Representations for fast arithmetic

Intensive Computation

Annalisa Massini
2019-2020

Lecture 17

Efficient number representations

- Representations different from the binary and 2’s complement representations are studied to obtain a faster arithmetic
- We need to consider the impact of changing representation on:
  - ***Standard operations of ALU:***
    - Zero recognition
    - Arithmetic comparison
    - Sign detection
  - ***Conversions:***
    - Forward conversion from binary to the new representation
    - Reverse conversion from the new representation to binary
- We consider the following examples of representation:
  - Redundant representations
  - Residue number systems

REDUNDANT NUMBER SYSTEMS

Redundant number systems

- Conventional radix-r systems use [0, r-1] digit set

  radix-10 → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- If the digit set (in radix-r system) contains more than **r** digits, the system is **redundant**
  - radix-2 → 0, 1, 2 or -1, 0, 1
  - radix-10 → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
  - radix-10 → -6, -5,- 4, -3, -2, -1, 0, 1, 2, 3, 4, 5
- **Redundancy** may result from adopting the digit set wider than radix and the number interpretation is conventional
- **Redundancy** – representation of numbers is *not unique*

Signed-digit numbers

- A **radix-r redundant signed-digit** number system is based on digit set S = {- β, -(β - 1), … , -1, 0, 1, … , α},

  where $1 \le \alpha, \beta \le r - 1$

- The digit set S contains more than r values → multiple representations for any number in signed digit format → redundant
- A symmetric signed digit has α = β

- **Carry-free addition** is an attractive property of redundant signed-digit numbers

Signed-digit numbers

- All digits have weights r^p (p-position, r-radix)
- Digits can have **signed values**
- Any set digit $[-\alpha, \beta]$ including 0, can be used
- If $\alpha+\beta+1 > r$ the numbering system is **redundant**

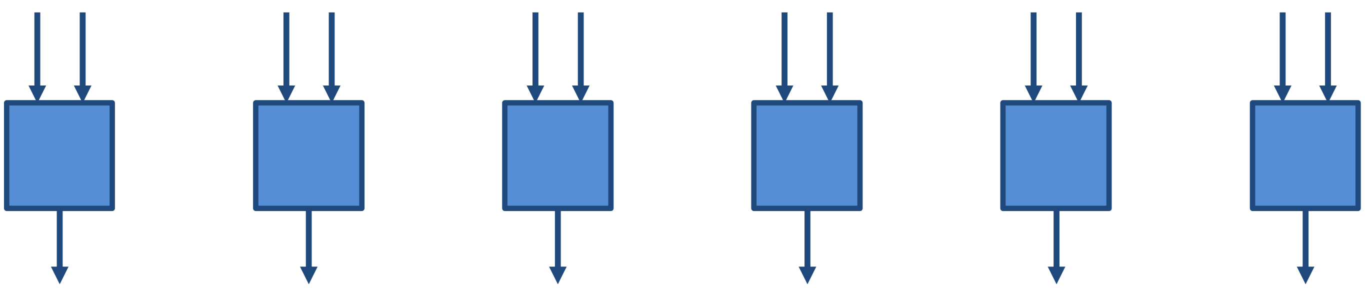
[-1,1] radix-2	→	1 -1 0 -1 0 = $6_{(10)}$	and 0 1 -1 1 0 = $6_{(10)}$
[-1,3] radix-4	→	1 -1 2 0 3 = $227_{(10)}$	and 1 -1 2 1 -1 = $227_{(10)}$
1111 (2's compl.)	→	-1 1 1 1 = -1	

Carry-free addition is an attractive property of redundant signed-digit numbers

Redundant number systems

- **Carry-free** addition → no carry propagation
- All digit additions can be done **simultaneously**
- Carry-free addition is possible with widening of the digit set

	1	2	3	4	5	6	radix-10, digit set [0,9]
+	4	5	6	7	8	9	radix-10, digit set [0,9]
	5	7	9	11	13	15	radix-10, digit set [0,18]



Redundant number systems

- Reduction of digit set by **carry propagation by only one position**

		11	9	17	10	12	18	radix-10, digit set [0,18]
		6	12	9	10	8	18	radix-10, digit set [0,18]
+		17	21	26	20	20	36	radix-10, digit set [0,36]
		↓	↓	↓	↓	↓	↓	
		7	11	16	0	10	16	Intermediate sums [0,16]
	↙	↙	↙	↙	↙	↙		
	1	1	1	2	1	2		Transfer digit set [0,2]
	1	8	12	18	1	12	16	sum [0,18]

