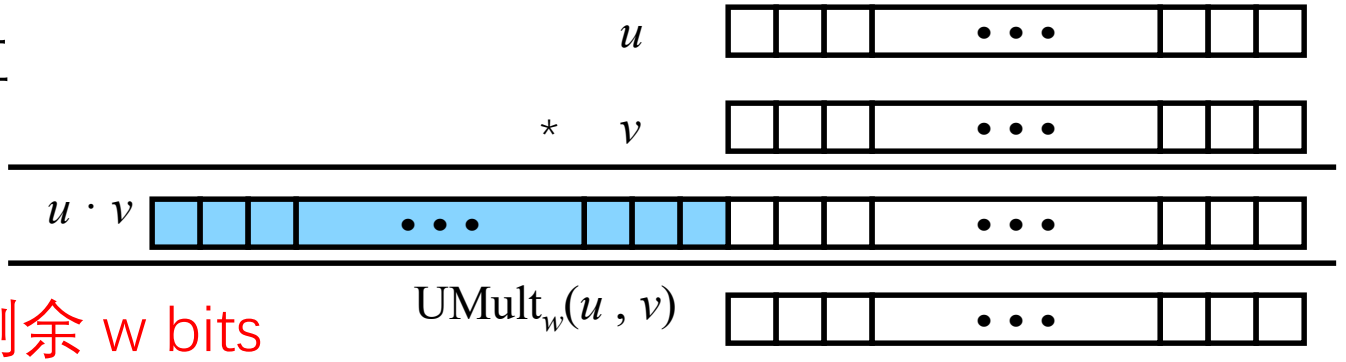
无符号数、整数的乘法

C语言中的无符号乘法

2个操作数 : w位

乘积: 2w 位

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:
- $UMult_w(u , v) = u \cdot v \mod 2^w$

补码乘法

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

• 无符号乘法和补码乘法位级的等价性

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

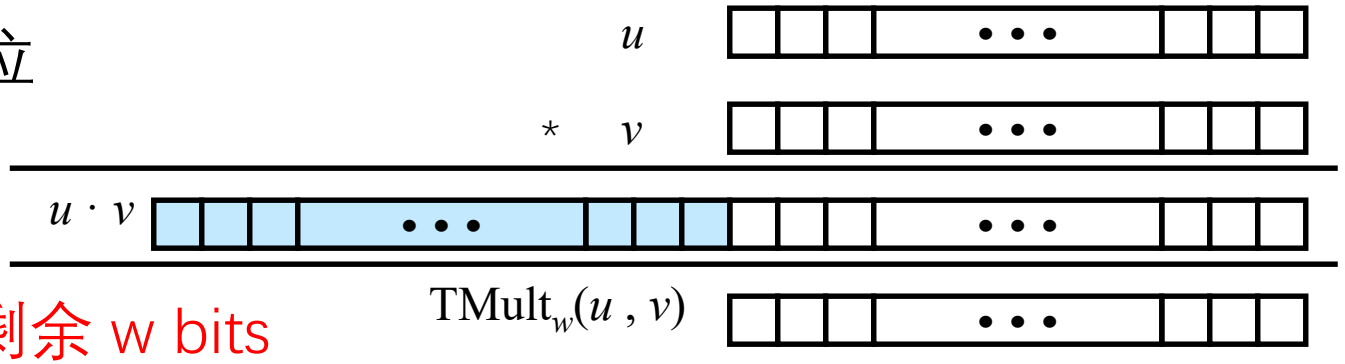
模式	x		y		x·y		截断后的x·y	
无符号	5	[101]	3	[011]	15	[001111]	7	[111]
补码	-3	[101]	3	[011]	-9	[110111]	-1	[111]
无符号	4	[100]	7	[111]	28	[011100]	4	[100]
补码	-4	[100]	-1	[111]	4	[000100]	-4	[100]
无符号	3	[011]	3	[011]	9	[001001]	1	[001]
补码	3	[011]	3	[011]	9	[001001]	1	[001]

- 编码直接相乘, 截断后的w位,
- 无论表示无符号数, 还是补码, 结果都是正确的!

