Pràctica 5: Introducció al simulador i8085

Introducció / Objectius

A partir d'aquesta pràctica comencen les pràctiques que es duran a terme sobre el processador i8085. Treballarem amb un simulador diferent al de RISC-V i, per tant, tindrem una estratègia prou distinta en quant a programació. Amb aquests exercicis, pretenem començar a veure quines diferències hi ha entre els dos simuladors, les desavantatges i avantatges que té cadascun, així com començar a veure les principals comandes del i8085. A continuació un resum dels objectius:

- Veure algunes comandes del i8085.
- Entendre procediments bàsics de programació en assemblador.
- Entendre com funcionen els punters de memòria (les etiquetes utilitzades).
- Fer uns programes d'exemple per iniciar-se al i8085.

Exercici I: Modes d'Adreçament del i8085

Escriviu dos programes diferents per sumar dues matrius de 5x1 i desar el resultat en una tercera, podeu fer servir adreçament indirecte, per parella de registres o per registres. Penseu en dues hipòtesi, primera, les matrius d'entrada s'han de conservar, és a dir el resultat s'emmagatzema sobre mat3 i segona, desem el resultat sobre una de les dues matrius d'entrada, mat3 és irrellevant. El programa hauria d'utilitzar un comptador, instruccions d'increment, comparació i salt condicional. Feu la suma de números hexadecimals i sense tenir en compte números que pugui donar overflow.

Programa 1: Suma dues matrius (***mat1***, ***mat2***) i guarda el resultat sobre ***mat2***.

A continuació adjunto el codi del programa 1, explicant cada pas amb comentaris i, a més, indicant què triga cada instrucció (cicles de rellotge) segons els apunts que tenim al conjunt d'instruccions del i8085.

```
;DADES

.data 50h       ;directiva per guardar dades

        mat1: db 1, 2, 3, 4, 5          ;l'etiqueta mat1 té el valor 50h

        mat2:db 6, 7, 8, 9, 0           ;l'etiqueta mat2 té el valor 55h

;PROGRAMA PRINCIPAL

.org 100h       ;directiva per senyalar on començarà el programa.

        LXI d, mat1             ;Utilitzem els registres DE per a carregar la posició
```

```
                        ;de memòria indicada per mat1.
                        ;Adreçament immediat (no accedeix a memòria, només
                        ;carrega el valor de la label mat1(que val 50h) a d).
                        ;[D,E] = 0050h
                        ;Triga 10 cicles.
        LXI h, mat2     ;Utilitzem els registres HL per a carregar la posició
                        ;de memòria indicada per mat2.
                        ;Adreçament immediat.
                        ;[H,L] = 0055h
                        ;Triga 10 cicles.
loop:
        LDAX d          ;Carreguem a l'acumulador el contingut de Mem[DE].
                        ;Adreçament indirecte de memòria (utilitzem el contingut
                        ;de DE per obtenir la posició de memòria) i implícit per
                        ;l'acumulador.
                        ;En la primera iteració carrega un 1.
                        ;7 cicles.
        ADD M           ;Acc <= Acc + Mem[M]
                        ;Adreçament implícit per l'acumulador i, donat que M és el
                        ;contingut dels registres H,L no tenim pas intermedi, per
                        ;tant és directe a memòria.
                        ;En la primera iteració teníem un 1 a l'acumulador, per tant
                        ;1 + 6 = 7.
                        ;Tarda 7 cicles.
        MOV M, a        ;Mem[M] <= Acc
                        ;Adreçament directe de l'acumulador i directe de
                        ;memoria.
```

```
                    ;En la primera iteració, tindrem un 7 a l'acumulador.
                    ;Sobreescriu el valor que teníem a M (un 6) i hi guarda un
                    ;7.

                    ;Tarda 7 cicles.
INX d               ;Incrementem en +1 el contingut de DE

                    ;Adreçament directe

                    ;En la primera iteració, [D, E] = 0050h. Li sumem 1, [D, E] =

                    ;0051h.

                    ;Tarda 6 cicles.
INX h               ;Incrementem en +1 el contingut de HL

                    ;També existeix la directiva "inr 'reg'", només

                    ;actua sobre un registre

                    ;En la primera iteració [H, L] = 0056h. Li sumem 1, [H, L] =
                    ;0057h
MVI a, mat2         ;Carreguem immediat 'mat2' a l'acumulador.

                    ;Adreçament directe de l'acumulador i immediat del mat2.

                    ;Posem a l'acumulador el valor de mat2, és a dir, 55h.

                    ;Tarda 7 cicles.
cmp e               ;Comparem el valor de l'acumulador amb el del registre e

                    ;Aquí mira si arriba al final. A la primera iteració, el registre
                    ;e tindrà el valor 51h.

                    ;Hem d'acabar el bucle quan el registre e tingui el valor
                    ;55h, perquè això vol dir que començaríem a agafar valors
                    ;de la segona matriu. Per tant, quan aquesta comparació
                    ;ens doni 0.

                    ;Tarda 4 cicles.
jnz loop            ;Salta a loop si no és zero.

                    ;Adreçament immediat.
```

```
                    ;Tarda entre 7 i 10 cicles, segons si segueix el programa
                    ;seqüencialment o ha d'anar-se'n altre cop a loop.
```

