

Section B: Digital Information

Representing numbers

Positional notation

Let b , called the **base**, be an integer such that $b \geq 2$, and let d_0, d_1, \dots, d_{b-1} be symbols, called **digits**, that represent the first b non-negative integers in order.

Any non-negative integer number can be written in exactly one way using these digits and **positional notation** as follows: For any

$$x_{i_s}, x_{i_s-1}, \dots, x_{i_0} \in \{d_0, d_1, \dots, d_{b-1}\},$$

the expression

$$(x_{i_s} x_{i_s-1} \dots x_{i_0})_b$$

means

$$x_{i_s} \times b^s + x_{i_s-1} \times b^{s-1} + \dots + x_{i_0} \times b^0$$

Decimal

$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is the set of **decimal digits**. These digits are used for representing numbers in base 10.

$$\begin{aligned} & (63403)_{10} \\ &= (6 \times 10^4 + 3 \times 10^3 + 4 \times 10^2 + 0 \times 10^1 + 3 \times 10^0)_{10} \end{aligned}$$

Binary

$\{0, 1\}$ is the set of **binary digits**, usually called **bits** for short. These digits are used for representing numbers in base 2.

$$\begin{aligned}(10111)_2 &= (1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)_{10} \\ &= (16 + 0 + 4 + 2 + 1)_{10} \\ &= (23)_{10}\end{aligned}$$

Octal

$\{0, 1, 2, 3, 4, 5, 6, 7\}$ is the set of **octal digits**. These digits are used for representing numbers in base 8.

$$\begin{aligned}(63403)_8 &= (6 \times 8^4 + 3 \times 8^3 + 4 \times 8^2 + 0 \times 8^1 + 3 \times 8^0)_{10} \\ &= (24576 + 1536 + 256 + 0 + 3)_{10} \\ &= (26371)_{10}\end{aligned}$$

Hexadecimal

$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ is the set of **hexadecimal digits**. These digits are used for representing numbers in base 16.

$$\begin{aligned}(3AD0F)_{16} &= (3 \times 16^4 + 10 \times 16^3 + 13 \times 16^2 + 0 \times 16^1 + 15 \times 16^0)_{10} \\ &= (196608 + 40960 + 3328 + 0 + 15)_{10} \\ &= (240911)_{10}\end{aligned}$$

Notational conventions

We often omit the parentheses, or omit both the parentheses and the subscript indicating the base when the omission is unlikely to cause misunderstanding.

Example: We often write 11010100_2 instead of $(11010100)_2$. In the middle of a page of computations in binary, we might write simply write 11010100 .

In the slides above you also saw that we sometimes group sums and products inside a single set of parentheses to indicate the base in which all the numbers are represented.

Converting from decimal to binary

Suppose that $n \in \mathbb{N}$ is represented in decimal. To write n in binary: write n as a sum of powers of 2. Start by finding the largest power of 2 that does not exceed n .

$$\begin{aligned}(71)_{10} &= (64 + 7)_{10} \\&= (64 + 4 + 3)_{10} \\&= (64 + 4 + 2 + 1)_{10} \\&= (2^6 + 2^2 + 2^1 + 2^0)_{10} \\&= (1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 \\&\quad + 1 \times 2^1 + 1 \times 2^0)_{10} \\&= (1000111)_2\end{aligned}$$

An example

$$\begin{aligned}(347)_{10} &= (256 + 91)_{10} \\&= (256 + 64)_{10} \\&= (256 + 64 + 27)_{10} \\&= (256 + 64 + 16 + 11)_{10} \\&= (256 + 64 + 16 + 8 + 3)_{10} \\&= (256 + 64 + 16 + 8 + 2 + 1)_{10} \\&= (2^8 + 2^6 + 2^4 + 2^3 + 2^1 + 2^0)_{10} \\&= (1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 \\&\quad + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)_{10} \\&= (101011011)_2\end{aligned}$$

Converting between binary and octal

Suppose that $n \in \mathbb{N}$ is represented in binary. To write n in octal: collect the bits into groups of 3, then replace each group of 3 bits by the appropriate octal digit.

$$\begin{aligned}(1011110)_2 &= (\underbrace{1}_{001} \underbrace{011}_{011} \underbrace{110}_{110})_2 \\ &= (\underbrace{001}_{001} \underbrace{011}_{011} \underbrace{110}_{110})_2 \\ &= (136)_8\end{aligned}$$

Reverse the process to go from octal to binary.

