

### 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 Conversión :

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

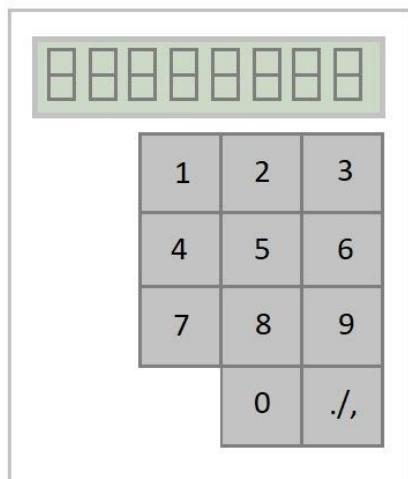
RATES NAMES CODES :	
USD -> COP:	1
USD -> VES:	2
COP -> USD:	3
COP -> VES:	4
VES -> USD:	5
VES -> COP:	6

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 única operación aritmética 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 ;-----;
;-----;

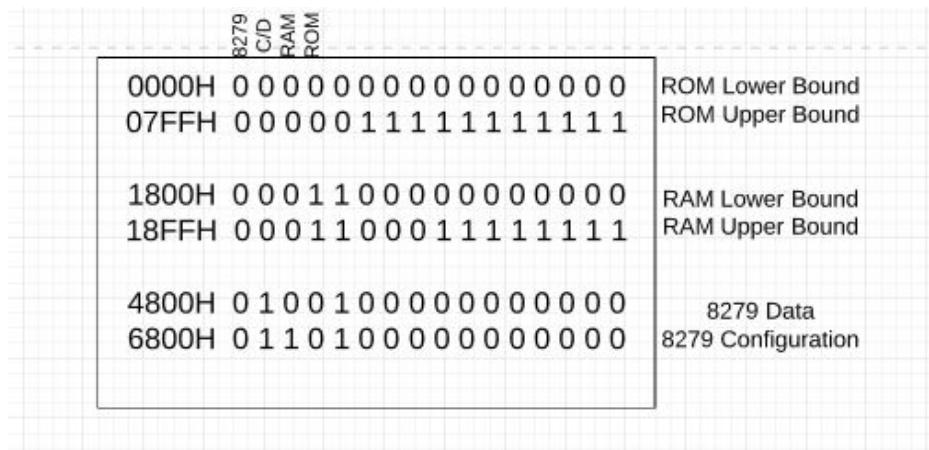
```

### 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 tambien 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:





### Mapa de Memoria ROM:

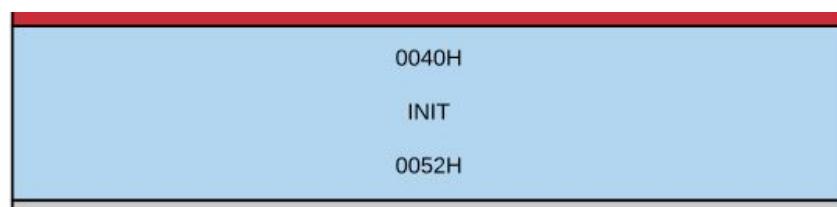
Las constantes e instrucciones se almacenan en la memoria **ROM** (ver mapa de memoria **ROM**), a continuación se dará una explicación detallada de cada uno de los segmentos que conforman la memoria **ROM**.

#### **Segmento 1:**



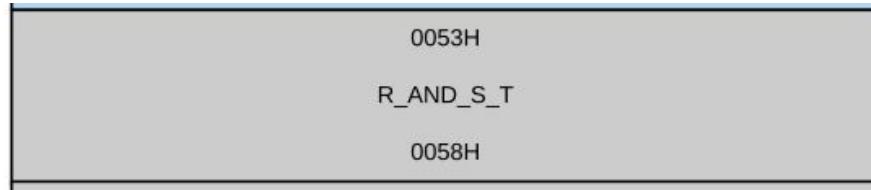
Zona de memoria **ROM** reservada por el 8085 es aquí donde se define el vector de interrupciones y el salto a la primera rutina del sistema monitor. El segmento va desde la posición en memoria 0000H hasta la posición 0039H.

#### **Segmento 2:**



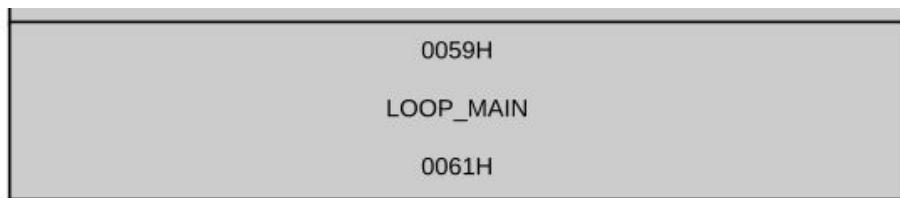
Segmento donde están guardadas las instrucciones de la rutina *INIT* (ver sección procesos). El segmento va desde la posición en memoria 0040H hasta la posición 0052H.

#### Segmento 3:



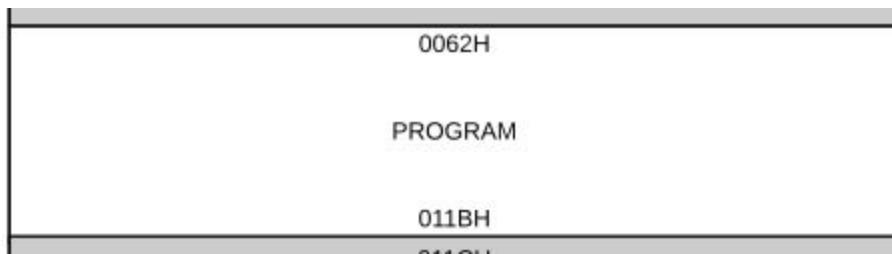
Segmento donde están guardadas las instrucciones de la rutina *R\_AND\_S\_T* (ver sección procesos). El segmento va desde la posición en memoria 0053H hasta la posición 0058H.

#### Segmento 4:



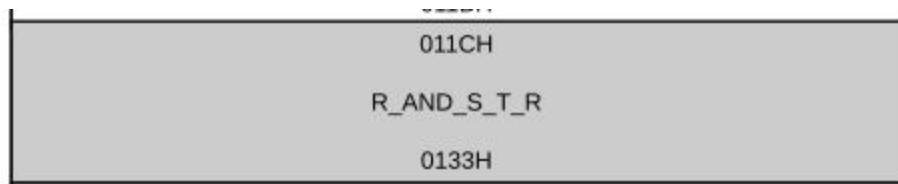
Segmento donde están guardadas las instrucciones de la rutina *LOOP\_MAIN* (ver sección procesos). El segmento va desde la posición en memoria 0059H hasta la posición 0061H.

#### 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:



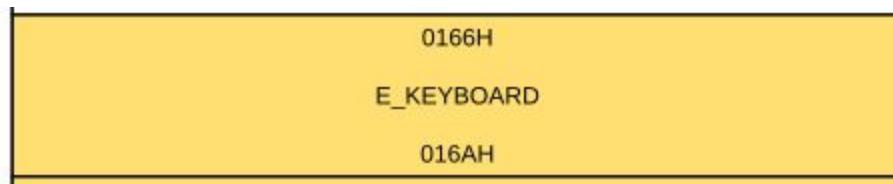
Segmento donde están guardadas las instrucciones de la rutina *R\_AND\_S\_T\_R* (ver sección procesos). El segmento va desde la posición en memoria 011CH hasta la posición 0133H.

#### 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.

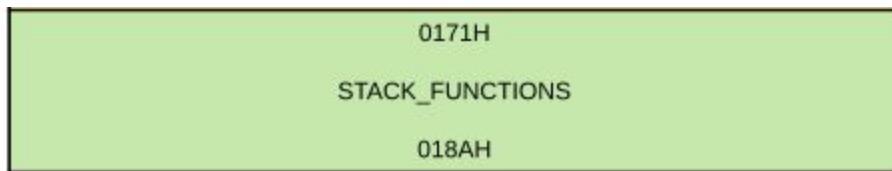
#### Segmento 10:



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

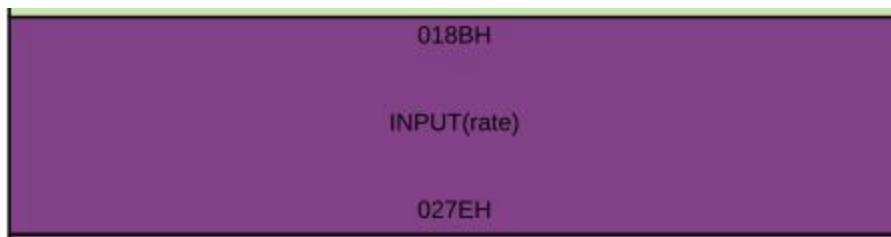
va desde la posición en memoria 016DH hasta la posición 0170H.

**Segmento 11:**



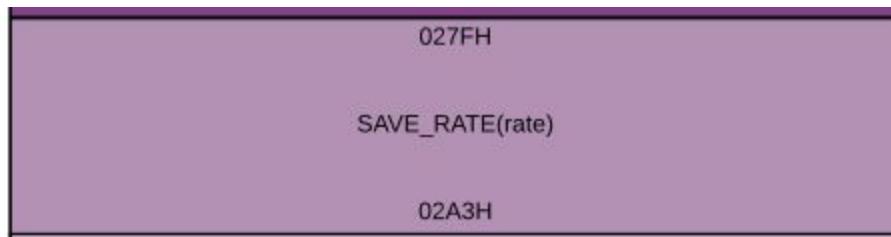
Segmento donde están guardadas las instrucciones de la rutinas *STACK\_FUNCTIONS* (*ver sección procesos*). El segmento va desde la posición en memoria 0171H hasta la posición 018AH.

**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:**



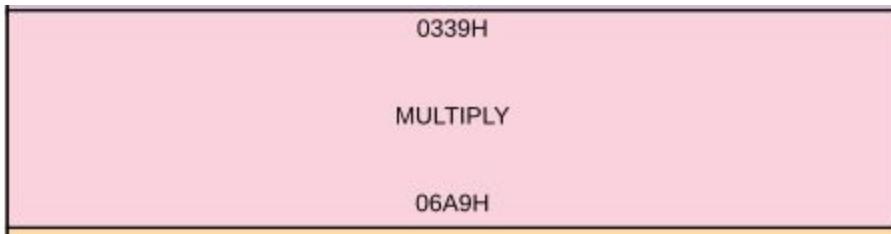
Segmento donde están guardadas las instrucciones de la rutina *SAVE\_RATE* (*ver sección procesos*). El segmento va desde la posición en memoria 027FH hasta la posición 02A3H.

**Segmento 14:**



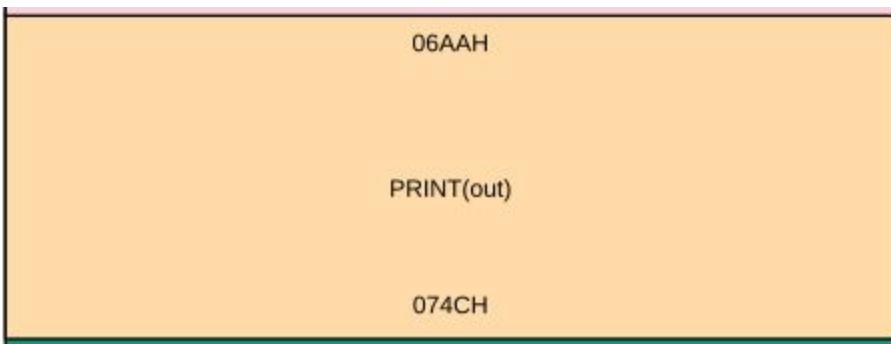
Segmento donde están guardadas las instrucciones de la rutina *ADJUST* (ver sección procesos). El segmento va desde la posición en memoria 02A4H hasta la posición 0338H.

**Segmento 15:**



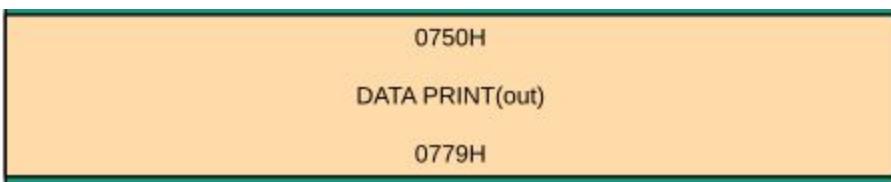
Segmento donde están guardadas las instrucciones de la rutina *MULTIPLY* (ver sección procesos). El segmento va desde la posición en memoria 0339H hasta la posición 06A9H.

**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.

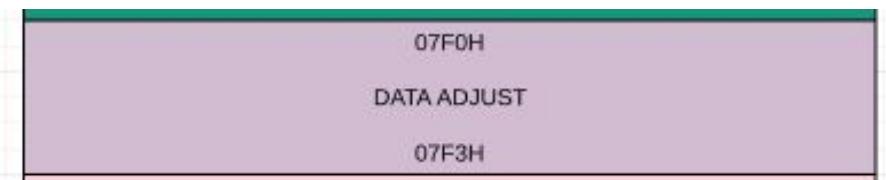
**Segmento 17:**



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.

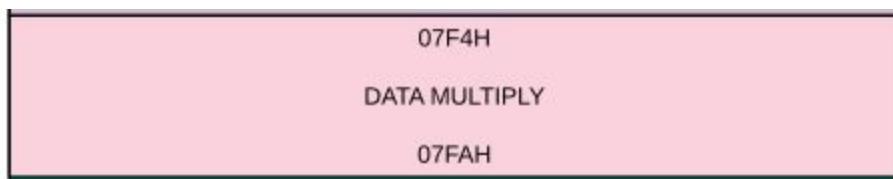
desde la posición en memoria 0750H hasta la posición 0779H.

#### **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:**



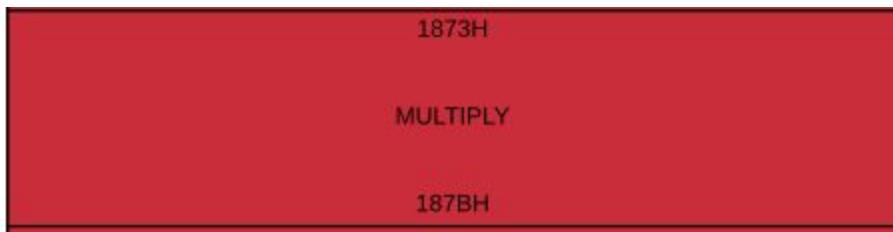
Segmento que comprende desde las posiciones de memoria 1828H hasta la 1868H, segmento reservado para la pila.

#### **Segmento 4:**



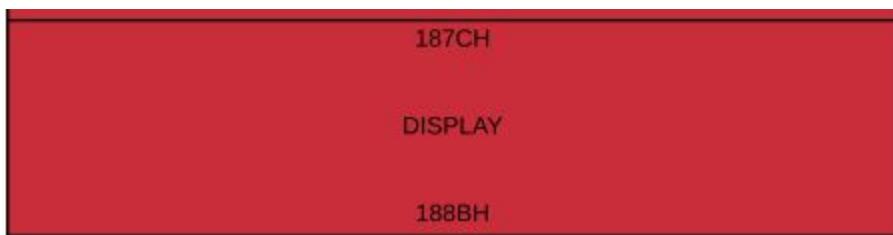
Segmento que comprende desde las posiciones de memoria 1869H hasta la 1872H, es la zona reservada para la rutina que lee del teclado *INPUT* (ver sección procesos). Este segmento es utilizado como buffer de entrada para el teclado, los números que son ingresados por el teclado son almacenados en esta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

### **Segmento 5:**



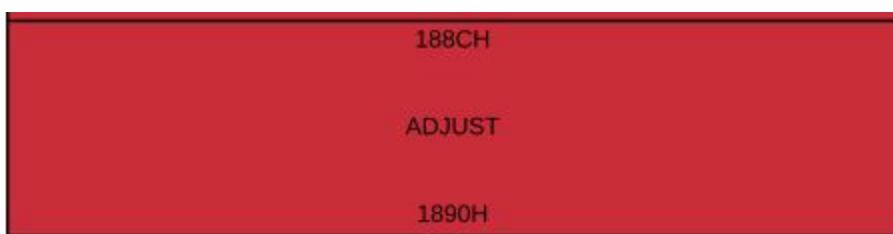
Segmento que comprende desde las posiciones de memoria 1873H hasta la 187BH, segmento reservado para la rutina *MULTIPLY* (ver sección procesos) es utilizado como buffer. La rutina *MULTIPLY* utiliza esta zona de memoria para guardar el resultado de la multiplicación.

### **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.

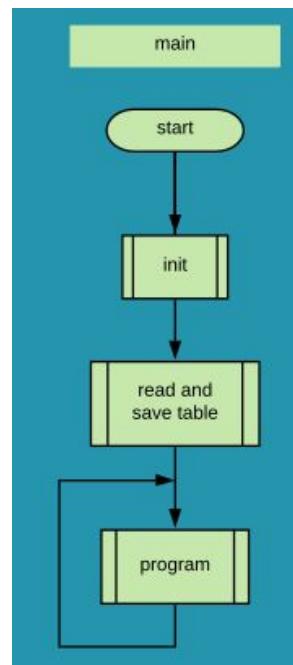
### Procesos:

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:**



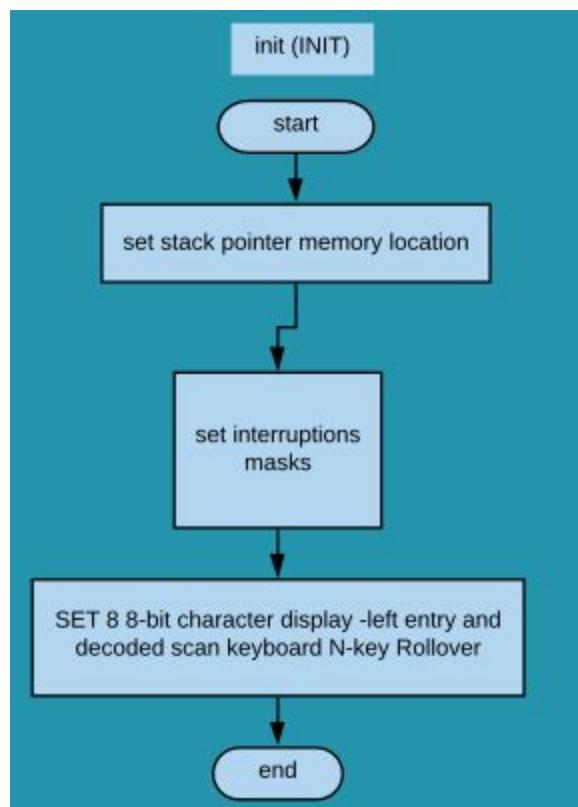
Este es el diagrama de flujo principal del sistema monitor y por lo tanto la vista de nivel superior del mismo.

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:

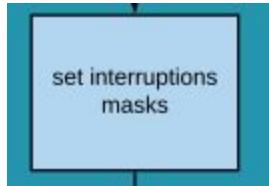
*Asignar al stack pointer la posición de memoria correspondiente:*



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;
```

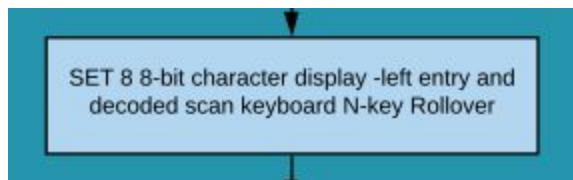
*Enmascarado de las interrupciones:*



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 point 5 interrupt  
SIM; apply the settings RTS masks
```

*Programación de componentes:*



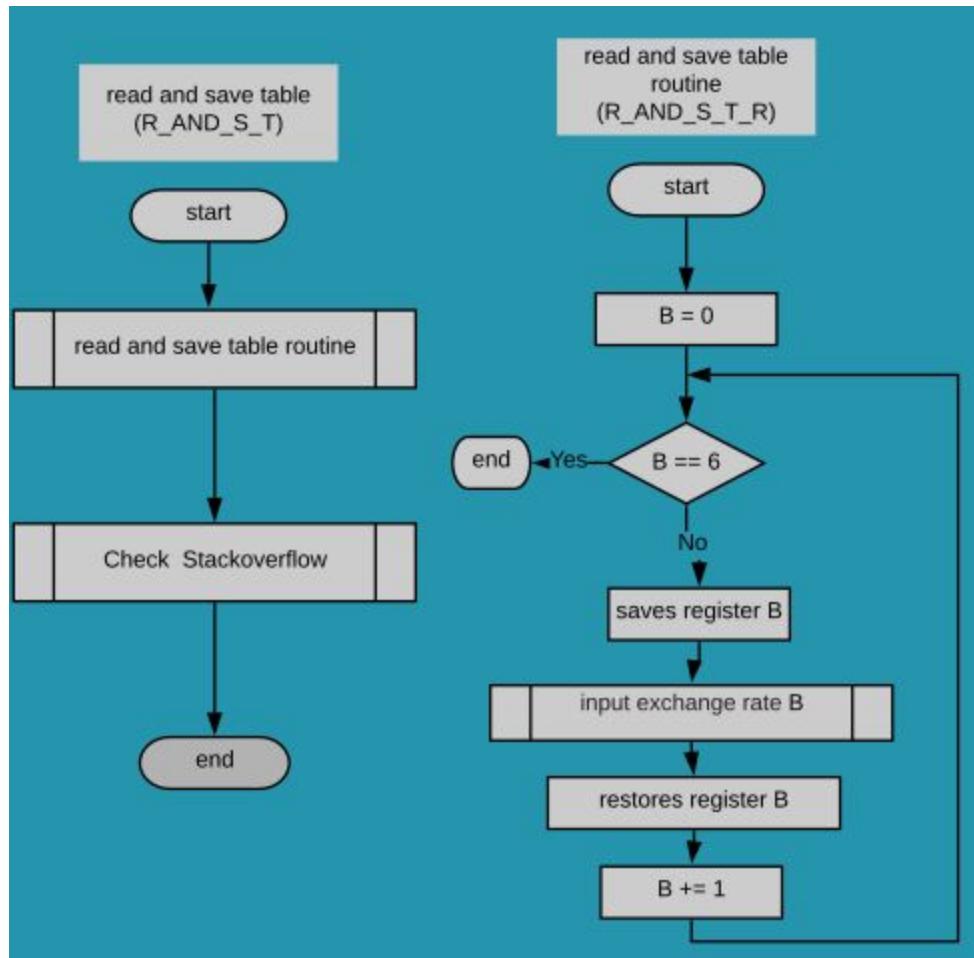
TronOs únicamente programa el 8279 ya que los demás componentes son usados con la configuración predeterminada de cada uno. Para el 8279 la configuración de pantalla usada es *8 8-bit character display left entry*. En el caso del teclado se utiliza la configuración *decoded scan keyboard N-key Rollover*, 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 ICS - 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
```

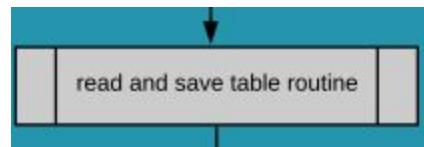
## Rutinas READ AND SAVE TABLE:

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*).

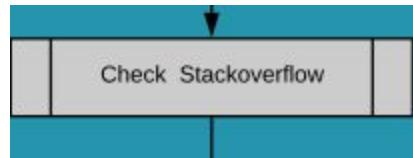


### Rutina read and save table (*R\_AND\_S\_T*):

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:

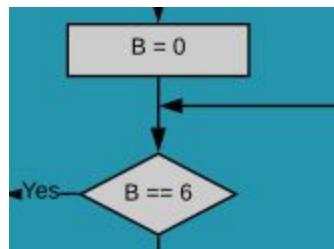
```

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

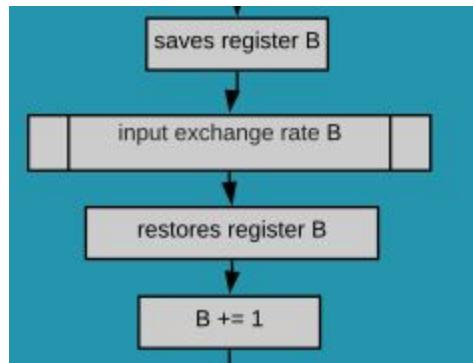
```

*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.

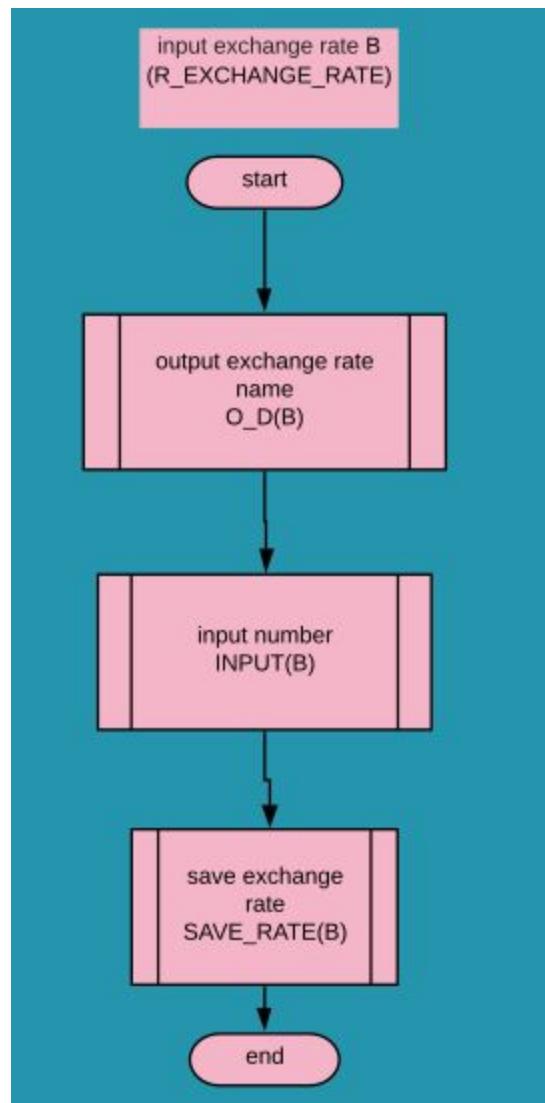


Luego en cada iteración se guarda el contenido del registro B en la zona reservada en memoria para ello (*ver sección memoria*), con el fin de que no se pierda en las llamadas a la subrutina de guardado de tasa de cambio, se llama a la rutina *R\_EXCHANGE\_RATE* (*ver sección input exchange rate*) una 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.



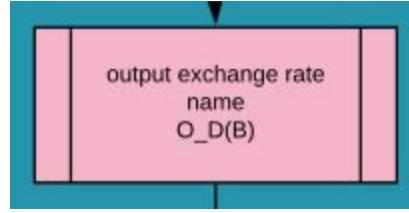
## Rutina input exchange rate B (R\_EXCHANGE\_RATE):

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.

