# IV. Reprezentări interne

- reprezentări interne elementare
  - fac parte din arhitectura calculatorului
  - deci sunt implementate în hardware
  - accesibile direct programatorilor
- structuri de date mai complexe
  - pe baza reprezentărilor elementare
  - definite și accesibile programatorilor prin software

# Reprezentări elementare

- date numerice
  - numere întregi, raționale
  - doar anumite submulțimi ale acestora
- date alfa-numerice
  - caractere etc.
- instrucțiuni
  - singurele specifice fiecărui sistem
  - deci nestandardizate și neportabile

# Studiul reprezentărilor

- reprezentări numerice

  $\mathrm{repr}(n_1)\ \mathit{op}\ \mathrm{repr}(n_2) = \mathrm{repr}(n_1\ \mathit{op}\ n_2)$ ???
- exemplu - dacă adunăm două variabile întregi, rezultatul va fi scris corect?
- erori de reprezentare
  - aproximări
  - depășiri

# Transmiterea informațiilor

- între diverse medii
  - între calculatoare/sisteme
  - între componente ale aceluiași calculator/sistem
- pot apărea erori de transmisie
  - datorită perturbărilor/funcționării incorecte
  - semnal digital - unii biți sunt inversați
  - se dorește detectarea apariției acestor erori
  - și chiar corectarea lor, unde e posibil

# Moduri de detectare/corectare

- adăugarea de biți suplimentari *redundanți*
- paritate - 1 bit suplimentar
  - permite detecția apariției unei erori (pe 1 bit)
  - paritate (im)pară: număr total (im)par de biți 1
- cod Hamming
  - 4 biţi de informaţie, 3 biţi suplimentari
  - permite detectarea/corecţia mai multor erori simultan

# Exemplu paritate impară

- emițător
  - are de trimis valoarea $(110)_2$
  - 2 biți (par) pe 1, deci bitul suplimentar va fi 1
  - se va trimite $(1101)_2$
- receptor
  - primește șirul de biți
  - dacă numărul de biți pe 1 este par - eroare
  - altfel - elimină bitul de paritate și obține $(110)_2$

# IV.1. Codificări alfanumerice

# Codificări alfanumerice

- calculatorul nu poate reprezenta direct caractere
  – sau alte informații nenumerice: imagini etc.
- fiecărui caracter îi este asociat în mod unic un număr
  – este de fapt o codificare
  – codificarea poate fi la nivel hardware (reprezentare elementară) sau software

# Standarde

- ASCII
  - fiecare caracter - 7 biți plus unul de paritate
- EBCDIC
  - fost concurent al ASCII
- ISO 8859-1
  - extinde codul ASCII: diacritice etc.
- Unicode, UCS
  - caractere non-latine

# Codul ASCII

- literele mici au coduri consecutive
  - în ordinea dată de alfabetul englez
  - 'a' - 97; 'b' - 98; ...; 'z' - 122
- similar - literele mari (65, 66, ..., 90)
- similar - caracterele care afișează cifrele zecimale
  - atenție: codul pentru caracterul '0' este 48 (nu 0)
- comparații lexicografice - comparator binar

# IV.2. Reprezentări interne numerice

# Scrierea pozițională

- este tot o reprezentare
  - 397 nu este un număr, ci reprezentarea unui număr
- introdusă de indieni/arabi
- factor implicit ataşat fiecărei poziţii din reprezentare
- esenţială pentru arhitectura calculatoarelor
  - permite algoritmi eficienţi de calcul (adunare...)

