By : Elnekave Mark 1.4.98

Class topic :

The 2's complement numbers representation.

1. 2's complement definition.
2. 2's complement properties.
3. Adding in 2's complement.
4. Hardware implementation of a 2's complement adder.

2's complement definition

Definitions :

$x \in \{0,1\}^n$

$a \in \{0,1\}^n$

a and x are identical. The difference is in their usage:

$x_0\ x_1\ \ldots\ldots\ x_{n-1}$ (x_i has a weight of 2^{-i})

$a_{n-1}\ a_{n-2}\ \ldots\ldots\ a_0$ (a_i has a weight of 2^i)

x_i represents Fractions, while a_i represents integers.

Functions:

$<\cdot> : \{0,1\}^* \rightarrow N$

$$<a> = \sum_{i=0}^{n-1} (a_i \cdot 2^i)$$

This function transforms a binary string into an integer (always positive).

B_n denotes the set of integers that are representable by n-bit binary strings.

$B_n = \{ 0, 1, \ldots\ldots 2^{n-1}\}$

Define bin_n, function that transforms integers into a binary string (always positive).

$bin_n : B_n \rightarrow \{0, 1\}^n$

Note that these are inverse functions :

$\forall\ a \in \{0, 1\}^n$: $bin_n (<a>) = a$

We further define the function :

$[\cdot] : \{0, 1\}^* \rightarrow Z$

by : $[a_{n-1}\ \ldots\ldots\ a_0] = -a_{n-1}2^{n-1} + < a_{n-2}\ \ldots\ldots\ a_0>$

T_n - The group of integer numbers that can be represented by n bits in the 2's complement representation.

$T_n = \{-2^{n-1}, \ldots\ldots\ldots, 2^{n-1}-1\}$

where -2^{n-1} is the smallest number possible in this representation, and $2^{n-1}-1$ is the largest.

Note that T_n is not symmetric, the absolute value of the smallest number (-2^{n-1}), is larger than the absolute value of the largest number ($2^{n-1}-1$).

Define the inverse function for [] :

$two_n : T_n \rightarrow \{0, 1\}^n$

So that : $\forall a \in \{0, 1\}^n$: $[two_n(a)] = a$

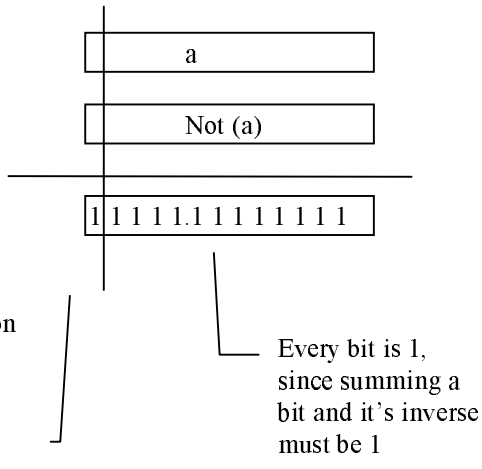
2's complement properties

1. Negative : $-[a] = not[a] + 1$
   The negative, of a number in the 2's complement representation, is it's logical not, plus one.
2. Sign extension : if $\tilde{a} = a_n \cdot a$, where $a_n = a_{n-1}$
   then $[\tilde{a}] = [a]$
   When duplicating the msb, the string representing the number is one bit longer but has the same value.

Proofs :

1. Negation of a 2's complement number

The condition $-[a] = not[a] + 1$
Is equivalent to $-1 = not[a] + [a]$
Consider the diagram :



So we've got :

$$[a] + [\bar{a}] = [1 \cdots\cdots 1] = -2^{n-1} + \sum_{i=0}^{n-2} (2^i) = -2^{n-1} + 2^{n-1} - 1 = -1 \qquad \text{Q.E.D}$$

Conclusion : in order to negate a 2's complement number, the bits should be inverted, and the result incremented by one.
This procedure is true for all numbers but one.
Consider -2^{n-1}, it's negative is not in T_n, so this procedure results in an n+1 bits long string, which may cause an error in some implementations.