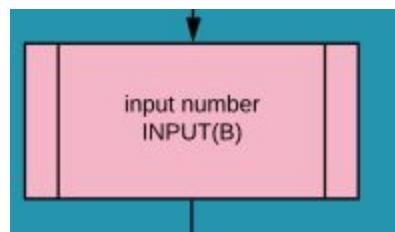
```
; PRINT_TO_DISPLAY pedimos que se ingrese el monto para el código B
MVI A, 43H; C
STA 187CH;
MVI A, 4FH; 0
STA 187DH;
MVI A, 44H; D
STA 187EH;
MVI A, 45H; E
STA 187FH;
MVI A, 00H; \n
STA 1880H;
CALL PRINT_ASCII
```

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
```

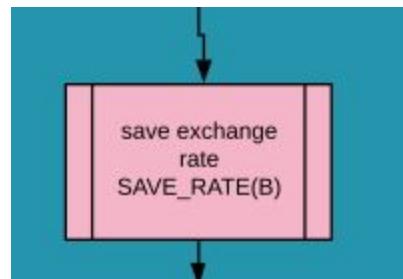
## 2.- se llama a la rutina INPUT.

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.

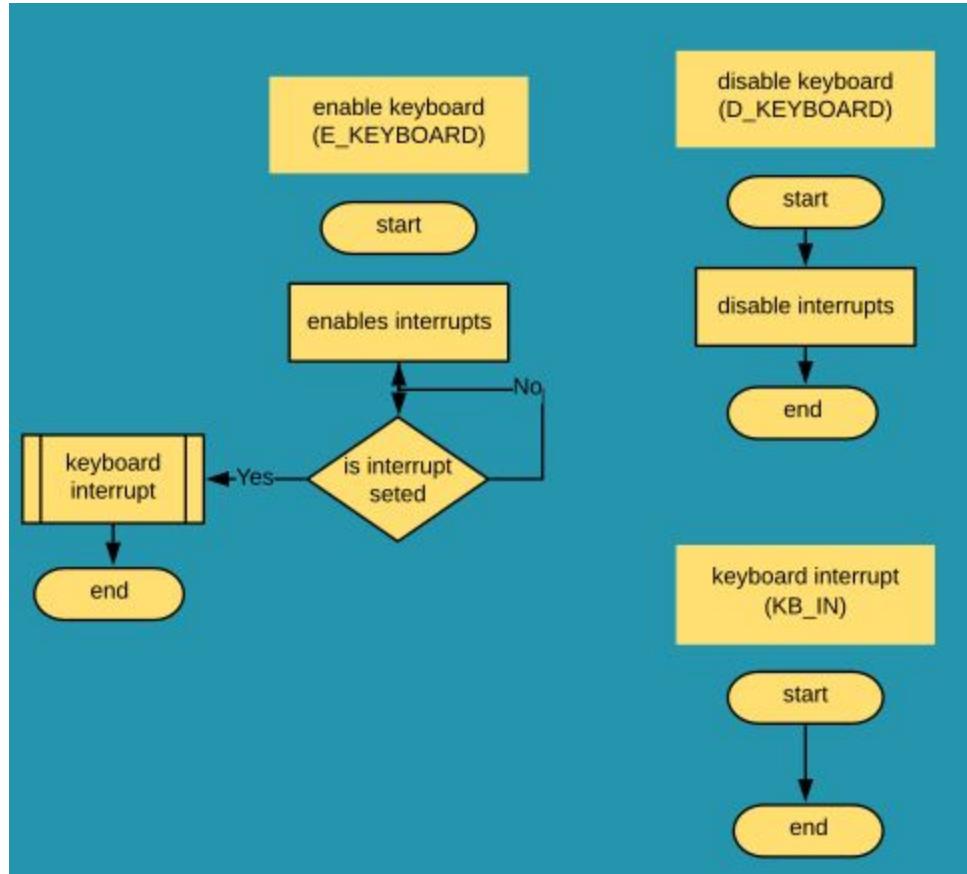


En código:

```
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:

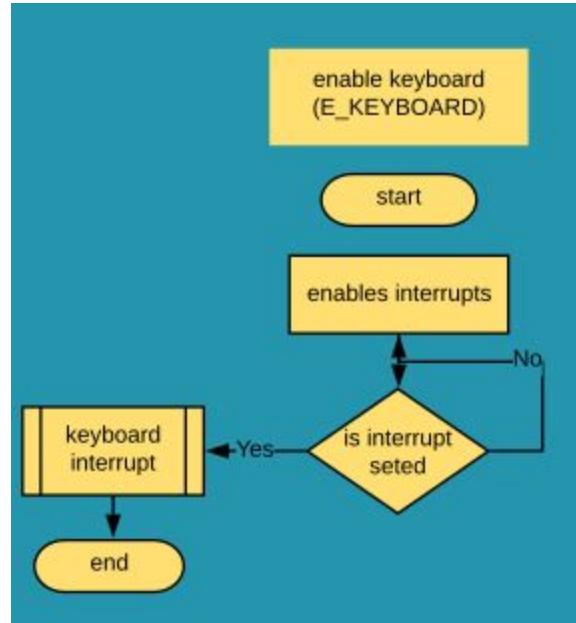


Con el fin de entender estas rutinas es preciso explicar brevemente el funcionamiento del teclado (*para más detalles ver sección INPUT*):

- 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*).

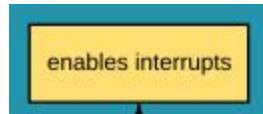
Ahora revisaremos más de cerca las rutinas utilitarias.

*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

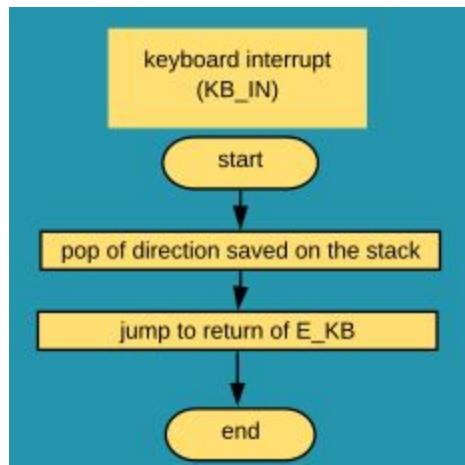
```

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 Cl. 23390971, Administrador de Recursos TronOS.*

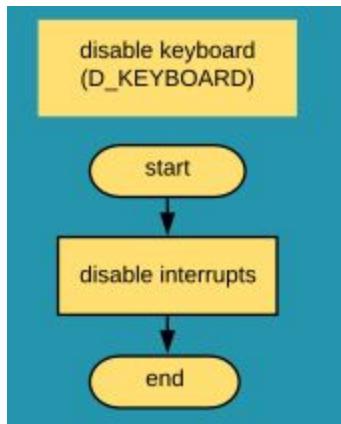
interrupciones. Sirve de “break” para el loop que se ejecuta en (*E\_KEYBOARD*).



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.

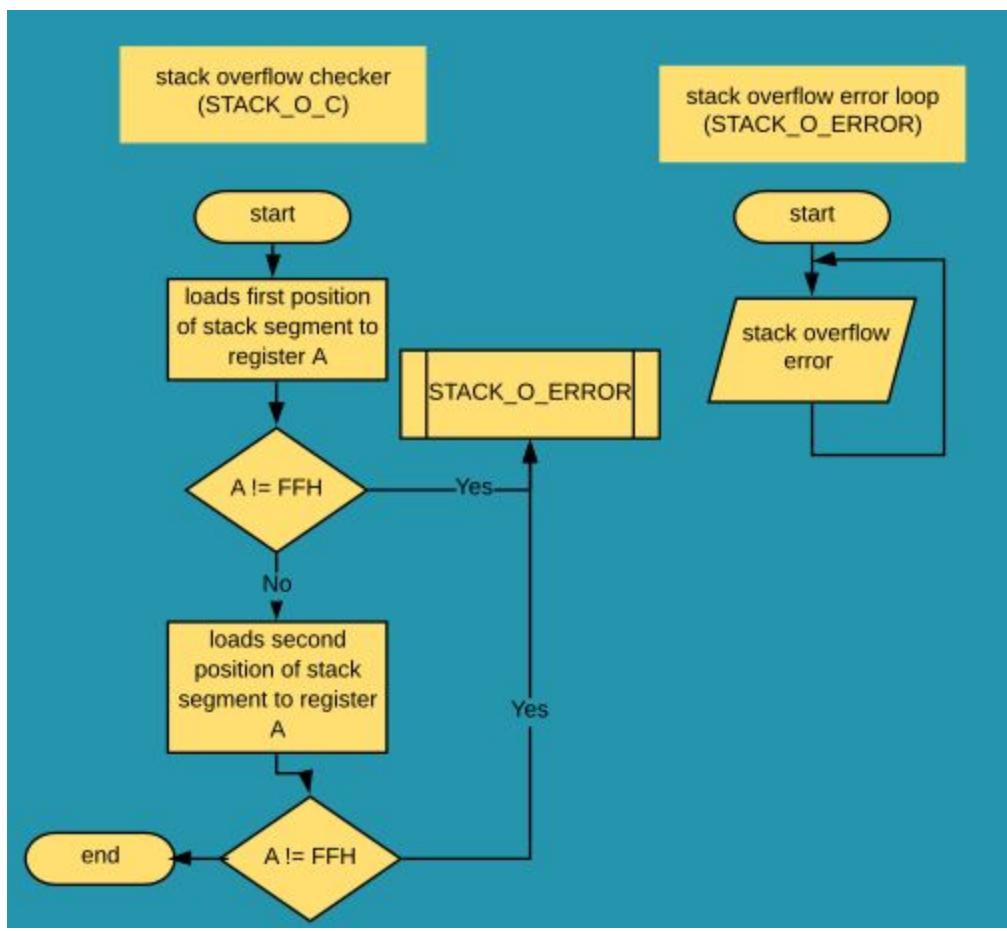
```

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.

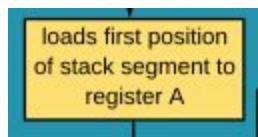


#### Rutina de chequeo de error STACK\_O\_C:

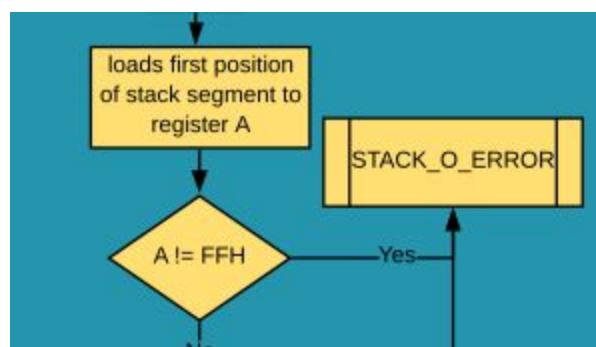
Subrutina encargada de verificar que la pila permanece sin desbordarse, TronOS en su inicialización guarda

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.

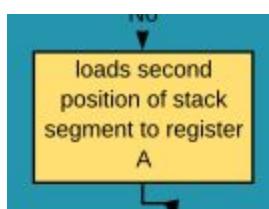
Para lograr esto la subrutina carga el contenido de la dirección 1821H, esta corresponde al primer byte del segmento destinado para la pila (ver sección memoria), en el acumulador.



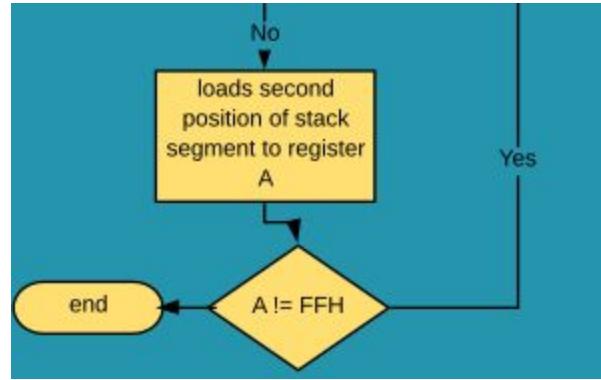
Luego compara el contenido del acumulador con el valor FFH, en caso de ser distinto ha habido un desborde de pila y se debe llamar a la función STACK\_O\_ERROR.



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:

```

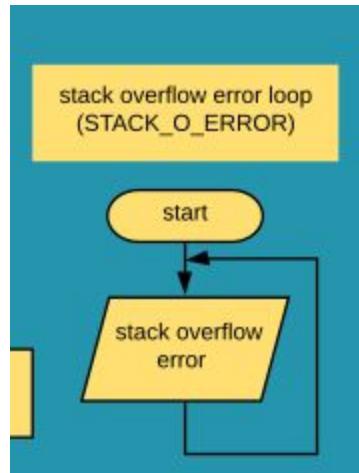
STACK_O_C:
;we save register A because is going to be modified
STA 1821H;

; 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;

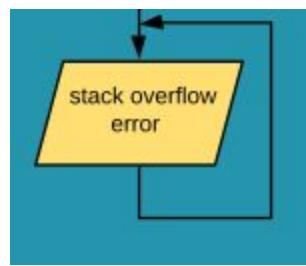
```

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.



## Rutina INPUT

La rutina INPUT tiene como función llevar los datos ingresados por el usuario desde el teclado, manejado por el controlador 8279, a la memoria principal (RAM 8156) y dejarlos disponibles en un espacio de memoria definido (buffer KEYBOARD en RAM) para su posterior uso.

Esta rutina (INPUT) hace uso de dos rutinas secundarias (INPUT\_LOOP y SAVE\_NC\_RAM) que permiten dividir en módulos el código. Hace uso de unas rutinas complementarias (S\_NUMBERC, S\_KEY\_ENTRY, S\_INIT, S\_INIT\_PORT\_POINTER, IN\_F8, IN\_F9, IN\_FA, IN\_FB, IN\_FC, IN\_FD, IN\_FE, IN\_FF) que son necesarias para generar una simulación de entrada de datos por teclado, ya que se está trabajando con un simulador. Y también hace uso de unas rutinas generales del monitor (E\_KEYBOARD y D\_KEYBOARD).

### Algunas definiciones

- **Carácter:** Cuando se haga referencia a un carácter en este contexto, se estará refiriendo a un byte de memoria que representa la posición de una tecla en el teclado controlado por el 8279, y en general representa la tecla en sí, es decir, un carácter. Los caracteres permitidos están definidos en la tabla del teclado.
- **Word-status:** Es una localidad de memoria dentro del controlador 8279 que contiene información del estado de la FIFO, como el número de caracteres por leer dentro de la FIFO. La dirección usada para leer el Word-status es 6800H, dirección de memoria dentro del controlador 8279.

### Contexto de la rutina INPUT

Para entender el contexto de la rutina INPUT hay que tener presente la configuración del monitor (los dispositivos que están conectados al microprocesador 8085), y en espacial los que están relacionados con la entrada de datos. La configuración de este Monitor tiene un teclado y un display conectados a los puertos de un controlador 8279 que a su vez esta conectado al microprocesador 8085, al que también está conectado una memoria RAM 8156.

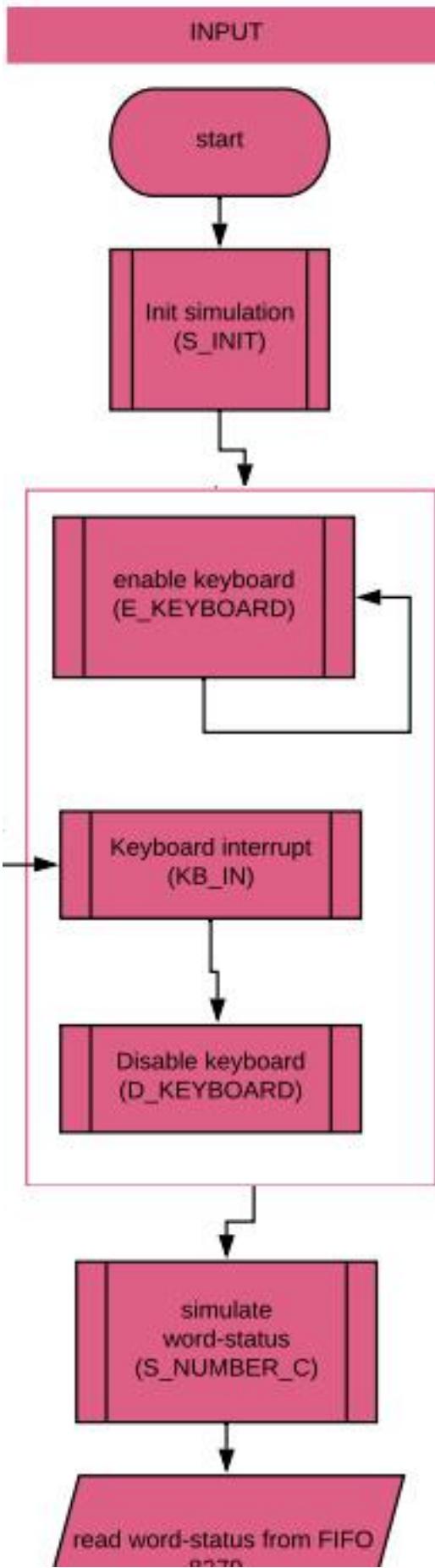
Por razones de diseño, el Monitor le va a pedir al usuario (usando la pantalla) en determinado momento que ingrese datos. Y el usuario tendrá que responder a esta solicitud, ingresando los datos requeridos a través del teclado. Datos que estarán formados por un conjunto de caracteres que el usuario ingresara al ir presionando las teclas. En este sentido, el controlador del 8279 se encarga de ir guardando en un buffer llamado FIFO las teclas presionadas por el usuario en orden de entrada (un máximo de 8 caracteres).

Entonces, ahí es donde se hace necesario la rutina INPUT, la cual se encarga de llevar los caracteres almacenados en el buffer FIFO del 8279, uno por uno, hasta la memoria principal RAM, y dejarlos disponibles en el buffer que se ha creado para esa función (buffer KEYBOARD).

### Formato del buffer KEYBOARD

Por razones de diseño, se ha definido que el buffer KEYBOARD, tendrá un espacio de 9 localidades de memoria en RAM, en donde en la primera localidad indicara el número de caracteres que se han leído. Y las siguientes direcciones tendrán los caracteres leídos desde la FIFO del 8279.

Las localidades definidas para el buffer KERYBOARD son 1869H hasta la 1872H, que son direcciones de RAM (dispositivo del tipo 8156). Ver el mapa de memoria para más información.



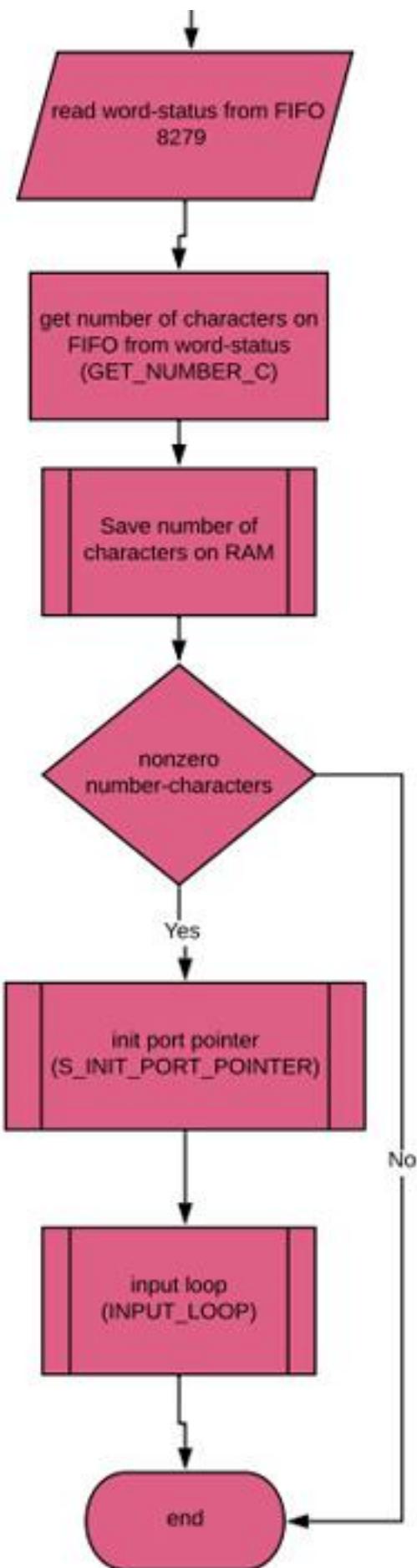
## RUTINA INPUT

**Condiciones de entrada:** La rutina input no tiene condiciones asociadas para poder ser llamada

**Condiciones de salida:** Cuando la rutina INPUT termina, en el buffer KEYBOARD va a estar disponible los datos ingresados por el usuario a través del teclado, según el formato especificado

### Procesos dentro de la rutina INPUT:

1. Al iniciarse, lo primero que hace es llamar a la subrutina complementaria que inicia la simulación (ver `S_INIT`).
2. Seguidamente se llama la rutina que habilita la interrupción por teclado (`E_KEYBOARD`), esta rutina genera un bucle, parando la ejecución de la rutina INPUT, del que solo se sale una vez se ha generado la interrupción por teclado indicando que hay información por leer en el controlador del teclado. Cuando dicha interrupción se genera, la rutina `KB_IN` (rutina general del monitor) intercepta la interrupción y devuelve el control a la rutina INPUT, que sigue su ejecución.
3. A continuación, ya que hay información que hay que leer proveniente del teclado, se deshabilita las interrupciones por teclado, al llamar la rutina general `D_KEYBOARD`.
4. Ya que se sabe que hay caracteres en la FIFO del controlador, lo primero que se hace es leer el **word-status** del controlador 8279 para saber, el estado de la FIFO y cuantos caracteres hay por leer. Como se está trabajando con un simulador, es necesario simular el Word-status, que se hace al llamar la rutina `S_NUMBER_C` (ver la rutina `S_NUMBER_C` para saber su funcionamiento).



5. Ya que se ha ejecutado la rutina que simula el Word-status, ahora si es posible leer el Word-status, por tanto se realiza esa acción de lectura.
6. El Word-status además de indicar el número de caracteres pendientes por leer en la FIFO del controlador, proporciona información adicional que para el caso no interesa, solo se quiere saber cuántos caracteres hay en la FIFO, y es por eso que se llama la rutina GET\_NUMBER\_C.

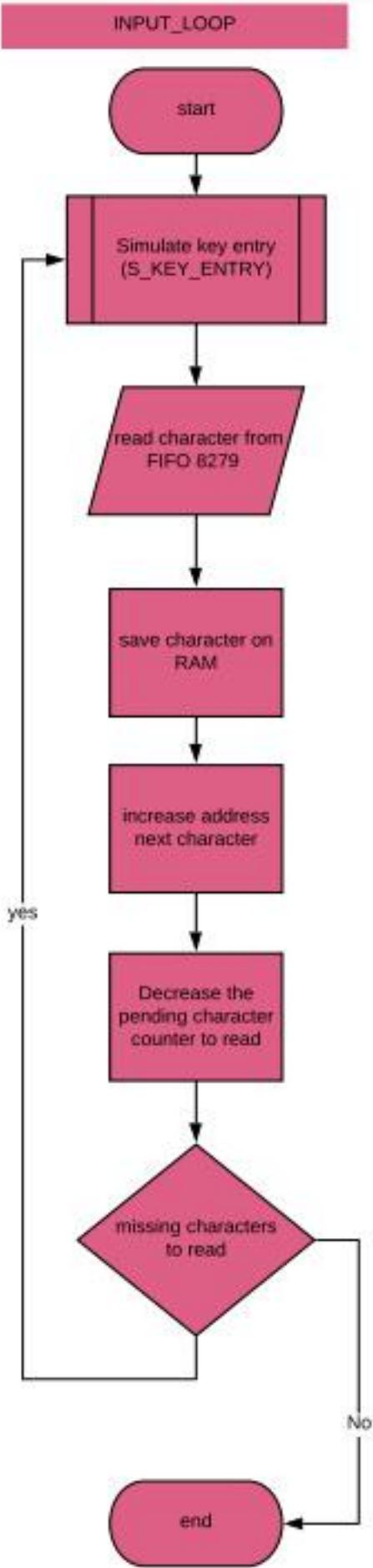
#### SUB-RUTINA GET\_NUMBER\_C

**Condiciones de entrada:** justo antes de llamar a esta rutina debe hacerse una lectura del Word-status que dejara en el registro A, la información correspondiente.

**Condiciones de salida:** luego de que esta rutina termine, deja en el registro B el número de caracteres que hay disponibles en la FIFO del 8279 para ser leídos.

#### Procesos dentro de la rutina GET\_NUMBER\_C:

- 1) Por simplificaciones de la simulación, el Word-status solo contiene el número de caracteres disponibles en la FIFO del 8279 para ser leídos. Por lo tanto, esta rutina solo mueve la información del Word-status del registro A al registro B
7. Seguidamente se llama a la rutina SAVE\_NC\_RAM (save number of characters on RAM) que guarda el número de caracteres disponibles para ser leídos, en RAM. Ver la rutina para más información.
8. En este punto, se da una decisión, si el número de caracteres disponibles para ser leídos es igual a cero, se salta al final de la rutina y la rutina INPUT termina. De lo contrario, se sigue la ejecución de la rutina INPUT.
9. A continuación, para proceder a leer cada uno de los caracteres disponibles en la FIFO del 8279, es necesario llamar a una rutina que tiene que ver con la simulación de la entrada de datos, ver la rutina S\_INIT\_PORT\_POINTER para más información.
10. Y por último, se llama a la rutina secundaria INPUT\_LOOP la cual genera un bucle que va leyendo carácter por carácter y los va guardando en las localidades de memoria correspondientes en el buffer KEYBOARD de la memoria RAM, ver rutina INPUT\_LOOP para más información.



## SUB-RUTINA INPUT\_LOOP

**Condiciones de entrada:** La rutina INPUT\_LOOP para ser llamada necesita que la precedan las ejecuciones de las rutinas GET\_NUMBER\_C, SAVE\_NC\_RAM y S\_INIT\_POINT\_POINTER.

- Ejecutadas esas rutinas, en el registro B va a estar el número de caracteres disponibles para ser leídos en el 8279, el cual se usara como contador y se ira decrementando a medida que se lee cada carácter.
- También estará inicializado el puntero (en los registros D y E) que indicara la dirección de memoria en RAM donde se guardara el carácter que se está leyendo.

**Condiciones de salida:** Cuando la rutina INPUT\_LOOP termina, en el buffer KEYBOARD va a estar disponible los datos ingresados por el usuario a través del teclado, según el formato especificado.

En cada ciclo que realiza la rutina INPUT\_LOOP ira guardando un carácter leído en el buffer KEYBOARD de la memoria RAM.

### Procesos dentro de la rutina INPUT\_LOOP:

Esta rutina realiza un número de ciclos que están determinados por el número de caracteres disponibles para ser leídos. En cada ciclo se realizan los siguientes procesos:

1. Se llama a la rutina S\_KEY\_ENTRY que simula la entrada de un carácter, al poner el carácter en la dirección del controlador 8279, que se usa para leer la FIFO (4800H). Ver la rutina S\_KEY\_ENTRY para más información.
2. Se hace una operación de lectura sobre la dirección 4800H, es decir, se hace una operación de lectura a la FIFO del 8279 (teniendo en cuenta que cada vez que se hace una lectura a la FIFO del 8279, el controlador elimina ese carácter, para tener disponible en la misma dirección el siguiente carácter disponible. El controlador 8279 se ha programado para que funcione de esta forma).
3. Seguidamente, se guarda el carácter leído, que se encuentra en el registro A, en la memoria RAM en la posición que indican los registros D y E.
4. Seguidamente se incrementa en una posición, el puntero (formado por los registros D y E) que indica la posición de memoria en la que se va a guardar el carácter que se va leer en el próximo ciclo.
5. Se decremento el contador (registro B) que indica el numero de caracteres que quedan disponibles por leer de la FIFO.
6. Se evalúa si quedan caracteres por leer en FIFO para mantener el ciclo o salir del ciclo y de la rutina. Esto se hace comparando si el registro B es cero. De ser cero indicaría que ya no hay caracteres por leer y la rutina saldría del ciclo y terminaría.

## SAVE\_NC\_RAM



## SUB-RUTINA SAVA\_NC\_RAM

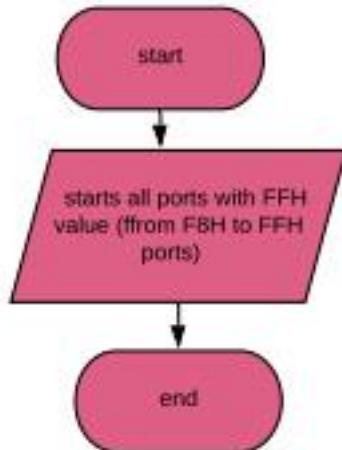
**Condiciones de entrada rutina SAVE\_NC\_RAM:** Depende de la rutina GET\_NUMBER\_RAM para poder ser ejecutada.

**Condiciones de salida rutina SAVE\_NC\_RAM:** Cuando esta rutina termina, en la primera posición (1869H en RAM) del buffer KEYBOARD, estará disponible número de caracteres que contendrá el buffer luego que la rutina INPUT termine.

### Procesos dentro de la rutina SAVE\_NC\_RAM:

1. Guarda el número de caracteres disponibles para leer de la FIFO, que para cuando se ejecuta esta rutina está en el registro B del microprocesador. En la primera localidad de memoria (1869H en RAM) del buffer KEYBOARD.
2. Estable la dirección 186AH en RAM como la localidad de memoria donde se guardara el primer carácter leído de la FIFO. Esto lo hace en los registros D y E, que funcionaran como un puntero que ira indicando donde se guardara cada carácter que se vaya leyendo de la FIFO del 8279.

