



Ένωση Πληροφορικών Ελλάδας

Workshop: «Ας φτιάξουμε τον δικό μας προγραμματιζόμενο μικρο-επεξεργαστή»

Χάρης Γεωργίου (MSc, PhD)

Ένωση Πληροφορικών Ελλάδας

Στόχοι:

- Πρώτος “καθολικός” φορέας εκπροσώπησης πτυχιούχων Πληροφορικής.
- Αρμόδιος φορέας εκπροσώπησης επαγγελματιών Πληροφορικής.
- Αρμόδιος επιστημονικός “συμβουλευτικός” φορέας για το Δημόσιο.
- Αρωγός της Εθνικής Ψηφιακής Στρατηγικής & Παιδείας της χώρας.

<https://www.epe.org.gr>



Τομείς παρέμβασης

Ποιοι είναι οι κύριοι τομείς παρεμβάσεων της ΕΠΕ;

- ① Εθνική Ψηφιακή Στρατηγική & Οικονομία
- ② Εργασιακά (ΤΠΕ), Δημόσιος & ιδιωτικός τομέας
- ③ Παιδεία (Α', Β', Γ')
- ④ Έρευνα & Τεχνολογία
- ⑤ Έργα & υπηρεσίες ΤΠΕ
- ⑥ Ασφάλεια συστημάτων & δεδομένων
- ⑦ Ανοικτά συστήματα & πρότυπα
- ⑧ Χρήση ΕΛ/ΛΑΚ
- ⑨ Πνευματικά δικαιώματα
- ⑩ Κώδικας Δεοντολογίας (ΤΠΕ)
- ⑪ Κοινωνική μέριμνα (ICT4D)





Harris Georgiou (MSc, PhD) – <https://github.com/xgeorgio/info>

- R&D: Associate post-doc researcher and lecturer with the University Athens (NKUA) and University of Piraeus (UniPi)
- Consultant in Medical Imaging, Machine Learning, Data Analytics, Signal Processing, Process Optimization, Dynamic Systems, Complexity & Emergent A.I., Game Theory
- HRTA member since 2010, LEAR / scientific advisor
- HRTA field operator (USAR, scuba diver)
- Wilderness first aid, paediatric (child/infant)
- Humanitarian aid & disaster relief in Ghana, Lesvos, Piraeus
- Support of unaccomp. minors, teacher in community schools
- Streetwork training, psychological first aid & victim support
- 3+4 books, 220+ scientific papers/articles (and 6 marathons)

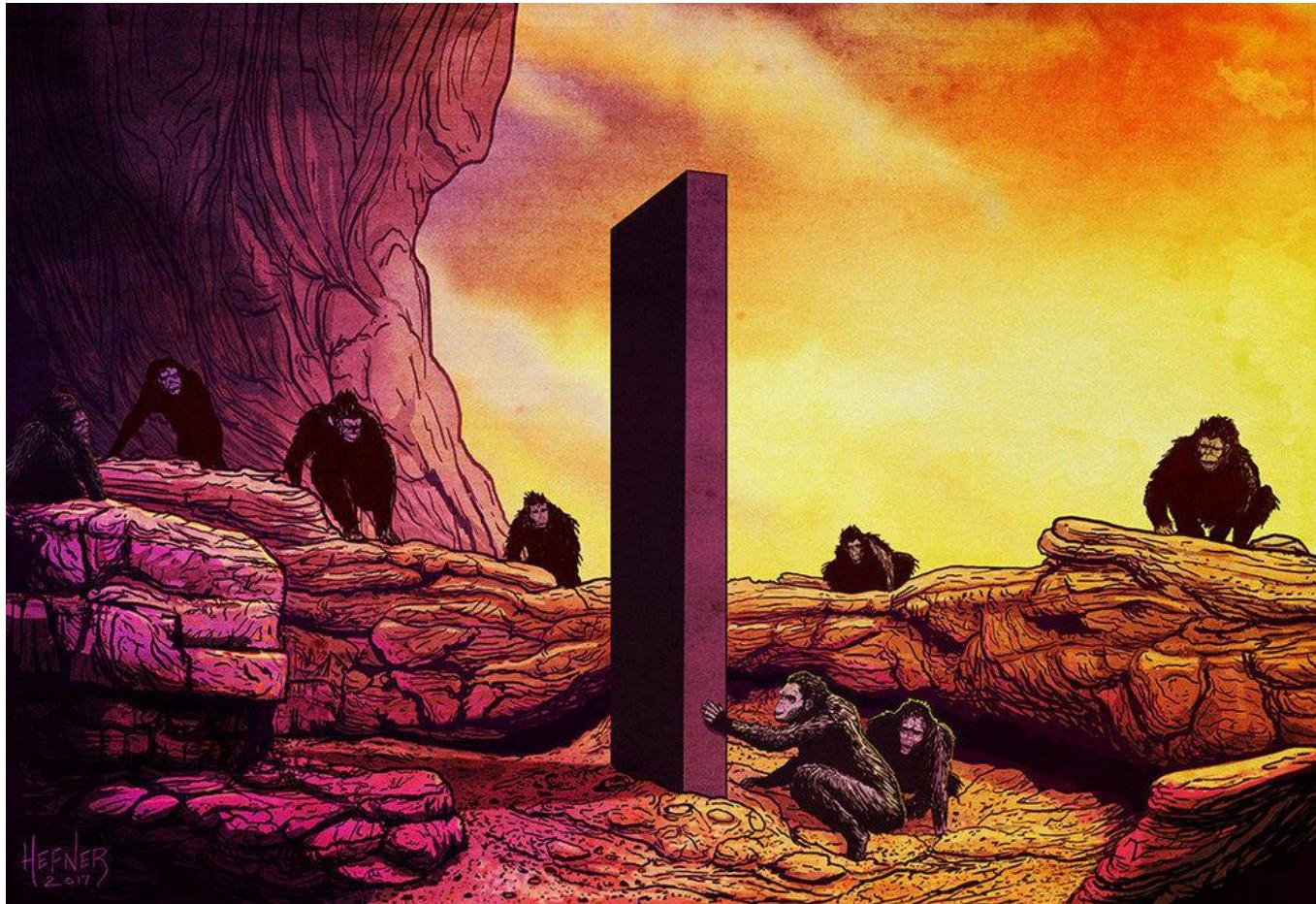
Επισκόπηση

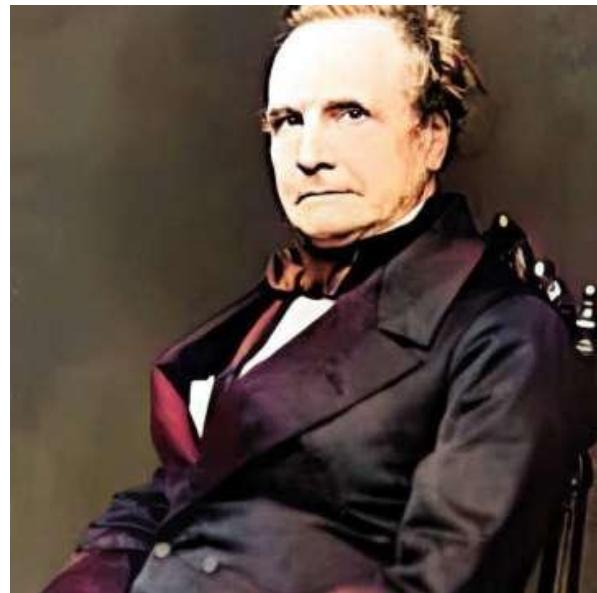
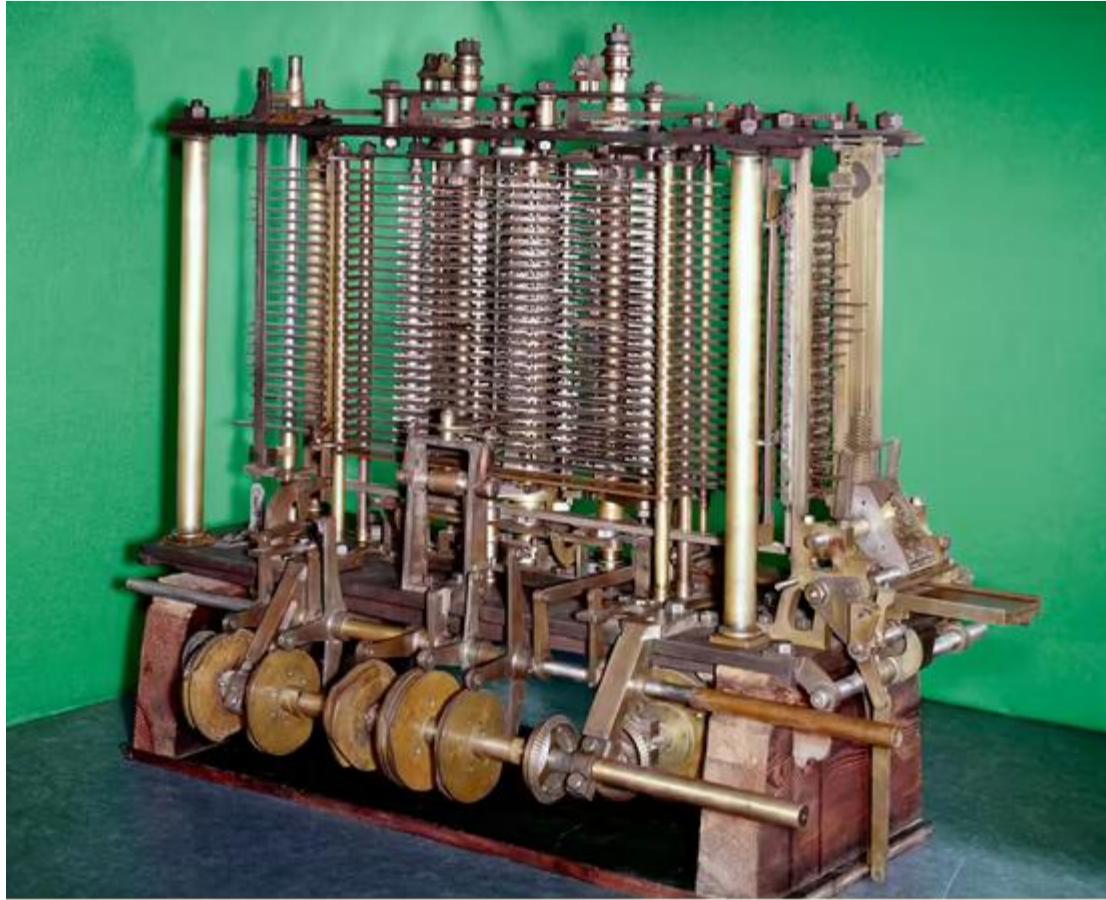
- Περιεχόμενα:
 - Μέρος I: Αριθμητική και Λογική στις μηχανές
 - Μέρος II: Προγραμματισμός μικρο-επεξεργαστών
- Σχετικό υλικό:
 - «Από τον άβακα στους σύγχρονους επεξεργαστές: Χτίζοντας τους Η/Υ (Μέρος I)», Χ. Γεωργίου, Δ' κύκλος ανοικτών μαθημάτων ΕΠΕ – <https://youtu.be/p3TgG0iQGjQ>
 - «Από τον άβακα στους σύγχρονους επεξεργαστές: Χτίζοντας τους Η/Υ (Μέρος II)», Χ. Γεωργίου, Δ' κύκλος ανοικτών μαθημάτων ΕΠΕ – <https://youtu.be/hfr1fuKVe8o>
 - «Hello x86: Low-level assembly coding for the 8086», @ApneaCoding – <https://youtu.be/tF16xTbd42w>
 - «BAM neural network in Arduino», @ApneaCoding – <https://youtu.be/RkM-rpSVD4I>

Μέρος Ι: Αριθμητική και Λογική στις μηχανές

1. Οι απαρχές των υπολογιστικών μηχανών
2. Ψηφιακά ηλεκτρονικά
3. Τεχνολογίες μικρο-επεξεργαστών
4. Προγραμματιζόμενοι μικρο-ελεγκτές







