Logarithm with correct rounding

Sid-Ali Zitouni-Terki

September 8th 2022

Outline

1. Introduction
2. Floating point numbers
   - Definition
   - Necessary tools
3. Steps to calculate crlog
4. Algorithm
   - Operator precision calculation
   - Logarithme
5. Conclusion

Introduction

Logarithm with correct rounding

Floating point numbers

- $x = s.M.\beta^{e-p+1}$ with s the **sign**, M the **mantissa** ($|M| \leq \beta^p - 1$), β a **base(radix)**, e the **exponent** and p the **precision**.

-

-

Format	*Sign*	*Exponent*	*Mantissa*
binary32	1 bit	8 bits	24 bits
binary64	1 bit	11 bits	53 bits
binary128	1 bit	15 bits	113 bits

Floating point numbers

- $x = s.M.\beta^{e-p+1}$ with s the **sign**, M the **mantissa** ($|M| \leq \beta^p - 1$), β a **base(radix)**, e the **exponent** and p the **precision**.

- 

Format	*Sign*	*Exponent*	*Mantissa*
*binary*32	1 bit	8 bits	24 bits
*binary*64	1 bit	11 bits	53 bits
*binary*128	1 bit	15 bits	113 bits

Floating point numbers

- $x = s.M.\beta^{e-p+1}$ with s the **sign**, M the **mantissa** ($|M| \leq \beta^p - 1$), β a **base(radix)**, e the **exponent** and p the **precision**.
- 

Format	*Sign*	*Exponent*	*Mantissa*
*binary*32	1 bit	8 bits	24 bits
*binary*64	1 bit	11 bits	53 bits
*binary*128	1 bit	15 bits	113 bits

- RNDN:
  - **Ties to even**
  - **Ties to away**
- RNDZ
- RNDU
- RNDD

Real Number	+140.8215064611465	+665.575295525412
RNDN(Ties to even)	+140.821506461146	+665.57529552541
RNDN(Ties to away)	+140.821506461147	+665.57529552541
RNDZ	+140.821506461146	+665.57529552541
RNDU	+140.821506461147	+665.57529552542
RNDD	+140.821506461146	+665.57529552541

- RNDN:
  - **Ties to even**
  - **Ties to away**
- RNDZ
- RNDU
- RNDD

Real Number	+140.8215064611465	+665.5752955525412
RNDN(Ties to even)	+140.821506461146	+665.575295552541
RNDN(Ties to away)	+140.821506461147	+665.575295552541
RNDZ	+140.821506461146	+665.575295552541
RNDU	+140.821506461147	+665.575295552542
RNDD	+140.821506461146	+665.575295552541

- RNDN:
  - **Ties to even**
  - **Ties to away**
- RNDZ
- RNDU
- RNDD

Real Number	+140.8215064611465	+665.5752955525412
RNDN(Ties to even)	+140.821506461146	+665.575295552541
RNDN(Ties to away)	+140.821506461147	+665.575295552541
RNDZ	+140.821506461146	+665.575295552541
RNDU	+140.821506461147	+665.575295552542
RNDD	+140.821506461146	+665.575295552541

- RNDN:
  - **Ties to even**
  - **Ties to away**
- RNDZ
- RNDU
- RNDD

Real Number	+140.8215064611465	+665.575295525412
RNDN(Ties to even)	+140.821506461146	+665.57529552541
RNDN(Ties to away)	+140.821506461147	+665.57529552541
RNDZ	+140.821506461146	+665.57529552541
RNDU	+140.821506461147	+665.57529552542
RNDD	+140.821506461146	+665.57529552541

- RNDN:
  - Ties to even
  - Ties to away
- RNDZ
- RNDU
- RNDD

-

Real Number	+140.8215064611465	+665.5752955525412
RNDN(Ties to even)	+140.821506461146	+665.575295552541
RNDN(Ties to away)	+140.821506461147	+665.575295552541
RNDZ	+140.821506461146	+665.575295552541
RNDU	+140.821506461147	+665.575295552542
RNDD	+140.821506461146	+665.575295552541

- $+0$ and -0
- $-\infty$ and $+\infty$
- NaN
- **Subnormal**
- **Overflow**
- **Underflow**

Definition (ulp(x))

*Let x a **floating-point** number, $x = d_0 d_1 d_2 d_3 d_4 \ldots d_{p-1} \beta^e$. we therefore have an error to represent it :$\left| d_0 d_1 d_2 d_3 d_4 \ldots d_{p-1} - \frac{x}{\beta^e} \right|$ which is the unit of the last place.*

-

Corollary

*If x is a **floating-point** number, $d_0 = 1$ and that its base is 2, then we have :*

$$ulp(x) \leq 2^{-52} |x|$$

.

