V. Arhitectura și organizarea calculatorului

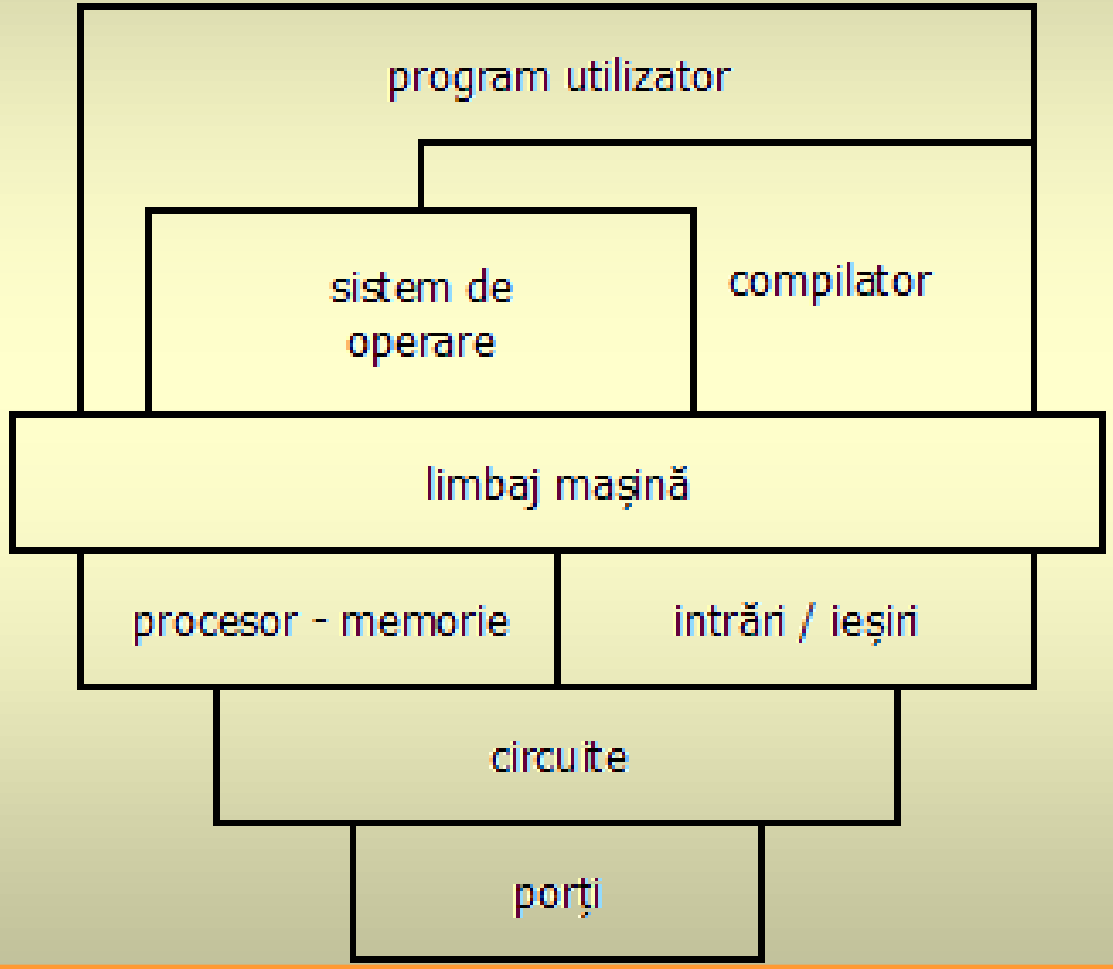
Calculatorul von Neumann (1)

- 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)

