

Modular arithmetic and 2's complement representation

Most computers choose a particular word length (measured in bits) for representing integers and provide hardware that performs various arithmetic operations on word-size operands. The current generation of processors have word lengths of 32 bits; restricting the size of the operands and the result to a single word means that the arithmetic operations are actually performing arithmetic modulo 2^{32} .

Almost all computers use a 2's complement representation for integers since the 2's complement addition operation is the same for both positive and negative numbers. In 2's complement notation, one negates a number by forming the 1's complement (i.e., for each bit, changing a 0 to 1 and vice versa) representation of the number and then adding 1. By convention, we write 2's complement integers with the most-significant bit (MSB) on the left and the least-significant bit (LSB) on the right. Also by convention, if the MSB is 1, the number is negative; otherwise it's non-negative.

A. How many different values can be encoded in a 32-bit word?

Each bit can be either "0" or "1", so there are 2^{32} possible values which can be encoded in a 32-bit word.

B. Please use a 32-bit 2's complement representation to answer the following questions. What are the representations for

zero

the most positive integer that can be represented

the most negative integer that can be represented

What are the decimal values for the most positive and most negative integers?

zero = 0000 0000 0000 0000 0000 0000 0000 0000

most positive integer = 0111 1111 1111 1111 1111 1111 1111 1111 = $2^{31}-1$

most negative integer = 1000 0000 0000 0000 0000 0000 0000 0000 = -2^{31}

C. Since writing a string of 32 bits gets tedious, it's often convenient to use hexadecimal notation where a single digit in the range 0-9 or A-F is used to represent groups of 4 bits using the following encoding:

| hex | bits | hex | bits | hex | bits | hex | bits |
|-----|------|-----|------|-----|------|-----|------|
| 0 | 0000 | 4 | 0100 | 8 | 1000 | C | 1100 |
| 1 | 0001 | 5 | 0101 | 9 | 1001 | D | 1101 |
| 2 | 0010 | 6 | 0110 | A | 1010 | E | 1110 |
| 3 | 0011 | 7 | 0111 | B | 1011 | F | 1111 |

Give the 8-digit hexadecimal equivalent of the following decimal and binary numbers: 37_{10} , -32768_{10} , $1101111010101101101111011101111_2$.

$37 = 00000025_{16}$

$-32768 = \text{FFFF}8000_{16}$

$1101\ 1110\ 1010\ 1101\ 1011\ 1110\ 1110\ 1111_2 = \text{DEADBEEF}_{16}$

D. Calculate the following using 6-bit 2's complement arithmetic (which is just a fancy way of saying to do ordinary addition in base 2 keeping only 6 bits of your answer). Show your work using binary (base 2) notation. Remember that subtraction can be performed by negating the second operand and then adding it to the first operand.

```

13 + 10
15 - 18
27 - 6
-6 - 15
21 + (-21)
31 + 12

```

Explain what happened in the last addition and in what sense your answer is "right".

```

13 = 001101    15 = 001111    27 = 011011
+ 10 = 001010  -18 = 101110    - 6 = 111010
=====
23 = 010111    -3 = 111101    21 = 010101

-6 = 111010    21 = 010101    31 = 011111
-15 = 110001   -21 = 101011   +12 = 001100
=====
-21 = 101011    0 = 000000    -21 = 101011 (!)

```

In the last addition, $31 + 12 = 43$ exceeds the maximum representable positive integer in 6-bit two's complement arithmetic (max int = $2^5 - 1 = 31$). The addition caused the most significant bit to become 1, resulting in an "overflow" where the sign of the result differs from the signs of the operands.

- E. At first blush "Complement and add 1" doesn't seem to an obvious way to negate a two's complement number. By manipulating the expression $A + (-A) = 0$, show that "complement and add 1" does produce the correct representation for the negative of a two's complement number. Hint: express 0 as $(-1 + 1)$ and rearrange terms to get $-A$ on one side and $XXX + 1$ on the other and then think about how the expression XXX is related to A using only logical operations (AND, OR, NOT).

Start by expressing zero as $(-1 + 1)$:

$$A + (-A) = -1 + 1$$

Rearranging terms we get:

$$(-A) = (-1 - A) + 1$$

The two's complement representation for -1 is all ones, so looking at $(-1 - A)$ bit-by-bit we see:

```

  1   ...  1   1
- AN-1 ... A1 A0
=====
~AN-1 ... ~A1 ~A0

```

where \sim is the bit-wise complement operator. We've used regular subtraction rules ($1 - 0 = 1$, $1 - 1 = 0$) and noticed that $1 - A_i = \sim A_i$. So, finally:

$$(-A) = \sim A + 1$$