

# Depășiri

- depășire inferioară
  - în forma normalizată, exponentul negativ nu poate fi reprezentat în câmpul caracteristicii
    - numărul va fi considerat 0
- depășire superioară
  - în forma normalizată, exponentul pozitiv nu poate fi reprezentat în câmpul caracteristicii
    - numărul va fi considerat  $\pm\infty$ , după caz

# Reprezentări denormalizate

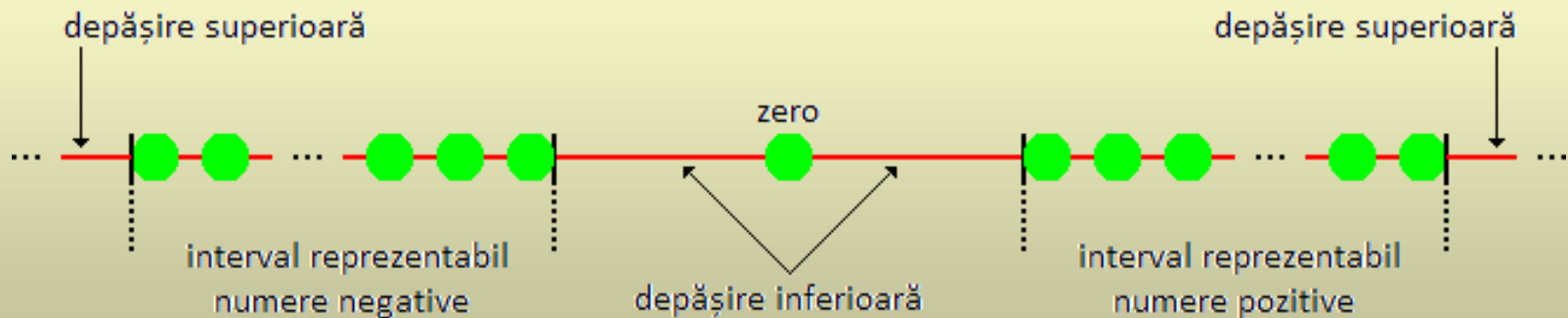
- număr mai mic în modul decât cea mai mică reprezentare normalizată nenulă
  - se renunță la normalizare
  - exponentul are valoarea minimă
    - simplă precizie: -127
    - dublă precizie: -1023
  - în acest caz, mantisa va fi 0.f, în loc de 1.f

## Aproximări (1)

- depășirea inferioară este de fapt aproximare
  - un număr nenul foarte mic este considerat 0
- care este precizia reprezentării în virgulă mobilă?
  - depinde de exponent
  - simplă precizie:  $2^{e-23}$
  - dublă precizie:  $2^{e-52}$
- exponent foarte mare - precizie foarte slabă

## Aproximări (2)

- exemplu:  $e = 123$ 
  - diferența între două numere consecutive reprezentabile exact este  $2^{123-23} = 2^{100} \approx 10^{30}$
- ce putem reprezenta exact?
  - numere raționale (nu reale) - doar o parte



# Aritmetica în virgulă mobilă

- fie două numere

$$x = m_x \cdot 2^{e_x}$$

$$y = m_y \cdot 2^{e_y}$$

- operațiile aritmetice elementare

$$x + y = (m_x \cdot 2^{e_x - e_y} + m_y) \cdot 2^{e_y}, \text{ dacă } e_x \leq e_y$$

$$x - y = (m_x \cdot 2^{e_x - e_y} - m_y) \cdot 2^{e_y}, \text{ dacă } e_x \leq e_y$$

$$x \cdot y = (m_x \cdot m_y) \cdot 2^{e_x + e_y}$$

$$x : y = (m_x : m_y) \cdot 2^{e_x - e_y}$$

# Adunarea în virgulă mobilă

- compararea exponenților
  - se egalizează prin deplasarea unei mantise
- adunarea mantiselor
  - în complement față de 2
- normalizarea sumei
  - dacă apare depășire - oprire
- rotunjirea mantisei rezultat la numărul permis de biți

# Înmulțirea în virgulă mobilă

- adunarea exponenților
- înmulțirea mantiselor
- normalizarea produsului
  - dacă apare depășire - oprire
- rotunjirea mantisei rezultat la numărul permis de biți
- determinarea semnului rezultatului

# Temă

- urmăriți pașii adunării și respectiv înmulțirii în virgulă mobilă pentru reprezentările numerelor scrise în baza zece ca  $-0,75$  și  $0,375$
- se va considera reprezentarea în simplă precizie



