1. REPREZENTAREA INFORMAȚIEI LA PROCESOARELE X86

1.1. ASPECTE GENERALE

Cum este reprezentată informația în interiorul unui sistem PC?

Memoria PC-ului conţine informaţie sub formă de numere binare, 0 şi 1 şi din acest motiv, spunem că PC-urile stochează informaţia în format ***binar***, nu zecimal. Cele două valori 0 și 1 constituie de fapt celula informațională de bază, numită „***bit***”. Practic, toată informația din PC este reprezentată sub formă de biți. Mai mult, acești biți sunt organizați sub formă de ***octeţi*** sau grupuri de octeţi în memoria sistemului sau în regiștrii procesorului; astfel, un multiplu de puteri ale lui 2 octeţi (deci 2^x octeţi) vor forma diferite structuri specifice tipului de procesor și vor purta numele de ***date fundamentale***, așa cum vom vedea ulterior.

Ce sistem de numerație vom folosi?

Oamenii folosesc în mod uzual ***sistemul de numeraţie zecimal*** pentru a se referi la numere în general (sistem adoptat în principal datorită structurii corpului omenesc: cele 10 degete de la mâini). Similar, la folosirea datelor în PC, s-a căutat un sistem de reprezentare al informaţiei mai apropiat de situaţiile sau stările care apar în cadrul lui și anume: ***trece*** sau ***nu trece*** curent printr-o porţiune de circuit. Acest sistem este cel binar, în care orice număr se va reprezenta ca șiruri de 0 și 1.

Raţionamentul e simplu, așa cum arată Figura 2.1: noi oamenii gândim în ***zecimal*** datorită structurii corpului nostru, în timp ce SC „gândesc” adaptat sistemului lor, adică în ***binar***. Pentru a putea realiza comunicarea cât mai ușoară între cele 2 entităţi însă, uzual se folosește sistemul ***hexazecimal***.

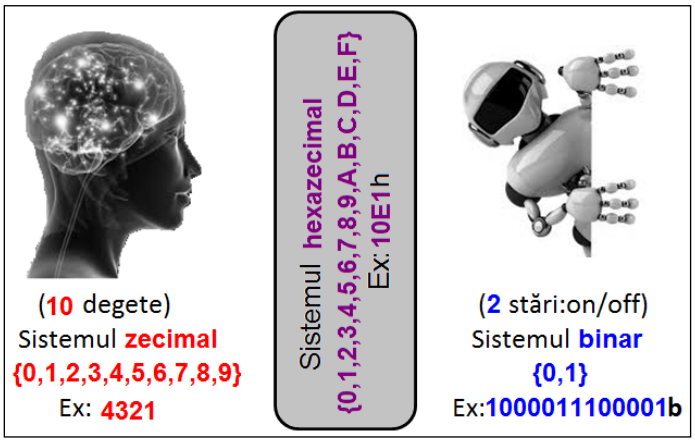

Figura 1.1 Legătura dintre sistemul zecimal, binar şi hexazecimal

Sistemul de numeraţie hexazecimal facilitează trecerea de la sistemul zecimal la cel binar sau invers, acesta fiind oarecum apropiat de ambele sisteme: permite exprimarea unor valori cu un număr rezonabil de cifre (asemănător sistemului zecimal) şi permite conversia relativ simplă în sistemul binar.

De exemplu, numărul din zecimal 4321 prin conversie în binar devine 1000011100001b, mult mai simplu de urmărit în hexazecimal: 10E1h, fiind *aproape* la fel de uşor de manevrat (de către noi, oamenii) ca şi cel scris în zecimal.

Cum s-a ajuns de la 1000011100001b la 10E1h ?

În capitolul următor sunt date regulile de transformare între diferite baze de numeraţie; deocamdată, reţinem faptul că pentru a transforma un număr din binar în hexazecimal e nevoie de *un număr multiplu de 4* de biţi care vor forma tetrade şi fiecare tetradă va fi apoi înlocuită cu cifra hexazecimală corespunzătoare. Astfel, numărul din binar 1000011100001b s-a scris **000**1.0000.1110.0001b, pentru a putea forma grupuri de câte 4 biți. Pornind dinspre dreapta spre stânga s-au format cifrele hexazecimale 1, apoi E, apoi 0 și în final încă o dată 1.

Într-o anumită situaţie, care sistem de numeraţie este mai potrivit ?

Nu se poate spune că există situaţii în care doar un anumit sistem de numeraţie e bun, iar altul nu. Este vorba doar de obișnuinţă: atunci când ne raportăm la conţinutul unor structuri din calculator, de exemplu regiștrii procesorului sau zone din memorie, în practică, informaţia este prezentată în format hexazecimal, chiar dacă în realitate, în computer informaţia respectivă este în binar. Aceasta, deoarece este mai uşor pentru noi, oamenii, să lucrăm cu valori în hexazecimal decât cu valori binare. Programatorii în limbaj de asamblare preferă forma hexazecimală în locul celei zecimale (fiind mai apropiată de forma de reprezentare din PC). Dacă dorim să implementăm o anumită operaţie (de ex. calculul 2149+586) este mai uşor pentru noi, oamenii, să gândim valorile în ***zecimal*** decât în hexazecimal sau binar. Deși intern, în regiștrii sau în memoria sistemului, informația este în binar, de cele mai multe ori, când ne vom referi la aceste operații în contextul reprezentării interne (la operații cu octeți), vom folosi sistemul ***hexazecimal***. Uneori, va fi însă esenţial să putem privi acele valori ale octeților în ***binar***; de exemplu, atunci când analizăm mai detaliat anumite situaţii precum setări/ resetări de biţi, complementări, etc.

Cum se citește sau interpretează informaţia stocată în interiorul PC-ului ?

În general, atunci când se face referire la un computer, Sistem de Calcul (SC) sau PC (Personal Computer) înţelegem un sistem capabil să prelucreze informaţii în mod inteligent. Aceste „informaţii” trebuie reprezentate (sau codificate) în interiorul sistemului într-un anumit format, folosind anumite reguli şi convenţii bine stabilite. Cu toate acestea, ***datele din PC pot avea semnificaţie diferită***: de exemplu, 67 poate reprezenta o valoare de temperatură, vârsta, o dimensiune sau un cod (codul ASCİİ al literei “C”). Practic, atât reprezentarea, cât şi interpretarea sunt esențiale.

2.1.1. TIPURI DE DATE: NUMERICE ŞI DE TIP CARACTER

Informaţia (sub forma datelor întâlnite în PC) spunem că poate fi de formă: ***numerică*** sau ***nenumerică***.
La rândul lor, valorile ***numerice*** (sau numerele mai simplu spus) pot fi împărţite în:

- ***numere întregi (fără virgulă)*** , de exemplu -5, 0, 3 și
- ***numere flotante (cu virgulă)*** de exemplu -0.333, 0.5, etc.

Numerelor care se pot reprezenta ca întregi (în general, în urma unei operaţii matematice gen împărţire) li se spune simplu, „numere întregi", însă celor care sunt de tip real cu virgulă (deşi acestea le includ pe cele întregi) li se spune „numere reprezentate în virgulă flotantă", termenul prescurtat fiind FP (floating point).

Preponderent, vom folosi numere întregi şi numere naturale, cărora le vom spune simplu: ***numere cu semn***, respectiv ***numere fără semn***.

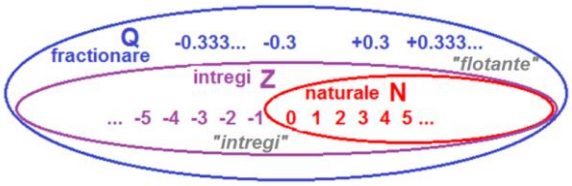

Figura 2.2. Ilustrarea domeniului numerelor naturale şi întregi („***întregi***") şi a celor fracţionare sau reale („***flotante***")

Pe de altă parte, informaţia ***nenumerică*** (alfanumerică sau de tip caracter cum se mai numeşte) este cea care apare sub formă de text. Termenul "***caracter***" (deşi ca reprezentare internă e tot o valoare numerică) se referă la orice simbol pe care sistemele de calcul ştiu să-l interpreteze: orice e ***tastabil*** (se poate prelua de la tastatură, chiar şi folosind mai multe combinaţii de taste) sau ***tipăribil*** (pe un sistem de afişare, de exemplu pe monitor) reprezintă un ***caracter***.

Pentru afişarea caracterelor pe ecran vom folosi așa numitele coduri ASCII: de exemplu, dacă vom folosi numărul 1000 0001b=129 (în zecimal), interpretat ca un ***cod Ascii***, atunci la afişarea pe ecran va apărea caracterul ***ü***.

Din punct de vedere al interpretării unei valori numerice, o valoare scrisă în binar de forma **10000001b** poate desemna numărul **129** sau numărul **-127**, în funcţie de convenţia[1] de reprezentare a numerelor: ***fără semn*** (corespunzător celor naturale, notate cu N) sau ***cu semn*** (corespunzător celor întregi, notate cu Z).

10000001b = *numărul 129* / *numărul -127* 10000001b = ***ü***

Figura 2.3 a) Interpretarea numerică a unei valori binare: *fără semn* (**uns**) şi *cu semn* (**s**); b) Interpretarea nenumerică a unei valori binare: cod (Ascii) al unui simbol/caracter (**car**)

La utilizarea unui SC este esenţială cunoaşterea tipurilor de date pe care le poate folosi acesta, dar şi înţelegerea modului de reprezentare şi de interpretare a acestor date. Astfel, pentru început, în PC vom avea 3 tipuri de informaţie, notate:

- ***uns*** (de la *unsigned*) pentru valorile numerice **fără semn**,
- ***s*** (de la *signed*) pentru valorile numerice **cu semn** şi respectiv
- ***car*** pentru valorile de tip **caracter** sau alfanumerice[2].

Oricare din aceste 3 tipuri de informaţie se reprezintă intern, în memoria SC sau în regiştrii CPU într-un mod pe care procesorul îl înţelege: în ***binar***, sub formă de ***biţi***. Pentru simplitate, la început, vom folosi preponderent ***interpretarea valorilor binare ca valori numerice fără semn***. Le vom considera astfel în cadrul Capitolului 3 pentru a ne fi mai ușor să deprindem atât regulile de conversie dintr-o bază în alta cât și regulile de realizare a operațiilor aritmetice în diferite baze. Ulterior, în Capitolul 4 le reluăm d.p.d.v. al numerelor **cu semn**.

2.1.2. BITUL

Datorită apariţiei circuitelor cu 2 stări stabile, s-a decis că cel mai potrivit sistem de reprezentare intern într-un sistem de calcul este ***sistemul binar***, cu cele 2 simboluri sau cifre binare: 0 şi 1.

Cea mai mică unitate de informaţie o constituie ***bitul*** (în engleză se scrie şi se pronunţă "bit", la plural "bits" de la **b**inary dig**its**). În timp, s-au cunoscut mai multe forme de implementare practică a biţilor: plecând de la comutatoare, relee şi diode, tranzistoare, iar în prezent circuite integrate, toate s-au bazat pe acelaşi principiu:

- să permită trecerea curentului electric, analogie cu **bit = 1**,
- să blocheze trecerea curentului electric, analogie cu **bit = 0**.

Bitul poate lua doar una din cele 2 valori: **0** sau **1** (*nu permite* sau *permite* trecerea curentului pe o porţiune de circuit).

Cu toate acestea, o singură cifră binară (0 sau 1) s-a constatat că deţine prea puţină informaţie pentru a fi utilă sub această formă singulară. Plecând tot de la sistemul zecimal, a apărut ideea de a reprezenta numerele binare prin *şiruri de biţi*, în analogie cu scrierea poziţională din sistemul zecimal, atât de uzuală, în care numerele sunt şiruri de cifre scrise secvenţial (pentru noi, oamenii, este foarte cunoscută această scriere a numerelor folosind sistemul zecimal: am fost "antrenaţi" încă de timpuriu să-l folosim).

De exemplu, 4321 îl citim ***patru mii trei sute douăzeci şi unu***, care scris folosind valori de biţi este 1000011100001b (să observăm că sunt 13 biți care compun acest număr). Modul cum a fost obținut acest număr în binar va fi descris în Capitolul 3.

---

[1] Vom vedea în capitolele următoare că în PC se foloseşte regula complement faţă de 2 pentru a reprezenta numerele negative.

[2] Modul de lucru cu aceste 3 tipuri de informație va fi abordat mai pe larg în următoarele capitole.

2.1.3. OCTET ŞI NIBBLE

Memorarea şi manevrarea unei valori scrise în binar (de exemplu 1000011100001b pt numărul 4321) e dificilă (dacă nu *imposibilă* pentru majoritatea) în limbaj uzual.

Datorită numărului mare de cifre necesare reprezentării unui număr în sistemul binar, s-au căutat soluţii care să-i ajute pe oameni să interacţioneze cu sistemul binar și deci indirect cu SC. Una dintre aceste soluţii a vizat *organizarea biţilor în grupuri de biţi.*

Cum se pot grupa biţii pentru a fi mai uşor de manevrat?

Organismele autorizate au realizat standardizarea dimensiunii acestor grupuri de biţi, apărând astfel noţiunea de ***octet*** (în engleză "byte"), care desemnează un ***grup de 8 biţi***.

Folosind octeţi, numărul scris în binar pe 13 biți sub forma 1000011100001b va trebui suplimentat cu încă 3 biţi (pentru a se scrie pe 16 biţi[3]) în extremitatea stângă; considerându-l ca ***număr fără semn***, cei 3 biți vor fi de valoare zero şi deci numărul 4321 se va scrie: 000 10000 11100001b.

Astfel, pentru ***numerele fără semn*** putem formula următoarea **regulă**:

> Biţii se grupează în octeţi întotdeauna pornind dinspre dreapta înspre stânga şi se completează până la umplerea structurii dorite, cu biţi având valoarea **0**.

Uneori se foloseşte noţiunea "***nibble***" (sau tetradă în română) pentru a desemna un ***grup de doar 4 biţi***.

1000011100001b = | 00010000 | 11100001 | (scris in binar ca octeti)
| 0001 | 0000 | 1110 | 0001 | (scris in binar ca tetrade)

Figura 2.4 Posibile grupări de biţi în *octeţi* sau *tetrade*

Cum sunt organizaţi/ numerotaţi biţii în cadrul unui octet ?

Numerotarea biţilor în cadrul unui octet începe de la bitul 0, desemnând bitul de pe poziţia cea mai din dreapta, numit și *cel mai puţin semnificativ bit,* scris ***c.m.p.s.*** *bit* (prescurtat şi **LSb** de la Least Significant bit).

În cealaltă extremă, pe poziţia cea mai din stânga, se află bitul 7, numit şi *cel mai semnificativ bit,* scris ***c.m.s.*** bit (prescurtat şi **MSb** de la Most Significant bit). Această numerotare este fictivă, doar că trebuie ţinut cont de ea (Figura 2.5).

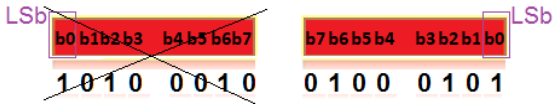

Figura 2.5. Numerotarea şi scrierea corectă a biţilor în cadrul unui octet

Care sunt submultiplii octetului ?

Nibble-ul (Figura 2.6) se folosește în special la exprimarea valorilor hexazecimale, aşa cum se va vedea în continuare. Trebuie menţionat faptul că nu se folosește nibble-ul la accesarea informaţiei din memorie sau din regiștrii procesorului. Pentru accesare, cea mai mică entitate de informație cu care va putea lucra procesorul este octetul.

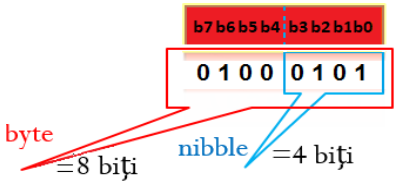

Figura 2.6. Reprezentarea unui nibble - submultiplu uzual al octetului

Ce valori reprezintă un octet ?

Octetul (byte) este un şir de 8 biţi şi reprezintă un standard unanim respectat. Orice combinaţie de 8 biţi poate reprezenta un octet; de exemplu: 01010101b, 11110000b, 00011101b, etc; totuşi, să nu uităm că valorile din binar se pot scrie mult mai uşor folosind cifre hexazecimale: 55h, F0h, 1Dh, etc.

2.1.4. REPREZENTAREA NUMERELOR PE N BIŢI

Câte combinaţii de biţi (sau numere binare) putem scrie folosind un octet ?

