汇编语言程序设计

(PDP-11)

王世业编著

复旦大学出版社

汇编语言程序设计

(PDP-11)

王世业　　编著

复旦大学出版社

内 容 简 介

本书是一本汇编语言程序设计课程的教科书。它是作者在原教学讲义的基础上，结合教学实践所累积的经验和资料精心编写的。本书选用典型的PDP-11计算机上宏汇编语言MACRO-11，系统地阐述汇编语言程序设计的基本概念和方法，对当今一些高级概念的程序结构作了介绍，并通过实例说明其实现方法。

本书概念清楚，叙理严谨，内容充实。按照教学要求，配有丰富的例题、习题、实习题。编者既为初学者提供具体的实习手段，又力求为汇编程序的开发起指导作用。本书可作为高等院校计算机有关专业的教材或教学参考书，也可供有关科技人员阅读。

汇编语言程序设计
(PDP-11)
王世业 编著
复旦大学出版社出版
(上海国权路579号)
新华书店上海发行所发行　　复旦大学印刷厂印刷
开本787×1092　1/16　印张18.625　字数460千
1987年5月第1版　1987年5月第1次印刷
印数1—9,000
统一书号：13253·057　　定价：3.15元

序　言

本书是一本汇编语言程序设计课程的教科书。它选用PDP-11计算机宏汇编语言MACRO-11，来阐述汇编语言程序设计的概念和方法。本书之作是力求能使读者了解计算机的运行特性，并获得汇编语言程序设计方面的必备知识和技能。

也许读者已熟悉象PASCAL或FORTRAN这样的高级语言，它所以易于为人们接受，是因为它隐藏了计算机硬件和指令系统的具体特征，使它和特定计算机具有无关性。用高级语言编写的程序，具有代数的或类似英语的风格，正是它的最显著优点。虽然如此，学会用机器语言进行程序设计仍是很有价值的。首先，在了解计算机体系结构和计算机运行特性时，或者在了解计算机的新思想或新概念时，一个好的方法是从汇编语言这一级上开始的。其次，仍有相当范围内需要使用特定计算机的具体特性，即要用机器语言进行精细的描述。例如，操作系统中绝大部分核心程序仍是用机器语言写的。此外，在微型机的开发应用中，汇编语言或机器语言的使用也占有相当的份量。

用数字形式的机器语言来编写程序是很不方便的。现在实际上大都使用的是和机器指令一一对应的汇编指令，这种汇编指令允许用符号地址和操作记忆符来表示。虽然用汇编语言编写的程序，也要经过一次转换(通常称为汇编)，但它与高级语言不同，每条汇编指令一般是对应一条机器指令。汇编语言之所以必要，正是因为它完全等效于机器语言，能够精确地描述具体计算机的硬件特性。

由于汇编语言是面向机器的，我们选定较有代表性的PDP-11小型计算机作为"模特儿"。本书在阐述汇编程序设计的一般概念基础上，系统地介绍程序设计的基本方法和技巧。书中运用大量程序实例来说明方法的实现原理，以帮助读者对各种概念的加深理解。同时也使读者在运用汇编语言描述算法、在严密的逻辑思维方面具有一定的素养。

下面简要地说明各章的内容：

第一章介绍有关PDP-11计算机上的数据类型和数据格式，简要说明PDP-11计算机主要结构：存贮器、处理机、I/O设备特性。同时介绍有关汇编程序设计的几个基本概念和一般知识。如果本章的某些内容已在前期课程中涉及，则这些内容也可掠过。

第二章介绍指令类型和格式。通过示例，阐述寻址方式的意义及使用特点，介绍基本指令集的指令技术特性。有些复杂的指令将在以后有关章节中逐步叙述。为了能有效地编写程序，在本章还简要地介绍MACRO-11汇编语言，说明编制程序的规格及要求。

第三章开始阐述基本的程序设计方法。学习程序设计方法的一个合适的途径是通过实例的分析，加深对概念的理解，掌握程序设计的一般方法。在介绍子程序时，先说明堆栈结构及先进后出原理，进而讨论子程序的调用规则。堆栈操作在刷新程序设计技术方面担负重要角色，这将在下一章给于介绍。

