Capítulo II. Más sobre Registros y Operaciones.

En el Capítulo anterior aprendimos a sumar, a mover datos entre registros y modificar su valor INCrementándolo o DECrementándolo. En éste veremos más operaciones que modificarán los datos contenidos en los registros, empezando por la….

RESTA. Es muy simple, tenemos una instrucción para restar, cuyo mnemónico es **SUB** y su ortografía es distinta a la de la suma: en este caso, ponemos a continuación de la instrucción el valor o registro que queremos restar del Acumulador. Esta resta es aplicable únicamente al Acumulador, por ejemplo SUB B o SUB &3F, restarán el byte contenido en el registro B o &3F respectivamente, del byte contenido en el Acumulador, guardándose en éste el resultado.

Como en nuestros registros y posiciones de memoria sólo cabe un Byte, los números que podemos utilizar en nuestras operaciones han de estar entre 0 y 255 decimal, &FF en hexadecimal, %11111111 en binario. Pero, ¿qué ocurre si el resultado de nuestra suma o resta se sale de este rango por arriba o por abajo?

Me gustó mucho la analogía que leí en algún sitio con el cuentakilómetros "antiguo" del coche: pasas por 7 - 8 - 9 - 0 Y TE LLEVAS UNA (sumas una al dígito siguiente), y sigues con 1 - 2 - 3… Y para la resta lo mismo: 2 - 1 - 0 – 9 - 8 – 7… Aquí también "te llevarías" una, pero tendrías que *restarla* del dígito superior… Nuestro Z80 también funciona así, sólo que la que nos llevamos ya sea para sumarla o restarla lo único que hace es activar un indicador, el **CARRY**, que está ahí para saber que se ha rebasado "la barrera" entre el cero y el 255, es decir, que ha habido *acarreo*. Por ejemplo:

```
LD A, &04
SUB &07
RET
```

Tras ejecutarse este programilla, tendríamos en el Acumulador…¡¡ exacto !!, lo habéis pillado: &FD, o sea 253 en decimal. Para los que no, recordad que

```
  4 -  3 -  2 -  1 -  0 - 255 - 254 - 253
&04 - &03 - &02 - &01 - &00 - &FF - &FE - &FD
```

¡Eh! Y el CARRY activado. Siempre que se hace un ADD o un SUB el Carry se activará o desactivará convenientemente según haya habido o no acarreo. Esto no sucede con las instrucciones **INC o DEC,** estas instrucciones **NO afectan al Carry**.

El Carry nos permite utilizar instrucciones que bifurcarán el programa dependiendo de si está activado o no, por ejemplo las de salto **JP**. Conoceremos las instrucciones de salto con este programa que nos muestra el juego ASCII de caracteres:

```
ORG &4000          ; Dirección de inicio, instrucción para el ensamblador.
4000 LD A, &20     ; Carga el Acumulador con &20, ASCII de Espacio.
4002 CALL &BB5A    ; Rutina del Firmware TXT OUTPUT, imprime en
                     pantalla el caracter cuyo ASCII está en Acumulador.
4005 ADD A,1       ; Suma uno al número contenido en el Acumulador.
4007 JP NC, &4002  ; Si No hay Carry salta (JumP) a la dirección &4002.
400A RET           ; "RETorna" a donde estaba.
```

En este programa la instrucción **JP NC, &4002** nos va mandando de nuevo al CALL y sumando uno al Acumulador mientras que no haya acarreo. El acarreo se producirá al sumar 1 a 255, la condición no cumplirá y seguirá el programa en la siguiente instrucción, el RET. Con INC A, el programa no funcionaría al no afectar al Carry.

Como este programa ya "hace algo" nos podría interesar guardarlo: en el MAXAM en ROM podemos guardar el Código Fuente (el listado en Ensamblador) con la opción S del menú, luego podremos recuperarle con L**.** Si estamos usando el MAXAM de WinAPE, File/Save nos guardará el Código Fuente.

