Encodings

Level 0x02

Quick Overview

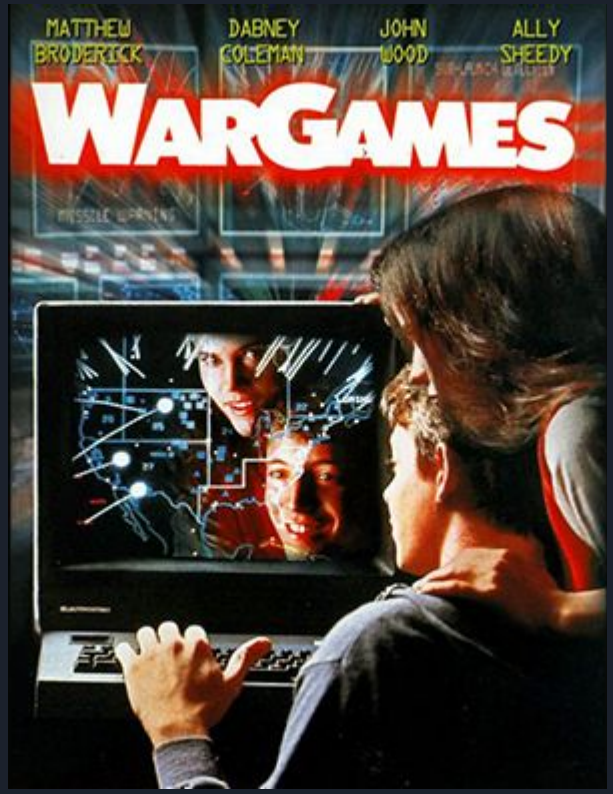
- Upcoming Events
- Encoding

Orlando B-Sides

- Sunshine CTF / HackUCF
  - 10AM Friday 9/27 - 10AM Sunday
  - Past years have been all skill levels
- B-Sides Conference
  - Free for students
  - 9AM - 6PM Saturday 9/28
  - See past year videos on youtube. Search “B-Sides Orlando” on youtube
  - 3 talk tracks
  - Villages (lockpicking, rf, career, soldering)

Wargames (1983)

- Required Hacker Viewing!
  - War dialing / modems
  - Changing grades
  - Hacking into military computers
  - Almost starting WW3
- 93% on Rotten Tomatoes



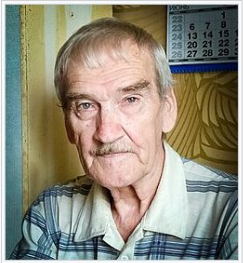
1983 Soviet nuclear false alarm incident

12 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

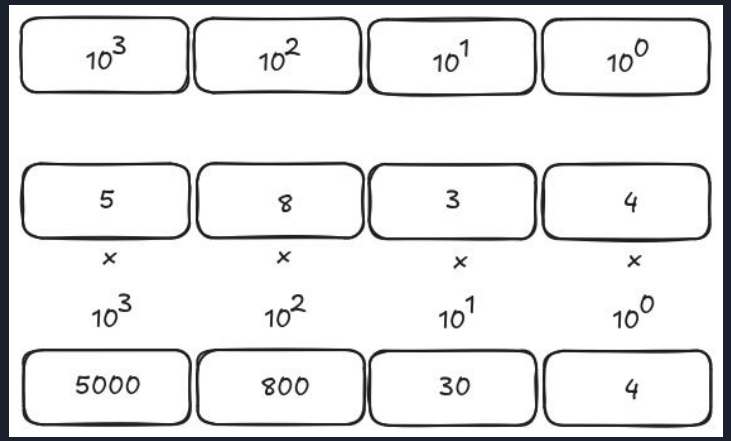
On 26 September 1983, during the Cold War, the Soviet nuclear early warning system Oko reported the launch of one intercontinental ballistic missile with four more missiles behind it, from the United States. These missile attack warnings were suspected to be false alarms by Stanislav Petrov, an engineer of the Soviet Air Defence Forces on duty at the command center of the early-warning system. He decided to wait for corroborating evidence—of which none arrived—rather than immediately relaying the warning up the chain of command. This decision is seen as having prevented a retaliatory nuclear strike against the United States and its NATO allies, which would likely have resulted in a full-scale nuclear war. Investigation of the satellite warning system later determined that the system had indeed malfunctioned.

