

Informe de la práctica 6: Pila y entrada/salida

Introducción general

En esta 6ª práctica de Introducción a los Ordenadores encontraremos los objetivos de cada una de las partes, los ejercicios planteados y las conclusiones.

PARTE I

Los objetivos de esta primera parte de la práctica, que corresponde a la parte guiada son:

- *Estudio del espacio de memorias del microprocesador.*
- *Comprender el funcionamiento de la pila.*
- *Ver un ejemplo de uso de subrutinas.*

Códigos en i8085 y RISC V

Ejecutad paso a paso el siguiente programa en i8085:

i8085	RISC V (como ayuda)
.define	.data
num 02h	mat1: .byte 1, 2
.data 00h	mat2: .byte 3, 4
mat1: db 1,2	mat3: .byte 0, 0
mat2: db 3,4	.text
mat3: db 0,0	li s0, 2
.data 20h	la a1, mat1
pila:	la a2, mat2
.org 600h	loop:

```
LXI H, pila
SPHL
MVI B, num
LXI D, mat1
LXI H, mat2
```

```
jal suma
addi s0, s0, -1
bgtz s0, loop
j end
```

loop:

```
CALL suma
DCR B
JNZ loop
NOP
HLT
```

suma:

```
addi sp, sp, -12
sw s1, 12(sp)
sw s2, 8(sp)
sw s3, 4(sp)
```

suma:

```
PUSH PSW
LDAX D
ADD M
STAX D
INX H
INX D
POP PSW
RET
```

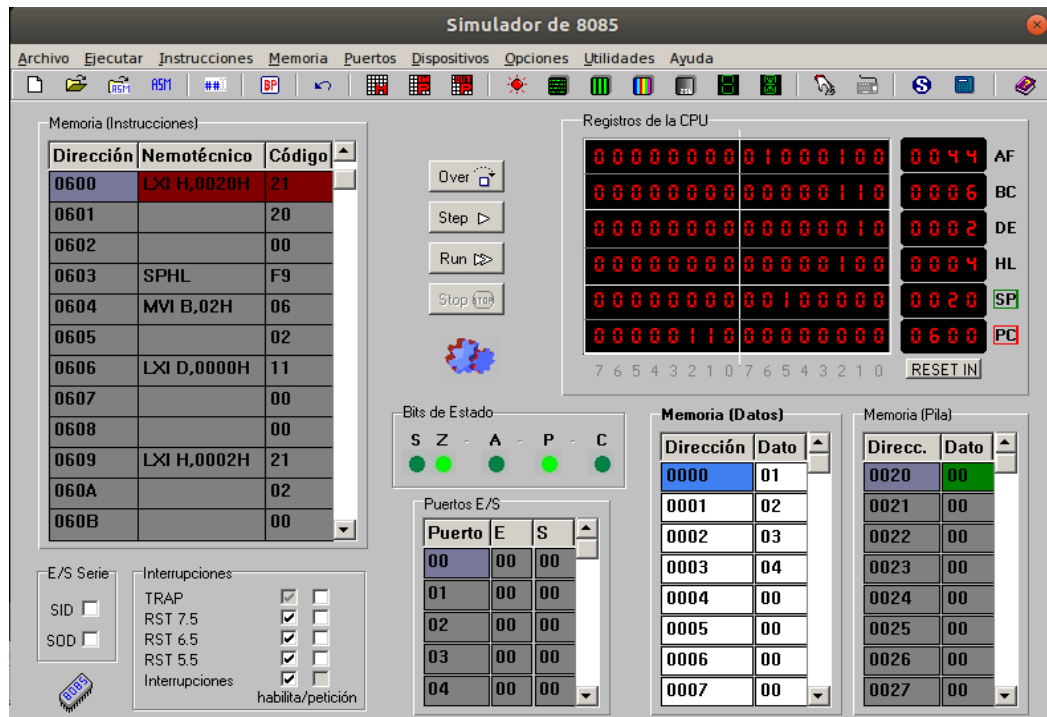
```
lb s1, 0(a1)
lb s2, 0(a2)
add s3, s1, s2
sb s3, 0(a1)

addi a1, a1, 1
addi a2, a2, 1

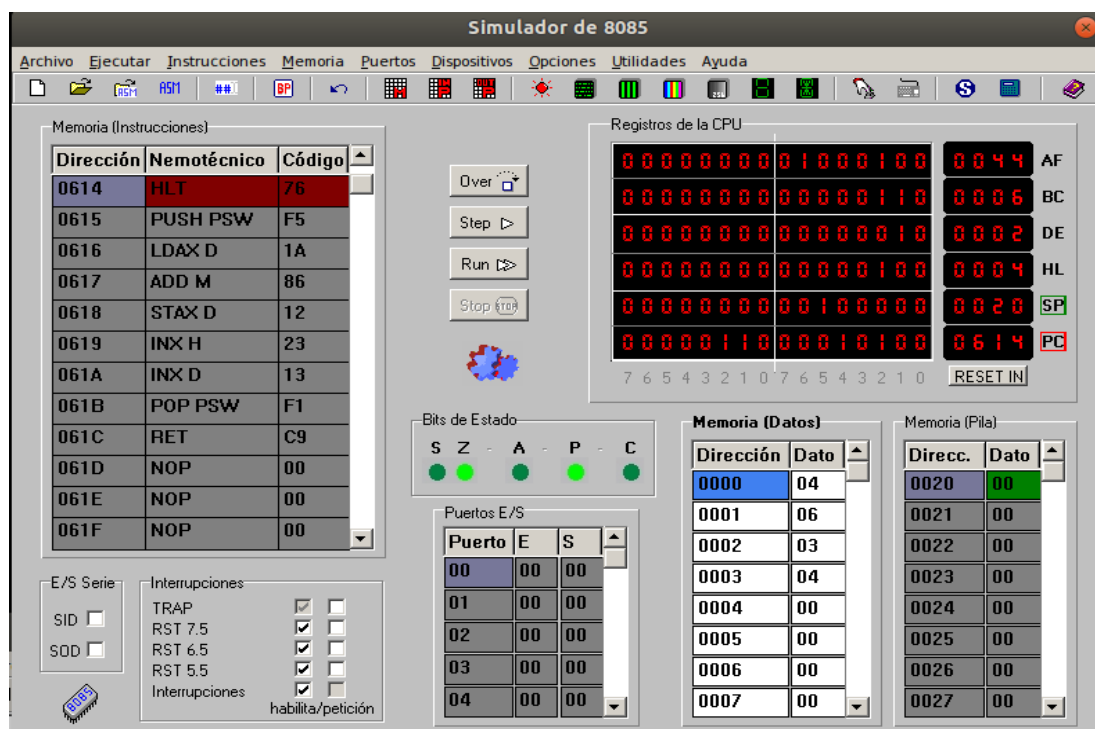
lw s3, 4(sp)
lw s2, 8(sp)
lw s1, 12(sp)
addi sp, sp, -12
ret
```

end:

