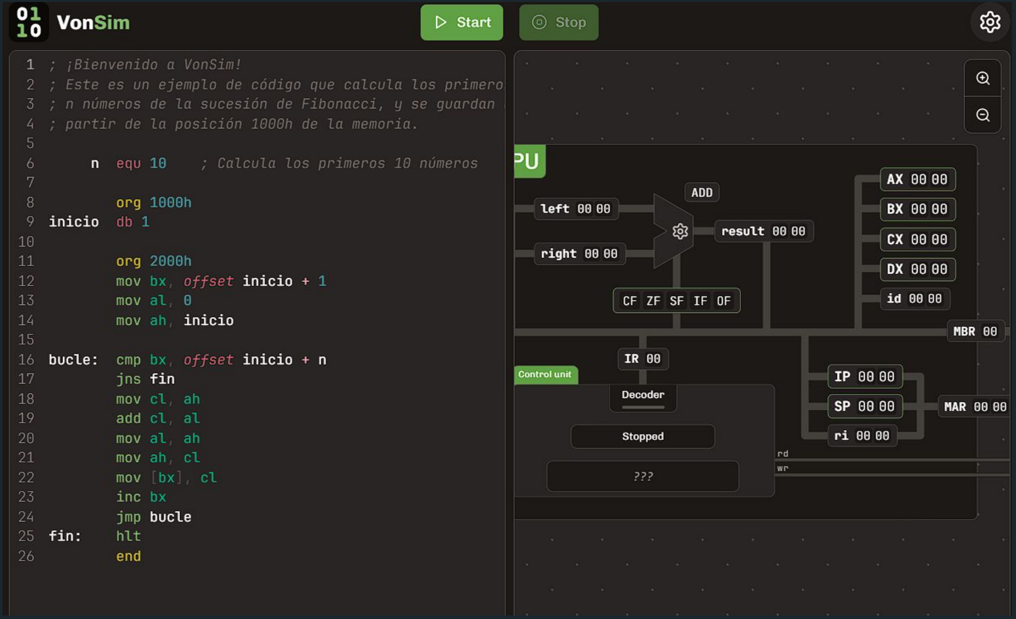
VonSim

Educational Simulator for Assembly

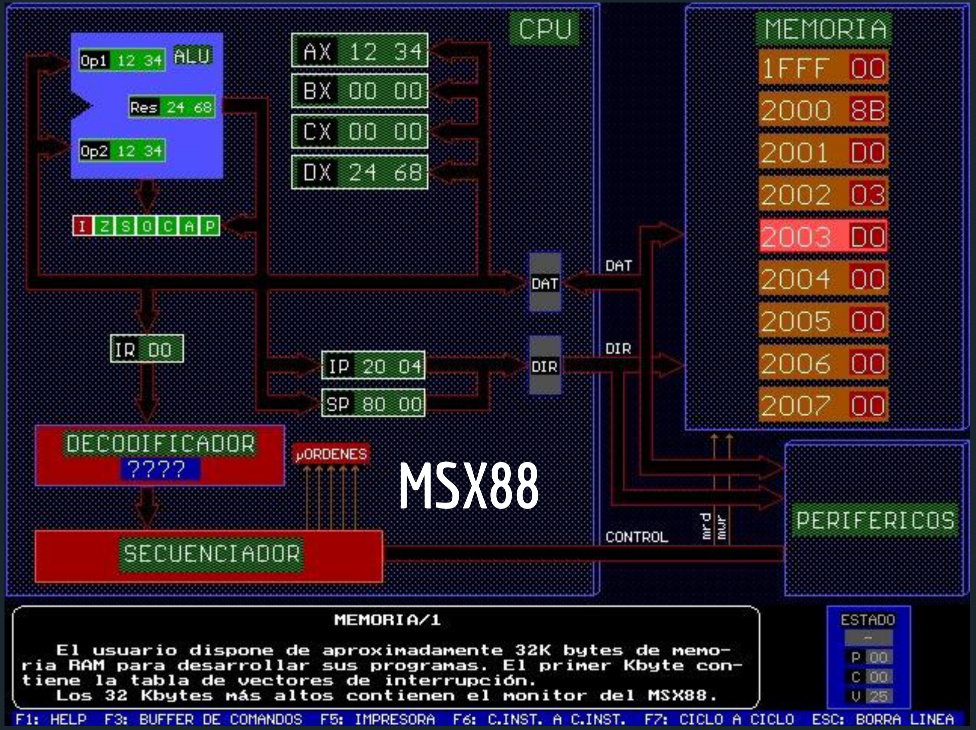


```
; ¡Bienvenido a VonSim!
; Este es un ejemplo de código que calcula los primero
; n números de la sucesión de Fibonacci, y se guardan
; partir de la posición 1000h de la memoria.

      n  equ 10    ; Calcula los primeros 10 números

         org 1000h
inicio   db 1

         org 2000h
         mov bx, offset inicio + 1
         mov al, 0
         mov ah, inicio

bucle:   cmp bx, offset inicio + n
         jns fin
         mov cl, ah
         add cl, al
         mov al, ah
         mov ah, cl
         mov [bx], cl
         inc bx
         jmp bucle
fin:     hlt
         end
```

🔗 https://vonsim.github.io/

Context

- Informatics School
  - Universidad Nacional de La Plata
- Courses that teach Assembly
  - Computer Organization
  - Computer Architecture
- ~1500 students per year
- Previously: **MSX88** simulator
  - VonSim 1: 2017
  - VonSim 2: 2023



AX 12 34
BX 00 00
CX 00 00
DX 24 68
IP 20 04
SP 80 00

1FFF	00
2000	8B
2001	D0
2002	03
2003	D0
2004	00
2005	00
2006	00
2007	00

MSX88

MEMORIA/1

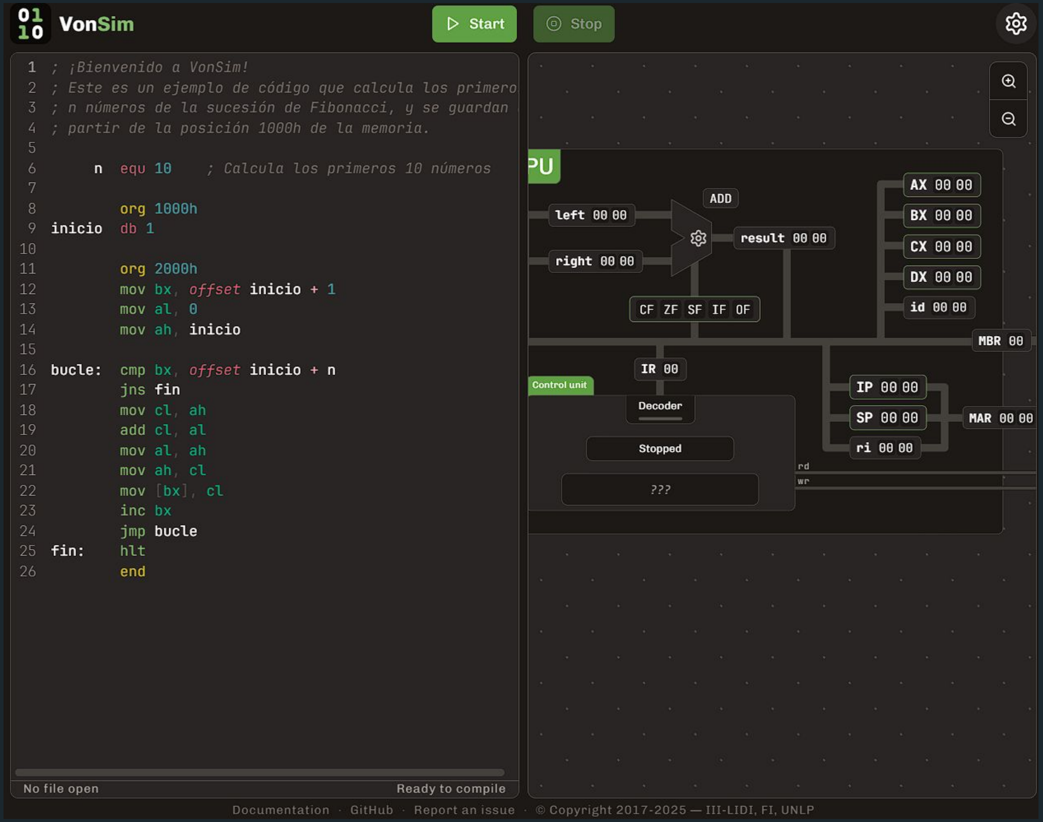
El usuario dispone de aproximadamente 32K bytes de memoria RAM para desarrollar sus programas. El primer Kbyte contiene la tabla de vectores de interrupción.
Los 32 Kbytes más altos contienen el monitor del MSX88.

