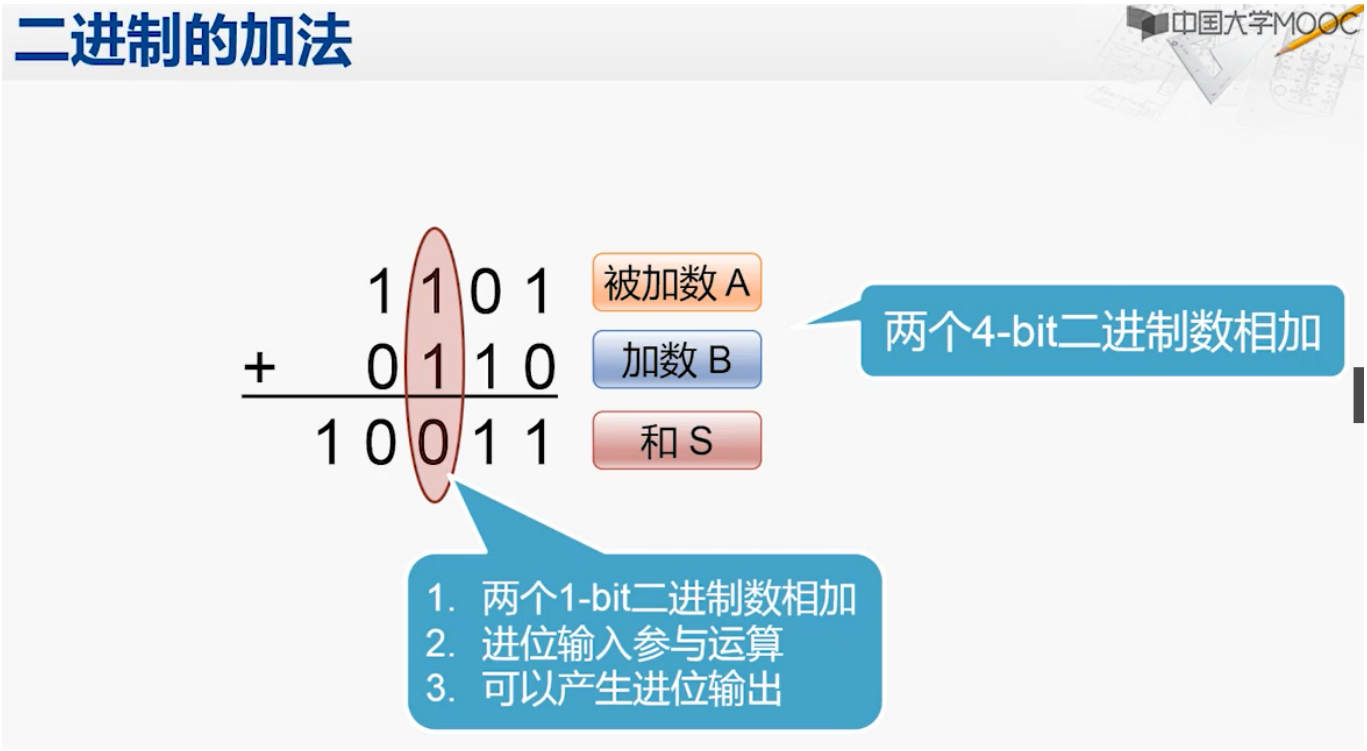
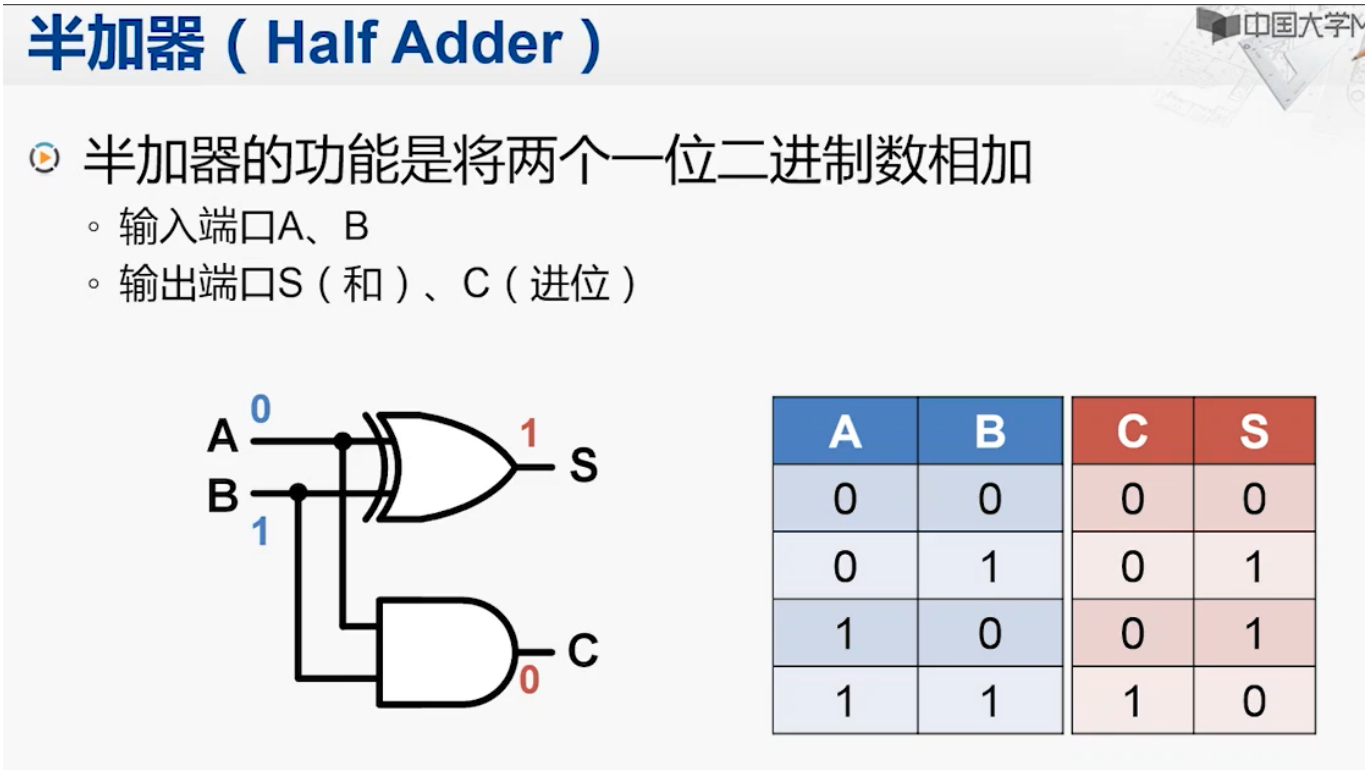


