3. Simulator de microprocesor (I)

Simulatorul Emu8086 (www.emu8086.com) este un emulator pentru microprocesor ce conţine (are integrat) un Asamblor 8086. Emulatorul rulează programele pe o Maşină Virtuală, emulând un hardware real precum ecranul monitorului, memoria şi dispozitivele de intrare/ieşire. Setul de instrucţiuni 8086 stă la baza tuturor microprocesoarelor, inclusiv Pentium şi Athlon. Toate instrucţiunile Intel, chiar şi directive precum MASM şi TASM sunt suportate de Emu8086, acesta oferind o soluţie completă în învăţarea limbajului de asamblare. Emu8086 rulează programele ca un microprocesor 8086 real: codul sursă este asamblat şi executat de către emulator pas cu pas, existând posibilitatea de a urmări modificările apărute în regiştri, flag-uri şi memorie în timpul rulării programelor.

Emu8086 include şi un Tutorial, plus o mulţime de programe date ca model. Emu include de asemenea şi câteva dispozitive externe virtuale, precum un robot, un motor pas cu pas, afişaj cu led-uri, intersecţie gestionată de semafoare, etc; aceste dispozitive pot fi modificate sau clonate, codul lor sursă fiind disponibil. Emulatorul permite crearea unui nou proiect (opţiunea **new**), vizualizarea unor exemple deja existente (opţiunea **code examples**), urmărirea tutorialului (opţiunea **quick start tutor**) sau deschiderea fişierelor recent utilizate (opţiunea **recent files**) aşa cum se poate urmări în figura 3.1.

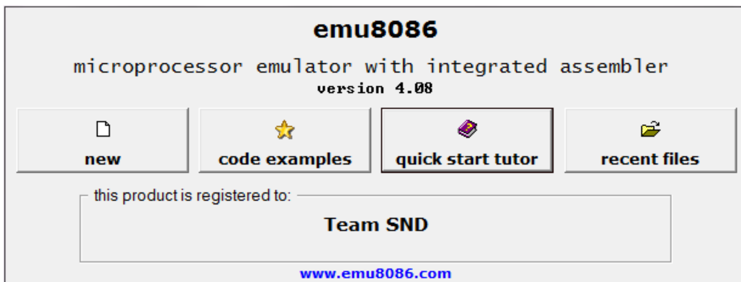


Fig. 3.1. Fereastra de început a Emu8086

După scrierea codului sursă (al aplicaţiei) acesta poate fi compilat, obţinându-se astfel un fişier binar cu extensia .bin ce poate fi salvat şi apoi executat.

3.1. Generalităţi despre Emu8086

Structura de bază a unui computer este prezentată în figura 3.2; se pot observa arhitectura şi componentele principale ale unui sistem de calcul.

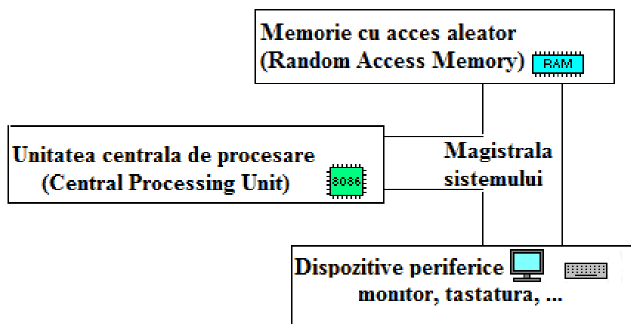


Fig.3.2. Arhitectură tipică de computer

Unitatea centrală de procesare (CPU) este "creierul" computerului. Toate calculele, deciziile şi mişcările datelor se realizează aici. CPU are în componenţă locaţii de stocare specifice numite *regiştri* şi *o unitate aritmetică şi logică* (ALU) unde se realizează procesările. Datele sunt luate din regiştri, procesate, iar rezultatele sunt stocate tot în regiştri. Există mai multe instrucţiuni, fiecare având un scop precis; colecţia tuturor instrucţiunilor se numeşte ***set de instrucţiuni***.

