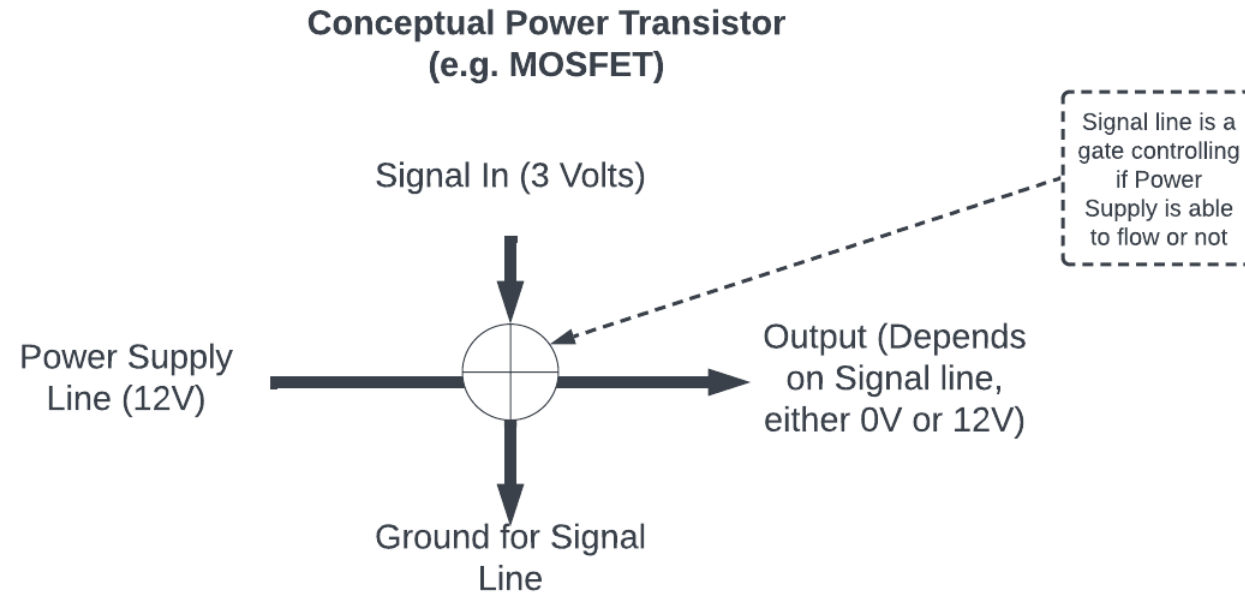


Mathematics of Computers
by
Matthew Zamora

Presented to Rockville Science Center's Numberphiles on 16Nov23

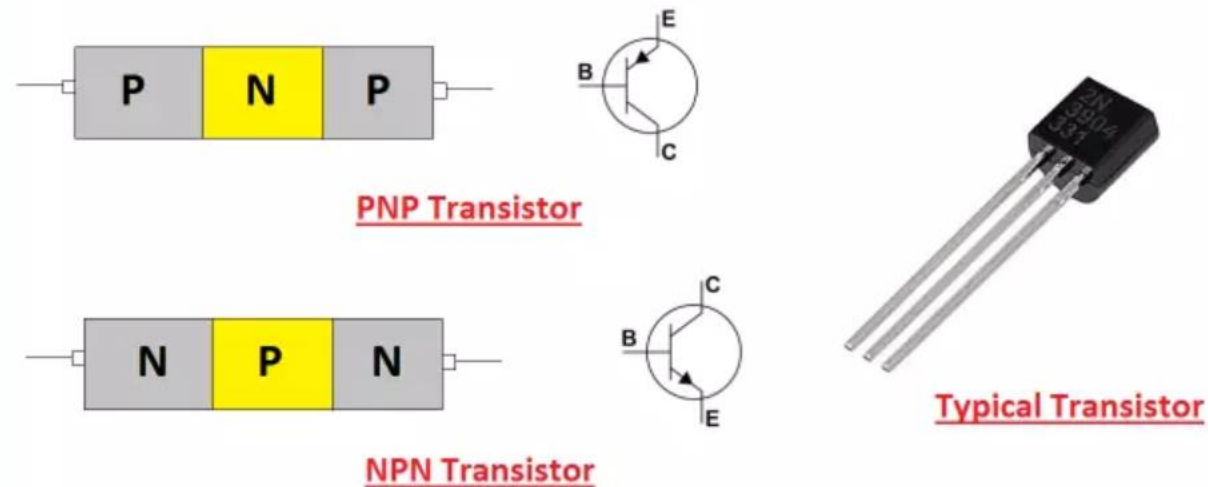
Transistors have a pass through voltageage that is moderated by a signal voltageage



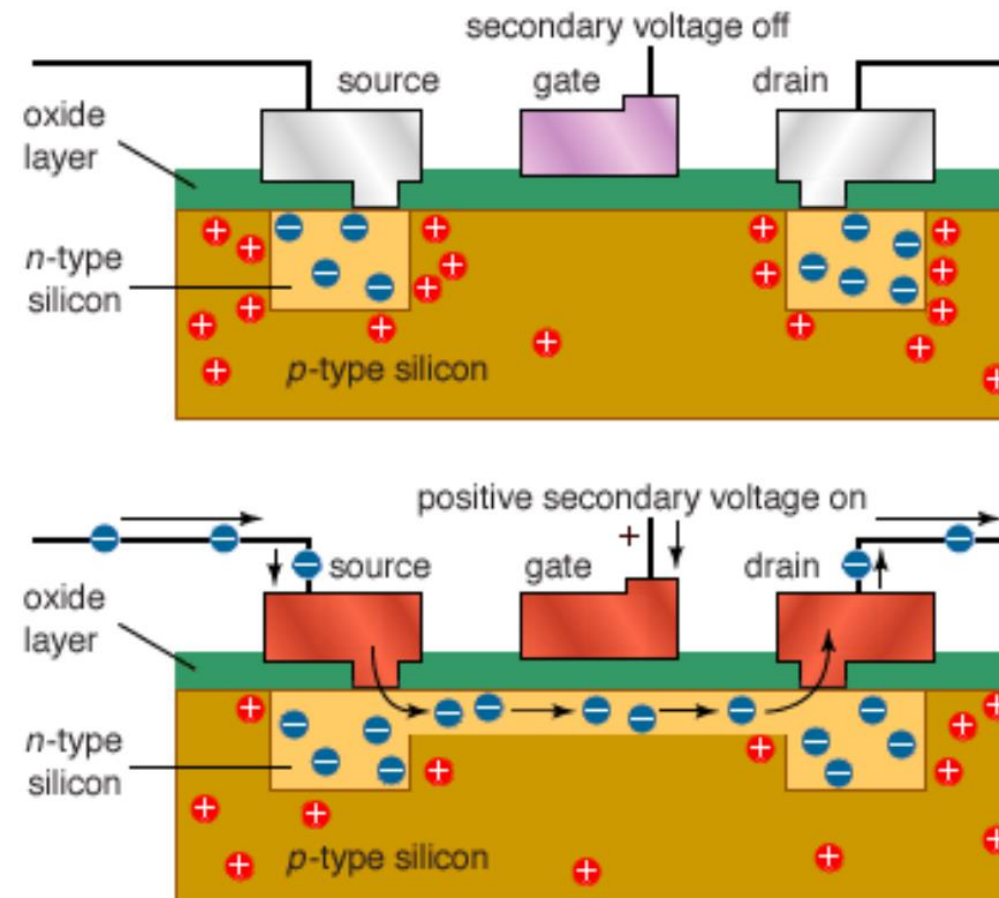
Notice how the signal voltage could be orders of magnitude lower than the through voltage, allowing for a step up voltage similar to how a CPU (low voltage) can turn 'on' an electric motor (high voltage).

