TSTE18 Digital Arithmetic
Seminar 12

Oscar Gustafsson

Alternative number representations

- Residue Number Systems
- Logarithmic Number Systems
- Decimal Number Systems

Residue Number Systems

- The idea is to use the residues of the numbers and perform operations on the residues
- Also called modular arithmetic since the residues are computed using the modulo function
- The residues will have shorter wordlengths so the computations are potentially faster
- Based on a set of relative prime modulis $\{M_1, M_2, \dots, M_K\}$
- Relative prime so, $\gcd(M_p, M_q) = 1,\ \forall 1 \leq p \neq q \leq N$
- The range of uniquely representable numbers is $R = \prod_{k=1}^{K} M_k$
- Can represent any continuous range of R numbers

Residue Number Systems

- First, define
$$A \text{ div } M = \left\lfloor \frac{A}{M} \right\rfloor$$
and
$$R = A \bmod M = A - M(A \text{ div } M)$$
as the operations leading to the integer and remainder part of an integer division, respectively
- Denote $X \bmod M$ as $\langle X \rangle_M$
- A basic identity is $\langle A \star B \rangle_M = \langle \langle A \rangle_M \star \langle B \rangle_M \rangle_M$ where $\star$ can be $+, -$ or $\times$
- This means that we can perform the modular reduction at will
- Modular division can be performed using the multiplicative inverse defined as $X^{-1} = A$ such that $\langle A \times X \rangle_M = 1$
- A modular inverse of A exists iff $\langle A \rangle_M \neq 0$ and $\gcd(A, M) = 1$

Residue Number Systems

- To determine the residue one can perform a normal integer division with remainder
- However, this is often to complex and faster methods exists for specific types of moduli
- For a moduli which is a power of the radix, e.g., 2^N it is enough to take the N least significant bits (digits)
- This also holds for two's (radix) complement representations
- Other commonly used modulis are on the form $2^N \pm 1$
- Here we can use

$$\langle A \rangle_{M-1} = A \bmod (M-1) = ((A \bmod M)+(A \operatorname{div} M)) \bmod (M-1)$$

and

$$\langle A \rangle_{M+1} = A \bmod (M+1) = ((A \bmod M)-(A \operatorname{div} M)) \bmod (M+1)$$

Residue Number Systems

- Example: reduce $(755)_{10} = (1011110011)_2$ using the modulis $7, 8, 9$

Residue Number Systems

- Example: operations modulo 3
- Addition $\langle A + B \rangle_3$

		A		
		0	1	2
	0	0	1	2
B	1	1	2	0
	2	2	0	1

- Subtraction $\langle A - B \rangle_3$

		A		
		0	1	2
	0	0	1	2
B	1	2	0	1
	2	1	2	0

Residue Number Systems

- Multiplication $\langle A \times B \rangle_3$

		A		
		0	1	2
	0	0	0	0
B	1	0	1	2
	2	0	2	1

- Multiplicative inverse $\langle A^{-1} \rangle_3$

	A	
0	1	2
-	1	2

Residue Number Systems

- Realization of modular additions and subtractions
- For $M = 2^N$ it is possible to simply use the N least significant bits of the addition/subtraction
- For an arbitrary M with $2^{N-1} < M < 2^N$ we first note that if $A + B \geq M$ then $\langle A + B \rangle_M = A + B - M$
- This condition is equivalent to $A + B + (2^N - M) \geq 2^N$ and $\langle A + B \rangle_M = A + B - M = \left\langle A + B + (2^N - M) \right\rangle_{2^N}$
- Therefore compute in parallel $C = \langle A + B \rangle_{2^N}$ and $C' = A + B + (2^N - M)$
- The correct output is $\langle C' \rangle_{2^N}$ if C' div $2^N = 1$, i.e., $A + B + (2^N - M) \geq 2^N$, i.e., the output carry bit is one, otherwise the correct output is C

Residue Number Systems

- Especially for $M = 2^N - 1$ the correct result is either $A + B$ or $\langle A + B + 1 \rangle_{2^N}$
- This can be realized by performing $A + B$ and then take the output carry and add at the LSB position
- A multiplication is an addition of shifted terms, so this can be realized either by performing modular additions of the partial product results or by performing a normal multiplication followed by a modular reduction
- It should be noted that quantization is not possible (in a straightforward way) using a residue number system, so the moduli set must be selected such that the moduli set range can cover all possible results and that moduli set should be used throughout all computations

