计 算 机 组 织
和
微 程 序 设 计

〔美〕朱 耀 汉 著
丁 佳 良 译

上 海 科 学 技 术 出 版 社

COMPUTER ORGANIZATION
AND MICROPROGRAMMING

Yaohan Chu

Prentice-Hall Inc., Englewood Cliffs, N. J. 1972

计算机组织和微程序设计

〔美〕朱耀汉 著

丁佳良 译

上海科学技术出版社出版

(上海瑞金二路 450 号)

新华书店上海发行所发行 上海市印刷三厂印刷

开本 787×1092 1/16 印张 21.5 字数 506,000

1979 年 2 月第 1 版 1979 年 2 月第 1 次印刷

印数 1—45,000

书号: 13119·736 定价: 2.00 元

作 者 序

计算机组织的说明就是机器功能方面的说明。在这里，既不讨论机器的具体元件，如电路、存储体、机柜、电缆等，也不讨论机器的详细逻辑如门、触发器、开关、指示灯、寄存器等的连接线路。功能方面的说明仅是叙述具体计算机的数据格式、数的表示方法、指令表、组态、微指令序列、时间关系、微命令和各种控制信号。简言之，所谓电子计算机组织，在这里理解为数字计算机中功能联系的说明和给定操作执行顺序的方法。

通常，描述计算机组织采用框图，并附有说明。这种方法尽管可以给出机器的一般概念，但它不能保证非常准确、详细。本书采用计算机设计语言(Computer Design Language—CDL)说明机器各个元素的特点及其工作序列。

用 CDL 语言说明计算机组织是详尽而确切的，并且是单义的。为了表示各种算法，使用了序列时间图，它可以说明各个平行过程、互相独立的循环和同时执行的序列。本书中既使用 CDL 语言的过程方案，也使用非过程方案。大量的例子证明 CDL 语言能够说明硬件实现的各种复杂算法。本书的对象是大学高年级学生。阅读本书不要求具备电子学方面的知识，除了电子技术和计算技术专业的学生外，本书也可供数学和经济学专业的学生阅读。CDL 语言的使用使得本书能引入大量例子，透彻地说明计算机内部的组织和算法。本书同其他叙述计算机组织的同类书籍的区别也就在此。在 CDL 语言的基础上，研制了以 CDL 语言的子集进行工作的模拟系统。其方案用在四个型号的计算机上，即 IBM 7094, UNIVAC 1108, CDC 6600 和 IBM 公司的 360 系统。UNIVAC 1108 方案既能以程序包方式工作，也能按会话终端的要求工作。IBM 7094 和 UNIVAC 1108 两种方案是由马里兰大学计算机科学中心研制的。CDC 6600 方案是马里兰州怀特-沃克城的海军研究实验室搞的。IBM 公司 360 系统的方案是多伦多大学计算系统小组提出的。

本书共有 11 章。第一章介绍 CDL 语言。第二章通过几个例子说明 CDL 语言的应用。这些例子都是简单的装置，如比较装置、代码转换器等。第三章叙述微程序控制的计算机和微程序设计过程的特点，阐明了简单的微程序控制计算机的组织原理和微程序调整的电子计算机。在第四、八和十一等章也还讨论了微程序控制的问题。第四章叙述定点平行二进制运算器，讨论了这种运算器的两种控制方案：一种以时序逻辑为基础的控制方案，一种为微程序控制方案。第五章叙述浮点平行二进制运算器。第四章中讨论的有小组和大组进位的平行加法器，类似于 IBM 7090 系列机中使用的加法器。第六章叙述串行二进制运算器和二-十进制运算器。第七章叙述随机存取存储器、相联存储器、虚拟存储器、存储器寻

