

V.2.

ARCHITECTURA
CALCULATORULUI:
VIZIUNI POSSIBILE

Arhitectura calculatorului (Computer Architecture)

- Ce este arhitectura unui calculator?

**Arhitectura
calculatorului**

=

**Arhitectura setului
de instrucțiuni (ISA)
+
organizarea mașinii**

**Arhitectura setului
de instrucțiuni**

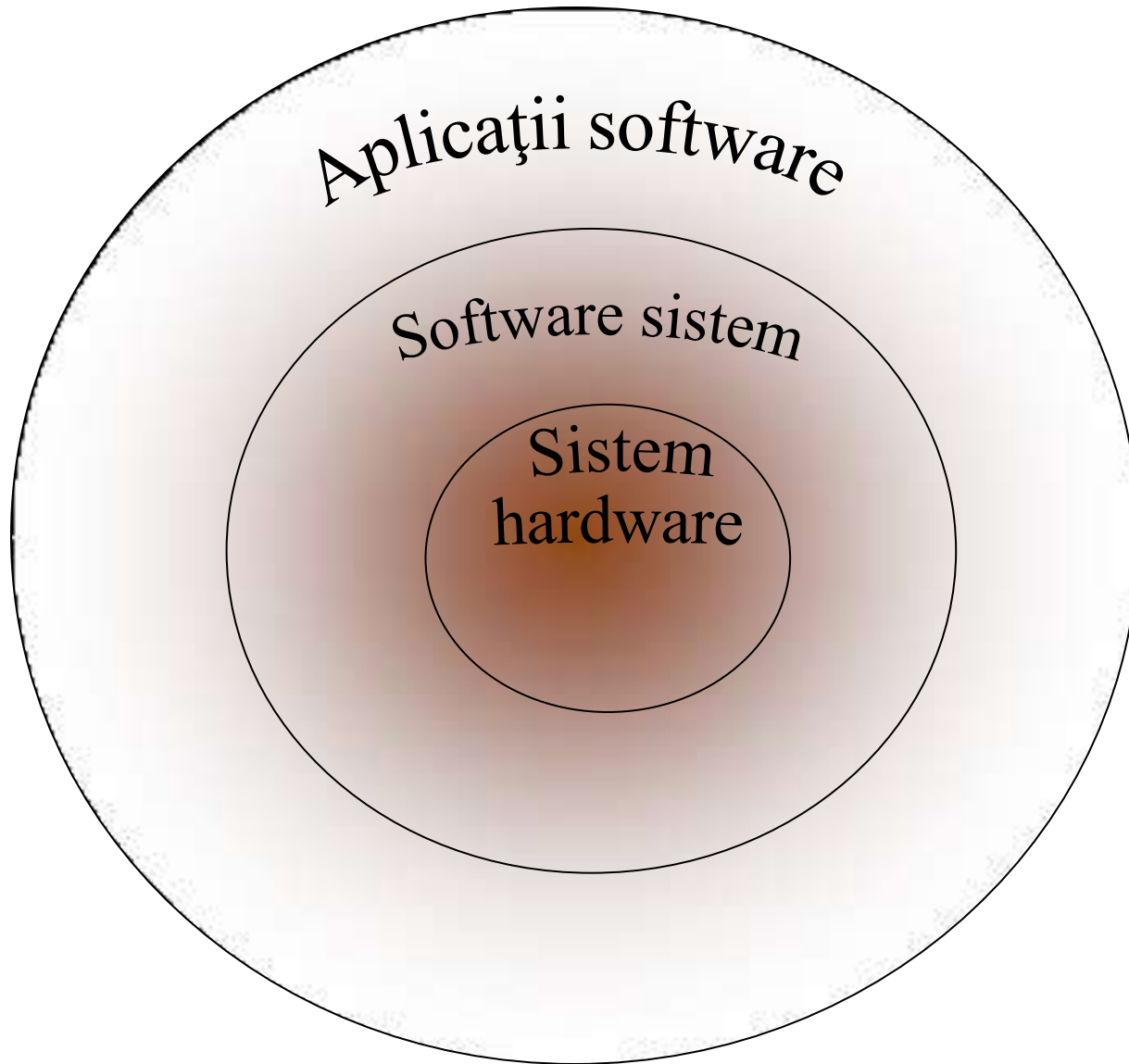
=

**Viziunea
programatorului
asupra mașinii**


V.2.1.

PROGRAMATOR ȘI UTILIZATOR

Viziunea utilizatorului

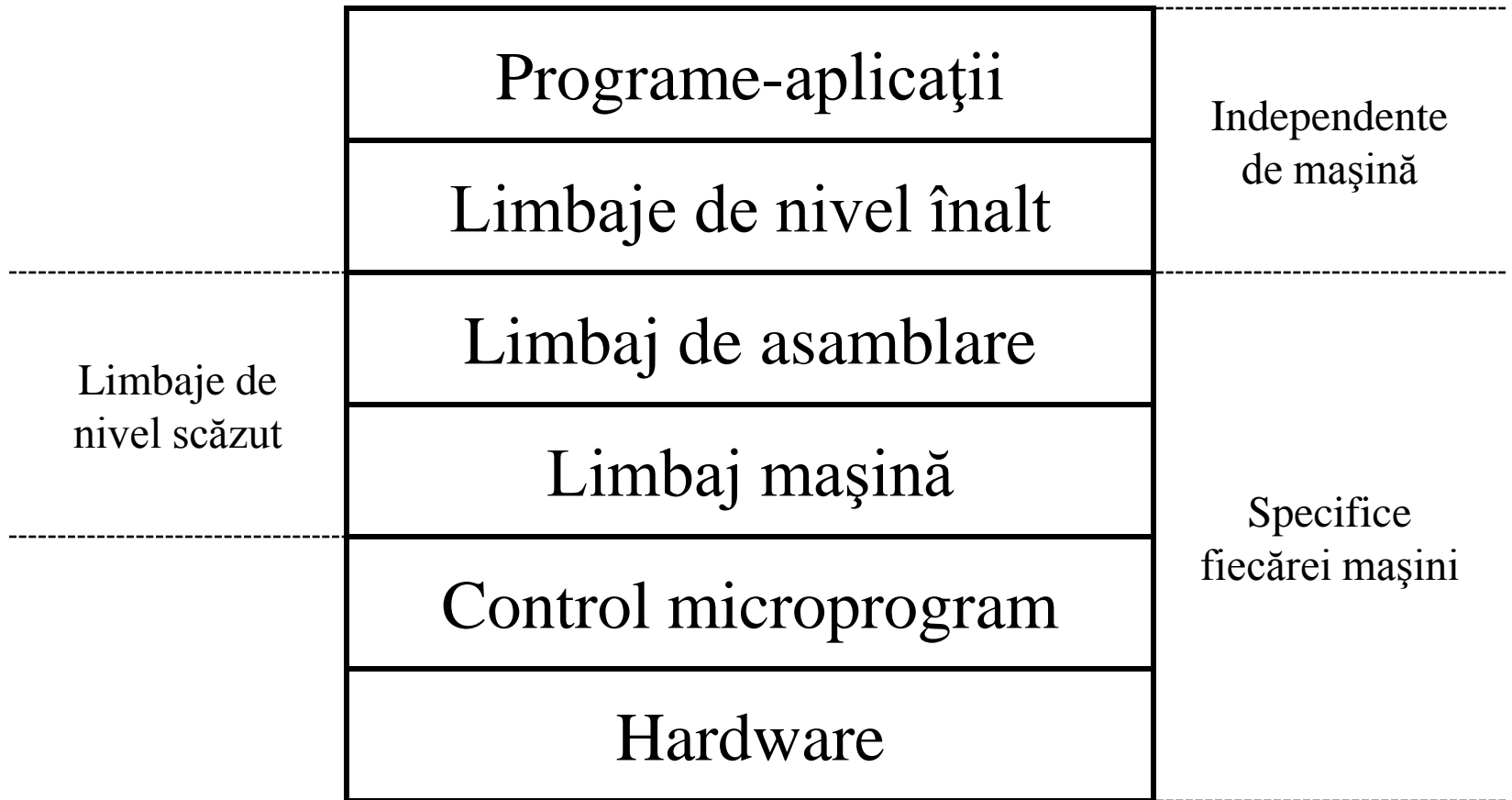


Viziunea programatorului

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

creșterea
nivelului
de abstracție
- Independente de mașină:
 - Limbaje de nivel înalt / programe - aplicații
- Specifice mașinii:
 - Limbajele mașină și de asamblare

