Índice

Ensamblador Z80. Estructura Instrucciones

- Las instrucciones en el ensamblador del Z80 tienen la siguiente estructura:

- Donde:
  - **Etiqueta:** Una nombre único que identifica de manera unívoca una línea del programa. Es opcional.
  - **Operación:** Acoge el nemónico de la operación ensamblador que se desea realizar.
  - **Operando(s):** Corresponde al operando de la operación. Puede ser simple o doble.
    - Si es doble, el primer operando es el destino y el segundo es el origen.
  - **Comentarios:** Debe comenzar por el símbolo ‘;’. Es un espacio para incluir comentarios sobre el programa. Es opcional.

Ensamblador Z80. Ejemplo

- Los **operandos pueden ser registros, direcciones de memoria, valores inmediatos, etiquetas,** etc., dependiendo del tipo de instrucción.
- Algunos ejemplos de instrucciones en ensamblador Z80 son:
  - LD A, 42 ; carga el valor 42 en el acumulador
  - LD B, C ; copia el contenido del registro C en el registro B
  - ADD A, B ; suma el contenido de A y B, almacenando el resultado en A
  - JP 1234h ; salta a la dirección de memoria 1234h
  - CALL mi_subrutina ; llama a la subrutina identificada por la etiqueta "mi_subrutina”
- Ya veremos más adelante con detenimiento cada una de estas instrucciones…

Ensamblador Z80. Estructura Programa

Directiva de comienzo: Indica en qué dirección de memoria está la primera instrucción del programa.

```
                org 200H

program         ld A,37H
                ld (210H),A
                ld C,A

                end program
```

Instrucciones del programa

Etiqueta de inicio: Marca la dirección de memoria del inicio.

Directiva de fin: Indica que el programa se ha acabado.

Ensamblador Z80. Representación.

- **Los números se pueden representar en diferentes bases** en el ensamblador. Algunas instrucciones admiten diferentes bases, otros son dependientes de una base concreta.
  - En **decimal**: 34D o, símplemente, 34. (Ojo 34D no funciona en ZEUS)
  - En **hexadecimal**: 34H o, si comienza por letra, 0A4H
  - En **binario**: 00110100B

```
          org 200H

Program   ld A,37H
          ld (210H),A ;source only A
          ld C,A

          end program
```

Ensamblador Z80. LD

- La instrucción que más utilizaremos en nuestros programas en ensamblador será sin duda la **operación de carga o instrucción LD**. Sirve para:
  - Meter un valor en un registro.
  - Copiar el valor de un registro en otro registro.
  - Escribir en memoria (en una dirección determinada) un valor.
  - Escribir en memoria (en una dirección determinada) el contenido de un registro.
  - Asignarle a un registro el contenido de una dirección de memoria.
- La **sintaxis de LD** en lenguaje ensamblador es:

LD DESTINO, ORIGEN

- La instrucción **LD** copia el valor contenido en el operando origen en el operando destino.
- Esta operación no es aritmética, ni lógica ni de desplazamiento, por lo que no afecta a ningún *flag* (F).

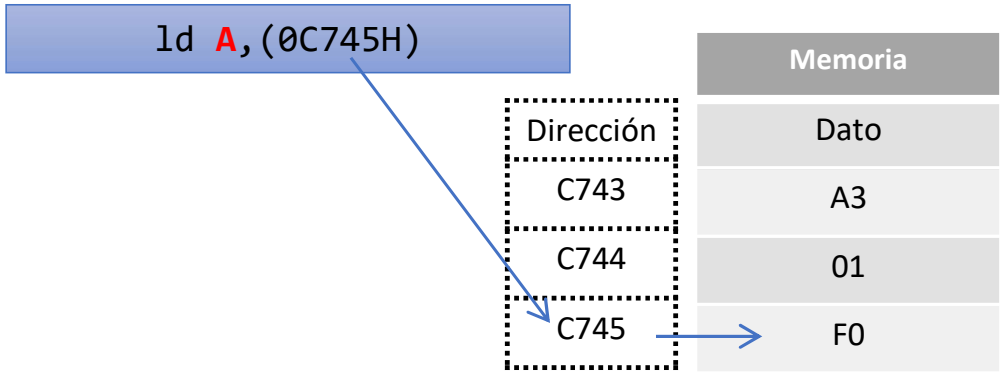
Ensamblador Z80. Direccionamiento

- **Direccionamiento** es la manera en la que se indica a una instrucción dónde están los operandos sobre los que debe operar.
- El ensamblador **soporta diferentes modos de direccionamiento** para acceder a datos y operar con ellos. Los principales modos son:
  - Direccionamiento **extendido**.
  - Direccionamiento **indirecto.**
  - Direccionamiento **indexado.**
  - Direccionamiento **a través de registro.**
  - Direccionamiento **implícito.**
  - Direccionamiento **inmediato.**
  - Direccionamiento **inmediato extendido.**

Ensamblador Z80. Direccionamiento

Direccionamiento extendido

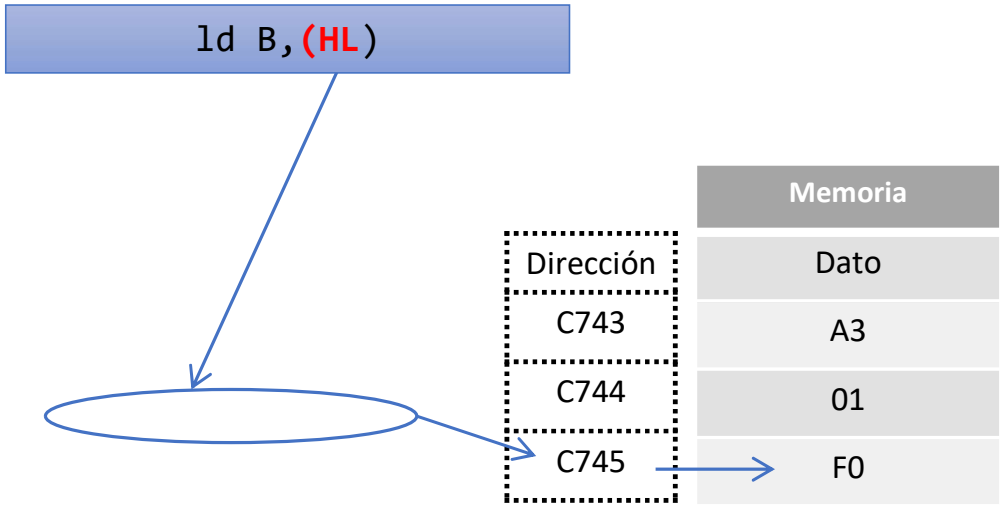
- El **operando es la dirección de memoria** donde está el dato.
  - Las operaciones con memoria solo pueden tener origen o destino A.
  - Si necesitamos cargarlo en otro registro, hay que hacerlo en dos pasos (ej. ld A, (0C745H) y después ld B, A)
  - El operando tiene 2 bytes, para contener una dirección completa de memoria del Z80.
  - Los registros dobles (BC, DE y HL) también pueden interactuar con la memoria



