Computer Architecture
Integer Arithmatic

Cain Susko

January 30, 2022

Queen's University
School of Computing

Addition

Unsigned Addition

when adding w-bit numbers, the sum may require more than w bits to represent, thus, we must apply modular arithmatic to the sum.
a **true** sum is one that *does not* need to use modular arithmatic. an **untrue** sum does.

$$sum = x +_w^u y \Rightarrow sum = x + y \bmod 2^w$$

Thus, any 4 bit integers x and y can compute the true sum $x + y$. this value increases linearly with x and y and if one were to graph this in 3D with axis x, y, $x + y$ one would get a plane.
If one were to graph these values again, but where $x+y \geq 2^4$. in other words, the sum cannot strictly be represented in 4 bits (untrue sums). one would see many identical planes representing the 'wrap around' effect of modular arithmatic.

Modular addition forms an **Albelian Group** which means it is closed, communative, and associative, with an identity of 0 and inverses for all operable elements.
respectively they are:

$$o \leq x +_w^u y \leq 2^w - 1$$
$$x +_w^u y = y +_w^u x$$
$$(x +_w^u y) +_w^u z = x +_w^u (y +_w^u z)$$
$$x +_w^u 0 = x$$
$$2^w - x$$

Signed Addition

this is also known as 2's Complement Addition
given the following equation, the classifications for the result are:

$$x +_w^t y$$

$$
\begin{aligned}
&2^{w-1} \leq x + y && \textbf{Positive Overflow} \\
&x + y < 2^{w-1} && \textbf{Negative Overflow} \\
&-2^{w-1} \leq x + y \leq 2^{w-1} && \textbf{Normal}
\end{aligned}
$$

note that $+_w^t$ and $+_w^u$, which are our mod operators, have the same bit level behaviour (in C). a true sum requires that there are $w + 1$ bits to use similar to Unsigned addition, graphing x, y and $x +_w^t y$ gives us many identical planes representing the range of $x +_w^t y$.

Complement and Incrament

the claim is this:
for 2's compement numbers, the following holds:

$$\neg x + 1 = -x$$

thus the **complement** of a number is:

$$
\begin{aligned}
&x && = 15213 = 0011101101101101 \\
&\neg x && = -15214 = 1100010010010010 \\
&\neg x + 1 && = -15213 = 1100010010010011
\end{aligned}
$$

thus $\neg x + 1$ is the complement of x.

Multiplication

multiplication can be done with both Signed and Unsigend integers. Just the same as in addition, we can have a value that is cannot be represented by w bits.

Unsigned	$0 \leq x * y \leq (2^w - 1)^2 = 2^{2w} - 2^{x+1} + 1$
Signed +	$x * y \geq (-2^{w-1} * 2^{w-1} - 1) = -2^{2w-2} + w^{w-1}$
Signed -	$x * y \leq (-2^{w-1})^2 = 2^{2w-2}$

w would have to change after each multiplication in order to maintain accuracy. otherwise we just use modular arithmetic

$$x *_w^t y$$

the overflow from this operator will be reported by the system unless amother solution is being used ie. increasing w

Power-of-2 by shifting

we can multiply ints by powers of 2 using the shift operator. For example:

$$x << k == x * 2^k$$

once we have the result for this operation, we then apply the same modular arithmatic for Signed or Unsigned ints respectively.

Shift Multiply

we can also multiply by numbers that are not powers of 2 by represented them through multiple shift operations:

$$(x << 5) - (x << 3) == x * 24$$

thus we can use this technique to multiply by any number// this is useful as most computers can do shifting and addition/subtraction much faster than multiplication. most compilers use this technique.

Shift Division

division can be emulated in the same way with the other shift operator. there is also a floor operator as we are only wokring with ints. we **always round down**

$$x >> k == \lfloor x/2^k \rfloor$$

note that this uses Logical shift

Summary

computer arithmetic is bound by the number of bits that we are using w and we must account for any overflow either by increasing w or by implementing modular arithmetic.