

Resumen:

TronOS es un pequeño sistema monitor para microprocesadores 8085 montados sobre una configuración específica, este sistema monitor programa y maneja un chip 8279 para controlar una pequeña pantalla de 7 segmentos y un teclado de 11 teclas, también maneja una memoria **RAM** 8156 y una **ROM** 8355. La subrutina principal de *TronOS* permite calcular el cambio de cierta cantidad de dinero en un determinado tipo de moneda a otro tipo de moneda diferente.

Configuración del Sistema:

La configuración del sistema puede verse en el archivo */tronOs/Configuration.pdf*

Convenciones:

Antes de entrar en la explicación de las distintas secciones que conforman *TronOS* debemos explicar las convenciones que se tomaron dentro del sistema monitor.

Numeros:

Los números en *TronOS* son representados usando 1 byte por cada dígito es decir, el número entero 55, se representa de la siguiente manera:

05	05
----	----

Donde cada casilla representa 1 byte, en el caso de los números decimales la coma “,” también es representada con 1 byte, cuyo código hexadecimal asignado es el “40H”. Por ejemplo el número 11.11 es representado como sigue:

01	01	40	01	01
----	----	----	----	----

Donde de nuevo cada casilla representa 1 byte.

Tasas de Conversion :

TronOS soporta 6 tasas de conversión donde fue asignado un código a cada una como se muestra en la siguiente tabla:


RATES NAMES CODES :		
USD -> COP:	1	
USD -> VES:	2	
COP -> USD:	3	
COP -> VES:	4	
VES -> USD:	5	
VES -> COP:	6	

Este formato facilita las operaciones de conversión, ya que conociendo el código de conversión se conoce la moneda inicial y la moneda destino.

La convención para las tasas también proporciona la facilidad de tener una unica operacion aritmetica adicional (además de las soportadas por el **CPU**), la multiplicación.

Teclado y Pantalla (E/S):

Al 8279 de nuestra configuración se le conecta una pantalla y un teclado como se muestra a continuación:

			0	====>	00H
			1	====>	01H
			2	====>	02H
			3	====>	03H
			4	====>	04H
			5	====>	05H
			6	====>	06H
			7	====>	07H
			8	====>	08H
			9	====>	09H
			./,	====>	40H

La rutina que lee del teclado representa las teclas con un código para cada tecla (Esta rutina será explicada en secciones posteriores). Por ejemplo si el “0” fue presionado será representado por el código hexadecimal “00H” en el caso de que haya sido presionado el “1” se representará con el código “01H” y de la misma forma para las demás teclas.

Para leer del teclado se debe llamar a la rutina **INPUT** (ver sección procesos), esta utiliza un buffer de 10 bytes (ver sección de memoria) donde almacena lo que se ha ingresado en el teclado.

Para imprimir en la pantalla existen dos funciones (ver sección procesos) una para imprimir números (dígitos) **PRINT_NUMBER** y otra para imprimir caracteres **PRINT_ASCII** (de nuevo, las funciones para imprimir en pantalla serán explicadas exhaustivamente en secciones posteriores). En ambos casos el procedimiento para imprimir por pantalla es el mismo, ambas usan un mismo buffer de 16 bytes donde se deben escribir los códigos de caracteres o dígitos que se desean imprimir y luego llamar a la función de impresión correspondiente. En el caso de imprimir números la primera entrada del buffer es la cantidad de dígitos que posee el número (incluyendo la coma).

Los caracteres que soporta *TronOS* son un subconjunto de la tabla **ASCII**, tanto los caracteres como sus códigos correspondientes se presentan en la tabla a continuación:

ASCII TABLE			
ASCII	CHAR	7CODE.0XB	7CODE.0XH
00H	\n	11111111B	FFH
20H		11111111B	FFH
40H	,		
41H	A	10001000B	88H
42H	B	00001000B	08H
43H	C	01101100B	6CH
44H	D	00001100B	0CH
45H	E	01101000B	68H
46H	F	11101000B	E8H
47H	G	00101000B	28H
48H	H	10011000B	98H
4AH	J	00001111B	0FH
4CH	L	01111100B	7CH
4FH	O	00001100B	0CH
50H	P	11001000B	C8H
53H	S	00101001B	29H
52H	R	11111000B	8FH
54H	T	11101100B	ECH

En el caso de los números la tabla con los códigos correspondientes a cada dígito es la siguiente:

HEX TABLE			
HEX	CHAR	7CODE.0XB	7CODE.0XH
00H	0	00001100B	0CH
01H	1	11111100B	FCH
02H	2	01001010B	4AH
03H	3	00001011B	0BH
04H	4	00011001B	19H
05H	5	00101001B	29H
06H	6	00101000B	28H
07H	7	10001111B	8FH
08H	8	00001000B	10H
09H	9	00001001B	09H
ABOUT:			
TABLE SIZE 10 ENTRIES FIXED			

Memoria:

TronOS administra la memoria con la técnica de segmentación pura debido a las características del mismo (todos los procesos son conocidos de antemano y se conoce la memoria exacta que usará cada uno de ellos). Se manejan segmentos separados para datos e instrucciones, Las instrucciones y constantes están en memoria **ROM** y los datos en memoria **RAM**. Los mapas de memoria para la **ROM** y la **RAM** así como también el **Memory Mapped I/O** completos pueden verse en */tronOs/memory.pdf*.

Memory Mapped I/O:

La configuración de nuestro sistema establece el siguiente Memory Mapped I/O:

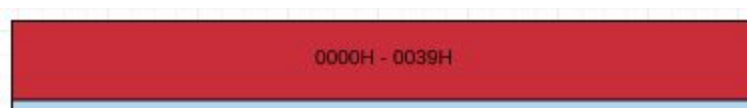
		8279 C/D	RAM	ROM	
0000H	0000000000000000				ROM Lower Bound
07FFH	0000011111111111				ROM Upper Bound
1800H	0001100000000000				RAM Lower Bound
18FFH	0001100011111111				RAM Upper Bound
4800H	0100100000000000				8279 Data
6800H	0110100000000000				8279 Configuration



Mapa de Memoria ROM:

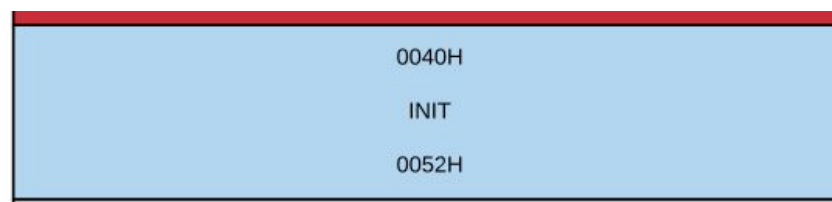
Las constantes e instrucciones se almacenan en la memoria **ROM** (ver mapa de memoria **ROM**), a continuación se dará una explicación detallada de cada uno de los segmentos que conforman la memoria **ROM**.

Segmento 1:



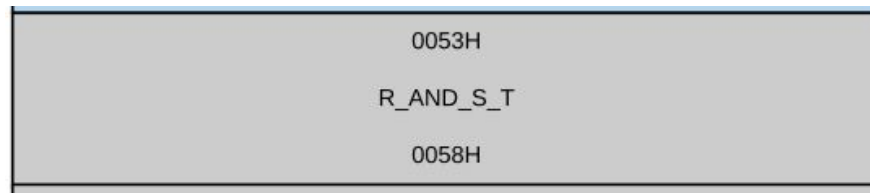
Zona de memoria **ROM** reservada por el 8085 es aquí donde se define el vector de interrupciones y el salto a la primera rutina del sistema monitor. El segmento va desde la posición en memoria 0000H hasta la posición 0039H.

