Accesul la resursele hardware prin întreruperi

Dacă în limbajele de nivel înalt (de exemplu C) se interacţionează cu utilizatorul prin intermediul unor funcţii predefinite precum scanf, printf, etc, aici, în limbajele de nivel scăzut, (în special în limbajul procesoarelor de 16 biți din familia x86), lucrurile nu stau așa simplu. Fiecare acţiune întreprinsă la tastatură sau la ecran, va trebui comandată din program prin intermediul unei întreruperi. Exact așa cum în cadrul programului în C când se întâlnește funcţia scanf programul nu merge mai departe, ci așteaptă apăsarea unei taste, exact la fel se tratează în limbajul de asamblare așa numitele întreruperi ecran și întreruperi tastatură cu serviciile lor. Se prezintă în continuare un program simplu scris în C versus asm, pentru o analogie rapidă. Programul care afișează un mesaj „Mesaj afisat" pe ecran și coboară cursorul pe linia următoare, în C se scrie:

```
#include<stdio.h>
void main()
{
 printf("Mesaj afisat \n");
  }
```

iar programul echivalent scris în limbaj de asamblare este:

```
; afișare pe bază de cunoaștere a unui caracter ca terminator de șir, cazul cu '$'
org 100h        ; p1_01              ; va rezulta un program executabil de tip com
.data
msg db "Mesaj afisat!",13,10, "$"   ; definirea mesajului în memorie
.code
mov dx, offset msg                   ; se încarcă offsetul șirului msg în registrul DX
mov ah, 09h                          ; serviciul de afișare al întreruperii cu tipul 21h
int 21h                              ; se apelează întreruperea int 21h, cu serviciul 09h
ret                                  ; revenire în S.O.
```

Diferenţa între cele două programe este că în limbajul de asamblare trebuie definite separat datele, variabilele, mesajele (nu sunt altceva decât șiruri de caractere) în cadrul unui segment de date (care se specifică folosind directiva *.data*), iar eventualele prelucrări sau instrucţiuni ale programului în zona de cod (precedată de directiva *.code*).
Pentru afișare, în cazul C s-a folosit funcţia predefinită *printf*, iar în cazul asm s-a folosit o întrerupere (serviciul 09h al întreruperii cu tipul 21h), aceasta având o anumită modalitate de apel (se specifică în registrul DX adresa de început în memorie a șirului de caractere ce se dorește a fi afișat pe ecran). Echivalentul lui \n din C pentru a coborî cursorul cu o linie, în limbaj de asamblare este 13,10 adică sunt specificate în clar codurile Ascii ale CR-Carriage Return și LF-LineFeed. La folosirea limbajului de asamblare există avantajul controlării procesului de preluare sau afișare mai îndeaproape, de la un nivel mai apropiat de cel al limbajului mașină folosit de CPU.

Un alt exemplu de program (care preia o tastă (o cifră între 0 și 9) de la tastatură și apoi o afișează pe ecran), scris în două variante:

1) Versiune fără definirea Val în memorie ca variabilă

```
org 100h        ; p1_02
.data

.code

mov ah,0h
int 16h         ; în AL va fi codul Ascii al tastei apăsate de utilizator

;serviciul de afișare caracter
mov ah,02h
mov dl,al       ; se depune în DL caracterul preluat de la tastatură
int 21h         ; apel întrerupere int 21h
ret
```

2) Versiune cu definirea Val în memorie ca variabilă

```
org 100h        ; p1_03
.data
Val db ?
.code
mov bx, offset Val      ; se definește BX ca pointer la zona Val
mov ah,0h
int 16h                 ; în AL va fi codul Ascii al tastei apăsate de utilizator
mov byte ptr [bx], al   ;se depune caracterul preluat în mem.
;serviciul de afișare caracter
mov ah,02h
mov dl,[bx]     ; se depune în DL elementul pointat de BX din memorie
int 21h         ; apel întrerupere int 21h
ret
```

Programul preia o tastă (indicat a fi o cifră între 0 și 9) de la tastatură și apoi o afișează pe ecran, echivalentul în C fiind dat în continuare:

```
#include <stdio.h>
int main()
{
   int val;
   scanf( "%d", &val)  ;          % preia cifra de la tastatură
   printf( "%d", val);            % afișează cifra pe ecran
   getchar();
   return 0;
}
```

Standardul ASCİİ

Pentru reprezentarea sau codificarea informaţiilor alfanumerice din PC s-a folosit îndelung codificarea standard ***ASCİİ (American Standard Coding for İnformation İnterchange),*** prima utilizare comercială fiind în anul 1963 în telecomunicaţii, apoi adoptat pe scară largă, atât pentru computere cât şi pentru echipamentele înrudinte. ASCİİ este un sistem bazat pe alfabetul englez; varianta standard, codificată pe 7 biţi, cuprinde 128 de caractere text (coduri/simboluri alfanumerice codificate ca valori întregi fără semn, între 0 şi 127 (sau 0h..7Fh)): ***33 neimprimabile*** (majoritatea fiind caractere de control învechite) şi ***95 imprimabile***.

Standardul foloseşte 7 biţi în varianta originală sau 8 biţi în varianta extinsă (numit şi ASCİİ-8) şi poate cuprinde: litere (mari, mici) – cele 26 litere din alfabetul englezesc, cifre zecimale (0...9), simboluri matematice(+-=), semne de punctuaţie (. , ; !), coduri de editare şi formatare a textului (SP-Space, BS-BackSpace, CR-CarriageReturn, LF-LineFeed), coduri de control al transferului de date/text (STX-start of text, ETX-end of text). Varianta extinsă a codului are în plus al 8-lea bit (MSb) care de-a lungul timpului a avut mai multe semnificaţii: întotdeauna ca 0, ca bit de paritate pt asigurarea protecţiei, folosit la extinderea alfabetului ASCİİ de la 128 la 256 simboluri, etc.

Tabel 1.1. Standardul ASCİİ pe 7 biţi

		$b_6b_5b_4$	000	001	010	011	100	101	110	111
Baza 10	*Baza 16*	Binar $b_3b_2b_1b_0$	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0	*0*	**0000**	(ctrl@)NUL	(ctrlP)DLE	**SP**	0	@	P	‘	p
1	*1*	**0001**	(ctrlA)SOH	(ctrlQ)DC1	!	1	**A**	Q	a	q
2	*2*	**0010**	(ctrlB)**STX**	(ctrlR)DC2	“	2	B	R	b	r
3	*3*	**0011**	(ctrlC)**ETX**	(ctrlS)DC3	#	3	C	S	c	s
4	*4*	**0100**	(ctrlD)EOT	(ctrlT)DC4	$	4	D	T	d	t
5	*5*	**0101**	(ctrlE)ENQ	(ctrlU)NAK	%	**5**	E	U	e	u
6	*6*	**0110**	(ctrlF)ACK	(ctrlV)SYN	&	6	F	V	f	v
7	*7*	**0111**	(ctrlG)BEL	(ctrlW)ETB	`	7	G	W	g	w
8	*8*	**1000**	(ctrlH)**BS**	(ctrlX)CAN	(	8	H	X	h	x
9	*9*	**1001**	(ctrlİ)(tab)HT	(ctrlY)EM	)	9	İ	Y	i	y
10	*A*	**1010**	(ctrlJ)**LF**	(ctrlZ)SUB	*	:	J	Z	j	z
11	*B*	**1011**	(ctrlK)VT	(ctrl[)**ESC**	+	;	K	[	k	{
12	*C*	**1100**	(ctrlL)FF	(ctrl\)FS	,	<	L	\	l	|
13	*D*	**1101**	(ctrlM)**CR**	(ctrl])GS	-	=	**M**	]	m	}
14	*E*	**1110**	(ctrlN)SO	(ctrl^)RS	.	>	N	^	n	~
15	*F*	**1111**	(ctrlO)Sİ	(ctrl_)US	/	?	O	_	o	DEL

Setul de caractere ASCİİ, aşa cum arată şi Tabelul 1.1, se poate diviza în 4 grupuri mari a câte 32 caractere (fiecare caracter este reprezentat de un număr):

1) *primul grup* cuprinde caractere speciale (între 0 şi 1Fh sau 31), neprintabile, numite şi „de control” pentru că realizează diferite operaţii de control asupra unor periferice. De exemplu, *Carriage Return* – poziţionează cursorul în partea stângă a liniei curente, *Line Feed* – mută cursorul în jos, *Back Space* –mută cursorul înapoi o poziţie (spre stânga). Diferitele caractere de control pot avea roluri diferite la periferice diferite, existând foarte puţine standardizări între dispozitive în acest sens;

2) *al doilea grup* include semne de punctuaţie, caractere speciale şi cifre zecimale, dar şi caracterul spaţiu - *space* (cod ASCİİ 20h sau 32);

3) *al treilea grup* de 32 caractere ASCİİ este în mare măsură dedicat caracterelor majuscule din alfabetul englezesc: “A”...”Z” pentru 41h...5Ah (65...90), în total 26 caractere diferite; celelalte 6 caractere sunt dedicate unor simboluri speciale.

