ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x05

ÎNMULȚIREA/ÎMPĂRȚIREA NUMERELOR ÎNTREGI, REPREZENTAREA ÎN VIRGULĂ MOBILĂ

Cristian Rusu

DATA TRECUTĂ

- **detaliile unui circuit de adunare binară**
- **multiplexare**
- **circuite secvențiale**
  - SR Latch
  - D Latch
  - D Flip Flop
  - circuit adunare secvențial
- **reprezentarea pe stări a unui circuit**

CUPRINS

- logică secvențială + combinatorială, un exemplu
- înmulțirea numerelor întregi binare
- împărțirea numerelor întregi binare
- reprezentarea numerelor în virgulă mobilă
- lucrul cu numerele în virgulă mobilă

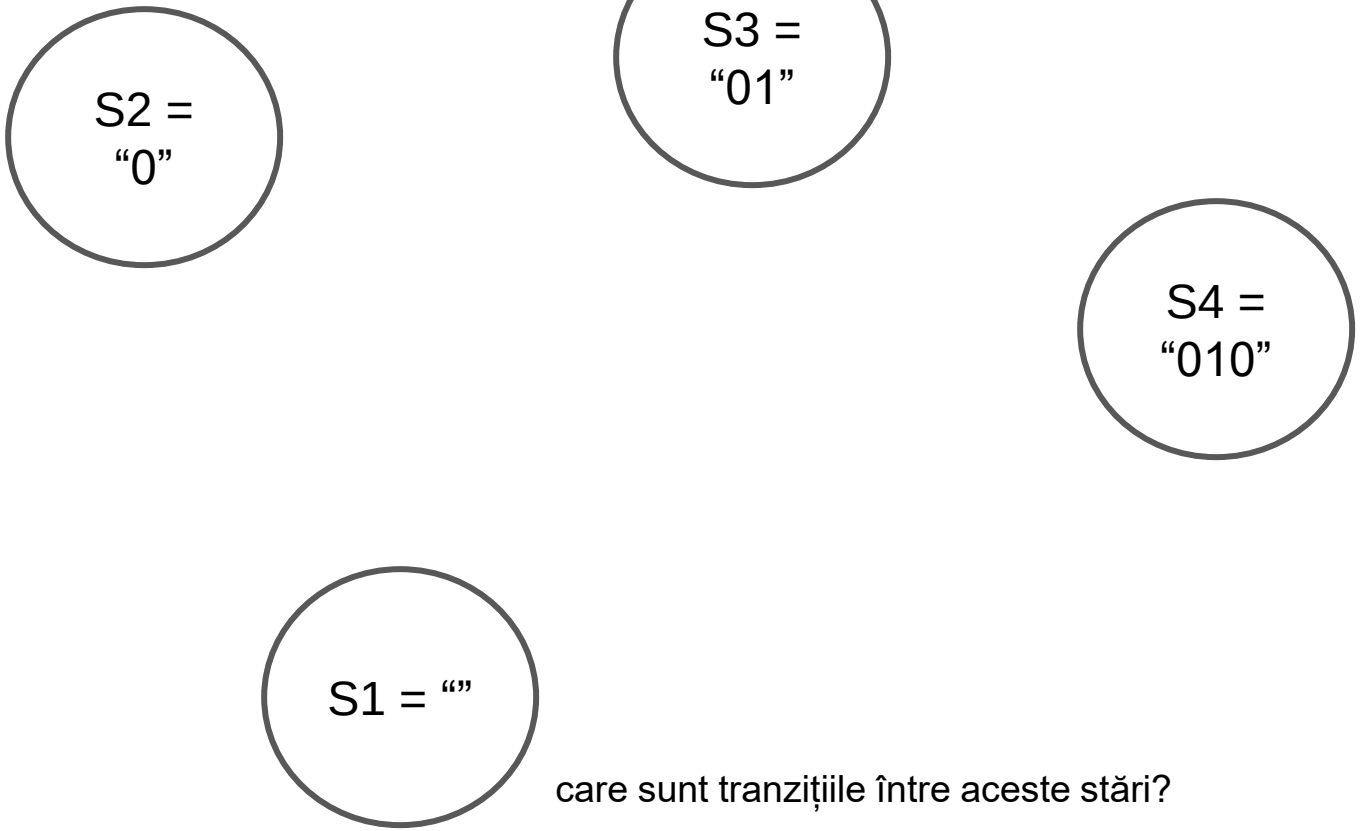
SECVENȚIAL, SEMINAR 0x02, EX 10

- **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 acestă secvență de biți este detectată în A atunci o variabilă Y este setată la 1, altfel această variabilă este 0.**
- **definim 4 stări**

.

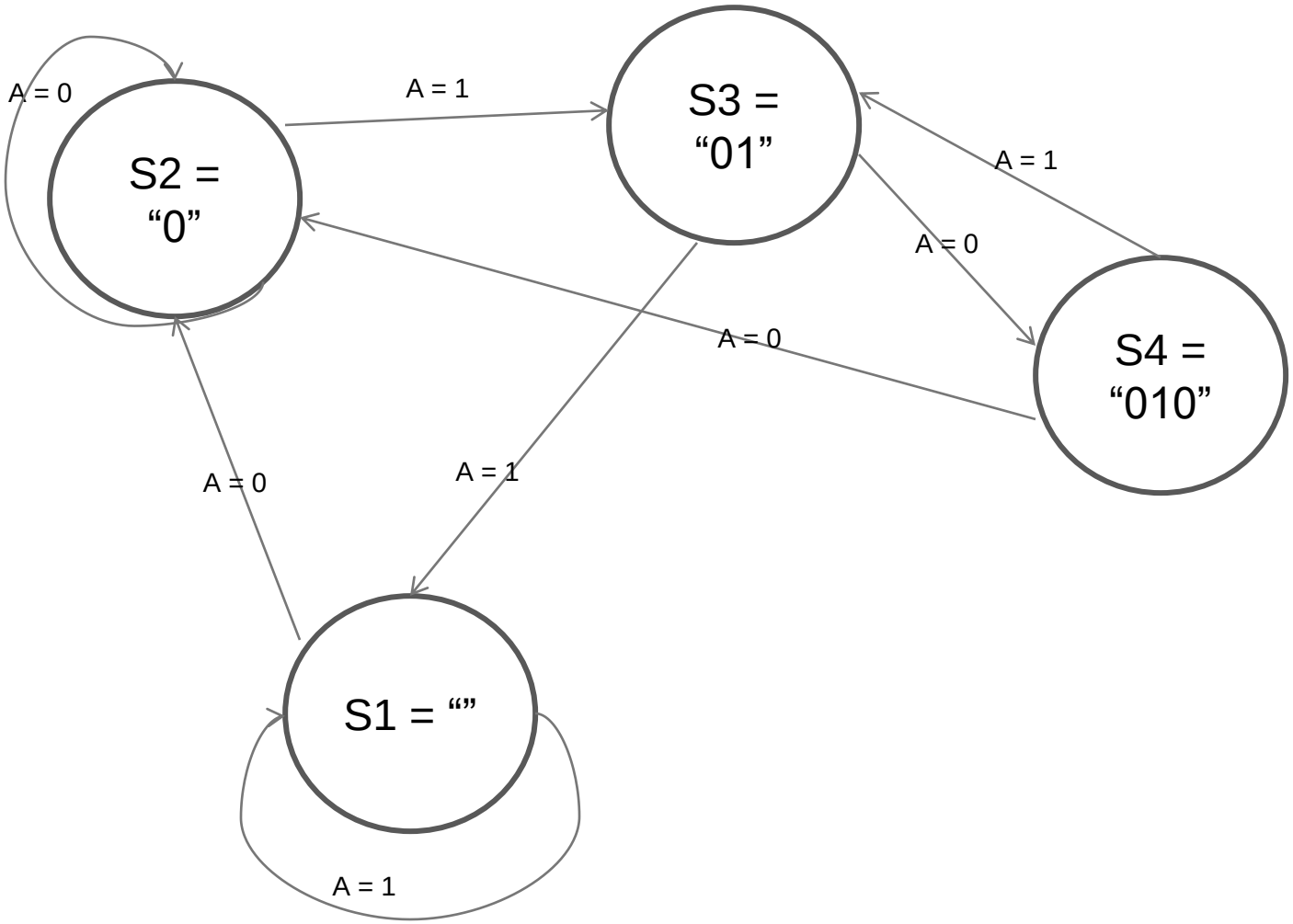
SECVENȚIAL, SEMINAR 0x02, EX 10

- **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**
- **definim 4 stări**