Signed digit representation

In summary:

- **Signed-digit representation** is a **positional system** with signed digits
- The representation may not be unique
- Signed-digit representation can be used to accomplish fast addition of integers because it can eliminate chains of dependent carries

MODIFIED SIGNED DIGIT REPRESENTATION

A. K. Cherri, M. A. Karim, “Modified-signed digit arithmetic using an efficient symbolic substitution”, Appl. Opt. (1988)

Modified signed digit representation

- The set of digit is $\{-1,0,1\} = \{\bar{1},0,1\}$

- The representation is **not unique**:

$$\bar{1}01\bar{1} = -8+2-1 = -7$$
$$\bar{1}001 = -8+1 = -7$$
$$\bar{1}1\bar{1}\bar{1} = -8+4-2-1 = -7$$

- The number of possible representation depends on the length of the sequence of digits

- To perform the addition, **truth table** are used

Modified signed digit representation

- Truth tables

		First addend		
		-1	0	1
Second addend	-1	0 -1	1 -1	0 0
	0	1 -1	0 0	-1 1
	1	0 0	-1 1	0 1

		First addend		
		-1	0	1
Second addend	-1	0 -1	-1 0	0 0
	0	-1 0	0 0	1 0
	1	0 0	1 0	0 1

- Three steps are needed to obtain the sum
  - **Left table** is applied in **step 1 and 3**
  - **Right table** is applied in **step 2**
- Output: lower row → **sum** – upper row → **complemented sum**

Modified signed digit representation

- Example

$$\begin{array}{ccccccc} 1 & \bar{1} & 0 & 1 & \bar{1} & \quad & 9 \\ \bar{1} & 1 & \bar{1} & 1 & 0 & \quad & -10 \end{array}$$

		First addend		
		-1	0	1
Second addend	-1	0 -1	1 -1	0 0
	0	1 -1	0 0	-1 1
	1	0 0	-1 1	0 1

		First addend		
		-1	0	1
Second addend	-1	0 -1	-1 0	0 0
	0	-1 0	0 0	1 0
	1	0 0	1 0	0 1

Modified signed digit representation

- Example

$$
\begin{array}{cccccccc}
 & 1 & \bar{1} & 0 & 1 & \bar{1} & & 9 \\
 & \bar{1} & 1 & \bar{1} & 1 & 0 & & -10 \\
\hline
 & 0 & 0 & 1 & 0 & 1 & & \\
0 & 0 & \bar{1} & 1 & \bar{1} & \mathit{0} & &
\end{array}
$$

		First addend		
		-1	0	1
Second addend	-1	0 -1	1 -1	0 0
	0	1 -1	0 0	-1 1
	1	0 0	-1 1	0 1

		First addend		
		-1	0	1
Second addend	-1	0 -1	-1 0	0 0
	0	-1 0	0 0	1 0
	1	0 0	1 0	0 1

Modified signed digit representation

- Example

$$
\begin{array}{cccccccc}
 & 1 & \bar{1} & 0 & 1 & \bar{1} & & 9 \\
 & \bar{1} & 1 & \bar{1} & 1 & 0 & & -10 \\
\hline
 & 0 & 0 & 1 & 0 & 1 & & \\
0 & 0 & \bar{1} & 1 & \bar{1} & \mathit{0} & & \\
\hline
 & 0 & \bar{1} & 0 & \bar{1} & 1 & & \\
0 & 0 & 1 & 0 & 0 & \mathit{0} & &
\end{array}
$$

		First addend		
		-1	0	1
Second addend	-1	0 -1	1 -1	0 0
	0	1 -1	0 0	-1 1
	1	0 0	-1 1	0 1

		First addend		
		-1	0	1
Second addend	-1	0 -1	-1 0	0 0
	0	-1 0	0 0	1 0
	1	0 0	1 0	0 1