# V. Arhitectura și organizarea calculatorului

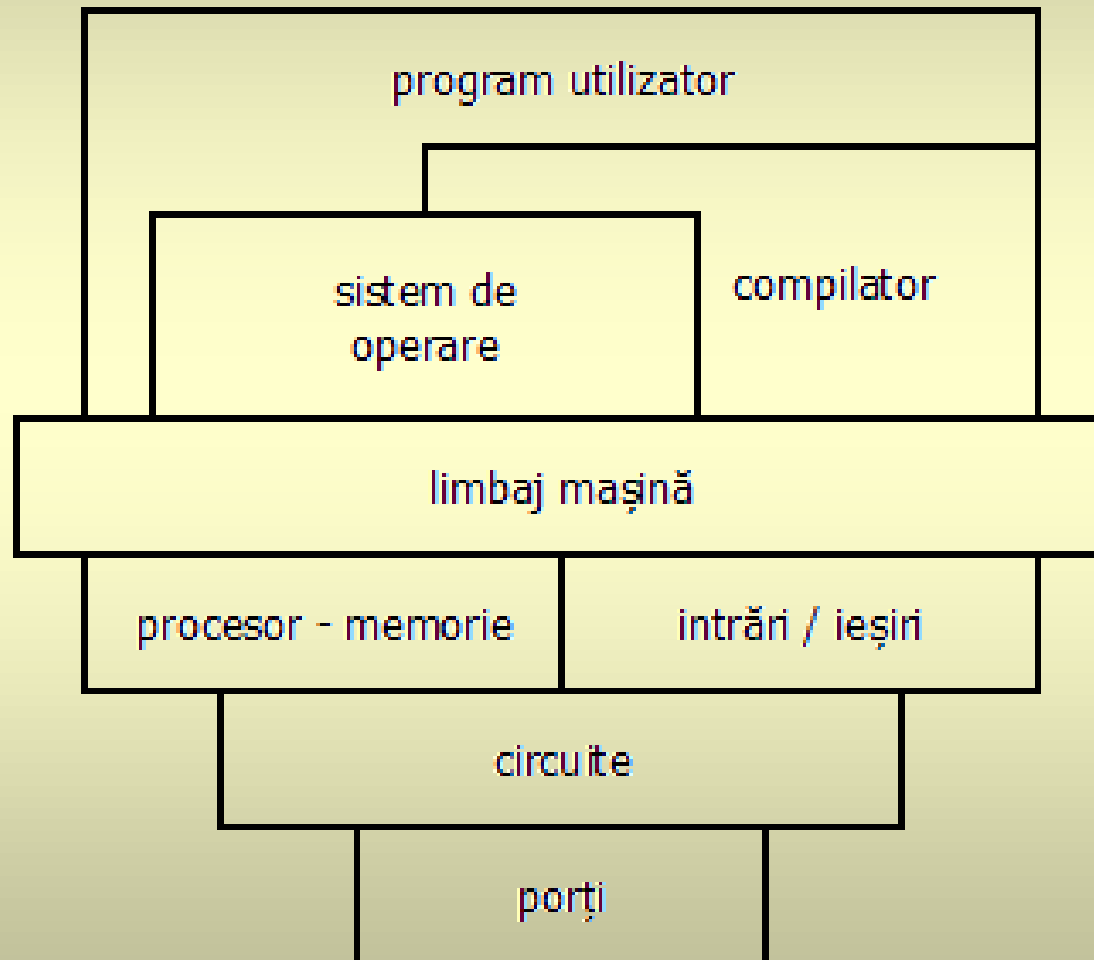
# Calculatorul von Neumann (1)

- program memorat
  - codurile instrucțiunilor sunt reținute în aceeași memorie ca și datele
- memoria
  - ideal infinită; timp de acces egal la orice locație
- după execuția unei instrucțiuni urmează
  - instrucțiunea memorată imediat după ea
  - sau instrucțiunea indicată de cea curentă (salt)

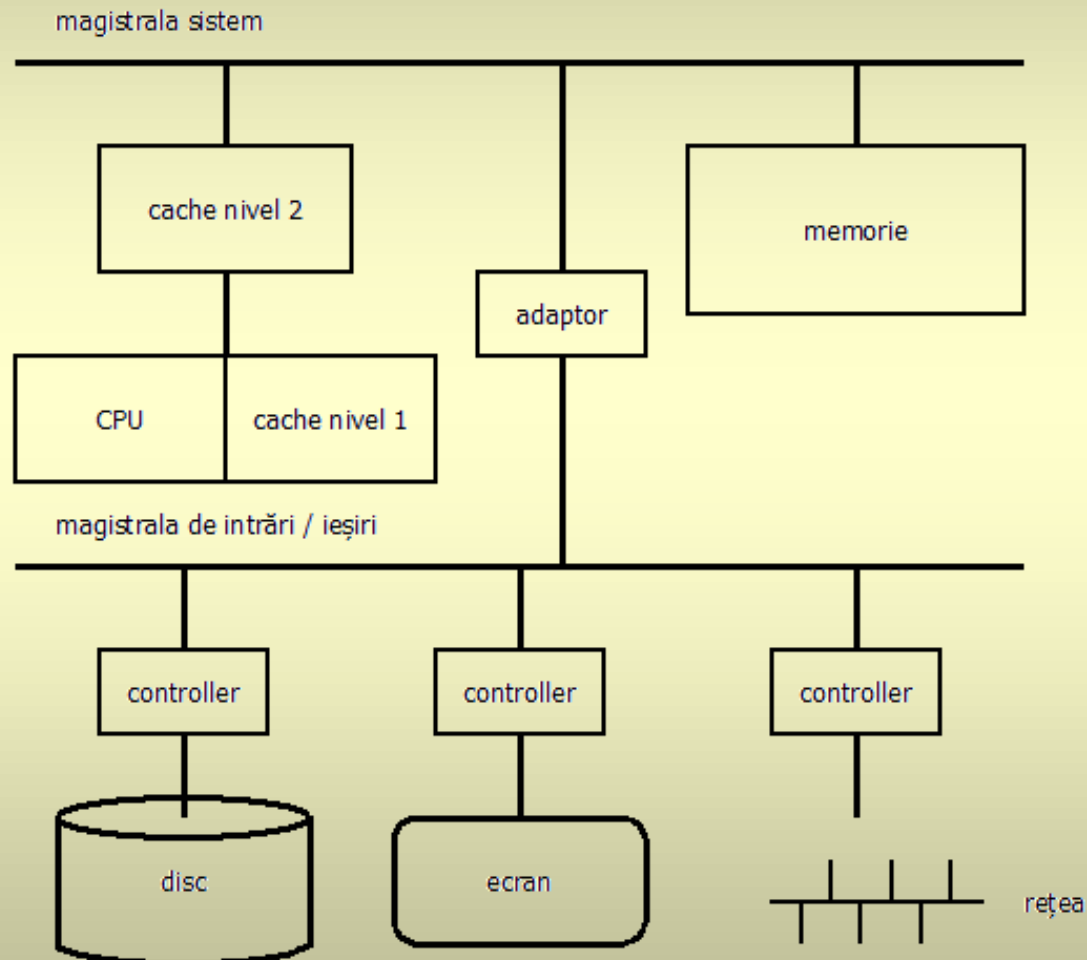
## Calculatorul von Neumann (2)