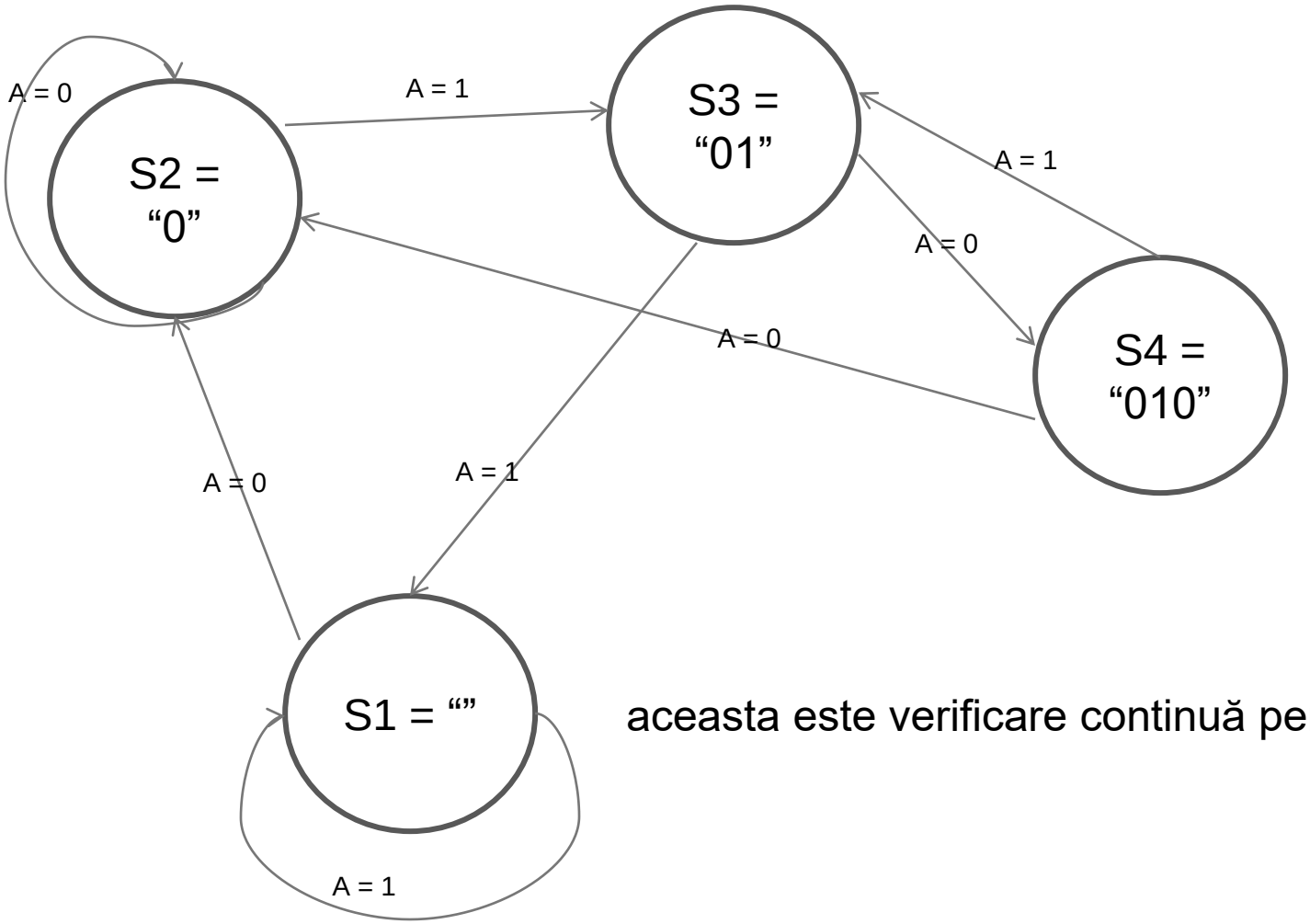


care sunt tranzițiile între aceste stări?

SECVENȚIAL, SEMINAR 0x02, EX 10



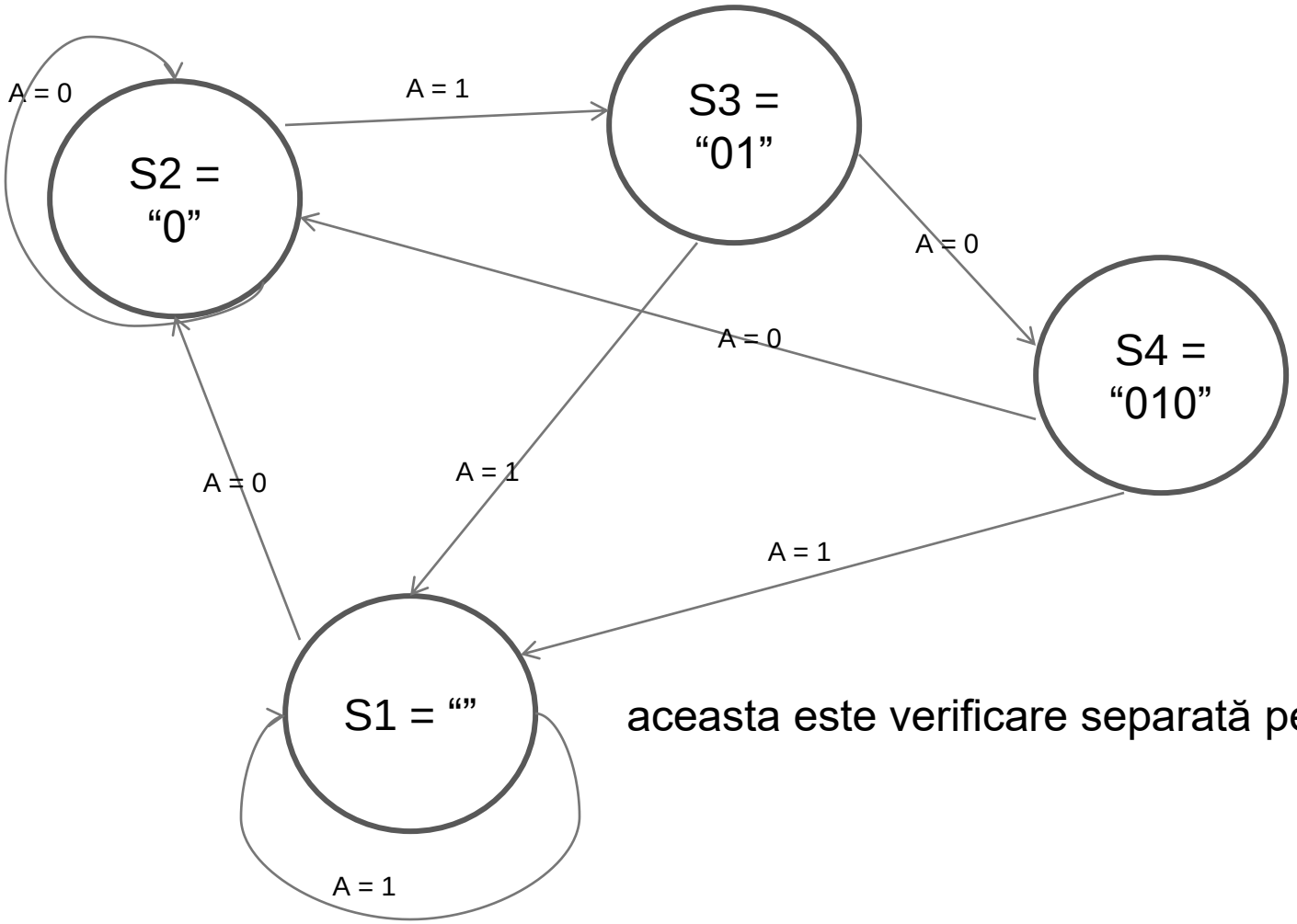
SECVENȚIAL, SEMINAR 0x02, EX 10



aceasta este verificare continuă pentru 010

.

