Redundant Representations
Ivor Page[1]

2.1 The Binary Signed Digit Representation (BSD)

Conventional unsigned binary is a positional number system with radix two and digit set $\{0, 1\}$. In any redundant system the digit set has larger cardinality (more values) than the radix. In the BSD system, the digit set is $[-1, 1] = \{-1, 0, 1\}$, usually written $\{\overline{1}, 0, 1\}$. Each number in a BSD system is represented by two binary vectors, where each digit of the number A is represented by its positive and negative components,

$$\begin{aligned} A^+ &= a^+_{k-1}, a^+_{k-2}, \cdots, a^+_1, a^+_0 \\ A^- &= a^-_{k-1}, a^-_{k-2}, \cdots, a^-_1, a^-_0 \end{aligned}$$

The following table shows the most common assignment of values to these bits:

Value	a^+ a^-
1	1 0
0	0 0
$\overline{1}$	0 1

It is easy to see that the system is redundant. For example, the decimal value +5 can be represented by 000101, 00011$\overline{1}$, 0010$\overline{1}\overline{1}$, 001$\overline{1}$01, 001$\overline{1}$1$\overline{1}$, etc.

The range of a k digit BSD system is symmetrical: $[-2^k + 1, 2^k - 1]$.

Conversion from 2's complement to BSD is trivial. Recall the value of a 2's complement number:

Value of $A = -a_{k-1}2^{k-1} + \sum_{i=0}^{k-2} a_i 2^i$

The equivalent BSD number X has:

$$\begin{aligned} x^-_{k-1} &= a_{k-1} \text{ sign digit} \\ x^+_{k+1} &= 0 \text{ sign digit} \\ \forall_{0 \le i < k-1}\ x^+_i &= a_i, \ \ x^-_i = 0 \end{aligned}$$

The sign digit of the 2's complement number is copied into the negative part of the corresponding position in the BSD number and the remaining bits of the 2's complement number

[1]University of Texas at Dallas

are copied into the corresponding positive components of the BSD number. Figure 1 illustrates the process.

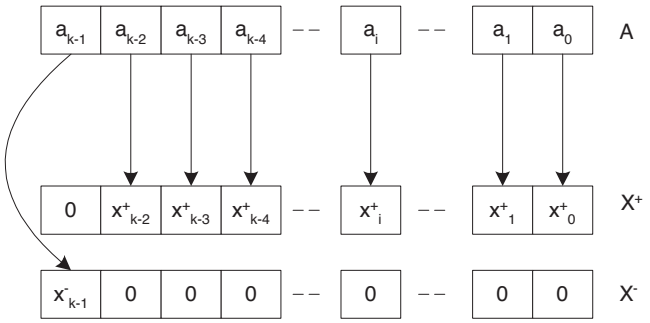

Figure 1: Conversion of 2's complement value A to BSD value X

Conversion of the BSD number X to 2's complement binary requires the subtraction of the bit vectors: $X^+ - X^-$, using a standard 2's complement adder. This is usually performed by a carry-look-ahead adder having $O(logk)$ processing time.

2.2 Carry Ripple Elimination

A BSD system limits the carry propagation path length to only 2 bits, thereby enabling constant time addition, independent of the number of bits being added.

The adder for two BSD numbers, X and Y, comprises two stages. In the first (upper) stage of digit position i, input bits x_i and y_i are combined to form an intermediate sum s_i and an intermediate carry c_{i+1}. These are BSD values. In the second (lower) stage, the sum bit s_i is combined with the carry-in from the previous stage, c_i, to generate the result bit z_i. Figure 4 shows the block diagram of the adder.

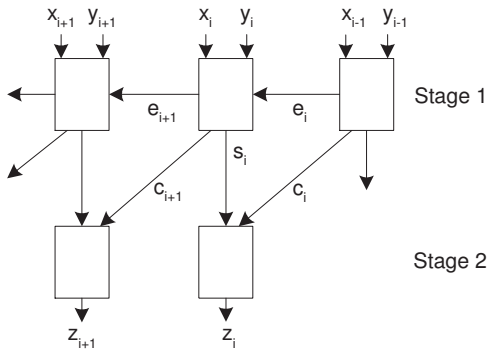

Figure 2: Two stage BSD Adder

Redundancy enables us to choose the values of the intermediate sum and carry so that the second stage can *absorb* the carry from the position to the right. Below is a table of possible intermediate values for the 9 values of x_i and y_i.

