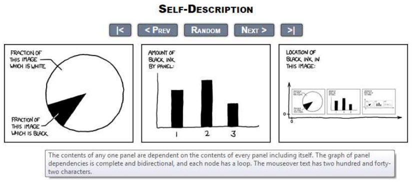
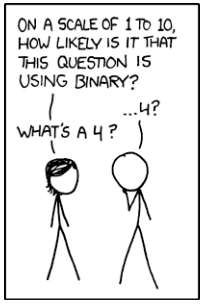

Self-Description
|<
< Prev
Random
Next >
>|
FRACTION OF THIS IMAGE WHICH IS WHITE
FRACTION OF THIS IMAGE WHICH IS BLACK
AMOUNT OF BLACK INK BY PANEL:
1
2
3
LOCATION OF BLACK INK IN THIS IMAGE:
0
0

The contents of any one panel are dependent on the contents of every panel including itself. The graph of panel dependencies is complete and bidirectional, and each node has a loop. The mouseover text has two hundred and forty-two characters.

CS 101 Today

There are only 10 types of people in the world: Those who understand binary and those who don't.


ON A SCALE OF 1 TO 10, HOW LIKELY IS IT THAT THIS QUESTION IS USING BINARY?
WHAT'S A 4 ?
...4?

My top-10 list of binary jokes...

Read Sections
4.1 to 4.2.2

Looking Back

Computing as composition

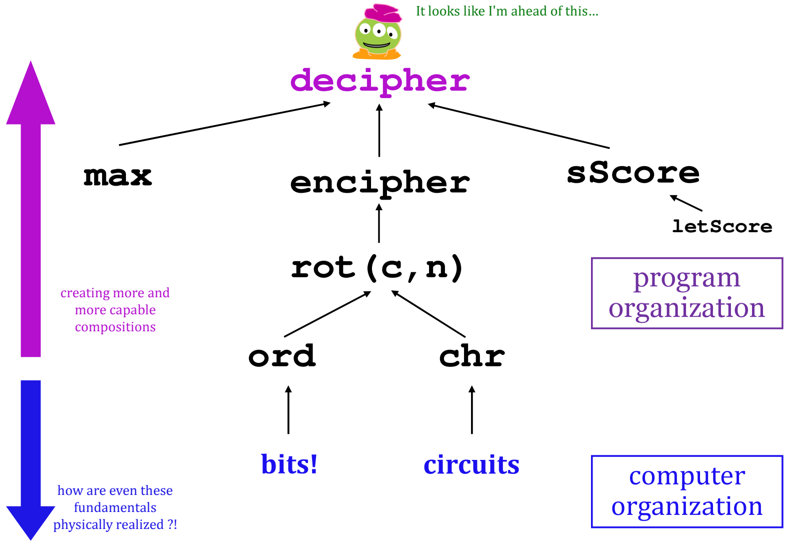
***clay* == functions**

Looking Forward

Computing as representation

clay* == data & *bits

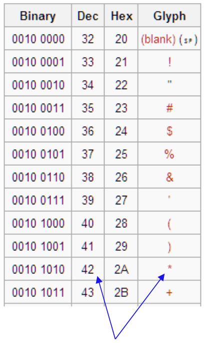
Some legs to stand on!


It looks like I'm ahead of this...
decipher
max
encipher
sScore
letScore
rot(c,n)
ord
chr
bits!
circuits
creating more and more capable compositions
how are even these fundamentals physically realized ?!

program organization

computer organization

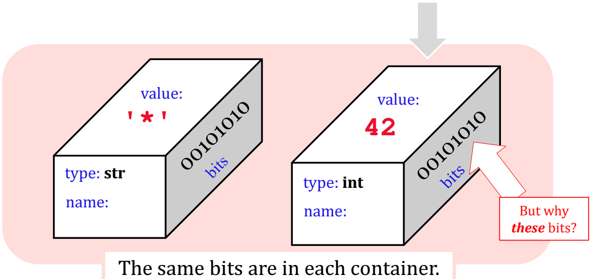
Binary Storage & Representation



Binary	Dec	Hex	Glyph
0010 0000	32	20	(blank) (SP)
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+

The SAME bits can represent different pieces of data, depending on **type**

8 bits = 1 byte = 1 box



The same bits are in each container.

Base 2

each column represents the base's next power

Base 10

THIRTYTWOs col. SIXTEENs col. EIGHTs column FOURs column TWOs column ONEs column

101010

TENs column ONEs column

42

4 tens + 2 ones

128's column SIXTYFOURs col THIRTYTWOs col. SIXTEENs col. EIGHTs column FOURs column TWOs column ONEs column

_ _ _ _ _ _ _ _

Write 123 in binary...

123

HUNDREDs column TENs column ONEs column

123

1 hundred + 2 tens + 3 ones

Beyond Binary

base 1		111111111111111111111111111111111111111111	digits: 1
base 2	32 16 8 4 2 1	101010	digits: 0, 1
base 3	27 9 3 1	1120	digits: 0, 1, 2
base 4	16 4 1		
base 5			
base 6			
base 7			
base 8			
base 9			
base 10		42	digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
base 11			
base 12			
. . .			
base 16			

Which of these *isn't* 42...?

222 60 54 46 39

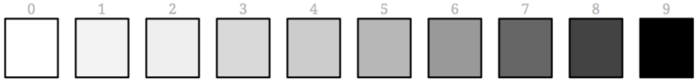
and what are the *bases* of the rest?

All 42s!

