Arhitectura calculatoarelor

Laboratorul 1.

1.1. Reprezentări numerice în virgulă fixă.

Reprezentările numerice folosite în calculatoare au un caracter *poziţional*. Prin aceasta înţelegem faptul că poziţia unei *cifre* în reprezentarea formală a *numărului* semnifică ponderea acelei cifre în raport cu o *bază de numeraţie*. Bazele de numeraţie folosite frecvent sunt:

- sistemul zecimal, cu baza de numeraţie 10;
- sistemul binar, cu baza de numeraţie 2, uşor de implementat în calculatoare dar dificil de folosit de către om;
- sistemul hexazecimal, folosit pentru reprezentarea simbolică a numerelor binare într-o formă acceptabilă pentru om.

Prin termenul de *virgulă fixă* înţelegem că unitatea aritmetică a unui calculator este proiectată în aşa fel încât să convertească coduri binare în numere şi să opereze cu aceste numere conform unei scheme fixe, în sensul că partea supraunitară şi cea subunitară a numărului au un număr fix de biţi. În cursul evoluţiei calculatoarelor au fost folosite, practic, două convenţii de reprezentare în virgulă fixă:

- numere subunitare (cu partea întreagă nulă)
- numere supraunitare (cu partea fracţionara nulă).

Calculatoarele actuale folosesc reprezentarea în virgulă fixă pentru numere supraunitare, fără parte fracţionară, adică pentru numere întregi, cu semn.

Notă: Faptul că virgula este fixă nu înseamnă că programele nu pot opera cu numere fracţionare. Aceasta înseamnă doar că modificarea convenţiei de plasare a virgulei trebuie asumată la nivelul software, ceea ce înseamnă un efort suplimentar pentru programator.

1.2. Reprezentarea naturală a numerelor pozitive.

Considerând baza de numeraţie 2, reprezentarea naturală este una poziţională, în care valoarea asociată unei cifre este dată de valoarea cifrei (0 sau 1) şi de poziţia sa în reprezentare. Astfel, numărului scris sub forma:

$$a_{n-1}a_{n-2}\dots a_1a_0$$

Unde a_i este o cifră binară, i se asociază valoarea:

$$a_{n-1} * 2^{n-1} + a_{n-2} * 2^{n-2} + \cdots + a_1 * 2 + a_2$$

În funcţie de valoarea lui n, avem următoarele lungimi de reprezentare tipice:

octet (byte)	n=8
cuvânt (word)	n=16
dublu cuvânt (long)	n=32
cvadruplu cuvânt (quad, double long)	n=64

Tabel 1.1 Denumirea reprezentărilor tipice în funcție de numărul biților

Denumirile de mai sus au un caracter istoric și sunt dependente de context. Astfel, la începuturile calculatoarelor, prin byte se desemna cea mai mică unitate de adresare a memoriei, cu valori, de la caz la caz, de 5, 6, 7, 8 sau 9 biți. În timp, pe măsura unificării practicilor, termenii de byte și octet au devenit sinonimi. Termenul *word* poate fi folosit în contextul unui limbaj de programare de nivel înalt, ca variabilă de doi octeți, sau în contextul implementării hardware. Astfel, dacă unitatea de calcul suportă operanzi de 32 de biți atunci *word* se poate folosi pentru a desemna un astfel de operand.

Această formă de reprezentare, corespunzând mulțimii numerelor naturale, este potrivită acelor variabile care, prin natura lor, nu pot lua valori negative, de exemplu indecși în șiruri sau tabele. Numărul maxim reprezentabil pe o lungime de *n* biți este 2^n-1. Pe un octet, acest număr este 255.

Avantajul reprezentării este că unitatea aritmetică de calcul poate sesiza situația în care rezultatul unei operații de scădere este negativ și poate semnala acest fapt ca pe o eroare.