;FI DEL PROGRAMA

```
    HLT             ;Sentència per indicar el final del programa.
```

Com podem veure, en aquest programa la instrucció que més tarda és la de ***LXI Reg***, ***immediat***. En aquesta, el que fem és carregar un nombre immediat al registre indicat. El que fem en aquest programa és bàsicament:

- Carreguem les posicions de memòria en els registres D, E per mat1; H, L per mat2.
- Carreguem el que hi hagi a la posició de memòria de D, E a l'acumulador (és a dir, el primer nombre de mat1).
- Sumem el contingut de l'acumulador amb allò que hi hagi a la posició de memòria del contingut de H, L. Com que H, L tenia la posició de memòria on hi ha el primer nombre de mat2, sumem el primer nombre de mat1 + el primer nombre de mat2.
- Copiem el resultat de l'operació anterior a la direcció de memòria que té H, L, és a dir, sobre el primer nombre de mat2.
- Incrementem els valors dels registres D, E i H, L per a poder operar amb els següents dos nombres de la matriu.
- Carreguem a l'acumulador l'immediat mat2 (la posició de memòria on es troba mat2) i comparem si aquest valor és igual al que trobem al registre e. Recordem que al registre E teníem el valor de la posició de memòria de l'últim nombre que havíem sumat + 1. Per tant, si aquests dos valors coincidissin, tindríem que ja hem arribat al final de tots els nombres que hem de sumar a mat2.
- Si el valor anterior és 0 podrem acabar el programa. Si no, saltarem a ***loop*** i tornarem a fer tot el procés, amb el següent nombre que correspongui dels vectors.

Programa 2: En següent programa es basa en el mateix, per tant, les explicacions corresponents poden ser trobades en el programa anterior amb tot detall. Aquests dos programes només difereixen en una cosa: ara no sobreescriurem els valors del segon vector, sinó que els guardarem a una altra posició de memòria. Per això, haurem d'usar les tres parelles de registres de les que disposem. El programa és el següent:

;DADES

```
.data 50h
```

```
        mat1: db 1, 2, 3, 4, 5
        mat2: db 6, 7, 8, 9, 0
        mat3: db 0, 0, 0, 0, 0
;PROGRAMA PRINCIPAL
.org 100h
        LXI d, mat1     ;Utilitzem els registres DE per a carregar la posició
                        ;de memòria indicada per mat1
                        ;Tindrem [D, E] = 0050h
                        ;Tarda 10 cicles.
        LXI h, mat2     ;Utilitzem els registres HL per a carregar la posició
                        ;de memòria indicada per mat2
                        ;Tindrem [H, L] = 0055h
                        ;Tarda 10 cicles.
        LXI b, mat3     ;Utilitzem els registres BC per a carregar la posició
                        ;de memòria indicada per mat3
                        ;Tindrem [B, C] = 005Ah
                        ;Tarda 10 cicles.
loop:
        LDAX d          ;Acc <= Mem[DE]
                        ;Per exemple, en la primera iteració tindrem:
                        ;Acc <= Mem[0050h] = 1. [Acc] = 1
                        ;Tarda 7 cicles.
        ADD M           ;Acc <= Acc + Mem[M]
                        ;Per exemple, en la primera iteració tindrem:
                        ;Acc <= [Acc] = 1 + Mem[0055h] = 6. [Acc] = 7
```

```
                    ;Tarda 7 cicles.
STAX b              ;Mem[BC] <= Acc
                    ;Per exemple, en la primera iteració ens dóna:
                    ;Mem[005Ah] = 7.
                    ;Tarda 7 cicles.
INR c               ;Incrementem en +1 el contingut de C
                    ;En el cas de la primera iteració, serà:
                    ; [C] = 5Ah + 1 = 5Bh
                    ;Tarda 10 cicles.
INR e               ;Incrementem en +1 el contingut de E
                    ;En el cas de la primera iteració:
                    ;[E] = 50h + 1 = 51h
                    ;Tarda 10 cicles.
INR l               ;Incrementem en +1 el contingut de L
                    ;En el cas de la primera iteració:
                    ;[L] = 55h + 1 = 56h.
                    ;Tarda 10 cicles.
MVI a, mat2         ;Carreguem immediat 'mat2' a l'acumulador.
                    ;L'acumulador tindrà [Acc] = 55h
                    ;7 cicles.
cmp e               ;comparem el valor de l'acumulador amb el del
                    ;registre e. Mirem si hem incrementat suficients
                    cops E de manera que ja haguem sumat tots els
                    nombres del vector.
                    ;Tarda 4 cicles.
JNZ loop            ;salta a loop si no és zero
```

```
;Tarda 7 o 10 cicles.
```

;FI DEL PROGRAMA

```
HLT
```

Altre cop, les operacions que més triguen són les de ***LXI*** i també les de ***INR***; aquesta última serveix per incrementar en 1 el valor del contingut d'una parella de registres, no només d'un registre.

Pregunta 1:

En què simplificaria molt el codi del programa un dels modes d'adreçament del simulador Ripes?

Al fer ús de la instrucció ***load adress (la)*** pertinent del simulador ***Ripes***, simplificaria bastant el codi ja que aquesta necessita un menor nombre de cicles.

Calculeu les mides del codi del vostre programa i el nombre de cicles per a la seva execució.

A continuació el càlcul de quant ocupa el nostre codi del programa 1, és a dir, quantes posicions de memòria principal ocupa el nostre codi. El cos del programa comença a la posició de memòria 100h i acaba a la 111h, en total, 28 posicions.

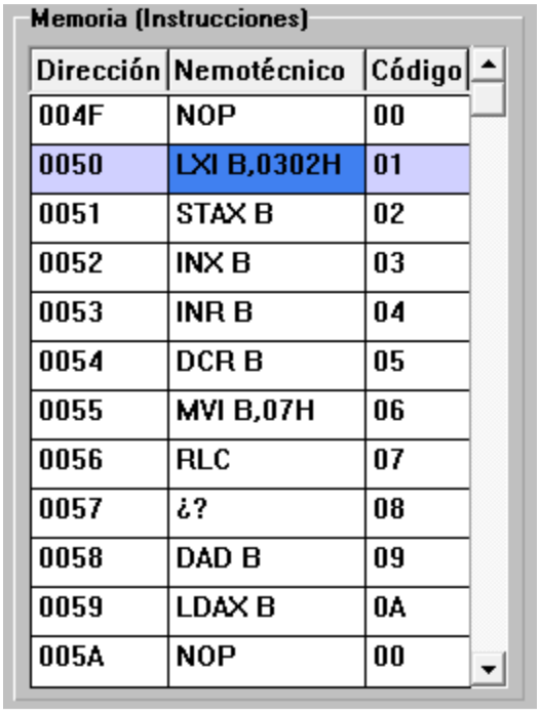
Memoria (Instrucciones)

Dirección	Nemotécnico	Código
0100	LXI D,0050H	11
0101		50
0102		00
0103	LXI H,0055H	21
0104		55
0105		00
0106	LDAX D	1A
0107	ADD M	86
0108	MOV M,A	77
0109	INX D	13
010A	INX H	23
010B	MVI A,55H	3E

Memoria (Instrucciones)

