Two's Complement Arithmetic

A.P. Thijssen and H.A. Vink

Delft University of Technology / Philips Research Eindhoven
Faculty of Information Technology and Systems
PO Box 5031, 26 00 GA Delft, The Netherlands
mail to A.P.Thijssen@its.tudelft.nl

Abstract

Two's complement arithmetic is generally structured as a collection of arithmetic rules, implementation rules and special tricks, of which the correctness and restrictions for application can hardly be verified systematically. This paper shows how two's complement arithmetic can be based on mapping the binary arithmetic onto a set of residue classes modulo 2^n. This set of residue classes appears to be a good basis for proving the two's complement arithmetic and for the subsequent implementation in hardware or software. This conclusion is true for addition and subtraction, as well as for multiplication and division. In this way an easy implementation and verification of binary arithmetic in hardware or software is possible.

Key words

Addition, binary arithmetic, multiplication, subtraction, two's/one's complement arithmetic

1. Introduction

The usual pencil-and-paper notation for signed integers is as sign-and-magnitude. In early days of computer arithmetic it was found that sign-and-magnitude arithmetic could not be implemented economically. The circuits became relatively complicated because of the separate processing of the sign bit, the use of addition as well as subtraction in the implementation, and the need for interchanging operands when the result of a subtraction is negative. Other number representation systems were found, e.g. the two's and the one's complement, with better properties for implementation. This paper focuses on a theoretical basis for two's complement arithmetic by mapping the integers of the two's complement domain D_2^n onto the set of residue classes [m] modulo 2^n. In the domain D_2^n the two's complement system for addition/subtraction and for multiplication/division is closed. All these operations can be performed in a way that is isomorphic with normal integer arithmetic in the corresponding integer domain. For one's complement arithmetic a similar approach is possible. Extensions to nine's and ten's complement arithmetic are simple. Another way of interpreting two's complement binary numbers is described by [Dattatreya, 1993]. In his approach the

bitstring $m = a_{n-1}a_{n-2}\cdots a_{n-1}a_0$ BIN/TC corresponding to a binary two's complement number is interpreted in the sign-and-magnitude domain as

$$m = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} 2^i \qquad (1)$$

In proving the implementation of binary arithmetic this difference in the signs of the bits of the binary representation introduce many implementation errors, in hardware as well as in software. The basic rules of two's complement arithmetic are discussed in [Sklar, 1972]. We assume the reader has some knowledge of the elementary rules of computer arithmetic.

Numbers modulo m

Two integers m_1 and m_2 are said to be congruent modulo integer m iff m is a divisor of $m_1 - m_2$. We denote this relation as $m_1 = m_2 \pmod m$. A congruence relation on the set of integers is reflexive, symmetric and transitive. It is an equivalence relation on the set of integers [Hartmanis, 1966]. An equivalence relation induces a partition of disjunct equivalence classes on the set of integers. For a congruence relation these classes are called *residue classes* modulo m. Modulo m there are m residue classes,

$$[0] = \{0 + k \cdot m \mid k \in I\} \text{ in which I is the set of integers}$$

$$\vdots$$

$$[m-1] = \{m - 1 + k \cdot m \mid k \in I\}$$

A residue class [n] (mod m) can be represented by *any* of its members, from which the other members can be generated by:

$$[n] = \{n + k \cdot m \mid k \in I\}$$

In this paper we represent a residue class [n] (mod m) by the unique integer R(n) from [n] with $0 \le R(n) < m$. In binary notation, $R(n)_{BIN}$, this decision leads to the shortest binary bitstrings for representing the residue classes.

2. The two's complement representation

In the two's complement system the integers m from the domain D_2^n:

$$-2^{n-1} \le m \le 2^{n-1} - 1 \qquad (2)$$

are mapped onto [m] (mod 2^n). This residue class is represented by $R_2^n(m)$, which is the smallest non-negative integer in the residue class [m] (mod 2^n). The binary representation of the integer number $R_2^n(m)$ is denoted as $R_2^n(m)_{BIN}$.

(The two's complement domain is slightly asymmetrical, but excluding -2^{n-1} will give problems with overflow detection in a hardware binary adder.)
In doing so we may write

$$0 \le m \le 2^n - 1 \leftrightarrow R_2^n(m) = m \leftrightarrow 00\cdots00_{BIN} \le R_2^n(m)_{BIN} \le 01\cdots11_{BIN} \quad (3.a)$$

for all non-negative integers in D_2^n and

$$-2^{n-1} \le m \le -1 \leftrightarrow R_2^n(m) = 2^n + m \leftrightarrow 10\cdots \le R_2^n(m)_{BIN} \le 11\cdots11_{BIN} \quad (3.b)$$

From 3.a and 3.b we conclude that the binary images of non-negative integers begin with a leading 0 and the images of the negative integers with a leading 1. In binary arithmetic this bit is the *sign bit*. However, 'sign' stems from the original integer domain D_2^n. On the residue class level 'sign' has no meaning, as all $R_2^n(m)$ are non-negative numbers.

