《计算机系统基础（四）：编程与调试实践》

真值与机器数

真值与机器数

整数的编码

浮点数的编码

整数的编码

带符号整数： char、short、int、long

无符号整数：unsigned

```
#include "stdio.h"
void main( )
{    int  ai = 100, bi = 2147483648, ci = -100;
     unsigned  au = 100, bu = 2147483648, cu = -100;
     printf("ai=%d,   bi=%d,  ci=%d \n", ai, bi, ci);
     printf("au=%u ,  bu=%u,  cu=%u \n", au, bu, cu);
}
```

上述代码的执行结果是什么?

带符号整数和无符号整数的编码

分析：

真值			机器数	
十进制	二进制	十六进制	编码	说明
100	0110 0100	64H	00000064H	
2147483648		80000000H	80000000H	
-100	-0110 0100	-64H	FFFFFF9CH	补码

带符号整数和无符号整数的编码

分析：

真值			机器数	
十进制	二进制	十六进制	编码	说明
100	0110 0100	64H	00000064H	
2147483648		80000000H	80000000H	
-100	-0110 0100	-64H	FFFFFF9CH	补码

变量	类型	机器数	真值
bi	int	80000000H	-2147483648
cu	unsigned	FFFFFF9CH	4294967196

总结：

带符号整数：补码

符号	数值

无符号整数：二进制编码

数值

浮点数的编码

```
#include "stdio.h"
 void main( )
{          int  ai = 100, bi = -100;
           float  af = 100, bf = -100;
           printf("ai=%d,   bi=%d \n", ai, bi );
           printf("af=%f ,  bf=%f \n", af, bf );
}
```

上述代码运行时，各变量的机器数分别是什么?

IA-32中的寄存器：定点寄存器组、浮点寄存器栈、多媒体扩展寄存器组

IA-32中指令类型：x86指令、x87浮点处理指令、MMX指令、SSE指令

IA-32中的浮点处理架构：x87 架构、SSE架构

真值与机器数

总结：

带符号整数：采用补码的表示，如int类型

符号	数值

无符号整数：二进制的表示，如unsigned int 类型

数值

浮点数：采用IEEE 754标准，如float类型

符号	阶码	尾数

谢谢！

《计算机系统基础（四）：编程与调试实践》

数据的宽度与存储

数据的宽度与存储

数据存储的宽度

数据存储的排列方式

数据存储的对齐方式

数据存储的宽度

C语言支持多种格式的整数和浮点数表示。下表给出了不同机器中C语言数值数据类型的宽度（字节数）。

C声明	典型32位机器	Compaq Alpha机器
char	1	1
short int	2	2
int	4	4
long int	4	8
char*	4	8
float	4	4
double	8	8

数据存储的排列方式

假设数据d=0x12345678，存储在0x00effe5c地址单元中。

0x00effe5c	0x00effe5d	0x00effe5e	0x00effe5f	
0x12	0x34	0x56	0x78	大端方式
0x78	0x56	0x34	0x12	小端方式

大端方式 最高有效字节存放在低地址单元中，
最低有效字节存放在高地址单元中。

小端方式 最高有效字节存放在高地址单元中，
最低有效字节存放在低地址单元中。

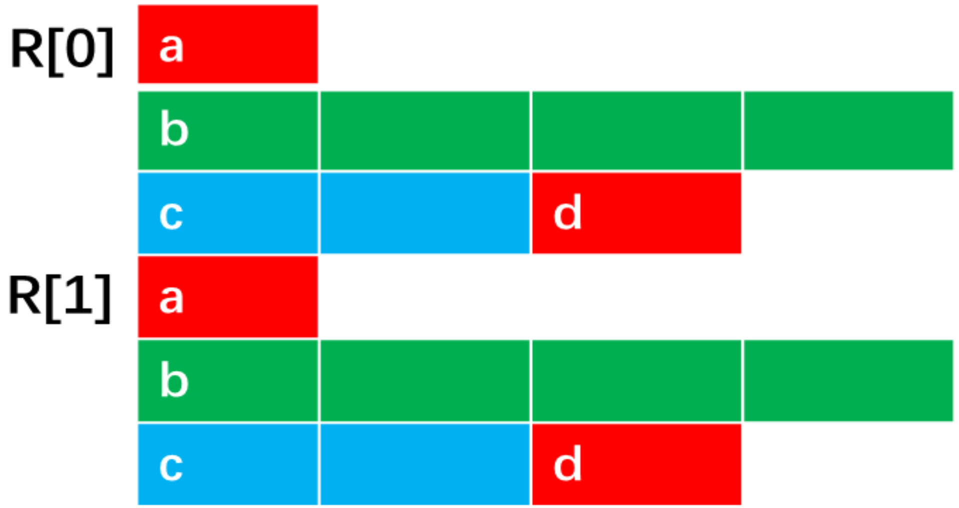
数据存储的对齐方式

```c
#include "stdio.h"
void main( )
{   struct record{
        char   a ;
        int    b ;
        short  c ;
        char   d ;
     } R[2] ;
    R[0].a=1;  R[0].b=2;  R[0].c=3;  R[0].d=4;
    R[1].a=5;  R[1].b=6;  R[1].c=7;  R[1].d=8;
    printf("数据存储时的边界对齐");
}
```

1. 查看在结构record中，成员变量a、b、c和d的边界对齐方式。
2. 查看数组元素R[0]和R[1]的边界对齐方式。
3. 计算数组R占用的字节数。record的定义是否可以优化？给出优化后的record定义，并计算record优化定义后数组R占用的字节数。

数据存储的对齐方式

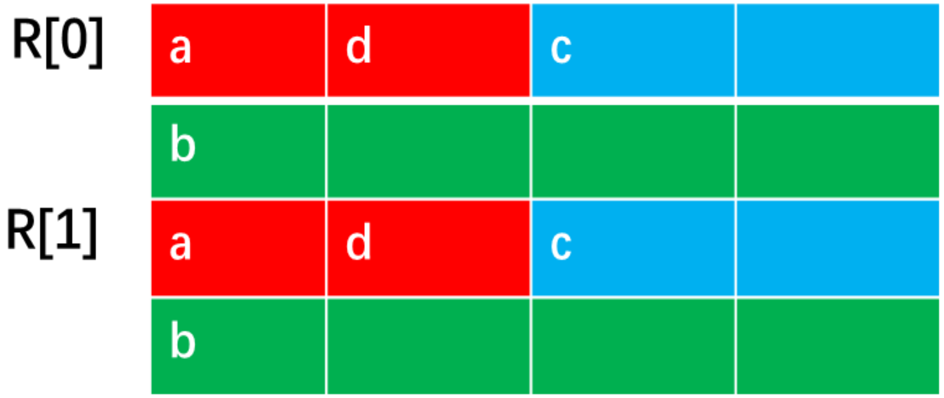
```
#include "stdio.h"
void main( )
{   struct record{
        char   a ;
        int    b ;
        short  c ;
        char   d ;
    } R[2];
    R[0].a=1;  R[0].b=2;  R[0].c=3;  R[0].d=4;
    R[1].a=5;  R[1].b=6;  R[1].c=7;  R[1].d=8;
    printf("数据存储时的边界对齐");
}
```



数组R占用（1+3+4+2+1+1）×2 = 24 字节

数据存储的对齐方式

```
#include "stdio.h"
void main()
{   struct record{
        char   a ;
        char   d ;
        short  c ;
        int    b ;
    } R[2];
    R[0].a=1;  R[0].b=2;  R[0].c=3;  R[0].d=4;
    R[1].a=5;  R[1].b=6;  R[1].c=7;  R[1].d=8;
    printf("数据存储时的边界对齐");
}
```



数组R占用（1+1+2+4) ×2 = 16 字节

数据存储的对齐方式

总结

1. 基本数据类型的对齐策略

基本类型	Windows	Linux
char	任意地址	任意地址
short	地址是2的倍数	地址是2的倍数
int	地址是4的倍数	地址是4的倍数
long long	地址是8的倍数	地址是4的倍数
float	地址是4的倍数	地址是4的倍数
double	地址是8的倍数	地址是8的倍数

2. struct结构体数据的对齐策略

结构体数据的首地址是4的倍数，成员变量按基本数据类型对齐

谢谢！

《计算机系统基础（四）：编程与调试实践》

数据类型的转换

数据类型的转换

整数之间的数据类型转换

整数和浮点数之间的转换

C语言中的自动类型转换

整数之间的数据类型转换

```
#include "stdio.h"
void main()
{   short           si=-100;
    unsigned short usi=si;
    int             i=usi;
    unsigned        ui=usi;
    int             i1=si;
    unsigned        ui1=si;
    int             i2=0x12348765;
    short           si2=i2;
    unsigned short usi2=i2;
    int             i3=si2;
    int             i4=4294967296;
    printf("si=%d,usi=%u,i=%d,ui=%u,i1=%d,ui1=%u\n",si,usi,i,ui,i1,ui1);
    printf("i2=%d,si2=%d,usi2=%u,i3=%d,i4=%d \n", i2,si2,usi2,i3,i4);
}
```

1. 这些赋值运算执行后，赋值运算左右两侧变量的值相等吗?
2. 程序运行过程中，各变量存储的机器数分别是什么?
3. 程序中i2赋值给si2，si2赋值给i3，i2和i3的值相等吗?
4. int型数据的范围是:

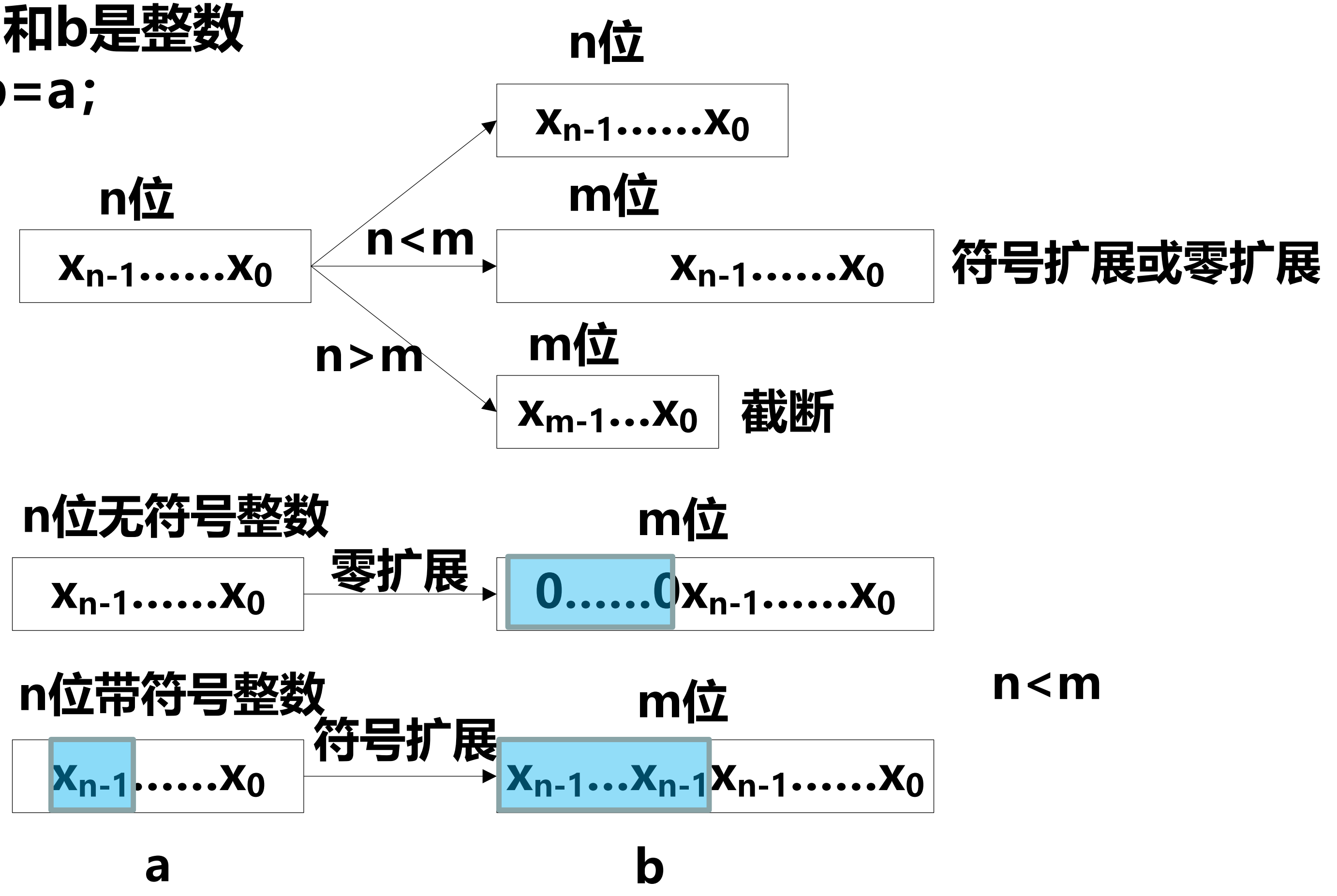
   -2147483648~2147483647

   i4的值是多少?

