Pràctica 6: Pila i entrada/sortida

Introducció / Objectius

En aquesta pràctica seguirem treballant amb el simulador i8085 i aprofundint en els seus conceptes i eines de treball, centrant-nos en el funcionament de la pila i l'adreçament d'entrada i sortida, i així poder entendre millor la distribució de memòria del microprocessador. A continuació un resum dels objectius:

- Entendre la distribució de memòria del microprocessador.
- Comprendre el funcionament i ús de la pila.
- Veure la memòria que ocupa la pila.
- Usar els dispositius d'entrada/sortida.
- Comprendre l'adreçament dels dispositius d'entrada sortida.
- Usar la interrupció TRAP per un algorisme senzill.

Part I

Preguntes:

1. **L'adreçament de la instrucció LXI és:**
   a) Directe
   b) Indirecte
   c) Immediat
   d) Implícit
2. **Quina instrucció guarda el PC a la Pila?**
   a) PUSH PC
   b) POP PC
   c) CALL
   d) MOV M, PC
3. **Quin espai ocupa en memòria la subrutina 'suma'?**
   La subrutina 'suma' ocupa l'espai de memòria des de la posició 619h fins a la posició 620h, és a dir, 8 bytes de memòria.
4. **Quants cicles triga en executar-se la subrutina 'suma'?**
   Comptarem totes les instruccions que té:
   - **PUSH** → 12 cicles.
   - **LDAX** → 7 cicles.
   - **ADD M** → 7 cicles.
   - **STAX** → 7 cicles.
   - **INX** → 6 cicles.
   - **INX** → 6 cicles.
   - **POP** → 10 cicles.

- **RET** → 10 cicles.

Observem que tenim en compte el **RETURN** i que només comptabilitzem un sol cop que s'executa, és a dir, no sumem tots els cops que s'executa. Fan un total de 65 cicles de rellotge.

Estudi de l'espai de memòries del microprocessador

Tasca 1:

Per fer-ho més fàcil i entenedor, en lloc de dibuixar tota la memòria (gairebé tota està buida) només posaré les parts que tenen codi implementat per nosaltres i, a més, ho faré per blocs.

.data 0b

Posició	Contingut
00h	**1h**
01h	**2h**
02h	**3h**
03h	**4h**

Les posicions **00h i 01h** corresponen a **mat1** i tenen els valors **1h i 2h**. Les posicions **02h i 03h** corresponen a **mat2** i tenen els valors **3h i 4h**. Les posicions de memòria **00h i 01h** comencen amb els valors **1h** i **2h**, però després són modificats (a la rutina suma mitjançant la instrucció ***STAX D***) i guarden els valors **4h** i **6h**.

Posició	Contingut
001Ch	**00h – guarda ACC i STT**
001Dh	**00h – guarda ACC i STT**
001Eh	**0Fh – guarda el PC**
001Fh	**06h – guarda el PC**
0020h	**00h**
0021h	**00h**

Les posicions **0020h i 0021h** corresponen a la pila. Tanmateix, amb la instrucció SPHL situem la pila a la posició de memòria **0020h**, per tant, quan cridem al **CALL SUMA**, guarda el PC a les posicions de memòria **001Fh i 001Eh**. Seguidament guardem el valor

de l'acumulador i del registre d'estats mitjançant la instrucció **PUSH PSW**, aquests valors es guarden a les posicions **001Ch i 001Dh**.

En resum, s'ha especificat quines són les instruccions que modifiquen dades a la memòria. Aquestes són:

- **STAX D**, la cridem per guardar el resultat de la suma sobre la tercera matriu.
- **CALL SUMA**, que guardarà el PC a la memòria.
- **PUSH PSW**, que guardarà l'acumulador i el registre d'estats a memòria.

Aquestes últimes instruccions necessiten el **RET** i el **POP** respectivament, per recuperar les dades.

Seguidament adjunto el cos del programa i identificaré on està el bloc '**loop**'.

Posició	Contingut
600h	**LXI H, 0020h**
601h	
602h	
603h	**SPHL**
604h	**MVI B, 02h**
605h	
606h	**LXI D, 0000h**
607h	
608h	
609h	**LXI H, 0002h**
60Ah	
60Bh	
60Ch	**CALL 0615h (loop)**
60Dh	
60Eh	

60Fh	**DCR B**
610h	**JNZ 060Ch**
611h	
612h	
613h	**NOP**
614h	**HLT**
615h	**PUSH PSW**
616h	**LDAX D**
617h	**ADD M**
618h	**STAX D**
619h	**INX H**
61Ah	**INX D**
61Bh	**POP PSW**
61Ch	**RET**

Des de la posició **615h** fins a la **61Ch** trobem la rutina '**loop**'. Des de la posició **60Ch** fins a la **610h** tenim la rutina '**suma**'. La resta de posicions especificades són del cos del programa principal, sense subrutines.

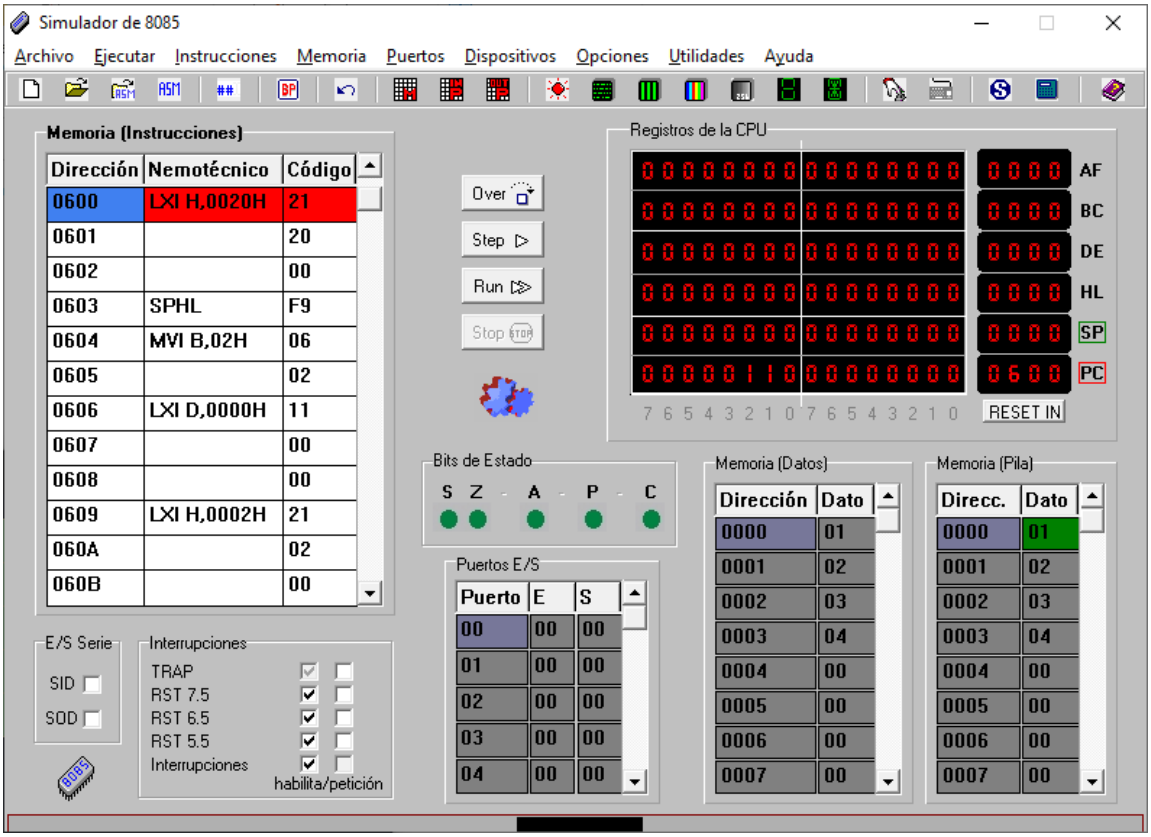
Funcionament de la pila

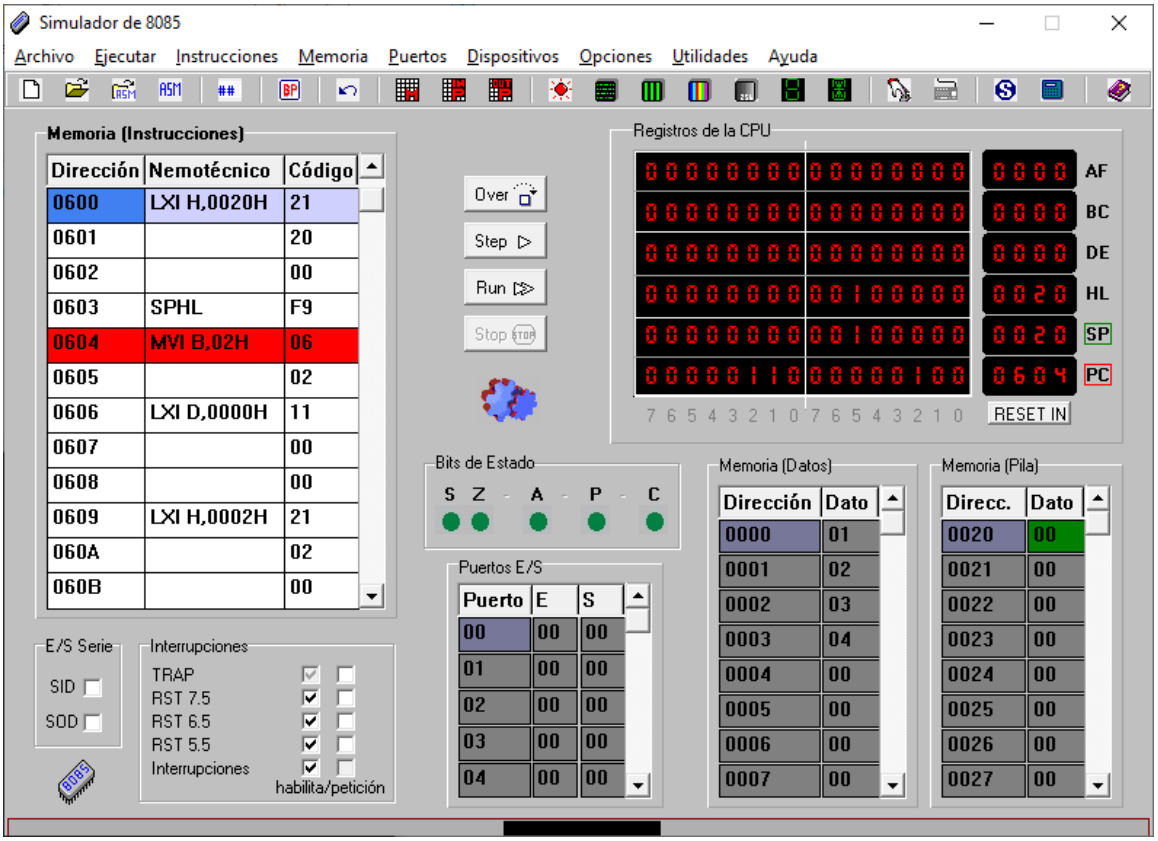
Tasca 2:

Situem la pila en una posició determinada mitjançant la instrucció **SPHL**, que posa la pila en la posició de memòria indicada pel contingut del parell de registres **H**, **L**. Després, usem també les instruccions **CALL** per guardar el **PC** a la pila (podem usar per la resta de registres la instrucció **PUSH**) i la instrucció **RET** per recuperar el **PC** (i **POP** per recuperar els registres).

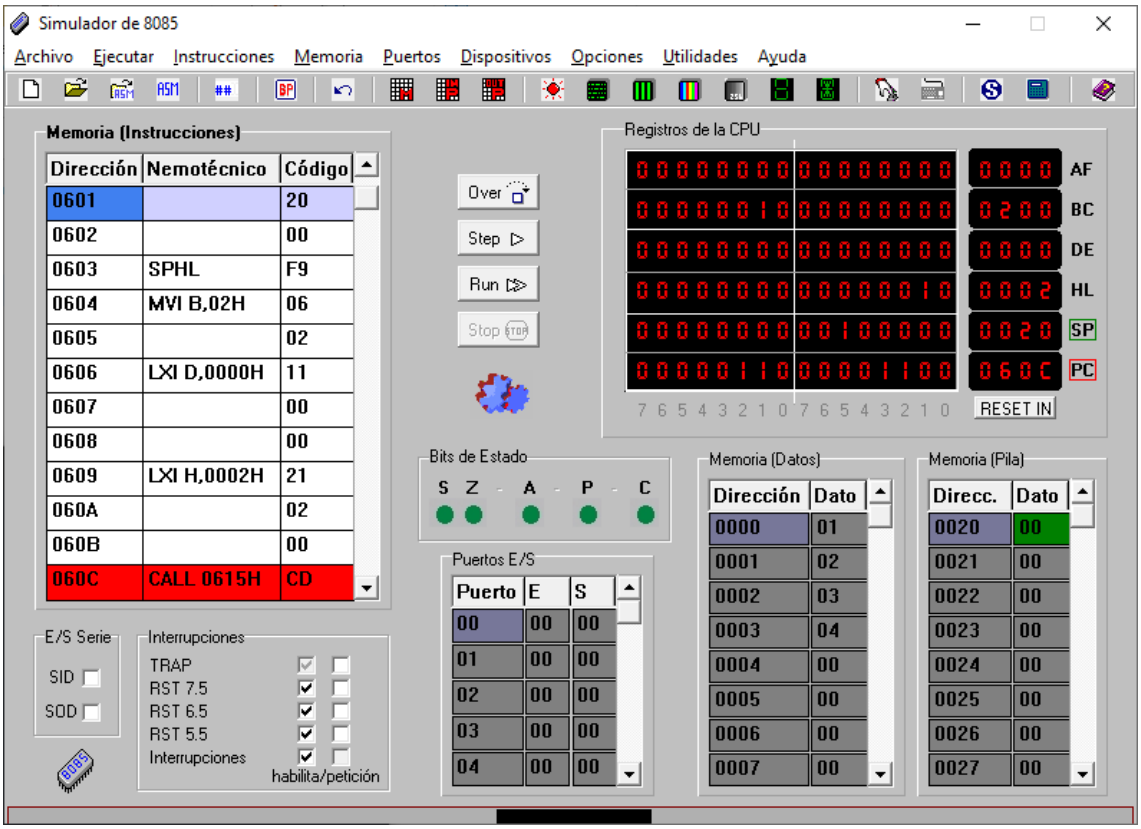
La pila creix en sentit contrari a com ho fa el codi principal. Per exemple, en aquest programa situem la pila a la posició **0020h**. La pila creix en sentit de posicions de memòria decreixent, és a dir, les primeres dades les guardarà en les posicions **001Fh** i **001Eh** (el PC) i a les següents, **001D** i **001C**, guardarà l'acumulador i el registre d'estats. Per tant, decreix en el sentit que les posicions de memòria creixen.

Modifiquem la pila amb la instrucció **SPHL**, al principi aquesta es troba situada a la posició **0000h** i, després d'executar-la, es troba a la posició **0020h**.

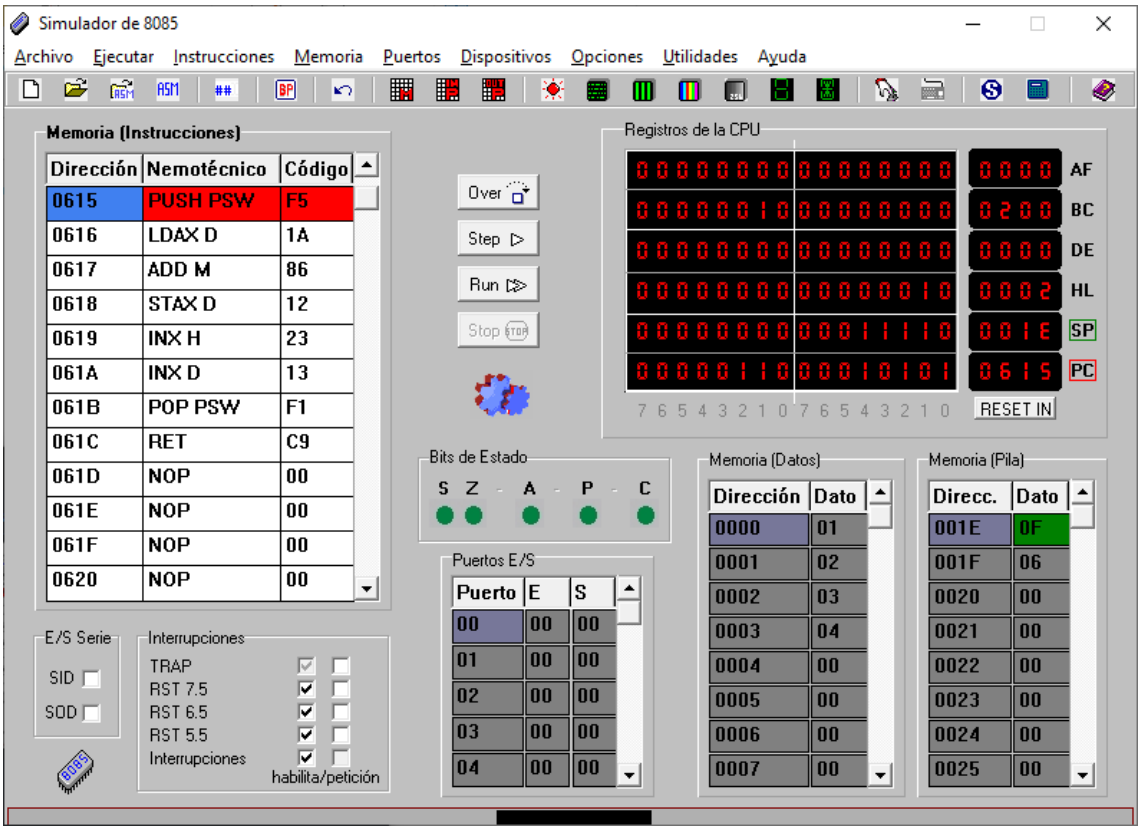




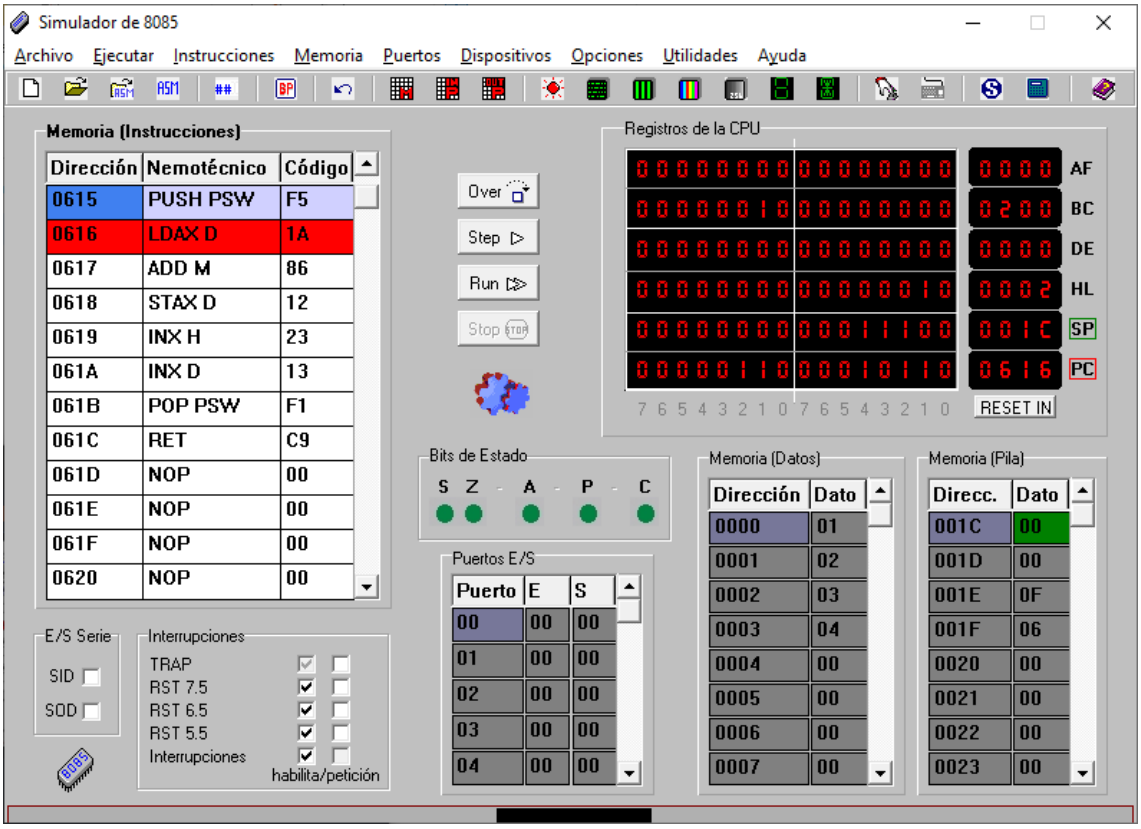
La següent instrucció que la modifica és la de **CALL SUMA**, abans d'executar-la la pila es troba a la posició **20h**.



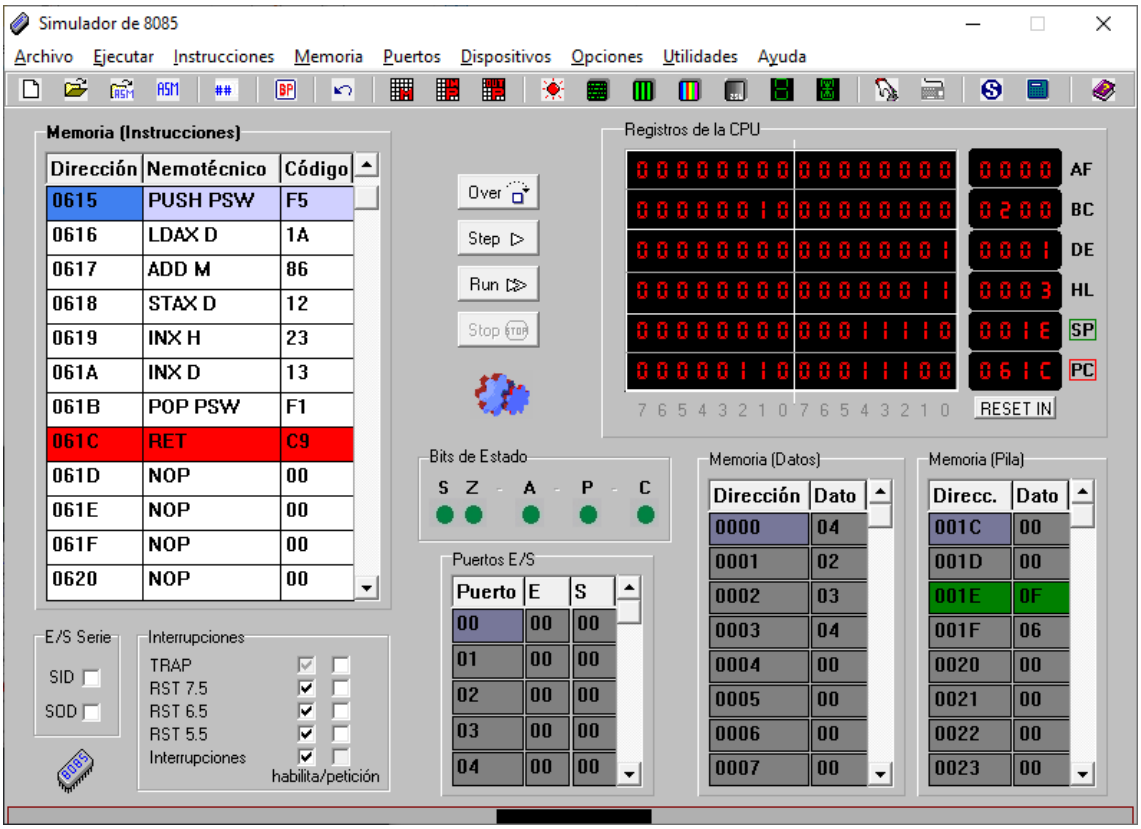
Després d'executar el **CALL** estem a la posició **001Eh**, ja que hem de guardar el program counter.

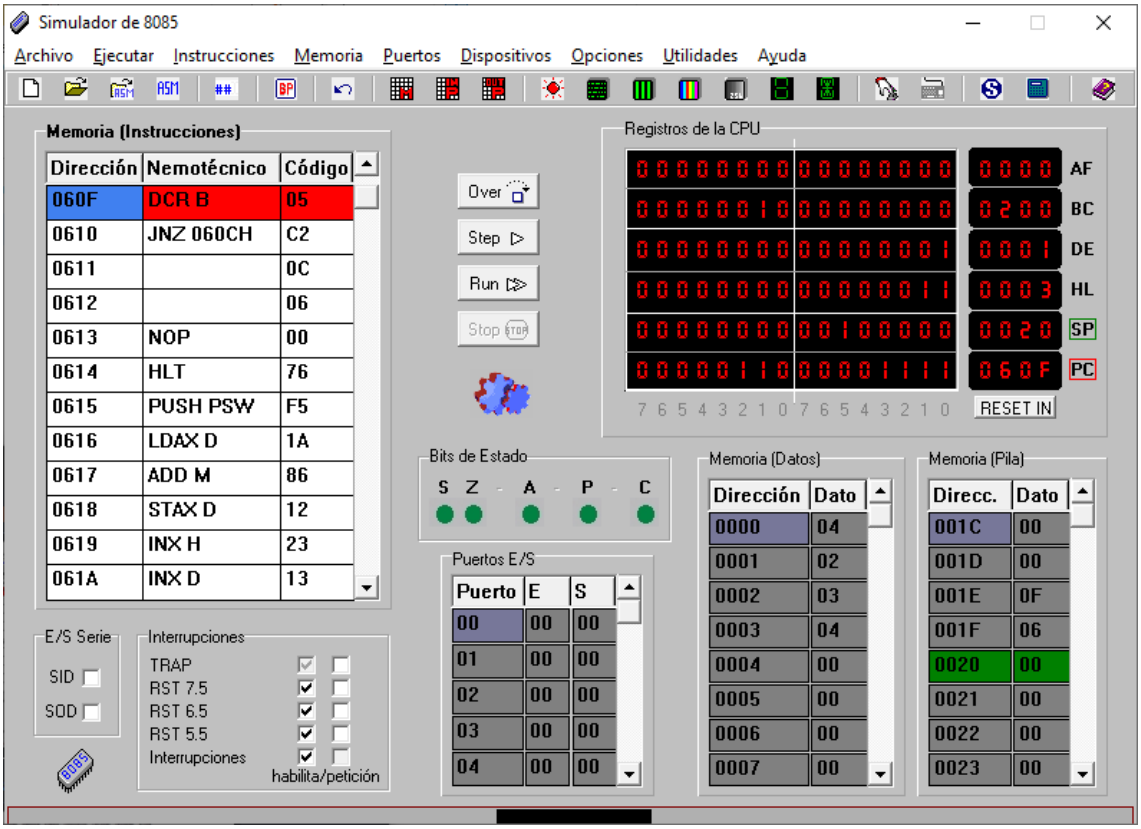


Aquesta imatge és l'anterior també d'executar el **PUSH**, ja que quan guardem els valors de l'acumulador i el registre d'estats, la pila es torna a moure:



Finalment, posaré les dues imatges corresponents al fer el **POP** i el **RETURN**:

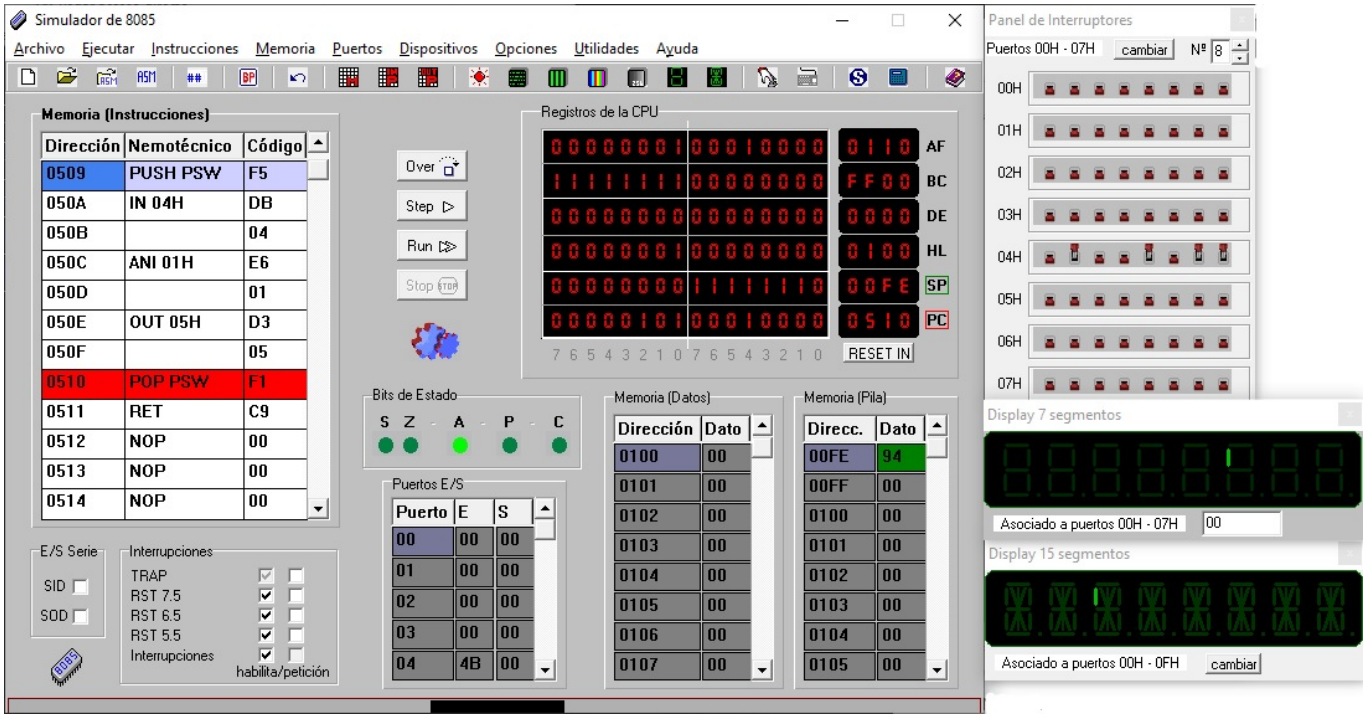




Part II

Tasca 3:

La subrutina '**ports**' agafa el valor que introduïm pel port **04h**, que tenim per defecte el panell d'interruptors i ho posa a l'acumulador. Es a un **AND** amb el valor guardat a l'acumulador i el valor **00000001** i es deixa a l'acumulador. Aquest valor és mostrat pel port **05h** mitjançant la instrucció **OUT 05h**. A més, abans havíem guardat el contingut de l'acumulador i el registre d'estats amb la instrucció **PUSH PSW** i ara ho recuperem amb el **POP**.



Part III

Se'ns demana fer un programa que accepti només nombres del 0 al 5 per teclat i els ensenyi pel display de 7 segments; així mateix, quan premem la lletra 'c' es netegi el display. Per fer aquest programa he suposat que l'usuari no premerà cap altre tecla que no siguin els nombres del 0 al 5 o la lletra 'c'; en cas de prémer una altre tecla, pot sortir un altre resultat o que el display es netegi.

L'algorisme és molt senzill. Per desenvolupar-ho primer he analitzat quina és la combinació de nombres binaris que corresponen a cada nombre que volem representar al display:

- **Zero**: 01110111b
- **Un**: 01000100b
- **Dos**: 00111110b
- **Tres**: 01101110b
- **Quatre**: 01001101b
- **Cinc**: 01101011b

També m'he fixat en quins valors corresponen als nombres entrats per teclat, des del 0 fins al 5: **30h**, **31h**, **33h**, **34h**, **35h**. I després la 'c': **43h**. De manera que com veiem, no es pot trobar una funció que expressi directament una correspondència entre els valors que necessitem, per representar-los al display de 7 segments, i els valors que prenen al ser introduïts per teclat. Per això, he decidit guardar a memòria aquests valors i fer una funció que, donat un valor per teclat (entre **30h** i **35h**, en el cas dels nombres; i el valor **43h** en el cas de la lletra 'c'), accedeixi a la posició de memòria on està guardat el valor corresponent a ensenyar pel display perquè posteriorment el carregui a l'acumulador, per poder-lo ensenyar mitjançant la instrucció **OUT**.

D'altra banda, el programa ha de començar a la posició **0024h**, que és on es troba el codi que s'executa quan es detecta la interrupció **TRAP**. Però no només volem introduir un nombre, per tant, el final del codi contempla un bucle infinit:

```
loop:
        JMP loop
```

D'aquesta manera, el programa no acaba mai a no ser que nosaltres ho especifiquem. Així, que quan haguem d'introduir un altre nombre, per la interrupció **TRAP**, el programa saltarà altre cop a la posició **0024h**, farà el que hem explicat abans, i es quedarà altre cop esperant a que introduïm un nombre.

El codi és el següent:

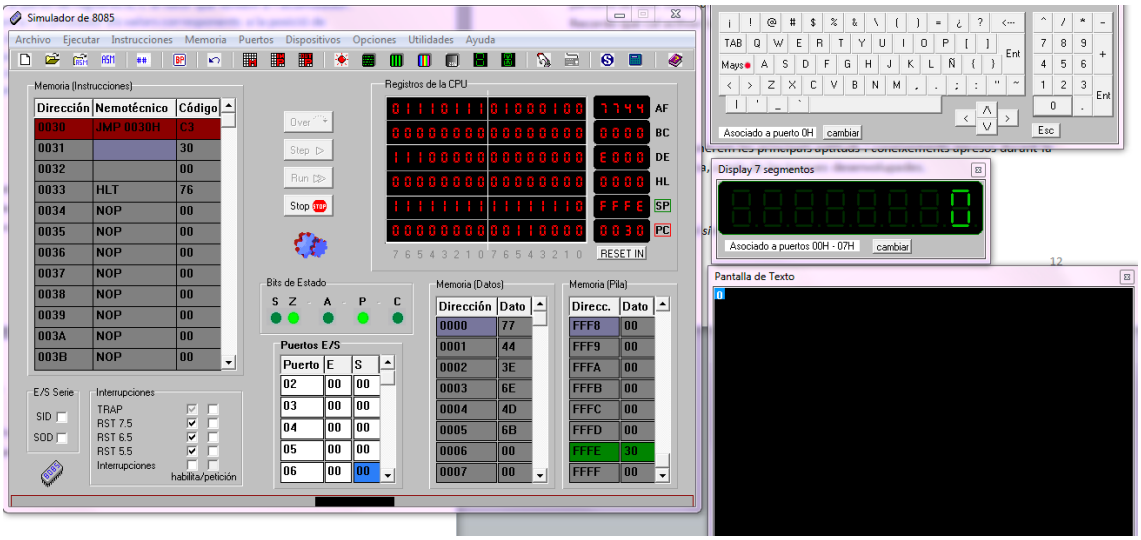
```
.data 00h ;valors necessaris per al display
    zero: db 01110111b
    un: db 01000100b
    dos: db 00111110b
    tres: db 01101110b
    quatre: db 01001101b
    cinc: db 01101011b
.data 13h ;valor per a netejar el display clear:
    db 00000000b
.org 24h
    IN 00h ;obtenim el valor del port 00h i el carreguem a l'acumulador.
    SUI 30h ;li restem 30 per obtenir la posició de memòria on hem
            ;guardat les dades per ensenyar-ho al display
    MOV C, A ;posem al parell de registres B, C el valor que teníem a l'acumulador.
    LDAX B ;Carreguem a l'acumulador els valors corresponents a la posició de
           ;memòria del contingut dels registres B, C. Havíem guardat el valor
           ;introduït – 30h per obtenir la posició de memòria adequada, estem
           ;accedint a un valor a ensenyar
    OUT 07h ;carreguem el valor de l'acumulador al display de 7 segments.
loop: ;bucle infinit
    JMP loop
HLT ;acabem
```

Tanmateix, com a proposta de millora, s'ha proposat ensenyar-ho també per la pantalla de text. Es pot fer fàcilment guardant el resultat de l'**IN** a la posició de memòria adequada. El programa modificat quedaria de la següent forma:

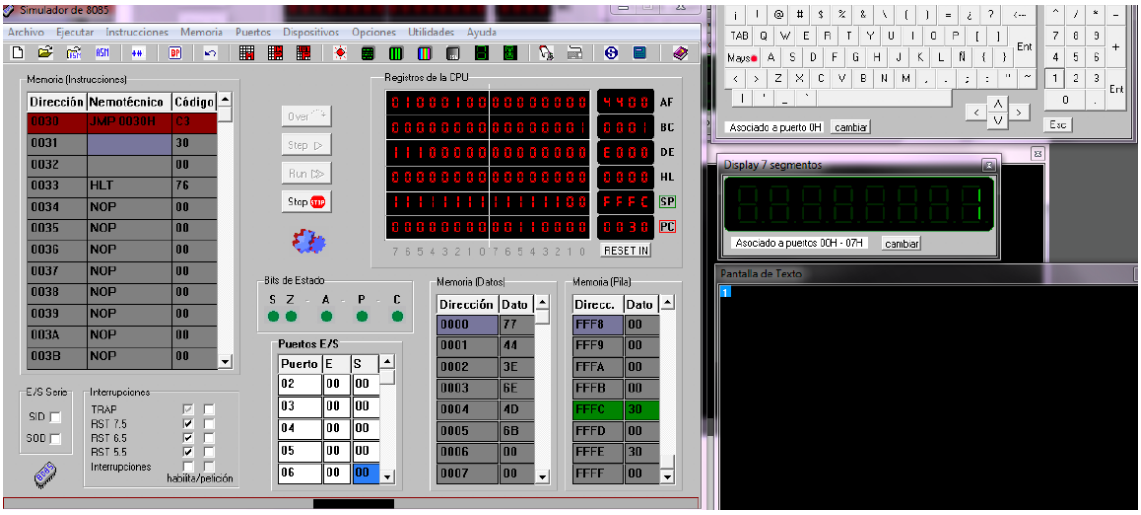
```
.data 00h ;valors necessaris per al display
    zero: db 01110111b
    un: db 01000100b
```

```
        dos: db 00111110b

        tres: db 01101110b

        quatre: db 01001101b

        cinc: db 01101011b

.data 13h ;valor per a netejar el display clear:

        db 00000000b

.org 24h

        IN 00h ;obtenim el valor del port 00h i el carreguem a l'acumulador.

        LXI D, E000h ;posem al registre D el valor E000h.

        STAX D ;guardem el que tenim a l'acumulador (és a dir, el valor d'allò que volem
                ;mostrar) a la posició de memòria que indica el valor del contingut del
                ;registre D, és a dir, estem posant a aquesta posició de memòria el valor
                ;del que ens han introduït a l'IN. Aquesta posició de memòria correspon a
                ;la primera posició de la pantalla de text.

        SUI 30h ;li restem 30 per obtenir la posició de memòria on hem

                ;guardat les dades per ensenyar-ho al display

        MOV C, A ;posem al parell de registres B, C el valor que teníem a l'acumulador.
        LDAX B ;Carreguem a l'acumulador els valors corresponents a la posició de
                ;memòria del contingut dels registres B, C. Havíem guardat el valor
                ;introduït – 30h per obtenir la posició de memòria adequada, estem
                ;accedint a un valor a ensenyar

        OUT 07h ;carreguem el valor de l'acumulador al display de 7 segments.

loop: ;bucle infinit

        JMP loop

HLT ;acabem
```

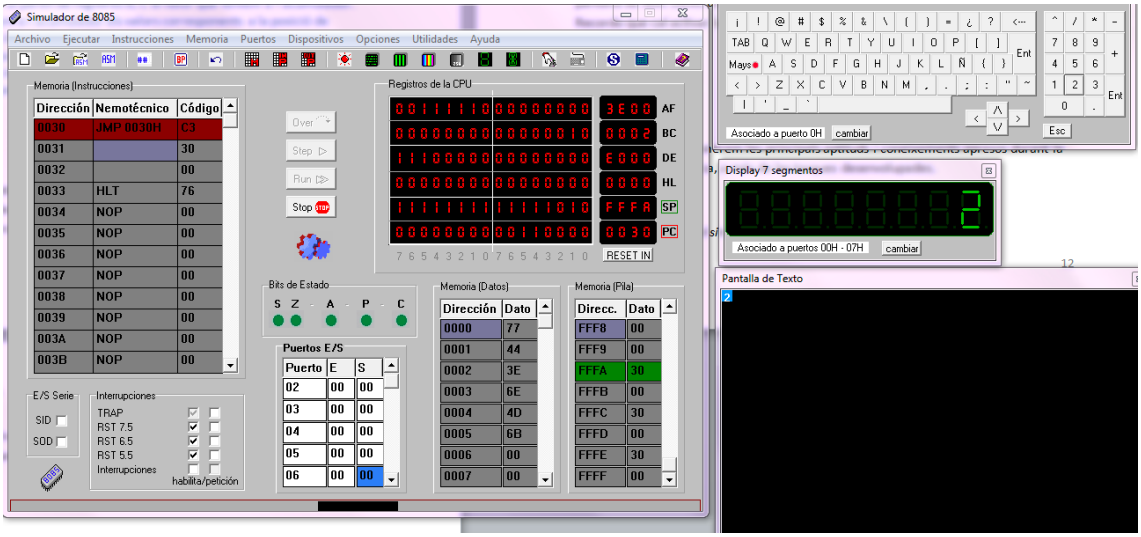
A continuació deixo unes captures de pantalla, cada una corresponent als diferents nombres. (0,1,2,3,4,5) representats tant amb el display de 7 segments com amb la pantalla de text. Deixo una última captura també del display netejat, utilitzant la lletra 'c'. Recordo que cal activar la interrupció **TRAP** perquè el programa funcioni correctament.



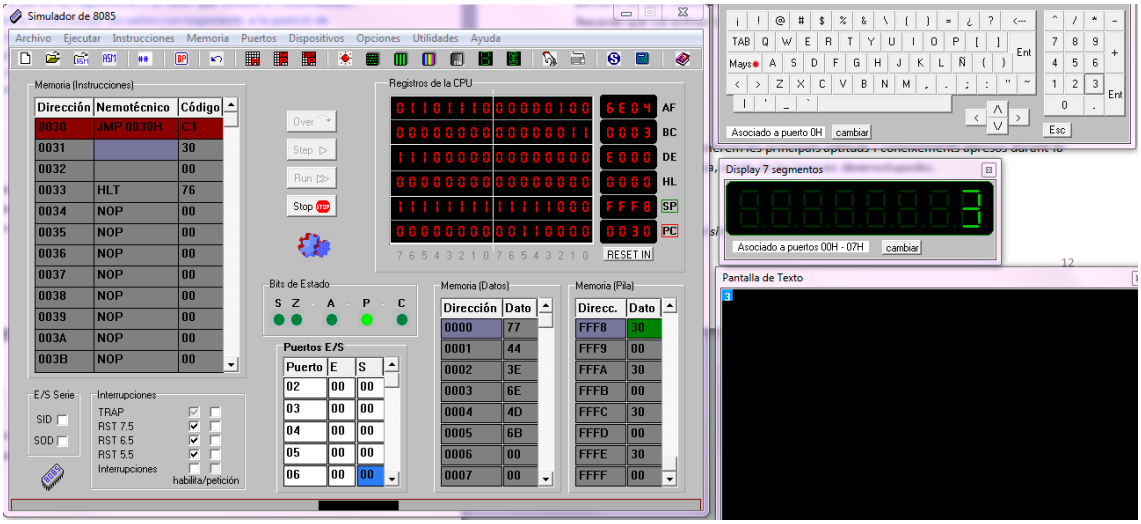
Representació del nombre 0



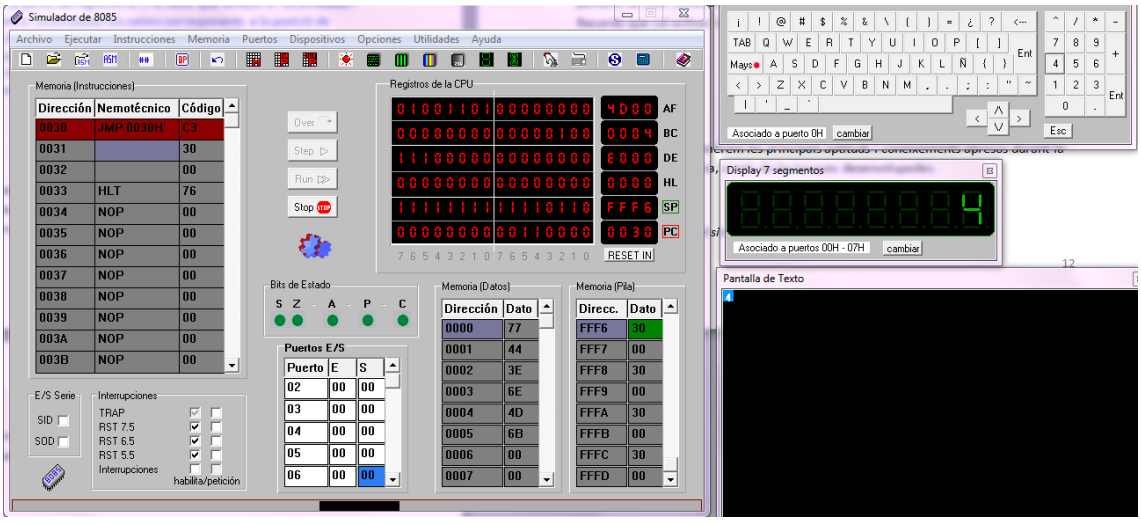
Representació del nombre 1



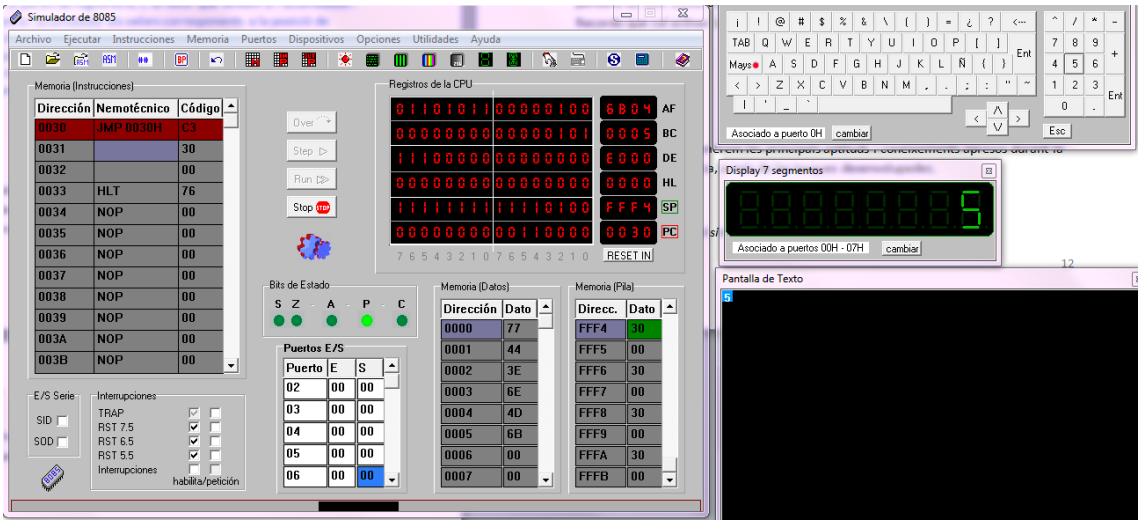
Representació del nombre 2



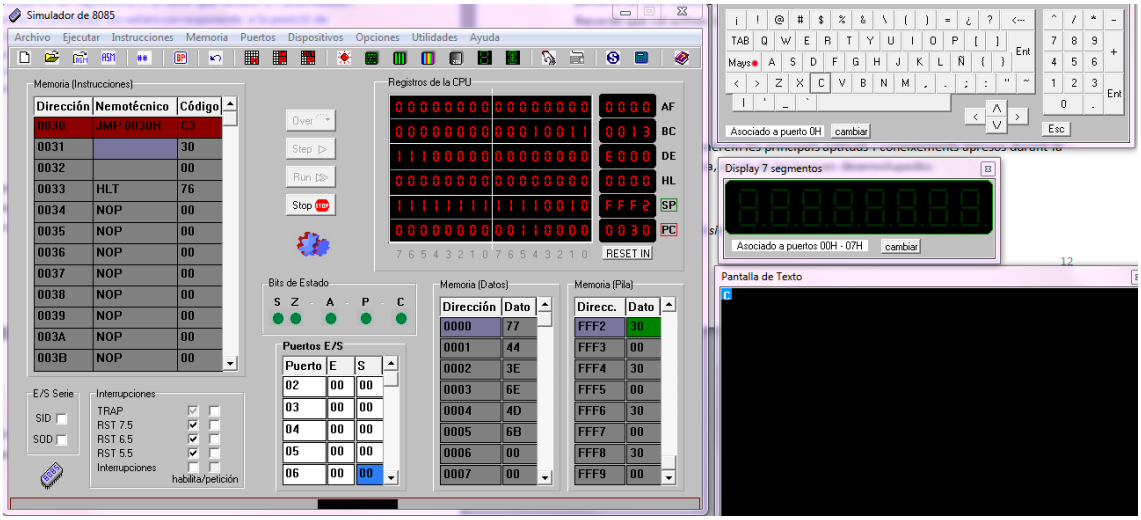
Representació del nombre 3



Representació del nombre 4



Representació del nombre 5



*Representació del **CLEAR** prement la tecla 'c'*

Conclusions

En aquesta pràctica s'ha proposat una altra solució al problema de sumar dos vectors i guardar-los en un tercer vector. S'ha estudiat la distribució de memòria del microprocessador i el funcionament dels dispositius d'entrada/sortida. A continuació un resum de les tasques realitzades:

- S'han realitzat els exercicis proposats a classe.
- S'ha analitzat un codi sobre la suma de vectors.
- S'ha analitzat el mapa de memòria d'un codi.
- S'ha analitzat el funcionament dels dispositius d'entrada/sortida.
- S'ha fet un petit programa que ensenya uns valors pel display de 7 segments.