What is a number?

TDDE25

Fö 2

Chap 1: Data Storage

Types of Numbers

Number Systems

Representation in Computers

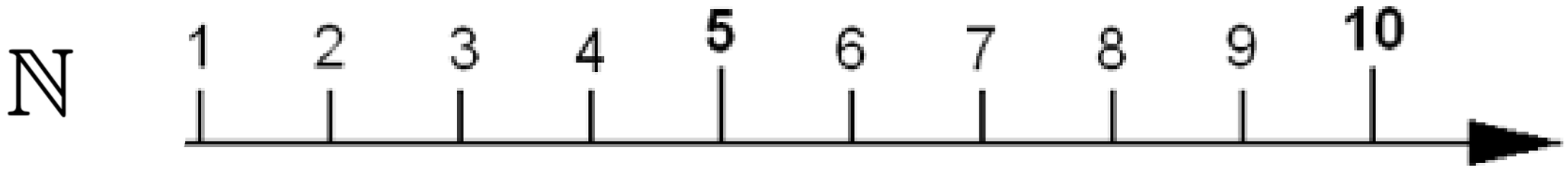
LINKÖPING UNIVERSITY

Types of Numbers

$\mathbb{N}$ $\mathbb{Z}$ $\mathbb{Q}$ $\mathbb{R}$ $\mathbb{C}$

Nine **Z**ulu **Q**ueens **R**uled **C**hina

Natural Numbers – Counting



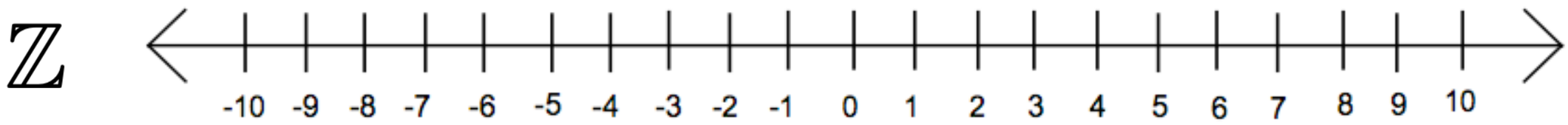
- Addition - Closed
- Multiplication - Closed
- Subtraction - **Not** Closed
- Division - **Not** Closed

Solving the Subtraction Problem:

- Discovery of Zero
- Discovery of negative numbers
  - Han Dynasty 220 BCE - 202 CE
  - Europe: 17th century

CE - Common era

Integers

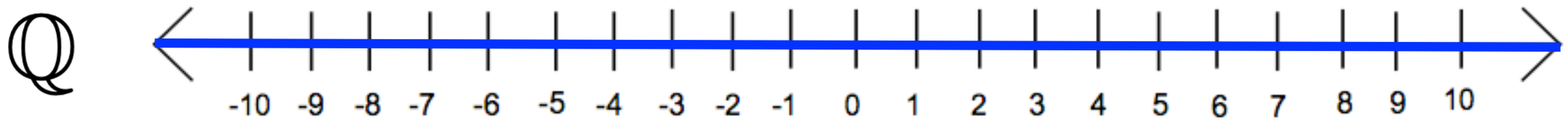


- Addition - Closed
- Multiplication - Closed
- Subtraction - Closed
- Division - **Not** Closed

Solving the Division Problem:

- Discovery of rational numbers
  - Ratios between two integers

Rational Numbers



- Addition – Closed
- Multiplication – Closed
- Subtraction – Closed
- Division – Closed

$$\frac{\text{Numerator}}{\text{Denominator}}$$

Numerator, Denominator are integers
The denominator can not be 0

Rational numbers are dense. Between any two of them, you can always find another!

- Between 0 and 1 there an infinite number of rationals!

Decimal Representation of Fractions

Rational numbers are simple quantities:

- They can be understood in finite terms;
- Yet they can be used to represent quantities as small or as large as we please

$$\frac{1}{3} = 0.33333333\ldots,$$

$$\frac{29}{12} = 2.416666666\ldots,$$

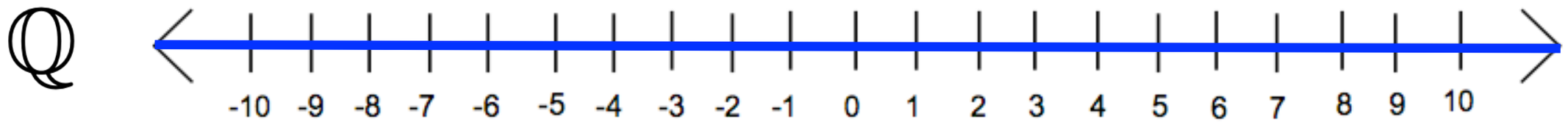
$$\frac{9}{7} = 1.\underline{285714}\underline{285714}\underline{285}\ldots,$$

$$\frac{237}{148} = 1.60135135135\ldots$$

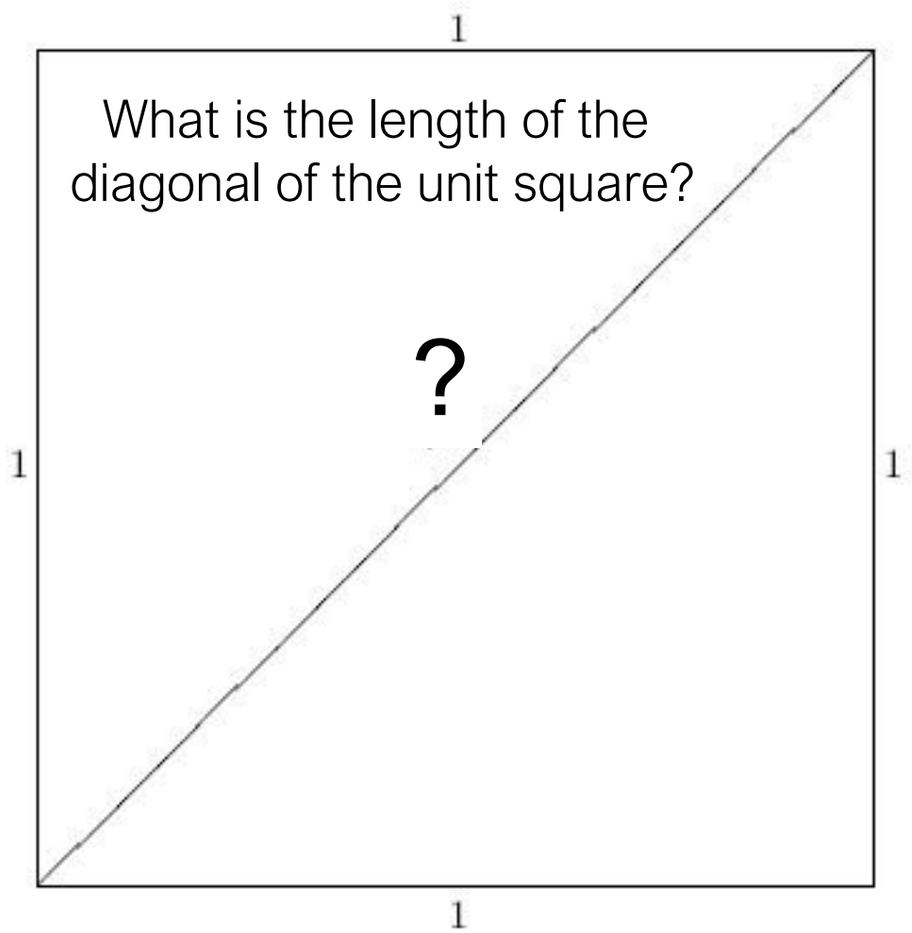
Decimal Representation

Fractions are ultimately periodic: after a certain point the infinite sequence of digits consists of some finite sequence of digits repeated indefinitely!

Are the Rational Numbers Adequate?



Since the rationals are dense, do we need any other quantity or can we even fit anything more on the number line?



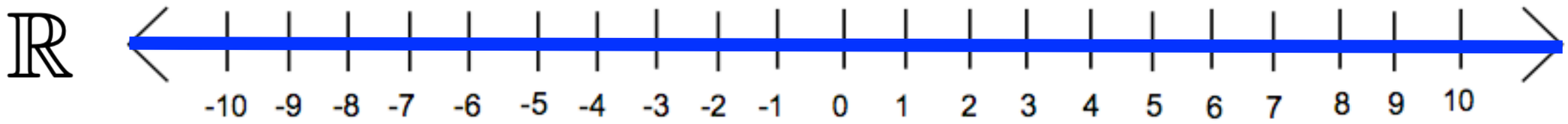
$$
\begin{aligned}
c^2 &= a^2 + b^2 \\
&= 1^2 + 1^2 \\
&= 1 + 1 = 2
\end{aligned}
$$

$$
\begin{aligned}
c^2 &= 2 \\
c &= \sqrt{2}
\end{aligned}
$$

Can **not** be expressed as the ratio of two integers!

Not irrational ➔ an irrational number!

Real Numbers



Real Numbers = Rational Numbers + Irrational Numbers

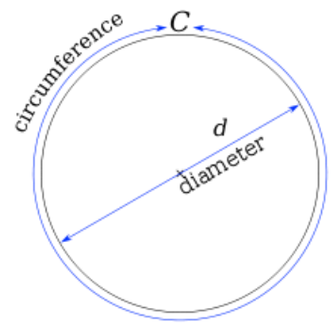
$$\sqrt{2} = 1.41421356237309504880168872\ldots$$

In decimal notation, an irrational number can **not** be represented as terminating or with eventually repeating decimals.

Some Special Real Numbers

Some important irrational numbers in engineering!

$$\pi = \frac{C}{d}$$



$$\pi = 3.14159265358979323846264338327 9\ldots$$

$$e^{\ln(x)} = x \qquad \ln(e^x) = x$$ Euler's number

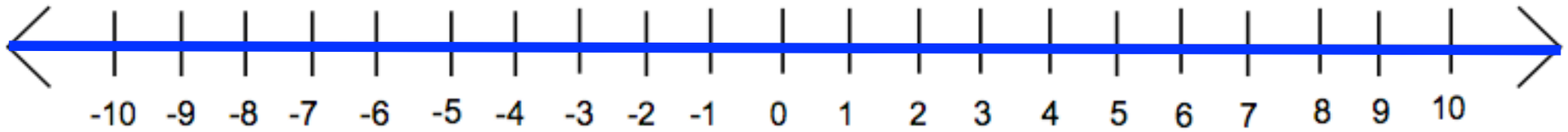
$$e = 2.71828182845904523536028747135 26\ldots$$

Since they are important numbers, we will want to represent them in a computer…

but can we? Let's get back to that later!

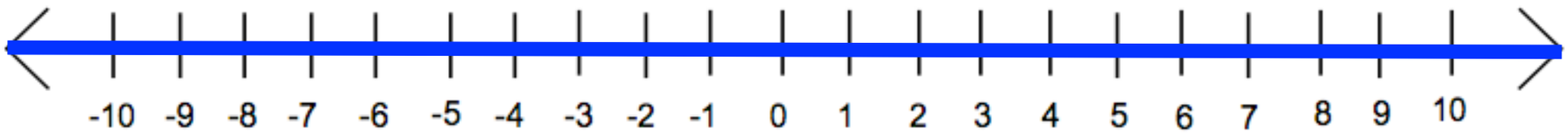
Countability

$\mathbb{Q}$



Dense

$\mathbb{R}$

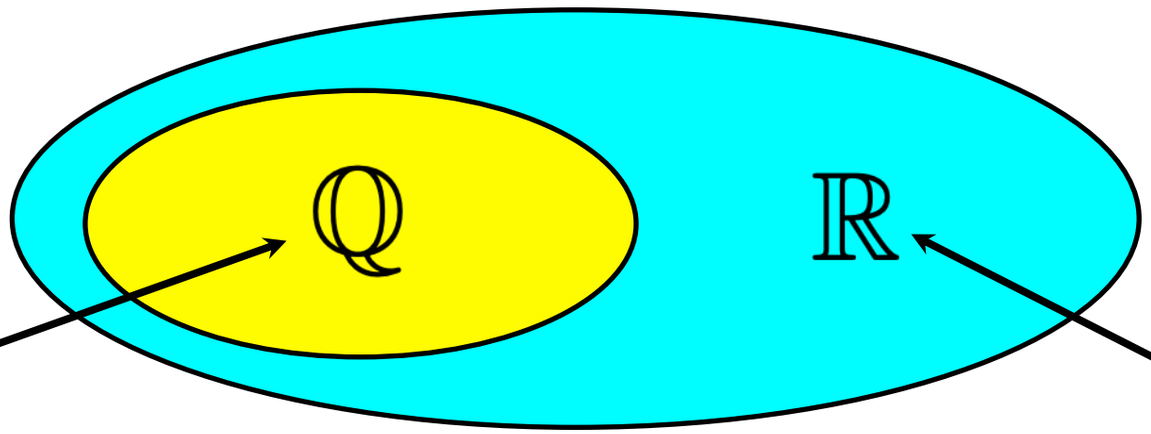


Continuous

∞

Are some infinite sets bigger than others?