This analytical engine, conceived by Charles Babbage in 1834, was designed to calculate any mathematical formula and to have even higher powers of analysis than his original difference engine. This portion of the mill was under construction at the time of his death. SSPL/GETTY IMAGES

Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Data										Working Variables.										Result Variables.			
					δV_2	δV_3	δV_4	δV_5	δV_6	δV_7	δV_8	δV_9	δV_{10}	δV_{11}	δV_{12}	δV_{13}	δV_{14}	δV_{15}	δV_{16}	δV_{17}	δV_{18}	δV_{19}	δV_{20}	δV_{21}	δV_{22}	δV_{23}		
1	\times	$\delta V_2 \times \delta V_3$	$\delta V_4, \delta V_5, \delta V_6$	$\{ \delta V_2 \\ \delta V_3 \\ \delta V_4 \\ \delta V_5 \\ \delta V_6 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
2	$-$	$\delta V_4 - \delta V_1$	δV_2	$\{ \delta V_4 \\ \delta V_1 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
3	$+$	$\delta V_5 + \delta V_1$	δV_2	$\{ \delta V_5 \\ \delta V_1 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
4	$+$	$\delta V_5 + \delta V_4$	δV_{11}	$\{ \delta V_5 \\ \delta V_4 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
5	$+$	$\delta V_{11} + \delta V_2$	δV_{11}	$\{ \delta V_{11} \\ \delta V_2 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
6	$-$	$\delta V_{12} - \delta V_2$	δV_{12}	$\{ \delta V_{12} \\ \delta V_2 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
7	$-$	$\delta V_2 - \delta V_1$	δV_{10}	$\{ \delta V_2 \\ \delta V_1 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
8	$+$	$\delta V_2 + \delta V_7$	δV_7	$\{ \delta V_2 \\ \delta V_7 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
9	$+$	$\delta V_6 + \delta V_7$	δV_{11}	$\{ \delta V_6 \\ \delta V_7 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
10	\times	$\delta V_{21} \times \delta V_{11}$	δV_{12}	$\{ \delta V_{21} \\ \delta V_{11} \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
11	$+$	$\delta V_{12} + \delta V_{13}$	δV_{12}	$\{ \delta V_{12} \\ \delta V_{13} \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
12	$-$	$\delta V_{10} - \delta V_1$	δV_{10}	$\{ \delta V_{10} \\ \delta V_1 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
13	$-$	$\delta V_6 - \delta V_1$	δV_6	$\{ \delta V_6 \\ \delta V_1 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
14	$+$	$\delta V_1 + \delta V_7$	δV_7	$\{ \delta V_1 \\ \delta V_7 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
15	$+$	$\delta V_6 + \delta V_7$	δV_8	$\{ \delta V_6 \\ \delta V_7 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
16	\times	$\delta V_8 \times \delta V_{11}$	δV_{11}	$\{ \delta V_8 \\ \delta V_{11} \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
17	$-$	$\delta V_6 - \delta V_1$	δV_6	$\{ \delta V_6 \\ \delta V_1 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
18	$+$	$\delta V_1 + \delta V_7$	δV_7	$\{ \delta V_1 \\ \delta V_7 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
19	$+$	$\delta V_6 + \delta V_7$	δV_9	$\{ \delta V_6 \\ \delta V_7 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
20	\times	$\delta V_{21} \times \delta V_{11}$	δV_{11}	$\{ \delta V_{21} \\ \delta V_{11} \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
21	\times	$\delta V_{21} \times \delta V_{11}$	δV_{12}	$\{ \delta V_{21} \\ \delta V_{11} \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
22	$+$	$\delta V_{12} + \delta V_{13}$	δV_{12}	$\{ \delta V_{12} \\ \delta V_{13} \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
23	$-$	$\delta V_{10} - \delta V_1$	δV_{10}	$\{ \delta V_{10} \\ \delta V_1 \}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			



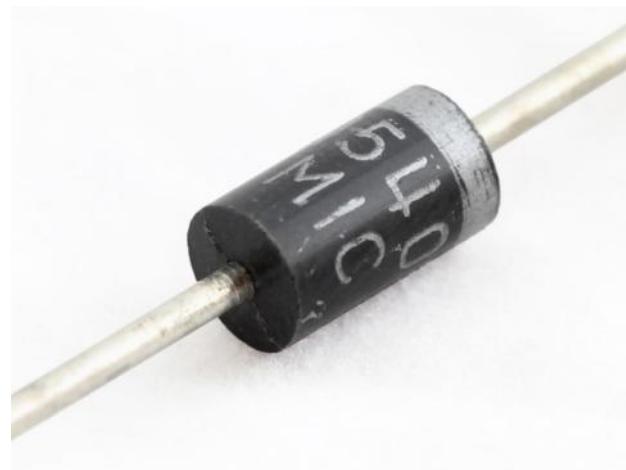
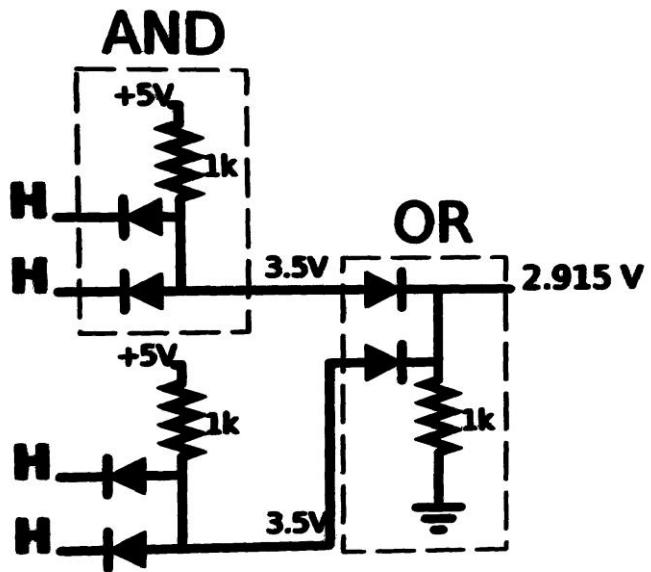
Lovelace

Diagram for the computation by the Engine of the

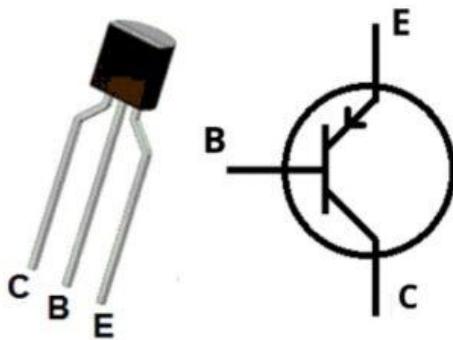
Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data		3V_1	3V_2	3V_3	3V_4	3V_5	3V_6
						3V_1	3V_2						
1	\times	${}^3V_2 \times {}^3V_3$	${}^3V_4, {}^3V_5, {}^3V_6$	$\left\{ \begin{array}{l} {}^3V_7 = {}^1V_7 \\ {}^3V_3 = {}^1V_3 \end{array} \right\}$	$= 2n$	1	2
2	-	${}^3V_4 - {}^3V_1$	3V_4	$\left\{ \begin{array}{l} {}^3V_2 = {}^2V_4 \\ {}^3V_1 = {}^1V_2 \end{array} \right\}$	$= 2n - 1$	1
3	+	${}^3V_3 + {}^3V_1$	3V_3	$\left\{ \begin{array}{l} {}^3V_3 = {}^2V_3 \\ {}^3V_1 = {}^1V_3 \end{array} \right\}$	$= 2n + 1$	1
4	\div	${}^3V_3 \div {}^3V_4$	${}^3V_{11}$	$\left\{ \begin{array}{l} {}^3V_3 = {}^0V_3 \\ {}^3V_4 = {}^0V_4 \end{array} \right\}$	$= \frac{2n - 1}{2n + 1}$	0	0
5	\div	${}^3V_{11} \div {}^3V_2$	${}^3V_{11}$	$\left\{ \begin{array}{l} {}^3V_{10} = {}^2V_{21} \\ {}^3V_2 = {}^1V_2 \end{array} \right\}$	$= \frac{1}{2} \cdot \frac{2n - 1}{2n + 1}$...	2
6	-	${}^3V_{13} - {}^3V_{11}$	${}^3V_{12}$	$\left\{ \begin{array}{l} {}^3V_{10} = {}^0V_{21} \\ {}^3V_{13} = {}^1V_{13} \end{array} \right\}$	$= -\frac{1}{2} \cdot \frac{2n - 1}{2n + 1} = \lambda_0$
7	-	${}^3V_3 - {}^3V_1$	${}^3V_{30}$	$\left\{ \begin{array}{l} {}^3V_2 = {}^1V_3 \\ {}^3V_1 = {}^1V_1 \end{array} \right\}$	$= n - 1 (= 3)$	1	...	8

	A	B	C	D	E	F	G	H	I
				Score	P=H	Rid	Length-Recd	Width-Recd	Weight
				Mean					
				Median					
				SD					
1	Pass	0.00	0.15	-0.0253	0.207389				
2	Fail	1.00	0.25	-0.168279	-0.067733				
3	Total	0.5	0.15	-0.1595	0.008818				
4	Length	44	3.76007794	3.300000	0.379613				
5	Width	69	3.76177778	-0.871407	-0.471617				
6	Height	203	3.7662242	2.300000	1.373701				
7	Area	1	1	0.333333	0.148000				
8	Diag	0.00	0.17	0.771000	0.117981				
9	Peri	1.00	0.27	1.0000	0.036213				
10	Std	0.00	0.17	-0.040	-0.126300				
11	Length	45.00	3.76007794	3.300000	0.379613				
12	Width	69.00	3.76177778	-0.871407	-0.471617				
13	Height	203.00	3.7662242	2.300000	1.373701				
14	Area	1.00	1	0.771000	0.148000				
15	Diag	0.00	0.17	0.771000	0.117981				
16	Peri	1.00	0.27	1.0000	0.036213				
17	Std	0.00	0.17	0.119000	-0.077013				
18	Length	44.60	3.76007794	-0.100	-0.069515				
19	Width	69.00	3.76177778	0.500000	-0.259811				
20	Height	203.00	3.7662242	2.757120	0.500362				
21	Area	1.00	1	0.523000	1.503619				
22	Diag	0.00	0.17	0.771000	-0.112889				

