



COMPsoc

Upcoming EGM

Tuesday 27th October
6:10PM in SAT B076

**We are looking for more execs to
help run the society!**

◇ Vice-President

*We don't have
these yet!*

◇ Careers Officer

*We are looking for extras
of these to help!*

Publicity Officer ◇

Events Officer ◇

Swipe for role info!

COMPSOC EGM



◇ Vice-President

- Assist the President in the completion of Presidential duties.
- Aid all other officers to ensure the society runs smoothly.
- In the absence of the President, speak with Presidential authority for matters which do not impact the society's long-term goals.

COMPSOC EGM



◇ Careers Officer

Responsible for:

- Promoting career opportunities to the membership.
- Leading events catered towards the career development of the membership.

COMPSOC EGM



◇ Publicity Officer

Responsible for:

- Publication and announcement of the society & its events.
- Maintaining the society's social media accounts / platforms.
- Producing and organising all marketing materials for the society.
- Jointly responsible with the Treasurer and President for maintaining and procuring publicity supplies (e.g. posters, branded freebies).

COMPSOC EGM



◆ Events Officer

Responsible for:

- Organising small & large-scale events, in coordination with other applicable members of the exec.
- If another exec is leading an event, the Events Officer must monitor progress and ensure the event is well-planned.
- Creating a run-of-show for each event (or ensuring one is created if lead by someone else).



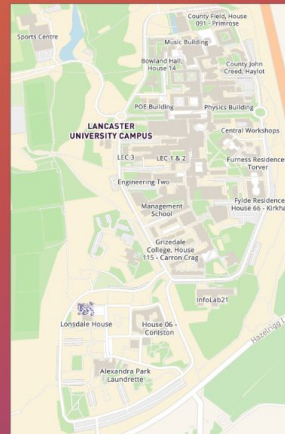


LU CompSoc

Scavanger hunt

Thursday 22nd
October, meeting at
Infolab21

We will sort you
into teams
Small prize for the
winning team



ALL WELCOME

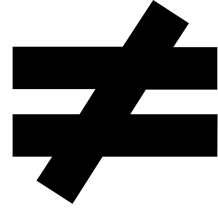
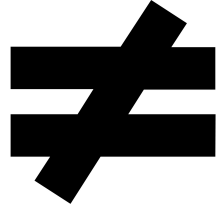


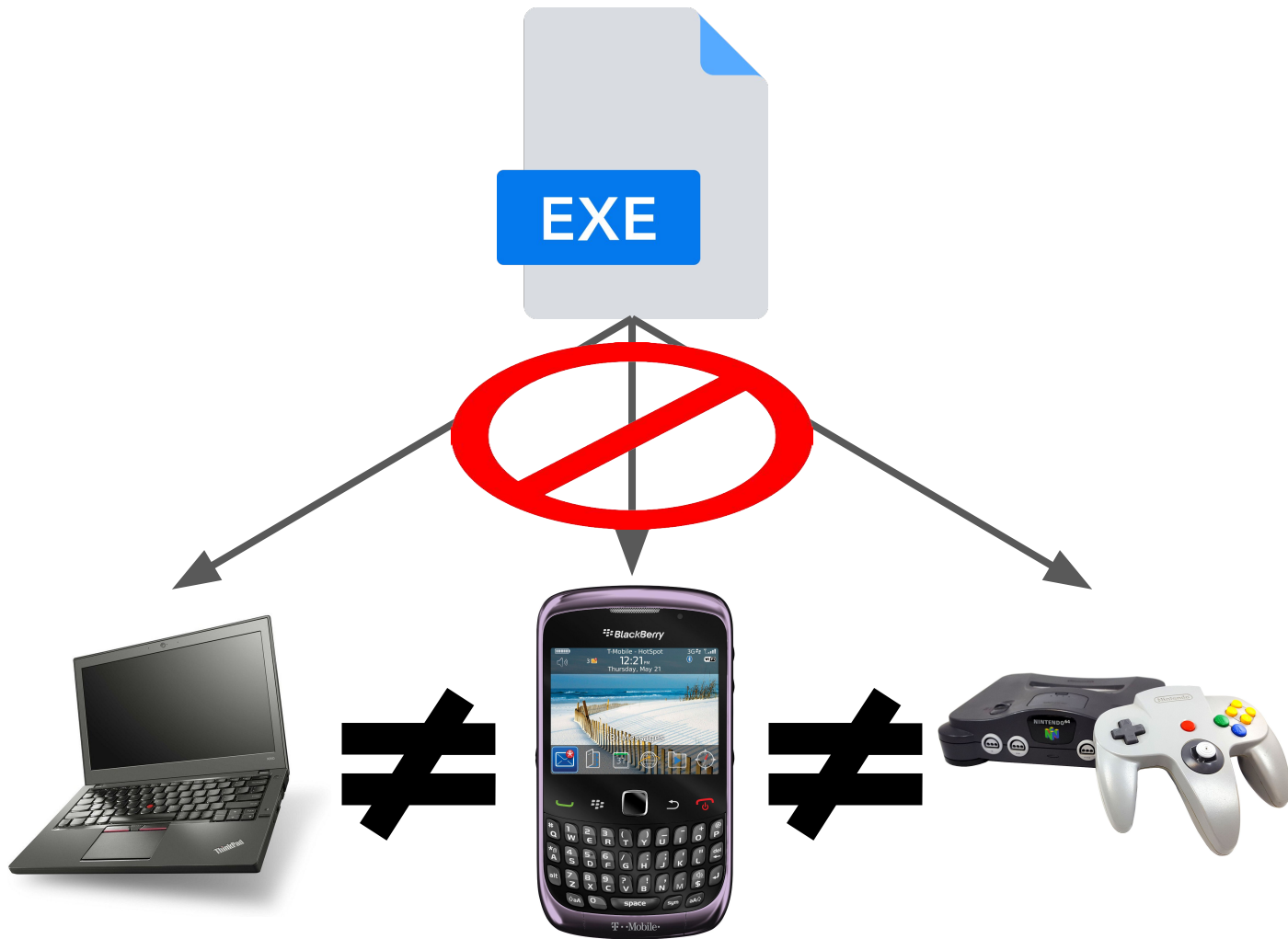
Intro to Emulation

What is Emulation?

