

CS 211: Computer Architecture

Data representation

What Do Computers Do?

Manipulate stored information

Information is data

- How is it represented?

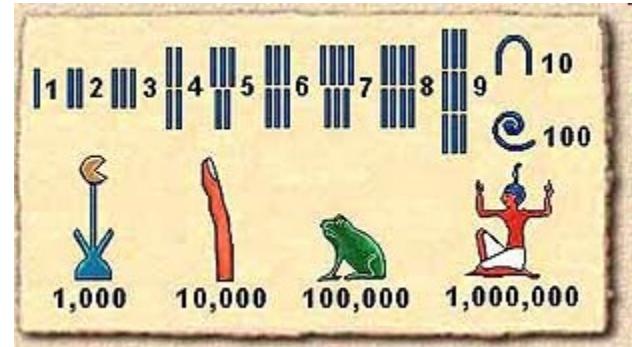
Basic information: numbers

Human beings have represented numbers throughout history

- Egyptian number system
- Roman numeral

Typically decimal

- Natural for humans



Discoveregypt.com

ROMAN NUMERALS	
I	VI
I	III I
II	VII
II	III II
III	VIII
III	III III
IV	IX
III	III III
V	X
III	III III
L	X
C	—
D	—
M	—
V	—
	XXXI
XI	XXXII
XII	XXXIII
XIV	XXXIV
XV	XXXV
XVI	XXXVI
XVII	XXXVII
XVIII	XXXVIII
XIX	XXXIX
XX	XL
XI	XLI
XII	XLII
XIII	XLIII
XIV	XLIV
XV	XLV
XVI	XLVI
XVII	XLVI
XVIII	XLVI
XIX	XLIX
XXX	L
	MMV
C	—
D	—
M	—
V	—
	MM
	MMIV
	MMIII
	MMII
	MMI
	MCMXCVI
	MCMXCVII
	MCMXCVIII
	MCMXCVIX

Number System

Comprises of

- Set of numbers or elements
- Operations on them
- Rules that define properties of operations

Need to assign value to numbers

Let us take decimal

$$123 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$$
$$\sum_{i=0}^n d_i \times 10^i$$

- Base 10
- Numbers are written as $d_n \dots d_2 d_1 d_0$
- Each digit is in [0-9]
- Value of a number is interpreted as $\sum_{i=0}^n d_i \times 10^i$

This is what summation
is saying

Binary Numbers

Base 2 = each digit is 0 or 1 *only 2 digits 0/1*

Numbers are written as $d_n \dots d_2 d_1 d_0$

Value of number is computed as $\sum_{i=0}^n d_i \times 2^i$

Binary representation is used in computers

- Easy to represent by switches (on/off)
- Manipulation by digital logic in hardware

Written form is long and inconvenient to deal with

Hexadecimal Numbers

Base 16

10₂
↑
binary

10₁₆
↑
hexadecimal

Each digit can be one of 16 different values

- Symbols = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

First 10 symbols (0 through 9) are same as decimal

- A=10, B=11, C=12, D=13, E=14, F=15

Numbers are written as $d_n \dots d_2 d_1 d_0$

$$\text{Value} = \sum_{i=0}^n d_i \times 16^i$$

010
b indicates binary
x indicates it is hexadecimal

0x0

2d binary
10 decimal

16 hexadecimal (or $\frac{10}{16^0}$)

A3 = a3 ✓
not case sensitive

$$A = 10 \cdot 16^1 + 3 \cdot 16^0 = 163$$

$$+ 3 \cdot 16^0 = 163$$

$$(60 + 3 = 163)$$

Octal Numbers

Base 8 = each digit is in [0-7]

Numbers are written as $d_n \dots d_2 d_1 d_0$

Value of number is computed as $\sum_{i=0}^n d_i \times 8^i$

103₈
216

$$= 1 \cdot 8^2 + 0 \cdot 8^1 + 3 \cdot 8^0$$

$$= 64 + 0 + 3$$

$$= 67$$

Converting Hex to Binary

1) $2/2 = 0$

6) $1/2 = 3$

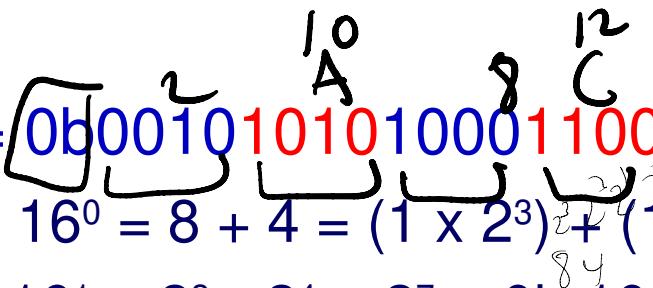
3) $1/2 = 1$

1) $1/2 = 0$

Each hexadecimal digit can be represented by 4 binary digits

- Why?