F1: HELP F3: BUFFER DE COMANDOS F5: IMPRESORA F6: C.INST. A C.INST. F7: CICLO A CICLO ESC: BORRA LINEA

VonSim

https://vonsim.github.io/

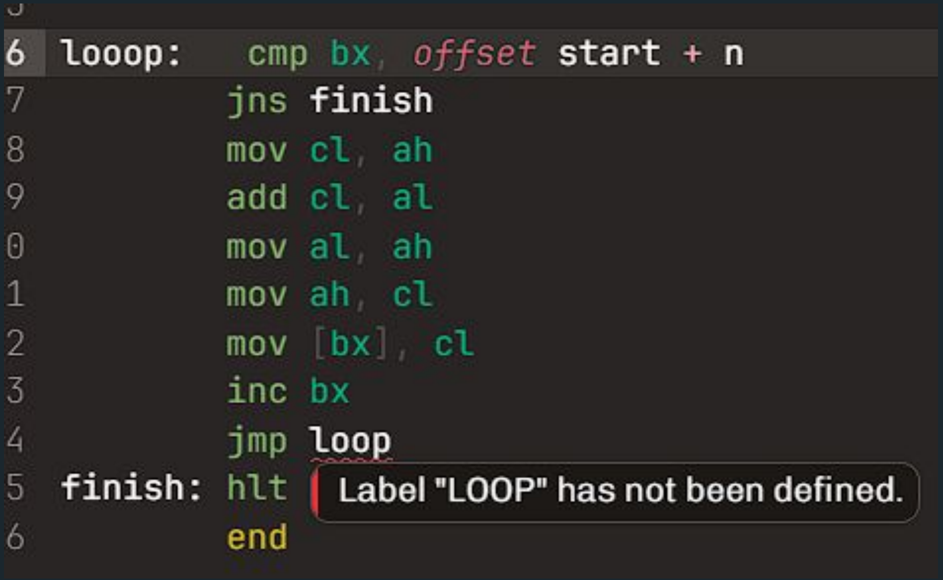
- Web App
- IDE
  - Compile & Ex
  - Debug
- Simplified Architecture
- Component Visualization
  - I/O Devices
- Integrated Examples
- Mobile/Tablet mode



```
; ¡Bienvenido a VonSim!
; Este es un ejemplo de código que calcula los primero
; n números de la sucesión de Fibonacci, y se guardan
; partir de la posición 1000h de la memoria.

      n  equ 10     ; Calcula los primeros 10 números

         org 1000h
inicio   db 1

         org 2000h
         mov bx, offset inicio + 1
         mov al, 0
         mov ah, inicio

bucle:   cmp bx, offset inicio + n
         jns fin
         mov cl, ah
         add cl, al
         mov al, ah
         mov ah, cl
         mov [bx], cl
         inc bx
         jmp bucle
fin:     hlt
         end
```

IDE

- Syntax Coloring
- Continuous compilation
- Detailed error messages with visual feedback

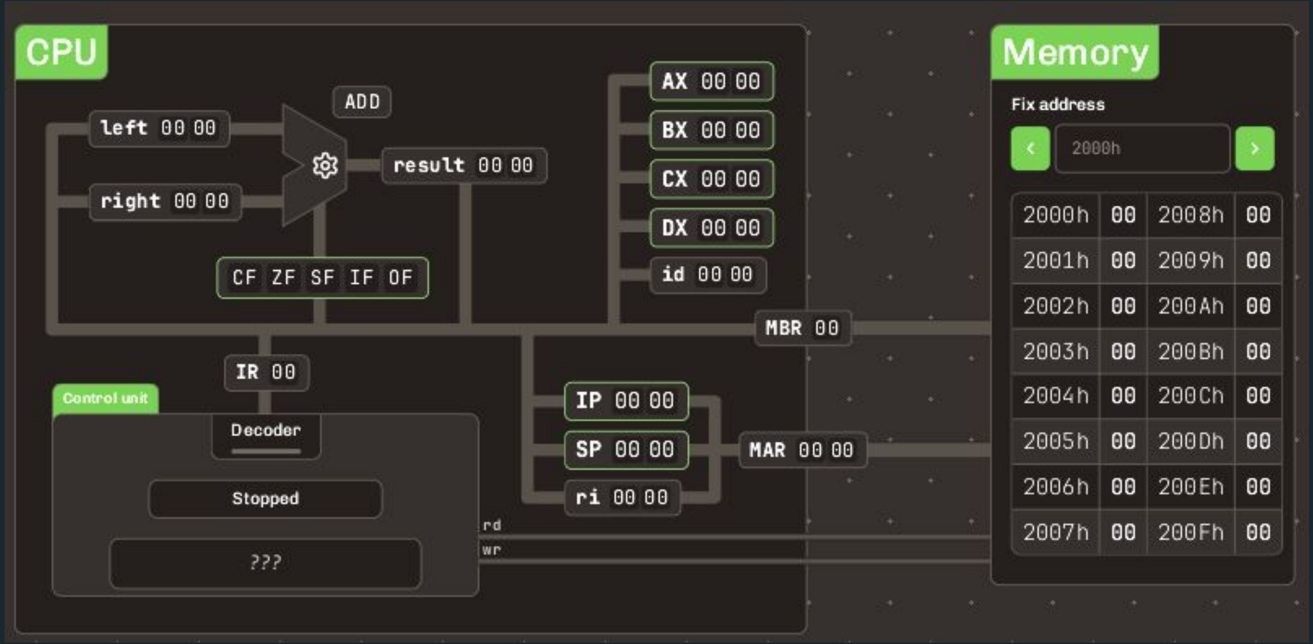


```
looop:     cmp bx, offset start + n
          jns finish
          mov cl, ah
          add cl, al
          mov al, ah
          mov ah, cl
          mov [bx], cl
          inc bx
          jmp loop
finish:   hlt
          end
```

Label "LOOP" has not been defined.