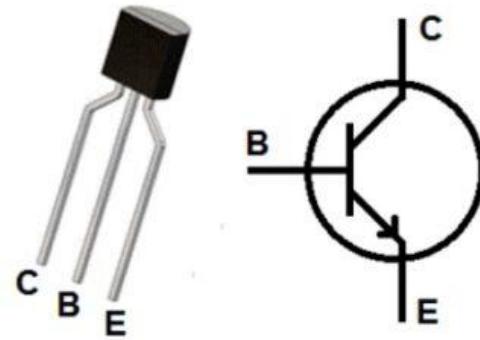
VOLTAGE DROPS ACROSS DIODES WOULD BE A FURTHER CONSTRAINT



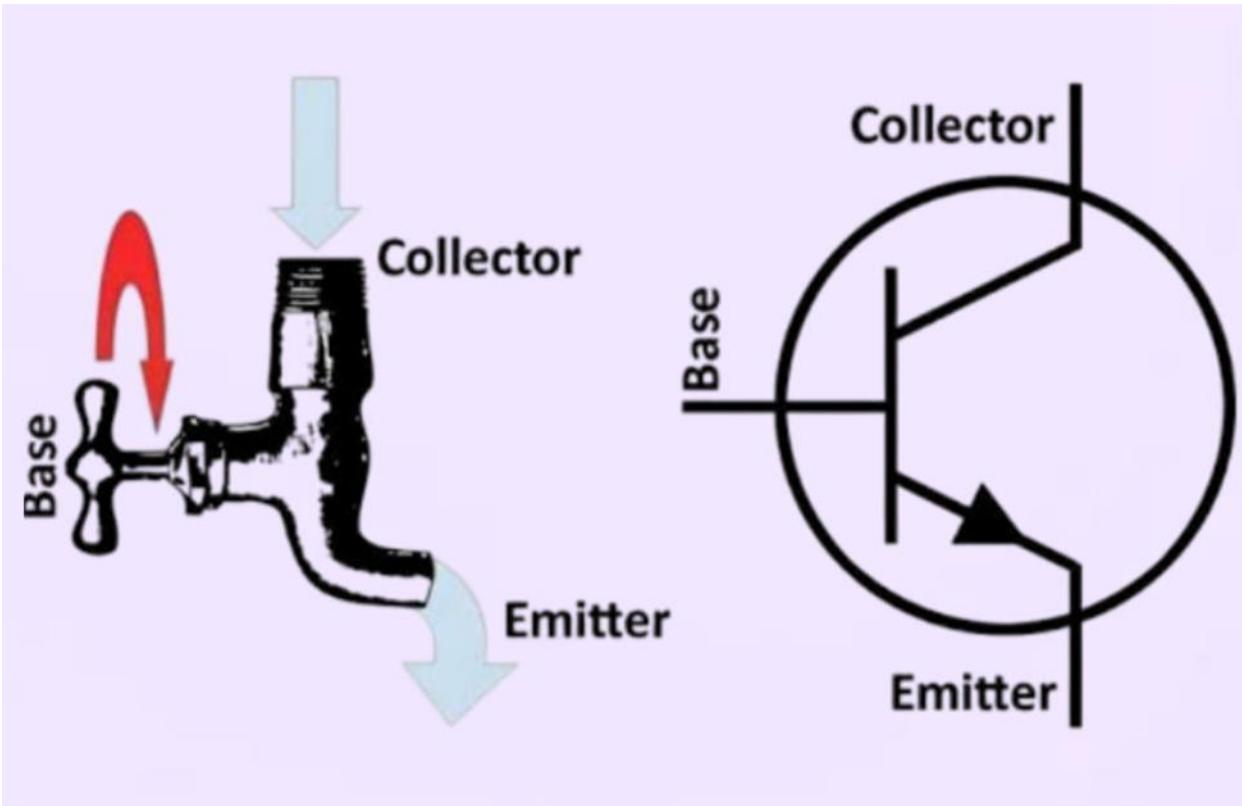
PNP Transistor



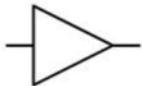
NPN Transistor



TRANSISTOR SYMBOL



Buffer



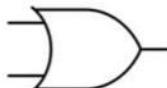
Input	Output
0	0
1	1

AND



A	B	Output
0	0	0
1	0	0
0	1	0
1	1	1

OR



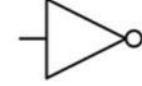
A	B	Output
0	0	0
1	0	1
0	1	1
1	1	1

XOR



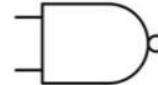
A	B	Output
0	0	0
1	0	1
0	1	1
1	1	0

Inverter



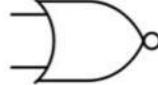
Input	Output
0	1
1	0

NAND



A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

NOR



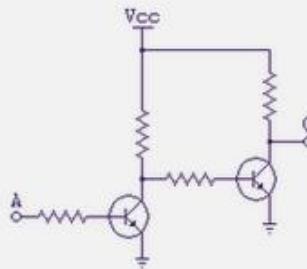
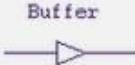
A	B	Output
0	0	1
1	0	0
0	1	0
1	1	0

XNOR

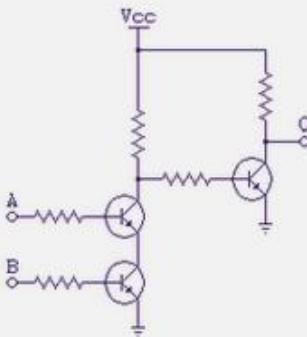
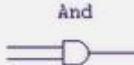


A	B	Output
0	0	1
1	0	0
0	1	0
1	1	1

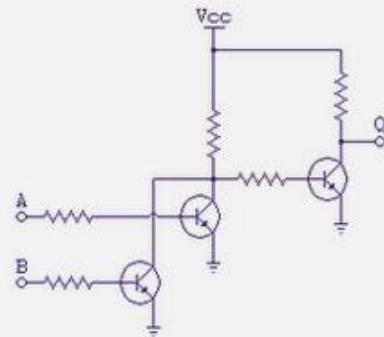
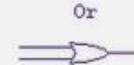
Buffer



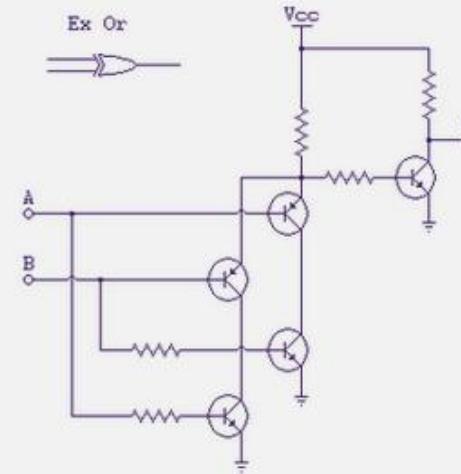
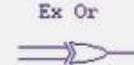
And



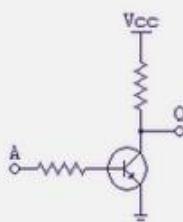
Or



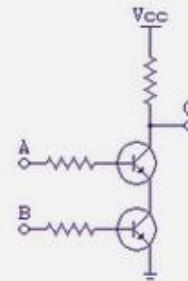
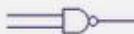
Ex Or



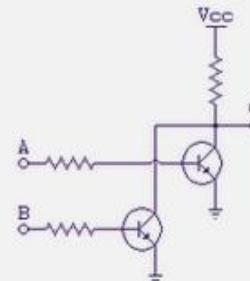
Inverter Not



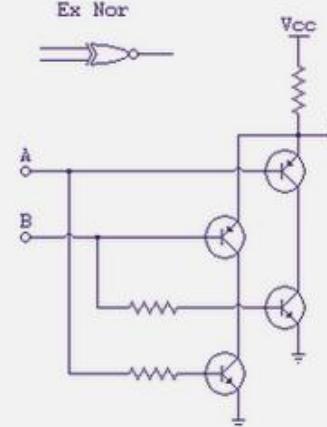
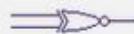
Nand

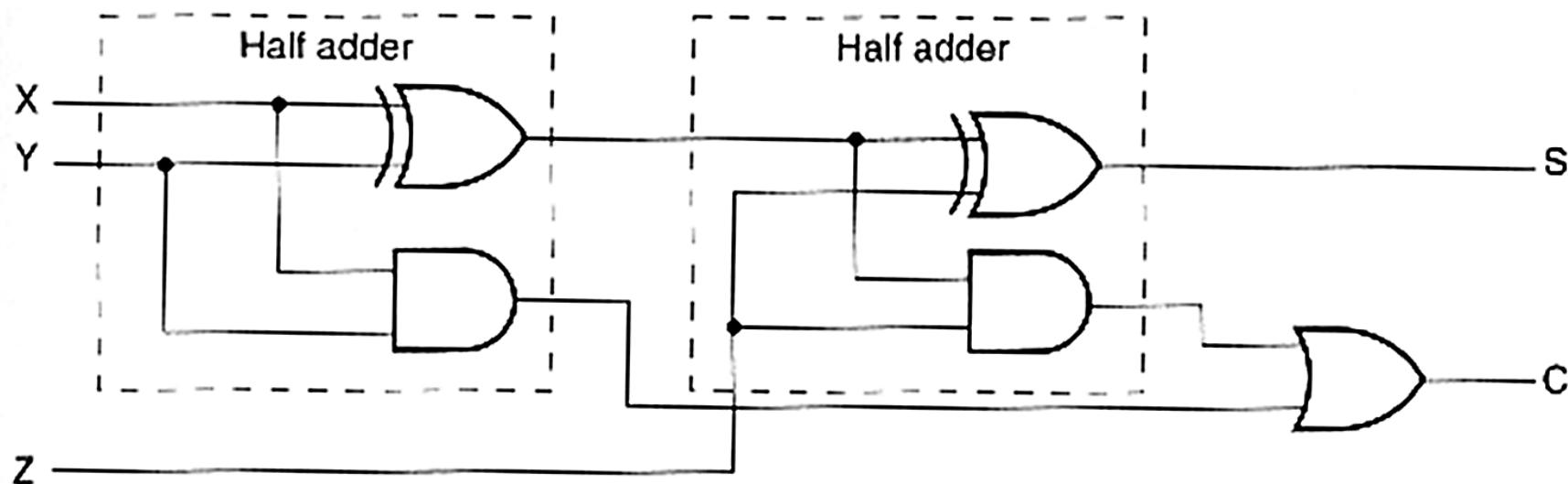


Nor

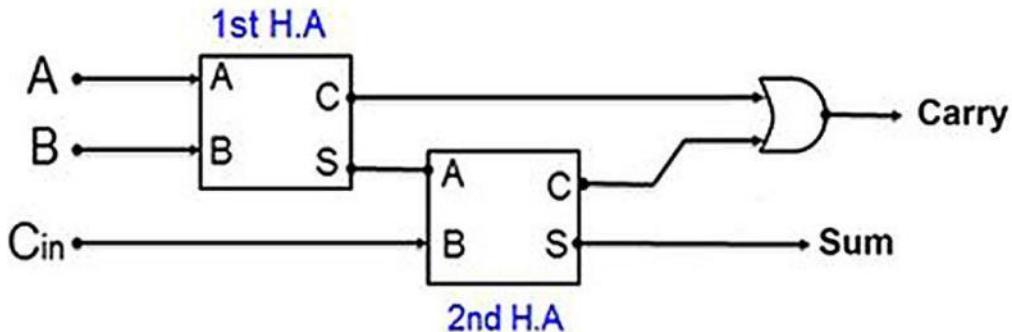


Ex Nor



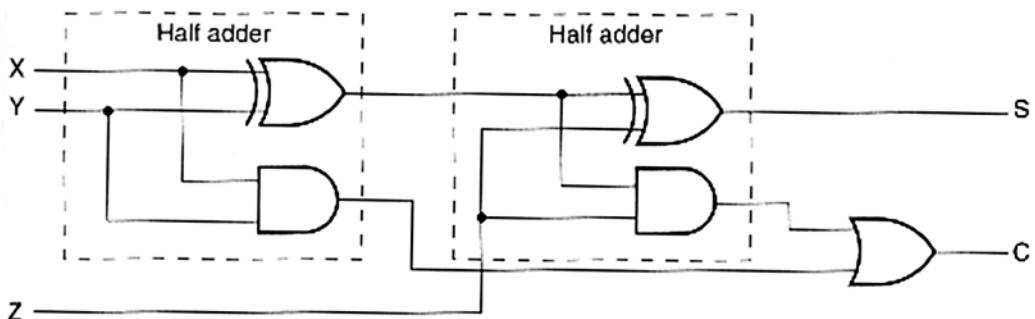


(a)



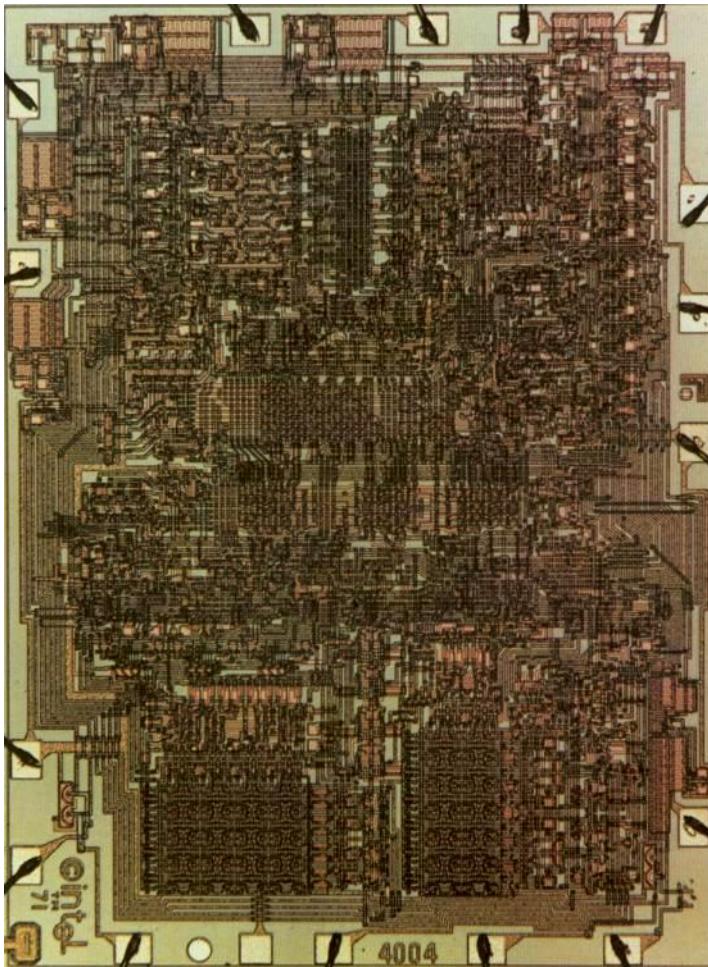
(b)