- adresa instrucțiunii următoare este reținută în permanență
  - într-un registru dedicat - PC (Program Counter)
  - și actualizată permanent, în funcție de tipul instrucțiunii curente ("normală" sau de salt)
- în fiecare moment, o singură instrucțiune este încărcată pentru execuție

# Arhitectura unui sistem de calcul



# Organizarea unui calculator



# **V.1. Arhitectura calculatorului: viziuni posibile**

# Programator vs. utilizator

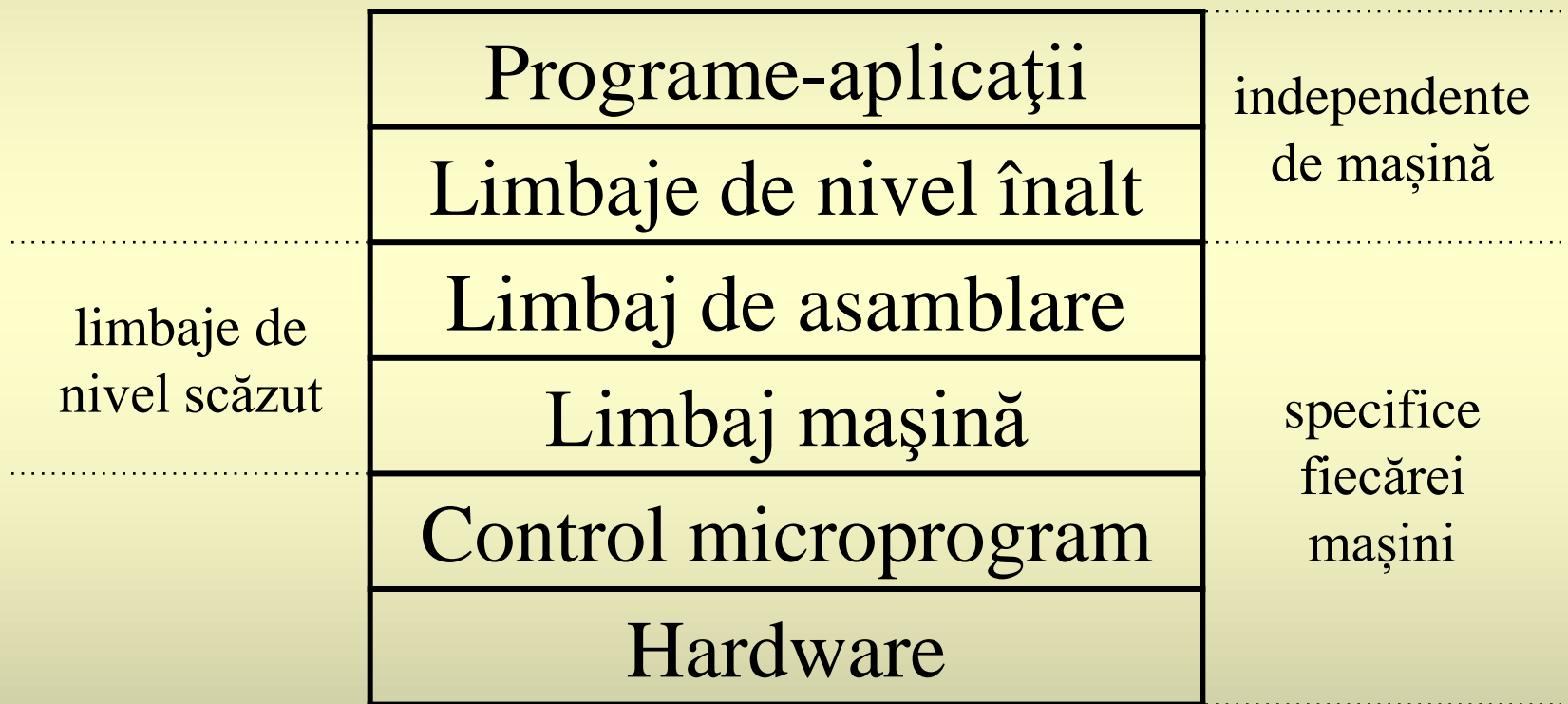
- arhitectura calculatorului - formată din
  - arhitectura setului de instrucțiuni (ISA)
  - organizarea mașinii
- viziunea utilizatorului
  - sistemul hardware
    - pe care rulează
  - sistemul de operare
    - peste care rulează
  - aplicații software

# Viziunea programatorului (1)

- reprezentată de arhitectura setului de instrucțiuni
- materializată prin diverse limbaje
- în ordine crescătoare a nivelului de abstractizare
  - limbajul mașină
  - limbajul de asamblare
  - limbaje de nivel înalt



