Programarea calculatoarelor - Limbaje

Curs 1 - Introducere

ÎN ACEST CURS...

1. Algoritmi
2. Limbaje de programare
3. Etapele dezvoltării unui program
4. Reprezentarea informaţiei în calculator
5. Tipuri de date şi de instrucţiuni in programare

1. ALGORITM

Ce este ?

- Problema → Algoritm → Soluție
- Date intrare → Algoritm → Date ieșire

||

Mulțime finită și ordonată de operații

Caracteristici

- Clar
- Universal
- Finit

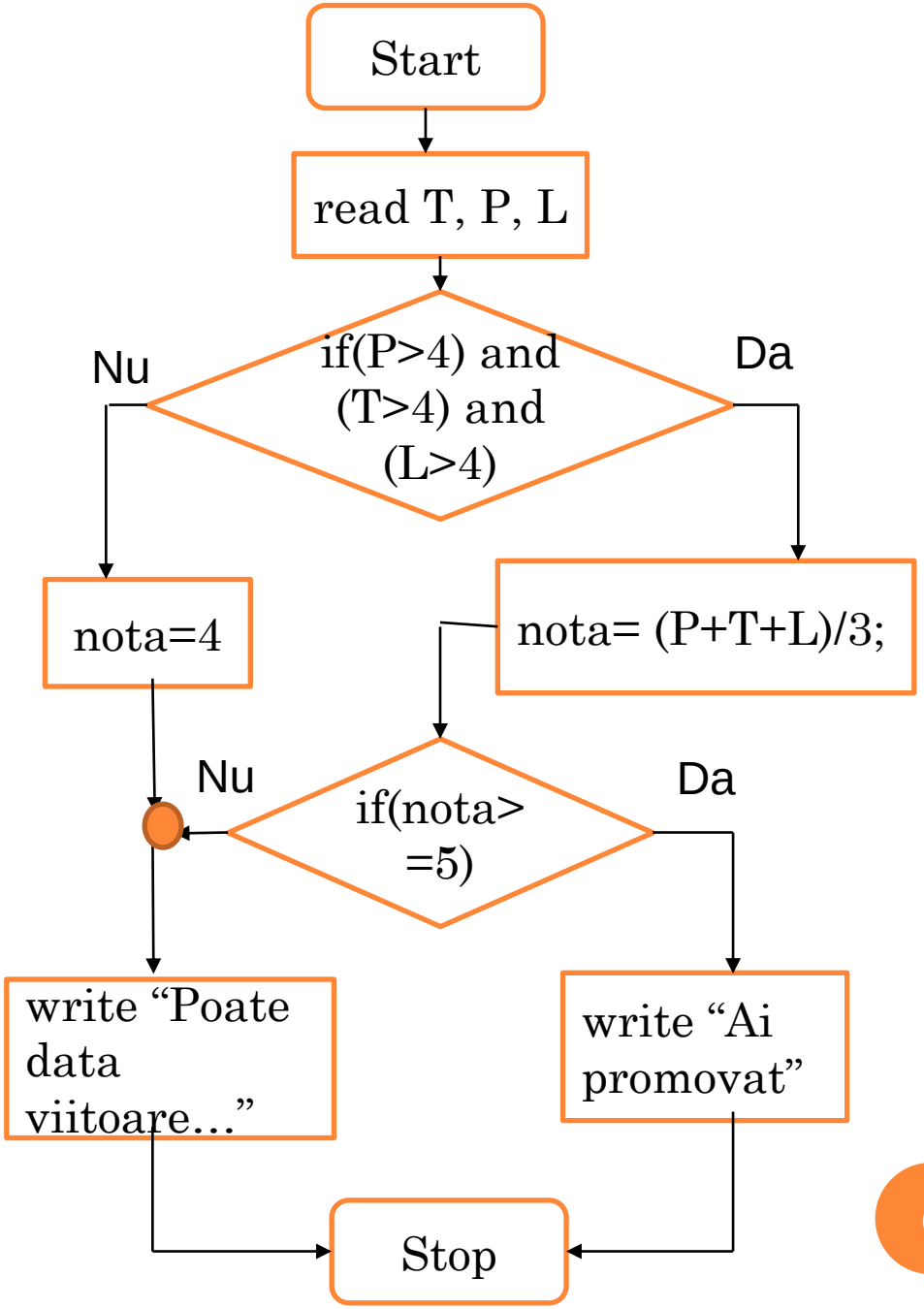
Cum reprezentăm algoritmii?

