Pràctica 7

Introducció / Objectius

En aquesta pràctica farem un programa que actuï com una calculadora amb quatre operacions bàsiques: suma (+), resta (-), AND (A) i OR (O). Tot i que la pràctica se'ns planteja per fer les operacions per separat i després ajuntar-les en un únic programa, he optat per donar-ho tot ja junt en una única aplicació. A continuació deixo un seguit dels objectius de la pràctica:

- Repassar els coneixements de teoria sobre el simulador i8085.
- Utilitzar els coneixements adquirits sobre interrupcions per a tractar-les adequadament.
- Programar una calculadora amb quatre operacions.
- Portar el control dels canals d'entrada i sortida.

Funcionament del codi

A continuació el codi explícit del programa, comentat pas per pas per tal de que sigui més fàcil la seva comprensió. El funcionament del codi és el següent:

- El codi relacionat amb la instrucció TRAP serveix per anar guardant a memòria allò que ens introdueixen per teclat. Només saltarem a una altre part del codi si ens introdueixen el signe "0" (3Dh), ja que serà el nostre indicador de que se'ns ha introduït tot allò necessari per fer una operació. Llavors procedirem a tractar-la.
- El nostre codi principal, comença a la posició de memòria 200h. Quan arribem a aquesta línia de codi, entenem que l'usuari ja ha introduït: ***nombre + operació + nombre + símbol "="***. Aprofitarem el fet que la nostra parella de registres HL apunta a la posició de memòria següent al darrer símbol que ens han introduït. Per evitar moure aquests apuntadors, escollirem una altra parella de registres (BC) per moure'ns entre aquestes posicions de memòria.
- L'algorisme ara és senzill. La nostra intenció és tenir en un registre el primer nombre introduït, en un altre registre el segon nombre introduït i finalment carregar a l'acumulador el valor que correspon a l'operació per poder decidir quina hem d'efectuar. Com que tenim les dades a la memòria, el que fem és anar carregant de memòria a registres.
- A partir d'aquí, l'algorisme depèn de l'operació que vulguem efectuar.
- La que porta menys problemes és **l'AND**, ja que ni tan sols comporta *carry*.
- Per la **suma** hem de tenir en compte que pot tenir un *carry*. Per tractar això, quan tinguem el resultat de la suma li restem Ah. Si ens dona positiu, llavors saltem a una altra part del codi que ens imprimirà un 1 després de l'igual i tot seguit el

resultat de la suma menys Ah (5 + 5 = Ah; Ah – Ah = 0h positiu → saltem, imprimim 1 i després 0h → imprimim 10).

- Per la **resta** fem el mateix, però no cal que restem. Només mirem si el resultat de fer la resta ens fa saltar el bit de signe. Si és així, imprimim un signe negatiu "-" al davant de l'operació i després invertim el resultat (com si multipliquéssim per (-1)) per tenir l'expressió correcta.
- Per la **OR** sí hem de tenir en compte un *carry* però l'algorisme és gairebé idèntic al de la suma, només cal que canviem les operacions aritmètiques per aquesta lògica. Tot i que els resultats són diferents, el funcionament és prou similar.

