Microprocesoare

Suport de Curs

48656c6c6f2021

01001000011001010110110001101100011011110010000000100001

Hello !

Anca Apătean

Conf. dr. ing., UTCN

Dep. Comunicatii

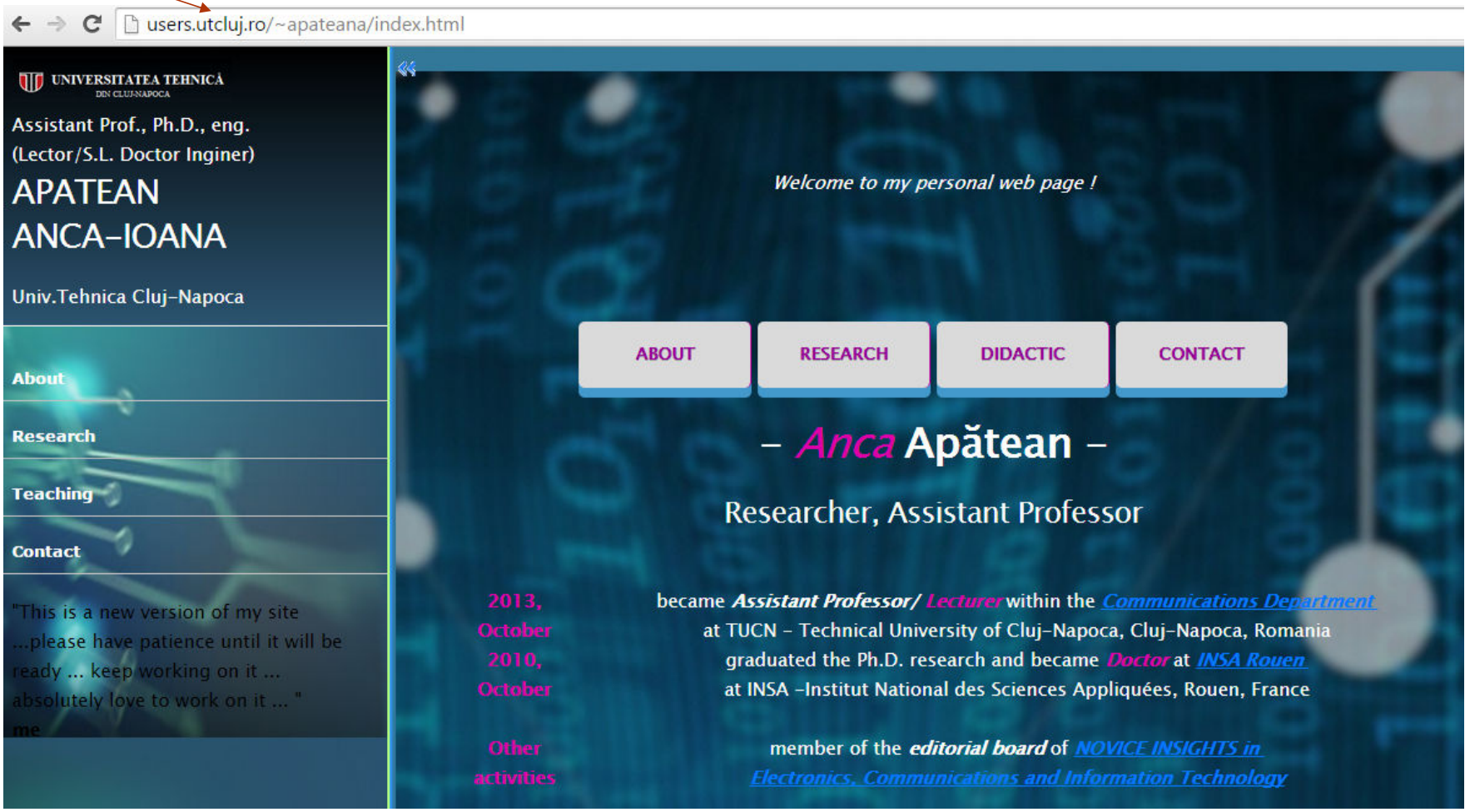
Anca Apătean

- ***Conf. dr. ing.*** UTCN, Dep. Comunicatii

anca.apatean@com.utcluj.ro
http://users.utcluj.ro/~apateana/

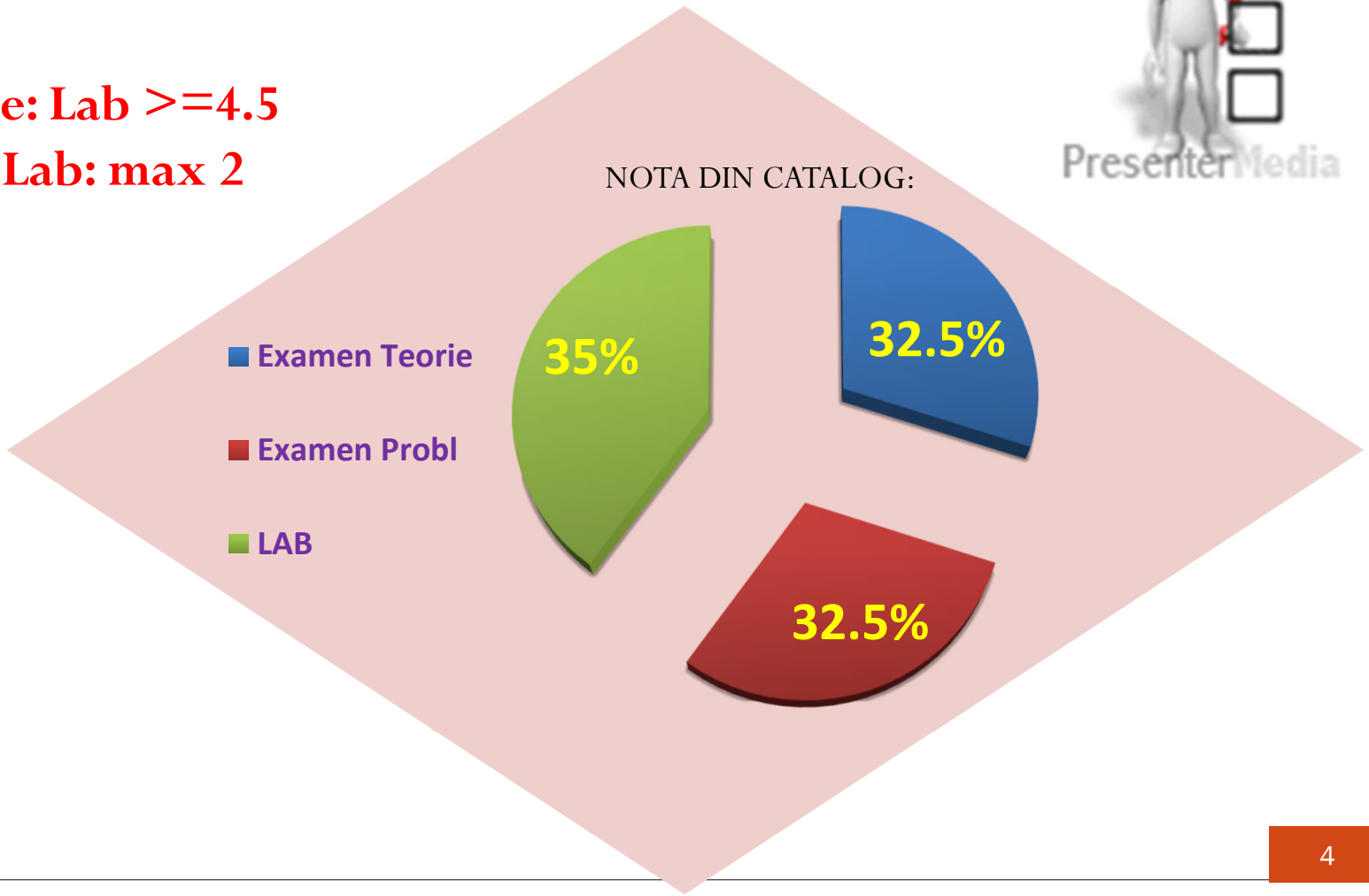
- Lucrari de ***DIPLOMA*** -> comisia TST -> exemple teme
  - Image, speech, signal Processing
  - ML, DM, business analytics, IoT
  - cu platforme de dezvoltare Arduino, Intel Galileo, Raspberry Pi, FRDM KL25Z + shielduri

Pe Site: users.utcluj.ro/~apateana



Conditie: Lab >=4.5

Nr. abs. Lab: max 2



Ce tipuri de ***computere*** / ***calculatoare*** / ***sisteme de calcul (SC)***

ne inconjoara

in prezent ?

Who said that ...

- *"All the problems of the world could be settled easily if men were only willing to think."*
  - **Thomas Watson – fondator IBM**

- **THINK - > motto**

- **men = robot ???**

↓

"WATSON "

WHAT is WATSON?

Answer: an intelligent ROBOT


THINK
$300,000
Who is Stoker?
((FOR ONE WELCOME OUR NEW COMPUTER OVERLORDS))
$1,000
$1,000,000
Who is Bram Stoker?
$ 17,973
$200,000
WHO IS BRAM STOKER?
$5600
WATSON VS. HUMANS
Round | Watson | Rutter | Jennings
1 (Mon.) | $5000 | $5000 | $200
2 (Tues.) | $35,734 | $10,800 | $4,800
3 (Wed.) | $77,147 | $21,600 | $24,000
Final prize | $1,000,000 | $200,000 | $300,000

- **THINK**

Anca APATEAN UTCN

“Il cumparam?” E cam vechi …nu ? Ce ati vazut recent “la magazin”?

Core i7 ? Care Gen.? Ce este ECS LIVA X???

ECS LIVA X - ???

= “all the **hardware** you need to build **a mini PC**”

“Redefineste dimensiunea PC-ului !”

http://www.legitreviews.com/ecs-liva-x-64gb-mini-pc-kit-review_157152

“Turn your TV into a PC”

Dar Galileo 2nd Gen? Raspberry PI ?

- **ECS LIVA X** -> "LIVA" ('**li**ving' + 'life (Vi**va**)' => LIVA)
  ECS LIVA X: sub $125 - Intel BayTrail-M **SoC**

Intel Celeron Processor N2807,

memorie 2GB 1333MHz DDR3L, 32GB sau 64GB memorie eMMC (embedded flash)+ wireless card cu suport 802.11b/g/n si Bluetooth 4.0. -> suporta Windows 8/8.1 sau Linux

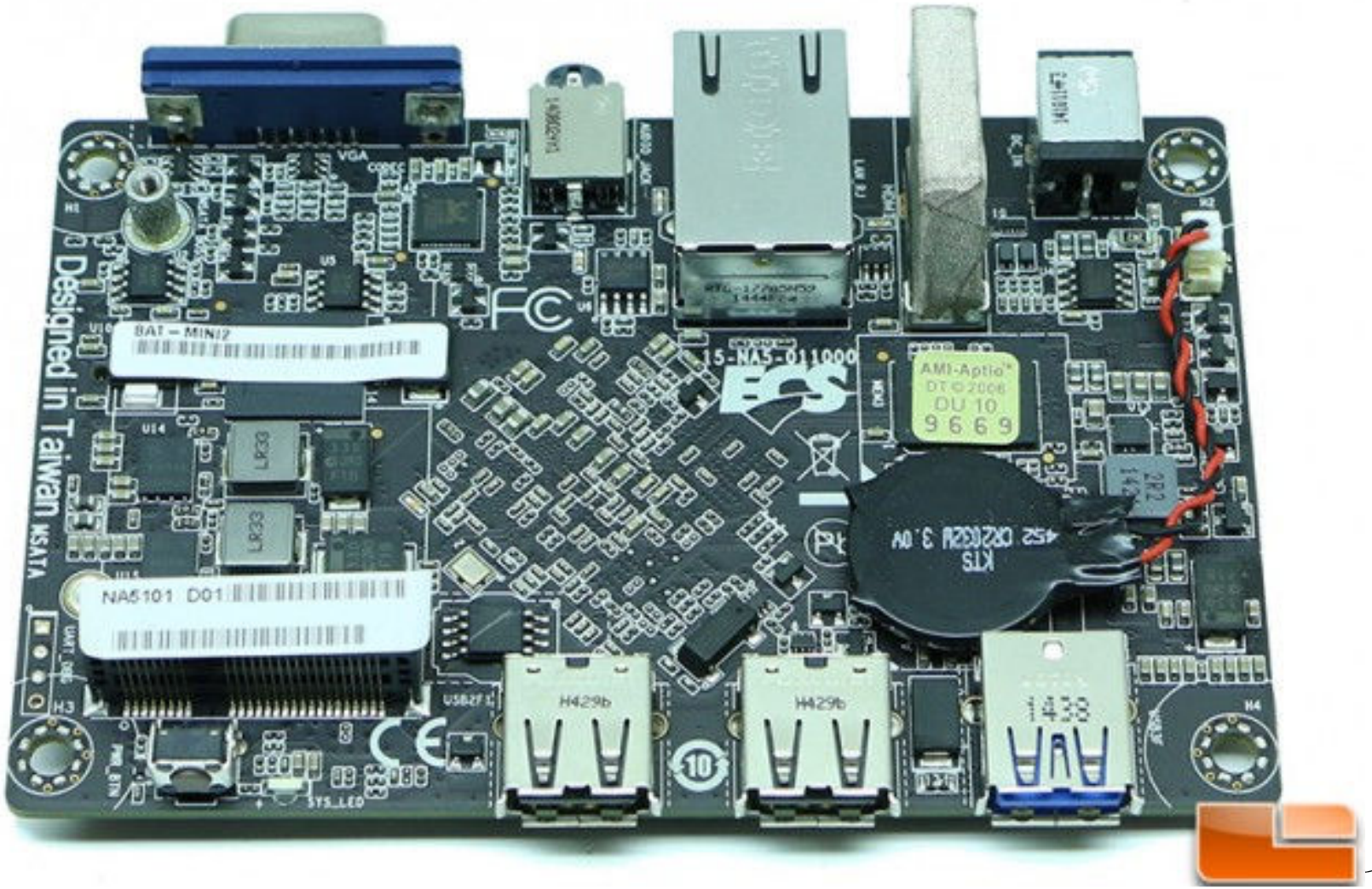
2nd gen (2014): ECS -Intel **Celeron** N2808, **22nm dual-core processor**,

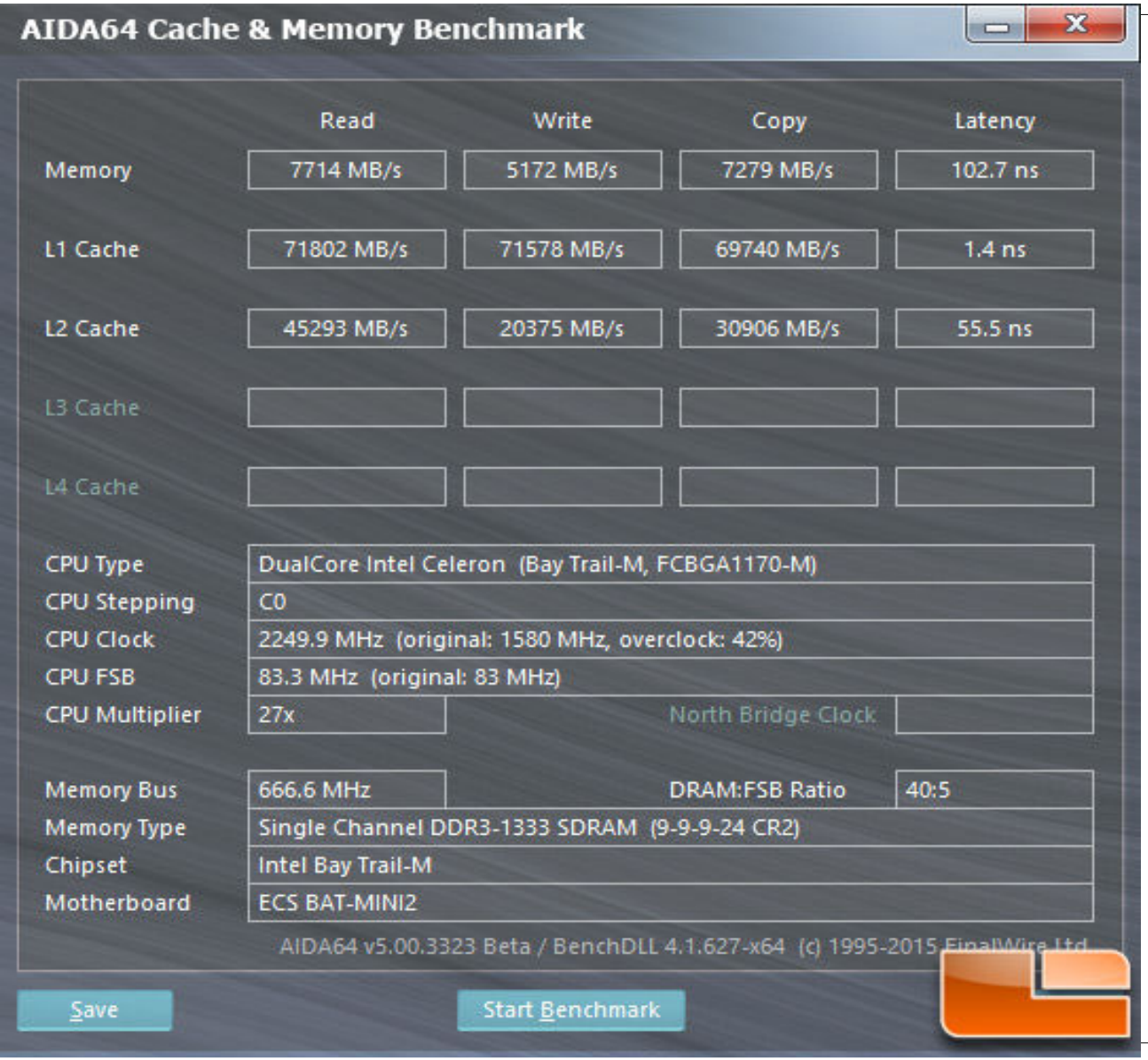
clock: 1.58GHz + turbo clock: 2.25 GHz.

- 1 sau 2 porturi USB 2.0 + 1 port USB 3.0 +

12V DC_IN port, HDMI port, Realtek RTL8111G Gigabit Fast Ethernet port, Realtek ALC283 3.5mm combo audio jack (Line Out & Mic In w/ optional adapter) + VGA port.

