VAX-11系列

体系结构手册

中国科学院沈阳计算所111组译

《小型微型計算机系統》编辑部

前　　言

《VAX—11系列体系结构手册》译自美国DEC公司1980年出版的《VAX—11 ARCHITECTURE HANDBOOK》。VAX—11系列是在PDP—11系列基础上发展起来的很有影响的32位超级小型计算机系列，具有指令完备、寻址灵活、数据类型多样、虚拟存储等特点，其结构面向操作系统，具有丰富的软件支持。我们将这本手册全文译出，以便从事计算机研制和应用的科技人员、工程技术人员、高等院校师生等参考借鉴。

参加本资料翻译工作的有张家齐、于开洲（第一、二、三、四章），佟大铁、白景春（第五章、附录E、F、G、术语解释），宋彦德、蔡恩俊（第六、七章），陈柏令、刘希明（第八章、附录A、B、C、D），黄永华（第九章），康宝祥、蔡志正（第十、十一、十二、十三章）。栾贵兴、邹铁坚同志担任全部资料的校对工作。

在翻译过程中，参考了五十六所翻译的78年版VAX—11/780结构手册的部分内容，特致谢意。

由于时间仓促，水平有限，谬误或不确切之处在所难免，切望读者批评指正。

一九八三年八月

目　录

第一章 VAX—11系列

引 言

VAX—11系列是PDP—11系列体系结构的有效扩展。术语VAX就是取自VAX—11系列的一个最显著的特点，即虚地址扩展（Virtual Address eXtension）。VAX—11与PDP—11均以字节寻址，且具有类似的I/O和中断结构，以及相同的数据格式。除大大地扩展了虚地址空间外，VAX—11还提供了更多的指令和数据类型，以及新的寻址方式，还给出了更加完善的存储管理和保护机构，以及辅助进程调度和同步的硬件。

虽然VAX本系列方式指令系统并不与PDP—11兼容，但它仍然是对PDP—11指令系统的逻辑扩展，且共享PDP程序设计的简易性。VAX—11和PDP—11之间的相似性允许人工直接将现存的PDP—11程序转到VAX—11上。大部分现有的用户方式的PDP—11程序不需要VAX—11的扩展功能，都可以毫不改变地在VAX—11所提供的PDP—11兼容方式下运行。

VAX—11是为那些要求能力强、具有精湛的高性能和虚拟存储的计算机系统的应用而设计的。作为一种强有力的计算工具，它能用于那些高速、时间要求严格的场合，亦能适于分时以及种类繁多的商业应用。

VAX系列概念/VAX体系结构

从一开始就确定了一个重要的目标，那就是要确保所有的VAX程序都可在VAX—11系列的每个成员上运行。这种所有系列成员所共同具有的确保软件兼容的属性的集合，统称为VAX体系结构。当新的计算机加入到VAX系列中来时，工程设计工作中的一项重要任务便是致力于保护兼容性。这个过程称之为体系结构控制，它确保为今天的VAX—11计算机编写的程序也能在未来的VAX—11上执行。这本手册提供了程序员为任意一台VAX计算机编写应用程序所应该熟悉的各方面资料。应着重强调，VAX—11体系结构是定义一种环境，而不是其实现。例如，体系结构提供给软件的有指令系统、寻址方式、数据类型等等。而不把内部总线结构、特殊实现的特权寄存器、执行速度等等归于体系结构之内。这些特殊实现的细节资料随每个处理机以适当的硬件手册方式提供。

体系结构目标

VAX—11体系结构是围绕大量特定的面向用户的目标而设计的：

- 虚地址空间的扩展（这是唯一最重要的目标）。曾在24位虚地址（能寻址16M字节）和32位虚地址（能寻址4G字节）之间进行设计选择。由于对更大的虚地址的需求从来都是转向新的体系结构的主要理由（如同VAX—11），故感到24位是过于目光短浅。毫无疑问，需要32位虚地址长度，以便使这种体系结构跨越整个八十年代。
- 最大限度地与PDP—11兼容是与虚地址空间的有效扩展和有效功能的增强相一致的。用下述一些方法满足与PDP—11兼容的目标：

1.用于表示数据的格式相同。

2.用于外部I/O设备的中间格式相同。又有相同的数据格式，这意味着PDP—11和VAX—11系统均可读取全部数据文件。

3.汇编语言语法和助记符基本相同。这非常便于那些受过训练的PDP—11程序员学习编写VAX—11代码。

4.除了易于将PDP—11程序转到VAX—11上运行外，在硬件上，VAX—11结构还含有一个PDP—11兼容方式，这个方式允许毫不改变地运行用户方式的PDP—11程序。例如，允许一个用户为PDP—11编写的旧的应用程序与VAX—11本系列方式程序都能同时在VAX—11上运行。