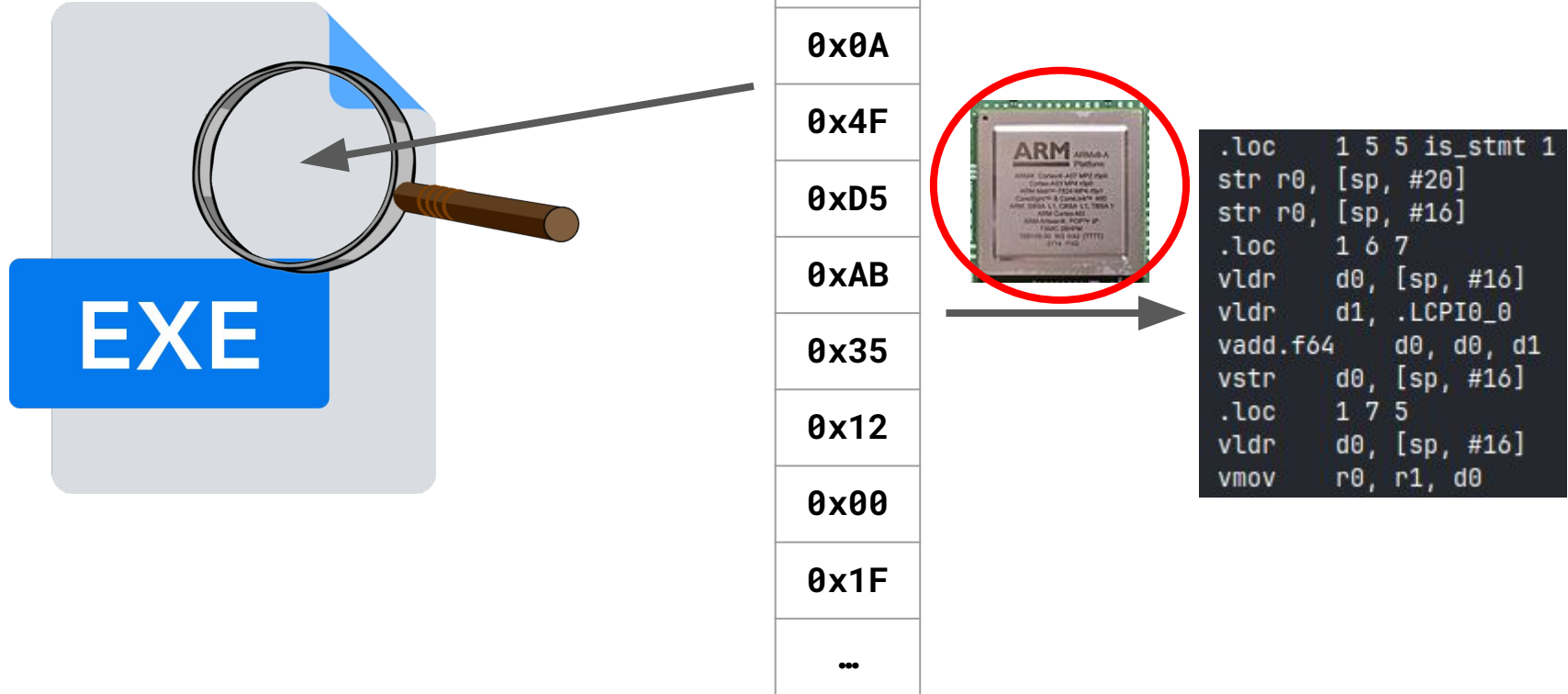
The act of a computer system posing as another computer system.



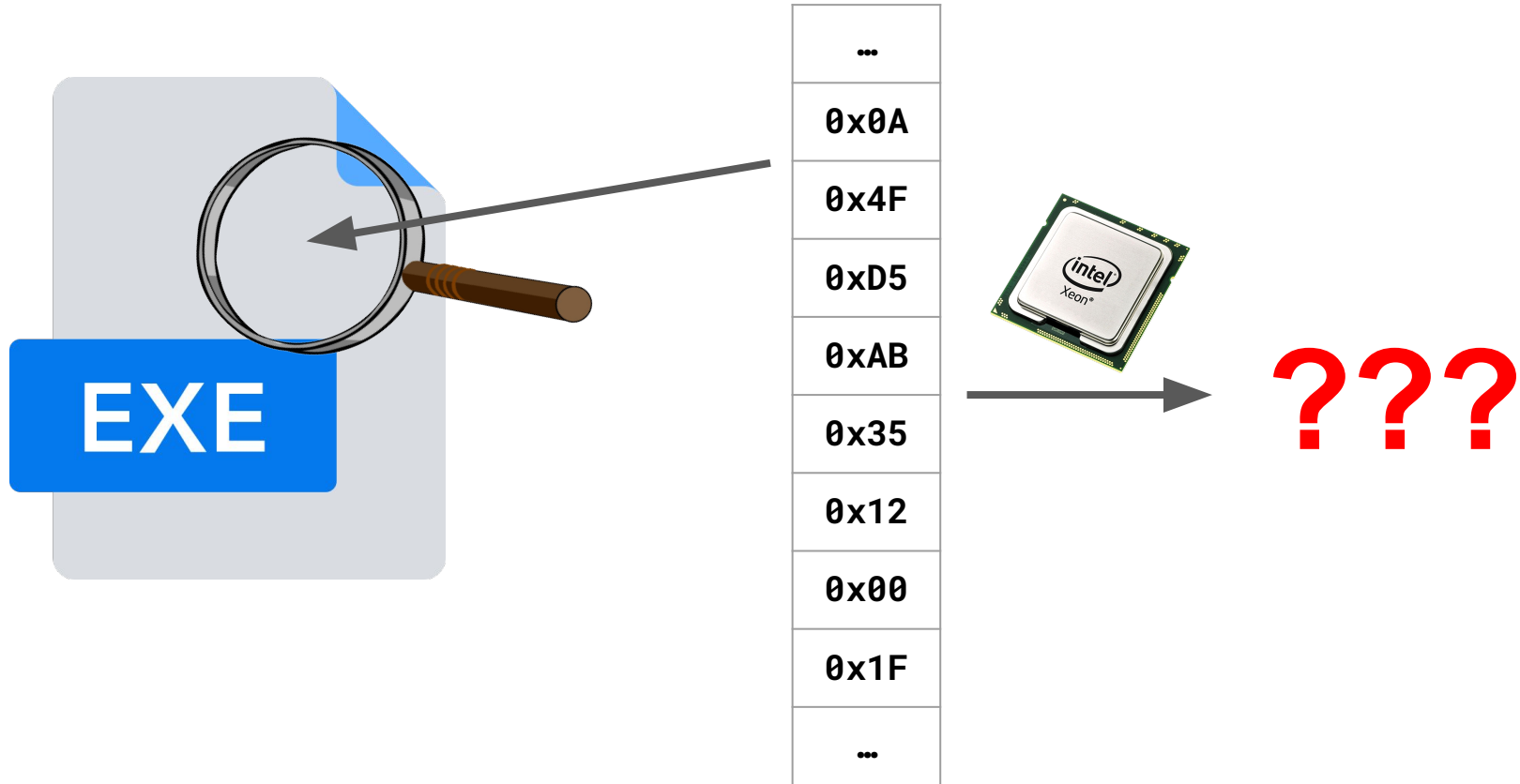




The Biggest Problem: Different CPUs



The Biggest Problem: Different CPUs



Different Peripheral Management



Can I use the
CD drive
please?