利用逻辑操作实现加法，真是巧妙



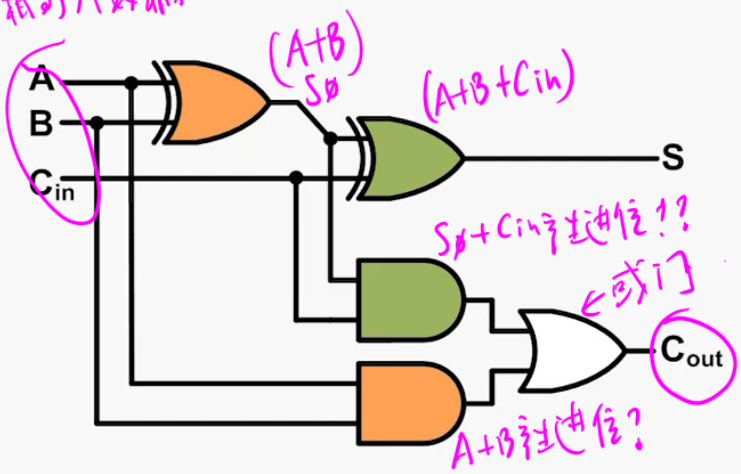
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

A+B 可以改写成（1位运算）：

- S = A ^ B => 加和结果；
- C = A & B => 进位结果；

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）



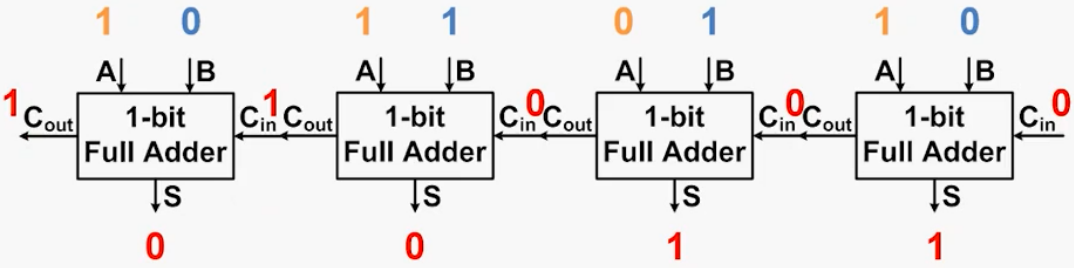
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

