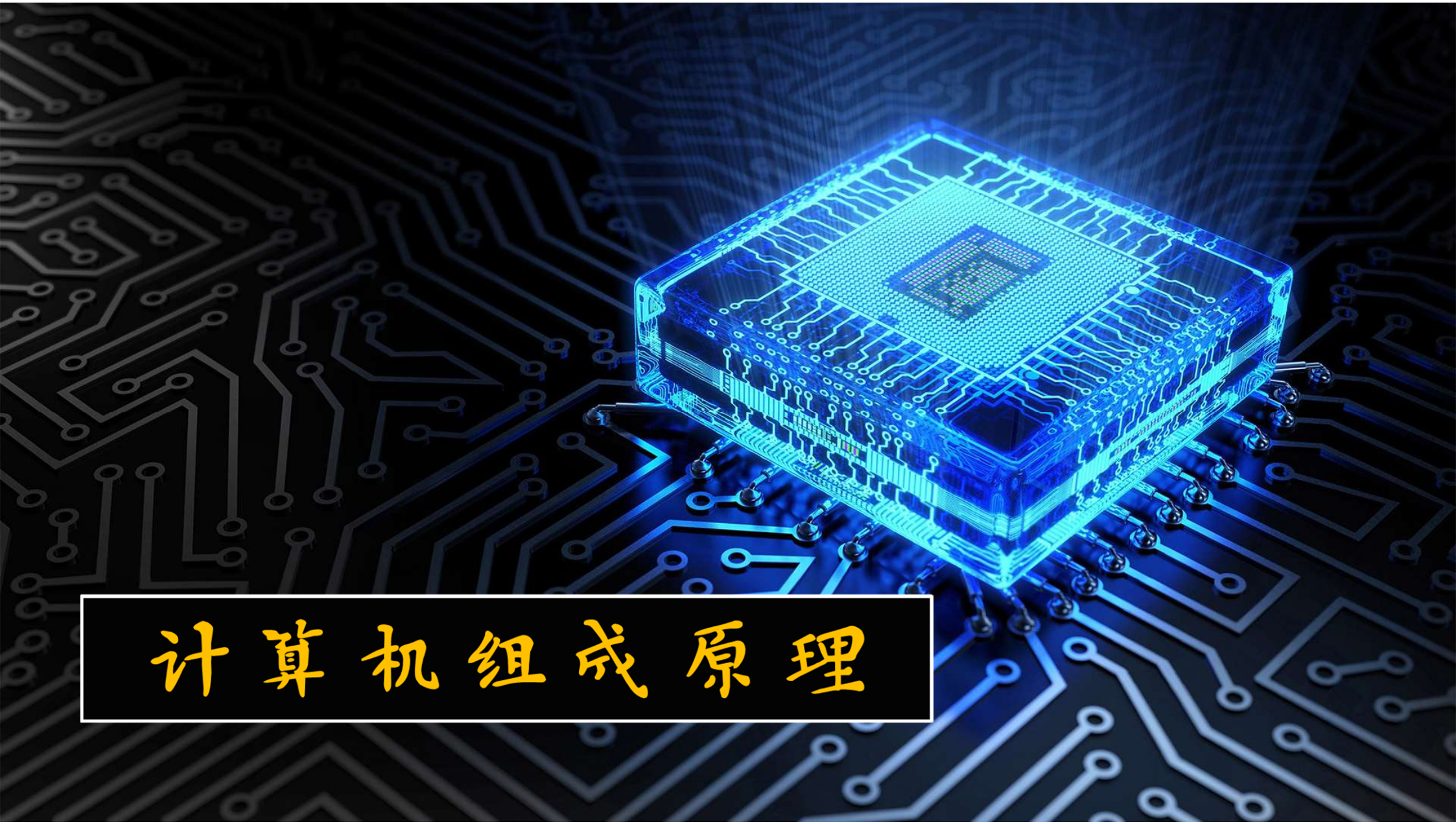
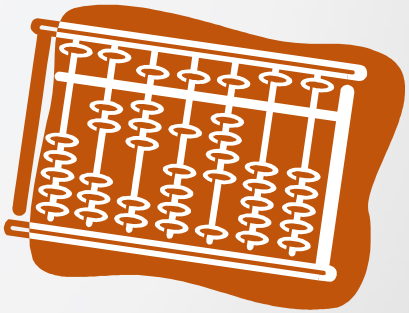

计算机组成原理

计算机组成原理

三、运算器



本章主要内容

- **定点补码加/减法运算**
- 定点乘法运算
- 定点除法运算
- 浮点运算
- 运算器组织

C语言中的运算

- 逻辑运算
  - 位运算　　“&”　“|”　“~”　“^”
  - 逻辑运算　　“&&”　“||”　“!”
  - 移位运算　　“<< ”　“>> ”
  - 位扩展　位截断　用于类型转换
- 算术运算
  - 无符号/符号整数的加减乘除运算
  - 变量与常数间的乘除运算
  - 浮点数加减乘除运算
- 如何用电路实现？

3.1 定点补码加/减法运算

- 运算方法及实现
  - 运算公式
  - 溢出检测
  - 逻辑实现
- 快速加法器

补码加减法的实现

- **补码加法**：$[X+Y]_{补} = [X]_{补} + [Y]_{补}$
  - 和的补码 = 补码的和
- **补码减法**：$[X-Y]_{补} = [X]_{补} - [Y]_{补} = [X]_{补} + [-Y]_{补}$
  - 差的补码 = 补码的差
  - 减法变加法，关键是求$[-Y]_{补}$
- **求补公式**： $[-Y]_{补} = -[Y]_{补}$
  - $[-Y]_{补}$=对 $[Y]_{补}$逐位取反, 再在最低位加 1