-

Definition (ulp(x))

*Let x a **floating-point** number, $x = d_0 d_1 d_2 d_3 d_4 \ldots d_{p-1} \beta^e$. we therefore have an error to represent it :$|d_0 d_1 d_2 d_3 d_4 \ldots d_{p-1} - \frac{x}{\beta^e}|$ which is the unit of the last place.*

-

Corollary

*If x is a **floating-point** number, $d_0 = 1$ and that its base is 2, then we have :*

$$ulp(x) \leq 2^{-52}|x|$$

.

-

Property

Property

$$If\ X = RNDN(x) \Rightarrow |X - x| \leq \frac{1}{2} ulp(x)$$

Property

$$If\ X = RNDN(x) \Rightarrow |X - x| \leq \frac{1}{2} ulp(X)$$

Property

$$If\ X \in \{RNDD(x), RNDU(x), RNDZ(x), RNDN(x)\} \Rightarrow |X - x| \leq ulp(x)$$

Property

$$If\ X \in \{RNDD(x), RNDU(x), RNDZ(x), RNDN(x)\} \Rightarrow |X - x| \leq ulp(X)$$

Unit Roundoff

$$u = \begin{cases} \frac{1}{2}.ulp(1) = \frac{1}{2}.2^{-52} = 2^{-53} & for \quad RNDN \\ ulp(1) = 2^{-52} & for \quad (RNDU, RNDZ, RNDD) \end{cases}$$

Sterbenz's lemma

Lemma (Sterbenz)

*In a **radix-β** floating-point system with **subnormal** numbers available, if x and y are finite **floating-point** numbers such that $\frac{y}{2} \leq x \leq 2.y$, then $x - y$ is exactly representable.*

Correct rounding methodology

1. $cr\log_{fast}$ computes an approximation y of $f(x)$ with a small error bound ϵ_1;
2. rounding test :$[y - \epsilon_1, y + \epsilon_1]$
3. $cr\log$ *accurate* computes an approximation y of $f(x)$ with a small error bound ϵ_2 with $|\epsilon_2| < |\epsilon_1|$
4. rounding test :$[y - \epsilon_2, y + \epsilon_2]$
5. $cr\log$ *advanced* give us the correctly rounded value.

Fast path

1. Special cases
2. Argument reduction
3. Polynomial approximation and evaluation
4. Result construction

Special cases

- input is NaN
- Input is negatif
- Input is $+0$ or -0
- Input is $+\infty$
- Input is a **subnormal** number. We transform it into **normal** number then we calculate as if it is normal.

Argument reduction

- $x = 2^E \times m$
- $log(x) = E \times \log(2) + \log(m)$
- Condition before to use **Tang**'s algorithm: $1 < m < 2$
  $1.b........b$ with $b \in \{0, 1\}$
- k significant bits after the initial 1, $i = Integer(k.bits)$
  $1.\underbrace{b..b}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$
- $\alpha_i = \frac{2^k}{2^k + i}$
- $m = \frac{m \times \alpha_i}{\alpha_i} \Rightarrow \log(m) = \log(m * \alpha_i) - \log(\alpha_i)$
- $1 < m' < 1 + \frac{1}{2^k}$ with $m' = m \times \alpha_i$
  $1.\underbrace{0..0}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$

Argument reduction

- $x = 2^E \times m$
- $log(x) = E \times \log(2) + \log(m)$
- Condition before to use **Tang**'s algorithm: $1 < m < 2$
  $1.b........b$ with $b \in \{0, 1\}$
- k significant bits after the initial 1, $i = Integer(k.bits)$
  $1.\underbrace{b..b}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$
- $\alpha_i = \frac{2^k}{2^k + i}$
- $m = \frac{m \times \alpha_i}{\alpha_i} \Rightarrow \log(m) = \log(m * \alpha_i) - \log(\alpha_i)$
- $1 < m' < 1 + \frac{1}{2^k}$ with $m^{'} = m \times \alpha_i$
  $1.\underbrace{0..0}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$

