

CSC 411

Computer Organization (Spring 2022)

Lecture 2: Number Systems, Bitwise Operations

Number Systems

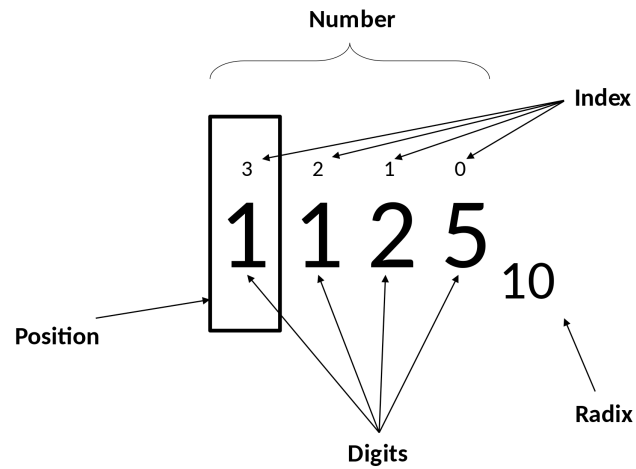
Number systems

- A way to represent numbers
 - numbers are expressed in a certain **base**
- Why study number systems in **CS**?
 - to understand data representation
- Examples of number systems
 - binary
 - decimal
 - octal
 - hexadecimal

Number Systems

System	Base	Digits
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

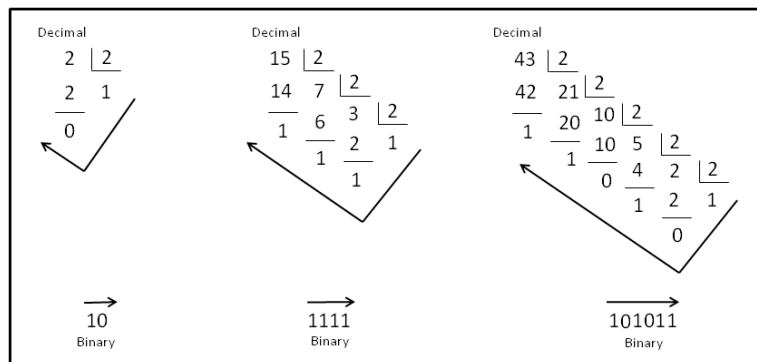
Positional notation



https://en.wikipedia.org/wiki/Positional_notation

Conversions to decimal

Conversions from decimal



https://en.wikiversity.org/wiki/Numeral_systems

Examples

Binary to hexadecimal

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Bin	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Oct	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17

Humans think in **base 10**. Computers think in **base 2**.
Humans use **base 16** to easily manipulate data in **base 2**.

Color codes

	Name ↕	Hex (RGB) ↕
	White	#FFFFFF
	Silver	#C0C0C0
	Gray	#808080
	Black	#000000
	Red	#FF0000
	Maroon	#800000
	Yellow	#FFFF00
	Olive	#808000
	Lime	#00FF00
	Green	#008000
	Aqua	#00FFFF
	Teal	#008080
	Blue	#0000FF
	Navy	#000080
	Fuchsia	#FF00FF
	Purple	#800080

Integer literals in C/C++

```
int d = 42;  
int o = 052;  
int x = 0x2a;  
int X = 0X2A;  
int b = 0b101010; // C++14
```

- ✓ **decimal-literal** is a non-zero decimal digit (1, 2, 3, 4, 5, 6, 7, 8, 9), followed by zero or more decimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- ✓ **octal-literal** is the digit zero (0) followed by zero or more octal digits (0, 1, 2, 3, 4, 5, 6, 7)
- ✓ **hex-literal** is the character sequence `0x` or the character sequence `0X` followed by one or more hexadecimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, A, b, B, c, C, d, D, e, E, f, F)
- ✓ **binary-literal** is the character sequence `0b` or the character sequence `0B` followed by one or more binary digits (0, 1)

Bits and Bytes

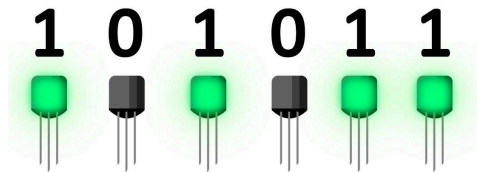
Bits and computers

• A bit can only have two values (states)

- easy to embed into physical devices

• Transistor

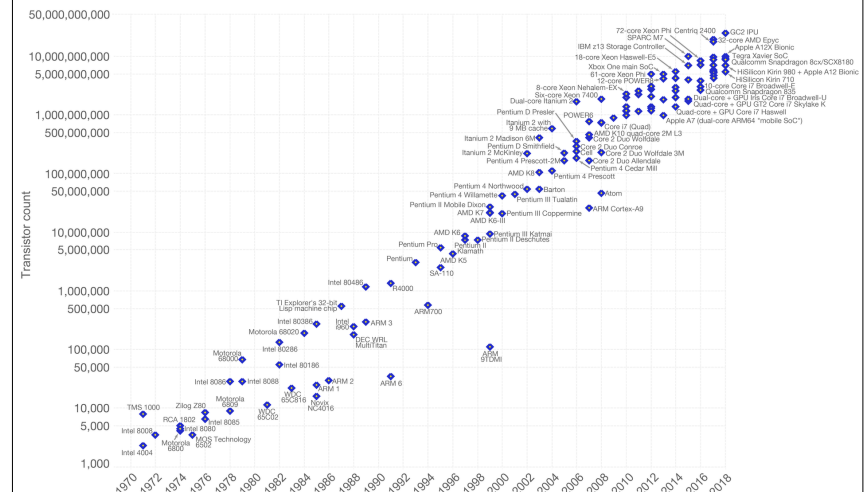
- processors have billions of transistors
- transistors can be switched on and off



Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.

