOSE/SIZEOF超数组的边界太多。

长格式的NEW/DISPOSE/SIZEOF超数组的边界不够。

NEW或SIZEOF超数组没给出边界。

对数值上不是有序类型的值进行ORD或WRD。

对无效的值或类型进行LOWER或UPPER。

对巨型数组，文件数组，或误紧缩数组进行PACK或UNPACK。

RETYPE的第一个参数并非是类型标识符。

RESULT的参数并非是函数标识符。

234 Type Invalid Assumed Integer (类型无效，假设为整型)

READ, WRITE, ENCODE或DECODE的参数并非是类型INTEGER, WORD, REAL, BOOLEAN, 枚举及指针；或者READ和WRITE 的参数，不是类型 CHAR, STRING, LSTRING；或者READFN的参数不是类型FILE。

235 Assumed File Input (假设是文件输入)

READFN的第一个参数不是文件，所以假定是INPUT。

236 Not File Assumed Text File (非文件，假设为文本文件)

READ或WRITE (或READLN或WRITELN) 的第一个参数被假定为文件，但这种假定不对；请明显地给出INPUT或OUTPUT以避免出现该信息。

237 Assumed Input (认为是输入)

INPUT不能作为程序参数给出。

238 Assumed Output (认为是输出)

OUTPUT不能作为程序参数给出。

239 LSTRING Expected (希望的是LSTRING)

READSET、ENCODE或DECODE的目标必须是一个LSTRING。

241 Invalid Segment Variable (无效的段变量)

变量驻留在段地址中，但需要有一个约定的段地址。可能要对变量进行局部拷贝。

242 File Parameter Expected Skip Statement (希望有文件参数，跳过语句)

243 Character Set Expected (期望字符集)

READSET期望有一个SET OF CHAR的参数。

244 Unexpeted Parameter Begin Skip (不希望参数，开始跳过)

EOF, EOLN和PAGE不需要多于一个参数。

245 Not Text File (非文本文件)

EOLN, PAGE, READLN和WRITELN只适用于文本文件。

247 Invalid Function (无效的函数)

〔不能用〕

248 Size Not Identical (长度非等同)

在RETYPE时给出的警告信息；可能或不可能象预料那样执行。

249 Procedural Type Parameter List Not Compatible (过程类型参数表不相容)

形成过程参数与实在过程参数的参数表不相容；参数个数不一样，函数结果类型或参数类型不一样，或属性错误。

250 Reserved (保留的)

251 Unexpected Parameter Begin Skip (不希望参数，开始跳过)

过程或函数没有参数，但找到左括号。

252 Cannot Use Procedure or Function As Parameter (不能把过程或函数当作参数使用)

253 Parameter Not Procedure Or Function Begin Skip (参数非过程或函数，开始跳过)

这里所希望的是过程参数，需要的是过程或函数。

254 Supertype Array Parameter Not Compatible (超数组类型参数不相容)

实在参数与形式参数一样或导出的超数组类型与形式参数不一样。

255 Compiler (编译程序)

256 VAR Or CONST Parameter Types Not Cempatible (VAR或CONST的参数类型不相容)

257 Parameter List Size Wrong Begin Skip (参数表长度错误，开始跳过) 参数太少或太多；只在太多时跳过。

258 Invalid Procedural Parameter To EXTERN (给EXTERN的是无效的过程参数)

用段内调用来引用实在过程或函数，所以不能传递给外部代码段，要对过程或函数给出PUBLIC属性来解决。

259 Invalid Set Constant For Type (类型的集合常量是无效的)

260 Unknown Identifier In Expression Assumed Zero (表达式中有未知的标识符，假定为零)

表达式中的标识符还没有定义过 (拼写错误?)

