# ICL-2900系列介绍

（上册）



上海市计算技术研究所印

# ICL-2900系列介绍

上册

上海市計标技术研究所印

# 前　言

在毛主席的革命路线指引下，无产阶级文化大革命以来，我国工人阶级坚决贯彻执行"独立自主，自力更生"的方针，电子计算机从无到有，从小到大，从单一机种发展到多机种，为我国的经济建设，国防建设和科学研究提供了有力的计算工具。

随着计算机的推广普及和软件的迅速发展，对计算机的要求也越来越高，为了提高计算机的平均使用效率（即硬件和软件之总体的效率），以软件和硬件这两大部分需要密切配合来设计计算机系统是一个重要的途径，英国ICL公司（国际计算机有限公司）的2900系列在这方面作了一定的努力，在设计系统时较好地处理了系统的优化问题，引进了扩名字地址，说明符，虚机器等思想，采用了虚拟存贮器，环保护，高速缓冲存贮器，差错管理等技术，使得使用方便，程序可重入，向上可扩展性等等，对我们具有一定的参考价值。

但是原书目的是宣扬他们的机器特点，说成"2000——你的未来系统"，是为ICL公司服务的，以便与IBM公司（美国的国际商业机器公司）竞争，我们TQ-20联合设计组遵照"洋为中用"的原则，将它译出，以供工作参考之用，由于时间匆促，水平所限，难免有所差错，再者术语词汇的译名未必妥当，希读者加以指正。

本资料由上海市计算技术研究所印发

上海市无线电十三厂
TQ－20联合设计组　上海市计算技术研究所
上海科技大学

一九七六年四月

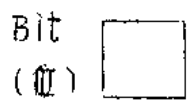
# 目　录

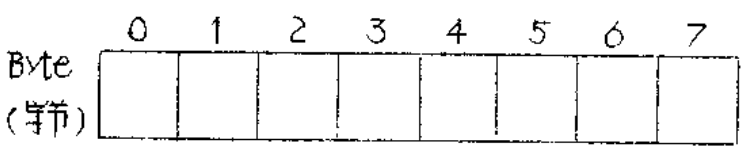
（上　册）

# 第一章　机器的基本构造

## 1.1　§ 1.1 微处机器的基本构造

我们可以说内存储器是由"字节"组成，一个字节，可能你们已知道，是8位指令的二进位。我们可以存取单个字节，或任意数目相连的字节——即字符串。"字"是由四个相连的字节组成——换句话说，即是32个二进位。