4) *ultimul grup* cuprinde caracterele minuscule ale alfabetului englezesc între 61h...7Ah, încă 5 simboluri speciale şi un caracter de control (delete).

Exemple: În Tabelul 1.1 se pot urmări grupurile de caractere, ca mai jos:
Litera A – se reprezintă ca numărul 65=41h (coloana 4 şi linia 1),
Litera M – se reprezintă ca nr. 77=4Dh (coloana 4 şi linia 13 - coresp. 0Dh),
Cifra 5 - se reprezintă ca numărul 35h (coloana 3 şi linia 5),
Spaţiu – se reprezintă ca numărul 20h (coloana 2 şi linia 0).

În concluzie, se pot specifica şi reţine următoarele: literele mari încep de la 41h, literele mici încep de la 61h, iar cifrele [0...9] sunt în gama [30h,...,39h].

Exemplu: Valoarea numerică a şirului ASCİİ „Salut” este 53h 61h 6Ch 75h 74h. Valoarea 53h, în zecimal e 83, iar în binar pe 7 biţi se scrie 1010011b, dar în memorie va fi stocată ca octet de valoare 01010011b. Un program care face depanare ar putea afişa această valoare ca „53” (fără să mai precizeze şi sufixul ***h*** de la hexa), dar dacă această valoare ar fi copiată în zona de memorie video, atunci pe ecran apare „S”, deoarece 53h este codul ASCİİ al lui „S”.

Stocarea caracterelor în memoria PC-ului se poate realiza prin mai multe seturi de caractere: standard ASCİİ (0–127), ASCİİ extins (0–255), ANSİ (0–255), Unicode (0–65,535), dar e important să existe o codificare standard a acestora deoarece programele pot ajunge pe alte PC-uri sau pot utiliza alte dispozitive periferice (deci e esenţial ca acestea să „vorbească aceeaşi limbă”).

Atât ASCİİ cât şi ANSİ au fost înlocuite de Unicode, care în primele 128 caractere îl substituie pe ASCİİ. Astfel, se poate deschide un fişier codificat ASCİİ pe un sistem care foloseşte Unicode, fără nici o problemă. Standardul ASCİİ a fost cea mai utilizată schemă de codificare pe www (World Wide

Web) până în 2007, fiind deturnat de **standardul Unicode** (include ASCİİ ca subset). Acesta a apărut din nevoia de a reprezenta o varietate mai largă de coduri în toate limbile internaţionale, standardizat, universal valabil. Unicode defineşte coduri pentru caractere, simboluri, semne de punctuaţie ce pot exista în toate limbile lumii, deci folosind orice alfabet sau simbol. Codul Unicode al caracterului @ este 40h (Tabelul 1.3). Apăsarea tastelor ***0040*** și apoi a combinaţiei ***Alt+X*** va produce apariţia caracterului @.

Coduri Ascii: Există o mare diferenţă între valori binare și coduri Ascii din punct de vedere al interpretării. De exemplu, un octet stocat în memorie de valoare 41h poate fi interpretat ca valoarea 65, deci ca număr zecimal, sau poate fi caracterul ‚A', care are codul Ascii 41h. Ca reprezentare însă, există o mare diferenţă între valoarea 1 (scrisă în hexazecimal ca 01h) și caracterul ‚1' scris în hexazecimal ca 31h: în primul caz se folosește valoarea 1, iar în al doilea caz spunem că se folosește caracterul ‚1' și ne referim la codul Ascii al lui, adică 31h.

Este foarte important atunci când lucrăm cu valori, să înţelegem cum trebuie interacţionat cu aceste valori. **De exemplu,** dacă se definește o valoare sau un element al unui șir (așa cum apare în exemplul de mai jos) ca 1, acesta nu e identic cu '1'. În primul caz e valoarea 1, interpretată ca și codul Ascii al caracterului ☺ în Figura 1.5 (adresa 07103h), pe când în cel de-al doilea caz e codul Ascii al caracterului ‚1', adică valoarea 31h (adresa 07105h). O altă diferenţă este observabilă când ne referim la valoarea A în hexa, sau mai corect 0Ah și 'A'. În primul caz valoarea 0Ah e echivalentă valorii 10 (la adresa 07107h), pe când 'A' este caracterul având codul Ascii 41h (la adresa 07108h). În Figura 1.5, în partea de cod s-a mai subliniat în plus și diferenţa între adresarea indirectă *fără*, respectiv *cu* registrul BX.

```
org 100h        ; p1_08
.data
Sir db 0,1,2,'1','2',0Ah,'A','B'        ; definite la adresa cu offset 102h ...109h
.code
   mov bx, offset sir
   mov al, sir[1]                       ; AL = 01h
   mov al, sir[3]                       ; AL = 31h
   mov ah, [bx+5]                       ; AH = 0Ah
   mov ah, [bx+6]                       ; AH = 41h
```

```
07102:  00  000  NULL
07103:  01  001  ☺
07104:  02  002  ☻
07105:  31  049  1
07106:  32  050  2
07107:  0A  010  NEWL
07108:  41  065  A
07109:  42  066  B
```

Figura 1.5. Exemplu de elemente ale unui șir atât valori binare cât și coduri Ascii

Interacţiunea dintre tastatură, programul sursă și ecran : formatare/deformatare Ascii

Atunci când se apasă pe tasta cu numărul 1, de la tastatură se va înregistra/ prelua codul Ascii al lui ‚1' și va fi disponibil în cadrul programului (intern, în PC). Invers, atunci când dorim să se afișeze pe ecran cifra 1, va trebui să formăm codul Ascii al acestei cifre și apoi să găsim o modalitate (folosind întreruperi specifice ecranului) de a trimite acest cod înspre ecran din cadrul programului (sau din interiorul PC).

Figura 1.6 ilustrează operaţiile ce trebuie realizate atunci când se dorește preluarea de la tastatură a 2 cifre zecimale, considerate numere fără semn, adunarea lor și afișarea sumei pe ecran. Înainte de realizarea sumei e nevoie de obţinerea valorilor 2 și 5 din ‚2' și ‚5'; astfel, se va scădea 30h sau ‚0' din fiecare. După obţinerea sumei ca valoarea 7, aceasta trebuie la rândul ei transformată în ‚7' pentru a putea fi tipărită pe ecran. Problemele se pot complica destul de mult prin faptul că funcţiile de afișare prezentate până acum nu pot afișa un număr format din mai mulţi digiţi, deci valori mai mari decât 9 (exemplele de cod *p1_02* și *p1_03*). Pentru a putea opera cu astfel de valori (>9), rezultatul va trebui considerat un șir de cifre, exact ca în scrierea poziţională. De exemplu, dacă se dorește afișarea numărului 123, se va considera că prima dată trebuie să apară pe ecran 1, apoi cu o poziţie a cursorului mai spre dreapta 2, iar în final, cu încă o poziţie a cursorului spre dreapta cifra 3. Inclusiv la preluarea valorilor de la tastatură se va considera acest aspect în sens invers, de exemplu dacă se introduce de la tastatură 1, urmat de 2, apoi de 3, va trebui compus numărul 123 (o sută două zeci și trei) ca fiind 1*100+2*10+3. În plus, pentru simplitate, se pp. că aceste numere sunt doar pozitive.

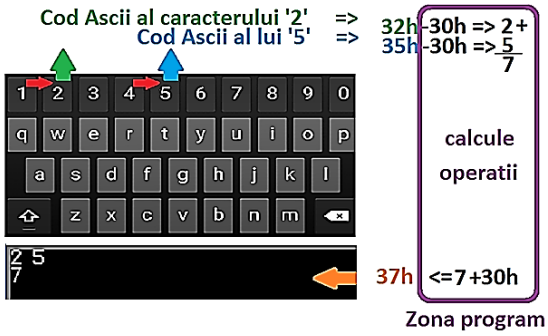

Figura 1.6. Transformări necesare la preluarea și afișarea numerelor

În cadrul programelor din acest volum se vor folosi doar valori mai mici decât 10. Atunci când se consideră numere (de exemplu se dorește calculul sumei, diferenţei, etc) ne referim la valoarea 2, nu la codul Ascii al caracterului/ tastei 2 (care este 32h). **Codurile Ascii ale cifrelor sunt: 30h ... 39h, ale literelor mici încep cu 61h, iar ale literelor mari încep cu 41h.**

În ceea ce privește tipul registrului în care se vor stoca aceste valori, la scrierea numerelor în zecimal e important de specificat convenţia folosită (cu semn sau fără semn) numărul respectiv putându-se scrie cu un anumit număr de biţi. Dacă se folosește operarea cu regiștri de 8 biţi, atunci valorile pot fi cuprinse:

- între 0 și 255 dacă se vor considera numere fără semn;
- între -128 și +127 dacă se consideră numere cu semn.

