计算机系统的两大功能之一——运算

计算机中数值与文字信息表示

进制

- 十进制是人们最熟悉的一种进位记数制，有0-9共10个计数符号
- 而在计算机中信息都是采用二进制表示的，只有0和1两个计数符号

进制转换

- 不同进制是可以转换的
  - 二进制→十进制

$(1\quad 0\quad 1\quad 0\quad 1\ .\ 1\quad 1)_2$

$= 1\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} = (21.75)_{10}$

  - 十进制→二进制

怎么计算?

十进制→二进制的转换过程

- 整数部分
  - 十进制整数除以2后取余数，直到商为0。将余数按先右(低位)到后左(高位)排列
- 小数部分
  - 十进制小数乘以2后取整数，将整数按先左(高位)到后右(低位)排列

2	100	0
2	50	0
2	25	1
2	12	0
2	6	0
2	3	1
2	1	1
	0	

```
  0.345
×     2
-------
  0.690
×     2
-------
  1.380
×     2
-------
  0.760
×     2
-------
  1.520
×     2
-------
  1.04
```

$(100.345)_{10}=(1100100.01011)_2$

十进制→二进制的转换过程

- 其他转换方法
  - 100

$=64+32+4$

$=2^6+2^5+2^2$

$=1*2^6+1*2^5+0*2^4+0*2^3+1*2^2+0*2^1+0*2^0$

$=1100100$

十进制→二进制的转换过程

- 特殊数据
  - 127、255、65535
    - 0111 1111 、1111 1111 、 1111 1111 1111 1111
  - 254、128
    - 1111 1110 、 1000 0000

可以快速转换的进制

八进制	对应二进制
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

十六进制	对应二进制	十六进制	对应二进制
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

转换示例

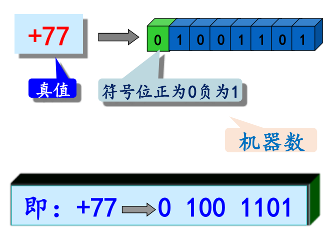
100 110 110 111. 010 100

$(4 \quad 6 \quad 6 \quad 7. \quad 2 \quad 4)_8$

0011 1011 0111. 0101

$(3 \quad B \quad 7. \quad 5)_{16}$

数值型数据在机器中的表示



有符号数的表示

- -4如何表示?

+4：**0** 000 0100

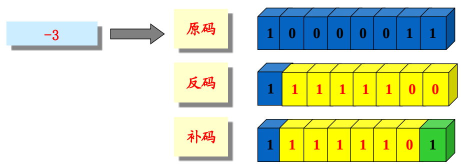
-4：**1** 000 0100

这种表示方式称为原码

原码、反码、补码

- 正数的原码、反码和补码的表示相同
- 负数反码：原码的符号位不变，数值位取反（0变1，1变0）
- 负数补码：反码加1

原码、反码、补码



原码的缺陷

+4	00000100
+ (-3)	+ 10000011
1	10000111

-7?

反码的缺陷

+4	00000100
+ (-3)	+ 11111100
1	100000000

0?

补码直接参与计算

+4	00000100
+ (-3)	+ 11111101
1	100000001

补码的由来

例1：有一只表指在9点，要拨到4点，有二种方法

逆时针拨 9 - 5 = 4

顺时针拨 9 + 7 = 12 + 4

所以：9 - 5 = 9 + （ - 5） ≡ 9 + （12 - 5）

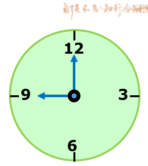
≡ 9 + 7 ≡ 12 + 4 模是12

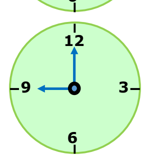
（ - 5）$_{补}$ = （12 - 5） = 7

例2：钟表的模是12，而8位二进制运算，模为256

90 - 20 = 90 + （ - 20） ≡ 90 + （256 - 20）

≡ 90 + 236 ≡ 326 ≡ 256 + 70





思考：补码的范围（单字节）

-0不存在!