El codi és el següent:

```
.org 100h

init: ;inicialitzacions

    MVI A,0h

    MVI B,0h

    MVI C,0h

    MVI D,0h

    MVI E,0h

    MVI H,0h

    MVI L,0h

    SPHL

    LXI H, E000h

    JMP loop

.org 24h ;interrupció: posem el valor a memòria

    IN 00h

    MOV M, A

    INX H
    CPI 3Dh
    JZ operar
```

```
    JMP loop

.org 200h

operar:

    ;no volem moure els punters de memòria que ens controlen la sortida per
    ;pantalla així que assignem uns punters auxiliars que utilitzarem per moure'ns
    ;per memòria i així posar els nombres a registres i mirar quina operació dur a
    ;terme

    MOV B, H

    MOV C, L

    ;Com que cada cop que s'introdueix un caràcter per teclat augmentem la posició
    ;dels punters de memòria, quan haguem arribat a aquest punt, els nostres
    ;punters apunten a la direcció posterior de l'igual i no volem fer res amb l'igual,
    ;així que necessitem decrementar la memòria dos cops per apuntar així al segon
    ;nombre que ens han introduït

    DCX B

    DCX B


    ;carreguem el segon nombre introduït de la memòria a l'acumulador i el posem
    ;a un registre i el convertim d'ASCII a nombre

    LDAX B

    SUI 30h

    MOV E, A


    ;ara carregarem el primer nombre introduït. Per això, ens saltem la posició de la
    ;operació (decrementem un) i tornem a decrementar per apuntar a la posició del
    ;nombre, llavors el passem a l'acumulador i després a un registre.

    DCX B

    DCX B

    LDAX B

    SUI 30h
```

```
        MOV D, A


        ;ara tenim els dos nombres carregats als registres, hem de mirar quina operació
        ;ens toca fer i operar, per això hem d'incrementar un la posició de la memòria

        INX B

        ;carreguem la operació a l'acumulador i la comparem per veure que tenim
        LDAX B
        CPI 2Bh
        JZ suma
        CPI 2Dh
        JZ resta
        CPI 41h
        JZ and
        CPI 4Fh
        JZ or

;////////////////////////////////// SUMA ////////////////////////////////////////

.org 250h

suma:

        ;tenim un nombre al registre D i un nombre al registre E, els hem de sumar i per
        ;fer això, necessitem tenir-ne un a l'acumulador i després sumar l'altre

        MOV A, D
        ADD E


        ;pot ser que la suma ens doni de més d'un dígit, si és així, ho hem de tractar; per
        ;comprovar-ho, mirem si ens dona negatiu al restar-li 10 (Ah)

        SUI Ah


        ;si ens dona positiu, llavors vol dir que tenim carry.


        JP carry


        ;si ens ha donat negatiu, nomes teníem una xifra i hem de convertir això que
        ;tenim en codi ASCII

        ADI 3Ah
```

```
        ;si hem arribat fins aquí, ho ensenyem per pantalla i saltem a loop

        MOV M, A
        INX H
        JMP loop

;/////////////////////////////////// CARRY ////////////////////////////////////

.org 300h

carry:

        ;si estem aquí vol dir que la suma ens ha donat de dos dígits, llavors només cal
        ;que posem un 1 al principi i el següent nombre és el que tenim guardat a
        ;l'acumulador

        MVI M, 31h

        INX H

        ;posem el contingut de l'acumulador en ASCII

        ADI 30h

        MOV M, A

        INX H

        JMP loop

;/////////////////////////////////// SUBBER ////////////////////////////////////

.org 350h

resta:

        ;posem un dels nombres a acumulador per treballar amb ell.

        MOV A, D

        SUB E


        ;ara tenim la resta a l'acumulador
        ;si ens ha donat negatiu, haurem de posar un signe
        ;podem comprovar-ho mirant el bit de signe
```

```
        JM signe


        ;si no ens ha donat negatiu, podem imprimir-ho directament i posem el valor
        ;ASCII corresponent
        ADI 30h
        MOV M, A
        INX H
        JMP loop

;////////////////////////////////// SIGNE ////////////////////////////////////////

.org 400h

signe:

        ;aquí estem si el resultat ens ha donat negatiu. Llavors hem de posar un signe
        ;negatiu al davant i posar el resultat correcte.

        ;com que està en negatiu, necessitem fer-li una inversió, guardem el nombre en
        ;un registre i ara ens és igual si sobreescrivim informació, ja que no la
        ;necessitem

        MOV D, A

        ;a l'acumulador posem un 0

        MVI A, 0h

        ;ara li restem el que tenim al registre on guardem el valor de la resta
        SUB D


        ;ja hem fet la inversió, ara només cal passar-lo a ASCII

        ADI 30h

        ;imprimim un '-' i després el valor

        MVI M, 2Dh

        INX H

        MOV M, A

        INX H
```

```
        JMP loop

;//////////////////////////////// AND ////////////////////////////////////

.org 450h

and:

        ;quan fem l'AND, com que sabem que 1 AND 1 = 1 i la resta dona 0 veiem que
        ;no pot donar mai més uns que el nombre més gran, es a dir, si fem 9 (1001b)
        AND qualsevol altre cosa, no ens pot donar cap altre 1 que no apareix ja al 9
        (1001b)

        ;per tant, no haurem de considerar la possibilitat de que ens doni un nombre
        de dues xifres

        MOV A, D

        ANA E

        ADI 30h

        MOV M, A

        INX H

        JMP loop

;//////////////////////////////// OR /////////////////////////////////////

.org 500h

or:

        ;quan fem l'OR sí que ens pot donar un nombre d'1 superior. Essencialment,
        ;estem fent una suma sense portar carris entre bits (si considerem el nombre en
        binari); aixíi, 8d OR 4d = 1000b OR 100b = 1100b = 12d

        ;per tant, aquíi hem de tenir en compte la possibilitat de tenir un carry, com a la
        suma, tanmateix, sabem que aquest carry mai passarà de 1

        MOV A, D

        ORA E

        ;un cop tenim feta la OR hem de mirar si tenim carry.

        SUI Ah
```

```
        ;si ens dóna positiu, llavors tenim carry.

        JP carryor

        ;si no ens ha donat positiu, no teníem carry
        ;reconvertim el nombre que teníem i l'imprimim per pantalla.

        ADI 3Ah

        MOV M, A

        INX H

        JMP loop

;////////////////////////// CARRYOR ////////////////////////////////////

.org 550h

carryor:

        ;fem el mateix procediment que a la suma, considerem un carry i després
        ;convertim el nombre que tenim guardat a l'acumulador

        MVI M, 31h

        INX H

        ADI 30h

        MOV M, A

        INX H

        JMP loop

;loop per no acabar el programa i esperar una entrada de l'usuari

.org 600h

loop:

        JMP loop
```

Respecte a les preguntes plantejades, realment ja estan respostes més a dalt, tot i així:

Com gestioneu el problema del signe? Com gestioneu el problema del overflow? Com gestioneu el problema del signe? I el problema del carry?

-***PROBLEMA DEL CARRRY A LA SUMA***. A la suma l'únic que pot passar és que tinguem *carry*: si sumem 5+5, 6+5, ..., sempre ens dóna un nombre de dues xifres. Clar que en hexadecimal això és diferent, perquè, per exemple, en el cas de 5 + 5 = Ah i en el cas de 9+9 = 12h.

Hem de tenir en compte que si intentem imprimir per pantalla aquest Ah o aquest 12h, per convertir-ho a ASCII hauríem de sumar-li 30h. Això ens donaria un 3Ah o un 42h, que no són valors associats al 10 o al 18, ni tan sols són valors de xifres. Llavors, el que volem fer és separar aquest nombre en dues xifres i tractar-lo.

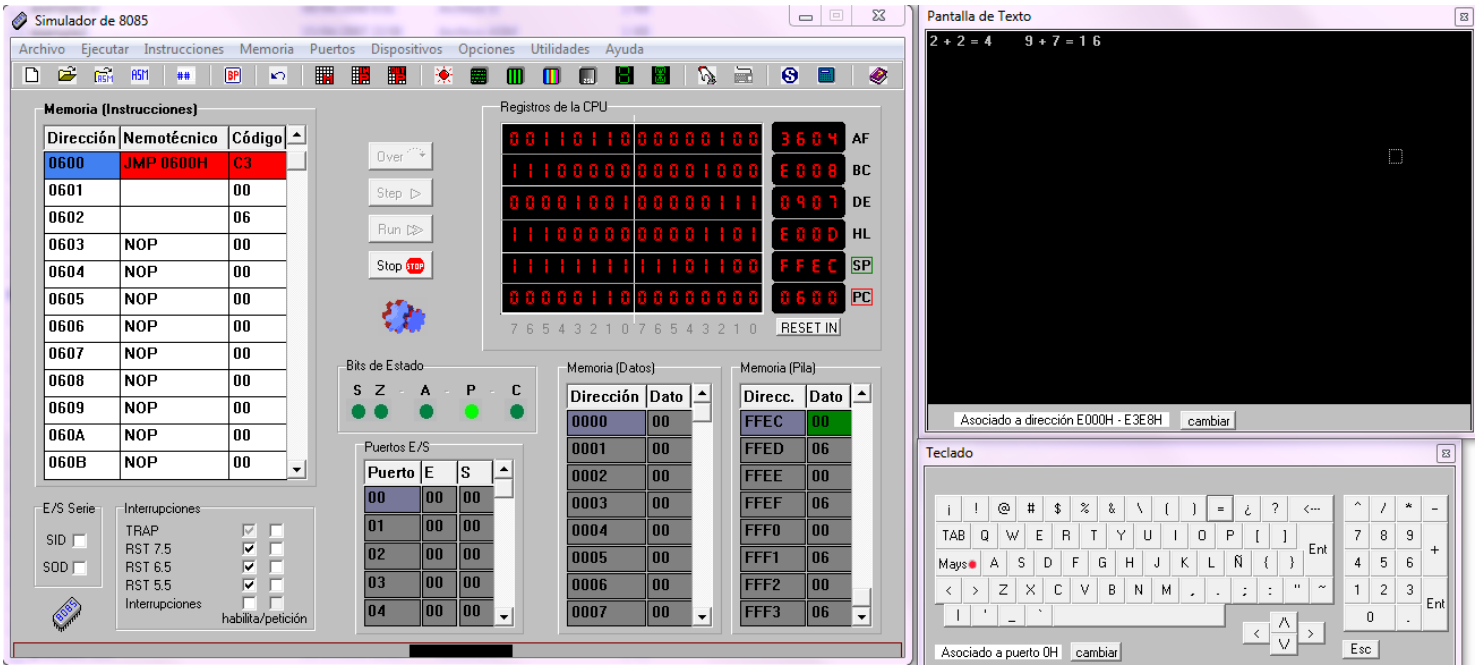
L'algorisme que ho soluciona és prou fàcil. Igual que en decimal restaríem 10 per tenir la unitat dels nombres, en hexadecimal restem Ah. Vegem que, per exemple 12h – Ah = 8h. Llavors, el que hem de fer és comprovar sistemàticament si quan restem Ah al nombre que tenim a l'acumulador, aquest ens segueix donant positiu. Si és així, llavors és que porta *carry*. Imprimim el *carry* al davant, que ja sabem que és 1, i el que queda imprimir al darrera és el resultat que ja tenim a l'acumulador. D'aquesta manera, ja tenim per pantalla el resultat ben ordenat.

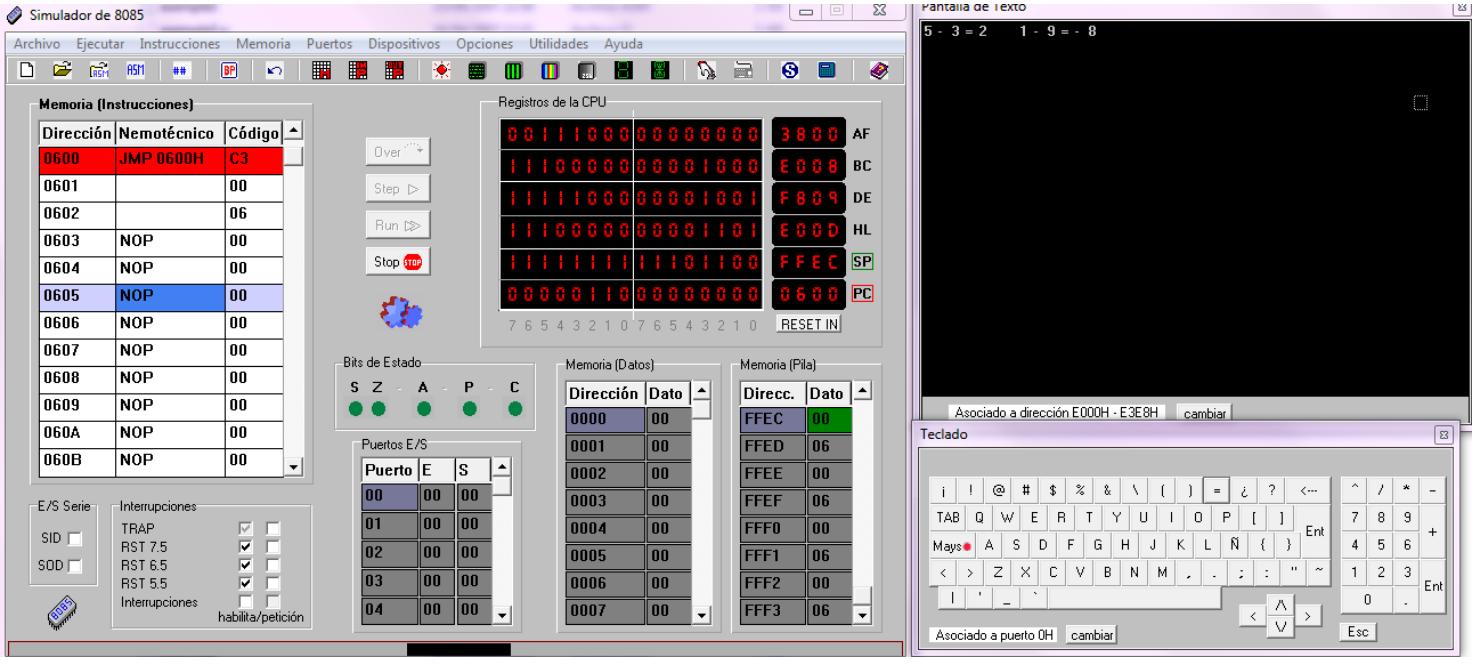
-***PROBLEMA DEL SIGNE A LA RESTA***. A la resta no hem de tenir en compte *carry*, ja que cap de les operacions que podem fer ens donaran més d'una xifra (com a màxim podem fer 0 – 9 = -9). Però sí que hem de tenir en compte el resultat negatiu.

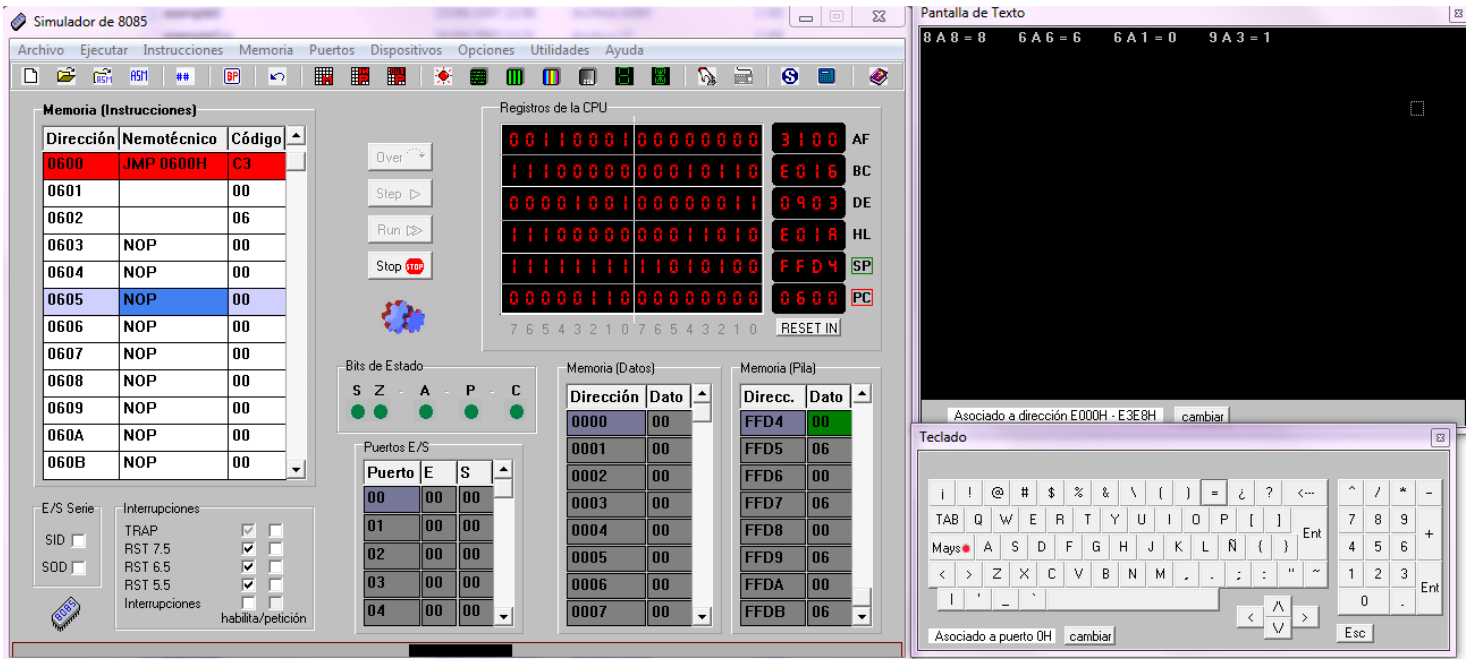
Aquest resultat és molt més fàcil de tractar, ja que, quan restem dos nombres i queda un resultat negatiu a l'acumulador, ens salta el bit de signe. Llavors només hem de comprovar-ho i, si està activat, saltar per posar el signe.

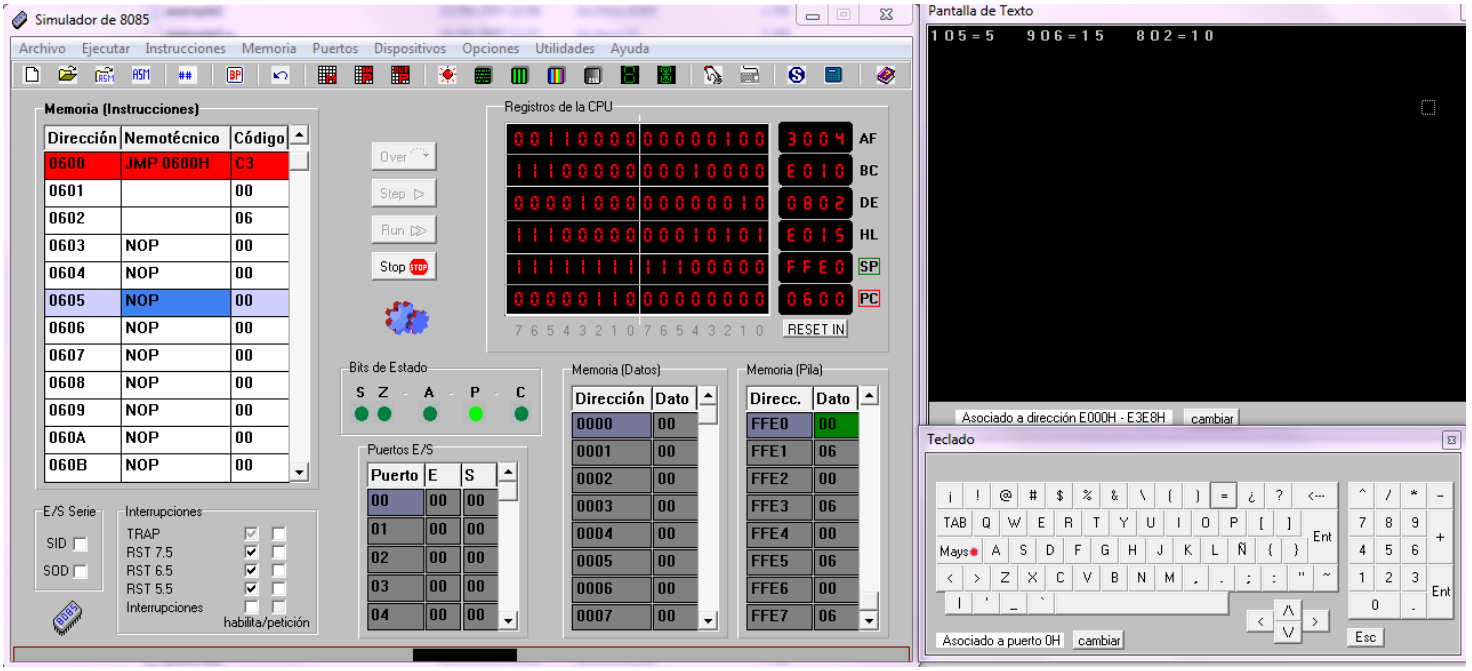
El primer que fem és imprimir el signe negatiu, ja que sabem que si hem arribat aquí és necessari. I després posar el resultat és prou fàcil: només hem d'invertir allò que tenim a l'acumulador. L'algorisme triat per fer-ho és el de posar un 0 a l'acumulador i restar-li allò que teníem abans. D'aquesta manera fem, per exemple, 0 - (-9) = 9. Així, el resultat queda imprès per pantalla correctament.

