OBJECTIUS:

L'objectiu principal d'aquesta pràctica és familiaritzar-me amb un nou simulador, el i8085, així com entendre les subrutines, la pila i el SP.

EXERCICI 1:

PREGUTA 1:

El mode d'adreçament directe (podent escollir els operands i on volem que es guardi l'operació) seria molt útil, ja que ens permetria operar amb molta més facilitat entre els diferents registres que tenim, estalviant-nos haver de passar per acumulador tota l'estona.

El que guarda el resultat en mat3 és més complicat, i el programa ocuparà 25h posicions de memòria (125-100), que, com cada posició de memòria guarda un byte, tindrà una mida total de 37 bytes, i, si comptem el número de cicles utilitzant la guia d'instruccions del simulador, agafant el màxim de cícles per instrucció, ens trobem que el programa tardarà 558 en executar-se. El que ignora mat3 és més senzill, i llavors sol ocupa 14h posicions de memoria, que si ho multipliquem per dos obtenim que ocuparà 20 bytes, i tardarà, amb els mateixos criteris que l'anterior, 342 cícles.

PREGUTA 2:

La gran majoria de les instruccions aritmetico-lògiques tardaran 4 (quan opera sol amb registres) o 7 (quan ha d'accedir en memòria) cícles en executar-se, tot i que hi ha algunes, com ara DCX, que tardaran 6.

Les mides mitjanes de les instruccions que tenim al programa que té en compte mat3 serà de 1.54166666667(37/24), i en el programa més curt 1.42857142857 (20/14). En quant als cícles, en el curt tindrem una mitja de 6.84 i en el més llarg, 5,58.

PREGUNTA 3:

```
.define
      num 5
.data 00h
      mat1: db 1,2,3,4,5
      mat2: db 6,7,8,9,0
      mat3: db 0,0,0,0,0
.org 100h
      LXI H, mat1
      LXI B, mat2
      MVI D, num
      DCR D
      loop:
            MOV E, M
            LDAX B
            ADD E
            STAX B
            INX H
            INX B
            DCR D
            LDAX D
            JP loop
hlt
```

```
.define
      num 5
.data 00h
      mat1: db 1,2,3,4,5
      mat2: db 6,7,8,9,0
      mat3: db 0,0,0,0,0
.org 100h
      LXI H, mat1
      LXI B, mat3
      MVI D, num
      DCR D

      carga:
            MOV A, M
            STAX B
            INX H
            INX B
            DCR D
            JP carga
            LXI H, mat2
            LXI B, mat3
            MVI D, num
            DCR D

      loop:
            MOV E, M
            LDAX B
            ADD E
            STAX B
            INX H
            INX B
            DCR D
            LDAX D
            JP loop
hlt
```

En aquests programes, carreguem els valors de les matrius en registres, i després fem un bucle per sumar i finalment guardar en memòria els resultats. La diferència entre els dos és que en el llarg, és necessari fer un bucle previ on carregarem la informació de la matriu1 a la matriu3, per així poder seguir el mateix procediment que en el programa més curt.

En els dos programes tenim que la instrucció que més cícles tarda en executar-se serà la LXI, que en tardarà 10.

PREGUNTA 4:

```
1  .data
2  mat1:  .byte 1, 2, 3, 4, 5
3  mat2:  .byte 6, 7, 8, 9, 0
4  mat3:  .byte 0, 0 ,0 ,0 ,0
5  num:  .word 5
6
7  .text
8
9  la a0, mat1
10 lw a3, 24(a0)
11
12 add a4, zero, zero
13
14 loop:
15 lb a1, 0(a0)
16 lb a2, 8(a0)
17 add a4, a1, a2
18 sb a4, 16(a0)
19 addi a0, a0, 1
20 addi a3, a3, -1
21 bgtz a3, loop
```

En el simulador de 5 etapes, aquest programa tardarà 56 cícles en executar-se, que són molt menys que els 558 cícles que tardarà en el i8085, fent uns cícles per instrucció promitjos de 1,44, també molt més baix que l'anterior , que és 5,58.