$\mathbb{N} \subseteq \mathbb{Q} \subset \mathbb{R}$



Countable

Uncountable

Number Systems and Bases

LINKÖPING UNIVERSITY

Number Systems: Bases

The base of a system specifies the number of digits used

Base-10: Decimal number system: Digits 0-9

Base-2: Binary number system:	Digits: 0-1
Base-8: Octal number system:	Digits: 0-7
Base-16: Hexadecimal number system:	Digits: 0-9, Letters: A-F

Number Systems: Positional Notation

Numbers are written and manipulated using positional notation.

Radix Point ↓
10^3 10^2 10^1 10^0 • 10^{-1} 10^{-2} 10^{-3}
$10^3 = 10 \times 100$, or 1000
$10^2 = 10 \times 10$, or 100
$10^1 = 10 \times 1$, or 10
$10^0 = 1$ (*any* number raised to the power of 0 equals 1)
$10^{-1} = 1 \div 10$, or .1
$10^{-2} = 1 \div 100$, or .01
$10^{-3} = 1 \div 1000$, or .001

7 3 6

$6 \times 10^0 = 6 \times 1 = 6$

$3 \times 10^1 = 3 \times 10 = 30$

$7 \times 10^2 = 7 \times 100 = 700$

Each position represents a power of 10

Positional Notation, Base 10

Decimal: Base-10:

$$7365 = 7 \times 10^3 + 3 \times 10^2 + 6 \times 10^1 + 5 \times 10^0$$

$$d_n \times 10^{n-1} + d_{n-1} \times 10^{n-2} + \cdots + d_2 \times 10^1 + d_1$$

Number Systems

How about base 2?

$$1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1$$

$$d_n \times 2^{n-1} + d_{n-1} \times 2^{n-2} + \cdots + d_2 \times 2^1 + d_1$$

Number Systems

How about any base?

$$d_n \times 10^{n-1} + d_{n-1} \times 10^{n-2} + \cdots + d_2 \times 10^1 + d_1$$

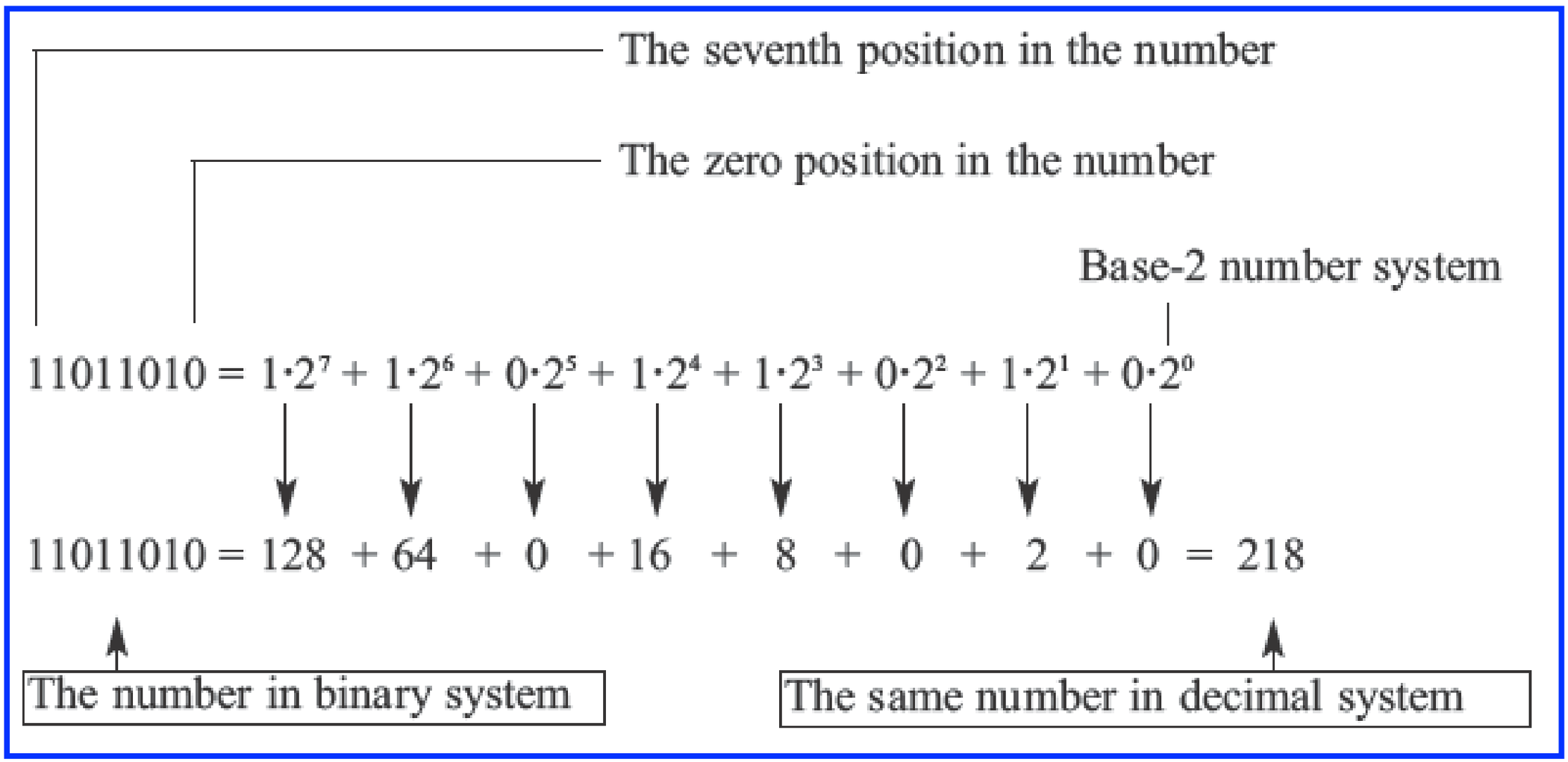
$$d_n \times 2^{n-1} + d_{n-1} \times 2^{n-2} + \cdots + d_2 \times 2^1 + d_1$$

If a number in the base-B number system has n digits, it is represented as the following polynomial, where d_i represents the digit in the i-th position

$$d_n \times B^{n-1} + d_{n-1} \times B^{n-2} + \cdots + d_2 \times B^1 + d_1$$

From Binary to Decimal

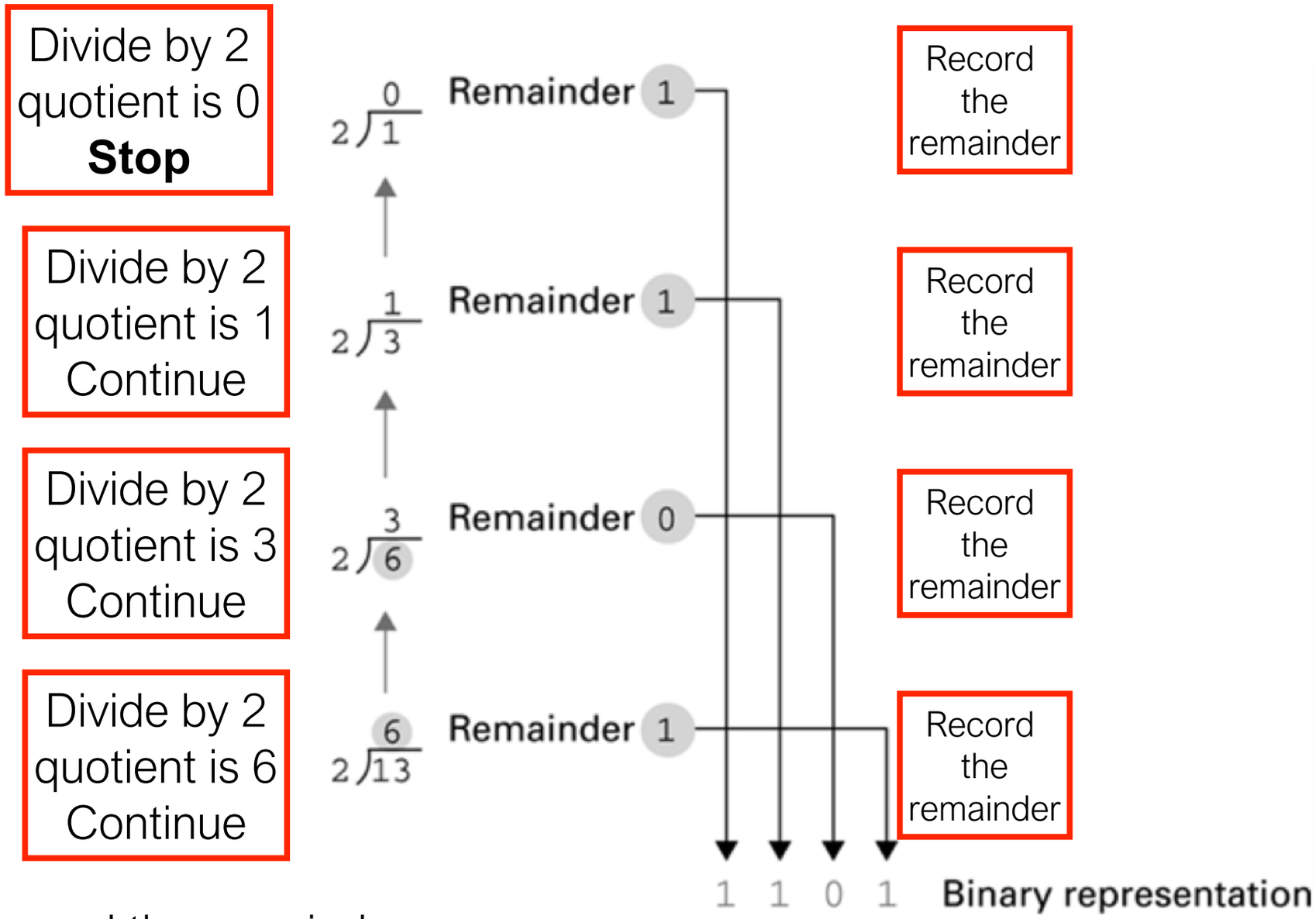
$$d_n \times 2^{n-1} + d_{n-1} \times 2^{n-2} + \cdots + d_2 \times 2^1 + d_1$$



How about from Decimal to Binary?

From Decimal to Binary

Translate 13 to binary:



1. Divide the value by 2 and record the remainder.
2. As long as the quotient obtained is not 0, repeat step 1
3. When the quotient of 0 has been obtained, the binary representation of the original value consists of the remainders listed from right to left in the order they were recorded.

Hexadecimal System: Base 16

$$d_n \times 16^{n-1} + d_{n-1} \times 16^{n-2} + \cdots + d_2 \times 16^1 + d_1$$

- Since there are only 10 decimal digits but the base is 16, we need additional digits: A,B,C,D,E,F
- Can be used as shorthand notation for long patterns of bits:
  Each group of 4 bits can be represented by a single symbol.
- 8 bits: 2 digits, 16 bits: 4 digits, 32 bits: 8 digits

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

8 + 2 2 + 1

10100011 becomes **A3**

A 3

Some Examples

What is 11000011 in hexadecimal notation?

