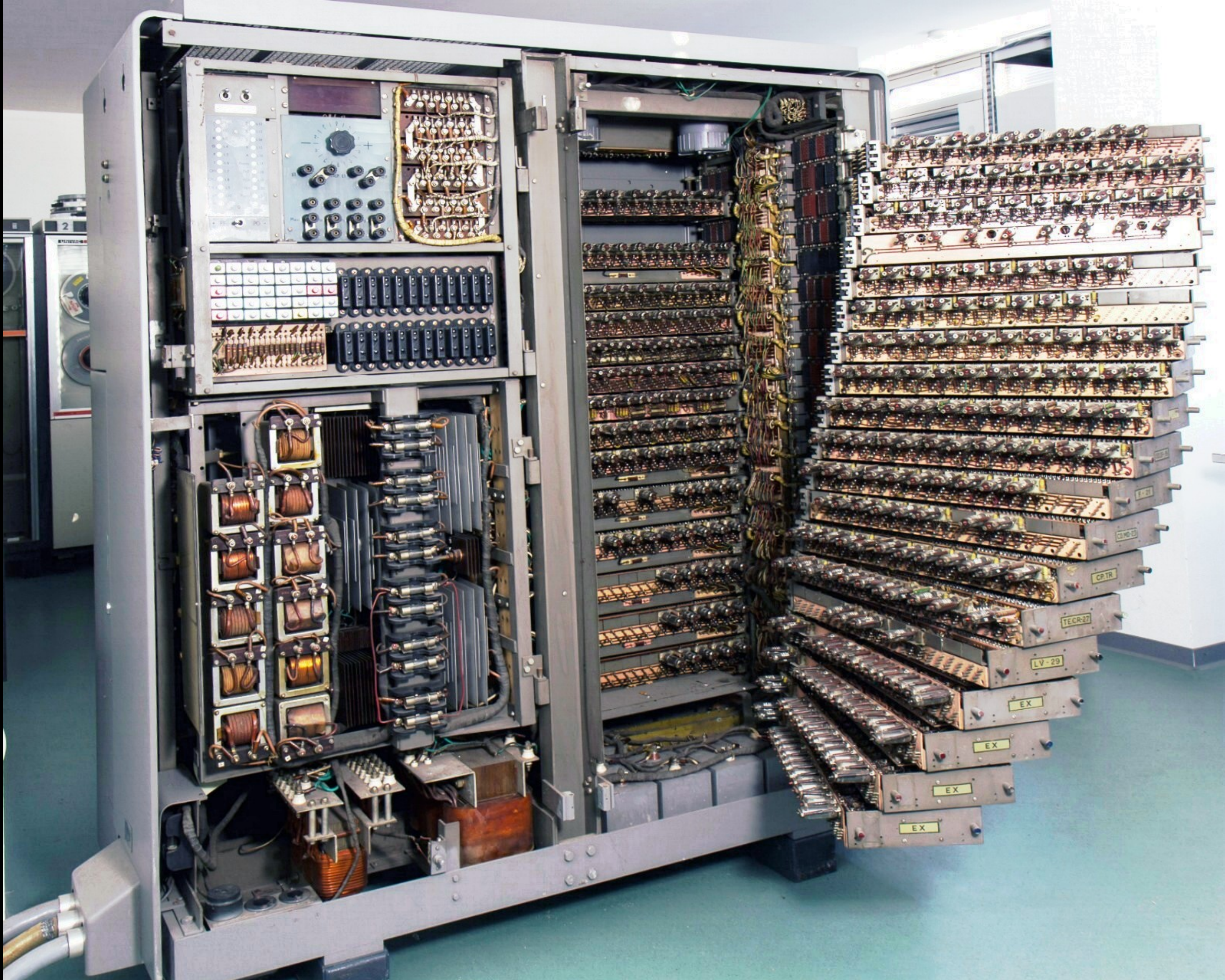
Hoon School Live · Lesson -1

Introduction to Computing



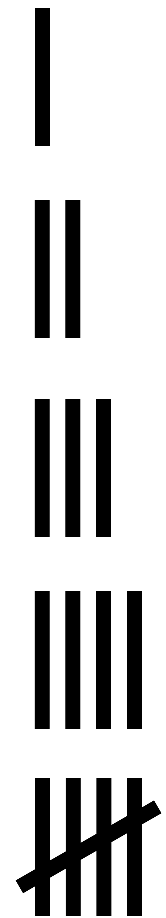
"Once, men turned their thinking over to machines in the hope that this would set them free. But that only permitted other men with machines to enslave them."

"'Thou shalt not make a machine in the likeness of a man's mind,'" Paul quoted.