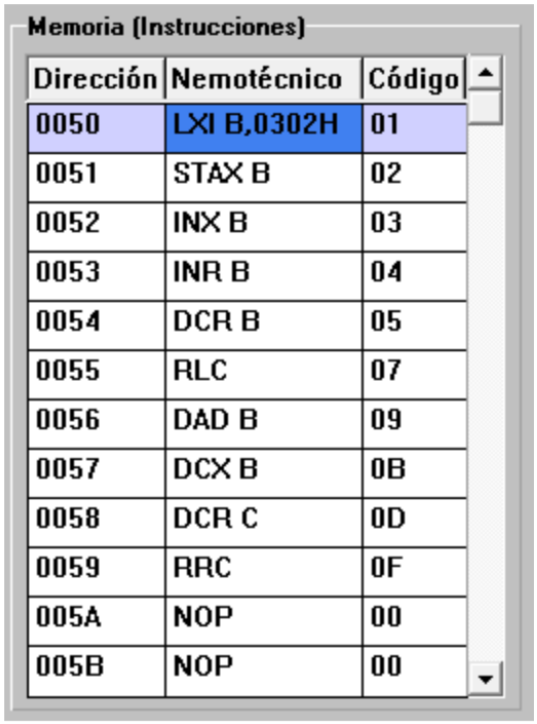
Dirección	Nemotécnico	Código
010B	MVI A,55H	3E
010C		55
010D	CMP E	BB
010E	JNZ 0106H	C2
010F		06
0110		01
0111	HLT	76
0112	NOP	00
0113	NOP	00
0114	NOP	00
0115	NOP	00
0116	NOP	00

Però a més, tenint en compte les posicions de memòria que utilitzem per guardar dades, van des de la 50h fins a la 59h, en total 10 posicions.



Dirección	Nemotécnico	Código
004F	NOP	00
0050	LXI B,0302H	01
0051	STAX B	02
0052	INX B	03
0053	INR B	04
0054	DCR B	05
0055	MVI B,07H	06
0056	RLC	07
0057	¿?	08
0058	DAD B	09
0059	LDAX B	0A
005A	NOP	00

Després d'haver executat el programa, els codis canvien:



Dirección	Nemotécnico	Código
0050	LXI B,0302H	01
0051	STAX B	02
0052	INX B	03
0053	INR B	04
0054	DCR B	05
0055	RLC	07
0056	DAD B	09
0057	DCX B	0B
0058	DCR C	0D
0059	RRC	0F
005A	NOP	00
005B	NOP	00

Hem de calcular els cicles que triga. Fora del **loop** tenim 20 cicles (del ***LXI***). El **loop** triga 54 cicles si saltem a **loop** i 51 cicles si no saltem. Tenim 5 iteracions, 4 amb salt i una sense, per tant 54·4 + 51 = 267. Més els 20 cicles de fora del **loop** fan un total de 287 cicles de rellotge. Si comptem també el **hlt** (4 cicles), fan 291 cicles.

Calculem ara quant ocupa en memòria el problema 2. La part de codi de programa en sí va des de la posició 100h fins a la posició 115h, ocupant en total 22 posicions de memòria.

Memoria (Instrucciones)

Dirección	Nemotécnico	Código
0100	LXI D,0050H	11
0101		50
0102		00
0103	LXI H,0055H	21
0104		55
0105		00
0106	LXI B,005AH	01
0107		5A
0108		00
0109	LDAX D	1A
010A	ADD M	86
010B	STAX B	02

Memoria (Instrucciones)

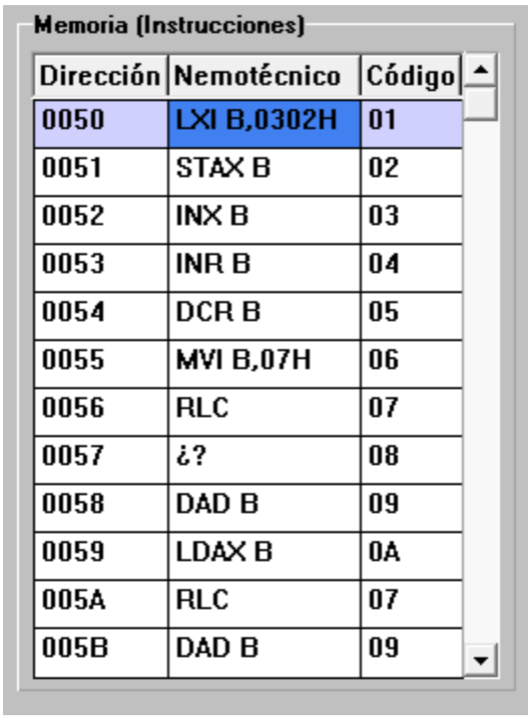
Dirección	Nemotécnico	Código
010B	STAX B	02
010C	INR C	0C
010D	INR E	1C
010E	INR L	2C
010F	MVI A,55H	3E
0110		55
0111	CMP E	BB
0112	JNZ 0109H	C2
0113		09
0114		01
0115	HLT	76
0116	NOP	00

