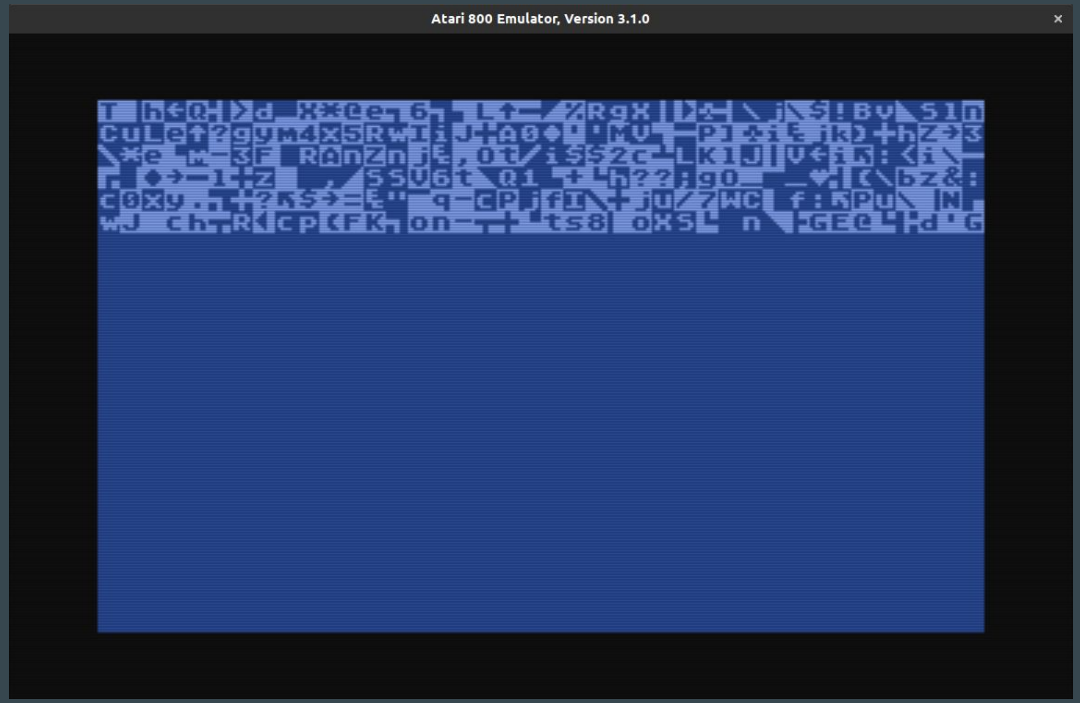
Atari 8 bit
Accediendo más allá de los 8 bits
•••

BitaBit S01E04
bitabit.catrinlabs.cl

Objetivos

- Acceder a más de 256 bytes de memoria en un loop
- Usar variables en memoria
- Ver un programa assembler como bloques

En el episodio anterior...




```
        org $2000

start   ldx #0

loop    lda 53770
        sta 40000,x
        inx
        cpx #240
        bne loop

        jmp start
```

