32位微型计算机原理与接口技术

仇玉章　主编
仇玉章　孙力娟
洪　龙　李爱群　编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书以 Intel 80486 微处理器为背景,讲述 32 位微型计算机原理、汇编语言程序设计和接口技术。全书共 16 章,内容主要包括:80486 微处理器结构、存储系统、80486 基本集指令、汇编语言程序设计、输入/输出系统、中断系统、串行通信、并行通信、DMA 传送、数模和模数转换以及汇编语言高级编程等。

与本书配套的实验指导书《32 位微型计算机原理与接口技术实验指导》同时由清华大学出版社出版。

本书可作为高等院校汇编语言程序设计、微机原理、接口技术等课程的教材,也可供从事微机应用的工程技术人员参考。

书　名:32 位微型计算机原理与接口技术
作　者:仇玉章　孙力娟　洪　龙　李爱群　编著
出版者:清华大学出版社(北京清华大学学研大厦,邮编 100084)
http://www.tup.tsinghua.edu.cn
印刷者:　　印刷厂
发行者:新华书店总店北京发行所
开　本:787× 1092 1/16　印张:26　字数:630 千字
版　次:2000 年 7 月第 1 版　　2000 年 7 月第 1 次印刷
书　号:ISBN 7-302-03880-5/TP・2331
印　数:0001～0000
定　价:00.00 元

前　言

本书编写的指导思想是:立足系统,面向应用,用汇编语言程序设计贯穿各个章节。无论是在汇编语言程序设计还是在接口技术的应用方面,本书内容力求全面,有一定深度,并具有较强的实用性。本书是编者多年教学和科研的经验总结。为了深入理解微机原理和接口技术,还另外编写了一本《32 位微型计算机原理与接口技术实验指导》与本书配套。

本书共分 16 章,内容包括:

第 1、2 章介绍微机系统基础知识。

第 3 章讲述 80486 CPU 结构。

第 4 章介绍 80486 基本集指令。

第 5 章介绍宏汇编语言的若干语法规定。

第 6 章介绍实模式汇编语言程序设计。

第 7 章介绍微机系统的输入/输出,并重点介绍 8254 定时/计数器及其应用。

第 8 章详细介绍微型机中断系统。

第 9 章介绍微机系统串行通信及其应用。

第 10 章介绍并行接口。

第 11 章介绍 DMA 传送及 8237A DMA 控制器。

第 12 章介绍数模和模数转换。

第 13 章介绍 486 微机的存储系统。

第 14 章介绍 80486 微型计算机的基本组成。

第 15 章介绍汇编语言高级编程,是汇编语言程序设计应用型章节。

第 16 章介绍 3 个高版本汇编语言调试软件的使用。

使用本书授课时,建议学时为 40+ 40 学时,其中汇编语言 40 学时,中断与接口技术 40 学时。从微机原理课程教学的基本任务出发,应重点讲授 CPU 结构、汇编语言程序设计、中断系统和微机系统常规接口等章节。

本书由李爱群老师编写第 1 章、第 2 章和第 16 章;洪龙老师编写第 3 章、第 13 章、第 14.3 和第 14.4 节;孙力娟老师编写第 7 章、第 9 章、第 10 章、第 11 章、第 14.1 节和第 14.2 节;仇玉章老师编写第 4 章、第 5 章、第 6 章、第 8 章、第 12 章、第 15 章、第 7.4 节、第 9.3 节和第 10.2 节,并任该书主编。谢秋丽老师完成部分章节的文字录入和图表绘制工作。

由于编者水平有限,书中难免有错漏之处,敬请读者批评指正。

作　者

1999 年 12 月

目　　录

第 1 章　数制和码制

1.1　计算机中的数制

数制是数的表示方法。在日常生活中,最常用的是十进制数。由于用电子器件表示两种状态比较容易实现,也便于存储和运算,因此在计算机中一般采用二进制数。因为二进制数书写格式冗长,不便阅读,所以,在程序设计中又往往使用十六进制数、八进制数、二-十进制数等。

1. 常用计数制

(1) 十进制数

在程序设计中,人们广泛使用十进制数。十进制数的特点是:每一位有 0～9 这 10 种数码,故基数为 10,高位权是低位权的 10 倍,加减运算的法则为“逢十进一,借一当十”。