Ensamblador Z80. Direccionamiento

Direccionamiento indirecto:

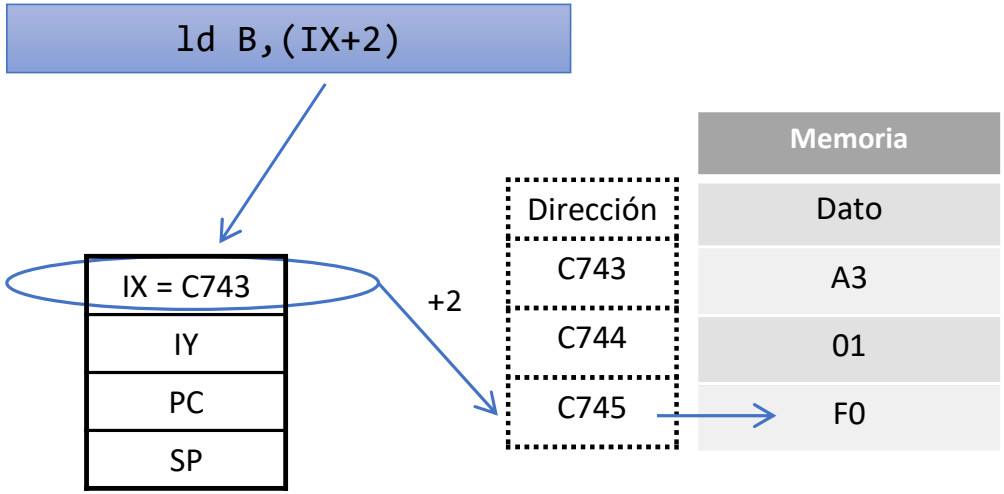
- El **operando es un par de registros** que contienen la dirección de memoria donde está el dato.
  - El operando tiene 2 registros de 1 byte cada uno, para contener una dirección completa de memoria.
  - Todos los registros (A, B, C, D, E, H y L) aceptan como dirección de origen la contenida en el registro (HL). El registro A también acepta, como origen, los registros (BC) y (DE). Todos también aceptan IX e IY



Ensamblador Z80. Direccionamiento

Direccionamiento indexado:

- El **operando es un registro de 2 bytes (IX e IY) más un desplazamiento**.
  - La dirección contenida en el registro más el desplazamiento, resulta en la dirección de memoria donde está el dato. Destino todos los registros de 8 bits (A..L).
  - Todos los registros (A, B, C, D, E, H y L) aceptan como dirección de origen la contenida en los registro IX e IY



Ensamblador Z80. Direccionamiento

Direccionamiento a través de registro:

- **El operando es un registro.**
  - No se accede a memoria, ya que el dato ya está contenido en el registro.
  - Origen y destino todos los registros de 8 bits (A…L)

```
ld B,C
```

Ensamblador Z80. Direccionamiento

Direccionamiento implícito:

- **No hay operandos, la instrucción actúa automáticamente sobre un registro** concreto.
  - No se accede a memoria, ya que el dato ya está contenido en el registro.

```
rlca
```

RLCA: rota A un bit hacia la izquierda.

Ensamblador Z80. Direccionamiento

Direccionamiento inmediato:

- El operando es el dato.
  - No se accede a memoria ni a registros.

```
ld B,133D
```

Direccionamiento inmediato extendido:

- Igual que el anterior, pero para números que se van a almacenar en registros de 2 bytes.
  - Registros de 2 bytes; BC, DE, LH, IX, IY..

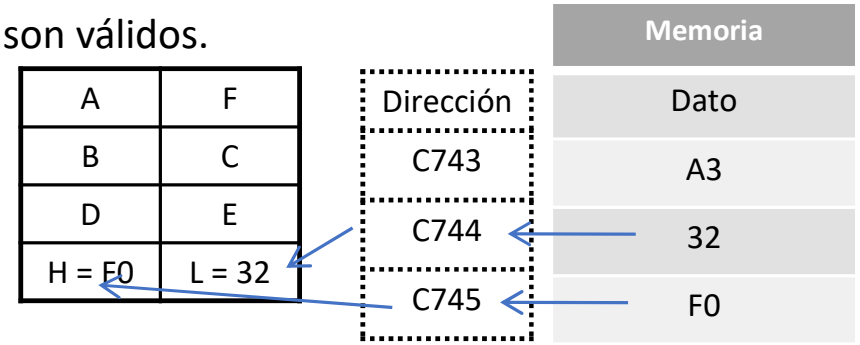
```
ld HL,8000H
```

Ensamblador Z80. LD

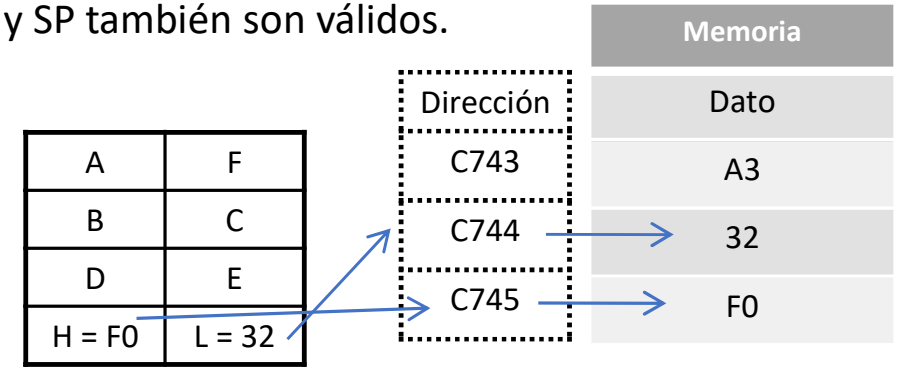
- **Atendiendo a los modos de direccionamiento, la instrucción LD se puede encontrar de las siguientes formas:**
  - LD A, (1000H); sólo registro A
  - LD A, (BC) ; DE y HL también son válidos
  - LD R, (HL) ; donde R = A, B, C, D, E, H, L
  - LD R, (IX + d) ; donde d = desplazamiento
  - LD R, (IY + d); ld HL, (IX+d o IY +d) da error, en cambio con BC y DE no da error
  - LD R, n ; donde n es un número desde 00 a FF y R = A, B, C, D, E, H, L. R puede sustituirse por (HL), (IX + d) e (IY + d)
- Viceversa para **almacenar datos del registro** en lugar de cargar datos en el registro.
  - LD (1000H), A; sólo registro A
- También se puede **transferir el contenido de registro a registro**:
  - LD R1, R2 ; donde R1 = A, B, C, D, E, H, L y R2 = A, B, C, D, E, H, L

Ensamblador Z80. LD

- LD también **admite direccionar datos de 2 bytes**:
  - LD HL, (0C744H); BC, DE, IX, IY y SP también son válidos.