CPU/Memory

- 8 half registers - 8 bits each: AL, AH, BL, BH, Cl, CH, DL, DH
  - Can be addressed as 4 16-bit registers: AX, BX, CX, DX
  - Low/high parts
- Especial state registers
- ALU
- Memory: 16838 bytes
  - 16 bit addresses



left 00 00

right 00 00

result 00 00

CF ZF SF IF OF

AX 00 00

BX 00 00

CX 00 00

DX 00 00

id 00 00

MBR 00

IR 00

IP 00 00

SP 00 00

ri 00 00

MAR 00 00

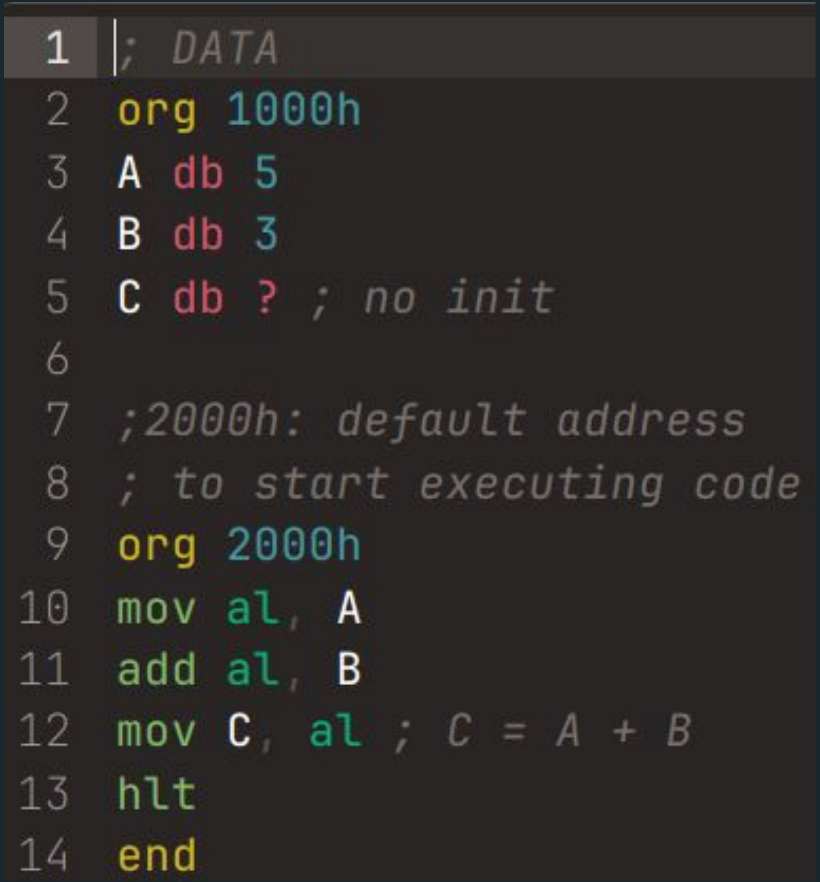
2000h

2000h	00	2008h	00
2001h	00	2009h	00
2002h	00	200Ah	00
2003h	00	200Bh	00
2004h	00	200Ch	00
2005h	00	200Dh	00
2006h	00	200Eh	00
2007h	00	200Fh	00

Simplified 8088 Assembly

- ORG ‹ADDRESS›
  - Explicit starting address for code/data
- END
- MOV dest, source
  - Data transfers
- OP dest[, source]
  - OP = ADD | SUB | OR | AND | XOR | NEG | NOR
- hlt
  - Stop execution

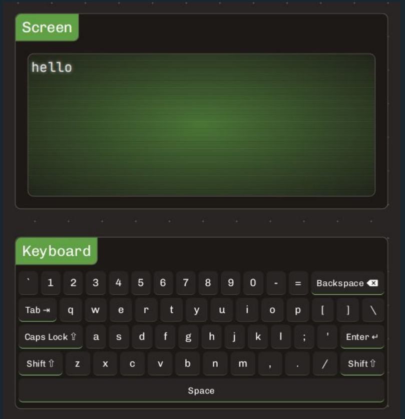
Compute C = A + B



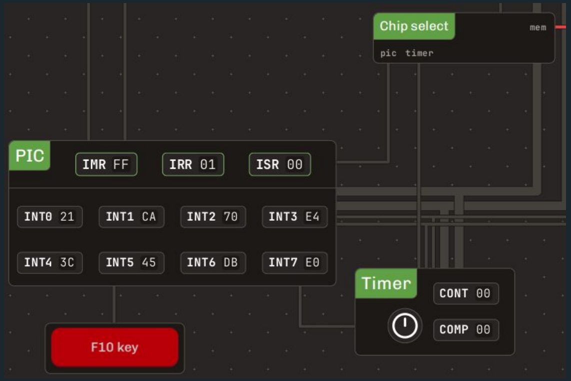
```
; DATA
org 1000h
A db 5
B db 3
C db ? ; no init

;2000h: default address
; to start executing code
org 2000h
mov al, A
add al, B
mov C, al ; C = A + B
hlt
end
```

Device I/O

- Keyboard & Screen
- PIC
  - → Timer
  - → F10 Button
  - → HANDSHAKE
- PIO
  - → LEDs & Switches
  - → Printer
- HANDSHAKE
  - → Printer


Screen
Keyboard
` 1 2 3 4 5 6 7 8 9 0 - = Backspace ⌫
Tab ⇥ q w e r t y u i o p [ ] \
Caps Lock ⇪ a s d f g h j k l ; ' Enter ↵
Shift ⇧ z x c v b n m , . / Shift ⇧
Space

```
hello
```



