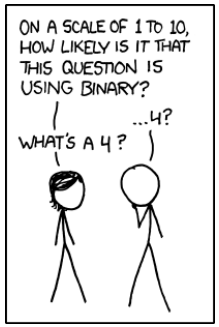
# CS 101 Today...

Our top-10 list of binary jokes...






← **Looking Back**

Computing as composition

***clay* == functions**

**Looking Forward** →

Computing as representation

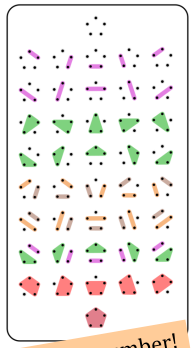
***clay* == data & *bits***



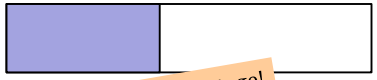
# Some 42's! *Which are fundamental?*

```
len("Reveal the answer to the ultimate question")
```


$42_{10}$ $101010_{2}$



5th Catalan number!



Percentage!

b2n n2b

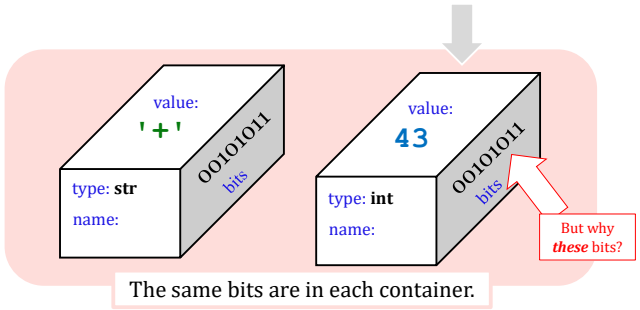


# Base-2 Storage & Representation

| Binary | Dec | Hex | Glyph |
|---|---|---|---|
| 0010 0000 | 32 | 20 | (blank) (␣) |
| 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




chr ord



**forty-two**
value

**42**
syntax



tens ones

## Base 2
"binary"

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

$101010_2$

## Base 10
"decimal"

TENs column ONEs column

$42_{10}$

4 tens + 2 ones

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

each column represents the base's next power

HUNDREDs column TENs column ONEs column

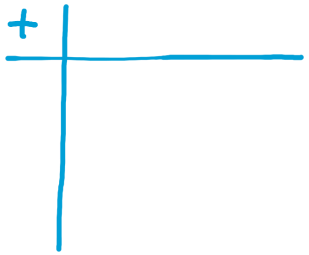
$123_{10}$

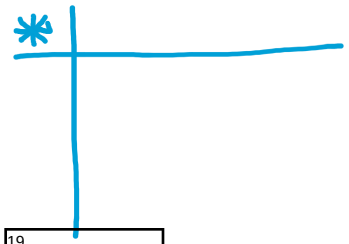
1 hundred + 2 tens + 3 ones



writing 123 in binary...

## Binary math







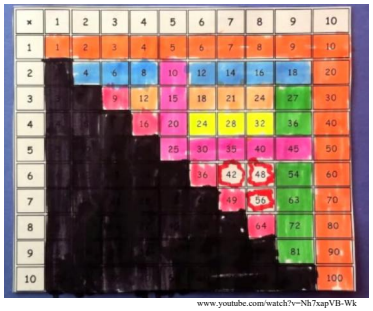
tables of one-digit facts

\+ Addition

* Multiplication

## Decimal math

| + | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 3 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 4 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 5 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 6 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 7 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 8 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 9 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |



www.youtube.com/watch?v=Nh7xapVB-Wk

## *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**

16 4 1

**base 4**
**base 5**
**base 6**
**base 7**
**base 8**
**base 9**
**base 10**
**base 11**
**base 12**
. . .
**base 16**

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

**222** **60** **54** **46** **39**

and what are the ***bases*** of the rest?

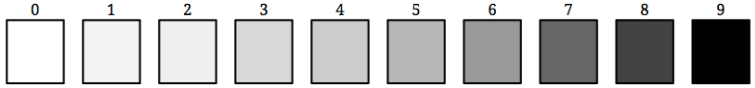
**42** digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**



# But ***why*** base-2?

# Could computer circuits represent decimals?

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




ten symbols ~ ten different voltages

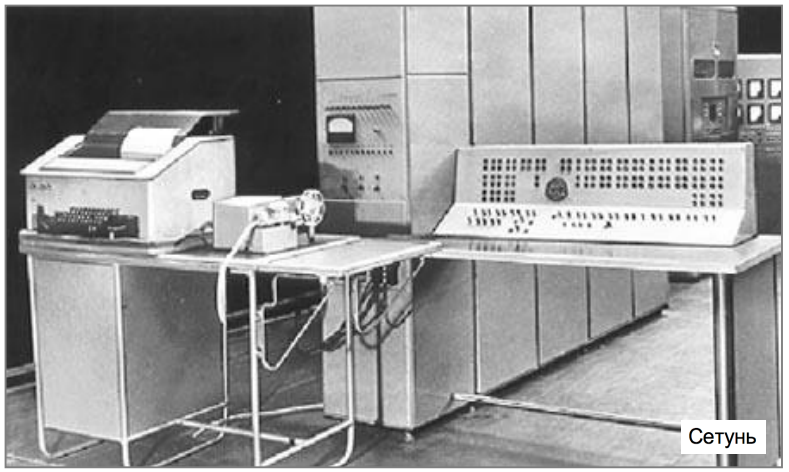
*What digits are these?*



# Ternary computers?

*Everything* should be base-3!

