中国计算机软件专业技术资格和水平考试（高级程序员级）

CASL

汇编语言程序设计

赵立辉　编著

中国电力出版社
www.infopower.com.cn

中国计算机软件专业技术资格和水平考试（高级程序员级）

CASL

汇编语言程序设计

赵立辉　编著

中国电力出版社

内 容 提 要

CASL 汇编语言是中国计算机软件专业技术资格和水平考试高级程序员级的必考内容。本书是讲述 CASL 汇编语言程序设计的专著，内容共分 6 章，详细讲述了 CASL 汇编语言的硬件基础、指令系统和寻址方式，以及怎样利用 CASL 汇编语言进行程序设计。本书第 5 章给出了大量的习题，并在附录中给出了习题答案。在第 6 章，收录了 1990～2001 历年中国计算机软件专业技术资格和水平考试 CASL 考题，并加以分析给出了解答。

本书讲解全面细致，举例典型实用，是准备参加高级程序员级水平考试、已通过初级程序员级或程序员级水平考试人员的一本非常难得的参考书，也可供各高校本科生学习 PC 汇编语言参考。

图书在版编目（CIP）数据

CASL 汇编语言程序设计：中国计算机软件专业技术资格和水平考试高级程序员级/赵立辉编著. —北京：中国电力出版社，2002.10

ISBN 7-5083-1304-6

Ⅰ. C… Ⅱ. 赵… 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 072420 号

中国电力出版社出版、发行

（北京三里河路 6 号 100044 http://www.infopower.com.cn）

汇鑫印务有限公司印刷

各地新华书店经售

*

2002 年 10 月第一版 2002 年 10 月北京第一次印刷

787 毫米×1092 毫米 16 开本 11.5 印张 277 千字

定价 **18.00** 元

前　言

CASL 汇编语言是中国计算机软件专业技术资格和水平考试高级程序员级的必考内容。写一本关于 CASL 汇编语言的书的念头已经有好几年了，我在 1994 年通过了程序员级水平考试，在 1996 年通过了高级程序员级水平考试，对 CASL 汇编语言的研究颇有心得和体会。在这几年中由于工作较忙，一直没有时间坐下来写这本书，现在是学校暑假时间，我终于可以静下心来安安静静地写书，将多年的愿望作一个了结。我希望这本书能赶在今年计算机考试之前顺利出版，为广大考生提供必要的帮助。

本书的读者对象是：

（1）准备参加高级程序员级水平考试的朋友。因为 CASL 汇编语言是高级程序员级水平考试的必考内容，所以如果你是其中的一员，这本书是你最需要的。

（2）通过初级程序员级或程序员级水平考试的朋友。如果你是其中的一员，我想你肯定已将通过高级程序员级水平考试作为新的目标了吧，那就请你阅读本书吧，你会得到你需要的知识。

（3）在校大学生，主要是计算机或电信专业的学生。虽然 CASL 汇编语言是一种抽象的、比较简单的语言，但它将汇编语言中最常用的指令提取出来形成了一套完整的系统。本书中讲解的很多程序设计基本方法对 PC 汇编同样适用。所以如果你有兴趣，不妨也看一看。

向一个优秀的裁缝学习，你会变成一个好裁缝。向一个优秀的厨师学习，你会变成一个好厨师。那么，当你将一个高级程序员的经验变成你自己的经验时，你会变成什么？再没有比这更简单的答案了。既然知道了答案，那你还犹豫什么？打开书开始学习吧！祝你好运！

