1. Introducere în microprocesoare

1.1 Generalităţi

Elementul de bază al unui calculator este reprezentat de microprocesor, un *chip* deosebit de complex plasat de obicei pe placa de bază a sistemului de calcul. Microprocesorul asigură procesarea datelor, adică interpretarea, prelucrarea şi controlul acestora, supervizează transferurile de informaţii şi controlează activitatea generală a celorlalte componente care alcătuiesc sistemul de calcul. Structura internă a microprocesorului este compusă din mai multe micromodule interconectate prin intermediul unor căi de comunicaţie numite magistrale sau busuri interne care pot transfera *date* şi *instrucţiuni* (sau *comenzi*). Datele şi instrucţiunile formează un program care este executat pe un sistem de calcul şi reprezintă informaţia procesată.

Microcalculatoarele tipice folosesc o unitate centrală de prelucrare (CPU – Central Processing Unit), un ceas (clock) şi interfeţe cu memoria şi cu dispozitivele externe de intrare/ieşire. Unităţile sunt interconectate prin magistrale care transferă informaţiile între acestea.

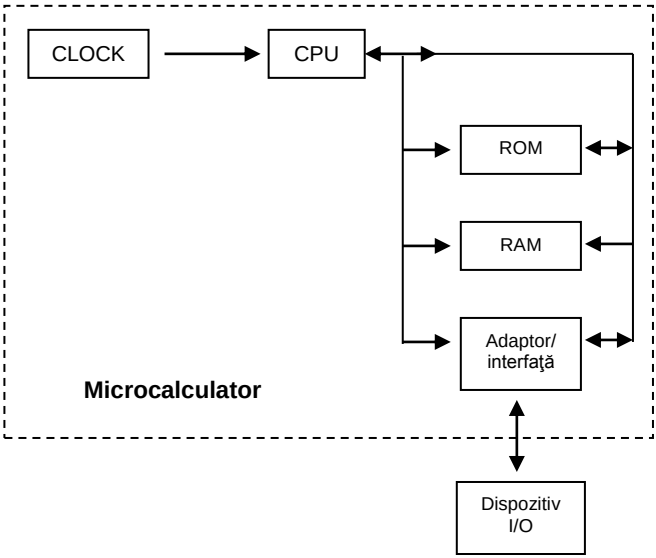

Fig.1.1. Exemplu de arhitectură de microcalculator

1.2 Exemplu de arhitectură de microprocesor

Microprocesorul conţine unitatea aritmetică şi logică (UAL) şi unitatea de control (UC). Acesta este conectat la memorie şi la dispozitivele de intrare/ieşire prin magistrale.
Informaţia este transmisă între unităţile microcalculatorului prin magistrale (bus-uri). Există câte o linie pentru fiecare bit de informaţie transmis, deci 16 linii pentru o magistrală de 16 biţi. Magistralele pot fi *de adrese*, *de date* şi *de control*.