$0x2A8C$ (hex) = $0b0010101010001100$ (binary)



$$\blacksquare 0xC = 12 \times 16^0 = 8 + 4 = (1 \times 2^3) + (1 \times 2^2) = 0b1100$$

$$\blacksquare 0x80 = 8 \times 16^1 = 2^3 \times 2^4 = 2^7 = 0b10000000$$

- And so on ...

So, to convert hex to binary, just convert each digit and concatenate

What about octal to binary? $3b(?)$

103

0 01 0 0001

0x

Converting Binary to Hex

16 bit

Do the reverse

- Group each set of 4 digits and change to corresponding digit in hex
- Go from right to left

Example

1011011110011100

start from 15
00110011

■ 0b1011011110011100 = 0xB79C

What about binary to octal?

110 011
6 3₈

group by
threes

just add
zero

Decimal to Binary

What's the largest p, q, r ... such that
the result is the most significant part?

$$21/2 = 10 \quad r_1 = 1 \quad n = 2^p + r_1, \text{ where } r_1 < 2^p$$

$$10/2 = 5 \quad r_2 = 0 \quad n - 2^p = 2^q + r_2, \text{ where } r_2 < 2^q$$

$$5/2 = 2 \quad r_3 = 1 \quad n - (2^p + 2^q) = 2^r + r_3, \text{ where } r_3 < 2^r$$

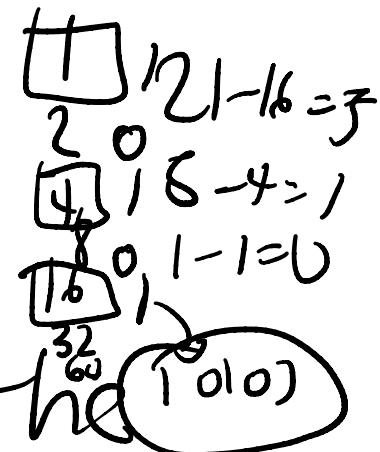
$$1/2 = 0 \quad r_4 = 1 \quad \dots \quad \text{most significant}$$

The above means that

- $n = (1 \times 2^p) + (1 \times 2^q) + (1 \times 2^r) + \dots + r_m$, where $r_m = n \% 2$
- Can you see why this now allows n to be easily written in binary form?

Example: convert 21 to binary

$$\blacksquare 21 = 2^4 + 5, 5 = 2^2 + 1 \quad 21 = 0b10101$$



Find the largest power of 2

$$\begin{aligned} 21/2 &= 10 \quad 1 \\ 10/2 &= 5 \quad 0 \\ 5/2 &= 2 \quad 1 \quad \text{0b}1010 \\ 2/2 &= 1 \quad 0 \\ 1/2 &= 0 \quad 1 \end{aligned}$$

Make a
chart

Decimal to Binary and Back

How to do the conversion algorithmically?

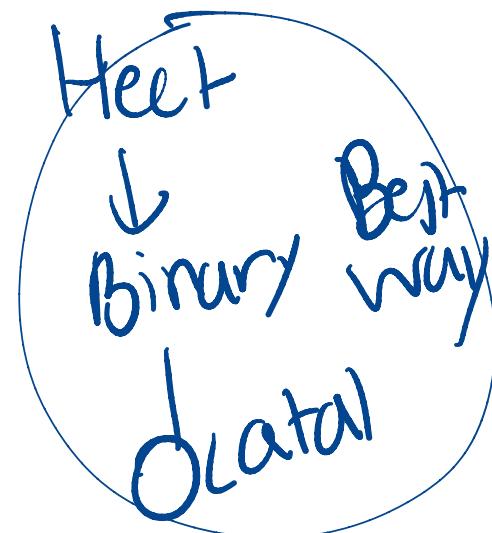
What about binary to decimal?

What about decimal to hex? Hex to decimal?

Decimal to octal? Octal to decimal?