Dacă se folosește operarea cu regiștri de 16 biţi, atunci valorile pot fi cuprinse între:

- între 0 și 65535 dacă se vor considera numere fără semn;
- între -32768 și +32767 dacă se consideră numere cu semn.

Astfel, la reprezentarea numerelor cu semn, dacă se dorește preluarea de la tastatură a unor valori în zecimal, numere negative, de exemplu -45, utilizatorul va trebui să apese pe rând tastele: minus, apoi 4, apoi 5, iar din program, să se compună această valoare în registru.
La lucrul cu numere binare sau hexazecimale nu va exista această problemă, întrucât numerele exprimate în aceste baze nu prezintă caracterul ‚-'. În schimb, va trebui verificat bitul MSb: dacă este 0 numărul va fi pozitiv, iar dacă este 1 numărul va fi negativ. Complexitatea problemei crește semnificativ atunci când toate aceste aspecte se reunesc și deci când se cere realizarea unui program care să poată prelucra multiple astfel de cazuri, precum: adunare de numere și pozitive și negative, atât în zecimal, cât și în binar sau hexazecimal, și nu doar de numere reprezentate pe un singur digit, ci pe mai mulţi digiţi. Dacă se consideră cazul preluării unui număr în binar, bit cu bit, de la tastatură, o soluţie ar fi ca acest număr să fie depus într-o zonă din memorie și de acolo să se opereze mai departe cu el; similar, în cazul afișării valorilor mai mari de 1 digit. Pentru a obţine cifrele numărului (ca valori), se poate folosi această metodă: resturile obţinute se vor depune în memorie și de acolo se vor lua rând pe rând (folosind afișare de șir) pentru a fi transpuse pe ecran.

Cum se pot prelua date de la tastatură ?

Tabelul 1.2 prezintă mai multe întreruperi cu serviciile lor, acestea putându-se folosi pentru preluarea datelor de la tastatură. Ulterior, sunt explicate mai pe larg și sunt prezentate exemple cu fiecare dintre acestea în cadrul programelor.

Cum se pot afișa date pe ecran ?

Modul TEXT și modul grafic: Ecranul poate fi folosit în 2 moduri principale:
a) ***mod de lucru grafic*** – în care ecranul e organizat ca o matrice de puncte inde-pendente numite pixeli, are dimensiunea 640x480 și pot fi afișate atât caractere (de diferite dimensiuni sau fonturi) cât și alte obiecte grafice (puncte, linii, cercuri, etc);
b) ***mod de lucru text*** – în care ecranul este împărţit în zone de dimensiunea 8x8 pixeli (ca niște căsuţe sau matrici) în care se poate afișa un singur caracter.
În mod text, caracterele sunt afişate pe linii, de la stânga spre dreapta ecranului şi de sus în jos. Poziţia curentă de afișare este indicată de *cursor*, care dacă ajunge pe ultima poziţie/coloană de pe un rând, trece pe rândul următor, pe prima coloană. În mod text, pentru fiecare caracter se păstrează în memorie două informaţii: codul Ascii al caracterului care se afişează şi culorile utilizate pentru desenarea caracterului şi respectiv a fundalului pe care se face afişarea.
În Figura 1.7. s-a prezentat modul de codificare al atributului de culoare: primii 4 biţi codifică fondul, iar următorii 4 biţi sunt pentru culoarea de scriere.

Figura 1.7. Atributul de culoare al caracterului la afișarea pe ecran

Afișarea pe ecran a unui caracter sau a unui șir de caractere se poate realiza folosind întreruperile prezentate în Tabelul 1.2, cu serviciile asociate lor. Atât preluarea de la tastatură cât și afișarea pe ecran pot fi implementate direct cu aceste întreruperi, pentru cazul unui caracter sau șir de caractere, dar în cazul șirurilor de caractere se mai pot folosi și diverse artificii: de exemplu, se poate folosi Varianta III care afișează un singur caracter pe ecran în mod teletype, dar reluând-o în mod repetat pentru fiecare element al șirului, deci cu folosirea unei bucle.

Întreruperile exemplificate sunt următoarele:

Tabel 1.2. întreruperi folosite pentru ***afișarea datelor pe ecran*** (stânga) și întreruperi folosite pentru ***preluarea datelor de la tastatură*** (dreapta)

- întreruperi folosite pentru ***afişarea datelor pe ecran***: afişarea caracterelor individuale, afişarea mesajelor sau textelor alcătuite din secvenţe de caractere, afişarea la o anumită poziţie pe ecran prin intermediul funcţiei de poziţionare a cursorului text, afişarea caracterelor folosind atribute de culoare;

Întreruperi folosite pt afișarea datelor pe ecran	***Exemplu de apel:***
Afişare ***şir de caractere***	
Varianta I)	**mov dx, offset sir**
; în DX e adresa de început a șirului	**mov ah,09h**
cu int 21h, serviciul 09h;	**int 21h**
sirul - terminat cu caracterul '$'	

- întreruperi folosite pentru ***preluarea datelor de la tastatură***, atât funcţii pentru preluarea individuală a caracterelor, cât şi funcţii pentru preluarea directă a unor şiruri sau secvenţe de caractere de la tastatură.

Întreruperi pt preluarea datelor de la tastatură	***Exemplu de apel:***
Preluare ***şir de caractere***	
Varianta I)	**mov dx, offset sir**
cu int 21h, serviciul 0Ah;	**mov ah,0Ah**
; în DX să fie adresa de început a șirului unde s-a alocat spaţiu în memorie pt el	**int 21h**
De exemplu: **sir db 20,22 dup (?)** ; un octet de val 20, inca 22 locatii initializate cu ? (la prima locatie ne va depune lungimea sirului tastat, urmat apoi de sirul in sine, Enter)	
-> Ana are multe mere (18 caractere)	
-> 20, 18, :A:, 'n', 'a' ,…, Enter	

Varianta II) ***Afișare caracter din DL*** **cu int 21h, serviciul 02h;** se afișează caracterul din reg. DL	**mov dl, 'A'** **mov ah,02h** **int 21h**
Varianta III) ***Afișare caracter din AL*** **cu int 10h, serviciul 0Eh;** se afișează caracterul din registrul AL (*"mod teletype"*)	**mov al, 'A'** **mov ah,0Eh** **int 10h**
Varianta IV) ***Poziţionare cursor + afișare colorată caracter din AL*** **cu int 10h, serviciul 09h** și poziţionare cursor cu int 10h, serviciul 02h.	**mov dl,0** ;coloana **mov dh,0** ; linia **mov ah,02h** **int 10h** ; pozitionare cursor **mov al, 'A'** **mov ah,09h** **mov BH, 0;**pagina de afisare **mov bl, 00100001b** ;culoare **mov cx,1** **int 10h ;** afișarea colorată

Tabel. 1.2. Întreruperi pentru afișarea datelor pe ecran

Preluare caracter în AL

Varianta II) cu **int 16h, serviciul 0h**;	preluare fără ecou	**mov ah,0h** **int 16h** ; AL=cod Ascii
Varianta III) cu **int 21h, serviciul 01h**;	preluare **cu ecou**	**mov ah,01h** **int 21h** ; AL=cod Ascii
Varianta IV) cu **int 21h, serviciul 08h**;	preluare fără ecou	**mov ah,08h** **int 21h** ; AL=cod Ascii

Tabel. 1.2. Întreruperi pt preluarea datelor de la tastatură

Afișarea pe ecran:

Varianta I) cu int 21h, serviciul 09h (mesajul să aibă inserat caracterul '**$**' (deci codul Ascii 24h sau 36 la sfârșit);
Varianta II) cu int 21h, serviciul 02h;
Varianta III) cu int 10h, serviciul 0Eh;
Varianta IV) cu int 10h, serviciul 09h;
Varianta V) folosind lungime șir în CX și oricare din var. I)...IV).

Figura 1-1. Execuţia programului cu așteptare de tastă și fără coborâre de cursor (stânga), respectiv cu coborâre de cursor (dreapta) folosind CR urmat de LF (dacă vrem să dăm ENTER, va trebui să apelăm funcția de afișare de 2 ori: afișăm 0Dh și apoi 0Ah)

Dacă nu se folosește caracterul '$' la serviciul de afișare din varianta I, poate să apară eroare, precum cea ilustrată în Figura 1-2 sau e posibil să se afișeze toate caracterele găsite până la întâlnirea primului caracter '$' din memorie.

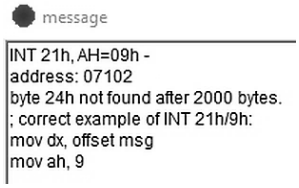

Figura 1-2. Posibil mesaj de eroare la folosirea întreruperii int **21h** cu serviciul **09h** pentru un șir care nu se termină cu caracterul '$'

Figura 1-3. Posibilitatea afișării colorate a fiecărui caracter prin utilizarea întreruperii int **10h** cu serviciul **09h**

În plus, dacă în registrul CX se depune o valoare mai mare, de exemplu 2 și se asigură o deplasare a cursorului tot cu 2, se poate obţine un efect așa cum este ilustrat în Figura 1-4.

