4. Simulator de microprocesor (II)

4.1. Definirea datelor în memorie: **traffic_lights.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 în figura 4.1: roşu, galben, verde (in aceasta ordine) pentru cele 4 semafoare.

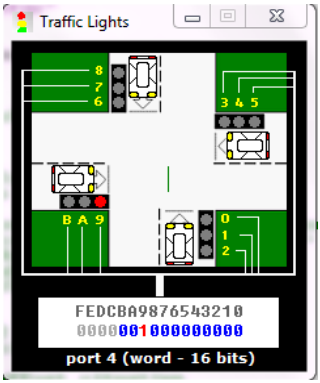

Fig.4.1. Interfaţa aplicaţiei semafor Traffic Lights

O variantă îmbunătăţită a programului **traffic_lights2.asm** prezentată în lucrarea anterioară, poate fi urmărită în programul **traffic_lights.asm,** care este prezentat în continuare:

```
 #start=Traffic_Lights.exe#
name "traffic"
;_______CONTROLUL BECURILOR SEMAFORULUI__________

        MOV AX, all_red
        OUT 4, AX
        MOV SI, offset situation
next:   MOV AX, [SI]
        OUT 4, AX

        ; secvenţa necesară introducerii unei întârzieri:
        MOV     CX, 4CH         ; asteaptă 5 secunde
        MOV     DX, 4B40H       ; 004C4B40h = 5,000,000
        MOV     AH, 86H
        INT     15H
        ADD SI, 2               ; trece la situaţia următoare
        CMP SI, sit_end         ; se verifică dacă s-a ajuns la
                                ; sfârşit
        JB  next                ; dacă nu, sare la eticheta next
        MOV SI, OFFSET situation  ;dacă da, reia de la început
        JMP next                ; salt necondiţionat la et.next
```

```
                                ;FEDC_BA98_7654_3210
situation       dw      0000_0011_0000_1100b
s1              dw      0000_0110_1001_1010b
s2              dw      0000_1000_0110_0001b
s3              dw      0000_1000_0110_0001b
s4              dw      0000_0100_1101_0011b
sit_end = $
all_red         equ     0000_0010_0100_1001b
```

Observaţii:

Programul foloseşte un tabel de date, creat prin intermediul directivei dw (define word). Secvenţa de mai sus defineşte *o zonă de memorie*, începând de la cuvântul adresabil prin numele „situation” de valoare 030Ch, urmat apoi de cuvântul s1 de valoare 069Ah, apoi s2 de valoare 0861h şi aşa mai departe, încheindu-se cu caracterul „$”.

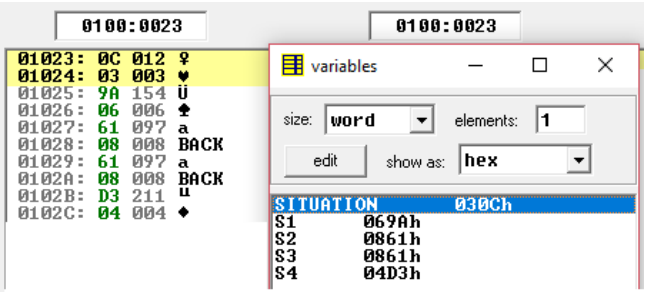

Fig.4.2. Vizualizarea zonei de memorie pentru aplicaţia semafor

Exerciţii şi teme (I)

1. Analizaţi programul şi variabilele definite în memorie. Urmăriţi funcţionarea semaforului, executând programul cu „run”.

2. Modificaţi variabilele din memorie astfel încât secvenţa de funcţionare a becurilor semaforului să respecte următorul algoritm: un singur semafor va afişa culoarea verde; semaforul opus celui „activat” pe verde, va afişa culoarea galben, iar celelalte 2 semafoare adiacente vor afişa culoarea roşu; apoi, următorul semafor va funcţiona după algoritmul propus şi tot aşa, pe rând, până la funcţionarea fiecăruia din cele 4 semafoare după algoritmul propus. Bucla se reia apoi şi continuă până la apăsarea unei taste, când se asigură ieşirea din program.

3. Propuneţi o secvenţă care să fie formată din cel puţin încă 2 variabile în plus. Observăti aceste variabile în memorie, în ambele moduri, aşa cum se sugerează în figura 4.2.