Residue Number Systems

- Example: add 5 and 3 using the modulis $7, 8, 9$

Residue Number Systems

- Conversion from RNS to binary can be performed using the Chinese Remainder Theorem
- Recall the range $R = \prod_{k=1}^{K} M_k$
- For an RNS number $\{A_1, A_2, \ldots, A_K\}$ in a moduli set $\{M_1, M_2, \ldots, M_K\}$ the corresponding binary representation can be computed as

$$\langle A \rangle_R = \left\langle \sum_{k=1}^{K} R_k \left\langle \frac{A_k}{R_k} \right\rangle_{M_k} \right\rangle_R$$

where $R_k = R/M_k$

Residue Number Systems

- Example: Convert $\{3, 4, 5\}$ in the moduli set $\{7, 8, 9\}$ to radix-2

Residue Number Systems

- Example: Compute $269 \times 59 + 4793$ in the moduli set $\{31, 32, 33\}$ and convert back to radix-2

Residue Number Systems

- Challenging functions in RNS includes
  - Reduce the magnitude/scale
  - Range increase: can be handled by adding more modulis
  - Comparison
  - Sign-detection

Logarithmic Number Systems

- In a base-L logarithmic number system (LNS) a number is represented by the corresponding exponent as

$$A = (-1)^{S_A} L^{E_A} \Rightarrow E_A = \log_L |A|,\ S_A = \text{ sign}(A)$$

- Without loss of generality we will assume $L = 2$ and not explicitly state the base
- The LNS can be seen as a floating-point number without the significand
- The exponent typically consists of both an integer and a fractional part
- Large dynamic range and approximately constant relative representation error
- We need one more bit to represent a zero value

Logarithmic Number Systems

- Multiplication and division become additions/subtractions:

$$C = A \times B \Rightarrow E_C = E_A + E_B,\ \ C = A/B \Rightarrow E_C = E_A - E_B$$

- Squaring and square roots become shifts:

$$B = A^2 \Rightarrow E_B = 2E_A,\ \ B = \sqrt{A} \Rightarrow E_B = E_A/2$$

- Exponents becomes multiplication:

$$C = A^B \Rightarrow E_C = BE_A$$

Logarithmic Number Systems

- Addition and subtraction become more complicated (assuming $|A| > |B|$):

$$C = A \pm B \Rightarrow E_C = \log|A \pm B| = \log\left|A\left(1 \pm \frac{B}{A}\right)\right|$$

$$= E_A + \log\left|1 \pm 2^{E_B - E_A}\right| = E_A + \Phi(E_B - E_A)$$

- In general we get

$$E_C = \max\{E_A, E_B\} + \Phi(-|E_A - E_B|)$$

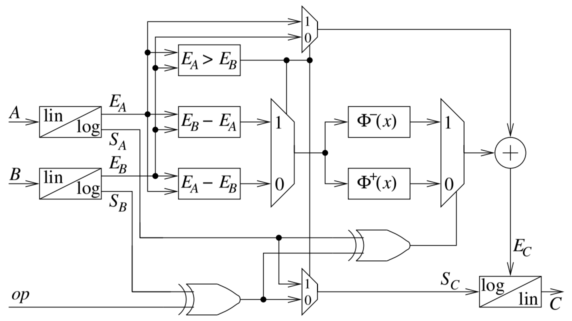
where

$$\Phi = \begin{cases} \Phi^+(x) = \log|1 + 2^x|, & S_\Phi = 0 \\ \Phi^-(x) = \log|1 - 2^x|, & S_\Phi = 1 \end{cases}$$

and $S_\Phi = S_A \oplus S_B \oplus (\overline{\text{add}}/\text{sub})$

Logarithmic Number Systems

- An LNS adder/subtracter require both functions $\Phi^+(x)$ and $\Phi^-(x)$



- Can perform both + and − at the same time as the operations will always use different tables

Arithmetic Operations Via LNS

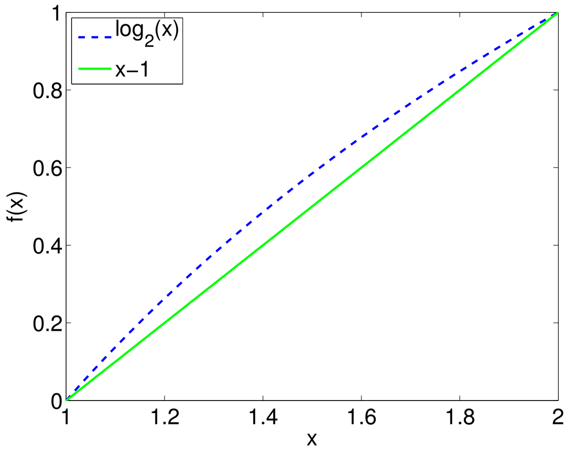
- It has been suggested to use LNS as a way to approximate multiplication, division etc.
- A simple way to determine the binary (base-2) logarithm of a number is to count the number of leading zeros forming the integer part of the logarithm and then determine the fractional part $\log_2(1+m) < 1$ for $0 \leq m < 1$

