整数乘法运算

南京大学
计算机科学与技术系
袁春风
email: cfyuan@nju.edu.cn
2015.6

整数的乘运算

- 通常，高级语言中两个n位整数相乘得到的结果通常也是一个n位整数，也即结果只取2n位乘积中的低n位。
  - 例如，在C语言中，参加运算的两个操作数的类型和结果的类型必须一致，如果不一致则会先转换为一致的数据类型再进行计算。

```
int mul(int x, int y)
{
    int z=x*y;
    return z;
}
```

x*y 被转换为乘法指令，在乘法运算电路中得到的乘积是64位，但是，只取其低32位赋给z。

整数的乘运算

在计算机内部，一定有$x^2 \geq 0$吗?

若x是带符号整数，则不一定！

如x是浮点数，则一定！

例如，当 n=4 时, $5^2=-7<0$！

```
0101
×  0101
------
0101
+ 0101
------
00011001
```

结果溢出

只取低4位，值为-111B=-7

```
int mul(int x, int y)
{
    int z=x*y;
    return z;
}
```

如何判断返回的z是正确值？

当 !x || z/x==y 为真时

什么情况下，乘积是正确的呢？

当 $-2^{n-1} \leq x*y < 2^{n-1}$（不溢出）时

即：乘积的高n位为全0或全1，并等于低n位的最高位！

即：乘积的高n+1位为全0或全1

若x、y和z都改成unsigned类型，则判断方式为

乘积的高n位为全0，则不溢出

整数的乘运算

结论：假定两个n位无符号整数x_u和y_u对应的机器数为X_u和Y_u，$p_u=x_u\times y_u$，p_u为n位无符号整数且对应的机器数为P_u；

两个n位带符号整数x_s和y_s对应的机器数为X_s和Y_s，$p_s=x_s\times y_s$，p_s为n位带符号整数且对应的机器数为P_s。

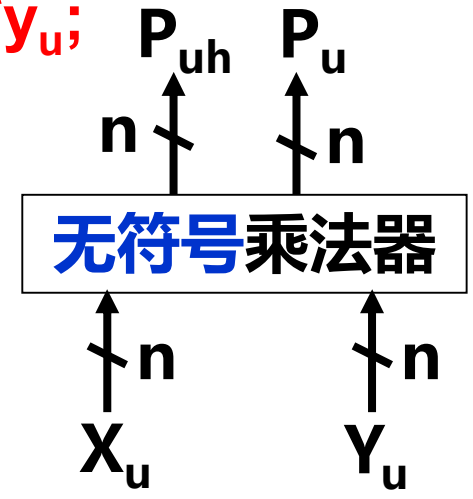
若$X_u=X_s$且$Y_u=Y_s$，则$P_u=P_s$。

无符号：若$P_{uh}=0$，则不溢出

带符号：若P_{sh}每位都等于P_s的最高位，则不溢出

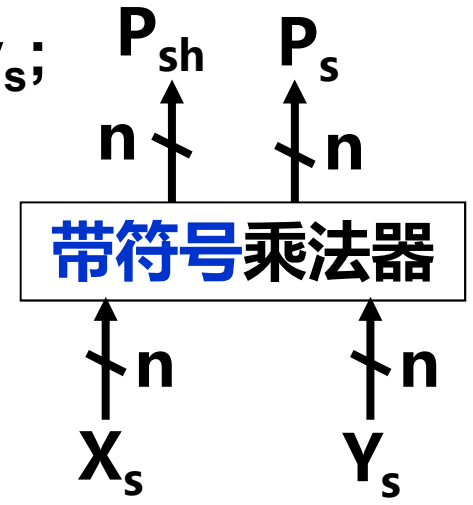
可用无符号乘来实现带符号乘，但高n位无法得到，故不能判断溢出。

$p_u=x_u*y_u;$



$P_{uh} \neq P_{sh}$

$p_s=x_s*y_s;$



整数的乘运算

- **X*Y的高n位可以用来判断溢出，规则如下：**
  - **无符号：若高n位全0，则不溢出，否则溢出**
  - **带符号：若高n位全0或全1且等于低n位的最高位，则不溢出。**

运算	x	X	y	Y	x×y	X×Y	p	P	溢出否
无符号乘	6	0110	10	1010	60	0011 1100	12	1100	溢出
带符号乘	6	0110	-6	1010	-36	1101 1100	-4	1100	溢出
无符号乘	8	1000	2	0010	16	0001 0000	0	0000	溢出
带符号乘	-8	1000	2	0010	-16	1111 0000	0	0000	溢出
无符号乘	13	1101	14	1110	182	1011 0110	6	0110	溢出
带符号乘	-3	1101	-2	1110	6	0000 0110	6	0110	不溢出
无符号乘	2	0010	12	1100	24	0001 1000	8	1000	溢出
带符号乘	2	0010	-4	1100	-8	1111 1000	-8	1000	不溢出

整数的乘运算

- 硬件不判溢出，仅保留2n位乘积，供软件使用
- 如果程序不采用防止溢出的措施，且编译器也不生成用于溢出处理的代码，就会发生一些由于整数溢出而带来的问题。
- 指令：分无符号数乘指令、带符号整数乘指令
- 乘法指令的操作数长度为n,而乘积长度为2n，例如：
  - IA-32中，若指令只给出一个操作数SRC，则另一个源操作数隐含在累加器AL/AX/EAX中，将SRC和累加器内容相乘，结果存放在AX（16位时）或DX-AX（32位时）或EDX-EAX（64位时）中。
  - 在MIPS处理器中，mult会将两个32位带符号整数相乘，得到的64位乘积置于两个32位内部寄存器Hi和Lo中，因此，可以根据Hi寄存器中的每一位是否等于Lo寄存器中的第一位来进行溢出判断。

乘法指令不生成溢出标志，编译器可使用2n位乘积来判断是否溢出！

整数乘法溢出漏洞

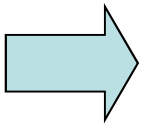
以下程序存在什么漏洞，引起该漏洞的原因是什么。

```
/* 复制数组到堆中，count为数组元素个数 */
int copy_array(int *array, int count) {
      int i;
      /* 在堆区申请一块内存 */
      int *myarray = (int *) malloc(count*sizeof(int));
      if (myarray == NULL)
            return -1;
      for (i = 0; i < count; i++)
            myarray[i] = array[i];
      return count;
}
```

2002年，Sun Microsystems公司的RPC XDR库带的xdr_array函数发生整数溢出漏洞，攻击者可利用该漏洞从远程或本地获取root权限。

攻击者可构造特殊参数来触发整数溢出，以一段预设信息覆盖一个已分配的堆缓冲区，造成远程服务器崩溃或者改变内存数据并执行任意代码。

当参数count很大时，则count*sizeof(int)会溢出。如count=$2^{30}+1$时，count*sizeof(int)=4。



堆（heap）中大量数据被破坏！

变量与常数之间的乘运算

- 整数乘法运算比移位和加法等运算所用时间长，通常一次乘法运算需要多个时钟周期，而一次移位、加法和减法等运算只要一个或更少的时钟周期，因此，编译器在处理变量与常数相乘时，往往以移位、加法和减法的组合运算来代替乘法运算。

  例如，对于表达式x*20，编译器可以利用$20=16+4=2^4+2^2$，将x*20转换为(x<<4)+(x<<2)，这样，一次乘法转换成了两次移位和一次加法。

- 不管是无符号数还是带符号整数的乘法，即使乘积溢出时，利用移位和加减运算组合的方式得到的结果都是和采用直接相乘的结果是一样的。