V.2.
ARHITECTURA CALCULATORULUI: VIZIUNI POSIBILE

Arhitectura calculatorului (Computer Architecture)

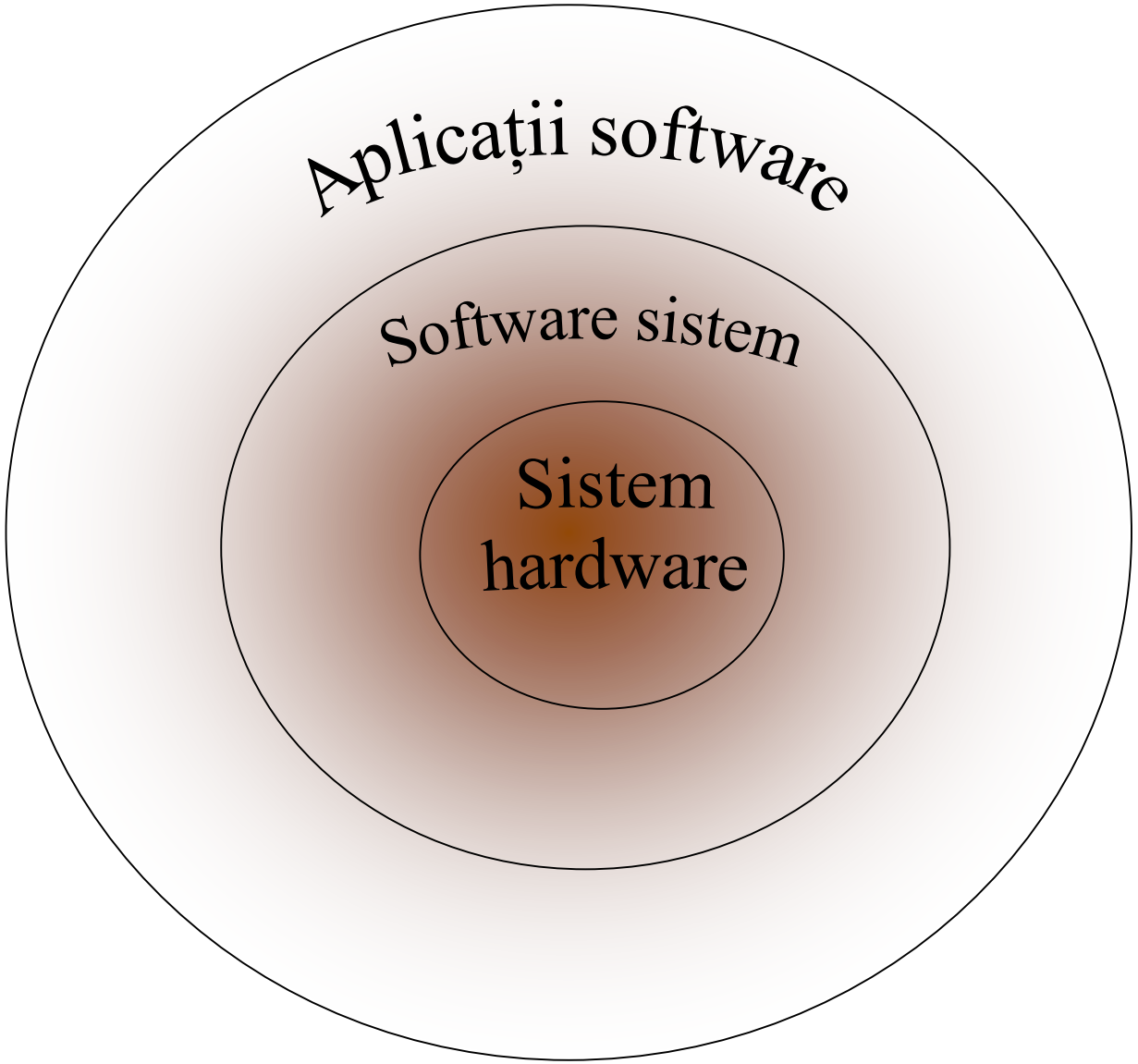
- Ce este arhitectura unui calculator?

Arhitectura calculatorului = **Arhitectura setului de instrucțiuni (ISA) + organizarea maşinii**

Arhitectura setului de instrucțiuni = **Viziunea programatorului asupra maşinii**

V.2.1.
PROGRAMATOR ŞI UTILIZATOR

Viziunea utilizatorului



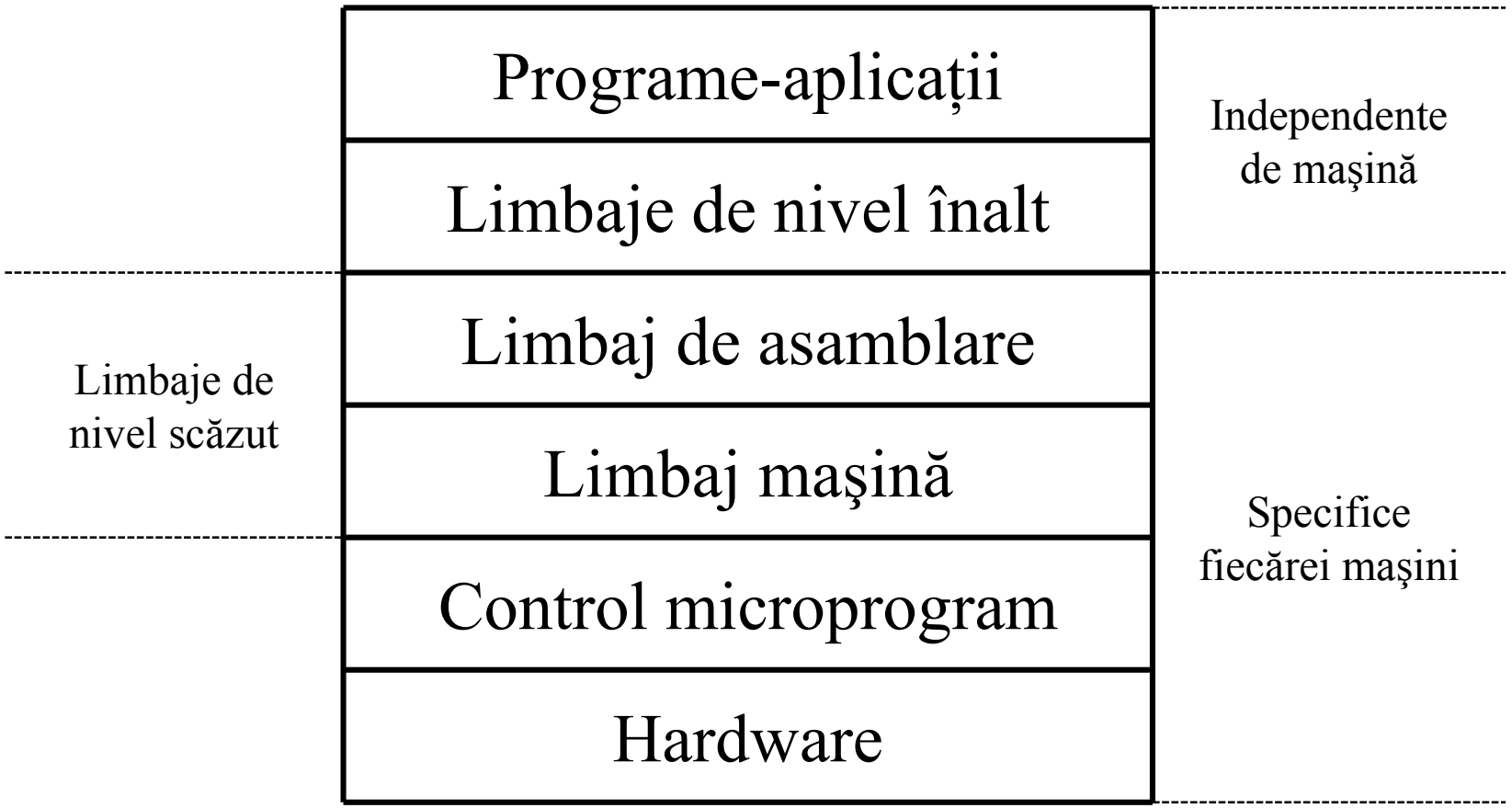
Viziunea programatorului

- Depinde de tipul şi nivelul limbajului utilizat
- O ierarhie a limbajelor:
  - Limbajul maşină
  - Limbajul de asamblare
  - Limbaje de nivel înalt
    - Programe - aplicații

creşterea nivelului de abstracție

- Independente de maşină:
  - Limbaje de nivel înalt / programe - aplicații
- Specifice maşinii:
  - Limbajele maşină şi de asamblare

Viziunea programatorului



Viziunea programatorului

- Limbajul maşină
  - caracteristic fiecărui procesor
  - constă din cuvinte peste alfabetul {0, 1}

