

# Binary Numbers

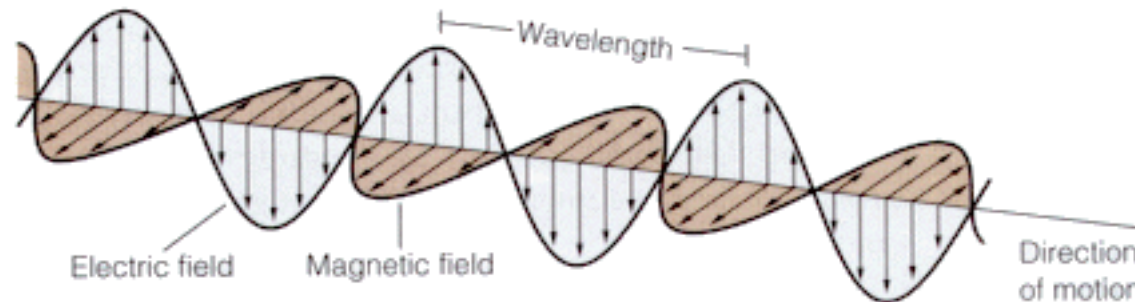
Wolfgang Schreiner  
Research Institute for Symbolic Computation (RISC-Linz)  
Johannes Kepler University

Wolfgang.Schreiner@risc.uni-linz.ac.at  
<http://www.risc.uni-linz.ac.at/people/schreine>

## Digital Computers

Today's computers are digital.

- **Digital:** data are represented by discrete pieces.
  - Pieces are denoted by the natural numbers: 0, 1, 2, 3 . . .
- **Analog:** data are represented by continuous signals.
  - For instance, electromagnetic waves.



## Character Encodings

- **ASCII**: American Standard Code for Information Interchange.
  - $128 = 2^7$  characters (letters and other symbols).
  - Also non-printable characters: LF (line-feed).
  - Represented by numbers  $0, \dots, 127$ .

ASCII code	Character
...	...
10	LineFeed (LF)
...	...
48–57	0–9
...	...
65–90	A–Z
...	...
97–122	a–z
...	...

## Character Encodings

- Text is a sequence of characters.

H i , H e a t h e r .  
72 105 44 32 72 101 97 116 104 101 114 46

- **ISO 8859-1** contains  $256 = 2^8$  characters (Latin 1).
  - First 128 characters coincide with ASCII standard.
- **Unicode** contains  $65534 = 2^{16} - 2$  characters.
  - First 256 characters coincide with ISO 8859-1 standard.

The **ASCII** characters are the same in all encodings.

## Binary Numbers

In which number system are numbers represented?

- Humans: decimal system.
  - 10 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Computers: binary system.
  - 2 digits: 0, 1.
  - A **bit** is a binary digit.
  - Physical representation: e.g. high voltage versus low voltage.

Binary numbers are physically easy to represent.

## Binary Numbers

- Decimal number 103:

$$1 * 10^2 + 0 * 10^1 + 3 * 10^0 = 1 * 100 + 0 * 10 + 3 * 1 = 103.$$

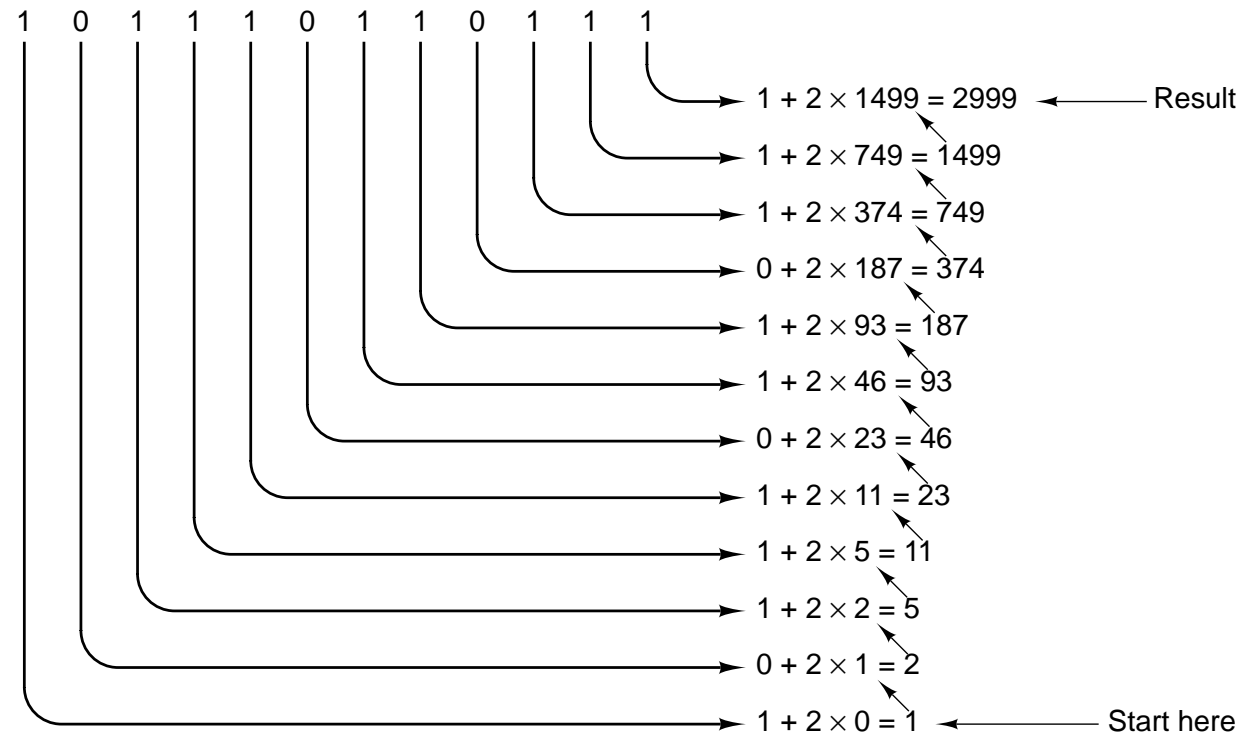
- Binary number 1100111:

$$\begin{aligned} 1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = \\ 1 * 64 + 1 * 32 + 0 * 16 + 0 * 8 + 1 * 4 + 1 * 2 + 1 * 1 = 103 \end{aligned}$$

- Character g: ASCII code 103.

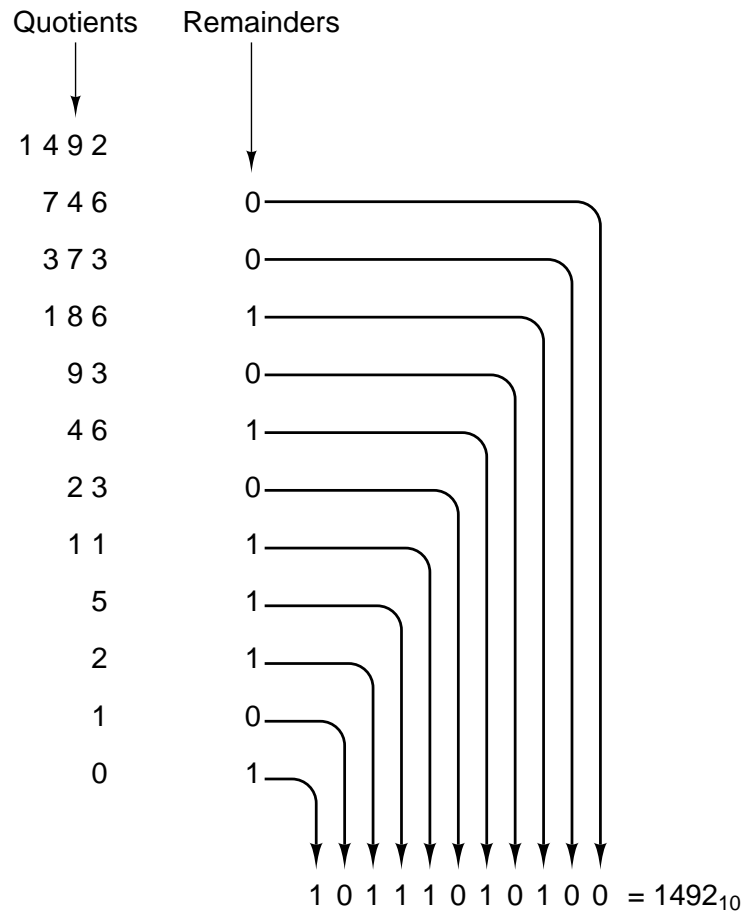
Binary number 1100111 is computer representation of g.

## Conversion of Binary to Decimal



Horner's scheme.

# Conversion of Decimal to Binary





## General Number Systems

- Any **base value (radix)  $b$**  possible.
  - Decimal system:  $b = 10$ .
  - Binary system:  $b = 2$ .
- $n$  digits  $d_{n-1} \dots d_0$  represent a number  $m$ :
$$m = d_{n-1} * b^{n-1} + \dots + d_0 * b^0 = \sum_{0 \leq i < n} d_i * b^i.$$
- Number bounds:  $0 \leq m < b^n$ .
  - Decimal system, 8 digits:  $0 \leq m < 10^8 = 100.000.000$ .
  - Binary system, 8 digits:  $0 \leq m < 2^8 = 256$ .
- Example: How many bit does it take to represent a character
  - in ASCII, in the ISO 8859-1 code, in Unicode?
  - $n > \log_b m$ .