Segmento 2:



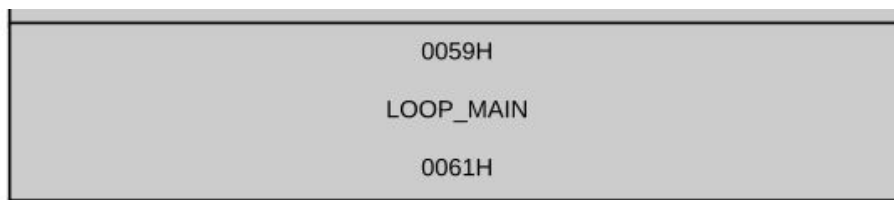
Segmento donde están guardadas las instrucciones de la rutina *INIT* (ver sección procesos). El segmento va desde la posición en memoria 0040H hasta la posición 0052H.

Segmento 3:



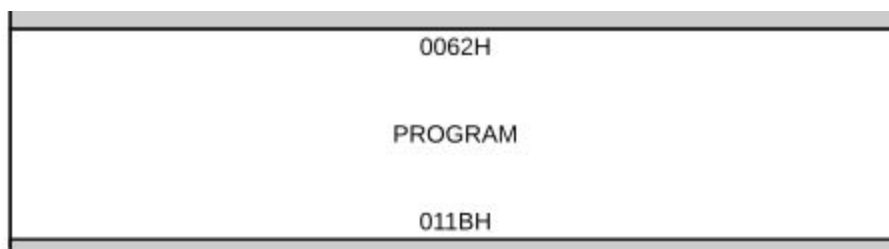
Segmento donde están guardadas las instrucciones de la rutina *R_AND_S_T* (ver sección procesos). El segmento va desde la posición en memoria 0053H hasta la posición 0058H.

Segmento 4:



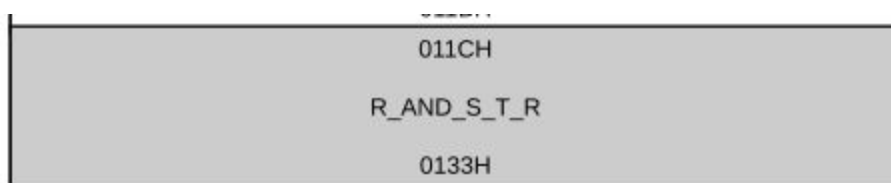
Segmento donde están guardadas las instrucciones de la rutina *LOOP_MAIN* (ver sección procesos). El segmento va desde la posición en memoria 0059H hasta la posición 0061H.

Segmento 5:



Segmento donde están guardadas las instrucciones de la rutina *PROGRAM* (ver sección procesos). El segmento va desde la posición en memoria 0062H hasta la posición 011BH.

Segmento 6:



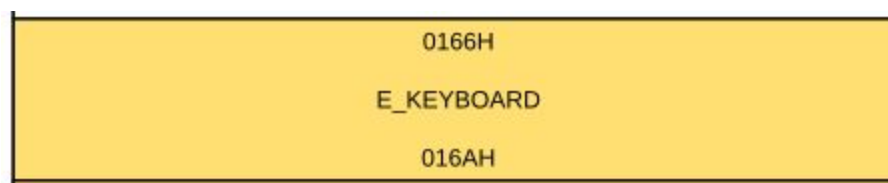
Segmento donde están guardadas las instrucciones de la rutina *R_AND_S_T_R* (ver sección procesos). El segmento va desde la posición en memoria 011CH hasta la posición 0133H.

Segmento 7:



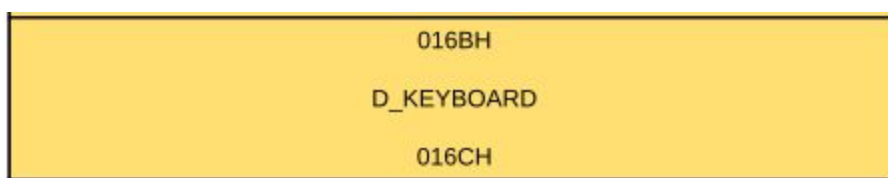
Segmento donde están guardadas las instrucciones de la rutina *R_EXCHANGE_RATE* (ver sección procesos). El segmento va desde la posición en memoria 0134H hasta la posición 0165H.

Segmento 8:



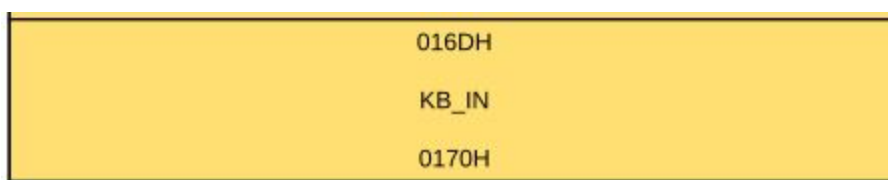
Segmento donde están guardadas las instrucciones de la rutina *E_KEYBOARD* (ver sección procesos). El segmento va desde la posición en memoria 0166H hasta la posición 016AH.

Segmento 9:



Segmento donde están guardadas las instrucciones de la rutina *D_KEYBOARD* (ver sección procesos). El segmento va desde la posición en memoria 016BH hasta la posición 016CH.

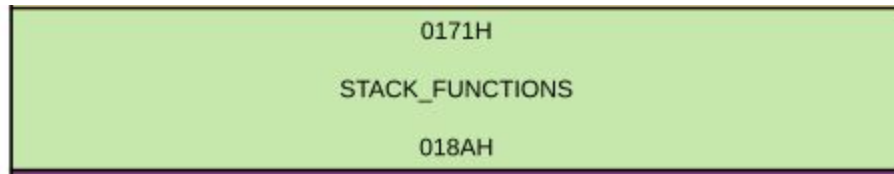
Segmento 10:



Segmento donde están guardadas las instrucciones de la rutina *KB_IN* (ver sección procesos). El segmento

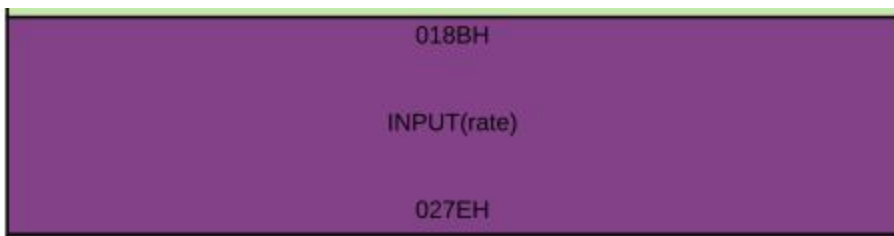
va desde la posición en memoria 016DH hasta la posición 0170H.

Segmento 11:



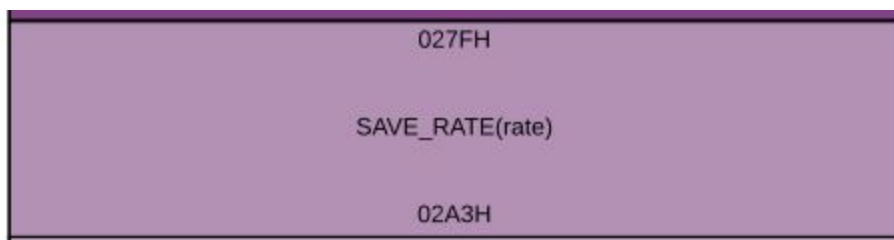
Segmento donde están guardadas las instrucciones de la rutinas *STACK_FUNCTIONS* (ver sección *procesos*). El segmento va desde la posición en memoria 0171H hasta la posición 018AH.

