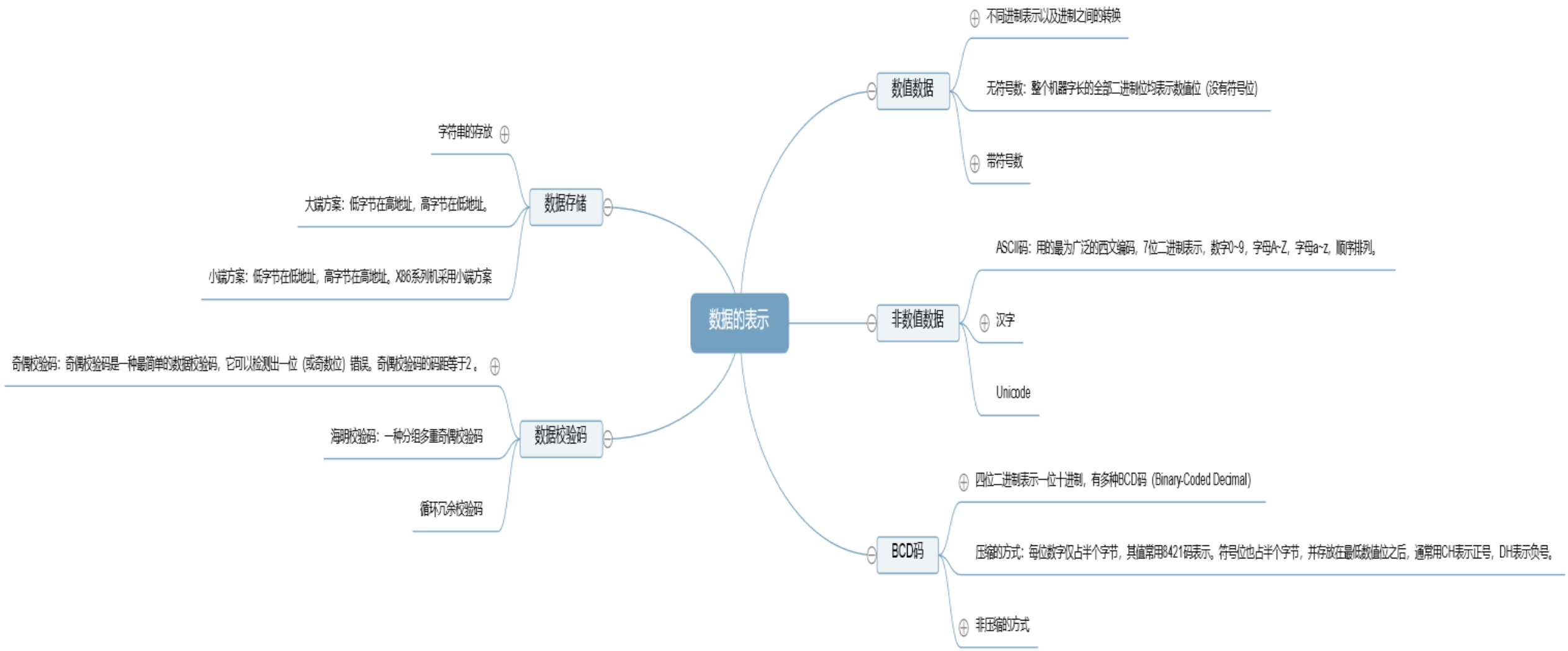
计算机组成与结构-数据的表示

人工智能专业

主讲教师：王 娟

数据的表示思维导图


数据的表示
数值数据
不同进制表示以及进制之间的转换
无符号数：整个机器字长的全部二进制位均表示数值位（没有符号位）
带符号数
非数值数据
ASCII码：用的最为广泛的西文编码，7位二进制表示，数字0~9，字母A~Z，字母a~z，顺序排列。
汉字
Unicode
BCD码
四位二进制表示一位十进制，有多种BCD码（Binary-Coded Decimal）
压缩的方式：每位数字仅占半个字节，其值常用8421码表示。符号位也占半个字节，并存放在最低数值位之后，通常用CH表示正号，DH表示负号。
非压缩的方式
数据存储
字符串的存放
大端方案：低字节在高地址，高字节在低地址。
小端方案：低字节在低地址，高字节在高地址。X86系列机采用小端方案
数据校验码
奇偶校验码：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。
海明校验码：一种分组多重奇偶校验码
循环冗余校验码

2.1 数值数据的表示

2.2 机器数的定点表示与浮点表示

2.3 非数值数据的表示

2.4 十进制数和数串的表示（自学）

2.5 不同类型的数据表示举例（自学）

2.6 数据校验码

数值数据表示的三要素

- 进位记数制
- 定、浮点表示
- 如何用二进制编码

即：要确定一个数值数据的值必须先确定这三个要素。

例如，机器数 01011001的值是多少?　　答案是：不知道！

- 进位记数制
  - 十进制、二进制、十六进制、八进制数及其相互转换
- 定/浮点表示（解决小数点问题）
  - 定点整数、定点小数
  - 浮点数（可用一个定点小数和一个定点整数来表示）
- 定点数的编码（解决正负号问题）
  - 原码、补码、反码、移码 （反码很少用）

数值数据的表示

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 ~ 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。

- $[X]_{原}$符号位加绝对值。当X为正数时，$[X]_{补}=[X]_{原}=[X]_{反}$。
- X为负数时，由$[X]_{原}$转换为$[X]_{补}$的方法：

①$[X]_{原}$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：

$$[X]_{原}=1.1110011000$$
$$[X]_{补}=1.0001101000$$

1 不变　0001 变反　1000 不变

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

零的机器数

在原码表示中，真值0有两种不同的表示形式：

$[+0]_{原}$ =00000

$[-0]_{原}$ =10000

在补码表示中，真值0的表示形式是唯一的。

$[+0]_{补}=[-0]_{补}$=00000

在反码表示中，真值0也有两种不同的表示形式：

$[+0]_{反}$=00000

$[-0]_{反}$ =11111

三种机器数的总结

(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。

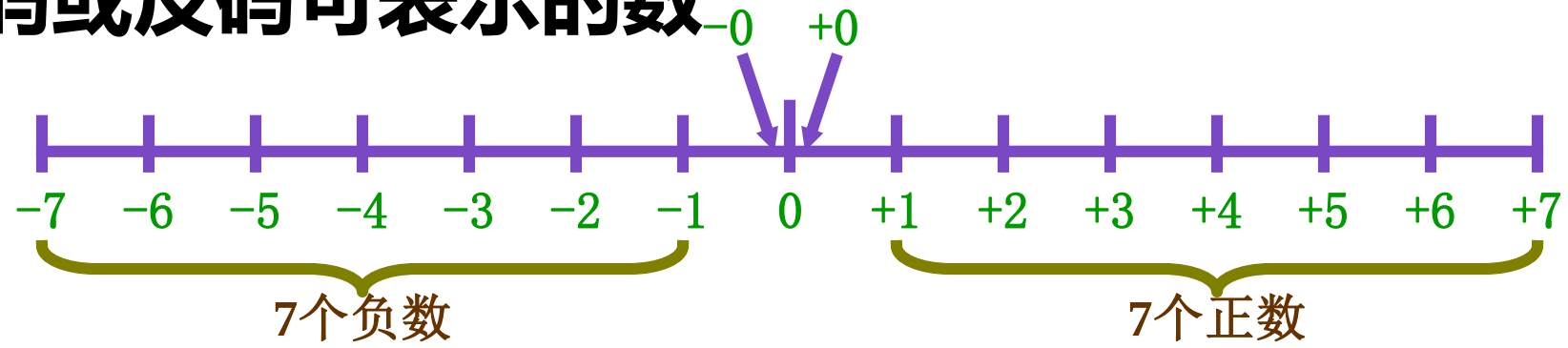
(3)对于真值0，原码和反码各有两种不同的表示形式，而补码只有唯一的一种表示形式。

(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）。

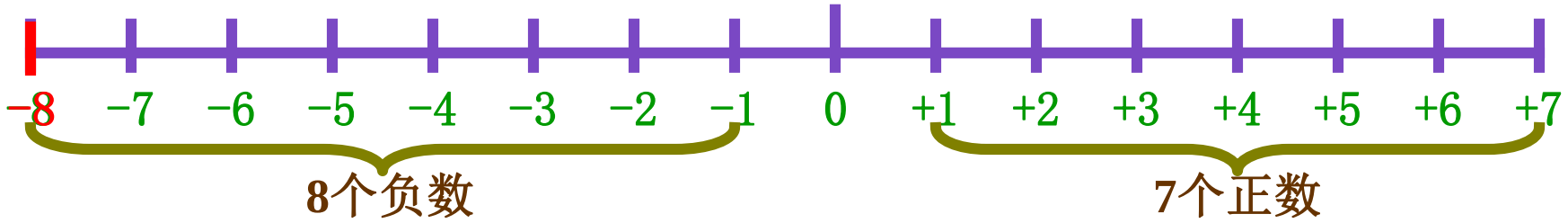
三种机器数范围示例

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）



（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	X2=-1011
[X1]原=0,0001011	[X2]原=1,0001011
[X1]补=0,0001011	[X2]补=1,1110101
[X1]反=0,0001011	[X2]反=1,1110100
X1=0.1011	X2=-0.1011
[X1]原=0.1011000	[X2]原=1.1011000
[X1]补=0.1011000	[X2]补=1.0101000
[X1]反=0.1011000	[X2]反=1.0100111