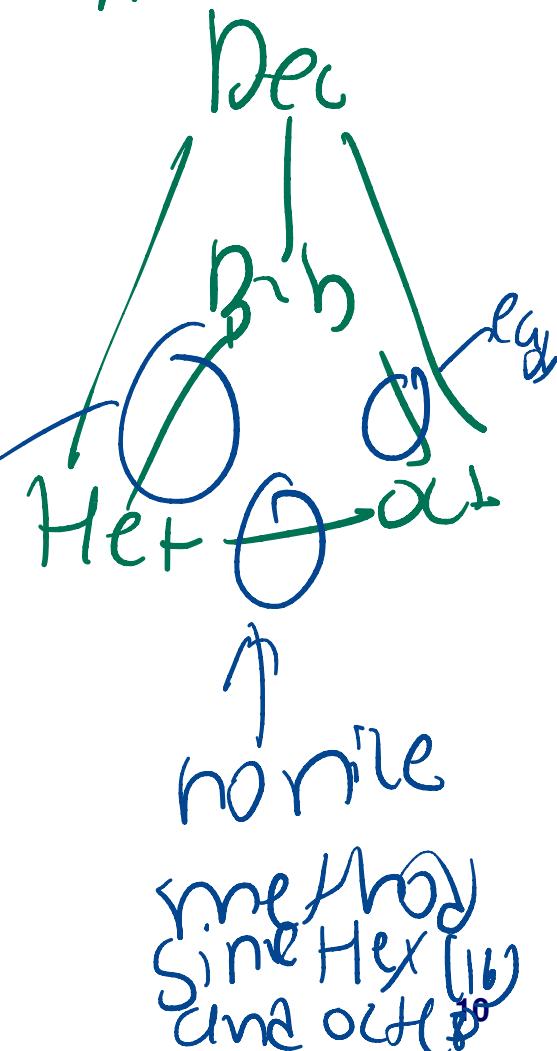
Hex to octal? Octal to Hex?

rocks



easy

Types of Conv.



Decimal and Binary fractions

23.57
10⁻¹ 10⁻² 10⁻³

In decimal, digits to the right of radix point have value $1/10^i$ for each digit in the i^{th} place

- 0.25 is $2/10 + 5/100$

Similarly, in binary, digits to the right of radix point have value $1/2^i$ for each i^{th} place

- Just the base is different

8.625 is 1000.101

- $.625 = 6/10 + 2/100 + 5/1000 = 1/2 + 1/8$

How to convert?

3 2 1 0 . -1 -2 -3

1000.101

8 4 2 1 2⁰ 2¹ 2² 2³

= 0.5 + 0.125

8.625

Decimal to Binary Example

Algorithm

```
Number = decimalFraction  
while (number > 0) {  
    number = number*2  
    if (number >=1) {  
        Output 1;  
        number = number-1  
    }  
    else {  
        Output 0  
    }  
}
```

turns a decimal to binary

Example: 0.625 to binary

$$\begin{array}{r} .75 \\ \times 2 \\ \hline 1.5 \\ \times 2 \\ \hline 0.5 \end{array}$$

ANS: 0.101

- $0.625 * 2 = 1.25 \rightarrow 1$
- output 1
- $0.25 * 2 = 0.5$
- output 0
- $0.5 * 2 = 1$
- output 1
- Exit

0.1
0.2 0
0.4 0
0.8 1
0.6
0.2 ↓

$$\begin{array}{r} 0.b_1 b_2 b_3 \\ \downarrow \\ b_1 b_2 b_3 \end{array}$$

Why does it work?



repetitive (really long)

How can we
represent
negative #?

Representing integers



How do we represent negative numbers in computers?

- Use a bit ... after all, that's how we store information, right?

Signed Magnitude: 

- $0100 = 4, 1100 = -4$
- $0011 = 3, 1011 = -3$

What is 1000?

- Have two zeros +0 (0000) and -0 (1000)
- As we shall see, inconvenient for arithmetic computations

Signed magnitude

Basically
we are
flipping

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

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

Signed magnitude

- What is $-1 + 1$?
- $101 + 1 = ?$
- $100?$

$$\begin{array}{r} 0(1) \\ + 1(0) \\ \hline 110 \end{array}$$

$\nearrow -1$

$$\begin{array}{r} 0(0) \\ + 0(0) \\ \hline 0 \end{array}$$

$\nearrow +2$

actual
work

$$\begin{array}{r} 0(1) (1) \\ 1(0) (-1) \\ \hline 110 -2 \end{array}$$

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

↑
not what
we wanted

One's Complement

Represent negative numbers by complementing positive numbers

Still have two zeros but arithmetic computation becomes easier

$$\begin{array}{r} 100 \\ \times 100 \\ \hline 000 \\ 000 \\ \hline 000 \end{array}$$

(table for binary and complement)

