

## Capitolul 1

### Introducere. Arhitectura 8086. Moduri de adresare

#### 1.1 Introducere. Necesitatea limbajului de asamblare

Limbajul de asamblare (ASM) permite înțelegerea la nivel de amănunt a ceea ce se întâmplă în realitate într-un calculator. Familiarizarea cu un asemenea limbaj este mai mult decât benefică pentru un programator, contribuind la eficiența programelor dezvoltate, indiferent de limbajul utilizat.

Există mai multe motive pentru care programarea în ASM este necesară. Codul generat în ASM se execută în general foarte rapid. Unele module de program trebuie implementate în ASM datorită acestei viteze de lucru. Uneori, o parte a unui program este scrisă într-un limbaj de nivel înalt, iar modulele critice sunt scrise ca proceduri ASM, apelate la rândul lor de modulele de nivel înalt.

Pe de altă parte, există situații în care e nevoie de acces direct la dispozitive de intrare/ieșire sau locații fizice de memorie, iar aceste operații nu pot fi executate din unele limbaje de nivel înalt. De exemplu, la calculatoarele personale, programele TSR și rutinele de tratare a întreruperilor sunt aproape totdeauna scrise în ASM.

Pe scurt, limbajul de asamblare oferă viteze de execuție și acces la hardware care nu pot fi disponibile (cel mai adesea) în limbajele de nivel înalt.

Un alt aspect importat este cel al dezvoltării de programe pentru echipamente dedicate. Nu toate programele executabile sunt destinate calculatoarelor de uz general. Dezvoltarea masivă a microprocesoarelor a făcut ca acestea să controleze în prezent funcționarea celor mai diverse obiecte tehnice, de la mașini de gătit sau de spălat rufe, până la echipamente de control industrial sau pentru comanda avioanelor. Toate aceste microprocesoare funcționează pe baza unor programe și nu trebuie să fii cine știe ce specialist ca să-ți dai seama că e puțin probabil ca un program care controlează un cuptor cu microunde sau un osciloscop să fi fost scris în PASCAL sau în BASIC. De fapt, imensa majoritate a programelor pentru asemenea echipamente dedicate este scrisă în limbaj de asamblare, pentru că într-un asemenea context, ceea ce contează este viteza de execuție și volumul foarte limitat de memorie.

Pe de altă parte, nu toate microprocesoarele sunt de uz general. Există microprocesoare specializate, destinate unor scopuri precise, cum ar fi procesoare de semnal, micro-controllere industriale, automate programabile etc. Pentru asemenea procesoare, nu se justifică (în general) dezvoltarea de compilatoare pentru limbaje de nivel înalt, programele fiind dezvoltate în limbajele specifice acestor procesoare, care sunt asemănătoare limbajelor de asamblare pentru procesoare de uz general.

Există și un puternic rol formativ al programării în limbaj de asamblare. Un programator nu ajunge niciodată la un nivel superior (de expert), dacă nu trece (măcar o dată în viață) prin dezvoltarea de programe ASM.

Cartea de față își propune să ofere cititorului un mijloc de a-și însuși limbajul de asamblare pentru procesoarele din familia Intel (IBM-PC). Ca mediu de dezvoltare, este folosită familia de produse software Borland (Turbo Assembler, Turbo Linker și Turbo Debugger).

O atenție sporită este acordată dezvoltării de aplicații mixte (în limbaje de nivel înalt și în ASM). Ca limbaj de nivel înalt tipic, s-a ales limbajul C, pentru care există medii de dezvoltare integrate oferite de firma Borland, total compatibile cu limbajul de asamblare.

În această carte nu se insistă prea mult pe interfața software cu sistemul BIOS sau cu sistemul DOS. Există multe manuale de firmă care descriu în detaliu funcțiile și subfuncțiile BIOS și DOS. Scopul acestei cărți este de a oferi mijloace de dezvoltare de module ASM eficiente și integrarea lor în aplicații complexe.

## **1.2 Noțiuni introductive de hardware. Registre. Stivă**

În ASM, calculatorul este văzut la nivelul hardware: adrese fizice de memorie, registre, întreruperi etc. Sunt necesare unele noțiuni pregătitoare.

Unitatea de bază a informației memorate în calculator este **bitul**. Un bit reprezintă o cifră binară (de aici și numele, care e o prescurtare de la binary digit), deci poate avea valorile 0 sau 1. Modelul hardware corespunzător este acela de **bistabil**. Un bistabil este deci un circuit electronic cu două stări stabile, codificate 0 și 1, capabil să memoreze un bit de informație.

Un grup de bistabili formează un **registru**. De exemplu, 8 bistabili formează un registru de 8 biți. Informația care se poate memora într-un asemenea registru poate fi codificată în binar, de la valoarea 00000000 (toți biții egali cu 0), până la valoarea 11111111 (toți biții egali cu 1). Este ușor de văzut că numărul combinațiilor care pot fi memorate este 256 (2 la puterea a 8-a). În general, un registru de n biți va putea memora  $2^n$  combinații distincte. Aceste combinații se numesc **octeți** sau **bytes** (dacă  $n = 8$ ), respectiv **cuvinte** (dacă  $n = 16, 32$  etc.). La procesoarele din familia Intel, cuvintele sunt de 16 sau 32 de biți (putând exista și cantități pe un număr mai mare de biți, dar toate multiplu de 8).

Memoria unui calculator este văzută ca o succesiune de octeți. Fiecare octet are asociată o adresă de memorie. Pentru a putea adresa memoria, e nevoie de un registru de adrese, a cărui lungime determină dimensiunea maximă a memoriei. Dacă avem un registru de adrese de 8 biți, atunci vom putea adresa  $2^8$  octeți de memorie. Procesorul 8086 are un registru de adrese de 20 de biți, deci poate adresa  $2^{20}$  octeți de memorie (sau 1 megaoctet de memorie). Zonele de memorie vor fi reprezentate grafic pe verticală, ca succesiuni de octeți sau cuvinte, de la adrese mici către adrese mari.

O adresă cu semnificație specială este adresa instrucțiunii care se execută în mod curent. Toate procesoarele au un registru destinat acestui scop, numit **contor program (Program Counter)**. Acest registru conține întotdeauna adresa instrucțiunii care urmează a fi executată.

O zonă specială de memorie este așa numita zonă de **stivă**. Stiva este un concept abstract de structură de date, dar procesoarele dispun de instrucțiuni mașină pentru operații cu o asemenea structură de date. O zonă de stivă este caracterizată de o adresă curentă (numită **adresa vârfului stivei**), adresată printr-un registru numit **SP (Stack Pointer, indicator de stivă)**. Operațiile de bază cu stiva sunt:

PUSH (X), care se poate descrie prin acțiunile:

$(SP) \leftarrow (SP) - 1$

$((SP)) \leftarrow X$

POP (X), care se poate descrie prin acțiunile:

$X \leftarrow ((SP))$

$(SP) \leftarrow (SP) + 1$

Notăția **(SP)** înseamnă "**conținutul lui SP**", iar **((SP))**, "**conținutul locației de memorie adresate de (SP)**". Din analiza celor două operații, se vede că structura de stivă este de tip "**Ultimul Intrat, Primul Ieșit**" sau "**Last In, First Out**", adică ultima cantitate care a fost "împinsă" în stivă printr-o operație **PUSH** va fi cea care se va extrage la următoarea operație **POP**. La procesorul 8086, cantitățile transferate în stivă sunt cuvinte de 16 biți, deci adresa curentă a vârfului stivei (conținută în SP) este adunată sau scăzută cu 2.

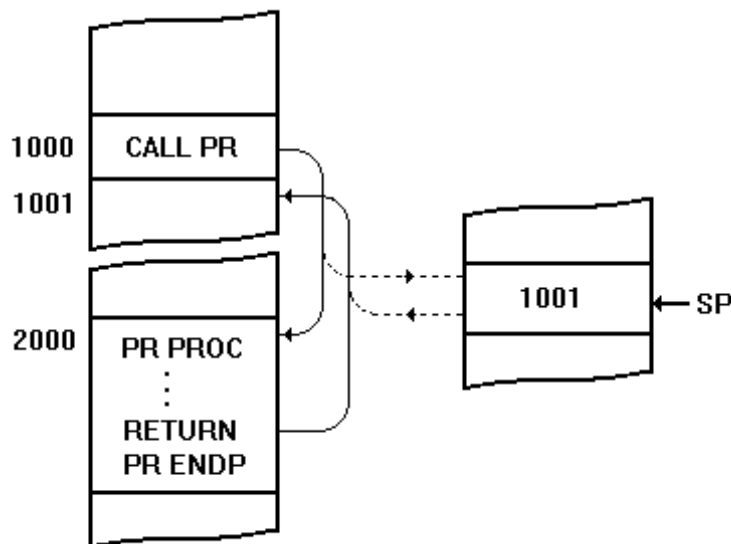
Stiva poate fi folosită explicit pentru salvări și refaceri de date. Ea este folosită implicit în mecanismul de apel de procedură. Când se apelează o procedură cu numele PR, printr-un apel generic de forma **CALL PR**, se va da controlul (adică se va produce un salt) la prima instrucțiune a procedurii. Acest salt nu înseamnă altceva decât că, în registrul contor program se va depune adresa primei instrucțiuni din procedură.

Se pune însă problema revenirii din procedură, la execuția unei instrucțiuni generice **RETURN**. Controlul trebuie să revină în programul apelant și să se execute instrucțiunea de după apelul de procedură.