Viziunea programatorului



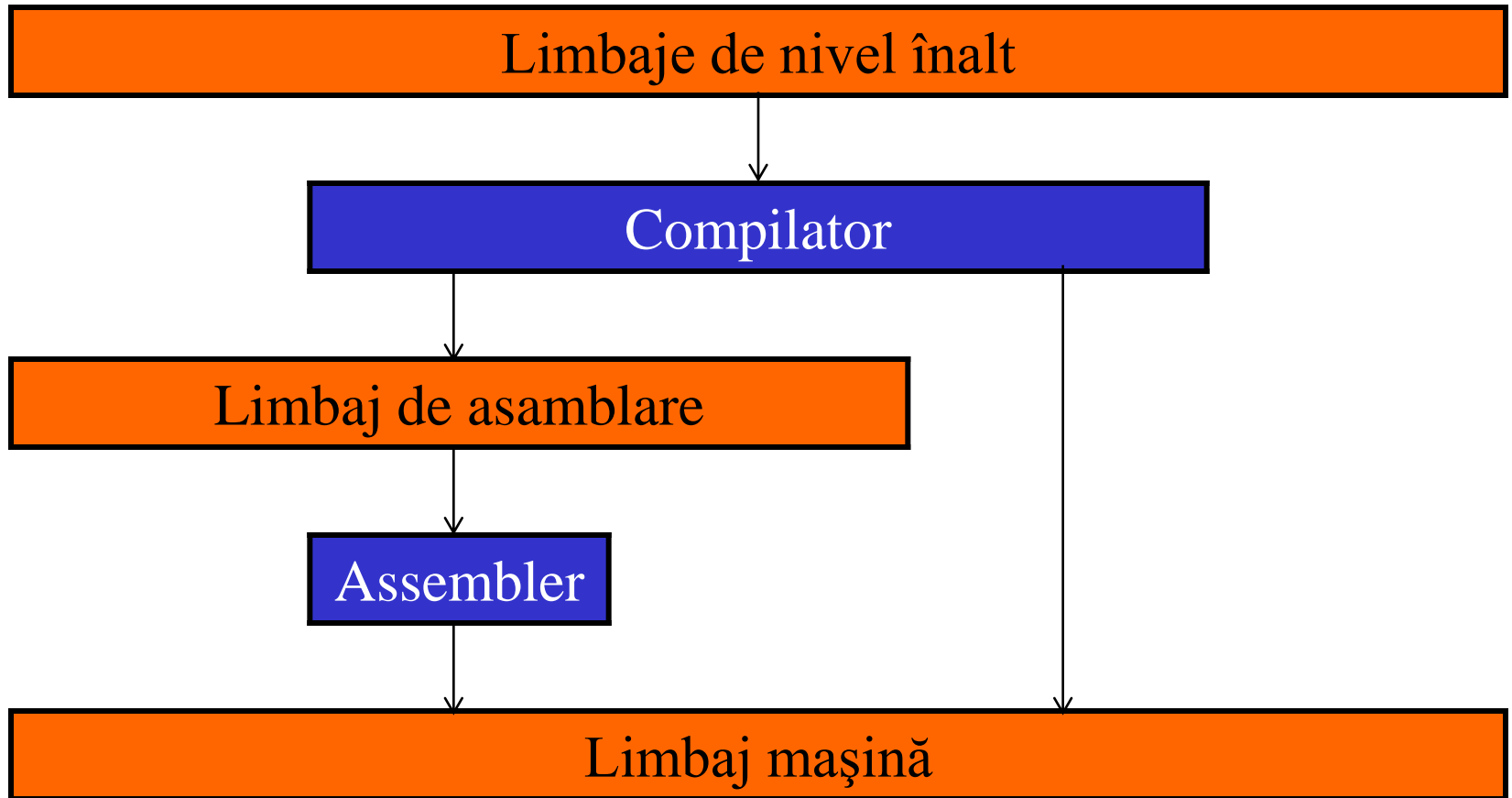
Viziunea programatorului

- Limbajul mașină
 - 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 mașină
inc count

Viziunea programatorului

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

Viziunea programatorului



Viziunea programatorului

- Arhitectura setului de instrucțiuni (Instruction Set Architecture - ISA)
 - specifică funcționarea unui procesor
 - definește 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 ușoară
 - Aceleași motive ca mai sus
- Programele sunt portabile
 - Conțin puține detalii dependente de mașină
 - Se pot folosi cu mici modificări sau chiar fără pe diverse tipuri de mașini
 - Traducerea în limbajul-mașină 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
 - » și din limbaj de nivel înalt se ajunge la programe în limbaj-mașină, dar compactitate mai mică
 - timp
 - » legile localizării cu mai puține excepții și pe ferestre mai mici, deci execuție mai rapidă (mai puține eșecuri)
 - **accesibilitate** la resursele hardware ale sistemului
 - » deoarece specificitățile mașinii sunt luate în considerare

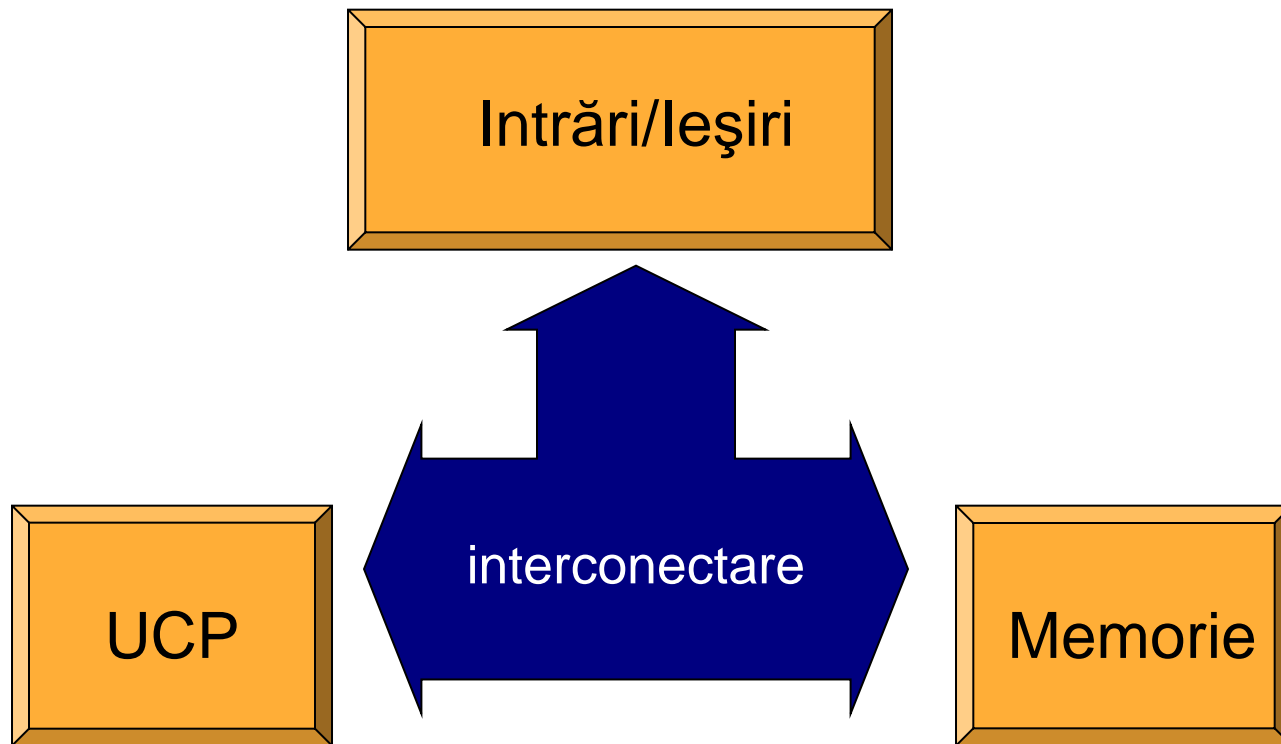
V.2.2.