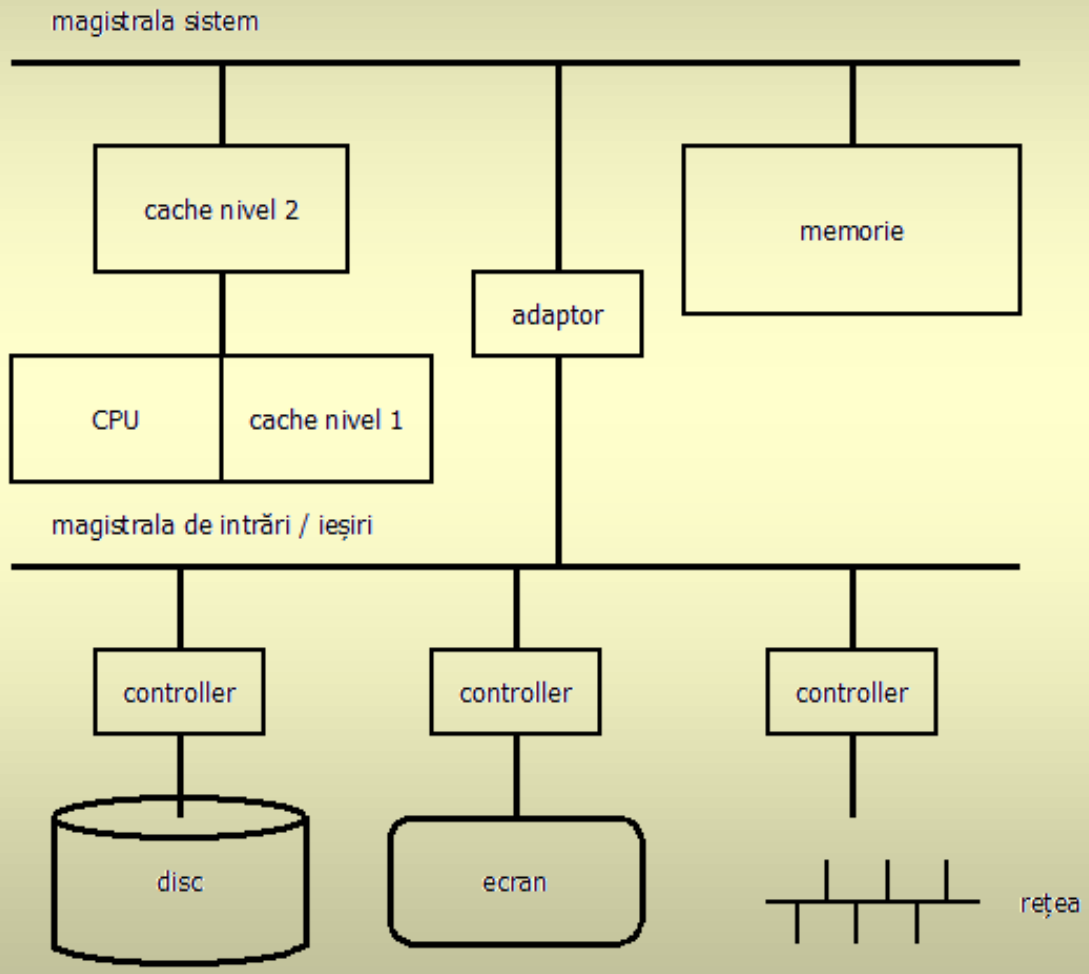
Calculatorul von Neumann (2)

- adresa instrucţiunii 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 instrucţiune este încărcată pentru execuţie

