UNIVERSITAT DE BARCELONA

Pràctica 6: Pila i entrada/sortida

Alejandro Guzman

19 Maig 2022

ÍNDEX

1 OBJECTIUS

Aquesta és la segona pràctica que realitzarem amb el simulador i8085 i la sexta de l'assignatura Introducció als Ordinadors.

Dividirem la pràctica en tres parts diferenciades, en la primera realitzarem un estudi de l'espai de memòries del microprocessador i ens ajudarà a comprendre el funcionament de la pila. També veurem un exemple d'utilització de subrutines. En la segona part l'objectiu principal és comprendre l'adreçament a ports E/S i com funciona l'entrada i sortida en el simulador i8085. Finalment en la tercera part dissenyarem un programa *assembler* que representi en un display de 7 segments els números del 0 al 5. Els números s'introduiran a partir del teclat. A més, ha de permetre l'opció d'esborrar el display; per això, en prémer la lletra 'c', es produirà un CLEAR del display.
En global aquesta pràctica té com a principal objectiu aprendre a utilitzar la pila i conèixer els conceptes d'entrada i sortida i com podem utilitzar-los en el simulador per tal de mostrar contingut en un display, etc.

2 QÜESTIONS PLANTEJADES AL GUIÓ DE PRÀCTIQUES

1 Part 1

En la primera part se'ns proporciona un programa escrit en els dos simuladors (el codi en Ripes és de reforç per entendre el programa). El codi en qüestió és el següent

```
.define
    num 02h
.data 00h
    mat1: db 1,2
    mat2: db 3,4
    mat3: db 0,0
.data 20h
    pila:
.org 600h
    LXI H, pila
    SPHL
    MVI B, num
    LXI D, mat1
    LXI H, mat2
loop:
    CALL suma
    DCR B
    JNZ loop
    NOP
    HLT
suma:
    PUSH PSW
    LDAX D
    ADD M
    STAX D
    INX H
    INX D
    POP PSW
    RET
```

L'algorisme en si realitza una suma de dues matris 2 x 1 i guarda el resultat sobreescrivint la primera matriu. La suma es realitza en una subrutina que es crida un total de dos cops (el total d'elements de cada matriu), aquesta realitza la suma en l'acumulador i guarda el resultat en la posició de memòria on apunta el registre D, que és l'element actual de l'iteració en la primera matriu.

De fet, ens pregunten com es faria per guardar el resultat en la matriu mat3, que en .data s'assigna un espai de memòria però mai s'arriba a utilitzar. Això es podria fer primer carregant mat1 a mat3 i després sumant mat2 amb mat3 i guardar-lo a mat3. El codi modificat és el següent:

```
.define
    num 02h
.data 00h
    mat1: db 1,2
    mat2: db 3,4
    mat3: db 0,0
.data 20h
pila:
    .org 600h
    LXI H, pila
    SPHL
    MVI A, num ; A <- num = 2
    LXI D, mat1 ; DE <- mat1
```

```
14      LXI H, mat2 ; HL <- mat2
15      LXI B, mat3 ; BC <- mat3
16  loop:
17      CALL suma
18      DCR A
19      JNZ loop
20      NOP
21      HLT
22  suma:
23      PUSH PSW
24      LDAX D ; A <- [DE]
25      ADD M ; A += [HL]
26      STAX B ;[BC] <- A
27      INX H
28      INX D
29      INX B
30      POP PSW
31      RET
```

En la implementació d'aquesta nova versió l'únic canvi que hem realitzat ha sigut afegir l'instrucció **INX B** en la subrutina 'SUMA' per tal d'accedir també als elements de mat3 i canviar **STAX D** per **STAX B** amb l'objectiu de guardar el resultat de la suma de cada element a mat3 enlloc de a mat1.

1.1 Preguntes del codi

Pregunta 1. Quin tipus d'adreçament utilitza la instrucció LXI?
L'adreçament de la instrucció LXI és immediat, ja que guarda a la parella de registres indicada un immediat de 16 bits (2 bytes) donat a la instrucció.