2.1 数值数据的表示

2.2 机器数的定点表示与浮点表示

2.3 非数值数据的表示

2.4 十进制数和数串的表示（自学）

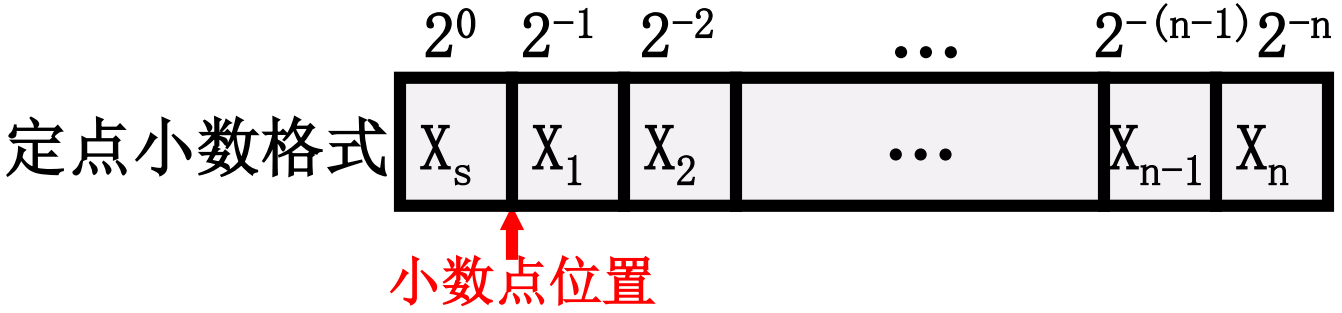
2.5 不同类型的数据表示举例（自学）

2.6 数据校验码

定点表示法

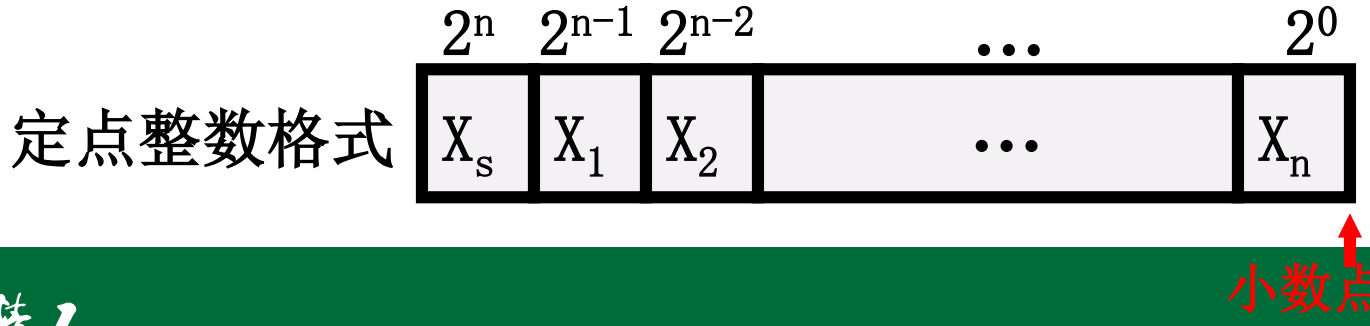
定点小数

小数点的位置固定在最高有效数位之前，符号位之后，记作$X_s.X_1X_2\cdots X_n$，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。



定点整数

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



表示范围

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：$-128 \sim 127$

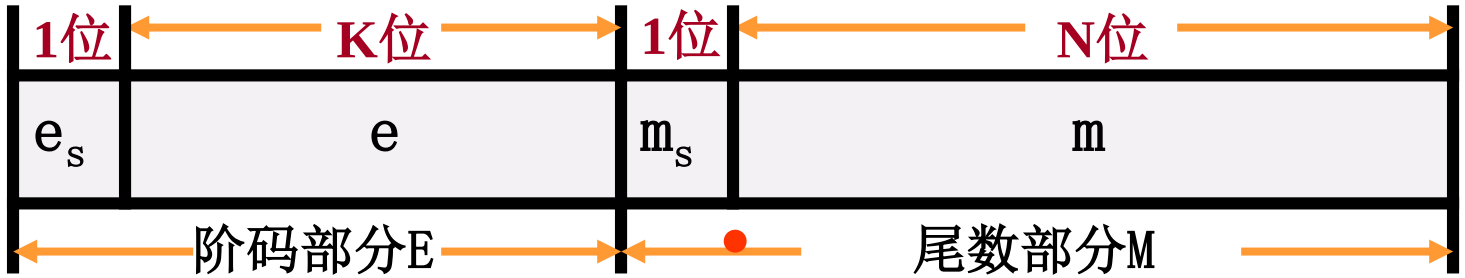
浮点表示法

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**。

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

如果r=2，则有$1/2 \le |M| < 1$。

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$x_s.1xx\ldots x$。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$1/2 \le M < 1$时，应有0.1xx…x形式，当$-1 \le M < -1/2$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。

$[X]_{移}$=偏置值+X

字长n+1位定点整数的移码形式为$X_0X_1X_2\cdots X_n$。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2^7。

例1：X=1011101

$[X]_{移}=2^7+X$

$=10000000+1011101$

$=11011101$

$[X]_{补}=01011101$

例2：X=-1011101

$[X]_{移}=2^7+X$

$=10000000-1011101$

$=00100011$

$[X]_{补}=10100011$

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$[+0]_{移}=[-0]_{移}=100...0$。

(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。

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。

十进制与短浮点格式的相互转换

例1：将$(100.25)_{10}$转换成短浮点数格式。

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

(2)非规格化数→规格化数　　$1100100.01=1.10010001\times2^6$

(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=10010001000000000000000

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?

2.1 数值数据的表示

2.2 机器数的定点表示与浮点表示

2.3 非数值数据的表示

2.4 十进制数和数串的表示（自学）

2.5 不同类型的数据表示举例（自学）

2.6 数据校验码

非数值数据的表示与存放

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

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

非数值数据的表示-汉字的表示

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）。

非数值数据的表示-汉字的表示

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

非数值数据的表示-汉字的表示

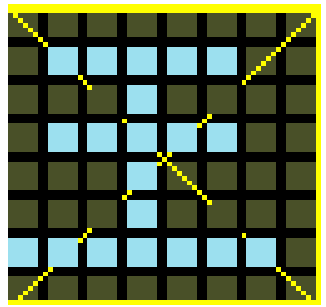
北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

3) 汉字字形码

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。

字形常见有两种描述方法：

- 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。

- 轮廓描述（图形方式）（自学）
  - 直线向量轮廓
  - 曲线轮廓（True Type字形）

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。

其他信息的表示

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

4）高级语言中的数据表示

2.1 数值数据的表示
2.2 机器数的定点表示与浮点表示
2.3 非数值数据的表示
2.4 十进制数和数串的表示（自学）
2.5 不同类型的数据表示举例（自学）
2.6 数据校验码

数据校验码

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>=C)