读者朋友如果发现本书中错误的地方可用下面信箱联系：zlh@mail.jhpu.net。

希望通过本书能使更多的人通过高级程序员级水平考试。

赵立辉

于湖北省荆州市江汉石油学院计算机科学系

目　　录

第 1 章　CASL 汇编语言基础

1.1　COMET 计算机的逻辑结构

计算机通常由输入设备、输出设备、控制器、运算器和存储器五部分组成。控制器、运算器就是我们通常所讲的 CPU，也称中央处理机。它能分析和执行算术与逻辑指令，控制整个计算机的运行。在 CPU 中的寄存器用来存放运算的中间结果。存储器能够存放数据、指令及运算的结果。输入设备、输出设备来完成人和计算机的通信。标准输入设备是键盘，标准输出设备是显示器。

COMET 计算机是一个抽象的计算机，它的逻辑结构见图 1.1。虽然它是一个抽象的计算机，但它具有一般计算机的所有部分。在 COMET 计算机的控制器中有 5 个通用寄存器 GR0、GR1、GR2、GR3、GR4，一个指令计数器 PC 和一个标志寄存器 FR。

注意：我们以后会非常频繁地使用这些寄存器来设计 CASL 汇编程序，因此，一定要熟练掌握它们的用法。

COMET 计算机的内存按字编址(这一点和 PC 机的内存按字节编址不同)，容量是 64KB 个字（地址范围从 0000H 到 FFFFH 或从 00000 到 65535）。关于 COMET 计算机知道这些就足够了。

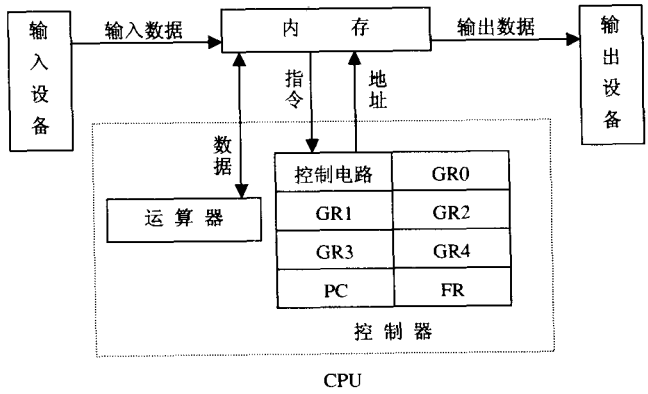


图 1.1　COMET 计算机的逻辑结构

1.2　CASL 汇编语言

1.2.1　CASL 汇编语言概述

CASL 汇编语言是定义在 COMET 计算机上的一种抽象的汇编语言，是计算机汇编语言

最基本的共同部分。它具有语句功能单一、程序格式规范等特点，因此便于汇编试题的标准化，便于统一检测应试者汇编语言的编程水平。

CASL 汇编语言原是 1987 年后日本计算机应用软件人员全国统考中使用的汇编语言文本。目前我国制定的计算机软件人员水平考试大纲中也规定使用 CASL 汇编语言。它是高级程序员的必考科目。CASL 汇编语言文本见附录 A。

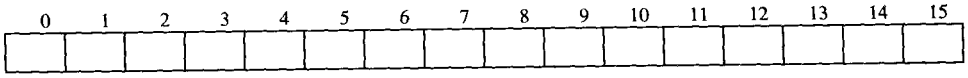
1.2.2 CASL 汇编语言硬件基础

CASL 汇编语言依赖 COMET 计算机的以下特点。

1. 内存储器

COMET 是一台字长为 16 位的定点计算机。主存储器的容量是 65536 字，按编号 0000～FFFF（十六进制）或 00000～65535（十进制）编址。