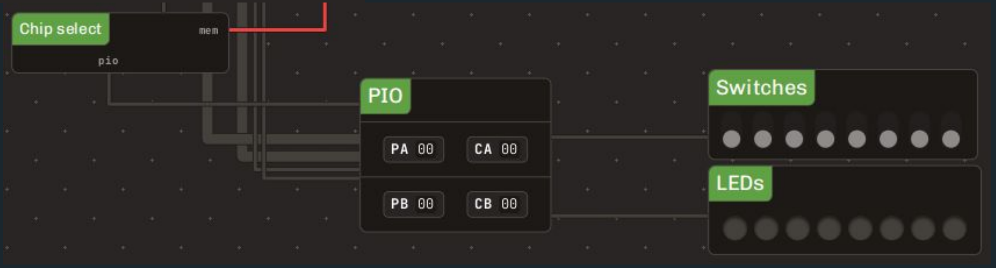
IMR FF IRR 01 ISR 00

INT0 21 INT1 CA INT2 70 INT3 E4

INT4 3C INT5 45 INT6 DB INT7 E0

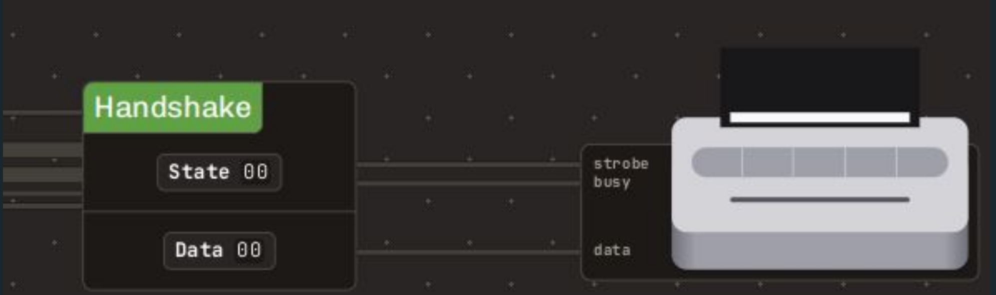
CONT 00

COMP 00



PA 00 CA 00

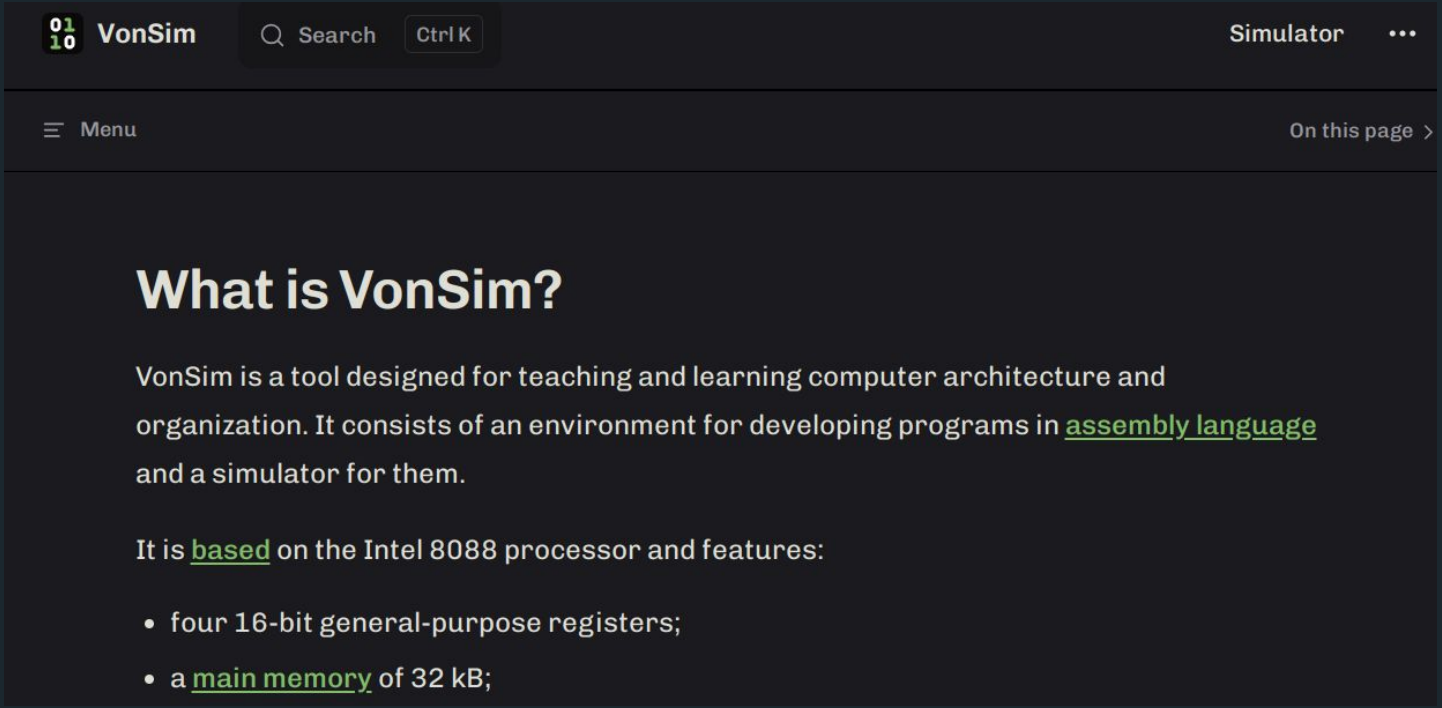
PB 00 CB 00



State 00

Data 00

VonSim 2 Docs



What is VonSim?

VonSim is a tool designed for teaching and learning computer architecture and organization. It consists of an environment for developing programs in assembly language and a simulator for them.

It is based on the Intel 8088 processor and features:

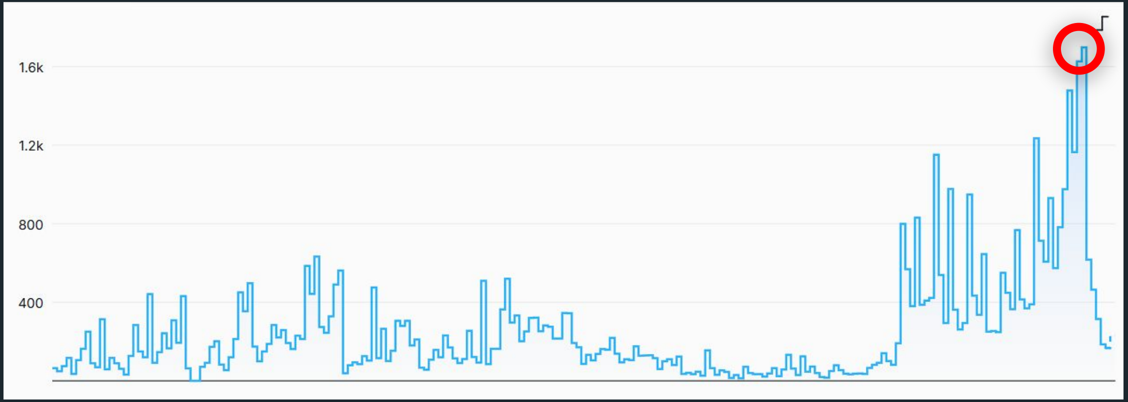
- four 16-bit general-purpose registers;
- a main memory of 32 kB;

https://vonsim.github.io/en/

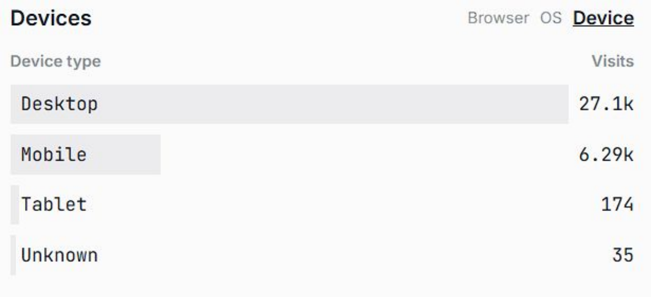
VonSim 2 Usage statistics

- Via OneDollarStats (2025/03 - 2025/10)

Visits per day



Devices



Devices

Device type	Visits
Desktop	27.1k
Mobile	6.29k
Tablet	174
Unknown	35

VonSim 2 Team

Juan Seery
https://github.com/JuanM04

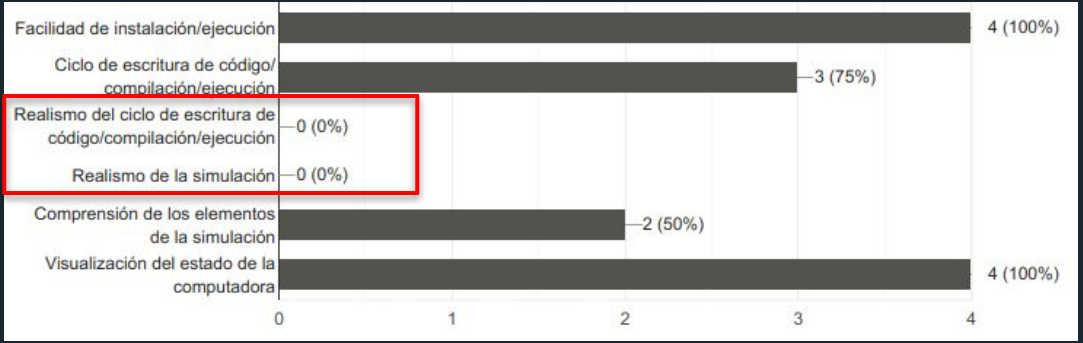
Facundo Quiroga
https://github.com/facundoq

