ADVANCED ARCHITECTURES
INTENSIVE COMPUTATION

REPRESENTATIONS FOR FAST ARITHMETIC

Annalisa Massini

Lecture 5

2023-2024

REDUNDANT NUMBER SYSTEMS

Efficient number representations

- To **speed up the execution of arithmetic operations**, representations other than binary and 2's complement have been studied in recent decades
- The most important examples of such representations are:
  - **Redundant number systems**
  - **Residue number systems**

Efficient number representations

- When a representation other than binary or 2's complement is adopted, it is important to consider the impact that the change of representation has on:
  - ***Standard operations of ALU:***
    - Zero and overflow recognition
    - Sign detection
    - Arithmetic comparison
  - ***Conversions:***
    - Forward conversion from binary to the new representation
    - Reverse conversion from the new representation to binary

Efficient number representations

- Addition is the main arithmetic operation since it is the building block in implementing other arithmetic operations
- All other operations speed and cost depend on the addition

- Speed and cost of addition mainly depend on the **carry propagation**

- The questions are:
  - Is it possible to **represent numbers** in such a way as **to limit carry propagation** when performing additions?

  - Is it possible to **represent numbers** in such a way that **carry propagation does not occur** during addition?

REDUNDANT NUMBER SYSTEMS

Computer Arithmetic – Algorithms and Hardware Designs – B. Parhami – 2nd Ed

Ch. 3 Redundant Number Systems

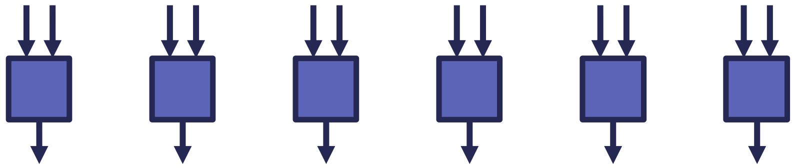
Computer Arithmetic Algorithms – I. Koren – 2nd Ed

Ch. 2 Unconventional Fixed-Radix Number Systems

Redundant number systems

- The most efficient way to execute an addition is **avoiding carry propagation** that is executing **carry-free addition**

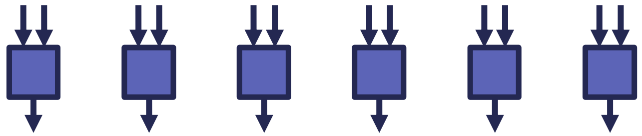
- Carry-free addition can be obtained by:
  - **widening of the digit set**
  - executing all digit additions **simultaneously**



Redundant number systems

- **Example - Let us consider radix r=10 and digit set [0, 9]**

$$
\begin{array}{cccccccl}
 & 5 & 7 & 8 & 2 & 4 & 9 & \text{operands radix-10, digit set [0,9]} \\
+ & 6 & 2 & 9 & 3 & 8 & 9 & \\
\hline
 & 11 & 9 & 17 & 5 & 12 & 18 & \text{result radix-10, digit set [0,18]}
\end{array}
$$



- If **for the result** we assume the **set [0, 18] as the set of digits**, the scheme works, but it is only valid for the first addition, not for the subsequent additions

Redundant number systems

- Consider now adding two numbers with **r=10** and **set [0, 18]**
- The sum of digits for each position is in **[0, 36]**
- It can be decomposed into an **interim sum** in **[0, 16]** and a **transfer digit** (carry) in **[0, 2]** → [0, 36] = 10 x [0, 2] + [0, 16]

		11	9	17	10	12	18	Operands digit in [0,18]
		6	12	9	10	8	18	
+		17	21	26	20	20	36	Result in 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]**

Redundant number systems

- Consider now adding two numbers with **r=10** and **set [0, 18]**
- The sum of digits for each position is in **[0, 36]**
- It can be decomposed into an **interim sum** in **[0, 16]** and a **transfer digit** (carry) in **[0, 2]** → [0, 36] = 10 x [0, 2] + [0, 16]

		11	9	17	10	12	18	Operands digit in [0,18]
		6	12	9	10	8	18	
+		17	21	26	20	20	36	Result in 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 in [0,18]**

Signed digit representation

- Hence:
  - We cannot do true *carry-free* addition
  - Carry propagates by **only one position** with r=10 and set [0, 18]

