



AI Mode All Images Videos News More ▾



Sign in



fast floor division in c++

6:57 am

In C++, regular integer division (`/`) **truncates towards zero**. Implementing an efficient floor division (which rounds towards negative infinity) depends entirely on whether your operands are **unsigned integers, power-of-two denominators, or signed integers**. [Stack Overflow +1](#)

Here are the fastest, branchless methods for each scenario.

1. Unsigned Integers (Always Fast)

If both `a` and `b` are positive (unsigned), **truncation is identical to flooring**. [Stack Overflow](#)

cpp

```
// Best performance: Pure hardware division
unsigned int div_floor(unsigned int a, unsigned int b) {
    return a / b;
}
```

Use code with caution.



2. Dividing by a Power of Two (Fastest)

If your denominator `b` is a compile-time constant power of two (2^k), you can use a **bitwise right shift**. This acts as an automatic floor division for both signed and unsigned integers, entirely bypassing hardware division instructions.

cpp

```
// Equivalent to floor(a / 8)
int div_floor_by_8(int a) {
    return a >> 3;
}
```

Use code with caution.



3. Signed Integers (Branchless Modulo Trick)

When dealing with signed integers where the signs might differ, regular division rounds towards zero, meaning `-1 / 2` results in `0` instead of the floored `-1`. [Open-Std.org](#)

To make this fast, use a **branchless bitwise formula**. Modern CPUs compute both the quotient (`/`) and remainder (`%`) in a single hardware instruction cycle, making this highly optimized: [blog.pkh.me](#)

cpp

```
int div_floor_signed(int a, int b) {
    int q = a / b;
    int r = a % b;
    // If there's a remainder and the operands have opposite signs, subtract
    return q - (r != 0 && ((a ^ b) < 0));
}
```

Use code with caution.



4. Compile-Time Constants (Multiplicative Inverse)

If your divisor `b` is known at compile-time (but is not a power of two), simply write `a / b`. Modern optimizing compilers (like GCC or Clang with `-O3`) automatically optimize integer division by a constant into a **fixed-point multiplication and shift** using a magic number.  [GitHub](#)

Combined with the signed adjustment, it remains incredibly performant:

cpp

```
int div_floor_by_constant(int a) {
    const int b = 3; // Compile-time constant
    int q = a / b;    // Compiler turns this into a fast multiply + shift
    int r = a % b;
    return q - (r != 0 && ((a ^ b) < 0));
}
```

Use code with caution. 

Performance Summary

Scenario	Strategy	Complexity
Unsigned variables	<code>a / b</code>	Baseline hardware division
Power-of-two divisor	<code>a >> shift</code>	~1 clock cycle (Fastest)
Constant divisor	<code>a / CONST</code> + Sign correction	~2-4 clock cycles (No division instruction)
Dynamic Signed variables	Branchless modulo trick	Standard hardware division + bitwise ops

To help tailor this optimization further, tell me:

- Are your numbers **signed or unsigned**?
- Is the divisor a **compile-time constant** or a runtime variable?
- What specific **CPU architecture** (e.g., x86_64, ARM) are you targeting?