- Limbaje dedicate formalizate – ALGOL (ALGorithmical Language)
- Pseudo limbaj, convențional/neconvențional
- Scheme logice
- Diferite diagrame: Booch sau mai general UML (succesiunea operaţiilor prin aşa-numitele “activity diagrams”)

PSEUDOCOD

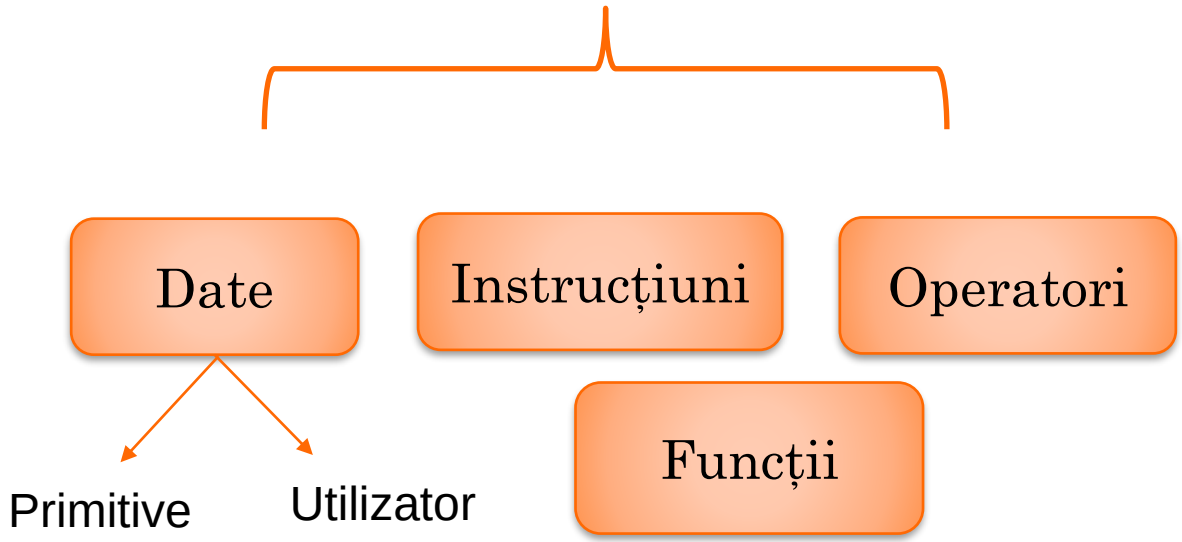
```
Algoritm Examen
begin
read P, T, L
if(P>4) and (T>4) and (L>4) then
    nota= (P+T+L)/3;
else
    nota=4
if(nota>=5) then
    write “Ai promovat”
else
     write “Poate data viitoare…”
end
```

Schema Logică



Care sunt elementele cu care lucrează algoritmii?

Date intrare → **Algoritm** → **Date ieşire**

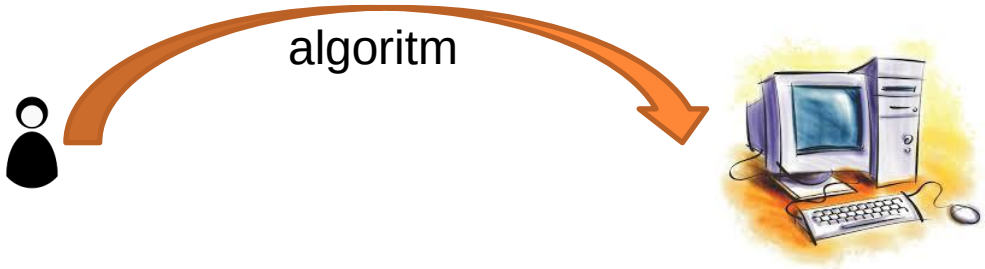


2. Limbaje de programare

Ce înţelegem printr-un program?

Program = rezultatul exprimării algoritmilor într-un *limbaj de programare*. Programele (*software)* se execută pe mașini *hardware.*