## Viziunea programatorului (2)



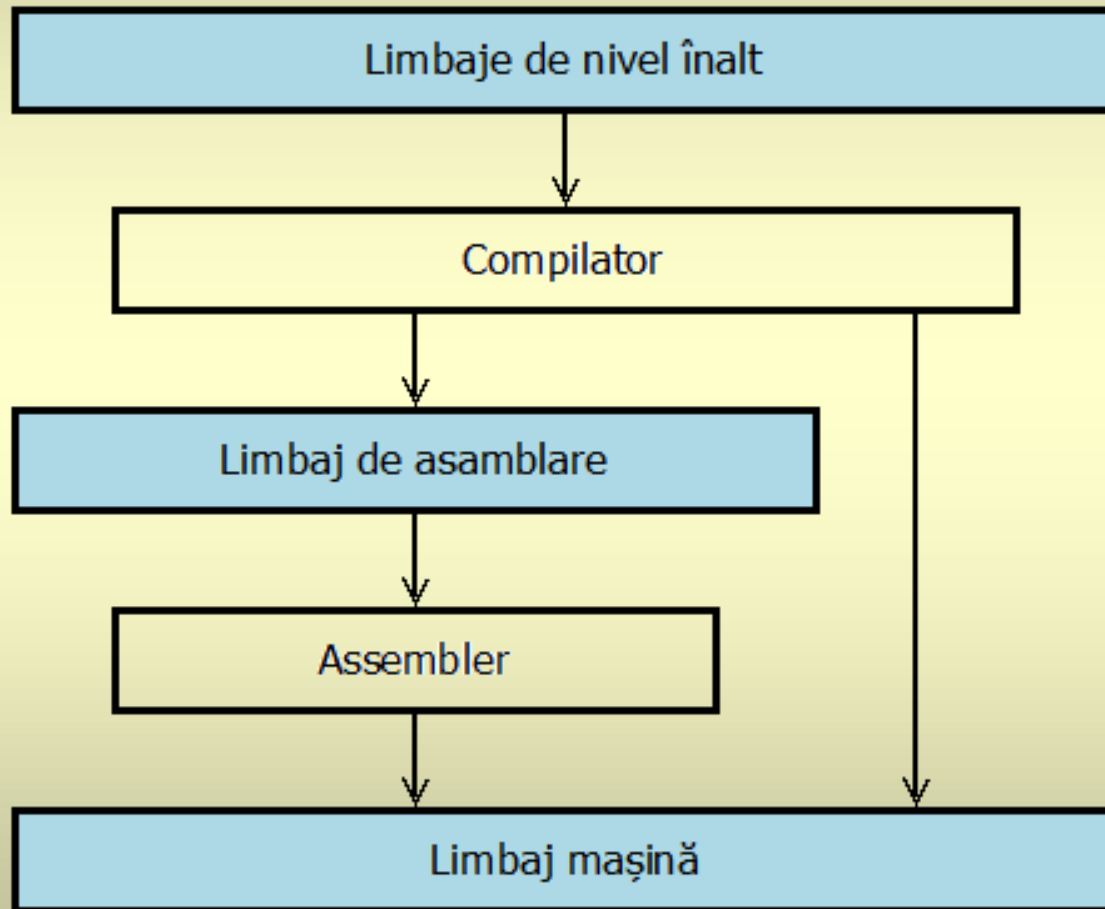
# Limbaje de nivel scăzut

- limbajul mașină
  - caracteristic fiecărui procesor
  - constă din cuvinte peste alfabetul  $\{0, 1\}$
- limbajul de asamblare
  - nivel (puțin) mai înalt
  - corespondență cu limbajul mașină (instrucțiuni)
- exemple
  - limbaj mașină: **0100 1010 1101 0010**
  - limbaj de asamblare: **sub myvar,5**

# Traducere (1)

- singurul limbaj înțeles de procesor este limbajul mașină
  - pentru a fi executat, un program trebuie tradus
- asamblor (assembler)
  - traduce limbajul de asamblare în limbaj mașină
- compilator
  - traduce un limbaj de nivel înalt în limbaj mașină
  - direct sau via limbajul de asamblare

# Traducere (2)



# Limbaje de nivel înalt - avantaje

- dezvoltare mai rapidă a programelor
  - instrucțiuni mai ușor de înțeles și mai puține
- întreținere mai ușoară a programelor
  - aceleași motive
- portabilitatea programelor
  - puține detalii dependente de mașină
    - dar nu deloc - portabilitatea nu e chiar 100%
  - fiecare limbaj necesită un compilator specific