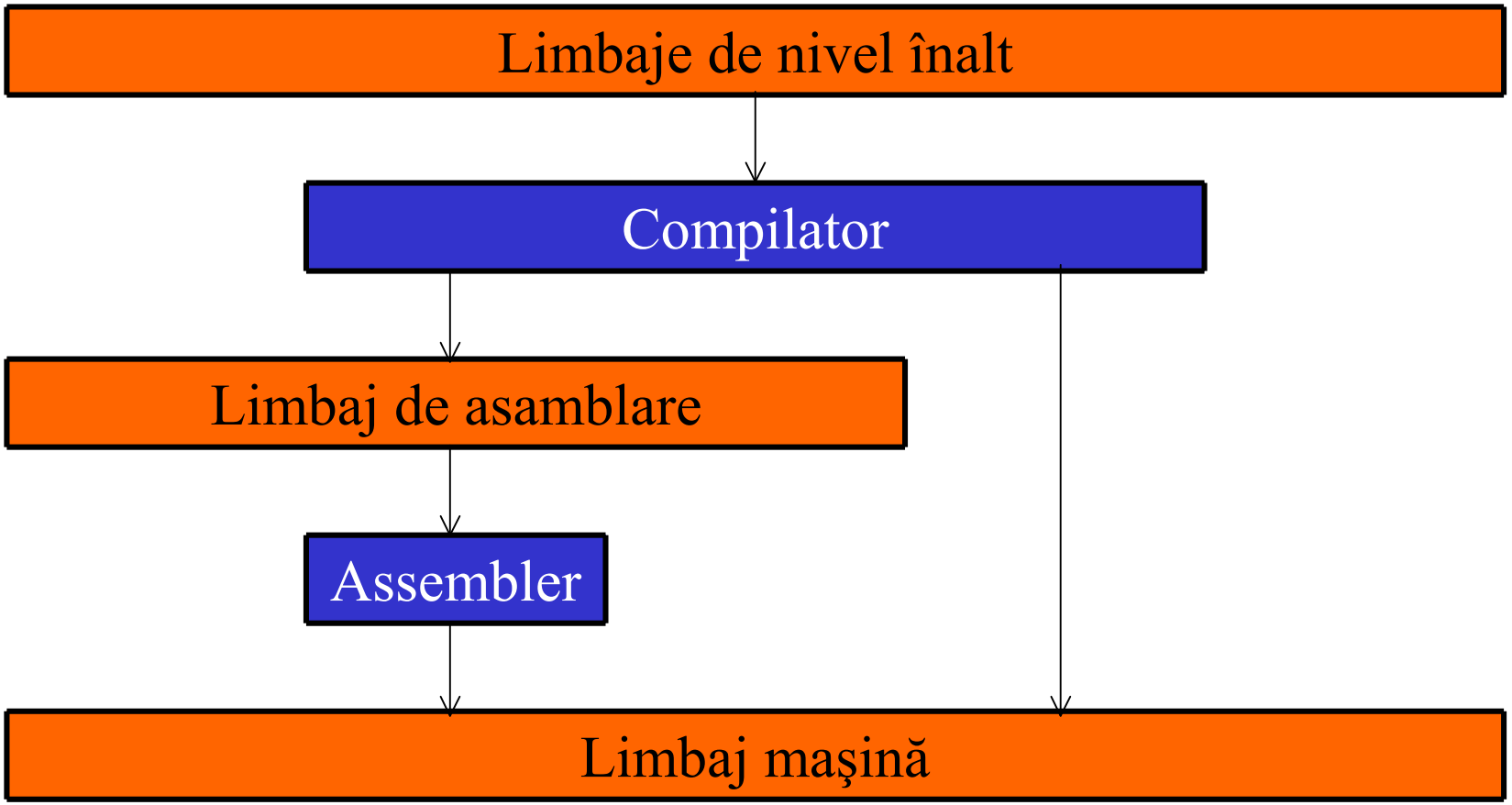
  1111 1111 0000 0110 0000 1010 0000 0000
- Limbajul de asamblare
  - nivel ceva mai înalt
    - mnemonice, adrese simbolice
  - mai apropiat de modul de înțelegere al omului
  - corespondență 1-la-1 cu majoritatea instrucțiunilor din limbajul maşină

  inc count

Viziunea programatorului

- **Asambloarele** (assemblers) traduc din limbaj de asamblare în limbaj maşină
  - dual: dez-asamblare
- **Compilatoarele** traduc din limbaje de nivel înalt în limbaj maşină
  - direct sau
  - indirect, via limbaj de asamblare
    - interpretoare

Viziunea programatorului



Viziunea programatorului

- Arhitectura setului de instrucțiuni (Instruction Set Architecture - ISA)
  - specifică funcționarea unui procesor
    - defineşte nivelul logic al procesorului
- O ISA poate fi implementată fizic în diverse feluri, care sunt
  - identice la nivel logic (interpretarea funcționării)
  - pot diferi ca
    - performanță (viteză)
    - preț

Viziunea programatorului: avantajele limbajelor de nivel înalt

- Dezvoltarea de programe este mai rapidă
  - Instrucțiuni de nivel înalt
    - Mai puține instrucțiuni de scris
- Întreținerea programelor este mai uşoară
  - Aceleaşi motive ca mai sus
- Programele sunt portabile
  - Conțin puține detalii dependente de maşină
    - Se pot folosi cu mici modificări sau chiar fără pe diverse tipuri de maşini
  - Traducerea în limbajul-maşină țintă urmează să fie făcută **automat** de un compilator specific calculatorului
    - Programele în limbaj de asamblare nu sunt portabile

Viziunea programatorului: La ce ajută programarea în limbaj de asamblare?

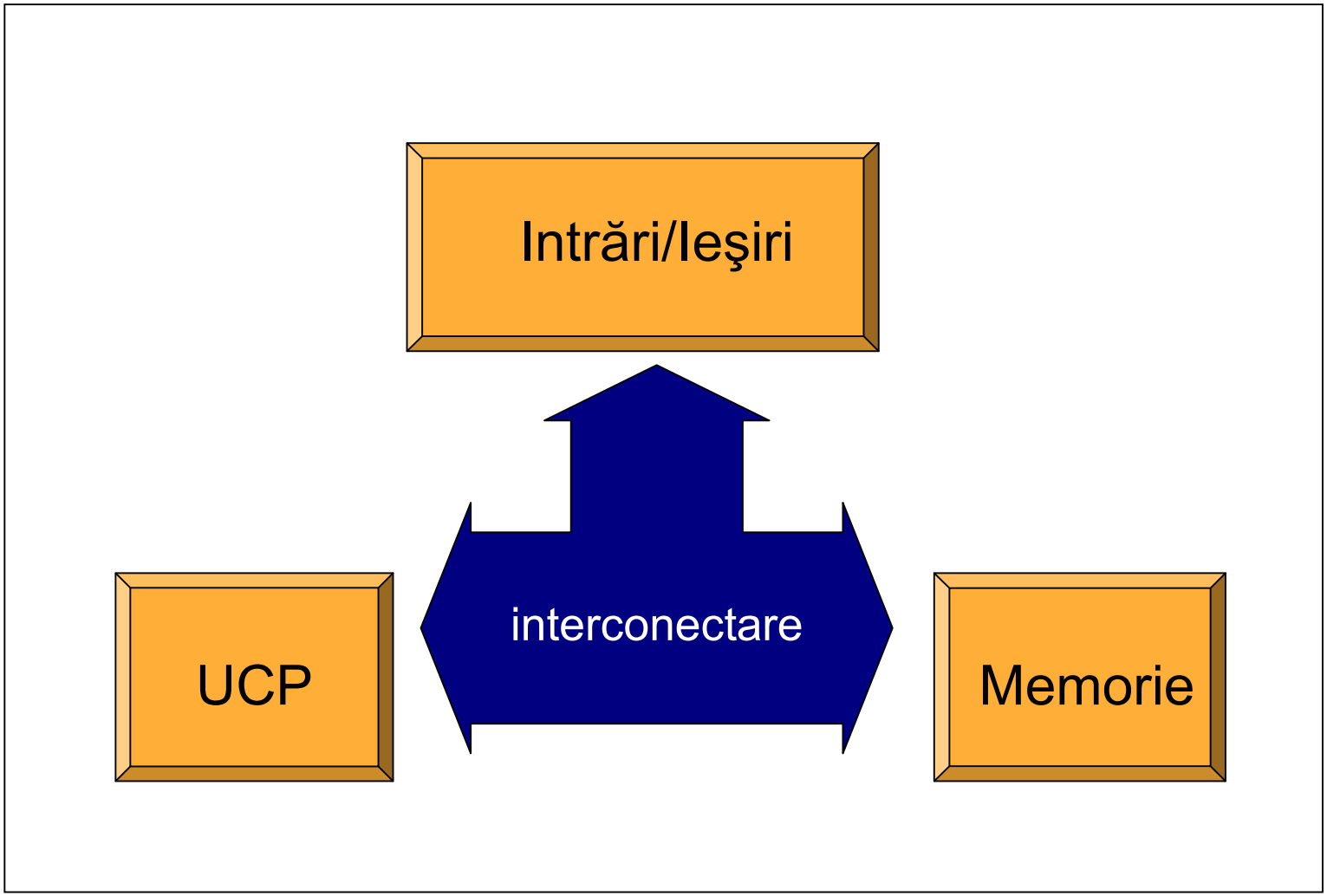
- Două mari avantaje:
  - **eficiență**
    - spațiu
      - » cod compact
      - » şi din limbaj de nivel înalt se ajunge la programe în limbaj-maşină, dar compactitate mai mică
    - timp
      - » legile localizării cu mai puține excepții şi pe ferestre mai mici, deci execuție mai rapidă (mai puține eşecuri)
  - **accesibilitate** la resursele hardware ale sistemului
    - » deoarece specificitățile maşinii sunt luate în considerare

