CASL 程序设计教程

刘克武　李　冰　李冬梅　编著

清 华 大 学 出 版 社

(京)新登字 158 号

内 容 简 介

CASL 是建立在一种假想机上的汇编语言系统,汇集了当今主流 PC 机的指令结构和功能。本书以程序设计为纲,全面系统地介绍了 CASL 汇编语言。全书共分 9 章,分别讲述了 CASL 程序设计环境、伪指令和宏指令、数的存取和传送、算术运算和算术操作、逻辑运算和逻辑操作、比较与转移、数据栈与子程序、程序设计基础和例题分析等内容。本书每章后面都附有习题,供读者练习参考。

本书可作为高等院校汇编语言程序设计课程的教材,也可供参加‘中国计算机软件专业技术资格和水平考试’的考生备考使用。

版权所有,翻印必究。

本书封面贴有清华大学出版社激光防伪标签,无标签者不得销售。

书　　名: CASL 程序设计教程
作　　者: 刘克武　李冰　李冬梅　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
责任编辑: 刘　彤
封面设计:
版式设计: 刘　路
印 刷 者: 北京昌平环球印刷厂
发 行 者: 新华书店总店北京发行所
开　　本: 787×1092 1/ 16　印张: 13　字数: 298 千字
版　　次: 2002 年 7 月第 1 版　2002 年 7 月第 1 次印刷
书　　号: ISBN 7-302-05507-6/ TP・3237
印　　数: 0001～5000
定　　价: 23.00 元

前　言

CASL程序设计教程

汇编语言是一种对计算机硬件有很强依赖性的低级语言,学习汇编语言可以深入到计算机内部了解计算机底层的部件功能及运作。掌握了汇编语言可以设计系统软件,实施工业过程控制自动化,汇编语言在计算机众多应用领域中是一种必不可少的,也是不可替代的计算机语言。

汇编语言因机器不同而异,随着计算机更新换代,汇编语言也在不断地推陈出新,从而使得汇编语言在使用时效上较为短暂。如何延长汇编语言的使用时效,使其像当今的高级语言一样也具有普遍性和通用性呢?CASL 汇编语言的出现和推广在很大程度上解决了汇编语言时效短的问题。

CASL 是建立在一种假想机 COMET 的汇编语言系统,这种超脱使得它不存在因机器不同而异的问题;CASL 又汇集了当今主流 PC 的指令结构和功能,因此它又不失其实用性。学习 CASL 可以打下良好的汇编语言基础,掌握了 CASL 可以轻松、快捷地学会当今主流 PC 的汇编语言。

本书以程序设计为纲,全面、系统地介绍了 CASL 汇编语言。内容由浅入深,循序渐进,一章一个重点,一章一个学习周期。全书共分 9 章,每章都有习题。1 至 7 章及附录 1 由刘克武编写;8、9 两章由李冰编写;附录 2、3 由李冬梅编写,刘克武负责全书修正、定稿。

本书不仅可以作为高等院校“汇编语言程序设计”课程的教材,还可以供“计算机软件专业技术资格和水平考试”的考生备考使用。

编　者

2002 年 5 月

目　录

CASL程序设计教程

第 1 章

CASL程序设计环境

CASL 是 Computer Assembly System Language 的简称,可译为计算机汇编语言系统。

由于汇编语言因计算机不同而异,所以当计算机的 CPU 更新换代时就会使得汇编语言随之而改变,因此造成汇编语言在使用时效上比高级语言短得多。CASL 汇编语言是建立在一个名为 COMET 假想计算机上的汇编语言系统,因此它在很大程度上克服了汇编语言使用时效短的问题。加之 CASL 集中了当今主流 PC 所使用的汇编系统的主要功能及指令形式,所以学习 CASL 与使用当今 PC 是紧密相关的。掌握了 CASL 程序设计,可以打下良好的汇编语言程序设计基础,具备了 CASL 基础,可以迅速学会当今建立在任何计算机上的汇编语言系统。因此可以说学习 CASL 即有广泛意义,又有现实意义。

汇编语言对计算机硬件有很大的依赖性,CASL 也不例外。因此学习 CASL 就需要了解 COMET 计算机,这就是 CASL 的硬件背景,而有关 CASL 的表述形式,指令格式,数的使用与表达等,这些内容则称为 CASL 的软件环境。CASL 的硬件背景与软件环境就是本章要介绍的 CASL 程序设计环境。

1.1 CASL 的硬件背景

1.1.1 COMET 计算机的结构

COMET 是一个 16 位机,每一个单元长度为两个字节。在 COMET 上可以进行 16 位的定点整数运算,逻辑运算,码制变换及字符处理等。

COMET 的组成有 CPU、内存、外部设备等 3 大部分,它与当今的个人计算机(Personal Computer)很相似,其结构示意图如图 1.1 所示。

1.1.2 COMET 的 CPU

CPU(Central Processing Unit)是中央处理器的简称。CPU 与 CASL 有关的部件有通用寄存器、指令计数器、标志寄存器、栈指针等。指令操作部件 OP 在 CASL 的描述中并不出现。

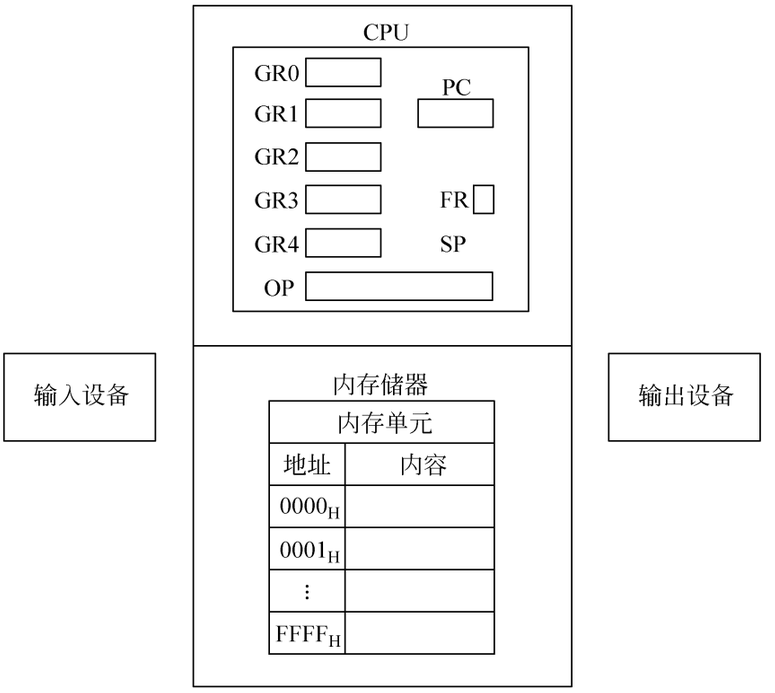


图 1.1 COMET 假想机结构

(1) 通用寄存器 GR(General Register)

COMET 的 CPU 设置了 5 个通用寄存器,它们分别是 GR0、RG1、GR2、GR3、GR4。这 5 个寄存器均在 CASL 指令中出现。

通用寄存器在 CASL 中既是一个存储空间,又是一个处理部件,使用它可以存储参加运算的数据,保留运算或处理的中间结果或最后结果。在运算或处理中,它可以作为累加器、移位运算器及个数计数器等。CASL 所有指令的描述几乎都与通用寄存器有关。

通用寄存器的空间长度为两个字节,16 位,它与内存单元的长度一样。因此,寄存器与单元相互交换数据十分方便。寄存器表示无符号数的范围为 0000_H ～$FFFF_H$,这个值正好是内存单元的地址范围。

通用寄存器可以作为地址寄存器及变址寄存器,但 GR0 不允许用作变址寄存器。

(2) 指令计数器 PC(Program Counter)

指令计数器也叫程序计数器,程序计数器记录着当前被执行的指令的地址。由于 CASL 指令长度为 4 个字节,32 位,占两个内存单元,所以执行完一条指令,指令计数器的内容自动加 2,表示为(PC) + 2→PC,而遇到转子、转移指令时除外。

(3) 标志寄存器 FR(Flag Register)

标志寄存器也叫标志寄存器,它的作用是记录指令执行后的状态。在 CASL 指令系统中有 12 条指令执行后都会对标志寄存器产生影响,或者说标志寄存器对这 12 条指令执行后的状态都予以记录。

标志寄存器的长度为两位,两位所能表示的状态最多为 4 种,这 4 种状态为 00、01、10、11。11 这种状态不使用,所以只有 3 种状态用于指令执行后的状态标志,具体规定如表 1.1。此外,两位标志寄存器,左侧的一位表示符号,0 为正,1 为负。

(4) 栈指针 SP(Stack Pointer)

数据栈也称堆栈,是一种特定的内存空间。在这个空间中可以存储数据,也可以取出

存入的数据。存入数据叫数据进栈,取出数据叫数据出栈。数据栈的特定性能是“先进后出”或“后进先出”。这个性能通俗地说就是,最先进入数据栈的数据,最后才能出栈。数据栈的这种有序出、入数据,不但体现出栈的空间含义,而且包含了一个先、后的时序概念。

表 1.1 3种用于指令执行后的状态标志规定

执行完运算或操作	执行完比较指令	FR中的状态
① 运算结果为正数	比较结果为大于	00
② 运算结果为零	比较结果为等于	01
③ 运算结果为负数	比较结果为小于	10

栈指针也叫栈顶指针,它在数据栈的操作过程中所描写的是数据栈的存数状况。数据进栈,栈顶上升,栈指针上浮;数据出栈,栈顶下降,栈指针下沉。但不管数据栈的数据怎样变化,栈指针永远指示着栈顶所在的单元地址。

在CASL中,栈指针规定用寄存器GR4。用圆括号表示GR4的内容,则(GR4)表示栈顶地址。而数据进栈,则有(GR4) - 1→GR4;数据出栈,栈指针变化为(GR4) + 1→GR4。

1.1.3 COMET的内存储器

(1) COMET的内存容量

COMET的内存容量为65536个字,即64K字(1K = 1024)。一个字的长度,即字长为两个字节,16位。16位的编号如下:

在16位的一个字长中,第0位为最高位,在表示有符号数时为数的符号位,第15位为最低位。

用一个字16位存储无符号数时,表示范围为0～12^{16} - 1,用十六进制表示时为0～FFFF。

用一个字16位存放有符号数的补码时,表示范围为 - 2^{15}～2^{15} - 1,即 - 32768～32767。

用一个字16位存储一个字符的常数时,高8位为0,低8位中的7位为美国信息交换标准代码ASCII。

(2) 内存单元与地址

一个单位的存储空间称为一个单元,65536个字,即65536个单元。由于单元编号,即地址从0开始,因此地址的表示范围为0～65535,用十六进制表示这个地址范围为0000_H～$FFFF_H$。在CASL中,内存单元地址使用标号地址,当表示连续的若干单元的地址时,只需要一个首地址。例如图1.2所示。

图 1.2　内存单元与地址

图 1 2 中 WK,DZ 是在 CASL 中使用的内存单元地址,由标号名表示,它很像高级语言中的变量名。而 DY,DATA 是在 CASL 中使用的连续单元的首地址,它们很像高级语言中的数组名。DY,DATA 为首地址的连续单元中的每一个单元地址很像数组元素名。怎样命名标号,如何定义内存单元将在第 2 章介绍。

1.2　CASL 的软件环境

1.2.1　CASL 的字符集

描写 CASL 的字符全体称为 CASL 的字符集,或者说,在 CASL 中允许使用的字符全体称为 CASL 的字符集。CASL 字符集如表 1.2 所示。

表 1.2　CASL 汇编字符集

高位 / 低位	2	3	4	5
0	间隔	0	@	P
1	!	1	A	Q
2	”	2	B	R
3	#	3	C	S
4	$	4	D	T
5	%	5	E	U
6	&	6	F	V
7	’	7	G	W
8	(	8	H	X
9	)	9	I	Y
A	*	:	J	Z
B	+	;	K	[
C	,	<	L	\
D	-	=	M	]
E	.	>	N	^
F	/	?	O	-

由CASL字符集可以看出字符的组成如下:

① 数字字符0～9共10个。

② 大写英文字母A～Z共26个。

③ 算术运算符、关系运算符、连接符及其他符号共24个。

全体字符合计共60个。

在CASL字符集中,列出了字符所对应的内码,所谓内码就是字符在计算机的代码。例如:

字母A在计算机中的内码为$41_H = 1000001_2 = 65_{10}$

字母B在计算机中的内码为$42_H = 1000010_2 = 66_{10}$

数字0在计算机中的内码为$30_H = 0110000_2 = 48_{10}$

数字1在计算机中的内码为$31_H = 0110001_2 = 49_{10}$

有了字符的内码,就可以在计算机中查询任何一个字符。借助内码我们还可以对字符信息进行筛选、变换等处理。

由字符集提供的内码可以看出,内码是一种7位码,它采用的是美国信息交换标准代码,即ASCII(American Standard Code for Information Interchange)。

1.2.2 CASL指令及结构

(1) CASL指令种类

CASL中有3类指令,即基本指令、伪指令和宏指令。

① 基本指令,是CASL汇编语言基本功能的体现,共23条。其中21条的写法有两种形式,一种形式为直接操作形式,另一种形式为间接操作形式。

② 伪指令,是对汇编编译程序进行说明的指令,这种指令在编译中并不产生目标代码。伪指令共有4条,所谓伪指令,从字面上说是"假指令",其含义为非基本指令。

③ 宏指令,是一条指令可以启动一个程序段的指令。宏指令共有3条,宏可以理解为"大",由于输入、输出等操作涉及到的部件较多,用一条基本指令概括不了,所以使用一个程序段来描述,为了使用上的方便,设定一条指令来启动这个程序段,担任启动的指令即为宏指令。

(2) CASL指令结构

CASL的基本指令,每条指令由下列4部分组成:

CASL指令结构			
①	②	③	④
[标号名] [	操作码 [	操作数 [	;注释
QS	LD	GR1,DY	;(DY)→GR1

①、②、③是指令的主体,它们之间在书写上要留出一个以上的空格:[。注释是使用者所附加的指令功能说明,以备阅读。表中所列出的"QS LD GR1,DY"是一条取数指令,它的功能是取内存单元DY中的内容,放在寄存器GR1中。

在汇编语言表述中,为了简捷、方便,通常使用方括号、圆括号来表达、描述某个事项,并给它们赋予特定的含义,例如:

[ ],方括号表示括号中的内容可以省略,也可以不省略,究竟是取是舍由使用者根据需要而定。

( ),圆括号表示其中的内容,如(DY)表示单元 DY 中的内容,(GR1)表示寄存器 GR1 中的内容,如果用两对括号,则表示"内容的内容"。例如,((GR1))表示 GR1 中内容的内容,同理((DY))表示 DY 单元中内容的内容。

① 标号名,从形式上看,它很像高级语言中语句的行号,但它并不表示指令的顺序,也不要求每条指令都加上标号名。只当该指令被其他指令调用时,才需要加上标号名。标号名实际上是该指令的标号地址,它描写的是该指令所在的位置。未加标号名的指令在程序中只体现出先、后的顺序位置。标号名的命名规定是,以字母开头,其后为字母或数字,长度为 6 个字母、数字之内。标号名中不允许用字母、数字以外的符号。

② 操作码,是 CASL 中规定的指令功能码,大都使用英文缩写,其目的是为了便于记忆。该功能码经汇编编译系统翻译后,在计算机中对应一条机器指令。例如 LD 是 LOAD 的缩写。

③ 操作数,从字面上讲,操作数往往被人们理解为"参加运算或操作的数据"。实际上,在汇编语言指令中,操作数这一项并不直接给出操作数本身,而是描述出操作数的来龙去脉,如与操作数有关的内存单元地址及存放操作数的寄存器等。例如在"LD GR,DY"指令中,操作数这一项所给出的是寄存器 GR1,作为操作数的临时寄存空间;DY 是内存单元地址,作为操作数的源发地。

④ 注释,它不属于指令的固有成分,注释是使用者附加的指令说明。按照 CASL 的规定,为指令加注释时,注释内容之前必须用分号";"用以与指令隔开。

下面再看几条指令书写的例子:

标号	操作码	操作数	注 释	功 能
[省]	LEA	GR1,6	;6→GR1	传送。将 6 传到寄存器 GR1。
[省]	ST	GR1,DY	;(GR1)→DY	存数。将 GR1 中的内容存于 DY 单元中。
[省]	ADD	GR1,DY	;(GR1) + (DY)→GR1	加法。GR1 中内容与单元 DY 中内容相加后送 GR1。

1.2.3 CASL 中的数

按照 CASL 的规定,在程序的指令中可以使用的数有 5 种,即有符号的十进制数、无符号的十进制数、十六进制数、字符常数和标号地址常数。

(1) 有符号的十进制数

在指令中直接写出十进制数,正数不加符号,负数加上符号,均为整数。

这种数在计算机中用二进制补码表示,一个数在 16 位的单元中,高位(第 0 位)为数

的符号,其他位(1～15 位)为数的尾数,小数点隐含在尾数的最低位之后。形式为:

从数的形式可以得出,指令中的十进制数为整数,在计算机中为补码,故数的范围为 -2^{15}～$2^{15}-1$,即 -32768～32767。例如:

指令中的数为 +1,则在计算机中为 0000000000000001;

指令中的数为 -1,则在计算机中为 1111111111111111;如用十六进制表示计算机中的 +1,则为 0001_H;表示 -1,则为 $FFFF_H$。

同理,若在计算机中得到的数为 $FFFE_H$,则该数为十进制的 -2。“有符号的十进制数在计算机中为补码”在使用中要特别注意,这是汇编语言的一个难点。

(2) 无符号的十进制数

无符号数也称逻辑数,由于这种数在计算机 16 位的一个单元中全为数值,没有符号,所以数的最小值为 16 位全 0,即 0;最大值为 16 位全 1。因此数的表示范围为 0～65535,用十六进制表示数的范围为 0000_H～$FFFF_H$。在指令中使用这种无符号数时,注意不要越界。

(3) 十六进制数

在 CASL 汇编程序中可以使用十六进制数,使用形式为数的前面加上“#”号,后跟 4 位十六进制数。例如 #0001, #01AB, #FFFF, #1234 等都是符合要求的十六进制数。但是,这种十六进制数不能在基本指令中出现,只能定义在伪指令中。

(4) 字符常数

在 CASL 中的字符常数,其形式是由单引号将字符括起来而构成的。例如'A','ABC','1234','A1','B2'等都是符合规定的字符常数。字符常数只能在伪指令中定义,不能出现在基本指令中。被定义的字符常数在计算机中,一个字符占一个单元,每个单元的高 8 位为 0,低 8 位从最低位开始存储字符所对应的 7 位 ASCII,形式如下:

由存储形式可以看出,常数'A'在计算机的一个单元中只占用了低 7 位存放字符所对应的 ASCII,其余位均为 0。

(5) 标号地址常数

标号地址常数,即标号名。它是以字母打头的字母、数字串。标号地址常数的用途为,一是作内存单元地址的代号,二是作指令的逻辑地址。标号常数只当汇编语言源程序转为目标程序时,它才对应实际的内存地址。

1.2.4 一个完整的 CASL 程序

一个完整的 CASL 程序大都包括基本指令、伪指令、宏指令。下面是实现 1 + 2 = ? 的 CASL 汇编源程序。

	计算 1 + 2 = ? 的 CASL 程序			
	标号	操作码	操作数	注　　释
①	JJ12	START		;开始
②		LD	GR3, S1	;(S1)→GR3。
③		ADD	GR3, S2	;(GR3) + (S2)→GR3
④		ST	GR3, JG	;(GR3)→JG,
⑤		EXIT		;程序执行结束。
⑥	S1	DC	1	;定义十进制常数 1
⑦	S2	DC	2	;定义十进制常数 2
⑧	JG	DS	1	;定义存结果的内存单元
⑨		END		;结尾

程序中,②、③、④为基本指令,②LD 为取数指令,执行该指令可以取出 S1 所定义的常数 1 放于寄存器 GR3 中。③ADD 为加法指令,执行该指令可将 GR3 中的当前内容 1 与 S2 中的内容 2 相加,结果又存在 GR3 中。④ST 为存数指令,执行该指令可将 GR3 中的相加结果存于内存单元 JG 中。⑤EXIT 为宏指令,执行该指令,程序执行终止。指令①、⑥、⑦、⑧、⑨为伪指令,⑥、⑦为定义常数指令,⑧为定义内存单元指令。由于⑥、⑦、⑧分别为②、③、④指令引用,所以在指令中加上了标号名。指令①为程序开头指令,其标号为程序名。指令⑨为程序结尾指令。每个程序必须有①、⑨指令,用以说明本程序的上、下界。

习　题　1

1. 解答 CASL 与硬件背景有关的问题。
   (1) 在 CASL 中使用的寄存器有哪几个?
   (2) 通用寄存器在 CASL 程序中的作用是什么?
   (3) 标志寄存器有什么功能?
   (4) 寄存器 GR0 与其他寄存器在使用上有什么不同?
   (5) 程序计数器如何跟踪程序的运行?
   (6) 栈指针用什么来描述?怎样描述数据进、出栈?
   (7) COMET 是一种什么样的计算机?
   (8) COMET 计算机内存有多大?内存的地址范围是多少?

(9) COMET的内存字长是多少位？CASL的一条指令和一个数据各占几个单元？

(10) 在CASL中怎样使用COMET的内存单元？

2. 解答CASL与软件环境有关的问题。

(1) CASL的字符集包括哪些字符？字符的内码是什么？

(2) 根据CASL字符集，在计算机中已知字符A，怎样查询字符Z？

(3) 根据CASL字符集，在计算机中数字字符6的内码是什么？参照数字字符6的内码怎样变成数值6？

(4) CASL的指令有哪3类？

(5) 什么叫伪指令，它在CASL中的作用是什么？

(6) 什么叫宏指令，为什么在CASL中设置宏指令？

(7) 在CASL指令中，操作数的含义是什么？

(8) 方括号[ ] 圆括号( )在CASL的描述中的作用是什么？

(9) CASL的标号名有什么作用？标号名的实质是什么？

(10) 标志寄存器的三种状态与CASL指令有何关系？

(11) 在CASL中可以使用的数有哪几种？其表现形式如何？

(12) 在计算机中-1与65535是一种什么关系？

第 2 章

伪指令、宏指令在程序中的作用

2.1 CASL 中的伪指令

在 CASL 中有 4 条伪指令。所谓伪指令,从字面上说是“假”指令的意思。但从实际作用上说,伪指令是不产生目标代码的指令。大家知道,用汇编语言编写的程序称为源程序,源程序在计算机中必须经过汇编编译系统翻译后变为目标程序,才能被执行,如图 2.1 所示。

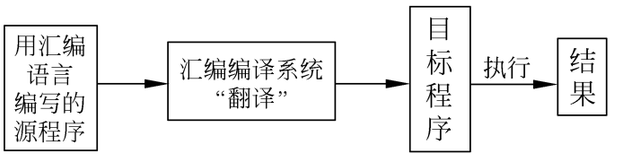


图 2.1 源程序的运行

用 CASL 写出的源程序,在汇编编译过程中需要对源程序加上必要的说明,承担说明任务的指令由 CASL 中的伪指令来完成。此外,由源程序变为目标程序,并不是源程序的每条指令都产生对应的目标程序代码。其中伪指令就不需要,也没有必要产生目标代码,这也是定义为“伪指令”的原因。

CASL 中 4 条伪指令的形式如表 2.1 所示。

表 2.1 CASL 的伪指令

指令名称	指令形式		
(1) 源程序开头伪指令	[标号名]	START	[标号名]
(2) 源程序结尾伪指令		END	
(3) 定义常数伪指令	[标号名]	DC	常数
(4) 定义单元伪指令	[标号名]	DS	单元个数

注:[ ]表示可省项。

2.1.1　源程序开头伪指令

(1) 指令形式

标号	操作码	操作数
[标号名]	START	[执行开始标号]

(2) 指令功能

本指令设置在每一个源程序的开头,为汇编编译程序说明本源程序的开始位置。任何一个 CASL 源程序的第一条必须用此指令。

(3) 指令用法

第一种用法,指令中有两个标号,形式如下:

```
①    BH      START     KS
②    C1      DC   1
③    C2      DC   2
④    C3      DC   3
⑤    KS      LD   GR1, C1
             LD   GR2, C2
             LD   GR3, C3
             …
```

指令①中有两个标号,标号 BH 是为其他程序调用本程序而加的标号;标号 KS 是为了指出本程序开始执行应该在第⑤条指令。这是因为指令①～④都是非执行性指令,执行时可以跳过。如本程序不被其他程序调用,也可省去 BH。

第二种用法,指令中只有前面一个标号,形式如下:

```
①    BH      START
②            LD   GR1, C1
③            LD   GR2, C2
④            LD   GR3, C3
               …
C1      DC   1
C2      DC   2
C3      DC   3
               …
```

本程序中的非执行性指令都在程序的后边,程序开始紧跟着执行性指令②、③、④…,故在指令①中不需要加第二个标号。

第三种用法,指令中前、后均不加标号,形式如下:

```
START
    LD   GR1, C1
```

```
        LD   GR2, C2
        LD   GR3, C3
          …
C1      DC   1
C2      DC   2
C3      DC   3
```

2.1.2 源程序结尾伪指令

(1) 指令形式

标号	操作码	操作数
空	END	空

(2) 指令功能

本指令设置在每一个源程序的末尾,用以对汇编编译程序进行说明。任何一个CASL 程序,其最后一条指令必为本指令。

(3) 指令用法

```
BH      START
        LD   GR1, C1
          …
        END
```

2.1.3 定义常数伪指令

(1) 指令形式

标　号	操作码	操作数
[标号名]	DC	常数

(2) 指令功能

借助本指令可以定义程序中使用的各种常数。这些常数包括,有符号的十进制整常数、无符号的十进制整数、十六进制数、字符常数、地址常数等。

操作码 DC 是 Define Constant 的缩写。

(3) 指令用法

定义十进制数

```
①   CS1     DC      65
```

指令①所定义的 65,它在计算机中是一个 16 位长的二进制数,用十六进制表示为 41_H,而常数的标号地址为 CS1。CS1 相当于高级语言中的变量名,而该指令的功能很像高级语言中的赋值语句,即赋予 CS1 的值为 65。在运算中需要使用 65 时,则调用 CS1。

```
②   CS2     DC      - 1
```

指令②所定义的 - 1,它在计算机中是二进制的 16 位补码。它的左端最高位是数的符号,用十六进制表示为 $FFFF_H$。

定义十进制无符号数

```
③　CS3　　DC　　65535
```

指令③所定义的无符号数 65535,它在计算机中是 16 位的全 1,与有符号数 - 1 是完全一样的,这一点要引起注意。

定义十六进制数

```
④　CS4　　DC　　# 6543
```

指令④定义的数是左端加 # 号的一个四位数,这是 CASL 规定的十六进制表示形式。

定义字符常数

```
⑤　CS5　　DC　　'A'
```

指令⑤定义了一个字符常数,该字符常数的长度为 1 个字符。字符在计算中用字符所对应的 7 位 ASCII 来表示,形式如下:

```
⑥　CS6　　DC　　'ABC'
```

指令⑥定义了含有 3 个字符的字符串常数,在计算机中将自动分配三个连续的单元依次存放'A'、'B'、'C'的 7 位 ASCII,形式如下:

字符常数	十六进制 ASCII	标号地址	计算机中的 ASCII
'A'	41_H	CS6 + 0	0000000001000001
'B'	42_H	CS6 + 1	0000000001000010
'C'	43_H	CS6 + 2	0000000001000011

定义地址常数

```
⑦　CS7　　DC　　CS
```

指令⑦定义了一个标号为 CS 的地址常数。假定 CS 定义的数为十进制的 32767,CS 的实际地址为 1000,而 CS7 的实际地址为 FFFF,则关系如下:

所定义的伪指令	单元地址	单元内容
CS　　DC　　32767	1000	$7FFF_H$
…	…	…
CS7　　DC　　CS	FFFF	1000

2.1.4 定义单元伪指令

(1) 指令形式

标　号	操作码	操作数
[标号名]	DS	单元个数

(2) 指令功能

借助本指令可以定义程序中使用的工作单元,用一条指令可以定义一个单元,也可以定义多个单元,由十进制整数给出定义的单元个数。当用一条指令定义多个单元时,计算机中将多个单元连续分配。因此,定义多个单元也只需给出第一个单元的标号名即可。当单元个数设定为 0 时,计算机将不予分配单元。DS 是 Define Storage 的缩写。

(3) 指令用法

一条指令定义一个单元

```
①   DY      DS   1
```

该单元的标号地址为 DY,它与高级语言中的变量名很相似。

一条指令定义多个单元

```
②   WK      DS   3
```

该指令定义了 3 个单元,WK 是第一个单元的标号地址,它很像高级语言中的数组元素名。3 个单元的关系如下:

地址	单　元
WK	第 1 个单元
WK + 1	第 2 个单元
WK + 2	第 3 个单元

多条指令定义多个单元

```
③   WK1     DS   2
④   WK2     DS   0
⑤   WK3     DS   3
```

指令③、④、⑤分别定义了 2、0、3 个单元,由于指令④定义的单元个数为 0 个,所以将不分配单元。3 条指令的单元分配如下:

地址	单元
WK1	
WK3	

指令③分配 2 个单元。

指令④不分配,继而为指令⑤分配 3 个单元。

2.2　CASL中的宏指令

在CASL中有3条宏指令。所谓宏指令,从字面说就是大指令。宏指令的实际含义是为了实现某种操作目的,把若干条指令集成为一条指令,这样形成的指令称为宏指令。每条宏指令在计算机中被汇编编译时又会展开成若干指令而执行。

CASL中的3条宏指令形式如表2.2所示。

表2.2　CASL中的3条宏指令及其形式

指令名称	指令形式		
(1) 输入宏指令	[标号名]	IN	标号1,标号2
(2) 输出宏指令	[标号名]	OUT	标号1,标号2
(3) 终止程序执行宏指令	[标号名]	EXIT	

2.2.1　输入宏指令

(1) 指令形式

标号	操作码	操作数
[标号名]	IN	标号1,标号2

(2) 指令功能

使用本指令可将由输入设备输入的字符数据存放在指定的输入缓冲区(缓冲区大小为80个字符)中。输入的字符数据为不超过80个字符的一个记录。输入缓冲区由标号1指定,输入字符个数由标号2设置,且标号1、标号2均需伪指令来定义。标号名表示本指令在程序中的位置,可根据需要取舍。IN是INPUT的缩写。

(3) 指令用法

定义输入80个字符的输入宏指令

```
①   SR      IN      HT, GS
            …
②   HT      DS      80
③   GS      DS      1
            …
```

指令①中的SR为输入宏指令的标号地址,该指令所定义的输入指令的环境由标号HT和GS两条伪指令给予说明。HT为输入缓冲区的首地址,GS是记录输入个数的单元地址,内存分配如下:

缓冲区地址	字符单元
HT + 0	80 个
HT + 1	字
HT + 2	符
HT + 3	的
HT + 4	缓
…	冲
HT + 79	区
GS	字符个数单元

定义输入'ABC'3 个字符的输入宏指令

```
④          IN      DATA1,DATA2
                …
⑤   DATA1  DS      3
⑥   DATA2  DS      1
                …
```

指令④所定义的输入宏指令省略了指令标号,DATA1 是输入缓冲区首地址,DATA2 是存放输入字符个数的单元标号地址。内存分配如下:

缓冲区地址	字符单元	
DATA1 + 0	0041_H	'A'
DATA1 + 1	0042_H	'B'
DATA1 + 2	0043_H	'C'
…	以前	
DATA1 + 79	内容	
DATA2	0003_H	

由于只有 3 个字符存入了输入缓冲区,所以字符'A'、'B'、'C'只占用了缓冲区的前 3 个单元,并存入了相应的 ASCII。其余的 77 个缓冲区单元仍然保留着以前的内容。而存放输入字符个数的单元 DATA2,则始终排在缓冲区 80 个单元之后。

2.2.2 输出宏指令

(1) 指令形式

标号	操作码	操作数
[标号名]	OUT	标号 1,标号 2

(2) 指令功能

使用本指令可以把输出缓冲区中的一个单元存放一个字符代码的字符代码还原成字符,并输出在显示器或打印机上,一次最多输出 80 个字符。标号 1 为输出缓冲区的首地

址,标号 2 为输出字符个数的单元标号地址。标号 1 和标号 2 都需要由伪指令定义。OUT 是 OUTPUT 的缩写。指令的标号名表示本指令在程序中的位置,根据需要决定取舍。

(3) 指令用法

定义输出'ABC'3 个字符的输出宏指令:

```
①   SC      OUT     D1, D2
            …
②   D1      DC      'ABC'
③   D2      DC      3
            …
```

指令①中的 SC 为输出宏指令的标号地址,该指令所定义的输出环境是由 D1、D2 为标号的两条伪指令给出。D1 为输出缓冲区的首地址,D2 为记录输出个数的单元地址。定义、执行本指令时,其结果如下:

输入地址	输出单元	显示结果
D1 + 0	0041_H	ABC
D1 + 1	0042_H	
D1 + 2	0043_H	
D2	0003_H	

由于指令①是输出已知的字符常数,所以 D1、D2 均用 DC 指令。

定义输出缓冲区全部字符的输出宏指令:

```
④   OUT     HCQ, ZFS
            …
⑤   HCQ     DS    80
⑥   ZFS     DS    1
            …
```

2.2.3　终止程序执行宏指令

(1) 指令形式

标号	操作码	操作数
[标号名]	EXIT	空白

(2) 指令功能

本指令设置在程序的逻辑结尾。执行本指令,终止程序的执行,并将执行权交给操作系统。指令中的标号名根据需要取舍,EXIT 为指令功能定义符。

(3) 指令用法

将输入的字符原原本本输出后程序停止:

```
KS      TART
        IN      D1,D2
        OUT     D1,D2
①       EXIT
D1      DS      80
D2      DS      1
        END
```

由于本程序的逻辑结尾是在输出指令完成之后,所以指令①设置在 OUT 指令之后。

习 题 2

1. 阅读并完善下列程序。

(1) 输入 80 个字符的输入程序

```
PRO1    START   ①
HCQ     DS      ②
ZFS     DS      ③
KSZ     IN      HCQ, ZFS
        EXIT
        ④
```

(2) 输入已知字符的输入程序

```
PRO2    ①
        IN      BH1, BH2
        ②
BH1     DC      'BEIJING'
BH2     DC      ③
        END
```

2. 按要求写出指令填入指令表中。

(1) 标号名为 C1 的十进制数 100

(2) 标号名为 C2 的十进制数 - 1

(3) 标号名为 C3 的十六进制数 10

(4) 标号名为 C4 的十六进制数 AB

(5) 标号名为 C5 的字符常数 CHINA

(6) 标号名为 C6 的字符常数 WANG

(7) 标号名为 C7 的地址常数 DY

(8) 标号名为 C8 的地址常数 BH

(9) 标号名为 C9,将 65535 用十六进制数表示

指　令　表

序号	标号	操作码	操作数
(1)			
(2)			
(3)			
(4)			
(5)			
(6)			
(7)			
(8)			
(9)			

3. 根据定义的指令,填写表中所列内容。

(1) CSA　DC　65535
(2) CSB　DC　-1
(3) CSC　DC　-32768
(4) CSD　DC　#ABCD
(5) CSE　DC　100
(6) CSF　DC　#0064
(7) CSG　DC　10
(8) CSH　DC　'ABCD'
(9) CSI　DC　'0123'

序号	单元标号地址	单元中用十六进制数表示的内容
(1)		
(2)		
(3)		
(4)		
(5)		
(6)		
(7)		
(8)		
(9)		

4. 根据定义的指令及内存地址分配,填写单元中的内容。

(1) BH1 DC -32767
(2) CS DC 32767
(3) DATA DC 'XYZ'
(4) A DC DATA
(5) B DS 3
(6) C DC 1
(7) WK DC OK
(8) BOT DC A
(9) OK DC BH1

序号	标 号	地 址	内 容
(1)	BH1	6000	
(2)	CS	6001	
(3)	DATA	6002	
		6003	
		6004	
(4)	A	6005	
(5)	B	6006	
		6007	
		6008	
(6)	C	6009	
(7)	WK	600A	
(8)	BOT	600B	
(9)	OK	600C	

第 3 章

数的存、取与传送

在 CASL 中,取数是指将内存储器某个单元中的数取出来放在 CPU 的寄存器中,数据的移动方向是由内存到 CPU。为了取出内存某个单元的数据,在取数指令中必须设定被取数据所在的单元地址。设定数据所在的单元地址,即所谓的寻址。在 CASL 中有两种寻址方式,一种是直接寻址,另一种是间接寻址,而在复杂的汇编语言中有七、八种寻址方式。简单地说,寻址就是寻找操作数所在的单元地址。寻址方式不同,指令的形式也不相同,而最终表现为决定操作数所在地址的方法不同。取数指令除需要设定取数的地址外,还需要为取出来的数安排临时存储空间,在 CASL 中,被取出的数都放在寄存器 GR 中。寄存器共有 GR0～GR4 5 个,可选择其中一个使用。在 CASL 中,取数指令有两种形式,一种是直接取数指令,另一种是间接取数指令。直接取数指令是在取数指令中用标号地址直接给出被取数的所在单元地址。间接取数指令是在取数指令中给出与取数地址有关的参数,借助规定的计算方法确定出被取数据的所在单元地址。

存数是指将 CPU 寄存器中当前暂存的数据,放回内存储器的某个单元的操作,数据的移动方向是由 CPU 到内存。在存数指令中要给出欲存数据当前所在的寄存器号,再给出数据将要存入内存的单元地址。存数也有个寻址的问题,如果直接给出存数单元的地址,称为直接寻址;如果间接给出存数单元的地址,称为间接寻址。对应这两种寻址方式,在 CASL 中有直接存数和间接存数等两种指令形式。

CASL 中的传送是指把数据送到寄存器 GR 中,广义的传送是指寄存器之间、存储单元之间、寄存器与单元之间的数据相互交换。在数据的传送中有两个问题必须弄清楚,一是数据由哪里传来?二是传来的是什么数据。CASL 中传送指令有两种形式,一种是直接传送,另一种是间接传送。所谓直接传送是指被传送的数据已设定在传送指令中,人们常把这种传送称之为立即数或立即寻址。直接传送明确指出,被传送数的源发地就在本指令,而数据即可在本指令中看到而不需要在别处寻找;间接传送则需要通过间接算法来确定被传送的数据。传送指令的职能是传送数据,它并不管传送数据是什么属性,被传送的数据可能是参加运算的数,也可能是用于比较的常数,还可能是内存单元的地址等。数的属性并不是由传送指令本身规定的,而是由使用者如何使用该数所决定的。因此,把传送指令解释成送地址是不全面的。传送指令与取数指令有相同之处,也有不同之处。相同之处在于数据的终点都是寄存器,也就是说如果不考虑传送、取数操作过程的话,两种指令的操作结果都在 GR 中。两种指令的不同之处是,传送指令只传递寻址结果,取数则是取其寻址结果中的内容。

3.1 取数的实现

3.1.1 直接取数指令

(1) 指令形式

标号	操作码	操作数
[标号名]	LD	GR, ADR

(2) 指令功能

使用本指令可将由 ADR 设定的单元中的数取至由 GR 设定的寄存器中。标号名表示本指令在程序中的位置,如本指令不被其他指令调用,则可以省去标号名。LD 是取数指令的功能代码,是 LOAD 的缩写。GR 是存放取出数的寄存器,可选用 GR0～GR4 的任一个。ADR 是被取数的单元地址,可用标号地址给出。而被取数由伪指令定义。

(3) 指令用法

取单元 DY 中的数放到寄存器 GR1 中的程序如下:

```
①   QS      LD      GR1, DY ; (DY)→GR1,即 26→GR1。
            …
DY      DC      26
            …
```

指令①为直接取数指令,QS 为本指令的标号名。执行本指令可以以把 DY 中的十进制数 26 取到寄存器 GR1 中,而 DY 中的值不变。数据交换如图 3.1 所示:

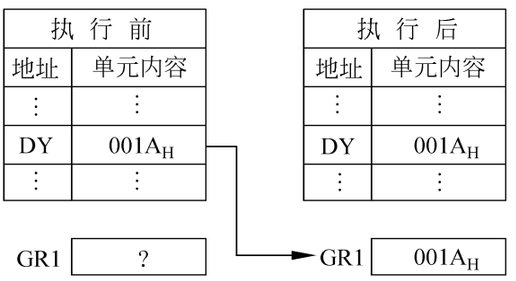


图 3.1 取数的实现

由图示可以看出,数据由 DY 单元被取至 GR1,且 DY 中的值不变。GR1 在执行本指令之前不管存有何值,一旦执行了本指令则被新值取代。

3.1.2 间接取数指令

(1) 指令形式

标号	操作码	操作数
[标号名]	LD	GR, ADR, XR

(2) 指令功能

使用本指令可以把由 ADR + (XR)所设定的内存单元地址中的数取出,并放在由 GR 所设定的寄存器中。

间接取数与直接取数的不同在于寻址,即确定取数的地址。用 E 表示有效地址,则取数地址分别为

E = ADR　　;直接取数的地址

E = ADR + (XR)　;间接取数的地址

不难看出,直接取数指令的取数地址在取数指令中由 ADR 直接给出;间接取数指令的取数地址由 ADR 加上 XR 中的内容给出。XR 称为变址寄存器,XR 中的内容在确定取数地址中是个间接成分。

(3) 指令用法

利用变址寄存 XR 的变化连续取数:

```
①   LQS     LD      GR2, SDZ, GR1
            …
SDZ     DS      12
            …
```

假定指令①执行之前的状态如下:

第一次执行指令①,取数地址及内容如下:

取数地址 E = ADR + (XR)

= SDZ + (GR1)

= SDZ + 0

= SDZ

取出的内容:(SDZ) = 0011_H

GR2

0011_H

修改 GR1,即(GR1) + 1→GR1。GR1 中变为 0001_H。

第二次执行指令①,取数地址及内容如下:

取数地址 E = ADR + (XR)
= SDZ + (GR1)
= SDZ + 1

取出的内容:(SDZ + 1) = 0016_H GR2 0016_H

修改 GR1,即(GR1) + 1→GR1。GR1 中变为 0002_H。

第三次执行指令①,取数地址及内容如下:

取数地址 E = ADR + (XR)
= SDZ + (GR1)
= SDZ + 2

取出的内容:(SDZ + 2) = 0022_H GR2 0022_H

概括以上过程可以得出,GR1 作为指令①的变址寄存器,其内容从初始状态开始,每执行一次指令①,GR1 的内容修改一次,即加 1。继而执行指令①去实施取数时,就改变了取数地址,取出的内容就是下一条数据了,从而实现了连续取数的目的。如何实现对 GR1 的修改,将在传送指令中介绍。

3.2 存数的运用

3.2.1 直接存数指令

(1) 指令形式

标号	操作码	操作数
[标号名]	ST	GR, ADR

(2) 指令功能

使用本指令可以把寄存器 GR 中的数存于由 ADR 设定的单元中去。标号名表示本指令在程序中的位置,如本指令不被其他指令调用,则可以省去标号名。ST 是 STORE 的缩写,GR 可在 GR0～GR4 任选一个使用,ADR 是内存单元的地址,用标号地址给出,并由伪指令定义。

(3) 指令用法

将寄存器 GR1 中的数存于内存单元 DY 中

```
①   CS    ST    GR1, DY    ;(GR1)→DY
        …
    DY    DS    1
        …
```

指令①为直接存数指令,CS 是本指令标号名。执行本指令可以把寄存器 GR1 中的数存到标号为 DY 的单元中去,而 GR1 中的内容不变。数据交换如图 3.2 所示:

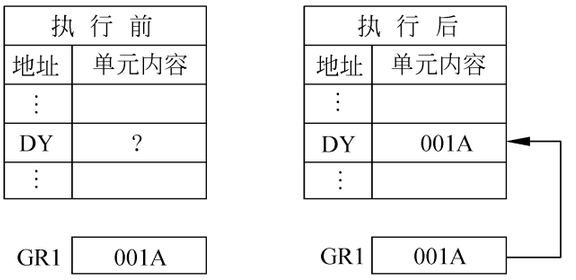


图 3.2　存数的实现

由图 3.2 可以看出,数据的交换是由寄存器到内存单元,与取数过程正好相反。

3.2.2　间接存数指令

(1) 指令形式

标号	操作码	操作数
[标号名]	ST	GR, ADR, XR

(2) 指令功能

使用本指令可以把寄存器 GR 中的数存于由 ADR + (XR)所设定的单元中去。标号名表示本指令在程序中的位置,如果本指令不被其他指令调用,则可以省去标号名。寄存器 GR 可在 GR0～GR4 中任选一个使用。ADR 为存数单元的首地址,XR 为变地址寄存器。存数单元需由伪指令定义。如果用 E 表示有效地址,则存数地址由下式计算:

$$E = ADR + (XR)$$

而直接存数的有效地址为 E = ADR。

(3) 指令用法

把寄存器 GR1 中的数,存于以 DY 为首地址的单元中:

```
①   LCS     ST      GR1, DY, GR2
            …
    DY      DS      5
            …
```

假定指令①在执行前变址寄存器 GR2 的初始状态为 0,且随着指令①的每次执行,变址寄存器的内容逐次加 1。被存数(GR1) = $FFFF_H$,则 5 次执行指令①,可将 GR1 的内容存入连续的 5 个单元中,如表 3.1 所示。

表 3.1　变址寄存器的变化

存数操作	变址寄存器 GR2 状态	有效地址	被存数	单元地址	单元内容
第 1 次	(GR2) = 0	E = DY + 0	(GR1)→	DY + 0	$FFFF_H$
第 2 次	(GR2) = 1	E = DY + 1	(GR1)→	DY + 1	$FFFF_H$
第 3 次	(GR2) = 2	E = DY + 2	(GR1)→	DY + 2	$FFFF_H$
第 4 次	(GR2) = 3	E = DY + 3	(GR1)→	DY + 3	$FFFF_H$
第 5 次	(GR2) = 4	E = DY + 4	(GR1)→	DY + 4	$FFFF_H$

不难看出,变址寄存器中内容的改变,直接影响存数单元地址的改变。因此,在间接存数指令中,借助变址寄存器的连续、有规律的变化可以形成连续的存数空间,从而实现成组的存储操作。当然,如何改变变址寄存器的内容?怎样控制存数次数?还需要借助其他指令实现。

把寄存器 GR1 中的数,存于以 M3 为末地址的 3 个单元中:

```
②   LCS     ST      GR1, M3, GR2
        …
M1      DS      1
M2      DS      1
M3      DS      1
        …
```

执行指令②之前,把变址寄存器的初始状态设定为 0。此后每执行一次指令②,变址寄存器 GR2 的内容减 1,这样就可以实现将 GR1 的内容存于以 M3 为末地址的连续的 3 个单元中,如表 3.2 所示。

表 3.2　改变变址寄存器与内存单元的内容

变址寄存器的内容变化	形成存数地址 E = M3 + (GR2)	内存单元	
		地址	内容
		…	…
改变(GR2) = -2	E = M3 + (-2) = M1	M1	(GR1)
改变(GR2) = -1	E = M3 + (-1) = M2	M2	(GR1)
初始(GR2) = 0	E = M3 + 0 = M3	M3	(GR1)

变址寄存器的内容变化,可以在某一个初始值由小到大每次加 1,也可以在某一个初始状态下由大到小每次减 1,还可以在某一个初始状态下加、减某一个常数而变化。由于变址寄存器的多种变化,可以形成由小到大相差为 1 的存数地址,也可以形成由大到小相差为 1 的存数地址,还可以形成以某一个地址为基准,向下、向上有规律变化的存储空间。由此可以看出,变址寄存器在成组存数及其他成组操作中的作用。这个作用可以归结为寻址,由于在指令中有了变址寄存器的作用,使得寻址,即寻找操作数的地址变得灵活多了。

3.3　传送的功能与作用

3.3.1　直接传送指令

(1) 指令形式

标号	操作码	操作数
[标号名]	LEA	GR, ADR

(2) 指令功能

执行本指令可将由 ADR 设定的内容传送到寄存器 GR 中。ADR 可以是十进制常数或标号地址。标号名表示本指令在程序中的位置,如本指令不被其他指令调用,则可以省略标号名。LEA 是本指令的功能代码,它是 LAAD EFFECTIVE ADDRESS 的缩写。GR 是保存传送内容的寄存器,可在 GR0～GR4 中任选一个使用,ADR 是设定的传送内容。执行本指令可由 GR 中的值形成 FR 的状态。

(3) 指令用法

将常数传送到寄存器 GR 中:

```
①   CCS     LEA     GR1, 6          ;6→GR1
```

执行指令①可将常数 6 传送到寄存器 GR1 中。

```
②           LEA     GR2, -1         ; -1→GR2
```

指令②省去了标号,执行该指令可将 -1 传送到 GR2 中,由于是负数,在计算机中用补码表示,所以 GR2 所接到的数据为

GR2: $FFFF_H$

```
③   CCC     LEA     GR3, 65535
```

指令③可将常数 65535 传送给寄存器 GR3。$65535 = 2^{16} - 1$,在 16 位长的寄存器中为全 1。显然,指令③所传送的 65535 与指令②所传送的 -1 在计算机中的表现形式是完全一样的。

将标号地址传送到寄存器 GR 中

```
④   CDZ     LEA     GR4, DY
            …
DY          DS      1
            …
```

执行指令④可将 DY 传送给寄存器 GR4。假如 DY 所对应的实际地址为 100,那么传入 GR4 的内容为 0064_H。

3.3.2　间接传送指令

(1) 指令形式

标号	操作码	操作数
[标号名]	LEA	GR, ADR, XR

(2) 指令功能

使用本指令可以把由 ADR + (XR)所指定的内容传送给寄存器 GR。而 ADR + (XR)所指定的内容可能是数,也可能是地址。间接传送指令与直接传送指令的根本不同在于表述传送成分的不同。如果把 E 作为传送对象,那么

E = ADR;由 ADR 直接表述传送内容

E = ADR + (XR);间接表述传送内容

如果把 E 看成是一个地址,则 E = ADR 为一个直接地址,而 E = ADR + (XR)是一个间接地址;如果把 E 看成是一个数,则 E = ADR 为一个立即数,而 E = ADR + (XR)是一个非立即数。执行本指令,可由 GR 中的值形成标志寄存器 FR 的状态。若(GR) > 0,则(FR) = 00;(GR) = 0,则(FR) = 01;(GR) < 0,则(FR) = 10。

(3) 指令用法

传送间接地址到寄存器 GR

```
①   CSJ     LEA     GR1, DY, GR2    ;(GR2) + DY→GR1
            …
    DY      DS      5
            …
```

执行指令①,如果变址寄存器 GR2 中的值为 2,则传送状况如图 3.3 所示:

两个寄存器之间的内容传送

```
②   GCS     LEA     GR1,0,GR2       ;(GR2)→GR1
```

执行指令②可将 E→GR1。因为 E = ADR + (XR) = 0 + (GR2) = (GR2),所以该指令完成的传送为(GR2)→GR1。

寄存器的内容加 1

```
③   GJ1     LEA     GR1,1,GR1       ;(GR1) + 1→GR1
```

执行本指令可将 E→GR1。因为 E = ADR + (XR) = 1 + (GR1) = (GR1) + 1,所以指令③所完成的传送为(GR1) + 1→GR1。如果把 GR1 看成是个计数器,那么该指令的功能就是计数器加 1。为什么间接传送指令具有寄存器加 1 的功能呢?其关键是把变址寄存器与接受传送内容的寄存器设定为同一个寄存器。利用这种指令形式还可以完成寄存器加某一个常数的操作。

寄存器的内容减 1

④　GJA1　LEA　GR2, - 1,GR2　;(GR2) - 1→GR2

执行指令④可将 E→GR2。因为 E = ADR + (XR) = - 1 + (GR2) = (GR2) - 1,所以该指令完成的操作为(GR2) - 1→GR2。

概括以上内容可以得出,间接传送指令可以借助变址寄存器的内容依次变化实现成组地址传送;将 ADR 设定为 0 可以完成寄存器之间的内容传送;使用同一个寄存器,并将 ADR 设定为 1 或 - 1,则可实现计数器加 1 或减 1 操作;如果把 ADR 设定为某一个常数,则可以完成寄存器加、减某一个常数的操作。可见,间接传送指令的功能并不局限于一般的传送,灵活地设置 ADR 和 XR 可以编写出多种不同功能的指令来。

此外,传送指令的执行结果会直接影响到标志寄存器 FR 的状态。由于传送结果在 GR 中,所以 GR 中的状态为正、为负还是为零就成了按大于、按小于、按等于转移的前提条件。于是常使用如下指令来形成转移的条件,

⑤　ZYTJ　LEA　GR1,0,GR1　;(GR1)→GR1

从表面上看,指令⑤是一个空操作,它完成的是 GR1 的值又原原本本地传给了 GR1。实际上,执行指令⑤可以根据 GR1 中的值做出相应的转移操作。因此,指令⑤的作用就如同一条比较指令。

3.4　程序设计训练

【问题 3-1】　已知以 SJ 为首地址的 5 个数据,设计一个程序依次将 5 个数取到寄存器 GR2 中?

