CSE 271 — Introduction to Digital Systems
Supplementary Reading
Representation of Signed Numbers

There are many ways to represent signed numbers. Typically the MSB of a bit string is used to represent the sign (the sign bit). Since the MSB is used to indicate the sign (0=plus, 1=minus), an n-bit number can only represent nonnegative numbers from 0 to $2^{n-1} - 1$ (instead of 0 to $2^n - 1$ as for unsigned numbers).

To ease the implementation of subtraction using digital circuits, we would also impose

Requirement A: The subtraction $N1 - N2$ can be carried out by the addition of the two numbers $N1$ and $(-N2)$. □

Here the addition is carried out similarly to that of unsigned numbers. If the Requirement A is satisfied by the representation, then in designing a digital system, subtraction circuits need not be separately designed once the addition circuits are available.

Let us look at the following candidate representations for signed numbers.

Signed-Magnitude Representation

In the signed-magnitude representation, a number consists of a magnitude string and a symbol indicating the sign of the number. The sign symbol is at the MSB. The rest of the bits form the magnitude and are interpreted similarly to unsigned numbers. For example, the 4-bit words $0110_2 = 6_{10}$, $1101_2 = -5_{10}$. Now consider $6 - 5$. Direct subtraction yields $0110 - 0101 = 0001$. However, if we express it as $6 + (-5)$ and carry out the addition, we have $0110 + 1101 = 10011$ and so the 4-bit sum word is 0011 (due to the 4-bit word length). Since $0011 \neq 0001$, the Requirement A is not satisfied.

1's Complement Representation

In the 1's complement representation, a nonnegative number is represented in the same manner as an unsigned number. A negative number $(-N)$ is represented by the 1's complement of the positive number N. The 1's complement of an n-bit number N is obtained by complementing each bit of N (or equivalently, by subtracting it from $2^n - 1$). For example, the 4-bit words $0110_2 = 6_{10}$, $0101_2 = 5_{10}$, and $1010_2 = -5_{10}$. Now consider $6 - 5$. Direct subtraction yields $0110 - 0101 = 0001$. However, if we express it as $6 + (-5)$ and carry out the addition, we have $0110 + 1010 = 10000$ and so the 4-bit sum word is 0000 (due to the 4-bit word length). Since $0000 \neq 0001$, the Requirement A is not satisfied.

2's Complement Representation

Now we introduce the 2's complement representation which satisfies the Requirement A. Due to this reason, it is the most commonly used representation for signed binary numbers. In the the 2's complement number system, we have the following representations.

Nonnegative Numbers: Represented in the same manner as an unsigned number.

Negative Numbers: A negative number $(-N)$ is represented by 2's complement of the positive number N.

The 2's complement of an n-bit number N is obtained by subtracting it from 2^n. Note $2^n - N = [(2^n - 1) - N] + 1$ and the operation $[(2^n - 1) - N]$ entails the complementing of each bit of N. So the 2's complement of N can simply be obtained by *complementing each bit of N and then adding* 1. The followings are some examples of 2's complement representations.

Examples. The 2's complement representation of the decimal number 6 is 0110. The 2's complement representation of -6 is obtained by the following procedure.

$$\begin{array}{rl} 6_{10} & = 0110 \\ & \quad 1001 \\ + & \quad\ \ 1 \\ \hline & 1010 = \ -6_{10} \end{array}$$ complement bits

Note that the MSB 1 indicates that 1010 represents a negative number. □

In fact, the 2's complement number system negates a number by taking its 2's complement. So the complement operation can also be applied to a negative number representation to obtain the corresponding positive number representation[1]. For example

$$\begin{array}{rl} -6_{10} & = 1010 \\ & \quad 0101 \\ + & \quad\ \ 1 \\ \hline & 0110 = \ \ 6_{10} \end{array}$$ complement bits

Remarks.

(a) Given a word size of n bits, the range of 2's complement binary numbers is -2^{n-1} through $2^{n-1} - 1$.

(b) The 2's complement of an n-bit all 0 string is itself.

(c) The 2's complement of an n-bit string with all 0's except for the MSB being 1 is itself. For example, the complement of a 4-bit word 1000 is 1000 and it represents $-2^3 = -8$ and has no positive counterpart (since 8 is not with the range). □

Decimal Equivalent Values for 2's Complement Binary Numbers. Given a binary number in 2's complement representation, there are two methods for determining its decimal equivalent value.

Method 1: If the MSB is 0, then the number is nonnegative and its value can be determined similarly to an unsigned number. If the MSB is 1, then the number is negative and its absolute value can be determined by taking the 2's complement of the given negative number. For example, given a 4-bit number $N = 1101$. We apply the following procedure to determine the 2's complement of N (i.e., the negation of N).

$$\begin{array}{rl} N = & 1101 \\ & 0010 \\ + & \quad 1 \\ \hline & 0011 = \ \ 3_{10} \ \text{(2's complement of N)} \end{array}$$ complement bits

Hence the decimal value of N is -3_{10}, i.e., $1101_2 = -3_{10}$.

