

Lecture 6
Real-valued data in embedded software

Wednesday Feb 2, 2011

Philipp Rümmer
 Uppsala University
 Philipp.Ruemmer@it.uu.se

- Floating-point arithmetic
- Fixed-point arithmetic
- Interval arithmetic

1/46

2/46

Real values

- Most control systems have to operate on real-valued data
 - Time, position, velocity, currents, ...
- Various kinds of computation
 - Signal processing
 - Solving (differential) equations
 - (Non)linear optimisation
 - ...

3/46

Real values (2)

- Safety-critical decisions can depend on accuracy of computations, e.g.
 - Correct computation of braking distances
 - Correct estimate of elevator position + speed
- It is therefore important to understand behaviour and pitfalls of arithmetic

4/46

Real values in computers

- Finiteness of memory requires to work with approximations
 - Rational numbers
 - Arbitrary-precision decimal/binary numbers
 - Floating-point numbers
 - Fixed-point numbers
 - Arbitrary-precision integers
 - Bounded integers

5/46

Real values in computers (2)

- Most computations involve **rounding**
 - Precise result of computation cannot be represented
 - Instead, the result will be some value close to the precise result (hopefully)
- **Is it possible to draw reliable conclusions from approximate results?**

6/46

Machine arithmetic in a nutshell

Idealised arithmetic

- Idealised domain of computation: \mathbb{R}
- Idealised operations:

Literals:	$0, 1, 2, 0.1, 0.288, \dots$
Basic operations:	$+, -, \cdot, /$
Irrational op.:	$\sqrt{\quad}$
Transcendental op.:	π, \exp, \sin, \dots

 - ... precise mathematical definitions
- Algorithms using those operations:
 - Equation solvers, optimisation, ...

7/46

8/46

Machine arithmetic

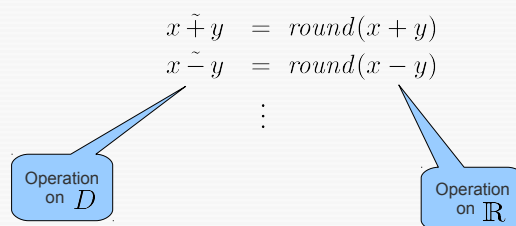
- Identify **domain** that can be represented on computers: $D \subseteq \mathbb{R}$
- Sometimes: further un-real values are added to domain D , such as $-0, \infty, -\infty$
- Define **rounding** operation:

$$\text{round} : \mathbb{R} \rightarrow D$$
 such that, for all $x \in \mathbb{R}$, $|\text{round}(x) - x|$ is “small”
- Lift** operations from \mathbb{R} to D ...

9/46

Machine arithmetic (2)

- Lift operations from \mathbb{R} to D
for $x, y \in D$:



10/46

Machine arithmetic (3)

- Conceptually**: machine operations are *idealised operations + rounding*
- Of course, **practical** implementation directly calculates on D

11/46

Floating-point arithmetic

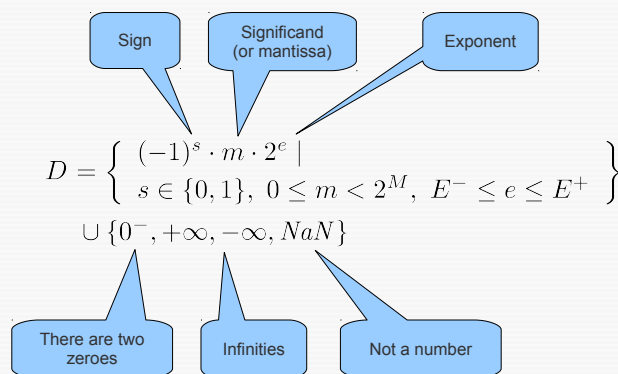
12/46

Overview

- Most common way of representing reals on computers
- Normally used: IEEE 754-2008 floats
- Many processors support floats directly (but **most micro-controllers do not**)
- In many cases: **Fast, robust, convenient** method to work with reals

13/46

Domain of (Binary) Floats



14/46

Float parameters

- Size of significand: M bit
- Size of exponent: E bit
- Defined range:

$$\begin{aligned} E^- &= 3 - 2^{E-1} - M \\ E^+ &= 2^{E-1} - M \end{aligned}$$

(two magical exponent values, signal NaN, etc.)

$$D = \left\{ \begin{array}{l} (-1)^s \cdot m \cdot 2^e \\ s \in \{0, 1\}, 0 \leq m < 2^M, E^- \leq e \leq E^+ \end{array} \right\} \cup \{-0, +\infty, -\infty, NaN\}$$



15/46

Binary encoding

- IEEE 754-2008 defines binary encoding in altogether $M + E$ bit (saving one bit)
- Typical float types:

	Significant size M	Exponent size E	Exponent range
binary16	11 bit	5 bit	-24 .. +5
binary32	24 bit	8 bit	-149 .. +104
binary64	53 bit	11 bit	-1074 .. +971
80-bit ext. prec.	65 bit	15 bit	-16446 .. +16319
binary128	113 bit	15 bit	-16494 .. +16271

16/46

- Always round real number to next smaller/greater floating-point number
- 5 rounding modes:
 - `roundNearestTiesToEven`
 - `roundNearestTiesToAway`
 - `roundTowardPositive` 
 - `roundTowardNegative` 
 - `roundTowardZero`

17/46

Problems with floating-point arithmetic

- Of course, never compare for equality!

18/46

Problem 1: performance

- Most micro-controllers don't provide a floating-point unit
→ All computations done in software
- E.g., on CORTEX M3, floating-point operations are more than 10 times slower than integer operations
- **Not an option if performance is important**

19/46

20/46

Problem 2: mathematical properties

- Floats don't quite behave like reals
 - No associativity: $(x + y) + z \neq x + (y + z)$
 - Density/precision varies
→ often strange for physical data
- Standard transformations of programs/expressions might affect result
- Can exhibit extremely unintuitive behaviour

21/46

Problem 3: rounding

- Rounding errors can add up and propagate (problematic when using **numerically unstable algorithms**)
- Usually, default rounding is not directed (round-to-nearest)
- Example (taken from Wikipedia):

```
function sumArray(array) is
  let theSum = 0
  for each element in array do
    let theSum = theSum + element
  end for
  return theSum
end function
```

22/46

Problem 4: inconsistent semantics

- In practice, semantics depends on:
 - Processor (not all are IEEE compliant)
 - Compiler, compiler options (optimisations might affect result)
 - Whether data is stored in memory or in registers → register allocation
- For many processors: 80bit floating-point registers; C/IEEE semantics only says 64bit

23/46

Problem 4: inconsistent semantics (2)

- Transcendental functions are not even standardised
- Altogether: it's a mess
- **Floats have to be used extremely carefully in safety-critical contexts**

24/46

Further reading

- Pitfalls of floats:
<http://arxiv.org/abs/cs/0701192>
- IEEE 754-2008 standard
- More concise definition of floats:
<http://www.philipp.ruemmer.org/publications/smt-fpa.pdf>

25/46

Overview

- Common alternative to floats in embedded systems
- *Intuitively*: store data as integers, with sufficiently small units
E.g.
floats in m \leftrightarrow integers in μm
- Performance close to integer arith.
- Uniform density, but smaller range
- Not directly supported in C
→ often has to be implemented by hand

27/46

Rounding

- Normally: rounding down

$$\text{round}(x) = \begin{cases} \max \{y \in D \mid y \leq x\} & \text{if } x \geq 0 \\ \text{undef} & \text{if } \neg \exists y \in D. y \leq x \end{cases}$$

- Extension to other rounding modes (e.g., rounding up) is possible

29/46

Operations: addition, subtraction

$$\begin{aligned} (m \cdot 2^{-P}) \tilde{+} (n \cdot 2^{-P}) &= (m + n) \cdot 2^{-P} \\ (m \cdot 2^{-P}) \tilde{-} (n \cdot 2^{-P}) &= (m - n) \cdot 2^{-P} \end{aligned}$$

- Simply add/subtract significands
- No rounding
- Over/underflows might occur

31/46

Fixed-point arithmetic

26/46

Domain of Fixed-point Arithmetic



$$\begin{aligned} D_{\text{unsigned}} &= \{m \cdot 2^{-P} \mid 0 \leq m < 2^M\} \\ D_{\text{signed}} &= \{m \cdot 2^{-P} \mid -2^{M-1} \leq m < 2^{M-1}\} \end{aligned}$$

- E.g, for $M = 3, P = 1$

$$D_{\text{unsigned}} = \{0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5\}$$

28/46

Implementation

- Normally:
 - Significand m is stored as integer variable (32bit, 64bit, signed/unsigned)
 - Exponent P is fixed upfront
- Operations:
Integer operations + shifting

On ARM CORTEX:
partly available as
native instructions

30/46

Operations: multiplication

$$(m \cdot 2^{-P}) \tilde{\cdot} (n \cdot 2^{-P}) = ((m \cdot n) \ggg P) \cdot 2^{-P}$$

- Multiply significands, shift result
- Shifting can cause rounding
- \ggg : shift with sign-extension

$$x \ggg y = \lfloor x \cdot 2^{-y} \rfloor$$

32/46

Operations: multiplication (2)

$$(m \cdot 2^{-P}) \cdot (n \cdot 2^{-P}) = ((m \cdot n) \gg P) \cdot 2^{-P}$$

- E.g.: $M = 3, P = 1$, unsigned

$$\begin{aligned} 0.5 \cdot 2.0 &= 1.0 \\ (1 \cdot 2^{-1}) \cdot (4 \cdot 2^{-1}) &= (1 \cdot 4 \gg 1) \cdot 2^{-1} \end{aligned}$$

$$\begin{aligned} 0.5 \cdot 1.5 &= 0.75 \\ (1 \cdot 2^{-1}) \cdot (3 \cdot 2^{-1}) &= (1 \cdot 3 \gg 1) \cdot 2^{-1} \end{aligned}$$

Rounding!