Arhitectura unui sistem de calcul



Organizarea unui calculator



V.1. Arhitectura calculatorului: viziuni posibile

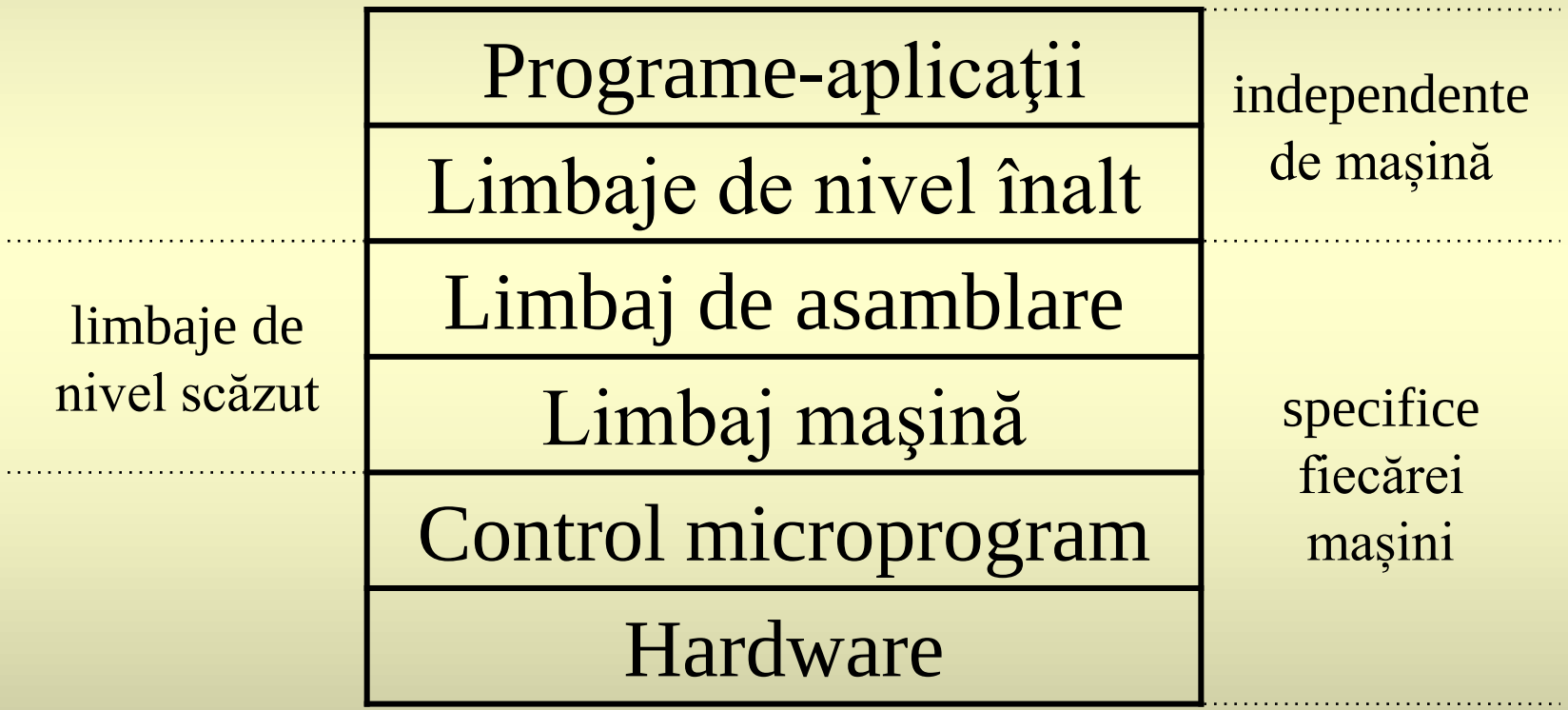
Programator vs. utilizator

- 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)