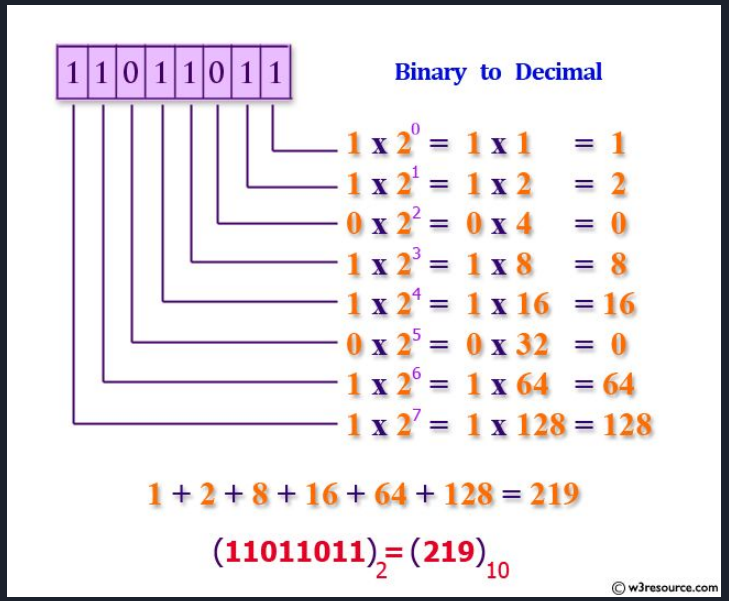


Petrov in 2016

Binary Numbers

- LSB = Least Significant Bit
- MSB = Most Significant Bit
- Python, C, C++:
  - `0b10101100`
- Each digit is $(BASE)^X$





Hexadecimal Numbers

- Base 16
- Python and C:
  - 0x23
- Python conversions?

DECIMAL	HEX	BINARY
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Hexadecimal Numbers

- Base 16
- Python and C:
  - 0x23
- Python:
  - `hex(255)`
  - `int("ff",16)`
  - `int("0xff",0)`
- C / C++ ?

DECIMAL	HEX	BINARY
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Hexadecimal Numbers

- Base 16
- Python and C:
  - 0x23
- Python:
  - `hex(255)`
  - `int("ff",16)`
  - `int("0xff",0)`
- C / C++:
  - `printf("%x", 255);`
  - `strtol("ff", NULL, 16);`
- Java ?

DECIMAL	HEX	BINARY
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Hexadecimal Numbers

- Base 16
- Python and C:
  - 0x23
- Python:
  - `hex(255)`
  - `int("ff",16)`
  - `int("0xff",0)`
- C / C++:
  - `printf("%x", 255);`
  - `strtol("ff", NULL, 16);`
- Java:
  - `Integer.parseInt("ff",16);`

DECIMAL	HEX	BINARY
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

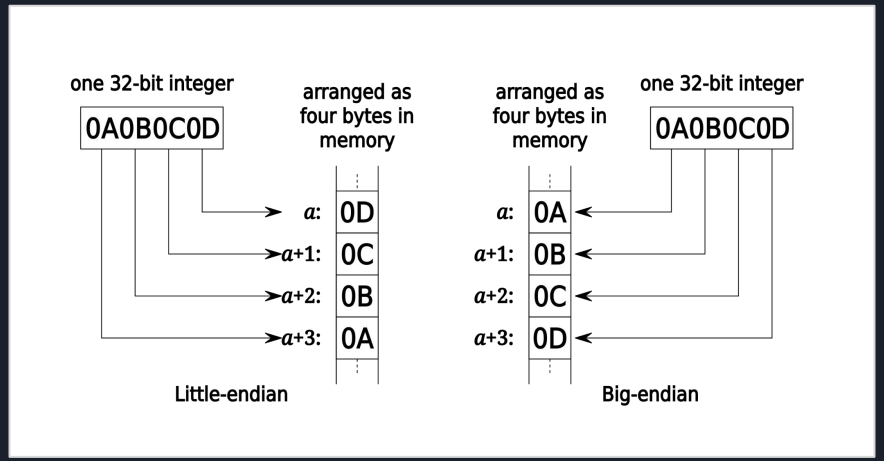
Words

- 8 bits = 1 Byte
- Word Length = Register Size
  - 6502 = 8-bit
  - X86 = 16-bit, then 32-bit
  - ARM = 32-bit
  - X86-64 = 64-bit
- Windows API
  - WORD = 16-bit
  - DWORD = 32-bit
  - QWORD = 64-bit