- El byte **menos significativo** está almacenado en la **posición indicada**. El byte **más significativo** en la **posición siguiente**. Esto se llama **Low Endian**.
- Viceversa para almacenar datos de un par de registros en dos direcciones consecutivas de memoria.
  - LD (0C744H), HL ; BC, DE, IX, IY y SP también son válidos.



Ensamblador Z80. LD

- **Transferencia entre registros 2 bytes:**
  - LD SP, HL; IX e IY también son válidos.
- **Carga de números de 2 bytes:**
  - LD BC, nn ; donde nn es un número entre 0000 y FFFF.
  - LD DE, nn
  - LD HL, nn
  - LD SP, nn
  - LD IX, nn
  - LD IY, nn

Ensamblador Z80. Intercambio

- En lugar de cargar o almacenar, es **posible intercambiar valores entre pares de registros con la instrucción EX**:
  - EX DE, HL
  - EX AF, AF’
  - EX (SP), HL
  - EX (SP), IX
  - EX (SP), IY
- Para intercambiar los valores de los registros actuales por los del segundo juego de registros:
  - EXX

Ejercicio 1

- Escribir un programa en ensamblador del Z80 que:
  - Se almacene en la dirección 8000h.
  - Cargue el contenido de la dirección de memoria 300h en el acumulador.
  - Copie el contendido del acumulador en el registro E.
  - Copie el contenido del registro E en el registro B.
  - Marque con la etiqueta “Aquí” cualquiera de las líneas del programa.
  - Que incluya un comentario en la última línea de programa

Solución

```
        org 8000h
        ld a, 300h
Aqui    ld e, a
        ld b, e     ; Última línea
```

Ejercicio 2

Escribir un programa en ensamblador del Z80 que:

- Se almacene en la dirección 8000h.
- Cargue en acumulador (A) el valor 5.
- Cargue en la dirección de memoria 8250h el contenido del acumulador.
- Cargue en acumulador el valor 4.
- Cargue en la dirección de memoria 8251h el contenido del acumulador.
- Cargue en el registro b el contenido del acumulador.
- Sume a y b
- Salvar el resultado en la dirección de memoria 8260h

Solución

```
org 8000h
        ld a, 5
        ld (8250h), a
        ld a, 4
        ld  (8251h), a
        ld b, a
        ld a, (8250h)
        add a, b
        ld (8260h), a
        halt
```

Ensamblador Z80. Aritmética y lógica

- La mayoría de las operaciones aritméticas en Z80 se realizan mediante el registro acumulador A.
- Las principales instrucciones aritméticas son:
  - ADD: Suma dos valores y almacena el resultado en un registro.
  - SUB: Resta dos valores y almacena el resultado en un registro.
  - INC: Incrementa el valor de un registro o de una ubicación de memoria en uno.
  - DEC: Decrementa el valor de un registro o de una ubicación de memoria en uno.
- Además, se cuenta con las siguientes instrucciones lógicas:
  - AND: Realiza una operación lógica AND entre dos valores y almacena el resultado en un registro.
  - OR: Realiza una operación lógica OR entre dos valores y almacena el resultado en un registro.
  - XOR: Realiza una operación lógica XOR entre dos valores y almacena el resultado en un registro.
  - NEG y CPL: Hacen complemento a 2 y niegan (respectivamente) lo almacenado en el acumulador.

Ensamblador Z80. Aritmética (ADD)

- ADD se encuentra en las formas:
  - ADD A, R ; donde R = A, B, C, D, E, H, L
  - ADD A, n ; donde n es un número de 00 a FF.
  - ADD A, (HL)
  - ADD A, (IX + d)
  - ADD A, (IY + d)
- ADD también acepta operaciones con registros dobles, de 16 bits. ADD D, S; donde D = HL, IX, IY y S = BC, DE. En este caso el resultado se queda en el primer registro doble, no interviene el registro A. INC y DEC también se pueden aplicar a todos los registros dobles. También admiten las sumas consigo mismo ADD D,D, es una forma de multiplicar por dos, que también se puede obtener con un desplazamiento a la izquierda.
- Si se desea que se sume teniendo en cuenta el acarreo anterior (indicado por el bit C del registro F), la instrucción a usar es ADC

Ensamblador Z80. Aritmética (ADD)

F

7	6	5	4	3	2	1	0
S	Z	-	H	-	P/V	N	C

- Flags activados por la operación:
  - S = 1 cuando el resultado es negativo.
  - Z = 1 cuando el resultado es cero.
  - H = 1 cuando hay acarreo en el bit 3.
  - P/V = 1 cuando hay overflow.
  - C = 1 cuando hay acarreo en el bit 7.

Ensamblador Z80. Ejemplo ADD

```
; este programa suma dos números y muestra el resultado en pantalla

LD A, 10     ; carga el primer número en A

LD B, 20     ; carga el segundo número en B

ADD A, B     ; suma A y B, almacenando el resultado en A

LD C, A      ; copia el resultado en el registro C

CALL mostrar_resultado     ; llama a una subrutina (faltaría implementarla)

; fin del programa
```

Ejercicio 3

Escribir un programa en ensamblador del Z80 que:

- Almacene los números 44H y 2AH en las direcciones de memoria 250H y 251H.
- Almacene la suma de ambos números en la dirección 260H.