址设备、栈存储器和动态输入器等设备的功能组织。同时还说明了类似于 IBM 360/85 中使用的缓冲存储器的组织。第八章讨论了各种控制组织的方案：使用时序逻辑的组织，微程序组织，异步组织和中央控制组织。第九章是在前几章介绍的主要功能装置的基础上，完整地给出机器的功能说明。我们取 IBM 360/40 机作为例子，它是360 系统中较为普及的型号之一。第十章叙述 360/40 机的通道组织，而第十一章以具体例子探讨机器软件用微程序实现的可能性。具体例子就是编写一道微程序，将浮动程序调整为存储器的真地址。

朱 耀 汉

目　录

第一章　计算机设计语言

通常，描述计算机组织采用框图，并附有说明。这种方法尽管可以帮助我们搞清楚一些基本概念，但要全面理解功能线路，就觉得它不够准确，也不够详细。因此，有必要研制描述计算机组织的高级语言。

近些年来，发表了有关这类语言的一些报导[2~18]。计算机设计语言(CDL—Computer Design Language[7]) 就是其中之一。这种语言能定义数字计算机内的功能组织、算法和操作执行序列，它包含强有力的说明手段，用于说明计算机的各个元素，如寄存器、译码器、开关、指示灯、存储体和终端输出。用 CDL 语言说明各元素、算法和操作，能保证所需的准确性，它包含有在位一级、字一级和位组一级说明的手段，以及能反映出同步信号和控制指令。利用这种语言也可以描述串行传送和并行传送，以及并行执行的操作。最后，掌握 CDL 语言不需要具备电子学方面的知识。

本章向读者介绍 CDL 语言，这在以后各章中都会用到。我们以一台简单的存储程序数字计算机为例讲解 CDL 语言，并在讲解语言的过程中讨论这台计算机的各个元素、微操作及其按给定顺序执行的方法。最后用 CDL 语言给出这台计算机的完整的说明。

1.1　说 明 语 句

现在我们研究一台存储程序计算机。这台计算机有六个寄存器，一台随机存取的存储体，三只开关，一只指示灯，一个译码器和一个同步信号源。机器的这些元素用说明语句定义。说明语句不仅表征每个元素，而且还赋给各个元素以符号名，在必要时还说明它们的功能。下面就来说明这台机器的每一种元素。

1.1.1　寄存器说明

这台计算机的六个寄存器是 R(0—23)，A(0—23)，C(0—14)，D(0—14)，F(0—5) 和 G。括号内界偶的值指出：R 和 A 两个寄存器有 24 位，寄存器中的每位都以自己的指标值标志。这样，最左边一位为 R(0) 或 A(0)，最右边一位为 R(23) 或 A(23)。R 是存储体的缓冲寄存器，A 是累加寄存器，操作分量的加、减或移位就在 A 寄存器内执行。C 和 D 两个寄存器各为 15 位。C 是存储体的地址寄存器，D 是程序寄存器(或下一条指令地址的寄存器)。G 寄存器的说明中无指标，表示它只有一位，用于控制机器的状态(停机/工作)。从刚才讨论的寄存器说明的例子就可看出用寄存器说明语句说明寄存器的特点：

```
Register, R(0—23),
          A(0—23),
          C(0—14),
          D(0—14),
          F(0—5),
          G。
```
(1.1)

为了阅读方便起见，寄存器说明中可以包括注解。例如，语句(1.1)可改写成下面的形式：

```
Register, R(0—23), $缓冲寄存器
          A(0—23), $累加寄存器
          C(0—14), $地址寄存器
          D(0—14), $程序寄存器
          F(0—5),  $控制寄存器
          G。      $启-停控制寄存器
```
(1.2)

注解以货币号$开始。

如果寄存器的某一部分所包含的各位有特殊含义，这部分寄存器就称为子寄存器(Subregister)。通过子寄存器说明可以给这个位组赋符号名。例如，子寄存器说明

```
Register,    R(0—23),
Subregister, R(OP)=R(0—5), R(ADDR)=R(9—23)。
```