·通过提供广泛的数据类型和寻址方式，而提高了位效率。

·结构的可扩充性——如此设计指令系统，以便用一种与目前所定义的操作符和数据类型相符的方法，将新的数据类型和操作符包括进去。所选择的指令格式为在将来希望增加功能时留有行之有效的和不受限制的改变余地，因此，体系结构的现行定义并未更多地限制未来的提高。

·由高级语言容易使用的目标分为若干个特定的要求：

1.操作、数据类型和寻址方式的正交性。这意味着，正在执行的操作(例如ADD、SUB等)、数据类型（整数、浮点数等）和寻址方式（直接、变址等）可独立地由编译程序考虑，它使编译程序更快，效率更高，且更容易实现。

2.不强制在长字边界上对界。这意味着，大于一个字节的数据项仍可置于任一字节边界，就如同某些语言中所要求的那样（例如FORTRAN COMMON结构）。

3.可变数量的操作数。通过选择一种指令格式，该指令格式允许每条指令有其自己“固有”数量的操作数，这样便可以有编译程序所需要的那种格式的指令。例如A＋B→C为ADDL3，A，B，C。而A＋B→B是ADDL2 A，B。

4.适当的高级原语。通常属于高级语言的操作，常常直接地构造在硬件中，例如，三地址运算（A＝B＋C），循环控制（FORTRAN DO和BASIC FOR循环为一条指令），以及输出格式编排(EDIT指令可用于COBOL PICTURE语句)。

·另外，增加了指令，使各种应用和操作系统代码效率更高。这类例子有硬件支持的队列，易于访问的可变长度的位字段，以及保存和恢复程序上下文关联的简单指令。

·一个最终的目标是使这种结构更广泛地适合于系统成本、性能和应用。这将允许由单一结构去满足大量用户的需求，而省去了与支持许多不同结构有关的不必要的花费。

表示法约定

本节给出这套手册中通用的表示法约定。

操作表示法

存储器（包括物理存储器和虚拟存储器）的表示法是以图上的最高点表示存储器的最低地址，从上至下逐步向较高的地址发展。图1—1中示出这种概念。

除另有注明外，所有的数量都以十进制表示；十进制在系统中无需标注基数，即缺省基数。而其它数制则将数的基数作为脚标给出，如下例所示：

$56A4C_{16}$

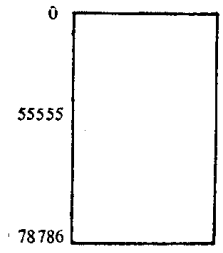

图 1 - 1 存储器编址方案

操作的表示法采用类似ALGOL的格式。例如，ADWC指令（带进位加）表示为：

sum←sum + add + C

它表示将“sum（和）”、“add（加）”和“C（进位）”各量相加，并将结果代替“sum（和）”。附录E中给出这种表示法的全部细节。

区间和范围

在英语中，区间是用“through”表示的，在符号表示形式中用双点“..”表示，并且包含首尾两项。例如，区间0至4或0..4包含整数0，1，2，3，4。

范围是用冒号隔开的一对数，包项首尾两项。例如，位7:3表示包含位7，6，5，4和3的位范围。

不可预测的和未定义的。

表示为不可预测的结果可因时间、执行和在一个执行中的指令而各不相同。管理变动命令（ECO）可更改不可预测的结果。软件决不应依赖于不可预测的结果。

表示未定义的操作可因时间、执行和在一个执行中的指令而各不相同。管理变动命令可更改未定义操作。实际上，操作可在空转到停止系统操作之间变化。非特权软件应避免使用未定义操作。

MBZ和保留字

表示为MBZ（必须为零）的字段决不许由软件用非零值填充。如果处理机在表示为MBZ的字段内遇到一个非零值，通常产生一个保留操作数故障或废弃。

某些可由特权软件存取的字段和值留做数字设备公司专用，并且特权软件不应将非零或保留值置入这些区域。

在某些情况下，一些未分配的值用“留给CSS和用户”来表示。仅这些值可用于非标准应用。已表明为数字设备公司留用的值和MBZ字段仅用于将来扩充标准结构。

第二章　VAX-11体系结构概述

引　　言

本章介绍VAX-11系列体系结构概况。术语“VAX-11系列体系结构”可简称为“VAX结构”。在任何情况中，它都是指汇编语言程序员所见到的那些系统特性，即概念上的结构和功能特点，而与数据流和控制的组织、逻辑设计及物理实现不同。因此，VAX-11系列体系结构定义与程序员在该系列全部处理机上所见到的功能特性一致。一个通用的系列体系结构的重要优点是能够执行任何一个VAX系列成员的软件。VAX-11系列体系结构分为两个不同的方面：应用程序员级和系统程序员级。在本手册中，详细地描述了应用程序员级结构，而超出本手册范围的系统程序员级结构的细节，可随时在相应的硬件手册中找到。

