2002年新题

中国计算机软件
专业技术资格
和水平考试
辅导丛书

CASL
汇编语言
基础知识与
试题精解

李琳 葛肃昌 主编

- CASL语言基础知识全面介绍
- 高级程序员应试能力训练
- 1990～2002年试题详细解析
- CASL汇编语言文本
  及历年程序员级CASL试题汇编

人民邮电出版社
POSTS & TELECOMMUNICATIONS PRESS

中国计算机软件
专业技术资格
和水平考试
辅导丛书

CASL
汇编语言
基础知识与
试题精解

李琳 葛肃昌 主编

人民邮电出版社

图书在版编目（CIP）数据

CASL 汇编语言基础知识与试题精解 / 李琳主编. —北京：人民邮电出版社，2002.12
ISBN 7-115-10916-8

Ⅰ. C… Ⅱ. 李… Ⅲ. 汇编语言—程序设计—水平考试—自学参考资料 Ⅳ. TP313

中国版本图书馆 CIP 数据核字（2002）第 092589 号

内 容 提 要

本书根据高级程序员考试大纲要求全面介绍了 CASL 汇编语言，并对 1990 年~2002 年高级程序员考试的 CASL 汇编语言试题进行了详细解析。

本书共分七章，第一章与第二章构成基础篇，介绍 CASL 汇编语言基础知识，包括 COMET 计算机系统与 CASL 汇编语言指令。第三章与第四章构成应用篇，介绍 CASL 汇编语言程序设计与数据结构。第五章、第六章与第七章构成应试篇，对 1990 年~2002 年高级程序员考试 CASL 汇编语言试题进行了解析。附录一为 CASL 汇编语言文本，附录二为 1990 年~1995 年程序员考试 CASL 汇编语言试题与标准答案。

本书内容丰富，资料详实，知识体系的安排有利于读者循序渐进地学习 CASL 汇编语言，提高解答试题的能力。本书可作为报考高级程序员考试的考生复习备考的参考书，也可作为对高级程序员考试有兴趣的读者学习 CASL 汇编语言的参考书。

中国计算机软件专业技术资格和水平考试辅导丛书

CASL 汇编语言基础知识与试题精解

◆ 主　　编　李　琳　葛肃昌
责任编辑　王文娟

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
读者热线　010-67132692
北京汉魂图文设计有限公司制作
北京朝阳展望印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：787×1092　1/16
印张：12
字数：284 千字　　2002 年 12 月第 1 版
印数：1-5 000 册　　2002 年 12 月北京第 1 次印刷

ISBN7-115-10916-8/TP・3235

定价：20.00 元

前　言

中国计算机软件专业技术资格和水平考试至今已经进行了十几年，被称为国内含金量最高的计算机资格认证考试。本书面向报考高级程序员级别考试的考生，按照计算机软件专业技术资格和水平考试的大纲要求，系统介绍了有关CASL汇编语言的知识内容，并收集了1990年至2002年的13份试题，对试题进行了详细解析，对试题所涉及的程序进行了解读，目的在于帮助读者掌握CASL汇编语言，提高应试能力。

在计算机软件专业技术资格和水平考试中，CASL汇编语言是一部分比较特殊的知识内容，因为在考试要求的知识内容中，其他各部分都有相应的专业课程，而CASL汇编语言则不同，不但没有相应的专业课程，而且参考书也非常少，即使是计算机专业的考生也必须与其他专业背景的考生一起从头学起。

备考过程中，学习CASL汇编语言的重点在于提高阅读程序的能力。只有提高阅读程序的能力，才能够提高解答试题的能力。因为在解答题目过程中，一个基本的要求是读者能够读懂程序。这要求读者理解试题中程序的算法，理解程序的结构，理解程序中各指令的功能。在有了上面的三重理解之后，读者才能够明确程序中需要实现的功能、已经实现的功能与尚未实现的功能这三部分内容。将上述三部分内容与程序结构结合起来，可以确定尚未实现的功能在程序中位置，而后构造实现这部分功能的CASL汇编语言指令，填写在合适的位置上，完成试题的解答。

本书第一章、第二章介绍CASL汇编语言基础知识，与统编教材相比，本书的特色是不局限于CASL汇编语言指令的介绍，而是在介绍指令的书写格式及基本功能之后侧重于介绍指令使用的注意事项，使用本指令能够实现的其他功能等。这两章的目的是帮助读者在头脑中形成一个关于CASL汇编语言的完整的知识体系，并对各指令的灵活运用有一个初步认识。

第三章与第四章介绍CASL汇编语言程序设计与数据结构。其中第三章侧重于CASL汇编语言程序设计中各种控制结构的实现，通过程序示例使读者对各种控制结构的实现有一个清晰的感性认识；第四章介绍使用CASL汇编语言实现各种数据结构，给出的程序示例只是实现方式之一，并不是唯一的实现方式。这两章的目的不是使读者能够使用CASL汇编语言编写程序，而是旨在提高读者在考试过程中对程序控制结构、复杂运算的识别能力。

第五章、第六章与第七章是1990年以来历年高级程序员级考试CASL汇编语言试题解析。在这部分内容中，首先按照上述的解题思路对试题进行解析，目的在于培养读者科学的解题思路，而后给出标准答案，再对程序进行解读，目的在于提高读者阅读程序的能力。最后，在附录中我们还给出1990年~1995年程序员级考试的CASL汇编语言试题与标准答案，未对该部分试题进行解析，供读者进行能力测试。

在写作过程中，作者一直是从一个考生的角度来理解并介绍CASL汇编语言的，试图尽量缩短本书内容与读者思维的距离，使读者能够更快更好地掌握CASL汇编语言。希望能通过本书的使用，读者能花较少的时间和精力，在考试中取得好成绩。

本书作者的电子邮件地址为sochange@163.net，欢迎读者来信探讨CASL汇编语言的学习。由于时间仓促，作者水平有限，错误疏漏在所难免，希望广大读者能够不吝指正。

编者

2002年10月

目 录

第一章 CASL 汇编语言基础知识

学习目的

掌握 CASL 汇编语言基础知识，了解 COMET 计算机系统，了解 CASL 汇编语言的指令种类与书写格式，为以后学习 CASL 汇编语言构建一个基本的知识框架。

1.1 COMET 计算机系统简介

COMET 计算机是 CASL 汇编语言运行的平台，在物理上它并不存在，只是一台虚拟的计算机。在 COMET 计算机的定义中，COMET 计算机系统包括硬件系统与指令系统两个部分，本节简要介绍 COMET 计算机系统，并介绍 CASL 汇编语言的种类与书写格式，有关 COMET 计算机的硬件系统将在 1.2 节中专门介绍。

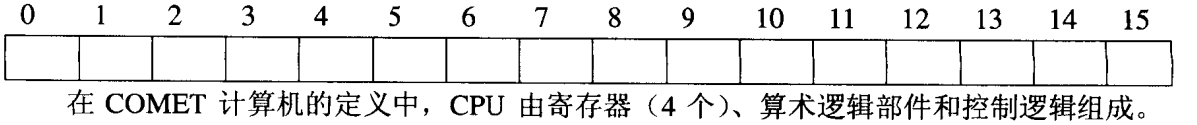
1.1.1 COMET 计算机系统

学习 CASL 汇编语言，首先要了解 COMET 计算机为编程提供了怎样的硬件环境，在编程时有哪些可以使用的硬件资源。由于 COMET 计算机只是一台虚拟的计算机，因此在这里只需要考虑：

- 内存，包括内存容量、字长、编址等；
- 指令系统，包括指令的类型及书写格式；
- 寄存器，包括寄存器的数量及其功能。

作为一台虚拟的计算机，COMET 计算机的指令系统可以映射到其他计算机上，与其他计算机有相似的逻辑结构。现代的计算机系统包括三部分：中央处理器（CPU）、存储器与输入输出设备。由于 COMET 计算机是一台物理上并不存在的计算机，因此在了解 COMET 计算机的时候只需要了解以上跟编程有关的部分就可以了。

在 CASL 汇编语言文本中规定 COMET 计算机是一台字长为 16 位的定点计算机，主存储器的基本存储单元是字，总容量为 65536 字，各个字的地址按照 0000~FFFF（十六进制）编号。一个字的 16 位二进制位采用自左至右的次序编号，即：



在 COMET 计算机的定义中，CPU 由寄存器（4 个）、算术逻辑部件和控制逻辑组成。

寄存器在程序运行时存储计算过程的各种信息；算术逻辑部件的功能是进行所有的算术和逻辑运算；由于 COMET 计算机是虚拟的计算机，它的控制逻辑对于我们是透明的。在 CASL 汇编语言中需要了解的还有两个部件，这就是标志寄存器 FR 与程序计数器 PC，其中 FR 的内容决定于具体指令的运算结果，在编程过程中是一个很重要的因素。

1.1.2 数字与 ASCII 码的机内表示

COMET 计算机是一台定点计算机，不能够处理浮点数。机内数据的字长也是 16 位，数据的二进制位也是从左到右编号，方式与内存各二进制位编号相同。

CASL 汇编语言文本中规定了 3 种 COMET 计算机能够处理的数据，即字符数据、不带符号的整数与带符号整数。

字符数据采用 ASCII 编码，但在写入一个存储单元或寄存器时，存储单元或寄存器的高 8 位（0~7 位）皆为 0，低 8 位（8~15 位）是字符数据的 ASCII 编码，即：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	字符数据的 ASCII 编码							

不带符号的二进制整数的 16 位二进制位全是数值，所以不带符号的二进制整数能表示的数的范围是[0，2^{16}-1]。

带符号整数采用补码表示，其中最高位是符号位，即：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
符号位	数据位														

当符号位为 0 时，该数是正数，当符号位为 1 时，该数是负数。因此带符号整数能表示的数的范围是[-2^{15}，2^{15}-1]。

1.1.3 CASL 汇编语言的指令种类与书写格式

CASL 汇编语言有三种指令：伪指令、宏指令与符号指令。伪指令共有 4 种，即 START、END、DS 和 DC。宏指令有 IN、OUT 和 EXIT 等 3 种。符号指令有 23 种。

CASL 汇编语言的每条指令应书写在一行之内，但是要注意每行最多不能够超过 72 个字符。每条指令包含标号、指令码、操作数与注释等 4 部分，即：

[标号] 指令码 [第一操作数] [，第二操作数] [；注释]

与其他计算机语言一样，标号只是在需要定义的时候书写，必要的时候可以在任何一行前定义标号，该标号也就表示该行指令在程序中的相对地址从而能够在程序的其他部分直接作为符号地址引用。在 DS、DC 两条指令中定义的标号与其他指令中定义的标号意义不同，我们将在后面详细讲解这个问题。注意标号应该从该行的第一个字符开始，标号的第一个字符必须是英文大写字母，后边的字符可以是英文大写字母或数字，标号长度不能超过 6 位。

在没有标号的时候，指令码从该行的第二个字符位置以后的任意字符位置开始；有标号时，则至少需要一个空格将标号与指令码隔开。操作数栏根据具体的指令码决定，与指令码之间至少有一个空格，但有些指令不需要操作数，如伪指令 END 与宏指令 EXIT。

一般来说，CASL 汇编语言指令的第一操作数总是装在一个通用寄存器中（PUSH、POP、

CALL 等指令除外)。第二操作数有三种情况，一是变址寄存器与偏移地址构成有效地址指定的操作数(取地址指令除外)；二是第二操作数直接作为有效地址指定的操作数，如在指令中使用标号、变量等；三是立即数作为第二操作数，这时指令直接引用第二操作数的数值进行运算。需要注意的是，在一条具体的 CASL 汇编语言指令中，当第二操作数属于第一种情况时，第二操作数一般是以“ADR，GRX”这种形式出现在指令中。

注释只是在需要的时候用来说明程序语句的功能，系统认为每行中从第一个“；”到行尾为该行的注释，使用指令 DC 时操作数中出现“；”的情况除外。同时，如果一行的第一字符位置“；”或第一个字符为“；”时，系统自动认为该行为注释。

具体指令的功能与使用，将在第二章中详细讲解。

1.2 COMET 计算机硬件简介

COMET 计算机虽然是一台物理上并不存在的计算机，但在其定义中还是对 COMET 计算机的硬件系统有相当明确的说明。对于基于硬件的汇编语言来说，这些硬件是编程过程中必不可少的资源，了解这些硬件的特点与其在程序中的使用，是学习 CASL 汇编语言的入门课。

1.2.1 寄存器

COMET 计算机有 5 个通用寄存器，分别是 GR0~GR4，每个寄存器的字长都是 16 位，各个二进制位的编号与存储单元(字)的二进制位编号相同，从左到右分别为 0~15，其中第 0 位是最高位，有时候用来表示有符号整数的符号，第 15 位是最低位。

通用寄存器是 CPU 中特定的存储单元，寄存器的读写速度远比存储器的读写速度快，所以经常用来保存程序运行的中间结果或地址等。在 5 个通用寄存器中，GR1~GR4 又可以用做变址寄存器，用来形成程序中的有效地址。而 GR4 同时又兼做栈顶指针寄存器，每当有进栈或出栈操作时，GR4 的值都会有相应的变化，有时可以通过人为的修改 GR4 的值来丢弃系统堆栈区的栈顶元素。

CASL 汇编语言中的寄存器与其他汇编语言的寄存器一样，是编程时重要的资源，这是因为许多重要的参数是通过寄存器来暂存或传递的，使这些寄存器成为运算中重要的参与元素。另外一方面，由于 CASL 汇编语言的特殊性，阅读一个程序时，理解各个寄存器在程序中的角色是理解程序的关键条件。如果不能够把握各个寄存器在程序中的作用，是谈不上理解程序并解答题目的。

在程序中使用寄存器时，通过 GR0~GR4 的寄存器名直接在指令中使用。一般在讨论 CASL 汇编语言时，我们用 GRI 来表示通用寄存器。

指令计数器 PC 记录程序运行过程中指令的地址，它可能是当前正在执行的指令，也可能是即将执行的下一条指令的地址。在顺序执行程序时，它一般指向下一条指令，而当执行转移指令时，需要根据转移条件是否满足来决定 PC 的内容。

系统装入程序时，如果在 START 语句后没有指定程序的启动地址，则将程序的首地址装入 PC，然后执行程序。如果在 START 后指定了程序的启动地址，则将该启动地址装入 PC，然后开始执行程序。但要注意，除了使用跳转指令能够改变 PC 内容外，其他时候 PC 都由系

统自动控制。

标志寄存器 FR 只有两个二进制位，是 CASL 汇编语言编程过程中的另外一个重要元素。FR 的状态值根据某些指令的执行结果由系统来设定。在程序中，LEA、ADD、SUB、AND、OR、EOR、CPA、CPL、SLA、SRA、SLL 和 SRL 等指令的结果都将影响 FR 状态值的变化；而有条件跳转指令的跳转条件是否成立，也是根据 FR 的状态值来判断的。除 CPL 与 CPA 之外，FR 状态值与具体指令运算结果的关系如表 1-1 所示（其中 GRI 表示存储指令执行结果的通用寄存器）。

表 1-1　　FR 状态值与具体指令运算结果的关系

FR 状态值	指令执行后 GRI 状态	指令执行结果的数据特征	比较指令执行情况
00	GRI 中的第 0 位为 0，其余各位不全为 0	正数	大于
01	GRI 中的 16 个二进制位全为 0	零	等于
10	GRI 中的第 0 位为 1	负数	小于

关于 FR 状态值在编程中的具体应用，我们将在后面具体介绍。

1.2.2　存储器

COMET 计算机的存储器共有 65536 个存储单元，存储单元的字长是 16 位，每个单元存储的二进制位组成一个字。存储单元中的 16 个二进制位的编号顺序与编号方式我们在前面已经讲过，即自左至右按照从 0~15 的顺序编号。需要注意的是，COMET 计算机内存的基本单元是字，这一点在处理 ASCII 字符时与别的语言有所不同。在 CASL 汇编语言中，每个 ASCII 字符的存储空间是一个字，但在这个字中，ASCII 字符的 8 位二进制编码只占存储单元的低 8 位二进制位，该存储单元的高 8 位二进制位皆为 0。

存储器是编程时的重要资源，在程序中经常要引用存储单元地址，直接引用存储单元地址的情况比较少见，常见的内存寻址方式是以下两种。

第一种方式是将存储区域首地址放在一个寄存器中，通过变址寄存器来指定存储单元地址。在这个过程中，变址寄存器中所存储的存储区域首地址作为基地址，指定存储单元相对于存储区域首地址的偏移量作为偏移地址，基地址加上偏移地址就构成指定存储单元的有效地址。

另一种方式是在程序中使用 DS 定义并保留一段存储区域，如果在 DS 指令中同时定义了标号的话，标号就成为这段内存首地址的符号地址，这样标号就成为基地址，指定存储单元相对于保留存储区域首地址的偏移量是偏移地址，基地址加上偏移地址就构成指定存储单元的有效地址。这样在程序中就可以以标号为基地址引用这段内存中任一存储单元的地址。如果在 DS 指令中没有定义标号，则该段存储区域的首地址由两部分组成，程序在内存中存放的首地址是基地址，该指令在程序中的相对地址是偏移地址，基地址加上偏移地址构成该段存储区域首地址的有效地址，但在程序中该段存储区域中存储单元地址的表示比较复杂，需要根据具体情况来确定。在 DC 指令中定义的标号与 DS 指令定义的标号有相同的作用。

1.2.3 输入/输出设备

COMET 计算机的输入/输出设备没有明确的定义，输入指令 IN 只是从输入装置输入数据，输出指令 OUT 也只是将指定数据输出到输出装置上。由于 COMET 计算机是一台虚拟的计算机，所以 COMET 计算机的输入/输出装置对用户来说是透明的，用户只需要通过 IN 与 OUT 两条宏指令来执行输入/输出操作即可。在输入时，记录之间的分隔符号不需要用户输入。在输出时，如果需要，记录之间的分隔符号由系统插入。用户只需要指定输入/输出数据存放的内存首地址与输入/输出的字符个数，并不需要也没有必要关心通过什么设备输入/输出以及输入/输出系统是如何工作的等其他细节性问题。

本章小结

通过对本章的学习，我们已经了解了 COMET 计算机系统的基本知识，尤其是与学习 CASL 汇编语言有关的内容，同时也了解了 CASL 汇编语言的概况。这些都是我们今后继续学习 CASL 汇编语言的基础。学习本章的过程中，要重点学习编程中经常使用的 COMET 计算机系统资源，尤其是数与 ASCII 字符的机内表示、标号及内存地址、通用寄存器的使用与 FR 标志寄存器的状态值等内容，应该是学习的重点，在今后的学习中我们会不断看到这方面的内容。

原书空白页

第二章 CASL 汇编语言指令系统

学习目的

通过对本章的学习，了解 CASL 汇编语言的指令系统，掌握重要指令的各种具体应用，并能够阅读用 CASL 汇编语言编写的程序。

2.1 伪指令与宏指令

CASL 汇编语言有 3 类指令，分别是伪指令、宏指令和符号指令。我们将在下一节介绍符号指令。伪指令为汇编程序在汇编过程中提供指示信息，用来指示程序结构及资源的使用等，并不直接产生机器代码。汇编语言源程序中，伪指令以指导性语句的形式独立地出现在源程序中。宏指令是汇编指令能够识别的预先作了定义的程序段，该程序段一旦定义之后，就形成宏指令。宏指令一般是完成比较复杂的、程序化的工作，该工作的过程对用户透明。

CASL 汇编语言有 4 种伪指令，分别是 START、END、DS 和 DC；有 3 种宏指令，分别是 IN、OUT 和 EXIT。在这一节中，我们按照这些指令的功能阐述这些语句的应用。

2.1.1 START、END 和 EXIT

START、END 和 EXIT 三条指令中，前两条指令是伪指令，EXIT 是宏指令，之所以把它们放在一起，主要是因为它们都涉及到程序运行的开始或结束。

伪指令 START 表示程序的开头，在程序的开始必须书写。该指令的格式为：

[LABEL]　START　[LABEL]　；注释

在这里，操作数一栏中的是这个程序中定义的标号，它指出程序的启动地址；如果在指令中没有这个标号，则表示程序从头开始执行。注意标号栏中的标号，这个标号可以作为其他程序进入该程序的入口。

与 START 相对应，伪指令 END 的使用非常简单，该指令表示程序的结束，是用 CASL 汇编语言写程序的固定的最后一句，在程序的结尾必须书写。汇编程序在对源程序进行汇编时，如果遇到伪指令 END，则停止汇编以后的语句。该指令的格式为：

END

伪指令 START 与 END 分别表示程序的开始和结束，而如果想在程序中根据执行情况结束程序的话，则可以使用宏指令 EXIT，该指令表示程序执行的终止，控制返回操作系统，

