

Atari 8 bit

Escribiendo en la pantalla con Assembler

...

Bitabit S01E03
bitabit.catrinlabs.cl

Objetivos

- Hacer nuestro primer programa en Assembler 6502
- Expandir el programa usando loops
- Comparar con BASIC

Prerequisitos

- Entender sistema numérico Hexadecimal
- Bitabit Extra : Sistemas numéricos

Assembler y código de máquina

- Cada procesador tiene su propio código de máquina (números)
- El código assembler es la versión humana (texto)
- Con un programa ensamblador convertimos assembler en código de máquina

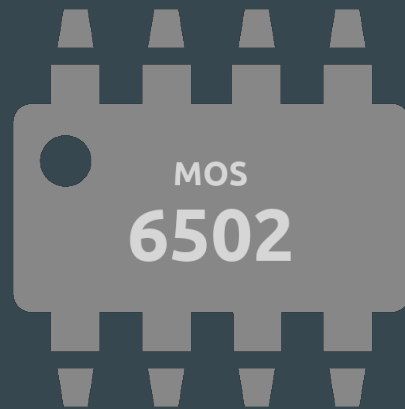
Código Assembler

```
INC X  
CALL PRINT  
INC Y  
CALL CLEAR
```

Ensamblador

Código de máquina

```
0D AA 5F 3D  
86 88 3A 20 FF
```



Algunos procesadores conocidos

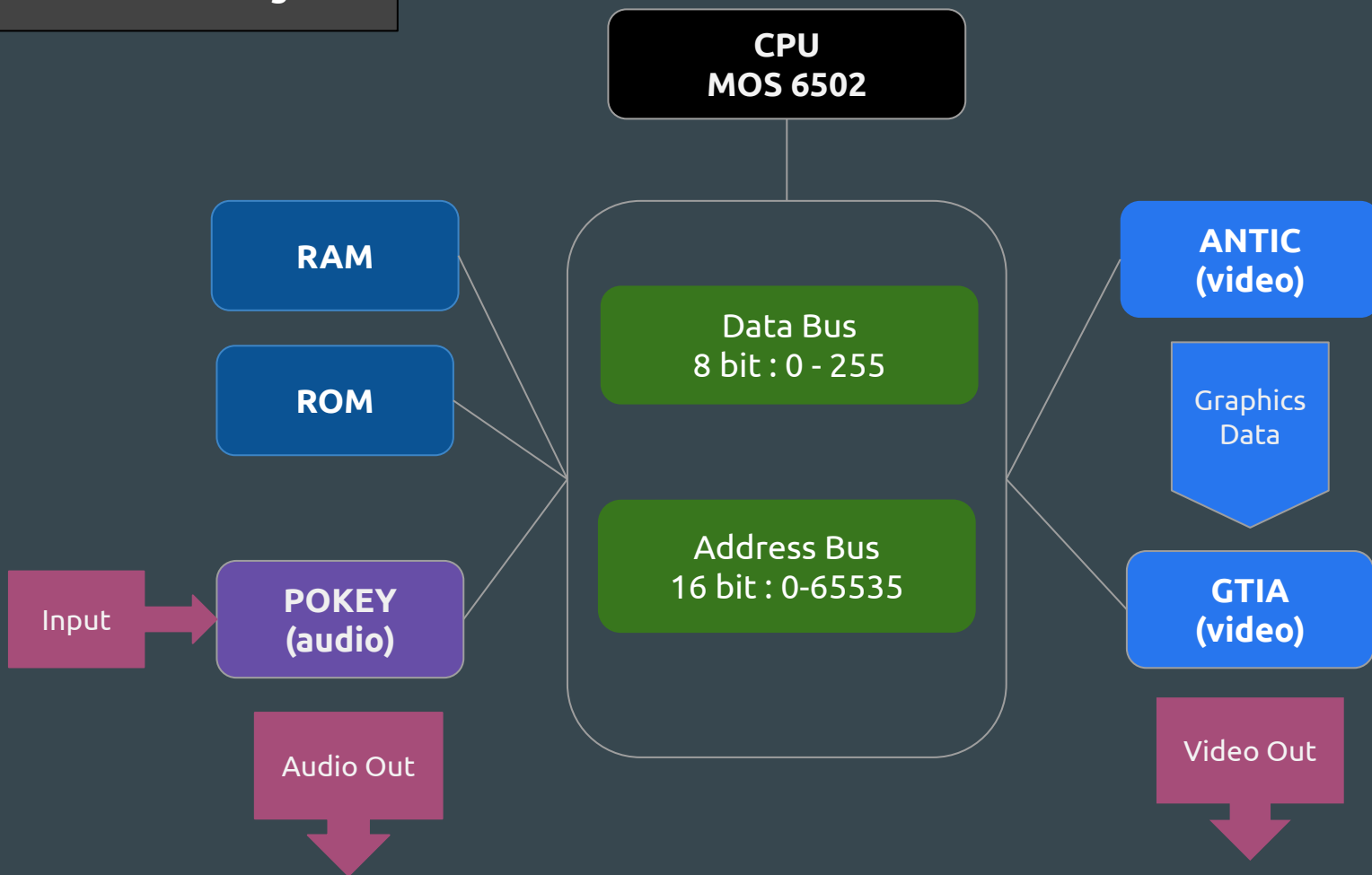
- De 8 bit
 - MOS **6502** (Atari, NES, Commodore, Apple II)
 - Zilog Z80 (ZX Spectrum, MSX)
- De 16 bit
 - Intel 8086, 80286 (PC)
 - Motorola 68000 (Commodore Amiga, Sega Genesis)
- De 32 bit
 - Intel 80386, 80486, Pentium (PC)
 - Algunos ARM (teléfonos, tablets, televisores, etc)
- De 64 bit
 - Intel Atom, Core 2, Core i3, i5, i7 (PC)
 - Algunos ARM

