- Seminar 0x00

7. Am discutat la curs de modul în care aflăm valoarea unui număr scris în format complement față de 2: inversăm biții și adunăm 1. Demonstrați că această procedură este corectă.

Let's examine two useful shortcuts when working with two's complement numbers. The first shortcut is a quick way to negate a two's complement binary number. Simply invert every 0 to 1 and every 1 to 0, then add one to the result. This shortcut is based on the observation that the sum of a number and its inverted representation must be $111\ldots111_{\text{two}}$, which represents -1. Since $x + \bar{x} = -1$, therefore $x + \bar{x} + 1 = 0$ or $\bar{x} + 1 = -x$. (We use the notation $\bar{x}$ to mean invert every bit in x from 0 to 1 and vice versa.)

care este cantitatea de informație din pachet când scoatem cărțile pentru prima dată?

este 0, cărțile sunt ordonate crescător

amestecăm aleator cărțile, câtă informație avem acum?

- în câte feluri putem combina cele 52 de cărți?
- 52!
- deci informația este $\log_2(52!)$
  - cum calculăm valoarea asta?
  - $\log_2(a \times b) = \log_2(a) + \log_2(b)$
  - $\log_2(52!) = 225.6$ biți
  - cu aproximarea lui Stirling: $\log_2(52!) \approx 52\log_2(52) - 52\log_2 e = 221.4$ biți
- algoritmic, cum amestecăm cărțile (eficient)?
  - aveți la dispoziție o funcție care returnează o valoare aleatoare în intervalul [0,1]

- Seminar 0x01

5. Se dă un număr natural x pe N biți. Câtă informație am câștigat dacă:

   (a) ni se spune despre x că are exact două valori "1" în reprezentarea sa binară;

   (b) ni se spune despre x că are exact $N/2$ valori "1" în reprezentarea sa binară;

   (c) ni se spune despre x că are o secvență continuă de $N/4$ biți de "1" în reprezentarea sa binară (restul biților sunt "0");

   (d) ni se spune despre x că are MSB setat la "1";

   (e) ni se spune despre x că este impar;

   (f) ni se spune despre x că este o putere a lui 2;

   (g) ni se spune despre x că are primii $N/2$ biți din reprezentarea sa binară setați la "0";

   (h) ni se spune despre x că este un număr prim (aici doar o estimare aproximativă este posibilă);

   (i) ni se spune despre x că are în reprezentarea sa binară un număr par de biți setați la "1";

   (j) ni se spune că $x = 42$.

a) $p = C_2^N / 2^N$

b) $p = C_{N/2}^N / 2^N$, **presupunem N par**

c) $p = (¾N + 1) / 2^N$, **presupunem N divizibil la 4**

d) $p = 2^{N-1} / 2^N = 1 / 2$

e) $p = 2^{N-1} / 2^N = 1 / 2$

f) $p = N / 2^N$

g) $p = 2^{N/2} / 2^N = 1 / 2^{N/2}$, **presupunem N par**

h) $p \approx [(2^N - 1) / \ln(2^N - 1)] / 2^N \approx [2^N / \ln(2^N)] / 2^N = 1 / \ln(2^N)$

i) $p = (1 + \sum_{i=1}^{N/2} C_{2i}^N) / 2^N = 1 / 2$, **presupunem N par**

j) $p = 1 / 2^N$

- Seminar 0x02

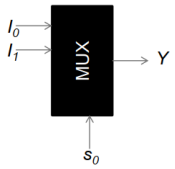
DE MORGAN, EX 3

- !(!A+!B) = AB
- !(!A!B) = A+B
- !(A+B+C) = !A!B!C
- !(ABC) = !A+!B+!C
- !(A+B)!A!B = !A!B
- !(AB)(!A+!B) = !A+!B
- !(A+B)(!A+!B) = !A!B
- !A!B!(AB) = !A!B
- C+!(CB) = 1
- !(AB)(!A+B)(!B+!B) = !A!B

