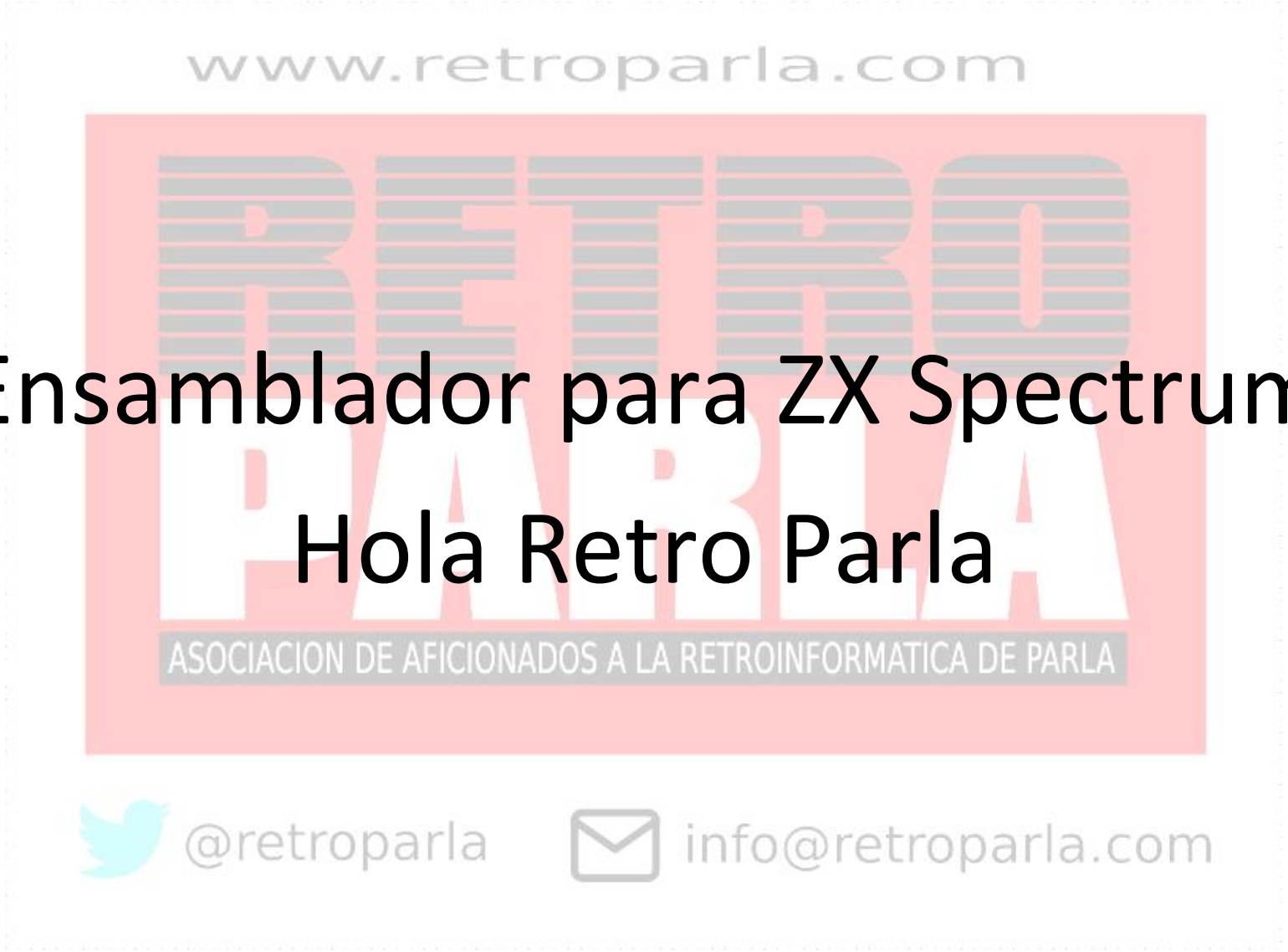

Ensamblador para ZX Spectrum

Hola Retro Parla

Contenido

Objetivo del curso

El objetivo principal es tener una primera toma de contacto con el microprocesador Z80, y con el lenguaje ensamblador.