C语言中带符号乘法

2个操作数 : w 位

乘积: 2w 位



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

- 用移位运算加速“乘以常数”操作

通过左移 实现乘 2的幂 运算

- **u << k** 等价于 $u \times 2^k$
- 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 函数

编译后的指令：

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

解释

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

本节内容

- 乘法溢出可能导致的问题

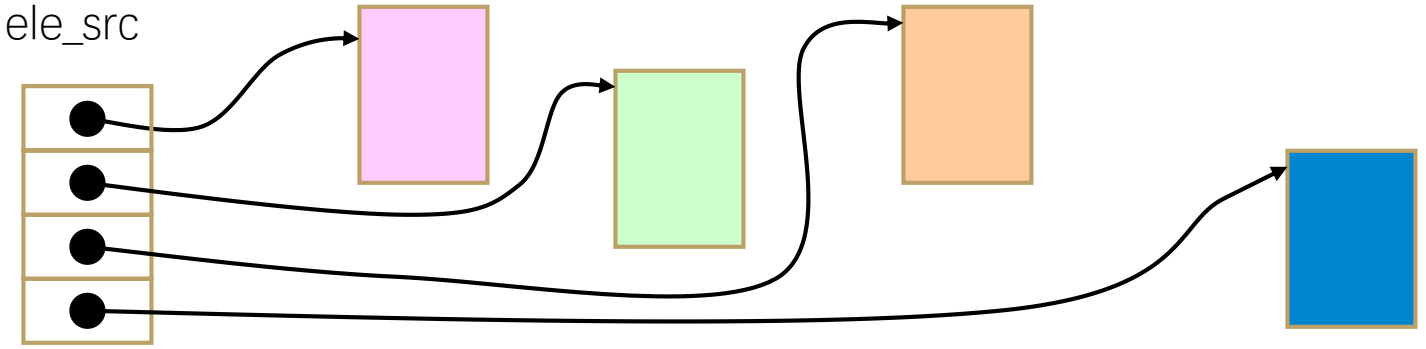
乘法溢出

- 乘法指令：一般没有判断溢出的功能
- 编译器：可能不生成整数乘法溢出的处理代码
- 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

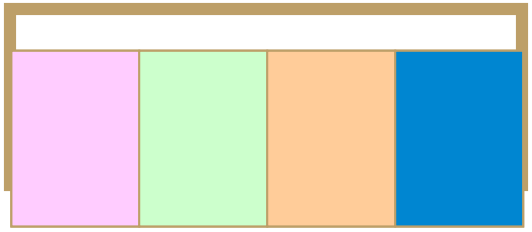
乘法溢出会影响代码安全性

- SUN XDR library

```
void* copy_elements(void *ele_src[], int ele_cnt, size_t ele_size);
```



malloc(ele_cnt * ele_size)



该函数被广泛运用于机器间的数据传输

实现代码

```
void* copy_elements(void *ele_src[], int
ele_cnt, size_t ele_size)
 {
    void *result = malloc(ele_cnt * ele_size);
    void *next = result;
    int i;
    for (i = 0; i < ele_cnt; i++) {
        /* Copy object i to destination */
        memcpy(next, ele_src[i], ele_size);
     /* Move pointer to next memory region */
     next += ele_size;
    }
    return result;
}
```

XDR 弱点

malloc(ele_cnt * ele_size)

- 假设:
  - **ele_cnt** = $2^{20} + 1$
  - **ele_size** = 4096 = 2^{12}
  - 乘法结果超过了32位，相乘得到结果为2^{12}
  - 不是正确的数组长度

- 那我们如何使这个函数更安全呢?

小结

- 无符号数乘法
- 有符号数乘法
- 用移位运算实现乘以常数
- 乘法的溢出