Argument reduction

- $x = 2^E \times m$
- $log(x) = E \times \log(2) + \log(m)$
- Condition before to use **Tang**'s algorithm: $1 < m < 2$
  $1.b.......b$ with $b \in \{0, 1\}$
- k significant bits after the initial 1, $i = Integer(k.bits)$
  $1.\underbrace{b..b}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$
- $\alpha_i = \frac{2^k}{2^k + i}$
- $m = \frac{m \times \alpha_i}{\alpha_i} \Rightarrow \log(m) = \log(m * \alpha_i) - \log(\alpha_i)$
- $1 < m' < 1 + \frac{1}{2^k}$ with $m^{'} = m \times \alpha_i$
  $1.\underbrace{0..0}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$

Argument reduction

- $x = 2^E \times m$
- $log(x) = E \times \log(2) + \log(m)$
- Condition before to use **Tang**'s algorithm: $1 < m < 2$
  $1.b.......b$ with $b \in \{0, 1\}$
- k significant bits after the initial 1, $i = Integer(k.bits)$
  $1.\underbrace{b..b}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$
- $\alpha_i = \frac{2^k}{2^k + i}$
- $m = \frac{m \times \alpha_i}{\alpha_i} \Rightarrow \log(m) = \log(m * \alpha_i) - \log(\alpha_i)$
- $1 < m' < 1 + \frac{1}{2^k}$ with $m' = m \times \alpha_i$
  $1.\underbrace{0..0}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$

Argument reduction

- $x = 2^E \times m$
- $log(x) = E \times \log(2) + \log(m)$
- Condition before to use **Tang**'s algorithm: $1 < m < 2$
  $1.b.......b$ with $b \in \{0, 1\}$
- k significant bits after the initial 1, $i = Integer(k.bits)$
  $1.\underbrace{b..b}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$
- $\alpha_i = \frac{2^k}{2^k + i}$
- $m = \frac{m \times \alpha_i}{\alpha_i} \Rightarrow \log(m) = \log(m * \alpha_i) - \log(\alpha_i)$
- $1 < m' < 1 + \frac{1}{2^k}$ with $m' = m \times \alpha_i$
  $1.\underbrace{0..0}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$

Argument reduction

- $x = 2^E \times m$
- $log(x) = E \times \log(2) + \log(m)$
- Condition before to use **Tang**'s algorithm: $1 < m < 2$
  $1.b.......b$ with $b \in \{0, 1\}$
- k significant bits after the initial 1, $i = Integer(k.bits)$
  $1.\underbrace{b..b}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$
- $\alpha_i = \frac{2^k}{2^k + i}$
- $m = \frac{m \times \alpha_i}{\alpha_i} \Rightarrow \log(m) = \log(m * \alpha_i) - \log(\alpha_i)$
- $1 < m' < 1 + \frac{1}{2^k}$ with $m' = m \times \alpha_i$
  $1.\underbrace{0..0}_{k\ \ bits}.....b$ with $b \in \{0, 1\}$

Argument reduction

- $x = 2^E \times m$
- $log(x) = E \times \log(2) + \log(m)$
- Condition before to use **Tang**'s algorithm: $1 < m < 2$
  $1.b........b$ with $b \in \{0,1\}$
- k significant bits after the initial 1, $i = Integer(k.bits)$
  $1.\underbrace{b..b}_{k\ \ bits}.....b$ with $b \in \{0,1\}$
- $\alpha_i = \frac{2^k}{2^k+i}$
- $m = \frac{m \times \alpha_i}{\alpha_i} \Rightarrow \log(m) = \log(m * \alpha_i) - \log(\alpha_i)$
- $1 < m' < 1 + \frac{1}{2^k}$ with $m^{'} = m \times \alpha_i$
  $1.\underbrace{0..0}_{k\ \ bits}.....b$ with $b \in \{0,1\}$

Polynomial approximation and evaluation

- $1 < m' < 1 + \frac{1}{2^k}$
- $\log(1 + t)$
- $t = m' - 1$
- $0 < t \leq \frac{1}{2^k}$

Polynomial approximation and evaluation

- $1 < m' < 1 + \frac{1}{2^k}$
- $\log(1 + t)$
- $t = m' - 1$
- $0 < t \leq \frac{1}{2^k}$

Polynomial approximation and evaluation

- $1 < m' < 1 + \frac{1}{2^k}$
- $\log(1 + t)$
- $t = m' - 1$
- $0 < t \leq \frac{1}{2^k}$

Polynomial approximation and evaluation

- $1 < m' < 1 + \frac{1}{2^k}$
- $\log(1 + t)$
- $t = m' - 1$
- $0 < t \leq \frac{1}{2^k}$

Result construction

- P the Polynomial approximation and evaluation
- $\log(x) = E.\log(2) + P(t) - \log(\alpha_i)$

Operator precision calculation

- $X = \circ(x)$
- relative error $= \frac{X-x}{|x|}$
- $X = x.(1+\epsilon)$ with $|\epsilon| \leq u$.