整数之间的数据类型转换

C语言中，整数的赋值不是在真值上的复制，而是在机器数上的赋值。

a和b是整数
b=a;



C语言中的“=”是赋值运算符，不同于数学上的等于符号“=”。

整数与浮点数之间的转换

```
#include "stdio.h"
int main()
{   int     i1=0x7fffffff,       i2, itemp;
    float  f1=0x987654321, f2,ftemp;
    ftemp=i1;
    i2=ftemp;          //i2=(int)(float)i1;
    itemp=f1;
    f2=itemp;          //f2=(float)(int)f1;
    printf("i1=%d,i2=%d,f1=%f,f2=%f\n", i1,i2,f1,f2);
}
```

1. 代码运行过程中，各变量存储的机器数分别是什么?
2. i1和i2的值相同吗？为什么?
3. f1和f2的值相同吗？为什么?

整数与浮点数之间的转换

整数与浮点数之间的转换，是在编码上的转换。

带符号整数：补码

	1位	31位
int	x_{31}	x_{30} …… x_0
	符号	数值位

浮点数：float、double IEEE 754标准

	1位	8位	23位
float	f_{31}	$f_{30}...f_{23}$	$f_{22}...f_0$
	符号	阶码	尾数

整数与浮点数之间的转换

i1=0x7fff ffff; ftemp=i1; i2=ftemp ;

i1:　0x7fff ffff　补码

= 0111 1111 1111 1111 1111 1111 1111 1111B

1.11 1111 1111 1111 1111 1111 1111 1111 $\times 2^{30}$　真值

+1　（尾数入操作）

$\approx 10.0 \times 2^{30} = 1.0 \times 2^{31}$

ftemp : 0 1001 1110 000 0000 0000 0000 0000 0000B　float

31+127=128+16+8+4+2

1.0×2^{31}　真值

=1000 0000 0000 0000 0000 0000 0000 0000B

i2:　1000 0000 0000 0000 0000 0000 0000 0000B　补码

=0x8000 0000

补码→float编码→补码，整数与浮点之间的转换不是机器数上的复制，而是编码上的转化。在int→float转换中，可能会有精度的损失。

整数与浮点数之间的转换

```
#include "stdio.h"
int main()
{   int     i1=0x7fffffff,       i2, itemp;
    float  f1=0x987654321, f2,ftemp;
    ftemp=i1;
    i2=ftemp;        //i2=(int)(float)i1;
    itemp=f1;
    f2=itemp;        //f2=(float)(int)f1;
    printf("i1=%d,i2=%d,f1=%f,f2=%f\n", i1,i2,f1,f2);
}
```

1. 代码运行过程中，各变量存储的机器数分别是什么?
2. i1和i2的值相同吗？为什么?
3. f1和f2的值相同吗？为什么?

整数与浮点数之间的转换

f1=0x987654321；itemp=f1; f2=itemp;

0x987654321=1001 1000 0111 0110 0101 0100 0011 0010 0001B

=1.001 1000 0111 0110 0101 0100 0011 0010 0001 $\times 2^{35}$

f1: 0 1010 0010 001 1000 0111 0110 0101 0100 float

=0x51187654 35+127=128+32+2

1.001 1000 0111 0110 0101 0100 $\times 2^{35}$ 真值

=1001 1000 0111 0110 0101 0100 0000 0000 0000B

itemp: 1000 0000 0000 0000 0000 0000 0000 0000B 补码

-111 1111 1111 1111 1111 1111 1111 1111B

+1

-1000 0000 0000 0000 0000 0000 0000 0000B 真值

$=-1.0\times 2^{31}$

f2: 1 1001 1110 000 0000 0000 0000 0000 0000B float

=0xcf000000 31+127=128+16+8+4+2

float编码→补码→float编码，整数与浮点之间的转换不是机器数上的复制，而是编码上的转化。在float→int转换中，可能会有溢出问题。

整数与浮点数之间的转换

总结：

1. **整数与浮点数转换时，是在编码格式上的转换。**
2. **在int ↔ float转换中，可能会有精度损失、溢出、小数丢弃等问题导致的数据不一致。**
3. **不同编译系统采用的编译优化有差异，同一程序在不同系统上运行，得到的结果可能不一样。**

C语言中的自动类型转换

已知 $f(n) = \sum_{i=0}^{n} 2^i = 2^{n+1}-1 = \overbrace{11\bullet\bullet\bullet\bullet\bullet\bullet1}^{n+1}$ B，计算 f(n)的C语言函数f1如下。

```
int f1( unsigned int n )
 {  int sum = 1, power = 1;
    int i;
    for (  i = 0; i <= n - 1; i ++ )
    {   power *= 2;
        sum += power;
     }
    return sum;
 }
```

1. 执行f1(0)时，为什么会出现死循环?
2. 为了得到正确的值，应该如何修改函数f1?

数据类型的转换

总结：

1. **整数与整数之间的转换是在机器数上的复制**
2. **整数与浮点数之间的转换是在编码上的转换**
3. **一个运算表达式中有不同数据类型时，C语言会自动进行类型转换**

谢谢！

《计算机系统基础（四）：编程与调试实践》

整数加减运算

整数加减运算

整数加减运算的电路

状态标志CF、ZF、SF和OF

整数加减运算结果的溢出问题

整数加减运算电路

补码加减运算公式：

$$[x+y]_{补} = [x]_{补} + [y]_{补} \quad (\bmod 2^n)$$

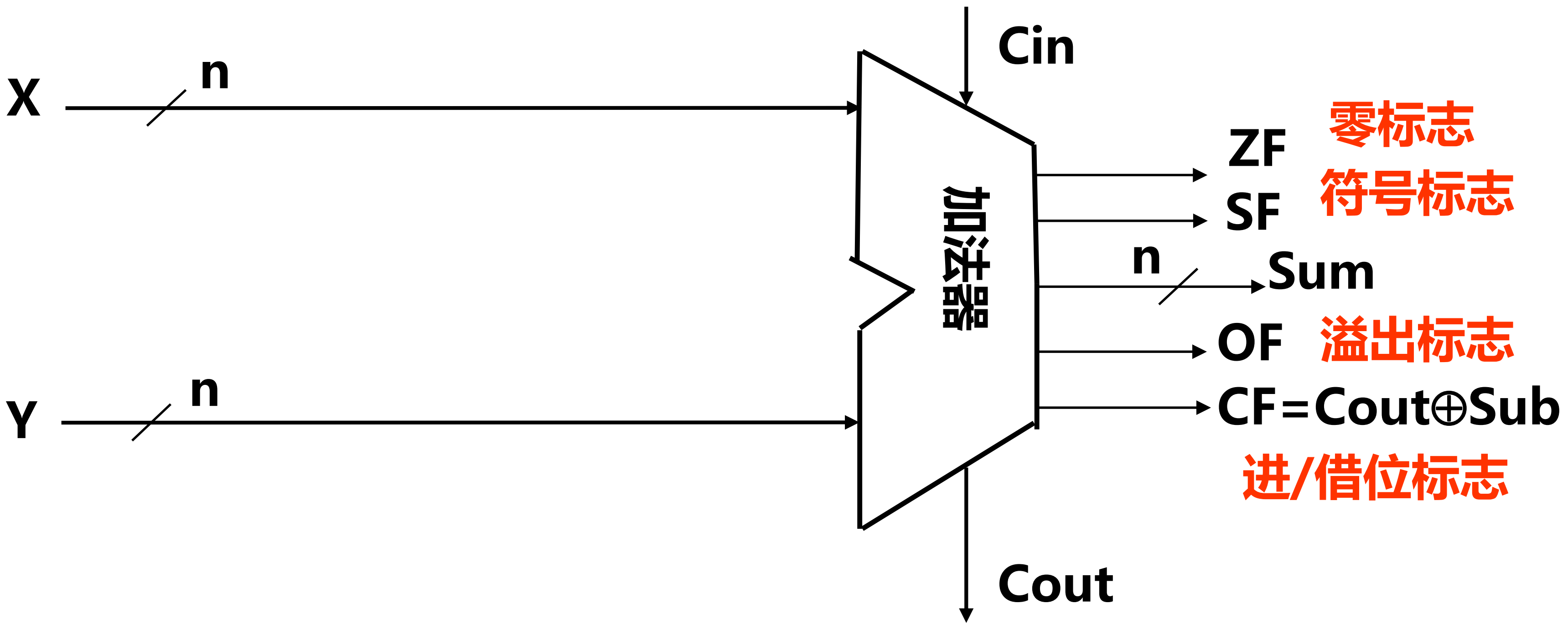
$$[x-y]_{补} = [x]_{补} + [-y]_{补} \quad (\bmod 2^n)$$

整数加减运算电路

补码加减运算公式：

$$[x+y]_{补} = [x]_{补} + [y]_{补} \quad (\bmod 2^n)$$

$$[x-y]_{补} = [x]_{补} + [-y]_{补} \quad (\bmod 2^n)$$

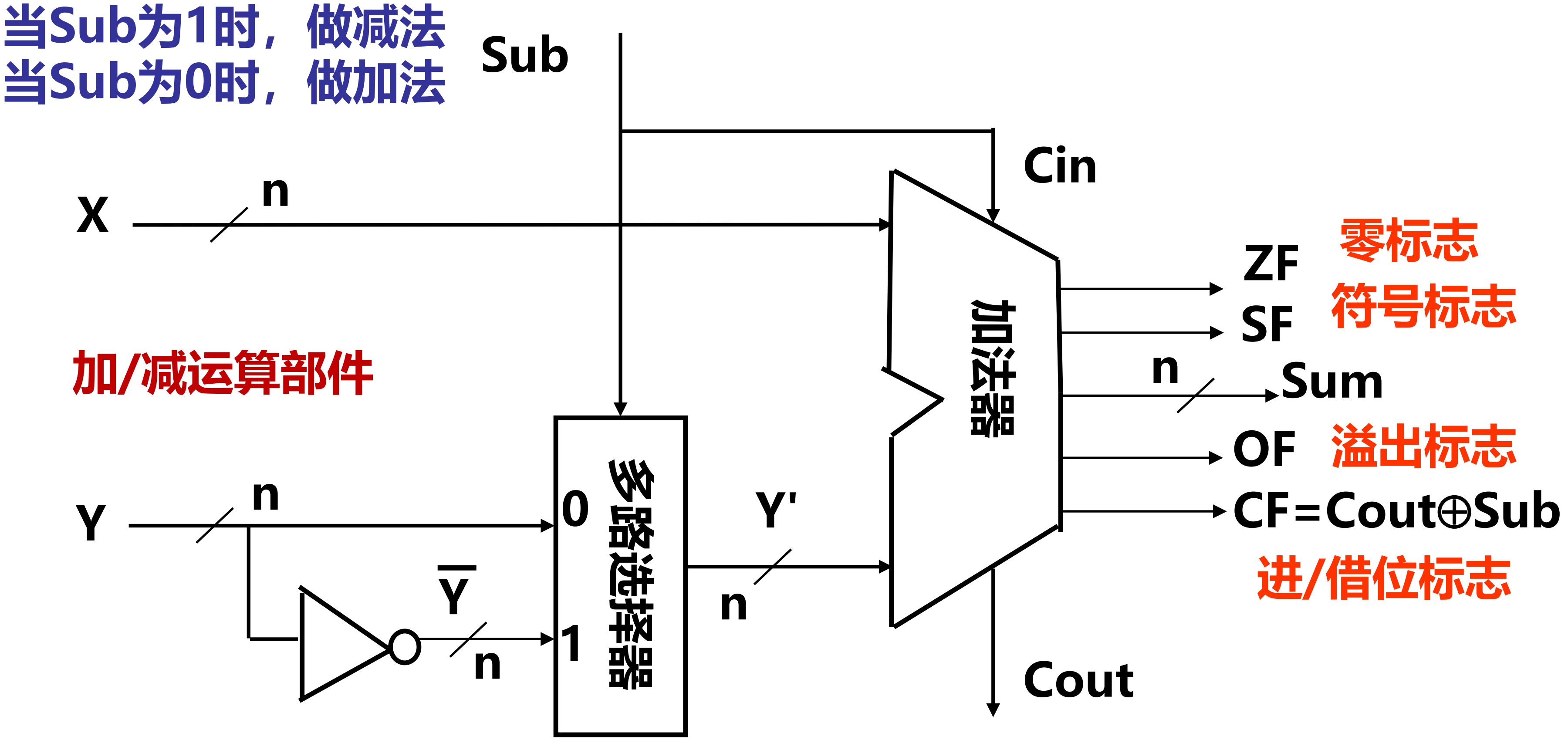


整数加减运算电路

补码加减运算公式：

$[x+y]_{补} = [x]_{补} + [y]_{补} \quad (\bmod 2^n)$

$[x-y]_{补} = [x]_{补} + [-y]_{补} \quad (\bmod 2^n)$



整数加减运算

```
#include "stdio.h"
void main( )
{   int   a=0x98765432,  b=0x87654321, c, d;
     unsigned int ua=0x98765432, ub=0x87654321, uc, ud;
     c=a+b;    uc=ua+ub;
     d=a-b;     ud=ua-ub;
     printf( “%d+(%d)=%d\n",a,b,c);
     printf("%u+%u=%u\n",ua,ub,uc);
     printf("%d-(%d)=%d\n",a,b,d);
     printf("%u-%u=%u\n",ua,ub,ud);
}
```