4-bit加法器

```
  1 1 0 1   被加数 A
+ 0 1 1 0   加数 B
-----------
1 0 0 1 1
进位C  和 S
```

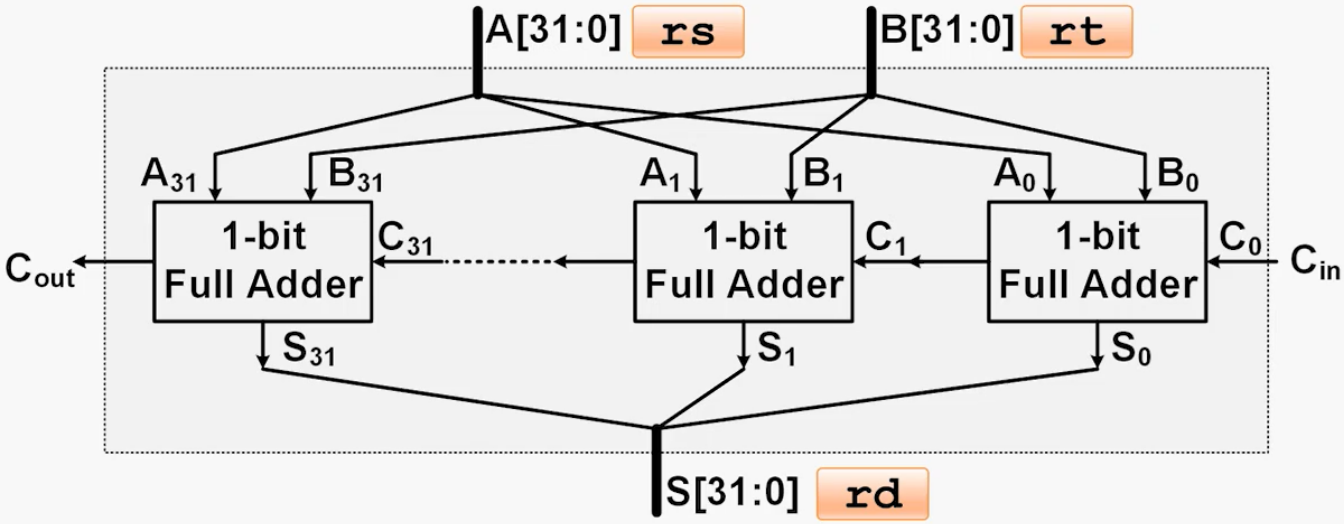
两个4-bit二进制数相加

由四个全加器构成的4-bit加法器



加法运算的实现示例

- add rd,rs,rt
- addu rd,rs,rt



“进位”和“溢出”示例

- 注意区分“进位”和“溢出”
  - 有“溢出”时，不一定有“进位”
  - 有“进位”时，不一定有“溢出”

无符号数：3 + 5 = 8
有符号数：3 + 5 = (-8)

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

有“溢出”，无“进位”

无符号数：14 + 12 = 10
有符号数：(-2)+(-4)=(-6)

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

有“进位”，无“溢出”