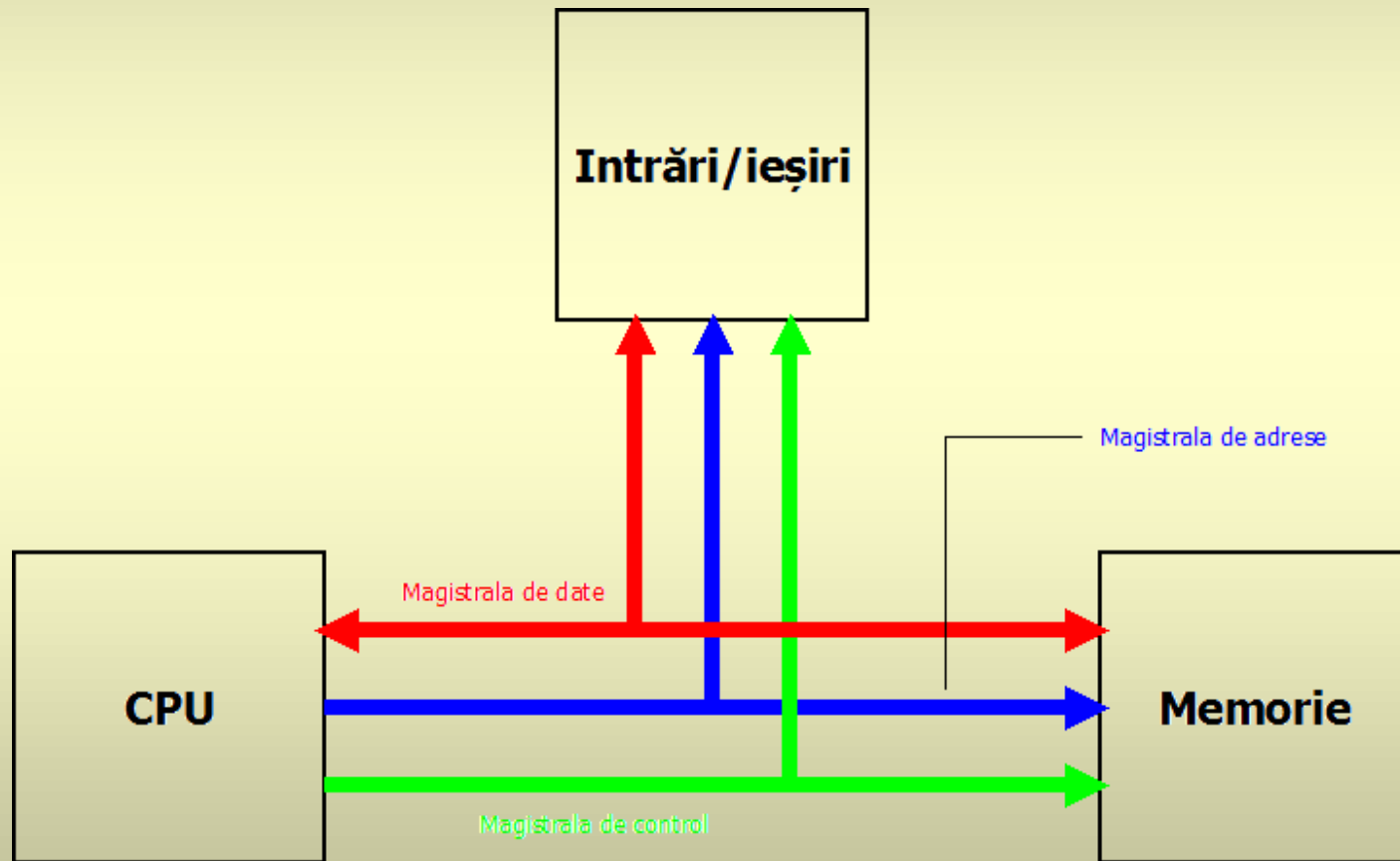
# Limbaj de asamblare - avantaje

- eficiență
  - cod mai compact - ocupă mai puțină memorie
  - viteză - execuție mai rapidă
  - și compilatoarele urmăresc eficiența
    - dar programatorul poate obține rezultate mai bune
- acces la resursele hardware
  - exemplu: procesoarele au biți care indică transportul și depășirea la adunare
    - nu sunt accesibile din limbajele de nivel înalt

# Arhitect vs. implementator

- arhitectul - proiectare de nivel înalt
  - componente complexe, fără a intra în detalii
  - cele principale
    - procesorul
    - memoria
    - dispozitivele de intrare/ieșire (I/O)
  - la care se adaugă magistralele prin care sunt interconectate