C 3

What is 11111101 in hexadecimal notation?

F D

What is E6 in decimal notation?

$$2^7 + 2^6 + 2^5 + 2^2 + 2^1 = 230$$

1110 0110

$E*16^1 + 6*16^0 = 230$

$14*16^1 + 6*16^0 = 230$

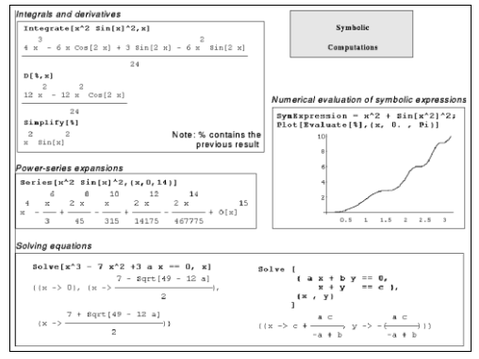
Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

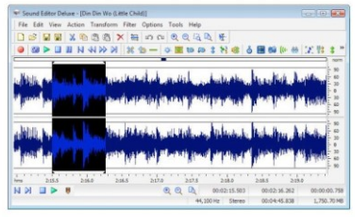
Information Storage and Processing

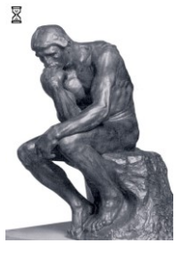
Goal: To understand how all modern computing systems
and computation using such systems
are based on binary numbers and operations on them!

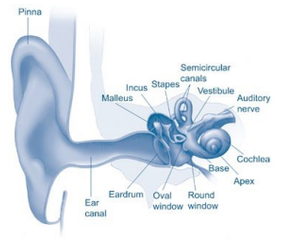
Computers execute binary computations. Any information we process is ultimately encoded and stored as binary numbers!

Images Video Text Symbols Sound









Any process on data is ultimately encoded and stored as binary numbers!

BITS - Binary Digits

Information in a computer is encoded as patterns of 1's and 0's.

Information = Bits + Context

The context provides an interpretation of the bit patterns!
The same bit pattern can mean different things.

For instance, a numerical value, character or a program instruction!

Logical Computations

1= true
0= false

And, Or
Not

Context: Logic

Numerical Computations

1 Addition, Subtraction
0 Multiplication, Division

Context: Numbers

BITS and BYTES

Most computers use blocks of 8 bits, called bytes, as the smallest addressable unit of memory

Byte = 8 bits

Binary 00000000_2 to 11111111_2

Decimal: 0_{10} to 255_{10}

Hexadecimal 00_{16} to FF_{16}

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Memory Size: Using Metric Prefixes

1000^m	10^n	Prefix	Symbol	Short scale	Long scale	Decimal
1000^8	10^{24}	yotta-	Y	Septillion	Quadrillion	1 000 000 000 000 000 000 000 000
1000^7	10^{21}	zetta-	Z	Sextillion	Trilliard	1 000 000 000 000 000 000 000
1000^6	10^{18}	exa-	E	Quintillion	Trillion	1 000 000 000 000 000 000
1000^5	10^{15}	peta-	P	Quadrillion	Billiard	1 000 000 000 000 000
1000^4	10^{12}	tera-	T	Trillion	Billion	1 000 000 000 000
1000^3	10^9	giga-	G	Billion	Milliard	1 000 000 000
1000^2	10^6	mega-	M	Million		1 000 000
1000^1	10^3	kilo-	k	Thousand		1 000
$1000^{2/3}$	10^2	hecto-	h	Hundred		100
$1000^{1/3}$	10^1	deca-	da	Ten		10
1000^0	10^0	*(none)*	*(none)*	One		1
$1000^{-1/3}$	10^{-1}	deci-	d	Tenth		0.1
$1000^{-2/3}$	10^{-2}	centi-	c	Hundredth		0.01
1000^{-1}	10^{-3}	milli-	m	Thousandth		0.001
1000^{-2}	10^{-6}	micro-	μ	Millionth		0.000 001
1000^{-3}	10^{-9}	nano-	n	Billionth	Milliardth	0.000 000 001
1000^{-4}	10^{-12}	pico-	p	Trillionth	Billionth	0.000 000 000 001
1000^{-5}	10^{-15}	femto-	f	Quadrillionth	Billiardth	0.000 000 000 000 001
1000^{-6}	10^{-18}	atto-	a	Quintillionth	Trillionth	0.000 000 000 000 000 001
1000^{-7}	10^{-21}	zepto-	z	Sextillionth	Trilliardth	0.000 000 000 000 000 000 001
1000^{-8}	10^{-24}	yocto-	y	Septillionth	Quadrillionth	0.000 000 000 000 000 000 000 001

What is a Googol?

$$10^{100}$$

What is a current estimate of atoms in the “observable” universe?

$$10^{80}$$

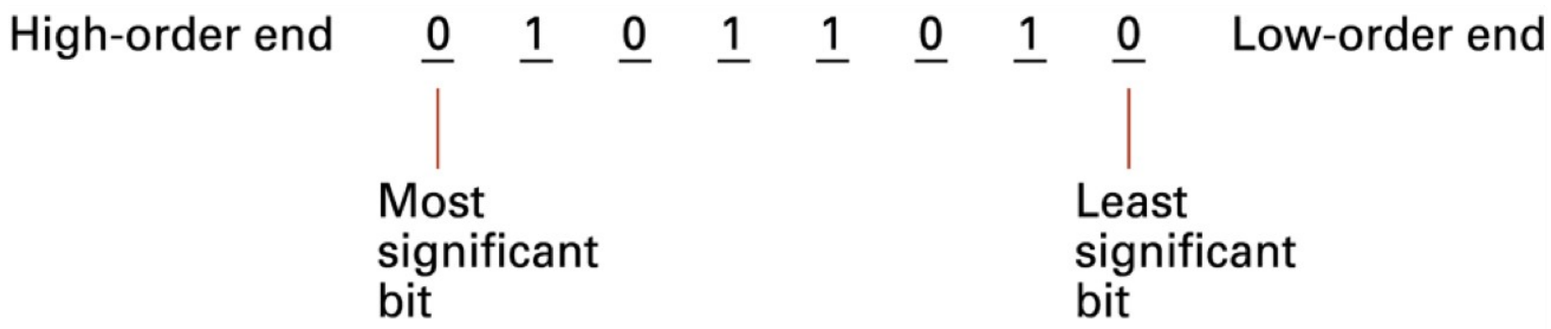
Bytes and Words

A Machine has “Word Size”

- Nominal size of integer data
- For a time, 16-bit words (2 bytes) were common
  - Numbers: 0 to 65535 (if positive)
  - Want to add larger numbers? Do multiple additions…
- Then for a long time, 32-bit words (4 bytes) were common
  - Hardware instructions processed 8, 16 or at most 32 bits at a time
- Most current machines, including phones, use 64-bit words (8 bytes)
  - Still support 8, 16, 32 bits; now also 64-bit numbers
- (And there are can be special instructions handling larger numbers; not covered here!)

Storage: Main Memory

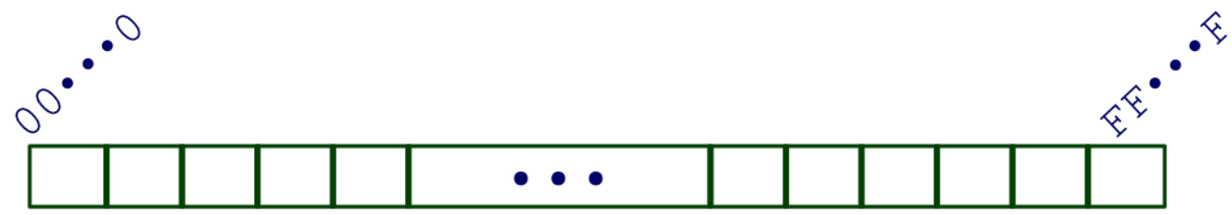
Memory Cell: is a unit of memory (usually a byte), organized in a bit order:



Address: A number that uniquely **identifies one cell** in the computer's main memory

- **Simple case**: These numbers are assigned consecutively starting at zero.
- Associates an order with the memory cells

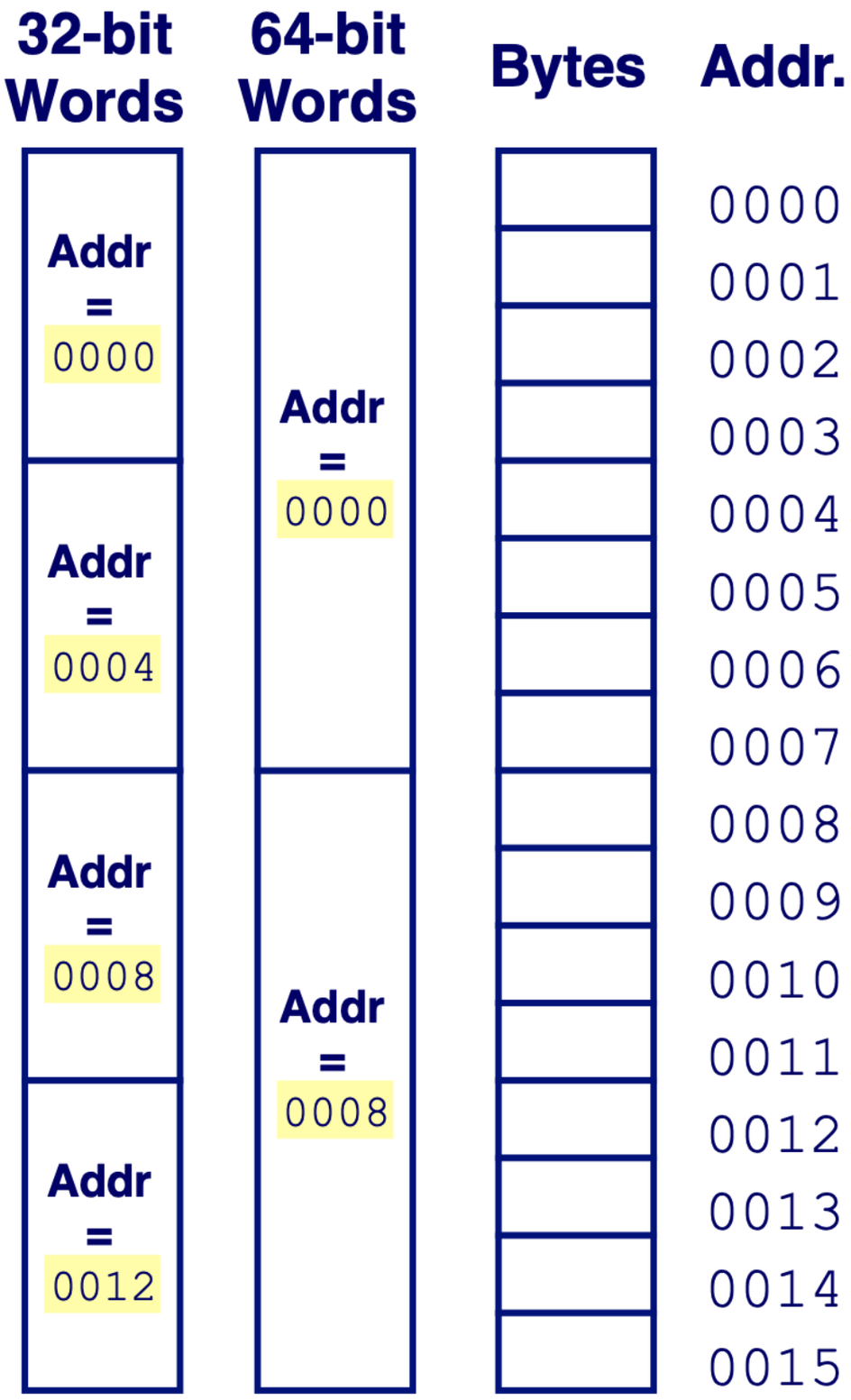
Byte Oriented Memory Organization



(**Virtual Memory**: Programs refer to virtual addresses – not covered here)