Two Types of Transistors – Control line either turns on or off the output


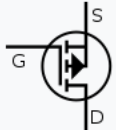
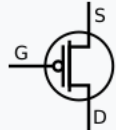
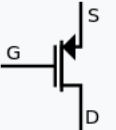

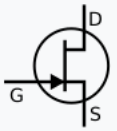
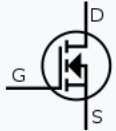
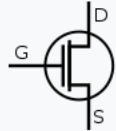
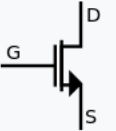
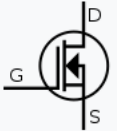
- P-type – giving the gate pin voltage, turns off the output
- N-type – giving the gate pin voltage turns on the output
- In practice we only need three pins instead of four – because the gate pin and power supply can share the same ground.



Turning on the gate (signal line) creates an electric field that creates a 'conductive' zone of the semiconductor allowing electricity to flow. Only a small voltage is needed to establish the conductive zone – but a high voltage can flow through the bridge.



Transistor Symbols

P-channel					
N-channel					
	JFET	MOSFET enh.	MOSFET enh. (no bulk)	MOSFET enh. (no bulk)	MOSFET dep.

Bit Addition

Addition		Result	Carry
0 + 0	=	0	0
0 + 1	=	1	0
1 + 0	=	1	0
1 + 1	=	0	1

Logic

- A Cat is a Dog, and a Hamster is a Mammal \rightarrow False and True \rightarrow False
- $1=1$ or $2=3$, True or False \rightarrow True
- It is not true that a Cat is a Dog, and a Hamster is a Mammal \rightarrow
(not False) and True \rightarrow (True) and True \rightarrow True

Abstraction #1 Voltage as True/False

- Let True be a wire with 5V, and False be a wire with 0V (grounded)

Abstraction #2

- Let numbers be represented as an ordered set of bits (a Boolean vector, or truth vector)
- Following n^2 order, represent arbitrary numbers as follow:

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Abstraction #2 – part B

- Letters can also be represented

Converting the text "hope" into binary				
Characters:	h	o	p	e
ASCII Values:	104	111	112	101
Binary Values:	01101000	01101111	01110000	01100101
Bits:	8	8	8	8

ComputerHope.com

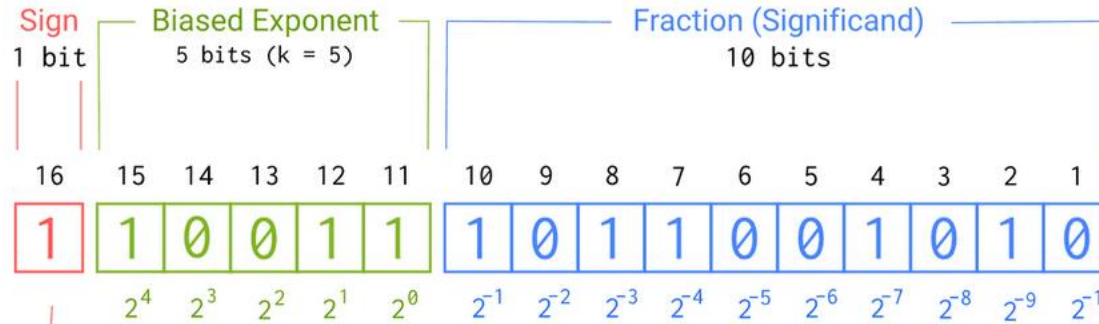
Character	Decimal Number	Binary Number	Character	Decimal Number	Binary Number
blank space	32	0010 0000	^	94	0101 1110
!	33	0010 0001	-	95	0101 1111
"	34	0010 0010	`	96	0110 0000
#	35	0010 0011	a	97	0110 0001
\$	36	0010 0100	b	98	0110 0010
A	65	0100 0001	c	99	0110 0011
B	66	0100 0010	d	100	0110 0100
C	67	0100 0011	e	101	0110 0101
D	68	0100 0100	f	102	0110 0110
E	69	0100 0101	g	103	0110 0111
F	70	0100 0110	h	104	0110 1000
G	71	0100 0111	i	105	0110 1001
H	72	0100 1000	j	106	0110 1010
I	73	0100 1001	k	107	0110 1011
J	74	0100 1010	l	108	0110 1100
K	75	0100 1011	m	109	0110 1101
L	76	0100 1100	n	110	0110 1110
M	77	0100 1101	o	111	0110 1111
N	78	0100 1110	p	112	0111 0000
O	79	0100 1111	q	113	0111 0001
P	80	0101 0000	r	114	0111 0010
Q	81	0101 0001	s	115	0111 0011
R	82	0101 0010	t	116	0111 0100
S	83	0101 0011	u	117	0111 0101
T	84	0101 0100	v	118	0111 0110
U	85	0101 0101	w	119	0111 0111
V	86	0101 0110	x	120	0111 1000
W	87	0101 0111	y	121	0111 1001
X	88	0101 1000	z	122	0111 1010
Y	89	0101 1001	{	123	0111 1011
Z	90	0101 1010		124	0111 1100
[91	0101 1011	}	125	0111 1101
/	92	0101 1100	~	126	0111 1110
]	93	0101 1101			

Abstraction #2 – part C

- In fact – let any arbitrary symbol be represented as a unique sequence of bits. What symbols are depend on how they are used.
Common uses:
 - Instruction sequences – instruct the CPU what operation to do next
 - Location in memory – a location to the start of a sequence of binary information in a matrix of bits (RAM)
 - Anything a program might be using – for example a floating point (decimal) of arbitrary (or fixed) decimal precision

