

## Decimal & Binary Representation Systems

---

Decimal & binary are positional representation systems

- each position has a value:  $d \cdot \text{base}^i$
- for example:  $321_{10} = 3 \cdot 10^2 + 2 \cdot 10^1 + 1 \cdot 10^0$
- for example:  $10100001_2 = 1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

The **general formula** for a positive number in:

- decimal:  $\sum_{i=0}^n a_i \times 10^{n-i}$ , where the  $a_i$  are between 0 & 9
- binary:  $\sum_{i=0}^m b_i \times 2^{m-i}$ , where the  $b_i$  are 0 or 1

## Decimal & Binary Representation Systems

---

Converting **binary** → **decimal**:

- add the factors
- $10100001_2 =$
- $1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 256 + 0 + 64 + 0 + 0 + 0 + 0 + 0 + 1 = 321$

Converting **decimal** → **binary**:

- decompose the decimal number into powers of 2
- $321 = 256 + 64 + 1 = 1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 10100001_2$

## Hexadecimal Representation System

---

The hexadecimal numbers:

- 0-9, a, b, c, d, e, f
- binary values 0000 to 1111
- easier to use than binary numbers (1 digit represents several binary values)
- quick conversion to binary numbers

The general formula for a hexadecimal number is:

- $\sum_{i=0}^n a_i \times 16^{n-i}$ , where the  $a_i$  are between 0 & f
- for example:  $141_{16} = 1 \times 16^2 + 4 \times 16^1 + 1 \times 16^0 = 321_{10}$

Converting binary  $\rightarrow$  hexadecimal:

- group into 4-bit numbers:  $101001011_2 = 1\ 0100\ 1011_2$
- translate each group into a hexadecimal digit:  
 $1\ 0100\ 1011_2 = 14B_{16}$

Converting hexadecimal  $\rightarrow$  binary

- expand each hex digit to a sequence of binary digits

## Useful Powers of 2

---

$$2^{10} = 1024_{10} \approx 10^3 = 1 \text{ K}$$

$$2^{20} \approx 10^6 = 1 \text{ M}$$

$$2^{30} \approx 10^9 = 1 \text{ G}$$

Used particularly in storage sizes:

- 16KB cache
- 64MB memory
- 4GB disk

## Representing Positive & Negative Numbers

---

Can represent  $2^n$  different values in  $n$  bits

For **unsigned integers**, the values are  $0..2^{32}-1$

Need a representation for **signed integers** with the following properties:

- an equal number of positive & negative numbers
- a unique representation for 0
- an easy hardware test for 0
- an easy hardware test for the sign
- easy hardware rules for addition/subtraction

## Two's Complement

---

Some definitions:

- **least significant bit (lsb)**: the least magnitude bit (or digit), the one at the *rightmost* position of the representation
- **most significant bit (msb)**: the greatest magnitude bit (or digit), the one at the *leftmost* position of the representation

Representation for signed integers

- 0 is a series of zeros
- positive numbers: msb = 0
- negative numbers: msb = 1

To represent a negative number:

- start with the representation for its positive value
- flip all the bits (1's to 0; 0's to 1)
- add 1 to the lsb using binary arithmetic

## Two's Complement

---

Examples with a 4-bit binary number:

- What is the representation for  $6_{10}$ ?
- What is the representation for  $-6_{10}$ ?
- What is the representation of 0?
- What is the range of positive numbers?
- What is the range of negative numbers?
- How do you represent  $6_{10}$  in an 8-bit binary number?
- How do you represent  $-6_{10}$  in an 8-bit binary number?
- How does the hardware recognize whether a number is positive or negative?
- How does the hardware recognize whether a number is zero?

## Addition/Subtraction in Two's Complement

---

### Addition

- do not treat the sign bit specially; perform an addition on all bits
- if add 2 numbers of opposite signs, this will work fine
- if add 2 positive numbers & result “appears” to be negative (msb = 1)
  - → **overflow** (value won't fit in “word size” number of bits)
  - generates an **exception** (unscheduled procedure call to the operating system) in the program (*we'll discuss exceptions at the end of the quarter*)
- if add 2 negative numbers & result “appears” to be positive (msb = 1)
  - → **underflow**
  - generates an exception in the program

### Subtraction

- take the 2's complement of the subtrahend & add it to the other operand

## Alternative Representations

---

Historically there have been other representations for signed integers, but they are no longer used

### Signed magnitude

- separate bit for the sign
- extra step to set it
- not clear where to store it
- has both positive & negative values for zero

### One's complement

- negative number is the complement of the absolute value
- + positive & negative values are balanced
  - largest positive value:  $2,147,483,647_{10}$
  - largest negative value:  $-2,147,483,647_{10}$
- has 2 values for zero
  - positive zero:  $00\dots00$
  - negative zero:  $11\dots11$

## A Bag of Bits

---

Bit patterns have no meaning

Their meaning depends on how they are interpreted:

- signed integers
- unsigned integers
- floating point numbers
- characters
- instructions

For data, the interpretation is determined by the instruction.