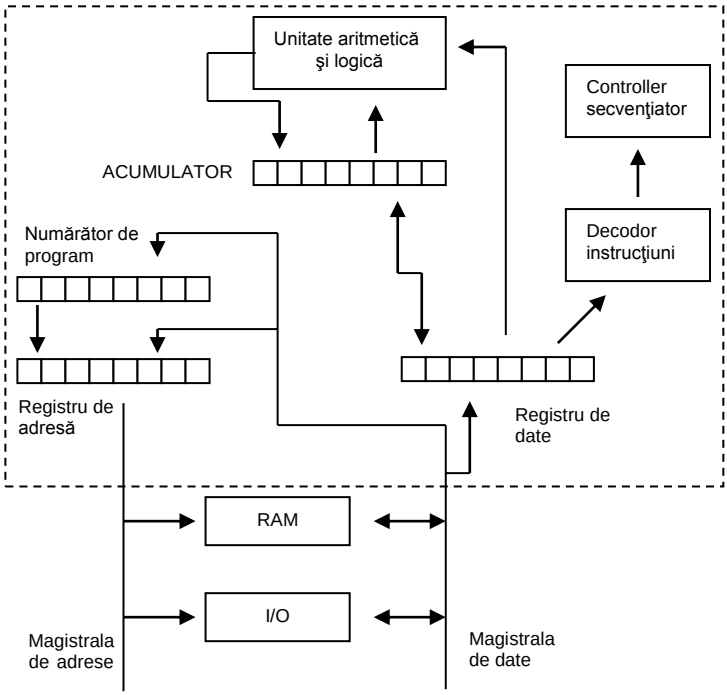

Fig.1.2. Exemplu de arhitectură de microprocesor

1.2.1 Unitatea aritmetică şi logică (UAL)

Toate operaţiile aritmetice/logice ale unui microprocesor au loc în unitatea aritmetică şi logică. Folosind o combinaţie de porţi şi bistabile interne, numerele pot fi adunate în mai puţin de o microsecundă, chiar şi în procesoarele mici. Operaţia ce trebuie executată este specificată de semnalele generate de unitatea de control prin decodificarea instrucţiunilor.

Datele asupra cărora se realizează operaţia pot proveni din memorie sau de la o intrare externă (port). Datele numerice sunt procesate pe baza unui set de instrucţiuni, având ca operaţii de bază adunarea, scăderea şi operaţiile logice.

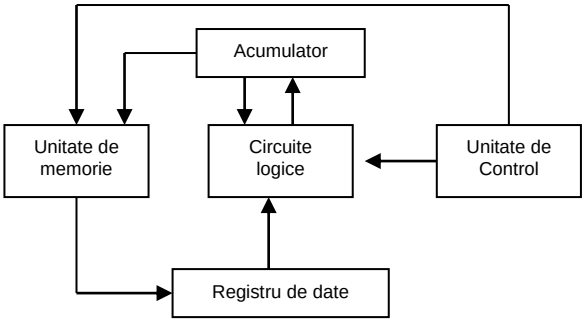

Fig.1.3. Unitatea aritmetică şi logică

1.2.2 Acumulatorul (Acc)

Acumulatorul este registrul principal al unităţii aritmetice şi logice a microprocesorului. Regiştrii sunt elemente de memorie care pot păstra datele. Acumulatorul conţine de obicei primul operand din datele implicate într-un calcul. De exemplu, dacă un număr din memorie este adunat la aceste date, suma înlocuieşte datele originale din acumulator. Acesta este locul de depozitare pentru rezultatele operaţiilor aritmetice, care pot fi apoi transferate în memorie sau la dispozitive periferice.

1.2.3 Unitatea de control a microprocesorului (UC)

Unitatea de control a microprocesorului coordonează operaţiile celorlalte unităţi prin generarea semnalelor de temporizare şi control. Funcţia microcalculatorului este de a executa programele stocate în memorie. Unitatea de control conţine logica necesară interpretării instrucţiunilor şi generării semnalelor necesare execuţiei acestor instrucţiuni. Cuvintele descriptive “fetch” şi “execute” sunt folosite pentru a descrie acţiunile unităţii de control. Aceasta citeşte instrucţiunea (fetch) prin generarea unei adrese şi a unei comenzi de citire către unitatea de memorie. Instrucţiunea de la adresa respectivă este transferată unităţii de control pentru decodare. Aceasta generează apoi semnalele necesare pentru execuţia instrucţiunii.

Cu toate că sunt componente electronice extrem de complexe, microprocesoarele execută instrucţiunile secvenţial. Microprocesoarele ce pot executa în paralel mai mult de o instrucţiune au o arhitectură superscalară, dispunând de două unităţi aritmetice şi logice. Succesiunea instrucţiunilor unui program este executată în general secvenţial. Printr-o

structură *pipeline* se pot executa instrucţiuni secvenţiale paralele, situaţie în care faze diferite ale unor instrucţiuni succesive se execută în acelaşi timp.
Toate instrucţiunile pe care le poate executa un microprocesor formează *setul de instrucţiuni al microprocesorului*. Acest set a fost proiectat şi optimizat pentru fiecare procesor în parte. Toate microprocesoarele Intel 80x86 inclusiv Pentium, au setul de instrucţiuni compatibil cu versiunile anterioare.