bro i am
literally a
Steam Deck

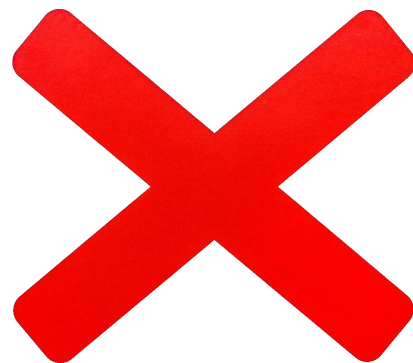


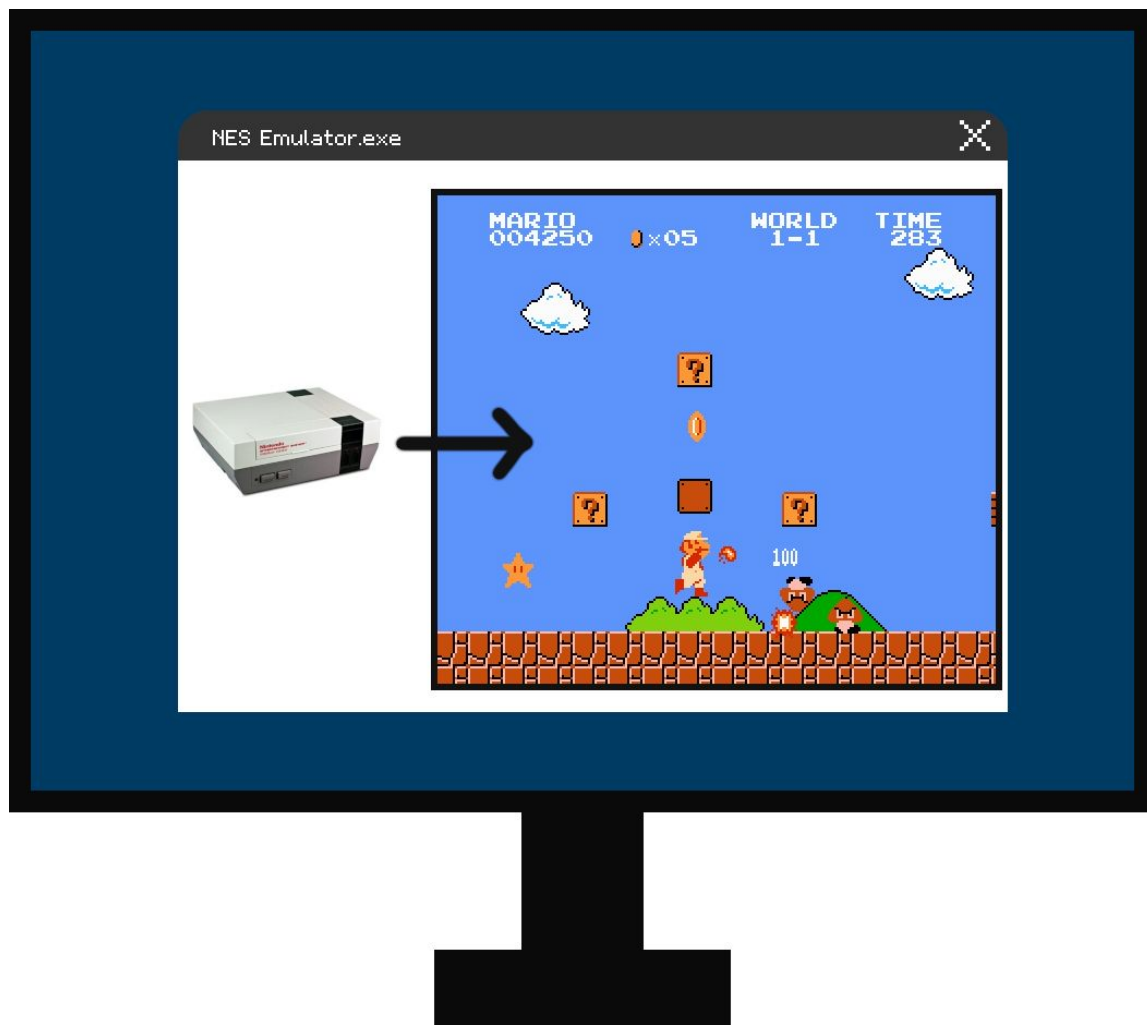
Even more differences



Even more differences





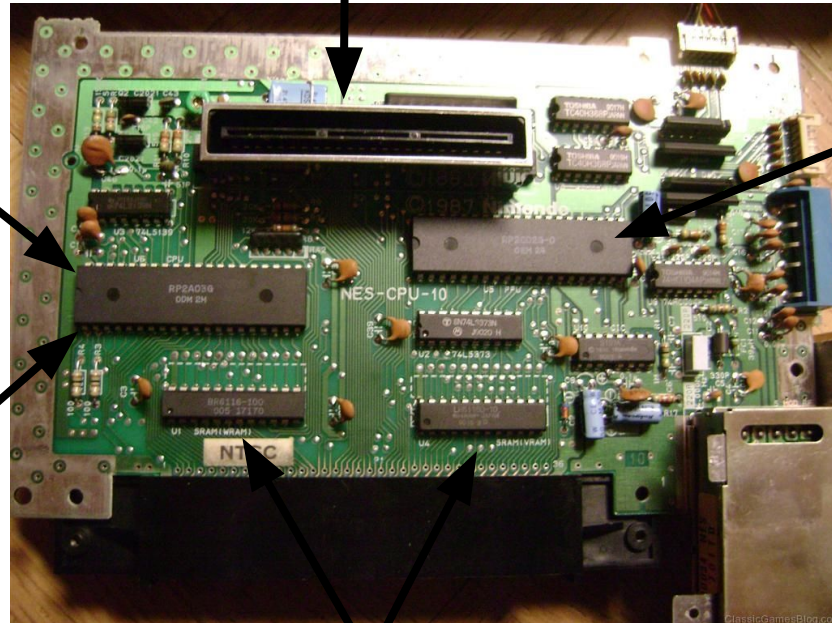


```
class CartridgeSlot {  
    ...  
}
```

```
class CPU {  
    ...  
}
```