Se observă că folosind ***un singur bit*** se pot reprezenta ***2 valori binare (0 şi 1)***,

pe 2 biţi se pot reprezenta ***4 valori binare (00, 01, 10, 11)***,

pe 3 biţi se pot reprezenta ***8 valori (000, 001, 010, 011, 100, 101, 110, 111),*** ș.a.m.d.

Astfel, se deduce simplu că ***pe un octet*** se pot reprezenta ***2^8 numere*** sau ***256 valori binare diferite***, de exemplu desemnând mulţimea numerelor întregi pozitive (naturale) de la 0 la 255 (valori numerice ***fără semn***) sau la fel de bine, ar putea desemna mulțimea numerelor întregi ***cu semn***, cuprinse între -127 și +128 (s-a ales ca jumătate să fie numere negative și jumătate să fie numere pozitive).

[3] Folosim 16 biţi, 32 biţi sau 64 biţi; în general: dimensiunea standardizată de reprezentare

Regulă:	**Folosind n biţi, putem scrie 2^n valori binare diferite.**

2.1.5. REPREZENTAREA PE AXA NUMERELOR

Reprezentarea numerelor zecimale pe axa orizontală este cunoscută din clasele primare, folosind segmente. Întrebarea care se pune aici este: cum se reprezintă *valorile binare* pe această axă? Similar cu reprezentarea valorilor zecimale, adică echidistant unele faţă de altele?

De exemplu, dacă vom considera valorile binare corespunzând numerelor fără semn 0,1,2,3, acestea se vor reprezenta cu câte un segment în plus faţă de valoarea anterioară, aşa cum arată Figura 2.7 din stânga? Cum se stabileşte atunci lungimea segmentului folosit la distanţiere? Sau, pentru a evidenţia împărţirea în cele 2^n părţi a unităţii, ar fi mai indicată reprezentarea asemănător fracţiilor pe axă, aşa cum arată Figura 2.7 din dreapta?

Figura 2.7 Reprezentarea valorilor binare pe axa numerelor zecimale pt n=1 bit, n=2 biţi şi n=3 biţi în 2 moduri: cu segmente identice (stânga), resp. având unitatea identică (dreapta)

2.1.6. REPREZENTAREA PE ROATA NUMERELOR

Reprezentarea valorilor ce se pot scrie pe un anumit nr de biţi se realizează deseori folosind ***roata numerelor***: în loc de a plasa valorile pe o singură axă orizontală, acestea se pot specifica pe un cerc cu atâtea secţiuni diametrale câte numere sunt de reprezentat pe acel nr de biţi. Avantajul acestei reprezentări este faptul că se sugerează existenţa unui *număr finit şi repetabil de valori pe un anumit număr de biţi* (de exemplu, adunând pe 4 biţi la 1110b încă 3, se obţine 0001b). Acest aspect va fi deosebit de important la lucrul cu regiștrii procesorului deoarece aceștia au dimensiunea fixă (8, 16, 32, 64, ... biți) și ne vom întâlni cu astfel de situații.

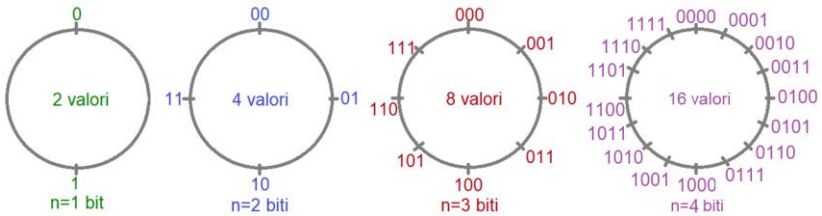

Figura 2.8 Reprezentarea valorilor binare pe roata nr pt n=1 bit, n=2 biţi, n=3 biţi şi n=4 biţi

Ca o regulă generală, se poate menţiona faptul că ***folosind un număr de n biţi, se pot reprezenta 2^n valori diferite*** (combinaţii unice de biţi), de exemplu, acestea reprezentând numerele naturale cuprinse între 0 şi 2^n-1. La fel de bine însă, acestea ar putea reprezenta numere întregi cuprinse între -2^{n-1} și $+2^{n-1}$-1.

Cu cât numărul de biţi este mai mare, cu atât ***sistemul este mai fin*** în reprezentarea valorilor şi cu atât roata din Figura 2.8 se împarte în mai multe secţiuni, rotindu-se cu o cursă mai mică pentru a trece la următoarea valoare.

Roata numerelor este utilă şi atunci când dorim să numărăm (din 1 în 1, din 2 în 2, din 4 în 4, etc) crescător sau descrescător în binar.

Ce semnificație pot avea cele 2^n valori binare ?

Semnificaţia dorită pentru cele 2^n valori binare, aşa cum vom vedea ulterior, depinde în special de problema de rezolvat: acestea pot fi interpretate ca valori numerice fără semn (***uns***), ca valori numerice cu semn (***s***) sau ca valori alfanumerice de tip caracter (***car***).

În Figura 2.8, dacă vom considera valorile în **convenția numerelor *fără semn***, pentru reprezentarea folosind 2 biţi, avem 4 valori posibile (00, 01, 10, 11) ce corespund numerelor din zecimal 0,1,2,3.

La reprezentarea folosind 3 biţi, cele 8 valori scrise în binar 000b, 001b, 010b, 011b, 100b, 101b, 110b, 111b corespund numerelor 0,1,2,3,4,5,6,7.

La reprezentarea folosind 4 biţi, cele 16 valori scrise în binar 0000b, 0001b, 0010b, 0011b, 0100b, 0101b, 0110b, 0111b, 1000b, 1001b, 1010b, 1011b, 1100b, 1101b, 1110b, 1111b corespund numerelor 0,1,2,...,14,15.

În Figura 2.8, dacă vom considera valorile în **convenția numerelor *cu semn***, pt reprezentarea folosind 2 biţi, cele 4 valori posibile (00, 01, 10, 11) vor corespunde numerelor din zecimal 0,+1,-2,-1.

La reprezentarea pe 3 biţi, cele 8 valori scrise în binar 000b, 001b, 010b, 011b, 100b, 101b, 110b, 111b corespund numerelor 0,+1,+2,+3, -4,-3,-2,-1.

La reprezentarea pe 4 biţi, cele 16 valori scrise în binar 0000b, 0001b, 0010b, 0011b, 0100b, 0101b, 0110b, 0111b, 1000b, 1001b, 1010b, 1011b, 1100b, 1101b, 1110b, 1111b corespund numerelor 0,+1,+2,...,+7,-8,-7, -6,.., -1.

Conversia intuitivă a numerelor pozitive și negative din zecimal în binar și invers

Trecerea din binar în zecimal:

convenția numerelor *fără semn:*

Dacă se specifică valoarea fără semn 110b aceasta are ca echivalent numărul zecimal 6, obținut prin ponderarea cifrelor binare cu puterile corespunzătoare ale bazei: $110b = 1*2^2 + 1*2^1 + 0*2^0 = 4+2+0=6$, unde fiecare cifră de 1 desemnează o cantitate (o putere a lui 2, în funcție de poziția cifrei în cadrul numărului binar).

convenția numerelor *cu semn:*

Dacă se specifică valoarea cu semn 110b aceasta are ca echivalent numărul zecimal -2, obținut prin ponderarea cifrelor binare cu puterile corespunzătoare ale bazei, exact ca în cazul anterior, dar aici, la numerele cu semn, întotdeauna cifra binară c.m.s. se consideră cu semnul minus, restul toate cu plus; astfel, vom avea: 110b = $-1*2^2 + 1*2^1 + 0*2^0 = -4+2+0=-2$.

Această regulă funcționează și în cazul valorilor numere pozitive, să luăm exemplul numărului 010b, care este +2 în zecimal:

010b = $-0*2^2 + 1*2^1 + 0*2^0 = -0+2+0=+2$.

Incercați să rețineți aceste 2 reguli, sunt extrem de importante pentru conversia corectă a numerelor cu semn și respectiv fără semn din binar în zecimal.

Trecerea din zecimal în binar:

Pentru a converti un număr din zecimal în binar, se efectuează împărțiri repetate la 2 și se culeg resturile în ordine inversă obținerii lor.

convenția numerelor *fără semn:*

Să considerăm, de exemplu, numărul zecimal 4:

4:2=2 rest 0, apoi 2:2=1 rest 0, și în final 1:1 =0, rest 1, deci numărul 4 din zecimal se va scrie 100b.

convenția numerelor *cu semn:*

număr pozitiv:

Să considerăm acum exemplul numărului zecimal cu semn +4:

Într-un prim pas, se procedează ca în cazul anterior:

4:2=2 rest 0, apoi 2:2=1 rest 0, și în final 1:1 =0, rest 1, deci numărul 4 din zecimal se va scrie 100b.

Dar apoi, este esențial să se țină cont de faptul că reprezentăm un număr cu semn, deci mai adăugăm un bit de 0 ca bit de semn => +4 = 0100b.

număr negativ:

Să considerăm acum exemplul numărului zecimal cu semn -4:

Într-un prim pas, se procedează ca în cazul anterior, deci se obține reprezentarea echivalentului pozitiv al numărului: +4=0100b

Dar apoi, este esențial să se țină cont de regula de obținere a numerelor negativeși anume regula complement față de 2: vom inversa toți biții reprezentării numărului +4 și vom aduna un bit de 1; astfel, vom obține: 1011b+1=1100b (aceasta este reprezentarea corectă a nr -4, dar el se poate și restrânge la doar 100b).

2.2. SISTEME DE NUMERAŢIE

Informaţia numerică sau numerele din PC se reprezintă prin intermediul unui ***sistem de numeraţie***, adică un ansamblu de reguli folosit pentru scrierea numerelor cu simboluri; aceste simboluri reprezintă ***cifrele*** *sistemului de numeraţie.*

Se numeşte ***bază*** sau *rădăcină* (*q*) a sistemului de numeraţie *numărul de simboluri permise* pentru reprezentarea unei cifre.

Sistemele de numeraţie pot fi:

- *nepoziţionale*: sistemul roman cu cifrele I, V, X, L, C, D, M
  -> 1, 5, 10, 50, 100, 500, 1000;
- *poziţionale*: sistemul arab (zecimal), binar, octal, hexazecimal
  -> 4321 = 4000 + 300 + 20 + 1.

2.2.1. SISTEME DE NUMERAŢIE UTILIZATE ÎN PC

La scrierea numerelor în contextul reprezentării informaţiei în PC se folosesc **sisteme de numeraţie poziţionale**.

Dintre sistemele de numerație poziționale, cele mai uzuale sunt:

sistemul **zecimal** – folosit de oameni în exprimarea numerelor,
sistemul **binar** – folosit pentru exprimarea valorilor în calculator,
sistemul **hexazecimal** – folosit în scrierea mai rapidă a valorilor din calculator.

Tabelul 2.1. Baza şi cifrele sistemelor de numeraţie binar, zecimal, hexazecimal şi octal

Sistem de numeraţie	Baza (q)	Cifrele sistemului de numeraţie
Binar	2	{**0,1**}
Zecimal	10	{**0,1,2,3,4,5,6,7,8,9**}
Hexazecimal	16	{**0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F**}
Octal	8	{**0,1,2,3,4,5,6,7**}

Sistemul zecimal este cel mai folosit sistem în limbajul uzual al oamenilor. Pentru reprezentarea unui număr în zecimal pot fi folosiţi cei 10 digiţi (cifrele 0-9), fiecare digit al unui număr având asociată o putere a lui 10 în funcţie de poziţia sa.

Similar, ***sistemul binar*** este cel mai potrivit pt reprezentarea valorilor numerice într-un PC: pentru a reprezenta un număr în binar, se folosesc doar 2 cifre, 0 şi 1, fiecare cifră a unui număr având asociată o putere a lui 2 în funcţie de poziţia sa.

Sistemul hexazecimal este folosit pentru reprezentarea prescurtată a numerelor binare. Pentru reprezentarea unui număr în hexazecimal, se folosesc cifrele de la 0 la 9, iar ca cifre suplimentare (pentru valorile 10...15) se folosesc literele A – F sau a-f, toate acestea desemnând mulţimea resturilor modulo 16.

Pentru a distinge numerele scrise în diferite baze de numeraţie, la sfârşitul numărului se adaugă o literă ce simbolizează baza, astfel:

B sau **b** pentru numerele scrise ***în binar*** (baza 2),
Q, O, **q** sau **o** pentru numerele scrise ***în octal*** (baza 8),
D sau **d** pentru numerele scrise ***în zecimal*** (baza 10) sau nimic,
H sau **h** pentru numerele scrise ***în hexazecimal*** (baza 16).

De regulă, numerele scrise în baza 10 nu trebuie marcate deoarece această bază se consideră implicită.

Există şi **alte moduri de notare**, cum ar fi:

- scrierea la sfârşitul numărului, în paranteză, a bazei: 10100001(2), 123AB(16) sau
- cu indice jos: 10100001_2, $123AB_{16}$ (aceasta nefiind recomandată în general).

Limbajul de asamblare este *case insensitive* în ceea ce priveşte elementele limbajului: specificarea directivelor, a numelor de regiştri, a mnemonicilor, a specificatorilor de format, a numelui variabilelor, a constantelor, procedurilor sau etichetelor şi chiar a valorilor hexazecimale poate fi realizată folosind majuscule sau minuscule (nu contează dacă se scrie codul cu litere mari sau mici). În schimb, la lucrul cu coduri Ascii, nu este totuna dacă scriem „Ana" sau „ana", de exemplu.

Care sistem de numeraţie îl folosim atunci când interacţionăm cu un SC*?*

La reprezentarea sau utilizarea informaţiei din PC de către oameni, în general se foloseşte sistemul hexazecimal pentru exprimarea valorilor, fiind un sistem mai apropiat de cel zecimal decât cel binar.

Concluzionând, **relaţia între cele 3 sisteme de numeraţie** este una foarte simplă:
- noi oamenii ***gândim*** valorile numerice ***în sistemul zecimal*** (din obișnuință),
- ***scriem*** valorile ***în hexazecimal*** în cadrul programelor care folosesc structuri ale procesorului (**ca programatori**), dar în interior, deoarece așa au fost construite,
- ***PC-urile folosesc reprezentarea binară***.

Indiferent că e nevoie să programăm procesorul sau doar să consultăm valorile regiștrilor lui, trebuie să putem trece de la un sistem de numeraţie la altul cât mai ușor și rapid. Astfel, vom vedea în capitolele următoare care sunt *regulile simple de trecere dintr-o bază în alta (*Figura 2.9*)*, aplicate asupra numerelor ***fără semn*** (Capitolul 3) și apoi vom generaliza aceste reguli asupra numerelor cu semn (Capitolul 4). Înainte însă, vom analiza mai în detaliu diferența dintre aceste 2 categorii de numere.

Figura 2.9 Conversii între cele 3 baze de numeraţie

2.3. CLASIFICAREA NUMERELOR: *FĂRĂ SEMN* VS. *CU SEMN*

Sistemul zecimal deține cifrele 0...9, iar numerele pot fi scrise fără semn (şi se consideră implicit pozitive) sau cu semn având semnul **+** sau **–** specificat înaintea numărului (pentru a desemna numere pozitive, respectiv negative).

Exemplul 2.1

Numărul scris sub forma **4** se va considera ***fără semn***, în timp ce
numărul scris sub forma **+4** va fi considerat pozitiv, iar
cel scris sub forma **-4** va fi negativ,
ultimele două fiind deci scrise în convenţia de reprezentare a numerelor ***cu semn***.

Similar sistemului zecimal, numerele folosite în PC pot fi considerate ***în binar*** ca reprezentând numere zecimale ***fără semn*** sau ***cu semn***, însă ele nu vor avea specificat simbolul + sau -. Similar, numerele hexazecimale nu se vor scrie niciodată precedate de semnul + sau -.

Trebuie subliniat faptul că **în nici un alt sistem de numeraţie (binar, hexazecimal, octal) înafară de cel zecimal nu există simbolul „-" sau „+" care să desemneze semnul unui număr.**

În cazul numerelor ***cu semn***, se va utiliza **o cifră suplimentară** pentru a indica semnul, aceasta fiind prezentă pe poziţia c.m.s. a reprezentării, în analogie cu scrierea folosită de oameni pentru marcarea semnului unui număr scris în zecimal.

Convenţional, la scrierea numărului zecimal ***cu semn***, **în binar** se atribuie:
cifra 0 pentru semnul plus şi
cifra 1 pentru semnul minus.
În cazul numerelor ***fără semn*** însă, fiecare bit de 1 din *reprezentarea binară* a numărului, indiferent de poziția lui, desemnează **o cantitate** (în funcție de poziția lui, reprezintă puterea lui 2 corespunzătoare).