Microprocesor	Set de instrucţiuni
8088/8086	115
80186	126
80286	142
80386	200
80486	206
Pentium	211
Pentium MMX	211+ 57

Tabel 1.1. Procesoarele Intel 80x86 şi setul de instrucţiuni

Instrucţiunile limbajelor evoluate (High Level Language) (C, Pascal, Prolog, Lisp, etc) necesită transformarea lor de către compilator în instrucţiuni simple, interpretabile de către microprocesor. Astfel, pentru fiecare instrucţiune a limbajului de nivel înalt, se generează una sau mai multe instrucţiuni aparţinând setului de instrucţiuni al microprocesorului.

Setul de instrucţiuni pe care un procesor îl poate executa şi care este limbajul de programare la nivel scăzut al microprocesorului formează *limbajul maşină* sau *codul maşină*. Pentru a facilita utilizarea limbajului maşină, la care instrucţiunile reprezintă o înşiruire de biţi, se utilizează **mnemonicile** care abreviază operaţia executată de instrucţiune. Setul de instrucţiuni prezentat sub forma mnemonicilor reprezintă *limbajul de asamblare*. Modul în care partea hardware este comandată de utilizator prin intermediul software-ului, utilizând diverse nivele de limbaj, este prezentată în figura 1.4.

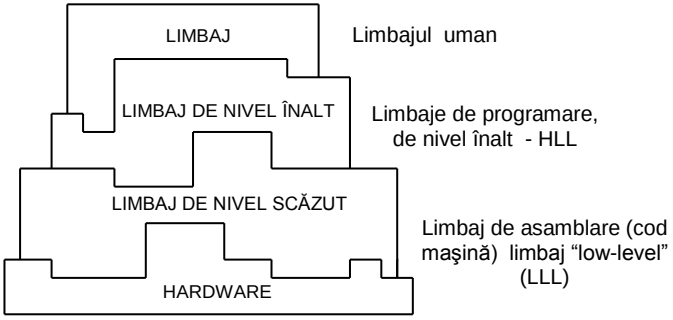

Fig.1.4. Ierarhizarea limbajelor

1.3 Arhitecturi de prelucrare

La baza majorităţii calculatoarelor actuale stau cele cinci criterii enunţate de von Neumann şi prezentate în figura 1.5.

Un calculator cu program memorat trebuie să posede:

- Intrare pentru un număr nelimitat de date şi instrucţiuni;
- Memorie din care se pot citi instrucţiuni şi operanzi şi în care se depun rezultate;
- Ieşire care să pună rezultatele la dispoziţia utilizatorului;
- Unitate de calcul (UAL – unitate aritmetică şi logică sau UE - unitate de execuţie) care să execute operaţii aritmetice şi logice asupra datelor din memorie;
- Unitate de comandă (sau control) - UC care să interpreteze instrucţiunile extrase din memorie şi să aleagă diferite acţiuni pe baza rezultatelor calculelor.

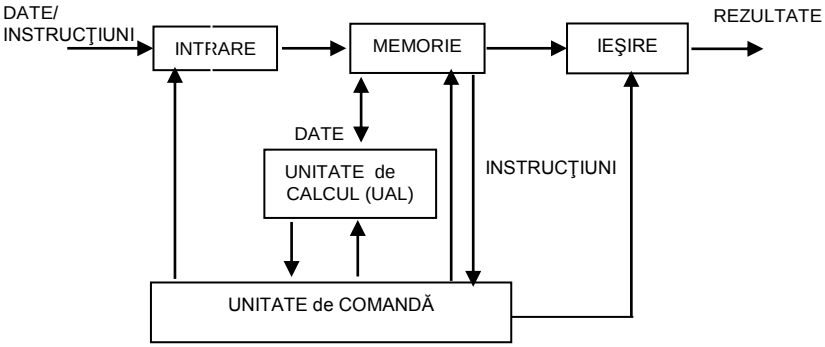

Fig. 1.5. Schema bloc a calculatorului cu program memorat

După modul în care se conectează procesorul la memorie, se extrag datele şi instrucţiunile sau se optimizează prelucrările, în practică se întâlnesc mai multe arhitecturi de procesare.

Arhitectura "von Neumann" sau SISD (single instruction single data) are următoarele caracteristici:

- extragerea datelor şi instrucţiunilor se face pe aceeaşi magistrală;
- instrucţiunile se extrag şi se execută secvenţial.