V.2.2.
ARHITECT ŞI IMPLEMENTATOR

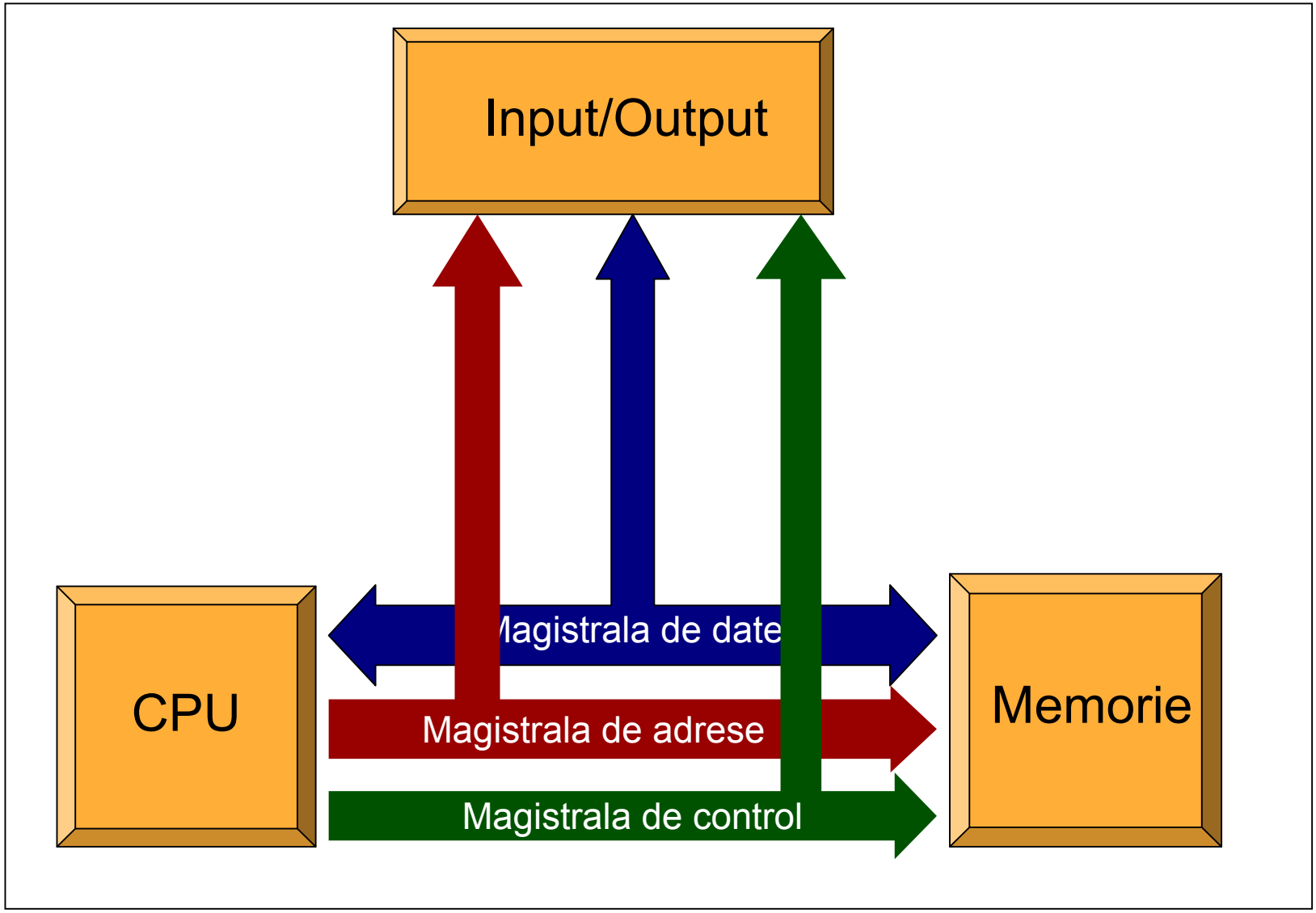
Viziunea arhitectului

- Proiectarea la nivel înalt
  - » ca şi arhitectul unei clădiri
  - Foloseşte "blocuri de construcție" de nivel înalt, pe care nu le detaliază până la ultimul nivel
    - » de exemplu, Unitatea Aritmetică şi Logică (UAL)
- Vede trei mari componente:
  - Procesorul
  - Memoria
  - Dispozitivele de intrare – ieşire (I/O devices)
- Legate prin interconexiuni (magistrala)