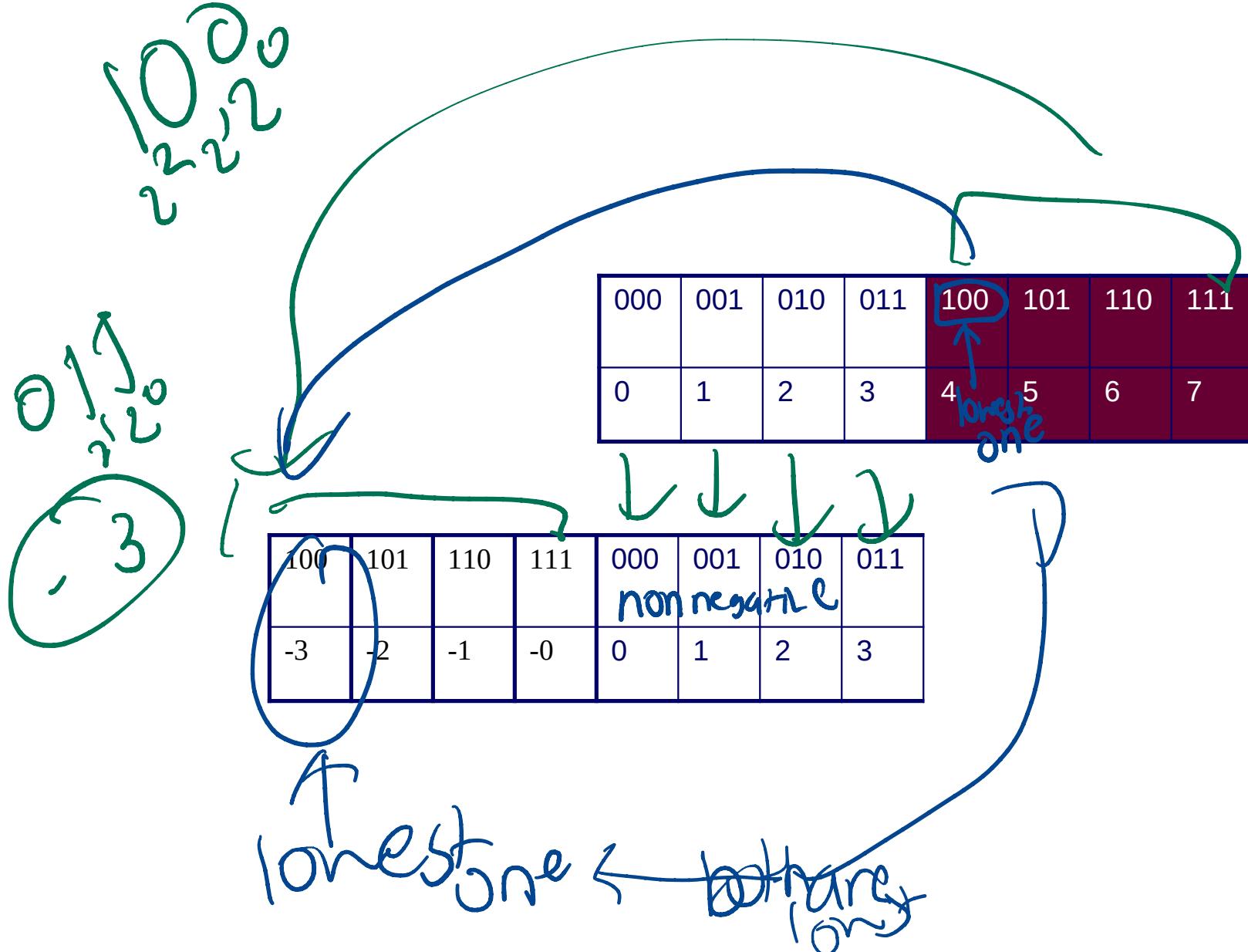
000	001	010	011	100	101	110	111
0	1	2	3	-3	-2	-1	-0

we complement

$$\begin{array}{r} 011 \\ \underline{\times} 100 \\ \hline 111 \end{array}$$

S6

One's complement



One's complement

- What is $-1 + 1$?
- $110 + 1 = 111$

$$\begin{array}{r} 110 \\ \underline{-01} \\ 111 \end{array} \Rightarrow 0$$

$$\begin{array}{r} 111 \\ \underline{-001} \\ \hline 1000 \end{array}$$

$$\begin{array}{r} 110 \\ \underline{-00} \\ 111 \end{array}$$

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

we want
to make
it bring up
we want to get rid of res
Cuz we don't want +/- 0

Two's Complement

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

100
22220

One's complement plus one

100
-4

- Most significant bit still gives the “sign”
- Trick: copy all ‘0’ bits from LSB till first ‘1’ bit. Copy ‘1’ bit, then flip all remaining bits till MSB.