(2) 二进制数

在计算机内部,所有信息都以二进制数形式出现。二进制数的特点是:只有两个不同的数字符号,即 0 和 1,因此基数为 2,高位权是低位权的 2 倍,加减运算的法则为“逢二进一,借一当二”。

(3) 十六进制数

十六进制数是二进制数的另一种书写格式。十六进制数的特点是:每一位有 0～9 和 A～F 这 16 种数码,因此基数为 16,高位权是低位权的 16 倍,加减运算的法则为“逢十六进一,借一当十六”。

(4) 八进制数

八进制数也是二进制数的另一种书写形式。把 3 位二进制数作为一组,每一组用等值的八进制数(实际上是十进制中的 0～7)来表示。八进制数的特点是:每一位有 0～7 这 8 种数码,因此基数为 8,高位权是低位权的 8 倍,加减运算的法则为“逢八进一,借一当八”。

(5) 二-十进制数

二-十进制数是计算机中十进制数的表示方法,二-十进制数用 4 位二进制数码表示 1 位十进制数。简称为 BCD(binary coded decimal)码。

用 4 位二进制数编码表示 1 位十进制数,有多种表示方法,常用的是 8421 BCD 码,它的表示规则以及与十进制之间的等价关系见表 1.1。

例如:$(3456)_{10} = (0011\ 0100\ 0101\ 0110)_{BCD}$

BCD 码是十六进制数的一个子集,1010～1111 是非法 BCD 码。

2. 数制转换

(1) 任意进制数→十进制数

二进制、十六进制以及任意进制的数转换为十进制数的方法较为简单,根据按权展开式

表 1.1　BCD 码与十进制数的转换

二进制数	十进制数	BCD 码	二进制数	十进制数	BCD 码
0000	0	0000	1000	8	1000
0001	1	0001	1001	9	1001
0010	2	0010	1010	10	非法 BCD 码
0011	3	0011	1011	11	非法 BCD 码
0100	4	0100	1100	12	非法 BCD 码
0101	5	0101	1101	13	非法 BCD 码
0110	6	0110	1110	14	非法 BCD 码
0111	7	0111	1111	15	非法 BCD 码

把每个数位上的代码和该数位的权值相乘,再求累加和即可得到等值的十进制数。如:

$$(1101.11)_2 = 1\times 2^3 + 1\times 2^2 + 1\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2} = (13.75)_{10}$$

$$(731)_8 = 7\times 8^2 + 3\times 8^1 + 1\times 8^0 = (473)_{10}$$

$$(E5A)_{16} = 14\times 16^2 + 5\times 16^1 + 10\times 16^0 = (3674)_{10}$$

(2) 十进制数→二进制数

当十进制数转换为二进制数时,根据该十进制数的类型来决定转换方法。

① 十进制整数→二进制数

方法为:“除 2 取余”,即十进制整数被 2 除,取其余数,商再被 2 除,取其余数,……,直到商为 0 时结束运算。然后把每次的余数按倒序规律排列就得到等值的二进制数。如:

$$N = (11)_{10} = (1011)_2$$

运算过程为:

11÷2 = 5　　余数 = 1　……D_0

5÷2 = 2　　余数 = 1　……D_1

2÷2 = 1　　余数 = 0　……D_2

1÷2 = 0　　余数 = 1　……D_3

所以　$N = D_3D_2D_1D_0 = (1011)_2$

② 十进制纯小数→二进制数

方法为:“乘 2 取整”,即把十进制纯小数乘以 2,取其整数(不参加后继运算),乘积的小数部分再乘以 2,取整……,直到乘积的小数部分为 0。然后把每次乘积的整数部分按正序规律排列,即为等值的二进制数。如:

$$N = (0.8125)_{10} = (0.1101)_2$$

运算过程为:

0.8125×2 = 1.625　　乘积的整数部分 = 1……D_{-1}

0.625×2 = 1.25　　乘积的整数部分 = 1……D_{-2}

0.25×2 = 0.5　　乘积的整数部分 = 0……D_{-3}

0.5×2 = 1.0　　乘积的整数部分 = 1……D_{-4}

所以　$N = (0.1101)_2$

有些纯小数,不断地“乘 2 取整”也不能使其乘积的小数部分为 0,此时只能进行有限次

运算,根据需要取其近似值。

③ 十进制带小数→二进制数

方法为:整数部分“除 2 取余”,小数部分“乘 2 取整”,然后再进行组合。例如:

$$(11.8125)_{10} = (1011.1101)_2$$

(3) 二进制数→十六进制数

以小数点为界,4 位二进制数为一组,不足 4 位用 0 补全,然后每组用等值的十六进制数表示。如:

$$(1101110.11)_2 = (0110\ 1110.1100)_2 = (6E.C)_{16}$$

在汇编语言中,十六进制数用后缀“H”表示。所以,$(1A2B)_{16}$应写成 1A2BH。

(4) 十六进制数→二进制数

把十六进制数的每一位用等值的 4 位二进制数来替换,如:

$$(17E.58)_{16} = (0001\ 0111\ 1110.0101\ 1000)_2 = (101111110.01011)_2$$

3. 字符的编码

计算机处理的信息除了数字之外还有字母、符号等非数值数据,字母、符号统称为字符。在微机系统中,键盘输入、打印输出和 CRT 显示的字符最常用的是美国信息交换标准代码,即 ASCII 码(American Standard Code for Information Interchange)。标准 ASCII 码用 7 位二进制数作为字符的编码,但由于计算机通常用 8 位二进制数代表一个字节,故标准的 ASCII 码也写成 8 位二进制数,但最高位 D7 位恒为 0。D6～D0 位代表字符的编码。表 1.2 为字符的 ASCII 码表。

表 1.2 标准 ASCII 码字符表

H / L	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENG	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{

续表

H / L	000	001	010	011	100	101	110	111
1100	FF	FS	,	〈	L	\	l	©¦
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	〉	N	↑	n	～
1111	SI	US	/	?	O	←	o	DEL

注：H 为高 3 位，L 为低 4 位。

NUL	空	DLE	数据键换码	SOH	标题开始
DC1	设备控制 1	STX	正文开始	DC2	设备控制 2
ETX	正文结束	DC3	设备控制 3	EOT	传输结束
DC4	设备控制 4	ENG	询问	NAK	否定
ACK	认可	SYN	同步字符	BEL	报警(可听见声音)
ETB	信息组传送结束	BS	退一格	CAN	作废
HT	横向制表	EM	纸尽	LF	换行
SUB	减	VT	纵向制表	ESC	换码
FF	走纸控制	FS	文字分隔符	CR	回车
GS	组分隔符	SO	移位输出	RS	记录分隔符
SI	移位输入	US	单元分隔符	SP	空格
DEL	删除				

1.2 计算机中的码制

计算机中使用的数有无符号数和有符号数两种。在计算机中表示一个有符号数的最常用的方法是：把二进制数的最高一位定义为符号位，符号位为 0 表示正数，符号位为 1 表示负数，这样就把符号“数值化”了。有符号数的运算，其符号位上的 0 或 1 也被看作数值的一部分参加运算。

通常，把用“+”、“-”表示的数称为真值数，把用符号位上的 0、1 来表示正、负的数称为机器数。机器数可以用不同的方法来表示，常用的有原码、反码和补码表示法。

1. 机器数的原码、反码和补码

数 X 的原码记作$[X]_{原}$，反码记作$[X]_{反}$，补码记作$[X]_{补}$。

例如，当机器字长 n= 8 时有

	符号		符号位	
	↓		↓	
设真值数	X = + 5= + 0000101	原码机器数写成	$[X]_{原}$ =	00000101
	X = - 5= - 0000101		$[X]_{原}$ =	10000101
	X = + 0= + 0000000		$[X]_{原}$ =	00000000
	X = - 0= - 0000000		$[X]_{原}$ =	10000000

设真值数 $X=+5=+0000101$　　反码机器数写成 $[X]_{反}=00000101$

$X=-5=-0000101$　　$[X]_{反}=11111010$

$X=+0=+0000000$　　$[X]_{反}=00000000$

$X=-0=-0000000$　　$[X]_{反}=11111111$

设真值数 $X=+5=+0000101$　　补码机器数写成 $[X]_{补}=00000101$

$X=-5=-0000101$　　$[X]_{补}=11111011$

$X=+0=+0000000$　　$[X]_{补}=00000000$

由上述例子可以得出以下结论:

① 机器数比真值数多一个符号位。

② 正数的原、反、补码与真值数相同。

③ 负数原码的数值部分与真值相同; 负数反码的数值部分为真值数按位取反; 负数补码的数值部分为真值数按位取反末位加 1。

④ 没有负零的补码, 或者说负零的补码和正零的补码相同。

⑤ 由于补码表示的机器数更适合运算, 为此, 计算机系统中负数一律用补码表示。

⑥ 机器字长为 n 位的原码数, 其真值范围是 $-(2^{n-1}-1)\sim+(2^{n-1}-1)$;

机器字长为 n 位的反码数, 其真值范围是 $-(2^{n-1}-1)\sim+(2^{n-1}-1)$;

机器字长为 n 位的补码数, 其真值范围是 $-(2^{n-1})\sim+(2^{n-1}-1)$。

2. 整数补码的运算

为了弄清楚补码数是怎样进行加减运算的, 先要引进几个概念。

(1) 模

模是计量器的最大容量。一个 4 位寄存器能够存放 0000～1111 共计 16 个数, 因此它的模为 2^4。一个 8 位寄存器能够存放 0…0～1…1, 共计 256 个数, 因此它的模为 2^8, 以此类推, 32 位寄存器的模是 2^{32}。

(2) 有模的运算

凡是用器件进行的运算都是有模运算。例如, 利用 32 位的运算器, 当运算结果大于等于 2^{32}的时候, 超出的部分被运算器自动“丢弃”(保存在进位标志寄存器中)。

(3) 求补运算

以下是一个由真值求补码的例子, 机器字长 n= 8。

设　　$X=+75$, 则 $[X]_{补}=01001011$;　　$X=-75$, 则 $[X]_{补}=10110101$。

即: 对 $[+X]_{补}$ 按位取反末位加 1, 就得到 $[-X]_{补}$。

对 $[-X]_{补}$ 按位取反末位加 1, 就得到 $[+X]_{补}$。

因此,“求补运算”就是指对一补码机器码进行“按位取反, 末位加 1”的操作。通过求补运算可以得到该数负真值的补码。

鉴于补码数具有这样的特征, 用补码表示有符号数, 则减法运算就可以用加法运算来替代, 因此, 在计算机中只需设置加法运算器就可以了。

(4) 整数补码的运算

补码加法的规则是:

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}$$

其中 X, Y 为正负数皆可, 符号位参加运算。

补码减法的规则是:

$$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$$

其中 X, Y 为正负数皆可, 符号位参加运算。

当真值满足下列条件时, 应用上述规则就可得到正确结果:

$$-2^{n-1}\leq(X, Y, X\pm Y)<2^{n-1}$$

其中 n 为字长, 运算以 2^n 为模。

【例 1.2.1】 设 X = 66, Y = 51, 以 2^8 为模, 补码运算求 X ± Y。

解: 因为 $[X]_{补}=01000010$, $[Y]_{补}=00110011$, $[-Y]_{补}=11001101$

$$\begin{array}{rr} [X]_{补} & 01000010 \\ +)\ [Y]_{补} & 00110011 \\ \hline [X+Y]_{补} & 01110101 \end{array} \qquad \begin{array}{rr} [X]_{补} & 01000010 \\ +)\ [-Y]_{补} & 11001101 \\ \hline [X-Y]_{补} & 1\ 00001111 \end{array}$$

↓ 被运算器丢失

所以 X + Y = + 117 X - Y = + 15

【例 1.2.2】 以 2^8 为模, 补码运算求 66 + 99, - 66 - 99。

解: 因为 $[66]_{补}=01000010$, $[99]_{补}=01100011$

$[-66]_{补}=10111110$, $[-99]_{补}=10011101$

$$\begin{array}{rr} [66]_{补} & 01000010 \\ +)\ [99]_{补} & 01100011 \\ \hline [66+99]_{补} & 10100101 \end{array} \qquad \begin{array}{rr} [-66]_{补} & 10111110 \\ +)\ [-99]_{补} & 10011101 \\ \hline [-66-99]_{补} & 1\ 01011011 \end{array}$$