指令格式如下：

[LABEL]　　EXIT

使用该指令的时候，一般在指令前有一个比较/跳转及程序终止前事务处理语句组。通过比较，来判断程序执行终止的条件是否已经实现。如果已经实现，则进行程序终止前事务处理，如程序执行结果的格式化及输出、恢复部分寄存器的内容，或者是其他与程序执行具体情况有关的操作。如果程序终止的条件尚未实现，则通过跳转语句回避指令 EXIT，继续执行程序。

注意比较伪指令 END 与宏指令 EXIT。END 在汇编时不产生可执行的机器指令代码，可以说，该指令是写给汇编程序的，指示汇编程序停止汇编以后的语句。而宏指令 EXIT 则不同，它是由汇编系统定义的一个程序段，在汇编过程中将形成可执行的机器指令码。

2.1.2　定义常数的宏指令 DC

在源程序中，我们经常要定义常数，如数字、字符、字符串等，这时候就要使用定义常数的宏指令 DC。宏指令 DC 的作用是定义和存储常数，指令格式：

[LABEL]　　DC　　常数

如果在 DC 指令中使用标号，则这个标号就成为指令所定义常数的存储地址的符号地址，或者是代表该指令所定义字符串的首地址。这里的常数可以是各种常数，汇编系统通过常数的书写形式自动识别常数的类型。

如果使用该指令定义数字，则可以定义十进制常数与十六进制常数。定义十进制常数的指令，可以这样书写：

DC　　n

这里的 n 是一个在 COMET 处理范围之内的十进制常数，如 368、-432 等，在汇编过程中，汇编程序将该十进制常数转换成二进制常数存储在一个字中。如果在指令中定义的常数超过了 COMET 的处理范围，则将该常数的低 16 位存储起来。

如果需要定义一个十六进制常数，则可以这样书写指令：

DC　　#h

指令中的“#”是必须书写的，而 h 则是一个十六进制常数。系统对该常数的处理与对十进制常数的处理相同。

连续定义十进制常数或十六进制常数能够起到定义一维数组的效果。如果我们连续定义了若干十进制或十六进制常数，在程序中就可以通过将第一个常数的地址装入寄存器的方式来访问第一个常数，然后通过寄存器内容的递加来逐个访问，相当于高级语言中的一维数组。有关数组的使用与实现，我们将在以后的章节中详细介绍。

定义字符常数与字符串常数的指令格式相同，指令格式如下：

DC　　'字符或字符串'

但定义字符与定义字符串还是有所不同的。定义字符常数的时候定义的是一个字符，只需要一个存储单元来存储该常数，而定义一个字符串时系统会定义一个连续的存储区域，该字符串中的字符按照从左到右的顺序依次存储在存储单元中。需要注意的是，在字符串中不允许出现“'”，而且不允许空字符串的出现。

定义地址常数是 CASL 汇编语言中的一个特殊之处，指令格式如下：

DC [LBALE]

一旦定义了一个地址常数，则操作数栏的标号所对应的地址将作为一个二进制数存储，这样，就可以在系统中对该地址进行处理了。如果操作数栏中的标号在程序中没有定义，汇编将保留地址的定义，并由操作系统处理。使用地址常数能够进行如同高级语言中的指针操作，这一点我们将在以后的章节中详细介绍。

2.1.3 定义存储区域指令 DS

定义存储区域的指令是伪指令 DS，其指令格式如下：

[LABEL] DS 定义的存储单元数

该指令用于保留指定数目连续的存储单元，如果指令中指定的存储单元数目为 0，表示该存储区域不存在，但标号栏中的标号仍然有效，这时它表示下一个字的地址。如果在指令中使用了标号，则标号表示所定义的存储区域的首地址，这时的标号实际上是变量名，也就是所定义存储区域首地址的符号地址。

由于汇编语言中没有如高级语言的简单变量与数组变量的区别，所以在 DS 指令中的存储单元数目就成为区分简单变量与数组变量的关键。如果在 DS 指令中只定义一个存储单元，则该存储单元存储的变量就是一个简单变量。如果在 DS 指令中定义的存储单元数目不止一个，则构成一个一维数组。一个或连续多个 DS 语句，可以构成一个多维数组。

需要说明的是，通过连续的 DC 指令实现的一维数组是一个常数数组，这些数组元素将成为程序处理的对象，如对指定的数据序列进行排序时，需要通过这样的一维数组来将数据序列引入程序。而通过指令 DS 实现的一维数组存储的将是随机变量，如在程序执行过程中输入的数据序列，作为程序处理结果的数据序列等。这两种数组的结合使用，将为 CASL 汇编语言编程提供良好的运算环境。

