电子数字计算机原理

第二册

北京大学计算机科学技术系 编著

科学出版社

1980

内 容 简 介

《电子数字计算机原理》第二册共分四章，第一章介绍常规控制的计算机的运算器和控制器，有三个附录，介绍加法器设计，两种舍入规则和计算机速度估算的一种方法．第二章介绍微程序控制的计算机中央处理器和微程序设计技术，有四个附录，分别介绍模型机指令表，增码规则，模型机微命令和部分微程序．第三章介绍故障诊断概念，讨论布尔差分对组合线路和时序线路的故障诊断，还介绍奇偶校验与奇偶预测．第四章介绍大型机设计的几个重要问题，如交叉存贮、先行控制、快速运算、数组处理机和多道程序，存贮器管理等．

本书可作为计算机专业的教学参考书，也可供有关部门的科技工作者阅读．

电子数字计算机原理

第二册

北京大学计算机科学技术系 编著

*

科学出版社 出版

北京朝阳门内大街 137 号

中国科学院印刷厂印刷

新华书店北京发行所发行　各地新华书店经售

*

1980 年 11 月第 一 版　开本 850×1168 1/32

1980 年 11 月第一次印刷　印张 22 7/8　插页 3

印数 0001—23,700　字数 606,000

统一书号 15031・247

本社书号 1511・15—8

定价：2.90 元

目　　录

第一章　控制器与运算器

控制器与运算器是电子数字计算机的重要部件之一．随计算机的使用范围不同其逻辑结构有所不同，我们仅就用于科学计算的计算机讨论其控制器与运算器的有关问题．

第一节　数、指令与指令系统

一、数

在计算机的数字系统中，一般设有浮点数、定点数、整数和逻辑数．所有这些数均用一串二进制代码表示，而在机器内部则表现为一串电信号．为了便于对下面的问题展开讨论，我们分别对浮点数、定点数、整数和逻辑数作一简单介绍．

1. 浮点数

浮点数由阶码（j）和尾数（S）组成，它在机器中的形式为

j	S

如果浮点数以 2 为底则可写成下面的表达式

$$N = 2^j \times S$$

浮点数的阶码（j）和尾数（S）各定为多少位取决于对计算机所要求的取值范围和精度．

若一计算机其浮点数的阶码 j 用八位二进制补码表示，其中有一位阶码符号 j_f，七位阶码可写成下面的形式

$$[j]_{补} = j_f\, j_1\, j_2\, j_3\, j_4\, j_5\, j_6\, j_7$$

阶码 j 用补码表示时其取值范围见表 1.1．

这种情况下阶码取值范围是

$$-128 \leqslant j \leqslant 127.$$

表 1.1

	j_f	j_1	j_2	j_3	j_4	j_5	j_6	j_7	十进制数值
正数	0	1	1	1	1	1	1	1	2的127次方
	0	1	1	1	1	1	1	0	2的126次方
	0	1	1	1	1	1	0	1	2的125次方
					⋮				⋮
	0	0	0	0	0	0	0	1	2的1次方
	0	0	0	0	0	0	0	0	2的0次方
负数	1	1	1	1	1	1	1	1	2的−1次方
	1	1	1	1	1	1	1	0	2的−2次方
	1	1	1	1	1	1	0	1	2的−3次方
					⋮				⋮
	1	0	0	0	0	0	0	0	2的−128次方

若一浮点数阶码部份为8位;尾数 S 为40位，其中有一位符号位 S_f, S_w 有39位，用补码形式表示，那么尾数 S_w 的取值范围是

表 1.2

S_f \ S_w	S_1	S_2	S_3……S_{38}	S_{39}	小 数 值
0	1	1	1……1	1	$1-2^{-39}$
0	1	1	1……1	0	$1-2^{-38}$
0	1	1	1……0	1	$1-(2^{-38}+2^{-39})$
⋮			⋮		⋮
0	0	0	0……0	1	2^{-39}
0	0	0	0……0	0	0
1	1	1	1……1	1	-2^{-39}
1	1	1	1……1	0	-2^{-38}
⋮			⋮		⋮
1	0	0	0……0	0	−1

即: $-1 \leqslant S \leqslant 1-2^{-39}$.