SIMPLIFICĂRI, EX 4

a) A+0 = A
b) !Ax0 = 0
c) A+!A = 1
d) A+A = A
e) A+AB = A
f) A+!AB = A+B
g) A(!A+B) = AB
h) AB+!AB = B
i) (!A!B+!AB) = !A
j) A(A+B+C+...) = A
k) subpuncte
   a) A+B
   b) 1
   c) 1
l) A+A!A = A
m) AB+A!B = A
n) !A+B!A = !A
o) (D+!A+B+!C)B = B
p) (A+!B)(A+B) = A
q) C(C+CD) = C
r) A(A+AB) = A
s) !(!A+!A) = A
t) !(A+!A) = 0
u) D+(D!CBA) = D
v) !D!(DBCA) = !D
w) AC+!AB+BC = AC+!AB
x) (A+C)(!A+B)(B+C) = AB+!AC
y) !A+!B+AB!C = !A+!B+!C
$(A+B)^2+(A+B)^3+A+3!A+A^3 = 1$

MUX, două intrări, un semnal s de selecție și o ieșire

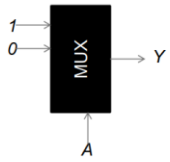


I_0	I_1	s_0	Y
*	*	0	I_0
*	*	1	I_1

care este relația ieșire-intrare?

- $Y = I_0\bar{s}_0 + I_1 s_0$

MUX, două intrări, un semnal s de selecție și o ieșire

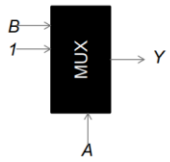


I_0	I_1	$s_0(A)$	Y
1	0	0	I_0
1	0	1	I_1

un MUX este un circuit universal, adică poate implementa porți NOT, OR și AND

- implementați o poartă NOT cu un MUX: Y = NOT A
- $Y = I_0\bar{s}_0 + I_1 s_0 = 1\bar{A} + 0A = \bar{A}$

MUX, două intrări, un semnal s de selecție și o ieșire

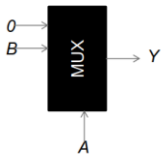


$I_0(B)$	I_1	$s_0(A)$	Y
B	1	0	I_0 (B)
B	1	1	I_1 (1)

un MUX este un circuit universal, adică poate implementa porți NOT, OR și AND

- implementați o poartă OR cu un MUX: Y = A 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)]

MUX, două intrări, un semnal s de selecție și o ieșire

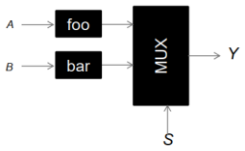


I_0	$I_1(B)$	$s_0(A)$	Y
0	B	0	I_0 (0)
0	B	1	I_1 (B)

un MUX este un circuit universal, adică poate implementa porți NOT, OR și AND

- implementați o poartă AND cu un MUX: Y = A AND B
- $Y = I_0\bar{s}_0 + I_1 s_0 = 0\bar{A} + BA = AB$

- **Y = S ? foo(A) : bar(B)**



I_0 foo(X)	I_1 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 foo(A) și bar(B)
  - doar că la ieșire vedem doar una dintre funcții (cea selectată de S)