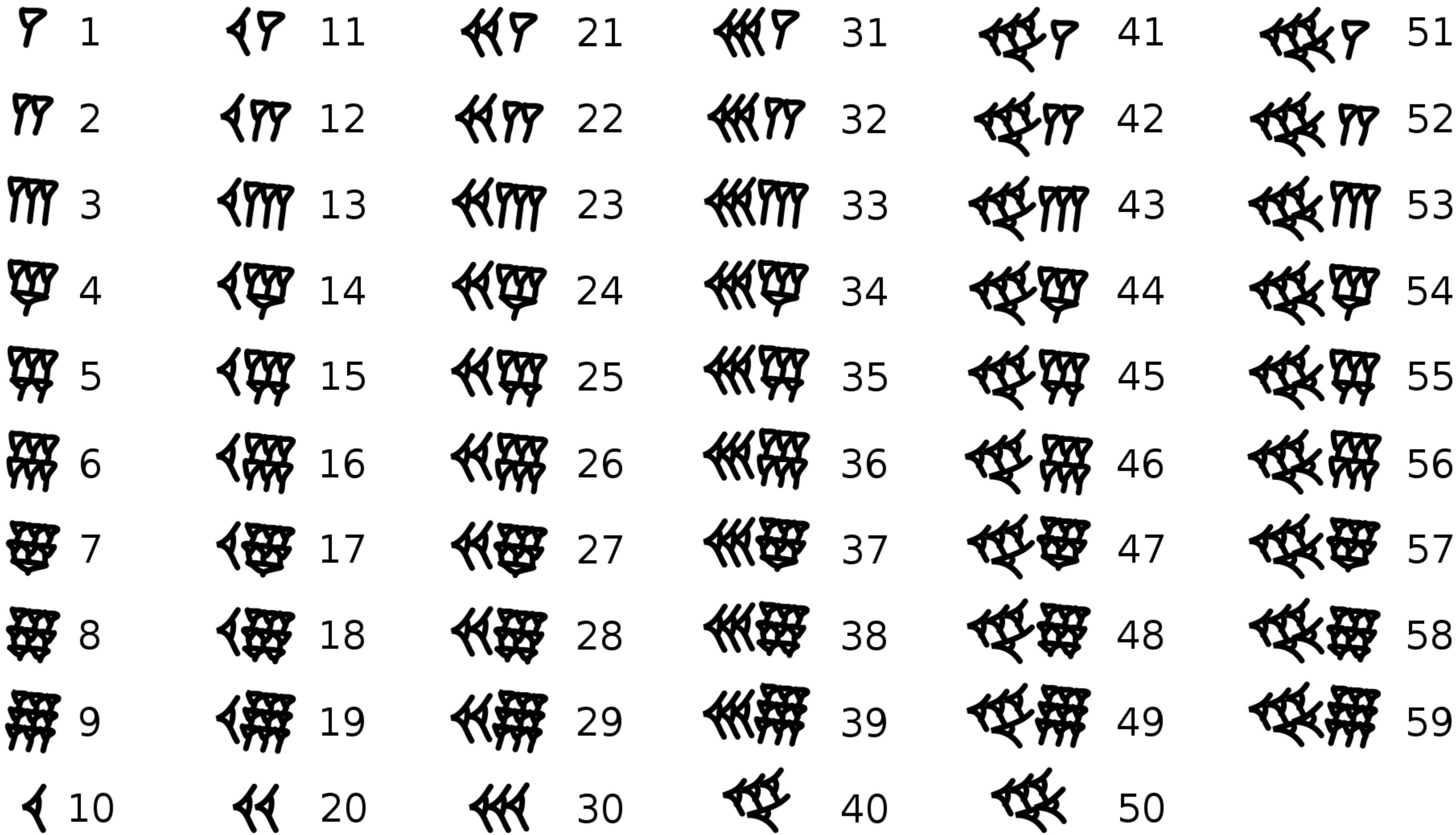
"Right out of the Butlerian Jihad and the Orange Catholic Bible," she said. "But what the O.C. Bible should've said is: 'Thou shalt not make a machine to counterfeit a *human* mind.' Have you studied the Mentat in your service?"

Computational Principles

- Representations are interchangeable.
- Any computation requires a representation.
- Computation requires *values*, *operations*, and *state*.
- All of these can be represented unambiguously on the machine.
- Higher-level languages map to lower-level representations for the machine.