- Programare:
  - ansamblul comunicării om-calculator



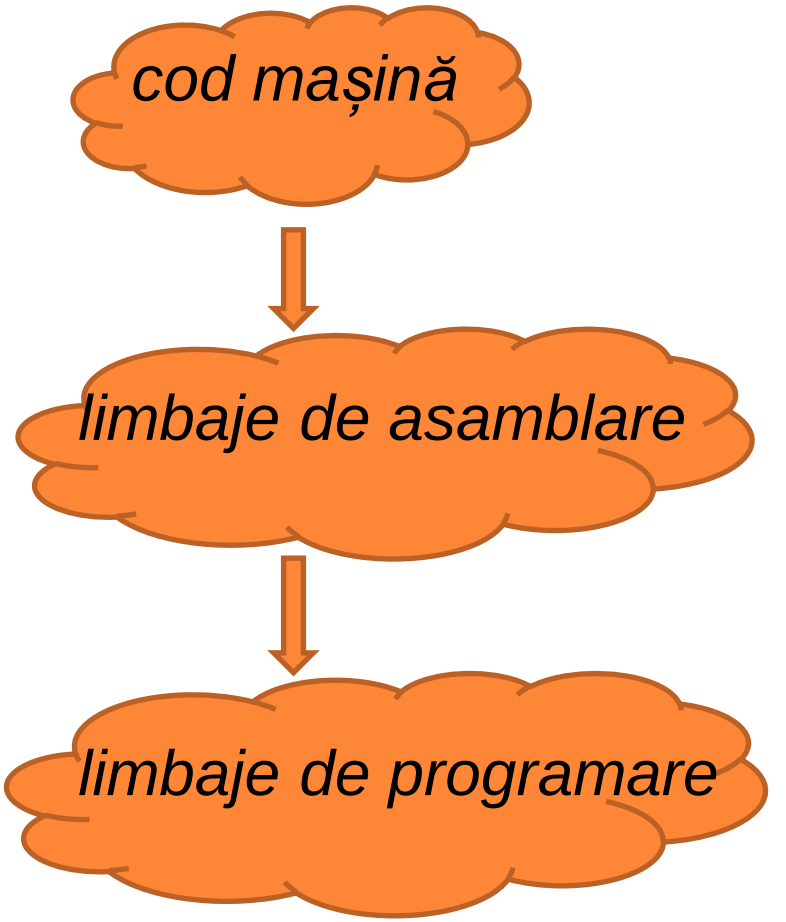
- Programarea - HMI (Human Machine Interaction)

Ce limbaj înţelege calculatorul?

Limbaj propriu = *limbaj maşină (cod maşină)*:

- programarea dificilă
- programul va funcţiona numai pe acel tip de calculator (neportabil)
- depanarea – greoaie

EVOLUȚIA PROGRAMĂRII



→ *valori numerice, fiecare cu o anumită semnificaţie*

1011000001010101_2=45141

→ *programare simbolică folosind mnemonice*

MOV BL, 055H

→ vocabular apropiat de limbajul natural

reguli precise, lipsite de ambiguităţi

[register] **int b=85;**

LP - "compilator“

- verifică corectitudinea unui text scris într-un LP (PS-Program Sursă)
- *faza de analiză* → obținerea unei forme intermediare
- *faza de sinteză* → Programul Obiect (PO) ce poate fi ulterior executat în *faza de execuție*

Ex: C/C++, C#

LP - "interpretor“

- analizează programul sursă şi execută fiecare instrucţiune fără a genera program obiect

Ex: Basic, PHP, Python, Perl, Ruby

LP mixte

Ex: Java

- Un LP poate fi definit prin sintaxa și semantica sa:

Sintaxa

set de reguli ce guvernează **alcătuirea** programelor; pentru descrierea sintaxei se folosesc *meta-languages* cum este BNF (Backus Naur Form) sau van Wijngaarden Form