ARHITECT ȘI IMPLEMENTATOR

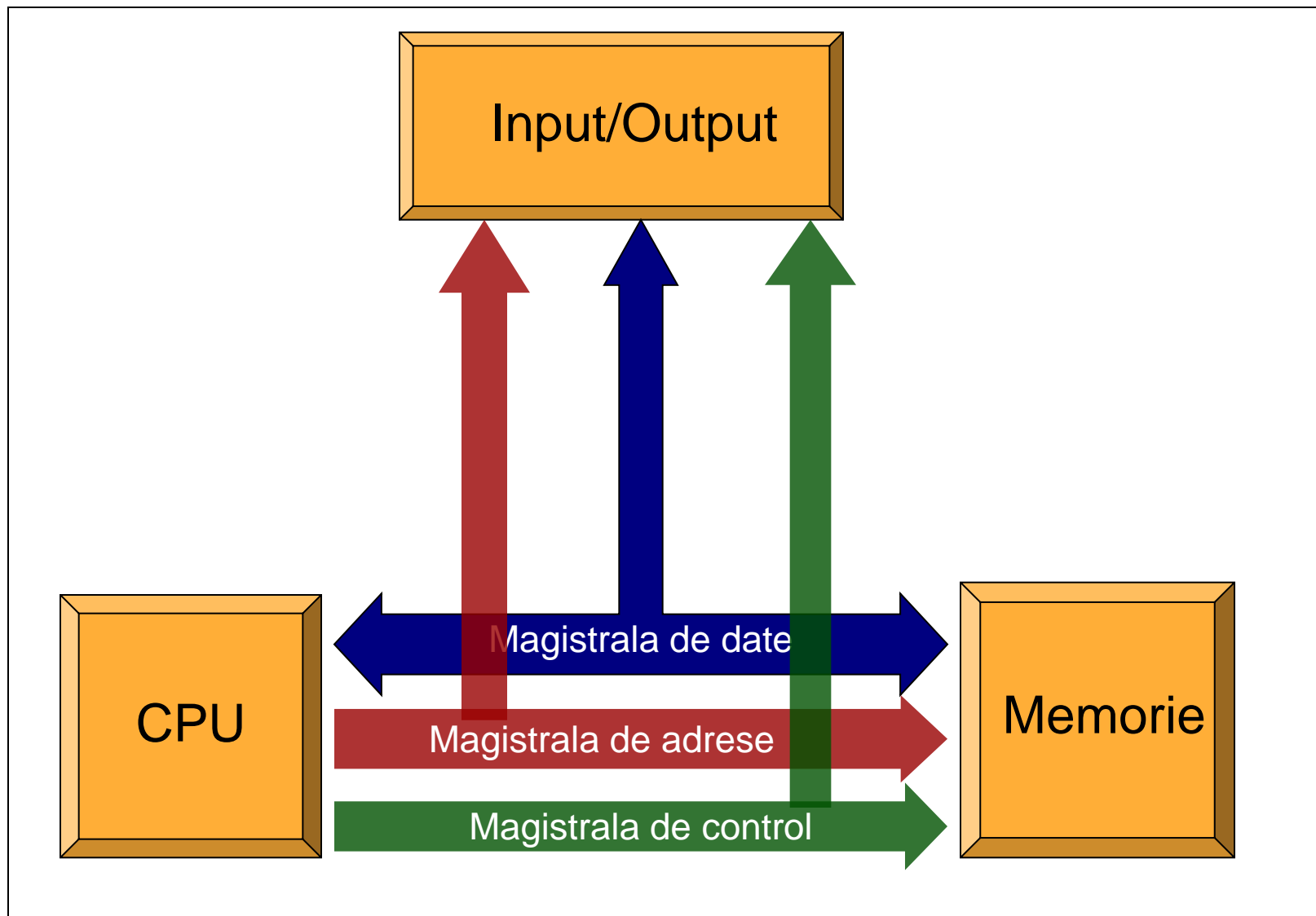
Viziunea arhitectului

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

Viziunea arhitectului



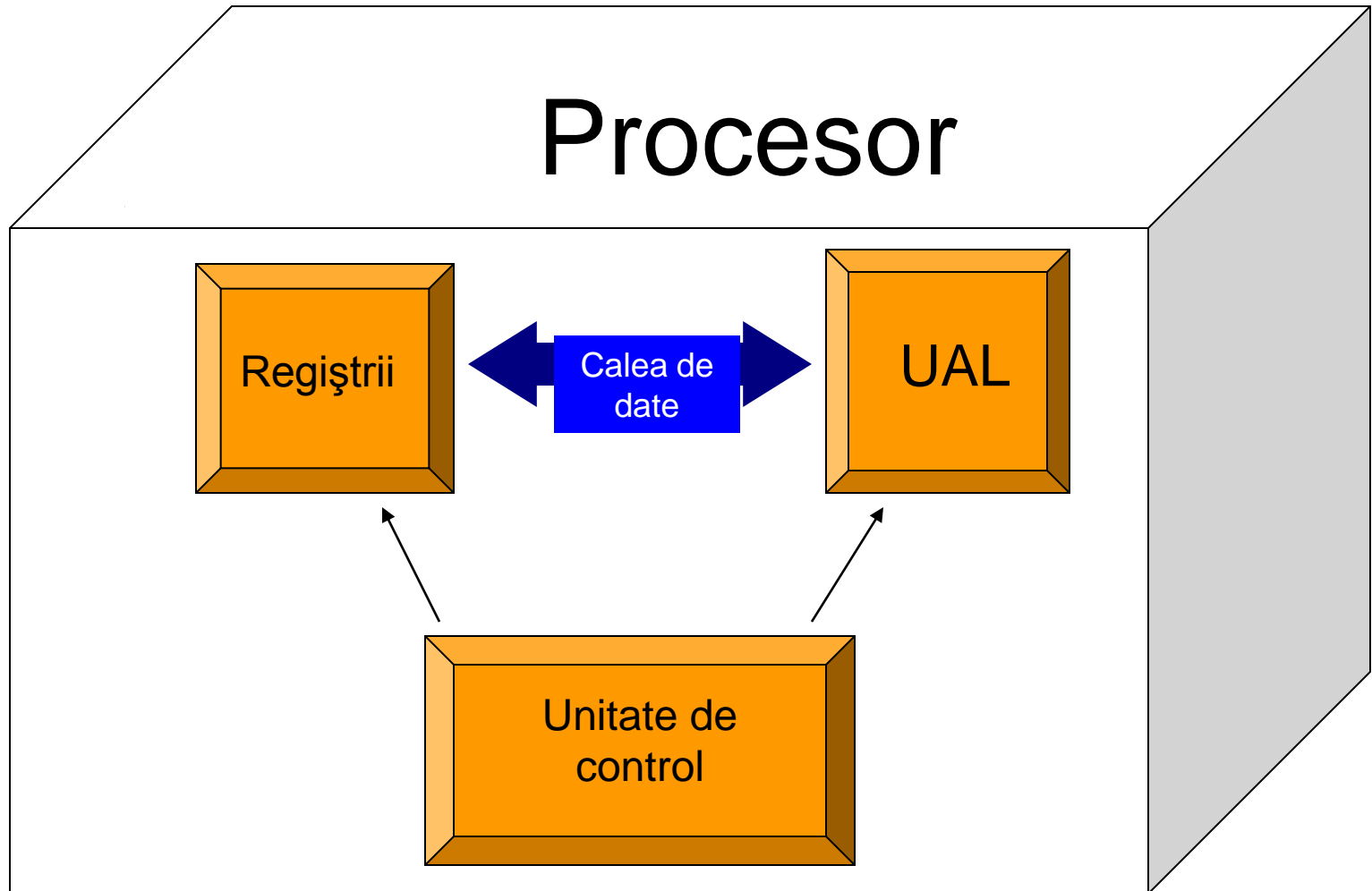
Viziunea arhitectului



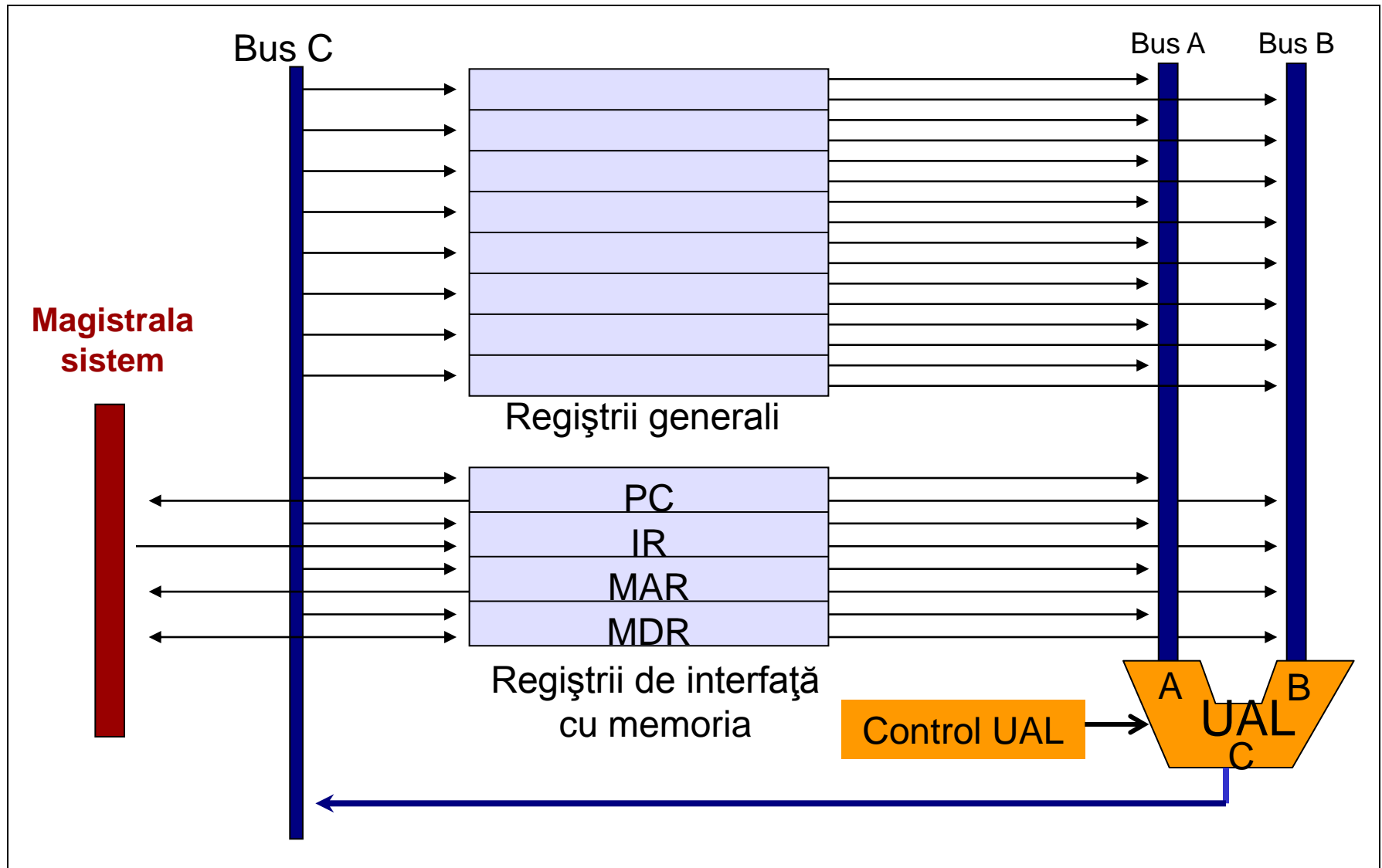
Viziunea implementatorului

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

Viziunea implementatorului