Si lo que queremos es guardar el archivo ya ensamblado (en binario) volvemos a BASIC y escribimos:

save "nombre", B, &4000, N

donde &4000 sería la dirección de inicio de nuestro programa y N el número de bytes de nuestro programa. Otra forma de hacerlo es incluyendo la linea **write "nombre"** en el código fuente. Al ensamblar, el MAXAM guardará una copia en disco.

Otro modo de usar JP sería saltando si sí hubiera Carry (JP C, &NN) o incluso sin condición, que sería un salto incondicional, similar a la instrucción BASIC *GOTO*.

Hay otra instrucción que también afecta al Carry y que opera igual que la resta, pero sin cambiar el byte contenido en el acumulador, digamos que simula la resta pero no guarda el resultado en ningún sitio: es **CP** (de ComParar).

Fijaos en que en una resta, si el número que restamos es mayor que el del Acumulador, nos activará el Carry pero si es menor no. Si son iguales, no nos activa el Carry (no hemos rebasado el "límite"), pero sí nos activa el **Zero**, otro indicador que se activa cada vez que el resultado de una operación sea cero.

Entonces CP lo que hace es restar lo que le indiquemos del byte contenido en el acumulador, y nos colocará los indicadores de Carry y Zero convenientemente pero no alterará el contenido del Acumulador (ni del otro registro si fuera el caso).

El indicador Zero nos permite aumentar nuestras instrucciones de salto condicional con una nueva condición: la dependencia del Zero. Lo vemos en este ejemplo con bucles anidados que dibuja un triángulo de asteriscos:

```
ORG &4000          ; Dirección de inicio, instrucción para el ensamblador.
4000 LD C, &08     ; Carga el registro C con &08.
4002 LD B, C       ; Copia el contenido de C en B.
4003 LD A, &2A     ; Carga el Acumulador con &2A, ASCII de asterisco.
4005 CALL &BB5A    ; Rutina del Firmware TXT OUTPUT, imprime en
                   pantalla el caracter cuyo ASCII está en el
                   Acumulador.
4008 DEC B         ; Decrementa el número contenido en B.
4009 JP NZ, &4005  ; Si No hay Zero salta (JumP) a la dirección &4005
400C LD A, &0A     ; Carga el Acumulador con &0A. Salto de línea
400E CALL &BB5A    ; Rutina del Firmware TXT OUTPUT, imprime en
                   pantalla el caracter cuyo ASCII está en el
                   Acumulador.
4011 LD A, &0D     ; Carga el Acumulador con &0D. Inicio de línea.
4013 CALL &BB5A    ; Rutina del Firmware TXT OUTPUT, imprime en
                   pantalla el caracter cuyo ASCII está en el
                   Acumulador.
4016 DEC C         ; Decrementa el número contenido en C.
4017 JP NZ, &4002  ; Si No hay Zero salta (JumP) a la dirección &4002.
401A RET           ; "RETorna" a donde estaba.
```

Señalaremos aquí que las JP son saltos absolutos, ya que existe también un juego de instrucciones de salto relativo, **JR**, en las que en vez de dar una dirección a donde saltar lo que hacemos es saltar varios bytes hacia delante o hacia atrás en la memoria indicándole solamente el número de bytes que queremos que varíe el **Contador de Programa PC** (¿recordáis?, es ese registro de 16 bits que "lleva la cuenta" de qué posición de memoria contiene la siguiente instrucción del programa, lo vimos en la Intro). Este número de bytes se llama *desplazamiento*, y puede ser positivo o negativo (para saltar adelante o atrás).

Como hemos visto que nuestro Z80 cuenta ciclicamente, ignorando los números negativos, hemos de adoptar aquí un convenio: cuando en algunos casos, como en este del desplazamiento, necesitemos números negativos (para poder saltar atrás) usaremos un sistema de numeración de 7 bits, estando el octavo bit destinado al signo y que estará a cero en los positivos y a uno en los negativos, quedando de la siguiente manera:

-128, -127, …….. -3, -2, -1, 0, 1, 2, 3, …… , 126, 127
&80, &81, …….… , &FD, &FE, &FF, &00, &01, &02, &03, …… , &7E, &7F
%10000000, ……, %11111111, %0, %1, ……., %01111111