Viziunea arhitectului



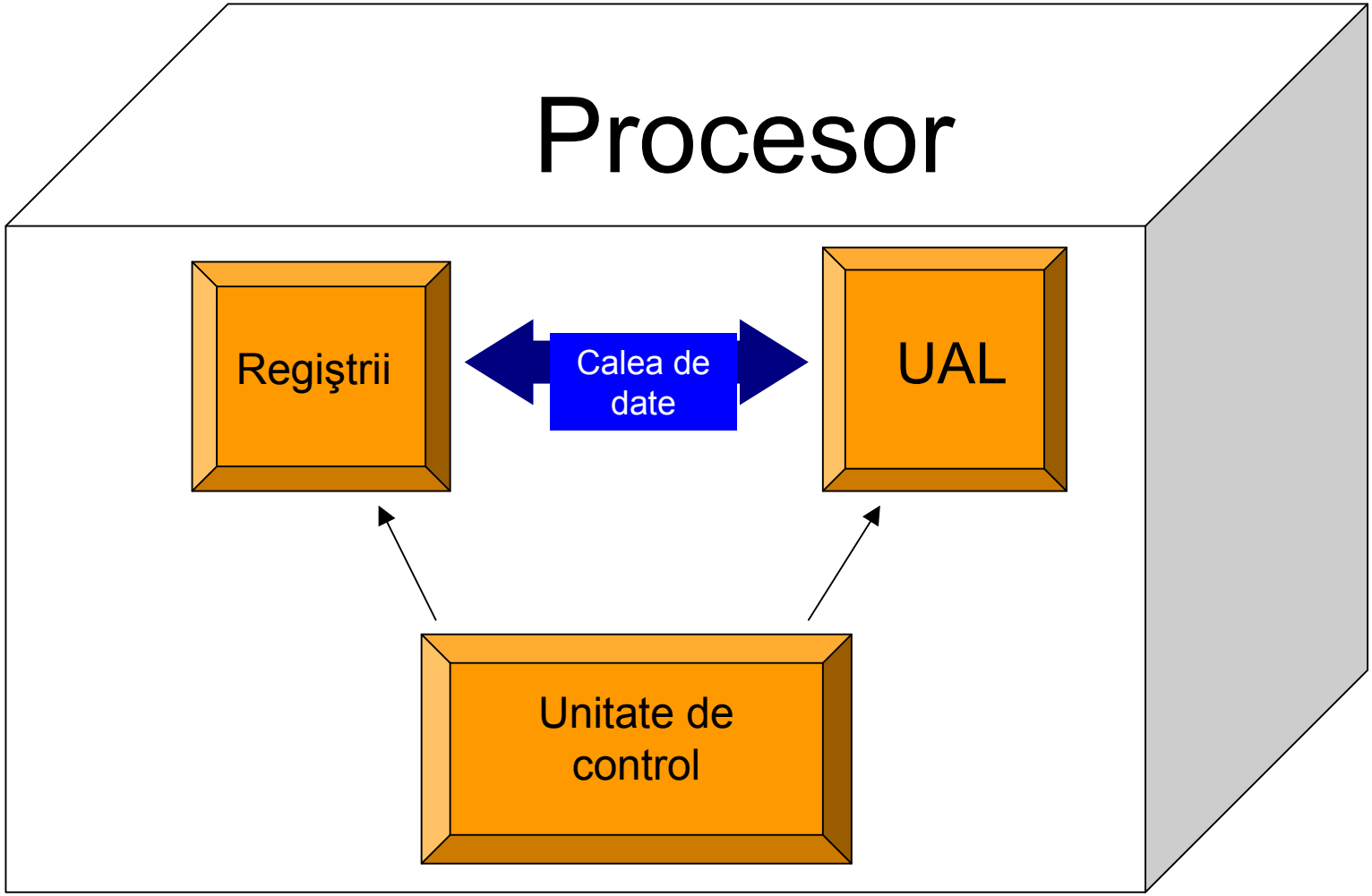
Viziunea arhitectului



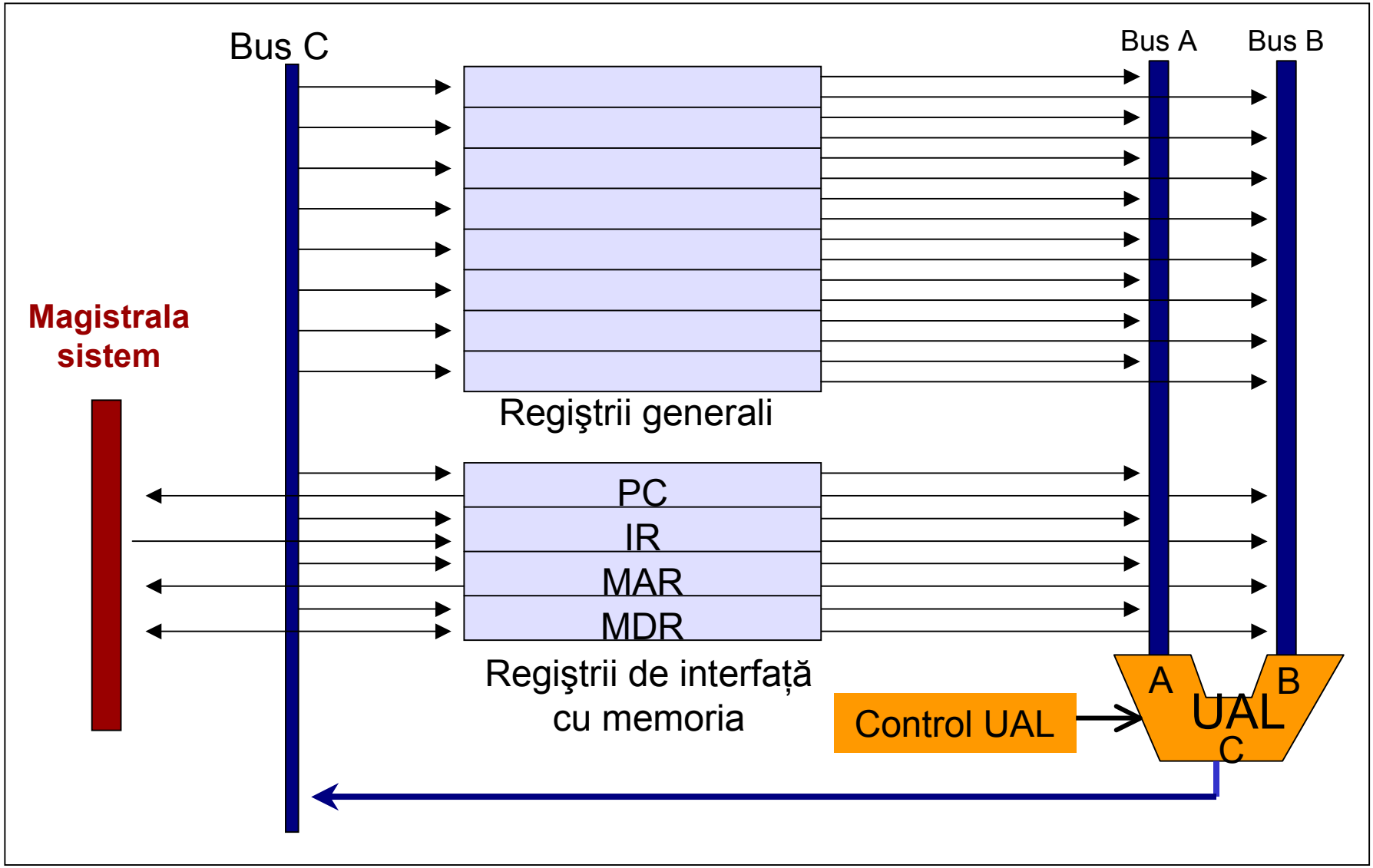
Viziunea implementatorului

- De la proiectul arhitectului coboară la nivelul porților logice digitale / al circuitelor
  - Exemplu: Procesorul constă din:
    - Unitatea de control
    - Calea de date (datapath)
      - » UAL
      - » regiştri
- Implementarea: proiectarea acestor componente (de nivel mai jos)

Viziunea implementatorului



Viziunea implementatorului – calea de date



Sensul transferurilor de date

- În procesor: regiştri $\rightarrow$ UAL $\rightarrow$ regiştri
- Între componente: memorie $\leftrightarrow$ regiştri

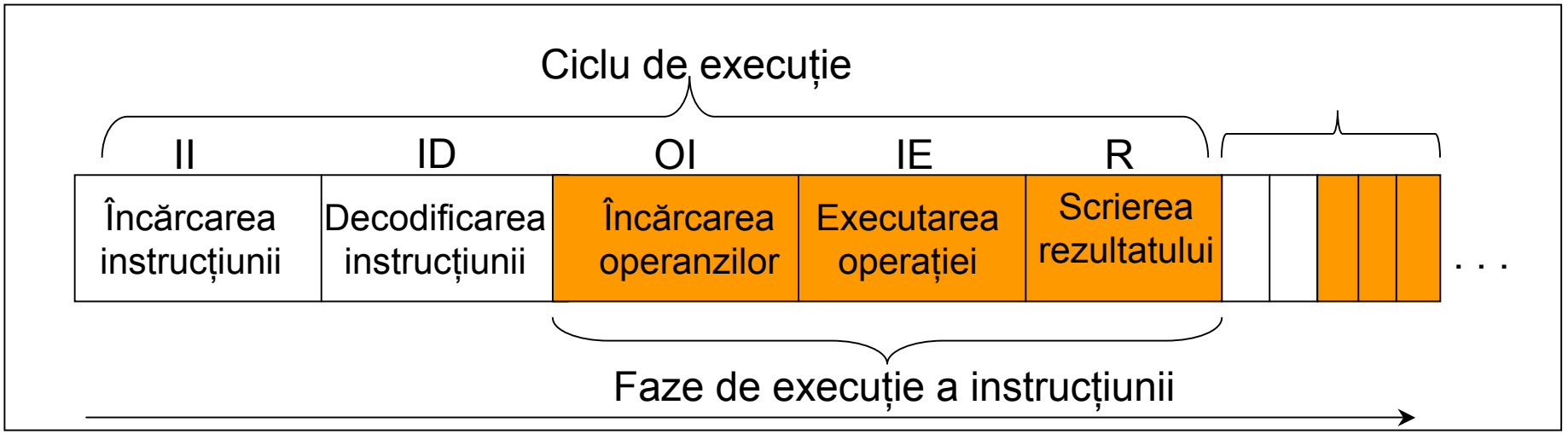
- Orice dată se prelucrează trecând prin regiştri
- Unele căi sunt asimetrice

V.3.
SCURTĂ TRECERE ÎN REVISTĂ A ORGANIZĂRII CALCULATORULUI

V.3.1.
PROCESORUL

Procesorul

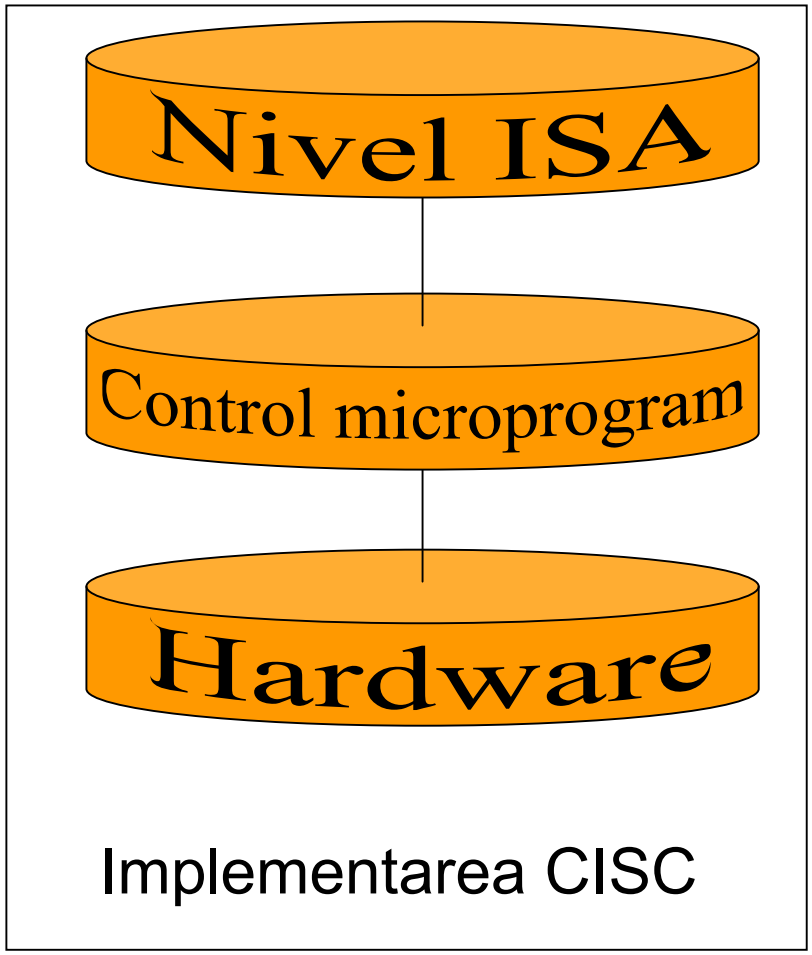
- Ciclul de execuție
  - Fetch (căutare şi încărcare)
  - Decodificare
  - Execuție operație
- Arhitectura von Neumann
  - Modelul programului memorat
    - Nici o deosebire la nivel fizic între datele de prelucrat şi instrucțiunile care le prelucrează
    - Deosebirile apar la execuție
    - Instrucțiunile se execută secvențial