Acest fapt este posibil deoarece la execuția instrucțiunii **CALL**, înainte de a se executa saltul la procedură, se salvează în stivă conținutul contorului program (adică adresa instrucțiunii imediat următoare apelului de procedură). Instrucțiunea **RETURN** nu face altceva decât să extragă această adresă din stivă și să o plaseze în contorul program, execuția continuând cu instrucțiunea de după apelul de procedură. Acest mecanism este ilustrat în Figura 1.1

Un program ASM (ca de altfel și unul în limbaj de nivel înalt) este organizat în trei spații de memorie:

- spațiul (zona) de cod - acesta cuprinde instrucțiunile care compun programul, sau codul;
- spațiul (zona) de date - acesta cuprinde datele statice definite în program (care se găsesc la adrese fixe);
- spațiul (zona) de stivă - acesta cuprinde stiva rezervată programului.



**Figura 1.1 Rolul stivei la apelul unei proceduri**

Zona de date poate fi detaliată în date constante, date inițializate sau neinițializate etc.

Trebuie remarcat că orice program executabil are structura de mai sus, indiferent de limbajul în care a fost scris, dar în ASM această structură este vizibilă și gestionată de către programator. Acesta poate chiar schimba semnificațiile zonelor respective. De exemplu, există situații (programe TSR) în care datele sunt definite în același spațiu de memorie cu instrucțiunile.

Pe lângă cele trei spații de memorie, mai există și așa numitul spațiu de memorie disponibilă. Acesta folosește la alocări dinamice de memorie (la momentul execuției programului) și este gestionat în mod indirect, prin apeluri către sistemul de operare.

### 1.3 Reprezentări interne ale datelor

#### 1.3.1 Baze de numerație

În tehnica de calcul, pe lângă baza de numerație 10, sunt folosite bazele de numerație 2, 8 și 16, iar sistemele de numerație respective se numesc binar, octal și hexazecimal. În sistemul hexazecimal, cifrele de la 10 la 15 se notează cu literele de la A până la F. Tabelul 1.1 ilustrează corespondența dintre sistemele zecimal, binar, octal și hexazecimal. Trebuie reținut faptul esențial că, în memoria calculatorului, informația de orice fel (date sau programe) este întotdeauna reprezentată în formă binară, deci ca secvențe de cifre 0 sau 1.

Deoarece este greu de operat cu numere mari în baza 2, pentru exprimarea unor cantități binare se folosesc bazele 8 și 16. De exemplu, numărul 255 se poate exprima în aceste baze prin  $255_{(10)} = 11111111_{(2)} = FF_{(16)} = 377_{(8)}$ .

Informația este organizată în calculator pe grupe de câte 8 cifre binare (biți). Un asemenea grup se mai numește și octet (sau byte). Octetul este unitatea de măsură în care se exprimă volumul memoriei unui calculator.

Zecimal	Binar	Octal	Hexazecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

**Tabelul 1.1 Reprezentarea numerelor în bazele 10, 2, 8 și 16**

Deoarece dimensiunea memoriei este totdeauna o putere a lui 2, multiplii folosiți pentru octeți nu sunt 100, 1000 etc., ci puteri adecvate ale lui 2. Astfel, un număr de  $2^{10} = 1024$  de octeți se mai numește și kilooctet sau kilobyte (pe scurt KO sau KB). Similar, un număr de  $2^{20} = 10264576$  octeți se numește megaoctet sau megabyte (MO sau MB). Se mai folosește și gigaoctetul (gigabyte-ul), care este egal cu 230 octeți. Așadar, multiplii folosiți în tehnica de calcul sunt:

- kilo =  $2^{10} = 1024$
- mega =  $2^{20} = 10264576$
- giga =  $2^{30} = 1073741824$

### 1.3.2 Reprezentarea numerelor întregi

Numerele întregi pot fi reprezentate pe unul sau mai mulți octeți, deci pe un număr  $n$  de biți. Deoarece fiecare bit poate lua două valori (0 și 1), numărul total de valori distincte este  $2^n$ .

În cazul numerelor fără semn, valoarea internă a biților (octeților) reprezintă chiar valoarea numărului. Este ușor de văzut că, dacă reprezentarea este pe  $n$  biți, domeniul posibil de valori este de la 0 la  $2^n - 1$ . Pentru  $n = 3$  rezultă domeniul din Tabelul 1.2.

În ceea ce privește reprezentarea numerelor cu semn, trebuie spus că valorile interne care se pot reprezenta (deci configurațiile de biți) sunt

aceleași ca la numere fără semn. Aceste configurații variază între 0000...0000 (toți biții egali cu 0) și 1111...1111 (toți biții egali cu 1).

Se pune deci problema ca, printr-o convenție adecvată, să se considere o parte din aceste configurații de biți ca reprezentând numere întregi pozitive, iar cealaltă parte numere negative.

<b>Reprezentare internă</b>	<b>Valoare</b>
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

**Tabelul 1.2 Reprezentarea numerelor fără semn pe 3 biți**

Exista mai multe sisteme de reprezentare, dintre care cel mai răspândit este sistemul de reprezentare în **complement față de 2**. În acest sistem, bitul cel mai semnificativ joacă un rol special, anume de a preciza semnul numărului (din acest motiv se numește bit de semn). Dacă bitul de semn este 0, numărul reprezentat este pozitiv, iar dacă bitul de semn este 1, numărul este negativ.

În ceea ce privește valoarea absolută (modulul) numărului, ea se obține diferențiat, funcție de semnul numărului, în felul următor:

- dacă bitul de semn este 0, atunci configurația internă reprezintă chiar valoarea numărului;
- dacă bitul de semn este 1, atunci valoarea absolută a numărului se obține prin complement față de 2, adică prin complementarea tuturor biților (0 devine 1 și reciproc) și prin adunarea apoi a valorii 1.

Această regulă se aplică pentru ambele sensuri de conversie (de la reprezentări interne la numere cu semn și reciproc).

De exemplu, fie numărul de biți  $n = 3$  și să considerăm reprezentarea 111. Bitul de semn este 1, deci este vorba de un număr negativ. Prin complementare se obține 000, iar prin adunare cu 1 se obține 001, adică valoarea absolută 1. Astfel, reprezentarea binară 111 pe 3 biți corespunde numărului -1.

Reprezentarea în complement față de 2 pe  $n$  biți are următoarele proprietăți generale:

- domeniul de reprezentare este  $-2^{n-1} \dots 2^{n-1}-1$ ;
- există o singură reprezentare pentru 0, anume 000...000;
- bitul cel mai semnificativ este bit de semn;
- calculul valorilor absolute pentru numere negative se face prin aplicarea regulii complementului față de 2.

În Tabelul 1.3 sunt ilustrate domeniile de valori pentru numere cu semn pe  $n=3$  biți.

Să remarcăm faptul important ca o aceeași reprezentare internă poate avea semnificații (interpretări) diferite, după cum se consideră reprezentarea cu semn sau fără semn. De exemplu, valoarea internă 1010 este interpretată ca -6 în reprezentarea cu semn și ca +10 în reprezentarea fără semn pe 4 biți. Se vede că diferența dintre aceste două valori posibile este chiar  $2n$  (în cazul de față 16).

De aici se poate deduce o altă regulă pentru aflarea rapidă a reprezentării numerelor negative: se adună  $2n$  iar valoarea obținută se reprezintă ca număr fără semn.

Reprezentare internă	Valoare
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

**Tabelul 1.3 Reprezentarea numerelor cu semn pe 3 biți**

De exemplu, ca să obținem reprezentarea lui -120 pe 8 biți, adunăm 256 și obținem valoarea pozitivă 136. Reprezentarea lui 136 în baza 2 va fi atunci reprezentarea în complement față de 2 a lui -120, adică 10001000 sau 88H.

În diverse sisteme de afișare a datelor numerice, în care este folosită numai baza 10, se utilizează codificarea de tip **BCD** care înseamnă "**Zecimal Codificat în Binar**" (în engleză, "**Binary Coded Decimal**").

În acest sistem, se reprezintă o cifră zecimală pe un grup de 4 biți (4 biți pot codifica 16 valori distincte, deci 6 valori nu vor fi folosite). Un octet va codifica deci 2 cifre zecimale. De exemplu, octetul cu valoarea 01011001 reprezintă codificarea BCD pentru numărul zecimal 59 (primul grup de 4 biți reprezintă cifra 5, iar al doilea grup cifra 9). Dacă este necesar, se pot considera  $n$  octeti pentru memorarea a  $2n$  cifre zecimale.

Calculatoarele dispun de obicei de instrucțiuni speciale pentru calcule în format BCD. Sunt unele probleme care trebuie avute în vedere, și anume operațiile de corecție care trebuie făcute după (sau înaintea) operațiilor în BCD, deoarece nu toate cele 16 combinații binare posibile sunt cifre BCD corecte.

Formatul BCD se mai numește și **BCD împachetat**, pentru a-l deosebi de formatul **BCD despachetat**, în care se reprezintă o cifră BCD pe un octet. În terminologia firmei INTEL (reflectată în denumirile instrucțiunilor procesoarelor), formatul BCD despachetat este denumit (oarecum incorect) format ASCII. Justificarea denumirii este că, prin adăugarea valorii 30H (codul ASCII al cifrei 0), se obține codul ASCII al cifrei reprezentate de octetul respectiv.

Toate cele trei sisteme de reprezentare a numerelor întregi ilustrează faptul că datele din memoria calculatorului se pot interpreta diferit. Aceste interpretări sunt făcute de programul care folosește datele. De exemplu octetul 10010100 poate reprezenta numărul zecimal pozitiv 148, numărul zecimal negativ -108 sau numărul zecimal pozitiv 94, după cum se lucrează cu sistemul de reprezentare fără semn, cu semn sau, respectiv, cu sistemul BCD.

### 1.3.3 Reprezentarea numerelor reale

Numerele reale (fracționare) se pot reprezenta în două moduri distincte, care sunt numite reprezentare în virgulă fixă și reprezentare în virgulă mobilă.