Modified signed digit representation

- Example

$$
\begin{array}{cccccccc}
 & 1 & \bar{1} & 0 & 1 & \bar{1} & & 9 \\
 & \bar{1} & 1 & \bar{1} & 1 & 0 & & -10 \\
\hline
 & 0 & 0 & 1 & 0 & 1 & & \\
0 & 0 & \bar{1} & 1 & \bar{1} & \mathit{0} & & \\
\hline
 & 0 & \bar{1} & 0 & \bar{1} & 1 & & \\
0 & 0 & 1 & 0 & 0 & \mathit{0} & & \\
\hline
 & 0 & 0 & 0 & 1 & \bar{1} & & 1 \\
0 & 0 & 0 & \bar{1} & 1 & & & -1
\end{array}
$$

		First addend		
		-1	0	1
Second addend	-1	0 -1	1 -1	0 0
	0	1 -1	0 0	-1 1
	1	0 0	-1 1	0 1

		First addend		
		-1	0	1
Second addend	-1	0 -1	-1 0	0 0
	0	-1 0	0 0	1 0
	1	0 0	1 0	0 1

RB - REDUNDANT BINARY NUMBER REPRESENTATION

G. A. De Biase, A. Massini “Redundant binary number representation for an inherently parallel arithmetic on optical computers”,

Appl. Opt., 32 (1993)

RB - Redundant Binary Representation

- An integer D obtained by

$$D = \sum_{i=0}^{n-1} a_i 2^{i - \lceil i/2 \rceil}$$

- This weight sequence characterizes the RB number representation and is:

$$\begin{array}{ccccccccc} \cdots & 8 & 8 & 4 & 4 & 2 & 2 & 1 & 1 \\ & r & n & r & n & r & n & r & n \end{array}$$

- ***All position weights are doubled***: the left digit is called *r* (*redundant*) and the right digit *n* (*normal*)

RB - Redundant Binary Representation

- **RB representation** of a number can be obtained from its binary representation by the following recoding rules:

  $0 \rightarrow 00 \qquad\qquad 1 \rightarrow 01$

- The RB number obtained in this way is in **canonical form**
- This coding operation is performable in parallel in constant time (one elemental logic step)

RB - Redundant Binary Representation

- Each RB number has a canonical form and several redundant representations

- Examples of ***unsigned*** RB numbers (canonical and redundant)

0	000	000000			
1	001	000001	000010		
2	010	000100	001000	000011	
3	011	000101	001001	001010	
4	100	010000	100000	001100	000111
5	101	010001	010010	100001	100010
6	110	010100	011000	101000	010011
7	111	010101	010110	101001	101010

Table for addition

- Addition is performed using a truth table

	00	01	10	11
00	00 00	10 00	00 01	10 01
01	00 01	10 01	00 10	10 10
10	00 01	10 01	00 10	10 10
11	00 10	10 10	00 11	10 11

Table for addition

- **Two steps**: parallel application of the table on all *rn* pairs
- Output: **sum** on the lower row and **zero** on the upper row

	00	01	10	11
00	00 00	10 00	00 01	10 01
01	00 01	10 01	00 10	10 10
10	00 01	10 01	00 10	10 10
11	00 10	10 10	00 11	10 11

RB - Redundant Binary Representation

- Example

0	0	0	1	0	1	1	1	8
0	0	1	1	0	1	1	0	11

	00	01	10	11
00	00 00	10 00	00 01	10 01
01	00 01	10 01	00 10	10 10
10	00 01	10 01	00 10	10 10
11	00 10	10 10	00 11	10 11

RB - Redundant Binary Representation

- Example

0	0	0	1	0	1	1	1	8
0	0	1	1	0	1	1	0	11
0	0	1	0	1	0	1	0	
0	1	0	0	1	1	0	*0*	

	00	01	10	11
00	00 00	10 00	00 01	10 01
01	00 01	10 01	00 10	10 10
10	00 01	10 01	00 10	10 10
11	00 10	10 10	00 11	10 11

RB - Redundant Binary Representation

- Example

0	0	0	1	0	1	1	1	8
0	0	1	1	0	1	1	0	11
0	0	1	0	1	0	1	0	*7*
0	1	0	0	1	1	0	*0*	*12*
0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	1	*0*	19

	00	01	10	11
00	00 00	10 00	00 01	10 01
01	00 01	10 01	00 10	10 10
10	00 01	10 01	00 10	10 10
11	00 10	10 10	00 11	10 11