VAX-11系列用户结构部分的那些属性是：

- 4×10^9字节虚地址空间
- 数据类型
- 指令格式
- 寻址方式
- 处理机状态字（处理机状态长字的低位字）
- 本系列方式指令系统中的用户方式指令
- 兼容方式指令系统
- 用户可见的异常处理

那些系统程序员级结构的属性是：

- 特权指令
- 处理机状态长字的高位字
- 进程结构
- 存储管理
- 中断结构和异常处理

那些VAX-11特定处理机的属性（在结构上未被定义为系列的一部分）是：

- 可写控存(WCS)和微机器的各种特点
- 高速缓存的容量和利用率
- 转换缓冲器的容量
- 总线宽度、电气规范、协议、带宽（吞吐量），等等
- 加速器
- 带有软盘（VAX-11/780所特有）的LSI控制台
- 主存极限
- 用于存放诊断所要求的特殊实现的信息的某些处理机寄存器，如错误和状态寄存器。

进程虚地址空间

大部分数据都存放在按8位字节寻址的存储器中。程序员用32位虚地址识别字节单元。因为它不是物理存储器单元的实际地址，所以将这种地址叫做虚地址。在操作系统控制下，由处理机将其转换为实地址。虚地址与物理存储器地址不同，它不是存储器中一个单元的唯一地址。使用同一虚地址的两个程序可能访问两个不同的物理存储器单元，或者，用不同的虚地址访问同一物理存储器单元。

32位虚地址可能编址的整个集合，称之为虚地址空间。可以把它视为一个标注为0至$2^{32}-1$个字节的阵列，一个在长度上近似为4×10^9字节的阵列。这个地址空间被划分为几个指定为某些用途的虚地址集合。这种指定为进程使用的虚地址集合占整个虚地址空间的一半。余下的一半虚地址空间的地址由操作系统占用和保护。

数据类型

一条指令操作数的数据类型指出以多少个存储位为单位进行处理，以及这个单位的解释是什么。处理机的本系列指令系统识别六种基本数据类型：整数和浮点数、字符串，压缩的十进制数、数字串，以及可变长位字段。对于这些数据类型的每一种，操作选择都把数据的长度和数据的解释立即通知处理机。当用户确定了字段的长度和用一个给定字节地址确定其相对位置时，处理机就能对该位字段进行加工处理。

这六种基本数据类型，有若干种变形。表2—1给出有效数据类型一览表，其中某些类型在图2—1中进行了图解说明。整数数据作为二进制值存储。一个整数可以作为一个字节、字、长字存储，或者，在某些情况中，作为一个四倍字存储。一个字节为八位，一个字为两个字节，一个长字为四个字节，一个四倍字为八个字节。处理机可以将一个整数解释为一个带符号的（二进制补码）值，或者解释为一个不带符号的值。符号由高阶位决定。

表2—1 数据类型

数据类型	长度	范围（十进制）	
整数		带符号	不带符号
字节	8位	-128至$+127$	0至255
字	16位	-32768至$+32767$	0至65535
长字	32位	-2^{31}至$+2^{31}-1$	0至$2^{32}-1$
四倍字	64位	-2^{63}至$+2^{63}-1$	0至$2^{64}-1$
浮点		$\pm2.9\times10^{-37}$至1.7×10^{38}	
F—浮点	32位	近似七个十进制位精度	
D—浮点	64位	近似为十六个十进制位精度	
压缩的十进制串	0至16字节（31位数）	数值：每字节两位数 符号：在最后字节的低半字节	
字符串	0至65535字节	每字节一个字符	
可变长位字段	0至32位	取决于说明	
数串	0至31字节（DIGITS）	$-10^{31}-1$至$+10^{31}-1$	
队列	≥两个长字/队列项	0至2×10^9个项	

采用一个带符号的8位余128的阶码和一个规格化的二进制小数存放浮点值，与PDP—11一样，采用四字节和八字节格式。在四字节格式中给出一个有效的24位小数，在八字节格式中给出一个有效的56位小数，而不把规格化位表示出来。四字节格式（称为F—浮点）给出约为七个十进制数位的精度；而八字节格式（称为D—浮点）给出约为16个十进制数位的精度。