补码加法公式证明

- $[X+Y]_{补}=[X]_{补}+[Y]_{补}$

1. x>0 y>0 (无需证明)
2. x>0 y<0
3. x<0 y>0 (2/3证明相同)
4. x<0 y<0

- 只需证明2/4两种情况即可

补码加法公式证明　x>0　y<0

$[x]_{补}=x$　$[y]_{补}=2+y$（设x，y为定点小数，1>x>0，-1≤y<0）

$[x]_{补}+[y]_{补}=x+2+y=2+(x+y)$

当 $x+y<0$ 时

$2+(x+y)=[x+y]_{补} \pmod 2$

当 $x+y>0$ 时

$2+(x+y)>2$　模舍去

$[x]_{补}+[y]_{补}=2+(x+y)=x+y \pmod 2$

$=[x+y]_{补} \pmod 2$

补码加法公式证明　x<0 y<0

$$[x]_{补}=2+x \quad [y]_{补}=2+y$$

$$[x]_{补}+[y]_{补}=(2+x)+(2+y)=(2+(2+x+y)) \mod 2$$

$$-2 \le x+y < 0$$

故 $0 \le 2+x+y < 2$

故 $(2+(2+x+y)) \mod 2 = (2+x+y)$

$$=[x+y]_{补} \mod 2$$

补码减法公式证明

$[X-Y]_{补} = [X]_{补} - [Y]_{补}$???

$[X-Y]_{补} = [X]_{补} + [-Y]_{补}$ (加法公式)

$[-Y]_{补} = -[Y]_{补}$?

$[-Y]_{补} + [Y]_{补} = [Y + (-Y)]_{补} = [0]_{补} = 0$

故$[-Y]_{补} = -[Y]_{补}$ 成立 $[X-Y]_{补} = [X]_{补} + [-Y]_{补} = [X]_{补} - [Y]_{补}$

$[-Y]_{补} = -[-Y]_{补}$

补码加法的溢出讨论

正常结果

```
  0 .1 0 1 0 1
+ 0 .0 1 0 0 0
-------------
  0 .1 1 1 0 1
```

```
  1 .1 0 1 0 1
+ 1 .1 1 0 0 0
-------------
1 1 .0 1 1 0 1
```

模数舍去

正常结果

正正得负

正溢出

```
  0 .1 0 1 0 1
+ 0 .1 1 0 0 0
-------------
  1 .0 1 1 0 1
```

```
  1 .0 0 1 0 1
+ 1 .1 1 0 0 0
-------------
1 0 .1 1 1 0 1
```

负负得正

负溢出

计算机如何识别运算结果是否溢出?

单符号数溢出检测1

- 溢出逻辑： **正正得负 负负得正**

- 设两数符号位为 $f_0 f_1$ ，和数符号位 f_s

- 溢出检测信号***Overflow (OF)***

$$Overflow = \bar{f}_0 \bar{f}_1 f_s + f_0 f_1 \bar{f}_s$$

单符号溢出检测方法2

$C_f=0, C_n=0$

```
  0 .1 0 1 0 1
+ 0 .0 1 0 0 0
-------------
  0 .1 1 1 0 1
```

正常结果

$C_f=1, C_n=1$

```
  1 .1 0 1 0 1
+ 1 .1 1 0 0 0
-------------
1 1 .0 1 1 0 1
```

正常结果

$C_f=0, C_n=1$

```
  0 .1 0 1 0 1
+ 0 .1 1 0 0 0
-------------
  1 .0 1 1 0 1
```

正溢出

$C_f=1, C_n=0$

```
  1 .0 0 1 0 1
+ 1 .1 1 0 0 0
-------------
1 0 .1 1 1 0 1
```

负溢出

符号位进位位C_f，最高位进位位C_n

$$Overflow = C_f \oplus C_n$$

双符号溢出检测方法

```
    0 0 .1 0 1 0 1
+   0 0 .0 1 0 0 0
-------------------
    0 0 .1 1 1 0 1
```

未溢出

```
    1 1 .1 0 1 0 1
+   1 1 .1 1 0 0 0
-------------------
  1 1 1 .0 1 1 0 1
```

未溢出

```
    0 0 .1 0 1 0 1
+   0 0 .1 1 0 0 0
-------------------
    0 1 .0 1 1 0 1
```

正溢出

<

```
    1 1 .0 0 1 0 1
+   1 1 .1 1 0 0 0
-------------------
  1 1 0 .1 1 1 0 1
```

负溢出

双符号位最高位永远是正确符号位

$$Overflow = f_1 \oplus f_2$$

有符号数溢出判断方法总结

- 判断方法：
- 根据加数和结果的符号位判断

$$Overflow = \bar{f}_0 \bar{f}_1 f_s + f_0 f_1 \bar{f}_s$$

- 根据次高位向最高位的进位/借位，最高位向进位位的进位/借位是否一致判断

$$Overflow = C_f \oplus C_n$$

- 根据结果双符号位是否一致判断

$$Overflow = f_1 \oplus f_2$$

- 哪种方案硬件实现更好？

无符号数溢出检测 unsigned char

3+8=11

```
  0 0 0 0 0 1 0 1
+ 0 0 0 0 1 0 0 0
-----------------
  0 0 0 0 1 1 0 1
```

255+254=253

```
  1 1 1 1 1 1 1 1
+ 1 1 1 1 1 1 1 0
-----------------
1 1 1 1 1 1 1 0 1
```