RB - Redundant Binary Representation

- In analogy with the 2's complement binary system, a **signed RB number** is obtained by

$$D = -\sum_{i=n-2}^{n-1} a_i 2^{i-\lceil i/2 \rceil} + \sum_{i=0}^{n-3} a_i 2^{i-\lceil i/2 \rceil} \qquad n \text{ even}$$

- The same procedure of the addition of two unsigned RB numbers obtains the algebraic sum of two signed RB numbers

RB - Redundant Binary Representation

- The additive inverse of an RB number is obtained by
  - following a procedure similar to that used in the 2's complement number system
  - taking into account that the negation of all RB representations of the number 0 is $(-2)_{10}$ whereas in the 2's complement binary system it is $(-1)_{10}$

- ***Procedure***
  - Step 1 - all digits of the RB number are complemented
  - Step 2 - algebraic sum between the RB canonical form of $(2)_{10}$ and the RB number
  - The output is the **additive inverse** of the considered RB number

RB - Redundant Binary Representation

- The **decoding** of RB numbers, with the correct truncation, can be performed with the following procedure that derives directly from the RB number definition

- ***Procedure***
  - The input is RB*n* and RB*r*
  - Binary addition RB*n* + RBr.
  - Only the first n/2 bits are considered
  - The output is the corresponding binary or 2's complement binary number

RB - Redundant Binary Representation

- **Zero and its detection**
- In the case of unsigned RB numbers the $(0)_{10}$ has only the RB canonical form and is easily detectable
- In the case of signed RB numbers, $(0)_{10}$ has many RB representations
- Example for six-digit signed RB numbers:

(000000)	(101011)	(101100)
(100111)	(010111)	(011100)

- The difficulty in detecting the $(0)_{10}$ can be overcome by using the number $(-1)_{10}$

RB - Redundant Binary Representation

- **Zero and its detection**
- In fact, any redundant representation of the number $(-1)_{10}$ is composed by pairs 01 or 10
- The canonical representation of the $(-1)_{10}$ can be obtained if the following rules are applied on all the *rn* pairs

$$10 \rightarrow 01 \qquad 01 \rightarrow 01$$

- Then, a RB number is a representation of $(0)_{10}$ if the result of an algebraic sum between an RB number and an RB representation of $(-1)_{10}$ is an RB representation of the number $(-1)_{10}$ again,

RB - Redundant Binary Representation

- **Zero and its detection**
- Then the procedure to detect the number $(0)_{10}$ is the following

Procedure

- Input - an RB number
- Step 1 - algebraic sum between the RB canonical form of $(-1)_{10}$ and the RB number
- Step 2 - application of rules to the result
- Output - the RB canonical form of $(-1)_{10}$ or of another RB number

RESIDUE NUMBER SYSTEM

Residue number systems

- Residue number systems are based on the *congruence* relation:
  - Two integers *a* and *b* are said to be ***congruent modulo m*** if *m* divides exactly the difference of *a* and *b*
  - We write $a \equiv b \pmod{m}$
- For example
  - $10 \equiv 7 \pmod{3}$
  - $10 \equiv 4 \pmod{3}$
  - $10 \equiv 1 \pmod{3}$
  - $10 \equiv -2 \pmod{3}$
- The number *m* is a *modulus* or *base*, and we assume that its values exclude 1, which produces only trivial congruences

Residue number systems

- In fact:
- If ***q*** and ***r*** are the **quotient** and **remainder**, respectively, of the integer division of *a* by *m* - that is: $a = q{:}m + r$

  → then, by definition, we have $a \equiv r \pmod{m}$

- The number ***r*** is said to be the ***residue*** of ***a*** with respect to ***m***, and we shall usually denote this by $r = |a|_m$

- The set of *m* smallest values, $\{0; 1; 2; \ldots ; m - 1\}$, that the residue may assume is called the set of *least positive residues modulo m*

Residue number systems

- Suppose we have a set $\{m_1; m_2; \ldots; m_N\}$ of N positive and pairwise **relatively prime** moduli
- Let M be the product of the moduli $M=m_1 x m_2 x \ldots x m_N$
- $[0; M\text{-}1]$ is the range of representation

- We write the representation in the form $<x_1; x_2; \ldots; x_N>$, where $x_i = |X|_{mi}$, and we indicate the relationship between X and its residues by writing $X \approx <x_1; x_2; \ldots; x_N>$

