Masterthesis

Implementation and Verification of IEEE-conform Floating Point Arithmetic

Jacqueline Stratmann

November 20, 2015

Content

Task

- Binary floating point arithmetic
- Identification of mandatory operations
- Definition
  - Textual
  - Flowcharts
- Verification
- Implementation using Quartz

Motivation

- IEEE Standard 754-2008
  - Textual description of results
  - Some decisions passed to programmer
  - No definition of calculation
  - *"It is hoped that language-defined methods for the control of expression evaluation and exceptions might be defined in coming years, so that it will be possible to write programs that produce identical results on all conforming systems."*
- Results should be reproducible and comparable
- No complete definition and verification until now

IEEE Standard 754-2008

- $\langle [x_n, ..., x_0] \rangle^{\mathbb{R}}_{B,m,e} =$
  $(-1)^{x_n} \cdot (\langle [x_{m-1}, ..., x_0] \rangle^{\mathbb{N}}_{B} \cdot B^{1-m}) \cdot B^{\langle [x_{n-1},...,x_m] \rangle^{\mathbb{N}}_{B} - \beta}$
- Special numbers: NaN, $\pm\infty$, ± 0, Subnormal numbers
- Different rounding directions (standard: roundTiesToEven)
- Exceptions
  - Invalid operation exception
  - DivideByZero exception
  - Overflow exception
  - Underflow exception
  - Inexact exception

Mandatory Operations

General computational operations
roundToIntegralTiesToEven
roundToIntegralTiesToAway
roundToIntegralTowardZero
roundToIntegralTowardPositive
roundToIntegralTowardNegative
roundToIntegralExact
nextUp
nextDown
remainder
minNum
maxNum
minNumMag
maxNumMag

LogBFormat operations
scaleB
logB

Conversion operations
convertFromHexCharacter
convertToHexCharacter

Arithmetic operations
addition
subtraction
multiplication
division
squareRoot
fusedMultiplyAdd
convertFromInt
convertToIntegerTiesToEven
convertToIntegerTowardZero
convertToIntegerTowardPositive
convertToIntegerTowardNegative
convertToIntegerTiesToAway
convertToIntegerExactTiesToEven
convertToIntegerExactTowardZero
convertToIntegerExactTowardPositive
convertToIntegerExactTowardNegative
convertToIntegerExactTiesToAway

Sign bit operations
copy
negate
abs
copySign

Comparison operations
compareQuietEqual
compareQuietNotEqual
compareSignalingEqual
compareSignalingGreater
compareSignalingGreaterEqual
compareSignalingLess
compareSignalingLessEqual
compareSignalingNotEqual
compareSignalingNotGreater
compareSignalingLessUnordered
compareSignalingNotLess
compareSignalingGreaterUnordered
compareQuietGreater
compareQuietGreaterEqual
compareQuietLess
compareQuietLessEqual
compareQuietUnordered
compareQuietNotGreater
compareQuietLessUnordered
compareQuietNotLess
compareQuietGreaterUnordered
compareQuietOrdered