#### *Reprezentarea în virgulă fixă*

În reprezentarea în virgulă fixă, se consideră un număr finit de cifre semnificative, atât pentru partea întreagă a numărului, cât și pentru cea fracționară. Considerând baza de numerație  $b$ ,  $n$  cifre pentru partea întreagă și  $m$  cifre pentru partea fracționară, un număr fracționar  $x$  se reprezintă prin expresia:

$$x = i_n \cdot b^n + \dots + i_1 \cdot b^1 + i_0 \cdot b^0 + f_1 \cdot b^{-1} + \dots + f_m \cdot b^{-m}$$

în care cifrele  $i_n \dots i_0$  reprezintă partea întreagă, iar cifrele  $f_1 \dots f_m$  reprezintă partea fracționară a numărului  $x$ . Cifrele sunt considerate în baza  $b$ . Un asemenea număr se scrie pozițional în forma:

$$i_n \dots i_1 i_0 . f_1 \dots f_m (b)$$

(în tehnica de calcul, virgula se notează cu un punct). Exemple de astfel de numere fracționare pot fi:

$$123.25_{(10)} \quad 1AB34.FF52_{(16)} \quad 1000110.11011_{(2)}$$

Într-o reprezentare internă concretă (pe un sistem de calcul), baza  $b$  și numărul de cifre de la stânga și de la dreapta virgulei ( $n$  și  $m$ ) sunt fixate aprioric. De exemplu, lucrând cu baza 10 și presupunând că o precizie de 0.001 este suficientă, se vor memora doar 3 cifre după virgulă. Similar, dacă se presupune că numerele care trebuiesc reprezentate nu depășesc valoarea 10000, se vor alocă 4 cifre la stânga virgulei. Se vor reprezenta astfel numere zecimale în domeniul 0.001 ... 9999.999, cu o precizie de 0.001.

Cifrele respective vor fi numite mii, sute, zeci, unități, zecimi, sutimi, miimi, etc. și se vor reprezenta fie pe câte un octet, fie pe câte 4 biți. Memorarea semnului unui astfel de număr se face într-un câmp separat. Tabelul 1.4 exemplifică reprezentarea în virgulă fixă a numerelor 1.130, -1230.192 și 10.000.



Sem n	Mi i	Sute	Zec i	Unităţ i	Zeci mi	Suti mi	Miimi
+	0	0	0	1	0	1	3
-	1	2	3	0	1	9	2
+	0	0	1	0	0	0	0

**Tabelul 1.4** Reprezentarea în virgulă fixă ( $b = 10$ ,  $n = 4$ ,  $m = 3$ )

Se observă că punctul zecimal nu se reprezintă, deoarece poziția sa este cunoscută (este fixă, între coloanele unităților și zecimilor). De altfel, numele metodei de reprezentare provine de la această proprietate.

Uneori, se reprezintă numărul real respectiv înmulțit cu 10 la o putere egală cu numărul de cifre după virgulă. De exemplu, numărul 1.130 se poate reprezenta intern ca numărul întreg 1130, iar numărul -1230.192 ca numărul întreg -1230192 (în complement față de 2, pentru a memora și semnul). În operațiile aritmetice care se vor face, se va ține seama că numerele sunt înmulțite cu 1000 și se va reprezenta punctul zecimal după a treia cifră de la dreapta la stânga.

Trebuie remarcat că, într-o astfel de reprezentare cu număr fix de câmpuri, nu se poate reprezenta decât un subset finit al numerelor reale, în exemplul de mai sus acest subset fiind cuprins între -9999.999 și +9999.999, cu pasul 0.001. Trebuie reținut de asemenea că toate valorile din subsetul respectiv sunt reprezentate exact.

De asemenea, este evident faptul că operațiile aritmetice de înmulțire și împărțire se fac într-un mod aproximativ. Înmulțind 1.121 cu 2.250 se obține valoarea teoretică 2.52225, dar aceasta nu se poate reprezenta în sistemul cu 3 cifre după punctul zecimal. Ca atare, rezultatul este rotunjit la valoarea cea mai apropiată care se poate reprezenta, anume 2.522. Operațiile de adunare și scădere se fac exact, cu condiția ca rezultatul să nu depășească limitele domeniului de reprezentare.

În concluzie, reprezentarea în virgulă fixă este caracterizată de numărul de cifre reprezentate și de numărul de cifre după virgulă (poziția virgulei).

Reprezentarea în virgulă fixă se folosește în unele sisteme de conducere cu calculator a mașinilor-unelte industriale (sisteme de poziționare) și - foarte important - în sistemele de programe financiare și contabile.

La acestea din urmă, reprezentarea în virgulă fixă are avantajul că toate puterile lui 10 (pozitive și negative) care fac parte din domeniul de valori sunt reprezentate exact.

De exemplu, pentru a reprezenta sume de bani, este suficient să avem două cifre după virgulă. La un bilanț financiar, când se adună sau se scad sume de bani, este esențial ca operațiile de adunare și scădere să se facă exact.

Din motivele prezentate, în programele aplicative pentru finanțe (în general pentru domeniul economic) se folosește sistemul de reprezentare în virgulă fixă.

**Reprezentarea în virgulă mobilă**

Reprezentarea în virgulă mobilă se folosește cu precădere în domeniile științifice și tehnice (sau, altfel spus, în toate domeniile în afară de cele economico-financiare).

În această reprezentare, numerele sunt considerate de forma:

$$(-1)^S \cdot M \cdot B^E$$

unde:

- S este numit **bit de semn** și este 1 pentru numere negative și 0 pentru numere pozitive;
- M este numită **mantisă** (sau **fracție**) și este un număr pozitiv subunitar reprezentat în baza B;
- B este numită **bază** (uzual este 2 sau 16);
- E este numit **exponent** și este un număr întreg cu semn.

Mantisa M se numește **normalizată**, dacă prima cifră după virgulă este cifră semnificativă (este diferită de zero).

Majoritatea sistemelor actuale folosesc baza  $B = 2$ . În acest caz, faptul că prima cifră a mantisei este diferită de 0 înseamnă că aceasta este obligatoriu 1. Astfel, mantisa M verifică condiția (scrisă în baza 2) :

$$0.1_{(2)} \leq M < 1$$

sau, în baza 10:

$$0.5_{(10)} \leq M < 1$$

Exponentul se reprezintă de obicei deplasat, în sensul că se memorează un număr de forma  $E + K$ , unde K este o constantă astfel aleasă încât  $E+K$  să fie totdeauna pozitiv. În felul acesta se rezolvă problema memorării semnului exponentului. De cele mai multe ori, deoarece prima cifră semnificativă a mantisei M este 1, această valoare nu se mai reprezintă, câștigându-se astfel un bit în spațiul de memorare (în care se poate memora bitul de semn S).

Cea mai des folosită reprezentare standardizată este cea numită în **simplă precizie**, descrisă în cele ce urmează. În acest sistem, mantisa M are 24 de biți, iar exponentul E are 8 biți și se reprezintă intern deplasat (adunat) cu 127. Mantisa este normalizată și înmulțită cu 2, astfel încât condiția de normalizare este  $1 \leq M < 2$ .

Bitul cel mai semnificativ al mantisei nu se mai reprezintă intern, deoarece este totdeauna 1. Astfel, un număr real în simplă precizie se reprezintă pe 32 de biți (4 octeți).

Notând cu f mantisa (fracția) care se reprezintă intern, cu e exponentul deplasat și cu s bitul de semn, valoarea unui număr real este dată de relația:

$$(-1)^S \cdot 1.f_{22}f_{21}...f_1f_0 \cdot 2^{e-127}$$

În care  $s$  este bitul de semn,  $e = e_7e_6...e_1e_0$  este exponentul deplasat cu 127, iar  $f = f_{22}f_{21}...f_1f_0$  este mantisa (fracția) normalizată. Bitul cu valoarea 1 din formula de mai sus (bitul 23 al mantisei) nu se reprezintă intern. Cei patru octeți sunt reprezentați în Tabelul 1.5, în ordinea așezării lor în memorie (în sens crescător al adreselor de memorie).

Nu toate combinațiile posibile de biți din tabel sunt reprezentări valide. Astfel, numărul 0.0 se reprezintă în mod unic prin  $s = 0$ ,  $f = 0$  și  $e = 0$ , aceasta fiind singura valoare pentru care  $e$  sau  $f$  pot fi nule. În afara valorii reale 0.0, exponentul deplasat trebuie să fie cuprins între 1 și 254.

Exponentul 0 este permis numai pentru reprezentarea valorii reale 0.0 (deci și cu  $f = 0$  și  $s = 0$ ). Exponentul 255 (0FFH) este folosit pentru reprezentarea unor valori de excepție. Astfel, reprezentările cu  $e = 0FFH$ ,  $f = 7FFFFFFH$  (toți biții 1) și  $s = 0$  sau 1 sunt folosite pentru valorile notate **+INF** și **-INF**, care sunt cazuri de excepție (depășiri aritmetice). Orice alte valori cu exponent 0 sau 255 sunt incorecte. Aceste valori incorecte mai sunt numite și **NAN** (de la **Not A Number**). Funcție de bitul de semn, pot exista valori +NAN sau -NAN.

f7	f6	f5	f4	f3	f2	f1	f0
f15	f14	f13	f12	f11	f10	f9	f8
e0	f22	f21	f20	f19	f18	f17	f16
s	e7	e6	e5	e4	e3	e2	e1

**Tabelul 1.5 Reprezentarea în virgulă mobilă în simplă precizie**

Alte valori reprezentative sunt:

- **cel mai mic număr pozitiv reprezentabil** este caracterizat de  $s = 0$ ,  $f = 0$ ,  $e = 1$  și are valoarea aproximativă  $1.17 \cdot 10^{-38}$ ;
- **cel mai mare număr pozitiv reprezentabil** este caracterizat de  $s = 0$ ,  $f = 7FFFFFF$ ,  $e = 0FEH$  și are valoarea aproximativă  $3.4 \cdot 10^{38}$ .

Să considerăm câteva exemple de reprezentare.