Input			Output	
A	B	Cin	Sum	Carry
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





Let's Build a Vacuum Tube 1-Bit Full Adder



INTEL 4004

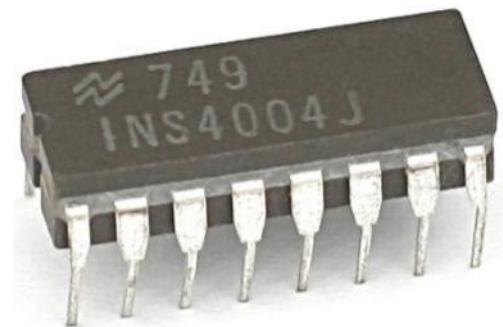
16-pin CPU for desktop calculators

4-bit addressing / instruction set

@ 740 kHz clock

Available: 15-Nov-1971

1st ever single-chip CPU



8088

8-Bit Microprocessor CPU
iAPX86 Family
FINAL

DISTINCTIVE CHARACTERISTICS

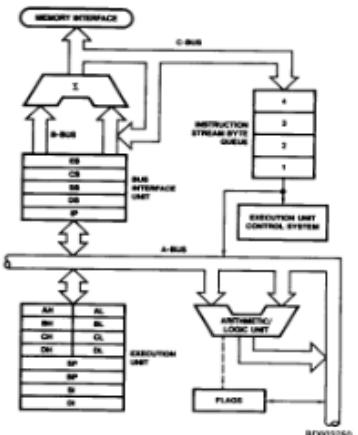
- 8-bit data bus, 16-bit internal architecture
- Directly addresses 1 Mbyte of memory
- Software compatible with 8086 CPU
- Byte, word, and block operations
- 24 operand addressing modes
- Powerful instruction set
- Efficient high level language implementation
- Three speed options: 5MHz 8088
8MHz 8088-2
10MHz 8088-1

GENERAL DESCRIPTION

The 8088 CPU is an 8-bit processor designed around the 8086 internal structure. Most functions of the 8088 are identical to the equivalent 8086 functions. The pinout is slightly different. The 8088 handles the external bus the same way the 8086 does, but it handles only 8 bits at a time. Sixteen-bit words are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time.

The 8088 is made with N-channel silicon gate technology and is packaged in a 40-pin Plastic dip, CERDIP or Plastic Leaded Chip Carrier.

BLOCK DIAGRAM



INSTRUCTION SET SUMMARY (continued)

ARITHMETIC

ADD = Add

Reg/memory with register to either

Immediate to register / memory

Immediate to accumulator

7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
0 0 0 0 0 d w	mod reg r/m		
1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w = 01
0 0 0 0 0 1 0 w	data	data if w = 1	

ADC = Add with carry:

Reg/memory with register to either

Immediate to register/memory

Immediate to accumulator

0 0 0 1 0 0 d w	mod reg r/m		
1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s:w = 01
0 0 0 1 0 1 0 w	data	data if w = 1	

INC = Increment:

Register/memory

Register

AAA = ASCII adjust for add

DAA = Decimal adjust for add

1 1 1 1 1 1 1 w	mod 0 0 0 r/m		
0 1 0 0 0 reg			
0 0 1 1 0 1 1 1			
0 0 1 0 0 1 1 1			

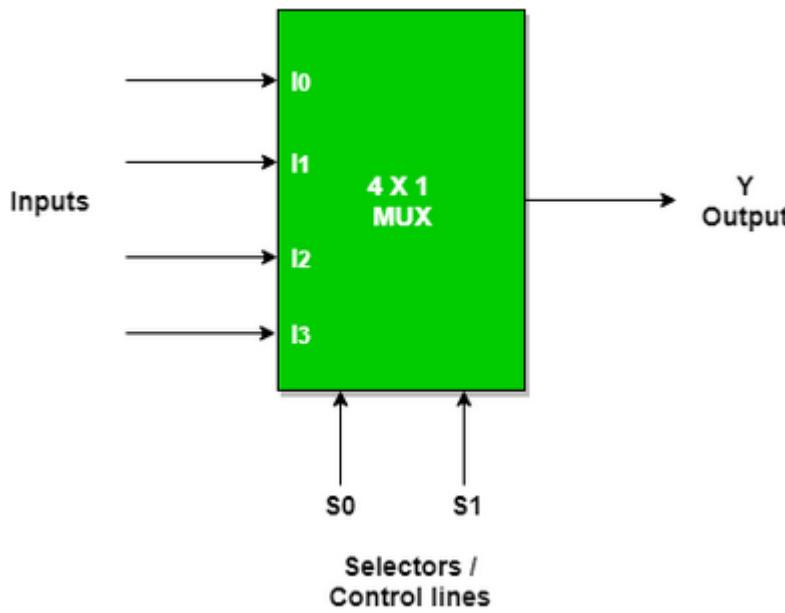
SUB = Subtract:

Reg/memory and register to either

Immediate from register/memory

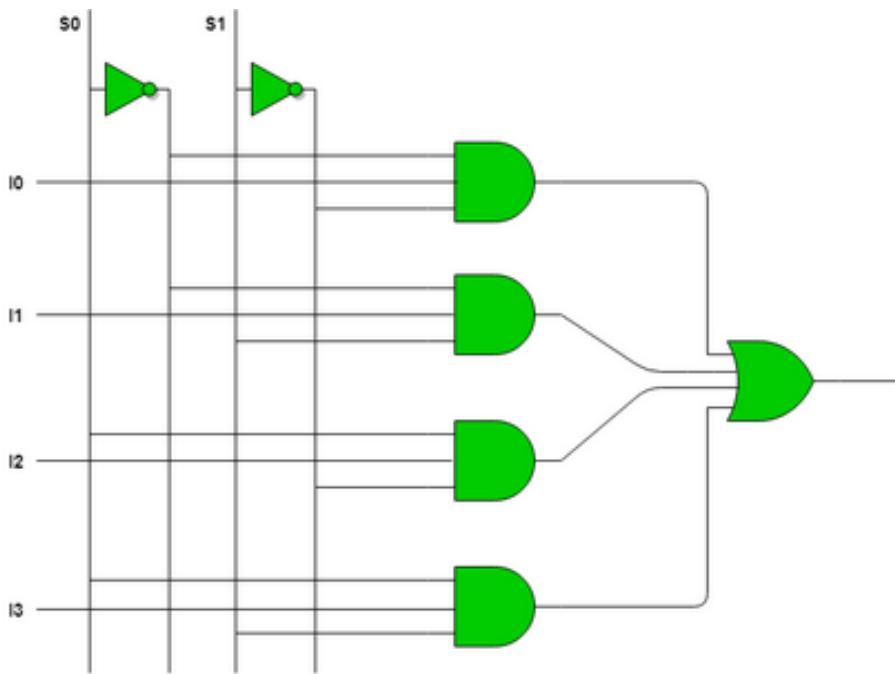
Immediate from accumulator

0 0 1 0 1 0 d w	mod reg r/m		
1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s:w = 01
0 0 1 0 1 1 0 w	data	data if w = 1	