注意尾数符号 S_f 表示浮点数 N 是正值还是负值. 上述48位

浮点数N最大正值为

$$N_{max} = 2^{+127} \times (1 - 2^{-39})$$

其最小负值为

$$N_{min} = -2^{+127}$$

那么它对应于十进制数的范围为

$$|N| \leqslant 2^{127} \approx 10^{38.2}$$

(设$2^{127}=10^x$，则$\log_{10}2^{127} = \log_{10}10^x$，因此，$x = 127\log_{10}2 \approx 38.2$)

由此可见，阶码位数决定浮点数的取值范围．阶码位数越多对应的浮点数取值范围越大，反之则小．根据实际情况，8 位阶码基本够用，所以大部分机器采用 8 位阶码，也有少数机器用到 11 位阶码的．

尾数 S_w 的位数主要用来决定浮点数的精度即有效数字的位数．上述的尾数 S_w 为 39 位二进制数，相当于十进制数的 11.7 位，也就是说上例中的浮点数折合为十进制数最多可能有 11.7 位有效数字．计算机中浮点数的精度是由计算对象的要求决定的．

目前常用的浮点数的长度有以下几种(如表 1.3)：

注意一个浮点数的取值范围与有效数字的位数，它们是两个不同概念．前者由阶码长度决定，而后者由 S_w 长度决定．

表 1.3

字长	阶符 j_f	阶码	数符 S_f	尾数	数值范围	精度(有效数字)
24 位	1	7	1	15	$-10^{38.2}$—$10^{38.2}$	十进制 4.5 位
32 位	1	7	1	23	同 上	十进制 6.9 位
48 位	1	7	1	39	同 上	十进制 11.7 位
64 位	1	7	1	55	同 上	十进制 16.6 位

现在有一些计算机的浮点数中阶码用增码表示．

若阶码仍为 8 位，用增码表示，其表达式为

$$N = 2^j \times 2^{128} \times S = 2^{j+128} \times S$$

用补码表示和用增码表示的对应关系如表 1.4．

表 1.4

阶码为补码	阶码为增码	
$j_f\ j_1\ j_2\ j_3\ j_4\ j_5\ j_6\ j_7$	$j_f\ j_1\ j_2\ j_3\ j_4\ j_5\ j_6\ j_7$	取 值
0 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	2 的 127 次方
0 1 1 1 1 1 1 0	1 1 1 1 1 1 1 0	2 的 126 次方
⋮	⋮	⋮
0 0 0 0 0 0 0 1	1 0 0 0 0 0 0 1	2 的 1 次方
0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0	2 的 0 次方
1 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1	2 的 −1 次方
1 1 1 1 1 1 1 0	0 1 1 1 1 1 1 0	2 的 −2 次方
⋮	⋮	⋮
1 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1	2 的 −127 次方
1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	2 的 −128 次方

阶码采用增码表示时，增码的最小值（即全 0 ）正好表示了所能表示的真值的最小值（注意在补码表示阶码时，此时阶码为 10000000，不是全“0”）. 这样判“0”线路简单. 关于增码的运算规则见第二章附录 II.

2. 定点数、整数

数 N 的表示为

S_f	S

S_f 为数的符号 (1 位).

S 为数值部份.

若规定小数点在 S_f 与 S 之间，则 N 为定点数.

若规定小数点在数 N 的最后，则 N 为整数.

采用定点运算的计算机在运算时无需对阶，所以结构简单. 定点数若与浮点数位数相同，由于无阶码部份，因而有效位数多，精度高. 但是定点数所表示的数的范围比浮点制要小得多. 一台计算机是采用浮点制还是定点制主要由机器的规模和应用场合所决定. 通常用于自动控制系统的专用计算机及小型计算机大多采用

定点制，而科学计算用大、中型计算机大多采用浮点制.

机器中存贮器单元号码(即地址码)、变址值和计数器的值，都可以看成是无符号位的正整数.

机器中的浮点数或定点数长度确定之后，整数长度与之相适应，或与其等长，或为它的一半，或为它的双倍长.

用补码表示的 24 位长的定点数与整数的取值情况见表 1.5.

表 1.5

$S_f S_1 S_2 S_3 \cdots\cdots S_{23} S_{24}$	整 数 值	小 数 值
0 1 1 1 …… 1 1	$2^{23}-1=8388607$	$1-2^{-23}$
0 1 1 1 …… 1 0	$2^{23}-2=8388606$	$1-2^{-22}$
⋮	⋮	⋮
0 0 0 0 …… 0 1	1	$+2^{-23}$
0 0 0 0 …… 0 0	0	0
1 1 1 1 …… 1 1	−1	-2^{-23}
1 1 1 1 …… 1 0	−2	-2^{-22}
⋮	⋮	⋮
1 0 0 0 …… 0 0	$-2^{23}=-8388608$	−1

定点数小数点位置　　整数小数点位置

3. 逻辑数

一般说来，逻辑数包括两种. 一种是机器中用来表示状态的二进制代码，如控制触发器，控制器中某些寄存器的信息(条件码 ω，程序状态字……)等等，这些二进制代码只具有数的形式而无正负之分，有的也无大小之分. 另一种是表示一个字符的二进制代码也是逻辑数，它纯粹是编码形式. 由于逻辑数的性质不同于前面的浮点数，定点数，所以它的运算法则也不同.