一个字的 16 位二进制位采用自左至右的次序编号，即：



其中 0～7 位是高 8 位，8～15 位是低 8 位。

一个字的 16 位可以是：

（1）不带符号的二进制非负整数。此时一个字能表示的数的范围是：

$$0 \leqslant X \leqslant 65535$$

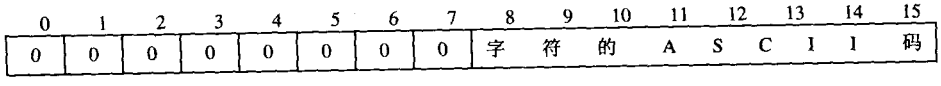
（2）用补码表示的带符号的二进制整数，此时一个字能表示的数的范围是：

$$-32768 \leqslant X \leqslant 32767$$

（3）地址常数。此时一个字能表示的地址写成十六进制时是：

$$0000 \sim FFFF$$

（4）字符数据。此时一个字的高 8 位皆应为零，低 8 位为字符的 ASCII 编码，即：



2. 程序中可访问的寄存器

COMET 具有 5 个通用寄存器 GR（16 位），一个指令计数器 PC（16 位）和一个标志寄存器 FR（2 位）。它们的作用是：

（1）GR（通用寄存器）。5 个通用寄存器的编号为 0、1、2、3、4，分别记为 GR0、GR1、GR2、GR3、GR4。这些通用寄存器用于算术、逻辑、比较、移位等运算，其中 GR1、GR2、GR3、GR4 通用寄存器还兼作变址寄存器。另外，GR4 还兼作栈指针（SP）用，栈指针是存放栈顶地址用的寄存器。

（2）PC（指令计数器）。在执行指令的过程中，PC 中存放着正在执行的指令的第一个地址（一条指令占二个字）。当指令执行结束时，置入下一条将要被执行的指令的第一个字的地址。也就是说，在指令执行结束时，一般是把 PC 的内容加 2，只有在执行转移指令且转移条件成立时，才将转移地址置入 PC 中。地址运算按 65536 取模。

（3）FR（标志寄存器）。在 ADD、SUB、AND、OR、EOR、CPA、CPL、SLA、SRA、SLL、SRL、LEA 等指令执行结束时，根据执行结果，将 FR 置成 00、01、10。它不会因其

他指令的执行而改变。

一般说来 FR 的第一位（左边）表示运算结果的符号（“0”为正，“1”为负）；第二位表示运算结果是否为零（“0”为非零，“1”为零）。ADD、SUB、AND、OR、EOR、SLA、SRA、SLL、SRL、LEA 执行后，FR 的值见表 1.1，CPA、CPL 执行后 FR 的值见表 1.2。

表 1.1　ADD 等执行后 FR 的值

	GR 中的数据		
	正	负	零
FR 的值	00	10	01

表 1.2　CPA、CPL 执行后 FR 的值

	GR 与有效地址比较		
	>	<	=
FR 的值	00	10	01

（4）COMET 的控制方式为顺序控制。指令由 32 位二进位构成，即双字长。

1.2.3　CASL 汇编语言的字符集

汇编语言的主要特点是符号化。符号是由特定的字符组成的，任何一种汇编语言都有它允许的字符集，CASL 汇编语言允许下列字符的集合：

（1）字母。大写字母 A～Z，若输入是小写字母，系统将小写字母和大写字母同样对待，也就是说，CASL 汇编语言对字母的大写和小写不敏感。

（2）数字。0～9。

（3）分隔符。空格（20H），制表符（09H），逗号“,”（2CH），行结束符（回车 CR）（0DH）。

（4）指示符。十六进制指示符“#”，字符串定界符单引号“’”，正负号“+、-”。

1.2.4　CASL 汇编语言的标识符定义

标识符即符号名字。CASL 名字有两类：①系统定义的名字，如加法操作符 ADD，宏指令名 IN，伪指令名 START 等，这类名字可以认为是系统的保留字，用户不得用这些保留字来定义用户自己的名字；②用户自定义名字，这类名字由用户自己定义，长度不要超过 6 个字符，如 BEGIN、NEXT 等，用户自定义名字用来定义标号（即符号地址）。

1.2.5　CASL 汇编语言的常数

CASL 汇编语言中，允许下列 4 种常数：不管是哪一种常数，在内存中占一个字长。

（1）十进制常数。由若干个 0～9 的数字组成。

（2）十六进制常数。由若干个 0～9 的数字或字母 A～F 组成的序列。为了和十进制常数区分，凡在程序中所使用的十六进制常数，必须以#开始，且长度为 4 位，如#000F。