加法溢出

128-1=127

```
  1 0 0 0 0 0 0 0
+ 1 1 1 1 1 1 1 1
-----------------
1 0 1 1 1 1 1 1 1
```

1-2=255

```
  0 0 0 0 0 0 0 1
+ 1 1 1 1 1 1 1 0
-----------------
  1 1 1 1 1 1 1 1
```

减法溢出

加法变大，减法变小

$$UOF = Sub \oplus C_{out}$$

有符号数溢出的软件检测方法

```
int tadd_ok(int x,int y) {
    int sum=x+y;
    int neg_over=x<0&&y<0&&sum>=0;
    int pos_over=x>=0&&y>=0&&sum<0;
    return !neg_over&&!pos_over;
}
```

- 软/硬件检测具有功能上的等效性和性能上的差异性！

二进制加法运算的电路实现

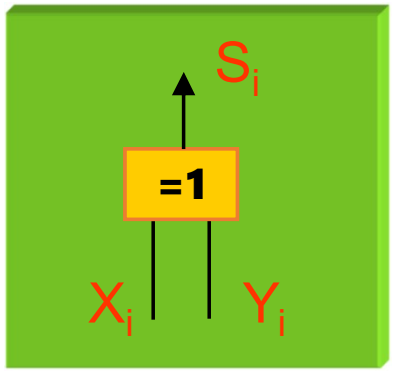
- 相同权值的各位逐位相加，进位从低位向高位传递
- 首先要考虑**一位加**法，然后考虑**进位链**

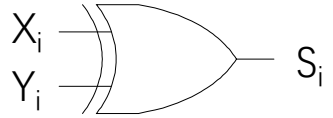
$$
\begin{array}{cccccc}
 & X_{n-1} & \cdots\cdots & X_2 & X_1 & X_0 \\
+ & Y_{n-1} & \cdots\cdots & Y_2 & Y_1 & Y_0 \\
\hline
 & ?_{n-1} & \cdots\cdots & ?_2 & ?_1 & ?_0
\end{array}
$$

一位加法逻辑电路实现

- **0 + 1 = 1　1 + 0 = 1**
- **1 + 1 = 0　0 + 0 = 0**
- 一个异或门即可实现两个1bit数相加
- 算术运算变成逻辑电路

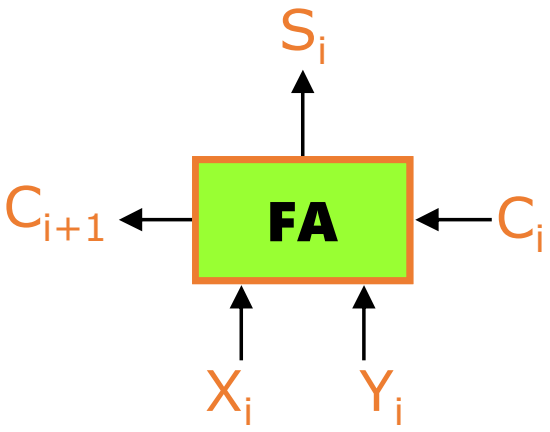
$$S_i = X_i \oplus Y_i$$





半加器HA

全加器



一位全加器

加数X_i	加数Yi	低位进位C_i	和数S_i	进位C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