Word Oriented Memory Organisation

- **Addresses Specify Byte Locations**
  - Address of first byte in word
  - Addresses of successive words differ by 4 (32-bit) or 8 (64-bit)



Measuring Memory Capacity

- **Kilobyte:** 2^{10} bytes = 1024 bytes
  - Example: 3 KB = 3 times1024 bytes

- **Megabyte:** 2^{20} bytes = 1,048,576 bytes
  - Example: 3 MB = 3 times 1,048,576 bytes

- **Gigabyte:** 2^{30} bytes = 1,073,741,824 bytes
  - Example: 3 GB = 3 times 1,073,741,824 bytes

Or: Kibibyte, Mebibyte, Gibibyte:
Indicate that it's a "binary prefix" (bi),
not a power of 1000

Representing Unsigned Integers in Computers

Unsigned Integers: Not negative, no +/– sign

w= # of bits

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

BinaryToUnsigned

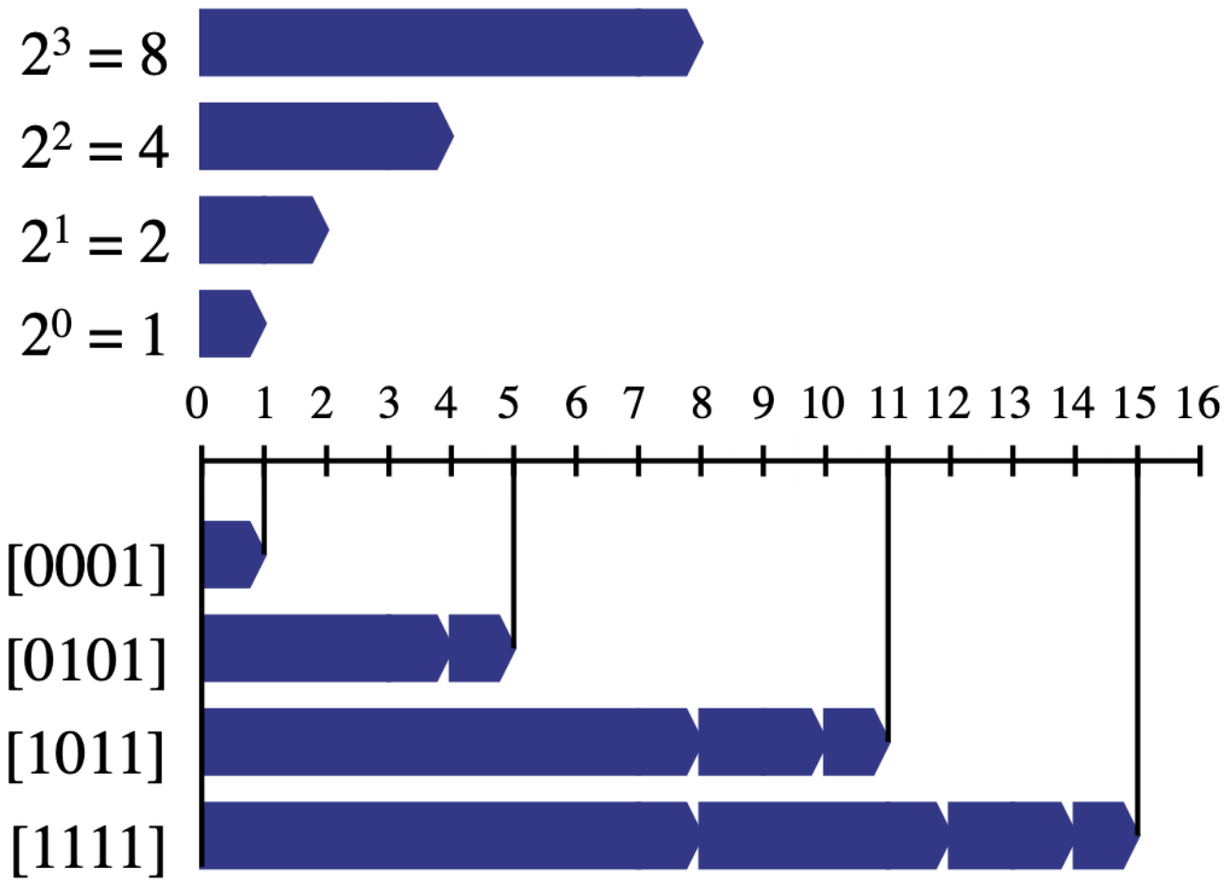
Numeric Range

UMin = 0

000…0

UMax = $2^w - 1$

111…1



$$\mathrm{B2U}(0001) = 0 \times 2^3 + 0 \times 2^2 + +0 \times 2^1 + 1 = 0 + 0 + 0 + 1 = 1$$

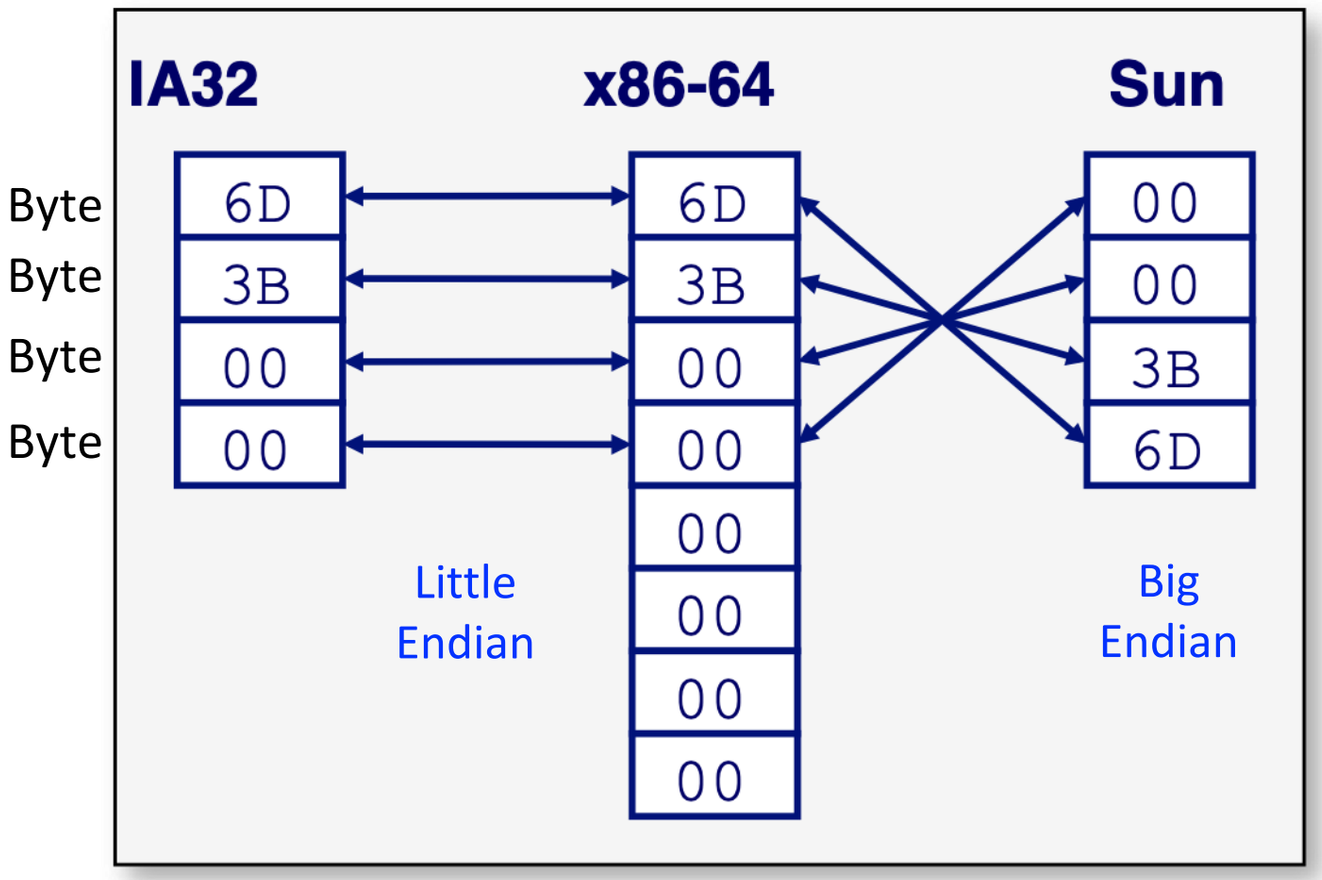
$$\mathrm{B2U}(0101) = 0 \times 2^3 + 1 \times 2^2 + +0 \times 2^1 + 1 = 0 + 4 + 0 + 1 = 5$$

$$\mathrm{B2U}(1011) = 1 \times 2^3 + 0 \times 2^2 + +1 \times 2^1 + 1 = 8 + 0 + 2 + 1 = 11$$

$$\mathrm{B2U}(1111) = 1 \times 2^3 + 1 \times 2^2 + +1 \times 2^1 + 1 = 8 + 4 + 2 + 1 = 15$$

Representing Unsigned Integers in various Machine Memory

```
long int C = 15213;
```



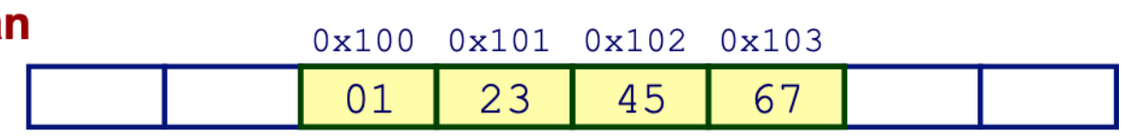
Decimal: 15213

Binary: 0011 1011 0110 1101

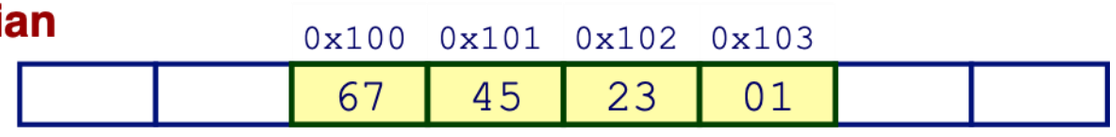
Hex: 3 B 6 D

- **Big Endian**
  - Least significant byte has highest address
- **Little Endian**
  - Least significant byte has lowest address
- **Example**
  - Variable x has 4-byte representation 0x01234567

Big Endian



Little Endian



Representing *Signed* Integers in Computers

X	
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Given 4 bits of memory, we can represent 2^4 unsigned numbers

(2^4 - 1 if 0 is included)

Suppose we wanted to represent both positive and negative numbers in memory……. Signed numbers

We would need to find an efficient mapping between our binary numbers and roughly $2^4/2$ positive numbers and $2^4/2$ negative numbers

Let's do that!

How about a ”sign bit”?

Could we use one bit to represent + or –
and the rest to represent the actual number?

Yes, and sometimes we do!

4 bits would allow us to represent +0 to +7,
and –0 to –7…

Might be a bit confusing to allow ”negative zero”!