第四章是讲解更为高级概念的程序结构及其程序设计方法，即递归子程序、协同子程序、重入子程序的设计方法；浮动程序的概念和实现；宏指令设计的基本内容。本章涉及的这几方

面内容对计算机软件的高一级科目的学习是有帮助的。

第五章是讨论计算机和外部世界的联系，介绍 I/O 设备的物理特性和它们的程序设计。了解中断的概念和它的实现，是探索计算机系统的唯一时机。本章详细地介绍 PDP-11 中断系统的特点和处理中断的基本方法。最后介绍输入输出程序系统的一般原理。

以上各章后都附有一定数量的习题，这些习题大部分是编者在多年的教学实践中精选出来的，在汇编语言程序设计中具有一定的代表性。

第六章的全部内容是为本课程的实习准备的。当教完第二章以后，即可边上机边阅读。本章对每种操作，有实例示范，实用易读。因此，第六章主要作为自学之用。

通常 PDP-11 计算机上都配备有多用户实时操作系统，即 RSX-11M。本章具体指导如何在这个环境下实习，包括如何使用几个软件工具，并说明基本操作的全过程。给出的实习题，可供选用。如果有好的上机条件，教师还可从各章的习题中选取一些作为实习题用。第六章不仅为初学者编写，对那些在汇编语言程序设计方面有经验，而想在 PDP-11 机上进一步开发应用的读者也是合适的。

在 RSX-11M 操作系统环境下的终端 I/O，汇编程序是使用宏指令 QIO$ 实现的，而了解宏指令 QIO$ 要到第四、第五章，但是初始实习在上完第二章即可进行，如何解决这个矛盾？编者是预先设计了几个 I/O 子程序，建立一个用户目标库（最好是把它们装入到系统目标库中），这样就可象高级语言一样，使用最简单的 CALL 语句，很容易在终端上实现程序的 I/O。有关子程序的入口是，字符输入 IN，字符输出 OUT，字符串输出 PRINT，十进制数输出 BTDS，十进制数输入 DTB 等。这几个子程序在附录 G 中被给出。

上机实习是学习汇编语言程序设计的重要手段。编者认为汇编语言程序设计的实习，最好有一台“物理机”，即没有操作系统控制的“裸机”，让读者亲自试验诸如存贮分配、设备的 I/O 操作，或中断程序的实际运行体验。因为在“物理机”一级上能帮助读者理解，构成一个更为复杂的“逻辑机”所需的主要操作特性。“逻辑机”能为用户提供丰富的资源，但却掩盖了机器的本来面目，使初学者缺乏这方面的实际体验。本书的目标是要使读者尽可能多地深入理解这方面的内容。

本书是在编者原讲义的基础上，结合以后的教学实践所累积的经验和资料编写的，但限于水平，无论在内容的构思、组织，还是在对例子的选择上，仍会有不妥或谬误之处，恳请读者批评指正。

在编写本书的过程中，得到了钱家骅、何永保、楼荣生、于玉等同志的宝贵支持。王春森同志审阅了全稿，并提出了许多有益的意见。在此一并鸣谢。

王世业

1985 年 4 月

目　　录

第一章　数和计算机

电子计算机的基本功能就是进行数的计算和数据的加工。无论计算机从外界接受的是什么数，例如，一个十进制数或一个字符常数，它在计算机中的内部形式都是二进制代码。这是因为计算机中，具有两个物理状态，在电子器件上容易实现而选择二进制。然而数的外部特征是各种各样的。诸如二进制、八进制、十进制、十六进制的整数，还有浮点数、字符常数等。