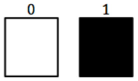
base 1		111111111111111111111111111111111111111111	digits: 1
base 2	128 64 32 16 8 4 2 1	101010	digits: 0, 1
base 3	81 27 9 3 1	1120	digits: 0, 1, 2
base 4	64 16 4 1	222	digits: 0, 1, 2, 3
base 5	125 25 5 1	132	digits: 0, 1, 2, 3, 4
base 6	216 36 6 1	110	digits: 0, 1, 2, 3, 4, 5
base 7	49 7 1	60	digits: 0, 1, 2, 3, 4, 5, 6
base 8	64 8 1	52	digits: 0, 1, 2, 3, 4, 5, 6, 7
base 9	81 9 1	46	digits: 0, 1, 2, 3, 4, 5, 6, 7, 8
base 10	100 10 1	42	digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
base 11	121 11 1	39	digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A
base 16	256 16 1	2A	Hexadecimal digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Two symbols is easiest!

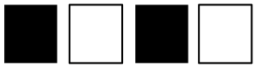
A computer has to differentiate *physically* among all its possibilities.



ten symbols ~ ten different voltages



two symbols ~ two different voltages



What digits are these?

Easy!

Extra! Can you figure out the **last binary digit** (bit) of **53** *without determining any other bits*? The last ***two***? ***3***? All?

...	**32s**	**16s**	**8s**	**4s**	**2s**	**1s**
...	16s	8s	4s	2s	1s	
...	8s	4s	2s	1s		
...	4s	2s	1s			
	2s	1s				

top-level reality!

____ ____ ____ ____ ____ ____

53

in the end,
we need
"53"-worth
of value

Lab 4: Computing in binary

base 10		base 2
100 10 1 141	=	128 64 32 16 8 4 2 1 '10001101'

```
def numToBinary(N):
    if N == 0:
        return ''
    elif N%2 == 0:
        return numToBinary(   ) +
    else:
        return numToBinary(   ) +
```

empty string means 0

?

If N is even, what is the final bit?

If N is odd, what is the final bit?

How much VALUE is left to convert!?

Reasoning, *bit by bit*

<< left-shift

>> right-shift

Python did always look a little shifty to me...

left-shift by 1

```
  11
 110
```

3 << 1

6

left-shift by 2

```
  11
1100
```

3 << 2

12

What does ***left-shifting*** do to the **value** of a #?

right-shift by 2

```
101010
  1010
```

42 >> 2

10

What does ***right-shifting*** do to the **value** of a #?

Reasoning, *bit by bit*

<< left-shift

>> right-shift

& and

| or

and (both) & | or (either)

bitwise and

```
5:  101
6:  110
&   100
```

5 & 6

4

bitwise and

```
11: 1011
 5: 0101
&
```

11 & 5

bitwise or

```
5:  101
6:  110
|   111
```

5 | 6

7

bitwise or

```
11: 1011
 5: 0101
|
```

11 | 5

Being bit-*wise*

Try these for a bit...

You **don't** need to convert to binary for these three...

7 << 1

left-shift

5 << 4

170 >> 2

right-shift

```
14:  1110
 9:  1001
```

You **do** need to use binary for these two!

14 | 9

or

14 & 9

and

In processors shifts, ands, ors, adds, and subtractions are ***very fast***, whereas multiplying, dividing, and mod are relatively **slow**.

Given this, what is a way to compute these expressions using *only* ***fast*** operations, maybe in combination?

N//4

N*7

N*17

N%16

extra sneaky!

Intel x86 processor instructions
and their speeds (2014)

In processors shift, and, or, add, and subtract are ***much faster*** than multiply, divide, and mod, which are ***relatively* slow**.

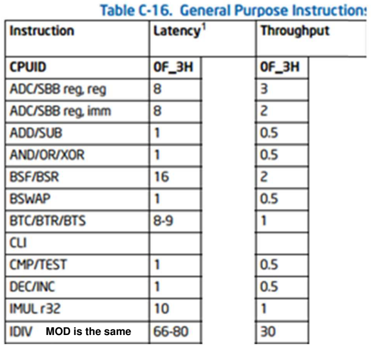


Table C-16. General Purpose Instruction:

Instruction	Latency[1]	Throughput
CPUID	0F_3H	0F_3H
ADC/SBB reg, reg	8	3
ADC/SBB reg, imm	8	2
ADD/SUB	1	0.5
AND/OR/XOR	1	0.5
BSF/BSR	16	2
BSWAP	1	0.5
BTC/BTR/BTS	8-9	1
CLI		
CMP/TEST	1	0.5
DEC/INC	1	0.5
IMUL r32	10	1
IDIV MOD is the same	66-80	30

Given this, what is a way to compute these statements using combinations from only the ***fast*** operations above?

`N/4` ⇨ `N >> 2`

`N*7` ⇨

`N*17` ⇨

`N%16` ⇨

Name(s): ______________________________

Quiz

In binary, I'm an 11-eyed alien!

Convert these two binary numbers ***to decimal***:

```
32  16  8   4   2   1
1   1   0   0   1   1          10001000
```

Convert these two decimal numbers ***to binary***:

```
32  16  8   4   2   1
```

28_{10} 101_{10}

Add these two binary numbers:

```
   101101
+    1110
---------
```

Multiply these binary numbers:

```
   101101
*    1110
---------



+
---------
```

WITHOUT converting to decimal !

```
    1
  529
+ 742
-----
 1271
```

Hint: Remember these algorithms? They're the same in binary!

```
   529
*   42
------
  1058
+ 2116
------
 22218
```

Extra! Can you figure out the last binary digit (bit) of 53 ***without determining any other bits?*** The last two? 3?