## S\_INIT



## RUTINA COMPLEMENTARIA S\_INIT

**Condiciones de entrada rutina S\_INIT:** No depende de ninguna otra rutina, aunque si debe ejecutarse al inicio de la simulación

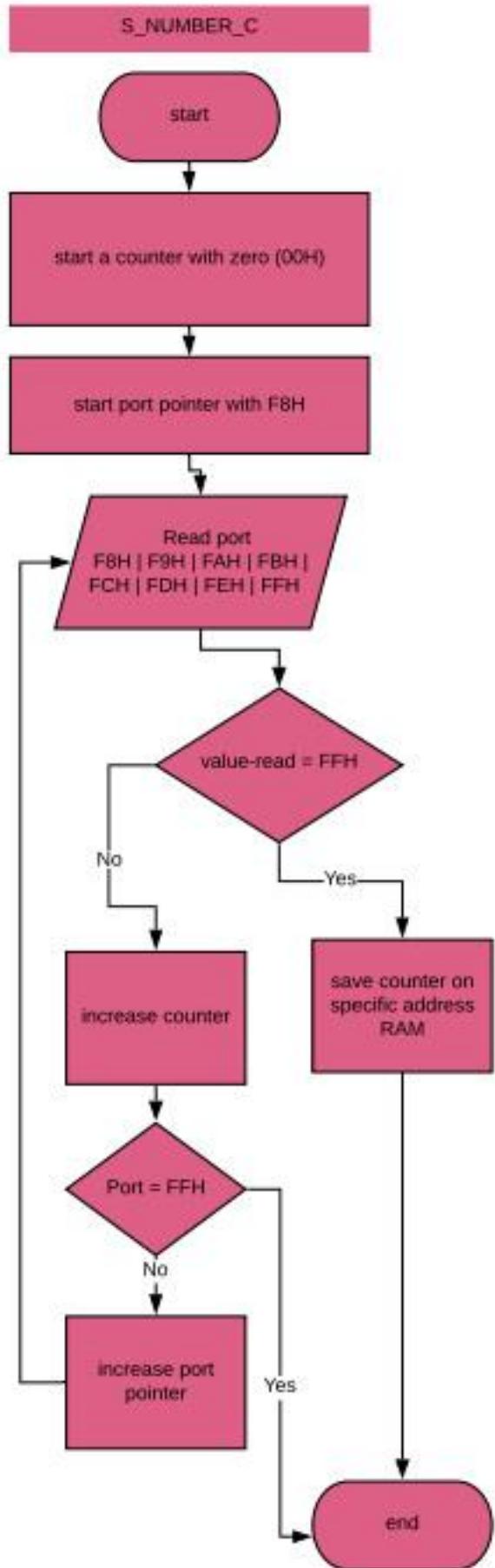
**Condiciones de salida rutina S\_INIT:** Cuando esta rutina termine los puertos (lectura/escritura) desde el F8H hasta el FFH del microprocesador, deben tener el valor FFH.

### Procesos dentro de la rutina S\_INIT:

1. Hace operaciones de escritura en los puertos del F8H al FFH

### NOTA IMPORTANTE rutina S\_INIT:

- Esta rutina se implementó bajo las condiciones con las que trabajaba el simulador **Jubin para el microprocesador 8085**, el cual implementa que los puertos de lectura/escritura son los mismos. Por razones explicadas en otro apartado, se tuvo que cambiar a otro simulador, el cual varia la forma en como tiene implementados los puertos. Los puertos lectura están separados de los puertos de escritura, lo que hace que esta rutina que la rutina no funcione y el usuario tenga que hacer manualmente esta parte de la simulación.



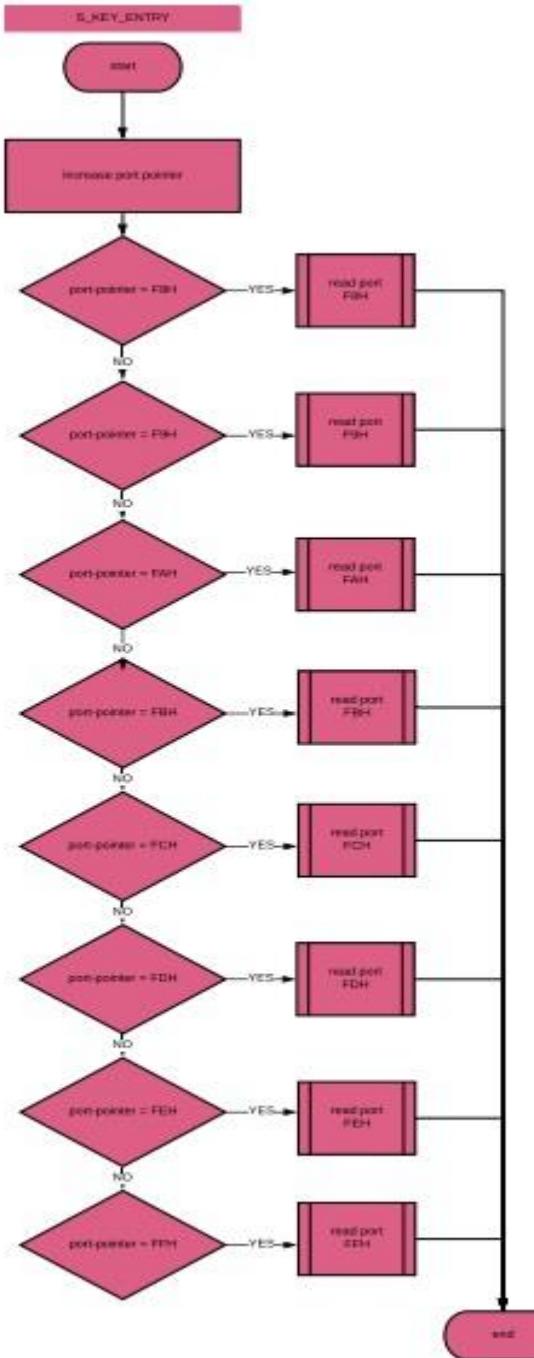
## RUTINA COMPLEMENTARIA S\_NUMBER\_C

**Condiciones de entrada rutina S\_NUMBER\_C:** Esta rutina simula el Word-status, por razones de simulación, es necesario simular la FIFO del 8279 también, la cual se simula con los últimos 8 puertos de lectura/escritura del microprocesador (del F8H al FFH). Entonces cuando esta rutina se vaya a ejecutar es necesario que en los puertos del F8H hasta el FFH, estén los caracteres ingresados por el usuario (número hexadecimales diferente de FFH, y en general que estén dentro de los definidos en la tabla del teclado), hasta un máximo de 8 caracteres. Si el usuario ha ingresado menos de 8 caracteres, los restantes deben rellenarse con el carácter definido como vacío FFH. Esto simulación de ingresar los caracteres en cada puerto que se realiza de forma manual por quien ejecuta el código y funge como usuario, debe realizarse en el momento en el que la ejecución de la rutina INPUT ha caído en el bucle generado por la rutina E\_KEYBOARD. Luego de ingresar los caracteres en el puerto también debe generarse la interrupción de teclado manualmente.

**Condiciones de salida rutina S\_NUMBER\_C:** Cuando la rutina S\_NUMBER\_C termina, se ha simulado el Word-status, y por lo tanto, la información del Word-status se deja en la dirección de lectura del mismo (6800H, dirección memoria del 8279), para que seguidamente se de la operación de lectura del Word-status.

### Procesos dentro de la rutina S\_NUMBER\_C:

1. Se establece el registro C como un contador que se inicia en 0
2. Se establece un Puntero y se inicia en F8H (ya que son los puertos del F8H con los que se están simulando la FIFO)
3. Se lee el valor del puerto que está siendo apuntado
4. Si el valor es diferente de FFH, se suma uno al contador. De lo contrario, se salta al proceso 6.
5. Si el puerto que está siendo apuntado es igual a FFH, es el último puerto, sería la última localidad de la FIFO y consecuencia, se sale del bucle y se realiza el proceso 6. De lo contrario se aumenta en uno el apuntador y se re repiten los procesos 3, 4 y 5.
6. En este punto el contador contiene el número de caracteres que hay disponibles en FIFO, es decir, representa al Word-status por lo tanto se guarda esa información en la dirección 6800H, simulando el Word-status.



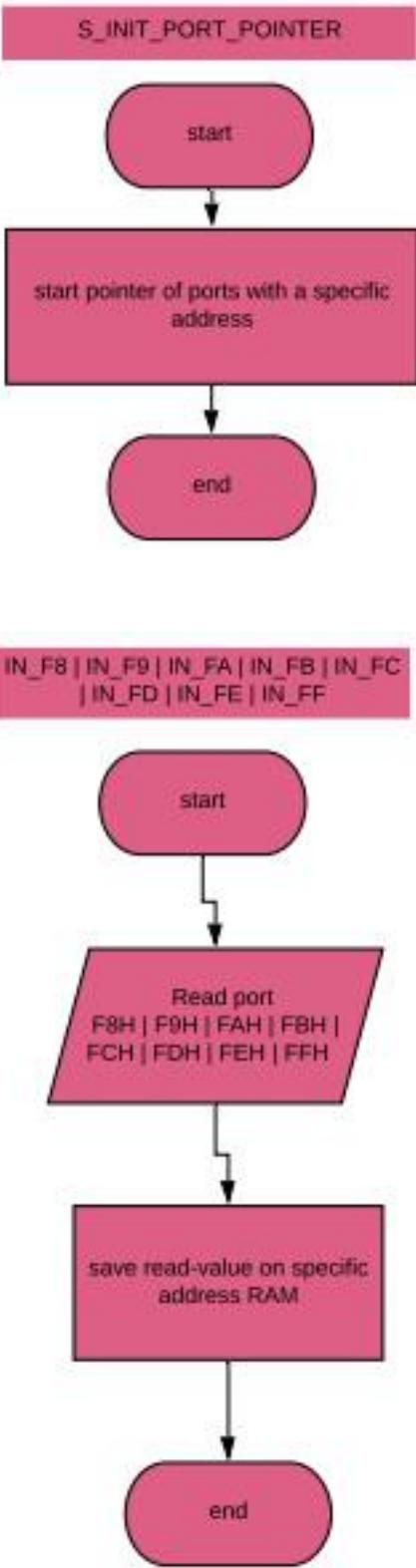
## RUTINA COMPLEMENTARIA S\_KEY\_ENTRY

**Condiciones de entrada rutina S\_KEY\_ENTRY:** La rutina S\_KEY\_ENTRY depende de la rutina S\_INIT\_PORT\_POINTER que inicializa el puntero que usa esta rutina. Y debe ser llamada dentro del contexto de la rutina INPUT\_LOOP

**Condiciones de salida rutina S\_KEY\_ENTRY:** Cuando la rutina termina S\_KEY\_ENTRY, el dato leído por el puerto que ha coincidido se guarda en la dirección 4800H (dirección de lectura de datos del controlador 8279), para que seguidamente se lea esa dirección como si se estuviese leyendo un carácter de la FIFO.

### Procesos dentro de la rutina S\_KEY\_ENTRY:

1. Compara el puntero que indica el puerto que debe ser leído con cada uno de los posibles puertos que pueden leerse (del F8H al FFH). Para el puerto que coincide se ejecuta la correspondiente rutina que lee el puerto (IN\_F8, IN\_F9,..., IN\_FF)



## RUTINA COMPLEMENTARIA S\_INIT\_POINTER

**Condiciones de entrada rutina S\_INIT\_POINTER:** No depende de ninguna otra rutina. Debe llamarse antes de la rutina INPUT\_LOOP. **Condiciones de salida rutina S\_INIT\_POINTER:** Establece el registro C como un puntero que indica un puerto a leer. Este puntero lo usa la rutina S\_KEY\_ENTRY.

### Procesos dentro de la rutina SAVE\_NC\_RAM:

1. Inicia el puntero (registro C) con el valor F7H

## RUTINAS COMPLEMENTARIAS IN\_F8, IN\_F9, IN\_FA, IN\_FB, IN\_FC, IN\_FD, IN\_FE, IN\_FF

Las rutinas IN\_F8, IN\_F9, IN\_FA, IN\_FB, IN\_FC, IN\_FD, IN\_FE, IN\_FF son rutinas separadas que implementan la misma funcionalidad para un puerto diferente.

**Condiciones de entrada rutina IN\_Fx, para x = 8,9,..., F:** La rutina IN\_Fx depende de la rutina S\_KEY\_ENTRY debe llamarse dentro de ese contexto

**Condiciones de salida rutina IN\_Fx, para x = 8,9,..., F:** Cuando la rutina IN\_Fx termina, el dato leído por el puerto Fx se guarda en la dirección 4800H (dirección de lectura de datos del controlador 8279).

### Procesos dentro de la rutina IN\_Fx, para x = 8,9,..., F:

1. Ejecuta una operación en el puerto Fx para x = 8, 9,..., F.
2. El valor leído en el puerto Fx, es guardado en la dirección 4800H.

## **PRINT\_ASCII**

Permite al usuario imprimir una cadena de caracteres en pantalla, utilizando el chip controlador de display 8279 para lograrlo, ofreciendo una abstracción que encapsula los detalles asociados al hardware.

### **ESPECIFICACION DEL BUFER**

La cadena a imprimir debe escribirse como una secuencia de bytes consecutivos en memoria, el espacio de memoria designado para la cadena a imprimir es un array de 16 bytes localizado en la dirección **827CH**.

### **CARACTERES SOPORTADOS**

La cadena a imprimir debe estar compuesta solamente de caracteres soportados, una violación a esto causara que la CPU en un lazo infinito en una rutina de **ERROR**, los caracteres soportados son los siguientes:

```
{ A, B, C, D, E, F, G, H, J, L, O, P, S, R, T, ' ' }
```

### **REPRESENTACIÓN DE LOS CARACTERES EN MEMORIA**

Cada carácter soportado tiene asignado un valor numérico que lo representa, el valor asociado a cada carácter es el mismo que especifica el estándar **ASCII**, con la unica excepcion del espacio en blanco, cuyo valor numerico es **20H**.

### **LONGITUD DE CADENA Y CARACTER DE FIN DE CADENA**

La longitud de toda cadena esta naturalmente limitada por el tamaño del bufer, cuya capacidad máxima es de 16 caracteres.

Sin embargo, para imprimir cadenas de menor longitud, se debe señalar el fin de la cadena con un carácter especial **00**, este byte que marca el final de la cadena, se deberá colocar en el byte próximo al ultimo carácter de la cadena a imprimir.

### **EJEMPLOS DE USO**

Esta secuencia de instrucciones imprime "DAFT" en la pantalla de display

```
MVI A, 44H ; D  
STA 827CH  
MVI A, 41H ; A  
STA 827DH  
MVI A, 46H ; F  
STA 827EH  
MVI A, 54H ; T  
STA 827FH  
MVI A, 00H ; FIN DE CADENA  
STA 8280H
```

```
CALL PRINT_ASCII
```

## DETALLES DE IMPLEMENTACIÓN

La rutina envía un comando al chip de display, que se encuentra mapeado en memoria bajo la dirección **6800H**.

El comando especifica que se estará escribiendo en la RAM de display de manera secuencial.

Para cada carácter en la cadena a imprimir, se enviará a la dirección de memoria **4800H** el byte correspondiente que representa al carácter de manera visual.

## EJEMPLO ENTRADA / SALIDA

Bufer = [ 'D', 'A', 'F', 'T', 00, ... ]

Secuencia que sera escrita en la dirección 4800H: [ 0CH, 88H, E8H, ECH ]

## PRINT\_NUMBER

Permite al usuario imprimir un numero decimal en pantalla, utilizando el chip controlador de display 8279 para lograrlo, ofreciendo una abstracción que encapsula los detalles asociados al hardware.

## ESPECIFICACION DEL BUFER

El numero a imprimir debe escribirse como una secuencia de bytes consecutivos en memoria, el espacio de memoria designado para esto es un array de 16 bytes localizado en la dirección **827CH**.

## FORMATO

El numero a imprimir debe cumplir con las siguientes reglas, que le permiten tener una representacion exacta como una secuencia de bytes almacenada en el bufer.

El primer byte de la secuencia indica la cantidad de digitos que conforman el numero (el punto decimal se debe tomar en cuenta).

Cada byte a excepcion del primero, contiene el valor del correspondiente digito, sin embargo, el penultimo byte siempre sera **40H**, valor que representa la coma.

## EJEMPLOS DE USO

La siguiente secuencia representa 1560.48: [ 07, 01, 05, 06, 00, 40H, 04, 08 ]

La siguiente secuencia representa 5.99: [04, 05, 40H, 09, 09 ]

La rutina se ajusta perfectamente al formato especificado, no cumplir con el, hara que la CPU acabe en un lazo infinito en una rutina de **ERROR**.

Todo numero que se desee imprimir debe contener al menos, 4 digitos y una coma.

## DETALLES DE IMPLEMENTACION

La rutina envia un comando al chip de display, que se encuentra mapeado en memoria bajo la dirección **6800H**.

El comando especifica que se estará escribiendo en la RAM de display de manera secuencial.

Para cada digito correspondiente al numero, se enviara a la dirección de memoria **4800H** el byte correspondiente que representa al digito de manera visual.

En el caso de la coma, el codigo de segmentos que representa al antepenultimo digito sera alterado para incluirla, pues el punto decimal se trata de uno de los siete segmentos que pueden ser encendidos.

## EJEMPLO ENTRADA / SALIDA

Bufer = [ 07, 01, 05, 06, 00, 40H, 04, 08 ]

Secuencia que sera escrita en la dirección 4800H: [ FCH, 29H, 28H, 04H, 19H, 10H ]

## ADJUST

Esta subrutina se creó con el propósito de ajustar el número a un formato específico(XX,XX) ante la entrada de un usuario, esto quiere decir que si el usuario ingresa un número sin decimales por ejemplo **12** internamente haciendo uso de esta rutina el número será **12,00** cabe destacar que para efecto de este sistema(monitor) cada número está representado en un registro que tiene el tamaño de 1 byte(8 bits), así mismo si el usuario ingresa un número(con parte entera) que no cumpla con dicho formato(XX,XX; dos enteros y dos decimales separados por coma) esta rutina realizará el ajuste ya mencionado, otro ejemplo para ilustrar esto; el usuario ha ingresado **1,2** al hacer uso de ADJUST el número pasará a tener la siguiente representación: **01,20**, esto se realiza con el fin de trabajar sobre un formato bien definido donde siempre operaremos un número de 5 caracteres(la coma se incluye como carácter).

### Funcionamiento

Se toma el número de caracteres que ha sido guardado por el 8279 en la posición 1869h de memoria, teniendo esto luego se realiza el ajuste según el número de dígitos que hayan sido guardados, una vez obtenido el número ajustado será guardado en una posición de memoria específica donde cada digito junto con el carácter “coma” (representado por 40h) estarán ubicados continuamente, estas posiciones de memoria serán descritas en el apartado siguiente ,cabe destacar que en caso de que hayan sido ingresado los 5 caracteres(4 números- dos en la parte entera y dos en la parte decimal-y la coma) se dejara tal y como esta simplemente se guardarán en la posición de memoria antes mencionada.

Para saber cuántos números hay en el buffer donde el 8279 guarda sus dígitos es necesario realizar una comparación, dicha comparación se realiza con números que previamente han sido guardados en partes específicas de la memoria y que cumplen esta función, las posiciones de la memoria donde se guardan dichos dígitos son las siguientes:

07FBh	01h
07FCh	02h
07FDh	03h
07FEh	04h
07FFh	05h

Tabla A1-Posiciones de dígitos auxiliares para la comparación.

La posición donde se guardara el buffer que representa el digito va desde la posición 188Ch hasta la 1890h con el siguiente formato:

Posición de memoria	Valor	Descripción del elemento guardado
<b>188Ch</b>	<b>0Xh</b>	Primer digito entero
<b>188Dh</b>	<b>0Xh</b>	Segundo digito entero
<b>188Eh</b>	<b>40h</b>	Coma
<b>188Fh</b>	<b>0Xh</b>	Primer digito parte decimal
<b>1890h</b>	<b>0Xh</b>	Segundo digito parte decimal

Tabla A2-Buffer donde guarda los dígitos el adjust.

Cabe destacar que la X es un valor entero entre 0 y 9.

## Representación de los datos en memoria

Posición de memoria	Valor	Descripción del elemento guardado
<b>1869h</b>	<b>04h</b>	# elementos
<b>186Ah</b>	<b>01h</b>	Primer dígito entero
<b>186Bh</b>	<b>02h</b>	Segundo dígito entero
<b>186Ch</b>	<b>40h</b>	Coma
<b>186Eh</b>	<b>03h</b>	Primer dígito parte decimal

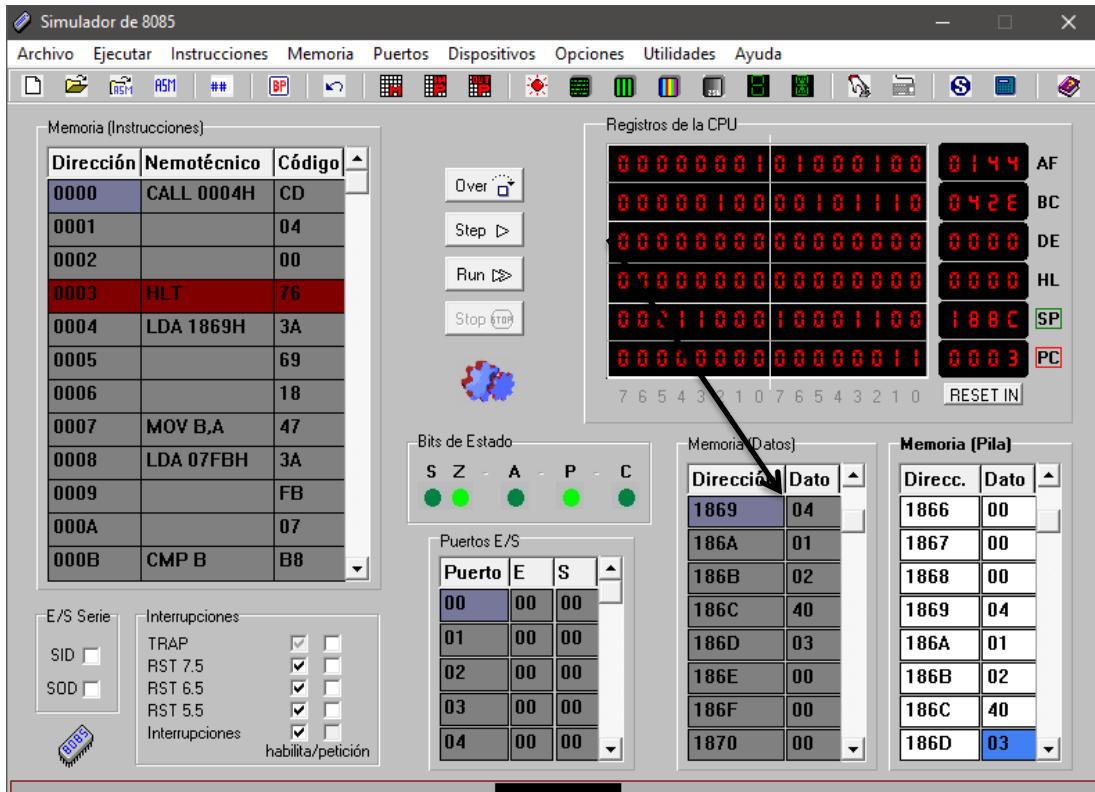
Tabla A3-Buffer 8279 con datos específicos.

Posición de memoria	Valor	Descripción del elemento guardado
<b>188Ch</b>	<b>01h</b>	Primer dígito entero
<b>188Dh</b>	<b>02h</b>	Segundo dígito entero
<b>188Eh</b>	<b>40h</b>	Coma
<b>188Fh</b>	<b>03h</b>	Primer dígito parte decimal
<b>1890h</b>	<b>00h</b>	Segundo dígito parte decimal

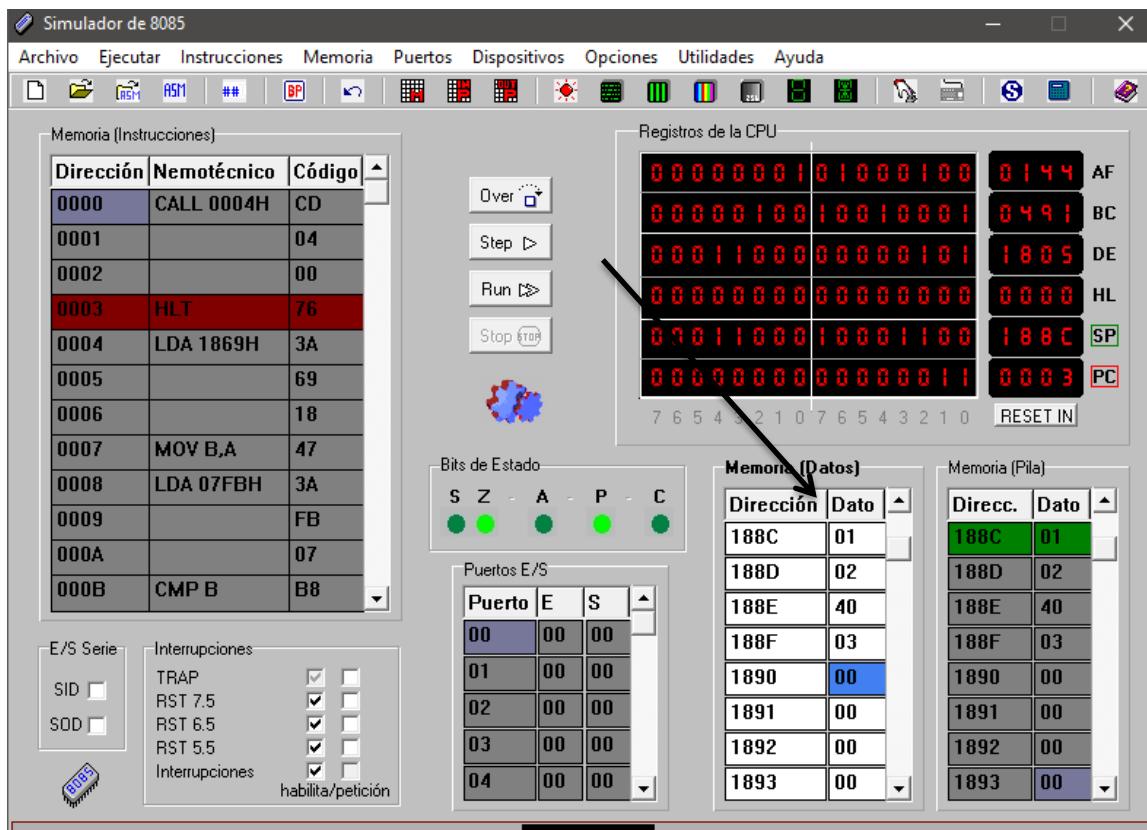
Tabla A4-Buffer de dígitos guardado por el adjust.

Nótese que el número ingresado fue 12.3 y el número ajustado es 12.30

## Ejemplo con el simulador



Ejemplo del simulador. Contenido que guardará el 8279 en las posiciones descritas.



**Ejemplo del simulador. Contenido que guardará el ADJUST en las posiciones descritas.**

A modo de explicación se realizó una prueba en el simulador de la rutina ADJUST por si sola para así tener una mejor comprensión de la misma obteniendo un resultado como el mostrado en las imágenes anteriores.

## SAVE RATE

Esta rutina guarda los datos ingresados los cuales corresponden a cada una de las conversiones que se pueden realizar las cuales son las siguientes:

### RATES NAMES CODES :

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

A cada dato ingresado se le realizará un ajuste con la subrutina ADJUST y luego de esto será guardado en una tabla la cual se representará como se mostró anteriormente con cada uno de los valores correspondientes a un tipo de conversión. Cada uno de estos valores de conversión corresponderá a un buffer que contiene 5 caracteres (4 dígitos y la coma) el cual representa el valor numérico para cada conversión, dichos valores se ingresarán uno a la vez de manera continua y serán guardados cada uno en la tabla, cada valor guardado en la tabla tendrá el siguiente formato:

USD ->COP	1	1800h Parte entera decena 1801h Parte entera unidad 1802h Coma 1803h Parte decimal decena 1804h Parte decimal unidad
USD ->VES	2	1805h Parte entera decena 1806h Parte entera unidad 1807h Coma 1808h Parte decimal decena 1809h Parte decimal unidad
COP ->USD	3	180Ah Parte entera decena 180Bh Parte entera unidad 180Ch Coma 180Dh Parte decimal decena 180Eh Parte decimal unidad
COP ->VES	4	180Fh Parte entera decena 1810h Parte entera unidad 1811h Coma 1812h Parte decimal decena 1813h Parte decimal unidad
VES ->USD	5	1814h Parte entera decena 1815h Parte entera unidad 1816h Coma 1817h Parte decimal decena 1818h Parte decimal unidad
VES -> COP	6	1819h Parte entera decena 181Ah Parte entera unidad 181Bh Coma 181Ch Parte decimal decena 181Dh Parte decimal unidad

Tabla S1-Dígitos y su correspondencia en direcciones de memoria

## Funcionamiento

Para esta subrutina simplemente se toma el dígito ajustado y se guarda en direcciones de memoria contiguas a partir de la 1800h donde cada 5 posiciones representa un dígito como se ilustró en la tabla anterior. Cada dígito que es un buffer con 5 posiciones se va guardando contiguamente en memoria hasta haber guardado los 6 dígitos correspondientes.