Segmento 12:



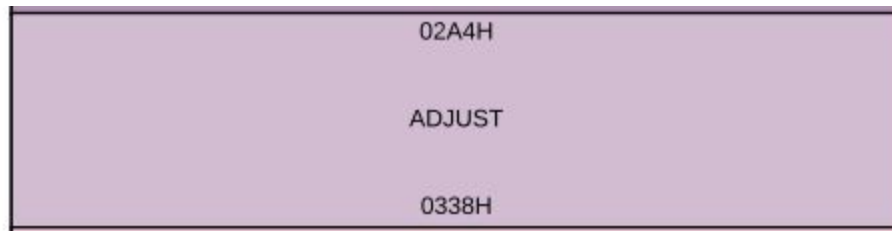
Segmento donde están guardadas las instrucciones de la rutina *INPUT* (ver sección *procesos*). El segmento va desde la posición en memoria 018BH hasta la posición 027EH.

Segmento 13:



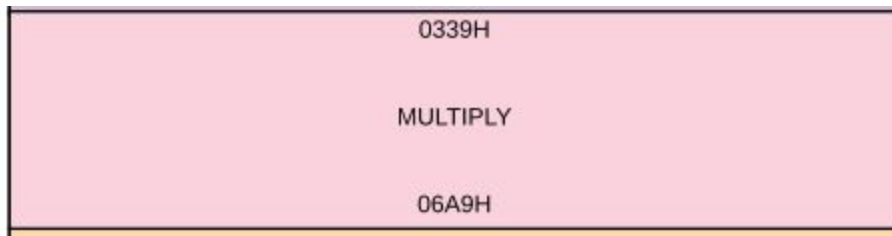
Segmento donde están guardadas las instrucciones de la rutina *SAVE_RATE* (ver sección *procesos*). El segmento va desde la posición en memoria 027FH hasta la posición 02A3H.

Segmento 14:



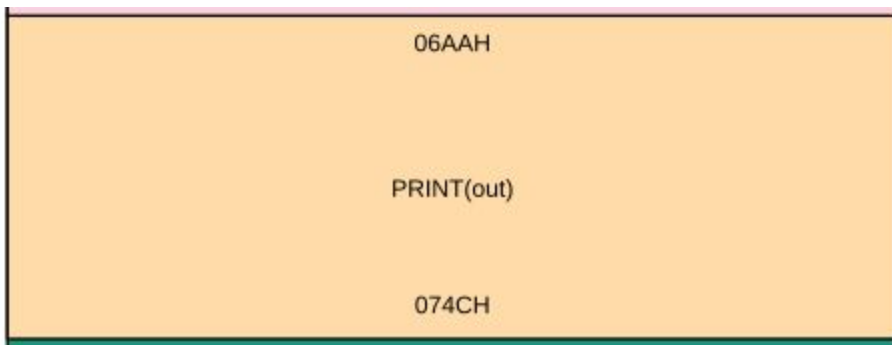
Segmento donde están guardadas las instrucciones de la rutina *ADJUST* (ver sección procesos). El segmento va desde la posición en memoria 02A4H hasta la posición 0338H.

Segmento 15:



Segmento donde están guardadas las instrucciones de la rutina *MULTIPLY* (ver sección procesos). El segmento va desde la posición en memoria 0339H hasta la posición 06A9H.

Segmento 16:



Segmento donde están guardadas las instrucciones de la rutina *PRINT* (ver sección procesos). El segmento va desde la posición en memoria 06AAH hasta la posición 074CH.

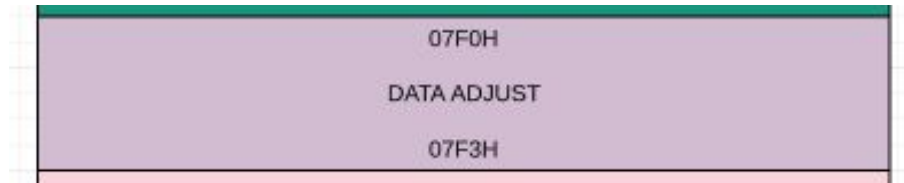
Segmento 17:



Segmento donde están guardadas las constantes de la rutina *PRINT* (ver sección procesos). El segmento va desde la posición en memoria 0750H hasta la posición 0779H.

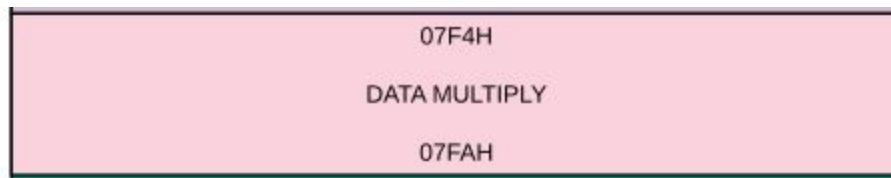
desde la posición en memoria 0750H hasta la posición 0779H.

Segmento 18:



Segmento donde están guardadas las constantes de la rutina *ADJUST* (ver sección procesos). El segmento va desde la posición en memoria 07F0H hasta la posición 07F3H.

Segmento 19:

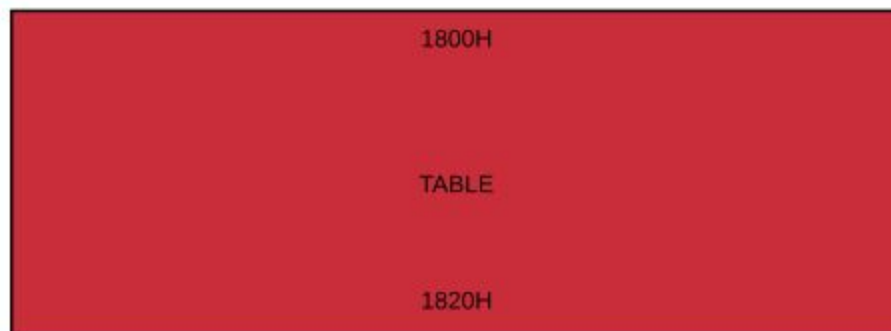


Segmento donde están guardadas las constantes de la rutina *MULTIPLY* (ver sección procesos). El segmento va desde la posición en memoria 07F0H hasta la posición 07F3H.

Mapa de Memoria RAM:

En los segmentos de memoria **RAM** se almacena la data variable de las rutinas tales como buffers de entrada y salida. También proporciona una zona “libre” la cual no está reservada para ningún proceso y puede ser utilizada por cualquiera de ellos, esta zona comprende desde la posición 189BH hasta la posición 18FF.

Segmento 1:



Segmento que comprende desde las posiciones de memoria 1800H hasta la 1820H, es aquí donde se almacena la tabla de conversiones hay un total de 6 posibles conversiones (ver sección convenciones), por lo

que son almacenadas 6 tasas de cambio, cada tasa de cambio es representada por un número de 4 dígitos más la coma, así que se necesitan 5 bytes por cada tasa de cambio. En total son requeridos 30 bytes para guardar la tabla de conversiones (los bytes sobrantes son usados por la rutina que guarda la tabla internamente (ver sección procesos)).