它表示R寄存器左边的6位赋以符号名R(OP)，而右边15位赋以符号名R(ADDR)。子寄存器R(OP)和R(ADDR)分别包含某条指令的操作码和地址部分。有时，在执行一些特殊操作时需将两个(或几个)寄存器或子寄存器连在一起。这种寄存器称为级联寄存器(Casregister)。

例如，级联寄存器说明

```
Register,    A(1—5), PB,
Casregister, D(1—6)=A-PB。
```

它定义级联寄存器D，由A寄存器同PB寄存器连在一起组成。若不必赋予新的名，则不需要用级联寄存器说明语句。我们讨论另一个使用级联寄存器说明的例子：

```
Register,    A(0—23), Q(0—23),
Subregister, Q(M)=Q(1—23),
Casregister, AQ(0—46)=A-Q(M)。
```
(1.3)

这里说明了A寄存器同Q(M)子寄存器连在一起的级联寄存器AQ。必须指出，级联寄存器不是又引进一个寄存器，而是由已经说明过的寄存器连在一起所形成的寄存器。

再者，必须说明由一组触发器(即一位的寄存器)形成的一些相同寄存器的总体。作为例子，我们看使用四个相同的寄存器A1(0—8)、A2(0—8)、A3(0—8)和A4(0—8)保存八位十进制数。很明显，不用说明四个寄存器，而是引进一个数组寄存器说明就能很方便地说明这组寄存器：

```
Array-register, A(1—4, 0—8), Q(1—4, 1—8)。
```
(1.4)

(1.4)的说明定义了一个名为A的数组触发器，有4行、9列，同时还定义了另一个数组，名为Q，它有4行、8列。每一列有四位，保存一个二进制编码的十进制数字(详细情况在第六章叙述)。为了指明数组的个别列或个别行，应省略不需要的指标，但保留指标之间的逗号。例如，A数组的第二行可以写成子寄存器A(2,)，而Q数组的第八列可写成子寄存器Q(,8)。

也可以引进数组级联寄存器代替级联寄存器。例如，上面说明的两个数组寄存器A和Q连在一起的结果，可用下面的数组级联寄存器说明来定义：

Array-casregister, AQ(1—4, 0—16)=A-Q。 (1.5)

最后，我们要指出，以后叙述的一些基本说明语句也可以应用数组寄存器和数组级联寄存器。

1.1.2 存储体说明

我们选定的数字计算机应有一随机存取的存储体，它由大量被称为存储单元的寄存器组成。存储体内的每个单元都有对应的地址，每个存储体同地址寄存器和缓冲寄存器相连。访问某存储单元时，应在同存储体开始交换之前将该单元的地址置于地址寄存器。简单的随机存取存储体用存储体说明定义。我们所讨论的机器的存储体说明如下：

```
Register, C(0—14),                    $ 地址寄存器
Memory, M(C)=M(0—32767, 0—23)。
```
(1.6)

存储体说明(1.6)定义了存储体 M，并指出 C 寄存器作为其地址寄存器。说明中还指出，存储体的容量为 32768 字，字长 24 位。说明寄存器要用一个指标，而说明存储体要用两个指标。

1.1.3 开关说明

手动控制开关一般装在计算机的控制台上，用于从外部控制机器的工作。因此，开关就是有一个或几个位置的输入设备。如果是一个位置的开关，则开关接通时就产生脉冲。如果是几个位置的开关，它就停留在拨好的位置上。因此，开关也是保存信息的元件。

为了定义开关，使用开关说明。我们讨论的机器中有三个开关，这些开关可说明如下：

```
Switch, POWER(ON),
        START(ON),
        STOP(ON)。
```
(1.7)

说明语句(1.7)定义三只单位置的开关，每只开关都有同样的位置名称 ON。开关 POWER 使机器进入初始状态，而开关 START 和 STOP 分别执行机器的启动和停机。开关说明可以定义有一组指标的开关(若说明的是开关列)，或有两组指标的开关(若说明的是开关数组)。