Este programa realiza la suma de 2 matrices 2x1, guardando el resultado en la primera de ella sobrescribiéndola:



Simulador antes de sumar las matrices"



"Simulador después de realizar la suma y guardar en mat1"

Preguntas

1. El direccionamiento de la instrucción LXI es:

- a) directo
- b) indirecto
- c) inmediato
- d) implícito

2. ¿Qué instrucción guarda el PC en la Pila?

- a) PUSH PC
- b) POP PC
- c) CALL
- d) MOV M, PC

3. ¿Qué espacio ocupa en memoria la subrutina 'suma'?

La subrutina «suma» ocupa, en memoria, desde la posición 615h (PUSH PSW) hasta la posición 61Ch (RET):

Memoria (Instrucciones)		
Dirección	Nemotécnico	Código
060C	CALL 0615H	CD
060D		15
060E		06
060F	DCR B	05
0610	JNZ 060CH	C2
0611		0C
0612		06
0613	NOP	00
0614	HLT	76
0615	PUSH PSW	F5
0616	LDAX D	1A
0617	ADD M	86

"Llamada a la subrutina «suma»"

Memoria (Instrucciones)		
Dirección	Nemotécnico	Código
0615	PUSH PSW	F5
0616	LDAX D	1A
0617	ADD M	86
0618	STAX D	12
0619	INX H	23
061A	INX D	13
061B	POP PSW	F1
061C	RET	C9
061D	NOP	00
061E	NOP	00
061F	NOP	00
0620	NOP	00

"Inicio de la subrutina «suma»"

4. ¿Cuántos ciclos tarda en ejecutarse la subrutina 'suma'?

Tenemos las siguientes instrucciones con sus respectivos ciclos:

PUSH PSW → 12 ciclos

LDAX D → 7 ciclos

ADD M → 4 ciclos

STAX D → 7 ciclos

INX H → 6 ciclos

INX D → 6 ciclos

POP PSW → 10 ciclos

RET → 10 ciclos

Sabemos que esta subrutina se implementa 2 veces, ya que nuestro contador «num» es 2. Por lo tanto, tenemos:

$$[12 + (7 \times 2) + 4 + (6 \times 2) + (10 \times 2)] \times 2 = 62 \times 2 = 124 \text{ ciclos totales}$$

Es decir, 62 ciclos por cada una de las veces que la implementamos.

Estudio del espacio de memorias del microprocesador

TAREA 1

Dibujad el mapa de memoria de datos: direcciones y contenido. Indicad las instrucciones que modifican los datos de la memoria. En cada una de ellas, indicad qué modificaciones se producen.

Situad la memoria de programa. Dentro de ella, localiza el sub-bloque que pertenece a la subrutina 'suma'.

Dibujaré el mapa de memoria de datos a través de unas tablas. En ellas estudiaré 3 apartados:

- La posición de memoria en la que estamos. [DIRECCIÓN]
- El contenido de dicha posición de memoria. [CONTENIDO]
- Las modificaciones que se llevan a cabo. [MODIFICACIONES]

DIRECCIÓN	CONTENIDO	MODIFICACIONES (al ensamblar)
00h	01	Colocamos el 1º número de mat1, que es 1
01h	02	Colocamos el 2º número de mat1, que es 1
02h	03	Colocamos el 1º número de mat2, que es 3
03h	04	Colocamos el 2º número de mat2, que es 4
04h	00	Colocamos el 1º número de mat3, que es 0, por eso no varía
05h	00	Colocamos el 2º número de mat3, que también es 0, por eso no varía

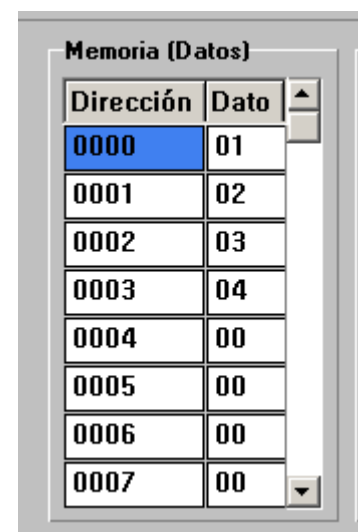
Esta 1ª tabla corresponde al siguiente código, cuyas posiciones en la memoria de datos son:

.data 00h

mat1: db 1,2 ; posiciones 00h y 01h

mat2: db 3,4 ; posiciones 02h y 03h

mat3: db 0,0 ; posiciones 04h y 05h



Dirección	Dato
0000	01
0001	02
0002	03
0003	04
0004	00
0005	00
0006	00
0007	00

Además los datos también aparecen en la memoria de la pila en las mismas posiciones.

"Matrices en la memoria de datos"

Después tenemos la posición 20h, que corresponde a la pila. En ella, aparentemente no hay cambios pero, para cuando ejecutemos la instrucción «CALL suma», si debemos comentar lo siguiente:

DIRECCIÓN	CONTENIDO	MODIFICACIONES (al ensamblar)
1Dh	00h	Guardaremos ACC y STT
1Eh	00h	Guardaremos ACC y STT
1Fh	00h	Guardaremos el PC
20h	00h	Guardaremos el PC

Ahora procederemos a estudiar el programa en si, que comienza en la posición de memoria 600h y termina en la 61Ch:

M E M O R I A D E I N S T R U C C I O N E S		
DIRECCIÓN	CONTENIDO	MODIFICACIONES (al ensamblar)
600h	LXI H, 0020H	Carga la posición de memoria de la pila en el par de registros HL \leq 0020h
601h		
602h		
603h	SPHL	Pone la pila en la posición de memoria indicada por los registros HL, que es 0020h
604h	MVI B,02H	Carga el contador «num» en el par de registros BC \leq 02h = 2
605h		
606h	LXI D, 0000H	Carga mat1 en el par de registros DE \leq mat1 = 0000h (1ª posición)
607h		
608h		
609h	LXI H, 0002H	Carga mat2 en el par de registros HL \leq mat2 = 0002h (1ª posición)
60Ah		
60Bh		
E T I Q U E T A « L O O P »		
60Ch	CALL 0615H	Llamamos a la subrutina suma y guardamos el PC en la pila (saltamos a la posición 615h)
60Dh		
60Eh		
60Fh	DCR B	Decrementamos el contador (1ª será de 2 a 1 y, en la 2ª vez, será de 1 a 0)

DIRECCIÓN	CONTENIDO	MODIFICACIONES (al ensamblar)
610h	JNZ 060CH	Mientras el contador no sea 0, volvemos a 60Ch (CALL suma); si es 0, a 613h
611h		
612h		
613h	NOP	No modifica nada.
614h	HLT	Fin del programa.
S U B R U T I N A « S U M A »		
615h	PUSH PSW	Copia 2 bytes en el stack que, como la pila avanza a la inversa, pasamos de 001Eh a 001Ch
616h	LDAX D	Carga el contenido de los registros DE en acumulador (1º será 1 y, en la 2ª vez que se realiza la subrutina, será 2)
617h	ADD M	Sumamos el contenido de A con el de HL (1º será 3 y, en la 2ª vez que se realiza la subrutina, será 4), guardándolo en A
618h	STAX D	Guardamos el resultado de la suma en mat2, que estaba en [DE] <= A
619h	INX H	Incrementamos la posición de memoria de los registros HL (de 02h a 03h)
61Ah	INX D	Incrementamos la posición de memoria de los registros DE (de 00h a 01h)
61Bh	POP PSW	Deshacemos lo que hicimos con la instrucción PUSH PSW: pasamos de la posición 001Ch a la posición 001Eh
61Ch	RET	Recuperamos el valor del PC y saltamos a la instrucción posterior a «CALL suma» (retroceder a la posición 60Fh)

Como podemos observar, la subrutina «suma» se encuentra en las posiciones de memoria 615h hasta la 61Ch, siendo las últimas de la tabla presentada más arriba.

Funcionamiento de la Pila

TAREA 2

Indicad el comienzo de la pila en el espacio de la memoria de datos del microprocesador.

La pila, en un inicio, se sitúa en la posición 00h para colocar el contenido de las variables mat1 y mat2:

- 00h <= 1
- 01h <= 2
- 02h <= 3
- 03h <= 4)

Aún así, una vez ejecutemos la instrucción SPHL, la pila se situará en la posición de memoria indicada en los registros HL: 0020h.

Además, para el siguiente apartado de la tarea 2, hay que mencionar que la pila avanza en sentido decreciente; es decir, si empezamos en la posición 20h, las siguientes posiciones de la pila serán 1Fh, 1Eh, 1Dh, 1Ch... Por lo tanto, las 2 primeras posiciones (1Fh y 1Eh) serán para guardar el PC y las 2 siguientes (1Dh y 1Ch), para el acumulador y el registro de estado.

Indicad qué instrucciones modifican la pila. Para cada una, completad la tabla siguiente:

En esta tabla aparecen todas las instrucciones de la subrutina para mostrar una mayor continuidad del código. En el caso de las instrucciones que NO realizan cambios en la pila, ésto se indica en el apartado «Cambio en la pila» :

Instrucción	Descripción	Cambio en la pila
SPHL	Pone la pila en la posición de memoria indicada por los registros HL, que es 0020h	Pasa de situarse en la posición de memoria 00h a la 20h.
CALL suma	Llamamos a la subrutina suma y guardamos el PC en la pila	Guardamos el PC en la pila, pasando de 20h a 1Eh , que son 2 bytes

Instrucción	Descripción	Cambio en la pila
PUSH PSW	Añade PSW en la pila.	Aumenta en 2 bytes, es decir, pasamos de 1Eh a 1Ch.
LDAX D	Carga el contenido de los registros DE en acumulador (1º será 1 y, en la 2ª vez que se realiza la subrutina, será 2)	No produce cambios en la pila, porque aun estamos trabajando con el acumulador. (Seguimos en 1Ch)
ADD M	Sumamos el contenido de acumulador con el de HL (1º será 3 y, en la 2ª vez que se realiza la subrutina, será 4), guardándolo en A	No produce cambios en la pila, porque aun estamos trabajando con el acumulador. (Seguimos en 1Ch)
STAX D	Guardamos el resultado de la suma en mat2, que estaba en [DE] <= A	Sobreescribimos la posición 00h de la pila con el resultado de la suma $01+03 = 04$
INX H	Incrementamos la posición de memoria de los registros HL (de 02h a 03h)	No produce cambios en la pila, porque estamos aumentando la posición de los registros HL.
INX D	Incrementamos la posición de memoria de los registros DE (de 00h a 01h)	No produce cambios en la pila, porque estamos aumentando la posición de los registros DE.
POP PSW	Retira PSW de la pila	Decrementa en 2 bytes, es decir, pasamos de la posición 1Ch a la posición 1Eh
RET	Recuperamos el valor del PC y saltamos a la instrucción posterior a «CALL sumar»	Decrementamos en 2 bytes, es decir pasamos de la posición 1Eh a la posición original 20h.

PARTE II

Los objetivos de esta segunda parte de la práctica, que corresponde ya a la parte no guiada son:

- Comprender el direccionamiento a puertos E/S.

El simulador permite comunicar el procesador con una serie de puertos E/S:

- Puertos de entrada: interruptores y teclado
- Puertos de salida: panel de LEDs, displays de 7 y 15 segmentos

Para acceder a estos recursos, debemos acceder a las direcciones adecuadas de los puertos.