## Other Number Systems

- **Octal** system:

- 8 digits 0, 1, 2, 3, 4, 5, 6, 7.
- One octal digit (6) can be represented by 3 bits (110).
- Conversion of binary number: 001100111:

001	100	111
1	4	7

- **Hexadecimal** system:

- 16 digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
- One hexadecimal digit (B) can be represented by 4 bits (1011).
- Conversion of binary number 01100111:

0110	0111
6	7

Easy conversion between binary and octal/hexadecimal numbers.

## Example Conversions

### Example 1

Hexadecimal

1 9 4 8 . B 6

Binary

0001 1001 0100 1000 . 1011 0110 0

Octal

1 4 5 1 0 . 5 5 4

### Example 2

Hexadecimal

7 B A 3 . B C 4

Binary

0111 1011 1010 0011 . 1011 1100 0100

Octal

7 5 6 4 3 . 5 7 0 4

Hexadecimal/octal numbers are shorter to write.

## Unsigned Binary Numbers

Unsigned integers with  $n$  bits: from 0 to  $2^n - 1$ .

Number	Unsigned Integer
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Computer representation of finite-precision integers.

## Signed Binary Numbers

Signed integers with  $n$  bits: from  $-2^{n-1}$  to  $2^{n-1} - 1$ .

Number	Unsigned Integer	Signed Integer
000	0	+0
001	1	+1
010	2	+2
011	3	+3
100	4	-4
101	5	-3
110	6	-2
111	7	-1

-4	-3	-2	-1	0	1	2	3
100	101	110	111	000	001	010	011

Two's complement representation.

## Signed Binary Numbers

Why this representation?

- Arithmetic independent of interpretation.

$$\text{Binary: } 010 + 101 = 111$$

$$\text{Unsigned: } 2 + 5 = 7$$

$$\text{Signed: } 2 - 3 = -1$$

- Computation of representation:

- Determine representation of  $-3$ :
- Representation of  $+3$ :  $011$ .
- Invert representation:  $100$ .
- Add 1:  $101$ .

Simple implementation in arithmetic hardware.

## Binary Arithmetic

Addend	0	0	1	1
Augend	+0	+1	+0	+1
Sum	0	1	1	0
Carry	0	0	0	1

- **Overflow:**

- Carry generated by addition of left-most bits is thrown away.
- Addend and augend are of same sign, result is of opposite sign.

All hardware arithmetic is finite-precision.

## Floating-Point Numbers

How to represent 1.375?

- Representation:  $(s, m, e)$ 
  - the **sign** bit  $s$  denotes +1 or -1,
  - the **mantissa**  $m$  is a  $n$  bit binary number representing the value  $m/2^n (< 1)$ ,
  - the **exponent**  $e$  is a binary number.

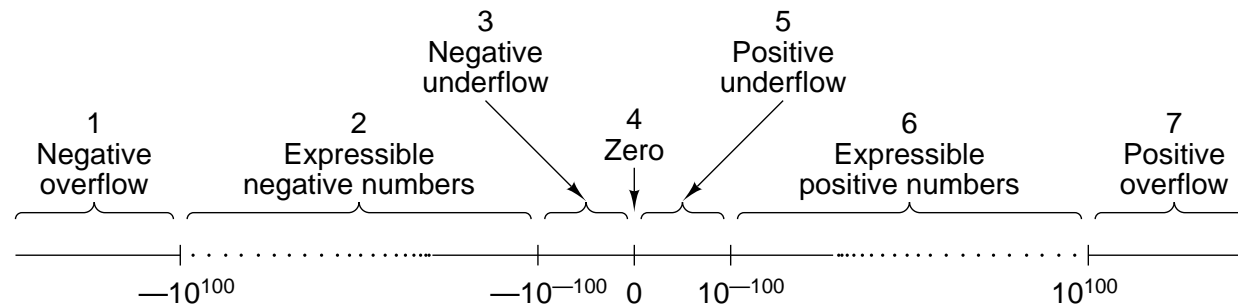
Value:  $s * m/2^n * 2^e$

- Example: Eight bit floating point: 0|01011|10
  - the first bit 0 represents the sign +1,
  - the five bit mantissa 01011 represents the fraction 11/32 (why?),
  - the two bit exponent is 2.