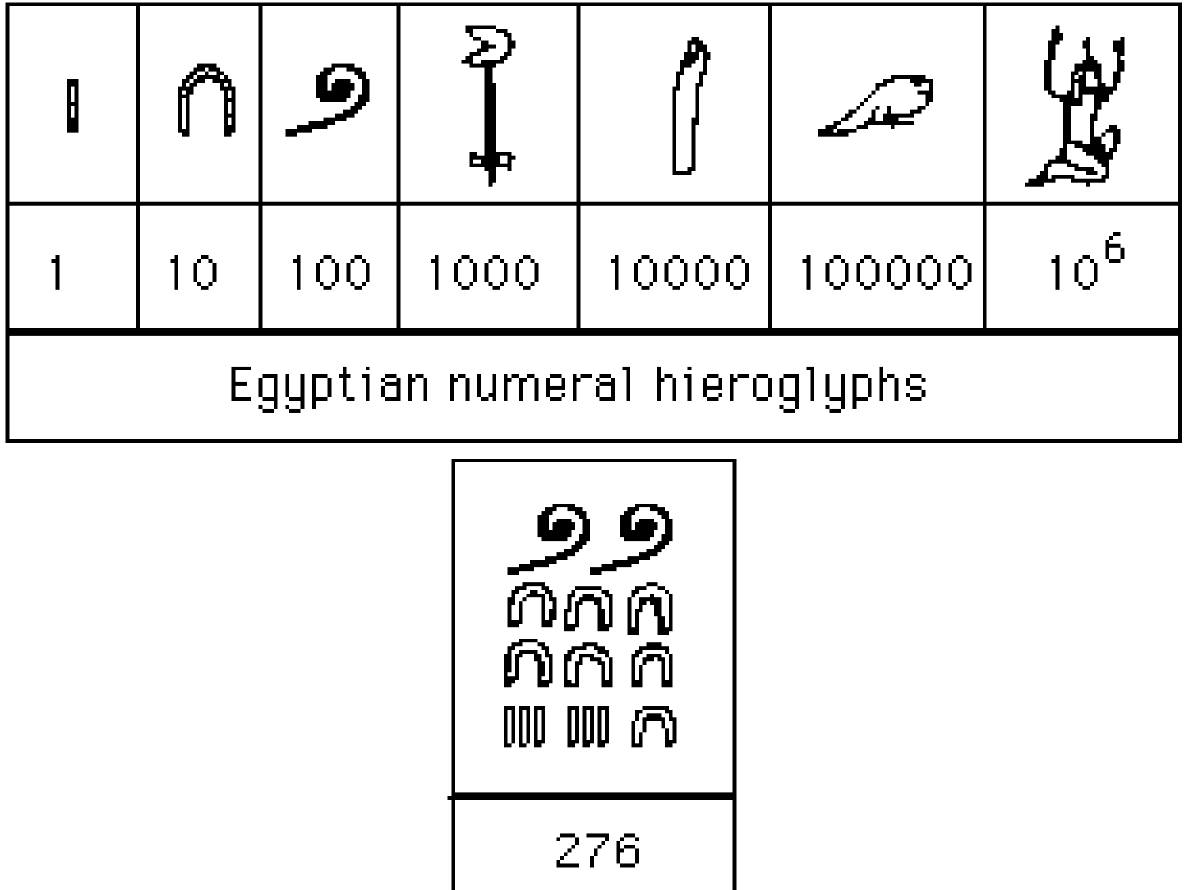


tally marks
(base 1)



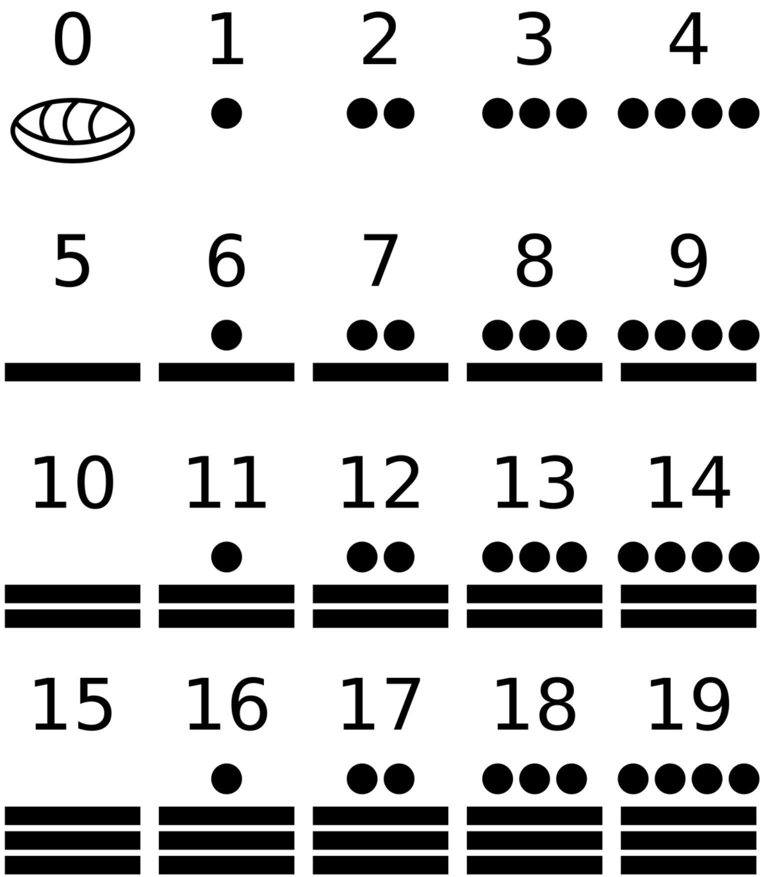
Babylonian numerals
(base 60)