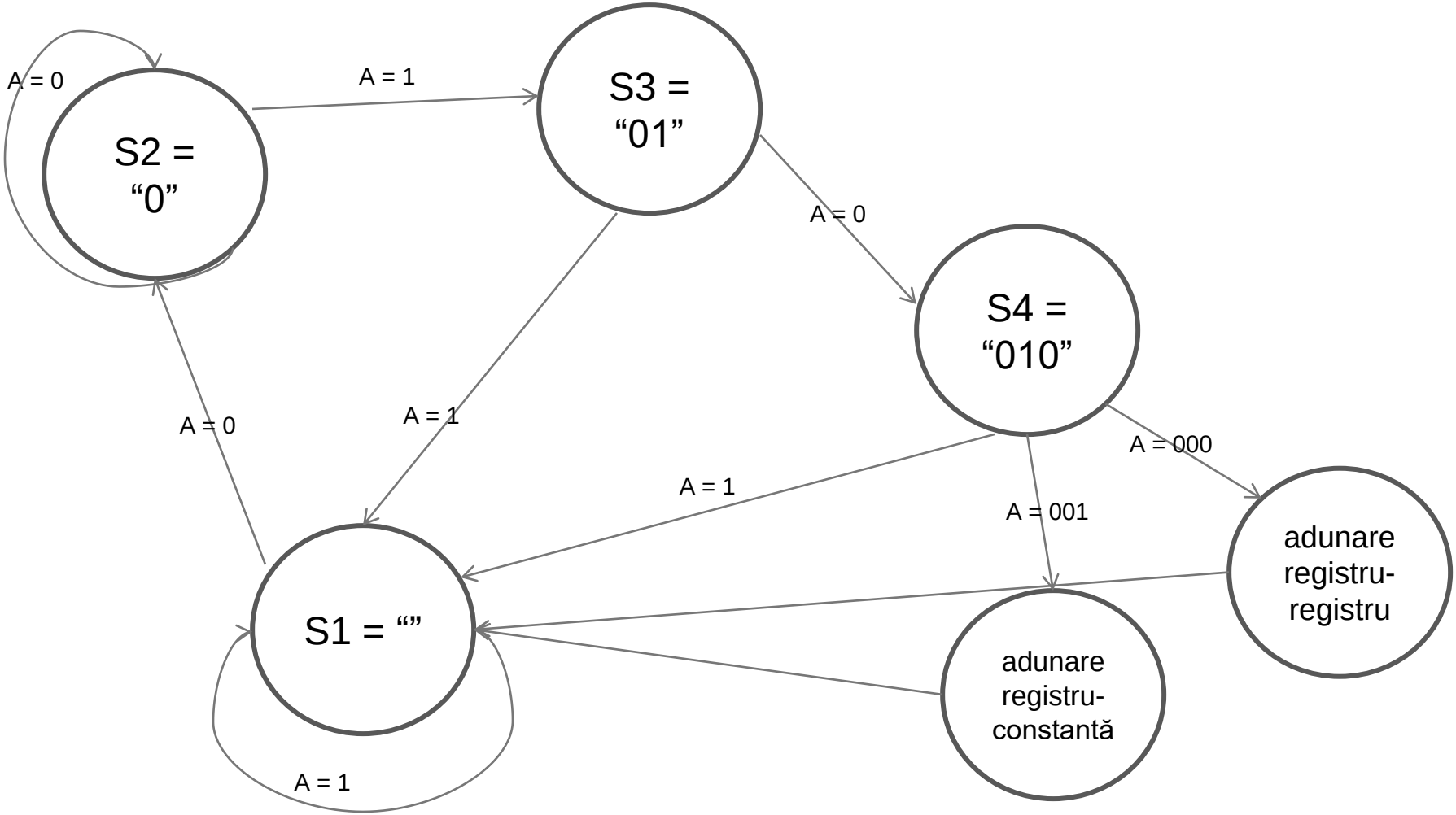
SECVENȚIAL, SEMINAR 0x02, EX 10



aceasta este verificare separată pentru 010

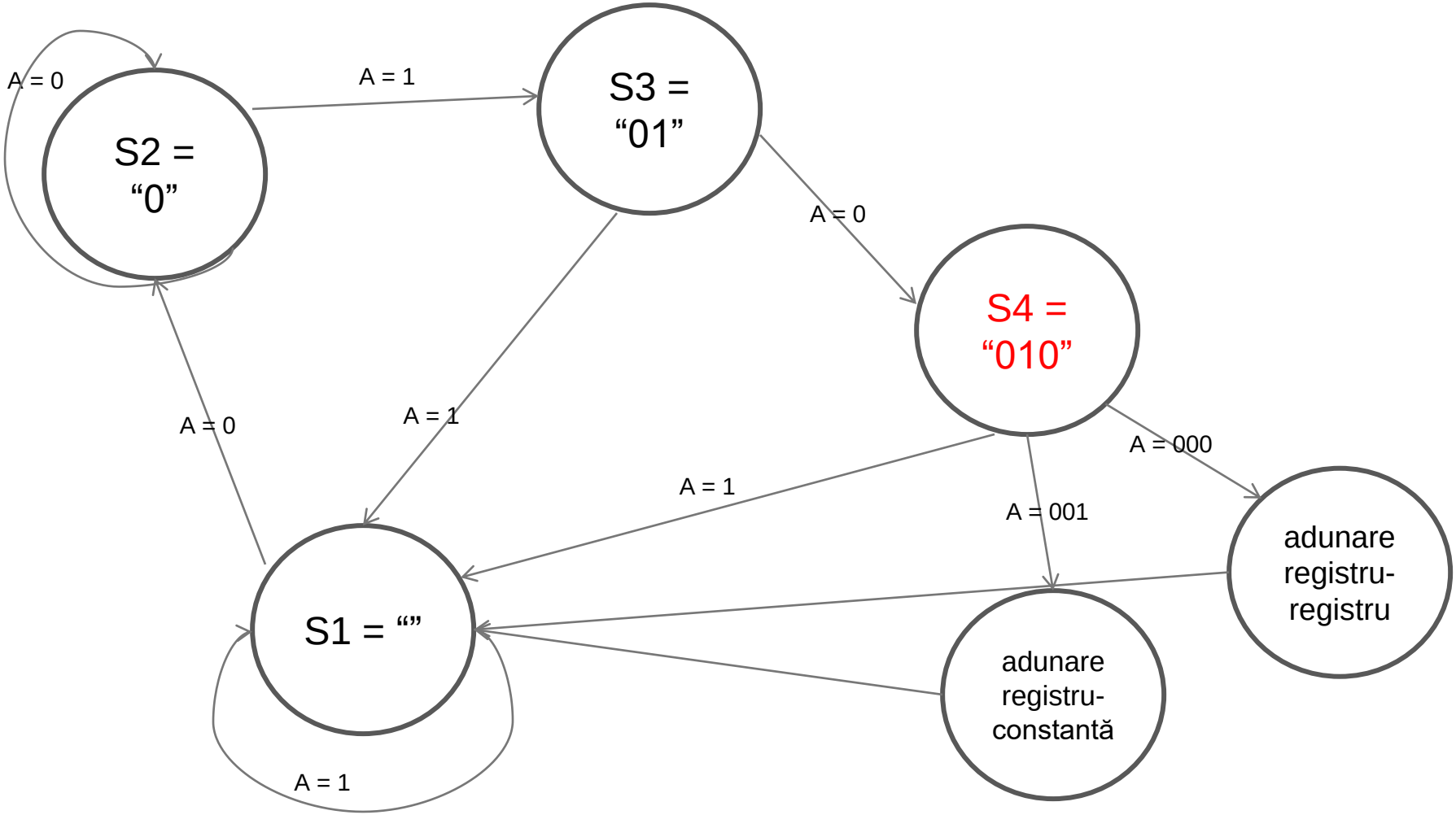
.