ECS LIVA X


VGA
Designed in Taiwan
MSATA
BAT - MINI2
NA5101 D01
AMI-Aptio
H429b
H429b



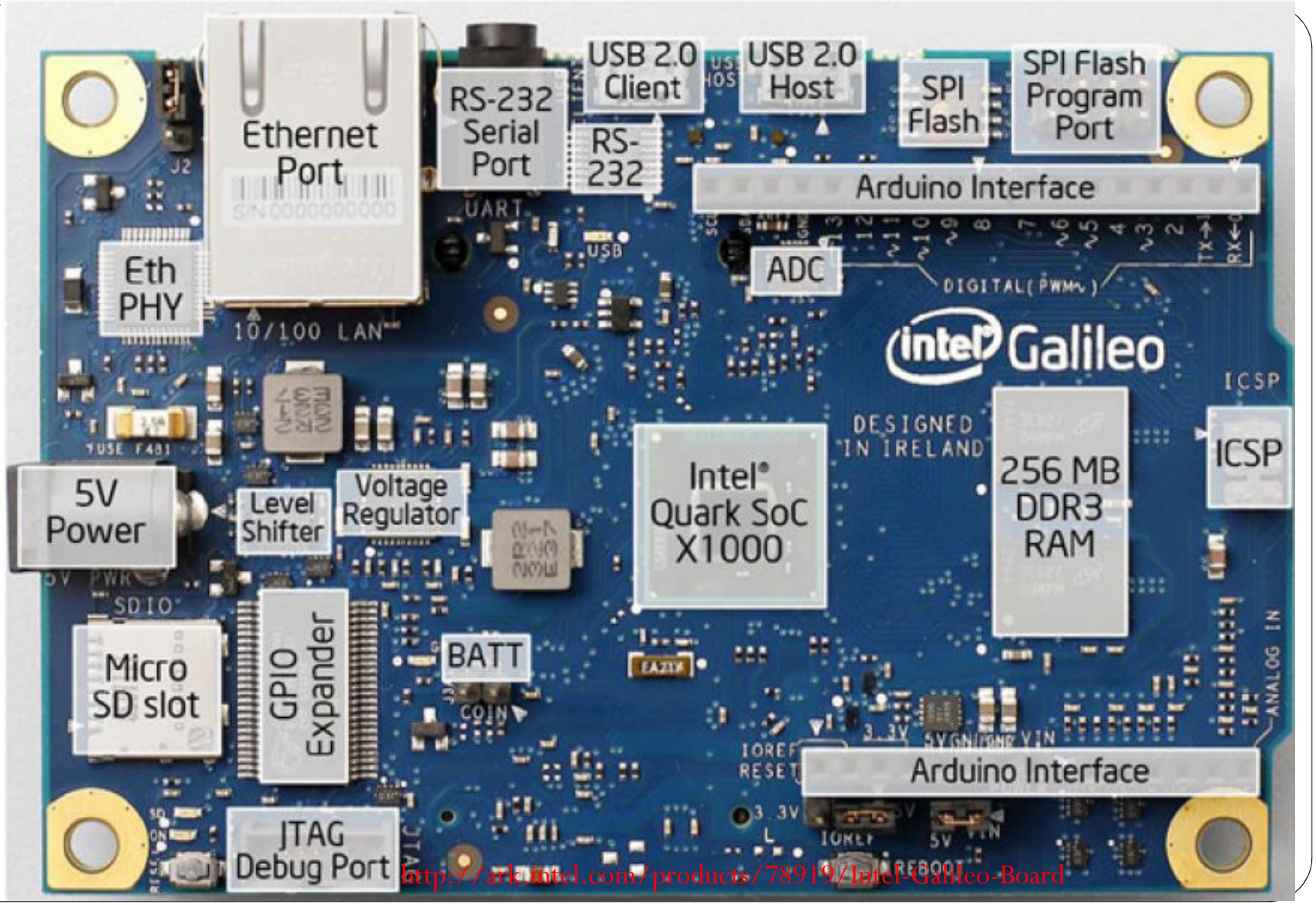
Cpuid ...

AIDA64 :

- viteza memoriei la citire: ~7,700 MB/s
- viteza memoriei la scriere: ~5,200 MB/s
- latenta memoriei: 102.7ns

http://www.legitreviews.com/

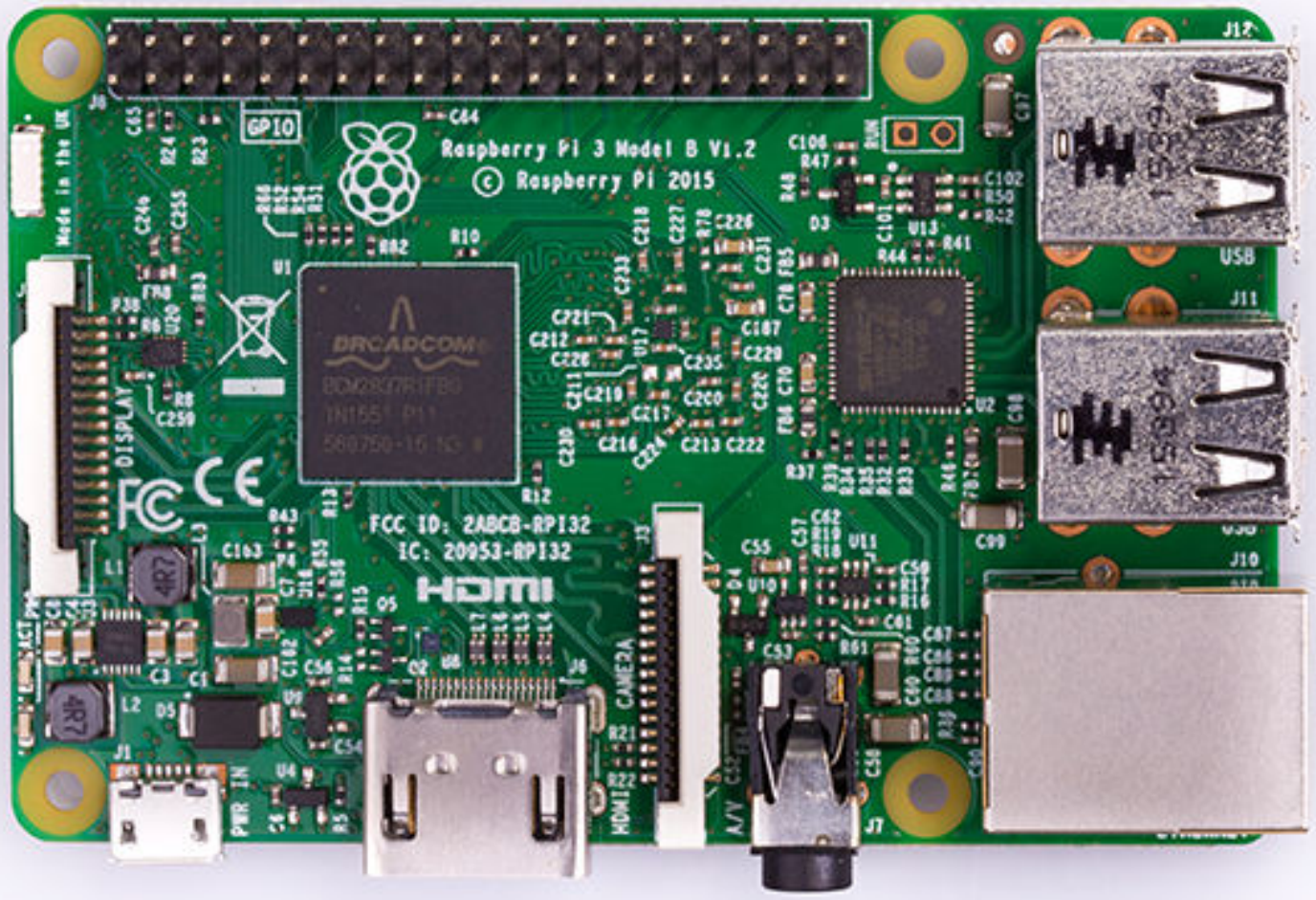
INTEL GALILEO



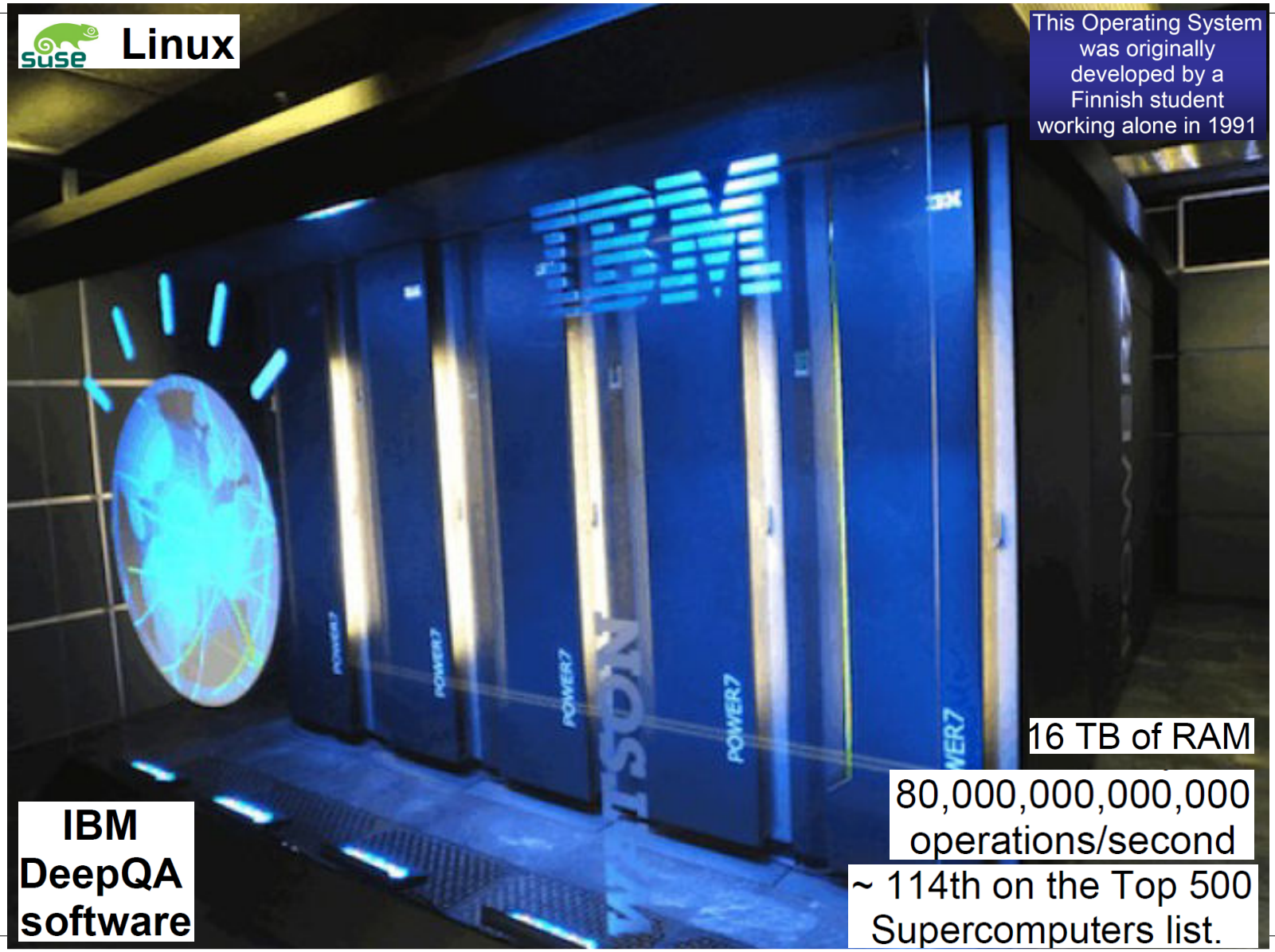
Studiu: Placa Raspberry Pi

- **Procesor la 1.2GHz pe 64 biti quad-core ARMv8 (de tip RISC)**
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

- (asem. cu R Pi 2):
- **1GB RAM**
- 4 porturi USB
- 40 pini GPIO
- Port Full HDMI
- Port Ethernet
- 3.5mm audio jack si
- composite video
- Interfata cu CAMERA (CSI)
- Interfata la Display (DSI)
- Slot pentru card Micro SD
- Nucleu pt grafica 3D VideoCore IV



Sa ne reamintim de Watson ...


Linux
This Operating System was originally developed by a Finnish student working alone in 1991
IBM
POWER7
16 TB of RAM
80,000,000,000,000 operations/second
~ 114th on the Top 500 Supercomputers list.
IBM DeepQA software

Watson are 90 Servere Power750:

2880 nuclee x(1 Server/32 cores) = 90 Servere;

10 Racks of IBM POWER 750 Servers

Hardware-ul lui Watson e capabil de aprox. 80 TeraFLOPs

are 16 TB RAM – de 2000 ori mai mult RAM ca un desktop (8GB)
- poate stoca echivalentul a 4000 DVD-uri

Watson e al 114-lea in lista *Top 500 Supercomputers*

Cost estimat de vanzare pentru un server IBM Power 750 : $34,500 USD.
pentru 90 Servere → ~ $3 Milioane USD

Arhitectura unui Computer

Anca APATEAN UTCN

vs

Organizarea unui Computer

Cum se proiecteaza un computer?
Cum functioneaza/lucreaza un computer?

Cum se proiecteaza un computer?

Anca APATEAN UTCN

=> Arhitectura: "Computer architecture"

- Aspectele *logice* ale sistemelor vazute de catre programator – ***detalii software***
- Atributele care vizeaza executia logica a unui program
- Ex: setul de instructiuni, nr. de biti folosit pt a reprezenta diferite tipuri de date (numere, caractere), tipurile de date, formatul instructiunii, modurile de adresare ale memoriei, mecanisme I/O.

Cum functioneaza/lucreaza un computer?

=> Organizare: "Computer organization"

- Aspectele *fizice* ale SC (computerelor) - ***detalii hardware*** transparente programatorului
- Unitatile operationale si interconexiunile lor ce realizeaza specificatiile arhitecturale
- Ex: schema circuitului, componentele interne, interfatarea intre computer si periferice, semnalele de control, tipurile (si tehnologia) de memorie
  - – conectori interni si externi pe PB, sloturi, interfete, cabluri, placi, dispozitive, etc

Componentele unui sistem de calcul

=

Sistem de calcul

=

HARDWARE (partile fizice)
= componentele electronice

SOFTWARE (partile logice)
= instructiuni, programe

Unitate de procesare

Memorie

Dispozitive intrare/iesire

Software de Sistem

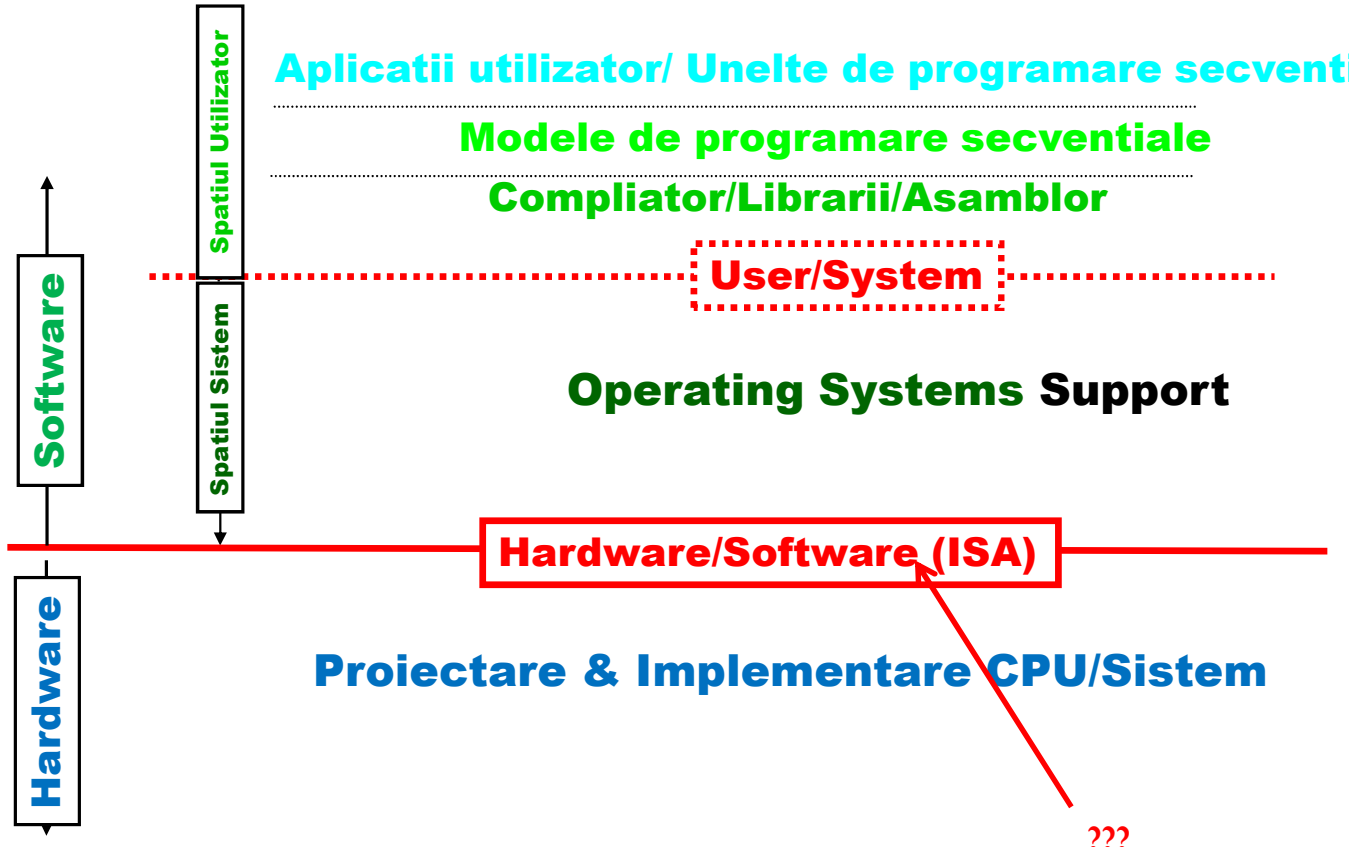
Sistem de operare

Aplicatii Software

USER

Abstractizarea *Arhitecturii Calculatoarelor* "Conventionale"

"Conventional " = Secvential / un singur CPU



Nivele de Abstractizare CRITICE

Delimitarea User / System (Utilizator/Sistem):
Ce se realizeaza în *spațiul utilizator.*
Care e suportul asigurat de **S.O.** (*în spațiul sistem*)
la rularea programelor utilizator.

- **(dpdv HARDWARE)**

La nivelul cel mai scazut:

- un computer (sistem de calcul SC) = ??? = un dispozitiv format din 3 parti principale:
  - **procesor**
    - - pt a interpreta si executa programele
  - **memorie**
    - – pt a stoca datele si programele
  - **echipamente periferice de I/O (intrare/iesire) raportat la SC**

    = reprezinta un mecanism pentru transferul datelor

    in/din interiorul sistemului de calcul din/in exterior

    (dinspre SC inspre utilizator sau invers)

    - Toate acestea implementate cu ajutorul dispozitivelor electronice, mecanice, optice, etc

- **(dpdv SOFTWARE)**
- **Cea mai mică unitate de informaţie: bit (engl.bit)**
- - mai multe forme de *implementare practică* a biţilor
  - comutatoare, relee şi diode, tranzistoare, circuite integrate = *hdw*
  - toate bazate pe acelaşi principiu:
    - sa permita trecerea curentului electric = 1 (led on)
    - sa blocheze trecerea curentului electric = 0 (led off)

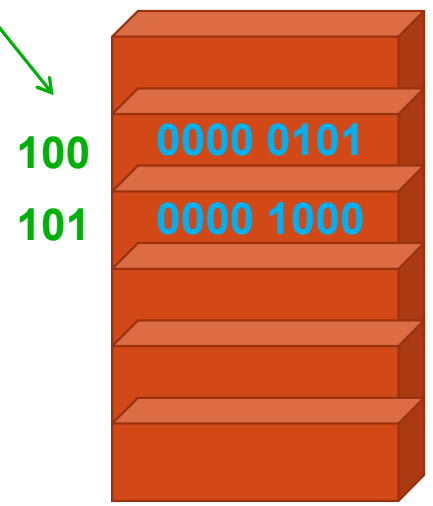
    - o singură cifră (0/1) conţine prea puţină informaţie pentru a fi utilă.
    - în scrierea poziţională numerele sunt şiruri de cifre,
    - **ex: numarul 1234 - o mie doua sute treizeci si patru**

      => ideea de a reprezenta numerele prin ***şiruri de biţi***
    - necesitatea grupării biţilor. (0000010011010010) => 0000 0100 1101 0010

      + o standardizare a dimensiunii şirurilor de biţi

      => notiunea de **octet** (engl. **BYTE**)

Adresa din memorie	Celula de memorie	*Valoarea* stocata
100	0000 0101	5
101	0000 1000	8



1. Ce este un microprocesor?
Care este rolul lui într-un sistem de calcul (SC)?

- **Microprocesorul**

= Unitatea centrală de procesare – UCP

în engleză: **CPU - Central Processing Unit**

= elementul de bază al unui sistem de calcul (SC), *(Este element de baza/ secundar?)*

= ***un cip complex***, plasat de obicei pe placa de bază (PB), *(Unde se afla?)*
în unitatea centrală a sistemului.