A més, tenint en compte les posicions que usem per guardar dades, van des de la 50h fins a la 59h si NO executem (perquè la tercera matriu té zeros) i un cop executat el programa ocupen des de la posició 50h fins a la 5Eh, en total 15 posicions de memòria.

Memoria (Instrucciones)

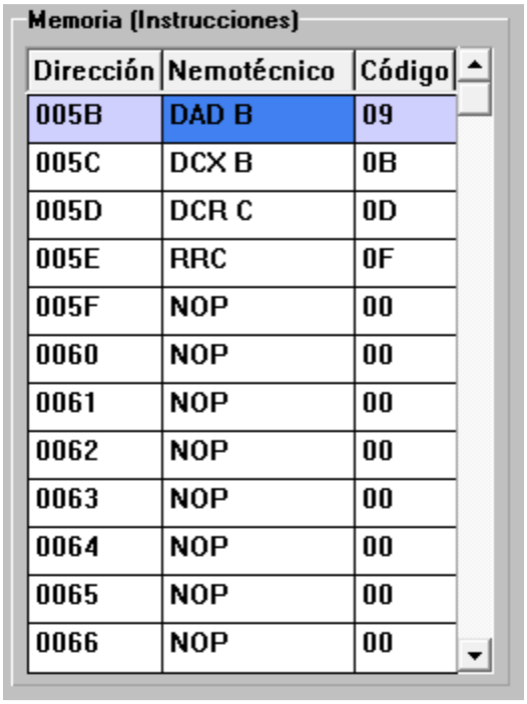
Dirección	Nemotécnico	Código
0050	LXI B,0302H	01
0051	STAX B	02
0052	INX B	03
0053	INR B	04
0054	DCR B	05
0055	MVI B,07H	06
0056	RLC	07
0057	¿?	08
0058	DAD B	09
0059	LDAX B	0A
005A	NOP	00
005B	NOP	00

Havent executat el programa:



Memoria (Instrucciones)

Dirección	Nemotécnico	Código
0050	LXI B,0302H	01
0051	STAX B	02
0052	INX B	03
0053	INR B	04
0054	DCR B	05
0055	MVI B,07H	06
0056	RLC	07
0057	¿?	08
0058	DAD B	09
0059	LDAX B	0A
005A	RLC	07
005B	DAD B	09



Memoria (Instrucciones)

Dirección	Nemotécnico	Código
005B	DAD B	09
005C	DCX B	0B
005D	DCR C	0D
005E	RRC	0F
005F	NOP	00
0060	NOP	00
0061	NOP	00
0062	NOP	00
0063	NOP	00
0064	NOP	00
0065	NOP	00
0066	NOP	00

Calculem els cicles de rellotge que triga. Tenim 3 operacions ***LXI*** fora del **loop**, això són 30 cicles. El **loop** tarda 72 cicles quan saltem i 69 quan no saltem. Tenim 5 iteracions, 4 amb salt i una sense salt, per tant, 4·72 + 69 = 357 cicles de rellotge. Sumem els 30 que havíem deixat fora del **loop** i fan un total de 387 cicles de rellotge. Si a més tenim en compte el **hlt** (4 cicles) tindrem al final 391 cicles de rellotge, 100 cicles més que al programa anterior.

Pregunta 2:

Quants cicles de rellotge triga en executar-se una instrucció aritmètic-lògica qualsevol? Feu servir el fitxer adjunt on especifica el ISA del 8085. Indica quina és la mida mitjana de les teves instruccions. Calcula els cicles per instrucció mitjà per aquests codis.