CE FACE UN PROCESOR



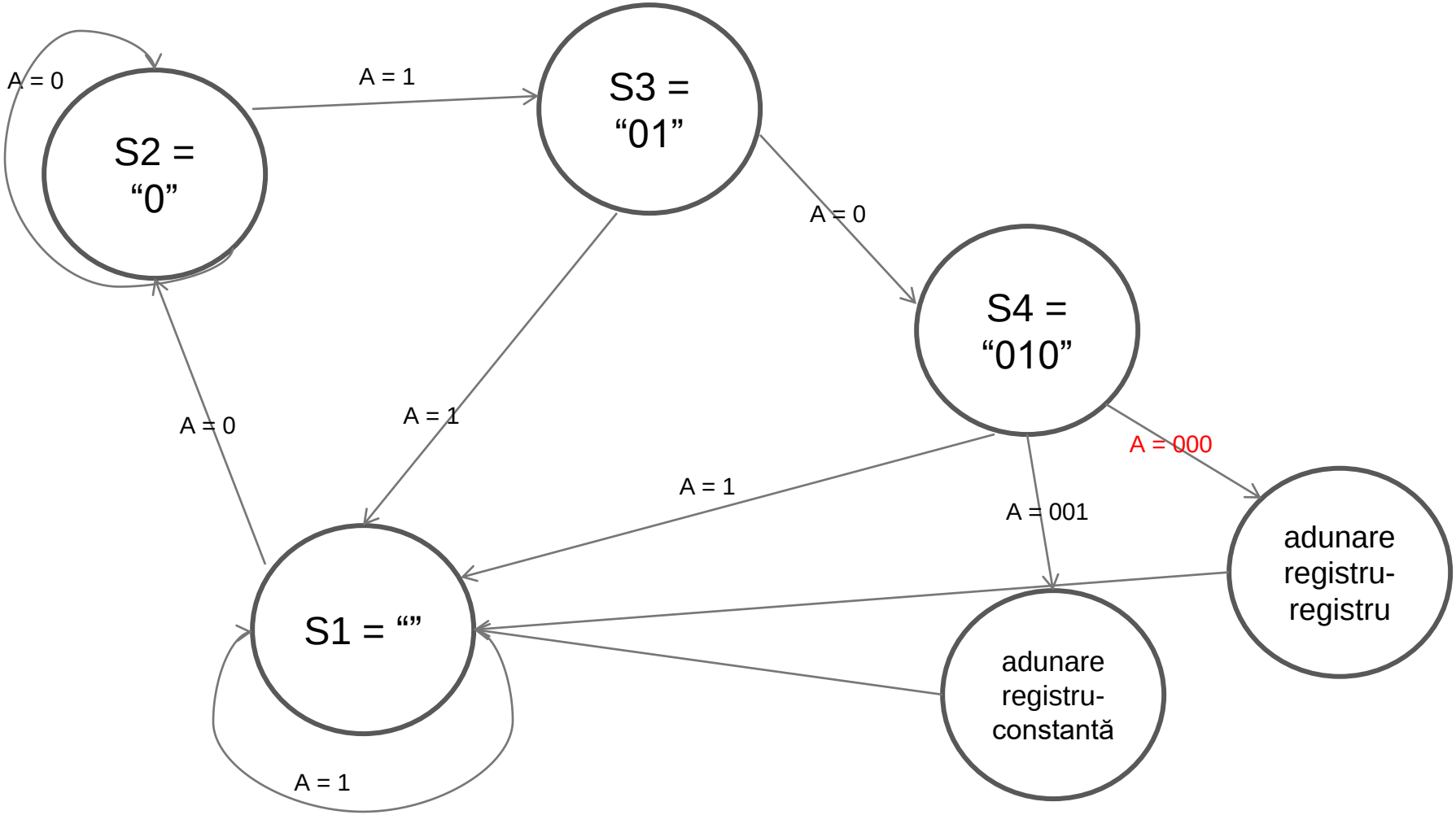
...0100001011010001010000100011001011 ...

CE FACE UN PROCESOR



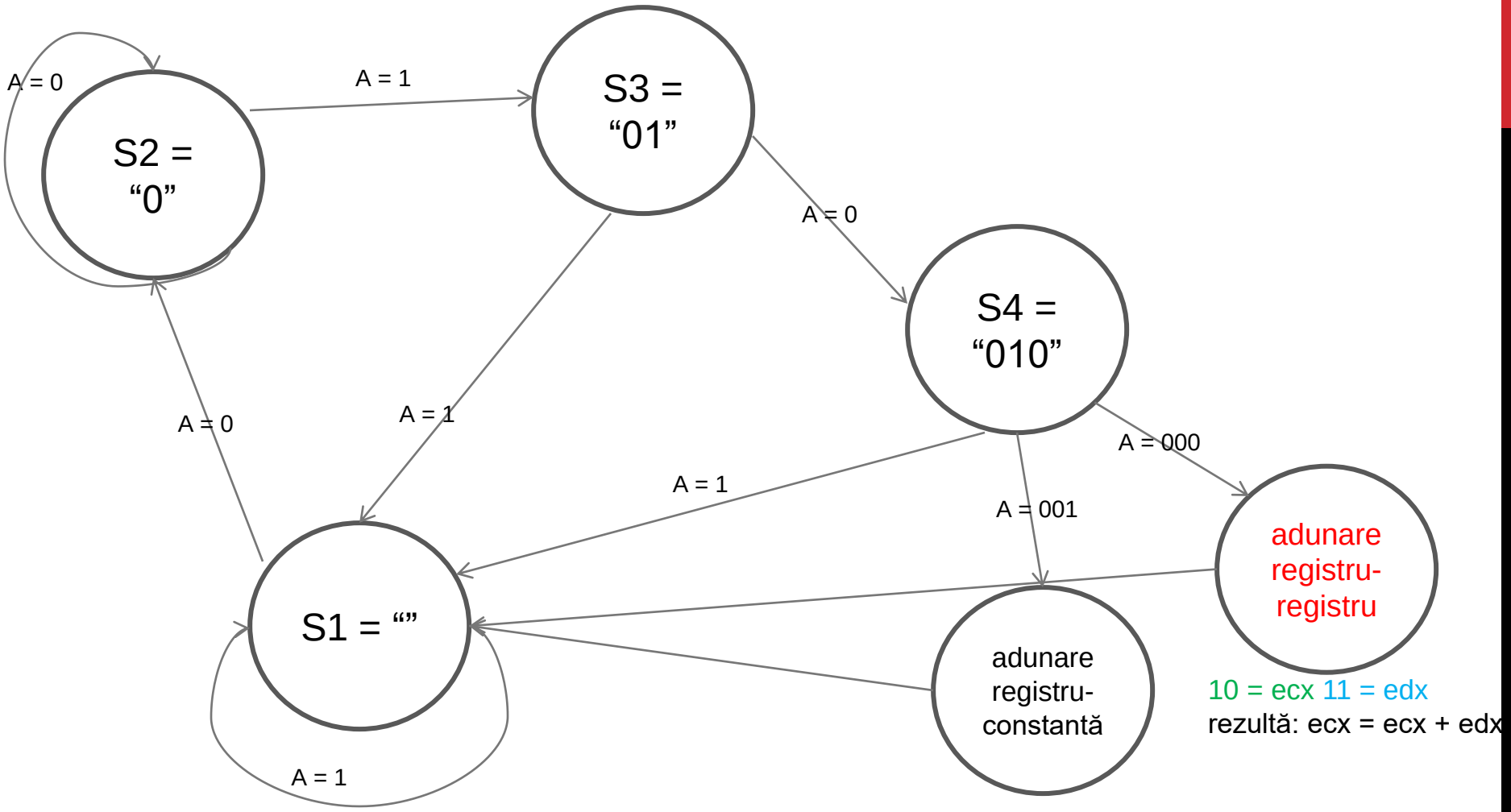
...010000101101000101000010001100101 1 ...

CE FACE UN PROCESOR



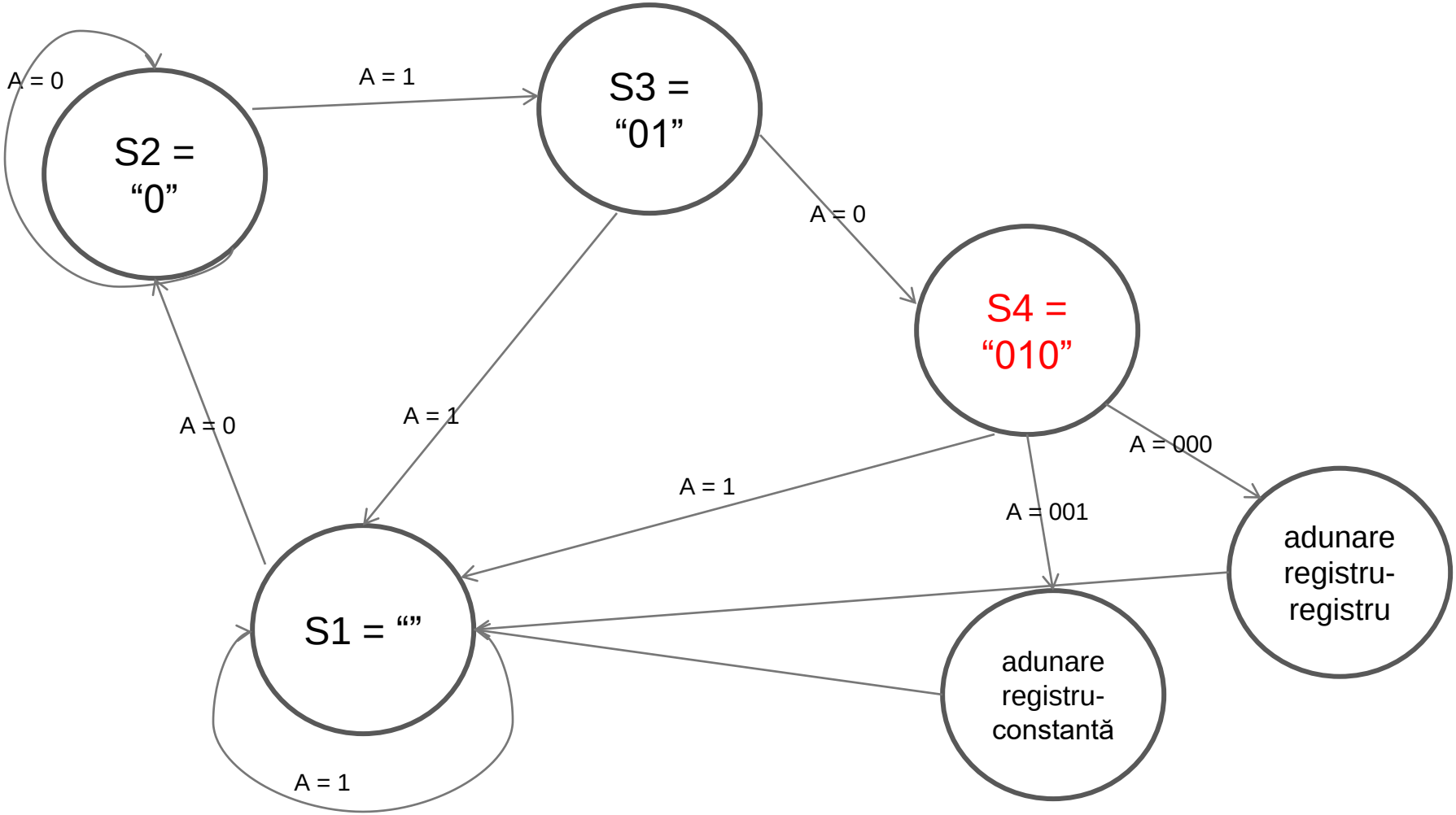
...010000101101000101000010001100101 1 ...