Este curso pretende que adquieras las nociones básicas suficientes, para desarrollar el primer programa que se hace en todos los lenguajes, “Hola Mundo”, en nuestro caso Hola Retro Parla.

Esperamos qué tras la finalización del curso, te animes a seguir aprendiendo ensamblador para Z80.

¿Qué es el Z80?

El Z80 es un microprocesador que salió al mercado en 1976, de la mano de Zilog.

Se hizo muy popular en los 80 al ser usado por ordenadores como los Sinclair ZX Spectrum, Amstrad CPC o los sistemas MSX. También se usó en videoconsolas como la Sega Master System y la Sega Game Gear. Otras videoconsolas como la Neo-Geo, la Sega Mega Drive y muchas máquinas arcades usan el Z80 como procesador de sonido.

Hoy en día se sigue usando en sistemas embebidos.

El Z80 es una CPU de tipo LOW-ENDIAN, lo que significa que cuando almacena en memoria valores de 16 bits, primero almacena el byte menos significativo y después el más significativo.

Para almacenar en la posición de memoria $8000 el valor $1234, el Z80 almacena en la posición $8000 el valor $34 y en la $8001 el valor $12.

Los microprocesadores ortogonales admiten cualquier operación entre registros; este no es el caso del Z80. El Z80 no es ortogonal, por lo que es necesario saber que operaciones son admitidas.

Con este curso se entrega una lista con las operaciones admitidas por el Z80.

Registros del Z80

Los registros son memoria de alta velocidad y baja capacidad, que se encuentran integrados en el microprocesador.

El Z80 tiene registros de 8 y 16 bits.

Registros de 8 bits	
A	Acumulador. Destino de las operaciones aritméticas, lógicas y de comparación de 8 bits.
F	Flags. Conjunto de banderas que dan información de las operaciones que se están realizando.
B, C, D, E, H, L	Registros de propósito general. El registro B se suele usar en bucles. La instrucción DJNZ lo usa como contador.

I	Registro de interrupción. Permite manejar 128 interrupciones distintas.
R	Registro de refresco de la memoria. Lo maneja el Z80 y cambia los bits del 0 al 6. Se puede usar para generar números pseudo aleatorios entre 0 y 127.

Registros alternativos	
A', F', B', C' D', E', H', L'	Sirven para hacer una copia temporal de los otros registros de 8 bits.

Registros de 16 bits	
AF	Formado por el registro A como byte más significativo y el F como byte menos significativo.

BC	Formado por el registro B como byte más significativo y el C como byte menos significativo. Se usa como repetidor o contador en operaciones como LDIR, etc.
DE, HL	Formado por el registro H como byte más significativo y el L como byte menos significativo o el registro D como byte más significativo y el E como byte menos significativo. Se usan generalmente para leer y escribir en una operación única, así como en las operaciones como LDIR, etc. El registro HL hace funciones de registro acumulador de 16 bits.
IX, IY	Acceso a memoria de forma indexada. LD(IX+desplazamiento), valor Valor puede contener de -128 a 127.

SP	Puntero de pila. Apunta a la posición actual de la cabeza de la pila.
PC	Contador de programa. Contiene la dirección de la instrucción actual a ejecutar.

Los opcodes de los registros son:

0. B
1. C
2. D
3. E
4. H
5. L
6. (HL)

7. A

Cada bit del registro F, flags, tiene un significado propio que cambia automáticamente según el resultado de operaciones anteriores.

Registro F - indicadores de flags								
Bit	7	6	5	4	3	2	1	0
	S	Z	F5	H	F3	P/V	N	C

- **Flag S (signo):** se pone a 1 si el resultado de la operación en complemento a dos es negativo.
- **Flag Z (cero):** se pone a 1 si el resultado de la operación anterior es 0. Muy útil en bucles.