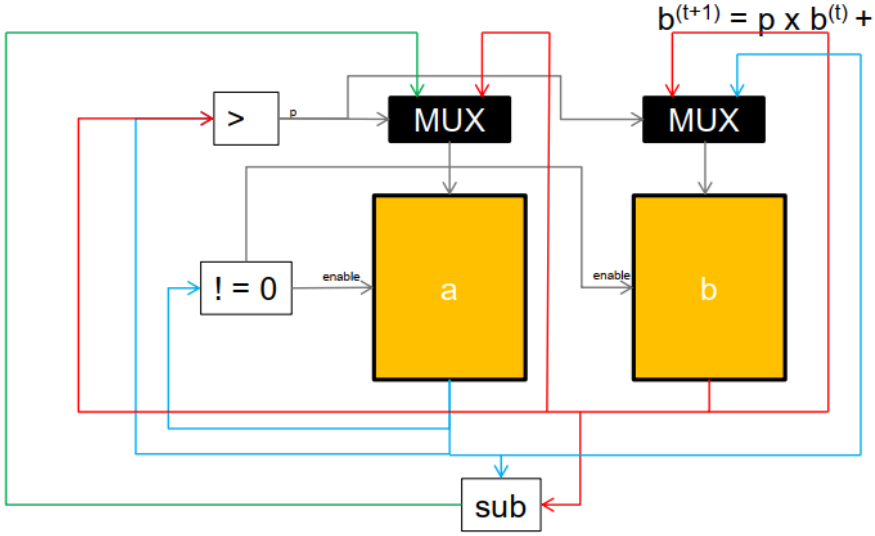
- **codul python:**

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

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

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

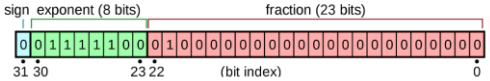
$b^{(t+1)} = p \times b^{(t)} + !p \times a^{(t)}$



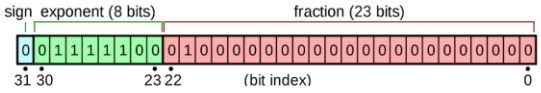
- Seminar 0x03

- **a x 2**
  - soluția: a << 1, sau a + a
- **a x 16**
  - soluția: a << 4
- **a x 3**
  - soluția: a << 1 + a
- **a x 7**
  - soluția: a << 3 – a
- **a / 8**
  - soluția: a >> 3
- **a mod 16**
  - soluția: a & 0x000F
- **a x 72**
  - soluția: a << 6 + a << 3
- **a mod 16**
  - soluția: a & 0x000F
- **a div 16**
  - soluția: (a & FFF0) >> 4, sau doar a >> 4
  - de asemenea: a = a & FFF0 + a & 000F = div cu 16 + mod cu 16
- **a x 2**
  - soluția: a = (a < 0) ? -((-a) << 1)) : a << 1 , pentru a numar intreg

CONVERSIA ÎN IEEE FP, EX. 7



- -1313.3125
  - partea întreagă este: 1313
  - partea fracționară: 0.3125
    - 0.3125 × 2 = 0.625 => 0
    - 0.625 × 2 = 1.25 => 1
    - 0.25 × 2 = 0.5 => 0
    - 0.5 × 2 = 1.0 => 1
  - deci, $1313.3125_{10} = 10100100001.0101_2$
  - normalizare: $10100100001.0101_2 = 1.01001000010101_2 \times 2^{10}$
  - mantisa este 01001000010101000000000
  - exponentul este 10 + 127 = 137 = 10001001_2
  - semnul este 1



- **împărțiți a la 4**
  - **soluția:**
    - **vrem exponentul, unde se află?**
      - **MASK = 0x7F800000**
    - **extragem exponent = (a & MASK) >> 23**
    - **dacă exponent > 1 atunci exponent = exponent – 2, altfel a = 0**
    - **trebuie să actualizăm a**
      - **a = (a & ~MASK) | (exponent << 23)**

- **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**
  - $0.2 = +\ 0.11001100110011001100110l000 \times 2^{-2}$
  - $0.3 = +\ 1.00110011001100110011001l000 \times 2^{-2}$

- **al treilea pas, adunăm**
  - $0.2 + 0.3 = 1.11111111111111111111111l000 \times 2^{-2}$

- **al patrulea pas, normalizare (dacă e necesar)**
  - $0.2 + 0.3 = 1.11111111111111111111111l00 \times 2^{-2}$

- **pasul final: $+ 1.11111111111111111111111 \times 2^{-2} = 0.4999999701976776$**

0x44361000 = 0b01000100001101100001000000000000

- S = 0
- E = 10001000
- M = 01101100001000000000000
- $(-1)^S\ 1.M * 2^{E-127} = (1)\ 1.01101100001000000000000\ 2^{136-127}$
  $= 1.01101100001000000000000\ 2^{9}$
  $= 728.25$