压缩的十进制数据以字节串存储。每个字节分为两个4位半字节。每个半字节存放一个十进制数位。首位，或最高有效位存放在首字节的高半字节中；第二位数存放在首字节的低半字节中；第三位数存放在第二个字节的高半字节中，依此类推。数的符号存放在串的最末字节的低半字节中。

字符数据仅仅是一个包含任意二进制数据的字节串，比如，ASCII码。串中的第一个字符存放在首字节中，第二个字符存放在第二个字节中，依此类推。包含十进制数ASCII码的字符串称为数字串。

任何一个数据项的地址都是该数据项所驻留的第一个字节的地址。所有整数、浮点数、压缩的十进制数串和字符数据可自任意字节边界开始存放。然而，位字段则不必从一个字节的边界开始。一个位字段仅是一组连续的位(0—32)，其启始位位置被指定为相对于一个给定的字节地址。本系列指令系统可将一个位字段解释为一个带符号或不带符号的整数。

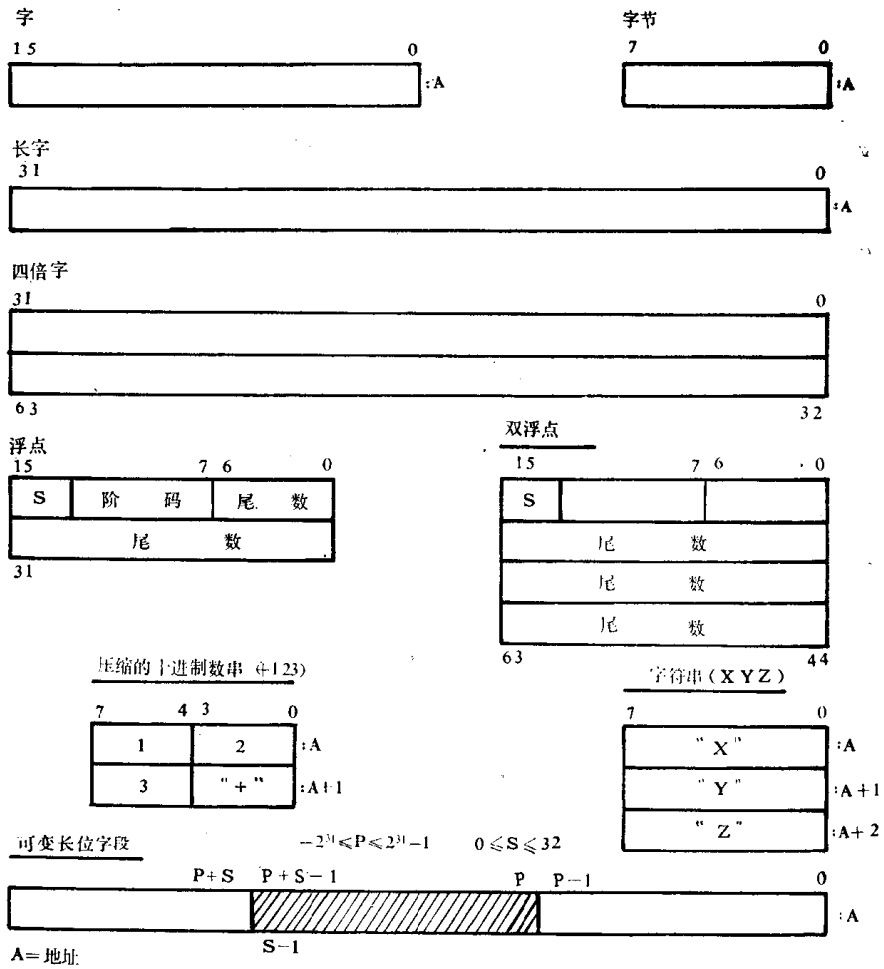

图2—1 数据类型表示法

通用寄存器和寻址方式

通用寄存器是处理机中的一个可用于临时存放数据和地址的单元。汇编语言程序员有16个32位通用寄存器，供本系列指令系统使用。某些寄存器还有特殊的用途。例如，一个寄存器被指定为程序计数器，它含有下一条要执行的指令的地址。三个寄存器被指定用于例行程序链接：栈指针、变元指针，以及栈帧指针。

指令操作数可以装入主存中、通用寄存器中，或者放在指令流自身之中。确定操作数位置的方法称为操作数寻址方式。处理机提供各种各样的寻址方式和寻址方式优化。一种寻址方式是操作数在寄存器中，若干种寻址方式是操作数在存储器中，使用寄存器