Value:  $+1 * 11/32 * 2^2 = 1.375$



## Reals and Floating Points



- Example: fraction with 3 decimal digits, exponent with 2 digits.
  1. Numbers between  $-0.999 * 10^{99}$  and  $-0.100 * 10^{-99}$ .
  2. Zero.
  3. Numbers between  $0.100 * 10^{99}$  and  $0.999 * 10^{99}$ .
- Real values are **rounded** to the closest floating point value.
  - **Overflows** and **underflows** may occur.

# Normalized Floating Point Numbers

Example 1: Exponentiation to the base 2

		$2^{-2}$	$2^{-4}$	$2^{-6}$	$2^{-8}$	$2^{-10}$	$2^{-12}$	$2^{-14}$	$2^{-16}$							
		↓	↓	↓	↓	↓	↓	↓	↓							
Unnormalized:	<b>0 1010100</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	
	Sign Excess 64 + exponent is 84 - 64 = 20	Fraction is $1 \times 2^{-12} + 1 \times 2^{-13} + 1 \times 2^{-15} + 1 \times 2^{-16} = 432$														

To normalize, shift the fraction left 11 bits and subtract 11 from the exponent.

Normalized:	<b>0 1001001</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
	Sign Excess 64 + exponent is 73 - 64 = 9	Fraction is $1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-4} + 1 \times 2^{-5} = 432$														

Example 2: Exponentiation to the base 16

		$16^{-1}$	$16^{-2}$	$16^{-3}$	$16^{-4}$	
		∧	∧	∧	∧	
Unnormalized:	<b>0 1000101</b>	<b>0000</b>	<b>0000</b>	<b>0001</b>	<b>1011</b>	
	Sign Excess 64 + exponent is 69 - 64 = 5	Fraction is $1 \times 16^{-3} + B \times 16^{-4} = 432$				

To normalize, shift the fraction left 2 hexadecimal digits, and subtract 2 from the exponent.

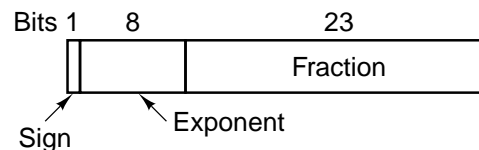
Normalized:	<b>0 1000011</b>	<b>0001</b>	<b>1011</b>	<b>0000</b>	<b>0000</b>	<b>0000</b>	<b>0000</b>	<b>0000</b>	<b>0000</b>	<b>0000</b>	<b>0000</b>	<b>0000</b>	<b>0000</b>	<b>0000</b>	<b>0000</b>	<b>0000</b>
	Sign Excess 64 + exponent is 67 - 64 = 3	Fraction is $1 \times 16^{-1} + B \times 16^{-2} = 432$														

Left-most digit of mantissa is always non-zero.

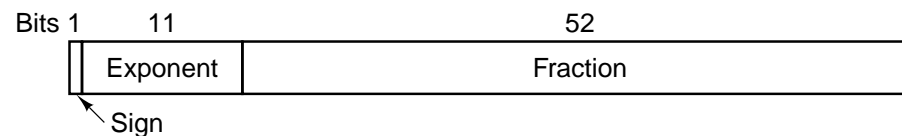
## IEEE Floating-Point Standard 754

IEEE standard for floating point representation (1985).

1. Single precision: 32 bits ( $= 1 + 8 + 23$ ).
2. Double precision: 64 bits ( $= 1 + 11 + 52$ ).
3. Extended precision: 80 bits (inside hardware units only).



(a)



(b)

## IEEE Floating Point Characteristics

Item	Single Precision	Double Precision
Smallest normalized number	$2^{-126}$	$2^{-1022}$
Largest normalized number	$2^{128}$	$2^{1024}$
Decimal range	$10^{-38}$ to $10^{38}$	$10^{-308}$ to $10^{308}$
Smallest denormalized number	$10^{-45}$	$10^{-324}$

- **Denormalized numbers:** first bit of mantissa 0.
  - Distinguished from normalized numbers by 0 exponent.
  - Used to represent very small floating point numbers.
  - Avoid underflows by giving up precision of mantissa.
  - Smallest value: 1 in the rightmost bit, rest 0.

## IEEE Numerical Types

Normalized	±	$0 < \text{Exp} < \text{Max}$	Any bit pattern
Denormalized	±	0	Any nonzero bit pattern
Zero	±	0	0
Infinity	±	1 1 1...1	0
Not a number	±	1 1 1...1	Any nonzero bit pattern

↙ Sign bit

- Two zeros (positive and negative).
- Plus and minus infinity (overflows).
- NaN (Not a Number = infinity / infinity).