

计算机组成与体系结构

数据表示、计算以及指令系统

计算机科学与技术

• 计算机系统的多层次结构

虚拟机器
(virtual machine)

应用语言机器(应用语言)

第6级

高级语言机器(高级语言)

第5级

汇编语言机器(汇编语言)

第4级

操作系统机器(作业控制语言)

第3级

实际机器
(actual machine)

传统机器(机器指令集)

第2级

微程序机器(微指令集)

第1级

数字逻辑

第0级

- 计算机体系结构

- 传统机器级的属性：功能特性

- **数据表示**：硬件能够直接识别和处理的数据类型和格式；
- **寻址方式**：最小寻址单位、寻址方式的种类和地址运算等；
- **指令集**：机器指令的操作类型、格式，指令间的排序和控制机制等；
- 存储系统；
- 输入输出系统；
-

- 数据的表示
- 数据的计算
- 指令系统

- 数据的表示与数据结构
- 传统机器数据表示：
 - 定点数：原码、反码、补码、移码
 - 浮点数：尾数、基数和阶码。IEEE754 标准。
 - 字符：常用ASCII码。
 - 汉字：输入码，汉字内码，汉字字模码。
- 高级数据表示：自定义的数据表示，向量数据表示、堆栈数据表示
- 数据校验原理：奇偶校验码

- **数据表示**是指计算机硬件能够直接识别和引用的数据类型。称它为**操作数的物理属性**或**操作数的表示**。如：定点、逻辑、浮点、十进制、字符、字符串等。
- **数据结构**是面向具体应用的，由软件系统处理的各种**数据结构**。除数据表示以外的数据类型，一般都是数据结构要研究的内容。称为**操作数的逻辑属性**或**操作数类型**。如树、图、阵列、队列、链表、栈、向量等。
- 确定哪些数据类型用数据表示实现，哪些数据类型用数据结构实现，是软件与硬件的取舍问题。

- 数据结构和数据表示都是数据类型的子集。
- 计算机系统结构研究的首要问题是：
 - 在所有的数据类型中，哪些用硬件实现，哪些用软件实现，并研究它们的实现方法。

目前，定点数、浮点数、逻辑数、十进制数、字符串等基本数据表示和变址操作是不可缺少的。但还需要为数据结构提供更进一步的支持，减少软件负担。

□均采用二进制表示和存储。

□定点数：原码、反码、补码、移码。定点数使用补码进行运算。

□浮点数：尾数、基数和阶码。IEEE754 标准。

□字符与字符串表示方法：常用ASCII码。

□汉字：输入码，汉字内码，汉字字模码。

数值数据的表示（简单回顾）



数制带后缀的表示：

十进制数（D）、二进制数（B）、八进制数（Q）、十六进制数（H）
在C语言中，八进制常数以前缀0开始，十六进制常数以前缀0x开始。

下标的表示方式： $(100)_2$

计算机中的数值数据

无符号数：就是整个机器字长的全部二进制位均表示数值位（没有符号位），相当于数的绝对值。

$N_1 = 01001$ 表示无符号数9 $N_2 = 11001$ 表示无符号数25

对于字长为 $n+1$ 位的无符号数的表示范围是 $0 \sim (2^{n+1}-1)$ 。

例如：字长为8位，无符号数的表示范围是 $0 \sim 255$ 。

带符号数的表示



最高位用来表示符号位，前例中的 N_1 、 N_2 在这里变为：

$N_1 = 01001$ (机器数) 表示带符号数+9 (真值)

$N_2 = 11001$ 不同的机器数表示不同的值，如：

原码时表示带符号数-9，补码则表示-7，反码则表示-6。

- $[X]_{\text{原}}$ 符号位加绝对值。当 X 为正数时， $[X]_{\text{补}} = [X]_{\text{原}} = [X]_{\text{反}}$ 。
- X 为负数时，由 $[X]_{\text{原}}$ 转换为 $[X]_{\text{补}}$ 的方法：
① $[X]_{\text{原}}$ 除掉符号位外的各位取反加“1”。② 自低位向高位，尾数的第一个“1”及其右部的“0”保持不变，左部的各位取反，符号位保持不变。

例如：

$$\begin{array}{l} [X]_{\text{原}} = 1.1110011000 \\ [X]_{\text{补}} = 1.\underline{000110}1000 \end{array}$$