Inputs		Intermediate Results for $c_i \in \{0, 1\}$		Intermediate Results for $c_i \in \{0, \overline{1}\}$	
x_i	y_i	c_{i+1}	s_i	c_{i+1}	s_i
1	1	1	0	1	0
1	0	1	$\overline{1}$	0	1
1	$\overline{1}$	0	0	0	0
0	1	1	$\overline{1}$	0	1
0	0	0	0	0	0
0	$\overline{1}$	0	$\overline{1}$	$\overline{1}$	1
$\overline{1}$	1	0	0	0	0
$\overline{1}$	0	0	$\overline{1}$	$\overline{1}$	1
$\overline{1}$	$\overline{1}$	$\overline{1}$	0	$\overline{1}$	0

In the cases $x_i y_i = 00, \overline{1}1, 1\overline{1}$ the intermediate sum and carry must both be zero because there is only one way to represent zero. In the case $x_i y_1 == 11$ the intermediate sum and carry must be 0 and 1 respectively because there is only one way to represent two in 2 bits. Similarly when $x_i y_i = \overline{11}$ the intermediate sum and carry must be 0 and $\overline{1}$.

In the remaining four cases the redundancy gives us a choice of intermediate results: $x_i y_i = 10, 01, 0\overline{1}, \overline{1}0$. The sum $x_i + y_i = 1, 1, \overline{1}, \overline{1}$ respectively. We choose the intermediate carry and sum values, c_{i+1} and s_i, depending on the expected carry in, c_i.

Consider $c_i \in \{0, 1\}$. This corresponds to the left column of the table above. In all nine cases $s_i \in \{0, \overline{1}\}$. If a carry-in of 1 occurs, the intermediate results in the first column enable the carry-in to be absorbed without further carry propagation.

Similarly, if $c_i \in \{0, \overline{1}\}$, the intermediate result, $s_i \in \{0, 1\}$, enables the carry to be absorbed without further carry propagation.

In general, we cannot know the exact value of the carry input to any stage until it is generated, but we can bound its value into one of the two subsets, $\{\overline{1}, 0\}$ or $\{0, 1\}$, simply by examining the values of the inputs to the column to the immediate right:

Inputs		Carry Out Subset
x_{i-1}	y_{i-1}	c_i
1	1	$\{0, 1\}$
1	0	$\{0, 1\}$
1	$\overline{1}$	0
0	1	$\{0, 1\}$
0	0	0
0	$\overline{1}$	$\{0, \overline{1}\}$
$\overline{1}$	1	0
$\overline{1}$	0	$\{0, \overline{1}\}$
$\overline{1}$	$\overline{1}$	$\{0, \overline{1}\}$

In three cases the carry out of any digit position is in $\{0, \overline{1}\}$, in three others it is in $\{0, 1\}$. In the remaining three, the carry must be zero. The logic signal e_i passes between the first (upper level) stages of neighboring pairs of digit positions in the BSD adder to inform position i of the carry-in subset. This signal does not propagate along the adder.

The longest paths through the adder are shown in bold in Figure 3.

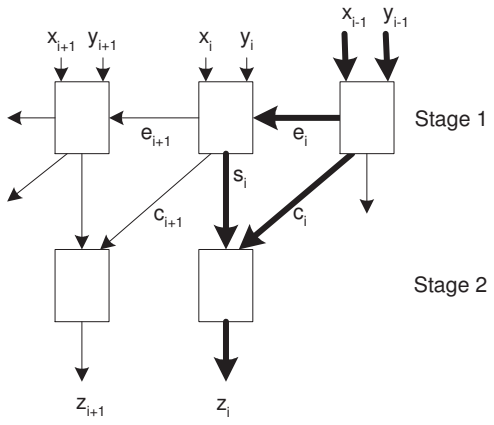

Figure 3: Longest Paths in BSD Adder

The three cases where the carry is zero can be assigned to either subset in the form of don't care input vectors to the logic circuit.