2.3.1. SCRIEREA NR ZECIMALE CU SEMN ÎN BINAR ŞI HEXAZECIMAL

Cum ne dăm seama ce semn are un număr (zecimal) dacă acesta este scris în binar sau hexazecimal?

Deşi semnul + sau – nu mai este prezent la scrierea în binar sau hexazecimal, numărul îşi păstrează semnul şi valoarea pe care le avea în zecimal:

Exemplul 2.2 Exemple de numere în binar, cu semn, pe 8 biţi:
pozitive : 0000 0101b, 0000 1010b, 0001 0000b, 0010 1110b, 0111 1111b și
negative: 1000 0000b, 10001 1101b, 1010 0111b, 1111 1111b, etc.

Extragem astfel regulile:

Pentru ***un număr cu semn scris în binar***:

Dacă un număr cu semn scris în binar are **MSb în 0**, atunci el este **pozitiv**,
iar dacă **MSb al lui este în 1**, atunci numărul este **negativ**.

Exemplul 2.3 Exemple de numere în hexazecimal, cu semn, pe 8 biţi:
pozitive : 05h, 0Ah, 10h, 2Eh, 7Fh și
negative: 80h, 9Dh, A7h, FFh, etc.

Pentru ***un număr cu semn scris în hexazecimal***:

Dacă nr scris în hexazecimal are **cifra c.m.s. între 0 și 7**, atunci el este **pozitiv**,
iar dacă această cifră este **între 8 și Fh**, atunci numărul este **negativ**.

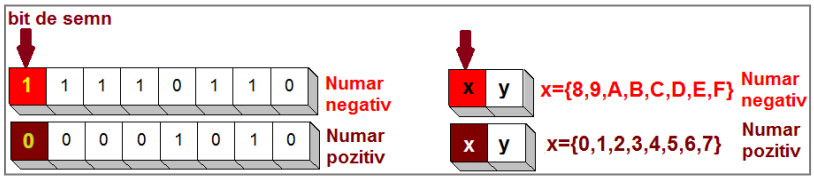

Figura 2.10. Reprezentare sugestivă a numerelor zecimale pozitive și negative (***cu semn***) a) în binar și b) în hexazecimal (pe 8 biţi)

2.3.2. REPREZENTAREA NUMERELOR CU SEMN ŞI FĂRĂ SEMN PE OCTET

Care sunt valorile fără semn şi cele cu semn reprezentabile pe 8 biţi?

Reamintesc aici că folosind un număr de 8 biţi se pot reprezenta 2^8=256 valori diferite. În Figura 2.11 s-a ilustrat grafic *gama numerelor* pe axa orizontală, cu poziţionarea celor 256 valori diferite ***fără semn*** vs. ***cu semn*** care se pot scrie ***folosind un octet***.

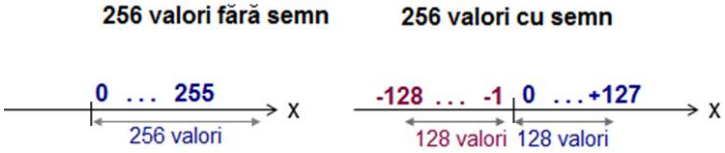

Figura 2.11. Reprezentarea a 256 valori **fără semn** vs. **cu semn**

Așa cum reiese și din Figura 2.11, folosind acelaşi număr de biţi, în reprezentarea numerelor fără semn se pot scrie de două ori mai multe numere pozitive decât în cea cu semn.

Problemele apar atunci când valorile depăşesc domeniul de reprezentare:

- de exemplu, dacă pe 8 biţi adunăm 1111 1111b+1=0000 0000b în loc de 1 0000 0000b (cât ne-am fi aşteptat), dar aceste situaţii sunt detectabile de către programatori şi pot fi gestionate în vederea evitării posibilelor erori; procesorul va semnala aceste situații prin așa numitele “flaguri” aritmetice;
- un alt posibil exemplu îl constituie transformarea unui număr din pozitiv în negativ prin adunare; dacă la un număr cu semn de forma 0111 1111b (care este 127) adunăm 1, de exemplu, acesta devine 1000 0000 (care este -128), ceea ce evident nu ar fi corect d.p.d.v. al operației matematice.

Vom vedea cum gestionează sistemul aceste situații abia în capitolele următoare. Deocamdată, e absolut necesar să ne acomodăm cu reprezentarea numerelor în cele 2 convenții: fără semn și cu semn.

Pentru a se evita ambiguităţile, la reprezentarea numerelor *cu semn* este obligatoriu să se ţină cont de ***gama numerelor***.

2.4. GAMA NUMERELOR

2.4.1. ÎNCADRAREA UNUI NUMĂR ÎN GAMA CORESPUNZĂTOARE LUI

La ce se referă termenul “gama numerelor” ?

Gama numerelor se referă la domeniul numerelor ce se pot reprezenta pe un număr fix de biţi; astfel, se poate deduce regula:

Folosind n biţi,
numerele ***fără semn*** care se pot reprezenta sunt în **gama $[0;2^n-1]$**, iar
numerele ***cu semn*** care se pot reprezenta sunt în **gama $[-2^{n-1};+2^{n-1}-1]$**.

De exemplu, folosind 16 biţi se pot reprezenta 65.536 valori diferite,
cuprinse în gama de valori [0; 65.535] pentru numere fără semn sau
[-32.768; +32.767] pt numere cu semn.

Presupunând că se poate deduce gama numerelor, ***cum se poate găsi numărul de biţi necesar pentru acea gamă***? Este uşor de remarcat că nu trebuie decât să facem legătura cu puterea lui 2 corespunzătoare unui capăt sau celuilalt.

Exemplul 2.4

Aflaţi numărul minim de biţi necesar reprezentării corecte a numărului 3.

Răspuns: Fiind un număr fără semn, se va utiliza relaţia corespunzătoare, încadrându-se numărul 3 în gama [0;3]; astfel, din relaţia 2^n-1=3 va rezulta că nr minim de biţi necesar scrierii corecte a lui 3 este n=2 biți.

Într-adevăr, 3 se va scrie corect în binar ca 11b.

În schimb, dacă enunţul s-ar fi referit la numărul +3, atunci acesta ar fi trebuit încadrat în gama [-4,+3] şi s-ar fi folosit una din relaţiile $+2^{n-1}$-1=+3 sau -2^{n-1}= -4 de unde rezultă n=3 biți. Astfel, am fi avut nevoie de un bit suplimentar pentru a reprezenta corect numărul cu semn +3, iar acest bit suplimentar ar fi fost 0, deoarece numărul este pozitiv și trebuie semnalat acest lucru.

Nu la fel de uşor stau lucrurile pentru reprezentarea numerelor negative (de ex. -3), însă se vor da explicaţiile necesare ulterior. Totuși, dacă enunţul s-ar fi referit la numărul -3, atunci tot în gama [-4,+3] ar fi fost încadrat şi s-ar fi obținut tot n=3.

Exemplul 2.5

Aflaţi numărul minim de biţi necesar reprezentării corecte a numerelor 8, +8 și -8.

Răspuns: Numărul 8, fiind un număr fără semn, se va încadra în gama [0;15]; astfel, din relaţia 2^n-1=15 va rezulta că nr minim de biţi necesar scrierii corecte a lui 8 este n=4 biți. Într-adevăr, 8 se va scrie corect în binar ca 1000b.

Pentru numărul +8, vom folosi gama [-16,+15] şi folosim una din relaţiile $+2^{n-1}$-1=+15 sau -2^{n-1}= -16 de unde rezultă n=5 biți. Deci avem nevoie de un bit suplimentar pentru a reprezenta corect numărul cu semn +8, acest bit suplimentar find 0: numărul +8 se scrie corect pe 5 biți ca 01000b.

Pentru numărul -8, vom folosi gama [-8,+7] şi folosim una din relaţiile $+2^{n-1}$-1=+7 sau -2^{n-1}= -8 de unde rezultă n=4 biți. Vom vedea ulterior că -8=1000b.

Din exemplele anterioare, cineva ar putea extrage în mod eronat regula că numerele cu semn + au nevoie întotdeauna de un bit suplimentar în reprezentarea binară comparativ cu numerele cu semn -. Această presupunere este falsă, să considerăm de exemplu cazul numerelor 5, +5 și -5.

Exemplul 2.6

Aflaţi numărul minim de biţi necesar reprezentării corecte a numerelor 5, +5 și -5.

Răspuns: Pentru 5, fiind un număr fără semn, încadrat în gama [0;7], va rezulta că nr minim de biţi necesar scrierii corecte a lui este n=3 biți. Într-adevăr, 5 se va scrie corect în binar ca 101b.

Pentru numărul +5, vom folosi gama [-8,+7] şi rezultă n=4 biți; numărul +5 se scrie corect pe 4 biți ca 0101b.

Pentru numărul -5, vom folosi tot gama [-8,+7] şi va rezulta tot n=4 biți, iar așa cum vom vedea ulterior, -5=1011b.

2.4.2. REALIZAREA OPERAȚIILOR ARITMETICE CONSIDERÂND GAMA

Atunci când operăm două numere care se scriu pe câte n biți, este posibil ca rezultatul să aibă nevoie de scrierea pe mai mulți biți, în funcție de operație. Să considerăm 2 cazuri separate: cel al adunării și cel al înmulțirii.

Adunarea a două numere scrise pe câte n biți fiecare

Să considerăm două numere care se scriu fiecare pe câte 4 biți, deci în gama numerelor fără semn [0;15], de exemplu: vom aduna 11 + 10 = 21, unde rezultatul depășește gama de 4 biți, având nevoie de un bit suplimentar în reprezentare (numărul 21 se încadrează corect în gama [0; 31] pe 5 biți și nu în [0;15] pe 4 biți).

Similar, am putea considera cazul a două numere cu semn care se scriu fiecare cu câte 4 biți, deci în gama [-8; +7]: de exemplu, să adunăm (+6) + (+5) = (+11) care nu mai încape în această gamă, va avea nevoie de gama [-16; +15], deci vor fi necesari 5 biți la reprezentarea lui.

Practic, indiferent că vorbim despre numerele cu semn sau cele fără semn, atunci când numerele scrise pe n biți și însumate sunt apropiate de extremitatea superioară a gamei, **rezultatul va avea nevoie de scrierea pe n+1 biți.**

Înmulțirea a două numere scrise pe câte n biți fiecare

Să considerăm din nou cele două numere care se scriu fiecare pe câte 4 biți, 11 și 10, de data aceasta pentru a realiza operația de înmulțire a lor. Produsul lor, 110 are nevoie de încadrare în gama [0;127] (nu există gamă mai mică potrivită pentru a-l încadra); astfel, produsul celor 2 numere va avea nevoie de 7 biți pentru a putea fi scris corect.

Similar, în cel de-al doilea caz, al numerelor cu semn +6 și +5, vom obține produsul +30 care se va încadra corect în gama numerelor cu semn [-32; +31]; astfel, produsul va avea nevoie de scrierea cu minim 6 biți.

Este normal să ne punem întrebarea: care este numărul minim de biți necesar pentru a scrie produsul a două numere care se scriu pe n biți fiecare? Putem afla răspunsul la această întrebare considerând valorile extreme ale gamei: de exemplu, să înmulțim 15 * 15 = 225, respectiv (+7) * (+7) = +49.

În primul caz, obținem gama [0; 255] pentru a scrie produsul numerelor fără semn (deci folosind 8 biți), respectiv [-64; +63] pentru a scrie produsul numerelor cu semn (deci folosind 7 biți). Mai rămâne doar să analizăm și cazul înmulțirii a 2 numere negative scrise de asemenea pe n biți: (-8) * (-8) = (+64), număr încadrat corect în gama numerelor cu semn [-128; +127], deci pe 8 biți.

Concluzionând, indiferent că vorbim despre numerele cu semn sau cele fără semn, atunci când numerele scrise pe n biți și înmulțite sunt apropiate de extremitățile gamei în care se reprezintă, **rezultatul se va scrie corect pe un număr de 2n biți**.

Exemple 2.7

Să se specifice numărul minim de biți necesar scrierii produsului următoarelor perechi de numere: a) 25 și 30; b) +30 și +40; c) -30 și -50.

Perechea de numere fără semn 25 și 30 se încadrează în gama [0; 31], deci pe 5 biți, iar rezultatul (750) se va scrie corect pe 10 biți, fiind încadrat în gama [0; 1023].

A doua pereche de numere cu semn +30 și +40 se încadrează în 2 game diferite: numărul +30 aparține gamei [-32; +31], deci pe 6 biți, iar numărul +40 aparține gamei [-64; +63], deci se scrie pe 7 biți; rezultatul (+1200) se va scrie corect pe 12 biți, fiind încadrat în gama [-2048; +2047].

A treia pereche de numere cu semn -30 și -50 se încadrează în aceleași game ca și perechea anterioară, scriindu-se deci pe 6 și respectiv 7 biți; rezultatul (+1500) se va scrie corect pe 12 biți, fiind încadrat în gama [-2048; +2047].

Exemple 2.8

Să se specifice numărul minim de biți necesar scrierii produsului următoarelor perechi de numere: a) 3 și 3; b) +3 și +3; c) -4 și -4.

Numărul fără semn 3 este situat în extremitatea gamei [0; 3], deci pe 2 biți, iar rezultatul (9) se va scrie corect pe 4 biți, fiind încadrat în gama [0; 15].

Numărul cu semn +3 se încadrează în gama [-4; +3], deci pe 3 biți; rezultatul (+9) se va scrie corect pe 5 biți, fiind încadrat în gama [-16; +15].

Numărul cu semn -4 se află la extremitatea cealaltă a gamei [-4; +3], scriindu-se tot pe 3 biți; rezultatul (+16) se va scrie corect în schimb pe 6 biți, fiind încadrat în gama [-32; +31].

2.5. SCRIEREA NUMERELOR SUB FORMĂ DE PUTERI

Cum putem scrie numerele multiplu de 2 la o putere în binar?

Dar numerele multiplu de 16 la o putere în hexazecimal?

În analogie cu scrierea numerelor multiplu de 10 la o putere, vom analiza și valorile binare multiplu de 2 la o putere, respectiv numerele multiplu de 16 la o putere, scrise în hexazecimal.

Tabelul 2.2 Scrierea numerelor multiplu de bază la puterile 1, 2, 3, unde baza=10,2,16

Baza: 10 (zecimal)	Baza: 2 (binar)	Baza: 16 (hexazecimal)
nr multiplu de 10: **x0**	nr multiplu de 2: **x0**b	nr multiplu de 16: **x0**h
nr multiplu de 100: **x00**	nr multiplu de 4: **x00**b	nr multiplu de 256: **x00**h
nr multiplu de 1000: **x000**	nr multiplu de 8: **x000**b	nr multiplu de 4096: **x000**h

Tot așa cum scriem un număr multiplu de 10, 100, 1000, etc în zecimal, vom putea scrie și în binar numere multiplu de 2, de forma: **x0**b, sau multiplu de 2^2=4: **x00**b, sau multiplu de 2^3=8: **x000**b; idem, putem scrie un număr multiplu de 16 în hexazecimal sub forma: **x0**h, sau multiplu de 16^2=256 sub forma: **x00**h, ș.a.m.d, unde x a reprezentat orice combinație de cifre în baza respectivă.

Exemple 2.9

Exemple de numere multiplu de 2, scrise în binar: 1101**0**b, 01011**0**b

Exemple de numere multiplu de 4, scrise în binar: 1011**00**b, 0101**00**b

Exemple de numere multiplu de 8, scrise în binar: 1101**000**b, 0111**000**b

Exemple de numere multiplu de 16, scrise în hexazecimal: 12A**0**h, 8FE1**0**h

Exemple de numere multiplu de 256, scrise în hexazecimal: 3B**00**h, DAB5**00**h

Exemple de numere multiplu de 4096, scrise în hexazecimal: 28C**000**h, 9AE7**000**h

3. LUCRUL CU NUMERELE FĂRĂ SEMN

Pentru început, vom analiza diferite metode de conversie a numerelor fără semn dintr-o bază în alta, iar apoi vom parcurge și regulile ce trebuie urmate atunci când operăm cu numere fără semn.

3.1. CONVERSII DE NUMERE FĂRĂ SEMN

Aceste reguli se vor mai relua în Capitolul 4, când se vor generaliza în vederea aplicării asupra numerelor cu semn. Pentru acomodare, aici le vom utiliza doar pentru ***conversia numerelor fără semn***.