Pregunta 2. Quina instrucció guarda el PC a la Pila?
La instrucció que guarda el PC a la Pila és la instrucció CALL. Abans de saltar a la posició de memòria especificada a la instrucció (on comença la subrutina) guardem a la pila el valor actual del PC per, un cop executada la subrutina, recuperar-lo amb la instrucció RET.

Pregunta 3. Quin espai ocupa en memòria la subrutina 'suma'?
La subrutina **suma** té vuit instruccions i consultant la documentació pertinent sabem que cadascuna ocupa un *byte* de memòria, per tant la subrutina en total ocupa 8 *bytes* de memòria.

Pregunta 4. Quants cicles triga en executar-se la subrutina 'suma'?

Per a realitzar el càlcul de quants cicles triga en executar-se la subrutina 'suma', ho fem amb el codi modificat per guardar el resultat a mat3. Aquest codi només varia el nombre de cicles de 'suma' per una instrucció, la única que afegim per accedir als elements de mat3; **INX B**.

Usant el document adjuntat al campus virtual amb el conjunt d'instruccions de l'arquitectura que estem usant, podem observar el següent (considerem que la subrutina comença un cop hem saltat a la posició "suma", en cas contrari caldria afegir els 18 cicles que triga en executar-se la instrucció "CALL suma"):
Llavors el nombre total de cicles de la subrutina 'suma' del codi modificat és de 71 cicles, tal i com es pot veure en la figura 2.1.

Instrucció	Nº de cicles en executar-se
PUSH PSW	12
LDAX D	7
ADD M	7
STAX B	7
INX H	6
INX D	6
INX B	6
POP PSW	10
RET 10	10

Figura 2.1: Taula d'instruccions de la subrutina 'suma'

1.2 Tasca 1: Dibuix de la memòria

A la figura 2.2 es troba el dibuix del mapa de memòria de dades amb les direccions i el contingut. També s'indiquen les instruccions que modifiquen les dades de la memòria i en cadascuna d'elles, les modificacions que es produeixen. A part es situa la memòria de programa i dins d'ella, es localitza el sub-bloc que pertany a la subrutina 'suma'.