- Numarul real 0.0 se reprezinta prin 4 octeți nuli.
- Numărul real 1 se exprimă prin semnul  $s = 0$ , mantisa 1.0 și exponentul 0 ( $1 = (-1)^0 \cdot 1.0 \cdot 2^0$ ). Frația internă este  $f = 0$ , iar exponentul deplasat este  $e = 127$  (sau  $e = 7FH$ ).
- Numărul real 2 se exprimă prin semnul  $s = 0$ , mantisa 1.0 și exponentul 1 ( $2 = (-1)^0 \cdot 1.0 \cdot 2^1$ ). Frația internă este  $f = 0$ , iar exponentul deplasat este  $e = 128$  (sau  $e = 80H$ ).
- Numarul real 0.5 se exprimă prin semnul 0, mantisa 1.0 și exponentul -1 ( $0.5 = (-1)^0 \cdot 1.0 \cdot 2^{-1}$ ). Frația internă este  $f = 0$ , iar exponentul deplasat este  $e = 126$  (sau  $e = 7EH$ ).
- Numarul real -1 se exprimă prin semnul 1, mantisa 1.0 și exponentul 0 ( $-1 = (-1)^1 \cdot 1.0 \cdot 2^0$ ). Frația internă este  $f = 0$ , iar exponentul deplasat este  $e = 127$  (sau  $e = 7FH$ ).

Scriind bitul  $s$ , biții fracției  $f$  și ai exponentului deplasat  $e$  ca în Tabelul 2.4, se obțin reprezentările pe 4 octeți de mai jos:

- $0.0 = 00\ 00\ 00\ 00$  ( $s = 0$ ,  $f = 0$ ,  $e = 0$ )
- $1.0 = 00\ 00\ 80\ 3F$  ( $s = 0$ ,  $f = 0$ ,  $e = 7FH$ )
- $2.0 = 00\ 00\ 00\ 40$  ( $s = 0$ ,  $f = 0$ ,  $e = 80H$ )

- $0.5 = 00\ 00\ 00\ 3F$  ( $s = 0, f = 0, e = 7EH$ )
- $-1.0 = 00\ 00\ 80\ BF$  ( $s = 1, f = 0, e = 7FH$ )

Reprezentarea în virgulă mobilă este caracterizată prin numărul de cifre al mantisei (fracției) și numărul de cifre al exponentului.

Să observăm că, la aceeași mantisă, putem avea exponenți diferiți, ceea ce înseamnă că virgula nu mai este pe o poziție fixă, ci se deplasează după cât de mare este exponentul. Acest fapt a condus la numele reprezentării (virgulă mobilă). Avantajul acestei reprezentări este domeniul foarte mare. În simplă precizie putem reprezenta numere cuprinse aproximativ între  $-10^{38}$  și  $10^{38}$ .

Trebuie de asemenea remarcat că se reprezintă corect numai numerele care se pot exprima ca sume de puteri (pozitive sau negative) ale lui 2 (în general ale bazei B). Exemplele de mai sus sunt în această categorie. Numerele 0.1, 0.01, 0.3 nu se reprezintă exact în virgulă mobilă cu baza 2. De aceea, acest sistem nu se folosește în programe pentru domeniile economico- financiare.

Calculule aritmetice sunt aproximative, dar precizia este de cele mai multe ori suficientă pentru aplicațiile uzuale. În reprezentarea în simplă precizie (pe 4 octeți), precizia asigurată este  $10^{-7}$ .

Reprezentarea în precizie dublă (8 octeți) este caracterizată de o mantisă normalizată pe 53 de biți și un exponent pe 11 biți (deplasat cu 1023). Valoarea unui număr real în dublă precizie este dată de expresia:

$$(-1)^s \cdot 1.f_{51}f_{50}...f_1f_0 \cdot 2^{e-1023}$$

În care  $s$  este bitul de semn,  $e = e_{10}e_9...e_1e_0$  este exponentul deplasat cu 1023, iar  $f = f_{51}f_{50}...f_1f_0$  este mantisa (fracția) normalizată. Bitul cu valoarea 1 din formula de mai sus (bitul 52 al mantisei) nu se reprezintă intern. Structura celor 8 octeți (în ordinea crescătoare a adreselor) este cea din Tabelul 1.6 .

Valorile corecte pentru exponentul deplasat sunt cuprinse între 1 și 1022. Numărul 0.0 se prezintă în mod unic cu  $s = f = e = 0$  iar valorile cu exponent 1023 sunt folosite pentru +INF și -INF.

f7	f6	f5	f4	f3	f2	f1	f0
f15	f14	f13	f12	f11	f10	f9	f8
f23	f22	f21	f20	f19	f18	f17	f16
f31	f30	f29	f28	f27	f26	f25	f24
f39	f38	f37	f36	f35	f34	f33	f32
f47	f46	f45	f44	f43	f42	f41	f40
e3	e2	e1	e0	f51	f50	f49	f48
s	e10	e9	e8	e7	e6	e5	e4

**Tabelul 1.6 Reprezentarea în virgulă mobilă în dublă precizie**

Reprezentarea în precizie extinsă (10 octeți) este caracterizată de o mantisă normalizată pe 64 de biți și un exponent pe 15 biți (deplasat cu 16383). Valoarea unui număr real în precizie extinsă este dată de expresia:

$$(-1)^s \cdot 1. f_{62}f_{61}...f_1f_0 \cdot 2^{e-16383}$$

în care s este bitul de semn,  $e = e_{14}e_{13}...e_1e_0$  este exponentul deplasat cu 16383, iar  $f = f_{62}f_{61}...f_1f_0$  este mantisa (fracția) normalizată. Bitul cu valoarea 1 din formula de mai sus (bitul 63 al mantisei) se reprezintă intern ca un bit totdeauna egal cu 1, cu excepția valorii zero, când acest bit este 0. Structura celor zece octeți (în ordinea crescătoare a adreselor) este cea din Tabelul 1.7.

f7	f6	f5	f4	f3	f2	f1	f0
f15	f14	f13	f12	f11	f10	f9	f8
f23	f22	f21	f20	f19	f18	f17	f16
f31	f30	f29	f28	f27	f26	f25	f24
f39	f38	f37	f36	f35	f34	f33	f32
f47	f46	f45	f44	f43	f42	f41	f40
f55	f54	f53	f52	f51	f50	f49	f48
1	f62	f61	f60	f59	f58	f57	f56
e7	e6	e5	e4	e3	e2	e1	e0
s	e14	e13	e12	e11	e10	e9	e8

**Tabelul 1.7 Reprezentarea în virgulă mobilă în precizie extinsă**

Microprocesoarele din familia 80x86 sunt dotate cu circuite specializate (coprocesoare matematice) care folosesc cele trei tipuri de reprezentări în virgulă mobilă de mai sus. Deasemenea, limbajul de asamblare specific acestor procesoare dispune de directive pentru definirea de constante reale în toate cele trei precizii.

#### 1.3.4 Reprezentarea datelor nenumerice

Pe lângă date numerice, în memoria calculatorului trebuie reprezentate și alte tipuri de date. Un exemplu important îl reprezintă caracterele alfabetice (litere mari și mici) caracterele numerice (cifre zecimale), semnele de punctuație și așa-numitele caractere de control (folosite pentru comanda diverselor echipamente periferice).

Deși există mai multe sisteme de codificare a acestor tipuri de date, pe marea majoritate a calculatoarelor actuale (inclusiv pe IBM-PC), se folosește codul standard **ASCII**. Denumirea sa provine de la inițialele **American Standard Code for Information Interchange** (**Codul Standard American pentru Schimb de Informații**).

Codul ASCII standard este un cod pe 7 biți, deci cuprinde 128 de caractere distincte. Un caracter ASCII se reprezintă pe un octet, în care bitul cel mai semnificativ este 0. Domeniul de valori este deci de la 0 la 127 (în zecimal), sau de la 00 la 7F (în hexazecimal).

Dintre cele 128 de caractere, 32 sunt caractere de control și nu au reprezentări grafice (nu sunt afișabile). Restul de 96 de caractere poate fi afișat pe ecranul calculatorului, la imprimantă etc. Caracterele afișabile se notează de obicei prin scrierea simbolului grafic respectiv între apostroafe. De exemplu, 'A' înseamnă codul ASCII corespunzător literei A mare, '!' înseamnă codul ASCII pentru semn de exclamare etc.

Calculatorul IBM-PC lucrează cu un așa-numit cod ASCII extins, în care se folosesc toți cei 8 biți ai unui octet. Setul ASCII standard este un subset al acestui cod ASCII extins.

Tabelul 1.8 cuprinde toate codurile ASCII. Coloanele corespund primei cifre hexa, iar liniile, celei de-a doua cifre. Primele două coloane (în afară de codul 20H) și ultimul cod (DEL sau 7F) reprezintă cele 32 de caractere de control.

Deoarece caracterele de control nu au reprezentări grafice, ele au nume speciale (exprimate de obicei prin prescurtări de 2 sau 3 litere). Dintre caracterele de control uzuale, pot fi amintite:

- **CR (Carriage Return)**. Este interpretat de perifericele de ieșire, provocând mutarea cursorului pe ecran la începutul liniei curente. La imprimantă, provoacă mutarea carului de tipărire la începutul liniei;
- **LF (Line Feed)**. Provoacă trecerea la linie nouă (atât la imprimantă cât și pe ecran);
- **TAB (Horizontal Tabulation)**. Provoacă deplasarea cursorului pe ecran sau a carului de tipărire cu un număr predefinit de poziții la dreapta, sau într-o poziție predefinită;
- **BS (Backspace)**. Provoacă deplasarea cursorului cu o poziție înapoi (la stânga) sau ștergerea caracterului de la stânga cursorului;
- **DEL (Delete)**. Provoacă ștergerea caracterului indicat de cursor;
- **BEL (Bel)**. Provoacă emiterea unui sunet de avertizare;

- **FF (Form Feed)**. Provoacă avansul hârtiei (formularului) la pagină nouă (la imprimantă).