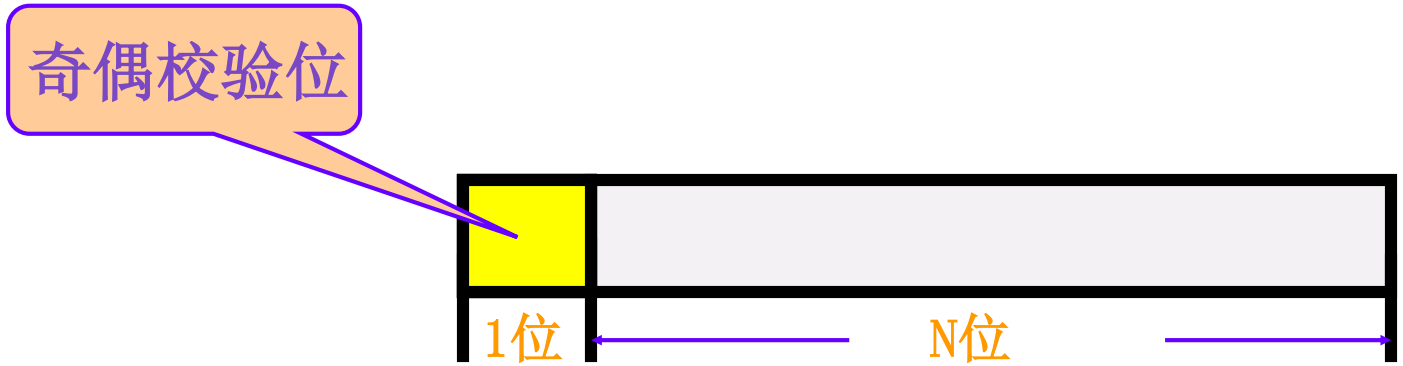
L: 编码的最小距离
D: 检错的位数
C：纠错的位数

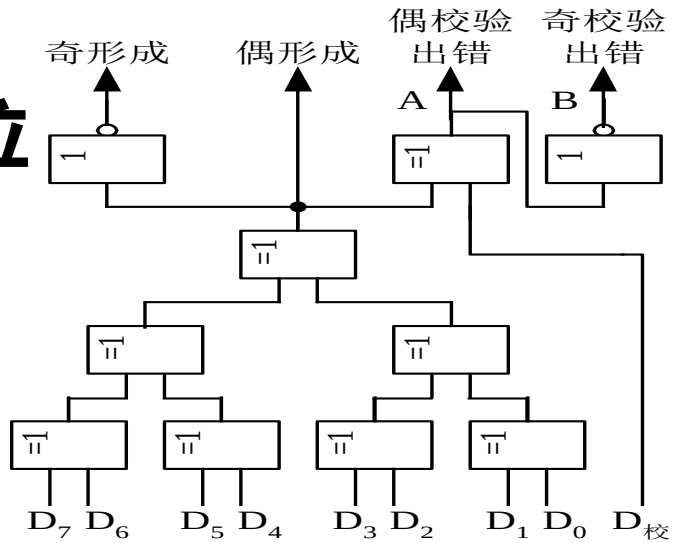
1.奇偶校验概念

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





数据校验码

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

数据校验码

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。

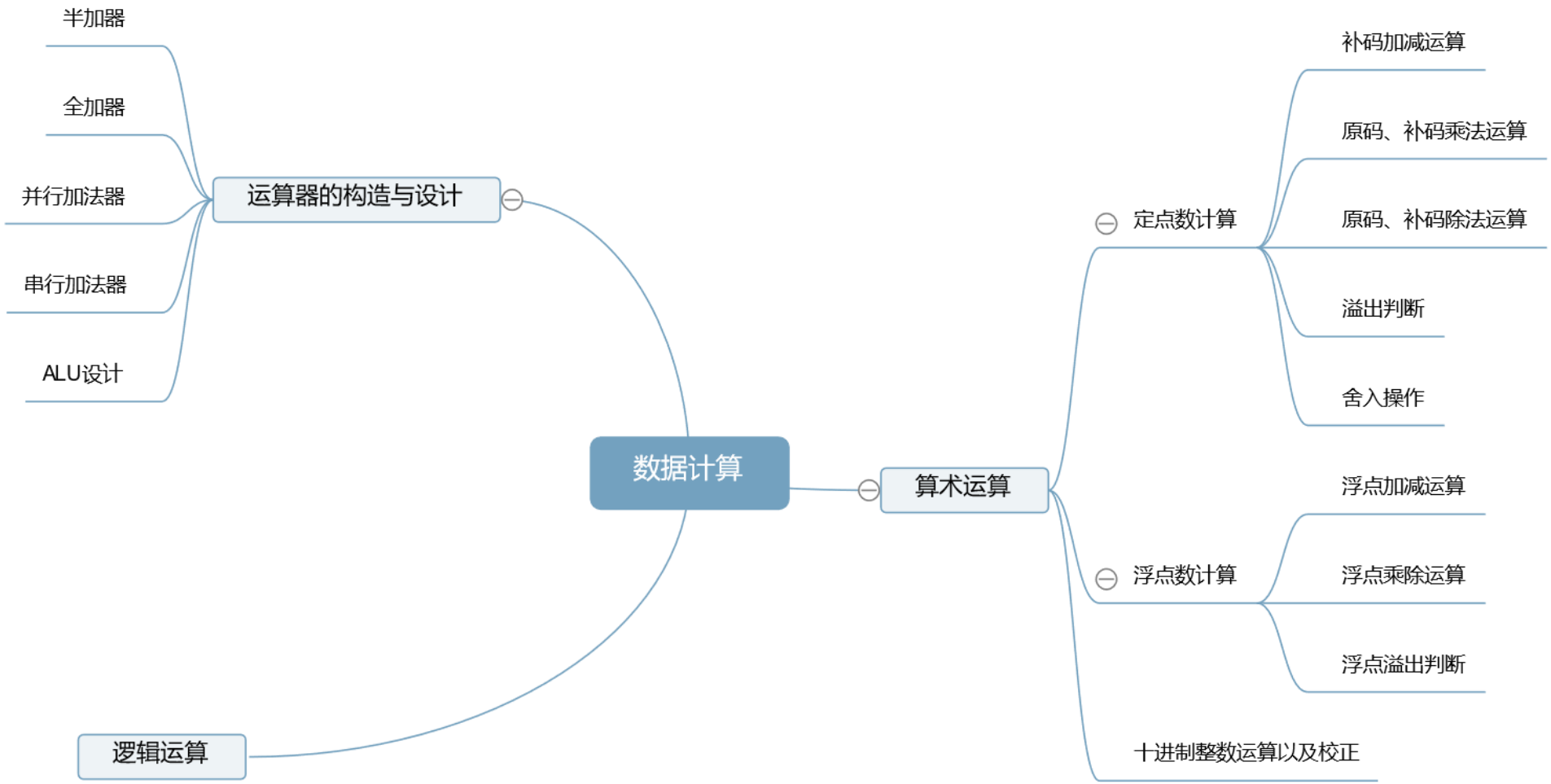
其他校验码

请自行查阅资料学习：

1）海明校验码

2）循环冗余校验码

数据计算学习思维导图


数据计算
运算器的构造与设计
半加器
全加器
并行加法器
串行加法器
ALU设计
逻辑运算
算术运算
定点数计算
补码加减运算
原码、补码乘法运算
原码、补码除法运算
溢出判断
舍入操作
浮点数计算
浮点加减运算
浮点乘除运算
浮点溢出判断
十进制整数运算以及校正

数据计算学习建议

掌握数值数据在计算机中实现算术运算和逻辑运算的方法，以及运算部件的基本结构和工作原理。

学习内容：教材第四章。

定点数计算：补码加减运算、补码的溢出判断与检测方法、补码常用舍入操作、定点乘除法运算（原码与补码）

浮点数计算：规格化浮点运算

十进制与逻辑运算：十进制整数的加减运算、逻辑运算与实现（自学）

运算器设计基础：进位的产生和传递，运算器的基本组成与设计

定点数运算

计算机内定点数一般用补码表示；思考为什么?

补码加减运算规则如下：

(1)参加运算的两个操作数均用补码表示；

(2)符号位作为数的一部分参加运算；

(3)若做加法，则两数直接相加；若做减法，则将被减数与减数的机器负数相加；

(4)运算结果用补码表示。

结论公式：

$$[X+Y]_{补} = [X]_{补} + [Y]_{补} \quad (1)$$

$$[X-Y]_{补} = [X]_{补} + [-Y]_{补} \quad (2)$$

其中，$[-Y]_{补}$ 称为$=[Y]_{补}$的机器负数，$[-Y]_{补}=[[Y]_{补}]_{变补}$

$[Y]_{补}$连同符号一起变反、末尾+1。

例1：A=0.1011，B=-0.1110，

求：A+B

∵$[A]_{补}$=0.1011，$[B]_{补}$=1.0010

$$\begin{array}{r} 0.1011 \\ +\ 1.0010 \\ \hline 1.1101 \end{array}$$

∴$[A+B]_{补}$=1.1101，A+B=-0.0011

例2：A=0.1011，B=-0.0010，

求：A-B

∵$[A]_{补}$=0.1011，$[B]_{补}$=1.1110，

$[-B]_{补}$=0.0010

$$\begin{array}{r} 0.1011 \\ +\ 0.0010 \\ \hline 0.1101 \end{array}$$

∴$[A-B]_{补}$=0.1101，A-B=0.1101

两数补码加减计算结果一定可用吗?

补码计算溢出判断

思考：什么时候会溢出?

溢出检测方法

设：被操作数为：$[X]_{补}=X_s,X_1X_2\ldots X_n$　　操作数为：$[Y]_{补}=Y_s,Y_1Y_2\ldots Y_n$

其和（差）为：$[S]_{补}=S_s,S_1S_2\ldots S_n$　　产生的进位为$C_s,C_1C_2\ldots C_n$

(1)硬件检测方法一：采用一个符号位

溢出$=\bar{X}_s\bar{Y}_sS_s+X_sY_s\bar{S}_s$

(2)硬件检测方法二：采用进位位

溢出$=C_s\overline{C}_1+\overline{C}_sC_1=C_s\oplus C_1$

(3)硬件检测方法三：采用变形补码（双符号位）

溢出$=S_{s1}\oplus S_{S2}$

双符号位溢出判断

在双符号位的情况下，把左边的符号位S_{s1}叫做真符，因为它代表了该数真正的符号，两个符号位都作为数的一部分参加运算。这种编码又称为变形补码。变形补码存储时仍然保存单符号位，运算时扩充成双符号位。