# Viziunea arhitectului

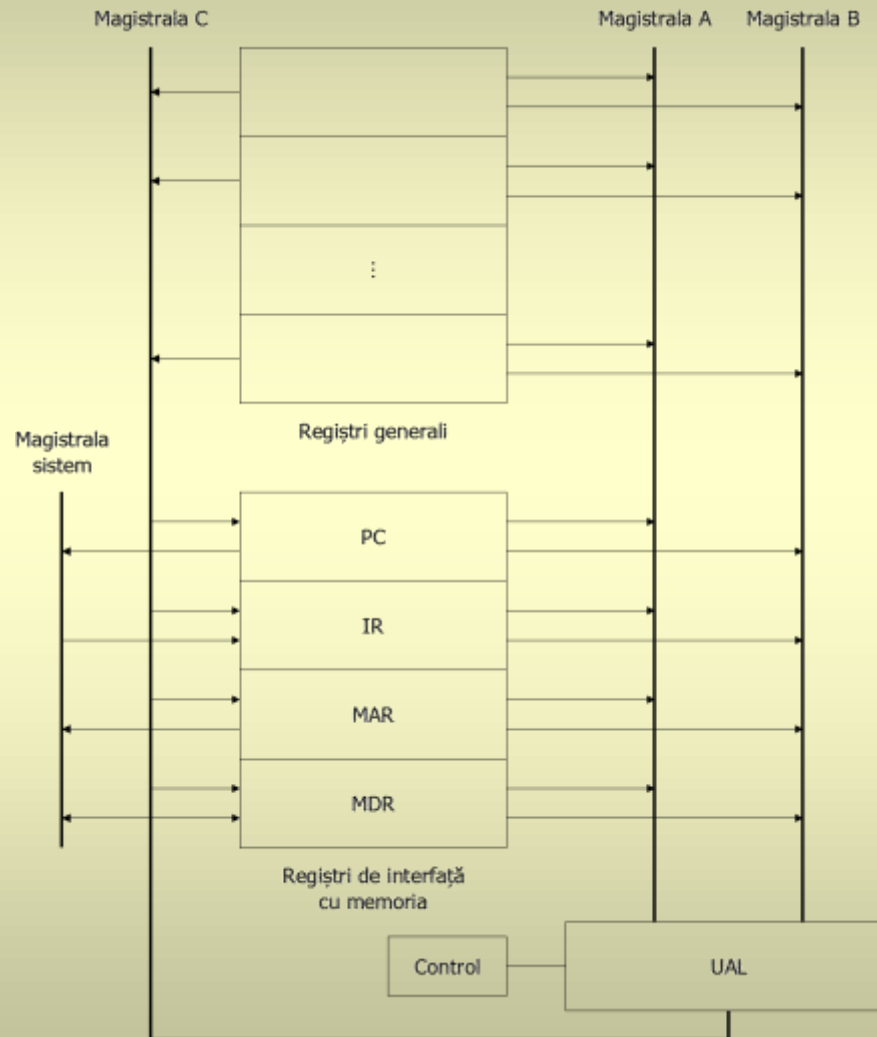




# Viziunea implementatorului

- proiectarea și detalierea componentelor de nivel înalt folosite de arhitect
  - până la nivelul porților logice
- exemplu: procesorul
  - unitatea de control
  - calea de date: regiștri, unitatea aritmetico-logică
- aceeași arhitectură a setului de instrucțiuni poate fi implementată în moduri diferite

# Calea de date



# V.2. Organizarea calculatorului

# Componentele principale

- Unitatea centrală de procesare (CPU)
- Memoria
- Dispozitive periferice (I/O = *input/output*)
- Magistrale
  - de date
  - de adrese
  - de control

# Unitatea centrală de procesare

- numită și procesor
- execută instrucțiunile indicate de programator
- realizează prelucrarea datelor
- coordonează funcționarea celorlalte componente

# Memoria

- stocarea informațiilor
  - date
  - instrucțiuni
- furnizarea informațiilor la cerere
- rol pasiv
  - "răspunde" la cererile venite din exterior
  - nu inițiază niciodată un transfer

# Dispozitivele periferice

- comunicarea cu exteriorul
- funcții - foarte variate
  - preluare date
  - afișare
  - imprimare
  - stocare (persistentă)
  - etc.

# Magistralele

- căi de legătură între CPU, memorie și periferice
- după informația care le parcurge
  - de date - date și instrucțiuni
  - de adrese - adrese pentru memorie și dispozitive periferice
  - de control - semnale prin care CPU comunică cu celelalte circuite și le controlează



## V.3. Memoria

# Tipuri de memorie

- ROM (*Read-Only Memory*)
  - conținutul său poate fi citit, dar nu și modificat
  - nevolatilă (nu își pierde conținutul la întreruperea alimentării)
- RAM (*Random Access Memory*)
  - conținutul său poate fi citit și modificat
  - volatilă (își pierde conținutul la întreruperea alimentării)

# Memoria ROM - tehnologii

- PROM (*Programmable ROM*) - conținutul său poate fi programat de utilizator
- EPROM (*Erasable PROM*) - poate fi șters și reprogramat de mai multe ori
  - UVEEPROM (*Ultra-Violet EPROM*) - ștergere prin expunere la radiație UV
  - EEPROM (*Electrical EPROM*) - ștergere prin impulsuri electrice

# Memoria RAM - tehnologii

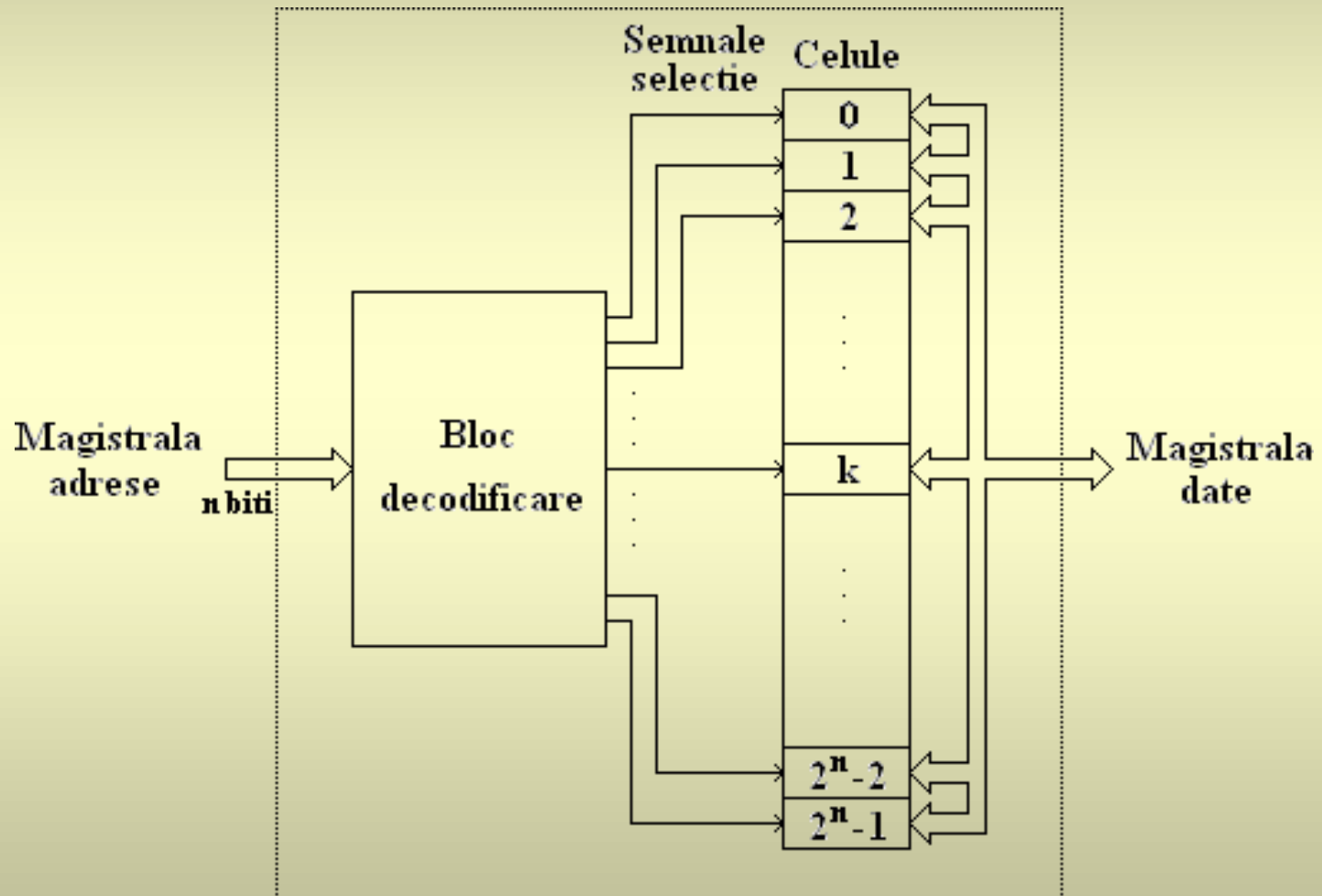
- SRAM (*Static RAM*)
  - viteză mare
  - preț ridicat
- DRAM (*Dynamic RAM*)
  - mai lentă
  - densitate de integrare mare → spațiu ocupat mic
  - preț mai redus