O serie de coduri de control se folosește în transmisiile de date, având semnificații specifice. De exemplu, **ACK (Acknowledge)** se folosește pentru confirmarea transmisiei corecte a unui bloc de date, iar **NAK (Negative Acknowledgement)** pentru semnalarea unei transmisii incorecte.

$2^4 2^5 2^6 \Rightarrow$ $2^3 2^2 2^1 2^0$ ↓	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

**Tabelul 1.8 Codul ASCII standard**

Dintre caracterele tipăribile uzuale, se pot aminti:

- Literele mari 'A' ... 'Z'
- Literele mici 'a' ... 'z'
- Cifrele zecimale '0' ... '9'
- Spatiul (SP), care are codul 20H și care este tipărit ca un spațiu liber (de fapt nu se tipărește nimic, ci se lasă o pauză); Caracterul SP (spatiu, ' ') se numeste si blank.

Valorile numerice ale codurilor ASCII se exprimă prin 2 cifre hexazecimale, în care prima cifra este limitată la 7).

## 1.4 Tipuri de date utilizate în limbaj de asamblare

Limbajul de asamblare 80x86 operează cu anumite tipuri de date de bază. Aceste tipuri sunt recunoscute de către procesor și fac parte integrantă din formatul anumitor instrucțiuni mașină. De asemenea, limbajul dispune de directive specifice pentru definirea acestor tipuri de date.

Caracteristic pentru un tip de date este domeniul de valori, care rezultă atât din tipul de date în sine, cât și dintr-o eventuală interpretare funcție de context. Tipurile de date sunt:

### **BYTE (1 octet)**

Acest tip de date ocupă 1 octet și poate fi reprezentat atât în memoria internă cât și într-un registru de 8 biți al procesorului. Interpretările tipului byte pot fi:

- întreg pe 8 biți cu sau fără semn;
- caracter ASCII.

Directiva pentru definirea datelor de acest tip este DB (Define Byte).

### **WORD (2 octeți)**

Acest tip de date ocupă 2 octeți și poate fi reprezentat atât în memoria internă cât și într-un registru de 16 biți al procesorului. Interpretările tipului word pot fi:

- întreg pe 16 biți cu sau fără semn;
- secvență de două caractere ASCII;
- adresă de memorie de 16 de biți.

Directiva pentru definirea datelor de acest tip este DW (Define Word). Partea mai puțin semnificativă este memorată la adrese mici. De exemplu, dacă presupunem întregul 1234H la adresa 1000H, atunci octetul 34H se va găsi la adresa 1000H iar octetul 12H la adresa 1001H. Similar se memorează și secvențele de două caractere ASCII.

### **DOUBLE-WORD (4 octeți)**

Acest tip de date ocupă 4 octeți și poate fi reprezentat atât în memoria internă, cât și într-o pereche de registre de 16 biți sau într-un registru de 32 de biți (la procesoarele de 32 de biți). Interpretările tipului dword pot fi:

- întreg pe 32 de biți cu sau fără semn;
- număr real în simplă precizie;
- adresă de memorie de 32 de biți.

Directiva pentru definirea datelor de acest tip este DD (Define Double-Word). Valorile mai puțin semnificative se memorează la adrese mici. În cazul adreselor pe 32 de biți, adresa de segment este memorată la adrese mari, iar deplasamentul (offsetul), la adrese mici.

### **QUAD-WORD (8 octeți)**

Acest tip de date ocupă 8 octeți și poate fi reprezentat atât în memoria internă, cât și într-o pereche de registre de 32 de biți (numai la procesoarele de 32 de biți). Interpretările tipului qword pot fi:



- întreg pe 64 de biți cu sau fără semn;
- număr real în dublă precizie;

Directiva pentru definirea datelor de acest tip este DQ (Define Quad-Word).

### **TEN-BYTES (10 octeți)**

Acest tip de date ocupă 10 octeți și poate fi reprezentat atât în memoria internă cât și într-unul din registrele coprocesoarelor matematice 80x87. Interpretările tipului tbyte pot fi:

- număr întreg reprezentat ca secvență de cifre BCD (împachetate), cu semn memorat explicit;
- număr real în precizie extinsă.

Directiva pentru definirea datelor de acest tip este DT (Define Ten-Bytes).

În reprezentarea întregilor, se consideră numere cu maxim 19 cifre zecimale. Un grup de două cifre se reprezintă pe 1 octet, partea mai puțin semnificativă fiind la adrese mici. În cadrul unui octet, cifra BCD mai semnificativă este în partea high a octetului. Cele 19 cifre BCD ocupă așadar 76 de biți. Ultimul grup de 4 biți (partea high a octetului aflat la adresa cea mai mare) este destinat memorării semnului. De fapt, semnul se memorează doar pe bitul cel mai semnificativ al acestui ultim octet. Asambloarele acceptă și numere cu 20 de cifre zecimale, atât tip cât cifra cea mai semnificativă, care se va reprezenta pe ultimul grup de 4 biți, nu intră în conflict cu bitul de semn.

Teoretic, valoarea maximă corect reprezentabilă este:

+ 9999 99999 99999 99999

iar valoarea minimă este:

- 9999 99999 99999 99999

dar se acceptă și valori de genul:

+ 79999 99999 99999 99999  
- 79999 99999 99999 99999

În care cifra cea mai semnificativă este reprezentată doar pe 3 biți.

Din motive de siguranță, este bine să ne limităm la valorile garantate, adică la cel mult 19 cifre zecimale și semn.

Să considerăm următorul exemplu de program:

```
.model small
.data
    b1      db      -1, 10, 17H, 0FFH
    b3      db      'a', 'b'
    b2      db      "abcdef", 0

    w1      dw      1234H, -1, 'AB'
    w2      dw      w1

    d1      dd      12345678H, -1
    d2      dd      1.0, -1.0, 0.5
    d3      dd      d1
```

```

        q1    dq    1000000000000002H, -1
        q2    dq    1.0, -1.0

        t1    dt    1234567890000012345
        t2    dt    -1234567890000012345
        t3    dt    999999999999999999
        t4    dt    -999999999999999999
        t5    dt    799999999999999999
        t6    dt    -799999999999999999
        t7    dt    1.0
end

```

În acest exemplu este definit doar un modul de date, folosind directivele și tipurile de date existente în limbaj. Liniile care încep cu un punct sunt directive care stabilesc modelul de memorie (.model) și, respectiv, definesc un segment de date (.data). Directiva end marchează sfârșitul programului.

Se definesc date cu cele 5 tipuri de directive (db, dw, dd, dq, dt), asociindu-se acestora și nume simbolice (b1, w2 etc.). Fișierul listing obținut în urma asamblării acestui program are (între altele) următorul conținut:

```

Turbo Assembler   Version 2.0   03/31/96 23:08:59   Page 1
      A.ASM
1 0000                      .model small
2 0000                      .data
3 0000 FF 0A 17 FF          b1    db    -1, 10, 17H, 0FFH
4 0004 61 62                b3    db    'a', 'b'
5 0006 61 62 63 64 65 66 00 b2    db    "abcdef", 0
6
7 000D 1234 FFFF 4142        w1    dw    1234H, -1, 'AB'
8 0013 000Dr                w2    dw    w1
9
10 0015 12345678 FFFFFFFF    d1    dd    12345678H, -1
11 001D 3F800000 BF800000+   d2    dd    1.0, -1.0, 0.5
12 3F000000
13 0029 00000015sr          d3    dd    d1
14
15 002D 1000000000000002+   q1    dq    1000000000000002H, -1
16 FFFFFFFFFFFFFFFFFF
17 003D 3FF0000000000000+   q2    dq    1.0, -1.0
18 BFF0000000000000
19
20 004D 01234567890000012345 t1    dt    1234567890000012345
21 0057 81234567890000012345 t2    dt    -1234567890000012345
22 0061 099999999999999999 t3    dt    999999999999999999
23 006B 899999999999999999 t4    dt    -999999999999999999
24 0075 799999999999999999 t5    dt    799999999999999999
25 007F F99999999999999999 t6    dt    -799999999999999999
26 0089 3FFF8000000000000000 t7    dt    1.0
27                                end

```

Prima coloană listează numărul liniei din fișierul sursă. Apoi este listată adresa și conținutul câmpului de date corespunzător unei linii sursă. Dacă acest conținut este listat pe mai multe linii, se folosește semnul + pentru a indica aceasta. Se remarcă reprezentările numerelor reale și ale

întregilor negativi. Simbolurile r și s indică faptul că este vorba de o adresă relativă (deplasament) sau de segment. De exemplu, variabila d3 conține adresa de segment 0000 și deplasamentul 0015H, adică adresa relativă (pe 32 de biți) a variabilei d1. La întregii BCD pe 10 octeți, se observă memorarea explicită a bitului de semn: reprezentările pentru t1 și t2 diferă doar prin acest bit.

Listingul nu indică și poziția fizică a octeților în memorie. Următorul listing reprezintă zona de memorie corespunzătoare modulului de date de mai sus:

```
ds:0000 FF 0A 17 FF 61 62 61 62 63 64 65 66 00 34 12 FF
ds:0010 FF 42 41 0D 00 78 56 34 12 FF FF FF FF 00 00 80
ds:0020 3F 00 00 80 BF 00 00 00 3F 15 00 68 53 02 00 00
ds:0030 00 00 00 00 10 FF FF FF FF FF FF FF FF 00 00 00
ds:0040 00 00 00 F0 3F 00 00 00 00 00 00 F0 BF 45 23 01
ds:0050 00 00 89 67 45 23 01 45 23 01 00 00 89 67 45 23
ds:0060 81 99 99 99 99 99 99 99 99 99 09 99 99 99 99
ds:0070 99 99 99 99 89 99 99 99 99 99 99 99 99 79 99
ds:0080 99 99 99 99 99 99 99 99 F9
```

Din examinarea acestui listing, se observă, de exemplu, reprezentarea internă a variabilei double-word d1, aflată la adresa 0015H: secvența de octeți este 78 56 34 12, deci la adrese mici se găsește partea mai puțin semnificativă. Similar, variabila de tip word 'AB' (aflată la adresa 0FH), este memorată prin secvența de octeți 42H 41H. Acest listing este obținut la încărcarea modului de date în memorie, când adresele de segment sunt relocate, deci corespund unor adrese fizice concrete. Adresa variabilei d1 (conținută în d3) apare aici în forma (5368:0015). Pentru valorile reale, se regăsesc reprezentările descrise în 1.3.3.