【方法分析】　假定内存中的 5 个数分布如下:

地址	数据
SJ	0010_H
+ 1	0016_H
+ 2	0028_H
+ 3	0032_H
+ 4	0009_H

如果这 5 个数都有各自独立的标号地址,那么用 5 条取数指令就可以完成将其取到寄存器的任务。而现在的问题是 5 个数只有一个存储首地址,各个存数单元都是由地址偏移量而区分的,因此确定数据的地址应该借助变址。运用变址有如下两种方法:

第一种方法是将首地址 SJ 置于寄存器之外,地址偏移量放在变址寄存器之中,从一个初始值开始让其逐次变化,取数指令的形计为

```
LD   GR2,SJ,GR1
```

其中,SJ 相当一个基址,GR1 为变址,GR1 的内容由初始值 0 开始逐次变化 0,1,2,3,4,于是可以得到取数地址 E = SJ + (GR1),即 SJ + 0, SJ + 1, SJ + 2, SJ + 3, SJ + 4。使用这种方法每取一次数,都要修改一次变址寄存器的内容。

第二种方法是将数据首地址置于寄存器中,地址偏移量作为其修正值,形式为

```
LEA   GR1,SJ      ;将首地址 SJ 传到 GR1 中
LD    GR2,0,GR1   ;以 GR1 中的内容加 0 为地址取数
LD    GR2,1,GR1   ;以 GR1 中的内容加 1 为地址取数
…     …           …                    …
```

这种用法,其寻址方式很像寄存器间接寻址。

【参考程序】

程序 1:

```
SFA   START
      LEA   GR1,0          ;变址寄存器 GR1 赋初值。
①LD   GR2,SJ,GR1    ;取 SJ+ (GR1) = SJ+ 0 中的数。
②LEA  GR1,1,GR1     ;变址寄存器加 1,(GR1) + 1 = 0 + 1→GR1。
③LD   GR2,SJ,GR1    ;取 SJ+ (GR1) = SJ+ 1 中的数。
④LEA  GR1,1,GR1     ;变址寄存器加 1,即 2→GR1。
⑤LD   GR2,SJ,GR1    ;取 SJ+ (GR1) = SJ+ 2 中的数。
⑥LEA  GR1,1,GR1     ;变址寄存器加 1,即 3→GR1。
⑦LD   GR2,SJ,GR1    ;取 SJ+ (GR1) = SJ+ 3 中的数。
⑧LEA  GR1,1,GR1     ;变址寄存器加 1,即 4→GR1。
⑨LD   GR2,SJ,GR1    ;取 SJ+ (GR1) = SJ+ 4 中的数。
      EXIT
SJ    DS    5
      END
```

【方法评述】 不难看出,在本程序中①、③、⑤、⑦、⑨指令是相同的,②、④、⑥、⑧指令也是相同的。如果今后采用循环操作,①～⑨可以压缩为两条指令。下面是第二种方法所对应的程序。

程序 2:

```
SCX   START
      LEA   GR1,SJ         ;数据首地址→GR1
      LD    GR2,0,GR1      ;取第 1 个数据
      LD    GR2,1,GR1      ;取第 2 个数据
      LD    GR2,2,GR1      ;取第 3 个数据
      LD    GR2,3,GR1      ;取第 4 个数据
      LD    GR2,4,GR1      ;取第 5 个数据
      EXIT
SJ    DS    5
      END
```

【问题 3-2】　求自然数 1～5 之和,并将结果存于 GR0 中。

【方法分析】　虽然我们尚未学习加法指令,但我们可以借助间接传送指令来完成自然数的累加问题。当然,像这种问题在高级语言中只需使用一个赋值语句就可以表达出来。

【参考程序】

```
CSQH    START
        LEA    GR1,0
        LEA    GR1,1,GR1     ;1 + (GR1) = 1 + 0→GR1
        LEA    GR1,2,GR1     ;2 + (GR1) = 2 + 1→GR1
        LEA    GR1,3,GR1     ;3 + (GR1) = 3 + 2 + 1→GR1
        LEA    GR1,4,GR1     ;4 + (GR1) = 4 + 3 + 2 + 1→GR1
        LEA    GR1,5,GR1     ;5 + (GR1) = 5 + 4 + 3 + 2 + 1→GR1
        LEA    GR0,0,GR1     ;1 + 2 + 3 + 4 + 5 之和→GR0
        EXIT
        END
```

【方法评述】　多个数据的求和一般是使用成组操作,借助循环累加来完成的。本题所介绍的求和,其目的是为了说明传送指令不单单是传送功能,还具有相加功能,而这个相加功能是寄存器中的值与指令中的常数相加。这一点与加法指令也不相同。

习　题　3

1. 按所给定的要求写出相应的指令。
   (1) 将 GR0 置 0
   (2) 将 GR1 加 1
   (3) 将 GR2 减 2
   (4) 将 GR3 的值传送给 GR4
   (5) 将 GR4 的值减 4 传送给 GR0
   (6) 根据 GR3 的值设定标志寄存器 FR
2. 阅读下列程序,按要求填写有关结果。
   (1) 执行本程序后,用十六进制数填写 GR0、GR1、GR2、GR3 中的内容。

```
L321    START
        LD   GR0,D1          GR0
        LD   GR1,D2          GR1
        LD   GR2,D3          GR2
        LD   GR3,D4          GR3
        EXIT
D1      DC   65535
D2      DC   - 32768
D3      DC   - 1
```

```
D4      DC   #FFFF
        END
```

(2) **执行本程序后**,DY1,DY2 **单元分别为二进制何值**?

```
L322    START
        LD   GR1,A
        ST   GR1,DY1        DY1
        LD   GR2,B
        ST   GR2,DY2        DY2
        EXIT
A       DC   '123'
B       DC   '234'
DY1     DS   1
DY2     DS   1
        END
```

(3) **执行本程序后**,GR1、GR2、GR3 **中为十六进制形式的何值**?

```
L323    START       KS
D1      DC   1              GR1
        DC   2              GR2
        DC   3              GR3
KS      LEA     GR1,0
        LD      GR1,D1,GR1
        LD      GR2,D1,GR1
        LD      GR3,D1,GR2
        EXIT
        END
```

3. **按要求完善程序。**

(1) **单元中的内容单向传送**,(A)**到**(B);

```
L331    START
        LD      ①    ,A
        ②       GR1,B
        EXIT
A       DS      1
③       DS      1
        END
```

(2) **寄存器的内容单向传送**,(GR1)**到**(GR2);

```
L332    START
        ST      ①    ,GD
```

```
        LD      ②      ,GD
        EXIT
GD      DS      1
        END
```

(3) 标号地址 DZ 传送到 DY 单元中。

```
L333    START
        ①       GR3,DZ
        ST      GR3,②
        EXIT
③       DS      1
DY      DS      1
        END
```

4. 编写程序。

(1) 设计一个将单元 X 和单元 Y 的内容互换的程序。

(2) 设计一个将寄存器 GR1 的内容与 GR2 内容互换的程序。

(3) 设计一个 M 单元的内容加 1,N 单元内容减 1 的程序。

(4) 设计一个将以 DY 为首地址连续 3 个单元冲 0 的程序。

(5) 设计一个将以 DZ 为末地址连续 3 个单元均送常数 12 的程序。

(6) 设计一个程序,依次将数据 5,4,3,2,1 相加求和后,结果存于 GR0 中。

第 4 章

算术运算及算术操作

在通常情况下,算术运算一般是指加、减、乘、除等运算。在 CASL 中,只设置了加法指令和减法指令,没有给出乘法和除法指令。但在 CASL 中设置了算术左移指令和算术右移指令,使用算术左、右移指令分别可以完成乘法和除法。因此,我们把算术左移和算术右移指令称为算术操作。

对于一个正整数 $8_{10} = 1000_2$,若在计算机中表示成 16 位的定点整数为

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

数的符号位↑　　↑小数点隐含在最低位之后

将其左移 1 位,

0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

其值变为 16,即左移一位相当乘以 2,即 8×2 = 16。

将其左移 2 位,

0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

其值变为 32,即左移两位相当乘以 $2^2 = 4$,即 8×4 = 32。

可以得出,一个数左移一位相当乘以 2^1,左移两位相当乘以 2^2,左移 n 位相当乘以 2^n。利用数的左移操作,可以完成乘法运算。但是,又可以看出左移所完成的乘法运算,其乘数都是 2 的幂。如果一个数乘以 3,5,6,7,9…,又该怎样实现呢?一般的方法是采用一种等价的算法,例如:8×3,可变为 3×8,即将 3 左移 3 位,而不是将 8 左移;也可以将 8×3 改写为 8×(2 + 1),这样可将 8 左移一位再加上 8 来实现 8×3 的运算。

除法运算,可以借助算术右移。例如,将 8 右移一位则有,

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

其值为 4,这相当于完成了 8÷2 = 4 的运算。不难推断,将某数右移一位相当于除以 2,右移两位相当于除以 4,右移 n 位相当于除以 2^n。但是,由于操作数在计算机中都是补码,所以右移后,即商也应该是补码,关于这方面的内容将在下面介绍。

4.1　加法运算

4.1.1　直接加法指令

(1) 指令形式

标号	操作码	操作数
[标号名]	ADD	GR, ADR

(2) 指令功能

执行本指令可把寄存器 GR 中的内容作为被加数,ADR 所设定的内存单元中的数作为加数,二者相加之和存放于寄存器 GR 中,即(GR) + (ADR)→GR。指令中的标号名,当本指令不被其他指令调用时则可省去。ADD 是 ADDARITHMATIC 的缩写,为本指令的功能代码。GR 是使用者所设定的寄存器,可在 GR0～GR4 中任意选用一个。ADR 是使用者所指定的内存单元地址,可用标号地址给出。

本指令相加的结果是正、零、负,会直接影响标志寄存器 FR 的状态。结果为正,(FR) = 00;结果为零,(FR) = 01;结果为负,(FR) = 10。

(3) 指令用法

GR1 中的值与 DY 单元中的值相加

```
①   JF      ADD     GR1, DY    ; (GR1) + (DY)→GR1
            …
DY      DC      # 0010
            …
```

指令①为直接加法指令,所谓直接加法是指内存单元地址 E = ADR 直接由 DY 给出。在计算机中的加法操作如图 4.1 所示。

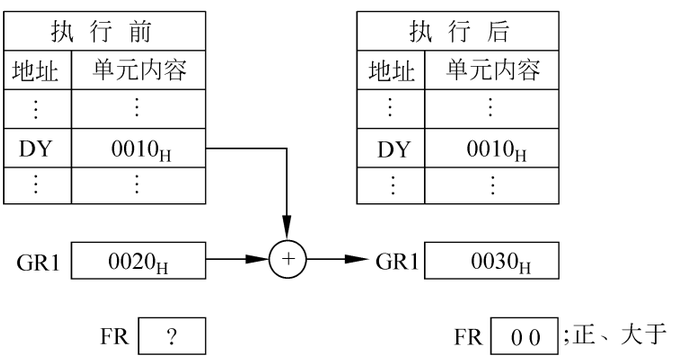


图 4.1　直接加法操作

可以看出,执行前与执行后,单元 DY 中的值不变。GR1 中的值在执行前为被加数,而在执行后变成了二数之和。标志寄存器在执行之前无论是何值,而在执行后却变成了 00。这是因为二数之和 0030_H 是个大于零的正数。

GR2 中的值与 DATA 中的值相加

```
②   JAFA    ADD     GR2,DATA   ; (GR2) + (DATA)→GR2
      …       …
    DATA    DS      1
      …       …
```

指令②在形式上与指令①完全相同,只是 DATA 为单元的标号地址,而指令①中的 DY 为标号地址常数。

4.1.2 间接加法指令

(1) 指令形式

标号	操作码	操作数
[标号名]	ADD	GR,ADR,XR

(2) 指令功能

执行本指令可把寄存器 GR 中的值作为被加数,将有效地址 E = ADR + (XR)中的值作为加数,二者相加之和存放于 GR 中。指令中,ADR 可视为一个基址,XR 为变址寄存器,其中的值可视为地址偏移量。加数所在的地址由基地与偏移地址共同形成,这是间接加法指令与直接加法指令惟一的不同之处。此外,本指令的操作结果直接影响标志寄存器 FR 的状态。

(3) 指令用法

将 GR2 中的值与 KS + 2 单元中的值求和

```
①   JJAF    ADD     GR2,KS,GR3   ; (GR2) + (KS + (GR3))→GR2
      …       …
    KS      DS      3
      …       …
```

指令①为间接加法指令,所谓间接加法是指操作数的内存单元地址 E 由 KS 和 GR3 的内容间接形成,即 E = KS + (GR3)。其中,GR3 为变址寄存器。如果在执行本指令时 GR3 中的值为 2,那么所形成的操作地址为 E = KS + (GR3) = KS + 2。在计算机中的加法操作如图 4.2 所示。

由示意图可以看出,执行前 GR2 中是被加数,做加法时先由 KS 和变址寄存器中的内容相加形成加数地址 KS + 2,然后取其内容与被加数相加,相加之和存于 GR2 中。由于结果为正数,所以标志寄存器 FR 被置为 00_H。此外,在形成加数的地址中,KS 为起始地址,即基址,GR3 的内容为偏移量,即变址。若以 KS 为基准,有规律地调解变址就可以找到处于任何位置的加数。因此,灵活地运用变址可以自如地实现寻址。

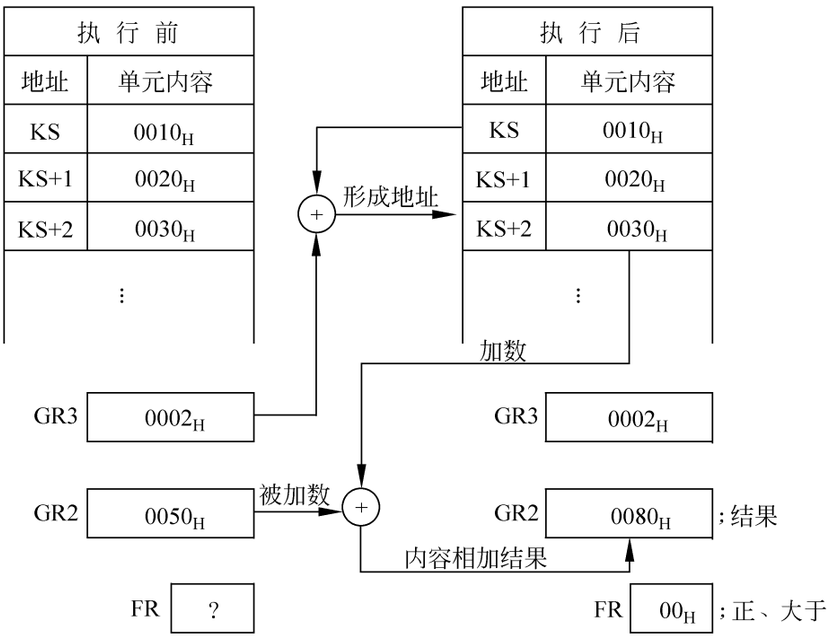


图 4.2　间接加法操作

4.2　减法运算

4.2.1　直接减法指令

(1) 指令形式

标号	操作码	操作数
[标号名]	SUB	GR, ADR

(2) 指令功能

执行本指令可把寄存器 GR 中的内容作为被减数,将 ADR 所设定的内存单元中的数作为减数,二者相减所得之差存放于寄存器 GR 中,即(GR) - (ADR)→GR。指令中的标号名,当本指令不被其他指令调用时则可省略。SUB 是 SUBTRACT ARITH MATIC 的缩写,为本指令的功能代码。GR 是使用者所设定的寄存器,可在 GR0～GR4 中任选一个使用。ADR 是使用者所指定的内存单元地址,可用标号地址给出。

本指令相减的结果是正、零、负,会直接影响标志寄存器 FR 的状态。结果为正,(FR) = 00;结果为零,(FR) = 01;结果为负,(FR) = 10。

(3) 指令用法

```
①   JFAS    SUB    GR1,DYS    ; (GR1) - (DYS)→GR1
            …
DYS         DS     1
            …
```

指令①为直接减法指令,执行本指令把 GR1 中的值减去由 DYS 直接给出的内存单元地址中的值,其结果存于 GR1 中。在计算机中减法操作如图 4.3 所示。

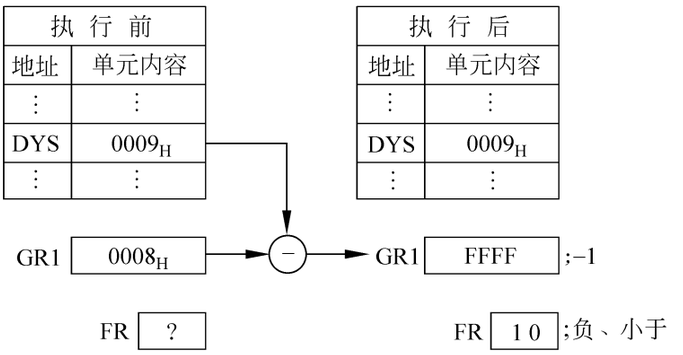


图 4.3 直接减法操作

由示意图可以看出(GR1) - (DYS) = 8 - 9 = - 1。由于在计算机中,数值性的数为补码,所以 - 1 在 GR1 中为 FFFF。由于相减的结果为负,使得标志寄存器 FR 产生相应的状态。

4.2.2 间接减法指令

(1) 指令形式

标号	操作码	操作数
[标号名]	SUB	GR,ADR,XR

(2) 指令功能

执行本指令把寄存器 GR 中的值作为被减数,将地址 E = ADR + (XR)中的内容作为减数,二者相减之差存于寄存器 GR 中。在本指令中减数所在的单元地址不是由 ADR 直接给出的,而是由 ADR 与 XR 中的内容相加而形成的,这是间接减法指令与直接减法指令的根本不同点。此外,该指令的操作结果会对标志寄存器 FR 产生直接影响。当操作结果为正数时,(FR) = 00;操作结果为零时,(FR) = 01;操作结果为负数时,(FR) = 10。

(3) 指令用法

将 GR2 中的值与 A + 2 单元中的值求差。

```
①  JJUS   SUB   GR2,A,GR3   ; (GR2) - (A + (GR3))→GR2
   …      …
   A      DC    1
   B      DC    2
   C      DC    3
   …      …
```

指令①为间接减法指令,执行指令①可将 GR2 中的内容与间接地址 E = A + (GR3)中的相减,其结果存于 GR2 中。二数之差的属性为正、零、负可使标志寄存器 FR 的值做相应的改变。相减示意图如图 4.4 所示。

由示意图可以看出,执行前 GR2 中为被减数,执行后为操作结果。做减法时,首先形成减数的地址:E = A + (GR3) = A + 2。而 A + 2 的地址为 C,所以减法操作为(GR2) - (C) = 0003_H - 0003_H = 0000_H。由于运算结果为 0,所以将标志寄存器置为 01,以示结果

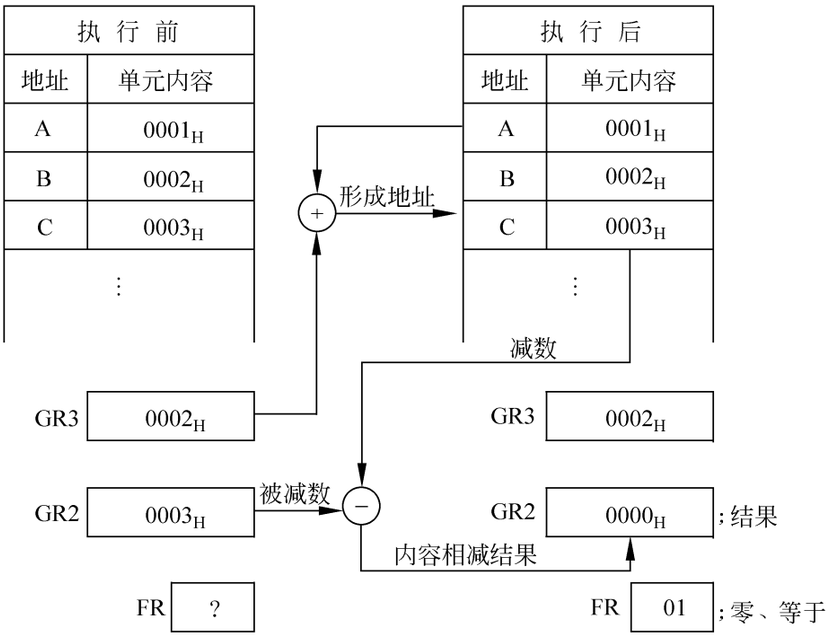


图 4.4　间接减法操作

为 0。由本例可以看出,变址寄存器在形成操作数的过程中是提供地址偏移量的作用。若变址寄存器 GR3 的值设定为 0,那么间接操作与直接操作就完全一样了。

4.3　算术左移操作

4.3.1　直接算术左移指令

(1) 指令形式

标号	操作码	操作数
[标号名]	SLA	GR, ADR

(2) 指令功能

算术左移是一种符号位不动的左移,直接算术左移是指左移位数直接给出的算术左移。

由于将某数算术左移 1 位相当于该数除以 2;左移两位相当于除以 4;左移 n 位相当于除以 2^n。因此,借助算术左移可以实现乘法。

执行本指令可以将寄存器 GR 中的数作为有符号数的补码,符号位不动,其他位依次左移由 ADR 直接设定的位数,而左移出现的空位自动补上 0,移位后的结果仍存于 GR 中。

指令中的标号名,如其他指令不调用本指令则可省略不写。SLA 是 SHIFT LEFT ARITHMETIC 的缩写,为本指令的功能代码。GR 为被左移的寄存器,可在 GR0～GR4 中任用一个。ADR 为直接给出的左移位数,其值为大于零的正整数。如果为 0,则不产生移位动作;如果为负数,则为是错误指令。此外,移位后的结果为正、零、负将使标志寄存器产生相应的状态。

(3) 指令用法

将寄存器 GR1 的内容算术左移 3 位

① SZYI SLA GR1,3 ;(GR1)/3→GR1

指令①将 GR1 中的数作为有符号数的补码,符号位不动,其他位依次左移 3 位,左移出现的空位自动补 0。左移后的结果仍在 GR1 中,但该结果对标志寄存器 FR 有直接影响。下面是算术左移的实例分析。

【例 1】

在例 1 中,GR1 在左移前其中的内容为全 1,这个值实际上为 - 1 的补码。全 1 算术左移 3 位后为 $FFF8_H$,这个值应该是 - 8 的补码。将补码 FFF8 返回原码为 8008_H,这个值确实为 - 8。从而证明 - 1 左移 3 位相当于 $-1\times2^3=-8$。由于左移结果为负,所以 FR 置为 10。

【例 2】

在例 2 中,GR1 左移前的内容为 $0080_H=2^7=128$。算术左移 3 位后的值为 $0400_H=2^{10}=1024$,这个数正好是 128×2^3,即 128 左移 3 位后,其值为 1024。由于左移后的结果为正,所以标志寄存器 FR 被置为 00。

4.3.2 间接算术左移指令

(1) 指令形式

标号	操作码	操作数
[标号名]	SLA	GR,ADR,XR

(2) 指令功能

执行本指令可将寄存器 GR 中的内容,其符号位不动其他位依次左移 E 位,而 E =

ADR + (XR)。也就是说,左移位数由 ADR 与 XR 的内容共同设定,但移位值必须是正整数。左移后所得到的结果仍然保留在 GR 中,而该结果是正、零、负均会使得标志寄存器 FR 产生相当的状态。间接算术左移指令与直接左移指令惟一的不同是在左移位数的设定上,直接左移位数由 ADR 直接给出,间接左移指令由 ADR + (XR)间接设定。

(3) 指令用法

将寄存器 GR2 的内容算术左移 4 位

① `SJZA　SLA　GR2,4,GR1` ;(GR2)↑4→GR2,0→GR1

指令①ADR = 4,为实现算术左移 4 位,GR1 中可设定为 0。这样,左移位数 E = ADR + (XR) = 4 + (GR1) = 4 + 0 = 4。

② `SJZB　SLA　GR2,0,GR1` ;(GR2)↑4→GR2,4→GR1

指令②ADR = 0,为实现算术左移 4 位,GR1 中可设定为 4。

将寄存器 GR2 的内容依次按照 5、4、3、2、1 进行算术左移

③ `SJZC　SLA　GR2,5,GR1` ;(GR2)↓→GR2,0、-1、-2、-3、-4 →GR1

指令③ADR = 5,则(GR1)依次设定为 0、-1、-2、-3、-4 就可以实现左移 5 位、4 位、3 位、2 位、1 位的左移操作。

从以上 3 条移位指令可以看出,不管 ADR 设定为何值,但它终究是个常数,使移位位数发生变化的是变址寄存器 GR1。若在指令执行中设法改变变址寄存器内容,使其按某种规律变化,就能实现相应的移位操作。这说明变址寄存器不光是可以改变操作数的地址,也可以改变移位的位数,以满足在程序设计中对各种移位操作的需要。

4.4　算术右移操作

4.4.1　直接算术右移指令

(1) 指令形式

标号	操作码	操作数
[标号名]	SRA	GR, ADR

(2) 指令功能

算术右移是一种符号位不动的右移,直接算术右移是指直接给出右移位数的算术右移。

由于将某数右移 1 位相当于该数除以 2,右移两位相当于除以 4,右移 n 位相当于除以 2^n。因此,借助算术右移操作可以实现除法运算。

执行本指令可以将寄存器 GR 的内容作为有符号数的补码,使符号位不动,其他位依次右移由 ADR 直接给的位数,而右移出现的空位自动补上与符号位相同的值,即数的符号为 0,则空位补 0;符号位为 1,则空位补 1。

(3) 指令用法

将寄存器 GR1 的内容算术右移 3 位

```
①    SYYI    SRA    GR1,3
```

; (GR1) 3\→GR1

指令①将 GR1 中的数视为有符号数的补码,符号位不动,其他位依次右移 3 位,右移出现的空位自动补上与符号位相同的值。右移结果仍存于 GR1 中,且该结果会影响标志寄存器 FR 的状态。下面是算术右移的实例分析。

【例 3】

在例 3 中,GR1 的内容在右移前为 + 8,右移时符号位不动,尾数的低 3 位被移出。由于符号位为 0,所以右移出现的 3 个空位自动补 3 个 0。右移 3 位的结果相当于 $8 \div 2^3 = 8 \div 8 = 1$,从而可以看出右移操作可以完成除法运算的功能。由于右移结果大于零,因此 FR 中的状态为 00。

【例 4】

在例 4 中,GR1 右移前其值为 $FFF8_H$,右移 3 位将低位的 3 个 0 移出,尾数的高 3 位空出。由于 GR1 中的值为负,所以空出的 3 位自动补上了 3 个 1。GR1 右移 3 位的结果为 $FFFF_H$。因为右移后的结果为负,所以标志寄存器 FR 被置为 10。对比 GR1 右移前和右移 3 位后的值可以发现,右移前 GR1 中为 - 8 的补码,右移 3 位后变为 - 1 的补码。这说明 - 8 右移 3 位相当于 $-8 \div 2^3 = -8 \div 8 = -1$。此外,GR1 在移位前为补码,右移后还应该是补码。为此,在右移中出现的空位自动补上与符号位相同的值,即符号为 0 补 0,符号为 1 补 1,恰恰可以实现右移的结果为补码的目的。

4.4.2 间接算术右移指令

(1) 指令形式

标号	操作码	操作数
[标号名]	SRA	GR,ADR,XR

(2) 指令功能

执行本指令可将寄存器 GR 中的值作为有符号数的补码,符号位不动,尾数依次右移 E 位,而右移后出现的空位自动补上与符号位相同的值。其中,右移位数由 ADR 与 XR 的内容共同间接形成,即 E = ADR + (XR)。而右移后出现的空位自动补上与符号位相同的值,这样可以形成右移结果的补码。间接算术右移的操作结果为正、零、负,可以使得标志寄存器 FR 产生相应的状态。

(3) 指令用法

将寄存器 GR1 中的内容算术右移 1 位

① `SYYA　SRA　GR1,1,GR2` ;(GR2) = 0,(GR1) 1\→GR1

指令①为使(GR1)右移 1 位,移位参数 ADR 设定为 1,(XR)设定为 0,则右移位数 E = ADR + (XR) = 1 + (GR2) = 1 + 0 = 1。

② `SYYB　SRA　GR1,0,GR2` ;(GR2) = 1,(GR1) 1\→GR1

指令②也可以使(GR1)右移 1 位,因为 ADR 设定为 0,XR 的内容设定为 1。这样右移位数 E = ADR + (XR) = 0 + (GR2) = 0 + 1 = 1。

将寄存器 GR1 中的内容依次右移 1、2、3、…位

③ `SYYC　SRA　GR1,1,GR3` ;(GR3) = 0,1,2,…,(GR1)\→GR1

指令③将 ADR 设定为 1,使 GR3 的内容按 0,1,2,…依次变化,于是右移位数 E = ADR + (XR)也随之发生变化。在这个变化中,起关键作用的参数是 GR3,即变址寄存器。由此不难得出,在移位中只要对变址寄存器设定合适的初始,并在指令执行中控制其变化就能实现各种所需要的移位。

4.5　程序设计训练

【问题 4-1】　计算(3 + 5 - 7)×8÷4 = ?

【方法分析】　本题是个算术运算问题,加、减法可用规定的加法和减法指令来描述,乘 8 可用左移 3 位实现,除 4 用右移 2 位完成。

【参考程序】

```
SSYS1   START
        LD      GR1,C3      ;(C3)→GR1,即 3→GR1
        ADD     GR1,C5      ;(GR1) + (C5)→GR1,即 3 + 5→GR1
        SUB     GR1,C7      ;(GR1) - (C7)→GR1,即 3 + 5 - 7→GR1
        SLA     GR1,3       ;(GR1)/3 →GR1,即(3 + 5 - 7)×8→GR1
        SRA     GR1,2       ;(GR1) 2\→GR1,即(3 + 5 - 7)×8÷4→GR1
        EXIT
C3      DC      3           ;定义常数 3
C5      DC      5           ;定义常数 5
C7      DC      7           ;定义常数 7
        END
```

【结果评述】 执行本程序可将计算结果存于 GR1。本程序使用伪指令定义了参加运算的常数 3、5、7,也可用传送指令来形成。如:

```
LEA     GR2,3
LEA     GR3,5
LEA     GR4,7
```

如果用传送的办法形成常数,则在运算中还需要借助工作单元。因为在 CASL 中没有两个寄存器直接相加的指令。比如要实现 3 + 5 = ?,不能用 ADD GR2,GR3,而只能先把一个操作数存到一个工作单元,另一个操作数在寄存器才能进行二数相加。例如:

```
LEA     GR2,3 ; 3→GR2,将一个数存于寄存器
LEA     GR3,5 ; 5→GR3
ST      GR3,DY ; 5→DY,将另一个数存于工作单元
ADD     GR2,DY ; 3+5→GR2
```

这种设计方法可能会使指令增多,使程序加长。

【问题 4-2】 设计一个实现某数的 10 倍的程序。

【方法分析】 对于有乘法指令的汇编来说,某数乘以 10 是个十分简单的问题。而在 CASL 中乘法是借助算术左移来完成,而左移的结果都只能是扩大 2 的幂倍。而 10 不是 2 的整数幂倍,因此扩大 10 倍就不能用一次左移来实现。

令 S 为某数,某数的 10 倍为 S×10。将 S×10 变换为下式,

$$S\times 10 = S\times(8+2) = S\times 8 + S\times 2$$

从该式出发,可以得出可将 S 左移 3 位,再将 S 左移 1 位,并将两次左移结果求和,便可得到 S 的 10 倍。

【参考程序】

```
SSYS2   START
        LD      GR2,S       ;S→GR2
        SLA     GR2,1       ;S×2→GR2
        ST      GR2,WK      ;S×2→WK
        SLA     GR2,2       ;(S×2)×4 = S×8→GR2
        ADD     GR2,WK      ;S×8 + S×2 = S×10→GR2
        EXIT
S       DS      1           ;某数 S
WK      DS      1           ;工作单元
        END
```

【结果评述】 执行本程序,某数 S 的 10 倍结果存于 GR2 中。在程序设计中并没有按照方法分析所得出的思路去设计,而是在 S×2 的基础上再乘以 4,从而得到 S×8。如果把 S×8 和 S×2 分别单独设计然后再相加的话,则程序段如下:

```
LD      GR2,S
```

```
SLA    GR2,1          ;S×2→GR2
ST     GR2,WK         ;S×2→WK
LD     GR2,S
SLA    GR2,3          ;S×8→GR2
ADD    GR2,WK         ;S×8 + S×2 = S×10→GR2
       …
```

可以看出,这种设计方法使用了 6 条基本指令,比参考程序多用一条指令。

【问题 4-3】　设计一个程序,将单元中存放的数字字符数据变为数值数据。

【方法分析】　字符在计算机中是 7 位的 ASCII,每一个字符占用一个字,并在字长 16 位的字中占低 7 位。对于数字字符来说,比如'3'、'4'、'5',其 ASCII 如下:

0 0 0 0 0 0 0 0	0 0 1 1	0 0 1 1	;'3'
0 0 0 0 0 0 0 0	0 0 1 1	0 1 0 0	;'4'
0 0 0 0 0 0 0 0	0 0 1 1	0 1 0 1	;'5'

由'3'、'4'、'5'的 ASCII 可以看出,如果将虚线以左的部分去掉,剩下虚线右侧的低 4 位,就得了所要求的数值数据。而利用算术左移 11 位的办法可以去掉虚线左侧的部分,再右移 11 位则可以得到最后结果。

【参考程序】

```
SSYS3   START
        LD      GR3,ZFSJ    ;字符数据→GR3
        SLA     GR3,11      ;(GR3)11→GR3
        SRA     GR3,11      ;(GR3)11→GR3
        ST      GR3,SZSJ    ;数值数据→SZSJ
        EXIT
ZFSJ    DS      1           ;单元中任一个字符数据
SZSJ    DS      1           ;数值数据结果
        END
```

【结果评述】　执行本程序可将单元 ZFSJ 中的字符数据转换为数值性数据存于单元 SZSJ 中。由于字符的 ASCII 都在一个单元中的低 7 位,而高位为 0。因此,我们可以把字符的 ASCII 本身也作为数值。如将字符'0'～'9'的 ASCII 用十六进制数表示的话,其值为 0030_H～0039_H。而数值 0～9 用十六进制表示为 0000_H～0009_H。不难看出二者之差为 0030_H。由于每个字符数据与其相应的数值数据之差为 0030_H,所以由字符数据转换成数值数据时可以用减去一个常数 0030_H 的方法;而数值数据转换为字符数据时加上一个常数 0030_H。

【问题 4-4】　设计一个程序,将单元 A 中的 5 转换成'5'存于 GR1;将单元 B 中的'6'转换

成 6 存于 GR2 中。

【方法分析】　由于 5 和'5'在计算机中只相差一个常数 0030_H；6 和'6'的差别也是相差一个常数 0030_H，所以实现相互的转换可以用加、减常数 0030_H 的方法。

【参考程序】

```
SZZH4   START
        LD      GR1,A       ;5→GR1
        ADD     GR1,C       ;5 转换为'5'→GR1
        LD      GR2,B       ;'6'→GR2
        SUB     GR2,C       ;'6'转换为 6→GR2
        EXIT
A       DC      5           ;数值常数 5
B       DC      '6'         ;字符常数'6'
C       DC      #0030       ;数字字符与数字相差的常数
        END
```

【结果评述】　执行本程序可以完成数值数据与字符数据的相互转换。在转换中使用了加、减法运算，这是因为数字 0～9 和字符'0'～'9'在计算中都可以看成是一种有符号数，且符号为 0。

【问题 4-5】　设计一个程序求某数的 0.75 倍。

【方法分析】　由于在 CASL 中没有乘、除法指令，也没有小数运算和浮点运算而只有整数运算，所以直接计算某数的 0.75 倍是实现不了的。但是，我们采用某种等价变换还是可以找到解决问题的办法的。

设某已知数为 S，则 S 的 0.75 倍可以用下式表示，

$$0.75\times S=\frac{3}{4}S=\frac{3S}{4} \qquad ①$$

上式虽然可以对 3S 右移两位来完除 4，但 3 乘 S 仍无法进行。为此，将 3 乘 S 进行变换，

$$3S=(2+1)S=2S+S \qquad ②$$

这样就可由②式先算出 3S，然后由①式计算 3S 除以 4。

如果由①式进行变换还可以得出另一种解决办法，

$$0.75\times S=\frac{3}{4}S=\frac{2}{4}S+\frac{1}{4}S=\frac{2S}{4}+\frac{S}{4}=\frac{S}{2}+\frac{S}{4}$$

这样就可以先将 S 右移一位算出 S/2，然后对 S 右移两位算出 S/4，两项相加便得到最后结果。

【参考程序】

```
LDQW5   START
        LD      GR4,S
        SRA     GR4,1       ;S/2→GR4
        ST      GR4,JG      ;S/2→JG
        SRA     GR4,1       ;S/2 右移 1 位得 S/4
        ADD     GR4,JG      ;S/4+S/2=0.75S
```

```
        ST      GR4,JG          ;结果→JG
        EXIT
S       DS      1
JG      DS      1
        END
```

【结果评述】　执行本程序,可将 S 单元的某数的 0 .75 倍算出并存于 JG 单元中。特别需要指出,如果 S 单元中的不是 4 的整倍数,则所算出的结果只是近似值。这种计算误差是因为右移截尾所产生的。例如,S 中的数为 2,即二进制的 10_2,将其右移 1 位为 1_2,再将其右移 1 位为 0,两项相加为 1 + 0 = 1。这个结果为 2 的二分之一,并不等于 2 的四分之三。

【问题 4-6】　设计一个程序,将单元 ZS 中的正数变成负数存于 FS。

【方法分析】　从数的属性上说,由正数变成负数,只需要改变数的符号就可以了。但是,数值性的数在计算机中是以补码形式表示的,而正数的补码和原码是一样的。因此,本题的实质是要求把单元中的正数补码变为负数补码。这样,一个正数的补码与其负数的补码就不只是相差一个符号了。例如:

+1 在 16 位字长中的补码为　0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

-1 在 16 位字长中的补码为　1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

那么,怎样把 +1 变为 -1 呢?方法很简单,只要用 0 减 1 就可以实现。因为 0 - 1 = -1,而 0 减任何正数都应该得到相对应的负数。

【参考程序】

```
ZBFS6   START
        LD      GR1,Z0
        SUB     GR1,ZS
        ST      GR1,FS
        EXIT
Z0      DC      0
ZS      DS      1
FS      DS      1
        END
```

【结果评述】　执行本程序可以把单元 ZS 中的正数(补码)变为该正数对应的负数(补码)。

习　题　4

1. 按要求写出相应指令。

(1) 寄存器 GR1 中的内容加上以 GR2 为地址中的内容。

(2) 寄存器 GR2 中的内容减去以 GR3 加 2 为地址中的内容。

(3) 将寄存器 GR3 中的值算术左移 3 位。

(4) 将寄存器 GR4 中的值算术右移 4 位。

(5) 已知 GR1 中为 2,寄存器 GR0 中的值加上单元 DY 加 2 中的值。

(6) 已知 GR2 中为 3,寄存器 GR1 中的值减去单元 DY 加 3 中的值。

(7) 将寄存器 GR3 中的值按 YW + (GR4)的值算术左移。

(8) 将寄存器 GR4 的内容加上以 GR1 为地址的单元内容。

2. 按要求完善程序。

(1) 将以 WZ 为首地址的数字字符数据变为数值存于以 SZ 为首的单元中。

```
WZBSZ   START
        LEA     GR1,0           ;0→GR1(变址寄存器)。
        LD      GR0,WZ          ;取第 1 个字符→GR0(工作寄存器)。
        SUB     GR0,CS30        ;数字字符减去常数 0030H,即变换。
        ST      GR0,SZ          ;数值→SZ。
        ADD     GR1,CS1         ;修改变址寄存器。
        LD      GR0,____,____   ;取第 2 个字符。
        SUB     ____,____       ;变换。
        ST      ____,____,____  ;数值存入单元。
        ADD     GR1,____        ;修改变址寄存器。
        LD      GR0,____,____   ;取第 3 个字符。
        SUB     ____,____       ;变换。
        ST      ____,____,____  ;数值存入单元。
        EXIT
CS30    DC      #0030
CS1     DC      1
SZ      DS      3
WZ      DC      '268'
        END
```

(2) 已知下列所示的 5 个数据,求其累加和。

地址	数据
	0016H
	0028H
SDZ	0030H
	0080H
	0062H
…	…

```
LJAH    START
        LEA     ____,SDZ
        LEA     GR0,____
        ADD     GR0,0,____
        ADD     ____,1,GR1
        ____    GR0,2,GR1
        ADD     GR0,-1,____
        ADD     GR0,____,____
        EXIT
        DC      #0016
        DC      #0028
SDZ     DC      #0030
        DC      #0080
        DC      #0062
        END
```

3. 阅读程序加上注解,并说出程序的功能。

(1)
```
BHJH    START
        LD      GR1,WS1
        SLA     GR1,11
        SRA     GR1,11
        LD      GR2,WS2
        SLA     GR2,11
        SRA     GR2,11
        LEA     GR1,1,GR2
        LEA     GR2,-2,GR1
        LEA     GR1,-1,GR1
        EXIT
WS1     DC      '1'
WS2     DC      '2'
        END
```

(2)
```
YSBH    START
        LEA     GR1,ZFS
        LD      GR2,ZFS
        SLA     GR2,8
        ADD     GR2,1,GR1
        ST      GR2,YSS
        LD      GR2,2,GR1
        SLA     GR2,12
        ST      GR2,YSDY
        LD      GR2,3,GR1
        SLA     GR2,12
        SRA     GR2,4
        ADD     GR2,YSDY
        ST      GR2,YSDY
        LD      GR2,4,GR1
        SLA     GR2,12
        SRA     GR2,8
        ADD     GR2,YSDY
        ST      GR2,YSDY
        LD      GR2,5,GR1
        SLA     GR2,12
        SRA     GR2,12
        ADD     GR2,YSDY
        ST      GR2,YSDY
        EXIT
ZFS     DC      'AB1234'
YSS     DS      1
```

```
YSDY    DS      1
        END
```

4. 程序设计。

(1) 将 GR1、GR2、GR3 中的值求和,其结果存于 GR0 中。

(2) 求单元 A 中值的 7 倍,结果存于单元 B 中。

(3) 将 x 单元的内容减去 y 单元的内容,加上 z 单元的内容后减 3,其结果存于 GR0 中。

第 5 章

逻辑运算及逻辑操作

逻辑运算源于逻辑代数,逻辑代数源于布尔代数。布尔代数是英国数学家乔治·布尔(George Boole)19 世纪提出来的一种代数,故以布尔命名,称之为布尔代数。布尔代数问世后近一个世纪没被人们所认识,直到 20 世纪 30 年代自动控制出现后,布尔代数的应用才被重视起来。