https://github.com/vonsim/vonsim

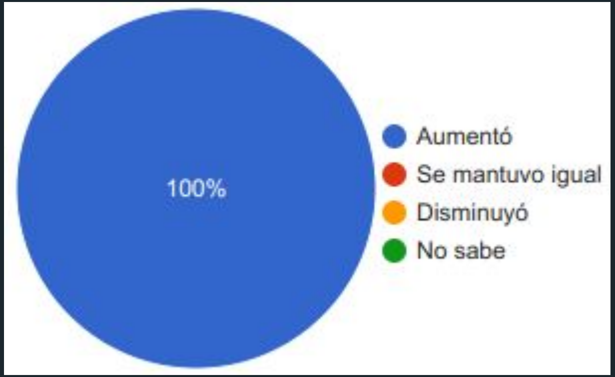
Evaluación docente

- Encuesta inicial (n=4)
- Uso mixto de ambos simuladores desde 2018

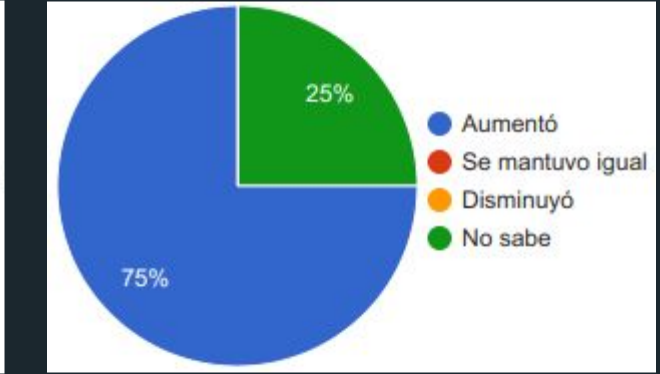
Vonsim (vs MSX88)



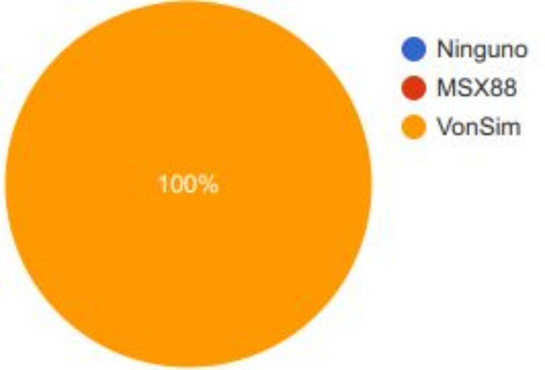
Motivación general



Tiempo de práctica



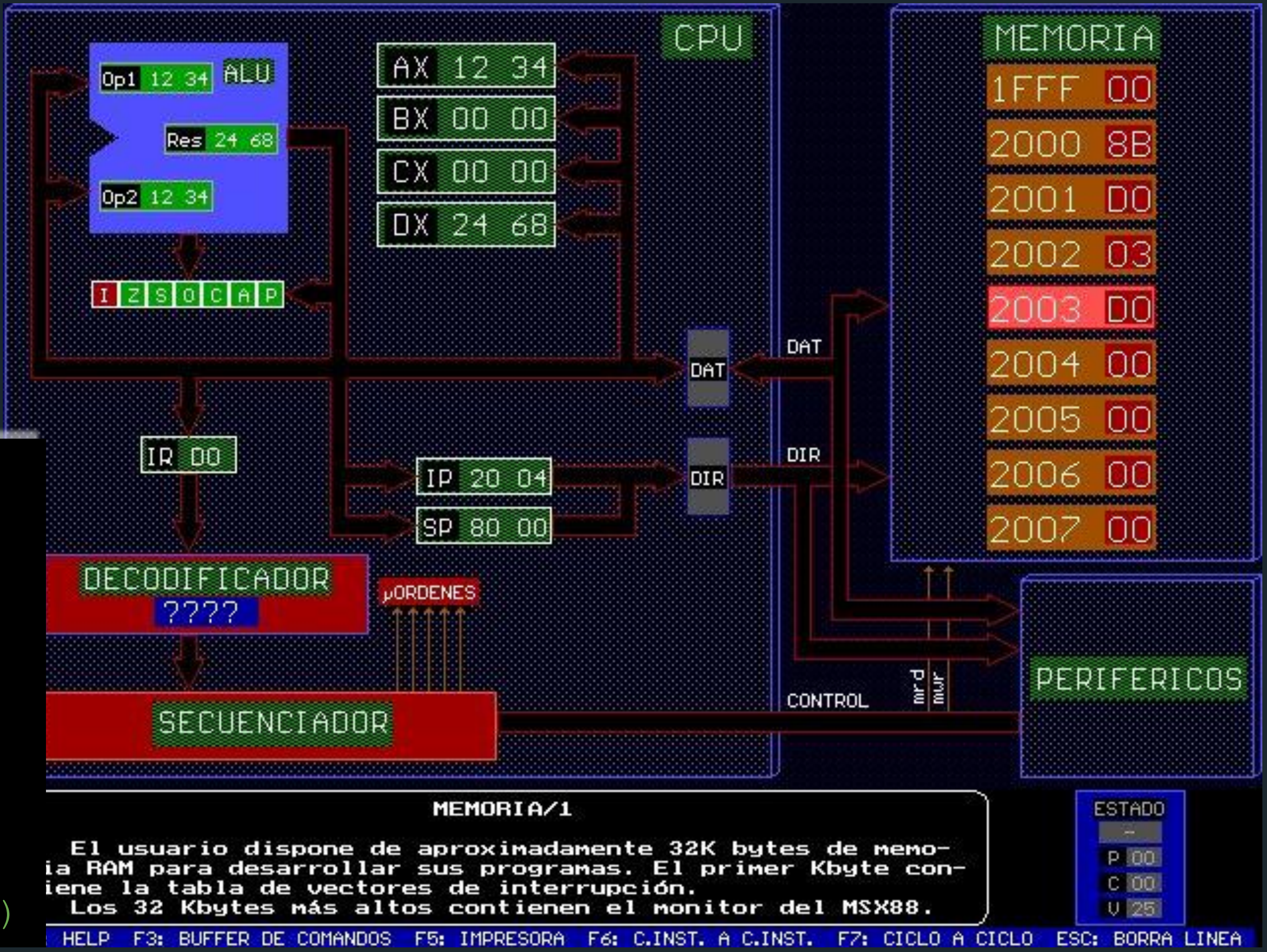
Simulador recomendado



MSX88

- 1990
- Console only
- Windows/Wine
- No editor

```
> asm88.exe foo.asm
foo.O generado
> link88 foo.O
foo.eje generado
> msx88
(abre simulador)
>> l foo
(carga programa)
>> g
(comienza ejecución)
```