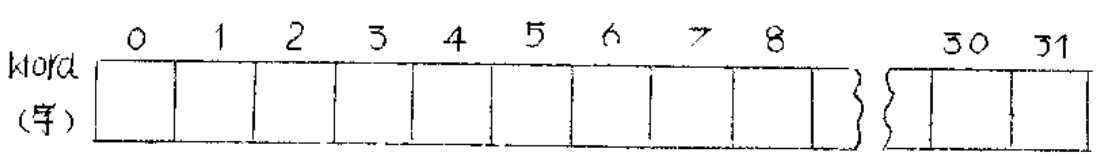




图 1.1

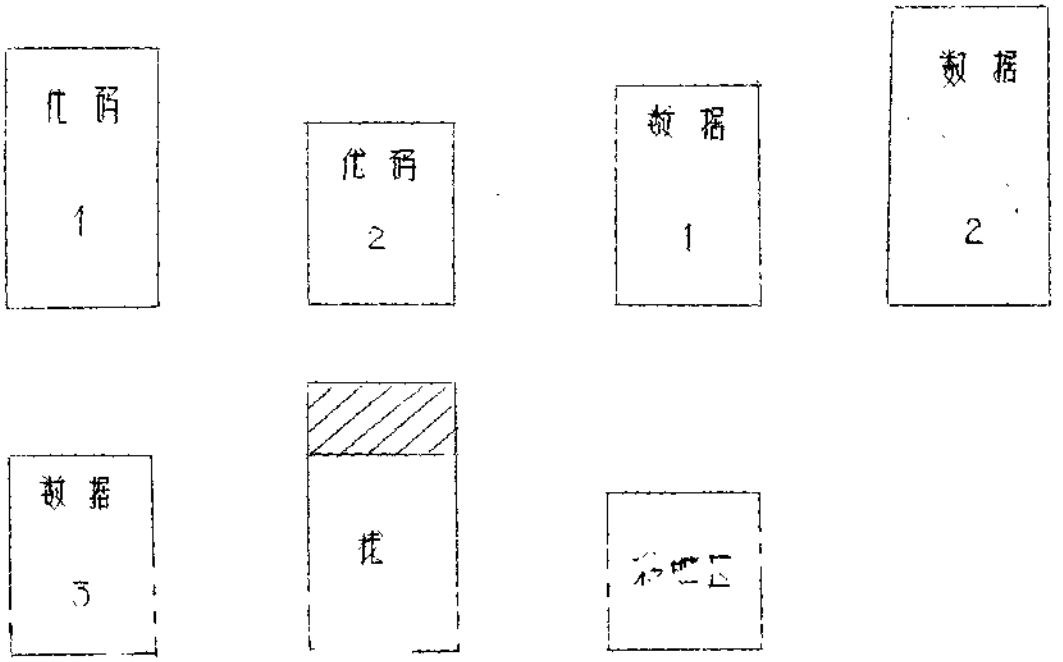


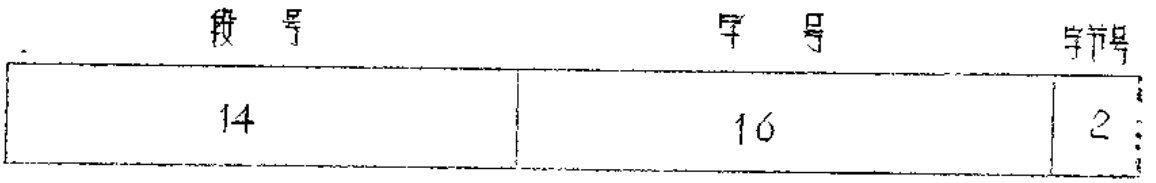

图 1.2

用新系列的术语来说，段就是一块存储区，如果你需要取指或存储一字一还是一段相邻的区域，它可以含有代码（程序），或者含有数据，但通常一段里面不同时包含二者。

现在定义的段与过去所用的段的含义有着全然不同的意义。

按 Fortran 或 1900 plan 的术语，它用来划出一个封闭的程序片，模向范围。它是一个编译单位，既含指令又有数据。现在流行把这样的一个单位叫做模块（module），用另一种说法，Fortran或plan术语中的段的概念，和新系列机的段则是一种硬件的概念。它关心的是内存的区域，即能由硬件来辨认的不同区域。因此，通常实现一道程序将会有一个或几个代码段，任意个（大于或等于0）数据段，一个堆栈段，还有几个别的段——这里可以指出一个，我们称之为秘区，以后再详细讲它——予过程将有些程序人员不必知道的，即他的程序所这不到的辅助区。