Operator precision calculation

- $X = \circ(x)$
- relative error $= \frac{X-x}{|x|}$
- $X = x.(1+\epsilon)$ with $|\epsilon| \leq u$.

Operator precision calculation

- $X = \circ(x)$
- relative error $= \frac{X - x}{|x|}$
- $X = x.(1 + \epsilon)$ with $|\epsilon| \leq u$.

Example of operator precision calculation

Algorithme

Algorithm : Add122
*Input : a **Double** number and* (b_h, b_ℓ) ***Double-Double** number*
Condition : $|a| \geq |b_h|$ *and* $|b_\ell| \leq u.|b_h|$
Output : s *and* t *are **Double** numbers.* s *is the main value and* t *is the error value.*

1. $s, \ell = Add112(a, b_h)$
2. $t = \ell + b_\ell$
3. *return* s, t

- $|a| \geq |b_h|$,
- $a > 0$ and $b_h > 0$
- $t = (\ell + b_\ell).(1 + \epsilon)$ with $|\epsilon| \leq u$
- $$t = \ell + b_\ell + \delta$$
  with $\delta = (\ell + b_\ell).\epsilon$
- $$|\delta| \leq |\epsilon|.(|\ell| + |b_\ell|)$$
- $|s + \ell| \geq (1 - u).|s| \Rightarrow (1 - u).|s| \leq |a + b_h| \Rightarrow |s| \leq \frac{1}{1-u}.|a + b_h| \Rightarrow |\ell| \leq \frac{u}{1-u}.|a + b_h|$
- $$|\delta| \leq |\epsilon|.(\frac{u}{1-u}.|a + b_h| + |b_\ell|)$$

Example of operator precision calculation

Algorithme

Algorithm : Add122
*Input : a **Double** number and (b_h, b_ℓ) **Double-Double** number*
Condition : $|a| \geq |b_h|$ and $|b_\ell| \leq u.|b_h|$
*Output : s and t are **Double** numbers. s is the main value and t is the error value.*

1. $s, \ell = Add112(a, b_h)$
2. $t = \ell + b_\ell$
3. *return* s, t

- $|\delta| \leq |\epsilon|.\frac{-u^2+2.u}{1-u}.|a+b_h|$
- As $\frac{-u^2+2.u}{1-u} \leq 2.u$ and $|\epsilon| \leq u$
- $$|\delta| \leq 2.u^2.|a+b_h|$$
- $$|a+b_h+b_\ell| \leq |a+b_h| + |b_\ell|$$
- $$|a+b_h+b_\ell| \geq (1-u)|a+b_h|$$
- $$|\delta| \leq |a+b_h+b_\ell|(\frac{1}{1-u}).2.u^2$$

Example of operator precision calculation

Algorithme

Algorithm : Add122
Input : a ***Double*** *number and* (b_h, b_ℓ) ***Double-Double*** *number*
Condition : $|a| \geq |b_h|$ *and* $|b_\ell| \leq u.|b_h|$
Output : s and t are ***Double*** *numbers. s is the main value and t is the error value.*

1. $s, \ell = Add112(a, b_h)$
2. $t = \ell + b_\ell$
3. *return* s, t

- $$|\delta| \leq 2.u^2.|a + b_h|$$
- $$|a + b_h + b_\ell| \leq |a + b_h| + |b_\ell|$$
- $$|a + b_h + b_\ell| \geq (1 - u)|a + b_h|$$
- $$|\delta| \leq |a + b_h + b_\ell|(\frac{1}{1 - u}).2.u^2$$
- $|\epsilon| \leq \frac{2.u^2}{1-u} \leq 2.u^2$
- $s + t = (a + (b_h + b_\ell))(1 + \epsilon)$ with $|\epsilon| \leq 2.u^2$

$crlog_{fast}$

Algorithme

Algorithm : $crlog_{fast}$ (Part 1)
*Condition x is a **double** number and is not to close to 1.*
Condition m_1 and i are results made after calculating x.
*Condition $table_{\alpha_i}$ $table_{\log(\alpha_i)}$ and are calculated with our method with **Tang** and **Gal's** method.*
*Condition $log(2)$ in **double-double**: $h_{\log 2} = 0x1.62e42fefa38p - 1$ and $l_{\log 2} = 0x1.ef35793c7673p - 45$*
***Input** : x is a **double** number, i is a **Integer** number and m_1 is **double** number.*
*Output : Approximation of $\log(x)$ in **double** number.*