Variabila t1 (aflată la adresa 004DH) este reprezentată prin secvența de octeți: 45 23 01 00 00 89 67 45 23 01, deci cifrele mai puțin semnificative sunt la adrese mici, iar în cadrul unui octet, cifra mai semnificativă este în partea high. Variabila t2 (aflată la adresa 0057H), prin secvența: 45 23 01 00 00 89 67 45 23 81, ceea ce arată că bitul de semn (reprezentat prin cifra 8) este în partea high a ultimului octet din reprezentare.

Valorile maxime și minime sunt reprezentate la adresele 0061H și 006BH, prin secvențele: 99 99 99 99 99 99 99 99 99 09, respectiv: 99 99 99 99 99 99 99 99 89.

Se observă și reprezentarea valorilor t5 și t6, în forma **99 99 99 99 99 99 99 99 79** și, respectiv, **99 99 99 99 99 99 99 99 F9**, în care apare cifra cea mai semnificativă 7, cu bitul de semn 0 sau 1.

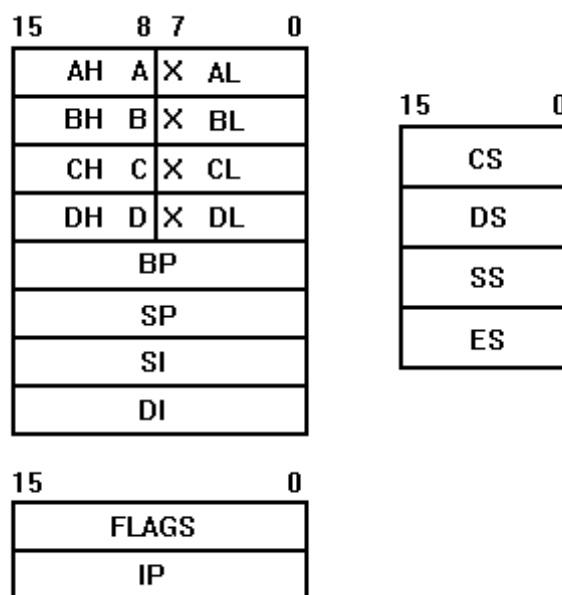
Trebuie precizat că formatele de reprezentare pentru date numerice (întregi sau reali) se regăsesc în aceeași formă și în limbajele de nivel înalt.

În Implementarea Borland PASCAL există caractere și întregi pe 2, 4 și 8 octeți, care corespund tipurilor de date byte, word, double word și quad-word. De asemenea există tipuri de date în virgulă mobilă pe 4, 8 și 10 octeți, în aceeași formă de reprezentare internă ca cea descrisă în 1.3.3.

În implementarea Borland C există tipul char (1 octet), tipuri întregi pe 2 și 4 octeți, cu sau fără semn (int, long, unsigned int, unsigned long) și tipuri în virgulă mobilă pe 4, 8 și 10 octeți (float, double, long double), cu aceleași reprezentări interne ca cele din 1.3.3

### **1.4 Arhitectura procesorului 8086. Formarea adresei fizice**

Arhitectura procesorului 8086, din punctul de vedere al programului utilizator, este ilustrată schematic în Figura 1.2. Sunt figurate registrele accesibile prin program.



**Figura 1.2 Registrele procesorului 8086**

Toate registrele sunt de 16 biți. O serie de registre (AX, BX, CX, DX) sunt disponibile și la nivel de octet, părțile mai semnificative fiind AH, BH, CH și DH, iar cele mai puțin semnificative, AL, BL, CL și DL. Denumirile registrelor sunt:

- AX - registru acumulator
- BX - registru de bază general
- CX - registru contor
- DX - registru de date
- BP - registru de bază pentru stivă (base pointer)
- SP - registru indicator de stivă (stack pointer)
- SI - registru index sursă
- DI - registru index destinație

Registrul notat FLAGS cuprinde flagurile procesorului, sau bistabililor de condiție, iar registrul IP (instruction pointer) este contorul program.

Denumirile registrelor de segment sunt:

- CS - registru de segment de cod (code segment)
- DS - registru de segment de date (data segment)
- SS - registru de segment de stivă (stack segment)
- ES - registru de segment de date suplimentar (extra segment)

Se observă că denumirile registrelor de segment corespund zonelor principale ale unui program executabil. Astfel, perechea de registre (CS:IP) va indica totdeauna adresa următoarei instrucțiuni care se va executa, iar perechea (SS:SP) indică totdeauna adresa vârfului stivei. Registrele DS și ES sunt folosite pentru a accesa date.

Procesorul 8086 dispune de adrese pe 20 de biți, fiind capabil să adreseze 1 megaoctet de memorie ( $2^{20}$ ). Se pune problema cum se formează adresa fizică pe 20 de biți (deci pe 5 cifre hexa), deoarece toate registrele procesorului sunt de 16 biți, putând codifica adrese în domeniul 0000...0FFFFH (pe 4 cifre hexa), deci într-un spațiu de maxim 64 KO.

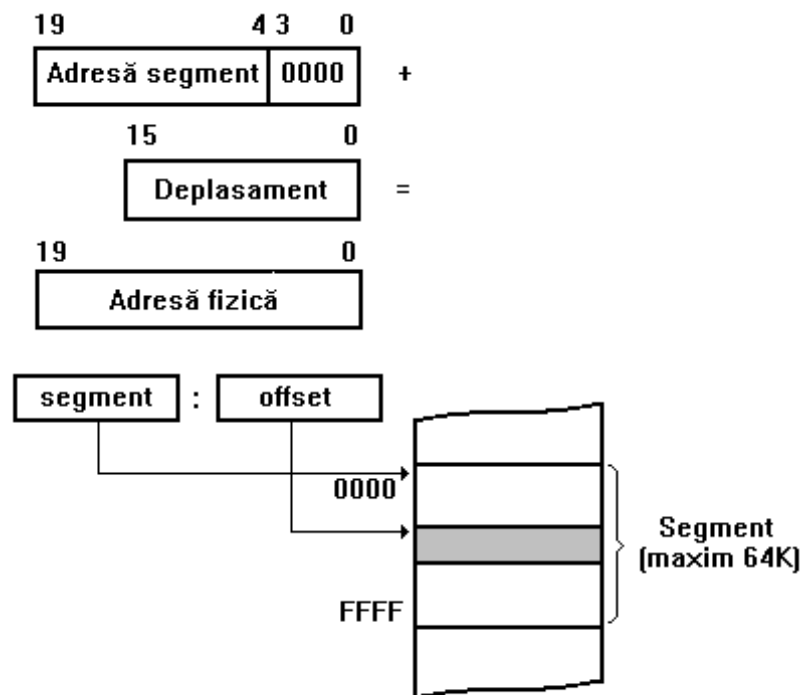
Memoria unui sistem cu procesor 8086 este divizată în segmente. Un segment este o zonă continuă de memorie, de lungime maximă de 64 KO, care începe la o adresă fizică multiplu de 4. Acest fapt înseamnă că ultima cifră hexa a adresei de început a unui segment este totdeauna 0. Ca atare, această cifră se poate omite și adresa de segment se poate reprezenta tot pe 16 biți. Adresele de început ale segmentelor se vor găsi întotdeauna într-unul din cele 4 registre de segment.

Adresarea în interiorul unui segment se realizează printr-un deplasament (offset) relativ la începutul segmentului. Deoarece un segment nu poate depăși 64 KO, deplasamentul se poate memora tot pe 16 biți. Deplasamentul poate fi o constantă sau conținutul unui registru care permite adresarea memoriei.

În concluzie, pentru adresarea unui octet de memorie, se folosesc două cantități pe 16 biți: o adresă de segment (conținută obligatoriu într-un registru de segment) și un deplasament. Deoarece ambele cantități sunt pe 16 biți, se vorbește de adrese (sau pointeri) de 32 de biți, deși adresa fizică este doar pe 20 de biți.

Formarea adresei fizice (pe 20 de biți) este realizată automat (prin hardware) de către o componentă a procesorului, conform Figurii 1.3.

Concret, adresa fizică se obține prin deplasarea adresei de segment cu 4 biți la stânga și prin adunarea deplasamentului. Pentru specificarea unei adrese complete (de 32 de biți), se folosește notația (segment:offset) sau (registru\_segment:offset). De exemplu, putem specifica o adresă prin (18A3:5B27) sau prin (DS:5B27).



**Figura 1.3 Formarea adresei fizice**

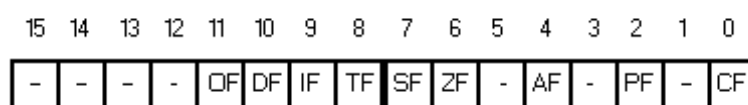
Trebuie remarcat faptul că asocierea (segment:offset) - adresă fizică nu este biunivocă, deoarece la o aceeași adresă fizică pot să corespundă mai multe perechi (segment:offset). De exemplu, perechile (18A3:5B27) și (18A2:5B37) reprezintă aceeași adresă fizică. În situația în care deplasamentul este redus la minim, adică în domeniul 0...F, corespondența devine biunivocă. Adresele de acest tip se numesc adrese normalizate sau pointeri normalizați.