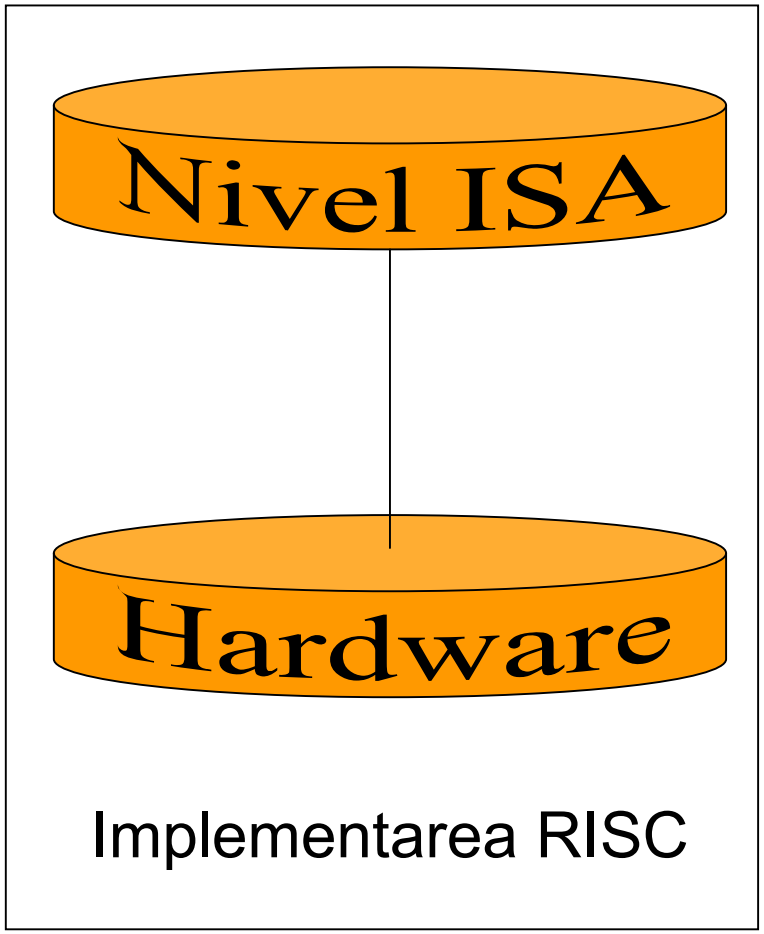


Procesorul

- Complex Instruction Set Computer (CISC)
  - Foloseşte instrucţiuni-maşină complexe
  - Operanzii pot fi în regiştrii procesorului sau în memorie
    - » Lungimea instrucţiunilor nu este fixă
  - De obicei se foloseşte **microprogramarea**
    - » O **instrucţiune** în limbaj maşină devine un **program** scris cu instrucţiuni şi mai simple
    - » Abia acestea din urmă sunt executate prin construcţie (**cablate**) şi nu ca un program
- îmbunătăţiri
  - Pipelining
    - Descompunerea în mai mulţi paşi a execuţiei fiecărei instrucţiuni
    - Execuţie simultană, pe paşi, a mai multor instrucţiuni succesive decalate
  - Reduced Instruction Set Computer (RISC)
    - Are instrucţiuni-maşină simple
    - Instrucţiunile mai complicate se obţin din cele simple ca secvenţe

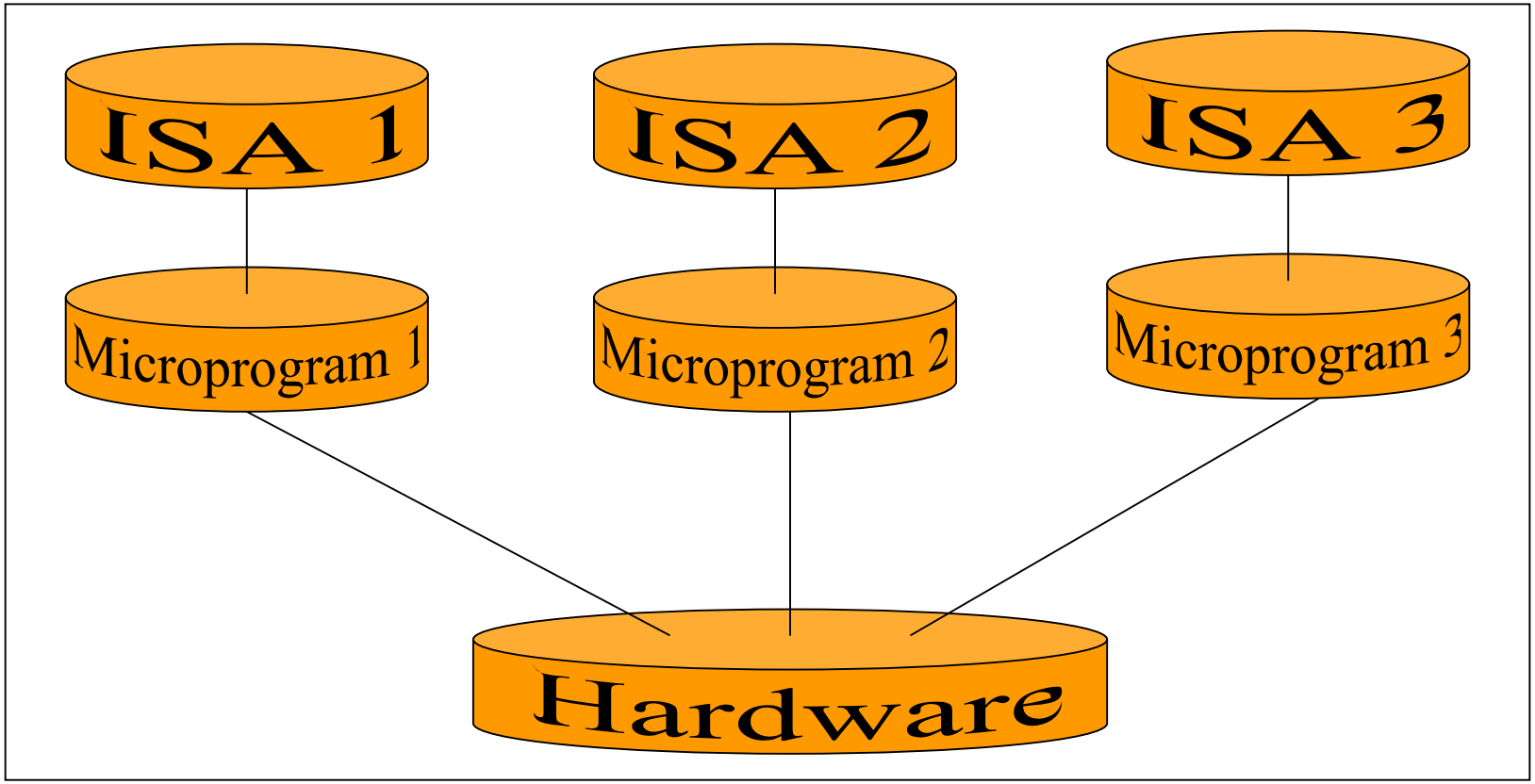
Procesorul


Nivel ISA
Control microprogram
Hardware
Implementarea CISC


Nivel ISA
Hardware
Implementarea RISC

Procesorul

- Microprogramare: dacă apar modificări la nivelul ISA, acestea se pot implementa schimbând doar microprogramul (nu circuitele)



V.3.2.
MEMORIA

Memoria

- Şir ordonat de **octeți** (nivel logic)
  - Fiecare octet are un predecesor (cu excepția primului) şi un succesor (cu excepția ultimului)
  - Nivel fizic: matrice de matrici; piste concentrice
- ***Adresa absolută*** a unui octet: numărul său de ordine din şir
  - memorie adresabilă la nivel de octet
    - fiecare octet are o adresă unică
  - numerotarea începe de la 0
  - adresa absolută indică poziția față de primul octet din memorie

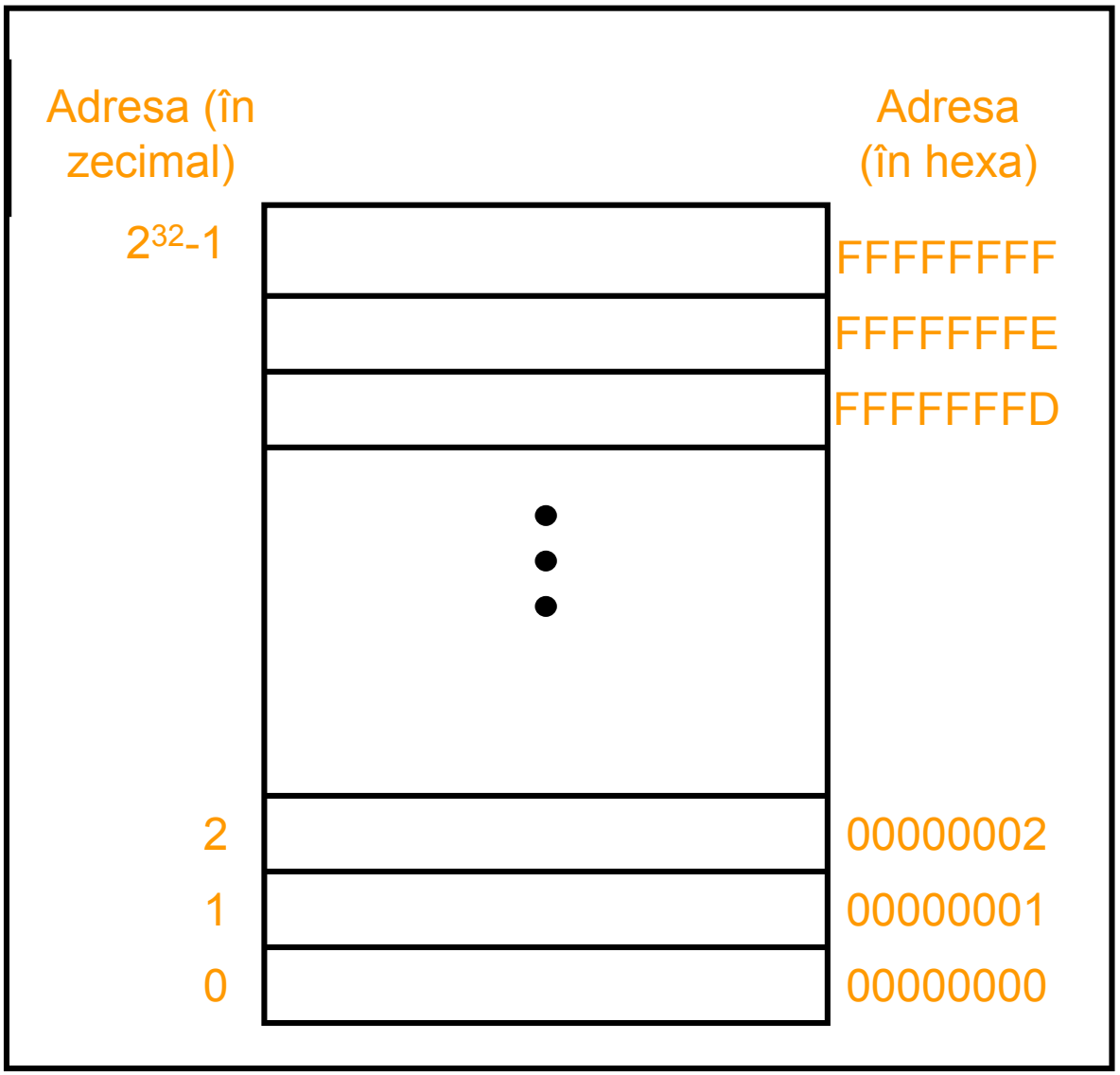
Alte tipuri de adrese

- ***Adresa relativă*** a unui octet: indică poziția sa față de un octet de referință, altul decât primul octet din memorie
  - Adresa absolută a octetului de referință se numeşte *adresă de bază*
  - De regulă, de la adresa de bază începe zona rezervată programului / datelor din care face parte octetul indicat prin adresă relativă
- ***Adresă simbolică***: identificator alfanumeric ("nume") ataşat adresei relative a unui octet
  - Exemplu: numele variabilelor
  - Corespondența adrese simbolice – adrese relative este făcută de compilator, printr-o tabelă