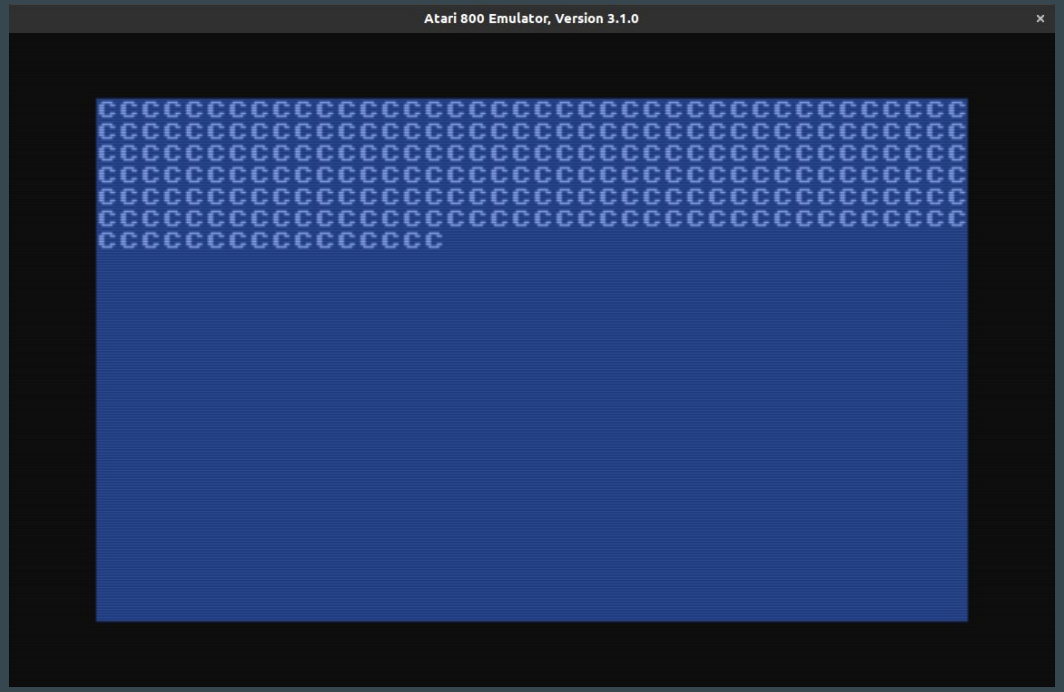
Usando el índice X hasta el máximo

X es un registro de 8 bit (0-255)

```
        ldx #0
        lda #35
loop    sta 40000,x
        inx
        bne loop
```

posición	valor
39999	??
40000	35
40001	35
40002	35
..	35
40253	35
40254	35
40255	35
40256	??

Prueba de loop sobre X completo



Acceso a la memoria hasta ahora

```
lda 40000
sta 40000
```

Acceso directo

```
lda 40000,x
sta 40000,x
```

Acceso indexado

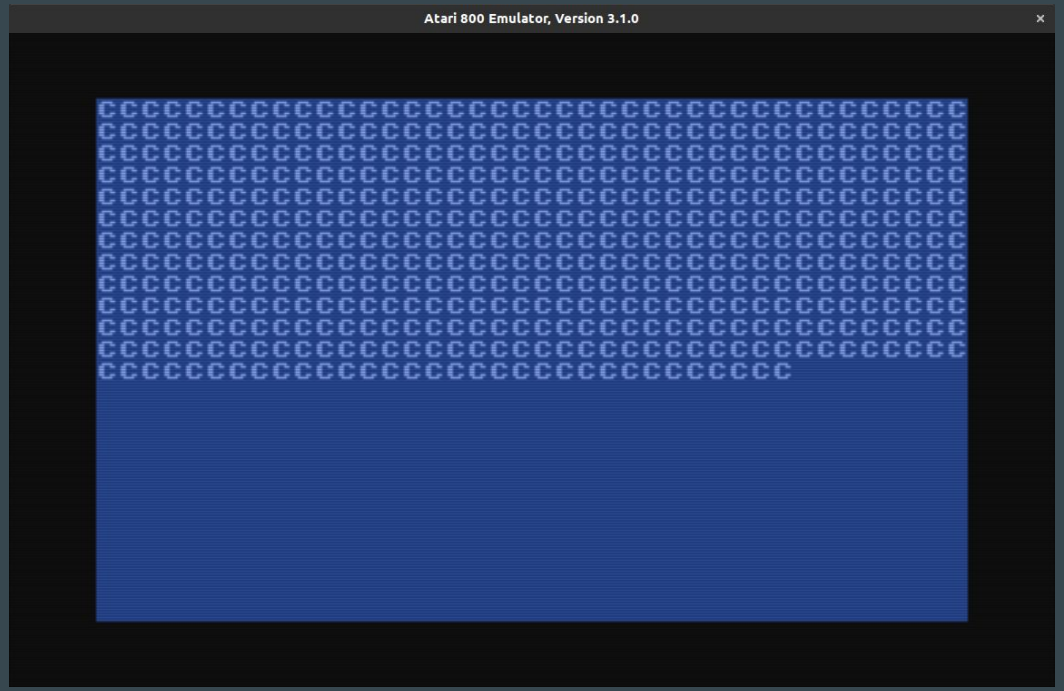
Una forma inocente de superar el límite

```
        ldx #0
        lda #35

loop1   sta 40000,x
        inx
        bne loop1

loop2   sta 40256,x
        inx
        bne loop2
```