(La ce se foloseste?)

- asigură procesarea datelor (interpretarea, prelucrarea şi controlul acestora),
- supervizează transferurile de informaţii
- controlează activitatea generală a celorlalte componente care alcătuiesc SC.

Microprocesor 8086 -> Simulator EMU8086

- În gen., orice SC conţine **o parte hardware** și **una software**
- vom insista pe ***aspectele software ale UCP***: **http://www.emu8086.com/**
- problema proiectării ***aplicaţiilor posibile ->*** interacţiunea cu utilizatorul
- abordarea de acest gen (cu o implicare minimală a aspectelor hardware),
  în gen. se optează pentru *folosirea unui simulator*.

- ***Simulatorul EMU8086*** este un emulator de ***microprocesor 8086***
- vizăm abordarea *arhitecturii familiei de procesoare x86* pe 16 biţi

- Simulatorul EMU8086 execută *programele* pe o Maşină Virtuală,
- emularea hardware-ului real: ecranul monitorului, memoria şi dispoz. de intrare/ ieşire - pot fi utilizate, accesate sau vizualizate din EMU, ca *dispozitive virtuale*

*(*pe un sistem cu procesor Core i3 (procesor pe 64 biţi), a fost emulat sau simulat un procesor 8086*)*

De ce microprocesorul 8086?

- ***Procesorul 8086 este procesor pe 16 biţi*** (proiectat de Intel)
-
- =“***baza*** unei întregi familii de microprocesoare”

  (cea care a luat ulterior numele de ***familia de procesoare x86***).
- =“o ***referinţă*** în domeniu” pentru procesoarele care i-au urmat,

  inclusiv cele din familia Pentium şi Athlon (chiar şi pentru cele actuale pe 64 biţi).
- Cu **simulatorul EMU8086** pot fi executate majoritatea instrucţiunilor Intel

  (pentru arhitectura de 16 biţi) şi chiar directive specifice MASM şi TASM.
- **Scop principal :**
  - **înţelegerea arhitecturii unui procesor pe 16 biti**
  - **scrierea primelor programe într-un limbaj de asamblare**

De ce Emu8086?

- ***EMU8086 suportă setul de instrucţiuni specific procesorului 8086 (familia x86).***
- ***Ce urmarim?***
  - Scrierea de programe care să folosească instrucţiuni x86,
  - obţinerea codului executabil cat mai rapid si usor
  - încărcarea programului spre execuţie si vizualizarea rezultatelor
- *facilitate cu ajutorul simulatorului*

- => Se încurajează ***scrierea primelor programe în limbaj de asamblare***,

  fără riscuri pentru sistemul gazdă și fără a ţine cont de **comenzile**

  (uneori greu de stăpânit de începători) ce ar trebui date în vederea

  obţinerii și apoi **execuţiei programului** folosind un **PROCESOR REAL**.

Ciclul complet de obţinere al unui fișier executabil în mod tradiţional

plecând de la ***cod scris în limbaj de asamblare*** implică următoarele etape:
asamblare, ***linkeditare***, ***depanare***, ***execuţie***.

Acest ciclu poate fi parcurs:

- 1) ***prin intermediul unui emulator*** (de ex. ***EMU8086***)

și atunci programele vor fi adaptate arhitecturii emulate,
în cazul de faţă a procesorului 8086;
astfel, vor rezulta programe care folosesc regiștri pe 16 biţi (AX, BX, …);

- 2) ***direct cu procesorul real***

și atunci *p*rogramele pot fi adaptate arhitecturii CPU real;
în prezent, de exemplu, se poate exploata la nivel de **Core i3, i5, i7**;
astfel, vor rezulta programe care folosesc regiștrii pe 16 biţi (AX, BX, …),
pe 32 biţi (**E**AX, **E**BX, …)
sau chiar pe 64 biţi (**R**AX, **R**BX, …).

2. Reprezentarea informaţiei în PC

(Curs 1+Curs 2) -> (Lab 1 + Lab 2)

- **2.1. Aspecte generale de reprezentare a informaţiei**
- ***Cum se citește/ interpretează informaţia stocată în PC ?***

(O valoare din memoria sistemului, de exemplu: 43 ?)

R:

interpretarea informaţiei poate fi diferită (depinde de context)

43h poate reprezenta o valoare de *temperatură*, *vârsta*, *o dimensiune* sau *un cod*

(codul ASCİİ al literei “C”)

+ o valoare scrisă în binar de forma **10000001**b

poate desemna ***numărul 129*** sau ***numărul -127***,

în funcţie de ***convenţia de reprezentare*** a numerelor: *fără semn* sau *cu semn*.

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei
- ***Cum se reprezintă datele din PC? Ce sistem de numeraţie se folosește?***

Oamenii -> ***sistemul de numeraţie zecimal*** pentru a se referi la date

(10 degete la mâini: cifrele 0,1,2,…,9).

Calculator (SC)? (**Cum este aici?**)

R: ***trece*** sau ***nu trece*** curent printr-o porţiune de circuit

=> ***sistemul binar***: 2 simboluri sau cifre binare: **0** şi **1**

2. Reprezentarea informaţiei în PC

- **2.1. Aspecte generale de reprezentare a informaţiei**
- ***Ce este bitul?***

R: Cea mai mică unitate de informaţie

(în engleză se scrie şi se pronunţă “bit”, la plural “bits” de la **b**inary dig**its**).

În timp - ***forme de implementare practică*** a biţilor: ***comutatoare***, ***relee şi diode***, ***tranzistoare***, iar în prezent ***circuite integrate***, toate s-au bazat pe acelaşi principiu:

- să permită trecerea curentului electric, analogie cu **bit = 1**,
- să blocheze trecerea curentului electric, analogie cu **bit = 0**,

întrucât bitul poate lua doar una din cele 2 valori: **0** sau **1** (nu permite/ permite).

2. Reprezentarea informaţiei în PC

- **2.1. Aspecte generale de reprezentare a informaţiei**
- ***Cum se pot grupa biţii pentru a fi mai ușor de manevrat?***

ideea de a reprezenta numerele binare prin *şiruri de biţi* scrise secvenţial

Exemplu: **4321** (îl citim *patru mii trei sute douăzeci şi unu*)

=> **1000011100001** -???? Cum il citim ? Il putem retine ?

R: Organismele autorizate -> standarde

=> noţiunea de ***octet*** (în eng. "byte") = un ***grup de 8 biţi***

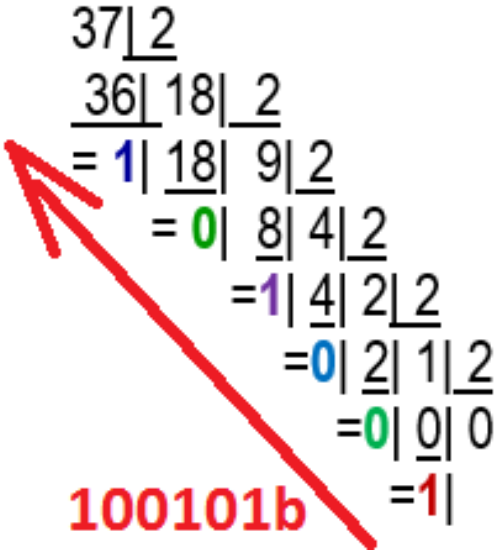
$\Rightarrow$ 1000011100001b = | ***00***0010000 | 11100001 | (scris in binar)

(biţii se grupează în octeţi întotdeauna pornind dinspre dreapta şi se completează – *biţii subliniaţi* - până la umplerea structurii de 16 biţi, cu valoarea bitului din extremitatea stângă, adică **0**).

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei
- ***Cum s-a ajuns de la 4321 la 0100001110000 1b?***

  (sau de la 37 la 0100101b?)



R:

- se efectuează împărţiri succesive la 2
- se culeg resturile în ordine inversă obţinerii lor,

- bitul de **0** apărând datorită faptului ca e scris ca ***un numar fara semn*** 4321 sau ***semnului pozitiv*** al numărului (daca se considera ***cu semn: +***4321) scris în zecimal.

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei
- ***Cum se poate trece mai ușor de la sistemul binar la cel zecimal sau invers?***

R: se utilizează sistemul de ***numeraţie hexazecimal***

e oarecum apropiat de ambele sisteme:

- permite exprimarea unor valori cu un număr rezonabil de cifre (asemănător sistemului zecimal) şi
- permite conversia relativ simplă în sistemul binar.

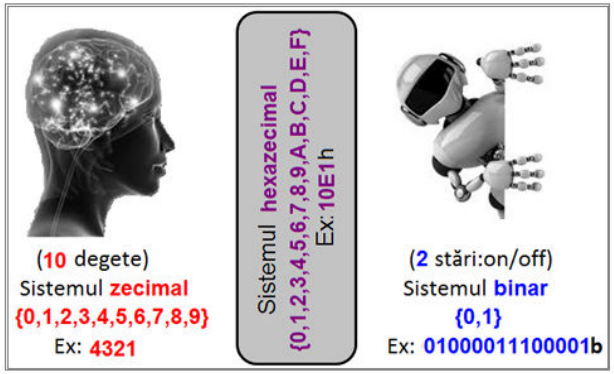


Figura 2.1 Legătura dintre sistemul zecimal, binar şi hexazecimal

⇒numărul din zecimal 4321 scris în binar 0001000011100001b

devine mult mai simplu de urmărit în hexazecimal: **10E1h**,

fiind *aproape* la fel de uşor de manevrat (de către noi, oamenii) ca şi cel scris în zecimal.

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei
- ***Cum s-a ajuns de la 0001000001110000 1b la 10E1h?***

R:

0001 | 0000 | 1110 | 0001|

Sistemul zecimal q=10	Sistemul binar q=2	Sistemul hexazecimal q=16	Sistemul octal q=8
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei
- ***Într-o anumită situaţie, care sistem de numeraţie e mai potrivit?***

R:

- noi oamenii gândim în **zecimal**

datorită structurii corpului nostru,

SC „gândesc” în sistemul lor,

adică în **binar**.

- Pt o comunicare cât mai ușoară între cele 2 entităţi, uzual se fol. sistemul ***hexazecimal***.

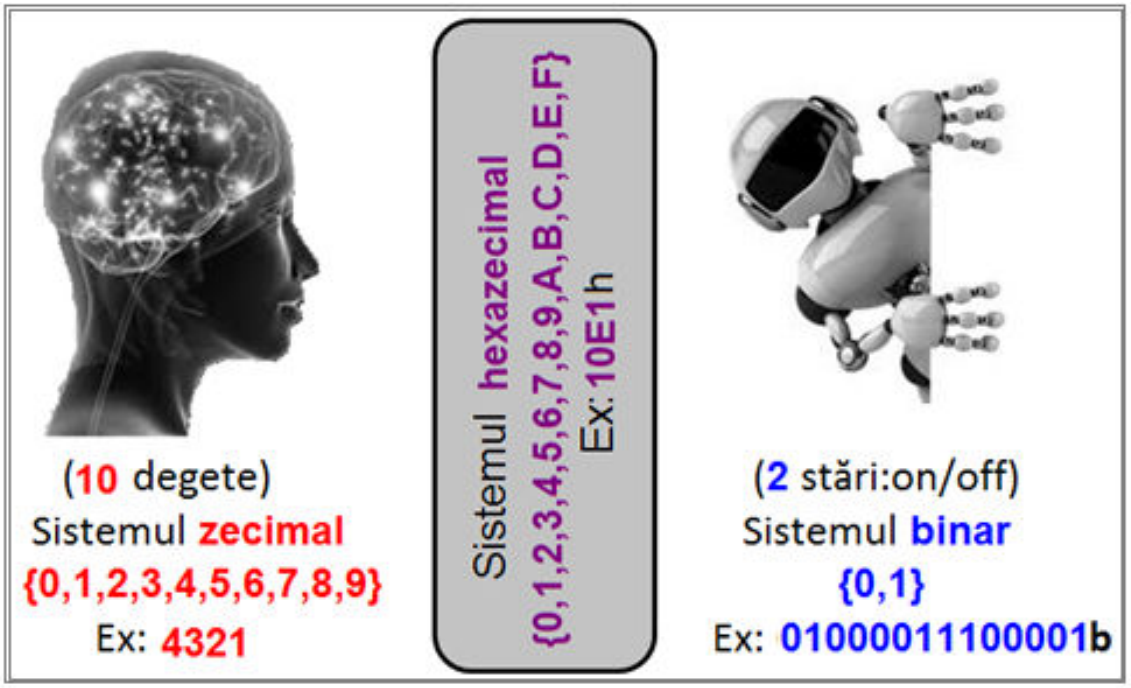


Figura 2.1 Legătura dintre sistemul zecimal, binar şi hexazecimal

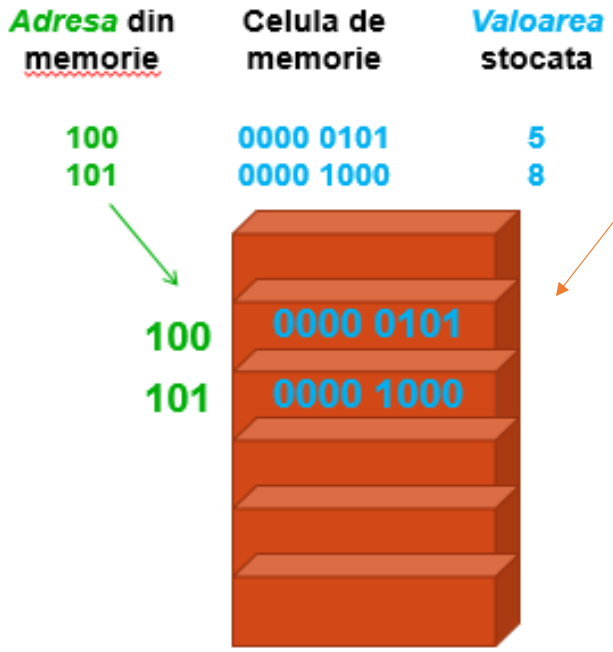
- Memoria PC-ului conţine doar numere sub forma unor secvenţe de 0 şi 1

  => se spune că SC stochează informaţia în format binar, nu zecimal.

Aceste secvenţe sau *înșiruiri de 0 și 1* sunt organizate

sub formă de ***octeţi*** sau ***grupuri de octeţi*** (multiplu de 2, deci 2^x octeţi)

în memorie sau în regiștrii interni ai procesorului.



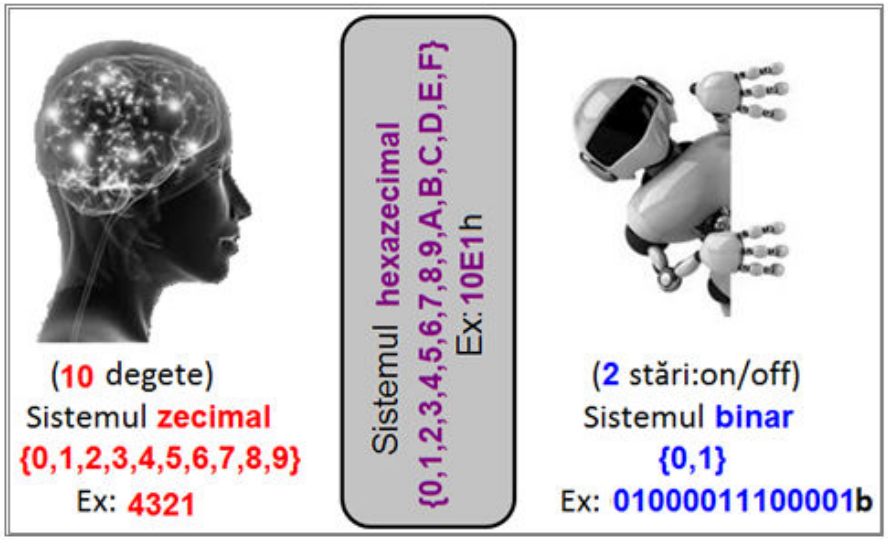


Figura 2.1 Legătura dintre sistemul zecimal, binar şi hexazecimal

2. Reprezentarea informaţiei în PC

- **2.1. Aspecte generale de reprezentare a informaţiei (cont.)**
- ***Ce este octetul ?***

R: ***Octetul*** **(*byte*)** este un şir (grup) de 8 biţi

= un standard unanim respectat.

Orice combinaţie de 8 biţi poate reprezenta un octet.

De exemplu: 01010101b, 11110000b sau 00011101, etc.

- ***Cum sunt ordonaţi/ identificaţi biţii în cadrul unui octet ?***

Noi vom folosi:

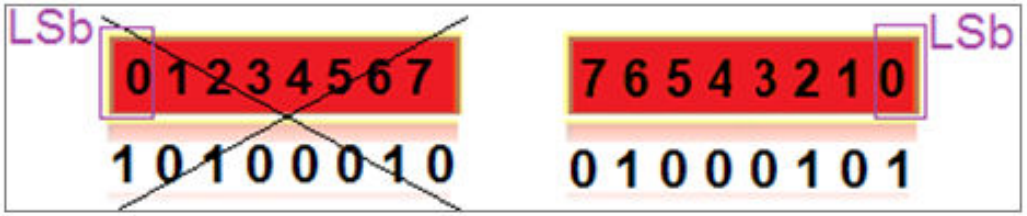


Figura 2.3. Scrierea corectă a biţilor în cadrul unui octet

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei (cont.)
- ***Câte combinaţii (sau numere binare) putem scrie folosind un octet ?***

R:

folosind un singur bit se pot reprezenta 2 valori (0 şi 1),

pe 2 biţi se pot reprezenta 4 valori (00, 01, 10, 11),

.

.

.

pe **un octet** se pot reprezenta 2^8 **numere**, adică **256 valori diferite**,

de exemplu numere întregi pozitive (naturale) de la 0 la 255.