现在国内使用的一种字符编码规则如表 1.6.

表 1.6

W_4	W_3	W_2	W_1	W_7	0	0	0	0	1	1	1	1
				W_6	0	0	1	1	0	0	1	1
				W_5	0	1	0	1	0	1	0	1
				列 / 行	0	1	2	3	4	5	6	7
0	0	0	0	0	空白 KB	转移 ZY	间隔	0	ω	P	“	p
0	0	0	1	1	序始 XS	机控$_1$ JK_1	!	1	A	Q	a	q
0	0	1	0	2	文始 WS	机控$_2$ JK_2	”	2	B	R	b	r
0	0	1	1	3	文终 WZ	机控$_3$ JK_3	#	3	C	S	c	s
0	1	0	0	4	送毕 SB	机控$_4$ JK_4	¥	4	D	T	d	t
0	1	0	1	5	询问 XW	否认 FR	%	5	E	U	e	u
0	1	1	0	6	承认 CR	同步 TB	∧	6	F	V	f	v
0	1	1	1	7	告警 GJ	组终 ZZ	,	7	G	W	g	w
1	0	0	0	8	退格 TG	作废 ZF	(	8	H	X	h	x
1	0	0	1	9	横表 HB	载终 ZTZ	)	9	I	Y	i	y
1	0	1	0	A	换行 HH	取代 QD	★	:	J	Z	j	z
1	0	1	1	B	纵表 ZB	扩展 KZ	+	;	K	[	k	←
1	1	0	0	C	换页 HY	卷隙 JX	·	<	L	\	l	¦
1	1	0	1	D	回车 HC	群隙 QX	-	=	M	]	m	→
1	1	1	0	E	移出 YC	录隙 LX	○	>	N	↑	n	—
1	1	1	1	F	移入 YR	元隙 YX	/	?	O	_	o	抹掉

二、指　令

一个教练员训练队伍时是靠教练员用声音发出一道道口令(如“立正”、“跑步走”、“卧倒”等)，指挥整个队伍协调地完成一项项动作的. 而计算机完成某种操作是由一串特殊的电信号(称为控制信息)控制机器的某些部件而达到的. 计算机中的这种控

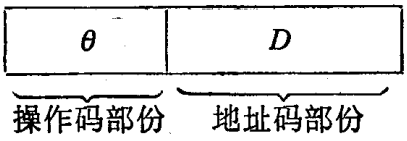
编码举例

字　　符	八进制代码	二进制代码
A	41	100,0001
B	42	100,0010
5	35	011,0101
⋮	⋮	⋮

制信息就是指令．不同的控制信息表示不同的指令，一条指令能使计算机完成某种操作，如算术运算中的＋，－，×，÷，逻辑运算中的逻辑乘、逻辑加，数码的传送，程序的分支(即转移)等．

一条指令完成某种控制功能，它必须包括的信息有两大部份，一部份是告诉计算机做哪一种操作；另一部份是要告诉机器从内存的那些单元取出参与运算的数(即操作数)，这次运算的结果放到何处去．前一部份叫操作码部份，后一部份叫地址码部份．

一条指令也是一串信息，它的特殊性是有控制作用(计算机中的数也是一种信息，但无控制作用)，然而作为信息的普遍性来说，它都可以用一串二进制代码来表示．指令的代码形式可写成



操作码 θ 一般占用五至八位二进制代码．θ 占用位数取决于所需要的指令的多少，若 θ 为 N 位，最多可表示 2^N 种指令．

地址码部份变化较多，随计算机的不同有许多不同的形式．

指令的具体形式与计算机内部结构有密切关系，不同的机器结构有不同的指令形式，我们着重讨论单累加器结构的机器的指令形式．

单累加器结构的机器的运算器一般由三个寄存器（即累加器 L，乘商寄存器 S 和接收寄存器 C)和一个加法器组成［参看本书第一册 293 页运算器框图］．在使用上，累加器 L 处于比较特殊的

地位，一般情况下运算结果存放于 L 中，L 可以与机器其他部件（如内存、变址存贮器等等）互相传送信息，因此 L 与其他两个寄存器 S 和 C 相比，功能显得十分突出．为了强调 L 的突出作用就称其为单累加器结构的计算机．