1. $h_r, \ell_r = Mul122(m_1, \alpha_i)$
2. $h, \ell = Add122(-1, h_r, \ell_r)$
3. $F_1 = f_4 + h.(f_5 + h.(f_6 + h.(f_7)))$
4. $hh_2, \ell\ell_2 = Mul222(h, \ell, h, \ell)$
5. $hh_4, \ell\ell_4 = Mul222(hh_2, \ell\ell_2, hh_2, \ell\ell_2)$
6. $ffh_0, ff\ell_0 = Mul122(f_1, h, \ell)$
7. $ffh_3, ff\ell_3 = Mul122(f_3, h, \ell)$
8. $ffh_2, ff\ell_2 = Add122(f_2, ffh_3, ff\ell_3)$
9. $fhx_2, ff\ell x_2 = Mul222(hh_2, \ell\ell_2, ffh_2, ff\ell_2)$
10. $ffhx_4, ff\ell x_4 = Mul122(F_1, hh_4, \ell\ell_4)$
11. $ffhx_0, ff\ell x_0 = Add222(ffh_0, ff\ell_0, ffhx_2, ff\ell x_2)$

$crlog_{fast}$

Algorithme

Algorithm : $crlog_{fast}$ (Part 2)

12. $h_{\mathbf{4}}, \ell_{\mathbf{4}} = Add222(ffhx_{\mathbf{0}}, ff\ell x_{\mathbf{0}}, ffhx_{\mathbf{4}}, ff\ell x_{\mathbf{4}})$
13. $h_{\mathbf{5}}, \ell_{\mathbf{5}} = Add122(\log_{\alpha_i}, h_{\mathbf{4}}, \ell_{\mathbf{4}})$
14. $e_{hlog\mathbf{2}}, e_{\ell log\mathbf{2}} = Mul122(e, h_{\log \mathbf{2}}, \ell_{\log \mathbf{2}})$
15. $h_{\mathbf{6}}, \ell_{\mathbf{6}} = Add222(e_{hlog\mathbf{2}}, e_{\ell log\mathbf{2}}, h_{\mathbf{5}}, \ell_{\mathbf{5}})$
16. *return* $(h_{\mathbf{6}}, \ell_{\mathbf{6}})$

$crlog_{accurate}$

Algorithme

Algorithm : $crlog_{accurate}$
*Condition x is a **double** number and is not to close to 1.*
Condition $m_{\mathbf{1}}$ and i are results made after calculating x.
*Condition $table_{\alpha_i}$ $table_{\log(\alpha_i)}$ are of tables of **Double-Double** and are calculated with **Tang**.*
*Condition $log(2)$ in **double-double**: $h_{\log \mathbf{2}} = 0x1.62e42fefa39efp - 1$ and $l_{\log \mathbf{2}} = 0x1.abc9e3b39803fp - 56$*
*Input : x is a **double** number, i is a **Integer** number and $m_{\mathbf{1}}$ is a **double** number.*
*Output : Approximation of $\log(x)$ in **double** number.*

1. $h_r, \ell_r = Mul122(m_{\mathbf{1}}, \alpha_{i\,h}, \alpha_{i\,\ell})$
2. $h, \ell = Add122(-1, h_r, \ell_r)$
3. $h_{\mathbf{1}}, \ell_{\mathbf{1}} = Mul222(G[0][0], G[0][1], h, \ell)$
4. $for\ (j = 1, j < 11, j++)$
5. $\quad h_{\mathbf{2}}, \ell_{\mathbf{2}} = Add222(G[j][0], G[j][1], h_{\mathbf{1}}, \ell_{\mathbf{1}})$
6. $\quad h_{\mathbf{1}}, \ell_{\mathbf{1}} = Mul222(h_{\mathbf{2}}, \ell_{\mathbf{2}}, h, \ell)$
7. $h_{\mathbf{5}}, \ell_{\mathbf{5}} = Add222(\log_{\alpha_i}, h_{\mathbf{1}}, \ell_{\mathbf{1}})$
8. $e_{hlog\mathbf{2}}, e_{\ell log\mathbf{2}} = Mul122(e, h_{\log \mathbf{2}}, \ell_{\log \mathbf{2}})$
9. $h_{\mathbf{6}}, \ell_{\mathbf{6}} = Add222(e_{hlog\mathbf{2}}, e_{\ell log\mathbf{2}}, h_{\mathbf{5}}, \ell_{\mathbf{5}})$
10. $return\ h_{\mathbf{6}} + \ell_{\mathbf{6}}$