Pero tenemos una pega a la hora de indicar un desplazamiento: el negativo de un número NO es el mismo número positivo con su octavo bit puesto a uno, y si no haced la prueba. Aunque lo podemos calcular facilmente, ya que corresponde con el **complemento a dos** de su positivo en binario. ¿Y cómo se hace eso? Es muy sencillo: tomamos el número positivo en binario y lo complementamos a dos, es decir, ponemos los bits que eran cero a uno y los que eran uno a cero, y después le sumamos uno. Lo vemos con el -6, sacado desde el 6 positivo:

```
6 =  %   0 0 0 0 0 1 1 0      ; ponemos el número positivo en binario
     %   1 1 1 1 1 0 0 1      ; complementamos a 1 (cambiamos 1 por 0)
                   +   1      ; sumamos 1 para complementarlo a 2
-6 = %   1 1 1 1 1 0 1 0      ; Nuestro -6, esto es &FA
```

Los cálculos de los desplazamientos nos los hará automáticamente el Ensamblador gracias a las etiquetas, pero creo que es interesante y útil conocer esta información…

Las instrucciones de salto relativo ocupan un byte menos que las de salto absoluto, aunque estas últimas son más rápidas. Por ahorro de espacio, podemos usar las de salto relativo siempre que nos sea posible (tienen un rango de utilización limitado a 127 y -128 bytes por delante y por detrás), pero si nos sobra espacio merece la pena usar las de salto absoluto por su rapidez.

Y para rematar los bucles y desplazamientos, hay una instrucción especial del Z80 destinada a la creación de bucles: **DJNZ d** (DecreaseB and Jump if Not Zero), que utiliza el registro B como un contador y le va restando uno y saltando el desplazamiento indicado (d) hasta que B llega a cero, momento en que continuará con la siguiente instrucción.

Los indicadores de Carry y el Zero forman parte de un registro especial de 8 bits del Z80, llamado F, de **Flags** o Banderas (no Antonio el de la Griffith, si no las de los barcos, ya sabeis de las de hacer señales). De sus 8 bits, sólo se usan 6, y son:

S	**Z**	**–**	**H**	**-**	**P/V**	**N**	**C.**
Signo	Zero	X	Half Carry	X	Paridad/ Desbordamiento	Suma/ Resta	Carry

Signo = Se pone a 1 si el bit más significativo de ciertas operaciones es uno.
Zero = Se pone a 1 si el resultado una operación es nulo (cero).
Half Carry = Como el Carry pero para las operaciones con medio byte, se pone a 1 si hay acarreo del bit 3 al 4.
Parid/Desb. = Será uno u otro según la instrucción: tras una operación lógica o de rotación/desplazamiento, P igual a 1 si hay un número par de bits puestos a 1; tras una operación aritmética, V igual a uno si hay desbordamiento.
Suma/Resta = Se pone a 1 si la operación precedente fue una resta y a 0 si fue suma.
Carry = Se pone a 1 si el resultado de una operación produce acarreo tanto si se trata de suma como de resta.

Estos indicadores o flags, menos H y N que no pueden verificarse, nos proporcionan más condiciones para nuestras instrucciones:

C = Hay acarreo C=1
Z = Igual Z=1
M = Negativo S=1
PO = Paridad impar P/V=1

NC = No hay acarreo C=0
NZ = No igual Z=0
P = Positivo S=0
PE = Paridad par P/V=0

Los saltos relativos JR sólo admiten las condiciones con Carry y Zero, mientras que los saltos absolutos JP admiten todas. También admiten todas las condiciones las instrucciones **CALL** y **RET**, que ya conocíamos en su versión incondicional.

Otro tipo de operaciones que podemos hacer con los registros son las clásicas **operaciones lógicas** siguientes que expongo junto a su tabla de verdad:

Bit A	Bit B	**AND**	**OR**	**XOR**
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Y que serán efectuadas bit a bit entre el Acumulador y el Operando, esto es el bit 0 del Acumulador con el bit 0 del Operando, el bit 1 del Acumulador con el bit 1 del Operando, … , bit 7 del Acumulador con el bit 7 del Operando, quedando el resultado en el Acumulador.

