Signed Numbers Conversions

J.Vijayasekhar

Department of Mathematics,
GITAM University, Hyderabad

Y.Srinivas

Department of Computer Applications,
GITAM University, Hyderabad

N.Vamsi Krishna
Department of Mathematics,
GITAM University, Hyderabad

Abstract

Signed integers are normally represented using 2's complement representation. Addition and subtraction of signed numbers is done in the same manner as for unsigned numbers. However carry (or borrow) is simple ignored. Unlike unsigned number carry (or borrow) does not mean overflow or error. Doubling of a signed number can be done by shift left. However, halving of a signed number can not be done by shift right. Hence special arithmetic instruction SAR (Shift arithmetic right) is needed.

We have defined an alternative representation for signed numbers. Here a positive number is represented by appended a zero (0) at right. Here a negative number is represented by inverting all bits in corresponding positive number. Two signed numbers are added by adding corresponding binary representation. After that carry is added to the result. Similarly two signed numbers are subtracted by subtracting corresponding binary representation. After that borrow is subtracted. Doubling and halving is done by ROL (Rotate left) and ROR (Rotate right) respectively. Following are drawbacks of our system.

(A) Addition is done in two stages. In the first stage the numbers are added. In the second stage carry is added. Carry can not be ignored as in 2's complement representation.
(B) Same holds for subtraction.
(C) When an odd number is halved then error results. In 2's complement representation approximate answer appears.

The advantage of our system is that entire arithmetic can be carried using ordinary logical instructions. No special instruction is needed. In 2's complement representation a special instruction SAR is needed. This instruction is not used for any other purpose.

***Keywords*: SAR, ROL, ROR, Overflow**

1. Representation

Following is the method of representation of signed numbers in 2's complement.

(D) Find binary representation of corresponding unsigned number.
(E) A positive number is represented by putting 0 at the beginning.
(F) A negative number is represented by putting 1 at the beginning. Moreover all bits at the left of right most 1 are inverted.

Following is the method for representation of signed numbers in present scheme.

(A) Find representation of the corresponding unsigned number.
(B) A positive number is represented by putting 0 at the end.
(C) A negative number is represented by putting 1 at the end. All other bits are inverted. In present scheme the representation of two numbers of same magnitude and different sign are complement of each other.

Unsigned Number	Binary Representation	2's Complement Scheme		Present Scheme	
		Positive	Negative	Positive	Negative
11	1011	+11=**0**1011	-11=**1**0101	+11=1011**0**	-11=0100**1**
18	10010	+18=**0**10010	-18=**1**01110	+18=10010**0**	-18=01101**1**
53	110101	+53=**0**110101	-53=**1**001011	+53=110101**0**	-53=0010101
27	11011	+27=**0**11011	-27=**1**00101	+27=11011**0**	-27=00100**1**
10	1010	+10=**0**1010	-10=**1**0110	+10=1010**0**	-10=0101**1**

Following method is used for converting a representation into signed number.

Representation	2's complement	Present scheme
010100	Left most bit is 0. Hence sign is (+) Since positive, hence no change. 010100 is 20 in absolute value. Hence 010100 is +20.	Right most bit 0. Hence sign is (+) Since positive, hence no change Drop right most bit.01010. 01010 is 10 in absolute value. Hence 010100 is +10
101011	Left most bit is 1. Hence sign is (-). Since negative, hence complement all bits before last one 010101. 010101 is 21 in absolute value. Hence 101011 is -21.	Right most bit 1. Hence sign is (-) Since negative, hence invert all bits 010100 Drop right most bit 01010. 01010 is 10 in absolute value Hence 010100 is -10
01111	Left most bit is 0. Hence sign is (+) Since positive, hence no change 01111 is 15 in absolute value Hence 01111 is +15	Right most bit 1. Hence sign is (-). Since negative, hence invert all bits and drop right most bit 1000 1000 is 8 in absolute value. Hence 01111 is -8.
101100	Left most bit is 1, hence sign is (-) Since negative, hence complement invert all bits before one 010100. 10100 is 20 in absolute value. Hence 010100 is +20	Right most bit is 0, Hence sign is (+) Since positive, hence no inversion Drop right most bit 10110 10110 is 22 in absolute value. Hence 101100 is +22

2. Addition

The numbers are added in an ordinary manner in both schemes. The only difference is in the method of handling carry.