Wikipedia

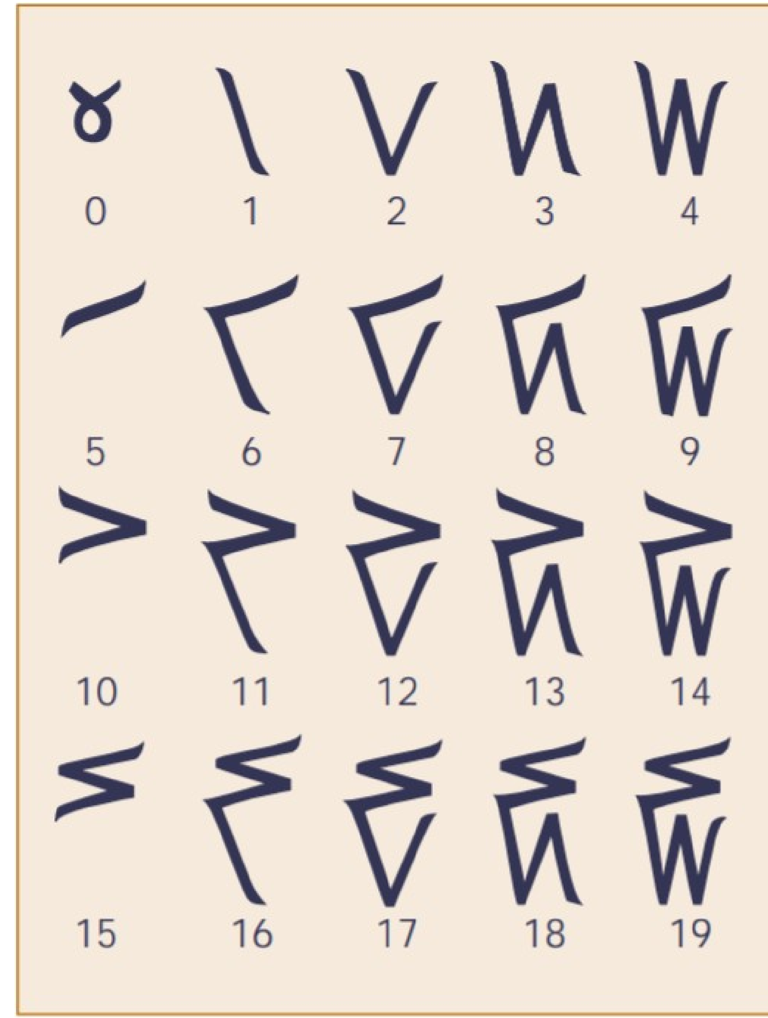


Egyptian numerals (base 10)

Wikipedia



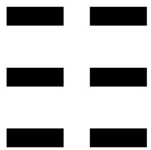
**Maya numerals
(base 20)**



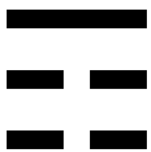
**Kaktovik (Inuit) numerals
(base 20)**

Wikipedia

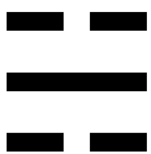
A *counting base* is the “tens” place, the unit you use to mark the “rollover”.



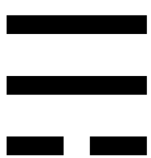
$000_2 = 0_{10}$



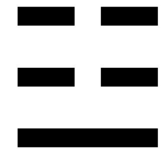
$001_2 = 1_{10}$



$010_2 = 2_{10}$



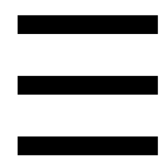
$011_2 = 3_{10}$



$100_2 = 4_{10}$

$101_2 = 5_{10}$

$110_2 = 6_{10}$



$111_2 = 7_{10}$

***I Ching* numerals**
(base 2)
[never actually used for counting]

$$1110.0001.1011_2$$

$$2^{11}+2^{10}+2^{9}+2^{8}+2^{7}+2^{6}+2^{5}+2^{4}+2^{3}+2^{2}+2^{1}+2^{0}$$

$$2048+1024+512+16+8+2+1 = 3611_{10}$$

Binary numerals
(base 2)

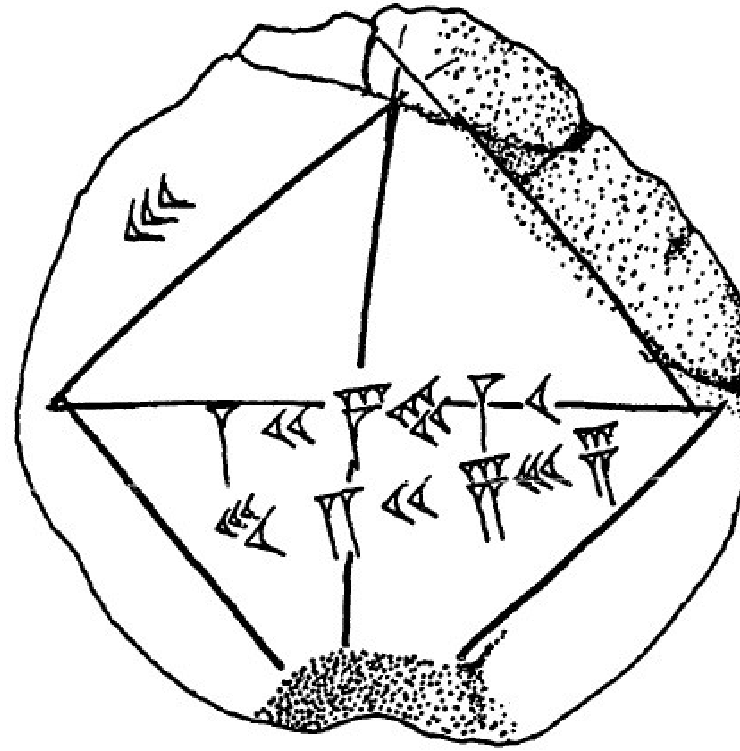
0_{16}	0_{10}	9_{16}	9_{10}
1_{16}	1_{10}	a_{16}	10_{10}
2_{16}	2_{10}	b_{16}	11_{10}
3_{16}	3_{10}	c_{16}	12_{10}
4_{16}	4_{10}	d_{16}	13_{10}
5_{16}	5_{10}	e_{16}	14_{10}
6_{16}	6_{10}	f_{16}	15_{10}
7_{16}	7_{10}	10_{16}	16_{10}
8_{16}	8_{10}	11_{16}	17_{10}

$$0000_2 = 0_{16} = 0_{10}$$
$$0010_2 = 2_{16} = 2_{10}$$
$$0100_2 = 4_{16} = 4_{10}$$
$$1010_2 = a_{16} = 10_{10}$$
$$1011_2 = b_{16} = 11_{10}$$
$$1.0000_2 = 10_{16} = 16_{10}$$
$$11.0010_2 = 32_{16} = 50_{10}$$