4.2. Instrucţiuni de comparare: meniul **examples**, **compare numbers**

Instrucţiunea **CMP op1,op2** funcţionează în felul următor: procesorul calculează diferenţa op1-op2 şi în funcţie de rezultat, setează flagurile (indicatorii):
Z dacă rezultatul este zero (deci dacă dacă op1=op2),
S dacă rezultatul este negativ (deci dacă op1<op2),
iar dacă op1 este mai mare decât op2 nu se setează nici unul.

Flagul C (Carry) indică un transport în afara domeniului de reprezentare a rezultatului, iar flagul A (Auxiliary) indică valoarea transportului de la bitul 3 la bitul 4 (între cifrele hexazecimale).
Se sugerează revenirea la lucrarea 2 în vederea revizuirii definiţiei flagurilor aritmetice şi a exemplelor prezentate.
Pentru vizualizarea flagurilor, din meniul View al ferestrei corespunzătoare fişierului de tip .com se va selecta *flags*, iar pentru vizualizarea rezultatului operaţiei de scădere se poate consulta conţinutul ALU din cadrul aceluiaşi meniu. Pentru o mai bună înţelegere a exemplelor prezentate, se va urmări şi fereastra *lexical flag analyzer*.

```
; 4 este egal cu 4
mov ah, 4       ; AH=4
mov al, 4       ; AL=4
cmp ah, al   ;AH-AL=0, deci Z=1, iar S=0, C=0

; (cu semn/ fără semn)
; 4 este mai mare (greater/above) decât 3
mov ah, 4       ;AH=4
mov al, 3       ;AL=3
cmp ah, al      ;AH-AL=1, deci Z=0, iar S=0, C=0

; (cu semn)
; 1 este mai mare (greater) decât -5
mov ah, 1       ; AH=1
mov al, -5      ; AL=-5 = 251 = 0fbh
cmp ah, al      ; AH-AL=1-(-5)=6, deci trebuie interpretat ca o
                ; scădere:
                        0000 0001b-
                        1111 1011b
                        0000 0110b, C=1, A=1, Z=0, S=0

; (fără semn)
; 1 este mai mic (below) decat 251
mov ah, 1       ; AH=1
mov al, 251     ; AL=251
cmp ah, al      ; AH-AL=1-251=6 trebuie interpretat ca o scădere
                ; cu împrumut
                        0000 0001b-
                        1111 1011b
                         0000 0110b, C=1, A=1, Z=0, S=0
```

```
; (cu semn)
; -3 este mai mic (less) decât -2
mov ah, -3      ; AH=-3
mov al, -2      ; AL=-2
cmp ah, al      ; AH-AL=-3-(-2)=-3+2=-1=0FFh, O=0,S=1,Z=0,A=1

; (cu semn)
; -2 este mai mare(greater) decât -3
mov ah, -2      ; AH=-2
mov al, -3      ; AL=-3
cmp ah, al      ; AH-AL=-2+3=+1=01h, O=0,S=0,Z=0,A=0

; (fără semn)
; 255 este mai mare(above) decât 1
mov ah, 255     ; AH=255=0FFh
mov al, 1       ; AL=1
cmp ah, al      ; AH-AL=254=0FEh, S=1, Z=0, C=0
```

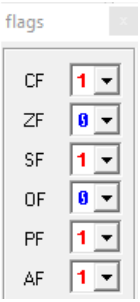

Fig.4.3 Vizualizarea flagurilor aritmetice pentru cel de-al 5-lea exemplu

Exerciţii şi teme (II)

Analizaţi ultimele 3 exemple individual, efectuând operaţiile în binar şi specificaţi valorile flagurilor şi abia apoi rulaţi exemplele cu emulatorul.

4.3. Folosirea întreruperilor: **z02.asm**

Întreruperea este un semnal transmis sistemului de calcul prin care acesta este anunţat de apariţia unui eveniment care necesită atenţie. În BIOS sunt scrise o serie de subrutine legate de echipamentele periferice ale sistemului, apelarea lor într-o aplicaţie făcându-se prin întreruperi, cu instrucţiunea **INT**, cu sintaxa **INT n**. O întrerupere poate avea mai multe servicii asociate, serviciul selectându-se prin încărcarea în registrul AH a unui număr specific acelui serviciu, înainte de apelarea întreruperii. Alţi parametri de apel ai serviciului se încarcă în anumiţi regiştri, după caz, aşa cum se va vedea într-o lucrare ulterioară, în care se vor studia întreruperile mai pe larg.