Half-precision 16 bits

trekhele.dev



= -27.15625

$$\text{exponent} = 2^4 + 2^1 + 2^0 = 19$$

$$\text{bias} = 2^{k-1} - 1 = 2^{5-1} - 1 = 15$$

$$\text{biased_exponent} = \text{exponent} - \text{bias} = 19 - 15 = 4$$

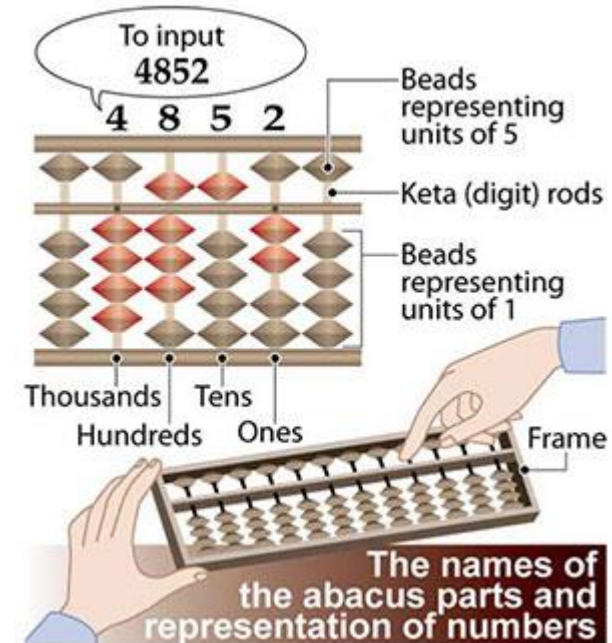
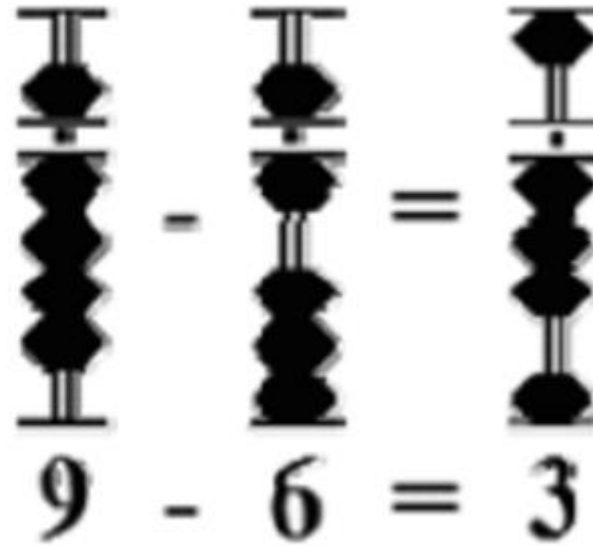
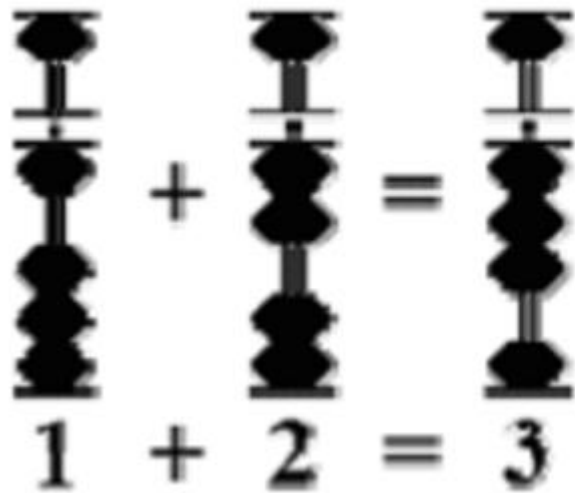
$$\text{fraction} = 2^{-1} + 2^{-3} + 2^{-4} + 2^{-7} + 2^{-9} =$$

$$= 0.5 + 0.125 + 0.0625 + 0.0078125 + 0.001953125 = 0.697265625$$

$$-1^1 \times 2^4 \times (2^0 + 0.697265625) = -27.15625$$

Math has a long history of abstracting numbers, and storing them by a base

An Abacus. Note how for 9, one bead represents 5.



Abstraction #3 – Addition Can Be Done Via Comparing the State of Two Bit Vectors

B I N A R Y A D D I T I O N

$$\begin{array}{r} + 110 \\ 111 \\ \hline 1101 \end{array}$$

$$\begin{array}{r} 6 \\ +7 \\ =13 \end{array}$$

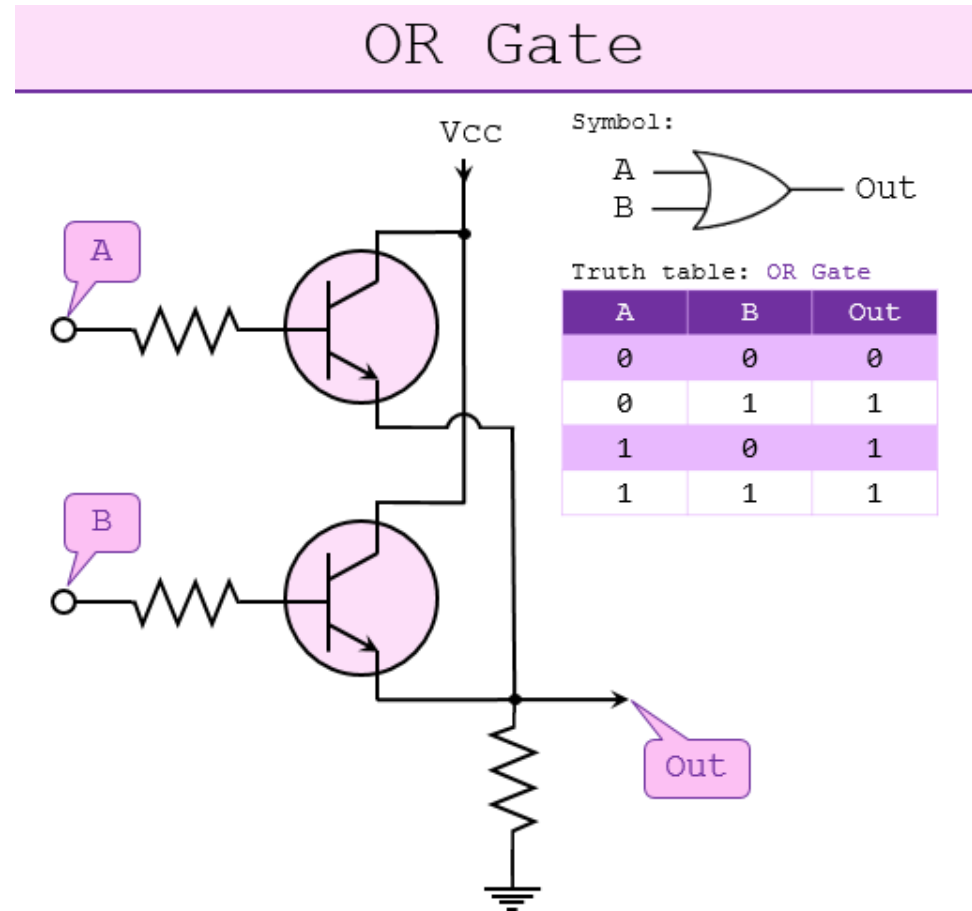
$$\begin{array}{r} + 10110 \\ 10111 \\ \hline 101101 \end{array}$$