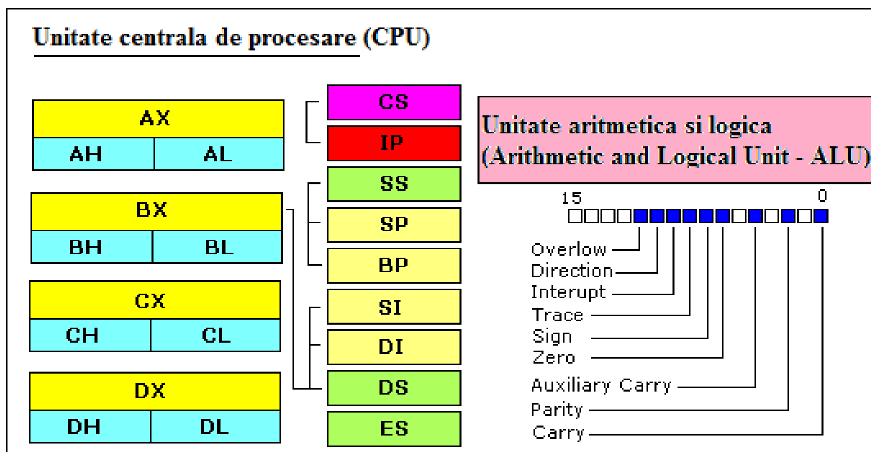


Fig.3.3. Unitatea centrală de procesare

CPU a familiei 8086 are 8 regiştrii de uz general, după cum urmează:

- **AX** - registru *acumulator* (divizat în **AH / AL**).
- **BX** – registru adresă de *bază* (divizat în **BH / BL**).
- **CX** – registru *contor* sau numărător (divizat în **CH / CL**).
- **DX** – registru de *date* (divizat în **DH / DL**).

- **SI** – registru *index sursă*.
- **DI** – registru *index destinaţie*.
- **BP** – pointer la *baza stivei*.
- **SP** – pointer la *stivă (adresa curentă)*.

Regiştrii de uz general de date sunt **AX, BX, CX, DX**; mărimea lor fiind de 16 biţi, ei pot păstra numere fără semn în domeniul 0..+65535 sau numere cu semn în domeniul –32768..+32767. Aceştia se folosesc ca locaţii temporare, mai degrabă decât ca locaţii de memorie, pentru că accesul la regiştrii este mai rapid. Regiştrii AL, BL, CL, DL şi AH, BH, CH, DH sunt părţile low şi high ale regiştrilor corespunzători pe 16 biţi.
*Regiştrii **DI** şi **SI*** sunt regiştri index (Destination Index şi Source Index) destinaţi lucrului cu şiruri de octeţi sau cuvinte.
*Regiştrii **SP** şi **BP*** sunt regiştri destinaţi lucrului cu stiva. Stiva se defineşte ca o zonă de memorie (LIFO – Last In First Out) în care pot fi depuse valori, extragerea lor ulterioară realizându-se în ordine inversă depunerii lor.
SP – Stack Pointer – pointează spre ultimul element introdus în stivă, iar
BP – Base Pointer – pointează către baza stivei.

Regiştrii cu scop special sunt **IP** şi **PSW**.
Registrul **IP** conţine adresa instrucţiunii curente care se execută; după ce s-a executat instrucţiunea curentă, IP este incrementat automat pentru a pointa spre instrucţiunea următoare. Instrucţiunile de salt modifică valoarea acestui registru astfel încât execuţia programului se mută la o nouă poziţie. Registrul IP se mai numeşte şi PC (Program Counter).
PSW (Program Status Word) este un registru ce conţine flagurile (1 bit fiecare) ce raportează starea CPU după execuţia fiecărei instrucţiuni. Aceşti regiştri cu scop special nu pot fi accesaţi direct, ei fiind modificaţi de către CPU pe durata execuţiei instrucţiunii.

Principalii indicatori de stare sunt: **C, S, O, Z**:
C (Carry) indică un transport în afara domeniului de reprezentare al rezultatului;
S (Sign) are valoarea 1 când rezultatul ultimei operaţii este un număr strict negativ, deci copiază bitul MSB al rezultatului;
O (Overflow) indică depăşire de gamă: dacă rezultatul ultimei instrucţiuni a depăşit spaţiul rezervat rezultatului, în cazul operanzilor consideraţi numere cu semn;
Z (Zero) are valoarea 1 dacă rezultatul ultimei instrucţiuni este egal cu zero.

Procesorul 8086 conţine 4 regiştri segment (**CS, DS, ES** şi **SS**), pe 16 biţi fiecare. Aceştia se folosesc pentru a selecta blocuri (numite segmente) din memorie. Regiştrii CS (Code Segment), DS (Data Segment), SS (Stack Segment) şi ES (Extra Segment) din BIU reţin adresele de început ale segmentelor active, corespunzătoare fiecărui tip de segment.

CPU poate accesa până la 1 MB de memorie RAM, adresele RAM fiind date uzual între paranteze drepte; de exemplu [7Ch] se citeşte ca “datele de la locaţia cu adresa 7Ch”, unde 7Ch este un număr hexazecimal.