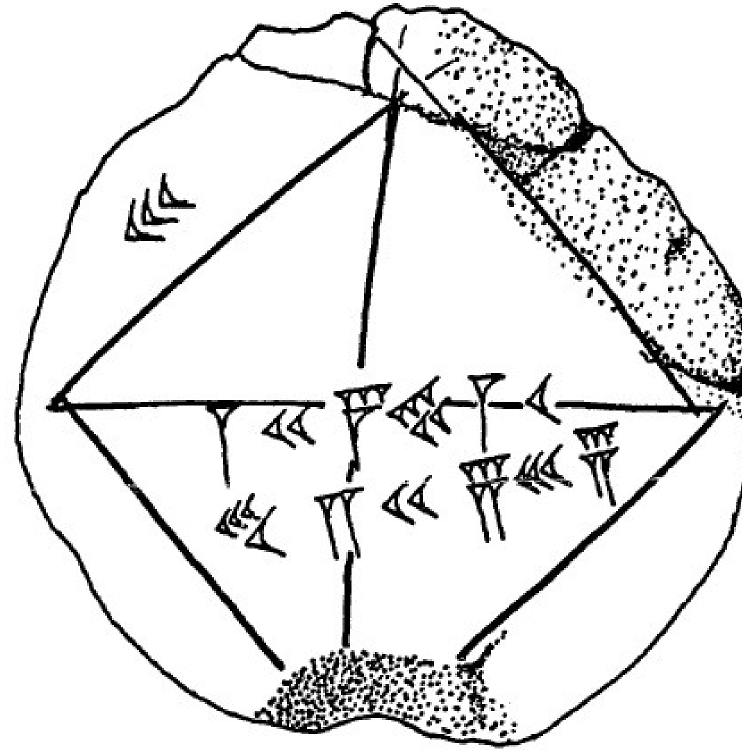
Hexadecimal numerals
(base 16)

Computational Principles

- Representations are interchangeable.



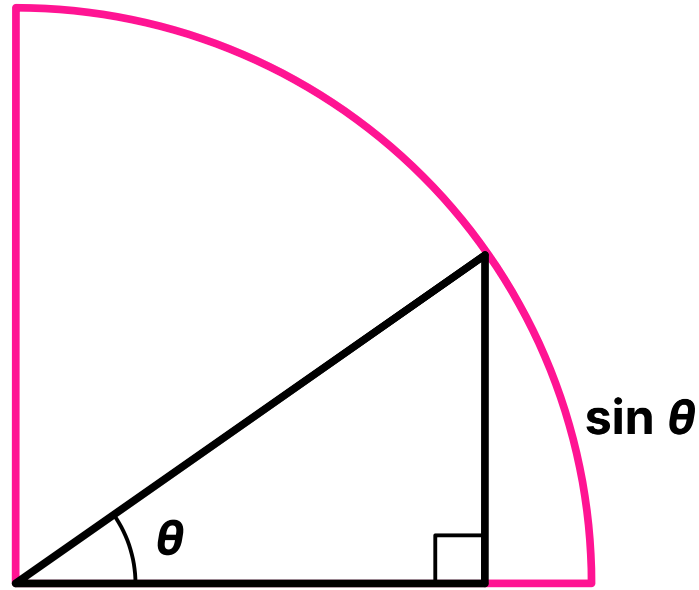
Babylonian tablet YBC 7289



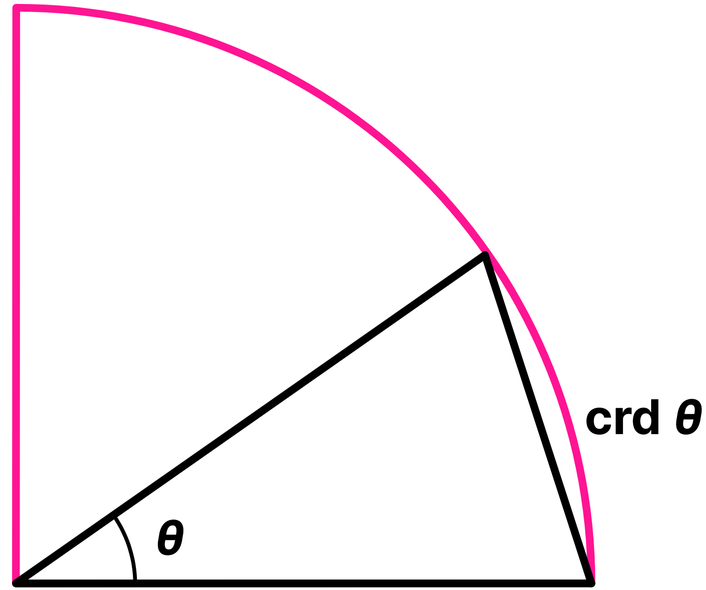
$$1 + \frac{24}{60} + \frac{51}{3600} + \frac{10}{216000} = 1.41417129\ldots$$