1.1.4 指示灯说明

指示灯一般装在计算机控制台上，用于监视机器的状态。可以说，指示灯就是有一个或几个位置的输出设备。如果是一个位置的指示灯，则接通时就亮，如果有几个位置，则它显示的就是所确立的状态。因此，指示灯也可以认为是保存信息的元件。

为了表征指示灯，引入专门的说明语句。计算机中溢出指示灯说明语句如下：

Light, LTOV(ON, OFF)。 (1.8)

指示灯 LTOV 有两个位置 (ON 和 OFF)。当执行加法而产生溢出时，指示灯就置于 ON 位置。指示灯说明中，若指示灯名有一个或两个指标，则它所定义的是指示灯列或指示灯数组。

1.1.5 终端说明

逻辑网络是指能实现一定功能、但不包含存储元件的互相连接的一些逻辑电路的总体。逻辑网络也称为组合线路。对两个无符号二进制数进行并行相加的并行加法器就是逻辑网络的例子。

逻辑网络由相应的终端说明定义。例如，我们讨论的样机的平行加法器可由下面的说明语句定义：

$$
\begin{aligned}
&\text{Comment,}\quad \text{平行加法器说明。}\\
&\text{Register,}\quad \mathrm{A}(0\text{—}23),\ \mathrm{R}(0\text{—}23),\\
&\text{Terminal,}\quad \mathrm{C}(23)=0,\\
&\qquad \mathrm{C}(0\text{—}22)=\mathrm{A}(1\text{—}23)*\mathrm{R}(1\text{—}23)+\mathrm{R}(1\text{—}23)*\mathrm{C}(1\text{—}23)\\
&\qquad\qquad +\mathrm{C}(1\text{—}23)*\mathrm{A}(1\text{—}23),\\
&\qquad \mathrm{SUM}(0\text{—}23)=\mathrm{A}(0\text{—}23)\oplus\mathrm{R}(0\text{—}23)\oplus\mathrm{C}(0\text{—}23)。
\end{aligned}
\tag{1.9}
$$

在这里，平行加法器的输入端是 A 和 R 寄存器，而输出端为 SUM。输出端 C(0—22) 定义了平行加法器每一位的进位值，C(23) 位赋 0。须注意，由(1.9)说明语句定义的平行加法器是将 A 和 R 寄存器内的两个二进制数相加，不计符号，但在相加时不形成溢出标志。

译码器是逻辑网络的又一个例子。译码器线路是将寄存器内的任意一个值转换成一个且仅转换成一个输出值。对 n 位寄存器而言，与它相连的译码器最多可以有 2^n 个输出值。利用终端说明来定义有大量输出端的译码器显然是很麻烦的，而译码器又是计算机常用的元件，据此，引进专门的说明来定义译码器。在我们的样机中，控制部件的译码器是利用下面的译码器说明来定义的：

$$
\begin{aligned}
&\text{Register,}\quad \mathrm{F}(0\text{—}4),\\
&\text{Decoder,}\quad \mathrm{K}(0\text{—}16)=\mathrm{F}。
\end{aligned}
\tag{1.10}
$$

此说明语句指出，K(0)、…、K(16) 是与寄存器 F 相连的译码器输出端。由于寄存器 F 是五位寄存器，故译码器最多可有 32 个输出端。但是，给定的界偶的值(0—16)指出，只需要前面 17 个输出端。

1.1.6 同步信号源说明

同步信号源(时钟)是产生一个或几个脉冲序列的电子部件，这些脉冲序列启动计算机逻辑电路中的不连续操作。两个相邻同步信号之间的时间间隔称为*同步信号源的周期*。这种信号源由同步信号源说明语句定义。例如，机器的三相同步信号源说明如下：

$$
\text{Clock,}\quad \mathrm{P}(1\text{—}3)。 \tag{1.11}
$$

(1.11)说明语句表示同步脉冲名为 P，因此，信号源各相(或各拍)分别称为 P(1)、P(2) 和 P(3)。所有三相中的同步脉冲都有不变的重复频率和相移(图 1.1)。分别属于三相的三个相邻的脉冲形成所谓同步循环(Synchrocycle)。被说明信号源的一个同步循环中的三个脉冲能控制分三步的脉冲序列。