计算机中特定的数据，也可以被解释为指令序列。计算机只有把它内部形式的代码解释成特定的表示形式，才能将数据当作某种信息使用。

本章介绍的是几种常用的数的表示及它们之间的转换方法。鉴于汇编语言是面向具体计算机的，还要介绍一些关于 PDP-11 计算机组织及有关计算机概念。

1.1　数的表示及其转换

1.1.1　十进制数

人们最常用的是十进制数。十进制数是以 10 为基，任何正整数 N 均可以表达为以 10 为幂的多项式，其一般形式为：

$$N = a_k*(10)^k + a_{k-1}*(10)^{k-1} + \cdots + a_1*(10)^1 + a_0*(10)^0$$

其中系数　　$a_i \in \{0,1,\cdots,9\}, i = 0,1,\cdots,k$，且 $a_k \neq 0$

把 N 简记为：

$$N = a_k a_{k-1} \cdots a_1 a_0$$

称 $a_k a_{k-1} \cdots a_1 a_0$ 为十进制数，它是数 N 的十进制表示。

在十进制数里，一个数字的值取决于它在整个数中的位置。

例如：

$$\begin{aligned} 347 &= 3*100 \quad (300) \\ &\quad +4*10 \quad (40) \\ &\quad +7*1 \quad (7) \end{aligned}$$

1.1.2　数的二进制和八进制表示

任意一个正整数 N 可以分解成一个以 2 为基的多项式：

$$N = l_n*(2)^n + l_{n-1}*(2)^{n-1} + \cdots + l_1*(2)^1 + l_0*(2)^0$$

其中系数　　$l_i \in \{0,1\}, i = 0,1,\cdots n$，且 $l_n \neq 0$。

把 N 简记为：

$$N = l_n l_{n-1} \cdots l_1 l_0$$

称　$l_n l_{n-1} \cdots l_1 l_0$ 为二进制数，且是数 N 的二进制表示。例如：

$$202 = 1*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0$$

其系数就是二进制数 11001010 的诸位数字，它称为十进制数 202 的二进制表示。

由于二进制数的每一数位，只有 0 和 1 两个状态，它可以用具有两个稳定的物理状态来表示，比如电位的高和低，灯的开和关，电流的大和小等，只要约定其中一种状态为"1"，则另一状态即为"0"。因此几乎全部数字计算机都是建立在二进制记数表示基础上的。

然而，尽管二进制对于计算机来说是何等适宜，何等方便，但人们书写起来终究是太麻烦了。所以常常采用八进制或十六进制来作为二进制的书写形式。因为用八进制或十六进制形式来表示二进制数不仅方便，也容易转换。只需从最低位开始将二进制数划成每三位（八进制）或四位（十六进制）一组，然后以等价的八进制数字或十六进制数字来代替这些位组，这就十分方便地完成了二进制到八进制或十六进制的转换。

【例 1.1】 $N = 010011100001_{(2)}$①有：

$$010\quad 011\quad 100\quad 001_{(2)} = 2341_{(8)}$$

$$0100\quad 1110\quad 0001_{(2)} = 4E1_{(16)}$$

另一方面，八进制数又作为人们习惯使用的十进制数转换为二进制数的过渡形式，因此在程序设计中广泛使用八进制数。今后本书在程序中没有说明或特别标记的数一律认为是八进制数。

读者可以把任何一个数 N 表示成以 8 为基的多项式。仍以十进制数 202 为例。

【例 1.2】 $202_{(10)} = 3*8^2 + 1*8^1 + 2*8^0$

其系数，3，1，2 就组成八进制数 312 的诸位数字，也称 312 是十进制数 202 的八进制表示。

同样，在八进制（或二进制，十六进制）数里，一个数字的值取决于它在整个数中的位置，值得注意的是它的基是 8（或 2，16）而不是 10。许多人在程序中常常搞错常数，就是因为没有注意它的基是什么。初学者更要注意这个问题，因为人们已习惯于使用十进制形式的数。

为了熟悉二进制数、八进制数、十六进制数、十进制数之间关系，在表 1.1 中列出部分数的对照表。

表 1.1　数的十进制、八进制、十六进制、二进制对照表

十进制	八进制	十六进制	二进制	十进制	八进制	十六进制	二进制
0	0	0	0	10	12	A	1010
1	1	1	1	11	13	B	1011
2	2	2	10	12	14	C	1100
3	3	3	11	13	15	D	1101
4	4	4	100	14	16	E	1110
5	5	5	101	15	17	F	1111
6	6	6	110	16	20	10	10000
7	7	7	111	24	30	18	11000
8	10	8	1000	32	40	20	100000
9	11	9	1001	64	100	40	1000000

