

# OPERAȚII PE BIȚI

Adunare:  $1 + 1 \rightarrow 0$  și carry 1

Scădere:  $0 - 1 \rightarrow 1$  și carry -1

$$\begin{array}{r} \text{C:} \\ 0101110011110011 \\ + 0111000011110000 \\ \hline 1100110111100011 \end{array}$$

Înmulțire: Verific dacă se înmulțește cu o putere a lui 2 (2, 4, 8, 16).

Atunci shiftz la stânga (1, 2, 3, 4).

bit - binary digit

baza x - are cifre de la 0 la x - 1

Numere pozitive:

Numărul Y din baza X =  $\sum_{i=0}^{n-1} b_i \cdot X^i$ , n este numărul de biți din reprezentare

$b_0$  e LSB,  $b_{n-1}$  e MSB

Conversii:

-  $B_{old} = 10$  are 10 cifre (0 - 9)

$B \rightarrow B^p$  - Grupăm din B în câte P cifre,  $p = \log_b B^p$

-  $B_{new} = 100$  are 100 de cifre (0 - 99)

Numere negative (Pentru care, uneori, e nevoie de circuite speciale)

$|S|$  - S e MSB, bit rezervat pentru semn, deci sunt disponibili  $n - 1$  biți din reprezentare!

Numărul -Y din baza X =  $-b_i \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$

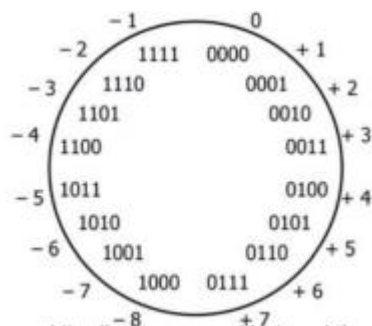
Complement față de doi:

$(-x)_{b10} \rightarrow b2$  1. Scriem  $|x|$  în binar 2. Inversăm biții 3. Adăugăm 1

$(-x)_{b2} \rightarrow b10$  1. Scriem numărul așa cum e 2. Inversăm biții 3. Adăugăm 1

Folosim sistemul binar: Același algoritm de adunare, pot fi folosite aceleași circuite pentru adunare naturală

• un exemplu explicit:  $(4215)_{10} = (1000001110111)_2$



2	4215		
2	2107	1	← LSB
2	1053	1	
2	526	1	
2	263	0	
2	131	1	
2	65	1	
2	32	1	
2	16	0	
2	8	0	
2	4	0	
2	2	0	
2	1	0	
	0	1	← MSB

Cel mai mare număr care se poate reprezenta pe N biți:  $2^N - 1$

Cel mai mare număr care se poate reprezenta (complement față de 2) pe N biți:  $2^{N-1} - 1$

Cel mai mic:  $-2^{N-1}$

MSB pe poziția:  $N - 1$  (maximal în reprezentare)

X – Număr natural. Biți necesari pentru reprezentare:  $\text{ceil}(\log_2 X)$

X – K cifre în HEX, de câți biți e nevoie pentru BIN:  $K * \log_2 16 = 4K$

X – K cifre în BIN, de câți biți e nevoie pentru HEX:  $\text{ceil}(K / 4)$

X – K cifre în DEC, de câți biți e nevoie pentru BIN:  $\text{ceil}(K * \log_2 10)$

HEX<sub>old</sub> BIN<sub>new</sub>

BIN<sub>old</sub> HEX<sub>new</sub>

DEC<sub>old</sub> BIN<sub>new</sub>

**De ce funcționează complementul față de 2?**

$$\begin{aligned}
 -\left(-2^N + \sum_{i=0}^{N-1} b_i 2^i\right) &= 2^N - \sum_{i=0}^{N-1} b_i 2^i \\
 2^{N+1} &= \sum_{i=0}^N 2^i + 1 \\
 &= \sum_{i=0}^{N-1} 2^i + 1 - \sum_{i=0}^{N-1} b_i 2^i \\
 &= \sum_{i=0}^{N-1} (1 - b_i) 2^i + 1 \\
 &= (\text{inversam bitii}) + 1
 \end{aligned}$$

## BINARY FIXED-POINT

...	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$	$2^{-5}$	$2^{-6}$	$2^{-7}$	...
-----	-------	-------	-------	-------	-------	-------	-------	-------	----------	----------	----------	----------	----------	----------	----------	-----

$2^{-1}$	0.5
$2^{-2}$	0.25
$2^{-3}$	0.125
$2^{-4}$	0.0625
$2^{-5}$	0.03125
$2^{-6}$	0.015625
$2^{-7}$	0.0078125
$2^{-8}$	0.00390625
$2^{-9}$	0.001953125
$2^{-10}$	0.0009765625
$2^{-11}$	0.00048828125
$2^{-12}$	0.000244140625
$2^{-13}$	0.0001220703125

**Ex: 101.101**

$$\begin{cases}
 101 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 4 + 1 = 5 \\
 101 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 0.5 + 0.125 = 0.625
 \end{cases}$$

$$\Rightarrow 101.101 = 5.625$$

$\Rightarrow$

**Ex: 3.75**

$$\begin{cases}
 3 = 1 \cdot 2^1 + 1 \cdot 2^0 \\
 0.75 = 1 \cdot 2^{-1} + 1 \cdot 2^{-2}
 \end{cases}$$

$$\Rightarrow 3.75 = 11.11$$

## LOGARITM ÎNTREG

9. Demonstrați că  $\lfloor \log_2 x \rfloor = i_{\max}$  unde  $x$  este un număr dat pe  $N$  biți iar  $i_{\max} = \max \{i \mid b_i = 1, \forall i = 0, \dots, N-1\}$  unde  $b_i$  reprezintă al  $i$ -lea bit din reprezentarea binară a numărului  $x$ . De exemplu: dacă  $x = 00101110$  (46 zecimal) atunci  $\lfloor \log_2 x \rfloor = 5$ .

- **arătați că**  $\lfloor \log_2 x \rfloor = i_{\max}$
- **pornim de la reprezentarea binară și aplicăm logaritmul**

$$\begin{aligned}x &= \sum_{i=0}^{N-1} b_i 2^i \\ \log_2 x &= \log_2 \left( \sum_{i=0}^{N-1} b_i 2^i \right) \\ &= \log_2 \left( 2^{i_{\max}} \left( \sum_{i=0}^{N-1} b_i \frac{2^i}{2^{i_{\max}}} \right) \right) \\ &= \log_2 2^{i_{\max}} + \log_2 \left( \left( \sum_{i=0}^{N-1} b_i \frac{2^i}{2^{i_{\max}}} \right) \right) \\ &= i_{\max} + C, \quad C < 1\end{aligned}$$

## OVERFLOW-SAFE BINARY-SEARCH

```
int binarySearch1(int arr[], int start, int end, int x)
{
    if (end >= start)
    {
        int mid = start + (end - start) / 2;

        if (arr[mid] == x)
            return mid;

        if (arr[mid] > x)
            return binarySearch1(arr, start, mid - 1, x);

        return binarySearch1(arr, mid + 1, end, x);
    }

    return -1;
}
```

# GENERAL / ISTORIC

ENIAC – 1945 – 1955 / USA – 1000 op/s (aka Electronic Numerical Integrator and Computer)

HPE CRAY – 2021 / USA – 1714 petaoperații/s

”compute” – Infrastructura hardware pe care rulează algoritmii

Putem scădea costul de calcul pentru ML prin: algoritmi eficienți, hardware dedicat

Varianta A – Cea mai rapidă

(Calculul a două matrice)

Complexitate:  $O(n^3)$

Operații elementare: Adunare, înmulțire

Numărul operațiilor elementare: 2 pentru fiecare for (pe lângă asta, în fiecare for sunt câte n operații elementare de incrementare)