Estas instrucciones modifican convenientemente los flags S, Z y P/V (que en este caso actúa como bit de paridad, poniéndose a uno si queda un número par de unos en el Acumulador), ponen N y C a cero, y OR y XOR ponen H a cero y AND a uno.

Y queda una serie de instrucciones que ya que hablamos de bits voy a citar:

CPL Complementa a uno un registro, es decir, le cambia los ceros por unos y viceversa. Equivalente a la operación lógica NOT (negación).

NEG Cambia el contenido de un registro por su negativo, o sea por su complemento a dos. Recordemos que para obtener un número "negativo" desde su positivo, lo que hacemos es sumarle uno a su complemento a uno.

DAA Efectúa un ajuste decimal del Acumulador, traduce su contenido a BCD (binario codificado como decimal). De momento, baste saber que existe esta instrucción.

BIT b,r Esta instrucción comprueba el bit b del registro r, poniendo el flag Zero a 1 si contiene un cero y a cero si no (si es un 1).

SET b,r y **RES b,r** Ponen a uno (SET) o a cero (RES) el bit b del registro r. Estas dos instrucciones no afectan a los flags.

SCF Pone el Carry a 1 y **CCF** Complementa el Carry (si es cero lo pone a uno y viceversa). Podemos poner el Carry a cero con OR A o AND A, pero recordemos que también se modifican otros flags.

Con las instrucciones que conocemos podemos sumar y restar números de 16 bits también, sumando/restando por separado sus bytes de bajo peso y de alto peso (por ejemplo, en &C14F, el byte de bajo peso es 4F y C1 el de alto) y recordando añadir/restar uno al byte alto si la suma/resta de los bytes de poco peso ha provocado acarreo, o sea que nos llevamos una. Para ello tendríamos que comprobar el Carry cada vez que hiciéramos una suma… pero no, ya que tenemos unas instrucciones específicas para esta tarea, que son **ADC** y **SBC** que lo que hacen es añadir o restar uno respectivamente a la suma que estamos efectuando si el Carry está activado. Pero en el próximo Capítulo veremos mejores formas de de trabajar con números de 16 bits…

Para cerrar este capítulo, en este ejemplo creamos una rutina que escribirá la palabra que introduzcamos con la primera letra en mayúscula y el resto en minúsculas, independientemente del estado de BLOQ MAYS y MAYS. Aprovecha una característica del sistema ASCII: las letras mayúsculas y minúsculas vistas en binario se diferencian en que su bit número 5 está a CERO en las mayúsculas.