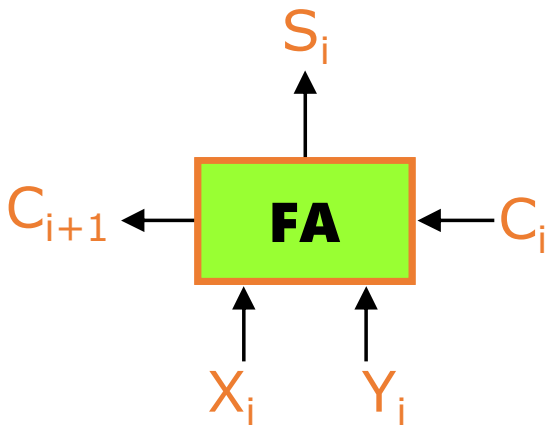
$$S_i = X_i \oplus Y_i \oplus C_i$$

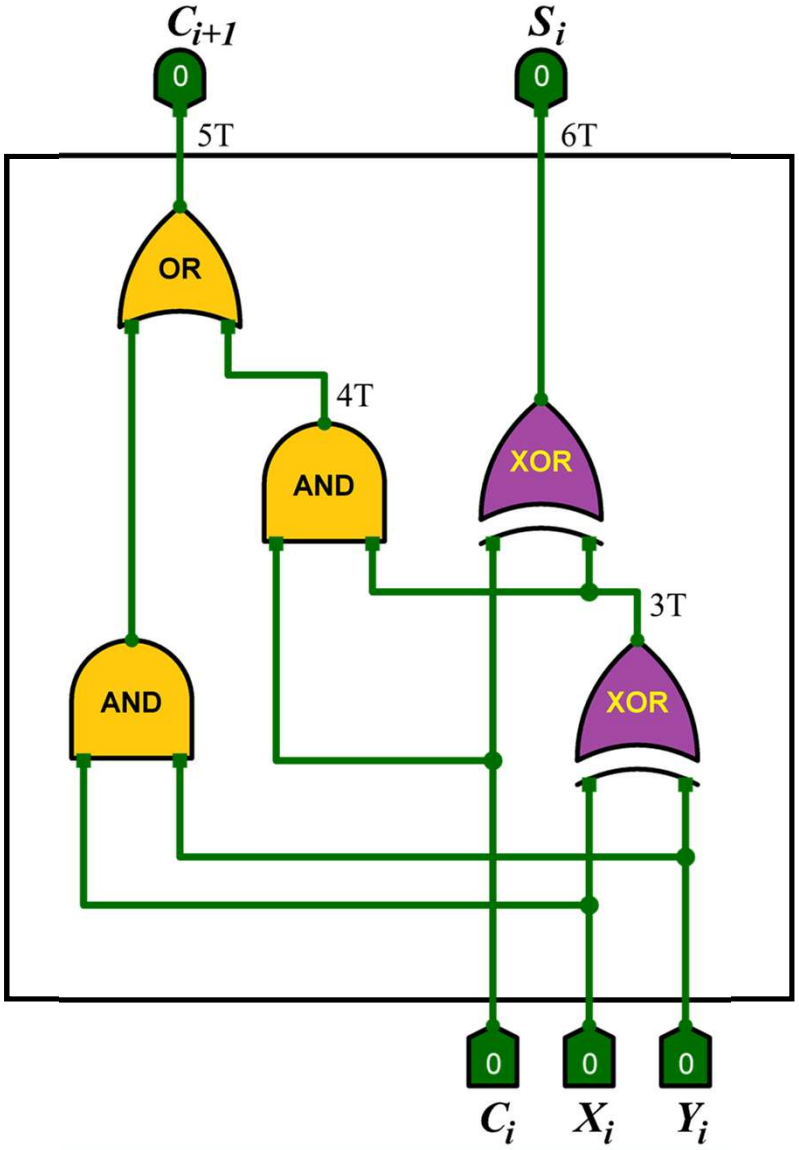
$$C_{i+1} = X_i Y_i + (X_i \oplus Y_i) C_i$$

