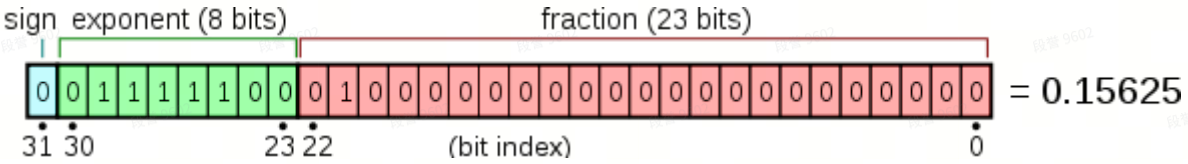
fp32 vs fp16 vs fp8

float32

单精度，4 byte， 32bit



1. sign：符号位，1代表负数， 0代表正数。
2. 指数部分，8个比特位， 全0和全1有特殊用途，所以是00000001~11111110， 也就是1到254， 减去偏置127，指数部分最终范围为-126 ~127。
3. 小数部分， 23个比特位，范围为 0 ~ $(2^{31}-1)/2^{31}$

所以一个最终的数据计算方式为：

$$(-1)^{sign} * 2^{exponent-127} * (1 + fraction/2^{23})$$

但是需要注意，有2个特殊情况，也就是上面说的指数位全0和全1的特殊用途。

1. exponent全0

   计算公式为：$(-1)^{sign} * 2^{exponent-126} * (0 + fraction/2^{23})$

2. exponent全1

   如果 fraction 全 0，则表示 +inf 或者 -inf （正负无穷）

   如果 fraction 不全为 0，则表示 NaN（表示“非数字”，NaN是一种特殊的符号，不等于任何其他的数字，包括自身，如对负数求平方根就会得到NaN）

我们写个代码验证一下：

```cpp
#include <iostream>
#include <bitset>

int main() {
    float a = 0.0f;
    unsigned int* p = reinterpret_cast<unsigned int*>(&a);  // 将浮点数的地址转换为unsigned int指针

    std::bitset<sizeof(float) * 8> bits(*p);  // 使用bitset解析float变量的二进制表示
    std::cout << "Original bits: " << bits << std::endl;  // 打印原始二进制位
```

```
10
11      // exponent全1
12      for(int i = 30; i > 30-8; i--)
13          bits.set(i, true);
14      // fraction全0
15      for(int i = 22; i >= 0; i--)
16          bits.set(i, false);
17
18      std::cout << "Modified bits: " << bits << std::endl;  // 打印修改后二进制位
19      *p = static_cast<unsigned int>(bits.to_ulong()); // 修改完成后，将bitset重新
    转换为float类型
20
21      std::cout << "Modified value: " << a << std::endl;  // 打印修改后的浮点数值
22      return 0;
23  }
24
```

```
Original bits: 00000000000000000000000000000000
Modified bits: 01111111100000000000000000000000
Modified value: inf
```

```
Original bits: 00000000000000000000000000000000
Modified bits: 01111111100000000000000000000001
Modified value: nan
```

上限值： exponent 设为 11111110，fraction 都为 1

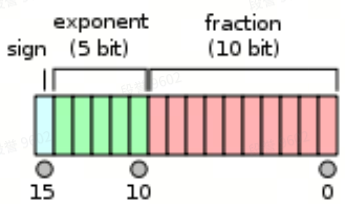
所以最大最小值为：-3.40282e+38 ~ 3.40282e+38，但这些数并不是等间隔分布的。 在不同的区间，间隔是不一样的。

有效动态范围：1.401298464324817e-45~3.4028234663852886e+38 注意这里不是从最小值到最大值， 而是说的正数的部分， 因为正负是对称的。

```
Original bits: 00000000000000000000000000000000
Modified bits: 01111111011111111111111111111111
Modified value: 3.40282e+38
```

```
Original bits: 00000000000000000000000000000000
Modified bits: 11111111011111111111111111111111
Modified value: -3.40282e+38
```

float16 / half



1. sign：符号位，1代表负数， 0代表正数。

2. 指数部分，5个比特位， 全0和全1有特殊用途，所以是00001~11110，也就是1到30， 减去偏置15，指数部分最终范围为-14 ~15。
3. 小数部分， 10个比特位， 范围为 0 ~ $1023/1024$