Adding two numbers m_1 and m_2 in D_2^n results in a sum $m_1 + m_2$ with

$$-2^n \le m_1 + m_2 \le 2^n - 2 \quad (4)$$

The domain of the sum exceeds D_2^n, but lies within D_2^{n+1}. Within D_2^{n+1} the addition of two numbers from D_2^n is closed. Within D_2^n there may be an overflow. The same holds for multiplication: within D_2^{2n} the multiplication of two binary n-bit numbers is always possible and within D_2^n not. The consequence is that in two's complement arithmetic we should extend the domain, or there must be an overflow detection. After an overflow the user must correct the result 'manually'.

3. The two's complement addition

Theorem 1 *The addition theorem*
When m_1, m_2 and $m_1 + m_2$ are in D_2^n then

$$R_2^n(m_1 + m_2) = R_2^n(m_1) + R_2^n(m_2) \pmod m \quad (5)$$

Proof
For m_1 and $m_2 \ge 0$, the proof follows directly from 3.a.

For m_1 and $m_2 < 0$, the proof follows from 3.b:

$$\begin{aligned} R_2^n(m_1) + R_2^n(m_2) \pmod{2^n} &= 2^n + m_1 + 2^n + m_2 \pmod{2^n} \\ &= 2^n + m_1 + m_2 \\ &= R_2^n(m_1 + m_2) \pmod{2^n} \end{aligned} \quad (6)$$

For the other combinations, m_1 or $m_2 < 0$, the proof is similar. □

In an n-bit binary adder the addition modulo 2^n can easily be implemented by skipping the highest order carry bit C_n in the n-bit binary adder.

Overflow detection

It can easily be verified that, when $m_1 + m_2 \geq 2^{n-1}$, their sum modulo 2^n is mapped onto a residue class in the domain of the negative numbers. We conclude that in binary the sign bit of $R_2^n(m_1 + m_2)_{BIN} \pmod m$ is a 1. This means that for non-negative integers m_1 and m_2 their sum $R_2^n(m_1)_{BIN} + R_2^n(m_2)_{BIN} \pmod{2^n}$, while adding in an n-bit binary full adder, the carry bits C_n and C_{n-1} are different. With no overflow $C_n = C_{n-1}$. So we have two criteria for overflow detection:

– the sign bit of the result is not as we expect (on the mathematical level);
– the carry bits in the binary adder are different (on the implementation level).

These criteria apply for the other combinations of m_1 and m_2 as well.

4. Subtraction in the two's complement

In two's complement the subtraction of $m_1 - m_2$ is done by the addition of $m_1 + (-m_2)$. When $R_2^n(m_1)$ and $R_2^n(m_2)$ are given, we need $R_2^n(-m_2)$ in order to use Theorem 1. For $R_2^n(-m_2)$ applies Theorem 2.

Theorem 2 *The complement theorem*

$$R_2^n(m) + R_2^n(-m) = 0 \pmod m \rightarrow R_2^n(-m) = 2^n - R_2^n(m) \pmod m \quad (7)$$

Proof

Simple from (3.a) and (3.b). □

From (7) it follows that

$$R_2^n(-m) = 2^n - R_2^n(m) = 2^n - 1 - R_2^n(m) + 1 \pmod{2^n} \quad (8)$$

or in binary

$$\begin{aligned} R_2^n(-m)_{BIN} &= 2^n - 1 - R_2^n(m)_{BIN} + 1 \pmod{2^n} \\ &= \text{Complement}(R_2^n(m)) + 1 \pmod{2^n} \end{aligned} \quad (9)$$

In determining $R_2^n(-m)$ (9) cannot be applied for $m = -2^{n-1}$ as $-m = +2^{n-1}$ is not in the two's complement domain D_2^n. This results in an overflow condition. Applying (9) to $m = -2^{n-1}$ we find $R_2^n(-2^{n-1})$. The difference is the modulus 2^n, as we should expect. Note that in a binary adder/complementer this overflow condition detects with $C_n \neq C_{n-1}$ as well. □

5. Expansion of the two's complement domain

During repeated addition of (binary) integers the sum may be greater than the appropriate integer domain D_2^n. Extension of the domain or an error handling procedure is necessary. Expanding the two's complement domain can be done with Theorem 3.

Theorem 3 *Domain expansion*

Let $R_2^n(m)$ be the representation of an integer m in D_2^n. The representation of m in D_2^{n+k} is

$$R_2^{n+k}(m) = a_{n-1}(2^{n+k} - 2^n) + R_2^n(m) \tag{10}$$

in which a_{n-1} is the 'sign' bit of the binary implementation. On the binary level this means a k-fold expansion of the sign bit.

