数据的机器级表示与运算

定点数的乘法

主讲人: 邓倩妮

上海交通大学

部分内容来自：

1. 《深入理解计算机系统》第三版，机械工业出版社，作者：Bryant,R.E.等
2. Computer Organization and Design, 4th Edition, Patterson & Hennessy

本节内容

- 乘法器的实现原理
- 无符号乘法
- 补码乘法与无符号乘法的位级等效性
- 用移位运算实现乘以常数
- 乘法溢出可能导致的问题

一位乘法电路实现

S=X*Y

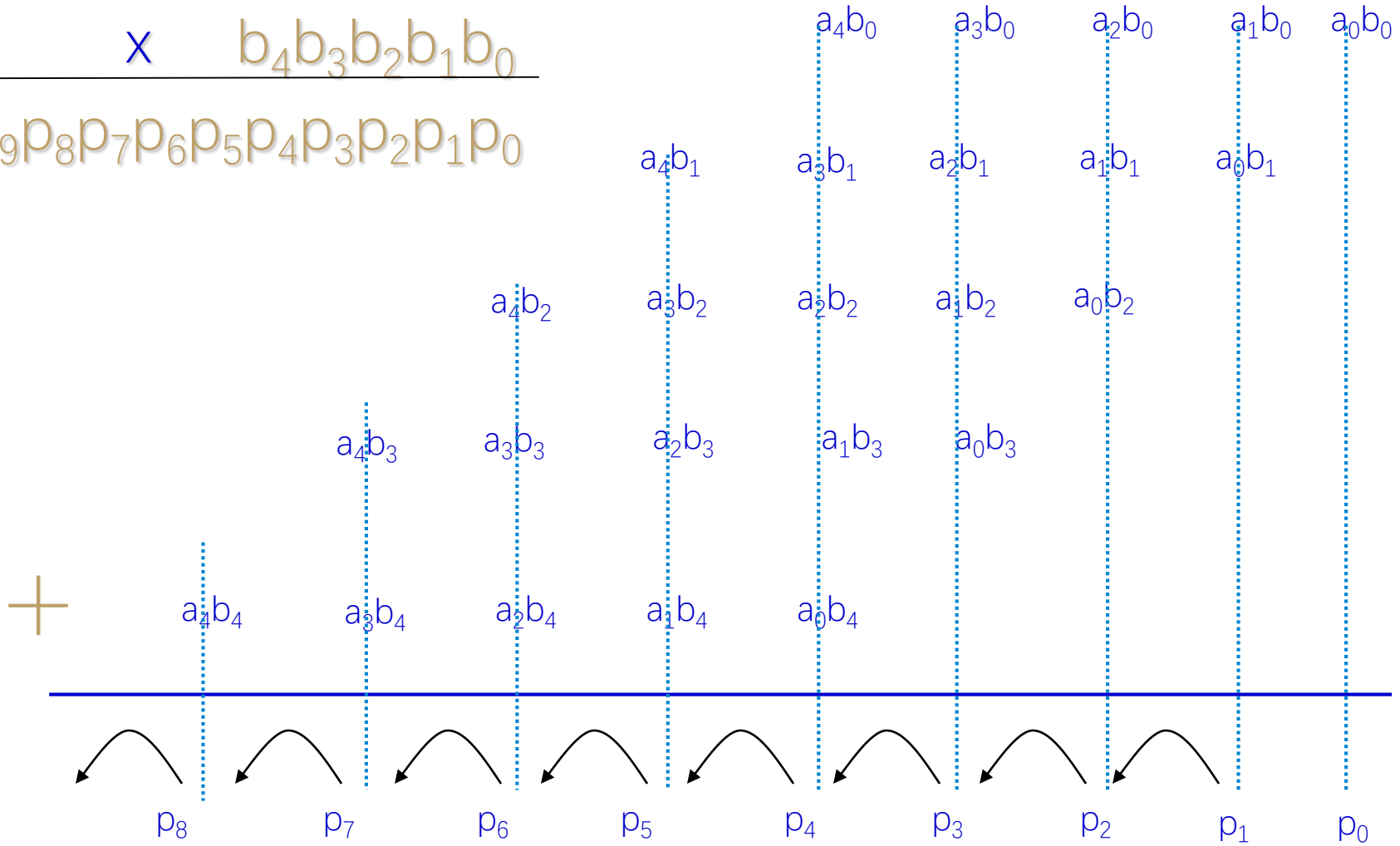