3.1.1. TRECEREA RAPIDĂ DINTR-UN SISTEM DE NUMERAŢIE ÎN ALTUL

Pentru a trece uşor valorile numerice mici (cuprinse între 0 și 15) dintr-un sistem de numeraţie în altul, dar în special între hexazecimal şi binar, se pot folosi 2 metode:

1) consultarea unui ***tabel*** (Tabelul 3.1) ***în care sunt scrise numerele de la 0 la 15 în toate bazele dorite*** sau
2) folosirea unui ***şablon*** (Figura 3.1) ***format din 4 cifre binare ce se poate folosi pentru a scrie o cifră hexazecimală***.

Tabelul 3.1. Numerele de la 0 la 15 scrise în sistemul zecimal, binar, hexazecimal şi octal

Sistemul zecimal q = 10	Sistemul binar q = 2	Sistemul hexazecimal q = 16	Sistemul octal q = 8
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	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

Într-o primă fază, până ne acomodăm cu regulile de conversie, putem folosi Tabelul 3.1, care arată toate combinaţiile posibile de 4 biţi scrise în cele 4 sisteme de numeraţie (mulţimea resturilor modulo q). Astfel, din tabel se poate observa de exemplu că numărul 11 din zecimal (pronunțat unsprezece) se scrie **B** în hexazecimal, **13** în octal (pronunțat unu-trei și nu treisprezece) şi **1011** în binar (pronunțat unu-zero-unu-unu și nu o mie unsprezece). Valoarea cea mai mare ce se poate scrie pe 4 biţi este 1111b şi reprezintă numărul 15, Fh sau 17q.

O altă posibilă variantă pentru a trece din hexazecimal în binar (sau invers) este prin folosirea șablonului din Figura 3.1, unde **b3**,**b2**,**b1**,**b0** sunt cifre binare (0/ 1) care se ponderează cu puterile corespunzătoare ale lui 2, adică **8**, **4**, **2**, resp. **1**. Acest șablon însă funcţionează doar pentru o singură cifră hexazecimală, deci numere care se scriu cu maxim 4 biţi:

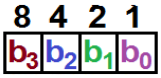

Figura 3.1 Șablon posibil la conversia din zecimal în binar sau invers pt o cifră hexazecimală

Cum aplicăm cele 2 metode pentru a converti din binar în hexazecimal ?

La mod general, la conversia unui număr din binar în hexazecimal, se procedează în felul următor: se formează grupuri de câte 4 cifre binare pornind dinspre c.m.p.s. bit (deci din dreapta spre stânga), înlocuind apoi fiecare grup cu cifra hexazecimală corespunzătoare (folosind Tabelul 3.1 sau şablonul din Figura 3.1).

Exemplul 3.1 Dacă vrem să transformăm numărul **1010**b în hexazecimal (folosind şablonul), după ponderarea cu puterile lui 2 corespunzătoare, vom obţine un opt şi un doi (nu avem nici 4 şi nici 1) care însumate dau zece (scris A în hexazecimal).

1010b = **1*8** + **1*2** = **8** + **2**=10=Ah;

Exemplul 3.2 Pentru 11010b vom obține 1Ah, astfel:

11010b = **000**11010b = **0001 1010** b = **1 A** h, unde s-au adăugat 3 biţi de 0 pentru a forma un grup complet de 4 biţi.

Cum aplicăm cele 2 metode pentru a converti din hexazecimal în binar ?

Fiecare cifră hexazecimală va fi înlocuită cu grupul corespunzător de 4 cifre binare.

Exemplul 3.3 Invers, dacă vrem să vedem cum se scrie numărul D din hexazecimal în binar (folosind şablonul), va trebui să-l scriem în zecimal ca 13 şi să-l descompunem în **8**+**4**+**1**, adică un opt, un patru şi un unu şi se va scrie în binar sub forma **1101**b, astfel : Dh=13 = **8** + **4** + **1** =**1*8** + **1*4** + **1*1** =**1101**b

Exemplul 3.4 Utilizând Tabelul 3.1 sau şablonul din Figura 3.1, este uşor de observat că numărul **9A**h scris în hexazecimal se va transforma în binar în şirul de 8 cifre binare: **1001 1010**b: 9Ah = **9 A** h = **1001 1010**b = 10011010b.

3.1.2. CONVERSIA DIN ZECIMAL ÎN BINAR ȘI INVERS

Conversia din zecimal în binar

Metoda 1 (**prin împărţiri succesive**):

Se împarte în mod repetat numărul din zecimal la baza 2 până la obţinerea câtului 0 (la fiecare împărţire, noul cât devine deîmpărţit) şi se culeg resturile obţinute în ordine inversă.

Metoda 2 (**prin scăderea puterii maxime**):

Se va căuta puterea cea mai mare a bazei care se poate scădea din numărul de reprezentat şi se efectuează scăderea, notând puterea respectivă. Există posibilitatea de a fi necesare mai multe scăderi ale aceleiaşi puteri, iar astfel se va nota multiplul puterii. Operaţia se repetă până la obţinerea diferenţei 0, noua valoare a descăzutului fiind numărul rămas după efectuarea scăderii.

Exemplul 3.5

Conversia numărului 37 din zecimal în binar folosind ***metoda 1*** (baza=2).

Sunt posibile 2 forme de scriere:

Forma de scriere I :

37 : 2 cât 18 rest **1**
18 : 2 cât 9 rest **0**
9 : 2 cât 4 rest **1**
4 : 2 cât 2 rest **0**
2 : 2 cât 1 rest **0**
1 : 2 cât 0 rest **1**

=> 37 = **1 0 0 1 0 1** $_b$

Forma de scriere II :

37| 2
36| 18| 2
= **1**| 18| 9| 2
= **0**| 8| 4| 2
=**1**| 4| 2| 2
=**0**| 2| 1| 2
=**0**| 0| 0
=**1**|

Exemplul 3.6

Conversia numărului 37 din zecimal în binar folosind ***metoda 2*** (baza=2):

Avem puterile lui 2 ≤ 37: {2^5=32, 2^4=16, 2^3=8, 2^2=4, 2^1=2, 2^0=1}

37 - **32** = 5 (se notează 2^5 o dată)
5 - **4** = 1 (se notează 2^2 o dată)
1 - **1** = 0 (se notează 2^0 o dată) =>

2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	**1**	0	**1**

pe poziţiile 5, 2, 0 se va pune (în general multiplul puterii) cifra 1, iar în rest cifra 0.

Exemplul 3.7

Vom converti numărul 4321 din zecimal în binar, folosind ***metoda împărţirilor succesive***:

Forma de scriere I :

4321 : 2 cât 2160 rest **1**
2160 : 2 cât 1080 rest **0**
1080 : 2 cât 540 rest **0**
540 : 2 cât 270 rest **0**
270 : 2 cât 135 rest **0**
135 : 2 cât 67 rest **1**
67 : 2 cât 33 rest **1**
33 : 2 cât 16 rest **1**
16 : 2 cât 8 rest **0**
8 : 2 cât 4 rest **0**
4 : 2 cât 2 rest **0**
2 : 2 cât 1 rest **0**
1 : 2 cât 0 rest **1**

4321=**1 0000 1110 0001** b

Forma de scriere II :

4321| 2
4320| **2160**| 2
= **1**| 2160|**1080**| 2
= **0**|1080|**540**| 2
=**0**|540|**270**| 2
=**0**| 270|**135**| 2
=**0**| 134|**67**| 2
=**1**|66|**33**| 2
=**1**|32|**16**| 2
=**1**|16|**8**| 2
=**0**|8|**4**| 2
=**0**|4|**2**|2
=**0**|2|**1**| 2
=**0**|0| **0**
=**1**

4321:2=2160 (reţinem restul **1,** care va fi bitul b0), iar câtul devine noul deîmpărţit,
deci vom avea 2160:2=1080 (reţinem restul **0** - va fi bitul b1),
apoi 1080:2=540 (reţinem restul **0** - va fi bitul b2) ș.a.m.d.,
iar ***când se obţine câtul 0 ne oprim*** și ***scriem aceste resturi în ordine inversă***:

4321 = **1 0 0 0 0 1 1 1 0 0 0 0 1** b

Exemplul 3.8

Folosind ***metoda prin scăderea puterii maxime***, vom avea:

Reprezentarea numărului 4321 în binar folosind metoda 2 (baza=2) implică scrierea puterilor lui 2 ≤ 4321:
{2^{12}=4096, 2^{11}=2048, 2^{10}=1024, 2^9=512, 2^8=256, 2^7=128, 2^6=64, 2^5=32 , 2^4=16, 2^3=8, 2^2=4, 2^1=2, 2^0=1}

4321 - **4096** = 225 (se notează 2^{12} o dată)
225 - **128** = 97 (se notează 2^7 o dată)
97 - **64** = 33 (se notează 2^6 o dată)
33 - **32** = 1 (se notează 2^5 o dată)
1 - **1** = 0 (se notează 2^0 o dată)

deci vom obține:

2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	0	0	1	1	1	0	0	0	0	1

pe poziţiile 12, 7, 6, 5, 0 se va pune cifra 1, iar în rest cifra 0.

Conversia din binar în zecimal

Conversia inversă, a numărului din binar în zecimal, se realizează în mod uzual prin ***metoda ponderării cu puterile lui 2***: pentru a obţine ***valoarea zecimală ca număr fără semn*** dintr-un număr scris în binar, de exemplu 1 0 0 0 0 1 1 1 0 0 0 0 1b, va trebui să ponderăm fiecare bit (cifră binară) cu puterea lui 2 corespunzătoare (puterile se scriu începând de la 0, dinspre dreapta spre stânga) şi apoi să adunăm:

Exemplul 3.9

Reprezentarea numărului 1 0000 1110 0001b din binar în zecimal, știind că este scris *ca număr fără semn*:

12 11 10 9 8 7 6 5 4 3 2 1 0

1 0000 1110 0001b = 1 0 0 0 0 1 1 1 0 0 0 0 1b = $1*2^{12}+1*2^7+1*2^6+1*2^5+1*2^0$

= 4096 + 128 + 64 + 32 + 1 = 4321

Exemplul 3.10

Reprezentarea numărului 100101b din binar în zecimal, știind că este scris *ca număr fără semn*:

$100101_b = 1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 32 + 4 + 1 = 37$

3.1.3. CONVERSIA DIN ZECIMAL ÎN HEXAZECIMAL ŞI INVERS

Pentru ***conversia din zecimal în hexazecimal*** sunt posibile 3 metode:

- se pot adapta cele 2 metode de conversie aplicate la conversia din zecimal în binar (dar baza va fi 16),
  Metoda 1 **– prin împărţiri succesive** şi
  Metoda 2 **– prin scăderea puterii maxime;**
  în general se alege metoda 1: se împarte numărul în mod repetat la 16 şi se culeg resturile în ordine inversă obținerii lor;
- o altă metodă, ***Metoda 3***, considerată în general și cea mai simplă, este cea cu ***trecere prin binar*** (așa cum am văzut deja la secțiunea 3.1.1): se transformă numărul din zecimal în binar şi apoi, formând grupuri de câte 4 biţi, se scriu cifrele hexazecimale corespunzătoare.

Metoda 1. Folosind metoda **prin împărţiri succesive** vom avea:

Exemplul 3.11

Conversia numărului 37 din zecimal în hexazecimal (baza=16)

37: 16 -> cât 2 rest 5
2: 16 -> cât 0 rest 2 -> 37 = 32 + 5 = $2*16^1 + 5*16^0 = 25_h$

Exemplul 3.12 Pentru un număr mai mare, de exemplu 4321, se va proceda similar:

4321:16 = 270 (reţinem restul 1 - va fi cifra hexa de ordin 0), apoi
270:16 = 16 (reţinem restul 14=E - va fi cifra hexa de ordin 1), apoi
16:16 = 1 (reţinem restul 0 - va fi cifra hexa de ordin 2), şi în final
1:16 = 0 (avem restul 1 – aceasta va fi cifra hexa de ordin 3)
s-a obţinut câtul 0, deci se scriu aceste resturi în ordine inversă: 4321=10E1h

Metoda 2. Folosind metoda **prin scăderea puterii maxime** vom avea:

Exemplul 3.13
Reprezentarea pentru 37 în hexazecimal (baza=16).
37 -16 = 21 (16^1)
21 -16 = 5 (16^1)
5 - 1 = 4 (16^0)
4 - 1 = 3 (16^0)
3 - 1 = 2 (16^0)
2 - 1 = 1 (16^0)
1 - 1 = 0 (16^0) => 25 h

Exemplul 3.14
Reprezentarea pentru 4321 în hexazecimal (baza=16).
4321 -4096 = 225 (16^3)
225 - 16 = 209 (16^1)
209 - 16 = 193 (16^1)
... (de încă 11 ori vom scădea 16)
17 - 16 = 1 (16^1)
1 - 1 = 0 (16^0) => 1 0 E 1 h

Metoda 3: Cu trecere prin binar vom avea:

Exemplul 3.15
Pentru 37:
37 = 100101b
= **00**100101b
= 0010 0101b = 25h

Exemplul 3.16
Pentru 4321:
4321= 1000011100001b
= **000**1000011100001b
= 0001 0000 1110 0001b = 10E1h

Se observă că atunci când numărului scris în binar îi lipsesc biți pentru a se forma grup de 4 biți necesari scrierii în hexazecimal, se adăugă biți de 0 în extremitatea stângă a numărului scris în binar. Acești biți de 0 nu vor afecta valoarea numărului zecimal, deoarece la ponderarea cu puterea lui 2, ei vor fi nuli.

Conversia din hexazecimal în zecimal

Conversia inversă, a numărului din hexazecimal în zecimal, se realizează similar celei din binar în zecimal, deci prin ***metoda ponderării cu puterile lui 16.***

Exemplul 3.17

Pentru reprezentarea numărului 25h din hexazecimal în zecimal vom avea:

$25h = 2*16^1 + 5*16^0 = 32 + 5 = 37$

iar pentru reprezentarea numărului 10E1h din hexazecimal în zecimal vom avea:

$10E1h = 1*16^3 + 0*16^2 + 14*16^1 + 1*16^0 = 4096 + 0 + 224 + 1 = 4321$

După cum am precizat, pentru conversia unui număr din zecimal în hexazecimal se poate opta pentru realizarea conversiei cu trecere prin binar, iar acest lucru este valabil și la conversia în sens invers:

Exemplul 3.18

Pentru reprezentarea numărului 25h din hexazecimal în zecimal vom avea:

$25h = 0010\ 0101b = 1*2^5 + 1*2^2 + 1*2^0 = 32 + 4 + 1 = 37$

iar pentru reprezentarea numărului 10E1h din hexazecimal în zecimal vom avea:

$10E1h = 0001\ 0000\ 1110\ 0001b = \mathbf{1*2^{12}+1*2^7+1*2^6+1*2^5+1*2^0}$

$= \mathbf{4096 + 128 + 64 + 32 + 1} = 4321$

3.1.4. CONVERSIA FOLOSIND BAZA OCTALĂ

Conversia numerelor din/ în octal

Se procedează în mod similar conversiei din/ în hexazecimal, singura diferenţă fiind la trecerea din binar în octal sau din octal în binar, unde în locul formării de grupuri de câte 4 biţi, se vor forma grupuri de câte 3 biţi şi se va folosi cifra în octal corespunzătoare (în loc de tetrade se folosesc triade). Se va ţine cont de asemenea de baza 8 a sistemului şi de cifrele octale asociate.

Exemplul 3.19

4321:8 = 540 (reţinem restul **1** - va fi cifra octală de ordin 0), apoi
540:8 = 67 (reţinem restul **4** - va fi cifra octală de ordin 1), apoi
67:8 = 8 (reţinem restul **3** - va fi cifra octală de ordin 2),
8:8 = 1 (reţinem restul **0** - va fi cifra octală de ordin 3),
1:8 = 0 (reţinem restul **1** - va fi cifra octală de ordin 3), ș.a.m.d. iar când se obţine câtul 0 se scriu aceste resturi
în ordine inversă: 4321=**10341**q
sau dacă se dorește folosirea metodei *cu trecere prin binar*:
=> 4321=**0**001000011100001b = **001 000 011 100 001**b = **10341**q sau **010341**q.

Exemplul 3.20

37:8 = 4 (reţinem restul **5** - va fi cifra octală de ordin 0), apoi
4:8 = 0 (reţinem restul **4** - va fi cifra octală de ordin 1).
Am obținut câtul 0, deci vom scrie aceste resturi
în ordine inversă: 37 = **45**q
sau dacă se dorește folosirea metodei *cu trecere prin binar*:
37 = **100 101**b = **45**q sau **045**q.

3.2. EXTENSIA ȘI CONTRACTAREA NUMERELOR

Putem transforma o valoare reprezentată pe 8 biţi la una pe 16 biţi, sau invers?