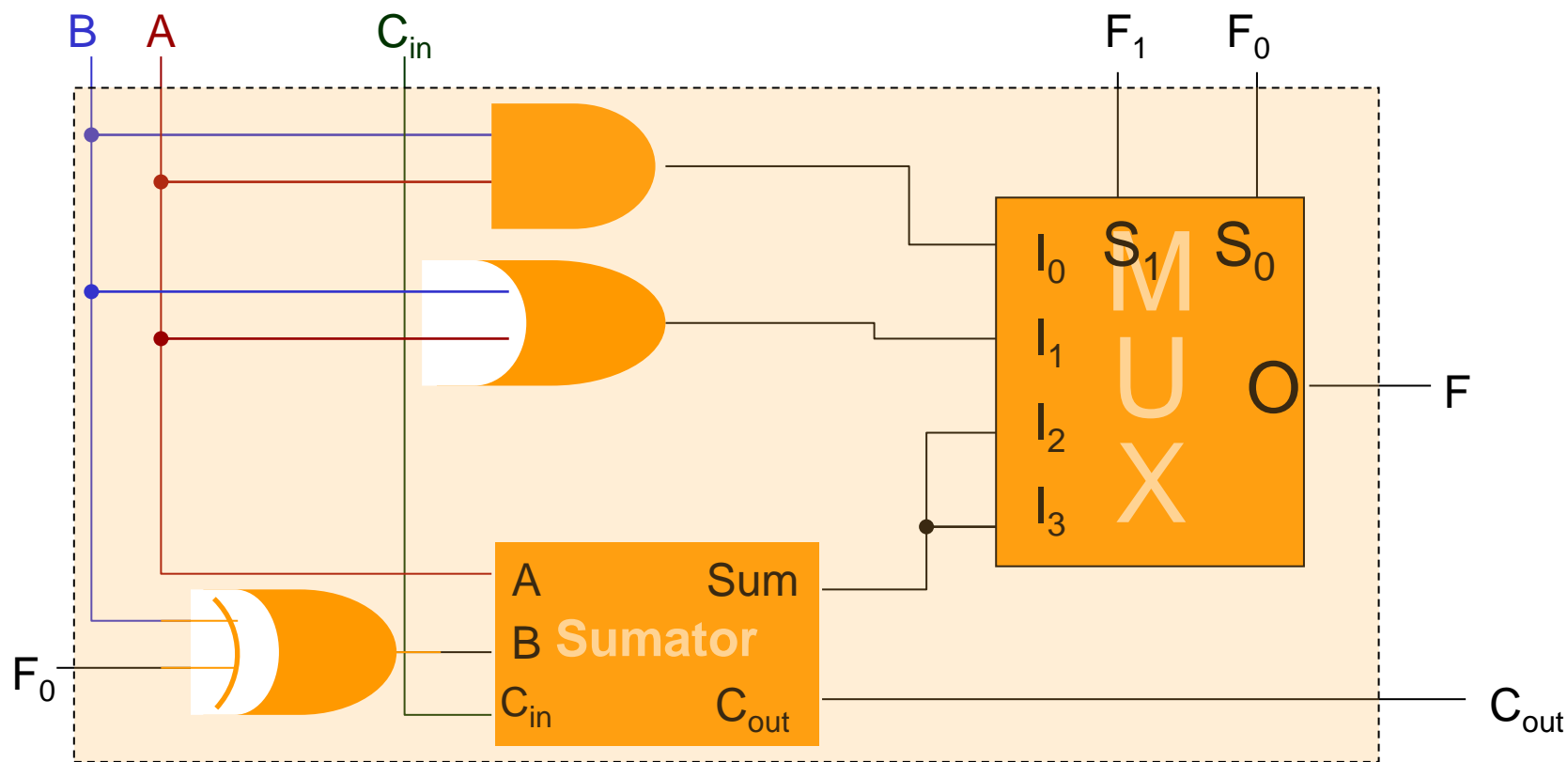
Viziunea implementatorului – calea de date



Sensul transferurilor de date

- În procesor: regiștri \rightarrow UAL \rightarrow regiștri
- Între componente: memorie \leftrightarrow regiștri
- Orice dată se prelucrează trecând prin regiștri
- Unele căi sunt asimetrice

Viziunea implementatorului



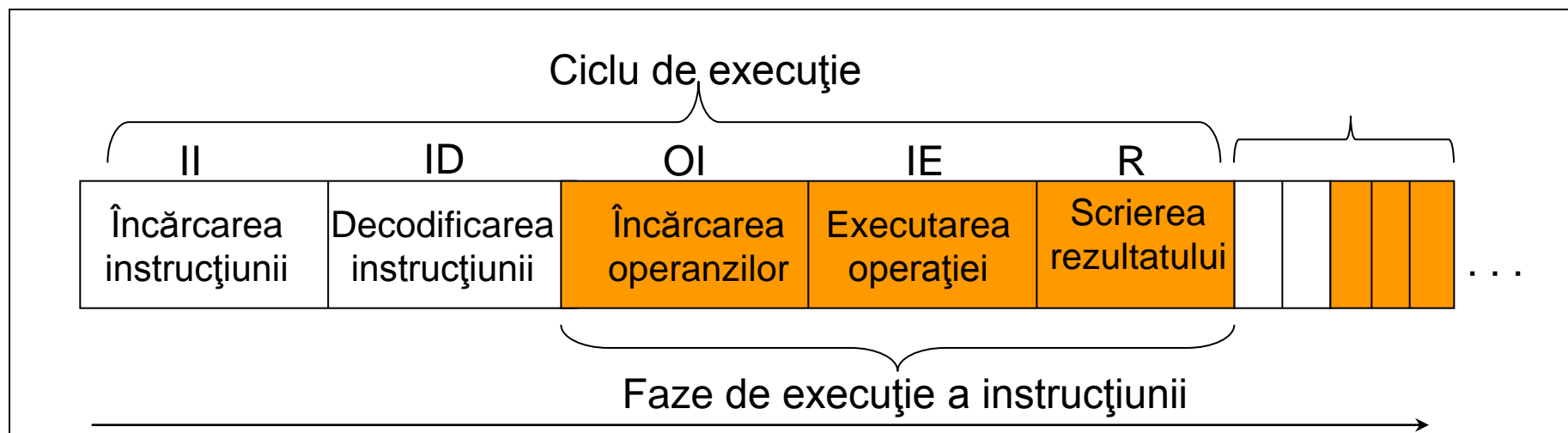
V.3.