Advantages:

- Only 1 zero
- Most convenient for arithmetic computations

Used in almost all computers today

Two's complement

One's complement
inverⁿer

$$\begin{array}{r} -3 \\ +1 \end{array} \rightarrow \begin{array}{r} 100 \\ 101 \end{array}$$

$$\begin{array}{r} 10 \\ +1 \\ \hline 110 \end{array}$$

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

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

$$\begin{array}{r} 010 \\ +1 \\ \hline \end{array}$$

$$\begin{array}{r} 011 \\ +1 \\ \hline 100 \end{array}$$

$$\begin{array}{r} 001 \\ 0 \\ 100 \end{array}$$

$$\begin{array}{r} 101 \\ 010 \\ \hline 011 \end{array}$$

0111

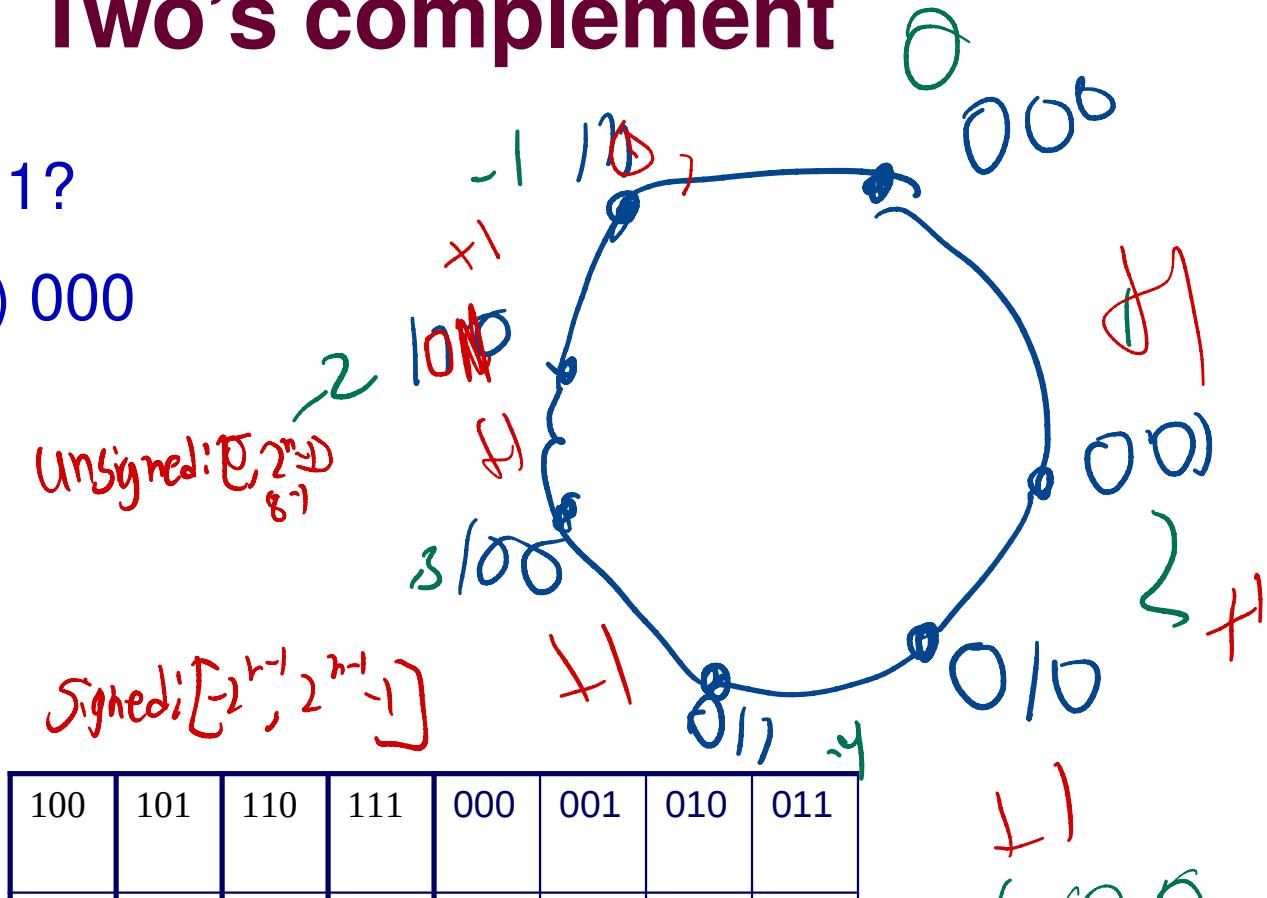
+1

1000

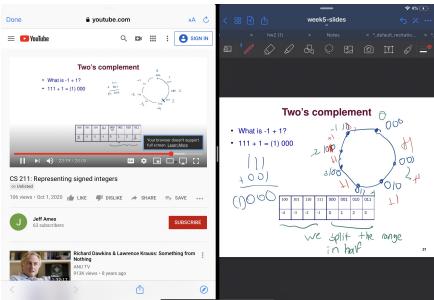
Two's complement

- What is $-1 + 1$?
 - $111 + 1 = (1) \ 000$

$$\begin{array}{r} 111 \\ + 00 \\ \hline 100 \end{array}$$



we split the range
in half



Two's complement

- To go from 3 to -3, flip all bits and add 1
- (or change rightmost 0 to 1, and all 1's to its right to 0's)
- How to go from -3 to 3?

$101 \rightarrow 010$

$$\begin{array}{r} 101 \\ 010 \\ + \quad \quad \quad \\ \hline 011 \end{array}$$

$010 + 011 = 101$

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

→ reverse it complement it

- add 1
- done

Done youtube.com

YouTube

Numerical Value of Two's Complement

Given a two's complement number of length n , written as $d_{n-1}d_{n-2}\dots d_0$

- Its value is interpreted as $d_{n-1}2^{n-1} + \sum_{i=0}^{n-2} d_i 2^i$

The range of values is then $[-2^{n-1}, 2^{n-1} - 1]$

- More negative numbers than positive (if we do not count 0)
- $101 = ?$
- $0101 = 101$

Your browser doesn't support this video.

CS 211: Representing signed integers

Unlisted

106 views · Oct 1, 2020

Jeff Ames

63 subscribers

Richard Dawkins & Lawrence Krauss: Something Nothing

ANU TV

913K views · 8 years ago

Numerical Value of Two's Complement