261 Identifier Wrong In Expression Assumed Zero (表达式中标识符错误，假定为零)

262 Ansumed Parameter Index Or Field Begin Skip (虚构的参数索引或字段，开始跳过)

在260或261出错后，圆括号或方括号中的东西，以及标识符后的点都被跳过。

265 Invalid Numeric Constant Assumed Zero（无效数值常量，假定为零）

在假定的INTEGER（或WORD）文字常量中译码错误。

267 Invalid Real Numeric Constant（无效的实型数值常量）

在假定的REAL类型文字常量中译码错误。

268 Cannot Begin Expression Skipped（不能开始表达式，跳过）

符号不能启动表达式，所以删掉。

269 Cannot Begin Expression Assumed Zero（不能开始表达式，假定为零）

符号不能启动表达式，所以插入零。

270 Constant Overflow（常量溢出）

除数为常量0（INTEGER或者WORD）的DIV或INOD。

271 Word Constant Overflow（字常量溢出）

对WORD操作数执行一元减（试用NOT字＋1）。

272 Word Constant Overflow（字常量溢出）

WORD常量减去WORD常量得到负的结果。

275 Invalid Range（无效的范围）

子界的下界大于上界（例如：2 1）。

276 CASE Constant Expected（期望CASE常量）

希望要给CASE语句或记录变体一个常量值。

277 Value Alredy in Use（已经在使用的值）

CASE语句或记录变体中，值已经赋过了（例如CASE1:3:XXX:2:YYY:END）。

278 Reserved（保留的）

279 Label Expected（希望有标号）

在BREAK，CYCLE，GOTO语句中，或在启动一个语句时，或在LABEL部分中，所需的标号没有找到。

280 Invalid Integer Label（无效整型标号）

非十进制记数法（例如，8#77）是不允许用在标号中的。

281 Label Assumed Declared（假定标号已经说明过）

该标号没有在LABEL部分中出现过。

283 Expression Not Boolean Type（表达式非布尔类型）

IF，WHILE或UNTIL后的表达式必须是BOOLEAN的。

284 Skip To End Of Statament（跳到语句末尾）

跳过不希望的ELSE或UNTIL子句。

285 Compiler（编译程序）

286 ；Ignored（忽略；）

ELSE前的分号总是错的，跳过去。

288 ；Skipped（跳过；）

OTHERWISE后的分号总是错的，跳过去。

289 Variable Expected For FOR Statement Begin Skip（FOR语句希望有变量，开始跳过）

变量标识符必须跟在FOR后面。

291 FOR Variable Not Ordinal Or Static Or Declared In Procedure（过程中，FOR变量不是有序的，静态，或说明过的）

FOR语句控制变量不应该是：

REAL类型或其他有序类型

数组，记录或文件类型的分量

指针类型或地址类型的引用

在栈或堆中，除非局部地加以说明

非局部说明的，除非在静态内存中

引用参数（VAR或VARS参数）。

292 Skip To:= （跳到:=处）

FOR语句中，这里希望进行赋值。

293 Reserved（保留）

294 GOTO Considered Harmful（认为GOTO有危害）

要使用的是$GOTO元命令，但这里是GOTO。

296 Label Not Loop Label（标号非循环标号）

BREAK或CYCLE标号不是在FOR，WHILE，或REPEAT前。

297 Not In Loop（不在循环中）

BREAK或CYCLE语句不在FOR，WHILE，或REPEAT中。

298 Record Expected Begin Skip（希望有记录，开始跳过）

WITH语句希望有一个记录变量。

300 Label Already In Use Previous Use Ignored（标号已经在使用，以前的使用被忽略）

301 Invalid Use Of proeedure Or Function Parameter（无效地使用过程或函数参数）

过程参数当作函数使用，反之也是如此。

303 Unknown Identifier Skip Statement（未知的标识符，跳过语句）

304 Invalid Identifier Skip Statement（无效的标识符，跳过语句）

305 Statement Not Expected（非希望的语句）

一个模块或未初化的IMPLEMENTATION带有主BEGIN……END语句。

306 Function Assignment Not Found（未发现函数赋值）

必须在函数体内给出函数值。

307 Unexpected END Skipped（不希望的END，跳过）

END是不希望的；有可能是漏掉了BEGIN，CASE，或RECORD。

308 Compiler（编译程序）

309 Attribute Invalid（无效的属性）

只能对过程或函数给出变量的属性，反之也是如此。或混用无效的属性，如PUBLIC和EXTERN。

310 Attribute Expected（希望有属性）

左括号表示有属性，但这并不是一个属性。

311 Skip To Identifier（跳到标识符上面）

312 Identifier Expected（希望有标识符）

313 Reserved（保留的）

314 Identifier Expected Skip To;（希望有标识符，跳到；上）

315 Type Unknown Or Invalid Assumed Interger Begin Skip（类型是未知的或是无效的，假设为整型，开始跳过）

316 Identifier Expected（希望有标识符）

PROCEDURE或FUNCTION后的标识符不在参数表中。

318、319 Compiler（编译程序）

320 Previous Forward Skip parameter List（以前的正向跳过参数表）

当定义一个正向（或接口）过程或函数时，参数表和函数返回类型是不能重复的。

321 Reserved（保留）

322 Reserved（保留）

323 Invalid Attribute In Procedure Or Functoin（过程或函数中的属性是无效的）

嵌套的过程或函数不能具有属性或是EXTERN。

324 Compiler（编译程序）

325 Already Forward（已是正向）

对同一个过程或函数，FORWARD不能使用两次。

326 Identifier Expected For Procedure Or Function（过程或函数希望有标识符）

327 Invalid Symbol Skipped（跳过无效的符号）

在接口中决不能使用FORWARD或EXTERN命令。

328 EXTERN Invalid With Attribute（EXTERN带有无效的属性）

EXTERN过程不能有PUBLIC属性。

329 Ordinal Type Identifier Expected Integer Assumed Begin Skip（希望有有序类型标识符，假设为整型，开始跳过）

330 Contains File Cannot Initialize（包含文件，不能初始化）

虽然允许把文件放在记录变体中，但这是不安全的，而且用通常的NEWF QQ 调 用不自动地进行初始化。

331 Type Identifier Expected Assume Character（希望有类型标识符，假设为字符）

一般信息；该标识符不是类型标识符。

332 Reserved（保留）

333 Not Supertype Assumed String（非超数组类型，假设为字符串）

看起来貌似超数组的标识符，但类型标识符不是超数组类型，所以假设为 STRING超数组类型。

334 Type Expected Integer Assumed（希望有类型，假定为整型）

335 Out Of Range 255 For LSTRING（LSTRING超出了255范围）

336 Cannot Use Supertype Use Designator（不能使用超数组类型，要使用指示符）

超数组类型必须是参考参数或指针参考。

337 Supertype Designator Not Found（超数组指示符没有找到）

所有上界都必须用巨型数组指示符表示。

338 Contains File Cannot Initialize（包含有文件，不能初始化）

虽然允许文件类型为超数组，但这是不安全的，而且用通常的 NEWFQQ 调用不自动地进行初始化。

339 Supertype Not Array Skip To; Assumed Integer（非超数组类型，跳到；上，假设为整型）

在类型子句中，关键字SUPER总是要跟有ARRAY。

340 Invalid Set Range Integer Zero To 255 Assumed（无效的集合范围，假定为整数0—255）

一集合的基本类型必须在0到255的子界之内。

341 File Contains File（文件包含有文件）

文件类型不能直接地或间接地包含有文件类型。

342 PACKED Identifier Invalid Ignored（忽略无效的PACKED标识符）

关键字PACKED后必须跟着ARRAY，RECORD，SET，或FILE；它不能跟有类型标识符。

343 Unexpected PACKED（不望希的PACKED）

关键字PACKED只能用于结构类型（上述）。

345 Skip To;（跳到；）

希望分号在说明末尾（不能在行末）。

346 Insert;（插入；）

希望分号在说明末（在行末）。

347 Reserved（保留）

348 UNIT Procedure Or Function Invalid EXTERN（UNIT过程或函数的EXTERN是无效的）

在IMPLEMENTATION中，任何未执行接口过程和函数必须在IMPLEMENTATION的开始处说明是EXTERN的，但该EXTERN在后面才出现。

350 Not Array Begin Skip（非数组，开始跳过）

VALUE部分中，跟有方括号的变量不是数组。

351 Not Record Begin Skip（非记录，开始跳过）

VALUE部分中，跟有点的变量不是一个记录类型。

352 Invalid Field（无效的字段）

在VALUE部分中，看作是字段的标识符不在记录中。

353 Constant Value Expected（希望是常量值）

在VALUE部分中，变量只能初始化成常量。

354 Not Assignment Operator Skip To;（非赋值运算符，跳到；上）

在VALUE部分中，没有找到赋值运算符。

355 Cannot Initialize Identifier Skip To;（不能初始化标识符，跳到；上）

VALUE部分中的符号不是固定（STATIC）内存这个层次所说明的变量，或具有EXTERN的属性。

356 Cannot Use Value Section（不能使用数值部分）

在IMPLEMENTATION中设置VALUE部分，并非在INTERFACE中设置。

357 Unknown Forward Pointer Type Assumed Integer（未知的正向指针类型，假定

是整型）

作为参考类型引用的标识符在TYPE(或者VAR)部分中已有说明。(切不可说明本身)。

358 Pointer Type Assumed Forward (假定指针类型是正向的)

在该TYPE 部分中，出现在其中的指针或地址类型所引用的类型已经在闭合作用域中说明过，但引用类型的标识符又在同一TYPE部分中再次进行说明，例如：

TYPE A = WORD; PRCOEDURE B; TYPE C = ∧A;
A = REAL;