Număr de operații:  $2 * n^3$

```
# varianta A
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

# PERSONALITĂȚI

*Richard Hamming* - “The purpose of computing is insight, not numbers.”. Codul Hamming pentru integritatea mesajelor transmise la distanțe mari.

*Blaise Pascal* – Creează Pascaline (1642) – Calculator mecanic, capabil de +-, jucăria aristocrației | Limbajul Pascal e numit în onoarea lui

*Gottfried Wilhelm Von Leibniz* – Studiază sistemul binar, extinde mașina lui Pascal (adaugă \* /)

*George Boole* – Scrie “The Laws of Thought” | Introduce logica booleană și analizează operațiile logice de bază: Negatie / Conjuncție / Disjuncție / Disjuncție exclusivă | Ele stau la baza teoriei informației

*Charles Babbage* – Proiectează Mașina Diferențială Nr. 2 – prima mașină de calcul mecanică programabilă – prototipuri de 13 tone – Tatăl calculatoarelor moderne

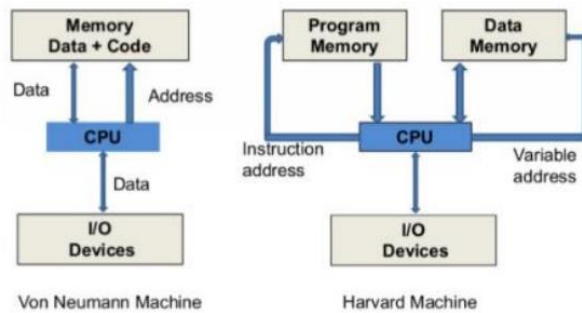
*Ada Lovelace* – Colaborează cu Babbage | Scrie un program care calculează numerele numere Bernoulli – Nu existau limbaje de programare, dar scrie pași pentru execuție – Primul programator

*Konrad Zuse* – Introduce o serie de calculatoare: Z1, Z2, Z3, Z4 | Folosește sistemul binar | Instrucțiuni stocate pe o bandă | Introduce reprezentarea și calculul în virgulă mobilă / floating point | Aproape totul în izolare, pe perioada celui de-Al II-lea Război Mondial.

*Alan Turing* – Decryptează rapid mesaje Enigma folosind mașina The Bomb - Brute-Force search | Introduce Mașina Turing pentru algoritmi Turing Complete – Sistem de analiză și recunoaștere a unui număr mare de date | Introduce Testul Turing – Pot mașinile să gândească?

*John Von Neumann* – Contribuie la crearea primului calculator electronic: ENIAC | Creează EDVAC – Sistem binar cu programe stocate. Introduce arhitectura von Neumann

***Arhitectura von Neumann:***



*Claude Shannon*: Părintele teoriei informației (o inventează literally) | Demonstrează că problemele de logică booleană se pot rezolva cu circuite electronice | Teorema de eșantionare Shannon – Nyquist: Analog <-> Digital fără a pierde date

*Grace Hopper* – Contribuitoare în dezvoltarea de limbaje high-level și scrie compilatorul COBOL

*Margaret Hamilton* - Scrie cod pentru software-ul de la bord pentru misiunea Apollo

*Barbara Liskov* – Design pattern: Principiul substituției Liskov + altele în distributed computing

*Rivest Shamir Adleman* – Creatorii primului sistem de criptare cu chei, public: RSA. Folosește numere prime. Foarte util în tranzacții bancare.

*Diffie Hellman* – Schimbul de cheie Diffie-Hellman – Soluția în a trimite mesaje secrete într-un canal de comunicație nesecurizat.

*Ritchie and Thompson* – Dezvoltatorii limbajului C | Pun bazele sistemelor open-source, creează Unix OS.

*Richard Stallman* – Contributori la GNU Project.

*Linus Torvalds* – Creatorul Linux și GIT (pentru controlul versiunilor)

*Steve Jobs* – Cofondează Apple, Pixar

*Bill Gates* – Fondator Microsoft <3 | Acum CEO este Satya Nadella

*Jeff Bezos* – Fondator & fost CEO Amazon

*Mark Zuckerberg* – CEO Meta

*Larry Page and Sergey Brin* – Fondatorii Google

Idei mari:

- De la mecanic la electric
- De la o mașină care face un singur lucru la o mașină programabilă
- Design modular (pe module)
- Teorie despre ce este posibil pe aceste mașini
- Dorința de a face lucrurile optim, la limită și fără risipă

După Al Doilea Război Mondial, dorința de cercetare în domeniul calculatoarelor crește în ritm exponențial

Actori importanți: Grupuri profesionale (IEEE, ACM, Bell Labs) și state (Statele Unite: DARPA)

# TEORIA PROBABILITĂȚILOR

$$P = \frac{\text{Numărul cazurilor favorabile}}{\text{Numărul cazurilor posibile}} \quad (\text{atunci când evenimentul nu este influențat de mediul exterior})$$

$P1 \cdot P2$  (probabilitatea ca două evenimente **INDEPENDENTE** să se întâmple, fără să fie influențate de mediul exterior) Este intersecția dintre evenimentul care se putea întâmpla primul și cele totale. Trebuie să ținem cont de cazurile care sunt favorabile pentru ambele evenimente.

## TEORIA INFORMAȚIEI

Informația: Date care afectează (aproape mereu reduce) incertitudinea despre un fenomen.  
Se poate acumula, este constantă (nu este creată/distrusă).

$$I(x_i) = \log_2 \left( \frac{1}{p_i} \right) \quad (\text{Câtă informație ne oferă o valoare în funcție de probabilitatea ca ea să apară})$$