Proof

For $m \geq 0$ it follows from (3.a/b) that

$$R_2^{n+1}(m) = R_2^n(m) \tag{11}$$

Then $a_{n-1} = 0$, so the theorem is true. For $m < 0$ from (3.a/b):

$$R_2^{n+1}(m) = 2^{n+1} + m = 2^n + 2^n + m = 2^n + R_2^n(m) = a_{n-1}2^n + R_2^n(m) \tag{12}$$

Thus

$$R_2^{n+1}(m) = a_{n-1}2^n + R_2^n(m) \text{ for all } m \in D_2^n \tag{13}$$

Repeated application of (13) proofs the theorem by induction. □

Conclusions

The two's complement addition and subtraction can be mapped on a set of residue classes modulo 2^n. It makes theorem proofing and algorithm verification very easy. Overflow is not an integral part of modulo arithmetic. However, on the binary implementation level the test for $C_n \neq C_{n-1}$ detects any overflow condition. (Note that in pipelined adders these carries may not be seen at the same pipeline level!)

So far the conversion of $R_2^n(m)$ into $\pm|m|$ has not been discussed. The conversion procedure can easily be derived from Formulae (3.a/b):

$$a_{n-1} = 0 \leftrightarrow |m| = R_2^n(m) = m \tag{14.a}$$

$$a_{n-1} = 1 \leftrightarrow |m| = 2^n - R_2^n(m) \tag{14.b}$$

On the binary level the conversion by (14.b) may be implemented by taking the complement of $R_2^n(m)+1$. □

6. Multiplication

Two's complement multiplication of integers in D_2^n must be done in D_2^{2n} or in D_2^n with an overflow detection. For multiplication in D_2^{2n} Theorem 4 applies.

Theorem 4 *The multiplication theorem*

$$R_2^{2n}(m_1 \cdot m_2) = R_2^{2n}(m_1) \cdot R_2^{2n}(m_2) \pmod{2^{2n}} \quad \text{with } m_1, m_2 \in D_2^n \qquad (15)$$

Proof

The proof can easily be found by verifying the theorem for all four cases of positive and negative integers in D_2^n. □

Example

With $m_1 = +3$ and $m_2 = -6$ we find

$$R_2^4(+3)_{BIN} = 0011_{BIN} \rightarrow R_2^8(+3) = 00000011_{BIN}$$

$$R_2^4(-6)_{BIN} = 1010_{BIN} \rightarrow R_2^8(-6) = 11111010_{BIN}$$

Multiplication of the binary 8-bit representations results into

```
                                0 0 0 0 0 0 1 1
                                1 1 1 1 1 0 1 0  x
                              ---------------------
                              | 0 0 0 0 0 0 0 0
                            0 | 0 0 0 0 0 1 1
                          0 0 | 0 0 0 0 0 0
                        0 0 0 | 0 0 0 1 1
                      0 0 0 0 | 0 0 1 1
                    0 0 0 0 0 | 0 1 1
                  0 0 0 0 0 0 | 1 1
                0 0 0 0 0 0 1 | 1
  ------------------------------------------------
  0 0 0 0 0 0 1 0 | 1 1 1 0 1 1 1 0  +
                  |
  skip mod 2^2n   | result is R2^8(-18)BIN
```

A drawback of direct multiplication of $R_2^{2n}(m_1)$ and $R_2^{2n}(m_2)$ is that the multiplication in a binary multiplier has 2n levels or steps in a series-parallel multiplier. In the well known Booth algorithm [Booth, 1951] the same multipli-

cation can be done in n steps. But Booth multiplication needs a number conversion in advance. Our algorithm implies faster in software, because of no number conversions.
For completeness we give Theorem 4.a. The proof is for the reader.

Theorem 4.a *The multiplication theorem*

$$R_2^{n+m}(m_1 \cdot m_2) = R_2^{n+m}(m_1) \cdot R_2^{n+m}(m_2) \pmod{2^{n+m}}$$
$$\text{with } m_1 \in D_2^n \text{ and } m_2 \in D_2^m \tag{16}$$

7. Other complement systems

In the past other complement systems have been in use. One of them is the one's complement. In the one's complement all calculations are done modulo $2^n - 1$. The one's complement theory develops in a similar way as we have shown for the two's complement. However, the one's complement has some drawbacks and nowadays the two's complement is most frequently be used. Reasons are the twofold representation of 0 in the one's complement, because of the second representation of $R_1^n(0) = 11\cdots11_{BIN}$ cannot be detected in a binary full adder or ALU. Domain extension also has some serious drawbacks while implementing multiplication and division in hardware.

8. Conclusions

There is a one-to-one mapping between two's complement arithmetic and the modulo 2^n arithmetic. On the modulo level theorem proofing and algorithm verification is much simpler. The result is a simpler verification of implementations in hardware and software.

Literature

1. A.D. Booth and B.A. Wooley, *A Two's Complement Parallel Array Multiplication Algorithm*, IEEE Trans. on Computers, Vol. C-22, 1973, pp. 1045-1047.
2. G.R.Dattatreya, *A systematic Approach to Teaching Binary Arithmetic in a First Course*, IEEE Trans. on Education, Vol. 36, Feb. 1993, pp. 163-168.
3. J. Hartmanis and R.E. Stearns, *Algebraic Structure Theory of Sequential Machines*, Prentice-Hall, Englewood Cliffs, N.J., 1966.
4. S. Sklar, *2's Complement Arithmetic Operators*, Computer Design, May 1972, pp. 115-121.