Segmento 2:



Segmento que comprende desde las posiciones de memoria 1821H hasta la 1827H, zona de memoria reservada por *TronOS*, esta zona de memoria puede ser utilizadas para múltiples propósitos, uno de ellos es guardar el estado del **CPU** al momento de llamar a una subrutina para restaurarlo en su retorno.

Segmento 3:



Segmento que comprende desde las posiciones de memoria 1828H hasta la 1868H, segmento reservado para la pila.

Segmento 4:



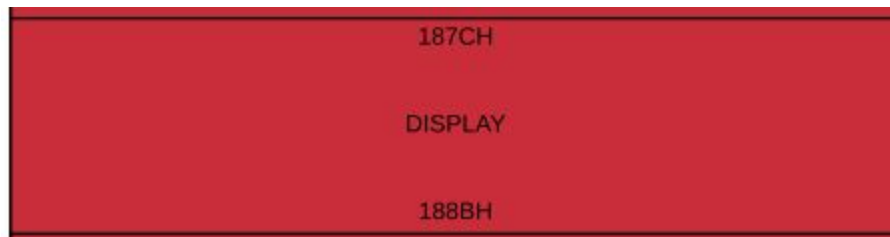
Segmento que comprende desde las posiciones de memoria 1869H hasta la 1872H, es la zona reservada para la rutina que lee del teclado *INPUT* (ver sección procesos). Este segmento es utilizado como buffer de entrada para el teclado, los números que son ingresados por el teclado son almacenados en esta zona de memoria.

Segmento 5:



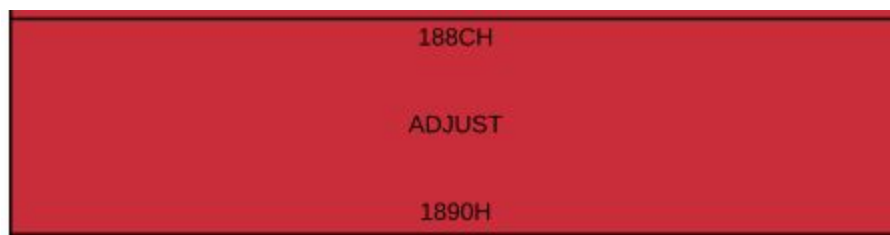
Segmento que comprende desde las posiciones de memoria 1873H hasta la 187BH, segmento reservado para la rutina *MULTIPLY* (ver sección procesos) es utilizado como buffer. La rutina *MULTIPLY* utiliza esta zona de memoria para guardar el resultado de la multiplicación.

Segmento 6:



Segmento que comprende desde las posiciones de memoria 187CH hasta la 188BH, utilizado como buffer de salida, es aquí desde donde las rutinas de impresión por pantalla leen lo que posteriormente se mostrará en la pantalla del sistema. Esto quiere decir que las rutinas que deseen mostrar algún mensaje o número por pantalla deben escribir en este buffer.

Segmento 7:



Segmento que comprende desde las posiciones de memoria 188CH hasta la 1890H, este segmento es utilizado por la rutina *ADJUST* (ver sección procesos). La cual toma lo que se ha leído del buffer de entrada del teclado y lo lleva a la convención de números utilizada por el sistema monitor, el número ajustado se escribe en esta zona de memoria.

Segmento 8:



Segmento que comprende desde las posiciones de memoria 1891H hasta la 189AH, es la zona de memoria donde las rutinas que deseen multiplicar deben colocar los operandos de la multiplicación.

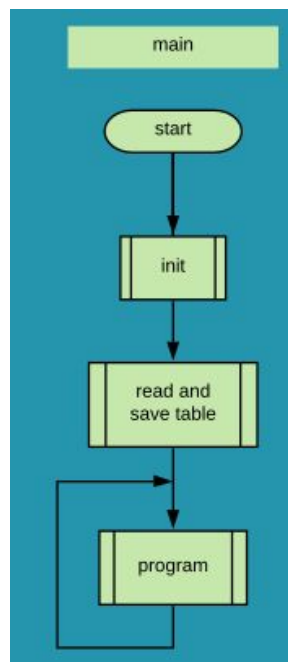
Procesos:

TronOS es monoprogramado es decir que el **CPU** ejecuta un solo proceso en cualquier instante de tiempo, es decir, se le entrega el 100% del **CPU** a cada proceso. Esto se debe a lo pequeño del sistema monitor y a las dificultades que trae consigo la multiprogramación. Los diagramas completos pueden encontrarse en el directorio */tronOS/Diagramas/*

Monitor

En esta sección se explicarán detalladamente cada una de las rutinas del sistema monitor, es decir toda la estructura que hace posible el funcionamiento del mismo. Los diagramas completos de la estructura del sistema monitor pueden verse en el archivo */tronOS/Diagramas/monitor.pdf*.

Diagrama principal:



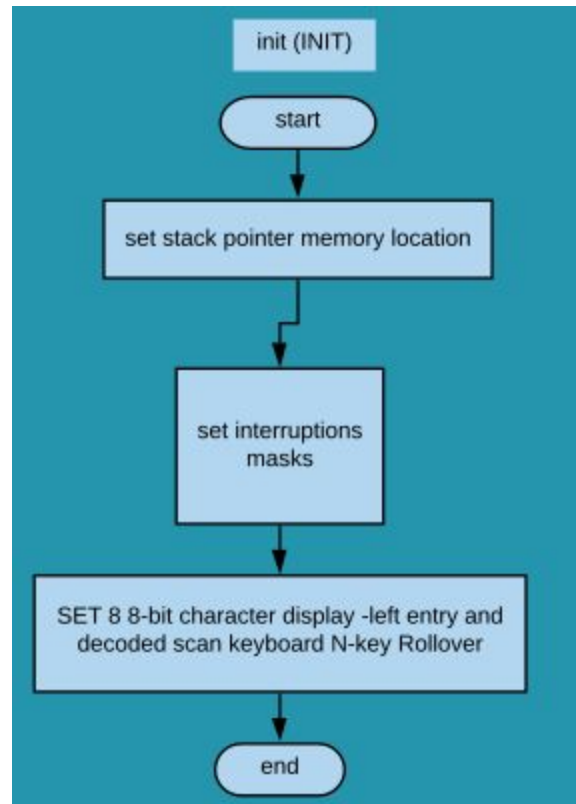
Este es el diagrama de flujo principal del sistema monitor y por lo tanto la vista de nivel superior del mismo.

Juan Diego Morón Flores Cl. 23390971, Administrador de Recursos TronOS.

Este diagrama empieza con una rutina de inicialización *INIT* (ver sección rutina *INIT*), luego lee y guarda la tabla de conversiones (ver sección rutinas *READ AND SAVE TABLE*) y finalmente ejecuta un lazo infinito ejecutando siempre el mismo programa principal (ver sección rutina *PROGRAM*).