$$x = 2^e(1+m) \Rightarrow \log_2(N) = e + \log_2(1+m) \qquad (1)$$

- A reverse conversion (computing the exponential) is performed by determining 2^m where m is the fractional part of the logarithm and then shift according to the integer part
- A simple approximation (often referred to as Mitchell's logarithm) is $\log_2(1+m) \approx m$
- Similarly $2^m \approx 1 + m$, i.e., concatenating a one before the fractional bits

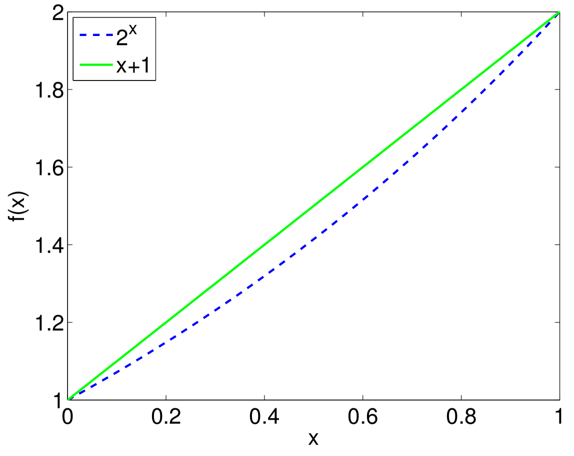
Arithmetic Operations Via LNS

- The approximation $\log_2(1+m) \approx m$ can be seen as a linear approximation of the function



Arithmetic Operations Via LNS

- The same holds for the approximation $2^m \approx 1+m$



Arithmetic Operations Via LNS

- Example: consider the multiplication 0.00111 $\times$ 0.01011
- We get $\log_2(0.00110) \approx 1101(=-3)+0.11 = 1101.110$ and $\log_2(0.01011) \approx 1110(=-2)+0.011 = 1110.011$
- Adding the logarithms results in $1101.110 + 1110.011 = 1100.001$, leading to the final result 0.0001001
- The exact answer is 0.0001011011
- Example: Consider computing the square root of 0.0011101
- $\log_2(0.0011101) = 101.1101$, which when diving by two gives $\log_2(0.0011101) = 110.11101$ and a final result of 0.0111101
- The correct (rounded) result is exactly the same in this case

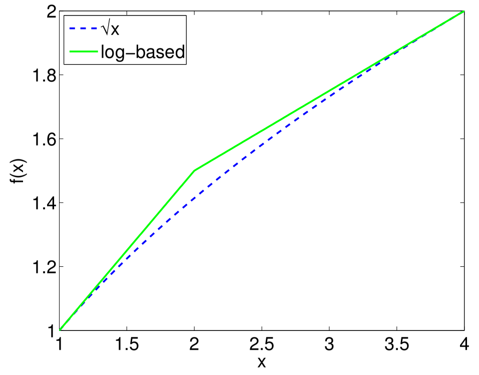
Arithmetic Operations Via LNS

- Consider the square root computation in the range $1 \le x < 2$

$$\sqrt{x} = 2^{\frac{\log_2(x)}{2}} \approx 2^{\frac{x-1}{2}} \approx 1 + \frac{x-1}{2} = \frac{x+1}{2}$$

- In the range $2 \le x < 4$ (equivalent to $\frac{1}{2} \le x \le 2$)

$$\sqrt{x} = 2^{\frac{\log_2(x)}{2}} \approx 2^{\frac{1+x-1}{2}} \approx 1 + \frac{x}{2}$$



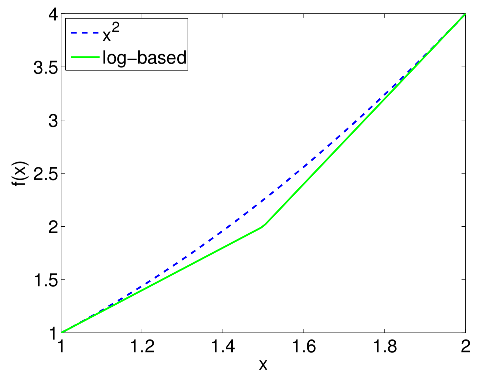
Arithmetic Operations Via LNS

- Consider the square computation with $1 \le x < 2$

$$x^2 = 2^{2\log_2(x)} \approx 2^{2(x-1)} = 2^{2x-2}$$

- $0 \le 2x - 2 < 2$ so split in two regions

$$\begin{aligned} 1 + (2x-2) &= 2x - 1, \quad x \le 1.5 \\ 2(1 + (2x-3)) &= 4x - 4, \quad x \ge 1.5 \end{aligned}$$