Figura 1-4. Posibilitatea afișării multiple a fiecărui caracter prin utilizarea întreruperii int **10h** cu serviciul **09h**

Preluarea de la tastatură:

Modul I) cu int 21h, serviciul 0Ah pentru ***preluare șir de caractere;***
Modul II) cu int 16h, serviciul 0h, sau
cu int 21h, serviciul 01h, sau
cu int 21h, serviciul 08h pentru ***preluare caracter.***

Modul I) În vederea unei ***preluări a întregului șir***, se poate folosi întreruperea **int 21h cu serviciul 0Ah**. În DS:DX se va specifica adresa de început a zonei în care se depune șirul citit de la tastatură; de exemplu, dacă apare specificat cu directiva **sir db 20, 22 dup('?')**; se va aloca un spaţiu de 22 locaţii în memorie (de tip byte), în care se vor putea prelua de la tastatură maxim 20 caractere, de la *offset sir+2*.

Caracterele preluate de la tastatură se vor regăsi în memorie începând cu adresa *offset sir + 2*, iar la adresele *offset sir + 0* și *offset sir + 1* vor fi așa numiţii „octeţi de control". Pentru exemplul specificat, va fi astfel:
- la *offset sir + 0* va fi valoarea 20, iar
- la *offset sir + 1* va fi numărul de caractere introdus de utilizator.

Concret, dacă utilizatorul va tasta 'ana are mere', adică 12 caractere, la *offset sir + 0* va fi valoarea 20, la *offset sir + 1* va fi valoarea 12 (nr. de caractere tastate), iar de la *offset sir + 2* se vor regăsi pe rând caracterele 'a', apoi 'n', apoi 'a', ș.a.m.d. până la adresa *offset sir + 13* inclusiv.

```
org 100h      ; p8_0                    ; asamblorul începe de la adresa 100h
.data
sir db 20, 22 dup('?')    ; alocare spaţiu pentru maxim 20 caractere în memorie
.code
lea dx, sir   ; DX=offset sir – necesar pentru apel întrerupere int 21h
mov ah, 0ah               ; serviciul 0Ah al întreruperii int 21h
int 21h       ; se apelează întreruperea de preluare caractere de la tastatură
```

Modul II) Un alt mod de preluare al șirului, ar fi folosind ***citirea de la tastatură caracter cu caracter***; tastele se vor prelua **până la apăsarea unui anumit caracter**, al cărui cod Ascii se va verifica prin program.
Reluarea programului în buclă se poate asigura prin salt necondiţionat, prin folosirea instrucţiunii *jmp eticheta*. Programul, odată ajuns în acel punct, reia de la eticheta specificată, neţinând cont de nici o condiţie. Ieșirea din buclă (pentru a nu fi una infinită) este asigurată prin verificarea codului tastei apăsate de utilizator cu codul Ascii al tastei Esc (prin folosirea instrucţiunii *cmp al, 1Bh;*). Se reamintește că prin folosirea int 16h cu serviciul 0h se preia o tastă de la utilizator, codul Ascii al acelei taste depunându-se în registrul AL.

2.1. Preluarea și afișarea unui caracter

Problema 2.1.

Să se scrie o secvenţă de program care să afișeze un caracter pe ecran.

a) Definiţi caracterul direct în registru, iar apoi în memorie (ca o variabilă de tip byte).

b) Transformaţi secvenţa de la a) astfel încât caracterul să fie preluat de la tastatură. Testaţi mai multe variante de *preluare a caracterului de la tastatură*.

c) Testaţi mai multe variante de *afișare a caracterului pe ecran*.

Rezolvare:

a) Pentru afișarea unui caracter pe ecran există mai multe posibilităţi, așa cum s-a prezentat anterior și așa cum se va putea urmări la rezolvarea punctului c). Deocamdată, pentru rezolvarea cerinţei de la punctul a) se va opta pentru **folosirea întreruperii int 21h cu serviciul 02h** (prezentată ca Varianta II în Tabelul 1.2). Această întrerupere va afișa pe ecran caracterul al cărui cod Ascii se află în registrul DL înainte de apelul int 21h.

Cerinţa de la punctul a) este ca acel caracter care se dorește a fi afișat să nu fie definit în memorie, ci să fie scris direct într-un registru (***versiunea a1)***). Astfel, s-a optat pentru depunerea lui direct în registrul necesar afișării – adică în registrul DL. ***Versiunea a2)*** a programului definește acest caracter în zona de memorie ca octet (byte) având valoarea 41h (deci codul Ascii al caracterului 'A'), acesta putând fi adresat din memorie cu numele *Val*. Accesul la această variabilă din memorie se va putea realiza în două moduri: fie direct prin numele Val, fie prin folosirea unui pointer la adresa unde se găsește Val în memorie. Tabelul 2.1 prezintă varianta cu accesare direct prin nume (adică Val) a variabilei din memorie.

Tabel 2.1. Rezolvarea problemei **P2.1 a)** în două variante

a1) Versiunea cu definire de data direct în registru

```
org 100h        ; p2_01_a1
.data



.code
; depune în DL codul Ascii al caracterului ce se dorește afișat
mov dl, 'A'     ; DL='A'
; serviciul de afișare caracter din DL
mov ah,02h      ; serviciul 02h
int 21h         ; apel int 21h cu serv. 02h pt afisare
ret
```

a2) Versiunea cu definire de dată în memorie

```
org 100h        ; p2_01_a2
.data
Val db 'A'      ; definește o variabilă
                ; de tip byte în memorie
.code
;preia Val din memorie în DL
mov dl, Val     ; DL='A'
; serviciul de afișare caracter din DL
mov ah,02h      ; serviciul 02h
int 21h         ; apel int 21h cu serv. 02h pt afisare
ret
```

În memorie, la adresele cu offset 100h și 101h asamblorul rezervă 2 locaţii necesare în vederea execuţiei programului. Datele definite în segmentul de date vor apărea în zona de memorie începând de la adresa cu offsetul 102h.

```
07100: EB 235 δ
07101: 01 001 ☺
07102: 41 065 A
```

Figura 2.1. Zona de memorie după depunerea variabilei ***Val*** (definită ca 'A') la adresa 0700h:0102h

Figura 2.2. Rezultatul execuţiei programului a) cu afișarea caracterului 'A'

Figura 2.3. Rezultatul execuţiei programului b2) cu afișarea caracterului 'A' pe ecran

b) În locul definirii datelor în regiștri sau în memorie, se poate opta pentru preluarea acestora de la tastatură.

Pentru aceasta, așa cum s-a prezentat în Tabelul 1.2, se pot folosi mai multe variante:

b1) cu int 16h, serviciul 0h

b2) cu int 21h, serviciul 1h

b1) într-o primă versiune se va folosi secvenţa de instrucţiuni:

```
mov ah, 0
int 16h
```

prin care se așteaptă apăsarea unei taste. Codul Ascii al tastei apăsate de utilizator se va regăsi în registrul AL după execuţia instrucţiunii int 16h. Programele din Tabelul 2.1 se vor modifica după cum urmează:

Tabel 2.2. Rezolvarea problemei **P2.1 b)** cu int 16h, serviciul 0h

b11) Versiunea cu afișare dată direct din registru, după preluarea ei de la tastatură	b12) Versiunea cu afișare dată din memorie, după preluarea acesteia de la tastatură
org 100h *; p2_01_b11* .data .code **mov ah, 0** **int 16h** ; caracterul preluat se va regăsi în AL **mov dl, al** ;trebuie copiat în DL **mov ah,02h** ; afișare **int 21h** ret	org 100h *; p2_01_b12* .data **Val db ?** .code **mov ah, 0** **int 16h** ; caracterul preluat se va regăsi în AL **mov Val, al** ; se copiază și în zona de memorie **mov dl, Val** ; se poate folosi și **mov dl,al** **mov ah,02h** ; afișare **int 21h** ret

La execuţia programului, este ciudat modul cum are loc interacţiunea cu utilizatorul; despre acest mod de preluare al tastei se spune că este „fără ecou”, aceasta însemnând că atunci când îl tastăm, acesta nu va fi afișat pe ecran (practic nu se va vedea nimic pe ecran, ca și cum nu am fi tastat nimic). Afișarea pe ecran a unui caracter ‘A’ (așa cum apare în Figura 2.2) se va datora secvenţei de afișare.

