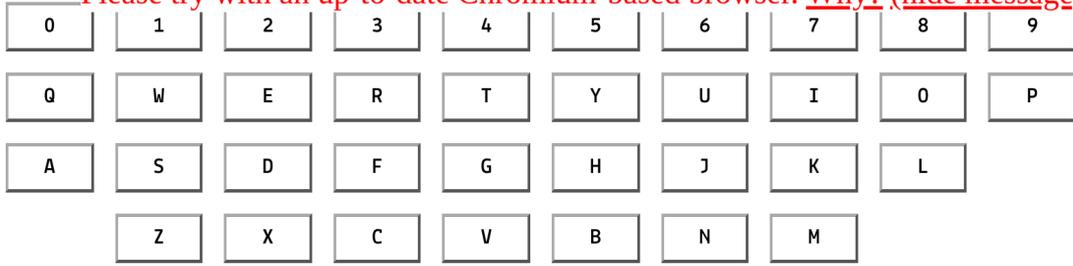


x86 CPU made in CSS

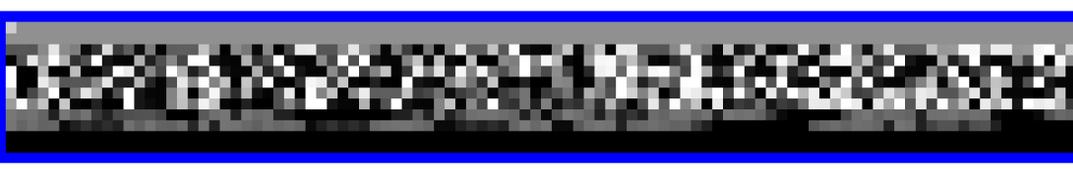
by Lyra Rebane

Your browser does not support the CSS features required to run this demo.

Hold a **Please try with an up-to-date Chromium-based browser. Why? (hide message)**



INST	AX	CX	DX	BX	SP	BP	SI	DI	IP	ES	CS	SS	DS
	0	0	0	0	1528	0	0	0	855	0	0	0	0



About

x86CSS is a working CSS-only x86 CPU/emulator/computer. Yes, the *Cascading Style Sheets* CSS. No JavaScript required.

What you're seeing above is a C program that was compiled using GCC into native 8086 machine code being executed fully within CSS.

GitHub / Fedi, Bluesky, Twitter

Frequently Asked Questions

Is CSS a programming language?

Do you really need to ask at this point?

How??

I plan on writing a blog post that explains how this works as well as many of the tricks used. Bookmark my blog or add it to your RSS reader.

Surely you still need a little bit of JavaScript?

Nope, this is CSS-only!

There is a script tag on this site, which is there to provide a clock to the CSS - but this is only there to make the entire thing a bit faster and more stable. The CSS also has a JS-less clock implementation, so if you disable scripts on this site, it will still run. **JavaScript is not required.**

My CSS clock uses an animation combined with style container queries, which means you don't need to interact with anything for the program to run, but it also means its a bit slower and less stable as a result. A hover-based clock, such as the one in Jane Ori's CPU Hack, is fast and stable, but requires you to hold your mouse on the screen, which some people claim does not count as turing complete for whatever reason, so I wanted this demo to be fully functional with zero user input.

But you still need HTML, right?

Not really... well, kind of?

This entire CPU runs in just CSS and doesn't require any HTML code, but there is no way to load the CSS without a <style> tag, so that much is required. In Firefox it is possible to load CSS with no HTML, but atm this demo only works in Chromium-based browsers.

Why does it only work in Chrome?

This project uses a few CSS features, such as if() statements, style queries, and custom @functions, that are not available in all browsers. At this time, it's only compatible with Chromium-based browsers, but hopefully that'll change at some point since the features used are in the CSS spec. I usually make sure to target Firefox in the projects I make, but it's just not realistic for me with this one.

What preprocessor do you use?

I straight up just write CSS! The CSS in base_template.html is handwritten in Sublime Text, but for the more repetitive parts of the code I wrote a python script.

What ai/model/llm do you use?

I do not use ai. I don't think you can build a project like this with an llm.

Is this practical?

Not really, you can get way better performance by writing code in CSS directly rather than emulating an entire archaic CPU architecture.

It is fun though, and computers are made for art and fun!

Can I write/run my own programs?

Yes, but you'll have to compile them yourself. See below.

Can it run doom?

No, not in its current state.

Have you figured out how to center a div?

If you feel the need to ask, please read this.

You were so preoccupied with whether or not you could that you didn't stop to think if you should

idk why everybody keeps saying that, obviously i thought about whether i should first and came to the conclusion that it would be fun and cool asf to do it

What's x86?

x86 is the CPU architecture most computers these days run on. Heavily simplified, this demo runs the same machine code in CSS that your computer does in its processor. To be fair, modern x86 is 64bit and has a bunch of useful extensions, so it's not quite the same - this here is the original 16bit x86 that ran on the 8086.

What's horse?

neigh.

Compatibility

This project implements most of the x86 architecture, but not literally every single instruction and quirk, because a lot of it is unnecessary and not worth adding.

The way I approached this project was by writing programs I wanted to run in C, compiling them in GCC with various levels of optimization, and then implementing every instruction I needed. This way I know I have everything I need implemented.