$$(\text{compare } \sqrt{2} \approx 1.41421356\ldots)$$

Babylonian tablet YBC 7289

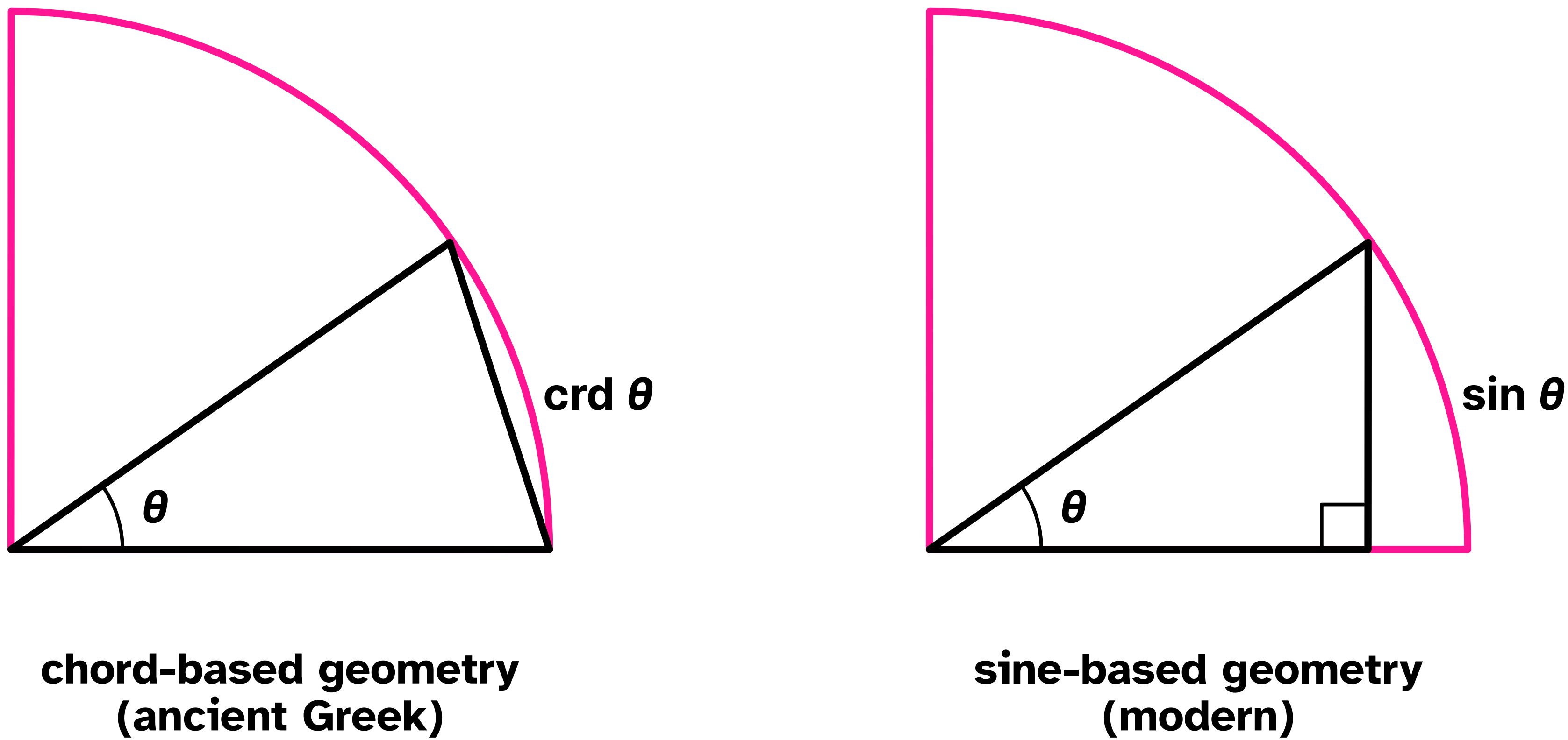


sine-based geometry
(modern)



chord-based geometry
(ancient Greek)

Wikipedia



$$\operatorname{crd}\ \theta = \sqrt{(1-\cos\theta)^2 + \sin^2\theta} = \sqrt{2-2\cos\theta} = 2\sin\left(\frac{\theta}{2}\right)$$

Wikipedia

Computational Principles

- Representations are interchangeable.
- Any computation requires a representation.

To multiply together two numbers:

1) Find the larger of the two.

2) Decompose it into powers of two.

3) Multiply each component by the corresponding power of two.

4) Add these together to find the multiple.

To multiply together two numbers:

$25 \times 13 = ???$

1) Find the larger of the two.

2) Decompose it into powers of two.

3) Multiply each component by the corresponding power of two.

4) Add these together to find the multiple.

To multiply together two numbers:

1) Find the larger of the two.

2) Decompose it into powers of two.

3) Multiply each component by the corresponding power of two.