O altă variantă de acces la informaţia din memorie ar fi prin folosirea pointerilor în locul folosirii numelui variabilei. Astfel, definind BX (de exemplu) ca pointer la locaţia de memorie (se reamintește că doar BX, BP, SI, DI sau un eventual deplasament se pot folosi la adresare), programul din dreapta din Tabelul 2.2, se va rescrie:

Tabel 2.3 Rezolvarea problemei **P2.1 b)** cu int 16h, serviciul 0h folosind pointer la zona de memorie unde a fost definită variabila

a3) Versiunea cu afișare dată definită în memorie, folosind pointer la memorie (adresare „bazată” cu BX)	b13) Versiunea cu afișare dată preluată de la tastatură, copiată ulterior în memorie, fol. pointer la memorie (adresare „bazată” cu BX)
org 100h *; p2_01_a3* .data **Val db ‘A’** .code ; se definește BX ca pointer la zona Val **mov bx, offset Val** ;serviciul de afișare caracter **mov ah,02h** **mov dl,[bx]** ;se depune în DL ; elementul pointat de BX din memorie **int 21h** ;apel întrerupere int 21h ret	org 100h *; p2_01_b13* .data **Val db ?** .code ; se definește BX ca pointer la zona Val **mov bx, offset Val** **mov ah,0h** **int 16h** ;în AL va fi codul Ascii ; al tastei apăsate de utilizator ;se depune caracterul preluat în mem. **mov byte ptr [bx], al** ;serviciul de afișare caracter **mov ah,02h** **mov dl,[bx]** ;se depune în DL ; elementul pointat de BX din memorie **int 21h** ;apel întrerupere int 21h ret

b2) În locul întreruperii **int 16h cu serviciul 0h – *preluare caracter fără ecou***, se poate folosi **int 21h cu serviciul 01h – *preluare caracter cu ecou.*** Astfel, primul caracter ‘A’ afișat pe ecran ***(***dintre cele două care apar în Figura 2.3) se va datora preluării lui de la tastatură („cu ecou”), iar cel de-al doilea se va datora afișării lui.

În secvenţele de program de la punctul b1) se vor modifica doar cele 2 instrucţiuni pentru preluare caracter de la tastatură, în rest programele rămânând neschimbate.

c) Afișarea caracterului pe ecran:

Mai multe variante de afișare a caracterului pe ecran sunt disponibile, așa cum s-a prezentat în Tabelul 1.2:

c1) int 21h cu serviciul 02h; (DL)

c2) int 10h cu serviciul 0Eh; (AL)

c3) int 10h cu serviciul 09h.

c1) Prin utilizarea **int 21h cu serviciul 02h** s-a văzut deja în cadrul programelor prezentate la punctele a) și b); **caracterul trebuie specificat în registrul DL.**

c2) O ***a doua modalitate*** de afișare a unui caracter pe ecran este prin folosirea **int 10h cu serviciul 0Eh; caracterul va trebui specificat în registrul AL**, iar în acest fel cursorul este mutat în mod automat pe poziţia imediat următoare, motiv pentru care se mai numește și ***mod teletype***. Astfel, versiunea de program a1) cu definirea datei direct în registru se va modifica după cum urmează:

```
...                 ; p2_01_c2
.code
mov al,'A'          ; în AL se depune codul Ascii al caracterului de afișat
mov ah,0eh          ; serviciul pentru afișare caracter în mod teletype
int 10h             ; apelul întreruperii int 10h
ret
```

c3) O ***a treia modalitate*** de afișare este prin utilizarea **întreruperii int 10h cu serviciul 09h** în locul celui teletype. Avantajul folosirii acestui mod este că vom avea la dispoziţie și un atribut de culoare și un număr de afișări al acelui caracter pe ecran (Figura 2.4). Inconvenientul este că va trebui ***poziţionat cursorul după fiecare caracter afișat***; totuși, aceasta se poate realiza simplu, folosind **întreruperea 10h, serviciul 02h**.

```
.code               ; p2_01_c3
mov dl,0            ; coloana de unde începe afișarea
mov dh,0            ; linia de unde începe afișarea
mov ah,02h          ; serviciul de poziţionare cursor
int 10h             ; apelul întreruperii int 10h
mov al, 'A'         ; în AL se depune codul Ascii al caracterului de afișat
mov ah,09h          ; serviciul pentru afișare caracter la cursor
mov bl, 11100001b   ; atribut de culoare: scris albastru, fond galben luminos
mov cx,1            ; CX=1 număr de afișări ale caracterului
int 10h             ; se apelează întreruperea de afișare caracter
ret                 ; revenire în S.O.
```

În plus, dacă în registrul CX se depune o valoare mai mare, de exemplu 5, se poate obţine un efect de afișare multiplă, așa cum se poate regăsi în Figura 2.4 b).

Figura 2.4. Posibilitatea afișării colorate a caracterului prin utilizarea întreruperii int 10h cu serviciul 09h, cu parametrul a) CX=1; b) CX=5

2.2. Preluarea și afișarea unui șir de caractere sau mesaj

Problema 2.2.

Să se scrie o secvenţă de program care să afișeze un mesaj pe ecran. Pentru simplitate, mesajul va fi definit ca șir de caractere în segmentul de date.

Rezolvare:

Pentru rezolvarea problemei se pot utiliza mai multe variante, după cum urmează:

Varianta I) cu int 21h, serviciul 09h;
Varianta II) cu int 21h, serviciul 02h;
Varianta III) cu int 10h, serviciul 0Eh ;
Varianta IV) cu int 10h, serviciul 09h;
Varianta V) folosind lungime șir în CX și oricare din variantele I)...IV).

Varianta I) Se poate folosi **întreruperea int 21h cu serviciul 09h, cu obligativitatea ca acel mesaj să aibă inserat caracterul '$' (deci codul Ascii 24h sau 36) la sfârșit.** Se pot folosi apostroafe sau ghilimele pentru definirea șirului de caractere în memorie.

```
; afișare pe bază de cunoaștere a unui caracter ca terminator de șir, cazul cu '$'
org 100h            ; p2_02_v1    ; va rezulta un program executabil de tip com
.data
msg db "My first assembly program!", "$"   ; definirea mesajului în memorie
.code
mov dx, offset msg      ; se încarcă offsetul șirului msg în registrul DX
mov ah, 09h             ; serviciul de afișare al întreruperii cu tipul 21h
int 21h                 ; se apelează întreruperea int 21h, cu serviciul 09h
ret                     ; revenire în S.O.
```

În loc de ***msg db "My first assembly program!", "$"***, se mai poate folosi:

msg db "My first assembly program!$", sau

msg db "My first assembly program!",24h, etc

Orice caracter în plus, chiar și spaţiu (dar specificat între ghilimele, apostroafe sau cu virgulă ca și cod Ascii) se va reprezenta în zona de memorie. În Figura 2.5 se poate observa, de exemplu, un caracter 'space' (la adresa 0711Ch), între '!' și '$', întrucât mesajul a fost definit sub forma:

msg db "My first assembly program! $"

Dacă se mai adaugă în plus instrucţiunile:

mov ah, 0

int 16h ;apelul întreruperii int 16h, serviciul 0h

cu care se așteaptă apăsarea unei taste înainte de instrucţiunea ***ret***, atunci ecranul la execuţie (Run) va arăta așa cum se poate observa în Figura 2.6 stânga (este afișat cursorul și se așteaptă apăsarea unei taste).

În schimb, dacă mesajul se va defini sub forma:

msg db "My first assembly program!",0Dh,0Ah,"$"

(și de asemenea rămân cele 2 instrucţiuni de preluare tastă ***mov ah, 0*** și ***int 16h***), ecranul va arăta așa ca în Figura 2.6 dreapta; se observă diferenţa între poziţiona-rea cursorului după afișarea mesajului. Astfel, concluzia este că dacă dorim coborârea cursorului, vom folosi 0Dh și 0Ah (sau 13 și 10) – codurile Ascii pentru CR-Carriage Return, respectiv LF-Line Feed, înainte de afișarea caracterului '$'.

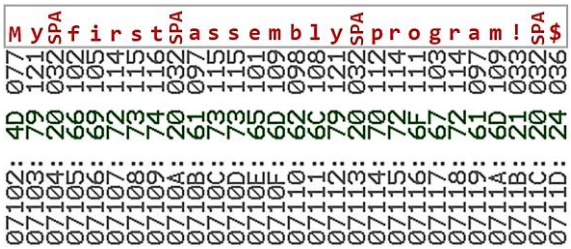

Figura 2.5. Zona de memorie după depunerea șirului de caractere *"My first assembly program! $"* începând de la adresa 0700h:0102h

Figura 2.6. Execuţia programului cu așteptare de tastă și fără coborâre de cursor (stânga), respectiv cu coborâre de cursor (dreapta) folosind CR urmat de LF

Dacă nu se folosește caracterul '$' la acest serviciu de afișare, poate să apară eroare, precum cea ilustrată în Figura 2.7 sau e posibil să se afișeze toate caracterele găsite până la întâlnirea primului caracter '$' din memorie.