MOS 6502

- Procesador de 8 bits, bus de direcciones de 16 bits
- Maneja datos entre 0-255 y direcciones de 0-65535 (64 Kilobytes)
- Primer procesador ultra económico (USD\$25 vs USD\$300)
- Impulsó la computación y consolas de videojuegos hogareñas
- Aun se fabrica en versiones modernas
- Es simple de programar



ATARI 800 Block Diagram



Flujo de trabajo

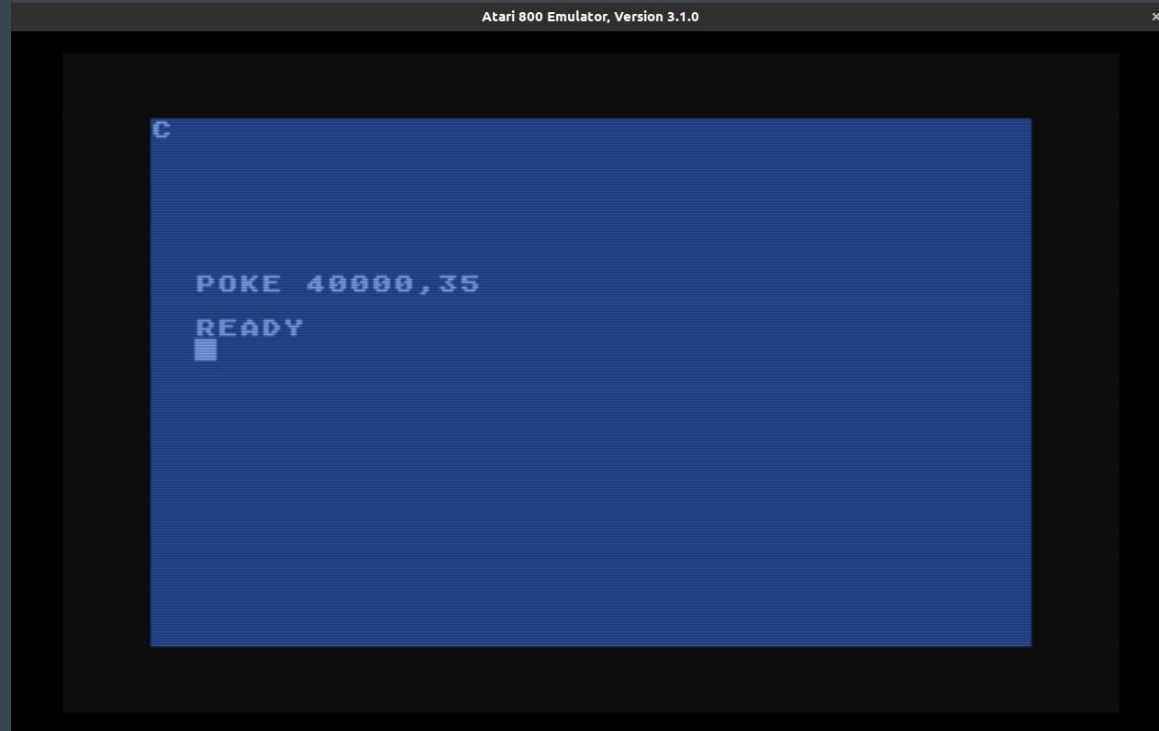
No usaremos un ensamblador en Atari, sino en un PC.