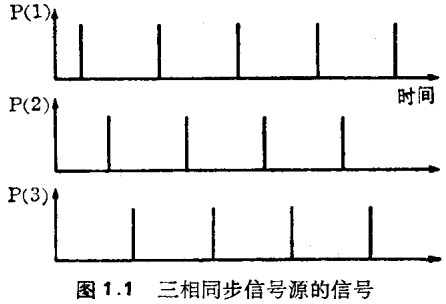

图 1.1　三相同步信号源的信号

1.2 微 语 句

计算机中实际实现的某些基本功能操作称为微操作。微操作用微语句来定义。下面我们介绍微语句中使用的操作的表示方法、表达式的例子和传送规则。

1.2.1 基本操作

微语句是一种符号，它表示逻辑网络在一个节拍周期内实现的某种功能。常用的操作称为基本操作。

表 1.1 包含了本书所使用的基本操作名单，并列出了相应的说明。

基 本 算 符　　表 1.1

算　　符	记　　号	说　　明
逻辑算符[1]		
NOT	′	非
OR	+	或
AND	*	与
EXOR	⊕	异或(按位加)
COIN	⊙	逻辑恒等
功能算符		
左移	shl	左移 1 位或几位[2]
右移	shr	右移 1 位或几位[2]
左环移	cil	左环移 1 位或几位
右环移	cir	右环移 1 位或几位
计数器加	countup 或 inc	计数器内容加 1
计数器减	countdn 或 dec	计数器内容减 1
算术算符		
加	add	两个无符号二进制数相加
减	sub	两个无符号二进制数相减

1) 逻辑操作按位执行。

2) 右边(左移时)或左边(右移时)空出的位填 0。

基本操作又分逻辑操作、功能操作和算术操作。五种逻辑操作 (NOT, OR, AND, EXOR, COIN) 的含义同布尔代数中规定的一样。并且，每种逻辑操作既适用于个别位，也适用于位组。另有六种功能操作(shl, shr, cil, cir, countup 或 inc, countdn 或 dec)。

移位操作(shl, shr)是将寄存器的内容左移或右移 1 位或几位；此时，右边或左边相应空出的位置填 0。循环移位操作 (cil, cir) 的功能类似于移位操作，区别仅在于寄存器左边和右边的位是"粘"在一起的(所以称为环形寄存器)。计数操作 (countup 或 inc, countdn 或 dec) 是将寄存器的内容加 1 或减 1。最后，还有两个算术操作 (add, sub)，分别为两个无符号二进制数的相加或相减。

这些基本操作可以是单边的，或双边的，或者属于这两者。单边操作涉及一个操作分量，双边操作涉及两个操作分量。NOT、countup 和 countdn 是单边操作，+、*、⊕、⊙、add 和 sub 是双边操作。表 1.1 中的其他操作既可以是单边的，也可以是双边的。下面举例说明这些操作的用法。

1.2.2 表达式

表达式是由常数、算符和(或)操作分量形成某种结构的组合。表达式给定微操作在发生转换前所执行的功能。表达式可以是操作分量，例如 A 或 OV；也可以是有一个操作分量的单边算符，例如 shl A 或 countup B；或者是双边算符，例如 A add R。表达式也可以是二进制、八进制或十进制常数。构成常数指标的规则如下：

十进制常数没有指标。

八进制常数始终有指标。

二进制常数仅当底不明显时有指标。

下面是表达式的一些例子：

$$\begin{array}{l} X' \\ \text{shl Y(ADDR)} \\ X \oplus Y \\ 3\ \text{shl X} \\ 5\ \text{cir Y} \end{array} \tag{1.12}$$