Resulta pero no es lo ideal



Necesitamos que la dirección sea variable

```
        ldx #0
        lda #35

loop1   sta 40000,x
        inx
        bne loop1

loop2   sta 40256,x
        inx
        bne loop2
```

Necesitamos una variable que apunte a una dirección de memoria

Necesitamos un puntero!

```
sta (PTR),x
```

El 6502 lo permite pero sólo con el índice Y

```
sta (PTR),y
```

Direccionamiento indirecto

- Se usa una variable de 16 bit en memoria
- Sólo se puede almacenar en los primeros 256 bytes (página cero)
- 256 bytes de RAM => 1 página

```
ldy #0
lda #35
sta (202),y
```

posición	valor
201	??
202	64
203	156
204	??

```
64 + 256 * 156 = 40000
```

Nueva versión usando punteros

```
        lda #64
        sta 202
        lda #156
        sta 203

        ldy #0
        lda #35

loop1   sta (202),y  ; 64 + 256 * 156 = 40000
        iny
        bne loop1

        inc 203      ; addr 203 => 157

loop2   sta (202),y  ; 64 + 256 * 157 = 40256
        iny
        bne loop2
```

posición	valor
201	??
202	64
203	156
204	??

Versión más flexible

```
        lda #<40000
        sta 202
        lda #>40000
        sta 203

        ldy #0
        lda #35

loop    sta (202),y  ; 64 + 256 * 156 = 40000
        iny
        bne loop

        inc 203      ; addr 203 => 157
        ldx 203
        cpx #158     ; página final
        bne loop
```

posición	valor
201	??
202	64
203	156
204	??

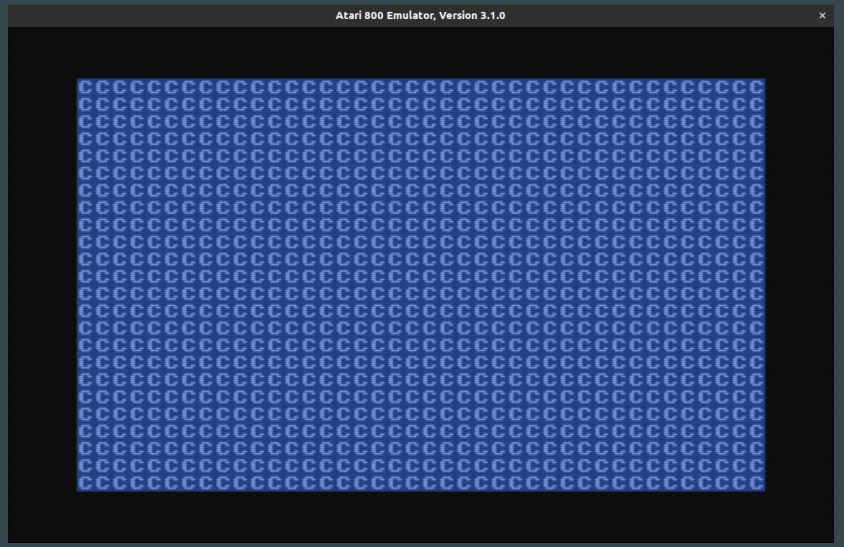
Ahora podemos cubrir la pantalla completa

- 40 columnas * 24 filas = 960 caracteres
- Dirección final 40960 => **Página 160**

```
ldx 203
cpx #158
```

→

```
ldx 203
cpx #160
```



Versión más *decente*

```
ini     = 40000
ptr     = 202
char_c = 35
```

```
        lda #<ini
        sta ptr
        lda #>ini
        sta ptr+1

        ldy #0
        lda #char_c

loop    sta (ptr),y
        iny
        bne loop

        inc ptr+1
        ldx ptr+1
        cpx #160
        bne loop
```

Visto como bloques

Declaraciones

```
ini    = 40000
ptr    = 202
char_c = 35
```

Inicialización

```
        lda #<ini
        sta ptr
        lda #>ini
        sta ptr+1

        ldy #0
        lda #char_c
```

Loop

```
loop    sta (ptr),y
        iny
        bne loop

        inc ptr+1
        ldx ptr+1
        cpx #160
        bne loop
```

La parte fea a mejorar

```
inc ptr+1
ldx ptr+1
cpx #160
```

¡Gracias!

Siguiente Episodio:
Usar direcciones y largos variables