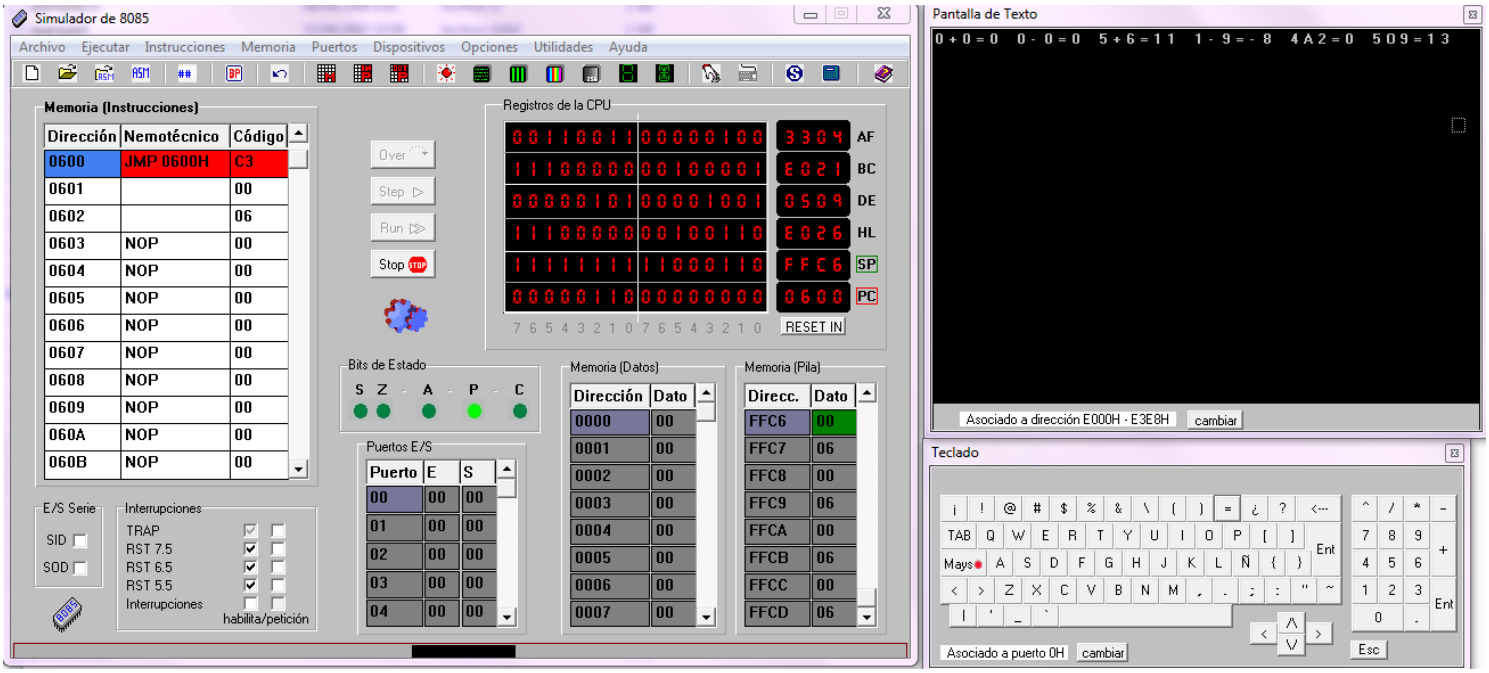
Afegeixo unes quantes imatges del funcionament del programa. Primer, cada operació per separat i, després, una mostra de totes juntes:






Simulador de 8085
Archivo Ejecutar Instrucciones Memoria Puertos Dispositivos Opciones Utilidades Ayuda
Memoria (Instrucciones)
Dirección Nemotécnico Código
0600 JMP 0600H C3
0601 00
0602 06
0603 NOP 00
0604 NOP 00
0605 NOP 00
0606 NOP 00
0607 NOP 00
0608 NOP 00
0609 NOP 00
060A NOP 00
060B NOP 00
Over
Step
Run
Stop
Registros de la CPU
AF BC DE HL SP PC
RESET IN
Bits de Estado
S Z A P C
Puertos E/S
Puerto E S
Memoria (Datos)
Dirección Dato
Memoria (Pila)
Direcc. Dato
E/S Serie
SID SOD
Interrupciones
TRAP RST 7.5 RST 6.5 RST 5.5 Interrupciones
habilita/petición
Pantalla de Texto
8A8=8 6A6=6 6A1=0 9A3=1
Asociado a dirección E000H - E3E8H cambiar
Teclado
Asociado a puerto 0H cambiar
Esc
Ent