Solución

```
ld a, 44h

ld (250h), a

ld a,  2ah

ld (251h), a

ld b, a

ld a, (250h)

add a, b

ld (260h), a
```

Ensamblador Z80. Aritmética (SUB)

- SUB realiza una operación de resta entre dos operandos y almacena el resultado en un registro especificado.
  - Es análoga a la operación de suma (ADD), pero el primer operando siempre es el acumulador A, por ello la sintaxis no se incluye.
  - Al contrario de ADD, sólo admite operaciones con registros de 8 bits.
  - También se puede hacer resta con Carry con SBC
- Flags activados por la operación:
  - S = 1 cuando el resultado es negativo.
  - Z = 1 cuando el resultado es cero.
  - H = 1 cuando hay acarreo en el bit 3.
  - P/V = 1 cuando hay overflow.
  - N (Subtract Flag): este bit se establece en 1 debido al uso de la instrucción SUB, y se borra en caso contrario.
  - C = 1 cuando hay acarreo en el bit 7.

Ejercicio 4

- Programar un programa que realice los siguientes pasos:
  - Carga el valor 10 en el acumulador A.
  - Carga el valor 5 en el registro B.
  - Resta el valor en B al valor en A mediante la instrucción SUB.

Solución

```
org 8000h   ; Dirección de inicio del programa

   ; Carga del primer número

   ld a, 10    ; Carga el valor 10 en el acumulador A

   ; Carga del segundo número

   ld b, 5     ; Carga el valor 5 en el registro B

   ; Resta de los números

   sub b       ; Resta el valor en B al valor en A

    ret        ; Finaliza el programa
```

Ensamblador Z80. Aritmética (INC y DEC)

- INC Incrementa en 1 el valor referido por el operando. Sólo se aplica sobre 8 bits, se puede usar para incrementar valores almacenados en posiciones de memoria
  - INC R ; donde R = A, B, C, D, E, H, L
  - INC (HL)
  - INC (BC)
  - INC (DE)
  - INC (IX + d)
  - INC (IY + d)
- La función DEC es análoga, pero decrementa en 1 el valor referido por el operando.
- Los flags S, Z y H se activan como en ADD.
- P/V se activa si el valor en R antes de la operación es 7FH (INC) o 80H (DEC).
- C no se ve afectado.

Ensamblador Z80. Lógica (AND)

- En ensamblador, la instrucción AND se utiliza para realizar una operación lógica AND a nivel de bit entre el acumulador y otro operando. El resultado se almacena en el acumulador y se actualizan los flags del registro F.
- La sintaxis general de la instrucción AND en ensamblador Z80 es la siguiente:

  AND operando

- Donde operando puede ser un registro, una dirección de memoria o un valor inmediato.
- Después de ejecutar la instrucción AND, los flags del registro F se actualizan de la siguiente manera:
  - El bit de signo (S) se pone a cero.
  - El bit de cero (Z) se pone a uno si el resultado es cero; de lo contrario, se pone a cero.
  - El bit de paridad/imparidad (P/V) se pone a uno si el resultado tiene un número par de bits activos; de lo contrario, se pone a cero.
  - El bit de sobrecarga/medio acarreo (H) se pone a uno si la operación ha generado un acarreo a nivel de bit 3-4; de lo contrario, se pone a cero.
  - El bit de acarreo (C) se pone a cero.

Ensamblador Z80. Lógica (AND)

- Registro: se especifica un registro de 8 bits como operando. Ejemplo: AND A.
- Inmediato: se especifica un valor de 8 bits como operando. Ejemplo: AND 0x0F.
- Directo: se especifica una dirección de memoria como operando. El valor se obtiene de la dirección de memoria especificada. Ejemplo: AND (0x1234).
- Indirecto: se especifica un registro que contiene una dirección de memoria como operando. El valor se obtiene de la dirección de memoria apuntada por el registro. Ejemplo: AND (HL) o AND (IX+d).
- Cada modo de direccionamiento tiene su propia sintaxis y su propio propósito. Por ejemplo, el modo inmediato se utiliza para realizar operaciones lógicas con valores constantes, mientras que el modo indirecto se utiliza para trabajar con valores almacenados en memoria. La elección del modo de direccionamiento adecuado depende de la tarea específica que se esté realizando.

Ensamblador Z80. Lógica (OR)

- La instrucción OR es una instrucción lógica que realiza una operación OR bit a bit entre el acumulador (registro A) y otro registro o valor. El resultado se almacena en el acumulador.
- La sintaxis es la siguiente:

OR registro/valor

  - Siendo el registro uno de los siguientes: B, C, D, E, H, L, A
  - O el valor inmediato de 8 bits en formato decimal o hexadecimal (prefijado por el símbolo #)

Ensamblador Z80. Lógica (XOR)

- La instrucción **OR establece las banderas (flags)** del registro F de la siguiente manera:
  - Z (Zero): se establece en 1 si el resultado de la operación es cero; en caso contrario, se establece en 0.
  - N (Add/Subtract): se establece en 0.
  - H (Half Carry): se establece en 0.
  - PV (Parity/Overflow): se establece en 1 si el número de bits a 1 en el resultado es par; en caso contrario, se establece en 0.
  - S (Sign): se establece en 1 si el bit más significativo (significante) del resultado es 1; en caso contrario, se establece en 0.
  - C (Carry): se establece en 0.

Ensamblador Z80. Lógica (XOR)

- La instrucción XOR sigue la sintaxis de AND y OR. Sus flags son:
  - Z (Zero flag): se establece en 1 si el resultado de la operación es cero, es decir, si todos los bits del acumulador y el operando son iguales a cero. En caso contrario, se establece en 0.
  - S (Sign flag): se establece en el bit más significativo del resultado de la operación.
  - P/V (Parity/Overflow flag): se establece en 1 si el número de bits a 1 en el resultado de la operación es par. En caso contrario, se establece en 0.
  - H (Half-carry flag): se establece en 1 si hubo un acarreo desde el bit 3 al bit 4 durante la operación. Si no, se establece en 0.
  - N (Add/Subtract flag): se establece en 0 para indicar que se ha realizado una operación de tipo XOR.
  - C (Carry flag): se establece en 0 ya que no se realiza ninguna operación de acarreo en la instrucción XOR.
- No modifica el contenido de los registros de propósito general (como B, C, D, E, H, L), sino que solo opera con el acumulador (A) y el operando especificado.

Ensamblador Z80. Labels

- Una etiqueta es un nombre simbólico asignado a una posición de memoria o una instrucción dentro del programa.
- Las etiquetas se utilizan para hacer referencia a direcciones de memoria específicas y facilitar la lectura y la comprensión del código.
  - Normalmente proporcionan un nombre significativo que identifica el propósito o la función de la ubicación de memoria. Por ejemplo:
    - En un programa que multiplica dos números, se podría utilizar una etiqueta "multiplicar" para referirse a la ubicación en el código donde se realiza la multiplicación, en lugar de referirse a la dirección de memoria específica.