Calculăm cantitatea de informație folosind X (variabila aleatoare):

N – Numărul de valori distincte:  $x_1, x_2 \dots x_n$

P – Fiecare valoare apare cu probabilitatea  $p_1, p_2 \dots p_n$

$p_i$  mai mic  $\rightarrow$  Obținem cantitate mai mare de informație

Putem privi formula astfel: 
$$I(x_i) = \log_2 \left( \frac{1}{\frac{1}{M}} \right) = \log_2 \left( \frac{N}{M} \right) = \text{rezultat în biți!}$$

N – Numărul total de evenimente

M – Numărul de evenimente favorabile

- **entropia**

- valoarea medie de informației primită despre o variabilă X

$$H(X) = E(I(X)) = \sum_{i=1}^N p_i \log_2 \frac{1}{p_i} = \sum_{i=1}^N -p_i \log_2 p_i$$

- $H(X)$  se numește entropia lui X
- $I(X)$  este informația despre X
- $E$  este "expected value", operația care calculează valoarea medie
- exemplu:  $X = \{A, B, C, D\}$  cu probabilități  $\{1/3, 1/2, 1/12, 1/12\}$

- $$H(X) = -\frac{1}{3} \log_2 \frac{1}{3} - \frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{12} \log_2 \frac{1}{12} - \frac{1}{12} \log_2 \frac{1}{12} = 1.626 \text{ biți}$$

I) 12 piese: 3 cuburi, 3 conuri, 3 sfere, 3 piramide | Fiecare formă are una dintre culorile: roșu, galben, albastru

A extrage o piesă roșie. Câtă informație primește copilul B, care vrea să afle tipul piesei?

Inițial: 12 variante, dar 4 piese sunt de culoare roșie, deci  $p_i = 4 \cdot \frac{1}{12}$

$$I(x_i) = \log_2 \left( \frac{1}{\frac{4}{12}} \right) = \log_2(3) \quad (x_i \text{ este o piesă roșie})$$

$$I(\text{bila albastra}) = \log_2 \left( \frac{1}{\frac{3}{8}} \right) = \log_2 \left( \frac{8}{3} \right) = 1.42 \text{ biti}$$

II) În urnă: 5 bile roșii, 3 bile albastre

Se extrage o bilă albastră.

$$H(\text{urna}) = \frac{5}{8} \log_2 \left( \frac{8}{5} \right) + \frac{3}{8} \log_2 \left( \frac{8}{3} \right) = 0.95 \text{ biti}$$

Primim I(bilă albastră) cantitate de informație

$$H(\text{urna după extragere}) = \frac{5}{7} \log_2 \left( \frac{7}{5} \right) + \frac{2}{7} \log_2 \left( \frac{7}{2} \right) = 0.86 \text{ biti}$$

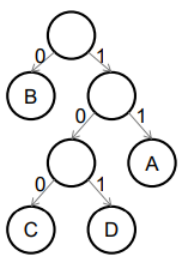
Entropia spune că: Nu se poate face o reprezentare pe biți mai optimă decât valoarea ei fără a pierde informație. (Ceva comprimat perfect se numește zgomot)

Este limita de compresie posibilă!

Codarea eficientă și unică se face cu arbori binari:

Frunzele – Codurile

Stânga/Dreapta – Decis de 0/1



Implementare: Algoritmul lui Huffman

(Optim când considerăm un singur caracter pe rând)

Input: Probabilitatea fiecărui eveniment, ex:  $\{1/3, 1/2, 1/12, 1/12\}$

Output: Codurile de pe un arbore binar

Cheia: O codare mai scurtă pentru event. cu probabilități mari, una mai scurtă --  
//-- mici.

Dacă toate evenimentele sunt echiprobabile, nu putem face nimic.

00100111010011    0 0 100 11 101 0 0 11  
B B C    A   D   B B A (Parcurgem arborele)

Detectarea erorilor: Ineficientă ca stocare.

Metoda I) Adăugăm biți de paritate simbolurilor, pentru a recupera informația.

Corecția erorilor: Distanță Hamming – O distanță mare poate duce și la corectarea erorilor

O distanță Hamming de  $2N+1$  poate corecta E erori. Distanța hamming minimă pentru 4 biți schimbați este 4.

**Orice comunicație / Stocare este redundantă!**

Aproximarea lui Stirling:  $\log_2(a!) \approx a \log_2(a) - a \log_2 e$



### Probabilitățile pentru fiecare caz:

(În principiu : Cât știm / Cât vrem să știm)

a) x are exact două valori "1" în reprezentarea sa binară:  $p = \frac{C_N^2}{2^N}$ , pp. n par.

b) x are exact N/2 valori "1" în reprezentarea sa binară (N par):  $p = \frac{C_N^{N/2}}{2^N}$ , pp. n par.

c) x are o secvență continuă de N/4 biți de "1" (restul sunt "0", N divizibil la 4) :

$$p = \frac{(3N/4+1)}{2^N}, \text{ pp. } 4 \mid n$$

d) x are MSB (Most Significant Bit) setat la "1":  $p = \frac{2^{N-1}}{2^N} = \frac{1}{2}$

e) x este impar:  $p = \frac{2^{N-1}}{2^N} = \frac{1}{2}$

f) x este o putere a lui 2:  $p = \frac{N}{2^N}$

g) x are primii N/2 biți din reprezentarea sa binară setați la "0" (N par):  $p = \frac{2^{N/2}}{2^N}$

h) x este un număr prim (aproximativ estimat):

$$p \approx \frac{[2^N / \ln(2^N)]}{2^N} = \frac{1}{\ln(2^N)}$$

i) x are un număr par de biți setați la "1" (N par):  $p = \frac{1 + \sum_{i=1}^{N/2} C_N^{2i}}{2^N} = \frac{1}{2}$

j) x = 42:  $p = \frac{1}{2^N}$

# ASSEMBLY x86

Utilizări: Securitate informatică: Hacking, Reverse Engineering | Optimizare: Dezvoltare jocuri și ML / AI | Debugging | Dezvoltare software low-level: Sisteme embedded / Sisteme de operare