50 of these ***Setun*** ternary machines were made at Moscow U. ~ **1958**

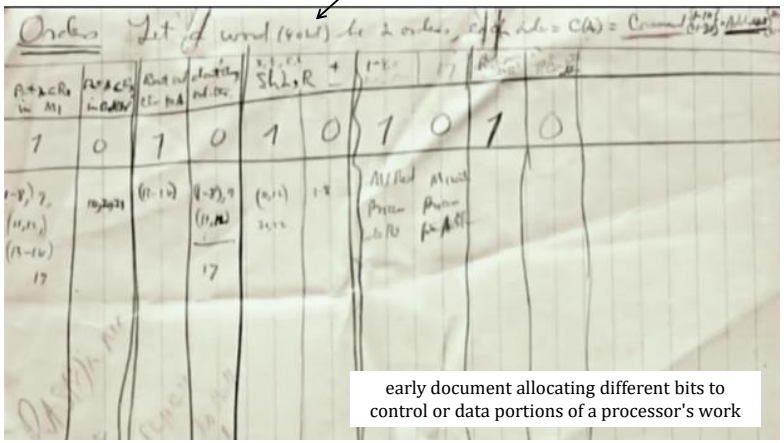



This project was discontinued in 1970... ***though not because of the ternary design!***



b.d. ~ binary digit ~ **bit**

"bit" first appeared in print in 1948 (Claude Shannon)



early document allocating different bits to control or data portions of a processor's work

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

All of them?



# Lab 4: *Computing to base-2*

| base 10 | base 2 |
|---|---|
| 100 10 1<br>53 | … 4 2 1<br>= |

This first step of **left-to-right** conversion into binary is tricky to program... ***Why?***

You mean ***aside*** from having to think in binary?

53

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

| ... | 32s | 16s | 8s | 4s | 2s | 1s |
|---|---|---|---|---|---|---|
| ... | 16s | 8s | 4s | 2s | 1s | |
| ... | 8s | 4s | 2s | 1s | | |
| ... | 4s | 2s | 1s | | | |
| ... | 2s | 1s | | | | |

top-level reality!

"next"-level reality...

__ __ __ __ __ __ bits!

53 value remaining

Converting to binary ~ starting from the right!



# Reasoning ~ *Value* vs. *Syntax*

53 → 530

What does ***left-shifting*** do to the **value** of a <u>decimal</u> #?

537 → 53

What does ***right-shifting*** do to the **value** of a <u>decimal</u> #?

<< left-shift

>> right-shift



*bitwise* Python: the left and right "shift operators"

# Reasoning, *bit by bit*

<< left-shift

>> right-shift

left-shift by 1

```
11
110
```

```
3 << 1
6
```

What does ***left-shifting*** do to the **value** of a <u>binary</u> #?

left-shift by 2

```
11
1100
```

```
3 << 2
12
```

right-shift by 1

```
101010
10101
```

```
42 >> 1
21
```

What does ***right-shifting*** do to the **value** of a <u>binary</u> #?

 1010 `42 >> 2` ?

Intel x86 processor instructions and their speeds

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

Given this, how can we compute these slow statements using ***fast*** operations?

Table C-15. General Purpose Instructions

| Instruction | Latency[1] | | Throughput | |
|---|---|---|---|---|
| CPUID | 0F_3H | 0F_2H | 0F_3H | 0F_2H |
| ADC/SBB reg, reg | 8 | 8 | 3 | 3 |
| ADC/SBB reg, imm | 8 | 6 | 2 | 2 |
| ADD/SUB | 1 | 0.5 | 0.5 | 0.5 |
| AND/OR/XOR | 1 | 0.5 | 0.5 | 0.5 |
| BSF/BSR | 16 | 8 | 2 | 4 |
| BSWAP | 1 | 7 | 0.5 | 1 |
| BTC/BTR/BTS | 8-9 | | 1 | |
| CLI | | | | 26 |
| CMP/TEST | 1 | 0.5 | 0.5 | 0.5 |
| DEC/INC | 1 | 1 | 0.5 | 0.5 |
| IMUL r32 | 10 | 14 | 1 | 3 |
| IMUL imm32 | | 14 | 1 | 3 |
| IMUL | | 15-18 | | 5 |
| IDIV + MOD | 66-80 | 56-70 | 30 | 23 |

`N//4` `N*17`

`N*7` `N%16`



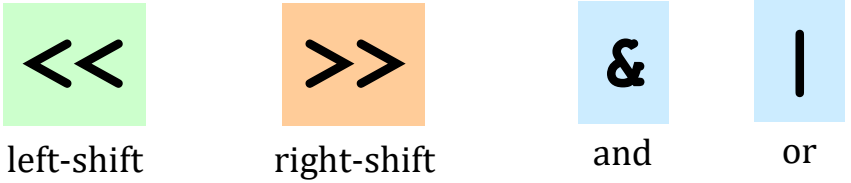
A `(N<<3) - N`

B `N-((N>>4)<<4)`

C `N >> 2`

D `(N<<4) + N`

# Reasoning, *bit by bit*




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

(blank)

Name(s) _____

# *Quiz*

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

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

32 16 8 4 2 1

110011 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
--------
```

**WITHOUT** converting to decimal !

***Multiply*** these binary numbers:

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

```
   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?