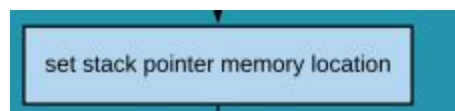
Rutina INIT:

Es la rutina de inicialización principal del sistema monitor es en este punto donde se programan los componentes conectados al 8085 y se enmascaran las interrupciones. El Diagrama de flujo de la misma es el siguiente:



Si vamos paso a paso por cada uno de sus bloques tenemos:

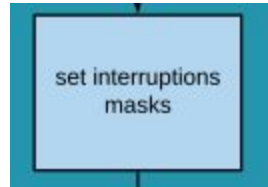
Asignar al stack pointer la posición de memoria correspondiente:



El stack debe estar apuntando a la posición en memoria correspondiente definida para el stack (ver sección *memoria*). Además en este punto se setean las banderas de overflow para el stack, estas banderas son el valor FFH en las dos primeras posiciones del segmento reservado para el stack estas banderas son usadas por las funciones que revisan la integridad del stack (ver sección de las rutinas *STACK*).

```
LXI SP, 1869H; sets stack pointer memory location
MVI A, FFH;
STA 1828H;
STA 1829H;
```

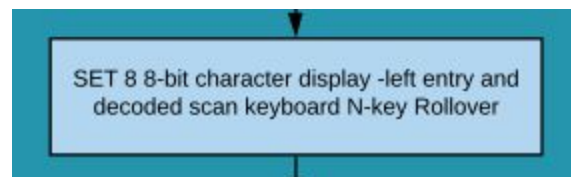
Enmascarado de las interrupciones:



Es en este punto donde son enmascaradas las interrupciones, *TronOS* solo utiliza la interrupción 7.5 destinada al 8279 como control del teclado.

```
MVI A,04H; prepare the mask to enable 7 poitin 5 interrupt
SIM; apply the settings RTS masks
```

Programación de componentes:



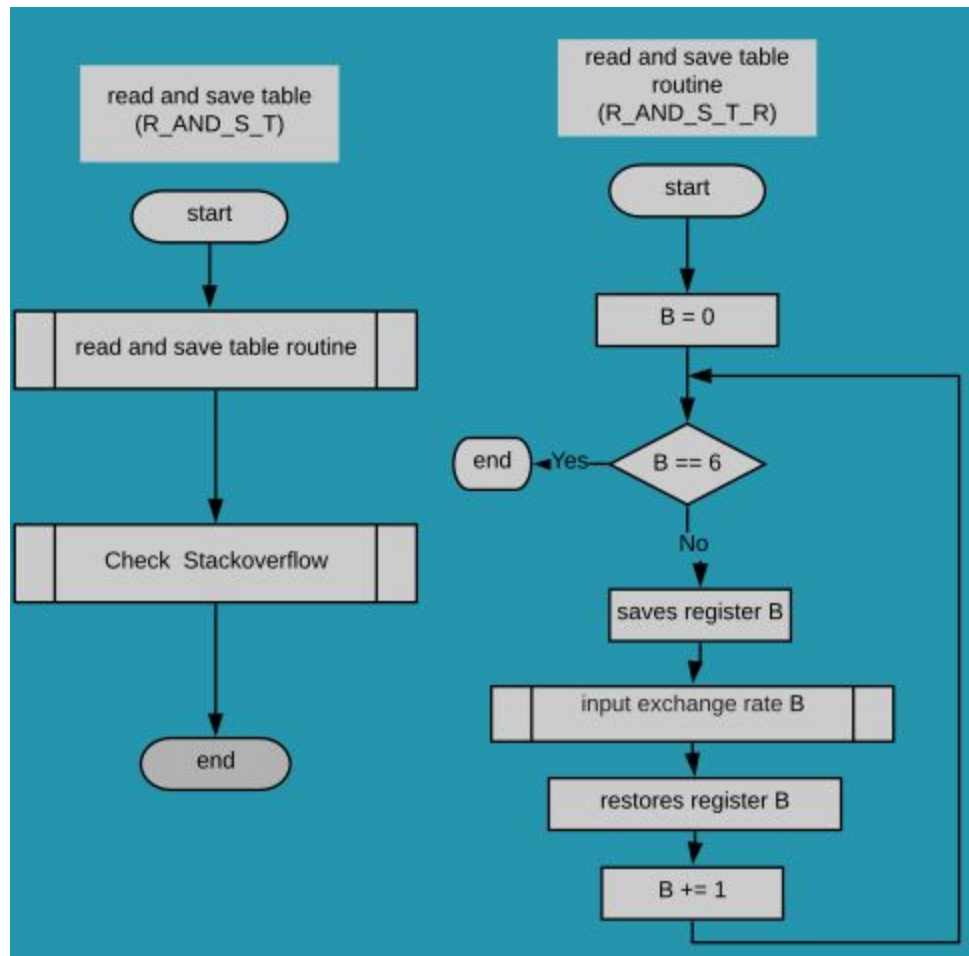
TronOs unicamente programa el 8279 ya que los demás componentes son usados con la configuración predeterminada de cada uno. Para el 8279 la configuración de pantalla usada es *8 8-bit character display left entry*. En el caso del teclado se utiliza la configuración *decoded scan keyboard N-key Rollover*, la cual hace que cuando sean presionadas dos teclas al mismo tiempo ambas sean guardadas en la FIFO del 8279.

Estas configuraciones se logra activando el 8279 en modo configuración y enviando 03H a los registros de configuración.

```
;SET 8 8-bit character display -left entry and decoded scan keyboard n-Key Rollover
LXI H, 6800H; sets !CS - A14 to 0 to activate 8279 and A13 - C/D to 1 to send a command to the 8279
MVI M, 03H; sets 000 [00] [011] that is the desire configuration
```

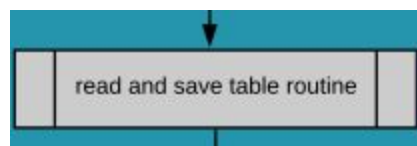
Rutinas READ AND SAVE TABLE:

La lectura y guardado de la tabla de conversiones se hace con la ayuda de dos subrutinas *read and save table* (*R_AND_S_T*) y *read and save table routine* (*R_AND_S_T_R*).

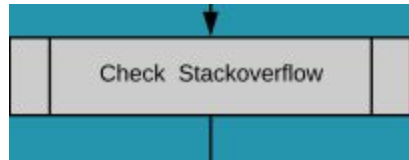


Rutina read and save table (R_AND_S_T):

Funge como rutina envoltorio para llamar a la subrutina *R_AND_S_T_R* (ver sección *read and save table routine*).



Para luego como último paso llama a la función que revisa si el stack fue desbordado o no.



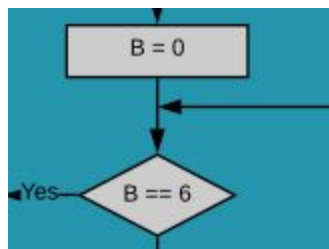
En el código:

```

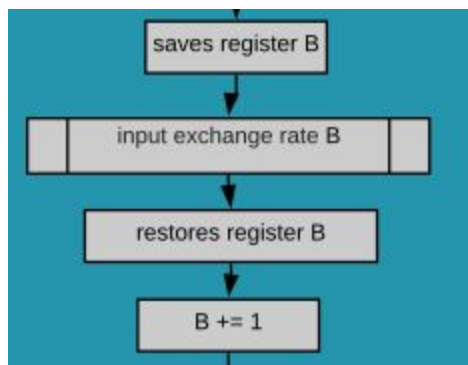
R_AND_S_T: ; read and save table
CALL R_AND_S_T_R; calls read and save table routine
CALL STACK_0_C; we ensure that there is not stack overflow
  
```

Subrutina read and save table routine (R_AND_S_T_R):

Esta subrutina controla el ciclo principal de llenado de la tabla utilizando el registro B como contador principal del ciclo el cual se detiene en el momento en que B alcanza el valor 6, ya que el ciclo se ejecuta 6 veces una vez por cada tasa de cambio. El primer paso que ejecuta la subrutina es inicializar lo necesario para la ejecución del ciclo en este caso al registro B se le asigna el valor 0.