En función de mantener la referencia de la posición actual de la tabla en cada iteración donde se guardan en memoria, se usan dos posiciones de memoria específicas para que esta referencia esté ahí las cuales son:

181Fh	18h
1820h	XXh

**Tabla S2-Referencia de la posición donde serán guardados los dígitos en tabla**

Donde las XX representan un número desde el 00h hasta 1Dh.

Como se mencionó al principio de la descripción de esta rutina el número a ser guardado en tabla debe ajustarse por lo que luego de hacer uso de la rutina **ADJUST** con dicho número ingresado, este número se encontrará en el buffer de dígitos guardado por el adjust (**Ver tabla A2**). Sabiendo que el número está en dicha posición se copia desde ahí cada número y se guarda en la tabla que comienza desde la posición **1800h** hasta la posición **181Dh**, donde cada 5 posiciones de memoria(buffer) corresponderá a un tipo de conversión específico(**Ver Tabla S1**). Por lo tanto en cada una de las iteraciones que se realizan sobre esta rutina(en este caso son 6 una por cada tipo de conversión) se guarda dicho buffer y se mantiene en la tabla(posiciones de memoria de **1800h** hasta **181Dh**),dicha tabla será luego referenciada para realizar una conversión específica y es ahí donde reside la importancia de la misma.

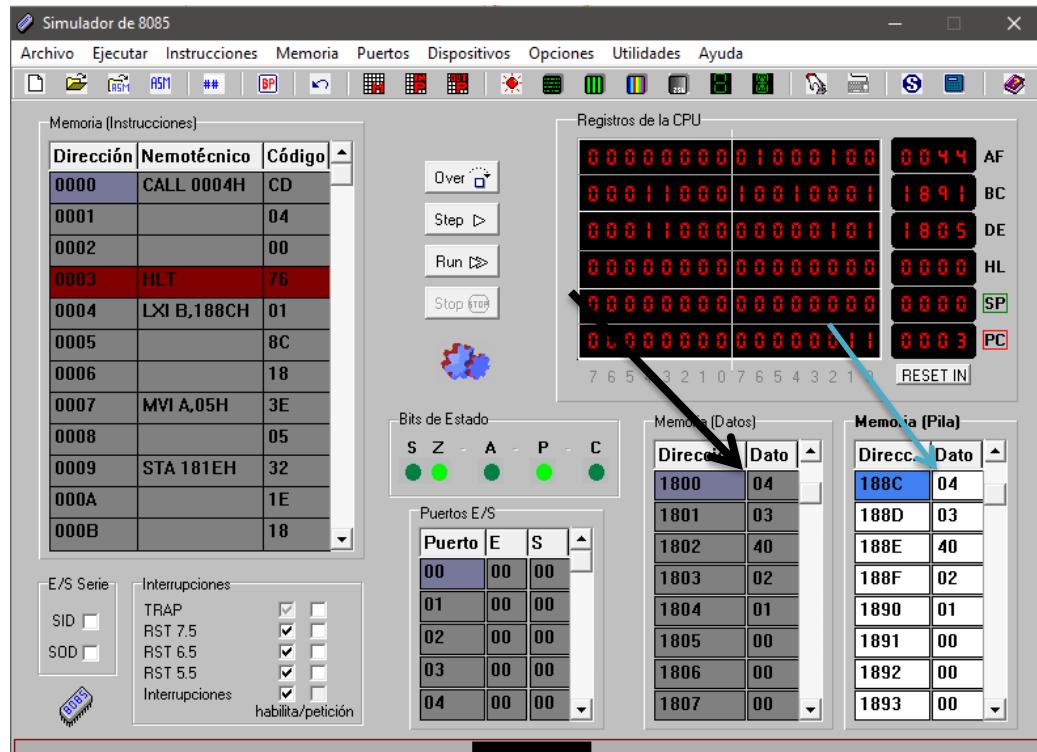
## Ejemplo con el simulador

En este apartado se mostrará el proceso de guardado de un número usando la rutina **SAVE RATE** con valores específicos usados a modo de ejemplo.

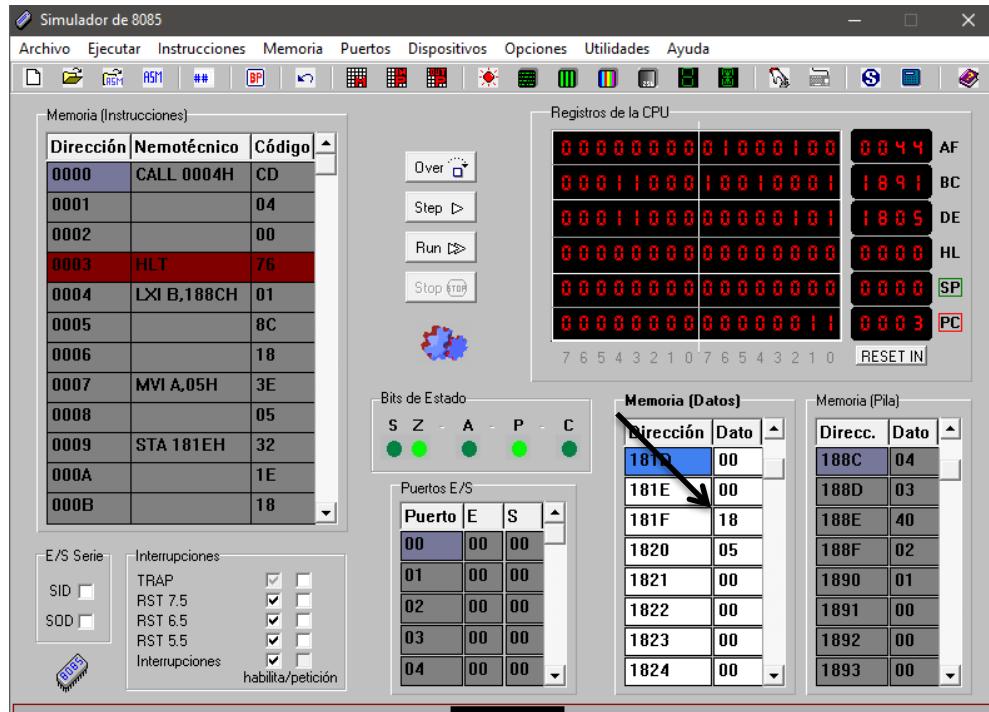
En las posiciones 188Ch hasta la 1890h estará lo guardado por el **ADJUST** que en este caso será lo siguiente:

Posición de memoria	Valor	Descripción del elemento guardado
<b>188Ch</b>	<b>04h</b>	Primer dígito entero
<b>188Dh</b>	<b>03h</b>	Segundo dígito entero
<b>188Eh</b>	<b>40h</b>	Coma
<b>188Fh</b>	<b>02h</b>	Primer dígito parte decimal
<b>1890h</b>	<b>01h</b>	Segundo dígito parte decimal

Sabiendo esto el ejemplo a mostrar se debe guardar lo que está en este buffer a las posiciones de memoria 1800h hasta 1804h que representaría el primer valor de la conversión, aparte de esto la referencia de la posición debe quedar en 1805h para que al momento de volver a llamar a la subrutina empiece a guardar los elementos desde dicha posición que corresponde al segundo número y así se comportará hasta haber guardado los 6 números necesarios en la tabla.



En azul lo guardado por el **ADJUST** en negro lo guardado por el **SAVE\_RATE**.



Posición donde está la referencia donde se guardará el número en tabla

## MULTIPLY

Esta rutina tiene como entrada un par de operandos que tienen el formato especificado por la rutina **ADJUST**. Estos operandos se encuentran en el segmento **MULTIPLY OPERANDS**, previamente definido en el mapa de memoria. La salida de este procedimiento es el resultado de la multiplicación de ambos operandos. Dicha salida es almacenada en memoria, en el rango de direcciones 1873H – 187BH.

1891H	Primer digito entero
1892H	Segundo digito entero
1893H	Coma
1894H	Primer digito parte decimal
1895H	Segundo digito parte decimal

**Posiciones donde estará el primer operando (Multiplicando)**

1896H	Primer digito entero
1897H	Segundo digito entero
1898H	Coma
1899H	Primer digito parte decimal
189AH	Segundo digito parte decimal

**Posiciones donde estará el segundo operando (Multiplicador)**

## Funcionamiento

Se cargan los dos operandos de las posiciones de memoria correspondientes y se realiza el llamado a la rutina que realizará la multiplicación. Posteriormente el resultado se almacena en un buffer temporal, el cual estará ubicado en el rango de direcciones 18B2H – 18ABH. Cabe destacar que dicho resultado está almacenado en orden inverso, es decir que cada dígito se lee desde la posición 18B2H hasta la 18ABH.

## Ejemplo con el simulador

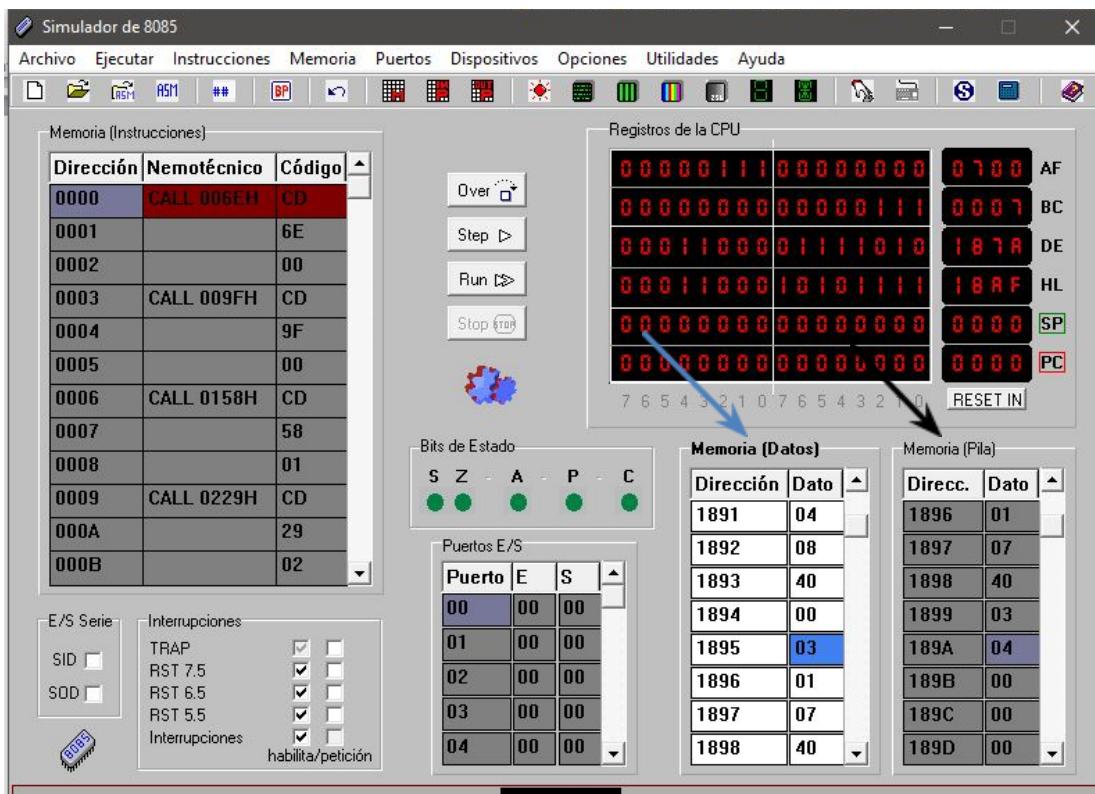
Para este ejemplo se usarán los siguientes operandos:

1891H	04H
1892H	08H
1893H	40H
1894H	00H
1895H	03H

Multiplicando → 48.03

1896H	01H
1897H	07H
1898H	40H
1899H	03H
189AH	04H

Multiplicador → 17.34



Ejemplo en el simulador Multiplicando(Apuntador azul), Multiplicador (Apuntador negro)

->18D9	02H
->18DA	01H
->18DB	02H
->18DC	09H
->18DD	01H
->18DE	09H
->18DF	00H
->18E0	04H
->18E1	04H
->18E2	01H
->18E3	01H
->18E4	02H
->18E5	06H
->18E6	03H
->18E7	03H
->18E8	03H
->18E9	00H
->18EA	08H
->18EB	04H
->18EC	00H

**Representación en memoria de cada uno de los productos calculados anteriormente**

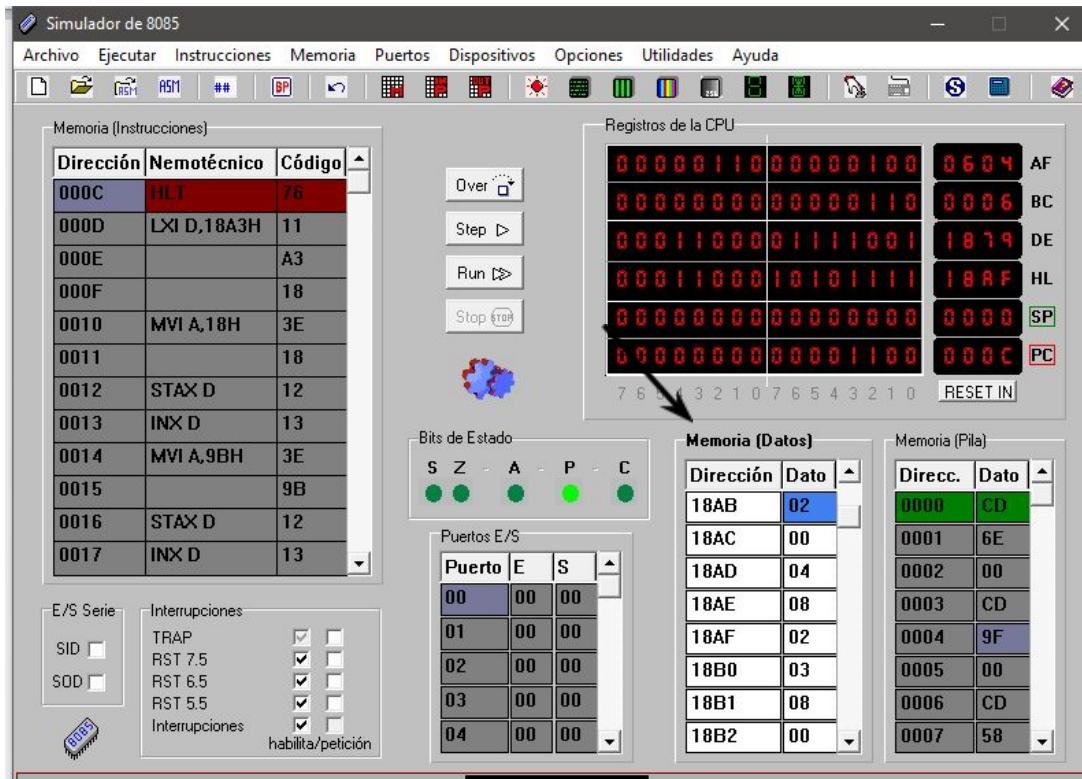
Después de obtener cada uno de los productos generados por el multiplicador (17.34) y el multiplicando (48.03), se obtiene el resultado final mediante la siguiente tabla, donde Ai representa cada uno de los dígitos que conforman dicho resultado y Ci representa el acarreo generado por la operación.

A0 = 18D9
A1 = 18DA + 18DE <- C1
A2 = 18DB + 18DF + 18E3 + C1 <- C2
A3 = 18DC + 18E0 + 18E4 + 18E8 + C2 <- C3
A4 = 18DD + 18E1 + 18E5 + 18E9 + C3 <- C4
A5 = 18E2 + 18E6 + 18EA + C4 <- C5
A6 = 18E7 + 18EB + C5 <- C6
A7 = 18EC + C6

El resultado está compuesto de la siguiente forma, los dígitos A7, A6, A5 y A4 conforman la parte entera y A3, A2, A1 y A0 la parte decimal. Para este caso, sería:

A3 = 08; A2 = 04; A1 = 00; A0 = 02 → **Parte decimal**

A7 = 00; A6 = 08; A5 = 03; A4 = 02 → **Parte entera**



**Resultado de la multiplicación almacenado en memoria (18ABH-18B2H)**

$$17.34 * 48.03 = 832.8402$$

Ya que el formato para mostrar el resultado obtenido anteriormente es una secuencia de caracteres de un tamaño máximo de 7 elementos, el resultado debe truncarse para cumplir este requerimiento. Realizando este procedimiento y agregando la coma se obtiene el resultado final que es almacenado en el rango de direcciones 1873H – 187BH.

Memoria (Datos)	
Dirección	Dato
1873	06
1874	08
1875	03
1876	02
1877	40
1878	08
1879	04
187A	08

En el rango de direcciones 1874H – 187BH se encuentran los dígitos que conforman el resultado final y en la dirección 1873H el tamaño de la secuencia de caracteres, es decir el número de bytes que conforman dicha secuencia. El contenido de esta última dirección (1873H) será de ayuda al momento de imprimir la secuencia de bytes a través del display 8279, ya que indica la cantidad máxima de caracteres a leer de esta secuencia.

## SUBRUTINA PROGRAM

### FUNCION GENERAL

La subrutina program se encarga de generar el comportamiento del monitor; donde, primero se llama a mostrar por el display la palabra 'CODE' de tal manera de pedir al usuario uno de los seis codigos para el exchange, de esta forma se llama a ingresar los datos por el teclado, una vez obtenido el codigo se procede a buscar dicho codigo en la tabla donde se encuentran las tarifas segun el codigo ingresado, ya encontrada la tarifa se guarda esta en la seccion de memoria de operandos. Luego se llama a mostrar la palabra 'BILL' en el display para que el usuario ingrese el monto a convertir y se procede a llamar a ingresar por el teclado dicho monto, como el monto que puede ingresar el usuario no es estandar, se llama a la funcion ADJUST para ajustar el monto ingresado en el formato xx,xx y asi una vez ajustado el monto se guarda dicho monto ajustado en la seccion de memoria donde se alojan los operandos. Terminado esta secuencia de pasos ya se poseen los operandos para la multiplicacion, y asi, se procede a llamar a la funcion MULTIPLY para multiplicar los operandos y obtener asi el resultado. Una vez obtenido el resultado se procede a mostrarlo en el display y asi finalizar la ejecucion de la subrutina.

### SECUENCIA DE PASOS DETALLADA:

#### Ingresar el primer operando

Utilizando el registro A se cargan las letras 'C' , 'O', 'D' y 'E' que respectivamente son '43H', '4FH', '44H' y '45H' (en ASCII) en las posiciones donde leera el display para asi mostrarlo y que en este caso empieza en la posicion '187CH' y termina en la posicion '1880H', teniendo en cuenta ademas que se carga el caracter '\n' que es '00H' en ASCII para identificar el final de la cadena. Una vez cargada la palabra 'CODE' en las posiciones del display se procede a llamar a la subrutina 'PRINT\_ASCII' para mostrar en el display los caracteres cargados.

```
;PRINT_TO_DISPLAY pedimos introducir el codigo
MVI A, 43H; C-> se mueve al registro A la data '43H', que representa el caracter 'C' en ASCII
STA 187CH; Carga letra C en la posicion del display 187CH
MVI A, 4FH; O-> se mueve al registro A la data '4FH', que representa el caracter 'O' en ASCII
STA 187DH; Carga letra O en la posicion del display 187DH
MVI A, 44H; D-> se mueve al registro A la data '44H', que representa el caracter 'D' en ASCII
STA 187EH; Carga letra D en la posicion del display 187EH
MVI A, 45H; E-> se mueve al registro A la data '45H', que representa el caracter 'E' en ASCII
STA 187FH; Carga letra E en la posicion del display 187FH
MVI A, 00H; \n -> representa el fin de linea
STA 1880H; Carga caracter '\n' en la posicion del display 1880H
CALL PRINT_ASCII
;termina salida por pantalla
```

Fig.p1: Mostrar en el display la cadena 'CODE'

El paso anterior se hace con el fin de decirle al usuario que ingrese el codigo por teclado; asi, luego de este paso se llama a la subrutina INPUT para que el usuario ingrese los datos en el teclado. Como se sabe que dicha subrutina INPUT guarda lo ingresado por teclado desde la posicion de memoria '1869H' con una longitud de 9 bytes, pero como se ingresa en este caso un codigo, es decir, un digito del 1al 6 (1 byte), se sabe que el codigo esta guardado en la posicion '186AH', de esta forma con ayuda del registro A usando el comando LDA se carga dicho codigo en el registro A (LDA 186AH) y se guarda dicho codigo en el registro B usando (MOV B,A).

```
CALL INPUT; ingresar el codigo por teclado
LDA 186aH; Carga codigo ingresado por el usuario por teclado
MOV B,A; Mover el codigo al registro B
```

Fig.p2: Pedir ingresar Codigo por teclado y cargarlo en Reg.B

Se sabe que la tabla de RATES según el código ingresado está desde la posición de memoria '1800H' hasta la '1820H' y que cada RATE ocupa 5 bytes con los códigos secuencialmente; es decir, el código 1 está desde la posición 1800H hasta la 1804H, el código 2 desde la posición 1805H hasta 1809H y así sucesivamente para los demás códigos. Sabiendo esto se carga la posición 1800H en el registro HL (LXI H, 1800H), viéndose algo como H → 18H y L → 00H. Se inicializa un contador en el registro D (MVI D, 00H) y se ingresa a la subrutina (proceso) 'INCREMENT\_D', la cual se encarga de decifrar qué código fue el ingresado y esto se hace usando el contador inicializado en el registro D, donde, primero se aumenta en '1' el contador (INR D), se mueve dicho contenido del registro D al registro A (MOV A,D) para poder comparar el registro D con el registro B (CMP B), si los registros D y B son iguales entonces salta a la subrutina OUT\_LOOP (JZ OUT\_LOOP) y si no es así entonces, se procede a aumentar la dirección guardada en el registro HL 5 bytes más allá, para ubicar la dirección de memoria donde se encuentra el siguiente RATE para el siguiente código. Este procedimiento se hace con la subrutina (proceso) 'INCREMENT\_C', donde primero cargamos en C → 00H (MVI C,00H) y en A → 06H (MVI A,06H), ingresamos al proceso y se incrementa el registro C en una unidad (INR C), se compara con registro A (CMP C) y si es igual es porque ya está ubicado en la posición de memoria del siguiente RATE y código entonces se salta a la subrutina (proceso) 'INCREMENT\_D' (JZ INCREMENT\_D) y si no entonces se incrementa la posición de memoria guardada en el registro HL anteriormente (posición de memoria de la tabla) en una unidad (INR L) y se procede a volver a ejecutar el mismo proceso INCREMENT\_C.

```
LXI H, 1800H; cargar dirección de la tabla de RATES en el registro HL
MVI D, 00H; Inicializar el contador
INCREMENT_D: INR D; D+=1 aumentar en una unidad el contador
    MOV A,D; Mover el contenido del registro D al registro A
    CMP B; if B == D , comparar el contenido del registro A con el contenido del registro B
    JZ OUT_LOOP; Si la comparación anterior es verdadera (Los contenidos son iguales), saltar a OUT_LOOP

    MVI C, 00H; Inicializar en 00H el registro C
    MVI A, 06H; Inicializar en 06H el registro A
INCREMENT_C: INR C; C+=1 , aumentar el contenido del registro C en una unidad
    CMP C; Comparar el contenido del registro C con el contenido del registro A
    JZ INCREMENT_D; Si la comparación anterior es verdadera (Los contenidos son iguales), saltar a INCREMENT_D
    INR L; incrementar posición en memoria
    JMP INCREMENT_C; Saltar sin condición a INCREMENT_C
```

Fig.p3: Ubicar dirección en la tabla de RATES según el código ingresado

Una vez finalizado el proceso de INCREMENT\_D e INCREMENT\_C, e ingrese a la subrutina OUT\_LOOP, se tendrá en el registro HL la dirección de memoria donde empieza el RATE del código ingresado por el usuario. De esta forma basta con cargar cada byte al registro A y cargarlo a la dirección de memoria de los operandos; es decir, se mueve el contenido de lo que está en memoria (Dirección que apunta el registro HL) al registro A (MOV A,M) y se carga a la primera dirección de los operandos (STA 1891H), aumentamos una unidad del registro HL (INR L) y se vuelve a mover el contenido del registro HL (MOV A,M) al registro A y se carga a la dirección que sigue de operandos (STA 1892H), así sucesivamente tres veces más, ya que el RATE ocupa 5 bytes de longitud (xx,xx).

```
OUT_LOOP: MOV A,M; primer byte del elemento en la tabla
    STA 1891H; cargamos primer byte del primer operando
    INR L; incrementar posición en memoria
    MOV A,M; segundo byte del elemento en la tabla
    STA 1892H; cargamos segundo byte del primer operando

    INR L; incrementar posición en memoria
    MOV A,M; tercer byte del elemento en la tabla
    STA 1893H; cargamos tercer byte del primer operando

    INR L; incrementar posición en memoria
    MOV A,M; cuarto byte del elemento en la tabla
    STA 1894H; cargamos cuarto byte del primer operando

    INR L; incrementar posición en memoria
    MOV A,M; quinto byte del elemento en la tabla
    STA 1895H; cargamos quinto byte del primer operando
```

Fig.P4: Cargar RATE de la tabla hacia la sección de memoria de Operandos

## Ingresar el segundo operando

En este paso se cargan las letras utilizando el registro A ‘B’ ‘I’ ‘L’ y ‘L’ que respectivamente son ‘42H’, ‘45H’, ‘4CH’ y ‘4CH’ (en ASCII) en las posiciones donde leera el display para asi mostrarlo y que en este caso empieza en la posicion ‘187CH’ y termina en la posicion ‘1880H’, teniendo en cuenta ademas que se carga el carácter ‘\n’ que es ‘00H’ en ASCII para identificar el final de la cadena. Una vez cargada la palabra ‘BILL’ en las posiciones del display se procede a llamar a la subrutina ‘PRINT\_ASCII’ para mostrar en el display los caracteres cargados.

```
;PRINT_TO_DISPLAY pedimos introducir el monto a convertir
MVI A, 42H; B -> se mueve al registro A la data '42H', que representa el caracter 'B' en ASCII
STA 187CH; Carga letra B en la posicion del display 187CH
MVI A, 45H; I -> se mueve al registro A la data '45H', que representa el caracter 'I' en ASCII
STA 187DH; Carga letra I en la posicion del display 187DH
MVI A, 4CH; L -> se mueve al registro A la data '4CH', que representa el caracter 'L' en ASCII
STA 187EH; Carga letra L en la posicion del display 187EH
MVI A, 4CH; L -> se mueve al registro A la data '4CH', que representa el caracter 'L' en ASCII
STA 187FH; Carga letra L en la posicion del display 187FH
MVI A, 00H; \n -> representa el fin de linea
STA 1880H; Carga caracter '\n' en la posicion del display 1880H
|CALL PRINT_ASCII
;termina salida por pantalla
```

Fig.p5: Mostrar en el display la cadena ‘BILL’

El paso anterior se hace con el fin de decirle al usuario que ingrese el monto a convertir por teclado; asi, luego de este paso se llama a la subrutina INPUT para que el usuario ingrese los datos en el teclado. Como el monto que puede ingresar el usuario no es estandar entonces se llama a la subrutina ADJUST (CALL ADJUST) para ajustar dicho monto en un formato de la forma xx.xx, de esta manera el monto ajustado por la subrutina ADJUST es guardado en la posicion de memoria 188CH hasta la 1890H, asi basta solo con cargar dicho monto en las posiciones de memoria de los operandos. Usando el registro HL se facilita esta tarea, se carga la posicion de memoria 188CH en el registro HL (LXI H, 188CH), luego se mueve el contenido de la direccion que apunta el registro HL al registro A (MOV A,M), es decir, se mueve el primer caracter del monto ajustado al registro A, y asi se procede a cargar lo que esta en el registro A (caracter del monto ajustado) a la primera posicion de memoria de el siguiente operando a guardar (STA 1896H), se incrementa en una unidad el registro HL (INR L) y se vuelve a ejecutar el mismo procedimiento planteado anteriormente 4 veces. Una vez finalizada esta carga de datos se tendra ya almacenados los operandos en su seccion de memoria especifica para ser leida por la subrutina MULTIPLY, la cual es llamada una vez finalizada la carga de los operandos.

```
CALL INPUT; leemos segundo operando (monto a convertir)
CALL ADJUST; ajustamos segundo operando (monto a convertir)

;ponemos en posicion el segundo operando (monto a convertir)
LXI H, 188CH; carga los Reg.H y L con la data 188CH
MOV A,M; primer byte del elemento ajustado
STA 1896H; cargamos primer byte del segundo operando

INR L; incrementar posicion de memoria
MOV A,M; segundo byte del elemento ajustado
STA 1897H; cargamos segundo byte del segundo operando

INR L; incrementar posicion de memoria
MOV A,M; tercer byte del elemento ajustado
STA 1898H; cargamos tercer byte del segundo operando

INR L; incrementar posicion de memoria
MOV A,M; cuarto byte del elemento ajustado
STA 1899H; cargamos cuarto byte del segundo operando

INR L; incrementar posicion de memoria
MOV A,M; quinto byte del elemento ajustado
STA 189AH; cargamos quinto byte del segundo operando

CALL MULTIPLY ;multiplicamos|
```