X_i	Y_i	Output
0	0	0
0	1	0
1	0	0
1	1	1

$$Output = X_i \bullet Y_i$$

一个与门就可以实现一位乘法

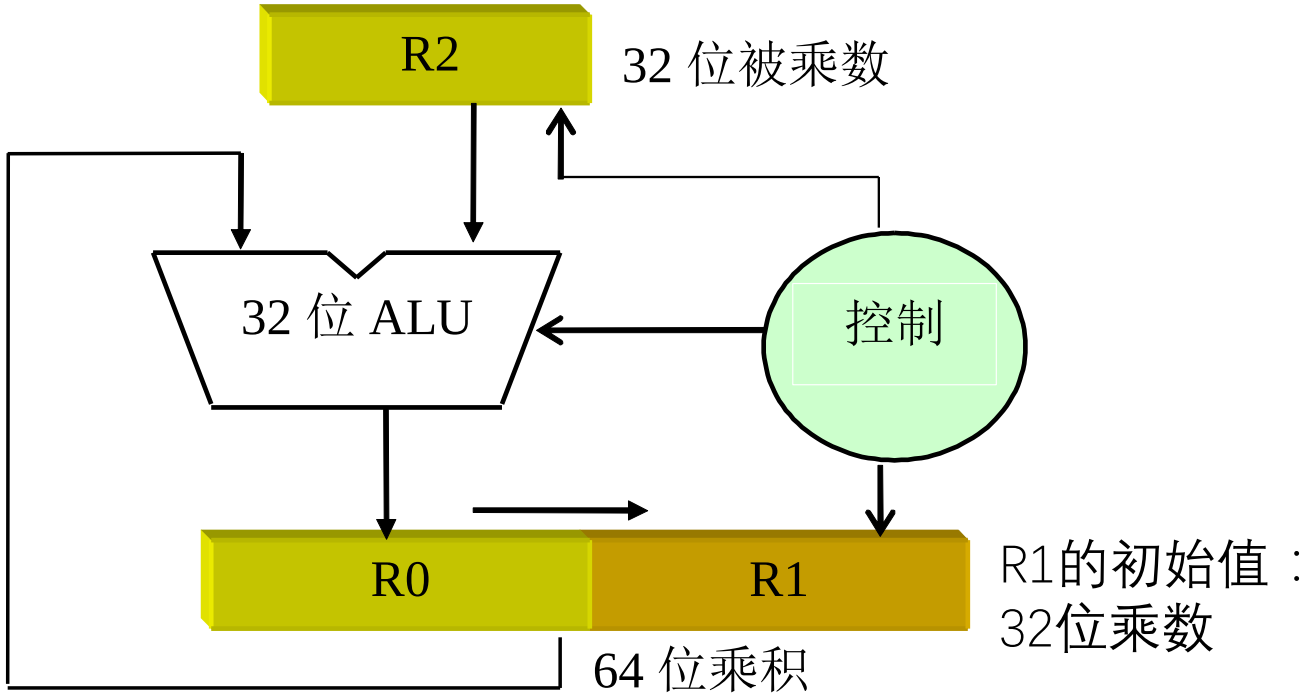
多位乘法

$$\begin{array}{r} a_4a_3a_2a_1a_0 \\ \times\quad b_4b_3b_2b_1b_0 \\ \hline p_9p_8p_7p_6p_5p_4p_3p_2p_1p_0 \end{array}$$