所谓正向类型的信息就是在这种情况下使用的（例如，REAL)。

359 Cannot Use Label Section (不能使用标号部分)

在IMPLEMENTATION中，设置LABEL部分，而不是在INTERFACE中设置。

361 Constant Expression Expected- Zero Assumed (希望有常量表达式，假定为零)

在CONST部分中，表达式不是常量。

362 Attribute Invalid (属性无效)

在VAR部分中，PUBLIC带有EXTERN。

364 Contains File Initialize Module (包含有文件初始化模块)

文件变量必须初始化，所以当一个文件变量在模块中说明时，必须调用文件初始化模块（作为无参数的过程）以初始化这些文件。

365 Reserved (保留的)

366 UNIT Identifier Expected Skip To; (希望有UNIT标识符，跳到；)

USES后没有一个设备标识符。

367 Initialize MODULE to Initialize UNIT (初始化MODULE为初始化UNIT)

USES 子句启动一个设备初始化调用，但要请求该调用，该模块必须作为一个过程来调用。

368 Identifier List Too Long Extra Assumed Integer (标识符表特别长，假定为整型)

在带有标识符表的USES子句中，在标识符表中所找到的标识符多于接口的分量。

369 End of UNIT Identifier List Ignored (不管UNIT标识符表的结束)

在带有标识符表的USES子句中，在表中所找到的标识符少于接口成分。

371 UNIT Identifier Expected (希望有UNIT标识符)

在INTERFACE短语之后；找不到UNIT标识符。

372 Compiler (编译程序)

373 Identifier In UNIT List Not Declared (UNIT表中的标识符没有说明过)

在接口UNIT表中，有一个标识符在接口体中没有说明过。

374 Program Identifier Expected (希望有程序标识符)

在关键字PROGRAM或MODULE之后没有标识符（致命的错误)。

375 UNIT Identifier Expected (希望有UNIT标识符)

在IMPLEMENTATION OF之后没有设备标识符（致命的错误)。

376 Program Not Found (没找到程序)

没有找到关键字 PROGRAM, MODULE, 或者 IMPLEMENTATION OF (致命的

错误)。如果源文件不是Paseal的编译对象，就可能出现该信息。

377 File End Expected Skip To End (希望文件末端，跳到文件末)

378 Program Not Found (没找到程序)

PROGRAM或初始化过的IMPLEMENTATION的主体，或者MODULE或其它IMPLEMENTATION的最终的END，没有找到。

2.后端错误

后端（优化程序和代码生成程序）会给出两种错误：用户错误和内部错误。所有的错误都与前端检测不到的那些限制有关。

无论是内部错误还是用户错误都会立即停止工作。这两种错误都会给出错误代码和大约的列表行号。

后端用户错误：

(1) Attempt to divide by zero (试图用零去除)

例如：A DIV 0。

(2) Overflow during integer Constant folding. (整型常量合并时溢出)

例如：MAXINT + A + MAXINT。

(3) Expression too Complex/Too many Internal Label (表达式太复杂/内部标号太多)

企图用中间值赋值来分解表达式。

后端内部错误：

这些错误有如下格式：

＊＊＊Internal Error NNN

NNN是内部错误号，它的范围是1到999。当发现有内部错误时几乎不能做什么工作，除非把情况报告给授权的IBM个人计算机供应者，或许要更改出错行附近的程序。

二、运行错误信息

1.文件系统错误

运行时发现的错误可分为文件系统错误和其他错误。文件系统错误码的范围为1000到1999。1000到1099是属于操作系统出错（来自设备U)，从1100到1199是属于Pascal文件系统出错（来自设备F)。

所有的文件系统错误都有如下格式：

error type error in file *filename* 错误代码

*error type*字段是以文件控制块中的ERRS字段为根据的。括号中的字母表示的是那个设备(U和F)可能产生错误。错误代码如下：

代码	状态
0	无错
1	保留的
2	保留的
3	操作(UF)
4	保留的
5	保留的

6　　保留的
7　　文件名（UF）
8　　设备满（U）
9　　保留的
10　　文件没找到（U）
11　　保留的
12　　保留的
13　　文件没打开(F)
14　　数据格式(F)
15　　行太长(UF)