Fig.p6: Pedir ingresar Por teclado el monto, ajustar el monto y cargar el monto ajustado en la sección de memoria de operandos

## Imprimir el resultado

La función MULTIPLY es la encargada en multiplicar los operandos anteriormente ingresados, en función a decisiones del usuario, el resultado de la multiplicación es alojada en los espacios de memoria en RAM desde 1873H hasta la 187BH. Se conoce que en la posición 1873H está el número de caracteres del resultado obtenido, así el primer paso es cargar el contenido de esta dirección de memoria en el registro A(LDA 1873H), se guarda en el registro D (MOV D,A) y se pasa o carga dicho número de caracteres a las direcciones del display (STA 187CH). El segundo paso es cargar los caracteres siguientes del resultado en los espacios de memoria del display, para esto se carga la dirección 1874H en el registro HL (LXI H, 1874H) que es donde empieza el resultado obtenido, inicializamos un contador en el registro B (MVI B, 01H) y para convención se aumenta el contenido del registro D en una unidad (INR D).

```
;PRINT_TO_DISPLAY mostramos resultado
LDA 1873H; mandamos al registro la cantidad de digitos
MOV D,A; Guardar cantidad de digitos en el registro D
STA 187CH; ya tenemos en el buffer del display la cantidad de digitos.
MVI B, 01H; inicializamos el registro B
LXI H, 1874H; carga los Reg.H y L con la data 1874H
INR D; incrementar en una unidad registro D
```

Fig.p7: Preparar datos para la carga del resultado al display

Luego se entra en la subrutina (proceso) llamada LOOP\_PROGRAM\_PRINT para realizar el proceso de carga de los caracteres del resultado hacia los espacios de memoria del display. Primero se mueve el contenido del registro D al registro A (MOV D,A) para tener el número de caracteres en el registro A y así compararlo con el registro B (CMP B); si el contenido de los registros A y B son iguales entonces quiere decir que ya se finalizó la carga de los caracteres del resultado a las secciones del display y así se puede saltar a la subrutina (proceso) PRINT\_RESULT para llamar en dicha rutina a PRINT\_NUMBER y así mostrar el resultado en el display. En caso contrario que los contenidos de los registros A y B no sean iguales, se procede a cargar un dígito de los espacios de memoria de MULTIPLY y pasarlo a los espacios de memoria de display, para esto primero se carga el contenido en memoria, es decir, el contenido en la dirección guardada en el registro HL (1874H) y pasarlo al registro A (MOV A,M), guardamos este contenido en el registro C (MOV C,A) y se incrementa la dirección en el registro HL (INR L), además se guarda esta dirección que apunta a la siguiente dirección del siguiente carácter del resultado en la pila (PUSH H), de esta manera se procede a cargar en el registro HL la dirección del display (LXI H, 187CH), se mueve al registro A la base de la posición de inicio de las direcciones del display (MVI A, 7CH) y se suma lo que está en el registro A con el contenido del registro B (ADD B) para ubicar la dirección de memoria en la que se va a cargar el carácter del resultado, se carga dicha ubicación en el registro HL (MOV L,A) y se procede a mover el carácter del resultado el cual se guardó en el registro C a la sección del display (MOV M,C). Finalizado este procedimiento se extrae de la pila el registro HL guardado anteriormente (POP H), se incrementa el contador contenido en el registro B (INR B) y se salta a la misma subrutina (JMP LOOP\_PROGRAM\_PRINT) y se repite el mismo procedimiento hasta tener cargados todos los caracteres del resultado a las secciones de lectura del display. De esta manera finaliza el funcionamiento del monitor y la subrutina PROGRAM.

```

LOOP_PROGRAM_PRINT: MOV A,D; Mover contenido del registro D al registro A
    CMP B; Comparar contenido del registro A con el contenido del registro B
    JZ PRINT_RESULT; termino de iterar salimos, Si la comparacion anterior es verdadera entonces salta a PRINT_RESULT
    MOV A,M; cargamos un digito
    MOV C,A; guardamos el digito
    INR L; incrementar en una unidad el registro L (HL)
    PUSH H; guardamos el registro par HL
    LXI H, 187CH; carga los Reg.H y L con la data 187CH
    MVI A,7CH; Mover data 7CH al registro A
    ADD B; Sumar contenido del registro A con el registro B
    MOV L,A; Mover contenido del registro A al registro L (HL)
    MOV M,C; Mover contenido del registro C a la memoria M (direccion que apunta el registro HL)
    POP H; Sacar de la pila el registro H ingresado anteriormente
    INR B; Incrementar en una unidad el contenido del registro B
    JMP LOOP_PROGRAM_PRINT; regresamos en el loop

PRINT_RESULT: CALL PRINT_NUMBER; imprimimos el resultado
    ;terminamos de imprimir el resultado
RET_PROGRAM: RET ; Finaliza

```

**Fig.p8: Cargar caracteres del resultado hacia las direcciones de lectura del display y mostrar dicho resultado en display**

## Simulación de TronOS

### Instrucciones Iniciales:

Una vez instalado el simulador proporcionado, el siguiente paso es abrirlo y cargar el código **/tronOs/MONITOR.asm**, tal y como se muestra a continuación:

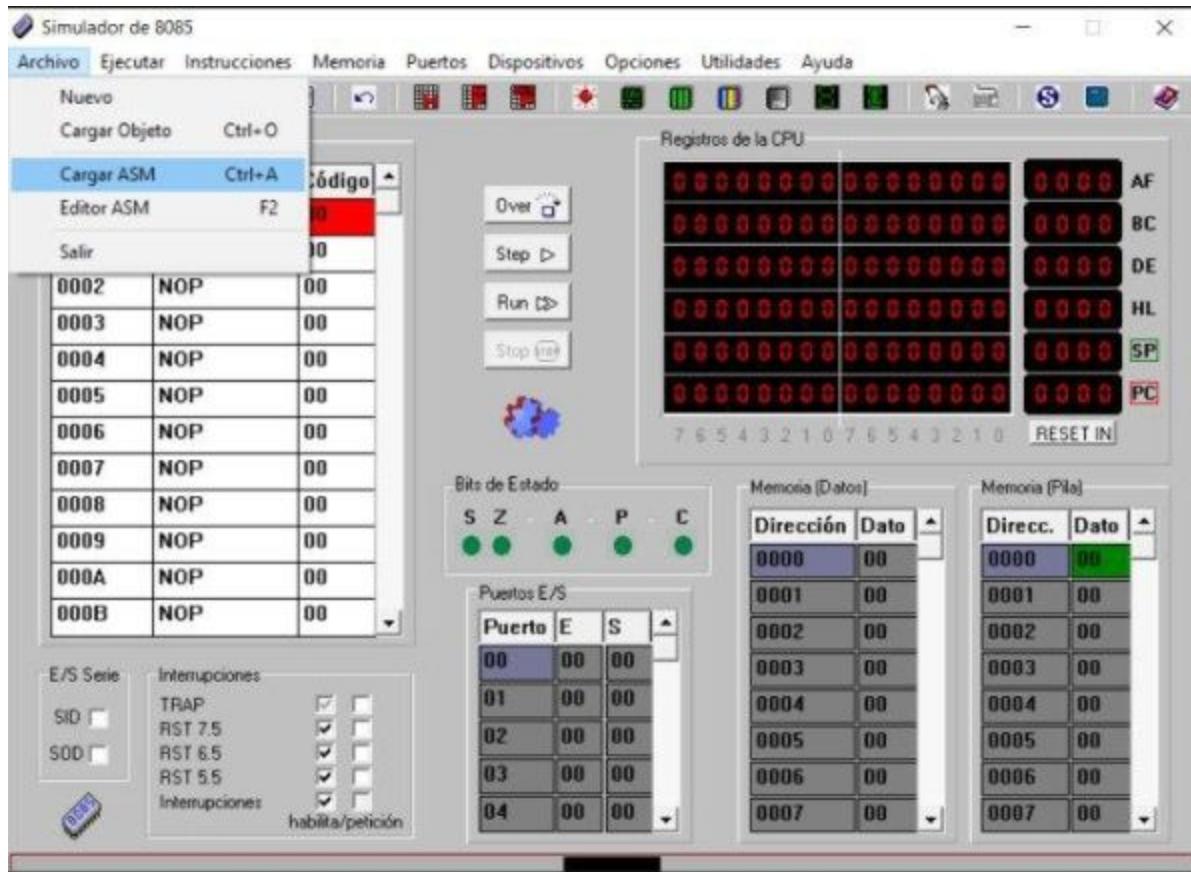


Figura 1.

Esto nos abrirá el editor de texto del simulador donde luego de presionar el botón *simular* el código de **TronOS** estará cargado en el simulador.

TronOs una vez cargado en el simulador se ve de la siguiente forma:

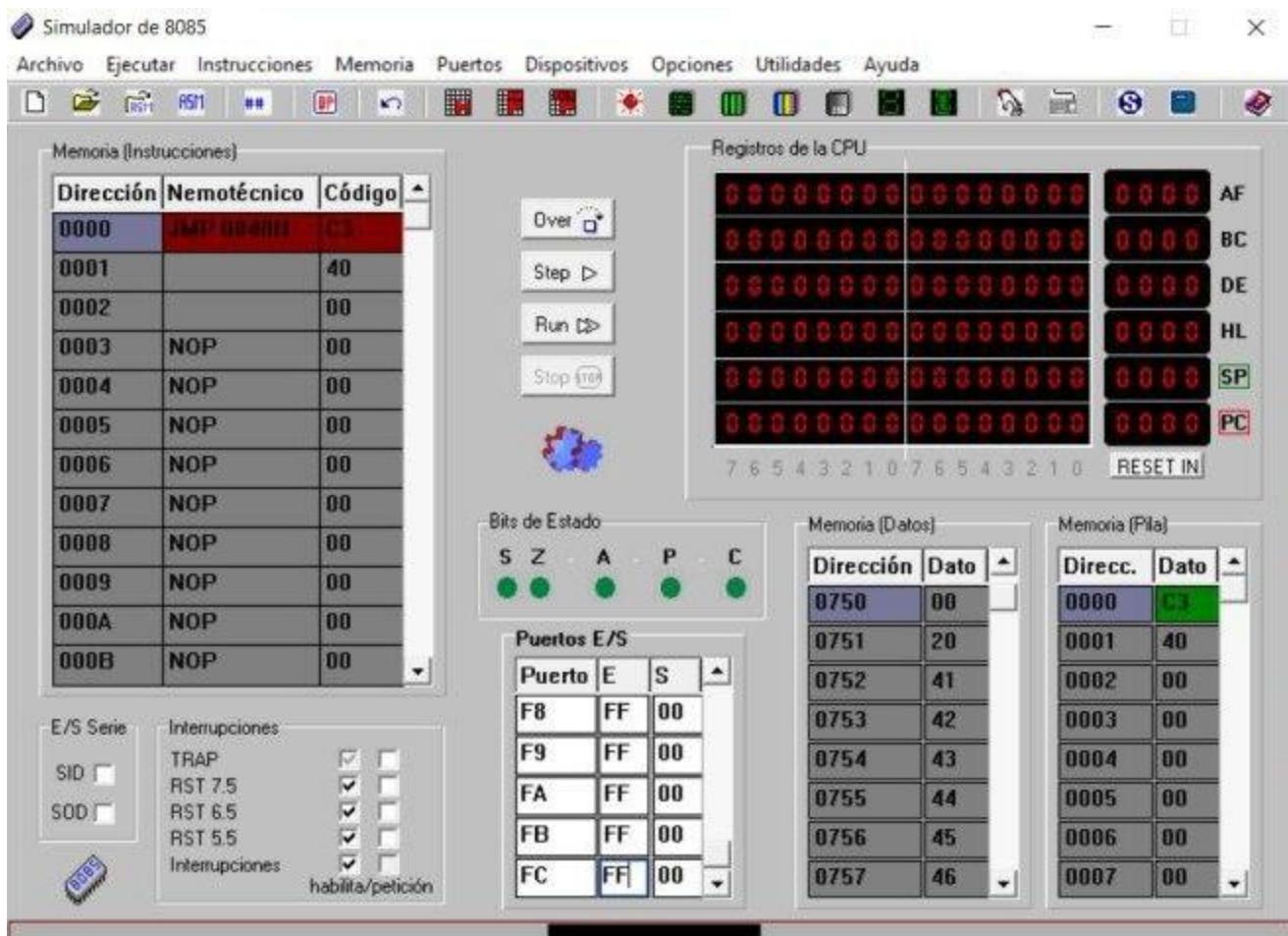


Figura 2.

Las instrucciones en nemotécnico y las direcciones de la misma dentro de la **ROM** pueden encontrarse en el panel de la izquierda.

Con el fin de simular la entrada del teclado son utilizados los puertos de entrada desde el **F8H** hasta el **FFH** donde el valor hexadecimal **FFH** indica que no ha sido ingresado nada “vacío”, por esta razón antes de empezar la simulación debe escribirse en los puertos de entrada desde el **F8H** hasta el **FFH** el valor **FFH**, como se muestra en la *Figura 2*. Luego de realizar estas indicaciones iniciales podemos empezar la simulación.

### Inicialización (INIT)

Tal y como se explicó en la sección de procesos el primer paso que ejecuta **TronOS** es la inicialización de las interrupciones, pila y componentes.

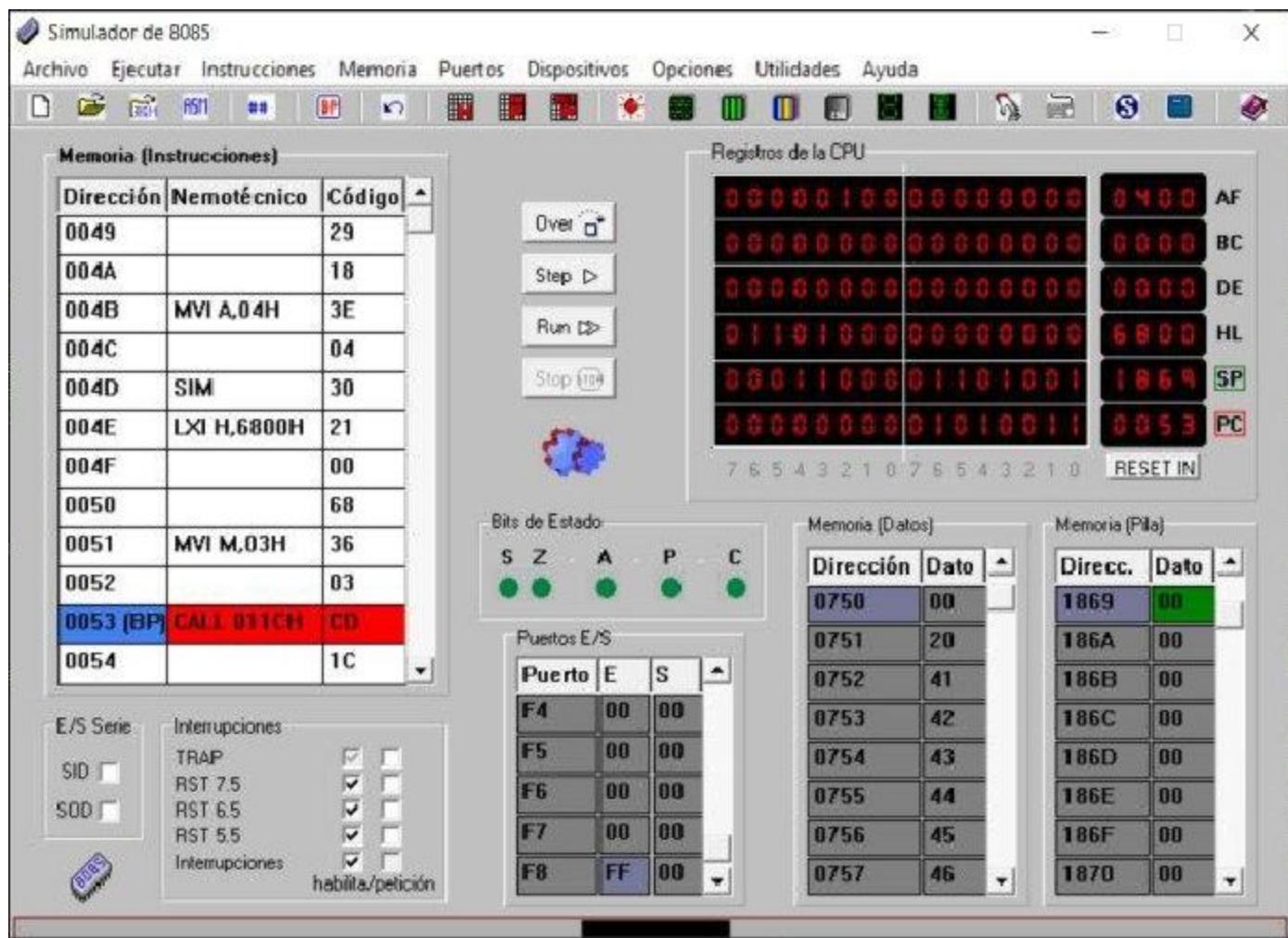


Figura 3.

La inicialización la podemos ver en la Figura 3. Donde el *stack pointer*, interrupciones y componentes han sido inicializados.

#### Lectura y Guardado de la Tabla R\_AND\_S\_T:

El siguiente paso es la lectura y guardado de la tabla, como se explicó anteriormente (ver sección procesos) se leen 6 tasas de cambio donde el proceso para cada una de las 6 es el mismo.

Primero escribimos en el buffer de salida de la rutina de impresión (ver sección memoria) el mensaje “CODE” con el fin de indicarle al usuario que debe ingresar un código de tasa de cambio (ver sección convenciones).

En la siguiente figura vemos como se ha escrito en el buffer de impresión los códigos necesarios para formar la palabra “CODE” en pantalla.

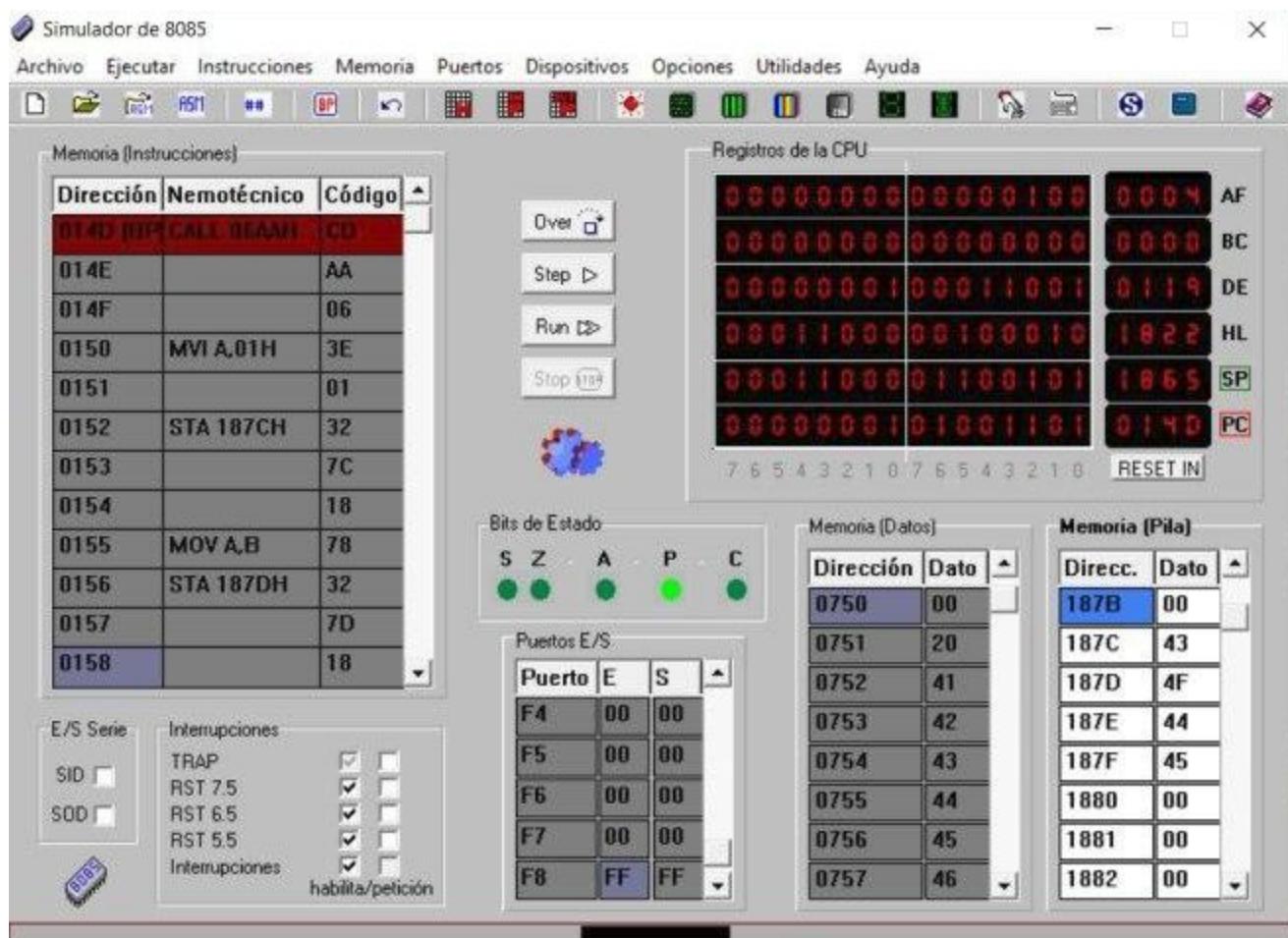


Figura 4

Luego mostramos al usuario que el código que se quiere leer el primer código indicado en el *registro B* esto se hace imprimiendo por pantalla el contenido del registro, como se muestra a continuación (escribiendo en el buffer de salida de impresión y llamando a la rutina *PRINT\_NUMBER* (ver secciones procesos y memoria)).

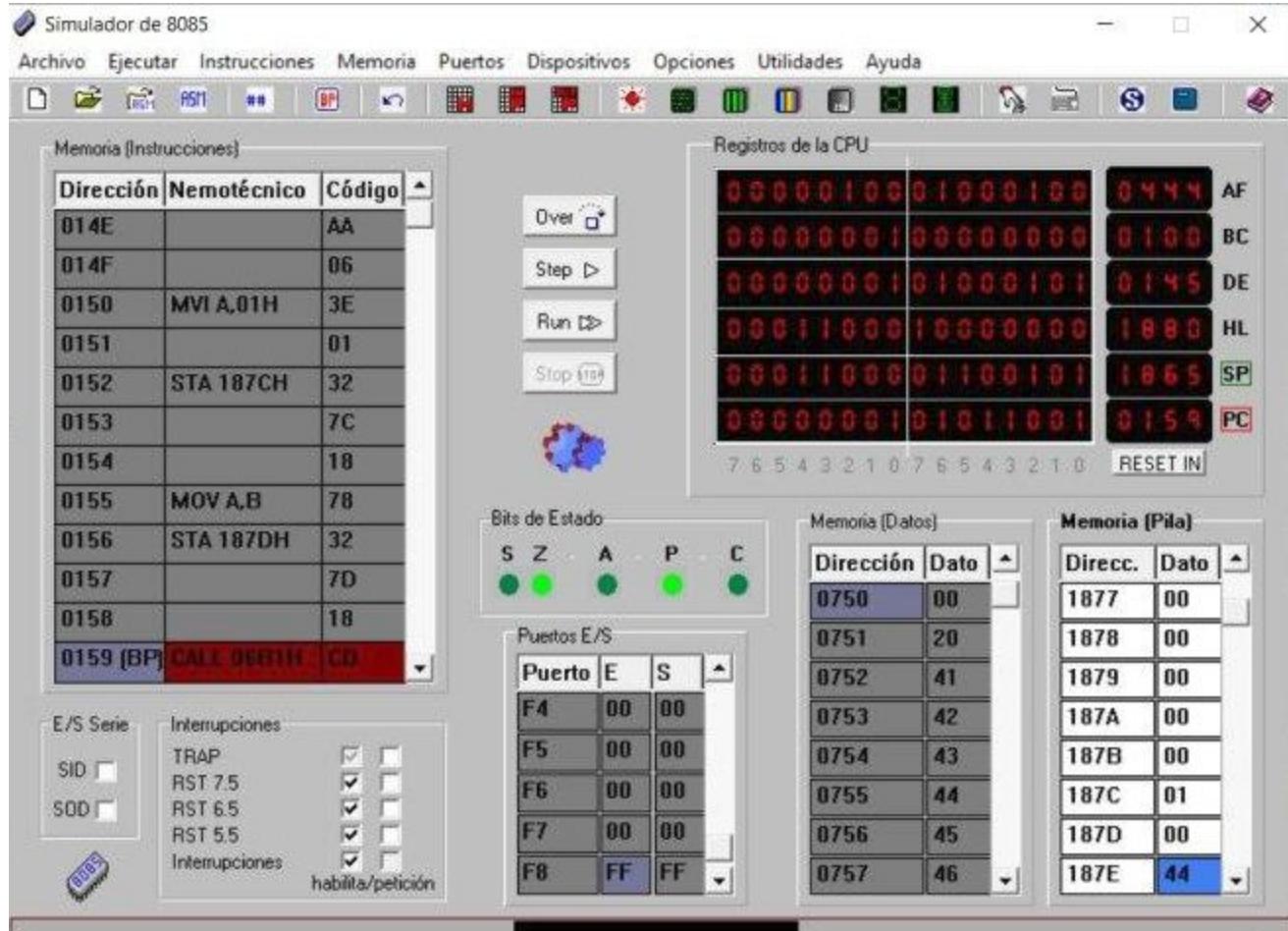


Figura 5.

Una vez se le ha pedido al usuario que ingrese el código el sistema monitor estará en espera de la tasa de cambio indicada, en este caso la primera tasa de cambio como se indicó. Para esto **TronOS** entra en un ciclo infinito a la espera de la tasa de cambio como se muestra en la siguiente figura.