## Structura memoriei (1)

- șir unidimensional de celule (locații)
- fiecare celulă are asociat un număr unic - *adresa*
- bloc decodificare - selectează locația cu adresa indicată
- dimensiunea circuitului de memorie - dată de numărul de biți de adresă

$$dimensiune = 2^{nr\_biti\_adresa}$$

## Structura memoriei (2)



# Spațiul adreselor de memorie

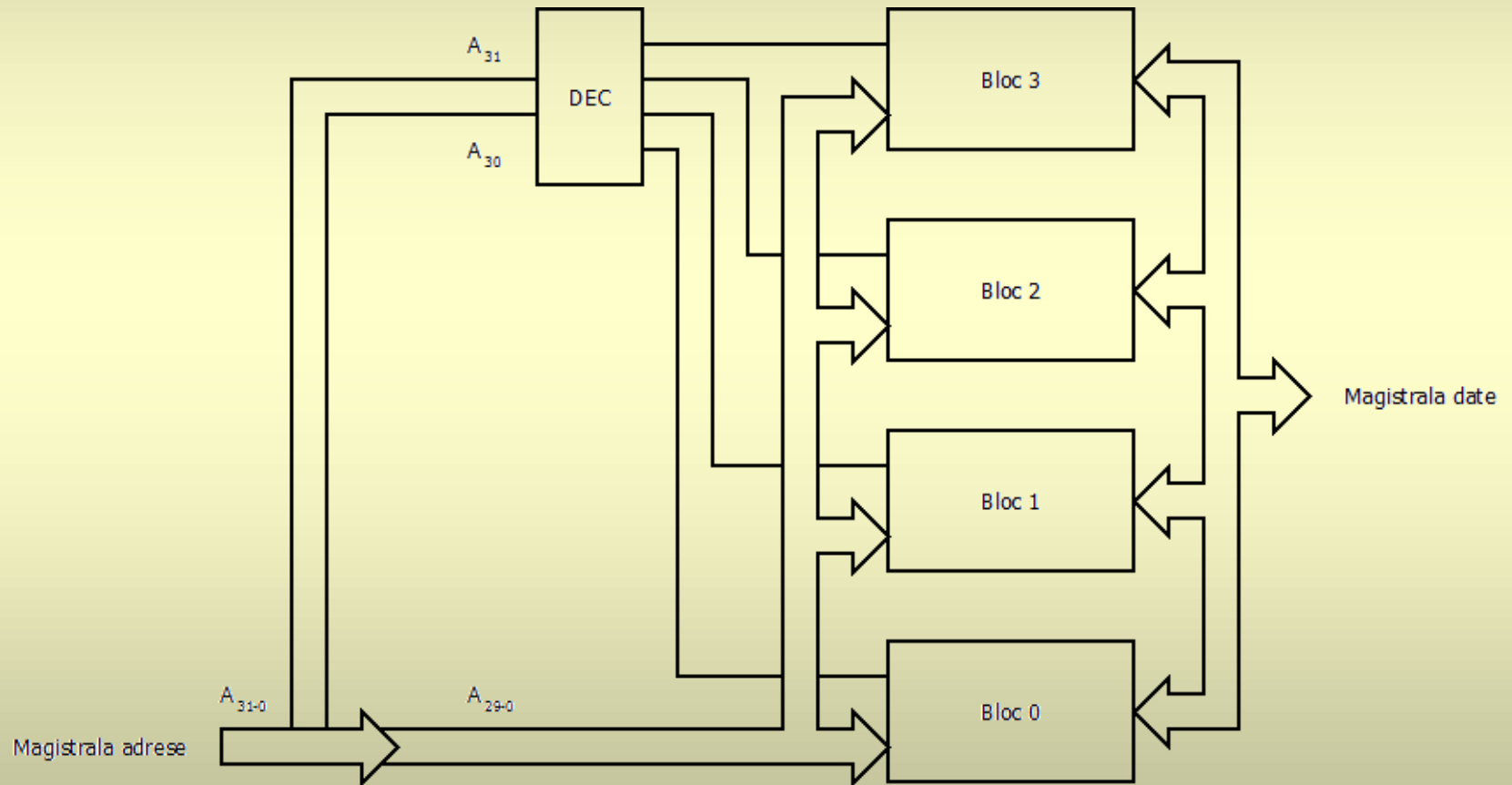
- limitat de capacitatea magistralei de adrese
  - deci este un element al arhitecturii calculatorului
  - exemplu: procesoare pe 32 biți - cel mult 4 GB ( $=2^{32}$ )
- nu se poate conecta memorie cu o capacitate mai mare decât maximul permis de magistrala de adrese
  - dar se poate mai puțin