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei (cont.)
- ***Care sunt submultiplii octetului ?***

- **Bit**
- Triada
- **Tetrada**

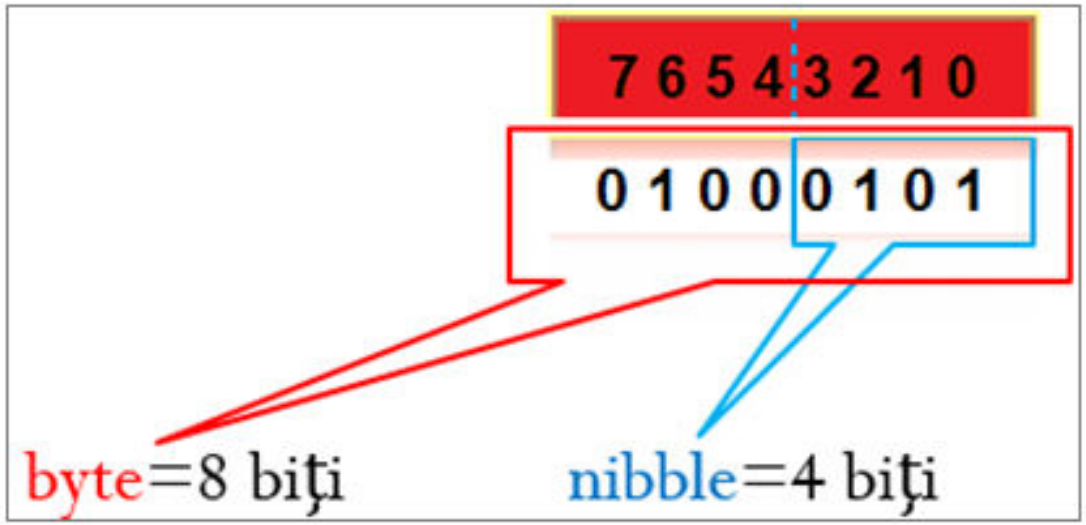


Figura 2.2 Reprezentarea nibble - submultiplu uzual al octetului

2. Reprezentarea informaţiei în PC

- **2.1. Aspecte generale de reprezentare a informaţiei (cont.)**
- ***Cum se reprezintă informaţia din memoria SC ?***
  - ***bit*** - poate fi 0 sau 1, adică 2^1 valori;
  - ***nibble*** - cu ajutorul unui nibble se pot reprezenta 2^4 =16 valori diferite;
  - ***octet*** - un număr de 8 biţi cu ajutorul cărora pot fi reprezentate 2^8 =256 valori diferite;
  - ***cuvânt*** - un nr de 16 biţi sau 2 octeţi = un cuvânt, nr. valorilor reprezentabile fiind 65 536= 2^{16};
  - ***dublucuvânt*** – un nr de 32 biţi sau 4 octeţi - permite reprezentarea a 2^{32} valori,
  - ***cvadruplucuvânt*** – un nr de 64 biţi sau 8 octeţi – se pot reprezenta 2^{64} valori.

- dimensiunile uzuale ale operanzilor în PC au fost: ***octet*** (în engleză *byte*),
  cuvânt (în engleză *word*),
  dublucuvânt (în engleză *doubleword*) şi
  cvadruplucuvânt (în engleză *quadword*)

- ***Cum se reprezintă informaţia din memoria SC ?***

octet 7 ... 0

Se pot folosi, in functie de CPU:

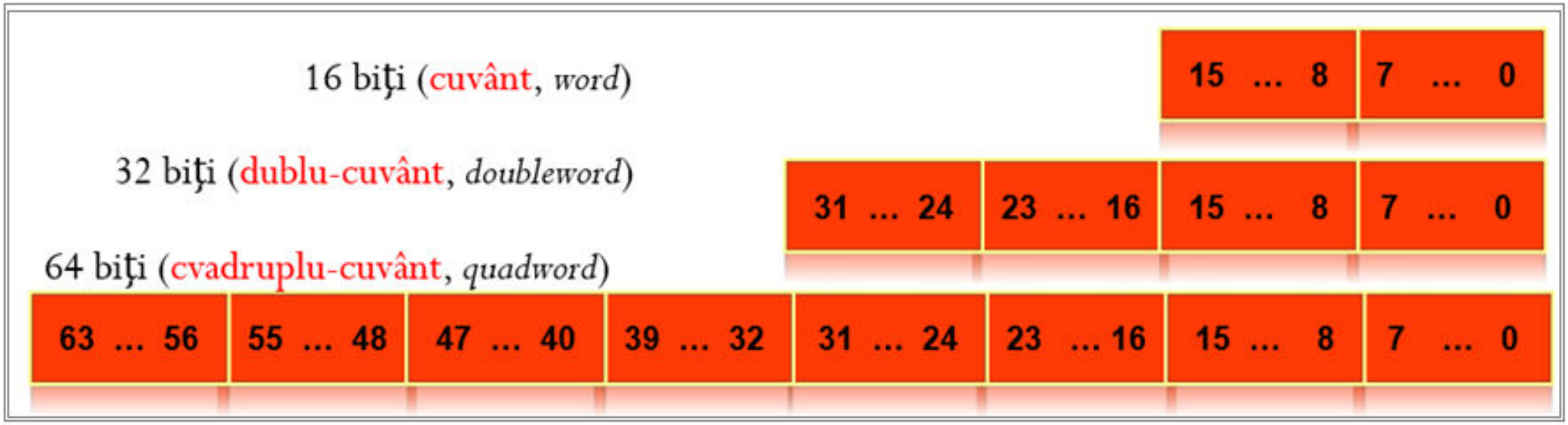
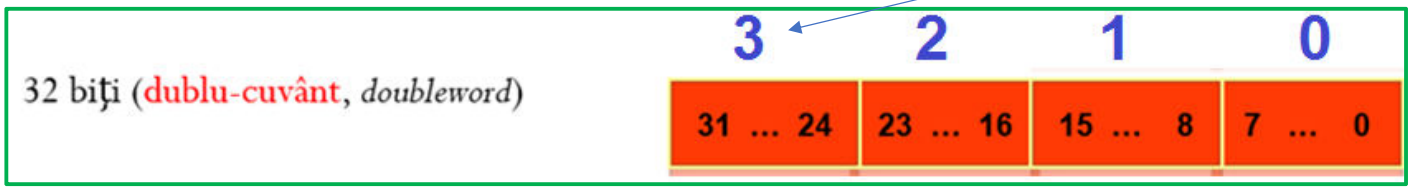


Figura 2.4 Multipli uzuali ai octetului: cuvânt, dublu-cuvânt şi cvadruplu-cuvânt

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei
- ***Cum putem identifica octeţii din cadrul unui cuvânt, dublucuvânt, etc ?***
- Uneori, se mai practică și numerotarea ***octeţilor*** sau ***tetradelor*** din cadrul structurii,
- Ex: pentru un dublucuvânt, despre **octetul având biţii 31...24** se spune că este de **rang 3**, numerotarea începând de la rangul 0 prin octetul cu biţii 7...0.



- Operanzii folosiţi în PC pot avea şi dimensiuni mai mari, dar ***numai multipli de dimensiunea octetului*** şi în plus, aceştia sunt în general doar puteri ale lui 2:
- 2^0 octeţi = 1 octet, 2^1 octeţi = 1 cuvânt, 2^2 octeţi = 1 dublucuvânt, etc

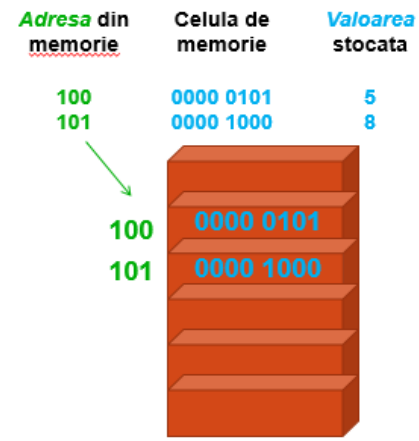
2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei (cont.)
- ***Cum putem accesa informaţia din memorie?***

Memoria internă a unui SC este văzută ca ***o succesiune de locaţii***,

fiecare **locaţie** având un număr fix de biţi (**analogie cu un dulap cu sertare**,

sertarele fiind toate de aceeaşi dimensiune).



Locaţia = unitatea elementară de adresare a unei memorii,
accesarea conţinutului realizându-se prin ***adresarea locaţiilor de memorie***;
⇒***folosind adresa, vom avea acces la conţinut***

⇒(**analogie** cu o ***etichetă*** care precizează *un număr de ordine pentru sertar*:
consultând eticheta accesăm conţinutul sertarului).

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei (cont.)
- ***Câţi octeţi putem accesa la un moment dat ?***

Figura 2.5: modul cum se depune în memorie octetul ***21h***, cuvântul ***43 21h*** sau dublucuvântul ***87 65 43 21h*** începând de la adresa 126 **-> urmatoarele adrese ocupate vor fi : (sens crescator) 127, 128, 129, … (la noi)**

ocupând 1 locaţie, 2 locaţii sau 4 locaţii succesive.

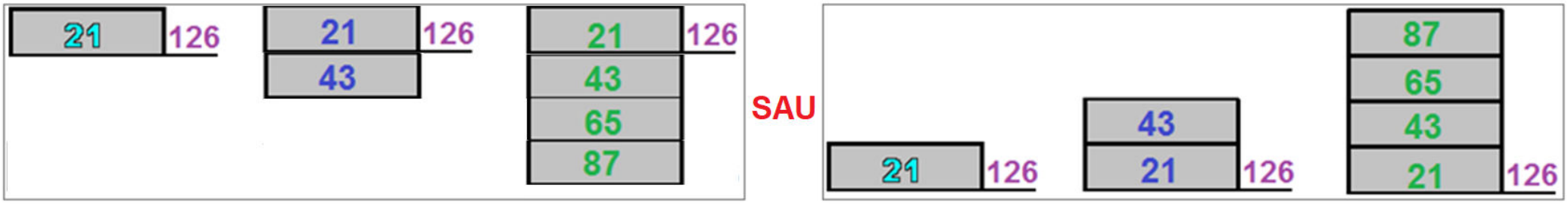


Figura 2.5 Octetul 21h, cuvântul 4321h și dublucuvântul 87654321h stocate în memorie începând de la adresa 126

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei (cont.)
- ***Cum putem accesa octeţii din memorie folosind adrese ?***
- Dacă dorim să accesăm ***doar octetul 21h*** (adică "să deschidem *un singur sertar* din dulap"),
  va trebui specificată cumva ***adresarea la nivel de octet***,

Pe un CPU de 16 biti, se mai poate folosi:

adresarea la nivel de cuvânt ("deschidem *2 sertare*") sau

Pe un CPU de 32 biti, se mai poate folosi:

adresarea la nivel de dublucuvânt ("deschidem *4 sertare* consecutive")

plecând de la o anumită adresă.

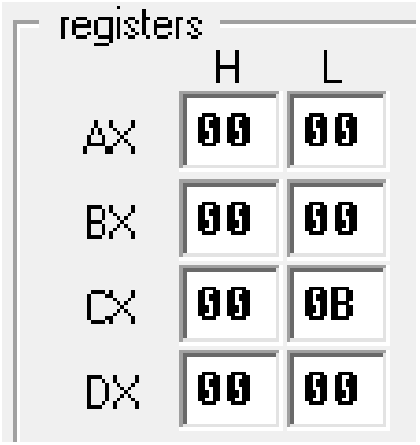
De exemplu, o instrucţiune de forma:

- **mov AL, [126]** ; *va accesa octetul din memorie* de la *adresa 126*

și îl ***va copia*** (sau muta după cum sugerează instrucţiunea ***mov***)
în *registrul AL* = registru pe 8 biţi al 8086.

- **mov AX, [126]** ; va accesa *cuvântul* din memorie de la *adresa 126*

și îl ***va copia*** în *registrul AX* = registru pe 16 biţi al procesorului 8086.



2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei (cont.)
- ***Câţi biţi putem accesa minim din memorie ?***
- La procesoarele Intel, ***unitatea minimă de adresare*** (locaţia de memorie) ***are 8 biţi***,

nu se poate accesa mai puţin, așa cum e cazul microcontrolerelor de exemplu (acces la nivel de bit).

- Dar memoria *se poate adresa și la nivel de* cuvânt (16 biţi) sau dublu-cuvânt (32 biţi),

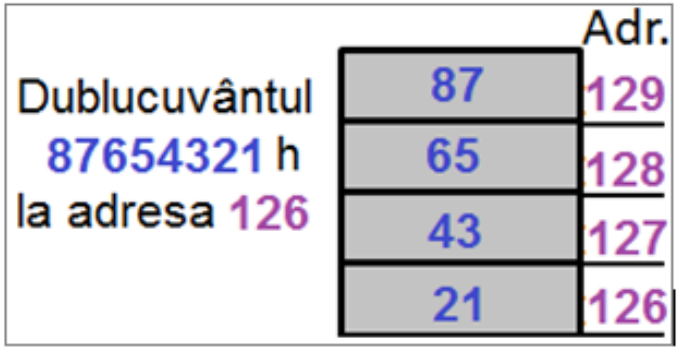
Figura alaturata:

Se poate adresa:

un octet de la oricare din adresele 126, 127, 128 sau 129,

un cuvânt - cel format de adresele 126 şi 127 sau 127 și 128 (obligatoriu locaţiile se vor considera succesive),

un dublucuvânt, aşa cum apare la locaţiile 126 ...129.



Convenţional, ***bitul c.m.p.s.*** (cel mai puţin semnificativ, în engleză LS - *least significant* bit) al unei valori, numerotat ca bitul b_0, se află în **poziţia cea mai din dreapta a valorii**,

bitul c.m.s. (cel mai semnificativ, în engleză MS - *most significant* bit) în **poziţia cea mai din stânga**, Figura 2.7.

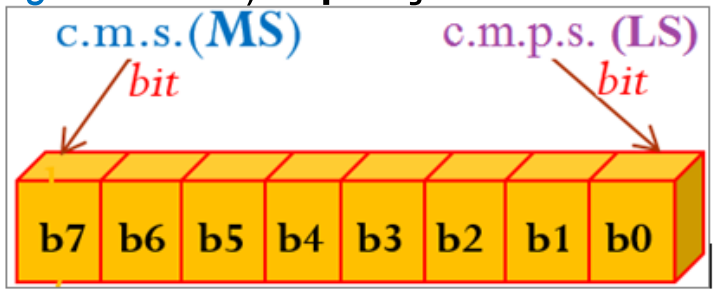


Figura 2.7 Numerotarea biţilor într-un octet: cel mai semnificativ (c.m.s.) şi cel mai puţin semnificativ (c.m.p.s.) *bit dintr-un octet*

- **Această organizare** a biţilor (într-un octet) poate fi **generalizată şi asupra octeţilor** (într-un cuvant) (Figura 2.8),

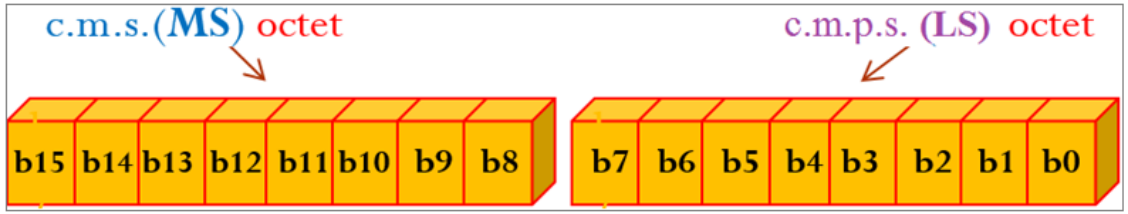


Figura 2.8 Cel mai semnificativ (c.m.s.) şi cel mai puţin semnificativ (c.m.p.s.) *octet dintr-un cuvânt*

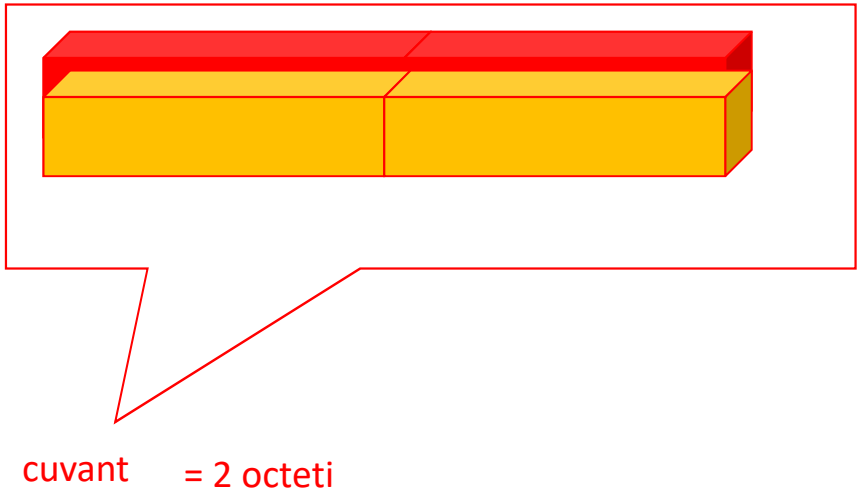
Resp. poate fi **generalizată şi asupra cuvintelor** (într-un dublucuvant) , etc.

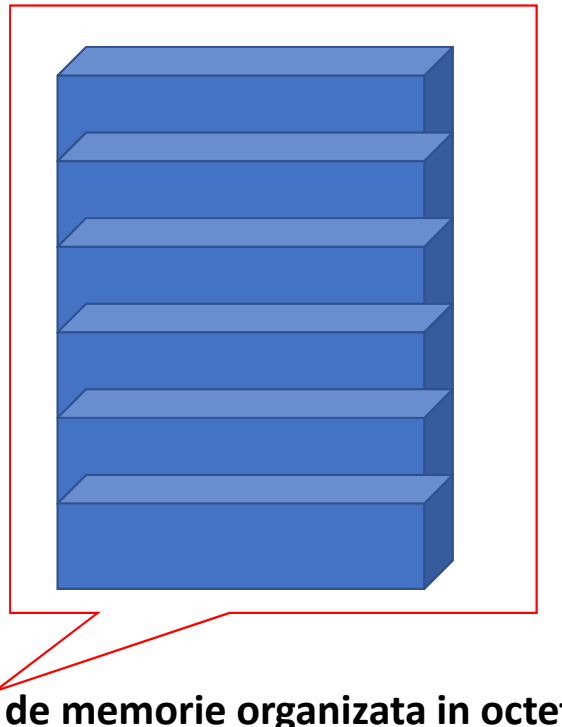
Stocarea datelor ce au o dimensiune mai mare decat busul de date al unui procesor se realizeaza in general la adrese consecutive de memorie.

De exemplu:

un CPU avand **memoria organizata in octeti**,

va putea adresa un cuvant prin preluarea /depunerea a **2 octeti** incepand de la adresa respectiva.





2. Reprezentarea informaţiei în PC

- **2.1. Aspecte generale de reprezentare a informaţiei (cont.)**
- ***“În care sertar va ajunge sacoul ?”***

Analogia cu dulapul cu sertare: să presupunem că trebuie sa depozităm în dulap un costum din 2 piese: sacou şi pantaloni.