Method 2: The decimal value for an n-bit 2's complement binary number is computed the same way as for an unsigned number using the formula of weighted summation of powers of 2, except that the power term corresponding to the MSB is (-2^{n-1}) instead of 2^{n-1}. For example, the decimal value of the 4-bit number $N = 1101$ can be computed as

$$N = 1 \times (-2^3) + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = -3_{10}$$

The justification of Method 2 is given in the footnote[2].

---

[1]To justify this, note that the 2's complement representation of an n-bit negative number $(-N)$ is given by $2^n - N$. Now take the 2's complement of $2^n - N$ yields $2^n - (2^n - N) = N$. This is equivalent to saying that the 2's complement of the representation of $(-N)$ gives us the representation of the corresponding positive number N.

[2]For positive numbers, the MSB is 0 so it has no contribution in evaluating the value. While for a negative number $(-N)$, its 2's complement representation is the unsigned binary representation for $D = 2^n - N$. Since the decimal value for D as an unsigned number is $D = \sum_{i=0}^{n-1} d_i \cdot 2^i$ with $d_{n-1} = 1$, we then have $-N = D - 2^n = 1 \times (-2^{n-1}) + \sum_{i=0}^{n-2} d_i \cdot 2^i$.

Sign Extensions. When dealing with hardware, we often need to increase the number of bits required to represent a signed number. In general, to extend an n-bit number to an m-bit number ($m > n$) which has the same decimal value, we simply pad the given n-bit number with ($m - n$) copies of its MSB to its left to form the corresponding m-bit number. For example, given a 4-bit number $1110_2 = -2_{10}$, we can extend it to an equivalent 8-bit number $11111110_2 = -2_{10}$. Similarly, we can extend $0011_2 = 3_{10}$ to $00000011_2 = 3_{10}$.

Comparison of Different Representations

Now let us compare the aforementioned three representations by studying the following table for 4-bit numbers.

Decimal	2's Complement	1's Complement	Signed-Magnitude
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000	0000 or 1111	0000 or 1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	—	—

From the above table, it can be observed why the 2's complement is preferred for arithmetic operations. If we start with 1000_2 (-8_{10}) and count up, we see that each successive 2's complement number all the way to 0111_2 (7_{10}) can be obtained by adding 1 to the previous one, ignoring any carries beyond the fourth bit position. The same cannot be said of signed-magnitude and 1's complement numbers. Because ordinary addition is just an extension of counting, 2's complement numbers can thus be added by ordinary binary addition, ignoring any carries beyond the MSB. The result will always be the correct sum as long as the range of the number system is not exceeded. This helps explain why the Requirement A is satisfied by 2's complement numbers. Moreover, note that the range of 2's complement numbers is larger than that of signed-magnitude and 1's complement (for which 0_{10} has 2 representations).

2's Complement Addition and Subtraction

Since 2's complement numbers satisfy the Requirement A mentioned at the beginning of this handout, we only need to consider the addition of 2's complement numbers. As we have mentioned, 2's complement numbers can thus be added by ordinary binary addition. Some examples are given on the next page.

In Examples (e) and (f), the result is incorrect since the decimal value of the sum exceeds the range of 4-bit 2's complement number system. In such cases, *overflow* is said to occur. As can be observed from the examples, in general, the addition of 2's complement numbers has the following properties.

(a) Addition of two numbers with different signs can never produce overflow and thus the result is always correct (ignoring the carries beyond the MSB). Such is the case for Examples (a) and (b).

(b) An addition overflows if the two addends' signs are the same, but the sum's sign is different from the addends. Such is the case for Examples (e) and (f). In Examples (c) and (d), the sums have the same sign as the addends, so the results are correct.

(c) Finally, here is *an easy method to determine whether overflow occurs: overflow occurs if and only if the carry bit c_{in} into and c_{out} out of the sign position (i.e., the MSB) are different.* Such is the case for Examples (e) and (f).

<div>

```
1 1          corresp.
             dec. oper.
  0110            +6
 +1101        +   −3
─────         ─────
10011 = +3        +3

       correct result

        Example (a)
```

```
             corresp.
             dec. oper.
  0100            +4
 +1001        +   −7
─────         ─────
 1101 = −3        −3

       correct result

        Example (b)
```

```
             corresp.
             dec. oper.
  0011            +3
 +0100        +   +4
─────         ─────
 0111 = +7        +7

       correct result

        Example (c)
```

```
1 1 1        corresp.
             dec. oper.
  1110            −2
 +1010        +   −6
─────         ─────
11000 = −8        −8

       correct result

        Example (d)
```

```
1            corresp.
             dec. oper.
  1101            −3
 +1010        +   −6
─────         ─────
10111 = +7        −9

      incorrect result

        Example (e)
```

```
1            corresp.
             dec. oper.
  0101            +5
 +0110        +   +6
─────         ─────
 1011 = −5       +11

      incorrect result

        Example (f)
```

</div>