CE FACE UN PROCESOR



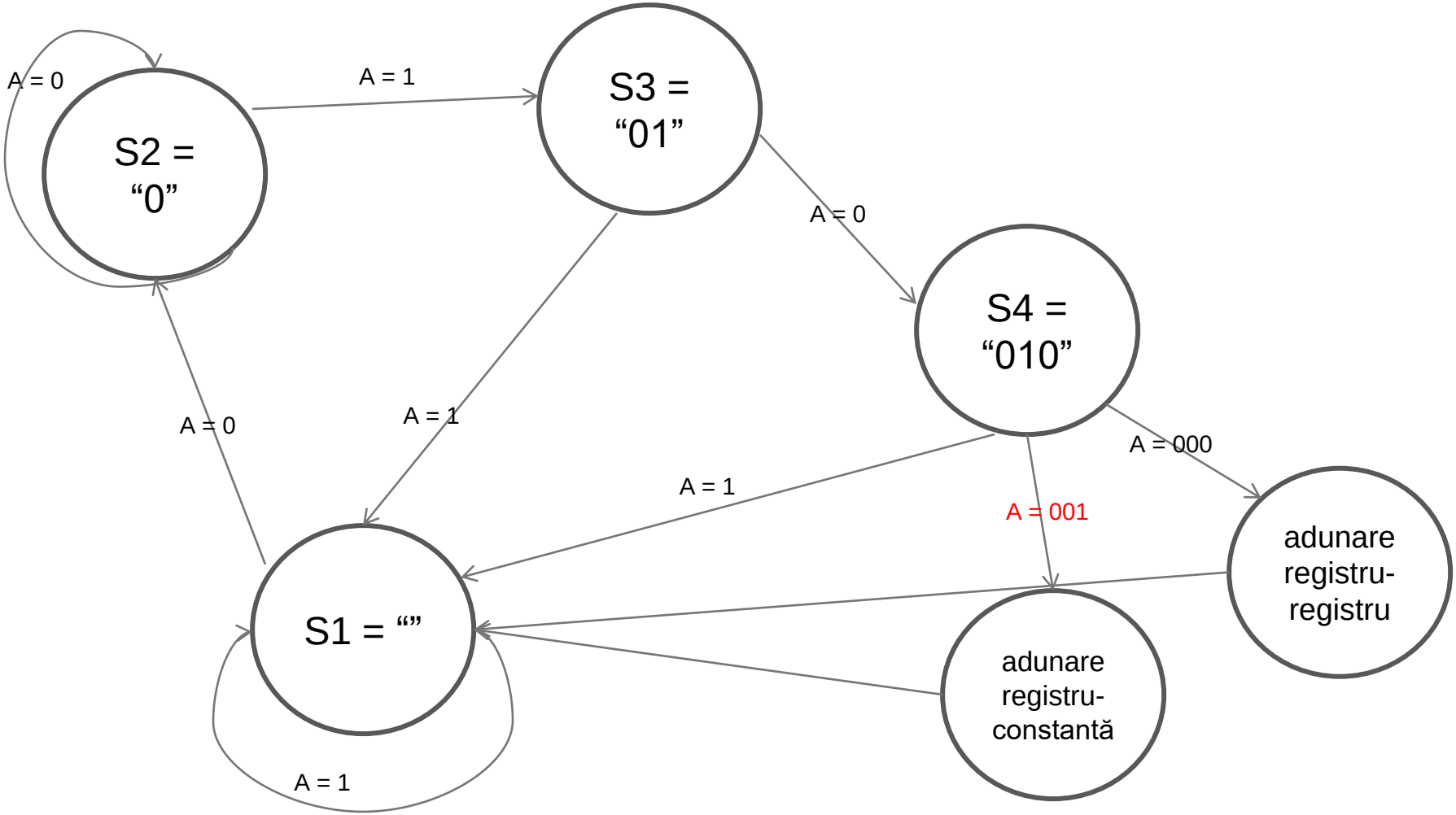
...01000010110100010100001000110010 11 ...

CE FACE UN PROCESOR



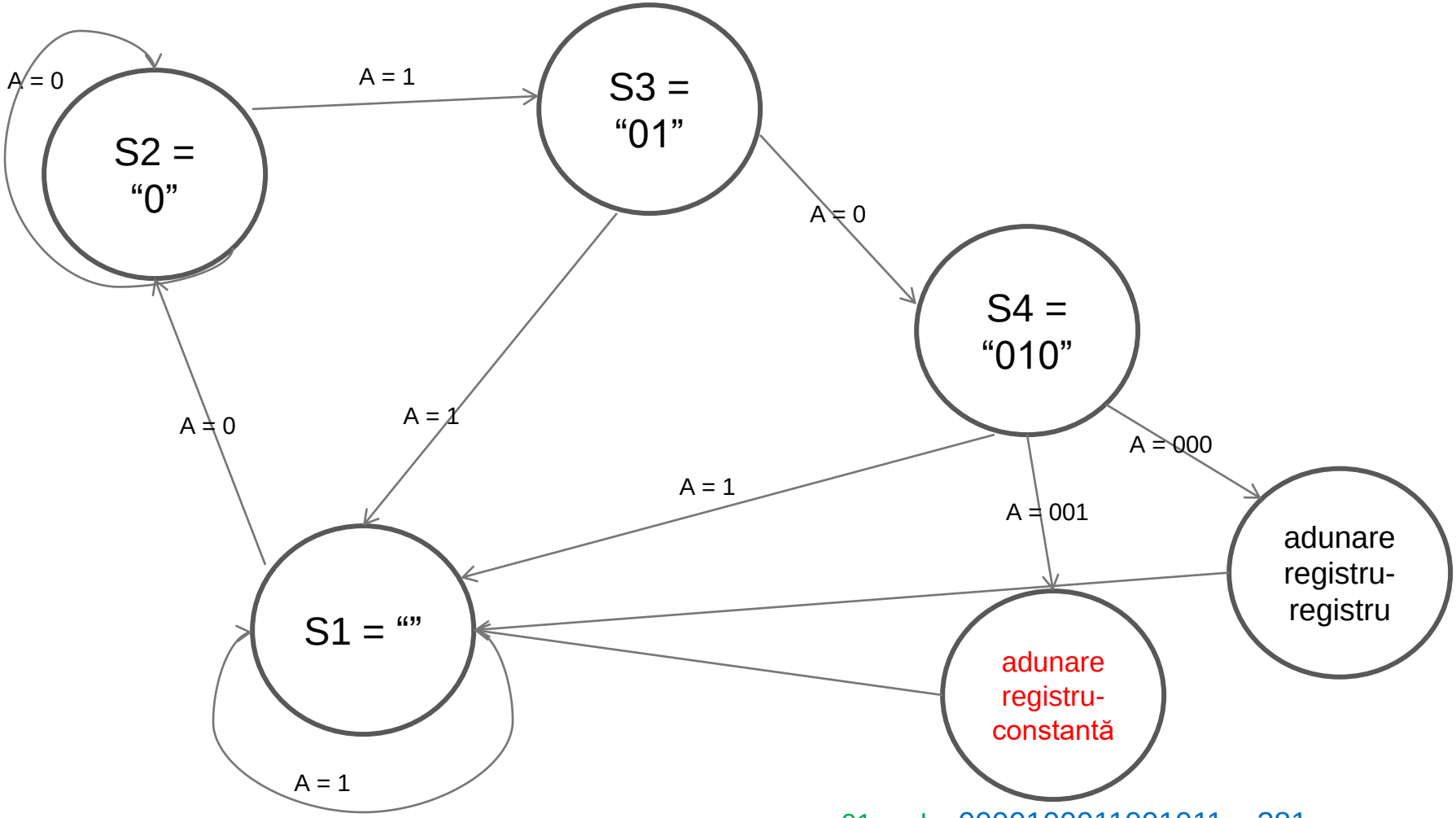
...0100001011010001010000100011001011 ...