Código

Ejecutad paso a paso el siguiente programa:

```
.data 100h  
    pila:  
    .org 24h  
    JMP ports  
    .org 500h  
    LXI H, pila  
    SPHL  
    CALL ports  
    NOP  
    HLT  
ports:  
    PUSH PSW  
    IN 04h  
    ANI 00000001  
    OUT 05h  
    POP PSW  
    RET
```

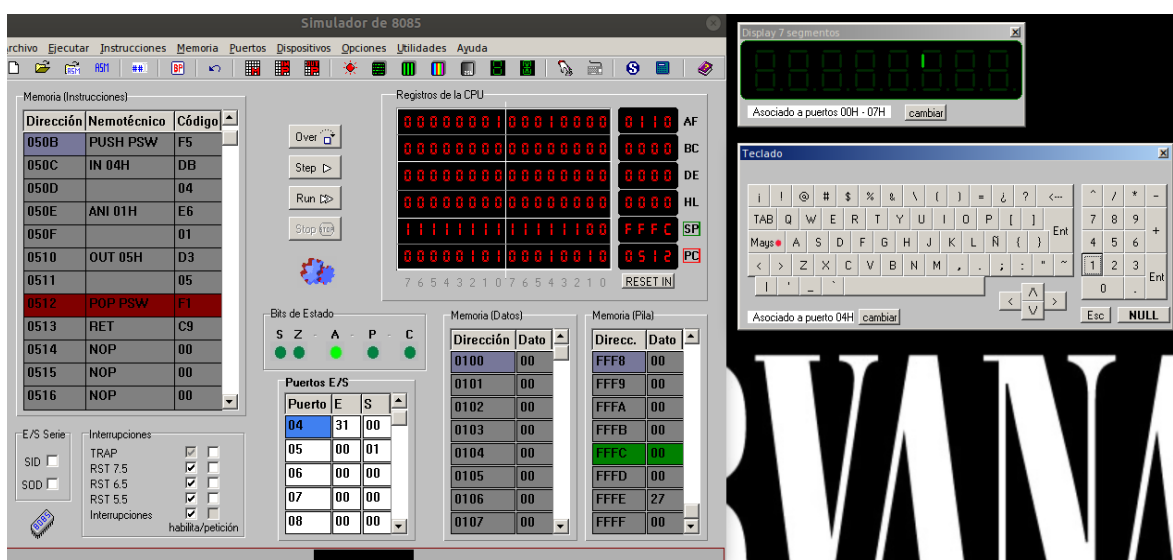
TAREA 3

¿Qué hace la subrutina 'ports'? Para ello, introducid datos con los interruptores o con el teclado; observad en un puerto de salida el resultado de la subrutina.

La subrutina «ports» lo que hace es:

1. Primero, aumenta la pila en 2 bytes con la instrucción PUSH PSW.
2. Segundo, mediante la instrucción IN 04h, coge el valor que introducimos por teclado en el puerto de entrada de 04h y lo guarda en acumulador. En este caso, he introducido «1» por teclado (la entrada será 31h).
3. Después ejecuta la instrucción ANI 00000001, que hace la operación lógica AND entre el contenido de acumulador y el valor 00000001, guardando el resultado en acumulador. Dicho resultado será 1.
4. El resultado guardado en acumulador se mostrará a través del puerto de salida de 05h, gracias a la instrucción OUT 05h.
5. Por último, decrementa la pila en 2 bytes con el POP PSW y retornamos el PC con RET

Para que se vea de forma más clara mostraré el display de 7 segmentos, el teclado con la tecla «1» presionada y asociado al puerto 04h y los cambios de los puertos E/S:



"Subrutina «ports» después de ejecutar OUT 05h, con el display de 7 segmentos"

PARTE III

Los objetivos de esta tercera y última parte de la práctica son:

- Diseñad un programa *assembler* que represente en un *display* de 7 segmentos los números del 0 al 5. Los números se introducirán a partir del teclado. Además, tiene que permitir la opción de borrar el display; para ello, al presionar la letra 'c', se producirá un CLEAR del *display*.

Propuestas de mejora:

El simulador 8085 tiene una parte de la memoria de datos que implementa una 'Pantalla de texto'. Modificad el programa para escribir la letra que quereis en la pantalla. En el menú "Opciones" → "de interrupciones" podeis habilitar que se genere una interrupción TRAP al presionar cualquier tecla del teclado. Ésto producirá el salto de la ejecución del programa a la subrutina que haya en la dirección de memoria 0024h. Aprovechad esta posibilidad para reproducir el funcionamiento continuo del teclado y la pantalla.

TAREA 4

Subid un fichero con:

- Explicación breve del algoritmo escogido.
- Programa ensamblador comentado.

El código expuesto más abajo representa, en un *display* de 7 segmentos, los números del 0 al 5 y hace un *clear* del display cuando se presiona la letra «C.» Para ello he asignado a unas variables iniciales llamadas *displayNúmero*, los números binarios que encienden los segmentos que representarían dichos números. De una forma más sencilla:

En «*displayCero*: db 0111011b» tenemos el número 77h en binario, que representa las 6 barritas que formarían el 0 en el display de 7 segmentos.

Lo mismo pasa con los display de 1 (44h), 2 (3Eh), 3 (6Eh), 4 (4Dh) y 5 (6Bh).

Además también tenemos el *display* para el *clear*, que será 0 para que no se encienda nada en el *display*.

En cuanto al programa, primero debemos resaltar que empezamos en la posición 24h debido al uso de la instrucción TRAP. Luego encontraremos un bucle infinito llamado *display*.

Dentro de este bucle, recogemos el valor pasado por el puerto de entrada de 00h a través del teclado. Si nos fijamos tenemos la siguiente correspondencia tecla - valor hexadecimal - memoria:

TECLA	0	1	2	3	4	5	C
VALOR HEXADECIMAL	30h	31h	32h	33h	34h	35h	43h
Posición en la memoria	00h	01h	02h	03h	04h	05h	13h

Con esta estrategia, colocaremos en esas posiciones de memoria las variables para el *display*.

A continuación, hacemos una resta del valor recogido en "IN 00h" y 30h para obtener la posición de memoria que nos interesa. Si el resultado coincide con alguno de los expuesto en la tabla, el display mostrará el número o lo limpiará, si introducimos «C». En el caso contrario, no mostrará nada ya que esa posición de memoria no representa nada dentro de nuestras variables.

Por último, volveremos a empezar el bucle hasta una nueva entrada.