$$\begin{array}{r} 22 \\ +23 \\ =45 \end{array}$$

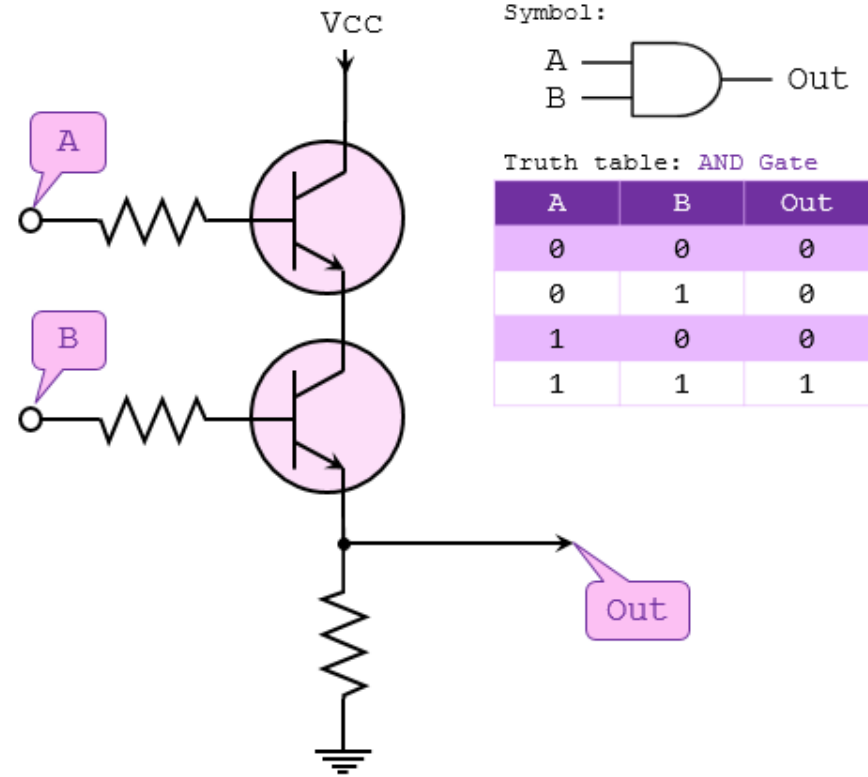
$$\begin{array}{r} 110101 \\ + 100110 \\ 100111 \\ \hline 10000010 \end{array}$$

$$\begin{array}{r} 53 \\ +38 \\ 39 \\ =130 \end{array}$$

Abstraction #4 Basic Logic can be represented by physical transistor configurations

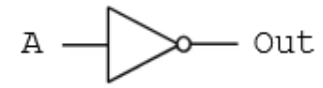


AND Gate



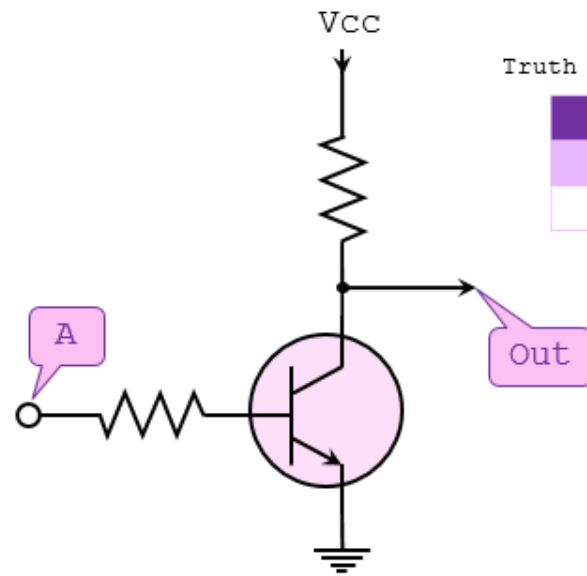
NOT Gate

Symbol:

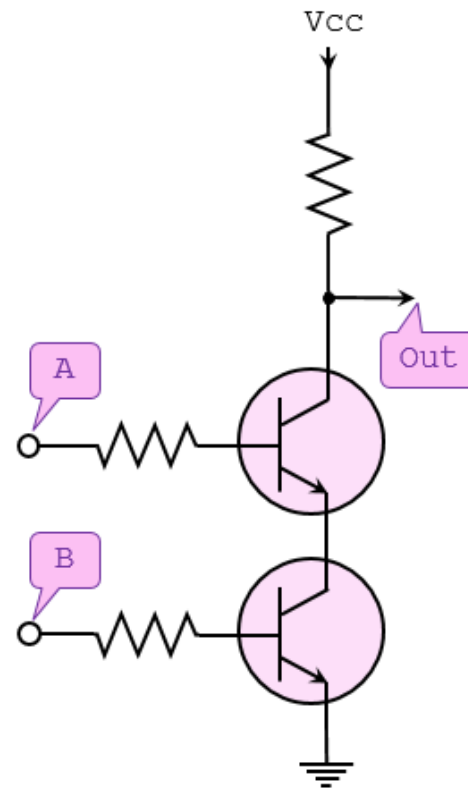


Truth table: NOT Gate

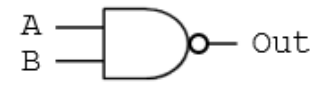
A	Out
0	1
1	0



NAND Gate



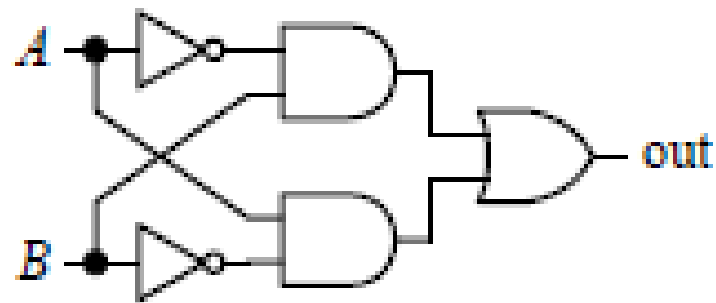
Symbol:



Truth table: NAND Gate

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

Exclusive OR (XOR)