4) Add these together to find the multiple.

$$25 \times 13 = ???$$

$$\underline{25} \times 13 = ???$$

To multiply together two numbers:

1) Find the larger of the two.

2) Decompose it into powers of two.

3) Multiply each component by the corresponding power of two.

4) Add these together to find the multiple.

$$25 \times 13 = ???$$

$$\underline{25} \times 13 = ???$$

$$25 = 16 + 8 + 1$$

To multiply together two numbers:

$$25 \times 13 = ???$$

1) Find the larger of the two.

$$\underline{25} \times 13 = ???$$

2) Decompose it into powers of two.

$$25 = 16 + 8 + 1$$

3) Multiply each component by the corresponding power of two.

$$16 \times 13 = 208$$
$$8 \times 13 = 104$$
$$1 \times 13 = 13$$

4) Add these together to find the multiple.

To multiply together two numbers:

$$25 \times 13 = ???$$

1) Find the larger of the two.

$$\underline{25} \times 13 = ???$$

2) Decompose it into powers of two.

$$25 = 16 + 8 + 1$$

3) Multiply each component by the corresponding power of two.

$$16 \times 13 = 208$$
$$8 \times 13 = 104$$
$$1 \times 13 = 13$$

4) Add these together to find the multiple.

$$25 \times 13 = 208 + 104 + 13 = 325$$

Ancient Egyptian multiplication algorithm

Computational Principles

- Representations are interchangeable.
- Any computation requires a representation.
- Computation requires *values*, *operations*, and *state*.

“Zero is not the successor of any natural number.”

Gödel, Escher, Bach (Hofstadter)

$$\forall a : \sim Sa = 0$$

“Zero is not the successor of any natural number.”

Symbol		*Codon*	*Mnemonic Justification*	
0	.....	666	Number of the Beast for the Mysterious Zero	
S	.....	123	successorship: 1, 2, 3, ...	
=	.....	111	visual resemblance, turned sideways	
+	.....	112	$1 + 1 = 2$	
·	.....	236	$2 \times 3 = 6$	
(	.....	362	ends in 2	
)	.....	323	ends in 3	
<	.....	212	ends in 2	*these three pairs*
>	.....	213	ends in 3	*form a pattern*
[	.....	312	ends in 2	
]	.....	313	ends in 3	
a	.....	262	opposite to ∀ (626)	
′	.....	163	163 is prime	
∧	.....	161	'∧' is a "graph" of the sequence 1-6-1	
∨	.....	616	'∨' is a "graph" of the sequence 6-1-6	
⊃	.....	633	6 "implies" 3 and 3, in some sense ...	
~	.....	223	2 + 2 is *not* 3	
∃	.....	333	'∃' looks like '3'	
∀	.....	626	opposite to a; also a "graph" of 6-2-6	

∀ a : ~ S a = 0

"Zero is not the successor of any natural number."

Gödel, Escher, Bach (Hofstadter)

Symbol		*Codon*	*Mnemonic Justification*	
0	.....	666	Number of the Beast for the Mysterious Zero	
S	.....	123	successorship: 1, 2, 3, . . .	
=	.....	111	visual resemblance, turned sideways	
+	.....	112	1 + 1 = 2	
·	.....	236	2 × 3 = 6	
(	.....	362	ends in 2	
)	.....	323	ends in 3	
<	.....	212	ends in 2	*these three pairs*
>	.....	213	ends in 3	*form a pattern*
[	.....	312	ends in 2	
]	.....	313	ends in 3	
a	.....	262	opposite to ∀ (626)	
′	.....	163	163 is prime	
∧	.....	161	'∧' is a "graph" of the sequence 1-6-1	
∨	.....	616	'∨' is a "graph" of the sequence 6-1-6	
⊃	.....	633	6 "implies" 3 and 3, in some sense . . .	
~	.....	223	2 + 2 is *not* 3	
∃	.....	333	'∃' looks like '3'	
∀	.....	626	opposite to a; also a "graph" of 6-2-6	

626,	262,	636,	223,	123,	262,	111,	666
∀	a	:	~	S	a	=	0

"Zero is not the successor of any natural number."

Gödel, Escher, Bach (Hofstadter)

626,262,636,626,262,163,636,362,262,112,123,262,163,323,111,123,362,262,112,262,163,323 axiom 3
∀ a : ∀ a ′ : (a + S a ′) = S (a + a ′)

626,262,163,636,362,123,666,112,123,262,163,323,111,123,362,123,666,112,262,163,323 specification
∀ a ′ : (S 0 + S a ′) = S (S 0 + a ′)

362,123,666,112,123,666,323,111,123,362,123,666,112,666,323 specification
(S 0 + S 0) = S (S 0 + 0)

626,262,636,362,262,112,666,323,111,262 axiom 2
∀ a : (a + 0) = a

362,123,666,112,666,323,111,123,666 specification
(S 0 + 0) = S 0

123,362,123,666,112,666,323,111,123,123,666 insert ‘123’
S (S 0 + 0) = S S 0

362,123,666,112,123,666,323,111,123,123,666 transitivity
(S 0 + S 0) = S S 0

Computers use *assembler* (or assembly) language as their fundamental binary language.

Each instruction has a determinate length.