 ↑ ↑ ↑
 不变 变反 不变

- X 为负数时，由 $[X]_{\text{原}}$ 转换为 $[X]_{\text{反}}$ 的方法：除掉符号位外的各位取反。

在原码表示中，真值0有两种不同的表示形式：

$$[+0]_{\text{原}} = 00000$$

$$[-0]_{\text{原}} = 10000$$

在补码表示中，真值0的表示形式是唯一的。

$$[+0]_{\text{补}} = [-0]_{\text{补}} = 00000$$

在反码表示中，真值0也有两种不同的表示形式：

$$[+0]_{\text{反}} = 00000$$

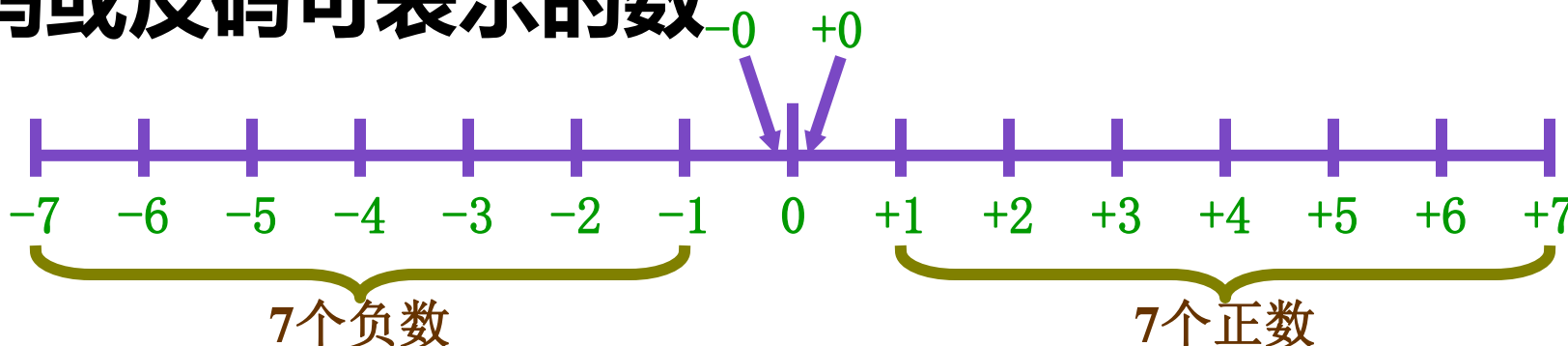
$$[-0]_{\text{反}} = 11111$$

三种机器数范围示例

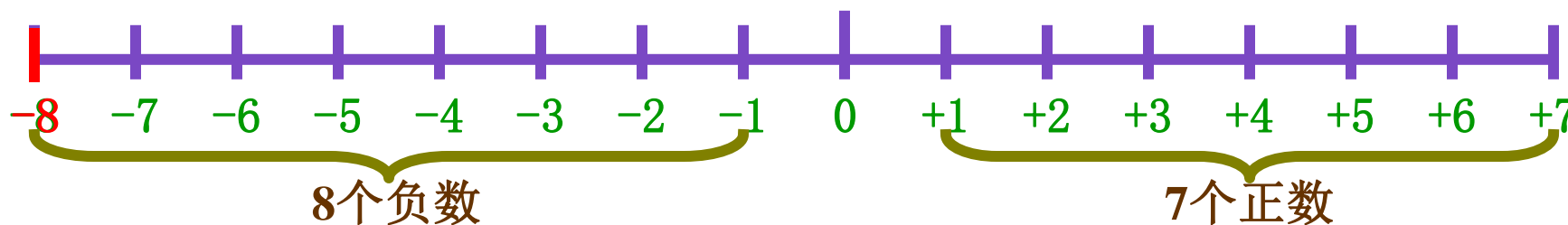


设机器字长**4**位（含1位符号位），以纯整数为例：

原码或反码可表示的数



补码可表示的数（多表示一个负数）



(1)对于正数它们都等于真值本身，而对于负数各有不同的表示。

(2)最高位都表示符号位，补码和反码的符号位可和数值位一起参加运算；但原码的符号位必须分开进行处理。

(3)对于真值0，原码和反码各有两种不同的表示形式，而补码只有唯一的一种表示形式。

(4)原码、反码表示的正、负数范围是对称的；但补码负数能多表示一个最负的数（绝对值最大的负数），其值等于 -2^n （纯整数）或 -1 （纯小数）。



(5) 现代计算机常用补码表示带符号数。

补码运算系统是**模**运算系统，符号位参加运算，加、减运算统一（**模和同余**）；数0的表示唯一，方便使用；比原码和反码多表示一个最小负数。

(6) 已知机器的字长，则机器数的位数应补够相应的位，填补原则是不改变数值大小。

举例：设机器字长为8位，则：

$$X1=1011$$

$$[X1]_{\text{原}}=0,0001011$$

$$[X1]_{\text{补}}=0,0001011$$

$$[X1]_{\text{反}}=0,0001011$$

$$X1=0.1011$$

$$[X1]_{\text{原}}=0.1011000$$

$$[X1]_{\text{补}}=0.1011000$$

$$[X1]_{\text{反}}=0.1011000$$

$$X2=-1011$$

$$[X2]_{\text{原}}=1,0001011$$

$$[X2]_{\text{补}}=1,1110101$$

$$[X2]_{\text{反}}=1,1110100$$

$$X2=-0.1011$$

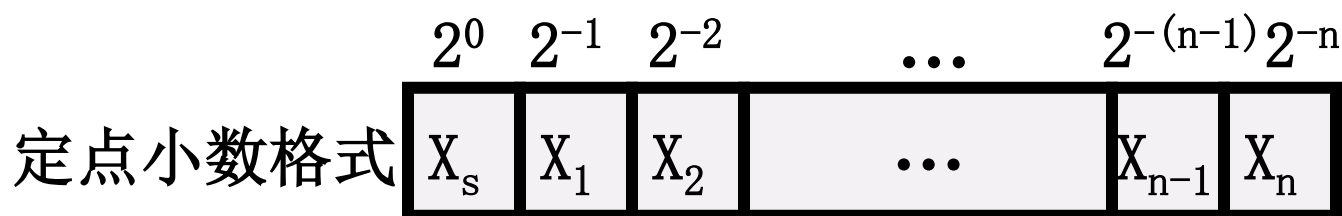
$$[X2]_{\text{原}}=1.1011000$$

$$[X2]_{\text{补}}=1.0101000$$

$$[X2]_{\text{反}}=1.0100111$$

定点小数

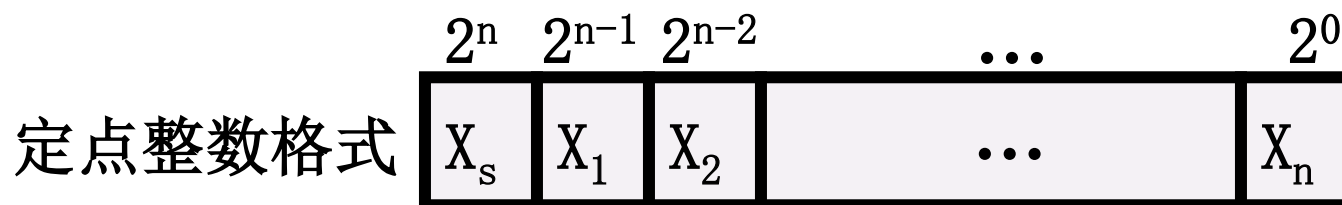
小数点的位置固定在最高有效数位之前，符号位之后，记作 $X_s.X_1X_2\dots X_n$ ，这个数是一个纯小数。定点小数的小数点位置是隐含约定的，小数点并不需要真正地占据一个二进制位。



↑
小数点位置

定点整数

小数点位置隐含固定在最低有效数位之后，记作 $X_sX_1X_2\dots X_n$ ，这个数是一个纯整数。



↑
小数点位置

若机器字长有 $n+1$ 位，则：

原码定点小数表示范围为： $-(1-2^{-n}) \sim (1-2^{-n})$

补码定点小数表示范围为： $-1 \sim (1-2^{-n})$

原码定点整数的表示范围为： $-(2^n-1) \sim (2^n-1)$

补码定点整数的表示范围为： $-2^n \sim (2^n-1)$

若机器字长有8位，则：

原码定点小数表示范围为： $-(1-2^{-7}) \sim (1-2^{-7})$

补码定点小数表示范围为： $-1 \sim (1-2^{-7})$

原码定点整数表示范围为： $-127 \sim 127$

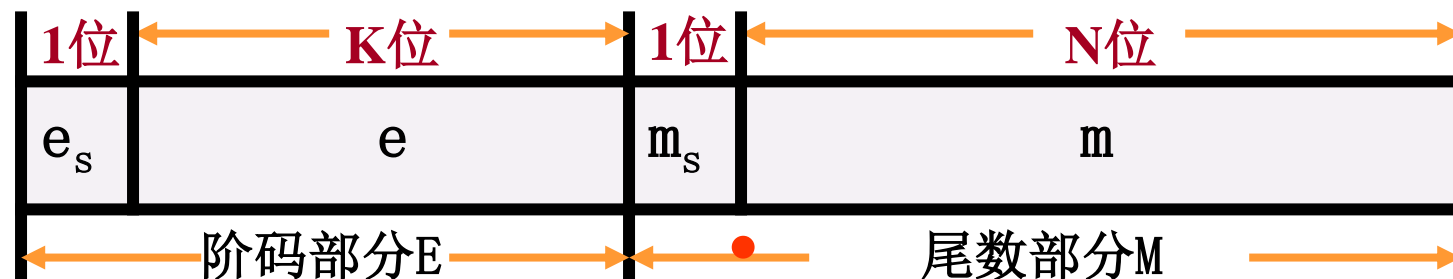
补码定点整数表示范围为： $-128 \sim 127$

小数点的位置根据需要而浮动，这就是浮点数。例如：

$$N = M \times r^E = M \times 2^E$$

式中： r 为浮点数阶码的底，与尾数的基数相同，通常 $r=2$ 。 E 和 M 都是带符号数， E 叫做阶码， M 叫做尾数。在大多数计算机中，尾数为纯小数，常用原码或补码表示；阶码为纯整数，常用移码或补码表示。

浮点数的一般格式：





浮点数的规格化表示

为了提高运算的精度，需要充分地利用尾数的有效数位，通常采取规格化的浮点数形式，即规定**尾数的最高数位必须是一个有效值**。

$$1/r \leq |M| < 1$$

如果 $r=2$ ，则有 $1/2 \leq |M| < 1$ 。

重要的结论：

在尾数用原码表示时，规格化浮点数的尾数的**最高数位总等于1**，形如 $x_s.1xx...x$ 。在尾数用补码表示时，规格化浮点数应满足**尾数最高数位与符号位不同** ($m_s \oplus m_1 = 1$)，即当 $1/2 \leq M < 1$ 时，应有 $0.1xx...x$ 形式，当 $-1 \leq M < -1/2$ 时，应有 $1.0xx...x$ 形式。

浮点数的表示范围



假定阶码和尾数均用补码表示，阶码为 $k+1$ 位，尾数为 $n+1$ 位，则浮点数的典型值如下。

	浮点数代码		真值
	阶码	尾数	
最大正数	01...1	0.11...11	$(1-2^{-n}) \times 2^{2^k-1}$
绝对值最大负数	01...1	1.00...00	$-1 \times 2^{2^k-1}$
最小正数	10...0	0.00...01	$2^{-n} \times 2^{-2^k}$
规格化的最小正数	10...0	0.10...00	$2^{-1} \times 2^{-2^k}$
绝对值最小负数	10...0	1.11...11	$-2^{-n} \times 2^{-2^k}$
规格化的绝对值最小负数	10...0	1.01...11	$(-2^{-1}-2^{-n}) \times 2^{-2^k}$

一般尾数为0，即当机器零处理。机器零的标准格式为尾数为0，阶码为绝对值最大的负数。请自行推导。

浮点数阶码的移码表示法



移码就是在真值 X 上加一个常数（偏置值），相当于 X 在数轴上向正方向平移了一段距离，这就是“移码”一词的来由，移码也可称为增码或偏码。

$$[X]_{\text{移}} = \text{偏置值} + X$$

字长 $n+1$ 位定点整数的移码形式为 $\mathbf{X_0}X_1X_2\dots X_n$ 。偏置值可以取 2^n ，或者 2^n-1 。

假定当字长8位时，偏置值为 $\mathbf{2^7}$ 。

例1: $X=1011101$

$$\begin{aligned}[X]_{\text{移}} &= \mathbf{2^7} + X \\ &= 10000000 + 1011101 \\ &= \mathbf{1}1011101 \\ [X]_{\text{补}} &= \mathbf{0}1011101\end{aligned}$$

例2: $X=-1011101$

$$\begin{aligned}[X]_{\text{移}} &= \mathbf{2^7} + X \\ &= 10000000 - 1011101 \\ &= \mathbf{0}0100011 \\ [X]_{\text{补}} &= \mathbf{1}0100011\end{aligned}$$

偏置值为 2^n 的移码具有以下特点



- (1)在移码中，最高位为“0”表示负数，最高位为“1”表示正数。
- (2)移码为全0时，它所对应的真值最小，为全1时，它所对应的真值最大。
- (3)真值0在移码中的表示形式是唯一的，即 $[+0]_{\text{移}} = [-0]_{\text{移}} = 100\dots0$ 。
- (4)移码把真值映射到一个正数域，所以可将移码视为无符号数，直接按无符号数规则比较大小。
- (5)同一数值的移码和补码除最高位相反外，其他各位相同。

浮点数的阶码常采用移码表示最主要的原因有：

- 便于比较浮点数的大小。阶码大的，其对应的真值就大，阶码小的，对应的真值就小。
- 简化机器中的判零电路。当阶码全为0，尾数也全为0时，表示机器零。

IEEE754标准浮点数 (复习用)