Ensamblador Z80. Etiquetas.

Los usos/instrucciones más comunes de las etiquetas son:

- Marcar el **comienzo y fin de un programa: ORG** nn y **END** label;
- Etiquetar un valor: label **EQU** nn ; (este valor no podrá cambiar en el programa)
- Reserva de memoria:
  - label **DEFB** n, n,… ; guarda el valor n (8 bits) y sucesivos (00-FF) en la dirección de memoria correspondiente al puntero de dirección y sucesivas.
  - label **DEFW** nn, nn,… ; guarda el valor nn (16 bits) y sucesivas (0000-FFFF) en la dirección de memoria correspondiente al puntero de dirección y sucesivas.
  - label **DEFS** nn ; reserva nn (0000-FFFF) casillas de memoria a partir del puntero de dirección.
    - A “label” se le asigna la dirección de la primera casilla reservada.
    - El valor de las casillas no se modifica.

Ensamblador Z80. Etiquetas.

Los usos/instrucciones más comunes de las etiquetas son:

- Marcar el **comienzo y fin de un programa: ORG** nn y **END** label;
- Etiquetar un valor: label **EQU** nn ; (este valor no podrá cambiar en el programa)
- Reserva de memoria:
  - ...
  - label **DEFM** “cadena” ; almacena una cadena de caracteres ASCII en memoria a partir del puntero de dirección.
    - A “label” se le asigna la dirección de la casilla reservada en memoria con el primer carácter.
    - Cada carácter ASCII ocupa 16 bits, por lo tanto se necesitan registro dobles para su manejo

Ejercicio 5

Escribir un programa en ensamblador del Z80 que:

- Reserve memoria suficiente y almacene en ella los números 44H, 2AH, A3H y 71H.
- Reserve memoria suficiente y almacene en ella los números 10H, E2H, 6BH y 18H.
- Reserve memoria suficiente y almacene en ella la suma los números anteriores por pares (44H + 10H, 2AH + E2H, …).

Solución

```
AppEntry        ld ix, numeros1
                ld iy, numeros2
                ld hl, resul
                ld b, 4
bucle           ld a, (ix)
                add a, (iy)
                ld (hl), a
                inc ix
                inc iy
                inc hl
                djnz bucle
fin             halt            ; Replace these lines
with your code
                jp fin          ;
numeros1        defb 44h, 2ah, 0a3h, 71h
numeros2        defb 10h, 0e2h, 6bh, 18h
resul           defs 4
```

```
                ld ix, oper1
                ld b, 4
bucle           ld a, (ix)
                add a, (ix + 4)
                ld (ix + 8), a
                inc ix
                djnz bucle
fin             halt
                jp fin

oper1           defb 44h, 2ah, 0a3h, 71h
oper2           defb 10h, 0e2h, 6bh, 18h
resul           defs 4
```

Ensamblador Z80. Desplazamiento y rotación

- Existen diversas instrucciones muy útiles para trabajar con números binarios y realizar operaciones de multiplicación o división por potencias de 2, así como para realizar operaciones de enmascaramiento y extracción de bits individuales. Los operandos son, todos los registros sencillos, y los registros indexados; (HL), (IX +d), (IY + d)
  - Las **instrucciones de desplazamiento** empiezan por S (shift), después se indica si el desplazamiento es a izquierda (L) o derecha (R) y finalmente si es desplazamiento lógico (L) o Aritmético (A)
    - Lista de instrucciones: SLA, SRA, SLL, SRL. Más nombre de Registro
  - **Rotación** empiezan por R (rotate) y puedes ser a la izquierda (L), a la derecha (R) y circulares (C) o no circulares:
    - Lista de instrucciones: RL, RR, RRC, RCL
  - Hay instrucciones en las que el **operando es el registro A**; RLCA, RLRCA, RLA, RRA

Ensamblador Z80. Comparador

- La instrucción CP compara el contenido del acumulador con el del operando.
  - No cambia el valor del acumulador!
- Se encuentra en las siguientes formas:
  - CP R ; donde R = A, B, C, D, E, H, L
  - CP n ; donde n es un número de 00 a FF.
  - CP (HL)
  - CP (IX + d)
  - CP (IY + d)
- Si los valores son iguales, el flag Z = 1.
- Si A > operando, entonces Z = 0 y C = 0.
- Si A < operando, entonces Z = 0, y C = 1.

Ensamblador Z80. Saltos condicionales

- Dos de las bases de todo lenguaje de programación son las sentencias condicionales (IF) y los bucles (FOR/WHILE). En ensamblador del Z80, dichas funcionalidades se implementan a través de los saltos condicionales.
  - Un salto hace que la siguiente instrucción en ejecutar no sea la inmediatamente posterior a la que se está ejecutando, sino la referenciada por una etiqueta o la relativa a una posición de memoria concreta. Las instrucciones posibles de salto son:
    - JP ; usada para saltar a una instrucción situada en cualquier zona de memoria.
    - JR ; más ligera, usada para saltar a una distancia de [-127, +128] direcciones desde la dirección de la instrucción actual.
  - Se pueden encontrar como:
    - JP “label” ; JP (HL) ; JP (IX) ; JP (IY); JP (8050H)
    - JR “label”

Ensamblador Z80. Saltos condicionales

- Para establecer un salto condicional, se puede añadir una condición como primer operando de la instrucción. Las condiciones posibles son:

nemónico	significado	JP	JR
-	incondicional		
NZ	Z = 0		
Z	Z = 1		
NC	C = 0		
C	C = 1		
PO	P/V = 0		
PE	P/V = 1		
P	S = 0		
M	S = 1		

- Ejemplos:
  - JP Z, etiqueta1 ; salta a “etiqueta1” si Z = 1
  - JR NC, etiqueta2 ; salta a “etiqueta2” si C = 0

Ensamblador Z80. Bifurcaciones

- La instrucción DJNZ “label” es un caso especial, denominado decremento y bifurcación.
- Esta única instrucción ejecuta todo lo siguiente:
  - Decrementa el valor contenido en el registro B en 1.
  - Comprueba si B = 0.
    - En caso afirmativo, el programa continúa con la siguiente instrucción.
    - En caso contrario, salta a la etiqueta “label”.
- Es la manera más sencilla de crear un bucle.
- Siempre utiliza el registro B, por lo que hay que tener cuidado de inicializarlo correctamente sin pisar ningún valor importante.

Ejercicio 6

Escribir un programa en ensamblador del Z80 que:

- Almacene 10 números en memoria.
- Sume los 10 números y deje el resultado en otra posición de memoria previamente reservada.