ÎMPĂRȚIREA RAPIDĂ, EX. 12

- **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**

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

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

- Seminar 0x04

- **un algoritm simplu de toUpper()**

```
void toUpper(char *buff, int count) {
    for (int i = 0; i < count; ++i)
    {
        if (buff[i] >= 'a' && buff[i] <= 'z')
            buff[i] -= 32;
    }
}
```

- **branchless?**

```
void toUpper(char *buff, int count) {
    for (int i = 0; i < count; ++i)
    {
      buff[i] -= 32*(buff[i] >= 'a' && buff[i] <= 'z');
    }
}
```

- Seminar 0x05

a) timpul total de acces memorie este (din curs)

1 ns + (0.1 x (5 ns + (0.01 x (10 ns + (0.002 x 50 ns)))))

în cazul nostru

1 ns + (A x (5 ns + (0.01 x (10 ns + (0.002 x 50 ns))))) = t_{RAM} / 2

b) 1 ns + (0.1 x (A ns + (0.01 x (10 ns + (0.002 x 50 ns))))) = t_{RAM} / 10

c) 1 ns + (0.1 x (5 ns + (0.01 x (10 ns + (A x 50 ns))))) = t_{RAM}

d) pentru L1, să trecem de la 1ns la 0.9ns ne costă 100$

pentru L2, să trecem de la 5ns la 4.5ns ne costă 25$

pentru L3, să trecem de la 10ns la 9ns ne costă 5$

$1x0.9^A + (0.1 \times (5x0.9^B + (0.01 \times (10x0.9^C + (0.002 \times 50))))) = t_{RAM}/1000$

vrem: minimize 100 x A + 25 x B + 5 x C

rezolvați pentru A, B și C

Test	Program A	Program B	A/B	B/A
1	9	3	3	0.33
2	8	2	4	0.25
3	2	20	0.1	10
4	10	2	5	0.2
Media	(a) 7.25	(a) 6.75	(g) 1.57	(g) 0.64

Test	Program A	Program B	A/B	B/A
1	9	3	3	0.33
2	8	2	4	0.25
3	2	20	0.1	10
4	10	2	5	0.2
Media	(a) 7.25	(a) 6.75	(g) 1.57	(g) 0.64

Concluzia: Program B este de 3 ori mai rapid decât Program A
Concluzia: Program A este de 2.7 ori mai rapid decât Program B

Nu luați media aritmetică a rapoartelor A/B sau B/A
Luați media geometrică a rapoartelor A/B sau B/A
- în acest caz, media rapoartelor este raportul medilor

Vrem să comparăm Program A vs. Program B: cine este mai rapid? A sau B?
- rulăm programele de mai multe ori
- comparăm linie cu linie în tabelul de mai sus
- pentru fiecare linie decidem cine câștigă (A sau B)
- apoi calculăm: care este probabilitatea ca A să fie mai rapid decât B dacă am observat că în n cazuri (din totalul de N) A este mai rapid decât B
- p-value

4. Se dau două numere complexe $x = a + bi$ și $y = c + di$. Răspundeți la următoarele întrebări:
   (a) scrieți explicit formula pentru $z = x \times y$;
   (b) câte adunări și înmulțiri se realizează pentru calculul lui z?
   (c) puteți să calculați z cu mai puține înmulțiri?
   (d) de câte ori ar trebui să fie mai lentă o înmulțire față de a adunare pentru ca rezultatul de la punctul precedent să fie eficient?
   (e) ideea de a înlocui o înmulțire cu mai multe adunări (în general, o operație dificilă cu o serie de operații simple) apare de mai multe ori în algoritmică (vedeți algoritmul lui Strassen).

a) z = (a+bi)x(c+di) = ac – bd + i(ad + bc)
b) 2 adunări, 4 înmulțiri
c) calculăm S1 = ac, S2 = bd și S3 = (a+b)x(c+d)
z = S1-S2 + i(S3-S1-S2)
5 adunări, 3 înmulțiri
d) C1 – costul unei adunări
C2 – costul unei înmulțiri
2C1 + 4C2 > 5C1 + 3C2
C2/C1 > 3