其中 X 和 Y 是寄存器，而 Y(ADDR) 是子寄存器。

1.2.3 微操作

微操作是将常数送寄存器，或者是将一个或两个寄存器的内容送入另一个寄存器，且在传送过程中对数据执行操作。任何微操作的特点就是在其执行过程中导致寄存器之间的数据传送。

正如 1.2 节中所指出的，微操作是用微语句来定义的，微语句由两部分组成：其一是说明操作分量和微操作的动作，其二是传送过程本身和微操作的结果。微语句的第一部分由表达式给出，而在第二部分中，有表示传送过程的箭头和存放结果的寄存器的名。我们分析一下 R 寄存器的内容送 A 寄存器的微操作。此微操作由下述微语句定义：

$$A \leftarrow R。 \tag{1.13}$$

须注意，这里箭头(←)是传送符号，而表达式(箭头右边)就是寄存器名。

必须强调指出，在微操作的执行过程中，传送数据不是瞬时发生的，也就是说要占一定的时间间隔。为了使微操作能够在一个节拍内完成，所以一拍的宽度通常取得大于传送时间。应该懂得，微语句中出现箭头符号始终有执行微操作所引起的传送延迟。

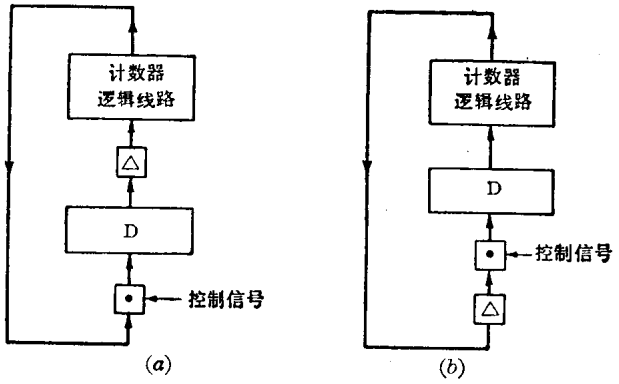

图 1.2 寄存器传送线路中的延迟

(a) 寄存器 D 的输出端加延迟；(b) 寄存器 D 的输入端加延迟

寄存器之间的传送延迟有时在实际使用中是有好处的。例如，“计数器加”微操作

$$D \leftarrow \text{countup D} \tag{1.14}$$

从图 1.2 可看出，是将寄存器 D 的输出信号送入计数器逻辑线路，然后再

将逻辑线路的输出重新加到寄存器 D。为了保证正确地执行此微操作，从寄存器输出端到输入端的信号通路上应有延迟。这延迟可以是逻辑线路的寄生延迟，但较为经常的是在信号通路上加延迟元件。图 1.2(*a*)和(*b*)分别为寄存器 D 的输出端和输入端加延迟的情况。

微操作分以下几类：

送常数微操作，

简单传送微操作，

单边微操作，

双边微操作。

这四种类型微操作的每一类的例子列于表 1.3。须注意，微操作中使用的常数不是八进制的就是二进制的。

1.2.4 条件微语句

若一个或几个微操作仅在满足一定条件时执行，则这微操作用条件微语句写出。有两种结构的条件微语句，即 IF-THEN 和 IF-THEN-ELSE，例如：

Register, G, F(0—5), C(0--14), D(0—14),

IF(G=0) THEN(F←10),

IF(G≠1) THEN(C←0, D←0)ELSE(F←11)。 (1.15)

上述条件微语句中，利用符号=(相等)或≠(不等)写出条件 G=0 或 G≠1。如果条件成立，则放在 THEN 和 ELSE (若有 ELSE 的话)两个字之间的微语句同时被执行。因此，这两个字之间的微语句书写的次序无多大关系。当条件不满足时，执行在 ELSE 字后面的微语句；若无 ELSE 结构，则相当于不执行任何动作。