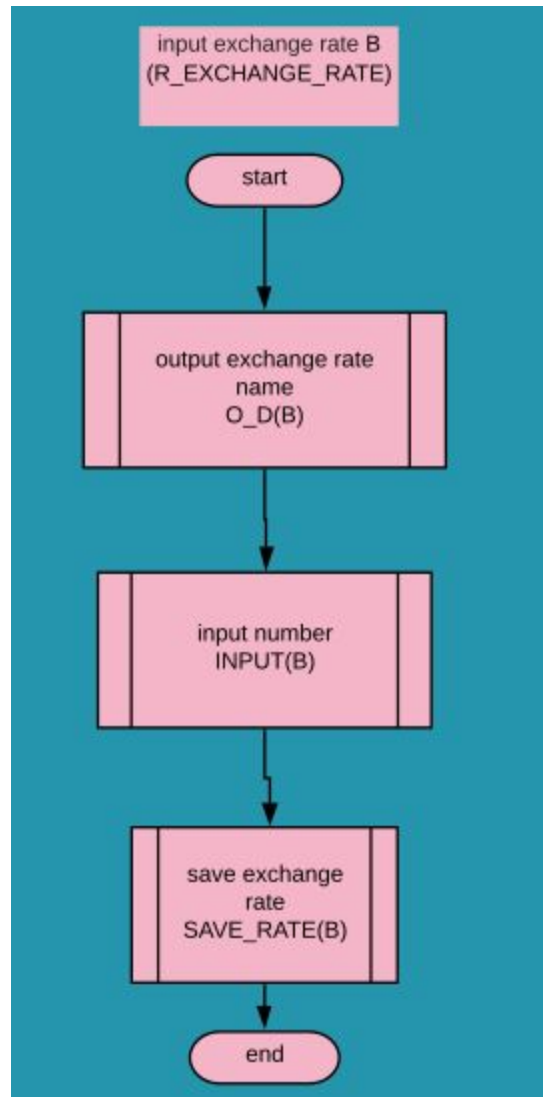


Luego en cada iteración se guarda el contenido del registro B en la zona reservada en memoria para ello (*ver sección memoria*), con el fin de que no se pierda en las llamadas a la subrutina de guardado de tasa de cambio, se llama a la rutina *R_EXCHANGE_RATE* (*ver sección input exchange rate*) una vez retorna de la llamada a la subrutina se restaura el valor del registro B para finalmente incrementarlo y proceder a la siguiente iteración de ciclo.



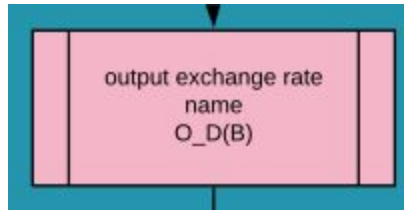
Rutina input exchange rate B (R_EXCHANGE_RATE):

Esta rutina es la encargada de leer una tasa de cambio y guardarla en la posición de la tabla correspondiente, la tasa de cambio y por lo tanto la posición en la tabla correspondiente a ella son indicadas en el registro B (ver sección de las rutinas READ and SAVE TABLE). El flujo principal de R_EXCHANGE_RATE es el siguiente:



Sigue 3 pasos concretamente:

1.- Imprime por pantalla solicitando al usuario que ingrese la tasa de cambio indicada por el código en el registro B.



En este paso primero se coloca en el buffer de salida de las funciones de impresión (*ver sección memoria*) la cadena “CODE”, y luego se llama a la rutina *PRINT_ASCII* (*ver sección de impresión por pantalla*) para que se muestre en pantalla el mensaje.

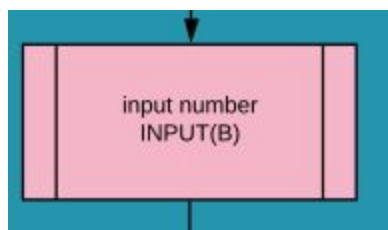
```
; PRINT_TO_DISPLAY pedimos que se ingrese el monto paraa el codigo B
MVI A, 43H; C
STA 187CH;
MVI A, 4FH; O
STA 187DH;
MVI A, 44H; D
STA 187EH;
MVI A, 45H; E
STA 187FH;
MVI A, 00H; \n
STA 1880H;
CALL PRINT_ASCII
```

Luego se imprime por pantalla el código indicado por el registro B. Para ello se copia en el buffer de impresión el número y se llama a la rutina *PRINT_NUMBER* (*ver sección de impresión por pantalla*).

```
MVI A, 01H;
STA 187CH;
MOV A,B;
STA 187DH;
CALL PRINT_NUMBER;
;termina salida por pantalla
```

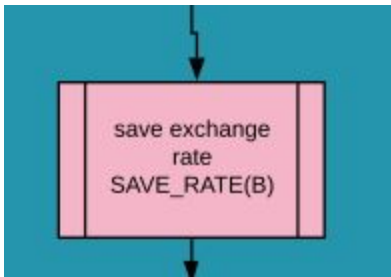
2.- se llama a la rutina *INPUT*.

Se llama a la rutina *INPUT* que deja en el buffer el número ingresado desde el teclado (*ver sección memoria*), el cual corresponde a la tasa de cambio que fue requerida al usuario en el paso anterior.



3.- Ajuste y guardado.

Finalmente se llama a las rutinas *ADJUST* y *SAVE_RATE* las cuales ajustan el número leído al formato de *TronOS* y lo guardan en la tabla respectivamente.