OurWorld
in Data



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)

The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

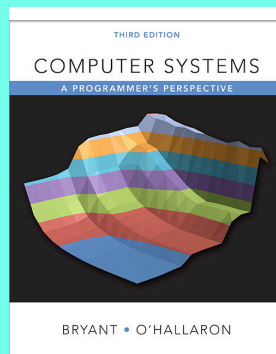
Licensed under CC-BY-SA by the author Max Roser.

Disclaimer

The following slides are from:

Computer Systems (Bryant and O'Hallaron)

A Programmer's Perspective



Everything is bits

- Each bit is 0 or 1
- By encoding/interpreting sets of bits in various ways
 - Computers determine what to do (instructions)
 - ... and represent and manipulate numbers, sets, strings, etc...
- Why bits? Electronic Implementation
 - Easy to store with bistable elements
 - Reliably transmitted on noisy and inaccurate wires

An Amazing & Successful Abstraction.

(which we won't dig into in 213)

Encoding Byte Values

Byte = 8 bits

- Binary 00000000_2 to 11111111_2
- Decimal: 0_{10} to 255_{10}
- Hexadecimal 00_{16} to FF_{16}
 - Base 16 number representation
 - Use characters '0' to '9' and 'A' to 'F'
 - Write $FA1D37B_{16}$ in C as
 - $0xFA1D37B$
 - $0xfa1d37b$

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

15213: 0011 1011 0110 1101
 3 B 6 D

Example Data Representations

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	8	8
float	4	4	4
double	8	8	8
pointer	4	8	8

Boolean Algebra

Developed by George Boole in 19th Century

- Algebraic representation of logic
 - Encode "True" as 1 and "False" as 0

And

- $A \& B = 1$ when both $A=1$ and $B=1$

&	0	1
0	0	0
1	0	1

Or

- $A | B = 1$ when either $A=1$ or $B=1$

	0	1
0	0	1
1	1	1

Not

- $\sim A = 1$ when $A=0$

~	
0	1
1	0

Exclusive-Or (Xor)

- $A \wedge B = 1$ when either $A=1$ or $B=1$, but not both

^	0	1
0	0	1
1	1	0

General Boolean Algebras

Operate on Bit Vectors

- Operations applied bitwise

$$\begin{array}{cccc} 01101001 & 01101001 & 01101001 & \\ \& 01010101 & | 01010101 & ^ 01010101 \\ \hline 01000001 & 01111101 & 00111100 & 10101010 \end{array}$$

- All of the Properties of Boolean Algebra Apply

Example: Representing & Manipulating Sets

Representation

- Width w bit vector represents subsets of $\{0, \dots, w-1\}$
- $a_j = 1$ if $j \in A$

▪ 01101001 { 0, 3, 5, 6 }

▪ 76543210

▪ 01010101 { 0, 2, 4, 6 }

▪ 76543210

Operations

- & Intersection 01000001 { 0, 6 }
- | Union 01111101 { 0, 2, 3, 4, 5, 6 }
- ^ Symmetric difference 00111100 { 2, 3, 4, 5 }
- ~ Complement 10101010 { 1, 3, 5, 7 }

Bit-Level Operations in C

Operations &, |, ~, ^ Available in C

- Apply to any "integral" data type
 - long, int, short, char, unsigned
- View arguments as bit vectors
- Arguments applied bit-wise

Examples (Char data type)

- ~0x41 →
- ~0x00 →
- 0x69 & 0x55 →
- 0x69 | 0x55 →

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

Bit-Level Operations in C

Operations &, |, ~, ^ Available in C

- Apply to any "integral" data type
 - long, int, short, char, unsigned
- View arguments as bit vectors
- Arguments applied bit-wise

Examples (Char data type)

- ~0x41 → 0xBE
 - ~0100 0001₂ → 0111 1110₂
- ~0x00 → 0xFF
 - ~0000 0000₂ → 1111 1111₂
- 0x69 & 0x55 → 0x41
 - 0110 1001₂ & 0101 0101₂ → 0100 0001₂
- 0x69 | 0x55 → 0x7D
 - 0110 1001₂ | 0101 0101₂ → 0111 1101₂

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

Contrast: Logic Operations in C

Contrast to Bit-Level Operators

- Logic Operations: &&, ||, !
 - View 0 as "False"
 - Anything nonzero as "True"
 - Always return 0 or 1
 - Early termination

Examples (char data type)

- !0x41 → 0x00
- !0x00 → 0x01
- !!0x41 → 0x01
- 0x69 && 0x55 → 0x01
- 0x69 || 0x55 → 0x01
- p && *p (avoids null pointer access)

Watch out for && vs. & (and || vs. |)...
Super common C programming pitfall!

Shift Operations

■ Left Shift: $x \ll y$

- Shift bit-vector x left y positions
 - Throw away extra bits on left
 - Fill with 0's on right

■ Right Shift: $x \gg y$

- Shift bit-vector x right y positions
 - Throw away extra bits on right
- Logical shift
 - Fill with 0's on left
- Arithmetic shift
 - Replicate most significant bit on left

■ Undefined Behavior

- Shift amount < 0 or \geq word size

Argument x	01100010
$\ll 3$	00010000
Log. $\gg 2$	00011000
Arith. $\gg 2$	00011000

Argument x	10100010
$\ll 3$	00010000
Log. $\gg 2$	00101000
Arith. $\gg 2$	11101000