En el Manual de Usuario del CPC podéis ver que las mayúsculas van de 65 a 90 y las minúsculas de 97 a 122. El programa transforma a mayúsculas (en la etiqueta *.inicial*) y minúsculas (en *.resto*) las letras que estén dentro de este rango (mediante SET y RES para activar o no el bit 5), volviendo a esperar una pulsación si el ASCII asociado a la tecla pulsada no está en este rango. Esto lo va comprobando con los bloques de CP y JP.

```
ORG &4000          ; Dirección de Inicio, instrucción para el Ensamblador.

LD A,1             ; Carga el Acumulador con 1 decimal. Forzamos Mode 1.
CALL &BC0E         ; Rutina del Firmware SCR SET MODE, establece el
                   modo que esté en el Acumulador (0, 1, 2).
LD H,&05           ; Carga H con &05
LD L,&07           ; Carga L con &07
CALL &BB75         ; Rutina del Firmware TXT SET CURSOR, posiciona el
                   cursor de texto en la columna H y y fila L

.inicial
CALL &BB18         ; Rutina del Firmware KM WAIT KEY, espera la
                   pulsación de una tecla y pone su ASCII en el Acumulador.
CP 65              ; Compara (resta) 65 del número que hay en acumulador.
JP C,inicial       ; Si hubo Carry (ASCII<65) vuelve a esperar una tecla.
CP 123             ; Compara (resta) 123 del número que hay en acumulador.
JP NC,inicial      ; Si no hubo Carry(ASCII> ó =123) vuelve a esperar tecla.
CP 97              ; Compara (resta) 97 del número que hay en acumulador.
JR NC,sigue        ; Si no hubo Carry (ASCII> ó =97) continúa en sigue.
CP 91              ; Compara (resta) 91 del número que hay en acumulador.
JP NC,inicial      ; Si no hubo Carry (ASCII> ó =91) vuelve a esperar tecla.

.sigue             ;
RES 5,A            ; Pone a 0 el bit 5 del Acumulador. Pone en mayúscula.
CALL &BB5A         ; Rutina del Firmware TXT OUTPUT, imprime en
                   pantalla el caracter cuyo ASCII está en el Acumulador.

.resto             ;
LD B,8             ; Carga B con 8. Nuestro nombre tendrá máximo 9 letras.

.bucle             ;
CALL &BB18         ; Rutina del Firmware KM WAIT KEY, espera la
                   pulsación de una tecla y pone su ASCII en el Acumulador.
```

```
CP 13           ; Compara 13 con el Acumulador. ASCII de Return.
RET Z           ; Si se había pulsado Return (Z=1),  RETorna.
CP 65           ; Compara (resta) 65 del número que hay en acumulador.
JP C,bucle      ; Si hubo Carry (ASCII<65) vuelve a esperar una tecla.
CP 123          ; Compara (resta) 123 del número que hay en acumulador.
JP NC,bucle     ; Si no hubo Carry(ASCII> ó =123) vuelve a esperar tecla.
CP 97           ; Compara (resta) 97 del número que hay en acumulador.
JR NC,sigue2    ; Si no hubo Carry (ASCII> ó =97) vuelve a esperar tecla.
CP 91           ; Compara (resta) 91 del número que hay en acumulador
JP NC,bucle     ; Si no hubo Carry (ASCII> ó =91) vuelve a esperar tecla.

.sigue2         ;
SET 5,A         ; Pone a 1 el bit 5 del Acumulador.
CALL &BB5A      ; Rutina del Firmware TXT OUTPUT, imprime en
                  pantalla el carácter cuyo ASCII está en el Acumulador.
DJNZ bucle      ; Decrementa B y si no hay Zero, salta a bucle.

RET             ; "Retorna" a donde estaba.
```

Bueno, como veis hemos introducido alguna rutina del Firmware más. La verdad es que nos serán de utilidad en muchas ocasiones. Podéis ampliar información sobre estas rutinas y más cosas en The Amstrad CPC Firmware Manual (en inglés) que podéis encontrar en :

http://www.cantrell.org.uk/david/tech/cpc/cpc-firmware/

Resumiendo:

SUB N o R	; Resta el byte N o el contenido en el registro R del que hay en Acumulador. Resultado queda en Acumulador.
CP N o R	; Compara (resta) el byte N o el contenido en el registro R con el que hay en el Acumulador. No se guarda el resultado, sólo se cambian flags.
ADC A, N o R	; Suma el byte N o el contenido en el registro R y el Carry al Acumulador, quedado allí el resultado
SBC N o R	; Resta el byte N o el contenido en el registro R y el Carry del Acumulador, quedando allí el resultado.
JP c, NN	; Salto condicional a la dirección NN.
JR c, d	; Salto relativo de **d** bytes. Sólo con Carry y Zero.
DJNZ d	; Decrementa B y si no es Zero salta **d** bytes.
CALL c, NN	; Llamada condicional a la dirección NN
RET c	; Retorno condicional a donde se llamó a la rutina.

Nota. En estas instrucciones, la condición es opcional y puede omitirse.

AND N o R	; AND lógico bit a bit entre el Acumulador y N o R.
OR N o R	; OR lógico bit a bit entre el Acumulador y N o R
XOR N o R	; XOR lógico bit a bit entre el Acumulador y N o R
CPL	; Complementa a uno el Acumulador.
NEG	; Complementa a dos el Acumulador.
DAA	; Ajusta el Acumulador en BCD.
BIT b,R	; Activa flag Zero si hay un 0 en el bit **b** del registro R.
SET b,R	; Pone a uno el bit **b** del registro R.
RES b,R	; Pone a cero el bit **b** del registro R.
SCF	; Pone a uno el Carry Flag.
CCF	; Complementa el Carry Flag.