- **În care sertar va ajunge sacoul: în cel de deasupra sertarului cu pantaloni, sau în cel de sub acesta?**

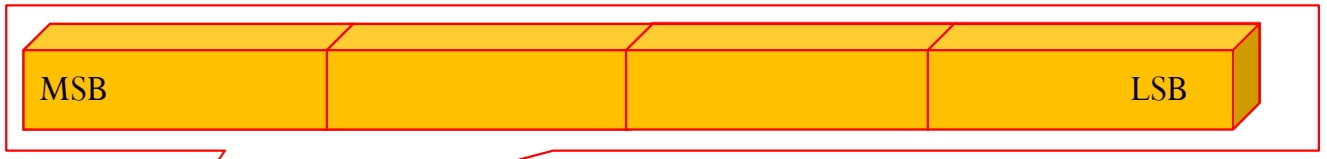
Este importantă această organizare deoarece la un moment dat am putea ruga pe cineva să ne livreze conţinutul sertarului 5 de exemplu (fiind siguri că acolo e sacoul).

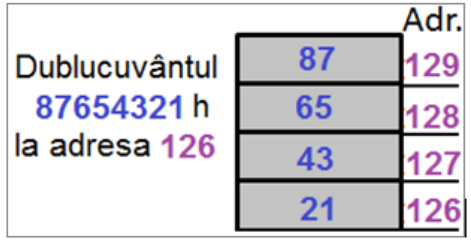
Octetul c.m.s. deasupra octetului c.m.p.s. sau invers?

Little sau Big END - ian ?

Reprezentarea numerelor pe octet (5)

La noi (x86): conventia **Little Endian** (procesoare Intel, AMD)

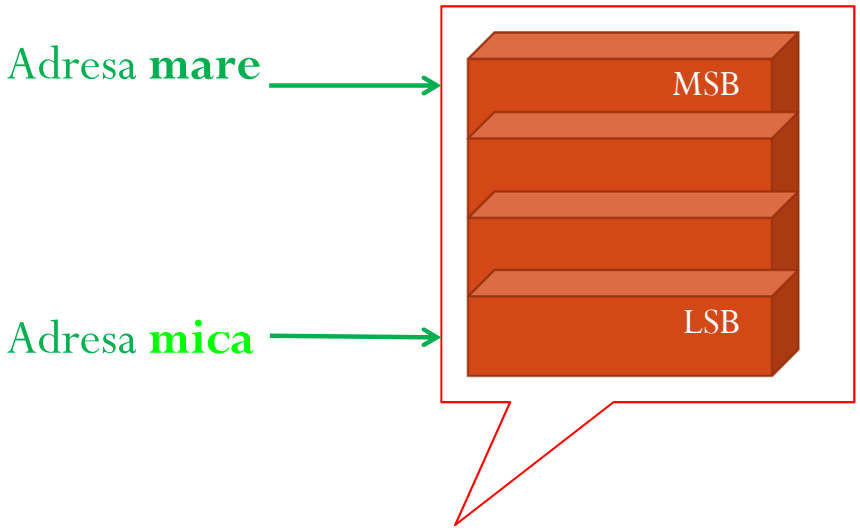




dublucuvant = 4 octeti

mov [126],87654321h

=>
[129] = 87h
[128] = 65h
[127] = 43h
[126] = 21h



Zona de memorie organizata in octeti

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei (cont.)
- ***Cum se depun octeţii unui cuvânt în memorie? (în sus sau în jos?)***
- **La adresa curentă e octetul c.m.s. sau c.m.p.s. (dintr-un cuvânt, de exemplu)?**
- Trebuie specificată familia din care face parte procesorul sau ***convenţia utilizată la reprezentare;*** altfel, poate fi greşit/ ambiguu.

Adresa	Conţinutul		Adresa	Conţinutul
0003	87	Dublucuvântul	**0003**	**21**
0002	65	**87.65.43.21h** în	0002	43
0001	43	memorie de tip	0001	65
0000	**21**	Little sau Big	0000	87
Little END-ian		END-ian	**Big END-ian**	

Dublu-cuvântul **87654321h** se depune în memorie folosind una din convenţiile:

- **Little Endian:** octetul **LSB**, adică cel de la sfârşitul structurii (“**END**”-ian) se depune în memorie la locaţia cu **adresa cea mai *mică* (“Little”)**

  convenţia este specifică procesoarelor din familia *Intel*

- **Big Endian:** octetul **LSB** se depune în memorie la locaţia cu **adresa cea mai *mare* (“Big”),** convenţia fiind specifică procesoarelor din familia *Motorola*

2. Reprezentarea informaţiei în PC

- 2.1. Aspecte generale de reprezentare a informaţiei
- ***Cum putem ilustra conţinutul zonei de memorie?***

(înspre adrese crescătoare sau înspre adrese descrescătoare?)

Desenul (fig) - valabil pt un CPU Intel, de ex., care respectă formatul ***Little Endian***.

! Se recomandă: utilizarea desenului cu ***adrese crescătoare***

(și în cadrul simulatorului **EMU8086** acestea sunt prezentate sub această formă)

Recomandarea ţine mai mult de practică, pentru a nu greși în interpretarea datelor din memorie.

Figura 2.10:
un *octet* *la adresa 118*,
un *cuvânt* *la adresele 122-123* şi
un *dublucuvânt* *la adresele 126-129*

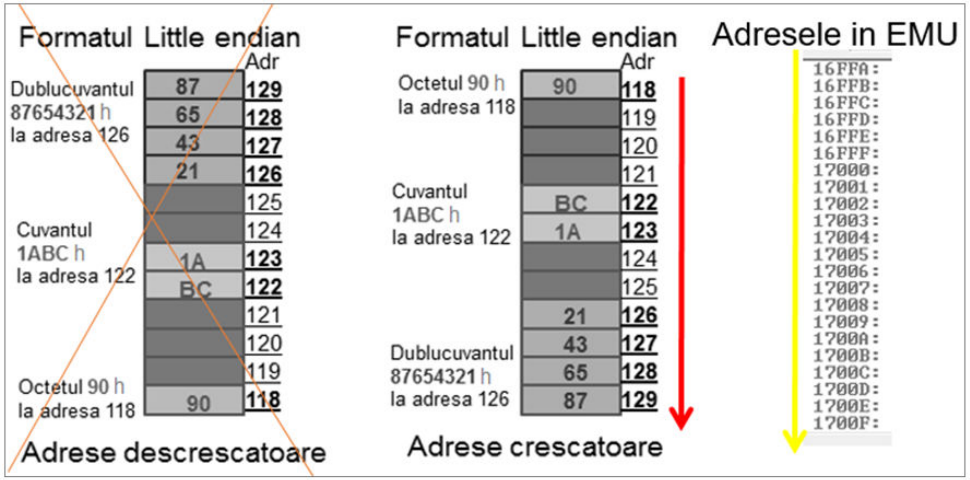


Figura 2.10 Formatul Little Endian ilustrat la adrese
a) descrescătoare sau b) crescătoare

2. Reprezentarea informaţiei în PC

Figura 2.10:
un *octet 90h* la *adresa 118*,
un *cuvânt 1ABCh* la *adresele 122-123* şi
un *dublucuvânt 87654321h* la *adresele 126-129*

Si adresa se exprima in gen. tot in hexa

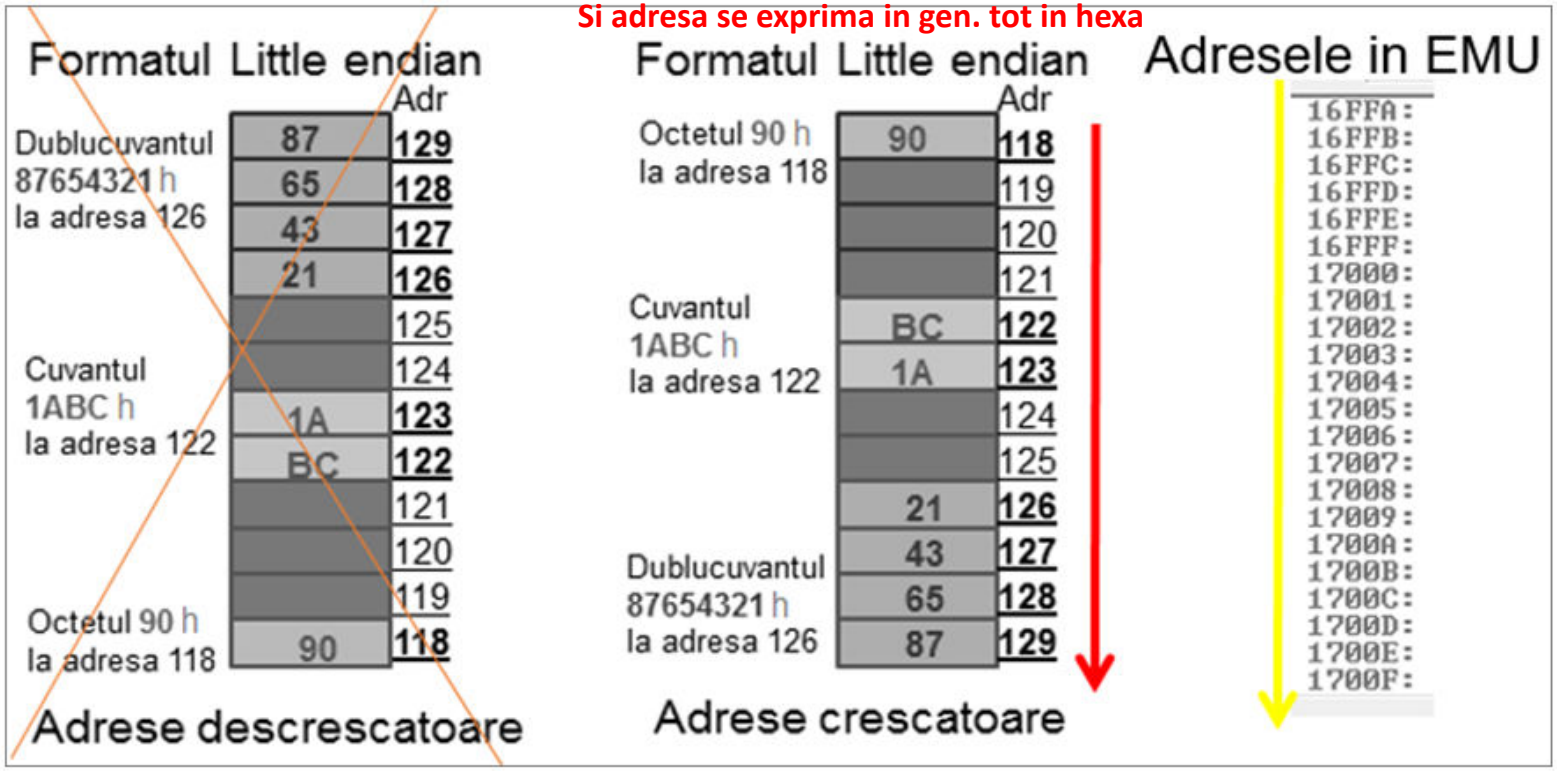


Figura 2.10 Formatul Little Endian ilustrat la adrese
a) descrescătoare sau b) crescătoare

Prefixe in PC

Prefixe pt unitati de masura pentru viteza si capacitate:

Kilo(K) = $10^3 \approx 2^{10}$

Mega(M) = $10^6 \approx 2^{20}$

Giga(G) = $10^9 \approx 2^{30}$

Tera(T) = $10^{12} \approx 2^{40}$

Peta (P) = $10^{15} \approx 2^{50}$

- Hertz (Hz)=cicluri de ceas (clock cycles) pe secunda=frecventa

  1MHz = 10^6 = 1,000,000Hz

  - Ex: viteza procesorului (in MHz, GHz).

- Byte (octet) = unitate de stocare

  1KB = 2^{10} = 1024 octeti (Bytes)

  1MB = 2^{20} = 1,048,576 octeti (Bytes)

  - Ex: memoria principala (RAM) - in MB, GB
  - capacitatea de stocare a HDD - in GB, TB.

Prefix	Simbol	Puterea lui 10	Echivalent Puterea lui 2
Yotta	Y	10^{24}	2^{80}
Zetta	Z	10^{21}	2^{70}
Exa	E	10^{18}	2^{60}
Peta	P	10^{15}	2^{50}
Tera	T	10^{12}	2^{40}
Giga	G	10^{9}	2^{30}
Mega	M	10^{6}	2^{20}
Kilo	K	10^{3}	2^{10}
mili	m	10^{-3}	
micro	μ sau u	10^{-6}	
nano	n	10^{-9}	
pico	p	10^{-12}	
femto	f	10^{-15}	

Prefixe in PC (2)

- Prefixe pt unitati de masura pentru timp si spatiu:

Milli (m) = 10^{-3}

Micro (μ) = 10^{-6}

Nano (n) = 10^{-9}

Pico (p) = 10^{-12}

Femto (f) = 10^{-15}

- Millisecunda = a 1000-a parte dintr-o secunda
  - Timpul de acces al HDD (Hard disk drive) in gen: 10 - 20 ms
- Nanosecunde = 1 parte dintr-o (secunda/1milion)
  - Accesul la memoria principala – in gen: 50 – 70ns
  - Micron (micrometru) = o parte a unui metru /1 million
  - Circuitele din cipurile computerelor – masurate in microni

-> 2.2. Reprezentarea informaţiei numerice

2.2.1. Sisteme de numeraţie

- ***Ce sistem de numeraţie se folosește la scrierea valorilor din PC ?***

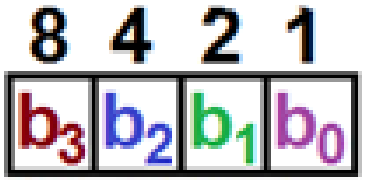
Informaţia numerică sau numerele din PC se reprezintă prin intermediul unui ***sistem de numeraţie;*** la scrierea numerelor (în contextul reprezentării informaţiei într-un SC) se folosesc **sisteme de numeraţie *poziţionale*; cele mai uzuale :**

- sistemul **zecimal** – folosit de oameni în exprimarea numerelor,
- sistemul **binar** – folosit pentru exprimarea valorilor în calculator,
- sistemul **hexazecimal** – folosit în scrierea mai rapidă a valorilor din calculator,
- sistemul **octal**.

Tabelul 2.1. Baza şi cifrele sistemelor de numeraţie binar, zecimal, hexazecimal şi octal

Sistem de numeraţie	Baza (q)	Cifrele sistemului de numeraţie
Binar	2	{0,1}
Zecimal	10	{0,1,2,3,4,5,6,7,8,9}
Hexazecimal	16	{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}
Octal	8	{0,1,2,3,4,5,6,7}

2.2.1. Sisteme de numeraţie



- ***Cum putem trece ușor dintr-un sistem de numeraţie binar in hexa** sau invers?*

1) prin folosirea șablonului:

unde **b3**,**b2**,**b1**,**b0** sunt cifre binare (0 sau 1)

care se ponderează cu puterile corespunzătoare ale lui 2, adică **8**, **4**, **2**, **1**.

- funcţionează **doar pentru o cifră hexazecimală**, deci numere care se scriu cu ***4 biţi***.

Ex1) 1010b => un opt + un doi = zece = Ah

Ex2) numărul D (din hexa) în binar

Dh= 13 -> **8+4+1** = un opt + un patru + un unu = **1101**b

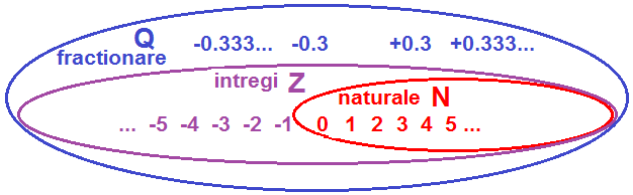
2) prin folosirea tabelului

Tabelul 2.2. Numere scrise în sistemul zecimal, binar, hexazecimal şi octal

Sistemul zecimal q=10	Sistemul binar q=2	Sistemul hexazecimal q=16	Sistemul octal q=8
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

- ***Ce fel de numere vom întâlni în EMU8086 ?***

(nu se fol. nr. fractionare in EMU)



2.2.2. Clasificarea numerelor fără semn vs. cu semn

- numărul scris sub forma **4** se va considera ***fără semn***,
- numărul scris sub forma **+4** va fi considerat pozitiv
- numărul scris sub forma **-4** va fi considerat negativ

} ***cu semn***.

- La scrierea numărului *cu semn,* **în binar** se atribuie:

cifra 0 pentru semnul plus şi ***cifra 1 pentru semnul minus***.

- !!! **în nici un alt sistem de numeraţie (binar, hexazecimal, octal)** înafară de cel **zecimal** nu există simbolul „**-**” sau „**+**” care să desemneze ***semnul unui număr***.

2.2.2. Clasificarea numerelor fără semn vs. cu semn

La ce se referă "gama numerelor" ?

- ***Gama numerelor*** -> domeniul valorilor ce se pot reprezenta pe un număr fix de biţi;

folosind n biţi,

- numerele ***fără semn*** -> în **gama $[0;2^n-1]$**, iar
- numerele ***cu semn*** -> în **gama $[-2^{n-1};+2^{n-1}-1]$**.

n=2 -> [0;3] [-2;+1]
n=3 -> [0;7] [-4;+3]
n=4 -> [0;15] [-8;+7]
n=5 -> [0;31] [-16;+15]
...

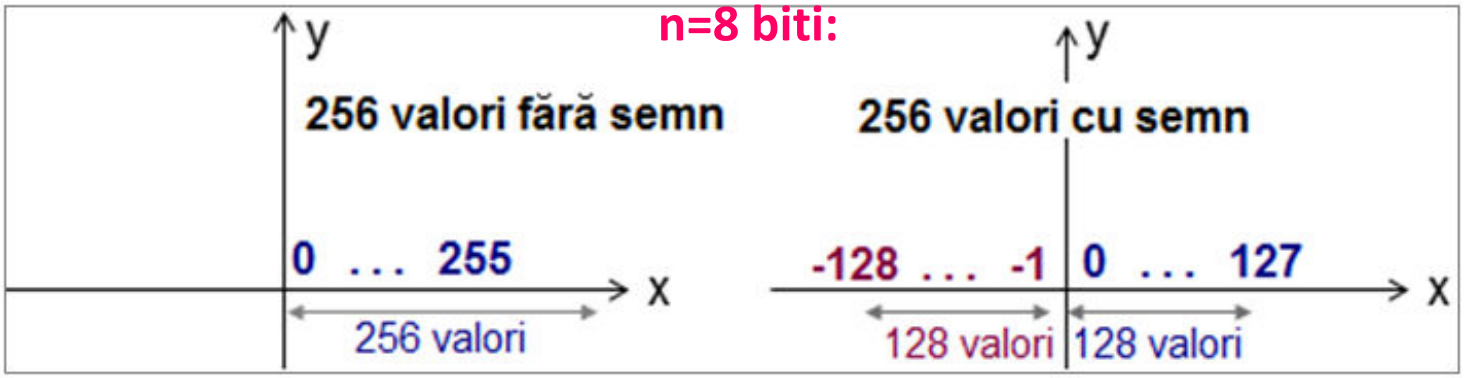


Figura 2.14. Reprezentarea a 256 valori fără semn vs. cu semn

2.2.2. Clasificarea numerelor fără semn vs. cu semn

Gama numerelor folosind n biţi,

- numerele ***fără semn*** -> în **gama [0;2^n-1]**, iar
- numerele ***cu semn*** -> în **gama [-2^{n-1};+2^{n-1}-1]**.