逻辑代数是研究二值变量的代数,所谓二值变量是指只有两个值的变量。二值变量的两个值一般是用 0 和 1 表示,而 0 和 1 又恰恰是二进制的两个基数字,因此逻辑代数与二进制就自然地结合在一起,成为计算机的运算工具。

在逻辑代数中规定了多种运算及操作,借助这些运算和操作既可以解决逻辑问题,又可以解决算术问题。在 CASL 中给出了 3 种逻辑运算:逻辑乘、逻辑加、逻辑异或;两种逻辑操作:逻辑左移、逻辑右移。这些逻辑运算和逻辑操作都是一种位操作,借助这些逻辑工具可以完成各种运算及处理。

5.1 逻 辑 乘

5.1.1 直接逻辑乘指令

(1) 指令形式

标号	操作码	操作数
[标号名]	AND	GR, ADR

(2) 指令功能

本指令可将寄存器 GR 中的内容与 ADR 单元中的内容按位进行逻辑乘,逻辑乘的结果又保留在寄存器 GR 中。用“∧”表示逻辑乘的运算符号,逻辑乘的法则为

0∧0 = 0

0∧1 = 0

1∧0 = 0

1∧1 = 1

不难看出,只有当逻辑乘运算符两端同时为 1 时,其结果才为 1。因此,逻辑乘的法则可简单地概括为“两端同时为 1 结果为 1,否则为 0”。在多位数进行逻辑乘时,也是运用该

法则进行对位相乘。例如,

$$\begin{array}{rl} & 10110111 \\ \wedge & 11000011 \\ \hline & 10000011 \end{array}$$

指令中的标号名,当本指令不被其他指令调用时可以省略。AND 是本指令的功能代码,GR 为操作寄存器,可在 GR0～GR4 中任选一个使用,ADR 是使用者设定的内存单元地址,直接用标号地址给出。

本指令的执行结果为正、零、负时,标志寄存器将产生相应的状态。

(3) 指令用法

将 GR1 中的值与单元 DY 中的值进行逻辑乘

```
①   LJC     AND     GR1, DY   ;(GR1)∧(DY)→GR1
    …       …
    DY      DC      #F0F0
    …       …
```

指令①为直接逻辑乘,所谓直接逻辑乘是指内存单元地址 E = ADR 直接由 DY 给出。在计算机中,逻辑乘如图 5.1 所示。

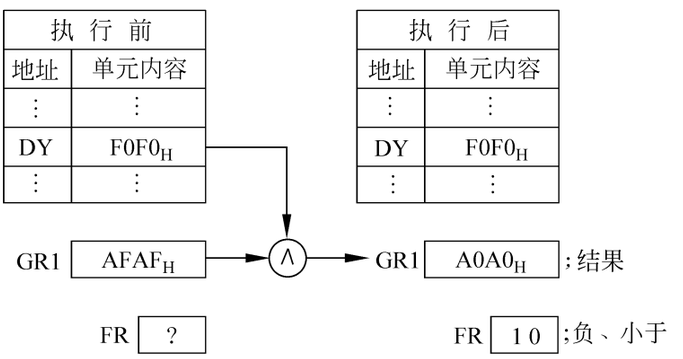


图 5.1 直接逻辑乘运算

GR1 的内容与 DY 中的内容进行逻辑乘时,是两个 16 位数对位进行逻辑乘:

$$\begin{array}{rl} (GR1): & 1010111110101111 = AFAF_H \\ \wedge & \\ (DY): & 1111000011110000 = F0F0_H \\ \hline \text{结果}: & 1010000010100000 = A0A0_H \rightarrow GR1 \end{array}$$

由于结果的最高位为 1,所以将该结果视为负数,从而使标志寄存器 FR 的内容变为 10,以表示负或小于。

截取 GR2 中内容的第 9 位到第 15 位,即低 7 位

```
②   JQZL    AND     GR2,JD7   ;(GR2)∧(JD7)→GR2
            …
    JD7     DC      #007F
            …
```

执行指令②,不管寄存器 GR2 中是什么值都将被截取低 7 位。由于操作结果仍然存放在 GR2 中,所以指令执行后 GR2 中保留原来的低 7 位,其余 9 位被置换为 0。由于结果的高位为零,所以标志寄存器的状态为正或大于。只有当 GR2 在操作前为全 0 时,操作后才会为全 0,从而使得标志寄存器状态为零。下面是截取实例。

【例 1】

【例 2】

由以上二例可以看到,不管被截取的内容是何值,均可以实现预定的截取结果。这是为什么呢?首先是运用了逻辑乘的法则,其次是设置了合适的常数。大家已经知道,逻辑乘是一种位运算,所谓位运算是一种按位进行的操作,它不涉及进位和借位的问题。在逻辑乘法则中规定 0∧1 = 0, |∧| = 1,从这个位操作中不难发现"1 有复制功能",如:

将 0 和 1 组合为 01,使其与 11 进制逻辑乘,也会得出复制结果,

在多位数的情况下,无论是什么数,只要用位数相同的全 1 常数与之进行逻辑乘,均能得到复制的结果。例如:

```
      5        6        7        8
   0101     0110     0111     1000   ;压缩在一个字中的 BCD 码。
∧  0000     0000     0000     1111   ;截取 8,对准 8 设全 1 常数。
-------------------------------------
   0000     0000     0000     1000
```

```
  0101  0110  0111  1000
∧ 0000  0000  1111  0000 ;截取7,对准7设全1常数。
  0000  0000  0111  0000

  0101  0110  0111  1000
∧ 0000  1111  0000  0000 ;截取6,对准6设全1常数。
  0000  0110  0000  0000

  0101  0110  0111  1000
∧ 1111  0000  0000  0000 ;截取5,对准5设全1常数。
  0101  0000  0000  0000
```

我们把运用逻辑乘全1常数进行截取的操作称为逻辑尺,使用逻辑尺可以方便、灵活地对数据进行截取、分割及变换等。

5.1.2 间接逻辑乘指令

(1) 指令形式

标号	操作码	操作数
[标号名]	AND	GR,ADR,XR

(2) 指令功能

执行本指令,可将寄存器GR中的内容与单元E中的内容进行按位逻辑乘,逻辑乘所得结果存入GR中,且结果的正、零、负属性会使标志寄存器FR产生相应的状态。其中单元E是由ADR与XR的内容之和得出的,即单元地址E = ADR + (XR),而XR为变址寄存器。间接逻辑乘与直接逻辑乘的惟一不同是在单元地址的指定上,直接操作单元地址由ADR直接给出,间接操作由ADR和XR的内容之和间接给出。

(3) 指令用法

将寄存器GR2中的值与单元DY + 2中的值进行逻辑乘

```
①  LJJC    AND    GR2,DY,GR1  ;(GR2)∧(DY + (GR1))→GR2
    …      …
   DY      DS     3
    …      …
```

指令①为间接逻辑乘指令,该指令执行前,假定GR1中为2。执行指令①将把GR2中的内容与DY + (GR1) = DY + 2中的值进行逻辑乘,逻辑乘的结果存于GR2中,该结果会使标志寄存器作相应的变化。逻辑乘示意图如图5.2所示。

由示意图可以看出,GR2的内容为第一个操作数,第二个操作数需由DY + (GR1)形成操作地址,其地址为DY + 2,取其内容为0031_H。GR2中的$001F_H$与0031_H进行按位逻辑乘运算。

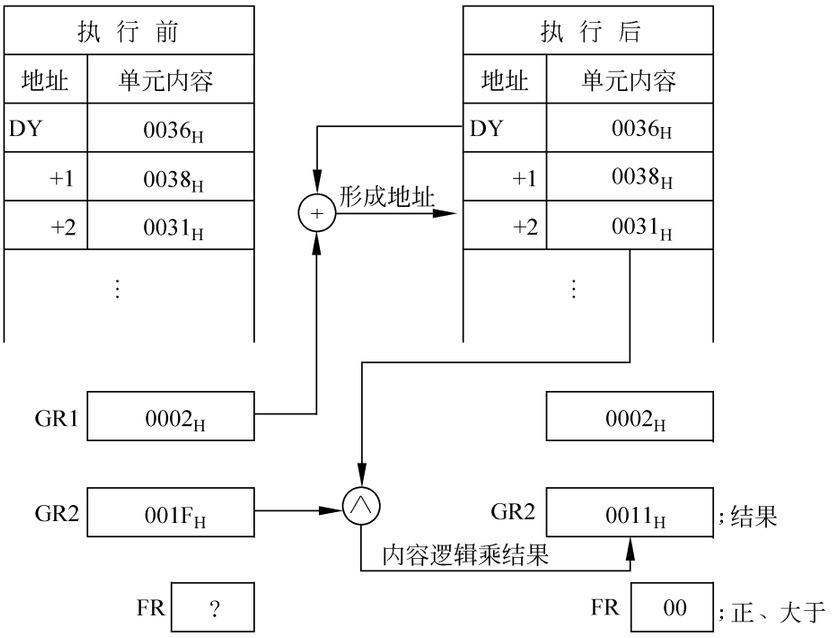


图 5.2　间接逻辑乘运算

(GR2):　0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 = 001F$_H$

(DY + 2):　0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 = 0031$_H$

结果:　0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 = 0011$_H$ →GR2

逻辑乘结果为 0011$_H$。由于该结果为正,所以 FR 被置为相应状态。

5.2　逻　辑　加

5.2.1　直接逻辑加指令

(1) 指令形式

标号	操作码	操作数
[标号名]	OR	GR, ADR

(2) 指令功能

本指令可将寄存器 GR 中的内容与 ADR 单元中的内容按位进行逻辑加,逻辑加的结果又保留在寄存器 GR 中。用“∨”表示逻辑加的运算符号,逻辑加的法则为

$$0 \vee 0 = 0$$
$$0 \vee 1 = 1$$
$$1 \vee 0 = 1$$
$$1 \vee 1 = 1$$

不难看出,在逻辑加运算中,只要在运算符的两端有 1 出现,则运算结果为 1,否则为零。因此,逻辑加运算可以概括为“有 1 为 1,否则为 0”的运算口诀。在多位运算中,也是运用该法则进行对位相加。例如,

$$\begin{array}{r} 10110101 \\ \vee\ 01001010 \\ \hline 11111111 \end{array}$$

指令中的标号名,当本指令不被其他指令调用时可以省略。OR 是本指令的功能代码,GR 为操作寄存器,可在 GR0～GR4 中任选一个使用,ADR 是使用者设定的内存单元地址,直接用标号地址给出。

本指令的执行结果为正、零、负时,将使标志寄存器产生相应的状态。

(3) 指令用法

将寄存器 GR2 中的值与单元 DY 中的值进行逻辑加

```
①   LJJ     OR      GR2, DY     ;(GR2)∨(DY)→GR2
    …       …
    DY      DC      #00FF
    …       …
```

指令①为直接逻辑加指令,所谓直接逻辑加是指内存单元地址直接由 DY 给出。这样,逻辑加操作便是 GR2 中的值与单元 DY 中的值对位进行逻辑加,其逻辑加示意图如 5.3 所示。

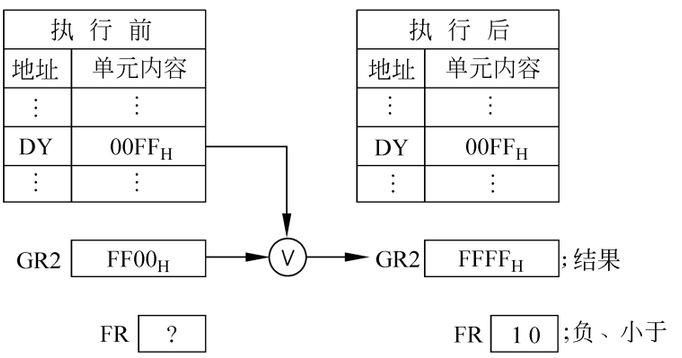


图 5.3 直接逻辑加运算

由图 5.3 可以看出,GR2 中值与 DY 中的值逻辑加的结果又保留在 GR2 中。由于逻辑加的结果是个负值,所以 FR 中被置为 10。在计算机中,两个操作数进行逻辑加是全字长 16 位对位进行操作,

$$\begin{array}{rl} (GR2): & 1111111100000000 = FF00_H \\ (CS): \vee & 0000000011111111 = 00FF_H \\ \hline 结果: & 1111111111111111 = FFFF_H \rightarrow GR2 \end{array}$$

本例是高 8 位 1 与低 8 位 1 进行逻辑加,其结果得到 16 位的全 1。由这个操作结果可以看出,逻辑加的特点体现了一种拼位操作,可以利用逻辑加的拼位作用进行数值信息向非数值信息的转换。

将寄存器 GR3 中的数值 3 转换成字符'3'

```
②   SZFS    OR      GR3, CS     ;(GR3)∨(CS)→GR3
    …       …
    CS      DC      #0030
    …       …
```

假定指令②在执行前 GR3 中为数值 3,执行指令②两个操作数将对位进行下列的逻辑加:

(GR3): 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
(CS): ∨ 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
结果: 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1

取逻辑加操作结果的低 7 位:0110011,正好是字符'3'的 ASCII,这说明在计算机中,3 已变成了'3'。而由 3 变为'3'是借助逻辑加在数值 3 的前面拼上了一个常数 0030_H。由于 0030_H 相当于十进制的 48,所以 3 转换为'3'时也相当于将 3 加上一个常数 48,即 $3 + 48 = 51_{10} = 0110011_2$。因此,在信息转换中,$0030_H$ 或 48_{10} 是经常使用的常数。

5.2.2 间接逻辑加指令

(1) 指令形式

标号	操作码	操作数
[标号名]	OR	GR,ADR,XR

(2) 指令功能

执行本指令,可将寄存器 GR 中的内容与间接地址 E 中的内容进行逻辑加,并将逻辑加的结果又存到寄存器 GR 中,而结果的正、零、负属性可使标志寄存器 FR 产生相应的状态。其中 E 是由 ADR 与 XR 的内容之和得出的,这是间接逻辑加与直接逻辑加惟一的不同。指令中的 XR 为变址寄存器,可在 GR1～GR4 中任选一个使用,变址寄存器不能使用 GR0。

(3) 指令用法

将寄存器 GR4 中的内容与单元 DY + 2 中的进行逻辑加

```
①   LJJJ    OR    GR4,DY,GR3    ;(GR4)∨(DY + (GR3))→GR4
    …       …
    DY      DS    3
    …       …
```

指令①为间接逻辑加指令,假定该指令执行前 GR3 中为 2。执行指令①将把 GR4 中的内容与 DY + (GR3) = DY + 2 中的内容进行逻辑加,逻辑加的结果存入 GR4 中,该结果会使标志寄存器 FR 作相应的变化。逻辑加示意图如图 5.4 所示。

由示意图可以看出,GR4 的内容为第一个操作数,第二操作数先由 DY + (GR3) = DY + 2 形成操作数的地址,然后取其内容与第一个操作数进行逻辑加,且为 16 位的对位逻辑加,

(GR4): 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 = $003F_H$
(DY + 2): ∨ 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 = 0641_H
结果: 0 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 = $067F_H$

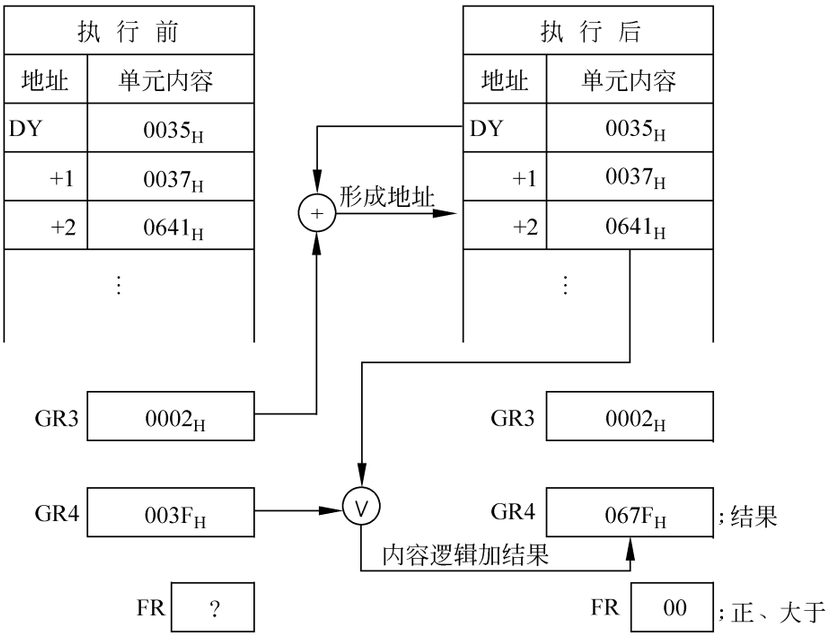


图 5.4　间接逻辑加运算

由于逻辑加结果 $067F_H$ 为正值,所以标志寄存器 FR 被置为 00,表示正、大于。

5.3　逻辑异或

5.3.1　直接逻辑异或指令

(1) 指令形式

标号	操作码	操作数
[标号名]	EOR	GR, ADR

(2) 指令功能

本指令可将寄存器 GR 中的内容与 ADR 单元中的内容按位进行逻辑异或运算,其结果保留在 GR 中。用 " 表示逻辑异或运算符,逻辑异或的法则为:

0 " 0 = 0

0 " 1 = 1

1 " 0 = 1

1 " 1 = 1

不难看出,只有当 0 和 1 或者 1 和 0 进行异或操作时,其结果为 1。因此,我们可以把异或运算法则归结为“相异为 1,否则为 0”。在多位数进行逻辑异或时,也是遵循该法则。例如:

$$\begin{array}{r} 10101100 \\ \text{"}\quad 01011100 \\ \hline 11110000 \end{array}$$

指令中的标号名,当本指令不被其他指令调用时可以省略。EOR 是本指令的功能代

码,GR 是操作寄存器,可在 GR0～GR4 中任选一个使用,ADR 是使用者设定的内存单元地址,用标号地址给出。

本指令的执行结果为正、零、负,均会使标志寄存器产生相应的状态。

(3) 指令用法

将寄存器 GR1 中的内容与单元 DY 中的内容进行逻辑异或

```
①  LJYH    EOR    GR1, DY   ;(GR1) " (DY)→GR1
   …       …
   DY      DC     #1234
   …       …
```

指令①为直接逻辑异或指令,所谓直接逻辑异或是指指令中的内存单元地址直接由标号地址 DY 给出。在计算机中逻辑异或如图 5.5 所示。

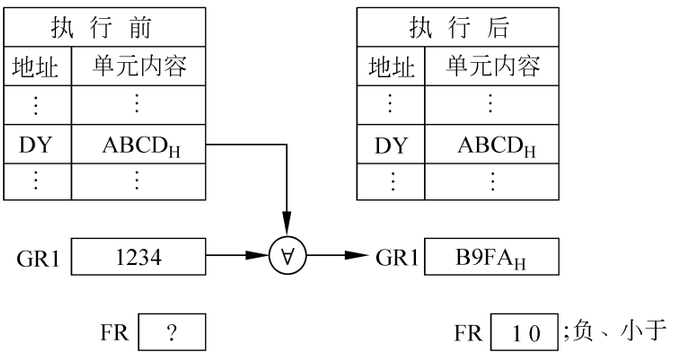


图 5.5　直接逻辑异或运算

GR1 中的内容与 DY 中的内容进行逻辑异或时,是两个 16 位数对位进行逻辑异或:

```
(GR1):  "  1 0 1 0 1 0 1 1 1 1 0 0 1 1 1 0 =  ABCD_H
(DY):      0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 =  1234_H
结果:      1 0 1 1 1 0 0 1 1 1 1 1 1 0 1 0 =  B9FA_H
```

由于结果的最高位为 1,所以将其视为负数,从而使标志寄存器 FR 置为 10,以表示负或小于。

将寄存器 GR2 中的内容逐位进行反转,即 0 变 1,1 变 0。

```
②  YHFZ    EOR    GR2,FZCS   ;(GR2) " (FZCS)→GR2
   …       …
   FZCS    DC     #FFFF
   …       …
```

指令②执行前,假定 GR2 中的内容 80F3$_H$,执行指令②其结果如下:

```
(GR2):  "  1 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 = 80F3_H
(FZCS):    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 = FFFF_H
结果:      0 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 = 7F0C_H
```

由异或结果不难看出,GR2 中的每一位都被反转了过来。所以能够实现反转,一是借助异或运算,二是选择适当的异或常数。为此,见以下实例分析。

【例 3】　将 00000000 反转。

【例 4】　将 11110000 反转。

【例 5】　将 10101010 反转。

由以上例分析可以得出,借助异或实现反转操作,不论被反转的内容是何值,只要选择位数相同的全 1 常数与其进行异或就可以实现反转。

将寄存器 GR3 中的负数原码变成反码

```
③   YMBF    EOR    GR3,BFCS   ;(GR3) " (BFCS)→GR3
      …       …
    BFCS    DC     #7FFF
      …       …
```

指令③执行前,不论 GR3 中是何值,只要是负数的原码均会随着指令③的执行变为反码。假定 GR3 中为 - 1 的原码,其转换如下:

```
(GR3):   1000000000000001;  - 1 的原码
(BFCS):  0111111111111111;  求反常数
结果:    1111111111111111;  - 1 的反码
```

不难发现,利用逻辑异或由负数的原码求其反码所使用的常数是高位为 0 而其他位为 1 的逻辑常数。

5.3.2　间接逻辑异或指令

(1) 指令形式

标号	操作码	操作数
[标号名]	EOR	GR,ADR,XR

(2) 指令功能

执行本指令可将寄存器 GR 中的内容与单元 E 中的内容进行按位逻辑异或，而逻辑异或所得到的结果又存入寄存器 GR 中。其中，单元 E 不是由 ADR 直接给出，而是由 ADR 与 XR 的内容间接给出，即 E = ADR + (XR)，这是间接异或指令与直接异或指令惟一的不同。间接异或指令的执行结果为正、零、负也会使标志寄存器 FR 产生相应的状态。

(3) 指令用法

将寄存器 GR2 中的内容与单元 DY + 2 中的内容逻辑异或

```
①  JJYE    EOR    GR2,DY,GR1   ;(GR2) " (DY + (GR1))→GR2
   …       …
   DY      DS     3
   …       …
```

指令①为间接逻辑异或指令，假定该指令执行之前，变址寄存器 GR1 中的值为 2，则间接地址 E = ADR + (XR) = DY + (GR1) = DY + 2。执行指令①，则(GR2)与(DY + 2)进行逻辑异或，其结果存于 GR2 中，并影响标志寄存器 FR，见图 5.6。

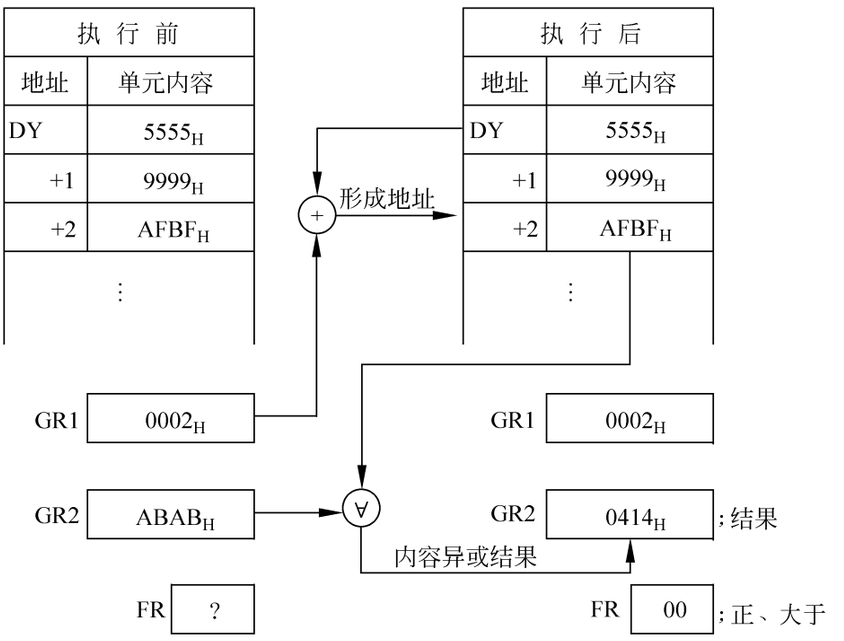


图 5.6　间接逻辑异或运算

由示意图可以看出，GR2 中的内容为第一个操作数，第二个操作数需由 DY 加上 GR1 中的内容形成操作数的地址，然后取其内容与第一个操作数相异或，如：

(GR2): " 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 1 = $ABAB_H$

(DY + 2): 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 = $AFBF_H$

结果: 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 = 0414_H

由于异或运算的最高位为 0，视其为正，所以标志寄存器置为 00 表示正或大于。

5.4 逻辑左移操作

5.4.1 直接逻辑左移指令

(1) 指令形式

标号	操作码	操作数
[标号名]	SLL	GR, ADR

(2) 指令功能

执行本指令,可将寄存器 GR 中的内容视为无符号数依次左移由 ADR 所指定的位数,左移出现的空位自动补上 0,左移结果又存于 GR 中,而左移后的结果属性为正、零、负均会使标志寄存器 FR 产生相应的变化。

指令中的标号名,当本指令不被其他指令调用时可以省略。SLL 为本指令的功能代码,它是 SHIFT LEFT LOGICAL 的缩写。GR 为操作寄存器,可在 GR0～GR4 中任选一个使用。ADR 为左移位数指定,其值为大于零的正整数。

(3) 指令用法

将寄存器 GR1 中的内容逻辑移左移 6 位

```
①   LJZY    SLL   GR1, 6
```

执行指令①可将寄存器 GR1 中的内容视为无符号的逻辑数,依次右移 6 位,左移出现的空位自动补 0,左移后的结果仍保留在 GR1,但左移结果会直接影响标志寄存器 FR 的内容,下面是逻辑左移实例。

【例 6】

由于是逻辑左移,所以在移位时连同符号位都移动。而移位后判别数的属性正、零、负仍然作为数值性数据。在例 6 中,左移 6 位后的最高位为 1,故视为负值,故使 FR 置为 10。

【例 7】

在例 7 中,左移前最高位为 1,左移后最高位为 0,故视为正值,标志寄存器 FR 置为 00 表示正或大于。

【例 8】

在例 8 中,左移前、后均为正值,这种情况的逻辑左移与算术左移在结果上是一致的,因此,在特定的情况下使用逻辑左移也能够完成乘法运算。在本例中,逻辑左移前的值为 1,即 2^0,左移 6 位后变为 $2^6 = 64$,所完成的运算是乘以 2^6。

5.4.2　间接逻辑左移指令

(1) 指令形式

标号	操作码	操作数
[标号名]	SLL	GR, ADR, XR

(2) 指令功能

执行本指令可将寄存器 GR 中的值连同符号位一起左移,而左移位数 E = ADR + (XR),也就是说左移位数不是由 ADR 直接给出,而是由 ADR 加上 XR 的内容来确定。左移后出现的空位自动补 0,左移后的结果仍然保留在寄存器 GR 中。左移结果为正、零、负直接影响标志寄存器 FR 的状态。

指令中的标号名,当本指令不被其他指令调用时可以省略。SLL 为本指令的功能代码,GR 为操作寄存器,可在 GR0～GR4 中任选一个使用。ADR 为左移常数,XR 为变址寄存器,XR 中的内容与 ADR 之和作为移位位数。

(3) 指令用法

将寄存器 GR2 中的内容逻辑左移 4 位

① `LZY1 SLL GR2, 4, GR1` ;(GR2)/4→GR2

指令①在执行前,假定变址寄存器 GR1 中的值为 0,则左移位数 E = ADR + (XR) = 4 + (GR1) = 4 + 0 = 4。

② `LZY2 SLL GR2, 0, GR1` ;(GR2)/4→GR2

指令②在执行前,假定变址寄存器 GR1 中的内容为 4,则左移位数 E = ADR + (XR) = 0 + (GR1) = 0 + 4 = 4。

③ `LZY3 SLL GR2, 2, GR1` ;(GR2)/4→GR2

指令③在执行前,假定变址寄存器 GR1 中的内容为 2,则左移位数 E = ADR + (XR) = 2 + (GR1) = 2 + 2 = 4。

由指令①、②、③可以得出,使用间接左移指令实现逻辑左移 4 位有多种写法,其核心是匹配 ADR 和(XR)。在间接逻辑左移中,ADR 可以看成一个初始基数,变址寄存器中的内容为变数,在基数恒定的情况下,调整变数,即调整变址寄存器的内容就可以实现所需要的左移。

5.5 逻辑右移操作

5.5.1 直接逻辑右移指令

(1) 指令形式

标号	操作码	操作数
[标号名]	SRL	GR, ADR

(2) 指令功能

逻辑右移是一种连同符号位一起移动的右移,直接逻辑右移是指直接给出右移位数的右移。

执行本指令,可将寄存器 GR 中的内容视为无符号数依次右移由 ADR 所设定的移位位数,右移出现的空位自动补上。右移结果又存于 GR 中,而结果属性为正、零、负均会使状态寄存器 FR 产生相应的状态。

指令中的标号名,当本指令不被其他指令调用时可以省略。SRL 为本指令的功能代码,它是 SHIFT RIGHT LOGICAL 的缩写。GR 为操作寄存器,可在 GR0～GR4 任选一个使用。ADR 为右移位数指定,其值为大于或等于 1 的正整数。如果右移位数设定为 0,则不产生右移;如果右移位数为负值,则为错误指令。

(3) 指令用法

将寄存器 GR1 的内容逻辑右移 4 位。

① `LJYY SRL GR1, 4` ;(GR1)4\→GR1

执行指令①,将寄存器 GR1 的内容连同符号位一起依次右移 4 位,右移出现的空位自动补 0,右移结果仍保留在 GR1 中,且该结果会影响标志寄存器 FR 的状态。下面是逻辑右

移的实例分析。

【例 9】

在例 9 中,GR1 在移位前为 9876H,如将其看成数值性数的话;它是个负数,逻辑右移 4 位后结果为 0987H,它是个正值,所以标志寄存器置为 00,表示正、大于。

【例 10】

在例 10 中,GR1 右移前内容为 0010H,右移 4 位后结果为 0001H。由此看出,对于一个正数来说,逻辑右移也有除法的运算功能。

5.5.2　间接逻辑右移指令

(1) 指令形式

标号	操作码	操作数
[标号名]	SRL	GR,ADR,XR

(2) 指令功能

执行本指令可将寄存器 GR 中的内容连同符号一起右移,而右移位数由 E = ADR + (XR)间接给出。右移后出现的空位自动补 0,右移的结果仍保留在寄存器 GR 中,右移结果为正、零、负均会使标志寄存器产生相应的状态。

指令中的标号名,当本指令不被其他指令调用时可以省略。SRL 为本指令的功能代码,GR 为操作寄存器,可在 GR0～GR4 中任选一个使用。ADR 为移位常数,XR 为变址寄存器。XR 中的内容与 ADR 之和作为逻辑右移的位数。

(3) 指令用法

将寄存器 GR2 的内容逻辑右移 4 位

① `JLYY1  SRL  GR2, 4, GR1` ;(GR2)4\→GR2

指令①在执行前,假定变址寄存器 GR1 的内容为 0,则逻辑右移位数 E = ADR + (XR) = 4 + (GR1) = 4 + 0 = 4。

② `JLYY2  SRL  GR2, 0, GR1` ;(GR2)4\→GR2

指令②在执行前,假定变址寄存器 GR1 中的值为 4,则逻辑右移位数 E = ADR + (XR) = 0 + (GR1) = 0 + 4 = 4。

由①、②可以看出灵活运用 ADR 和(XR)的匹配可以组合出多种移位指令,从而实现同一移位的目的。但需要指出的是,变址寄存器 XR 在间接移位指令中主要是利用它的内容不断变化而实现成组移位操作。

5.6 程序设计训练

【问题 5-1】 将压缩在 YSZF 单元中的字符数据'5'、'6'分解后并转换成数值数据 5、6,分别存于单元 A、B 中。

【方法分析】 字符数据在计算机中是字符的 ASCII,字符'5'、'6'在一个 16 位的单元中,'5'的ASCII 在高 8 位,'6'的 ASCII 在低 8 位,而实际的 ASC 码只需 7 位,压缩形式为:

0 0 1 1 0 1 0 1	0 0 1 1 0 1 1 0
'5'	'6'

使用逻辑乘可以方便地将其分开,实际上用适当的逻辑常数,不但可以将其分开,还可以一次转换成数值性数据。例如 $0F00_H$ 与压缩数据进行逻辑乘,可以得到数值 5;用 000F 与压缩数进行逻辑乘,可以得到数值 6。

【参考程序】

```
FJZH    START
        LD      GR2,YSZF    ;取压缩数→GR2
        AND     GR2,C1      ;利用逻辑尺取 5
        ST      GR2,A       ;将数值 5→A
        LD      GR2,YSZF    ;取压缩数→GR2
        AND     GR2,C2      ;利用逻辑尺取 6
        ST      GR2,B       ;将数值 6→B
        EXIT
YSZF    DC      #3536       ;压缩在一起的'5'、'6'
C1      DC      #0F00       ;取 5 的逻辑常数
C2      DC      #000F       ;取 6 的逻辑常数
A       DS      1
B       DS      1
        END
```

【结果评述】 执行本程序,不仅可以将压缩在一个单元中的字符数据分开,而且可以同时

转换为数值数据。本程序采用的是逻辑尺,也可以使用逻辑左、右的方法实现。

【问题 5-2】　将数值 7、8 转换为'7'、'8'并压缩在单元 X 中。

【方法分析】　数值 7、8 与字符'7'、'8'在计算机中的代码只是高 3 位不同,二者比较如下:

7:	0000000000000111		8:	0000000000001000	
'7':	0000000000	0110111	'8':	0000000000	0110000
		ASCII			ASCII

将 7 转换为'7',将 8 转换为'8'都只需拼上高 3 位即可。然后'7'和'8'的 ASCII 再拼在一个单元中即达到要求,如下所列:

0	0 1 1 0 1 1 1	0	0 1 1 1 0 0 0
'7'		'8'	

了解了上述内在关系,转换及压缩合并是不难做到的。

【参考程序】

```
ZHYS    START
        LD      GR1,C8      ;取 8→GR1
        OR      GR1,G3      ;拼上高 3 位
        ST      GR1,X       ;'8'→X 单元
        LD      GR1,C7      ;取 7→GR1
        OR      GR1,G3      ;拼上高 3 位,由 7 转为'7'
        SLL     GR1,8       ;'7'移到高 8 位
        OR      GR1,X       ;'7'、'8'压缩在一起
        ST      GR1,X       ;结果存于单元 X 中
        EXIT
C7      DC      7
C8      DC      8
G3      DC      #0030
X       DS      1
        END
```

【结果评述】　在本程序中,数值数据转为非数值数据以及数据的压缩均采取逻辑加进行拼位的办法。由于 7、8 及'7'、'8'在计算机中的代码以数值而论均为正数,所以变换和压缩也可以用移位和加法来实现。

【问题 5-3】　将压缩在 YSS 中的 5、6、7、8 分解为 4 个独立的数,存放在以 FJ 为首地址的 4 个单元中。

【方法分析】　根据问题的要求,本程序所要完成的任务如下图所示:

借助逻辑尺可以将压缩在 YSS 中的 4 个数据从低位开始截取,截取一个向 FJ 单元保存一个。截取第二个时,先将数据右移,使被截取的数据位于低 4 位,借助逻辑尺截取低 4 位。存放独立的数据时利用间接存数指令 ADR 设定为 FJ,变址寄存器初始状态为 0,则第一个独立数据的地址 E = ADR + (XR) = FJ + 0 = FJ。以后通过修改变址寄存器,即变址寄存器加 1 形成第 2 个独立数据的地址。依次类推,形成所有独立数据的存储地址 FJ + 1、FJ + 2、FJ + 3。

【参考程序】

```
FJCF        START
            LEA     GR2,0           ;变址寄存器初始值0→GR2
        ①   LD      GR1,YSS         ;取压缩数→GR1
    ⑤       AND     GR1,CF          ;取低4位,分解出第1个独立数
            ST      GR1,FJ,GR2      ;存第1个独立数
            LEA     GR2,1,GR2       ;变址寄存器加1
        ②   LD      GR1,YSS         ;取压缩数
            SRL     GR1,4           ;逻辑右移4位,第2个数到低位
    ⑥       AND     GR1,CF          ;取第2个独立数
            ST      GR1,FJ,GR2      ;存第2个独立数
            LEA     GR2,1,GR2       ;变址寄存器加1
        ③   LD      GR1,YSS         ;取压缩数
            SRL     GR1,8           ;逻辑右移8位,第3个数到低位
    ⑦       AND     GR1,CF          ;取第3个独立数
            ST      GR1,FJ,GR2      ;存第3个独立数
            LEA     GR2,1,GR2       ;变址寄存器加1
        ④   LD      GR1,YSS         ;取压缩数
            SRL     GR1,12          ;逻辑右移12位,第4个数到低位
    ⑧       AND     GR1,CF          ;取第4个独立数
            ST      GR1,FJ,GR2      ;存第4个独立数
            EXIT
YSS         DC      #5678           ;压缩在同一单元的5、6、7、8
CF          DC      15              ;取低4位的逻辑常数
FJ          DS      4               ;定义以FJ为首地址的4个单元
            END
```

【结果评述】　执行本程序可以完成将压缩在同一个单元的 4 个数的分解以及 4 个独立数的依次保存。由于现在还没有介绍循环及成组操作,所以程序为顺序结构。因此,程序中出现了大量的重复指令,例如指令①、②、③、④是完全一样的,指令⑤、⑥、⑦、⑧也是重复的,类似问题程序中还有。这说明本程序可以精炼和压缩,解决的办法是使用循环操作。

【问题 5-4】　已知单元 X 中为自然数,如果它是偶数,则在单元 Y 中存入 0;如果它为奇数,则在单元 Y 中存入 1。

【方法分析】　无论是奇数还是偶数,它们在计算机中都是二进制的整数。判断一个自然数是奇还是偶,只需判断其最低位是零还是为 1。如果最低位为 1,则为奇数;如果最低位为 0,则为偶数。到目前为止,虽然还没有介绍比较、判断指令,但可以借助逻辑尺取出 X 单元的最低位,不管它是 0 还是 1,将其放入 Y 单元就满足了问题的要求。因为自然数的最后一位,恰恰是奇、偶数的表征,不需要比较、判断。

【参考程序】

```
BJO     START
        LD      GR2,X       ;取 X 单元中的自然数→GR2
        AND     GR2,C       ;取其最低位→GR2
        ST      GR2,Y       ;最低位→Y,奇则 1→y,偶则 0→Y
        EXIT
X       DS      1
Y       DS      1
C       DC      #0001       ;取最低位的逻辑常数
        END
```

【结果评述】　本程序利用偶数二进制最低位必为 0,奇数最低位必为 1 的特征,免去了判断比较,就可以分出奇、偶数的要求。

【问题 5-5】　已知 FSYM 单元中为负数的原码,将其变为反码存放在 FSBM 单元中。

【方法分析】　负数的原码求其反码的方法是数的符号位不动,尾数逐位求反。借助异或一个固定常数的方法,可以方便地将一个负数的原码变成反码。见下列运算实例。

【例 1】　-1 的原码:　1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 = 8001_H

异或常数:　0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 = $7FFF_H$

-1 的反码:　1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 = $FFFE_H$

【例 2】　-2 的原码:　1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 = 8002_H

异或常数:　0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 = $7FFF_H$

-2 的反码:　1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 = $FFFD_H$

不难得出,一个负数的原码与固定常数 $7FFF_H$ 相异或,就可以将负数的原码变为对应的反码。

【参考程序】

```
YBFM    START
        LD      GR1,FSYM    ;取单元中的负数原码
```

```
        EOR     GR1,EHCS      ;异或固定常数变为反码
        ST      GR1,FSFM      ;反码存入给定单元
        EXIT
FSYM    DS      1             ;负数原码单元
FSFM    DS      1             ;负数反码单元
EHCS    DC      #7FFF         ;用于变反码的异或常数
        EXIT
        END
```

习 题 5

1. 阅读下列程序,逐条加上注释并分析执行结果。

(1)
```
TEST1   START
        LEA     GR1,16
        SLL     GR1,1
        LEA     GR1,16,GR1
        ADD     GR1,A
        ST      GR1,B
        EXIT
A       DC      6
B       DS      1
        END
```

(2)
```
TEST2   START
        LEA     GR2,32
        ST      GR2,C
        SRL     GR2,1
        OR      GR2,C
        OR      GR2,D
        ST      GR2,E
        EXIT
C       DS      1
D       DC      8
E       DS      1
        END
```

(3)
```
TEST3   START
        LEA     GR3,1
        SLL     GR3,3
        ST      GR3,M
        SLL     GR3,6
        OR      GR3,M
        ST      GR3,M
        SLL     GR3,3
```

```
        OR      GR3,M
        ST      GR3,M
        EXIT
M       DS      1
        END
```

2. **根据题目要求,填空完善下列程序。**

(1) **按计算式**(4×MS + MS)×2 **计算** MS **的** 10 **倍。**

```
TK1     START
        LD      GR0,MS
        [          ];MS×4
        ADD     GR0,[    ]
        [          ]
        EXIT
MS      DC      6
        END
```

(2) **按计算式** 2×MS + 8×MS **计算** MS **的** 10 **倍。**

```
TK2     START
        LD      GR0,MS
        [          ];MS×2
        ST      GR0,DY
        [          ]
        [          ]
        EXIT
MS      DC      6
DY      DS      1
        END
```

(3) **将文字字符** 'A'、'B' **压缩在** YS **单元中。**

```
TK3     START
        LD      GR1,W1
        [      ]  GR1,[      ]
        OR      GR1,[      ]
        ST      GR1,YS
        EXIT
W1      DC      'A'
W2      DC      'B'
YS      DS      1
        END
```

(4) **将单元** YS **中的数高** 8 **位和低** 8 **位分别存放于** AS **和** BS **的低** 8 **位中。**

```
TK4     START
        LD      GR1,YS
        [      ] GR1,[      ]
        ST      GR1,AS
        LD      GR1,[      ]
        [      ] GR1,8
        [      ] GR1,[      ]
        ST      GR1,BS
        EXIT
YS      DC      #4142
AS      DS      1
BS      DS      1
        END
```

3. **程序设计。**

(1) **设计一个程序,将单元** DY **中的高** 8 **位与低** 8 **位互换。**

(2) **设计一个程序,将压缩在** YS **单元中的** 1234,**拆分为** 13 **和** 24,**并分别存入单元** A **和** B **中。**

(3) **已知** FYM **单元中为负数的原码,设计一个求其补码的程序,并将补码存于** FBM。

(4) **设计一个程序按**(6A + 9A)×2 **求** A **的** 30 **倍。**

第 6 章

比较与转移

在程序设计中,不论是实现计算问题还是处理问题都离不开比较。为此,在 CASL 中设置了两种比较指令,一种是算术比较,另一种是逻辑比较。所谓算术比较是把比较的双方都视为有符号的数而进行比较,逻辑比较则是把比较的双方都视为无符号数所进行的比较。在程序设计中,比较一般来说并不是程序设计的最终目的,而比较只是为了区分不同情况,以便采取不同的计算或处理方案,描述不同的计算或处理方案是由不同的程序段来实现的,因此当通过比较区分出不同情况后,必然要转移到对应该情况的计算或处理的程序段去实施所设定的计算或处理。因此,比较和转移就紧密地结合了起来。在 CASL 中设置了 5 种转移指令,这 5 种转移指令是无条件转移,大于、等于转移,小于转移,不等于转移和等于转移。下面将分别介绍比较及转移指令的功能及使用。

6.1 算术比较及逻辑比较

6.1.1 算术比较指令

(1) 指令形式

标号	操作码	操作数	备 注
[标号名]	CPA	GR,ADR	直接 E = ADR
[标号名]	CPA	GR,ADR,XR	间接 E = ADR + (XR)

(2) 指令功能

执行本指令,可把寄存器 GR 中的内容与单元 E 中的内容均视为有符号数的补码进行比较,并根据比较结果产生相应的标志位而赋予标志寄存器,作为转移的先决条件,且不改变被比较的内容。

在计算机中,比较的第一步是寄存器的内容减去单元中的内容,即(GR) - (E);第二步是根据相减结果分 3 种情况产生标志寄存器的内容,如:

当(GR) > (E)时,(FR) = 00,即为正

当(GR) = (E)时,(FR) = 01,即为零

当(GR) < (E)时,(FR) = 10,即为负

指令中的标号名表示本指令在程序中的位置,当本指令不被其他指令调用时,标号名

可省略。CPA 为本指令的功能代码,它是 COMPARE ARITHMETIC 的缩写。GR 为操作寄存器,可在 GR0～GR4 任选一个。内存单元地址用 E 表示,在直接算术比较指令中 E = ADR,而 ADR 用标号地址给出;在间接算术比较指令中 E = ADR + (XR),而 ADR 用标号地址或常数给出,XR 在 GR1～GR4 选用一个,XR 不能使用 GR0。

(3) 指令用法

寄存器 GR1 与单元 DY 中的内容进行算术比较:

```
①   SBJ     CPA     GR1, DY     ;(GR1)～(DY)
    …       …
    DY      DS      1
    …       …
```

指令①为直接算术比较指令,执行本指令将把 GR1 中的内容与单元 DY 中的内容相减后视其结果形成标志寄存器 FR 的状态,作为转移的先决条件,请见下列比较实例。

【例 1】 (GR1): 0006H (DY): 0005H

二者比较:

0006H - 0005H = 0001H 结果为正,则(FR): 0 0

【例 2】 (GR1): FFFFH (DY): FFFFH

二者比较:

FFFFH - FFFFH = (- 1) - (- 1) = 0 结果为 0,则(FR): 0 1

【例 3】 (GR1): 000AH (DY): 000BH

二者比较:

000AH - 000BH = 1010 - 1011 = - 1 结果为负,则(FR): 1 0

将寄存器 GR2 中的内容与单元 DY + 2 中的内容进行算术比较

```
        LEA     GR3,2           ;2→GR3
②   JSBJ    CPA     GR2, DY, GR3    ;(GR2)～(DY + 2)
    …       …
    DY      DS      3
    …       …
```

指令②为间接算术比较指令,指令②在执行前已在变址寄存器 GR3 中送入了 2,所以执行指令②时所进行的比较是 GR2 中的内容与 DY + 2 单元中内容的比较。

实现(GR2)与(DY + 2)的比较也可以使用下列指令

```
        LEA     GR3, DY         ; DY→GR3
③   JSBJ    CPA     GR2, 2, GR3     ; (GR2)～(DY + 2)
    …       …
    DY      DS      3
    …       …
```

指令③在执行前,变址寄存器 GR3 中已存入标号地址 DY,因此执行指令③所形成的内存单元地址为

$$\begin{aligned} E &= ADR + (XR) \\ &= 2 + (GR3) \\ &= 2 + DY \\ &= DY + 2 \end{aligned}$$

这样指令③所进行的比较为(GR2)与(DY + 2)的比较,它与指令②的作用是一样的。

6.1.2　逻辑比较指令

(1) 指令形式

标号	操作码	操作数	备　注
[标号名]	CPL	GR, ADR	直接 E = ADR
[标号名]	CPL	GR, ADR, XR	间接 E = ADR + (XR)

(2) 指令功能

执行本指令,可把寄存器 GR 中的内容与单元 E 中的内容均视为无符号数进行比较,并根据比较结果产生相应的标志位而赋予标志寄存器,作为转移的先决条件,且不改变被比较的内容。

在计算机中,比较的第一步是寄存器的内容减去单元中的内容,即(GR) - (E);第二步是根据相减结果为正、零、负来产生标志寄存器的内容,如:

当(GR) > (E)时,(FR) = 00,即为正

当(GR) = (E)时,(FR) = 01,即为零

当(GR) < (E)时,(FR) = 10,即为负

指令中的标号名表示本指令在程序中的位置,当本指令不被其他指令调用时,可省略标号名。CPL 为本指令的功能代码,它是 COMPARE LOGICAL 的缩写。GR 为操作寄存器,可在 GR0～GR4 中任选一个。内存单元地址用 E 表示,在直接逻辑比较指令中 E = ADR,而 ADR 用标号地址给出;在间接逻辑比较指令中 E = ADR + (XR),而 ADR 用标号地址或常数给出,XR 为变址寄存器,可在 GR1～GR4 中任选一个,但不能使用 GR0。

(3) 指令用法

寄存器 GR3 的内容与单元 DY 中的内容进行逻辑比较:

```
①   LBJ     CPL     GR3, DY    ;(GR3)～(DY)
    …       …
    DY      DS      1
    …       …
```

指令①为直接逻辑比较指令,执行本指令可把 GR3 中的内容与单元 DY 中的内容相减,并视其相减所得结果而形成标志寄存器 FR 的状态,作为转移的先决条件,见下列比较实例。

【例 4】 (GR3): FFFF$_H$ (DY): 7FFF$_H$

二者比较:

FFFF$_H$ - 7FFF$_H$ = 65535 - 32767 = 32768 结果为正,则(FR): 0 0

【例 5】 (GR3): 0011$_H$ (DY): 0022

二者比较:

0011 - 0022 = 17 - 34 = - 17 结果为负,则(FR): 1 0

将寄存器 GR3 中的内容与单元 DY + 1 的内容进行逻辑比较:

```
                LEA     GR4,1           ; 1→GR4
②   JLB     CPL     GR3, DY, GR4        ; (GR3)～(DY + 1)
    …       …
DY  DS      2
    …       …
```

指令②为间接逻辑比较指令,指令②在执行前已在变址寄存器 GR4 中存入了 1,所以在执行指令②所进行的比较是寄存器 GR3 中的内容与 DY + 1 单元中内容的比较。

实现(GR3)与(DY + 1)的比较也可以用下列形式的指令:

```
                LEA     GR4, DY         ; DY→GR4
③   JJLB    CPL     GR3, 1, GR4         ; (GR3)～(DY + 1)
    …       …
DY  DS      2
    …       …
```

指令③在执行前,已在变址寄存器中存入了标号地址 DY,因此在执行指令③所形成的内存单元地址为

$$\begin{aligned} E &= ADR + (XR) \\ &= 1 + (GR4) \\ &= 1 + DY \\ &= DY + 1 \end{aligned}$$

这样指令③所进行的比较为(GR3)与(DY + 1)的比较,它与指令②的作用是完全一样的。

6.2 无条件转移及条件转移

6.2.1 无条件转移指令

(1) 指令形式

标号	操作码	操作数	备 注
[标号名]	JMP	ADR	直接 E = ADR
[标号名]	JMP	ADR, XR	间接 E = ADR + (XR)

(2) 指令功能

执行本指令将无条件地按照使用者所设定的地址去执行。在通常情况下,指令的执行是按顺序逐条进行的,而控制执行顺序的是指令计数器 PC,也就是说 PC 记录着当前被执行指令的地址,执行完一条指令,PC 中的地址依次修改到下一条被执行的指令地址。由于 CASL 汇编语言指令长度占两个编址单元,所以每执行完一条指令修改为下一条指令时(PC) + 2→PC,而执行无条件转移指令时则为转移入口指令地址送 PC,即 E→PC。这一切都是自动完成,不需要使用者干预。

指令中的标号名表示本指令在程序中的位置,当本指令不被其他指令调用时,标号名可省略。JMP 为本指令的功能代码,它是 UNCONDITIONAL JUMP 的缩写。在直接无条件转移指令中 ADR 为转移地址,可设置为标号地址;在间接形式下,转移地址 E 由两项给出,ADR 为标号地址或常数,XR 为变址寄存器,可在 GR1～GR4 中任选一个使用。

无条件转移指令最终的目的是改变程序的顺序执行状况,跳过一至几条指令或一个程序段,按照使用者的要求无条件地转向所设定的地址,该指令与高级语言中的 GO TO 语句相类似。

(3) 指令用法

无条件转向 RUK

```
①   WTJ     JMP     RUK
```

执行本指令,将无条件地转向标号地址 RUK 去执行。指令①为直接无条件转移指令,转移示意如下:

由示意图可以看出,转移指令可以向下转移,也可以向上转移。

无条件地按变址寄存器 GR1 所设定的地址转移

```
            LEA     GR1, RUK    ; RUK→GR1
②   JWZ     JMP     0,GR1       ; 转向 RUK
```

指令②为间接无条件转移指令,由于在执行指令②之前已在变址寄存器 GR1 中传送了标号地址 RUK,所以转移地址为

$$\begin{aligned} E &= ADR + (XR) \\ &= 0 + (GR1) \\ &= 0 + RUK \end{aligned}$$

= RUK

指令②与指令①相比,二者在形式上虽然不同,但转移结果是完全一样的。

间接无条件转移指令,灵活设置 ADR 和(XR)还可以写出如下转移指令

③ WZ1 JMP,DY,GR2

转移地址为 E = DY + (GR2)

④ WZ2 JMP, - 1,GR3

转移地址为 E = (GR3) - 1

6.2.2 大于、等于(非负)转移指令

(1) 指令形式

标号	操作码	操作数	备 注
[标号名]	JPZ	ADR	直接 E = ADR
[标号名]	JPZ	ADR,XR	间接 E = ADR + (XR)

(2) 指令功能

执行本指令,当标志寄存器的状态(FR) = 00 或(FR) = 01 时,则按照所指定的地址 E 转移,否则执行下一条指令。

由于本指令以标志寄存器的状态作为转移的判据,所以凡是对标志寄存器产生影响的指令都可以作为本指令的先导指令:

① 以比较指令为先导指令时,当(GR)≥(E)时,则转移成立。

② 以计算型指令为先导指令时,当计算结果非负时,则转移成立。

③ 以传送指令为先导指令时,当寄存器 GR 中的值非负时,则转移成立。

指令中的标号名表示本指令在程序中的位置,如果本指令不被其他指令调用时,可省去标号名。JPZ 为本指令的功能代码,它是 JUMP ON PLUS OR ZERO 的缩写。转移地址 E,在直接形式下,转移地址为 ADR,而 ADR 用标号地址给出;在间接形式下,转移地址由 ADR 和(XR)共同设定。

(3) 指令用法

比较转移:当(GR1)≥(DY)转移,否则执行下一条

指令①借助比较的结果,作为转移条件。由于(GR1) > (DY),所以当执行指令①时,转移成立,转向标号地址 ZY。

运算转移:当计算结果为非负值时转移,否则执行下一条

指令②以减法的结果,作为转移条件。由于(GR1) - (DY) = 65 - 66 = - 1,结果为负值,因此不满足转移条件而不产生转移,继续执行下一条指令。

传送转移:当传送给操作寄存器的结果大于、等于零时转移,否则执行下一条指令。

在指令③之前既没有比较指令,也没有运算指令,其转移的先决条件是借助此前传送给寄存器 GR1 的值,由于(GR1) = 0,满足转移条件,因此执行指令③时,将转向 DDZ 去执行。但值得注意的是,寄存器的内容并不是在任何条件下都可以作为转移的先决条件,而仅仅限于与标志寄存器有关系的指令才可以利用寄存器的内容作为转移指令的判据。像取数指令 LD,存数指令 ST 均与标志寄存器无关,因此在这些指令中所形成的寄存器内容,不能作为条件转移指令的判据。从而在 LD、ST 指令之后设置条件转移指令是无任何意义的。

6.2.3　小于(负)转移指令

(1) 指令形式

标号	操作码	操作数	备　注
[标号名]	JMI	ADR	直接 E = ADR
[标号名]	JMI	ADR, XR	间接 E = ADR + (XR)

(2) 指令功能

执行本指令,当标志寄存器的状态(FR) = 10 时,则按照所指定的转移地址转移,否则执行下一条指令。

由于本指令以标志寄存器的状态作为转移的判据,所以凡是与标志寄存器状态有关系的指令都可以作为本指令的先导指令,如比较指令、计算型指令、传送指令等,而取数指令、存数指令均与 FR 的状态无关,所以它们不能作为本指令的先导指令。

指令中的标号名表示本指令在程序中的位置,如果本指令不被其他指令调用时,标号名可省略。JMI 为本指令的功能代码,它是 JUMP ON MINUS 的缩写。在直接指令形式下,转移地址 E = ADR,而 ADR 用标号地址给出;在间接指令形式下,转移地址 E = ADR + (XR)。

(3) 指令用法

比较转移:寄存器中的内容与单元中的内容进行比较,当(GR2) < (DY)时,则转移;否则执行下一条指令。

指令①以算术比较结果作为转移的先决条件,由于(GR2) < (DY),所以当执行指令①时满足转移条件而转移到 XZ 去执行。

运算转移:当计算结果为负时则发生转移,否则执行下一条指令。

指令②以减法指令作为先导指令,由于(GR2) - (CS) = 2 - 3 = - 1,结果为负,满足转移条件,执行指令②时,则转向 FZ 去执行。

传送转移:当传送给操作寄存器的结果为负值时则转移,否则执行下一条指令。

指令③在执行前已将 - 1 传到 GR2 中,执行指令③时,由于 GR2 中的值为负,所以转移到 WL 去执行。

6.2.4　不等于(非零)转移指令

(1) 指令形式

标号	操作码	操作数	备　注
[标号名]	JNZ	ADR	直接 E = ADR
[标号名]	JNZ	ADR, XR	间接 E = ADR + (XR)

(2) 指令功能

执行本指令,当标志寄存器的状态为(FR) = 00 或(FR) = 10 时,则按照所指定的地址 E 转移,否则执行下一条指令。

由于本指令以标志寄存器的状态作为转移的判据,所以凡是对标志寄存器产生影响的指令都可以作为本指令的先导指令。

指令中的标号名表示本指令在程序中的位置,如果本指令不被其他指令调用时,指令中可省去标号名。JNZ 为本指令的功能代码,它是 JUMP ON NON ZERO 的缩写。转移地址 E,在直接形式下,转移地址为 ADR,而 ADR 用标号地址给出;在间接形式下,转移地址由 ADR 和(XR)共同设定。

(3) 指令用法

以比较指令为先导实施转移:

指令①执行前,利用比较指令(GR3)与(DY)相比较,若二者不相等则满足转移条件。由于 8 与 9 不相等,所以执行指令①时,转向 BD 去执行。

以计算型指令为先导,根据计算结果的属性转移:

指令②在执行前先执行加法指令,(GR3) + (JS) = 9 + (- 12) = - 3,其结果为非零而(FR) = 10,接下来执行指令②满足转移条件,则转向 FL。

以传送指令为先导,以寄存器内容的属性为依据而转移:

指令③在执行前为传送指令,该传送指令并无实质性的运算或处理,但 LEA 指令影响标志寄存器 FR 的状态,如果 GR3 中的内容是不为零的正值或负值,那么执行指令③时转 FLZ,否则执行下一条。

6.2.5 等于(零)转移指令

(1) 指令形式

标号	操作码	操作数	备 注
[标号名]	JZE	ADR	直接 E = ADR
[标号名]	JZE	ADR, XR	间接 E = ADR + (XR)

(2) 指令功能

执行本指令,当标志寄存器的状态为(FR) = 01 时,即对于运算结果为零时,则产生转移,否则执行下一条指令。

由于本指令以标志寄存器的状态为转移的判据,所以凡是对标志寄存器产生影响的指令都可以作为本指令的前置指令,如比较指令、计算型指令、传送指令等。

指令中的标号名表示本指令在计算机中的位置,如果本指令不被其他指令调用时,在指令中可省去标号名。JZE 是本指令的功能代码,它是 JUMP ON ZERO 的缩写。转移地址 E,在直接形式下为 ADR,用标号地址给出;在间接形式下转移地址由 ADR 和(XR)两项共同设定。

(3) 指令用法

以比较指令为前置指令实施转移:

指令①执行前为逻辑比较,(GR4)与(DY)为全等,因此执行指令①时满足转移条件,转 DZ 去执行。

以运算指令为前置指令实施转移:

指令②执行前,GR4 的内容与 FF 中的内容进行异或运算。由于 - 1 在计算机中的补码为 $FFFF_H$,所以两个 $FFFF_H$ 进行异或,其结果为全 0。当执行指令②时,由于异或指令的结果为 0 而使标志寄存器 FR 中的值置为 01,这正符合指令②的转移条件,从而使指令②转移成立而转向 LZ 去执行。

以传送为前置指令实施转移:

指令③执行前由传送指令将 0 送给 GR4,使标志寄存器的状态置为 01 状态,执行指令③时正符合转移条件,从而转 CZ 去执行。

6.3　程序设计训练

【问题 6-1】　将常数 100 存入以 DY 为首地址的 60 个单元中。

【方法分析】　把一个数存入多个内存单元所涉及的主要指令是取数和存数,从设计思想上说,最简单的方法是取出 100 后连续用 60 条存数指令就可以完成本问题的要求。从程序设计的优化来说,这种设计方法使用了大量重复性的指令,不符合优化要求。此外,用简单的存、取指令可以完成 60 单元存放 100 的问题,但要完成 600 个、6000 个,甚至更多的单元存放 100 的问题,就会大大地增加了编译程序的工作量。这种设计方法显然是不可取的。

解决本问题要采用成组操作的方法,所谓成组操作对本问题来说,就是把 60 条存数指令转化为一条存数指令反复执行 60 次,这样就避免了大量使用重复指令的问题。而要实现成组处理,主要有两个问题,一个是计数问题,另一个是存数地址的改变问题。设置一个计数器,并借助比较指令可以准确地控制操作次数;使用变址寄存器,每存储一次修

改一次存数地址,这样做就可以构造出简练的程序。

【参考程序】

```
CZCS    START
        LD      GR0,CS          ;取(CS) = 100→GR0
        LEA     GR1,0           ;0→GR1,变址初始值
CF      ST      GR0,DY,GR1      ;100 存入 DY、DY + 1…DY + 59
        LEA     GR1,1,GR1       ;变址加 1
        CPL     GR1,C6          ;是否操作了 60 次
        JNZ     CF              ;未操作完转重复操作
        EXIT
DY      DS      60              ;定义 60 个存储单元
CS      DC      100             ;定义常数 100
C6      DC      60              ;定义操作个数 60
        END
```

【结果评述】 执行本程序可以完成下列取数及存储:

由于地址偏移量的改变与计数器的计数是同步变化的,且终值相同,故在程序中省去了计数器,用变址寄存器代替了计数器。

从存数的方式来说,可以从第一个单元开始,也可以从最后一个单元开始。如果从第一个单元开始,计数器初始值为 0,存储单元的地址为首地址,地址改变每次加 1,计数器也是每操作一次加 1。如果从最后一个单元开始,计数器初值为总个数,存储单元的地址为末地址,地址改变为每次减 1,计数器每操作一次减 1。下面的程序就是从后向前存数的。

```
CZDC    START
        LD      GR0,CS          ;取(CS) = 100→GR0
        LEA     GR1,60          ;地址偏移量 60→GR1
CF      LEA     GR1, - 1,GR1    ;59、58、57…1、0→GR1
        JMI     WL              ;是否操作完?
        ST      GR0,DY,GR1      ;100 存入 DY + 69、DY + 58…DY + 1、DY
        JMP     CF              ;转重复操作
```

```
WL      EXIT
DY      DS      60              ;定义 60 个存储单元
CS      DC      100             ;定义常数 100
        END
```

【问题 6-2】　在以 ZF 为首地址的 26 个单元中任意存放着 26 个英文字母'A'～'Z',设计一个查询程序,找出'A'放在单元 A 中,并把'A'所在的单元地址放在单元 AZ 中。

【方法分析】　字符在计算机中所存储的是字符所对应的 ASCII,使用字符'A'的内码 0041H 作为基准常数,借助逻辑比较可以方便地找到字符'A'。利用变址寄存器跟随着查询地址的动态变化,在找到字符'A'的同时也记录下了字符 A 距离首地址的偏移量,二者之和则为'A'的所在单元地址。下面是处理本题的示意图:

【参考程序】

```
CXA     START
        LEA     GR2,0           ;变址寄存器初值 0→GR2
CF      LD      GR1,ZF,GR2      ;取被查询字符→GR1
        CPL     GR1,CA          ;是否为字符'A'
        JZE     CL              ;找到则转,否则执行下一条
        LEA     GR2,1,GR2       ;地址改变
        JMP     CF              ;转去重找
CL      ST      GR1,A           ;将'A'存入单元 A
        LEA     GR2,ZF,GR2      ;形成'A'的所在地址
        ST      GR2,AZ          ;将'A'的所在地址存入 AZ
        EXIT
ZF      DS      26
A       DS      1
AZ      DS      1
```

```
CA      DC      #0041
        END
```

【结果评述】 在本程序中,查询字符'A'采取的是逐一比较,如果在大量的字符数据中用这种查询方法查找所需字符的话,所花费的查找时间较多。若采用二分法,0.618法等会加快查找速度,但程序要复杂些。

【问题6-3】 设计一个程序,求自然数1～16之和及平均值,分别存入A和B。

【方法分析】 求和一般是采用累加的方法,累加要设置一个寄存器作为累加器,累加器的初始值通常设定为0或者存入第一个被累加的数。本题被累加的数是自然数,数的产生可由计数器从1开始逐次加1得到,与此同时,数的累加个数也由计数器来控制。计算平均可利用算术右移。

【参考程序】

```
ZSLJ    START
        LEA     GR0,0           ;0→GR0,累加器充0
        LEA     GR1,1           ;1→GR1,计数器置1
LOOP    ADD     GR0,0,GR1       ;自然数累加
        CPL     GR1,SL          ;是否等于16
        JZE     WL              ;等转
        LEA     GR1,1,GR1       ;计数器加1
        JMP     LOOP            ;循环累加
WL      ST      GR0,A           ;累加和存入A
        SRA     GR0,4           ;求平均
        ST      GR0,B           ;平均存入B
        EXIT
A       DS      1
B       DS      1
SL      DC      16
        END
```

【结果评述】 执行本程序1～16的和存入A,其平均值存入B。在本程序中比较指令使用了CPL,它比CPA的执行速度要快。本程序累加是从1开始到16,也可以从16到1累加。由于在CASL中所保留的值均为整数,所以用右移求平均可能产生误差。

【问题6-4】 设计一个统计程序,在50个整实数中统计正数的个数。

【方法分析】 通常的正计数器,每操作一次加上1。统计计数器则是符合统计要求时加1,否则不加1。判定是否符合条件只需要在计数之前加上比较指令。这样解决本问题的思路是逐一比较50个整实数,若为正,则计数器加1,否则计数器不加1。而被比较的数据用变址寄存器的改变依次被取出就可以实现所要求的统计。

【参考程序】

```
TJZS    START
        LEA     GR4,0           ;0→GR4 数据个数计数器
        LEA     GR3,0           ;0→GR3 统计计数器置初值
        LEA     GR2,0           ;0→GR2 变址寄存器置初值
```

```
CF      LD      GR1,SS,GR2      ;取实数→GR1
        CPA     GR1,ZS          ;是否为正数
        JPZ     TJ
        JMP     XG
TJ      LEA     GR3,1,GR3       ;统计计数
XG      LEA     GR4,1,GR4       ;数据个数修改
        LEA     GR2,1,GR2       ;修改取数地址
        CPL     GR4,WS          ;是否统计完
        JNZ     CF
        EXIT
SS      DS      60
ZS      DC      0
WS      DC      50
        END
```

【结果评述】　执行本程序,可在 SS 为首地址的 50 个实整数中统计出正数的个数存放在 GR3 中。在统计中把 0 也计入在整实数中,修改常数 ZS,可以改变统计数的范围。在程序中,被统计数的计数采用的是从 0 开始的正计数,也可以将计数器的初值设定为 50,每操作一次计数器减 1,当计数器减至 0 时作为操作的终点。

【问题 6-5】　在 20 个单元中存放着数字字符 '0'～'9',将其变成数字放回原单元。

【方法分析】　数字字符与其对应的数字只相差一个常数,因此将字符变换成数字只需减掉一个常数即可。例如,字符 '9' 的内码为 0039_H,去掉 0030_H 则变为 9;从数值关系上说 '9' 的内码为 57_{10},57 减去 48 等于 9。因此从字符数据转换为数值数据的方法有多种。此外,实现本题的程序设计还涉及到数的取、存及操作个数控制,这方面的处理方法在前几题中均已使用过,在此就不再进行分析。

【参考程序】

```
WBS     START
        LEA     GR1,20          ;操作数据个数 20→GR1
        LEA     GR2,0           ;变址寄存器初始值 0→GR2
CF      LD      GR3,WS,GR2      ;取字符数据→GR3
        SUB     GR3,CS          ;字符数据减去 48
        ST      GR3,WS,GR2      ;将数值存于原处
        LEA     GR2,1,GR2       ;变址修改
        LEA     GR1,-1,GR1      ;个数减 1
        JNE     CF              ;不为零转,否则下
        EXIT
WS      DS      20
CS      DC      48
        END
```

【结果评述】　执行本程序可将以 WS 为首地址 20 个单元中的数字字符数据变换为数值数据放回原单元。如果数据变换从 20 单元的最后一个单元开始变换的话,则程序设计可以简练为个数计数器与变址寄存器共用一个。见如下示意:

```
        …
        LEA     GR2,19          ;地址偏移量→GR2
CF      LD      GR3,WS,GR2      ;取最后一个单元数据
        SUB     GR3,CS          ;变换
        ST      GR3,WS,GR2      ;放回
        LEA     GR2,-1,GR2      ;变址改变
        JPZ     CF              ;大于、等于转
        EXIT
        …
```

习 题 6

1. 根据问题的要求填空完善下列程序。

(1) 本程序是按 GR1 中的值产生转移的程序。当 GR1 为负值时转 FZ,且在 GR0 中送 -1;当 GR1 中为零时转 LZ,且在 GR0 中送 0;当 GR1 中的值非负、非零时是转 FFL,且在 GR0 中送 1。

```
GRZY    START
        LEA     GR1,0,GR1
        [          ]
        [          ]
        [          ]
FZ      LEA     GR0,[      ]
        JMP     WL
[      ] LEA    GR0,0
        JMP     [      ]
FFL     LEA     [          ]
[      ] EXIT
        END
```

(2) 寄存器 GR1 和 GR2 内容进行比较,大值存入 MAX。

```
GBJ     START
        ST      [      ],MAX
        CPA     GR2,MAX
        [          ]
        ST      [      ] MAX
JS      EXIT
MAX     DS      1
        END
```

2. 阅读下列程序,加上注释并说出程序的主要功能。

(1)
```
TYFZ    START
        EOR     GR2,DY
        JPZ     TH
        JMI     YH
TH      LEA     GR1,1
        JMP
YH      LEA     GR1,2
        EXIT
DY      DS      1
        END
```

(2)
```
SZCL    START
        LEA     GR1,0
        LEA     GR2,3
JXZ     LD      GR3,BH,GR1
        JMP     0,GR3
ACL     LD      GR4,DY
        SRA     GR4,1
        ST      GR4,A
        JMP     XG
BCL     LD      GR4,DY
        SRA     GR4,2
        ST      GR4,B
        JMP     XG
CCL     LD      GR4,DY
        SRA     GR4,3
        ST      GR4,C
XG      LEA     GR2,-1,GR2
        JNZ     JXZ
        EXIT
DY      DS      1
BH      DC      ACL
        DC      BCL
        DC      CCL
A       DS      1
B       DS      1
C       DS      1
        END
```

3. 程序设计。

(1) 设计一个程序比较 A、B 两个数,大者存于 D,小者存入 X。

(2) 设计一个程序比较 A、B、C 3 个数,找出最大数。

(3) 设计一个能统计寄存器 GR1 中所含 1 的个数的程序。

(4) 在以 DY 为首地址的 30 单元中找出最小的一个数。

(5) 在以 SJ 为首地址的 100 个单元中统计出正、零、负数各自的个数。

第 7 章

数据栈与子程序

数据栈描述的是数据存储空间,子程序是一种程序结构,从表面上看二者并没有什么关系,而事实上它们却存在着内在联系,也正是因为数据栈与子程序的紧密关系,所以才把这两部分内容放在一起介绍。

大家知道,程序是指令时有机组合,而这种有机组合在程序执行过程中的主要体现是程序的执行顺序。表征程序执行顺序的部件是程序计数器,也称指令计数器。指令计数器中记录着当前正被执行的指令地址,一旦遇到转移指令时,指令计数器中就变成了转移到的指令地址。可以想像,指令计数器只是在指令执行中的一个瞬时地址存储器,它经历了程序执行的全部路径,但每次留下来的却只是一个当前被执行的指令地址。

能不能把程序的执行过程全部记录下来呢?有没有必要记录这个过程呢?简单地说,既有必要,又有可能。我们借助程序的执行过程可以方便、准确地查找程序的执行错误及错误发生的位置,而实施程序执行跟踪的基础则是数据栈。数据栈可以按照程序的执行顺序依次记录下程序的执行地址,数据栈可以记录转子程序的返回地址,并依次释放返回地址,从而使多次转子、多重转子返回无误,这就是数据栈与子程序的内在联系。

7.1 数据栈及使用

7.1.1 栈的基本概念

(1) 栈的定义

栈是具有某种特定功能的内存空间。栈是用来存储数据的,所以也称数据栈。栈存储数据如同在一个箱子中堆放书籍,先进入箱子的书籍在箱子的底部,后放入箱子的书籍在顶部。因此,栈也称堆栈。

由于先放入栈中的数据在栈的底部,后进入栈的数据在顶部,所以当从栈中取数据时,后进入的数据先被取出,先进入的数据后被取出。数据栈的这种特定功能称之谓“后进先出”或说“先进后出”。因此数据栈被定义为具有后进先出功能的连续的内存空间。由于数据栈底部的内容是“以往”的,顶部的内容是“当前”的,所以数据栈不仅仅体现出空间的含义,而且具有时间的概念。人们利用数据栈在空间上和时间上对运行着的程序进行写实,不仅可以保证程序的有序执行,还可以方便、准确地检查、分析程序运行中所发生的错误。

(2) 栈的结构

栈作为一种特定的存储空间可以用硬件来构造,也可以用软件来设定。在 CASL 中是用软件实现的。栈是由若干连续的存储单元,一端被固定,另一端为数据的进、出口。栈用栈底、栈顶、栈指针来描述,如图 7.1 所示。

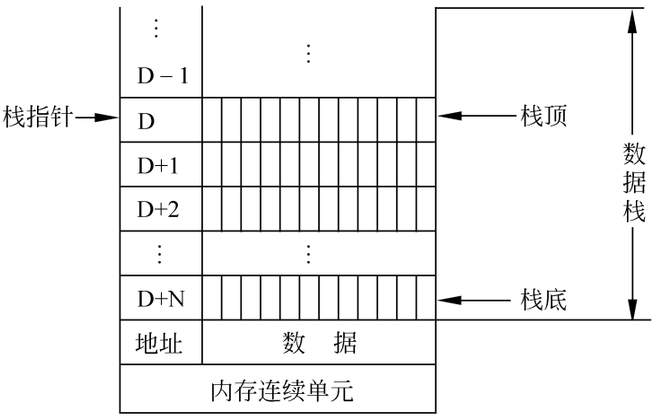


图 7.1　栈的结构

图 7.1 中,有竖线的部分表示存入栈中的数据。

① 栈底:在栈的空间中被固定的一端,即栈的最下部的那个单元。

② 栈顶:栈中数据堆的最高点,栈顶不一定是栈空间大小的顶点,它描写的是数据在栈中的最高位置。

③ 栈指针:表示数据在栈中存储状态的指示标志,栈指针在栈的操作中始终指示着栈顶地址。所以栈指针也称栈顶指针。

数据存入数据栈时称为进栈。由于数据栈一端是固定的,所以人们把数据进栈称之为压入,而数据出栈时称为弹出。

(3) 栈的状态

数据在栈中的存储状态有 3 种,即空栈、未满、存满,如图 7.2 所示。

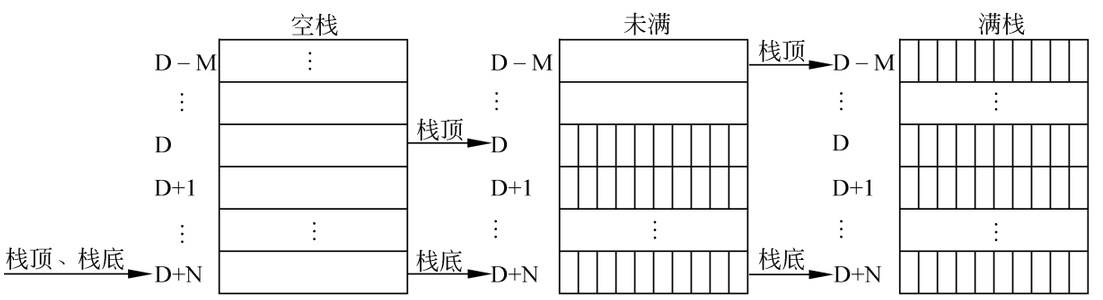


图 7.2　栈的状态

由示意图可以看出,空栈是未存入数据的数据栈,空栈是栈顶和栈底相重合的数据栈。随着数据的进栈,栈顶上升,栈顶上升的标志是栈顶地址减小。从满栈到未满栈或者从未满栈到空栈,是数据的出栈过程。数据出栈的表征是栈顶下降,栈顶下降的标志是栈顶地址增大。不难得出,栈顶地址的改变表征着数据栈的状态。

(4) 栈的使用

数据存入数据栈为进栈,从数据栈取数据为出栈。因此,数据栈的操作有两种,一种

是进栈操作,另一种为出栈操作。为此,在 CASL 中设置了进栈指令和出栈指令。由于进栈和出栈都会改变栈顶地址,而表示栈顶地址变化的是栈指针。在 CASL 中专门设置了一个栈指针寄存器 GR4,凡是栈操作,寄存器 GR4 自动做了栈顶指针,GR4 始终保留着栈顶地址。

① 进栈操作,栈顶上升,栈顶地址减小。因此,每进行一次进栈操作,GR4 便做一次减 1 的操作,即(GR4) - 1→GR4。

② 出栈操作,栈顶下降,栈顶地址增大。因此,每进行一次出栈操作,GR4 便做一次加 1 操作,即(GR4 + 1)→GR4。

7.1.2 进栈指令

(1) 指令形式

标号	操作码	操作数	备 注
[标号名]	PUSH	ADR	直接 E = ADR
[标号名]	PUSH	ADR + (XR)	间接 E = ADR + (XR)

(2) 指令功能

本指令可以把由 E 设定的数据压入数据栈。

执行本指令,首先是栈指针自动减 1,即(GR4) - 1→GR4,使栈顶上升,然后是数据 E 进栈,即 E 到(GR4)所指示的地址中,从而完成数据进栈操作。

标号名表示本指令在程序中的位置,可根据需要取、舍。PUSH 为本指令的功能代码,它是 PUSH EFFECTIVE ADDRESS 的缩写。在直接形式下,ADR 为设定的进栈数据;在间接形式下 ADR + (XR)为设定的进栈数据。进栈数据的形式可以是十进制常数,也可以是寄存器中的内容以及 CASL 中使用的其他常数。

(3) 指令用法

将考试成绩 100 进栈,

```
①    PUSH    100
```

执行指令①可将数据 100 压入数据栈中。数据 100 进栈示意如图 7.3 所示。

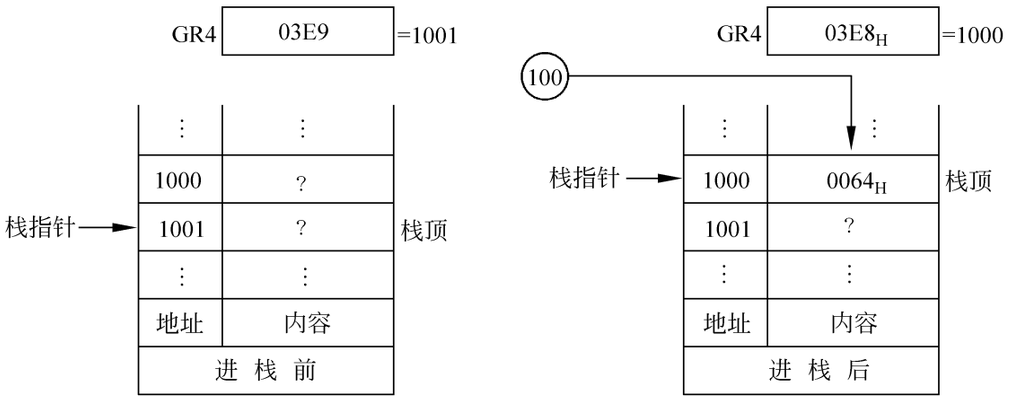


图 7.3 数据进栈

由示意图可以看出,进栈前栈指针指向栈顶地址 1001,而这个地址保留在指针寄存器 GR4 中。100 进栈时,首先(GR4 - 1)→GR4 使栈顶上升,栈顶地址由 1001 变为 1000,然后 100 = 0064_H 进栈。这一切在计算机中都是自动完成的,并不需要使用者干预。

将寄存器 GR1 的内容进栈,

② `PUSH　0,GR1`

指令②为间接形式的进栈指令。执行指令②进栈数据为

$$\begin{aligned} E &= ADR + (XR) \\ &= 0 + (GR1) \\ &= (GR1) \end{aligned}$$

该指令所完成的操作是将 GR1 中的内容进栈。

将寄存器 GR2 的内容加 1 后进栈,

③ `PUSH　1,GR2`

指令③为间接形式的进栈指令。执行指令③进栈数据为

$$\begin{aligned} E &= ADR + (XR) \\ &= 1 + (GR2) \\ &= (GR2) + 1 \end{aligned}$$

显然,指令③所完成的操作是 GR2 的内容加 1 后进栈。

将寄存器 GR3 的内容减 1 后进栈,

④ `PUSH　-1,GR3`

指令④为间接形式的进栈指令。执行指令④进栈数据为

$$\begin{aligned} E &= ADR + (XR) \\ &= -1 + (GR3) \\ &= (GR3) - 1 \end{aligned}$$

不难看出,指令④所完成的操作是 GR3 的内容减 1 后进栈。

7.1.3　出栈指令

(1) 指令形式

标号	操作码	操作数
[标号名]	POP	GR

(2) 指令功能

本指令可以把栈顶地址中的内容弹出到所设定的寄存器中。执行本指令,首先是弹出栈顶数据,((GR4))→GR,然后是栈顶下落,(GR4) + 1→GR4,栈指针指向新栈顶。

指令中的标号表示本指令在程序中的位置,可根据需要取、舍。POP 为本指令的功能代码,它是 POP UP 的缩写。GR 是存放出栈数据的寄存器,可在 GR0～GR3 中任选一个,由于 GR4 是栈指针寄存器,所以不能在指令中使用。

(3) 指令用法

将数据栈中的3个数据出栈分别存于GR1,GR2,GR3,

①	POP GR1
②	POP GR2
③	POP GR3

执行指令①、②、③可将栈顶数据依次弹入寄存器GR1、GR2、GR3。数据出栈的示意如图7.4所示。

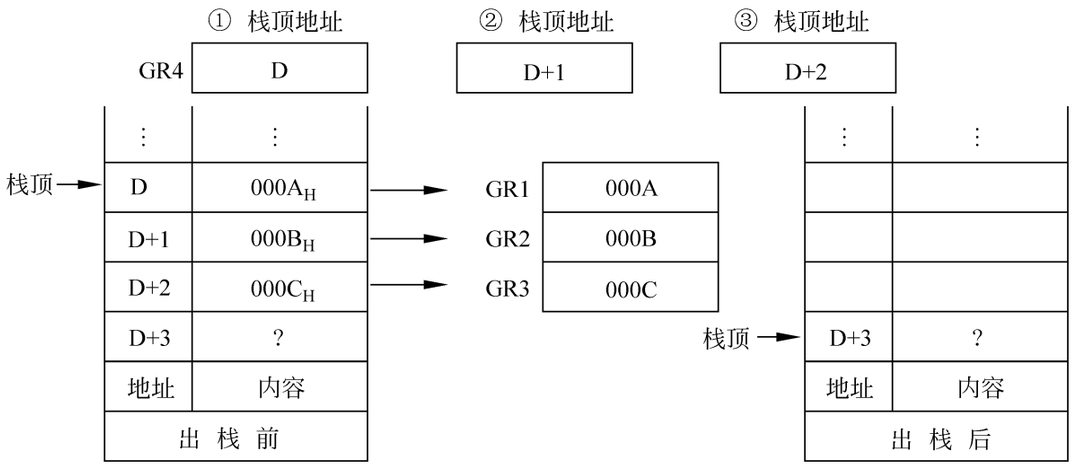


图7.4 数据出栈

出栈指令执行前,栈顶地址为D存于栈指针寄存器GR4中。

执行指令①,栈顶数据先出栈,即((GR4)) = (D) = 000A$_H$ →GR1。然后栈顶下降,即(GR4) + 1 = D + 1→GR4,新栈顶地址为D + 1,为下次出栈操作做好了准备。

执行指令②,栈顶数据先出栈,即((GR4)) = (D + 1) = 000B$_H$ →GR2。然后栈顶下降,即(GR4) + 1 = D + 1 + 1 = D + 2→GR4,新栈顶地址为D + 2,为下次出栈操作做好了准备。

执行指令③,栈顶数据先出栈,即((GR4)) = (D + 2) = 000C$_H$ →GR3。然后栈顶下降,即(GR4) + 1 = D + 2 + 1 = D + 3→GR4,新栈顶地址为D + 3,为下次出栈操作做好了准备。

7.2 子程序及使用

7.2.1 子程序的基础知识

(1) 子程序与主程序

把具有某种功能并多次被执行的程序段,从一个程序中相对独立出来所形成的程序为子程序。子程序是一种可以被其他程序调用的程序,而调用子程序,且不被其他程序调用的程序被称为主程序。

(2) 主、子程序的连接

一个程序和另一个程序的连接可以使用 GO TO 语句,而使用这种无条件转移语句虽然可以把两个程序连接了起来,但由于这种转移语句是“一去不复返”的转移,所以转出去再想回来就不行了。如果使用两个 GO TO 语句,我们还是可以实现从哪儿转出又转回哪儿。例如:

程序①从 M + 1 转程序②,为了使转移能够返回,在程序②的 Z + 4 处又用了一条转移语句返回到原转出去的下一条指令,这样完成了程序①调用程序②的连接问题。如果我们把程序①看成主程序,把程序②看成子程序,这就是主、子程序的连接问题。然而,如果在实用中使用这种连接办法还是不太方便。因此,在语言系统中都设置了专门的转子、返主语句或指令,用以进行主、子程序的连接。

在 CASL 中,CALL 指令用于转子,RET 指令用于返主。使用 CALL 指令转子时,可以把返回地址自动送入数据栈;在子程序中遇到返主指令 RET 时,又可以自动将返回地址出栈并按返回地址返回主程序。

7.2.2　转子指令

(1) 指令形式

标号	操作码	操作数	备　注
[标号名]	CALL	ADR	直接 E = ADR
[标号名]	CALL	ADR, XR	间接 E = ADR + (XR)

(2) 指令功能

执行本指令,将按照使用者所设定的子程序入口地址 E 转向子程序去执行。

指令中的标号名表示本指令在程序中的位置,根据需要取、舍。CALL 为本指令的功能代码,它是 CALL SUBROTINE 的缩写。在直接形式下转子入口地址 E = ADR,而 ADR 用标号地址给出;在间接形式下转子入口地址 E = ADR + (XR),而 XR 为变址寄存器,只能在 GR1～GR3 中任选一个,这是因为在 CALL 中 GR0 不能用作变址寄存器,CALL 指令与栈操作有关,GR4 为栈顶指针,因而也不能选用 GR4 作变址。

执行本指令的同时,在计算机内部将自动产生一个进栈的操作:先是栈顶上升,即(GR4) - 1→GR4,后是将 CALL 指令所在地址的下一个地址送入数据栈,以备返回指令使用,接下来是转向子程序入口地址,即将入口地址 E 送给指令计数器 PC。

(3) 指令用法

转子到入口地址为 ZK 的子程序,

① M CALL ZK

指令①为直接形式下的转子指令。执行指令①则转向入口地址为 ZK 的子程序去执行。在计算机内部,该指令的执行情况如图 7.5 所示。

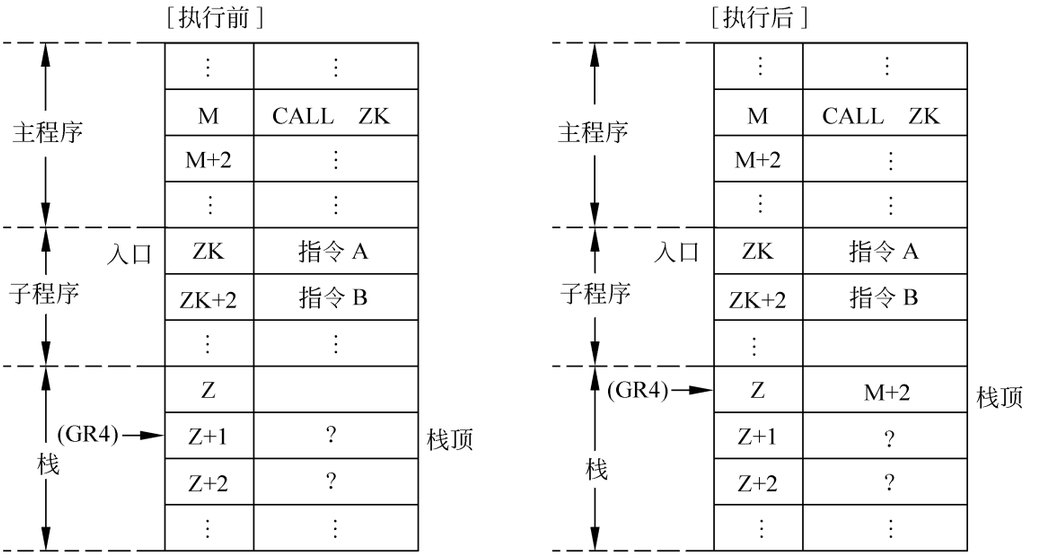


图 7.5 转子指令操作

由示意图可以看出,在执行转子指令前,在主程序的 M 地址设置了一条转子指令,子程序的入口地址为 ZK。由于 CALL 的每一条指令占两个单元,所以 CALL 指令的下一条指令地址为 M + 2,而这个地址为子程序的返回地址。

执行转子指令 CALL ZK,需要将返回地址 M + 2 进栈,为此数据栈的栈顶上升,

即 (GR4) - 1→GR4

= Z + 1 - 1→GR4

= Z(为新栈顶)→GR4

随之,返回地址进栈,

即 (PC) + 2→Z

= M + 2→Z

接下来是转移到子程序的入口地址,如何实现这个转移呢?在计算机中承担执行指令顺序的是指令计数器 PC,PC 中的地址就是当前被执行的指令地址,所以实现转移到 ZK,就是把 ZK 送入 PC,即 ZK→PC。

按照寄存器 GR1 的内容转子,

② MM CALL 0,GR1

指令②为间接形式的转子指令。执行指令②转子入口地址为 E = ADR + (XR) = 0 + (GR1) = (GR1)。显然,转子入口地址取决于 GR1 的内容。如果在执行 CALL 指令前在

GR1 中送入 ZK,即:

```
LEA   GR1,ZK
CALL  0,GR1
```

这样与指令①的作用就一样了。使用间接形式的转子指令,由于指令中含有变址寄存器,通过变址的改变可以用同一条转子指令转到不同入口的子程序。

按标号地址与 GR2 的内容转子,

③

```
MMM    CALL    ZK,GR2
```

指令③中的 ZK 可以看成是进入子程序的基址或者是入口地址之一,修改变址寄存器 GR2 的内容,则可以用同一条转子指令转入一个有多个入口的子程序。例如:

```
       LEA     R2,6
MMM    CALL    ZK,GR2
       LEA     GR2, - 2,GR2
       JPZ     MMM
       EXIT
```

不难得出,转子指令 MMM 相当 4 条转子指令,而转入一个有 4 个入口的子程序,如下:

7.2.3　返主指令

(1) 指令形式

标号	操作码	操作数
[标号名]	RET	空

(2) 指令功能

执行本指令将自动返回主程序转子指令的下一条指令的地址去执行。

指令中的标号名表示本指令在程序中的位置,可根据需要取舍。RET 为本指令的功

能代码,它是 RETURN 的缩写。

执行本指令为什么能够自动、准确地返回主程序呢？其原因是在执行本指令的同时,在计算机中还有下列操作:

① 将返回地址出栈,即把转子时保留在数据栈中的返回地址((GR4))弹出数据栈。

② 恢复栈指针,在转子时栈顶曾上升并保留了返回地址,因此返主指令执行时要弹出返回地址使栈顶下降恢复到转子前的状态。简单地说,恢复栈指针的操作为(GR4) + 1→GR4。

③ 将返回地址送到指令计数器,即((GR4))→PC。

(3) 指令用法

本指令始终被设置在一个子程序的逻辑结尾,它与转子指令成对出现。转子 CALL 与返主 RET 关系如图 7.6 所示。

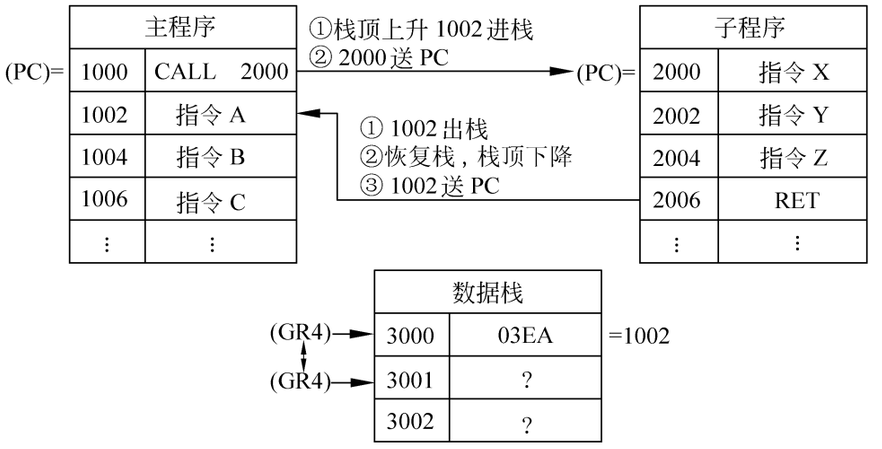


图 7.6 CALL 与 RET 关系

7.3 程序设计训练

【问题 7-1】 利用数据栈计算(GR1) + (GR2) + (GR3),并将结果存于 GR0。

【方法分析】 利用数据栈在计算中存储计算使用的初始数据或者计算的中间结果,与使用一般的存储单元有两点不同之处。第一点,数据栈有“先进后出”或说“后进先出”功能,由于此功能体现着一种空间顺序,这给有序操作提供了方便。第二点,取用存入数据栈的数据不需要标号地址,只需要借助栈顶指针 GR4 就可以方便地完成存入数据栈数据的取用。在此我们把使用普通单元中的数据与使用栈中数据做一个比较。例如:

```
LD      R0,DY          取普通单元 DY 中的数据
LD      GR0,0,GR4      ;取数据栈中的数据
ADD     GR0,DY         ;与普通单元中的数据相加
ADD     GR0,0,GR4      ;与栈顶地址中的数据相加
```