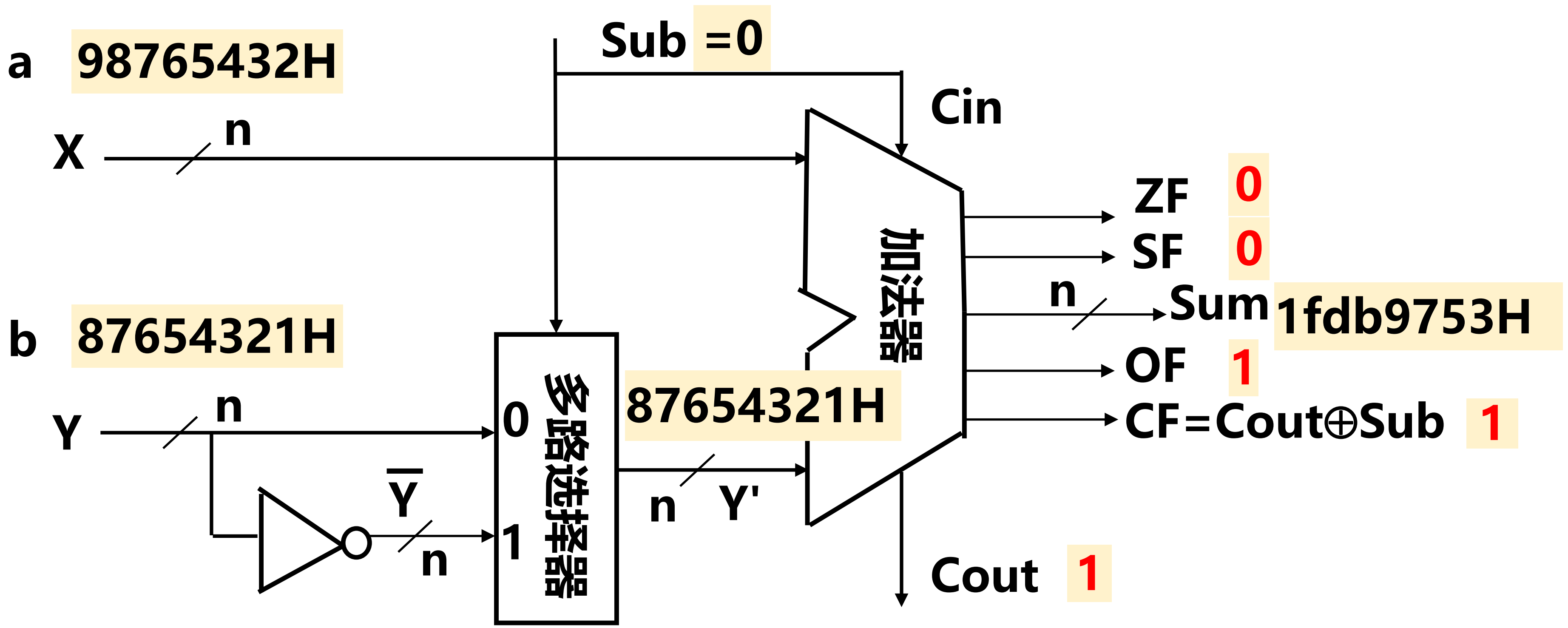
1. 运行程序，查看各变量的机器数。
2. 查看实现带符号整数加、减法运算和无符号整数加、减法运算的指令。
3. 在整数加减运算电路图上，分别标注出运算a+b、ua+ub、a-b和ua-ub时的输入和输出内容，以及加法器的输入内容 。
1. 每次加减运算后，计算标志位OF、SF、ZF和CF的值。

整数加减运算电路

```
  98765432H        1001 1000 0111 0110 0101 0100 0011 0010
 +87654321H       +1000 0111 0110 0101 0100 0011 0010 0001
 ----------       -----------------------------------------
 11fdb9753H       10001 1111 1101 1011 1001 0111 0101 0011
```

Cout=1

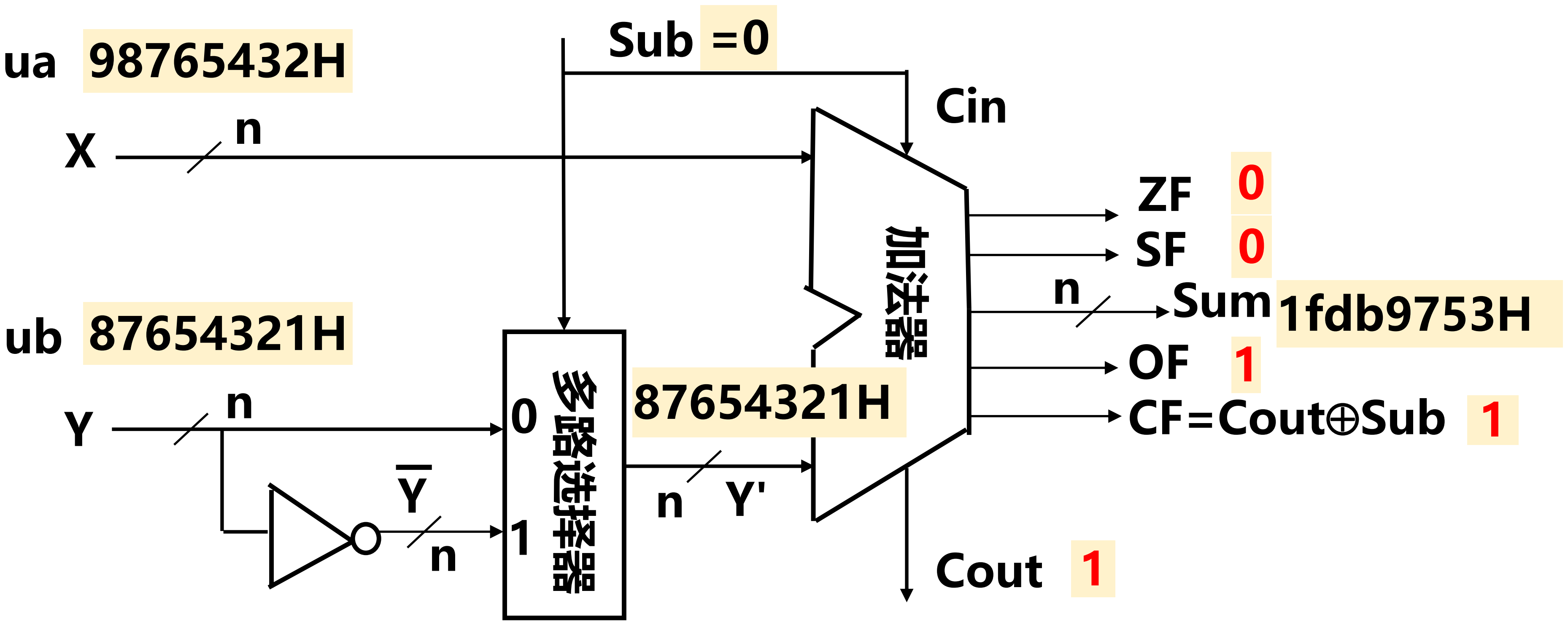
EFL=0000 0a07H =0……0 1010 0000 0111B



整数加减运算电路

```
  98765432H      1001 1000 0111 0110 0101 0100 0011 0010
 +87654321H     +1000 0111 0110 0101 0100 0011 0010 0001
 11fdb9753H     10001 1111 1101 1011 1001 0111 0101 0011
 Cout=1
```

EFL=0000 0a07H =0……0 1010 0000 0111B

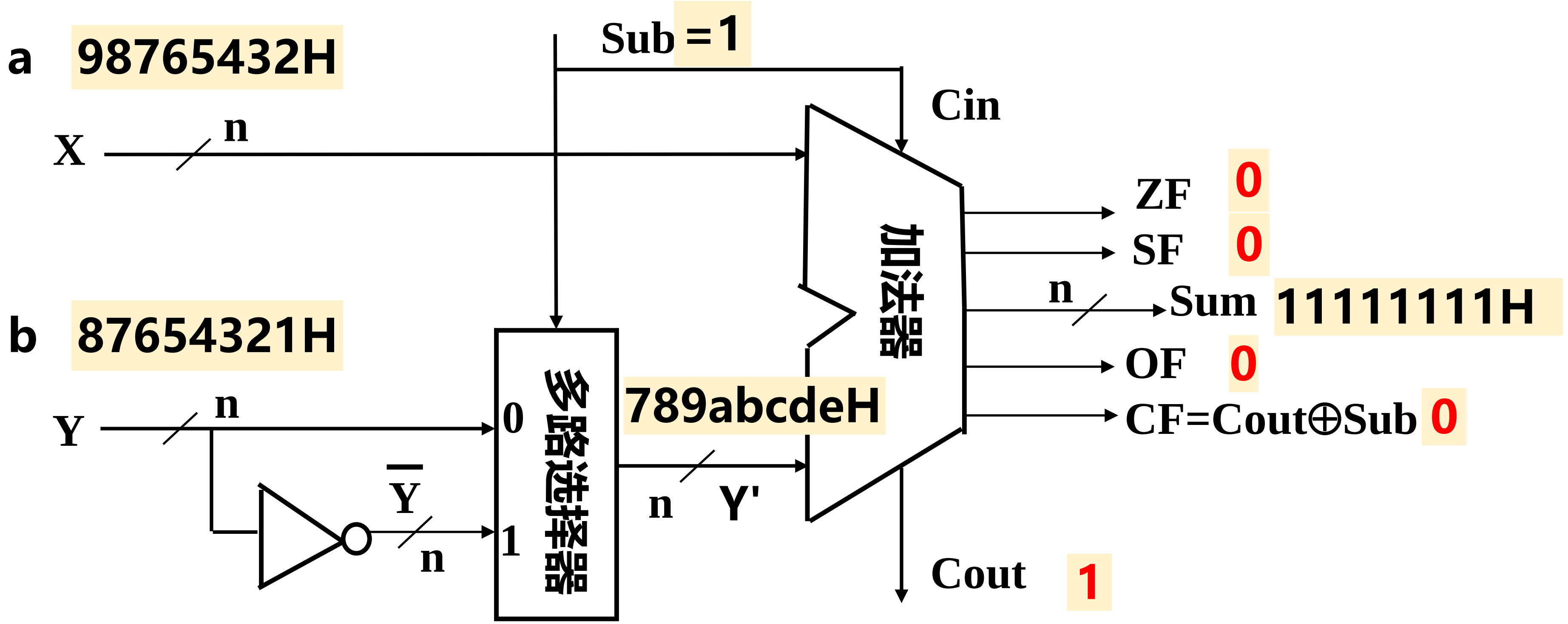


整数加减运算电路

98765432H	1001 1000 0111 0110 0101 0100 0011 0010
789abcdeH	0111 1000 1001 1010 1011 1100 1101 1110
+ 1	+ 1
11111 1111H	10001 0001 0001 0001 0001 0001 0001 0001