CE FACE UN PROCESOR



...0100001011010001010000100011001011 ...

CE FACE UN PROCESOR

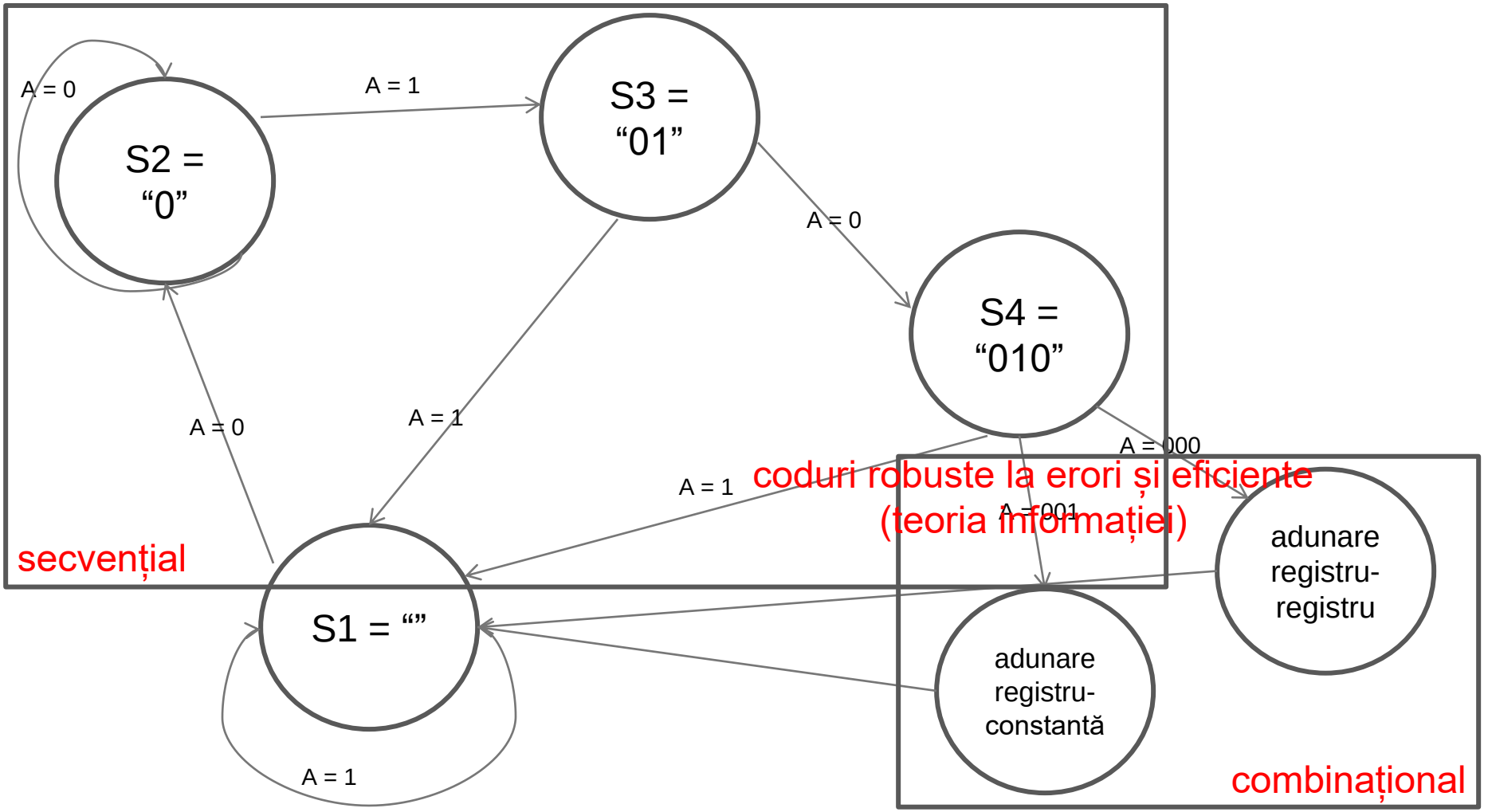


01 = ebx 0000100011001011 = 381

rezultă: ebx = ebx + 381

...01000010110100010100001000110010 11 ...

CE FACE UN PROCESOR



cod mașină ...0100001011010001010000100011001011 ...

CONȚINUT NOU PENTRU CURS

- **înmulțirea numerelor întregi binare**
- **împărțirea numerelor întregi binare**
- **reprezentarea numerelor în virgulă mobilă**
- **lucrul cu numerele în virgulă mobilă**

ÎNMULȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a x b**
  - *a* și *b* pe N biți
  - *s* pe ?? biți

1 1 1 0	*a*
0 1 0 1	*b*
	s

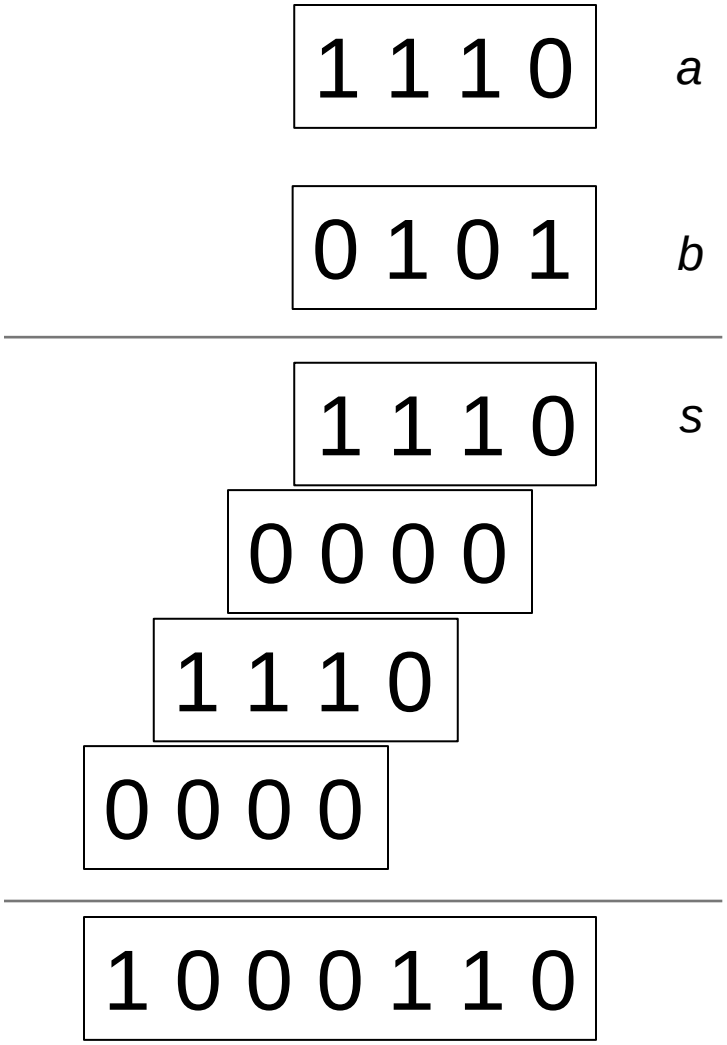
ÎNMULȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a x b**
  - *a* și *b* pe N biți
  - *s* pe 2N biți