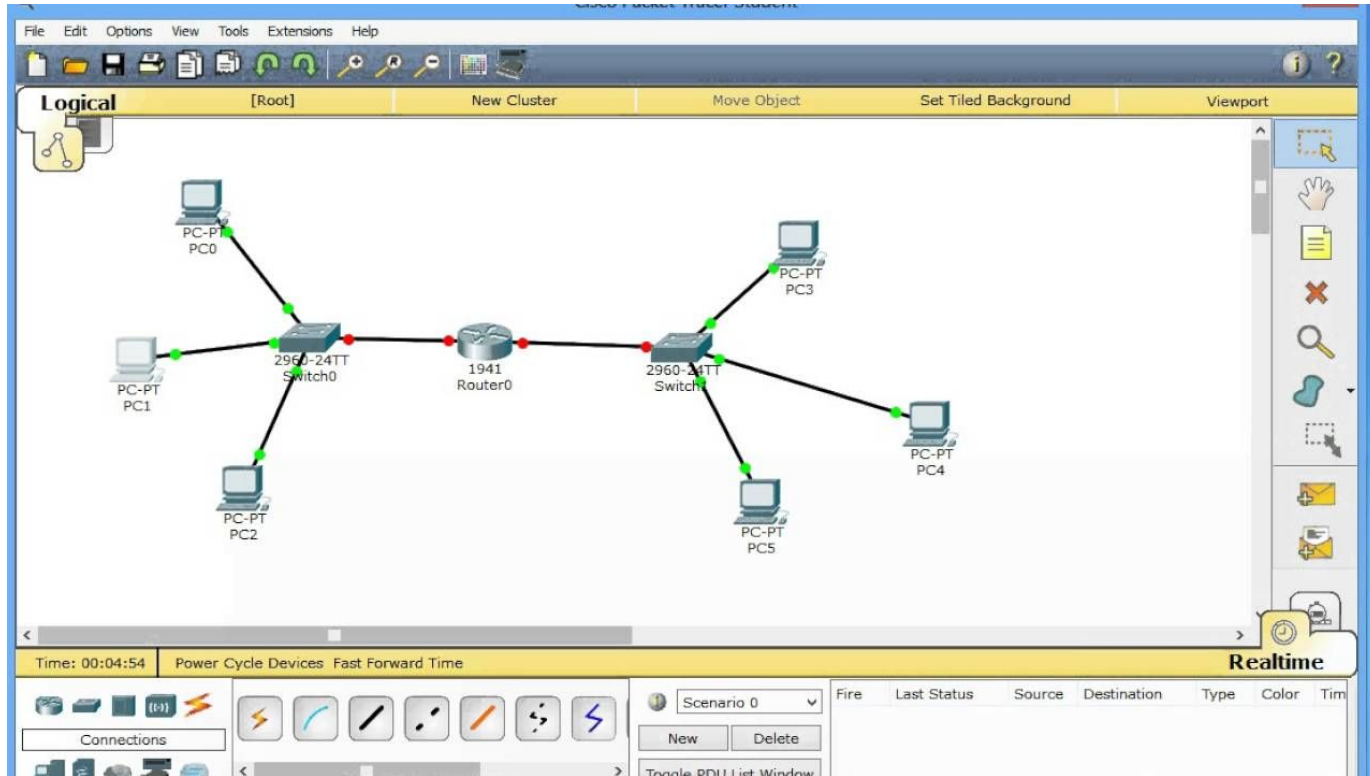
```
class PPU {  
    ...  
}
```

```
class APU {  
    ...  
}
```



```
class RAM {  
    ...  
}
```

Emulating networks



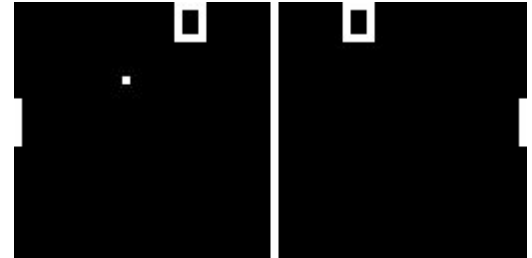
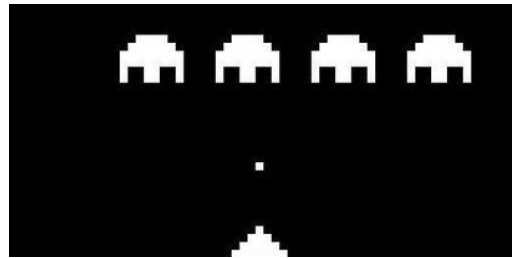
Let's try making an emulator!

The logo for CHIP-8, featuring the text "CHIP-8" in a white, pixelated, monospace-style font. The letters are blocky and have a jagged, digital appearance. The "C" is a simple square outline. The "H" is formed by two vertical bars connected at the top and bottom. The "I" is a single vertical bar. The "P" has a vertical bar on the left and a horizontal bar at the top right. The "-" is a simple horizontal line. The "8" is composed of two stacked circles, each formed by a square outline. The entire logo is set against a solid black rectangular background.

CHIP-8

CHIP-8

Chip8 is a virtual system with its own interpreted assembly-like language, often used for games.



<https://tobiasvl.github.io/blog/write-a-chip-8-emulator/>

Specifications

CHIP-8 has the following components:

- *Memory*: CHIP-8 has direct access to up to 4 kilobytes of RAM
- *Display*: 64 x 32 pixels (or 128 x 64 for SUPER-CHIP) monochrome, ie. black or white
- A *program counter*, often called just "PC", which points at the current instruction in memory
- One 16-bit *index register* called "I" which is used to point at locations in memory
- A *stack* for 16-bit addresses, which is used to call subroutines/functions and return from them
- An 8-bit *delay timer* which is decremented at a rate of 60 Hz (60 times per second) until it reaches 0
- An 8-bit *sound timer* which functions like the delay timer, but which also gives off a beeping sound as long as it's not 0
- 16 8-bit (one byte) general-purpose *variable* registers numbered 0 through F hexadecimal, ie. 0 through 15 in decimal, called V0 through VF
 - VF is also used as a *flag register*; many instructions will set it to either 1 or 0 based on some rule, for example using it as a carry flag

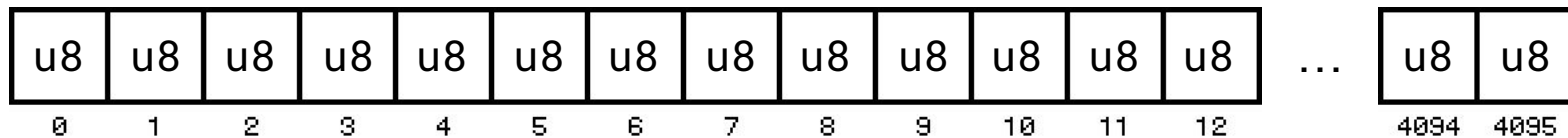
Display

A classic Chip8 system outputs a 64x32 image of binary pixels (black or white) which can be toggled on or off.