由于 GR4 中的内容是栈顶地址,所以 GR4 内容的内容则为栈顶数据。也就是说,(GR4)为栈顶地址;((GR4))为栈顶数据。

本题要求利用数据栈,一是利用栈的存储,二是利用栈指针使用存入栈中的数据。

【参考程序】

```
ZYS     START
    ①   PUSH  0,GR1        ;GR1 的内容进栈
    ②   PUSH  0,GR2        ;GR2 的内容进栈
        LEA   GR0,0,GR3    ;GR3 的内容传送到累加寄存器
    ③   ADD   GR0,0,GR4    ;累加栈顶数据,即(GR3) + (GR2)→GR0
    ④   ADD   GR0,1,GR4    ;(GR3) + (GR2) + (GR1)→GR0
    ⑤   POP   GR2          ;恢复 GR2
    ⑥   POP   GR1          ;恢复 GR1
        EXIT
        END
```

【结果评述】　执行本程序,可将(GR3) + (GR2) + (GR1)的结果存入 GR0。本程序借助栈存储了参加求和的数据,见指令①和②。执行指令①、②前后,数据栈的状况如下:

栈顶	?
	?
地址	内容
使用前的数据栈	

栈顶→

(GR4)	(GR2)	;执行指令②
(GR4) + 1	(GR1)	;执行指令①
	?	;以前数据
	?	;以前数据
地址	内容	
使用后的数据栈		

可以看出,执行了指令①、②后,原数据栈栈顶上升,(GR1)和(GR2)依次压入栈中,栈指针 GR4 中的内容为栈顶地址,该地址中是后进栈的 GR2 的内容。指令③、④是使用数据栈中的数据,指令③中引用了 GR4,它的内容的内容,即((GR4))正好是后入栈的(GR2)。指令④中又引用了 GR4,指令中操作数的地址为

$$E = ADR + (XR)$$
$$= 1 + (GR4) = (GR4) + 1$$

这个地址中的内容恰恰为(GR1)。在此需要说明的是,指令③引用了 GR4,并取出了(GR2),这并不意味着 POP 指令的弹出作用。因此,执行指令③不存在栈顶下降的问题。指令⑤、⑥为恢复栈指令,凡是使用了数据栈,都必须恢复,这是使用栈的规定。由指令⑤、⑥可以看出,恢复数据栈就是把压入的内容再弹出来放在原来的存储空间,使栈顶恢复到使用前的状态。值得注意的是进栈顺序和恢复顺序是相反的,例如:

```
PUSH    0,GR1       ;① 先 GR1
PUSH    0,GR2       ;② 再 GR2
  …
POP     GR2         ;① 先 GR2
POP     GR1         ;② 再 GR1
  …
```

【问题 7-2】 利用子程序及主、子结构计算(A×B) + (C×D) + (E×F)并将结果存入GR0,但 A、B、C、D、E、F 均为正整数。

【方法分析】 由于(A×B)、(C×D)、(E×F)的计算式均为二数相乘,所以可以设计一个相对独立的程序,即子程序来完成二数相乘。而在主程序中提供参加相乘的二数,然后再把 3 次二数相乘的结果累加起来便得出最后结果。

二数相乘可以采用连加的办法,例如 3×8 可以转化为按照 3 的个数去累加 8,即 8 + 8 + 8;再如,5×6 为 5 个 6 或者 6 个 5 之和。因此,在子程序中用一个乘数做计数器,另一个乘数作为累加数就可以完成二数相乘。

【参考程序】

```
ZZC     START
        LEA     GR0,0           ;结果累加器置初值 0
        LD      GR1,A           ;乘数 A→GR1
        LD      GR2,B           ;乘数 B→GR2
        CALL    RSC             ;转子做二数相乘,即 A×B
        ADD     GR0,W           ;累加第一项
        LD      GR1,C           ;乘数 C→GR1
        LD      GR2,D           ;乘数 D→GR2
        CALL    RSC             ;转子做二数相乘,即 C×D
        ADD     GR0,W           ;累加第二项
        LD      GR1,E           ;乘数 E→GR1
        LD      GR2,F           ;乘数 F→GR2
        CALL    RSC             ;转子做二数相乘,即 E×F
        ADD     GR0,W           ;累加第三项,即结果→GR0
        EXIT
RSC     LEA     GR3,0           ;0→GR3
        ST      GR3,W           ;0→W
CF      LEA     GR1,-1,GR1      ;一个乘数的个数倒计数
        JMI     FH              ;个数计完转
        LEA     GR3,0,GR2       ;另一个乘数→GR3
        ADD     GR3,W           ;累加另一个乘数
        ST      GR3,W           ;二数乘积→W
        JMP     CF              ;转 CF
FH      RET                     ;返主
A       DS      1
B       DS      1
C       DS      1
D       DS      1
E       DS      1
F       DS      1
W       DS      1
        END
```

【结果评述】 执行本程序可将(A×B) + (C×D) + (E×F)的计算结果存于 GR0。在主

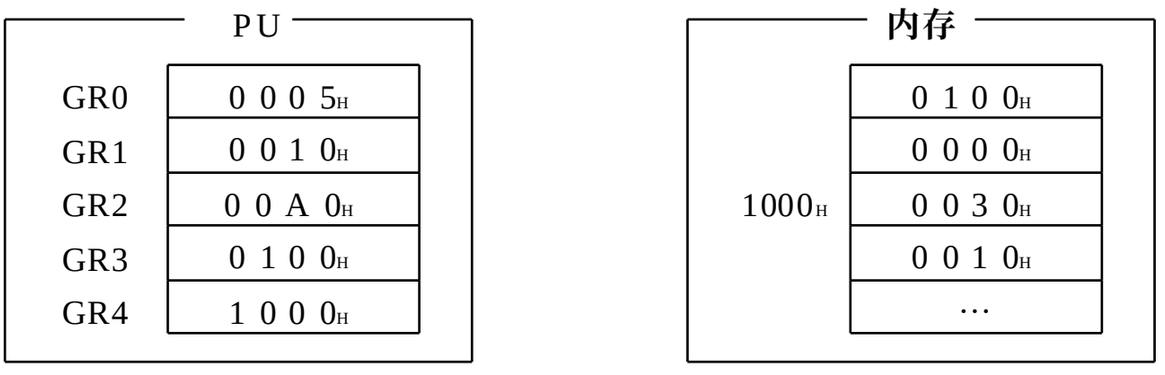
程序中,为了说明上的直观使用了 3 组指令相同的程序段,因而使程序显得冗长。在这点上可以进行压缩简化。

习　题　7

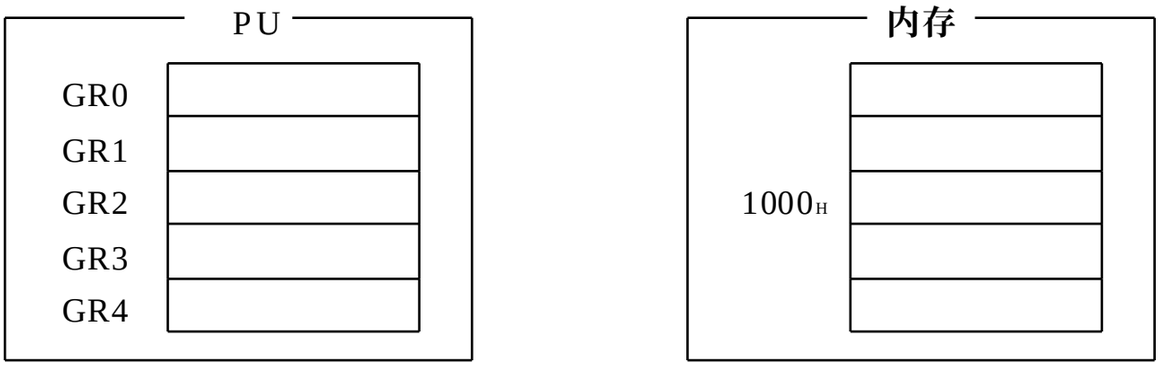
1. 按要求写出栈操作指令。
   (1) 将十进制数 1626 压入栈中。
   (2) 将十六进制数 $FFFF_H$ 压入栈中有哪几种写法?
   (3) 将标号名,即标号地址 ABC 压入栈中。
   (4) 将寄存器 GR1 的内容进栈。
   (5) 将寄存器 GR2 的内容减 6 进栈。
   (6) 将寄存器 GR3 的内容加 81 进栈。
2. 栈指针(GR4)的使用。
   (1) 在栈操作下,用 LEA 指令修改栈指针,使栈顶上升。
   (2) 在栈操作下,用 LEA 指令修改栈指针,使栈顶下降。
   (3) 用 ST 指令将 GR0 中的内容写入栈顶。
   (4) 用 LD 指令取出栈顶数据放入 GR1 中。
   (5) 已知数据栈共占 5 个单元,且存满数据,若想取出栈底数据,用 LD 指令如何实现将该数据存入 GR2?
3. 用栈指令传送数据。
   (1) 用进、出栈指令将 62 存入 GR1 中。
   (2) 用进、出栈指令将寄存器 GR2 的内容传送到 GR3 中。
   (3) 用进、出栈指令将 GR1 的内容与 GR2 的内容互换。
4. 分析下列程序段,回答所设定的问题。

```
① PUSH   0,GR1
② PUSH   1,GR2
③ POP    GR1
④ POP    GR0
```

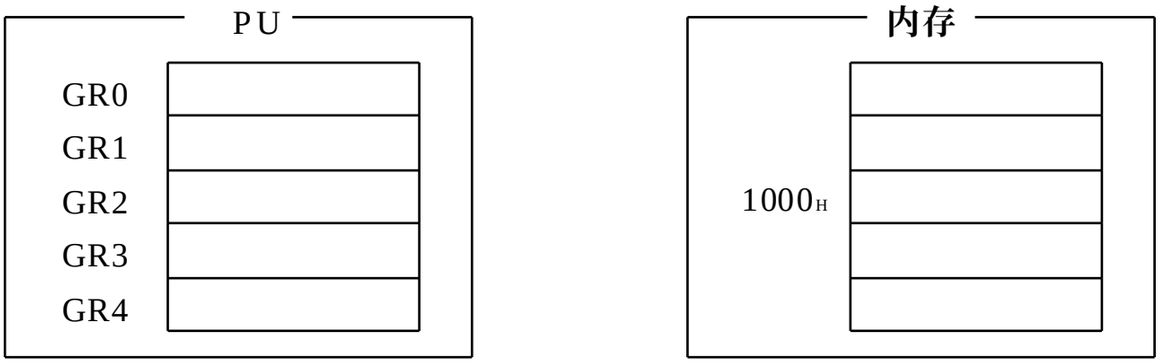
已知上列指令执行前 CPU 和内存的状况如下:



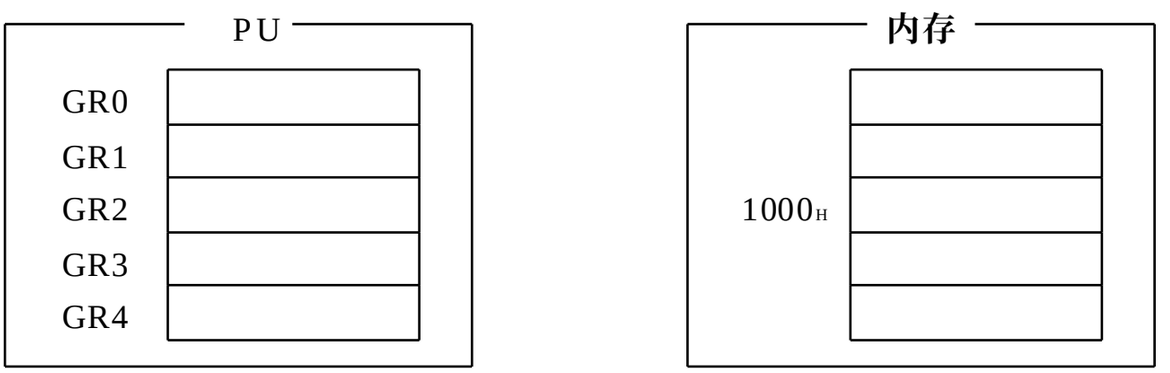
(1) 执行了指令①后,填写 CPU 和内存的状况。



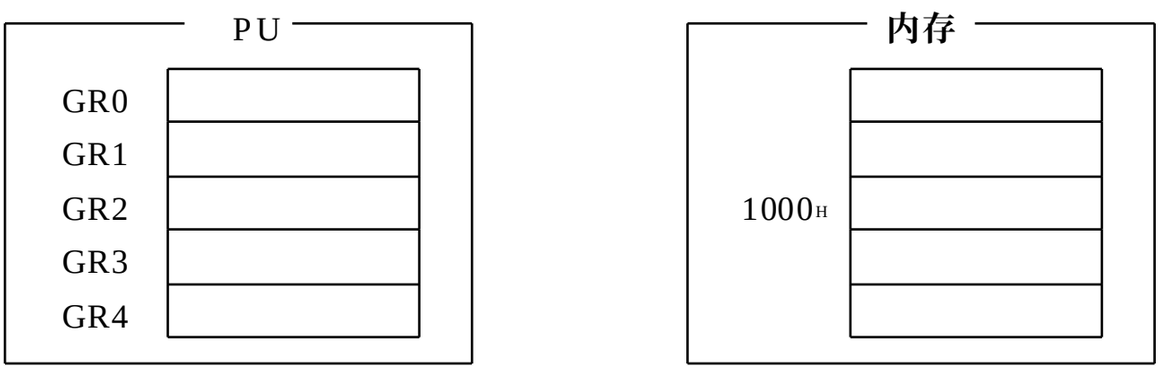
(2) 执行了指令②后,填写 CPU 和内存的状况。



(3) 执行了指令③后,填写 CPU 和内存的状况。



(4) 执行了指令④后,填写 CPU 和内存的状况。



5. 阅读程序,回答问题。

下列程序是一个含有进栈、出栈、转子和返主指令的程序。该程序已经分配了实际地址。

地址	序号	标号	操作码	操作数
0000		ZZC	START	
0002	①		LEA	GR1,10
0004	②		LEA	GR2,20
0006	③		PUSH	-1
0008	④		PUSH	0,GR1
000A	⑤		PUSH	0,GR2
000C	⑥		CALL	SUB
000E	⑦		POP	GR2
0010	⑧		POP	GR1
0012	⑨		POP	GR0
0014	⑩		EXIT	
0016	1	SUB	LEA	GR3,1,GR3
0018	2		RET	
			END	

程序执行前,数据栈的状况为

栈指针(GR4)→

FFFA	
FFFB	
FFFC	
FFFD	
FFFE	
FFFF	?
地址	**单元内容**
数据栈	

栈顶

(1) **地址** 000A$_H$ **的指令⑤执行后,数据栈的状况如何?**

(GR4):

FFFA	
FFFB	
FFFC	
FFFD	
FFFE	
FFFF	
地址	**单元内容**
数据栈	

(2) **地址** 000C$_H$ **的指令⑥执行后,数据栈的状况如何?**

(GR4):

FFFA	
FFFB	
FFFC	
FFFD	
FFFE	
FFFF	
地址	单元内容
数据栈	

(3) 地址 0018_H 的指令 2 执行后,数据栈的状况如何?

(GR4):

FFFA	
FFFB	
FFFC	
FFFD	
FFFE	
FFFF	
地址	单元内容
数据栈	

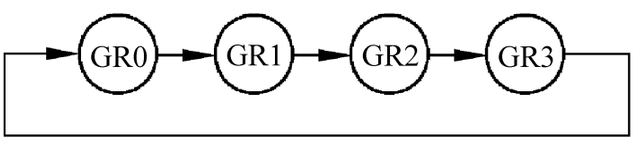
(4) 程序执行完后,寄存器 GR0、GR1、GR2 中的内容是什么?

(GR0):	
(GR1):	
(GR2):	

6. 下面是一个主程序多次调用子程序主——子结构的程序段,在给定的流程中填写出该程序的执行顺序,如图 7.7 所示。

7. 程序设计。

(1) 设计一个程序,利用栈操作完成下列寄存器间的数据交换。



(2) 已知自然数 N,设计一个程序计算:

$$H = \sum_{i=1}^{8} N_i + \sum_{i=1}^{12} N_i + \sum_{i=1}^{18} N_i$$

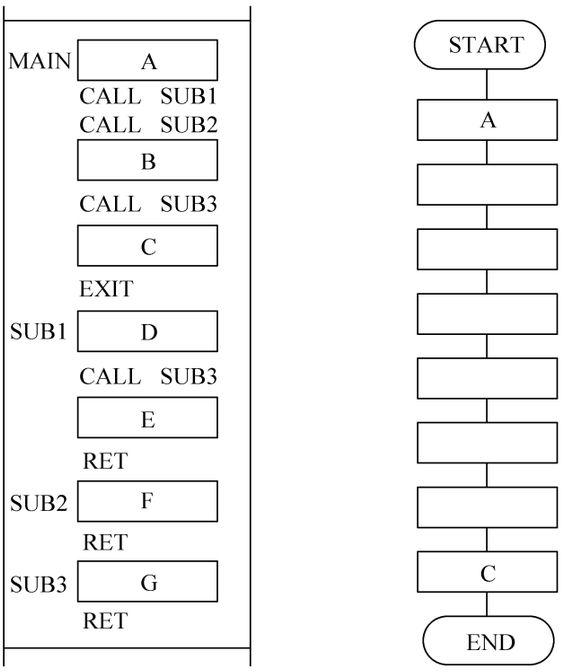


图 7.7　主程序多次调用子程序流程

第 8 章

程序设计基础

使用 CASL 进行程序设计要了解 CASL 的软、硬件环境,要熟悉 CASL 的指令系统,要掌握 CASL 描述问题的方法等。如果从广义上讨论程序设计,它所涉及到的基础和概念就更多了。像程序组织、算法描述、程序风格、程序质量、程序优化以及结构化程序设计等。显然,在 CASL 程序设计中最实际的目标是能够阅读 CASL 程序和设计 CASL 程序。为此,本章主要介绍程序流程图、程序结构、CASL 指令的功能与用法、CASL 对一些基本问题的描述方法等。

8.1 程序流程与结构

8.1.1 程序流程图

(1) 解题过程

使用 CASL 语言进行程序设计,一般需要下列几步:

提出问题是编写程序的第一步,提出问题是指明确需要解决的问题。分析问题是指找出解决问题的方法,如处理方案、计算方法等。画流程图是指按照确定的处理方案或计算方法,用一种直观的表达方式表达出程序设计思想。再根据流程图所表示的逻辑关系,用适当的指令编排出解决所提问题的程序。此外,程序编好后一般需要静态检查认为无误后,经上机调试,发现问题再经修改,直至得出正确结果。

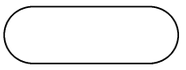
(2) 程序流程图及常用符号

程序流程图也叫程序框图,常简称框图。流程图是表达程序编排顺序、指令执行过程及内在关联的逻辑关系图。流程图是编写程序的向导,画出正确的流程图来引导编写程序可以提高编程序的速度和效率。借助流程图还可以检查、分析程序。因此,流程图在程序设计中还是相当重要的。

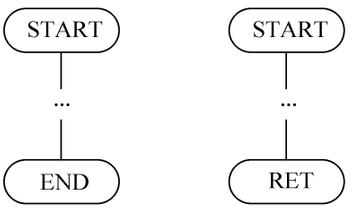
流程图分为粗框图和细框图,粗框图一般用来表示调度软、硬件资源称为系统调度框图,细框图展示的是计算或处理的逻辑关系。

流程图常用图形及符号如下:

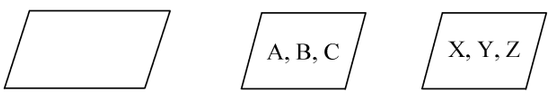
①



表示开始或结束。如果是开始框,则框内写 START;如果是结束框,则框内写 END 或 RET。此框总是放在流程图的开头和末尾。例如:

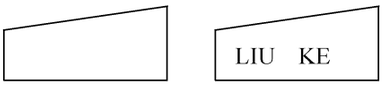


②



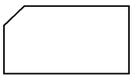
表示广义的输入、输出。所谓广义的输入、输出是指没有确定输入、输出设备的输入、输出。例如,要输入 A,B,C 或想输出 X,Y,Z 时,则可以把输入、输出的内容写在框内。

③



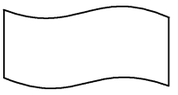
表示在控制台或键盘用手进行输入。使用时将输入内容写在框内。

④



表示卡片输入。近年来很少使用这种介质输入,在 20 世纪 80 年代使用很广泛。

⑤

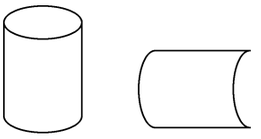


表示纸带输入。目前很少使用。

⑥

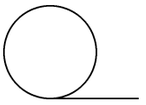
表示软磁盘,既可输入也可输出。

⑦



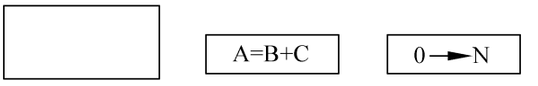
表示使用硬磁盘做输入、输出。如果读、写盘文件,常使用右图。

⑧



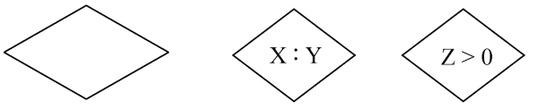
表示使用磁带,读出、写入带文件。

⑨



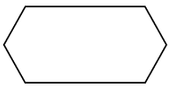
表示计算或处理。使用时将计算式或表达式写在框内。

⑩

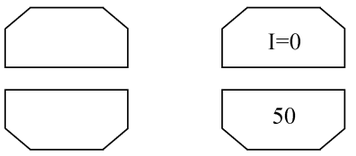


表示判断、比较。使用时可把比较对象或判别式写在框内。

⑪



表示准备。常用在循环之前,表示所做的各项准备,如计数器设定初值,累加器清 0 等。

2 

表示循环体的上、下界,两个框成对使用。

3 

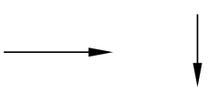
表示显示输出。使用时可把显示输出的内容写在框内。

4 

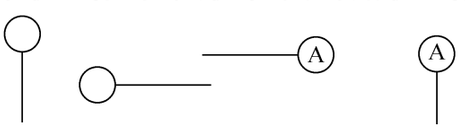
表示打印输出,使用时可在框内写上打印内容。

5 

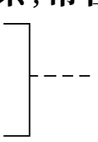
表示子程序、使用时可在框内写上子程序名或转子入口地址。

6 

表示流线。在画流程图时用流线把各功能框连接起来。

7 

表示连接。在画流程图时遇到一张纸或一页画不下时,为了和另一页的流程图表示连接关系,常在圆圈内写上数码编号或字母。例如 A 与 A 是连在一起的。

8 

表示注解。在流程图中,某一框或某一处需要予以注解时,用此框标出,并在虚线之后加注说明。

8.1.2 程序结构

现实中的程序设计问题尽管种类繁多、难易不同、要求各异。然而,描写、刻画现实问题的程序,就其结构而言却只有有限的几种。掌握了这几种程序结构就可以面对所有问题来选择程序设计所采用的结构了。常用的程序结构有顺序结构、分支结构和循环结构。使用这三种结构设计出来的程序分别称为顺序程序、分支程序和循环程序。此外,含有子程序的主—子程序也可以说是一种程序结构。

(1) 顺序程序

顺序程序只能描述一些简单的计算或处理问题。顺序程序所描写的处理过程是一种不需要进行判断、比较的顺序过程。顺序程序中的指令都是依次排列,而在程序执行过程中只被执行一次。

顺序程序的流程图结构如图 8.1 所示。

由顺序程序的流程图可以看出,从流程图开头到结尾全部由一条流线贯通起来。下面是 序程序的实例。

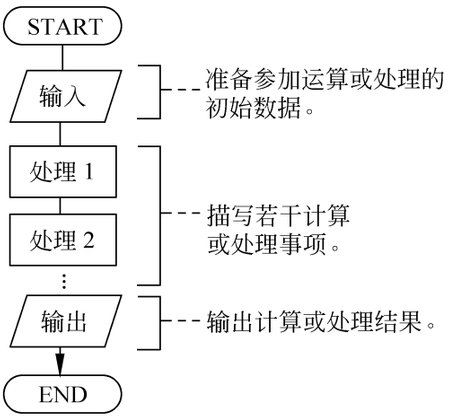


图 8 .1　顺序程序流程图

【例 1】　考试科目为数学、政治、外语,求 3 科总分?

【程序流程图】

如图 8 .2 所示。

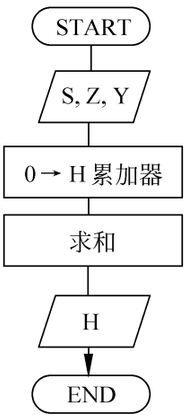


图 8 .2　求 3 科总分流程图

【参考程序】

```
QH      START
        LEA     GRO,O       ;累加器初值置 0
        ADD     GRO,S       ;累加 S
        ADD     GRO,Z       ;累加 Z
        ADD     GRO,Y       ;累加 Y
        ST      GRO,H       ;和→H
        EXIT
D       DS      1           ;S,数学成绩
        DS      1           ;Z,政治成绩
        DS      1           ;Y,外语成绩
H       DS      1           ;H,三科成绩之和
        END
```

(2) 分支程序

分支程序可以描写复杂的实际问题,在程序中不仅有计算和处理,而且有比较、判断。分支程序从流程图上看不是由一条流线贯通所有的处理,而是以比较、判断框为节点分为两条

上的流线连接所有的图框,最终都汇集在结束框处。图 8 3 是分支程序的基本结构流程图。

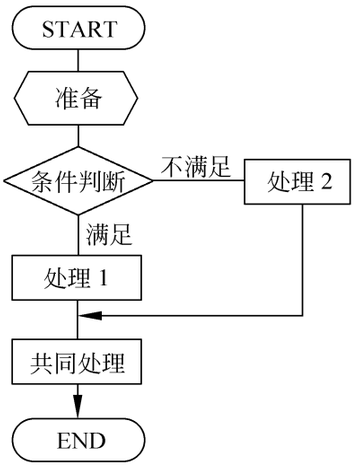


图 8 .3　分支程序流程图

分支程序和顺序程序相比,不同之处就是条件判断和分支。一个条件判断可以产生两个分支、3 个分支或多个分支。到底有多少分支,要看问题需要。下面是一个两分支的程序实例。

【例 2】　考试科目为数学、政治、外语,3 科总分达到 180 分为合格,设计出相应的检验程序?

【程序流程图】

如图 8 .4 所示。

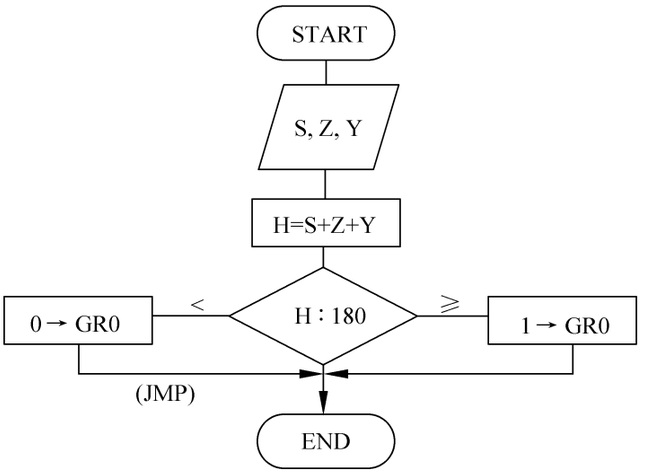


图 8 .4　检验 3 科总分是否达标的程序流程图

【参考程序】

```
JY      START
        LEA   GR0,0      ;0→GR0 累加器
        ADD   GR0,S      ;累加 S
        ADD   GR0,Z      ;累加 Z
        ADD   GR0,Y      ;累加 Y
        ST    GR0,H      ;和→H
        CPA   GR0,DB     ;H～180
        JPZ   HG         ;达标转 HG
BH ①    LEA   GR0,0      ;未达标 0→GR0
```

```
      ②    JMP   WL        ;转终止
HG ③       LEA   GR0,1     ;达标 1→GR0
WL         EXIT
S          DS    1         ;数学成绩
Z          DS    1         ;政治成绩
Y          DS    1         ;外语成绩
H          DS    1         ;3 科成绩和
DB         DC    180       ;达标常数
           END
```

在分支程序中,最容易发生的错误是逻辑上的错误,所谓逻辑上的错误,就是混淆两种不同的处理。例如,在上列程序中,指令①是未达标所进行的处理,指令③是达标处理。如果没有指令②将两种处理隔开,而在程序运行时就会出现执行了指令①,又去执行指令②的逻辑错误。在分支程序设计中,忘掉指令②的情况经常发生。从流程图上看,两种处理经纬分明,但一对应指令往往就容易出现问题,其原因出在使用无条件转移指令上。在顺序程序中,流程图上的流线只起连接及表示指令顺序的作用,它不对应具体的指令。而在分支程序中必有一条流线对应一条无条件转移指令。掌握了这一点,就会在很大程度上避免在分支程序设计中出现逻辑错误。最后再强调一下,无条件转移指令是分支程序中的一堵墙,用来隔开两种不同处理。

(3) 循环程序

循环程序所描写的是同一种处理需要反复进行多次操作的问题。例如计算一个班 50 人,每个人 6 科成绩的总分就需要循环 50 次来算 6 科成绩。在循环过程中,为了判断是否达到了处理所要求的次数,也需要判断比较,在这点上循环程序与分支程序有共同之处。从流程图来看,循环程序流程图中的流线具有迂回、闭合的特征。从程序执行顺序来说,反复操作的部分反复被执行,而其他部分按　序执行。图 8 .5 是循环程序流程图的基

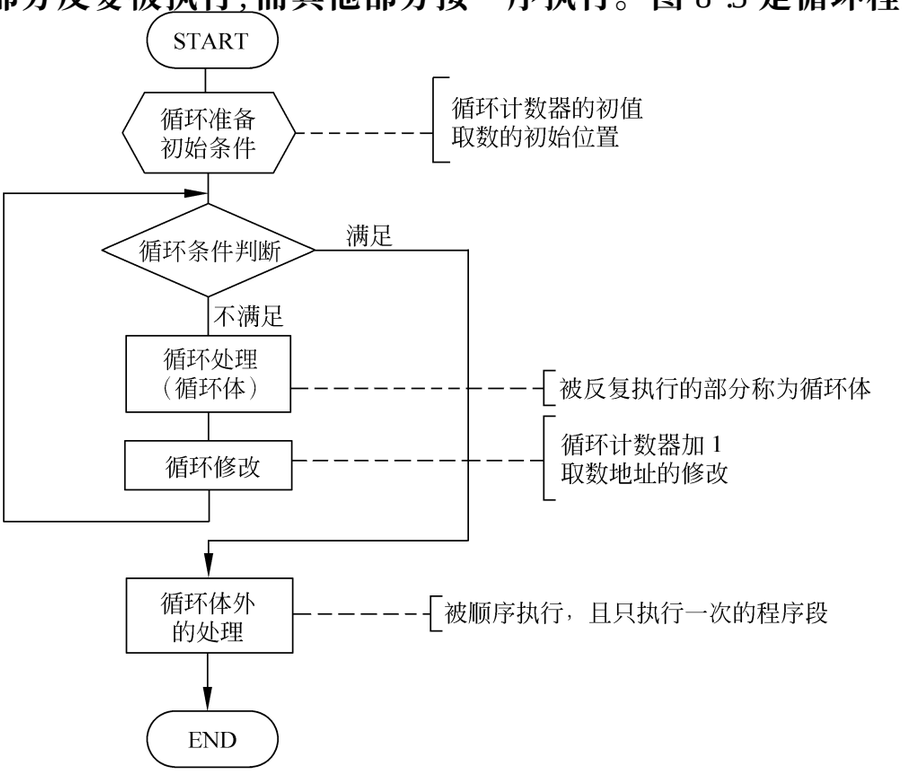


图 8 .5　循环程序流程图

本结构。

下面是循环程序的实例。

【例 3】 现有 40 名考生数学、政治、外语等 3 科的考试成绩，成绩的存放形式如下所示。求出每人 3 科的总成绩，存放在 ZF 为首地址的 40 个单元中。

3 科考试成绩		
序号	地址	分数
1	CJ	数学 1
	CJ + 1	政治 1
	CJ + 2	外语 1
2	CJ + 3	数学 2
	CJ + 4	政治 2
	CJ + 5	外语 2
…	…	…

每个人的总成绩		
序号	地址	总 分
1	ZF	总分 1
2	ZF + 1	总分 2
3	ZF + 2	总分 3
4	ZF + 3	总分 4
5	ZF + 4	总分 5
6	ZF + 5	总分 6
…	…	…

【程序流程图】

如图 8 .6 所示。

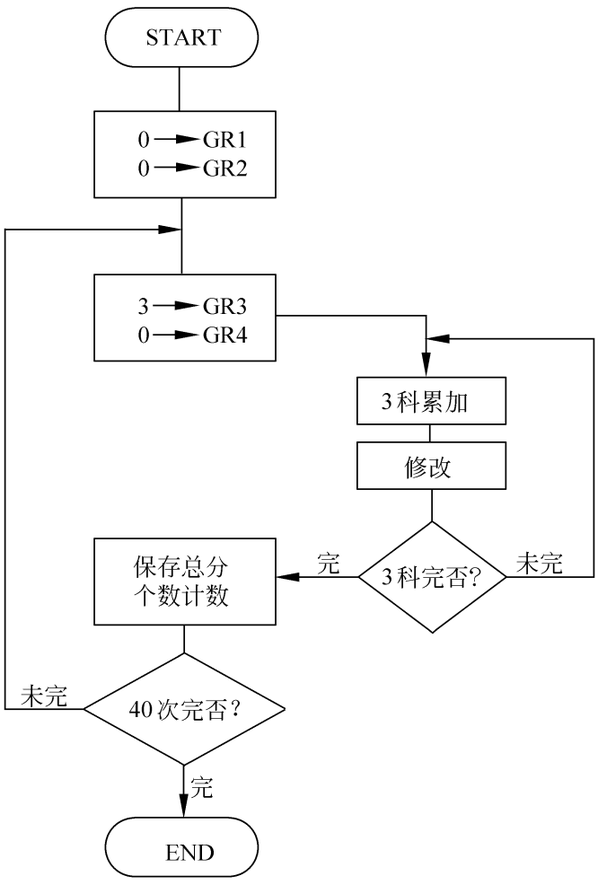


图 8 .6 例 3 的程序流程图

【参考程序】

```
ZF    START
      LEA   GR1,0          ;总分个数计数器 0→GR1
      LEA   GR2,0          ;变址初值 0→GR2
JX    LEA   GR3,3          ;3 科计数 3→GR3
      LEA   GR4,0          ;3 科累加器 0→GR4
LJ    ADD   GR4,CJ,GR2     ;成绩累加
      LEA   GR2,1,GR2      ;变址修改
      LEA   GR3,-1,GR3     ;累加个数修改
      JNZ   LJ
      ST    GR4,ZF,GR1     ;保存总分
      LEA   GR1,1,GR1      ;个数计数
      CPL   GR1,SS         ;(GR1)～40
      JMI   JX             ;未到 40 转
      EXIT
CJ    DS    120            ;40 人的 3 科成绩
ZF    DS    40             ;40 个总分存储单元
SS    DC    40             ;常数 40
      END
```

(4) 子程序

从程序结构上说,子程序可能是顺序程序,也可能是分支程序或循环程序。人们提出子程序的概念,目的有两个。一个是可以使程序流程图既在宏观上清晰,又在微观上细致,第二个目的则是为了扩大程序的应用范围,人们把一些通用的问题编成子程序可以提供给更多的人使用。下面用实例来说明使用子程序时流程图的画法。

【例 4】　设计一个主—子结构的程序,统计 60 人 3 科成绩的合格人数。要求:数学、政治、外语 3 科的总分要达到 180 分,且外语要达到 60 分才算通过。各科成绩的存放形式如下所示。

数学成绩	
地址	分数
SX	第 1 人分数
SX + 1	第 2 人分数
SX + 2	第 3 人分数
SX + 3	第 4 人分数
…	…

政治成绩	
地址	分数
ZZ	第 1 人分数
ZZ + 1	第 2 人分数
ZZ + 2	第 3 人分数
ZZ + 3	第 4 人分数
…	…

外语成绩	
地址	分数
WY	第 1 人分数
WY + 1	第 2 人分数
WY + 2	第 3 人分数
WY + 3	第 4 人分数
…	…

【程序流程图】

如图 8 .7 所　。

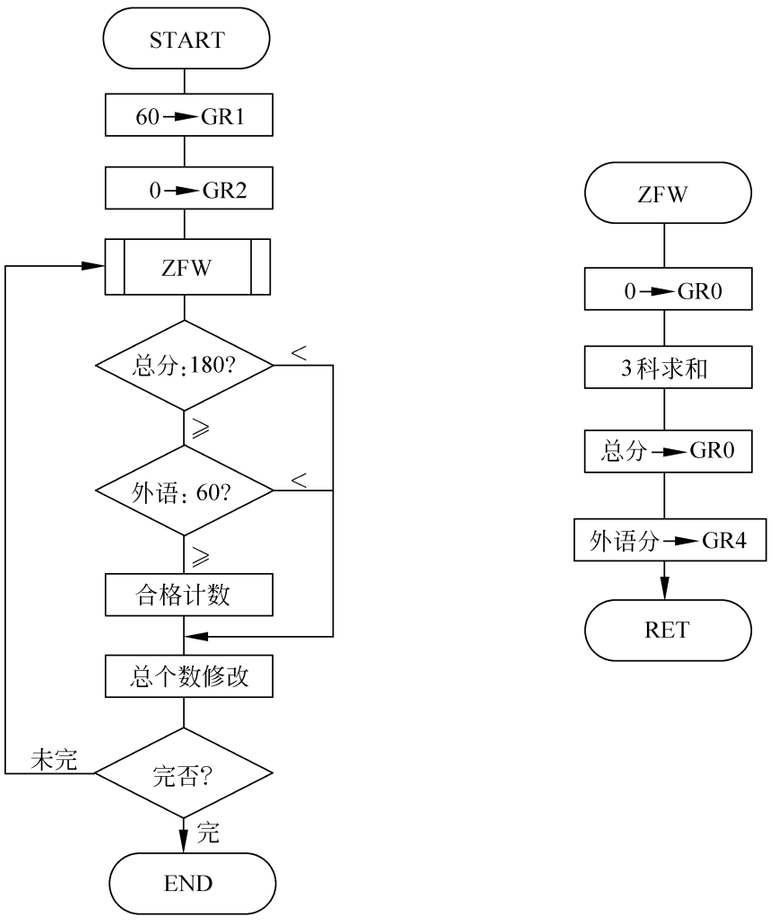


图 8.7　例 4 的程序流程图

【参考程序】

```
MTJ     START
        LEA     GR1,60
        LEA     GR2,0
JXZ     CALL    ZFW
        CPA     GR0,YBB
        JMI     XG              ;小于 180 转
        CPA     GR4,LSF
        JMI     XG              ;小于 60 转
        LEA     GR2,1,GR2       ;统计合格人数
XG      LEA     GR1,-1,GR1      ;总数修改
        JNZ     JXZ             ;非零转
        EXIT
ZFW     LEA     GR0,0           ;0→GR0 累加器
        ADD     GR0,SX,GR1      ;累加数学
        ADD     GR0,ZZ,GR1      ;累加政治
        ADD     GR0,WY,GR1      ;累加外语,总分→GR0
        LD      GR4,WY,GR1      ;外语→GR4
        RET
SX      DS      60
```

```
ZZ      DS      60
WY      DS      60
YBB     DC      180
LSF     DC      60
        END
```

8.2　CASL 指令功能及运用

8.2.1　CASL 指令系统

(1) 取数、存数、传送指令

指令名称		操作码	操作数	功　能	(FR)
① 取数	LoaD	LD	GR,adr[,XR]	(E)→GR	无
② 存数	STore	ST	GR,adr[,XR]	(GR)→E	无
③ 传送	Load Effective Address	LEA	GR,adr[,XR]	E→GR	有

(2) 算术运算及移位指令

指令名称		操作码	操作数	功　能	(FR)
④ 加法	ADD Arithmetic	ADD	GR,adr[,XR]	(GR)+(E)→GR	有
⑤ 减法	SUB tract Arithmetic	SUB	GR,adr[,XR]	(GR)-(E)→GR	有
⑥ 算术左移	Shift Left Arithmetic	SLA	GR,adr[,XR]	(GR)⟋E→GR	有
⑦ 算术右移	Shift Right Arithmetic	SRA	GR,adr[,XR]	(GR)E⟍→GR	有

(3) 逻辑运算及移位指令

指令名称		操作码	操作数	功能	(FR)
⑧ 逻辑乘	AND	AND	GR,adr[,XR]	(GR)∧(E)→GR	有
⑨ 逻辑加	OR	OR	GR,adr[,XR]	(GR)∨(E)→(GR)	有
⑩ 逻辑异或	Exclusive OR	EOR	GR,adr[,XR]	(GR)⊻(E)→GR	有
11 逻辑左移	Shift Left Logical	SLL	GR,adr[,XR]	(GR)⟋E→GR	有
12 逻辑右移	Shift Right Logical	SRL	GR,adr[,XR]	(GR)E⟍→GR	有

(4) 比较指令

指令名称		操作码	操作数	功能	(FR)
13 算术比较	Com Pare Arithmetic	CPA	GR,adr[,XR]	(GR)～(E)=(FR)	有
14 逻辑比较	Com Pare Logical	CPL	GR,adr[,XR]	(GR)～(E)=(FR)	有

(5) 转移指令

指令名称		操作码	操作数	功能	(FR)
15 无条件转	unconditional JuMP	JMP	adr[,XR]	无条件转 E	无
16 大于、等于(非负)转	Jump on Plus or Zero	JPZ	adr[,XR]	满足条件转 E	00,01
17 小于(负)转	Jump on MIus	JMI	adr[,XR]	满足条件转 E	10
18 等于(零)转	Jump on ZEro	JZE	adr[,XR]	满足条件转 E	01
19 不等于(非零)转	Jump on Non Zero	JNZ	adr[,XR]	满足条件转 E	00,10

(6) 栈操作指令

指令名称		操作码	操作数	功　能
0 进栈	PUSH effective address	PUSH	adr[,XR]	(GR4) - 1→GR4, E进栈
1 出栈	POP up	POP	GR	栈顶数据→GR, (GR4)+1→GR4

(7) 转子、返主指令

指令名称		操作码	操作数	功　能
2 转子	CALL subroutine	CALL	adr[,XR]	转向E, (PC)+2的地址进栈
3 返主	RETurn from subroutine	RET	空白	栈顶数据出栈→PC, 返主

(8) 伪指令及宏指令

指令名称		操作码	操作数	功　能
① 程序开头伪指令	START	START	[执行开始标号]	放在每一个程序开头
② 程序结尾伪指令	END	END	空白	放在每一个程序末尾
③ 定义常数伪指令	Define Constant	DC	常数	定义CASL程序中的常数
④ 定义单元伪指令	Define Storage	DS	单元个数	定义CASL程序中的单元
⑤ 输入宏指令	INput	IN	标号1,标号2	输入数据到输入缓冲区
⑥ 输出宏指令	OUTput	OUT	标号1,标号2	输出数据到输出缓冲区
⑦ 程序执行结束宏指令	EXIT	EXIT	空白	控制程序执行结束

8.2.2 指令在程序设计中的运用

(1) 向寄存器送常数

在CASL程序设计中,寄存器常用作累加器、计数器、地址寄存器、地址偏移量寄存器、比较常数寄存器和中间结果寄存器等。因此,不管是为寄存器设置初值,还是让寄存器保存中间结果,都需要向寄存器传送常数。向寄存器传送常数可以使用LEA,LD,PUSH,POP等指令,常用的方式有下列几种。

①
```
CS1     LEA     GR0,1024        ;1024 = 0400H →GR0
```

②
```
CS2     LD      GR0,C           ;0400 = 1024→GR0
        …
C       DC      # 0400
```

③
```
CS3     PUSH    1024            ;1024 进栈
        POP     GR0             ;1024 出栈→GR0
```

(2) 寄存器间的数据传送

使用LEA指令的间接形式,ST与LD指令的组合以及栈操作指令都可以完成寄存器之间的数据传送。

①
```
GS      LEA     GR0,0,GR1       ;(GR1)→GR0
```

②
```
GS2     ST      GR1,W           ;(GR1)→W
        LD      GR0,W           ;(W) = (GR1)→GR0
        …
W       DS      1
```

③
```
GS3     PUSH    0,GR1           ;(GR1)进栈
        POP     GR0             ;(GR1)出栈→GR0
```

(3) 寄存器间、单元间的数据互换

运用栈操作可以方便地互换两个寄存器的内容,使用LD和ST指令可以互换两个单元中的数据。

①
```
HHG     PUSH    0,GR1           ;先(GR1)进栈
        PUSH    0,GR2           ;后(GR2)进栈
        POP     GR1             ;(GR2)先出→GR1
        POP     GR2             ;(GR1)后出→GR2
```

②
```
HHD     LD      GR0,DY          ;(DY)→GR0
        LD      GR1,NC          ;(NC)→GR1
        ST      GR1,DY          ;(GR1) = (NC)→DY
        ST      GR0,NC          ;(GR0) = (DY)→NC
```

(4) 寄存器中、单元中加常数

①
```
GJC     LEA     GR1,60,GR1
```
;(GR1) + 60→GR1

②
```
GJS     ADD     GR1,LS
        …
LS      DC      60
```
;(GR1) + (LS) = (GR1) + 60→GR1

③
```
DJC     LD      GR1,DY
        LEA     GR1,60,GR1
        ST      GR1,DY
        …
DY      DS      1
```
;(DY)→GR1
;(GR1) + 60 = (DY) + 60→GR1
;(DY) + 60→DY

(5) 寄存器、单元中数的倍数

2 倍、4 倍、8 倍…

①
```
ZB2     SLA     GR1,1
```
;2×(GR1)→GR1

②
```
ZB4     SLA     GR2,2
```
;4×(GR2)→GR2

③
```
ZB8     SLA     GR3,3
```
;8×(GR3)→GR3

1/ 2、1/ 4、1/ 8…

④
```
F2      SRA     GR1,1
```
;(GR1)的 1/ 2

⑤
```
F4      SRA     GR2,2
```
;(GR2)的 1/ 4

⑥
```
F8      SRA     GR3,3
```
;(GR3)的 1/ 8

⑦ 5 倍
```
WB      LD      GR0,DY
        SLA     GR0,2
        ADD     GR0,DY
        …
DY      DC      12
```
;4 倍的(GR0)
;5 倍的(GR0)

⑧ 10 倍

```
SB      LD      GR0,DY
        SLA     GR0,2       ;(GR0)的4倍
        ADD     GR0,DY      ;(GR0)的5倍
        SLA     GR0,1       ;(GR0)的10倍
        …
DY      DC      12
```

⑨ 3/4(0.75)倍

```
        LD      GR1,DY
        SRA     GR1,1       ;1/2
        ST      GR1,WK      ;1/2→WK
        LD      GR1,DY
        SRA     GR1,2       ;1/4
        ADD     GR1,WK      ;1/4+1/2=3/4
        …
DY      DC      12
WK      DS      1
```

(6) 字符数字变数值数字

① 移位变换

```
YB      LD      GR2,DY      ;'6'
        SLL     GR2,12      ;去掉高3位
        SRL     GR2,12      ;复位为6
        …
DY      DC      '6'
```

② 逻辑尺截取

```
LJ    LD      GR2,DY      ;'6'
      AND     GR2,LC      ;取低 4 位为 6
      …
DY    DC      '6'
LC    DC      #000F
```

③ 减基数转换

```
JF    LD      GR2,DY      ;'6'为 0110110 = 54
      SUB     GR2,JS      ;54 - 48 = 6
      …
DY    DC      '6'
JS    DC      48
```

(7) 数值数字变字符数字

① 拼位变换

```
PW    LD      GR3,SZ      ;取 9
      OR      GR3,CS      ;拼上高位变为'9':0111001
      …
SZ    DC      9
CS    DC      #0030
```