Numbers	2's Complement Scheme	Present Scheme
+9 and -14	(A) +9 is 01001 (B) -14 is 10010 Addition 11011 It is -5 because 00101 is +5	(A) +9 is 10010 (B) −14 is 00011 Addition 10101 10101 is -5 because 01010 is +5
	Carry (if any) is ignored	Carry (if any) is added to the result
-10 and +12	(A) -10 is 10110 (B) +12 is 01100 Addition **1**00010 Carry is ignored. Hence result is 00010. it is +2	(A) −10 is 01011 (B) +12 is 11000 Add **1**00011 1 Carry addition 00100 It is +2

3. Expansion and Compression

When adding numbers of different sizes, the size of the smaller number (in magnitude) is expanded, so that they get the same size. To increase the size of an unsigned number additional 0's are put in the beginning (LSB). To

increase the size of a signed number the corresponding method is followed by increasing the size of unsigned representation.

Unsigned Number	Binary Representation	2's Complement Scheme		Present Scheme	
		Positive	Negative	Positive	Negative
12	01100	+12=001100	-12=110100	+12=011000	-12=100111
14	001110	+14=0001110	-14=1110010	+14=0011100	-14=1100011
20	010100	+20=0010100	-20=1101100	+20=0101000	-20=1010111

In brief the method of size expansion is as follows:

(A) In 2's complement scheme the size of a number is expanded by putting the copy of the MSB before MSB. +12 is 01100. It can be also written as 0001100. Similarly -12 is 10100. It can also be written as 1110100.
(B) In present scheme the size of a number is expanded by putting the copy of LSB at MSB. +12 is 11000. It can also be written as 0011000. Similarly –12 is 00111. It can also be written as 1100111.

Method of addition of the numbers of different sizes can be understood from the table.

Numbers	2's Complement Scheme	Present Scheme
+20 and +1	+20 is 010100 and +1 is 01. Size of +1 is increased. +20 is 010100 +1 is 000001 010101 It is +21	+20 is 101000 and +1 is 10. Size of +1 is increased. +20 is 101000 +1 is 000010 Add 101010 It is +21
+26 and -4	+26 is 011010 -4 is 111100 (expand) Add 010110 It is +22	+26 is 110100 –4 is 110111 (expand) add 101011 1 101100 It is +22
During arithmetic operation it is also possible that the size of result is small. Hence size compression takes place. It is done by removing most significant bits. (A) In 2's complement representation if MSB and second most significant bits are same then MSB is removed. (B) In present scheme if MSB=LSB then this removal can be done.		
-27 and +22	-27 is 100101 +22 is 010110 Add 111011 It is -5 It's size is compressed as 1011	–27 is 001001 +22 is 101100 addition 110101 It is –5. It's size is compressed as 0101.

4. Subtraction

The numbers are subtracted in similar manner

Numbers	2's Complement Scheme	Present Scheme
+13 and +4	+13 is 01101 +4 is 00100 (size expansion) 01001 It is +9	+13 is 11010 +4 is 01000 (size expansion) 10010 It is +9
	Borrow (if any) is ignored	Borrow (if any) is subtracted from the result
+23 and -5	+23 is 010111 -5 is 111011 (size expansion) **1** 011100 It is +28	+23 is 101110 –5 is 110101 (size expansion) 111001 1 Borrow Subtraction 111000 It is +28

Overflow

During addition and subtraction it is possible that the size of the result is more than the size of the bigger number (in magnitude). It causes overflow. The solution of this problem is that the size of the bigger number is increased by 1 (by putting copy of the MSB in the beginning in 2's complement scheme and copy of the LSB in the beginning in present scheme).

Numbers	2's Complement Scheme	Present Scheme
+10 and +12	(A) +10 is 01010 (B) +12 is 01100 10110 It is -10	(A) +10 is 10100 (B) +12 is 11000 01100 1 01101 It is –9
+10 and +12	Let us increase the size (A) +10 is 001010 (B) +12 is 001100 010110 It is +22	Let us increase the size. (A) +10 is 010100 (B) +12 is 011000 101100 It is +22

5. Multiplication and **Division** by 2

A signed number in 2's complement representation can be doubled by SHL (shift left) operation. Here every bit is shifted left. The left most bit is removed. Zero (0) is appended to the right. A signed number in 2's complement representation can be halved by SAR (shift arithmetic right) operation. Here every bit is shifted right. The right most is removed. Left most bit is retained. The difference between SHR (shift right) and SAR is that in SHR the left most bit is made 0.