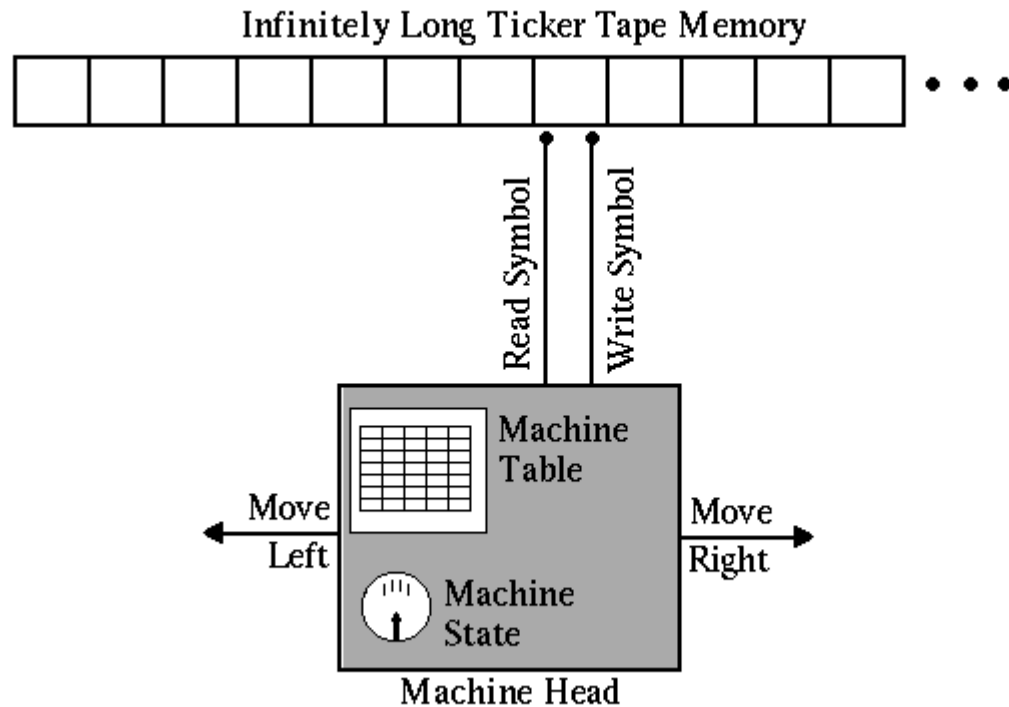


equivalent circuit

A	B	out
0	0	0
0	1	1
1	0	1
1	1	0

The output of an XOR gate goes HIGH if the inputs are different from each other. XOR gates only come with two inputs.

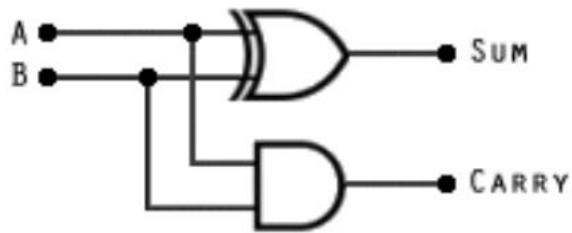
Universal Turing Machine



Any arbitrary algorithm can be run as a series of steps assuming:

- There is infinite memory tape that can be read from, and written to
- A machine can read the tape, and do conditional steps based on the value loaded from the tape:
 - Move to a given location and trigger a read
 - Move to a given location and trigger a write based on 'state' set from the previous reads
- With that – any programming language can be written!
- RAM is the memory, and the CPU is the machine fetching data from RAM, and doing an operation on the data – eventually writing back to RAM (this writing can directly control your monitor or peripheral devices too!)
- Note that the “Von-Neumann architecture” is the practical implementation of a Turing machine used in modern computers as a state-machine.

Abstraction #5 -Bit Addition Is a Set of Logic Gates



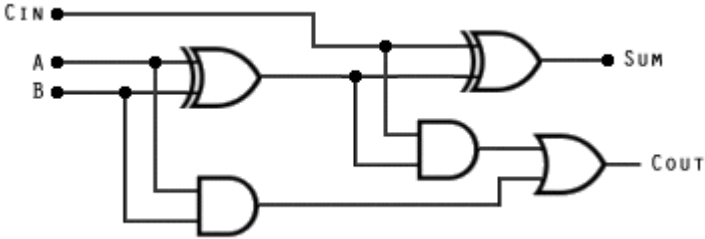
The half-adder

Half-adder truth table

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Name	Symbol & notation	Explanation
NOT	$A \rightarrow \bar{A}$	The inverter NOT simply accepts an input and outputs the opposite .
AND	$A, B \rightarrow AB$	All inputs must be positive (1) before the output is positive (1 or ON)
NAND <small>*Not AND</small>	$A, B \rightarrow \overline{A \cdot B}$	Same as AND, but the outcome is the inverse (NOT) . So, perform AND first, then apply NOT to the output.
OR	$A, B \rightarrow A+B$	At least one input must be positive (1) to give a positive output (1 or ON). All inputs could also be positive.
NOR <small>*Not OR</small>	$A, B \rightarrow \overline{A+B}$	Same as OR, but the outcome is the inverse (NOT) . So, perform OR first, then apply NOT to the output.
XOR <small>*Exclusive OR</small>	$A, B \rightarrow A \oplus B$	Only one input can be positive (1) to give a positive output (1 or ON). If both are positive, the output is negative (0 or OFF)
XNOR <small>*Exclusive Not OR</small>	$A, B \rightarrow \overline{A \oplus B}$	All inputs must be the same (either high or low) for a positive output (1). Otherwise, the output is negative (0 or OFF)

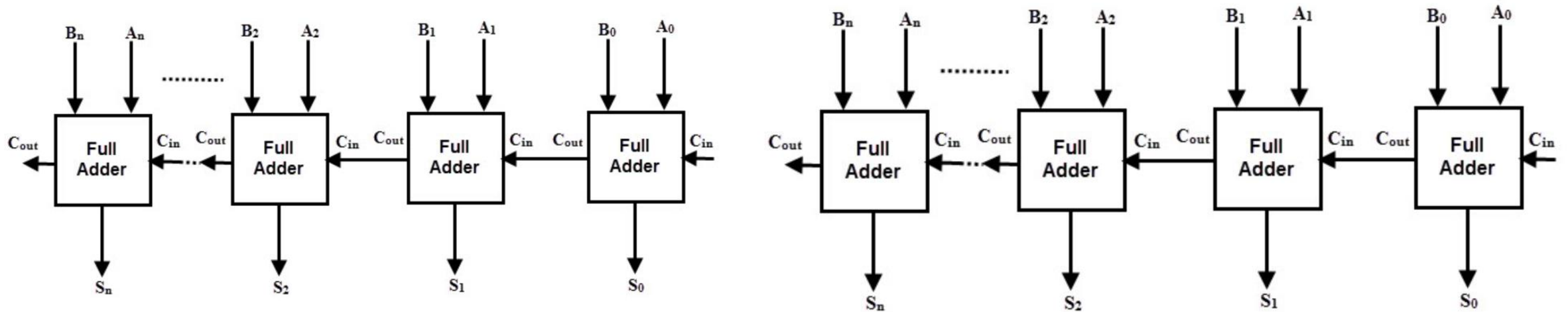
The Full Bit Adder



Full-adder truth table

Inputs			Outputs	
A	B	C _{IN}	SUM	C _{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The ADD instruction a one byte (8 bits) adder