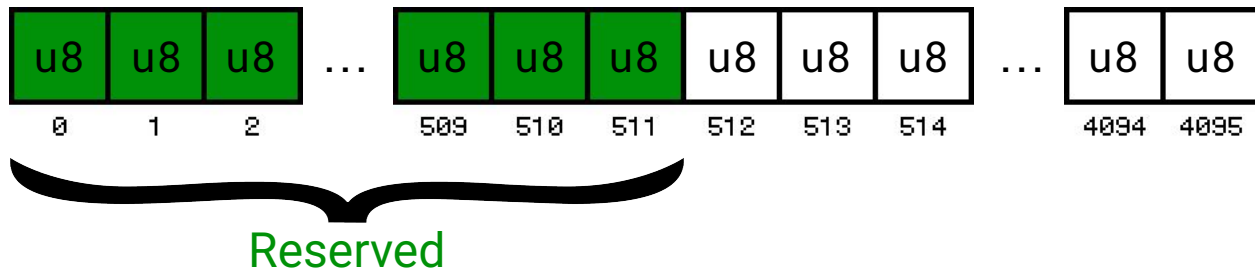


Memory

The Chip8 has 4KB (4096 bytes) of RAM, which contains all the data the system has access to at any given time.



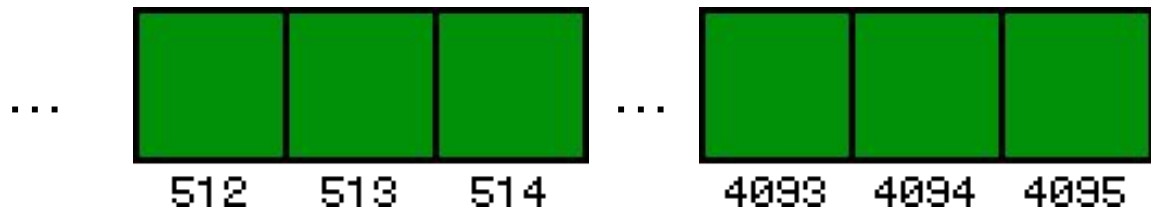
Memory



The first 512 bytes are **reserved**, meaning they aren't meant to be written to by a running program.

Extra note: The bytes 0x50 (80) to 0xD0 (208) often store font data preloaded for any program that wants to draw text.

Memory



Bytes from 512 onwards store the running program.

CPU Registers

V0
V1
V2
...
VF



Sixteen 16-bit registers labelled **V0** to **VF**.

They can all store numbers from 0-65535.

Base 10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

CPU Registers

V0
V1
V2
...
VF



VF is used by many arithmetic instructions to store carry bits.

CPU Registers



Another 16-bit register **I**, which is often used to point to sprite data.



Stands for **Program Counter**, a 16-bit value which stores the address of the next instruction to be executed.

Timers

Two 8-bit (0-255) timers which go down by one 60 times per second until they reach zero.

Delay Timer
Sound Timer



The sound timer plays a beep sound as long as it's above 0.

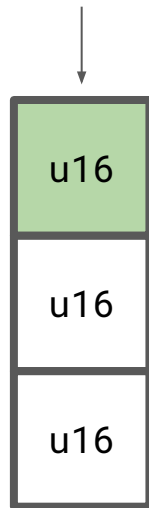
Stack

The stack consists of 16-bit values which can be added to and removed from the top.

It's often used for functions, since you can keep track of where they can return to.

The stack can be as large as you want, but historically most programs don't expect a stack with more than 16 elements.

Can only add or
remove from here



And that's it!

```
#define STACK_SIZE 16

struct System {
    u8 ram[4096];

    u8 display[DISPLAY_WIDTH*DISPLAY_HEIGHT];

    u16 pc;

    u8 stack_data[STACK_SIZE];
    u8 stack_length;

    u8 delay_timer;
    u8 sound_timer;

    u8 registers[16];
    u16 I;
};
```

But how do you control it?



Emulating the CPU

In other words, we need to emulate the Fetch-Decode-Execute cycle of the CPU.

[illegible]

00 E0	61 01	60 08	A2 50	D0 1F	60 10	A2 5F	D0 1F
60 18	A2 6E	D0 1F	60 20	A2 7D	D0 1F	60 28	A2 8C
D0 1F	60 30	A2 9B	D0 1F	61 10	60 08	A2 AA	D0 1F
60 10	A2 B9	D0 1F	60 18	A2 C8	D0 1F	60 20	A2 D7
D0 1F	60 28	A2 E6	D0 1F	60 30	A2 F5	D0 1F	12 4E
0F 02	02 02	02 02	00 00	1F 3F	71 E0	E5 E0	E8 A0
0D 2A	28 28	28 00	00 18	B8 B8	38 38	3F BF	00 19
A5 BD	A1 9D	00 00	0C 1D	1D 01	0D 1D	9D 01	C7 29
29 29	27 00	00 F8	FC CE	C6 C6	C6 C6	00 49	4A 49
48 3B	00 00	00 01	03 03	03 01	F0 30	90 00	00 80
00 00	00 FE	C7 83	83 83	C6 FC	E7 E0	E0 E0	E0 71
3F 1F	00 00	07 02	02 02	02 39	38 38	38 38	B8 B8
38 00	00 31	4A 79	40 3B	DD DD	DD DD	DD DD	DD DD
00 00	A0 38	20 A0	18 CE	FC F8	C0 D4	DC C4	C5 00
00 30	44 24	14 63	F1 03	07 07	77 17	63 71	00 00
28 8E	A8 A8	A6 CE	87 03	03 03	87 FE	FC 00	00 60
90 F0	80 70						