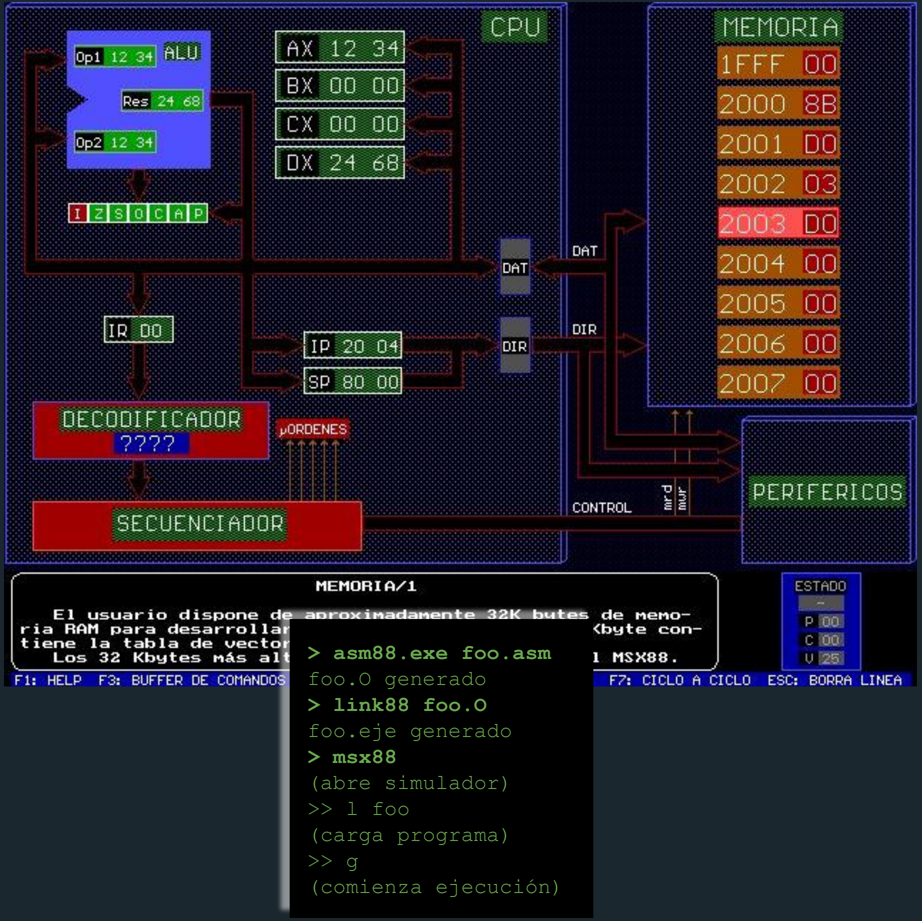
1FFF	00
2000	8B
2001	D0
2002	03
2003	D0
2004	00
2005	00
2006	00
2007	00

MEMORIA/1

El usuario dispone de aproximadamente 32K bytes de memo-
ia RAM para desarrollar sus programas. El primer Kbyte con-
iene la tabla de vectores de interrupción.
Los 32 Kbytes más altos contienen el monitor del MSX88.

MSX88

- Pedagogical Challenges
  - Confusing error messages
  - Long compile/test cycle
- Practical difficulties
  - Does not work in mobile/tablet
  - Requires use of console



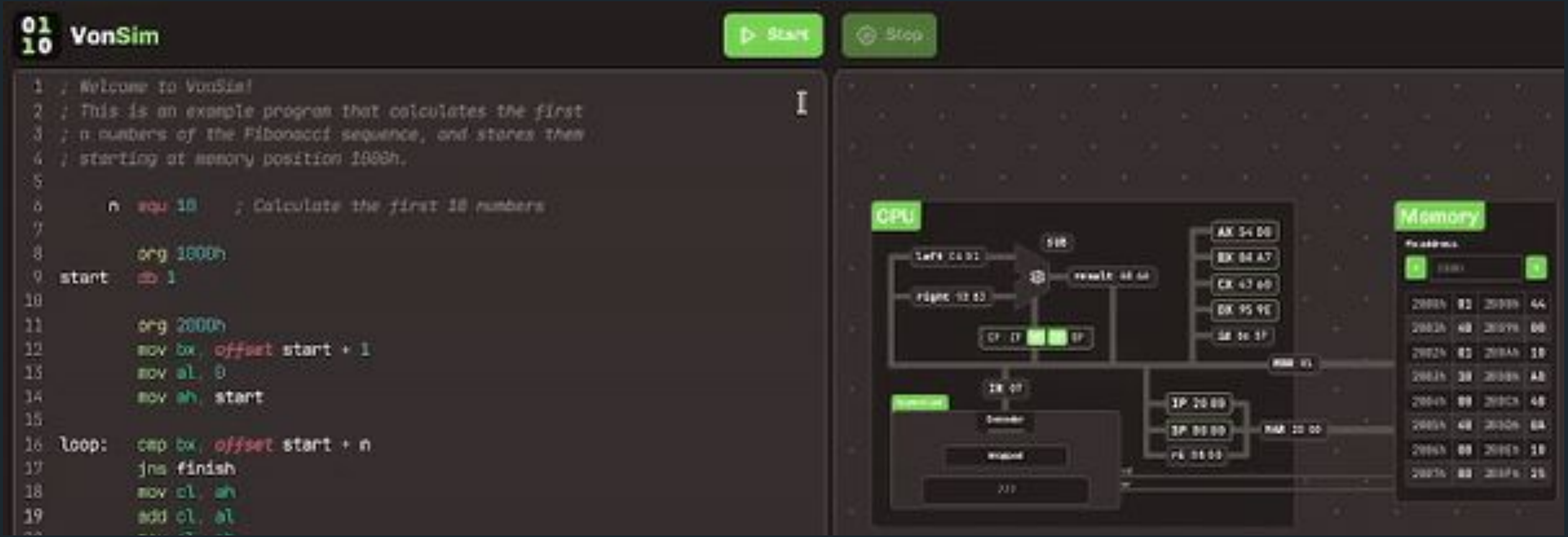
MEMORIA/1

El usuario dispone de aproximadamente 32K bytes de memo-
ria RAM para desarrollar Kbyte con-
tiene la tabla de vector
Los 32 Kbytes más alt l MSX88.

F1: HELP F3: BUFFER DE COMANDOS F7: CICLO A CICLO ESC: BORRA LINEA

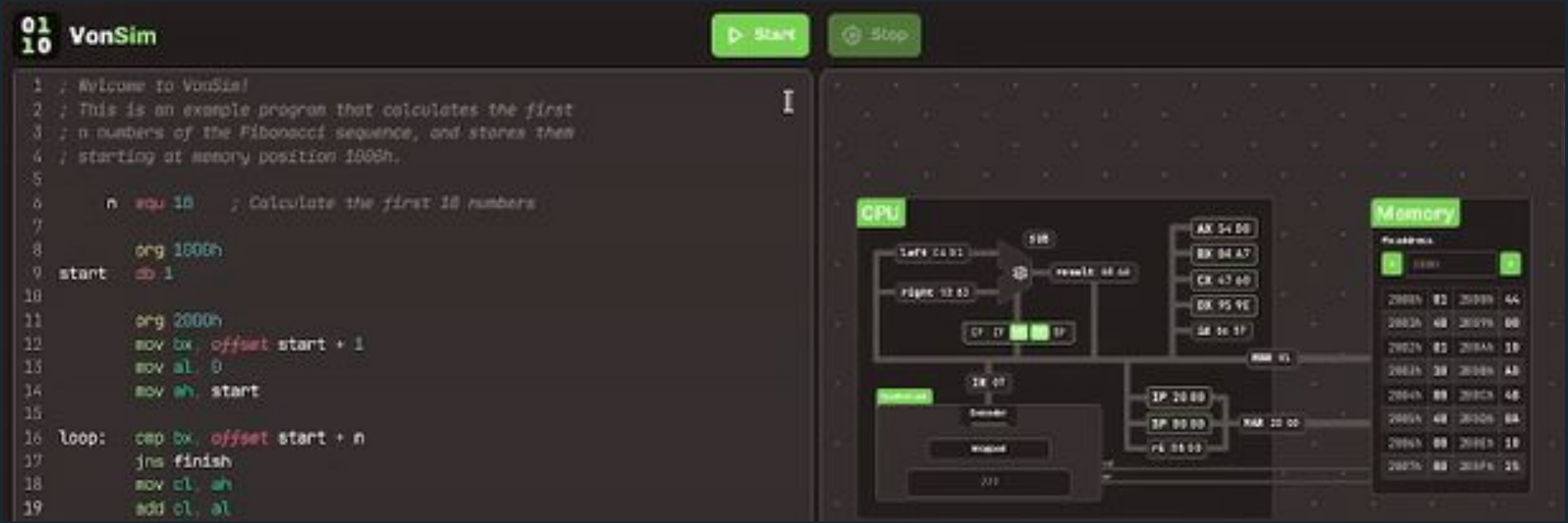
```
> asm88.exe foo.asm
foo.O generado
> link88 foo.O
foo.eje generado
> msx88
(abre simulador)
>> l foo
(carga programa)
>> g
(comienza ejecución)
```