多位乘法必须先计算一位乘法，然后累加求和

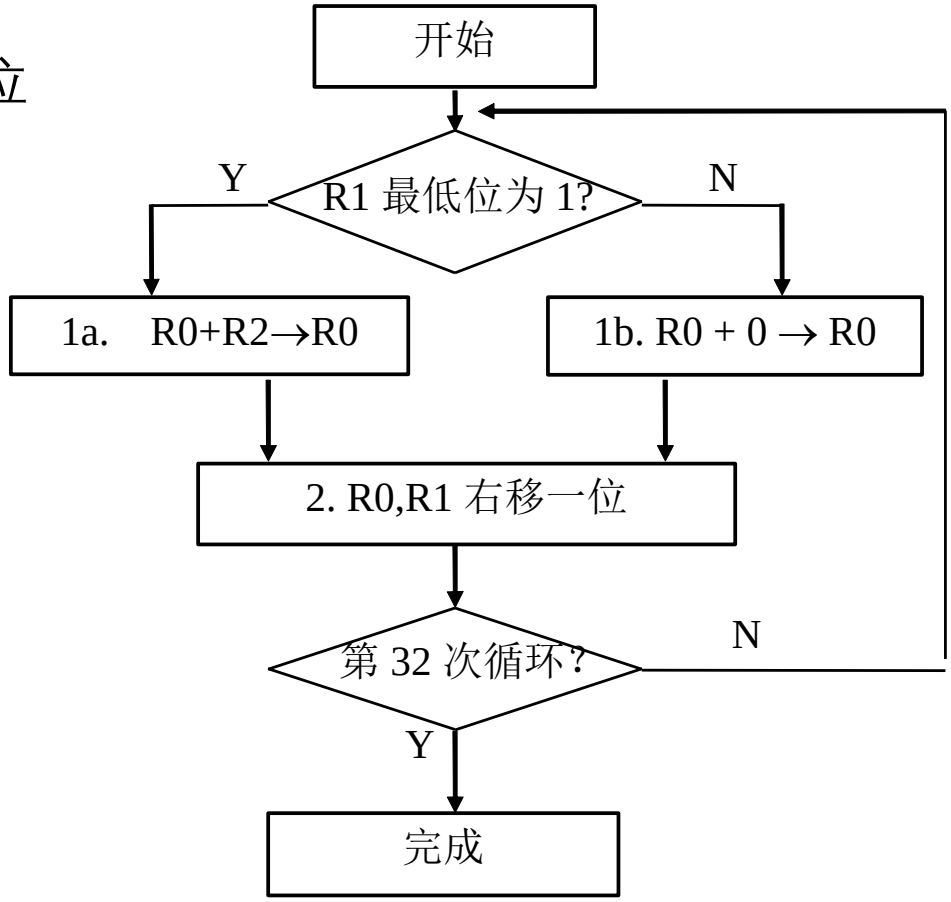
用加法器实现乘法电路



32 位原码（或：无符号数）一位乘法的方案

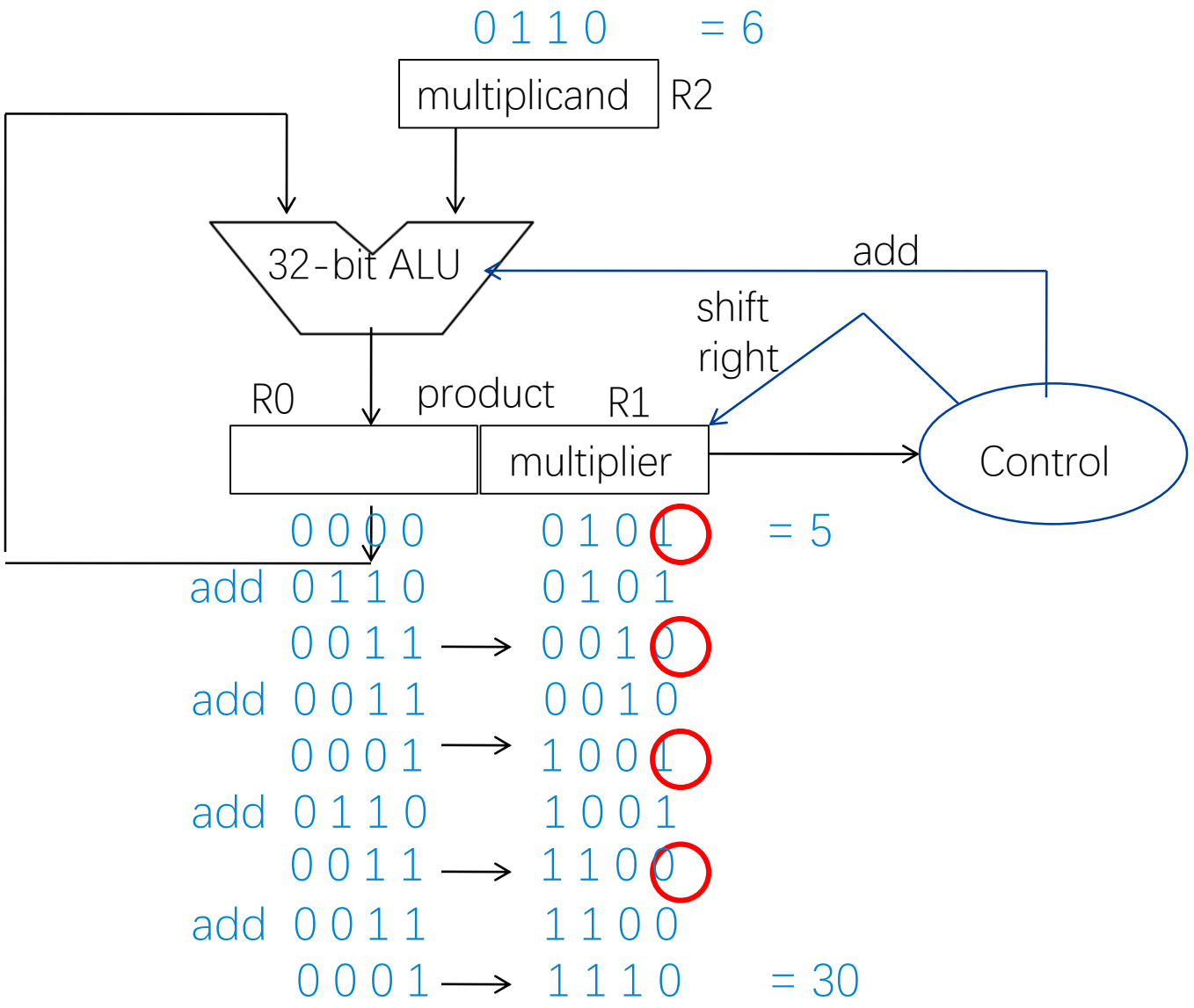
乘法流程

寄存器长度：32位
R0： 初始值为0
R1: 乘数
R2： 被乘数



实现 32 位定点原码一位乘法的流程图

用加法实现无符号乘法计算过程举例