00000000 ⇒ 0　　-128 ⇐ 10000000

00000001 ⇒ +1　　-127 ⇐ 10000001

……　　……

01111111 ⇒ +127　　-1 ⇐ 11111111

移码

- 移码的符号位：0负　1正

$$[X]_{补} \xleftrightarrow{\text{符号位取反}} [X]_{移}$$

- 唯一零：补码（0000 0000）、移码（1000 0000）
- 移码多用于浮点数中表示阶码

移码举例

- 真值+3
  - 原码表示：011 → 0000 0011
  - 反码表示：011 → 0000 0011
  - 补码表示：011 → 0000 0011
  - 移码表示：111 → 1000 0011

正数扩展：
- 在符号位和数值位之间补0

- 真值-3
  - 原码表示：111 → 1000 0011
  - 反码表示：100 → 1111 1100
  - 补码表示：101 → 1111 1101
  - 移码表示：001 → 0111 1101

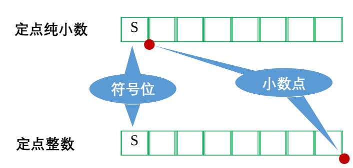
负数扩展：
- 原码在符号位和数值位之间补0
- 反码、补码和移码都在符号位和数值位之间补1

练习

- 85和-85的8位原码、反码、补码和移码表示。

定点数和浮点数

- 定点数：



定点数和浮点数

阶符	阶码	数符	尾数

例如：字长为 16 位，其中阶码为 6 位（含一位阶符），尾数为 10 位（含一位数符），阶码用移码，尾数用补码的形式。
尾数的位数决定数的精度
阶码的位数决定数的范围

$$-0.000110101(B) = -0.110101000\times2^{-11}$$
$$= 1.001011000\times2^{011101}$$

0	11101	1	001011000

规格化的形式：尾数的绝对值大于等于0.1并且小于1，从而唯一地规定了小数点的位置。

英文字符表示——ASCII码

- American Standard Code for Information Interchange，128个常用字符，用7位二进制编码，从0到127。其中控制字符：0～32，127；剩余是普通字符。

$d_3d_2d_1d_0$位	$0d_6d_5d_4$位 000	001	010	011	100	101	110	111
0000	NUL	DEL	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	SO	RS	.	>	N	↑	n	~
1111	SI	HS	/	?	O	←	o	DEL

英文字符表示——ASCII码

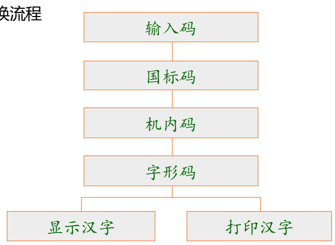
ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
0	NUL	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	.	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	.	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	/	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	DEL

汉字编码

- 输入码
- 国标码
- 内码
- 字形码

汉字编码

编码交换流程



汉字编码

- 输入码
  - (1) 数字编码，例如国标码、区位码；
  - (2) 字音编码，例如搜狗拼音、微软拼音；
  - (3) 字型编码，例如五笔输入法；
  - (4) 形音编码，结合字音编码和字型编码的优点。

GB 2312-80

- 双字节表示，分为区码和位码
- 包括202个一般符号，60个序号，22个数字，52个拉丁字母，169个日文假名，48个希腊字母，56个俄文字母，26个汉语拼音符号；37个汉语注音字母，6763个汉字
- 6763个汉字分为两级。第一级汉字3755个按汉语拼音顺序排列；第二级汉字3008个按笔画顺序排列

