Capítulo I.

Programar en Ensamblador es más sencillo de lo que creemos. La dificultad al programar en código máquina se encuentra en que tenemos que indicar al procesador cada cosa que queremos que haga. Pero nuestro programa Ensamblador nos facilitará bastante las cosas.

Este es el primer ejemplo que vamos a analizar en este tutorial de Ensamblador. En él hay una serie de comandos que forman parte de las instrucciones del procesador y otra que forma parte del programa ensamblador. Este programa suma los números 3 y 5 y mete el resultado en la dirección de memoria &4040.

```
ORG &4000          ; Dirección de Inicio, instrucción para el Ensamblador
4000 LD A, &03     ; Mete 3 en el Acumulador
4002 ADD A, &05    ; Suma 5 al número que está en el Acumulador
4004 LD (&4040), A ; Mete el contenido del Acumulador en la dirección
                   &4040
4007 RET           ; "Retorna" a donde estaba
```

Empecemos: **ORG** es la instrucción del programa Ensamblador y le indica la dirección de memoria de inicio donde se va a generar el código: al decir ORG &4000 estamos diciéndole que el programa se ensambla (se traduce a los bytes que entiende el procesador) a partir de la dirección &4000.

El resto de operaciones que salen: LD, ADD y RET son instrucciones del procesador.

Recordemos que el código ASM es casi lo mismo que escribir el programa en código máquina pero un poco más fácil, usamos mnemónicos en lugar de los bytes que entiende el procesador. Estos mnemónicos los traduce el programa ensamblador. Hemos pensado que el MAXAM en ROM o el que viene integrado en el WinAPE son las mejores alternativas por claridad y facilidad de uso y nos referiremos a éstos a la hora de generar los listados de los programas.

Si utilizamos MAXAM en ROM, para introducir nuestro programa, arrancamos MAXAM escribiendo |m, seleccionamos ***T – Text Editor*** y ***E – Edit Text*** y simplemente tecleamos nuestro programa. Cuando hayamos terminado, con ESC volveremos al menú anterior y con ***A – Assemble Text***, ensamblaremos el código en memoria.

Si utilizamos el MAXAM integrado de WinAPE, entramos en el Assembler pulsando F6 o el botón correspondiente, elegimos ***New*** en el menú ***File*** para teclear nuestro listado y ***Assemble-Assemble*** para colocar el programa en la memoria.

Con la instrucción **LD A,&03**, le estamos pidiendo al Z80 que meta el valor &03 en el Acumulador, que es el registro de trabajo del Z80. El LD A (de LoaD A) es en realidad el mnemónico de la instrucción con el opcode &3E, que es el byte que el Ensamblador meterá en la memoria y que entenderá nuestro Z80. Vamos a olvidarnos de los opcodes, ya hablaremos de ellos más adelante.

ADD A,&05 indica que se sume el número &05 al que hay en el acumulador. El valor nuevo se guarda en el acumulador. Si lo ponemos en fórmula matemática, A=A+&05.

LD (&4040),A indica al Z80 que el número que tiene el acumulador lo guarde en una posición de memoria, en &4040.

En este caso, hemos utilizado la dirección &4040 como variable ***resultado***. En Ensamblador tenemos un par de métodos para definir variables:

```
resultado equ &4040  ; indicamos al programa ensamblador que siempre que encuentre la palabra
                     ; resultado en el código fuente, al ensamblar lo traduzca por &4040.
LD (resultado),A

resultado db &00     ; en este caso no sabemos cuál es la dirección de resultado, depende de la
                     ; posición de esta línea en el listado pero el programa ensamblador calcula
                     ; cual es el valor y lo usa para sustituir la palabra resultado al ensamblar.
LD (resultado),A
```

De esta manera, nos ahorramos el tener en cuenta en que dirección se guarda cada variable de nuestro programa, refiriéndonos a ella con su etiqueta.

RET, lo que hace es que retorne el programa a la última llamada (CALL). Es como un RETURN que retorna al GOSUB que lo llamó. Si es el final del programa, como en este caso, retornará al Basic.

Si introducimos nuestro programa en memoria y lo ejecutamos desde Basic con **CALL &4000**, veremos que aparentemente nada ha ocurrido. Pero si tecleamos **PRINT PEEK(&4040)** tendremos nuestro flamante resultado: **8**.