Confirmance predications
is754version1985
is754version2008

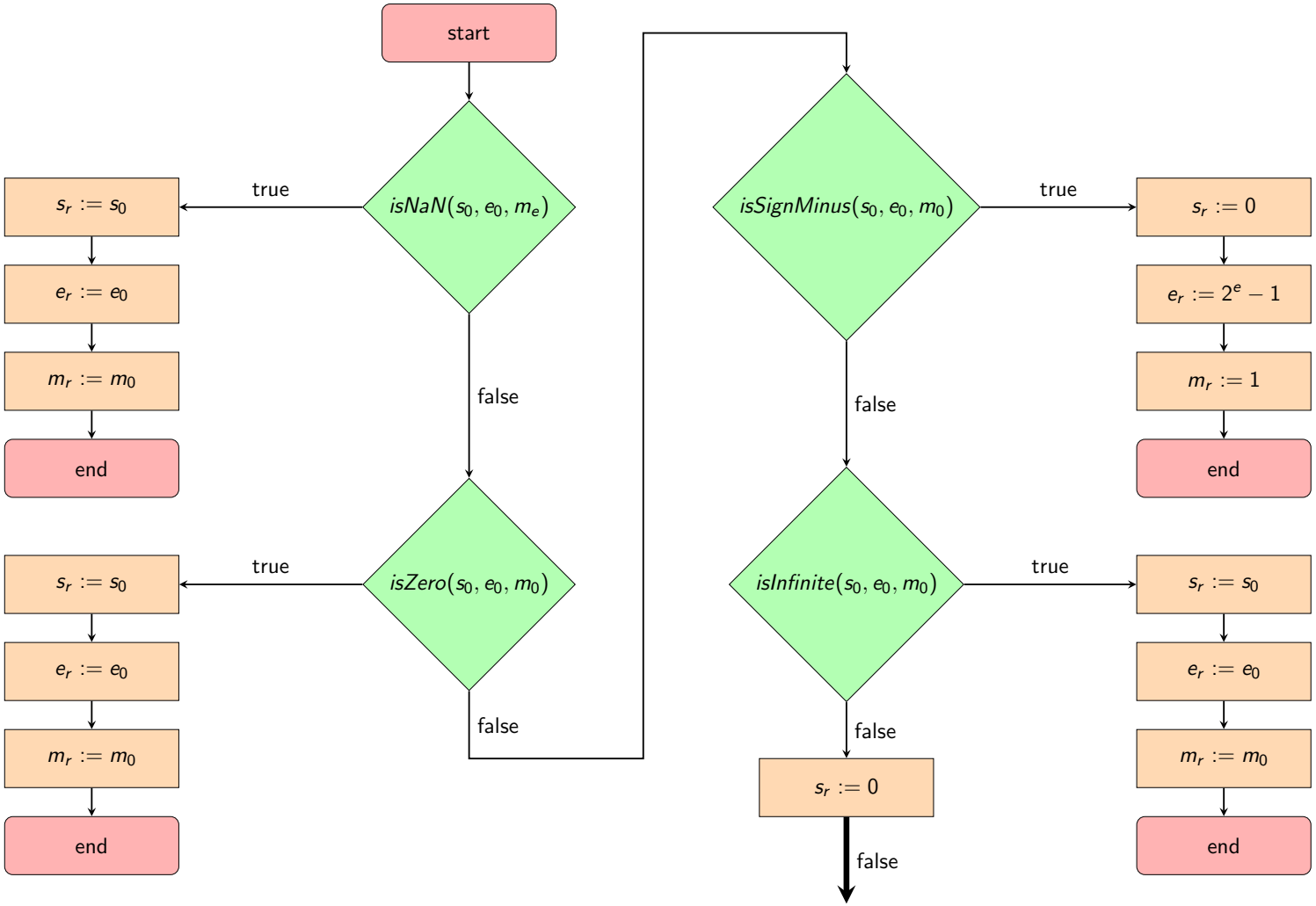
General non-computational operations
class
isSignMinus
isNormal
isFinite
isZero
isSubnormal
isInfinite
isNaN
isSignaling
isCanonical
radix
totalOrder
totalOrderMag

Flag operations
lowerFlags
raiseFlags
testFlags
testSavedFlags
restoreFlags
saveAllFlags

Square Root

- Special Cases (IEEE Standard 754-2008)
  - $\sqrt{NaN} := NaN$
  - $\sqrt{-0} := -0$
  - $\sqrt{x < 0} := NaN$
  - $\sqrt{\infty} := \infty$
- Used calculation rules
  - $\sqrt{x \cdot y} = \sqrt{x} \cdot \sqrt{y}$
  - $\sqrt{x^y} = x^{\frac{y}{2}}$
  - $\sqrt{m \cdot b^e} = \sqrt{m} \cdot b^{\frac{e}{2}}$