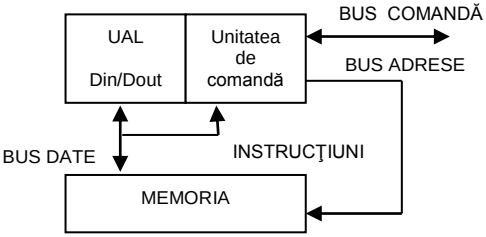

Fig. 1.6. Arhitectura SISD

Arhitectura Harvard are memoria partajată în memorie de date şi memorie de program, permiţând o prelucrare mai eficientă şi executarea paralelă a diferitelor faze ale instrucţiunilor (pipeline). Este utilizată cu precădere la procesoarele de semnal (DSP) şi microcontrolere.

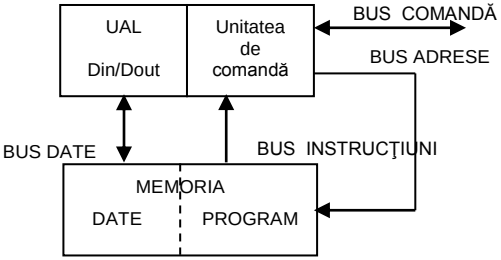

Fig. 1.7. Arhitectura Harvard

Arhitectura SIMD (single instruction multiple data) numită şi arie de procesoare sau *tablou sistolic* permite execuţia aceleiaşi instrucţiuni pe mai multe date simultan, ceea ce implică existenţa mai multor unităţi aritmetice. Arhitectura e utilă şi la prelucrări multimedia, fiind întâlnită la Pentium MMX.

Arhitectura MIMD (multiple instruction multiple data) sau Data Flow conţine mai multe procesoare care rulează programe diferite operând cu date diferite în paralel, conlucrând la rezolvarea unei aplicaţii (task)

1.4 Reprezentarea numerelor

În calculator, memoria conţine numere. Datorită logicii booleene pe care este conceput hardware-ul, calculatoarele stochează informaţia în format binar, nu zecimal. *Sistemul zecimal* foloseşte 10 digiţi (0-9), fiecare digit al unui număr având asociată o putere a lui 10 în funcţie de poziţia sa, deci sistemul de numerotaţie este un sistem poziţional:

$$234=2*10^2+3*10^1+4*10^0$$

*Sistemul **binar*** foloseşte doar două cifre, 0 şi 1, fiecare cifră a unui număr având asociată o putere a lui 2 în funcţie de poziţia sa:

$$11001_2=1*2^4+1*2^3+0*2^2+0*2^1+1*2^0=16+8+1=25_{10}$$

Adunarea în sistemul binar este intuitivă:

$$\begin{array}{r} 11011_2 \\ \underline{+10001_2} \\ 101100_2 \end{array}$$

*Sistemul **hexazecimal*** foloseşte baza 16 şi este folosit pentru reprezentarea prescurtată a numerelor binare. Se folosesc cifrele de la 0 la 9, iar ca cifre suplimentare se folosesc caracterele A – F. Modul de transformare în zecimal este asemănător:

$$2BD_{16}=2*16^2+11*16^1+13*16^0=512+176+13=701_{10}$$

Conversia hexa-binar şi invers se face foarte uşor: fiecare cifră hexa este transformată într-un număr binar pe 4 cifre, şi invers, grupuri de câte 4 cifre binare se transformă într-o cifră hexa:

0110	0000	0101	1010	0111	1110
6	0	5	A	7	E

Convenţiile de reprezentare a numerelor întregi

Numerele întregi pot fi reprezentate a) fără semn sau b) cu semn.

*a) Reprezentarea **fără semn*** este intuitivă: numărul 200_{10} va fi reprezentat în binar ca 11001000_2, în hexazecimal ca şi C8h.

b) Numerele întregi ***cu semn*** (pozitiv sau negativ) sunt însă mai complicat de reprezentat. Există 3 convenţii de reprezentare a numerelor cu semn, toate folosesc ca *bit de semn* bitul cel mai semnificativ al reprezentării. Bitul de semn este 0 pentru un număr pozitiv şi 1 pentru un număr negativ.