La operația de adunare este posibil ca rezultatul să depășească capacitatea de reprezentare. Pentru exemplificare adunăm numerele 65_{10} și 192_{10}, reprezentate pe octet.

$$\begin{array}{ll} 65_{10} & 0100\ 0001\ + \\ 192_{10}, & \underline{1100\ 0000} \\ & 1\ 0000\ 0001 \end{array}$$

Rezultatul, considerat pe 9 biți este reprezentarea binară a numărului 257_{10}. În fapt, rezultatul constă doar din cei mai puțin semnificativi 8 biți, cu valoarea 00000001, iar bitul al nouălea, cu valoarea 1, este memorat separat sub numele de *transport* sau *carry*. Semnificația bitului de transport depinde de modul de reprezentare folosit. La reprezentarea naturală a numerelor pozitive semnificația este de depășire a capacității de reprezentare, ceea ce este echivalent cu un rezultat eronat pe componenta de 8 biți.

La operația de scădere avem în vedere necesitatea de a face un *împrumut* din rangul anterior, atunci când este cazul. Dacă împrumutul trebuie făcut dinspre bitul de rang 2^n spre rangul 2^{n-1} atunci el se manifestă prin poziționarea fanionului *carry* și semnifică un rezultat negativ, deci eronat.

1.3. Reprezentarea semn-mărime.

În majoritatea cazurilor se cere ca domeniul de apartenență al variabilelor să fie mulțimea numerelor întregi. Una dintre reprezentările posibile este cea numită *semn-mărime*, care derivă din reprezentarea comună în formă scrisă. Deoarece semnul poate lua două valori logice, i se

va asocia un bit din cei 8, 16,... ai reprezentării, iar ceilalți se vor folosi pentru a reprezenta magnitudinea numărului (valoarea absolută).

Prin convenție, bitul cel mai semnificativ, a_{n-1}, se folosește pentru a reprezenta semnul, cu valoarea 0 pentru „+" și 1 pentru „-". Rezultă o reprezentare simetrică, în care fiecare număr pozitiv are o pereche negativă. Numărul maxim reprezentabil, ca valoare absolută, este jumătate din cel reprezentabil ca număr natural. Pe octet acest număr este 127.

Avantajele reprezentării semn-mărime sunt:

- este o reprezentare naturală în raport cu percepția umană;
- la nivelul unității aritmetice facilitează realizarea operațiilor de înmulțire și împărțire.

Dezavantaje:

- există două coduri pentru numărul 0;
- necesită tratarea semnului printr-un proces de decizie complicat.

În continuare comentăm dezavantajele.

Existența a două coduri pentru numărul zero, respectiv „+0" și „-0" complică implementarea la nivel de logică hardware a uneia dintre cele mai frecvente operații dintr-un calculator, anume comparația cu zero. Practic, trebuie prevăzute două comparatoare și apoi o operație *sau logic* (OR). De asemenea, este necesară încorporarea în unitatea aritmetică a unei scheme de unificare a reprezentării lui 0, înainte de introducerea în calcul a oricărui operand.

Operațiile de adunare și scădere necesită tratarea semnelor și compararea magnitudinii operanzilor, urmate de un proces de decizie. Aceste operații le facem, mental, fără să sesizăm, atunci când adunăm/scădem numere în baza 10, pe baza experienței deprinse prin învățare și exercițiu. Astfel:

(+3) + (+5)

este o operație imediată, în timp ce

(+3) + (-5) presupune:

- extragerea valorii absolute a operanzilor [3, 5]
- schimbarea ordinii operanzilor [3, 5 → 5, 3]
- operația de scădere [5 - 3 = 2]
- schimbarea semnului rezultatului [2 → -2]

Implementarea la nivel hardware a unui sumator/scăzător devine o operațiune extrem de costisitoare, în termeni de complexitate a schemei, iar performanțele de viteză vor fi foarte slabe.

1.4. Reprezentarea în complement de 1.

Obiectivul adoptării unui cod de reprezentare a numerelor întregi, altul decât semn-mărime, este de a simplifica schema sumatorului, respectiv scăzătorului. Ideal, un astfel de dispozitiv,

va putea aduna/scădea două numere într-o singură fază, indiferent de semnele și magnitudinile lor.