（3）字符串常数。字符串常数是指用单引号“’”括起来的一个或多个字符，其值是各个字符所对应的 ASCII 码，字符串常数一般用伪指令 DC 定义。

（4）地址常数。地址常数是一个已定义的标号。标号所对应的地址值，可作为一个字的二进制数据使用。

第 2 章　CASL 汇编语言的指令系统与寻址方式

2.1　CASL 汇编语言指令系统概述

2.1.1　CASL 汇编语言指令系统组成及分类

CASL 汇编语言的指令系统由 4 种伪指令（START、END、DS、DC）、3 种宏指令（IN、OUT、EXIT）和 23 条符号指令共 30 条指令组成。伪指令见 2.3.1 小节，宏指令见 2.3.2 小节。表 2.1 列出的是 23 条符号指令。

表 2.1　CASL 汇编语言符号指令表

指令类别	名　称	书写格式	
		指令码	操作数
传送类	取数	LD	GR,ADR[,XR]
	存数	ST	GR,ADR[,XR]
	取地址	LEA	GR,ADR[,XR]
算术运算类	加法	ADD	GR,ADR[,XR]
	减法	SUB	GR,ADR[,XR]
逻辑运算类	逻辑乘	AND	GR,ADR[,XR]
	逻辑加（逻辑或）	OR	GR,ADR[,XR]
	按位加（逻辑异或）	EOR	GR,ADR[,XR]
比较类	算术比较	CPA	GR,ADR[,XR]
	逻辑比较	CPL	GR,ADR[,XR]
移位类	算术左移	SLA	GR,ADR[,XR]
	算术右移	SRA	GR,ADR[,XR]
	逻辑左移	SLL	GR,ADR[,XR]
	逻辑右移	SRL	GR,ADR[,XR]
转移类	无条件转	JMP	ADR[,XR]
	大于等于转	JPZ	ADR[,XR]
	小于转	JMI	ADR[,XR]
	不等转	JNZ	ADR[,XR]
	等于转	JZE	ADR[,XR]
堆栈类	进栈	PUSH	ADR[,XR]
	退栈	POP	GR
子程序类	调用	CALL	ADR[,XR]
	返回	RET	

2.1.2 CASL 汇编语言的指令格式

COMET 的机器指令结构虽未明确定义，但不失一般性，即由操作码和地址码两部分组成。操作码说明了本条指令要进行的运算，如加法、移位等，地址码说明了进行某种运算时所需要的操作数或操作数的地址。一条 CASL 汇编语言指令在内存中占两个字的空间（即 CASL 汇编语言指令为定长指令），可将符号指令的目标码设想成如图 2.1 所示。图中：OP 为指令码，由 8 位组成的代码，泛指 23 条指令；GR 为通用寄存器，由 4 位组成的代码，分别表示 GR0～GR4 等 5 个通用寄存器；XR 为变址寄存器，由 4 位组成的代码，分别表示 GR1～GR4 等 4 个通用寄存器；ADR 为形式地址，由 16 位组成，取值范围 0～65535。

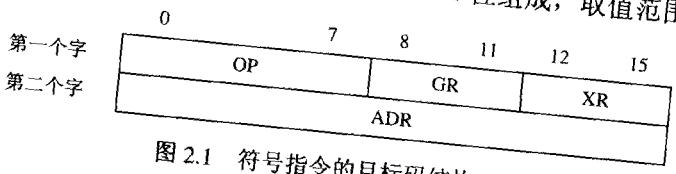


图 2.1 符号指令的目标码结构

CASL 的每条指令书写在一行内（最多不超过 72 个字符），它的书写格式见表 2.2。

表 2.2 CASL 汇编语言指令书写格式

标号（可选）	指令码	操作数（可选）	注释（可选）
[LABEL]	START	[LABEL]	
	END	空 白	
[LABEL]	DC	常 数	
[LABEL]	DS	区域的字数	
[LABEL]	IN	ALABEL,NLABEL	
[LABEL]	OUT	ALABEL,NLABEL	
[LABEL]	EXIT	空 白	
[LABEL]	符号指令参照 2.2.1		