GB 2312-80（一般符号）

1区	00	01	02	03	04	05	06	07	08	09
00		(SP)	、	。	•	ˉ	ˇ	¨	〃	々
10	—	～	‖	…	‘	’	“	”	〔	〕
20	〈	〉	《	》	「	」	『	』	〖	〗
30	【	】	±	×	÷	∶	∧	∨	∑	∏
40	∪	∩	∈	∷	√	⊥	∥	∠	⌒	⊙
50	∫	∮	≡	≌	≈	∽	∝	≠	≮	≯
60	≤	≥	∞	∵	∴	♂	♀	°	′	″
70	℃	$	¤	￠	￡	‰	§	№	☆	★
80	○	●	◎	◇	◆	□	■	△	▲	※
90	→	←	↑	↓	〓					
2区	00	01	02	03	04	05	06	07	08	09
00	i	ii	iii	iv	v	vi	vii	viii	ix	x
10								1.	2.	3.
20	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.
30	14.	15.	16.	17.	18.	19.	20.	(1)	(2)	(3)
40	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
50	(14)	(15)	(16)	(17)	(18)	(19)	(20)	①	②	③
60	④	⑤	⑥	⑦	⑧	⑨	⑩	€		㈠
70	㈡	㈢	㈣	㈤	㈥	㈦	㈧	㈨	㈩	
80		Ⅰ	Ⅱ	Ⅲ	Ⅳ	Ⅴ	Ⅵ	Ⅶ	Ⅷ	Ⅸ
90	Ⅹ	Ⅺ	Ⅻ							

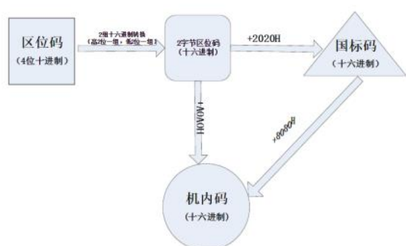
GB 2312-80（汉字）

16 区	00	01	02	03	04	05	06	07	08	09
00		啊	阿	埃	挨	哎	唉	哀	皑	癌
10	蔼	矮	艾	碍	爱	隘	鞍	氨	安	俺
20	按	暗	岸	胺	案	肮	昂	盎	凹	敖
30	熬	翱	袄	傲	奥	懊	澳	芭	捌	扒
40	叭	吧	笆	八	疤	巴	拔	跋	靶	把
50	耙	坝	霸	罢	爸	白	柏	百	摆	佰
60	败	拜	稗	斑	班	搬	扳	般	颁	板
70	版	扮	拌	伴	瓣	半	办	绊	邦	帮
80	梆	榜	膀	绑	棒	磅	蚌	镑	傍	谤
90	苞	胞	包	褒	剥					
56 区	00	01	02	03	04	05	06	07	08	09
00		亍	丌	兀	丐	廿	卅	丕	亘	丞
10	鬲	孬	噩	丨	禺	丿	匕	乇	夭	爻
20	卮	氐	囟	胤	馗	毓	睾	鼗	丶	亟
30	鼐	乜	乩	亓	芈	孛	啬	嘏	仄	厍
40	厝	厣	厥	厮	靥	赝	匚	叵	匦	匮
50	匾	赜	卦	卣	刂	刈	刎	刭	刳	刿
60	剀	剌	剞	剡	剜	蒯	剽	劂	劁	劐
70	劓	冂	罔	亻	仃	仉	仂	仨	仡	仫
80	仞	伛	仳	伢	佤	仵	伥	伧	伉	伫
90	佞	佧	攸	佚	佝					

区位码、国标码与机内码

- 以汉字“白”为例：
  - 区位码：1655
  - 国标码：1037H+2020H = 3057H
  - 机内码：B0D7H
    （=国标码+8080H）