O adresă completă de 32 de biți este memorată cu offsetul la adrese mici și cu adresa de segment la adrese mari. Adresele complete se pot obține cu directiva DD (Define Double-Word).

**Registrul de flaguri (bistabili de condiție)** al procesorului 8086 are configurația din Figura 1.4. O serie de flaguri sunt flaguri aritmetice: acestea sunt poziționate la 0 sau la 1 ca urmare a unor operații aritmetice sau logice. Celelalte flaguri controlează anumite operații ale procesorului.

Semnificația flagurilor este următoarea:

- **CF (Carry Flag, bistabil de transport)** - semnifică un transport sau un împrumut din/în bitul cel mai semnificativ al rezultatului, de exemplu la operații de adunare sau de scădere.



**Figura 1.4 Registrul de flaguri al procesorului 8086**

- **PF (Parity Flag, flag de paritate)** - este poziționat în așa fel încât numărul de biți egali cu 1 din octetul cel mai puțin semnificativ al rezultatului, împreună cu flagul PF, să fie impar; altfel formulat, suma modulo 2 a tuturor biților din octetul c.m.p.s. și a lui PF să fie 1.
- **AF (Auxiliary Carry Flag, bistabil de transport auxiliar)** - indică un transport sau un împrumut din/în bitul 4 al rezultatului.
- **ZF (Zero Flag, bistabil de zero)** - este poziționat la 1 dacă rezultatul operației este 0.
- **SF (Sign Flag, bistabil de semn)** - este poziționat la 1 dacă b.c.m.s. al rezultatului (bitul de semn) este 1.
- **OF (Overflow Flag, bistabil de depășire)** - este poziționat la 1 dacă operația a condus la o depășire de domeniu a rezultatului (la operații cu sau fără semn).
- **TF (Trap Flag, bistabil de urmărire)** - dacă este poziționat la 1, se forțează o întrerupere, pe un nivel predefinit, la execuția fiecărei instrucțiuni; acest fapt este util în programele de depanare, în care este posibilă rularea pas cu pas a unui program.
- **IF (Interrupt Flag, bistabil de întreruperi)** - dacă este poziționat la 1, procesorul ia în considerație întreruperile hardware externe; altfel, acestea sunt ignorate.
- **DF (Direction Flag, bistabil de direcție)** - precizează sensul (crescător sau descrescător) de variație a adreselor la operațiile cu șiruri de octeți sau de cuvinte.

Flagurile CF, PF, AF, ZF, SF și OF sunt numite flaguri aritmetice. Flagurile TF, IF și DF sunt numite flaguri de control.

### 1.5 Moduri de adresare

Pentru a scrie un program ASM într-o formă cât mai simplă, se pot folosi directivele simplificate de definire a segmentelor. Acestea sunt:

- `.model small` (precizează un model de memorie)
- `.code` (definire de segment de cod)
- `.stack n` (definire de segment de stivă)
- `.data` (definire de segment de date)
- `end` etichetă (sfârșit logic al programului)

În principiu, segmentul de cod va cuprinde programul executabil (instrucțiuni), iar segmentul de date va cuprinde date definite de utilizator. Segmentul de stivă nu este folosit explicit. Comentariile se scriu folosind simbolul `;`. Ceea ce urmează după `;` până la sfârșitul liniei curente, este considerat comentariu.

Instrucțiunile procesorului 8086 pot avea unul sau doi operanzi, care pot fi registre sau operanzi aflați în memorie. Când există doi operanzi, unul este obligatoriu un registru (deci nu pot fi doi operanzi în memorie).

Modurile de adresare specifică modul în care se calculează adresa operandului aflat în memorie. Se folosesc denumirile de **adresă de segment** (AS) pentru adresa de început a segmentului în care se găsește operandul și **adresă efectivă** (AE), pentru deplasamentul operandului în cadrul segmentului respectiv. Adresa de segment și adresa efectivă formează **adresa fizică** (AF), conform mecanismului descris în 1.4. Adresa de segment este furnizată de unul din cele 4 registre de segment.

Operanzii pot fi de tip octet sau de tip word, iar operațiile se pot face numai între operanzi de același tip. Tipul operandului aflat în memorie rezultă de obicei din context (de obicei din registrele folosite). În situațiile

În care acest lucru nu este posibil, se folosește operatorul ptr al limbajului de asamblare, în forma byte ptr sau word ptr, pentru a explicita tipul operandului.

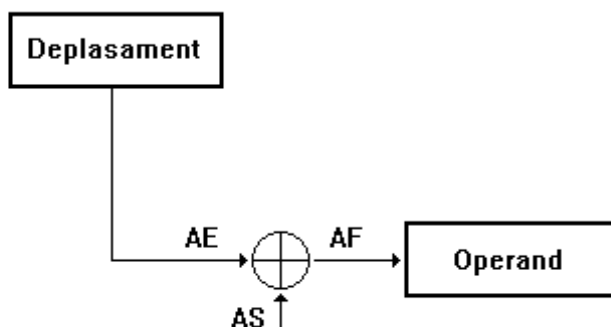
- **Adresare imediată**

În acest mod de adresare operandul apare chiar în instrucțiune. De exemplu:

```
mov ax, 1      ; Pune în AX valoarea 1
add bx, 2      ; Adună 2 la BX
```

- **Adresare directă**

Adresa efectivă a operandului este furnizată printr-un deplasament în interiorul segmentului curent (vezi Figura 1.5).



**Figura 1.5 Adresare directă**

```
.data
    VAL    dw 1
.code
    mov    bx, VAL      ; Pune în BX conținutul locației VAL
                        ; În formatul instrucțiunii, expresia VAL se va
                        ; înlocui cu deplasamentul în cadrul segmentu-
                        ; lui de date
    add    cx, [100]    ; Deplasament explicit
```

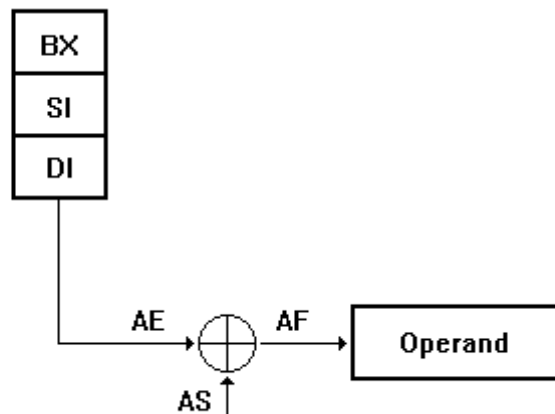
Prima instrucțiune înseamnă "pune conținutul locației VAL în BX", ceea ce va conduce la o valoare a registrului BX egală cu 1. A doua instrucțiune înseamnă "adună la CX conținutul locației de memorie de la adresa efectivă 100". De observat prezența parantezelor drepte: fără ele, instrucțiunea ar fi însemnat "adună valoarea 100 la registrul CX". În ambele instrucțiuni, operandul aflat în memorie este considerat de tip word.

Acest mod de adresare folosește registrul DS ca registru implicit de segment.

- **Adresare indirectă (prin registre)**

Adresa efectivă a operandului este dată de conținutul registrelor BX, SI sau DI (vezi Figura 1.6). Registrul implicit de segment este DS.



**Figura 1.6 Adresare indirectă**

În textul sursă se scrie, de exemplu:

<code>mov ax, [bx]</code>	; Pune în AX conținutul locației de
	; memorie de la adresa dată de BX
<code>mov [di], cx</code>	; Pune conținutul lui CX în locația
	; de la adresa dată de DI.
<code>add byte ptr [si], 2</code>	; Adună 2 la octetul aflat la adresa dată de SI

În primele două instrucțiuni, operandul aflat în memorie este considerat de 16 biți (datorită celui alt operand, care este un registru de 16 biți). Dacă a trei instrucțiune ar fi scrisă:

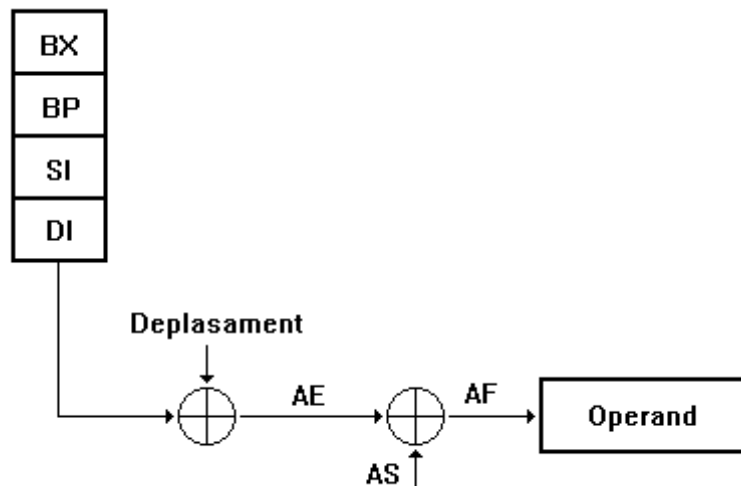
```
add [si], 2
```

asamblorul nu ar ști tipul operandului din memorie. Instrucțiune poate fi interpretată ca "adună 2 la octetul de la adresa dată de SI" sau "adună 2 la cuvântul de la adresa dată de SI" și s-ar obține un mesaj de eroare la asamblare. Folosirea operatorului `ptr` elimină această ambiguitate.

- **Adresare bazată sau indexată**

Adresa efectivă se obține adunând la unul din registrele de bază (BX sau BP) sau la unul din registrele index (SI sau DI), un deplasament constant pe 8 sau 16 biți (vezi Figura 1.7). Registrul de segment implicit este DS (dacă se folosesc BX, SI sau DI) și, respectiv, SS (dacă se folosește BP).

Forma de memorare a deplasamentului (pe 8 sau 16 biți) este determinată de asamblor. În textul sursă se scrie pur și simplu un deplasament constant.