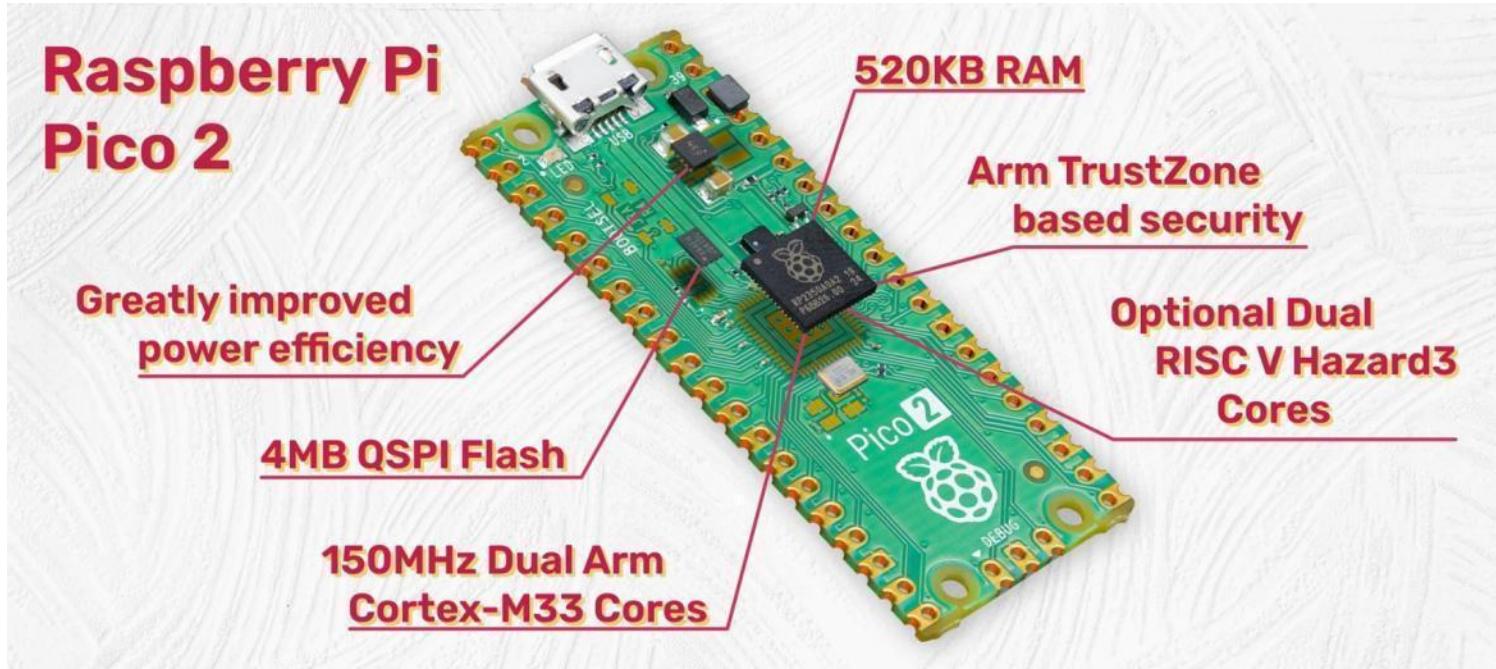


S0	S1	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

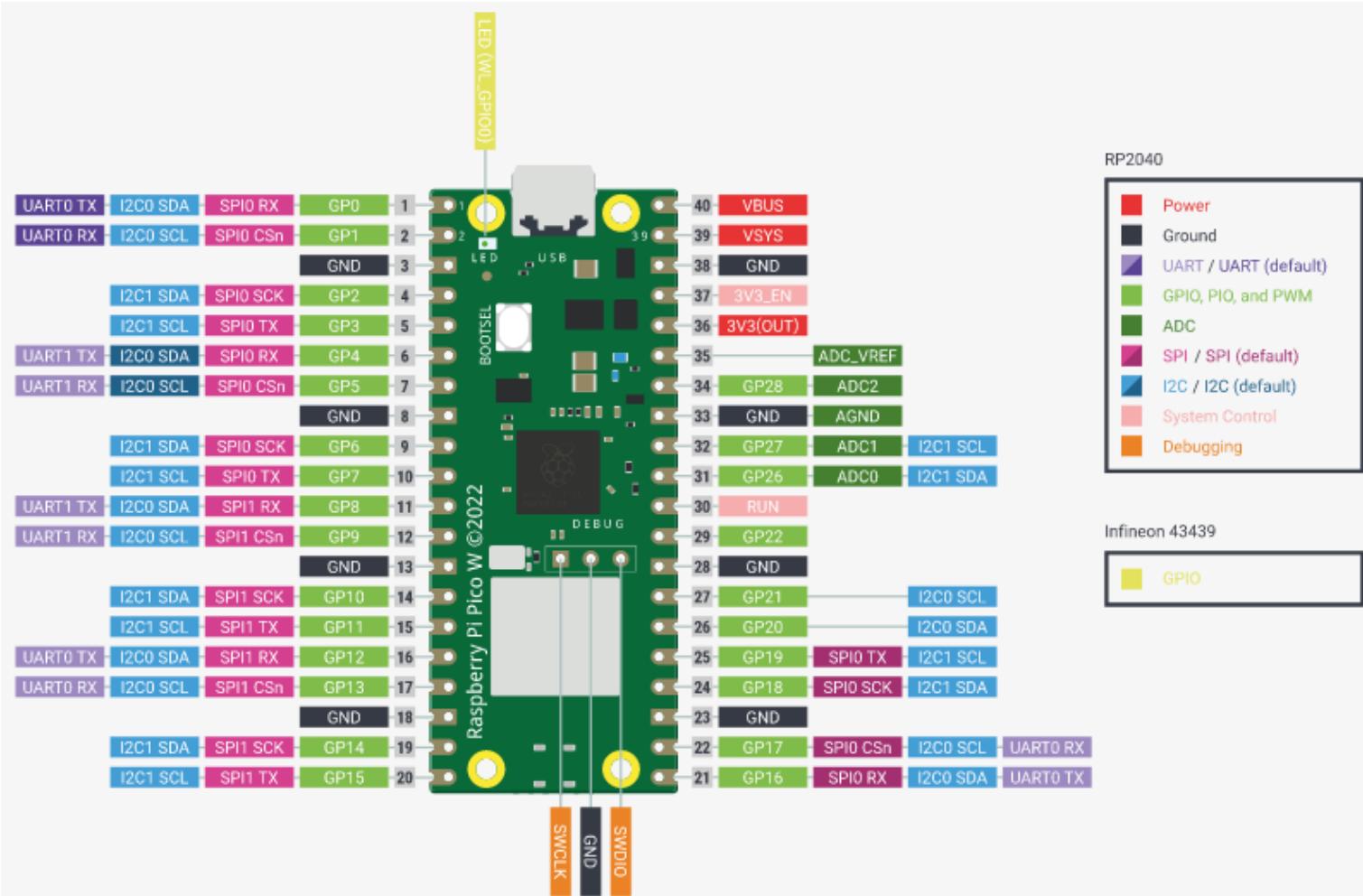
- When $S1S0=00$, the input $I0$ is selected.
- When $S1S0=01$, the input $I1$ is selected.
- When $S1S0=10$, the input $I2$ is selected.
- When $S1S0=11$, the input $I3$ is selected.

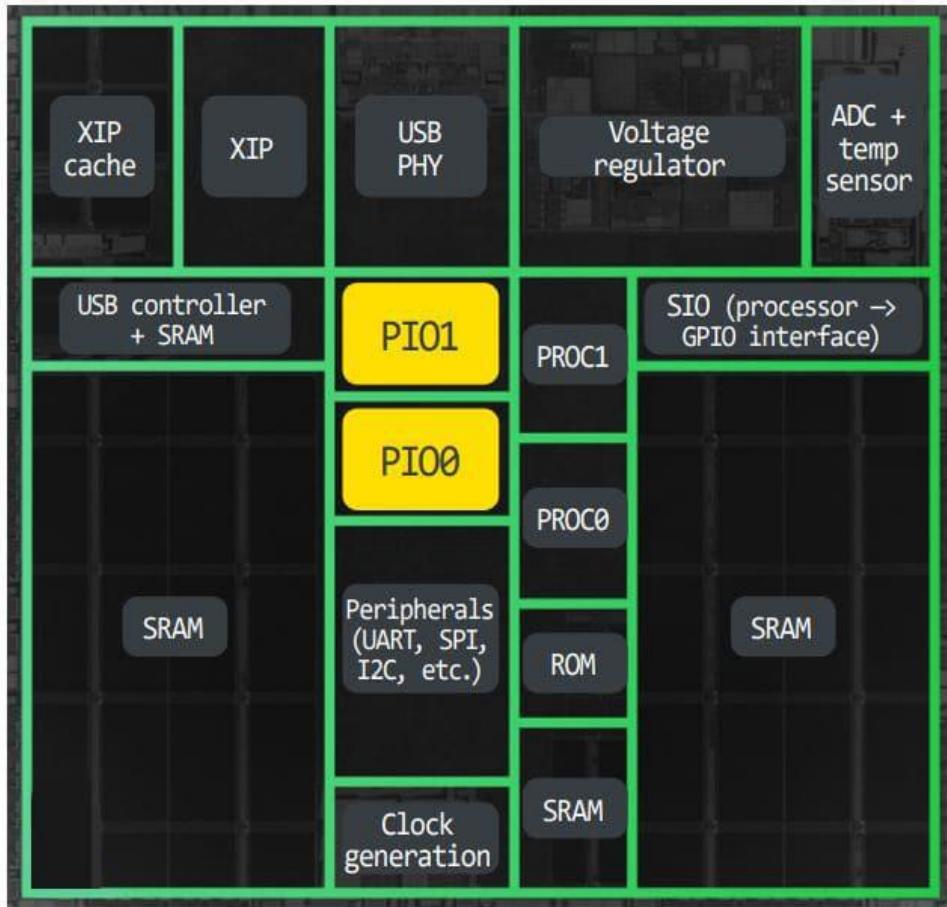
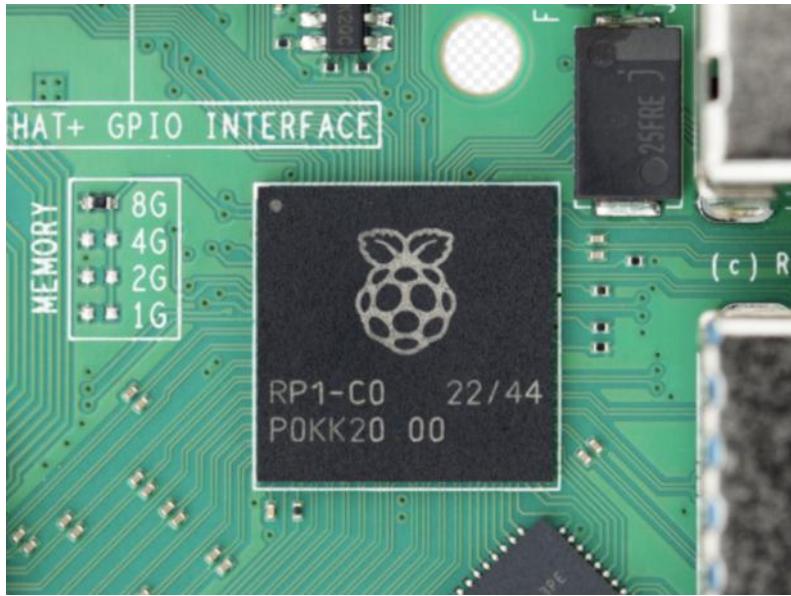


Raspberry Pi Pico 2



- 520KB RAM \approx 133,120 “synapse” weights (32-bit FP variables)
- FP arithmetic (FxMadds) \approx 300 per sample @ 44100 khz
- FP benchmark \approx 1.5 MFLOPS @ 133 MHz (\approx 11 Hz “brain”)
- *RPi 5 (SBC): 4 cores / x16 memory / x37 speed*



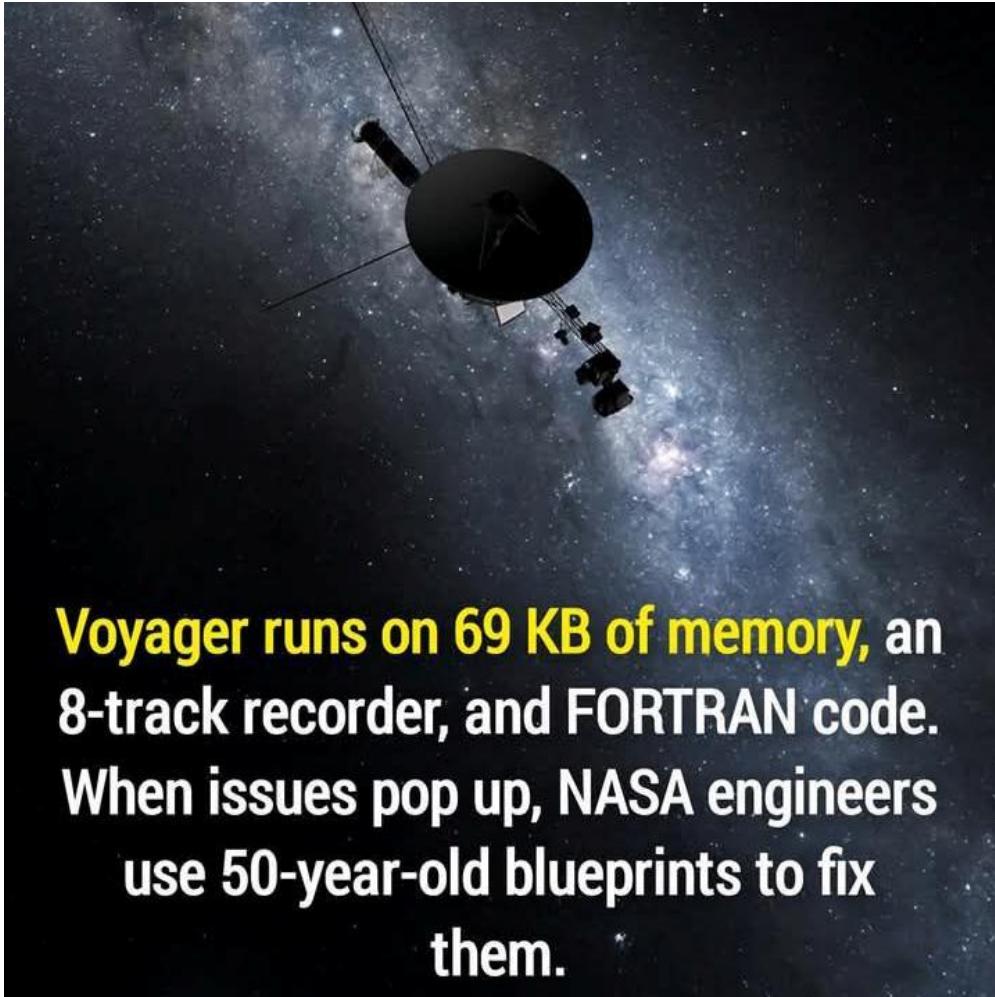


PIO Instruction Set

Mnemonic	Short Description
JMP	Set program counter to specified address if specified condition is true. Otherwise, no operation.
WAIT	Stall until some condition is met.
IN	Shift specified number of bits from specified source into the Input Shift Register.
OUT	Shift specified amount of bits out of the Output Shift Register and write those bits to specified destination.
PUSH	Push the contents of the Input Shift Register into the RX FIFO as a single 32-bit word. Clear Input Shift Register to all-zeroes.
PULL	Load a 32-bit word from the TX FIFO into the Output Shift Register.
MOV	Copy data from the specified source to the specified destination.
IRQ	Set or clear the IRQ flag selected by the specified index argument.
SET	Write specified immediate value to specified destination.