- Anyway, we refer to this scheme as ***carry-free*** addition
- Propagation of carries can be eliminated by a **lookahead scheme**:
  - Instead of first computing the transfer into position i based on the digits x_{i-1} and y_{i-1} and then combining it with the interim sum, we can determine s_i directly from x_i, y_i, x_{i-1}, and y_{i-1}

Signed digit representation

- The key to do carry-free addition is the **redundancy** introduced widening the digit set
- BUT, we do **not need this much redundancy** in decimal number system for carry-free addition: the **digit set [0, 11]** will work
- **Example**

		11	10	7	11	3	8	Operands digit in [0,11]
		7	2	9	10	9	8	
+		18	12	16	21	12	16	Result in digit set [0,22]
		↓	↓	↓	↓	↓	↓	
		8	2	6	1	2	6	Intermediate sums [0,9]
	↙	↙	↙	↙	↙	↙		
	1	1	1	2	1	1		Transfer digit set [0,2]
	1	9	3	8	2	3	6	**Sum in [0,11]**

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

  radix-2 → 0, 1

- 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 mantaining the conventional number interpretation
- **Redundancy** may imply representation of numbers is ***not unique***

Signed-digit numbers

- Digit sets of the form [−α, β] were studied for redundant number representations
- This class is called **generalized signed-digit (GSD) representation**

- A **radix-r redundant signed-digit** number system is based on digit set

$$S = \{-\alpha, -(\alpha - 1), \ldots, -1, 0, 1, \ldots, \beta\}$$

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

- The digit set S contains $\alpha + \beta + 1$ values
- Hence multiple representations for any number in signed digit format → redundancy

Signed-digit numbers

- Main characteristics:
  - All digits have **weights r^p** (p-position, r-radix)
  - Digits have **signed values**
  - Any set digit [−α, β] including 0, can be used, and also [−α, α]
  - If $\boldsymbol{\alpha+\beta+1 > r}$ the number system is **redundant**
- **Examples**

[-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	

- An important parameter of a GSD number system (but can be applied to any digit set) is its ***redundancy index***: $\boldsymbol{\rho = \alpha + \beta + 1 - r}$

Signed-digit numbers

- **Example**
  - radix-10 digit set $[\bar{9}, 9]$
  - If n=2 the range is $\bar{9}\bar{9} \leq x \leq 99$ which includes 199 numbers
  - With two digits (x_1, x_0), each having 19 possibile values, there are 19^2=361 representations
  - Hence some numbers have more than one representation and the number system is **redundant**
  - **For example** $(01) = (1\bar{9}) = 1$ **and** $(0\bar{2}) = (\bar{1}8) = -2$
  - However the representation of **0 is unique and so is that of 10**
  - 361-199=162 redundant representations → 81% **redundancy** (162/199)
  - ***redundancy index***: $\rho = \alpha + \beta + 1 - r = 9 + 9 + 1 - 10 = 9$

Signed-digit numbers

- The amount of redundancy can be reduces by restricting the digit set to the symmetric set $[\bar{a}, a]$ with $\left\lceil \frac{r-1}{2} \right\rceil \leq a \leq r-1$
- **Example**
  - For r=10 the range for a is $5 \leq a \leq 9$
  - If $a = 6$ for n=2 there are 133 numbers in the range $\bar{6}\bar{6} \leq x \leq 66$
  - Each bit has 13 possible values – there are 13^2=169 representations
  - Now **1** has **only one representation** (01) because $(1\bar{9})$ **is not valid**,
  - 4 has two representations (04) **and** $(1\bar{6})$

  - **Redundancy** is 27% (169-133/133) → 36 redundant representations
  - ***redundancy index***: ***ρ*** **= 6 + 6 + 1 – 10 = 3**

Signed-digit numbers

- The original motivation to introduce SD numbers is to eliminate carry propagation chains in addition/subtraction so that execution time is independent of length of operands
- Anyway, SD numbers are useful for multiplication and division
- **Addition algorithm**
  - Consider $(x_{n-1}, \ldots, x_0) \pm (y_{n-1}, \ldots, y_0) = (s_{n-1}, \ldots, s_0)$
  - Breaking the carry chains requires an algorithm in which sum digit s_i depends only on the four operand digits x_i, y_i, x_{i-1}, and y_{i-1}
  - **Step 1**: Compute carry digit c_i and interim sum u_i

$$c_i = \begin{cases} 1 & \text{if } (x_i + y_i) \geq a \\ \bar{1} & \text{if } (x_i + y_i) \leq \bar{a} \\ 0 & \text{if } (x_i + y_i) < a \end{cases} \qquad u_i = x_i + y_i - rc_i$$