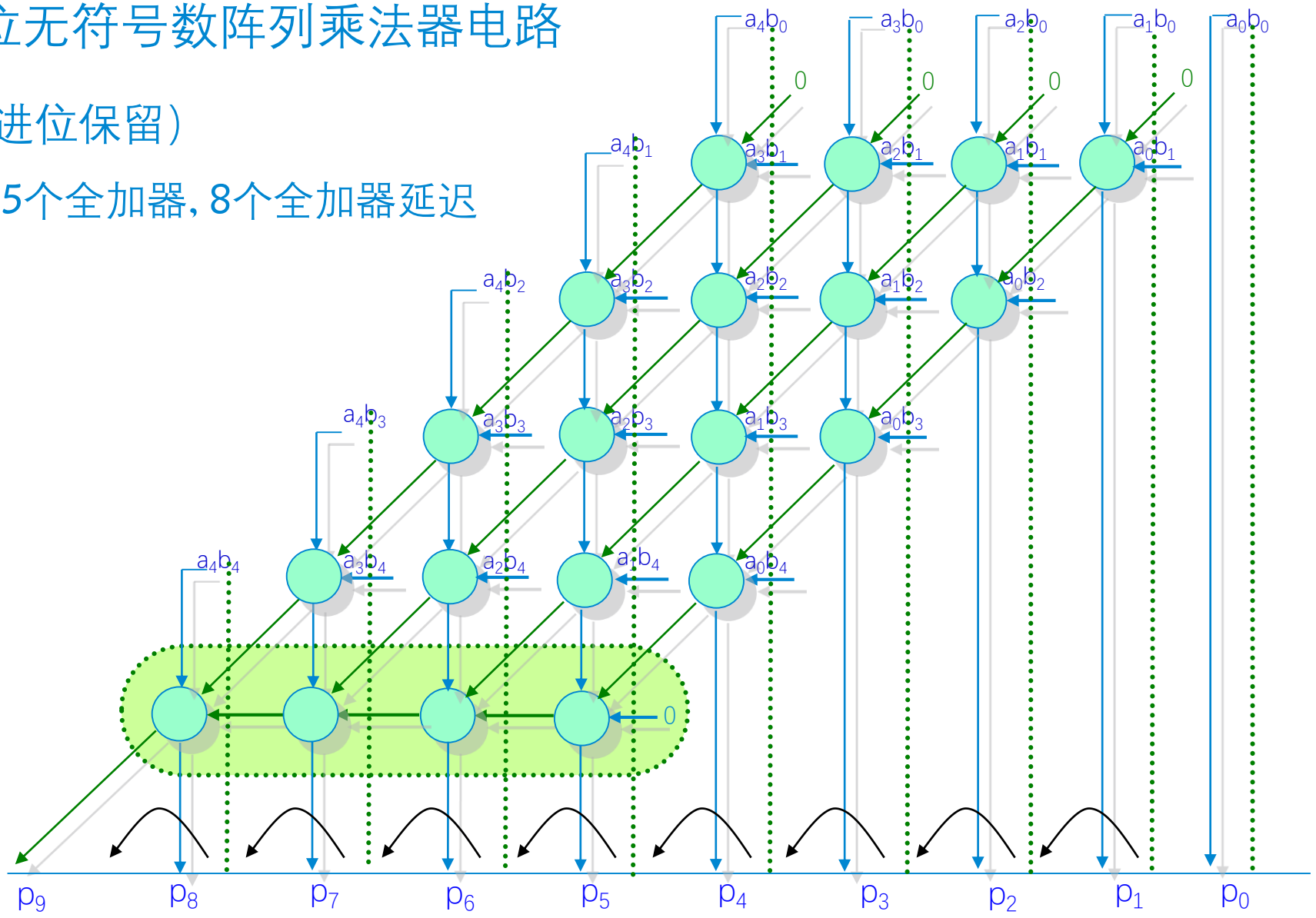
用加法器实现乘法

- 模拟手算乘法的过程
- 每次循环结束前，对加法结果的右移，等价于手算过程的一次按位乘法的结果与已有结果的向左错一位的加法
- 特点：
  - 简单
  - 速度慢： 32次加法实现一次乘法

5位无符号数阵列乘法器电路

（进位保留）

4*5个全加器，8个全加器延迟



使用的器件： n(n-1)个FA； 延迟时间： (n-1)FA +(n-1)FA

本节内容