Spațiul adreselor de memorie

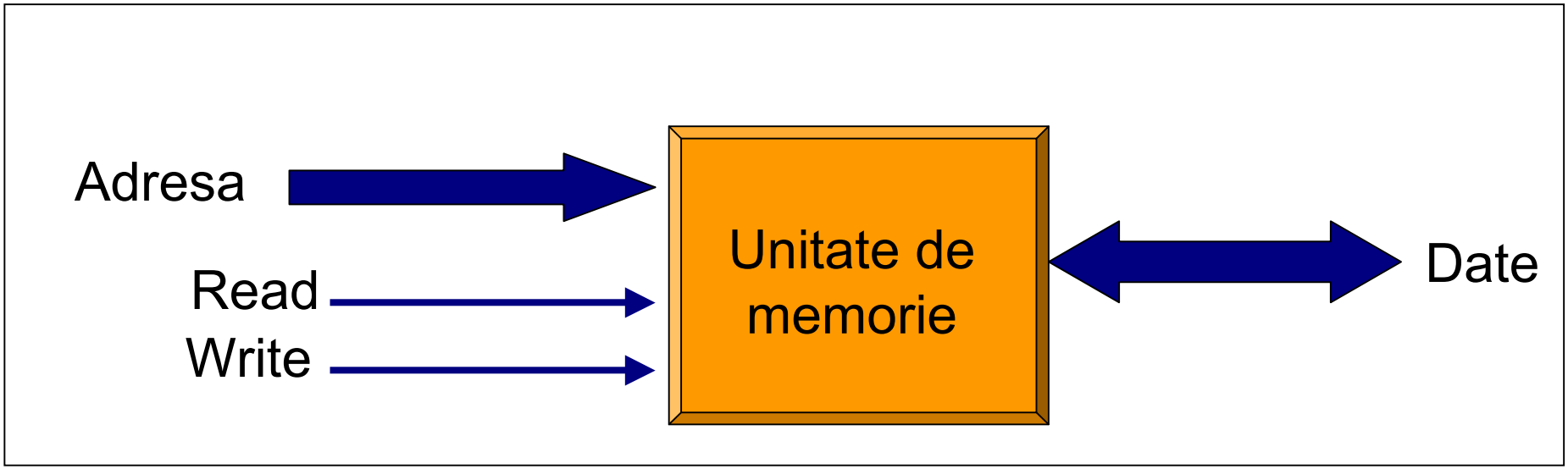
- Element esențial al arhitecturii unui calculator
- Dimensiunea sa este strâns legată de capacitatea magistralei de adrese
- La Pentium, capacitatea magistralei de adrese este de 32 de biți
  - Spațiul de adrese ≤ 4GB ($\mathbf{2^{32}}$)

Memoria - adrese absolute



Memoria

- Unitatea de memorie - comunicare cu exteriorul
  - Adresare
  - Date
  - Semnale de control
    - Read
    - Write



Ciclul "citire memorie" (Read) – activitatea procesorului

1. Aducerea adresei de citire pe busul de adrese (MAR)
2. Emiterea semnalului de control "citire memorie" (UC)
3. Aşteptare până când memoria găseşte data de citit
   - E nevoie de starea de aşteptare (***wait state***) dacă memoria utilizată este înceată
4. Citirea datei de pe busul de date (MDR)
5. Inactivarea semnalului de control "citire memorie" (UC)

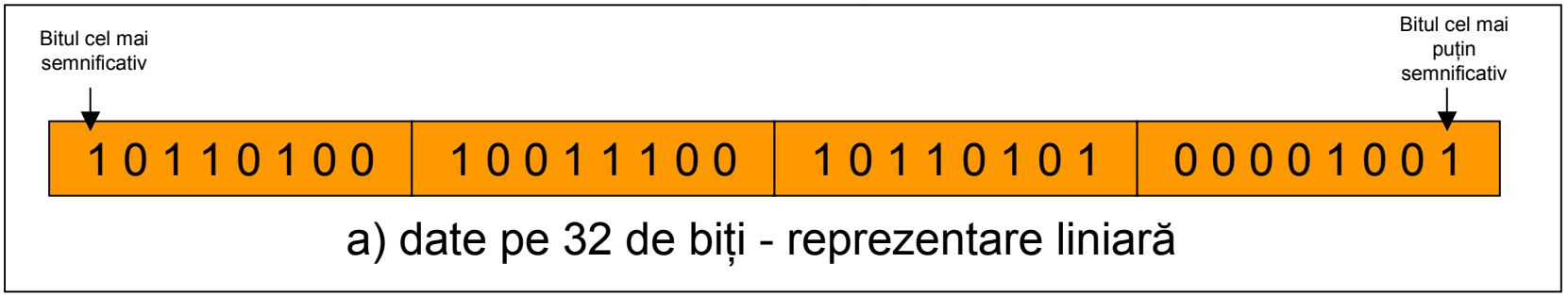
- Pentium: o citire necesită trei **cicluri de ceas** (tacte)
  - Clock 1: paşii 1 şi 2
  - Clock 2: pasul 3
  - Clock 3 : paşii 4 şi 5

Ciclul "scriere memorie" (Write) – activitatea procesorului

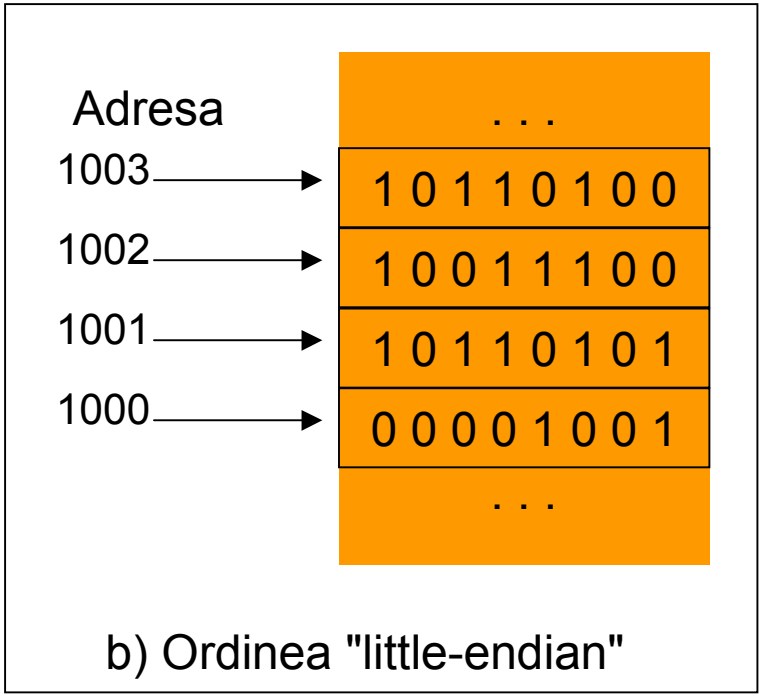
1. Aducerea adresei de scriere pe busul de adrese
2. Activarea semnalului de control "scriere memorie"
3. Aducerea datei de scris pe busul de date
4. Aşteptare până când memoria efectuează scrierea
   - starea de aşteptare (***wait state***) dacă memoria este înceată
5. Inactivarea semnalului de control "scriere memorie"

- La Pentium, o scriere necesită trei cicluri de ceas
  - Clock 1: paşii 1 şi 2
  - Clock 2: pasul 3
  - Clock 3 : paşii 4 şi 5