↓ 被运算器丢失

得到 66 + 99 = - 91 - 66 - 99 = + 91

由上两例可以看出, 不论被加数、加数是正数还是负数, 只要直接用它们的补码(包括符号位)相加, 当结果不超出补码表示范围时, 运算结果即为正确的补码。但当运算结果超出补码表示范围时, 结果就不正确了。

3. 无符号数

在处理某些问题时, 若参与运算的数都是正数, 如学生成绩、职工工资、字符编码、内存地址等, 则存放这些数时再保留符号位已没有实际意义。为了扩大寄存器所能表示的数的范围, 则取消符号位。这样, 一个数的最高位不再是符号位而是数值的一部分了, 这样的数被称为“无符号数”。因此,

8 位字长的无符号数其数值范围是 0～255;

32 位字长的无符号数其数值范围是 0～4294967295。

计算机部件只知道寄存器的内容是一串 0、1 代码。也就是说, 只有程序员才能决定一个数的物理意义。

假设一个 8 位寄存器的内容是 $(11111111)_2$。若它是无符号数, 其值等于 255; 若它是补码数, 其真值等于 - 1; 若它是反码数, 其真值等于 - 0。

4. 进位和溢出

例 1.2.1 和例 1.2.2 引出了两个极为重要的概念: 进位和溢出。在介绍了无符号数以后, 我们可以进一步讨论溢出的判断方法。

(1) 进位

运算之后, 符号位向更高位的进位称为进位。进位值无论是 0 还是 1, 都被运算器“丢弃”, 而保存在“进位标志触发器”中。对于有符号数的运算, 进位值不能统计在运算结果之中。对于无符号数的运算, 进位值是结果的一部分, 不能随意丢弃。

(2) 溢出

运算结果超出了寄存器所能表示的范围, 称为溢出。计算机中如何表示溢出呢? 在计算机中设置了一个“溢出标志触发器”, 当参与运算的两个数, 其符号位相同而与结果的符号位相异时, 溢出标志触发器置 1, 否则置 0。

进位和溢出是两个不同的概念, 以加法为例, 两数相加, 有进位不等于有溢出, 无进位也不代表无溢出。进一步讲, 溢出标志为 1, 不一定表示运算结果是错的。因为计算机硬件即运算器仅仅“知道”相加的两个数是一串 0、1 代码, 按照“逢二进一”的法则, 最高位有进位时, 令进位标志为 1, 两数最高位相同而与结果最高位相异时, 令溢出标志为 1, 如此而已。至于相加的两个数是有符号数还是无符号数, 是由程序员指定的, 运算器并“不知道”。

那么程序员如何判断溢出? 如果相加的两个数是有符号数, 运算之后应判断溢出标志, 溢出标志为 1, 运算结果是错误的, 否则结果是正确的。如果相加的两个数是无符号数, 运算后应测试进位标志, 进位标志为 0, 结果正确, 进位标志为 1, 结果是错误的。也就是说, 两数相加是否有溢出, 这和参加运算的数的物理意义密切相关。什么是进位, 什么是溢出, 计算机如何表示进位与溢出, 程序员怎样判断溢出, 这些基本概念对后期的程序设计是非常有用的。

1.3 浮点数基本概念

1. 浮点数

在计算机中如何表示二进制带小数? 由此引出了“浮点数”的概念。

一个二进制带小数可以写成多种等价形式, 例如:

$$\pm 101101.0101 = \pm 1.01101010101 \times 2^{+5}$$

$$\pm 101101.0101 = \pm 0.101101010101 \times 2^{+6}$$

$$\pm 101101.0101 = \pm 0.0101101010101 \times 2^{+7}$$

$$\pm 101101.0101 = \pm 101101010101 \times 2^{-4}$$

写成统一的格式: $\pm S \times 2^{\pm J}$

↓ ↓ ↓ ↓

尾符 尾数 阶符 阶码

(1) 用阶码和尾数两部分共同表示一个数, 这种表示方法称为数的浮点表示法。

(2) 阶码的物理意义: 阶码表示小数点的实际位置。

如: