SUBRUTINAS DE LA ROM

CAPITULO I

==========

En esta serie de artículos vamos a estudiar las subrutinas de la ROM que componen el Sistema Operativo de nuestro ordenador. Pero primero pasaremos revista a algunos conceptos básicos sobre Assembler, y al microprocesador Z-80.

==========

Jesús Alonso Rodriguez

INTRODUCCION:

Tal vez algun lector se pregunte qué utilidad tiene estudiar las subrutinas de la ROM cuando, de hecho, el ordenador sabe manejarlas muy bien en respuesta a nuestras órdenes en Basic.

En principio, se puede contestar que para programar correctamente, es necesario tener el mayor conocimiento posible del ordenador con el que se trabaja, por lo que toda informacion es poca. Pero hay una razon mas importante, si se conocen bien las subrutinas de la ROM y la forma de usarlas, se pueden hacer cosas que el Basic no permite.

Por otro lado, los programadores de código máquina encontrarán en la ROM una serie de rutinas ya hechas, que les permitiran ahorrarse trabajo, ya que en muchos casos, se podran ahorrar la construccion de una rutina propia, utilizando una de la ROM.

Finalmente, para quienes se estén introduciendo en el mundo del Código Máquina, el estudio de la ROM resultará muy interesante para aprender y depurar técnicas de programacion.

Esta serie pretende dirigirse tanto a aquellos lectores con conocimientos de programacion en Assembler, como a quienes tan solo se interesan por el Basic. Confiamos en que ambos encontrarán útiles los siguientes capítulos.

ESTRUCTURA DE LA SERIE

En cada capítulo trataremos una o varias subrutinas de la ROM, llevando un orden lógico que nos permita aproximarnos con facilidad al conocimiento del Sistema Operativo. Dado que la ROM del Spectrum se organiza como un gran número de subrutinas relacionadas entre sí, a veces será necesario hacer referencia a rutinas no estudiadas todavía, en estos casos se hará de forma que el lector comprenda la funcion de la rutina, aun cuando su estudio detallado se haga en un capítulo posterior.

Para cada rutina habrá una serie de apartados fijos, que serán: Funcion, Descripcion, Entradas alternativas, Condiciones de entrada, Registros empleados, Condiciones de salida y Posibles utilidades. Complementados con los que fueran necesarios en cada caso, así como figuras, tablas o listados, cuando se requieran para una mejor comprension.

En cuanto a la nomenclatura, utilizaremos la propia del assembler, los nombres de las variables serán los utilizados en el manual, y las etiquetas las correspondientes al código fuente del Sistema.

Los números se representarán en hexadecimal, indicándose con una "h." al final del número; a continuacion irá ese mismo número en decimal, con una "d." para indicarlo; en este segundo caso, se utilizará la notacion americana de "coma" para separar los "millares" y "punto" para los "decimales".

Por ejemplo: la direccion de memoria "cuatro mil quinientos treinta y cinco" (correspondiente a la rutina del comando NEW) se representará: 11B7h. (4,535d.)

ASSEMBLER Y CODIGO MAQUINA

Si descendemos al nivel de máquina (forma de trabajar del microprocesador) las únicas instrucciones que este entiende son números que ocupan las distintas posiciones de memoria. El Z-80 ejecuta cerca de 700 instrucciones que se codifican utilizando úno, dos o tres números comprendidos entre "cero" y FFh. (255d.), a estos números se les denomina "Código Máquina" porque es el único código que el microprocesador es capaz de entender de forma directa.

Para los humanos, sería imposible recordar 700 números y las operaciones que cada uno realiza, por lo que se han inventado los "lenguajes de programacion". Los lenguajes de programacion son sistemas de códigos intermedios entre el lenguaje humano y el de la máquina. Se dice que un lenguaje es de mas alto nivel cuanto mas se acerca al lenguaje humano, y de mas bajo nivel, cuanto mas se acerca al lenguaje de la máquina. El Basic es un lenguaje de alto nivel, mientras que el Assembler es el lenguaje de mas bajo nivel posible.