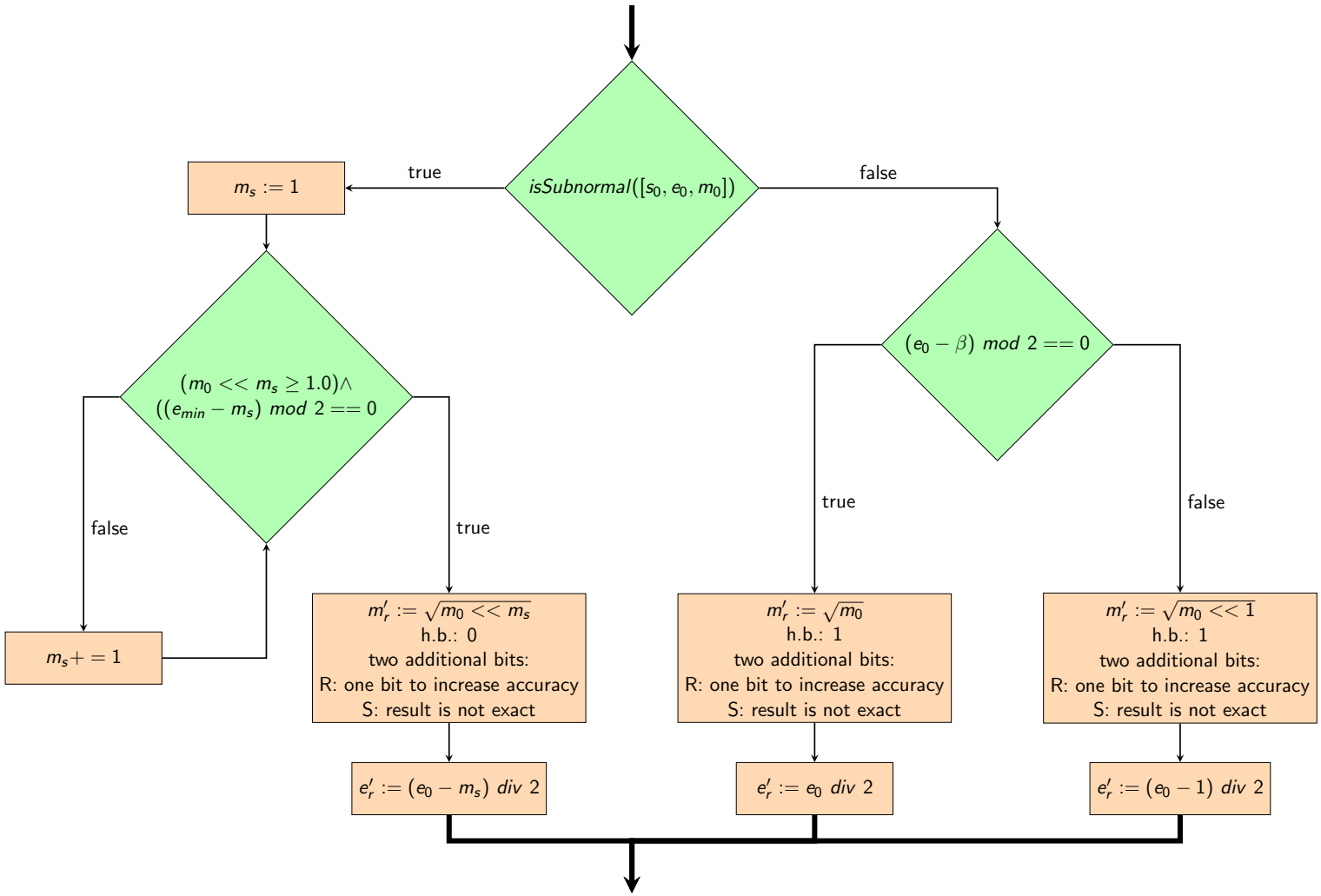
Flowchart (part 1/3)



Flowchart (part 1/3) summary

- $\sqrt{NaN} := NaN$
- $\sqrt{\pm 0} := input$
- $\sqrt{-x} := NaN$
- $\sqrt{\infty} := \infty$
- Only positive, finite and non-zero inputs remaining
- $s_r := 0$