Modul şi semn: numărul va fi reprezentat din două parţi: bit de semn şi valoare absolută. Numărul 56_{10}=38h va fi reprezentat ca $\underline{0}0111000_2$, cu bitul de semn subliniat, iar -56 va fi $\underline{1}0111000_2$. Cea mai mare valoare reprezentabilă pe octet va fi +127=$\underline{0}1111111_2$, respectiv cea mai mică valoare -127=$\underline{1}1111111_2$. Această metodă are dezavantaje în organizarea logică a UC. Zero nu este nici pozitiv nici negativ, deci reprezentările $\underline{1}0000000_2$ şi $\underline{0}0000000_2$ sunt echivalente.

Complement faţă de 1 (C1): se calculează prin complementarea fiecărui bit din reprezentare. Reprezentarea lui +56, $\underline{0}0111000_2$ va fi deci $\underline{1}1000111_2$. Prin urmare, -56 va fi $\underline{1}1000111_2$. Bitul de semn a fost schimbat automat prin complementare. Complementul faţă de 1 are aceleaşi dezavantaje ca şi metoda de reprezentare în modul şi semn: două reprezentări echivalente pentru zero şi aritmetică complicată.

Complement faţă de 2 (C2): se obţine astfel: se adună 1 la reprezentarea în C1. C1 s-a folosit în primele calculatoare, iar C2 este reprezentarea standard folosită în calculatoarele de azi pentru numerele întregi. Reprezentarea C2 a lui 56 este: $\underline{0}0111000_2$, iar pentru +56 avem:

$$\begin{array}{ll} \text{-> C1} & \underline{1}1000111_2 \\ & \underline{+\qquad 1} \\ & \underline{1}1001000_2 \end{array}$$

Există o metodă rapidă de calcul a complementului faţă de 1, pentru a nu trece prin reprezentarea binară: se scade din F fiecare cifră din reprezentarea hexazecimală a numărului. +56 este 38h. Scăzând fiecare cifră din Fh obţinem C7h care se reprezintă ca 11000111_2 (acelaşi rezultat pentru C1).
Adunarea a două numere în C2 poate produce transport (Carry), dar acesta nu este folosit. Toate datele sunt reprezentate pe o lungime fixă (ca număr de biţi), deci adunarea a doi octeţi va produce ca rezultat tot un octet:

$$\begin{array}{ll} \underline{1}1111111_2 & \\ \underline{+\qquad 1} & \\ \underline{0}0000000_2 & \text{Carry=1} \end{array}$$

Ca o consecinţă, în reprezentarea C2 există o singură notaţie pentru zero. Vom vedea mai târziu cum detectăm acest transport (Carry), momentan este suficient de ştiut că nu este reprezentat în rezultat.
Prin folosirea sistemului C2, pe 8 biţi se pot reprezenta numere întregi de la -128 la +127; pe 16 biţi se pot reprezenta numerele între -32768 şi +32767.

Număr	*C2 pe 8 biţi*	*C2 pe 16 biţi*
+32767	nu se poate	7FFFh
-32768	nu se poate	8000h
-128	80h	FF80h
-1	FFh	FFFFh

UC nu ştie ce anume reprezintă un anumit octet (sau cuvânt). În limbaj de asamblare nu există tipuri de date ca în limbajele de nivel înalt. Modul de reprezentare al datelor este determinat de instrucţiunea folosită. Valoarea FF_{16} este considerată ca reprezentând -1 în reprezentarea cu semn sau 255 în reprezentarea fără semn, în funcţie de aplicaţie. Dacă vom considera numerele reprezentate pe trei biţi, vom avea în cele 3 convenţii valorile:

Număr binar	Modul şi semn	Complement faţă de 1	Complement faţă de 2
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	-0	-3	-4
101	-1	-2	-3
110	-2	-1	-2
111	-3	-0	-1

1.5 Reprezentarea datelor în memorie

Folosind un număr de 8 biţi se pot reprezenta 2^8=256 valori diferite. În figura 1.8 s-a ilustrat grafic *gama numerelor* pe sistemul de axe, cu poziţionarea celor 256 valori diferite ***fără semn*** versus ***cu semn*** care se pot scrie ***folosind un octet***.