Magistralele sunt seturi de linii folosite pentru transportul semnalelor. Un computer pe 16 biţi are uzual regiştri pe 16 biţi şi 16 fire/linii într-un bus (magistrală).
Bus-ul de date se foloseşte pentru transportul datelor între CPU, RAM şi porturile I/O. Simulatorul are un bus de date de 16 biţi.
Bus-ul de adrese se foloseşte pentru a specifica ce adresă RAM sau port I/O va fi folosit. Simulatorul are un bus de adrese de 16 biţi.
Bus-ul de control are o linie pentru a determina dacă se accesează RAM sau porturi I/O; are de asemenea o linie pentru a determina dacă datele sunt scrise sau citite: CPU citeşte date când acestea intră în CPU şi scrie date când acestea ies din CPU către RAM sau porturi.
Ceasul sistem constă în impulsuri periodice generate astfel încât componentele să se sincronizeze. Simulatorul lucrează cu o viteză de cateva instrucţiuni pe secundă, ajustabilă în limite mici.

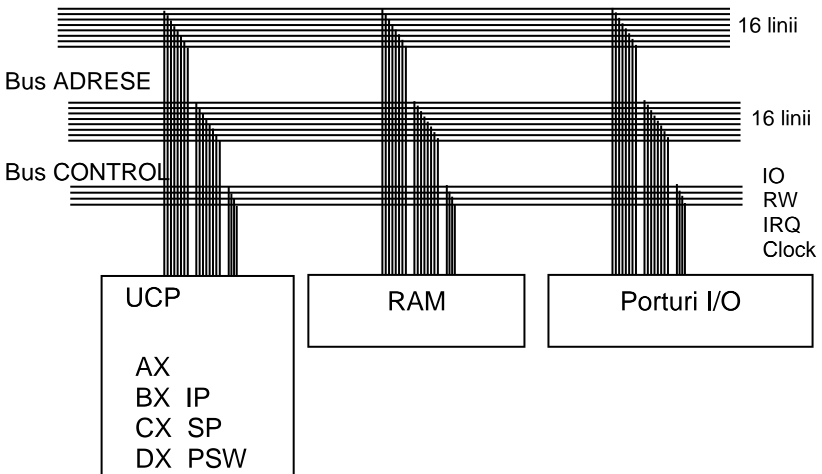


Fig. 3.4. Arhitectură tipică de calculator

3.2. Caracteristicile emulatorului:

- CPU de 16 biţi
- Pe 16 biţi se pot adresa 2^{16} porturi I/O;
- Periferice simulate pe unele dintre aceste porturi
- Asamblor
- Help on-line

- Rulare pas cu pas a programului
- Rulare continuă a programului
- Posibilitatea de modificare a ceasului procesor

Utilizarea emulatorului

La pornirea emulatorului, va apărea o fereastră asemănătoare celei din figura 3.5, în care se poate observa zona de editare a programului sursă (fişier de tip asm), iar în partea de sus meniul cu opţiunile: file, edit, bookmarks, assembler, emulator, math, ascii codes, help.

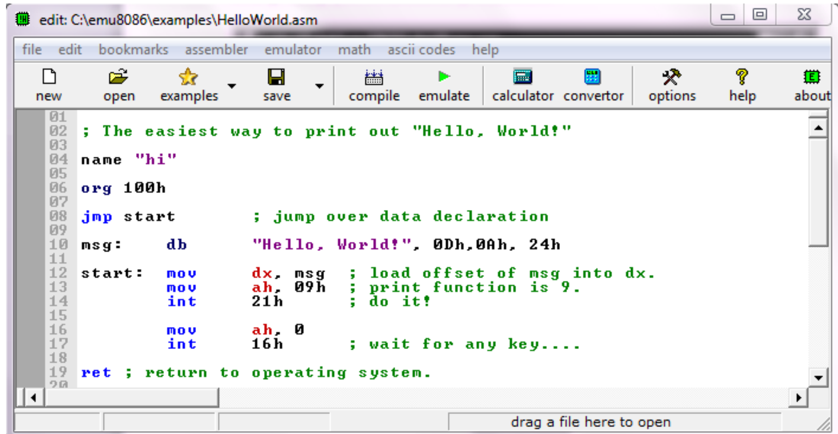
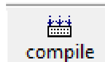


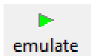
Fig.3.5. Fereastra de editare a emulatorului

Pentru a scrie şi a rula un program nou folosind emulatorul, se va folosi optiunea „new" şi se va selecta un şablon de tip COM, aşa cum se poate urmări în figura 3.6.