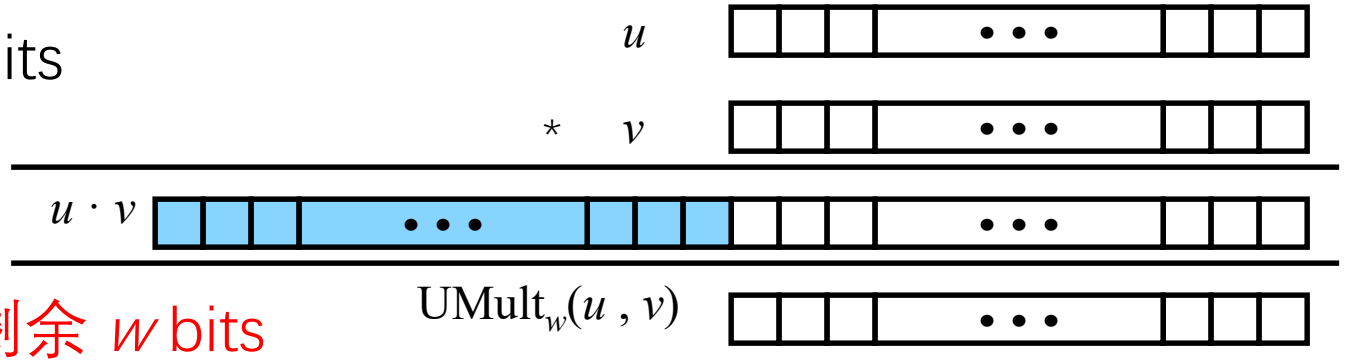
- 乘法器的实现原理
- 无符号乘法
- 补码乘法与无符号乘法的位级等效性
- 用移位运算实现乘以常数
- 乘法溢出可能导致的问题

C语言中的无符号乘法

2个操作数：w bits

乘积: 2*w bits

抛弃前 w bits，剩余 w bits



- 标准的乘法： 忽略运算结果的高w位（w: 数据的长度）
- 实现取模运算：
- $\text{UMult}_w(u, v) = u \cdot v \bmod 2^w$