1.3 我们来看图 1.3 新系列机的"地址"为 32 位，它




32 位地址格式

图 1.3

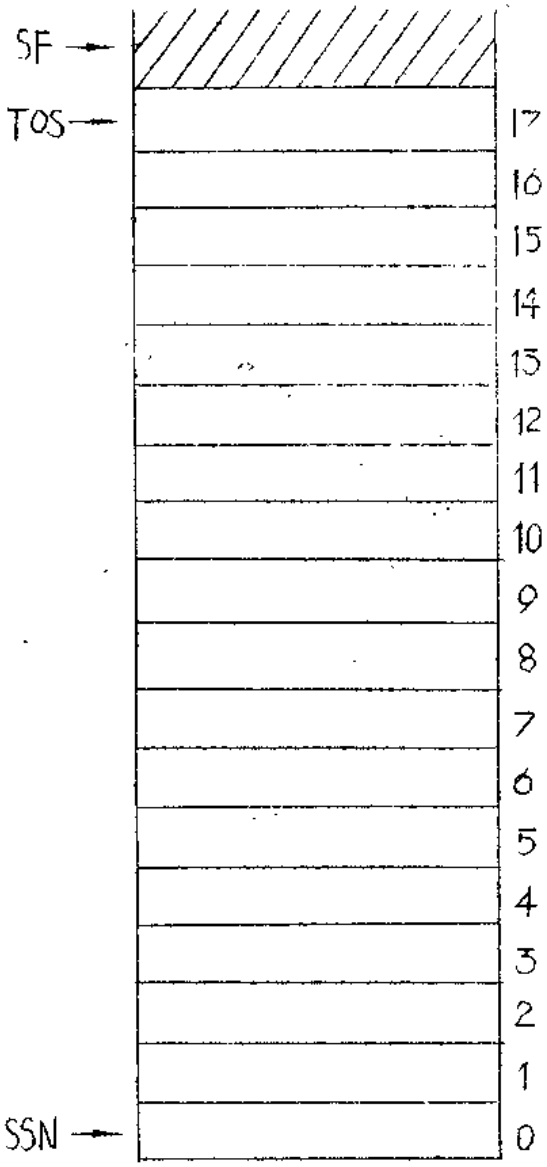
分成段号，占最高14位；字号——接下来的16位，还有2位字节号——即一个字有4个字节。

由于段号有14位，故最多可以有16K段，字号（原文译为字节号——译注）占16位，故每段可有64K字，或256K字节。

1.4 按台机器是建立在使用栈的基础上的，见图 1.4 它的下推栈或叫栈，按先进后出原则操作。栈上每个存贮单元是一个字。

栈高是一个硬件寄存器——它是一个指针，指向栈中第一个尚未用的字，若栈底为0，则寄存器 ST 的内容就等于栈中的字

的了数；对于 1.4 叁，SF 的内容是 18。




18 了字的栈

图 1.4

当一了字被送上栈顶（缩写为 TOS），栈高寄存器就自动加 1。同样，如果一了双字被送上栈顶，SF就自动加 2。我们把"把字送上栈顶"这项动作叫做进栈（Stacking）。同样，如果有一了字退栈，即从栈顶上被拿掉，SF就自动减 1。