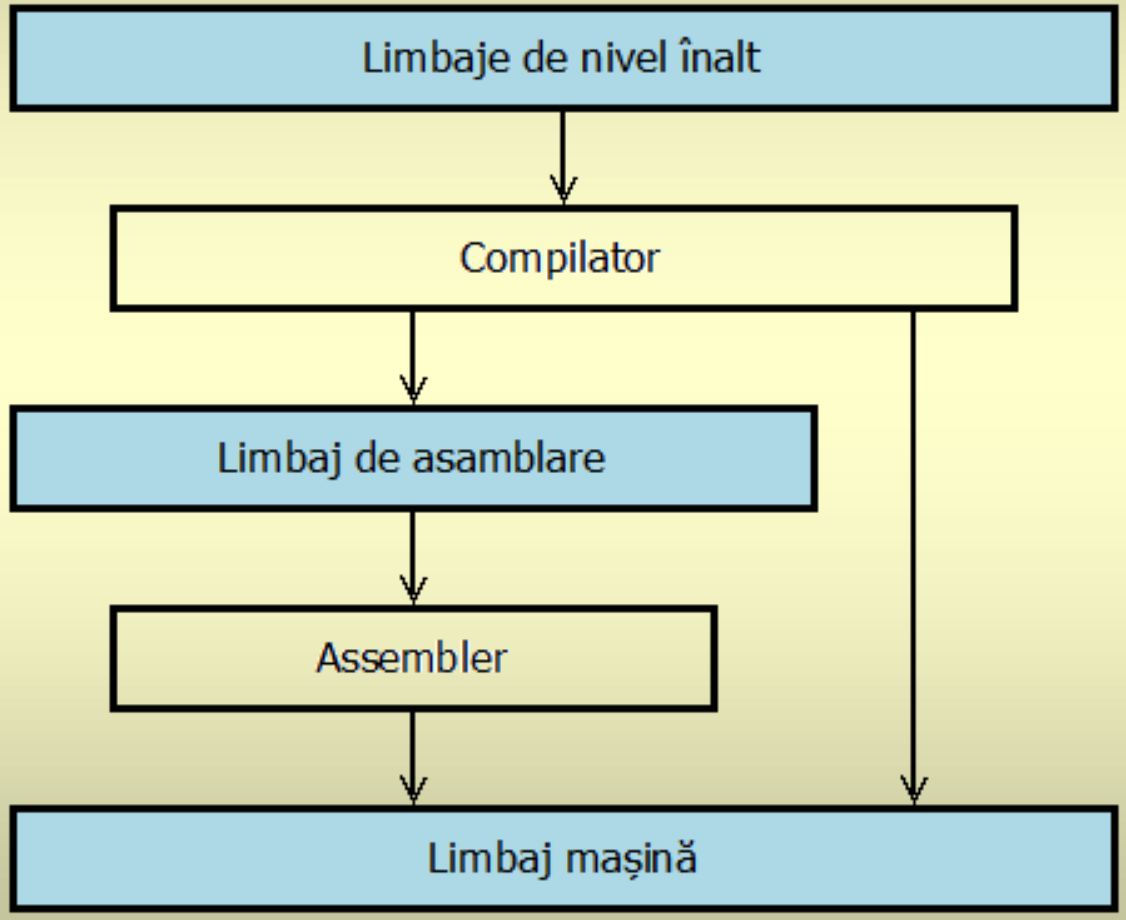
Limbaje de nivel scăzut

- 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
Compilator
Limbaj de asamblare
Assembler
Limbaj mașină

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
  - puţine detalii dependente de maşină
    - dar nu deloc - portabilitatea nu e chiar 100%
  - fiecare limbaj necesită un compilator specific