- 指向操作数
- 指向一个操作数表
- 指向一个操作数地址表

还有变址寻址方式，该方式是变址修改方式，在存储器中定位一个操作数。最后，还有识别指令流中操作数位置的寻址方式，包括常数和转移指令地址。表2—2中概括列出VAX-11的寻址方式。

表2—2 寻址方式

字面值（立即）	{S↑ I↑}#常数	
寄存器	Rn	
寄存器间接	(Rn)	变址〔Rx〕
自减	-(Rn)	
自增	(Rn)+	
自增间接	@(Rn)+	
（绝对）	@*地址	
位移	{B↑ W↑ L↑}位移量(Rn)地址	
位移间接	@{B↑ W↑ L↑}位移量(Rn)地址	

n = 0至15　　x = 0至14

堆栈、子程序和过程

堆栈是一个连续编址的数据项的阵列，它以后进先出的原则使用一个通用寄存器访问该阵列。数据项从堆栈的低地址端压入和弹出。当压入数据项时，堆栈向较低地址方向生长；弹出数据项时，堆栈向较高地址方向压缩。VAX软件结构的详细说明，请参见附录C(DIGITAL调用标准）。

可以在程序地址空间的任何地方建立堆栈，并用任意一个寄存器指出栈内的现行块。然而，操作系统自动地保留每个进程的部分地址空间，供堆栈数据结构占用。用户软件通过一个被指定为栈指针的通用寄存器访问它的称做用户栈的堆栈数据结构。当用户运行一个程

序映象时，操作系统自动地提供分派给用户栈的地址域。

堆栈是一个功能甚强的数据结构，因为它可用于将变元有效地传送到例行程序。特别是因为处理机可以自动地用栈指针处理例行程序连接，故堆栈结构允许编制可重入例行程序。例行程序也可是递归的，因为变元保留在堆栈之中，以供同一程序的逐次调用。

处理机提供两类例行程序调用指令：用于子程序的和用于过程的。在一般情况下，子程序是用跳转到子程序或转移到子程序指令进入例行程序。

处理机为过程提供的例行程序连接比为子程序提供的更为完善。处理机自动地保留和恢复寄存器的内容，以便保护交叉过程调用。处理机为将变元表传送到被调用的过程而提供两种方法：传送栈内变元的方法，或者传送存储器中另外地方的变元地址的方法。处理机还通过将一个通用寄存器作为指针，去标识过程有其连接数据的栈内位置，从而构成一个过程调用嵌套“日记”(Journal)。这种每个过程的栈数据记录（通称为栈帧）能使其从过程正确地返回，甚至当过程遗留栈内数据时也是如此。另外，用户和操作系统软件用栈帧追踪全部嵌套调用的途径，以便处理错误和调试程序。

栈指针、变元指针和栈帧指针

栈指针是一个专门指派为供栈结构所使用的寄存器。通过使用栈指针，自减寻址方式可将数据项压入栈内，并用自增方式将数据项从栈内弹出。可以在不使用寄存器间接寻址方式的情况下，访问和修改栈顶单元。可用位移量寻址方式访问栈内其余单元。

处理机指定寄存器14做为栈指针，供子程序转移或跳转指令和过程调用指令使用。在调用例行程序时，处理机自动地将紧跟着例行程序之后的那条指令的地址保存在堆栈里。它用程序计数器和栈指针来执行该操作。在进入子程序之前，程序计数器包含继转移、跳转或调用指令之后的那条指令的地址。栈指针包含栈内最后一数据项的地址。处理机将程序计数器的内容压入堆栈。在从子程序返回时，处理机自动地通过将返回地址退栈，恢复程序计数器。

另外，对于过程调用指令，处理机指定寄存器12做为变元指针，寄存器13做为栈帧指针。变元指针用于将变元表地址传送到一个被调用的过程，栈帧指针用于保持被嵌套的调用指令的踪迹。

变元表是一个形式数据结构，它包含正被调入的过程所需要的变元。变元可是实际数值、数据结构地址，或者其它过程的地址。变元表可用下述两种方法之一传送：仅传送其地址；或者传送在用户栈内的整个表。第一种方法用于传送长变元表，或者必须保留的表。第二种方法通常在调用不需要变元的过程，或在动态地构成一个变元表时使用。

当一个过程调用指令发出后，处理机用变元指针将变元传送给过程。在从过程返回时，只要变元已经传送到堆栈，处理机便自动地将变元从过程弹出。

这种方法调用指令的重要意义在于，嵌套调用可被追踪到前面的任意一级。通过采用栈帧指针，调用指令可以始终保持嵌套调用的踪迹。栈帧指针包含着在过程调用期间压入堆栈的各项目的栈地址。过程调用期间压入堆栈的项目集合称为调用帧或栈帧。因为现行帧寄存器的先前内容保存在每个调用帧中，故嵌套的帧形成一个链接的数据结构，当在任何一个过程中出现错误或异常条件时，这种链接结构可以在任意一级处拆开。