1.1.3　数制间的转换

考虑到八进制数与二进制数相互转换极为方便，所以主要介绍八进制与十进制数之间的

① 在不易区别数制时，常以一个下标来表示这个数的数制基。

转换方法。

十进制数转换成八进制数(10→8)

十进制数 N 转换成八进制数的方法是：

除 8 取余

即用 N 除 8 得商 N_1 余数 r_0，余数 r_0 就是八进制数的最低位数码；再用第一次除所得的商 N_1 除以 8 得商 N_2 和余数 r_1，余数 r_1 是八进制数的倒数第二位数码；……，继续上面过程，直到最后一次除而得商为 0 时，相应的余数 r_m 就是八进制数的最高位数码。

【例 1.3】 将十进制数 4718 转换成八进制数。

$$
\begin{array}{r|ll}
8 & 4718 & \\
8 & 589 & (N_1)\,r_0 = 6 \\
8 & 73 & (N_2)\,r_1 = 5 \\
8 & 9 & (N_3)\,r_2 = 1 \\
8 & 1 & (N_4)\,r_3 = 1 \\
 & 0 & (N_5)\,r_4 = 1
\end{array}
$$

所以，$4718_{(10)} = 11156_{(8)}$

八进制数转换成十进制数(8→10)

八进制数 $N_{(8)} = f_m f_{m-1} \cdots f_1 f_0$ 展开为多项式：

$$N_{(8)} = f_m * 8^m + f_{m-1} * 8^{m-1} + \cdots + f_1 * 8^1 + f_0 * 8^0$$

因为 $f_i \in \{0,1,\cdots,7\}, j = 0,1\cdots,m$，数制基 8 均可视为十进制记数系统中的数码，所以只要按十进制记数系统中的运算法则计算此多项式值，就可得到 $N_{(8)}$ 的十进制数表示形式。

【例 1.4】 将 $427_{(8)}$ 转换成十进制数

$$
\begin{aligned}
427_{(8)} &= 4*8^2 + 2*8^1 + 7 \\
&= 256 + 16 + 7 \\
&= 279_{(10)}
\end{aligned}
$$

同理，十进制与二进制数之间的转换方法也易于得到。但是一般在手工转换时，常是通过八进制形式的转换而得到。二进制或八进制数转换为十进制数，均是用十进制记数系统的运

表 1.2　2 和 8 的幂

2 的幂	8 的幂	十进数值	2 的幂	8 的 幂	十进数值
2^0	8^0	1	2^9	8^3	512
2^1		2	2^{10}		1024
2^2		4	2^{11}		2048
2^3	8^1	8	2^{12}	8^4	4096
2^4	0	16	2^{13}		8192
2^5		32	2^{14}		16384
2^6	8^2	64	2^{15}	8^5	32768
2^7		128	2^{16}		65536
2^8		256	2^{17}		131072

算规则，其值以 2 或 8 为基的幂构成的多项式值，在表 1.2 中给出一部份 2 和 8 的幂。

【例 1.5】 将二进制数 1011100 转换成十进制数

$$\begin{aligned}1011100_{(2)} &= 2^6+2^4+2^3+2^2\\ &= 64+16+8+4\\ &= 92_{(10)}\end{aligned}$$

或者先把二进制数 1011100 转成八进制数，再由 8 进数转为十进制数，即：

$$\begin{aligned}1011100_{(2)} &= 134_{(8)}\\ &= 1*8^2+3*8^1+4*8^0\\ &= 64+24+4\\ &= 92_{(10)}\end{aligned}$$

任意数制间转换的一般方法

在程序设计中，通常遇到的是二进制、八进制、十六进制、十进制数之间的转换。事实上，对任意数制都是同样的原理给出转换，其一般形式为：

对 R(R>1)进制数

$$a_na_{n-1}\cdots a_1a_0 = a_n*R^n + a_{n-1}*R^{n-1} + \cdots + a_1*R^1 + a_0*R^0$$

其中：

$$a_j\in\{0,1,\cdots,R-1\},j=0,1,\cdots,n \text{ 且 } a_n\neq 0$$

转换成 P(P>1)进制数

$$b_mb_{m-1}\cdots b_1b_0 = b_m*P^m + b_{m-1}*P^{m-1} + \cdots + b_1*P^1 + b_0*P^0$$