(1) 设备U错误

1000 Write Error When Closing File（关闭文件时写操作错误）

1001 Reserved（保留的）

1002 Filename Extension With More Than 3 Character（文件扩展名多于3个字符）

1003 Error During Creation Of New File（建立新文件时出错）

1004 Error During Open Of Existing File（打开现存文件时出错）

1005 Filename With More Than 8 Or Zero Characters（文件名多于8个字符或没字符）

1006 Reserved（保留的）

1007 Total Filename Length Over 12 Character（文件名的总长度超过12个字符）

1008 Write Error When Advancing To Next Record（前移到下一记录时写操作出错）

1009 File Too Big (over 65535 Logical Sectors)（文件太大（超过65535个逻辑扇区））

1010 Write Error When Seeking To Direct Record（寻找直接记录时写操作出错）

(2) Pascal文件系统错误

1100 ASSIGN Or READFN OF Filename To Open File（给打开文件进行 ASSIGN 或READFN OF 文件名）

1101 Reference To Buffer Variable Of Closed File（对关闭文件的缓冲变量进行引用）

1102 Textfile READ OR WRITE Call To Closed File（文本文件的READ 或 WRITE 调用已关闭文件）

1103 READ When EOF Is True (Sequential Mode)（当EOF是真时，进行READ(顺序方式)）

1104 READ To REWRITE File, OR WRITE to RESET File (Sequential Mode)（对REWRITE的文件进行READ，或对RESET的文件进行WRITE（顺序方式））

1105 EOF Call To Closed File（对关闭的文件进行EOF调用）

1106 GET Call To Closed File（对关闭的文件进行GET调用）

1107 GET Call When EOF is True (SEQUENTIAL Mode)（当 EOF 是真时，进行GET调用（顺序方式））

1108 GET Call To REWRITE File (SEQUENTIAL Mode) (对REWRITE的文件进行GET调用(顺序方式))

1109 PUT Call To Closed File (SEQUENTIAL Mode) (对关闭的文件进行 PUT 调用(顺序方式))

1110 PUT Call To RESET File (SEQUENTIAL Mode) (对RESET 的文件进行 PUT 调用 (顺序方式))

1111 Line Too Long In DIRECT Textfile (DIRECT文本文件中行太长)

1112 Decode Error In Textfile READ BOOLEAN (在文本文件 READ BOOLEAN 中译码出错)

1113 Value Out Of Range In Textfile READ CHAR (在文本文件READ CHAR中数值超出范围)

1114 Decode Error In Textfile READ INTEGER (在文本文件READ INTEGER中译码出错)

1115 Decode Error In Textfile READ SINT (Integer Subrange)(在文本文件READ SINT时译码出错 (整型子界))

1116 Decode Error In Textfile READ REAL(在文本文件READ REAL中译码出错)

1117 LSTRING Target Not Big Enough In READSET (在READSET 中 LSTRING 的目标不够大)

1118 Decode Error In Textfile READ WORD (在文本文件READ WORD中译码错误)

1119 Decode Error In Textfile READ BYTE (在文本文件READ BYTE中译码出错)

1120 SEEK Call To Closed File (对关闭的文件进行SEEK调用)

1121 SEEK Call To File Not In DIRECT Mode (对不处在DIRECT方式下的文件进行SEEK调用)

1122 Encode Error (Field Width>255) In Textfile WRITE BOOLEAN (在文本文件进行WRITE BOOLEAN时编码出错(字段宽度大于255))

1123 Encode Error (Field Width>255) In Textfile WRITE INTEGER (在文本文件进行WRITE INTEGER时，编码出错 (字段宽度大于255))

1124 Encode Error (Field Width>255) In Textfile WRITE REAL (在文本文件进行WRITE REAL时编码出错 (字段宽度大于255))

1125 Encode Error (Field Width>255) In Textfile WRITE WORD (在文本文件进行WRITE WORD时，编码出错 (字段宽度大于255))

2.其它运行错误

非文件系统错误码的范围为2000到2999，在某些情况下，元命令控制着是否要进行错误检验；在其他情况下，总要进行错误检验，若有控制检验错误的元命令，则用下表加以说明：

2000..2049内存错误

因为栈与堆是相向生长的，所以这些错误是有关的：例如，不使用 $STACKCK+和栈溢出，则栈溢出就可以引起“堆是无效的”错误。

2000 Overflow (溢出)

当调用一个过程或函数时，栈 (结构) 溢出内存，检查是否是 $STACKCK+，和处在

某些其他情况。

2001 No Room In Heap (堆中没有空间)

当处在 NEW (GETHQQ) 过程时，堆中已经没有足够的空间可供新变量使用，总要进行捕获。

2002 Heap Is Invalid (堆是无效的)

当处在NEW (GETHQQ) 过程中时，提示堆结构的分配算法是错误的，总要进行捕获。

2003 Reserved (保留的))

2031 NIL Pointer Reference (NIL指针引用)

DISPOSE或$NILCK+发现有一个带有NIL值的指针(例如：0)。

2032 Uninitialized Pointer (未初始化的指针)

DISPOSE或$NILCK+发现有未初始化 (值为1) 的指针，如果使用$INITCK，指针只能取得此值。

2033 Invalid Pointer Range (无效的指针范围)

DISPOSE 或 $NILCK+发现有不是指向堆的指针即是无效的指针，有可能是指向从堆中删去的DISPOSE块和返回给系统的指针。

2034 Pointer To disposed VAR (指向已解除的VAR的指针)

DISPOSE 或 $NILCK+发现有指向已经解除的堆块指针，对同一个变量调用两次DISPOSE是无效的。

2035 Long DISPOSE Sizes Unequal (长格式DISPOSE的长度不等)

当使用长格式的DISPOSE时，变量的实际长度不等于已知标记值的长度。

2050..2099有序算术

2050 No CASE Value Matches Selector (没有CASE值与选择符配对)

在没有OTHERWISE子句的CASE语句中，没有一个分支语句含有的CASE常量值与选择符表达式值相等，要检验是否使用了$RANGECK。

2051 Unsigned Divide By Zero (不带符号被零除)

WORD值被零除；要检验是否使用$MATHCK+。

2052 Signed Divide By Zero (带符号被零除)

INTEGER值被零除；要检验是否使用了$MATHCK+。

2053 Unsigned Math Overflow (不带符号的数学溢出)

WORD结果不在0...MAXWORD范围之内；要检验是否使用了$MATHCK+。

2054 Signed Math Overflow (带符号的数学溢出)

INTEGER结果不在—MAXINT,+MAXIN，范围之内；要检验是否使用了$MATHCK+。

2055 Unsigned Value Out of Range (不带符号的值超出范围)

赋值或数值参数中的源数值超出了目标数值的范围；目标可以是WORD (包括BYTE)的子界，或CHAR，或枚举类型，这种情况也可能出现在SUCC和 PRED 函数中，和指定一个LSTRING 的长度时发生，这些情况是用 $RANGECK+来检验的，当数组索引超出边界和数组是不带号的索引类型时也产生这种信息，这是用$INDEX CK+来检验的。

2056 Signed Value Out of Range (带符号的值超出范围)

同上，但适用于INTEGER类型及其子界。

2100..2149类型REAL算术

2100 REAL Divide By Zero (REAL被零除)

2101 REAL Math Overflow (REAL数学溢出)

2102 SIN Or COS Argument Range (SIN或者COS的自变量范围)(太大)

2103 EXP Argument Range (EXP自变量范围)(太大)

2104 SQRT Of Negative Argument (负自变量的SQRT)

2105 LN of Non-Positive Argument (非正自变量的LN)

2106 TRUNC/ROUND Argument Range (TRUNC/ROUND自变量范围)

2131 Tangent Argument Too Small (正切自变量太小)

2132 Arcsin Or Arccos of REAL>1.0(REAL大于1.0的反正弦或反余弦)

2133 Negative Real To Real Power (负实数的实数幂)

2150..2199结构类型错误

2150 String Too Long In COPYSTR (COPYSTR中的字符串太长)

2151 LSTRING Too Long In Intrinsic Procedure (内部过程中的LSTRING太长)

2180 SET Element Greater Than 255 (SET的元素大于255)

2181 SET Element Out of Range (SET元素超出范围)

集合赋值中的数值或集合的数值参数太大了，以致于不能供目标集合用，使用了$RANGECK，就检验之。

2200..2999其他错误

2400 Reserved (保留的)

2450 Unit Version Number Mismatch (设备的文本号不符合)

在UNIT初始化期间，用户（使用USES的那一个）和接口的实现已暴露了编译接口的文本号不相等；总要捕获。

4-4 COBOL语言的使用

在IBM PC发展COBOL程序和发展FORTRAN程序有相同的环境，基本相同的操作过程。(COBOL只要求至少64K内存)。这节主要指出COBOL在使用上与FORTRAN的不同之处，详细操作见4-2节内容。