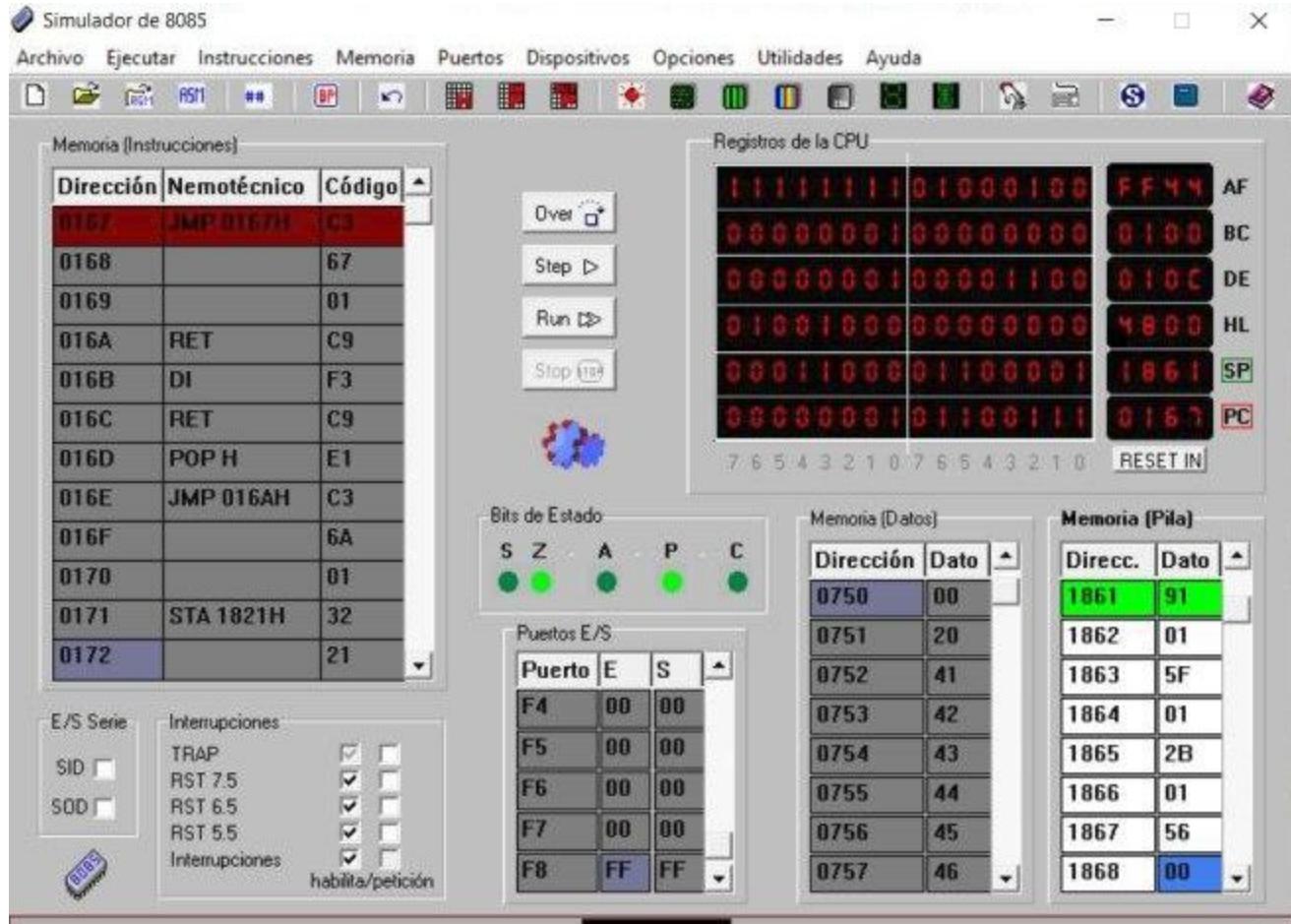


Figura 6.

Para ingresar la tasa de cambio correspondiente debemos escribirla en el formato aceptado por **TronOS** (ver sección *convenciones*) en los puertos de entrada desde el *F8H* hasta el *FFH*, en este caso ingresamos el número *11.00* y activaremos la interrupción 7.5 con el fin de indicarle al sistema monitor que el número fue escrito.

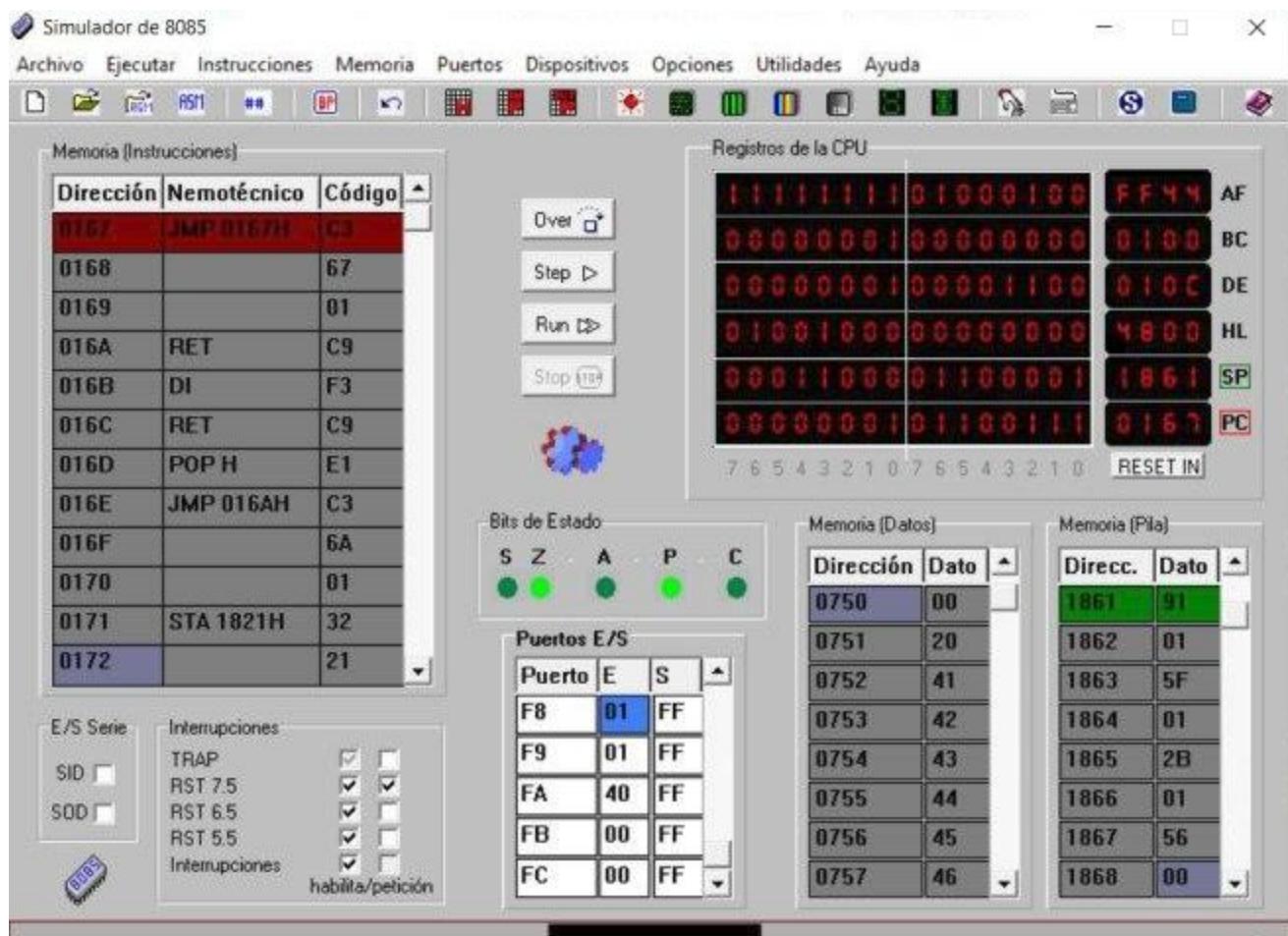


Figura 7.

Una vez el número ha sido ingresado se llama a la rutina *ADJUST* (ver sección procesos) para que ajuste el monto y lo deje listo para la multiplicación y a la rutina *SAVE\_RATE* (ver sección procesos) para guardar la tasa de cambio en la tabla.

El ciclo de llenado sigue ahora requiriendo la tasa de cambio con código 2, para esto se imprime por pantalla la palabra “CODE” y el contenido del registro *B*. En la imagen siguiente vemos cómo se imprime por pantalla el contenido del registro *B* el cual contiene el número 2.

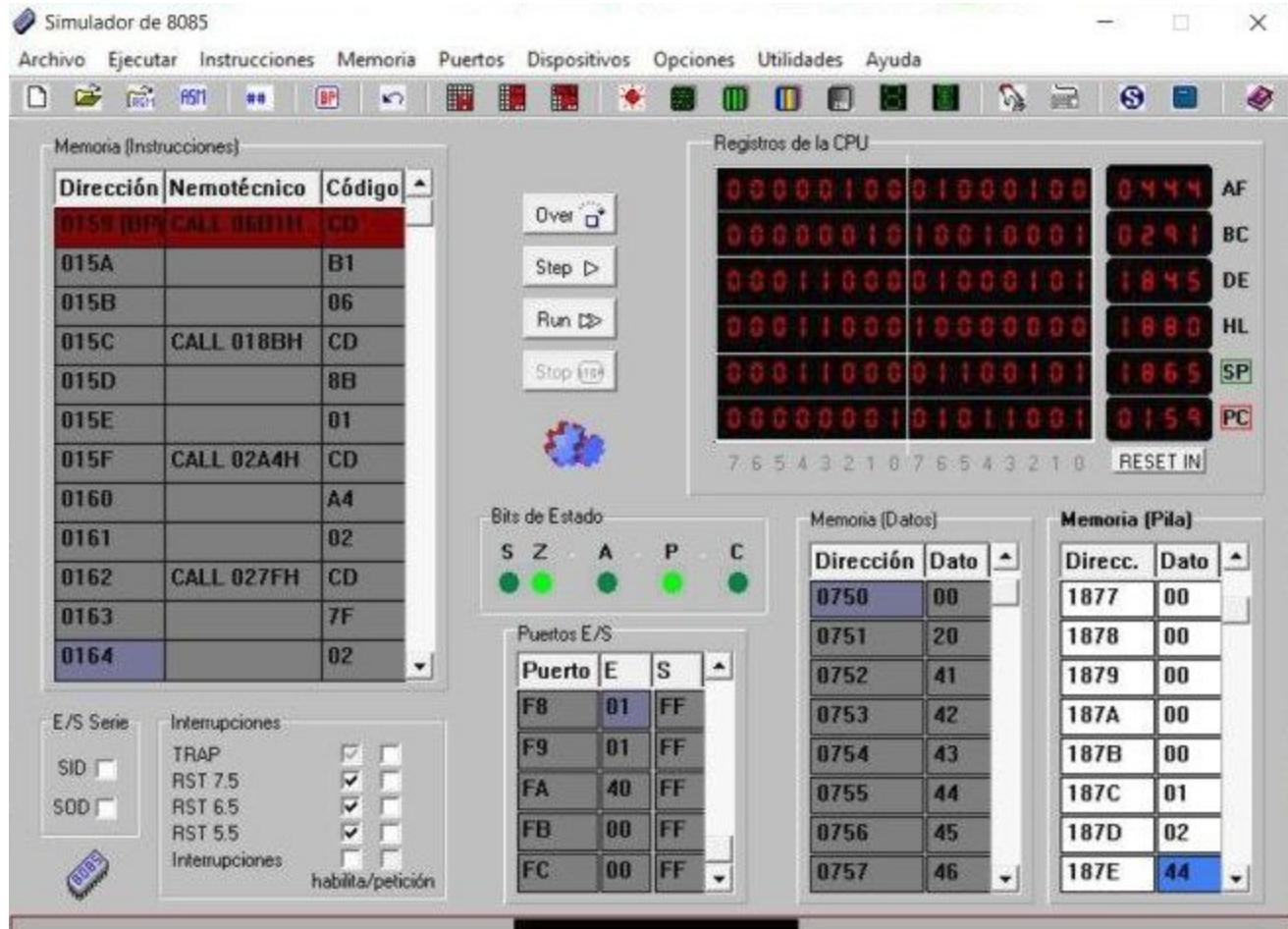


Figura 8.

Nuevamente **TronOS** se ha detenido a la espera de la tasa de cambio 2, es por eso que ingresamos la tasa de cambio (ingresando en los puertos de entrada en el formato **TronOS** como se indica) y activando la interrupción 7.5. En este caso el monto de la tasa de cambio ingresada fue el 2.

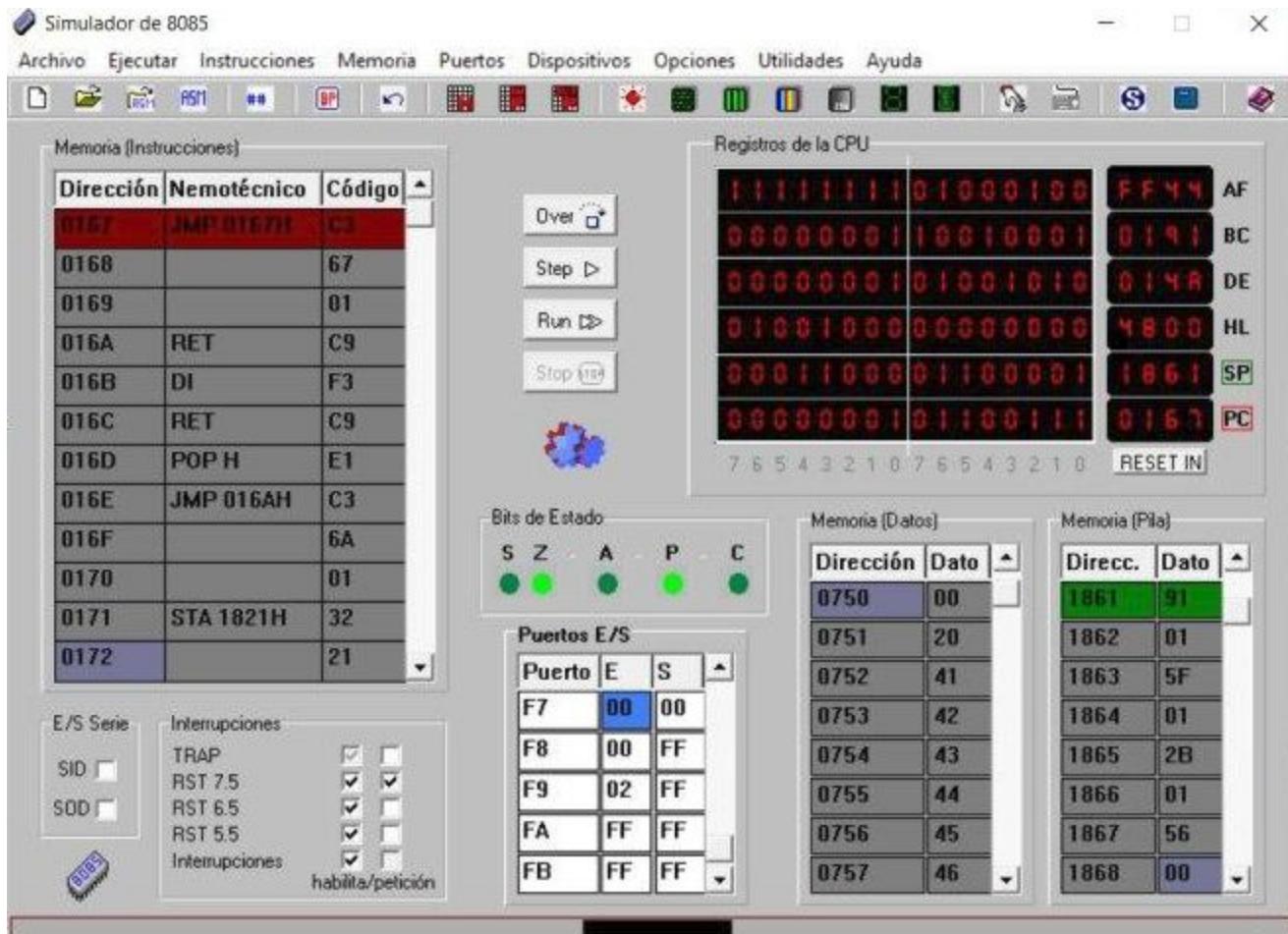


Figura 9.

De la misma forma se realiza el procedimiento para las 4 tazas restantes en donde fueron ingresados los montos 20.3, 20.3, 10.5 y 99.

En la siguiente figura vemos una captura del ingreso del último valor de tasa de cambio (99).

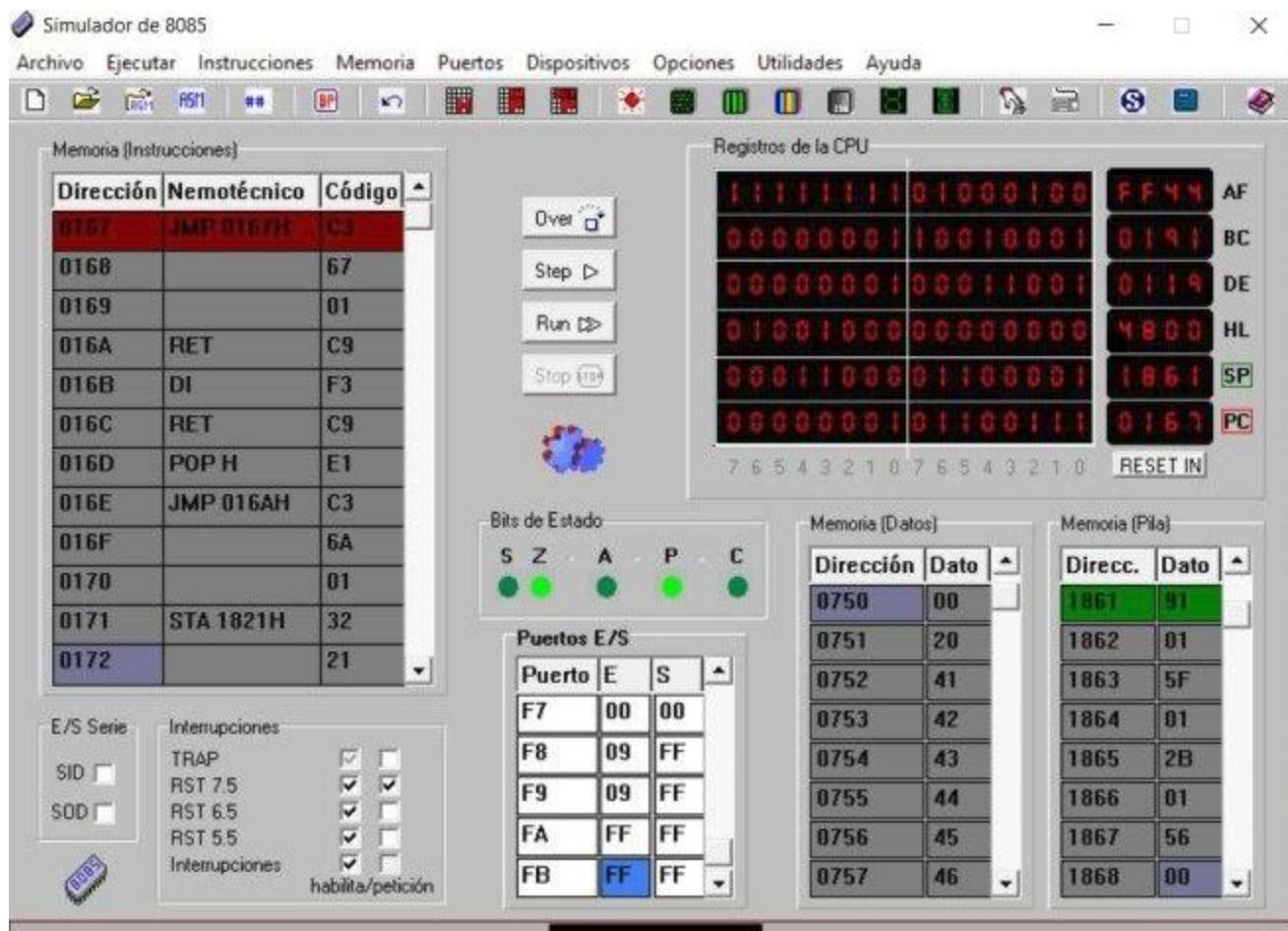
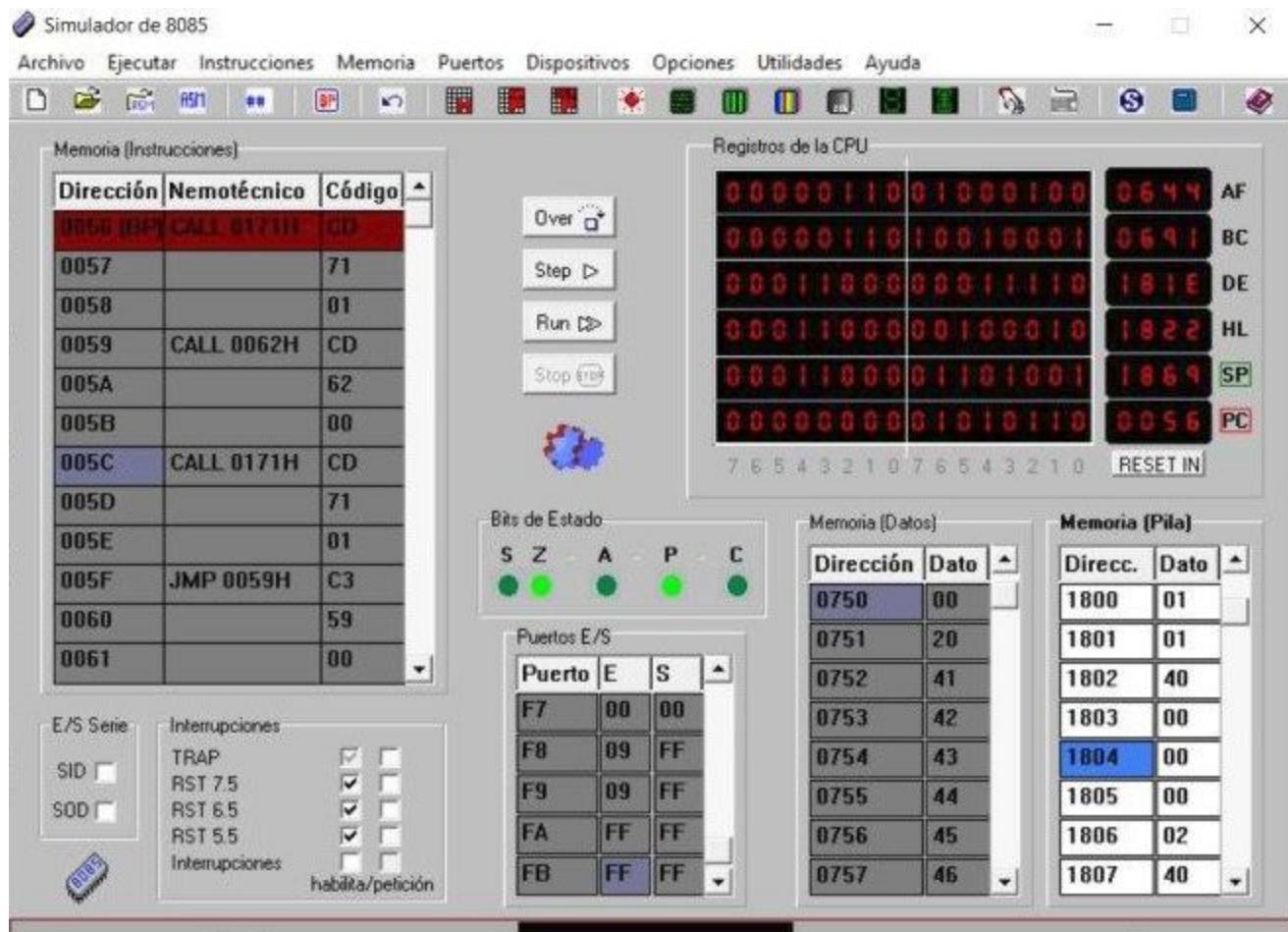


Figura 10.

En este punto la tabla ha sido llenada en su totalidad por lo que si vamos a la posición de memoria que le corresponde a la tabla de tasas de cambio vemos cómo están los 6 valores ingresados ya ajustados y en sus posiciones correspondientes.

Primer valor:



Segundo valor:

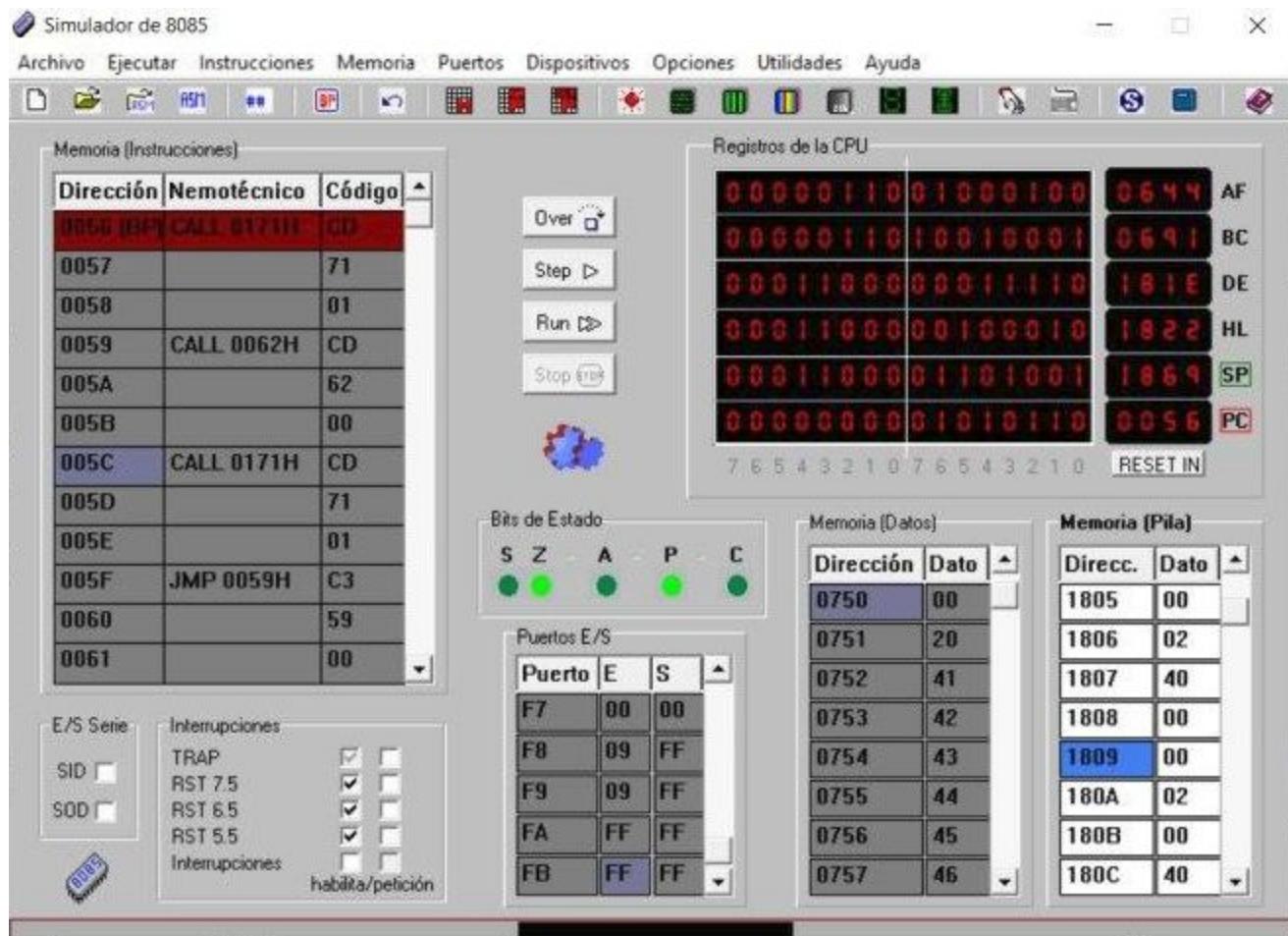


Figura 12.