汉字区位码、国标码（交换码）及机内码转换关系图



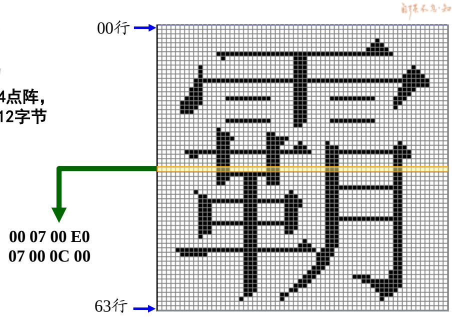
汉字编码

- 字形码
  - 字形码是指字形的点阵信息的数字代码。存放在汉字库中。字型码有显示字形码和打印字形码两种。根据输出的去向将汉字输出在显示器上或打印机上。

汉字编码

- 字形码

  64×64点阵，
  共占512字节

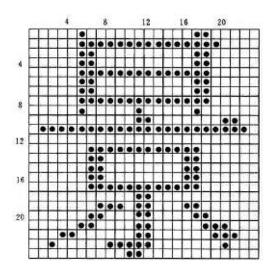


练习

多少×多少点阵？

占用几个字？

每个字节的16进制表示是多少？



Unicode编码

- 使用标准方案来展示世界上所有语言中的所有字符。
- Unicode编码的实现方式：
  - UTF-8：一种变长的编码方案，使用 1~6 个字节来存储；
  - UTF-32：一种固定长度的编码方案，不管字符编号大小，始终使用 4 个字节来存储；
  - UTF-16：介于 UTF-8 和 UTF-32 之间，使用 2 个或者 4 个字节来存储，长度既固定又可变。

Unicode编码

- UTF-8编码规则：
  - 0xxxxxxx：单字节编码形式，和ASCII编码完全一样，因此UTF-8是兼容ASCII的；
  - 110xxxxx 10xxxxxx：双字节编码形式；
  - 1110xxxx 10xxxxxx 10xxxxxx：三字节编码形式；
  - 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx：四字节编码形式。

二进制算术运算

二进制算术运算基础

加法运算法则：

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10

例：求 $(10011.01)_2 + (100011.11)_2 = ?$

```
        1 0 0 1 1 . 0 1
+)    1 0 0 0 1 1 . 1 1
      1 1 0 1 1 1 . 0 0
```

练习：求 $(1011011)_2 + (1010.11)_2 = ?$

二进制算术运算基础

减法运算法则：

0 − 0 = 0

1 − 0 = 1

1 − 1 = 0

10 − 1 = 1

(0 − 1)

例：求 $(10110.01)_2 - (1100.10)_2 = ?$

```
      1 0 1 1 0 . 0 1
-)      1 1 0 0 . 1 0
        1 0 0 1 . 1 1
```

练习：求 $(1010110)_2 - (1101.11)_2 = ?$

二进制算术运算基础

乘法运算法则：

0×0 = 0

1×0 = 0

0×1 = 0

1×1 = 1

例：求 $(1101.01)_2 \times (110.11)_2 = ?$

```
              1 1 0 1 . 0 1
×)              1 1 0 . 1 1
              1 1 0 1   0 1
            1 1 0 1 0   1
          0 0 0 0 0 0
        1 1 0 1 0 1
      1 1 0 1 0 1
    1 0 1 1 0 0 1 . 0 1 1 1
```

二进制算术运算基础

除法运算法则：

0÷0＝0

1 ÷0＝（无意义）

0 ÷1 ＝0

1 ÷1＝1

例：求（100011）$_2$÷（101）$_2$ ＝ ？

```
        111
101 )100011
     101
      0111
       101
       0101
        101
          0
```


进位与借位

- 加法运算时，最高位向上产生进位
- 减法运算时，最高位不足，需借位

```
   10111101        10111101
+  11011111     -  11011111
  110011100       111011110
```


溢出

- 运算结果超出运算器所能表示的范围

```
   100          01100100
+  100   ➡   +  01100100
   200          11001000
                  -56 ✗
```


有进位是否就会溢出?

```
   -67          10111101
+  -33   ➡   +  11011111
  -100         110011100
                  -100
```

有进位，未溢出！

无进位就一定不溢出吗?

```
   +66          01000010
+  +99   ➡   +  01100011
  +165          10100101
```

无进位，但溢出！

如何准确判断溢出?