双符号位的含义如下：

$S_{s1}S_{s2}=00$ 结果为正数，无溢出

$S_{s1}S_{s2}=01$ 结果正溢

$S_{s1}S_{s2}=10$ 结果负溢

$S_{s1}S_{s2}=11$ 结果为负数，无溢出

变形补码的本质是扩大了模，对于定点小数来说，模为4，对于字长为n+2位的整数来说，模为2^{n+2}

n位加法

进位产生 用G_i表

进位传递函数 用P_i表示

- 基本加法器

1位全加器的逻辑表达式为

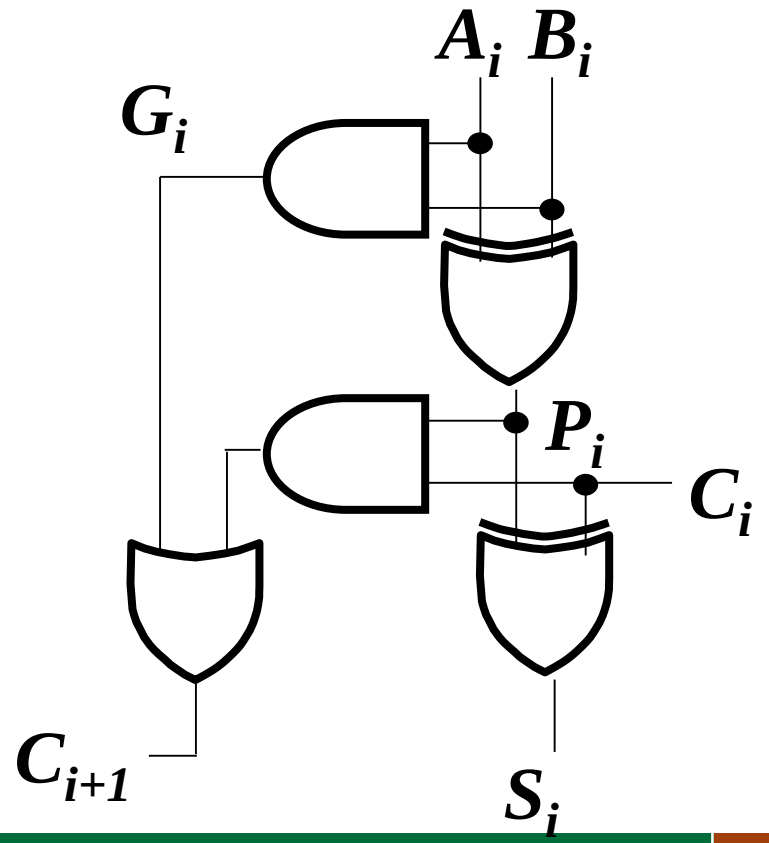
$$S_i=A_i\oplus B_i\oplus C_{i-1}$$

$$C_i=A_iB_i+(A_i\oplus B_i)C_{i-1}$$

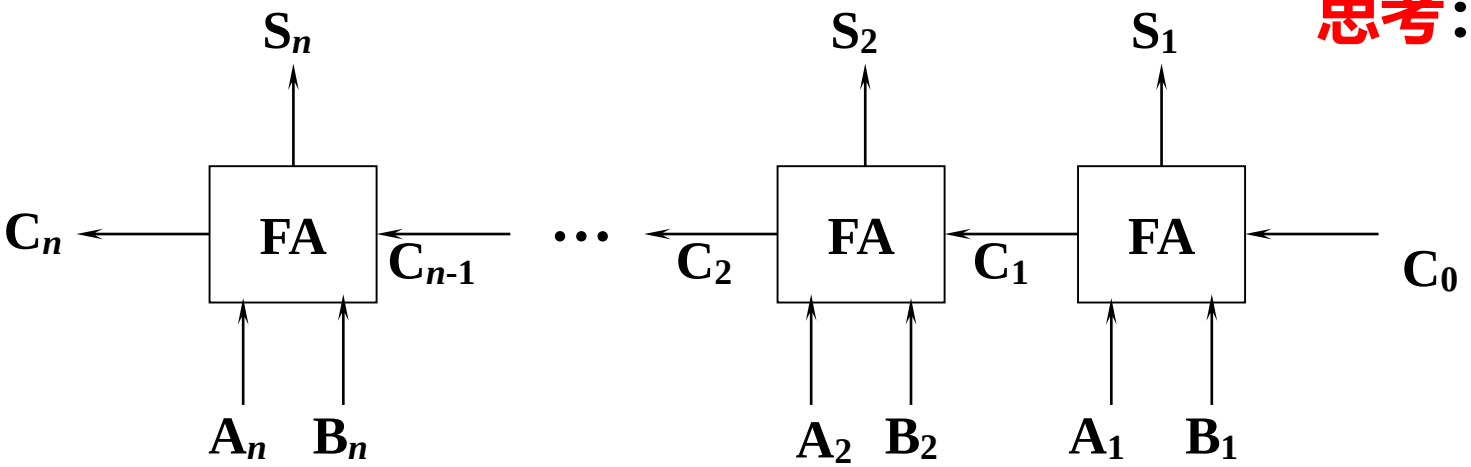
串行加法器：只有**一个全加器**，数据逐位串行送入加法器进行运算。如果操作数长*n*位，加法就要分*n*次进行，每次只能产生一位和。

并行加法器：并行加法器由**多个全加器**组成，其位数的多少取决于机器的字长，数据的各位同时运算。

Carry Generating Function
Carry Propagating Function



n位行波进位加法器



思考：进位延迟与进位的关系？能更快吗？如何改进？

每形成一级进位的延迟时间为2ty。在字长为n位的情况下，若不考虑G_i、P_i的形成时间，从$C_0 \rightarrow C_n$的最长延迟时间为2nty。

提高并行加法器速度的关键是尽量加快进位产生和传递的速度。

超前进位加法器

$C_1=G_1+P_1C_0$

$C_2=G_2+P_2C_1=G_2+P_2G_1+P_2P_1C_0$

$C_3=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1C_0$

$C_4=G_4+P_4G_3+P_4P_3G_2+P_4P_3P_2G_1+P_4P_3P_2P_1C_0$

……

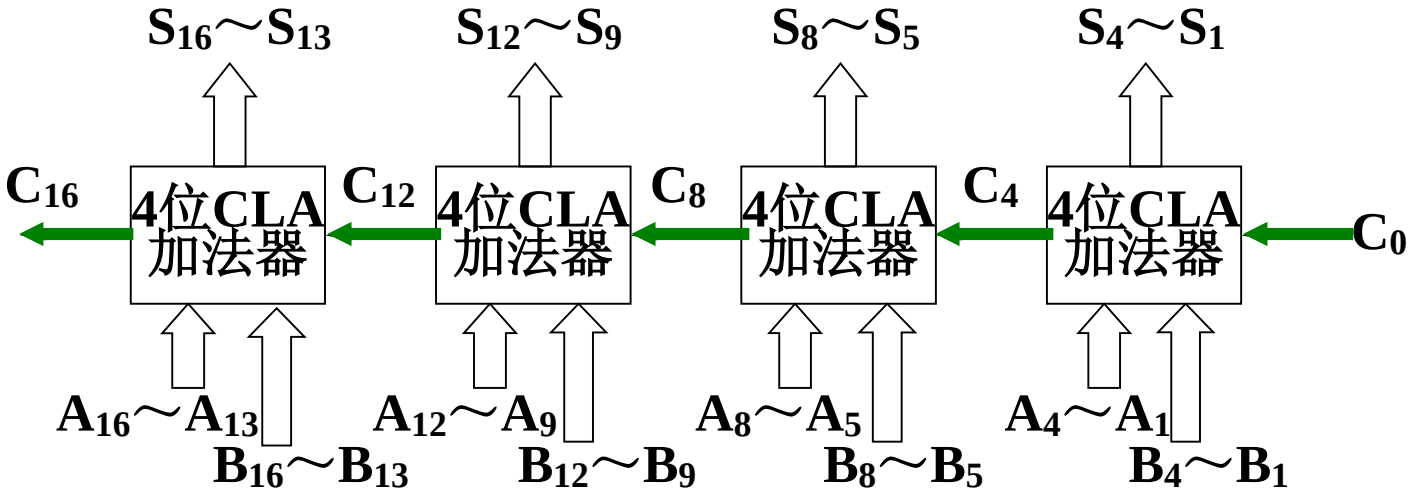
每个进位都不需要等待低位，直接计算可以得到，这种方法实现的加法器就被称为**超前进位加法器**（Carry_lookahead Adder，CLA）。

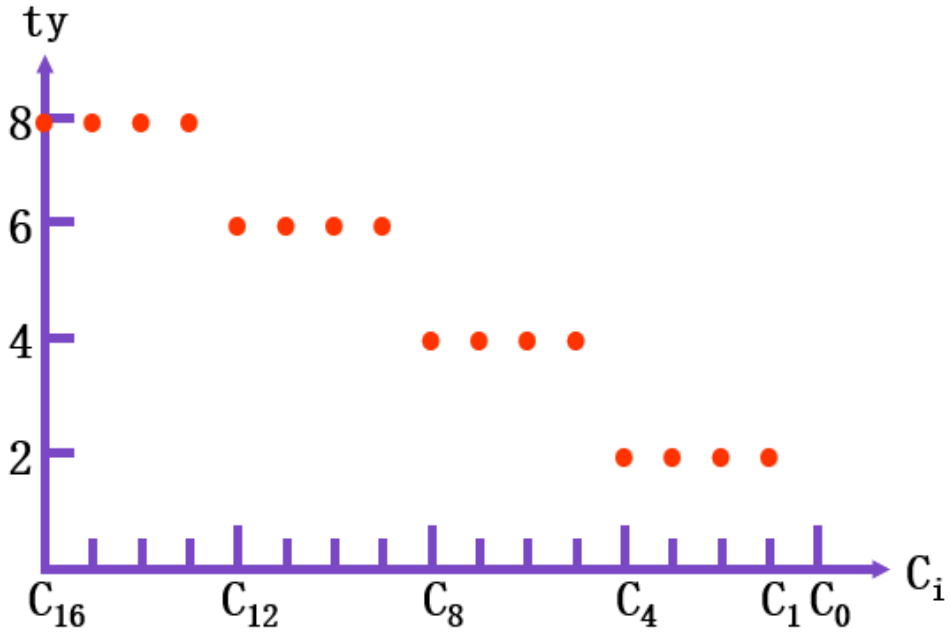
若不考虑G_i、P_i的形成时间，从$C_0 \rightarrow C_n$的最长延迟时间仅为2ty。

但随着加法器位数的增加，C_i的逻辑表达式会变得越来越长，实现十分复杂，**如何解决？ 分组控制规模**

(1)单级先行进位方式

这种进位方式又称为**组内并行、组间串行**方式。以16位加法器为例，可分为四组，每组四位。第1小组组内的进位逻辑函数C_1、C_2、C_3、C_4的表达式与前述相同，$C_1 \sim C_4$信号是同时产生的，从C_0出现到产生$C_1 \sim C_4$的延迟时间是2ty。