Fetch

00E0



00	E0	61	01	60	08	A2	50	D0	1F	60	10	A2	5F	D0	1F
60	18	A2	6E	D0	1F	60	20	A2	7D	D0	1F	60	28	A2	8C
D0	1F	60	30	A2	9B	D0	1F	61	10	60	08	A2	AA	D0	1F
60	10	A2	B9	D0	1F	60	18	A2	C8	D0	1F	60	20	A2	D7
D0	1F	60	28	A2	E6	D0	1F	60	30	A2	F5	D0	1F	12	4E
0F	02	02	02	02	02	00	00	1F	3F	71	E0	E5	E0	E8	A0
0D	2A	28	28	28	00	00	18	B8	B8	38	38	3F	BF	00	19
A5	BD	A1	9D	00	00	0C	1D	1D	01	0D	1D	9D	01	C7	29
29	29	27	00	00	F8	FC	CE	C6	C6	C6	C6	00	49	4A	49
48	3B	00	00	00	01	03	03	03	01	F0	30	90	00	00	80
00	00	00	FE	C7	83	83	83	C6	FC	E7	E0	E0	E0	E0	71
3F	1F	00	00	07	02	02	02	02	39	38	38	38	38	B8	B8
38	00	00	31	4A	79	40	3B	DD	DD	DD	DD	DD	DD	DD	DD
00	00	A0	38	20	A0	18	CE	FC	F8	C0	D4	DC	C4	C5	00
00	30	44	24	14	63	F1	03	07	07	77	17	63	71	00	00
28	8E	A8	A8	A6	CE	87	03	03	03	87	FE	FC	00	00	60
90	F0	80	70												

PC: 512

Fetch

6101



PC: 514

[illegible]

Fetch

6008



PC: 516

[illegible]

Decode

The regular Chip8 has **34** instructions.

Machinecode	Octo Instruction	Comments
00E0	clear	Exit a subroutine Call a subroutine
00EE	return	
1NNN	jump NNN	
2NNN	NNN	
3XNN	if vx != NN then	
4XNN	if vx == NN then	
5XY0	if vx != vy then	
6XNN	vx := NN	
7XNN	vx += NN	
8XY0	vx := vy	
8XY1	vx = vy	Bitwise OR
8XY2	vx &= vy	Bitwise AND
8XY3	vx ^= vy	Bitwise XOR
8XY4	vx += vy	vf = 1 on carry
8XY5	vx -= vy	vf = 0 on borrow
8XY6	vx >>= vy	vf = old least significant bit
8XY7	vx <= vy	vf = 0 on borrow
8XYE	vx <<= vy	vf = old most significant bit
9XY0	if vx == vy then	Jump to address NNN + v0 Random number 0-255 AND NN
ANNN	i := NNN	
BNNN	jump0 NNN	
CXNN	vx := random NN	vf = 1 on collision
DXYN	sprite vx vy N	
EX9E	if vx -key then	Is a key not pressed?
EXA1	if vx key then	Is a key pressed?
FX07	vx := delay	Wait for a keypress
FX0A	vx := key	
FX15	delay := vx	
FX18	buzzer := vx	
FX1E	i += vx	
FX29	i := hex vx	Set i to a hex character
FX33	bcd vx	Decode vx into binary-coded decimal
FX55	save vx	Save v0-vx to i through (i+x)
FX65	load vx	Load v0-vx from i through (i+x)

```

00 E0 61 01 60 08 A2 50 D0 1F 60 10 A2 5F D0 1F
60 18 A2 6E D0 1F 60 20 A2 7D D0 1F 60 28 A2 8C
D0 1F 60 30 A2 9B D0 1F 61 10 60 08 A2 AA D0 1F
60 10 A2 B9 D0 1F 60 18 A2 C8 D0 1F 60 20 A2 D7
D0 1F 60 28 A2 E6 D0 1F 60 30 A2 F5 D0 1F 12 4E
0F 02 02 02 02 02 00 00 1F 3F 71 E0 E5 E0 E8 A0
0D 2A 28 28 28 00 00 18 B8 B8 38 38 3F BF 00 19
A5 BD A1 9D 00 00 0C 1D 1D 01 0D 1D 9D 01 C7 29
29 29 27 00 00 F8 FC CE C6 C6 C6 C6 00 49 4A 49
48 3B 00 00 00 01 03 03 03 01 F0 30 90 00 00 80
00 00 00 FE C7 83 83 83 C6 FC E7 E0 E0 E0 E0 71
3F 1F 00 00 07 02 02 02 02 39 38 38 38 38 B8 B8
38 00 00 31 4A 79 40 3B DD DD DD DD DD DD DD DD
00 00 A0 38 20 A0 18 CE FC F8 C0 D4 DC C4 C5 00
00 30 44 24 14 63 F1 03 07 07 77 17 63 71 00 00
28 8E A8 A8 A6 CE 87 03 03 03 87 FE FC 00 00 60
90 F0 80 70

```

Machinecode	Octo Instruction
00E0	clear

Machinecode	Octo Instruction	Comments
00E0	clear	Exit a subroutine Call a subroutine
00EE	return	
1NNN	jump NNN	
2NNN	NNN	
3XNN	if vx != NN then	
4XNN	if vx == NN then	
5XY0	if vx != vy then	
6XNN	vx := NN	
7XNN	vx += NN	
8XY0	vx := vy	
8XY1	vx = vy	
8XY2	vx &= vy	
8XY3	vx ^= vy	
8XY4	vx += vy	Bitwise OR
8XY5	vx -= vy	Bitwise AND
8XY6	vx >>= vy	Bitwise XOR
8XY7	vx <= vy	vf = 1 on carry
8XYE	vx <<= vy	vf = 0 on borrow
9XY0	if vx == vy then	vf = old least significant bit
ANNN	i := NNN	vf = 0 on borrow
BNNN	jump0 NNN	vf = old most significant bit
CXNN	vx := random NN	Jump to address NNN + v0 Random number 0-255 AND NN
DXYN	sprite vx vy N	
EX9E	if vx -key then	
EXA1	if vx key then	
FX07	vx := delay	
FX0A	vx := key	
FX15	delay := vx	
FX18	buzzer := vx	
FX1E	i += vx	
FX29	i := hex vx	
FX33	bcd vx	
FX55	save vx	
FX65	load vx	

?

00E0	5XY0	8XY3	9XY0	EXA1	FX29
00EE	6XNN	8XY4	ANNN	FX07	FX33
1NNN	7XNN	8XY5	BNNN	FX0A	FX55
2NNN	8XY0	8XY6	CXNN	FX15	FX65
3XNN	8XY1	8XY7	DXYN	FX18	
4XNN	8XY2	8XYE	EX9E	FX1E	

00E0	5XY0	8XY3	9XY0	EXA1	FX29
00EE	6XNN	8XY4	ANNN	FX07	FX33
1NNN	7XNN	8XY5	BNNN	FX0A	FX55
2NNN	8XY0	8XY6	CXNN	FX15	FX65
3XNN	8XY1	8XY7	DXYN	FX18	
4XNN	8XY2	8XYE	EX9E	FX1E	

Parameters	
X	The second nibble of the opcode.
Y	The third nibble of the opcode.
N	The fourth nibble of the opcode.
NN	The last two nibbles of the opcode together (second byte).
NNN	The last three nibbles of the opcode together.