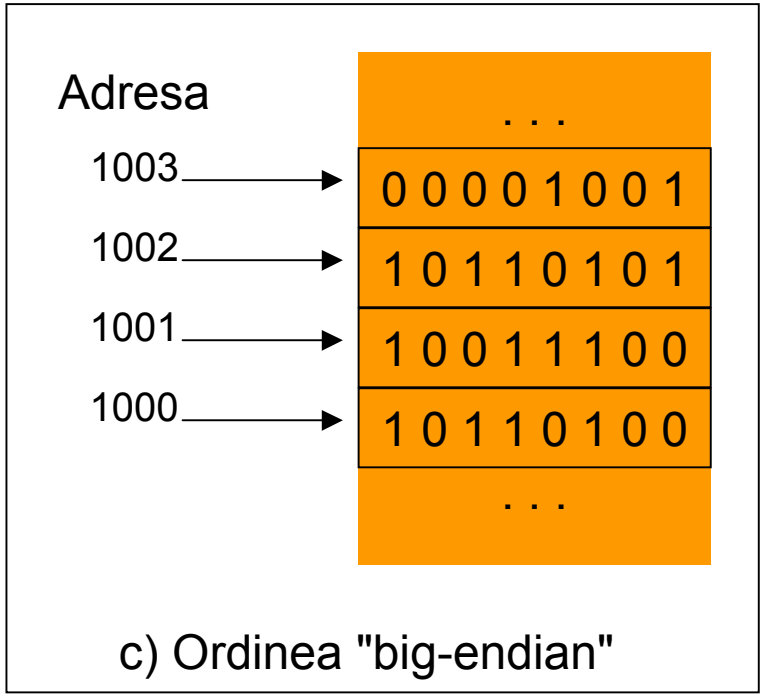
Ordinea octeților



a) date pe 32 de biți - reprezentare liniară



b) Ordinea "little-endian"



c) Ordinea "big-endian"

V.3.3.
INTRĂRI - IEŞIRI

Intrări-ieşiri (Input/Output)

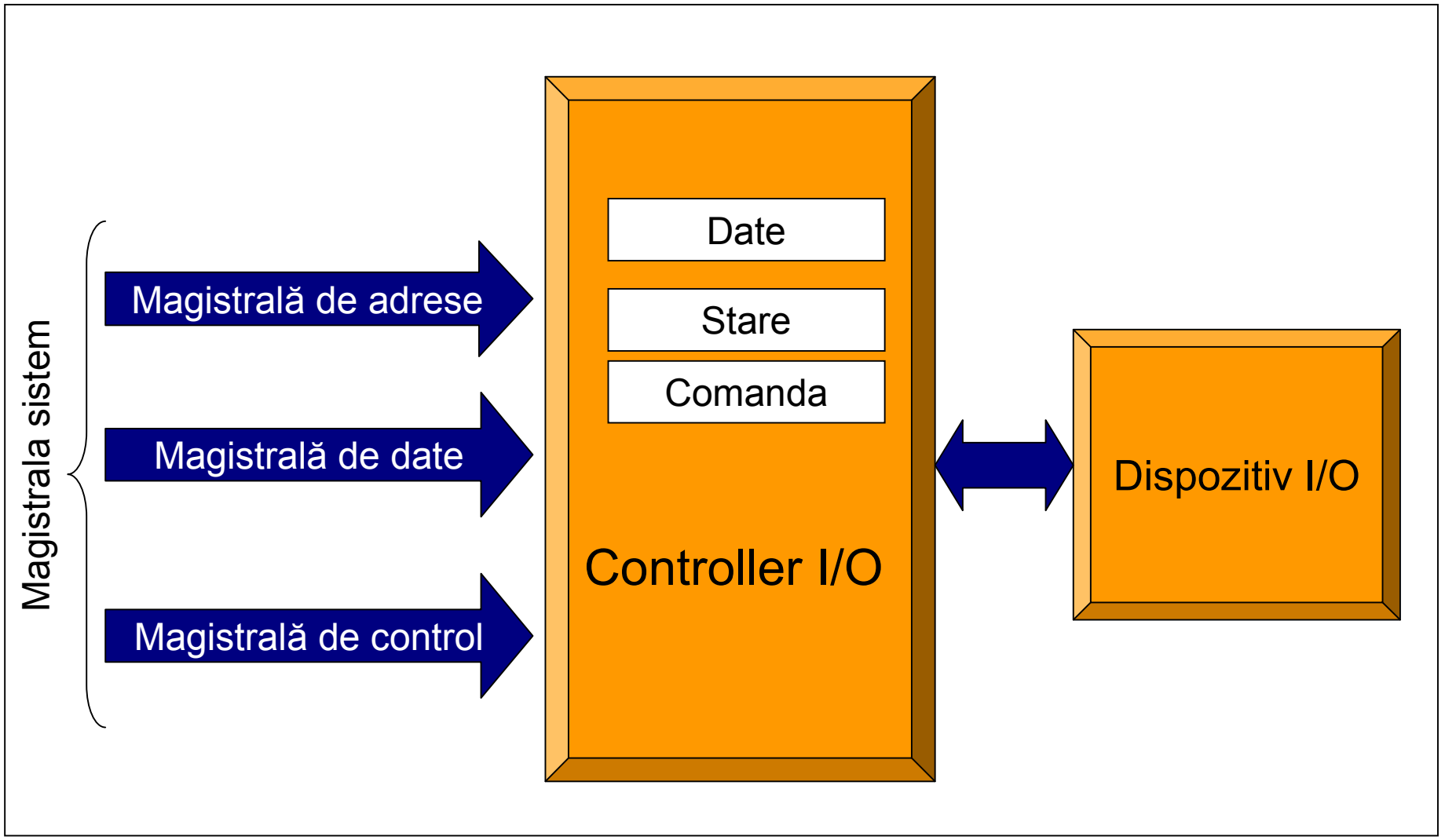
- Perifericele de intrare-ieşire (I/O devices) au interfața gestionată de câte un **I/O controller**

- Gestionează detalii legate de operațiile de nivel jos (periferic)

Intrări-ieşiri

- Mai multe feluri de a "vedea" I/O
  - I/O Proiectat pe memorie (memory-mapped I/O)
    - Citirile / scrierile sunt "văzute" ca şi când s-ar face din / în memorie
    - Aceleaşi semnale de control ca pentru citire/scriere din/în memorie
    - Aşa fac cele mai multe procesoare
  - I/O izolate
    - Spațiul de adrese pentru "locațiile" I/O diferă de cel al memoriei
    - Semnale de control R/W diferite de ale memoriei
    - Pentium foloseşte I/O izolate
      - dar şi I/O proiectate pe memorie

Comunicare cu perifericele



Modalități de transmitere date I/O

- Intrări-ieşiri programate (programmed I/O)
  - Programul foloseşte o buclă "busy-wait"
  - Se aplică transferurilor pentru care se ştie de la început momentul când vor apărea
- Acces direct la memorie (Direct Memory Access - DMA)
  - Un controller special (DMA controller) efectuează transferurile
- I/O gestionate prin întreruperi (Interrupt-driven I/O)
  - Întreruperile sunt folosite pentru a iniția şi / sau termina transferuri de date
    - Tehnică extrem de eficientă
    - Utilizată pentru transferuri despre care nu se ştie de la început când vor apărea

Interconexiuni

- Componentele unui sistem de calcul sunt interconectate prin **magistrale** (**bus-uri**)
  - Bus: mai multe fire pentru comunicații de date
- La diverse niveluri, se folosesc mai multe bus-uri
  - Magistrale "on-chip"
    - Interconectează UAL şi regiştrii
      - Magistralele A, B, şi C din exemplul de la pag. 20
    - Există bus de date şi bus de adrese – pentru legături cu memoria cache din *chip*
  - Magistrale interne
    - PCI, AGP, PCMCIA
  - Magistrale externe
    - Seriale, paralele, USB, IEEE 1394 (FireWire)
  - Conectori, porturi