由上表可知，CASL 每条指令由标号（可选项）、指令码、操作数（可选项）和注释（可选项）4 栏构成，每一栏的书写规则如下：

（1）标号栏。从第一个字符位置开始，最多不超过 6 个字符位置。

（2）指令码栏。在无标号时，从第二个字符以后的任意字符位置开始；在有标号时，标号后面至少有一个空白。从其后的任意字符位置开始。

（3）操作数栏。指令码后至少有一个空白，其后到第 72 个字符位置，不能继续到下一行。

（4）注释栏。行里有分号（；）时，其后直到终了作为注释处理。在注释栏里可以书写任何字符。

LABEL 泛指标号，标号最多不超过 6 个字符，开头必须是英文大写字母，以后可为英文大写字母或数字。

在表 2.2 中，用空白表示的栏里不得写入字符。

2.2 CASL 汇编语言的寻址方式

寻址方式就是指令寻找操作数的方式。CASL 汇编语言的寻址方式有 4 种：

（1）无地址方式。

这种语句除标号外，只有一条操作符，如指令 RET（仅此一条）。

（2）寄存器方式：GR。

这种语句只有一操作数，这个操作数在指令给出的寄存器中，如 POP GR0。

（3）直接地址方式（两种）。

1）一地址直接地址方式：ADR。这种语句的地址只有一个，由 ADR 指示，有效地址 E 由 ADR 直接得到：E= ADR。如“JZE FINISH”就是一地址直接地址方式，FINISH 是内存中某一条指令的地址。这种寻址方式常用来实现直接转移或子程序直接调用。

2）两地址直接地址方式：GR、ADR。这种语句有两个操作数，一个操作数在 GR 中，另一个操作数在内存中，它的有效地址为 E=ADR，如“AND GR2，MASK，”MASK 是内存中某一个变量的地址。

（4）变址地址方式（两种）。

1）一地址变址地址方式：ADR、XR。这种语句只有一个操作数在内存中，其有效地址为 E= ADR+（XR），如“PUSH 0，GR1”。在转移或子程序间接调用时常用这种寻址方式。

2）两地址变址地址方式：GR，ADR，XR，这种语句有两个操作数：一个操作数在 GR 中，另一个操作数在内存中，它的有效地址为 E= ADR+（XR），如“LD GR2，INBUF，GR1”。两地址变址地址方式在程序中常用来对数组进行操作，INBUF 可看成是数组名（实际上在 CASL 汇编语言中无数组类型），GR1 可看成是数组 INBUF 的下标，INBUF 是数组在内存中的首地址，一但分配完毕后便不再发生变化，通过 GR1 的变化可以取得从 INBUF 开始存放的 10 个数中的任何一个值。