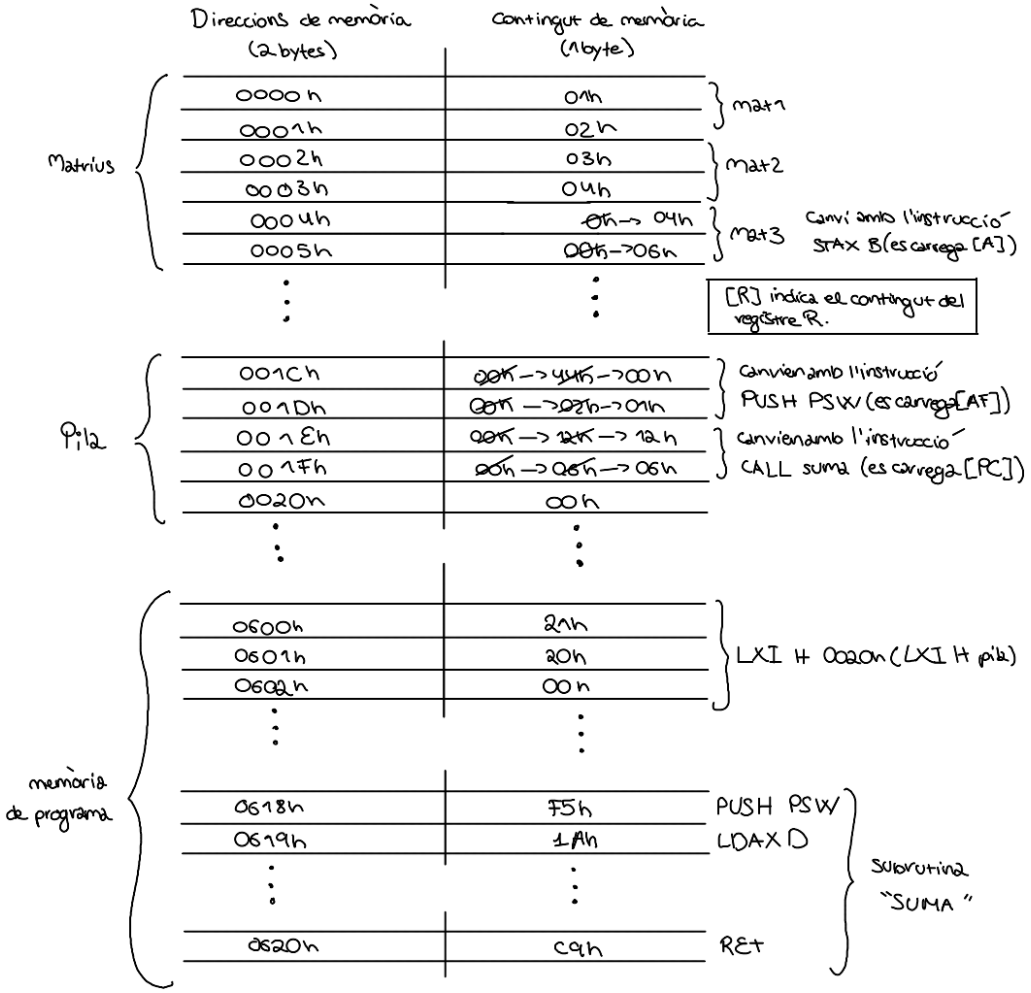

Figura 2.2: Mapa de memòria de dades

1.3 Tasca 2: Funcionament de la pila

La pila comença a la posició de memòria 0020h. Això ho podem observar al simulador al executar el programa. En l'execució es pot observar que la pila creix cap avall, llavors comença a escriure a la posició 001Fh, però creix des de la 0020h.

A la figura 2.3 es mostra quines instruccions modifiquen la pila i com la modifiquen.

Instrucció	Descripció	Canvi de la pila
PUSH PSW	Afegeix PSW a la pila	Augmenta en 2 *bytes*
POP PSW	Guarda a PSW el top de la pila (aquell valor al qual apunta SP, seguint l'estructura LIFO)	Decrementa en 2 *bytes*
CALL suma	Afegeix PC a la pila i salta a la posició etiquetada per suma, per iniciar l'execució de la subrutina	Augmenta en 2 *bytes*
RET	Guarda a PC el top de la pila	Decrementa en 2 *bytes*

Figura 2.3: Taula d'instruccions que modifiquen la pila

2 PART II

En aquesta segona part començarem a analitzar els conceptes d'entrada i sortida. Per començar ens donen el següent codi.

```
1  .data 100h
2      pila:
3  .org 24h
4       JMP ports
5  .org 500h
6      LXI H, pila
7      SPHL
8      CALL ports
9      NOP
10     HLT
11 ports:
12     PUSH PSW
13     IN 04h
14     ANI 00000001
15     OUT 05h
16     POP PS
```

2.1 Tasca 3

Què fa la subrutina 'ports'? Per això, introduïu dades amb els interruptors o amb el teclat; observeu en un port de sortida el resultat de la subrutina.
Aquest programa utilitza dos ports per a fer lectura i escriptura. En la subrutina 'ports' en primer lloc guarda la paraula d'estat del programa a la pila, per no perdre-la; carrega a l'acumulador l'entrada del port 04h, realitza l'operació lògica AND amb el valor de l'acumulador (aquest mateix) i l'immediat 1 i finalment escriu el resultat al port 05h, el qual pertany al *display de 7 segments*. En el *display* el port 05h tal i com està configurat, pertany al sisé nombre, per tant si

s'introdueix el valor 1 per teclat (31h), el resultat de l'operació AND és 31h AND 1 = 1 i llavors s'encén el primer element del sisè nombre del *display*, d'igual forma passa amb qualsevol element introduït, ja que l'operació AND sempre donarà el mateix resultat i per tant, el *display* no variarà si no s'introdueix un 0 (amb això s'apagarà).