33/46

Operations: division

$$(m \cdot 2^{-P}) / (n \cdot 2^{-P}) = ((m \ll P) \div n) \cdot 2^{-P}$$

- Shift numerator, then divide by denominator
- \div : integer division, rounding towards zero
- Same potential problem as with multiplication

35/46

Further reading

- Fixed-point arithmetic on ARM CORTEX:
<http://infocenter.arm.com/help/topic/com.arm.doc.dai0033a/>

37/46

Interval methods

39/46

Operations: multiplication (3)

$$(m \cdot 2^{-P}) \cdot (n \cdot 2^{-P}) = ((m \cdot n) \gg P) \cdot 2^{-P}$$

- **Problem:** with this naïve implementation, overflows can occur during computation even if the result can be represented

$$\begin{aligned} 1.5 \cdot 2.0 &= 3.0 \\ (3 \cdot 2^{-1}) \cdot (4 \cdot 2^{-1}) &= (3 \cdot 4 \gg 1) \cdot 2^{-1} \end{aligned}$$

Intermediate result 12 cannot be stored in 3bit

34/46

Further operations:
pow, exp, sqrt, sin, ...

- ... can be implemented efficiently using Newton iteration, shift-and-add, or CORDIC

36/46

In practice ...

- Fixed-point operations are often implemented as macros

38/46

Motivation

- Is it possible to draw reliable conclusions from computations done with limited precision?

40/46

Motivation (2)

- **Yes** ... if we can control the rounding!
- E.g.

Safe if '+' rounds down

```
if (a + b >= 0) {
  // do something dangerous that
  // that only works if a+b >= 0
}
```

41/46

Interval methods

- Common in scientific computing, numerical mathematics; rich body of research
- Often an extremely efficient method to obtain estimates on the precise result of computations
- Are sound in the sense that no wrong answers are given, but can fail since error bounds might be too large to be useful

42/46

Idealised interval arithmetic

- Domain of real intervals:

$$\mathbb{IR} = \{x = [\underline{x}, \bar{x}] \mid \underline{x} \in \mathbb{R} \cup \{-\infty\}, \bar{x} \in \mathbb{R} \cup \{+\infty\}, \underline{x} \leq \bar{x}\}$$

- Mathematical operations are lifted to intervals, e.g.:

$$\begin{aligned} x + y &= [\underline{x} + \underline{y}, \bar{x} + \bar{y}] \\ x - y &= [\underline{x} - \bar{y}, \bar{x} - \underline{y}] \end{aligned}$$

- Generally defined such that:

$$x \text{ op } y \supseteq \{x \text{ op } y \mid x \in \underline{x}, y \in \underline{y}\}$$

43/46

Machine interval arithmetic

- Intervals represented using two machine numbers (floating-point or fixed-point):

$$\mathbb{ID} = \{x = [\underline{x}, \bar{x}] \mid \underline{x} \in D \cup \{-\infty\}, \bar{x} \in D \cup \{+\infty\}, \underline{x} \leq \bar{x}\}$$

- Operations are used *rounding outward*:

$$x \tilde{+} y = [\text{round}_{\downarrow}(\underline{x} + \underline{y}), \text{round}_{\uparrow}(\bar{x} + \bar{y})]$$

- This ensures that the precise result is always within the computed machine interval

44/46

Example

- Fixed-point arithmetic with rounding:

$$\begin{aligned} 0.5 \tilde{\cdot} 1.5 &= 0.5 \\ (1 \cdot 2^{-1}) \tilde{\cdot} (3 \cdot 2^{-1}) &= (1 \cdot 3 \gg 1) \cdot 2^{-1} \end{aligned}$$

- In interval arithmetic:

$$\begin{aligned} [0.5, 0.5] \tilde{\cdot} [1.5, 1.5] &= [\text{round}_{\downarrow}(0.75), \text{round}_{\uparrow}(0.75)] \\ &= [0.5, 1.0] \end{aligned}$$

Inputs are singleton intervals

Output bounds the precise result

45/46

Further reading

- Library on interval methods:

<http://www.cs.utep.edu/interval-comp/>

46/46