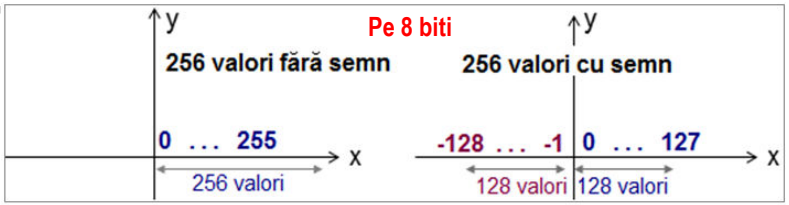


Figura 2.14. Reprezentarea a 256 valori fără semn vs. cu semn

Exemplul1: Aflaţi numărul minim de biţi necesar reprezentării corecte a numărului 3.

Răspuns: Fiind un număr **fără semn** -> se încadrează numărul 3 în gama [0;3];

din relaţia 2^n-1=3 => numărul minim de biţi necesar scrierii corecte a lui 3 este **n=2**.

Într-adevăr, 3 se va scrie corect în binar ca 11b.

Exemplul2: Aflaţi numărul minim de biţi necesar reprezentării corecte a numărului +3.

nr +3 (**cu semn**) -> încadrat în gama [-4,+3] => relaţia $+2^{n-1}-1=+3$ sau $-2^{n-1}=-4$ => **n=3**.

Astfel, avem nevoie de un bit suplimentar pentru a reprezenta corect numărul cu semn +3, iar acest bit suplimentar va fi **0** deoarece numărul **este pozitiv**.

2.2.2. Clasificarea numerelor fără semn vs. cu semn

- ***Cum ne dăm seama ce semn are un număr (scris in hexa sau binar) ?***

- dacă un număr scris în hexazecimal are **cifra c.m.s. între 0 și 7**, atunci el este **pozitiv**,
  iar dacă **cifra c.m.s.** este **între 8 și Fh**, atunci el este **negativ**.

Exemple de numere cu semn pe 8 biţi, ***pozitive*** : 05h, 0Ah, 10h, 2Eh, 7Fh
negative: 80h, 9Dh, A7h, FFh, etc.

- regula similară pentru un număr scris în binar:

dacă un număr scris în binar are **MSb în 0**, atunci numărul este **pozitiv**,
dacă are **MSb în 1**, atunci numărul este **negativ**.

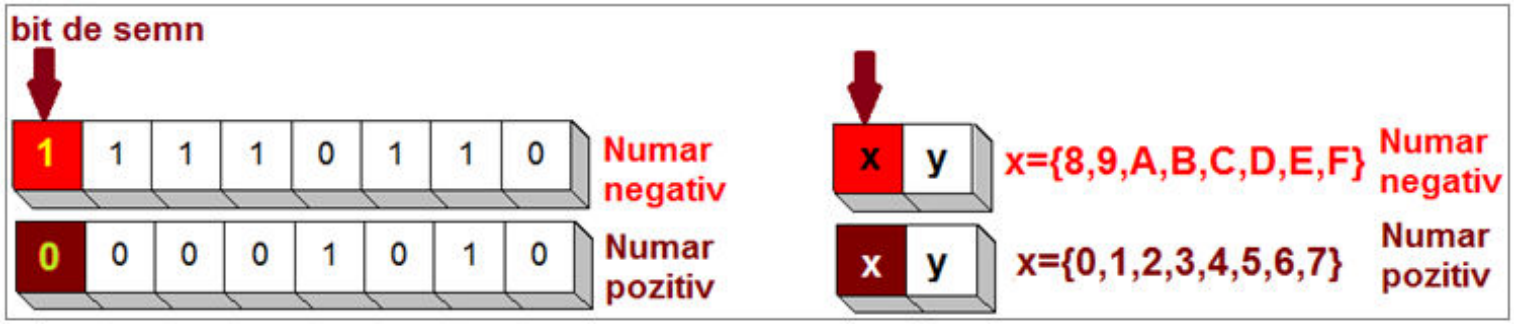


Figura 2.15. Reprezentare sugestivă a numerelor pozitive și negative
a) în binar și b) în hexazecimal (pe 8 biţi)

2.2.2. Clasificarea numerelor fără semn vs. cu semn

- ***Câte numere pozitive/ negative putem scrie folosind un anumit număr de biţi ?***

- în *reprezentarea numerelor* ***fără semn*** se pot scrie **de două ori mai multe numere pozitive** decât în cea ***cu semn***.

  Problemele apar atunci când valorile depăşesc domeniul de reprezentare,

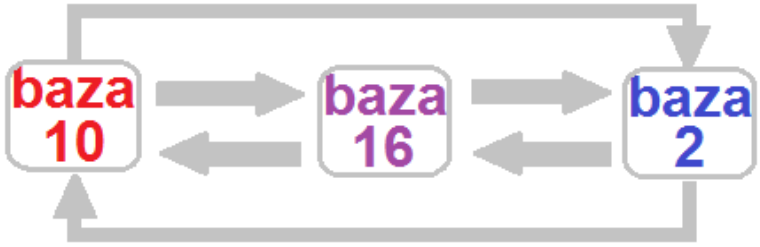
  de exemplu dacă adunăm 1111+1=0000b în loc de 10000b

- **2.2.3. Conversii simple de numere dintr-o bază în alta**
- ***Cum distingem baza în care s-a reprezentat un număr ?***

La sfârşitul numărului **se adaugă o literă ce simbolizează baza**, astfel:

B sau **b** pentru numerele scrise în ***binar*** (baza 2),
Q, **O**, **q** sau **o** pentru numerele scrise în ***octal*** (baza 8),
D sau **d** pentru numerele scrise în ***zecimal*** (baza 10) sau nimic,
H sau **h** pentru numerele scrise în ***hexazecimal*** (baza 16).

- ***Care sunt regulile de conversie dintr-o bază în alta ?***



2.2.3. Conversii de numere <u>fără semn</u>

- ***Care sunt regulile de conversie dintr-o bază în alta ?***

(h->b) Conversii din hexazecimal în binar

Din Tabel: orice cifră hexazecimală va fi înlocuită cu un şir de 4 cifre binare corespunzătoare.

9Ah = **9 A** h = **1001 1010**b = 10011010b

(b->h) Conversii din binar în hexazecimal

se formează ***grupuri de câte 4 cifre binare***
pornind dinspre c.m.p.s. bit (deci din dreapta spre stânga),
înlocuind fiecare grup cu cifra hexazecimală corespunzătoare.

Ex1) 10011010b = **1001 1010**b = **1001 1010**b= 9Ah

Ex2) 11010b =**000**11010b = **0001 1010** b = **1 A** h,

(s-au adăugat 3 biţi de 0 pentru a forma un grup complet de 4 biţi)

Tabelul 2.2. Numere scrise în sistemul zecimal, binar, hexazecimal şi octal

Sistemul zecimal q=10	Sistemul binar q=2	Sistemul hexazecimal q=16	Sistemul octal q=8
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

2.2.3. Conversii de numere fără semn

- *Care sunt regulile de conversie dintr-o bază în alta ?*

(d->b) Conversii din zecimal în binar

Metoda 1: Se împarte în mod repetat numărul din zecimal la baza 2 până la obţinerea câtului 0 şi se culeg resturile obţinute în ordine inversă.

Sunt posibile 2 metode de scriere:

Forma de scriere I :

37 : 2 cât 18 rest 1
18 : 2 cât 9 rest 0
9 : 2 cât 4 rest 1
4 : 2 cât 2 rest 0
2 : 2 cât 1 rest 0
1 : 2 cât 0 rest 1

Forma de scriere II :

37| 2
36| 18| 2
= 1| 18| 9| 2
= 0| 8| 4| 2
=1| 4| 2| 2
=0| 2| 1| 2
=0| 0| 0
=1|

Metoda 2: Se va căuta puterea cea mai mare a bazei care se poate scădea din numărul de reprezentat şi se efectuează scăderea, notând puterea respectivă.

Există posibilitatea de a fi necesare mai multe scăderi ale aceleiaşi puteri, iar astfel se va nota multiplul puterii.

Operaţia se repetă până la obţinerea diferenţei 0, noua valoare a descăzutului fiind numărul rămas după efectuarea scăderii.

Avem puterile lui 2 <=37: $\{2^5=32 , 2^4=16, 2^3=8, 2^2=4, 2^1=2, 2^0=1\}$

37 - 32 = 5 (se notează 2^5 o dată)

5 - 4 = 1 (se notează 2^2 o dată)

1 - 1 = 0 (se notează 2^0 o dată) =>

2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	0	1

pe poziţiile 5, 2, 0 se va pune (în general multiplul puterii) cifra 1, iar în rest cifra 0.

=> $37 = 100101_b$

2.2.3. Conversii de numere fără semn

- *Care sunt regulile de conversie dintr-o bază în alta ?*

(d->b) Conversii din zecimal în binar

- Numărul 37, scris ca *număr fără semn* :

=> $37 = 1\,0\,0\,1\,0\,1_b$

- Dacă se consideră că este vorba despre reprezentarea ca *număr cu semn :*
  - numărul **+ 37** în binar, corect s-ar fi scris:

$\underline{0}100101_b = 0*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 0 + 32 + 4 + 1 = +37$

$+37 = \underline{0}\,1\,0\,0\,1\,0\,1_b$

  - numărul **- 37** în binar s-ar fi scris:

$1011011_b = -1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = -64 + 16 + 8 + 2 + 1 = -37$

$-37 = 1\,0\,1\,1\,0\,1\,1_b$???

(a se vedea **conversia numerelor *cu semn***)

2.2.3. Conversii de numere fără semn

- *Care sunt regulile de conversie dintr-o bază în alta ?*

- **(h->d) Conversii din hexazecimal în zecimal**

La conversia numerelor fără semn, se procedează asemănător ***conversiei din binar în zecimal***, doar că puterile considerate vor fi ale bazei 16:

$25h = 2*16^1 + 5*16^0 = 32 + 5 = 37$

- **(d->h) Conversii din zecimal în hexazecimal**

Se pot adapta cele 2 metode de conversie aplicate la ***conversia din zecimal în binar***,

Metoda 1. (cu împărţire la baza 16 şi se culeg resturile):

37: 16 -> cât 2 rest 5

2: 16 -> cât 0 rest 2 -> 37 = 32 + 5 = $2*16^1 + 5*16^0 = 2\ 5_h$

Metoda 2. Folosind metoda cu scădere a bazei la diferite puteri

37 -16 = 21 (16^1)
21 -16 = 5 (16^1)
5 -1 = 4 (16^0)
4 -1 = 3 (16^0)
3 -1 = 2 (16^0)
2 -1 = 1 (16^0)
1 -1 = 0 (16^0) => 2 5 h

Metoda 3. : (cea mai simplă) - cu trecere prin binar:

se transformă numărul din zecimal în binar şi apoi formând grupuri de câte 4 biţi (tetrade) se scriu cifrele hexa corespunzătoare: 37=100101b = 0010 0101b = 25h

2.2.3. Conversii de numere fără semn

- *Care sunt regulile de conversie dintr-o bază în alta ?*

- **(h->d) Conversii din hexazecimal în zecimal**
- La conversia numerelor fără semn, se procedează asemănător ***conversiei din binar în zecimal***, doar că puterile considerate vor fi ale bazei 16:

10E1h=**1***16^{3}+**0***16^{2}+**14***16^{1}+**1***16^{0}=**4096**+**0**+**224**+**1**=4321

- **(d->h) Conversii din zecimal în hexazecimal**
- Se pot adapta cele 2 metode de conversie aplicate la ***conversia din zecimal în binar***,

Metoda 1. (cu împărţire la baza 16 şi se culeg resturile):

4321:16= 270 (reţinem restul **1** - va fi cifra hexa de ordin 0),
apoi 270:16=16 (reţinem restul **14=E** - va fi cifra hexa de ordin 1),
apoi 16:16=1 (reţinem restul **0** - va fi cifra hexa de ordin 2),
şi în final 1:16=0 (reţinem restul **1** – aceasta va fi cifra hexa de ordin 3);
s-a obţinut câtul 0, deci se scriu aceste resturi în ordine inversă: 4321=**10E1**h

Metoda 2. Folosind metoda cu scădere a bazei la diferite puteri

4321 - 4096 = 225 (16^{3})
225 - 16 = 209 (16^{1})
209 - 16 = 193 (16^{1})
193 - 16 = 177 (16^{1})
. . . . . .
17 - 16 = 1 (16^{1})
1 - 1 = 0 (16^{0}) => **10E1** h

3 2 1 0
4096,256,16,1
de 14 ori

Metoda 3. : (cea mai simplă) - cu trecere prin binar: se transformă numărul din zecimal în binar şi apoi formând grupuri de câte 4 biţi se scriu cifrele hexa corespunzătoare.
4321=0001000011100001b = **0001 0000 1110 0001**b = **10E1**h

(nr <-> q) Conversia numerelor din/ în octal

Se procedează în mod similar conversiei din/ în hexazecimal, singura diferenţă fiind la trecerea din binar în octal (sau invers), unde în locul formării de grupuri de 4 biţi, se vor forma **grupuri de câte 3 biţi** şi *se va folosi cifra în octal corespunzătoare*; se folosesc **triade** (în loc de tetrade).

Se va ţine cont de baza 8 a sistemului şi de cifrele octale asociate ({0,1,2,…,7}).

Exemplu:

Metoda 1 :

4321:8= 540 (reţinem restul **1** - va fi cifra octală de ordin 0),
apoi 540:8=67 (reţinem restul **4** - va fi cifra octală de ordin 1),
apoi 67:8=8 (reţinem restul **3** - va fi cifra octală de ordin 2), ș.a.m.d.
…
iar când se obţine câtul 0 se scriu aceste resturi în ordine inversă: 4321=**0**10341q

Metoda 3 *cu trecere prin binar*: 4321=0001000011100001b =**0 001 000 011 100 001**b = **0**10341q

! Verificare: Cum verificam un numar din binar in zecimal?

(problematice sunt numerele care incep cu bit de 1)

Numerele care incep cu bit de 0, sunt *simplu de convertit in zecimal* :

Exemplul 1) Numarul $0\,1\,1\,0\,1\,1_b$ = ? (pe 6 biti)

- **Fara semn:** $011011_b = 0*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 0 + 16 + 8 + 2 + 1 = 27$

Cum se extinde la 8 biti ? Dar la 16 biti? R: $0001\ 1011_b$ =1Bh=001Bh

- **Cu semn:** $011011_b = -0*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = -0 + 16 + 8 + 2 + 1 = +27$
- Cum se extinde la 8 biti ? Dar la 16 biti? R: $0001\ 1011_b$ =1Bh=001Bh

Exemplul 2) Numarul $0\,1\,1\,1\,1\,1\,1\,1_b$ = ? (pe 8 biti)

- **Fara semn:**

$01111111_b = 0*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 0+64 +32+ 16 + 8 + 4 + 2 + 1 = 127$

Cum se extinde la 16 biti? R: 0000000001111111_b =007Fh

- **Cu semn:**

$01111111_b = -0*2^6 + 1*2^7 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = -0+64 +32+ 16 + 8 +4+ 2 + 1 = +127$

- Cum se extinde la 16 biti? R: 0000000001111111_b =007Fh

! Verificare: Cum verificam un numar din binar in zecimal?

(problematice sunt numerele care incep cu bit de 1)

Numerele care incep cu bit de 1, trebuie convertite cu atentie din binar in zecimal !!!

Exemplul 1) Numarul $1\,0\,1\,1\,0\,1\,1_b$ = ? (pe 7 biti)

- **Fara semn:** $1011011_b = 1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 64 + 16 + 8 + 2 + 1 = 91$

Cum se extinde la 8 biti ? Dar la 16 biti? R: 01011011_b =5Bh=005Bh

- **Cu semn:** $1011011_b = -1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = -64 + 16 + 8 + 2 + 1 = -37$
- Cum se extinde la 8 biti ? Dar la 16 biti? R: 11011011_b =DBh=FFDBh

Exemplul 2) Numarul $1\,1\,1\,1\,1\,1\,1\,1_b$ = ? (pe 8 biti)

- **Fara semn:**

$11111111_b = 1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 128+64 +32+ 16 + 8 + 4 + 2 + 1 = 255$

Cum se extinde la 16 biti? R: 0000000011111111_b =00FFh

- **Cu semn:**

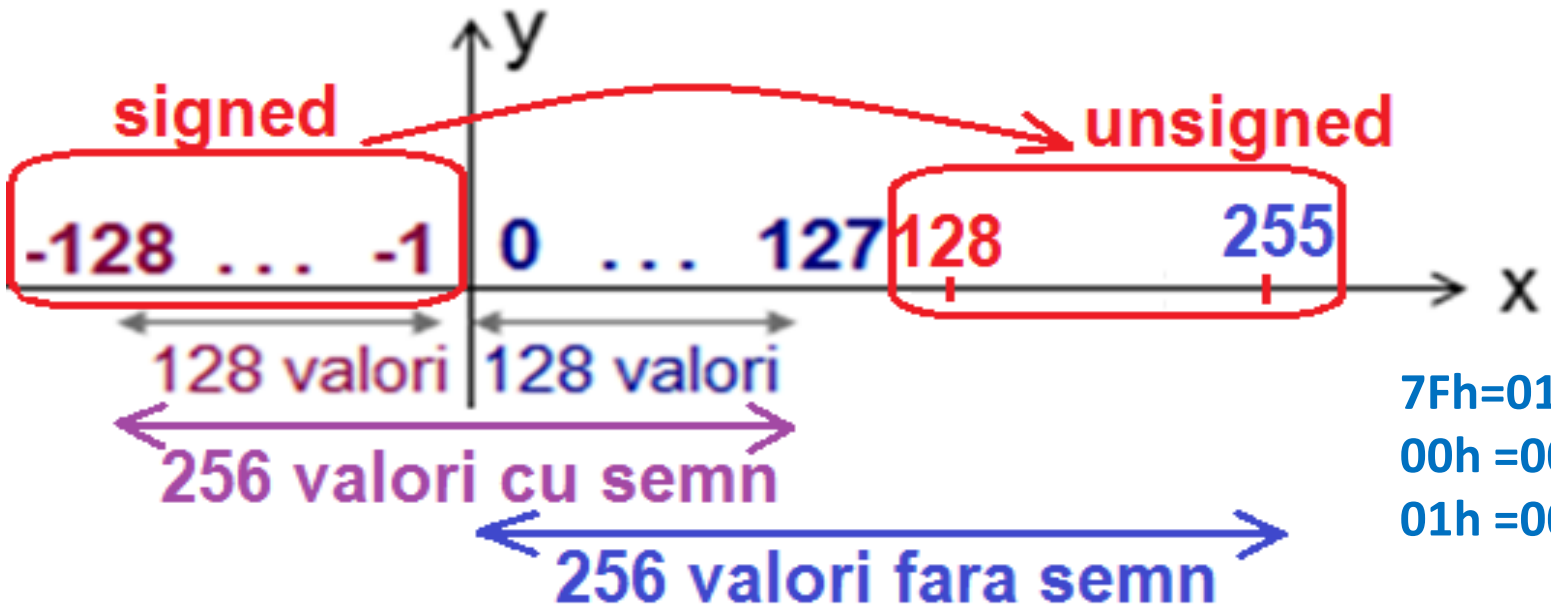
$11111111_b = -1*2^6 + 1*2^7 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = -128+64 +32+ 16 + 8 +4+ 2 + 1 = -1$

- Cum se extinde la 16 biti? R: 1111111111111111_b =FFFFh

Concluzie: 91(uns) = -37(s), 255(uns)=-1(s)

Prescurtari:
uns = **uns**igned
s = **s**igned

Valori cu semn si fara semn pe n=8 biti :



7Fh=0111 1111b = +127=127
00h =0000 0000b = 0 = 0
01h =0000 0001b = +1 = 1

- valorile foarte mari (unsigned) au echivalent valori negative (signed) !!!