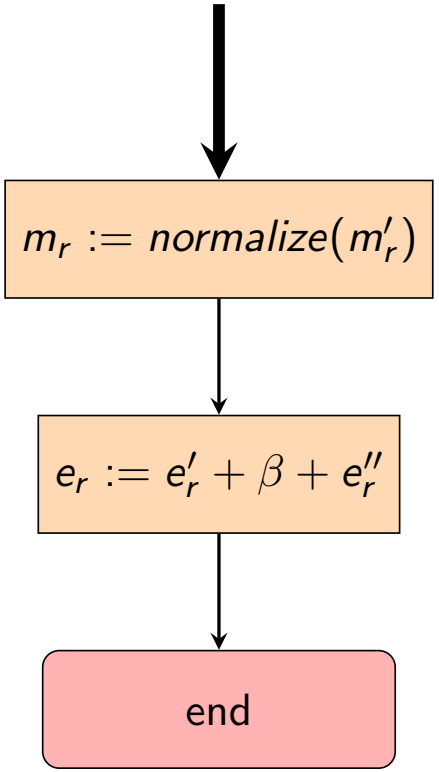
Flowchart (part 2/3)



Flowchart (part 2/3) summary

- Mantissa calculated
- Exponent calculated
- Both are not normalized at this point

Flowchart (part 3/3)



Verification

- Result of calculation has to be correct
  $\Rightarrow$ Calculation of square root is correct
- No lack of accuracy during normalization
  $\Rightarrow$ Normalization is correct

Calculation of square root is correct

$$\sqrt{x} = \begin{cases} 2^{(e_{min}-m_s)\ div\ 2} \cdot \sqrt{1.m_{x_{m-1-m_s}} \ldots m_{x_0} \underbrace{0\ldots0}_{m_s}} & \text{if } x \textit{ subnormal}, (e_{min} - m_s) \textit{ even} \\ 2^{(e_{min}-m_s-1)\ div\ 2} \cdot \sqrt{1m_{x_{m-1-m_s}}.m_{x_{m-2-m_s}} \ldots m_{x_0} \underbrace{0\ldots0}_{m_s+1}} & \text{if } x \textit{ subnormal}, (e_{min} - m_s) \textit{ odd} \\ 2^{(e_x-\beta)\ div\ 2} \cdot \sqrt{1.m_x} & \text{if } x \textit{ normal}, (e_0 - \beta) \textit{ even} \\ 2^{(e_x-\beta-1)\ div\ 2} \cdot \sqrt{1m_{x_{m-1}}.m_{x_{m-2}} \ldots m_{x_0} 0} & \text{else} \end{cases}$$

Case 1: x subnormal, $(e_{min} - m_s)$ even

$$\sqrt{2^{e_{min}} \cdot 0.m_x} = \sqrt{2^{e_{min}-m_s} \cdot 1.m_{x_{m-1-m_s}} \ldots m_{x_0} \underbrace{0\ldots0}_{m_s}}$$

$$= \sqrt{2^{e_{min}-m_s}} \cdot \sqrt{1.m_{x_{m-1-m_s}} \ldots m_{x_0} \underbrace{0\ldots0}_{m_s}}$$

$$= 2^{\frac{e_{min}-m_s}{2}} \cdot \sqrt{1.m_{x_{m-1-m_s}} \ldots m_{x_0} \underbrace{0\ldots0}_{m_s}}$$

$$= 2^{(e_{min}-m_s)\ div\ 2} \cdot \sqrt{1.m_{x_{m-1-m_s}} \ldots m_{x_0} \underbrace{0\ldots0}_{m_s}}$$