Los pasos son

1. Instalar un ensamblador (mads)
2. Editar el código
3. Compilar (generar ejecutable)
4. Probar con el emulador
5. Volver al paso 2

Ejemplo 1: Escribir un caracter con ASM

En el episodio anterior...



Versión en Assembler “teórico”

```
store 40000, 35
```

El registro "A": Acumulador

- Un registro es como una variable interna del procesador
- Es el registro principal del 6502
- Es de 8 bits
- Permite realizar
 - Operaciones aritméticas como suma y resta
 - Operaciones lógicas como and, or, xor
 - Otras operaciones a nivel de bits
- También permite leer y escribir datos en memoria

Versión en Assembler 6502

- LdA = Load Accumulator
- StA = Store Accumulator

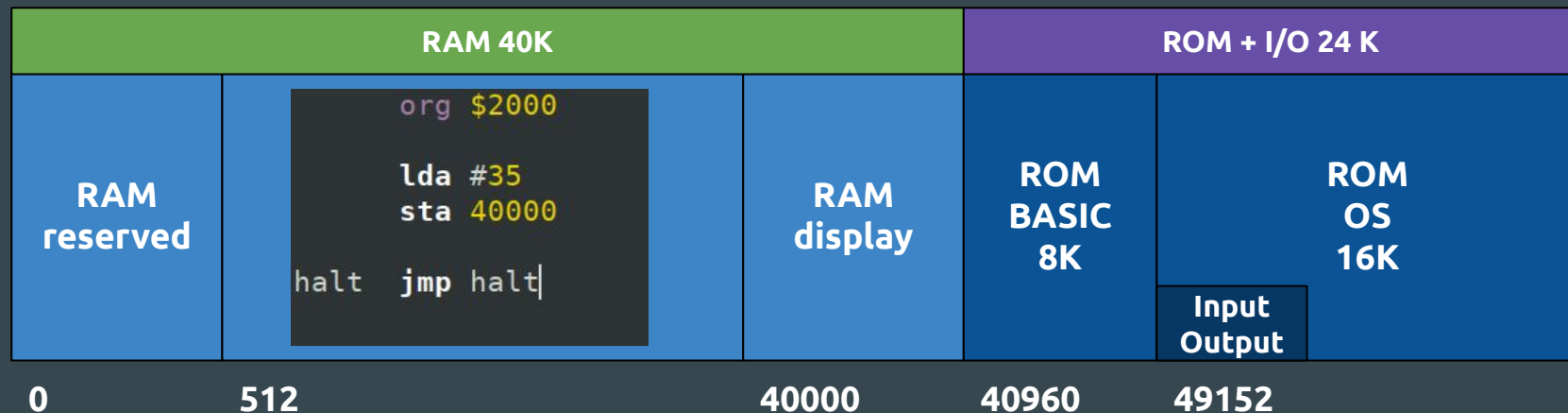
~~store 40000, 35~~

lda #35
sta 40000

Versión ejecutable completa

```
org $2000  
  
lda #35  
sta 40000  
  
halt jmp halt|
```