全加器逻辑实现之一

$$S_i = X_i \oplus Y_i \oplus C_i$$

$$C_{i+1} = X_i Y_i + (X_i \oplus Y_i) C_i$$

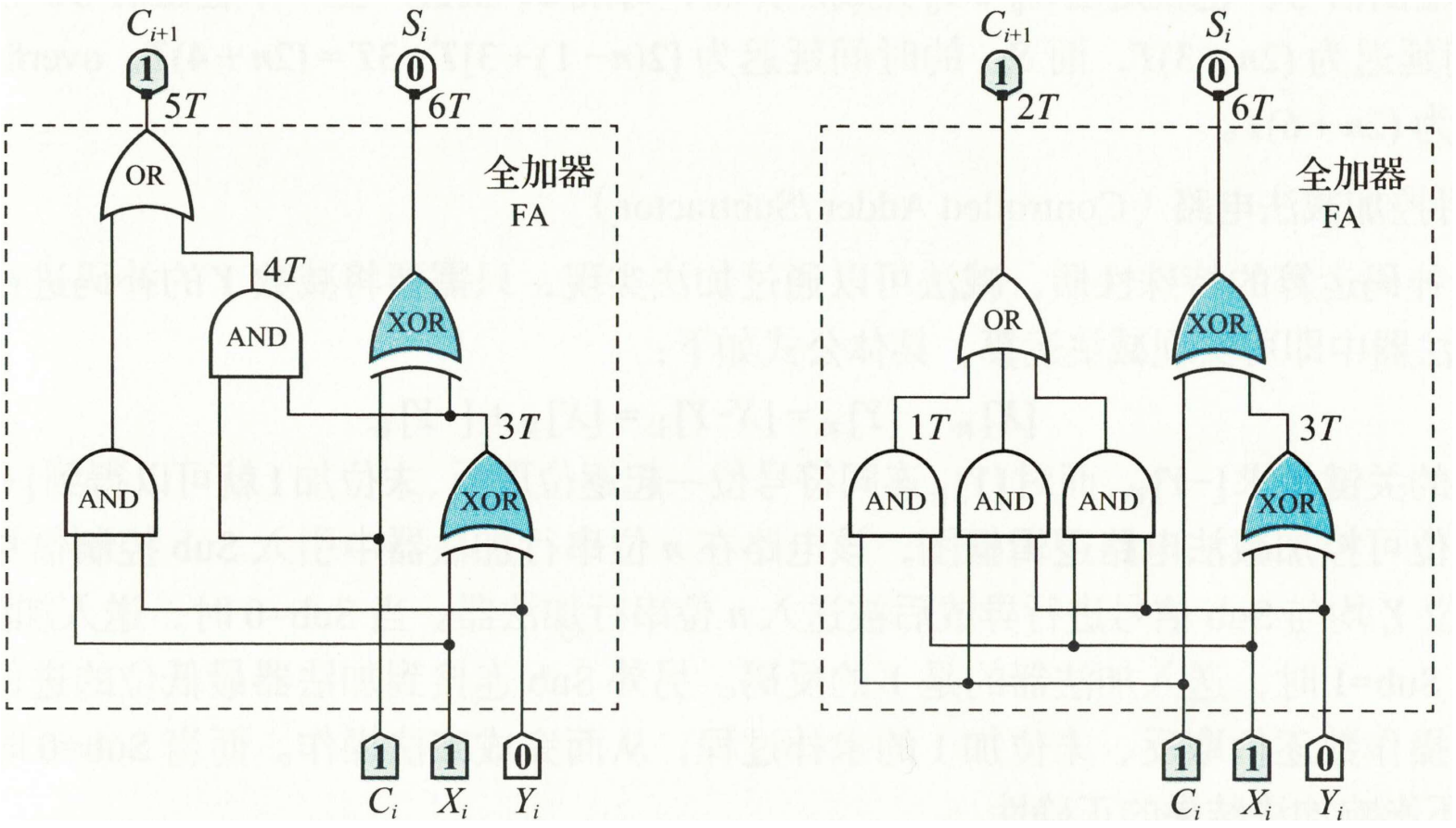




和：6级门电路延迟 6T

进位：5级门电路延迟

全加器逻辑实现之二

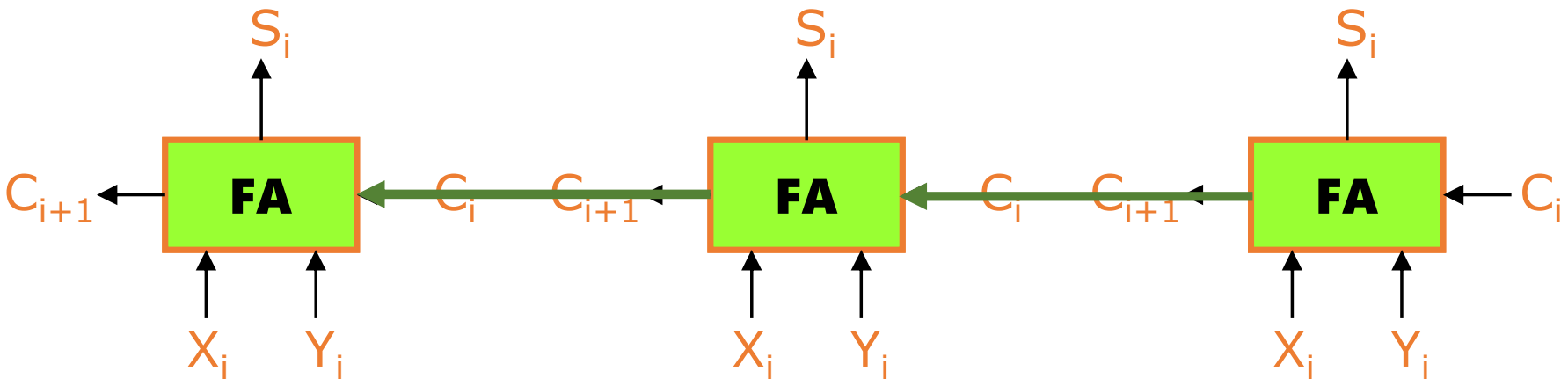


$$C_{i+1} = X_i Y_i + (X_i \oplus Y_i) C_i$$

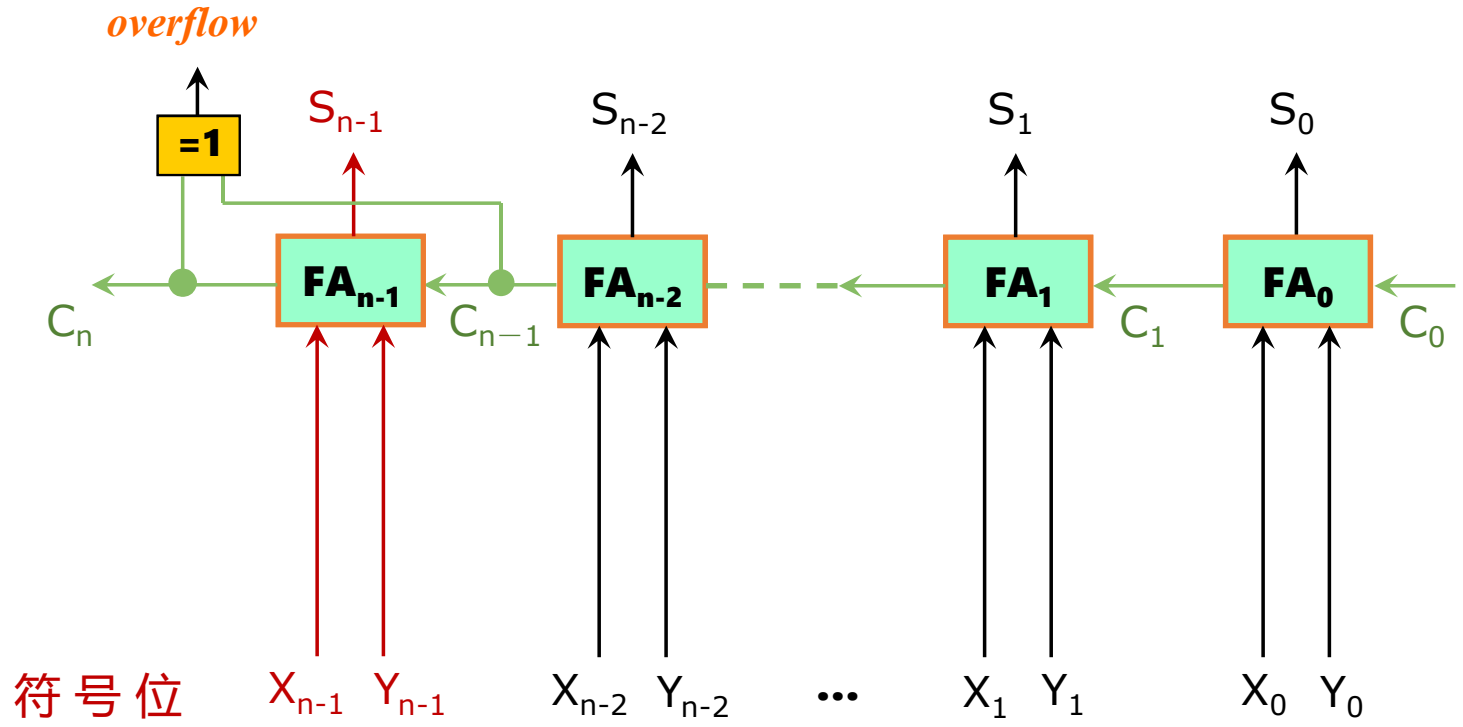
$$C_{i+1} = X_i Y_i + X_i C_i + Y_i C_i$$