Explicarem per exemple què triga una instrucció **ADD**. En trobem de dos tipus, ***ADD Reg*** i ***ADD M***. El primer, efectua la suma del registre especificat i el que tinguem a l'acumulador i ho deixa a l'acumulador:

[Acc] <= [Acc]+[Reg]

Aquesta triga 4 cicles de rellotge. En canvi, l'operació ***ADD M***, que també realitza una suma, primer ha d'identificar el contingut dels registres ***H,L*** i anar a la posició de memòria que indica tal contingut. Suma això amb el que hi havia a l'acumulador i ho deixa a l'acumulador.

[Acc] <= [Acc]+[Mem[H,L]]

Aquesta tarda 7 cicles de rellotge. L'increment en cicles de rellotge és degut a que aquesta última operació ha d'accedir a memòria. No sols això, primer ha d'accedir al contingut dels registres ***H,L*** per després accedir al contingut de memòria indicat i realitzar la operació.

El codi del programa mostrat pel professor utilitza les següents instruccions:

- **LXI**: triga 10 cicles en executar-se i ocupa 3 bytes.
- **MVI**: triga 10 cicles en executar-se i ocupa 2 bytes.
- **MOV**: triga 4 cicles en executar-se i ocupa 1 byte.
- **LDAX**: triga 7 cicles en executar-se i ocupa 1 byte.
- **ADD**: triga 4 cicles en executar-se i ocupa 1 byte.
- **STAX**: triga 7 cicles en executar-se i ocupa 1 byte.
- **INX**: triga 6 cicles en executar-se i ocupa 1 byte.
- **DCR**: triga 4 cicles en executar-se i ocupa 1 byte.
- **JNZ**: triga 7 cicles en executar-se quan no realitza el salt i 10 cicles quan realitza el salt, i ocupa 3 bytes.
- **HLT**: triga 4 cicles en executar-se i ocupa 1 byte.

De manera que podem dir que la mida mitjana del nostre codi és de 1,5 bytes i els cicles per instrucció mitja que triguen és de 6,6 cicles.

Pregunta/Tasca 3:

Pugeu el vostre codi i marqueu quina és la instrucció del vostre programa que triga més cicles en executar-se.

Les instruccions del nostre codi (adjuntat anteriorment) que triguen més en executar-se són **LXI** i **MVI**, trigant les dues 10 cicles en executar-se. També cal mencionar que la instrucció **JNZ** triga també 10 cicles quan realitza el salt.

Pregunta/Tasca 4:

Traduïu el codi per fer-lo servir amb el simulador Ripes. Quants cicles triga en executar- se? Compareu els resultats (mida de codi, accessos a memòria i cicles promig per instrucció) amb els valors obtinguts per l'i8085.

```
.data
    mat1: .word 1,2,3,4,5
    mat2: .word 6,7,8,9,0
    mat3: .word 0,0,0,0,0
```

```
        count: .word 5
.text
main:
        lw a5, count
        la a0, mat1
        la a1, mat2
        la a2, mat3
loop:
        lw a3, 0(a0)
        lw a4, 0(a1)
        add a3, a3, a4
        sw a3, 0(a2)
        addi a0, a0, 4
        addi a1, a1, 4
        addi a2, a2, 4
        addi a5, a5, -1
        bgt a5, zero, loop
end:
        nop
```

El mateix programa en el simulador **Ripes** triga 71 cicles en executar-se, on s'executen 54 instruccions, de manera que la mida del nostre codi és de 216 bytes.

Exercici II: Subrutines

Fent ús del programa anterior, realitzeu una subrutina que codifiqui una zona de memòria. La zona de memòria s'indicarà posant al registre doble **HL** l'adreça de començament de la zona de dades a codificar. Aquestes dades es consideren com els paràmetres d'entrada de la subrutina. La codificació es farà mitjançant una **XOR** entre cada byte de la zona de dades i la clau. Els valors dels registres no han de quedar afectats per la crida a la subrutina.

Feu un programa que faci us d'aquesta subrutina per provar-la amb diferents combinacions de valors de la clau i les dades per codificar.

