Journal of Universal Computer Science, vol. 3, no. 10 (1997), 1121-1125
submitted: 23/6/97, accepted: 2/9/97, appeared: 28/10/97 © Springer Pub. Co.

A Note on Correctness Proofs for Overflow Detection Logic in Adders for d-th Complement Numbers

Bernd Rederlechner
(Telekom Entwicklungszentrum Südwest, Saarbrücken, Germany
Bernd.Rederlechner@ezsw.telekom.de)

Jörg Keller
(FernUniversität-GH Hagen, Germany
Joerg.Keller@FernUni-Hagen.de)

Abstract: When adding n-bit 2-th complement numbers, the result can be outside the range representable with n bits. A well–known theorem justifies the common overflow logic: Let $a, b \in \{0,1\}^n$ be the 2-th complement representations of signed integers $[a]$ and $[b]$, respectively, and let $c_0 \in \{0,1\}$ be the carry-in bit. Then, $[a] + [b] + c_0 \in \{-2^{n-1}, \ldots, 2^{n-1} - 1\}$ if and only if $c_n = c_{n-1}$, where c_i denotes the carry–bit from position $i-1$ to position i when adding the binary numbers a and b. We present a proof of this theorem which is much shorter than previous proofs. This simplification can save valuable time in computer science classes. With a small extension the proof even holds for d-th complement numbers. Although the proof technique is known by some specialists, nobody seems to have written it up. With this note, it is once documented in a precise form, thus avoiding re-invention.

Key Words: d-ary arithmetic, correctness proof, computer science education, overflow testing

Category: B.2, K.3.2

1 Introduction

Most microprocessors use 2-th complement numbers as internal representation for signed integers. The addition of 2-th complement numbers is done by adders for binary numbers. However, the sum of two n-bit 2-th complement numbers could be outside the range representable by n bits. There are two common methods to detect such an overflow. Either, one compares the two most significant carry bits. Alternatively, one doubles the most significant input bits, uses $(n+1)$-bit arithmetic and compares the two most significant sum bits. In both cases, an overflow has occured if and only if the bits are not equal.

To justify that adders for binary numbers correctly add 2-th complement numbers and that the above method to detect an overflow is correct, most proofs proceed in the following way: depending on the signs of the numbers to be added, different cases are discussed. [Hwang (79)], [Scott (85)] and [Spaniol (81)] introduce d-th complement numbers (also called radix-d numbers). Hwang applies the above technique for a more general proof and fills about two pages. Difficult notations make readability worse. [Spaniol (81)] leaves the proof about overflow logic to the reader and only proves that summation is correct in the absence of overflows. [Scott (85)] uses a different definition, argues informally about the correctness of summation in absence of overflows, and shifts the overflow detection for 2-th complement numbers into the excercises. The remaining texts that we investigated treat only 2-th complement numbers, which is sufficient for a basic course in computer architecture or arithmetic. [Hotz (72)], [Keller and Paul (95)] and [Omondi (94)] use the presented scheme with three cases, consuming about one to one and a half pages without notations. [Hoffmann (83)] derives a formula depending on the most significant bits of the numbers to be added and the most significant bit of the sum. He completes the proof by putting in all possible values for these bits, and thus discusses nine cases. [Hotz (90)] and [Savage (76)] shift the theorem into the exercise part, [Koren (93)] shows the different cases only informally, and [Hennessy and Patterson (90)] even justify nothing at all.

While none of these proofs is difficult, they are rather lengthy. Hence, presenting the proof in a computer architecture class takes away valuable time from other topics. We present a proof which is much shorter than previous proofs. It is easy to present as it only employs simple equivalence transformations and a sharp look at the last line. No cases are distinguished, and no additional lemmata are needed. Only some notations are introduced, the most of which will already be present when the proof is used in a course on the subject. With a small extension, the proof also works for d-th complement numbers with arbitrary base $d \geq 2$.

While the technique used is apparently known by some specialists, nobody seems to have written it up. To avoid that the technique is re-invented over and over, it is hereby once documented in a precise form.

2 Notations

We use notations from [Keller and Paul (95)], the definition of d-th complement is taken from [Hwang (79)] and [Spaniol (81)]. Let $B_d = \{0, \ldots, d-1\}$. For $a' = (a_{n-2}, \ldots, a_0) \in B_d^{n-1}$, we define $\langle a' \rangle_d = \langle a_{n-2}, \ldots, a_0 \rangle_d = \sum_{i=0}^{n-2} a_i \cdot d^i$ and call a' the *d-ary representation* of $\langle a' \rangle_d$. To represent signed integers, we introduce a sign digit $a_{n-1} \in \{0, d-1\} =: B'_d$. By $a = (a_{n-1}, a') \in B'_d \times B_d^{n-1}$ we represent the number $[a]_d$ defined as

$$[a]_d = [a_{n-1}, a']_d = -\frac{a_{n-1}}{d-1} \cdot d^{n-1} + \langle a' \rangle_d = \begin{cases} \langle a' \rangle_d & \text{if } a_{n-1} = 0 \\ \langle a' \rangle_d - d^{n-1} & \text{if } a_{n-1} = d-1 \ . \end{cases}$$

The string a is called the *d-th complement representation* of $[a]_d$. Obviously, $\langle a' \rangle_d \in S_{n-1}^d := \{0, \ldots, d^{n-1} - 1\}$ and $[a]_d \in T_n^d := \{-d^{n-1}, \ldots, d^{n-1} - 1\}$. We will omit the subscript d if the base d is clear from the context.