Solución

```
org AppFirst                ; Start of application

AppEntry    ld b, 10
            ld hl, numeros
bucle       add a, (hl)
            inc hl
            djnz bucle

fin         halt
            jp fin          ;

numeros         defb 1,1,1,1,1,1,1,1,1,1
```

Ejercicio 7

Escribir un programa en ensamblador del Z80 que:

- Almacene un valor de una cifra en el registro C.
- Almacene un valor de una cifra en el registro D.
- Compare ambos valores y deje en el acumulador una cifra que indique el resultado de la comparación (1: C > D; 0: C = D; -1: D > C).

Solución

```
                org AppFirst            ; Start of application
inicio          ld c, 2
                ld d, 2
                ld a, d
                cp c
                jp z, iguales               ; A=0,  z=1 son iguales
                jp nc, dmayor   ; A=-1, z=0 son diferentes y c=0 a>operando, d>c
                                ; A=1,  z=0 son diferentes y c=1 a<operando, d<c
                ld a, 1                     ; aquí llegaría cuando d<c
                jp Inicio
iguales         ld a,0
                jp  Inicio
dmayor              ld a,-1
                jp  Inicio
```

Ensamblador Z80. Rutinas

- Las rutinas son pequeños fragmentos de código incluidos en un programa ensamblador para realizar una función concreta.
- Las rutinas se deben etiquetar en su primera línea con una etiqueta aclaratoria de su función.
- Una rutina etiquetada “label” se puede invocar en cualquier parte del código, mediante la instrucción CALL “label”.
- La última instrucción de una rutina debe ser siempre RET. Cuando se ejecuta esta instrucción, el programa continúa por la instrucción siguiente a la llamada a la rutina (CALL).
- La llamada a la rutina puede ser condicional: CALL condición, “label” ; donde condición es cualquiera de las condiciones válidas para la instrucción de salto JP.
- Análogamente, la instrucción RET también puede ser condicional: RET condición ; donde condición es cualquiera de las condiciones válidas para la instrucción de salto JP.

Ejercicio 8

- Escribir un programa en ensamblador del Z80 que:
  - Almacene un valor de una cifra en el registro C.
  - Almacene un valor de una cifra en el registro D.
  - Compare ambos valores y deje en una dirección de memoria previamente reservada una cifra que indique el resultado de la comparación (1: C > D; 0: C = D; -1: D > C).
- Reescribir el programa para que efectúe 5 comparaciones consecutivas con 5 cifras diferentes, almacenando los resultados en 5 posiciones consecutivas de memoria.

Solución

```
org AppFirst

AppEntry    ld b, 5
            ld ix, num1
bucle       ld c, (ix)
            ld d, (ix + 5)
            call compara
            ld (ix + 10), a
            inc ix
            djnz bucle
            jp fin

compara     ld a, d
            cp c
            jp z, iguales
            jp nc, mayor
            ld a, -1
            ret
```

```
iguales     ld a, 0
            ret
mayor       ld a, 1
            ret

fin         halt
            jp fin          ;

num1          defb 1,1,2,2,3
num2          defb 3,3,2,2,1
resul         defs 5
```

Ensamblador Z80. Pila

- La pila en Z80 es accesible por pares de registros de 1 byte o por registros de 2 bytes.
- PUSH RR es la instrucción para almacenar en la pila el contenido de RR.
- POP RR es la instrucción para extraer el valor contenido en la cima de la pila y almacenarlo en RR.
- RR puede ser: BC, DE, HL, AF, IX e IY.
- La pila se suele utilizar como método ágil de guardado temporal de los valores de un par de registros.
- También es útil para invertir el orden de una cadena.

Ejercicio 9

- Escribir un programa en ensamblador del Z80 que:
  - Almacene una cadena de caracteres ASCII de tamaño par.
  - Copie la cadena en otro lugar de la memoria, pero invertida. Se hará uso de la pila para llevar a cabo la operación.

Solución

```
org AppFirst

AppEntry    ld b, 4
            ld hl, cadena
bucle       ld a, (hl)
            push af
            inc hl
            djnz bucle
            ld b, 4
            ld hl, invertida
bucle2      pop af
            ld (hl), a
            inc hl
            djnz bucle2

fin         halt
            jp fin          ;

cadena      defm "hola"
invertida   defs 4
```

Ensamblador Z80. CPI

- A veces es necesario encontrar un cierto valor dentro de una cadena de valores.
- La instrucción **CPI** automatiza este proceso, comparando el byte almacenado en el acumulador con el byte almacenado en la dirección de memoria contenida en HL.
- Al ejecutarse CPI, se realizan tres operaciones:
  - A – (HL) ; en A se almacena el **octeto buscado**
  - HL + 1 ; HL sirve de **puntero** para recorrer todas las posiciones de la cadena
  - BC – 1 ; BC sirve como **contador** (byte counter) del número de bytes comparados
- El flag Z = 1 si A = (HL). El flag P/V = 1 mientras BC – 1 no sea 0.
- La instrucción **CPIR** repite este proceso de manera automática hasta que BC = 0.
- BC debe inicializarse con el tamaño de la cadena. Si BC = 0 antes de ejecutar la instrucción, el bucle se repite 64k veces o hasta que A = (HL).

Ejercicio

- Escribir un programa en ensamblador del Z80 que:
  - Almacene una secuencia de 5 números consecutivamente.
  - Busque un número concreto dentro de dicha secuencia.
  - Deje en A una cifra indicando si lo ha encontrado o si no (1: encontrado; 0: no encontrado).

Solución

```
 org AppFirst

AppEntry     ld bc, 5
             ld hl, cadena
             ld a, 4
             cpir
             jp z, encontrado
             ld a, 0
             jp fin
encontrado   ld a, 1

fin           halt
             jp fin

cadena            defb 1,2,3,4,5
```