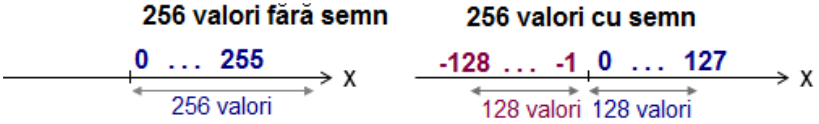

Fig. 1.8. Reprezentarea a 256 valori fără semn vs. cu semn

În timp, dimensiunile uzuale ale operanzilor în PC au fost: ***octet*** (în engleză *byte*), ***cuvânt*** (în engleză *word*), ***dublucuvânt*** (în engleză *doubleword*) şi ***cvadruplucuvânt*** (în engleză *quadword*), aşa cum apare în figura 1.9; pentru a fi cât mai uşor de urmărit, s-au reprezentat biţii grupaţi în octeţi.

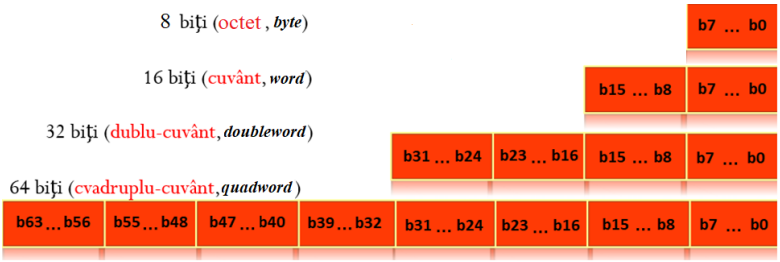

Fig. 1.9 Octetul şi multiplii uzuali ai octetului: cuvânt, dublucuvânt şi cvadruplucuvânt

Reprezentarea din figura 1.10 se referă la modul cum se depune în memorie octetul ***21h***, cuvântul ***43 21h***, şi respectiv dublucuvântul ***87 65 43 21h***, începând de la adresa 126 (ocupă 1 locaţie, 2 locaţii sau 4 locaţii succesive).

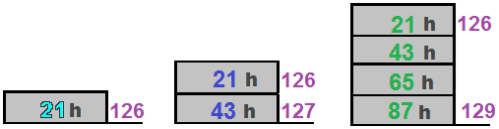

Fig. 1.10 Octetul 21h, cuvântul 4321h şi dublucuvântul 87654321h stocate în memorie începând de la adresa 126

Dublu-cuvântul 87654321h se depune în memorie folosind una din convenţiile:

- Little Endian: octetul **LSB**, adică cel de la sfârşitul structurii (“**END**”-ian) se depune în memorie la locaţia cu **adresa cea mai mică (“Little”)** – această convenţie este specifică procesoarelor din familia *Intel* – figura 1.11a);

- Big Endian: octetul **LSB** se depune în memorie la locaţia cu **adresa cea mai mare (“Big”),** convenţia fiind specifică procesoarelor din familia *Motorola* – figura 1.11b).

Adresa	Conţinutul		Adresa	Conţinutul	
0003	87 h	Dublucuvântul	**0003**	**21** h	
0002	65 h	**87.65.43.21**h	0002	43 h	
0001	43 h	în memorie de tip	0001	65 h	
0000	**21** h	**Little** sau **Big END**-ian	0000	87 h	
Little END-ian			**Big END-ian**		

Fig. 1.11 Depunerea dublucuvântului 87654321h în memorie după
a) convenţia Little Endian (stânga); b) convenţia Big Endian (dreapta)

Interacţiunea dintre utilizator și SC se realizează prin intermediul dispozitivelor de intrare-ieșire, de exemplu al tastaturii și al ecranului. Pentru a interacţiona cu acestea, SC vehiculează *coduri ASCII* atât la preluarea unui caracter de la tastatură cât și la afișarea unei valori pe ecran.

Exemple de coduri Ascii: cifrele 0...9 au codurile Ascii 30h ...39h, literele mici încep de la valoarea 61h ce corespunde lui ‚a’, iar literele mari încep de la valoarea 41h ce corespunde lui ‚A’.

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”.

Există o mare diferenţă între ***valori binare*** și ***coduri Ascii*** din punct de vedere al interpretării. 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 1.12 (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 hexazecimal, sau mai corect 0Ah și ‘A’. În primul caz, valoarea 0Ah este echivalentă numărului 10 (la adresa 07107h), pe când ‘A’ este caracterul având codul Ascii 41h (la adresa 07108h).