Tercer valor:

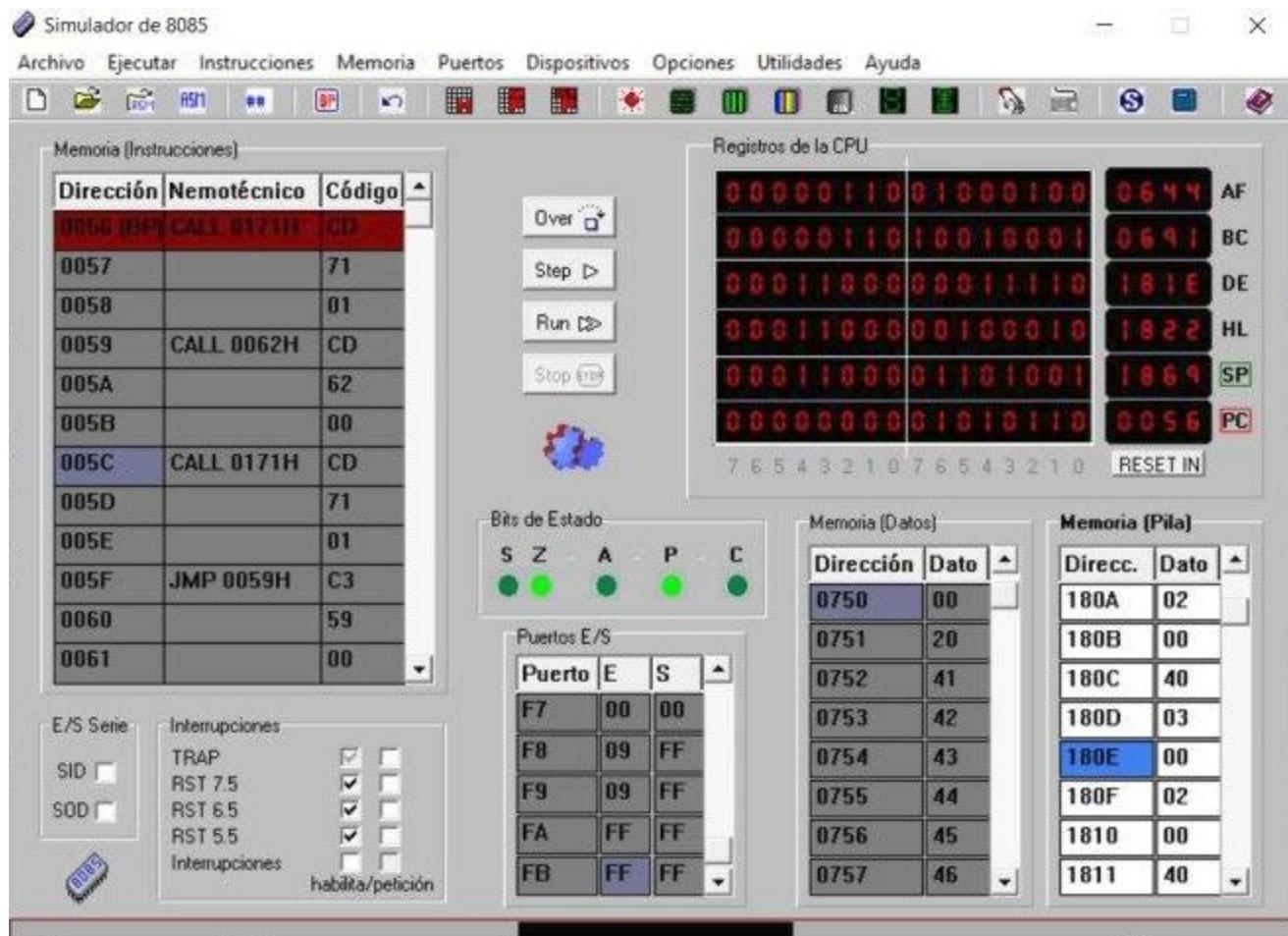


Figura 13.

Cuarto valor:

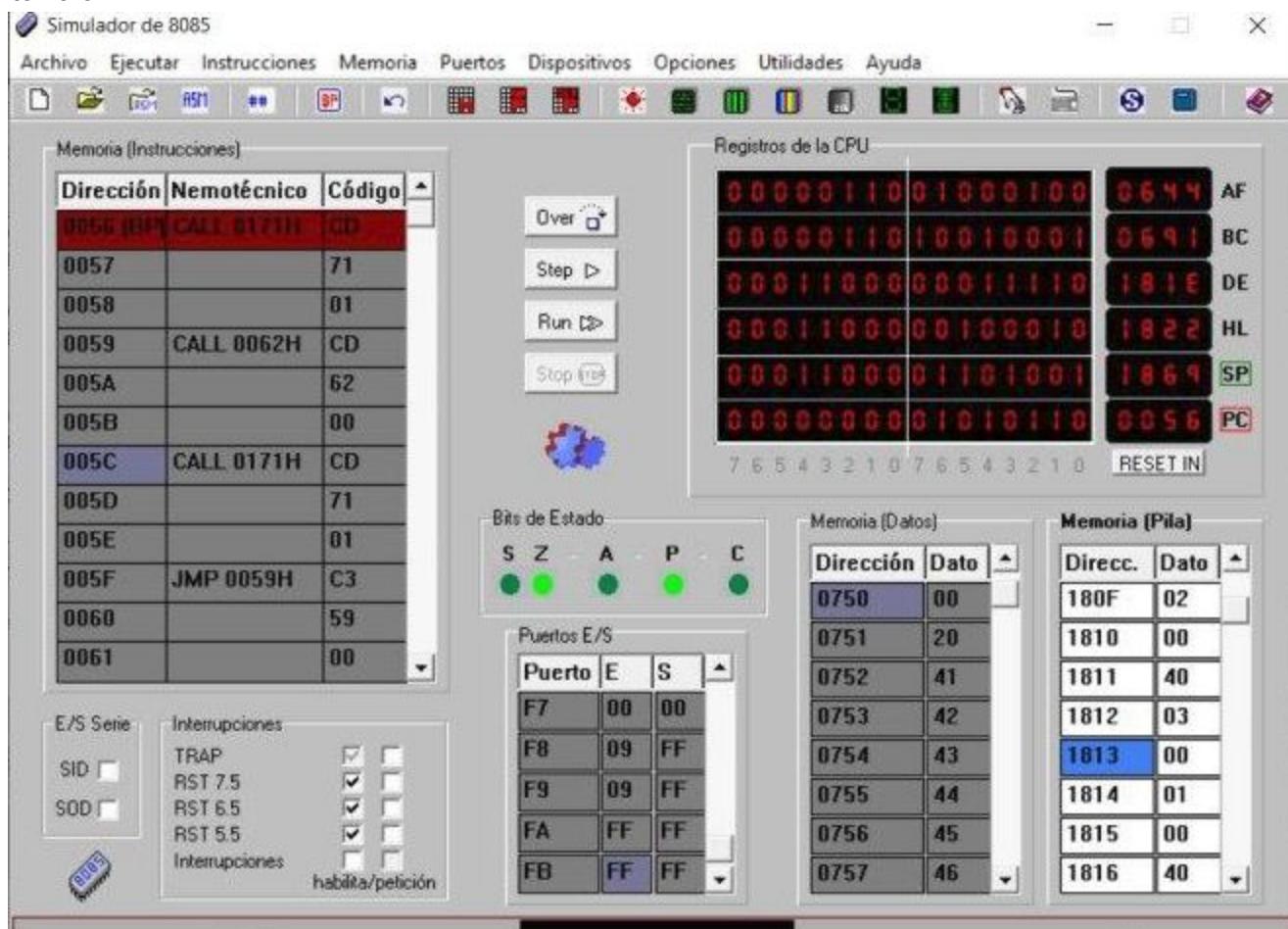
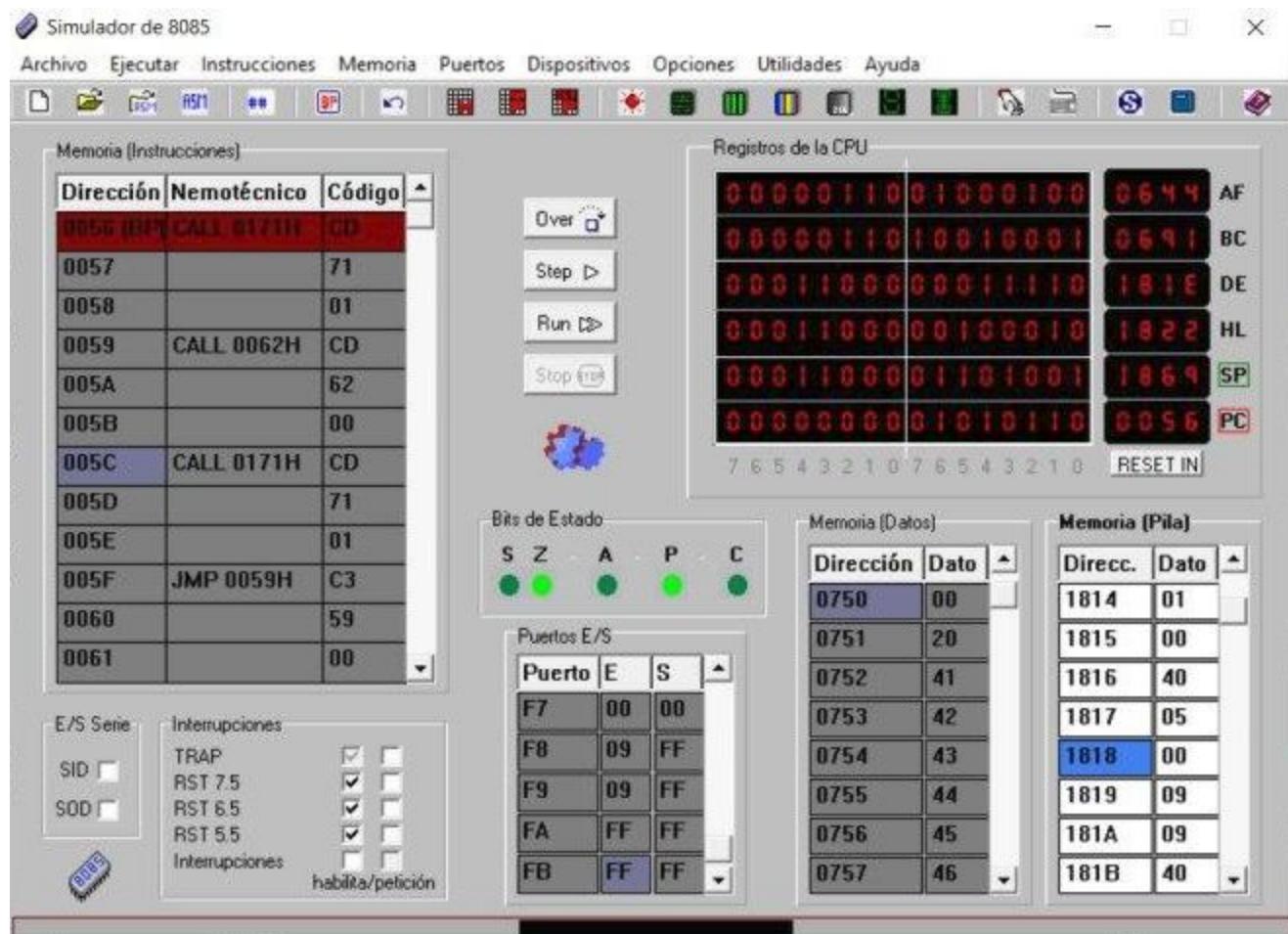
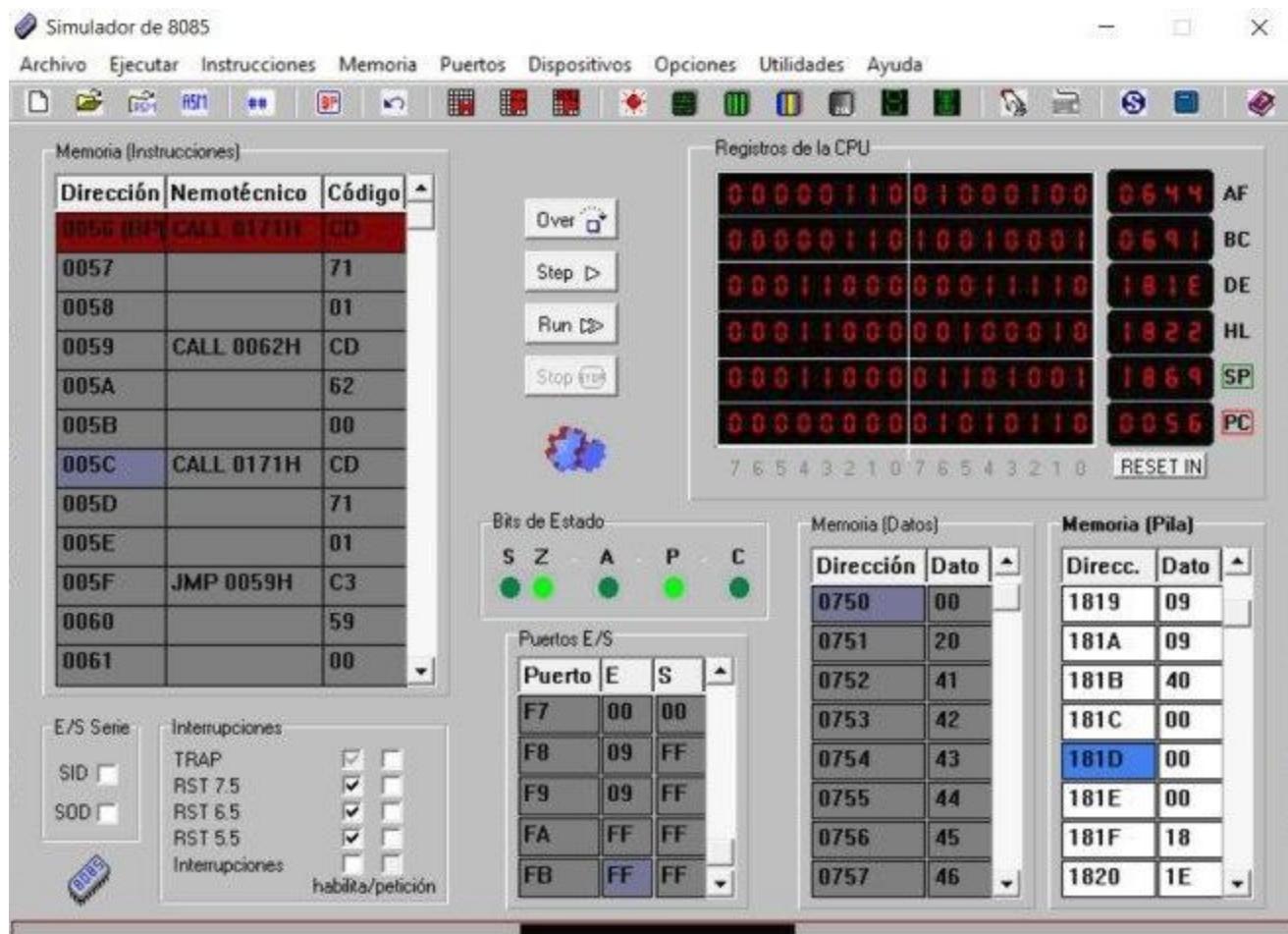


Figura 14.

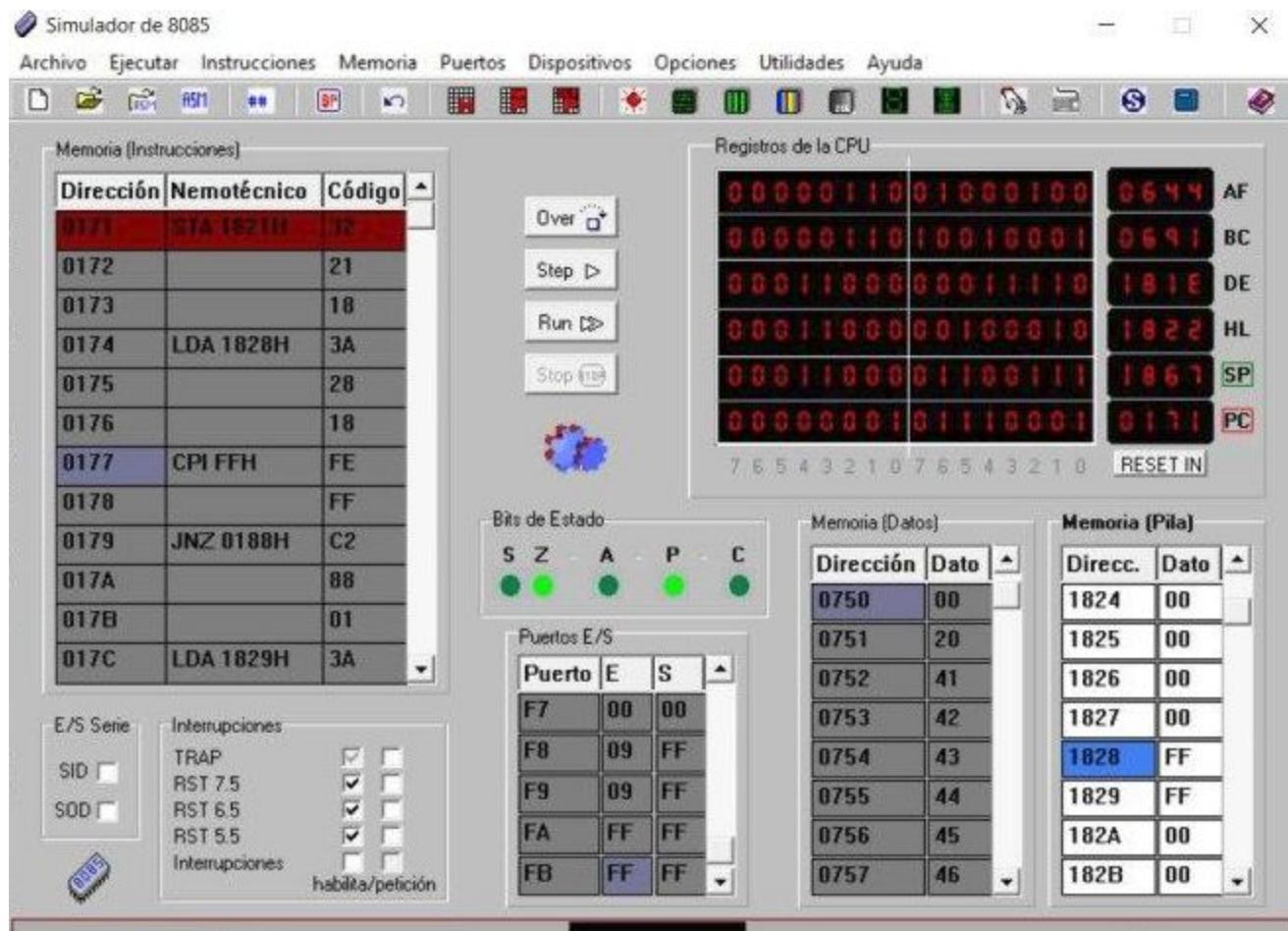
Quinto valor:



Sexto valor:



Para finalizar la lectura y guardado de la tabla debemos ver si no ha ocurrido un desborde de pila, esto lo hacemos con la rutina utilitaria *STACK\_O\_C* (ver sección procesos). Vemos como en las dos primeras posiciones de la zona destinada a la pila se encuentran los valores *FFH* indicando que no ha ocurrido un desborde de pila (ver sección procesos).



## Programa principal PROGRAM:

La siguiente figura muestra una captura del sistema monitor justo antes de entrar en la ejecución del programa principal.

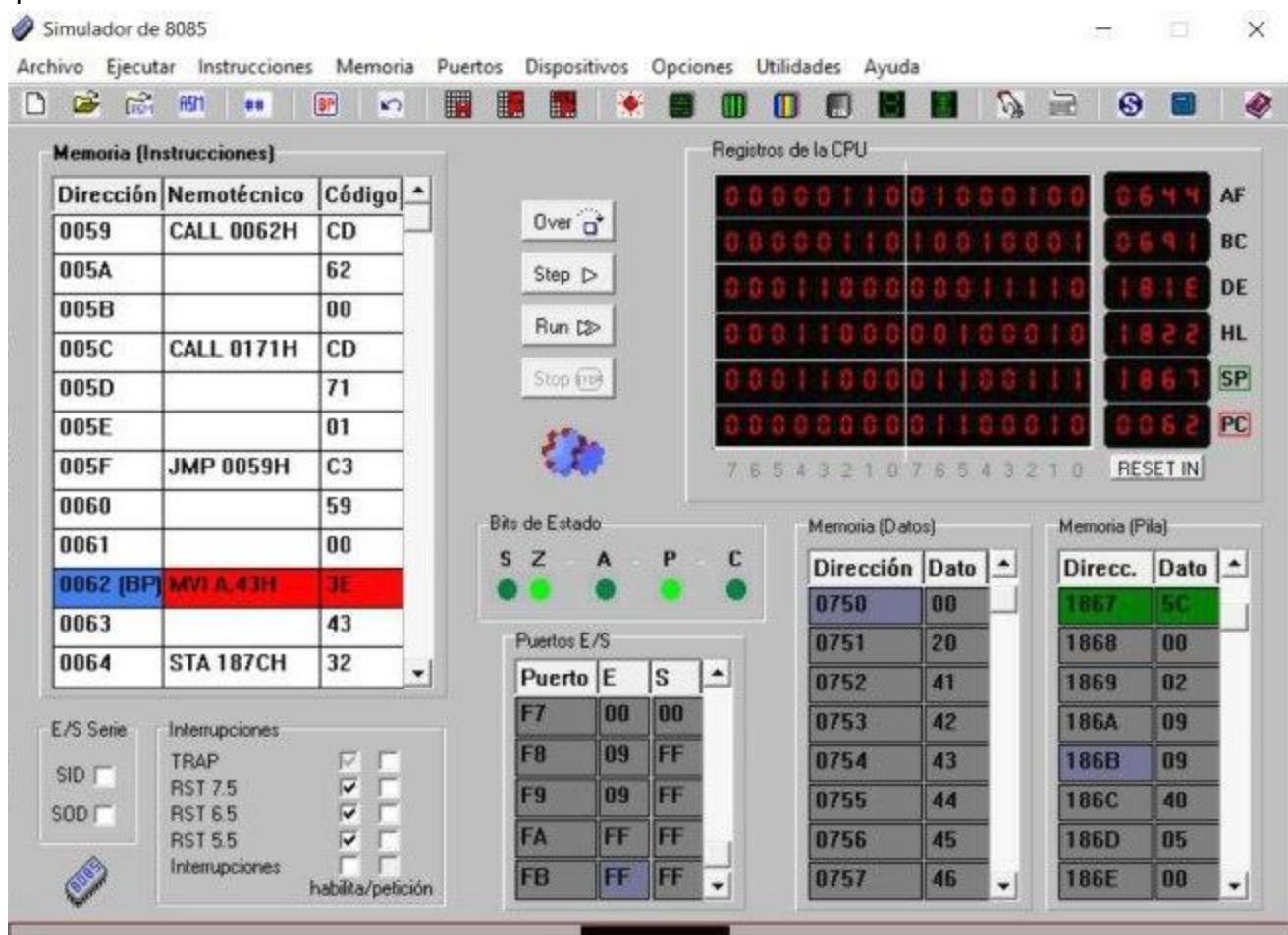


Figura 18.

Como vimos en la sección de procesos la primera acción que ejecuta la rutina *PROGRAM* es indicarle al usuario que ingrese el código de conversión esto se hace mostrando por pantalla la palabra “CODE”.

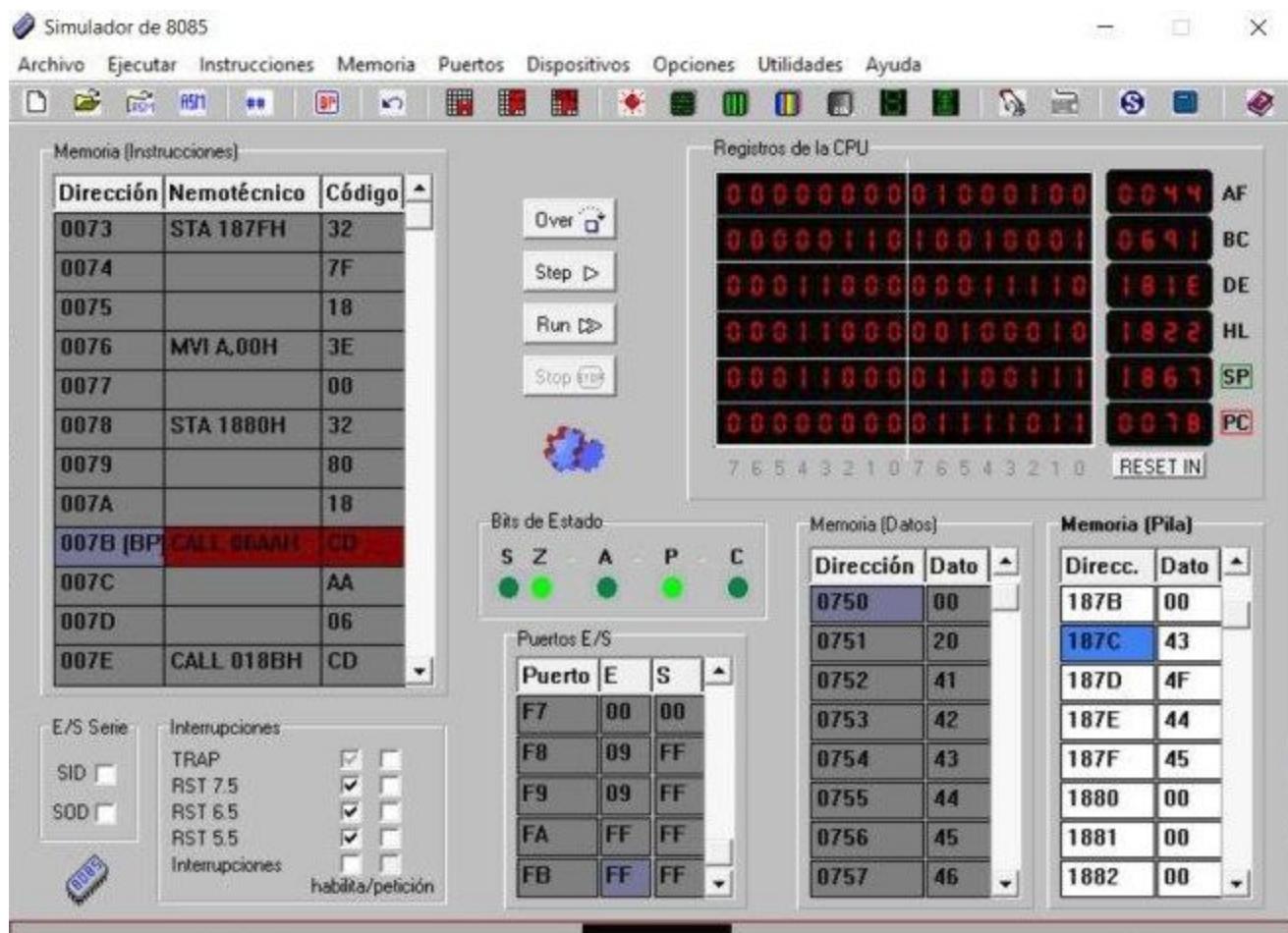


Figura 19.

Recordemos que ingresar un código de cambio (ver sección convenciones) nos da mucha información por ejemplo, si es ingresado el código número 1 este *indica que se llevará de dólares a pesos colombianos USD->COP*, así que solo restaría ingresar un monto en dólares y **TronOS** lo llevará a pesos colombianos. En esta simulación queremos utilizar el código número 4. En la siguiente captura vemos como el sistema está esperando para que sea ingresado el código por lo que ingresamos el numero 4.

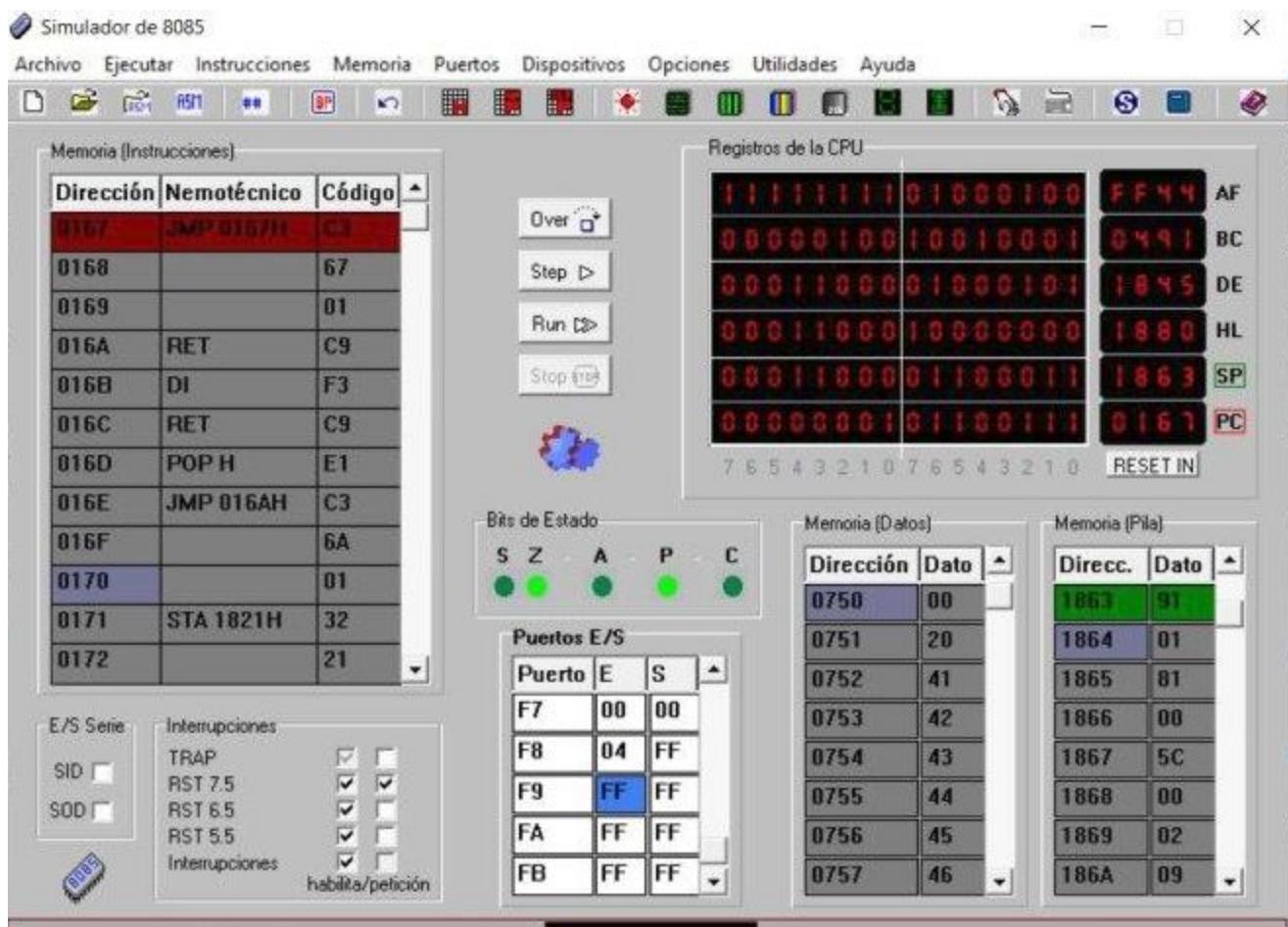


Figura 20.