Este es el código comentado:

```
.data 00h                                ; Números para el display
    displayCero: db 01110111b           ; 77h que será 0 en el display
    displayUno:  db 01000100b           ; 44h que será 1 en el display
    displayDos:   db 00111110b           ; 3Eh que será 2 en el display
    displayTres:  db 01101110b           ; 6Eh que será 3 en el display
    displayCuatro: db 01001101b         ; 4Dh que será 4 en el display
    displayCinco: db 01101011b         ; 6Bh que será 5 en el display
```

```
.data 13h                                ; Clear del código

displayClear: db 00000000b              ; 00h que limpiará el display

.org 24h                                  ; Usamos la TRAP instruction

display:                                  ; bucle infinito

    IN 00h                               ; recogemos el valor introducido, a través del
                                          ; teclado, en el puerto de entrada de 00h,
                                          ; guardándolo en acumulador

    SUI 30h                              ; restamos acumulador y 30h, guardando el
                                          ; resultado otra vez en acumulador

    MOV E, A                             ; movemos el contenido de A al registro E

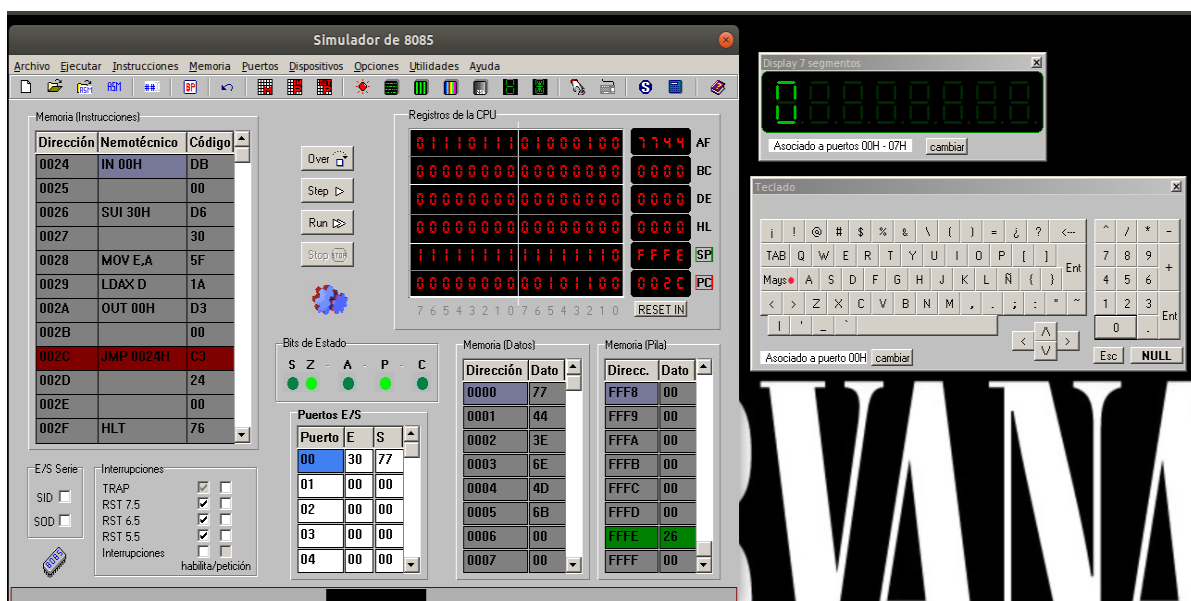
    LDAX D                               ; cargamos el contenido que hay en la posición de
                                          ; memoria guardada en [DE] en acumulador

    OUT 00h                              ; mostramos en el display, a través del puerto de
                                          ; salida de 00h, el valor guardado en A

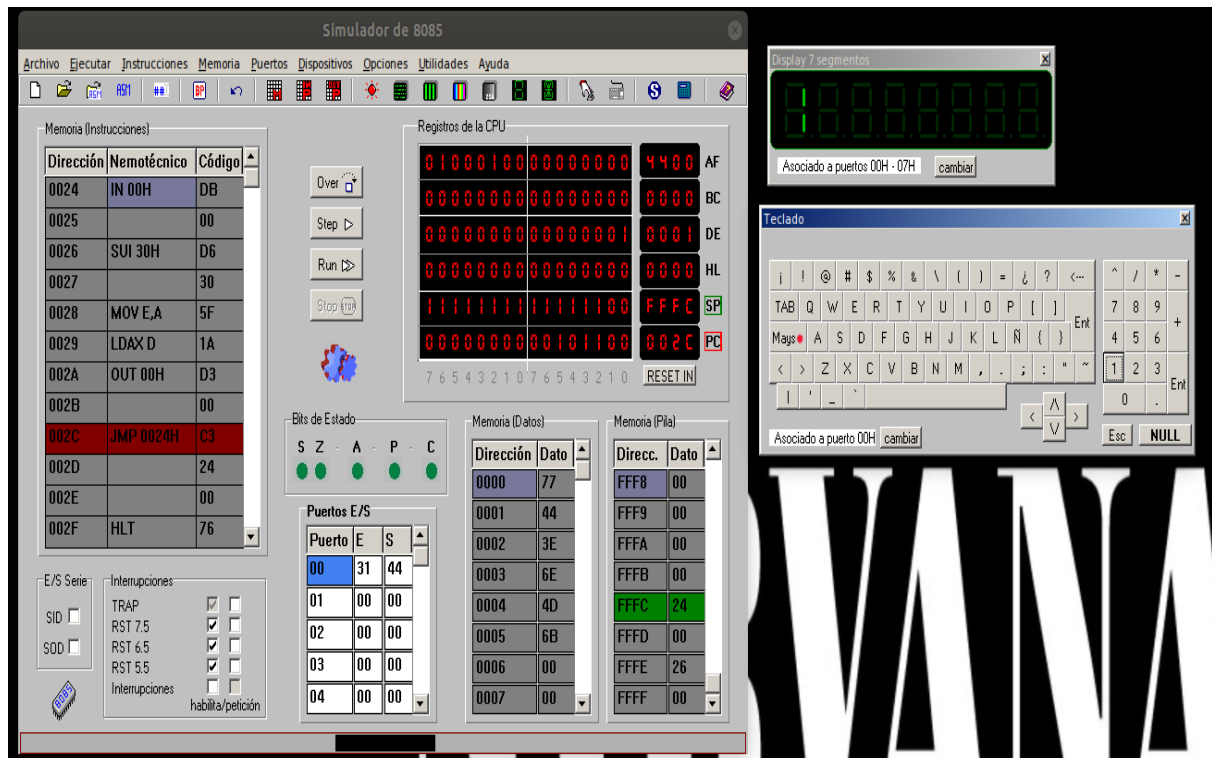
    JMP display                           ; saltamos incondicionalmente a display

    HLT                                  ; fin
```