SCURTĂ TRECCERE ÎN REVISTĂ
A ORGANIZĂRII
CALCULATORULUI

V.3.1.
PROCESSORUL

Procesorul

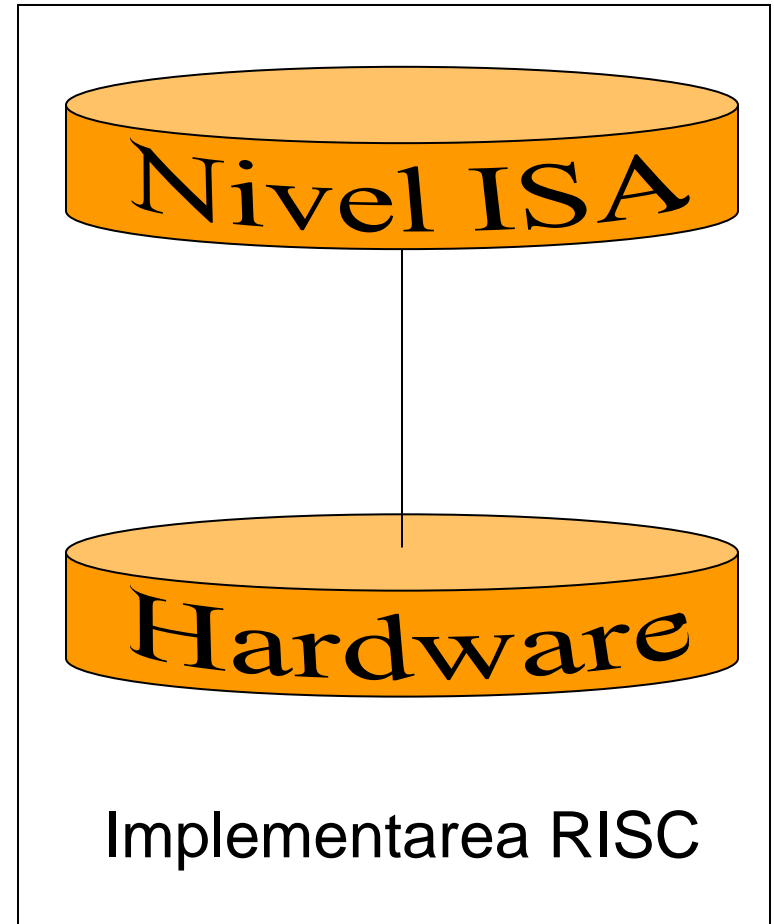
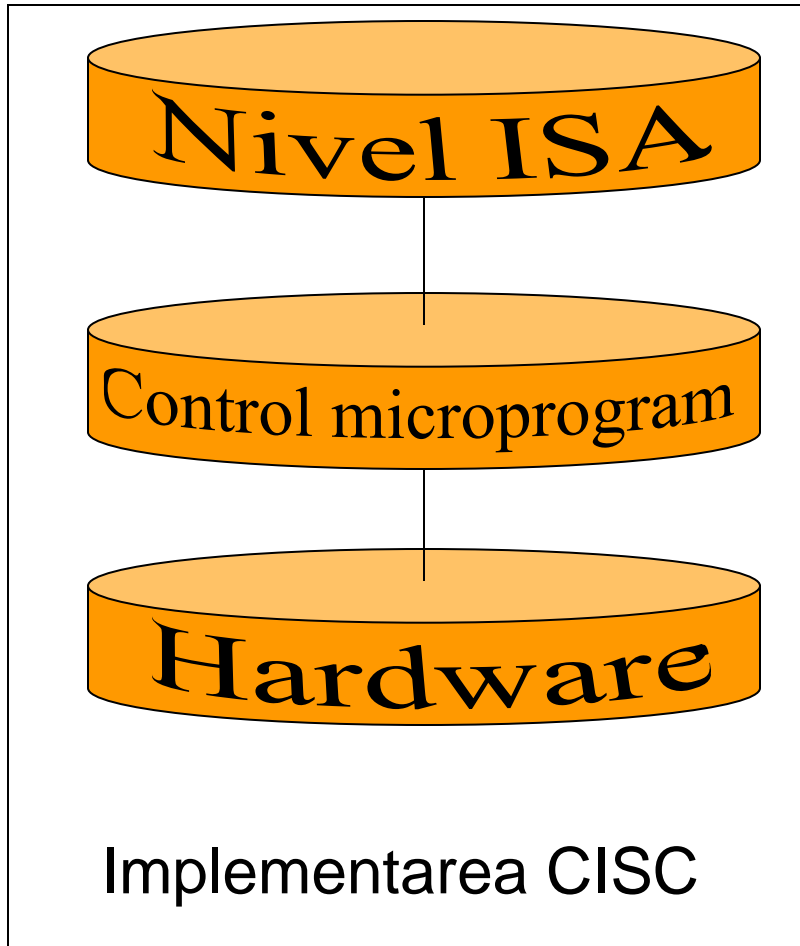
- Ciclul de execuție
 - Fetch (căutare și încărcare)
 - Decodificare
 - Execuție operație
- Arhitectura von Neumann
 - Modelul programului memorat
 - Nici o deosebire la nivel fizic între datele de prelucrat și instrucțiunile care le prelucrează
 - Deosebirile apar la execuție
 - Instrucțiunile se execută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



Procesorul

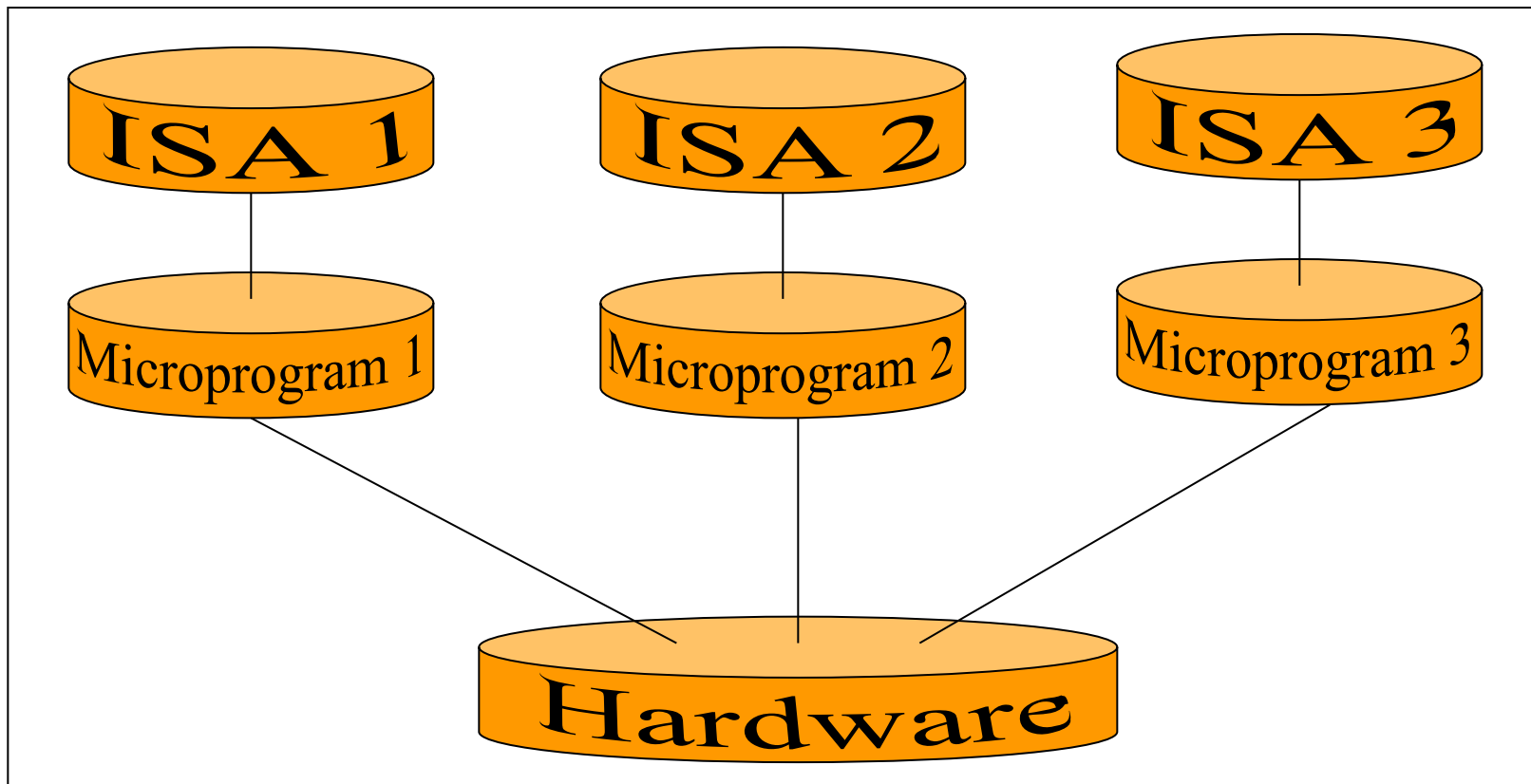
- Complex Instruction Set Computer (CISC)
 - Folosește instrucțiuni-mașină complexe
 - Operanzii pot fi în regiștrii procesorului sau în memorie
 - » Lungimea instrucțiunilor nu este fixă
 - De obicei se folosește **microprogramarea**
 - » O **instrucțiune** în limbaj mașină devine un **program** scris cu instrucțiuni și mai simple
 - » Abia acestea din urmă sunt executate prin construcție (**cablate**) și nu ca un program
- Îmbunătățiri
 - Pipelining
 - Descompunerea în mai mulți pași a execuției fiecărei instrucțiuni
 - Execuție simultană, pe pași, a mai multor instrucțiuni succesive decalate
 - Reduced Instruction Set Computer (RISC)
 - Are instrucțiuni-mașină simple
 - Instrucțiunile mai complicate se obțin din cele simple ca secvențe