El Assembler es, simplemente, la traduccion de los números que constituyen el Código Máquina a unas palabras simbólicas llamadas "nemónicos" que a los humanos nos son mas fáciles de recordar. Por ejemplo, en Assembler, para indicarle al microprocesador que cargue el registro A con el contenido de la direccion apuntada por el registro doble HL, escribiríamos:

```
LD A,(HL)
```

El nemónico LD es abreviatura de "LoaD" (cargar) y los paréntesis que rodean a HL significan "la direccion apuntada por...". Esta instruccion se ensambla en Código Máquina como: 7Eh. (126d.). A la instruccion en Assembler se le denomina "Código Fuente" y a su codificacion, "Código Objeto". Existen programas capaces de traducir el Código Fuente a Código Objeto, estos programas se denominan "Ensambladores".

BREVE DESCRIPCION DEL Z-80:

Como el lector habrá deducido ya, la programacion en Assembler es distinta para cada microprocesador, de hecho, cada uno tiene su propio Assembler. Nosotros vamos a utilizar, logicamente, el Assembler del Z-80, por lo que será util tener un cierto conocimiento del mismo.

El microprocesador Z-80 se compone de una serie de registros que son como posiciones de memoria, pero internas al microprocesador, y con los cuales realiza este sus operaciones, tiene tambien una unidad logico-aritmética y toda la circuiteria necesaria para realizar la lectura y decodificacion de memoria y controlar las operaciones que se realizan internamente.

Permite dos modos de interrupcion: "Interrupcion no enmascarable" (NMI) con vector de interrupcion fijo (no se usa en el Spectrum) e "Interrupcion Enmascarable" (MI) con tres modos de interrupcion posibles y con posibilidad de definirle el vector de interrupcion (la MI se utiliza en el Spectrum para leer el teclado y actualizar el reloj de tiempo real). Se denomina "vector de interrupcion" a la direccion de memoria donde salta el microprocesador cuando recibe una peticion de interrupcion.

Por otro lado, el Z-80 es capaz de regenerar la RAM dinámica sin que el programador tenga que preocuparse por ello.

El Z-80 tiene 16 registros de 8 bits, que se pueden agrupar de 2 en 2 para formar registros de 16 bits, y 5 registros fijos de 16 bits con utilidades especiales. Estos últimos son:

"PC" o contador de programa
"SP" o puntero de pila
"IX" o índice x
"IY" o índice y
"IR" o vector de interrupcion y registro de regeneracion

En cuanto a los registros de 8 bits, se dividen en dos grupos de 8 registros alternativos cada uno, y se denominan: A, B, C, D, E, F, H, L, A', B', C', D', E', F', H' y L'.

El registro A se denomina "acumulador" porque es el que recibe los resultados de la mayoría de operaciones que realiza el Z-80. El registro F es el indicador de estado, y está compuesto por los "FLAGS" o indicadores de "cero", "paridad", "overflow", "suma/resta", etc. que se utilizan implicitamente en los saltos condicionales.

Los registros se pueden agrupar de la siguiente forma: AF, BC, DE, HL, A'F', B'C', D'E' y H'L'. A los cuatro últimos los denominaremos: AF', BC', DE' y HL'. El propio microprocesador se encarga de considerar los registros como sencillos o dobles, en funcion de la instruccion a la que esté respondiendo.

El conjunto de instrucciones del Z-80 es muy amplio y abarca: carga de registros y posiciones de memoria, búsqueda, intercambio de registros, trasferencia de bloques, operaciones aritméticas y lógicas, operaciones de entrada/salida, control CPU, saltos y bifurcaciones, etc. No podemos entrar aquí a analizar cada una detalladamente, ya que eso requeriría otra serie, pero existe abundante bibliografía sobre el tema.

LOS REINICIOS (RST) DE PAGINA CERO:

Existen una serie de instrucciones que fuerzan al microprocesador a saltar a una subrutina que empieza en una de las primeras direcciones de memoria, cada una de estas instrucciones solo ocupan un byte (frente a los tres de una llamada a subrutina) por lo que se utilizan para las subrutinas de uso mas frecuente. Se llaman: RST 0, RST 8, RST 10, RST 18, RST 20, RST 28, RST 30 y RST 38. En ellas los programadores de Sinclair han colocado las siguientes rutinas:

RST 0 : Inicializacion (START)
RST 8 : Reinicio de error (ERROR-1)
RST 10: Rutina general de salida (PRINT-OUT)
RST 18: Recoge un caracter (GET-CHAR)
RST 20: Recoge siguiente caracter (NEXT-CHAR)
RST 28: Entrada al calculador (FP-CALC)
RST 30: Hacer BC espacios (BC-SPACES)
RST 38: Rutina de la interrupcion enmascarable (MASK-INT)

Empezaremos nuestro estudio de la ROM por estas rutinas, y concretamente, en el próximo capítulo veremos las rutinas de inicializacion (START).

---------- o ----------

SUBRUTINAS DE LA ROM

CAPITULO II

==========

Una vez introducidos en materia, en este capítulo veremos la primera de las rutinas de la ROM. Se trata de la inicializacion del sistema que se ejecuta de forma automática cada vez que se conecta el ordenador.

==========

Jesús Alonso Rodriguez

FUNCION

El microprocesador comienza a trabajar desde el mismo momento que se conecta el ordenador, el circuito de AUTO-RESET se encarga de que en ese instante todos los registros estén a cero, incluido el PC (contador de programa), por lo que la rutina de inicializacion arranca desde la direccion cero.

La finalidad de esta rutina es comprobar la memoria disponible, y fijar una serie de variables y parámetros que necesitará la rutina de ejecucion principal, para empezar a trabajar.

DESCRIPCION

La mayor parte de la rutina de inicializacion se utiliza tambien cuando se ejecuta un comando NEW, esta parte se denomina "START/NEW" y comienza en la direccion 11CBh. (4,555d.). Esta rutina se comporta de forma distinta, en funcion del contenido del registro A, ("00" para START, "FF" para NEW).

Estudiaremos primero la rutina START, luego la rutina NEW, y finalmente, la START/NEW.

RUTINA "START"

Esta rutina arranca desde la direccion cero, y realiza las siguientes funciones:

- .- Desactiva la interrupcion enmascarable (MI)
- .- Pone a "00" el registro A (Acumulador), que en este caso, se usa como un FLAG para indicar a la rutina START/NEW si se trata de una inicializacion completa (RESET) o de la ejecucion de un comando NEW.
- .- Pone a FFFFh. (65,535d.) el registro doble DE, este es el valor de la máxima RAM posible.
- .- Salta a la rutina START/NEW

RUTINA "NEW"

Se trata de la rutina correspondiente a la ejecucion del comando NEW, no se ejecuta al conectar el ordenador, pero sin embargo, se la puede considerar parte de la inicializacion del sistema, de hecho, hace el papel de un "WARM RESET" (Arranque en "caliente").

Su mision es borrar el programa Basic y sus variables, e inicializar el resto de las variables del sistema, excepto RAMTOP, P_RAMT, RASP, PIP y UDG; tampoco borra los Gráficos Definidos por el Usuario ni nada que se encuentre por encima de RAMTOP. Comprueba la memoria, y no altera el valor de RAMTOP a menos que encuentre un error en una posicion de memoria.

Comienza en la direccion 11B7h. (4,535d.) y realiza las siguientes funciones:

- .- Desactiva la interrupcion enmascarable (MI)
- .- Pone a "FF" el registro A, por medio de esto se diferencia en la rutina START/NEW desde donde se entra.
- .- Copia en el registro doble DE el valor de la variable RAMTOP
- .- Salva en los registros alternativos BC', DE' y HL' las variables P_RAMT, RASP, PIP y UDG.
- .- Continua en la rutina START/NEW.

RUTINA "START/NEW"

Esta rutina es comun para el comando NEW y la inicializacion (START), con las siguientes diferencias:

Desde Start: Registro A conteniendo "00"
Registro DE conteniendo "FFFF"

Desde NEW: Registro A conteniendo "FF"
Registro DE conteniendo el valor de RAMTOP

La rutina arranca en la direccion 11CBh. (4,555d.) y se divide en varias partes que realizan las siguientes funciones:

Posicion 11CB (START/NEW)

- .- Pone blanco el color del borde
- .- Espera 24T. veces, el tiempo T. es la cuarta parte de un ciclo de instruccion del Z-80 (esto se hace ejecutando seis NOPs)

Posicion 11DA (RAM-CHECK)

- .- Chequea toda la memoria RAM desde la direccion 3FFFh. (16,383d.) hasta la direccion apuntada por el contenido del registro DE.

  Aqui hay implícita una diferencia entre la entrada desde START o desde NEW, en el primer caso chequea toda la memoria posible; mientras que en el segundo caso, solo chequea hasta el contenido de la variable RAMTOP y si no detecta ningun error, no lo cambia. Esto se hace así para no deteriorar la forma de los gráficos definidos por el usuario, que el comando NEW no tiene que inicializar.

.- Si se entra desde START crea las variables P_RAMT, UDG, RASP y PIP con los valores:

P_RAMT: Direccion del último octeto de la memoria disponible.
UDG: Contenido de P_RAMT menos A7h. (167d.)
RASP: 40h. (64d.)
PIP: Cero

y copia el formato de los caracteres de la A a la U en el espacio entre UDG y P_RAMT.

.- Crea la variable RAMTOP. En el caso de START le dá el valor UDG menos 1. En el caso de NEW, le dá el valor que tenía, a no ser que detectara un error en la memoria, en cuyo caso, le dá el valor de la direccion del octeto anterior al error. (Esto resulta muy util para permitir el volcado de software que llevan a cabo algunos interfaces durante la inicializacion).

.- Crea la variable CHARS con el valor 3C00h. (15,360d.) que es la posicion donde están definidos los caracteres menos 100h. (256d.)

.- Almacena 3Eh. (62d.) en la direccion de RAMTOP.

.- Carga el registro SP (Puntero de pila de máquina) del microprocesador con el valor de RAMTOP menos 1

.- Crea la variable ERR_SP con el valor de RAMTOP menos 3

.- Activa el modo 1 de interrupcion

.- Carga el registro IY (uno de los índices) con la direccion de la variable ERR_NR

.- Activa la interrupcion enmascarable (MI), a partir de este momento, cada 20 milisegundos se actualiza el reloj de tiempo real y se produce la lectura del teclado.

.- Crea la variable CHANS con el valor ~~5C3A~~h. (~~23,610d.~~) (23734 5CB6

.- Pone valores iniciales al area que direcciona CHANS, estos valores son: F4, 09, A8, 10, 4B, F4, 09, C4, 15,53, 81, 0F, C4, 15, 52, F4, 09, C4, 15, 50, 80.

En principio hay 4 canales: "K" (Teclado y parte inferior de la pantalla), "S" (Parte superior de la pantalla), "R" (Area de trabajo) y "P" (Impresora). Para cada uno se usan 5 bytes, los dos primeros direccionan la rutina que hace el OUTPUT por ese canal, los dos segundos, la rutina que hace el INPUT, y el quinto es el código del caractér que dá nombre al canal. La forma es esta:

CANAL "K"
Output: PRINT-OUT: 09F4h. (2,548d.)
Input: KEY-INPUT: 10A8h. (4,264d.)
Código: K: 4Bh. (75d.)

```
CANAL "S"
 Output: PRINT-OUT: 09F4h. (2,548d.)
 Input:   REPORT-J: 15C4h. (5,572d)
 Código:         S: 53h. (83d.)

CANAL "R"
 Output:  ADD-CHAR: 0F81h. (3,969d.)
 Input:   REPORT-J: 15C4h. (5,572d.)
 Código:         R: 52h. (82d.)

CANAL "P"
 Output: PRINT-OUT: 09F4h. (2,548d.)
 Input:   REPORT-J: 15C4h. (5,572d.)
 Código:         P: 50h. (80d.)

MARCA DE FIN: 80h. (128d.)
```