For base $d = 2$, string $a' \in \{0,1\}^{n-1}$ is the binary representation of the non-negative integer $\langle a' \rangle_2 = \sum_{i=0}^{n-2} a_i \cdot 2^i \in S_{n-1} = \{0, \ldots, 2^{n-1} - 1\}$, and $a = (a_{n-1}, a') \in \{0,1\}^n$ is the 2-th complement representation of integer $[a]_2 = -a_{n-1} \cdot 2^{n-1} + \langle a' \rangle_2 \in T_n = \{-2^{n-1}, \ldots, 2^{n-1} - 1\}$.

When adding numbers $\langle a \rangle_d$ and $\langle b \rangle_d$ with d-ary representations $a, b \in B_d^n$ and a carry–in bit $c_0 \in \{0,1\}$, the sum is $\langle c_n, s \rangle_d = \langle a \rangle_d + \langle b \rangle_d + c_0$ where $s = (s_{n-1}, s') \in B_d^n$ and $c_n \in \{0,1\}$. For all $1 \leq i \leq n$, we denote the carry bit from position $i-1$ to position i by c_i, i.e.

$$\begin{aligned} \langle c_i, s_{i-1}, \ldots, s_0 \rangle_d &= c_i \cdot d^i + \langle s_{i-1}, \ldots, s_0 \rangle_d \\ &= \langle a_{i-1}, \ldots, a_0 \rangle_d + \langle b_{i-1}, \ldots, b_0 \rangle_d + c_0 \ . \end{aligned} \quad (1)$$

We have

$$a_i + b_i + c_i = d \cdot c_{i+1} + s_i,\ 0 \leq\ i\ \leq n-1\ . \tag{2}$$

The common overflow logic and the use of d-ary adders for d-th complement numbers is justified by the following Theorem.

Theorem 1. *Let $a, b \in B'_d \times B_d^{n-1}$ and $c_0 \in \{0,1\}$, and let $c_n \in \{0,1\}$ and $s \in B_d^n$ such that $\langle a\rangle + \langle b\rangle + c_0 = \langle c_n, s\rangle$. Then,*

(a) $[a]+[b]+c_0 \in T_n$ if and only if $c_n = c_{n-1}$.
(b) If $[a]+[b]+c_0 \in T_n$, then $[a]+[b]+c_0 = [s]$.

Before we prove Theorem 1, note that [Scott (85)] defines

$$T_n^d = \{-d^n/2, \ldots, d^n/2-1\} \text{ (assume } d \text{ to be even)},$$

and for $a \in B_d^n$

$$[a]_d = -\left\lfloor \frac{a_{n-1}}{d/2} \right\rfloor \cdot d^n + \langle a\rangle_d = \begin{cases} \langle a\rangle_d & \text{if } a_{n-1} < d/2 \\ \langle a\rangle_d - d^n & \text{if } a_{n-1} \geq d/2\ . \end{cases}$$

For example $T_4^{10} = \{-5000, \ldots, 4999\}$, $[0000]_{10} = 0$, $[4999]_{10} = 4999$, $[5000]_{10} = -5000$, and $[9999]_{10} = -1$. For $d = 2$, this definition equals the one given by [Hwang (79)] and [Spaniol (81)].

However, for $d \geq 4$, only part (b) of Theorem 1 holds with Scott's definition. Part (a) does not hold as two counterexamples show. First, let $x = d/2-1$, $x' = d-2$, $c_0 = 0$, and $a = b = (x, 0, \ldots, 0) \in B_d^n$. Then, $[a]_d = [b]_d = d^n/2 - d^{n-1} \in T_d^n$, but $[a]_d + [b]_d + c_0 = d^n - 2d^{n-1} \notin T_d^n$. If we add a and b with a d-ary adder, we obtain $s = (x', 0, \ldots, 0) \in B_d^n$, and for all $i = 1, \ldots, n$, $c_i = 0$ and thus $c_n = c_{n-1}$.

Second, let $\tilde{x} = d-1$, $a = b = (0, \tilde{x}, \ldots, \tilde{x}) \in B_d^n$, and $c_0 = 0$. Then, $[a]_d = [b]_d = d^{n-1} - 1 \in T_d^n$, and $[a]_d + [b]_d + c_0 = 2d^{n-1} - 2 \in T_d^n$. If we add a and b with a d-ary adder, we obtain $s = (1, \tilde{x}, \ldots, \tilde{x}, x') \in B_d^n$, $c_n = 0$, and $c_i = 1$ for all $i = 1, \ldots, n-1$, and thus $c_n \neq c_{n-1}$.