大多数计算机的浮点数采用IEEE 754标准，其格式如下，IEEE754标准中有三种形式的浮点数。



类型	数符 m_s	阶码 E	尾数 m	总位数	偏置值	
短浮点数	1	8	23	32	7FH	127
长浮点数	1	11	52	64	3FFH	1023
临时浮点数	1	15	64	80	3FFFH	16383

以短浮点数为例讨论浮点代码与其真值之间的关系。最高位为数符位；其后是8位阶码，以2为底，阶码的偏置值为**127**；其余23位是尾数。为了使尾数部分能表示更多一位的有效值，IEEE754采用**隐含尾数最高数位1**（即这一位1不表示出来）的方法，因此尾数实际上是**24**位。应注意的是，**隐含的1是一位整数（即位权为 2^0 ）**，在浮点格式中表示出来的23位尾数是纯小数，并用原码表示。

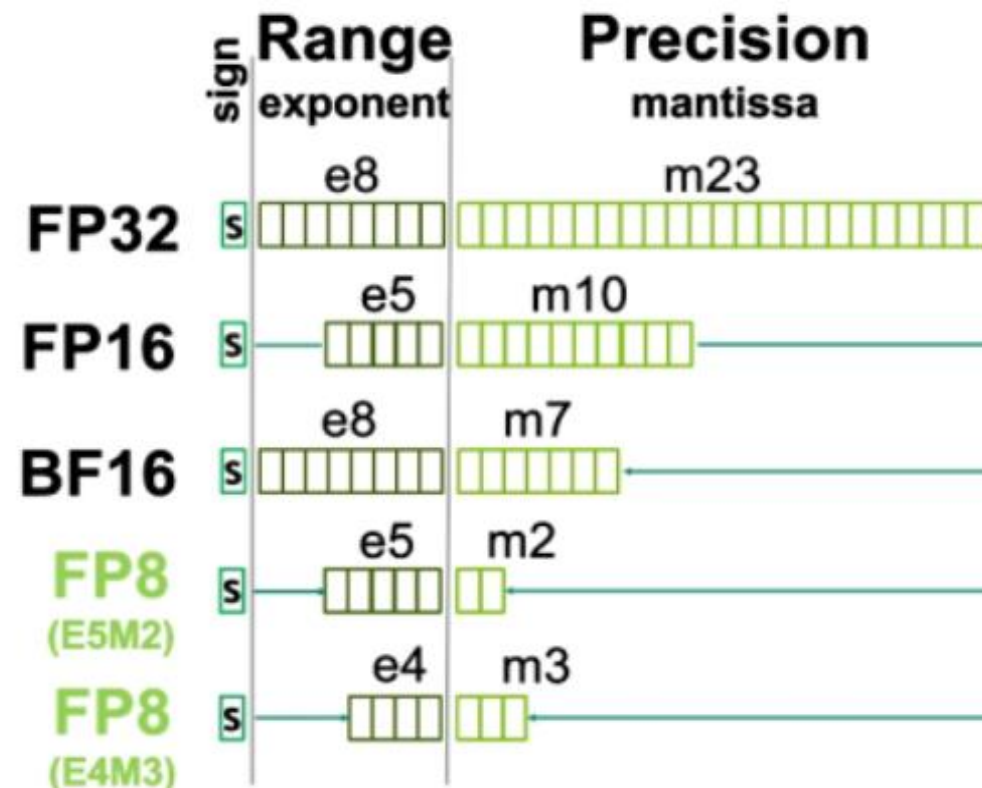
※补充说明:



- (1) 为了确保浮点数表示的唯一性, 约定 $0 \leq M < 1$;
- (2) E 为全0且 M 非全0: 非规范浮点数 (E 偏移126),
则: $F_{\text{真}} = (-1)^S \times M \times 2^{E-126}$
- (3) E 为全0且 M 为全0: 表示浮点数 0
- (4) $1 \leq E \leq 254$: 数是规范浮点数 (E 偏移127),
则: $F_{\text{真}} = (-1)^S \times (1 + M) \times 2^{E-127}$
- (5) E 为全1 (255) : M 为全0, 则 $F_{\text{真}} = \pm \infty$
- (6) E 为全1 (255) : M 非全0时, 代码无效(NaN);

• 更少的位数->能耗/存储效率

FP8首次出现在2022年4月,
Nvidia 发布的最新一代高性能
GPU架构: H100。H100
TensorCore中引入了一种新的浮
点类型FP8。





十进制与短浮点格式的相互转换

例：将 $(100.25)_{10}$ 转换成短浮点数格式。

(1)十进制数 \rightarrow 二进制数 $(100.25)_{10} = (1100100.01)_2$

(2)非规格化数 \rightarrow 规格化数 $1100100.01 = 1.10010001 \times 2^6$

(3)计算移码表示的阶码 (偏置值 + 阶码真值)

$$1111111 + 110 = 10000101$$

(4)以短浮点数格式存储该数。

符号位=0

阶码=10000101

尾数=100100010000000000000000

短浮点数代码为

0;100 0010 1;100 1000 1000 0000 0000 0000

表示为十六进制的代码：42C88000H。

思考：已知短浮点数如何转十进制数？

字符ASCII码



1.ASCII字符编码

常见的ASCII码用七位二进制表示一个字符，它包括10个十进制数字、52个英文大写和小写字母、34个专用符号和32个控制符号，共计128个字符。

数字和英文字母按顺序排列，只要知道其中一个的二进制代码，不要查表就可以推导出其他数字或字母的二进制代码。

字符串存放方式（自学）

(1)向量法 (2)串表法

$b_6b_5b_4$ $b_3b_2b_1b_0$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	RO	RS	.	>	N	↑	n	~
1111	SI	US	/	?	O	_	o	DEL

1) 汉字输入码

包括：数字码、拼音码、字形码

数字码：常用汉字区位码

区位码将汉字编码GB2312-80中的6763个汉字分为94个区，每个区中包含94个汉字（位），区和位组成一个二维数组，每个汉字在数组中对应一个唯一的区位码。汉字的区位码定长4位，前2位表示区号，后2位表示位号，区号和位号用十进制数表示，区号从01到94，位号也从01到94。

国标码 = 区位码（十六进制） + 2020H

拼音码：以汉字拼音为基础的输入方法（如微软拼音）

字形码：根据汉字的书写形状来进行编码（如五笔字型）。

2) 汉字内码

汉字可以通过不同的输入码输入，但在计算机内部其内码是唯一的。汉字内码以国标码为基础。GB2312-80简称国标码。该标准共收集常用汉字6763个，其中一级汉字3755个，按拼音排序；二级汉字3008个，按部首排序；另外还有各种图形符号682个，共计7445个。每个汉字、图形符号都用两个字节表示，每个字节只使用低七位编码。

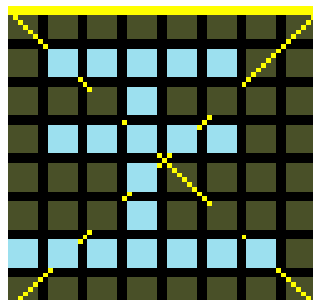
因为汉字处理系统要保证中西文的兼容，当系统中同时存在ASCII码和汉字国标码时，将会产生二义性。

汉字机内码 = 汉字国标码 + 8080H = 区位码转成16进制 + A0A0H

3) 汉字字模码

汉字字形码是指确定一个汉字字形点阵的代码，又叫汉字字模码或汉字输出码。汉字形状的描述信息集合形成字库。

字模点阵描述（图像方式）



00H 7CH;
10H 7CH;
10H 10H;
FEH 00H;

在一个汉字点阵中，凡笔画所到之处，记为“1”，否则记为“0”。根据对汉字质量的不同要求，可有 16×16 、 24×24 、 32×32 或 48×48 的点阵结构。显然点阵越大，输出汉字的质量越高，每个汉字所占用的字节数也越多。

汉字的输入码、内码、字模码分别是用于计算机输入、内部处理、输出三种不同用途的编码，不要混淆。

请自行查阅资料：

- 1) 汉字编码的发展
- 2) 统一代码 (Unicode)
- 3) BCD码 (8421, 余三码, 格雷码) 原理以及十进制数串的表示

• 两类

□ **原子类型**：例如实型、整型、布尔型、字符型等。

□ **结构类型**：例如数组、队列、链表、栈、向量等。

常用十进制表示数值，Typedef 或 Class 定义或描述结构型数据类型。

```
# define LIST_INIT_SIZE 100;
# define LIST_INC      10;
Typedef struct {
    ElemType          *elem;
    int                length;
    int                listsize;
}
```

```
Class Student {
    private String IDNo;
    private String name;
    private String sex;
    private Date birthday;
}
```


□抽象数据类型：数学模型及其一组操作，与计算机内部如何表示和实现无关。

□高级数据表示，填补高级语言与机器语言的语义差异

➤例如：实现二维数组加： $A = A + B$ ，其中A和B均为 200×200 的二维数组。试分析CPU与主存之间的通信量。

```
int a[200][200], b[200][200];
for(i=0; i<200; i++) {
    for(j=0; j<200; j++) {
        a[i][j]=a[i][j]+b[i][j];
    }
}
```