lost it + this again

Given a two's complement number of length n, written as $\underline{3/2} = 1$

$d_{n-1} \dots d_1 d_0$

- Its value is interpreted as

$$-8 + 5 = -3 \quad \text{map 3} \quad \text{confirm}$$

$$1101 \rightarrow 0010+1\cdot 3$$

1101
3210

$$2^0 + 2^3 + 1 = 5$$

The range of values is then

$$[-2^{n-1}, 2^{n-1}]$$

$$h^1[n-1] b^{[n-1]}$$

$$-2^2 + 7 = \underline{?}$$

- More negative numbers than positive (if we do not count 0)

$$\sim 8 + 3y$$

10

oak

100

$$x_1 + 2x_0 + x_4 \\ y + 5 \\ -8 + 5 =$$

0 0 0
0 0 0

10

3 OC₂-3)

111
111

Mo Content (x)	T.S. (kg/cm²)	Y.S. (kg/cm²)	Elong. (%)	H.R.B. (kg/mm²)
0.5	300	200	10	100
1.0	350	220	12	120
1.5	380	240	14	140
2.0	400	260	16	160
2.5	420	280	18	180
3.0	440	300	20	200
3.5	460	320	22	220
4.0	480	340	24	240
4.5	500	360	26	260
5.0	520	380	28	280
5.5	540	400	30	300
6.0	560	420	32	320
6.5	580	440	34	340
7.0	600	460	36	360
7.5	620	480	38	380
8.0	640	500	40	400
8.5	660	520	42	420
9.0	680	540	44	440
9.5	700	560	46	460
10.0	720	580	48	480

101
222
2
~~4x4~~
+ 2x4

101
1
2
3
2
2
3
1

101
222
2
~~4x4~~
+ 2x4

ASCII

A character is stored as 1 byte according to the ASCII standard

Originally used only 128 values (7 bits)

- One bit could be used for error detection (will discuss later)

Subsequently extended to use all 256 values

Languages like Chinese need more

original/earlier implementation
represent ~~one byte~~ hello

ASCII table

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	'	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	:	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

0x68

turns
string
 $16 \times 6 + 8$

0x65

0 1 1 0 0 1 00
6 8

hex to binary

$15/2 = 7\ 1$

$7/2 = 3\ 1$

$3/2 = 1\ 1$

$1/2 = 0\ 1$

Expanded version
(that's 13 characters)

$6/2 = 3\ 0$

$3/2 = 1\ 1$

$1/2 = 0\ 1$

Unicode and UTF-8

What about characters for other languages?