More common: Two's Complement

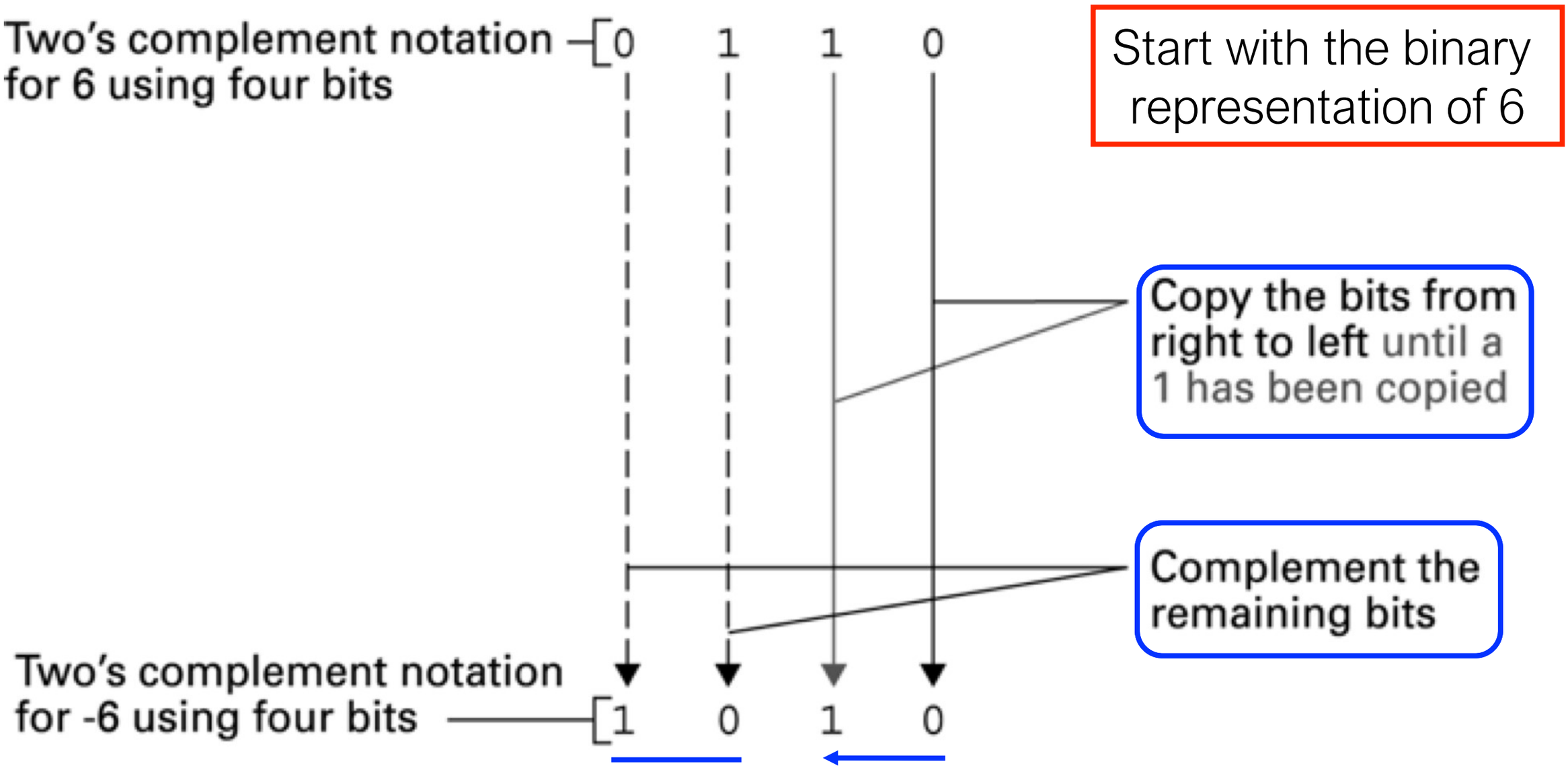
- Two's complement: <u>First bit represents -2^{w-1}, not 2^{w-1}</u>
- Binary 1010:
  -8 + 0 + 2 + 0 = -6

- **To convert -6 to binary using two's complement:**
- Start with the positive: 6
- Represent it in binary: 0110 (4 + 2)
- Flip the bits: 1001
- Add 1: 1010…
  This is the internal representation of -6
- Note: <u>**First bit is set**</u>, indicating a negative number!

More common: Two's Complement

- What happens with zero?
- Represent it in binary: 0000
- Flip the bits: 1111
- Add 1: Result is 10000,
  but we only work with 4 bits: 0000 (bit 5 disappears)
- So minus zero is zero

Another method, instead of adding 1



Two's Complement Examples

a. Using patterns of length three

Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	−1
110	−2
101	−3
100	−4

b. Using patterns of length four

Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

Comparison with Unsigned

X	B2U(*X*)	B2T(*X*)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	–8
1001	9	–7
1010	10	–6
1011	11	–5
1100	12	–4
1101	13	–3
1110	14	–2
1111	15	–1

Addition & Subtraction converted to Two's Complement

The Payoff!

Problem in base ten		Problem in two's complement		Answer in base ten
3 + 2	→	0011 + 0010 0101	→	5
-3 + -2	→	1101 + 1110 1011	→	-5
7 + -5	→	0111 + 1011 0010	→	2

Example: 7 - 5

For addition:

- Same algorithm as for binary addition.
- any extra carry bit generated on the left is truncated.
- addition of any combination of signed integers uses the same algorithm and circuitry!

For subtraction:

- negate the number subtracted and then add both together.
- subtraction of any combination of signed integers uses the same algorithm and circuitry for addition plus an additional circuit for negation of an integer!

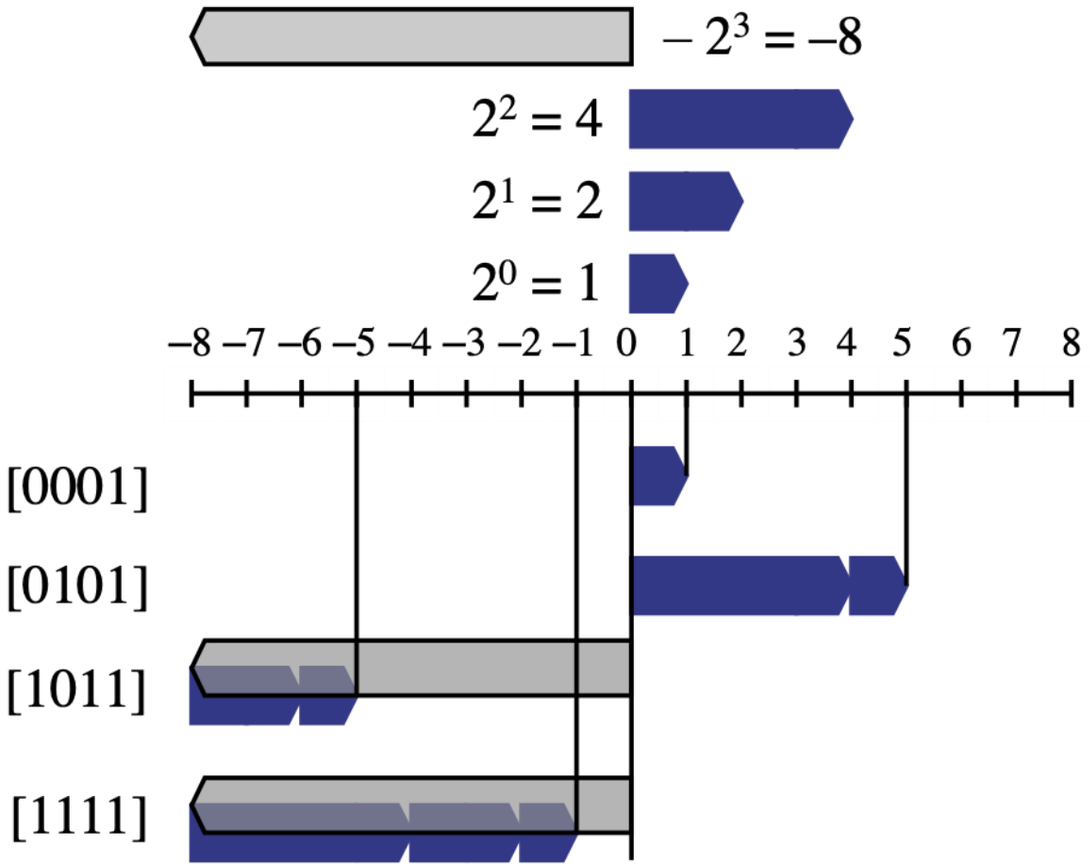
Representing Signed Integers in Computers

Two's Complement Decoding

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

Sign Bit

1 = negative Int
0 = positive Int



w=4

Numeric Range

TMin = -2^{w-1}
100…0

TMax = $2^{w-1} - 1$ Half of 2^w -1
011…1

$$\mathrm{B2T}(0001) = -0 \times 2^3 + 0 \times 2^2 + +0 \times 2^1 + 1 = 0 + 0 + 0 + 1 = 1$$

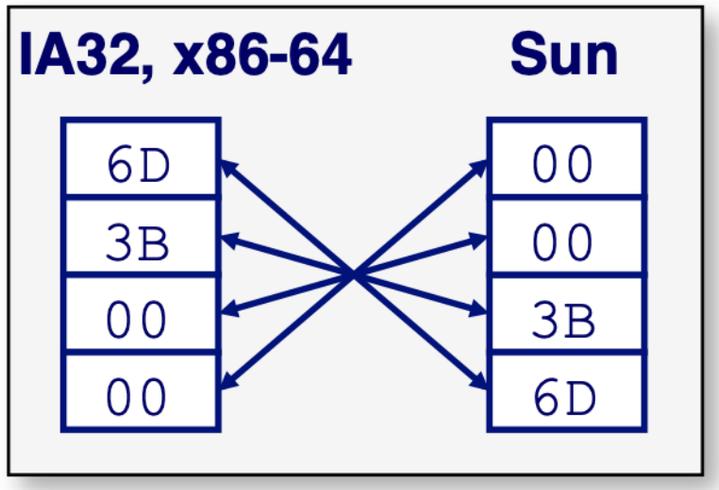
$$\mathrm{B2T}(0101) = 0 \times 2^3 + 1 \times 2^2 + +0 \times 2^1 + 1 = 0 + 4 + 0 + 1 = 5$$

$$\mathrm{B2T}(1011) = \mathbf{-1} \times 2^3 + 0 \times 2^2 + +1 \times 2^1 + 1 = \mathbf{-8} + 0 + 2 + 1 = \mathbf{-5}$$

$$\mathrm{B2T}(1111) = \mathbf{-1} \times 2^3 + 1 \times 2^2 + +1 \times 2^1 + 1 = \mathbf{-8} + 4 + 2 + 1 = \mathbf{-1}$$

Representing Signed Integers in various Machine Memory

```
int A = 15213;
```

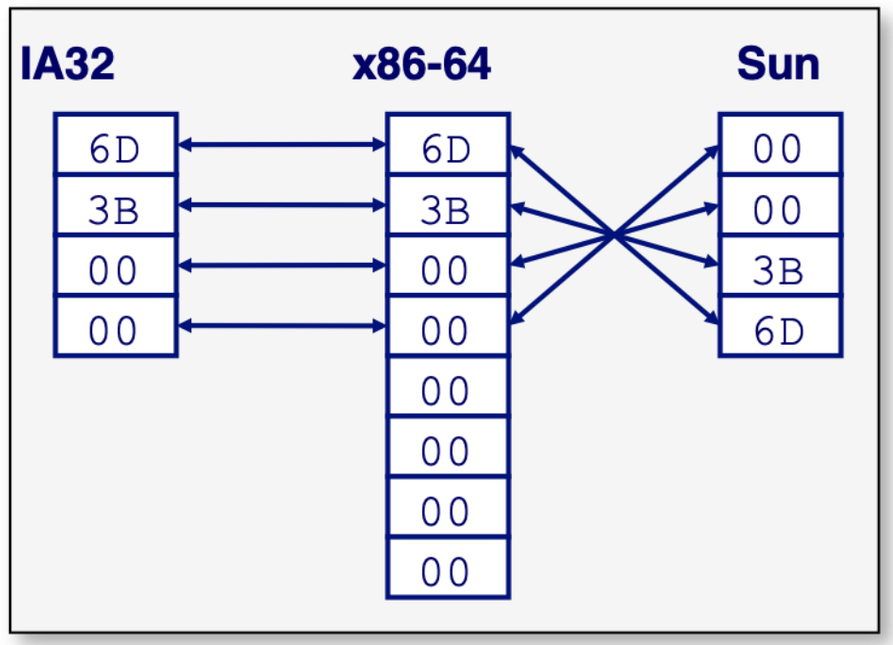


Little Endian

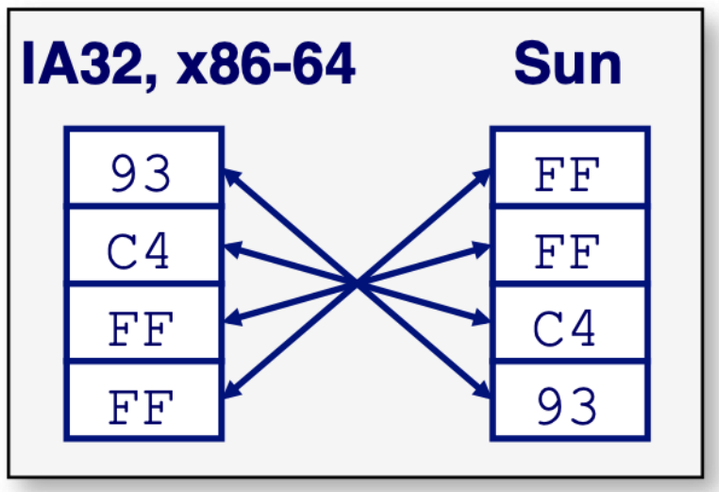
Big Endian

Decimal:	15213
Binary:	0011 1011 0110 1101
Hex:	3 B 6 D

```
long int C = 15213;
```




```
int B = -15213;
```



Two's Complement Representation

Decimal:	-15213
Binary:	1111 1111 1111 1111 1100 0100 1001 0011
Hex:	F F F F C 4 9 3

Numeric Ranges of Integer Representations

Unsigned Values

$UMin = 0$

000…0

$UMax = 2^w - 1$

111…1

Two's Complement Values

$TMin = -2^{w-1}$

100…0

$TMax = 2^{w-1} - 1$

011…1

Half

For Different Word Sizes

	W			
[Bits]	**8**	**16**	**32**	**64**
UMax	255	65,535	4,294,967,295	18,446,744,073,709,551,615
TMax	127	32,767	2,147,483,647	9,223,372,036,854,775,807
TMin	-128	-32,768	-2,147,483,648	-9,223,372,036,854,775,808

Signed Integers in Excess Notation (unusual!)