其中：

$$b_j\in\{0,1,\cdots P-1\},j=0,1,\cdots,m \text{ 且 } b_m\neq 0$$

有如下两种规则：

(1) 按照 R 进制数字系统中运算规则——**"逢 R 进 1，借 1 当 R"**，对 $a_na_{n-1}\cdots a_1a_0$ 用 **"除 P 取余"**(P 要表示成 R 进制数)的方法逐次求得 P 进制数 $b_mb_{m-1}\cdots b_1b_0$。

(2) 或者按照 P 进制数字系统中的运算规则——**"逢 P 进 1，借 1 当 P"**计算多项式

$$\begin{aligned}&a_n*R^n + a_{n-1}*R^{n-1} + \cdots + a_1*R^1 + a_0\\ &= (\cdots((0*R+a_n)*R+a_{n-1})*R+\cdots+a_1)*R+a_0\end{aligned}$$

的值，计算时要求 $a_j(j=0,1,\cdots,n)$，R 均要表示成 P 进制数，这样多项式的值即为相应的 P 进制数 $b_mb_{m-1}\cdots b_1b_0$。

这个连乘加式子是有用的，我们用程序的方法把十进制数 $a_na_{n-1}\cdots a_1a_0$ 转化为二进制数，通常就是用这个办法。这里 $R=1010_{(2)}$，十进数字 a_j 也以二进制形式从高位到低位逐个做一次乘加操作，直到最低位 a_0 为止，就得到 $a_na_{n-1}\cdots a_1a_0$ 的二进制数。

【例 1.6】 用连乘加形式把十进制数 4718 化成二进制数。为了演示方便，我们用八进制形式来代替二进制数

$$\begin{aligned}&(((0*1010+100)*1010+111)*1010+1)*1010+1000\\ &= (((0*12+4)*12+7)*12+1)*12+10\\ &= ((4*12+7)*12+1)*12+10\\ &= (57*12+1)*12+10\\ &= 727*12+10\end{aligned}$$

$=11156_{(8)}$ （计算机总是以二进制数形式存在和运算的）

$=1001001101110_{(2)}$

事实上每次做同样动作的一个操作，对计算机来说是适宜不过的，从中正是体现出计算机的优越性。这个连乘加式子也正是适应程序设计而有它存在意义。

1.1.4 二进制数的运算

在二进制记数系统中，运算的规则和十进制系统中运算规则相类似，只要记住“逢 2 进 1，借 1 当 2”。因为二进制数只有 0，1 两个数字，其加法乘法表十分简单。

加法表为：

$$0+0=0$$
$$1+0=0+1=1$$
$$1+1=10$$

乘法表为：

$$0*0=1*0=0*1=0$$
$$1*1=1$$

举例说明：

```
    1110
+)   110
------
   10100

    10101
×)   1110
--------
    00000
   10101
  10101
+)10101
---------
 100100110
```

```
    1011
-)   110
------
     101

      11101(商)
    ---------
110) 10110011
       110
    ---------
       1010
        110
    ---------
        1000
         110
    ---------
         1011
          110
    ---------
           101(余数)
```

这些例子告诉我们，数采用二进制表示后，就能利用逻辑代数这个工具，它为计算机逻辑设计提供了方便。

八进制数的运算也可得到类似规则，不过仍然要记住“逢 8 进 1，借 1 当 8”。仅给一个加法例子：

```
  67036
+ 52147
-------
 141205
```

1.2 数在计算机中的表示

存在于计算机内的数据都是二进制代码形式，这些代码的含义则由特定的解释给出，使代码作为有用的数据使用。

1.2.1 负数的表示

无符号整数的二进制形式和机内表示可以是一致的，但是负(整)数在计算机中如何表示呢，或者说数的符号如何表示。根据二进制数码具有0、1两个状态，我们约定正号“+”和负号“-”分别用数码0,1两个状态来表示。这样在计算机中符号也就“数码化”了。比如：

$$N_1 = +10110_{(2)},$$
$$N_2 = -00110_{(2)}$$

在计算机中可以表示为

$$N_1: \ 010110$$
$$N_2: \ 100110$$

为了区别原来的数与它在计算机中的表示形式，把符号“数码化”的数在计算机中表示形式称为**机器码**。原来的数称为机器码的**真值**。例如数$-00110_{(2)}$的机器码是100110，其真值是-00110。机器码的码位比真值的码位多1,因为符号位占1位，即用$n+1$位机器码来表示n位的真值。在数的前面直接冠以数码化的符号位所得到的机器码，通常称为数的“符号-幅值”表示。这样表示使数的正负性很直观①。但这种机器码因对计算机的运算相对复杂而不宜采用。由于数的补码表示有其优越性。因而现在计算机普遍采用补码来表示负数的方法。