Codul scris se numeşte *limbaj de asamblare* (*.asm), iar trecerea lui într-un limbaj care să fie înţeles de către CPU se obţine prin compilare, cu opţiunea



În urma acestei operaţii, va rezulta un fişier de tip .com ce se va salva local, iar dacă se doreşte încărcarea codului în emulator, se va folosi butonul



După încărcarea programului în emulator, va apare fereastra din figura 3.7 în care se pot urmări în partea de jos, dinspre stânga spre dreapta, în cele 3 zone distincte:

1) regiştrii emulatorului,

2) adresa fizică pe 20 de biţi (exprimată cu 5 cifre hexazecimale) şi conţinutul de la acea adresă (specificat în hexazecimal, zecimal şi Ascii)

3) codul sursă al programului în limbaj maşină, înainte de a fi asamblat.

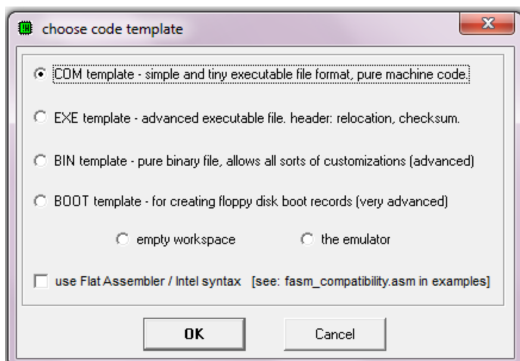


Fig. 3.6. Alegerea şablonului pentru un nou program

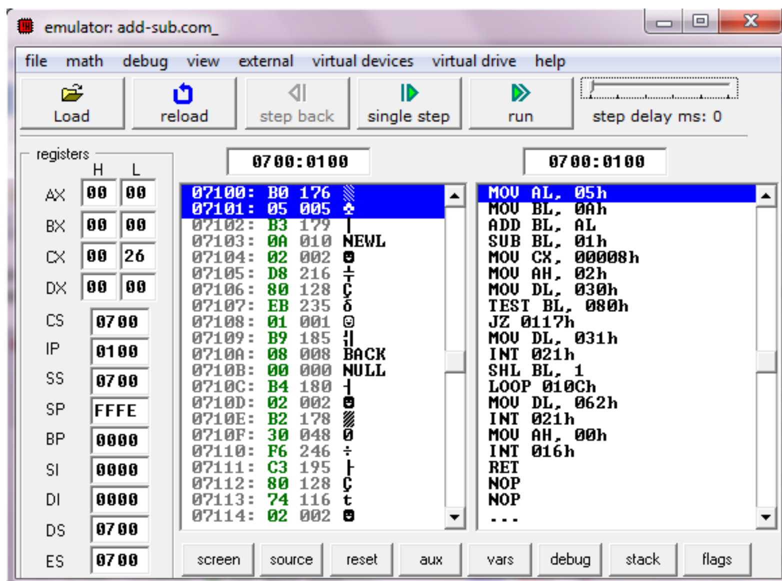
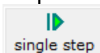
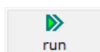


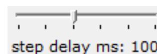
Fig.3.7. Incărcarea unui fişier în emulator

Rularea unui program se poate face pas cu pas, apăsând în mod repetat



sau în mod continuu, cu



Viteza de execuţie poate fi selectată cu ajutorul cursorului 

In figura 3.7, dacă se va da dubluclick în caseta corespunzătoare regiştrilor, se va deschide o nouă fereastră *Extended value viewer*, ce conţine valoarea acelui registru exprimată în hexazecimal, binar şi zecimal. Prin intermediul acestei ferestre, valoarea din registru poate fi modificată direct, în timpul rulării. O operaţie de dubluclick asupra unei zone din memorie, va lista cuvântul din memorie aflat la locaţia selectată. Reamintim că octetul cel mai puţin semnificativ se găseşte la adrese mai mici (conform convenţiei *Little END-ian*).

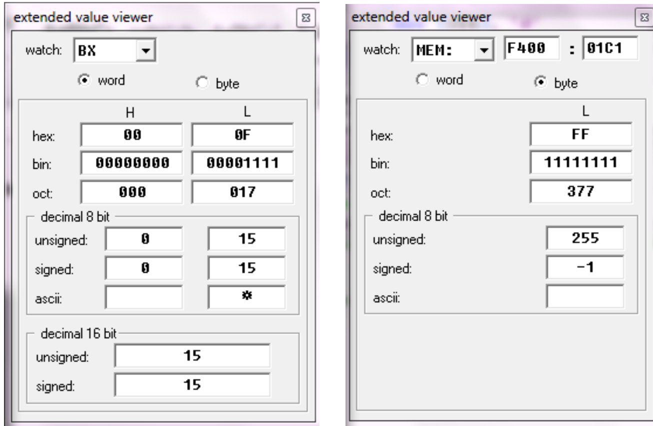


Fig.3.8. Fereastra de vizualizare şi modificare a regiştrilor (stânga) sau a unei zone din memorie (dreapta)

Emulatorul rulează programele pe un computer virtual, fapt care blochează accesarea hardware-ului real, precum driverele hard şi memoria.

În urma execuţiei fiecărei instrucţiuni, putem urmări modificarea conţinutului regiştrilor din CPU. De asemenea, există posibilitatea de a vizualiza conţinutul memoriei sau al ALU, valorile flag-urilor, aşa cum se poate urmări în figurile 3.9 şi 3.10. Selectarea informaţiei dorite spre vizualizare se realizează din meniul View al emulatorului (figura 3.7, sus)

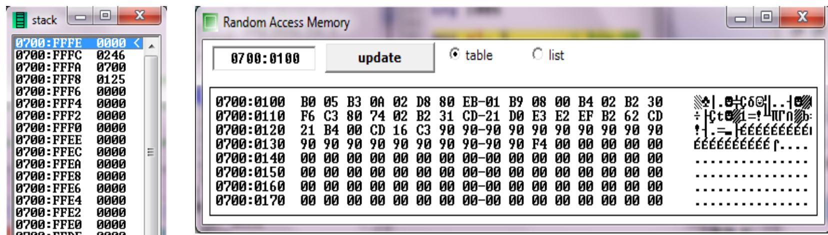


Fig.3.9. Ferestre de vizualizare a conţinutului stivei şi al memoriei

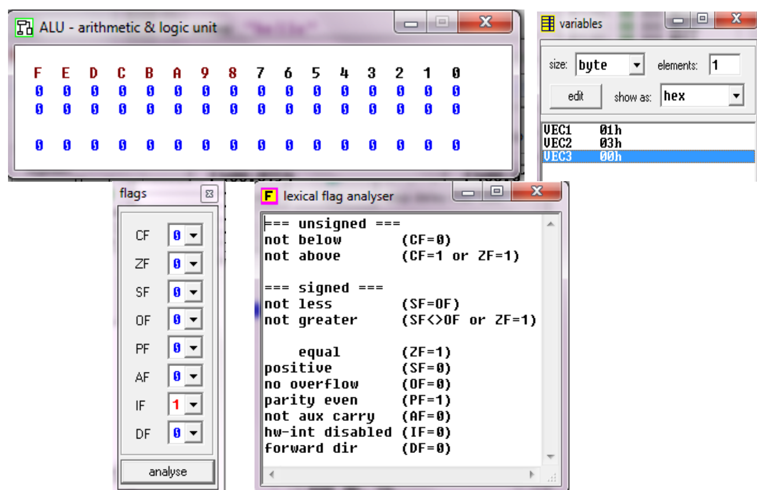


Fig.3.10. Ferestre de vizualizare a conţinutului ALU, al variabilelor definite în memorie şi al flag-urilor

Ecranul emulatorului (obţinut tot din meniul View, fereastra din figura 3.7.) poate fi folosit pentru datele de ieşire, fiind suportat şi modul color.

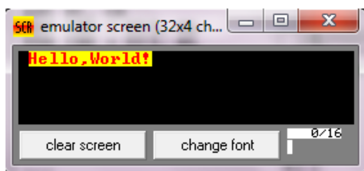


Fig.3.11. Fereastra de vizualizare a ecranului

Din meniul Math, se pot deschide ferestrele din fig. 3.12, corespunzătoare:
- calculatorului (*expression evaluator*) ce poate fi folosit pentru operaţii logice şi aritmetice cu valori hexazecimale, octale, binare şi zecimale şi resp.
- convertorului (*base convertor*), prin care numerele pot fi convertite dintr-o bază de numeraţie în alta.

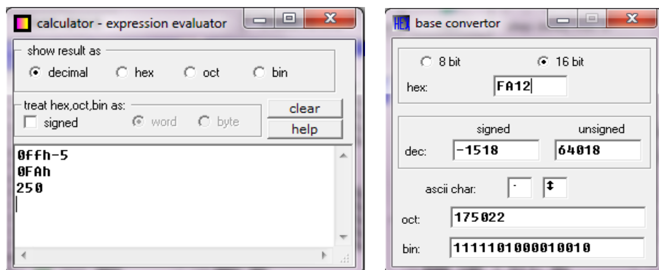


Fig.3.12. Ferestrele calculatorului şi convertorului

Emu8086 include câteva dispozitive virtuale cu ajutorul cărora se pot realiza experimente: acestea includ un termometru, un ecran virtual, un sistem semafor, un motor pas cu pas, un robot, un afişaj cu cristale lichide, o imprimantă şi altele, cu interfeţe aşa cum se poate urmări în figura 3.13.