- **Flag H (acarreo BCD):** se pone a 1 cuando en operaciones BCD existe un acarreo del bit 3 al 4.
- **Flag P/V (paridad/desbordamiento):** sirve para dos tareas. En operaciones que modifican el bit de paridad, se pone a 1 cuando el número de unos del resultado es par. En operaciones que modifican el bit de desbordamiento, se pone a 1 cuando el resultado de la operación necesita más de 8 bits para ser representado.
- **Flag N (resta):** se pone a 1 si la última operación fue una resta.
- **Flag C (acarreo):** se pone a 1 si el resultado de la operación anterior necesita un bit extra para ser representado. Ese bit es el bit extra que se necesita.

No se puede acceder directamente al registro F y no todas las operaciones le afectan.

Memoria del ZX Spectrum 48K

La memoria del ZX Spectrum 48K está dividida en cuatro bloques de 16 KB, 16384 bytes.

El primer bloque, que va desde la posición $0000 a $3FFF (0 a 16.383), se corresponde con la ROM. La ROM está mapeada sobre la dirección $0000.

En segundo bloque, que va desde la posición $4000 a $7FFF (16.384 a 32.767), corresponde al área de pantalla, buffer de impresora, variables de sistema, etc.

El tercer bloque, que va desde la posición $8000 a la $BFFF (32.768 a 49.151), es memoria RAM de propósito general.

El cuarto bloque, que va desde la posición $C000 a la $FFFF (49.152 a 65.535), es memoria RAM de propósito general.

A grandes rasgos, la distribución del segundo bloque, $4000 a $7FFF, es la siguiente:

- **$4000 a $57FF (16384 a 22527)**: definición de píxeles en pantalla. La pantalla tiene una resolución de 256x192. Cada byte representa 8 píxeles. Ocupa 6KB, 6144 bytes.
- **$5800 a $5AFF (22528 a 23295)**: definición de atributos de color en pantalla. En este caso la resolución es de 32x24, caracteres. Cada byte da color a una zona de 8x8 píxeles. Define en los bytes 0 a 2 el color de tinta (de 0 a 7), en los bytes 3 a 5 el color de fondo (de 0 a

7), en el byte 6 el brillo (0/1) y el byte 7 parpadeo (0/1). Ocupa 768 bytes.

- **$5B00 a $5BFF (23296 a 23551):** Buffer de impresora. 256 bytes que se pueden usar si no tenemos impresora o no la usa el programa.
- **$5C00 a $5CB5 (23552 a 23733):** Variables de sistema.
- **$7FFF:** Pila. El puntero de la pila suele apuntar a esta dirección y según se ponen cosas en la pila, decrece.

Decimal, binario, hexadecimal

Estamos acostumbrados a ver los números en su representación decimal, una secuencia de dígitos en los que cada uno puede representar un valor del 0 a 9. Es lo que conocemos como números en base 10 o notación decimal.

En informática esto no es así; los ordenadores solo trabajan con dos valores, 0 y 1. Estos son conocidos como números binarios o en base 2.

La forma común de representar números en ensamblador es con notación hexadecimal o base 16.

En los números hexadecimales, cada dígito puede representar un valor del 0 a 15. A partir del 9 se usan letras, siendo A = 10, B = 11, C = 12, D = 13, E = 14 y F = 15. Cada dígito hexadecimal representa 4 bits. Solo con

ver el número, sabemos de cuántos bits se compone. Normalmente hablamos de bits múltiplos de 8; 8, 16, 32, 64, etc.

Una de las tareas más tediosas, si no tenemos una calculadora a mano, es la conversión de números a sus distintas bases.

Una forma sencilla de hacerlo es saber el valor que vale cada bit. En nuestro caso, números de 8 y 16 bits, que son con los que puede trabajar el Z80.

Para poder hacer las conversiones de manera sencilla, nos guiaremos con la siguiente tabla, en la que se muestra cuánto vale cada bit cuando está a 1:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
32718	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

De esta forma, simplemente sumando o restando, podemos convertir números de una a otra base.