- 无符号数运算
  - 相加有可能溢出，相加前在前加0作为符号位，如果结果的符号为1，则溢出
  - 小减大肯定溢出
  - 大减小不会溢出
- 有符号数运算
  - 同号相减或异号相加肯定不会溢出
  - 同号相加或异号相减则可能溢出，如果同号相加时结果符号与加数符号相反，或者异号相减时结果符号与减数符号相同，则结果溢出

CPU怎么判断溢出?

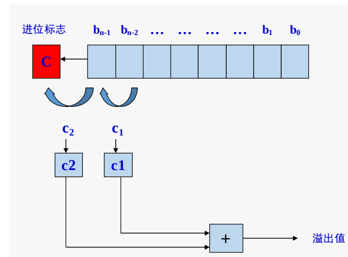
两个有符号二进制数相加或相减时，

若Cs⊕Cp＝1，则结果溢出。

Cs为符号位的进(借)位

Cp为最高数值位的进(借)位

CPU怎么判断溢出?



二进制逻辑运算

计算机的逻辑运算

- 基本逻辑关系
  - 逻辑与(AND)
  - 逻辑或(OR)
  - 逻辑非(NOT)
  - 逻辑异或（XOR)

逻辑运算基础

与运算符: • × ∧ ∩ And

运算法则:

$0 \wedge 0 = 0$

$0 \wedge 1 = 0$

$1 \wedge 0 = 0$

$1 \wedge 1 = 1$

例:

10101111•10011101 ＝ ？

```
      1 0 1 0 1 1 1 1
∧)    1 0 0 1 1 1 0 1
---------------------
      1 0 0 0 1 1 0 1
```

练习:

10111001•11110011 ＝ ？

逻辑运算基础

或运算符: ＋ ∨ ∪ Or

运算法则:

$0 \vee 0 = 0$

$0 \vee 1 = 1$

$1 \vee 0 = 1$

$1 \vee 1 = 1$

例:

10101010＋01100110＝ ？

```
      1 0 1 0 1 0 1 0
∨)    0 1 1 0 0 1 1 0
---------------------
      1 1 1 0 1 1 1 0
```

练习: 10100001+10011011 ＝ ？

逻辑运算基础

非运算符：在变量上加“—”

运算法则：

$\overline{1} = 0$

$\overline{0} = 1$

例：$\overline{10101100} = 01010011$

逻辑运算基础

异或运算符　+

运算法则：

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$

例：10101010 ＋ 00001111= ？

$$\begin{array}{r} 10101010 \\ +\ 00001111 \\ \hline 10100101 \end{array}$$

运算的电路实现

加法器运算电路

加法运算法则

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

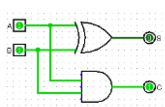
$1 + 1 = 10$

输入		输出	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

加法器运算电路

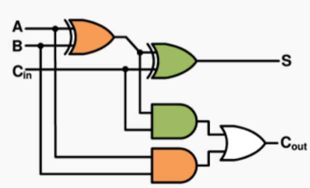
- 可以用异或门及与门构成加法电路（半加器）
- 所谓半加指只求本位的和，暂不管低位的进位

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



全加器（Full Adder）

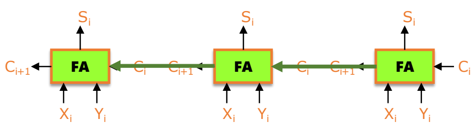
- 全加器由两个半加器构成
  - 输入端口A、B、C_{in}（进位输入）
  - 输出端口S（和）、C_{out}（进位输出）