Pentru numerele reprezentate în calculator se pot aplica ***operaţii de extensie*** la un număr mai mare de biţi (de exemplu de la 8 la 16 biţi, de la 16 biţi la 32 biţi, etc). Operaţia de extensie se poate aplica de fapt, la orice număr de biţi: de exemplu, în cazurile analizate mai sus, de la 7 la 8 biţi sau de la 14 la 16 biţi.

În mod similar, se poate dori o operaţie în sens invers, de la un număr mai mare de biţi înspre unul mai mic (de exemplu după realizarea unei operaţii de înmulţire la care se asigură automat din procesor dimensiune dublă pentru stocarea rezultatului, dar rezultatul obţinut este mic şi deci s-ar putea reduce ca dimensiune, îl vom scrie într-un registru de dimensiune mai mică); acestea se numesc ***operaţii de contracţie sau contractare***. Operaţia de contractare se poate aplica de fapt, ca în cazul extensiei, la orice număr de biţi: de exemplu, de la 9 la 8 biţi sau de la 16 la 14 biţi, etc.

Dacă operaţiile de extensie a numerelor se pot realiza prin anumite instrucţiuni specifice în limbajul procesoarelor x86 (vom vedea ulterior de exemplu instrucţiunea CBW care convertește un operand de tip byte la unul de tip word, deci asigură conversia valorii respective de la 8 la 16 biţi), nu se poate afirma același lucru și despre operaţiile de contractare (nu există "*instrucţiuni de contractare*").

Aceste aspecte ţin mai mult de manevrarea valorilor în regiștri (scalarea corectă a operanzilor în regiștri sau în zone din memorie) decât de folosirea lor în cadrul instrucţiunilor; toate aceste aspecte sunt deosebit de importante în programarea în limbaj de asamblare.

3.2.1. EXTENSIA ŞI CONTRACTAREA NUMERELOR FĂRĂ SEMN

Pentru a realiza operaţia de extensie a unui număr, este esenţială cunoaşterea convenţiei în care se consideră acel număr: este ***un număr considerat fără semn*** sau este ***un număr considerat cu semn***? Aceasta, deoarece:

Extensia unui număr fără semn se realizează ***întotdeauna cu bit de 0***.

Exemplu 3.21
Extinderea numărului ***fără semn*** 128 = 80h pe 16 biţi va da 0080h, iar pe 32 biţi va da 0000 0080h.
Indiferent că am scris valoarea pe pe 8 biţi, 16 biţi sau pe 32 biţi, în zecimal ea trebuie să rămână aceeaşi (ca număr fără semn): 128.

Regula aplicată la verificare pentru numere fără semn este: toţi termenii reprezintă cantităţi, acestea fiind ignorate doar dacă cifra respectivă este 0.

Valoarea numărului ***N*** în baza 10 (N fiind scris în baza ***2***) se va putea calcula după relaţia (3.1):

$$N(10) = a_n 2^n + a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \ldots + a_1 2^1 + a_0 2^0 \qquad (3.1).$$

Pentru ***contractarea unui număr scris ca număr fără semn***, deoarece la extensie s-au adăugat doar biţi de 0 (întotdeauna), este evident că nu vom putea elimina (tăia) biţi de 1 din reprezentarea numărului iniţial, deoarece aceştia nu au fost adăugaţi (la o eventuală extensie).

Exemple 3.22
a) FF80h nu poate fi contractat; FF40h nu poate fi contractat;
b) 0020h poate fi contractat la 20h, sau chiar mai mult, e posibil inclusiv la un număr de doar 6 biţi[4] (îl privim ca 10 0000b);

Exemple 3.23

1 = 1b (scris pe un singur bit) = 0000 0001b (scris pe 8 biţi)
= 01h (scris pe 8 biţi) = 0001h (scris pe 16 biţi) = 0000 0001h (scris pe 32 biţi)
2 = 10b (scris pe 2 biţi) = 0000 0010b (scris pe 8 biţi)
= 02h (scris pe 8 biţi) = 0002h (scris pe 16 biţi) = 0000 0002h (scris pe 32 biţi)
4 = 100b (scris pe 3 biţi) = 0000 0100b (scris pe 8 biţi)
= 04h (scris pe 8 biţi) = 0004h (scris pe 16 biţi) = 0000 0004h (scris pe 32 biţi)
255 = 1111 1111b (scris pe 8 biţi)
= FFh (scris pe 8 biţi) = 00FFh (scris pe 16 biţi) = 0000 00FFh (scris pe 32 biţi)
512 = 10 0000 0000b (scris pe 10 biţi) = 0010 0000 0000 (scris pe 12 biţi)
= 200h (scris pe 12 biţi) = 0200h (scris pe 16 biţi)

3.3. OPERAȚII CU NUMERE FĂRĂ SEMN

Ne referim aici la operaţii de adunare şi scădere care **nu** generează transport în exterior, așa cum apare, de exemplu, la operaţia din zecimal: 31+94=125, nici sub formă de transport înafara reprezentării numerelor *la adunare* şi nici sub formă de împrumut din exterior *la scădere* (precum la 125-31, de exemplu). Vom trata aceste cazuri ulterior, atunci când vom vorbi și despre flagurile procesorului.

3.3.1. ADUNAREA ȘI SCĂDEREA NUMERELOR BINARE

Atunci când operăm două valori scrise în binar (vom aborda aici doar operații de bază, precum adunarea și scăderea), rezultatul va trebui scris tot în binar.

Pentru realizarea calculului, în general se procedează astfel:

- se foloseşte ***regula aritmetică*** de la adunarea a 2 biţi, respectiv de la scăderea lor, sau
- se foloseşte ***roata numerelor*** – utilă în special atunci când numărul de biţi folosit pentru reprezentarea valorii respective este mic şi atunci când una dintre valorile de operat este de asemenea mică.

Adunarea şi scăderea numerelor în binar folosind reguli aritmetice

Operaţia de adunare în binar se realizează asemănător celei din zecimal: la adunarea a 2 cifre binare (a se consulta regula din Tabelul 3.2, din stânga) există posibilitatea de a apărea transport (dacă adunăm 1+1 şi valoarea sumei este 2, deci egală cu baza). Acest transport va fi *văzut ca o unitate* ce se va propaga înspre stânga, deci înspre cifra binară de rang imediat superior.

[4] Trebuie ţinut cont de faptul că în arhitectura x86 nu există registru de 6 biţi

Tabelul 3.2 Reguli de obţinere a biţilor la adunarea şi scăderea valorilor în binar

***Adunarea**: adunarea a două cifre binare*

a	b	transport	Suma a+b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

***Scăderea:** scăderea a două cifre binare*

a	b	împrumut	Diferența a-b
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Exemple 3.24 Să adunăm 2 valori care nu generează transport între cifrele operate, de exemplu, 43 + 24 = 67 pentru a ne reaminti modul de operare la adunarea în zecimal, așa cum am învățat în clasele primare. Un exemplu similar, în binar, de operație care nu generează transport între cifrele binare, poate fi reprezentat de adunarea binară a valorilor 0100b și 1010b. Cele 2 exemple nu reprezintă același calcul; au fost alese astfel, doar pentru că procedeul de lucru este asemănător. Similar, am ales apoi alte perechi de valori care operate să producă transport între cel puțin o cifră zecimală (3+9=12), respectiv binară (1+1=2).

Fără transport:

în zecimal	în binar
43 +	0100 b +
24	1010 b
67	1110 b

Cu transport:

în zecimal	în binar
1	1 1 1
43 +	0101 b +
29	0111 b
72	1100 b

Pentru verificarea operației în binar, se recomandă conversia tuturor valorilor din binar în zecimal (atât termenii adunării cât şi suma obţinută) şi verificarea corectitudinii matematice a calculului realizat în zecimal.

Pentru cele 2 exemple de la adunarea în binar, avem:

4 + 10 = 14 („corect") şi 5 + 7 = 12 („corect").

Operaţia de scădere în binar se realizează şi aceasta asemănător celei din zecimal: la scăderea a 2 cifre binare (regula din Tabelul 3.2, din dreapta) există posibilitatea de a apărea nevoia de împrumut (dacă bitul descăzut este mai mic decât bitul scăzător, precum atunci când operăm 0-1); acest împrumut se va considera de la cifra de rang următor şi va fi *văzut ca* ***o unitate egală cu baza*** pe poziţia biţilor unde se va realiza scăderea (deci se va propaga înspre dreapta).

Exemple 3.25

Asemănător Exemplului 3.21, în partea stângă vom scădea 2 valori pentru care nu apare nevoia de împrumut între cifrele zecimale (43-21=22), respectiv binare (0110b-0100b=0010b); în partea dreaptă, am considerat alte 2 perechi de valori zecimale, respectiv binare, pentru care se generează transport între cifrele operate. De exemplu, la operarea cifrelor zecimale 3-9 va fi nevoie de împrumut de la cifra zecimală superioară. Împrumutul va fi văzut ca baza, deci 10. În schimb, la operarea în binar, împrumutul va fi văzut ca 2.

Fără transport:

în zecimal	în binar
43 -	0110 b -
21	0100 b
22	0010 b

Cu transport:

în zecimal	în binar
. 10	. 2
43 -	1011 b -
29	0101 b
14	0110 b

Similar, pentru verificare, se recomandă conversia tuturor valorilor din binar în zecimal (atât descăzutul şi scăzătorul, cât şi diferenţa) şi verificarea corectitudinii calculului realizat în zecimal.

Pentru cele 2 exemple de la scăderea în binar, avem:

6 – 4 = 2 („corect") şi 11 - 5 = 6 („corect").

Greşeli frecvent întâlnite:

De-a lungul carierei didactice, am întâlnit cazuri în care unele persoane au convertit o valoare scrisă în zecimal în binar transformând fiecare cifră zecimală în echivalentul ei binar (aşa cum am procedat la conversia din hexazecimal în binar). O astfel de modalitate este total eronată, nu vom proceda astfel niciodată ! Doar valorile hexazecimale pot fi transformate în binar cifră cu cifră, nu şi cele zecimale: 23 = ~~0010~~ ~~0011~~b = 17h = 0001 0111b

O altă greşeală sau confuzie des întâlnită: unii credeau că dacă la adunarea a 2 valori în zecimal a apărut transport, atunci şi la adunarea aceloraşi 2 valori dar în altă bază, de exemplu în binar, apare transport. Nu trebuie să facem astfel de analogii niciodată; să NU analizăm în acest fel operaţiile de adunare şi scădere în baze diferite. Ele nu prezintă similitudini, atenție !

Adunarea şi scăderea numerelor în binar folosind roata numerelor

Putem folosi roata numerelor pe post de socotitoare, până ne obişnuim cu regulile de adunare şi scădere în binar sau pentru verificarea unui calcul; totuşi, aceasta implică desenarea roţii adaptată la acel număr de biţi pe care se operează. În plus, este indicat a scrie numărul mai mare ca fiind primul număr (cel de la care se începe deplasarea pe roată).

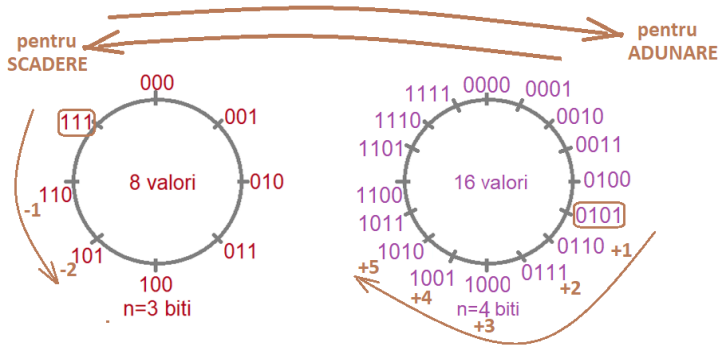

Figura 3.2 Reprezentarea valorilor binare pe roata numerelor pentru n=3 biţi şi n=4 biţi cu ilustrarea operației de scădere 111b-2=101b pe 3 biți și respectiv 0101b+5=1010b pe 4 biți

Exemple 3.26

Vom realiza următoarele calcule în binar, considerând numerele reprezentate pe 3 biţi. Vom folosi roata numerelor pentru a observa modul de realizare al operaţiei de scădere 7-2=5 (Figura 3.2, stânga, cu 8 valori):

7 - 2 = 5 are operaţie echivalentă în binar: 111b - 010b = 101b (identificând poziția valorii 111b pe roata numerelor, se parcurg doi pași pe roata numerelor în sens invers acelor de ceasornic şi se obţine 101b)

Exemple 3.27

Vom considera acum numerele reprezentate pe 4 biţi și vom folosi roata numerelor pentru a observa modul de realizare al operaţiei de adunare 5+5=10 (Figura 3.2, dreapta, cu 16 valori):

5 + 5 = 10 are operaţie echivalentă în binar: 0101b + 0101b = 1010b (plecând de la valoarea 0101b, se parcurg cinci paşi pe roata numerelor în sensul acelor de ceasornic şi se obţine 1010b)

La adunare, se recomandă scrierea numărului mai mare ca prim termen, cel de la care se începe numărarea; astfel, pe roată ne vom deplasa mai puțini pași. De exemplu, dacă vom vrea să operăm 4+10 (deci să începem pe roată de la poziția numărului 4 și să ne deplasăm 10 pași pe roată), mai indicat ar fi să operăm 10+4 (să începem pe roată de la poziția numărului 10 și să ne deplasăm 4 pași).

3.2.2. ADUNAREA ȘI SCĂDEREA NUMERELOR HEXAZECIMALE

Atunci când operăm două numere scrise în hexazecimal, rezultatul va trebui scris tot în hexazecimal. Intuitiv, putem afirma că aici nu este utilă reprezentarea folosind roata numerelor, deci vom opera doar ***folosind regulile aritmetice***.

Adunarea numerelor hexazecimale

Operaţia de adunare în hexazecimal se realizează asemănător celei din zecimal: la adunarea a 2 cifre hexazecimale, există posibilitatea de a apărea transport.

Dacă adunăm de exemplu A+7, vom avea de fapt 10+7=17 şi atunci când valoarea sumei depășește baza (16 în acest caz și nu 2 ca la binar), va apărea un transport înspre cifra de rang superior. Exact la fel procedam în clasele primare când transportam zece. Acest transport va fi ***văzut ca o unitate*** ce se va propaga înspre stânga, deci înspre cifra hexazecimală de rang imediat superior. Trebuie subliniat că o valoare 16 care pleacă de pe poziția cifrei hexazecimale curente, va ajunge ca o unitate, deci un 1 pe poziția imediat superioară.

Exemple 3.28

Fără transport:

în zecimal		în hexazecimal		
43	+	1A	h	+
24		32	h	
67		4C	h	

Cu transport:

în zecimal		în hexazecimal		
1		1		
43	+	1A	h	+
29		37	h	
72		51	h	

Pentru verificare, se recomandă conversia tuturor valorilor din hexazecimal în zecimal (atât termenii adunării cât şi suma obţinută) şi verificarea corectitudinii matematice a calculului realizat în zecimal.

Pentru cele 2 exemple de la adunare, avem:

26 + 50 = 76 („corect”) şi 26 + 55 = 81 („corect”).

Scăderea numerelor hexazecimale

Operaţia de scădere în hexazecimal se realizează şi aceasta asemănător celei din zecimal: la scăderea a 2 cifre hexazecimale, există posibilitatea de a apărea nevoia de împrumut (dacă cifra descăzutului este mai mică decât cea a scăzătorului); acest împrumut se va considera de la cifra de rang următor şi va fi văzut ca o unitate egală cu baza (deci 16 în acest caz) pe poziţia cifrei hexazecimale unde se va realiza scăderea (deci se va propaga înspre dreapta).

Exemple 3.29

Fără transport:					***Cu transport:***				
în zecimal		în binar			în zecimal		în binar		
					. 10		. 16		
43	-	4A	h	-	43	-	4A	b	-
21		16	h		29		1E	b	
22		34	h		14		2C	b	

La scăderea cu transport, cum operația A-E nu se putea realiza, s-a împrumutat o unitate de la cifra superioară. Astfel, vom scădea pe E din 16+A și vom obține C.

Similar, pentru verificare, se recomandă conversia tuturor valorilor din binar în zecimal (atât descăzutul şi scăzătorul, cât şi diferenţa) şi verificarea corectitudinii calculului realizat în zecimal.

Pentru cele 2 exemple de la scădere, avem:

74 – 22 = 52 („corect") şi 74 - 30 = 44 („corect").

Cum se aseamănă operaţiile de adunare şi scădere în diferite baze ?