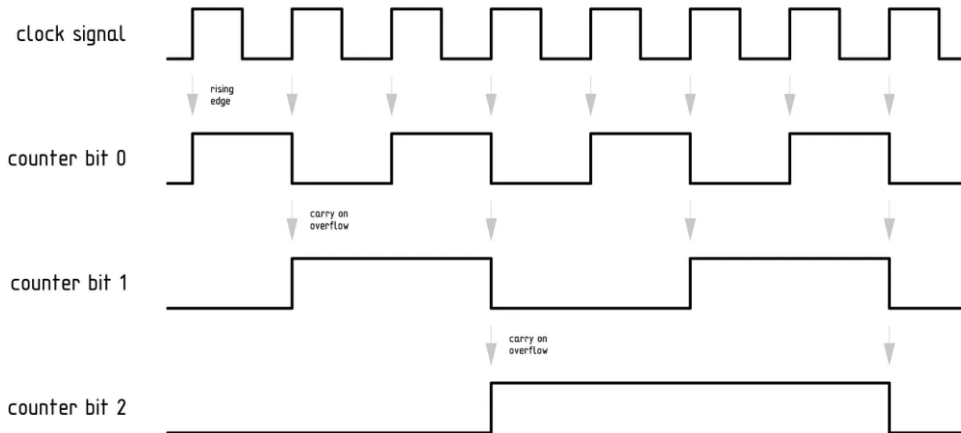
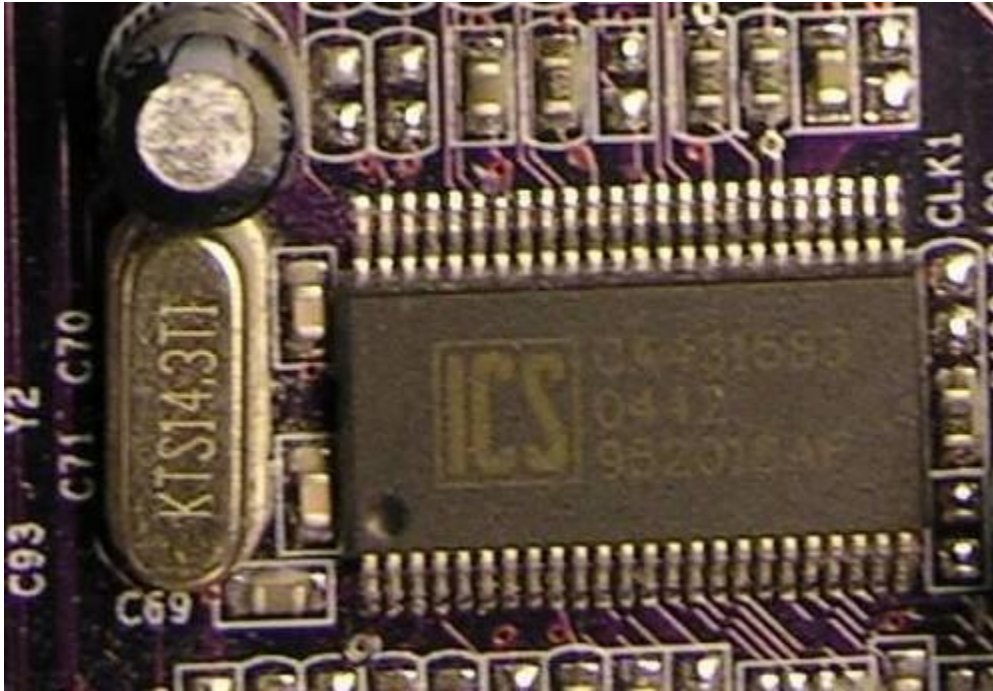
Abstraction #6 – The CPU has instructions that ‘trigger’ transistor-based logic

- Notice the “ADD” instruction that will compare one byte (e.g. 00000001 + 00000010) and move the result to ‘results’ location called a register (result: 00000011)
- There is also a “MOV” which moves to a specified location in memory
- Critically there is a “CMP” which compares two values, and will move to location A if the first values is than the second value, otherwise it will go to location B. This is like the ‘if this then do this else do that’ type of logic in programming language. This allows for dynamic program branching.
- One instruction is called ‘per clock tick’ – a square wave