The PIOs' instruction set consists of just nine different instructions that are highly specialized on bit manipulation.

```
pull
mov x, ~osr
pull
mov y, osr
jmp test           ; this loop is equivalent to the following C code:
incr:             ; while (y--)
    jmp x-- test   ;     x--;
test:              ; This has the effect of subtracting y from x, eventually.
    jmp y-- incr
    mov isr, ~x
    push
```



**Voyager runs on 69 KB of memory, an
8-track recorder, and FORTRAN code.
When issues pop up, NASA engineers
use 50-year-old blueprints to fix
them.**

Μέρος II: Προγραμματισμός μικρο-επεξεργαστών

1. Ρωτώντας το ChatGPT για assembly... (2024)
2. Άλλη μια προσπάθεια με το ChatGPT... (2025)
3. Φτιάχνοντας κώδικα «με το χέρι»
4. Γιατί έχει (ακόμα) σημασία



The screenshot shows the Thonny IDE interface for MicroPython development. On the left, the file tree shows a project structure with files like `boot.py`, `main.py`, and `webrepl_cfg.py`. The main area displays the contents of `boot.py`:

```
1 # This file is executed on every boot (including wake
2 import time
3 import esp
4 import webrepl
5 from mything import wireless_network
6
7
8 def reboot():
9     machine.reset()
10
11 def webpython():
12     webrepl.start()
13
14 def no_debug():
15     esp.osdebug(None)
16
17
18 WIFI_SSID = 'Covid-19-Laboratory'
19 WIFI_PASSWORD = 'topsecret'
20
21 time.sleep(1)
22 wifi = wireless_network()
23 wifi.connect(WIFI_SSID, WIFI_PASSWORD)
24 webpython()
25
```

To the right, a shell window is open, showing the MicroPython REPL connected to an ESP32. The shell provides basic help, network configuration examples, control commands, and interrupt information.

```
WebREPL connected
>>> help()
Welcome to MicroPython on the ESP32!
For generic online docs please visit https://docs.micropython.org/
For access to the hardware use the 'machine' module:
import machine
pin12 = machine.Pin(12, machine.Pin.OUT)
pin12.value(1)
pin13 = machine.Pin(13, machine.Pin.IN, machine.Pin.PULL_UP)
print(pin13.value())
i2c = machine.I2C(scl=machine.Pin(21), sda=machine.Pin(22))
i2c.scan()
i2c.writeto(addr, b'i234')
i2c.readfrom(addr, 4)

Basic WiFi configuration:
import network
sta_if = network.WLAN(network.STA_IF); sta_if.active(True)
sta_if.scan()                                     # Scan for available access points
sta_if.connect("<AP_name>", "<password>") # Connect to an AP
sta_if.isconnected()                             # Check for successful connection

Control commands:
CTRL-A      -- on a blank line, enter raw REPL mode
CTRL-B      -- on a blank line, enter normal REPL mode
CTRL-C      -- interrupt a running program
CTRL-D      -- on a blank line, do a soft reset of the board
CTRL-E      -- on a blank line, enter paste mode

For further help on a specific object, type help(obj)
For a list of available modules, type help('modules')

>>>
```

OneCompiler

HelloWorld.asm

Assembly Hello World!

+ AI NEW ASSEMBLY ▾ RUN ▶

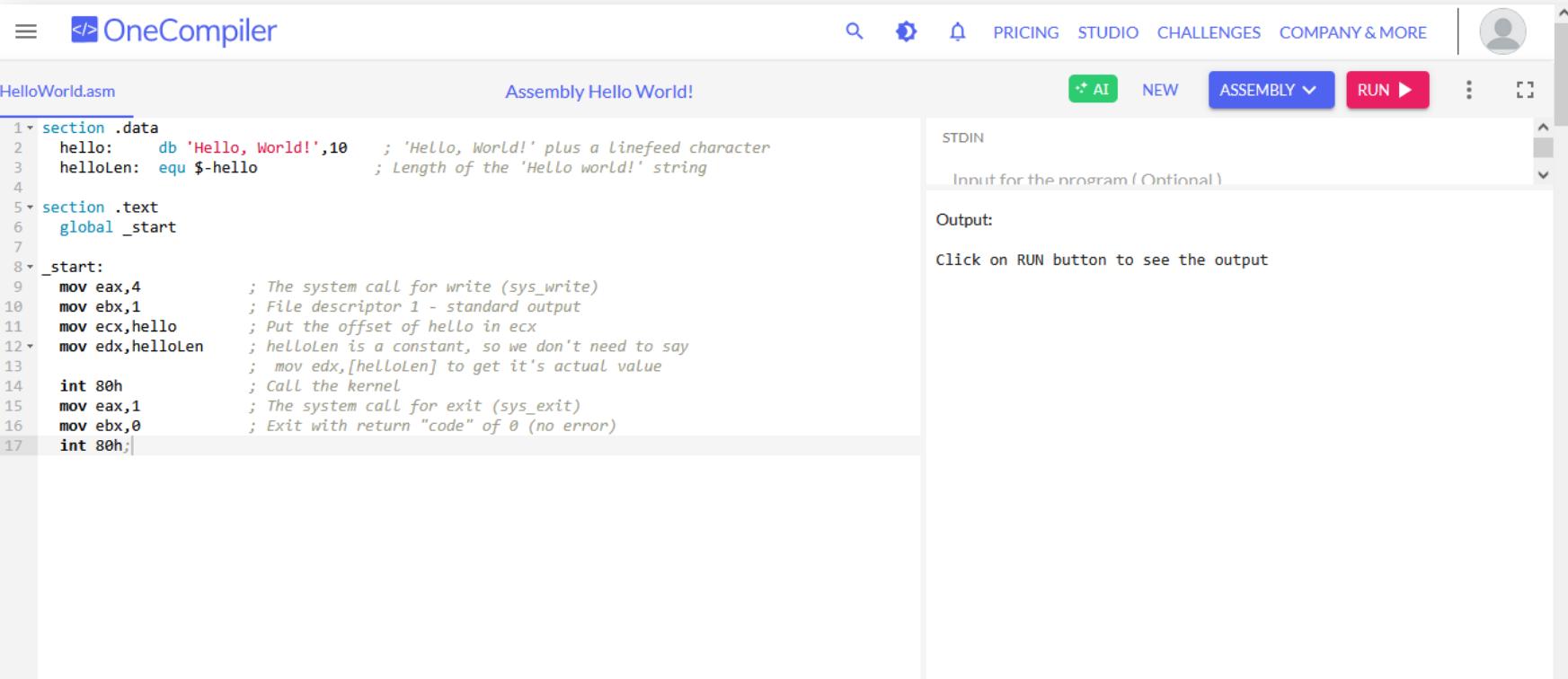
```
1 section .data
2 hello: db 'Hello, World!',10 ; 'Hello, World!' plus a linefeed character
3 helloLen: equ $-hello ; Length of the 'Hello world!' string
4
5 section .text
6 global _start
7
8 _start:
9 mov eax,4 ; The system call for write (sys_write)
10 mov ebx,1 ; File descriptor 1 - standard output
11 mov ecx,hello ; Put the offset of hello in ecx
12 mov edx,helloLen ; helloLen is a constant, so we don't need to say
13 ; mov edx,[helloLen] to get it's actual value
14 int 80h ; Call the kernel
15 mov eax,1 ; The system call for exit (sys_exit)
16 mov ebx,0 ; Exit with return "code" of 0 (no error)
17 int 80h;
```

STDIN

Input for the program (Optional)

Output:

Click on RUN button to see the output





```
In [1]: # setup the matplotlib graphics library and configure it to show
# figures inline in the notebook
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

```
In [2]: # make qutip available in the rest of the notebook
from qutip import *

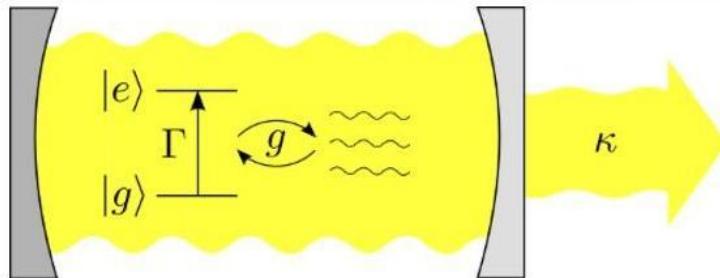
from IPython.display import Image
```

Introduction and model

Consider a single atom coupled to a single cavity mode, as illustrated in the figure below. If there atom excitation rate Γ exceeds the relaxation rate, a population inversion can occur in the atom, and if coupled to the cavity the atom can then act as a photon pump on the cavity.

```
In [3]: Image(filename='images/schematic-lasing-model.png')
```

Out[3]:



ChatGPT



Examples

"Explain quantum computing in simple terms" →



Capabilities

"Got any creative ideas for a 10 year old's birthday?" →



Limitations

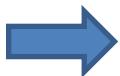
May occasionally generate incorrect information

"How do I make an HTTP request in Javascript?" →

May occasionally produce harmful instructions or biased content

Trained to decline inappropriate requests

Limited knowledge of world and events after 2021



Write a routine in assembly for raspberry pi pico using pure RP2040 PIO assembly for multiplication of two 32 bit integers in a PIO core.

To implement **32-bit integer multiplication using RP2040 PIO assembly**, we must work within the constraints of the PIO's instruction set, which is limited and designed for deterministic I/O rather than general-purpose computation.

That said, you *can* implement multiplication using shift-add, just like how multiplication is done in hardware (or early CPUs).

Below is a complete example of **32-bit multiplication using RP2040 PIO assembly**, multiplying two 32-bit unsigned integers, using the classic shift-and-add algorithm. This will compute a 64-bit result (stored in `OSR` or `X/Y` depending on how you adapt it).



You

refine the previous code so it is implemented via micropython but using PIO for the multiplication



ChatGPT

To implement multiplication using PIO in MicroPython on the Raspberry Pi Pico, you'll need to utilize the RP2040's PIO (Programmable Input/Output) feature. However, MicroPython doesn't have direct support for PIO as of my last update. You would typically interact with PIO by writing C/C++ code and interfacing it with MicroPython.

1. PIO Assembly Code (`multiplication.pio`):

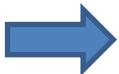
```
assembly
program multiplication_pio:
    wrap_target
        set x, 0
        set y, 0
        in pins, 32
        mov isr, y
        in pins, 32
        imul y, isr, y
        mov isr, x
        in pins, 32
        imul x, isr, x
        jmp wrap_target
```

 Copy code

PIO Instruction Set

Mnemonic	Short Description
JMP	Set program counter to specified address if specified condition is true. Otherwise, no operation.
WAIT	Stall until some condition is met.
IN	Shift specified number of bits from specified source into the Input Shift Register.
OUT	Shift specified amount of bits out of the Output Shift Register and write those bits to specified destination.
PUSH	Push the contents of the Input Shift Register into the RX FIFO as a single 32-bit word. Clear Input Shift Register to all-zeroes.
PULL	Load a 32-bit word from the TX FIFO into the Output Shift Register.
MOV	Copy data from the specified source to the specified destination.
IRQ	Set or clear the IRQ flag selected by the specified index argument.
SET	Write specified immediate value to specified destination.

The PIOs' instruction set consists of just nine different instructions that are highly specialized on bit manipulation.



Write a routine in assembly for raspberry pi pico using pure RP2040 PIO assembly for a 1-bit half-adder gate in a PIO core.

Creating a **1-bit half-adder** using pure **RP2040 PIO assembly** is a great example of using the PIO for custom digital logic emulation. A **half-adder** takes two input bits **A** and **B**, and produces:

- **SUM = A XOR B**
- **CARRY = A AND B**

The RP2040's PIO doesn't directly support logical operators like XOR or AND, but you can **emulate them** using its instruction set (**IN**, **OUT**, **JMP**, **MOV**, **SET**, **PUSH**, **PULL**, etc.), bit manipulation, and conditional jumps.

PIO Half-Adder Assembly

```

pio                                ⌂ Copy ⌂ Edit

.program half_adder
.side_set 2          ; We'll use 2 side-set bits to output SUM and CARRY

.wrap_target
    in pins, 2        ; Read 2 bits from input pins -> bits 0 = A, 1 = B

    ; At this point, ISR[0] = A, ISR[1] = B

    ; Copy inputs
    mov x, isr         ; Save inputs in X

    ; --- Compute SUM = A XOR B ---
    mov y, x           ; Copy A and B to Y
    ↓

; --- Compute SUM = A XOR B ---
    mov y, x           ; Copy A and B to Y
    push block         ; Save X for later (CARRY step)