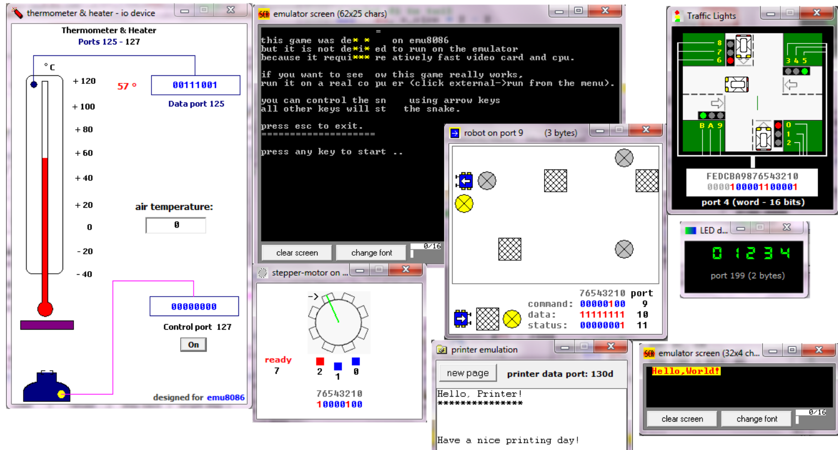


Fig.3.13. Dispozitive virtuale existente în emulator

Exemplele prezentate în tutorial reprezintă o introducere pas cu pas în comenzile şi tehnicile de programare de nivel scăzut.

3.3 Aplicaţii

Fiecare program are unul sau mai multe exerciţii asociate, unele dintre ele simple, altele mai complexe.

3.3.1 Operaţii aritmetice simple: **2_sample.asm**

Programul **2_sample.asm** foloseşte instrucţiunile MOV, ADD, SUB. Pentru a rula programul apăsaţi Single Step până la terminarea programului. Observaţi modificările din regiştri în timpul rulării pas cu pas. În momentul modificării valorii într-un registru, acest lucru este semnalat prin colorare în albastru.

Observaţii:

Comentarii – orice text aflat după “;” nu este parte a programului şi este ignorat de către simulator. Comentariile se folosesc pentru a explica ce face programul. Un bun programator foloseşte multe comentarii, iar acestea nu trebuie să repete pur şi simplu codul, ci să furnizeze explicaţii suplimentare.

Directiva *name „nume”:* name "add-sub" este folosită pentru a specifica numele fişierului de tip .com ce se va crea în urma compilării.

Directiva *org valoare:* org 100h se foloseşte pentru a specifica adresa la care se va asambla programul. Valoarea 100h este 256 în zecimal.

MOV – este prescurtarea pentru operaţia de mutare MOVE. În acest exemplu, numerele sunt copiate în regiştri pentru a putea realiza operaţia aritmetică. MOV copiază datele dintr-o locaţie în alta, datele de la locaţia iniţială rămânând nemodificate.
Aritmetica – ADD se foloseşte pentru a aduna doi operanzi, în cazul de faţă conţinutul a doi regiştri. O altă versiune este folosirea sa pentru a aduna un număr la un registru. Rezultatul adunării se va depune în operandul destinaţie, cel din stânga. SUB se foloseşte pentru a scădea conţinutul a doi regiştri: din operandul din stânga se scade cel din dreapta, rezultatul fiind depus apoi în cel din stânga.
END – dacă apare în program orice text după această directivă va fi ignorat, END fiind ultima comandă, specifică emulatorului.

Programul **2_sample.asm** este prezentat în continuare:

```
name "add-sub"
org 100h
;______CALCULEAZĂ 5 PLUS 10 ŞI 15 MINUS 1____________________

       MOV AL, 5      ;copiază în reg. AL valoarea 5=00000101b
       MOV BL, 10     ;copiază în reg. BL valoarea 10=00001010b
       ADD BL, AL     ;5+10=15 (000Fh) valoare stocată în BL
       SUB BL, 1      ;15-1=14 (000Eh) valoare stocată în BL
```

Programul continuă cu instrucţiuni necesare afişării rezultatului în binar, care vor fi ignorate în lucrarea de faţă şi se vor studia într-o lucrare ulterioară. Pentru a studia aritmetica, se vor consulta valorile regiştrilor aşa cum se sugerează în figura 3.8. Pentru a urmări rezultatele obţinute, se poate consulta conţinutul ALU, aşa cum arată figura 3.9. De asemenea, se pot folosi calculatorul şi convertorul din figura 3.11 pentru verificarea corectitudinii calculelor.

Exerciţii şi teme (I)

1. Scrieţi un program care realizează scăderea a două numere, folosind instrucţiunea SUB;
2. Scrieţi un program care realizează înmulţirea a două numere, folosind instrucţiunea MUL;
3. Scrieţi un program care realizează împărţirea a două numere, folosind instrucţiunea DIV;
4. Scrieţi un program care realizează împărţirea la zero. Amintiţi-vă să nu faceţi această împărţire altă dată.

