CSE-413 Computer Architecture
Lecture 4

Signed and Unsigned Number

Introduction

- Numbers are kept in computer hardware as a series of high and low electronic signals, and so they are considered base 2 numbers.

Just as base 10 numbers are called *decimal numbers*, *base 2 numbers are called binary numbers.*

- *A* single digit of a binary number is thus the "atom" of computing, since all information is composed of binary digits or *bits.*

- This fundamental building block can be one of two values, which can be thought of as several alternatives: high or low, on or off, true or false, or 1 or 0.

Cont.

Generalizing the point, in any number base, the value of *ith digit d is*

$$d \times \text{Base}^i$$

where *i* starts at 0 and increases from right to left. We subscript decimal numbers with *ten and binary numbers with two*

1011_{two}

represents

$$\begin{aligned}
&(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)_{ten} \\
&= (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1)_{ten} \\
&= 8 + 0 + 2 + 1_{ten} \\
&= 11_{ten}
\end{aligned}$$

Cont.

We number the bits 0, 1, 2, 3, . . . from right to left in a word. The drawing below shows the numbering of bits within a MIPS word and the placement of the number 1011_{two}

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

(32 bits wide)

- Since words are drawn vertically as well as horizontally, leftmost and rightmost may be unclear.

- Hence, the phrase least significant bit is used to refer to the rightmost bit (bit 0 above) and most significant bit to the leftmost bit (bit 31).

Cont.

- The MIPS word is 32 bits long, so we can represent 2^{32} different 32-bit patterns.
- It is natural to let these combinations represent the numbers from 0 to $2^{32} - 1$ ($4{,}294{,}967{,}295_{ten}$):

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} - 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} - 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} - 2_{ten}$
$\ldots \qquad \ldots$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} - 4{,}294{,}967{,}293_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} - 4{,}294{,}967{,}294_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} - 4{,}294{,}967{,}295_{ten}$

Cont.

That is, 32-bit binary numbers can be represented in terms of the bit value times a power of 2 (here *xi means the ith bit of x):*

$$(x31 \times 2^{31}) + (x30 \times 2^{30}) + (x29 \times 2^{29}) + \ldots + (x1 \times 2^{1}) + (x0 \times 2^{0})$$

- Hardware can be designed to add, subtract, multiply, and divide these binary bit patterns.

- If the number that is the proper result of such operations cannot be represented by these rightmost hardware bits, *overflow is said to have occurred.*

Cont.

Computer programs calculate both positive and negative numbers, so we need a representation that distinguishes the positive from the negative.

The convention for representing signed binary numbers is called two's complement representation:

Cont.

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$
... ...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2,147,483,645_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2,147,483,646_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2,147,483,647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2,147,483,648_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -2,147,483,647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -2,147,483,646_{ten}$
... ...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = -3_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1_{ten}$

Two's complement does have one negative number, $-2,147,483,648_{ten}$, that has no corresponding positive number.

Question

Explain with example why sign and magnitude form is rarely used for computer arithmetic?

Signed Number

- Two's complement representation has the advantage that all negative numbers have a 1 in the most significant bit.
- Consequently, hardware needs to test only this bit to see if a number is positive or negative (with the number 0 considered positive).
- This bit is often called the *sign bit. By recognizing the role of the sign bit, we* can represent positive and negative 32-bit numbers in terms of the bit value times a power of 2:

$$(x31 \times -2^{31}) + (x30 \times 2^{30}) + (x29 \times 2^{29}) + \ldots + (x1 \times 2^{1}) + (x0 \times 2^{0})$$

- The sign bit is multiplied by -2^{31}, and the rest of the bits are then multiplied by positive versions of their respective base values.

Example.

What is the decimal value of this 32-bit two's complement number?

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$$

Solution

Substituting the number's bit values into the formula above:

$$\begin{aligned}
&(1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \ldots + (1 \times 2^{2}) + (0 \times 2^{1}) + (0 \times 2^{0}) \\
&= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^{2} + 0 + 0 \\
&= -2{,}147{,}483{,}648_{ten} + 2{,}147{,}483{,}644_{ten} \\
&= -4_{ten}
\end{aligned}$$

Negation Shortcut

Negate 2_{ten}, and then check the result by negating -2_{ten}.

$$2_{ten} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two}$$

Negating this number by inverting the bits and adding one,

$$\begin{array}{lr}
 & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} \\
+ & 1_{two} \\
\hline
= & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} \\
= & -2_{ten}
\end{array}$$

Negation Shortcut-Cont.

Going the other direction,

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two}$$

is first inverted and then incremented:

$$
\begin{array}{rr}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} \\
+ & 1_{two} \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} \\
= & 2_{ten}
\end{array}
$$

Sign Extension Shortcut

- how to convert a binary number represented in *n bits* to a number represented with more than *n bits*.

- The shortcut is to take the most significant bit from the smaller quantity—the sign bit—and replicate it to fill the new bits of the larger quantity.

- The old bits are simply copied into the right portion of the new word. This shortcut is commonly called *sign extension*.

Example

Convert 16-bit binary versions of 2_{ten} and -2_{ten} to 32-bit binary numbers.

Solution

The 16-bit binary version of the number 2 is

$$0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$$

It is converted to a 32-bit number by making 16 copies of the value in the most significant bit (0) and placing that in the left-hand half of the word. The right half gets the old value:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$$

Cont.

Let's negate the 16-bit version of 2 using the earlier shortcut. Thus,

$0000\ 0000\ 0000\ 0010_{two}$

becomes

$$
\begin{array}{l}
\ \ 1111\ 1111\ 1111\ 1101_{two} \\
+ \qquad\qquad\qquad\qquad\quad 1_{two} \\
\hline
\end{array}
$$

$- \ 1111\ 1111\ 1111\ 1110_{two}$

Creating a 32-bit version of the negative number means copying the sign bit 16 times and placing it on the left:

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} - -2_{ten}$