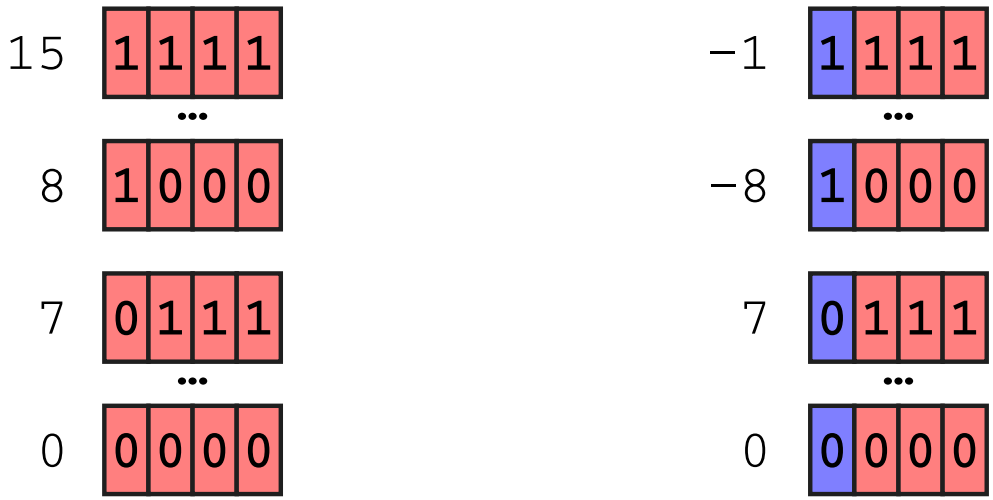
Fixed-point arithmetic

David Barina

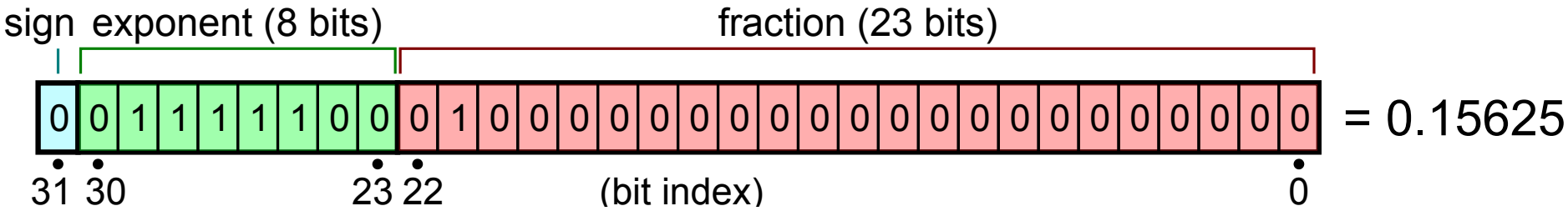
May 23, 2014

Integer

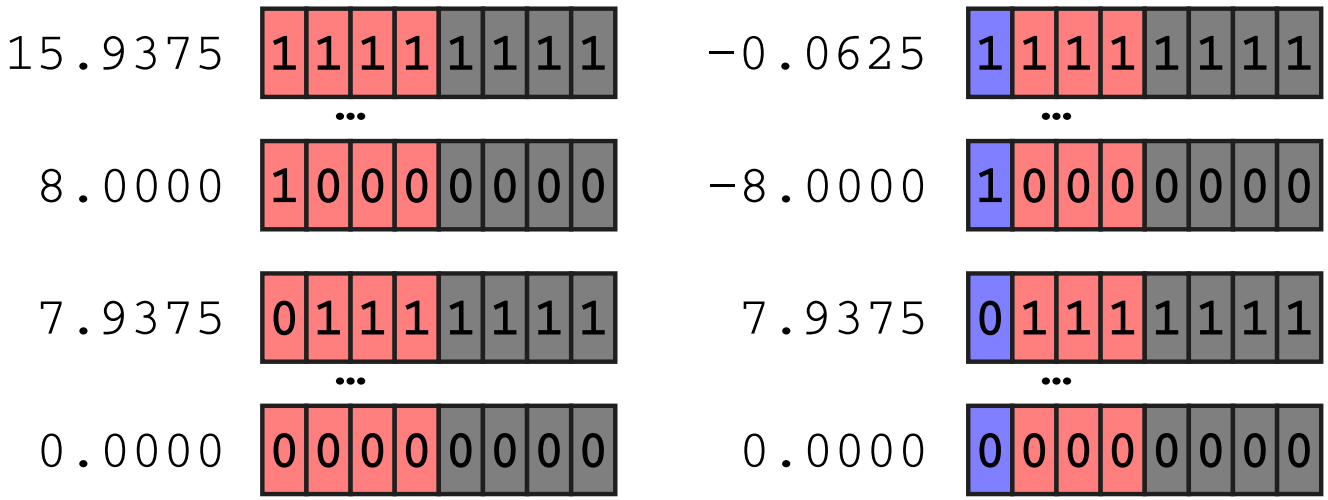


Floating point

$$a \times 2^b$$



Fixed point


15.9375
1 1 1 1 1 1 1 1
8.0000
1 0 0 0 0 0 0 0
7.9375
0 1 1 1 1 1 1 1
0.0000
0 0 0 0 0 0 0 0
−0.0625
1 1 1 1 1 1 1 1
−8.0000
1 0 0 0 0 0 0 0
7.9375
0 1 1 1 1 1 1 1
0.0000
0 0 0 0 0 0 0 0