② 加常数转换

```
JC    LD      GR3,SZ      ;取 8
      ADD     GR3,CS      ;8 + 48 = 56 = 70₈ = 0111000
      …
SZ    DC      8
CS    DC      48
```

(8) 循环控制

① 正计数循环累加

```
XHZ  LEA   GR1,0          ;计数器设初值 0→GR1
     LEA   GR0,0          ;累加器初值 0→GR0
JX   ADD   GR0,DY,GR1     ;累加第 1 个数…
     LEA   GR1,1,GR1      ;计数
     CPL   GR1,CS         ;次数比较
     JNZ   JX
     …
DY   DS    50            ;累加单元
CS   DC    50            ;比较常数
```

② 倒计数循环累加

```
XHD  LEA   GR1,49         ;计数器初值 49→GR1
     LEA   GR0,0          ;累加器初值 0→GR0
JX   ADD   GR0,DY,GR1     ;累加最后 1 个数…
     LEA   GR1, - 1,GR1   ;倒计数
     JPZ   JX
     …
DY   DS    50            ;累加单元
```

由以上两种循环控制的累加可以得出,循环需要计数器,累加需要累加器,且累加器初值为 0。循环要通过计数和比较来保证循环操作次数,且比较和转移要匹配才能确保正确的循环次数。

正计数循环与倒计数循环有所不同,第一个不同表现在计数器的初值设定上,正计数初值设为 0,倒计数的初值设定为循环次数减 1。由于计数的初值不同,导致两种循环产生了第二个不同,即计数方式不同。正计数用加 1 计数,倒计数用减 1 计数。在循环中,由于计数器又兼做变址寄存器,因此使得两种循环存在着第三个不同,这个不同表现在取数顺序上,见下列图示。

正计数循环取数		
(GR1)	地址	数据及取出顺序
0	DY	89 ①
1	DY + 1	66 ②
2	DY + 2	72 ③
3	DY + 3	86 ④
4	DY + 4	93 ⑤
5	DY + 5	81 ⑥
…	…	…
48	DY + 48	92 ⑨
49	DY + 49	88 ⑩
50	停止循环	

倒计数循环取数		
(GR1)	地址	数据及取出顺序
0	DY	89 ⑩
1	DY + 1	66 ⑨
2	DY + 2	72 ⑧
3	DY + 3	86 ⑦
4	DY + 4	93 ⑥
5	DY + 5	81 ⑤
…	…	…
48	DY + 48	92 ②
49	DY + 49	88 ①

(9) 分支控制

①

寄存器的内容按负、零、正分支		
FZ1	LEA	GR1,0,GR1
	JMI	ZA
	JZE	ZB
	JMP	ZC
	…	
ZA	A 处理	
ZB	B 处理	
ZC	C 处理	
	…	

分支流程图如图 8 .8 所示:

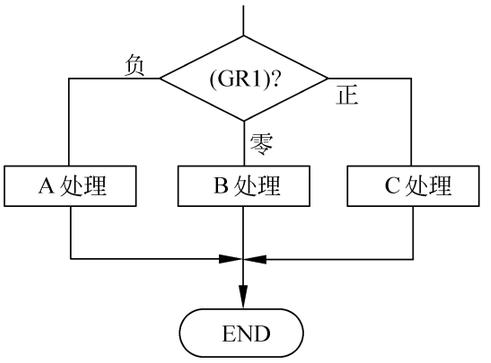


图 8.8　分支流程图

②

寄存器内容按 0、1、2 分支

```
FZ2     LD      GR2,DZ,GR1      ;(GR1)为 0、1、2,取(DZ+0)…
        JMP     0,GR2           ;按(DZ+0) = ADZ 转…
ADZ     A 处理
        JMP     WLT
BDZ     B 处理
        JMP     WLT
CDZ     C 处理
WLT     EXIT
DZ      DC      ADZ             ;转移入口地址 1
        DC      BDZ             ;转移入口地址 2
        DC      CDZ             ;转移入口地址 3
        END
```

③

单元内容按 1、2、3、4 分支			
FZ3	LD	GR1,DY	
	LEA	GR1, - 1,GR1	
	JZE	ZY1	;(DY) = 1,则(GR1) = 0 转
	LEA	GR1, - 1,GR1	
	JZE	ZY2	
	LEA	GR1, - 1,GR1	
	JZE	ZY3	
	LEA	GR1, - 1,GR1	
	JZE	ZY4	
	JMP	WL	
ZY1	…		
ZY2	…		
ZY3	…		
ZY4	…		
WL	EXIT		

8.3　程序设计训练

【问题 8-1】　对 A,B,C 3 个不相等的整数,按由大到小排序为 A > B > C。

【方法分析】　3 个数由大到小排序,可以用求最大值的办法。先在 3 个数中找到一个最大值,然后再在两个数中找出最大值,从而得到排序结果。为了节省比较次数,找极值普遍采用比较、置换的方法。下面用具体个例,分析该方法的操作办法。

假定 3 个数开始状态为

A	B	C
1	2	3

第一步是 A 和 B 进行比较,若 A > B 则数据不动;若 A < B,则 A,B 数据互换。在本例中 A = 1,B = 2,A < B,则 A,B;置换为,

A	B	C
2	1	3

第二步是 A 和 C 进行比较,由于 A < C,故 A,C 置换为,

A	B	C
3	1	2

第三步是 B 和 C 进行比较，由于 B < C，故 B，C 置换为，

A	B	C
3	2	1

经以上三步，A，B，C 3 个数便排列为 A > B > C 的形式了。

【程序流程图】

如图 8.9 所示。

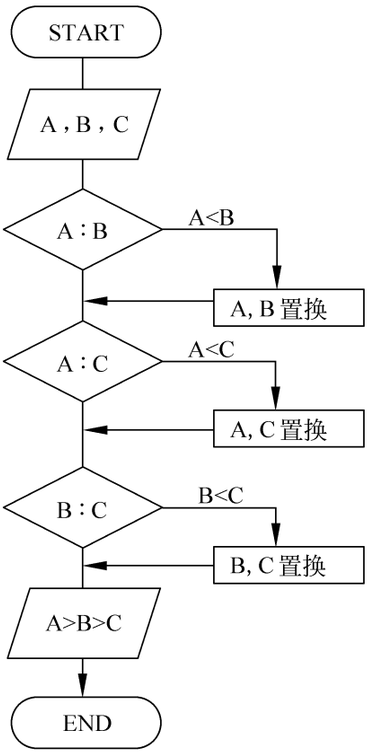


图 8.9 【问题 8-1】的程序流程图

【参考程序】

```
SPX     START
        LD      GR1,A       ;A→GR1
        LD      GR2,B       ;B→GR2
        LD      GR3,C       ;C→GR3
        ST      GR2,W       ;B→W 工作单元
        CPA     GR1,W       ;A:B
        JPZ     AYC         ;A≥B 转
        PUSH    0,GR1       ;A,B 置换
        PUSH    0,GR2
        POP     GR1
        POP     GR2
AYC     ST      GR3,W       ;C→W 工作单元
        CPA     GR1,W       ;A:C
        JPZ     BYC         ;A≥C 转
```

```
        PUSH    0,GR1       ;A,C 置换
        PUSH    0,GR3
        POP     GR1
        POP     GR3
BYC     ST      GR3,W       ;C→W 工作单元
        CPA     GR2,W       ;B:C
        JPZ     WL          ;B≥C 转结束
        PUSH    0,GR2       ;B,C 置换
        PUSH    0,GR3
        POP     GR2
        POP     GR3
WL      EXIT
A       DS      1
B       DS      1
C       DS      1
W       DS      1
        END
```

【结果评述】　执行本程序可将 3 个不相等的正整数 A,B,C 由大到小依次排列起来。在本程序中三次比较对应三次置换,因此可以把进行置换的程序段提炼出来变为子程序,使整个程序成为主—子结构的程序。

【问题 8-2】　已知二正整数 X、Y,求 X 与 Y 的乘积?

【方法分析】　二数相乘一种做法是把一个乘数作为计数器,按照计数的个数去累加另一个乘数。第二种做法是采取被乘数移位累加的方法,以 9×5 为例图示如下:

```
      1 0 0 1        ;被乘数
  ×   0 1 0 1        ;乘数
-------------
      1 0 0 1        ;乘数低位为 1,则加一次被乘数
    0 0 0 0          ;乘数次低位为 0,则不加被乘数
  1 0 0 1
0 0 0 0
-------------
  1 0 1 1 0 1 = 45
```

这种相乘的方法可归纳为,以乘数的最低位为基准,若低位为 1,则加一次被乘数,若低位为 0 则不加被乘数。将乘数右移 1 位得到一个新的低位,继续判断是 1 还是零,直到乘数右移为 0 停止。被乘数随着乘数的操作,每操作一次右移 1 位,从而形成不同的累加值,

第 1 次的累加值为:(被乘数)×2^0

第 2 次的累加值为:(被乘数)×2^1

第 3 次的累加值为:(被乘数)×2^2

依次类推。

【程序流程图】

如图 8 .10 所示。

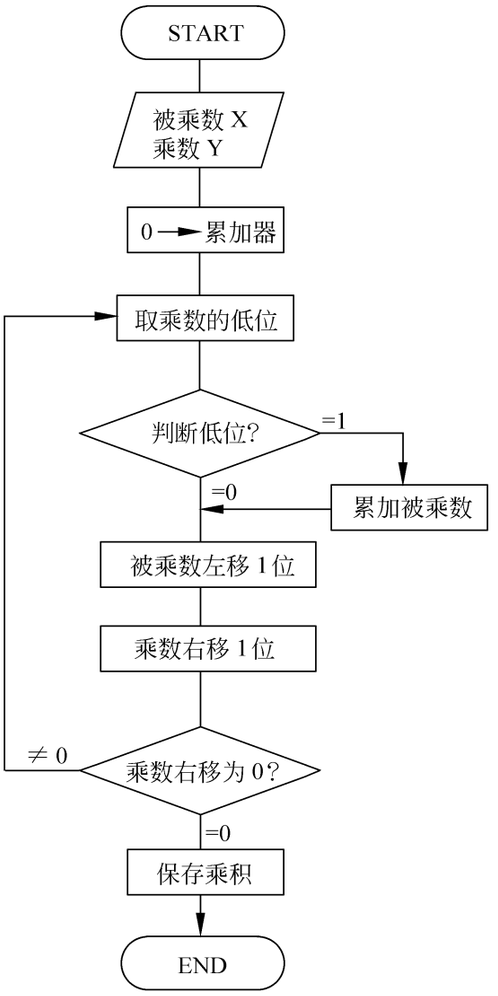


图 8 .10 【问题 8-2】程序流程图

【参考程序】

```
RSC     START
        LD      GR0,X       ;取被乘数 X→GR0
        ST      GR0,W       ;X→W
        LD      GR1,Y       ;取乘数 Y→GR1
        LEA     GR2,0       ;0→GR2 累加器
JX      LEA     GR3,1       ;1→GR3
        ST      GR1,Z       ;(GR1)→Z,即 Y→Z
        AND     GR3,Z       ;取 Y 的低位
        JZE     LZ          ;如 Y 的低位为 0 转 LZ,不加 X
        ADD     GR2,W       ;如 Y 的低位为 1,则加上个 X
LZ      LD      GR4,W       ;取 X
        SLA     GR4,1       ;将 X 左移 1 位
        ST      GR4,W       ;将左移 1 位的 X→W
        SRL     GR1,1       ;将乘数 Y 右移去掉低位,形成当前低位
        JNZ     JX          ;未完则继续,完则执行下一条
        ST      GR2,2       ;结果→Z
```

```
        EXIT
X       DS      1       ;被乘数单元
Y       DS      1       ;乘数单元
Z       DS      1       ;工作单元及结果单元
W       DS      1       ;工作单元
        END
```

【结果评述】　执行本程序,X,Y 的乘积保存在 GR2 累加器中,同时存于单元 Z 中。本题也可以设计为主—子结构的程序,用子程序专门承担被乘数左移,乘数右移,而在主程序中控制判断、累加。

【问题 8-3】　在以 TBL 为首地址的 20 个单元中存放着 20 个正整数,设计一个程序由大到小排序。

【方法分析】　对于多个数据的排序,最基本的方法是采用多次求极值。而求极值的方法是选定一个基准值作为极值,经与其他值比较、置换而得到极值,而多次求极值便得出排序数列。以 5 个数为例经 4 次求极值便得出排序结果。

第 1 次求极值,以 A 为基准值,与其他数比较、置换,

A　B　C　D　E

[1]　3　5　4　2

比　小则置换,

[3]　1　5

比　小则置换,

[5]　1　3　4

比　大则不置换,

[5]　1　3　4　2

比　大则不置换,

经 4 次比较、置换,得出第 1 个极大值在 A 中。

第 2 次求极值,以 B 为基准值,与 A 以外的值比较、置换,

A　B　C　D　E

5　[1]　3　4　2

比　小则置换,

5　[3]　1　4

比　小则置换,

5　[4]　1　3　2

比　大则不置换,

经 3 次比较、置换,得出第 2 个极大值在 B 中。

第 3 次求极值,以 C 为基准值,与 A,B 以外的值比较、置换,

A	B	C	D	E
5	4	1	3	2

比　　小则置换,

A	B	C	D	E
5	4	3	1	2

比　　大则不置换,

经 2 次比较置换,得出第 3 个极大值在 C 中。

第 4 次求极值,以 D 为基准值,与 A,B,C 以外的值比较、置换,

A	B	C	D	E
5	4	3	1	2

比　　小则置换,

5	4	3	2	1

;排序结果。

由以上操作过程可以得出,5 个数排序,经 4 次求极值便可产生排序结果。本题为 20 个数据,经 19 次求极值便可以得出排序结果。由以上求极值的过程还可以看出,做完一次求极值操作,基准值的地址与被比较值的取数地址都需要做相应的修改。

【参考程序】

```
PX    START
      LEA     GR1,0
QJZ   LEA     GR2,1,GR1
      LD      GR3,TBL,GR1     ;取基准值
JXB   CPA     GR3,TBL,GR2     ;基准值与被比较值比较
      JPZ     BZH             ;大于转
      LD      GR0,TBL,GR2     ;小于置换
      ST      GR0,TBL,GR1
      ST      GR3,TBL,GR2
BZH   LEA     GR2,1,GR2       ;修改被比较值取数地址
      CPA     GR2,C20         ;是否比较完
      JMI     JXB             ;未完转 JX 继续比
      ADD     GR1,C1          ;换基准值
      CPA     GR1,C19         ;是否做完求极值
      JMI     QJZ             ;未完转求极值
      EXIT                    ;求完所有次极值停止
TBL   DS      20
C1    DC      #0001
C19   DC      #0013
C20   DC      #0014
      END
```

【结果评述】　执行本程序可将以 TBL 为首地址的 20 个单元中的数由大到小排序。如果想实现由小到大排序,可将比较、置换的判据改为:如果基准值大于被比较值则置换;基准

值小于被比较值则不置换。

【问题 8-4】　评定 5 分制计分的考试成绩有优、良、中、差四级。5 分为优、4 分为良、3 分为中、2 分为差。判断单元中的成绩,并在成绩前加上等级标志,标志和分数压缩在同一单元。

若为 5 分,则处理为:

'A'	5

若为 4 分,则处理为:

'B'	4

若为 3 分,则处理为:

'C'	3

若为 2 分,则处理为:

'D'	2

【方法分析】　在通常情况下,由于本题对考试成绩有 4 种评定标准,而表示这 4 种标准有 4 种形式,因而采取 4 分支的分支程序结构是很自然的。我们从 4 分支程序设计开始,通过程序设计的实践来讨论本题的解决办法,并借助此实践来分析使用 CASL 指令进行程序设计的技术问题。

【程序流程图】

如图 8 .11 所示。

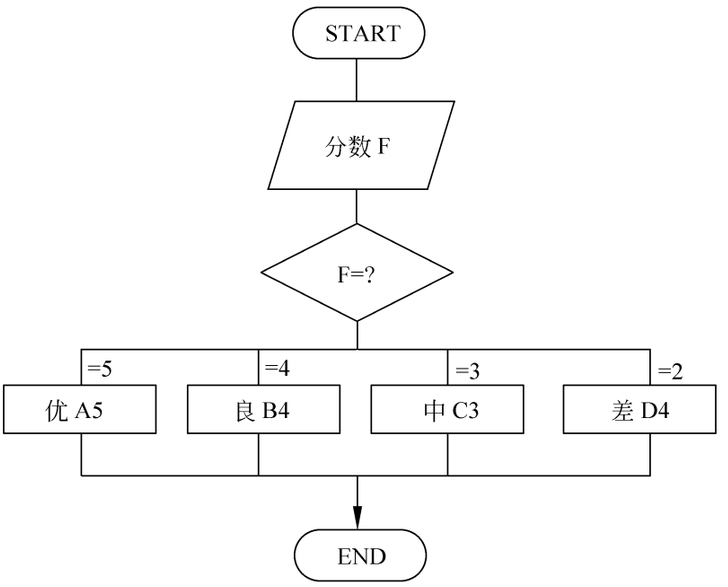


图 8 .11　【问题 8-4】的程序流程图

【参考程序】

```
PD1     START
        LD      GR1,FS          ;取分数→GR1
        CPA     GR1,C5          ;5?
        JZE     CLA             ;转处理优
        CPA     GR1,C4          ;4?
        JZE     CLB             ;转处理良
        CPA     GR1,C3          ;3?
        JZE     CLC             ;转处理中
CLD     LD      GR2,CD          ;取'D'
        SLL     GR2,8           ;左移 8 位
```

```
        OR      GR2,FS          ;D与2压缩在一起
        ST      GR2,FS          ;放回
        JMP     WL
CLC     LD      GR2,CC          ;取'C'
        SLL     GR2,8           ;左移8位
        OR      GR2,FS          ;C与3压缩在一起
        ST      GR2,FS          ;放回
        JMP     WL
CLB     LD      GR2,CB          ;取'B'
        SLL     GR2,8           ;左移8位
        OR      GR2,FS          ;B与4压缩在一起
        ST      GR2,FS          ;放回
        JMP     WL
CLA     LD      GR2,CA          ;取'A'
        SLL     GR2,8           ;左移8位
        OR      GR2,FS          ;A与5压缩在一起
        ST      GR2,FS          ;放回
        EXIT
ES      DS      1               ;5分制分数
CA      DC      'A'             ;优标志A
CB      DC      'B'             ;良标志B
CC      DC      'C'             ;中标志C
CD      DC      'D'             ;差标志D
        END
```

【结果评述】 执行本程序可将单元FS中的5分制分数加上优、良、中、差标志后放回原单元中。

从程序的完备性上说,应加入对错误分数判断的功能,当分数不是5,4,3,2分时予以处理,这样可使该程序具有排除错误及其处理的功能。

从程序的简练性上说,在本程序中CLD,CLC,CLB,CLA等4段程序基本上是功能相同的程序段,因此可以提炼为子程序,将本问题设计为主—子结构的程序。见下列程序。

【参考程序】

```
PD2     START
        ①  LD      GR1,C5          ;取常数5
        ②  SUB     GR1,FS          ;形成5-5=0,5-4=1,5-3=2,5-2=3
        ③  LD      GR1,CLK,GR1     ;在GR中形成转子入口地址
           CALL       0,GR1        ;5分转CLA,4分转CLB,3分转CLC,2分转CLD
           EXIT
CLA        LD      GR2,CA          ;取A
           JMP     YS
CLB        LD      GR2,CB          ;取B
           JMP     YS
CLC        LD      GR2,CC          ;取C
```

```
        JMP     YS
CLD     LD      GR2,CD      ;取 D
YS      SLL     GR2,8       ;标志左移 8 位
        OR      GR2,FS      ;拼上分数
        ST      GR2,FS      ;放回分数单元
        RET
C5      DC      5
FS      DS      1
CLK     DC      CLA
        DC      CLB
        DC      CLC
        DC      CLD
CA      DC      'A'
CB      DC      'B'
CC      DC      'C'
CD      DC      'D'
        END
```

在本程序中把程序 1 中的 4 段压缩处理改为一段子程序。程序中,先由指令①,②形成地址偏移量 0,1,2,3 放在 GR1 中,然后由指令③形成取以 CLK 为首地址的单元内容存于 GR1 中,

当成绩为 5 分时,地址偏移量为 0,取 CLK + 0 单元内容为 CLA;

当成绩为 4 分时,地址偏移量为 1,取 CLK + 1 单元内容为 CLB;

当成绩为 3 分时,地址偏移量为 2,取 CLK + 2 单元内容为 CLC;

当成绩为 2 分时,地址偏移量为 3,取 CLK + 3 单元内容为 CLD;

当在指令④以 GR1 的内容为转子入口地址时,正好形成 4 个不同的转子入口地址,CLA,CLB,CLC,CLD,从而完成 4 种不同的处理。

在本题中,由于优、良、中、差的代号 A,B,C,D 的内码是已知的,考试成绩也只有 5,4,3,2 四种。因此成绩等级标准和分数的搭配可以设定为固定的常数,

① 考试成绩为 5 分,'A'与 5 压缩在同一单元时,

'A'	5
‖	
01000001	00000101

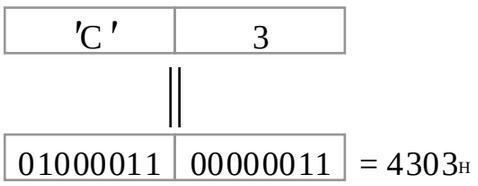
$= 4105_H$

② 考试成绩为 4 分,'B'与 4 压缩在同一单元时,

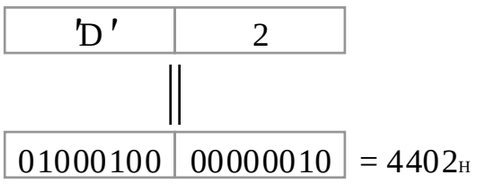
'B'	4
‖	
01000010	00000100

$= 4204_H$

③ 考试成绩为 3 分,'C'与 3 压缩在同一单元　,



④ 考试成绩为 2 分，'D'与 2 压缩在同一单元时，



若在程序中存放 4 个压缩常数，不仅可以把压缩处理的程序段大大简化，而且可以从根本上改变程序的结构。

【参考程序】

```
PD3     START
        LD      GR1,CS          ;取 5→GR1
        SUB     GR1,FS          ;5 - 5;5 - 4;5 - 3;5 - 2→GR1
        LD      GR1,YSS,GR1     ;取压缩常数 A5;B4;C3;D2
        ST      GR1,FS          ;放回原分数单元
        EXIT
FS      DS      1               ;考试成绩单元
CS      DC      5               ;常数 5
YSS     DC      #4105           ;A5
        DC      #4204           ;B4
        DC      #4303           ;C3
        DC      #4402           ;D2
        END
```

本程序为顺序结构而不是分支结构。由本例可以说明一个道理，即程序结构是可以转化的，在一定条件下，复杂的程序可以简化为简单程序；分支程序可以转化为主—子结构的程序，有时还可以转化为顺序程序。程序设计很像人走路，不加思考地或者固执地“一条道走到黑”不一定是条捷径。

习 题 8

1．阅读程序说出其功能。

```
(1)  C      START
            IN      B1,B2
            LEA     GR1,3
            CPA     GR1,B2
            JZE     WL
            OUT     ER,C5
     WL     EXIT
```

```
    B1    DS    80
    B2    DS    1
    ER    DC    'ERROR'
    C5    DC    5
          END
```

(2)
```
    C     START
      ①   LD    GR1,DY
      ②   LEA   GR1,10,GR1
      ③   ST    GR1,DY
      ④   ADD   GR2,CS
      ⑤   PUSH  10,GR3
      ⑥   POP   GR3
          EXIT
    DY    DS    1
    CS    DC    10
          END
```

(3)
```
    FH    START
          EOR   GR1,FF
          LEA   GR1,1,GR1
          EXIT
    FF    DC    #FFFF
          END
```

(4)
```
    DYF   START
          LEA   GR1,-1
          EOR   GR1,DY
          LEA   GR1,1,GR1
          ST    GR1,DY
          EXIT
    DY    DS    1
          END
```

(5)
```
    ZLJ   START
          LEA   GR1,0
          LEA   GR0,0
    JX    ADD   GR0,ZDY,GR1
          LEA   GR1,1,GR1
          CPL   GR1,CS
          JNZ   JX
          EXIT
    CS    DC    50
    ZDY   DS    50
```

```
        END
```

(6)
```
FLJ     START
        LEA     GR1,49
        LEA     GR0,0
JX      ADD     GR0,FDY,GR1
        LEA     GR1,-1,GR1
        JPZ     JX
        EXIT
FDY     DS      50
        END
```

2．按要求填空完善程序，并加上注释。

(1) 下列程序是主—子结构的求和程序，执行结果在 GR0 中。

```
QH1     START
        LEA     GR1,DATA
        [          ]
        CALL    ZK
        EXIT
ZS      DC      6
DATA    C       87
        DC      92
        DC      69
        DC      84
        DC      73
        DC      86
ZK      LEA     GR0,0
        LEA     GR2,[    ]
        JMI     FH
        ADD     [          ]
        LEA     GR1,1,GR1
        JMP
FH      RET
        END
```

(2) 下面是对 30 个单元中的数据求总和的程序，执行结果保存在单元 ZH 中。

```
ZH2     START
        [          ]
        LEA     GR2,0
        CPA     GR2,ZS
        [  ]    BC
JX      ADD     [          ]
```

```
        LEA     GR2,1,GR2
        [          ]
        JMI     JX
BC      ST      GR1,ZH
        EXIT
ZS      DS      1
ZH      DS      1
DY      DS      30
        END
```

3．程序设计。

(1) 已知在 26 个单元中倒序存放着英文字母如下所示，

单元地址	ZM	+1	+2	+3	+4	+5	+6	+7	…	+19	+20	+21	+22	+23	+24	+25
	Z	Y	X	W	V	U	T	S	…	G	F	E	D	C	B	A

设计一个程序，用栈操作将排列顺序正过来，如下所示，

单元地址	ZM															+25
	A	B	C	D	E	F	G	…	S	T	U	V	W	X	Y	Z

(2) 设计一个评分程序，对歌曲大赛参赛者计算总分及平均分。已知有 10 个评委为参赛者打分，要求去掉一个最高分，去掉一个最低分，计算出总分放在 ZF 单元，平均分保留在 PJ 单元中。

(3) 编一个程序，形成裴波那契数列：

1 1 2 3 5 8 13 21……

第 9 章

程序设计例题及分析

9.1 自然数的运算与操作

9.1.1 数列的形成 1

1. 设计说明

(1) 设计一个程序,形成数列 2,3,5,9,17,33,65,129…前 8 项,并计算出前 8 项之和。

(2) 所形成的数列存放在以 SL 为首地址的 8 个单元中,前 8 项之和存于单元 H。

2. 参考程序

```
SLXC    START
        LEA     GR2,2           ;数列的第 1 项 2→GR2
        ST      GR2,SL          ;存入第 1 项
        LEA     GR3,0,GR2       ;累加器 GR3 的初值为 2
        LEA     GR1,1           ;变址、送初值 1→GR1
JX      SLA     GR2,1           ;乘 2
        LEA     GR2,-1,GR2      ;减 1
        ST      GR2,SL,GR1      ;存入后项
        ST      GR2,W           ;后项→W→GR2
        ADD     GR3,W           ;求和→GR3
        LEA     GR1,1,GR1       ;修改
        CPA     GR1,C7          ;后 7 项是否完?
        JNZ     JX              ;未完继续
        ST      GR3,H           ;8 项之和→H
        EXIT
SL      DS      8
C7      DC      7
H       DS      1
        END
```

3. 方法概要

(1) 数列 2,3,5,9…,前项 S1 与后项 S2 的关系为:(S1 - 0.5)×2 = S2。

(2) 由于在 CASL 中小数运算不方便,所以把前项与后项的关系变换为:

$$(S1 - 0.5) \times 2 = 2 \times S1 - 1$$

根据该式,从第 1 项的开始,依次算出后项,并求出前 8 项的累加和。

9.1.2　数列的形成 2

1. 设计说明

(1) 设计一个程序,形成数列 2,3,4,6,9,13,19,28,41,60,88……的前 11 项。

(2) 所形成的数列存放在以 SL 为首地址的 11 个单元中。

2. 参考程序

```
SL2     START
        LEA     GR1,SL          ;数列首地址→GR1
        LEA     GR2,2
        ST      GR2,S1          ;第 1 项→S1
        ST      GR2,0,GR1       ;第 1 项 2→SL
        LEA     GR2,1,GR2
        ST      GR2,S2          ;第 2 项→S2
        ST      GR2,1,GR1       ;第 2 项 3→SL + 1
        LEA     GR2,1,GR2
        ST      GR2,S3          ;第 3 项→S3
        ST      GR2,2,GR1       ;第 3 项 4→SL + 2
        LEA     GR3,3           ;变址初值 3→GR3
JX      ADD     GR2,S1          ;第 3 项与第 1 项之和形成下一项
        ST      GR2,SL,GR3      ;存入后项
        ST      GR2,S4
        LD      GR1,S2
        ST      GR1,S1          ;置换 S1
        LD      GR1,S3
        ST      GR1,S2          ;置换 S2
        LD      GR1,S4
        ST      GR1,S3          ;置换 S3
        LEA     GR3,1,GR3       ;修改
        CPA     GR3,C11         ;是否完成 11 项
        JNZ     JX              ;未完转
        EXIT
SL      DS      11
C11     DC      11
        END
```

3. 方法概要

(1) 本数列的前 3 项直接由常数 2,3,4 存入数列 SL 的前 3 个单元。

(2) 数列从第 4 项开始,就有了固定的规律。我们把第 4 项称为 S4,把其前的项依次称为 S3,S2,S1,则有,

①	②	③	④
S1	S2	S3	S4 ……
2	3	4	6 ……

S4 = S1 + S3

根据上式可以形成从第 4 项开始的所有项。在形成过程中,每产生一个后项,通过项间置换,为下一个后项准备好相邻的前 3 项。

9.1.3 最大公约数

1.设计说明

(1) 单元 M,N 中均为正整数,求出它们的最大公约数存放的 D 单元中。

(2) 求两数的最大公约数的一种快捷的方法是互除法,假定 M = 15, N = 20,则互除法的操作如下:

M	D	N	互除操作	商	余数	备注
15	/	20	M÷N,即 15÷20	0	20	未除尽
	/		N÷M,即 20÷15	1	5	未除尽
	5		余数相除,20÷5	4	0	除尽

当互除除尽时,此时的除数则为两数的最大公约数。

2.参考程序

```
GUS     START
        LD      GR1,M
        ST      GR1,WM
        LD      GR2,N
        ST      GR2,WN
JX      CPA     GR1,WN
        JZE     WL
        JMI     MXN
        SUB     GR1,WN   ;M - N→GR1
        ST      GR1,WM   ;(GR1)→WM
        JMP     JX
MXN     SUB     GR2,WM   ;N - M
        ST      GR2,WN
        JMP     JX
WL      ST      GR1,D
        EXIT
M       DS      1
N       DS      1
D       DS      1
```

```
WM      DS    1
WN      DS    1
        END
```

3.方法概要

(1) 由于在 CASL 中没有除法指令,故采取连减的方法。

(2) 先确定两数的大数,并由大数减小数,将所得差代替大数。再由大数减小数,将所得差代替当前的大数。直到两数相等为止,此时便得到两数的最大公约数。仍以 M = 15, N = 20 为例,具体操作如下:

M	N	操 作 方 法
15	20	N > M,则 N - M = 20 - 15 = 5,将 5 视为 N
15	5	M > N,则 M - N = 15 - 5 = 10,将 10 视为 M
10	5	M > N,则 M - N = 10 - 5 = 5,将 5 视为 M
5	5	M = N,则 5 为两数的最大公约数

9.1.4　求和

1.设计要求

(1) 求 N 个自然数之和,ZH = $\sum_{i=1}^{n}$ = N_i。

(2) 设计为主—子结构,且用递归的方法。

2.程序流程图

如图 9.1 所示。

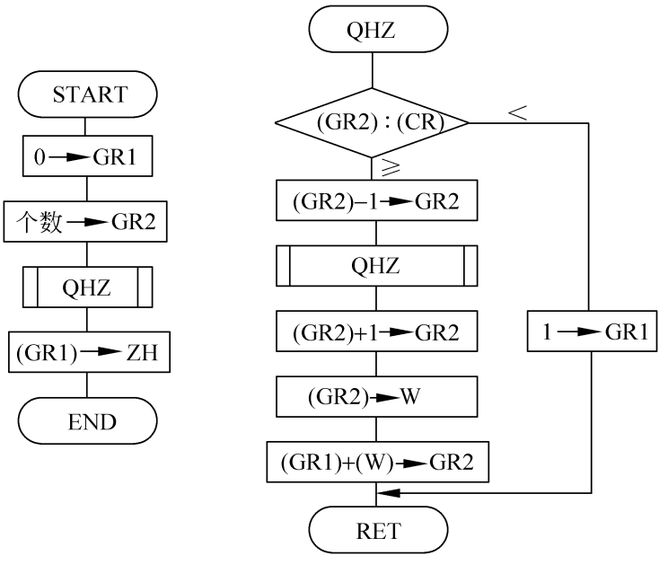


图 9.1　递归求和程序流程图

3.参考程序

```
ZQH     START
```

```
        LEA     GR1,0           ;0→GR1
        LD      GR2,GS          ;个数→GR2
        CALL    QHZ             ;转子
        ST      GR1,ZH          ;和保留在ZH中
        EXIT
QHZ     CPA     GR2,CR          ;(GR2):2
        JMI     KS              ;个数小于2转
        LEA     GR2,-1,GR2      ;(GR2-1)→GR2
        CALL    QHZ
        LEA     GR2,1,GR2
        ST      GR2,W
        ADD     GR1,W           ;1+2;1+2+3;1+2+3+4;1+2+3+4+5+…
        JMP     FH
KS      LEA     GR1,1
FH      RET
GS      DS      1
ZH      DS      1
CR      DC      2
W       DS      1
        END
```

4.方法概要

(1) 递归的方法在程序设计中可以简单地说成是“自身调用自身”的方法,因此从程序结构上说是一种多重转子的程序。

(2) 以计算 $\sum_{i=1}^{5} N_i$ 为例,

① $\sum_{i=1}^{5} N_i$ 可以变为 $5+\sum_{i=1}^{4} N_i$,故首先需要计算 $\sum_{i=1}^{4} N_i$;

② $\sum_{i=1}^{4} N_i$ 可以变为 $4+\sum_{i=1}^{3} N_i$,故首先需要计算 $\sum_{i=1}^{3} N_i$;

③ $\sum_{i=1}^{3} N_i$ 可以变为 $3+\sum_{i=1}^{2} N_i$,故首先需要计算 $\sum_{i=1}^{2} N_i$;

④ $\sum_{i=1}^{2} N_i$ 可以变为 $2+\sum_{i=1}^{1} N_i$,故首先需要计算 $\sum_{i=1}^{1} N_i$。

从而得出从 N=1 开始,然后求 2+1,再求 3+(2+1),继而求 4+(3+2+1),最后计算 5+(4+3+2+1)便得出结果。

9.1.5 角谷猜想的验证

1.设计说明

(1) 任意一个自然数,如果它是个偶数则用 2 除之,如果它是个奇数则乘 3 加 1。这样操作不仅每次可以得到一个新的自然数,而且最终可以得到的结果为 1。这就是角谷猜想。

(2) 任意的自然数已设定在单元 ZRS 中,编一个程序对角谷猜想进行验证,并将最终结果存入 ZRS 单元。

2.参考程序

```
JGCX    START
JX      LD    GR1,ZRS       ;取出自然数→GR1
        LEA   GR2,0,GR1     ;自然数→GR2
        AND   GR2,C1        ;最后一位→GR2
        JZE   CR            ;偶数转,奇数下
        SLA   GR1,1         ;乘 2
        ADD   GR1,ZRS       ;加本身,即乘 3
        LEA   GR1,1,GR1     ;加 1
        ST    GR1,ZRS       ;保存结果
        JMP   PY
CR      SRA   GR1,1         ;除 2
        ST    GR1,ZRS       ;保存结果
PY      CPA   GR1,C1        ;是否为 1
        JNE   JX            ;不为 1 继续操作
        EXIT
ZRS     DS    1
C1      DC    1
        END
```

3.方法概要

(1) 实现本程序的主要环节是判断自然数为奇数还是偶数。判断的方法是检验最低位,若最低位为 0,则自然数为偶数;若最低位为 1,则自然数为奇数。

(2) 对偶数处理是用 2 除,借助右移 1 位可实现。对奇数处理是乘 3 加 1,实现乘 3 可用先乘 2 再加本身的方法来完成。

(3) 假定自然数为 6,其操作过程及最终结果如下:

[6]:除 2→[3]:乘 3 加 1→[10]:除 2→[5]:乘 3 加 1→[16]:除 2→[8]:除 2→[4]:除 2→[2]:除 2→[1]。

不难看出,角谷猜想是正确的。

9.2 数制转换

9.2.1 十进制数转换成二进制数

1.设计说明

(1) 由外部设备输入的 5 位十进制正整数,存放在以 SJ 为首地址的 5 个内存单元中。

(2) 设计一个数制转换程序将十进制数转换成二进制数存放在 RJ 单元中。

(3) 在 CASL 中进行十进制到二进制(以下简写为 10→2)的转换所需要完成的工作如下:

2.参考程序

```
SZR     START
        LEA     GR1,4           ;将 ASCII→BCD
JX      LD      GR0,SJ,GR1      ;取 ASCII
        SLL     GR0,12          ;去掉前 12 位,变为 BCD
        SRL     GR0,12          ;4 位 BCD 移到低 4 位
        ST      GR0,ST,GR1      ;放回原处
        LEA     GR1, -1,GR1     ;修改
        JPZ     JX
        LEA     GR1,0
        ST      GR1,RJ
CL      LD      GR0,SJ,GR1      ;取一位十进制数,即 4 位 BCD
        ADD     GR0,RJ
        ST      GR0,RJ
        SLL     GR0,2           ;4 倍
        ADD     GR0,RJ          ;5 倍
        SLL     GR0,1           ;10 倍,即乘 10
        ST      GR0,RJ
        LEA     GR1,1,GR1       ;修改
        CPA     GR1,SV          ;是否处理完 4 位
        JMI     CL              ;未完继续
        LD      GR0,SJ,GR1      ;取最低位
        ADD     GR0,RJ          ;加上前 4 位
        ST      GR0,RJ          ;存二进制数
        EXIT
SJ      DS      5
RJ      DS      1
SV      DC      4
```

3.方法概要

(1) 由于输入的 5 位十进制数是 5 个字符所对应的 ASCII,所以第一步借助移位把 ASCII 变为 BCD。

(2) 用 BCD 表示的十进制数,与二进制数有一种对应关系,例如,十进制数的 32,用

BCD 表示为 00110010,若用 10 乘上高 4 位的 BCD,再加上低 4 位的 BCD,就可以得到 32 所对应的二进制数,

```
    0 0 1 1      ;高 4 位的 BCD。
×   1 0 1 0      ;乘 10。
-----------
    0 0 0 0
  0 0 1 1
0 0 0 0
0 0 1 1
-----------
  1 1 1 1 0
+   0 0 1 0      ;低 4 位的 BCD。
-----------
1 0 0 0 0 0      ;32 对应的二进制数。
```

(3) 如果十进制数是个 3 位数,则可以用 100 乘高 4 位的 BCD,用 10 乘中间 4 位的 BCD,然后相加再加上低 4 的 BCD,也会得到十进制数所对应的二进制数。例如十进制数的 128,其 BCD 为:000100101000,则有,

$(0001)\times100_{10} + (0010)\times10_{10} + 1000$

$= (0001)\times1100100 + (0010)\times1010 + 1000$

$= 1100100 + 10100 + 1000$

$= 10000000$

可以推断,5 位的(用 BCD 表示的)十进制数,可以从高 4 位的 BCD 开始,依次乘 10000,1000,100,10,1,然后相加就是 5 位十进制数所对应的二进制数。

(4) 为了运算上的方便,我们可以把 5 位十进制数转换二进制的计算式进行改写。令 B5,B4,B3,B2,B1 为由高位到低位的 5 个 BCD,则有,

$B5\times10000 + B4\times1000 + B3\times100 + B2\times10 + B1\times1$

$= B5\times10^4 + B4\times10^3 + B3\times10^2 + B2\times10^1 + B1$

$= (((B5\times10 + B4)\times10 + B3)\times10 + B2)\times10 + B1$

这个计算式就是我们得出的 5 位用 BCD 码表示的十进制数转换成二进制数的计算式。该计算式的口诀为:“高位不断乘 10 加低位,再乘 10 再加低位,直加到最后一个低位为止”。本例就是按这个运算法则设计出的转换程序。

9.2.2　二进制数转换成十进制数

1.设计说明

(1) 已知单元 RJS 存放着一个二进制的正整数,其值大小在 0～32767 的范围内。

(2) 将该数转换为 5 位的十进制数,存放在以 SJS 为首地址的 5 个单元中。例如,

2.参考程序

```
RZS     START
        LD      GR1,RJS         ;被转换的二进制数→GR1
        LEA     GR2,0           ;变址初值 0→GR2
HW      LE2     GR3,0           ;次数计数器初值 0→GR3
JCS     SUB     GR1,SS,GR2      ;二进制数相当十进制数多少?
```

```
        CPA     GR1,L           ;二进制数与 0 比较
        JMI     JD              ;小于 0 表示不够减
        LEA     GR3,1,GR3       ;转换计数
        JMP     JCS             ;继续减常数
JD      ST      GR3,SJS,GR2     ;保存 1 位十进制数
        ADD     GR1,SS,GR2      ;恢复多减的值
        LEA     GR2,1,GR2       ;修改换常数及换位
        CPA     GR2,WV          ;是否已转换了 5 位
        JMI     HW              ;未完换位继续
        EXIT
RJS     DS      1               ;被转换的二进制数
SJS     DS      5               ;5 位十进制数单元
SS      DC      #2710           ;十进制数 10000
        DC      #03E8           ;十进制数 1000
        DC      #0064           ;十进制数 100
        DC      #000A           ;十进制数 10
        DC      #0001           ;十进制数 1
L       DC      0
WV      DC      5
        END
```

3.方法概要

(1) 由于在 CASL 中一个单元表示有符号数最大值为 32767,表示无符号数为 65535。因此,二进制到十进制转换的十进制数最大为一个 5 位的数。

(2) 二进制到十进制转换采取“减已知常数计次数”的方法,先从二进制数中减去 10000,然后检验二进制数是否够减,如够则再减 10000,直减到不够减为止,记录下够减的次数,则得到十进制数的万位。接下来从减 10000 剩下的二进制数中减 1000,也记下

够减的次数,便得到十进制数的千位。以后减 100、减 10、减 1,均将够减的次数保留起来,便得到十进制数的百位、十位、个位。例如:

9.2.3　二进制数转换成十六进制数

1.设计说明

(1) 二进制数在内存单元 RDY 中。

(2) 将二进制数转换为 4 位的十六进制数放在以 SLS 为首地址的 4 个内存单元中。设计要求如下:

2.参考程序

```
RZSL    START
        LEA     GR1,0           ;十六进制数存放变址寄存器初值
        LEA     GR2,12          ;移位常数
CL      LD      GR3,RDY         ;取二进制数
        SRL     GR3,0,GR2       ;将被处理的 4 位移至低 4 位
        AND     GR3,D4          ;取 4 位
        CPA     GR3,SA          ;是否是 A～F
        JMI     SZ              ;转处理 0～9
        LEA     GR3,7,GR3       ;变换为 A～F,多加 7
SZ      LEA     GR3,48,GR3      ;变换为 0～9 加 48
        ST      GR3,SLS,GR1     ;存放 1 位十六进制数
```

```
        LEA     GR1,1,GR1     ;修改存数变址寄存器
        LEA     GR2,-4,GR2    ;修改移位常数
        JPZ     CL            ;未处理完4次转
        EXIT
D4      DC      #000F         ;取低4位逻辑常数
SA      DC      10            ;常数10,区别0～9与A～F
RDY     DS      1             ;二进制数存放单元
SLS     DS      1             ;十六进制数存放单元
        END
```

3.方法概要

(1) 在CASL中,数只要进入计算机就变成了二进制的形式。由于字符进入计算机是ASCII,所以数在计算机中只有两种可能,一是真正的二进制数,另一种是字符的信息代码。

(2) 本题实际上是将4位二进制数转换成字符所对应的ASCII。例如:

1010→A→1000001='A'

0001→1→0110001='1'

显然,4位二进制数0001转换为0110001,只需加上班0030_H即可,

```
      0 0 0 1     ;4位二进制数,其值为1
+ 0 1 1 0 0 0 0   ;转换常数0030
-----------------
  0 1 1 0 0 0 0   ;'1'的7位ASCII
```

对于4位二进制数,其值在10以上时,例如1010,转换关系为,

```
      1 0 1 0     ;4位二进制数,其值为10
+ 0 1 1 0 0 0 0   ;转换常数
-----------------
  0 1 1 1 0 1 0   ;为':'的ASCII
+       1 1 1     ;转换常数0007
-----------------
  1 0 0 0 0 0 1   ;'A'的7位ASCII
```

(3) 由以上分析不难得出转换方法如下:

① 4位二进制数的值为0～9,转换时加0030_H。

② 4位二进制数的值为A～F,转换时加0030_H和0007_H,即加上0037_H。

9.2.4 十六进制数转换成二进制数

1.设计说明

(1) 十六进制数有16个基数字,在计算机中这16个基数字用0～9,A～F表示。十六进制数与二进制数有直接对应关系,十六进制数的每1位直接对应4位二进制数。

(2) 现有输入到内存单元的一个4位的十六进制数,该数存在以SL为首地址的4个

内存单元中,设计一个数制转换程序完成转换,并将转换结果保留在 RJ 单元　。

2.参考程序

```
SLDR    START
        LEA     GR1,0           ;工作及结果寄存器初值 0→GR1
        LEA     GR2,0           ;计数兼变址初值 0→GR2
BH      LD      GR3,SL,GR2      ;取十六进制数的 1 位
        CPL     GR3,SA          ;基数字是 0～9 还是 A～F
        JPZ     AF              ;转处理 A～F
        SUB     GR3,SJ          ;0～9 减 0030H
        JMP     GT
AF      SUB     GR3,SA          ;A～F 减 0041H
        LEA     GR3,10,GR3      ;加 10
GT      ST      GR3,WK          ;十六进制数 1 位变为 4 位二进制数结果
        SLL     GR1,4           ;为低 4 位留出位置(第 1 次为空移)
        OR      GR1,WK          ;拼成二进制数
        LEA     GR2,1,GR2       ;修改计数及变址
        CPA     GR2,CS          ;是否处理完 4 位
        JMI     BH              ;未完转
        ST      GR1,RJ          ;保存转换后的二进制数
        EXIT
CS      DC      4
SJ      DC      #0030
SA      DC      #0041
WK      DS      1
SL      DS      4
RJ      DS      1
        END
```

3.方法概要

(1) 在通常情况下,十六进制数换成二进制数很简单,只需将十六进制数的每一位拆成 4 位就转换过去了。但在 CASL 中就没有那么简单了。其原因是输入的十六进制数只是形式上的表示,而不是真正的十六进制数。例如:

① 4 位的十六进制数,

1 2 3 4H　　;形式上的十六进制数。

② 入到内存单元后变为,

'1'
'2'
'3'
'4'

;每一位占一个单元,这仍然是形式上的十六进制数。

③ 十六进制数 1234_H 在计算机中按照 CASL 的规定是'1'、'2'、'3'、'4'的 ASCII,数的每 1 位占一个 16 位长的单元,且在单元的低 7 位,即 9～15 位,形式如下:

'1'			0	1	1	0	0	0	1
'2'			0	1	1	0	0	1	0
'3'			0	1	1	0	0	1	1
'4'			0	1	1	0	1	0	0

$= 0031_H$

$= 0032_H$

$= 0033_H$

$= 0034_H$

(2) 十六进制数 1234_H 按 1 位变 4 位的方法转换成二进制数应该得: 0001001000110100_2,而我们的问题是要把,

4 个单元					一个单元
0031	0032	0033	0034	转换为	0001001000110100

显然,实现这个转换必须弄清 0031 与 0001 的关系,0032 与 0010 的关系……。此外,如果在十六进制数中含有 A～F 时,则要解决'A'变 1010,'B'变 1011,'F'变 1111 的问题。

(3) 由以上分析可以得出解题的方法如下:

① 如果十六进制数的 1 位是 0～9 的基数字,则减去 0030_H。

② 如果十六进制数的 1 位是 A～F 的基数字,则减去 0041_H 后再加 1010。

③ 将 4 位十六进制数的每 1 位都进行转换后,再依次拼成一个二进制数。

9.3 四则运算

9.3.1 倍数运算

1.设计说明

(1) 已知在 GR1 中存放着一个数 x,其值的范围为 $-3200 \leq x \leq 3200$,设计一个程序,对其进行倍数运算。

(2) 若 GR2 中为 0,计算 $5x$ = ?