Simulador de 8085
Archivo Ejecutar Instrucciones Memoria Puertos Dispositivos Opciones Utilidades Ayuda
Memoria (Instrucciones)
Dirección Nemotécnico Código
0600 JMP 0600H C3
0601 00
0602 06
0603 NOP 00
0604 NOP 00
0605 NOP 00
0606 NOP 00
0607 NOP 00
0608 NOP 00
0609 NOP 00
060A NOP 00
060B NOP 00
Over
Step
Run
Stop
Registros de la CPU
AF BC DE HL SP PC
RESET IN
Bits de Estado
S Z A P C
Puertos E/S
Puerto E S
Memoria (Datos)
Dirección Dato
Memoria (Pila)
Direcc. Dato
E/S Serie
SID SOD
Interrupciones
TRAP RST 7.5 RST 6.5 RST 5.5 Interrupciones
habilita/petición
Pantalla de Texto
105=5 906=15 802=10
Asociado a dirección E000H - E3E8H cambiar
Teclado
Asociado a puerto 0H cambiar
Esc
Ent


Simulador de 8085
Archivo Ejecutar Instrucciones Memoria Puertos Dispositivos Opciones Utilidades Ayuda
Memoria (Instrucciones)
Dirección Nemotécnico Código
0600 JMP 0600H C3
0601 00
0602 06
0603 NOP 00
0604 NOP 00
0605 NOP 00
0606 NOP 00
0607 NOP 00
0608 NOP 00
0609 NOP 00
060A NOP 00
060B NOP 00
Over
Step
Run
Stop
Registros de la CPU
AF BC DE HL SP PC
RESET IN
Bits de Estado
S Z A P C
Puertos E/S
Puerto E S
Memoria (Datos)
Dirección Dato
Memoria (Pila)
Direcc. Dato
E/S Serie
SID SOD
Interrupciones
TRAP RST 7.5 RST 6.5 RST 5.5 Interrupciones
habilita/petición
Pantalla de Texto
0+0=0 0-0=0 5+6=11 1-9=-8 4A2=0 509=13
Asociado a dirección E000H - E3E8H cambiar
Teclado
Asociado a puerto 0H cambiar
Esc
Ent

Respecte a les preguntes plantejades per respondre:

Quina diferència hi ha entre la suma i la OR?

i. Són iguals
ii. **La OR és una operació lògica i la suma és una operació aritmètica**
iii. La OR és una operació aritmètica i la suma és una operació lògica
iv. Cap de les anteriors és correcta

Podem establir una classificació on el tipus d'operació de la OR s'anomena lògica (com AND, NOR, XOR, NXOR, NAND, INV...) i l'operació de la suma s'anomena aritmètica (suma, resta, multiplicació, divisió...).

La instrucció STA 1234h

i. És una operació que carrega el contingut de la posició de memòria 1234h en l'acumulador
ii. **Fa servir adreçament directe**
iii. Fa servir adreçament immediat
iv. Totes són certes

La instrucció STA 1234h guarda una còpia del contingut actual de l'acumulador a la posició de memòria 1234h. Per tant, la primera opció ja és incorrecte. Fa servir adreçament directe, ja que l'adreça es proporciona directament. No hi ha cap tipus d'adreçament immediat, ja que no apareix cap immediat a la operació. I, evidentment, la última no pot ser cerca si ja fallen anteriors.

Conclusions

En aquesta pràctica s'ha proposat una solució al problema plantejat, que demanava realitzar un programa que realitzés les funcions d'una aplicació calculadora, amb les operacions de suma, resta, AND i OR.

- S'han repassat els coneixements obtinguts a teoria.
- S'ha realitzat els exercicis proposats a la pràctica.
- S'han respost les qüestions plantejades.
- S'ha treballat amb els dispositius d'entrada/sortida.
- S'ha treballat amb el funcionament de les interrupcions.