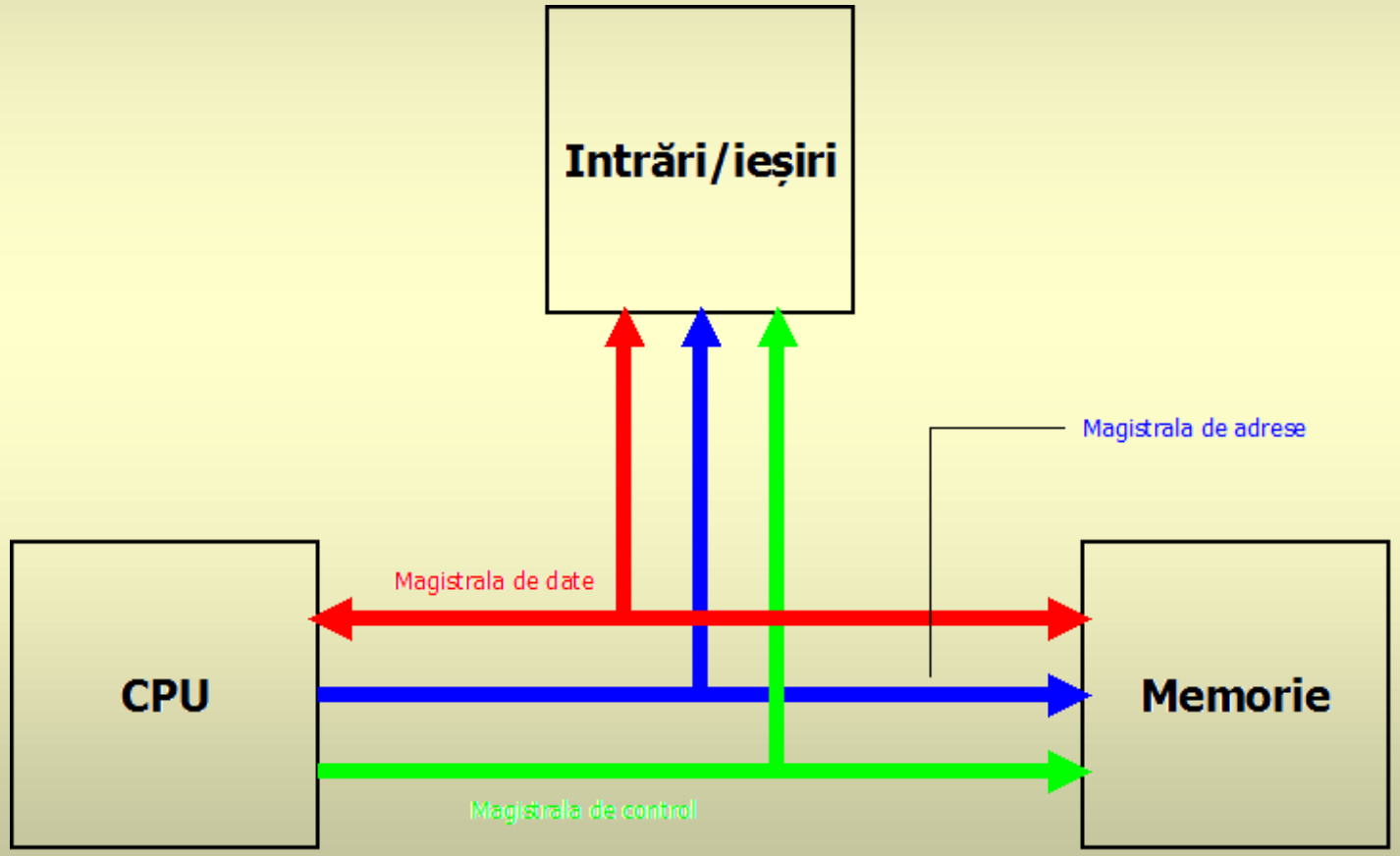
Limbaj de asamblare - avantaje

- 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

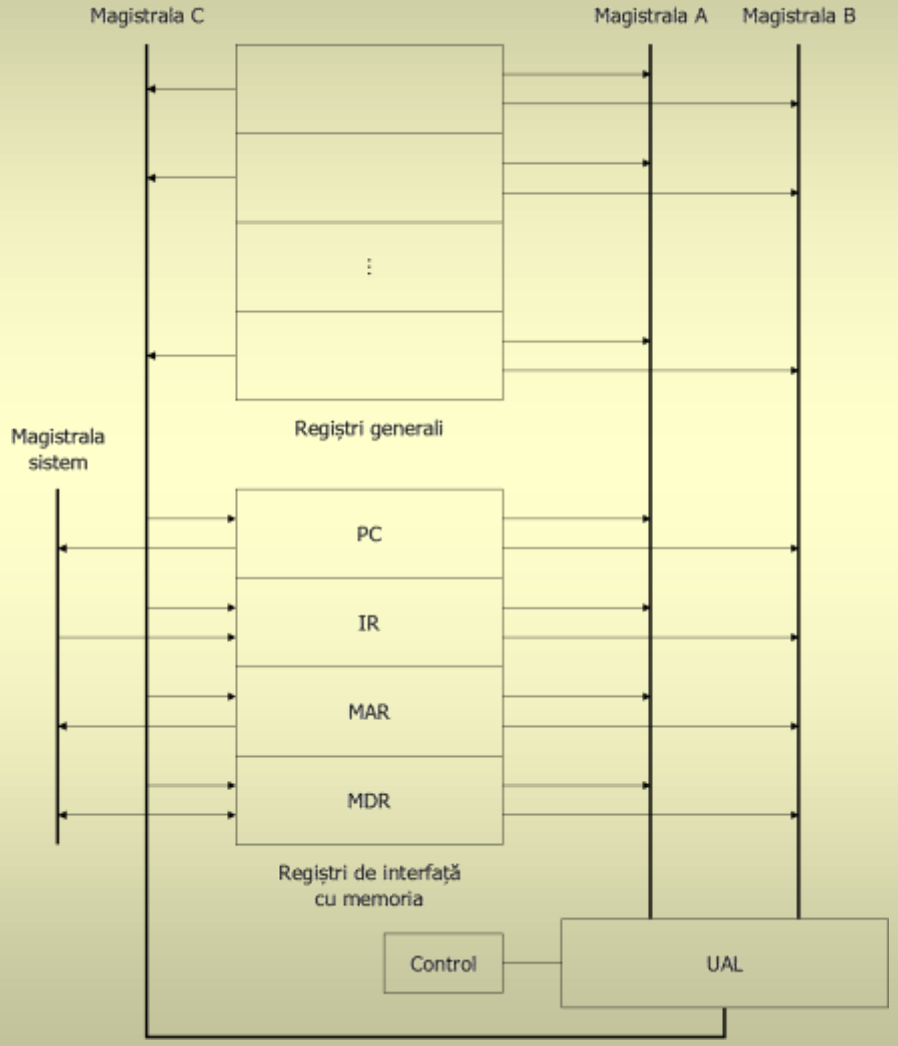
Viziunea arhitectului



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 şi procesor
- execută instrucţiunile indicate de programator
- realizează prelucrarea datelor
- coordonează funcţionarea celorlalte componente