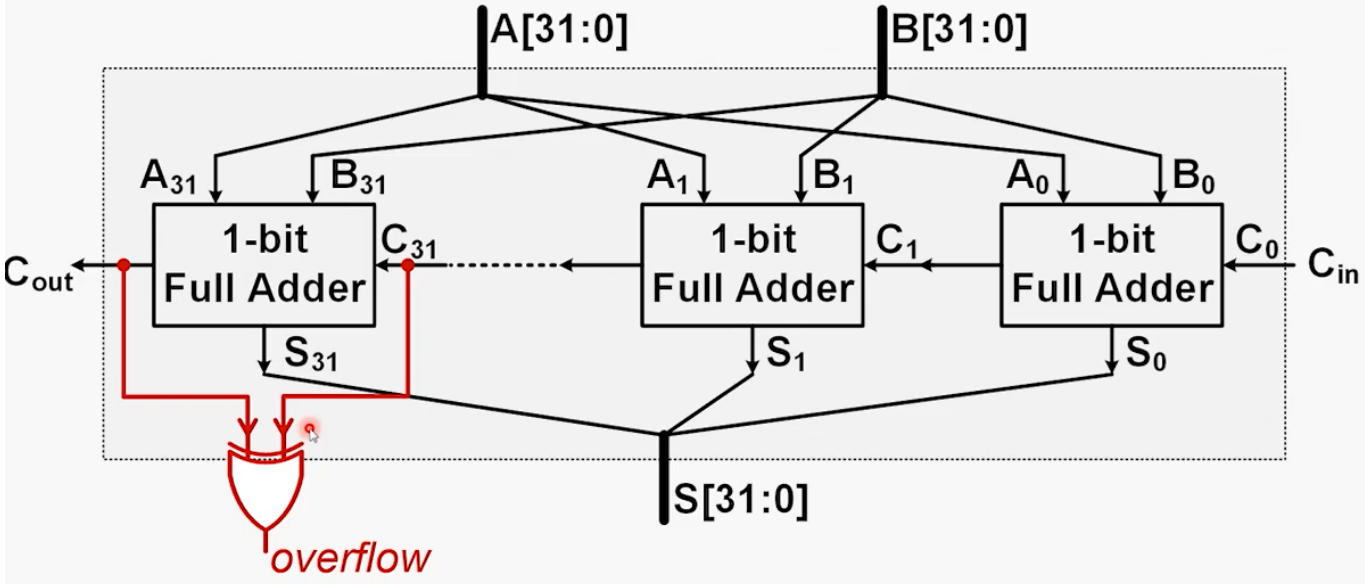
溢出是针对有符号数来说的。

检查加法运算结果是否溢出

- “溢出”（overflow）
  - 运算结果超出了正常的表示范围
- “溢出”仅针对有符号数运算
  - 两个正数相加，结果为负数
  - 两个负数相加，结果为正数

“溢出”的检查方法

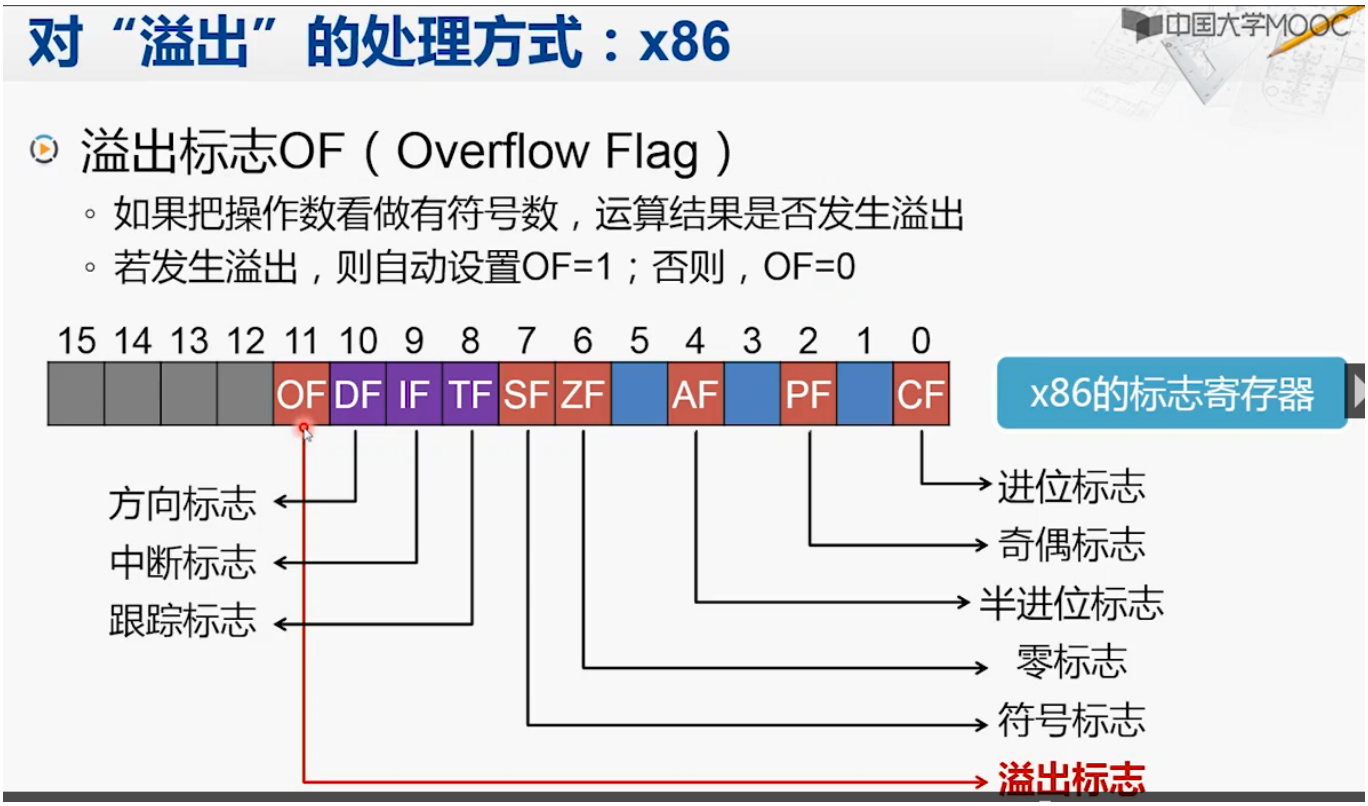
- “最高位的进位输入”不等于“最高位的进位输出”