栈底是另由一了叫做"栈段号"（Stack segment number）的硬件寄存器确定的。"栈段号"SSN有 14 位。"栈高"SF有 16 位。

1.5 从叁 1.5 里，可以看到我们怎样得到栈高的整了 32 位地址——整了 32 位字节地址。

SSN 及 SF 这两了寄存器是连在一起的，其最低二位是二了。用以给出一了 32 位的按字排齐的（Word-aligned）字节地址。

另一了和栈有关的寄存器是"局部各字基址"LNB。这是类

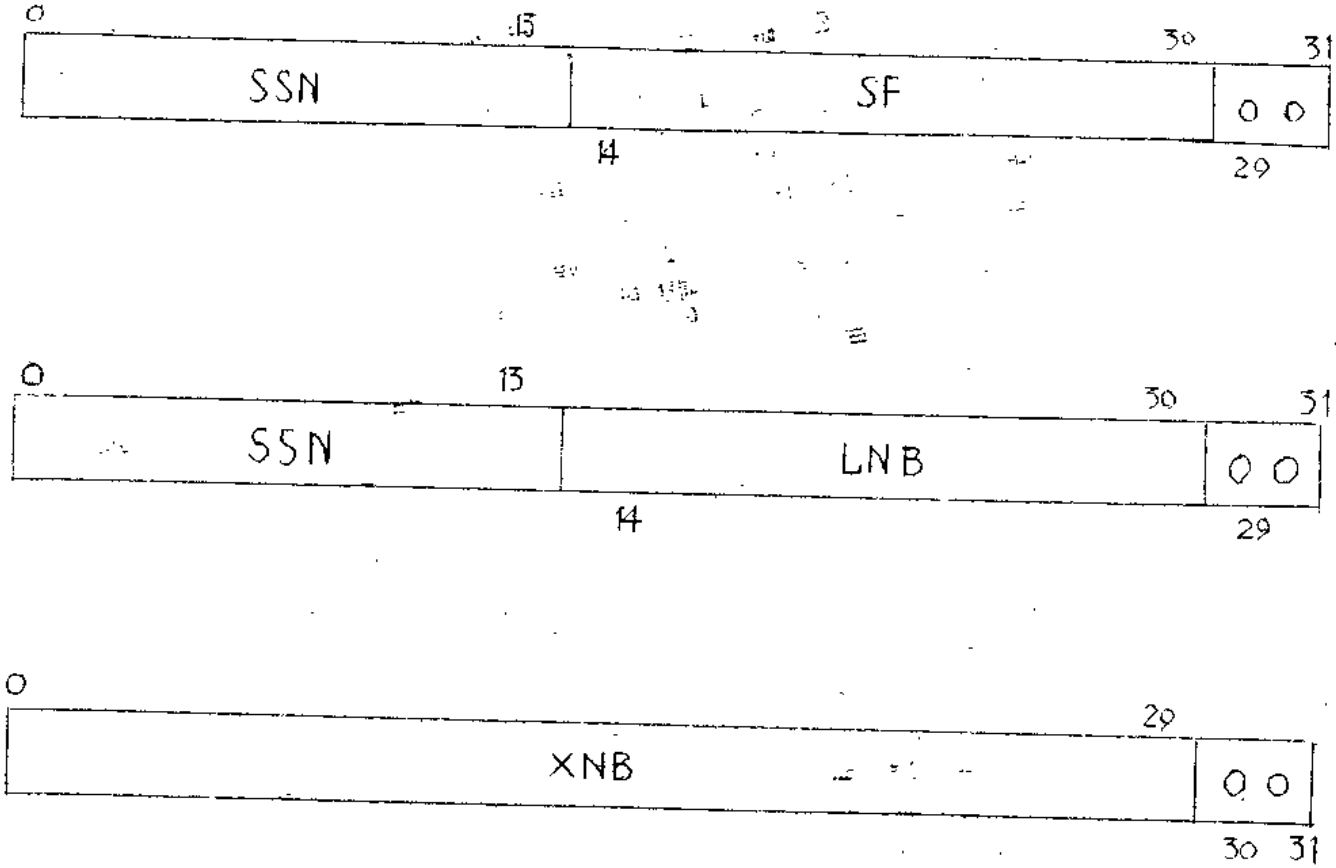



图 1.5

似SF的另一个硬件寄存器，它也是16位，也跟SSN连在一起，以同样的方式给出一个字节地址。

"局部名字基址" LNB，是给软件用来区别我们把它叫做局部名字空间的栈区的，这是一个过程（procedure）的工作空间。"局部名字基址"寄存器必须指向栈的内部。这就是说LNB的内容永远要小于SF的内容，只要LNB或SF一改变，硬件就要检查LNB是否小于SF。