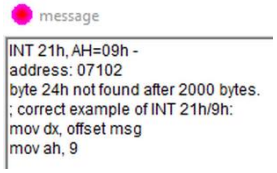

Figura 2.7. Posibil mesaj de eroare la folosirea întreruperii int 21h cu serviciul 09h pentru un șir care nu se termină cu caracterul '$'

Varianta II) O ***a doua modalitate*** de afișare a unui șir de caractere pe ecran este prin folosirea afișării unui caracter (posibilităţile prezentate la Problema 2.1) dar aplicate în mod repetat, pentru fiecare element al șirului. O posibilitate este folosirea **int 21h cu serviciul 02h; caracterul va trebui specificat în registrul DL** (asemănătoare cu versiunea c1) de la Problema 2.1).

```
; afișare pe bază de cunoaștere a unui caracter terminator de șir, cazul '?' (în locul '?' se poate folosi orice caracter
tastabil, cu condiţia ca acesta să nu apară și în interiorul șirului de caractere cu care se va lucra).
org 100h            ; p2_02_v2   ; se va asambla în memorie începând cu adresa 100h
.data                            ; segmentul de date – se definesc datele programului
msg db "My first assembly program!?"          ; mesajul ce se dorește a fi afișat
.code                            ; segmentul de cod – se scriu instrucţ. programului
mov si, offset msg               ; în SI se depune adresa de început (offset) mesaj
mov ah,02h                       ; serviciul folosit la afișare
iar:    mov dl,[si]              ; caracterul care se va afișa va fi depus în DL
        inc si                   ; se poziţionează pe următorul element din șir
        cmp dl,'?'               ; s-a ajuns la sfârșit?
        je gata                  ; dacă DA, sare la eticheta gata
        int 21h                  ; se apelează întreruperea pt afișare caracter din DL
    jmp iar                      ; se repetă pentru următorul element al șirului
gata:   mov ah, 0                ; dacă s-au terminat toate elementele de prelucrat,
        int 16h                  ; se apelează int 16h cu 0 pentru a aștepta o tastă ca sa iasa afara din program
ret                              ; revenire în S.O.
```

Varianta III) O ***a treia modalitate*** de afișare a unui șir de caractere pe ecran este similară cu Varianta II, dar se folosește **int 10h cu serviciul 0Eh; caracterul va trebui specificat în registrul AL** (asemănătoare cu versiunea c2) de la Probl.2.1). Avantajul major al acestui mod se remarcă în acest caz (de afișare a mai multor caractere), întrucât nu se controlează suplimentar și poziţia cursorului. Segmentul de date, precum și ultima parte sunt identice cu cele din programul anterior.

```
.code     ; p2_02_v3
mov si, offset msg       ; în SI se depune adresa de început (offset) mesaj
mov ah,0Eh               ; serviciul folosit la afișare – modul teletype
iar:    mov al,[si]      ; caracterul care se va afișa va fi depus în AL
        inc si           ; se poziţionează pe următorul element din șir
        cmp al,'?'       ; s-a ajuns la sfârșit?
        je gata          ; dacă DA, sare la eticheta gata
        int 10h          ; se apelează întreruperea pt afișare caracter din AL
    jmp iar
```

Varianta IV) O ***a patra modalitate*** de afișare este și aceasta similară cu Varianta II, dar prin utilizarea **întreruperii int 10h, serviciul 09h** în locul celei teletype. Avantajul este că vom avea la dispoziţie atributul de culoare și posibilitatea afișării în mod repetat a acelui caracter. Inconvenientul, așa cum s-a menţionat anterior la versiunea c3) la Problema 2.1, este că va trebui ***poziţionat cursorul după fiecare caracter afișat*** (realizată în general cu **întreruperea 10h, serviciul 02h**).

```
.code     ; p2_02_v4_1
mov si, offset msg       ; în SI se depune adresa de început (offset) mesaj
mov bl, 00100001b        ; atributul de culoare cu care se începe programul
mov dl,0                 ; coloana de unde începe afișarea
mov dh,0                 ; linia de unde începe afișarea
iar:    mov ah,02h       ; serviciul de poziţionare cursor
        int 10h          ; apelul întreruperii int 10h
        mov ah,09h       ; serviciul pentru afișare caracter la cursor
        mov al,[si]      ; caracterul de afișat se ia din șir cu index SI
        inc si           ; SI e pointer la șir, se poziţionează pe următ. element
        mov cx,1         ; CX=1 număr de afișări ale caracterului
        add bl, 11h      ; BL=BL+11h pt modificare fond și culoare de scris
```

```
        cmp al,'?'      ; se verifică sfârșitul șirului
        je gata         ; dacă s-a ajuns la sfârșit, se sare la eticheta gata
        int 10h         ; altfel, se apelează întreruperea de afișare caracter
        add dl,1        ; se pregătește pt cursor cu 1 poziţie mai spre dreapta
    jmp iar             ; se reia bucla în mod automat
gata:   mov ah, 0       ; dacă s-au terminat toate elementele de prelucrat,
        int 16h         ; se apelează int 16h cu 0 pt a aștepta o tastă ca sa iasa afara din program
ret                     ; revenire în S.O.
```

Figura 2.8. Posibilitatea afișării colorate a fiecărui caracter prin utilizarea întreruperii int 10h cu serviciul 09h

În plus, dacă în registrul CX se depune o valoare mai mare, de exemplu 2 și se asigură o deplasare a cursorului tot cu 2, se poate obține un efect așa cum este ilustrat în Figura 2.9. Programul folosit este identic cu cel anterior, se modifică doar 2 instrucţiuni, așa cum se prezintă în continuare:

```
...         ; p2_02_v4_2
        mov cx,2            ; CX=2 – număr de afișări ale caracterului
        ...
        add dl,2            ; pregătește pentru cursor cu 2 poziţii mai spre dreapta
    ...
```

Figura 2.9. Posibilitatea afișării multiple a fiecărui caracter prin utilizarea întreruperii int 10h cu serviciul 09h

Varianta V) O ***altă modalitate de afișare*** **a unui șir de caractere pe ecran este prin folosirea lungimii șirului; această variantă poate fi combinată cu oricare din cele anterioare de la punctele II), ..., IV).**

Această variantă de rezolvare folosește întreruperea int 21h cu serviciul 02h care afișează pe ecran caracterul al cărui cod Ascii se află în registrul DL; totuși, se poate folosi oricare din celelalte două variante cu int 10h (serviciul 0Eh sau serviciul 09h) și atunci caracterul trebuie depus în registrul AL. La parcurgerea șirului, în acest caz se exploatează faptul că se cunoaște numărul de elemente ale șirului, nefiind nevoie de specificarea unui terminator de sfârșit de șir. Nu se face o verificare a valorii elementelor, ci doar se verifică dacă s-au parcurs atâtea elemente câte are șirul. Aceasta se realizează prin folosirea registrului CX (inițializat la început cu numărul dorit de parcurgeri) și utilizarea instrucţiunii ***loop***. Această instrucţiune înlocuiește de fapt un grup de 3 instrucţiuni: ***dec CX; cmp CX,0;*** și ***jnz etichetă;*** fiind foarte utilă în astfel de cazuri de implementare a buclelor în program.

```
; afișare pe bază de cunoaștere a lungimii șirului
org 100h        ; p2_02_v5
.data
msg db "My first assembly program!"     ; nu se folosește terminator de șir
lung EQU $-msg          ; lungimea șirului se poate găsi folosind operatorul $
                        ; care indică adresa curentă de după șirul msg
.code
mov si, offset msg      ; în SI se depune adresa de început (offset) mesaj
mov ah,02h              ; serviciul folosit la afișare caracter pe ecran
mov cx,lung             ; în CX se depune numărul de caractere de prelucrat
iar:    mov dl,[si]     ; în DL se depune caracterul de afișat
        inc si          ; se poziţionează pe următorul element din șir
        int 21h         ; se apelează întreruperea pt afișare caracter din DL
   loop iar             ; instrucţiunea loop se poate inlocui cu dec CX; cmp CX,0; și jnz iar;
mov  ah, 0              ; dacă s-au terminat toate elementele de prelucrat,
int 16h                 ; se apelează int 16h cu 0 pentru a aștepta o tastă
ret                     ; revenire în S.O.
```

Problema 2.3.

Să se scrie o secvenţă de program care să preia de la tastatură un șir de caractere, să depună acest șir în memorie și apoi să-l afișeze după modelul prezentat la Problema 2.2 variantele II), ..., V).

Rezolvare:

Diferenţa faţă de problema anterioară este că șirul nu se mai definește în memorie ci este preluat de la tastatură (deci fără necesitatea de acces la programul sursă); astfel, se introduce posibilitatea interacţionării cu utilizatorul. Pentru rezolvarea problemei se propun mai multe moduri de lucru:

Modul I) cu int 21h, serviciul 0Ah pentru ***preluare șir de caractere;***

Modul II) cu int 16h, serviciul 0h, sau

cu int 21h, serviciul 01h, sau

cu int 21h, serviciul 08h pentru ***preluare caracter.***

Programul va trebui să funcţioneze pentru orice șir de caractere introdus de utilizator până la apăsarea tastei Enter. Şirul nu se va defini în cadrul programului în segmentul de date: se va aloca spaţiu în memorie pentru șir, dar nu se va defini (nu avem de unde ști ce va tasta utilizatorul).