Analogia modului de realizare al operaţiilor în binar şi hexazecimal cu modul de realizare al operaţiilor în zecimal: adunarea în binar şi hexazecimal se va realiza similar modului de adunare a valorilor în zecimal, cu **transportarea unei unităţi** (unde această unitate reprezintă ***10*** în zecimal, ***2*** în binar şi ***16*** în hexazecimal) ***în caz de depășire.*** Reamintesc aici că cifrele sistemului de reprezentare aparţin unui *sistem modulo n*, unde n este baza sistemului de reprezentare. Depăşirea poate să apară şi în caz de *adunare* (transport dinspre dreapta spre stânga), dar şi în caz de *scădere* (văzut ca împrumut, dinspre stânga spre dreapta), exact aşa cum procedam în clasele primare.

Exemple 3.30

în binar					în zecimal				în hexazecimal				
		1 1 1					1				1 1		
010101	b	+			43	+			0C2B	h	+		
100111	b				27				1E36	h			
111100	b				70				2A61	h			

Exemple 3.31

în binar					în zecimal				în hexazecimal				
		*					*				* *		
100111	b	-			63	-			6B34	h	-		
010101	b				24				4DF1	h			
010010	b				39				1D43	h			

Exemple 3.32

în binar					în zecimal				în hexazecimal				
		. 1 2 . 2					5 9 10				5 15 15 16		
1001011	b	-			603	-			6 B 3 4	h	-		
0110101	b				294				4 D F 8	h			
0010110	b				309				1 D 3 C	h			

3.2.3. OPERAREA CU NUMERE DE FORMA PUTERI ALE LUI 2

Așa cum am văzut în secțiunea 2.4, numerele multiplu de (baza)^n se vor putea scrie în baza respectivă, sub forma: ***x*0...0**, unde ***x*** reprezintă ***orice combinație de cifre*** în acea bază, iar numărul de biți de 0 este egal cu **n**.

Cum adunăm un multiplu de 2 la un număr oarecare, direct în binar ?

Pentru a extrage regula de adunare din m în m, unde m se scrie ca o putere de-a lui 2, vom analiza două situaţii posibile:

a) Numărarea să înceapă de la 0;
b) Numărarea să înceapă de la un număr diferit de 0.

În Figura 3.2, pentru reprezentarea pe 3 biţi, dacă dorim să adunăm din 2 în 2, pornind de la 0, se observă că trecem din 000b în 010b (pentru situaţia a)), respectiv dacă dorim să începem de la 1, trecem din 001b în 011b (pentru situaţia b); în ambele cazuri, roata s-a învârtit cu 2 paşi spre dreapta, în sensul acelor).

Folosind reprezentarea pe 4 biţi, vom obţine: din 0000b, va rezulta 0010b, (pentru situaţia a)), respectiv trecem din 0001b în 0011b (pentru situaţia b); o altă posibilă situaţie pentru b) este să începem, de exemplu, de la 3: din 0011b trecem în 0101b.

În cadrul exemplelor de mai sus, s-au subliniat biții de pe poziția unde trebuie realizată operația, respectiv cei afectați de operație.

Exemple 3.33 Folosind roata numerelor pe 3 biţi, se vor analiza următoarele operaţii de adunare:

[(**0 + 2**)**+2**] **+2** = 6 operaţie echivalentă în binar: **000b + 2 = 010b**,

apoi **010b + 2 = 100b** şi în final: **100b + 2 = 110b**

[(**1 + 2**)**+2**] **+2** = 7 operaţie echivalentă în binar: **001b + 2 = 011b**,

apoi **011b + 2 = 101b** şi în final: **101b + 2 = 111b**

Analizând aceste situaţii, se poate extrage următoarea regulă:

Pentru a aduna un număr scris în binar cu un număr m putere a lui 2 (care se poate scrie $m=2^n$), se adună un bit de 1 pe poziţia n a acelui număr scris în binar (adunare cu transport); *astfel, nu vor putea fi afectaţi decât biţii de ordin superior, cei de ordin inferior rămânând identici cu cei ai numărului iniţial.*

Exemple 3.34 Să operăm de data aceasta folosind regula aritmetică:

Dacă vrem să adunăm la numărul 01**0**1b (care în zecimal este 5) valoarea $2=2^1$, atunci vom aduna cu transport un bit de 1 la bitul de **ordin 1**, obţinând 01**1**1b (care în zecimal este 7), caz în care nu a apărut transport înspre stânga.

Dacă vrem să adunăm la numărul 101**1**1b (care în zecimal este 23) valoarea $2=2^1$, atunci vom aduna cu transport un bit de 1 la bitul de **ordin 1**, obţinând 110**0**1b (care în zecimal este 25), caz în care a apărut transport înspre stânga.

Dacă vrem să adunăm la numărul 0**1**01b (care în zecimal este 5) valoarea $4=2^2$, atunci vom aduna cu transport un bit de 1 la bitul de **ordin 2**, obţinând 1**0**01b (care în zecimal este 9), caz în care a apărut transport înspre stânga.

5+	01**0**1b +	23+	101**1**1b +	5+	0**1**01b +
2	10b	2	10b	4	100b
7	01**1**1b	25	1*10***0**1b	9	1**0**01b

Din Tabelul 3.1, se poate observa că numărând din 2 în 2 în binar (vrem să analizăm numerele pare), obţinem: 000b, 010b, 100b, 110b (pentru 0, 2, 4, 6), etc. Aceste valori ale șirului numerelor pare se pot deduce ușor, dacă ținem cont de formula dată în secțiunea 2.4 și anume că *un număr multiplu de 2* se scrie în binar sub forma: ***x0b***, unde x poate fi orice combinaţie de biţi. Observați modul cum se modifică partea superioară a numerelor din șir datorită transportului care apare la adunare: **00**0b, **00**10b, **01**00b, **01**10b, **10**00b, **10**10b, **11**00b, **11**10b, etc.

Similar, se poate observa regula corespunzătoare numerelor multiplu de 4: 0000b, 0100b, 1000b, 1100b (pt 0, 4, 8, 12), etc. Aceste valori ale șirului numerelor din 4 în 4 se pot deduce ușor, dacă ținem cont de formula dată în secțiunea 2.4 și anume că *un număr multiplu de 4* se scrie în binar sub forma: ***x00b***, unde x poate fi orice combinaţie de biţi. Observați modul cum se modifică partea superioară a numerelor din șir datorită transportului care apare la adunare: **0**000b, **0**100b, **1**000b, **1**100b, **10**000b, **10**100b, **11**000b, **11**100b, etc.

Astfel, vom extrage următoarea regulă:

Un număr multiplu de 2^n *se va scrie* ***în binar*** *sub forma:* ***x00...0b,***
unde ***x*** *este* ***orice combinaţie de biţi,*** *iar* ***numărul de biţi de 0*** *(de după x) este* ***n.***

Exemplu 3.35

Numărul 10100b este multiplu de 2 (are b0=0), dar şi multiplu de 4 (b1b0=00).

Exemplu 3.36

Specificaţi toţi divizorii de forma 2^x ai numărului 28h.

Pentru a putea specifica divizorii de forma 2^x ai unui număr, este mult mai uşor ca acel număr să fie scris în binar şi să se observe numărul de biţi de 0 dinspre dreapta spre stânga. Astfel, 28h se scrie 00101000b şi se observă că are ca divizor pe 2 (b0=0), dar şi pe 2^2 (b1b0=00), respectiv şi pe 2^3 (b2b1b0=000).

Exemplu 3.37

Dacă primul element al unui sir de valori este 5 și urmatoarele valori se vor obtine in ordine crescatoare, prin adunarea unui 4, care sunt urmatoarele 6 elemente ale sirului ? La prima vedere, este o banală întrebare la care găseam răspunsul încă din clasele primare, însă vrem o metodă rapidă care să funcționeze în binar.

4. LUCRUL CU NUMERELE CU SEMN

În capitolul anterior, am văzut cum lucrăm cu numerele *fără semn*; de exemplu, am văzut cum reprezentăm un număr de forma 8. În acest capitol, vom extinde discuția și în ceea ce privește reprezentarea numerelor de forma +8 sau -8, deci ne vom ocupa de numerele *cu semn*.

4.1. CONVERSII DE NUMERE CU SEMN

Am văzut în secțiunea 3.1.2 că pentru a converti un număr din zecimal în binar, putem folosi 2 metode și anume: ***Metoda 1*** (**prin împărţiri succesive**), respectiv ***Metoda 2*** (**prin scăderea puterii maxime**). Exact la fel vom proceda și aici pentru numere pozitive, cu o mică excepție: trebuie neapărat să adăugăm un bit de 0 în fața reprezentării binare obținute, acest bit de 0 sugerând faptul că e un număr care are semnul + (de exemplu, discutăm despre +8 și nu despre 8).

Ca regulă generală, pentru a nu greși la conversia numărului din zecimal în binar, este mai mult decât indicată ***verificarea numărului de biți necesar scrierii corecte a unui număr, întotdeauna înainte de efectuarea operației efective de conversie din zecimal.*** În general, greșeala apare din cauza ignorării bitului de semn care trebuie adăugat la reprezentarea unui număr pozitiv ca număr cu semn.

Reamintesc din secțiunea 3.1. că folosind **n biți**,

dacă se consideră numerele ***fără semn***, gama numerelor este: $0 \div 2^n-1$, iar

dacă se consideră numerele ***cu semn***, gama numerelor este: $-2^{n-1} \div +2^{n-1}-1$,

unde cele 2^n ***numere reprezentabile pe n biți*** formează *inelul claselor de echivalență a resturilor modulo* 2^n.

Exemplul 4.1

Așa cum am văzut în capitolul anterior, reprezentarea numărului întreg **37** ca număr *fără semn*, scris în binar este:

$37 = 1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 1\,0\,0\,1\,0\,1_b$

Dacă este vorba despre numărul *cu semn* **+37**, atunci reprezentarea corectă se va obține prin adăugarea unui bit suplimentar de 0 ca bit de semn, astfel:

$+37 = 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 0\,1\,0\,0\,1\,0\,1_b$

iar pentru a se obține numărul **-37**, numărul în binar se va scrie[5]:

$-37 = -1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 1\,0\,1\,1\,0\,1\,1_b$

Care este MSb la numerele reprezentate în convenția cu semn ?

În general, la scrierea numerelor cu **semn (pozitiv sau negativ)**, plecând de la numărul scris fără semn (ca valoare pe n biți), am văzut că se folosește în plus un bit (sunt n+1 biți în loc de n biți), deci se mai adaugă un bit suplimentar pentru specificarea semnului. Bitul de semn se consideră a fi bitul c.m.s. (MSb) al reprezentării, având valoarea ***"0" dacă numărul este pozitiv*** sau ***"1" dacă numărul este negativ.*** Se mai observă din exemplele prezentate că nu seamănă neapărat ceilalți biți din reprezentarea numărului cu semn + și a celui cu semn – (din contră, sunt chiar foarte diferiți); regulile folosite sunt specificate în secțiunile următoare, după acomodarea cu folosirea corectă a gamei numerelor.

Să nu cădem în capcana de a spune că reprezentarea numerelor negative se realizează ca și la numerele positive, doar că bitul de semn e 1 în loc de 0. E total greșită o astfel de interpretare, deoarece în PC se folosește reprezentarea în C2. Dacă s-ar fi folosit reprezentarea în MS, ar fi fost adevărat, dar în sistemele de calcul regula de reprezentare a numerelor negative este C2: la tratarea numerelor negative (de exemplu pentru a obține reprezentarea binară a numărului -8), modul de lucru va fi diferit față de cel utilizat la obținerea corespondentului pozitiv +8.

4.1.1. PAȘI PENTRU CONVERSIA CORECTĂ A UNUI NUMĂR ÎN BINAR

Care sunt pașii pentru conversia corectă în binar a unui număr cu semn ?

Pentru a se evita ambiguitățile, la reprezentarea numerelor *cu semn* este obligatoriu să ținem cont de gama numerelor.

Astfel, la conversia unui număr *cu semn* din zecimal în binar,

I. prima dată se va determina numărul minim de biți necesar scrierii corecte a numărului (folosind gama numerelor) și abia apoi

II. se va trece la conversia numărului în noua bază.

Majoritatea greșelilor care pot apărea la conversie se datorează faptului că din grabă, se sare direct în pasul II și nu se mai ține cont de gama numerelor, numărul cu semn fiind astfel reprezentat eronat, pe un număr mai mic de biți.

De exemplu, dacă la +37 nu punem bitul MSb în 0 (și îl scriem doar 100101b*), la conversia pentru obținerea numărului -37 vom greși.*

Exemplul 4.2

Numărul *fără semn* 37 se încadrează în gama [0;63] ce folosește un număr de minim 6 biți: $[0;+2^n-1] \Rightarrow 2^n-1=63 \Rightarrow 2^n=64=2^6 \Rightarrow n=6 \Rightarrow 37 = 100101b$

În schimb, numărul *cu semn* +37 se încadrează corect în gama [-64,+63] ce folosește un număr de minim 7 biți:

$[-64;+63] = [-2^{n-1};+2^{n-1}-1] \Rightarrow 2^{n-1}=64=2^6 \Rightarrow n-1=6 \Rightarrow n=7 \Rightarrow +37 = 0100101b$

Acest bit MSb în **0** va fi esențial la obținerea numărului -37.

Exemplul 4.3

Numărul *fără semn* 4321 se încadrează în gama [0;8191] ce folosește un număr de minim 13 biți:

$[0; 8192] = [0;+2^n-1] \Rightarrow 2^n-1=8191 \Rightarrow 2^n=8192=2^{13} \Rightarrow n=13$

$\Rightarrow 4321 = 1\ 0000\ 1110\ 0001b = 10E1h$

În schimb, numărul *cu semn* +4321 se încadrează corect în gama [-8192,+8191] ce folosește un număr de minim 14 biți:

$[-8192;+8191] = [-2^{n-1};+2^{n-1}-1] \Rightarrow 2^{n-1}=8192=2^{13} \Rightarrow n-1=13 \Rightarrow n=14$

$\Rightarrow +4321 = 01\ 0000\ 1110\ 0001b = 10E1h$

Subliniez că ***numărul cu semn + 4321 se va reprezenta corect pe n=14 biți și nu pe doar 13 biți;*** deci +4321 se va scrie corect **0**1 0000 1110 0001b și nu doar ca 1 0000 1110 0001b, cum s-a scris 4321 ca număr fără semn.

[5] Pentru conversia numărului negativ s-a folosit codul complementar (convenția complement față de 2)

Care e diferenţa la reprezentarea numerelor pozitive în convenţia cu semn versus cea fără semn ?

Ca o concluzie, la exprimarea ***valorilor pozitive*** în ***convenţia cu semn***, se observă că reprezentarea *e aproape identică* cu cea corespunzătoare în ***convenţia fără semn***, dar ***se mai adaugă un bit (MSb) de 0***.

Este extrem de important să ţinem cont de acest aspect, în special atunci când va trebui să obţinem corespondentul negativ al acelui număr. Dacă nu adăugăm acest bit de 0 în fața reprezentării obținute la numerele fără semn, vom greși la conversia numerelor negative. Am subliniat de mai multe ori acest lucru, deoarece am constatat că mulți greșesc aici.

4.1.2. CONVENŢII PENTRU OBŢINEREA NUMERELOR NEGATIVE (CU SEMN)

Care sunt convenţiile ce se pot folosi la reprezentarea unui număr negativ în convenţia cu semn ?

Pentru a reprezenta ***numerele cu semn negativ*** se pot utiliza 3 convenţii:

1. **Modul şi semn (MS)**
2. **Complement faţă de 1 (C1)**
3. **Complement faţă de 2 (C2)**

Numerele pozitive se reprezintă simplu, ca şi cum ar fi numere fără semn, dar se mai adaugă un bit de zero în extremitatea stângă (ca bit de semn, deci MSb) pentru a specifica faptul că acesta a fost un număr cu semnul + în zecimal; în schimb, la reprezentarea numerelor negative se va proceda în unul dintre modurile:

1. Modul şi semn (MS, numit şi "**cod direct**"):

Reprezentarea foloseşte *bit de semn* urmat de *modul* (numit și *valoare absolută*). Cea mai mare şi cea mai mică valoare reprezentată pe octet sunt +127=**0**1111111b şi respectiv -127=**1**1111111b. Se reprezintă valoarea absolută a numărului pe n-1 biţi în binar (n fiind numărul minim de biţi necesar scrierii corecte a numărului) şi apoi se adaugă un bit de semn pe poziţia bitului MSb care va fi 0 (dacă numărul e pozitiv) sau 1 (dacă numărul e negativ).