# Decodificarea adreselor

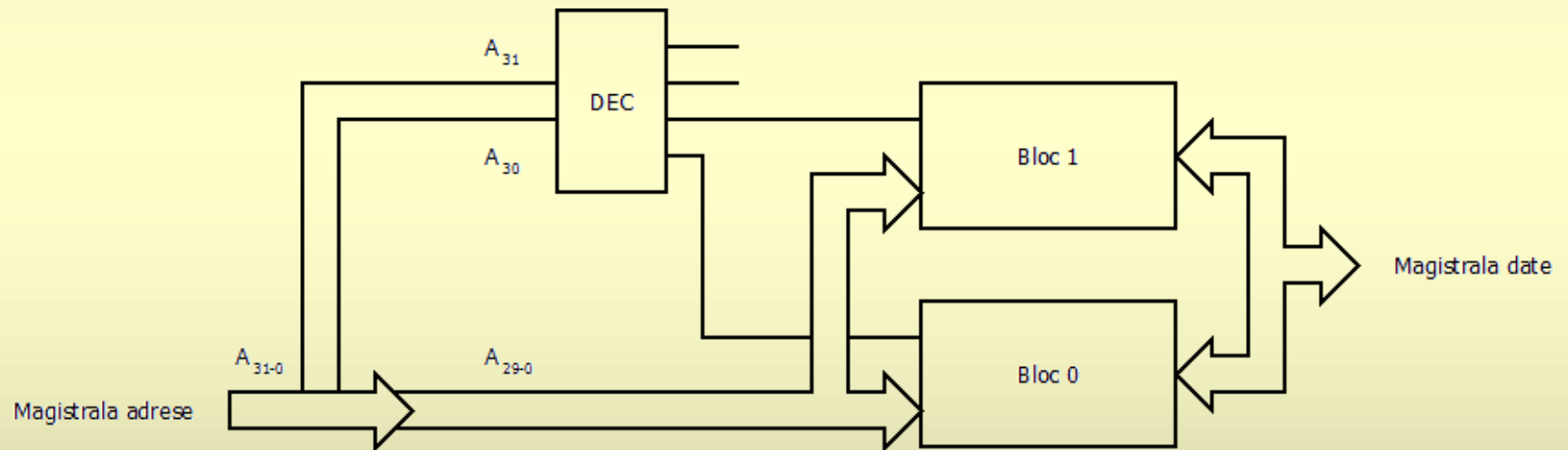
- exemplu: procesor pe 32 biți
- avem 4 circuite (blocuri) de memorie de câte 1 GB fiecare
  - total 4 GB - maximum posibil
- pentru o adresă dată, cum este selectat circuitul potrivit de memorie?
- dar dacă avem doar 2 circuite de câte 1 GB?



# Decodificare totală



# Decodificare parțială



# Tipuri de adrese

- absolută
  - numărul de ordine asociat unei celule
  - numerotarea începe de la 0
- relativă
  - poziția față de un octet de referință
  - exemplu: indicele unui element într-un tablou
- simbolică
  - identificator alfanumeric atașat unei adrese
  - exemplu: numele unei variabile

## Ordinea octeților (1)

- o variabilă poate ocupa mai multe locații de memorie
  - consecutive
- exemplu: variabilă de tipul *unsigned int*
  - ocupă 4 octeți
  - presupunem că adresele ocupate sunt 150÷153
  - care dintre octeți se găsește la adresa cea mai mică? dar la cea mai mare?

## Ordinea octeților (2)

- decizia este luată la proiectarea unității de procesare (CPU)
  - care realizează citirile și scrierile în memorie
- variante
  - *little endian* - octetul cel mai puțin semnificativ la adresa cea mai mică
  - *big endian* - octetul cel mai semnificativ la adresa cea mai mică

# Exemplu

unsigned int x = 0xB67A49E3; // B67A49E3<sub>(16)</sub>

*little endian*

*big endian*

adresă

valoare

adresă

valoare

	...
153	B6 <sub>(16)</sub> = 10110110 <sub>(2)</sub>
152	7A <sub>(16)</sub> = 01111010 <sub>(2)</sub>
151	49 <sub>(16)</sub> = 01001001 <sub>(2)</sub>
150	E3 <sub>(16)</sub> = 11100011 <sub>(2)</sub>
	...

	...
153	E3 <sub>(16)</sub> = 11100011 <sub>(2)</sub>
152	49 <sub>(16)</sub> = 01001001 <sub>(2)</sub>
151	7A <sub>(16)</sub> = 01111010 <sub>(2)</sub>
150	B6 <sub>(16)</sub> = 10110110 <sub>(2)</sub>
	...