Modul I) În vederea unei preluări a întregului șir, se poate folosi întreruperea **int 21h cu serviciul 0Ah**. În DS:DX se va specifica adresa de început a zonei în care se depune șirul citit de la tastatură; de exemplu, dacă apare specificat cu directiva **sir db 20, 22 dup('?')**; se va aloca un spaţiu de 22 locaţii în memorie (de tip byte), în care se vor putea prelua de la tastatură maxim 20 caractere, de la *offset sir+2*.

Caracterele preluate de la tastatură se vor regăsi în memorie începând cu adresa *offset sir + 2*, iar la adresele *offset sir + 0* și *offset sir + 1* vor fi așa numiţii „octeţi de control”. Pentru exemplul specificat, va fi astfel:
- la *offset sir + 0* va fi valoarea 20, iar
- la *offset sir + 1* va fi numărul de caractere introdus de utilizator.

Concret, dacă utilizatorul va tasta ‘ana are mere’, adică 12 caractere, la *offset sir + 0* va fi valoarea 20, la *offset sir + 1* va fi valoarea 12 (nr. de caractere tastate), iar de la *offset sir + 2* se vor regăsi pe rând caracterele ‘a’, apoi ‘n’, apoi ‘a’, ș.a.m.d. până la adresa *offset sir + 13* inclusiv. Ilustrarea zonei de memorie în acest caz se poate vizualiza în Figura 2.10.

```
org 100h          ; p2_03_m1_1  ; se spune asamblorului să înceapă de la adresa 100h
.data
sir db 20, 22 dup('?')     ; alocare spaţiu pentru maxim 20 caractere în memorie
.code
lea dx, sir        ; DX=offset sir – necesar pentru apel întrerupere int 21h
mov ah, 0ah        ; serviciul 0Ah al întreruperii int 21h
int 21h            ; se apelează întreruperea de preluare caractere de la tastatură
mov bx, dx         ; BX=offset sir – necesar la adresare
mov ch, 0          ; CH=0
mov cl, ds:[bx+1]  ; în CX va fi lungimea șirului
```

O altă posibilă variantă de definire a șirului în memorie în vederea preluării acestuia cu int 21h, serviciul 0Ah este următoarea:

```
org 100h
.data    ; p2_03_m1_2
nrMax db 22     ; se definește o variabilă nrMax (primul octet de control)
lung db ?                ; lung se va iniţializa doar după ce s-au preluat caracterele de la
                         ; utilizator – acesta este cel de-al doilea octet de control
sir db 21 dup (?)        ; șirul efectiv de caractere tastate de utilizator (inclusiv Enter)
.code
...
lea dx, nrMax    ; se va scrie în loc de instrucţiunea lea dx, sir
...
; add si,2       ; nu va mai fi necesară în acest caz, deci se va comenta
; după ce s-au preluat caracterele șirului și s-au depus în memorie,
; sunt luate de acolo pe rând, pentru afișare cu Varianta II) de la Problema 2.2
```

Cu varianta ***sir db 20, 22 dup('?')***, se definește un prim octet de valoare 20 și apoi se alocă un spaţiu de 22 locaţii în care primul va fi pentru a stoca lungimea șirului, iar ultimul pentru Enter, în final rămânând tot 20 caractere efective.

Deoarece în program există variabila *lung* ca fiind lungimea șirului preluat de la tastatură, nu va mai fi nevoie de preluarea acestei informaţii din memorie, astfel nici instrucţiunile *mov ch,0* și *mov cl, byte ptr [sir+1]* nu vor mai fi necesare (deci se vor comenta așa cum se arată mai jos):

```
...
mov si, offset sir          ; se poziţionează SI pe începutul sir în memorie
add si,2                    ; decalaj datorat octeţilor de control

mov ah,02h                  ; serviciul de afișare tastă
;mov ch, 0                  ; CH=0
;mov cl, byte ptr [sir+1]   ;  sau mov cl, lung   ; în CX va fi lungimea șirului

iar:    mov dl,[si]         ; se depune în DL elementul pointat de SI din sir
        inc si              ; mută pointerul spre următorul element
        int 21h             ; apel întrerupere int 21h pentru afișare
   loop iar                 ; se repetă de câte ori specifică registrul CX

mov ah, 0                   ; se așteaptă o tastă
int 16h
ret                         ; revenire în S.O.
```

```
07102: 14 020 ¶
07103: 0C 012 ♀
07104: 61 097 a
07105: 6E 110 n
07106: 61 097 a
07107: 20 032 SPA
07108: 61 097 a
07109: 72 114 r
0710A: 65 101 e
0710B: 20 032 SPA
0710C: 6D 109 m
0710D: 65 101 e
0710E: 72 114 r
0710F: 65 101 e
```

Figura 2.10. Zona de memorie după preluarea șirului de caractere ***"ana are mere"*** (folosind int 21h cu serviciul 0Ah) începând de la adresa 0700h:0102h

Modul II) Un alt mod de preluare al șirului, ar fi folosind ***preluarea tastă cu tastă*****,** prin utilizarea întreruperii **int 16h cu serviciul 0h;** tastele se vor prelua **până la apăsarea unei anumite taste**, al cărei cod Ascii se va verifica prin program.

Reluarea programului în buclă este asigurată prin salt necondiţionat, prin folosirea instrucţiunii *jmp eticheta*. Programul, odată ajuns în acel punct, reia de la eticheta specificată, neţinând cont de nici o condiţie. Ieșirea din buclă (pentru a nu fi una infinită) este asigurată prin verificarea codului tastei apăsate de utilizator cu codul Ascii al tastei Esc (prin folosirea instrucţiunii *cmp al, 1Bh;*). Se reamintește că prin folosirea int 16h cu serviciul 0h se preia o tastă de la utilizator, codul Ascii al acelei taste depunându-se în registrul AL.

```
; programul verifică tasta ESC (cod Ascii 1Bh sau 27), dar înlocuibil cu orice tastă
org 100h     ; p2_03_m2
.data
msg db 20 dup(?)        ;se alocă spaţiu în memorie pt șir cu 20 elemente octet
.code
mov dx, offset msg      ; DX va fi pointer la zona rezervată pentru acel șir
xor bx, bx              ; BX se fol. pt a depune elementele șirului în memorie
iar:  mov ah, 0         ; se așteaptă o tastă
    int 16h             ; apel întrerupere de preluare tastă
    cmp al, 1Bh         ; dacă e 'ESC', revine în program; altfel, tot preia taste
    je stop             ; dacă s-a terminat preluarea, se sare le eticheta stop
    mov msg [bx], al    ; altfel, se depune în memorie codul Ascii al tastei
```

```
    inc bx              ; se incrementează BX la fiecare tastă apăsată
  jmp iar               ; se reia în buclă
; după ce s-au preluat caracterele șirului și s-au depus în memorie,
; sunt luate de acolo pe rând, pentru afișare cu Varianta II) de la Problema 2.2
stop:  mov si, offset msg  ; se poziţionează SI pe începutul msg în memorie
       mov ah,02h          ; serviciul de afișare caracter
       mov cx,bx           ; se depune în CX numărul de elemente ale șirului
afis:  mov dl,[si]         ; se depune în DL elementul pointat de SI din msg
       inc si              ; se mută pointerul spre următorul element
       int 21h             ; apel întrerupere int 21h pentru afișare
  loop afis                ; se repetă de câte ori specifică registrul CX
mov ah, 0                  ; se așteaptă o tastă înainte de ieșire
int 16h
ret                        ; revenire în S.O.
```

În locul **int 16h cu serviciul 0h** pentru ***preluare caracter fără ecou***, se mai poate folosi: **int 21h cu serviciul 01h** pentru ***preluare caracter cu ecou*** sau
int 21h cu serviciul 08h pentru ***preluare caracter fără ecou.***

Problema 2.4.

Să se scrie o secvenţă de program care să preia de la tastatură un șir de caractere, să depună acest șir în memorie și apoi să-l afișeze după modelul prezentat la Problema 2.2 varianta I).