În codul complement de 1 numerele pozitive se reprezintă la fel ca în codul semn-mărime. Numerele negative se reprezintă, prin convenție, inversând bit cu bit reprezentarea numărului pozitiv.

Astfel, considerând o lungime de un octet,

numărul +19 se reprezintă $0001\ 0011_{C1}$
numărul -19 se reprezintă $1110\ 1100_{C1}$

Complementând fată de 1 un număr negativ se obține, așa cum era de așteptat, numărul pozitiv care are aceeași magnitudine, fapt ce se poate verifica imediat pe exemplul de mai sus.

Față de reprezentarea semn-mărime se păstrează semnificația bitului de semn, dar la numere negative ceilalți biți nu mai formează o reprezentare pozițională.

Se poate demonstra că adunarea a două numere în complement de 1 se poate face cu următorul algoritm:

- se adună numerele bit cu bit, tratând toți biții nediferențiat, inclusiv cei de semn;
- se adună la rezultat bitul carry.

Exemplu 1: se calculează (+5) + (-3)

unde +5 = $0000\ 0101_{C1}$, +3 = $0000\ 0011_{C1}$ şi -3 = $1111\ 1100_{C1}$

$$\begin{array}{r} 0000\ 0101\ + \\ \underline{1111\ 1100} \\ 1|\ 0000\ 0001\ + \\ \underline{\qquad\qquad 1} \\ 0000\ 0010 \end{array}$$

Exemplu 2: se calculează (-5) + (+3)

unde +5 = $0000\ 0101_{C1}$, -5 = $1111\ 1010_{C1}$ şi +3 = $0000\ 0011_{C1}$

$$\begin{array}{r} 1111\ 1010\ + \\ \underline{0000\ 0011} \\ 0|\ 1111\ 1101\ + \\ \underline{\qquad\qquad 0} \\ 1111\ 1101 \end{array}$$

Rezultatul, $1111\ 1101_{C1}$ corespunde numărului -2_{10}, ceea ce se poate verifica imediat deoarece prin complementare rezultă $0000\ 0010_{C1}$, adică $+2_{10}$. Operația de scădere se realizează prin complementarea scăzătorului, urmată de adunare.

Condiția de corectitudine a rezultatului este ca el să se încadreze în intervalul numerelor reprezentabile în cod C1 (vezi exerciții). De exemplu, încercarea de a aduna 64_{10} $(0100\ 0000_{C1})$ cu 65_{10} $(0100\ 0001_{C1})$ duce la rezultatul $0\,|\,1000\ 0001_{C1}$, care se interpretează ca număr negativ.

1.5. Reprezentarea în complement de 2.

În codul complement de 2 numerele pozitive se reprezintă la fel ca în codul semn-mărime, respectiv la fel ca în codul complement de 1. Numerele negative se reprezintă, prin convenție, adunând 1 la reprezentarea numărului în complement de 1.

Exemplificăm pentru numărul -19_{10}:

$+19_{10}$ se reprezintă $0001\ 0011$
-19_{10} se reprezintă $1110\ 1100_{C1}$

respectiv

$$\begin{array}{r} 1110\ 1100 \ + \\ \underline{\qquad\quad 1} \\ 1110\ 1101_{C2} \end{array}$$

Complementând încă o dată față de 2 obținem din nou numărul pozitiv:

acesta este -19_{10} $1110\ 1101_{C2}$
complementăm față de 1 $0001\ 0010$ +
adunăm 1 $\underline{\qquad\quad 1}$
rezultă $0001\ 0011$

Spre deosebire de codurile semn-mărime și complement de 1, există o singură reprezentare pentru numărul 0 (verificați la exerciții).

Operația de adunare se face adunând numerele bit cu bit, tratând toți biții nediferențiat, inclusiv cei de semn. Spre deosebire de adunarea în cod complement de 1, nu mai este necesar pasul suplimentar, de corecție a rezultatului prin adunarea bitului carry. Operația de scădere se face complementând al doilea operand și adunându-l la primul.