条件微语句中可以出现另一个条件微语句。例如，在下述微语句中：

Register, A(2—1),

IF(A(1)=0) THEN (A(1)←1)

ELSE (IF(A(2)=0) THEN (A(2)←1, A(1)←0)

ELSE (A←0)) (1.16)

在第一个 ELSE 后面是另一个条件微语句。

(1.16)中对于个别位写出的条件，也能对于位组给出。例如，在条件微语句

Register, C(5—0), $ 计数器

IF(C=77_8)THEN(……) (1.17)

中，列出的条件是检验计数器 C 是否同 77_8 相符，这实际上检验了 6 位。

条件也可以规定为只能取 0 (假)或 1 (真)的逻辑变量。因此，在列出条件表达式时可使用逻辑算符。例如，X(1)=1、Y(2)=0 和 START 开关置于 ON 位置的条件可写成

IF((X(1)=1)*(Y(2)=0)*(START=ON))THEN(……)。 (1.18)

当使用逻辑算符时，条件表达式显得比较复杂，同时，较为冗长，读起来也困难。但是，如果分别将每个条件写成布尔变量的形式，则条件表达式就能得到简化。因此，如果我们

以 X(1) 代替 X(1)=1 或 X(1)≠0,

以 Y(2) 代替 Y(2)=0 或 Y(2)≠1,

以 START(ON) 代替 START=ON,

则上述条件表达式就获得如下紧凑的形式：

IF(X(1)*Y(2)*START(ON))THEN(……)。 (1.19)

这种简化方法在下述那种情况下就不值得使用，

X(1)≠Y(2)，

C=77_8， (1.20)

F(0—2)=R(4—6)，

这里是单独两位比较，或是位组比较，或是与常数比较。

1.2.5 专用算符

表 1.1 列出的基本算符不能适用于所有微操作的说明。在这种情况下就要定义专用算符。由于算符是和逻辑线路相当的，故专用算符的定义包含了逻辑线路的说明。定义专用算符时要取一个符号名，它应区别于所有其他算符的名。

专用算符是用 CDL 语言的表达式写出。例如，我们来看一个两位计数器，它给出 00、10、11、01、00、…序列。我们给这计数器赋以名 ct2。算符 ct2 定义如下：

```
Operator,  D←ct2 D(0—1),                                   (1.21)
/begin/    IF(D=00)THEN(D←10),
           IF(D=10)THEN(D←11),
           IF(D=11)THEN(D←01),
           IF(D=01)THEN(D←00),
           End of operator
```

在(1.21)中，D 是形式变量，此时它既表示输入，也表示输出。算符定义本身放在标号 /begin/ 和 End of operator (算符结束标志)之间，这个定义由四个条件微语句组成。

专用算符的逻辑线路的定义也可以利用布尔方程写出。例如，有一平行加法器，它将两个无符号二进制数相加，并送出溢出信号。此专用算符的名为 addov，它定义如下：

```
Operator,  OV-A(0—23)=X(0—23) addov Y(0—23),               (1.22)
Terminal,  C(23)=0,
           C(0—22)=X(1—23)*Y(1—23)+Y(1—23)*C(1—23)
                  +C(1—23)*X(1—23),
/begin/    A(0—23)=X(0—23)⊕Y(0—23)⊕C(0—23),
           OV=X(0)*Y(0)+Y(0)*C(0)+C(0)*X(0),
           End of operator
```

在(1.22)中，变量 X 和 Y 是逻辑线路的输入信号，OV 和 A 是输出信号。对于内部变量的定义，即平行加法器中的进位，使用终端说明。

1.3 序　　列

为了完成实际的功能通常需要几个微操作。而为了在电子计算机内实现一系列不同的功能，就需要具备控制大量微操作执行序列的手段。为此，我们引进标号、执行语句和控制序列等概念。