Rezolvare:

Programul va trebui să funcţioneze pentru orice șir de caractere introdus de utilizator până la apăsarea tastei Enter. Totuși, dacă utilizatorul va introduce caractere '$', deoarece la afișare acesta va fi delimitatorul sfârșitului de șir, întâlnirea primului caracter '$' va determina oprirea afișării.
Şirul nu se va defini în cadrul programului în segmentul de date: se va aloca spaţiu în memorie pentru șir, dar nu se va defini (nu avem de unde ști ce va tasta utilizatorul). Se va folosi **întreruperea int 21h cu serviciul 0Ah** care va prelua un întreg șir de caractere și îl va depune în memorie.
Pentru a-l afișa apoi după modelul prezentat la Problema 2.2 Varianta I), va trebui depus în memorie caracterul '$' la sfârșitul șirului. Mai multe probleme din celelalte capitole ale cărţii vor folosi acest model de program; astfel, este important de reţinut că partea de început este necesară în vederea preluării șirului în memorie, depunerii caracterului '$' ca marcaj de sfârșit de șir și apoi pentru plasarea registrului BX pe primul element al șirului, respectiv înscrierea în CX a numărului de elemente ale șirului. Eventualele prelucrări vor fi inserate după această zonă.

```
org 100h            ; p2_04         ; spune asamblorului să înceapă de la adresa 100h
.data
sir db 20, 22 dup('?')     ; alocare spaţiu de maxim 20 caractere în memorie
; va transforma în majusculă literele mici de pe poziţie impară
.code
lea dx, sir         ; DX=offset sir – necesar pentru apel întrerupere int 21h
mov ah, 0ah         ; serviciul 0Ah al întreruperii int 21h
int 21h             ; se apelează întreruperea de preluare caractere de la tastatură

mov bx, dx          ; BX=offset sir – necesar la adresare
mov ah, 0           ; AH=0
mov al, ds:[bx+1]           ; în AX va fi lungimea șirului
mov si, ax          ; necesar la instrucţiunea de mai jos
mov byte ptr [bx+si+2], '$' ;pune '$' la sfârșitul șirului fol. adresare bazată indexată

mov cx, ax          ; CX=lungimea șirului
jcxz iesire         ; este șirul nul ? dacă DA (CX=0), sare la sfârșitul programului
add bx, 2           ; sare peste caracterele de control (de la adresa 102h și 103h)
```

```
; eventualele prelucrări în cadrul programului se vor insera în această zonă
afisare:
; controlul cursorului cu serviciul 02h al întreruperii cu tipul 10h
mov ah,02h      ; serviciul 02h al întreruperii int 10h pentru control poziţie cursor
mov dl,10       ; coloana 10
mov dh,3        ; linia 3
mov bx,0        ; pagina video 0
int 10h         ; apel întrerupere cu tipul 10h
; afișarea șirului specificat cu DS:DX, terminat cu caracterul '$'
lea dx, sir+2   ; în DX e pointer la șir
mov ah, 09h     ; serviciul 09h al int 21h pentru afișare șir terminat cu '$'
int 21h         ; apel întrerupere cu tipul 21h
iesire:  ret    ; revenire în S.O.
```

2.3. Preluarea de la tastatură și afișarea pe ecran a unui număr în baza 10 (un digit)

Până acum, s-a lucrat doar cu valori coduri Ascii ale diverselor caractere; în continuare, se va presupune că ceea ce se dorește a fi afișat sau ceea ce se va prelua de la utilizator reprezintă numere (formate din cifre binare, zecimale sau hexazecimale după cum se va specifica în cadrul problemei).

Problema 2.5.

Să se scrie o secvenţă de program care să preia de la tastatură 2 numere în baza zece fără semn (reprezentate pe câte un singur digit), să depună aceste numere în memorie, să calculeze suma lor și apoi să afișeze această sumă pe ecran. Pentru a se afișa corect, se vor introduce numere astfel încât suma lor să nu depășească valoarea 9.

v1) Varianta de rezolvare clasică, fără subrutine, fără macrouri:

```
org 100h  ; p2_05_1
.data
a db ?          ;se alocă doar spaţiu în memorie pentru variabila a, dar nu se definește
b db ?          ;se alocă doar spaţiu în memorie pentru variabila b, dar nu se definește
suma db ?       ;se alocă doar spaţiu pentru variabila suma, dar nu se definește
.code
; preluarea primului număr de la tastatură
mov ah,0        ; serviciul de preluare tastă de la tastatură, întreruperea 16h cu serviciul 0h (în AH)
int 16h         ; dacă de exemplu se va apăsa tasta 2, în AL se va returna codul Ascii al lui 2 => AL=32h
sub al,30h      ; pentru a obţine valoarea 2, se va scădea 30h din AL ("deformatare Ascii")

mov a,al        ; se mută în memorie la locaţia rezervată pentru a

; preluarea celui de-al doilea număr de la tastatură
mov ah,0        ; serviciul de preluare tastă de la tastatură,  întreruperea 16h cu serviciul 0h (în AH)
int 16h         ; dacă de exemplu vom apăsa tasta 6, în AL se va returna codul Ascii al lui 6 => AL=36h
sub al,30h      ; pentru a obţine valoarea 6 vom scădea 30h din AL ("deformatare Ascii")

mov b,al        ; se mută în memorie la locaţia rezervată pentru b

add al, a       ; în AL este deja valoarea b, deci peste ea se adaugă a=> AL=8
mov suma,al     ; se mută valoarea sumei în mem. la locaţia rezervată pt. suma

add al,30h      ; se transformă valoarea de afișat în cod Ascii => AL=38h ("formatare Ascii")
mov ah,0Eh      ; se afișează folosind serviciul 0Eh, mod teletype
int 10h         ; apel întrerupere int 10h

ret             ; revenire în S.O.
```

Este indicată urmărirea și înţelegerea subrutinelor folosite înainte de parcurgerea instrucţiunilor din programul principal.

v2) Aceeași problemă se va rezolva în continuare cu subrutine:

```
; segmentul de date e identic cu cel din varianta anterioară
.code    ; p2_05_2
call preiaCifra    ; prin apelul acestei subrutine s-a preluat prima cifră,  de ex., în AL s-a obţinut 2 dacă s-a apăsat tasta '2'
mov a,al           ; se mută în memorie la locaţia rezervată pentru a
call preiaCifra    ;prin apel, s-a preluat cea de-a doua cifră, de ex, în AL s-a obţinut 6 dacă s-a apăsat tasta '6'
mov b,al           ; se mută în memorie la locaţia rezervată pentru b
add al, a; AL=a+b
mov suma, al       ; suma=a+b=2+6=8
call afisCifra     ; se va afișa valoarea 8 pe ecran
ret                ; revenire în S.O.
; în simulator, abia după ret se vor așeza procedurile
; procedura de mai jos nu se va putea folosi și pentru litere, alte caractere, etc; ea va funcţiona doar pentru cifre întrucât se scade 30h – specific cifrelor
; în AL se va depune valoarea numărului preluat de la tastatură
preiaCifra proc        ; se va considera că în AL returnează valoarea;
        mov ah,0       ; serviciul de preluare tastă de la tastatură întreruperea 16h cu serviciul 0h (în AH)
        int 16h        ; dacă, de exemplu, se va apăsa tasta 2, în AL se va returna codul Ascii al lui 2=> AL=32h
        sub al,30h     ; pentru a obţine valoarea 2, se va scădea 30h din AL
        ret            ; revenire din surutină
preiaCifra endp
; procedura de mai jos nu se va putea folosi și pentru litere, alte caractere, etc
; ea va funcţiona doar pentru cifre întrucât se adună 30h – specific cifrelor.
; anterior apelului, în AL se va depune valoarea numărului ce se dorește a fi afișat
afisCifra proc         ; se va considera că în AL primește valoarea de afișat
        mov ah,0Eh     ; serviciul de afișare mod teletype
        add AL,30h     ; se formează cod Ascii necesar afișării
        int 10h        ; întreruperea 10h cu serviciul 0Eh (în AH)
        ret            ; revenire din surutină
afisCifra endp
```

v3) Aceeași problemă se va rezolva în continuare cu macrouri:

```
; p2_05_3  ;în cazul macrourilor, acestea trebuie definite înainte de zona de cod unde vor fi folosite, sau chiar înainte de directive .data, nu ca în cazul subrutinelor unde erau definite la sfârșit, după ret.
preiaCifra macro       ; se va folosi macrou în loc de subrutină
        mov ah,0       ; serviciul de preluare tastă de la tastatură întreruperea 16h cu serviciul 0h (în AH)
        int 16h        ; dacă, de exemplu, se va apăsa tasta 2, în AL se va returna codul Ascii al lui 2=> AL=32h
        sub al,30h     ; pentru a obţine valoarea 2, se va scădea 30h din AL
endm
afisCifra macro        ; se va folosi macrou în loc de subrutină
        mov ah,0Eh     ; serviciul de afișare, mod teletype
        add AL,30h     ; se formează valoarea de afișat ca și cod Ascii
        int 10h        ; apel întrerupere pentru afișare caracter
endm
.data   ;...           ; segmentul de date e identic cu cel din varianta anterioară
.code
preiaCifra             ; folosind macroul (se și transformă în valoare binară)   se preia o tastă și se …
mov a,al               ; se depune în memorie, în variabila a
preiaCifra             ; folosind macroul (se și transformă în valoare binară)
mov b,al               ; se depune în memorie, în variabila b
add al, a              ; se adună AL=a+b
mov suma, al           ; valoarea sumei se depune în memorie, în variabila suma
afisCifra              ; se invocă macroul de afișare caracter pe ecran
ret                    ; revenire din subrutină
```