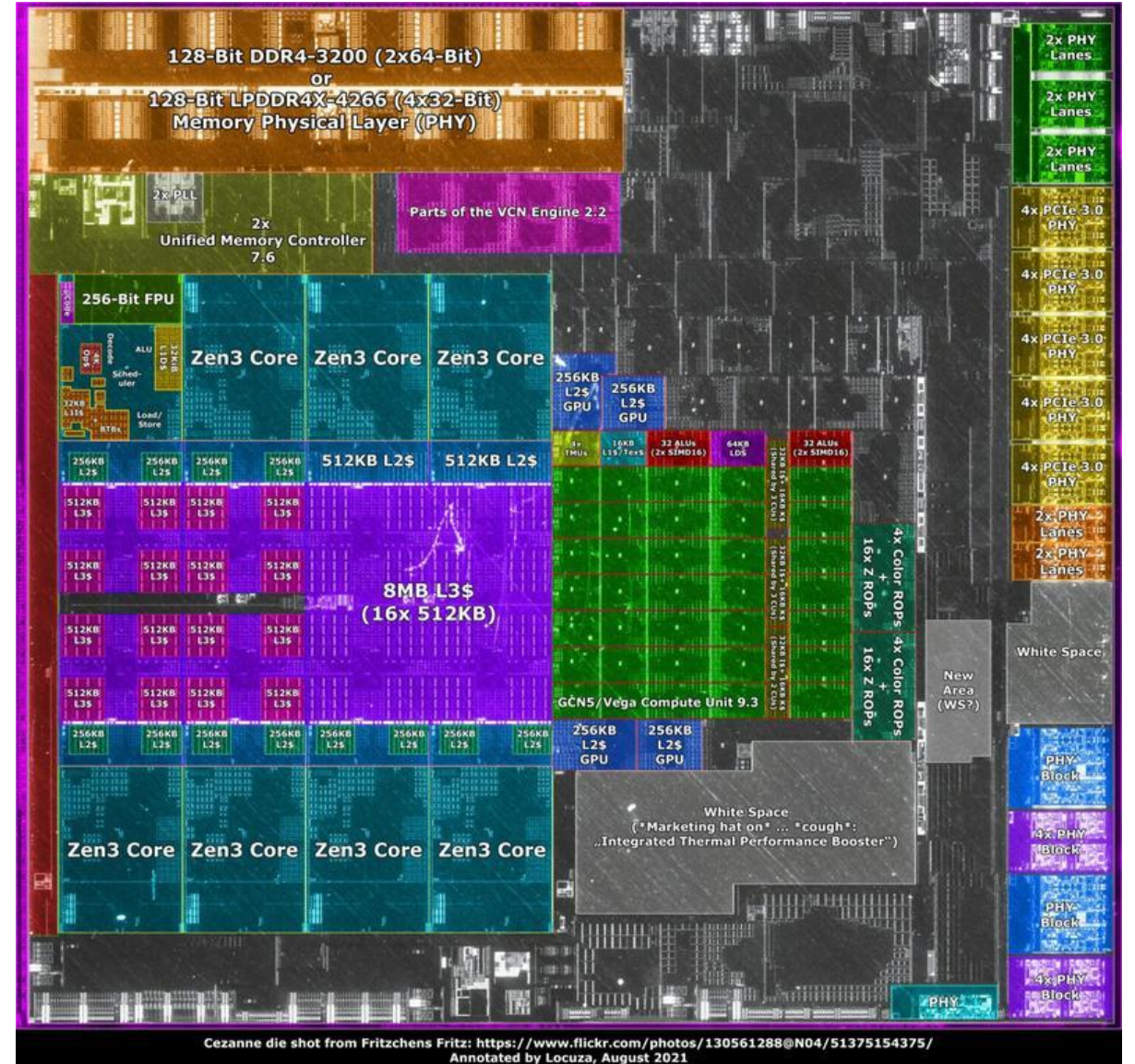
Fig 5: ZipCPU Instructions

Normal instructions				Compressed	
00000	SUB	10000	CMP	000	SUB
00001	AND	10001	TEST	001	AND
00010	ADD	10010	LW	010	ADD
00011	OR	10011	SW	011	CMP
00100	XOR	10100	LH	100	LW
00101	LSR	10101	SH	101	SW
00110	LSL	10110	LB	110	LDI
00111	ASR	10111	SB	111	MOV
01000	BREV	11000	LDI	Reserved for FPU	
01001	LDILO	11001			
01010	MPYUHI	Special Insn		11010	FPADD
01011	MPYSHI			11011	FPSUB
01100	MPY	11100	BREAK	11100	FPMPY
01101	MOV	11101	LOCK	11101	FPDIV
01110	DIVU	11110	SIM	11110	FPI2F
01111	DIVS	11111	NOOP	11111	FPF2I

CPU and Microprocessors



<https://lcamtuf.substack.com/p/clocks-in-digital-circuits>

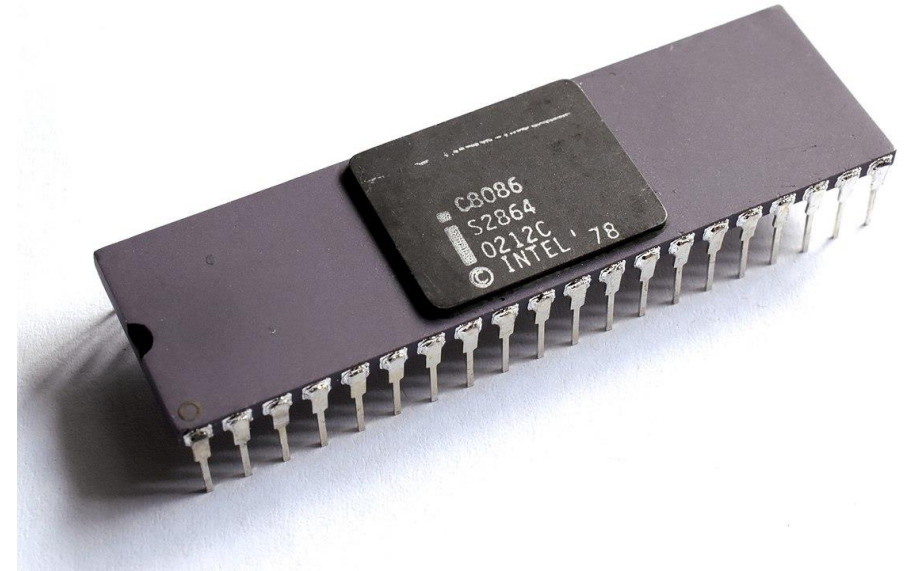


Cezanne die shot from Fritzchens Fritz: <https://www.flickr.com/photos/130561288@N04/51375154375/>
Annotated by Locuza, August 2021

Abstraction #7

- All programs and algorithms can be decomposed to a standard set of no more than 81 basic instructions – standardized by Intel in 1978 – (this is the x86 instruction set, named for the original chips called the 8086 and the 8088).
- More complex instructions (such as multiply or divide) can be expressed as a set of similar instructions done in order
- Modern CPUs/GPUs extend the 81

https://en.wikipedia.org/wiki/X86_instruction_listings



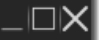
Notice how one “Multiply” step of
 3×3 or `0011 MUL 0011`

can also be decomposed into multiple
ADD steps:

`0011 ADD 0011 = 0110` (i.e. 6)
`0110 ADD 0011 = 1001` (i.e. 9)

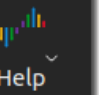


DSView v1.2.1



USB 2.0 DSLogic PLUS Options 2.00 s @ 1 MHz

Mode Start Instant Trigger Decode Measure Search Display File



Protocol

Protocol search...

hex 0:UART 100%

Protocol List Viewer

search

Matching Items: 200

0:UART: RX/TX

0	00
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09
10	0A
11	0B
12	0C
13	0D
14	0E
15	0F

Trigger Setting...

Simple Trigger

Advanced Trigger

Trigger Position: 1 %

Total Trigger Stages: 1

Stage Trigger Serial Trigger

Trigger Time: 2022-09-07 19:11:11

DSView v0.9.9

DSLogic Plus 3.36 s @ 10 MHz

Options Mode Start Instant Trigger Decoder Measure Search File

LA 838.40ms +1839.04ms +1839.68ms +1840.32ms +1840.96ms +1841.60ms +1842.24ms +1842.88ms +1843.52ms +1844.15ms +1844.79ms +1845.43ms +1846.07ms +1846.71ms

0:UART 2 D t e s t i n g [00]

0 1 2 3 4 5 6 7

Protocol

+ 1:UART

0:UART 100%

Protocol List Viewer

search

Matching Items: 24

0:UART: RX/TX

0 Start bit

1 t

2 Stop bit

3 Start bit

4 e

5 Stop bit

6 Start bit

7 s

8 Stop bit

9 Start bit

10 t

11 Stop bit

12 Start bit

13 i

14 Stop bit

15 Start bit

16 n

17 Stop bit

18 Start bit

Decoder Options

0:UART

0:UART: RX/TX

RX/TX (UART transceiver line) * 2

Baud rate 9600

Data bits 8

Parity type none

Check parity? no

Stop bits 1.0

Bit order lsb-first

Data format ascii

Invert Signal? yes

* Required channels

Start Decode Start From

End Decode End to

Stack Decoder

OK Cancel

Trigger Time: 2019-09-05 23:09:51 Thu

Abstraction #9 – Operating Systems keep the CPU busy and work with both input and output devices

- An Operating System runs a loop of CPU instructions that follow a sequence of CPU operations – (including moving things into and out of memory).
- Critically the OS can handle ‘conditional’ input like key strokes and mouse movements – to affect the ‘flow’ of CPU logic using conditional jump statements like “CMP” discussed before.