.- Inicializa las variables DATADD, PROG, VARS, E_LINE, WORKSP, STKBOT y STKEND con los valores:

```
DATADD:   (CHANS) + 14h.
PROG:     (DATADD) + 1
VARS:     (DATADD) + 1
E_LINE:   (PROG) + 1
WORKSP:   (E_LINE) + 1
STKBOT:   (E_LINE) + 1
STKEND:   (E_LINE) + 1
```

.- Pone las marcas de FIN (80h.) antes del area de Programa y despues del area de Variables.
.- Inicializa las variables ATTR-P, ATTR-T, y BORDCR con el valor 38h. (56d.) correspondiente a: FLASH 0, BRIGHT 0, PAPER 7, INK 0.
.- Inicializa las variables REPDEL y REPPER con los valores 23h. y 05h.
.- Inicializa las variables KSTATE-0 y KSTATE-4 con el valor FFh.
.- Carga la variable STRMS con los valores: /01, 00,/06, 00,/0B, 00,/01,°00,/01,⁴00,/06,²00,/10,³00./
.- Activa el bit 1 de FLAG3 indicando "printer en uso"
.- Limpia el buffer de impresora (CALL CLEAR-PRB)
.- Inicializa la variable DF-SZ con el valor 02h.
.- Limpia la pantalla (CALL CLS)
.- Imprime el mensaje: "(c) 1982 Sinclair Research LTD" (CALL PO-MSG con A=0 y DE= 1538h.)
.- Activa bit 5 de TV-FLAG indicando que la parte baja de la pantalla requerirá ser limpiada.
.- Salta a la rutina de ejecucion principal (JR MAIN-1)

ENTRADAS ALTERNATIVAS:

En este caso, solo se puede entrar desde cero para una inicializacion total, o desde 11B7h. (4,535d.) para un NEW (RANDOMIZE USR 4535 tendrá el mismo efecto que NEW, y RANDOMIZE USR 0 tendrá el mismo efecto que desconectar y conectar el equipo, o que pulsar el RESET).

CONDICIONES DE ENTRADA:

En START y NEW no hay ninguna condicion de entrada, en START/NEW (direccion 11CBh. o 4,555d.) hay que entrar con la interrupcion desactivada y el registro A conteniendo "00" para un RESET total o "FF" para un NEW, asimismo, en caso de NEW, el registro DE deberá contener el valor de RAMTOP, y "FFFF" en el caso de un RESET total.

REGISTROS EMPLEADOS:

En este caso, se altera el contenido de todos los registros.

CONDICIONES DE SALIDA:

Se produce la inicializacion de la máquina, por lo que se pierde el control del programa, retornando a través de la "MAIN EXECUTION ROUTINE" (Rutina de Ejecucion Principal)

POSIBLES UTILIDADES:

En este caso, la utilidad mas evidente es la de inicializar el ordenador sin necesitar desconectarlo (para aquellos que no dispongan de RESET).

Hay que hacer incapié en que la forma de chequear la memoria, permite a los interfaces volcar su software durante la inicializacion. La rutina RAM-CHECK primero llena la memoria de arriba a abajo, y posteriormente, la lee de abajo a arriba. Donde lee un dato distinto del que há escrito, se detiene, y apunta la direccion inmediata inferior como valor de P_RAMT, por lo que el software colocado por el interface en la parte alta de la memoria, queda a salvo de borrados, incluso con NEW.

---------- o ----------

SUBRUTINAS DE LA ROM

CAPITULO III

==========

Una vez vistas las rutinas que permiten al sistema inicializarse y empezar a trabajar, veremos en este capítulo las restantes subrutinas de página cero.

==========

Jesús Alonso Rodriguez

RUTINAS DE PAGINA CERO:

Estas rutinas ocupan desde la posicion 0000 a la 028Dh. (653d.). En esta parte de la memoria se encuentran las rutinas correspondientes a los "reinicios" (Restart) de página cero y algunas subrutinas relacionadas con ellas, en la figura 1 se muestra un mapa de esta parte de la memoria.

Suele ser comun a todos los sistemas operativos usar las primeras posiciones de memoria para tablas de saltos, tratamiento de interrupciones, rutinas de inicializacion, etc. En general, aquellas rutinas básicas para atender al propio sistema operativo y al usuario.