1960	PDP-1	18 bit
1960	Elliott 803	39 bit
1961	IBM 7030 (Stretch)	64 bit
1961	IBM 7080	*n* c
1962	GE-6xx	36 bit
1962	UNIVAC III	25 bit
1962	Autonetics D-17B Minuteman I Guidance Computer	27 bit
1962	UNIVAC 1107	36 bit
1962	IBM 7010	*n* c
1962	IBM 7094	36 bit
1962	SDS 9 Series	24 bit
1963 (1966)	Apollo Guidance Computer	15 bit
1963	Saturn Launch Vehicle Digital Computer	26 bit

Endianness

- Big-endian:
  - Power PC, networking
- Little-endian:
  - X86, ARM



Octal

- Base 8 numbering system
- Python:
  - oct(65535) = 0o177777
- C / C++
  - printf(“%o”, 65535) = 0177777
- Bash / Linux:
  - od

```
$echo "hello" | od -
0000000 062550 066154 005157
0000006
```

Executable Files

- Linux - permissions bits
  - Permission bits for user, then group, then others
  - r = read, w = write, x = executable
  - ```
    $ ls -l
    -rwxrwxr-x 1 mwales mwales 16784 Feb  1  2023 a.out
    -rw-rw-r-- 1 mwales mwales    26 Feb  1  2023 flag.txt
    -rwxrwxr-x 1 mwales mwales  3969 Feb  3  2023 judge.py
    -r[illegible]- 1 mwales mwales   330 Feb  2  2023 solution.c
    ```

    
  - File permissions have 3 groups (from left to right)
    - user / Owner
    - group
    - Others
  - `chmod 0744 myfile.txt`
    ```

ASCII

- 7 bits
- Not all printable
- Linux:
  - `hexdump -C`
  - `xxd`
  - `man ascii`

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

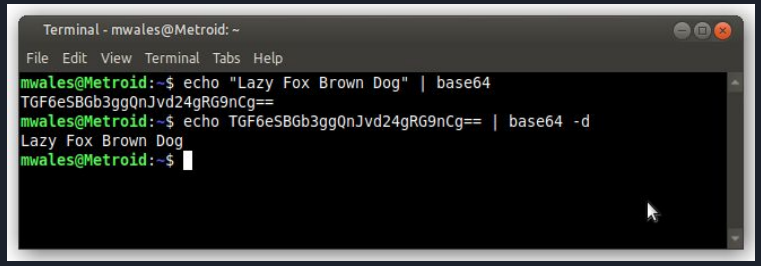
asciicharstable.com

Examining Binary Files

- **`cat`**-ing or viewing binary data in normal text editor usually not fun
- **`strings`**: show us just the printable sequences of characters
- **`file`** / **`binwalk`**: look for magic numbers / bytes in the files
- **`hexdump`** / **`xxd`**
- **`od`**: octal dump 🤮