Majoritatea programelor prezentate includ o parte de exerciţii pentru învăţare, pentru a garanta înţelegerea exemplului. În acest caz, puteţi folosi toţi regiştrii de 8 biţi sau de 16 biţi şi operaţiile ADD, SUB, MUL şi DIV. Studiaţi aceste instrucţiuni (din Help-ul simulatorului) înainte de a le utiliza. Observaţi ce se întâmplă dacă încercaţi să faceţi împărţirea la zero.

3.3.2. Intrări şi ieşiri de date: **simple_io.asm**

Programul **simple_io.asm** se studiază pentru a arăta cum se pot accesa porturile (virtuale), având adresele posibile de la 0 la 0FFFFh. Programul foloseşte instrucţiuni specifice porturilor de ieşire (OUT) şi de intrare (IN) şi date de dimensiuni diferite: când intervine registrul AL, datele sunt pe octet, iar când se foloseşte registrul AX, datele sunt pe cuvânt. Pentru a rula programul, apăsaţi Single Step până la terminarea programului. Urmăriţi fereastra corespunzătoare portului virtual:

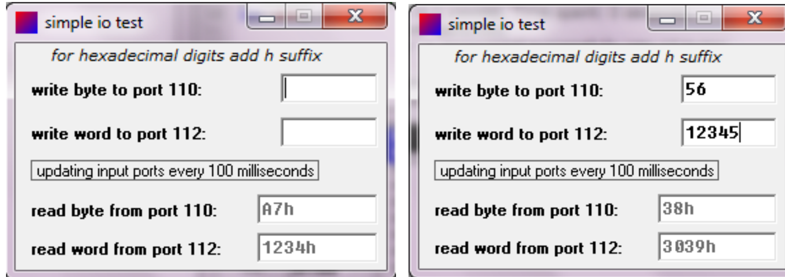


Fig.3.14. Interfaţa aplicaţiei Simple_io înainte şi după introducerea datelor de la utilizator

Observaţii:

OUT val,AL sau OUT val,AX – această instrucţiune trimite conţinutul registrului AL sau AX (deci octet sau cuvânt), la portul de ieşire cu adresa *val*. Circuitul virtual este legat pe portul de ieşire 110 (pentru octet) sau 112 (pentru cuvânt).

IN AL, val sau *IN AX, val* – preia un octet sau un cuvânt de la portul cu adresa *val*. Emulatorul aşteaptă introducerea unui octet sau cuvânt şi scrie (eventual şi transformă) valoarea în hexazecimal în registrul AL sau AX.

Programul **simple_io.asm** este prezentat în continuare:

```
#start=simple.exe#
#make_bin#
name "simple"

;______CITESTE OCTET şi CUVANT DE LA PORT__________________
       MOV AL, 0A7h    ;scrie octetul 0A7h la portul 110
       OUT 110, AL

       MOV AX, 1234h   ;scrie cuvântul de valoare 1234h la
       OUT 112, AX     ;portul cu adresa 112

       MOV AX, 0       ;reset registru.

       IN AL, 110      ;citeşte în AL octet de la portul 110
       IN AX, 112      ;citeşte în AX cuvant de la portul 112
```

Exerciţii şi teme (II)

1. Modificaţi programul a.î. valoarea implicită pe port pentru octet să fie 12h.
2. Modificaţi programul astfel încât valoarea implicită pe port pentru cuvânt să fie 3456h.

3.3.3. Utilizarea numerelor hexazecimale: **traffic_lights2.asm**

Programul **traffic_lights2.asm** foloseşte instrucţiunile MOV, OUT, ROL şi JMP. Luminile semafoarelor sunt controlate prin trimiterea datelor la portul 4, deci se va folosi instructiunea *OUT 4, AX*.

Avem de controlat 12 becuri in figura 3.15: roşu, galben, verde (în această ordine) pentru cele 4 semafoare (primul-cel de jos, al doilea-cel din dreapta, al treilea-cel de sus, al patrulea-cel din stânga). Acest lucru poate fi realizat cu ajutorul a doi octeţi, deci 16 biţi, din care nu vom folosi cei mai semnificativi 4 biţi. Prin setarea unui bit la 1, becul corespunzător se aprinde.

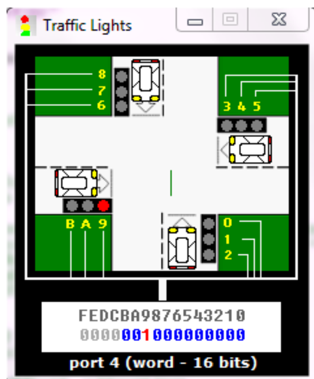


Fig.3.15. Interfaţa aplicaţiei semafor (Traffic Lights)

Observaţii:

Etichete şi instrucţiunea JMP – etichetele marchează în program poziţii care vor fi folosite de comenzile de salt. În acest program, toate instrucţiunile sunt repetate la infinit sau până la apăsarea butonului Stop. Numele de etichete trebuie să înceapă cu o literă sau cu caracterul “_”, în nici un caz cu o cifră.

