Real Numbers

How do we represent real numbers?

Several issues:

- How many digits can we represent?
- What is the range?
- How accurate are mathematical operations?
- Consistency...

Is $a + b = b + a$?

Is $(a + b) + c = a + (b + c)$?

Is $(a + b) - b = a$?

Real Numbers

How do we represent real numbers?

Several issues:

- How many digits can we represent?
- What is the range?
- How accurate are mathematical operations?
- Consistency...

Is $a + b = b + a$?

Is $(a + b) + c = a + (b + c)$?

Is $(a + b) - b = a$?

Fixed Point

Basic idea:

0 1 0 0 1 0 1 0

radix point is here

Choose a *fixed* place in the binary number where the radix point is located.

For the example above, the number is

$$(010.01010)_2 = 2 + 2^{-2} + 2^{-4} = (2.3125)_{10}$$

How would you do mathematical operations?

Floating-Point

Some problematic numbers....

$$6.023 \times 10^{23}$$
$$6.673 \times 10^{-11}$$
$$6.62607 \times 10^{-34}$$

Scientific computations require a number of digits of precision...

But they also need *range*

$\Rightarrow$ permit the radix point to move

$\Rightarrow$ *floating-point numbers*

Floating-Point: Scientific Notation

$$\overbrace{\pm}^{\text{sign}}\ \underbrace{6.023}_{\text{mantissa}} \times \underbrace{10}_{\text{radix (base)}}{}^{\overbrace{23}^{\text{exponent}}}$$

- Number represented as:
  - mantissa, exponent
- Arithmetic
  - multiplication, division: perform operation on mantissa, add/subtract exponent
  - addition, subtraction: convert operands to have the same exponent value, add/subtract mantissas

Fixed Point

Basic idea:

0 1 0 0 1 0 1 0

radix point is here

Choose a *fixed* place in the binary number where the radix point is located.

For the example above, the number is

$$(010.01010)_2 = 2 + 2^{-2} + 2^{-4} = (2.3125)_{10}$$

How would you do mathematical operations?

Floating-Point

Some problematic numbers....

$$6.023 \times 10^{23}$$
$$6.673 \times 10^{-11}$$
$$6.62607 \times 10^{-34}$$

Scientific computations require a number of digits of precision...

But they also need *range*

$\Rightarrow$ permit the radix point to move

$\Rightarrow$ *floating-point numbers*

Floating-Point: Scientific Notation

$$\overset{\text{sign}}{\pm}\;\underbrace{6.023}_{\text{mantissa}} \times \underbrace{10}_{\text{radix (base)}}{}^{\overset{\text{exponent}}{23}}$$

- Number represented as:
  - mantissa, exponent
- Arithmetic
  - multiplication, division: perform operation on mantissa, add/subtract exponent
  - addition, subtraction: convert operands to have the same exponent value, add/subtract mantissas

Floating-Point: Scientific Notation

$$\overset{\text{sign}}{\pm}\ \underbrace{6.023}_{\text{mantissa}} \times \underbrace{10}_{\text{radix (base)}}{}^{\overbrace{23}^{\text{exponent}}}$$

Representation:

- IEEE 754 standard
- Standardized in mid-80s
- Single precision: 32 bits
- Double precision: 64 bits

Both supported by the MIPS processor.

Floating-Point Basics

Several design issues.

- Format Choices:
  - representation of mantissa (aka significand), exponent, sign
  - normal forms

$$6.023 \times 10^{23} = 0.6023 \times 10^{24} = \ldots$$

  - range and precision
- Arithmetic:
  - equalize exponents for add/subtract
  - inexact results, rounding
  - exceptional conditions/errors

IEEE Single Precision Format

S	exponent (E)	mantissa (M)
1	8	23

- Uses *biased exponent*, actual exponent is $(E - 127)$. 127 is called the bias (*excess* 127 bias)
- Number:

$$(-1)^S(1.M) \times 2^{E-127}$$

when $0 < E < 255$.

- The implied 1 is referred to as a hidden bit.
- Double-precision: 64 bits, 11-bit exponent, excess 1023 bias, 52-bit significand

Normalized Numbers

Significand is of the form $1.x$.