En aquest exercici, agafarem el programa anterior i passarem una 'màscara' per les dades. Bàsicament, un cop acabat el programa i realitzades les sumes, passarem una **XOR** per totes les dades que tenim guardades per tal de deixar-les 'codificades'. Per fer això, usarem una subrutina que cridarem gairebé al final del programa.

Pregunta 4:

Quina instrucció fem servir en tots dos casos per assignar la posició inicial al registre SP?

La instrucció **SPHL**.

Pregunta 5:

Quina es la instrucció utilitzada per guardar el PC en la pila quan treballem amb subrutines? I per recuperar de nou el valor del PC?

La instrucció **CALL** és la que guarda la posició del *program counter* per després tornar-hi. Per recuperar aquesta posició utilitzem la instrucció **RET** (retorn) que no trobem en aquest programa, ja que degut a l'optimització, els procediments que són corrents amb l'ús de la pila no els hem d'utilitzar.

Tasca 2:

Pugeu el codi creat amb les subrutines:

```
;DADES

.data 50h

    mat1: db 1, 2, 3, 4, 5

    mat2: db 6, 7, 8, 9, 0
```

```
        mat3: 0, 0, 0, 0, 0, 0
;PROGRAMA PRINCIPAL
.org 100h
        LXI d, mat1         ;Els registres DE carreguen la posició
                            ;de memòria mat1
        LXI h, mat2         ;Els registres HL carreguen la posició
                            ;de memòria mat2
        LXI b, mat3         ;Els registres BC carreguen la posició
                            ;de memòria indicada per mat3
loop:
        LDAX d              ;Acc <= Mem[DE]
        ADD M               ;Acc <= Acc + Mem[M]
        STAX b              ;Mem[BC] <= ACC
        INR c               ;Incrementem en +1 el contingut de C
        INR e               ;Incrementem en +1 el contingut de E
        INR l               ;Incrementem en +1 el contingut de L
        MVI a, mat2         ;Carreguem immediat 'mat2' a l'acumulador.
        cmp e               ;Comparem el valor de l'acc. amb el registre e.
        JNZ loop            ;salta a loop si no és zero
        call codifica       ;Subrutina per a codificar la memòria
;FI DE PROGRAMA
acaba:
        HLT
;SUBRUTINA DE CODIFICACIÓ
codifica:
```

```
        LXI h, mat1     ;Posició inicial de memòria
        MVI b, FFH      ;Carreguem la màscara a B, en aquest cas FFh
        MVI c, 00h      ;Inicialitzem a 0 el registre C.
        MVI d, 15d      ;Carreguem les posicions totals de memòria a
                        ;codificar a D.
xor:
        mov a, M        ;Carreguem el valor de la posició de memòria a codificar
        XRA b           ;A <= B XOR A (Operem la XOR de B amb A)
        MOV M, a        ;Guardem l'element a codificat a la mateixa posició de
                        ;memòria
        INR c           ;Incrementem en +1 el valor de C
        INX H           ;Incrementem en +1 el valor de HL
        MOV a, c        ;Carreguem el valor de C a l'acumulador
        cmp d           ;Comparem el valor de l'acumulador amb D.
        JNZ XOR         ;Salta si Z= 0
        JMP acaba       ;Salt incondicional per acabar el programa.
```

En aquest programa, per fer la subrutina, utilitzem un procés similar al que hem usat per sumar les components de la matriu. Guardem els valors de les posicions de memòria als registres **HL**, i les posicions de memòria que haurem d'emmascarar (que en aquest cas són 15) i efectuem un bucle molt semblat als explicats anteriorment. Un cop acabat, saltem al final del programa i acabem.

Conclusions

En aquesta pràctica hem realitzat uns petits programes per practicar i començar a utilitzar el simulador i8085. A continuació un resum del que hem realitzat i après:

- Ens hem familiaritzat amb les instruccions del i8085 i les seves especificacions.
- Hem entès el procediment de programació i em començat a veure l'entorn de programació.
- Hem après el funcionament dels punters de memòria.
- Sabem com mesurar la mida d'una instrucció i els cicles que triga en executar-se.
- Sabem com calcular els cicles i la mida d'un programa.
- Comprendre el funcionament de les subrutines i el seu ús en la codificació dels programes.