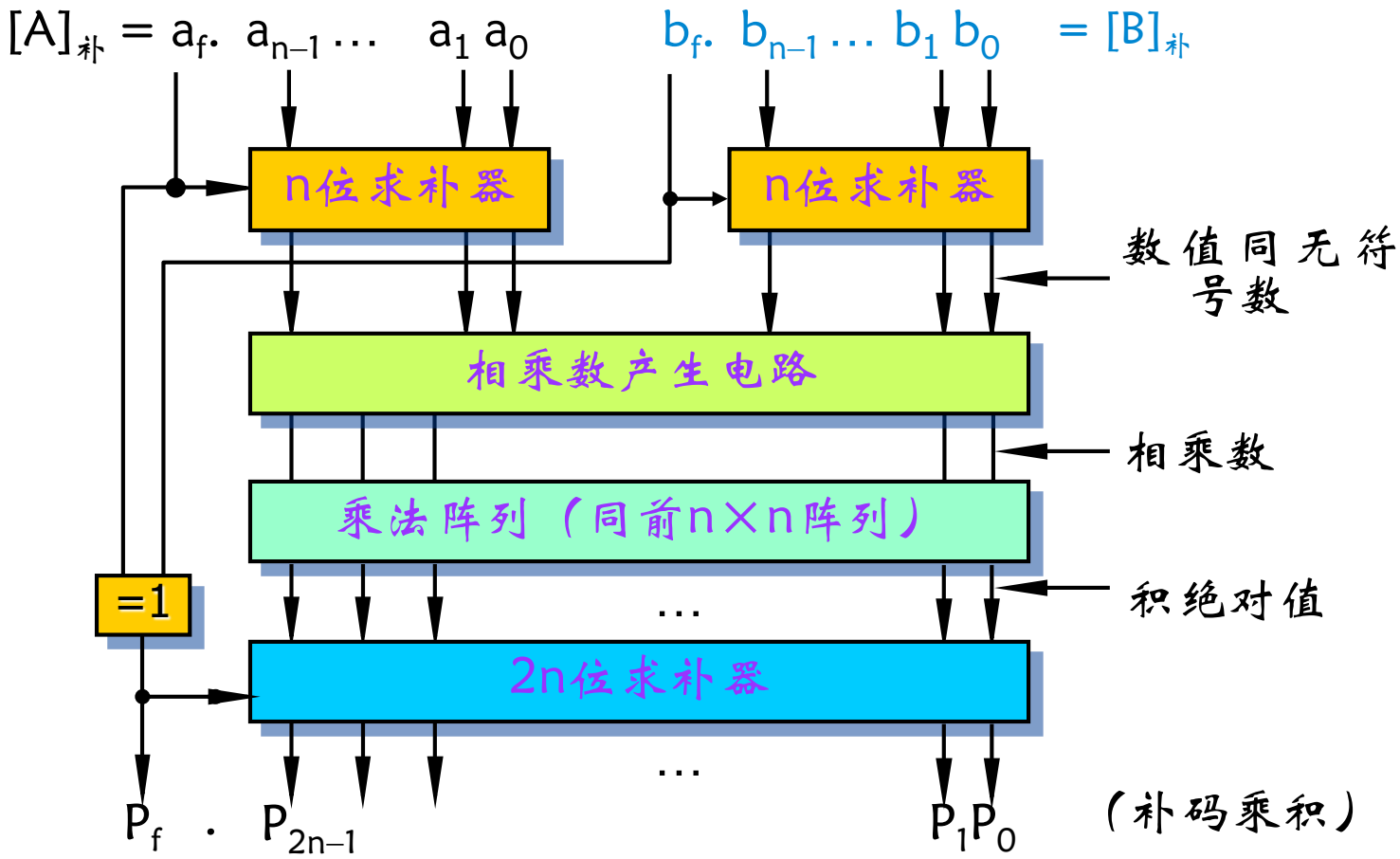
补码乘法

- $[X]_{补}+[Y]_{补}=[X+Y]_{补}$、 $[X]_{补}-[Y]_{补}=[X]_{补}+[-Y]_{补}=[X-Y]_{补}$
- 不存在类似的补码乘法公式
- 例: (-2)×3

```
          1 1 1 0
    x     0 0 1 1
    -------------
          1 1 1 0
        1 1 1 0
      0 0 0 0
    0 0 0 0
    -------------
    0 1 0 1 0 1 0
```