Example: $0.375 = (0.011)_2 = +(1.1)_2 \times 2^{-2}$

0	01111101	10000000000000000000000
+	$(125)_{10}$	$(0.1)_2$

Example: $-3.25 = -(11.01)_2 = -(1.101)_2 \times 2^1$

1	10000000	10100000000000000000000
–	$(128)_{10}$	$(0.101)_2$

Example: $0 = (0)_2 = ?$

Zero Exponent

$E = 0$ is special for this reason.

- Zero significand: number is 0
- Non-zero significand: *denormalized number*

$$(-1)^S(0.M) \times 2^{-126}$$

- Denormal numbers are used to extend the range of floating-point numbers.
- Double-precision: exponent would be -1022.
- Some hardware does not implement denormal arithmetic, but uses software emulation

On a Sun UltraSparc, > 80x slowdown...

Adding Normalized Numbers

To calculate $X + Y$, assuming $|Y| \geq |X|$:

- Alignment of radix point (denormalize smaller number)
  - $d := Exp(Y) - Exp(X)$, set $Exp := Exp(Y)$
  - $Sig(X) := Sig(X) >> d$
- Add the aligned components
  - $Sig = Sig(X) + Sig(Y)$
- Normalize the result
  - Shift Sig left/right, changing Exp
  - Check for overflow in Exp
  - Round; repeat if not normalized

Adding Normalized Numbers

Example: 4-bit significand

$1.0110 \times 2^3 + 1.1000 \times 2^2$

- Align

$$\begin{array}{rrcl} & 1.0110 & \times & 2^3 \\ + & 0.1100 & \times & 2^3 \end{array}$$

- Add

$$10.0010 \times 2^3$$

- Normalize

$$1.0001 \times 2^4$$

Adding Normalized Numbers

Example: 4-bit significand

$1.0001 \times 2^3 - 1.1110 \times 2^1$

- Align

$$\begin{array}{rll} & 1.0001 & \times\ 2^3 \\ - & 0.01111 & \times\ 2^3 \end{array}$$

- Subtract

$$0.10011 \times 2^3$$

- Normalize/Round

$$1.0011 \times 2^2$$

Without extra bit, result would be 1.0010×2^2

Accuracy

IEEE standard: want result to be as accurate as possible

- Maximum error: $\frac{1}{2}$ ulp (units in last place) when compared to infinite precision arithmetic
- Alignment step can be problematic!
- How many bits are actually needed for arithmetic?
- Extra bit in last example: *guard bit*

Rounding

Standard specifies 4 different rounding modes:

- round to nearest even (default)
- round toward $+\infty$
- round toward $-\infty$
- round toward 0

How many bits are necessary to correctly implement the standard?

Remember, the maximum permissible error is $\frac{1}{2}$ ulp.

Round Bit

Example: 4-bit significand

$1.0000 \times 2^0 - 1.0001 \times 2^{-2}$

- Align

$$\begin{array}{lll} & 1.0000 & \times\ 2^0 \\ - & 0.010001 & \times\ 2^0 \end{array}$$

- Subtract

$$0.101111 \times 2^0$$

- Normalize/Round

$$1.01111 \times 2^{-1}$$

$$1.1000 \times 2^{-1} \text{ (simple round up)}$$

Without extra bit, result would be 1.0111×2^{-1}

Sticky Bit And Round To Nearest Even

Example: 4-bit significand

$1.0000 \times 2^0 + 1.0001 \times 2^{-5}$

- Align

$$\begin{array}{lll} & 1.0000 & \times\ 2^0 \\ + & 0.000010|001 & \times\ 2^0 \end{array}$$

- Add

$$1.000010|001 \quad \times\ 2^0$$

- Normalize/Round

$$1.0001 \times 2^0, \text{ or}$$
$$1.0000 \times 2^0$$

Sticky bit: keep track of whether the bits "shifted out" are non-zero.

Infinity and NaNs

Sources of error:

- If the result is too large to be represented

  $\Rightarrow \pm\infty$

- What about $0/0$, $\infty - \infty$?

  $\Rightarrow$ "not a number" (NaN)

- NaNs propagate

  NaN $+x =$ NaN

- Can be used to initialize floating-point variables.

Representation: exponent is all "1"s (255 or 2047). If significand is 0, ∞; otherwise NaN.

Exceptions

- Invalid Operation
  - $\infty - \infty$, $0 \times \infty$, etc.
  - square root of a negative number
- Overflow
- Divide by Zero
- Underflow
  - denormal result or non-zero result underflows to zero
- Inexact
  - rounding error is not zero