Nada mejor que un ejemplo:

5FA0h →0101 1111 1010 0000 → 32 + 128 + 256 + 512 + 1024 + 2048 + 4096 + 16384 = 24480

Como se puede ver en el ejemplo, la conversión binario/hexadecimal es directa, haciéndola de 4 en 4 bits:

F0h → 1111 0000 3Ah → 0011 1010 CCh → 1100 1100 78h → 0111 1000

0001 0000 → 10h 0100 0101 → 45h 1010 1010 → AAh 0010 0011 → 23h

Etiquetas, variables y constantes

Las etiquetas permiten hacer referencia a posiciones de memoria, por nombres en lugar de por valores numéricos. El programa ensamblador es el que se encarga, al crear el código objeto, de sustituir las etiquetas por direcciones de memoria.

Si no se usan etiquetas, cada vez que se modificase el código, habría que recalcular las direcciones de memoria donde saltan JR, JP o CALL.

El ensamblador sustituye la etiqueta por la dirección de memoria de la instrucción siguiente a dicha etiqueta.

Las etiquetas se usan para definir rutinas o datos, pudiendo ser estos en formato numérico o de texto, y constantes o variables.

Para definir datos se usan las siguientes directivas:

- EQU: define constantes, **nombre EQU valor**.
- DB / DEFB: define bytes, **nombre DB 1, $FF, %10101010**.
- DM / DEFM: define message, **nombre DM "Hola Mundo"**.
- DW / DEFW: define word, **nombre DW $0040**.
- DS / DEFS: define space, **nombre DS $08**.

DB, DM, DW o DS no se ensamblan, por lo que al ponerlas antes del último RET del programa, se ejecutarán como si fueran instrucciones del Z80.

ORG y END

ORG y END son dos de las directivas más importantes de las que vamos a usar.

Con ORG cargamos nuestro código en una dirección de memoria concreta. Se pueden poner varios ORG para cargar distintas partes del programa, en distintas zonas de memoria.

Con END indicamos donde finaliza el programa y una dirección de autoinicio.

Ahora ya podemos realizar nuestro primer programa:

```
org $8000
ret
end $8000
```

Grabamos el archivo como holaretroparla.asm y compilamos con pasmo:

```
pasmo --name HolaRetroParla --tapbas holaretroparla.asm holaretroparla.tap --public
```

Como este comando lo vamos a utilizar intensivamente, podemos crear un archivo .bat para, de esa manera, solo tener que ejecutarlo, y no tener que escribir constantemente el comando.

Instrucciones de carga

Las instrucciones de carga sirven para cargar un valor en un registro, copiar un valor de un registro a otro, cargar un valor en memoria, cargar un registro en memoria y cargar en un registro un valor de memoria.

La sintaxis de las instrucciones de carga es:

LD Destino, Origen

Donde destino puede ser un registro o una posición de memoria, y origen puede ser un valor, un registro o una posición de memoria.

Los valores pueden ser de 8 y 16 bits.

Las instrucciones de carga no afectan al registro F, excepto LD A, I y LD A, R.

Vamos a recuperar nuestro programa y justo debajo de ORG, vamos a poner las siguientes líneas:

```
ld hl, $4000
ld (hl), $ff
```

Con estas líneas, vamos activar todos los bits de la primera dirección de memoria de la VideoRam.

Instrucciones RST

Las instrucciones RST se usan para realizar un salto a una dirección concreta mediante una instrucción de un solo opcode.

Existen varias, pero en nuestro caso solo vamos a usar RST $10 (16), que imprime el Ascii que hay en el registro A.

Volvemos a recuperar nuestro programa, quitamos las líneas que habíamos añadido y en su lugar añadimos las siguientes:

```
ld a, 'H'
rst $10
```

Compilamos y vemos como la letra H se imprime en pantalla.

Incrementos y decrementos

Se usan para incrementar (INC) o decrementar (DEC) en una unidad el contenido de determinados registros y posiciones de memoria apuntadas por HL, IX o IY.