8XY4

8XY4	$vx += vy$	$vf = 1$ on carry
------	------------	-------------------

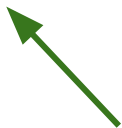
Adds the value of **Y** directly to **X**, and sets **VF** to 1 if it overflows.

8**3****4**4



8XY4	$vx += vy$	$vf = 1$ on carry
------	------------	-------------------

Adds the value of **V****4** directly to **V****3**, and sets **VF** to 1 if it overflows.



87B4



8XY4	<code>vx += vy</code>	<code>vf = 1 on carry</code>
------	-----------------------	------------------------------

Adds the value of **vB** directly to **v7**, and sets **VF** to 1 if it overflows.



Fetch 

6532

00E0	5XY0	8XY3	9XY0	EXA1	FX29
00EE	6XNN	8XY4	ANNN	FX07	FX33
1NNN	7XNN	8XY5	BNNN	FX0A	FX55
2NNN	8XY0	8XY6	CXNN	FX15	FX65
3XNN	8XY1	8XY7	DXYN	FX18	
4XNN	8XY2	8XYE	EX9E	FX1E	

Fetch →

6532

00E0	5XY0	8XY3	9XY0	EXA1	FX29
00EE	6XNN	8XY4	ANNN	FX07	FX33
1NNN	7XNN	8XY5	BNNN	FX0A	FX55
2NNN	8XY0	8XY6	CXNN	FX15	FX65
3XNN	8XY1	8XY7	DXYN	FX18	
4XNN	8XY2	8XYE	EX9E	FX1E	

6532

00E0	5XY0	8XY3	9XY0	EXA1	FX29
00EE	6XNN	8XY4	ANNN	FX07	FX33
1NNN	7XNN	8XY5	BNNN	FX0A	FX55
2NNN	8XY0	8XY6	CXNN	FX15	FX65
3XNN	8XY1	8XY7	DXYN	FX18	
4XNN	8XY2	8XYE	EX9E	FX1E	

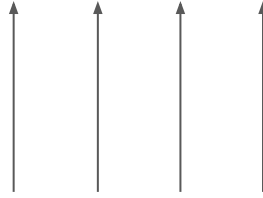
Machinecode	Octo Instruction	Comments
00E0	clear	
00EE	return	Exit a subroutine
1NNN	jump NNN	
2NNN	NNN	Call a subroutine
3XNN	if vx != NN then	
4XNN	if vx == NN then	
5XY0	if vx != vy then	
6XNN	vx := NN	
7XNN	vx += NN	
8XY0	vx := vy	
8XY1	vx = vy	Bitwise OR
8XY2	vx &= vy	Bitwise AND
8XY3	vx ^= vy	Bitwise XOR
8XY4	vx += vy	vf = 1 on carry
8XY5	vx -= vy	vf = 0 on borrow
8XY6	vx >>= vy	vf = old least significant bit
8XY7	vx <<= vy	vf = 0 on borrow
8XYE	vx <<= vy	vf = old most significant bit
9XY0	if vx == vy then	
ANNN	i := NNN	
BNNN	jump0 NNN	Jump to address NNN + v0
CXNN	vx := random NN	Random number 0-255 AND NN
DXYN	sprite vx vy N	vf = 1 on collision
EX9E	if vx -key then	Is a key not pressed?
EXA1	if vx key then	Is a key pressed?
FX07	vx := delay	
FX0A	vx := key	
FX15	delay := vx	Wait for a keypress
FX18	buzzer := vx	
FX1E	i += vx	
FX29	i := hex vx	Set i to a hex character
FX33	bcd vx	Decode vx into binary-coded decimal
FX55	save vx	Save v0-vx to i through (i+x)
FX65	load vx	Load v0-vx from i through (i+x)

6XNN: Set register VX to the value NN.

16-bit



6532



?

Bit Manipulation: & Operator

	0
&	0
	0

	0
&	1
	0

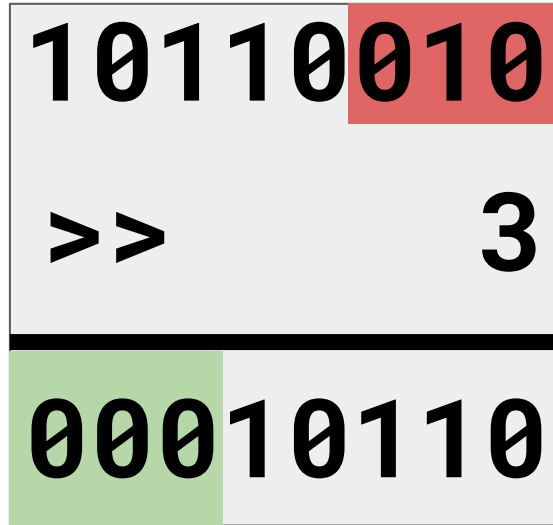
	1
&	0
	0

	1
&	1
	1

Bit Manipulation: & Operator

	0	1	1	0	= 6
&	0	1	0	1	= 5
	0	1	0	0	= 4

Bit Manipulation: Bit Shifts



6XNN

& 0F00

0X00

>> 8

000X

0110 XXXX NNNN NNNN

0000 1111 0000 0000

0000 XXXX 0000 0000

0000 0000 0000 XXXX

```
(instruction & 0x0F00) >> 8
```

BNNN

& 0FFF

0NNN

1011 **NNNN NNNN NNNN**

0000 1111 1111 1111

0000 **NNNN NNNN NNNN**

Execute

00E0	5XY0	8XY3	9XY0	EXA1	FX29
00EE	6XNN	8XY4	ANNN	FX07	FX33
1NNN	7XNN	8XY5	BNNN	FX0A	FX55
2NNN	8XY0	8XY6	CXNN	FX15	FX65
3XNN	8XY1	8XY7	DXYN	FX18	
4XNN	8XY2	8XYE	EX9E	FX1E	