1 1 1 0	*a*
0 1 0 1	*b*
	s

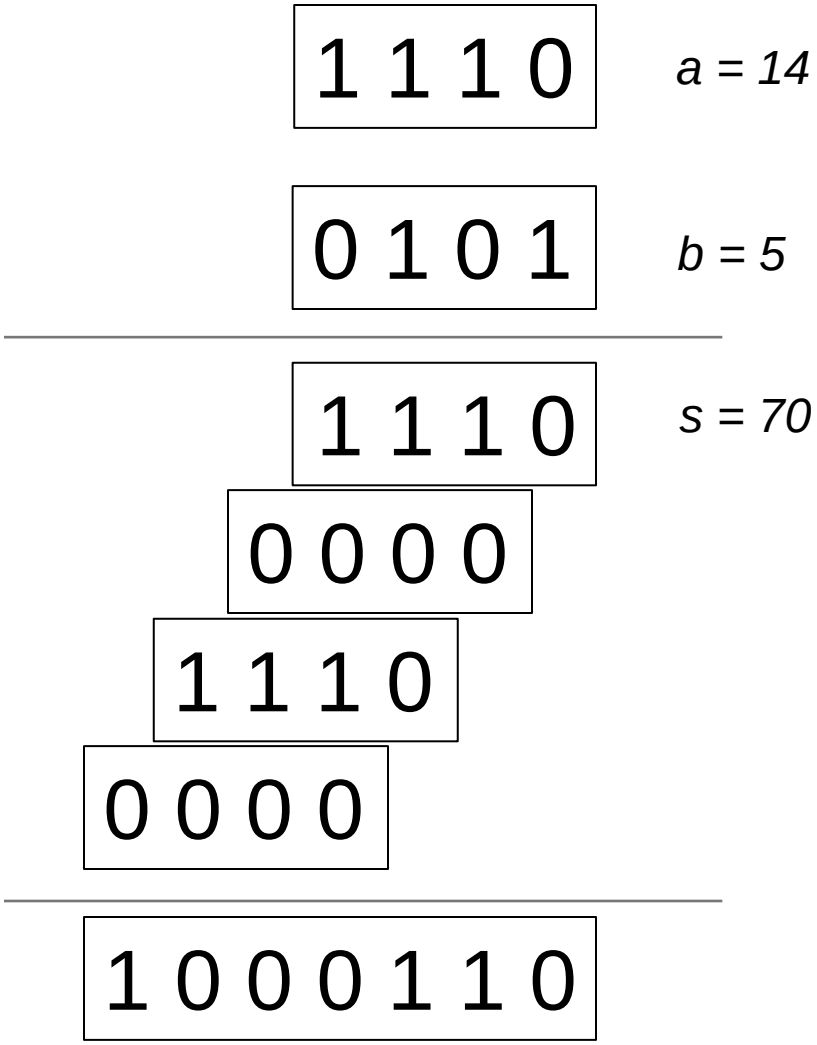
ÎNMULȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a x b**



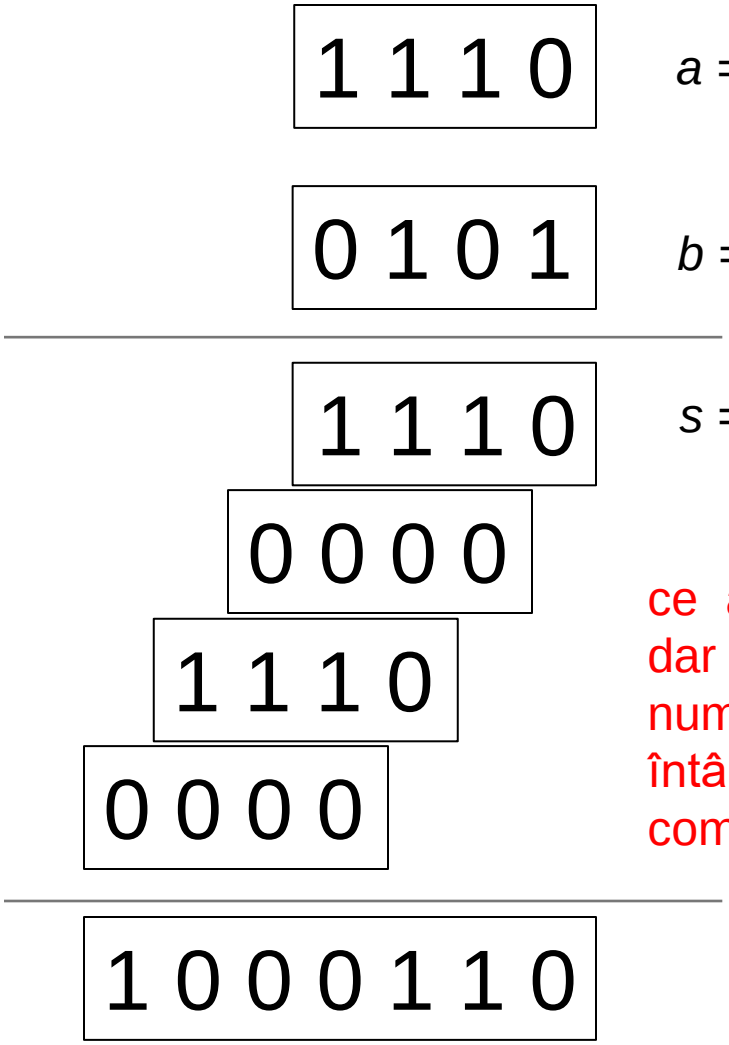
ÎNMULȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a x b**



ÎNMULȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a x b**



ce am făcut aici este corect, dar am presupus că am primit numere naturale. ce se întâmplă dacă *a* și *b* sunt în complement față de doi?