Cout=1

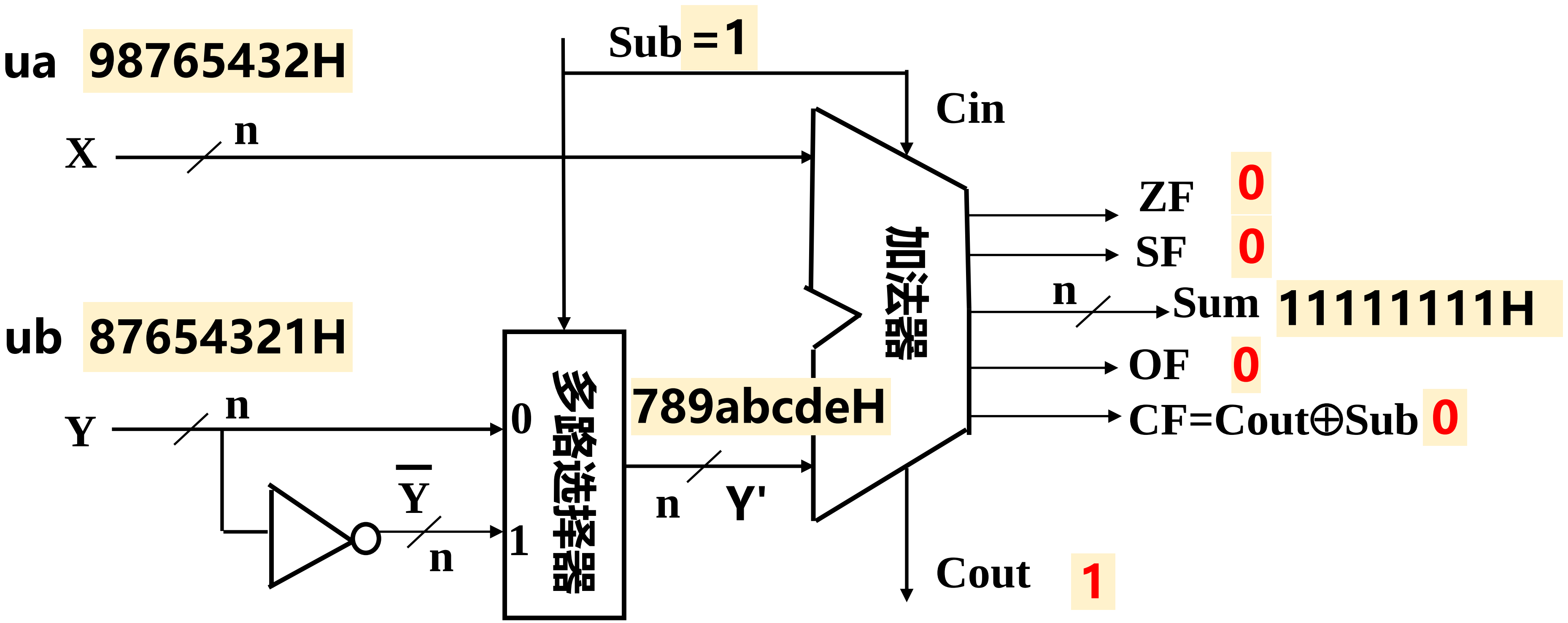
EFL=0000 0206H =0……0 0010 0000 0110B



整数加减运算电路

98765432H	1001 1000 0111 0110 0101 0100 0011 0010
789abcdeH	0111 1000 1001 1010 1011 1100 1101 1110
+ 1	+ 1
11111 1111H	10001 0001 0001 0001 0001 0001 0001 0001
Cout=1	

EFL=0000 0206H =0……0 0010 0000 0110B



整数加减运算电路

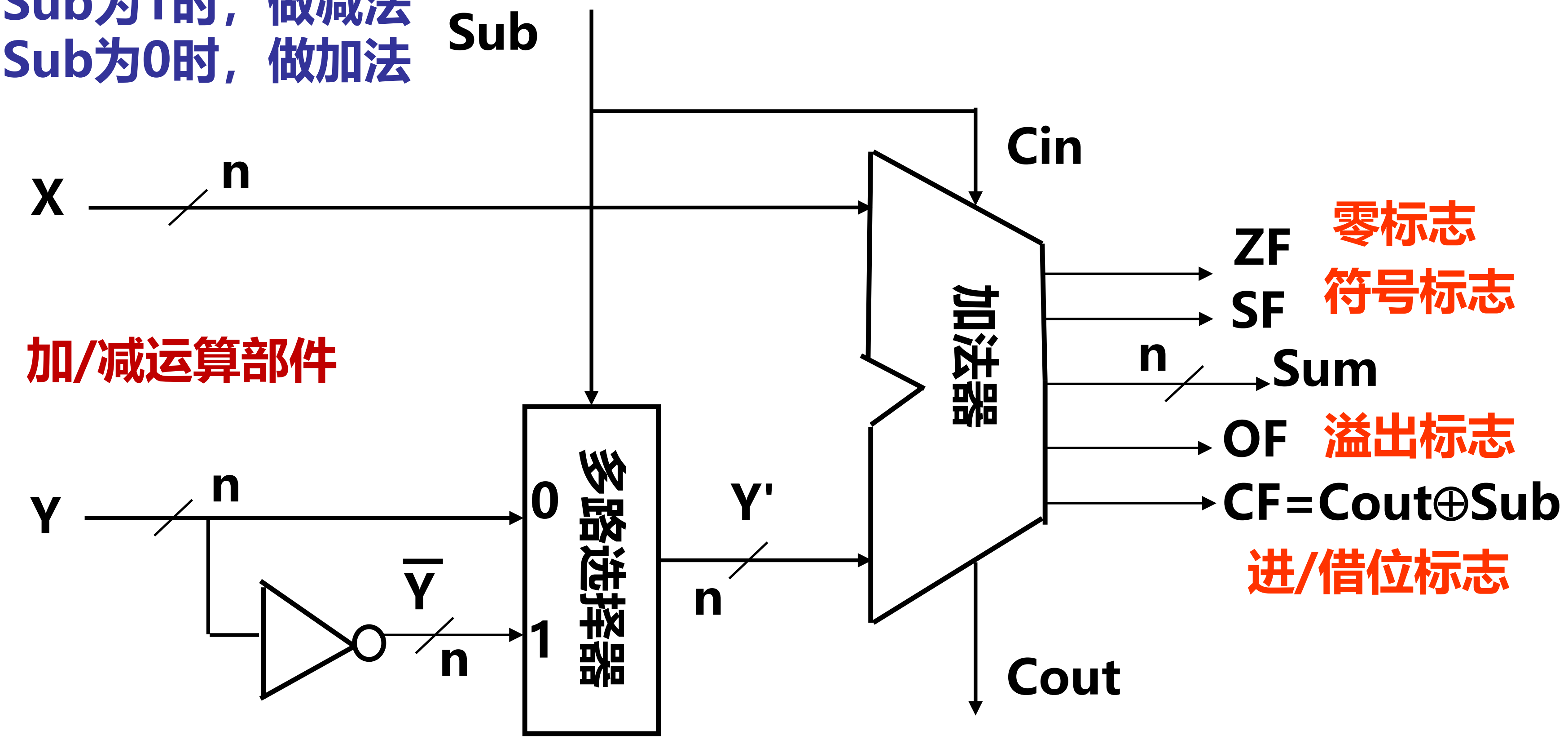
补码加减运算公式：

$[x+y]_{补} = [x]_{补} + [y]_{补} \quad (\bmod 2^n)$

$[x-y]_{补} = [x]_{补} + [-y]_{补} \quad (\bmod 2^n)$

当Sub为1时，做减法
当Sub为0时，做加法

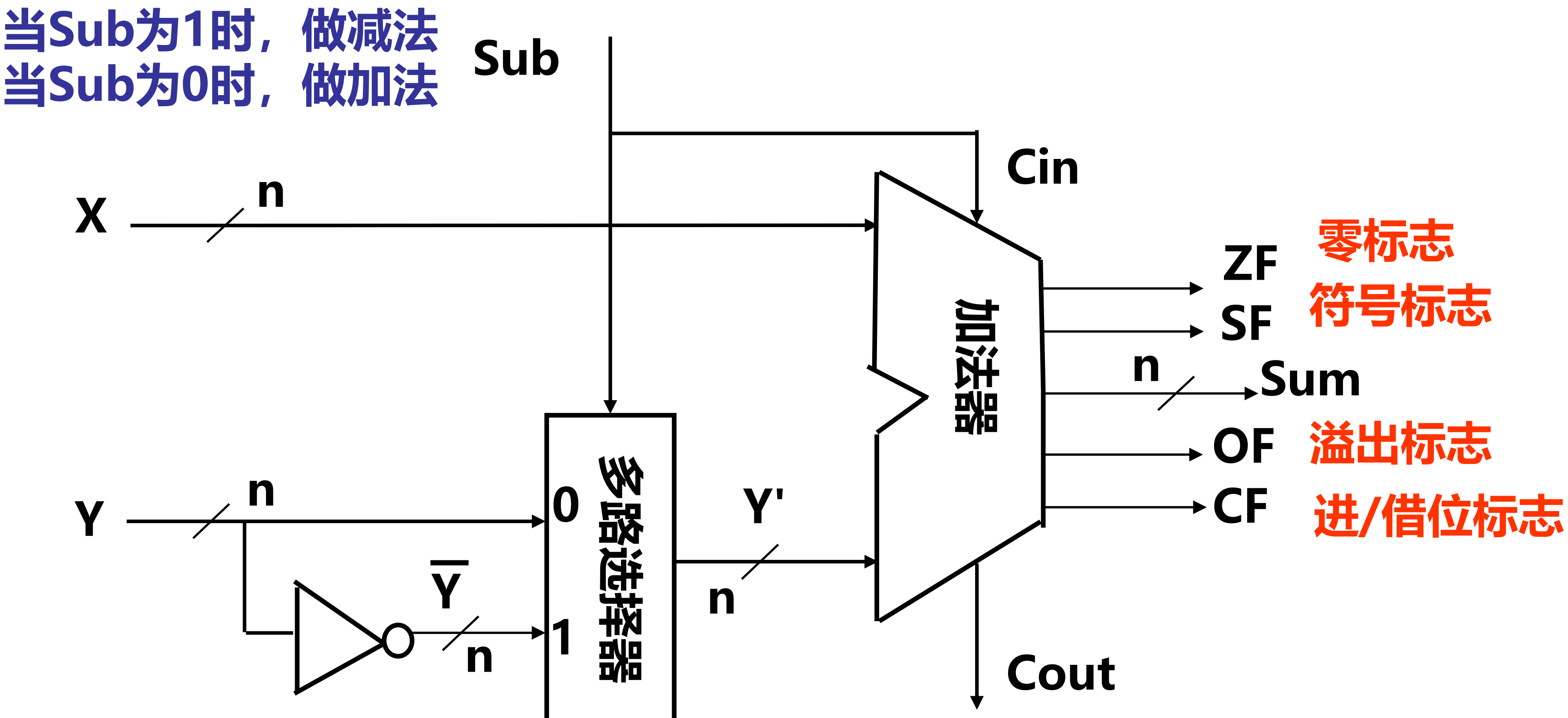
加/减运算部件



状态标志CF、ZF、SF和OF

$OF = X_{n-1}\, Y'_{n-1}\, \overline{Sum}_{\,n-1} + \overline{X}_{n-1}\, \overline{Y'}_{n-1}\, Sum_{\,n-1}$

$CF = Cout \oplus Sub$



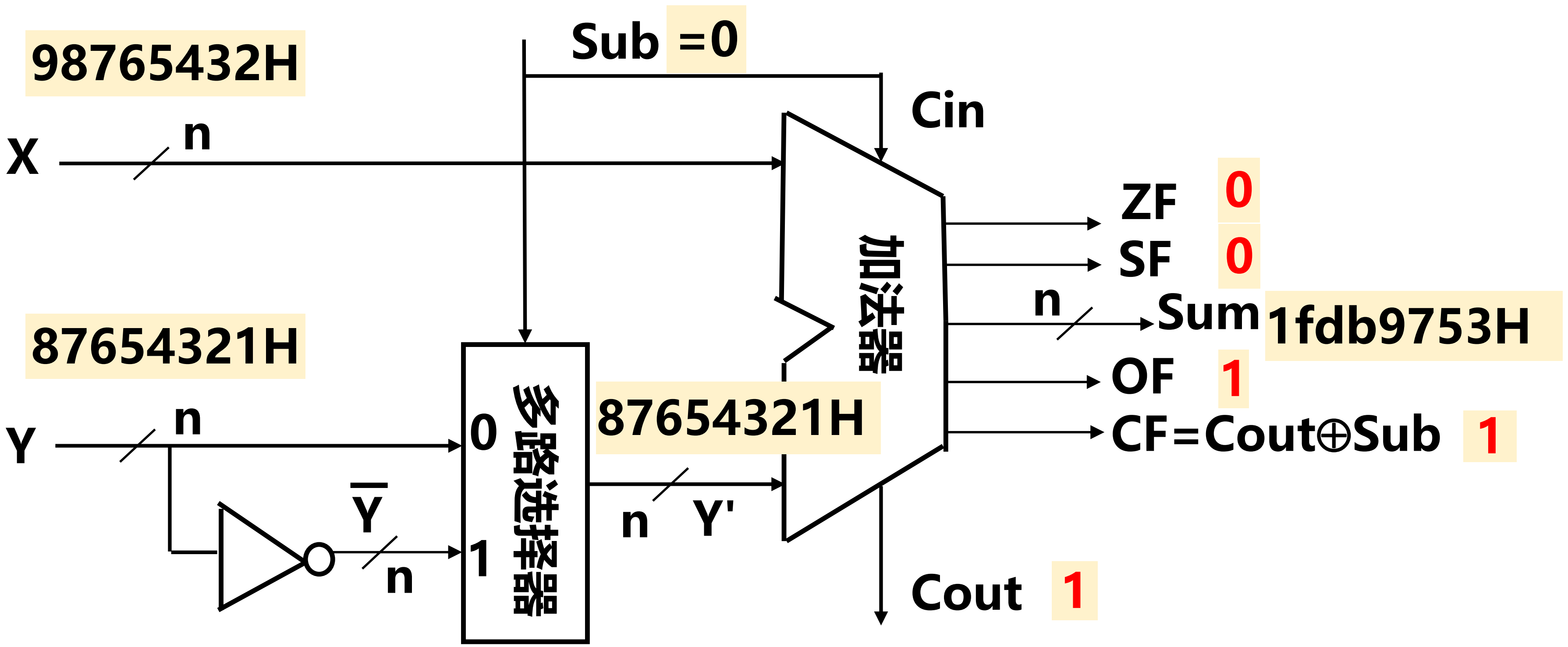
状态标志CF、ZF、SF和OF

98765432H	1001 1000 0111 0110 0101 0100 0011 0010
+87654321H	+1000 0111 0110 0101 0100 0011 0010 0001
11fdb9753H	10001 1111 1101 1011 1001 0111 0101 0011