2.3 CASL 汇编语言指令系统详述

2.3.1 CASL 汇编语言的伪指令

伪指令供汇编系统确定汇编程序的结构，定义汇编程序中使用的常数、变量和数据区。下面对 4 种伪指令逐一介绍。

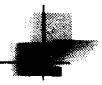
1. START 伪指令

书写格式：[LABEL] START [LABEL]。

功能：表示程序的开头，即在程序的开始必须书写。

说明：操作数栏中的标号是这个程序中定义的标号，它指出该程序的启动地址，在默认的情况下，程序从头开始执行。

标号栏中的标号可以作为其他的程序进入该程序的入口。



2. END 伪指令

书写格式：END

功能：表示程序的终止。

说明：在程序的末尾书写。

3. DC 伪指令

书写格式：[LABEL]　DC　常数。

功能：用来指定和存储常数，常数分十进制常数、十六进制常数、地址常数和字符串常数 4 种。

说明：标号栏中的标号是代表被指定的十进制常数、十六进制常数、地址常数的存储地址或代表被指定的字符串常数的存储区域的第一字的地址。

（1）十进制常数：DC n。

用 n 指定一个十进制数（-32768≤n≤65535），并将 n 转换成二进制数存储在一个字中。如果 n 超出规定的范围，则将其低 16 位存储起来。

对 32768～65535 的十进制数也可以用负的十进制常数表示。

（2）十六进制常数：DC #h。

用 h 指定一个 4 位十六进制数（0000≤h≤FFFF），并将 h 对应的二进制数存储在一个字中（在 h 的前面必需写上#）。

（3）地址常数：DC LABEL。

将标号 LABEL 所对应的地址作为一个字的二进制数存储。若 LABEL 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。

（4）字符串常数：DC '字符串'。

将字符串从左开始的每个字符转换成字符数据（参阅 1.2.5 小节），并依次把字符数据存储在连续的各字中。

4. DS 伪指令

书写格式：[LABEL] DS 区域的字数。

功能：用来保留指定的字数的存储区域。

说明：区域用字数，用十进制常数（≥0）指定。

标号栏中的标号是代表被保留的存储区域的第一个字的地址。

区域的字数为 0 时，存储区域不存在，但是标号中的标号仍有效，即代表下一字的地址。

2.3.2　CASL 汇编语言的宏指令

宏指令是根据事先定义的指令串和操作的信息，生成指令功能的指令串。CASL 中有进行输入、输出及结束程序等宏指令，而没有定义输入、输出符号指令，这类处理由操作系统完成。

程序中出现宏指令时，CASL 生成调用操作系统的指令串，但是，生成的指令串的字数不定。

执行宏指令时，GR 的内容保持不变而 FR 的内容不确定。

1. IN 宏指令

书写格式：[LABEL]　IN ALABEL,NLABEL

功能：宏指令 IN，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被顺序存放在标号为 ALABEL 开始的区域内，已输入的字符个数以二进制数存放在标号为 NLABEL 的字中。记录之间的分隔符号不输入。

2. OUT 宏指令

书写格式：[LABEL] OUT ALABEL NLABEL

功能：宏指令 OUT，将存放在从标号 ALABEL 开始的区域中的字符数据，作为一个记录向输出装置输出，输出的字符个数由标号为 NLABEL 的字中的内容指定。输出时，若要记录间的间隔符号，由操作系统自动插入输出。

3. EXIT 宏指令

书写格式：[LABEL] EXIT

功能：宏指令 EXIT，表示程序执行终止，控制返回操作系统。

2.3.3 CASL 汇编语言的符号指令

符号指令即 CASL 的可执行指令，共 23 条。它的一般格式是：[标号]操作符 [GR[ADR[,XR]]][；注释]。

其中位于[]中的部分为可选项，GR 为通用寄存器（包括 GR0、GR1、GR2、GR3、GR4），XR 为变址寄存器（包括 GR1、GR2、GR3、GR4）。

注意：GR0 不能出现在 XR 部分。

2.3.3.1 传送类指令

这类指令有 3 条：LD 和 ST 用于数据传送，LEA 用于地址传送。下面分别说明。

1. 数据装入指令 LD（可理解为 Load）

格式：[标号]　LD　GR,ADR[,XR]