Converting between binary and hexadecimal

Suppose that $n \in \mathbb{N}$ is represented in binary. To write n in hexadecimal: collect the bits into groups of 4, then replace each group of 4 bits by the appropriate hexadecimal digit.

$$\begin{aligned}(1011110)_2 &= (\underbrace{1011}_{13} \underbrace{110}_{6})_2 \\ &= (\underbrace{0101}_{5} \underbrace{1110}_{14})_2 \\ &= (5E)_{16}\end{aligned}$$

Reverse the process to go from hexadecimal to binary.

Why binary?

Many real world phenomena exist in one of two states: current flowing or no current flowing; a north pole or south pole of a magnet; at each location on a laser disc, we either burned a tiny hole or we did not burn a tiny hole. By interpreting one of the states as 0 and the other as 1, we can represent information. We simply need to agree upon how to encode information in the form of 0's and 1's. It is therefore natural to represent numbers in binary, rather than encode decimal digits as strings of 0's and 1's and then try to make numerical meaning from them.

Why octal and hexadecimal?

A page of binary is difficult for humans to read.

The same information is written in a more compact form on a page, and is more easily read by humans, in octal or hexadecimal.

Conversions between binary, octal and hexadecimal are convenient.

Some vocabulary

A digit in base 2 is called a **bit**. This is simply a contraction of **binary digit**. It was first used in information theory by Tukey in 1948.

A block of 8 bits is called a **byte**. It was first used by IBM around the 60's when they started using an 8-bit character code known as EBCDIC.

A block of 4 bits is called a **nibble**.

A sequence of several adjacent bytes is called a **word**. The number of bytes varies, depending on the purpose of the word. For example, a 2-byte word can store non-negative integers in the range from 0 to $2^{16} - 1 = 65535$.

Addition in decimal

In base 10:

$$\begin{array}{r} 123 + 678 : \\ \begin{array}{r} 1 \\ + 6 \\ \hline 1 \end{array} \begin{array}{l} \text{carry digits} \end{array} \\ \hline 8 \end{array}$$

Addition in binary

In base 2:

$$\begin{array}{r} 111_2 + 10_2 : \\ \begin{array}{r} 1 1 \\ + 1 \\ \hline 1 1 \end{array} \quad \begin{array}{l} \text{carry bits} \\ \hline 1 0 1 \end{array} \end{array}$$

Addition in hexadecimal

In base 16:

$$456_{16} + ABC_{16} :$$

	4	5	6	
+	<i>A</i>	<i>B</i>	<i>C</i>	
	1	1		carry bits
	<i>F</i>	1	2	

Representing positives, negative and zero

Representing positives, negative and zero

When writing positive and negative integers in binary, we may use a - or + in front of the integer to represent its sign.

We may write $+(1101)_2$ for $(13)_{10}$
and $-(1101)_2$ for $-(13)_{10}$.

How can we, in a computer, represent integers in a range that includes negative integers?

Naive approach

How can we, in a computer, represent integers in to a range that includes negative integers?

Naive approach

How can we, in a computer, represent integers in to a range that includes negative integers?

FIRST IDEA: Use an extra **sign bit**. Fix a number of bits to use for each integer, and the left-most bit is 0 if the number is positive, and 1 if the number is negative.

In such a scheme with say, a byte to represent an integer:

$(00001101)_2$ would represent $(13)_{10}$

$(10001101)_2$ would represent $(-13)_{10}$

and $(00000000)_2$ and $(10000000)_2$ would both represent zero.

Naive approach

How can we, in a computer, represent integers in to a range that includes negative integers?

FIRST IDEA: Use an extra **sign bit**. Fix a number of bits to use for each integer, and the left-most bit is 0 if the number is positive, and 1 if the number is negative.

In such a scheme with say, a byte to represent an integer:

$(00001101)_2$ would represent $(13)_{10}$

$(10001101)_2$ would represent $(-13)_{10}$

and $(00000000)_2$ and $(10000000)_2$ would both represent zero.