Nuestro Z80 tiene aparte del Acumulador otros 6 registros de 8 bits más, que se llaman B, C, D, E, H y L. La instrucción LD A, &03 se puede aplicar a cualquiera de los otros registros de 8 bits del procesador, o sea, que podemos introducir un byte en cualquiera de ellos. Pero no sucede lo mismo con el ADD o el LD (&4040),A. Estas instrucciones no son aplicables a los otros registros. Si tenemos que meter el contenido de, por ejemplo, B en una posición de memoria, no podemos hacer LD (&4040),B ya que esta instrucción no existe, tendríamos que usar el Acumulador como intermediario. Lo vemos en este otro ejemplo junto con alguna cosilla más:

```
ORG &4000          ; Dirección de Inicio, instrucción para el Ensamblador.
4000 LD B, &05     ; Mete 5 en el registro B.
4002 LD C, &03     ; Mete 3 en el registro C.
4004 LD A, B       ; COPIA el contenido de B en el Acumulador.
4005 ADD A, C      ; Suma el número del registro C al que está en el
                   Acumulador
4006 LD (&4040), A ; Mete el contenido del Acumulador en la dirección &4040
4009 RET           ; ”Retorna” a donde estaba.
```

Este programa tiene el mismo resultado que el anterior, pero dando un pequeño rodeo: "carga" los números a sumar en B y C y que así podamos ver la posibilidad de mover datos entre registros. Como la instrucción de suma de 8 bits ADD es sólo aplicable al acumulador, COPIAMOS el contenido de B en el Acumulador. Luego le sumamos el contenido del registro C: aquí vemos que no sólo podemos sumar al Acumulador el número que le indiquemos, sino también uno que esté contenido en otro registro.

Resultaba más sencillo el otro programa, pero éste ilustra bien el hecho de que podemos tener cargados los registros a modo de parámetros (hemos sumado el CONTENIDO de un registro). Este uso de parámetros afectará, por ejemplo, a las **Rutinas del Firmware**. Estas rutinas del Firmware nos serán de gran ayuda en estos comienzos, ya que nos ofrecen operaciones básicas ya hechas. Ejemplos claros de esto son la que se encuentra en &BB5A llamada **TXT OUTPUT**, que nos mostrará en pantalla el carácter cuyo código ASCII se encuentra en el Acumulador, o **TXT SET CURSOR** en &BB75 que posiciona el cursor de texto en el número de columna guardado en H y el de fila guardado en L. Vemos su uso en este ejemplo:

```
ORG &4000          ; Dirección de Inicio, instrucción para el Ensamblador.
4000 CALL &BB6C    ; Rutina del Firmware TXT CLEAR WINDOW. Borra la
                     pantalla de texto, lo que sería un CLS
4003 LD H, &14     ; Carga &14 en el registro H. Columna &14, 20 decimal.
4005 LD L, &0C     ; Carga &0C en el registro L. Fila &0C, 12 decimal.
4007 CALL &BB75    ; Rutina del Firmware TXT SET CURSOR, posiciona el
                     cursor de texto en la columna H y y fila L
400A LD A, &2A     ; Carga &2A en el Acumulador, ASCII de asterisco.
400C CALL &BB5A    ; Rutina del Firmware TXT OUTPUT, imprime en
                     pantalla el carácter cuyo ASCII está en el Acumulador.
400F RET           ; "Retorna" a donde estaba.
```

Hay instrucciones que nos permitirán modificar el contenido de un registro por ejemplo **INC**, que INCrementará el contenido del registro en una unidad o **DEC**, que DECrementará una unidad el número contenido en el registro en cuestión. De esto se podría deducir que si insertamos estas líneas antes del RET podremos pintar dos asteriscos:

```
400F INC L         ; INCrementa el registro L en una unidad, aumenta el
                     número de fila.
4010 CALL &BB75    ; Rutina del Firmware TXT SET CURSOR, posiciona el
                     cursor de texto en la columna H y y fila L
4013 CALL &BB5A    ; Rutina del Firmware TXT OUTPUT, imprime en
                     pantalla el carácter cuyo ASCII está en el Acumulador.
```

Probadlo. ¿Qué ha ocurrido? Resulta que el TXT SET CURSOR, corrompe el contenido del Acumulador, H y L, como hacen también otras rutinas. Para prevenir esto deberíamos haber movido su contenido a otros registros antes de llamar a la rutina y luego recuperarlo, recordad que disponemos también de B, C, D Y E y que sabemos mover contenidos de un registro a otro, así que será fácil modificar nuestro programa. ¿Quién se anima?

Repasamos:

ORG NN	; Dirección de Inicio, instrucción para el Ensamblador.
CALL NN	; Llamada a una dirección, que puede ser una rutina del Firmware, es un GOSUB del que volvemos con
RET	; "Retorna" a donde estaba.
LD R, N	; Carga el byte N en el registro R, que puede ser A, B, C, D, E, H o L
LD R, R'	; Copia el contenido del registro R en el registro R'.
LD A, (NN)	; Mete el contenido del Acumulador en la dirección NN.
LD (NN), A	; Carga el contenido de la dirección NN en el Acumulador.
ADD A, N	; Suma el byte &N al que hay en el Acumulador.
ADD A, R	; Suma el byte contenido en el registro R al que hay en el Acumulador. El resultado se queda en el Acumulador.
INC R	; INCrementa el byte contenido en R en una unidad.
DEC R	; DECrementa el byte contenido en R en una unidad.