Cout=1

$OF = X_{n-1}\, Y'_{n-1}\, \overline{Sum_{n-1}} + \overline{X_{n-1}}\, \overline{Y'_{n-1}}\, Sum_{n-1} = 1$

EFL=0000 0a07H =0……0 1010 0000 0111B



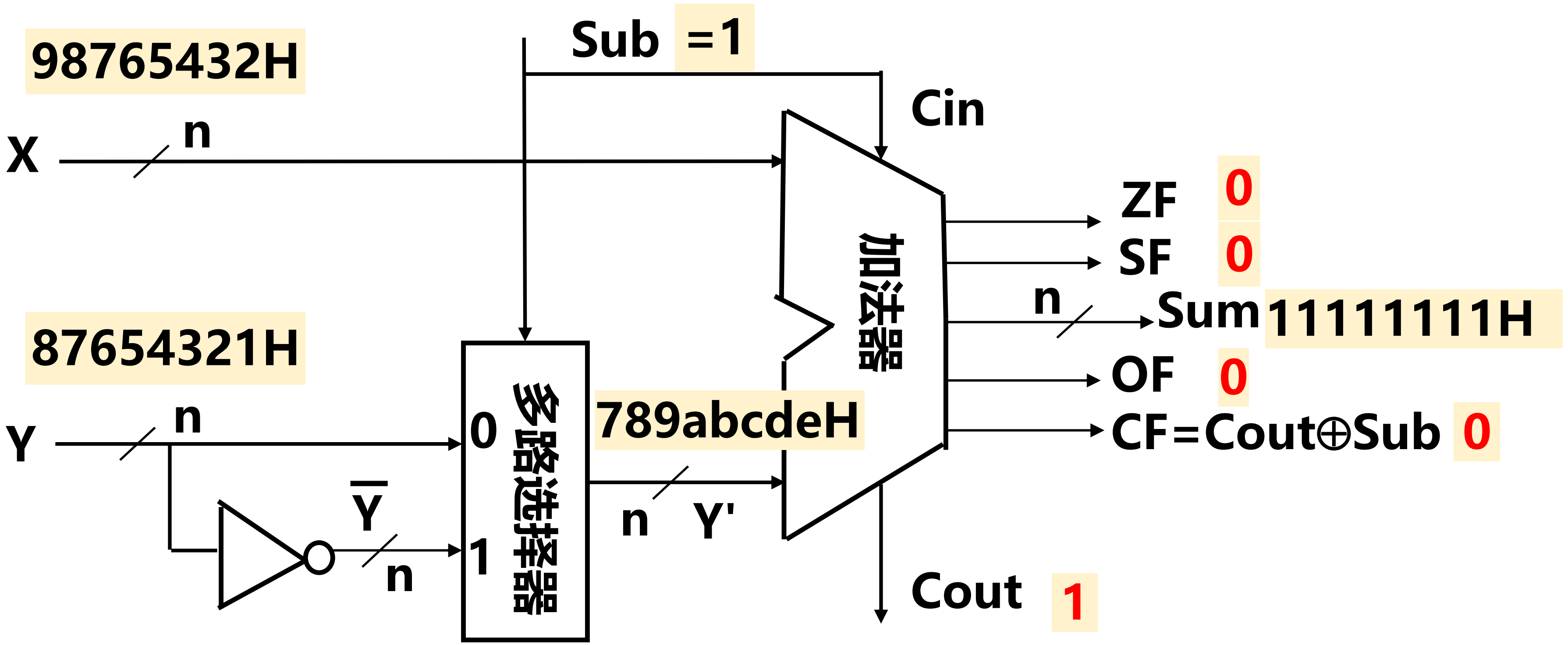
状态标志CF、ZF、SF和OF

98765432H	1001 1000 0111 0110 0101 0100 0011 0010
789abcdeH	0111 1000 1001 1010 1011 1100 1101 1110
+ 1	+ 1
11111 1111H	10001 0001 0001 0001 0001 0001 0001 0001

Cout=1

$$OF=X_{n-1}\,Y'_{n-1}\,\overline{Sum_{n-1}}+\overline{X_{n-1}}\;\overline{Y'_{n-1}}\,Sum_{n-1}=0$$

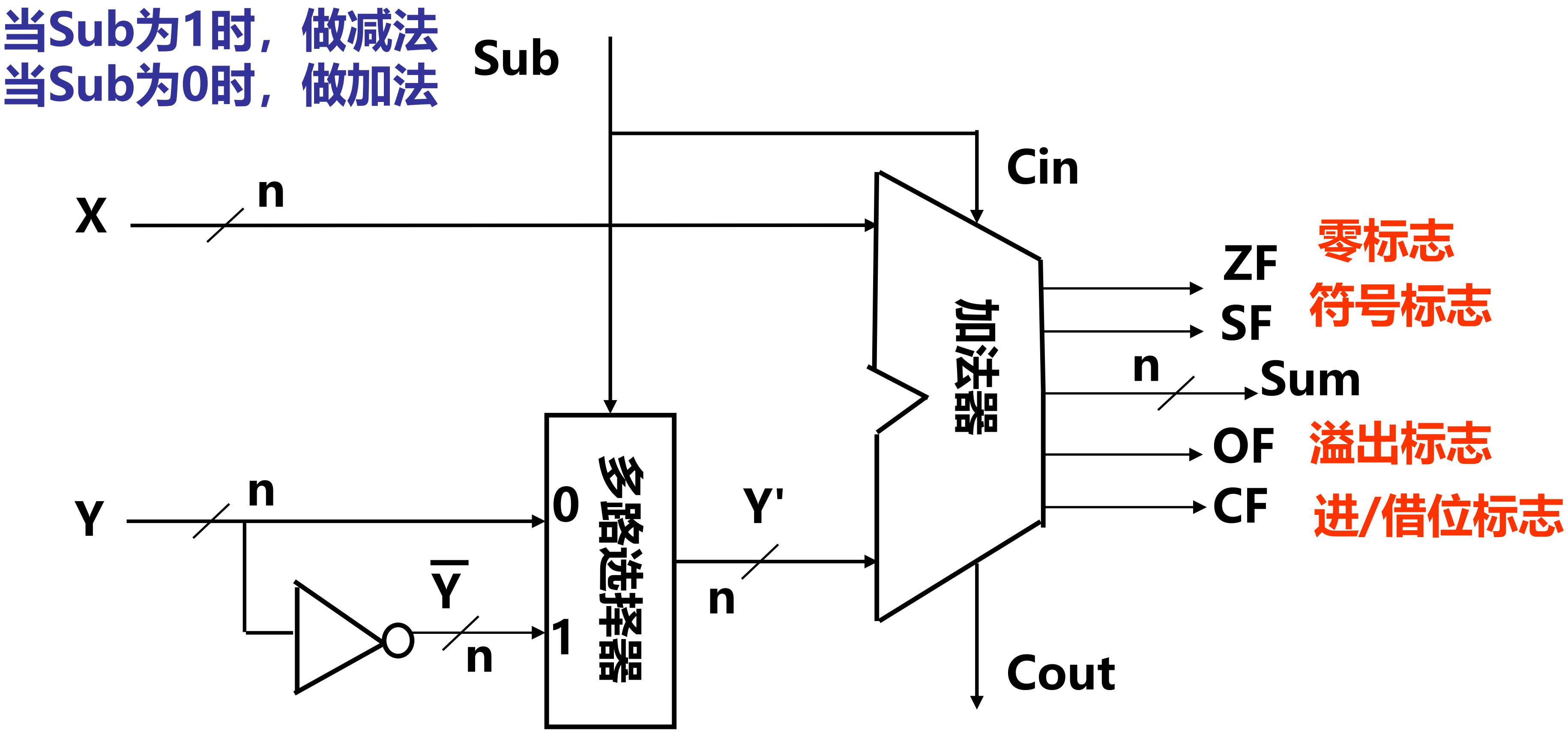
EFL=0000 0206H =0……0 0010 0000 0110B



状态标志CF、ZF、SF和OF

$$OF = X_{n-1}\,Y'_{n-1}\,\overline{Sum}_{\,n-1} + \overline{X}_{n-1}\,\overline{Y'}_{n-1}\,Sum_{\,n-1}$$