În Assembly x86, rezultatul înmulțirii poate fi stocat pe 64 de biți ( $\%edx * 2^{32} + \%eax$ )

Shift:  $|S|XXX.XXXX \gg 2 = 00|S|X.XXXX$  – Se pierde bitul de semn!

SAR/SAL – Shift aritmetic, ține cont de semn.

jmp \*%eax – Sare la o adresă

aritate 1 – Operația se aplică unui singur număr

.asciz -> Lungimea șirului + 1 pentru \n

S1: "ASC", S2: "FMI" -> syscall de print pentru \$S1 cu lungimea 5 -> ASCFM

$\neg$  NOT (are aritate 1)       $\wedge$  AND       $\vee$  OR      Tsunotshi DX – eu irl | ORNOTAND

CMP – Compară două valori și setează flag-uri:

- Zero Flag (ZF)  $op1 = op2$
- Sign Flag (SF) Rezultatul comparației este negativ
- Carry Flag (CF)  $op1 < op2$

$$a(b, c, d) = a + b + c \cdot d$$

operand	Descriere	Semn (Da/Nu)	Flaguri setate	Condiție Flag
jc	jump dacă este carry setat	Nu	CF (Carry Flag)	CF = 1
jnc	jump dacă nu este carry setat	Nu	CF	CF = 0
jo	jump dacă este overflow setat	Nu	OF (Overflow Flag)	OF = 1
jno	jump dacă nu este overflow setat	Nu	OF	OF = 0
jz	jump dacă este zero setat	Nu	ZF (Zero Flag)	ZF = 1
jnz	jump dacă nu este zero setat	Nu	ZF	ZF = 0
js	jump dacă este sign setat	Da	SF (Sign Flag)	SF = 1
jns	jump dacă nu este sign setat	Da	SF	SF = 0
jb	jump if below (unsigned $op2 < op1$ )	Nu	CF	CF = 1
jbe	jump if below or equal (unsigned $op2 \leq op1$ )	Nu	CF, ZF	CF = 1 or ZF = 1
ja	jump if above (unsigned $op2 > op1$ )	Nu	CF, ZF	CF = 0 and ZF = 0
jae	jump if above or equal (unsigned $op2 \geq op1$ )	Nu	CF	CF = 0
jl	jump if less than (signed $op2 < op1$ )	Da	SF, OF	SF $\neq$ OF
jle	jump if less than or equal (signed $op2 \leq op1$ )	Da	SF, OF, ZF	SF $\neq$ OF or ZF = 1
jg	jump if greater than (signed $op2 > op1$ )	Da	SF, OF, ZF	SF = OF and ZF = 0
jge	jump if greater than or equal (signed $op2 \geq op1$ )	Da	SF, OF	SF = OF
je	jump if equal ( $op1 = op2$ )	Nu	ZF	ZF = 1
jne	jump if not equal ( $op1 \neq op2$ )	Nu	ZF	ZF = 0

TABELA 1. Operandi de salt condiționat, flagurile setate și condițiile lor

# CIRCUITE

## CIRCUIT DIGITAL COMBINAȚIONAL

La ieșire, este o combinație (funcție logică care combină) toate sau o parte a intrărilor

Eficiente în general. Problema majoră:

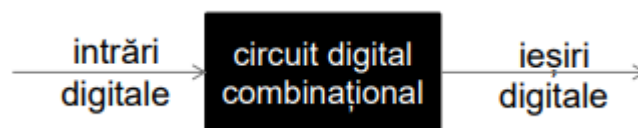
sunt one-shot

- Nu putem itera

- Nu permit niciun fel de logică internă / memorie internă (nu are stări interne)

- Sunt prea simple: Pui un semnal digital constant la intrare și ai un semnal digital constant la ieșire

- Logica combinațională este insuficientă pentru anumite implementări



Timp de propagare  $t_p$ : Timp maxim necesar pentru a produce la ieșire semnale digitale corecte și valide

Pentru fiecare intrare, trebuie să știm care e ieșirea

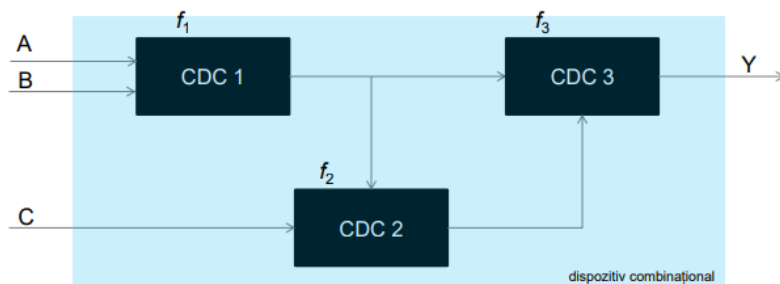
A	B	C	X	Y
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

Dispozitiv combinațional

Elementele sale: Circuite combinaționale

O intrare este conectată la exact o ieșire / la o constantă

Fără cicluri în graful direcțional al dispozitivului



Funcția dispozitivului:  $Y = f_3(f_1(A, B), f_2(f_1(A, B), C))$  – Ne uităm ce intră în Y șamd.

Timpul total de propagare:  $t_{p, total} = t_{p, 1} + t_{p, 2} + t_{p, 3}$  (cea mai lungă cale)  
*Timpul maxim după care avem o ieșire validă dacă avem intrări valide*

Un computer care funcționează la 1GHz trimite comenzi o dată la 1ns

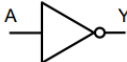
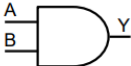


De ce folosim semnale digitale în loc de analogice:

- Într-un sistem analogic zgomotul se acumulează
- Într-un sistem digital, avem corecțiile de zgomot (avem margini)

De ce folosim sistemul binar?

- Număr redus de stări (2) (pentru HEX ar trebui 16, deci ar trebui distinse 16 nivele de voltaj)
- Pentru baza 4, am avea 4 nivele, deci am fi de 2 ori mai eficienți

**Tot ce facem pe sistemul de calcul trebuie redus la circuite care sunt porți logice**

Operația	Valori	Notație															
NOT	<div></div> <table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table> <td><math>\bar{A}</math> (A bar, A complement) sau !A</td>	A	Y	0	1	1	0	$\bar{A}$ (A bar, A complement) sau !A									
A	Y																
0	1																
1	0																
AND	<div></div> <table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> <td>AB</td>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1	AB
A	B	Y															
0	0	0															
0	1	0															