Aplicaţia de faţă foloseşte întreruperea INT 21h cu serviciul 09h, care afişează un şir de caractere aflat la locaţia adresată de DS:DX, şir care trebuie să se termine cu caracterul $. (a se consulta documentaţia, la secţiunea Tutoriale, lista întreruperilor).

```
.stack 64h      ; segmentul de stivă
.data           ; segmentul de date

msg db "Hello, World", 24h    ;se defineşte un şir de octeţi,
;în segmentul de date, putând fi identificat prin numele
;variabilei msg şi fiind iniţializat cu: codul Ascii al
;caracterului ‚H', urmat de codul Ascii al caracterului ‚e',
;s.a.m.d.; şirul se termină cu caracterul ce are codul Ascii
;24h, adică $.
.code
    mov ax, @data
    mov ds, ax

    mov dx, offset msg
    mov ah, 9
    int 21h
.exit
```

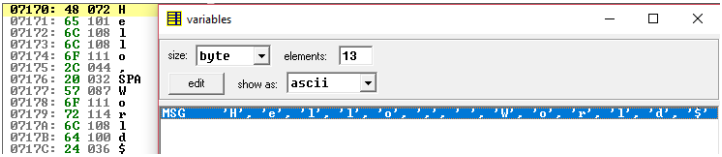

Fig.4.4 Vizualizarea variabilei de tip şir în memorie

Exerciţii şi teme (III)
Analizaţi lista întreruperilor din tutorial şi propuneţi o altă metodă de afişare a unui şir de caractere pe ecran (folosind o altă întrerupere şi/sau serviciu).

4.4. Citirea unor date de la tastatură: **keybrd.asm**

Programul **keybrd.asm** foloseşte instrucţiunile IN, CMP, JNZ, JZ şi ilustrează folosirea funcţiilor tastaturii. Aplicaţia foloseşte bufferul tastaturii (de 16 biţi, vizualizat jos în fereastră, lângă opţiunea change font) atunci când se tipăreşte foarte repede. Codul aplicaţiei se repetă în buclă până la apăsarea tastei „esc", orice alt caracter fiind afişat pe ecran (Figura 4.5).

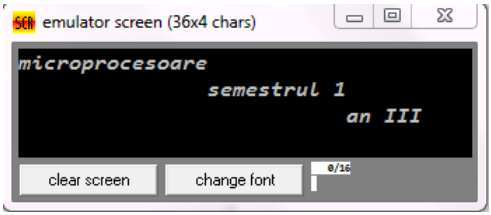

Fig.4.5 Fereastra aplicaţiei keybrd.asm

Observatii:
CMP AL,1Bh – compară conţinutul registrului AL cu codul ASCII al tastei Esc (codul ASCII al tastei Esc este 1Bh).

JZ Rep – JZ vine de la Jump if Zero, adică salt dacă flagul Z este setat. Programul va face un salt la adresa marcată de eticheta Rep. O instrucţiune asemănătoare este JNZ, adică Jump if Not Zero, în cazul în care flagul Z nu este setat. În acest program, instrucţiunea CMP setează flag-urile. Instrucţiunile aritmetice setează de asemenea stările flag-urilor.

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

```
name "keybrd"
org     100h

;_______INTRARE DE LA TASTATURA__________
mov dx, offset msg     ; afişează mesaj de primire
       mov ah, 9
       int 21h
;============================
wait_for_key:                  ; buclă infinită pt preluarea şi
                               ; afişarea tastelor
       mov   ah, 1             ; verifică dacă există tastă
                               ; nepreluată din buffer
       int     16h
   jz   wait_for_key
       mov     ah, 0           ; preia tasta de la tastatură,
                               ; ştergând-o din buffer
       int     16h
       mov     ah, 0eh         ; afisează pe ecran tasta
       int     10h
       cmp     al, 1bh         ; se verifică dacă s-a apăsat
                               ; 'esc' pentru a ieşi
       jz      exit
       jmp  wait_for_key
;============================
exit:    ret
msg  db "Type anything...", 0Dh,0Ah
     db "[Enter] - carriage return.", 0Dh,0Ah
     db "[Ctrl]+[Enter] - line feed.", 0Dh,0Ah
     db "You may hear a beep", 0Dh,0Ah
     db "    when buffer is overflown.", 0Dh,0Ah
     db "Press Esc to exit.", 0Dh,0Ah, "$"
end
```

Exerciţii şi teme (VI)
1. Rulaţi programul keybrd.asm setând viteza maximă la execuţie (întârziere de 0 msec), dar şi la o viteză mai mică (de exemplu întârziere de 200 msec) din cursorul pentru timp. Urmăriţi ambele situaţii.

2. Tastaţi cât mai repede cu putinţă, introducând mai mult de 16 caractere. Ce observaţi?