Semantica

set de reguli ce determină **semnificația** propozițiilor unui limbaj

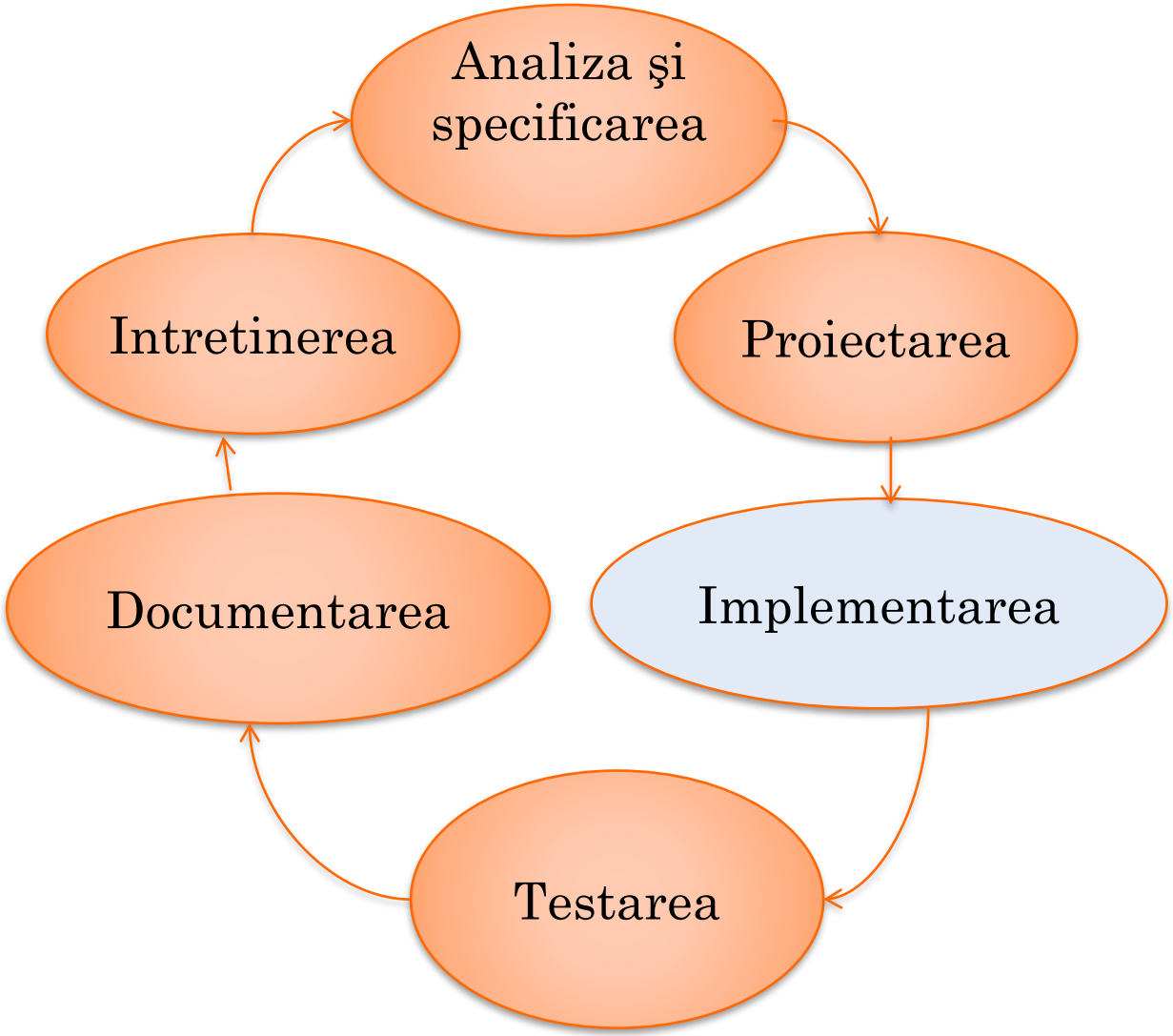
- Clasificarea LP
  - după **dependenţa de hardware**:
    - *LP de nivel jos*: cod maşină, asamblare
    - *LP de nivel înalt*: nu depind de hardware
    - *LP de nivel mediu*: combină avantajele LP de nivel înalt cu funcţionalitatea LP de nivel scăzut
  - după **prelucrările** pe care le efectuează:
    - *LP procedurale* → Basic, Pascal, C
    - *LP declarative* → Lisp, Prolog, SQL, HTML, CSS
    - *LP obiectuale* → C++, C#, Java, Eiffel, Smalltalk
    - *LP funcționale, matematizabile* → Haskell, ML (metalanguage) , etc.

Obs! C++1y are facilităţi funcţionale

3. Etapele dezvoltării unui program

- Programarea →activitatea de elaborare a unui program

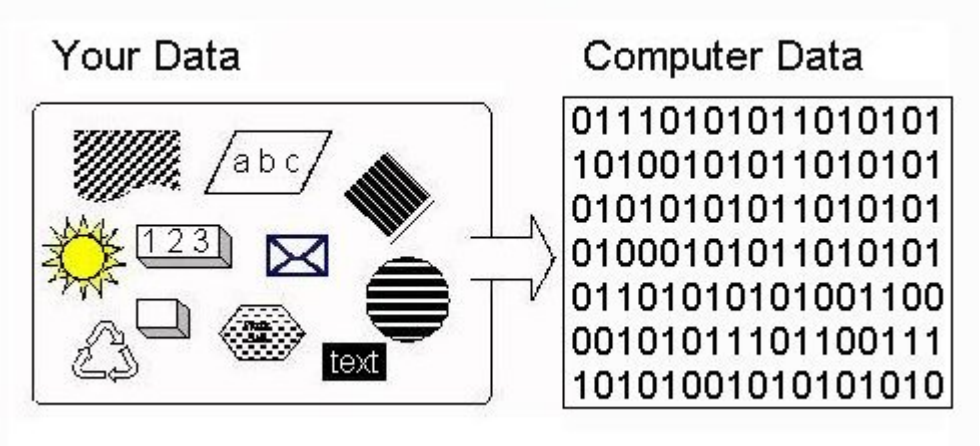
- Etape:



- Etapele implementării:
  - editarea programului ➔ programul sursă
  - compilarea modulelor ➔ modulele obiect
  - legarea modulelor obiect (ediţie de legături sau "link-editare") ➔ programul executabil final
  - depanare program: cu un depanator

- Mediu de programare: integrează fazele de editare, compilare, editare legături şi depanare