En sistemas operativos no escritos para ROM, suelen encontrarse tambien en esta zona, las variables básicas tales como R_T_CLOCK (Reloj de tiempo real, en el Spectrum:"FRAMES"), Fecha, Hora, Tamaño de memoria, etc.

En el caso del Spectrum, y por estar su sistema escrito para ROM, las variables del sistema se encuentran por encima de la direccion 3FFFh. (16,383d.).

A continuacion, detallaremos todas las rutinas de esta zona.

START:

Posiciones 0000 a 0007h.

Se entra con RST 0

Esta rutina se trató ampliamente en el capítulo II dedicado a la inicializacion del sistema.

ERROR-1:

Posiciones 0008 a 000Fh. (8 a 15d.)

Se entra con RST 8 seguido de un literal con el código de error, menos uno (por ejemplo, para el error 2 "Variable not found" el literal deberá ser "1").

Por esta rutina sale siempre el intérprete de Basic, tanto si detecta un error, como si termina la ejecucion correctamente, ya que el mensaje "OK," es tratado como un mensaje mas de error.

La forma de entrar a esta rutina es mediante la instruccion "RST 8" (que se ensambla como CFh. 207d.) seguida de un literal, que es simplemente, un byte ensamblado a continuacion del código, y que indica el mensaje que hay que imprimir, por ejemplo, si se desea imprimir el mensaje "Variable not found" (código 2), la forma sería:

```
CF    RST 8      ;Llama rutina error
01    DEFB +01   ;Mensaje con codigo 2
```

>DESCRIPCION:

Posicion 0008 (ERROR-1)

.- Pasa el valor de la variable CH-ADD a X-PTR, es la posicion donde se ha detectado el error (recuérdese que la terminacion correcta con mensaje "OK," es tratada como un error).

.- Continua en la rutina ERROR-2

Posicion 0053h. 83d. (ERROR-2)

.- Carga en el registro "L" el código de error, definido en la posicion siguiente a la instruccion RST 8

.- Continua en la rutina ERROR-3

Posicion 0055h. 85d. (ERROR-3)

.- Carga en la variable ERR-NR el código de error desde el registro "L".

.- Carga el registro "SP" con el contenido de la variable ERR-SP, con lo que se limpia la pila de máquina salvando solo el retorno del primer CALL que se hubiera hecho.

.- Salta con JP 16C5 a la rutina SET-STK que limpiará la pila del calculador y provocará un retorno a la direccion apuntada por el último dato de la pila de máquina, que será la direccion de retorno del primer CALL que se haya hecho, con lo que se vuelve al intérprete de Basic, que a su vez, se encargará de llamar a la rutina PO-MSG que será la que finalmente imprima el mensaje en pantalla.

Estas rutinas se estudiaran con detalle cuando se trate la rutina de ejecucion principal. El sistema operativo del Spectrum está lo suficientemente "enmarañado" como para que sea imposible seguirle la pista a cada rutina en particular, es mas facil comprenderlo todo cuando se tenga una vision global del sistema.

De momento, puede sernos util saber que podemos retornar al Basic desde C/M ejecutando un RST 8 seguido del código FFh. (255d.) para el mensaje "OK,". Esto nos permite retornar con éxito aun cuando nuestra rutina hubiera desordenado el stack de máquina, si bien se interrumpirá la ejecucion del programa Basic con el mensaje "OK,". Pruebe la siguiente rutina en código máquina:

```
E5    PUSH HL   ;Desordenamos el stack
C5    PUSH BC   ;metiendo numeros
D5    PUSH DE   ;sin significado.
CF    RST  8    ;Llamamos rutina de error
FF    DEFB #FF  ;Mensaje "0 OK,"
```

Parece que el retorno es imposible porque hemos desordenado el stack, pero compruebe que si ejecuta "RANDOMIZE USR" a la direccion de comienzo, obtiene: "0 OK, 0:1"

Para quienes no dispongan de ensamblador, pueden ejecutar:

```
10 CLEAR 29999
20 FOR n=30000 TO 30004
30 READ a: POKE n,a
40 NEXT n
50 DATA 229,197,213,207,255
```

Y a continuacion: RANDOMIZE USR 30000

PRINT-A-1:

Posiciones 0010 a 0012h. (16 a 18d.)