Abstraction #10 – Peripheral Devices are viewed as memory locations by the CPU

- Consider that memory is just a sequence (vector) or array (matrix) of bits. Some 'locations' of the memory sequence are reserved for special functionality physically baked into the CPU. For example – in BIOS mode we can get a pointer to video memory that starts at 0xb8000; - Writing to this part of 'memory' displays a character to a position on the screen.
- Input devices trigger events that 'write' to memory accessibly by the CPU. The OS will loop and check for these events. Certain features like 'interrupts' occur when values are seen in special bits in memory. For example a mouse movement writes a 1 to an interrupt that makes the OS focus on updating the mouse process, before returning to other processes like updating the clock in the screen.

Abstraction #11-hundreds – The OS has so many layers of abstraction it's nuts

- The OS has many layers of abstraction starting with the Kernel – which manages:
 - Multi-threading of a process (true parallel processing on a single core does not exist – but can be faked well by switching tasks imperceptibly fast)
 - Multiple-cores – true parallelisms does exist, e.g. if a CPU has 8 cores – but the OS needs to do special tricks to make sure programs all ‘finish’ in the correct order for data to make sense.
 - Hardware is abstracted as files using Hardware Abstraction Layer
 - Programs are abstracted as processes that can run in ‘parallel’, as threads
 - Processes are given ‘stacks’ of memory in RAM that are theoretically independent of each other program.
 - Task scheduling
 - Interacting with Hardware Via Drivers
 - User accounts and user permissions
 - Files are chunks of memory. Files are wrapped with meta data and special attributes in memory.
 - And on...

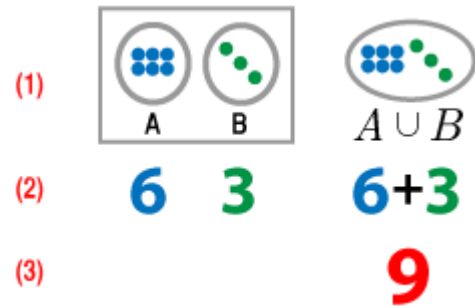
Abstraction #hundreds – thousands – Programming languages and frameworks are build on the OS

- A user writes code “if (1+1 == 2) then print ‘True statement’ ” this is converted to CPU instructions that use both comparison “CMP” and writing to memory via “MOV” and this is what compiling a program from text to binary is.
- Various programming languages save time by inventing things like garbage collection (removing data from the process stack in memory), pointers (virtual locations/address of data in memory), and object oriented design (ability to quickly reuse chunks of code to make copies of similar things)
- Developers develop framework of text-based code to share with other developers to save time – this may be open source frameworks (could be thousands of man-hours of work saved). A developer can then run a single line of code to call something in a framework that is millions of lines long. For example, this is how to grab a webcam picture, and display it in ONLY four lines:

```
import cv2                # import a framework, in this case a Python module
vid = cv2.VideoCapture(0) # define a video capture object, i.e. a connection to your physical webcam
ret, frame = vid.read()   # grab the picture from the webcam
cv2.imshow('frame', frame) # Display the resulting frame on screen
```

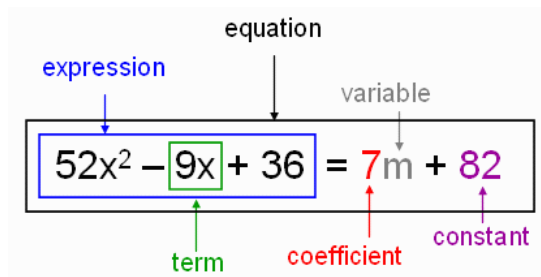

Abstractions in Math

- Set Theory



- Coordinate Transformations

- Algebra



2-dimensional [\[edit\]](#)

Let (x, y) be the standard [Cartesian coordinates](#), and (r, θ) the standard [polar coordinates](#).

To Cartesian coordinates [\[edit\]](#)

From polar coordinates [\[edit\]](#)

$$x = r \cos \theta$$

$$y = r \sin \theta$$

$$\frac{\partial(x, y)}{\partial(r, \theta)} = \begin{bmatrix} \cos \theta & -r \sin \theta \\ \sin \theta & r \cos \theta \end{bmatrix}$$

$$\text{Jacobian} = \det \frac{\partial(x, y)}{\partial(r, \theta)} = r$$

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

Abstractions in Math, cont.

- Einstein Field Equations

The Einstein field equations (EFE) may be written in the form:[\[5\]\[1\]](#)

$$G_{\mu\nu} + \Lambda g_{\mu\nu} = \kappa T_{\mu\nu}$$

where $G_{\mu\nu}$ is the [Einstein tensor](#), $g_{\mu\nu}$ is the [metric tensor](#), $T_{\mu\nu}$ is the [stress–energy tensor](#), Λ is the [cosmological constant](#) and κ is the Einstein gravitational constant.

- Instead of the longer format:

$$\begin{aligned} & \frac{1}{2}g^{\alpha\beta}\partial_\alpha\partial_\mu g_{\beta\nu} + \frac{1}{2}g^{\alpha\beta}\partial_\alpha\partial_\nu g_{\mu\beta} - \frac{1}{2}g^{\alpha\beta}\partial_\alpha\partial_\beta g_{\mu\nu} - \frac{3}{2}g^{\alpha\beta}\partial_\mu\partial_\nu g_{\alpha\beta} - \frac{1}{2}g^{\beta\lambda}g^{\alpha\rho}\partial_\alpha g_{\rho\lambda}\partial_\mu g_{\beta\nu} \\ & - \frac{1}{2}g^{\beta\lambda}g^{\alpha\rho}\partial_\alpha g_{\rho\lambda}\partial_\nu g_{\mu\beta} + \frac{1}{4}g^{\beta\lambda}g^{\alpha\rho}\partial_\nu g_{\alpha\lambda}\partial_\mu g_{\rho\beta} + \frac{1}{4|g|}g^{\alpha\beta}\partial_\beta |g|\partial_\nu g_{\mu\alpha} - \frac{1}{4|g|}g^{\alpha\beta}\partial_\beta |g|\partial_\alpha g_{\mu\nu} \\ & - \frac{1}{4|g|}g^{\alpha\beta}\partial_\beta |g|\partial_\mu g_{\alpha\nu} + \Lambda g_{\mu\nu} = \frac{8\pi G}{c^4}T_{\mu\nu} \end{aligned}$$

ChatGPT

- Prompt -> Lexical parsed keys -> vectorized input -> knowledge graph traversal (relationship nodes and weights) -> floating point weights across a neural network -> vectorized output -> iterative generative diminishing return loops on language -> vectorized output -> text output

Homework -

- Without Googling it – can you craft a byte subtractor using only the logic gates AND, OR, NOT?