另外还有一个用作栈针的寄存器，叫做外名字基址 XNB，XNB和有不同，它有30位。连同最低2位$_0$在一起，给出了一个完整的32位的按字排齐的字节地址。

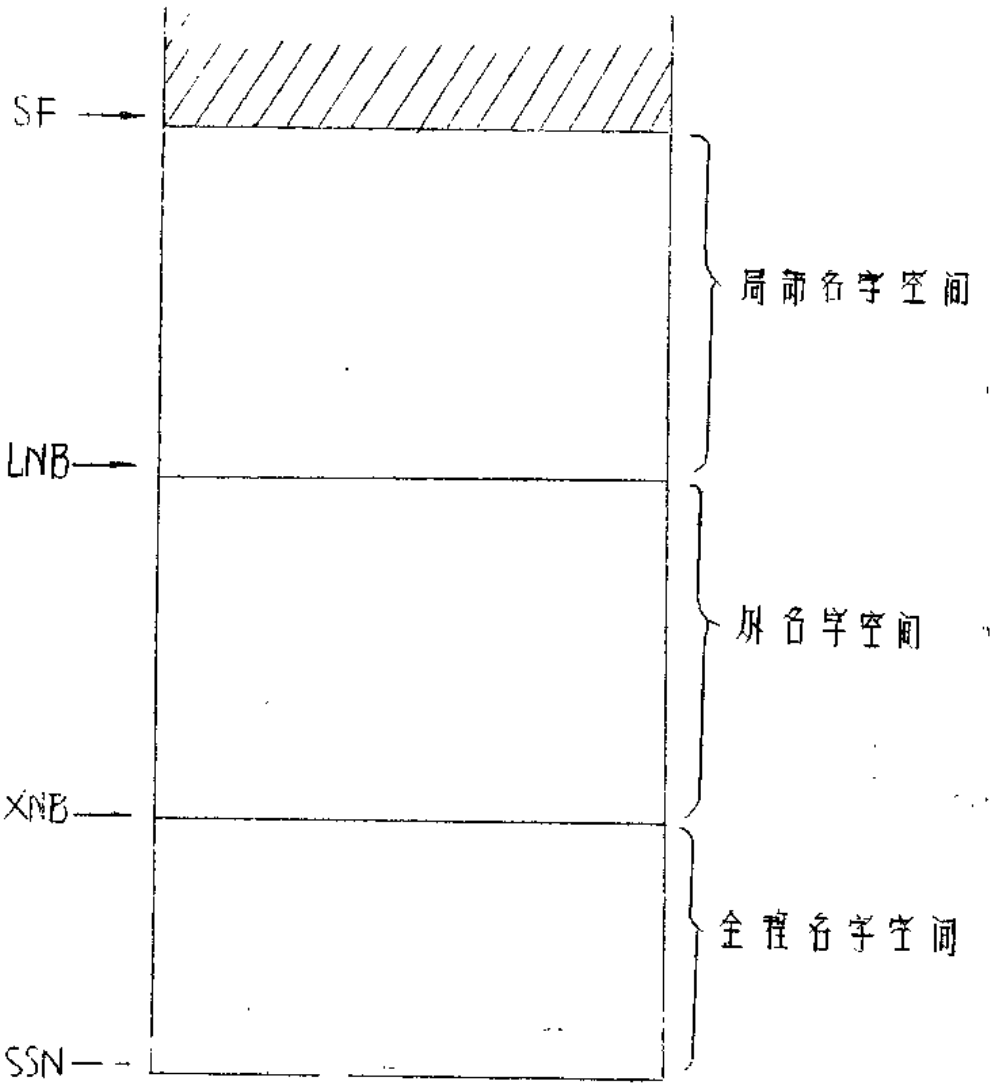
由于XNB有30位，高位14位，它确定了段号，可以跟SSN的内容一样；当然，如果是这样的话，那末XNB便是指向栈内的，但它也不一定要等于栈段号，因此XNB还可用来指向任何一段。ICL的各软件小组对XNB的用法将会各不相同。例如，Algol小组即编制Algol编译程序的小组——将用XNB指向栈内，即指向所谓"词法外层"（enclosing lexical levels）。预测的小组则用XNB指示测的区域，寄执器中其它的段。

1.6 在图1.6的栈图表中XNB是指向栈内的，当然LNB确定了局部名字空间；XNB确定了外名字空间；而SSN即栈底，则规定了全程名字空间，这个部分是打算用于对整个程序公用的数据的。

用于寻址的还有二个硬件寄存器，一个是PC即程序计数器。PC指向的是正在执行的指令的字节地址，PC是按半字排列的，因为每条指令都是以半字边界开始的。一条指令可以是半字长或者是全字长。PC有31位，为取得字节地址，最低边拼上一个0。