此外机器码还有反码表示。虽然反码表示可使运算简化，但由于零的反码表示有二异性，故一般不采用。

1.2.2 数的补码表示

定义 假定具有n位二进制数码的数

$$X = \pm X_{n-1}\cdots X_1X_0$$

其补码定义为

$$[X]_{补码} = \begin{cases} X & 2^n > X \geqslant 0 \\ 2^{n+1} + X & 0 > X \geqslant -2^n \end{cases}$$

记$[X]_{补码} = Y_nY_{n-1}\cdots Y_1Y_0$是$n+1$位机器码。例如(以$n=6$说明)：

$$A = +111101$$
$$B = -101101$$

由定义可以求出其补码

$$[A]_{补码} = 0111101 \qquad (A>0)$$
$$[B]_{补码} = 2^7 + (-101101)$$
$$= 1010011 \qquad (B<0)$$

现对A,B及其补码分别做两个相加运算

① “符号-幅值”表示的机器码又称为数的原码表示。

$$A+B=+111101+(-101101)$$
$$=+010000$$
$$[A]_{补码}+[B]_{补码}=0111101+1010011$$
$$=10010000$$

因为定义规定，补码比原来真值多 1 位，即只能是 7 位，故 10010000 的最高位“1”丢去后即为 0010000。应注意的是，丢去的这个 1，在数值上正好是 2^7，因此在 mod 2^7 意义下做加法，即

$$[A]_{补码}+[B]_{补码}=0010000(\mathrm{mod}2^7)$$

而这时

$$A+B=+010000$$

其补码正好也是 0010000。

则有

$$[A+B]_{补码}=[A]_{补码}+[B]_{补码}(\mathrm{mod}2^7)$$

就是说这个等式成立是有条件的，n + 1 位计算机上两数相加是在 mod 2^{n+1} 定义下求得和数。

因为补码定义是在同余概念下建立的，可以说$[X]_{补码}$与 X 对模 2^{n+1} 是同余的，因此补码的定义亦可表示为：

$$[X]_{补码}\equiv X(\mathrm{mod}2^{n+1})$$

因为

$$B=-101101=-1*2^7+1010011$$
$$[B]_{补码}=1010011=0*2^7+1010011$$

B 对 2^7 相除的余数与$[B]_{补码}$是相等的，所以

$$[B]_{补码}\equiv -101101(\mathrm{mod}2^7)=1010011$$

可以说 B 和$[B]_{补码}$对模 2^7 是同余的。即它们分别用 2^7 去整除这两个整数，所得的余数是相同的。（因为整除定义余数一定大于等于零。）

在同余意义下的整数运算，其符号可以象数值一样参加运算，而且运算结果在 mod2^{n+1} 意义下是相等的，故采用补码系统运算会带来很多方便。

补码的简单求法

设 $X=\pm X_{n-1}X_{n-2}\cdots X_1X_0$

根据补码的定义，当 $X\geqslant 0$ 时

$$[X]_{补码}=X$$

当 $X<0$ 时，由

$$[X]_{补码}=2^{n+1}+X$$

可以推导出

$$[X]_{补码}=1\overline{X}_{n-1}\overline{X}_{n-2}\cdots\overline{X}_1\overline{X}_0+1$$

其中 $\overline{X}_i=1-X_i$，称为对 X_i 取反。请读者推导这个结论。

规则：正数的补码是其本身，负数的补码是把幅值各位取反(0 变 1，1 变 0) 最高位之前冠以 1(2^n)，最后末尾 + 1(2^0)。

计算机代码的各位取反是很方便的，因此要存贮一个负数就可用这个规则。

例如：