# Baze de numeraţie

- orice număr natural d>1
- mulţimea cifrelor în baza d: {0,1,…,d-1}
- calculatorul lucrează în baza d=2
  - tehnic: cel mai uşor de implementat fizic 2 cifre
  - teoretic: baza 2 se "potrivește" cu logica booleană
    - ca simboli şi ca operaţii
    - operațiile se pot implementa prin funcții booleene

# Limitele reprezentărilor

- în practică, numărul de cifre este finit
- exemplu - numere întregi fără semn
  - pe 1 octet: $0 \div 2^8-1$ (= 255)
  - pe 2 octeți: $0 \div 2^{16}-1$ (= 65535)
  - pe 4 octeți: $0 \div 2^{32}-1$ (= 4294967295)
- orice număr mai mare (sau mai mic) decât limitele nu va putea fi reprezentat corect

# Scrierea poziţională

- fie baza $d \in N^{*}-\{1\}$
- și reprezentarea dată de șirul de cifre

  $a_{n-1}a_{n-2}\ldots a_1a_0a_{-1}\ldots a_{-m}$
- numărul corespunzător reprezentării este

$$\sum_{i=-m}^{n-1}(a_i \times d^i)_{(10)}$$

- $d^i$ este factorul implicit asociat poziției i
  - inclusiv pentru puteri negative

# Treceri dintr-o bază d în baza 10

- conform formulei anterioare
- virgula rămâne în același loc
- exemplu

  $5E4,D_{(16)} = 5\times16^{2} + 14 \times16^{1} + 4 \times16^{0} +13 \times16^{-1} = 20480 + 3584 + 64 + 0,8125 = 24128,8125_{(10)}$

# Trecerea din baza 10 în baza d

Exemplu: $87{,}35_{(10)} = 1010111{,}01(0110)_{(2)}$

partea întreagă

$87 / 2 = 43$ rest 1

$43 / 2 = 21$ rest 1

$21 / 2 = 10$ rest 1

$10 / 2 = 5$ rest 0

$5 / 2 = 2$ rest 1

$2 / 2 = 1$ rest 0

$1 / 2 = 0$ rest 1

$87_{(10)} = 1010111_{(2)}$

(cifrele se scriu de jos în sus)

partea fracționară

$0{,}35 \times 2 = 0{,}7 + 0$

$0{,}7 \times 2 = 0{,}4 + 1$

$0{,}4 \times 2 = 0{,}8 + 0$

$0{,}8 \times 2 = 0{,}6 + 1$

$0{,}6 \times 2 = 0{,}2 + 1$

$0{,}2 \times 2 = 0{,}4 + 0$

$0{,}4 \times 2 = 0{,}8 + 0$

(perioadă)

$0{,}35_{(10)} = 0{,}01(0110)_{(2)}$

# Conversii între baze

- o bază este o putere a celeilalte baze
  - $d_1 = d_2^k \Rightarrow$ fiecărei cifre în baza $d_1$ îi corespund exact k cifre în baza $d_2$
- ambele baze sunt puteri ale numărului *n*
  - conversia se poate face prin intermediul bazei *n*

$703{,}102_{(8)} = 111\ 000\ 011{,}001\ 000\ 010_{(2)} =$

$= 0001\ 1100\ 0011{,}0010\ 0001\ 0000_{(2)} =$

$= 1C3{,}21_{(16)}$

# Aproximare și depășire

- o reprezentare are *n* cifre la partea întreagă și *m* cifre la partea fracționară
  - *n* și *m* sunt finite
- dacă numărul necesită mai mult de *n* cifre la partea întreagă, se produce depăşire
- dacă numărul necesită mai mult de *m* cifre la partea fracționară, apare o aproximare
  - de cel mult $2^{-m}$

# IV.3. Reprezentările BCD și în exces

# Reprezentarea BCD

- numerele sunt reprezentate ca şiruri de cifre în baza 10
  - fiecare cifră este reprezentată pe 4 biţi
- utilitate
  - aplicații de tip business (financiar etc.)
  - afișaje în baza 10 (temperatură etc.)
- calculele sunt dificil de efectuat
  - adunare - nu se poate utiliza direct un sumator