6. Există multe feluri de a calcula îmbunătățirea de performanță care este posibilă într-un sistem multi-core. Considerăm că avem un program care are $0 \leq p \leq 1$ dintre instrucțiuni (interpretat ca procentaj) paralelizabile. Considerăm că avem la dispoziție $s \geq 1$ procesoare sau că sistemul beneficiază de o accelerare a performanței de δ ori. Două modalități de a calcula accelerarea execuției, legea lui Amdahl și legea lui Gustafson, respectiv:

$$S_{\text{Amdahl}} = \frac{1}{(1-p) + \frac{p}{s}} \quad \text{și} \quad S_{\text{Gustafson}} = 1 - p + \delta p. \tag{1}$$

Cerințe:
   (a) calculați S_{Amdahl} și $S_{\text{Gustafson}}$ pentru $p = 0.5$, $s = 8$ și $\delta = 8$;
   (b) încercați să deduceți voi legile lui Amdahl și Gustafson;
   (c) ce se întâmplă cu S_{Amdahl} și $S_{\text{Gustafson}}$ dacă $p = 0$? Care este interpretarea rezultatului?
   (d) calculați S_{Amdahl} și $S_{\text{Gustafson}}$ dacă $s = 1$ și $\delta = 1$? Care este interpretarea rezultatului?
   (e) calculați $L_{\text{Amdahl}} = \lim_{s \to \infty} S_{\text{Amdahl}}$ și verificați dacă S_{Amdahl} este mereu sub sau peste această limită. Ce interpretare are valoarea L_{Amdahl}?
   (f) calculați $L_{\text{Gustafson}} = \lim_{\delta \to \infty} S_{\text{Gustafson}}$;
   (g) explicați diferențele dintre legile lui Amdahl și Gustafson.

a) S_{Amdahl} = 1,78 și $S_{\text{Gustafson}}$ = 4,5
b) pentru Amdahl
dacă un program are timpul de execuție T atunci T = (1-p)T + pT (facem distincția între partea paralelizabilă și cea secvențială), dacă avem s core-uri atunci pT devine p/sT, deci accelerarea (raportul dintre timpul inițial și nou timp cu s core-uri) este S = T/((1-p)T + p/s T) = 1/ (1-p + p/s)
pentru Gustafson
sistemul este capabil să execute E = (1-p)E + pE iar partea care se poate îmbunătății este doar pE, care devine δpE, deci execuția este îmbunătățită ((1-p)E + δpE)/E = 1 – p + δp
c) S_{Amdahl} = 1 și $S_{\text{Gustafson}}$ = 1 (nicio îmbunătățire)
d) S_{Amdahl} = 1 și $S_{\text{Gustafson}}$ = 1 (nicio îmbunătățire)
e) L_{Amdahl} = 1/(1-p), S_{Amdahl} < 1/(1-p)
f) $L_{\text{Gustafson}}$ = ∞
g) verificați explicațiile de la punct b) și țineți cont de rezultatele la limitele pentru p, s și δ

7. Considerăm un sistem de calcul de 32 de biți. Sistemul poate realiza operațiile următoare: operații aritmetice/logice (1 ciclu), operații de citire/scriere date în memorie (2 cicli) și operații de branch/salt (3 cicli). Avem un program care are în componență 20% operații aritmetice/logice, 50% operații de citire/scriere (împărțite egal) și 30% operații de branch/salt. Cerințe:

   (a) calculați numărul mediu de cicli de ceas pe instrucțiune;

   (b) observăm că operațiile aritmetice/logice sunt mereu grupate cu operații de citire. Ne decidem să adăugăm o nouă instrucțiune care realizează și citirea și operația aritmetică/logică (totul într-un singur ciclu). Care este noul număr mediu de cicli de ceas pe instrucțiune?

   (c) care este timpul de execuție pentru program înainte și după modificarea de la punctul anterior (notăm numărul de instrucțiuni al programului cu N și frecvența sistemului cu f)?