Procesorul



Procesorul

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



V.3.2.
MEMORIA

Memoria

- Șir ordonat de **octeți** (nivel logic)
 - Fiecare octet are un predecesor (cu excepția primului) și un succesor (cu excepția ultimului)
 - Nivel fizic: matrice de matrici; piste concentrice
- ***Adresa absolută*** a unui octet: numărul său de ordine din șir
 - 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

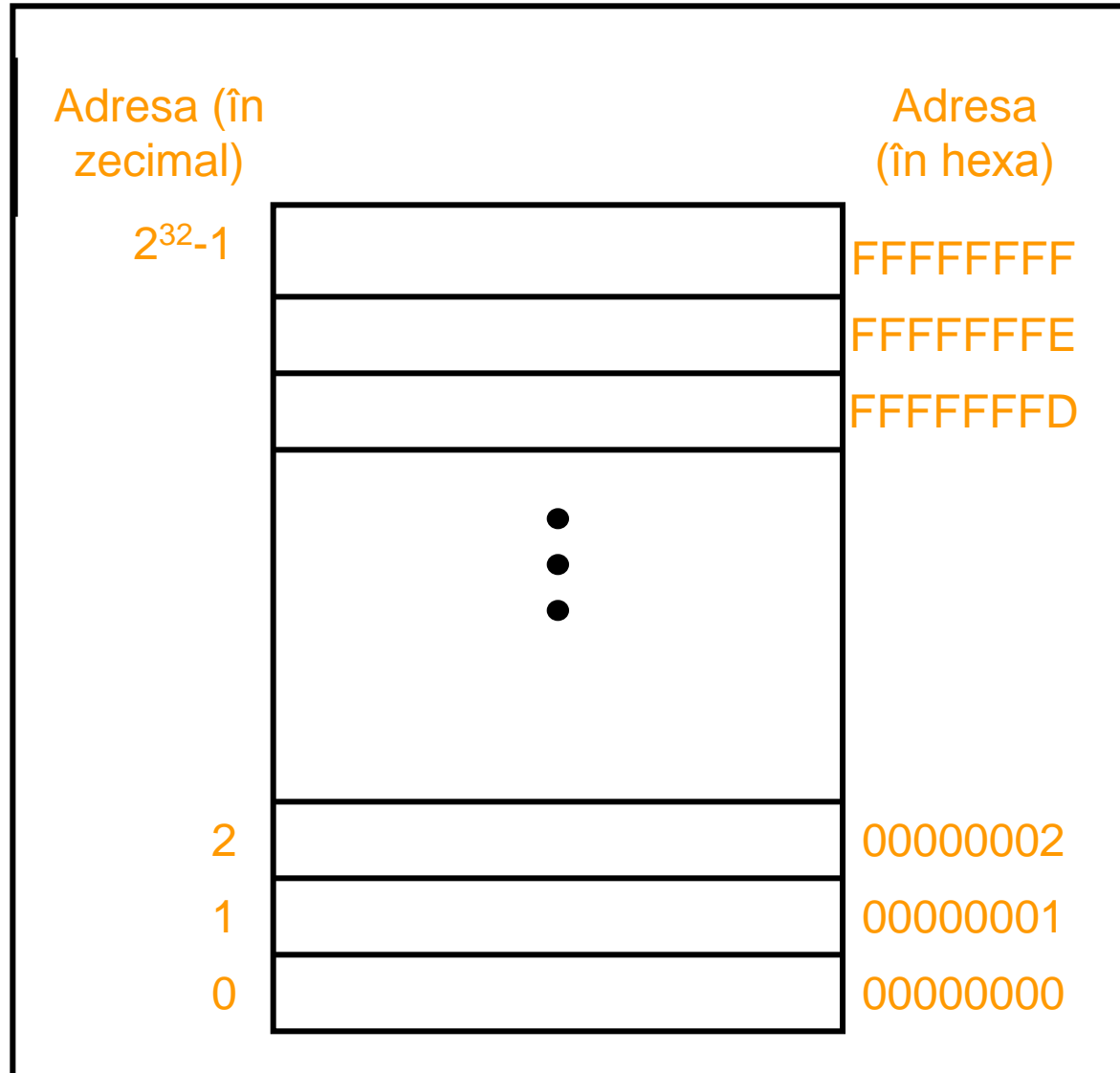
Alte tipuri de adrese

- ***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 numește *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") atașat 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 $\leq 4\text{GB}$ (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

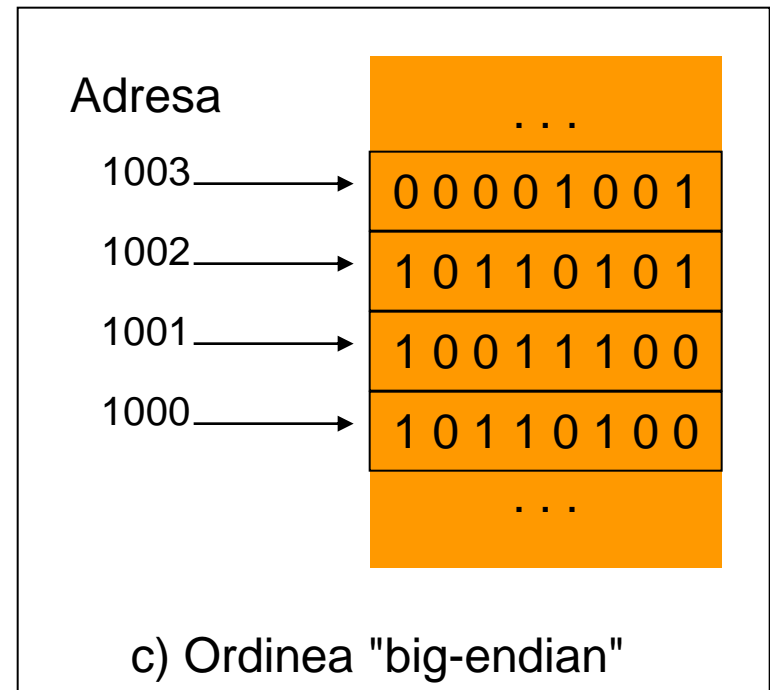
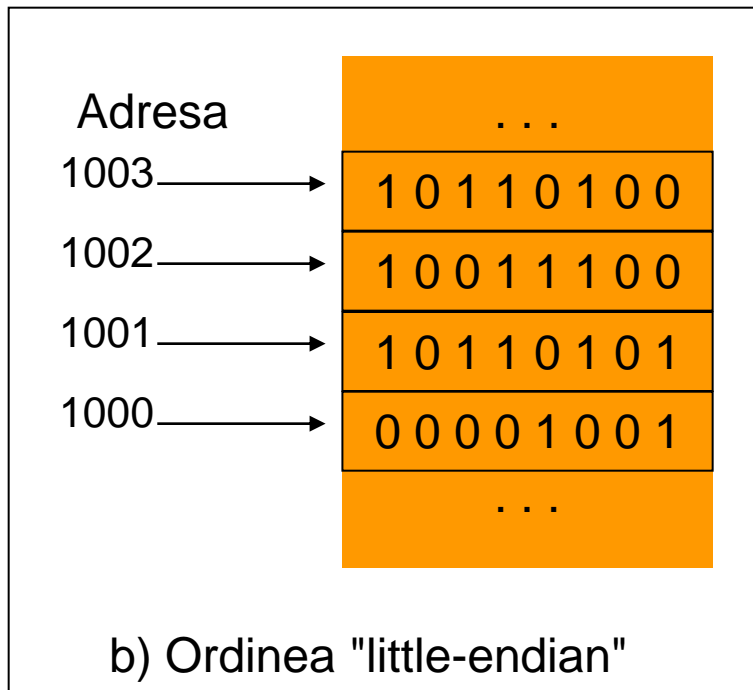
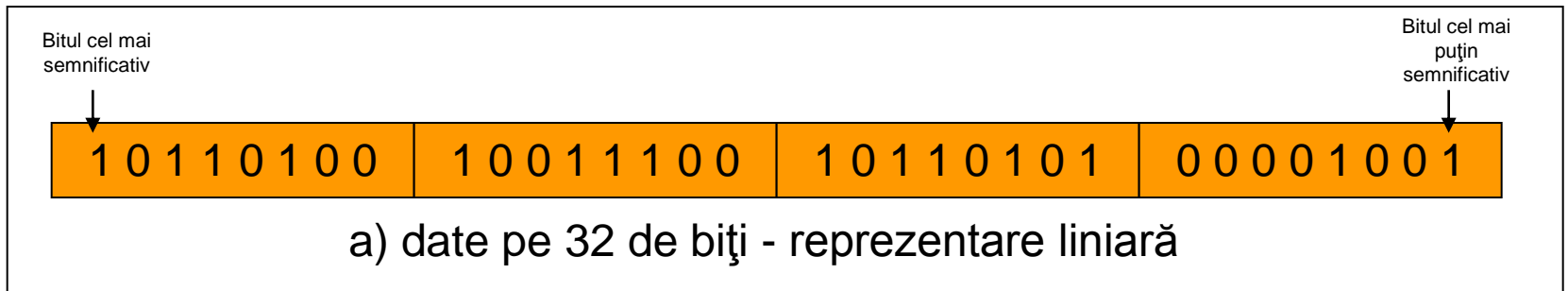
1. Aducerea adresei de citire pe busul de adrese (MAR)
 2. Emiterea semnalului de control "citire memorie" (UC)
 3. Așteptare până când memoria găsește data de citit
 - E nevoie de starea de așteptare (*wait state*) dacă memoria utilizată este înceată
 4. Citirea datei de pe busul de date (MDR)
 5. Inactivarea semnalului de control "citire memorie" (UC)
- Pentium: o citire necesită trei **cicluri de ceas** (tacte)
 - Clock 1: pașii 1 și 2
 - Clock 2: pasul 3
 - Clock 3 : pașii 4 și 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. Așteptare până când memoria efectuează scrierea
 - starea de așteptare (*wait state*) dacă memoria este înceată
 5. Inactivarea semnalului de control "scriere memorie"
- La Pentium, o scriere necesită trei cicluri de ceas
 - Clock 1: pașii 1 și 2
 - Clock 2: pasul 3
 - Clock 3 : pașii 4 și 5