B是变址寄存器，还可做修改用，它有全部32位。

1.7 从被下来的图1.7可以看到所有这些寄存器的排列法。它首先给出基本的地址格式——段，字，字节




带有指针的栈

图 1.6

SF ——确定到栈段中的某个字，LNB 也与此类似；

xNB ——确定到任一段内的某个字；

SSN ——只确定到段号；

PC ——确定到一个半字；

B ——确定到一个字节地址。

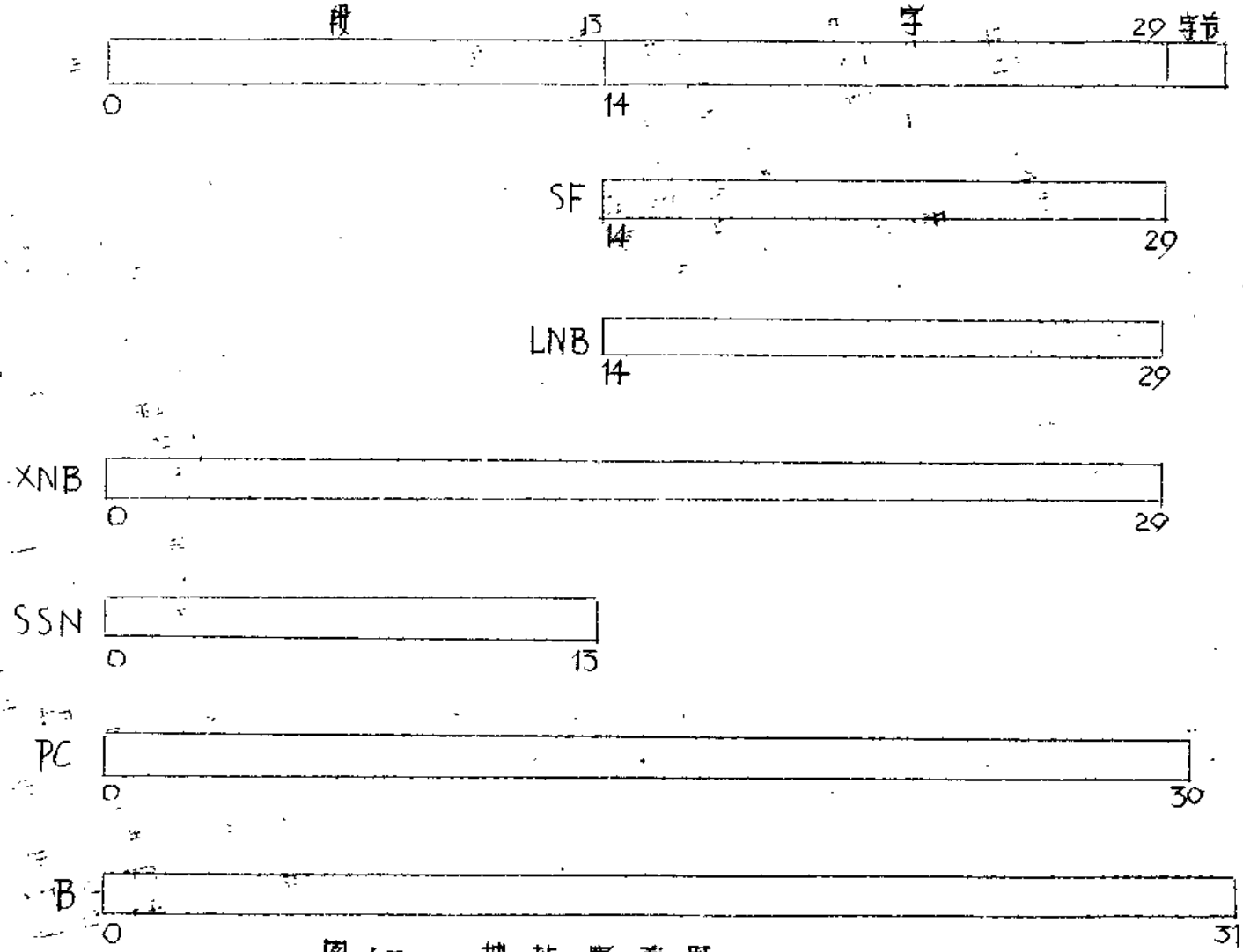



图 1.7　地 址 寄 存 器

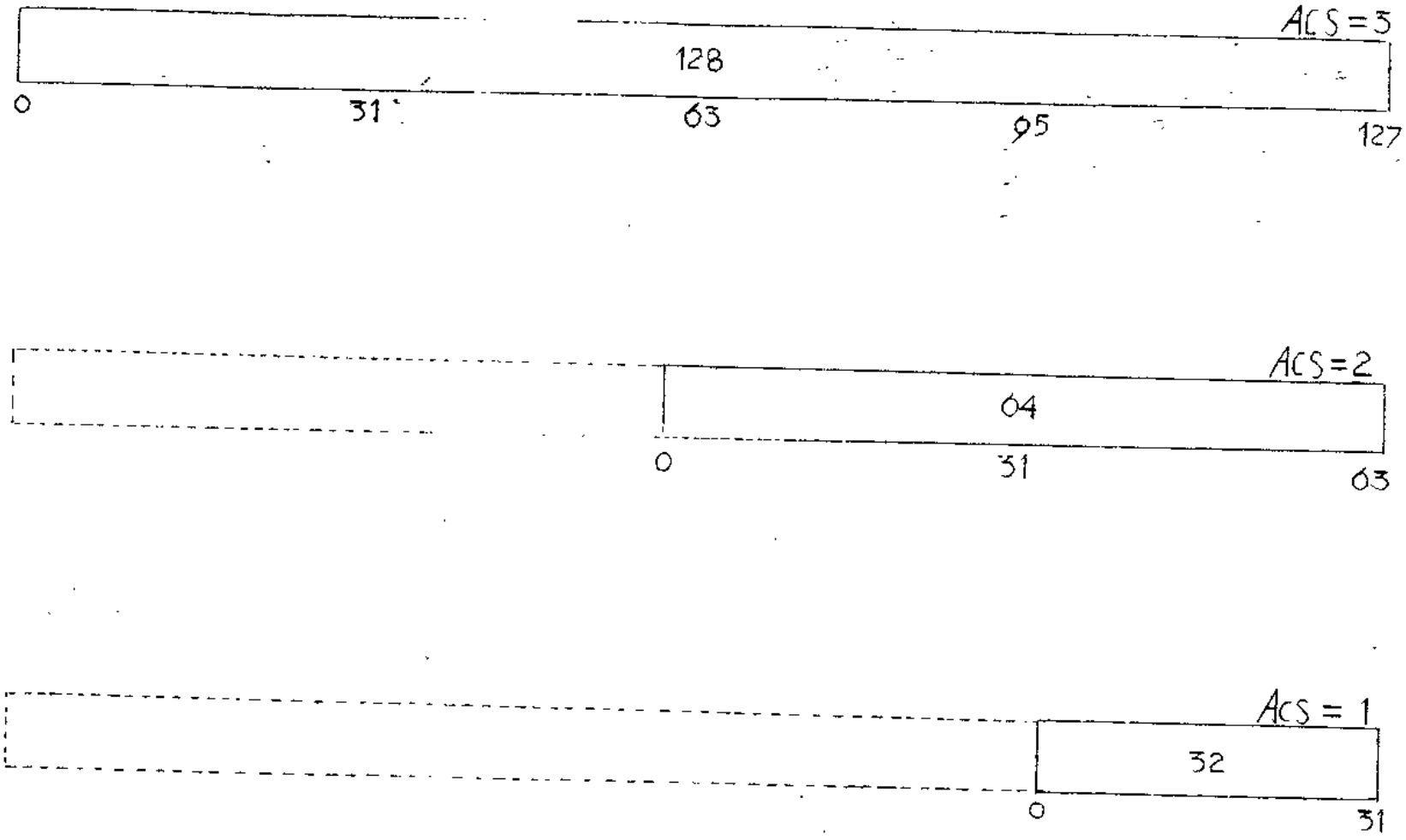



图 1.8　　累加器

SF，LNB，XNB反pc的最低几部被拼上若干了0，其结果便是给硬件提供了一了32位的地址。

1.8 新系列机有一了累加器，ACC，见图1.8。

累加器可以是1字或2字或4字长，其大小系根据一了2位寄存器，ACS的置位情况而定。

累加器的操作部介总是在寄存器的低位端。

累加器是实行算术与逻辑操作的寄存器；不过还有另一种用处，即用于当我们要把数据从内存的某一部介复写到另一部介去时，尤其在栈与内存的其余部介之间要进行复写时。

1.9 图1.9给出了所包含的一些指令代码的功能

靠右边那列字母为各该功能的记忆码。详见附录14

首先，就是一了简单明晰的Load（装入）指令，其记忆码为L，这动作就是简单把把地址里的某项数据装入累加器。被装入数据的长度则由ACS（累加器大小寄存器）的置位情况来决定，且装入指令不改变置位情况。