处理机状态字

处理机状态字（处理机长字的低位字）是一个专用处理机寄存器，程序用它检测其状

态，并控制同步错条件。处理机状态字含有两组位字段：

- 条件码
- 允许自陷标志

条件码表明一次特定的逻辑操作或算术运算的结果。例如，当减法运算的结果产生一个负数时，该减法指令置位N 位，当结果为零时，它置位Z 位。然后，可用条件转移指令将控制传送给一个处理条件的代码序列。

有两类涉及到用户进程的自陷：跟踪自陷和算术自陷。跟踪自陷由调试程序和性能评价程序使用。算术自陷包括：

- 整数、浮点，或十进制数串溢出，在这种情况中，它们的结果太大，以致于不能在给定的格式中存放。
- 整数、浮点或十进制数串用零除，其中所提供的除数是零。
- 浮点下溢，在这种情况中，其结果太小，以致于不能用给定的格式来表示。

对于算术自陷，有两种方法可用于处理整数溢出，1)浮点下溢，以及2)十进制数串溢出。如果在处理机状态字中允许自陷位被清除，则处理机将忽略整数和十进制数串溢出以及浮点下溢。如果要检测这些条件，则用户可以直接检验条件码（用条件转移指令），或者使能自陷位。如果允许自陷位已被建立，则处理机将象对待异常那样对待整数和十进制数串溢出以及浮点下溢（参见第十二章，异常）。在任何情况下，浮点溢出和用零除自陷总是被允许

条件码

用户程序可以测试算术运算或逻辑操作的结果。处理机为这种用途提供一组条件码和条件转移指令。条件码指出前一次算术运算或逻辑操作是否得出负的或零结果，是否有进位或借位，或者是否产生溢出。有各式各样的条件转移指令：有用于溢出和进位或借位的，也有用于带符号和不带符号的有关测试。

指 令 格 式

本系列方式指令有一种可变长格式，指令可以在任意字节的边界开始。可变长指令格式不仅使代码更为紧凑，它还意味着指令系统易于扩充。对于现存的指令来说，操作码是一个单字节，并在其后有 0 至 6 个操作数说明符，其数目取决于指令。操作数说明符可为1至10个字节长，由寻址方式决定（对于更进一步的细节，请参见第五章：指令格式和寻址方式）。图2—2示出自减方式传送长指令，是一个以操作码开始，后面有两个操作数说明符的字节串。这个例子中采用的起始单元是00003000。当处理机执行完一条指令时，程序计数器含有下一条指令的首字地址。程序计数器的操作对程序员是完全透明的。

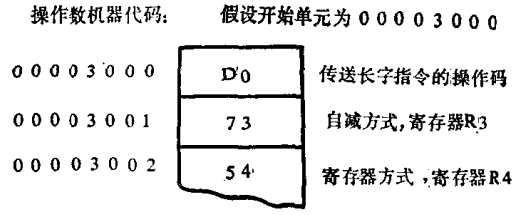

图2—2 自减传送长指令

程序计数器本身可用于识别操作数。汇编程序将许多类型的操作数访问转换成使用程序计数器的寻址方式。使用程序计数器的自增方式，也称立即方式，它用于确定的成行的常数，而不是采用字面值方式寻址。使用程序计数器的自增间接方式，或称绝对方式，用于访问绝对地址。使用程序计数器的位移间接方式用于标识采用从现行单元位移的操作数。

使用程序计数器的寻址允许编制与位置无关代码。与位置无关代码在其被链接后，可在虚地址空间中的任何地方被执行，因为程序链接信息可被指定做为虚地址空间中的绝对地址，并且全部其余地址可相对于现行指令给出。

指令系统

在任何一个时间里，处理机执行两种指令系统中的一种：本系列方式或兼容方式。在本系列方式中，处理机执行一大组变长指令，识别各类数据类型，并使用16个32位通用寄存器。在兼容方式中，处理机执行一组PDP-11 指令，识别整数数据，并使用 8 个16位通用寄存器。本系列方式是机器的基本指令执行状态，而兼容方式则是次要的状态，这两种指令系统联系紧密，并且它们的程序设计特点非常相似。用户进程可包含本系列方式映象程序或者兼容方式映象程序。