3 Proof for 2-th Complement Numbers

Proof. The definition of 2-th complement numbers gives:

$$\begin{aligned}
[a]+[b]+c_0 &= -2^{n-1}(a_{n-1}+b_{n-1}) + \langle a'\rangle + \langle b'\rangle + c_0 \\
&= -2^{n-1}(a_{n-1}+b_{n-1}) + \langle c_{n-1}, s'\rangle \\
&= -2^{n-1}(a_{n-1}+b_{n-1}-c_{n-1}) + \underbrace{\langle s'\rangle}_{\in S_{n-1}}
\end{aligned} \tag{3}$$

With $d = 2$ and $i = n-1$, we can transform (2) to $a_{n-1} + b_{n-1} = 2 \cdot c_n + s_{n-1} - c_{n-1}$. Together with (3), we have

$$\begin{aligned}
[a]+[b]+c_0 &= -2^n(c_n - c_{n-1}) - 2^{n-1}s_{n-1} + \langle s'\rangle \\
&= -2^n(c_n - c_{n-1}) + \underbrace{[s]}_{\in T_n}
\end{aligned} \tag{4}$$

If $c_n = c_{n-1}$, then the term $(c_n - c_{n-1})$ in (4) equals zero and the "if" direction of (a) as well as (b) holds. If $c_n \neq c_{n-1}$, then $-2^n(c_n - c_{n-1}) \in \{-2^n, 2^n\}$, and for all $[s] \in T_n$ one observes that $[s] \pm 2^n \notin T_n$. This proves the "only if" direction of (a).

4 Extension to d-th Complement Numbers

If we repeat the proof for 2-th complement numbers with the formulae for arbitrary d, we get the following equivalent to (4):

$$[a]_d + [b]_d + c_0 = -\frac{d^n}{d-1}(c_n - c_{n-1}) \; + \; \underbrace{[s]_d}_{\in T_n^d}$$

Again, we see that the "if" direction of (a) and (b) hold. However, $[s]_d \pm d^n/(d-1)$ is not necessarily outside T_n^d because $d^n/(d-1) < 2d^{n-1}$ for $d \geq 3$.

If $c_n \neq c_{n-1}$, then either $c_{n-1} = 0$ and $c_n = 1$ or vice versa. If $c_{n-1} = 0$ and $c_n = 1$, then $a_{n-1} = b_{n-1} = d-1$ because of (2), and $\langle a' \rangle + \langle b' \rangle + c_0 \leq d^{n-1} - 1$ because of (1). It follows that

$$[a] + [b] + c_0 = -d^{n-1} \underbrace{\left(\frac{a_{n-1}}{d-1} + \frac{b_{n-1}}{d-1} \right)}_{=2} + \underbrace{\langle a' \rangle + \langle b' \rangle + c_0}_{\leq d^{n-1}-1} \leq -d^{n-1} - 1 \; .$$

If $c_{n-1} = 1$ and $c_n = 0$, then $a_{n-1} = b_{n-1} = 0$ because of (2), and $\langle a' \rangle + \langle b' \rangle + c_0 \geq d^{n-1}$ because of (1). It follows that

$$[a] + [b] + c_0 = -d^{n-1} \underbrace{\left(\frac{a_{n-1}}{d-1} + \frac{b_{n-1}}{d-1} \right)}_{=0} + \underbrace{\langle a' \rangle + \langle b' \rangle + c_0}_{\geq d^{n-1}} \geq d^{n-1} \; .$$

Hence, in both cases $[a] + [b] + c_0 \neq T_n$.

References

[Hennessy and Patterson (90)] Hennessy, J. L., Patterson, D. A.: "Computer Architecture: A Quantitative Approach"; Morgan Kaufmann, San Francisco (1990)

[Hoffmann (83)] Hoffmann, R.: "Rechenwerke und Mikroprogrammierung, 2nd Edtion"; Oldenburg, München (1983)

[Hotz (72)] Hotz, G.: "Informatik: Rechenanlagen"; Teubner, Stuttgart (1972)

[Hotz (90)] Hotz, G.: "Einführung in die Informatik"; Teubner, Stuttgart (1990)

[Hwang (79)] Hwang, K.: "Computer Arithmetic. Principles, Architecture, and Design"; Wiley & Sons, News York (1979)

[Keller and Paul (95)] Keller, J., Paul, W. J.: "Hardware Design"; Teubner, Leipzig (1995)

[Koren (93)] Koren, I.: "Computer Arithmetic Algorithms"; Prentice Hall, Englewood Cliffs (1993)

[Omondi (94)] Omondi, A. R.: "Computer Arithmetic Systems. Algorithms, Architecture and Implementation"; Prentice Hall, Englewood Cliffs (1994)

[Savage (76)] Savage, J. E.: "The Complexity of Computing"; Wiley & Sons, New York (1976)
[Scott (85)] Scott, N. R.: "Computer Number Systems and Arithmetic"; Prentice-Hall, Englewood-Cliffs (1985)
[Spaniol (81)] Spaniol, O.: "Computer Arithmetic"; Wiley, New York (1981)