```
mwales@Metroid:~/scratch/hidden_test$ hexdump -C test_file.txt
00000000  4e 6f 74 20 6d 75 63 68  20 74 6f 20 73 65 65 20  |Not much to see |
00000010  69 6e 20 68 65 72 65 2e  2e 2e 0a                 |in here....|
0000001b
mwales@Metroid:~/scratch/hidden_test$ hexdump test_file.txt
0000000 6f4e 2074 756d 6863 7420 206f 6573 2065
0000010 6e69 6820 7265 2e65 2e2e 000a
000001b
mwales@Metroid:~/scratch/hidden_test$ xxd test_file.txt
00000000: 4e6f 7420 6d75 6368 2074 6f20 7365 6520  Not much to see
00000010: 696e 2068 6572 652e 2e2e 0a              in here....
mwales@Metroid:~/scratch/hidden_test$ od test_file.txt
0000000 067516 020164 072555 064143 072040 020157 062563 020145
0000020 067151 064040 071145 027145 027056 000012
0000033
mwales@Metroid:~/scratch/hidden_test$
```

base64



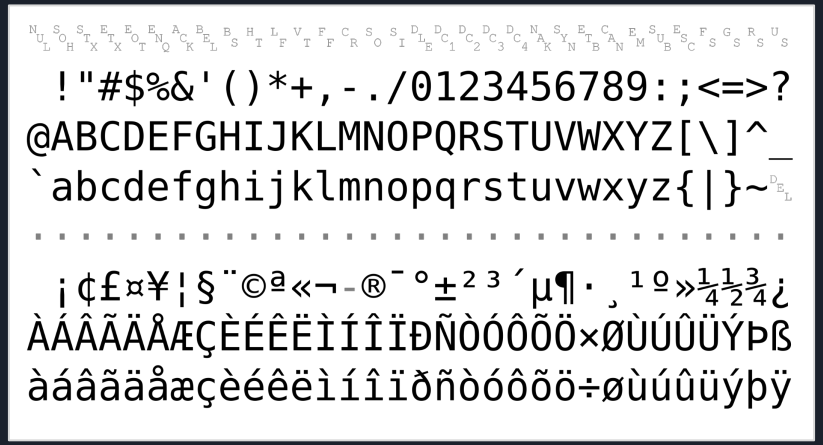
- A byte represents 0 - 255.
- Printable ASCII is 32 - 127 (94 characters)
- `base64` uses 64 characters (6 bits)
  - A-Z a-z
  - 0-9
  - + /
  - = (padding, it's always at the end)
- You can use base64 to encode binary data into printable text (3 binary bytes -> 4 `base64` characters)
- Uses
  - Email attachments
  - Dumping a binary file out via serial shell
  - Simple obfuscation
- `base64 -d` decodes base64 back into binary/ASCII

Latin-1

- What about the other 128 characters we could represent with 1 byte?
- What about other latin characters
- ISO/IEC 8859 aka Latin-1

```
>>> b"\xa5\xa6\xa7".decode("latin1")
'¥¦§'
```

- What about other languages
- 😠😠 WHAT ABOUT EMOJIs!!!! 😠😠



Unicode

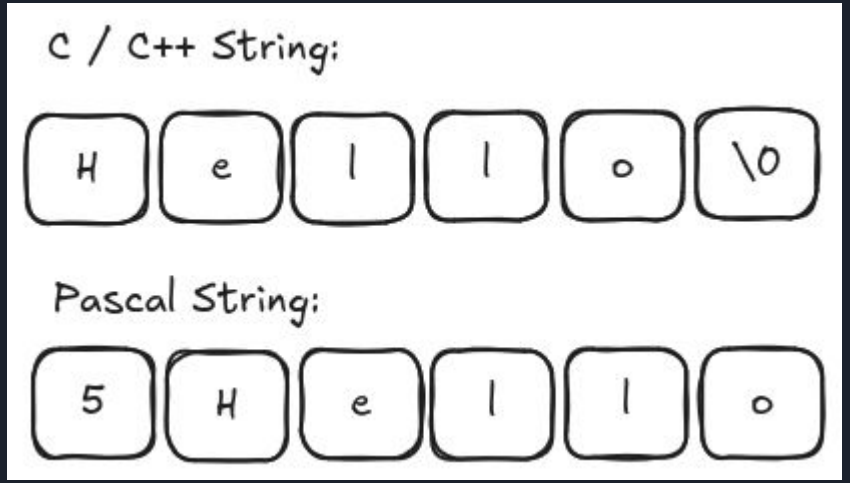
- Supports 1.1 million characters U+0000 to U+10FFFF (code points)
- 3,790 emoji so far 😍
- Encodings:
  - UTF-8
    - Compatible with ASCII 🔥
    - 8-bits for ASCII, more otherwise
    - Unix / Linux 🐧 / Web / HTML
  - UTF-16
    - 2 bytes per characters
    - Windows 🪟 🤦
-

```
>>> "😻".encode("utf-8")