功能：将有效地址中的操作数取出来送到通用寄存器 GR 中。E＝ADR[+(XR)]，GR=(E)。用于从内存向通用寄存器读数据。

执行此指令后，不影响标志寄存器 FR 的值。

举例：

例 1　LD GR0,20；将有效地址 20 中的操作数取出来送到 GR0 中。

但是，这种用法不多，因为一个变量经过汇编后，和它对应的内存地址是由操作系统安排的，程序员一般不知道。我们一般用变量名来代替直接地址，在例 2 中将说明这一点。

例 2　LD GR0,VAR；将变量 VAR 中的操作数取出来送到通用寄存器 GR0 中。

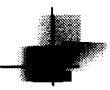
例 3　LD GR0,INBUF,GR1；E= INBUF+（GR1），从 INBUF 开始的一段数据中读一个数据送到 GR0 中。这是两地址变址寻址方式，一般用来通过 XR 的变化读一段数据。

2. 数据存储指令 ST（可理解为 Store）

格式：[标号]　ST　GR,ADR[,XR]

功能：将通用寄存器 GR 中的数据存储到有效地址中。E＝ADR[+(XR)]，(E)=(GR)。用于将通用寄存器中的数据写入内存。

执行此指令后，不影响标志寄存器 FR 的值。



举例：

例 1　ST GR0,20；将 GR0 中的数据存储到内存地址 20 中。

但是这种用法不多，因为一个变量经过汇编后，和它对应的内存地址是由操作系统安排的，程序员一般不知道。我们一般用变量名来代替直接地址。在例 2 中将说明这一点。

例 2　ST GR0,VAR；将通用寄存器中的操作数存储到变量 VAR 中。

例 3　ST GR0,DATA,GR1；E= DATA+（GR1），将 GR0 中的操作数存储到从 DATA 开始的一段内存的某单元中。这是两地址变址寻址方式，一般用来通过 XR 的变化写一段数据。

3. 地址传送指令 LEA（可理解为 Load EA,EA 指有效地址 Effective Address）

LEA 指令在 CASL 汇编语言中使用非常频繁，对本条指令一定要熟练掌握。

格式：[标号]　LEA　GR,ADR[,XR]。

功能：将有效地址传送给 GR,E＝ADR[+(XR)],GR=E。

执行此指令后，影响标志寄存器 FR 的值。

举例：假定有伪指令 INBUF DC '0123456789'，经汇编后在内存中的分配见表 2.3（假定从内存地址 18 开始分配）。

表 2.3　伪指令汇编后内存分配

	INBUF	DC	'0123456789'
18	0		'0'
19	0		'1'
20	0		'2'
21	0		'3'
22	0		'4'
23	0		'5'
24	0		'6'
25	0		'7'
26	0		'8'
27	0		'9'

例 1　LEA GR1,18；将有效地址 18 传送给 GR1，即 GR1 中存放的是变量 INBUF 的地址。但是我们一般不这样使用，而常用指令“LEA GR1，INBUF”，这两者是等价的。

例 2　LEA GR1,1,GR1；GR1 中的地址加 1，在例 1 中 GR1 已指向有效地址 18，本条指令执行后，GR1 中的有效地址为 19，即指向存放字符“1”的单元。

例 3　LEA GR1,-1,GR1；GR1 中的地址减 1，在例 2 中已指向有效地址 19，本条指令执行后，GR1 中的有效地址为 18，即指向存放字符“0”的单元。

例 4　LEA GR1,3,GR1；GR1 中的地址加 3，在例 3 中 GR1 已指向有效地址 18，本条指令执行后，GR1 中的有效地址为 21，即指向存放字符“3”的单元。

例 5　LEA GR1,-2,GR1；GR1 中的地址减 2，在例 4 中已指向有效地址 21，本条指令执行后，GR1 中的有效地址为 19，即指向存放字符“1”的单元。

我们都学过 C 程序设计，下面我们以 C 程序为例来加深对 LEA 的理解：

```
#include <stdio.h>
main()
    {
        char inbuf[]="0123456789";
        char *p=&inbuf;
        printf("%c ",*p);
        ++p;
        printf("%c ",*p);
        --p;
        printf("%c ",*p);
        p+=3;
        printf("%c ",*p);
        p-=2;
        printf("%c \n",*p);
    }
```