- Example: in the residue system {2, 3, 5}, M=30 and

  8 → <0, 2, 3>

  16 → <0, 1, 1>

Residue number systems

- Every number $X < M$ has a **unique representation** in the residue number system, which is the sequence of residues $<|X|_{mi} : 1 \leq i \leq N>$
- A partial proof of uniqueness is as follows:
  - Suppose X_1 and X_2 are two different numbers with the **same *residue representation***
  - Then $|X_1|_{mi} = |X_2|_{mi}$, and so $|X_1 - X_2|_{mi} = 0$
  - Therefore $X_1 - X_2$ is the least common multiple (**lcm**) of m_i
  - But if the *mi* are relatively prime, then their **lcm** is *M*, and it must be that $X_1 - X_2$ is a multiple of *M*
  - So it cannot be that $X_1 < M$ and $X_2 < M$
  - Therefore, the representation $<|X|_{mi} : 1 \leq i \leq N>$ is unique and may be taken as the representation of *X*

Residue number systems

- The number M is called the *dynamic range* of the RNS, because the number of numbers that can be represented is M
- For unsigned numbers, that range is $[0; M - 1]$
- Representations in a system in which the moduli are not pairwise relatively prime will be not be unique: two or more numbers will have the same representation

	Relatively prime			Relatively non-prime		
N	m1=2	m2=3	m3=5	m1=2	m2=4	m3=6
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	0	2	2	0	2	2
3	1	0	3	1	3	3
4	0	1	4	0	0	4
5	1	2	0	1	1	5
6	0	0	1	0	2	0
7	1	1	2	1	3	1
8	0	2	3	0	0	2
9	1	0	4	1	1	3
10	0	1	0	0	2	4
11	1	2	1	1	3	5
12	0	0	2	0	0	0
13	1	1	3	1	1	1
14	0	2	4	0	2	2
15	1	0	0	1	3	3

Residue number systems

- The computation of the residues in the case of negative numbers is obtained by complementing the residues:

$$\langle X \rangle_{m_i} = \begin{cases} \langle X \rangle_{m_i} & \text{if } X \geq 0 \\ \left\langle m_i - \langle |X| \rangle_{m_i} \right\rangle_{m_i} & \text{if } X < 0 \end{cases}$$

	m1=2	m2=3	m3=5	
0	0	0	0	
1	1	1	1	
2	0	2	2	
3	1	0	3	
4	0	1	4	
5	1	2	0	
…	….	….	….	
14	0	2	4	
15	1	0	0	
16	0	1	1	-14
17	1	2	2	-13
18	0	0	3	-12
…	….	….	….	…

Residue number systems

- Ignoring other, more *practical*, issues, the best moduli are probably **prime numbers**

- For computer applications, it is important to have moduli-sets that facilitate both efficient representation and balance, meaning that the *differences between the moduli should be as small as possible*

Residue number systems

- Take, for example, the choice of 13 and 17 for the moduli that are adjacent prime numbers
- The dynamic range is 221
- With a straightforward binary encoding:
  - 4 bits will be required to represent 13
  - 5 bits will be required to represent 17

Residue number systems

- The representational efficiency is:
  - In the first case 13/16
  - In the second case is 17/32

- If instead we chose 13 and 16, then the representational efficiency:
  - is improved to 16/16 in the second case
  - but at the cost of **reduction in the range** (down to 208)

- With the better balanced pair, 15 and 16, we would have:
  - a better efficiency 15/16 and 16/16
  - A greater range: 240

Residue number systems

- It is also useful to have ***moduli that simplify*** the implementation of the ***arithmetic operations***

- This means that arithmetic on residue digits should ***not deviate too far from conventional arithmetic***, which is just **arithmetic modulo a power of 2**

- A common choice of prime modulus that does not complicate arithmetic and which has good representational efficiency is $m_i = 2^i - 1$

Residue number systems

- Not all pairs of numbers of the form $2^i - 1$ are relatively prime
- It can be shown that that 2^j - 1 and 2^k - 1 are relatively prime **if and only if** *j* and *k* are relatively prime

- For example:
  - 2^4-1= 15 15=3x5
  - 2^5-1= 31 31 *prime*
  - 2^6-1= 63 63=3x7
  - 2^7-1= 127 127 *prime*
  - 2^8-1= 255 255=3x5x17