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



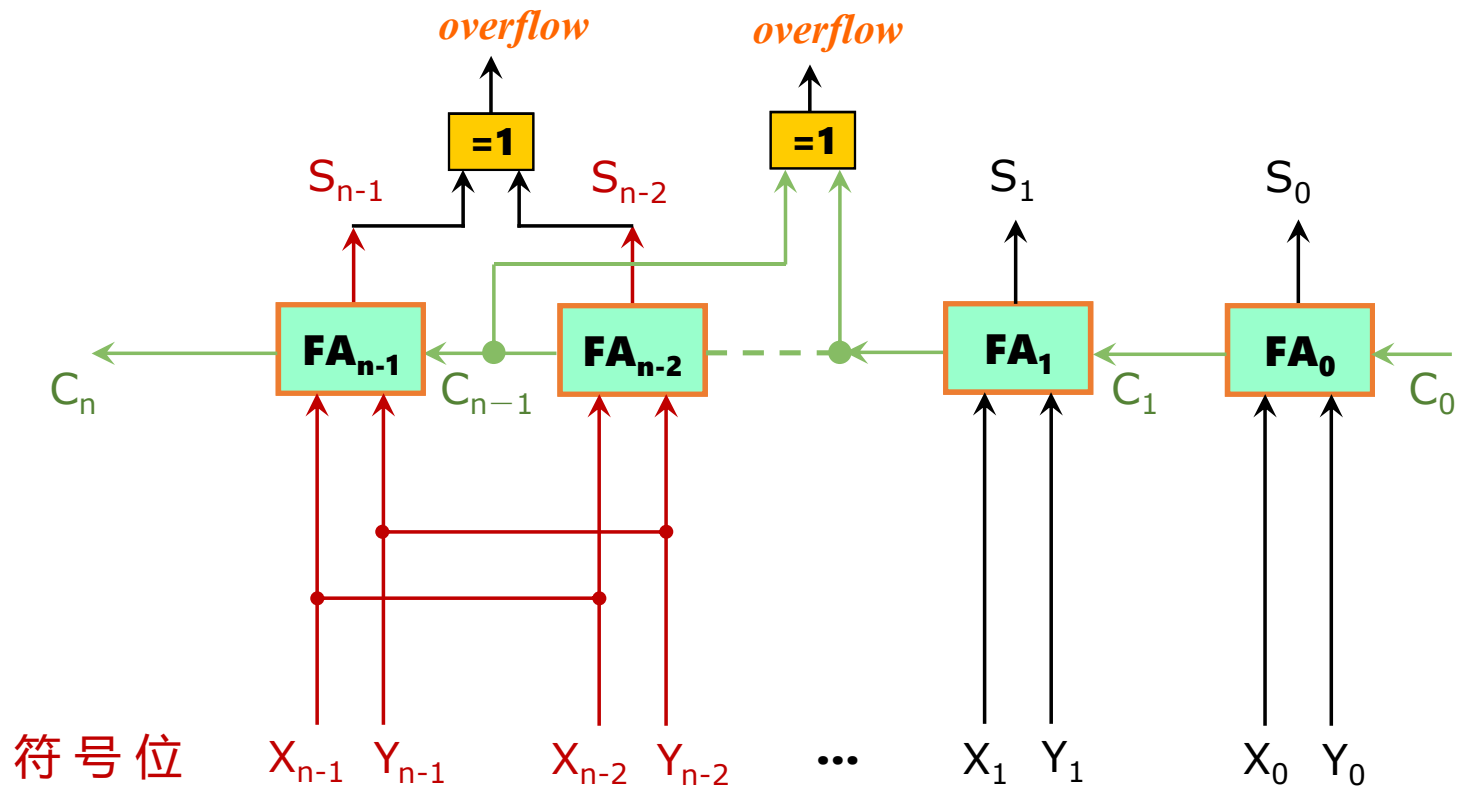
单符号位补码串行进位加法器电路(ripple carry adder)