# Adunarea BCD (1)

$$\begin{array}{lcll} 5 & = & 0101 & + \\ 3 & = & 0011 & \\ \hline 8_{(10)} & = & 1000 & = 8_{BCD} \end{array}$$

$$\begin{array}{lcll} 5 & = & 0101 & + \\ 8 & = & 1000 & \\ \hline 13_{(10)} & = & 1101 & \end{array}$$

$$\neq 13_{BCD} =$$
$$= 0001\ 0011$$

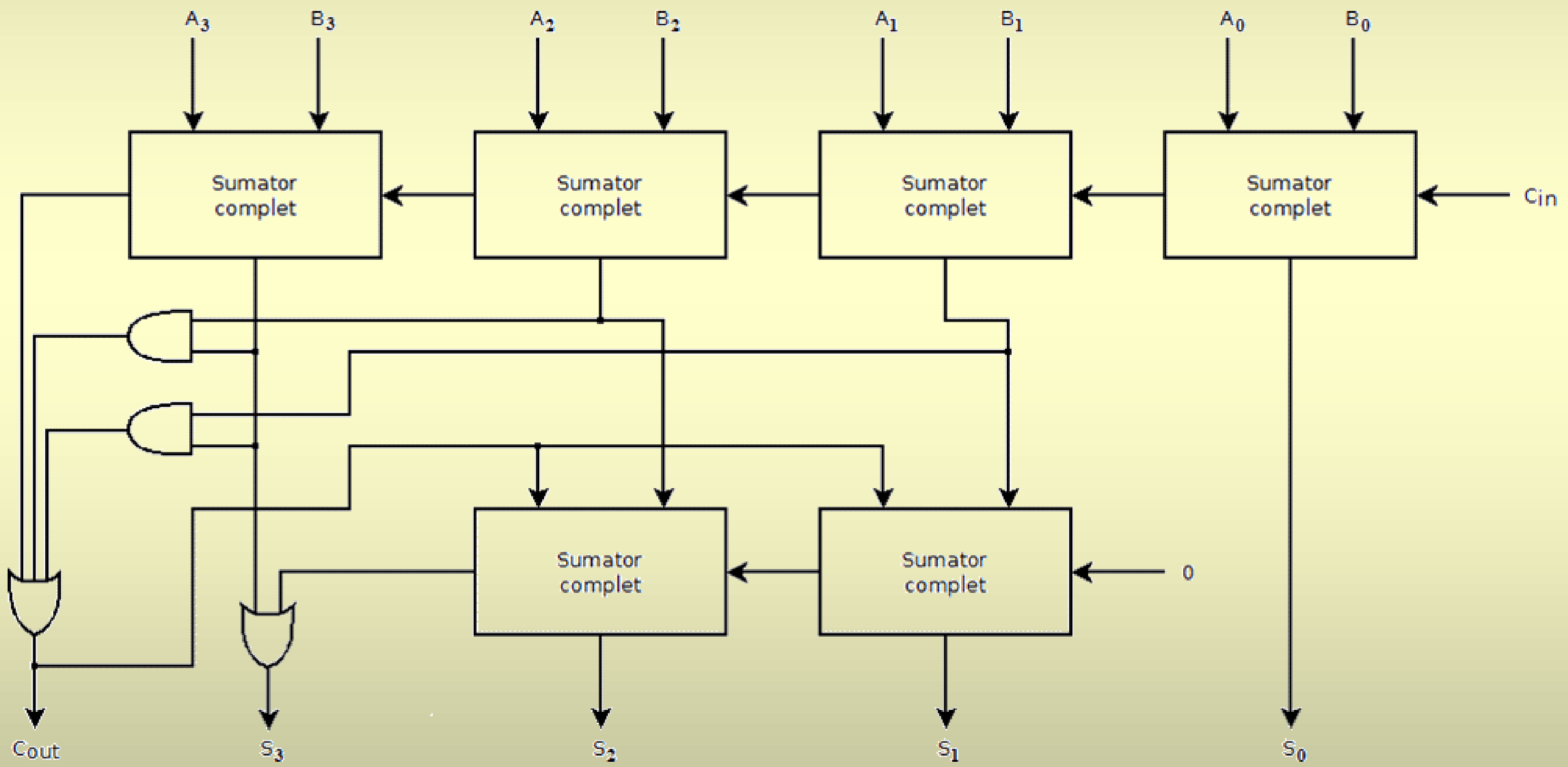
problemele apar atunci când suma cifrelor depăşeşte 9