Residue number systems

- Many moduli sets are based on these choices, but there are other possibilities; for example, moduli-sets of the form $\{2^n-1; 2^n; 2^n+1\}$ are among the most popular in use

- At least four considerations for the selection of moduli
  - The selected moduli must provide an **adequate range** whilst also ensuring that RNS representations are **unique**
  - The **efficiency of binary representations**; a balance between the different moduli in a given moduli-set is also important
  - The **implementations of arithmetic units** for RNS should to some extent be compatible with those for conventional arithmetic, especially given the legacy that exists for the latter
  - The **size of individual moduli**

Residue number systems

- One of the primary **advantages** of RNS is that certain RNS-arithmetic operations do not require carries between digits
- But, this is so only between *digits*
- Since a digit is ultimately represented in binary, there will be carries between bits, and therefore it is important to ensure that digits (→ the moduli) are **not too large**

Residue number systems

- Small digits make it possible to realize cost-effective table-lookup implementations of arithmetic operations
- But, on the other hand, if the moduli are small, then a large number of them may be required to ensure a sufficient dynamic range
- The choices depend on applications and technologies

Residue number systems

Negative numbers

- As with the conventional number systems, any one of the radix complement, diminished-radix complement, or sign-and-magnitude notations may be used in RNS
- The merits and drawbacks of choosing one over the other are similar to those for the conventional notations
- However, the **determination of sign** is much *more difficult* with the residue notations, as is **magnitude-comparison**
- This problem imposes many limitations on the application of RNS and we deal with just the positive numbers

Residue number systems

Basic arithmetic

- Addition/subtraction and multiplication are easily implemented with residue notation, depending on the choice of the moduli
- Division is much more difficult due to the difficulties of sign-determination and magnitude-comparison

Residue number systems

Basic arithmetic

- Residue **addition** is carried out by individually adding corresponding digits
- A **carry-out** from one digit position is **not propagated** into the next digit position
- As an example, with the moduli-set {2; 3; 5; 7}:
  - Operand 1 17 <1; 2; 2; 3>
  - Operand 2 19 <1; 1; 4; 5>
  - Result 36 <0; 0; 1; 1>

Residue number systems

Basic arithmetic

- **Subtraction** may be carried out by negating (in whatever is the chosen notation) the subtrahend and adding to the minuend
- This is straightforward for numbers in diminished-radix complement or radix complement notation
- For sign-and-magnitude representation, a slight modification of the algorithm for conventional sign-and-magnitude is necessary:
  - the sign digit is fanned out to all positions
  - addition proceeds as in the case for unsigned numbers but with a conventional sign-and-magnitude algorithm.

Residue number systems

Basic arithmetic

- **Multiplication** too can be performed simply by multiplying corresponding residue digit-pairs, relative to the modulus for their position → multiply digits and ignore or adjust an appropriate part of the result

- As an example, with the moduli-set {2; 3; 5; 7}:
  - Operand 1 11 → <1; 2; 1; 4>
  - Operand 2 13 → <1; 1; 3; 6>
  - Product 323 → <1; 2; 3; 3>

Residue number systems

Basic arithmetic

- Basic fixed-point division consists, essentially, of a sequence of subtractions, magnitude-comparisons, and selections of the quotient-digits
- But **comparison** in RNS is a difficult operation, because RNS is not positional or weighted
- Example:
  - moduli-set $\{2; 3; 5; 7\}$
  - the number represented by $<0; 0; 1; 1>$ is almost twice that represented by $<1; 1; 4; 5>$
  - but this is far from apparent

Residue number systems

Forward conversion

- The most direct way to convert from a conventional representation to a residue one is to divide by each of the given moduli and then collect the remainders
- This is a **costly** operation if the number is represented in an **arbitrary radix** and the **moduli are arbitrary**
- If number is represented in **radix-2** (or a radix that is a power of two) and the moduli are of a suitable form (e.g. 2^n-1), then these procedures that can be implemented with more efficiency

Residue number systems

Reverse conversion

- The conversion from residue notation to a conventional notation is more difficult (conceptually, if not necessarily in the implementation) and so far has been one of the major impediments to the adoption use of RNS
  - One way in which it can be done is to assign weights to the digits of a residue representation and then produce a positional (weighted) mixed-radix representation that can then be converted into any conventional form
  - Another approach involves the use of the Chinese Remainder Theorem, which is the basis for many algorithms for conversion from residue to conventional notation