Memoria

- stocarea informaţiilor
  - date
  - instrucţiuni
- furnizarea informaţiilor la cerere
- rol pasiv
  - "răspunde" la cererile venite din exterior
  - nu iniţiază niciodată un transfer

Dispozitivele periferice

- comunicarea cu exteriorul
- funcţii - foarte variate
  - preluare date
  - afişare
  - imprimare
  - stocare (persistentă)
  - etc.

Magistralele

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

V.3. Memoria

Tipuri de memorie

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

Memoria ROM - tehnologii

- PROM (*Programmable ROM*) - conţinutul său poate fi programat de utilizator
- EPROM (*Erasable PROM*) - poate fi şters şi reprogramat de mai multe ori
  - UVEPROM (*Ultra-Violet EPROM*) - ştergere prin expunere la radiaţie UV
  - EEPROM (*Electrical EPROM*) - ştergere prin impulsuri electrice

Memoria RAM - tehnologii

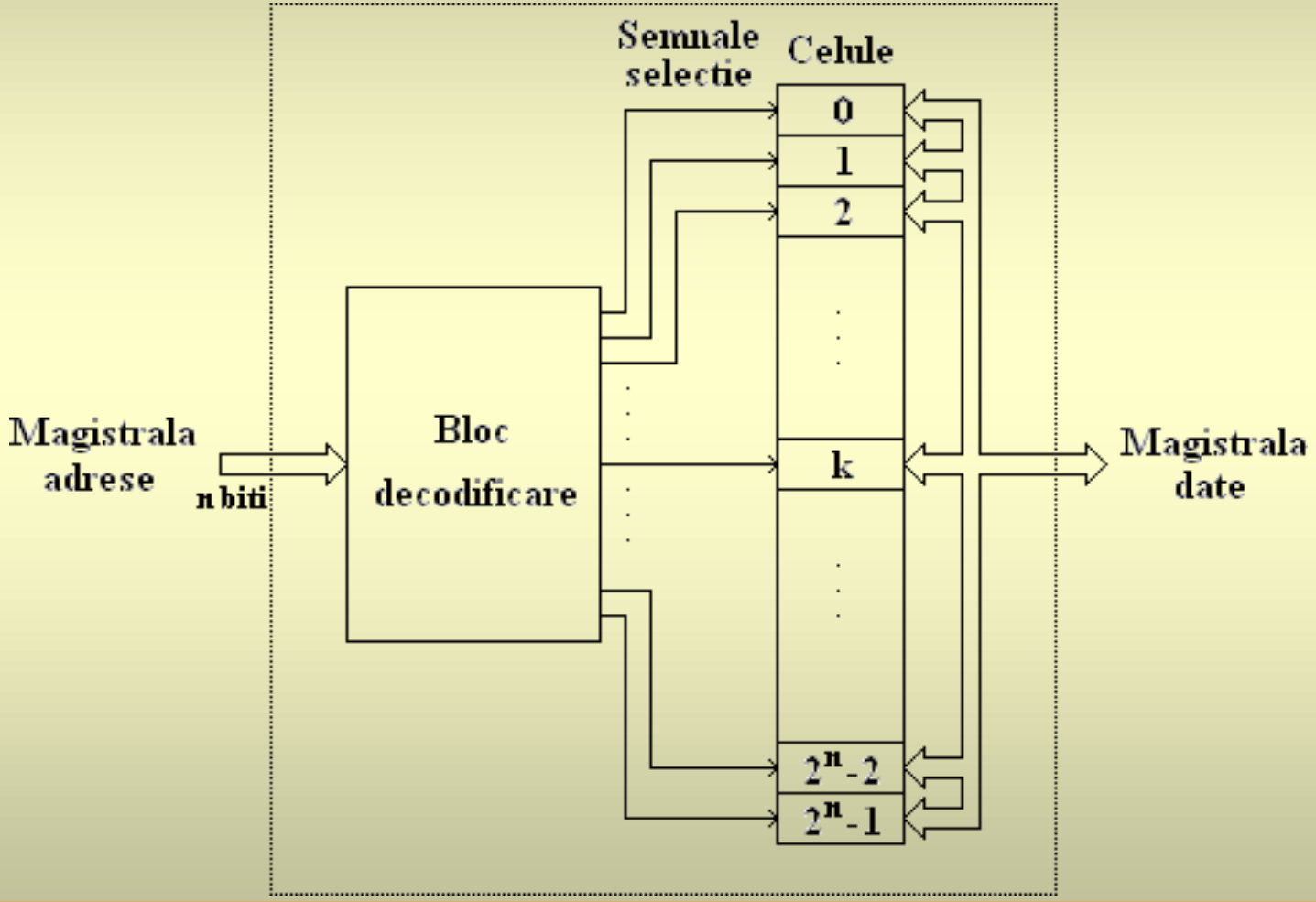
- SRAM (*Static RAM*)
  - viteză mare
  - preţ ridicat
- DRAM (*Dynamic RAM*)
  - mai lentă
  - densitate de integrare mare → spaţiu ocupat mic
  - preţ mai redus