2. Sign extension of a 2's complement number
Define $\tilde{a} = a_n \cdot a$, where $a_n = a_{n-1}$
Then :
$[\tilde{a}] = -2^n \cdot a_n + 2^{n-1} \cdot a_{n-1} + < a_{n-2} \ldots\ldots a_0 >$
but $a_n = a_{n-1}$, so :

$[\tilde{a}] = -2^{n-1} \cdot a_{n-1} + < a_{n-2} \ldots\ldots a_0 > = [a]$ Q.E.D

(true for both $a_{n-1}=0$, and $a_{n-1}= 1$)

Clearly, this process of adding a sign extension bit can be repeated any number of times.

Also a_n can be deleted, if $a_n = a_{n-1}$ (the inverse of the sign extending operation)

* The geometric interpretation of 2's complement :

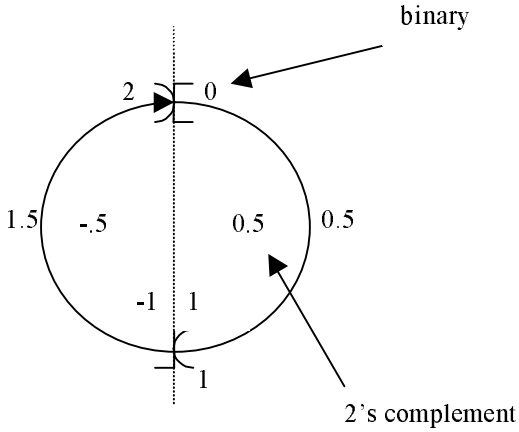
Consider x of the form : $x = x_0.x_1 \ldots$

In binary interpretation - this string represents a number of the interval [0,2)

In 2's complement – the interval is [-1,1)

2's complement can be interpreted as a mapping function :

f: [-1:1) -> [0,2)



In the right half of the circle, the two representations are equal, for example, the string that represents the number 0.5 is the same.

But in the left half, the difference between the two representations is 2, for example, the string that represents the number 1.5, in binary representation, represents -0.5 in 2's complement.

$a \in [0,1]$ => -a -> 2-a

And that's the origin of the name 2's complement.

Adding in 2's complement

Theorem:
Let,
$a, b \in \{0,1\}^n$
$c_0 \in \{0,1\}$
$s \in \{0,1\}^n$
$c_n \in \{0,1\}$

And assume that $<a>+<b>+c_0=<c_n \cdot s>$ ($\cdot$ - denotes concatenation)
Mark the carry bit i by c_i, namely
$<a_{i-1} \dots a_0>+<b_{i-1} \dots b_0>+c_0=c_i \cdot 2^i+<s_{i-1} \dots s_0>$

Define, $z=[a]+[b]+c_0$
Then,
1. if $z \in T_n$ then $[s]=z$ and $c_n = c_{n-1}$
2. if $z \notin T_n$ then $c_n \neq c_{n-1}$
3. (the extension of 2.)
   if $c_n=0$ and $c_{n-1}=1$ then $z>2^{n-1}-1$
   if $c_n=1$ and $c_{n-1}=0$ then $z<2^{n-1}-1$

This theorem means that the addition of 2 numbers (n bits long), in the 2's complement representation, produces the correct sum, represented in the 2's complement representation (providing that there is no overflow).
Overflow occurs in either of the conditions defined in the 3'rd part of the theorem.