    ; Isolate A (bit 0)
    mov isr, y
    in isr, 1
    mov y, isr         ; Y now has A

    ; Isolate B (bit 1)
    pull block
    mov isr, x
    in isr, 1
    mov x, isr         ; X now has B

    ; Compute A XOR B = (A | B) & ~(A & B)
    ; Simulated via branching logic, use jump ↓ handle cases
    ↓

```

ChatGPT, 2025

```

jmp y, a_is_1
jmp sum_0          ; A = 0, B = 0 → SUM = 0

a_is_1:
    jmp x, sum_0      ; A = 1, B = 1 → SUM = 0
    jmp sum_1          ; A = 1, B = 0 → SUM = 1

sum_1:
    set pins, 0b01 side ; SUM = 1 (on pin 2), CARRY = 0 (on pin 3)
    jmp end

sum_0:
    set pins, 0b00 side ; SUM = 0, CARRY to be decided next

    ; --- Compute CARRY = A & B ---
end:
    ; A & B is 1 only if both bits are 1
    mov isr, x           ; Restore B
    mov x, isr
    ↓

mov isr, x           ; Restore B
mov x, isr
pull block          ; Restore A
mov y, isr

jmp y, carry_check
set pins, 0b00 side ; A = 0 → CARRY = 0
jmp done

carry_check:
    jmp x, carry_1      ; A = 1 and B = 1 → CARRY = 1
    set pins, 0b00 side ; B = 0 → CARRY = 0
    jmp done

carry_1:
    set pins, 0b10 side ; CARRY = 1 (pin 3), SUM already set
done:
    nop
.wrap
    ↓

```

⚠ Limitations

- No direct arithmetic: PIO has no ADD, MUL, or even INC.
- Only 32 instructions per program (without chaining).
- Carry handling requires clever tricks or CPU support.

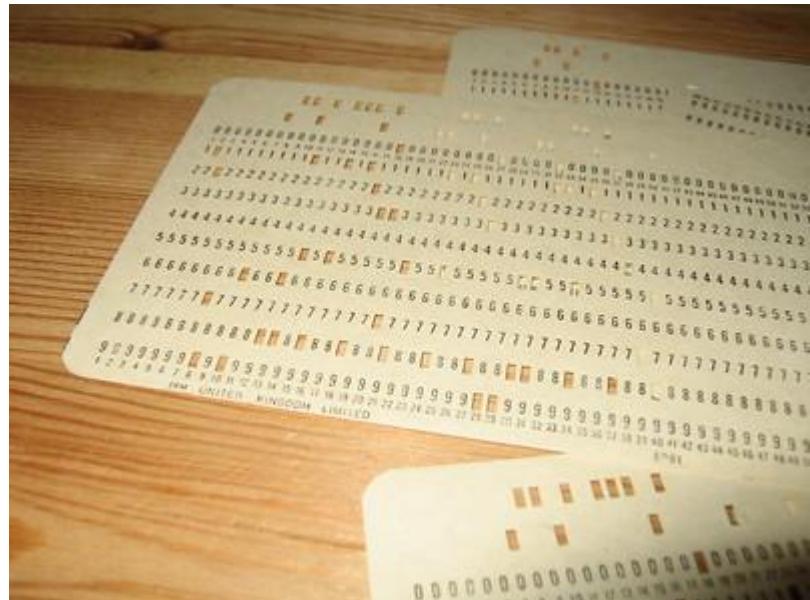
✓ Conclusion

While you *can* create a multiplication loop in PIO, it's highly recommended to offload the actual addition and 64-bit management to the main CPU unless you're doing very low-bit-width operations.

Would you like a complete example that does partial multiplication (like 8×8 or 16×16) fully in PIO, or a hybrid solution where PIO does the bit-loop and the CPU accumulates the result?



Reality, 2025: Micropython + PIO assembly (hand-made...)



Reality, 2025: Micropython + PIO assembly (hand-made)

```
pio_gates.py ✘ x |  
1  # run in IDLE with:  
2  #     exec(open('filename').read())  
3  
4  import rp2  
5  
6  @rp2.asm_pio()  
7  def gate_and_2bit():          # AND(x,y) logic gate  
8      wrap_target()  
9      pull()                  # read bit 0  
10     mov(x, osr)  
11     pull()                  # read bit 1  
12     mov(y, osr)  
13     jmp(x_not_y, "and0")    # x<>y => and:0  
14     jmp("output")          # x==y => and:x  
15     label("and0")  
16     set(x, 0)  
17     label("output")  
18     mov(isr, x)  
19     push()  
20     wrap()
```

FIFO-based
AND gate

```
22
23     @rp2.asm_pio()
24     def gate_or_2bit():          # OR(x,y) logic gate
25         wrap_target()
26         pull()                  # read bit 0
27         mov(x, osr)
28         pull()                  # read bit 1
29         mov(y, osr)
30         jmp(x_not_y, "or1")    # x<>y => or:1
31         jmp("output")          # x==y => or:x
32         label("or1")
33         set(x, 1)
34         label("output")
35         mov(isr, x)
36         push()
37         wrap()
38
39
40     @rp2.asm_pio()
41     def gate_not_1bit():        # NOT(x) logic gate
42         wrap_target()
43         pull()                  # read bit 0
44         mov(x, osr)
45         jmp(not_x, "x_zero")   # x==0 => not:1
46         set(x, 0)                # x==1 => not:0
47         jmp("output")
48         label("x_zero")
49         set(x, 1)
50         label("output")
51         mov(isr, x)
52         push()
53         wrap()
```

FIFO-based
OR gate

FIFO-based
NOT gate

```
57     def gate_xor_2bit():          # XOR(x,y) logic gate
58         wrap_target()
59         pull()                   # read bit 0
60         mov(x, osr)
61         pull()                   # read bit 1
62         mov(y, osr)
63         jmp(x_not_y, "xor1")    # x<>y => xor:1
64         set(x, 0)                # x==y => xor:0
65         jmp("output")
66         label("xor1")
67         set(x, 1)
68         label("output")
69         mov(isr, x)
70         push()
71         wrap()
72
73
74     @rp2.asm_pio()
75     def gate_nand_2bit():        # NAND(x,y) logic gate
76         wrap_target()
77         pull()                   # read bit 0
78         mov(x, osr)
79         pull()                   # read bit 1
80         mov(y, osr)
81         jmp(x_not_y, "nand1")   # x<>y => nand:1
82         jmp(not_x, "nand1")    # x==y==0 => nand:1
83         set(x, 0)                # x==y==1 => nand:0
84         jmp("output")
85         label("nand1")
86         set(x, 1)
87         label("output")
88         mov(isr, x)
89         push()
90         wrap()
```

FIFO-based
XOR gate

FIFO-based
NAND gate

Reality, 2025: Micropython + PIO assembly (hand-made)

```
@rp2.asm_pio()
def gate_and_or_2bit():
    wrap_target()           # AND(x,y) / OR(x,y) logic gates
    pull()                 # mode select: 0=OR, 1=AND
    mov(x, osr)            # read bit 0
    pull()                 # read bit 1
    mov(y, osr)
    jmp(x_not_y, "xdiffy") # x<>y => and/or:(read mode)
    pull()                 # mode not needed, just clear input
    jmp("output")          # x==y => and/or:x
    label("xdiffy")
    pull()                 # x<>y, mode => and/or:mode=(0,1)
    mov(x, osr)
    label("output")
    mov(isr, x)
    push()
    wrap()
```

FIFO-based
AND,OR gate

Reality, 2025: Micropython + PIO assembly (hand-made)

```
@rp2.asm_pio()
def gate_and_or_not_2bit():      # AND(x,y) / OR(x,y) / NOT(res) logic gates
    wrap_target()               # mode1 select: 0=OR, 1=AND / mode2 select: 1=NOT(res)
    pull()                      # read bit 0
    mov(x, osr)
    pull()                      # read bit 1
    mov(y, osr)
    jmp(x_not_y, "xdiffy")     # x<>y => and/or:(read mode)
    pull()                      # mode not needed, just clear input
    jmp("output1")              # x==y => and/or:x
    label("xdiffy")
    pull()                      # x<>y,mode => and/or:mode=(0,1)
    mov(x, osr)
    label("output1")            # check NOT flag
    pull()
    mov(y, osr)
    jmp(not_y, "output")        # NOT flag=0 => normal output
    jmp(not_x, "x_zero")         # NOT flag=1 => invert output
    set(x, 0)
    jmp("output")
    label("x_zero")
    set(x, 1)
    label("output")
    mov(isr, x)
    push()
    wrap()
```

FIFO-based
AND,OR,NOT gate

Reality, 2025: Micropython + PIO assembly (hand-made)

```
@rp2.asm_pio()
def gate_halfadd_2bit():      # HALFADD(x,y) logic gate
    wrap_target()
    pull()                  # read bit 0
    mov(x, osr)
    pull()                  # read bit 1
    mov(y, osr)
    jmp(x_not_y, "xdiffy")  # x<>y => ...
    jmp(not_x, "output")   # x==y==0 => S=0,C=0
    set(x, 0)               # x==y==1 => S=0,C=1
    jmp("output")
    label("xdiffy")
    set(x, 1)
    set(y, 0)
    label("output")
    mov(isr, x)             # output bit 0: S
    push()
    mov(isr, y)             # output bit 1: C
    push()
    wrap()
```

FIFO-based
1-bit Half-Adder gate

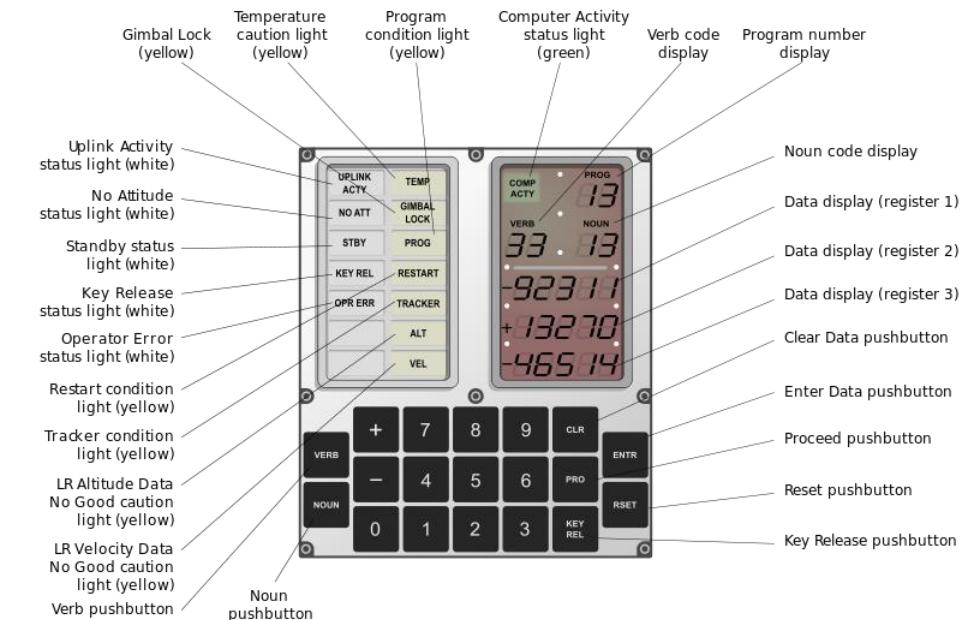
Reality, 2025: Micropython + PIO assembly (hand-made)