ATARI 800 Memory Map



Demo : Compilar y ejecutar

Código de máquina generado

```
fcatrin@asusmini:~/git/bitabit/code/S01E03/asm$ atari800 01_single.obx
Using Atari800 config file: /home/fcatrin/.atari800.cfg
Created by Atari 800 Emulator, Version 3.1.0

joystick 0 not found
joystick 1 not found
Video Mode: 768x480x32 windowed, pixel format: BGR16
OpenGL initialized successfully. Version: 3.0 Mesa 19.2.8
OpenGL Pixel Buffer Objects available.
   0    0 A=23 X=FF Y=FF S=FB P=---*----- PC=2005: 4C 05 20 JMP $2005
> m 2000
2000: A9 23 8D 40 9C 4C 05 20 00 00 00 00 00 00 00 00 .#.@.L. ....
2010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
2020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
2030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
2040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
2050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
2060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Código de máquina desensamblado

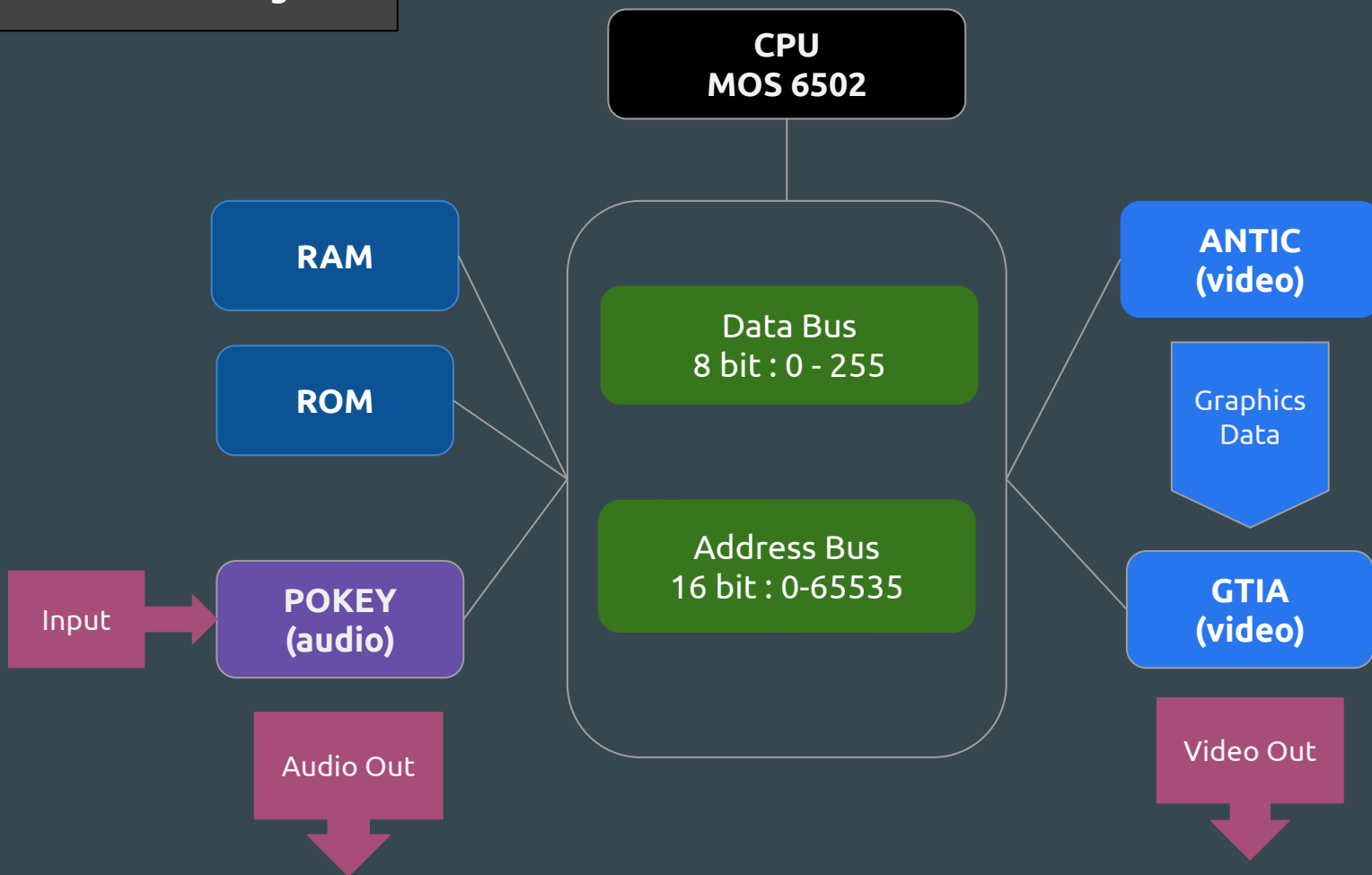
```
> d 2000
2000: A9 23      LDA #$23
2002: 8D 40 9C   STA $9C40
2005: 4C 05 20   JMP $2005
2008: 00        BRK
2009: 00        BRK
200A: 00        BRK
200B: 00        BRK
200C: 00        BRK
200D: 00        BRK
200E: 00        BRK
200F: 00        BRK
2010: 00        BRK
2011: 00        BRK
```