1	0	0															
1	1	1															
OR	<div></div> <table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> <td><math>A + B</math></td>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1	$A + B$
A	B	Y															
0	0	0															
0	1	1															
1	0	1															
1	1	1															
XOR	<div></div> <table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> <div><math>A \oplus B = \bar{A}B + A\bar{B}</math></div> <td><math>A \oplus B</math></td>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0	$A \oplus B$
A	B	Y															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

A

B

C

intrări  
digitale

circuit digital  
combinational

ieșiri  
digitale

X

Y

A	B	C	X	Y
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

o expresie booleană care conține regulile din tabel?

•

$X = \bar{A}BC + \bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$

poartă	întârziere (ps)	suprafață (μm²)
AND-2	50	25
NAND-2	30	15
OR-2	55	26
NOR-2	35	16
AND-4	90	40
NAND-4	70	30
OR-4	100	42
NOR-4	80	32

N\* - Mai rapide, mai mici ca suprafață față de \*

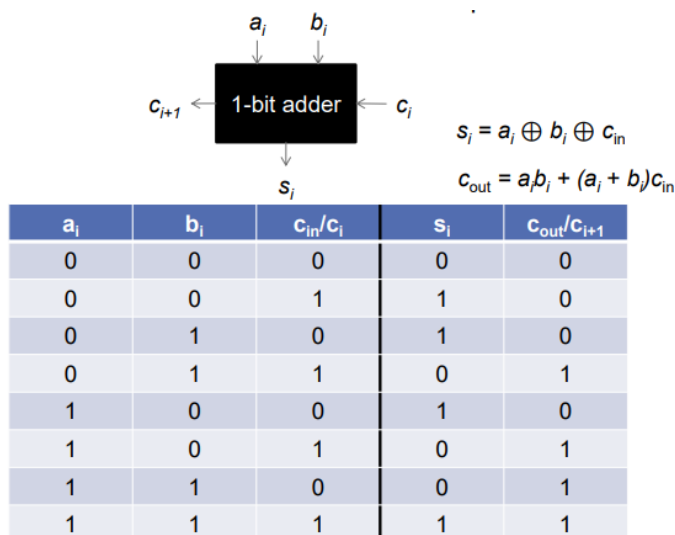
\*-2 - Mai rapide, mai mici ca suprafață față de \*-4

Tranzistoare CMOS: Circuitele analogice sunt mai eficiente pe logică negată (mai puține componente electrice)

CIRCUIT	INPUT	OUTPUT	NR INTRĂRI	NR IEȘIRI
ADUNARE	A (N biți) B (N biți)	S (N + 1 biți)	2N	N + 1

Cât de mare va fi circuitul?  $(N + 1)2^{2N-1}$

Putem defini un circuit bloc fundamental pe care bazăm totul:



Folosim circuitul în cascadă

$$t_p = N \cdot t_{p, \text{bit-adder}}$$

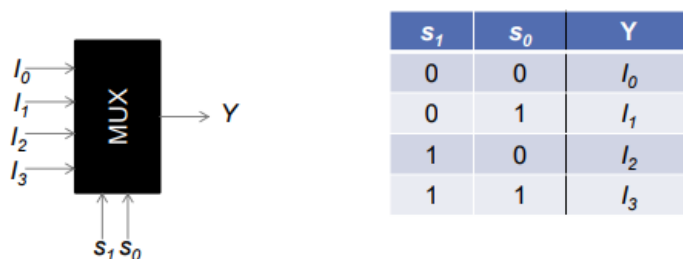
Trebuie așteptată calcularea biților de carry

Putem generaliza

- notăm  $g_i = a_i b_i$  și  $p_i = a_i + b_i$ 
  - dacă  $g_i = 1$  atunci  $c_{out} = 1$
  - dacă  $g_i = 0$  și  $p_i = 0$  atunci  $c_{out} = 0$
  - dacă  $g_i = 0$  și  $p_i = 1$  atunci  $c_{out} = c_{in}$

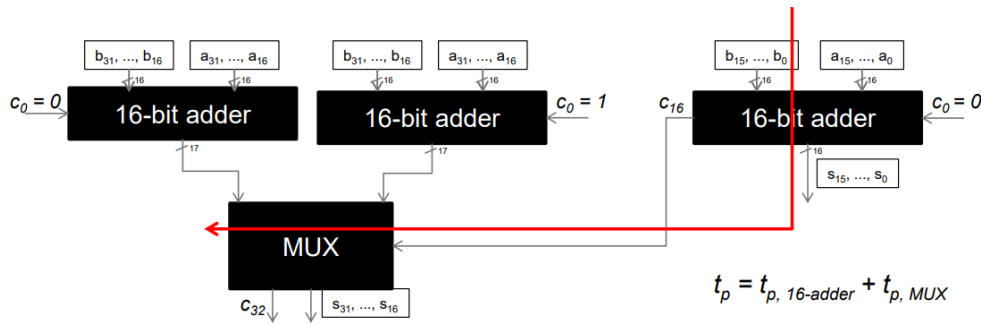
## Multiplexor (MUX ☺)

Circuitul selectează: un semnal digital de la intrare pe baza unui semnal de activare S  
Util pentru implementare hardware pentru "if", "case", operații shift



Inputuri:  $2^n \rightarrow n$  linii de selecție

## Îmbunătățire: Adunare pe 32 de biți



Putem aplica aceeași idee pentru circuitele de 16 biți de mai sus:  $t_p = O(\log_2 N)$

## CIRCUITE SECVENȚIALE

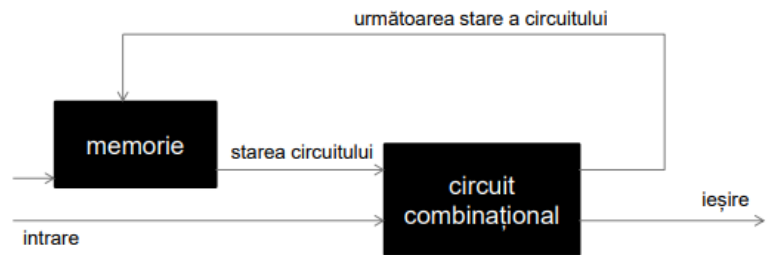
Permit elemente de tip "memorie" > putem adăuga o stare circuitului (există stare internă)

Au variabila de timp: Intrările / Ieșirile nu sunt fixe | Număr variabil de pași în rezolvare

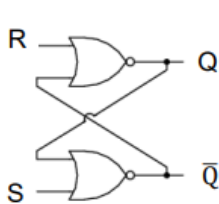
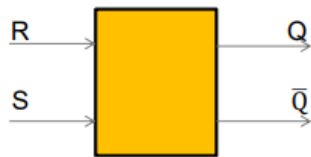
Biții pe care îi reprezentăm - voltaj.

Energia electrică - dificil de stocat (provoacă fenomenul de scurgere / leakage)

Pentru a memora ceva > Refresh din când în când pentru actualizarea nivelului de energie electrică



**SR Latch** (Set-Reset Latch) | Memorează un bit de informație | Bun, dar are două intrări



S	R	Q	$\bar{Q}$
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

nu se schimbă nimic

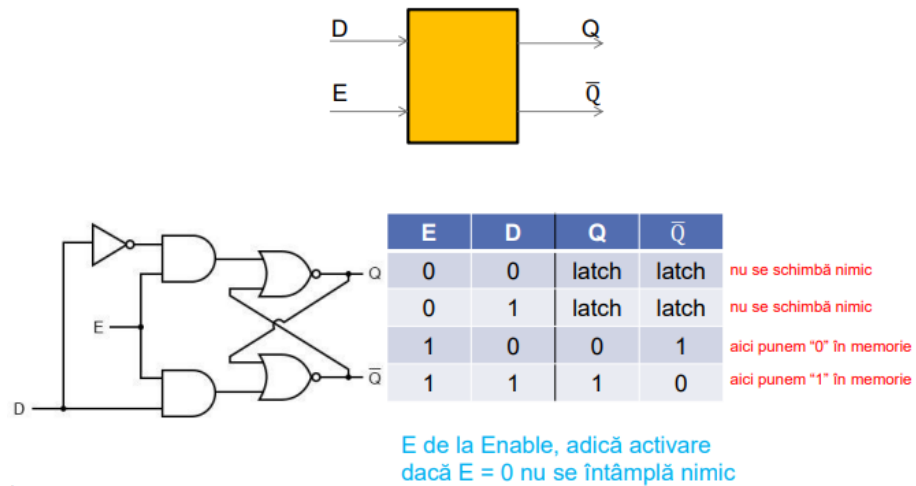
aici punem "0" în memorie

aici punem "1" în memorie

stare invalidă

**D Latch** | Bun, are o intrare dar vrem să sincronizăm mai multe dispozitive

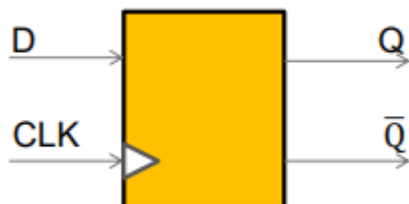
Se activează când E este activ. Vrem să se activeze când E crește.



## D Flip-Flop

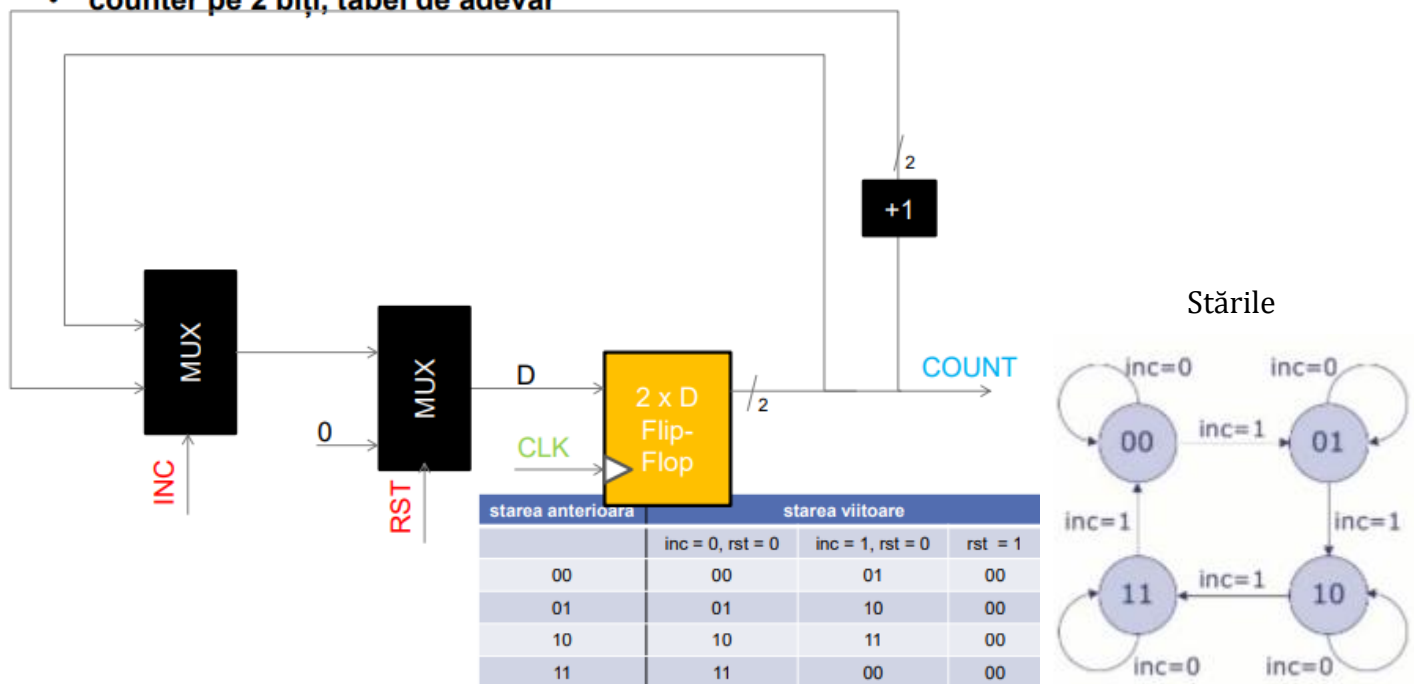
E devine clock (ceasul sistemului)

CLOCK: La un interval fix de timp (la un ciclu), sistemul face ceva



**Registru** = Un set de câteva D Flip Flops care au același CLK

- counter pe 2 biți, tabel de adevăr



## LEGILE DE MORGAN

- $\neg(\neg A + \neg B) = AB$
- $\neg(\neg A \neg B) = A + B$
- $\neg(A + B + C) = \neg A \neg B \neg C$
- $\neg(ABC) = \neg A + \neg B + \neg C$
- $\neg(A + B) \neg A \neg B = \neg A \neg B$
- $\neg(AB) (\neg A + \neg B) = \neg A + \neg B$
- $\neg(A + B) (\neg A + \neg B) = \neg A \neg B$
- $\neg A \neg B \neg(AB) = \neg A \neg B$
- $C + \neg(CB) = 1$
- $\neg(AB) (\neg A + B) (\neg B + \neg A) = \neg A \neg B$

## SIMPLIFICĂRI

$(A + C)(AD + A\bar{D}) + AC + C$   
 $(A + C)A(D + \bar{D}) + AC + C$  //distribuiem, invers  
 $(A + C)A + AC + C$  //suma variabila si complement  
 $A((A + C) + C) + C$  //distribuiem, invers  
 $A(A + C) + C$  //asociem, idempotent  
 $AA + AC + C$  //distribuiem  
 $A + (A + 1)C$  //idempotent, identitate, factor  