Se permiten las siguientes operaciones:

INC r	DEC r
INC rr	DEC rr
INC (HL)	DEC (HL)
INC (IX+N)	DEC (IX+N)
INC (IY+N)	DEC (IY+N)

Las operaciones de incremento/decremento sobre registros de 16 bits, no afectan al registro F, mientras que las de 8 bits afectan de distinta manera.

	Flags					
Instrucción	**S**	**Z**	**H**	**P**	**N**	**C**
INC r	*	*	*	V	0	-
INC (HL)	*	*	*	V	0	-
INC (ri+N)	*	*	*	V	0	-
INC rr	-	-	-	-	-	-
DEC r	*	*	*	V	1	-
DEC rr	-	-	-	-	-	-

- No afecta, * afecta, 0 = se pone a 0, 1 = se pone a 1, V = Overflow

Vamos a recuperar de nuevo nuestro programa, pero esta vez lo vamos a dejar como al principio y vamos a empezar a utilizar todo lo visto hasta este momento.

```
org $8000           ; Dirección donde cargamos el programa.
ld hl, msg          ; Carga en HL la dirección de memoria del mensaje.
ld a, (hl)          ; Carga el primer carácter en A.
rst $10             ; Imprime el carácter.
inc hl              ; Apunta HL al siguiente carácter.
ld a, (hl)          ; Carga el carácter en A.
rst $10             ; Imprime el carácter.
ret
msg: defm 'Hola Retro Parla'
end $8000
```

¿Qué pasa si msg:… lo ponemos justo debajo de org $8000?

Operaciones lógicas

Se realizan a nivel de bits, entre dos bits. Las operaciones lógicas son:

- **AND:** multiplicación lógica. Solo es 1 si los dos bits son 1.
- **OR:** suma lógica. Es 1 si alguno de los bits es 1.
- **XOR:** or exclusivo. Es 0 cuando los dos bits son iguales y 1 cuando son distintos.

Bit 1	Bit 2	AND	OR	XOR
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

El formato de las instrucciones lógicas es:

AND ORIGEN

OR ORIGEN

XOR ORIGEN

El destino siempre es el registro A. XOR A = LD A, 0.

Las instrucciones lógicas afectan al registro F.

	Flags					
Instrucción	**S**	**Z**	**H**	**P**	**N**	**C**
AND s	*	*	*	P	0	0
OR s	*	*	*	P	0	0
XOR s	*	*	*	P	0	0

- No afecta, * afecta, 0 = se pone a 0, 1 = se pone a 1, P = Paridad

Cambios de flujo de programa

Permite cambiar el flujo del programa, de manera absoluta o relativa, con o sin condiciones.

Los saltos absolutos se realizan con JP y pueden ser:

- **JP NN:** salta a la dirección NN. NN puede ser una etiqueta.
- **JP (HL):** salta a la dirección de memoria contenida en el registro HL. Al valor de HL, no a la dirección apuntada por el registro.
- **JP (registro índice):** salta la dirección contenida por IX o IY.
- **JP NZ, NN:** salta si el indicador de cero (Z) está a cero.
- **JP Z, NN:** salta si el indicador de cero (Z) está a uno.
- **JP NC, NN:** salta si el indicador de acarreo (C) está a cero.
- **JP C, NN:** salta si el indicador de acarreo (C) está a uno.

- **JP PO, NN:** salta si el indicador de paridad/desbordamiento (P/O) está a cero.
- **JP PE, NN:** salta si el indicador de paridad/desbordamiento (P/O) está a uno.
- **JP P, NN:** salta si el indicador de signo está a cero, positivo.
- **JP M, NN:** salta si el indicador de signo está a uno, negativo.

Los saltos relativos se realizan con JR, y es relativo a la instrucción actual, saltando un número de bytes que va de -128 a 127. Las rutinas que solo tienen saltos relativos son reubicables:

- **JR N:** salta a una dirección de memoria que esta a N bytes. N puede ser una etiqueta.
- **JR NZ, N:** salta si el indicador cero (Z) está a cero.