Ordinea octeților



V.3.3.
INTRĂRI - IEȘIRI

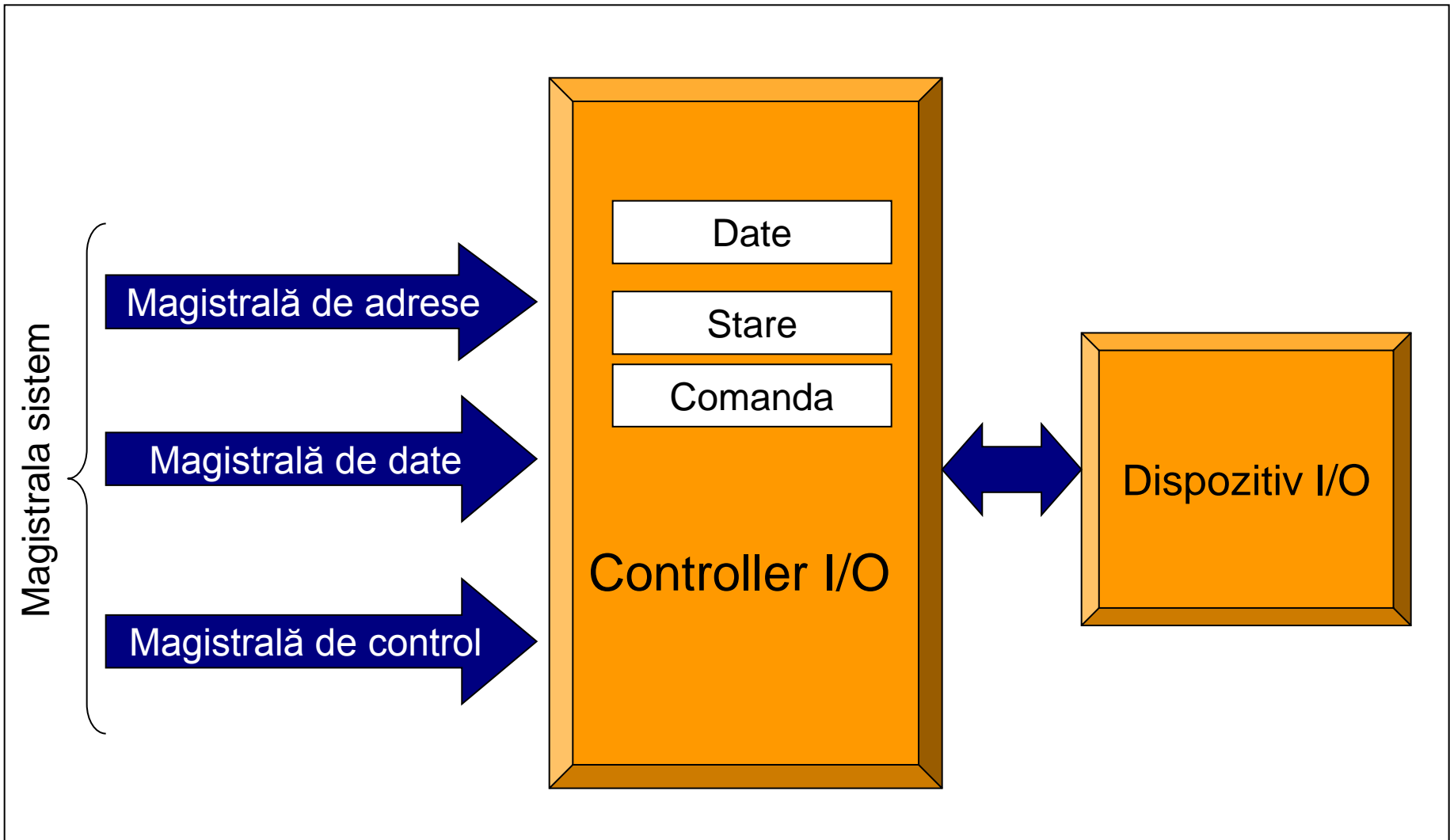
Intrări-ieșiri (Input/Output)

- Perifericele de intrare-ieșire (I/O devices) au interfața gestionată de câte un **I/O controller**
- Gestionează detalii legate de operațiunile de nivel jos (periferic)

Intrări-ieșiri

- Mai multe feluri de a "vedea" I/O
 - I/O Proiectat pe memorie (memory-mapped I/O)
 - Citirile / scrierile sunt "văzute" ca și când s-ar face din / în memorie
 - Aceleași semnale de control ca pentru citire/scriere din/în memorie
 - Așa 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 folosește I/O izolate
 - dar și I/O proiectate pe memorie

Comunicare cu perifericele



Modalități de transmitere date I/O

- Intrări-ieșiri programate (programmed I/O)
 - Programul folosește o buclă "busy-wait"
 - Se aplică transferurilor pentru care se știe 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 și / sau termina transferuri de date
 - Tehnică extrem de eficientă
 - Utilizată pentru transferuri despre care nu se știe 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 și regiștrii
 - Magistralele A, B, și C din exemplul de la pag. 20
 - Există bus de date și 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.