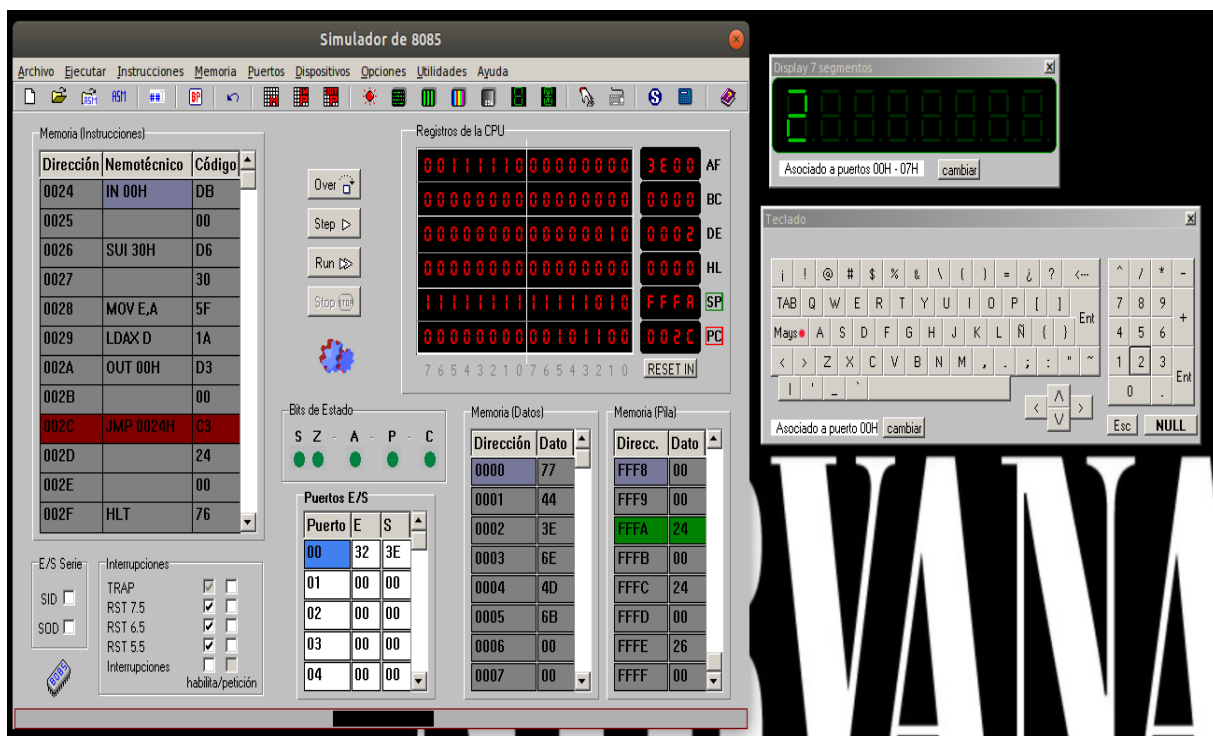
Véamos como funciona el código a través de unas imágenes ilustratorias:



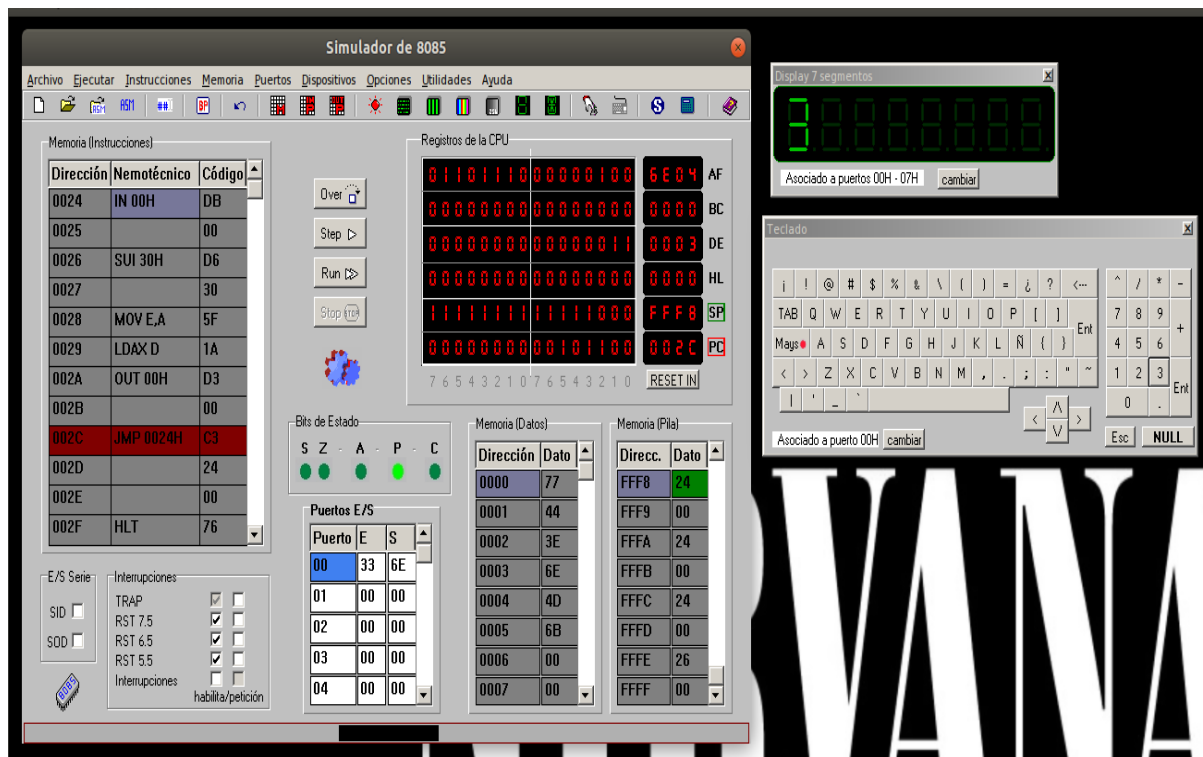
"Display de 7 segmentos con el número 0"