  - **Step 2**: Calculate the final sum : $s_\mathrm{i} = u_i + c_{i-1}$

Signed-digit numbers

Example

- r=10 $a = 6$ $x_i \in \{-6, \cdots, 0,1, \cdots, 6\}$
- $c_i = \begin{cases} 1 & \text{if } (x_i + y_i) \geq 6 \\ \bar{1} & \text{if } (x_i + y_i) \leq \bar{6} \\ 0 & \text{if } (x_i + y_i) < 6 \end{cases}$ and $u_i = x_i + y_i - 10c_i$
- Conventional addition becomes

$$\begin{array}{rl} 3\,6\,4\,5 & x \\ +\,1\,4\,5\,6 & y \\ \hline 5\,1\,0\,1 & s \end{array} \qquad \begin{array}{rl} 3\,6\,4\,5 & x \\ +\,1\,4\,5\,6 & y \\ \hline 0\,1\,1\,1\;\; & c \\ 4\,0\,\bar{1}\,1 & u \\ \hline 5\,1\,0\,1 & s \end{array}$$

- Carry bits shifted to left to simplify execution of second step

Signed-digit numbers

- This addition algorithm can be used for **conversion**
  - Conversion from **decimal number to SD**
    - Consider each digit as the sum x_i+y_i and $\{-6, \cdots, 0, 1, \cdots, 6\}$
    - **Example** - converting decimal 6849 to SD

$$
\begin{array}{rl}
6849 & x_i + y_i \\
\hline
1101 & c_i \ \text{ computed using } x_i + y_i \\
\bar{4}\bar{2}4\bar{1} & u_i \ \text{ computed as } x_i + y_i - 10c_i \\
\hline
1\bar{3}\bar{2}5\bar{1} & s_i \ \text{ computed as } u_i + c_{i-1}
\end{array}
$$

  - Conversion from **SD to decimal**
    - Subtract digits with negative weight from positive weight digits
    - **Example** - converting $1\bar{3}\bar{2}5\bar{1}$ to decimal

$$
\begin{array}{r}
10050 \\
-\ 03201 \\
\hline
6849
\end{array}
$$

Signed-digit numbers

Choice of digit set to guarantee no new carry

- Sum digit $s_i = u_i + c_{i-1}$ must satisfy $|s_i| \leq a$
- Since $|c_{i-1}| \leq 1$, the condition $|u_i| \leq a - 1$ must hold for all x_i and y_i
- Largest $x_i + y_i$ is $2a$ for which $c_i = 1$ and $u_i = 2a - \mathrm{r}$
- Since $|u_i| \leq a - 1$ then $2a - r \leq a - 1$ holds and $a \leq r - 1$
- Smallest $x_i + y_i$ for which $c_i = 1$ is a and so $u_i = a - \mathrm{r} < 0$ that implies $|u_i| = r - a$
- Substituting $|u_i| = r - a$ into $|u_i| \leq a - 1$ we get $2a \geq r + 1$
- So the digit set must satisfy $\left\lceil \frac{r+1}{2} \right\rceil \leq a \leq r - 1$
- For example, to guarantee no new carries in previous algorithm, SD decimal numbers must satisfy $a \geq 6$

Binary Signed-digit numbers

Addition algorithm for the case r=2

- For r=2 we have only $a = 1$ and only one possible digit set $x_i \in \{-1,0,1\} = \{\bar{1},0,1\}$
- Interim sum and carry in addition algorithm are:

$$c_i = \begin{cases} 1 & \text{if } (x_i + y_i) \geq 1 \\ \bar{1} & \text{if } (x_i + y_i) \leq \bar{1} \\ 0 & \text{if } (x_i + y_i) = 0. \end{cases} \qquad u_i = (x_i + y_i) - 2c_i$$

- **Summary of rules**

$x_i y_i$	00	01	$0\bar{1}$	11	$\bar{1}\bar{1}$	$1\bar{1}$
c_i	0	1	$\bar{1}$	1	$\bar{1}$	0
u_i	0	$\bar{1}$	1	0	0	0

- Note that $10, \bar{1}0, \bar{1}1$ *are not included* since addition is commutative
- Since in the binary case $a \geq \left\lceil \frac{r+1}{2} \right\rceil = 2$ cannot be satisfied, there is no guarantee a new carry will not be generated in step 2