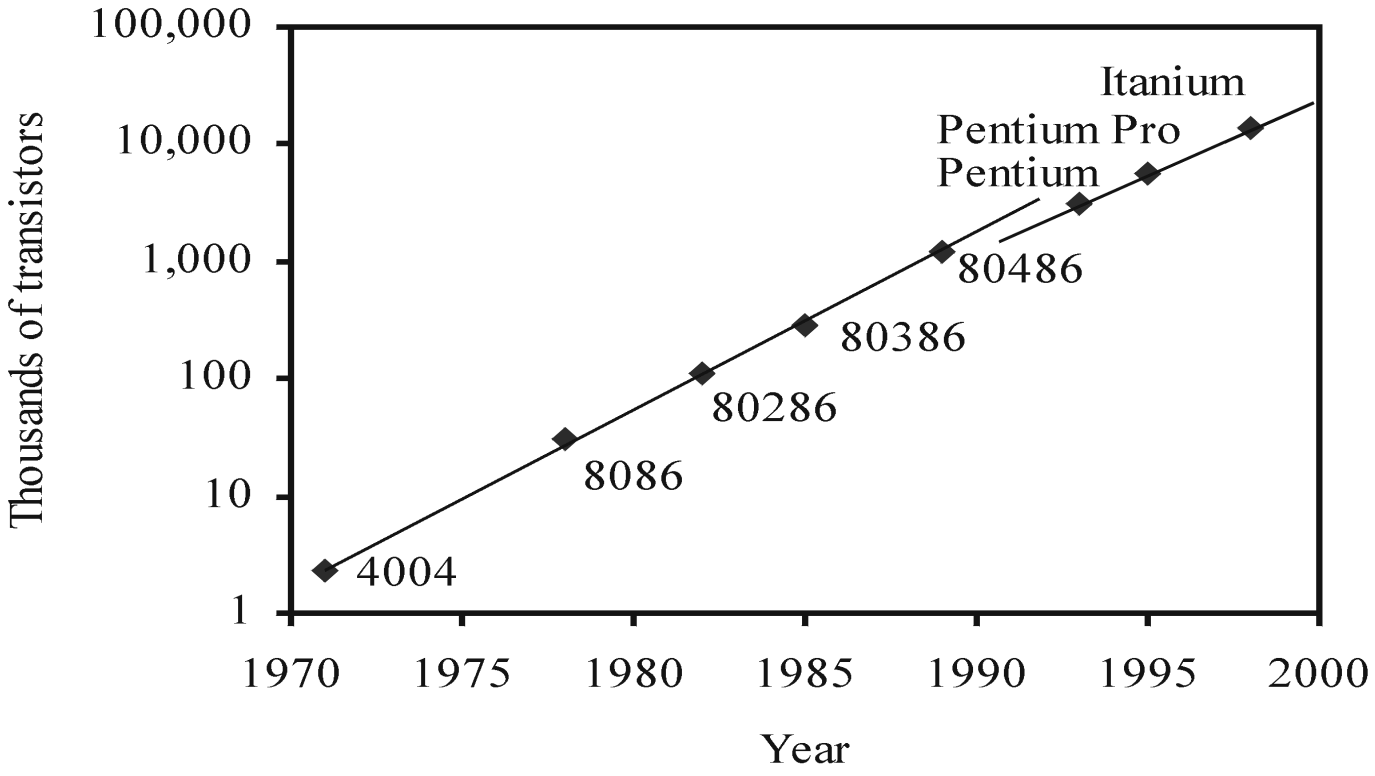
V.4.
O PERSPECTIVĂ ISTORICĂ

- Pre-generații
  - "Difference engine" a lui Charles Babbage
- Generația tuburilor cu vid
  - Anii 1940 - 1950
- Generația tranzistorilor
  - Aproximativ anii 1950 - 1960
- Generația circuitelor integrate
  - 1960 - 1970
- Generația VLSI
  - De la mijlocul anilor '70
  - ULSI etc.

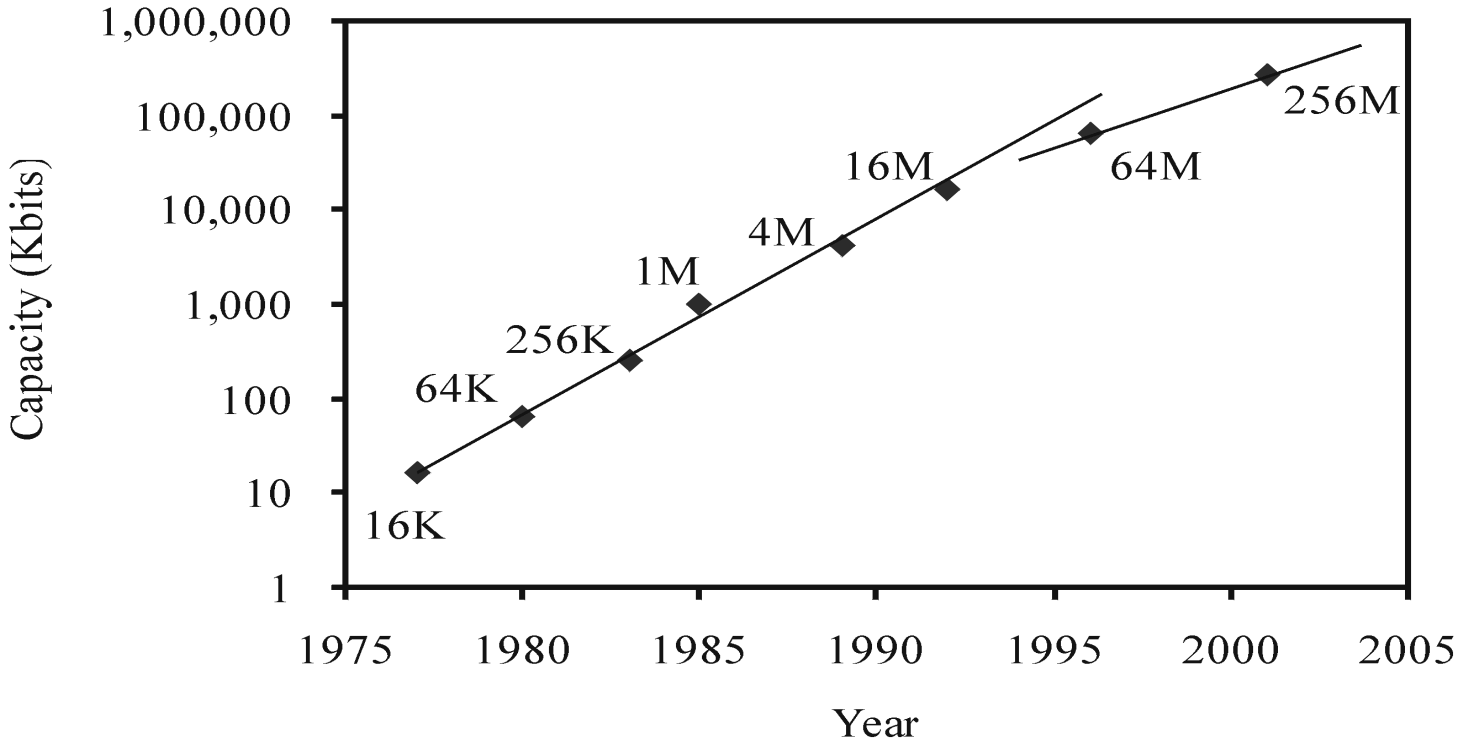
Evoluția tehnologică

- Densitatea tranzistorilor
  - Până prin 1990, se dubla la fiecare 18-24 luni
  - De atunci, dublare la fiecare 2 ani şi jumătate
- Densitatea memoriei
  - Până în 1990, quadruplare la fiecare 3 ani
  - De atunci, la fiecare 5 ani
- Capacitățile diverselor tipuri de discuri
  - La 3.5"
  - La 2.5"
  - La 1.8" (e.g., periferice portabile USB)

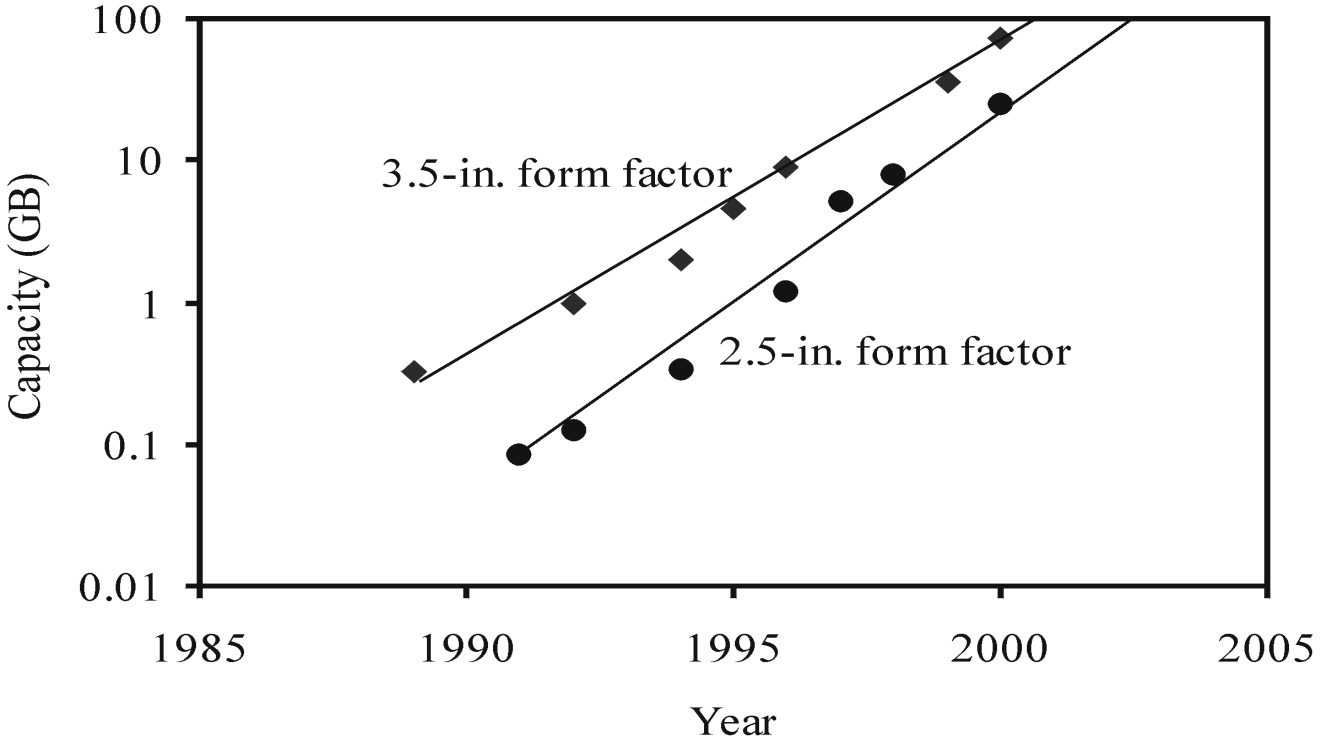
Evoluția tehnologică: integrare



Evoluția tehnologică (memorie)



Evoluția tehnologică (discuri)



Submultipli;
scara binară a multiplilor

(sub)multiplu	**zecimal**	**binar**
p (pico)	**10^{-12}**	**-**
n (nano)	**10^{-9}**	**-**
μ (micro)	**10^{-6}**	**-**
m (mili)	**10^{-3}**	**-**
Unitatea	**1 (=10^{0})**	**1 (=2^{0})**
K (kilo)	**(10^{3})**	**2^{10}**
M (mega)	**(10^{6})**	**2^{20}**
G (giga)	**(10^{9})**	**2^{30}**
T (tera)	**(10^{12})**	**2^{40}**
P (peta)	**(10^{15})**	**2^{50}**