(2)多级先行进位方式

又称组内并行、组间并行进位方式。

字长为16位的两级先行进位加法器，第一小组的最高位进位C_4：

$$C_4=G_4+P_4G_3+P_4P_3G_2+P_4P_3P_2G_1+P_4P_3P_2P_1C_0=G_1^*+P_1^*C_0$$

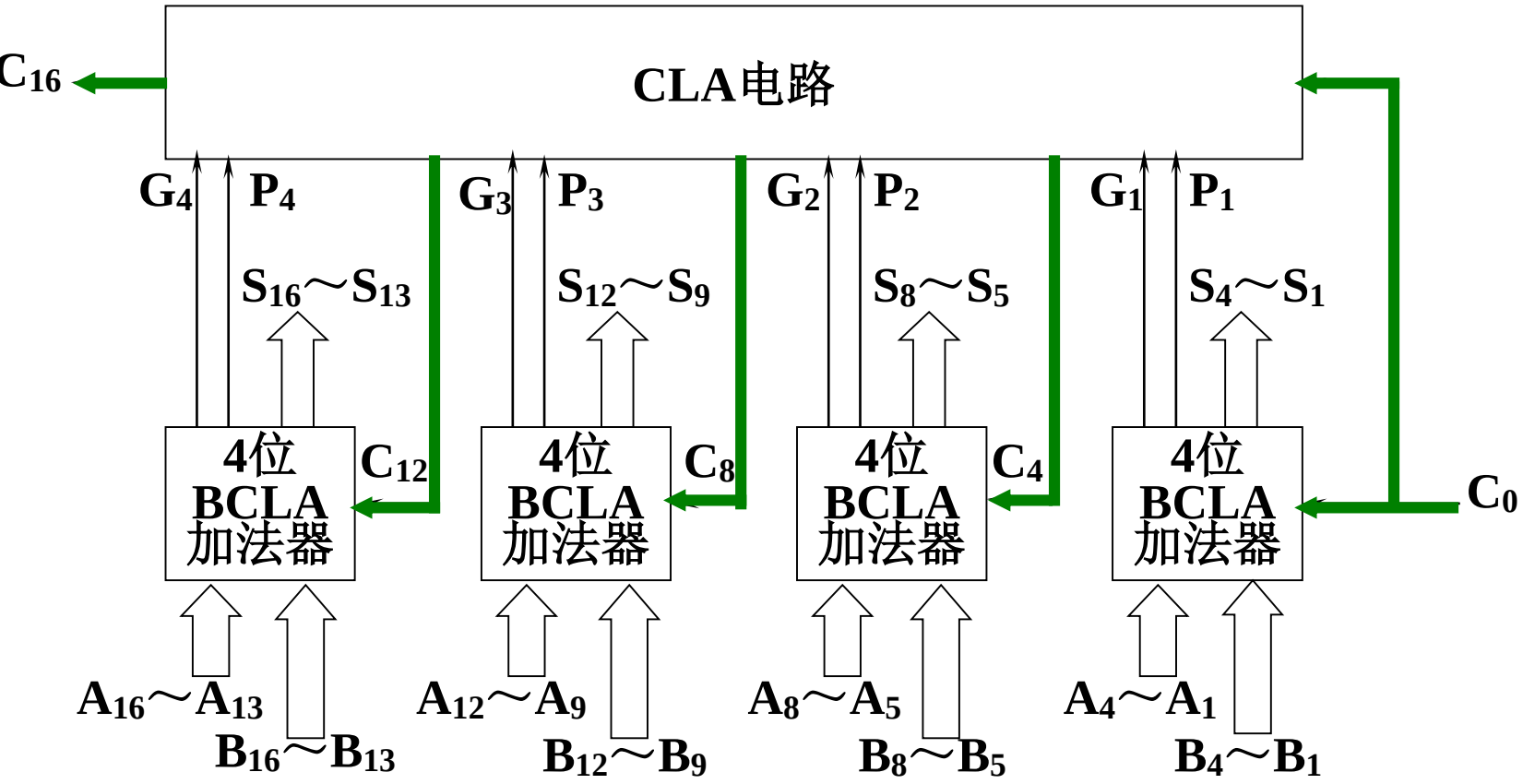
依次类推：

$$C_8=G_2^*+P_2^*G_1^*+P_2^*P_1^*C_0$$

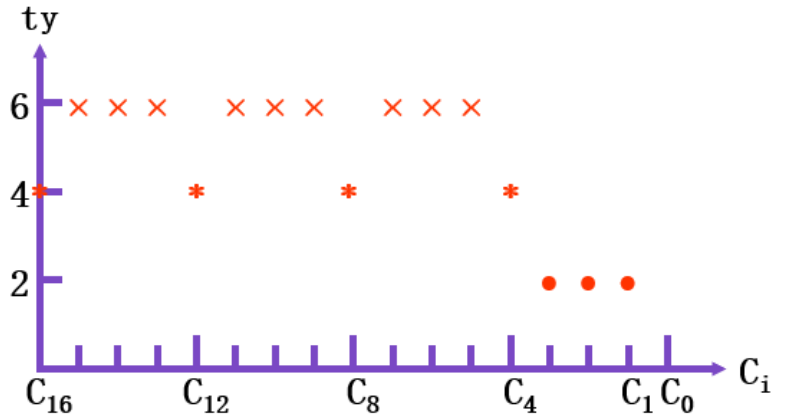
$$C_{12}=G_3^*+P_3^*G_2^*+P_3^*P_2^*G_1^*+P_3^*P_2^*P_1^*C_0$$

$$C_{16}=G_4^*+P_4^*G_3^*+P_4^*P_3^*G_2^*+P_4^*P_3^*P_2^*G_1^*+P_4^*P_3^*P_2^*P_1^*C_0$$

(2)多级先行进位方式



若不考虑G_i、P_i的形成时间，C_0经过2ty产生第1小组的C_1、C_2、C_3及所有组进位产生函数G_i^*和组进位传递函数P_i^*；再经过2ty，产生C_4、C_8、C_{12}、C_{16}；最后经过2ty后，才能产生第2、3、4小组内的C_5～C_7、C_9～C_{11}、C_{13}～C_{15}。



定点乘法运算

原码一位乘法类似于手工计算（自学）。

乘积 $P = |X| \times |Y|$　　符号 $P_s = X_s \oplus Y_s$

补码一位乘法：Booth乘法

乘法运算需要3个寄存器：

A寄存器：部分积与最后乘积的高位部分，初值为0。

B寄存器：被乘数X。

C寄存器：乘数Y，运算后C寄存器中不再需要保留乘数，改为存放乘积的低位部分。

Booth乘法

Booth乘法规则如下：

① 参加运算的数用补码表示；

② 符号位参加运算；

③ 乘数最低位后面增加一位附加位Y_{n+1}，其初值为0；

④ 由于每求一次部分积要右移一位，所以乘数的最低两位Y_n、Y_{n+1}的值决定了每次应执行的操作；

判断位 Y_n Y_{n+1}	操 作
0 0	原部分积右移一位
0 1	原部分积加$[X]_{补}$后右移一位
1 0	原部分积加$[-X]_{补}$后右移一位
1 1	原部分积右移一位

⑤ 移位按补码右移规则进行；

⑥ 共需做n+1次累加，n次移位，第n+1次不移位。

例：已知X=-0.1101，Y=0.1011；求X×Y。

$[X]_{补}$=1.0011→B，$[Y]_{补}$=0.1011→C，0→A　　$[-X]_{补}$=0.1101

	A	C	附加位	说明
	0 0.0 0 0 0	0.1 0 1 1	0	
$+[-X]_{补}$	0 0.1 1 0 1			C_4C_5=10，$+[-X]_{补}$
	0 0.1 1 0 1			
→	0 0.0 1 1 0	1 0 1 0	1 1	部分积右移一位
+0	0 0.0 0 0 0			C_4C_5=11，+0
	0 0.0 1 1 0			
→	0 0.0 0 1 1	0 1 0 1	0 1	部分积右移一位
$+[X]_{补}$	1 1.0 0 1 1			C_4C_5=01，$+[X]_{补}$
	1 1.0 1 1 0			
→	1 1.1 0 1 1	0 0 1 0	1 0	部分积右移一位
$+[-X]_{补}$	0 0.1 1 0 1			C_4C_5=10，$+[-X]_{补}$
	0 0.1 0 0 0			
→	0 0.0 1 0 0	0 0 0 1	0 1	部分积右移一位
$+[X]_{补}$	1 1.0 0 1 1			C_4C_5=01，$+[X]_{补}$
	1 1.0 1 1 1			

∵ $[X\times Y]_{补}$=1.01110001

∴ X×Y=−0.10001111

思考：还可以更快吗？

自学：补码两位乘法

定点除法运算

1.原码比较法、恢复余数法和不恢复余数法（原码加减交替法）（自学）

2.补码加减交替除法规则，求新余数公式： $[r_{i+1}]_{补}=2[r_i]_{补}+(1-2Q_i)\times[Y]_{补}$

$[X]_{补}$与$[Y]_{补}$	第一次操作	$[r_i]_{补}$与$[Y]_{补}$	上商	求新余数$[r_{i+1}]_{补}$的操作
同号	$[X]_{补}-[Y]_{补}$	①同号（够减）	1	$[r_{i+1}]_{补}=2[r_i]_{补}-[Y]_{补}$
		②异号（不够减）	0	$[r_{i+1}]_{补}=2[r_i]_{补}+[Y]_{补}$
异号	$[X]_{补}+[Y]_{补}$	①同号（不够减）	1	$[r_{i+1}]_{补}=2[r_i]_{补}-[Y]_{补}$
		②异号（够减）	0	$[r_{i+1}]_{补}=2[r_i]_{补}+[Y]_{补}$