4-4-1 主盘与文件 供COBOL编译，连接使用的主盘是COBOL和LIBRARY。各主盘包括下述文件：

```
COBOL——COBOL.COM
       COBOL1.OVR
       COBOL2.OVR
       COBOL3.OVR
       COBOL4.OVR
       REBUILD.EXE
       RUNED.BAT
       RUNEC.BAT
LIBRARY——COBOL1.LIB
         COBOL2.LIB
```

COBRUN. EXE
LINK. EXE

为了使用方便建议将系统盘上的COMMAND. COM 文件复制到：LIBRARY

4-4-2 建立 COBOL 源程序 建立源程序的步骤和操作过程与 FORTRAN，相同这里不再赘述。应该注意打入的命令格式为：

EDLIN B：文件名. COB

.COB 是 COBOL 源程序约定的文件类型。

4-4-3 编译源程序 COBOL 程序的编译只需一次，这和 FORTRAN、PASCAL 不同，但其编译过程和 FORTRAN 语言的第一次编译过程基本相同。所有的屏幕显示如下：

Source filename〔.COB〕：myfile
Object filename〔MYFILE. OBJ〕：
Source Listing〔NUL. LST〕：myfile
（myfile 是假设的文件名）
当编译结束时，屏幕显示出：
No Error or Warnings

如果编译程序检测到一个错误或发送一个警告信息，则错误或警告信息与以下的信息一起显示在屏幕上：

1 *Error or warning*

如果编译程序发现到错误，用户必须在连接程序之前在自己的源程序中寻找并解决。

其它操作与 FORTRAN 相同。

4-4-4 错误信息 这里列出了在编译和运行 IBM COBOL 程序时可能会遇到的所有错误信息。每一个信息都附有发生该错误的简短说明。

这些信息分成 2 部分：

.编译时的错误

.运行期间的错误

一、编译时的错误

IBM—PC 在编译期间能够检测出两种不同类型的错误：

命令输入错误和跟 DOS——有关的 I/O 操作遇到问题时产生的错误。

COBOL 程序中句法错误。

1. 命令输入和 DOS 有关 I/O 的错误

在编译期间，每当产生错误时，就显示下列信息，并附有简短说明。

信息	说　明
? Bad filename	文件名没有根据 DOS 规则来构成
? File not found	输入时，指定了一个不存在的文件名。
? Bad switch：/x	引用了一个编译不承认的开关项
? Command error：'x'	在命令行中有一个无效的项（x）。
? Can't create file	输出文件不能打开。
? Disk x full	在指定驱动器中的磁盘上已经装满信息了。如果 x是一个空格，则说明是约定的驱动器。
? Overlay n not found	COBOL 编译复盖文件之一（COBOLn. OVR)不在磁盘

上。

? Memory Full　　请看下面的说明。

? Compiler Error　　请看下面的说明。

必须注意有两个不常出现，也显示在控制台上的错误信息。一个是? Memory full。当内存不够装入编译程序从用户源程序中得到的所有符号和其它信息时，就产生这种错误。它指出程序太大，必须减掉一点或分成独立的编译模块。

在编译时，数据名和过程名的符号表通常占据了最大的用户空间。所有的名字都需要和该名字中字符一样多的字节，即每个数据名通常需要大约10个字节，每个过程名需要2个字节。平均说来在编译期间，数据区的每一行大约需要内存区的14个字节，过程区的每一行大约需要3—1/4个字节。

另一个错误信息是：

? Compiler Error in Phase n at address. 当编译产生混乱时，则出现该错误。这通常是由4个问题之一所引起的：

源程序不对，有时能用编译越来越大块的用户程序来确定错误，从只有几行开始，直到错误重新出现。

源程序的磁盘被损坏。

编译文件或复盖文件之一被损坏。在这种情况下，应该重新复制一个盘片。

栈溢出也可能产生这个错误。可以试用/P选择项来确定这一类型的错误。遇到这两种错误信息，编译立即停止。

2.句法错误

此类信息列在被编译的程序列表的底部，同时也显示在屏幕上。它们由二部分组成。

(1) 有关的源程序行号或文件的错误信息，即4位十进制数，后跟一个冒号(：)。

(2) 编译程序检查出来错误的英语说明。如果使用了/F/或/W/ 开头，则只是一个警告错误，否则这个错误就较为严重，可以阻止进行连接和运行。

在编译结束时，如果存在错误或警告，它们总要显示在屏幕上。这将使用户能够简单地修改程序。

错误信息一般按字母排列，而带有/F/和/W/ 警告字母的信息列在信息的后部。每个信息的下面是有关的处理意见或解释。

A FILE-ID NAME IS UNDEFINED.

没有定义VALUE OF FILE-ID子句中指定的数据名。

A PARAGRAPH DECLARATION IS REQUIRED HERE.

一个EXIT 语句后面没有节或段的标题。

AREA A NOT BLANK IN CONTINUATION LINE.

在继续行的A区出现字符。

AREA-A VIOLATION;RESUMPTION AT NEXT
PARAGRAPH/SECTION/DIVISION/VERB.

在8—12列之一开始的项不能解释为部分标题，节名，段名，文件说明，01或77级(层)号。

CLAUSES OTHER THAN VALUE DELETED.

88级项目的数据描述中包含了除VALUE IS.外的一个描述子句。

ELEMENT LENGTH ERROR.

引号括起来的文字长度超过了120个字符，或数值文字超过了 18 位数位，或标识符/名超过了30个字符。

ERRONEOUS FILENAME IS IGNORED.

一个没有被指定为文件名的项出现在需要文件名的地方。

ERRONEOUS QUALIFICATION; LAST DECLARATION USED.

用于数据名的定义符是不正确的，或者不是唯一的。

ERRONEOUS SUBSCRIPTING; STATEMENT DELETED.

给某一数据名提供了太少或太多的下标。

EXCESSIVE LITERAL POOL OR DISPLAY STRING LENGTH.