# Adunarea BCD (2)

- soluţie
  - se adună 6 (0110) atunci când suma depăşeşte 9
- temă: de ce?

$$
\begin{array}{lcrl}
5 & = & 0101 & + \\
8 & = & \underline{1000} & \\
 & & 1101 & + \\
6 & = & \underline{0110} & \\
 & & 1\ 0011 & = 13_{BCD}
\end{array}
\qquad
\begin{array}{lcrl}
9 & = & 1001 & + \\
7 & = & \underline{0111} & \\
16_{(10)} & = & 1\ 0000 & \neq 16_{BCD} \\
6 & = & \underline{0110} & \\
 & & 1\ 0110 & = 16_{BCD}
\end{array}
$$

# Sumator BCD


A3
B3
A2
B2
A1
B1
A0
B0
Sumator complet
Sumator complet
Sumator complet
Sumator complet
Cin
Sumator complet
Sumator complet
0
Cout
S3
S2
S1
S0

# Reprezentarea în exces

- pornește de la scrierea pozițională
  - numere pozitive
  - pe $n$ biți, intervalul reprezentabil este $0 \div 2^n-1$
- reprezentarea Excess-$k$
  - pentru fiecare șir de biți, din valoarea care îi corespunde  în scrierea pozițională se scade $k$
  - intervalul reprezentabil devine $-k \div 2^n-k-1$

# Exemplu: Excess-5

| Binar | Zecimal | Excess-5 | Binar | Zecimal | Excess-5 |
|---|---|---|---|---|---|
| 0000 | 0 | -5 | 1000 | 8 | 3 |
| 0001 | 1 | -4 | 1001 | 9 | 4 |
| 0010 | 2 | -3 | 1010 | 10 | 5 |
| 0011 | 3 | -2 | 1011 | 11 | 6 |
| 0100 | 4 | -1 | 1100 | 12 | 7 |
| 0101 | 5 | 0 | 1101 | 13 | 8 |
| 0110 | 6 | 1 | 1110 | 14 | 9 |
| 0111 | 7 | 2 | 1111 | 15 | 10 |

# IV.4. Reprezentări în virgulă fixă

# Reprezentări numerice: probleme

- reprezentarea semnului
  - nu există simbol special, doar cele pentru cifre
- virgula
  - trebuie cunoscută în orice moment poziția sa
- operații aritmetice
  - implementare cât mai eficientă
  - nu este posibil pentru toate operațiile simultan
  - trebuie decis pe care le optimizăm

# Codificări în virgulă fixă

- semnul - este folosit unul dintre biți
- virgula
  - are întotdeauna aceeași poziție în șirul de cifre
    - deci poziția nu mai trebuie memorată
- operații implementate eficient
  - adunare, scădere
- codificările - pe **n**+**m** biţi (**n**≥1, **m**≥0)
  - **m**=0 - numere întregi
  - **n**=1 - numere subunitare

# Codificări redundante

- codificare redundantă
  – există cel puțin un număr cu două reprezentări diferite
  – probleme la operațiile aritmetice
- codificările folosite în practică
  – reprezentarea numerelor pozitive - la fel ca la numere fără semn; diferențe - numere negative
  – unele prezintă două reprezentări pentru 0