Example:

0	$\overline{1}$	$\overline{1}$	1	0	$\overline{1}$	0	1	$\overline{1}$	1	$X = -333$
0	0	$\overline{1}$	1	1	$\overline{1}$	0	1	0	1	$Y = -43$
1	1	0	0	1	0	0	1	0	0	e
0	1	0	0	1	0	0	0	$\overline{1}$	0	s
$\overline{1}$	$\overline{1}$	1	0	$\overline{1}$	0	1	0	1	0	c
$\overline{1}$	0	1	0	0	0	1	0	0	0	$Z = -376$

The e and c signals have been left-shifted so each column of e, s and c values, are the inputs to a digit position. The redundancy is used in the 2^1, 2^5, and 2^8 positions, where the e signals are used to select the appropriate intermediate sum and carry signals for those stages. In each case the carry is absorbed in the second (lower) stage.

It is instructive to carry out such an example on paper by dealing with the digit positions in random order. Doing so illustrates the fact that the addition process takes constant time.

Since BSD representation requires double the number of bits used in the 2's complement scheme, it is impractical to consider its use for storing all values within a computer. Values must therefore be converted to BSD before calculations are performed, and results must be converted back to 2's complement. Since conversion from BSD to 2's complement requires a log-time addition, nothing is gained by using BSD if only pairs of values are added or subtracted.

The real advantage of BSD, and other competing redundant number systems, comes when many arithmetic operations must be performed within the system before results must be converted back to 2's complement. One such case is the multiplication of integers. If two k bit values are to be multiplied, the two arguments can be converted in constant time (actually only one of them needs to be converted) and the BSD scheme can be used to sum the k partial products.

If we have $k/2$ BSD adders, then pairs of partial products can be added, giving $k/2$ intermediate results. These can then be added in pairs, and so on. The result of such a scheme is obtained in $O(log\ k)$ time. Conversion back to 2's complement also takes $O(log\ k)$ time, so multiply is achieved in log time.

Further details of this multiplier will be considered when we get to the section on multiply. By using techniques invented by Booth the number of partial products can be reduced by a factor of 2, 4 or 8 without unduly complicating the logic of the multiplier.

2.3 General Redundant Systems

2.3.1 Digit Sets

We have seen the BSD system which uses the digit set $\{\overline{1}, 0, 1\}$, also written $[-1, 1]$. Many other digit sets are possible, including both non-redundant and redundant number systems. Avizienis defined a class of number systems, known as Ordinary Sign Digit (OSD) systems, which have symmetrical digit sets, $[-\alpha, \alpha]$. and radix $r > 2$, where $\lceil r/2 \rceil + 1 < \alpha < r - 1$. These number systems allow at least $\lceil r/2 \rceil + 3$ different digit values and are therefore redundant (non-redundant systems have exactly r digit values).

More general redundant systems have been studied more recently with digit sets, $[-\alpha, \beta]$. These are known as Generalized Sign-Digit (GSD) representations. Examples are the radix 2 BSD system with digit set $[-1, 1]$, the high redundancy radix 2 system with digit set $[-7, 7]$, and the radix 2 system with digit set $[0, 2]$. None of these is an OSD system.

The *Redundancy Index* is defined as $\rho = \alpha + \beta + 1 - r$, which gives the amount by which the cardinality of the digit set exceeds the radix.

Carry-free addition is possible if and only if one of the following conditions is met:

1. $(r > 2) \wedge (\rho \geq 3)$
2. $(r > 2) \wedge (\rho = 2) \wedge (\alpha \neq 1) \wedge (\beta \neq 1)$

2.3.2 Carry-Free Addition Algorithms

Unlike the BSD system studied earlier, which only has a *Limited Carry-Free Addition Algorithm*, we will first study a generalized system that is not limited. In an unlimited system, the redundancy in the number system enables carry-free addition without each stage requiring any knowledge of the input carry. Recall that in the BSD system, in each digit position, the first stage of the adder received a logic signal from the cell to its right specifying whether the carry from that stage would be in $\{\overline{1}, 0\}$ or $\{0, 1\}$.

For unlimited carry-free addition, certain constraints must be observed. The addition process is as follows:

- In the first stage, compute the positional sums, $p_i = x_i + y_i$.
- Separate p_i into the transfer digit t_{i+1} and the interim sum $w_i = p_i - rt_{i+1}$.
- In the second stage, add w_i to t_i to produce the sum digit s_i.

