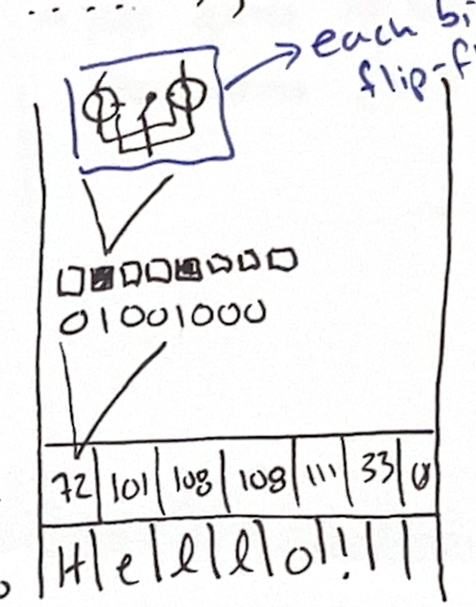
```c
#include <stdio.h>

int main() {
  char hello[] = "Hello!";
  char hello2[] = { 72, 101, 108, 108, 111, 33, 0 };
                    H   e   l   l   o   !  \0
  puts(hello);
  puts(hello2);
}
```

```
$ gcc hello.c -o hello
$ ./hello
Hello!
Hello!
```

creating an array of characters

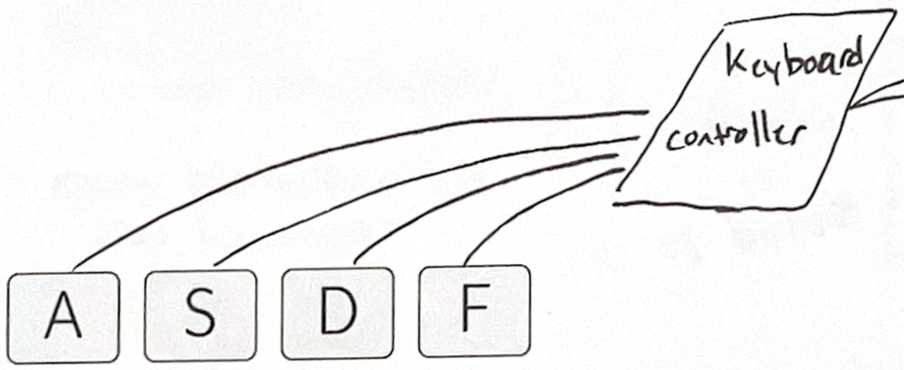
```
char s[] = "....";
char s2[] = {.....};
```



(computers store things w/ metal and electricity, not memory models!)

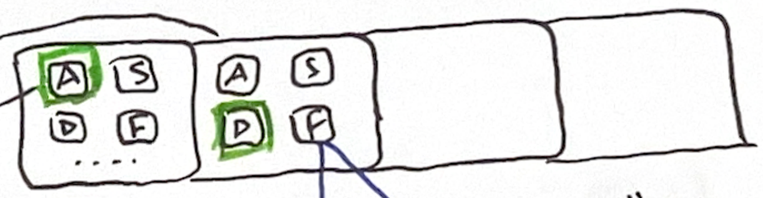
ASCII - most common byte ↔ char mapping

What happens when you press a key?



Inside my computer's memory

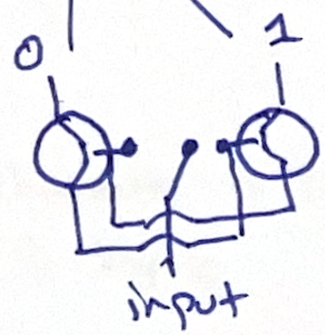
last 10 pressed keys



how many keys are there?