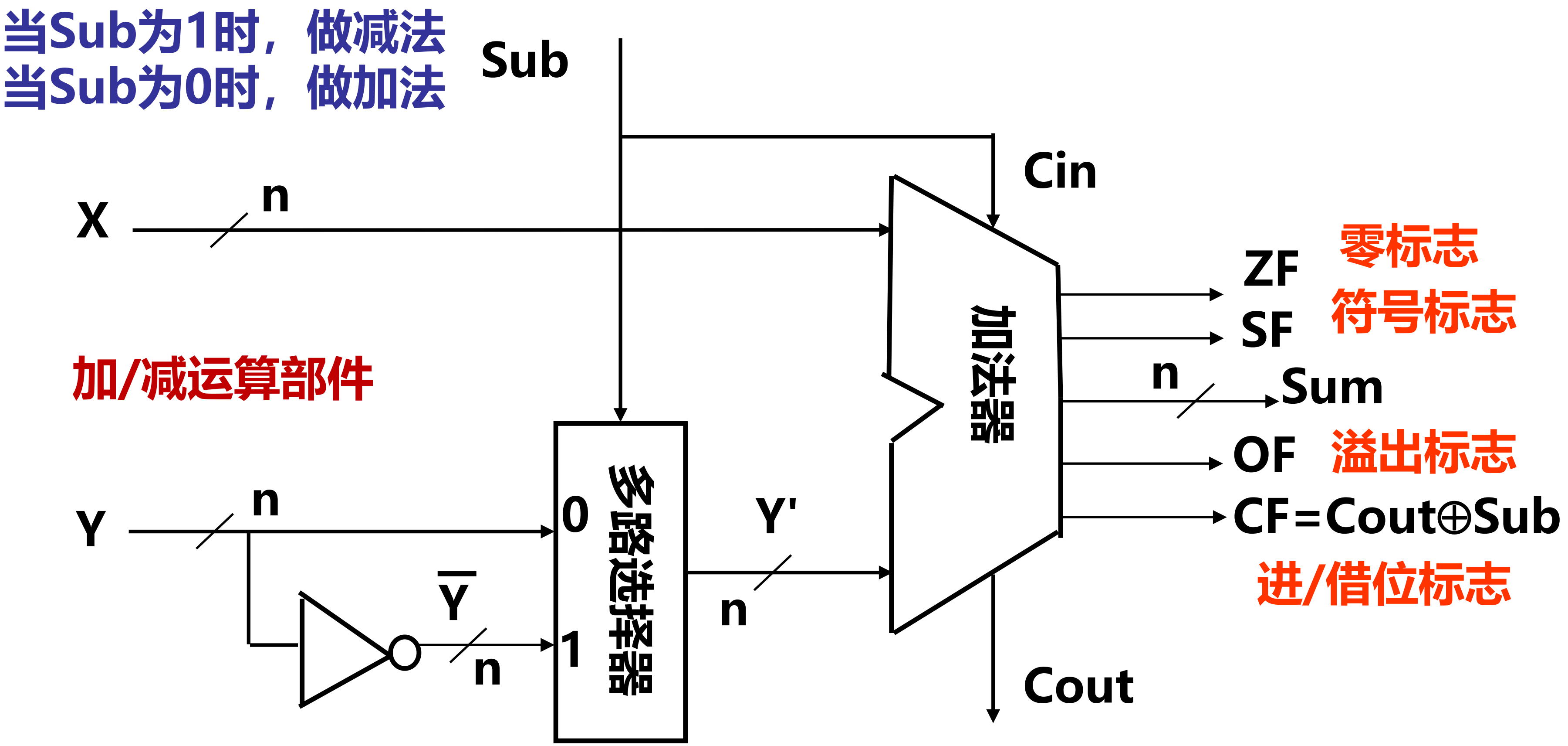
$$CF = Cout \oplus Sub$$



整数加减运算结果的溢出问题

无符号整数：用CF状态表示加减运算后是否有进位或借位，
OF值无意义。

带符号整数：用OF状态表示加减运算后结果是否溢出，
CF值无意义。



整数加减运算结果的溢出问题

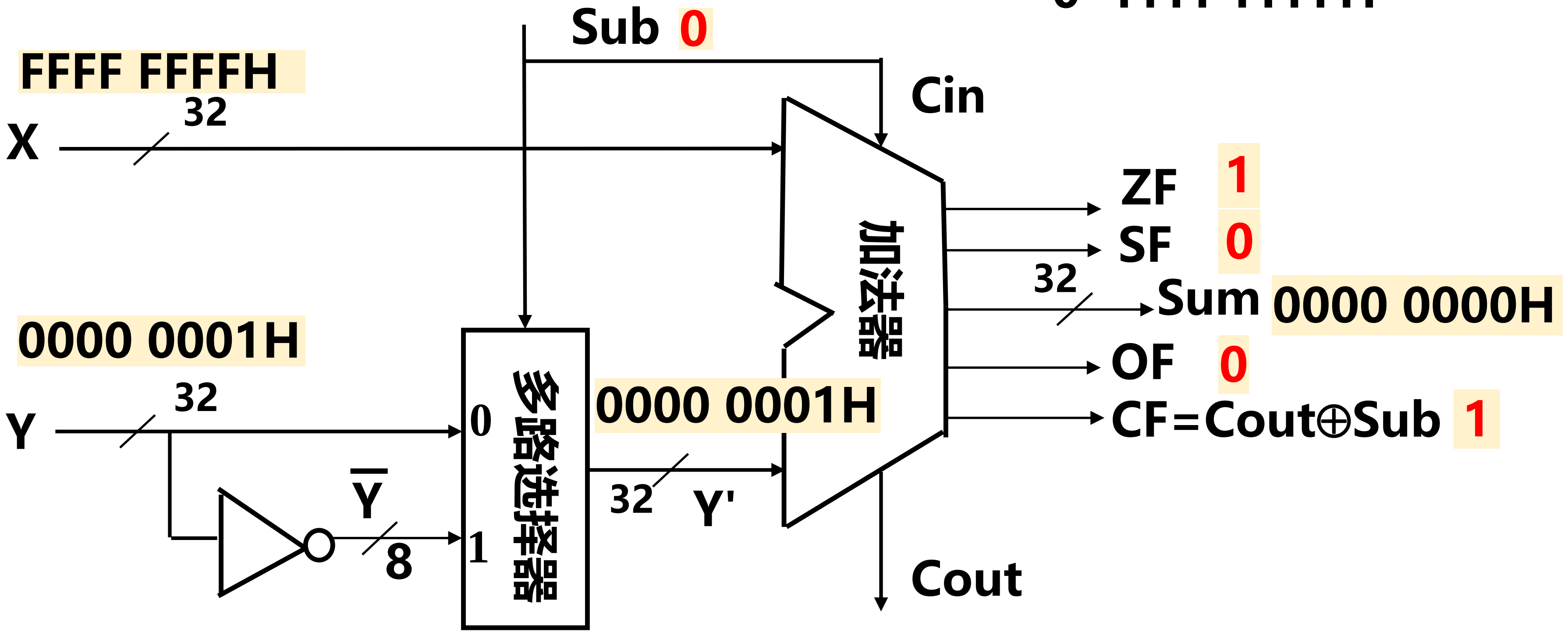
FFFF FFFFH	1111 1111 1111 1111 1111 1111 1111 1111
+ 1H	0000 0000 0000 0000 0000 0000 0000 0001
10000 0000H	1 0000 0000 0000 0000 0000 0000 0000 0000

Cout=1

CF=Cout⊕Sub=1

无符号整数加法运算中，用CF表示进位，CF=Cout

当n=32时，无符号整数的表示范围是:
0~FFFF FFFFH



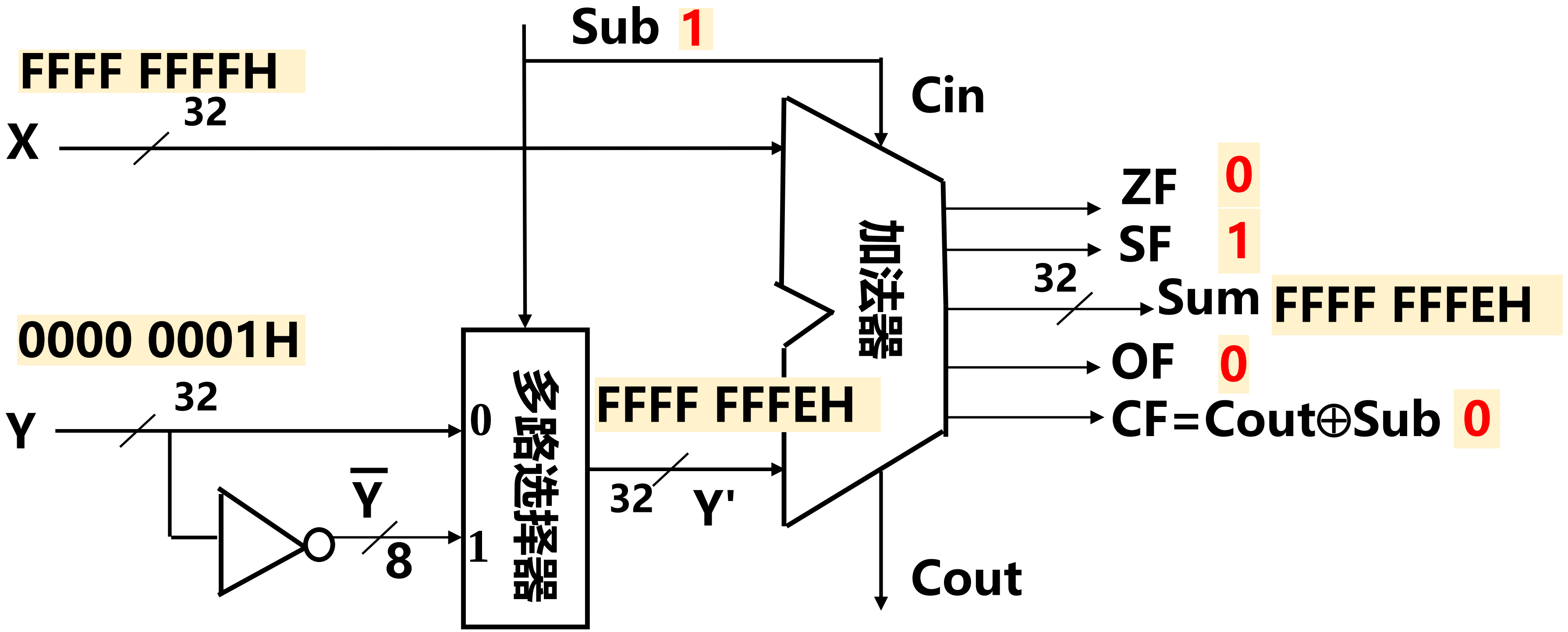
整数加减运算结果的溢出问题

```
  FFFF FFFFH        FFFF FFFFH
-         1H        FFFF FFFEH
-------------     +          1
0 FFFF FFFEH      ------------
                  1 FFFF FFFEH
```

FFFF FFFFH-1H
=FFFF FFFFH+（-1H+2^{32}）
因为够减，所以2^{32}没使用到，成为Cout值

Cout=1　　CF=Cout⊕Sub=0

无符号整数减法运算中，用CF表示借位，CF=$\overline{\text{Cout}}$



整数加减运算结果的溢出问题

```
  0000 0001H        0000 0001H
 -FFFF FFFFH        0000 0000H
 -----------      +          1
  FFFF FFFEH      ------------
                    0000 0002H
```