- **JR Z, N:** salta si el indicador cero (Z) está a uno.
- **JR NC, N:** salta si el indicador de acarreo (C) está a cero.
- **JR C, N:** salta si el indicador de acarreo (C) está a uno.

Las instrucciones JP y JR no afectan al registro F.

Volvemos a nuestro programa y vamos a usar las operaciones lógicas y los cambios de flujo para imprimir todo el mensaje.

```
org $8000           ; Dirección donde cargamos el programa.
ld hl, msg          ; Carga en HL la dirección de memoria del mensaje.

Bucle:
ld a, (hl)          ; Carga un carácter de la cadena.
or a                ; Comprueba si A es 0. A OR A = 0 solo si A = 0.
jr z, Fin           ; Si A = 0, salta a la etiqueta Fin.
rst $10             ; Imprime el carácter.
inc hl              ; Incrementa HL para apuntar al siguiente carácter.
jr Bucle            ; Bucle hasta que A = 0.

Fin:
ret
msg: defm 'Hola Retro Parla', $00   ; Cadena terminada en 0 = null.
end $8000
```

Subrutinas

Son bloques de código que hacen una acción concreta, al que se puede llamar en múltiples ocasiones. Se usa CALL para saltar a una subrutina y RET para retornar.

CALL es similar a JP, pero antes de saltar hace un PUSH PC, para guardar por dónde va el programa, y RET hace POP PC, para volver por dónde iba.

Igual que en JP y JR, se pueden hacer llamadas condicionales. De igual manera se pueden hacer retornos condicionales:

CALL FLAG, NN **RET FLAG**

Volvemos a nuestro programa para, gracias a CALL, llamar a alguna rutina de la ROM que va a hacer que empiece a ser más interesante.

```
org $8000           ; Dirección donde cargamos el programa.

; Variable de sistema donde están los atributos de la pantalla 1, principal. FBPPPIII.
ATTR_S: equ $5c8d

; Variable de sistema donde está el atributo actual. FBPPPIII.
ATTR_T: equ $5c8f

; ------------------------------------------------------
; Rutina de la ROM similar al AT de Basic.
; Posiciona el cursor en las coordenadas especificadas.
; Entrada: B -> Coordenada Y.
;          C -> Coordenada X.
; Para esta rutina, la esquina superior izquierda de la pantalla es (24, 33).
; Altera el valor de los registros A, DE y HL.
```

```
; ------------------------------------------------------
LOCATE: equ $0dd9

; ------------------------------------------------------
; Rutina de la ROM semejante al CLS de Basic.
; Borra la pantalla usando los atributos cargados en
; en la variable de sistema ATTR_S.
; Altera el valor de los registros AF, BC, DE y HL.
; ------------------------------------------------------
CLS: equ $0daf

Inicio:
ld a, $0e           ; Carga en A los atributos de flash, brillo, papel y tinta.
ld hl, ATTR_T       ; Carga en HL la dirección de memoria donde se encuentran los atributos
                    ; de la pantalla principal.
ld (hl), a          ; Carga en memoria los atributos de la pantalla principal.

ld hl, ATTR_S       ; Carga en HL la dirección de memoria donde se encuentran los atributos
                    ; actuales.
```

```
ld (hl), a          ; Carga en memoria los atributos actuales.

call CLS            ; Limpia la pantalla usando los atributos de ATTR_T.

ld b, $18-$0a       ; Carga la coordenada Y en B.
ld c, $21-$08       ; Carga la coordenada X en C.
call LOCATE         ; Posiciona el cursor.

ld hl, msg          ; Carga en HL la dirección de memoria del mensaje.

Bucle:
ld a, (hl)          ; Carga un carácter de la cadena.
or a                ; Comprueba si A es 0. A OR A = 0 solo si A = 0.
jr z, Fin           ; Si A = 0, salta a la etiqueta Fin.
rst $10             ; Imprime el carácter.
inc hl              ; Incrementa HL para apuntar al siguiente carácter.
jr Bucle            ; Bucle hasta que A = 0.

Fin:
```

```
jr Fin              ; Bucle infinito.

msg: defm 'Hola Retro Parla', $00       ; Cadena terminada en 0 = null.
end $8000
```

Puertos de entrada y salida

Por último, vamos a ver los puertos de entrada y salida, que se usan entre otras cosas para leer el teclado, el joystick, etc.