Figura 2.4: *Display* al pulsar la tecla 1

3 PART III

En l'última part de la pràctica està destinada a acabar d'entendre la comunicació entre el microprocessador i els perifèrics d'entrada i sortida. Llavors se'ns proposa realitzar un programa que representi al *display* de 7 segments del simulador els nombres del 0 al 5 entrats per teclat. A més, si l'usuari prem la tecla c es produirà un *CLEAR* del *display*.

A continuació es mostra el codi realitzat, comentat i posteriorment s'explicaran diversos exemples d'execució.

```
.define
    ;nombres com a sortida per tal de mostrar correctament els nombres
    zero 77h
    one 44h
    two 3Eh
    three 6Eh
    four 4Dh
    five 6Bh
    .org 00h
    pila:
    ; iniciem la pila al final de la memoria de dades
    LXI H, pila
    SPHL
    loop:
    JMP loop
.org 24h
    CALL ports
    RET
ports:
    IN 00h ; polsem la tecla desitjada i carreguem el valor a l acumulador
    ; processem la sortida pel display
    CPI 30h ; veiem si l entrada es 0 (input tecla 0: 30h, tecla 1: 31h...)
    JNZ nozero ; si el flag de zero no esta activat, no hem polsat el zero
    ;cas entrem 0
    MVI A,zero ; inserim a l acumulador el valor que volem a la sortida per mostrar 0
    OUT 00h ; mostra el numero 0 al display
    RET
nozero:
    CPI 31h ; comprovem si l entrada es 1
```

```
30      JNZ noone ; si el flag de zero no esta activat, no hem polsat 1
31      ; cas entrem 1, mateixa estructura pels altres numeros
32      MVI A,one
33      OUT 00h
34      RET
35  noone:
36      CPI 32h
37      JNZ notwo
38      MVI A,two
39      OUT 00h
40      RET
41  notwo:
42      CPI 33h
43      JNZ nothree
44      MVI A,three
45      OUT 00h
46      RET
47  nothree:
48      CPI 34h
49      JNZ nofour
50      MVI A,four
51      OUT 00h
52      RET
53  nofour:
54      CPI 35h
55      JNZ nofive
56      MVI A,five
57      OUT 00h
58      RET
59  nofive:
60      ; l entrada al polsar c es 63h
61      SUI 63h ; comprovem si la lletra introduida es c
62      JNZ ret ; si no hem entrat c, retornem directament
63      OUT 00h ; netegem la sortida si hem entrat c
64  ret:
65      RET
```

En primer lloc definim 6 constants, una per a cada nombre del 0 al 5: "zero", "one", etc. Aquestes constants s'han precalculat prèviament per tal de saber quin valor s'ha d'afegir al port de sortida per tal que es mostri en el *display* el nombre introduït.

Al *display* de 7 segments cada bit dels valors (usem nombres d'un byte) representa la il·luminació d'un segment en concret. L'esquema és el que es mostra a la figura 2.5.

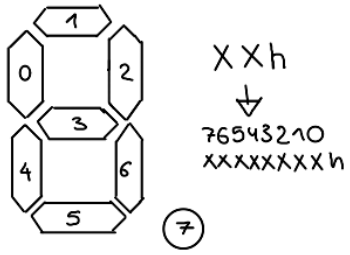

Figura 2.5: Esquema del *display* de 7 segments

Segons l'esquema mostrat, un zero en l'output del *display* hauria de ser una entrada de 01110111b = 77h, un u una entrada de 01000100b = 44h, etc. D'aquesta manera aconseguim establir certes constants per tal de mostrat els nombres al *display*.