O instrucţiune de genul *JMP start* va cauza un salt în program la eticheta *start* şi reluarea instrucţiunilor cuprinse între cele 2. Eticheta destinaţie a saltului se încheie cu “:”, de exemplu *start*:

OUT 4, AX – această instrucţiune copiază conţinutul registrului AX la portul de ieşire 4, unde este legat circuitul de semafoare. Un 1 binar are ca efect aprinderea becului, iar un 0 stingerea lui.

Controlul becurilor – se poate observa din figura 3.15 care este becul (din cele 12) controlat de fiecare bit. Prin setarea bitului corespunzător (binar) şi transformarea în hexazecimal a întregului număr, se poate schimba modul de funcţionare al semafoarelor.

Directiva *EQU* este folosită pentru a defini constante, de exemplu *red EQU 0000_0001b* defineşte constanta red cu valoarea binară 0000_0001b.

Instrucţiunea *nop* provine de la *no operation* şi introduce o întârziere în prelucrarea datelor.

ROL – bitul MSb trece atât în C (Carry) cât şi în bitul LSb din operand.

Instructiunea *ROL* este de fapt cea care se execută atunci când apare semnul „<<”. De exemplu: *MOV AX, green << 3* va încărca în registrul AX valoarea constantei green, deplasată spre stânga cu 3 poziţii; deci 0000_0000_0000_0100b va deveni 0000_0000_0010_0000b; astfel, bitul 5 va deveni 1, adică becul verde de la al doilea semafor va fi aprins.

Instrucţiunea jmp start asigură rularea (execuţia) continuă a programului.

```
#start=Traffic_Lights.exe#
name "traffic2"

;____CONTROLUL BECURILOR SEMAFORULUI _2________
       yellow_and_green      equ      0000_0110b
       red                   equ      0000_0001b
       yellow_and_red        equ      0000_0011b
       green                 equ      0000_0100b
       all_red               equ      0010_0100_1001b

start:
       nop

       mov ax, green                 ; controlează biţii 0,1,2
       out 4, ax
       mov ax, yellow_and_green
       out 4, ax
       mov ax,red
       out 4, ax
       mov ax, yellow_and_red
       out 4, ax

       mov ax, green << 3            ; controlează biţii 3,4,5
       out 4, ax
       mov ax, yellow_and_green << 3
       out 4, ax
       mov ax,red << 3
       out 4, ax
       mov ax, yellow_and_red << 3
       out 4, ax
```

```
        mov ax, green << 6            ; controlează biţii 6,7,8
        out 4, ax
        mov ax, yellow_and_green << 6
        out 4, ax
        mov ax,red << 6
        out 4, ax
        mov ax, yellow_and_red << 6
        out 4, ax
        mov ax, green << 9            ; controlează biţii 9,A,B
        out 4, ax
        mov ax, yellow_and_green << 9
        out 4, ax
        mov ax,red << 9
        out 4, ax
        mov ax, yellow_and_red << 9
        out 4, ax

        mov ax, all_red        ;aprinde toate becurile roşii
        out 4, ax
        mov ax, all_red << 1   ;aprinde toate becurile galbene
        out 4, ax
        mov ax, all_red << 2   ;aprinde toate becurile verzi
        out 4, ax
jmp start
```

Exerciţii şi teme (III)

1. Modificaţi programul astfel încât secvenţa de funcţionare a becurilor semaforului să fie realistă; se vor da comenzi astfel (într-un singur program):

1 - se vor aprinde becurile roşii de la semafoarele 2 şi 4, împreună cu becurile verzi de la semafoarele 1 şi 3, apoi

2 - se vor aprinde becurile roşu şi galben de la semafoarele 2 şi 4, împreună cu becurile galbene de la semafoarele 1 şi 3, apoi

3 - se vor aprinde becurile verzi de la semafoarele 2 şi 4, împreună cu becurile roşii de la semafoarele 1 şi 3, apoi

4 - se vor aprinde becurile galbene de la semafoarele 2 şi 4, împreună cu becurile roşii şi galbene de la semafoarele 1 şi 3 şi apoi ciclul se reia.

2. Propuneţi o altă funcţionare pentru semafor, astfel încât să nu ducă la ciocniri de autovehicule.

3. Consideraţi introducerea de întârzieri în execuţia secvenţelor de instrucţiuni.