Binary Signed-digit numbers

If operands do not have $\bar{1}$ → new carries are **not generated**

- **Example**
  - In conventional representation a carry propagates from least to most significant position
  - Here no carry propagation chain exists

$$
\begin{array}{cccccccc}
 & & 1 & 1 & \cdots & 1 & 1 & \\
+ & & 0 & 0 & \cdots & 0 & 1 & \\
\hline
 & 1 & 1 & 1 & \cdots & 1 & & c_i \\
 & & \bar{1} & \bar{1} & \cdots & \bar{1} & 0 & u_i \\
\hline
 & 1 & 0 & 0 & \cdots & 0 & 0 & s_i
\end{array}
$$

If operands have $\bar{1}$ → new carries may be **generated**

- **Example**
  - If $x_{i-1}y_{i-1}= 01$ and $c_i=1$ and if $x_iy_i= 01$ and $u_i=1$ **then** $s_i = u_i + c_i = 1+1$ and a **new carry** is generated
  - Stars indicate positions where new carries are generated and must be allowed to propagate

$$
\begin{array}{ccccccccc}
 & & 0 & \bar{1} & 1 & \bar{1} & 1 & 1 & \mathbf{-9} \\
+ & & 1 & 0 & 0 & \bar{1} & 0 & 1 & \mathbf{29} \\
\hline
 & 1 & \bar{1} & 1 & \bar{1} & 1 & 1 & & c_i \\
 & & \bar{1} & 1 & \bar{1} & 0 & \bar{1} & 0 & u_i \\
\hline
 & & * & * & * & 1 & 0 & 0 & s_i
\end{array}
$$

Binary Signed-digit numbers

x_iy_i	00	01	$0\bar{1}$	11	$\bar{1}\bar{1}$	$1\bar{1}$
c_i	0	1	$\bar{1}$	1	$\bar{1}$	0
u_i	0	$\bar{1}$	1	0	0	0

- Combination $c_{i-1} = u_i = 1$ occurs when $x_iy_i = 0\bar{1}$ and $x_{i-1}y_{i-1}$ is 11/01
- To avoid setting $u_i = 1$ we can set $c_i = 0$ and $u_i = \bar{1}$
- Similarly, $c_{i-1} = u_i = \bar{1}$ when $x_iy_i = 01$ and $x_{i-1}y_{i-1}$ is $\bar{1}\bar{1}/0\bar{1}$ and to avoid setting $u_i = \bar{1}$ we can set $c_i = 0$ and $u_i = 1$
- **No new carries** are generated if c_i and u_i are determined by examining the two bits $x_{i-1}y_{i-1}$
- c_i and u_i can still be calculated **in parallel** for all bit positions

x_iy_i	00	01	01	$0\bar{1}$	$0\bar{1}$	11	$\bar{1}\bar{1}$
$x_{i-1}\,y_{i-1}$	–	neither is $\bar{1}$	at least one is $\bar{1}$	neither is $\bar{1}$	at least one is $\bar{1}$	–	–
c_i	0	1	0	0	$\bar{1}$	1	$\bar{1}$
u_i	0	$\bar{1}$	1	$\bar{1}$	1	0	0

Binary Signed-digit numbers

$x_i y_i$	00	01	01	$0\bar{1}$	$0\bar{1}$	11	$\bar{1}\bar{1}$
$x_{i-1}\, y_{i-1}$	–	neither is $\bar{1}$	at least one is $\bar{1}$	neither is $\bar{1}$	at least one is $\bar{1}$	–	–
c_i u_i	0 0	1 $\bar{1}$	0 1	0 $\bar{1}$	$\bar{1}$ 1	1 0	$\bar{1}$ 0

- Repeating the example before with the new table we obtain

$$
\begin{array}{rcccccccl}
 & & 0 & \bar{1} & 1 & \bar{1} & 1 & 1 & \mathbf{-9} \\
+ & & 1 & 0 & 0 & \bar{1} & 0 & 1 & \mathbf{29} \\
\hline
0 & 0 & 0 & \bar{1} & 1 & 1 & & & \\
 & & 1 & \bar{1} & 1 & 0 & \bar{1} & 0 & \\
\hline
 & & 1 & \bar{1} & 0 & 1 & 0 & 0 &
\end{array}
$$

- Note that pair $1\,\bar{1}$ is equivalent to pair 01
- Note also that direct summation of the two operands results in $1\bar{1}1\bar{1}00$ that is equivalent to 010100, all representing **20**