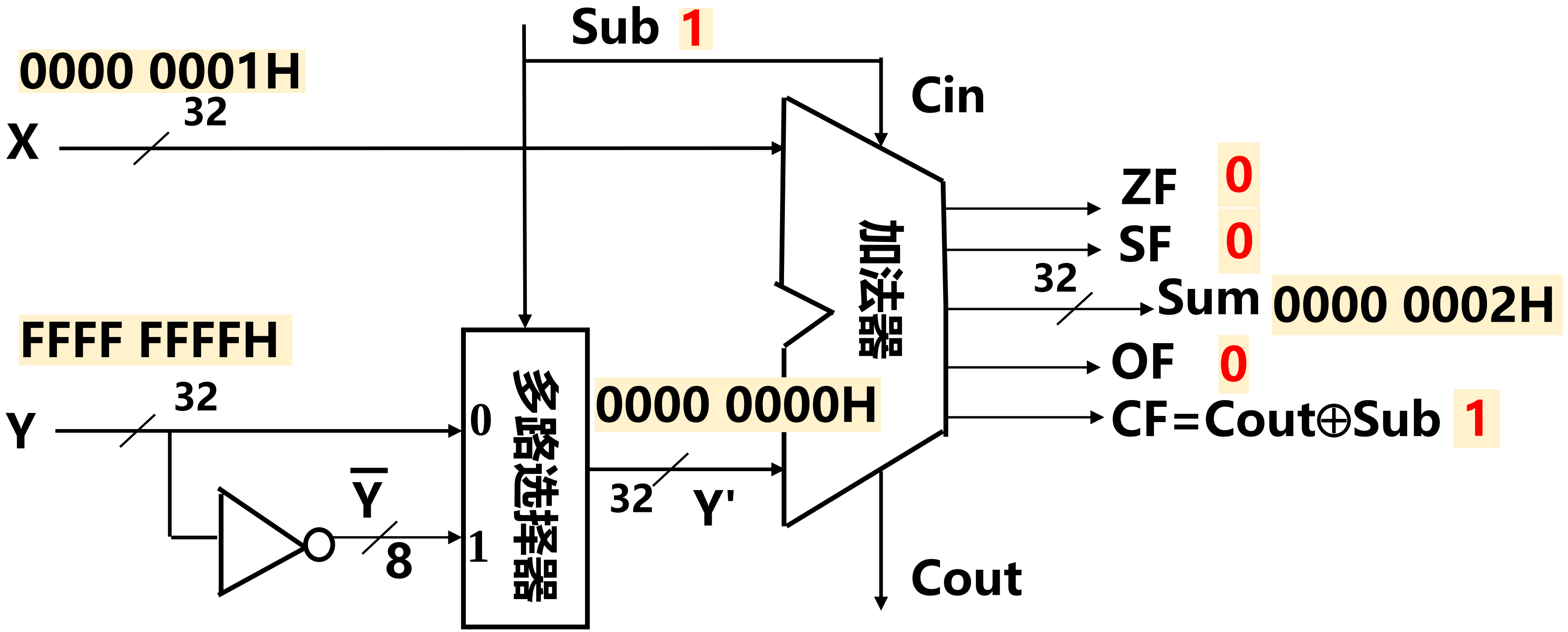
1H-FFFF FFFFH

=1H+（-FFFF FFFFH+2^{32}）

因为不够减，所以2^{32}被用于借位

Cout=0　　CF=Cout⊕Sub=1

无符号整数减法运算中，用CF表示借位，CF=$\overline{\text{Cout}}$



整数加减运算结果的溢出问题

无符号整数加减运算的总结：

无符号整数的加法运算中，CF表示进位， $CF = Cout \oplus \overset{0}{Sub} = Cout$

无符号整数的减法运算中，CF表示借位， $CF = Cout \oplus \underset{1}{Sub} = \overline{Cout}$

CF对带符号整数的运算无意义。

当A和B都是无符号整数时，标志位信息的应用

	CF	ZF	说明
A-B		1	A=B
A-B	1	0	A<B
A-B	0	0	A>B

整数加减运算结果的溢出问题

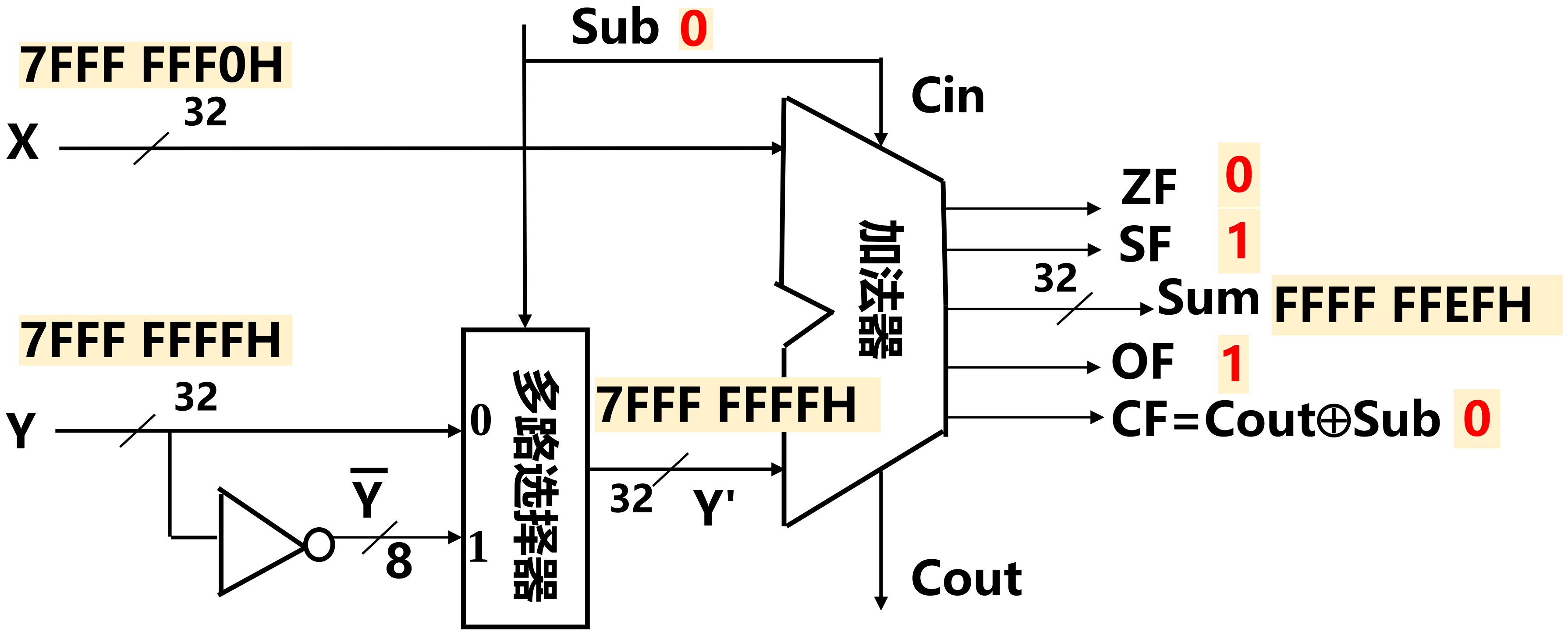
7FFF FFF0H	0111 1111 1111 1111 1111 1111 1111 0000
+7FFFFFFFH	+0111 1111 1111 1111 1111 1111 1111 1111
FFFF FFEFH	1111 1111 1111 1111 1111 1111 1110 1111

$$OF=\overline{X_{n-1}}\,\overline{Y'_{n-1}}Sum_{n-1}+X_{n-1}Y'_{n-1}\overline{Sum_{n-1}}=1$$

带符号整数加法运算中，用OF表示溢出。

int类型整数的表示范围是：

-0x8000 0000~0x7FFF FFFF



整数加减运算结果的溢出问题

(-7FFF FFF0H)-7FFF FFFFH

8000 0010H	1000 0000 0000 0000 0000 0000 0001 0000
8000 0000H	1000 0000 0000 0000 0000 0000 0000 0000
+ 1	+ 1
1 0000 0011H	10000 0000 0000 0000 0000 0000 0001 0001

$OF=\overline{X_{n-1}}\,\overline{Y'_{n-1}}Sum_{n-1}+X_{n-1}Y'_{n-1}\overline{Sum_{n-1}}=1$

带符号整数加法运算中，用OF表示溢出。

int类型整数的表示范围是：

-0x8000 0000~0x7FFF FFFF

(-7FFF FFF0H)-7FFF FFFFH 结果的补码应该是：-(7FFF FFFFH+7FFF FFFFH)+2^{32}

但加法器上计算的是：

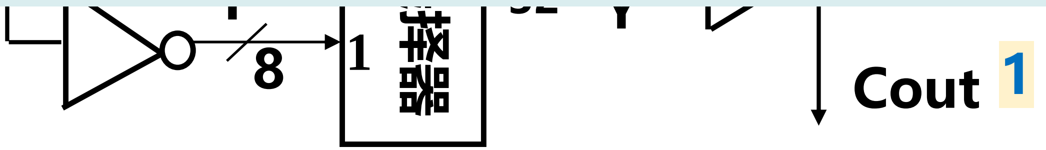
(-7FFF FFF0H+2^{32})+(-7FFF FFFFH +2^{32})=-(7FFF FFFFH+7FFF FFFFH)+2^{32}+2^{32}

带符号整数运算时，Cout值与溢出无关，与运算的结果也无关

-8000 0000H +2^{32}=8000 0000H

-(7FFF FFFFH+7FFF FFFFH) <-8000 0000H 时，超出了最小负数表示范围

-(7FFF FFFFH+7FFF FFFFH) +2^{32}=0 •••••••B



整数加减运算结果的溢出问题

带符号整数加减运算总结：

1. 带符号整数的加减法运算中，用OF判断溢出

   $$OF=\overline{X_{n-1}}\,\overline{Y'_{n-1}}\,Sum_{n-1}+X_{n-1}\,Y'_{n-1}\,\overline{Sum_{n-1}}=1$$

2. 从加减运算的角度来看：

   加法时，同号相加，和与两加数异号，则溢出；

   减法时，异号相减，差与被减数异号，则溢出。

当A和B都是带符号整数时，标志位信息的应用

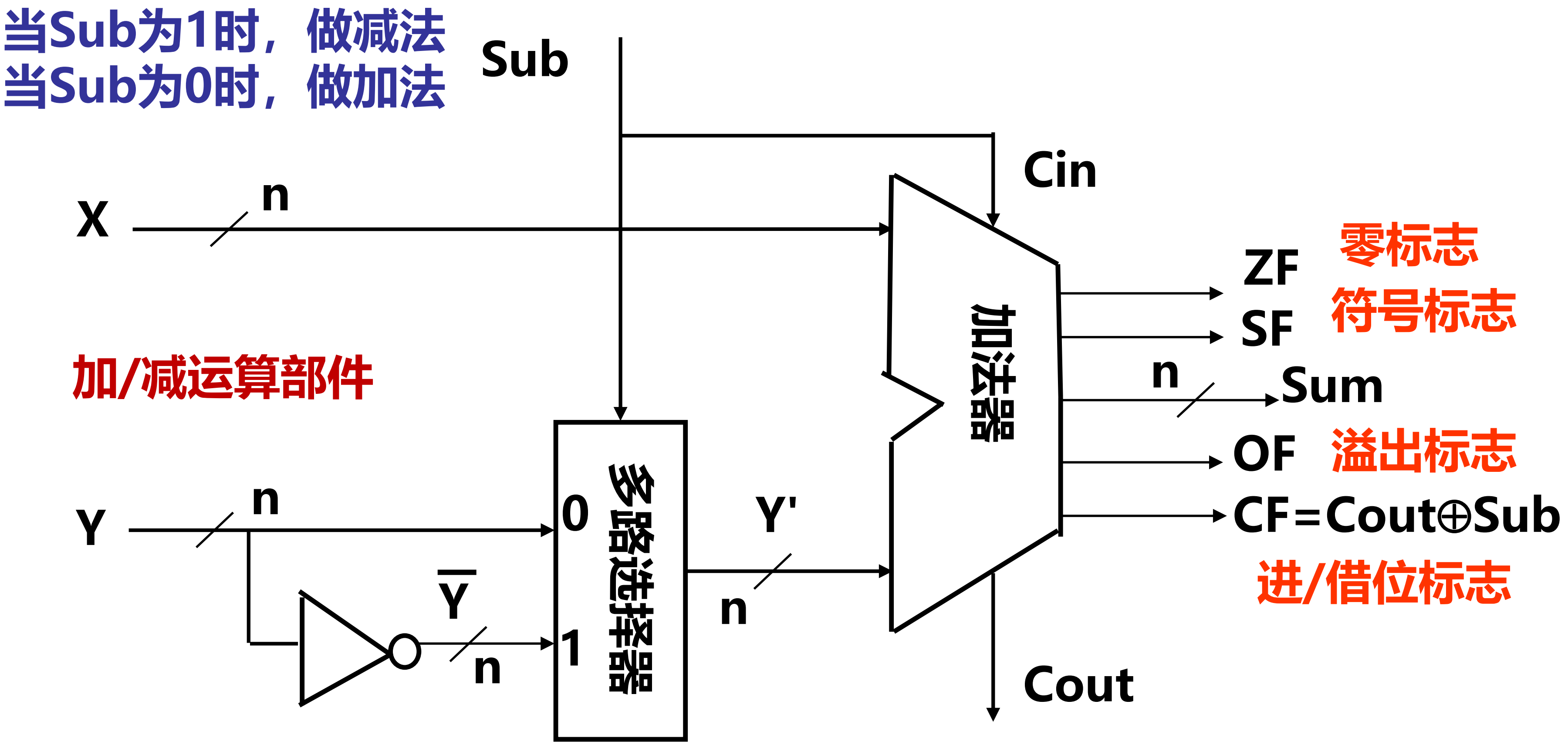
	SF	OF	ZF	说明
A-B			1	A=B
A-B	1	0	0	A<B
A-B	1	1	0	A>B
A-B	0	0	0	A>B
A-B	0	1	0	A<B
A-B	SF!=OF and ZF==0			A<B
A-B	SF!=OF OR ZF==1			A<=B
A-B	SF==OF and ZF==0			A>B
A-B	SF==OF OR ZF==1			A>=B

总 结

补码加减运算公式：

$$[x+y]_{补} = [x]_{补} + [y]_{补} \quad (\bmod 2^n)$$

$$[x-y]_{补} = [x]_{补} + [-y]_{补} \quad (\bmod 2^n)$$



谢谢！

《计算机系统基础（四）：编程与调试实践》

浮点数的表示和基本运算

浮点数的表示和基本运算

IEEE 754浮点数标准

尾数的舍入处理

浮点数的基本运算

IEEE 754浮点数标准

1. IEEE 754浮点数的基本格式

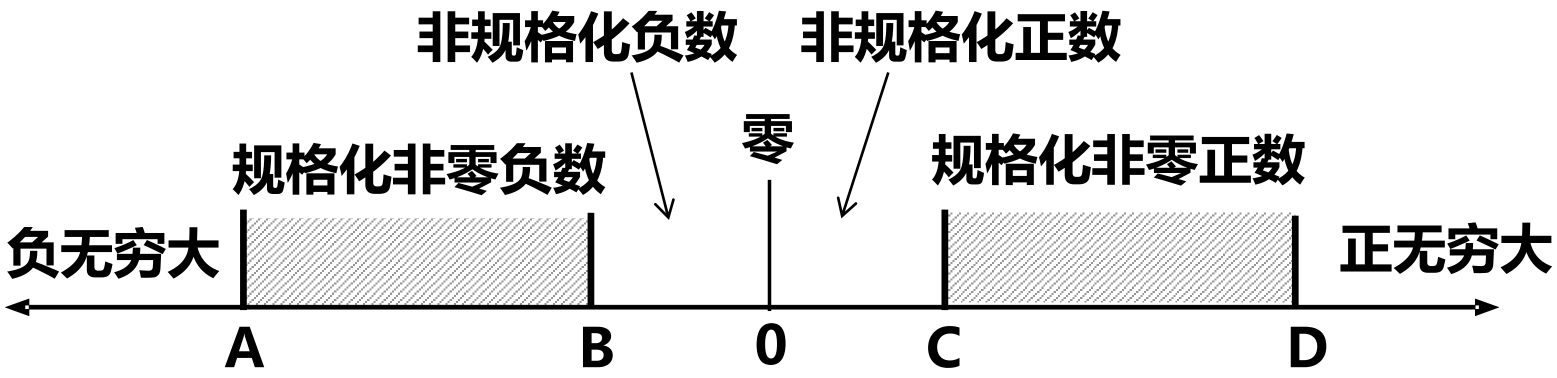
① 32位单精度浮点格式，即float格式

1位	8位	23位
符号	阶码	尾数

② 64位双精度浮点格式，即double格式

1位	11位	52位
符号	阶码	尾数

2. IEEE 754标准中的数据按值的分类



IEEE 754浮点数标准

IEEE 754浮点数32位单精度格式各类值的编码

	1位	8位	23位
正零和负零	0/1	0	0
非规格化	0/1	0	f≠0
规格化非零	0/1	1-254	f
无穷大	0/1	255	0
无定义数	0/1	255	f≠0