This is **NOT** how integers are usually represented.

A better idea

How can we, in a computer, represent integers in to a range that includes negative integers?

BETTER IDEA: Use an extra **sign bit**, but the binary representation of a negative is chosen according to a scheme called two's complement...

Toggle plus one

We fix a number of bits, say t , to use for each integer. A string of t bits $d_1d_2 \dots d_t$ is interpreted as a number in one of two ways, depending on the value of d_1 :

- if $d_1 = 0$, then the bit string $(0d_2d_3 \dots d_t)$ represents $(d_2d_3 \dots d_t)_2$
- if $d_1 = 1$, then the bit string $(1d_2d_3 \dots d_t)$ represents $(d_2d_3 \dots d_t)_2 - 2^{t-1}$, which is equivalent to $-[(e_2e_3 \dots e_t)_2 + 001_2]$ where

$$e_i = \begin{cases} 1, & \text{if } d_i = 0 \\ 0, & \text{if } d_i = 1. \end{cases}$$

The process of evaluating $(1d_2d_3 \dots d_t)_2$ is often described as: ‘toggle’ all bits, add one, then negate.

Example: 4-bit signed integers

When we use nibbles to represent integers, then $t = 4$ and we have:

nibble	decimal value	nibble	decimal value
0000	0	1000	-8
0001	1	1001	-7
0010	2	1010	-6
0011	3	1011	-5
0100	4	1100	-4
0101	5	1101	-3
0110	6	1110	-2
0111	7	1111	-1

In this case, we say we are using **4-bit signed integers**.

Example

Suppose we represent integers using 4-bit signed integers. Which integer is represented by 1101?

Example

Suppose we represent integers using 4-bit signed integers. Which integer is represented by 1101?

Since the first bit in the bit-string is one, 1101 represents a negative integer and that integer is

$$(101)_2 - 2^3 = 5 - 8 = -3$$

OR

Since the first bit in the bit-string is one, 1101 represents a negative integer and that integer is

- drop the first bit to get 101
- toggle all the bits to get 010
- add one to get 011.
- Convert to decimal to get 3
- Multiply by -1 to get that 1101 represents $(-3)_{10}$.

Example: 8-bit signed integers

When we use bytes to represent integers, then $t = 8$ and we have:

nibble	decimal value	nibble	decimal value
00000000	0	10000000	-128
00000001	1	10000001	-127
00000010	2	10000010	-126
⋮	⋮	⋮	⋮
01111110	126	11111110	-2
01111111	127	11111111	-1

In this case, we say we are using **8-bit signed integers**.

Example

Suppose we represent integers using 8-bit signed integers. Which integer is represented by 10110101?

Example

Suppose we represent integers using 8-bit signed integers. Which integer is represented by 10110101?

Since the first bit in the bit-string is one, 10110101 represents a negative integer and that integer is

$$(0110101)_2 - 2^7 = ((1 + 4 + 16 + 32) - 128)_{10} = (53 - 128)_{10} = (-75)_{10}.$$

OR

Since the first bit in the bit-string is one, 10110101 represents a negative integer and that integer is

- drop the bit to get 0110101
- toggle all the bits to get 1001010
- add one to get 1001011.
- Convert to decimal to get $1 + 2 + 8 + 64 = 75$
- Multiply by -1 to get that 1101 represents $(-75)_{10}$.

Comparison

Using the Naive approach with 8 bits:

- we can represent all integers in

$$\{-127, -126, -125, \dots, 127\}$$

- zero is represented by 00000000 and 100000000
- Designing processes for addition and subtraction is a hassle

Comparison (cont.)

Using the two's complement approach with 8 bits:

- we can represent all integers in

$$\{-128, -127, -126, \dots, 127\}$$

- zero is represented by only by 00000000
- Our usual addition process for positive numbers in binary works no matter whether the numbers are positive or negative.
- Converting 5, say, to -5 is an easy process: *toggle all bits then add one.*
- To accomplish $x - y$: Find the representation of $-y$ from the representation for y ; then add the representations for x and $-y$.