已知： X=0.1000，Y=−0.1010；**求**X÷Y

$[X]_{补}$=0.1000→A，$[Y]_{补}$=1.0110→B，0→C，$[-Y]_{补}$=0.1010

	0 0.1 0 0 0	0.0 0 0 0	
$+[Y]_{补}$	1 1.0 1 1 0		$[X]_{补}$、$[Y]_{补}$异号，$+[Y]_{补}$
	1 1.1 1 1 0	0.0 0 0 1	$[r_i]_{补}$、$[Y]_{补}$同号，商1
←	1 1.1 1 0 0		左移一位
$+[-Y]_{补}$	0 0.1 0 1 0		$+[-Y]_{补}$
	0 0.0 1 1 0	0.0 0 1 0	$[r_i]_{补}$、$[Y]_{补}$异号，商0
←	0 0.1 1 0 0		左移一位
$+[Y]_{补}$	1 1.0 1 1 0		$+[Y]_{补}$
	0 0.0 0 1 0	0.0 1 0 0	$[r_i]_{补}$、$[Y]_{补}$异号，商0
←	0 0.0 1 0 0		左移一位
$+[Y]_{补}$	1 1.0 1 1 0		$+[Y]_{补}$
	1 1.1 0 1 0	0.1 0 0 1	$[r_i]_{补}$、$[Y]_{补}$同号，商1
←	1 1.0 1 0 0		左移一位
$+[-Y]_{补}$	0 0.1 0 1 0		$+[-Y]_{补}$
	1 1.1 1 1 0	1.0 0 1 1	末位恒置1

除法运算需要3个寄存器：

A寄存器：存放被除数X，最后A寄存器中剩下的是扩大了若干倍的余数。运算过程中A寄存器的内容将不断地发生变化。

B寄存器：存放除数Y。

C寄存器：存放商Q，它的初值为0。

$[商]_{补}=1.0011$

$[余数]_{补}=1.1110\times2^{-4}$

$[X\div Y]$

$=1.0011+$

$\therefore 商=-0.1101$

$$\begin{array}{r} 1.1110\times2^{-4} \\ \hline 补\ 1.0110 \end{array}$$

$余数=-0.0010\times2^{-4}$

$$X\div Y=-0.1101+\frac{-0.0010\times2^{-4}}{-0.1010}$$

浮点加减运算

设两个非0的规格化浮点数分别为

$A=M_A\times 2^{E_A}$

$B=M_B\times 2^{E_B}$

$$A\pm B=(M_A,E_A)\pm(M_B,E_B)=\begin{cases}M_A\pm M_B\times 2^{-(E_A-E_B)},E_A) & E_A>E_B\\ (M_A\times 2^{-(E_B-E_A)}\pm M_B,E_B) & E_A<E_B\end{cases}$$

(1)对阶： 规则是：小阶向大阶看齐，每右移一位，阶码加1。

(2)尾数加/减 $M_A\pm M_B\rightarrow M_C$

(3)尾数结果规格化，设尾数用双符号位补码表示，经过加/减运算之后，可能出现以下六种情况：

浮点加减规格化

① 00.1 x x … x　　② 11.0 x x … x

③ 00.0 x x … x　　④ 11.1 x x … x

⑤ 01.x x x … x　　⑥ 10.x x x … x

第①、②种情况，已是规格化数。

第③、④种情况需要使尾数左移以实现规格化，这个过程称为左规。尾数每左移一位，阶码相应减1，直至成为规格化数为止。（左规可能需进行多次）

第⑤、⑥种情况在定点加减运算中称为溢出；但在浮点加减运算中，只表明此时尾数的绝对值大于1，而并非真正的溢出。这种情况应将尾数右移以实现规格化。这个过程称为右规。尾数每右移一位，阶码相应加1。（右规最多进行一次）

当尾数之和（差）出现10.x x x … x或01.x x x … x时，并不表示溢出，只有将此数右规后，再根据阶码来判断浮点运算结果是否溢出。

浮点数的溢出情况由阶码的符号决定，若阶码也用双符号位补码表示，

当：$[E_C]_{补}$=01，x x x … x，表示上溢。此时，浮点数真正溢出，机器需停止运算，做溢出中断处理。

$[E_C]_{补}$=10，x x x … x，表示下溢。浮点数值趋于零，机器不做溢出处理，而是按机器零处理。

浮点数加减计算举例见教材（自学）。

浮点乘除运算

设两个非0的规格化浮点数分别为

$$A=M_A\times 2^{E_A}$$

$$B=M_B\times 2^{E_B}$$

则浮点乘法和除法为

$$A\times B=(M_A\times M_B)\times 2^{(E_A+E_B)}$$

$$A\div B=(M_A\div M_B)\times 2^{(E_A-E_B)}$$

尾数按照定点小数的乘除法进行，阶码按照定点整数加减法进行。

阶码为移码时应注意：两个浮点数的阶码相加，当阶码用移码表示的时候，应注意要减去一个偏置值2^n。两浮点数的阶码相减，当阶码用移码表示时，应注意要加上一个偏置值2^n。