Degut a la naturalesa d'adreçament indirecte en el simulador i8085 es fan molts més accessos a memòria, ja que no guardarem la majoria dels valors amb els que operarem a registres, sinó que els anirem a buscar a memòria de dades. En canvi, al ripes ja tindrem els operands a registres i no farà falta accedir un altre cop a memòria.

En quant a la mida del programa, ocupara 44 bytes (com podem observar si mirem la memòria de programa), que serà major que els 37 que ocupava en el simulador i8085. Això segurament és degut a que, en el simulador ripes, totes les instruccions han d'ocupar 4 bytes de memòria, mentre que al altre pot ocupar menys (ja hem vist que el promig és menor que 4).

EXERCICI 2:

PREGUNTA 4:

La posició inicial del SP es assignada quan cridem la subrutina, amb l'instrucció CALL subrutina. Al executar aquesta instrucció, SP començarà apuntant a l'inici de la cua (en aquest simulador, la posició FFFF), on guardarà el valor de PC abans de començar la subrutina, per així poder retornar al programa principal un cop s'hagi acabat d'executar la subrutina. Just després, SP apuntarà a la posició FFFE, ja que guardar el PC ocuparà dues posicions de memòria.

PREGUNTA 5:

Per guardar el PC, a diferencia dels altres registres en els que és necessari fer un PUSH per guardar-los en la pila, el PC es guarda automanticament al executar la instrucció CALL subrutina. Per recuperar el valor, tampoc cal fer un POP, sinó que al executar la instrucció de fi de subrutina (RET) ja es recupera el seu valor.

TASCA 2:

```
.define
    num 5
    clau 55h
.data 00h
    mat1: db 1,2,3,4,5
    mat2: db 6,7,8,9,0
    mat3: db 0,0,0,0,0
.org 100h
    LXI H, mat1
    LXI B, mat3
    MVI D, num
    DCR D

    carga:
        MOV A, M
        STAX B
        INX H
        INX B
        DCR D
        LDAX D
        JP carga
        LXI H, mat2
        LXI B, mat3
        MVI D, num
        DCR D

    loop:
        MOV E, M
        LDAX B
        ADD E
        STAX B
        INX H
        INX B
        DCR D
        LDAX D
        JP loop
    call sub_codificadora
hlt

.org 150h
sub_codificadora:
      PUSH H
      PUSH B
      PUSH D
      PUSH PSW
      LXI H, mat3
      MVI D, num
      DCR D
codifica:
      MOV A, M
      XRI clau
      MOV M, A
      INX H
      DCR D
      LDAX D
      JP codifica
      POP PSW
      POP D
      POP B
      POP H
RET
```

Quan acabem d'executar el codi que hem fet al apartat 1, cridem la subrutina amb sub_codificadora. Per tal de no modificar el valor dels registres, primer fem els push necessaris per guardar a la pila els valors dels registres. Després, carreguem en els registres totes les dades que necessitarem per fer la codificació, com ara la posició de memòria a codificar (LXI H, mat3) i el número de posicions de memòria a codificar (MVI D, num).

Un cop fet els passos inicials, entrem al bucle on codificarem. Primer, carreguem a acumulador el valor de la posició de memòria guardat al registre HL (MOV A, M), i després fem contra acumulador el XOR amb la clau. Ara, guardem en memòria el resultat obtingut. Incrementem el valor de HL, decrementem el valor de D, carreguem el valor de D a l'acumulador i, utilitzant la instrucció JP comprovem que el valor de l'acumulador sigui positiu o 0. Iterem d'aquesta forma fins que D sigui negatiu.

Al sortir del bucle, recuperem tots els valors que havíem guardat en pila, utilitzant les instruccions POP.

CONCLUSIONS:

El simulador i8085 és molt diferent al que utilitzavem anteriorment (Ripes). En aquest disposem de molts menys registres, les operacions sempre es fan contra acumulador i els modes d'adreçament acostumen a ser indirectes, en comparació amb el ripes, que normalment eren directes. Les subrutines són una eina molt útil per a realitzar operacions sense modificar registres, i he entès el seu funcionament junt amb el del Stack Pointer i la pila, que fins aquest moment no haviem vist funcionar a la pràctica.