```
org $2000

lda #35
sta 40000

halt jmp halt|
```

ATARI 800 Block Diagram



Tips Hexadecimales

Conversión decimal -> hexadecimal rápida

- Dividir por 16
- Resultado = Dígito más significativo (H / High)
- Resto = Dígito menos significativo (L / Low)

Decimal 35

Dividido en 16 = 2

Resto = 3

Hexadecimal : 23

Conversión hexadecimal -> decimal rápida

- Multiplicar dígito más significativo por 16
- Sumar el dígito menos significativo
- Decimal = $H * 16 + L$

Hexadecimal : 23

$$\text{Decimal} = 2 * 16 + 3 = 35$$

Tips : Manejo de 16 bits

Direcciones de 16 bits

- Se representan por dos bytes (8 + 8 bits)
- En memoria primero el byte menos significativo
- En notación primero el byte más significativo

```
> d 2000
2000: A9 23      LDA #$23
2002: 8D 40 9C   STA $9C40
2005: 4C 05 20   JMP $2005
2008: 00        BRK
2009: 00        BRK
200A: 00        BRK
200B: 00        BRK
200C: 00        BRK
200D: 00        BRK
200E: 00        BRK
200F: 00        BRK
2010: 00        BRK
2011: 00        BRK
```


Conversión de dirección de 16 bits a 2 bytes

- Se divide por 256
- Resultado = más significativo (**H** / High)
- Resto = menos significativo (**L** / Low)

$40000 / 256 = 156$

Resto = 64

Bytes 64 156

Hexadecimal: 40 9C

```
> d 2000
2000: A9 23      LDA #$23
2002: 8D 40 9C   STA $9C40
2005: 4C 05 20   JMP $2005
2008: 00                BRK
2009: 00                BRK
200A: 00                BRK
200B: 00                BRK
200C: 00                BRK
200D: 00                BRK
200E: 00                BRK
200F: 00                BRK
2010: 00                BRK
2011: 00                BRK
```

Conversión de 2 bytes a dirección de 16 bits

- Byte más significativo se multiplica por 256
- Se suma byte menos significativo
- $ADDR = H * 256 + L$

Hexadecimal: 40 9C

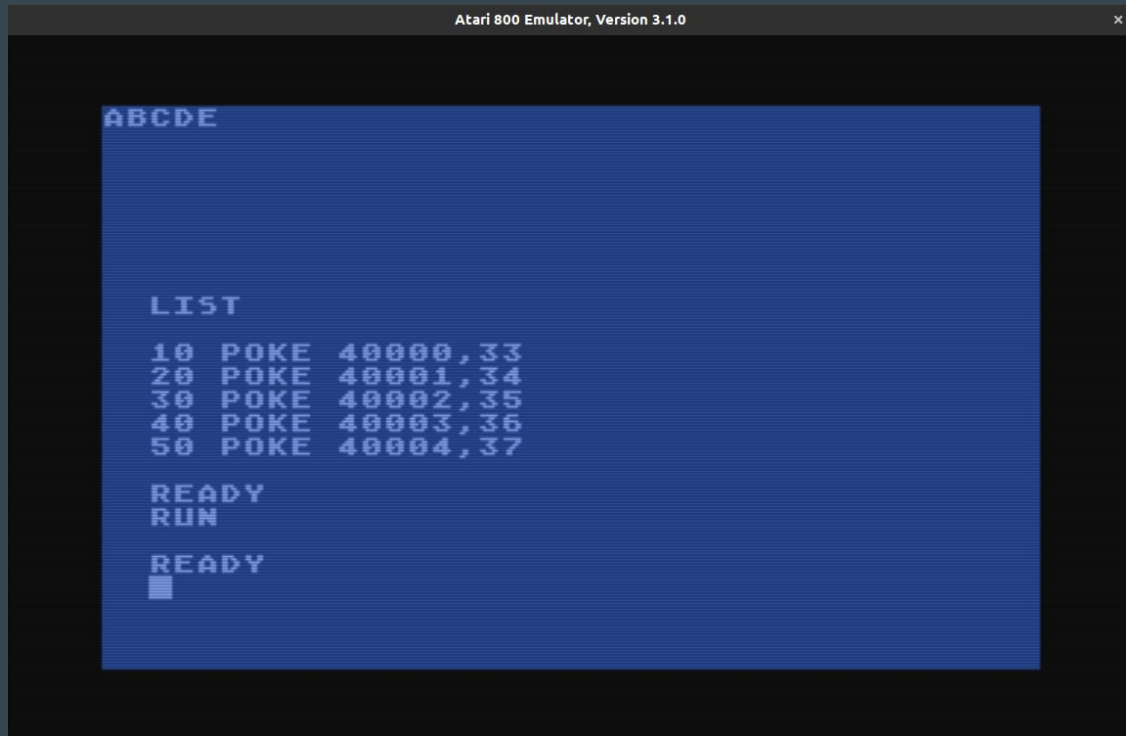
Bytes 64 156

$$156 * 256 + 64 = 40000$$