Rezumând, avem următoarele avantaje ale reprezentării în complement de doi, față de semn-mărime și complement de unu:

- Există o singură reprezentare pentru zero;
- Schimbarea semnului se face prin complementare iar scăderea se face prin adunarea descăzutului cu complementul scăzătorului;
- Adunarea se face într-un singur pas, fără a fi necesară o corecție;
- La adunare, bitul de semn este tratat la fel ca ceilalți biți.

Aceste caracteristici permit implementarea hardware unitară și simplă a unui sumator/scăzător combinaţional, făcând din codul complement de doi codul acceptat de toți producătorii pentru reprezentarea internă a numerelor întregi.

1.6. Sistemul hexazecimal

Este un sistem poziţional, la fel ca cel binar, care este folosit în matematică și domeniul computaţional pentru a reprezenta numerele în baza 16. Această formă este agreată, comparativ cu sistemul binar, pentru că este mai prezentabilă pentru oameni. Sistemul conține o mulțime de 15 simboluri, echivalentul cifrelor în baza 10. Această mulțime de element este compusă din cifre, intervalul 0-9, și caracterele alfabetice de la A până la F. Fiecărui simbol în hexazecimal îi corespunde un jumătate de byte, 4 biți. În Tabel 1.2 se găsesc *cifrele* hexazecimale și echivalentul acestora în baza 2 respectiv baza 10.

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

Tabel 1.2 Echivalentul simbolurilor din baza 16 în bazele 2 și 10

Pentru schimbarea bazei de reprezentare a unui număr din baza 2 în hexazecimal este necesar să grupăm biții numărului câte 4 începând cu cel mai puțin semnificativ bit, poziția cu puterea 0.

Exemplu 1: Reprezentarea numărului 20_{10} în hexazecimal. În acest exemplu vom folosi un byte pentru reprezentarea numerelor.

Reprezentăm numărului 20_{10} în baza 2	$0001\ 0100_2$
Grupăm biții câte 4	grupul 1 – 0100 grupul 2 – 0001
Verificăm echivalentul grupurilor în Tabel 1.2	$0100_2 - 4_{16}$ $0001_2 - 1_{16}$
Rezultatul se obține prin concatenarea celor două simboluri	14_{16}

Reprezentarea hexazecimală se regăsește sub mai multe forme. De obicei, în matematică se folosește $5A_{16}$. În programare, de exemplu limbajul C, valoarea este precedată de grupul de simboluri *0x*, precum în 0x5A. De asemenea, în limbajul de asamblare mai putem regăsi valoarea succedată de caracter *H*, precum în 5AH, care reprezintă valoarea $5\mathrm{A}_{16}$.

1.7. Exerciții.

1. Să se reprezinte în semn-mărime, complement de 1 și complement de 2 următoarele valori: 29, -45, -90.

2. Care este reprezentarea în hexazecimal a valorii binare 111100_2? Care este reprezentarea valorii $10\mathrm{A}_{16}$ în binar?

3. Cum se poate transforma o valoare din baza 10 în baza 16, fără a apela la reprezentarea binară? Dar invers, din hexazecimal în zecimal? Pentru primul caz considerați valoare 33_{10} iar pentru al doilea caz considerați valoare $\mathrm{A1}_{16}$.

4. Folosindu-vă de metodele folosite de la exercițiul 3, precizați care sunt reprezentările zecimale ale următoarelor valori: 23_4, 73_8, 57_6.

5. Să se reprezinte numărul 0 în cele trei moduri. Să se verifice relația 0+0=0, în toate variantele, în cele trei reprezentări. Să se verifice relația x-x=0, în cele trei reprezentări.

6. Să se determine domeniul de reprezentabilitate (numărul maxim, pozitiv și cel minim, negativ) în cele trei moduri de reprezentare. Se vor calcula valorile, în baza 10, pentru dimensiuni de 8, 16 și 32 de biți.

7. Să se aleagă două numere, primul pozitiv și al doilea negativ, să se reprezinte în cele trei moduri și să se facă operațiile de adunare și scădere între ele.