若 GR2 中为 1,计算 $10x$ = ?

若 GR2 中为 2,计算 $0.75x$ = ?

若 GR2 中为 3,计算 $0.375x$ = ?

计算结果存放在 JG 单元中。

2.参考程序

```
JSBS    START
        LD      GR2,RK,GR2      ;形成入口地址
        JMP     0,GR2           ;按(GR2)转各段入口
WB      LEA     GR3,0,GR1       ;x→GR3
```

```
        ST      GR3,JG          ;x→JG
        SLA     GR3,2           ;4x
        ADD     GR3,JG
        ST      GR3,JG          ;5x→JG
        JMP     WL
SB      LEA     GR3,0,GR1       ;x→GR3
        ST      GR3,JG
        SLA     GR3,2           ;4x
        ADD     GR3,JG          ;5x
        SLA     GR3,1           ;10x
        ST      GR3,JG          ;10x→JG
        JMP     WL
D75     LEA     GR3,0,GR1       ;x→GR3
        SRA     GR3,1           ;1/2x
        ST      GR3,JG
        SRA     GR3,1           ;1/4x
        ADD     GR3,JG          ;(1/4+1/2)x
        ST      GR3,JG          ;3/4x=0.75x→JG
        JMP     WL
D375    LEA     GR3,0,GR1       ;x→GR3
        SRA     GR3,2           ;1/4x
        ST      GR3,JG
        SRA     GR3,1           ;1/8x
        ADD     GR3,JG          ;(1/8+1/4)x
        ST      GR3,JG          ;3/8x=0.375x→JG
        EXIT
RK      DC      WB              ;5x 入口
        DC      SB              ;10x 入口
        DC      D75             ;0.75x 入口
        DC      D375            ;0.375x 入口
JG      DS      1
        END
```

3. 方法概要

(1) 从算法来说，计算 $5x$ 时，先求出 $4x$，再加 x。计算 $10x$ 时，先求 $5x$，再求其 2 倍，便得出 $10x$。计算 $0.75x$，即 $\frac{3}{4}x$。先通过右移求 $\frac{1}{2}x$ 和 $\frac{1}{4}x$，$\frac{1}{2}+\frac{1}{4}\quad x=\frac{3}{4}x=0.75x$。计算 $0.375x$，即 $\frac{3}{8}x$。借助移位求 $\frac{1}{4}x$ 及 $\frac{1}{8}x$，$\frac{1}{4}+\frac{1}{8}\quad x=\frac{3}{8}x=0.375x$。

(2) 从程序控制上说，将 4 段程序的入口地址，WB、SB、D75、D375 作为地址常数存于以 RK 为首地址的 4 个单元中，而 RK 相当一个基址，在执行过程中分别形成 4 个转移入口，

```
LD   GR2,RK,GR2      若 GR2 中为 0，则形成 JMP   WB
LD   GR2,RK,GR2      若 GR2 中为 1，则形成 JMP   SB
```

LD GR2,RK,GR2 若 GR2 中为 2,则形成 JMP D75

LD GR2,RK,GR2 若 GR2 中为 3,则形成 JMP D375

因为在执行 LD 指令时,可将(E)→GR2,而 E = ADR + (XR) = RK + (GR2),当(GR2) = 0 时,E = RK,而(E) = (RK) = WB,依次类推。

9.3.2 乘、除法

1.设计说明

(1) 已知 A,B 两数均为小于 128 的自然数,设计一个对两数进行乘、除运算的程序。

(2) 当 A < B 时计算 A×B 存入 C;当 A > B 时计算 A÷B 并将商和余数均存于 D,商放在 D 单元的高 8 位,余数放在 D 单元的低 8 位;当 A = B 时不进行运算。

2.参考程序

```
CCW     START
        LD      GR1,A           ;数 A→GR1
        ST      GR1,WK          ;数 A→WK
        CPA     GR1,B           ;A:B
        JMI     CN              ;A < B 转乘
        JZE     BD              ;A = B 转不操作
        LEA     GR3,0           ;0→GR3(商数)
CW      SUB     GR1,B           ;A - B
        JPZ     JS              ;转商计数
        ADD     GR1,B           ;恢复被除数
        ST      GR1,D           ;余数→D
        SLL     GR3,8           ;商移至高 8 位
        OR      GR3,D           ;商和余数拼在一起
        ST      GR3,D           ;商、余→D
        JMP     WL
JS      LEA     GR3,1,GR3       ;求商
        JMP     CW
CN      LEA     GR0,0           ;0→GR0(乘积)
JX      LD      GR1,WK          ;被乘数→GR1
        AND     GR1,Y           ;取其低位
        JZE     BJ              ;低位为 0 转
        ADD     GR0,B           ;求积
        LD      GR1,WK          ;被乘数→GR1
        SRL     GR1,1           ;被乘数右移 1 位
        ST      GR1,WK
        JZE     CJ              ;转保存积
        JMP     JX
CJ      ST      GR0,C           ;积→C
WL      EXIT
A       DS      1               ;被乘数、被除数
B       DS      1               ;乘数、除数
```

```
C       DS      1               ;积
D       DS      1               ;商及余
Y       DC      # 0001          ;低位 1
WK      DS      1               ;工作单元
        END
```

3 . 方法概要

(1) 乘法,判断被乘数的低位,若低位为 1,则累加一次乘数;若低位为 0,则不加乘数。将被乘数右移 1 位,仍判低位为 1 还是为 0,而决定是否累加乘数,直至操作完被乘数的所有位。当被乘数为 0 时,所得到的乘数累加值,即为 A、B 之积。

(2) 除法,被除数减除数,若结果为正,则计数器加 1,被除数继续减除数一直减到出现结果为负面停止,此时说明已不够除了,因而要恢复多减一次除数的被除数,该值即为余数。下面是 7÷2 = ? 的操作演示。

7÷2 = ?						
操作	被除数	除 数	被除数减除数	判结果	商计数	恢复值即余
第①次	7	2	7 - 2 = 5	为正	0 + 1 = 1	——
第②次	5	2	5 - 2 = 3	为正	1 + 1 = 2	——
第③次	3	2	3 - 2 = 1	为正	2 + 1 = 3	——
第④次	1	2	1 - 2 = - 1	为负	不计数	——
恢复	- 1	2	——	——	——	- 1 + 2 = 1
结果					商为 3	余数为 1

9.4　极值与排序

9.4.1　求极值

1 . 设计说明

(1) 已知在内存中存放着范围在 - 32768～32767 的一组数据,每一个数据占一个单元。要求在这组数据中求出极大值和极小值。

(2) 数据的首地址已在 GR1 中,数据的个数在 GR2 中。

(3) 该程序设计为子程序,求出极大值后放在 GR2 中,极小值放在 GR3 中后返回主程序。

2 . 程序流程图

如图 9.2 所示。

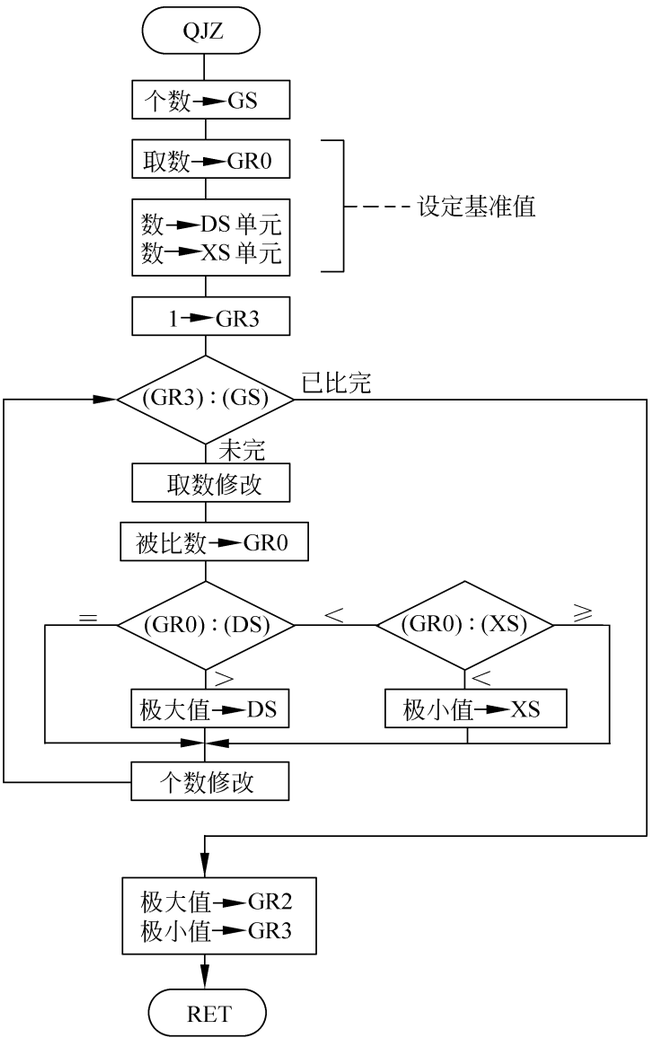


图 9.2　求极值程序流程图

3. 参考程序

```
QJZ     START
        ST      GR2,GS          ;数据个数→GS
        LD      GR0,0,GR1       ;取数(基准值)→GR0
        ST      GR0,DS          ;基准值→DS 极大值单元
        ST      GR0,XS          ;基准值→XS 极小值单元
        LEA     GR3,1           ;计数器初值 1→GR3
JX      CPA     GR3,GS          ;是否完?
        JPZ     BL              ;已完转 BL
        LEA     GR1,1,GR1       ;取数修改
        LD      GR0,0,GR1       ;取被比较数→GR0
        CPA     GR0,DS          ;与大数比
        JMI     YXB             ;小转
        JZE     ZSJ             ;等转计数修改
        ST      GR0,DS          ;保留大数
        JMP     ZSJ             ;转计数修改
YXB     CPA     GR0,XS          ;与小数比
        JPZ     ZSJ             ;转计数修改
```

```
        ST      GR0,XS          ;保留小数
ZSJ     LEA     GR3,1,GR3       ;计数修改
        JMP     JX              ;转继续
BL      LD      GR2,DS          ;保存大数→GR2
        LD      GR3,XS          ;保存小数→GR3
        RET                     ;返主
GS      DS      1               ;个数工作单元
DS      DS      1               ;大数工作单元
XS      DS      1               ;小数工作单元
        END
```

4.方法概要

(1) 本程序采取逐一比较的方法从一组数据中找出一个最大值和一个最小值。实现的方法是,用第一个数作为基准值,并将该数既假定为极大值,又假定为极小值分别放在大数单元和小数单元。基准值设定后便从第二个数开始进行逐一比较,并在比较中运用"留大弃小、留小弃大"的原则保留大值和小值。待将所有数比较完毕,极大、极小值就找到了。

(2) 程序中所使用的寄存器及职能如下,

GR1:开始存放着数据首地址,并在指令中出现在变址寄存器的位置上,每进行一次比较操作,修改一次。GR1 的修改表示取数位置的修改。

GR2:开始由主程序转子时存放着数据个数,到了子程序后便将所存个数传给了个数工作单元而完成任务。在程序执行到最后,按要求 GR2 作了存放最大值结果的空间。

GR3:操作个数计数器,开始时初值为 1,以后每进行一次取数、比较操作,则进行一次加 1 修改。全部操作完成后,GR3 按要求存放了最小值结果。

GR0:工作寄存器,承担传送、保存中间数的任务。

9.4.2　扣除极值的评分

1.设计说明

(1) 由 5 名裁判为一项体育运动比赛打分,裁判打分的分数范围为 0～10 分的正整数。

(2) 评分办法是去掉一个最高分,去掉一个最低分,剩下的 3 个分数求其平均,并要求平均分保留一位小数。例如,5 名裁判的打分为 6,7,7,8,8 时,则去掉一个 8 分和一个 6 分,最后得分为 22/3=7.3 分。

(3) 由于在 CASL 中只能保留整数,所以小数部分要扩大 10 倍,即 0.3 要变成 3 存入。

(4) 要求,GR1 中放最后得分的整数部分,GR2 放最后得分的小数部分。如按上面所举例子,GR1 最后应存入 7,GR2 应存入 3。

(5) 本程序设计为子程序,转子时已将 5 位裁判的打分放在了 5 个内存单元,而分数单元的首地址已在 GR3 中。

2.参考程序

```
PF      START
        ST      GR4,SP          ;保存栈顶地址
```

```
        LEA   GR0,0          ;最高分初值 0→GR0
        LEA   GR1,10         ;最低分初值 10→GR1
        LEA   GR2,0          ;总分累加器初值 0→GR2
        LEA   GR4,4
JX      ADD   GR4,0,GR3      ;取分数并累加
        CPA   GR0,0,GR3      ;比较,是否为高分?
        JPZ   ZD             ;大于、等于转,即分数低转
        LD    GR0,0,GR3      ;高分置换
ZD      CPA   GR1,0,GR3      ;比较是否为低分
        JMI   QSG            ;小于转,即分数高转
        LD    GR1,0,GR3      ;低分置换
QSG     LEA   GR3,1,GR3      ;修改取数
        LEA   GR4,-1,GR4
        JPZ   JX
        ST    GR0,WK         ;最高分数→WK
        SUB   GR2,WK         ;总分去掉一个最高分
        ST    GR1,WK         ;最低分数→WK
        SUB   GR2,WK         ;去掉一个最低分,扣除后的总分
        LEA   GR1,-1         ;商的计数器初值
CFA     LEA   GR1,1,GR1      ;求平均分,整数部分→GR1
        LEA   GR2,-3,GR2     ;多次减 3,计次数代替除 3
        JPZ   CFA            ;继续求商,直到(GR2) - 3 为负值
        LEA   GR2,3,GR2      ;将余数恢复为正值
        ST    HR2,WK         ;余数→WK
        ADD   GR2,WK         ;建立余数与第 1 位小数的关系
        ADD   GR2,WK
        LD    GR4,SP         ;恢复栈指针
        RET
WK      DS    1
SP      DS    1
        END
```

3.方法概要

(1) 实现本设计的第一个问题就是求极值,即找出一个最高分和一个最低分。求极值首先要给出基准值,设定基准值的方法不止一个,而在本例中把求极大值的基准值设定为 0,把求最小值的基准值设定为 10。显然,这样的基准值一旦与被比较值比较时,第一次就会被置换掉。

(2) 实现本例的第二个问题是求总分,由于求极值时,基准值要和 5 个评委的打分一一比较,也就是说需要求极值的 5 个得分要依次被取出。所以在求极值的同时,通过对 5 个分数的累加,与极值一起就把总分求出来了。

(3) 实现本例的第三个问题是去掉一个最高分,去掉一个最低分。由于有了 5 个分数之和及最高、最低分后,两者相减就产生了实际得到的总分。

(4) 本例的最终要求是计算参赛者的(去掉高、低分)平均分,因此要将总分除以 3。在本例中,除 3 的作法是采取连续减 3,记录减 3 次数的方法。例如,三个分数 7、7、8 之

和为 22,计算 22÷3 = ?时,从 22 中连续减 3,直减到 22 变为负值停止,减 3 的次数为 7,则商为 7。

(5) 在本例中还要求计算平均分时保留一位小数。在 CASL 中既没有除法指令,更没有小数运算。而本例采取把连续减 3 所剩下的余数进行特殊处理的方法把余数保留了下来。在此用实际例子来说明处理的方法,如 22÷3 = 7 并余 1,余数为 1 则第 1 位小数(扩大 10 位)必为 3。这说明当余数不为 0 时,余数为第 1 位小数的 3 倍。依次由余数转换成小数,并以整数形式保存下来。

(6) 本例中一开头使用了一条指令为,

```
ST      GR4,SP
```

最后又使用了一条指令为,

```
LD      GR4,SP
```

这两条指令似乎与本程序没有关系。它们的真正用意又是什么呢?由于在本例中使用的寄存器较多 GR0～GR4 都用上了,而本程序为子程序,也许在主程序中使用了栈操作指令,而栈操作与栈顶指针 GR4 有关。因此,在子程序中使用 GR4 之前先用 ST 指令把 GR4 的内容保存下来,子程序返主前又把 GR4 的内容恢复以备主程序继续使用。而更直接的原因是转子指令 CALL 的执行,就有一个返回地址进栈的隐含操作,而执行返主指令 RET 时,又有一个出栈操作,均与 GR4 有关。在此提醒使用者,在子程序中使用 GR4,上述两条指令是必不可少的。

9.5　数据处理

9.5.1　数据压缩

1.设计说明

(1) 在以 SJ 为首地址的 80 个单元中存放着 80 个数字字符数据,设计一个数据压缩处理程序,先将数字字符数据变为 BCD 码,再将 4 个 BCD 码压缩在一个单元中。

(2) 压缩后的数据存放在以 YS 为首地址的 20 个单元中。

2.参考程序

```
YS    START
      LEA     GR1,0
      LEA     GR2,0
      ST      GR2,WK
S1    LEA     GR3,4
S2    LD      GR0,SJ,GR2      ;取字符数据
      AND     GR0,C4          ;取其 BCD 码
      OR      GRO,WK          ;拼入工作单元
      ST      GR0,WK
      LEA     GR2,1,GR2       ;取数修改
      CPL     GR2,C80         ;是否处理完 80 个数据
      JZE     WL              ;处理完则转
```

```
        LEA     GR3, -1,GR3     ;压缩个数修改
        JZE     CY              ;是否完成4个,完成转,未完下
        LD      GR0,WK          ;取压缩数
        SLL     GR0,4           ;左移4位为下一个数留出空位
        ST      GR0,WK
        JMP     S2              ;继续处理
CY      LD      GR0,WK          ;取压缩完的数
        ST      GR0,YS,GR1      ;存入YS为首地址单元
        LEA     GR1,1,GR1       ;修改存数
        JMP     S1              ;继续处理
WL      EXIT
SJ      DS      80
WK      DS      1
C4      DC      #000F
C80     DC      80
YS      DS      20
        END
```

3.方法概要

(1) 本程序的任务是将4个数变换后压缩在16位长的一个单元中,初始条件及压缩结果图示如下,

数字字符数据		
地址	ASCII	
SJ	0033_H	'3'
+1	0036_H	'6'
+2	0038_H	'8'
+3	0032_H	'2'
…	…	

压缩后的数据	
地址	BCD码
YS	3682_H
	…

(2) 由字符的ASCII转换为BCD,只需取ASCII的低4位,因此数据处理比较简单。

(3) 在程序控制上,每变换4个BCD码,则需合并在同一单元,并存入指定的空间。在程序中采用了左移4位及或运算拼位的方法将4个BCD码依次拼入同一个单元。

9.5.2 将负数变为绝对值

1.设计说明

(1) 设计一个数据检查、处理程序,检查数据区中的数据有多少个负数,并将负数转为绝对值。

(2) 数据区的首地址已在 GR1 中,数据区的长度在 GR2 中。检查出的负数个数存于 GR0 中,负数转换为绝对值后仍存放于原负数单元。

2.参考程序

```
JFBZ    START
        LEA     GR0,0          ;负数个数计数器初值 0→GR0
        LEA     GR2,0,GR2      ;是否检查完?
JX      JZE     WL             ;已完,转结束
        LD      GR3,0,GR1      ;被检查数→GR3
        LEA     GR3,0,GR3      ;是否为负数?
        JPZ     XG             ;数为正或零转
        ADD     GR0,CY         ;负数计数
        EOR     GR3,CF         ;负数全字取反
        LEA     GR3,1,GR3      ;加 1 变为绝对值
        ST      GR3,0,GR1      ;绝对值放回原处
        LEA     GR1,1,GR1      ;修改数据单元
        LEA     GR2,-1,GR2     ;修改个数
        JMP     JX
WL      EXIT
CY      DC      1
CF      DC      #FFFF
        END
```

3.方法概要

(1) 本程序涉及到将一个负数转换为绝对值的问题,采取的方法是全字求反加 1。由于负数在计算机中使用的补码,由补码返回原码是数的尾数求反加 1,这与变绝对值有所不同。下面以 -1 为例进行一下比较就更清楚了。

负数真值	-1	变换结果
16 位补码	1111111111111111	——
尾数求反加 1	1000000000000001	-1 的原码
全字求反加 1	0000000000000001	-1 的绝对值

(2) 在程序中,实现全字求反使用了异或全 1 的方法,见指令:EOR GR3,CF。在统计负数个数时,要求使用 GR0,而 GR0 在 CASL 中不能用作变址。因此,在计数器进行加 1 操作时,不能使用 LEA GR0,1,GR0 指令,只能使用 ADD GR0,CY 指令,这一点要引起注意。

9.5.3　在非数值信息中统计数字、字母和符号的个数

1.设计说明

(1) 在以 ZF 为首地址的 80 个单元中存放着非数值信息,每个单元中为 1 个字符的 ASCII。

(2) 设计一个统计程序,统计出数字、字母、其他符号各有多少个,分别放在 SZS(数字个数)、ZMS(字母个数)、FHS(符号个数)单元中。

(3) 程序设计为主—子结构。

2.参考程序

```
TJGS    START
        LEA     GR1,48          ;数字下界
        LEA     GR2,58          ;数字上界
        CALL    TJZ
        ST      GR1,SZS         ;数字个数→SZS
        LEA     GR1,65          ;字母下界
        LEA     GR2,91          ;字母上界
        CALL    TJZ
        ST      GR1,ZMS         ;字母个数→ZMS
        LD      GR3,80          ;总个数
        SUB     GR3,SZS         ;减数字个数
        SUB     GR3,ZMS         ;减字母个数
        ST      GR3,FHS         ;符号个数→FHS
        EXIT
TJZ     ST      GR1,XJ          ;下界→XJ
        ST      GR2,SJ          ;上界→SJ
        LEA     GR1,79          ;变址器初值 79→GR1
        LEA     GR2,0           ;计数器初值 0→GR2
JX      LD      GR3,ZF,GR1      ;取信息字符
        CPL     GR3,XT          ;与下界比较
        JMP     DX
        CPL     GR3,SJ          ;与上界比较
        JPZ     DX
        LEA     GR2,1,GR2       ;个数计数
DX      LEA     GR1,-1,GR1      ;取数修改
        JPZ     JX
        RET
ZF      DS      80
SZS     DS      1               ;数字个数单元
ZMS     DS      1               ;字母个数单元
FHS     DS      1               ;符号个数单元
XJ      DS      1               ;下界工作单元
SJ      DS      1               ;上界工作单元
        END
```

3.方法概要

(1) 非数值数据在计算机中为字符的 ASCII,因此统计数字字符 0～9,用其 ASCII 的上、下界(48～57);统计大写英文字母 A～Z,用其 ASCII 的上、下界(65～90)。数字、字母统计出各自的个数后,在总数 80 中扣除便得出其他字符的个数。

(2) 在主程序中给出所要筛选字符 ASCII 的上、下界,转入子程序完成相应的个数统计后返回主程序。

9.5.4　自动阅卷及评分

1.设计说明

(1) 试卷上有 16 道判断题,每题 1 分,满分 16 分。

(2) 考生用答题卡答题,答题卡上从左至右有 1～16 依次排列的题号,题号下方是 16 个答题格,供填写答案使用,形式如下:

答题卡

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

(3) 答题方法为,若认为试题正确则答 1,若认为试题错误则答 0,每个考生答完试题后在答题卡上是一个 16 位的 0、1 序列。

(4) 已知标准答案存放在 BZ 单元中,考生的答案在 GR1 中,准考证号 8 位在 GR0 的低 8 位中,设计一个阅卷、评分程序,最终将考生的准考证号(占高 8 位)与考试成绩(占低 8 位)压缩在 CJ 单元中。

2.参考程序

```
UJPF    START
        LEA     GR3,16          ;计数器初值 16→GR3
        LEA     GR2,0           ;分数累加器初值 0→GR2
        EOR     GR1,BZ          ;阅卷,即答卷与标准答案操作结果→GR1
JX      LEA     GR1,0,GR1       ;判断是否计分
        JMI     ZW
        LEA     GR2,1,GR2       ;累加计分→GR2
        SLL     GR1,1           ;逐题阅卷
        LEA     GR3,-1,GR3      ;修改
        JNZ     JX
        SLL     GR0,8           ;准考证号移至高 8 位
        ST      GR0,WK
        OR      GR2,WK          ;准考证号与成绩压缩在一起
        ST      GR2,CJ          ;结果→CJ
        EXIT
BZ      DS      1               ;标准答案单元
CJ      DS      1               ;最后结果单元
WK      DS      1               ;工作单元
        END
```

3.方法概要

(1) 阅卷采取的办法是用标准答案与考生答案进行异或操作,假定标准答案为全 1,考生答卷也是全 1,两者操作结果为,

```
        1111111111111111    ;标准答案
⊻       1111111111111111    ;考生答卷
-----------------------------
        0000000000000000    ;异或结果
```

如果考生答卷全为0,则两者操作结果为,

```
        1111111111111111    ;标准答案
⊻       0000000000000000    ;考生答卷
-----------------------------
        1111111111111111    ;异或结果
```

由这两种不同情况与标准答案的异或操作可以得出,

① 考生答对,操作结果为0;

② 考生答错,操作结果为1。

如果把这个操作结果作为有符号的数看待,并将0,1左移到符号位,必然有答对为正,答错为负的状况。

(2) 评分,对操作结果逐位左移到符号位,判断其正、负。答对为正,计1分;答错为负,不计分。逐位进行判正、负后就得出了最后成绩。

(3) 最终成绩得出后与准考证号借助或运算拼在一起即可。

9.6 码制变换

9.6.1 原码、补码和移码

1.设计说明

(1) 已知在SZ为首地址的80个单元中存放着数值性整数的原码,设计一个码制变换程序对数据进行变换,并将变换结果存放在原单元中。

(2) 如果数据为正,则将其变为移码;如果数据为0,则不变;如果数据为负,则将其变为补码。

2.参考程序

```
MB      START
        LEA     GR1,0
JX      LD      GR2,SZ,GR1
        LEA     GR2,0,GR2
        JMI     FS              ;负转
        JZE     CS              ;零转
        OR      GR2,FH          ;补码变移码
        JMP     CF
FS      EOR     GR2,KF          ;求反码
        LEA     GR2,1,GR2       ;加低位1
CF      ST      GR2,SZ,GR1      ;放回
        LEA     GR2,-1,GR1      ;修改
```

```
        JPZ     JX              ;未完继续
        EXIT
SZ      DS      80
FH      DC      #8000           ;符号,即最高位 1
KF      DC      #7FFF           ;除最高位外的全 1
        END
```

3.方法概要

(1) 补码与移码只相差一个符号位,所以由补码变移码时,只需改变符号位。由于正数的原码与补码相同,所以当数据为正数时,该数则为补码,将符号置为 1,即为移码。

(2) 当数据为负数时,通过求反码加上低位 1,即得补码。

9.6.2　奇校验编码

1.设计说明

(1) 在以 ZM 为首地址的 60 单元中存放着字符数据。

(2) 在字符的 7 位 ASCII 的左边,即第 8 位作为校验位,将字符代码变换为奇校验码仍放在原单元中。

2.参考程序

```
JJY     START
        LEA     GR1,0           ;计数兼变址初值 0→GR1
ZXH     LEA     GR2,6           ;异或次数 6→GR2
        LD      GR3,ZM,GR1      ;取单元中的 ASCII
        ST      GR3,DM          ;ASCII→DM 工作单元
        AND     GR3,CY          ;取其低位
        ST      GR3,W           ;低位→W
JX6     LD      GR3,DM          ;ASCII→GR3
        SRL     GR3,1           ;右移 1 位
        ST      GR3,DM          ;保留动态 ASCII
        AND     GR3,CY          ;取低位→GR3
        EOR     GR3,W           ;两两异或→GR3
        ST      GR3,W           ;保留异或中间结果→W
        LEA     GR2,-1,GR2      ;修改异或次数
        JNZ     JX6             ;未完,则继续
        LEA     GR3,0,GR3       ;异或结果(产生 FR 状态)
        JNZ     XG
        LD      GR3,ZM,GR1      ;取字符的 ASCII
        OR      GR3,C80         ;拼上校验位
        ST      GR3,ZM,GR1      ;奇校验码放回原单元
XG      LEA     GR1,1,GR1
        CPL     GR1,C60         ;是否做完 60 个数据
        JMI     ZXH             ;继续
        EXIT
```

```
ZM      DS      60
C60     DC      60
CY      DC      #0001
DM      DS      1
W       DS      1
C80     DC      #0080
        END
```

3.方法概要

(1) 将7位ASCII变为奇校验码需增加一个附加位,即奇校验位。按本题的要求,附加位放在ASCII的左边。以字符'A'为例,形式如下:

(2) 附加位的值可用编码方程求出。令7位ASCII由左至右依次为A7,A6,A5,A4,A3,A2,A1,则编码方程表示为,

$$J = A7 \quad A6 \quad A5 \quad A4 \quad A3 \quad A2 \quad A1$$

式中 为异或运算符,J表示附加的校验位。

(3) 由已知的ASCII代入编码方程求J,当求出的J为0,则说明已知信息中含有偶数个1,而附加位填1,即构成了奇校验码。当求出的J为1,则说明已知信息中含有奇数个1,而附加位填0,即构成了奇校验码。

附 录 1

CASL使用说明

CASL 是 Computer Assembly System Language 的简称,即计算机汇编语言系统。由于 CASL 是建立在假想机 COMET 上的汇编语言系统,所以使用 CASL 需要了解与 COMET 结构有关的硬件背景。CASL 不仅与硬件有关,像指令的表述、数的形式、数的编码等内容都有它自身的定义及规定,统称为 CASL 的软件环境。

附 1.1 CASL 的硬件背景

(1) COMET 的 CPU

与 CASL 有关的 CPU 部件有通用寄存器、变址寄存器、标志寄存器、指令计数器、栈指针等。

① 通用寄存器

通用寄存器的代号为 GR,有 5 个,长度为 16 位。5 个通用寄存器分别用 GR0,GR1,GR2,GR3,GR4 表示。这 5 个寄存器都可以在 CASL 的有关指令中直接出现。根据指令的功能和寄存器出现的位置不同,其职能也有所不同。在 CASL 中,寄存器可以作为运算器、移位器、累加器、计数器、数据暂存器、地址寄存器等。CASL 中的所有运算及操作都在寄存器中进行。在 CASL 的指令中大都包含寄存器。

② 变址寄存器

变址寄存器用 XR 表示,在指令中是用来形成地址偏移量,即变址参数的寄存器。CASL 规定,使用通用寄存器中的 GR1,GR2,GR3,GR4 作为变址寄存器,而 GR0 不能用作变址寄存器。在一条指令中若出现两个寄存器,右边的一个寄存器则为变址寄存器。由于指令的功能不同,有时出现在变址寄存器位置的寄存器并不一定真正起变址的作用。

③ 标志寄存器

标志寄存器用 FR 表示,长度为两位。标志寄存器也称标志寄存器,用来存放有关指令的执行后的状态。在 CASL 中有 12 条指令与标志寄存器有关。两位标志寄存器有 00,01,10,11 四种状态,使用其中的 3 种,即 00,01,10 来表示指令的执行状态。其含义如下:

00: 表示运算结果为正数,或者比较结果为大于。

01: 表示运算结果为零,或者比较结果相等。

11: 表示运算结果为负数,或者比较结果为小于。

标志寄存器 FR 并不出现在 CASL 指令中，其作用是隐含的，当 LEA，ADD，SUB，AND，OR，EOR，SLA SRA，SLL，SRL，CPA，CPL 12 条指令执行后，其结果将会使 FR 置为 00，01，10 三种状态之一。

④ 指令计数器

指令计数器 PC，其作用是记录当前被执行指令的首地址。执行完一条指令后。由于 CASL 指令占两个单元，所以 PC 的内容自动加 2。只当转移、转子指令执行时，PC 被置为转移、转子地址。PC 所描写的是程序执行的动态过程，与具体指令功能没有什么关系，也不在指令中出现。

⑤ 栈指针

栈指针用 SP 表示，它指示的是堆栈的栈顶地址。当 CASL 程序启动时，操作系统将设置一个堆栈区，并将该区的末地址加 1 置于 SP 中。在 CASL 中规定，使用寄存器 GR4 作为栈指针。数据进栈，栈顶上升，则(GR4) - 1→GR4；数据出栈，栈顶下降，则(GR4) + 1→GR4。在 CASL 中，除进栈指令 PUSH，出栈指令 POP 与栈指针 GR4 有关外，转子 CALL，返主 RET 也隐含发生进、出栈的操作。因此使用上述 4 条指令时，均会涉及到 GR4，使用中不能随便破坏 GR4 中的内容。

(2) COMET 的内存

① 内存容量

COMET 的内存容量为 65536 个字，字长 16 位。65536 个字，即 64K 字，它等于 128KB。

② 内存编址

从 0 地址开始，65536 个单元的编址范围为 0～65535，若用 4 位十六进制数表示其地址为 0000_H～$FFFF_H$，而在 CASL 中并不直接使用这种物理地址。

附 1.2 CASL 的软件环境

(1) CASL 的字符集

描写 CASL 使用的字符全体，即字符集如下表。

CASL 汇编字符集

高位 / 低位	2	3	4	5
0	间隔	0	@	P
1	!	1	A	Q
2	”	2	B	R
3	#	3	C	S
4	$	4	D	T
5	%	5	E	U

续表

高位 / 低位	2	3	4	5
6	&	6	F	V
7	’	7	G	W
8	(	8	H	X
9	)	9	I	Y
A	*	:	J	Z
B	+	;	K	[
C	,	<	L	\
D	-	=	M	]
E	.	>	N	^
F	/	?	O	-

由 CASL 字符集可以看出,描写、表述 CASL 的所有字符,其内码使用的是美国信息交换标准代码,即 ASCII。例如:

数字字符 0～9 的 7 位内码为 0110000～0111001,用十六进制表示为 30_H～39_H,与 BCD 码相差一个常数 30_H。

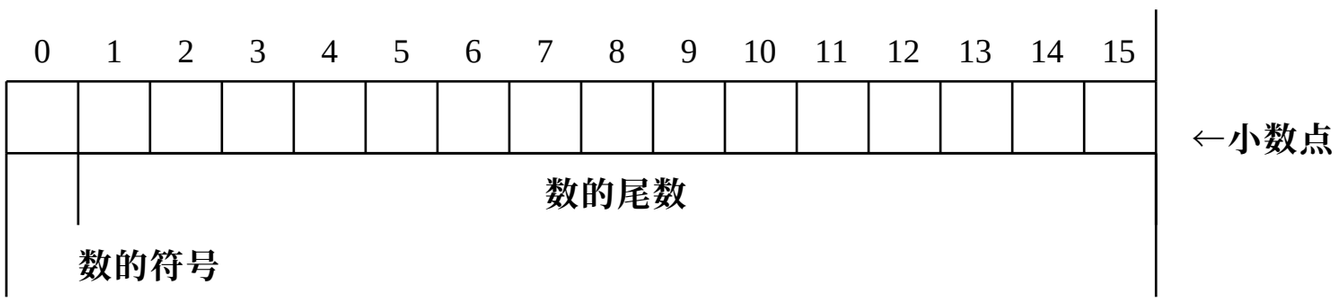
大写英文字母 A～Z 的 7 位内码为 1000001～1011010,用十六进制表示为 41_H～$5A_H$。

(2) CASL 中的数

在 CASL 中允许直接使用的数有带符号的十进制数,无符号的十进制形式的逻辑数、十六进制数、字符常数、标号地址常数等 5 种。

① 带符号的十进制数

数的符号用 0 表示正号,用 1 表示负号。数在 16 位长的一个字中,符号占 1 位,尾数占 15 位,用补码表示,其范围为 -2^{15}～$2^{15}-1$,即 -32768～+32767。数的表示形式如下:



② 无符号的逻辑数

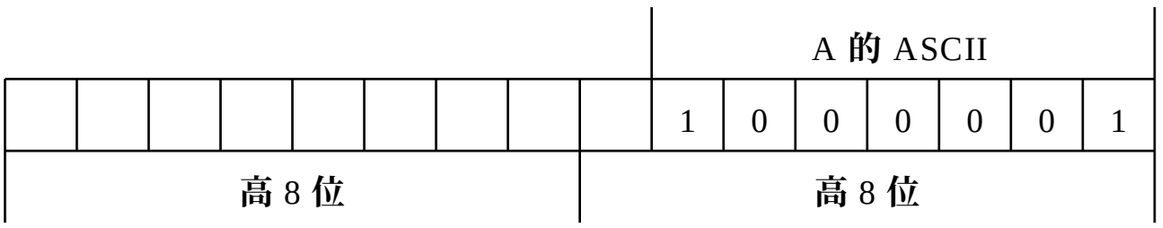
数在 16 位的一个字中,最小值为 0,最大值为 $2^{16}-1$,数的表示范围为 0～65535。

③ 十六进制数

使用十六进制数均要求为四位数,且在其前加上标识符 # 号。例如, # 1234, # 1A2B, # 0000, # ABCD 等都是符合要求的十六进制数。数在一个 16 位的字中,最小值为 # 0000,最大值为 # FFFF,数的表示范围为 # 0000～ # FFFF。

④ 字符常数

在 CASL 中使用的字符常数是由单引号括起来的字符构成,一个字符常数可以是一个字符,也可以是多个字符。例如, 'A', '2B', 'ABC', '3 + 2 = ?'都是符合要求的字符常数。字符常数在计算机中,一个字符占一个内存单元,每个单元的低 7 位存入字符对应的 7 位 ASCII,其他位为 0,以字符常数 'A' 为例,其形式如下:

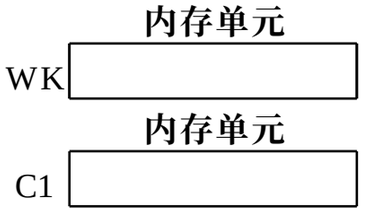


⑤ 标号地址常数

标号地址常数由大写英文字母打头,其后跟字母或数字,长度在 6 个字符以内。例如,A,B2,C3,DY,WK,DATS 等均为标号地址常数。标号地址常数也叫标号名,它很像高级语言中的变量名或者数组元素名,在 CASL 中经常使用标号名表示内存的一个单元地址,或者表示一组单元的首地址。例如:

内存单元	
地址	单元内容
DY	0041
+ 1	0042
+ 2	0036
…	…

'A'
'B'
'6'



图中,标号名 DY 为一组单元的首地址,WK,C1 分别为一个内存单元的地址。

(3) CASL 的指令

CASL 中有三类指令,即基本指令、伪指令、宏指令。基本指令是 CASL 汇编语言基本功能的体现,共有 23 条指令。伪指令是对汇编编译进行说明的指令,共有 4 条。宏指令是一条指令可以启动一个程序段的指令,共有 3 条。

CASL 的指令结构如下表所示。

CASL 基本指令结构

①	②	③	④
[标号名]␣	操作码␣	操作数␣	;注释
QS	LD	GR1,DY	;(DY)→GR1

①、②、③是指令的主体,它们之间在书写上要留出一个以上的空格;即 ␣。注释是为了阅读由使用者所加的说明,注释之前需加分号。

在汇编语言中为了表述方便,常用方括号和圆括号,其含义如下:

[ ],方括号表示括号中的内容可以省略。

( ),圆括号表示其中的内容,如 DY 为标号地址,而(DY)为以 DY 为地址的内存单元中的内容。再如,GR1 为寄存器代号,而(GR1)代表寄存器 GR1 中的内容。如果使用两对圆括号,如((GR1))表示寄存器 GR1 中内容的内容。

上表中给出的是一条取数指令 QS LD GR1,DY,其含义是取内存单元 DY 中的内容,放到 CPU 寄存器 GR1 中。

在 CASL 的 23 条基本指令中,有 21 条指令的表示形式有两种,一种是直接操作形式,另一种是间接操作形式。下面是基本指令的两种形式,

CASL 基本指令的形式

标 号	操 作 码	操 作 数	备 注
[标号名]	功能代码	GR,ADR	直接形式 E = ADR
[标号名]	功能代码	GR,ADR,XR	间接形式 E = ADR + (XR)

直接形式与间接形式的惟一不同是在确定有效地址 E 上,直接指令中的 E = ADR,间接指令中的 E = ADR + (XR),两者相差(XR)。显然,这种差别表现为寻址的不同。

附 1.3 CASL 的指令系统

(1) 基本指令

指令名称	指令形式	指令功能	备 注
① 取数	LD GR,ADR [,XR]	(E)→GR	
② 存数	ST GR,ADR [,XR]	(GR)→E	
③ 传送	LEA GR,ADR [,XR]	E→GR	形成(FR)
④ 加法	ADD GR,ADR [,XR]	(GR) + (E)→GR	形成(FR)
⑤ 减法	SUB GR,ADR [,XR]	(GR) - (E)→GR	形成(FR)
⑥ 算术左移	SLA GR,ADR [,XR]	/E (GR)→GR [注 1]	形成(FR)
⑦ 算术右移	SRA GR,ADR [,XR]	(GR) E\→GR [注 2]	形成(FR)
⑧ 逻辑乘	AND GR,ADR [,XR]	(GR)∧(E)→GR [注 3]	形成(FR)
⑨ 逻辑加	OR GR,ADR [,XR]	(GR)∨(E)→GR [注 4]	形成(FR)
⑩ 逻辑异或	EOR GR,ADR [,XR]	(GR)"(E)→GR [注 5]	形成(FR)
11 逻辑左移	SLL GR,ADR [,XR]	/E (GR)→GR [注 6]	形成(FR)

续表

指令名称	指令形式	指令功能	备　注
12 逻辑右移	SRL GR,ADR [,XR]	(GR)_E\→GR [注 7]	形成(FR)
13 算术比较	CPA GR,ADR [,XR]	(GR):(E) [注 8]	形成(FR)
14 逻辑比较	CPL GR,ADR [,XR]	(GR):(E) [注 9]	形成(FR)
15 无条件转移	JMP ADR [,XR]	E→(PC)无条件转向 E	
16 大于、等于转移	JPZ ADR [,XR]	满足条件转向 E,否则执行下一条。 [注 10]	
17 小于转移	JMI ADR [,XR]	满足条件转向 E,否则执行下一条。 [注 11]	
18 不等于转移	JNZ ADR [,XR]	满足条件转向 E,否则执行下一条。 [注 12]	
19 等于转移	JZE ADR [,XR]	满足条件转向 E,否则执行下一条。 [注 13]	
20 进栈	PUSH ADR [,XR]	E 进栈 [注 14]	隐含使用 GR4
21 出栈	POP GR	栈顶数据→GR [注 15]	隐含使用 GR4
22 转子	CALL ADR [,XR]	转向子程序入口地址 E [注 16]	隐含进栈操作
23 返主	RET (空)	返回主程序 [注 17]	隐含出栈操作

[注 1]: 算术左移,符号位不动,尾数左移 E 位,左移出现的空位自动补 0。

[注 2]: 算术右移,符号位不动,尾数右移 E 位,右移出现的空位自动补上与符号位相同的值,以便形成补码。

[注 3]: 逻辑乘,对位相乘,其法则为 0∧0 = 0,0∧1 = 0,1∧0 = 0,1∧1 = 1,即两端同时为 1 结果为 1,否则为 0。

[注 4]: 逻辑加,对位相加,其法则为 0∨0 = 0,0∨1 = 1,1∨0 = 1,1∨1 = 1,即有 1 为 1,否则为 0。

[注 5]: 逻辑异或,对位进行异或,其法则为 0 ∨ 0 = 0,0 ∨ 1 = 1,1 ∨ 0 = 1,1 ∨ 1 = 0,即相异为 1,相同为 0。

[注 6]: 逻辑左移,16 位依次左移 E 位,左移出现的空位自动补 0。

[注 7]: 逻辑右移,16 位依次右移 E 位,右移出现的空位自动补 0。

[注 8]: 算术比较是把比较双方视为有符号数的补码,先(GR) - (E),然后根据结果形成相应的标志位送入标志寄存器 FR。当(GR) > (E)时,00→FR;当(GR) = (E)时,01→FR;当(GR) < (E)时,10→FR。FR 中的值作为转移的先决条件。

[注 9]: 逻辑比较是把比较双方视为逻辑数进行比较,并根据比较结果形成标志位送入标志寄存器 FR。当(GR) > E时,00→FR;当(GR) = (E)时,01→FR;当(GR) < (E)时,10→FR。FR 中的值作为转移的先决条件。

[注 10]: 大于、等于转移,当标志寄存器 FR 中的内容为 00 或 01 时,则发生转移。如果以比较作为先导指令时,当(GR) > (E)或(GR) = (E)时,则发生转移。如果以运算作为先导指令时,当运算结果为正或为 0 时,则发生转移。

[注 11]: 小于转移,当标志寄存器 FR 中的内容为 10 时,则发生转移。如果以比较作为先导指令时,当(GR) < (E)时,则发生转移。如果以运算作为先导指令时,当运算结果为负时,则发生转移。

[注 12]: 不等于转移,当标志寄存器 FR 中的内容为 00 或 10 时,则发生转移。如果以比较作为先导指令时,当(GR) > (E)或(GR) < (E)时,则发生转移。如果以运算作为先导指令时,当运算结果为正或为负时,则发生转移。

[注 13]: 等于转移,当标志寄存器 FR 中的内容为 01 时,则发生转移。如果以比较作为先导指令时,当(GR) = (E)时,则发生转移。如果以运算作为先导指令时,当运算结果为 0 时,则发生转移。

[注 14]: 进栈,先(GR4) - 1→GR4,即栈顶上升,后将 E 送入栈中,GR4 为栈指针,它始终存放着栈顶地址。

[注 15]: 出栈,先将栈顶地址中的内容弹出到寄存器 GR 中,然后(GR4) + 1→GR4,即栈顶下沉。

[注 16]: 转子,伴随转子隐含进栈操作,该操作将返回地址压入栈中。进栈操作过程为(GR4) - 1→GR4(栈顶上升),(PC) + 2→(GR4)(返回地址进栈)。

[注 17]: 返主,伴随返主隐含出栈操作,该操作将存入栈中的返回地址弹出并送入指令计数器。出栈操作过程为((GR4))→PC,(GR4) + 1→(GR4)。

(2) 伪指令

指令名称	指令形式			指令功能
① 程序开头	[标号名]	START	[标号名]	说明源程序的开头
② 程序结尾	空白	END	空白	说明源程序的结尾
③ 定义常数	[标号名]	DC	(CASL 常数)	定义程序中使用的常数
④ 定义单元	[标号名]	DS	(单元个数)	定义程序中使用的内存单元

指令名称	指令形式			指令功能	备注
① 输入	[标号名]	IN	标号 1,标号 2	从外部设备输入字符数据	[注 1]
② 输出	[标号名]	OUT	标号 1,标号 2	输出字符数据	[注 2]
③ 终止程序执行	[标号名]	EXIT	空白	控制程序执行终止	