Clearly $-2\alpha \leq p_i \leq 2\beta$ since x_i, y_i have digit set $[-\alpha, \beta]$.

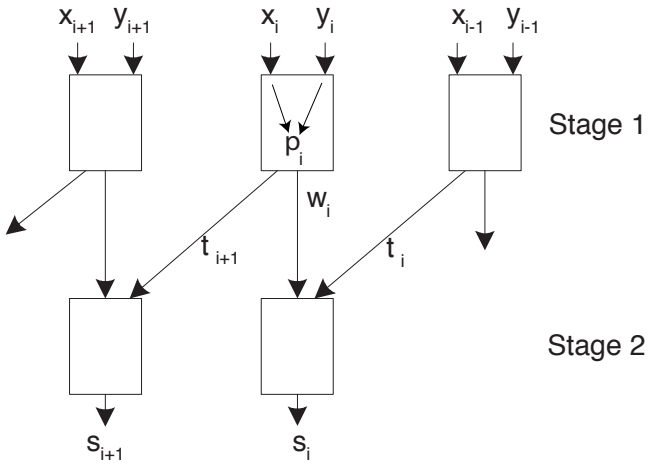

Figure 4: Two stage GSD Adder

The next step is to decide on the minimum range of the digit set for the transfer digit, t_i. Minimizing this range will minimize the number of logic signals needed for t_i.

Assume that the transfer signal t_i has the digit set $[-\lambda, \mu]$. If a carry-in of $t_i = -\lambda$ occurs, the result after the second stage is $s_i = w_i - \lambda$. The result must be in the range $[-\alpha, \beta]$, meaning $-\alpha \leq w_i - \lambda$, or, $-\alpha + \lambda \leq w_i$. This condition enables the most negative possible value of t_i to be absorbed in the second stage.

Similar reasoning leads to the result that, for a carry-in of β to be absorbed, $w_i \leq \beta - \mu$. Combining these, we get:

$$-\alpha + \lambda \leq p_i - rt_{t+1} \leq \beta - \mu$$

To find the smallest value of λ allowed, we take the first inequality, $-\alpha + \lambda \leq p_i - rt_{t+1}$, and substitute the most negative values for the transfer digit, $t_{i+1} = -\lambda$, and for the interim sum digit, $p_i = -2\alpha$:

$$\begin{aligned} -\alpha + \lambda &\leq -2\alpha + r\lambda \\ \alpha &\leq \lambda(r-1) \\ \lambda &\geq \frac{\alpha}{r-1} \end{aligned}$$

By similar reasoning,

$$\mu \geq \frac{\beta}{r-1}$$

The next step is to choose the transfer digit value t_{i+1} by comparing the position sum p_i with $\lambda + \mu + 1$ constants. Call these constants *break points*, C_j for $-\lambda \leq j \leq \mu + 1$. The transfer digit is chosen to be j if and only if $C_j \leq p_i < C_{j+1}$. For $t_i \in [-\lambda, \mu]$ the break points will be $C_{-\lambda} \cdots C_{\mu}$ We will illustrate how these constants are chosen by an example.

Example:

For $r = 10$ and digit set $[-5, 9]$, we need $\lambda \geq 5/9$ and $\mu \leq 1$. We choose the minimal values, $\lambda_{min} = \mu_{min} = 1$ so that the transfer signal, $t_i \in [-1, 1]$ can be implemented using only two logic signals.

Note that $p_i \in [-10, 18]$. For each value in this range, we must select the interim sum and carry signals, w_i and t_{i+1} such that $s_i = w_i + t_i \in [-5, 9]$ for any $t_i \in [-1, 1]$.

Here is a table enumerating all the possible values of p_i, $p_i + t_i$, and the values of possible outputs, w_i and t_{i+1}.