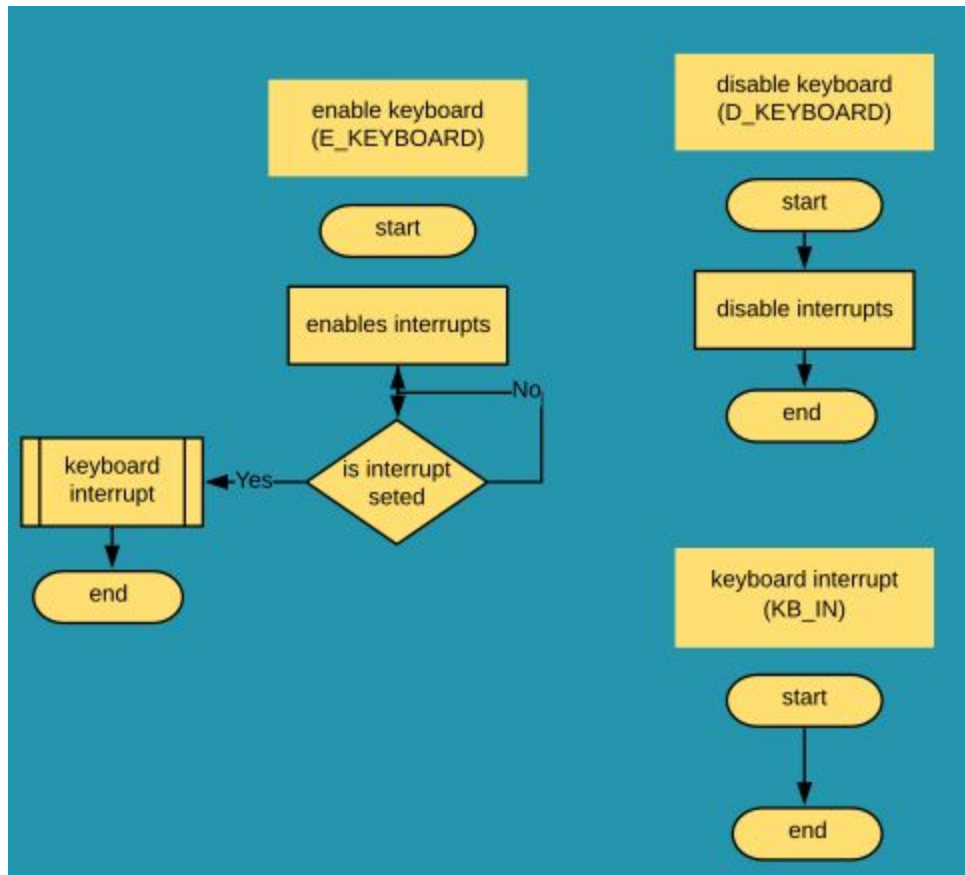


En código:

```
CALL INPUT  
CALL ADJUST  
CALL SAVE_RATE
```

Utilidades para el teclado:

La estructura principal del sistema monitor maneja proporciona funciones utilitarias para el manejo del teclado estas funciones trabajan con las interrupciones ofreciendo funcionalidades tales como la habilitación y deshabilitación de las interrupciones que conllevan a la habilitación y deshabilitación del teclado ya que este trabaja con la interrupción 7.5. Estas rutinas utilitarias tienen los siguientes diagramas:

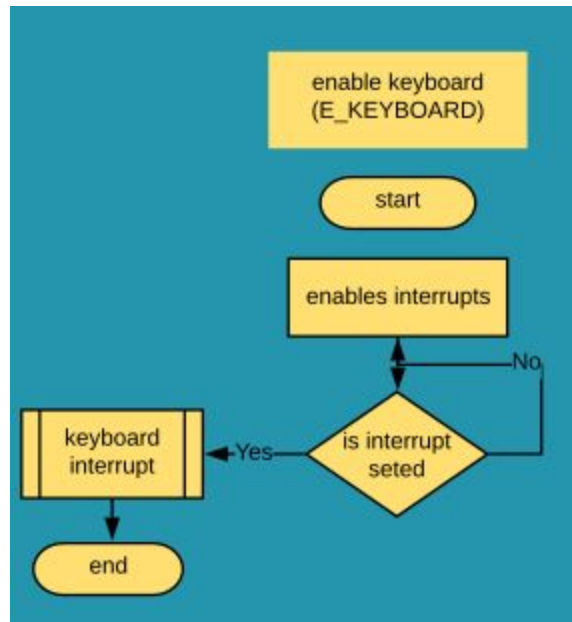


Con el fin de entender estas rutinas es preciso explicar brevemente el funcionamiento del teclado (*para más detalles ver sección INPUT*):

- 1.- El teclado es habilitado, habilitando las interrupciones.
- 2.- El usuario ingresa el numero desde el teclado.
- 3.- Es activada la interrupción indicando que el usuario a ingresado el número.
- 4.- Son deshabilitadas las interrupciones, por lo tanto el teclado es deshabilitado.
- 5.- El número ingresado desde el teclado se encuentra en el buffer correspondiente (*ver sección memoria*).

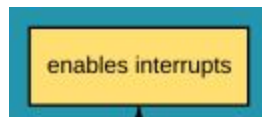
Ahora revisaremos más de cerca las rutinas utilitarias.

Habilitación del teclado, enable keyboard (E_KEYBOARD):



Esta utilidad tiene como función principal habilitar el teclado antes de que ella sea llamada el teclado se encuentra deshabilitado y por lo tanto si alguna tecla es presionada simplemente esta no tendrá ningún efecto en el sistema.

Para realizar esta función *E_KEYBOARD* habilita las interrupciones.



Luego espera hasta que la interrupción 7.5 sea activada, una vez es activada la interrupción se llama a la rutina de servicio de interrupción del teclado (*KB_IN*), para indicar que el número ingresado se encuentra disponible en el FIFO del 8279.

En código esta rutina es bastante simple:

```

E_KEYBOARD: ;enables keyboard by EI, and waits until key is pressed
            EI; enables keyboard interrupts
LOOP_EK:   JMP LOOP_EK; waits until keyboard interrupts
E_KEYBOARD_RET: RET
  
```

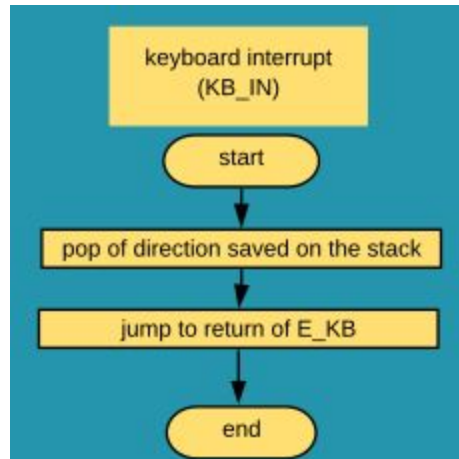
Tome en cuenta que la llamada a (*KB_IN*) se hace con el vector de interrupciones ya que esta es una rutina de servicio de interrupción.

Rutina de Servicio de interrupción de teclado, keyboard_interrupt (KB_IN):

Esta es una rutina de servicio de interrupción de teclado que se llama con la ayuda del vector de

Juan Diego Morón Flores CI. 23390971, Administrador de Recursos TronOS.

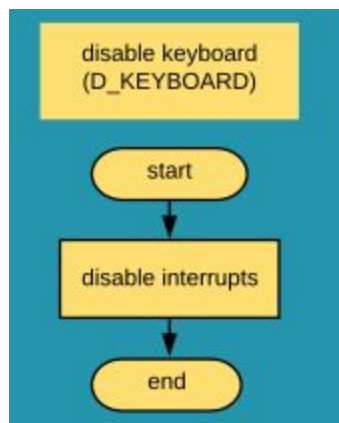
interrupciones. Sirve de “break” para el loop que se ejecuta en (E_KEYBOARD).



El primer paso que ejecuta la rutina es sacar del stack el registro par HL el cual se guarda automáticamente una vez que se llama al servicio de interrupción. Luego hace un *JUMP* hasta el return de la rutina de activación del teclado indicando que el número ingresado desde el teclado se encuentra en la FIFO del 8279. En código la rutina es bastante sencilla:

```
KB_IN: ;keyboard interrupt function
      POP H;
      JMP E_KEYBOARD_RET; jumps to E_KEYBOARD RET instruction
```

Rutina de deshabilitación del teclado, disable keyboard (D_KEYBOARD):



Rutina utilitaria bastante simple usa la instrucción el 8085 para deshabilitar las interrupciones y por ende el teclado.