Exemplul 4.4

Numărul +37 = **0**10 0101**b**, de unde -37 va fi: -37 = **1**10 0101**b** în convenţia MS

Codificarea în MS pentru numere negative se realizează foarte simplu, modificând doar bitul de semn. Deşi în această convenţie operaţiile de înmulţire şi împărţire se realizează simplu, la operaţiile de adunare şi scădere unitatea aritmetică ar trebui să ţină cont de semnul operanzilor, de unde ar rezulta circuite hardware mai complexe. Această metodă are dezavantaje şi în organizarea logică a unităţii centrale, deoarece există 2 reprezentări pentru 0: reprezentările 1000 0000b şi 0000 0000b care sunt echivalente (plus şi minus zero), această reprezentare nefiind deci considerată eficientă de către inginerii proiectanți. S-a renunțat la ea în PC.

2. Complement faţă de 1 (C1) (numit şi "**cod invers**"):

se obține prin complementarea fiecărui bit din reprezentarea numărului corespunzător pozitiv.

Exemplul 4.5

numărul +37 = 010 0101b, de unde rezultă -37 = 101 1010b în convenţia C1

Reprezentarea în C1 este uşor de realizat de către partea hardware, dar algoritmii de înmulţire şi împărţire sunt mai complecşi decât la reprezentarea MS. În plus, ca şi în cazul convenţiei MS, există 2 reprezentări pentru 0; şi această reprezentare este considerată ineficientă, s-a renunțat și la aceasta în PC.

3. Complement faţă de 2 (C2) (numit şi "***cod complementar***"):

Regula de a obţine reprezentarea unui număr negativ în reprezentarea C2 este: la reprezentarea obţinută în convenţia C1 a numărului corespunzător pozitiv, se mai adaugă un bit de 1, prin adunare cu transport.

Exemplul 4.6

numărul -37 = 101 1010b (scris în C1) + **1** = 101 1011b în convenţia C2

Care este semnificaţia pe care o poate lua un număr scris pe n biţi în reprezentarea fără semn versus cea cu semn (în oricare din convenţiile MS, C1 și C2) ?

Este important de subliniat că un număr scris pe n biţi poate avea semnificaţie diferită, în funcţie de convenţia folosită pentru reprezentare; exemple pentru numere scrise pe doar 4 biţi sunt prezentate în Tabelul 4.1.

Care dintre convenţiile MS, C1, C2 s-a ales pentru reprezentarea numerelor negative în PC ?

Deşi algoritmul pentru înmulţire şi împărţire pentru ***operanzi reprezentaţi în C2*** este mai complex decât cel corespunzător codului MS, aceasta este convenţia care s-a utilizat la procesoarele actuale, datorită următoarelor avantaje:

- la reprezentarea numerelor în C2 se implementează doar operaţia de adunare, deoarece scăderea unui număr din alt număr e echivalentă matematic cu adunarea complementului faţă de 2 a scăzătorului la descăzut; astfel, circuitele electronice pentru adunare şi scădere nu trebuie să examineze semnul operanzilor, vor efectua întotdeauna doar adunări;

- codificarea în C2 printr-un circuit electronic este uşor de realizat;
- convenţia C2 are o singură reprezentare pentru zero (00...0, deci oferă un cod în plus faţă de convenţiile MS şi C1);
- un întreg în reprezentarea C2 poate fi uşor extins la un format mai mare (pe un număr mai mare de biţi) fără a i se altera valoarea.

Tabelul 4.1. Reprezentarea unui număr pe 4 biţi în diferite convenţii de reprezentare

Numărul în hexazecimal baza 16	Reprezentare în binar baza 2	Semnificaţia numărului în zecimal în convenţie			
		fără semn	cu semn		
			MS	C1	C2
Numere pozitive în convenţia cu semn					
0h	0000b	0	0	0	0
1h	0001b	1	1	1	1
2h	0010b	2	2	2	2
3h	0011b	3	3	3	3
4h	0100b	4	4	4	4
5h	0101b	5	5	5	5
6h	0110b	6	6	6	6
7h	0111b	7	7	7	7
Numere negative în convenţia cu semn					
8h	1000b	8	-0	-7	-8
9h	1001b	9	-1	-6	-7
Ah	1010b	10	-2	-5	-6
Bh	1011b	11	-3	-4	-5
Ch	1100b	12	-4	-3	-4
Dh	1101b	13	-5	-2	-3
Eh	1110b	14	-6	-1	-2
Fh	1111b	15	-7	-0	-1

Exemplul 4.7 Numărul 8 din zecimal, scris doar pe 4 biţi, poate fi scris ca 8 în reprezentarea fără semn, -0 în MS, -7 în C1 şi -8 în C2.

Astăzi, toate calculatoarele folosesc reprezentarea numerelor în convenţia complement faţă de 2, celelalte două forme (MS şi C1) nemaifiind utilizate. Astfel, în continuarea materialului, nu voi mai preciza acest lucru, se va considera (dacă nu se specifică altfel) că implicit, dacă e vorba de un număr negativ, acesta a fost reprezentat în C2.

4.1.3. REGULI ALTERNATIVE DE OBŢINERE A UNUI NUMĂR NEGATIV ÎN C2

Pentru conversia unui număr negativ în C2, pe lângă ***definiţie*** (C2=C1+1) se mai pot utiliza încă ***3 reguli alternative*** [ref2]:

Regula alternativă 1. prin parcurgere dinspre dreapta spre stânga şi complementarea anumitor biţi;

Regula alternativă 2. prin scădere direct în binar pe un anumit număr de cifre binare;

Regula alternativă 3. prin scădere direct în hexazecimal pe un anumit număr de cifre hexazecimale.

Toate cele 3 metode sau reguli alternative **pleacă de la corespondentul pozitiv** al numărului ce se doreşte a fi scris.

În plus, la reprezentarea numerelor, se va ţine cont de gama numerelor cu semn.

Regula alternativă 1 (RA1): Pentru a reprezenta -x în binar, se pleacă de la +x scris în binar şi se parcurge reprezentarea dinspre dreapta spre stânga, copiind toţi biţii, până la primul bit de 1 inclusiv; restul biţilor se inversează.

Exemplul 4.8 Vrem să-l reprezentăm pe -4 în binar; astfel, se pleacă de la reprezentarea lui +4 = 0100b dinspre dreapta spre stânga, se caută primul bit de 1 şi până la el (inclusiv) se copiază; restul biţilor se complementează (sau inversează). Astfel, obţinem: 1100b ca reprezentare a lui -4 în binar, pe 4 biţi.

Regula alternativă 2 (RA2): Pentru a reprezenta pe -x în binar pe n biţi, se pleacă de la +x scris în binar pe n biţi şi se scade acesta (operaţia se realiz. deci în binar) dintr-un număr care are n cifre binare în 0, precedate de un 1.

Exemplul 4.9 Vrem să-l reprezentăm pe -4 în binar pe 4 biţi; astfel, se pleacă de la reprezentarea lui +4 = 0100b şi se scade acest număr din numărul 10000b (se observă numărul format dintr-un bit de 1 urmat de 4 biţi de 0).

10000b-
0100b

- reprezentarea lui -4 va fi deci 1100b şi nu 01100b, pentru că aşa am precizat de la început, că dorim un număr reprezentat pe 4 biţi (nu-l vom considera deci pe 5 biţi, pentru că ar fi eronat).

0**1100b**

Regula alternativă 3 (RA3): Regula este identică cu cea anterioară, dar este în hexazecimal şi nu în binar. Pentru a reprezenta numărul -x în hexazecimal pe n cifre hexa, se pleacă de la numărul +x scris în hexazecimal pe n cifre hexa şi se scade acesta (operaţia se desfăşoară deci în hexazecimal) dintr-un număr care are n cifre hexazecimale în 0 precedate de 1.

Exemplul 4.10 Vrem să-l reprezentăm pe -4 în hexazecimal pe o cifră hexazecimală; astfel, se pleacă de la reprezentarea lui +4:

10h-
4h

+4=4h şi se scade aceasta din numărul 10h (se observă că este format din cifra hexazecimală 1 urmată de o cifră hexazecimală 0).

0**Ch**

- reprezentarea lui -4 va fi deci Ch=1100b şi nu 0Ch=01100b, pentru că aşa am precizat de la început, că dorim un număr reprezentat pe o singură cifră hexazecimală.

Regula alternativă 2 poate fi văzută ca ***regula alternativă 4 (cu scădere în zecimal)*** şi aceasta reprezintă o ***modalitate rapidă de conversie în C2*** des utilizată: ***dacă se dorește obținerea unui nr -x pe n biți (determinat cu gama numerelor) se pleacă de la reprezentarea lui +x şi se scade (în zecimal) acest număr din numărul 2^n . Rezultatul obținut în convenția fără semn (ca număr fără semn) se scrie în binar şi reprezintă numărul -x (negativ) în reprezentarea cu semn.***

Exemplul 4.11

Pentru a obține numărul **-4** scris pe 4 biți, îl vom scădea pe **4** din 2^4=16 şi vom avea: **16 -4** = **12** şi scriind apoi numărul fără semn în binar, obținem: 12 = **1100**b => **-4** = **1100**b

Exemplul 4.12 Se va obține numărul **-37** prin mai multe modalități: prin definiție şi prin cele 4 reguli alternative.

Definiție (C2=C1+1): se pleacă de la +37 care se reprezintă în binar (dar nu înainte de a încadra pe -37 în gama potrivită ([-64;+63]) şi deci de a stabili nr minim de biți necesar în reprezentarea lui (7 biți)): +37 = 0100101b. Apoi, aplicând complement față de 1, din +37 se obține: 1011010b la care se mai adună 1 şi rezultă în final:

-37=1011011b.

RA1 se copiază dinspre dreapta spre stânga toți biții până la primul bit de 1 inclusiv, apoi se inversează toți: +37 = 0100101b, de unde vom obține: ***-37=1011011b***.

+37 = 0100101b
=>-37=1011011b

RA2: se va scădea +37=0100101b scris cu 7 biți în binar din 10000000b:
de unde vom obține numărul -37 scris tot pe 7 biți: ***-37=1011011b=5Bh***.

10000000b –
0100101b
01011011b

RA3: se va scădea +37=0100101b scris în hexazecimal cu 2 cifre hexazecimale, deci îl vom scrie pe 8 biți: +37=**0**0100101b=25h (am adăugat un bit de 1) din 100h:
de unde vom obține numărul -37 scris tot pe 2 cifre hexazecimale, deci cu 8 biți: ***-37=DBh=11011011b***.

100h –
25h
0DBh

RA4: se va scădea +37 din numărul zecimal 2^7=128, deci vom avea:
Iar rezultatul obținut, numărul 91, se va scrie în binar: 91=1011011b care va fi echivalentul pe 7 biți al numărului -37, deci vom avea:
-37=1011011b.

128 –
37
91

Exemplul 4.13

Vom repeta raționamentul de mai sus pentru obținerea numărului **-4321**.

Definiție (C2=C1+1): se pleacă de la +4321 care se reprezintă în binar (dar nu înainte de a încadra pe -4321 în gama potrivită ([-8192;+8191]) şi deci de a stabili nr minim de biți necesar în reprezentarea lui (14 biți)): +4321=10E1h= 01000011100001b. Apoi, aplicând complement față de 1, din +4321 se obține: 10111100011110b la care se mai adună un 1 şi rezultă în final: **-4321=10111100011111b=10 1111 0001 1111b = 2F1Fh** (unde aşa cum vom vedea în secțiunea următoare, prima cifră hexazecimală nu a fost extinsă corect, deoarece s-au considerat 2 biți de 0 ca biții $b_{15}b_{14}$).

RA1 se copiază dinspre dreapta spre stânga toți biții până la primul bit de 1 inclusiv, apoi se inversează toți: +4321 = 01000011100001b, de unde vom obține: ***-4321=10111100011111b***.

+4321 = 01000011100001b
=> -4321=10111100011111b

RA2: se va scădea +4321=01000011100001b scris cu 14 biți în binar din 100000000000000b (numărul are 14 biți de 0, precedați de 1):
de unde vom obține numărul -4321 scris tot pe 14 biți: ***-4321=10111100011111b=2F1F***.

0 1 1 1 1 1 1 1 1 1 1 1 1 1 2
100000000000000b –
01000011100001b
010111100011111b

RA3: se va scădea +4321=01000011100001b scris în hexazecimal cu 4 cifre hexazecimale, deci îl vom scrie pe 16 biți: +4321=**00**01000011100001b=10E1h (s-au adăugat 2 biți de 0) din 10000h şi vom obține numărul -4321 scris tot pe 4 cifre hexazecimale, deci cu 16 biți:
-4321=EF1Fh=1110111100011111b.

0 15 15 15 16
10000h –
10E1h
0EF1Fh

RA4: se va scădea +4321 din numărul zecimal $2^{14}=2^4*2^{10}=16*1024=16384$, deci vom obține numărul 12063, care se va scrie în binar: 12063=1011100011111b (unde pentru transformare am aplicat regula scăderii puterii maxime: 12063-8192=3871, apoi 3871-2048=1823, apoi 1823-1024=799, apoi 799-512=287, apoi 287-256=31, apoi 31-16=15, apoi 15-8=7, apoi 7-4=3, apoi 3-2=1 şi în final 1-1=0) care va fi echivalentul pe 14 biți al numărului -4321, deci vom avea: ***-4321=10111100011111b***

16384–
4321
12063

Concluzionând, am obținut următoarele valori pt numerele **-37** şi respectiv **-4321**:

-37= 1011011b=11011011b,
scrise în hexazecimal 5Bh,
respectiv DBh.

-4321=10111100011111b
=1110111100011111b,
scrise în hexazecimal ca 2F1Fh,
respectiv EF1Fh.

Observăm că RA3 furnizează rezultate diferite atât în binar cât şi în hexazecimal versus definiţia sau celelalte 3 reguli alternative şi deci este normal să ne întrebăm: ***Care valori sunt corecte şi care nu sunt corecte? De ce s-au obţinut rezultate diferite? Care este oare explicaţia?***

Astfel, întrebarea care se pune aici este: dacă numărul -37 se scrie corect (după cum a rezultat din definiţie) ca -37= ***1011011b***, cum se va scrie acest număr în hexazecimal ? Se poate scrie doar cu 7 biţi în hexazecimal? Evident, răspunsul la ultima întrebare este negativ. Şi atunci, ***cum se scrie numărul -37 în hexazecimal***, ca ***5Bh*** sau ca ***DBh***? Dacă privim cele 2 valori în binar, vom observa că diferă doar un singur bit: 5Bh=01011011h, respectiv DBh=11011011b.

Similar, pentru numărul -4321, am observat că din definiţie a rezultat ***10111100011111b***, care e scris cu 14 biţi; pentru scrierea în hexazecimal, ne lipsesc deci 2 biţi: cum să scriem aceşti 2 biţi în faţa numărului, ca 00 sau ca 11? Deci ***-4321 se va scrie în hexazecimal*** ca ***2F1Fh*** sau ca ***EF1Fh*** ?

Deci practic, ne întrebăm: ***Cum se extinde corect un număr negativ, cu bit de 0 sau cu bit de 1? Mai mult, putem extinde cu orice număr de biţi? Cum vom verifica noul număr că este obţinut corect ?***

4.2. EXTENSIA ȘI CONTRACTAREA NUMERELOR

4.2.1. EXTENSIA ŞI CONTRACTAREA NUMERELOR CU SEMN ŞI FĂRĂ SEMN

Pentru a realiza operaţia de extensie a unui număr, este esenţială cunoaşterea convenţiei în care se consideră acel număr: este ***un număr considerat fără semn*** sau este ***un număr considerat cu semn***? Aceasta, deoarece:

- ***extensia unui număr fără semn*** se realizează ***întotdeauna cu bit de 0***, în timp ce
- ***extensia unui număr cu semn***, se realizează ***cu bitul de semn***.

Exemplu: Extinderea numărului ***fără semn*** 80h pe 16 biţi va da 0080h, iar pe 32 biţi va da 0000 0080h.
Indiferent că am scris valoarea pe pe 8 biţi, 16 biţi sau pe 32 biţi, în zecimal ea trebuie să rămână aceeaşi (ca nr fără semn): 128.

Exemple: dacă se consideră numere ***cu semn:***
a) Valoarea 80h, ca nr cu semn, se extinde ca FF80h pe 16 biţi sau FFFF FF80h pe 32 biţi; indiferent că am scris valoarea pe 8 biţi, pe 16 biţi sau pe 32 biţi, în zecimal ea trebuie să fie aceeaşi (ca nr cu semn): -128.

b) Numărul 40h, scris pe 8 biţi, se extinde ca 0040h pe 16 biţi sau 0000 0040h pe 32 biţi;
Similar, indiferent că am scris valoarea pe 8 biţi, pe 16 biţi sau pe 32 biţi, în zecimal ea trebuie să fie aceeaşi (ca nr cu semn): +64.
c) -37= 1011011b se extinde corect ca 11011011b şi va fi DBh
d) -4321=10111100011111b, se extinde corect ca 1110111100011111b şi deci va fi: EF1Fh.