—错误的结果

用无符号乘法器实现补码乘法



signed vs. unsigned乘积截断后的的位级表示

- 无符号乘法和补码乘法位级的等价性

例如： w=3, 可表达的无符号数： 0~7，取模为8的运算
可表达的有符号数： -4 ~ +3，取模为8的运算

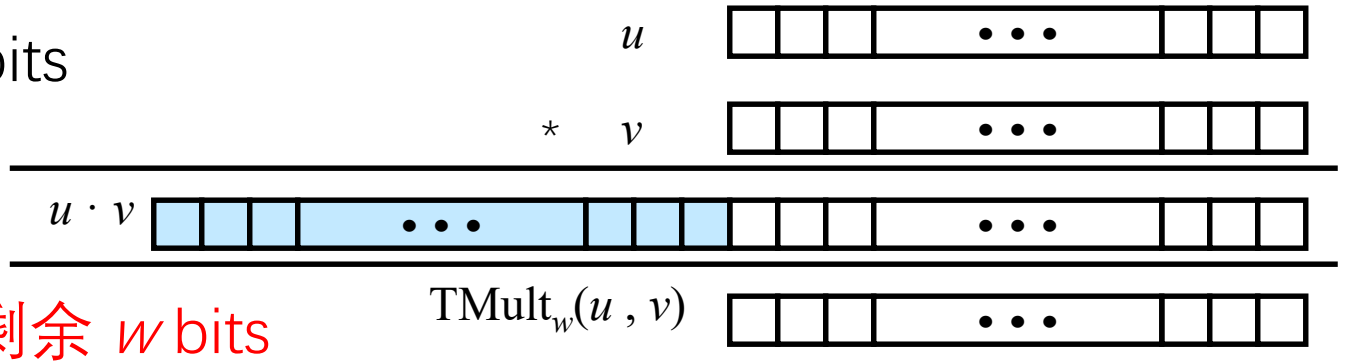
模式	x		y		$x \cdot y$		截断的$x \cdot y$	
无符号	5	[101]	3	[011]	15	[001111]	7	[111]
补码	−3	[101]	3	[011]	−9	[110111]	−1	[111]
无符号	4	[100]	7	[111]	28	[011100]	4	[100]
补码	−4	[100]	−1	[111]	4	[000100]	−4	[100]
无符号	3	[011]	3	[011]	9	[001001]	1	[001]
补码	3	[011]	3	[011]	9	[001001]	1	[001]

- signed vs. unsigned 乘法运算的结果不一样
- 但是： 截断后的w位，是一样的!

C语言中带符号乘法

2个操作数：w bits

乘积: 2*w bits



抛弃前 w bits，剩余 w bits

- 标准的乘法： 忽略运算结果的高w位
- 无符号乘法的部件，可以给带符号乘法使用

本节内容

- 乘法器的实现原理
- 无符号乘法
- 补码乘法与无符号乘法的位级等效性
- 用移位运算实现乘以常数
- 乘法溢出可能导致的问题

通过左移 实现乘 2的幂 运算

- **u << k** 等价于 **u * 2^k**
- 对无符号数、带符号数 均有效

- 例如：
  - **u << 3 == u * 8**
  - **(u << 5) – (u << 3) == u * 24**
  - 移位比乘法计算速度快
  - 编译器能自动产生优化代码

编译器对（常量）乘法的优化

```
long mul12(long x)
{
  return x*12;
}
```

C Function

Compiled Arithmetic Operations

```
leaq (%rax,%rax,2), %rax
salq $2, %rax
```

Explanation

```
t <- x+x*2
return t << 2;
```

- 乘以一个常量，编译器能自动产生优化代码

本节内容

- 乘法器的实现原理
- 无符号乘法
- 补码乘法与无符号乘法的位级等效性
- 用移位运算实现乘以常数
- 乘法溢出可能导致的问题

乘法溢出

- 乘法指令：一般没有判断溢出的功能
- 编译器：也可能不生成相应的整数乘法溢出处理代码
- 程序员要自己采取防止溢出的措施
- 例如：$X^2 > 0$ 数学上是正确的，可能在程序中是错误的
- 代码：
- int x, y;
- x= 65535;　// 机器码： 0000 FFFFH
- y= x*x ; // 机器码 ： FFFE 0001H，代表整数 -131 071

小结

- 乘法器的实现原理
- 无符号数乘法
- 有符号数乘法
- 乘法的溢出

谢谢！