b'\xf0\x9f\x98\xbb'
>>> "и".encode("utf-8")
b'\xd0\xb8'
```

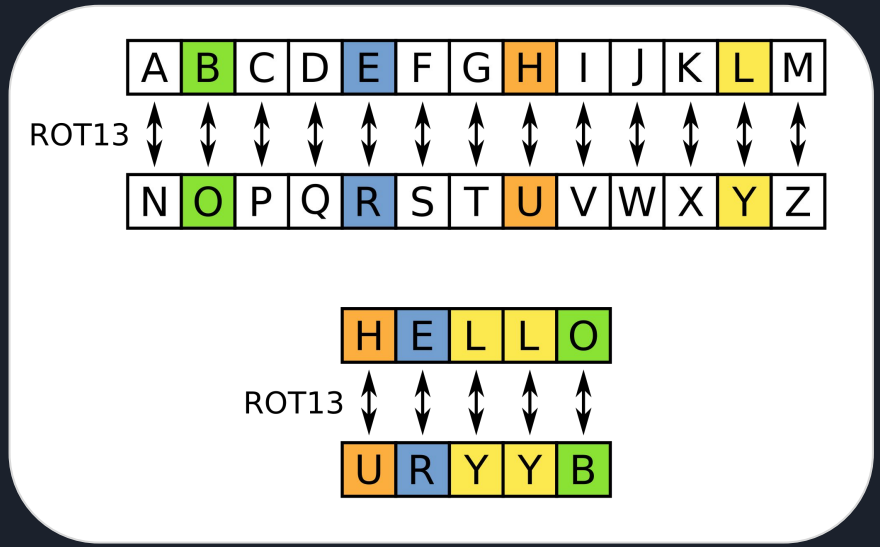
Strings

- C / C++
  - Null terminated
  - 0 index
- Pascal
  - Length prefix
  - 1 index



rot13

- Simple substitution cipher
- Caesar cipher (invented in Rome)
- Rotate alphabet 3 characters



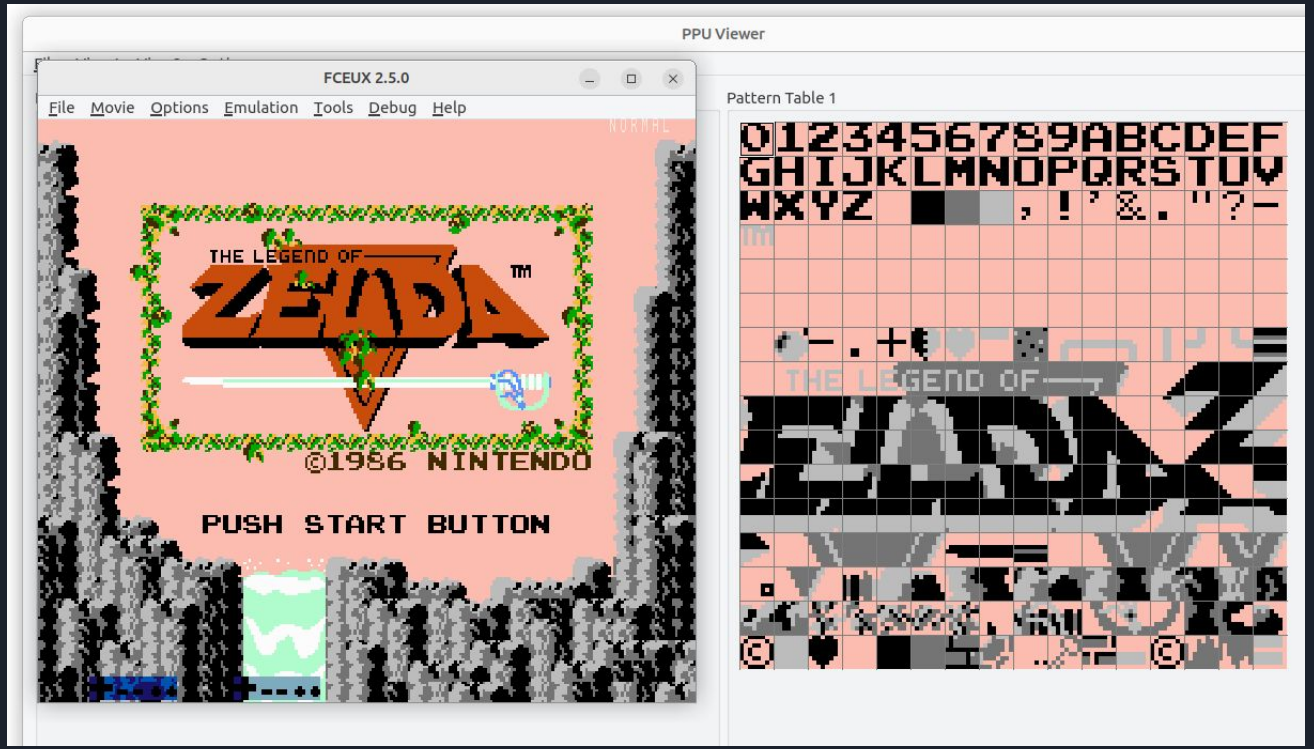
guvf vf qhzo

this is dumb

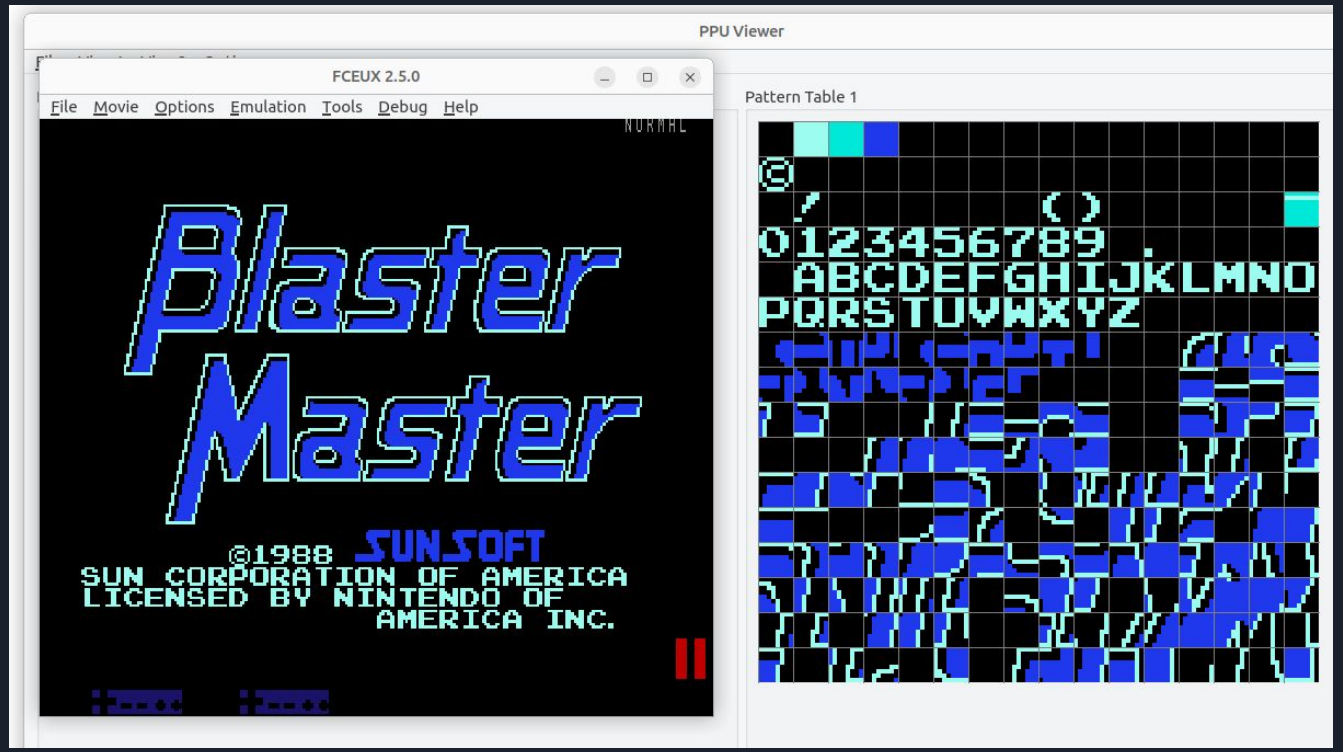
irk ones gnat

vex barf tang

NES Games


PPU Viewer
FCEUX 2.5.0
File Movie Options Emulation Tools Debug Help
THE LEGEND OF ZELDA
©1986 NINTENDO
PUSH START BUTTON
Pattern Table 1

NES Games


PPU Viewer
FCEUX 2.5.0
File Movie Options Emulation Tools Debug Help
NORMAL
Blaster Master
©1988 SUNSOFT
SUN CORPORATION OF AMERICA
LICENSED BY NINTENDO OF AMERICA INC.
Pattern Table 1

Attributions

- https://www.w3resource.com (binary)
- https://commons.wikimedia.org/wiki/File:32bit-Endianess.svg
- Benjamin D. Esham for ROT13 picture
- https://en.wikipedia.org/wiki/Word_(computer_architecture)
- https://en.wikipedia.org/wiki/ISO/IEC_8859-1#/media/File:Latin-1-infobox.svg
-