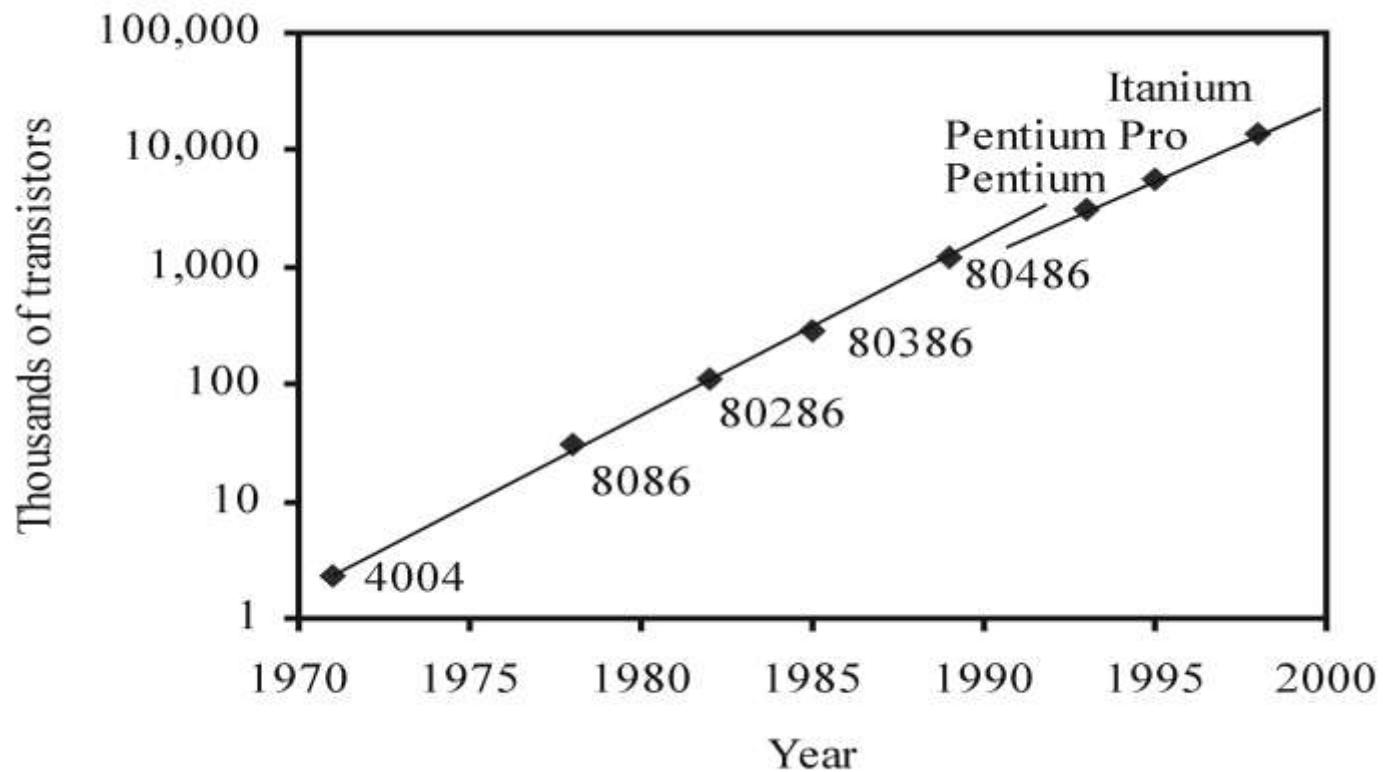
O PERSPECTIVĂ ISTORICĂ

- 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.

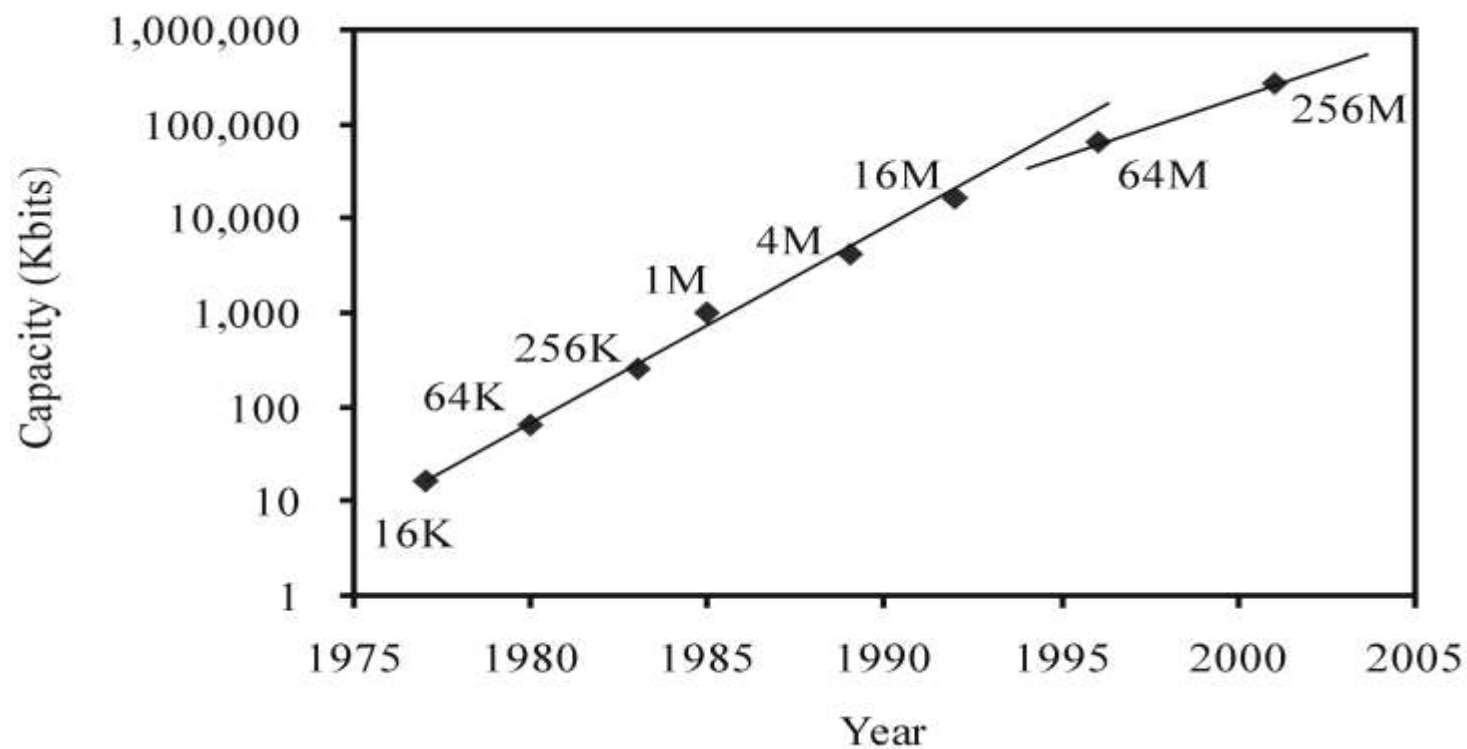
Evoluția tehnologică

- Densitatea tranzistorilor
 - Până prin 1990, se dubla la fiecare 18-24 luni
 - De atunci, dublare la fiecare 2 ani și 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)

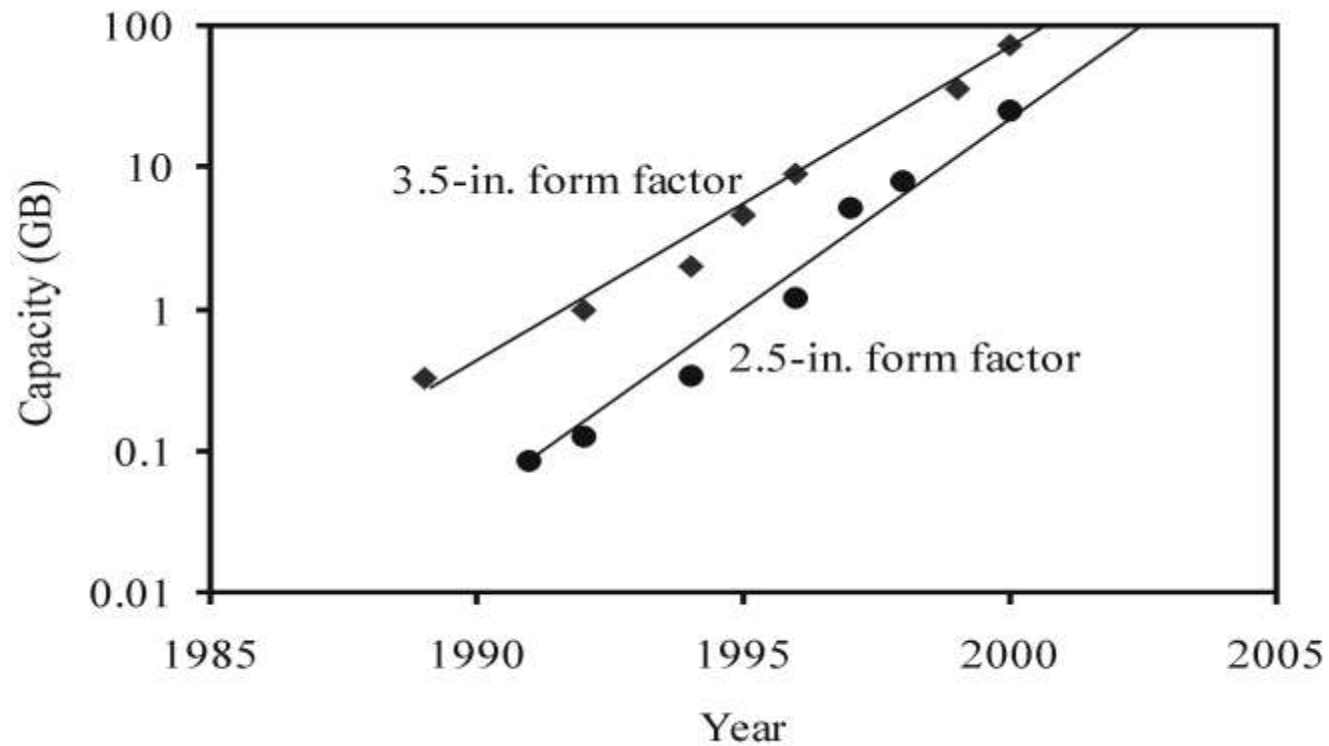
Evoluția tehnologică: integrare



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}