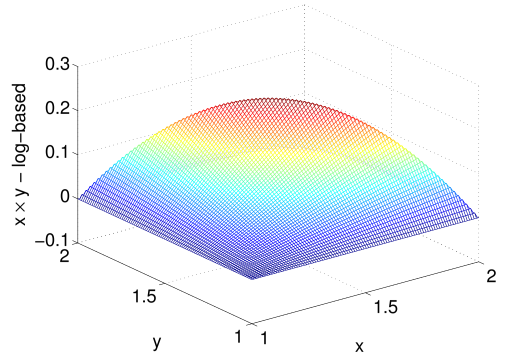
Arithmetic Operations Via LNS

- For multiplication with $1 \le x, y < 2$

$$xy = 2^{\log_2(x)+\log_2(y)} \approx 2^{(x-1)+(y-1)} = 2^{x+y-2}$$

- $0 \le x + y - 2 < 2$ so split in two regions

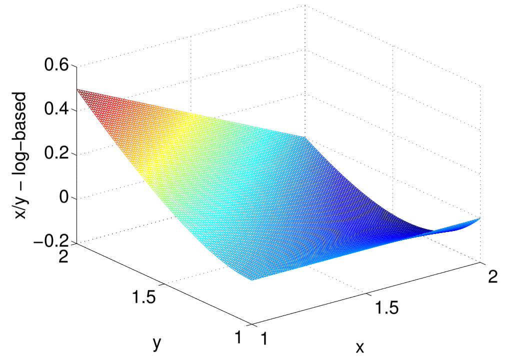
$$\begin{aligned} 1 + (x+y-2) &= x + y - 1, \quad x + y - 2 \le 1 \\ 2(1 + (x+y-3)) &= 2x + 2y - 4, \quad x + y - 2 \ge 1 \end{aligned}$$



Arithmetic Operations Via LNS

- For division with $1 \le x, y < 2$

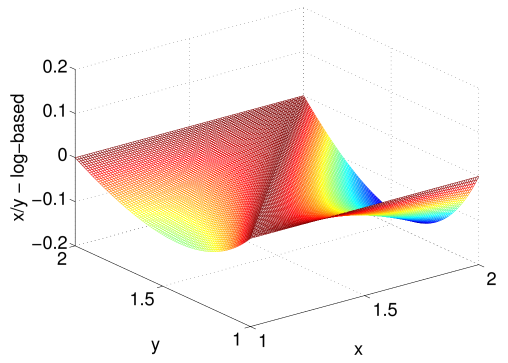
$$\frac{x}{y} = 2^{\log_2(x)-\log_2(y)} \approx 2^{(x-1)-(y-1)} = 2^{x-y} \approx 1 + x - y$$



Arithmetic Operations Via LNS

- Alternatively, as $2^m \approx 1 + m$ deviates more for negative numbers close to minus one,

$$\begin{aligned} 1 + (x-y) &= 1 + x - y, \quad x \ge y \\ \tfrac{1}{2}(1 + (x-y+1)) &= 1 + \tfrac{x-y}{2}, \quad x \le y \end{aligned}$$



Logarithmic Number Systems

- If the application has a high dynamic range and/or many multiplications/division/exponentiations the LNS can be attractive
- The cost is conversion to and from LNS as well as the costly additions/subtractions

Decimal Number Systems

- In some applications (read money) the problem of representing decimal numbers using radix-2 representations is really unwanted
- You do not want to deposit 14.15 SEK to your bank account already containing 78.54 SEK and end up with 92.68999567 SEK
- Some observations related to this:
  - Decimal numbers can be represented in BCD, requiring four bits per decimal digit, or in groups of three digits using 10 bits (max 1024)
  - IEEE 754 defines three different decimal formats: `decimal32`, `decimal64`, and `decimal128` containing 7, 16, and 34 decimal digits for the significand
  - Two different ways to represent the significand
  - The representation can be seen as in general de-normalized (implicit prefix = 0)
  - However, for efficient use of the bits the exponent and signficand share some of the bits, leading to an implicit prefix of the significand in some cases