本系列指令含有一个操作码(opcode)和零个或多个操作数，这些都通过数据类型和存取方式来描述。本系列指令系统包括240条以上的不同的指令。大多数操作可以用于一种以上的数据类型，可以用任何一种寻址方式寻址。因此，本系列指令系统提供了各种各样的可供选择的指令。

虽然指令的数量多，但是本系列指令系统却是一个非常易于学习的固有的程序设计语言。许多指令直接与高级语言语句相对应，并且也很容易由汇编程序助记符联想到指令功能。

为选择合适的指令，仅需要熟悉各种操作、数据类型和寻址方式。例如，ADD 操作可用于各种长度的整数、浮点数或压缩的十进制操作数，每个操作数都可直接在寄存器（压缩的十进制数除外）中、存储器中找到，或者间接地通过存放在寄存器中或存储器单元中的指针找到。

本系列指令系统

处理机执行的指令系统在操作系统控制下，由本系列方式或兼容方式选择。本系列方式指令系统以 240 个以上的不同的操作码为基础。这些操作码可根据其功能归类，用于一般数据类型操作的指令有：

- 整数和浮点指令
- 压缩的十进制数指令
- 字符串指令
- 位字段指令

用于使用专用数据类型的指令有：

- 队列操作指令
- 地址操作指令
- 通用寄存器操作指令

提供基本程序流控制，并允许用户调用过程的指令有：

- 转移、跳转和选择转移指令
- 子程序调用指令

• 过程调用指令

整数和浮点指令

逻辑和算术运算处理程序指令说明了怎样把一条指令中的操作码、数据类型和寻址方式组合在一起。大多数为整数数据提供的操作也供浮点和压缩的十进制数据使用。例外的是用于整数数据的严格的逻辑操作(如：位清除、置位、补码)，用于整数数据的多字算术运算指令（如：带进位的加/减和扩展的乘和除），用于浮点数据的扩展乘取整和用于浮点数据的多项式指令。

算术运算指令包括双操作数和三操作数两种格式，这就省去了对暂时操作数的传输。双操作数指令用两个操作数中的一个放存结果，比如“置A等于A加B”。三操作数指令有效地实现高级语言语句，其中，用两个不同的的变量求第三个量，比如“置C等于A加B”。三操作数指令可用于整数和浮点数据，而对于压缩的十进制数据也有与之等效的指令。

为了说明指令系统和寻址方式，先看一看 FORTRAN 语言语句：

A(I) = B(I)*C(I)

其中A、B和C是静态分配的REAL*4数组，而I是INTEGER*4，执行这个操作的代码序列是：

```
MOVL    I,R0                        ；传送长字I到一个寄存器
MULF3   B[R0],C[R0],A[R0]           ；三操作数浮点乘
```

扩展除法（EDIV）指令用一个长字去除一个四倍字整数，并产生一个长字的商和一个长字的余数。

扩展乘取整（EMOD）指令以一个浮点数乘上一个带有一个扩展精度浮点数（扩展八位用于9或19个十进制有效数位精度），并分别送出整数部分和小数部分。这条指令对保护整个的三角函数求值的输入精度特别有用。

求多项式计算（POLY）指令用一个采用 Horner 法的系数表求多项式的值。这条指令广泛地用于高级语言的数学库中，例如象Sin和Cos一类的操作。

压缩的十进制数指令

许多用于整数和浮点数据的操作也用于压缩的十进制数串。它们是：

- 传送压缩的十进制数串（MOVP），用于将一个压缩的十进制数串从一区拷贝到另一区；算术移位并舍入压缩型（ASHP），用于在传送时按给定的10的幂，成比例地增大或缩小一个十进制数串，并有选择地舍入其值。
- 比较压缩型（CMPP），用于比较两个压缩的十进制数串。比较有两种方案：三操作数指令（CMPP3）用于等长串，而四操作数指令（CMPP4）用于不同长度的串。
- 转换指令。这些转换指令允许在压缩的十进制数格式和通常所用的数格式之间进行转换。带有后嵌入符的数允许各种符号编码，包括分区的和附加穿孔的。
- 压缩数加（ADDP）和压缩数减（SUBP）用于两个压缩十进制数串的相加或相减。有两种情况，以结果替换加数或减数（ADDP4或SUBP4），或者将结果存放到第三个串中（ADDP6或SUBP6）。
- 乘压缩型（MULP）或除压缩型（DIVP），用于两个压缩的十进制数串相乘或相除，并将结果存放于第三个串中。

另外，压缩的十进制数指令包含一种专用压缩的十进制数串，用于字符串转换指令，这种指

供输出格式安排：编辑指令。

编辑指令

字符串压缩编辑指令（EDITPC）提供输出格式化数据功能。指令将已知的压缩十进制数串转换为采用所选择的模式操作符的字串符。模式操作符允许建立具有下述特点中任意一个的数值输出字段：