# Reprezentarea prin modul şi semn

- notație: A+S

$$val_{A+S}^{n,m}(a_{n-1}a_{n-2}\ldots a_1a_0a_{-1}\ldots a_{-m})=$$

$$=\begin{cases} a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m} & \text{dacă } a_{n-1}=0 \\ -(a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m}) & \text{dacă } a_{n-1}=1 \end{cases}$$

- similar cu scrierea în baza 2
  – bitul cel mai din stânga reprezintă semnul
  – virgula este implicită

# Modul şi semn - limite

- pe n+m biți sunt $2^{n+m}$ reprezentări diferite
  – dar numai $2^{n+m}-1$ numere diferite
  – redundantă: $val_{A+S}^{n,m}(00...0)=val_{A+S}^{n,m}(10...0)=0$
- valorile extreme reprezentabile

$max_{A+S}^{n,m}=val_{A+S}^{n,m}(01...1)=2^{n-1}-2^{-m}$

$min_{A+S}^{n,m}=val_{A+S}^{n,m}(11...1)=-\left(2^{n-1}-2^{-m}\right)$

  – deci numerele reprezentabile sunt în intervalul $[-(2^{n-1}-2^{-m}); +(2^{n-1}-2^{-m})]$

# Modul şi semn - precizie

- numerele reprezentabile exact încep cu min=$-(2^{n-1}-2^{-m})$
  - și continuă cu pasul $2^{-m}$
- celelalte numere din interval - aproximare
  - eroarea - cel mult $2^{-m}$
- deci precizia reprezentării este $2^{-m}$
- pentru n+m fixat
  - numere mai mari = precizie mai slabă și invers

# Exemple (1)

$val_{A+S}^{8,0}(00110011)=2^5+2^4+2^1+2^0=51$

$val_{A+S}^{6,2}(00110011)=2^3+2^2+2^{-1}+2^{-2}=12,75$

sau

$val_{A+S}^{6,2}(00110011)=val_{A+S}^{8,0}(00110011):2^2=51:4=12,75$

$val_{A+S}^{4,4}(00110011)=2^1+2^0+2^{-3}+2^{-4}=3,1875$

sau

$val_{A+S}^{4,4}(00110011)=val_{A+S}^{8,0}(00110011):2^4=51:16=3,1875$

# Exemple (2)

$$val_{A+S}^{8,0}(10110011)=-(2^5+2^4+2^1+2^0)=-51$$

$$val_{A+S}^{4,4}(10110011)=-(2^1+2^0+2^{-3}+2^{-4})=-3,1875$$

sau

$$val_{A+S}^{4,4}(10110011)=val_{A+S}^{8,0}(10110011):2^4=-51:16=-3,1875$$

$$min_{A+S}^{8,0}=val_{A+S}^{8,0}(11111111)=-127$$

$$min_{A+S}^{4,4}=val_{A+S}^{4,4}(11111111)=-7,9375$$

sau

$$min_{A+S}^{4,4}=min_{A+S}^{8,0}:2^4=-127:16=-7,9375$$

# Exemple (3)

$\max_{A+S}^{8,0}=\mathrm{val}_{A+S}^{8,0}(01111111)=127$

$\max_{A+S}^{4,4}=\mathrm{val}_{A+S}^{4,4}(01111111)=7,9375$

sau

$\max_{A+S}^{4,4}=\max_{A+S}^{8,0}:2^4=127:16=7,9375$

- intervale reprezentabile
  - $A+S^{8,0}$: [-127; 127] $\rightarrow$ 255 numere, din 1 în 1
  - $A+S^{4,4}$: [-7,9375; 7,9375] $\rightarrow$ 255 numere, din 0,0625 în 0,0625 (=1:16)

# Operații în A+S

- adunare/scădere
  - stabilirea semnului rezultatului (comparaţie)
  - aplicarea algoritmilor cunoscuți
- înmulţire/împărţire
  - similar algoritmilor cunoscuți
- în general mai complex decât ne-am dori
  - nu putem utiliza pur și simplu un sumator "clasic" pentru adunare