在某一个段中包含的文字总长度大于4096个节。

EXCESSIVE NUMBER OF FILES/4KB WORKING-STORAGE BLOCKS.

(所描述的文件数)+(WORKING-STORAGE的大小除以4KB，并四舍五入)+(在LINKAGE SECTION 中01级和77级的项数) 之和大于14。

EXCESSIVE OCCURS NESTING IS IGNORED.

OCCURS子句的嵌套超过了3级。

EXCESSIVE SEGMENT NUMBER.

节头包含了一个大于99的节号。

EXCESSIVE SEGMENT NUMBER IN DECLARATIVES.

在描述(DECLARATIVES)域中节头包含了一个大于49的节号。

FILE NOT SELECTED; ENTRY BYPASSED.

给FD定义一没有出现在任何SELECT 句子中的文件名。

FILL CHARACTER CONFLICT.

在格式3的ACCEPT语句中，同时指定了SPACE-FILL和ZERO-FILL。

FRACTIONAL EXPONENT OR NEGATIVE SCALED BASE (99P).

在COMPUTE语句中，指数是一个带有小数点的数值文字，或者是用一位数字描述的数字数据项插到一个被假设为小数点的右边；或者在幂基底的 PICTURE (**前面的项) 中包含了字符P作为最右边一位数。

GROUP ITEM, THEREFORE PIC/JUST/BLANK/SYNC IS IGNORED.

一个只允许使用初等数据项的短句被直接用在一个较高级号的一个项的描述中。

GROUP SIZE GREATER THAN 4095; LENGTH SET TO 1.

在01以外的级中，所说明的项大于4095个字节。

ILLEGAL CHARACTER.

遇到一个非法字符。

ILLEGAL COPY FILENAME.

复制文件的文件名出错。

ILLEGAL MOVE OR COMPARISON IS DELETED.

MOVE 语句的操作数或相关条件的种类不一致。

IMPERATIVE STATEMENT REQUIRED.

STATEMENT DELETED.

一个条件语句被包含在不是IF语句的条件语句中。

IMPROPER CHARACTER IN COLUMN 7.

在7列中遇到一个非法字符。

IMPROPER PICTURE. PIC XASSUMED.遇到一个非法的PICTURE子句。

IMPROPER PUNCTUATION

遇到不正确的标点符号，例如，一个逗号或句号后面必须有一个空格。

IMPROPER REDEFINITION IGNORED.

在REDEFINES 子句中指定的数据名与当前数据名不同级，或者数据名被带有较低级号的项分开。

IMPROPERLY FORMED ELEMENT.

对于某一项遇到了不正确的句法，例如，在一个数字文字中，可能用了多个小数点。

INCOMPLETE (OR TOO LONG) STATEMENT DELETED.

一个动词直接跟在一个局部的语句格式后面，或者另外一个可接受的语句，在编译要读入它时，显得太大了。

INVALID KEY SPECIFICATION.

一个相对文件或索引文件的关键项不应该被注上下标，或者关键项与类型中或 USAGE 中的文件结构不一致。

INVALID QUOTED LITERAL

发生了文字长度为零，不适当的结构，或漏掉了后括号的情况。

INVALID RECORD SIZE (S) IGNORED.

一个FD的RECORD 子句有错误。

INVALID SELECT-SENTENCE.

FILE-CONTROL段中SELECT句子的句法不正确。

INVALID VALUE IGNORED.

VALUE IS 短句中指定的值是一个不适当的格式化的文字。

JUSTIFICATION CONFLICT.

在格式3的 ACCEPT 语句中，LEFT-JUSTIFY和RIGHT-JUSTIFY两项同时被指定。

KEY DECLARATION OF THIS FILE IS NOT CORRECT.

对于一个相对文件来说，漏掉了RELATIVE KEY 子句，或对于某一索引文件来说漏掉了RECORD KEY 子句。

KEYS MAY ONLY APPLY TO AN INDEXED/RELATIVE FILE.

对一个顺序文件或行顺序结构的文件指定了一个 RECORD KEY或 RELATIVE KEY子句。

LITERAL TRUNCATED TO SIZE OF ITEM.

VALUE IS短句中指定的文字比已描述的数据项长。

MISORDERED/REDUNDANT SECTION PROCESSED AS IS.

标识部分，设备部分或数据部分中的某一节发生秩序混乱或重复。

NAME OMITTED; ENTRY BYPASSED.

某一数据描述项中遗漏了数据名。

NON-CONTIGUOUS SEGMENT DISALLOWED.

两个同号，且大于49的节被一个或多个不同号的节分开。

NO PICTURE; ELEMENTARY ITEM ASSUMED TO BE BINARY.

对于一个初等数据项没有给出 PICTURE.

OCCURS DISALLOWED AT LEVEL 0 1/77, OR COUNT TOO HIGH.

一个 OCCURS 子句出现在01级或77级的数据描述项中，或所指定的出现次数大于1023。

OMITTED WORD 'SECTION' IS ASSUMED HERE.

数据区的某一节标题中漏掉了所需的字SECTION.

PROCEDURE-NAME IS UNRESOLVABLE.

在一个不是完全合格的，或者不是唯一的节名或过程名中存在有参考值。

PROCEDURE RANGE NOT IN CURRENT SEGMENT.

大于49的某一节中的 PERFORM 语句涉及到大于49的不同节中的一个过程。

PROCEDURE RANGE SPANS SEGMEENTS.

在PERFORM 语句中指定的过程范围（过程名1 THRU 过程名2)包含了大于49的各个节中的段，或者是小于等于49和大于49的各个节中的段。

REDUNDANT FD PROCESSED AS IS.

同一文件名出现在不止一个的文件描述中。

REWRITE VALID ONLY FOR A DISK FILE.

REWRTE 语句中的文件名项是一个分配给了PRINTER的文件。

SEMANTICAL ERROR IN SCREEN DESCRIPTION.

在5种不同的情况下都可能产生这个信息：

SCREEN SECTION 不是用01级的屏幕项说明开始；

01级的屏蔽项描述不包括一个屏蔽名；

一组屏蔽项用一个只允许对初等项的子句来描述；

一个初等屏蔽项的描述漏了FROM,TO,USING，或 VALUE；

一个屏蔽项的描述包含了不相容的子句（如USING和VALUE)。

SIGN CLAUSE IGNORED FOR UNSIGNED ITEM.