~~Unicode~~ *allows to use multiple languages*
~~Unicode is a standard that defines more than 107,000 characters across 90 scripts (and more ...)~~

Unicode can be implemented by different character encodings

Most common: UTF-8

- Variable length encoding of Unicode: 1-4 bytes for each character
- 1-byte form is reserved for ASCII for backward compatibility

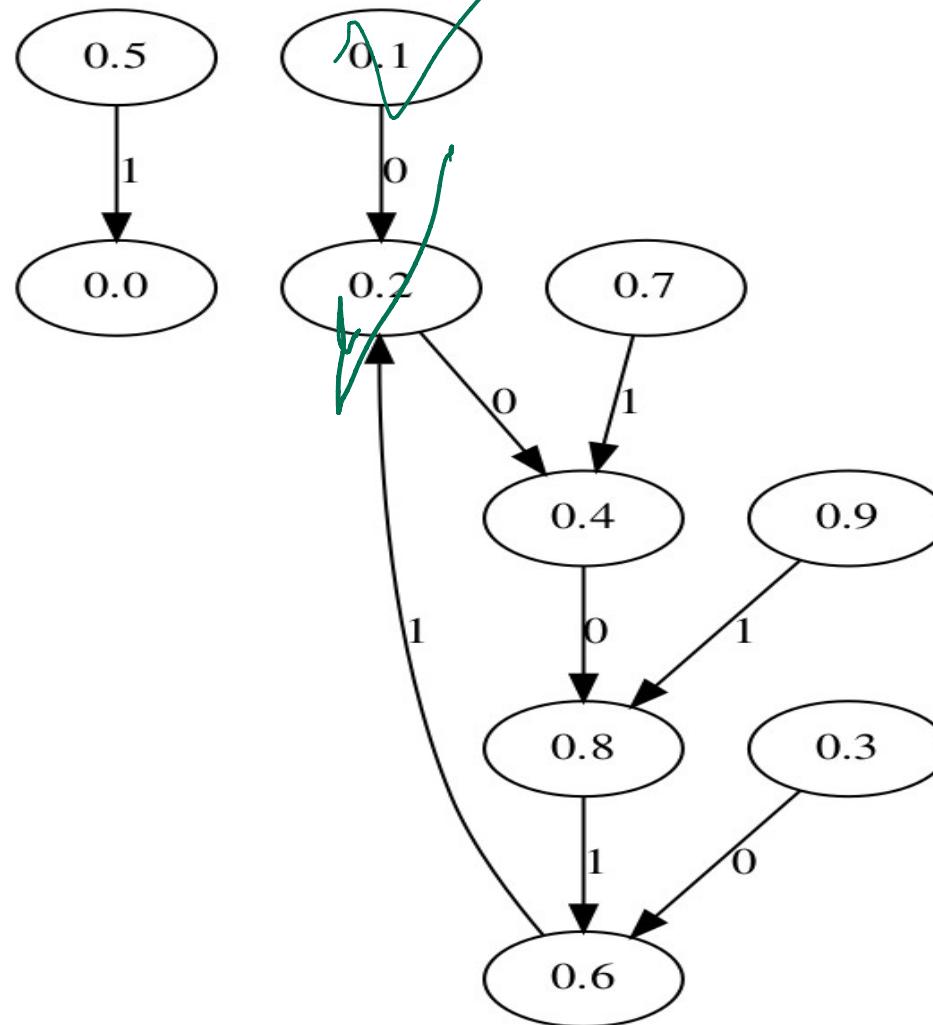
h e l l o

chars ≠ #bytes
Char
wchar_t

Floating point

- Recall that 0.1 can't be represented exactly in binary:
- $0.1 * 2 = 0.2$ (0)
- $0.2 * 2 = 0.4$ (0)
- $0.4 * 2 = 0.8$ (0)
- $0.8 * 2 = 1.6$ (1)
- $0.6 * 2 = 1.2$ (1)
- $0.2 * 2 = 0.4$ (0) ...
- What about 0.2, 0.3, 0.4, ...?

Floating point



Only 0.0 and 0.5 are exact! (for single digit fractional values)

But others multi-digit fractionals may also be, e.g., 0.125.

Floating point comparison

- float f1 = ...; $0.1 + 0.1 + 0.1$

$0.1 + 0.1 + 0.1$

- float f2 = ...; 0.3

0.3

- if ($f1 == f2$) ... // dangerous!
↑ look at the literal binary values (they are approximates) // slight rounding error

- We actually want to ask if they're really similar:

$|f1 - f2| < \epsilon$ // are they close enough

- if ($\text{fabs}(f1 - f2) < 1e-6$) ...

threshold

Safer way of comparing two floating point numbers

Floating point

Integers typically written in ordinary decimal form

- E.g., 1, 10, 100, 1000, 10000, 12456897, etc.