$crlog_{advanced}$

Algorithme

***Algorithm** : $crlog_{advanced}$*
***Input** : x is a **double** number, i is a **Integer** number.*
***Output** : **Approximation** of $\log(x)$ in **double** number.*

1. $h_r, m_r, \ell_r = Mul133(m_1, \alpha_{i\,h}, \alpha_{i\,m}, \alpha_{i\,\ell})$
2. $h, m, \ell = Add133(-1, h_r, \ell_r)$
3. $h_1, m_1, \ell_1 = Mul333(U[0][0], U[0][1], U[0][2], h, m, \ell)$
4. $for\ (j = 1, j < 14, j++)$
5. $\quad h_2, m_2, \ell_2 = Add333(U[j][0], U[j][1], U[j][2], h_1, m_1, \ell_1)$
6. $\quad h_1, m_1, \ell_1 = Mul333(h_2, m_2, \ell_2, h, m, \ell)$
7. $h_5, m_5, \ell_5 = Add333(\log_{\alpha_i\ h}, \log_{\alpha_i\ m}, \log_{\alpha_i\ \ell}, h_1, m_1, \ell_1)$
8. $e_{hlog2}, e_{mlog2}, e_{\ell log2} = Mul133(e, h_{\log 2}, m_{\log 2}, \ell_{\log 2})$
9. $h_6, m_6, \ell_6 = Add333(e_{hlog2}, e_{mlog2}, e_{\ell log2}, h_5, m_5, \ell_5)$
10. $m_\ell = m_6 + \ell_6$
11. $return\ h_6 + m_\ell$

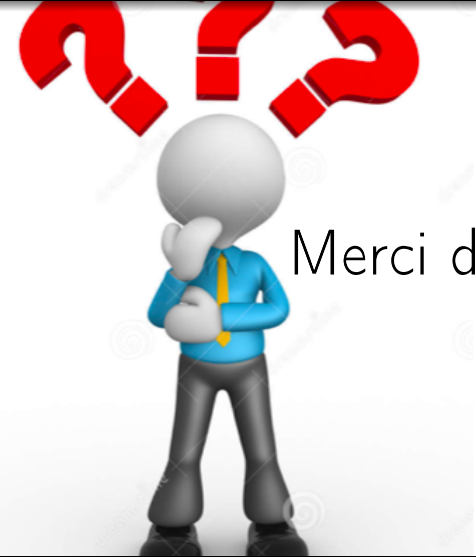
log

Algorithme

Algorithm: log
Condition $:x$ ***is a double number.***
Condition $:err_{fast} = 0x1.6b6b11ea279ecp - 59$ ***and*** $err_{accurate} = 0x1.810c9a86fc45ep - 99$
Output $:\log(x)$ ***in double number.***

1. $h_{\mathbf{6}}, \ell_{\mathbf{6}} = cr\log_{fast}(x)$
2. $left = h_{\mathbf{6}} + FMA(-err_{fast}, h_{\mathbf{6}}, \ell_{\mathbf{6}})$
3. $right = h_{\mathbf{6}} + FMA(+err_{fast}, h_{\mathbf{6}}, \ell_{\mathbf{6}})$
4. $if(right == left)$
5. ***return*** $left$
6. $h_{\mathbf{6}}, l_{\mathbf{6}} = cr\log_{accurate}(x)$
7. $left = h_{\mathbf{6}} + FMA(-err_{fast}, h_{\mathbf{6}}, \ell_{\mathbf{6}})$
8. $right = h_{\mathbf{6}} + FMA(+err_{accurate}, h_{\mathbf{6}}, \ell_{\mathbf{6}})$
9. $if(right == left)$
10. ***return*** $left$
11. ***return*** $cr\log_{advanced}(x)$

- This code has been verified thanks to **Sage** and the **MPFR library** in **C**.
- ./perf.sh log:
  84.164 :number of cycles of our logarithm
  15.918 : number of cycles of logarithm of system library (**GNU libc**).
- More information about COREMATH project:
  `https://core-math.gitlabpages.inria.fr`



Merci de votre attention!