一个带有USAGE IS DISPLAY 的数值项的PICTURE被描述成无符号的，但又存在一个 SIGNIS 子句。

SINGLE-SPACING ASSUMED DUE TO IMPROPER ADVANCING COUNT.

WRITE语句中BEFORE 或 AFTER短句的操作数大于120。

SOURCE BYPASSED UNTIL NEXT FD/SECTION.

文件描述中的某一错误阻止了进一步的分析。

STATEMENT DELETED BECAUSE INTEGRAL ITEM IS REQUIRED.

一个数值数据项,(它的 PICTURE 把数字指定到小数点右边,)被用在需要整数的地方。

STATEMNT DELETED BECAUSE OPERAND IS NOT A FILENAME.

出现在需要的文件名之处的一个名字没有作为文件名说明。

STATEMNT DELETED DUE TO ERRONEOUS SYNTAX.

存在着一个句法错误，对于这个错误没有更多的说明信息可以利用。

STATEMENT DELETED DUE TO NON-NUMERIC OPERAND.

一个字母数字或字母数字编辑项被用作一个算术语句的操作数，一个数值编辑项被用作

除结果以外的操作数，或者一个数的长度大于18位十进制数。

SUBSCRIPT 0 OR OVER MAX.NO.OCCURRENCES,
1 USED.

用作下标的文字与有关的OCCURS 子句所定义的范围不一致。

SUBSCRIPT OR INDEX-NAME IS NOT UNIQUE.

一个需要限制的名被用作下标。

SYNTAX ERROR IN SCREEN DESCRIPTION.

一个屏幕项描述包含了一个不被承认的子句，或者结构不适当的句子或者多余的子句。

UNRECOGNIZABLE ELEMENT IS IGNORED.

所需的关键字漏掉，或者一个数据名或过程名没有说明。

USING-LIST ITEM LEVEL MUST BE 01/77.

用在FROCEDURE DIVISION标题中USING列表的名字没有在01级或77级处说明。

VALUE DISALLOWED-OCCURS/REDEFINES/TYPE/SIZE CONFLICT.

对于用OCCURS或REDEFINE 子句描述的数据项(或者包括在用OCCURS或 REDEFINE 子句描述的项中) 指定了一个VALUE IS子句；或者 VALUE IS子句中给出的文字与描述的项的PICTURE 不一致。

VALUE OF FILE-ID REQUIRED.

VALUE OF FILE-LD子句没有在分配给DISK的文件的文件描述中指定。

VARYING ITEM MAY NOT BE SUBSCRIPTED。

由PERFORM语句的VARYING短句控制的数据项被指定为下标。

/F/ FILE NEVER CLOSED.

该文件没有CLOSE语句。

/F/ FILE NEVER OPENED.

该文件没有OPEN语句。

/F/ INCONSISTENT READ USAGE.

对某一文件有一个OPEN INPUT语句，但没有READ语句，反之亦然。

/F/ INCONSISTENT WRITE USAGE.

对某一文件存在一个OPEN OUTPUT语句，但没有WRITE语句，反之亦然。

/W/ BLANK WHEN ZERO IS DISALLOWED.

BLANK WHEN ZERO 短句出现在一个字母数字的描述中或字母数字编辑项的描述中。

/W/ DATA DIVISION ASSUMED HERE.

漏了DATA DIVISION 标题。

/W/ DATA RECORDS CLAUSE WAS INACCURATE.

DATA RECORDS 子句中给出的记录名与跟随文件描述的记录描述不一致。

/W/ FD-VALUE IGNORED SINCE LABELS ARE OMITTED.

VALUE OF FILE-ID子句被用于分配给PRINTER的文件描述中。

/W/ FILE SECTION ASSUMED HERE.

遗漏了 FILE SECTION的节标题。

/W/ INVALID BLOCKING IS IGNORED.

FD的BLOCK子句中有错误。

/W/ 'LABEL RECORD STANDARD' REQUIRED.

在分配给DISK的文件的FD中不存在LABEL RECORD (S) STANDARD短句。

/W/ LABEL RECORDS OMITTED ASSUMED FOR PRINTER FILE.

分配给PRINTER的文件的文件描述中遗漏了LABEL RECORDS OMITTED子句。

/W/ LEVEL 01 ASSUMED.

一个记录描述，使用了01之外的级(层)号。

/W/ PERIOD ASSUMED AFTER PROCEDURE-NAME DEFINITLON.

节标题或段标题没有用一个句号来结束。

/W/ PICTURE IGNORED FOR INDEX ITEM.

用USAGE IS INDEX短句描述的数据项也有一个PICTURE短句。

/W/ PROCEDURE DIVISION ASSUMED HERE.

漏写PROCEDURE DIVISION的部分标题。

/W/ RECORD MAX DISAGREES WITH RECORD CONTAINS; LATTER SIZES PREVAIL.

FD的RECORD CONTAINS子句中所指定的记录大小与有关记录描述的大小不一致。

/W/ REDUNDANT CLAUSE IGNORED.

相同的子句被指定了多次。

/W/ RIGHT PARENTHESIS REQUIRED AFTER SUBSCRIPTS.

下标的括号漏了。

/W/ TERMINAL PERIOD ASSUMED ABOVE.

数据描述项或段没有用句号结束。

/W/ WORKING-STORAGE ASSUMED HERE.

WORKING-STORAGE的标题遗漏了。

二、运行期间错误

有些程序错误不能被编译程序检查出来，但运行它时，就会引起程序的夭折，每种错误信息都带有模块名 (PROGRAM-ID)，并且如果目标行号存在（编译时可选项),则程序停止在处理那个语句的编辑行号上，这个信息以下列格式显示：

```
**RUN-TIME ERR:
reason (看下面的列表)
line number (选择项)
Program-id
```

对于程序结束的可能原因和附加解释，在下面列出。

信　息	解　释
REDUNDANT OPEN.	企图打开一个已经打开的文件
DATA UNAVAILABLE.	企图引用一个未打开的文件中的某一记录的数据，或者已到达“AT END”条件。
SUBSCRIPT FAULT.	下标有一个非法值（通常小于1)。
INPUT/OUTPUT.	不可恢复的I/O错误，在用户的COBOL 程序中，没