 $A + C$  //identitate de doua ori

$A + 0 = A$   
 $\neg A \times 0 = 0$   
 $A + \neg A = 1$   
 $A + A = A$   
 $A + AB = A$   
 $A + \neg AB = A + B$   
 $A(\neg A + B) = AB$   
 $AB + \neg AB = B$   
 $(\neg A \neg B + \neg AB) = \neg A$   
 $A(A + B + C + \dots) = A$

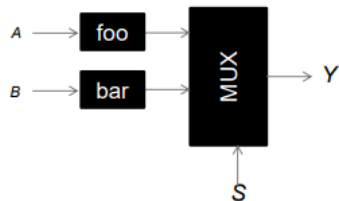
$AB + A \neg B = A$   
 $\neg A + B \neg A = \neg A$   
 $(D + \neg A + B + \neg C)B = B$   
 $(A + \neg B)(A + B) = A$   
 $C(C + \neg CD) = C$   
 $A(A + AB) = A$   
 $\neg(\neg A + \neg A) = A$   
 $\neg(A + \neg A) = 0$   
 $D + (D \neg CBA) = D$   
 $\neg D \neg(DBCA) = \neg D$   
 $AC + \neg AB + BC = AC + \neg AB$   
 $(A + C)(\neg A + B)(B + C) = AB + \neg AC$   
 $\neg A + \neg B + AB \neg C = \neg A + \neg B + \neg C$   
 $(A + B)^2 + (A + B)^3 + A + 3\neg A + A^3 = 1$

$A + A \neg A = A$



- implementați o poartă NOT cu un MUX:  $Y = \text{NOT } A$
- $Y = I_0 \bar{s}_0 + I_1 s_0 = 1\bar{A} + 0A = \bar{A}$
- implementați o poartă AND cu un MUX:  $Y = A \text{ AND } B$
- $Y = I_0 \bar{s}_0 + I_1 s_0 = 0\bar{A} + BA = AB$
- implementați o poartă OR cu un MUX:  $Y = A \text{ OR } B$
- $Y = I_0 \bar{s}_0 + I_1 s_0 = B\bar{A} + 1A = A + B\bar{A} = A + B$  [vezi ex. 4 f)]

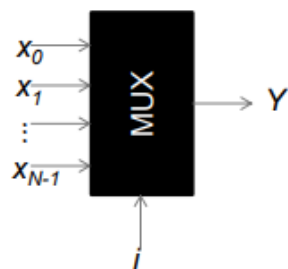
- $Y = S ? \text{foo}(A) : \text{bar}(B)$



$I_0$ $\text{foo}(X)$	$I_1$ $\text{bar}(Y)$	S	Y
*	*	0	$I_1$
*	*	1	$I_0$

- care e diferența cu un limbaj de programare?
  - indiferent de valoarea lui S, se execută  $\text{foo}(A)$  și  $\text{bar}(B)$
  - doar că la ieșire vedem doar una dintre funcții (cea selectată de S)

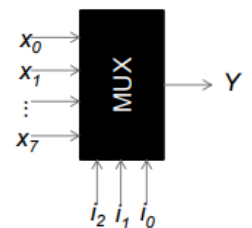
Vrem să accesăm  $x[i]$



Intrări: Vectorul x  
 Semnal: Indexul i  
 Dimensiunea intrării:  $N - 1 - 0 + 1 = N$   
 Dimensiunea lui S:  $\text{ceil}(\log_2 N)$

$s_0(i)$	Y
000	$I_0(x_0)$
001	$I_1(x_1)$
010	$I_2(x_2)$
011	$I_3(x_3)$
100	$I_4(x_4)$
101	$I_5(x_5)$
110	$I_6(x_6)$
111	$I_7(x_7)$

Câte MUX cu 2 intrări pentru a simula MUX cu N intrări?  
 $N - 1$



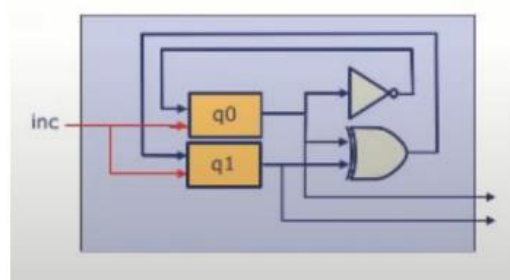
## Circuit secvențial COUNTER pe 2 biți

Relația dintre starea anterioară și starea viitoare

$$q_0^{(t+1)} = !INC \times q_0^{(t)} + INC \times !q_0^{(t)}, q_1^{(t+1)} = !INC \times q_1^{(t)} + INC \times (q_1^{(t)} \otimes q_0^{(t)})$$

$q_1^{(t)}$	$q_0^{(t)}$	$q_1^{(t+1)}$	$q_0^{(t+1)}$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Desenul circuitului secvențial (stările sunt q0 și q1)



- aici INC e pe post de semnal Enable

c) Codul Gray – cod binar, proprietate: diferența de la un simbol la altul este un singur bit care se schimbă (Pentru 3 biți, codul Gray: 000, 001, 011, 010, 110, 111, 101, 100).

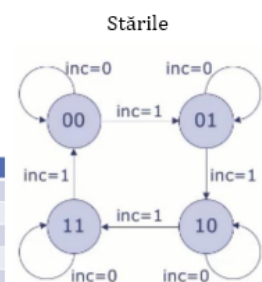
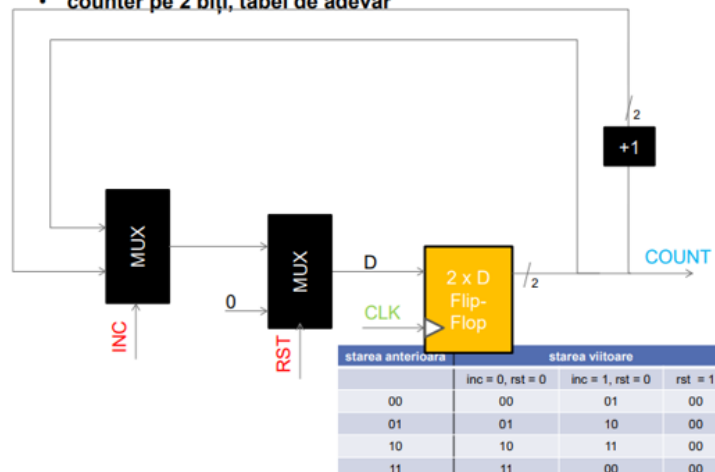
$q_2^{(t)}$	$q_1^{(t)}$	$q_0^{(t)}$	$q_2^{(t+1)}$	$q_1^{(t+1)}$	$q_0^{(t+1)}$
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	1

$$q_0^{(t+1)} = !q_2^{(t)}!q_1^{(t)}!q_0^{(t)} + !q_2^{(t)}!q_1^{(t)}q_0^{(t)} + q_2^{(t)}q_1^{(t)}!q_0^{(t)} + q_2^{(t)}q_1^{(t)}q_0^{(t)} \\ = q_2^{(t)}q_1^{(t)} + !q_2^{(t)}!q_1^{(t)}$$

$$q_1^{(t+1)} = \dots$$

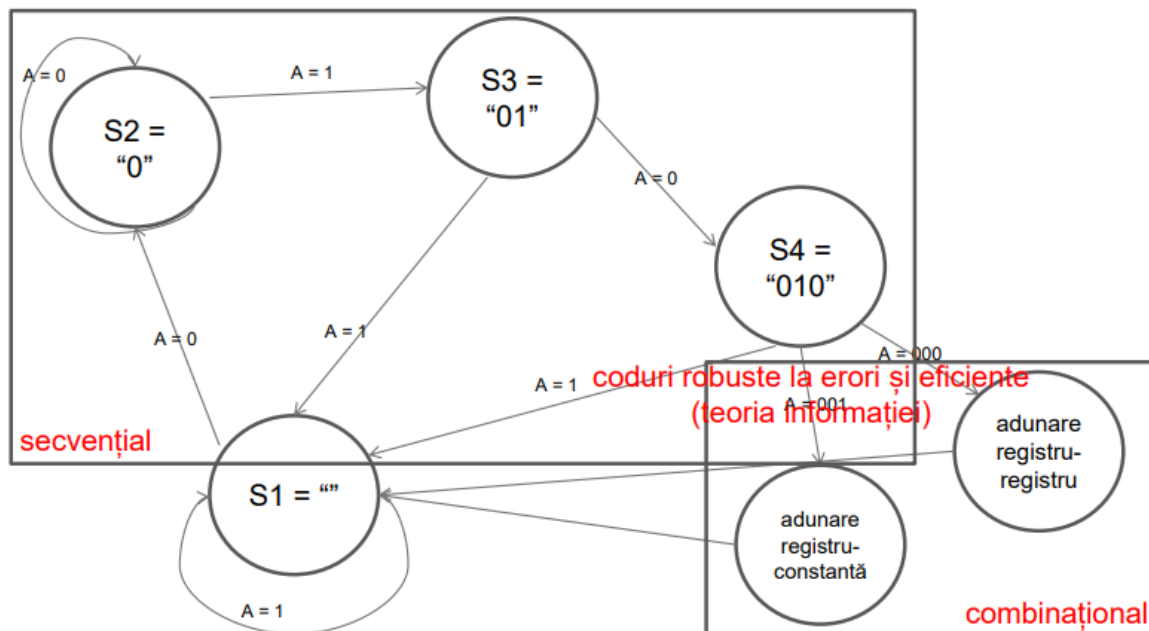
$$q_2^{(t+1)} = \dots$$

- counter pe 2 biți, tabel de adevăr



Un semnal digital A poate lua valori  $\{0, 1\}$  în timp iar noi vrem să detectăm dacă semnalul are valoarea 010 la un moment dat. Dacă această secvență de biți este detectată în A atunci o variabilă Y este setată la 1, altfel această variabilă este 0. Cerințe:

- desenați diagrama de stări și tranzițiile între aceste stări;
- cum arată circuitul secvențial asociat diagramei realizate anterior?
- calculați tabelul de stări pentru circuitul secvențial;
- scrieți expresiile pentru logica combinațională.



cod mașină ...0100001011010001010000100011001011 ...

## Circuit secvențial pentru CMMDC

def cmmdc(a, b): if a == b: return b elif a > b: return cmmdc(a-b, b) else: return cmmdc(b, a)

Avem variabilele:  $a^{(t)}$ ,  $b^{(t)}$

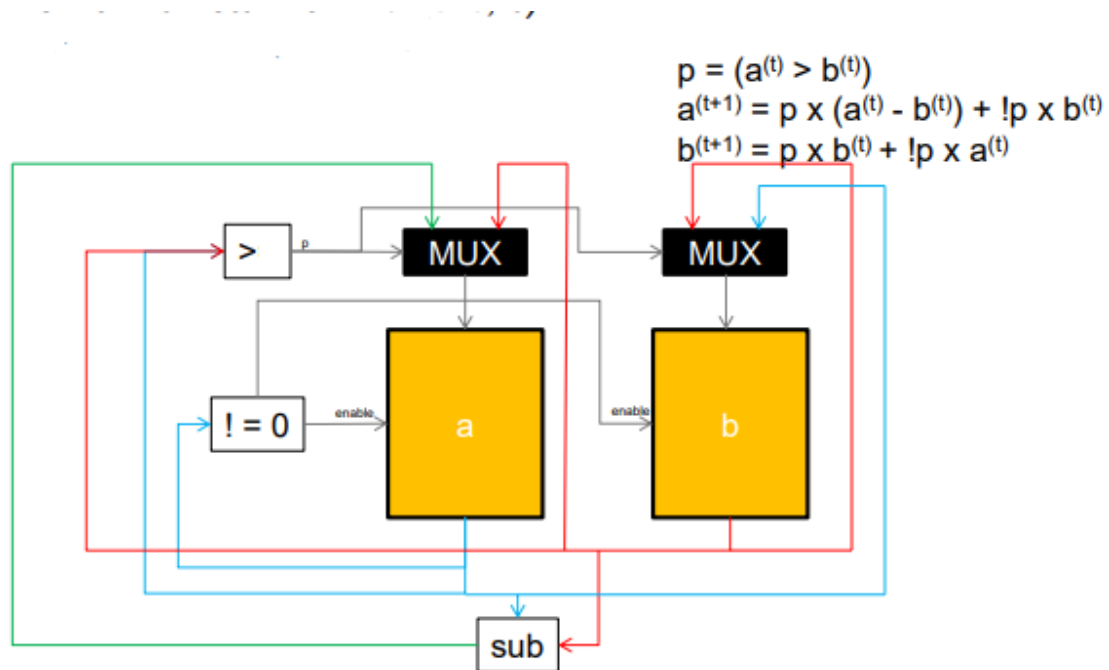
Ecuatiile de evoluție:

facem ceva doar dacă  $a^{(t)} \neq 0$

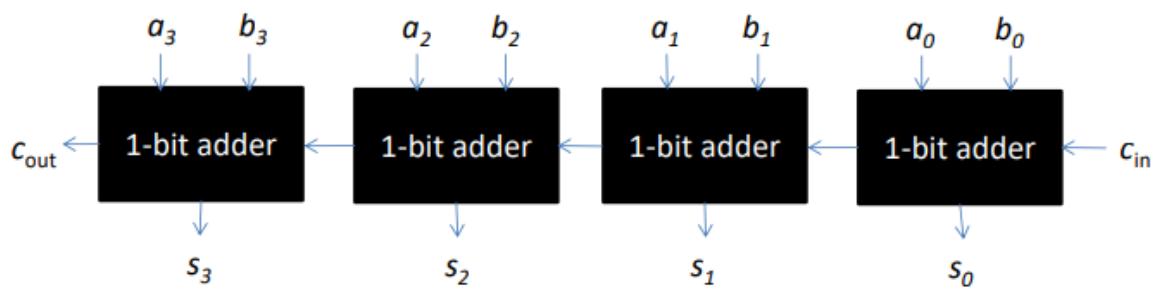
$p = (a^{(t)} > b^{(t)})$

$a^{(t+1)} = p \times (a^{(t)} - b^{(t)}) + !p \times b^{(t)}$

$b^{(t+1)} = p \times b^{(t)} + !p \times a^{(t)}$



## Circuit de adunare pe 4 biți



Circuit de adunare pe 4 biți.

# ÎMPĂRȚIREA NUMERELOR ÎNTREGI

Tratăm împărțitorul ca număr natural în baza 10, fie el  $n$  în exemplu:

```
int Divide(NrMare x, int n)
//x = x / n, returneaza x%n
{
    int i, r=0;
    for(i=x[0]; i>0; i--)
    {
        r=2*r+x[i];
        x[i]=r/n;
        r%=n;
    }
    for(; x[0]==0 && x[0]>1;)
        x[0]--;
    return r;
}
```

- $s = a \div b$

- ce se întâmplă dacă  $a$  sau  $b$  sunt variabile negative?
- rezultatul este negativ dacă  $a$  și  $b$  au semne diferite (XOR logic)
- în general
  - $a = s \times b + r$
  - semnul lui  $r$  este semnul lui  $a$

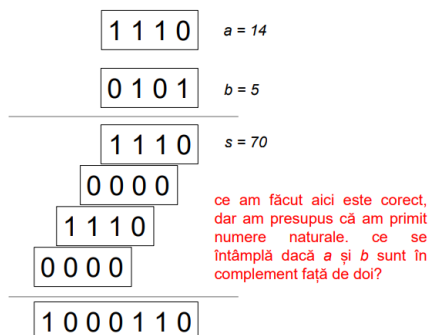
# ÎNMULȚIREA NUMERELOR ÎNTREGI

De la dreapta la stânga

Dacă suntem în  $B$  pe bit 1, copiem  $A$

Dacă suntem în  $B$  pe bit 0, punem 0 peste tot

## A și B naturale



## A și B întregi

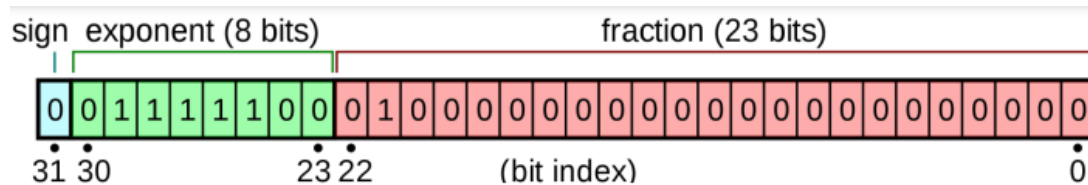
Extindem numerele:

$A = -2$        $A = 1110$        $A = 1111.1110$

$B = 5$        $B = 0101$        $B = 0000.0101$

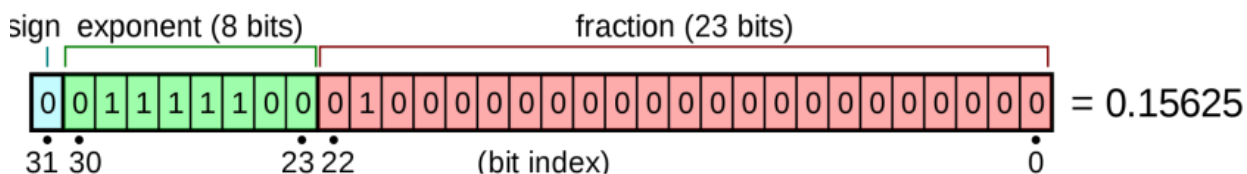
Se calculează la fel ca la numere naturale, dar **rezultatul este în complement față de doi**.

## FLOATING POINT



- -1313.3125

- partea întreagă este: 1313
- partea fracționară: 0.3125
  - $0.3125 \times 2 = 0.625 \Rightarrow 0$
  - $0.625 \times 2 = 1.25 \Rightarrow 1$
  - $0.25 \times 2 = 0.5 \Rightarrow 0$
  - $0.5 \times 2 = 1.0 \Rightarrow 1$
- deci,  $1313.3125_{10} = 10100100001.0101_2$
- normalizare:  $10100100001.0101_2 = 1.01001000010101_2 \times 2^{10}$
- mantisa este 010010000101010000000000
- exponentul este  $10 + 127 = 137 = 10001001_2$
- semnul este 1



[illegible]

Schimbati semnul lui a:  $a = a \wedge (1 \ll 31)$

Exponential pe: 0x7F80.0000

Extragem exponentul: (a & 0x7F80.0000) >> 23

**Pentru a împărți la 4**

**Exponent > 1 => Exponent = Exponent - 2 | Altfel: a = 0**

$$a = (a \& \sim \text{MASK}) \mid (\text{exponent} \ll 23)$$

SEMN	$a \gg 31$
EXPONENT	$(a \gg 23) \& 0x000000FF$
MANTISĂ	$a \& 0x007FFFFF$
Nr. în baza 10 cu mantisă de M biti	$ \log_{10} 2 \times M  \approx M/3$

abs(a)	a = a & ~(1 << 31)
a x 2	a << 1 sau a + a
a x 2	(a < 0) ? -((-a) << 1): a << 1
a x 16	a << 4
a x 3	a << 1 + a
a x 7	a << 3 - a
a / 8	a >> 3
a % 16	a & 0x000F
a % m	a & (m - 1)
a / 16	(a & FFF0) >> 4
a x 72	a << 6 + a << 3
a / 16 + a % 16	a & FFF0 + a & 000F

d) 0xDEADBEEF = 0b11011110101011011011111011101111

- S = 1
- E = 10111101
- M = 0101101101111101110111
- $(-1)^S 1.M \cdot 2^{E-127} = (-1) 1.01011011011111011101111 2^{189-127}$   
 $= -1.01011011011111011101111 2^{62}$   
 $= -6259853398707798000$

- setați s = 0, e = 0, f = 0
- $a = (-1)^0 \times 1.00...00 \times 2^{-127} = 2^{-127} \neq 0$

- 0.2 + 0.3

- primul pas, trecem fiecare număr în format

- $0.2 = + 1.10011001100110011001100 \times 2^{-3} = 0.19999998807907104$
- $0.3 = + 1.00110011001100110011001 \times 2^{-2} = 0.29999998211860657$

- al doilea pas, alinierea

- $0.2 = + 0.11001100110011001100110|000 \times 2^{-2}$
- $0.3 = + 1.00110011001100110011001|000 \times 2^{-2}$

- al treilea pas, adunăm

- $0.2 + 0.3 = 1.1111111111111111111111|000 \times 2^{-2}$

- **a / 19**

$$a \times \frac{1}{19} \approx \frac{a \times \frac{2938661835}{2^{32}} + \frac{a - a \times \frac{2938661835}{2^{32}}}{2^1}}{2^4}$$

$$a \times \frac{1}{19} \approx (a \times 2938661835 \times 2^{-32} + (a - a \times 2938661835 \times 2^{-32}) \times 2^{-1}) \times 2^{-4}$$

$$a \times \frac{1}{19} \approx a \times \frac{7233629131}{137438953472}$$

- **soluția generală**

$$\frac{a}{D} \approx \frac{\frac{aC}{2^X} + \frac{a - \frac{aC}{2^X}}{2^Y}}{2^Z}$$

$$D \approx \frac{2^{X+Y+Z}}{C \times (2^Y - 1) + 2^X}$$

- **calculați aproximarea binară**
  - soluția: 0.099999904632568359375
- **care este diferența dintre valoarea calculată și 0.1**
  - soluția: 0.1 - 0.099999904632568359375
- **care este eroarea (de timp) după 100 de ore de operare**
  - soluția: 100x60x60x10x(0.1 - 0.099999904632568359375) ≈ 0.34
- **care este eroarea dacă reprezentăm 0.1 în formatul IEEE 754 FP?**
  - soluția: 100x60x60x10x(0.1 - 0.09999999403953552) ≈ 0.021
- **dacă rachetele SCUD pot atinge o viteză MACH 5, care este distanța pe care racheta o poate parcurge în timpul eroare calculat?**
  - soluția: 1715 m/s \* 0.34 s ≈ 583 m, 1715 m/s \* 0.021 s ≈ 36 m

## FAST INVERSE SQUARE ROOT, Q III