Índice

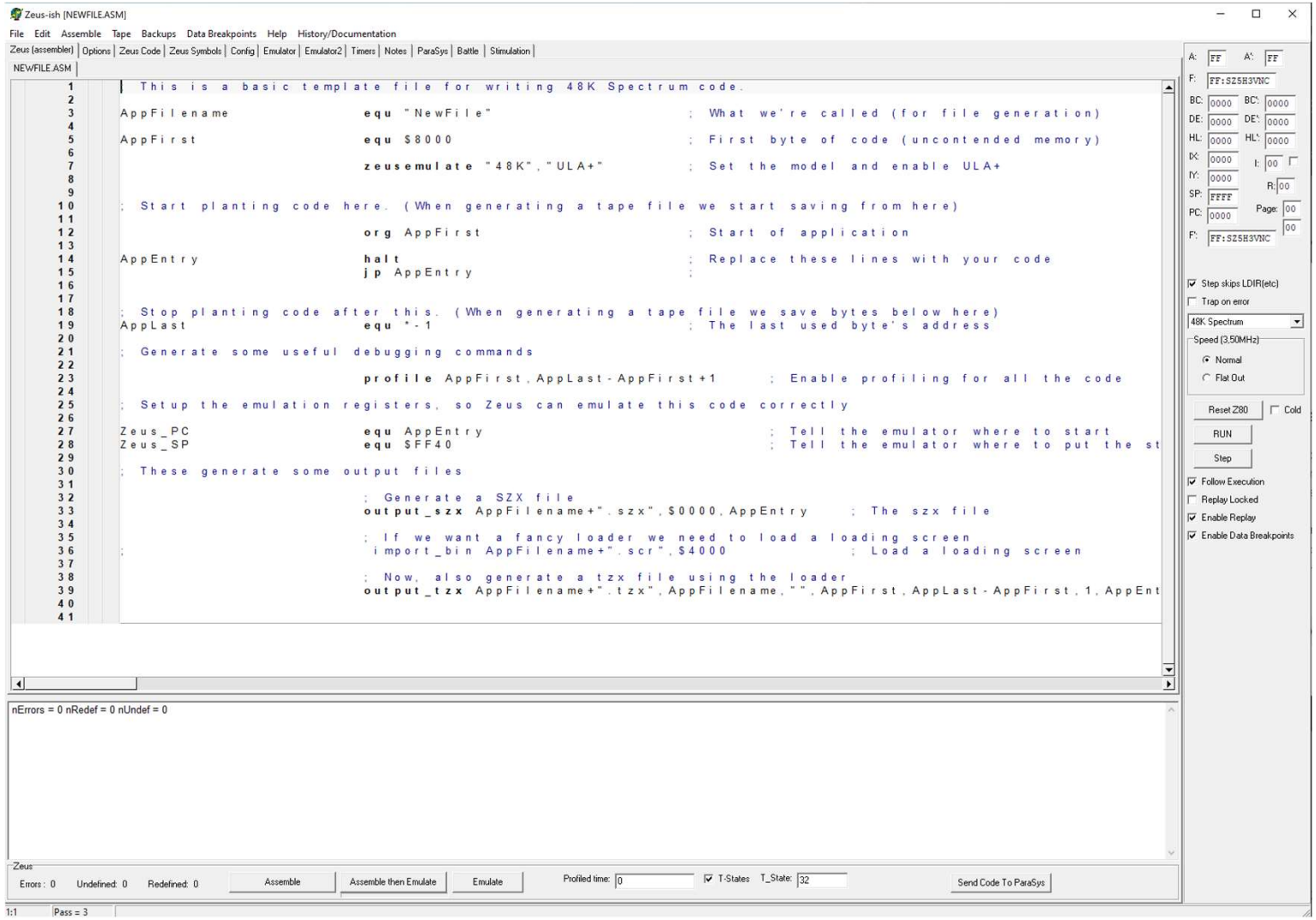
Ensamblador Z80. ¿Cómo funciona?

- Cuando ensambla un programa, el ensamblador realiza una primera pasada en la que analiza todo el programa (escrito en ensamblador). En esta pasada:
  - Encuentra todas las etiquetas definidas y crea la tabla de símbolos.
  - Verifica la sintaxis e identifica errores.
- Luego, efectúa una segunda pasada en la que analiza cada instrucción para reconocer:
  - su nemónico y, por lo tanto, la instrucción que se debe ejecutar.
  - los operandos, que pueden ser registros, números o etiquetas. Esto es posible dado que el ensamblador conoce los nombres de los registros, los dígitos, las letras D y H, los símbolos aritméticos y lógicos y los símbolos definidos en la tabla de símbolos.
- El ensamblador ignora todos los campos de comentarios.

Ensamblador ZEUS

- Zeus es un ensamblador para el procesador Z80 que fue desarrollado específicamente para el ordenador Sinclair ZX Spectrum.
  - Es un ensamblador de dos pasos que permite escribir programas en lenguaje ensamblador Z80 y ensamblarlos para su ejecución en el ZX Spectrum.
  - Zeus ofrece una interfaz de usuario fácil de usar que permite editar y ensamblar código en tiempo real, ver y modificar el contenido de la memoria del Spectrum, y ejecutar programas directamente en el emulador de Spectrum incorporado.
  - También incluye características avanzadas como la capacidad de importar y exportar programas y datos en varios formatos, y soporte para la depuración de programas con la ayuda de un emulador de depuración incorporado.

Aspecto ZEUS

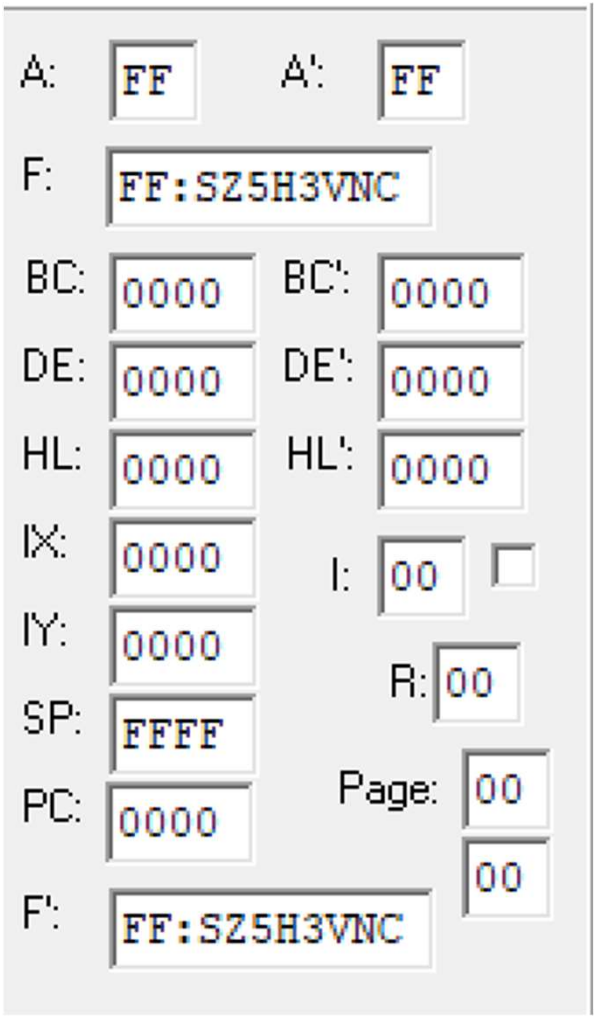


Pestañas ZEUS

- **Zeus(assembler):** Pestaña principal donde escribimos nuestro programa
- **Zeus Code:** Contiene una vista de la memoria.
- **Zeus Symbols:** Contiene las direcciones de los símbolos del sistema.
- **Emulator:** Nos permite emular en caos de tener algo que hacer por pantalla

