load accumulator 16.1

```
code = bytearray([
  0xa9, 0xff,          # lda #$ff
  0x8d, 0x02, 0x60,    # sta $6002 -> Output Register "B"
                       ↳ store accumulator
  0xa9, 0x55,          # lda #$55
  0x8d, 0x00, 0x60,    # sta $6000 -> Output Port "B"

  0xa9, 0xaa,          # lda #$aa
  0x8d, 0x00, 0x60,    # sta $6000

  0x4c, 0x05, 0x80,    # jmp $8005
  ])

rom = code + bytearray([0xea] * (32768 - len(code)))

rom[0x7ffc] = 0x00
rom[0x7ffd] = 0x80

with open("rom.bin", "wb") as out_file:
  out_file.write(rom)
```

→ bootstrap

little endian ↳ 0x00, 0x60 = 6000

Ce bootstrap prend les vecteurs 0x7ffc et 0x7ffd
↳ va à l'adresse indiquée par les vecteurs
0x4c : jump à l'adresse 6005, il faut compter toutes les 0x avant.

bootstrap sequence entière :

eb60
eb60
ffff
eb60
01f7
01f6
01f5

{ 0x55 = 01010101
  0xaa = 10101010
↳ va faire clignoter les leds

SRAM

6 Transistors

⇓

Le code est stocké dans les ~~capacitors~~ Transistor

\+

Pas besoin de refresh

Rapide

—

Plus large

Plus cher

DRAM

1 Transistor + 1 Capacitor

⇓

Courant stocké dans le capacitor

\+

Plus dense

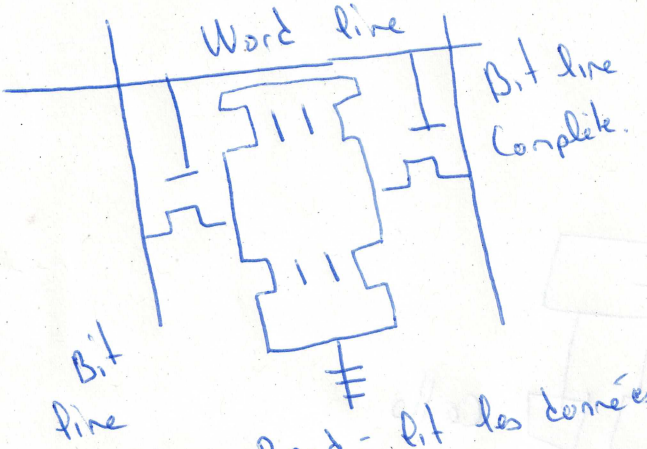
Moins cher

—

Refresh

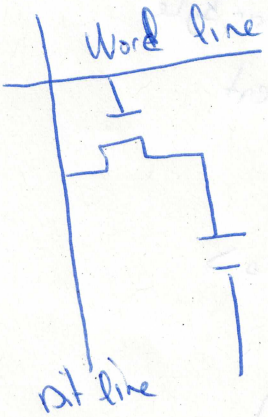
moins Rapide

le courant va en sortir des Capacitor



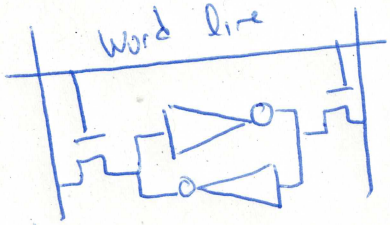
Bit line : Read = lit les données
Write = donne les données

Word line : sélectionne la cellule qui doit être écrite



bit/Word line : fonctionnent pareil

= -> Capacitor



=> schéma simplifié

bit line

1 = transistor

▷○ = Inverter

EE PROM

1. Donne le protocole
2. ~~Cherche~~ Cherche l'adresse et lit
3. Rend les données et write

simplifié

↑ Isole les données

EEPROM

Adresse par byte

plus lent

< 1 MB

$1 \cdot 10^6$ Rewrites >

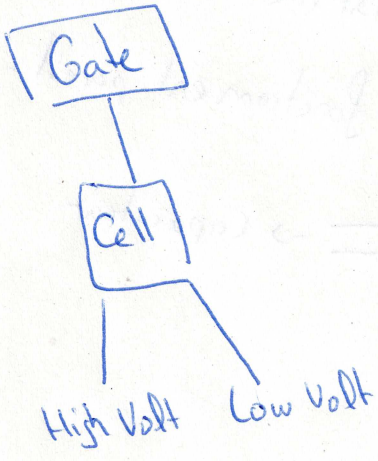
Flash

par Block

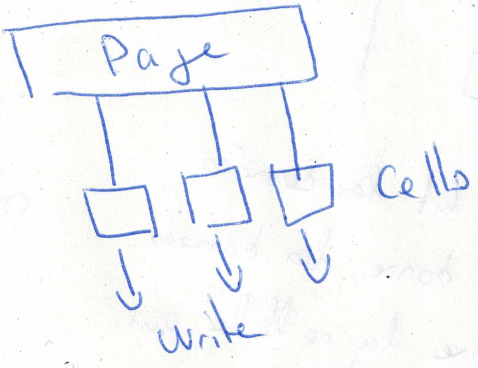
plus rapide

Terabyte

$\sim 1 \cdot 10^4$



Memory Cell



Memory Block