0.12456897×10^7

But, can also be written in scientific notation

- E.g., 1×10^4 , 1.2456897×10^7

What about binary numbers?

- Works the same way
- $0b100 = 0b1 \times 2^2$

Scientific notation gives a natural way for thinking about floating point numbers

- $0.25 = 2.5 \times 10^{-1} = 0b1 \times 2^{-2}$

How to represent in computers?

$$123 = 1.23 \times 10^2$$

Can be written
in scientific
notation

$$1100_2 = 1.1 \cdot 2^3$$

five more from
frac place

$$0.11$$

\downarrow

$$1.1 \times 10^{-1}$$

$$1.23$$

IEEE floating point standard

most computers follow it

Most computers follow IEEE 754 standard

float datatype
Single precision (32 bits) float
double datatype
Double precision (64 bits) double

Extended precision (80 bits)

S	Exponent	Fraction
---	----------	----------

$$(m) \cdot 2^e$$

Floating point in C

(Single)

32 bits single precision (type float)

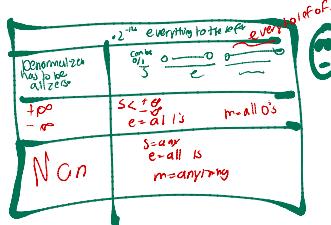
- 1 bit for sign, 8 bits for exponent, 23 bits for mantissa
 - Sign bit: 1 = negative numbers, 0 = positive numbers
 - Exponent is power of 2
- Have 2 zero's *(zero)*
- Range is approximately -10^{38} to 10^{38}

m | e | e

we can divide it in three part

64 bits double precision (type double) *instead of 8 bit*

- 1 bit for sign, 11 bits for exponent, 52 bits for mantissa
- Majority of new bits for mantissa → higher precision
- Range is -10^{308} to $+10^{308}$



Numerical Values

Three different cases:

- Normalized values

Normalized Case

- exponent field $\neq 0$ and exponent field $\neq 2^k-1$ (all 1's)
- exponent = binary value – Bias
 - » Bias = $2^{k-1}-1$ (e.g., 127 for float)
- mantissa = 1.(mantissa field)
- Ex: (sign: 0, exp: 1, mantissa: 1) would give $0b1.1 \times 2^{-126}$

- Denormalized values

another type of value

- exponent field = 0
- exponent = $1 - \text{Bias}$ (e.g., -126 for float)
- Mantissa = mantissa field (no leading 1)
- Ex: (sign: 0, exp: 0, mantissa: 10) would give $0b10 \times 2^{-126}$

- Special values: represent $+\infty$, $-\infty$, and NaN



0.0 how can we represent it with floating point



\rightarrow sign is all ones
ex all 1's
m= all zero

$S < 0$
 $e = \text{all } 1's$
 $m = \text{all zero}$

$S = \text{any } e = \text{all } 1's$
 $m = \text{any } e = \text{all } 1's$

$[0]^{-126} = 2^{-126}$
except 33
all zero

Decimal to IEEE Floating Point

$$0.5 + 0.121729 / 5.625$$

In binary

$$101_2 \cdot 101_2 = 1.01101 \times 2^6$$

Exponent field has value 2

- add 127 to get 129

Exponent is 10000001

Mantissa is 01101

Sign bit is 0

$\begin{array}{r} 01000000 \\ \times 01101000 \\ \hline 00000000 \end{array}$

$$\begin{array}{r} 64 \\ \hline 128 \\ - 64 \\ \hline 64 \end{array}$$

$$|b|/2 = 80$$

8/23/0

$$4/2 = 20$$

$$2/2=1$$

$$1/2 = 0$$

A horizontal row of six orange circles. Each circle is connected to its neighbors by red lines forming a chain. From the top of each circle, a green line extends vertically upwards.

One more example

Convert 12.375 to floating point representation

Binary is 1100.011

$$1.100011 \times 2^3$$

$$\text{Exponent} = 127 + 3 = 130 = 0b\boxed{10000010}$$

is 130

Mantissa = 100011

Sign = 0

0
—
sign

10000010

100011 b1nct0f2ew

Extended precision

80 bits used to represent a real number

1 sign bit, 15 bit exponent, 64 bit mantissa

20 decimal digits of accuracy

10^{-4932} to 10^{4932}

Not supported in C