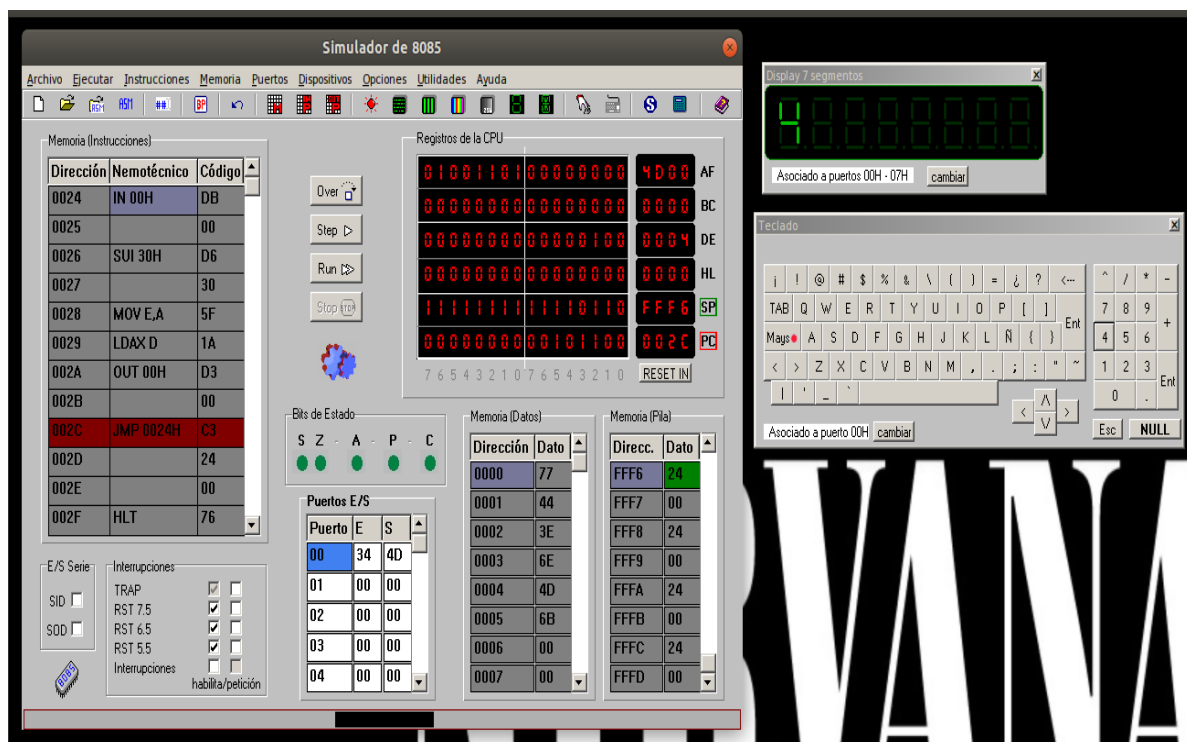
"Display de 7 segmentos con el número 1"



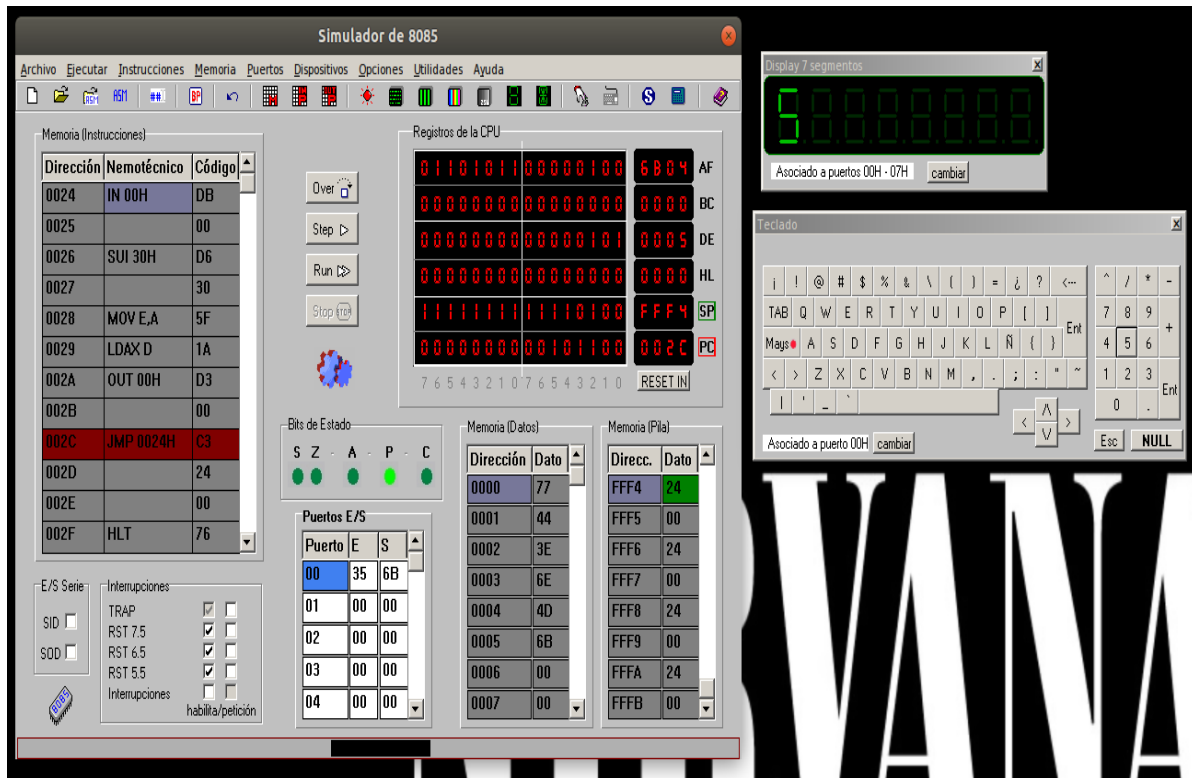
"Display de 7 segmentos con el número 2"