Se entra con RST 10 teniendo en el registro "A" el código del caracter a imprimir o un código de control.

Esta rutina efectua un salto a la rutina PRINT-A-2 situada en la direccion 15F2h. (5618d.) por lo que dá lo mismo hacer RST 10 que hacer CALL 15F2, pero en el primer caso solo se utiliza un byte. Esta es una de las rutinas de uso mas frecuente por el sistema operativo, y por esa razon se ha previsto la posibilidad de entrar con un "Restart" de página cero.

El funcionamiento se estudiará cuando se vea la rutina PRINT-A-2, pero podemos adelantar que se trata de una de las rutinas mas potentes del sistema, pudiendo manejar Tokens y códigos de control. De momento, puede hacer pruebas ejecutando RST 10 con diversos códigos en el acumulador; verá que con llamadas a RST 10 puede hacer todo lo que con PRINT.

Es importante señalar que RST 10 es capaz de dirigir la informacion a cualquier canal, por lo que es necesario abrir un canal antes de utilizarla, podemos anticipar que esto se hace colocando en el registro "A" el número de canal, y llamando a la rutina CHAN-OPEN mediante: CALL #1601 que se ensambla: CD,01,16h. (205,1,22d.)

GET-CHAR:

Posiciones 0018 a 001Fh. (24 a 31d.)

Esta rutina comprueba el contenido de la posicion de memoria direccionada por la variable CH-ADD, si es un caracter imprimible retorna, si no incrementa la variable CH-ADD y repite la prueba hasta que encuentre un caracter imprimible.

Utiliza las subrutinas: GET-CHAR, TEST-CHAR, SKIP-OVER, NEXT-CHAR y CH-ADD+1

>DESCRIPCION:
Posicion ØØ18h. 24d. (GET-CHAR)
.- Carga en el registro "A" el contenido de la posicion direccionada por CH-ADD.
.- Continúa en TEST-CHAR

Posicion ØØ1Ch. 28d. (TEST-CHAR)
.- Efectua la rutina SKIP-OVER que devuelve el flag de acarreo activado si el caracter no es imprimible.
.- Retorna si no tiene el acarreo activado
.- Si el acarreo está activado, salta a la rutina NEXT-CHAR.

NEXT-CHAR:

Posiciones ØØ2Ø a ØØ24h. (32 a 36d.)

Apunta al siguiente caracter segun CH-ADD y retorna si es imprimible, si no sigue incrementando CH-ADD.

Se entra con RST 2Ø y un valor determinado en la variable CH-ADD. El intérprete de Basic llama repetidamente a estas dos rutinas segun se vá moviendo a lo largo de las lineas de programa, para comprobar los distintos códigos.

En realidad, cabe considerar a NEXT-CHAR como una entrada alternativa de GET-CHAR.

>DESCRIPCION:
Posicion ØØ2Øh. 32d. (NEXT-CHAR)
.- Efectua la subrutina CH-ADD+1 (incrementar CH-ADD en 1)
.- Vuelve a la posicion TEST-CHAR de la rutina anterior.

FP-CALC:

Posiciones ØØ28 a ØØ2Ah. (4Ø a 42d.)

Se entra con RST 28 seguido de una serie de literales que indican las operaciones a realizar por el calculador.

Al igual que en PRINT-A-1, se trata de el punto de entrada a otra de las mas potentes rutinas del sistema, en este caso, la rutina del calculador en coma flotante. De la misma forma que antes, se salta directamente al calculador, situado en la direccion 335Bh. (13,147d.)

El funcionamiento de esta rutina, que es tan complejo como potente, se verá con detalle cuando estudiemos las rutinas de cálculo.

BC-SPACES:

Posiciones ØØ3Ø a ØØ37h. (48 a 55d.)

Esta rutina deja un número de posiciones libres en el area de trabajo desplazando hacia arriba el resto de la memoria. Este número de posiciones viene definido por el contenido del par de registros "BC".