   (d) suplimentar modificării de mai sus, presupunem că îmbunătățim sistemul de calcul și mărim frecvența cu 20%. Care este accelerarea de execuție a programului?

a) 2.1

b) 1.7

c) înainte de modificare N x 2.1 / f, după N x 1.7 / f

d) 1.48

- Observatii:
  1. la b) fiind 20% op aritmetice/logice atunci doar 20% din 50% vor deveni un ciclu restul ramand la 2. Deci calculul vine de la 0.2 * 1 + 0.3 * 2 + 0.3 * 3
  2. la d) calculul vine de la (2.1 * 1.2) / 1.7

8. Considerăm că rulăm același program pe două sisteme diferite. Pe sistemul X, programul execută 80 de instrucțiuni aritmetice/logice, 40 de operații citire/scriere memorie și 25 de instrucțiuni de branch/salt. Pe sistemul Y, programul execută 50 de instrucțiuni aritmetice/logice, 50 de operații citire/scriere memorie și 40 de instrucțiuni de branch/salt. Pe ambele sisteme, operațiile aritmetice/logice au 1 ciclu, operațiile de citire/scriere au 3 cicli iar operațiile de branch/salt au 5 cicli. Cerințe:

   (a) calculați ponderea tipurilor de operații pentru fiecare sistem;

   (b) calculați numărul mediu de cicli de ceas pe instrucțiune pentru fiecare sistem;

   (c) presupunem că frecvența sistemului Y este cu 20% mai ridicată decât cea a sistemului X. Care sistem execută programul mai repede?

a) p = 80/145, p = 40/145 și p = 25/145 pentru sistemul X

p = 50/140, p = 50/140 și p = 40/140 pentru sistemul Y

b) 2.24 pentru sistemul X și 2.86 pentru sistemul Y

c) cpu time pe sistemul X = 145 x 2.24 / f

cpu time pe sistemul Y = 140 x 2.86 / (1.2 x f)

accelerarea este raportul valorilor: Y / X ≈ 1.03 (sistemul X este cu 3% mai rapid decât Y)

- Observatii:
  1. formula timp de executie pentru un program = N / f
     N = numarul de instructiuni al programului
     f = frecventa sistemului
  2. CPU time pe un sistem: (N * numar mediu de cicli de ceas pe instructiune) / f

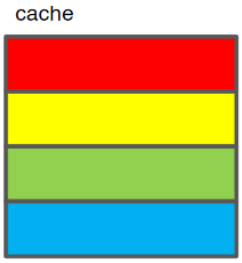
9. Considerăm un sistem de calcul pe 32 de biți care are un cache de 16 Kbytes. Cerințe:

   (a) considerăm că memorile sunt împărțite în blocuri de 32 de bytes. Câte blocuri sunt posibile în memoria principală? Câte blocuri sunt în memoria cache?

   (b) observați că numărul de blocuri din memoria principală poate să fie mult mai mare decât numărul de blocuri din memoria cache (e normal, memoria principală este mult mai multă decât memoria cache). Copiem bucăți din memoria principală în memoria cache pe blocuri. Trebuie să știm unde găsim în cache un byte pe care am vrea să îl citim din memoria principală. Sugerați moduri de a face corespondența între memoria principală și memoria cache.

a) $2^{32} / 2^5 = 2^{27}$, $(2^4 \times 2^{10}) / 2^5 = 2^9 = 512$ **blocuri**

b) **un exemplu în care cache are doar 4 blocuri, cea mai simplă idee: un bloc din memoria principală dacă e în cache poate să fie doar într-o poziție din cache cu aceeași culoare**



1) trebuie să știm ce culoare suntem

2) după ce știm culoare, trebuie să știm care block din memoria principală este cel corect (din toate cele roșii)

3) trebuie să știm unde în bloc este byte-ul pe care îl vrem

18 biți (ce rămâne pentru a identificare care bloc de aceeași culoare e cel corect)	9 biți (pentru că sunt 2^9 = 512 blocuri/culori în cache)	5 biți (pentru că sunt 2^5 = 32 bytes posibili în bloc)