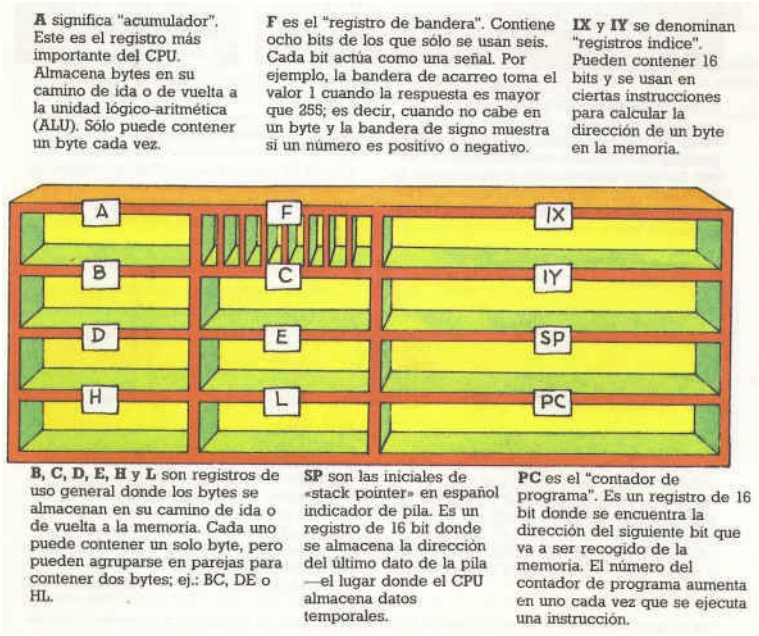
Capítulo 0. Introducción.

Bueno, por fin está aquí el esperado (espero !!) Capítulo Cero del Tutorial de Assembler.

En él estableceremos algunos conceptos que nos serán de utilidad a lo largo del Tutorial. Antes que nada, vamos a presentar al protagonista de este asunto: Nuestro **Z80**:



A él es a quien nos dirigiremos sin intermediarios (léase Basic) en la programación en Código Máquina. Nos iremos refiriendo a lo largo de todo el tutorial a lo que muestra esta foto, así que ya podéis ir enmarcando una copia…

En este primer acercamiento, vamos a fijarnos en la parte de la izquierda… Esos misteriosos cajones vacíos (se llaman registros) es todo con lo que contamos para hacer nuestros programas. Y por ellos irán pasando desde la memoria los datos que manejarán nuestros programas de camino a la pantalla (que en realidad es una zona de la memoria). En definitiva, lo ÚNICO que hace nuestro ordenador es mover datos de una parte de la memoria a otra, pasando por el Z80, poniendo en la parte de memoria que corresponde a la pantalla lo que se tiene que mostrar. Bueno, con sólo mover datos no parece que se pueda conseguir gran cosa, así que alguna cosilla más si que hace: pequeñas operaciones o procesos con los datos (por eso se llama procesador), y esto es lo que vamos a tratar…

Yo creo que a estas alturas todo el mundo que lea esto sabe que nuestro CPC es un ordenador de 8 bits: pues bien, resulta que en cada uno de esos cajones-registros (y en cada pequeña porción de la memoria) caben precisamente 8 bits. ¿Y que es un bit? Ya sabéis, lo de los unos y ceros, todos hemos oído hablar alguna vez hablar de los unos y ceros y todo eso: el **sistema binario**. O sea que el rollo va solo de números.

Vamos a hablar primero un poco de estos y otros números. No os asustéis ya los de Letras que esto es muy sencillo: el sistema binario es la manera que tienen de utilizar los números (y toda información que manejan) los aparatos electrónicos: si hay corriente, es un 1 y si no la hay, es un 0. ¿Y cómo cuenta así? Pues muy sencillo, empieza con el 0, luego el 1… se le han acabado los números. Nosotros, cuando se nos acaban los números le cascamos un 1 delante para seguir contando y empezamos otra vez (después del 9, anteponemos un 1 y empezamos desde el principio, esto es, el 10, el 11, el 12…). Aquí lo mismo. Sería 0, 1, 10, 11, 100, 101, 110, 111, 1000….

Entonces, para que nuestro ordenador nos entienda, tenemos que decirle todo lo que queremos que haga introduciéndole paquetes de 8 bits mediante este sistema, por ejemplo 01100101, ya que es la única información que él maneja. Y esto es totalmente cierto !!