通过这种方式实现的数组，在操作上还是通过将首地址装入寄存器，然后通过对寄存器的操作来实现的，不同的是在多维数组中需要引入多维数组元素地址的计算公式。

2.1.4 数据输入与输出指令

CASL 汇编语言中没有定义进行输入/输出操作的符号指令，而是通过 IN 与 OUT 两条宏指令来实现数据的输入与输出，关于输入/输出操作的具体实现与工作过程，则是对编程用户透明的。程序中需要进行输入/输出操作时，由 CASL 生成调用操作系统的指令串，但生成的指令串的字数不定。执行宏指令时，通用寄存器的内容保持不变，但标志寄存器 FR 的内容不确定。

实现输入数据的宏指令是 IN，指令格式如下：

[LABEL] IN ALABEL，NLABEL

该指令从输入装置上输入一个记录，记录中的信息（字符）依次按字符数据的形式存放在以标号 ALABEL 为首地址的存储区域中，实际输入的字符个数则存放在标号 NLABEL 为首地址的存储单元中。

输入指令的执行需要宏指令 DS 的配合，最基本的要求是通过 DS 定义用于记录输入数据及输入数据字符个数的存储区域/存储单元。语句执行时，操作系统从输入缓冲区中的起始

地址开始将字符数据输入系统，从指令所指定的存储区域的起始地址开始依次存放输入的字符数据。每个输入的字符数据都按照 ASCII 编码的方式存储，这一点需要注意，尤其是在程序中需要输入数字的时候，ASCII 字符格式的数字需要转换成数字之后才能够参与运算并产生正确的运算结果，尤其是多位数字的输入与转换。

如果输入记录的长度小于指令中指定的存储区域的长度，则指定存储区域的剩余部分内容保持不变；如果输入记录的长度大于指定存储区域的长度，系统将丢弃多余的字符。我们看下面的两组指令：

```
        IN      STOR,   NUM
STOR    DS      8
NUM     DS      1
```

```
        IN      STOR,   NUM
STOR    DS      7
NUM     DS      1
```

在分别执行上述两组指令的时候，我们输入相同的数据“12345678”，第一组指令执行完毕之后，存储区域 STOR 从第一个存储单元起依次是“1”、“2”.....“8”，存储单元 NUM 中是二进制数字 8；而在第二组指令执行完毕之后，存储区域 STOR 从第一个存储单元起依次是“1”、“2”.....“7”，存储单元 NUM 中是二进制数字 7。

实现输出数据的宏指令是 OUT，指令格式如下：

```
[LABEL]     OUT     ALABEL，NLABEL
```

该指令向输出装置输出一个记录，记录中的信息（字符）依次按字符数据的形式存放在以标号 ALABEL 为首地址的存储区域中，需要输出字符的个数则存放在标号 NLABEL 为首地址的存储单元中。

需要注意的是系统将指定存储区域中存放的信息默认为 ASCII 编码的字符，不论存储单元的第 0~7 位是否为 0，都只按照存储单元第 8~15 位所对应的 ASCII 字符输出。这样，在系统中进行数学计算后需要输出结果时，首先必须对运算结果进行格式化，否则将不能正确输出程序结果。

与输入指令不同的是，输出指令所需要的存储区域也可以由常数来定义，这里我们不多讲了，下面说明在程序中如何组织输出结果长度不定的输出操作，算法思想如下：

程序中的运算过程

计算结果的格式化并依次存入 ALABEL

格式化后的运算结果长度存入 NLABEL

```
[LABEL]     OUT     LABEL，NLABEL
[ALABEL]    DS      [足够存储格式化后结果的一个长度]
[NLABEL]    DS      1
```

当然，如果程序的运算结果本身就是一个字符串，则运算结果的格式化工作就可以省略，而且在这种情况下运算结果的长度也可以在运算过程中产生。但如果运算结果是一个数字，格式化的工作就相对麻烦一些，需要将数字按位转换成 ASCII 字符，而这个转换过程也将产生格式化后运算结果的长度。

2.2 符号指令

CASL 汇编语言的符号指令与 COMET 机器指令具有一一对应的关系，每条符号指令在