假设数组a和b都在主存中，并且没有使用缓存或寄存器优化，没有向量数据表示。

基于理论的估计，假设了一些简化的条件。实际情况可能会因为编译器优化、CPU缓存和其他硬件特性而有所不同。

X86指令

```
mov ecx, 200
outer_loop:
    mov ebx, 200
    inner_loop:
        mov eax, offset a[i][j]
        add [eax], b[i][j]
        dec ebx
        jnz inner_loop
    dec ecx
    jnz outer_loop
```

主体部分
取指令 $2 \times 40,000$ 条，
读或写数据 $3 \times 40,000$ 个，
共要访问主存储器 $5 \times 40,000$ 次以上。

IBM370汇编指令（寄存器操作）

LR 1,=200 // 将200加载到寄存器1（外层循环计数器）

LR 2,=200 // 将200加载到寄存器2（内层循环计数器）

BEGIN_OUTER_LOOP:// 外层循环

BEGIN_INNER_LOOP:// 内层循环

// 计算数组索引

LA R3,A(R4,R5) // 假设R4和R5分别指向i和j，R3指向a[i][j]

LA R4,B(R4,R5) // R4指向b[i][j]

AR R3,R3,R4 // R3 = R3 + R4加法操作 (a[i][j] += b[i][j])

ST R3,A(R4,R5) // 存储结果回a[i][j]

LR 2,2-1// 内层循环计数器减1

BR 2,BEGIN_INNER_LOOP // 如果R2 != 0，跳转回内层循环开始

LR 1,1-1// 外层循环计数器减1

BR 1,BEGIN_OUTER_LOOP // 如果R1 != 0，跳转回外层循环开始

主体部分：

取指令 $2 + 4 \times 40,000$ 条，
读或写数据 $3 \times 40,000$ 个，
共要访问主存储器 $7 \times 40,000$ 次以上

这些指令并不是程序的主体部分，而是用于控制循环的执行流程，因此没有计入总的取指次数和访存次数中。

➤如果有向量数据表示，只需要一条指令：

向量加	A向量参数	B向量参数	C向量参数
-----	-------	-------	-------

□减少访问主存（取指令）次数： $4 \times 40,000$

□缩短程序执行时间一倍以上

➤可见，引入一些高级数据表示，对计算机性能的提升具有一定的意义。

- 有以下几种高级数据表示：

- 自定义数据表示
- 向量数据表示
- 堆栈数据表示

- 自定义数据表示

- 由**数据本身**来表明数据类型，使计算机内的数据具有自定义能力。
- **目的：** **缩短**机器语言与高级语言对数据属性的说明之间的**语义差距**。
- 分为两类：带标志符的数据表示、数据描述符

1. 带标志符的数据表示

- 从处理**运算符**和数据类型的关系上看，高级语言和机器语言的**差距很大**。
- **高级语言用类型说明语句指明数据的类型**，让数据类型直接与数据本身联系在一起，运算符不反映数据类型，是通用的。

□ 例如：在FORTRAN中，浮点数相加

REAL I, J

I=I+J (‘+’ 通用)

□ 在说明I、J的数据为实型后，用通用的 “+” 运算符就可实现实数加法。

• 1. 带标志符的数据表示

➤ 传统的机器语言程序却正好相反，它用**操作码**指明操作数的类型

□ 同一种操作指令通常需要很多条，IBM370系列机，仅加法指令就有8条。

例如：

✓ 浮点加法指令

✓ 定点加法指令

✓

浮加	I	J
定加	I	J

➤ **编译程序**：将高级语言中的数据类型说明语句和运算符变换成机器语言中的不同类型指令的操作码，并验证操作数与运算符是否一致。

➤ **编译程序负担很重。**

- 为缩短这种语义差距，在机器中设置带标志符的数据表示，
让每个数据都带有**类型标志**：

类型标志：指明数据值部分的类型，如为二进制整数，还是十进制整数，浮点数，字符串或地址字。

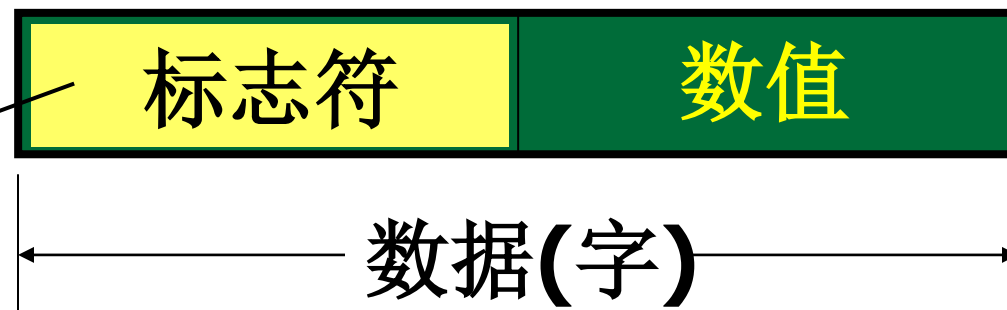


图 带标志符的数据表示

• 1. 带标志符的数据表示

目的：将数据类型与数据本身直接联系在一起，使机器语言中的**操作码**与高级语言中的**运算符**一致。
标志符由编译器或其它系统软件建立，对程序员**透明**。

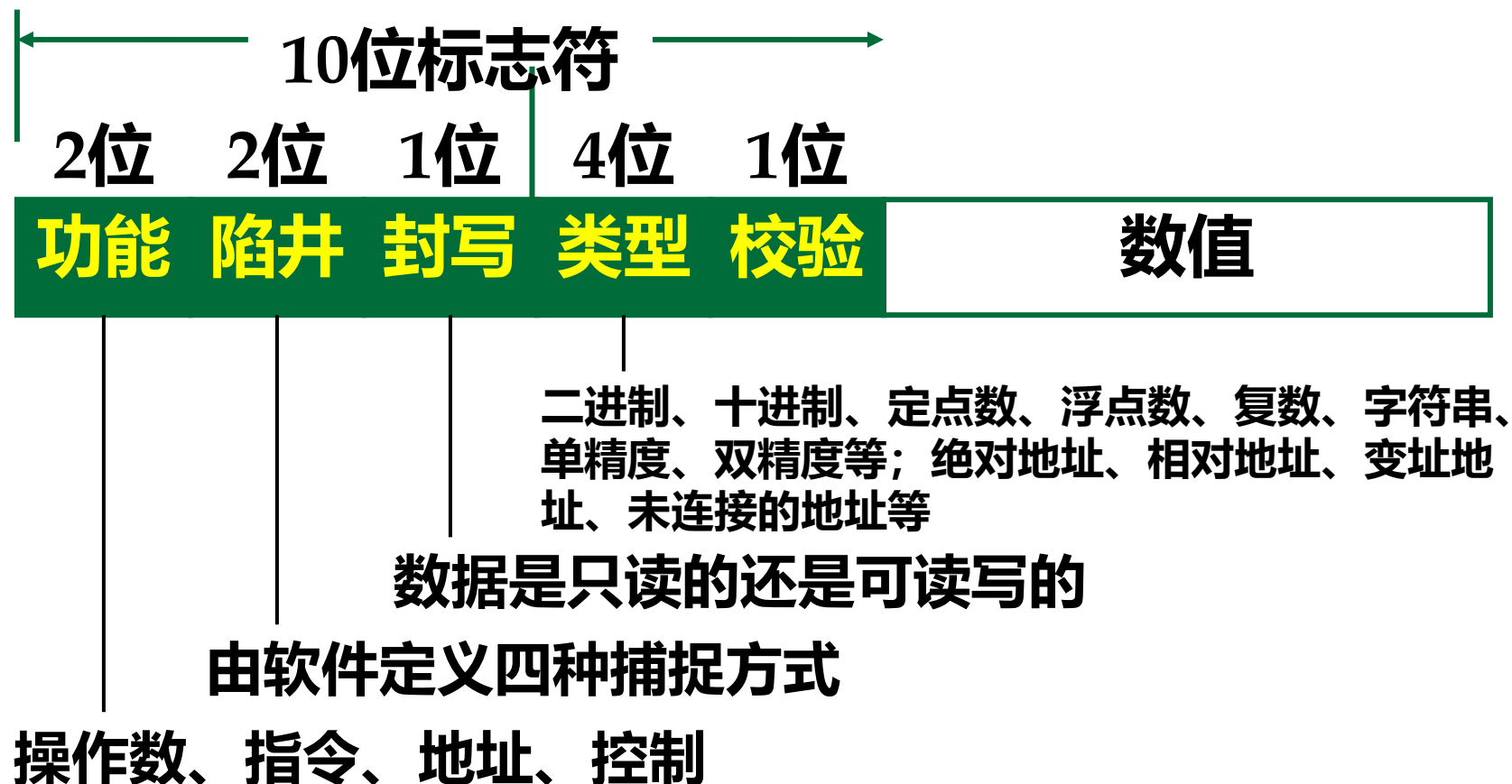


图 在R2巨型机中带标志符的数据表示方法

• 1. 带标志符的数据表示

➤ 优点:

- 提高了操作码的通用性，简化了指令系统和程序设计；
- 便于实现一致性检查，可由硬件直接快速地检测出多种程序错误；
- 能够由硬件自动完成数据类型转换；
- 支持了数据库系统的实现与数据类型无关的要求；
- 为软件调试和开发提供了支持。



1. 带标志符的数据表示

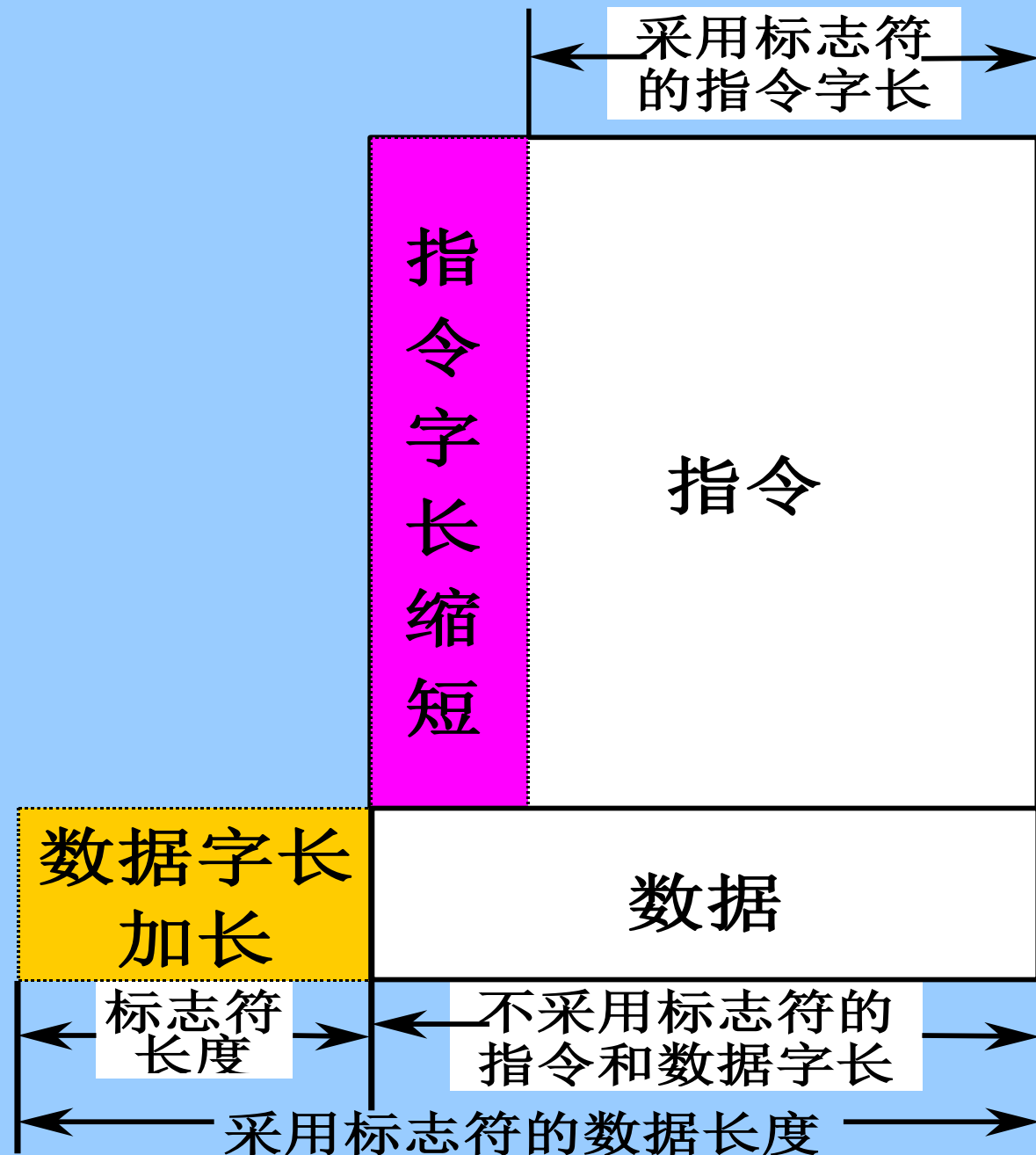
➤ 缺点:

- 每个数据字因增设标志符，可能会使程序所占用的主存空间增加（**不一定，见下一页例子**）。
- 采用标志符会降低指令的执行速度。程序的设计时间、编译时间和调试时间缩短。
- 数据和指令的长度可能不一致。
- 硬件复杂度增加。