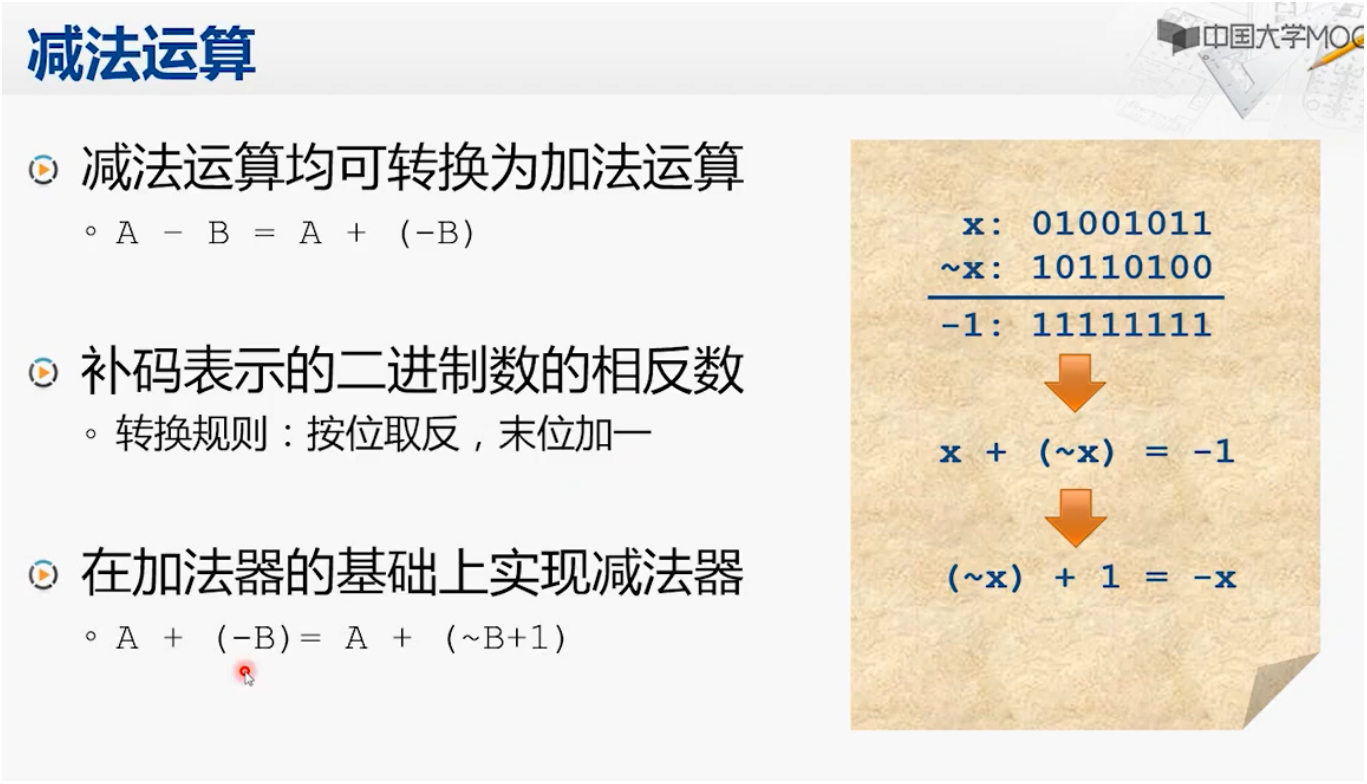


加法运算不区分有符号数和无符号数，全都是通过上面一套加法器来实现，按位进行相加。

MIPS根据溢出是否要处理，分为两个指令。

x86根据溢出标志位来检测是否发生了溢出。





联想到-B AND A， -B跟A进行按位与操作，其实是做**ALIGN DOWN** 如果B是一个只有1bit的数值:

```
 B:      0001 0000
-B:      1111 0000
~B + 1 : 1110 1111 + 1 => 1111 0000
-B AND A: 等价于将A的低4bits消零
```

利用加法器来实现减法很巧妙：