无符号，有符号数加法器有无区别？

运算规则相同，溢出判断不同

双符号位补码加法器电路



资源开销增加，延时增加，故硬件上不会采用双符号补码运算电路

补码减法电路实现

补码减法可以变加法

$$[X]_{补} - [Y]_{补} = [X]_{补} + [-Y]_{补}$$

关键是求$[-Y]_{补}$

方法：将$Y_{补}$连同符号位一起逐位取反末位加一

$$[-Y]_{补} = [\ -[Y]_{补}\ ]_{补}$$

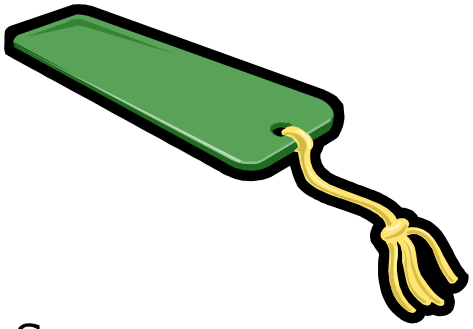
加法器的改造---对Y的输入进行选择

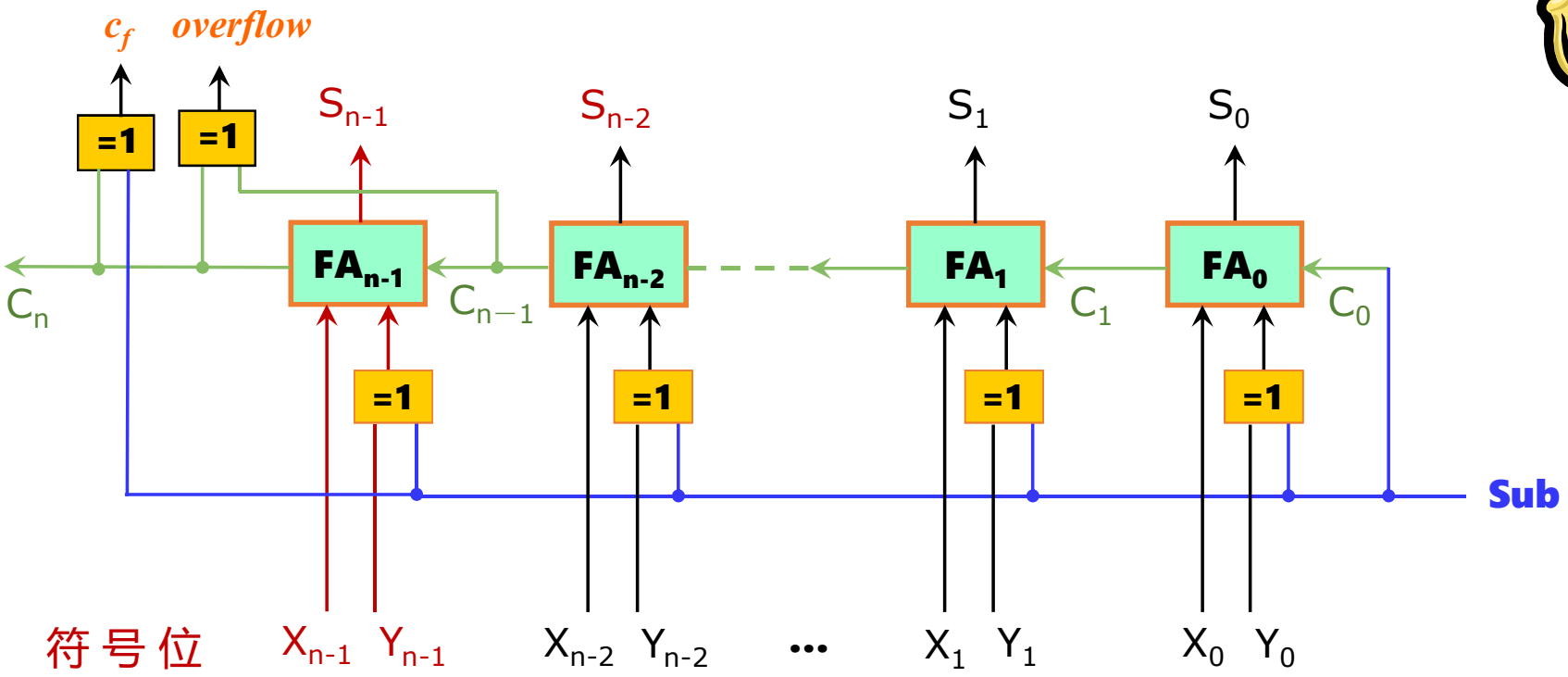
- 引入运算控制位 **Sub**
  - Sub=0 时作加法，送入加法器的是$Y_{补}$
  - Sub=1 时作减法，送入加法器的是$[-Y]_{补}$
    - 对 $Y_{补}$ **逐位取反**，末位加一