The first number represents the *operation*, which determines the interpretation of the other numbers (*values* or *addresses*, which access stored *state*).

```
0b11.0101.1110.0000.0000.0000.0110.0101.1110.0000.0001.0000.0111.0000.0010.0101.1110.0000.0010
```

```
0x35e.0006.5e01.0702.5e02
```

```
MOVMA   5E00   ; Move data from address 0x5e00 into A
MOVBA   5E01   ; Move data from address 0x5e01 into A
ADDAB          ; Add A and B and put result in A
MOVAM   5E02   ; Move data from A to address 0x5e02
```

Intel 8088 assembler language; example from https://faculty.etsu.edu/tarnoff/ntes2150/assembly/assembly.htm

Computers use *assembler* (or assembly) language as their fundamental binary language.

Each instruction has a determinate length.

The first number represents the *operation*, which determines the interpretation of the other numbers (*values* or *addresses*, which access stored *state*).

```
0b11.0101.1110.0000.0000.0000.0110.0101.1110.0000.0001.0000.0111.0000.0010.0101.1110.0000.0010
```

```
0x35e.0006.5e01.0702.5e02
```

```
MOVMA    5E00    ; Move data from address 0x5e00 into A
MOVBA    5E01    ; Move data from address 0x5e01 into A
ADDAB            ; Add A and B and put result in A
MOVAM    5E02    ; Move data from A to address 0x5e02
```

Intel 8088 assembler language; example from https://faculty.etsu.edu/tarnoff/ntes2150/assembly/assembly.htm

- 8-bit: $2^8 = 256$ [NES, IBM System/360]
- 16-bit: $2^{16} = 65{,}536$ [8088, SNES]
- 32-bit: $2^{32} \approx$ 4MM [Pentium Pro, PS]
- 64-bit: $2^{64} \approx 1.8 \times 10^{19}$ [modern CPUs]

Computational Principles

- Representations are interchangeable.
- Any computation requires a representation.
- Computation requires *values*, *operations*, and *state*.
- All of these can be represented unambiguously on the machine.

Fibonacci sequence

$$F_n = F_{n-1} + F_{n-2}$$

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Fibonacci sequence

$$F_n = F_{n-1} + F_{n-2}$$

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

```
def fib(n):
  if n <= 2:
    return 1
  return fib(n-1) + fib(n-2)
```

Fibonacci sequence

$$F_n = F_{n-1} + F_{n-2}$$

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

```python
def fib(n):
  if n <= 2:
    return 1
  return fib(n-1) + fib(n-2)
```

```
 0 LOAD_FAST                0 (n)
 2 LOAD_CONST               1 (2)
 4 COMPARE_OP               1 (<=)
 6 POP_JUMP_IF_FALSE       12
 8 LOAD_CONST               2 (1)
10 RETURN_VALUE
12 LOAD_GLOBAL              0 (fib)
14 LOAD_FAST                0 (n)
16 LOAD_CONST               2 (1)
18 BINARY_SUBTRACT
20 CALL_FUNCTION            1
22 LOAD_GLOBAL              0 (fib)
24 LOAD_FAST                0 (n)
26 LOAD_CONST               1 (2)
28 BINARY_SUBTRACT
30 CALL_FUNCTION            1
32 BINARY_ADD
34 RETURN_VALUE
```

Python 3.9

Computational Principles

- Representations are interchangeable.
- Any computation requires a representation.
- Computation requires *values*, *operations*, and *state*.
- All of these can be represented unambiguously on the machine.
- Higher-level languages map to lower-level representations for the machine.

The *lambda calculus* is a formal mathematical specification for computation.

1) Mathematical functions are anonymous.

$$f(x) = (x+1)^2 \;\Rightarrow\; (x) \mapsto (x+1)^2 \;\Rightarrow\; \lambda x.(\lambda x.x^2\ (\lambda x.x + 1\ \ x))$$

2) The lambda calculus consists of *lambda terms* and defines a set of formal operations for manipulating them.

TRUE $:= \lambda x.\lambda y.x$
FALSE $:= \lambda x.\lambda y.y$

AND $:= \lambda p.\lambda q.p\ q\ p$
OR $:= \lambda p.\lambda q.p\ p\ q$
NOT $:= \lambda p.p$ **FALSE TRUE**
IFTHENELSE$:= \lambda p.\lambda a.\lambda b.p\ a\ b$

I $:= \lambda x.x$
S $:= \lambda x.\lambda y.\lambda z.x\ z\ (y\ z)$
K $:= \lambda x.\lambda y.x$
B $:= \lambda x.\lambda y.\lambda z.x\ (y\ z)$
C $:= \lambda x.\lambda y.\lambda z.x\ z\ y$
W $:= \lambda x.\lambda y.x\ y\ y$
ω/Δ$:= \lambda x.x\ x$
Ω $:=$ **ω ω**

3) Other (equivalent) logic systems exist; Urbit's Nock language is most closely related to the *SKI combinator calculus*.

Computational Principles

- Urbit consists of these parts:
  - **Nock** is a *virtual machine*, a computational behavior specification (like assembler).
  - **Hoon** is a *high-level language* which compiles to Nock (like C or Python).
  - **Arvo** is the Urbit OS, the *event handler* and *event log* which together define *system state*.
  - **Azimuth** is the *identity* system.