Pentru ***contractarea unui număr scris ca număr fără semn***, deoarece la extensie s-au adăugat doar biţi de 0 (întotdeauna), este evident că nu vom putea elimina (tăia) biţi de 1 din reprezentarea numărului iniţial, deoarece aceştia nu au fost adăugaţi (la o eventuală extensie).

Exemple:
a) FF80h nu poate fi contractat; FF40h nu poate fi contractat;
b) 0020h poate fi contractat la 20h, sau chiar mai mult, e posibil inclusiv la un număr de doar 6 biţi[6];

Pe de altă parte, la ***contractarea unui număr scris ca număr cu semn***, trebuie să ţinem cont de semnul lui: dacă este un număr negativ (deci care începe cu bit de 1), e posibil să aibă mai mulţi astfel de biţi de 1 (biţi identici cu bitul de semn) şi toţi aceşti biţi ar fi putut rezulta dintr-o operaţie de extensie. Similar, dacă numărul este pozitiv şi după bitul de semn 0, mai urmează şi alţi biţi de 0. Astfel, se vor putea contracta aceşti biţi cu respectarea condiţiei ca măcar 1 bit să rămână.

Exemple:
a) FF80h poate fi contractat cu semn la 80h: FF80h=1111 1111 1000 0000b = ~~1111 1111~~ 1000 0000b=80h;
b) FF40h nu poate fi contractat cu semn la doar 8 biţi, pentru că scrierea corectă se realizează pe un număr de minim 9 biţi[7]: FF40h=1111 1111 0100 0000b = ~~1111 111~~1 0100 0000b = 1 0100 0000b
c) 0040h poate fi contractat cu semn la 40h, deci la un număr de doar 8 biţi: 0040h=0000 0000 0100 0000b = ~~0000 0000~~ 0100 0000b.
d) 0020h poate fi contractat la 20h, sau chiar mai mult, e posibil inclusiv la un număr de doar 7 biţi[8];

Se observă că sunt posibile ***4 situaţii diferite***: *extensia* sau *contractarea* unui *nr fără semn* / *cu semn*.

[6] Trebuie ţinut cont de faptul că în arhitectura x86 nu există registru de 6 biţi
[7] Trebuie ţinut cont de faptul că în arhitectura x86 nu există registru de 9 biţi
[8] Trebuie ţinut cont de faptul că în arhitectura x86 nu există registru de 7 biţi

Dacă se doreşte **extensia unui număr fără semn**, atunci extensia se realizează ***întotdeauna folosind valoarea 0***. Operaţia de extensie fără semn a unui număr poate fi realizată folosind procesorul 8086 foarte simplu, întrucât nu implică decât adăugarea de zerouri în faţa numărului; se poate folosi instrucţiunea MOV, pentru a muta zerouri în partea superioară a operandului destinaţie. De exemplu, dacă se va folosi accumulatorul AL ca având stocată o valoare ce se dorește a fi extinsă fără semn, registrul AH va trebui încărcat cu 0 (putem folosi instrucţ. *mov AH,0*) și atunci se va putea considera că registrul AX conţine numărul iniţial, extins de la 8 biţi la 16 biţi.	**Extensia cu semn a unui număr** este esenţială atunci când se operează două valori *cu semn,* dar *de dimensiuni diferite*. Operaţiile de extensie cu semn a unui număr pot fi realizate folosind procesorul 8086 prin instrucţiunile CBW, CWD care extind numărul de la byte la word, respectiv de la word la doubleword.
Contractarea numerelor cu semn: Contractarea numerelor cu semn este posibilă pentru acele numere care au un număr de mai mulţi biţi pornind de la c.m.s. bit de aceeași valoare (ori toţi 0, ori toţi 1).	**Contractarea numerelor fără semn:** Această operaţie este foarte sensibilă întrucât în interpretarea numerelor fără semn, orice bit de 1 reprezintă *o cantitate* ce nu se poate neglija sau elimina pur și simplu.

Exemplu: Să presupunem că în urma unei operaţii de adunare a 2 numere considerate ***cu semn***, scrise pe câte 8 biți fiecare, s-a obţinut suma +160 care se scrie corect pe 9 biți: **0** 1010 0000=0A0h. Cum în procesorul 8086 nu există registru (sau la mod general operand) de 9 biţi, această valoare va trebui extinsă pe 16 biţi, deci se va scrie 0000 0000 1010 0000b=**00**A0h (s-a folosit extinderea cu semn).

Exemplu: Să presupunem că în urma unei operaţii de adunare a 2 numere considerate ***fără semn***, scrise pe câte 8 biți fiecare, s-a obţinut suma 336, valoare care se scrie corect pe minim 9 biți: 336=**1** 0101 0000b=150h, dar cum procesorul nu are registru de 9 biţi, valoarea va trebui extinsă (prin extensie fără semn de la 9 la 16 biţi); pe 16 biţi vom avea: 336=0000 0001 0101 0000b=**01**50h (s-a folosit extinderea fără semn).

Exemplu: Să presupunem că în urma unei operaţii de adunare a 2 numere considerate ***cu semn***, scrise pe câte 8 biți fiecare, s-a obţinut suma -176 care se scrie corect pe 9 biți: -176=**1** 0101 0000=150h (! valoarea nu e corect scrisă în hexazecimal pe 12 biţi, ci doar pe 9 biţi). Cum în procesorul 8086 nu există registru de 9 biţi, această valoare va trebui extinsă (prin extensie cu semn) pe 16 biţi, la -176=1111 1111 0101 0000b=**FF**50h.

Exemple:

1 = 0000 0001b = 01h (scris pe 8 biţi) = 0001h (scris pe 16 biţi) = 0000 0001h (scris pe 32 biţi)
-1 = 1111 1111b = FFh (scris pe 8 biţi) = FFFFh (scris pe 16 biţi) = FFFF FFFFh (scris pe 32 biţi)
2 = 0000 0010b = 02h (scris pe 8 biţi) = 0002h (scris pe 16 biţi) = 0000 0002h (scris pe 32 biţi)
-2 = 1111 1110b = FEh (scris pe 8 biţi) = FFFEh (scris pe 16 biţi) = FFFF FFFEh (scris pe 32 biţi)
4 = 0000 0100b = 04h (scris pe 8 biţi) = 0004h (scris pe 16 biţi) = 0000 0004h (scris pe 32 biţi)
-4 = 1111 1100b = FCh (scris pe 8 biţi) = FFFCh (scris pe 16 biţi) = FFFF FFFCh (scris pe 32 biţi)
8 = 0000 0100b = 08h (scris pe 8 biţi) = 0008h (scris pe 16 biţi) = 0000 0008h (scris pe 32 biţi)
-8 = 1111 1000b = F8h (scris pe 8 biţi) = FFF8h (scris pe 16 biţi) = FFFF FFF8h (scris pe 32 biţi)

255 = 1111 1111b = FFh (scris pe 8 biţi) = 00FFh (scris pe 16 biţi) = 0000 00FFh (scris pe 32 biţi)
-128 = 1000 0000b = 80h (scris pe 8 biţi) = FF80h (scris pe 16 biţi) = FFFF FF80h (scris pe 32 biţi)
+128 = 0 1000 0000b = 080h (scris pe 9 biţi) = 0080h (scris pe 16 biţi) = 0000 0080h (scris pe 32 biţi)
+127 = 0111 1111b = 7Fh (scris pe 8 biţi) = 007Fh (scris pe 16 biţi) = 0000 007Fh (scris pe 32 biţi)
-512 = 10 0000 0000b = 200h (scris pe 10 biţi) = E00h (scris pe 12 biţi) = FE00h (scris pe 16 biţi)
+ 512 = 010 0000 0000b = 200h (scris pe 11 biţi) = 200h (scris pe 12 biţi) = 0200h (scris pe 16 biţi)

4.2.2. VERIFICAREA CORECTITUDINII REALIZĂRII EXTENSIEI SAU CONTRACTĂRII

Care este regula pentru verificarea faptului că o extensie sau o contractare s-a realizat corect?

Pentru verificarea corectitudinii unei extinderi sau contractări, putem aplica
relaţia (3.1) dacă nr e considerat ***fără semn***, respectiv
relaţia (3.2) dacă nr este ***cu semn*** adaptată la baza 2.

Regula aplicată la verificare pentru numere fără semn este:

- toţi termenii reprezintă cantităţi, acestea fiind ignorate doar dacă cifra respectivă este 0

Valoarea numărului ***N*** în baza 10 (N fiind scris în baza **2**) se va putea calcula după relaţia:

$$N(10) = a_n 2^n + a_{n-1} 2^{n-1} + a_{n-2} 2^{n-2} + \ldots + a_1 2^1 + a_0 2^0. \qquad \text{(relaţia 3.1)}$$

Regula aplicată la verificare pentru numere cu semn este:

- primul termen *este nul* doar pentru ***numere pozitive*** deoarece cifra binară este 0, dar
- pt ***numere negative*** primul termen trebuie considerat cu semn – pentru că acesta e o *cantitate nenulă* întotdeauna.

Valoarea numărului ***N*** în baza 10 (N fiind scris în baza ***2***) se va putea calcula după relaţia:

$$N(10) = -a_n 2^n + a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \ldots + a_1 2^1 + a_0 2^0. \qquad \text{(relaţia 3.2)}$$

În cadrul relaţiilor de mai sus, a_n, a_{n-1}, ..., a_0 reprezintă cifrele binare (în baza ***q***=***2***) ale numărului N.

Ca o regulă în ceea ce privește reprezentarea numerelor, trebuie subliniat că: pentru transformarea unui număr din binar în zecimal, ***primul termen din descompunerea binară este considerat cu semnul + sau – după cum numărul este pozitiv sau negativ, dar toţi ceilalţi termeni sunt consideraţi pozitivi*** (această regulă se aplică indiferent dacă numărul este pozitiv sau negativ).

Exemplu: Aplicând relaţia 3.2, valoarea **1 1 0 1** $_b$ se scrie: $-1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$.
Această regulă poate fi folosită în special atunci când dorim să verificăm dacă o extensie s-a realizat corect sau nu.

Exemplu: Numărul +37 scris pe 7 biţi este 010 0101b şi poate fi extins pe 8 biţi astfel: **00100101**b; transformat înapoi în zecimal (pentru verificare), acesta se va scrie: $0+0+2^5+0+0+2^2+0+2^1$=+37, deci este corect chiar şi după extensie.

Exemplu: Numărul -37 în C2 a fost extins de la 7 biţi la 8 biţi: -37 = **1**1011011b; transformat înapoi în zecimal, va fi:
$-2^7+2^6+0+2^4+2^3+0+2^1+2^0 = -128+64+16+8+2+1$=-37 deci este corect.

Cum se poate realiza conversia unui număr din pozitiv în negativ sau invers pe CPU?

Pentru realizarea conversiei, procesorul are implementată o instrucţiune care va realiza în mod automat conversia[9]; de exemplu, dacă vom avea valoarea +37, o vom putea obţine simplu pe -37 sau invers: din -37 vom putea obţine +37. Extensia se va realiza cu bitul de semn; astfel, în faţa MSb se vor adăuga biţi de 0 sau de 1 corespunzător MSb, până la atingerea dimensiunii dorite.

Ca regulă, vom reţine:

Un număr (considerat a fi cu semn) scris pe n biţi în C2 se poate extinde la orice număr dorit de biţi, mai mare decât n, cu condiţia extinderii corespunzătoare a semnului, deci cu adăugarea de biţi identici cu MSb în stânga.

În schimb, numerele considerate a fi fără semn, se vor extinde întotdeauna cu biţi de 0.

3.6. INTERPRETAREA DUALĂ A VALORILOR NUMERICE

Cele 2 convenţii de reprezentare a numerelor (*cu semn* şi *fără semn*) ilustrează faptul că datele din memoria PC-ului se pot interpreta diferit. Unitatea centrală de prelucrare (sau simplu spus, procesorul) nu ştie semnificaţia valorii binare (reprezentate pe octet, cuvânt, etc), deci nu ştie dacă să o interpreteze ca un număr fără semn sau ca un număr cu semn, decât dacă se precizează în mod explicit acest lucru (prin una din instrucţiunile care ajung la procesor).

Această interpretare dual poate fi observată în Figura 3.3, unde s-au reprezentat numerele pe 1 bit, pe 2 biţi şi pe 3 biţi în ambele convenţii; de exemplu, **valoarea 1 din binar** poate fi interpretată ca fiind:

- ***fără semn*** şi atunci ne referim la numărul **1** –> reprezentată în interiorul roţii sau
- ***cu semn*** şi atunci ne referim la numărul **-1** –> reprezentată în exteriorul roţii.

Similar, o combinaţie de biţi, de exemplu **101b** poate fi văzută ***ca numărul fără semn 5*** sau ***ca numărul cu semn -3***.

Reprezentarea numerelor fără semn şi cu semn pe roata numerelor

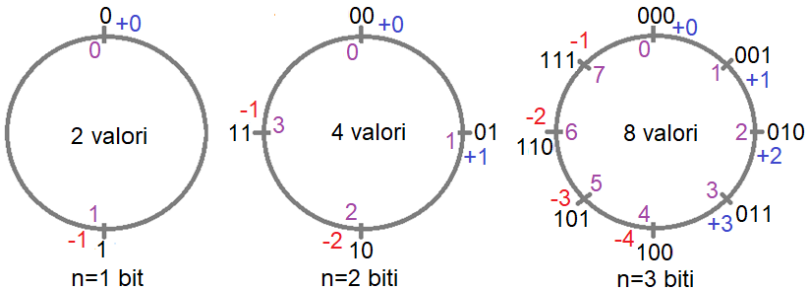

Figura 3.3 Reprezentarea valorilor binare pe roata numerelor pt n=1 bit, n=2 biţi şi n=3 biţi şi interpretarea acestora *ca numere fără semn* (în interiorul roţii) sau *ca numere cu semn* (în exteriorul roţii)

Aşa cum se observă şi din Figura 3.4, valorile din prima jumătate a roţii nu diferă prea mult în cele 2 convenţii, însă valorile din a II-a jumătate pot ridica probleme atunci când nu se ţine cont de reprezentarea folosită.

[9] Este vorba de instrucţiunea NEG (realizează operaţia C2); există şi instrucţiunea NOT (care realizează operaţia C1)

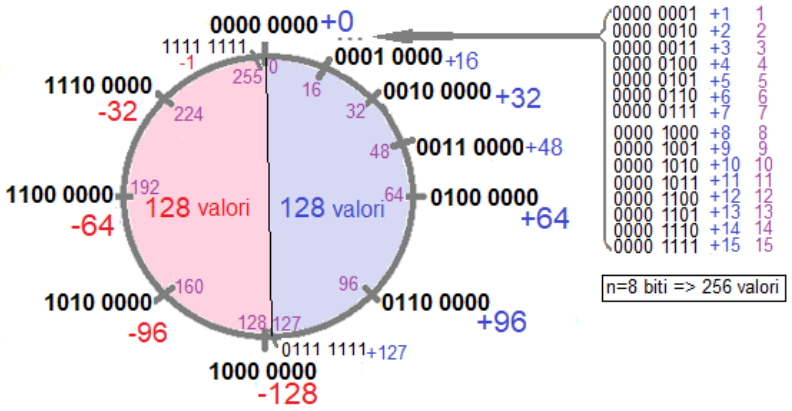

Figura 3.4. Reprezentarea valorilor binare pe roata numerelor pt n=8 biţi şi interpretarea acestora *ca numere fără semn* (în interiorul roţii) sau *ca numere cu semn* (în exteriorul roţii)

O altă posibilă problemă poate să apară la operarea cu aceste valori, de exemplu la adunarea a 2 numere pozitive (din prima jumătate a roţii) să rezulte un nr din a II-a jumătate (se adună 2 numere pozitive şi suma este un număr negativ).

Sir de octeţi, numere cu semn : 2,-4, 8, 5, -16, -1, -8, -10

02h	0FCh	08h	05h	F0h	FFh	F8h	F6h
100h	*101h*	*102h*	*103h*	*104h*	*105h*	*106h*	*107h*

Sir de octeţi, numere fără semn : 2,252, 8, 5, 240, 255, 248, 246

02h	0FCh	08h	05h	F0h	FFh	F8h	F6h
100h	*101h*	*102h*	*103h*	*104h*	*105h*	*106h*	*107h*

Ca reprezentare în memorie, sunt identice ! dar ca interpretare a valorilor, sunt diferite