Binary Signed-digit numbers

Multiple number representations

- **Minimal SD representations** include the minimal number of nonzero digits and is important for fast multiplication and division algorithms
  - Nonzero digits → add/subtract operations
  - Zero digits → shift-only operations

Example

- Among the representations of **7**, $100\bar{1}$ is the minimal representation

- The canonical Booth recoding algorithm generates minimal SD representations of given binary numbers

	8	4	2	1
	0	1	1	1
	1	$\bar{1}$	1	1
	1	0	$\bar{1}$	1
	1	0	0	$\bar{1}$
1	$\bar{1}$	$\bar{1}$	1	1
		$\vdots$		

Binary Signed-digit numbers

Encoding

- Any hardware implementation of GSD arithmetic requires the choice of a **binary encoding scheme** for the digit values
- In the case of binary SD numbers there are 4x3x2=24 ways to encode the three values 0, 1 and -1 using 2 bits, x^h and x^l (high and low)
- Only nine are distinct encodings under permutation and logical negation, but **only two** have been used in practice:

x	Encoding 1 $x^h\ x^l$	Encoding 2 $x^h\ x^l$
0	0 0	0 0
1	0 1	0 1
$\bar{1}$	1 0	1 1

Binary Signed-digit numbers

- **Encoding 2**: the two bits follow the 2's complement representation
- **Encoding 1**: the two bits are associated to the same power of 2
  - is sometimes preferable
  - Satisfies $x=x^l-x^h$ and **11** is a valid value of 0
  - Simplifies conversion from SD to 2's complement subtracting the sequence of high bits from the sequence of low bit using 2's complement arithmetic
  - This requires a complete binary adder
  - (A simpler conversion algorithm exists)

x	Encoding 1 x^h x^l	Encoding 2 x^h x^l
0	0 0	0 0
1	0 1	0 1
$\bar{1}$	1 0	1 1

Signed digit representation

Conversions

- Since input numbers provided from the outside (machine or human interface) are in standard binary or decimal and outputs must be presented in the same way, **conversions** between binary or decimal and GSD representations are required

- The **conversion** from redundant representation essentially involves carry propagation and is thus **rather slow**
- Conversion is done generally at the input and output, i.e. not very often
- Thus, if long sequences of computation are performed between input and output, the conversion overhead can become negligible

Signed digit representation

- **Storage overhead** (number of bits used to represent a GSD digit compared to a standard digit in the same radix) can appear as a disadvantage of redundant representations
- However, with advances in VLSI technology, this is no a major drawback

- Properties of GSD representations are important for the implementation of arithmetic support functions, that are:
  - **zero detection**
  - **sign test**
  - **overflow handling**

Signed digit representation

Zero detection

- In a GSD number system, the integer 0 may have multiple representations
- **Example** in radix 4 and set [-1, 5], the three-digit numbers

  0 0 0 and −1 4 0 both represent 0
- Note that in the special case of $\alpha < r$ and $\beta < r$, zero is uniquely represented by the all-0s vector

- So, despite redundancy and multiple representations, **comparison** of numbers for equality can be simple in this common special case, since it involves subtraction and detecting the all-0s pattern

Signed digit representation

Sign test

- Sign test is more **difficult** and so any *relational comparison* such as <, ≤, etc.
- The **sign** of a GSD number in general **depends on all its digits**
- Thus, sign test is **slow** if done through signal propagation (ripple design) or **expensive** if done by a fast lookahead circuit (that is in contrast for it is trivial sign test for 2's-complement)
- In the special case of $\alpha < r$ and $\beta < r$, the sign of a number is identical to the sign of its most significant nonzero digit
- But even in this special case, determination of sign requires scanning of all digits, a process that can be as slow as worst-case carry propagation

Signed digit representation

Overflow handling

- Overflow handling is also more difficult in GSD arithmetic
- Consider the addition of two k-digit numbers
- Such an addition produces a transfer-out digit t_k
- Since t_k is produced using the worst-case assumption about yet unknown t_{k-1} , we can get an overflow indication (t_k != 0) even when the result can be represented with k digits
- It is possible to perform a test to see whether the overflow is real and, if it is not, to obtain a k-digit representation for the true result
- However, this test and conversion are fairly slow

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

- Addition can be executed applying 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

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

		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:

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

- The RB number obtained in this way is in **canonical form**
- This encoding 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