则输出为：0 1 0 3 1

我们可以列出一个 GR1 和指针变量 p 的功能对照表见表 2.4。

表 2.4　指针变量与变址寄存器功能对照

指针变量 p	寄存器 GR1
p=&inbuf	LEA GR1,INBUF
++p	LEA GR1,1,GR1
--p	LEA GR1,-1,GR1
p+=3	LEA GR1,3,GR1
p-=2	LEA GR1,-2,GR1

其实在 CASL 汇编语言中，当将内存地址做为常数使用后，LEA 就具有了下面的含义：

（1）给寄存器赋初值。

```
LEA GR1,18      ；给 GR1 赋初值 18。
LEA GR1,0       ；给 GR1 赋初值 0。
LEA GR1,-1      ；给 GR1 赋初值-1。
```

（2）实现寄存器加或者减一个数。

```
LEA GR1,1,GR1   ；GR1 加 1。
LEA GR1,-1,GR1  ；GR1 减 1。
LEA GR1,3,GR1   ；GR1 加 3。
LEA GR1,-2,GR1  ；GR1 减 2。
```

（3）实现寄存器加或者减一个数后，送到另一个寄存器中（注意：XR 的值不发生变化）。

LEA GR0,1,GR1 ；GR1 加 1→GR0,GR1 值不变。

LEA GR2,-1,GR1 ；GR1 减 1→GR2,GR1 值不变。

LEA GR3,3,GR1 ；GR1 加 3→GR3,GR1 值不变。

LEA GR4,-2,GR1 ；GR1 减 2→GR4,GR1 值不变。这条指令可在不改变 GR1 值的情况下判断和常数 2 的关系。因为 GR4 中保存的是 GR1 的值，系统根据 GR4 的值设置 FR，根据 FR 可知道 GR1 和常数 2 的关系。

（4）实现寄存器之间的传送。

LEA GR0,0,GR1 ；(GR1) →GR0。

LEA GR2,0,GR3 ；(GR3) →GR2。

LEA GR3,0,GR2 ；(GR2) →GR3。

LEA GR4,0,GR1 ；(GR1) →GR4。

LEA GR1,0,GR1 ；将一个寄存器的值传送给它自己，这有什么用途呢？这个指令是根据 GR1 的值来设置 FR，供后面的指令使用。

2.3.3.2 算术运算类

算术运算类有两条指令 ADD（加法）和 SUB（减法）。

1. 加法指令 ADD（可理解为 Addtion）

格式：[标号] ADD GR,ADR[,XR]。

功能：将通用寄存器 GR 的内容和有效地址中的内容相加，结果送 GR,E＝ADR[+(XR)]，GR=(GR)+(E)。

执行此指令后，影响标志寄存器 FR 的值。

举例：

例 1 ADD GR0,20 ；E=20，GR0+（20）→GR0。这种方式我们很少使用。

注意：20 是内存地址，如果理解成 GR0+20→GR0 就错了。

例 2 ADD GR0,VAR；E=VAR，GR0+（VAR）→GR0。

例 3 ADD GR0,INBUF,GR1；E= INBUF+（GR1）GR0+（E）→GR0。这种方式常用来加数组中的某一个数据。

例 4 假定有如下指令：

```
ASCII   DC   'A'
LD   GR1, ASCII  ；将字符'A'的 ASCII 送给 GR1
ADD GR1,32,GR1 ；将字符'A'的 ASCII 转变成其对应的小写字母'a'送给 GR1。从附录 B 的 ASCII 表可以看到，一个大写字母字符和它对应的小写字母字符之间的 ASCII 编码值差 32。请读者记住：这是将大写字母字符'A'～'Z'转变为对应的小写字母字符'a'～'z'的第一种方法。
```

例 5 假定有如下指令：

```
DATA   DC   1
```