Let us take a word HYDERABAD

(A) After shift left it will become YDERABAD0.
(B) After SAR it will become HHYDERABA.
(C) After SHR it will become 0HYDERABA.

A signed number in present scheme can be doubled by ROL (rotate left) operation. In rotate left operation every bit is shifted left and the left most bit (MSB) is transferred to the right (LSB). Similarly it can be halved by ROR (rotate right) operation. Here every bit is shifted to the right and the right most bit (MSB) is transferred to the left (LSB).

Let us take a word HYDERABAD

(A) After rotate left it will become YDERABADH.
(B) After rotate right it will become DHYDERABA.

Since during doubling the size of the number increases hence to avoid overflow the size is increase by 1 in the beginning. Similarly after halving a number the size decreases. Hence MSB can be removed.

Unsigned Number	Binary Representation With expansion	2's Complement Scheme SHL (Shift left)		Present Scheme ROL(Rotate left)	
10	01010	+10=001010	+20=010100	+10=010100	+20=101000
7	0111	+7=00111	+14=01110	+7=01110	+14=11100
11	01011	-11=110101	-22=101010	-11=101001	-22=010011

Unsigned Number	Binary Representation	2's Complement Scheme SAR (Shift arithmetic right)		Present Scheme ROR (Rotate right)	
10	1010	+10=01010	+5=00101	+10=10100	+5=01010
20	10100	-20=101100	-10=110110	-20=010111	-10=101011
7	111	+7=0111	+3=0011	+7=1110	-4=0111
13	1101	-13=10011	-7=11001	-13=00101	+9=10010

In 2's complement representation halving an odd number produces nearest integer. In present scheme halving of an odd number produces error. When +7 is divided by 2 then perfect division can not take place. Hence erroneous result comes. A number is odd number if last two bits differ. In this case the result is wrong.

6. Conclusion

Following table shows the comparison of both methods

2's complement	**Present scheme**
1. In the representation of a positive number zero (0) is appended at the left.	1. In the representation of a positive number zero (0) is appended at the right.
2. A negative number is represented by inverting only those bits, which are to the left of right most 1 in the representation of corresponding unsigned number.	2. A negative number is represented by inverting all bits in the representation of corresponding positive number.
3. In the addition of two signed numbers the carry is ignored.	3. In the addition of two signed numbers carry is added to the result.
4. In the subtraction of two signed number borrow is ignored.	4. In the subtraction of two signed number borrow is subtract from the result.
5. A number is doubled by shift left (SHL) operation.	5. A number is doubled by ROL (Rotate left) operation.
6. A number is halving by shift arithmetic right (SAR) operation.	6. A number is halving by rotate right (ROR) operation.
7. In halving an odd number lower bound is taken.	7. In halving an odd number error results.
8. In the representation of an odd number left most and right most bits differ.	8. In the representation of an odd number right most two bits differ.

Reference

[1] Ivan Flores, *The Logic of Computer Arithmetic*, Prentice-Hall (1963)

[2] Ytha Yu and Charles marut, Assembly language programming and organization of IBM PC, Mc Graw-Hill, 1992

[3] Peter Norton and John Socha, Peter Norton's Assembly language book for the IBM PC, Prentice-Hall, 1993.

[4] John F. Wakerly, *Digital Design Principles & Practices*, Prentice Hall, 3rd edition 2000, page 47

[5] Israel Koren, *Computer Arithmetic Algorithms*, A.K. Peters (2002), ISBN 1-56881-160-8

[6] David J. Lilja and Sachin S. Sapatnekar, *Designing Digital Computer Systems with Verilog*, Cambridge University Press, 2005 online

[7] Digital Design and Computer Architecture" by David Harris, David Money Harris, Sarah L. Harris. 2007. Page 18.

[8] *Digital Logic and Computer Design*, M. Morris Mano.