	有规定AT END子句，INVALIDKEY子句，FILE STATUS项或描述节，故不产生相应的动作。
NON-NUMERIC DATA.	每当一个数据项的内容不符合所给出的PICTURE时，就产生这种情况。如果这是一个从属于使用Numberic测试的错误时用户应该检查输入数据。(因为此时输入编辑尚未完成)。
PERFORM OVERLAP.	一个非法的PERFORM序列，例如，在执行A段程序时，在从它退出之前又要去初始化另一个PERFORM A.
ILLEGAL READ.	企图读一个不是以INPUT或I-O方式打开的文件。
ILLEGAL WRITE.	对于顺序存取文件来说，企图写入一个未以OUTPUT方式打开的文件，或对于随机或动态存取文件来说，企图写入一个未以OUTPUT或I-O方式打开的文件。
ILLEGAL REWRITE.	企图重写一个未以I-O方式打开的文件的记录。
REWRITE; NO READ.	当最后一个操作不是成功的READ时，企图重写一个顺序存取方式文件的记录。
OBJ.CODE ERROR.	遇到一个未定义的目标程序指令。仅当在内存或磁盘文件中程序的绝对形式被破坏时才会产生。
GO TO (NOT SET).	试图去执行一个还没有改换成为一个目标有关的无效GO语句。
FILE LOCKED.	企图打开一个原先以CLOSE WITH LOCK关闭的文件。
DELETE; NO READ.	当最后一个READ操作不成功时，企图删除一个顺序存取方式文件中的记录。
ILLEGAL DELETE.	未以I-O方式打开相对文件，
ILLEGAL START.	未以INPUT或I-O方式打开文件。
SEGmn LOAD ERR.	当企图装入复盖段49+dd时产生此种错误，这里dd是与十六进制数nn等值的十进制数。
NEED MORE MEMORY.	因为没有足够的动态分配存贮器，索引文件的管理非正常终止。

下列错误信息以＊＊COBOL：开始，以区别与之类似的DOS错误信息。

信　息	解　释
＊＊COBOL：FILE'filename' NOT FOUND.ENTER NEW DRIVE LETTER。	没有找到被连接的文件，程序段文件，或公共运行时间文件。
＊＊COBOL:PROGRAM TOO BIG TO FIT IN MEMORY.	没有足够的内存空间可以装入连接程序或公共运行期间的文件。
＊＊COBOL：ERROR IN	装入连接或公共运行期间的EXE文件中产生的

EXEFILE.　　错误。

4-5 LINK 信息 所有信息除了警告信息之外，都使连接终止。因此，在找出问题并且解决之后，必须重新运行LINK。

除非把列表文件传送给CON，否则错误信息会出现在列表文件和显示屏上；如果把列表文件传送到CON，则错误信息不会被显示。

下面列出连接程序信息的清单。

About to generate. EXE file

调换磁盘并按下任意键。

An internal failure has occurred

把这个问题报告给IBM-PC。厂商(IBM Personal Computer Dealer)

Attempt to access bata outside of segment bounds

目标模块有可能坏了。

Bad Numeric Parameter

在/STACK的参数中发现一个无效的数。

Cannot find file filename.

调换磁盘并按下任意键。如果在未能找到目标模块的磁盘上 VM.TMP 已经打开或列表文件已经打开，则这个错误是不可恢复的。

Cannot find library libraryname

送入新的驱动器字母号。

Cannot open overlay

Cannot open temporary file

目录满了。

DVP record too complex

在汇编源程序所建立的目标模块中出现了问题。在扩展之前，一个独立的DUP需要1024个字节。

Fixup offset exceeds field width

一个机器语言处理指令提供了一个用NEAR属性代替了FAR属性的地址。

Invalid format file

某一库文件有错。

Invalid object module

目标模块格式不对或不完整（如在半中间停止了语言处理)。

Invalid Switch

连接程序在命令行中或在一个提示中发现一个无效参数。

Out of space on list file

Out Of space on run file

Out Of space on VM.TMP

磁盘空间不够装VM.TMP文件。

Program size exceeds capacity of linker

正在处理的装入模块太大。

Segment Size exceeds 64K

企图组合相同名字的段，组合结果使段需要大于64K的内存。64K字节是段寻址极限。

Stack Size Exceeds 64K

在/STACK参数中发现一个大于65536的数。

Symbol defined more than once

连接程序发现了二个或多个由某一单符号名定义的模块。

Symbol table capacity exceeded

极限大约30K，请用较短的和/或者较少的名字

There was/were number errors detected

Too many libraries specified

极限是8个库文件。

Too many external symbols in one module

每个模块限制256个外部符号。

Too many groups

包括了DGROVP，极限是10.

Too many public symbols in one module

极限是1024个公用符号。

Too many segments or classes.

极限是256(段和类一起占用)。

Too many overlays

极限是64。

Unexpected end—of—file on library

Unexpected end—of—file on VM.TMP

包含VM.TMP的磁盘已经退出。

Unresolved external reference

未能找到一个调用语句的索引。

VM.TMP is an illegal file name and has been ignored

VM.TMP不能用来作为目标文件名。

参 考 书 目

〔1〕《微型计算机COBOL程序设计》 何克抗 裴广生编 北京师范大学出版社出

〔2〕《苹果II微型计算机和结构化BASIC语言编程》 王飞龙主编 湖北科学技术出版社

〔3〕《微型计算机BASIC语言》 林卓然编 中山大学出版社

〔4〕《APPLE-II微型计算机 FORTRAN-80使用手册》 中国微型电脑应用协会武汉分会

〔5〕《APPLE-II微型计算机 CP/M操作系统》 中国微型电脑应用协会武汉分会

〔6〕《APPLE-PASCAL 使用指南》 中国微型电脑应用协会武汉分会

〔7〕《CROMEMCO 微型计算机软件资料汇编》(一)(二)(三) 清华大学计算中心译编 清华大学出版社

〔8〕《IBM PC BASIC 与应用》天津鸿翔软件工厂

〔9〕《APPLE-II MBASIC使用手册》 北京电脑公司资料组

〔10〕《APPLE-II CP/M 操作手册》长沙市电子研究所翻印
〔11〕《IBM-PC 磁盘·操作系统(μS-DOS)》中国科学院科技咨询开发服务讲习班
科海新技术联合开发中心
〔12〕《TM 个人计算机 PASCAL 编译程序》福建省福州市开大电脑服务公司
〔13〕《IBM PC 磁盘操作系统》松岗电脑图书资料有限公司出版钱旭光、松开荣合译
〔14〕《TM 个人计算机 COBOL 编译程序》福建省福州市开大电脑服务公司
〔15〕《TM 个人计算机 FORTRAT 编译程序》福建省福州市开大电脑服务公司
〔16〕《MICROSOFT COBOL》
〔17〕《MICROSOFT PASCAL》