- 前零填补
- 前零保护
- 前星号填补保护
- 浮动符号
- 浮动货币符号
- 专用符号表示法
- 插入字符
- 为零时返填空白

字符串指令

对字节串操作的字符串指令是：

- 传递字符串指令（具有转换选择）
- 比较字符串指令
- 单字符搜索指令
- 子字符串搜索指令

对于字符串，有两种基本的传送指令格式。传送字符串指令(MOVC3和MOVC5)简单地将字符串从一区拷贝到另一区。它们优化了块传送操作。五操作数方式为用户提供了填充符，以便指令用于将目的单元填充为一个给定的长度。

传送转换字符串(MOVTC)和传送转换直到换码符(MOVTUC)指令实际上用一个转换表产生新的字符串。比较字符串(CMPC)指令提供逐字符的字节串比较。定位字符（LOCC）和跳过字符(SKPC)指令是为在字符串中搜索某单个字符。匹配字符串(MATCHC)指令与定位指令相似，但它定位多字符子串。MATCHC 检出首次与一个给定的子串相符合的字符串。跨越字符串(SPANC)和扫描字符串(SCANC）指令是用于寻找字符类的成员的检索指令。

变址指令

变址指令(INDEX)为一个定长的数据类型(整数或浮点）的数组计算变址值，和为位字段、字符串和十进制数串数组计算变址值。它把下标、下标的上下界、数组的长度、给定的变址，和为计算变址的目的操作数作为变元。对于采用下标界的高级语言，它体现出计算中的检验范围，并且，它通过消除不变的表达式，使变址计算优化。

可变长位字段指令

位字段指令允许用户定义。存取和修改那些长度和位置都是由用户确定的字段。位置是用基地址或寄存器和带符号的位移确定的。若位字段是存在存储器中，则其位移范围可大到2^{32}—1位（约为16M字节）。若位字段在寄存器中，则位移可大到31。任意长（0至32位）

位字段可用以紧密地存放数据结构标题信息，用于状态码或者建立用户数据类型。位字段指令使用户易于操纵位字段。

插入字段和抽出字段指令将数据存放到位字段中，或者从位字段中取回。比较字段和找第1位指令允许用户测试一个段的内容。比较字段指令抽取一个位字段，然后与一个已知的长字比较。找第1位指令对位字段中第一个置1位或清零位定位。

队列指令

处理机有六种允许容易地构造和维持队列数据结构的指令。用队列指令操作的队列是环形的，双向数据项目链接表。

队列项目的第一个长字含有正向指针，指出队列的后续项目。而下一个长字含有反向指针，指出队列的前一个项目。

提供两种类型队列：绝对的和自相对的。绝对队列使用虚地址指针，而自相对队列使用相对位移指针（更进一步的细节，请参照第四章：数据类型）。

地址操作指令

由于处理机提供多种多样的寻址方式，就很容易通过保持基地址和寄存器中的变址对数据结构进行访向，地址处理亦是频繁的。处理机提供两种不用实际寻址一个单元中的数据便可得到该单元的地址的指令：

- 传送地址（MOVA）指令，它向指定的寄存器或存储器单元中存放字节、字、长字（和浮点），或四倍字（和双浮点）数据地址。
- 压地址（PUSHA）入栈指令，它将字节、字、长字（和浮点），或四倍字（和双浮点）数据存入堆栈。

通用寄存器操作指令

通用寄存器操作指令允许任何一个用户程序在一个操作中保留或装入通用寄存器，检测处理机状态长字，置位或清除处理机状态字中的状态位。

转移、跳转和选择转移指令

控制转移指令的两种基本类型是转移和跳转指令。转移和跳转二者都在程序计数器中装入新的地址。用转移指令时，位移量加到程序计数器的当前内容上，以便获得新的地址。用跳转指令时，装入用户指定的地址，使用一种正常的寻址方式。

由于大多数转移都是转到相对地与现行指令较近的位置，并且转移指令比跳转指令更为有效，所以，处理机给出各式各样的转移指令以供从中选择。有两种无条件转移指令和多种条件转移指令。条件转移指令包括：

- 按位转移指令；
- 置位和位清除转移指令，若已置位或清除，便转移；
- 增加或减少一计数值，并把它与限定值进行比较，按照有关的条件转移指令；
- 计算转移指令，用这条指令，转移可引到依赖于计算值的几个地址中的一个。

条件转移（B）指令允许控制转移到任何一个与处理机状态字中的一个或几个条件码的状态有关的单元。有三组条件转移指令：