我们也可以在装入之前把大小重置为32，64或128，然后再装入累加器。对此则可使用下面三条指令LSS，LSD，或LSQ中的一条。再有就是使累加器当前内容进栈，然后再从某特定地址装入累加器的指令。首先，是简单的STACK & LOAD，使始终使用现行大小的置位情况。该组的其它指令则是使用现行置位大小把累加器内容进栈，然后把大小重置为32或64或128，再使用新的置位大小从特定地址装入累加器，

如果我们只要累加器进栈，不需要重新装入新的内容，就可以使用地址为TOS（栈顶）的存贮指令ST。

1.10 现在让我们看图1.10，看一了怎样进行工作的例子。假设我们要把三项数据——ALPHA，BETA和GAMMA，从内存传送到栈上。假定ALPHA为1字长，BETA为2字长，GAMMA为4字长。

| | | | | |
|---|---|---|---|---|
| | | Load | Acc | L |
| Set size = 32 / 64 / 128 | & | Load | Acc | LSS / LSD / LSQ |
| Stack, Set size = 32 / 64 / 128 | & | Load | Acc | SLSS / SLSD / SLSQ |
| Stack & | | Load | Acc | SL |
| Stack Acc | | | | ST/TOS |

図 1.9

```
ALPHA   -1   字
BETA    -2   字
GAMMA   -4   字

LSS/ALPHA
SLSD/BETA
SLSQ/GAMMA
ST/TOS
```

图 1.10

下面就是我们要使用的指令。

首先，把大小置为 32 再把 ALPHA 装入累加器，指令写成 LSS/ALPHA（我们将假定有一个编译程序，这个编译程序将把我们的源代码以 LSS/ALPHA 等被译成为与此种指令及操作数相适应的二进形式）。

然后我们必须改变累加器大小使它成为 64，并装入 BETA。为了做这个动作，就要用 SLSD。这条指令的功能是使 ALPHA 进栈，并把二个字的数据置 BETA 装入累加器，同样，我们再使 BETA 进栈并把大小置为 128，装入 GAMMA。这是使用指令 SLSQ.

最后，再用指令 ST，地址为 TOS，把 GAMMA 存到栈顶上。

现在再来看使用累加器和栈如何实行简单的算术运算。新系列的指令代码允许使用 4 种不同算术运算，或者更确切些说，它允许以四种不同的方式处理我们正对它进行加工的二进形式。——

第一，作为有符号的整数，这时，最高位就是符号位，负数用 2 的补码形式，

第二，作为逻辑字或无符号整数。

第三，作为浮点数

最后，作为压缩的十进制数。

1.11 表 1.11是把二进形式处理为有符号整数的算术操作。

注意，这些指令开头的字母都是 I，它表示整数（integer）。（全部指令代码功能及其它们的对列在附录 1 A 中）。还将发现所有的浮点指令均以 R 开头，它是实数（real numbers）的缩写；

# INTEGER ARITHMETIC （整数运算）

IAD /operand – add operand to Acc　把操作数加到 ACC

ISB / operand – subtract operand from Acc　从ACC减去操作数

IRSB/operand – subtract Acc from operand　从操作数减去 Acc（即反减）
(i.e. reverse subtract)

IMY /operand – multiply Acc by operand　用操作数乘 Acc

IDV /operand – divide ACC by operand　用操作数除 ACC

IRDV/operand – divide operand by ACC　用 ACC 除操作数（即反除）
(i.e. reverse divide)

图 1.11