De esta manera podemos ver como en el buffer de entrada del teclado (ver sección memoria) se encuentra el código que acabamos de ingresar.

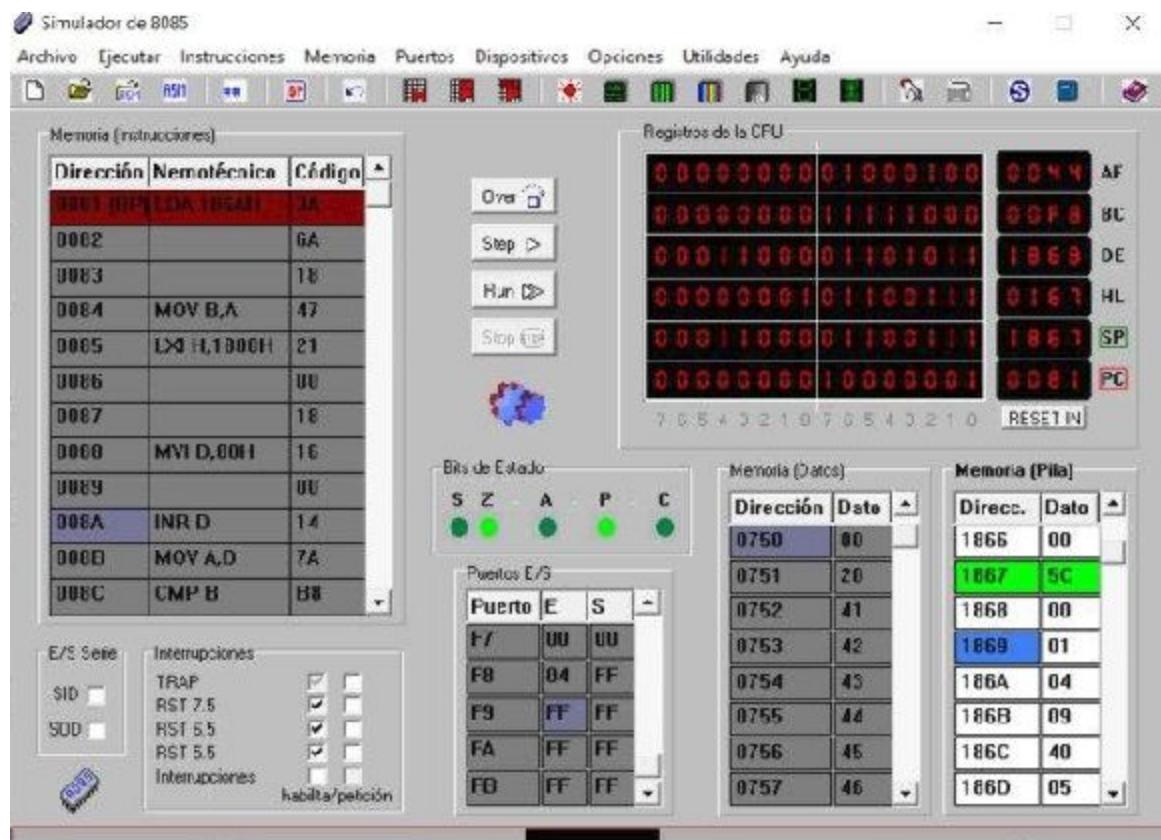


Figura 21.

Una vez sabemos que código se va a ingresar es necesario indicarle al usuario que ingrese el monto que va a convertir para esto imprimimos por pantalla la palabra *BILL* como se mostró anteriormente.

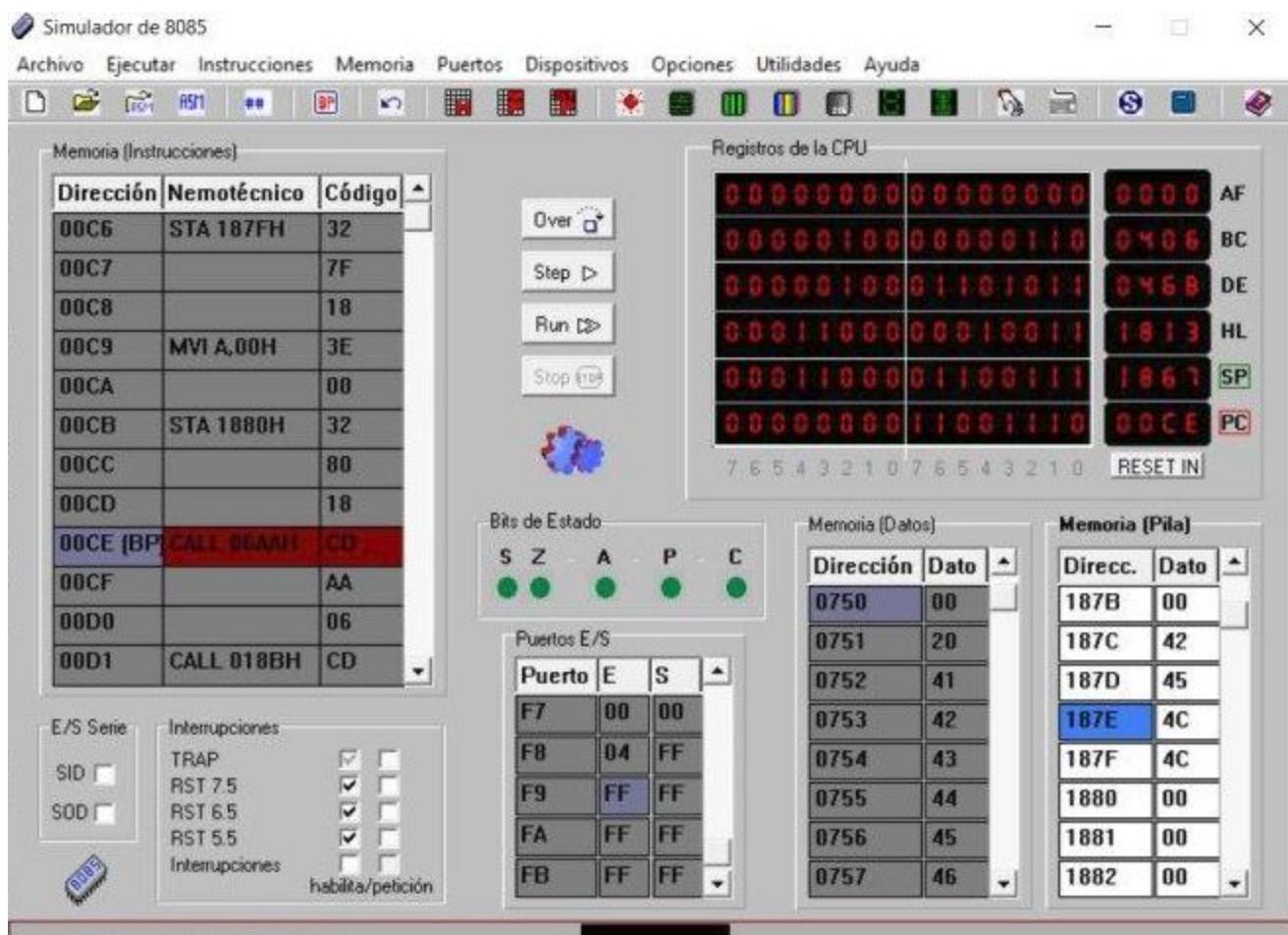


Figura 22.

En esta simulación ingresamos el valor 44

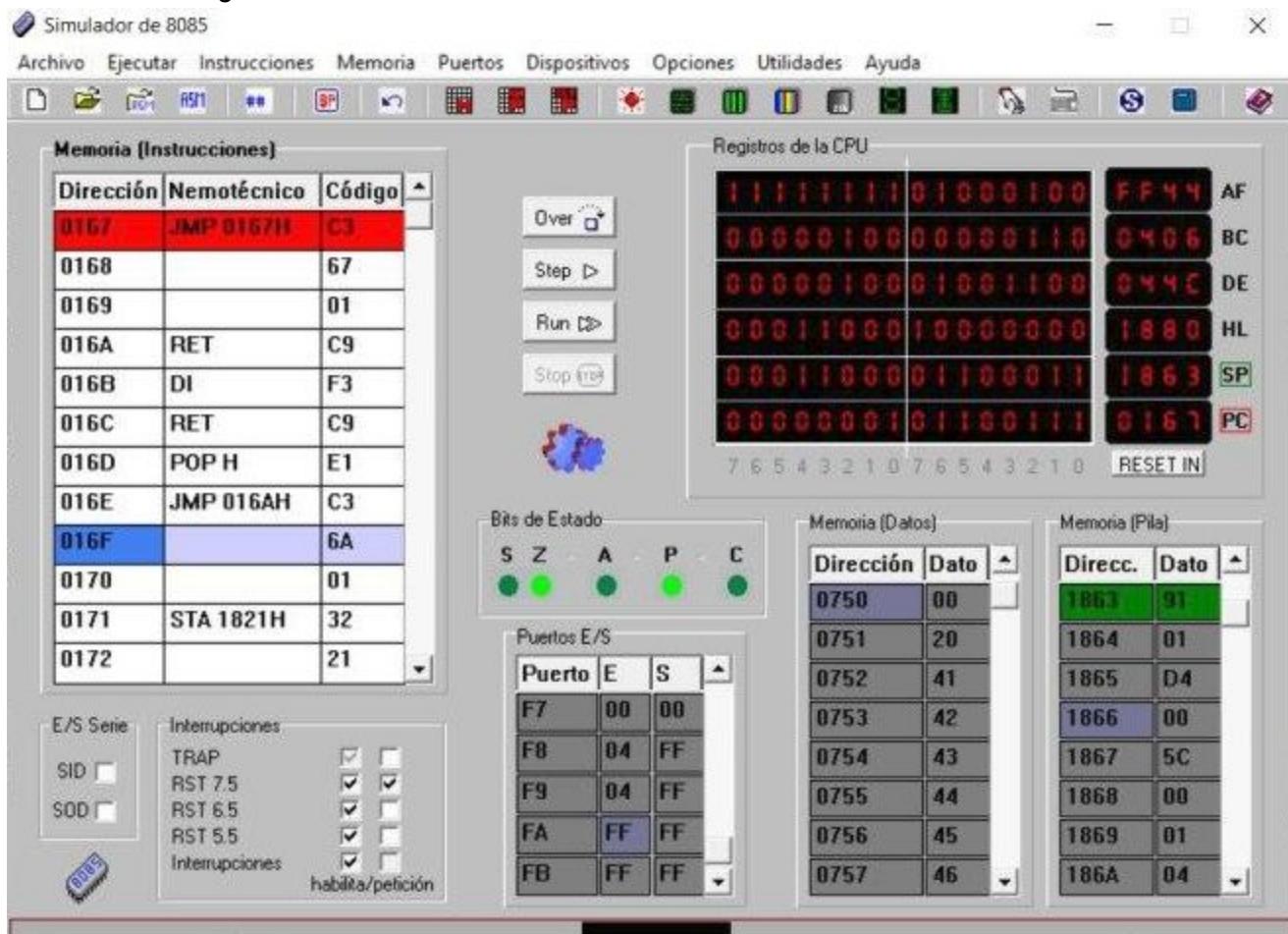


Figura 23.

El monto ingresado es ajustado, por lo que podemos ver en el buffer correspondiente a la rutina *ADJUST* (ver sección *processos*) el monto ajustado.

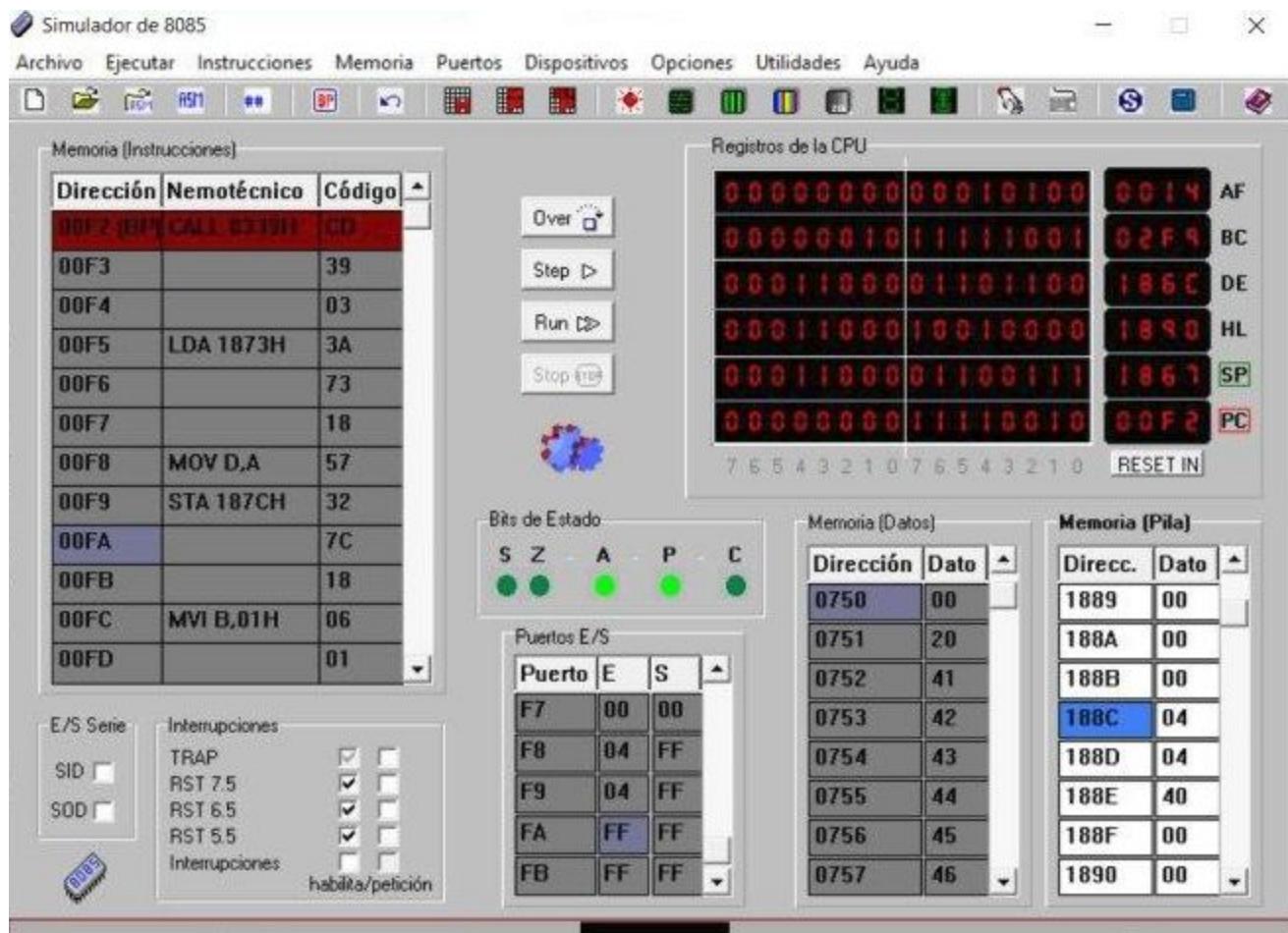


Figura 24.

En este punto ya podemos hacer la multiplicación solo resta colocar los valores de los operando en el buffer correspondiente (ver sección memoria). En la siguiente captura podemos observar como los operandos se encuentran en el buffer correspondiente.

El valor 20.3 es el primer operando, obtenido con el código ingresado (4) y buscando en la tabla de tasas de cambio el valor de la tasa para el código deseado .

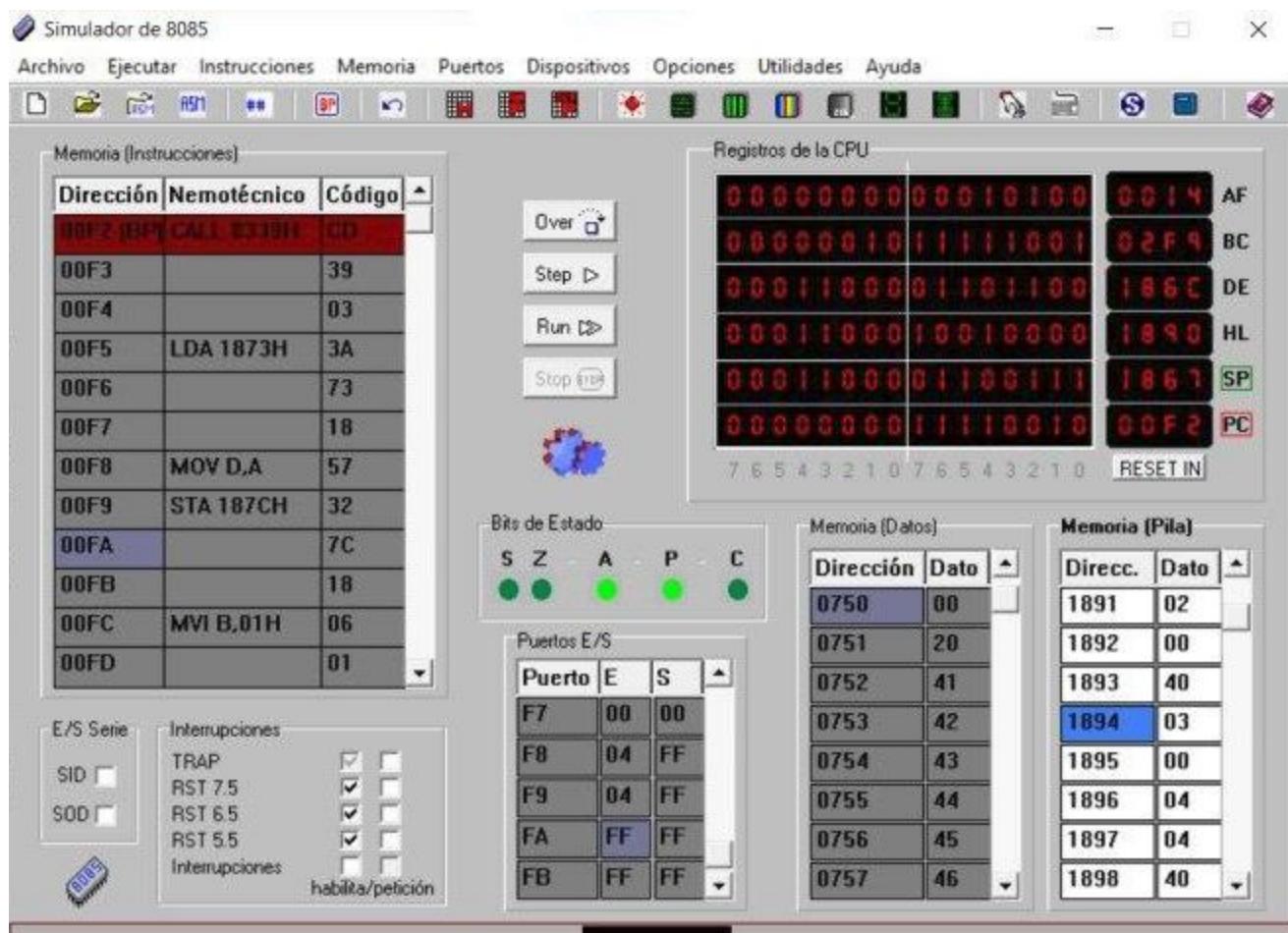


Figura 25.

El segundo operando tiene el valor 44.00 (monto ingresado que se desea convertir).

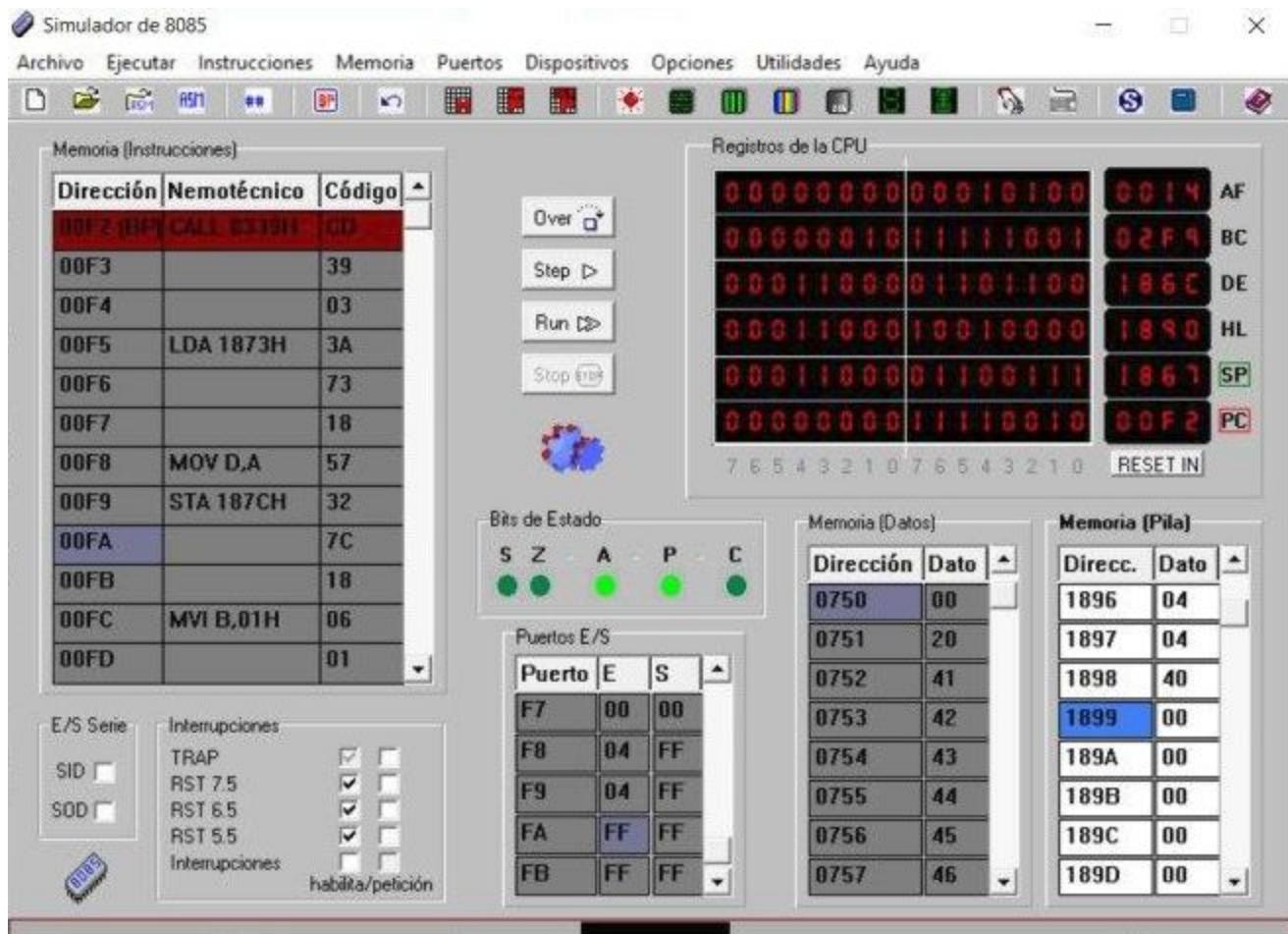


Figura 26.

Finalmente en el buffer de resultado podemos ver el resultado de la operación  $20.3 * 44 = 893.2$ .

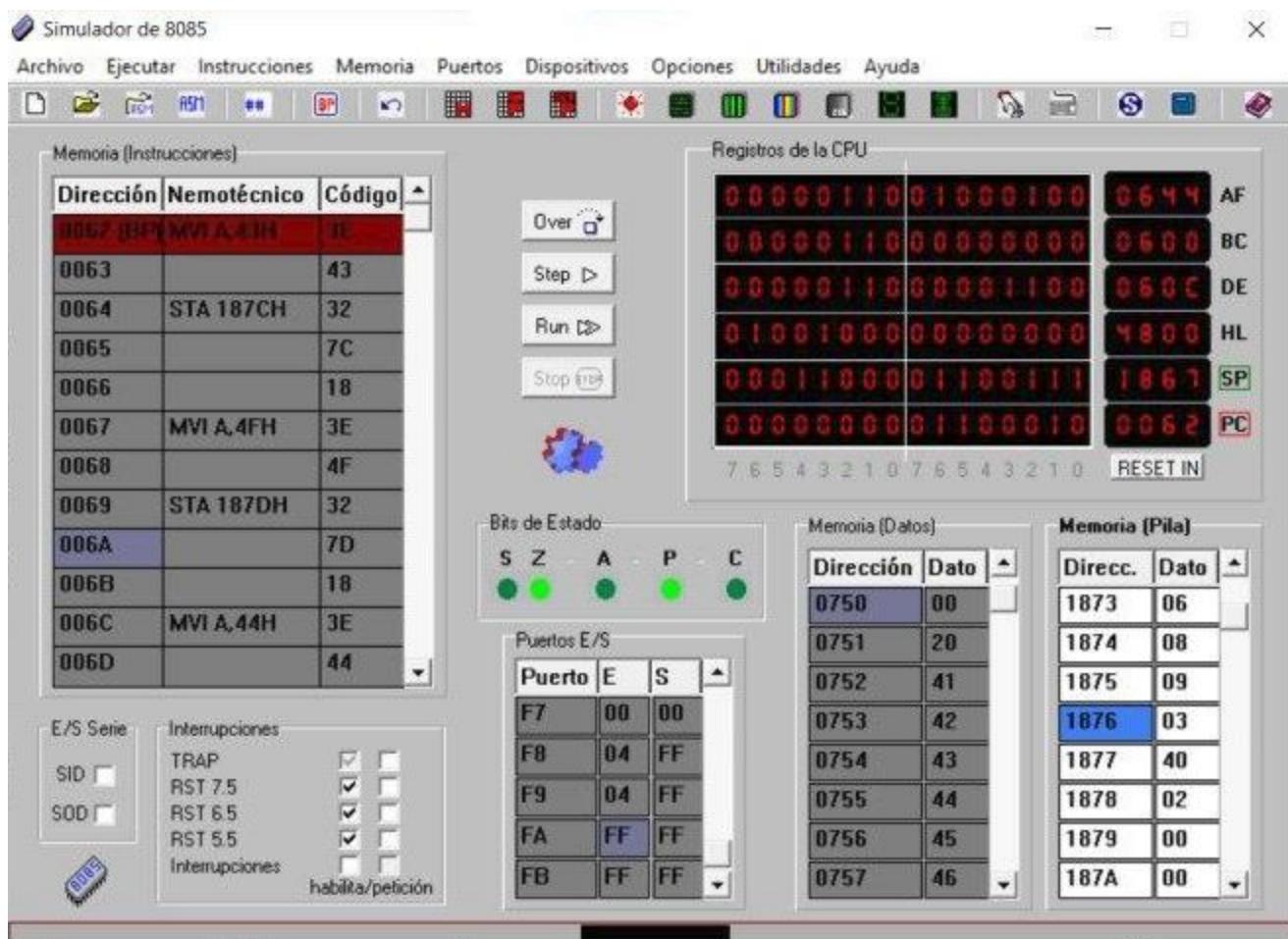


Figura 27.

Con esto la ejecución del programa principal a terminado, sin embargo al encontrarse en un ciclo infinito la rutina *PROGRAM* se ejecutará hasta que el sistema *TronOS* se apague.