[注 1]: 标号 1,输入缓冲区的首地址,该地址为标号地址。输入缓冲区最大为 80 单元,每个字符占一个单元。输入缓冲区由伪指令定义。

标号 2,存放输入字符个数单元的标号地址。该单元由内指令定义。

[注 2]: 标号 1,输出缓冲区的首地址,该地址为标号地址。输出缓冲区最大为 80 个单元,每个字符占一个单元。输入缓冲区由伪指令定义。

标号 2,存放输出字符个数单元的标号地址。

附录 2

CASL与机器语言

任何一种汇编语言都对应着一套机器指令,即机器语言,CASL 是建立在假想机 COMET 上的汇编语言,所以它没有对应的机器语言,为了说明汇编语言与机器语言的关系,在此参照有关资料设计了一套对应于 CASL 的机器语言。

附 2 .1　机器指令与 CASL 指令的对应关系

一条 CASL 指令占两个单元,长度为 32 位。32 位的一条机器指令与 CASL 指令的对应关系如下:

	(1)	(2)	(3)	(4)
CASL 指令	操作码	GR	XR	ADR
机器指令	8 位	4 位	4 位	16 位

(1) 操作码,操作码用 8 位,最多可以表示 256 种指令,在 CASL,中只有 23 条基本指令。因此余量很大。

(2) GR,在 CASL 中只有 5 个通用寄存器,用 4 位表示寄存器最多可以表示 16 个。因此也有很大的余量。5 个通用寄存器的代号分配如下:

寄存器	GR0	GR1	GR2	GR3	GR4
二进制代号	0000	0001	0010	0011	0100
十六进制代号	0	1	2	3	4

(3) XR,在 CASL 中 XR 为变址寄存器,可以作为变址寄存器的只有 GR1～GR4,而 GR0 不能用作变址寄存器。因此,变址寄存器的代号只有 1～4。

(4) ADR,为内存地址,16 位地址可以表示的地址范围为 0000～FFFF,它正好覆盖了 COMET 内存的全部地址。

附 2.2 机器指令的编码

下面给出的是 CASL 23 种基本指令操作码与机器指令操作码的对应关系。

CASL 基本指令操作码与机器指令操作码的对应关系

序号	指令名称	操作码	机器指令操作码	
1	取数	LD	01	00000001
2	存数	ST	02	00000010
3	传送	LEA	03	00000011
4	加法	ADD	04	00000100
5	减法	SUB	05	00000101
6	算术左移	SLA	06	00000110
7	算术右移	SRA	07	00000111
8	逻辑乘	AND	08	00001000
9	逻辑加	OR	09	00001001
10	逻辑异或	EOR	0A	00001010
11	逻辑左移	SLL	0B	00001011
12	逻辑右移	SRL	0C	00001100
13	算术比较	CPA	0D	00001101
14	逻辑比较	CPL	0E	00001110
15	无条件转移	JMP	0F	00001111
16	大于、等于转移	JPZ	10	00010000
17	小于转移	JMI	11	00010001
18	不等于转移	JNZ	12	00010010
19	等于转移	JZE	13	00010011
20	转子	CALL	14	00010100
21	返主	RET	15	00010101
22	进栈	PUSH	16	00010110
23	出栈	POP	17	00010111

附 2.3 伪指令和宏指令的设定

(1) 伪指令 START,END 是为汇编编译指明 CASL 源程序的开始和结尾,因而它们不具体对应机器指令代码。定义常数 DC,和定义单元 DS 指令与机器代码的关系如下:

C1	DC	# 000F	0011000F	;假定 C1 的地址为 0011
C2	DC	'A'	00120041	;假定 C2 的地址为 0012
C3	DC	10	0013000A	;假定 C3 的地址为 0013
C4	DC	- 1	0014FFFF	;假定 C4 的地址为 0014

(2) 宏指令与机器指令的关系可以设定为

输入宏指令	IN	20
输出宏指令	OUT	21
终止程序执行	EXIT	22

附 2.4 CASL 程序转为机器语言程序实例

根据前面的设定,可以把 CASL 程序转换为机器语言程序,在此给出了一个转换实例。

地址	机器指令	CASL 程序		
			START	
0030	03100000		LEA	GR1,0
0032	03300000		LEA	GR3,0
0034	0123005A	B1	LD	GR2,SL,GR3
0036	0E200056		CPL	GR2,A
0038	1000003E		JPZ	B2
003A	05200054		SUB	GR2,L
003C	0F000042		JMP	B3
003E	05200056	B2	SUB	GR2,A
0040	0322000A		LEA	GR2,10,GR2

续表

地址	机器指令	CASL 程序		
0042	02200058	B3	ST	GR2,WK
0044	0B100004		SLL	GR1,4
			START	
0046	09100058		OR	GR1,WK
0048	03330001		LEA	GR3,1,GR3
004A	0D300052		CPA	GR3,CS
004C	11000034		JMI	B1
004E	0210005E		ST	GR1,RJ
0050	22000000		EXIT	
0052	00000004	CS	DC	4
0054	00000030	L	DC	'0'
0056	00000041	A	DC	'A'
0058		WK	DS	1
005A		SL	DS	4(占 4 个单元地址)
005E		RJ	DS	1
			END	

附录 3

习题答案

【习题 1】

1.(1) CASL 中使用的寄存器有通用寄存器 GR、标志寄存器 FR。通用寄存器的编号为 GR0,GR1,GR2,GR3,GR4。

(2) 通用寄存器在 CASL 程序中主要作用是寄存参加运算的数,保留计算或处理的中间结果和最后结果。此外它还作为累加器、计数器、移位操作器等,这些功能实际上是作为运算器使用。

(3) 标志寄存器的作用是记录指令执行后的状态,用 00,01,10 三种形式之一表示该状态。

(4) 寄存器 GR0 不能用作变址寄存器。

(5) 程序计数器中记录着当前被执行的指令地址,当指令执行结束后,程序计数器的内容自动加 2 跳到下一条要执行的指令位置上,但遇有转移、转子指令时,程序计数器的内容则按实际转移地址改变。程序计数器的功能是隐含的,并不在指令中体现。

(6) 栈指针用 GR4 来描述,GR4 中的内容永远保留着栈顶地址,数据进、出栈,使得栈顶上升、下降,GR4 中的内容也随之改变。数据进栈,栈顶上升,则 GR4 的改变为:

$$(GR4) - 1 \rightarrow GR4$$

数据出栈,则 GR4 的改变为

$$(GR4) + 1 \rightarrow GR4$$

如下图所示。

(7) COMET 是一个字长 16 位的定点整数机,在 COMET 上可以进行整数运算、逻

辑运算、码制变换及字符处理等。

(8) COMET 计算机内存 65536 个字,字长 16 位。因此内容为 64K 字,128K 字节,即 128KB。内存的地址范围为 0～65535,用十六进制表示为 0000_H ～$FFFF_H$。

(9) COMET 内存字长 16 位,一条指令占两个字,一个数据占一个字,即一个单元。

(10) 在 CASL 中使用内存单元有两种形式,使用单个的单元时,给该单元命名一个标号名,这个标号名就是该单元的逻辑地址,如命名 DY,则表示以 DY 为地址的内存单元。如果使用连续的多个单元,则只命名单元的首地址。例如,使用以 DATA 为首地址的 10 个单元,则第一个单元的地址为 DATA,以后的地址依次加上偏移量 1、2、3、4、5、……,最后一个单元的逻辑地址为 DATA + 9。

2 . (1) CASL 的字符集共有 60 个字符。其中,数字字符 0～9 共 10 个;大写英文字母 A～Z 共 26 个;其他符号 24 个。字符的内码为 ASCII。

(2) A 的内码若以数值而论为十进制的 65,已知 A 求 Z 只需在内码上加 25 即可,即 Z 的内码为 90。

(3) 字符 6 的七位 ASCII 为:0110110,按十进制数值论为 54,将'6'变为 6 只需做 54 - 48 = 6。若以二进制内码变换,可由 0110110 - 0110000 = 0110。由此可知,字符数字与数值数字相差的基数为 $0110000 = 30_H = 48_{10}$。

(4) CASL 的指令有基本指令、伪指令和宏指令三类。

(5) 伪指令在 CASL 中是只起说明作用,不产生目标代码的指令。

(6) 宏指令是一条指令可以启动一个程序段的指令。由于输入、输出等操作涉及的部件较多,用一条基本指令很难描述与概括,所以在汇编语言中采用宏指令来启动一个特定功能的程序段,这样使用起来即方便,又简捷。

(7) 在 CASL 指令中,操作数往往并不直接给出参加操作的数,而是给出数的来源与去向相关的存储空间,如内存单元和寄存器。其他汇编语言也是如此。

(8) 方括号所括起的项,表示是可以省略也可以填写的项。圆括号表示其内容,如果圆括号括上寄存器,则表示寄存器中的内容。例如(GR1)表示 GR1 中的内容,而不加圆括号 GR1,只代表寄存器号。假定 GR1 中存放着一个地址,则(GR1)为地址,如表示((GR1)),则为内容的内容,即地址中的内容,不言而喻,((GR1))为某地址中的数据。这种表示方法在汇编语言中已成为一种惯例。

(9) CASL 的标号名代表的是逻辑地址,用它可表示内存单元,也可以表示一条指令所在的位置。而标号名实际上经汇编编译后变成了内存的物理地址了。

(10) 标志寄存的内容或说状态 00,01,10 是由指令的作用产生的。一条指令的计算结果为正、零、负,则标志寄存器的内容依次为 00,01,10。而当实施比较、转移时,比较产生标志寄存器的状态,转移则按标志寄存器的内容决定转移是否成立。

(11) 在 CASL 中使用的数有:

① 有符号的十进制数; 在 - 32768～32767 范围内写在指令中。

② 无符号的逻辑数; 在 0～65535 范围内写在指令中。

③ 十六进制数; 用 # 号打头后跟 4 位十六进制数,用在伪指令中。

④ 字符常数; 用单引号括起来写在伪指令中。

⑤ 标号地址常数; 以字母打头后跟字母、数字,6 位以内,写在指令中。

(12) 在计算机中,负数用补码表示,所以 - 1 为

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;共 16 个 1。

而 65535 为无符号数,在计算机中为

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;共 16 个 1。

所以 - 1 和 65535 在计算机中是完全一样的。

【习题 2】

1. (1) ① KSZ ② 80 ③ 1 ④ END
   (2) ① START ② EXIT ③ 7

2. (1) C1 DC 10
   (2) C2 DC -1
   (3) C3 DC #0010
   (4) C4 DC #00AB
   (5) C5 DC 'CHINA'
   (6) C6 DC 'WANG'
   (7) C7 DC DY
   (8) C8 DC BH
   (9) C9 DC #FFFF

3. (1) CSA FFFF
   (2) CSB FFFF (-1 的 16 位补码)
   (3) CSC 8000 (-32768 的补码)
   (4) CSD ABCD
   (5) CSE 0064
   (6) CSF 0064
   (7) CSG 000A
   (8) CSH 0041 (字符 'A' 的 ASCII)
   CSH + 1 0042 (字符 'B' 的 ASCII)
   CSH + 2 0043 (字符 'C' 的 ASCII)
   CSH + 3 0044 (字符 'D' 的 ASCII)
   (9) CSI 0030 (字符 '0' 的 ASCII)
   CSI + 1 0031 (字符 '1' 的 ASCII)
   CSI + 2 0032 (字符 '2' 的 ASCII)
   CSI + 3 0033 (字符 '3' 的 ASCII)

4. (1) 8001 (-32767 的补码)
   (2) 7FFF
   (3) 0058 ('X' 的 ASCII)
   0059 ('Y' 的 ASCII)
   005A

(4) 6002

(5) 以前内容
以前内容
以前内容

(6) 0001

(7) 600C

(8) 6005

(9) 6000

【习题 3】

1．(1) 第一种是利用传送指令:

```
LEA   GR0,0
```

第二种是利用取数指令:

```
LD   GR0,L
…
L    DC   0
```

(2) 利用间接传送指令:

```
LEA   GR1,1,GR1
```

(3) 利用间接传送指令,将 ADR 设定为 - 2:

```
LEA   GR2, - 2,GR2
```

(4) 利用间接传送指令,将 ADR 设定为 0:

```
LEA   GR4,0,GR3
```

(5) 利用间接传送指令,将 GR4 进行“自我传送”:

```
LEA   GR4,0,GR4
```

2．(1) (GR0):FFFF$_H$
(GR1):8000$_H$
(GR2):FFFF$_H$
(GR4):FFFF$_H$

(2) (DY1):0000000000110001 ;字符'1'的 ASCII,字符'23'不被保留。
(DY2):0000000000110010 ;字符'2'的 ASCII,字符'34'不被保留。

(3) (GR1):0001$_H$;(D1 + 0)→GR1,即(D1) = 1→GR1
(GR2):0002$_H$;(E)→GR2,E = D1 + (GR1) = D1 + 1,(D1 + 1) = 2→GR2
(GR3):0003$_H$;(E)→GR3,E = D1 + (GR2) = D1 + 2,(D1 + 2) = 3→GR2

3．1) ① GR1 ② ST ③ B

(2) ① GR1 ② GR2

(3) ① LEA ② DY ③ DZ

4．(1)

```
DYHH   START
       LD     GR1,X          ;(X)→GR1
       LD     GR2,Y          ;(Y)→GR2
```

```
        ST      GR1,Y           ;(GR1) = (X)→Y
        ST      GR2,X           ;(GR2) = (Y)→X
        EXIT
X       DS      1
Y         S     1
        END
```

(2)
```
JCHH    START
        ST      GR1,A           ;(GR1)→A
        ST      GR2,B           ;(GR2)→B
        LD      GR1,B           ;(B) = (GR2)→GR1
        LD      GR2,A           ;(A) = (GR1)→GR2
        EXIT
A       DS      1
B         S     1
        END
```

(3)
```
DYB1    START
        LD      GR1,M           ;(M)→GR1
        LEA     GR1,1,GR1       ;(GR1) + 1→GR1,即(M) + 1→GR1
        ST      GR1,M           ;(M) + 1→M
        LD      GR2,N           ;(N)→GR2
        LEA     GR2,-1,GR2      ;(GR2) - 1→GR2,即(N) - 1→GR2
        ST      GR2,N           ;(N) - 1→N
        EXIT
M       DC      6
N         C     9
        END
```

(4)
```
DYC0    START
        LEA     GR0,0           ;0→GR0。
        LEA     GR1,0           ;变址寄存器初始值为 0。
        ST      GR0,DY,GR1      ;0→DY + (GR1)单元,即 0→DY 单元。
        LEA     GR1,1,GR1       ;变址寄存器加 1,即(GR1) +1→GR1。
        ST      GR0,DY,GR1      ;0→DY + 1 单元。
        LEA     GR1,1,GR1       ;变址寄存器加 1,即(GR1) +1→GR1。
        ST      GR0,DY,GR1      ;0→DY + 2 单元。
        EXIT
DY      DS      3               ;定义以 DY 为首地址的 3 个连续单元。
        END
```

(5)
```
DZM0    START  KS
D1      DS 1                    ;DZ - 2 单元。
D2      DS 1                    ;DZ - 1 单元。
```

```
DZ      DS  1                    ;定义 DZ 单元。
KS       EA   R0,12              ;常数 12→GR0。
        LEA GR1,0                ;0→GR1,变址寄存器初值为 0。
        ST   GR0,DZ,GR1          ;12→DY + 0 单元。
        LEA GR1, - 1,GR1         ;变址寄存器减 1。
        ST   GR0,DZ,GR1          ;0→DZ - 1 单元。
        LEA GR1, - 1,GR1         ;变址寄存器减 1。
        ST   GR0,DY,GR1          ;0→DZ - 2 单元。
        EXIT
        END
```

(6)
```
QHCX    START
        LEA     GR1,0            ;0→GR1
        LEA     GR1,5,GR1        ;5 + (GR1) = 5 + 0→GR1
        LEA     GR1,4,GR1        ;4 + (GR1) = 5 + 4→GR1
        LEA     GR1,3,GR1        ;3 + (GR1) = 5 + 4 + 3→GR1
        LEA     GR1,2,GR1        ;2 + (GR1) = 5 + 4 + 3 + 2→GR1
        LEA     GR1,1,GR1        ;1 + (GR1) = 5 + 4 + 3 + 2 + 1→GR1
        LEA     GR0,0,GR1        ;(GR1)→GR0
        EXIT
        END
```

【习题 4】

1．
```
(1) ADD     GR1,0,GR2
(2) SUB     GR2,2,GR3
(3) SLA     GR3,3
(4) SRA     GR4,4
(5) ADD     GR0,DY,GR1
(6) SUB     GR1,DY,GR3
(7) SLA     GR3,YW,GR4
(8) ADD     GR4,0,GR1
```

2．1)
```
WZBSZ   START
        LEA     GR1,0
        LD      GR0,WZ
        SUB     GR0,CS30
        ST      GR0,SZ
        ADD     GR1,CS1
        LD      GR0,WZ,GR1
        SUB     GR0,CS30
        ST      GRO,SZ,GR1
        ADD     GR1,CS1
        LD      GR0,WZ,GR1
```

```
        SUB     GR0,CS30
        ST      GR0,SZ,GR1
        EXIT
CS30    DC      #0030
CS1     DC      1
SZ      DS      3
WZ      DC      '268'
        END
```

(2)
```
LJAH    START
        LEA     GR1,SDZ         ;基准地址 SDZ→GR1
        LEA     GR0,0           ;累加寄存器充 0
        ADD     GR0,0,GR1       ;累加 SDZ 中的数
        ADD     GR0,1,GR1       ;累加 1 + (GR1) = SDZ + 1 中的数
        ADD     GR0,2,GR1       ;累加 2 + (GR1) = SDZ + 2 中的数
        ADD     GR0,-1,GR1      ;累加 - 1 + (GR1) = SDZ - 1 中的数
        ADD     GR0,-2,GR1      ;累加 - 2 + (GR1) = SDZ - 2 中的数
        EXIT
        DC      #0016
        DC      #0028
SDZ     DC      #0030
        DC      #0080
        DC      #0062
        END
```

3.(1)
```
BHJH    START
        LD      GR1,WS1         ;(WS1)→GR1,即'1'→GR1。
        SLA     GR1,11          ;(GR1)左11→GR1,即'1'的 ASCII 左移 11 位。
        SRA     GR1,11          ;'1'的 ASCII 去掉 0030H 变为 1→GR1。
        LD      GR1,WS1         ;'2'→GR1。
        SLA     GR2,11          ;'2'的 ASCII 左移 11 位。
        SLA     GR2,11          ;'2'的 ASCII 去掉 0030H 变为 2→GR2。
        LEA     GR1,1,GR2       ;1 + (GR2) = 1 + 2→GR1。
        LEA     GR2,-2,GR1      ;- 2 + (GR1) = 1 + 2 - 2 = 1→GR2。
        LEA     GR1,-1,GR1      ;- 1 + (GR1) = 1 + 2 - 1 = 2→GR1。
        EXIT
WS1     DC      '1'
WS2     DC      '2'
        END
```

本程序有两个功能,第一是将数字字符'1'和'2',变为数值 1 和 2,并分别存于 GR1 和 GR2 中;第二是将 GR1 和 GR2 中的值互换。

(2)
```
YSBH    START
        LEA     GR1,ZFS         ;地址 ZFS→GR1
        LD      GR2,ZFS         ;[ | 'A' ] →GR2
        SLA     GR2,8           ;[ 'A' | ] →GR2
        ADD     GR2,1,GR1       ;[ 'A' | 'B' ] →GR2
        ST      GR2,YSS         ;'A'、'B'压缩后→YSS
        LD      GR2,2,GR1       ;'1'→GR2
        SLA     GR2,12          ;[ 1 | | | ] →GR2
        ST      GR2,YSDY        ;[ 1 | | | ] →YSDY
        LD      GR2,3,GR1       ;'2'→GR2
        SLA     GR2,12          ;去掉 0030H
        SRA     GR2,4           ;[ | 2 | | ] →GR2
        ADD     GR2,YSDY        ;[ 1 | 2 | | ] →GR2
        ST      GR2,YSDY        ;[ 1 | 2 | | ] →YSDY
        LD      GR2,4,GR1       ;'3'→GR2
        SLA     GR2,12          ;去掉 0030H
        SRA     GR2,8           ;[ | | 3 | ]
        ADD     GR2,YSDY        ;[ 1 | 2 | 3 | ] →GR2
        ST      GR2,YSDY        ;[ 1 | 2 | 3 | ] →YSDY
        LD      GR2,5,GR1       ;'4'→GR2
        SLA     GR2,12          ;去掉 0030H
        SRA     GR2,12          ;[ | | | 4 ] →GR2
        ADD     GR2,YSDY        ;[ 1 | 2 | 3 | 4 ] →GR2
        ST      GR2,YSDY        ;[ 1 | 2 | 3 | 4 ] →YSDY
        EXIT
ZFS     DC      'AB1234'
YSS     DS      1
YSDY    DS      1
        END
```

本程序是对字符常数进行处理的程序,执行本程序可将'AB1234'中的'A'和'B'压缩在一个单元 YSS 中,变为 ['A' 'B']; 还将'1234'分别数值化后压缩在同一个单元 YSDY 中。

4. 1)
```
JCQH    START
        ST      GR2,JH          (GR2)→JH
        ADD     GR3,JH          ;(GR3) + (GR2)→GR3
        ST      GR3,JH          ;(GR3) + (GR2)→JH
        ADD     GR1,JH          ;(GR1) + (GR3) + (GR2)→GR1
        LEA     GR0,0,GR1       ;(GR1)→GR0
        EXIT
```

```
        JH      DS   1
                END
```

(2)
```
Q7A     START
        LD      GR1,A           ;(A)→GR1
        ST      GR1,B           ;(A)→B
        SLA     GR1,3           ;8×(A)→GR1
        SUB     GR1,B           ;8×(A) - (A) = 7×(A)→GR1
        ST      R1,B            ;7×(A)→B
        EXIT
A       DS      1
B       DS      1
        END
```

(3)
```
CHC     START
        LD      GR1,X           (X)→GR1
        SUB     GR1,Y           ;(X) + (Y)→GR1
        ADD     GR1,Z           ;(X) + (Y) + (Z)→GR1
        LEA     R0, - 3,GR1     ; - 3 + (GR1) = (GR1) - 3→GR0
        EXIT
X       DS      1
Y       DS      1
Z       DS      1
        END
```

【习题 5】

1.(1)
```
TEST1   START
        LEA     GR1,16          ;16→GR1
        SLL     GR1,1           ;(GR1)左6 →GR1;(GR1) = 2^5
        LEA     GR1,16,GR1      ;2^5 + 2^4
        ADD     GR1,A           ;2^5 + 2^4 + 6
        ST      R1,B            ;0110110 = '6'→B
        EXIT
A       DC      6
B       DS      1
        END
```

本程序的功能是形成字符'6',即得到'6'的 ASCII。

(2)
```
TEST2   START
        LEA     GR2,32          ;32→GR2
        ST      GR2,C           ;32→C;即 2^5 →C
        SRL     GR2,1           ;16→GR2;即 2^4 →GR2
        OR      GR2,C           ;2^5 + 2^4 = 110000→GR2
        OR      GR2,D           ;拼为 111000→GR2
```

```
        ST      R2,E        ;0111000→E;即'8'→E
        EXIT
C       DS      1
D       DC      8
E       DS      1
        END
```

(3)
```
TEST3   START
        LEA     GR3,1       ;1→GR3
        SLL     GR3,3       ;2^3→GR3
        ST      GR3,M       ;2^3→GR3
        SLL     GR3,6       ;2^9→GR3
        OR      GR3,M       ;2^9 + 2^3→GR3
        ST      GR3,M       ;2^9 + 2^3→GR3
        SLL     GR3,3       ;2^12 + 2^6→GR3
        OR      GR3,M       ;2^12 + 2^9 + 2^6 + 2^3→GR3
        ST      GR3,M       ;0001001001001000→M
        EXIT
M       DS      1           ;存放压缩数据 1234H 的单元
        END
```

2.(1)
```
TK1     START
        LD      GR0,MS      (MS)→GR0
        SLL     GR0,2       ;4×MS
        ADD     GR0,MS      ;4×MS+MS
        SLL     GR0,1       ;(4×MS+MS)×2
        EXIT
MS      DC      6
        END
```

(2)
```
TK2     START
        LD      GR0,MS
        SLL     R0,1        ;MS×2
        ST      GR0,DY      ;2×MS→DY
        SLL     GR0,2       ;(2×MS)×4=8MS→GR0
        ADD     R0,DY       ;2×MS+8×MS
        EXIT
MS      DC      6
DY      DS      1
        END
```

(3)
```
TL3     START
        LD      GR1,W1      ;取'A'
        SLL     GR1,8       ;左移8位
```

```
        OR      GR1,W2          ;'A'和'B'拼在一起
        ST      GR1,YS          ;'A'、'B'压缩在YS中
        EXIT
W1      DC      'A'
W2      DC      'B'
YS      DS      1
        END
```

(4)
```
TK4     START
        LD      GR1,YS          ;(YS)→GR1
        SRL     GR1,8           ;取高8位
        ST      GR1,AS          ;(YS)的高8位→AS的低8位
        LD      GR1,YS          ;(YS)→GR1
        SLL     GR1,8           ;取(YS)的低8位
        SRL     GR1,8           ;移至低8位
        ST      R1,BS           ;(YS)的低8位→BS的低8位
        EXIT
YS      DC      #4142
AS      DS      1
BS      DS      1
        END
```

3.(1)
```
CXCH    START
        LD      GR1,DY          ;(DY)→GR1
        SLL     GR1,8           ;低8位移到高8位
        ST      GR1,WK
        LD      GR1,DY
        SRL     GR1,8           ;高8位移到低8位
        OR      GR1,WK          ;拼位
        ST      R1,DY
        EXIT
DY      DS      1
WK      DS      1
        END
```

(2)
```
FH      START
        LD      GR2,YS          1234=0001001000110100→GR2
        SRL     GR2,12          ;1→低4位
        SLL     GR2,4           ;1→低8位
        ST      GR2,A           ;1→A的低8位
        LD      GR2,YS          ;1234→GR2
        SLL     GR2,8           ;3→高4位
        SRL     GR2,12          ;3→低4位
```

```
        OR      GR2,A          ;13 合并
        ST      GR2,A          ;13→A
        LD      GR2,YS         ; 1234→GR2
        SLL     GR2,12         ;4→高 4 位
        SRL     GR2,12         ;4→低 4 位
        ST      GR2,B          ;4→B 的低 4 位
        LD      GR2,YS         ;1234→GR2
        SLL     GR2,4          ;2→高 4 位
        SRL     GR2,12         ;2→低 4 位
        SLL     GR2,4          ;2→低 8 位
        OR      GR2,B          ;24 合并
        ST      R2,B           ;24→B
        EXIT
YS      DS      1
        END
```

(3)

```
BM      TART
        LD      GR3,FYM        ;取负数原码
        EOR     GR3,QFM        ;变反码
        LEA     GR3,1,GR3      ;低位加 1
        ST      R3,FBM         ;负数的补码→FBM
        EXIT
FYM     DS      1
FBM     DS      1
QFM     DC      #7FFF
        END
```

(4)

```
BAS     START
        LD      GR4,A          ;取 A
        SLL     GR4,1          ;2A
        ADD     GR4,A          ;3A
        SLL     GR4,1          ;6A
        ST      GR4,B          ;6A→B
        SRL     GR4,1          ;3A
        ADD     GR4,B          ;9A
        ADD     GR4,B          ;6A + 9A
        SLL     GR4,1          ;(6A + 9A) ×2
        ST      R4,A           ;A 的 30 倍→A
        EXIT
A       DS      1
B       DS      1
        END
```

【习题 6】

1.(1)
```
GRZY    START
        LEA     GR1,0,GR1       ;由 GR1 形成标志寄存器状态
        JMI     FZ              ;负转
        JZE     LZ              ;零转
        JMP     FL              非负、非零转
FZ      LEA     GR0,-1          ;-1→GR0
        JMP     WL              ;转结束
LZ      LEA     GR0,0           ;0→GR0
        JMP     WL              ;转结束
FFZ     LEA     GR0,1           ;1→GR0
WL      EXIT
        END
```

(2)
```
GBJ     START
        ST      GR1,MAX         ;(GR1)假定为大值→MAX
        CPA     GR2,MAX         ;(GR2)～(GR1)
        JMI     JS              ;(GR2)<(GR1)转,大则下
        ST      R2,MAX          ;(GR2)→MAX
JS      EXIT
MAX     DS      1
        END
```

2.(1)
```
TYFZ    START
        EOR     GR2,DY          ;(GR2)"(DY)
        JPZ     TH              ;两者符号相同转
        JMI     H               两者符号不同转
TH      LEA     GR1,1           ;1→GR1
        JMP     TZ
YH      LEA     GR1,2           ;2→GR1
        EXIT
DY      DS      1
        END
```

本程序的功能为,若寄存器 GR2 和单元 DY 中的值同号,则在寄存器 GR1 中送 1;若两者符号不同,则在 GR2 中送 2。

(2)
```
SZCL    START
        LEA     GR1,0           ;0→GR1,变址寄存器初值
        LEA     R2,3            ;3→GR2,次数
JXZ     LD      GR3,BH,GR1      ;取标号→GR3
        JMP     0,GR3           ;按 GR3 中的标号转
```

```
ACL     LD      GR4,DY      ;(DY)→GR4
        SRA     GR4,1       ;1/2
        ST      GR4,A       ;(DY)的1/2→A
        JMP     XG
BCL     LD      GR4,DY      ;(DY)→GR4
        SRA     GR4,2       ;1/4
        ST      GR4,B       ;(DY)的1/4→B
        JMP     XG
CCL     LD      GR4,DY      ;(DY)→GR4
        SRA     GR4,3       ;1/8
        ST      GR4,C       ;(DY)的1/8→C
XG      LEA     GR2,-1,GR2  ;次数减1
        JNZ     JXZ         ;不为0转
        EXIT
DY      DS      1
BH      DC      ACL         ;处理1入口地址
        DC      BCL         ;处理2入口地址
        DC      CCL         ;处理3入口地址
A       DS      1
B       DS      1
C       DS      1
        END
```

本程序的功能是求单元DY中值的1/2,1/4,1/8,分别存入A,B,C单元。从逻辑结构上看,当GR1中的值为0、1、2时分别转入不同的入口进行3种不同的处理。就3种处理本身只是一种示意,重要的是3种入口地址的形成对于程序设计更重要。

3.(1)
```
DXS     START
        ST      GR1,D
        CPA     R2,D        (GR2)～(GR1)
        JMI     CX          ;(GR2)<(GR1)转
        ST      GR2,D       ;大数→D
        ST      R1,X        小数→X
        JMP     WL
CX      ST      GR2,X       ;小数→X,大数已在D
        EXIT
D       DS      1
X       DS      1
        END
```

(2)
```
SSB     START
        LD      GR1,A
        CPA     R1,B        A～B
```

```
        JMI     BBC
        CPA     GR1,C           ;A～C
        JMI     CD
        ST      R1,D            A大
        JMP     WL
BBC     LD      GR2,B
        CPA     GR2,C           ;B～C
        JMI     CD
        ST      GR2,D           ;B大
        JMP     WL
CD      LD      GR3,C
        ST      GR3,D           ;C大
        EXIT
A       DS      1
B       DS      1
C       DS      1
D       DS      1
        END
```

(3)
```
TJWS    START
        LEA     GR4,0           ;个数累加器置初值
        LEA     R3,16           ;变址,移位位数控制
JX      LEA     GR2,0,GR1       ;(GR1)→GR2工作寄存器
        SLL     GR2,-1,GR3      ;左移去高位
        SRL     GR2,15          ;右移至最低位
        ST      GR2,W           ;一位→W
        ADD     GR4,W           ;逐位累加
        LEA     GR3,-1,GR3      ;移位位数修改
        JNZ     JX              ;(GR3)非零转
        ST      GR4,GS          ;1的个数→GS
        EXIT
W       DS      1
GS      DS      1
        END
```

(4)
```
ZXZ     START
        LEA     GR1,0           ;0→GR1计数器置初值
        LD      R3,DY           假定的最小值→GR4
JX      CPA     GR3,DY,GR1      ;比较
        JMI     XG              ;转修改
        LD      GR3,DY,GR1      ;置换最小值
XG      LEA     GR1,1,GR1       ;计数器加1
        CPL     GR1,SS          ;是否为30次
        JNZ     JX              ;未比较完继续
        EXIT
```

```
        DY      DS      30
        SS      DC      30
                END
```

(5)
```
ZFLJ    START
        LEA     GR4,99          ;操作次数,地址偏移量→GR4
        LEA     GR3,0           ;正数计数器→GR3
        LEA     GR2,0           ;负数计数器→GR2
        LEA     GR1,0           ;零数据计数器→GR1
        LEA     R0,0            ;比较基准值0→GR0
JX      CPA     GR0,ST,GR4      ;数据与零比较
        JMI     FZ              ;负转
        JZE     LZ              ;零转
        LEA     GR3,1,GR3       ;正数计数
        JMP     XG
FZ      LEA     GR2,1,GR2       ;负数计数
        JMP     XG
LZ      LEA     GR1,1,GR1       ;零计数
XG      LEA     GR4,-1,GR4      ;操作次数,变址减1
        JPZ     JX
        EXIT
ST      DS      100
        END
```

【习题 7】

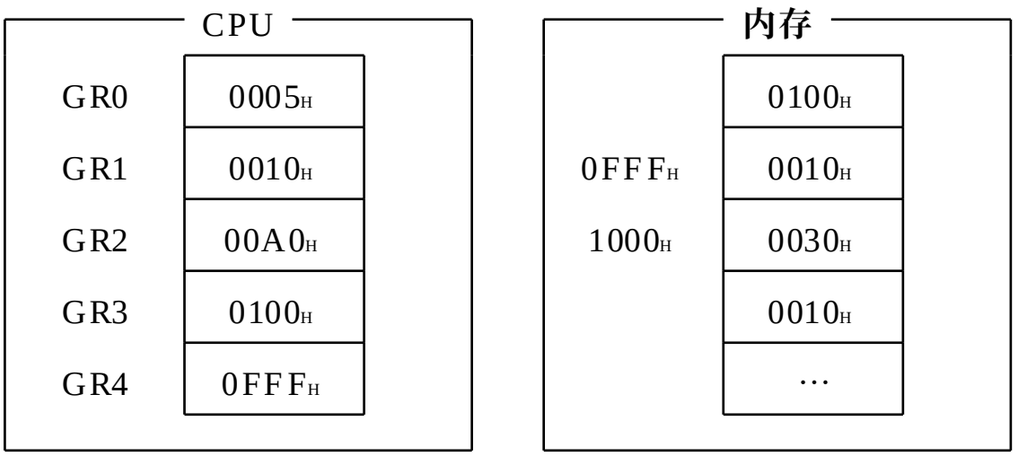
1．1) PUSH 1626
(2) PUSH #FFFF;PUSH -1;PUSH 65535
(3) PUSH ABC
(4) PUSH 0,GR1
(5) PUSH -6,GR2
(6) PUSH 81,GR3

2．1) LEA GR4,-1,GR4
(2) LEA GR4,1,GR4
(3) ST GR0,0,GR4
(4) LD GR1,0,GR4
(5) LD GR2,-4,GR4

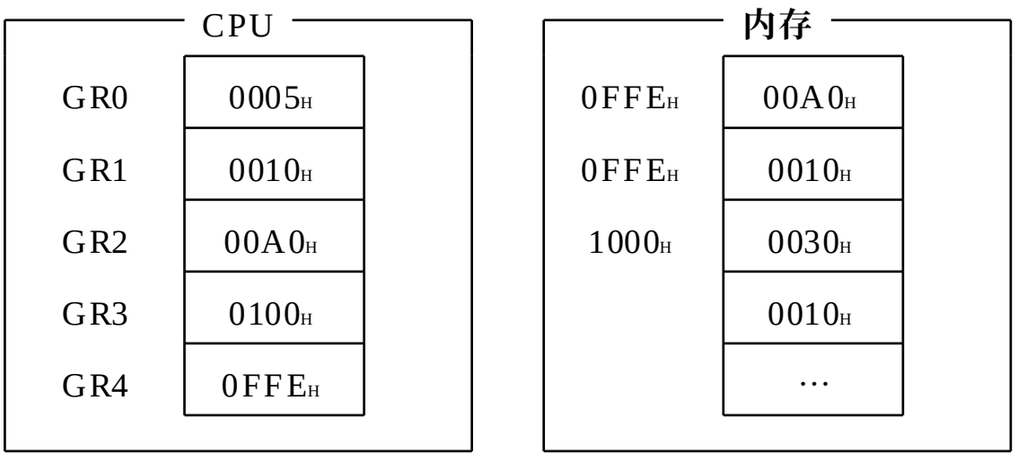
3．1) USH 62
POP GR1
(2) USH 0,GR2
POP GR3
(3) USH 0,GR1

```
PUSH    0,GR2
POP     GR1
POP     GR2
```

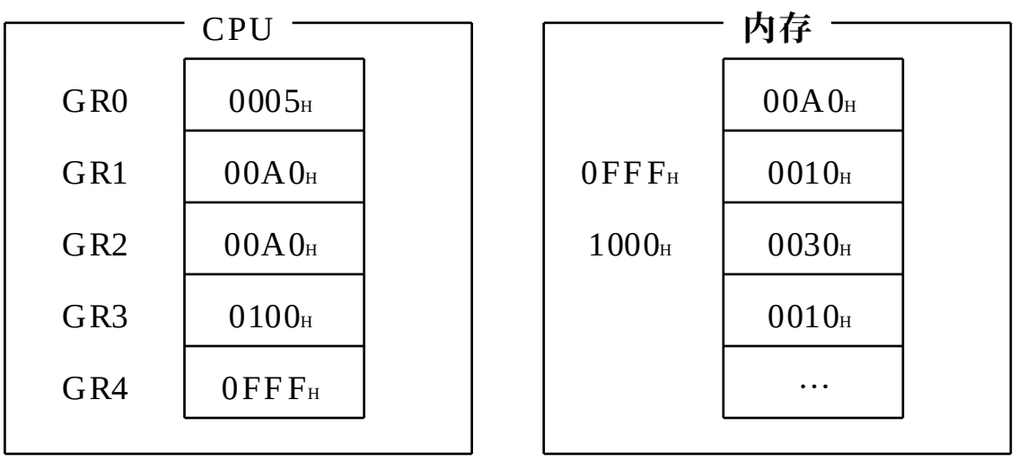
4.(1) **指令①执行后**



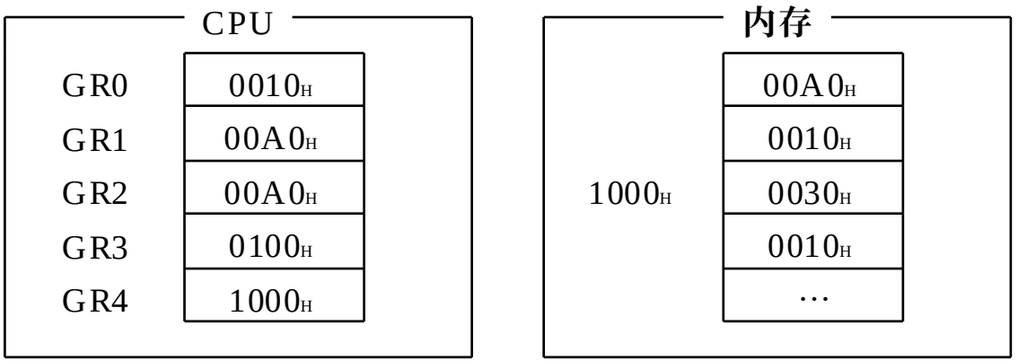
(2) **指令②执行后**



(3) **指令③执行后**



(4) **指令④执行后**



5．(1) **指令⑤执行后**：

(GR4)：FFFCH →

FFFC	20
FFFD	10
FFFE	-1
FFFF	?
地址	**单元内容**
数据栈	

=0014H
=000AH
=FFFFH

(2) **指令⑥执行后**：

(CALL **指令的下一地址进栈**)→

(GR4)：000EH

(PC)：0016

FFFB	000E
FFFC	20
FFFD	10
FFFE	-1
FFFF	?
地址	**单元内容**
数据栈	

=0014H
=000AH
=FFFFH

(3) **指令 2 执行后**：

(**执行** RET **返回地址出栈**)

(GR4)：FFFCH

(PC)：000E

→

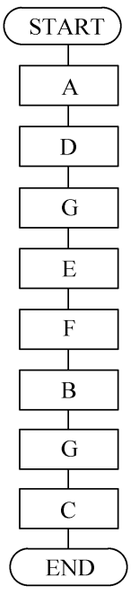
FFFB	000E
FFFC	20
FFFD	10
FFFE	-1
FFFF	?
地址	**单元内容**
数据栈	

=0014H
=000AH
=FFFFH

(4)

(GR0):	-1 = FFF$_H$
(GR1):	10 = 0 0 0 A$_H$
(GR2):	20 = 0 0 1 4$_H$

6．



7．(1)

```
JCS     START
        PUSH    0,GR0
        PUSH    0,GR1
        PUSH    0,GR2
        PUSH    0,GR3
        POP     GR0             ;(GR3)→GR0
        POP     GR3             ;(GR2)→GR3
        POP     GR2             ;(GR1)→GR2
        POP     GR1             ;(GR0)→GR1
        EXIT
        END
```

栈
(GR3)
(GR2)
(GR1)
(GR0)
…

(2)

```
QH      START
        LEA     GR0,0           ;0→GR0 累加器
        LEA     GR1,8           ;8→GR1
        CALL    LJ              ;转子计算 1 + 2 + … + 8 = H1
        ST      GR2,W           ;
        ADD     GR0,W           ;H1 →GR0
```

```
        LEA     GR1,12          ;12→GR1
        CALL    LJ              ;转子计算 1 + 2 + … + 12 = H2
        ST      GR2,W           ; H2 →W
        ADD     GR0,W           ; H1 + H2 →GR0
        LEA     GR1,18          ;18→GR1
        CALL    LJ              ;转子计算 1 + 2 + … + 18 = H3
        ST      GR2,W           ; H3 →W
        ADD     GR0,W           ; H1 + H2 + H3 →GR0
        ST      R0,H            H1 + H2 + H3
        EXIT
LJ      LEA     GR2,0           ;0→GR2 累加器
JX      ST      GR1,W           ;自然数→W
        ADD     GR2,W           ;累加
        LEA     GR1, - 1,GR1    ;形成 N 个自然数
        JNZ     JX
        RET
W       DS      1
H       DS      1
        END
```

【习题 8】

1 . (1) 该程序的功能为:检查输入的字符,若多于 3 个则输出错误标志“ ERROR ”。

(2) 指令①、②、③完成为单元 DY 的内容加常数 10。

指令④是为寄存器 GR2 加常数 10。

指令⑤、⑥为寄存器 GR3 加常数 10。

(3) 将寄存器 GR1 中的数反号,例如 1 变 - 1; - 1 变 1。

(4) 将单元 DY 中的内容反号。

(5) 将以 ZDY 为首地址的 50 个单元中的内容,采用正计数从第 1 个单元开始到第 50 个为止求累加和。

(6) 将以 FDY 为首地址的 50 个单元中的内容,采用倒计数,从最后一个单元开始到第一个单元为止进行累加求和。

2 . (1)

```
QH      START
        LEA     GR1,DATA        单元首地址→GR1
        LD      GR2,ZS          ;取累加个数 6→GR2
        CALL    K               转子累加
        EXIT
ZS      DC      6               ;总数
DATA    DC      87              ;以 DATA 为单元首地址中的求和数据
        DC      92
        DC      69
        DC      84
```

```
        DC      73
        DC      86
ZK      LEA     GR0,0           ;累加器初值 0→GR0
JX      LEA     GR2, -1,GR2     ;个数减 1
        JMI     FH
        ADD     GR0,0,GR1       ;累加和→GR0
        LEA     GR1,1,GR1       ;修改累加单元地址
        JMP     JX
FH      RET
        END
```

(2)

```
ZH2     START
        LEA     GR1,0           ;累加器初值 0→GR1
        LEA     GR2,0           ;个数计数器初值 0→GR2
        CPA     GR2,ZS          ;已达到累加个数否 ?
        JZE     BC              ;相等转保存结果
JX      ADD     GR1,DY,GR2
        LEA     GR2,1,GR2       ;修改
        CPA     GR2,ZS          ;是否到 30 次
        JMI     JX              ;未到则继续
BC      ST      GR1,ZH
        EXIT
ZS      DS      1               ;放个数单元
ZH      DS      1               ;保存总和单元
DY      DS      30              ;30 个数据单元
        END
```

3.(1)

```
DX      START
        LEA     GR1,0           ;0→GR1 计数、变址寄存器
JX1     LD      GR2,ZM,GR1      ;取第一个字母……
        PUSH    0,GR2           ;字母 Z 进栈
        LEA     GR1,1,GR1       ;修改
        CPA     GR1,C26
        JMI     JX1
        LEA     GR1,0           ;0→GR1 计数、变址寄存器
JX2     POP     GR2
        ST      GR2,ZM,GR1
        LEA     GR1,1,GR1
        CPA     GR1,C26
        JMI     JX2
        EXIT
ZM      DS      26
        END
```

```
(2) PF      START
            LEA     GR1,0           ;0→GR1 变址 1
    JZ9     LEA     GR2,1,GR1       ;1→GR2 变址 2;2→GR2;3→GR2…
            LD      GR3,PF,GR1      ;第一个分数
    JZ1     CPA     GR3,PF,GR2      ;第一个与第二个分数比较
            JPZ     BD              ;数 1≥数 2 不动
            LD      GR0,PF,GR2
            ST      GR0,PF,GR1      ;数 2 到数 1
            ST      GR3,PF,GR2      ;数 1 到数 2
    BD      LEA     GR2,2,GR2       ;修改比较次数
            CPA     GR2,C10         ;是否比较完
            JMI     JZ1             ;未完转
            ADD     GR1,C1          ;修改求极值次数
            CPA     GR1,C9          ;是否做完 9 次求极值
            JMI     JZ9
            LEA     GR1,8
            LEA     GR0,0
    QH      ADD     GR0,PF,GR1      ;累加第 9 个分数,第 8 个,第 7 个…。
                                    ;去掉最低分。
            LEA     GR1, -1,GR1     ;修改
            JNZ     QH              ;累加到最高分停止,未到则继续求和
            ST      GR0,ZF          ;总分→ZF
            SRA     GR0,3           ;平均分,即总分除以 8
            ST      GR0,PJ          ;平均分→PJ
            EXIT
    PF      DS      10              ;10 个评委的评分
    C1      DC      1
    C9      DC      9
    C10     DC      10
    ZF      DC      1
    PJ      DC      1
            END
```

本题的设计思想为:

① 先将 10 个评委的打分由大到小排序。

② 对于排好序的分数进行累加,累加不计第 1 个分数,即去掉一个最高分;也不计最后一个分数,即去掉一个最低分。8 个分数累加得到总分。

③ 将总分右移 3 位,即除 8 得到平均分。

第二种做法是将 10 个分数求和,在求和过程中求一个极大值和一个极小值。由 10 个分数总和减去一个极大值,再减去一个极小值而得到总分,将总分右移 3 位得到平均分。

```
(3) PPS     START
            LEA     GR1,1
            LEA     GR2,1
            ST      GR1,PP          ;第1项,1→PP单元
            ST      GR2,PP,GR1      ;第2项,1→PP+1
            LEA     GR4,0
            LEA     GR3,1
JX          LD      GR0,PP,GR4      ;取第1项
            ADD     GR0,PP,GR3      ;第1项加第2项
            LEA     GR3,1,GR3       ;修改
            ST      GR0,PP,GR3      ;前两项和送后项
            LEA     GR4,1,GR4       ;修改
            CPL     GR4,C6
            JMI     JX
            EXIT
PP          DS      8               ;8个裴波那契数单元
C6          DC      6
            END
```

裴波那契数的特点是,从第3项开始,后项为前两项之和。按照该规律进行程序设计是较容易的。