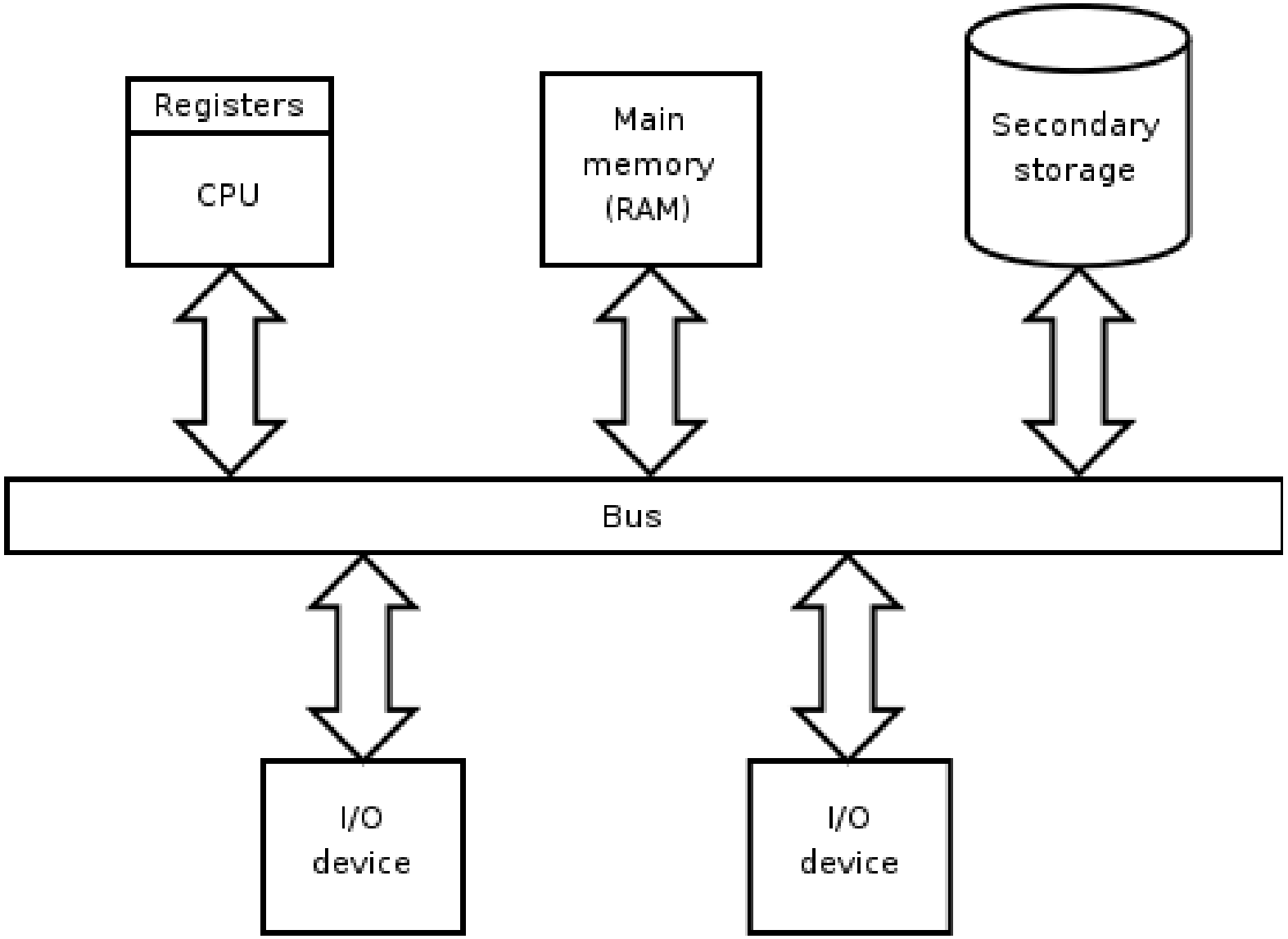
4. Reprezentarea informației în calculator



- DOAR CIFRELE BINARE 0,1 --- numite şi ***biţi***
- Aceştia se grupează în:
  - ***octeţi*** (bytes) sau
  - ***cuvinte*** (words) (2, 4, 8 octeţi)

- Organizarea informaţiei în biţi, octeţi, cuvinte → în toate componentele unui calculator (unitate centrală, memorie internă, dispozitive periferice)