A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

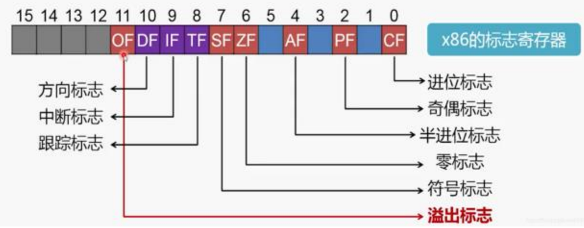
n位加法器

- n位加法器包含n个全加器
- 将n个一位全加器串联
- 低位进位输出连接到高位进位输入



溢出标志OF（Overflow Flag）

- 如果把操作数看做有符号数，运算结果是否发生溢出
- 若发生溢出，则自动设置OF=1；否则，OF=0



串行进位加法器

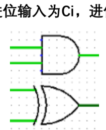
- 串行进位加法器是串行执行的，其高位的运算要依赖低位的进位，所以当输入数据的位数较多时，会形成很大的延迟并可能成为芯片的瓶颈。

并行进位加法器

- 也叫先行进位加法器
- 设二进制加法器第i位为Ai，Bi，输出为Si，进位输入为Ci，进位输出为C(i+1)，按运算算法则则有：

并行进位加法器

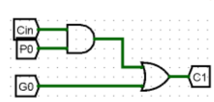
- 设二进制加法器第i位为Ai，Bi，输出为Si，进位输入为Ci，进位输出为C(i+1)，按运算算法则则有：
  - $S_i = A_i + B_i + C_i$
  - $C_{i+1} = A_i * B_i + (A_i + B_i) * C_i$
    - ◆令：$G_i = A_i * B_i$ $P_i = A_i + B_i$
    - 则有：
    - ◆$C_{i+1} = G_i + P_i * C_i$



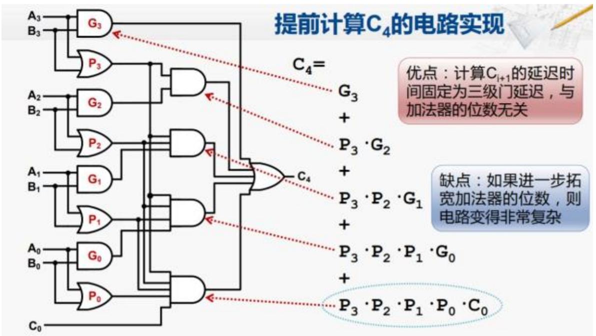
$G_i = A_i * B_i$

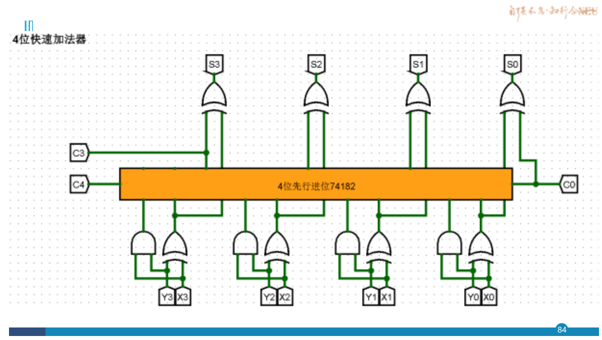
$P_i = A_i + B_i$

并行进位加法器



- 4位并行进位加法器设计
  - $C_0=C_{in}$;
  - $C_1=G_0+P_0*C_0$;
  - $C_2=G_1+P_1*C_1=G_1+P_1*(G_0+P_0*C_0) = G_1+P_1*G_0+P_1*P_0*C_0$;
  - $C_3=G_2+P_2*C_2=G_2+P_2*G_1+P_2*P_1*G_0+P_2*P_1*P_0*C_0$;
  - $C_4=G_3+P_3*C_3=G_3+P_3*G_2+P_3*P_2*G_1+P_3*P_2*P_1*G_0+P_3*P_2*P_1*P_0*C_0$;
  - $C_{out}=C_4$;





其他算术运算

- 减法→加法
- 乘法→多次加法
- 除法→多次减法→多次加法