Benefits

Advantages

- instructions for the integer arithmetic are sufficient
- no need for FP instructions (FPU, SSE)
- constant resolution in the whole range
- higher accuracy (small dynamic ranges)

Drawbacks

- limited range
- overflow
- less accuracy (greater dynamic ranges)

Notation

Qm.n

Q3.4	1 sign bit, 3 integer, 4 fractional bits
Q1.30	32 bits in total
Q15.16	32 bits in total

Examples

type	name	bits	resolution	range
INT	integer	32	1	$\pm 2^{31} \approx \pm 2.1 \times 10^{9}$
FP	single	32	$2^{-23} \approx 1.2 \times 10^{-7}$	$\pm 2^{128} \approx \pm 3.4 \times 10^{38}$
FX	Q15.16	32	$2^{-16} \approx 1.5 \times 10^{-5}$	$\pm 2^{15} \approx \pm 3.2 \times 10^{4}$
FX	Q1.30	32	$2^{-30} \approx 9.3 \times 10^{-10}$	± 2

Conversion (Qm.n)

INT → FX

```
y = x << n;
```

FX → INT

```
k = 1 << (n-1);
y = (x + k) >> n;
```

FP → FX

```
#include <math.h>
y = round( x * (1 << n) );
```

FX → FP

```
y = (double)x / (1 << n);
```

Rounding (Qm.n)

Truncation

```
int floor(int x) {
    return x & ~((1<<n)-1);
}
```

Ceiling

```
int ceil(int x) {
    return floor(x + ((1<<n)-1));
}
```

Rounding

```
int round(int x) {
    return floor(x + (1<<(n-1)));
}
```

Operations (Qm.n)

Zero

```
z = 0;
```

Unit

```
z = 1 << n;
```

One half

```
z = 1 << (n-1);
```

Negation

```
z = -x;
```

Absolute value

```
z = abs(x);
```

Operations (Qm.n)

Addition

```
z = x + y;
```

Subtraction

```
z = x - y;
```

Multiplication by an integer

```
z = x * i;
```

Multiplication

```
k = 1 << (n-1);
z = ( x * y + k ) >> n;
```

Division

```
z = (x << n) / y;
```

Multiplication (Q15.16)

Multiply $a.b \times c.d$

```
uint32_t result_low = (b * d);
uint32_t result_mid = (a * d) + (b * c);
uint32_t result_high = (a * c);

uint32_t result = (result_high << 16) + result_mid
    + (result_low >> 16);
```

Trick

Division by a constant

$$a/b \rightarrow a \times c \quad c = 1/b$$

```
z = a / 5;

// 1/5 = 0.2 = 858993459 in Q31.32
z = (a * 858993459) >> 32;
```

Functions

Sine by Taylor series [libfixmath]

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{!7}\left[+\frac{x^9}{!9} - \frac{x^{11}}{11!}\right]$$

Sine by polynomial [Fixmath]

16 segments, polynomial of degree 4, Remez alg.

$$\sin(x) = c_0 + c_1 x^1 + c_2 x^2 + c_3 x^3 + c_4 x^4$$

Functions

Inverse square root [Fixmath]

$$y = 1/\sqrt{x}$$

Newton's method

$x \rightarrow a \times 2^b$

$$y = y\,(3 - a\,y^2)/2$$

Square root [Fixmath]

$x \rightarrow a \times 2^b$

$$c = 1/\sqrt{a} \times a = \sqrt{a}$$

$y \leftarrow c \times 2^{b/2}$

Functions

Square root [libfixmath]

abacus algorithm

```
while (one != 0) {
        if (x >= y + one) {
                x = x - (y + one);
                y = y +  2 * one;
        }
        y /= 2;
        one /= 4;
}
```

Functions

Reciprocal value [Fixmath]

$$y = 1/x$$

Newton's method
$x \to a \times 2^b$

$$y = y\,(2 - ay)$$

Division

$$z = x/y \implies z = x \times (1/y)$$

Functions

Exponential function [libfixmath]

$x > 0 \implies e^{-x} = 1/e^{x}$

$$e^{x} = \sum_{n=0}^{\infty} \frac{x^{n}}{n!}$$

Natural logarithm [libfixmath]

Newton's method

$$y = y + \frac{x - e^{y}}{e^{y}}$$

Functions

Base-2 logarithm [libfixmath]

$$y = \log_2(x) \quad |Qm.n$$

```
y = 0;
while( x >= 2 ) {
    x /= 2;
    y++;
}
for_each(n) {
    x *= x;
    y *= 2;
    if( x >= 2 ) {
        x /= 2;
        y++;
    }
}
```

Functions

Base-2 logarithm [Fixmath]

$$y = \log_2(1 + x)$$

16 segments, polynomial of degree 4/11, Remez alg.

Base-2 exponential [Fixmath]

$$y = 2^x$$

1/32 segments, polynomial of degree 7/3, Remez alg.

Libraries

libfixmath http://code.google.com/p/libfixmath/

Fixmath http://savannah.nongnu.org/projects/fixmath/