```
> d 2000
2000: A9 23      LDA #$23
2002: 8D 40 9C   STA $9C40
2005: 4C 05 20   JMP $2005
2008: 00                BRK
2009: 00                BRK
200A: 00                BRK
200B: 00                BRK
200C: 00                BRK
200D: 00                BRK
200E: 00                BRK
200F: 00                BRK
2010: 00                BRK
2011: 00                BRK
```

Ejemplo 2: Imprimir varios caracteres

Versión en BASIC



The image shows a screenshot of an Atari 800 Emulator window. The window title bar reads "Atari 800 Emulator, Version 3.1.0" and has a close button (X) on the right. The main display area is black with a blue rectangular window containing BASIC code. The code is as follows:

```
ABCDE

LIST

10 POKE 40000,33
20 POKE 40001,34
30 POKE 40002,35
40 POKE 40003,36
50 POKE 40004,37

READY
RUN

READY
█
```

Versión Assembler

Basic:

```
10 poke 40000, 33
20 poke 40000, 34
30 poke 40000, 35
40 poke 40000, 36
50 poke 40000, 37
```

```
org $2000

lda #33
sta 40000

lda #34
sta 40001

lda #35
sta 40002

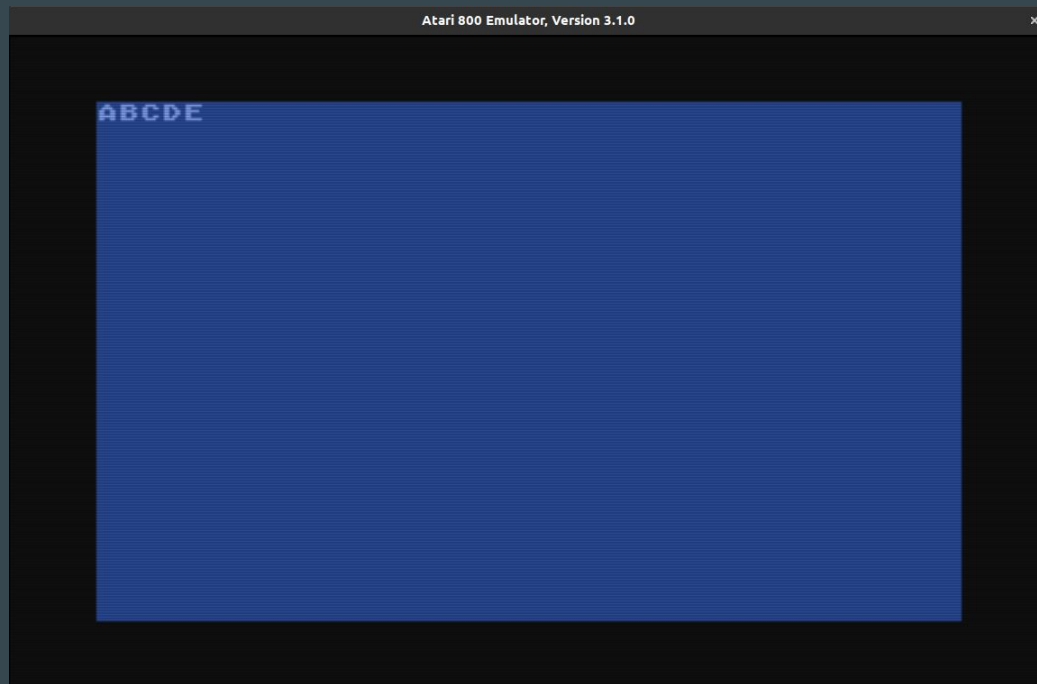
lda #36
sta 40003

lda #37
sta 40004

halt jmp halt
```

Demo: Compilar y ejecutar

Resultado en pantalla



Código de máquina en memoria