$$Input = Y_i \oplus Sub$$
$$C_0 = Sub$$

Y_i	Sub	Input
Y_i	0	Y_i
Y_i	1	$\overline{Y_i}$

可控加减法电路



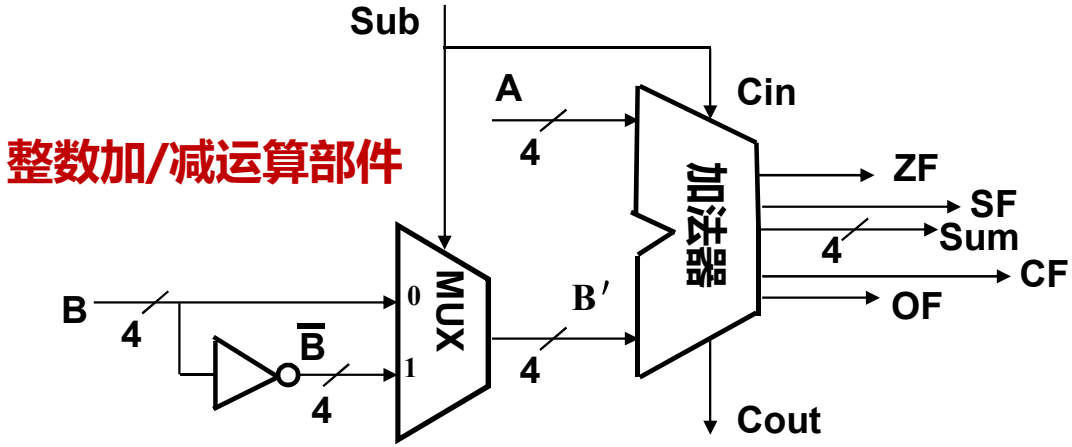


上述电路可实现有符号/无符号的加法/减法运算

控制信号Sub如何产生？

加法器的必要功能---产生运算标志位

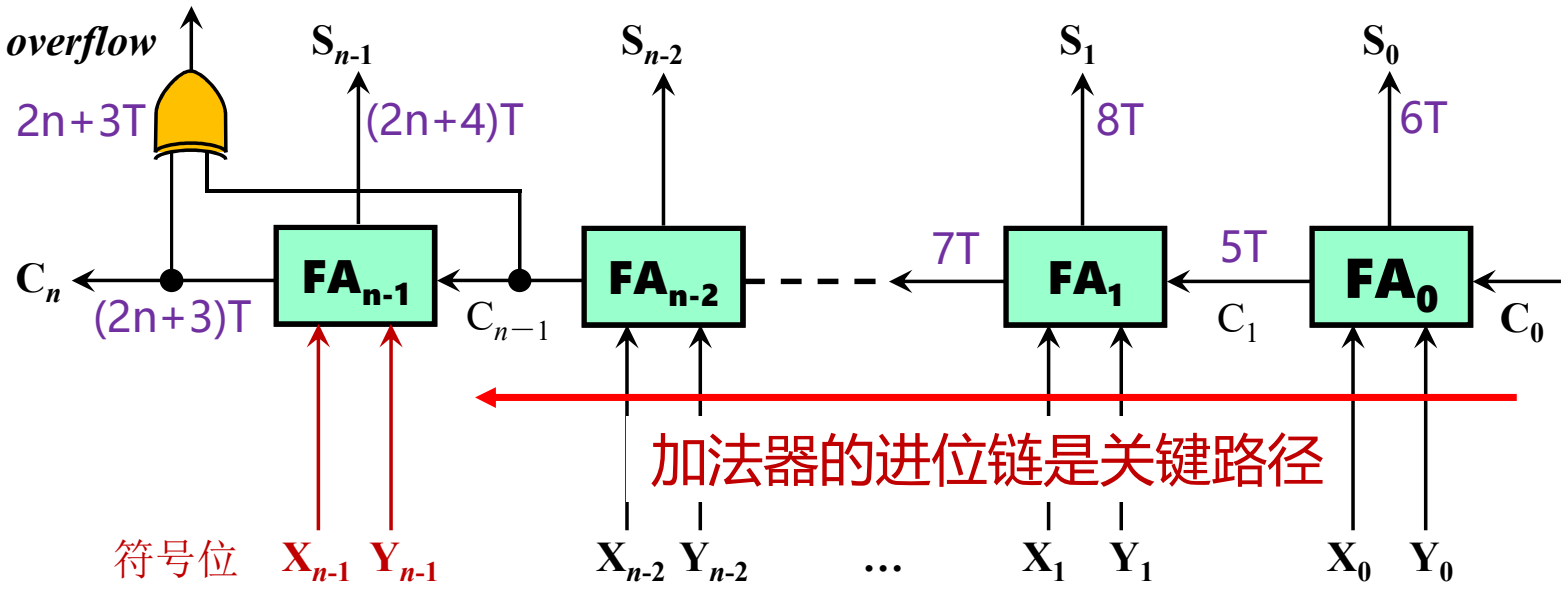
- 存放运算标志的寄存器称为程序/状态字或标志寄存器
  - 每个标志对应标志寄存器中的一个标志位。
  - IA-32中的EFLAGS寄存器　(MIPS 无标志寄存器)
    - ZF（结果为零）　SF（结果为负数）
    - CF（进位/借位）　OF（有符号溢出）　$OF=C_{n-1}\oplus C_{out}$

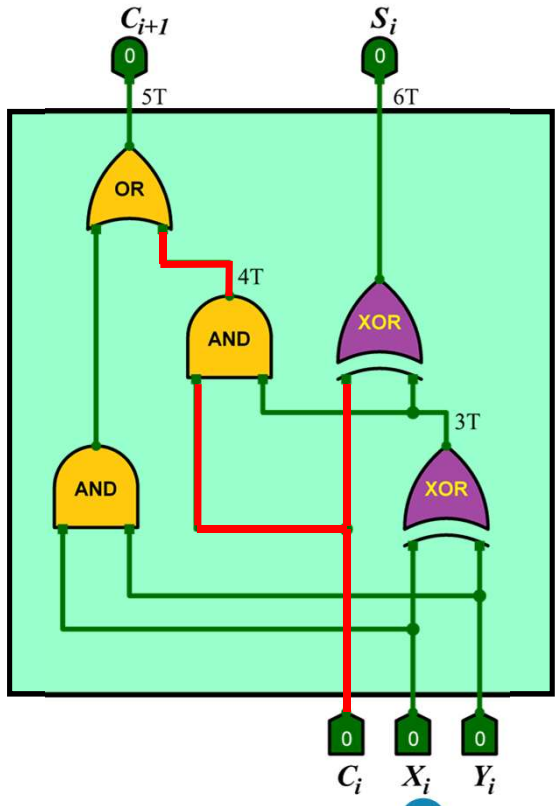


ZF=S(0) | S(1) | S(2) | S(3)
SF=S(3)

加法器的输入输出位数相同

多位加法器之串行加法器计算延时





n个全加器延迟？　关键路径在哪里？

快速加法器

- 如何缩短加法器的关键路径？
- 思路之一：能否提前产生各位的进位输入
- 先行进位加法器

进位依赖关系分析

$$S_i = X_i \oplus Y_i \oplus C_i$$

$$C_{i+1} = X_i Y_i + (X_i \oplus Y_i) C_i$$

$G_i = X_i Y_i$　　进位生成函数　Generate

$P_i = X_i \oplus Y_i$　　进位传递函数　Propagate

$$C_{i+1} = G_i + P_i C_i$$

A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

如何打破进位依赖？　进位旁路、进位选择和超前进位（先行进位）