~100

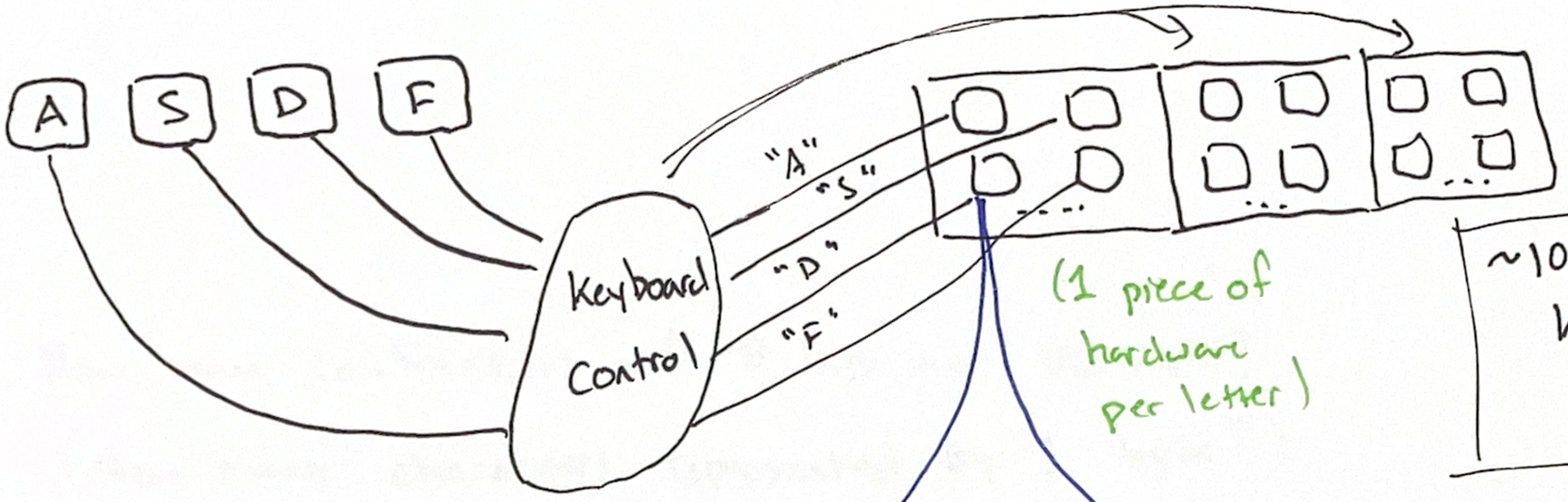
each box stores if particular key was pressed



"Flip Flop Gate"

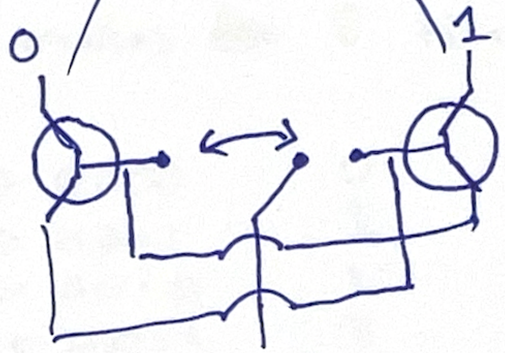
SRAM

- cheap (silicon)
- small
- fast (100's flips/sec)
- accurate



(1 piece of hardware per letter)

~100 gates to store a key press (a-z A-Z punc. 0-9)



input

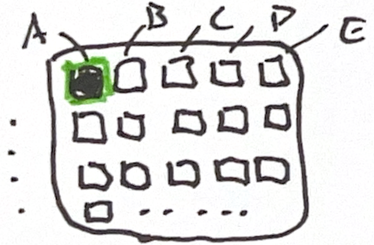
"flip flop" gate.
send a signal 0 or 1 on input and flip flop "sticks" to that side (requires power)

SRAM
- cheap (silicon)
- small
- fast (100M times/sec)
- accurate

(How) Can we do better than 100 gates per character / per key? By "better" I mean fewer flip-flops flip flops per character

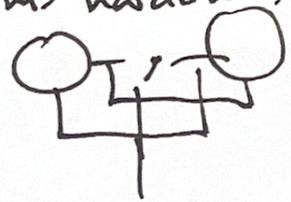
current picture:

each key/char



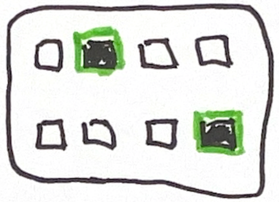
100 little boxes

each little box is this hardware:

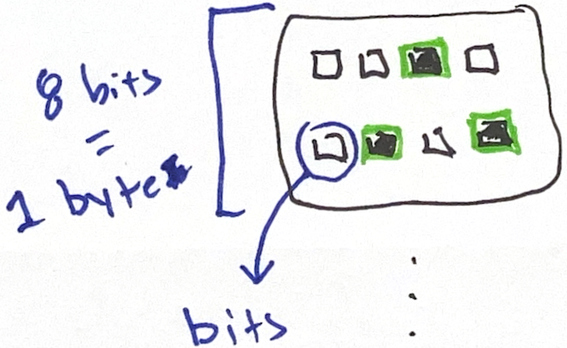


Proposal

each key/char



(this combo means 'A')



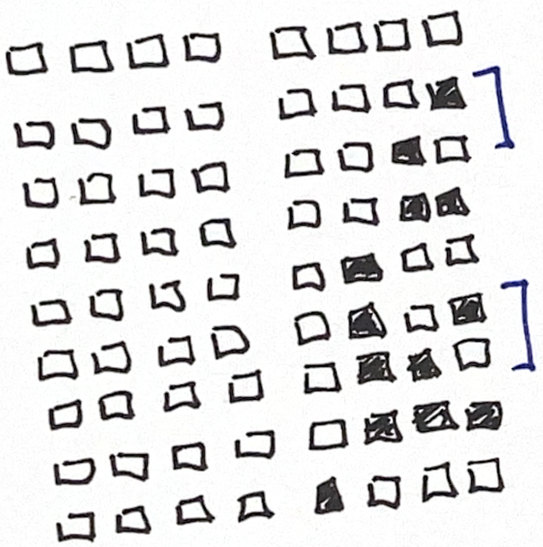
(this combo means '%')

one character is represented by 1 byte

How many combinations of 8 bits are possible?

(How many characters represented by 1 byte?)

Combinatorics mindset: 2 choices 8 times $= 2^8 = 256$



0000 0000	0
0000 0001	1
0000 0010	2
0000 0011	3
0000 0100	4
0000 0101	5
0000 0110	6
0000 0111	7
0000 1000	8
⋮	⋮
1111 1111	255

there are 256 rows!

what would the 01 ... for 255 be?

```c
#include <stdio.h>

int main() {
  char hello[] = "Hello!";
  char hellonum[] = { 72,        101,       108,       108,       111,       33,        0 };
  char hellobin[] = { 0b1001000, 0b1100101, 0b1101100, 0b1101100, 0b1101111, 0b100001, 0b0 };

  puts(hello);
  puts(hellonum);
  puts(hellobin);

  printf("%c %c %c\n", hello[0], hello[1], hello[2]);

  printf("%d %d %d\n", hello[0], hellonum[0], hellobin[0]);
  printf("%c %c %c\n", hello[0], hellonum[0], hellobin[0]);
}
```

0b _____ means writing a number (a single byte here) directly as bits

"Binary Literal"

%c : print a value as char

%d : print a value as decimal #

```
$ gcc hellobin.c -o hellobin
$ ./hellobin
Hello!
Hello!
Hello!
H e l
72 72 72
H H H
```