		Option 1			Option 2			p_i
p_i	$p_i + t_i$	$w_i + t_i$	w_i	t_{i+1}	$w_i + t_i$	w_i	t_{i+1}	$abcdef$
-10	[-11, -9]	[-1, 1]	0	-1				10110
- 9	[-10, -8]	[0,2]	1	-1				10111
-8	[-9, -7]	[1, 3]	2	-1				11000
-7	[-8, -6]	[2, 4]	3	-1				11001
-6	[-7, -5]	[3, 5]	4	-1				11010
-5	[-6, -4]	[4, 6]	5	-1				11011
-4	[-5, -3]	[5, 7]	6	-1	[-5, -3]	-4	0	11100
-3	[-4, -2]	[6, 8]	7	-1	[-4, -2]	-3	0	11101
-2	[-3, -1]	[7, 9]	8	-1	[-3, -1]	-2	0	11110
-1	[-2, 0]				[-2, 0]	-1	0	11111
0	[-1, 1]				[-1, 1]	0	0	00000
+1	[0, 2]				[0, 2]	1	0	00001
+2	[1, 3]				[1, 3]	2	0	00010
+3	[2, 4]				[2, 4]	3	0	00011
+4	[3, 5]				[3, 5]	4	0	00100
+5	[4, 6]				[4, 6]	5	0	00101
+6	[5, 7]	[-5, -3]	-4	+1	[5, 7]	6	0	00110
+7	[6, 8]	[-4, -2]	-3	+1	[6, 8]	7	0	00111
+8	[7, 9]	[-3, -1]	-2	+1	[7, 9]	8	0	01000
+9	[8, 10]	[-2, 0]	-1	+1				01001
+10	[9, 11]	[-1, 1]	0	+1				01010
+11	[10, 12]	[0, 2]	1	+1				01011
+12	[11, 13]	[1, 3]	2	+1				01100
+13	[12, 14]	[2, 4]	3	+1				01101
+14	[13, 15]	[3, 5]	4	+1				01110
+15	[14, 16]	[4, 6]	5	+1				01111
+16	[15, 17]	[5, 7]	6	+1				10000
+17	[16, 18]	[6, 8]	7	+1				10001
+18	[17, 10]	[7, 9]	8	+1				10010

Consider the following tables where these quantities are tabulated.

For $p_i \in [-10, -5]$ $t_{i+1} = -1$, in order to keep the output sum value in range.

For $p_i \in [-4, -2]$, $t_{i+1} \in [-1, 0]$ is possible, but for $p_i \in [-1, 5]$, t_{i+1} must be 0.

For $p_i \in [6, 8]$, $t_{i+1} \in [0, 1]$ is possible, but for $p_i \in [9, 18]$, t_{i+1} must be +1.

Here are all these restrictions on the value of the carry-out, t_{i+1}, for each range of values of p_i.

$t_i \in [-1, 1]$					
p_i	[-10,-5]	[-4,-2]	[-1,5]	[6,8]	[9,18]
t_{i+1}	-1	{-1,0}	0	{0,1}	1

The redundancy gives us choices for the value of t_{i+1} for $p_i \in [-4, -2]$ and $p_i \in [6, 8]$. This enables us to choose the break points in between the ranges to minimize the number of bits needed to store them and to also simplify the logic in comparing p_i values against these break points. $C_0 = -4$ and $C_1 = 8$ would be suitable choices. The table becomes:

$t_i \in [-1, 1]$			
p_i	[-10,-5]	[-4,8]	[9,18]
t_{i+1}	-1	0	1

With these break points, the logic for choosing the carry-out is simple:

If p_i is a 6 bit 2's complement value with bit vector $< abcdef >$,

$$\begin{aligned} t_{i+1} = -1 \quad & \text{if} \quad a(\overline{c} + \overline{d}) \\ t_{i+1} = 1 \quad & \text{if} \quad \overline{a}(b + c) \\ t_{i+1} = 0 \quad & \text{otherwise} \end{aligned}$$

The result is an addition algorithm that is carry-free without the need for estimation of the carry-in from the previous stage.

Here is an example of the adder in operation:

A		-4	0	6	8	-3	9	$= -393,221_{10}$
B		2	-5	4	9	1	7	$= 154,917_{10}$
p_i		-2	-5	4	9	1	7	
w_i		-2	0	0	7	-2	6	
t_i	0	-1	1	1	0	1	0	
s_i	0	-3	1	1	7	-1	6	$= -288,304_{10}$

2.4 Example

1. Determine the minimum numeric range of the interstange transmission signal t_i in a radix 16 General Sign Digit redundant system with digit set $[-5, 15]$. Compute suitable break-points for the output and transmission signal.