Machinecode	Octo Instruction	Comments
00E0	clear	Exit a subroutine Call a subroutine
00EE	return	
1NNN	jump NNN	
2NNN	NNN	
3XNN	if vx != NN then	
4XNN	if vx == NN then	
5XY0	if vx != vy then	
6XNN	vx := NN	
7XNN	vx += NN	
8XY0	vx := vy	
8XY1	vx = vy	Bitwise OR
8XY2	vx &= vy	Bitwise AND
8XY3	vx ^= vy	Bitwise XOR
8XY4	vx += vy	vf = 1 on carry
8XY5	vx -= vy	vf = 0 on borrow
8XY6	vx >>= vy	vf = old least significant bit
8XY7	vx <= vy	vf = 0 on borrow
8XYE	vx <<= vy	vf = old most significant bit
9XY0	if vx == vy then	Jump to address NNN + v0 Random number 0-255 AND NN
ANNN	i := NNN	
BNNN	jump0 NNN	
CXNN	vx := random NN	
DXYN	sprite vx vy N	vf = 1 on collision
EX9E	if vx -key then	Is a key not pressed?
EXA1	if vx key then	Is a key pressed?
FX07	vx := delay	Wait for a keypress
FX0A	vx := key	
FX15	delay := vx	
FX18	buzzer := vx	
FX1E	i += vx	
FX29	i := hex vx	Set i to a hex character
FX33	bcd vx	Decode vx into binary-coded decimal
FX55	save vx	Save v0-vx to i through (i+x)
FX65	load vx	Load v0-vx from i through (i+x)

DXYN

Used to render a sprite.

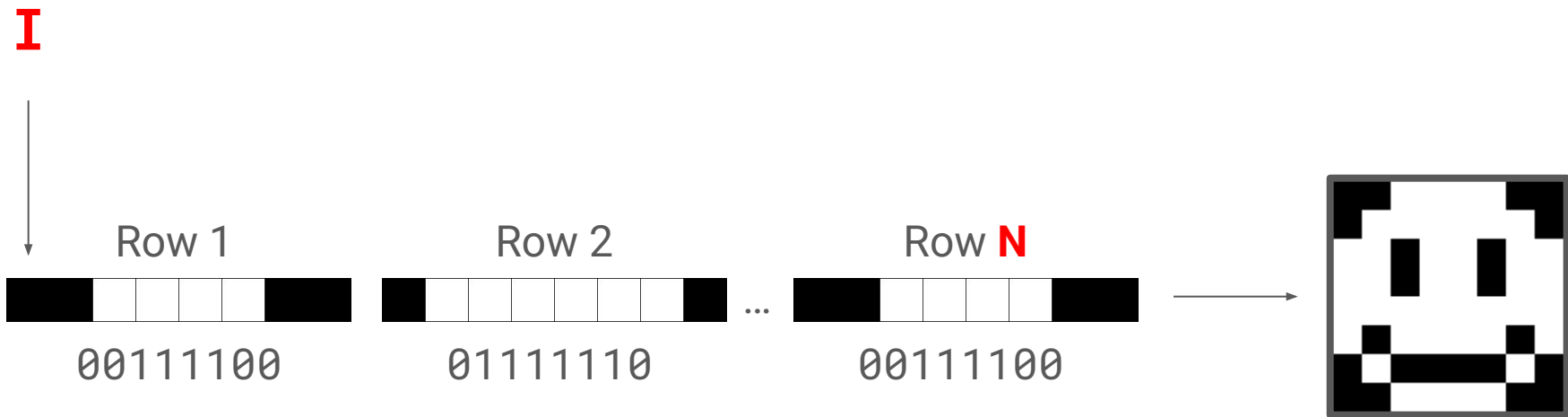
Sprites on the Chip8 are always 8 pixels wide, and can be from 1 to 15 pixels high.

N is the height of the sprite being rendered.



DXYN

The sprite data is read from RAM, and starts at the address stored in **I**.

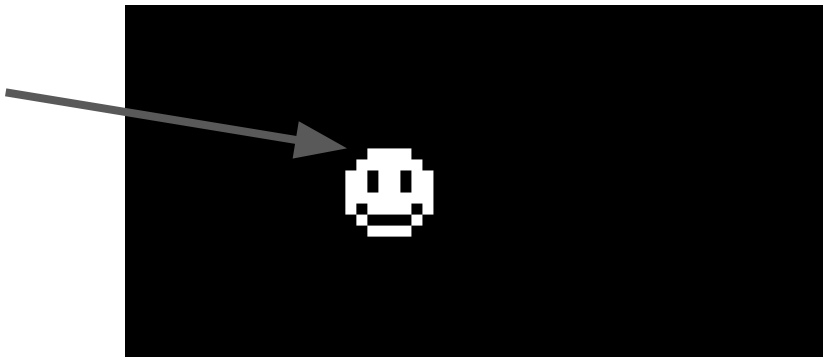


DXYN

The sprite is then rendered at the coordinates (**vx**, **vy**).

The output pixels are the pixels currently in the region to be rendered XORed with the sprite's pixels.

(**vx**, **vy**)



Let's see some code!

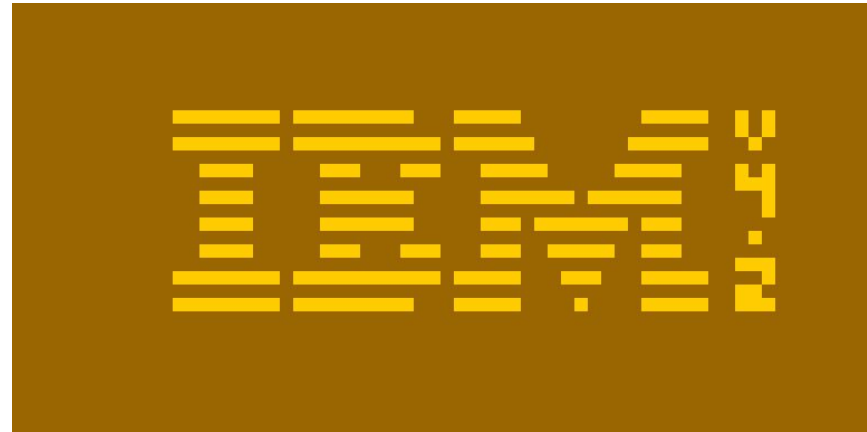
On GitHub there is a Chip8 emulator in C that has already been started.

<https://github.com/LUCompSoc/chip8-demo>

It is coded to load a program showing the IBM logo on it, but only two of the necessary instructions are completed.

Your task is to fill in the remaining instructions and get the program running.

Jump to instruction	1NNN
Set register VX to NN	6XNN
Add NN to VX (no carry bit changes!)	7XNN
Set index register to NNN	ANNN



Once you've done this, feel free to implement as many instructions as you can and try running some other programs!