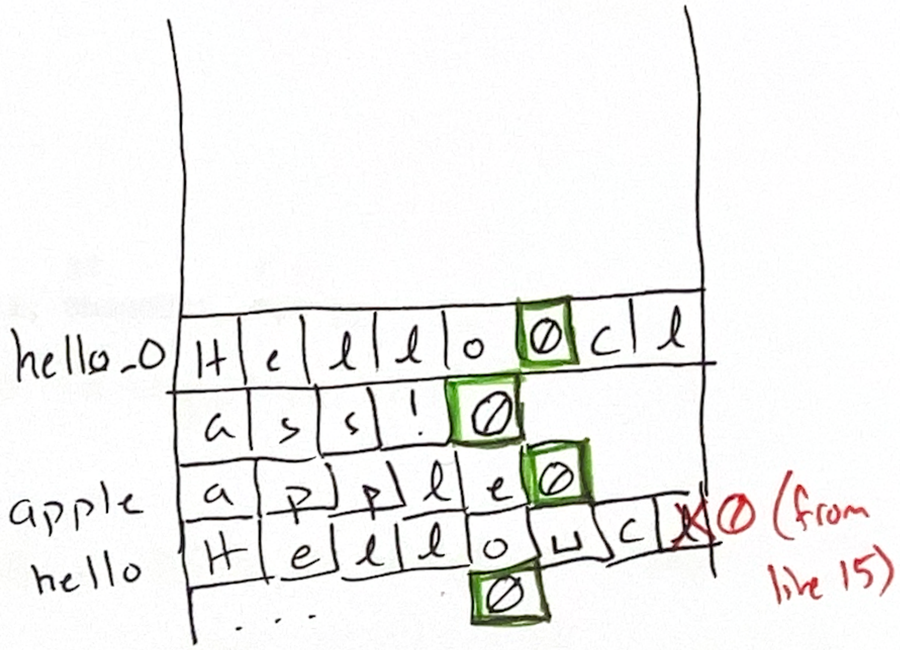
puts

Line 12

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
  char hello[] = "Hello class!";
  char apple[] = "apple";
  char hello_0[] = "Hello\0class!";

  printf("%ld\n", strlen(hello));
  printf("%ld\n", strlen(apple));

  printf("%s, length: %ld\n", hello_0, strlen(hello_0));

  hello[7] = 0;

  printf("%s, length: %ld\n", hello, strlen(hello));

  return 0;
}
```

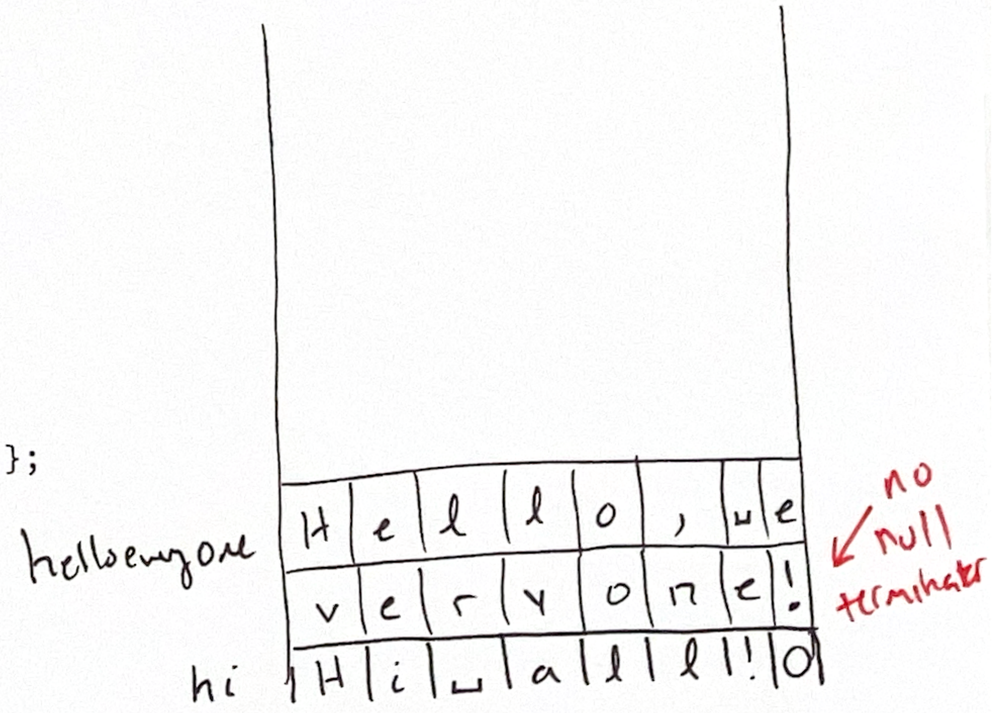
these are null terminated because they are string literals (true for all string literals)

Hello 5




```
$ gcc strlen.c -o strlen
$ ./strlen
12
5
Hello, length 5
Hello c, length: 7
```

```c
#include <stdio.h>

int main() {
    char hi[] = "Hi all!";
    char helloeveryone[] = { 'H', 'e', 'l', 'l', 'o', ',', ' ',
                             'e', 'v', 'e', 'r', 'y', 'o', 'n', 'e', '!' };

    puts(hi);
    puts(helloeveryone);
}
```




```
$ gcc adjacent.c -o adjacent
$ ./adjacent
```

Hi all!
Hello, everyone!Hi all!