Pero lo podemos hacer más sencillo. Partamos nuestro paquete de 8 bits en dos mitades iguales (o sea en dos paquetes de cuatro bits o nibbles). Con un paquete de cuatro bits se pueden hacer 16 combinaciones distintas, que podéis ver en la tabla de más abajo. Estas 16 combinaciones nos vienen al pelo ya que se corresponden con el número de dígitos que tiene el **sistema hexadecimal**. ¿Otra cosa nueva?, la cosa se complica… Pues no !!

Sigue siendo valida la explicación para el binario, lo único que nuestros números base en vez de ser el 0 y el 1 en el binario o del 0 al 9 en el decimal que usamos en la vida real, tiene más, 16 para ser exactos: utiliza del 0 al 9 y sigue con A, B, C, D, E y F.
O sea, que contando, sería… 8, 9, A, B, C, D, E, F, 10, 11, …,19, 1A, 1B, … , FF, 100, 101, …

Entonces, siguiendo con lo que estaba diciendo, cada combinación de 4 bits se corresponde con un dígito hexadecimal de la siguiente manera:

Binario	Hexadecimal	Decimal
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15
10000	10	16
1101011	6B	107
10101010	AA	170

En la tabla se puede apreciar que nuestros 8 bits se corresponden con sólo dos dígitos hexadecimales (podemos poner ceros a la izquierda hasta alcanzar los 8 bits). Siempre será más fácil decirle al Z80 9-C que 1001-1100 !!! Esta tabla nos puede valer como referencia para pasar un número de un sistema a otro, aunque no es difícil: vamos a poner la primera norma: para no liarnos, a un número binario le pondremos delante un % y a un hexadecimal un & para no confundirlos con números en sistema decimal.

Podemos **pasar un número en binario o en hexadecimal a decimal** simplemente sumando sus múltiplos de las potencias que tienen como base "la base" de su sistema, esto es, al igual que el decimal 3705 se entiende también como la suma de

$$5x10^0 + 0x10^1 + 7x10^2 + 3x10^3,$$

el binario %10010110 es en decimal la suma de

$$0x2^0 + 1x2^1 + 1x2^2 + 0x2^3 + 1x2^4 + 0x2^5 + 0x2^6 + 1x2^7 =$$
$$0x1 + 1x2 + 1x4 + 0x8 + 1x16 + 0x32 + 0x64 + 1x128 = \mathbf{150}$$

y el hexadecimal &4F7A es en decimal

$$10x16^0 + 7x16^1 + 15x16^2 + 4x16^3 =$$
$$10x1 + 7x16 + 15x256 + 4x4096 = \mathbf{20346}$$

También podemos **pasar de decimal a binario o hexadecimal** simplemente dividiendo el número que queremos pasar por la base del sistema-destino (entre 2 para binario y entre 16 para hexadecimal) y los restos y el ultimo cociente nos irán dando los dígitos que buscamos de menor a mayor.

```
115 | 2
 15  57 | 2
 1   17  28 | 2
      1  08  14 | 2
      0   0   7 | 2
              1   3 | 2
                  1  1
```

Con lo que 115=%1110011

```
17786 |16
17     1111 | 16
 18     151   69 | 16
  26      7    5    4
  10
```

Así que, 17786=&457A

Más sencillo resulta movernos entre binario y hexadecimal, ya que cada dígito hexadecimal se corresponde con 4 bits, como vimos en la tabla de más arriba, y no tenemos más que sustituir unos con otros para pasar de un sistema a otro…

Pero de todos modos, ¿para qué querría yo decirle al Z80 &C9, &5F ó &36? Resulta que estos números el Z80 los va a interpretar como instrucciones. Sabemos ya que en Basic, las instrucciones de nuestros programas van precedidas de un número de línea que indica al ordenador donde empieza el programa y va ejecutando las instrucciones en ese riguroso orden de menor a mayor número de línea, sin saltarse ni una, mientras que no le especifiquemos otra cosa en alguna instrucción, como hace por ejemplo la archiconocida GOTO.