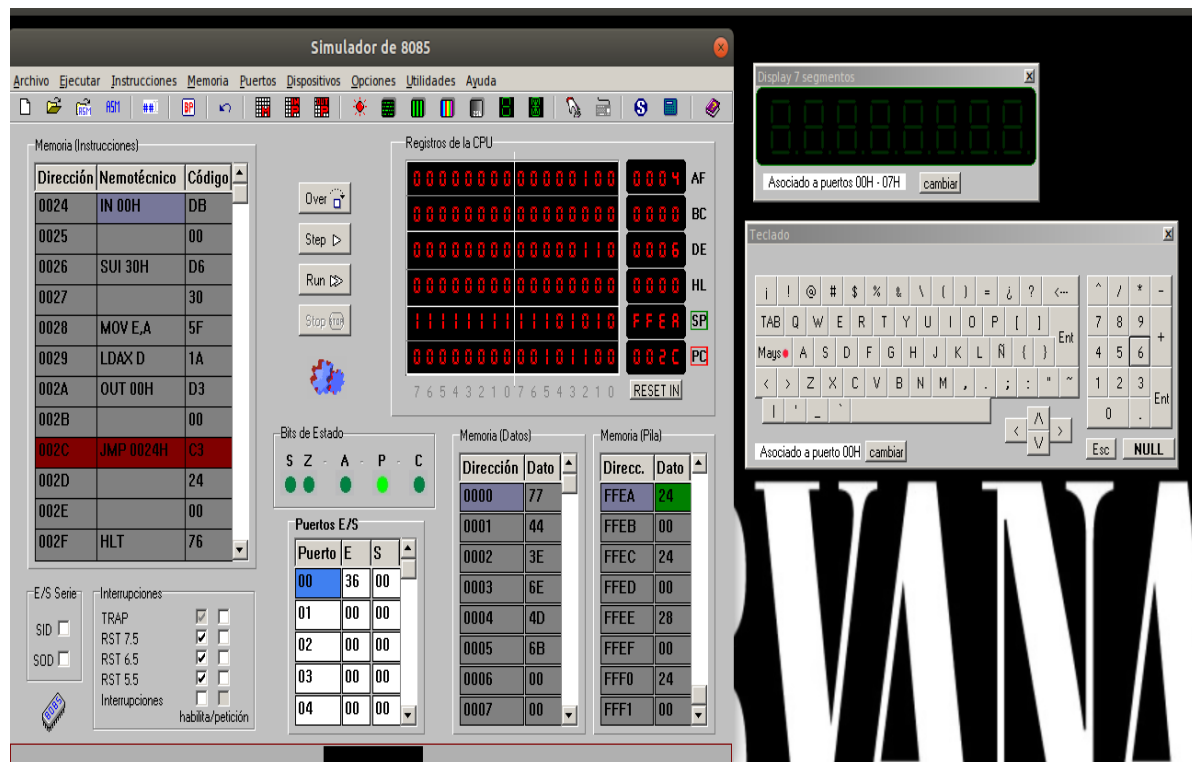
"Display de 7 segmentos con el número 3"



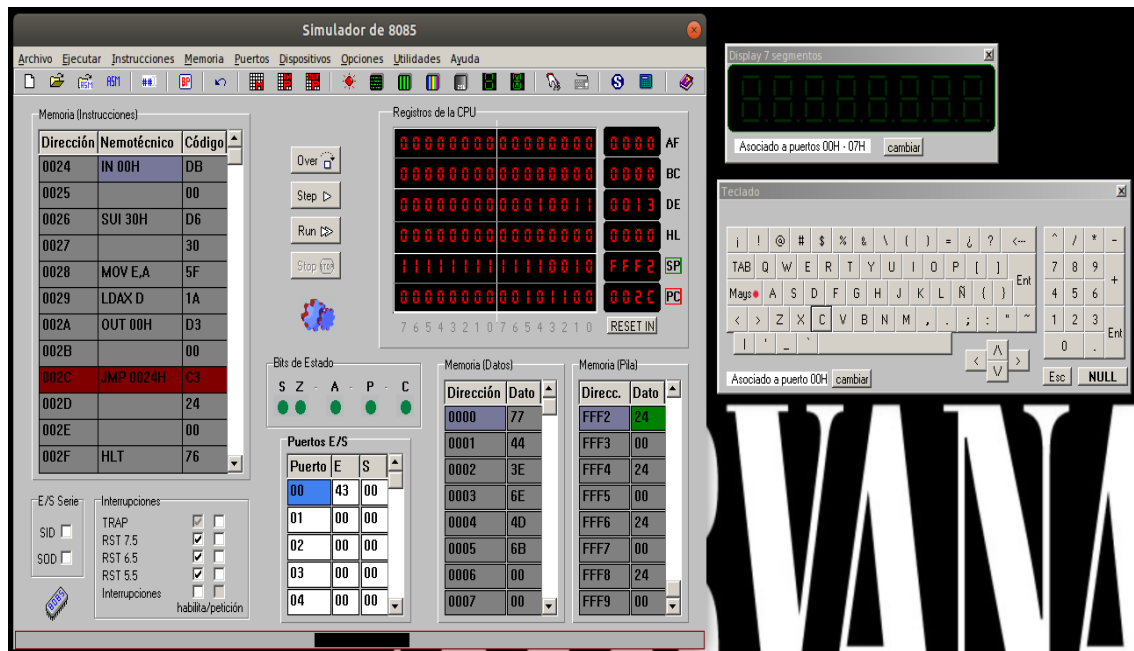
"Display de 7 segmentos con el número 4"



"Display de 7 segmentos con el número 5"



"Display de 7 segmentos con el número 6
(no funciona porque no está implementado)"



"Display de 7 segmentos con la letra C "

Conclusiones

En esta práctica hemos profundizado en el uso y estudio de las subrutinas con i8085, además de introducir los displays, teclado, etc. Para ello hemos necesitado:

- Realizar las 3 partes de este informe, con sus respectivas tareas, tablas y explicaciones.
- Seguir el directo, donde se explicaba la práctica, además de consultar los PDFs de i8085 (power point que explica el simulador, pdf de las instrucciones, guía de prácticas ...)
- En cuanto a la PARTE 1: he mirado la guía de instrucciones para poder explicar las tablas de la forma más clara posible y la información sobre la pila que hay en el campus.
- En cuanto a la PARTE 2: ha sido un continuo prueba y error con el simulador i8085, ejecutando el código paso a paso y anotando los cambios en la pila, los puertos y los registros.
- En cuanto a la PARTE 3: he buscado información en internet para complementar la del directo de prácticas, además de ir haciendo pruebas con el simulador y los displays.

En resumen, doy por cumplidos los objetivos propuestos pero en cuanto a los displays, falta trabajar en ellos.