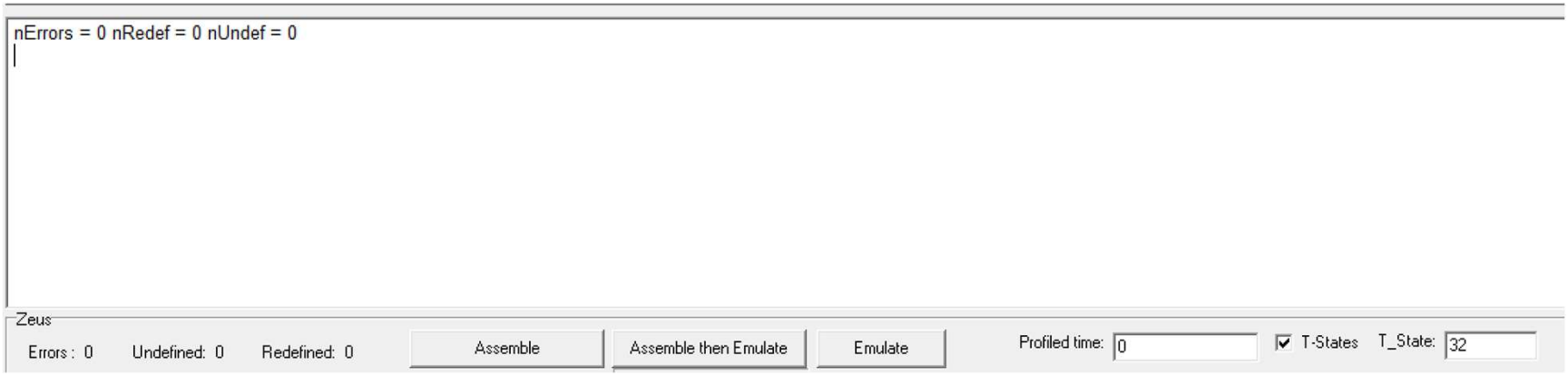
Registros ZEUS

- En la pestaña Zeus 8assembler), a la derecha.
- Podemos ver el valor de todos los registros
- Podemos estar pendientes de:
  - Cambio de valores
  - Los flags
  - El contador de programa
  - ...



Errores ZEUS

- En la misma pestaña, contamos con una consola que muestra los posibles errores, con un log y el posible origen de los problemas del ensamblador.
- Si el programa no ensambla, mirar aquí la presencia de errores
- Los botones de debajo sirven para ensamblar, y ensamblar y emular.



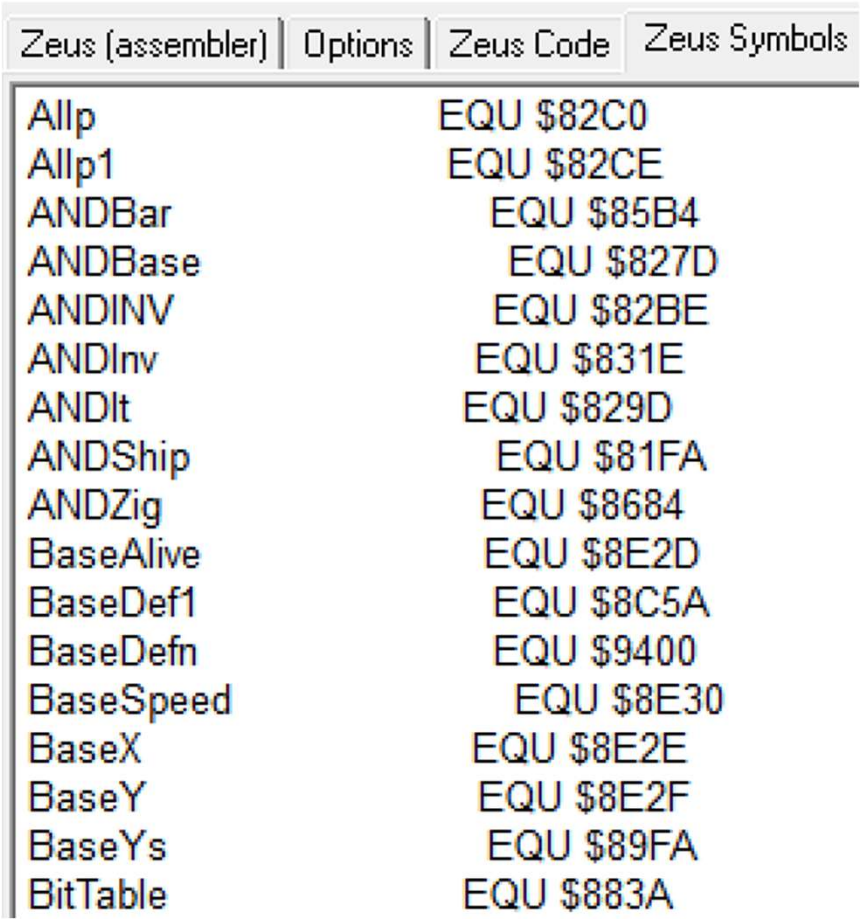
Memoria ZEUS

- En la pestaña Zeus code, podemos ver una imagen de la memoria.
- Aquí tenemos todo lo que hay en nuestro Spectrum.
- Se organiza según este esquema:

&0000	to	&3FFF	ROM
&4000	to	&57FF	Screen Memory
&5800	to	&5AFF	Screen Memory (Colour Data)
&5B00	to	&5BFF	Printer Buffer
&5C00	to	&5CBF	System Variables
&5CC0	to	&5CCA	Reserved
&5CCB	to	&FF57	Available Memory (between PROG and RAMTOP)
&FF58	to	&FFFF	Reserved

Simbolos ZEUS

- En la pestaña Zeus Symbols, encontramos las posiciones de memoria donde se guardan los símbolos del sistema (etiquetas).
- Estos estarán en las posiciones que nosotros hayamos detallado en el programa.
- Por ejemplo:
  - AppEntry
  - Inicio
  - Fin
  - ...



Emulador ZEUS