Solo vamos a ver uno de sus usos, que es la asignación del color para el borde de la pantalla.

Para asignar el borde vamos a usar la instrucción OUT, usando el puerto de salida $FE.

Vamos a usar un sencillo programa para ver como cambiar el borde:

```
org $8000
ld a, $01
out ($fe), a
ret
end $8000
```

Hola Retro Parla

Y ahora ya solo queda mostrar el listado completo de nuestro programa.

```
; ------------------------------------------------
; Hola Retro Parla.
; ------------------------------------------------
org $8000           ; Dirección donde cargamos el programa.

; Variable de sistema donde están los atributos de la pantalla 1, principal. FBPPPIII.
ATTR_S: equ $5c8d

; Variable de sistema donde está el atributo actual. FBPPPIII.
ATTR_T: equ $5c8f

; ------------------------------------------------
; Rutina de la ROM similar al AT de Basic.
; Posiciona el cursor en las coordenadas especificadas.
```

```
; Entrada: B -> Coordenada Y.
;          C -> Coordenada X.
; Para esta rutina, la esquina superior izquierda de la pantalla es (24, 33).
; Altera el valor de los registros A, DE y HL.
; -----------------------------------------------------
LOCATE: equ $0dd9


; -----------------------------------------------------
; Rutina de la ROM semejante al CLS de Basic.
; Borra la pantalla usando los atributos cargados en la variable de sistema ATTR_S.
; Altera el valor de los registros AF, BC, DE y HL.
; -----------------------------------------------------
CLS: equ $0daf


Inicio:
ld a, $0e           ; Carga en A los atributos de flash, brillo, papel y tinta.
ld hl, ATTR_T       ; Carga en HL la dirección de memoria donde se encuentran
                    ; los atributos de la pantalla principal.
ld (hl), a          ; Carga en memoria los atributos de la pantalla principal.
```

```
ld hl, ATTR_S       ; Carga en HL la dirección de memoria donde se encuentran
                    ; los atributos actuales.
ld (hl), a          ; Carga en memoria los atributos actuales.

ld a, $01           ; Carga en A el color del borde.
out ($fe), a        ; Manda al puerto $fe el color del borde.

call CLS            ; Limpia la pantalla usando los atributos de ATTR_T.

ld b, $18-$0a       ; Carga la coordenada Y en B.
ld c, $21-$08       ; Carga la coordenada X en C.
call LOCATE         ; Posiciona el cursor.

ld hl, msg          ; Carga en HL la dirección de memoria del mensaje.

Bucle:
ld a, (hl)          ; Carga un carácter de la cadena.
or a                ; Comprueba si A es 0. A OR A = 0 solo si A = 0.
```

```
jr z, Fin           ; Si A = 0, salta a la etiqueta Fin.
rst $10             ; Imprime el carácter.
inc hl              ; Incrementa HL para apuntar al siguiente carácter.
jr Bucle            ; Bucle hasta que A = 0.

Fin:
jr Fin              ; Bucle infinito.

msg: defm 'Hola Retro Parla', $00       ; Cadena terminada en 0 = null.
end $8000
```

El listado en Basic, sería algo así como:

```
10 BORDER 1: PAPER 1: INK 6: CLS: PRINT AT 10, 10; "Hola Retro Parla"
20 GO TO 20
```

Enlaces de interés

http://clrhome.org/table/

http://z80-heaven.wikidot.com/math

https://wiki.speccy.org/programacion/indice

https://wiki.speccy.org/programacion/recursos_prog/indice

http://www.profesorenlinea.cl/matematica/Numeros_Bases_Numericas.html