Adresa	in hexa	in zecimal	Cod Ascii caracter
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

Fig. 1.12. Exemple de valori interpretate ca numere binare sau coduri Ascii

Variabilele identifică datele, formând operanzi pentru instrucţiuni. Pentru declararea variabilelor se utilizează directive care alocă şi iniţializează memoria în unităţi de octeţi, cuvinte, dublu-cuvinte, astfel:

- directiva ***db*** *(Define Byte)* declară octeţi sau şiruri de octeţi;
- directiva ***dw*** *(Define Word)* declară cuvinte sau şiruri de cuvinte;
- directiva ***dd*** *(Define Doubleword)* declară dublucuvinte sau şiruri de dublucuvinte;

De exemplu, pentru obţinerea datelor aşa cum apar în figura 1.12 s-a folosit definirea unei variabile cu numele „sir” de tip octet, de forma:

sir db 0,1,2,'1','2', 0Ah, 'A','B'

1.6 Exerciţii şi teme

1. Se vor studia modalităţile de conversie a numerelor din bazele 2, 8,10, 16 şi operaţii aritmetice corespunzătoare.

2. Să se convertească următoarele numere din baza 10 în baza 8 şi 16:
   267; 1089; 530; 104; 708

3. Reprezentaţi în C2 pe 8 şi 16 biţi numerele:
   -204; -23; -67;

4. Să se calculeze în baza 16:

8Ah+	E6h-	-3*	5B16h+	22CAh+	3C51h-
56h	18h	6	8FEh	43Bh	2FDh

5. Realizaţi un desen care să ilustreze conţinutul memoriei dacă se va depune (după convenţia Little End-ian) dubucuvântul 12345678h, urmat de cuvântul ABCDh în memorie, începând de la adresa 102h; specificaţi şi adresa în hexazecimal.

6. Să se convertească următoarele numere din baza 10 în baza indicată, ţinând cont de faptul că sunt numere fără semn:
   a) 249; b) 251; c) 254; d) 247;
   Numărul: _____d = _____b = _____h

7. Scrieţi următoarele numere în binar folosind minim 8 biţi, considerând numerele fără semn:
a) 1= .....b; b) 7= .....b; c) 2516= .....b;
d) 1250=.....b; e) 258= .....b;

8. Repetaţi exerciţiul 6. considerând numerele cu semn.

9. Repetaţi exerciţiul 7. considerând numerele cu semn.

10. Scrieţi următoarele numere în hexazecimal folosind minim 2 cifre hexazecimale:
a) 1=.....h; b) 7=.....h; c) 2516=.....h; d) 1250=.....h; e) 258=.....h;

11. Scrieţi următoarele numere din binar în zecimal, ştiind că sunt numere fără semn:
a) 0000 0010b = ... ; b) 000000000001b = ...;
1111110000b = ...; 111110000001b = ...;
c) 000000000000111b = ...; d) 0000000000111b = ...;
10000000b = ...; 1010101010b = ...;

12. Scrieţi următoarele numere din binar în zecimal, ştiind că sunt numere cu semn:
a) 0000 0010b = ... ; b) 000000000001b = ...;
1111110000b = ...; 111110000001b = ...;
c) 000000000000111b = ...; d) 0000000000111b = ...;
10000000b = ...; 1010101010b = ...;

13. Scrieţi următoarele numere din binar în hexazecimal:
a) 0000 0010b = ... ; b) 000000000001b = ...;
1111110000b = ...; 111110000001b = ...;
c) 000000000000111b = ...; d) 0000000000111b = ...;
10000000b = ...; 1010101010b = ...;

14. Câţi octeţi se ocupă în memorie, dacă se folosește următoarea directivă?
a) a db 12h,34h,90h b) a dw -1,4,-4 c) a db 127,-128
b dw 56h,78h b db 2,-8 b dw -127,128

Nr. octeţi pt a: ____
Nr. octeţi pt b: ____

15. Scrieţi o directivă prin care să definiţi variabila *nume* de tip octet care să conţină șirul Ascii al caracterelor care determină prenumele dvs. Ilustraţi printr-un desen modul cum apare această variabilă în memorie.