Aquí sucede lo mismo pero empleando las pequeñas porciones o **posiciones de la memoria** como números de línea, ya que estas posiciones están numeradas para diferenciar unas de otras. Esta numeración es lo que llamaremos dirección de memoria. Seguro que os habéis fijado que en la foto de nuestro Z80, la zona de la derecha tiene los registros más largos. Esto es porque esa parte es para trabajar con direcciones y esos registros son de 16 bits, pero de momento no hablaremos de eso, basta saber ahora que las direcciones de memoria tienen 16 bits. Se numeran de 0 a 65535 (65536 posiciones contando la posición cero) si lo decimos en sistema decimal, &0 a &FFFF si lo decimos en hexadecimal, y %0 a %1111111111111111 (16 unos) si lo decimos en binario, ya que es la cifra más alta que podemos alcanzar con 16 bits y ya hemos dicho que el binario es el que marca la pauta a seguir.

El registro de la zona derecha marcado como **PC es el Contador de Programa** y en su interior se encuentra la dirección de memoria que se ejecutará a continuación. En esa dirección de memoria estará el primer paquete de ocho bits de nuestro programa. Por cierto, no lo mencionado hasta ahora: a cada uno de estos paquetes se les llama BYTE y así los llamaremos a partir de ahora. El Z80 leerá lo que hay en esa dirección que está en el Contador de Programa, por ejemplo en la &4000, luego leerá la &4001 y así sucesivamente.

Vamos a poner un primer ejemplo ya, que veo muchas bocas abiertas, presentando primero otro registro, esta vez de 8 bits llamado A (el de arriba a la izquierda en la foto): se llama Acumulador y es el registro más importante de nuestro Z80 (o casi) ya que sobre él se harán la mayoría de operaciones con los datos. Haremos un sencillo programa que sumará 3 y 5 y pondrá el resultado en una dirección de memoria. La instrucción Basic POKE, introduce en la dirección de memoria que le indiquemos el valor que pongamos tras la coma (esto ya casi los sabíamos, como buenos jugones que somos). Así que encenderemos el CPC y escribiremos:

```
POKE &4000, &3E
POKE &4001, &03
POKE &4002, &C6
POKE &4003, &05
POKE &4004, &32
POKE &4005, &40
POKE &4006, &40
POKE &4007, &C9
```

Vemos que lo primero que se encontrará el Z80 cuando ejecute nuestro programa es "&3E". Esto es el **código de** una **operación** (llamado también **opcode**) que el Z80 reconoce y que le pide que meta el byte que se encuentre a continuación en el Acumulador. Y eso hace: carga el Acumulador con &03.

Lo siguiente que se encuentra es el &C6, otro opcode que le pide que sume el byte que se encuentra a continuación al que está en el Acumulador, así que le suma el &05 al &03, teniendo ahora en el Acumulador &08.

El siguiente opcode, &32, significa: mete lo que tenga ahora mismo el Acumulador en la dirección que se encuentra a continuación, esto es en &4040, (recordemos que las direcciones de memoria requieren dos bytes).

El último opcode &C9 le indica que es el "fin del programa" y que vuelva a lo que hacía antes de empezar, esto es al Ready del Basic.

Muy bien, tenemos el programa en memoria, ¿pero cómo lo ejecutamos? Fácil. Llamamos a la posición de memoria de la primera "instrucción" de nuestro programa: escribimos **CALL &4000**

Vale. ¿Qué ha pasado? No vemos nuestro resultado &08 por ningún lado, pero mirad el lado bueno, por lo menos el CPC no se ha colgado, ni ha dado error, ni nada parecido, seguro que más de uno habéis tenido alguna desagradable experiencia con el CALL.

Nuestro resultado está en la dirección &4040, como le dijimos al Z80 que hiciera. Para los incrédulos, escribid PRINT PEEK(&4040), y veréis que responde el CPC… PEEK es otra función del Basic que nos permite ver lo que hay en la dirección de memoria que le indiquemos.

Fijaos la que hemos montado para sumar un par de números, con todos esos pokes y tal y esos extraños opcodes, por lo menos, nos hará falta tener una tabla al lado en la que ir consultando si para meter un número en el Acumulador había que poner &3E, ¿o era &E3? Nada de eso !!

Aquí es donde aparece por primera vez el **programa Ensamblador** que es el que nos librará de todos esos problemas. Para empezar, las instrucciones se las iremos dando con **mnemónicos**, esto es, en vez de escribir el opcode de cada instrucción, escribiremos una palabra que además nos recordará su funcionamiento, y él se encargará de traducírselas al Z80. Por ejemplo, para la primera instrucción de nuestro programa, &3E, que mete el byte que se encuentre a continuación en el Acumulador se escribirá LD A (de LoaD A). Nuestro programa en ensamblador quedaría

LD A, &03
ADD A, &05
LD (&4040) A
RET

Bastante más claro, ¿no?