所以一个最终的数据计算方式为：

$$(-1)^{sign} * 2^{exponent-15} * (1 + fraction/1024)$$

但是需要注意，有2个特殊情况，也就是上面说的指数位全0和全1的特殊用途。

1. exponent全0

   计算公式为：$(-1)^{sign} * 2^{exponent-14} * (0 + fraction/1024)$

2. exponent全1

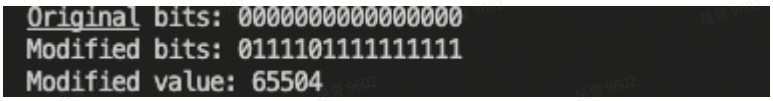
   如果 fraction 全 0，则表示 +inf 或者 -inf （正负无穷）

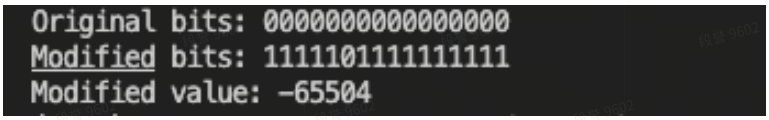
   如果 fraction 不全为 0，则表示 NaN

c++ 没有 float16的数据类型，cuda有，我们也写一段代码来看看 float16的最大和最小值：

```cpp
#include <cuda_fp16.h>
#include <stdio.h>
#include <bitset>
#include <iostream>

int main() {
    half a = __float2half(0.0f);

    unsigned int* p = reinterpret_cast<unsigned int*>(&a);  // 将浮点数的地址转换为unsigned int指针

    std::bitset<sizeof(half) * 8> bits(*p);  // 使用bitset解析half变量的二进制表示
    std::cout << "Original bits: " << bits << std::endl;  // 打印原始二进制位

    // exponent:11110, fraction:1111111111
    bits.set(15, true);
    for(int i = 14; i >= 0; i--){
        if(i == 10)
            bits.set(i, false);
        else
            bits.set(i, true);
    }

    std::cout << "Modified bits: " << bits << std::endl;  // 打印修改后二进制位
    // 修改完成后，将bitset重新转换为float类型
    *p = static_cast<unsigned int>(bits.to_ulong());
```

```
26
27      std::cout << "Modified value: " << __half2float(a) << std::endl;  // 打印修改后的浮点数值
28      return 0;
29  }
```





可以看出，表示范围为：-65504 ～ 65504，但这些数并不是等间隔分布的。 在不同的区间， 间隔是不一样的。

有效动态范围： 5.960464477539063e-08 ~65504， 注意这里不是从最小值到最大值， 而是说的正数的部分， 因为正负是对称的。

cuda_fp16

由于 c++ 没有 float16数据类型，所以一般使用cuda定义的 half，关于 cuda half 的一些function在 cuda 官方doc 中可以查询：

注意，有的函数在 host 端和 device 都可以调用，有的只能在 device 端调用，同时，不同的cuda版本之间也有不同，如 cuda11.3中 comparison 类函数只能在 device 端调用，但是cuda12.2中也可以在 host 端调用。

早期版本中，处理 half 的相关函数几乎都只能在 device 中调用，cuda 12.0 之后的版本，很多function也支持了 host端调用。

> Half Arithmetic Functions
>
> Half Comparison Functions
>
> Half Precision Conversion and Data Movement
>
> Half Match Functions

下面是一个简单的demo，定义两个half，然后做加减乘除运算，最后转化为 float 输出（std::cout, printf 无法直接输出 half，half 未重载相关运算符）

此外，说一个数值越界的问题，当数值上溢或者下溢时，直接取最大值或者最小值，如下：

```
1   #include <cuda_fp16.h>
2   #include <stdio.h>
3   #include <bitset>
4   #include <iostream>
5
```

```cpp
int main() {
    half a = __float2half(65504.0f + 1.0f);
    half b = __float2half(-65504.0f - 1.0f);
    std::cout << "a = " << __half2float(a) << std::endl;
    std::cout << "b = " << __half2float(b) << std::endl;
}
```

```
a = 65504
b = -65504
```