➤ 优点明显，得到了广泛的应用。

自定义数据表示

- 图2-2两种情况下程序所占存储空间的比较
- 当  面积小于  面积时，程序所占空间减少



• 2. 数据描述符

➤ **目的：**支持向量、数组、记录等数据，减少标志符所占用的空间。

□因为这些数据的数据元素的属性都相同。

➤ **描述数据块的属性，与数据分开存放。**

➤ **与带标志符的数据表示的区别：**

□ **标志符**要与每个数据相连，两者合存在一个存储器单元中；而**描述符**则和数据分开放；

□要访问数据集中的元素时，必须先访问描述符，这就至少要增加一级寻址；

□ **描述符**可看成是程序一部分，而不是数据一部分，因为它是专门来描述要访问的数据的特性。

• 2. 数据描述符

- Burroughs公司生产的B-6700机中采用的数据描述符表示方法。

- 101时表示描述符

- 000时表示数据。

描述符



数据



高级数据表示-自定义数据表示

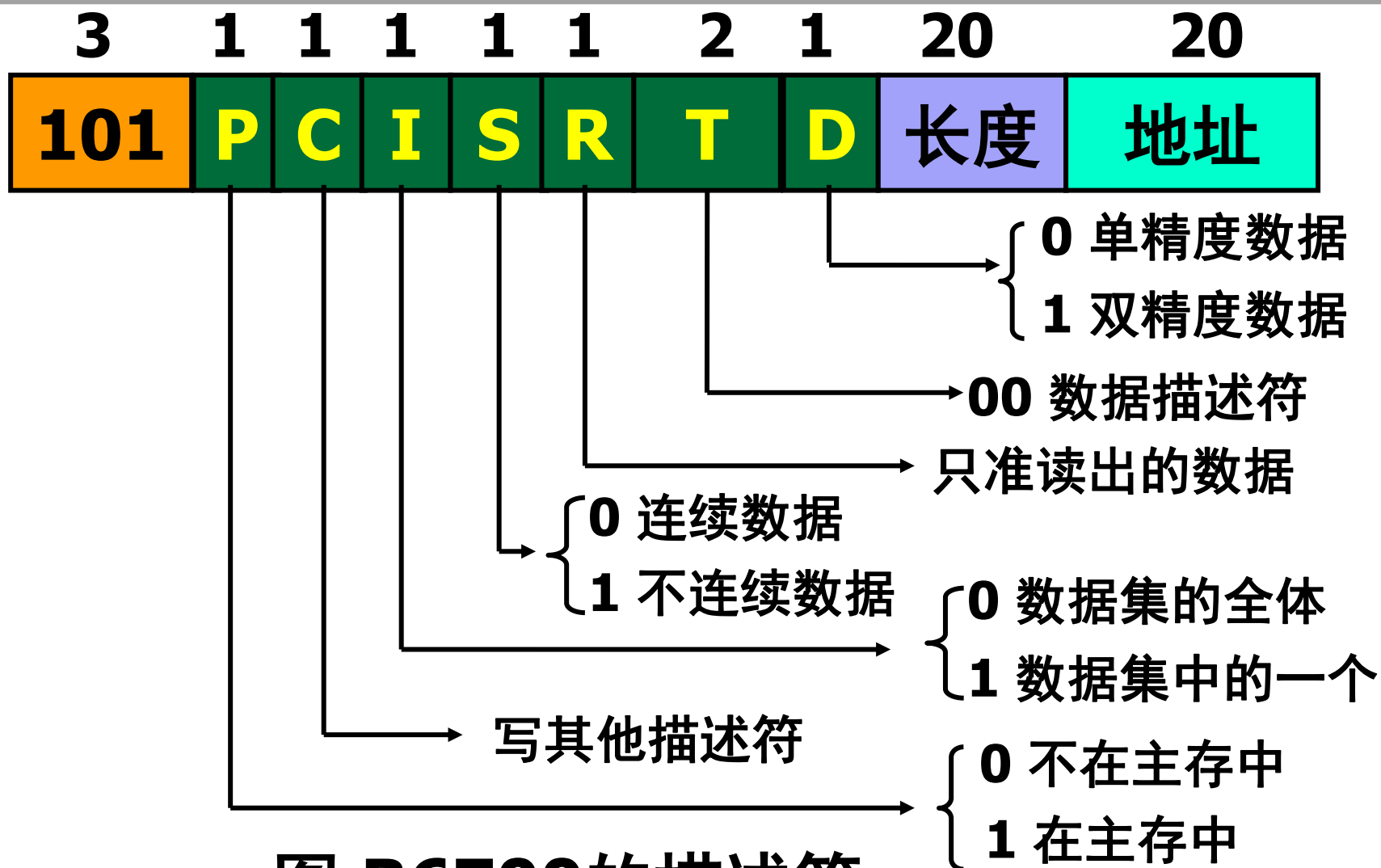
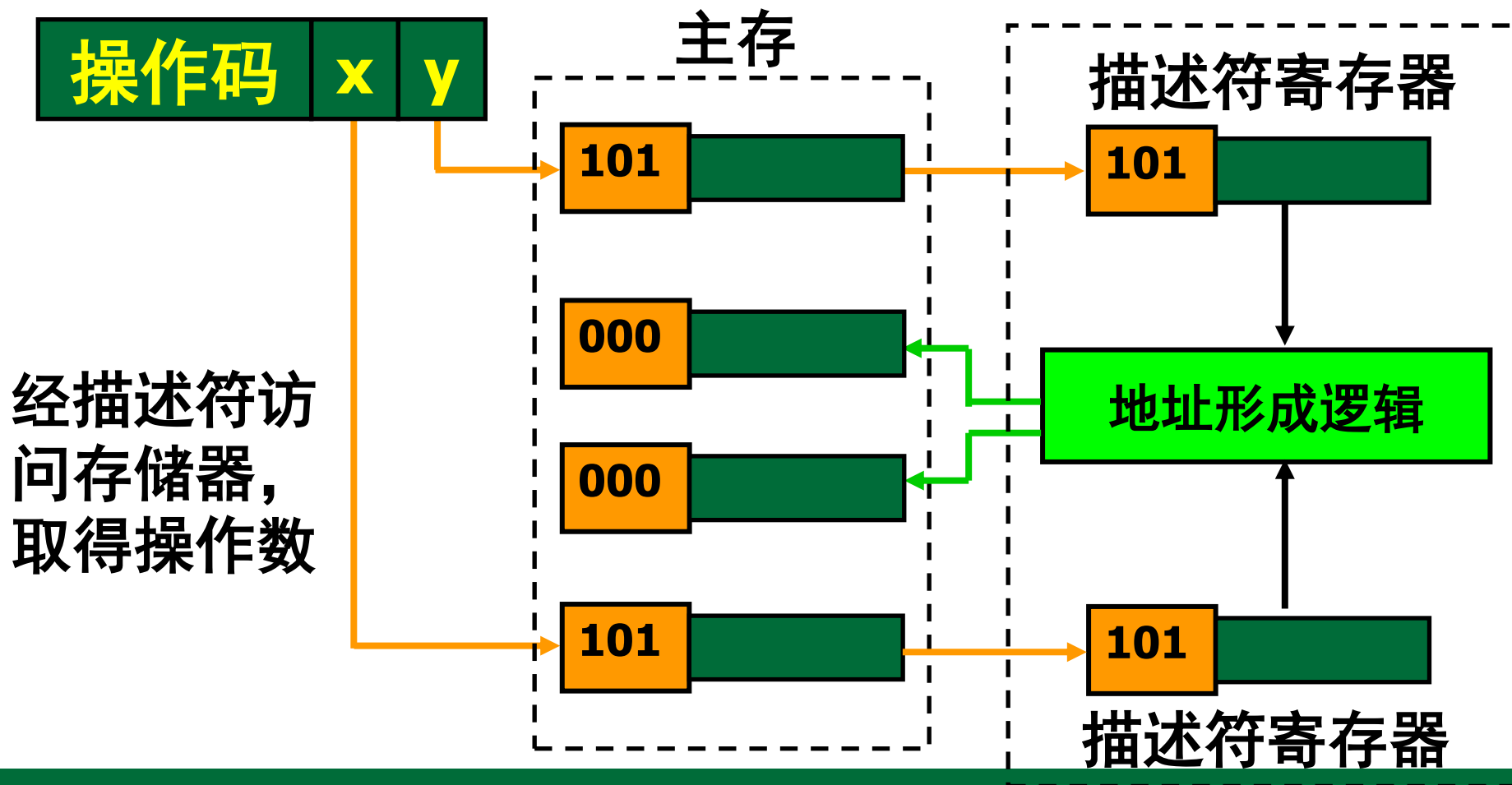


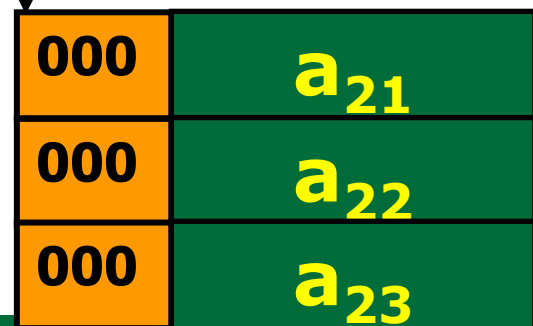
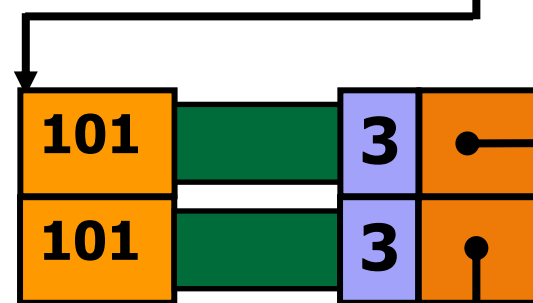
图 B6700的描述符

• 2. 数据描述符



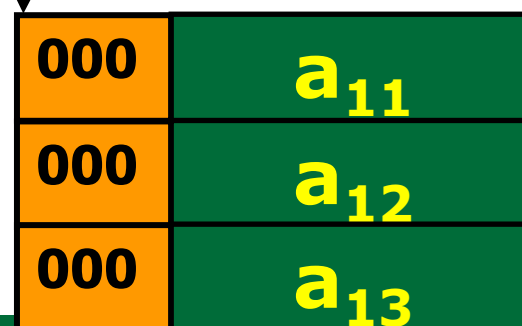
• 2. 数据描述符

将描述符
按树形连
接可以描
述多维阵
列（数组）



2×3二维阵列

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$



• 2. 数据描述符

➤ 优点:

- 将描述符按树形连接，可以描述复杂和多维数据结构；
- 为向量、数组等数据结构的实现提供了一定的支持，有利于简化编译，可以比变址法更快地形成元素地址。

• 向量数据表示

➤例如, 要计算 $c_i = a_{i+5} + b_i$ $i=10, 11, \dots, 1000$

用FORTRAN语言写成的有关DO循环部分为

```
DO 40 I = 10, 1000
```

```
40 C(I) = A(I+5) + B(I)
```

在具有向量、数组数据表示的向量处理机上, 表现出在硬件上设置有丰富的向量或阵列运算指令, 配置有以流水或阵列方式处理的高速运算器, 只需用一条如下的**向量加法**指令:

向量加	A向量参数	B向量参数	C向量参数
-----	-------	-------	-------

➤ **优点：**提高了向量、数组的运算速度。

- 快速形成元素地址；

- 实现数据块的预取；

- 用一条向量、数组指令，借助流水和并行运算等处理方式，可以同时实现对整个向量、数组的高速处理；

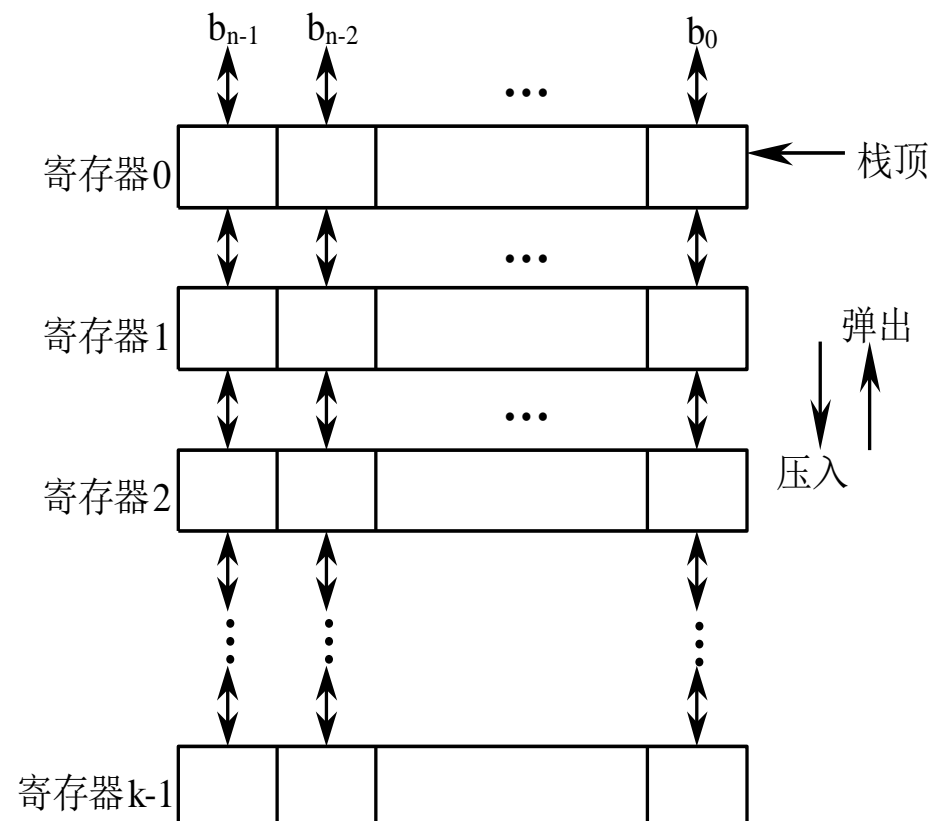
- 硬件处理稀疏向量和交叉阵列，节省存储空间，节省处理时间。

堆栈是一种按特定顺序进行存取的存储区，这种特定顺序可归结为“**后进先出**”（LIFO）或“**先进后出**”（FILO）。

堆栈主要用来暂存中断断点、子程序调用时的返回地址、状态标志及现场信息等，也可用于子程序调用时参数的传递，所以用于访问堆栈的指令只有进栈（压入）和出栈（弹出）两种。