```
> m 2000
2000: A9 21 8D 40 9C A9 22 8D 41 9C A9 23 8D 42 9C A9  .!.@..".A..#.B..
2010: 24 8D 43 9C A9 25 8D 44 9C 4C 19 20 00 00 00 00 $.C..%.D.L. ....
2020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
2030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
2040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
2050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```


Código de máquina desensamblado

```
> d 2000
2000: A9 21      LDA #$21
2002: 8D 40 9C   STA $9C40
2005: A9 22      LDA #$22
2007: 8D 41 9C   STA $9C41
200A: A9 23      LDA #$23
200C: 8D 42 9C   STA $9C42
200F: A9 24      LDA #$24
2011: 8D 43 9C   STA $9C43
2014: A9 25      LDA #$25
2016: 8D 44 9C   STA $9C44
2019: 4C 19 20   JMP $2019
201C: 00        BRK
201D: 00        BRK
201E: 00        BRK
201F: 00        BRK
2020: 00        BRK
2021: 00        BRK
2022: 00        BRK
2023: 00        BRK
2024: 00        BRK
```

```
org $2000
```

```
lda #33
sta 40000
```

```
lda #34
sta 40001
```

```
lda #35
sta 40002
```

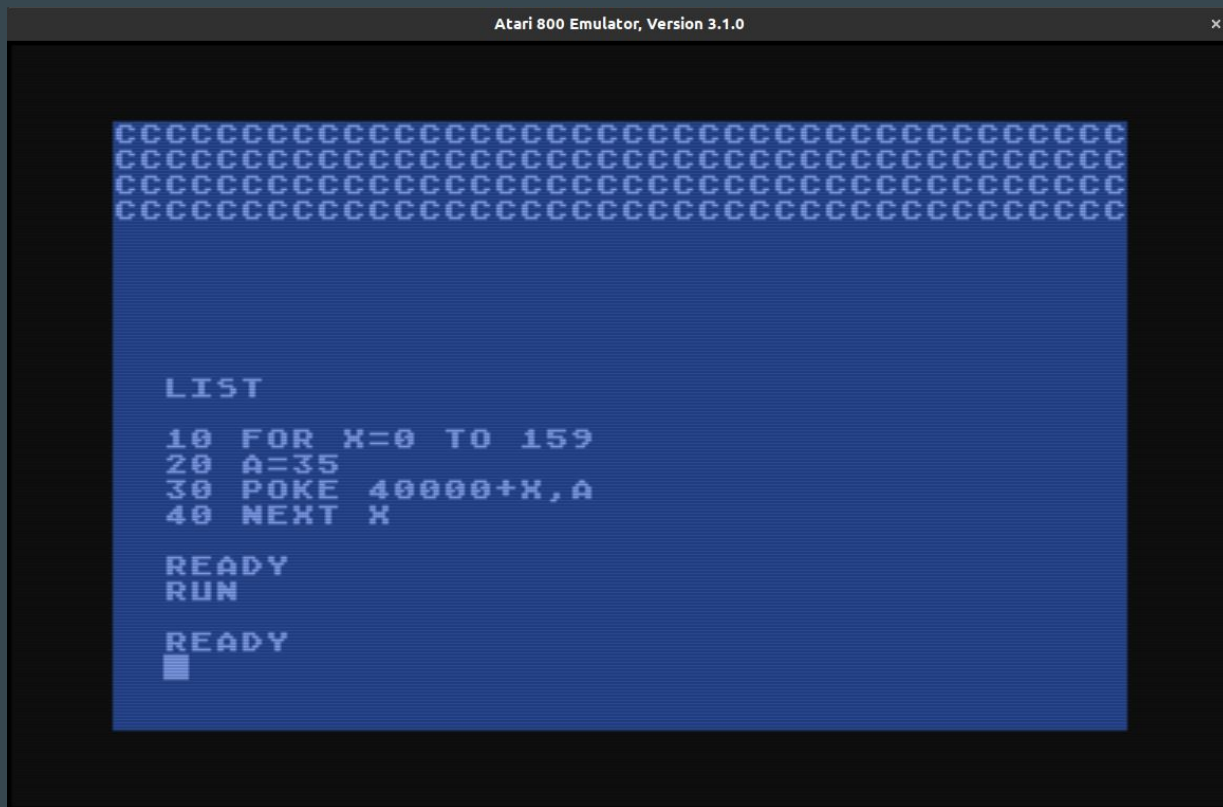
```
lda #36
sta 40003
```

```
lda #37
sta 40004
```

```
halt jmp halt
```