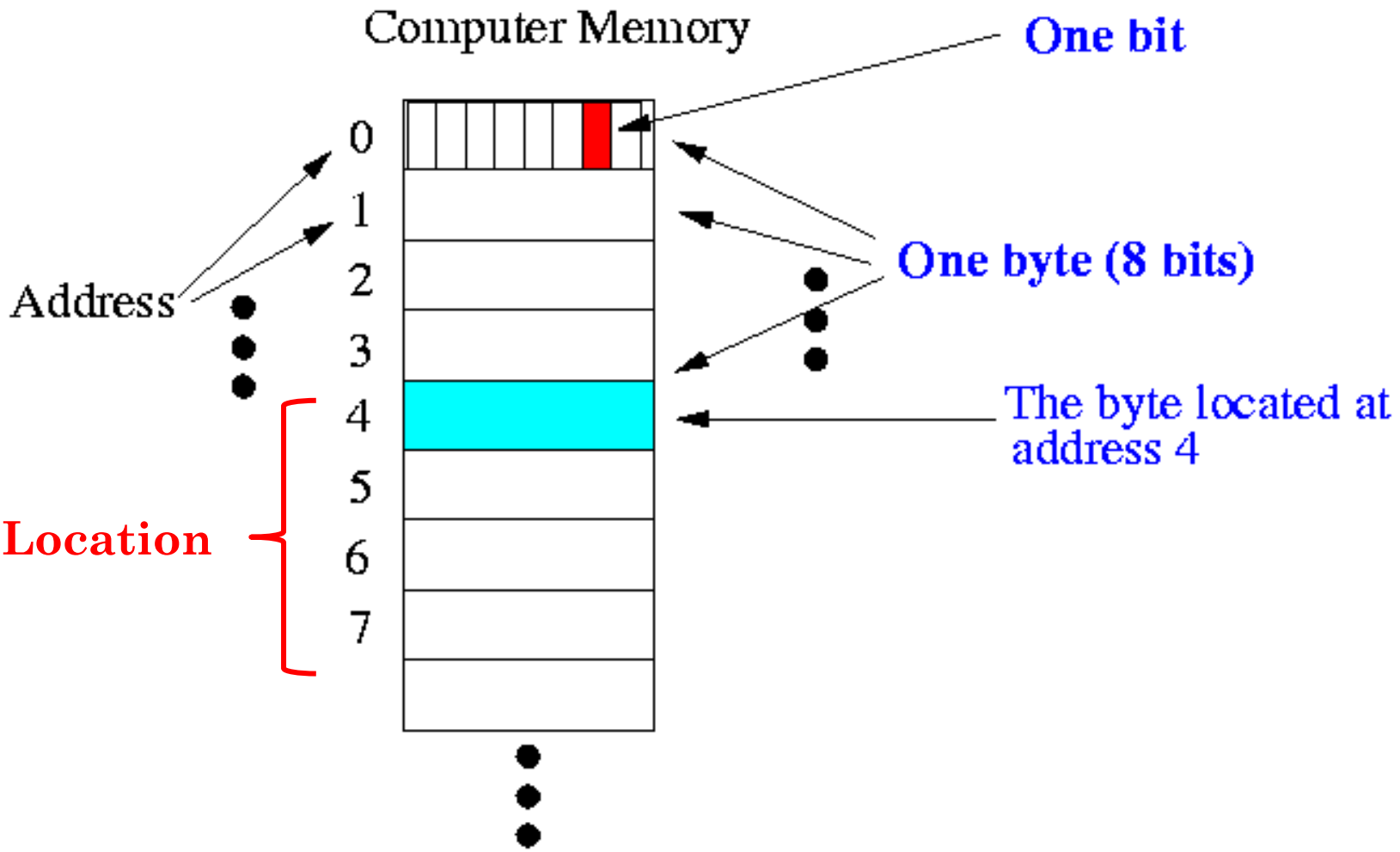
Arhitectura von neumann



Unde este stocată o anumită informaţie în memorie?

- ***adresa***: număr de ordine al unui octet (cuvânt) în memorie (L-value);

- ***locaţia***: zona din memorie rezervată unei anumite informaţii (octet sau cuvânt ce are o valoare estimată, R-value)


Computer Memory
One bit
0
1
2
3
4
5
6
7
Address
One byte (8 bits)
The byte located at address 4
Location

Cum poate informaţia de un anumit tip (text, valoare numerică, imagine, sunet, etc.) să fie reprezentată în calculator prin cifre binare (0 şi 1)?

- *R:* ***prin codificare***

- Cu ajutorul cifrelor binare se pot reprezenta direct numai numere naturale

- Reprezentarea numerelor întregi negative, a numerelor reale sau a caracterelor → folosind **convenţii de reprezentare**

- Codul:
  - o corespondenţă între simbolurile a două alfabete, având drept scop trecerea de la o formă de reprezentare a informaţiei la alta

- La calculatoare - codificarea:
  - realizează reprezentarea informaţiei într-o formă accesibilă calculatorului pornind de la forma uzuală de reprezentare

Cod alfanumeric

- Se foloseşte pentru reprezentarea caracterelor (cifre, litere şi alte caractere)
- **Codul ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)
  - foloseşte 7 cifre binare pentru a reprezenta cifre zecimale, litere mari şi mici, semne de punctuaţie, operatori aritmetici şi logici, simboluri pentru controlul comunicării şi simboluri pentru editare
- Pentru reprezentarea caracterelor speciale → codul ASCII8 (ASCII extins) pe 8 cifre binare
- Exemple:

A - 41h	a - 61h	0 - 30h
B - 42h	b - 62h	1 - 31h
LF - 0ah	CR - 0dh	ESC - 1bh

Cod alfanumeric

- **Codul UNICODE (din 15 Septembrie 2021 – 14.0.0):**

https://www.unicode.org/versions/Unicode14.0.0/
standard pe **32 de biți** ce permite reprezentarea și manipularea textelor în majoritatea sistemelor de scriere

(144,697 caractere în prezent)

-8 biți (UTF-8), 16 biți (UTF-16) și 32 biți (UTF-32)

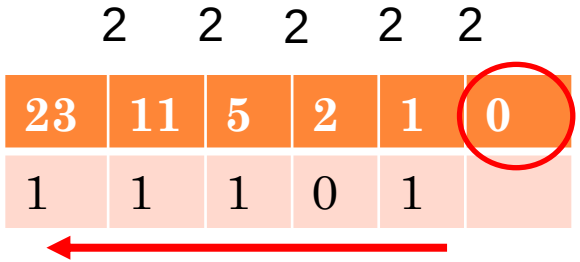
"UTF" - Unicode Transformation Format (modul de codificare)

Fiecare dintre cele 3 forme de codare oferă un mecanism pentru reprezentarea caracterelor Unicode, fiecare prezintă anumite avantaje în medii diferite

! poate fi folosit și în limbajul C/C++ prin setarea unei opţiuni

Cod complement

- Numerele întregi pozitive sunt reprezentate în calculator în binar, **pe multiplu de 8 biţi**
- Conversia numerelor întregi pozitive din baza 10 în baza 2 – regula împărţirii cu baza
  - împărțiri succesive la 2, până când câtul împărțirii este egal cu 0
  - restul fiecărei împărțiri = un simbol din reprezentarea numărului binar
  - rezultatul se obține prin considerarea simbolurilor în ordine inversă (de la ultimul rest la primul)



Rezultat pe 8 biţi: 00010111

Cod BCD (Binary Coded Decimal)

Permite codificarea binară a numerelor zecimale, fără a le transforma efectiv în binar

- Fiecare cifră zecimală este codificată separat în binar
- Exemplu: 245 0010.0100.0101

- In general se poate folosi structura:

  | Semn | Nr. cifre | Cifre...|

- Exemplu: -12345

  ***1111*.*0101*.0001.0010.0011.0100.0101**

Formatele de reprezentare BCD în virgulă fixa şi mobilă - domeniile financiar, comercial şi industrial (evită erorile prin rotunjire)

Reprezentarea numerelor întregi negative

- Deoarece nu se pot reprezenta semnele +, - se foloseşte o cifră binară (prima din stânga) pentru semn, cu următoarea convenţie:
  - 0 - pozitiv
  - 1 - negativ

- Există mai multe posibilităţi, dintre care s-a impus reprezentarea în C2 (Complement față de 2)

Reprezentarea în complement faţă de 1 (C1)

- Se reprezintă numărul pozitiv în baza 2
- se complementează fiecare cifră binară (0->1 şi 1 ->0)
- Ex. reprezentarea numărului întreg -7

$$\text{POZ: } (7)_{10} = (00000111)_2$$

$$\text{C1: } (-7)_{10} = (11111000)_2$$

Reprezentarea în complement faţă de 2 (C2)

- Se reprezintă numărul în C1
- Se adună numărul 1 în binar

$$\text{POZ: } (7)_{10} = (00000111)_2$$

$$\text{C1: } (-7)_{10} = (11111000)_2$$

$$+1$$

$$\text{C2: } (-7)_{10} = (11111001)_2$$

N = 1 0001
C1 1110 + 1
C2: 1111 → -1

N = 0 0000
C1 1111 + 1
C2: 0000 → 0 (ultimul 1 se pierde)

N = 7 0111
C1 1000 + 1
C2: 1001 → -7

N = 8 1000 (!!! un număr pozitiv nu poate fi reprezentat cu 1 pe prima poziţie (pozitia de semn))
C1 0111 + 1
C2: 1000 → **-8**

OBS: Pentru 4 biţi : $-2^3 <= N <= 2^3 - 1$

In general n biţi : $\mathbf{-2^{n-1} <= N <= 2^{n-1} - 1}$

Operaţii în C2

- *Adunarea:*
  - se adună cele două numere în mod obişnuit, cifră cu cifră, inclusiv pentru poziţia de semn, ignorând transportul de la poziţia de semn
- *Scăderea:*
  - se adună descăzutul cu C2 al scăzătorului
- *Inmulțirea/Impărțirea:* reprezentarea în C2 a numerelor

- La efectuarea de operații poate să apară fenomenul de *depășire aritmetică* (Arithmetic overflow) când se depășește domeniul de valori alocat numărului

- Exemple :

4	**Ad:**	0100+	**Sc:**	0100+	
3		0011		1101	C2(-3)
		------		------	
		0111 = 7	1←	0001	

Inmulţire 0100 *
0011

0100 +
0100

1100 = 12

Inmulțire:

–4x3 = –12

4: 00000100;

-4 (C2) → inversare și adunare 1: 11111011 + 1 = 11111100

3: 00000011

înmulțire: 11111100 *

00000011

11111110100 eliminare biții cei mai semnificativi

11110100 => bitul cel mai semnificativ este 1, deci e un număr negativ; se inversează și se adună 1: 00001011 + 1 = 00001100 = 12_{10}

=> 11110100 = –12

Reprezentarea numerelor reale

- Orice număr real X poate fi exprimat in baza *b* sub forma:

$$X = X' * b^{exp}$$

  - unde:
    - *X'* este mantisa (semnificant) şi dă precizia de reprezentare
    - *exp* este exponentul şi dă ordinul de mărime

  - Diferite standarde de reprezentare
  - *Mantisa normalizată:* un singur digit se află înaintea punctului zecimal
  - Acum în C++: un singur bit (cu valoarea 1) înaintea virgulei în reprezentarea binară (baza 2)

Ex: $5.75_{10} = 101.11_2 * 2^0 = 1.0111_2 * 2^2$

- Formatul IEEE pe 32 biţi : (754)

  31 30 … 23 22 … 0

  |S | Exponent | Mantisa |

- Domeniul de reprezentare:
  - $+-(3,4x10^{-38} ... 3,4x10^{+38})$
- Precizia: 7 cifre zecimale
- Această reprezentare se mai numeşte şi **reprezentarea în *virgulă mobilă*** sau ***flotantă***

- Tipuri:
  - simplă precizie (32 biţi)/ precizie 7 cifre zecimale
  - dublă precizie (64 biţi)/ precizie 15 cifre zecimale
  - precizie extinsă (80 biţi)

- **Reprezentarea pe 32 biţi** – tipul float în C

- 1 bit: semnul (0 → pozitiv, 1 → negativ)
- 8 biți: exponent + 127
- 23 biți: partea fracţionară în reprezentare binară

- **Reprezentarea pe 64 biţi** – tipul double în C

- 1 bit: semnul (0 →pozitiv, 1 →negativ)
- 11 biţi: exponent + 1023
- 52 biți: partea fracţionară în reprezentare binară

  - Limitarea preciziei → nu se pot reprezenta toate cifrele.
  - Aproximările făcute → *erorile de rotunjire* (rounding errors)

- Pentru numărul: $\mathbf{5.75}_{10} = \mathbf{101.11}_2 * 2^0 = \mathbf{1.0111}_2 * 2^2$
- Deoarece normalizarea oricărui număr binar (excepție 0) este de forma **1**.xxxxx, 1 din față nu este necesar ->1 din față nu este salvat în calculator (hidden bit) -> spatiu suplimentar de 1 bit – posibilitatea unei precizii mai mari
- Reprezentarea în calculator

  - S = pentru ***Semn*** = 0 (număr pozitiv)

  - Exponent binar → $2^K = 2^{\text{Exponent -127}}$

    Exponent = 2 + 127 = 129 = $(1000\ 0001)_2$
- Mantisa (întreagă) 1.0111
- ***Mantisa*** (fără bitul ascuns) 0111
- Rezultat: 0 10000001 01110000000000000000000

= 0100 0000 1011 1000 0000 0000 0000 0000

- **4 0 B 8 0 0 0 0** (hexazecimal)

- **Erori de reprezentare (rotunjire)**
  - Exemplu: valoarea lui π are un număr infinit de zecimale
  - C/C++ poate stoca maximum 64/80 de biți de informație pentru o valoare reală
  - rezultă o eroare de aproximare a numărului exact

- **Erori de depășire**
  - presupunem că avem două numere întregi stocate în memorie pe câte 4 biți fiecare -> valoarea maximă stocată = 15
  - rezultatul adunării celor două numere doresc să îl stochez tot pe 4 biți
  - care este rezultatul adunării 15+1?

5. Date şi tipuri de Instrucțiuni în Programare

- -Tipuri de **date elementare:**
  - **-** numerice
  - - logice
  - - caracter
  - - pointer
  - - referințe
- -Tipuri de **date agregate** (structurate):
  - - tablouri
  - - șiruri de caractere
  - - liste
  - - articole
  - - fișiere

INSTRUCȚIUNI

- - asignare
- - condițională/generalizată (**if, switch**…case)
- - cicluri (anterioare, posterioare): **while**(), **for** (…), **do…while**()
- alte tipuri

De reținut…

- Ce este un algoritm
- Ce este un limbaj de programare
- Etapele dezvoltării unui program
- Cum este reprezentată informația într-un calculator (numere întregi pozitive și negative – reprezentarea in C2)
- Cum sunt reprezentate numerele reale (reprezentarea in virgula mobila, mantisa si exponent)
- Cum se realizează operaţiile aritmetice în binar
- Care sunt datele și instrucțiunile de bază în programare