最后规格化。

运算器是在控制器的控制下实现其功能的，运算器不仅可以完成数据信息的算逻运算，还可以作为数据信息的传送通路。

1.运算器的基本组成

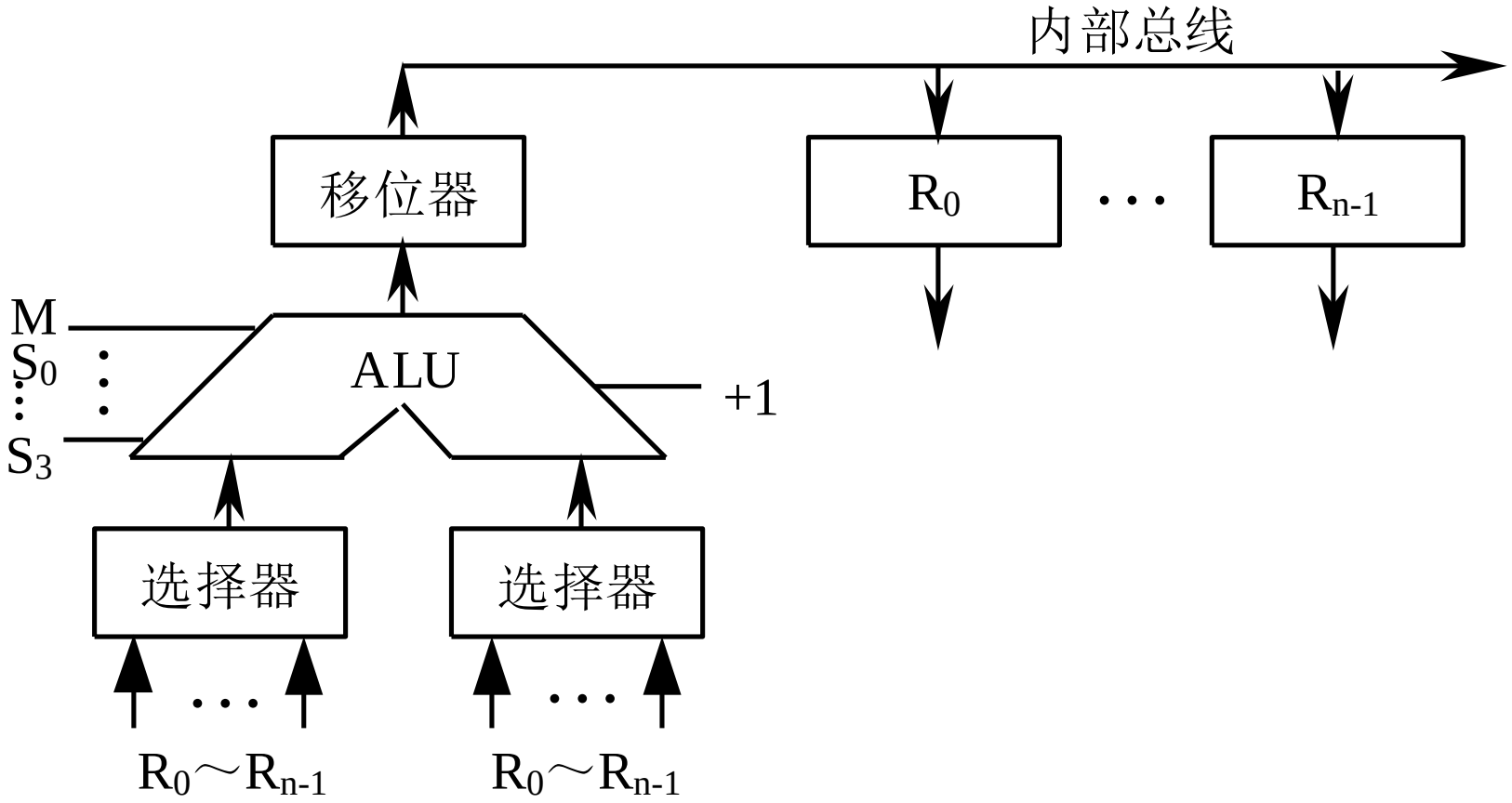
基本的运算器包含以下几个部分：

实现基本算术、逻辑运算功能的ALU，

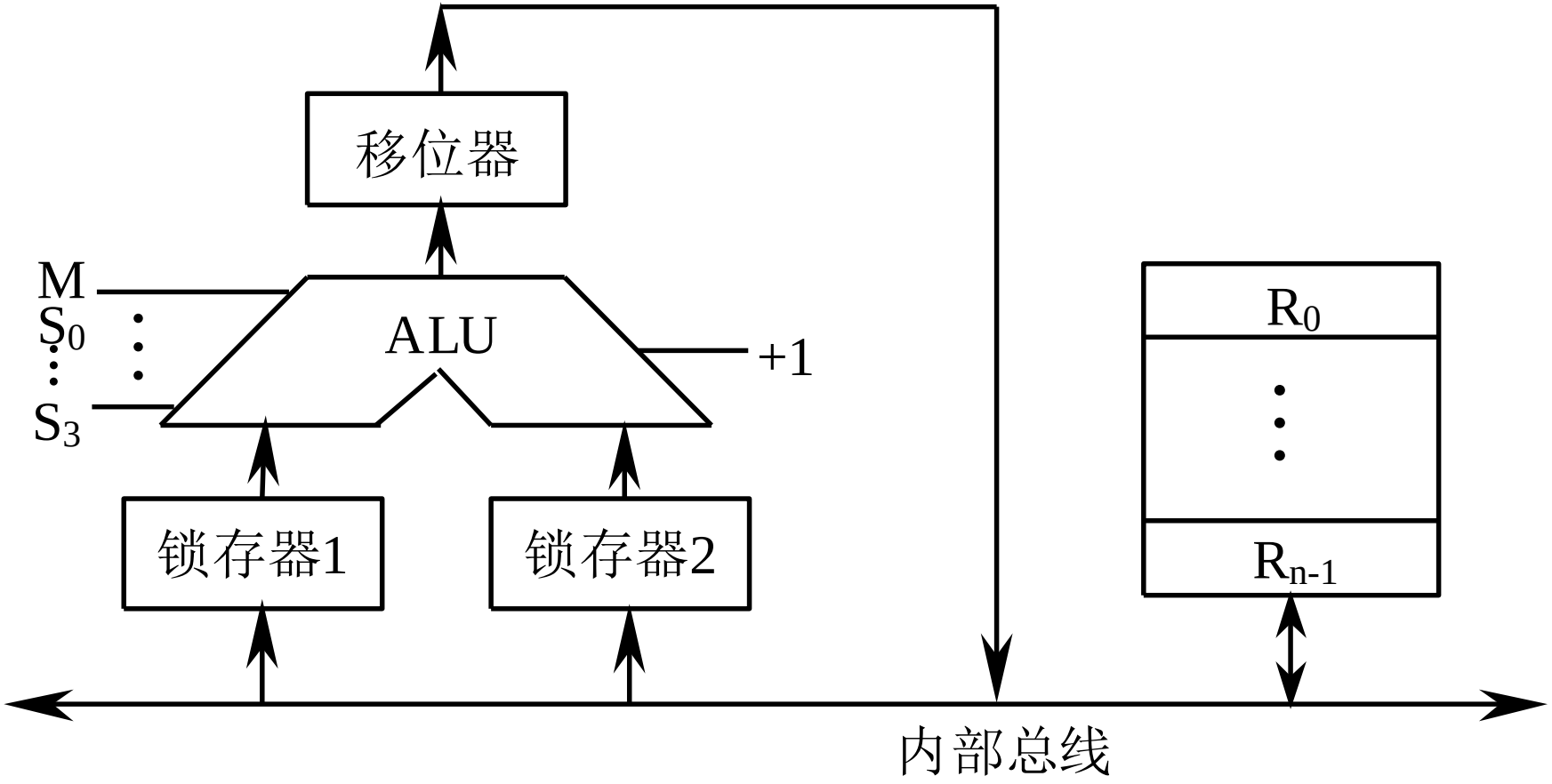
提供操作数与暂存结果的寄存器组，

有关的判别逻辑和控制电路等。

(1)带多路选择器的运算器



(2)带输入锁存器的运算器



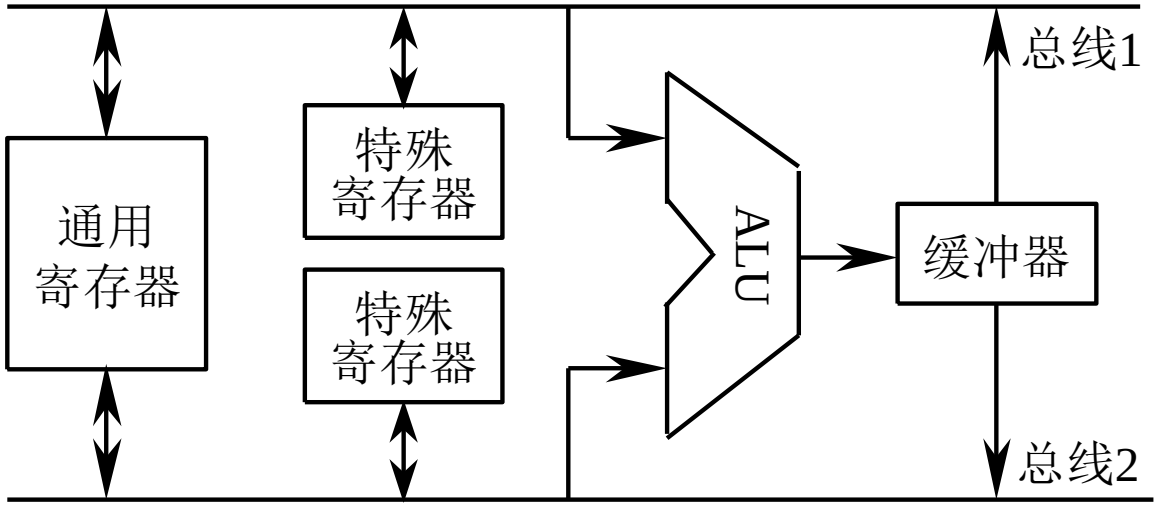
2.运算器的内部总线结构

(1)单总线结构运算器

运算器实现一次双操作数的运算需要分成三步。

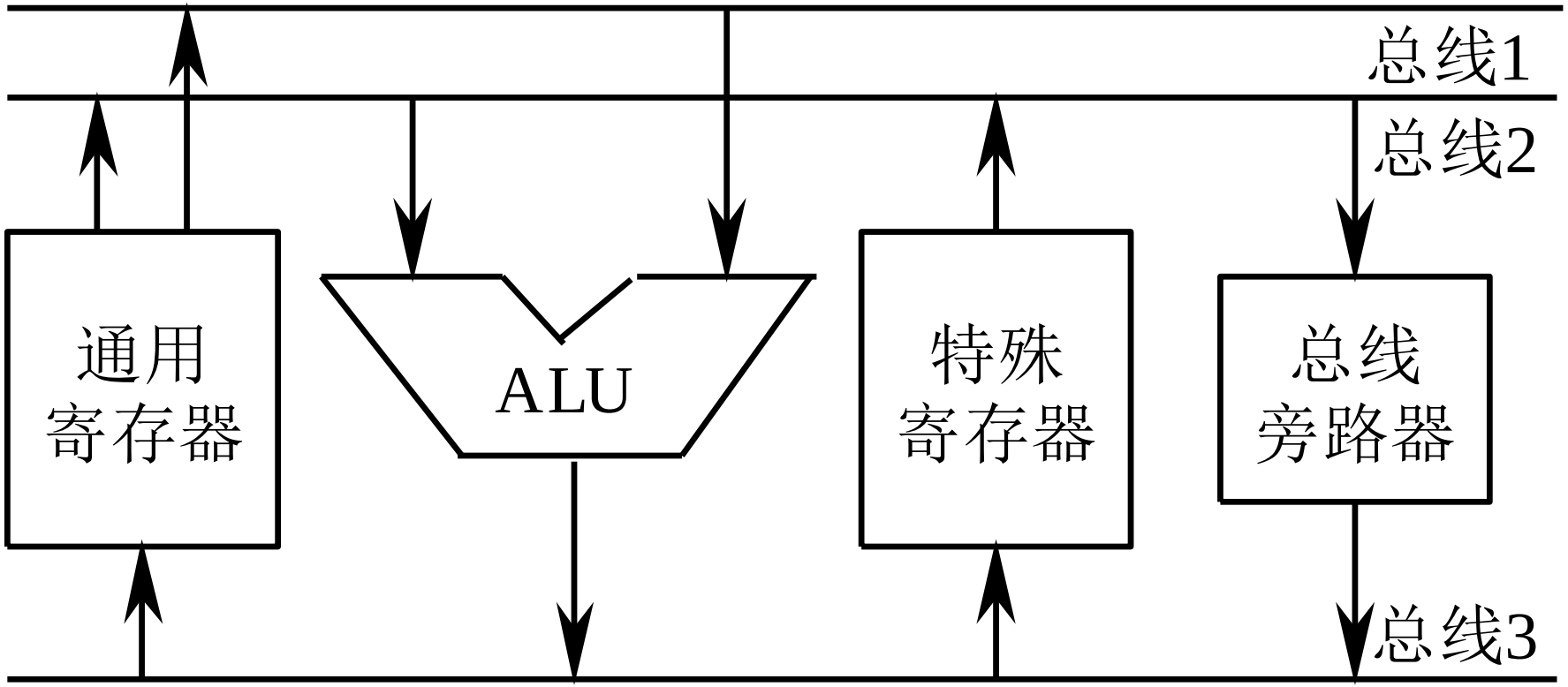
(2)双总线结构运算器

运算器实现一次双操作数的运算需要两步。



(3)三总线结构运算器

实现一次双操作数的运算仅需要一步。



1. ALU电路

ALU即算术逻辑单元，它是既能完成算术运算又能完成逻辑运算的部件。前面已经讨论过，无论是加、减、乘、除运算，最终都能归结为加法运算。因此，ALU的核心首先应当是一个并行加法器，同时也能执行像“与”、“或”、“非”、“异或”这样的逻辑运算。由于ALU能完成多种功能，所以ALU又称多功能函数发生器。