1. 寄存器堆栈

用一组专门的寄存器构成寄存器堆栈，又称为硬堆栈。这种堆栈的栈顶是固定的，寄存器组中各寄存器是相互连接的，它们之间具有对应位自动推移的功能，即可将一个寄存器的内容推移到相邻的另一个寄存器中去。



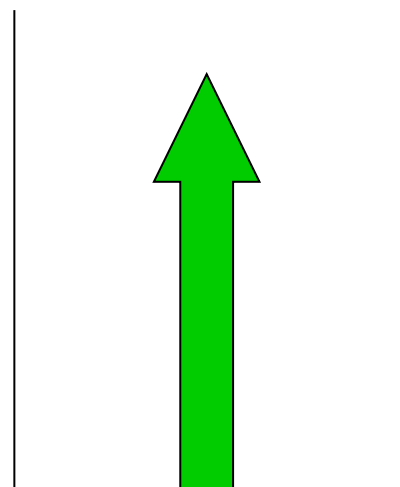
2.存储器堆栈



从主存中划出一段区域来作堆栈，这种堆栈又称为软堆栈，堆栈的大小可变，栈底固定，栈顶浮动，故需要一个专门的硬件寄存器作为堆栈栈顶指针**SP**，简称栈指针。栈指针所指定的主存单元，就是堆栈的栈顶。

自底向上生成
方式的堆栈

堆
栈
区



低地址

高地址

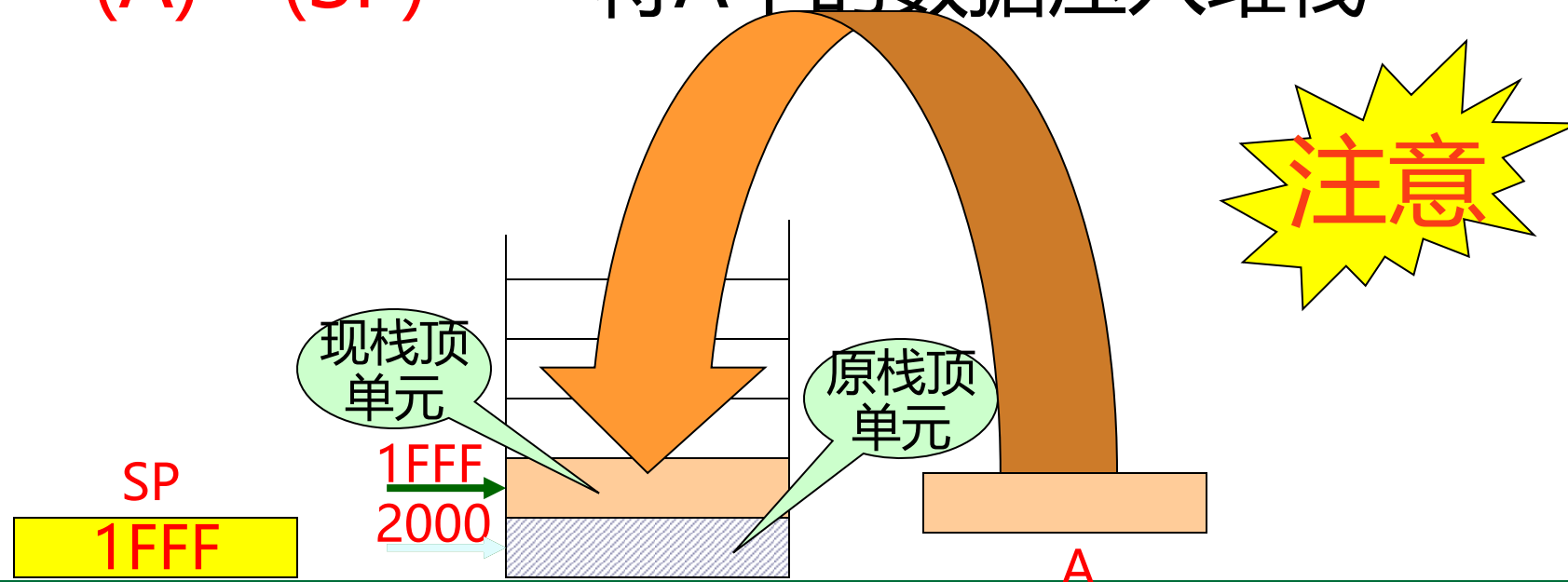
堆栈的栈底地址大于栈顶地址，通常栈指针始终指向**栈顶的满单元**。进栈时，SP的内容需要先自动减1，然后再将数据压入堆栈。

$(SP)-1 \rightarrow SP$

$(A) \rightarrow (SP)$

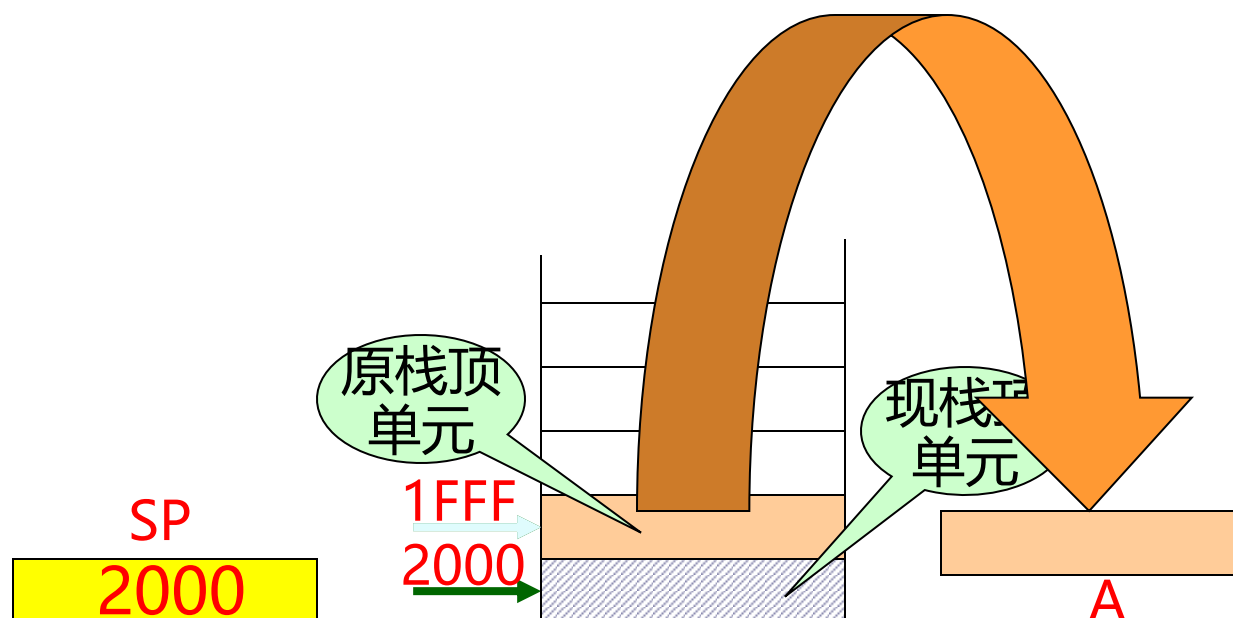
修改栈指针

将A中的数据压入堆栈

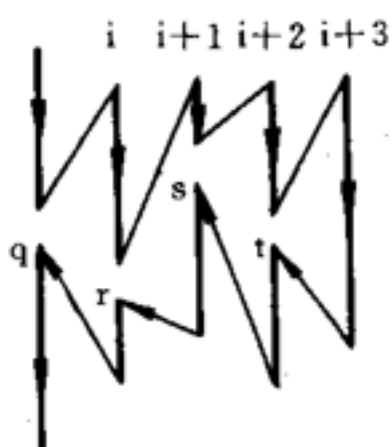


出栈时，需要先将堆栈中的数据弹出，然后SP的内容再自动加1。

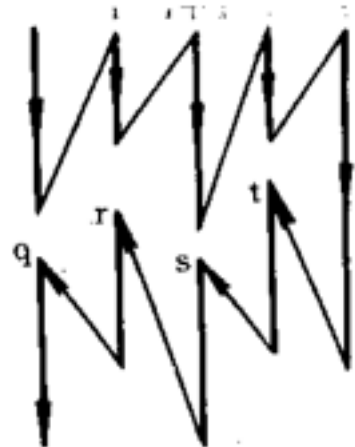
$((SP)) \rightarrow A$ 将栈顶内容弹出，送入A中
 $(SP) + 1 \rightarrow SP$ 修改栈指针