```

552 float Q_rsqrt( float number )
553 {
554     long i;
555     float x2, y;
556     const float threehalfs = 1.5F;
557
558     x2 = number * 0.5F;
559     y = number;
560     i = * ( long * ) &y; // evil floating point bit level hacking
561     i = 0x5f3759df - ( i >> 1 ); // what the duck?
562     y = * ( float * ) &i;
563     y = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
564     // y = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be removed

```

- **prima instrucțiune interesantă e pe linia 560**



- **dacă avem în M mantisa și exponentul în E atunci**

- numărul nostru în FP este  $2^{23} \times E + M$
- iar valoarea numărului este  $(1 + M / 2^{23}) \times 2^{E-127}$
- observăm că:  $\log_2 \left( \left(1 + \frac{M}{2^{23}}\right) \times 2^{E-127} \right) = \log_2 \left(1 + \frac{M}{2^{23}}\right) + \log_2 (2^{E-127})$   
 $= \log_2 \left(1 + \frac{M}{2^{23}}\right) + E - 127$   
 $\approx \frac{M}{2^{23}} + E - 127 + \mu$  (am folosit  $\log_2(1+x) \approx x, \mu$  este o corectie)  
 $= \frac{1}{2^{23}} (2^{23} \times E + M) + \mu - 127$   
 $= \frac{1}{2^{23}} (\text{reprezentarea pe biti}) + \mu - 127$

- **de ce calculăm  $\log(y)$ ? de fapt vrem  $1/\sqrt{y}$ . dar:**

$$\log_2 \left( \frac{1}{\sqrt{y}} \right) = \log_2 (y^{-1/2}) = -\frac{1}{2} \log_2(y) = -(i \gg 1)$$

- $\pi \approx 3.14159265 = (-1)^0 1.10010010000111111011010 2^{b10000000-127}$
- $+0 = (-1)^0 1.000000000000000000000000 2^{00000000-127}$
- $-0 = (-1)^1 1.000000000000000000000000 2^{00000000-127}$
- signaling NaN: 0x7F800001 sau 0x7FBFFFFFF sau între 0xFF800001 și 0xFFBFFFFFF
- quiet NaN: 0x7FC00000 sau 0x7FFFFFFF sau între 0xFFC00000 și 0xFFFFFFF

## CONSECINȚE FLOATING POINT

- $(0.1 + 0.2) == 0.3$  versus  $(0.2 + 0.3) == 0.5$  (rotunjiri)
- $\text{math.sqrt}(3) * \text{math.sqrt}(3) == 3$  versus  $\text{math.sqrt}(3*3) == 3$
- $(0.7 + 0.2) + 0.1$  versus  $(0.7 + 0.1) + 0.2$  (nu avem asociativitatea)
- diferența cu numere întregi
  - dacă folosim tip de date întreg:  $16777216 + 1 = 16777217$
  - dacă folosim tip de date FP:  $16777216.0 + 1 = 16777216.0$
  - $\text{float}(123456789101112) + 1.0 = 123456789101113.0$
  - $\text{float}(1234567891011121) + 1.0 = 1234567891011122.0$
  - $\text{float}(12345678910111213) + 1.0 = 1.2345678910111212\text{e}+16$

# ARHITECTURA CALCULATOARELOR MODERNE

Pornirea sistemului

- Buton de power ON/OFF > Realizează alimentarea cu electricitate a componentelor
- CPU este activat > Caută pornește BIOS
  - Testează componentele HW (RAM, I/O, HDD, etc.)
  - Încarcă BIOS (scris pe placa de bază) din ROM (read only memory) în RAM pentru execuție

BIOS: Știe cât e ceasul (CMOS Real-Time Clock) și HW, se accesează cu F2 la pornirea sistemului

CPU/BIOS: Pornesc Boot Code (caută sistemul de operare)

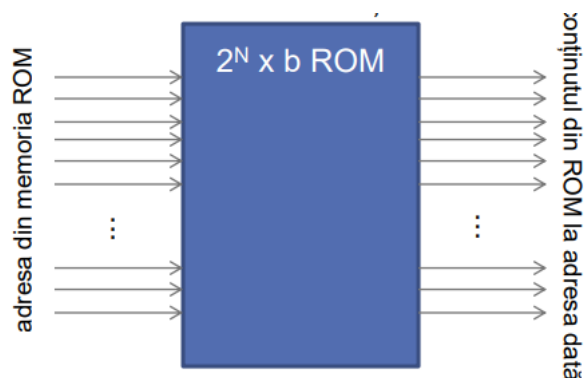
- În general, SO este pe HD (poate fi și pe CD, stick). Se încarcă în RAM pentru execuție

Componente

TOT BIOS: Scris în ROM, câteodată în Programmable ROM, Erasble Programmable ROM

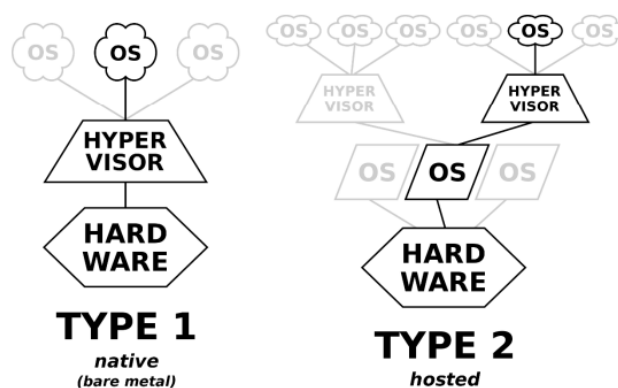
„Să scriem în ROM”: „burning” sau „flashing” the ROM

Este un circuit combinațional



OS preia controlul de la BIOS

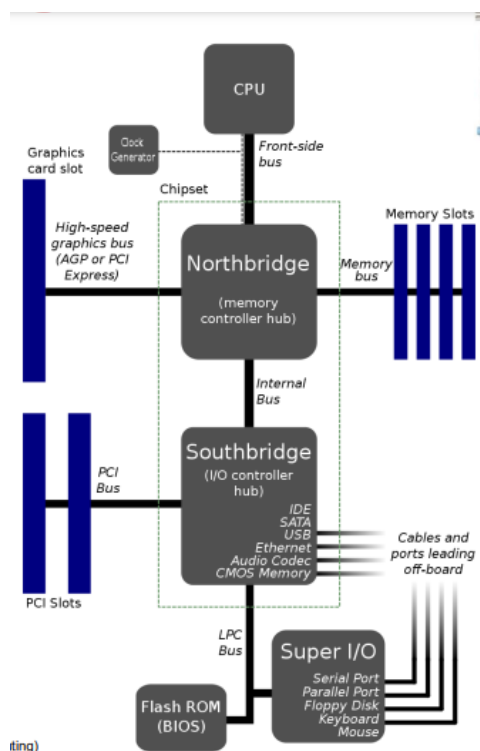
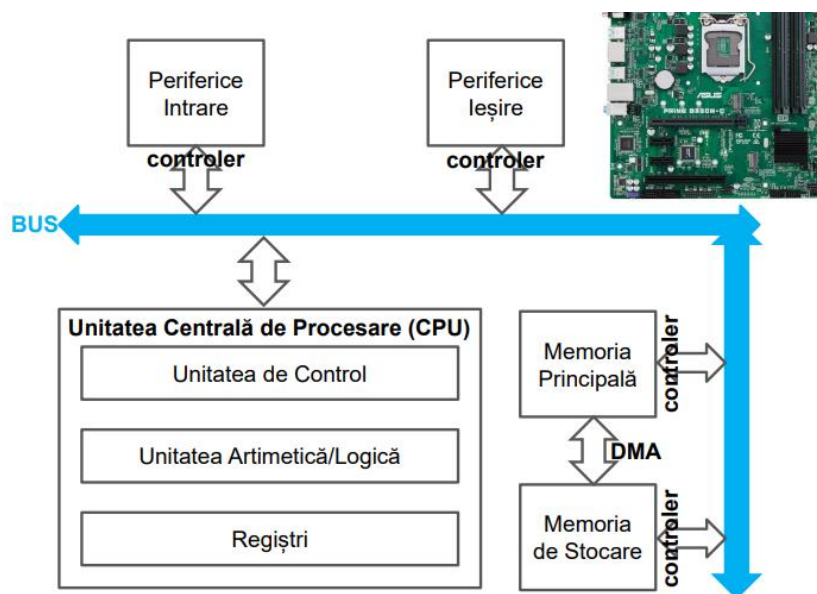
- Doar OS are acces la periferice, prin drivere
- Virtualizare/Emulare/Containere(Dockere)



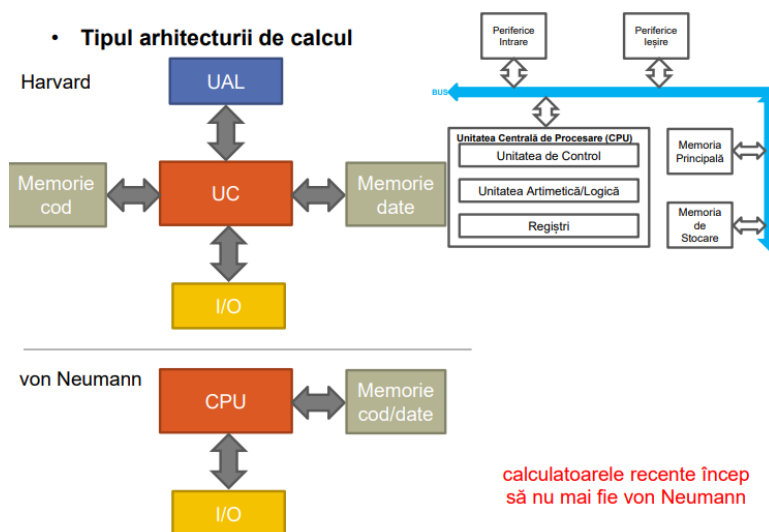
OS: Oferă imagine abstractizată a memoriei pentru fiecare proces pornit.  
 OS pornește > Sistemul de calcul intră în ciclul obișnuit de procesare  
 (Secvența boot se încheie)

Un sistem de calcul trebuie să

- Calculeze > Să execute instrucțiuni
- Să comunice > Să transfere biți între componente electronice
- Să stocheze > Date - folosite de instrucțiuni | Instrucțiuni pentru execuție



#### • Tipul arhitecturii de calcul



calculatoarele recente încep  
 să nu mai fie von Neumann

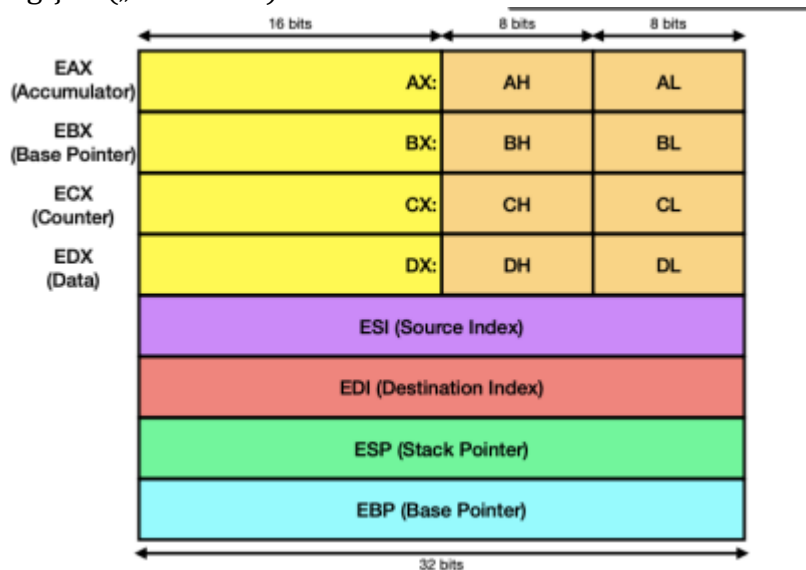
CPU (5 componente) – Creierul unității de calcul / Execută instrucțiuni

1. Clock

- Circuit special care generează „ceasul”
- Frecvența la care operează CPU (calculare + sincronizarea componentelor secvențiale)

- Frecvență mare -> Mai bine | Se măsoară în MHz sau GHz

2. Regiștri („memoria”)



3. UAL („operații”) – Unitatea aritmetică logică

- Operații logice
- Operații aritmetice cu int/float
- Operații speciale: sqrt, exp, trig

4. BUS