Data transfer animations



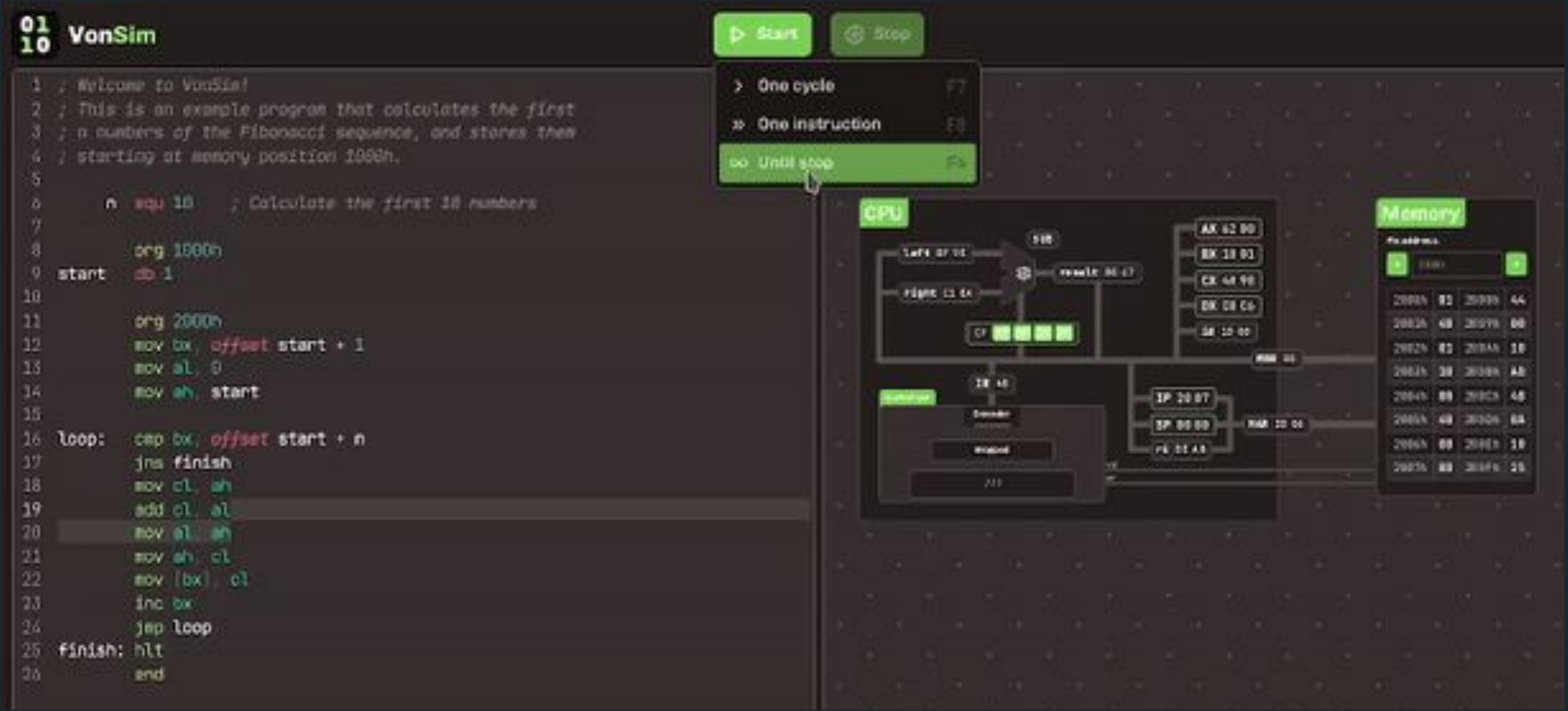
```
; Welcome to VonSim!
; This is an example program that calculates the first
; n numbers of the Fibonacci sequence, and stores them
; starting at memory position 1000h.

     n  equ 10     ; Calculate the first 10 numbers

        org 1000h
start   db 1

        org 2000h
        mov bx, offset start + 1
        mov al, 0
        mov ah, start

loop:   cmp bx, offset start + n
        jns finish
        mov cl, ah
        add cl, al
```

Instruction by instruction execution



```
; Welcome to VonSim!
; This is an example program that calculates the first
; n numbers of the Fibonacci sequence, and stores them
; starting at memory position 1000h.

      n  equ 10     ; Calculate the first 10 numbers

         org 1000h
start    db 1

         org 2000h
         mov bx, offset start + 1
         mov al, 0
         mov ah, start

loop:    cmp bx, offset start + n
         jns finish
         mov cl, ah
         add cl, al
```