IEEE 754浮点数标准

规格化数的真值与机器数的对应关系：

真值： $+/-1.xxxxxxxxxx * 2^{E}$

机器数：

1 bit	8 bits	23 bits
0/1	E+127	xxxxxxxxxxx

例如

真值：5.0=1.01B*2^2　　2+127=1000 0001B

机器数：0 1000 0001 010 0000 0000 0000 0000 0000B

=40a0 0000H

机器数：

1 bit	8 bits	23 bits
S	Exponent	Significand

真值： $(-1)^{S} \times (1 + \text{Significand}) * 2^{(\text{Exponent}-127)}$

例如

机器数：40a0 0000H

=0 1000 0001 010 0000 0000 0000 0000 0000B

真值：1.01B*2^2　　1000 0001B-127=2

IEEE 754浮点数标准

非规格化数的真值与机器数的对应关系：

真值： $+/-0.xxxxxxxxxxx * 2^{-126}$

机器数：

1 bit	8 bits	23 bits
0/1	00000000	xxxxxxxxxxx

例如

真值： $1e\text{-}40=10^{-40}\approx 0.000\ 0001\ 0001\ 0110\ 1100\ 0010B*2^{-126}$

机器数： 0 0000 0000 000 0001 0001 0110 1100 0010B

=0001 16C2H

机器数：

1 bit	8 bits	23 bits
S	00000000	Significand

真值： $(-1)^S \times Significand * 2^{-126}$

例如

机器数： 0001 16C2H

=0 0000 0000 000 0001 0001 0110 1100 0010B

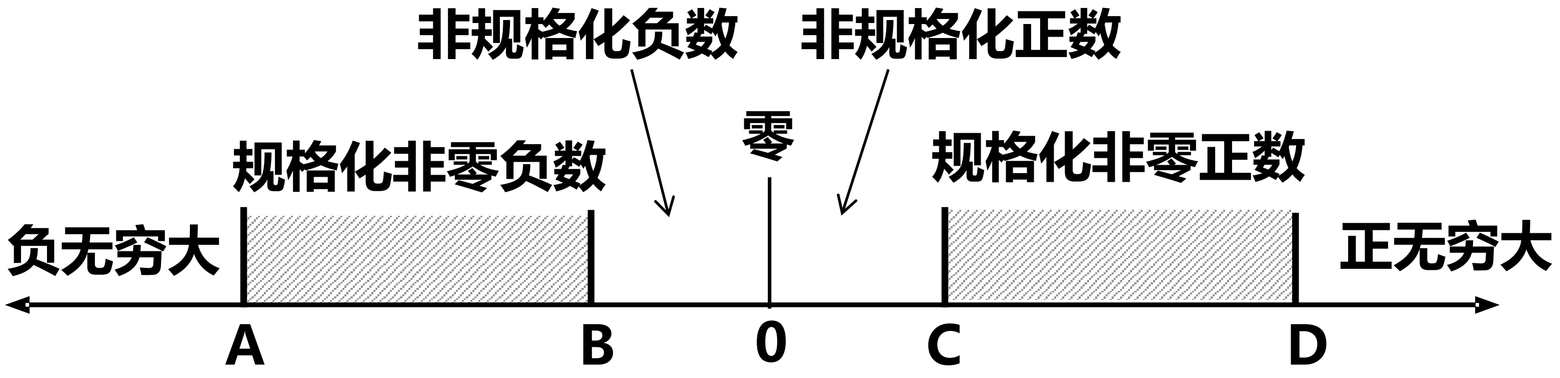
真值： $0.000\ 0001\ 0001\ 0110\ 1100\ 0010B*2^{-126}$

尾数的舍入处理

32位单精度浮点格式，即float格式

1位	8位	23位
符号	阶码	尾数

24位精度

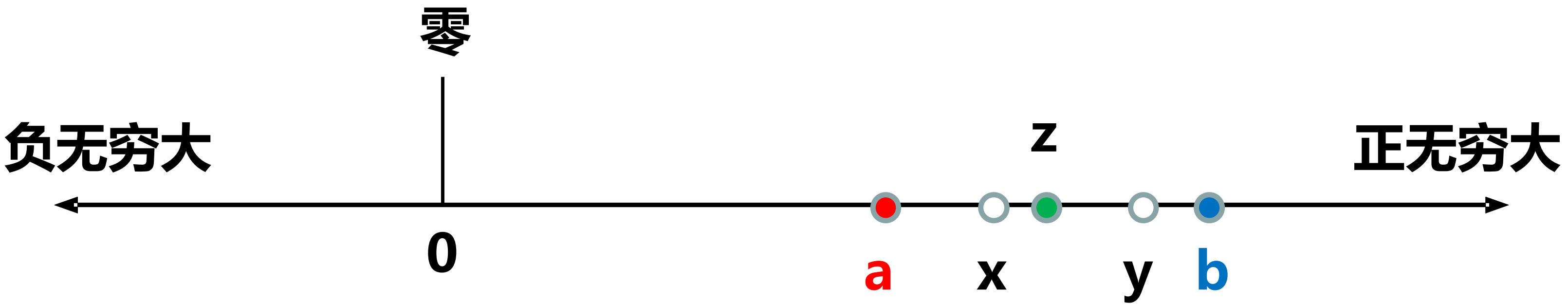


0.1=0.000 1100 1100 1100 1100 1100 1100 1100 ·····B

尾数的舍入处理

IEEE754标准提供四种舍入模式：

①	就近舍入（中间值舍入到偶数）	x≈a、y≈b、z≈?
②	朝+∞方向舍入	x≈b、y≈b、z≈b
③	朝-∞方向舍入	x≈a、y≈a、z≈a
④	朝0方向舍入	x≈a、y≈a、z≈a



a、 b： 两个连续的浮点格式可表示的数据

x、y、z：需要浮点编码的数据，z=(a+b)/2

尾数的舍入处理

就近舍入方法（以32位单精度浮点格式为例）：

真值的尾数:

$1.x_1x_2 \cdots\cdots\cdots\cdots\cdots\cdots x_{n-1}x_nB$　　假设 n>23

$= 1.\ x_1x_2\cdots x_{23}x_{24}x_{25}x_{26}\cdots x_nB$　　粘位处理

$= \begin{cases} 1.\ x_1x_2\cdots x_{23}x_{24}x_{25}0\ B & \text{若}x_{26}\cdots x_n\text{全为0} \\ 1.\ x_1x_2\cdots x_{23}x_{24}x_{25}1\ B & \text{若}x_{26}\cdots x_n\text{不全为0} \end{cases}$

需要截断的位只有3位了，有8种编码000~111，把100看成是中间值

① 000~011　小于100，舍，$1.\ x_1x_2\cdots x_{23}$

② 101~111　大于100，入，$1.\ x_1x_2\cdots x_{23} + 0.0\cdots01$　（最低位加1）

③ 100
- 若$x_{23}=0$，则　舍，$1.\ x_1x_2\cdots x_{23}$
- 若$x_{23}=1$，则　入，$1.\ x_1x_2\cdots x_{23} + 0.0\cdots01$　（最低位加1）

尾数的舍入处理

就近舍入方法（以32位单精度浮点格式为例）：

数据	尾数	“粘位”处理	舍入规则	新尾数
8000000H	1.…00 0000B	1.…00 000B	就近舍	1.…00 B
8000001H	1.…00 0001B	1.…00 001B	就近舍	1.…00 B
8000014H	1.…01 0100B	1.…01 010B	就近舍	1.…01 B
8000017H	1.…01 0111B	1.…01 011B	就近舍	1.…01 B
8000008H	1.…00 1000B	1.…00 100B	中间数，舍	1.…00 B
8000018H	1.…01 1000B	1.…01 100B	中间数，入	1.…10 B
8000019H	1.…01 1001B	1.…01 101B	就近入	1.…10 B
800000CH	1.…00 1100B	1.…00 110B	就近入	1.…01 B
800000DH	1.…00 1101B	1.…00 111B	就近入	1.…01 B

浮点数的基本运算

1. 设两个规格化浮点数分别为 $A=M_a \cdot 2^{E_a}$ $B=M_b \cdot 2^{E_b}$,则：

$A \pm B = (M_a \pm M_b \cdot 2^{-(E_a-E_b)}) \cdot 2^{E_a}$ （假设Ea>=Eb）

$A * B = (M_a * M_b) \cdot 2^{E_a+E_b}$

$A / B = (M_a / M_b) \cdot 2^{E_a-E_b}$

2. 浮点运算部件

早期：
- 浮点协处理器芯片（FPU）：8087、80287
- CPU：8086/8088、80286/80386

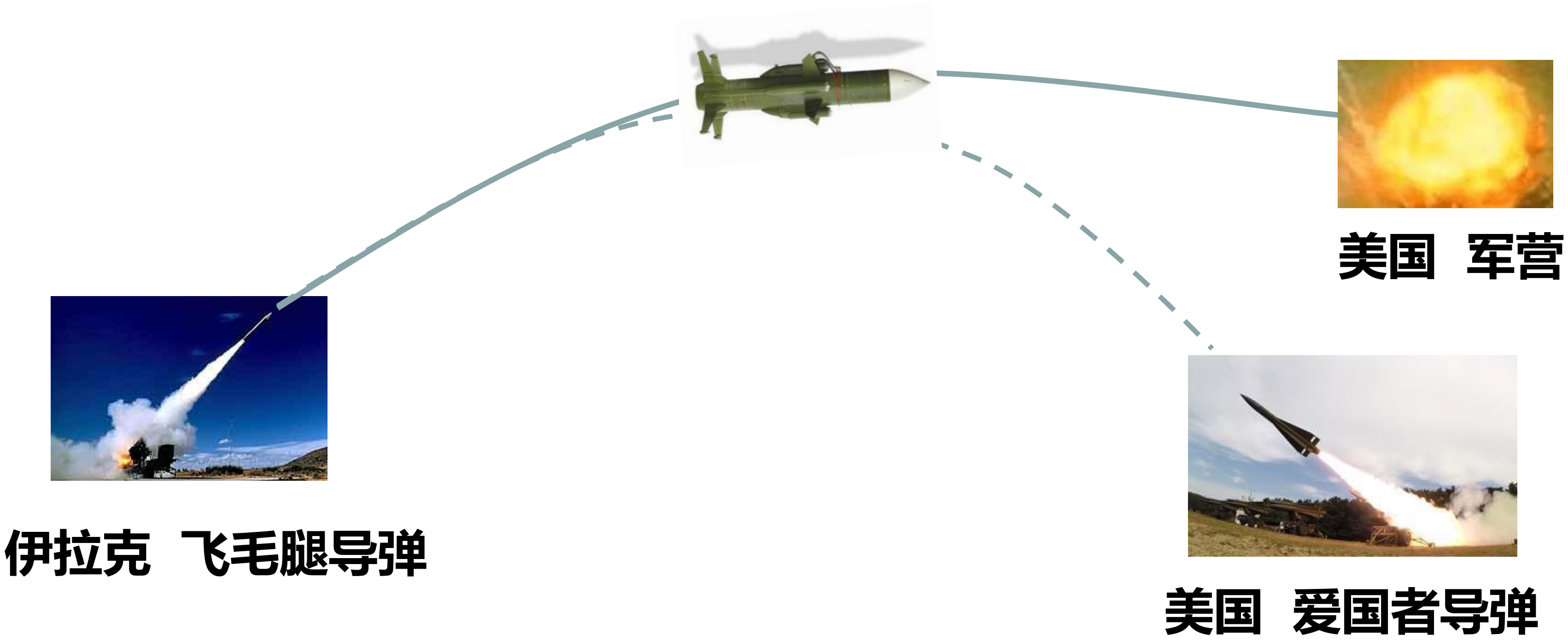
现在：CPU
- 定点运算部件
- 浮点运算部件

3. 浮点数的运算中有对阶、舍入、溢出等问题，导致运算结果会出现大数吃小数、精度误差、结果异常等问题。

4. 爱国者导弹定位错误的案例分析：差之毫厘，失之千里

浮点数的基本运算

(1) 事故：爱国者导弹定位错误。



(2) 原因：0.1的计算机表示误差

0.1的误差很小，但运算后的累计误差就大了。

浮点数的基本运算

(3) 数据：

爱国者导弹系统的内置时钟，每隔0.1秒计数一次；

爱国者已经连续工作100小时；

飞毛腿导弹的飞行速度约为2000米/秒；

(4)爱国者系统时钟的误差导致计算的距离偏差是多少?

(5) 分析：

0.1=0.000 1100 1100 1100 1100 1100 [1100]•••B

程序：x=0.000 1100 1100 1100 1100 1100B　24位定点小数表示0.1

时间误差：(0.1-x)*100*60*60*10≈0.3433秒

距离误差：2000*0.3433=686.6米

(6) 讨论：

对 0.1采用不同的表示方式，计算的距离误差分别是多少?

float格式、32位定点小数、就近舍入后的24位定点小数

谢谢！