Proof :

Let, $a'=a_{n-2} \dots a_0$
$a=a_{n-1} \cdot a'$

Then, by definition,
$z = [a]+[b]+c_0 =$
$= -2^{n-1}a_{n-1}+<a'>+-2^{n-1}b_{n-1}+<b'>+c_0 =$
$= -2^{n-1}(a_{n-1}+b_{n-1})+<a'>+<b'>$

but : $a_{n-1} + b_{n-1} + c_{n-1} = s_{n-1} + 2c_n$ (c_n has double weight)
so $(a_{n-1}+b_{n-1})$ can be replaced by : $s_{n-1} + 2c_n - c_{n-1}$
and $<a'>+<b'> = <s>$ since it's normal binary adding

So we get :
$z = -2^{n-1}s_{n-1}+<s'>-2^{n-1}(2c_n-2c_{n-1})$

and $-2^{n-1}s_{n-1}+<s'> = [s]$ so finally :

$z = [s] - 2^n(c_n - c_{n-1})$

Now we can diagnose the cases :

a. $c_n = c_{n-1}$ then $z = [s]$

b. $c_n = 1$ and $c_{n-1} = 0$ then

$z = [s] - 2^n <= 2^{n-1} - 1 - 2^n = -2^{n-1} - 1$ (since $[s] <= 2^{n-1} - 1$)

therefore $z \notin T_n$ (because z is too small)

c. $c_n = 0$ and $c_{n-1} = 1$ then

$z = [s] + 2^n >= -2^{n-1} + 2^n = 2^{n-1}$

therefore $z \notin T_n$ (because z is too big)

Q.E.D

Note that an ordinary adder does not output c_{n-1}. The outputs of an ordinary adder are $<s>$ and c_n, so, in order to use it for 2's complement adding the output c_{n-1} must be available.

Note too that condition 3. is equal to $XOR(c_n, c_{n-1})$, resulting in that when $XOR(c_n, c_{n-1}) = 1$ then $z \notin T_n$

Hardware implementation of a 2's complement adder.

Definition :

An additive circuit with inputs :

$a, b \in \{0,1\}^n$

$c_0 \in \{0,1\}$

And outputs :

$s \in \{0,1\}^n$

$ovf, neg \in \{0,1\}$

The circuit's products :

1. if $z \in T_n$ then $[s]=z$ and $ovf=0$
2. if $z \notin T_n$ then $ovf=1$
3. if $z<0$ then $neg=1$

According to the theorem, a normal binary adder is sufficient. It functions as a 2's complement adder too (since [s]=z, unless there is an overflow).

$<a>+<b>+c_0=<s>+2^n c_n$

Additionally, the "neg", and "ovf" flags need to be computed, in order to indicate a negative result, and overflow of the result (accordingly),

It was previously shown that : $ovf=XOR(c_n,c_{n-1})$

Claim :

$neg=XOR(c_n,a_{n-1},b_{n-1})$

Proof :

Consider a" defined as $a"=a_n \cdot a$ (sign extension by one bit)

And z defined as $z=[a"]+[b"]+c_0$

[a], [b] are n-bit long, so cannot be bigger than $2^{n-1}-1$, by definition of 2's complement.

Therefore [a"],[b"] cannot be bigger than $2^{n-1}-1$, since they are only sign extension of [a] and [b] respectively.

But z can reperesent 2^n-1 numbers, since it is n+1 bit long.

Therefore In this adding there cannot be an overflow, since [a"], [b"]$<=2^{n-1}$,

so that $z <= 2(2^{n-1}-1) + 1 = 2^n - 1$,

In other words, it is always true that $z \in T_{n+1}$,

Therefore (according to the theorem),

$<a">+<b">+c_0=<c_{n+1} \cdot s">$

and $z=[s"]$ (overflow is impossible, because $z \in T_{n+1}$)

so sign(z)=s''_n (the sign of z is the n'th bit of s")

but $s''_n = a_n \oplus b_n \oplus c_n =$

$= a_{n-1} \oplus b_{n-1} \oplus c_n$ ($a_n=a_{n-1}$, $b_n=b_{n-1}$ according to the sign extension theorem)

Q.E.D

Block diagram of the 2's complement adder :

It was proven that a normal adder sums 2's complement numbers correctly. The negative, and overflow flags (neg, ovf), are usually implemented too, in order to indicate a negative result, and overflow (accordingly).

We have reached the following implementations for the flags :

neg=XOR(c_n,a_{n-1},b_{n-1})

ovf=XOR(c_n,c_{n-1})