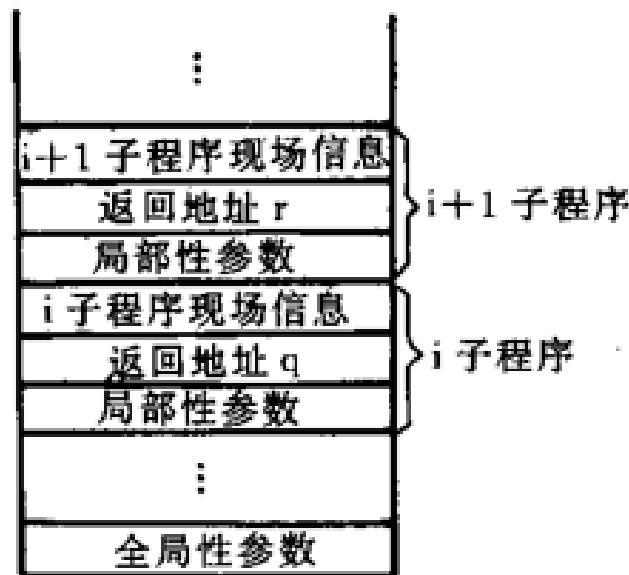
- 堆栈数据结构在编译和子程序调用中经常用到。
- 为了能高效实现堆栈数据结构，不少大型机乃至微型机都设有堆栈数据表示。
- 具有堆栈数据表示的机器称为**堆栈机器**。



嵌套调用



间接递归
直接递归



用堆栈实现子
程序的嵌套和
递归调用

➤堆栈机器则不同，表现为：

- 有由若干高速寄存器组成的**硬件堆栈**，通过附加控制电路，让它与主存中的堆栈区在逻辑上组成一个整体，使堆栈的**访问速度**是寄存器的，但**容量**是主存的；
- 有丰富的堆栈操作类指令，功能很强，可以直接对堆栈中的数据进行处理；
- 有力地支持高级语言的编译，简化了编译，缩小了高级语言与机器语言的语义差距。



- 有力地支持子程序的嵌套调用和递归调用;
- 堆栈机器的存储效率高, 因为:
 - ✓及时释放不用单元;
 - ✓在访问堆栈时较多地使用零地址指令;
 - ✓在访问主存时一般采用相对寻址, 使访存的地址码位数减少, 从而使堆栈机器上程序的总位数, 以及执行所要用到的存储单元数减少。

算术赋值语句
 $F = A * B + C / (D - E)$

逆波兰表达式
 $AB * CDE - / +$

作为编译时的中间语言, 可以直接生成堆栈机器指令程序。

• 确定数据表示的原则

- 一是缩短实现的时间;
- 二是减少所需存储空间;

- 三是这种数据表示的通用性和利用率

□通用性：例如是否能高效支持多种数据结构，所花费的硬件代价是否带来性能上的提高。

□利用率：例如硬件设备量是否过多等。

- 从70年代开始，经过近10年的实践和探索，目前除了基本数据表示外，已根据使用环境分别引入了较复杂的堆栈数据表示、自定义数据表示和向量数据表示。
- 数据表示在不断发展**，如：矩阵、树、图、表及自定义数据表示等。
- 现有的基本数据表示也还存在着一些问题。关于这个问题的讨论，请同学们阅读相关资料。

数据校验码是指那些能够发现错误或能够自动纠正错误的**数据编码**，又称之为“**检错纠错编码**”。

- 编码的最小距离：合法代码集中任意两组合法代码之间二进制位数的最少差异。编码的检错、纠错能力与编码最小距离直接相关。
- 公式描述为： $L-1 = D+C(D \geq C)$

L: 编码的最小距离

D: 检错的位数

C: 纠错的位数

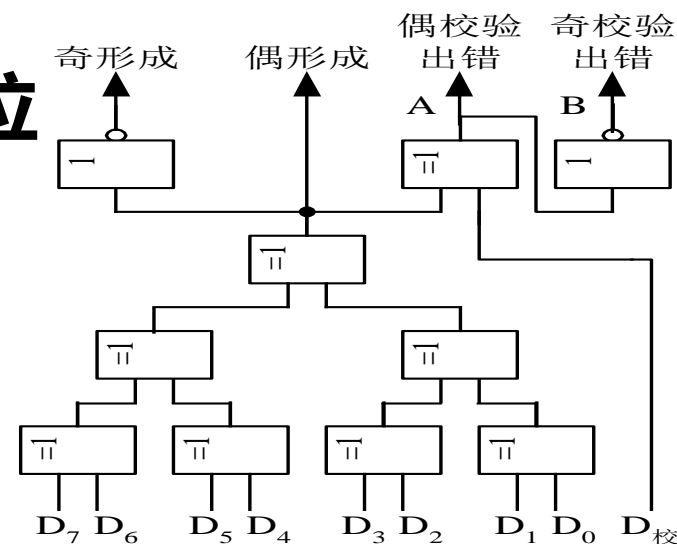
1. 奇偶校验概念

奇偶校验码是一种最简单的数据校验码，它可以检测出**一位**（或奇数位）错误。奇偶校验码的码距等于2。

奇偶校验实现方法是：由若干位有效信息（如一个字节），再加上一个二进制位（校验位）组成校验码，然后根据校验码的奇偶性质进行校验。

奇偶校验码（N+1位）= N位有效信息 + 1位校验位

奇偶校验位



校验位的取值（0或1）将使整个校验码中“1”的个数为奇数或偶数，所以有两种可供选择的校验规律：

奇校验——整个校验码（有效信息位和校验位）中“1”的个数为奇数。

偶校验——整个校验码中“1”的个数为偶数。

有效信息（8 位）	奇检验码（9 位）	偶检验码（9 位）
00000000	100000000	000000000
01010001	001010001	101010001
01111111	001111111	101111111
11111111	111111111	011111111

3.交叉奇偶校验

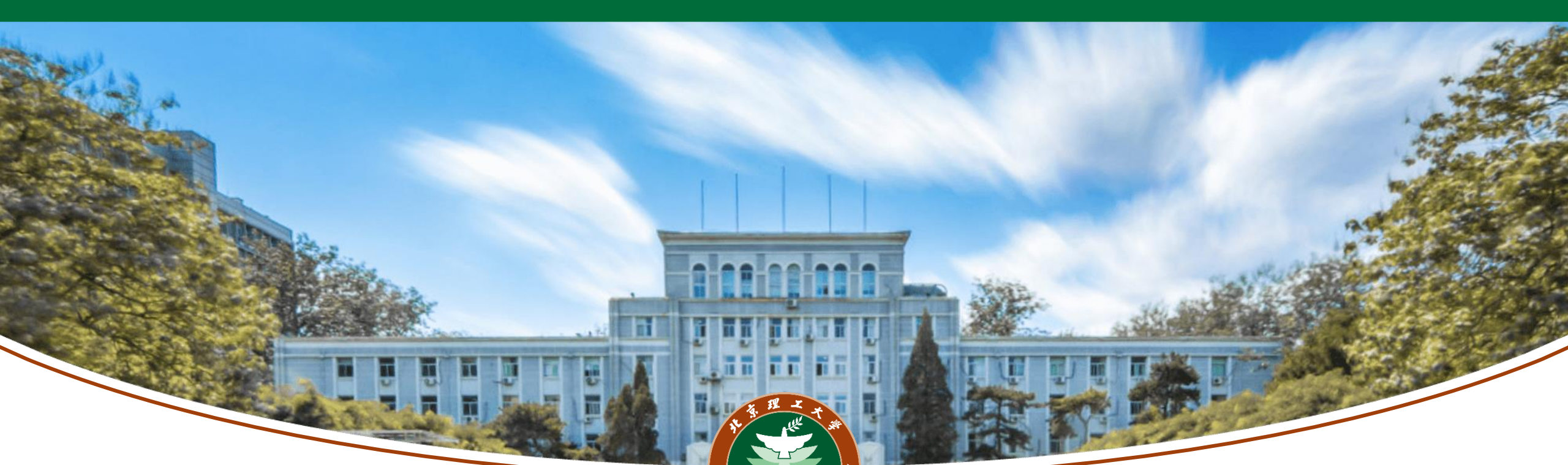
计算机在进行大量字节（数据块）传送时，不仅每一个字节有一个奇偶校验位做横向校验，而且全部字节的同一位也设置一个奇偶校验位做纵向校验，这种横向、纵向同时校验的方法称为交叉校验。

第1字节	1	1	0	0	1	0	1	1	→	1
第2字节	0	1	0	1	1	1	0	0	→	0
第3字节	1	0	0	1	1	0	1	0	→	0
第4字节	1	0	0	1	0	1	0	1	→	0
	↓	↓	↓	↓	↓	↓	↓	↓		
	1	0	0	1	1	0	0	0		

交叉校验可以发现两位同时出错的情况，假设第2字节的 a_6 、 a_4 两位均出错，横向校验位无法检出错误，但是第 a_6 、 a_4 位所在列的纵向校验位会显示出错，这与前述的简单奇偶校验相比要保险多了。

请自行查阅资料学习：

- 1) 海明校验码
- 2) 循环冗余校验码



感谢聆听