There is some behaviour that's wrong, and some things are missing (e.g. setting the CF/OF flag bits). That's okay.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D
0	ADD	ADD	ADD	ADD	ADD	ADD	PUSH	POP	OR	OR	OR	OR	OR	OR
1	ADC	ADC	ADC	ADC	ADC	ADC	PUSH	POP	SBB	SBB	SBB	SBB	SBB	SBB
2	AND	AND	AND	AND	AND	AND	ES:	DAA	SUB	SUB	SUB	SUB	SUB	SUB
3	XOR	XOR	XOR	XOR	XOR	XOR	SS:	AAA	CMP	CMP	CMP	CMP	CMP	CMP
4	INC	INC	INC	INC	INC	INC	INC	INC	DEC	DEC	DEC	DEC	DEC	DEC
5	PUSH	PUSH	PUSH	PUSH	PUSH	PUSH	PUSH	PUSH	POP	POP	POP	POP	POP	POP
6	--	--	--	--	--	--	--	--	--	--	--	--	--	--
7	JO	JNO	JB	JNB	JZ	JNZ	JBE	JA	JS	JNS	JPE	JPO	JL	JGE
8	GRP1	GRP1	GRP1	GRP1	TEST	TEST	XCHG	XCHG	MOV	MOV	MOV	MOV	MOV	LEA
9	NOP	XCHG	XCHG	XCHG	XCHG	XCHG	XCHG	XCHG	CBW	CWD	CALL	WAIT	PUSHF	POPF
A	MOV	MOV	MOV	MOV	MOVSB	MOVSW	CMPSB	CMPSW	TEST	TEST	STOSB	STOSW	LODSB	LODSW
B	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
C	--	--	RET	RET	LES	LDS	MOV	MOV	--	--	RETF	RETF	INT	INT
D	GRP2	GRP2	GRP2	GRP2	AAM	AAD	--	XLAT	--	--	--	--	--	--
E	LOOPNZ	LOOPZ	LOOP	JCXZ	IN	IN	OUT	OUT	CALL	JMP	JMP	JMP	IN	IN
F	LOCK	--	REPNZ	REPZ	HLT	CMC	GRP3a	GRP3b	CLC	STC	CLI	STI	CLD	STD
G	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP	ROL	ROR	RCL	RCR	SHL	SHR
G	TEST	--	NOT	NEG	MUL	IMUL	DIV	IDIV	TEST	--	NOT	NEG	MUL	IMUL
G	INC	DEC	--	--	--	--	--	--	INC	DEC	CALL	CALL	JMP	JMP

Compiling

You can run your own software in this emulator!

If you have 8086 assembly ready to go, clone my repo, and put the assembly in a file called *program.bin*. Then, put the address of the *_start()* function in *program.start* as a number. Once that's set, you can just run *build_css.py* with Python3 (no dependencies required!) and the output will be in *x86css.html*.

If you want to write C code, you can do so using gcc-ia16 (you can build it yourself or install it from the PPA). The *build_c.py* script does the build with the correct flags and also makes the *program.bin/start* files. Don't forget to run *build_css.py* after! This building setup works on both Linux and WSL1/2 (I haven't tried on macOS).

By default there is 0x600 bytes (1.5kB) of memory, but this can be increased in the *build_css.py* file as necessary. The program gets loaded at memory address 0x100. There's some custom I/O addresses for you to be able to interface with the program.

Here's an example program:

```
static const char STR_4BYTES[] = "hell";
static const char STR_8BYTES[] = "o world!";

void (*writeChar1)(char);
void (*writeChar4)(const char[4]);
void (*writeChar8)(const char[8]);
char (*readInput)(void);

int _start(void) {
    // Set up custom stuff
    writeChar1 = (void*)(0x2000);
    writeChar4 = (void*)(0x2002);
    writeChar8 = (void*)(0x2004);
    readInput = (void*)(0x2006);
    int* SHOW_KEYBOARD = (int*)(0x2100);

    // Write a single byte to screen
    writeChar1(0x0a);
    // Write 4 bytes from pointer to screen
    writeChar4(STR_4BYTES);
    // Write 8 bytes from pointer to screen
    writeChar8(STR_8BYTES);
    // Write a character from custom charset
    writeChar1(0x80);

    while (1) {
        // Show numeric keyboard
        *SHOW_KEYBOARD = 1;
        // Read keyboard input
        char input = readInput();
        if (!input) continue;
        *SHOW_KEYBOARD = 0;
        // Echo input
        writeChar1(input);
        break;
    }

    while (1) {
        // Show alphanumeric keyboard
        *SHOW_KEYBOARD = 2;
        char input = readInput();
        if (!input) continue;
        *SHOW_KEYBOARD = 0;
        writeChar1(input);
        break;
    }

    return 1337;
}
```

Credits

gretz/thanks to:

- Jane Ori for the original CPU Hack
- Soo-Young Lee for the 8086 instruction set reference
- mlsite.net for the 8086 opcode map
- crtc.pl for the 8086 opcode map
- poll
- cob

Your browser does not support the CSS features required to run this demo.
Please try with an up-to-date Chromium-based browser. [Why?](#) ([hide message](#)).

Feb 2026