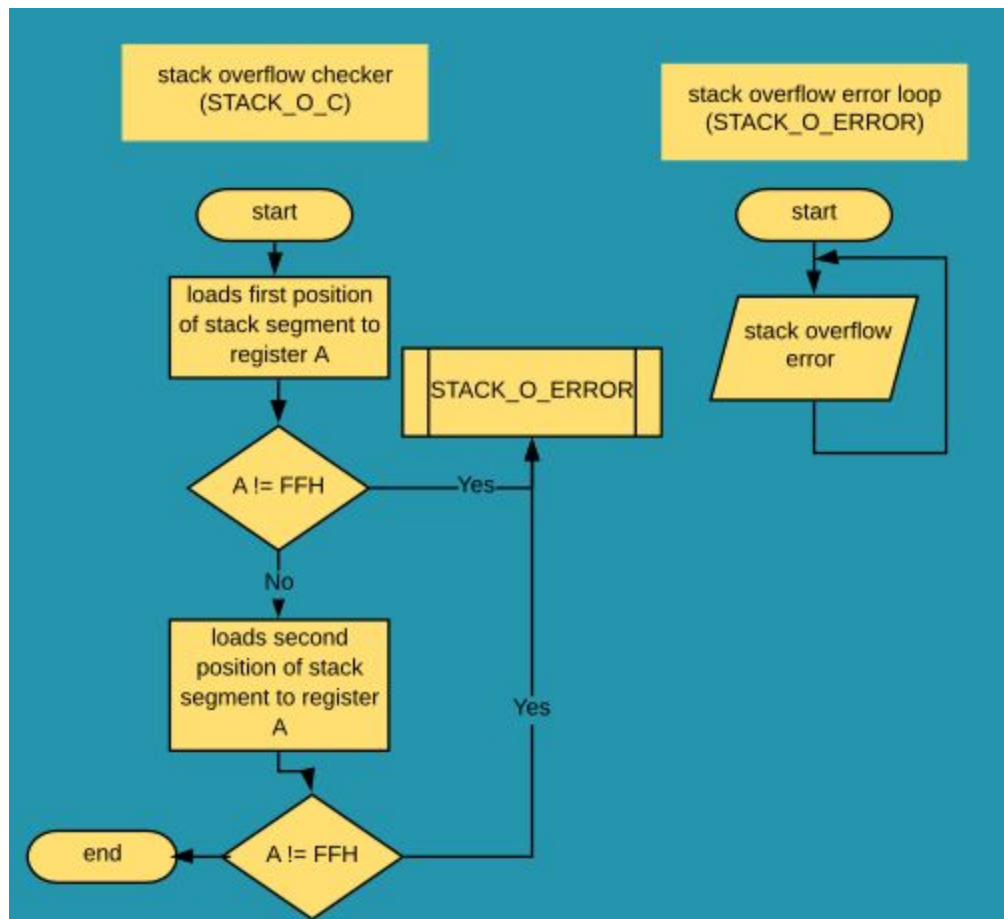
```

D_KEYBOARD:
    DI; disables keyboard interrupts
    RET

```

Rutinas de la pila, stack routines **STACK_O_C** y **STACK_O_ERROR**:

Durante la ejecución de cierto proceso o subrutina es posible que la pila del sistema monitor se desborde por lo que debe existir un mecanismo que avise cuando esto pase de esta forma tronOS sabrá cuando la pila ha sido desbordada. En la estructura del sistema monitor se proporcionan dos rutinas **STACK_O_C** y **STACK_O_ERROR** las cuales hacen el papel de chequear el desborde de pila y avisar al sistema monitor del desborde respectivamente.

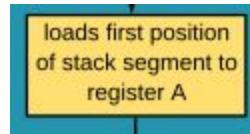


Rutina de chequeo de error **STACK_O_C**:

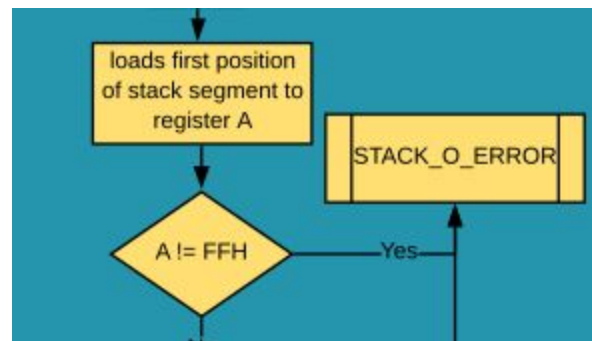
Subrutina encargada de verificar que la pila permanece sin desbordarse, *TronOS* en su inicialización guarda

en los primero bytes de la zona reservada en memoria para la pila el valor FFH (*ver secciones de memoria e INIT*), el cual sirve como bandera para indicar que la pila no ha sido desbordada, el caso en que alguna de estas dos posiciones sean alteradas por un desborde de pila estos dos bytes tendrán un valor distinto de FFH y por lo tanto habrá ocurrido un desborde.

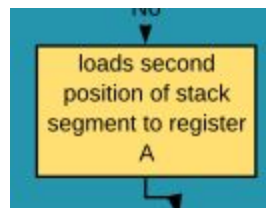
Para lograr esto la subrutina carga el contenido de la dirección 1821H, esta corresponde al primer byte del segmento destinado para la pila (*ver sección memoria*), en el acumulador.



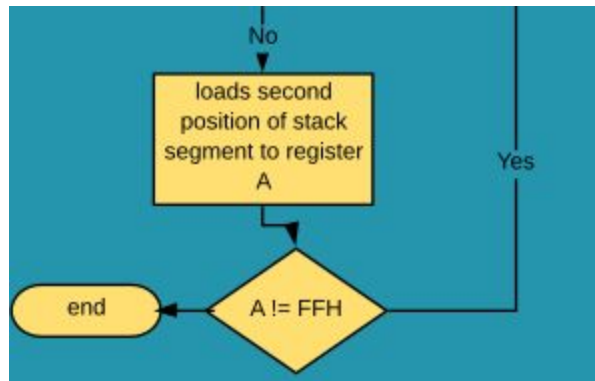
Luego compara el contenido del acumulador con el valor FFH, en caso de ser distinto ha habido un desborde de pila y se debe llamar a la función `STACK_O_ERROR`.



En caso contrario debemos mirar la dirección 1822H por lo que el contenido de la misma se carga al acumulador, esta corresponde al segundo byte del segmento destinado para la pila (*ver sección memoria*).



Finalmente comparamos de nuevo el contenido del acumulador con el valor FFH en caso de ser distintos se llamará a la rutina `STACK_O_ERROR` en caso contrario el chequeo ha terminado con éxito y no ha ocurrido un desborde de pila.



El diagrama de flujo llevado a código tiene la siguiente forma:

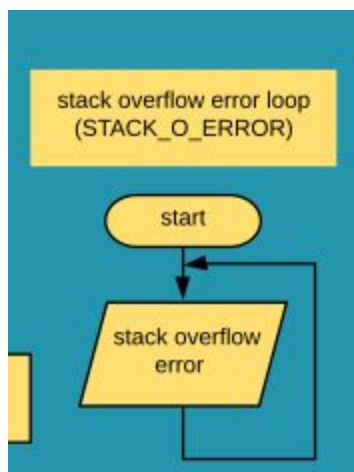
```

STACK_O_C:
;we save register A because is going to be modified
STA 1821H;

; we check the first flag
LDA 1828H
CPI FFH;
JNZ STACK_O_ERROR; if the flag is not FFH then a stack overflow error occurs
; we check second flag
LDA 1829H
CPI FFH;
JNZ STACK_O_ERROR; if the flag is not FFH then a stack overflow error occurs

LDA 1821H;there is no stack overflow we restore register A
RET;
  
```

Rutina de aviso de desborde de pila STACK_O_ERROR:



Rutina utilitaria del sistema que avisa si ha ocurrido un desborde de pila y detiene completamente la ejecución

del sistema monitor entrando en un ciclo infinito.