单累加器计算机的指令形式有三种，即三地址指令、二地址指令和一地址指令．

三地址指令：其指令形式为

θ	D_1	D_2	D_3

其地址码部份有三个地址：

D_1 为第一操作数在内存的单元地址，

D_2 为第二操作数在内存的单元地址，

D_3 为本次运算结果存入内存中去的单元地址．

指令功能是

$$(D_1)\theta(D_2) \rightarrow D_3.$$

假设 $\theta = 000001$ 时为加法指令，为了直观和书写简便，我们用符号“+”代表加法指令的操作码．加法指令的功能可表示成

$$(D_1) + (D_2) \rightarrow D_3$$

其含义是：先把内存中 D_1 单元所存的内容[用 (D_1) 表示]也就是被加数送到运算器的 L 中，再把内存中 D_2 单元的内容（即加数）送到运算器的 C 寄存器，作加法运算，所得的和数存入内存的 D_3 单元中去．

二地址指令：其指令形式为

θ	D_1	D_2

指令功能为

$$(D_1)\theta(D_2) \rightarrow D_2$$

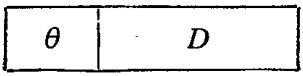
仍以加法指令为例，加法指令功能表示为

$$(D_1) + (D_2) \rightarrow D_2$$

其含义是：把内存中 D_1 单元的被加数送到运算器的累加器

L中，再把内存中D_2单元的加数送入运算器的C寄存器，作加法运算，所得的和数存入内存的D_2单元中去(注意执行加法指令之前D_2单元存的是加数，指令执行之后D_2单元存的是和数，D_1单元的内容发生了变化)．

一地址指令：其指令形式为



其指令功能为

$$(L)\theta(D)\rightarrow L$$

(L)表示L中所存的内容．

仍以加法指令为例，可写成

$$(L)+(D)\rightarrow L$$

其含义是：假设累加器L已事先存入了被加数，再把内存D单元中的加数送到C寄存器，作加法运算，所得的和数存入累加器L．

可以看出，以上三种指令形式都没有用专门的地址码表示下一条指令的地址．这是因为在一般情况下指令总是顺序执行的，因此只要设一个指令计数器，每执行完一条指令后，指令计数器自动加1就可以指出下一条指令地址．

对于单累加器结构的计算机来说，三种指令形式各有什么优缺点呢？

我们从一个简单例子入手，用三种指令形式编程序，比较它们之间的特点．现在计算

$$y=\frac{a\times b+c\times d}{e}-f$$

其中a、b、c、d、e和f是原始数据，它们按表1.7的方式存入内存．

用三地址指令编的程序如表1.8(a)．

计算结果$y=\frac{a\times b+c\times d}{e}-f$存入内存的第$m+13$号单元中，整个计算共用5条指令，执行每一条指令要访问内存4次

表 1.7

单元地址	单元内容	
$m+1$	a	常数单元(原始数据)
$m+2$	b	
$m+3$	c	
$m+4$	d	
$m+5$	e	
$m+6$	f	
$m+7$		工作单元
$m+10$		
$m+11$		
$m+12$		
$m+13$		

表 1.8(a)

指令地址	指令				说明
	θ	D_1	D_2	D_3	
$m+20$	×	$m+1$	$m+2$	$m+7$	乘法: $a \times b \rightarrow m+7$
$m+21$	×	$m+3$	$m+4$	$m+10$	乘法: $c \times d \rightarrow m+10$
$m+22$	+	$m+7$	$m+10$	$m+11$	加法: $a \times b + c \times d \rightarrow m+11$
$m+23$	÷	$m+11$	$m+5$	$m+12$	除法: $\frac{a \times b + c \times d}{e} \rightarrow m+12$
$m+24$	−	$m+12$	$m+6$	$m+13$	减法: $\frac{a \times b + c \times d}{e} - f \rightarrow m+13$

表 1.8(b)

指令地址	指令			说明
	θ	D_1	D_2	
$m+20$	×	$m+1$	$m+2$	乘法: $a \times b \rightarrow m+2$
$m+21$	×	$m+3$	$m+4$	乘法: $c \times d \rightarrow m+4$
$m+22$	+	$m+2$	$m+4$	加法: $a \times b + c \times d \rightarrow m+4$
$m+23$	÷	$m+4$	$m+5$	除法: $\frac{a \times b + c \times d}{e} \rightarrow m+5$
$m+24$	−	$m+5$	$m+6$	减法: $\frac{a \times b + c \times d}{e} - f \rightarrow m+6$

(即一次取指令，二次取数，一次存中间结果或存结果)，执行这一