A continuació s'analitzarà el codi en profunditat. Primerament s'inicialitza el SP (*Stack pointer*, la posició inicial de la pila) i es farà que la pila creixi des del final del programa. A continuació entrem a un bucle infinit que s'aturarà quan salti una interrupció, és a dir, quan l'usuari escolli un caràcter al teclat. Per això és important tenir activades al simulador les interrupcions TRAP al polsar en el teclat. La interrupció fa saltar a la posició 24h guardant el PC a la pila per tal de recuperar-ho després.

La subrutina "ports" es crida a cada interrupció que es realitzi pel fet de polsar una tecla de les que indica l'enunciat. Per tant, en aquesta subrutina carreguem a l'acumulador el valor ASCII de la tecla escollida per l'usuari (entrada del port 00h). A continuació comença la gestió per seleccionar el número a mostrar: es compara el contingut de l'acumulador, on s'ha carregat el valor ASCII de la tecla polsada, si coincideix amb el codi ASCII de cadascun dels valors del 0 al 5 mitjançant una estructura d'instruccions condicionals. En cas de que es compleixi algun, carreguem a l'acumulador la constant que s'havia calculat que mostra el nombre al segment i s'introdueix com a sortida al port 00h.

A la figura 2.5 es mostra un exemple, on s'ha polsat l'1 al teclat i l'entrada llavors és el codi ASCII d'aquest mateix nombre, és a dir, 31h, i per tant la sortida al *display* de 7 segments és la constant definida prèviament 44h, que fa que aparegui l'1 al segment.

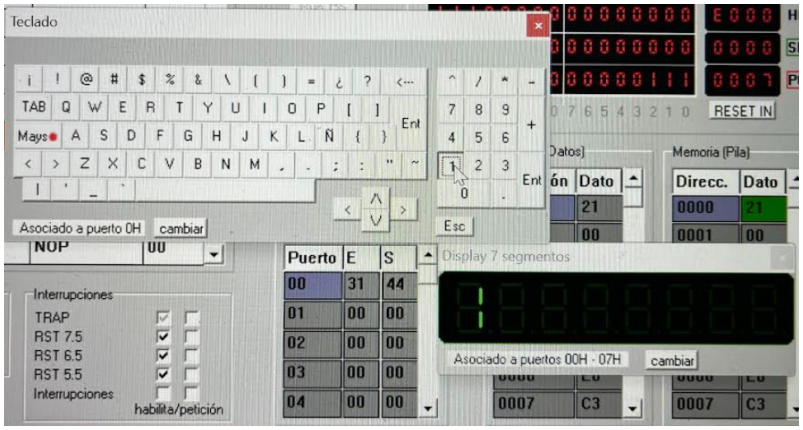

Figura 2.6: *Display* al pulsar el número

En cas de no haber entrat cap número dels de contempla el programa, llavors es comprova si s'ha introduït el caràcter 'c' restant 63h (ASCII de 'c') al contingut de l'acumulador, llavors si dona 0 efectivament s'haurà introduït aquesta tecla i per tant es neteja el *display*. De no ser així, no s'ha seleccionat cap entrada que contemplava el programa i es torna al bucle infinit.

3 Conclusions

En la segona pràctica utilitzant el simulador i8085 s'ha vist en el primer exercici el funcionament de la pila i s'ha acabat d'assimilar els conceptes presentats a la primera pràctica d'aquest simulador, com les instruccions que guarda el PC a la pila, quines instruccions modifiquen el contingut d'aquesta, etc.

En el segon i el tercer exercici s'ha introduït el concepte de perifèrics d'entrada i sortida. Aque-

stes dues últimes parts de la pràctica han tingut un grau d'interés personal més interessant, degut a que s'ha vist com relacionar els ports d'entrada com el teclat amb el microprocessador, així com aquest mateix amb els ports de sortida. L'última part hem possat en pràctica l'introducció d'aquests conceptes utilitzant per primer cop el *display* del simulador.

Com a conclusió final, ha sigut una pràctica interessant que ha introduït conceptes molt importants com són la pila o la entrada i sortida de l'ordinador per tal de comunicar perifèrics.