Quick run without animations

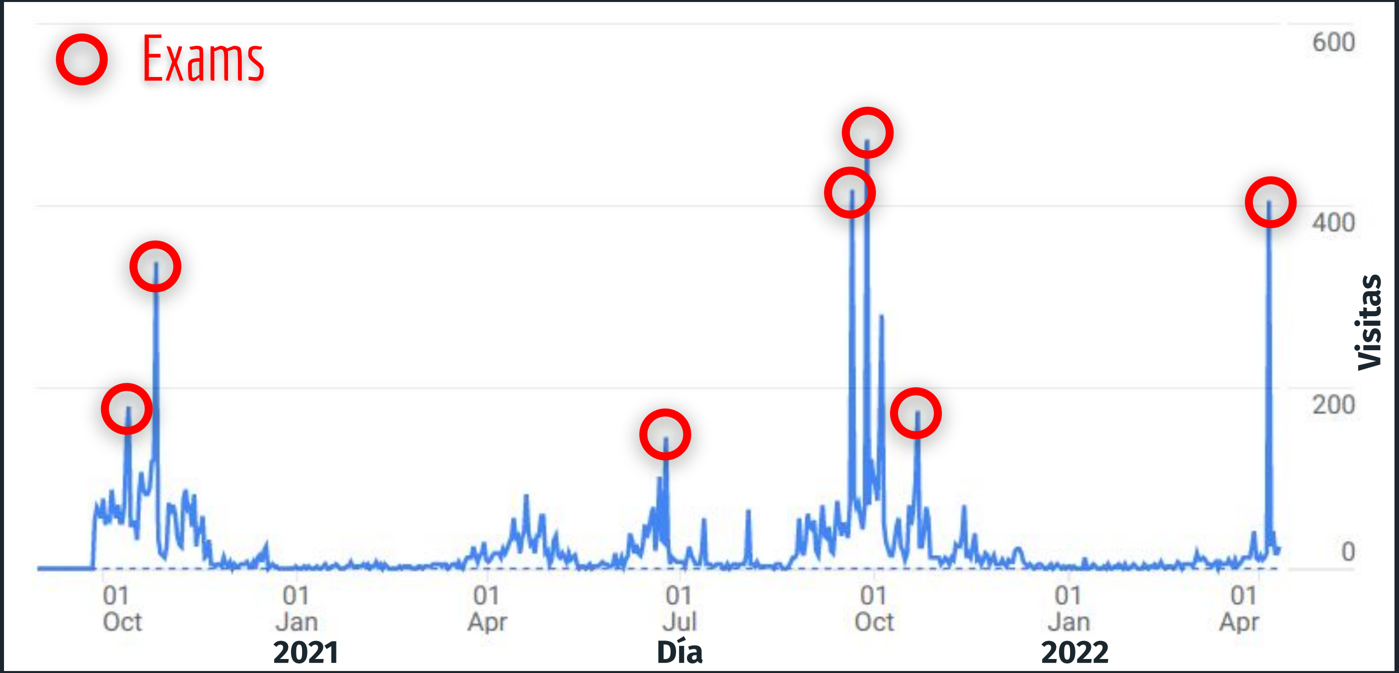


```
; Welcome to VonSim!
; This is an example program that calculates the first
; n numbers of the Fibonacci sequence, and stores them
; starting at memory position 1000h.

      n   equ 10    ; Calculate the first 10 numbers

          org 1000h
start     db 1

          org 2000h
          mov bx, offset start + 1
          mov al, 0
          mov ah, start

loop:     cmp bx, offset start + n
          jns finish
          mov cl, ah
          add cl, al
          mov al, ah
          mov ah, cl
          mov [bx], cl
          inc bx
          jmp loop
finish:   hlt
          end
```

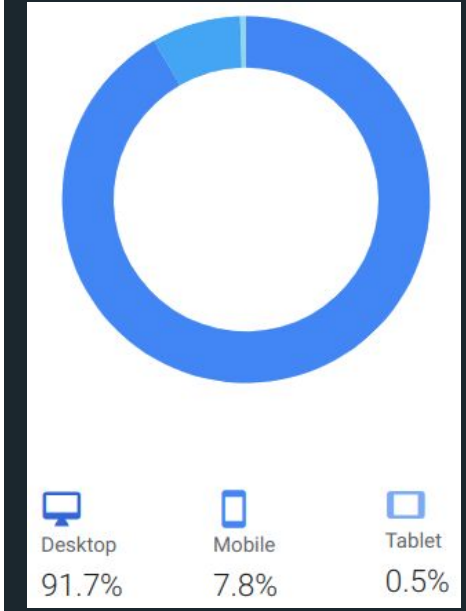
VonSim 1 Usage statistics

- Via Google Analytics (2020/10 - 2021/04)

Visits per day

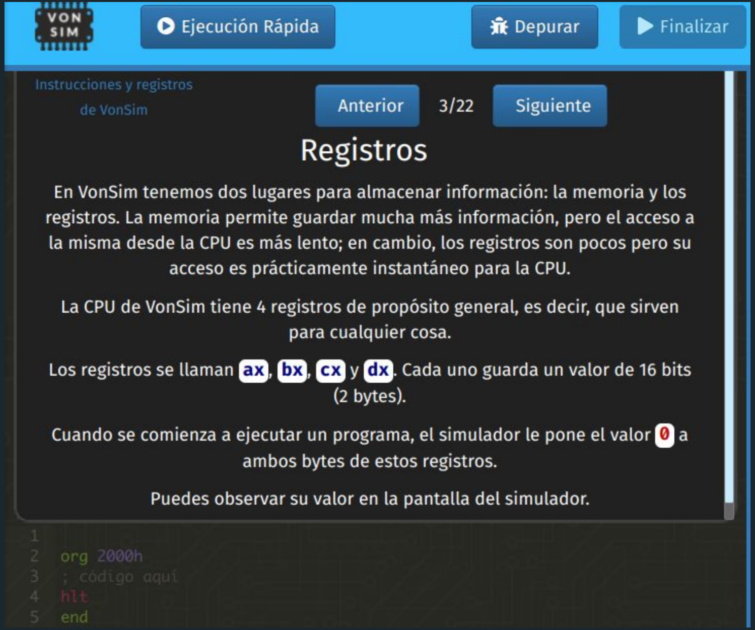


Devices



Tutoriales

- Integrados en la aplicación
- Ejercicios incluidos



Registros

En VonSim tenemos dos lugares para almacenar información: la memoria y los registros. La memoria permite guardar mucha más información, pero el acceso a la misma desde la CPU es más lento; en cambio, los registros son pocos pero su acceso es prácticamente instantáneo para la CPU.

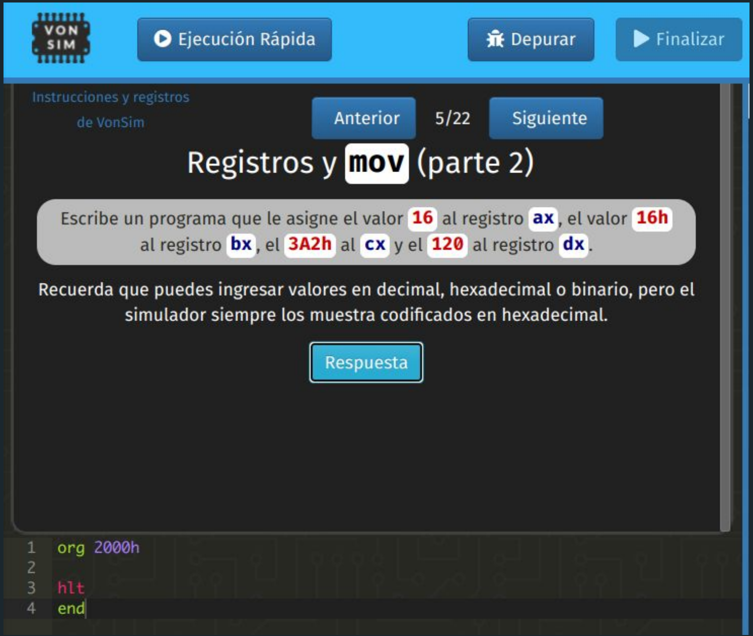
La CPU de VonSim tiene 4 registros de propósito general, es decir, que sirven para cualquier cosa.

Los registros se llaman `ax`, `bx`, `cx` y `dx`. Cada uno guarda un valor de 16 bits (2 bytes).

Cuando se comienza a ejecutar un programa, el simulador le pone el valor `0` a ambos bytes de estos registros.

Puedes observar su valor en la pantalla del simulador.

```

org 2000h
; código aquí
hlt
end
```



Registros y `mov` (parte 2)

Escribe un programa que le asigne el valor `16` al registro `ax`, el valor `16h` al registro `bx`, el `3A2h` al `cx` y el `120` al registro `dx`.

Recuerda que puedes ingresar valores en decimal, hexadecimal o binario, pero el simulador siempre los muestra codificados en hexadecimal.

```
org 2000h

hlt
end
```