Bit pattern	Value represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	-1
0110	-2
0101	-3
0100	-4
0011	-5
0010	-6
0001	-7
0000	-8

- Select pattern length to be used
- Write down all different patterns in the order they would appear if counting from bottom up.
- Pick the 1st pattern with 1 as most significant bit to represent 0
- Patterns preceding 0 are used for -1, -2, -3 ...
- Patterns proceeding 0 are used for 1, 2, 3, ...

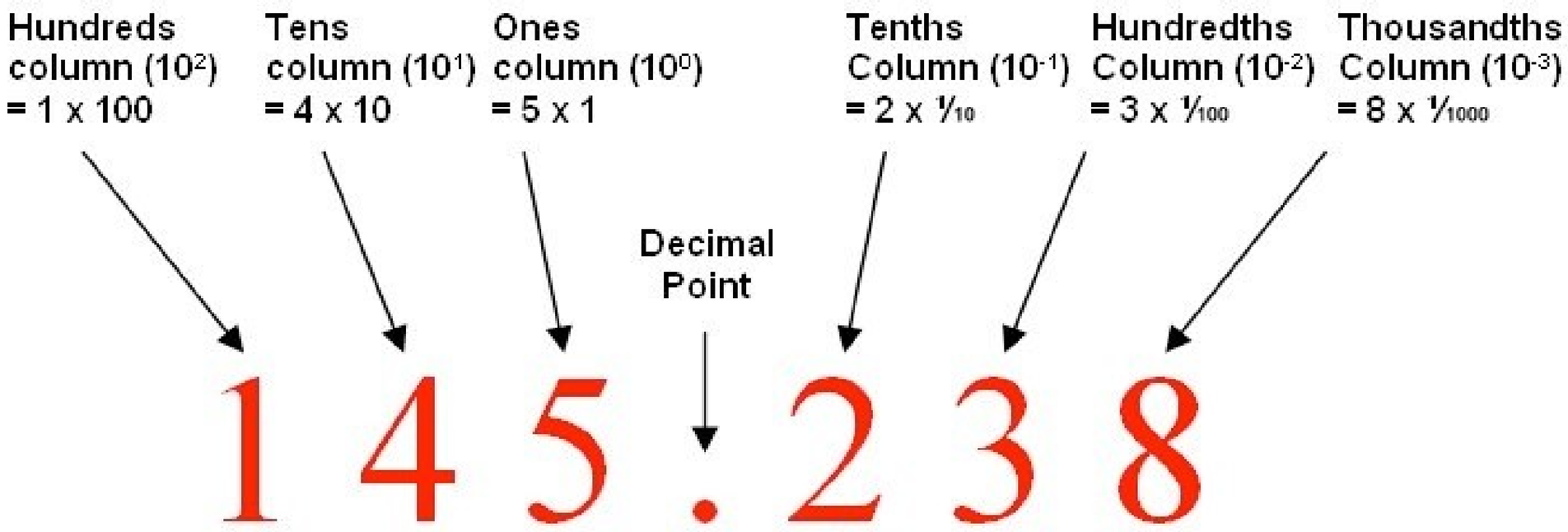
If the pattern length is x, the difference between the bit pattern value and the value represented is 2^{x-1}

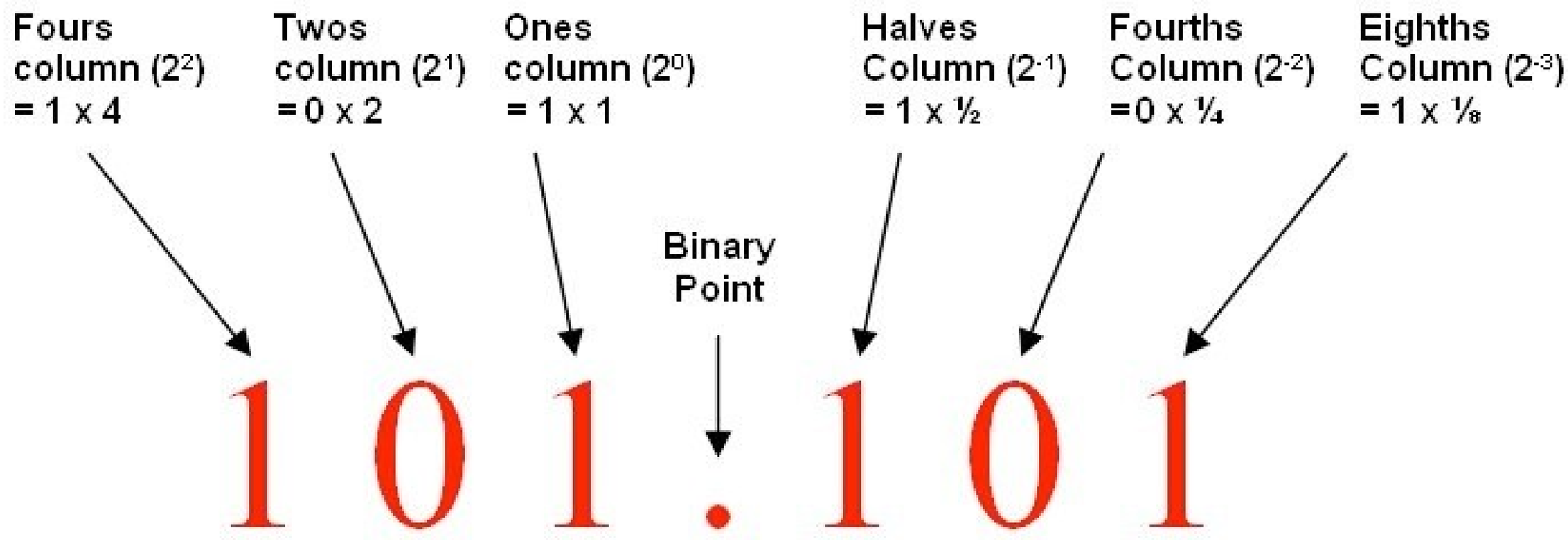
x=4, excess-8 notation, x= 5, excess-16 notation

$$1111 = 15 : 15 - 7 = 8 = 2^3$$

Bit pattern — Value Represented

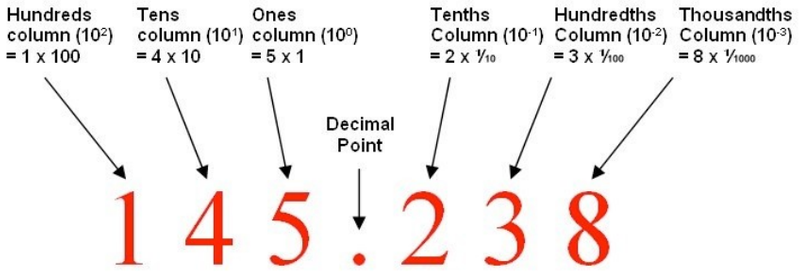
Representing Fractions (Rational Numbers)

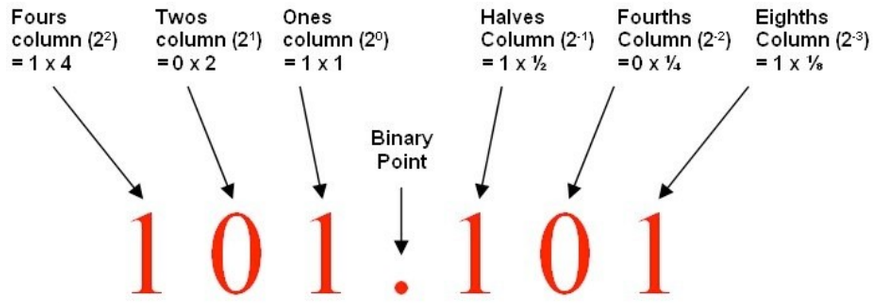


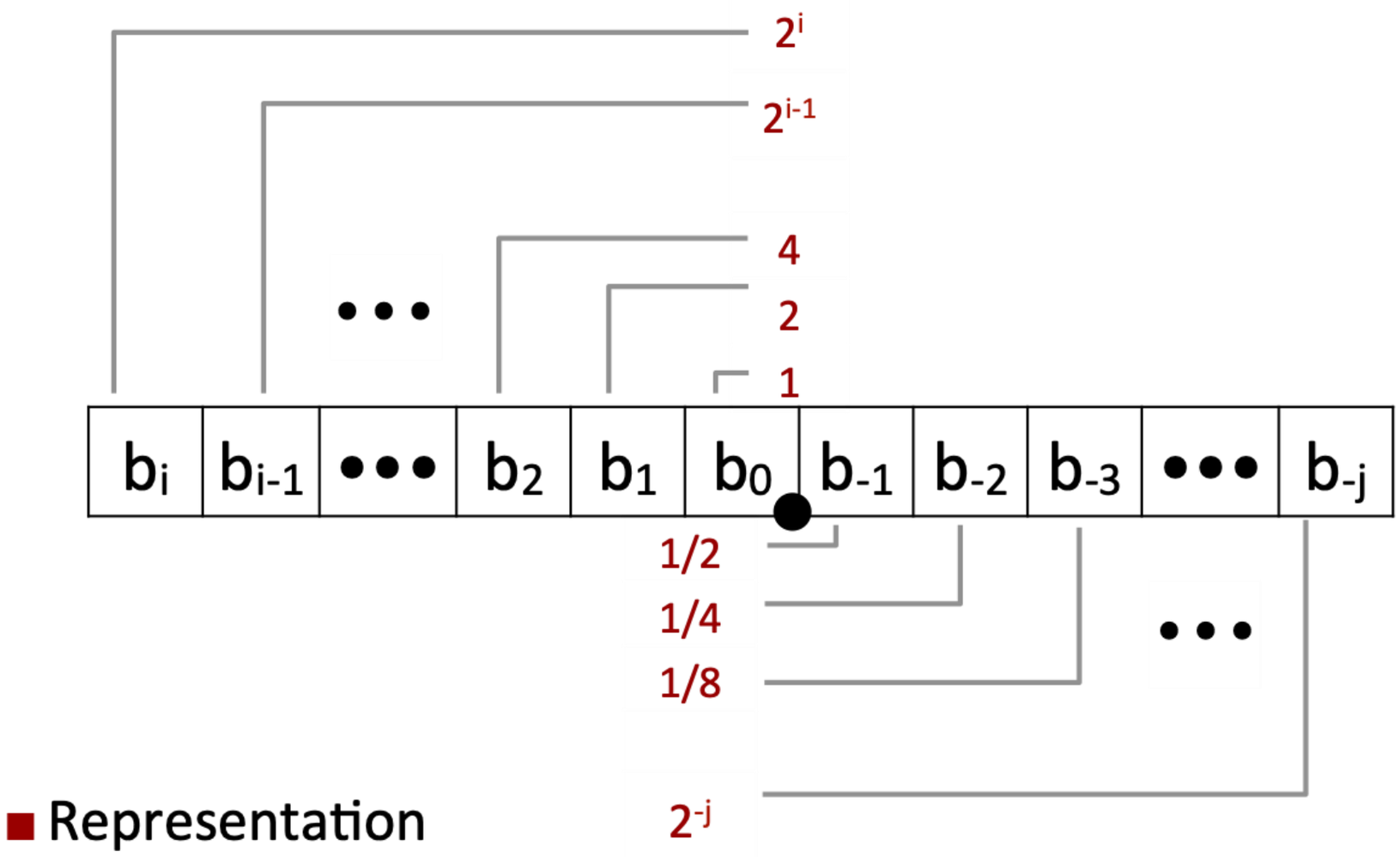


Radix Point

Representing Fractions (Rational Numbers)