Ejemplo 3 : Usar un ciclo para repetir

Version BASIC



En assembler teórico

```
lda #35  
sta 40000+x
```

Registros índices X e Y

- Son registros auxiliares de 8 bits
- Se usan para
 - Contar
 - Acceder elementos de un array

```
lda #35  
sta 40000,x
```

Si $x = 3$

addr		valor
-----+-----		
40000		?
40001		?
40002		?
40003		35
40004		?
40006		?

Versión assembler de ciclo completo

Basic:

```
10 for x = 0 to 159
20 a = 35
30 poke 40000 + x, a
40 next x
```

Assembler:

```
ldx #0
lda #35
next sta 40000, x
inx
cpx #160
bne next
```

Código real ejecutable

```
org $2000  
  
ldx #0  
lda #35  
loop sta 40000,x  
inx  
cpx #160  
bne loop  
  
halt jmp halt
```

Demo: compilar y ejecutar

Código de máquina

```
> m 2000
2000: A2 00 A9 23 9D 40 9C E8 E0 A0 D0 F8 4C 0C 20 00
2010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
2020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
2030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
2040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
> d 2000
2000: A2 00      LDX #$00
2002: A9 23      LDA #$23
2004: 9D 40 9C   STA $9C40,X
2007: E8         INX
2008: E0 A0      CPX #$A0
200A: D0 F8      BNE $2004
200C: 4C 0C 20   JMP $200C
200F: 00         BRK
2010: 00         BRK
2011: 00         BRK
2012: 00         BRK
2013: 00         BRK
```

```
org $2000

ldx #0
lda #35
loop sta 40000,x
inx
cpx #160
bne loop

halt jmp halt
```

Ejemplo 4: Imprimir muchos caracteres

Versión BASIC

Atari 800 Emulator, Version 3.1.0

```
& 8H H\T X t 36 WI-G= \sb M>n1e F
Gy12 p+Nu/B !aN4xi: teLSuyc"7F> L/i#K118L
8M/MH.OqI> K NqS*!bqK s- %Q) A U6! t fT - 7
K/Mw jN5q6HFsc ;fI <ML. %$ JSx q@> aP5, L
f/00 .U Q t5&Ir <1A; A+k$> 3TZ05 L wa- > L
_8G K k A04v0 W je-se?KdWU/dHm c5 E5 : < 8<
```

LIST

```
10 FOR X=0 TO 239
20 A=PEEK(53770)
30 POKE 40000+X,A
40 NEXT X
50 GOTO 10
```

READY
RUN
█

Versión assembler de ciclo completo

Basic:

```
10 for x = 0 to 239
20 a = peek(53770)
30 poke 40000 + x, a
40 next x
50 goto 10
```

Assembler:

```
start    ldx #0
loop     lda 53770
         sta 40000, x
         inx
         cpx #240
         bne loop
         jmp start
```

Código ejecutable real

```
                org $2000  
  
start    ldx #0  
  
loop     lda 53770  
         sta 40000,x  
         inx  
         cpx #240  
         bne loop  
  
         jmp start
```

Demo: Compilar y ejecutar

Código de máquina

```
> m 2000
2000: A2 00 AD 0A D2 9D 40 9C E8 E0 F0 D0 F5 4C 00 20
2010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
> d 2000
2000: A2 00      LDX #$00
2002: AD 0A D2   LDA $D20A ;RANDOM
2005: 9D 40 9C   STA $9C40,X
2008: E8        INX
2009: E0 F0      CPX #$F0
200B: D0 F5      BNE $2002
200D: 4C 00 20   JMP $2000
2010: 00        BRK
2011: 00        BRK
2012: 00        BRK
2013: 00        BRK
```

```
org $2000

start  ldx #0

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

       jmp start
```

Gracias!

Siguiente Episodio:
Superar la barrera de los 8 bits