Case 2: x subnormal, $(e_{min} - m_s)$ odd

$$\sqrt{2^{e_{min}} \cdot 0.m_x} = \sqrt{2^{e_{min}-m_s-1} \cdot 1m_{x_{m-1-m_s}}.m_{x_{m-2-m_s}}...m_{x_0}\underbrace{0...0}_{m_s+1}}$$

$$= \sqrt{2^{e_{min}-m_s-1}} \cdot \sqrt{1m_{x_{m-1-m_s}}.m_{x_{m-2-m_s}}...m_{x_0}\underbrace{0...0}_{m_s+1}}$$

$$= 2^{\frac{e_{min}-m_s-1}{2}} \cdot \sqrt{1m_{x_{m-1-m_s}}.m_{x_{m-2-m_s}}...m_{x_0}\underbrace{0...0}_{m_s+1}}$$

$$= 2^{(e_{min}-m_s-1)\ div\ 2} \cdot \sqrt{1m_{x_{m-1-m_s}}.m_{x_{m-2-m_s}}...m_{x_0}\underbrace{0...0}_{m_s+1}}$$

Case 3: x normal, $(e_x - \beta)$ even

$$\sqrt{2^{e_x-\beta} \cdot 1.m_x} = \sqrt{2^{e_x-\beta}} \cdot \sqrt{1.m_x} = 2^{\frac{e_x-\beta}{2}} \cdot \sqrt{1.m_x}$$
$$= 2^{(e_x-\beta) \ div \ 2} \cdot \sqrt{1.m_x}$$

Case 4: x normal, $(e_x - \beta)$ odd

$$\sqrt{2^{e_x-\beta} \cdot 1.m_x} = \sqrt{2^{e_x-\beta-1} \cdot 1m_{x_{m-1}}.m_{x_{m-2}}...m_{x_0}0}$$
$$= \sqrt{2^{e_x-\beta-1}} \cdot \sqrt{1m_{x_{m-1}}.m_{x_{m-2}}...m_{x_0}0}$$
$$= 2^{\frac{e_x-\beta-1}{2}} \cdot \sqrt{1m_{x_{m-1}}.m_{x_{m-2}}...m_{x_0}0}$$
$$= 2^{(e_x-\beta-1)\ div\ 2} \cdot \sqrt{1m_{x_{m-1}}.m_{x_{m-2}}...m_{x_0}0}$$

Normalization is correct

Remains to be proven:

- $\sqrt{1x_n.x_{n-1}...x_0} \geq 1.0$
- $\sqrt{1.x_n...x_0} \geq 1.0$

Case 1: $\sqrt{1x_n.x_{n-1}...x_0} \geq 1.0$

$$
\begin{array}{lll}
 & x \geq 1.0 & \mid \sqrt{};\ x = 1x_n.x_{n-1}...x_0 \geq 0 \\
\Rightarrow & \sqrt{x} \geq \sqrt{1.0} & \mid 1.0 = 1.0^2 \\
\Rightarrow & \sqrt{x} \geq \sqrt{1.0^2} & \\
\Rightarrow & \sqrt{x} \geq 1.0 &
\end{array}
$$

Case 2: $\sqrt{1.x_n...x_0} \geq 1.0$

$$
\begin{array}{lll}
 & x \geq 1.0 & \mid \sqrt{};\ x = 1.x_n...x_0 \geq 0 \\
\Rightarrow & \sqrt{x} \geq \sqrt{1.0} & \mid 1.0 = 1.0^2 \\
\Rightarrow & \sqrt{x} \geq \sqrt{1.0^2} & \\
\Rightarrow & \sqrt{x} \geq 1.0 &
\end{array}
$$

Verification: Summary

- Shift left: cannot occur
- Shift right: reduces accuracy $\Rightarrow$ no additional bits required
- Rounding: two additional bits R and S
- Calculation of square root of mantissa
  - Known fixed point operation
  - Already verified by others

Implementation

- Quartz
  - Synchronous language
- Validation by unit testing (”drivenby“)
- Results compared to results of 32-bit floating point calculations (C++)
  - Extensive: Addition, Multiplication, Division, Square Root
- Not implemented:
  - Exceptions
  - Operations based on strings

Future Work

- Optimizations
  - Speed
  - Resource usage
  - ...
- Add optional operations

End

Thank you for your attention!