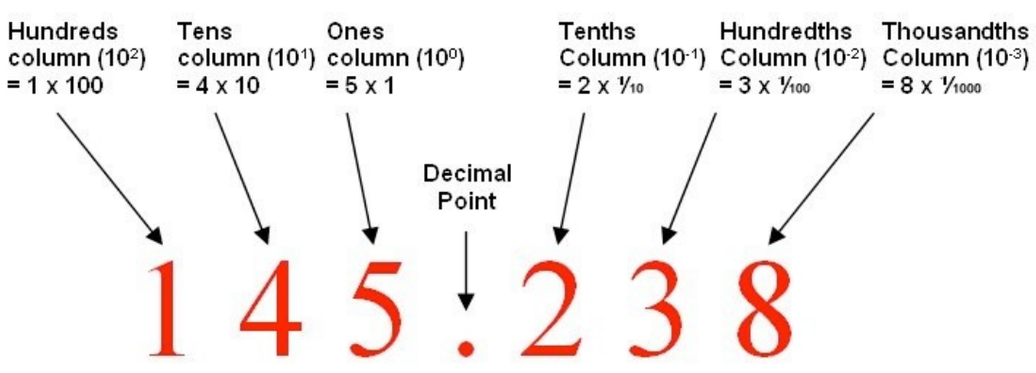


- Representation
  - Bits to right of "binary point" represent fractional powers of 2
  - Represents rational number:

$$\sum_{k=-j}^{i} b_k \times 2^k$$

Division & Multiplication

Base-10



Decimal Point

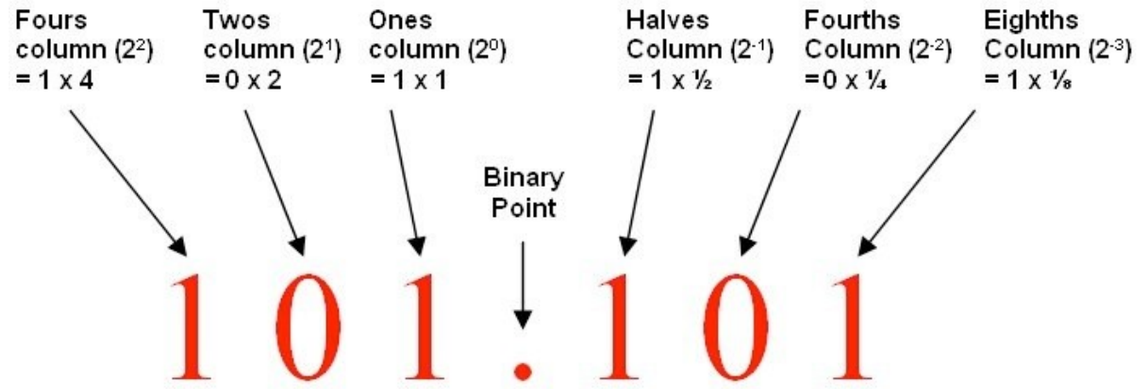
Shifting the decimal point 1 position to the left has the effect of dividing the number by 10.

Shifting the decimal point 1 position to the right has the effect of multiplying the number by 10.

145.238

14.5238
1452.38

Base-2



Radix Point

Shifting the radix point 1 position to the left has the effect of dividing the number by 2.

Shifting the radix point 1 position to the right has the effect of multiplying the number by 2.

101.11 = 5 3/4

10.111 = 2 7/8
1011.1 = 11 1/2

Representable Numbers

Assuming finite length encodings:

- **Limitation**
  - Can only exactly represent numbers of the form $x/2^k$
  - Other rational numbers have repeating bit representations

- **Value** **Representation**
  - 1/3 $0.0101010101[01]\ldots_2$
  - 1/5 $0.001100110011[0011]\ldots_2$
  - 1/10 $0.0001100110011[0011]\ldots_2$

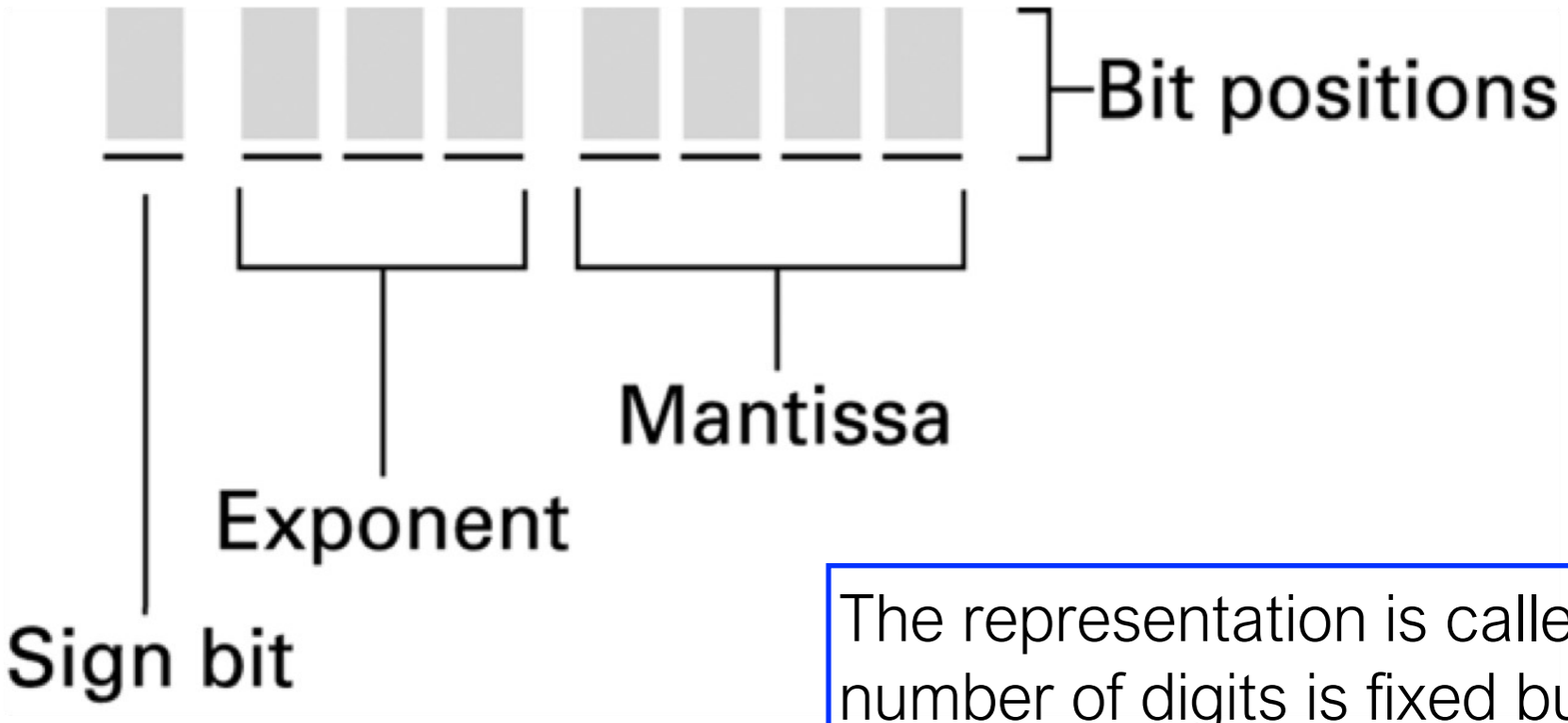
So, we must eventually truncate!

Can approximate with increasing accuracy by lengthening the binary representation

Floating Point Representation

Representing a real value in a computer:

- The sign - positive or negative
- The mantissa - consists of digits in the value with the radix point assumed to the left
- The exponent - which determines how the radix point is shifted relative to the mantissa (can be positive or negative)



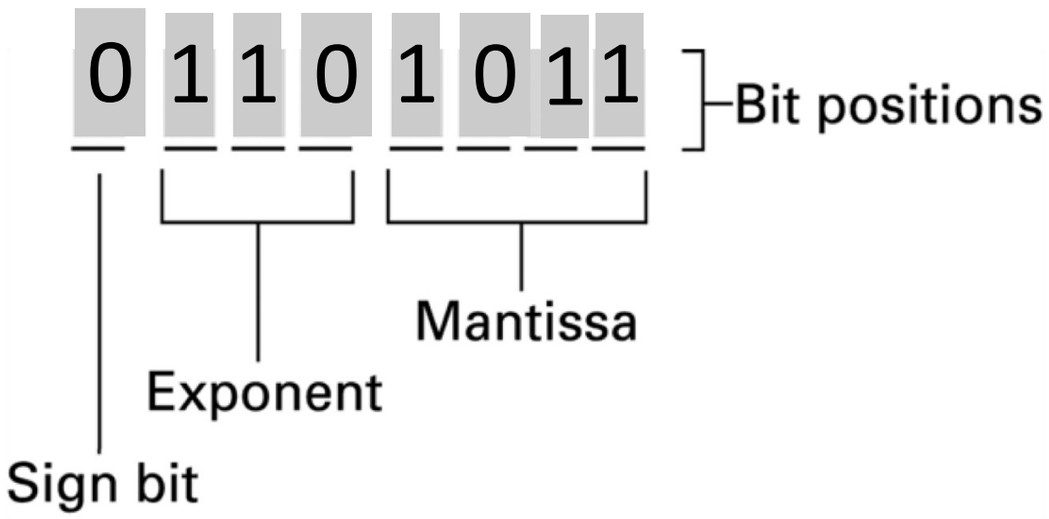
Let's Assume an 8-bit fixed representation (usually much larger 64 bits):

- sign: 1 bit,
- exponent: 11 bits,
- mantissa: 52 bits

The representation is called floating point because the number of digits is fixed but the radix point floats.