Structura memoriei (1)

- şir unidimensional de celule (locaţii)
- fiecare celulă are asociat un număr unic - *adresa*
- bloc decodificare - selectează locaţia cu adresa indicată
- dimensiunea circuitului de memorie - dată de numărul de biţi de adresă

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

Structura memoriei (2)



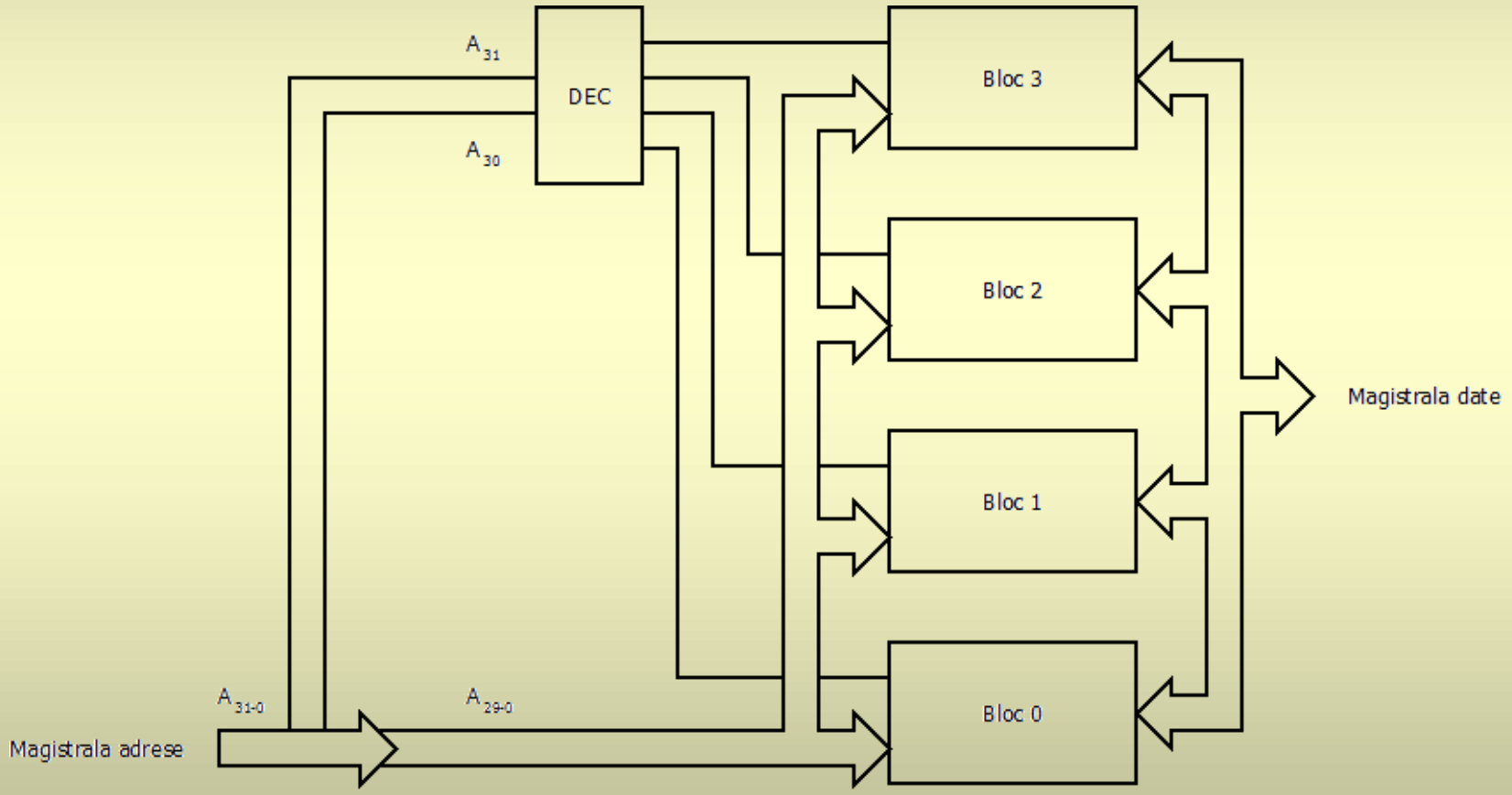
Spațiul adreselor de memorie

- 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

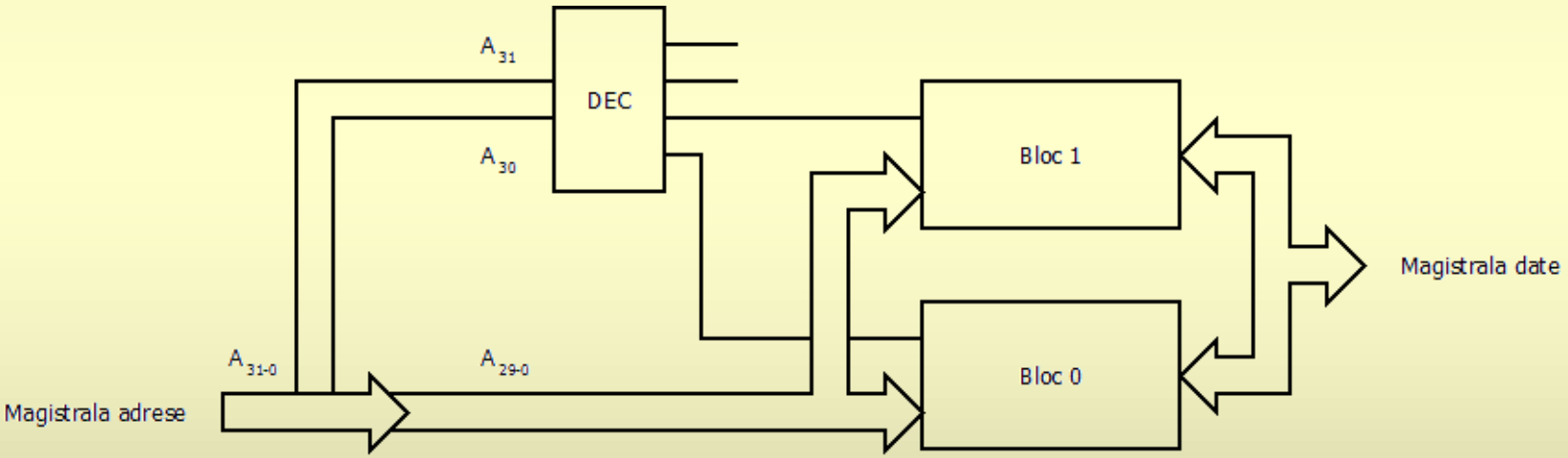
Decodificarea adreselor

- exemplu: procesor pe 32 biţi
- 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 parţială



Tipuri de adrese

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

Ordinea octeţilor (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 octeţilor (2)

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

Exemplu

unsigned int x = 0xB67A49E3; // $B67A49E3_{(16)}$

big endian

adresă	valoare
	. . .
153	$B6_{(16)} = 10110110_{(2)}$
152	$7A_{(16)} = 01111010_{(2)}$
151	$49_{(16)} = 01001001_{(2)}$
150	$E3_{(16)} = 11100011_{(2)}$
	. . .

little endian

adresă	valoare
	. . .
153	$E3_{(16)} = 11100011_{(2)}$
152	$49_{(16)} = 01001001_{(2)}$
151	$7A_{(16)} = 01111010_{(2)}$
150	$B6_{(16)} = 10110110_{(2)}$
	. . .