Representing Data: Integers
Positive and Negative Integers

Representing Data

Problem Statement: Digital computers can only represent 0's and 1's but we need many different types of data to be represented:

- Numbers – integers, fractions, irrationals like $\pi, \sqrt{2}$
- Text (characters, strings)
- Logical – true, false
- Images (videos)
- Sound
- Machine instructions
- …

Binary Numbers

Decimal Number System (base 10)

weight: 10^3 10^2 10^1 10^0

3,108

digit (0, 1, 2, …9)

value: $3*10^3 + 1*10^2 + 0*10^1 + 8*10^0 = 3{,}108$

Binary Number System (base 2)

weight: 2^3 2^2 2^1 2^0

0101

binary digit, bit (0 or 1)

value: $0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 4 + 1 = 5$

n-bit Values

An n-bit unsigned integer can represent 2^n different values: from 0 to 2^n-1

2^2	2^1	2^0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Examples

- Express each of the following binary numbers in decimal. Note that we often write bits in groups of 4 for convenience.
  - 1011
  - 1101 1001
- Express each of the following decimal numbers in binary. Also try to think of a general algorithm for conversion from decimal to binary!
  - 12
  - 53

Examples

- Express each of the following binary numbers in decimal. Note that we often write bits in groups of 4 for convenience.
  - $(1011)_2 = 2^3 + 2^1 + 2^0 = 8 + 2 + 1 = 11$
  - $(1101\ 1001)_2 = 2^7 + 2^6 + 2^4 + 2^3 + 2^0 = 128 + 64 + 16 + 8 + 1 = 217$
- Express each of the following decimal numbers in binary. Also try to think of a general algorithm for conversion from decimal to binary!
  - $12 = 8 + 4 = 2^3 + 2^2 = 1100$
  - $53 = 32 + 21 = 32 + 16 + 5 = 32 + 16 + 4 + 1 = 2^5 + 2^4 + 2^2 + 2^0 = 11\ 0101$
- For a general algorithm, we can use something like long division.

Converting from decimal to binary

- To convert **from decimal to binary**, successively divide by 2 and track remainders.

$$53 = 2 * 26 + 1$$
$$26 = 2 * 13 + 0$$
$$13 = 2 * 6 + 1$$
$$6 = 2 * 3 + 0$$
$$3 = 2 * 1 + 1$$
$$1 = 2 * 0 + 1$$

- Once the multiplier is 0, stop. For the binary representation: arrange the remainders in reverse order: $(53)_{10} = (11\ 0101)_2$.
- We use reverse order because we essentially find the bits from smallest to largest. (E.g., the first bit is the digit corresponding to 2^0, simply indicating whether the integer is even or odd, and this shows up in the last digit of a binary or decimal representation).
- Note that this process is essentially nested multiplication!

$$53 = \mathbf{1} + 2 * \left(\mathbf{0} + 2 * \left(\mathbf{1} + 2 * \left(\mathbf{0} + 2 * \left(\mathbf{1} + 2 * (\mathbf{1} + 2 * 0)\right)\right)\right)\right)$$

Powers of 2

If you don’t already have some powers of 2 memorized, you will probably find it useful to know powers of 2 up to at least 2^{10}.

Power	Value
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024

Negative Numbers

- **Sign magnitude** representation
  – Sign bit is 0 for positive numbers, 1 for negative

sign bit

0 0 0 1 1 1 0 1 = 29

1 0 0 1 1 1 0 1 = -29

- **One's complement** representation
  – Invert (complement) each bit of a positive number to get the negative representation of that number

0 0 0 1 1 1 0 1 = 29

1 1 1 0 0 0 1 0 = -29

- What are some problems with sign-magnitude and one's complement?
  – Two representations for zero (0000, 1000 for sign magnitude; 0000, 1111 for one's compl)
  – Arithmetic (addition/subtraction) not straightforward when negative numbers involved

Two's Complement

Used to represent integers in virtually all computers today

Several equivalent definitions

Given an n-bit number X

- The n-bit two's complement representation of -X is $2^n - X$

n=8

```
  1 0 0 0 0 0 0 0 0  = 256
-   0 0 0 1 1 1 0 1  = 29
---------------------
    1 1 1 0 0 0 1 1  = -29
```

Most significant bit indicates sign (0=pos.; 1=neg.)

- The two's complement of -X is the (one's complement of -X) + 1

```
                    0 0 0 1 1 1 0 1  = 29
One's complement:   1 1 1 0 0 0 1 0
                                + 1
                    ---------------
                    1 1 1 0 0 0 1 1
```

Why does this work?

X + two's comp(-X) = 2^n

X + one's comp(-X) = 111…1 (n 1's) =

$2^0 + 2^1 + \ldots + 2^{n-1} = 2^n - 1 =$ X + two's comp(-X) – 1

=> one's comp(-X) + 1 = two's comp(-X)

Two's Complement (cont.)

Another definition:

- In the two's complement representation, the sign bit has negative weight

n=8 $\quad 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1 = -29$

$$1*(-2^7) + 1*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0$$
$$= -128 + 64 + 32 + 0 + 0 + 0 + 2 + 1$$
$$= -29$$

Fast way to generate two's complement numbers

- Starting at least significant bit (rightmost), scan right to left
- Copy bits up to and including the first '1' bit
- Complement (invert) each of the remaining bits

0 1 0 1 1 | 1 0 0 = 92

1 0 1 0 0 | 1 0 0 = -92

complement | copy

$-2^7 + 2^5 + 2^2 = -128 + 32 + 4 = -92$

Examples

- Express the following integers in binary using 8 bits.
  - 38
  - -38, using sign magnitude representation
  - -38, using one’s complement representation
  - -38, using two’s complement representation
- Can you think of ways to verify your results?

Examples

- Express the following integers in binary using 8 bits.
  - 38 = 2*19 + 0
    19 = 2*9 + 1
    9 = 2*4 + 1
    4 = 2*2 + 0
    2 = 2*1 + 0
    1 = 2*0 + 1
    Thus, 38 = 0010 0110.
  - -38, using sign magnitude representation: 1010 0110
  - -38, using one’s complement representation: 1101 1001
  - -38, using two’s complement representation: 1101 1010
- Verify: X + one’s complement (-X) = all 1’s
- Verify: X + two’s complement (-X) = 2^n (1 followed by n 0’s), but because we are limited to n bits, we get 0 (disregard leading 1)
  Also can calculate by computing sum using negative weight for first bit:
  -128+64+16+8+2=-38

Summary of Representing Integers

- Base 2 really works just like base 10… but we humans have to think about it
- We can represent 2^n integers using n bits; specific numbers depend on choice of signed or unsigned
- Most computers use two's complement to represent signed integers