- a positive exponent shifts the radix point to the right
- a negative exponent shifts the radix point to the left

Decoding the value 01101011

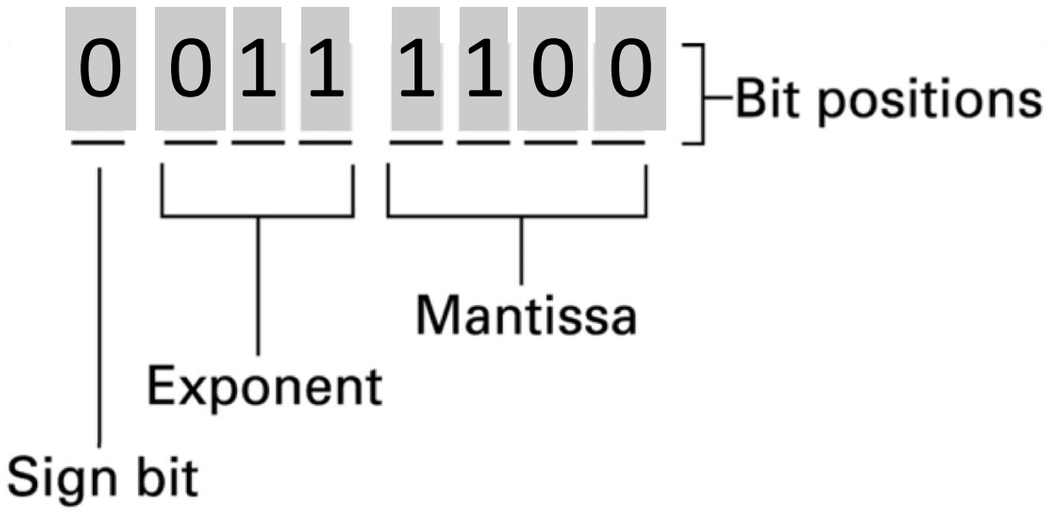


Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

Exponent encoded in excess-3

	. 1 0 1 1	Extract Mantissa
2	1 1 0	Extract Exponent Interpret in excess-3
	1 0 . 1 1	Move radix point to the right 2
	2+0+1/2+1/4 = **2 3/4**	Translate to decimal form
	+ 2 3/4	Check sign bit 0 is positive 1 is negative
	2.75	

Decoding the value 00111100



Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

Exponent encoded in excess-3

. 1 1 0 0 — Extract Mantissa

-1 — 0 1 1 — Extract Exponent
Interpret in excess-3

. 0 1 1 0 0 — Move radix point to the left 1

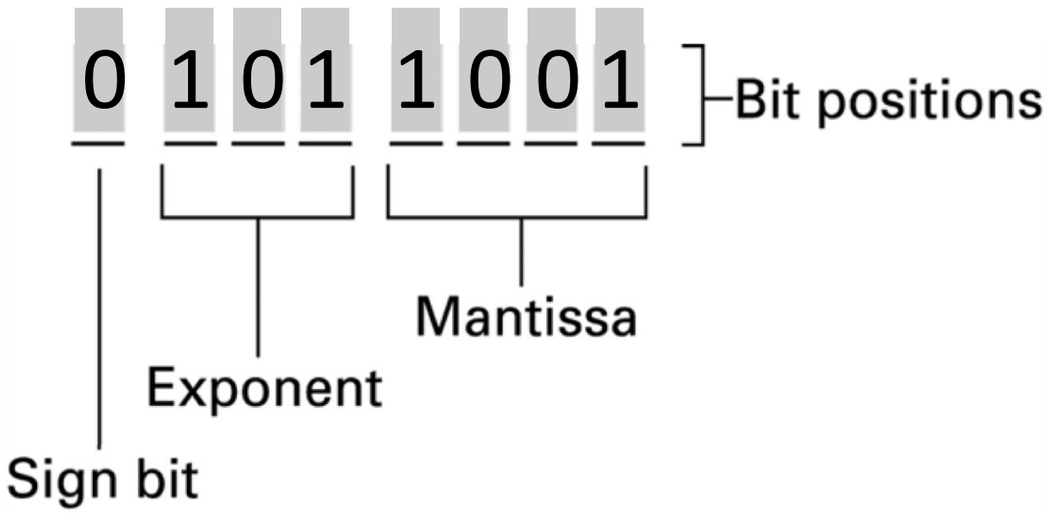
0/2+1/4+1/8 = 3/8 — Translate to decimal form

+ 3/8 — Check sign bit
0 is positive
1 is negative

0.375

LINKÖPING UNIVERSITY

Encoding the value 1+1/8 (1.125)

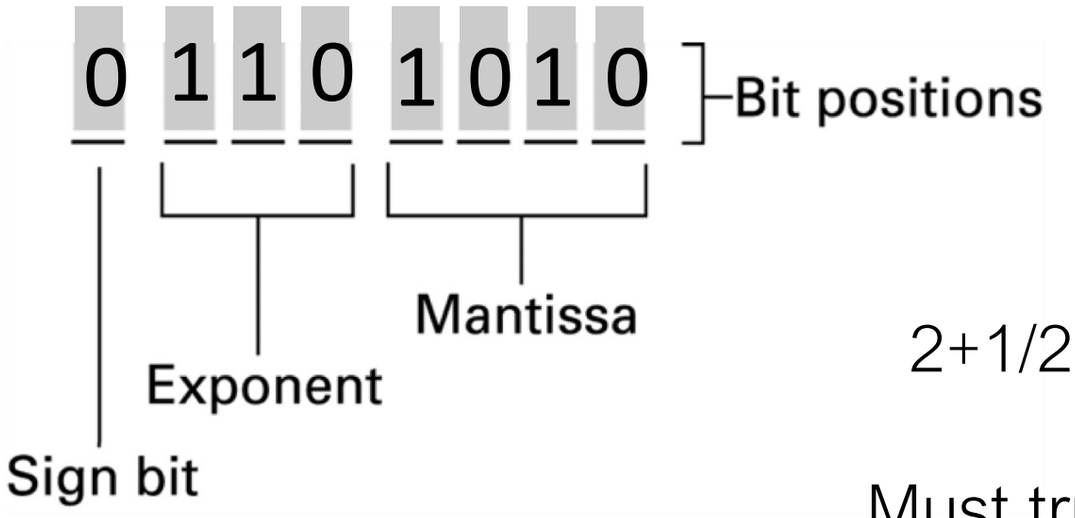


Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

Exponent encoded in excess-4

1 . 0 0 1	Encode the number in binary notation
1 0 0 1	Copy the pattern into the mantissa (left to right) Starting with leftmost 1
. 1 0 0 1	Imagine the radix point to the left of the mantissa value
1 to the right (+1)	Determine direction and length to move radix point to recover the original binary number
1 0 1	Encode in exponent field using excess notation
0	Encode sign bit 0 is positive 1 is negative

Truncation Errors: Encoding the value 2 5/8 (2.625)



Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

Exponent encoded in excess-4

2+1/2+1/8=2 5/8

Must truncate the rightmost 1 (1/8)

1 0 . 1 0 1

1 0 1 0

. 1 0 1 0

2 to the right (+1)

1 1 0

0

Encode the number in binary notation

Copy the pattern into the mantissa (left to right)
Starting with leftmost 1

Imagine the radix point to the left of the mantissa value

Determine direction and length to move radix point to recover the original binary number

Encode in exponent field using excess notation

Encode sign bit
0 is positive
1 is negative

Result:
01101010 = 2 1/2
Truncation or Rounding error!

Truncation Errors

Can cause significant problems in applications requiring high precision!

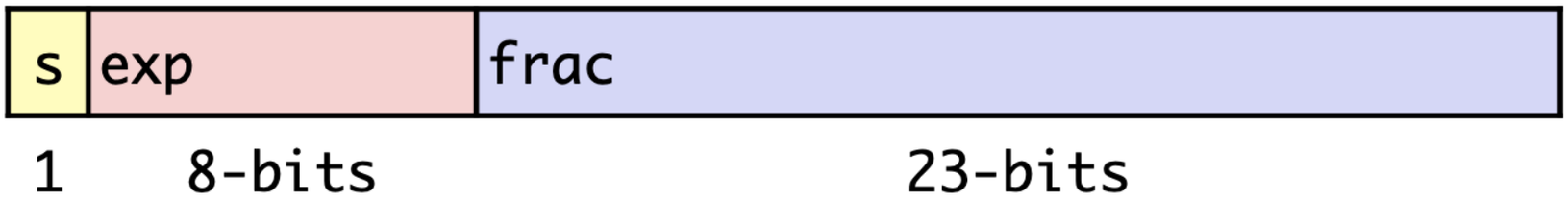
There are many more repeating decimal numbers in binary than in decimal notation. Example: 1/10

Arithmetic with accumulated rounding errors can be problematic:
8 bit representation:
2 ½ + 1/8 + 1/8 ➔ 2 ½ + 1/8 ➔ 2 ½ (small parts disappear)
1/8 + 1/8 + 2 ½ ➔ ½ + 2 ½ ➔ 3 (accumulate small parts first)

Topic: Numerical Analysis

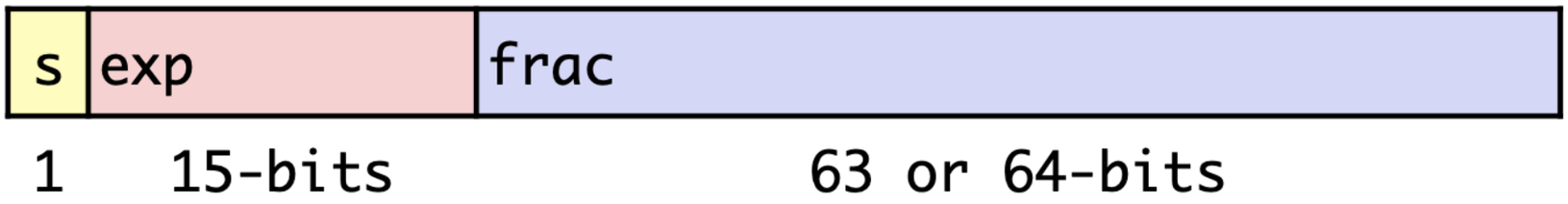
Worrying about precision

- Single precision: 32 bits



- Double precision: 64 bits

- Extended precision: 80 bits (Intel only)



10^{-37} to 10^{38}
Precision: 7 decimal digits

Single Precision

- Encodes very small real numbers and very large real numbers
- very accurate precision to 7 decimals...

IEEE Floating Point Standard

- IEEE Standard 754
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats
  - Supported by all major CPUs

- Driven by numerical concerns
  - Nice standards for rounding, overflow, underflow
  - Hard to make fast in hardware
    - Numerical analysts predominated over hardware designers in defining standard

The High Cost of Floating Point Overflow!

On June 4, 1996, the **US$500 million** Ariane 5 spacecraft was launched for the first time.

Sadly, the primary cause was found to be a piece of software which had been retained from the previous launcher's systems and which was not required during the flight of Ariane 5. The software was used in the Inertial Reference System (SRI) to calculate the attitude of the launcher. In Ariane 4, this software was allowed to continue functioning during the first 50 seconds of flight as it could otherwise delay launching if the countdown was halted for any other reason, this was not necessary for Ariane 5. Additionally, the software contained implicit assumptions about the parameters, in particular the horizontal velocity that was safe for Ariane 4, but not Ariane 5.

The failure occurred because the horizontal velocity exceeded the maximum value for a 16 bit unsigned integer when it was converted from it's signed 64 bit representation. This failure generated an exception in the code which was not caught and thus propagated up through the processor and ultimately caused the SRI to fail. The failure triggered the automatic fail-over to the backup SRI which had already failed for the same reason. This combined failure was then communicated to the main computer responsible for controlling the jets of the rocket, however, this information was misinterpreted as valid commands. As a result of the invalid commands, the engine nozzles were swung to an extreme position and the launcher was destroyed shortly afterwards.

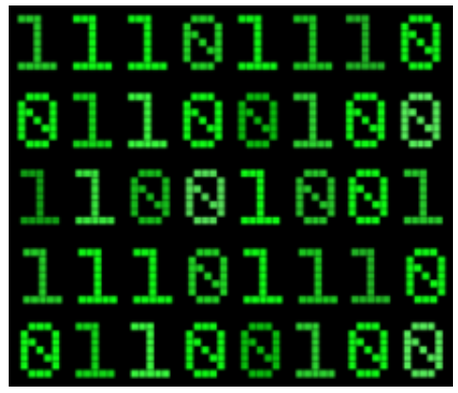
The High Cost of Floating Point Overflow!

On June 4, 1996, the **US$500 million** Ariane 5 spacecraft was launched for the first time

- Piece of software retained from Ariane 4, used to calculate launcher attitude
- Contained implicit assumptions about safe horizontal velocities (different in Ariane 5)
- Horizontal velocity exceeded max for 16 bit unsigned integer
- Velocity was actually safe, but overflow ➔ software crash
- Failed over to backup system; failed for the same reason
- Failure communicated to main computer, misinterpreted as valid commands
- Engine nozzles swung to an extreme position
- Launcher destroyed

Final Words

Do programmers and computer engineers work with ones and zeros?



Yes… in the same way authors and journalists work with the alphabet