Exemple:

FFh = **1**111 1111b = -1 = 255 (cel mai mare nr negativ)

80h = **1**000 0000b = -128 = 128 (cel mai mic nr negativ)

Cum sunt considerate numerele in CPU? Fara semn sau cu semn?

- Adunarea a 2 valori pe 16 biti: FF12h + 1234h = ?

1 Daca le consideram in ***conventia fara semn*** (pe 16 biti):

FF12h = **65298** +
1234h = 4660
69958 ← atat ne da NOUA

2 Daca le consideram in ***conventia cu semn*** (pe 16 biti):

FF12h = **- 238** +
1234h = 4660
4422 ← atat ne da NOUA

DAR pe CPU ??? : FF12h + 1234h = ????? **1** sau **2** ?

```
mov ax,0ff12h
mov bx,1234h
add ax,bx
```

→ AX: H = 11, L = 46 in hexa => 1146h= **4422** **2**

!!! Dar la consultarea flagurilor CPU, se vede **CF=1** --> Ce semnificatie are **CF=1**?

CF=1 ≡ "*a existat un transport inafara domeniului de reprezentare al nr*"

Acel transport = 2^{16} (operatia s-a realizat la nivel de 16 biti) => 2^{16} =65536"

… si atunci **4422+65536= *69958*** **1**

!!! FF12h + 1234h = 11146h !!!

2.2.4. Conversii de numere cu semn

- ***Care este MSb la numerele reprezentate în convenţia cu semn ?***

Gama numerelor folosind n biţi,

numerele ***fără semn*** -> în **gama [0;2^n-1]**, iar

numerele ***cu semn*** -> în **gama [-2^{n-1};+2^{n-1}-1]**.

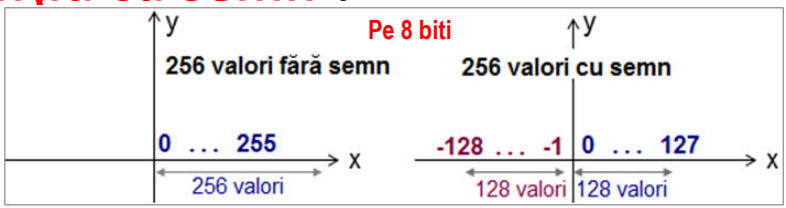


Figura 2.14. Reprezentarea a 256 valori fără semn vs. cu semn

MSb = cel mai din stanga bit (b_{n-1})
cu conditia ca acel numar sa fie scris pe ***numarul corect de n biti***
(dupa cum spune ***Gama numerelor***)

n=2 ->	***[0;3]***	***[-2;+1]***
n=3 ->	***[0;7]***	***[-4;+3]***
n=4 ->	***[0;15]***	***[-8;+7]***
n=5 ->	***[0;31]***	***[-16;+15]***
n=6 ->	***[0;63]***	***[-32;+31]***
n=7 ->	***[0;127]***	***[-64;+63]***
n=8 ->	***[0;255]***	***[-128;+127]***
n=9 ->	***[0;511]***	***[-256;+255]***
n=10 ->	***[0;1023]***	***[-512;+511]***
. . .		

Exemple:

3 = 11b -> n=2 biti, dar +3 = 011b -> n=3 biti, iar -3= 101b -> n=3 biti
4=100b -> n=3 biti, dar +4=0101b -> n=4 biti, iar -4=100b -> n=3 biti

PASI:

I. Se gaseste ***gama*** cea mai potrivita pentru scrierea numarului (fara semn/cu semn)
II. Se gaseste ***n*** = nr minim de biti necesar scrierii corecte
III. Se scrie nr. in binar dupa ***regulile corespunzatoare (fara semn/cu semn)*** pe acel nr de biti

2.2.4. Conversii de numere cu semn

- ***Care este MSb la numerele reprezentate în convenţia cu semn ?***

Gama numerelor folosind n biţi,

numerele ***fără semn*** -> în **gama [0;2^n-1]**, iar

numerele ***cu semn*** -> în **gama [-2^{n-1};+2^{n-1}-1]**.

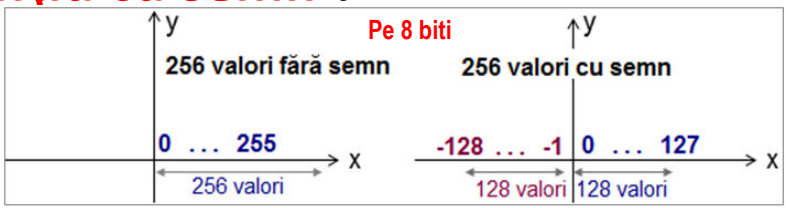


Figura 2.14. Reprezentarea a 256 valori fără semn vs. cu semn

MSb = cel mai din stanga bit (b_{n-1})

cu conditia ca acel numar sa fie scris pe ***numarul corect de n biti***
(dupa cum spune ***Gama numerelor***)

n=2 ->	***[0;3]***	***[-2;+1]***
n=3 ->	***[0;7]***	***[-4;+3]***
n=4 ->	***[0;15]***	***[-8;+7]***
n=5 ->	***[0;31]***	***[-16;+15]***
n=6 ->	***[0;63]***	***[-32;+31]***
n=7 ->	***[0;127]***	***[-64;+63]***
n=8 ->	***[0;255]***	***[-128;+127]***
n=9 ->	***[0;511]***	***[-256;+255]***
n=10 ->	***[0;1023]***	***[-512;+511]***
. . .		

Exemple:

3 = 11b -> n=2 biti, dar +3 = 011b -> n=3 biti, iar -3= 101b -> n=3 biti
4=100b -> n=3 biti, dar +4=0101b -> n=4 biti, iar -4=100b -> n=3 biti

DAR daca registrii nostri au 8 biti? Cum scriem aceste valori **pe 8 biti**?

3 = 00000011b, +3 = 00000011b, -3 = 11111101b
4 = 00001100b, +4 = 00000101b, -4 = 11111100b

Regula pentru extensie (ce fel de biti se insereaza *in fata lui MSb* pt a ajunge la dimensiunea dorita):

La numerele ***fara semn***, extensia se realizeaza intotdeauna cu **0**
La numerele ***cu semn***, extensia se realizeaza intotdeauna cu **MSb**

- **2.2.4. Conversii de numere cu semn**
- ***Care sunt pașii pentru conversia corectă în binar a unui număr cu semn ?***

Exemplul 1:

Numărul *fără semn* 125 se încadrează în gama [0,+127] ce foloseşte un număr de minim 7 biţi:

$[0,+2^n-1] \Rightarrow 2^n-1=127 \Rightarrow 2^n=128=2^7 \Rightarrow n=7 \Rightarrow$ **125 = 111 1101b**

Exemplul 2:

Numărul *cu semn* +125 se încadrează corect în gama [-128,+127] ce fol. un nr. de minim 8 biţi:

$[-128,+127] = [-2^{n-1},+2^{n-1}-1] \Rightarrow 2^{n-1}=128=2^7 \Rightarrow n-1=7 \Rightarrow n=8$

Numărul cu semn + 125 se va reprezenta corect pe n=8 biţi, şi nu pe doar 7 biţi;

deci **+125 = 0111 1101b**, și nu doar **111 1101b** cum s-a scris 125 ca număr fără semn

(și care ar fi -3 ca număr cu semn).

Reprezentare	Gama numerelor fără semn	Puterea lui 2
octet	0 … 255	$0 \dots 2^8-1$
cuvânt	0 … 65 535	$0 \dots 2^{16}-1$
dublucuvânt	0 … 4 294 967 295	$0 \dots 2^{32}-1$

Reprezentare	Gama numerelor cu semn	Puterea lui 2
octet	-128 … +127	$-2^7 \dots +2^7-1$
cuvânt	-32 768 … +32 767	$-2^{15} \dots +2^{15}-1$
dublucuvânt	-2 147 483 648 … +2 147 483 647	$-2^{31} \dots +2^{31}-1$

- **2.2.4. Conversii de numere cu semn**
- ***Cum ne putem verifica (conversia corectă în binar a unui număr cu semn) ?***

Pentru transformarea numărului din binar în zecimal, primul termen din descompunerea binară este considerat cu semnul **+** sau **–** după cum numărul este pozitiv sau negativ, toţi ceilalţi termeni fiind consideraţi pozitivi.

Regula care se aplică la verificare:

- MSb *este nul* doar pentru ***numere pozitive*** (cifra binară este 0),
- pt ***numere negative,*** MSb trebuie considerat **cu semn negativ** (va fi o *cantitate nenulă* întotdeauna).

Exemplul 1: Numărul *fără semn* 125 se scrie cu un număr de minim 7 biţi: **125 = 111 1101b,** dar se poate extinde la **125=0111 1101b**
Transformat inapoi pentru verificare: $0+2^6+2^5+2^4+2^3+2^2+0+2^0=125$ (punand 0, nu am adaugat ***CANTITATE, adica*** 2^7)

Exemplul 2: Numărul *cu semn* +125 se scrie cu un nr. de minim 8 biţi: **+125 = 0111 1101b**, și nu doar **111 1101b** = 125 ca număr fără semn (și care ar fi -3 ca număr cu semn).
Transformat inapoi pentru verificare: $+0+2^6+2^5+2^4+2^3+2^2+0+2^0=+125$ (punand 0, nu am adaugat ***CANTITATE, adica*** $+2^7$)

111 1101b = - 3?
Transformat inapoi pentru verificare: $\mathbf{-2^6}+2^5+2^4+2^3+2^2+0+2^0=$ **-64**+32+16+8+4+1 = -3 (***CANTITATE =*** $\mathbf{-2^6}$)

Exemplul 3: Numărul +37 scris pe 7 biţi este 010 0101b şi poate fi extins pe 8 biţi astfel: **0**0100101b;
transformat înapoi în zecimal (pentru verificare), acesta se va scrie: $\mathbf{0+0+2^5+0+0+2^2+0+2^1}$=+37, deci este corect chiar şi după extensie.

Exemplul 4: Numărul -37 în C2 a fost extins de la 7 biţi la 8 biţi: -37 = **11011011**b;
transformat înapoi în zecimal, va fi: $\mathbf{-2^7+2^6+0+2^4+2^3+0+2^1+2^0=}$ **-128+64+16+8+2+1**=-37 deci este corect.

- **2.2.4. Conversii de numere cu semn**
- La exprimarea ***valorilor pozitive*** în ***convenţia cu semn***
  - reprezentarea *e aproape identică* cu cea de la ***convenţia fără semn***: ***se mai adaugă doar un bit (MSb) de 0***.
- La exprimarea ***valorilor negative*** în ***convenţia cu semn:*** 3 conventii:
  - ***MS - Modul şi semn*** (MS, numit şi "cod direct"): se pastreaza marimea (modulul) si se inverseaza semnul

+127=**0**1111111**b** ; -127=**1**1111111**b**

+37 = **010 0101b** ; -37 = **110 0101b**

  - ***C1 – Complement faţă de 1*** (C1) (numit şi "cod invers"): se inverseaza toti bitii

+127=**0**1111111**b** ; -127=**1**0000000**b**

+37 = **010 0101b** ; -37 = **101 1010b**

  - ***C2 - Complement faţă de 2*** (C2)

(numit şi "*cod complementar*"):

se inverseaza toti bitii si se adauga apoi inca un 1

+127=**0**1111111**b** ; -127=**1**0000000**b +1 =**

-127=**1**0000001**b**

+37 = **010 0101b** ; -37 = **101 1010b +1**

-37 = **101 1011b**

Tabelul 2.5. Reprezentarea unui număr pe 3 biţi în diferite convenţii

Numărul în zecimal baza 10	Reprezentare în binar baza 2	Semnificaţia numărului în convenţie			
		fără semn	cu semn		
			MS	C1	C2
0	000	0	0	0	0
1	001	1	1	1	1
2	010	2	2	2	2
3	011	3	3	3	3
4	100	4	-0	-3	-4
5	101	5	-1	-2	-3
6	110	6	-2	-1	-2
7	111	7	-3	-0	-1

- **2.2.7.** Numere fracţionare
- **2.2.8.** Reprezentarea numerelor mixte
- **NU ABORDAM**

pentru reprezentare: $\mathbf{N=a_{n-1}\ a_{n-2}\ ...a_1\ a_0,\ a_{-1}\ a_{-2}\ ...\ a_{-(m-1)}\ a_{-m}}$ (2.1)

pt calculul valorii: $\mathbf{N}=a_{n-1}*\mathbf{q^{n-1}} + a_{n-2}*\mathbf{q^{n-2}} +...+a_1*\mathbf{q^1} + a_0 + a_{-1}*\mathbf{q^{-1}} + a_{-2}*\mathbf{q^{-2}} + ... a_{-m}*\mathbf{q^{-m}}$ (2.2)

Orice număr poate fi scris ca având **o parte întreagă** (obtinuta ca pana acum) şi **o parte fracţionară**, separate prin virgula binară (sa obt ca in model)

0,35 *2
0,7 *2 $a_{-1} = 0$
1,4 *2 $a_{-2} = 1$
0,8 *2 $a_{-3} = 0$
1,6 *2 $a_{-4} = 1$
1,2 *2 $a_{-5} = 1$
0,4 $a_{-6} = 0$
0,35 = 0,**0**1**0**1**1**0...b

Alte conversii:

6,35 = **110,0101 10...**b

25,C$_h$ = **2*16**^{1} + **5*16**^{0} + **12*16**$^{-1}$ = **32** + **5** + **0,75** = **37,75**

1010 0011,01b = **1010 0011,0100**b = **A3,4**h= **0A3,4**h

0A3,4h = **A3,4**h = **1010 0011,0100**b = **1010 0011,01**b

2.2.6. Interpretarea valorilor numerice

- Convenţiile de reprezentare a numerelor

  -> datele din memoria PC-ului se pot interpreta diferit.

- UCP nu ştie semnificaţia octetului (sau cuvântului, etc) -> ??? (fara semn/cu semn) ???

Există instrucţiuni ale CPU care ţin cont de convenţia de reprezentare a numărului (fără semn sau cu semn) ?

- Instrucţiunea folosită *poate sugera* modul de reprezentare al datelor
  - există instrucţiuni :
    - înrudite ("pereche") MUL – IMUL, DIV - IDIV, una pt nr *fără semn*, iar cealaltă pt *cu semn*;
    - care considera automat numerele ca fiind cu semn: CBW, CWD, CDQ, …
  - Dar majoritatea instructiunilor nu au astfel de mecanisme: numerele pot fi considerate fara semn sau cu semn

    Ex: add, sub, pe biti, etc

  - la programarea în limbaj de asamblare: ***programatorul este cel care trebuie să se asigure de folosirea corectă a acestor instrucţiuni***, în funcţie de tipul datelor.
- Definirea datelor este diferită faţă de limbajele de nivel înalt, deoarece în asamblare nu există posibilitatea de a defini *tipuri de date*.

De ex: valoarea C0h poate fi interpretată ca numărul fără semn 192 sau numărul cu semn -64 în C2.

!!! implicit, dacă e vorba de un număr negativ, acesta a fost reprezentat în C2

2.2.6. Interpretarea valorilor numerice

- Convenţiile de reprezentare a numerelor

  -> datele din memoria PC-ului se pot interpreta diferit.

- UCP nu ştie semnificaţia octetului (sau cuvântului, etc) -> ??? (fara semn/cu semn) ???

Există instrucţiuni ale CPU care ţin cont de convenţia de reprezentare a numărului (fără semn sau cu semn) ?

- Instrucţiunea folosită *poate sugera* modul de reprezentare al datelor
  - există instrucţiuni :
    - înrudite ("pereche") MUL – IMUL, DIV - IDIV, una pt nr *fără semn*, iar cealaltă pt *cu semn*;
    - care considera automat numerele ca fiind cu semn: CBW, CWD, CDQ, …

  - Dar majoritatea instructiunilor nu au astfel de mecanisme: numerele pot fi considerate fara semn sau cu semn

    Ex: add, sub, pe biti, etc

```
Mov al, -4          mov al,-4
Mov bl, 2
Mul bl              imul BL
```

2.2.6. Interpretarea valorilor numerice

Există vreo metodă de a verifica dacă rezultatul obţinut în urma unei operaţii aritmetice este corect ?

- În gen., atunci când se realizează o operaţie aritmetică, datorită dublei interpretări a valorilor (*fără semn* sau *cu semn*) rezultatul obţinut poate să nu fie atât de sugestiv pe cât ar trebui pentru a fi interpretat corect.
- ***De exemplu***, să considerăm **o operaţie de adunare pe 3 biţi**;

dacă adunăm la nr 3 încă un 1, din 011b se transformă în 100b pentru care sunt posibile cele 2 interpretări:

- dacă numărul se consideră interpretarea fără semn, atunci rezultatul obţinut **este corect** 100b=4 (am adunat 3+1 și am obţinut 4);
- în schimb, dacă se consideră interpretarea cu semn, atunci rezultatul **nu mai este corect**, pentru că interpretarea lui 100b ca număr cu semn este - 4, și nu 4 ! Altfel spus, am adunat 3 cu 1 și am obţinut -4 în loc de 4; aceasta din cauză că rezultatul ar fi avut nevoie de încă un bit pentru a fi stocat în mod corect (ca 0100b)– spunem că *am depășit domeniul sau gama de reprezentare a numărului,* sau în engleză se folosește termenul ***overflow***.

- Concluzie: dacă am fi avut un bit suplimentar, rezultatul ar fi putut fi interpretat **corect în ambele situaţii** (în primul caz folosind extensia valorii).
- Problema: regiștrii existenţi în CPU nu-și modifică forma;
  - la 8086 aceștia rămân tot timpul de 8 sau 16 biţi, nu putem avea regiștri de 9 sau 17 biţi pentru a stoca date.
  - DAR - ca programatori avem indicatori (“flags”) care pot juca acest rol de bit suplimentar.

    (aici in special e vorba de Carry flag)

2.2.6. Interpretarea valorilor numerice

Ce sunt flagurile aritmetice și cum ne pot ajuta ele în interpretarea rezultatelor?

- pentru a ajuta la gestionarea corectă a unor situaţii excepţionale, se pot interpreta așa numitele ***flaguri aritmetice***.

Acestea pot arăta una din următ. situaţii:

- ***Overflow*** – apare (sau spunem că se setează flagul corespunzător) când se depăşește gama de reprezentare a numărului: dacă rezultatul obţinut nu a încăput în gama destinată stocării lui, atunci există un indicator care va avea valoarea 1; altfel, acest indicator va avea valoarea 0.
- ***Carry*** sau ***Borrow*** - apare atunci când se realizează un transport: există un indicator ce va avea valoarea 1 în cazul în care ultima operaţie efectuată a generat un transport în/ din afara domeniului de reprezentare a numărului și valoarea 0 în caz contrar;
- - operaţie de transport *în afara rezultatului*, probabil *obţinut printr-o operaţie de adunare*, sau
- - operaţie de transport *din afara rezultatului* (caz în care își schimbă denumirea în borrow, în română împrumut) și probabil s-a *obţinut printr-o operaţie de scădere*;
- ***Zero*** - semnalizează dacă rezultatul ultimei operaţii este egal cu zero.
- ***Sign*** - semnalizează dacă rezultatul ultimei operaţii este un număr strict negativ.

Ipotetic: sa pp. ca avem registri de 4 biti !!! (pentru simplitate)

Exemple: Se vor considera operaţiile realizate pe 4 biţi, deci și rezultatul va fi pe 4 biţi:

- a) 2h+2h=4h -> **C=0, Z=0, S=0, O=0**;
- b) 3h+7h=Ah -> **C=0, Z=0, S=1, O=1**

a apărut depășire de gamă: +10 s-ar fi scris corect pe 5 biţi, folosind gama [-16;+15], iar ca numere fără semn nu a apărut transport înafara domeniului de reprezentare, numărul 10 este scris corect;

- c) 7h+Bh=2h -> **C=1, Z=0, S=0, O=0**

ca numere cu semn, se operează 7+(-5) =2, dar ca numere fără semn, a apărut transport înafara domeniului de reprezentare;

- d) Ah+Bh=5h -> **C=1, Z=0, S=0, O=1**

a apărut depășire de gamă: se operează (-6)+(-5)=-11 și care s-ar fi scris corect pe 5 biţi, folosind gama [-16;+15], iar ca numere fără semn a apărut transport înafara domeniului de reprezentare, numărul 10+11=21 s-ar fi scris corect pe 5 biţi, în gama [0; 31];

- e) 8h+8h=0h -> **C=1, Z=1, S=0, O=1**

a apărut depășire de gamă: se operează (-8)+(-8)=-16 și care s-ar fi scris corect pe 6 biţi, folosind gama [-32;+31], iar ca numere fără semn a apărut transport înafara domeniului de reprezentare, numărul 8+8=16 s-ar fi scris corect pe 5 biţi, în gama [0; 31].

CPU nu are de unde să știe ce semnificaţie vrem noi să dăm numerelor din PC

- CPU reprezintă numărul în binar și atât; ce poate face ulterior cu acest număr este să interpreteze bitul c.m.s. al acestei valori ca fiind un bit ce arată semnul numărului (sau să nu îl interpreteze astfel).
  - ***Exemplu:*** valoarea -1 care se reprezintă în binar ca FFFFh, pentru CPU poate însemna la fel de bine și 65535 (adică în reprezentarea fără semn).
- Va trebui să acordăm importanţă la interpretarea valorilor cu care lucrăm:
- adunarea: nu avem de unde ști că adunăm două numere pozitive și rezultatul intră în depășire și deci obţinem (pe același număr de biţi) un rezultat negativ, *decât dacă interpretăm noi corect rezultatele și flagurile*.
  - ***Exemplu:*** 3h + 7h= Ah – care e un număr negativ dacă e interpretat în convenţia cu semn; am adunat două nr pozitive (se vede simplu, după c.m.s. bit care este 0) și am obţinut un număr negativ: 3 cu 7 și am obţinut 10 sau -6, depinde cum vrem să-l interpretăm.
- ordonarea unui sir de valori – e important sa stim daca sunt numere fara semn / cu semn
  - ***Exemplu:*** ordonati numerele (reprezentate pe 4 biti): fh, 5f, 2h, 9h, 7h
- ? Ce rezulta daca nr sunt considerate uns? Dar s?

2.2.9. Operaţii de bază în diverse sisteme de numeraţie

- **2.2.9.1. Operaţii aritmetice:**
- **Analogie cu operaţiile realizate în zecimal:**
- ***Adunarea* a două numere** în baza q este o operaţie modulo q, deci cifra cu valoarea cea mai mare va fi q-1. Dacă rezultatul adunării depășește această valoare, va apărea un transport. Biţii de transport au fost indicaţi deasupra, cu o culoare diferită, așa cum se notau aceștia în clasele primare, când am învăţat să adunăm numerele în zecimal. De exemplu, la adunarea în hexazecimal, Bh+6h=11+6=17 care se va scrie 1h, adică 17-***16*** (*baza*) și o unitate merge mai departe, cu transport la cifra de rang mai mare.
- La ***scăderea* a două numere** de rang i, dacă cifra descăzutului este mai mică decât cifra scăzătorului, apare un împrumut (unitatea este 1, adică BAZA) de la rangul i+1.

Adunarea/scaderea a 2 numere pe n biţi va produce un rezultat tot pe n biţi, însă e posibil ca acesta să fie eronat dacă nu se consideră posibilele situaţii arătate de Carry și Overflow.

Exemple: Să se *însumeze* numerele de mai jos:

1 1 1	1	1	1	1 1
010101b+	21_{10} +	23_{10} = 10111b +	0A2Bh+	$6B34_{16}$ +
100111b	39_{10}	19_{10} = 10011b	1236h	$4DF1_{16}$
111100b	60_{10}	42_{10}=101010b	1C61h	$B925_{16}$

Exemple: Să se *scadă* numerele. Vom avea:

100111b-	39_{10} -	23_{10} = 10111b -	2A3Bh-	$6B34_{16}$ -
010101b	21_{10}	18_{10} = 10010b	154Dh	$4DF1_{16}$
010010b	18_{10}	5_{10} = 00101b	14EEh	$1D43_{16}$

```
 1 1 1
0111b+          7 +              +7 +
0101b           5                +5
1100b          12 (fără semn)    - 4 (cu semn)
```

Din calculele de mai sus scrise în zecimal, se poate observa că dacă numerele sunt interpretate în reprezentarea cu semn, rezultatul este eronat, întrucât +12 ar fi avut nevoie de 5 biţi în reprezentare, al 5-lea bit fiind 0; numărul corect s-ar fi scris 01100b (Overflow=1, Carry=0).

```
 1 1
 1110b+         14 +             - 2 +
 0101b           5               +5
 0011b           3 (fără semn)   +3 (cu semn)
```

Din calculele de mai sus scrise în zecimal, se poate observa că dacă numerele sunt interpretate ca numere fără semn, de data aceasta, rezultatul este eronat, întrucât 17 ar fi avut nevoie de 5 biţi în reprezentare, al 5-lea bit fiind 1; numărul corect s-ar fi scris 10011b (Overflow=0, Carry=1).

Pe ce lungime trebuie stocat rezultatul unei anumite operaţii ?

Concret, la realizarea operaţiilor aritmetice de până acum, vom putea lucra astfel:

- - ***se pot aduna 2 numere*** **scrise pe n biţi**, rezultatul obţinut va fi tot pe n biţi, însă trebuie interpretate flagurile întrucât ar putea apărea depășire de capacitate (practic unele rezultate ar putea avea nevoie de scrierea pe n+1 biţi); pentru realizarea operaţiei de adunare în simulator se va folosi operatorul +, iar pe procesorul 8086 operaţia a fost implementată prin instrucţiunea **ADD**;
- - ***se pot scădea 2 numere*** **scrise pe n biţi**, rezultatul obţinut va fi tot pe n biţi, însă trebuie interpretate flagurile întrucât la scădere ar putea apărea depășire de capacitate în sensul nevoii de împrumut - borrow (practic unele rezultate ar putea avea nevoie de scrierea pe n+1 biţi pentru a marca faptul că a fost nevoie de un împrumut pentru a putea realiza scăderea); pentru realizarea operaţiei de scădere în simulator se va folosi operatorul -, iar pe procesorul 8086 operaţia a fost implementată prin instrucţiunea **SUB**;
- - ***se pot înmulţi 2 numere*** **scrise pe n biţi**, rezultatul obţinut va fi pe un număr dublu de biţi, adică 2n biţi; aceasta înseamnă că va trebui să ţinem cont de acest aspect atunci când vom citit rezultatul obţinut - trebuie interpretat corect tot rezultatul, așa cum apare el scris pe toţi cei 2n biţi; pentru realizarea operaţiei de înmulţire în simulator se va folosi operatorul *, iar pe procesorul 8086 operaţia a fost implementată în 2 forme, pentru considerarea valorilor ca numere fără semn, respectiv cu semn, prin instrucţiunile **MUL** și **IMUL**;
- - invers, ***se poate realiza împărţirea a 2 numere***, dacă deîmpărţitul se consideră scris pe 2n biţi, iar împărţitorul pe n biţi. Rezultatele obţinute (scrise tot pe n biţi fiecare) ce se pot considera vor fi câtul și restul împărţirii, iar operatorii corespunzători sunt /, respectiv %. Pentru implementarea operaţiilor corespunzătoare pe procesorul 8086, se folosește doar o singură instrucţiune având 2 forme: **DIV** și **IDIV** (pentru considerarea valorilor ca numere fără semn, respectiv cu semn); citirea câtului sau a restului se va realiza diferit, adică rezultatele obţinute se vor furniza în structuri diferite ale procesorului. Prin cunoașterea regulilor prezentate mai sus (de către programator), se va putea lucra ușor cu oricare din aceste operaţii la nivel de programe scrise în limbaj de asamblare.

2.2.9.2. Operaţii logice pe şiruri de biţi

- Există ***4 operaţii logice*** majore pe biţi: **NOT**, **AND**, **OR** şi **XOR**, iar Tabelul 2.8 furnizează tabelele de adevăr corespunzătoare. Operaţiile logice se realizează asupra şirurilor de biţi, deci se va aplica o funcţie logică fiecărui bit în parte (din reprezentarea numărului), la acest tip de instrucţiuni neexistând transport.

Tabelul 2.8. Reguli de obţinere a valorilor la diferite operaţii logice efectuate în binar

NOT a unei cifre binare

NOT	0	1
	1	0

AND între 2 cifre binare

AND	0	1
0	0	0
1	0	1

OR între 2 cifre binare

OR	0	1
0	0	1
1	1	1

XOR între 2 cifre binare

XOR	0	1
0	0	1
1	1	0

- Operaţia **NOT** (Negare logică bit cu bit) are ca efect negarea tuturor biţilor numărului din binar (de fapt calculează complementul faţă de 1 al acestuia).
- ***Exemplu:*** NOT 1234h ;
- 1234h= 0001 0010 0011 0100b => not 1234h =1110 1101 1100 1011b =0EDCBh
- **Operaţiile AND** (ŞI logic bit cu bit), **OR** (SAU logic bit cu bit) şi **XOR** (SAU-exclusiv bit cu bit) realizează operaţiile corespunzătoare, pe procesorul 8086 instrucţiunile având exact același nume, iar în simulator fiind disponibili și următorii operatori:
- operatorul **~** pentru operaţia de inversare – **NOT** – inversează valoarea tuturor biţilor;
- operatorul **&** pentru operaţia logică **AND** la nivel de bit;
- operatorul **^** pentru operaţia logică **XOR** la nivel de bit;
- operatorul **|** pentru operaţia logică **OR** la nivel de bit.

- operaţiile AND și OR se utilizează în special atunci când se doreşte **mascarea** (se folosesc biţi de 0, respectiv de 1 pentru mască) anumitor biţi, în timp ce instrucţiunea XOR se foloseşte când se doreşte **complementarea** anumitor biţi.

Tabelul 2.9. Mascarea biţilor folosind operaţiile AND, OR și XOR

AND	OR		XOR
xxxx xxxx operand	xxxx xxxx operand		xxxx xxxx operand
0000 1111 masca	0000 1111 masca		0000 1111 masca
0000 xxxx rezultat	xxxx 1111 rezultat		xxxx $\overline{xxxx}$ rezultat

- ***Exemple:*** Presupunând valorile 0110b și 0011b, operaţiile logice pot fi realizate astfel:
- Operaţia not: ~ 0110b = 1001b
- ~0011b = 1100b
- Operaţia and: 0110b & 0011b = 0010b
- Operaţia xor: 0110b ^ 0011b = 0101b
- Operaţia or: 0110b | 0011b = 0111b

2.2.9.3. Operaţii de deplasare a şirurilor de biţi

- ***Deplasarea spre stânga este echivalentă cu o înmulţire.***
- ***Exemplu:*** Numărul 0011b care este 3 în zecimal, deplasat înspre stânga cu o poziţie, va produce numărul 0110b, care este 6, iar deplasat cu 2 poziţii va produce 1100b care este 12, interpretat ca număr fără semn.

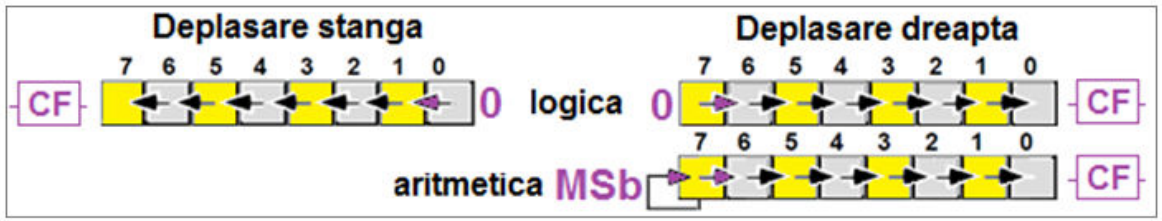


Figura 2.16. Reprezentarea operaţiilor de deplasare spre stânga și spre dreapta

- **La o operaţie de deplasare (logică) spre stânga,** pe locul bitului LSb, adică bitul b0, se va introduce un 0, iar bitul MSb, și anume b7 va ajunge în flagul Carry. Operaţia de deplasare spre stânga cu o poziţie este echivalentă cu înmulţirea valorii cu 2^1. În general se foloseşte mnemonica **SHL** (shift left) pentru a desemna o astfel de operaţie în cadrul procesorului 8086, dar în simulator se poate folosi și operatorul **<<**.
- ***Exemplu:*** numărul 6 scris în binar şi deplasat la stânga cu o poziţie va furniza numărul 12, considerat ca număr fără semn: 0110b << 1 = 1100b.

- ***Deplasarea spre dreapta este echivalentă cu o împărţire.***
- ***Exemplu:*** Numărul 0110b care este 6 în zecimal, deplasat înspre dreapta cu o poziţie, va produce numărul 0011b, care este 3, iar deplasat cu 2 poziţii va produce 0001b care este 1.
- **O operaţie de deplasare (logică) spre dreapta** funcţionează în mod asemănător celei spre stânga, doar că datele se deplasează în sens opus, spre dreapta, aşa cum arată Figura 2.16 (din dreapta). Totuși, aici trebuie menţionat că există 2 posibilităţi, așa cum reiese și din figură: pe locul bitului MSb, adică bitul b7, se va introduce:
- - ori **un 0**, caz în care se spune că s-a realizat o ***deplasare logică spre dreapta***,
- - ori **un bit identic cu bitul MSb**, caz în care se spune că s-a realizat o ***deplasare aritmetică spre dreapta***.
- În general, se foloseşte mnemonica **SHR** (shift right), pentru a desemna o operaţie de ***deplasare logică*** spre dreapta în cadrul procesorului 8086, dar în simulator se poate folosi și operatorul **>>**. Operaţia de ***deplasare aritmetică*** spre dreapta se poate obţine folosind mnemonica **SAR** (shift aritmetic to right) în cadrul procesorului 8086, dar în simulator nu are atribuit vreun operator.
- Operaţia de deplasare spre dreapta rotunjeşte rezultatul înspre întregul cel mai apropiat, care e mai mic sau egal cu rezultatul. În oricare din cazurile de deplasare spre dreapta, bitul LSb, și anume b0 va ajunge în flagul Carry (CF).

- ***Exemplu:*** Numărul 0101b care este 5 în zecimal, deplasat înspre dreapta cu o poziţie, va produce numărul 0010b, care este 2 (bitul de 1 care s-a pierdut poate fi găsit în flagul Carry), iar deplasat cu 2 poziţii va produce 0001b care este 1 (flagul Carry va conţine acum un bit de 0).
- La modul general, o înmulţire cu 2^n a numărului, înseamnă o deplasare spre stânga cu n biţi, iar o împărţire cu 2^n a numărului, înseamnă o deplasare spre dreapta cu n biţi.
- Există situaţii când sunt necesare operaţii de deplasare a numerelor fără semn, iar aceste deplasări trebuie realizate prin operaţii de deplasare logică; în situaţiile când se dorește deplasarea numerelor cu semn, se vor folosi operaţii de deplasare aritmetică, întrucât acestea nu vor modifica semnul numerelor, ci doar valoarea lor. Similar, la deplasarea spre stânga trebuie ţinut cont de semnul numărului și de posibilele alterări ale acestuia prin operaţia de deplasare (pentru a nu obţine un rezultat eronat).
- ***Exemple:*** Numere fără semn: 0011b << 2 = 1100b adică 3 x 4 = 12
- Numere cu semn: 1010b >> 1 = 1101b adică – 6 : 2= - 3

2.2.9.4. Operaţii de rotire a şirurilor de biţi

Aceste operaţii de rotire la nivel de șiruri de biţi se comportă asemănător cu cele de deplasare, cu diferenţa că bitul care iese înafara reprezentării este cel care completează din cealaltă direcţie rezultatul, așa cum arată Figura 2.17.

Figura 2.17. Reprezentarea operaţiilor de rotire spre stânga și spre dreapta

Operaţiile de rotire mai au o variantă disponibilă și anume prin implicarea flagului Carry în cadrul operaţiei de rotire. Acesta acţionează ca o celulă suplimentară, așa cum reiese din Figura 2.18.

Figura 2.18. Reprezentarea operaţiilor de rotire spre stânga și spre dreapta cu CF

- Aceste operaţii de rotire a șirurilor de biţi, indiferent că folosesc sau nu CF în operaţia de rotire, nu au atribuit vreun operator în simulator, dar pe procesor acestea există prin instrucţiunile specifice: **ROL** și **ROR**, resp. cu implicarea lui Carry Flag: **RCL** și **RCR**.
- ***Exemple***: Valoarea 0011b=3, rotită spre stânga cu 2 poziţii va furniza valoarea 1100b; ca operaţie, aceasta nu are definit un operator în cadrul simulatorului. Aceeași valoare, dar deplasată cu 4 poziţii, va furniza tot 0011b.
- ***Exemple***: Dacă se presupune că valoarea din flagul Carry este 1, deci CF=1 și se repetă prima operaţie din exemplul anterior (dar prin rotire spre stânga cu participarea flagului Carry), atunci se va obţine 0111b după prima rotire, și apoi 1110b după cea de-a doua. În cadrul celei de-a doua operaţii, rezultatul final, deci după 4 rotiri va fi: 1001b și CF=1.
- ***Exemple***: Fie valoarea 101100b care se dorește a fi rotită spre dreapta, fără participarea CF cu 2 și respectiv 4 poziţii; în primul caz rezultatul va fi 110010b, iar în cel de-al doilea caz rezultatul va fi 001011b.
- ***Exemple***: Același exemplu, pentru valoarea 101100b, dar acum rotită spre dreapta cu participarea CF=0, va furniza rezultatele 001011 și CF=0, respectiv în cel de-al doilea caz rezultatul va fi 100010b și CF=1.

Concluzie finala Cap 2:

- Exista instructiuni specifice numerelor fara semn / cu semn

- Dar exista si instructiuni care nu interpreteaza deloc valorile din registrii !!! E rolul programatorilor sa faca acest lucru !