**Figura 1.7 Adresare bazată sau indexată**

Se pot folosi diverse forme de scriere, ca în exemplul de mai jos, în care bx este (din punct de vedere al mecanismului de adresare) adresă de bază, iar VAL este un deplasament constant. Din punct de vedere al programului, care își propune să acceseze elementele tabloului de octeți VAL, semnificația este exact inversă: VAL este adresa de început a tabloului (deci o adresă de bază), iar bx este un indice (un deplasament).

```
.data
    VAL    dw    10 dup (0)    ; Definire de tablou de 10 octeți
.code
    mov    bx, 5
    mov    ax, VAL[bx]
    mov    ax, bx[VAL]
    mov    ax, [bx+VAL]
    mov    ax, [bx].VAL
```

Este posibil și alt mod de acces, în care se încarcă registrul BX cu deplasamentul tabloului VAL (folosind operatorul offset) și se folosește un indice explicit.

```
    mov    bx, offset VAL
    mov    ax, 5[bx]
    mov    ax, bx[5]
    mov    ax, [bx+5]
    mov    ax, [bx].5
```

În această situație, adresa de bază (conținută în BX) este adresa de început a tabloului VAL.

Trebuie remarcat că, în textul sursă, toate formele de adresare se scriu cu operatorul de indexare (paranteze drepte). De exemplu, se poate scrie:

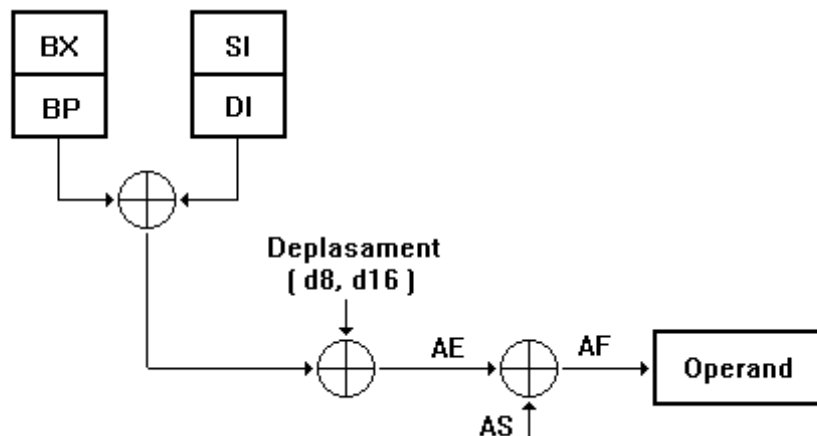
```
    mov    ax, [bx]           ; Adresare indirectă
    mov    ax, [100]         ; Adresare directă
    mov    ax, [bx+100]      ; Adresare bazată
```

De remarcat că folosirea registrului BP cu un deplasament nul permite o formă de scriere asemănătoare cu adresarea indirectă, dar codificată intern ca adresare bazată:

```
    mov    ax, [bx]           ; Adresare indirectă
    mov    ax, [bp]          ; Adresare bazată cu deplasament nul
```

### • Adresare bazată și indexată

Adresa efectivă este formată prin adunarea unuia din registrele de bază (BX sau BP) cu unul din registrele index (SI sau DI) și cu un deplasament de 8 sau 16 biți (vezi Figura 1.8).



**Figura 1.8 Adresare bazată și indexată**

Registrele de segment implicite sunt DS (dacă se folosește bX cu Si sau cu DI) și, respectiv, SS (dacă se folosește BP cu SI sau cu DI). Deplasamentul poate fi și nul. În textul sursă se folosește operatorul de indexare []:

```
mov ax, [bx][si]
mov ax, [bx+si+7]
mov ax, [bx+si].7
mov ax, [bp][di][7]
```

În descrierea modurilor de adresare, s-a precizat de fiecare dată registrul implicit de segment care participă la formarea adresei fizice a operandului aflat în memorie. Acest registru implicit este DS în toate mmodurile de adresare care nu implică registrul BP și, respectiv, SS, în modurile de adresare în care participă registrul BP.

Aceste reguli implicite pot fi modificate prin folosirea prefixelor de segment. Un prefix de segment este un octet care apare înaintea codului instrucțiunii mașină și care identifică

explicit registrul de segment folosit. În textul sursă se scrie un registru de segment, urmat de :, înaintea operandului aflat în memorie:

```
mov bx, ds:[bp+7]
mov ax, cs:[si][bx+3]
mov ax, ss:[bx]
```

Evident, gestiunea a ce se găsește în segmentele adresate în acest fel cade în sarcina programatorului.

## 1.6 Codificarea internă a instrucțiunilor

Instrucțiunile procesorului 8086 pot fi codificate pe un număr de octeți cuprins între 1 și 6. Codificarea cuprinde codul instrucțiunii (tipul operației), modul de adresare, deplasamentul operandului aflat în memorie (pe 8 sau 16 biți) sau adresa efectivă (în cazul adresării directe),

date (operanzi) imediate, pe 8 sau 16 biți. Pe lângă formatul propriu-zis, mai poate exista un prefix de segment sau un prefix de repetare.

Formatul general al instrucțiunilor 8086 este ilustrat în Figura 1.9.

La instrucțiunile cu doi operanzi, un operand este obligatoriu de tip registru și este codificat de câmpul REG, iar celălalt este registru sau operand în memorie și este codificat de câmpul R/M sau de câmpurile R/M și MOD. Semnificația câmpurilor din Figura 1.9 este următoarea:

- **Câmpul D (destinație)** - codifică sensul operației:

D = 0      R/M <--- R/M operație REG

D = 1      REG <--- REG operație R/M

De exemplu, codificările instrucțiunilor:

```
add    [bx], si
add    si, [bx]
```

vor diferi numai prin câmpul D (aceeași operație și aceiași operanzi, dar diferă sensul).

- **Câmpul W (word)** - codifică lungimea operanzilor (octet sau cuvânt):

W = 0      operație la nivel de octet

W = 1      operație la nivel de cuvânt

De exemplu, instrucțiunile:

```
add    ax, bx
add    al, bl
```

vor diferi doar prin câmpul W.

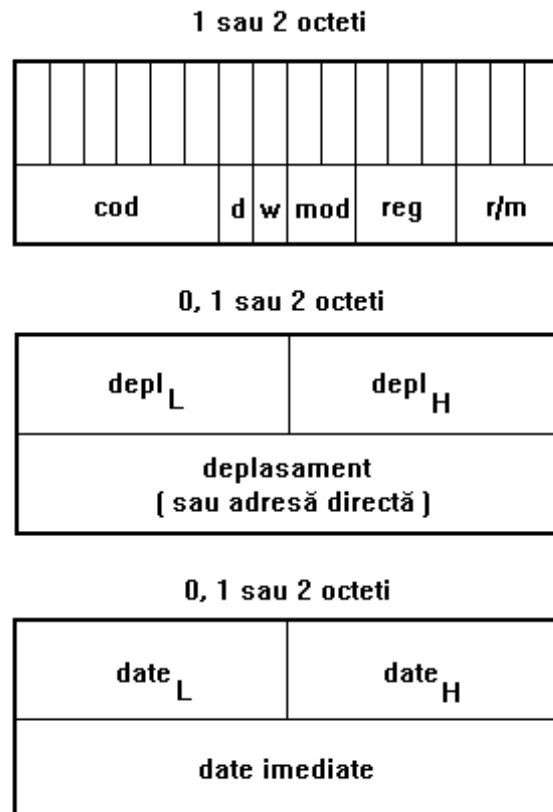


Figura 1.9 Formatul instrucțiunilor 8086

**Câmpul REG** identifică operandul de tip registru.

**Câmpul R/M** identifică al doilea operand de tip registru sau un registru care participă la formarea adresei efective a operandului aflat în memorie.

**Câmpul MOD** codifică modul de adresare:

- **MOD = 11** înseamnă că al doilea operand este tot de tip registru (codificat de câmpul R/M).
- **MOD = 00** înseamnă adresare directă (R/M = 6), indirectă (R/M = 4, 5, 6) sau adresare bazată și indexată cu deplasament nul (R/M = 0, 1, 2, 3). În aceste moduri de adresare, octeții 3 și 4 din Figura 1.9 lipsesc din formatul instrucțiunii (cu excepția cazului R/M = 6, adică al adresării directe).
- **MOD = 01** înseamnă adresare bazată (R/M = 6, 7), indexată (R/M = 4, 5) sau bazată și indexată (R/M = 0, 1, 2, 3), toate cu un deplasament de 8 biți. În aceste moduri de adresare, octetul 4 din Figura 1.9 lipsește din formatul instrucțiunii.
- **MOD = 10** este asemănător cu MOD = 01, dar deplasamentele sunt pe 16 biți. În aceste moduri de adresare, octeții 3 și 4 din Figura 1.9 sunt prezenți în formatul instrucțiunii.

Situațiile de mai sus sunt descrise sintetic în Tabelul 1.10.

MOD → R/M ↓	11		00	01	11
	W=0	W=1			
<b>0</b>	AL	AX	(BX)+(SI)	(BX)+(SI)+d8	(BX)+(SI)+d16
<b>1</b>	CL	CX	(BX)+(DI)	(BX)+(DI)+d8	(BX)+(DI) +d16
<b>2</b>	DL	DX	(BP)+(SI)	(BP)+(SI)+d8	(BP)+(SI) +d16
<b>3</b>	BL	BX	(BP)+(DI)	(BP)+(DI)+d8	(BP)+(DI) +d16
<b>4</b>	AH	SP	(SI)	(SI) +d8	(SI) +d16
<b>5</b>	CH	BP	(DI)	(DI) +d8	(DI) +d16
<b>6</b>	DH	SI	directă	(BP) +d8	(BP) +d16
<b>7</b>	BH	DI	(BX)	(BX) +d8	(BX) +d16

**Tabelul 1.10 Codificarea modurilor de adresare**