```
189     #sm = rp2.StateMachine(0, gate_nor_2bit, freq=-1)
190     #sm = rp2.StateMachine(0, gate_halfadd_2bit, freq=-1)
191     #sm = rp2.StateMachine(0, gate_and_or_2bit, freq=-1)
192     sm = rp2.StateMachine(0, gate_and_or_not_2bit, freq=-1)
193
194     sm.active(1)
195
196     for a in range(2):
197         for b in range(2):
198             sm.put(a)
199             sm.put(b)
200
201             #res=sm.get()
202             #print("%d.AND.%d -> %d" % (a,b,res) )
203             #print("%d.OR.%d -> %d" % (a,b,res) )
204             #print("NOT.%d -> %d" % (a,res) )
205             #print("%d.XOR.%d -> %d" % (a,b,res) )
206             #print("%d.NAND.%d -> %d" % (a,b,res) )
207             #print("%d.NOR.%d -> %d" % (a,b,res) )
208
209             resC=sm.get()
210             print("%d.HALFADD.%d -> (C,S)=(%d,%d)" % (a,b,resC,res))
211
212     sm.active(0)
213
```

Hamilton in 1969, standing next to listings of the software she and her MIT team produced for the Apollo project

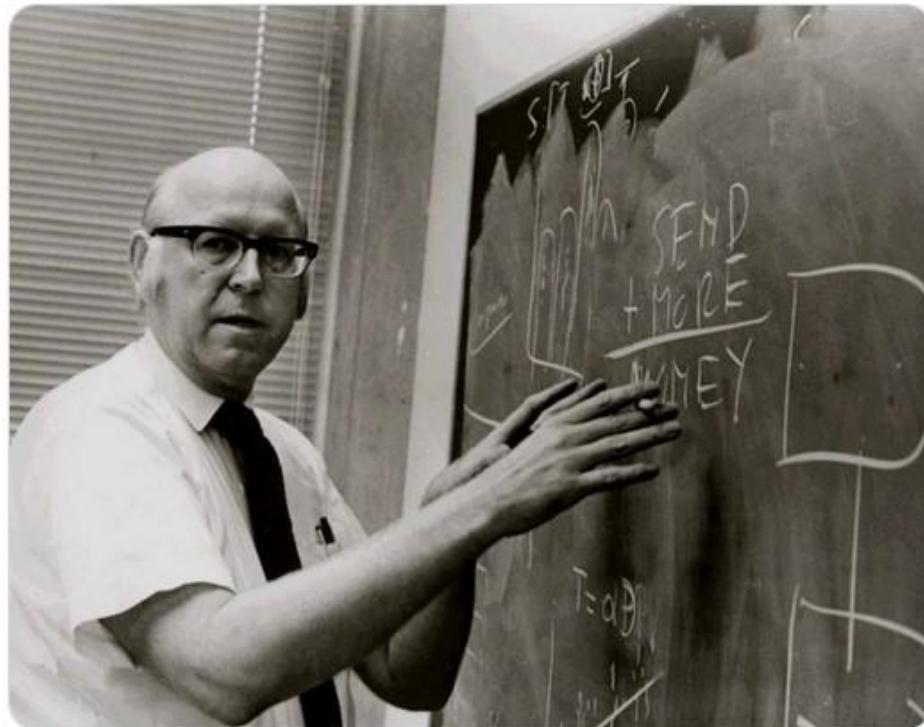


206	NEXTCORE	CAF	COREINC
207		ADS	LOCCTR
208		CCS	EXECTEM2
209		TCF	NOVAC3
210		LXCH	EXECTEM1
211		CA	Q
212		TC	BAILOUT1
213		OCT	1202
			# NO CORE SETS AVAILABLE.



MIT CSAIL  @MIT_CSAIL · 13h

Born #otd in 1927: Allen Newell, who in 1955 — before the term “AI” even existed — co-wrote what’s widely considered the world’s first AI program, Logic Theorist. This work proved 38 out of 52 classic math theorems.



MEMORANDUM
RM-3739-RC
JUNE 1963

IPL-V PROGRAMMERS' REFERENCE MANUAL

Edited by Allen Newell

The RAND Corporation
1700 Main St. • Santa Monica • California

Σύνοψη

- Περιεχόμενα:
 - Μέρος I: Αριθμητική και Λογική στις μηχανές
 - Μέρος II: Ψηφιακή τεχνολογία επεξεργαστών
- Σχετικό υλικό:
 - «Από τον άβακα στους σύγχρονους επεξεργαστές: Χτίζοντας τους Η/Υ (Μέρος I)», Ξ. Γεωργίου, Δ' κύκλος ανοικτών μαθημάτων ΕΠΕ – <https://youtu.be/p3TgG0iQGjQ>
 - «Από τον άβακα στους σύγχρονους επεξεργαστές: Χτίζοντας τους Η/Υ (Μέρος II)», Ξ. Γεωργίου, Δ' κύκλος ανοικτών μαθημάτων ΕΠΕ – <https://youtu.be/hfr1fuKVe8o>
 - «Hello x86: Low-level assembly coding for the 8086», @ApneaCoding – <https://youtu.be/tF16xTbd42w>
 - «BAM neural network in Arduino», @ApneaCoding – <https://youtu.be/RkM-rpSVD4I>

```

MOVE 1 TO DATA-C(N-T).
ADD 1 TO N-CHANGED.
GO TO LOOP-SCAN.

SELECT-CLZ.
ADD DATA-X(N-T) TO SUM2-X.
ADD DATA-Y(N-T) TO SUM2-Y.
ADD 1 TO N-CLZ.
IF DATA-C(N-T) EQUAL 2 GO TO LOOP-SCAN.
MOVE 2 TO DATA-C(N-T).
ADD 1 TO N-CHANGED.

```

```

LOOP-
  91   id : Integer := 0; -- target ID (counter)
  92   det : Integer := 0; -- detection slots in sequence
  93   pwr : Integer := 0; -- rel. power of detection
  94   pwr0 : Integer := delimit; -- rel. power baseline (adapt
  95   disp : Boolean := False; -- target reporting (#flag)
  96
  begin
    -- process the FOV slots --
    for p in 1..(seekerData'Length)-1 loop
      -- rel. power is current detection 'step'
      pwr := abs(seekerData(p+1)-seekerData(p));
      if pwr >= delimit then
        -- detection valid, continue analysis
        if pwr > pwr0 then
          -- strong new 'step' from baseline (new target)
          pwr0 := pwr; -- update the baseline
          det := 0; -- reset the run-length
          disp := False; -- enable target reporting
        end if;
      end if;
    end loop;
  end;
  109  det;
  110  -- if
  111  i;
  112  d;
  113  p;
  114  d;
  115  p;
  116  d;
  117

```

```

sketch.ino    diagram.json    Library Manager
  220  lcd.setCursor(0,1);
  221  lcd.print(" 1.0.0");
  222  delay(PRINT_DELAY);

  223  lcd.clear();
  224  lcd.setCursor(0,0);
  225  lcd.print("Training...");

  228  train_weights();
  229  delay(PRINT_DELAY); // just for display pause
  230

  231  lcd.setCursor(0,1);
  232  lcd.print("> Finished!");
  233  delay(1000);
  234 }

  236 // continuous loop routine
  237 void loop( void )
  {
    display_weights();
    display_input_tests();
    display_output_tests();
  }

```

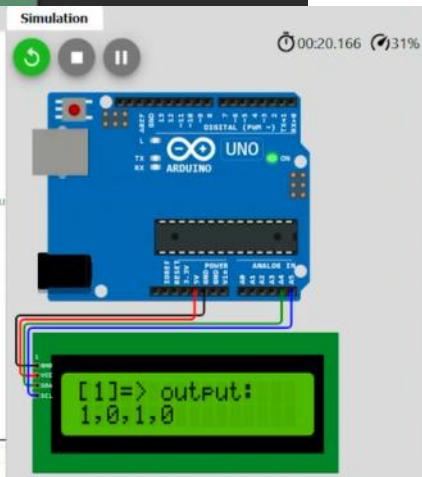
STDIN
Input for the program (Optional)

Output:

```

x86_64-linux-gnu-gcc-9 -c HelloWorld.adb
HelloWorld.adb:1:01: compilation unit expected
x86_64-linux-gnu-gnatmake-9: "HelloWorld.adb" com

```



- Hamming (7,4) error correction codes in **R**
- Kmeans clustering in **COBOL**
- Bi-directional Associative Memory (BAM) in **Arduino/C**
- Linear Regression in **SQL, Matlab**
- ...

YouTube:

@ApneaCoding



<https://www.youtube.com/@apneacoding>

<https://www.facebook.com/apneacoding>

Github:

@xgeorgio



<https://github.com/xgeorgio>

<http://apneacoding.blogspot.com>

Ένας ψηφιακός κόσμος γεμάτος γνώση για όλους

Σύμφωνα με το Καταστατικό της Ένωσης Πληροφορικών Ελλάδας, ένας από τους βασικούς σκοπούς της λειτουργίας της είναι η προώθηση της γνώσης και χρήσης των πληροφορικών αγαθών από το κοινωνικό σύνολο και η εξάλειψη της τεχνοφοβίας και του "αναλφαβητισμού" στην Πληροφορική.



<https://courses.epe.org.gr>

Σχετικά με τα ανοικτά μαθήματα της Ένωσης Πληροφορικών Ελλάδας:

- ✓ Τα μαθήματα πραγματοποιούνται εξ ολοκλήρου διαδικτυακά, ζωντανά μέσω της πλατφόρμας Zoom.
- ✓ Η συμμετοχή σε όλα τα μαθήματα είναι ελεύθερη για οποιονδήποτε από οποιδήποτε στην Ελλάδα ή στο εξωτερικό.
- ✓ Δεν υπάρχει οικονομικό κόστος ή άλλες προϋποθέσεις συμμετοχής.
- ✓ Οι Εισιγητές είναι μέλη της Ένωσης Πληροφορικών Ελλάδας και πραγματοποιούν τα μαθήματα εθελοντικά.
- ✓ Τα μαθήματα μαγνητοσκοπούνται και παραμένουν διαθέσιμα για σύγχρονη παρακολούθηση στο Αρχείο Μαθημάτων.
- ✓ Η εκπαίδευση που παρέχεται μέσω των ανοικτών διαδικτυακών μαθημάτων είναι άτυπη και δεν παρέχονται βεβαιώσεις παρακολούθησης στους συμμετέχοντες.





Ανοικτά Μαθήματα // Ένωση Πληροφορικών Ελλάδας 1 year ago

⋮

Καθώς προετοιμάζεται ο Γ' Κύκλος ανοικτών διαδικτυακών μαθημάτων από την ΕΠΕ, και ταυτόχρονα το κανάλι στο Youtube εμπλουτίζεται συνεχώς με νέο περιεχόμενο, θα θέλαμε τη γνώμη σας.

Ποιά από τα παρακάτω θεωρείτε πιο σημαντικά; Ποια θα θέλατε να παρακολουθήσετε;

44 votes

- Διαδικτυακά μαθήματα - Εισαγωγικά θέματα 27%
- Διαδικτυακά μαθήματα - Προχωρημένα θέματα 55%
- Παρουσιάσεις - Διαλέξεις 7%
- Συνεντεύξεις με προσκεκλημένους 2%
- Ζωντανές συζητήσεις (live streaming) 9%

Ερώτηση 1

<https://hiu.gr/s/1f>



Ασχολούμαι με τον προγραμματισμό:

0 votes

Συστηματικά - Επαγγελματικά 0%

Συστηματικά - Χόμπι ή διασκέδαση 0%

Περιστασιακά - Λίγο 0%

Σπάνια - Καθόλου 0%

Ερώτηση 2

<https://hiu.gr/s/1g>



Τα εργαστηριακά μαθήματα (workshops) είναι:

0 votes

Πολύ χρήσιμα και ενδιαφέροντα

0%

Χρήσιμα, όχι πολύ ενδιαφέροντα

0%

Ενδιαφέροντα, όχι πολύ χρήσιμα

0%

Αδιάφορα, δεν τα προτείνω

0%

Ερώτηση 3

<https://hiu.gr/s/1h>



Στα εργαστηριακά μαθήματα (workshops) θα ήθελα:

0 votes

Προγραμματισμό low-level (assembly, micro-controllers/GPIO, κτλ) 0%

Προγραμματισμό mid/high-level (C, C++, Java, Python, κτλ) 0%

Ειδικές εφαρμογές (web stacks, DB, networking, κτλ) 0%

Αναλυτική Δεδομένων / Μηχανική Μάθηση (Weka, Python, Matlab, κτλ) 0%

Ερωτήσεις



Χάρης Γεωργίου (MSc,PhD)
<https://www.linkedin.com/in/xgeorgio>
<https://methodd.substack.com>