2.4位ALU芯片

74181是四位算术逻辑运算部件（ALU），又称多功能函数发生器，能执行16种算术运算和16种逻辑运算。

A0、B0～A3、B3：操作数输入端；

F0～F3：输出端；

C_n'：进位输入端；

C_{n+4}':进位输出端；

G*：组进位产生函数输出端；

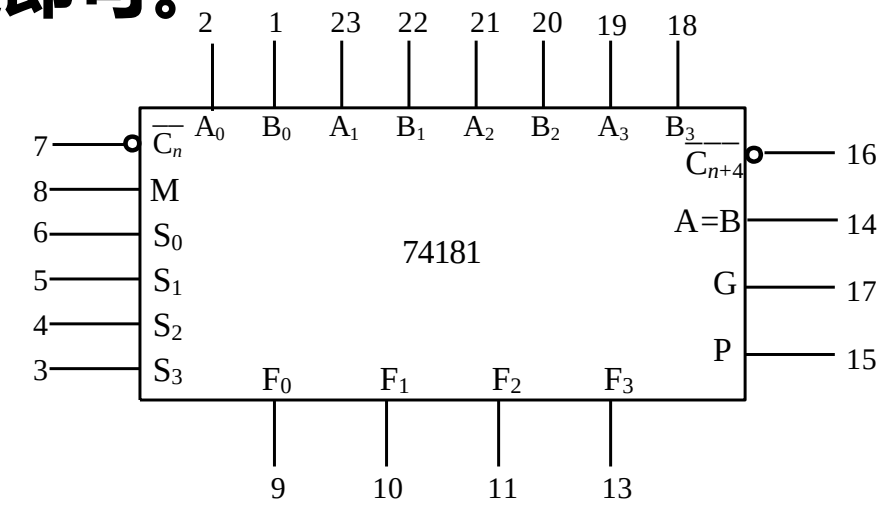
P*：组进位传递函数输出端；

M：工作方式，M=0为算术操作，M=1为逻辑操作；

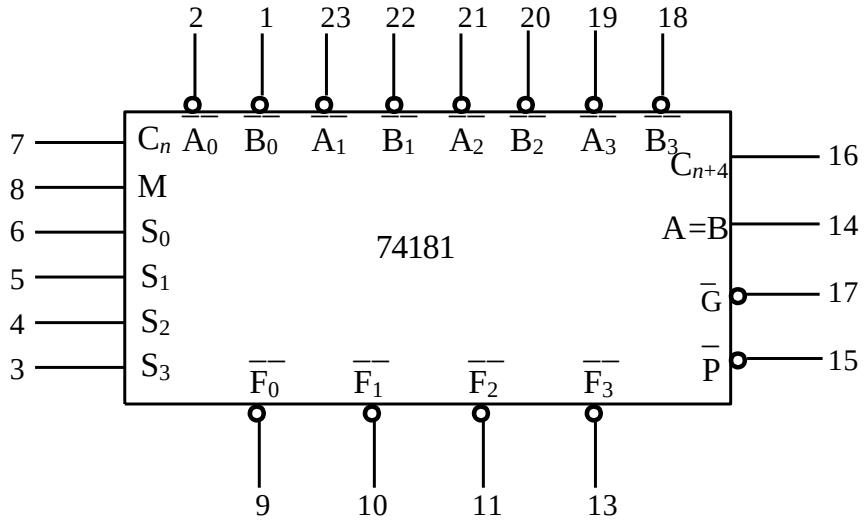
$S_0 \sim S_3$：功能选择线。

74181的4位作为一个小组，组间既可以采用串行进位，也可以采用并行进位。

当采用组间串行进位时，只要把前片的C_{n+4}与下一片的C_n相连即可。



(a)



(b)

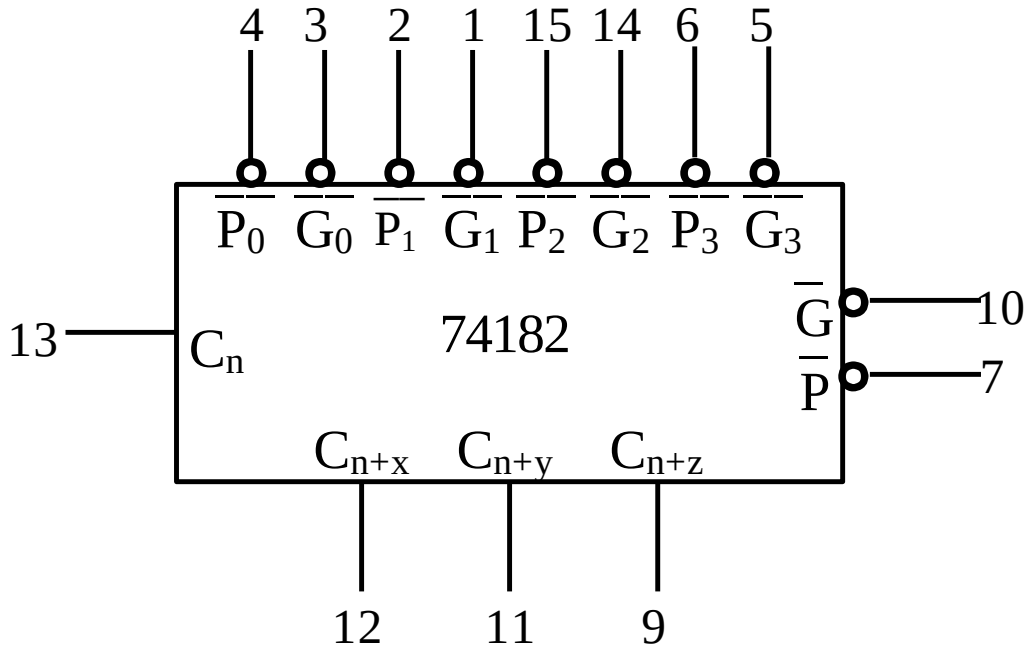
运算器的基本组成与实例

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

工作选择 $S_3S_2S_1S_0$	负逻辑			正逻辑		
	逻辑运算（M=1）	算术运算（M=0）C_n=0（无进位）	算术运算（M=0）C_n=1 有进位）	逻辑运算（M=1）	算术运算（M=0）$\overline{C_n}$=1（无进位）	算术运算（M=0）$\overline{C_n}$=0 有进位）
0000	$F=\overline{A}$	F=A 减 1	F=A	$F=\overline{A}$	F=A	F=A 加 1
0001	$F=\overline{AB}$	F=AB 减 1	F=AB	$F=\overline{A+B}$	F=A+B	F=(A+B)加 1
0010	$F=\overline{A}+B$	$F=A\overline{B}$ 减 1	$F=A\overline{B}$	$F=\overline{A}B$	$F=A+\overline{B}$	$F=(A+\overline{B})$加 1
0011	F=1	F=减 1	F=0	F=0	F=减 1	F=0
0100	$F=\overline{A+B}$	$F=A$ 加$(A+\overline{B})$	$F=A$ 加$(A+\overline{B})$加 1	$F=\overline{AB}$	F=A 加 $A\overline{B}$	F=A 加 $A\overline{B}$ 加 1
0101	$F=\overline{B}$	F=AB 加$(A+\overline{B})$	F=AB 加$(A+\overline{B})$加 1	$F=\overline{B}$	F=(A+B)加 $A\overline{B}$	F=(A+B)加 $A\overline{B}$ 加 1
0110	$F=\overline{A\oplus B}$	F=A 减 B 减 1	F=A 减 B	$F=A\oplus B$	F=A 减 B 减 1	F=A 减 B
0111	$F=A+\overline{B}$	$F=A+\overline{B}$	$F=(A+\overline{B})$加 1	$F=A\overline{B}$	$F=A\overline{B}$ 减 1	$F=A\overline{B}$
1000	$F=\overline{A}B$	F=A 加(A+B)	F=A 加(A+B)加 1	$F=\overline{A}+B$	F=A 加 AB	F=A 加 AB 加 1
1001	$F=A\oplus B$	F=A 加 B	F=A 加 B 加 1	$F=\overline{A\oplus B}$	F=A 加 B	F=A 加 B 加 1
1010	F=B	$F=A\overline{B}$ 加(A+B)	$F=A\overline{B}$ 加(A+B)加 1	F=B	$F=(A+\overline{B})$加 AB	$F=(A+\overline{B})$加 AB 加 1
1011	F=A+B	F=A+B	F=(A+B)加 1	F=AB	F=AB 减 1	F=AB
1100	F=0	F=A 加 A*	F=A 加 A 加 1	F=1	F=A 加 A*	F=A 加 A 加 1
1101	$F=A\overline{B}$	F=AB 加 A	F=AB 加 A 加 1	$F=A+\overline{B}$	F=(A+B)加 A	F=(A+B)加 A 加 1
1110	F=AB	$F=A\overline{B}$ 加 A	$F=A\overline{B}$ 加 A 加 1	F=A+B	$F=(A+\overline{B})$加 A	$F=(A+\overline{B})$加 A 加 1
1111	F=A	F=A	F=A 加 1	F=A	F=A 减 1	F=A

3. ALU的应用

当采用组间并行进位时，需要增加一片先行进位部件（74182）。



74182可以产生三个进位信号C_{n+x}、C_{n+y}、C_{n+z}，并且还产生大组进位产生函数G**和大组进位传递函数P**，可供组成位数更长的多级先行进位ALU时用。

$$C_{16}=G_4^*+P_4^*G_3^*+P_4^*P_3^*G_2^*+P_4^*P_3^*P_2^*G_1^*+P_4^*P_3^*P_2^*P_1^*C_0$$

$$=G_1^{**}+P_1^{**}C_0$$

大组进位产生函数G_1^{**}

大组进位传递函数P_1^{**}

74181和74182的结合可组成各种位数的ALU部件。

8片74181和2片74182构成的32位两级行波ALU。各片74181输出的组进位产生函数和组进位传递函数作为74182的输入，而74182输出的进位信号C_{n+x}、C_{n+y}、C_{n+x}作为74181的输入，74182输出的大组进位产生函数和大组进位传递函数可作为更高一级74182的输入。

感谢聆听