   Answer:

   The range is $[-\lambda, \mu]$ where:

$$\lambda \geq \frac{\alpha}{r-1} = \frac{5}{15}$$

$$\mu \geq \frac{\beta}{r-1} = \frac{15}{15}$$

   $t_i \in [-1, 1]$. The ranges of t_i values as a function of p_i are:

p_i	[-10,-2]	[-4,14]	[12,30]
t_i	-1	0	+1

   Suitable breakpoints: $p_i \in [-10, 30]$. That's a 6 bit value. Consider values of p_i near the breakpoints:

Decimal	binary ABCDEF
-10	110110
..	.....
-5	111011
-4	111100
-3	111101
-2	111110
..	.....
12	001100
13	001101
14	001110
..	.....
30	011110

   Values of $p_i < -10$ cannot occur and can therefore be used as don't cares in the equations for t_i.

   Let's code t_i as follows:

	Logic Signal Encoding	
t_i	t_{i1}	t_{i0}
-1	1	0
0	0	0
+1	0	1

Where t_i is the two bit value $t_{i1}t_{i0}$.

For t_{i1}, the A,B signals are 11 throughout the range [-10,-2]. A k-map base on the signals CDEF yields the eqn $t_{i1} = AB(\overline{C} + \overline{D})$ with a break point between -5 and -4. Then $t_i = -1$ for the range $p_i \in [-10, -5]$

For t_{i0}, the A signal is 0 throughout the range [0,30]. To simplify the calculation, ignore the F signal and draw a k-map based on BCDE only. The k-map yields $t_{i0} = B + CD$ and puts the break point between 11 and 12. Then $t_i = +1$ for the range $p_i \in [12, 30]$.

2. Compute the range of a k bit system using the above digit set.

   Answer:

   Each digit is in the range [-5,15] and the radix is 16, so the most positive k bit value has a 15 in each digit. This is the same as hex. Its value is $16^k - 1$. The most negative value is $-5(16^{k-1} + 16^{k-2} + \cdots + 16^1 + 16^0) = -5(16^k - 1)/15 = -(16^k - 1)/3$.

3. Compute the efficiency of the above system, where efficiency is the size of the range supported by the system divided by the size of the range of a binary system using the same number of bits. For example, a k bit BSD system has range $-(2^k - 1)$ to $+(2^k - 1)$ and uses $2k$ bits. The size of the range is $(2^k - 1) + (2^k - 1) + 1 = 2^{k+1} - 1$. A binary system using $2k$ bits has a maximum range size of 2^{2k}. The efficiency of a k bit BSD system is then:

$$\frac{2^{k+1} - 1}{2^{2k}} \approx \frac{1}{2^{k-1}}$$

   Answer:

   The proposed system requires 5 bits per digit (to represent 21 different values). A k bit GSD system with these parameters has range $-(16^k - 1)/3$ to $16^k - 1$. The size of the range is $(16^k - 1)/3 + 1 + 16^k - 1 \approx 4 \times 16^k/3$. A binary system with $5 \times k$ bits has range size 2^{5k}. The efficiency of the GSD system in k digits is then $4 \times 16^k/(3 \times 2^{5k}) = 4/(3 \times 2^k)$. For a 16 digit system the efficiency is about 0.004%.

4. Is the representation of zero unique in the above GSD system? Explain.

   Answer:

   Yes, since the range of each digit does not exceed the radix in either direction, only one representation of zero is possible.

5. Devise an efficient way to negate a GSD number in this system.

   Answer:

   One way is to use the system to subtract the value from zero.

   Negative values will often require more digits than positive values since the range is unbalanced.

   Here is a digit-by-digit approach for this problem where the digit set is $[-5.9]$:

   When negating a negative digit, all that is necessary is to change the sign of the digit since the positive digit range exceeds the negative digit range.

   When negating a positive digit, if the value of the digit is in $[0, 5]$ just negate it.

   Otherwise, subtract it from 16. The result is the correct value for that digit position, but you must also carry -1 into the next column to the left. The carries will not propagate further.

   Example:

```
     -(-5  12  15   4  -4   1) = -4394047

=     ( 5   4   1  -4   4  -1) Intermediate result
     +(-1  -1   0   0   0   0) Carries
      ------------------------
=     ( 4   3   1  -4   4  -1) =  4394047
```