ÎNMULȚIREA NUMERELOR ÎNTREGI

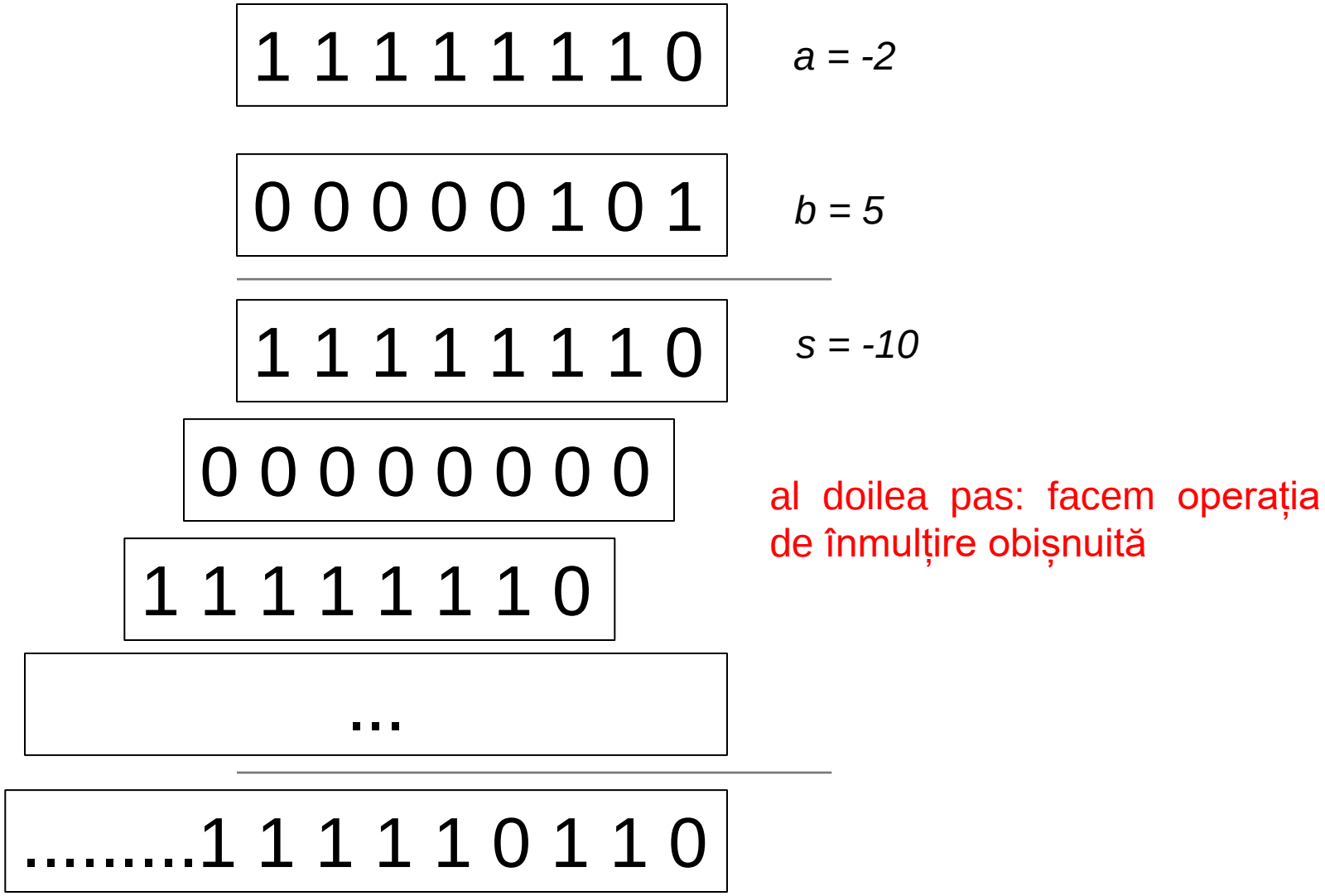
- **exemplu, s = a x b**

1 1 1 0	*a = -2*
0 1 0 1	*b = 5*
	s = -10

primul pas: extindem operanzii pe 8 biți

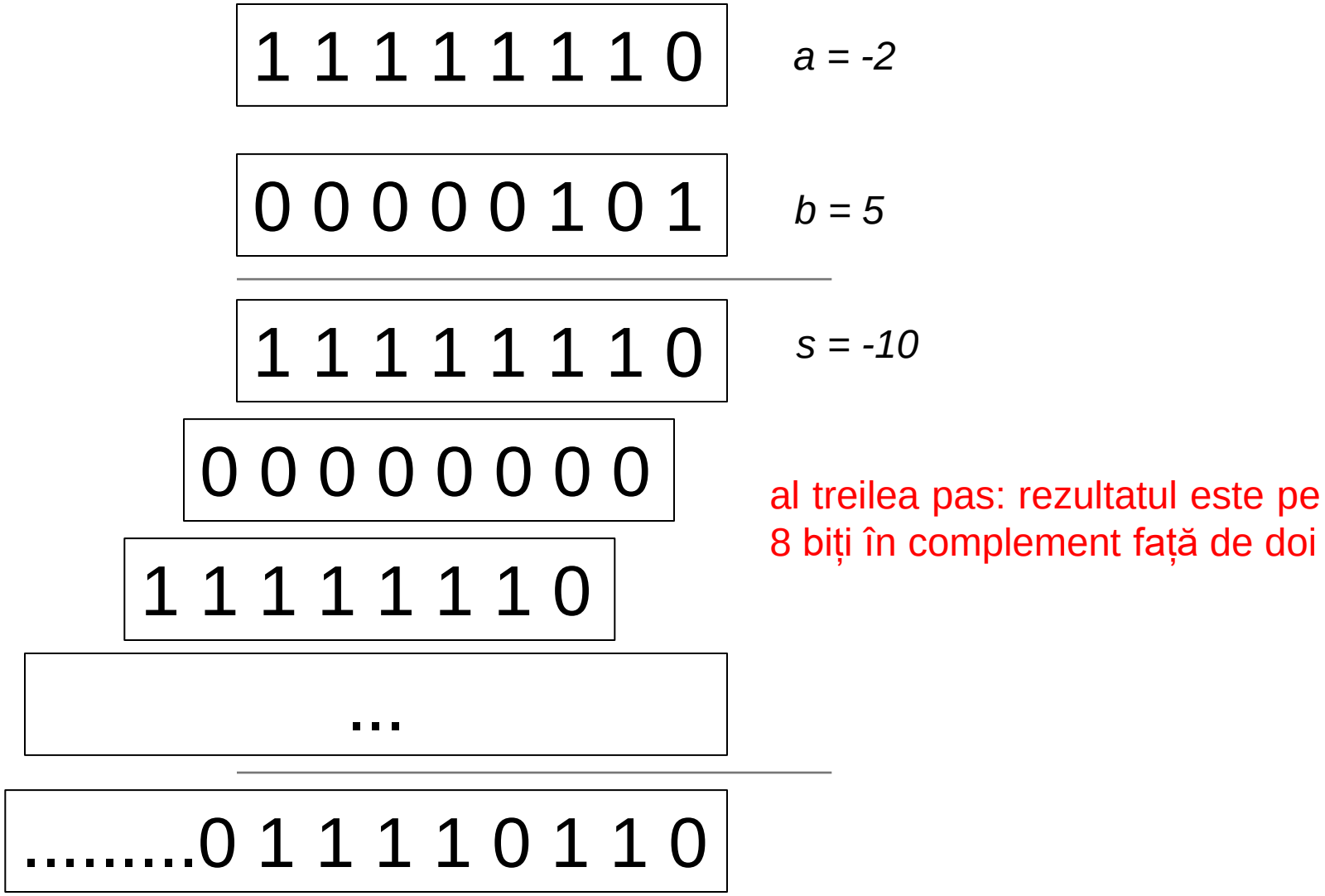
ÎNMULȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a x b**



ÎNMULȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a x b**

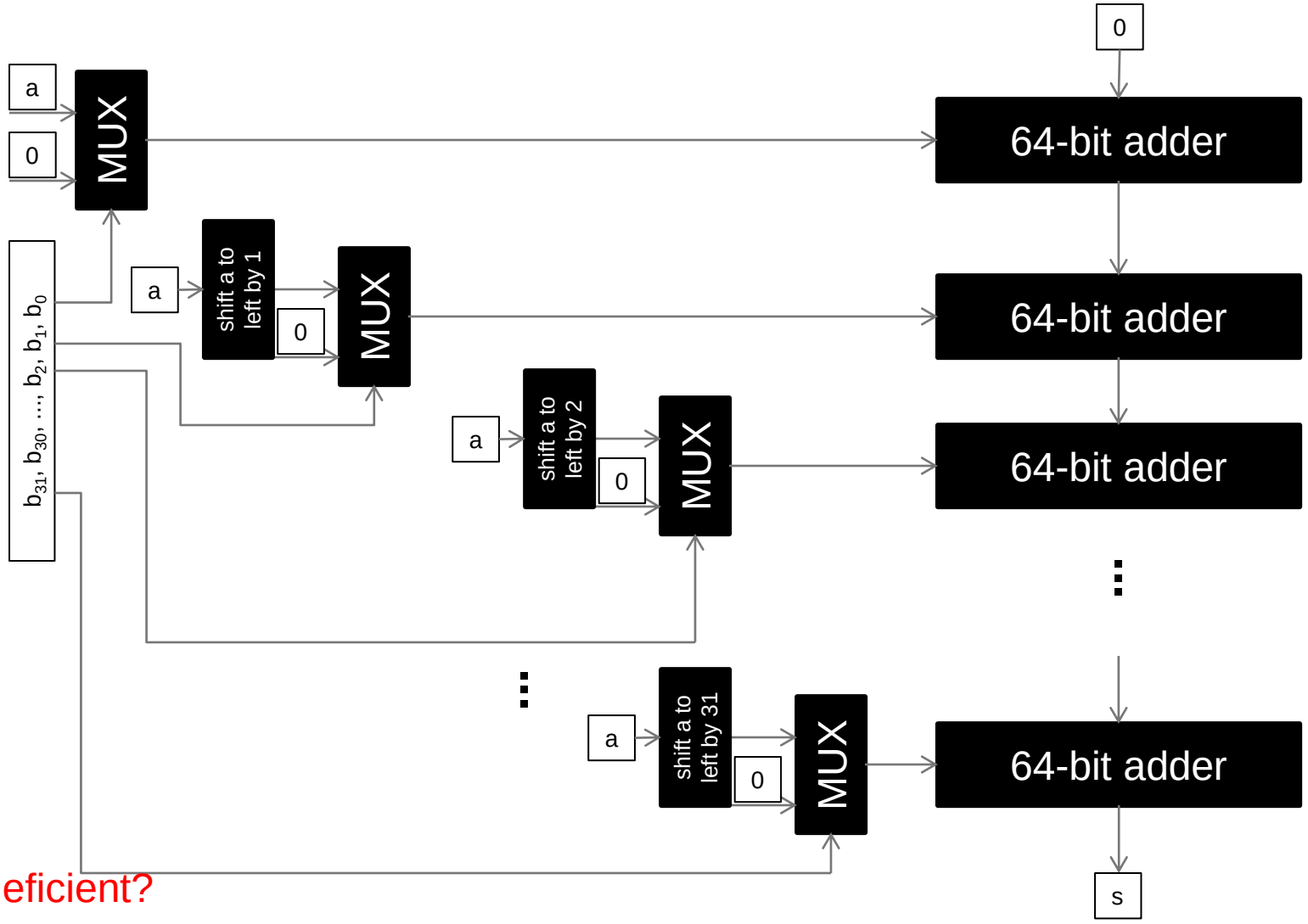


ÎNMULȚIREA NUMERELOR ÎNTREGI

- **circuitul combinațional**
  - s = a x b, a și b sunt numere pe 32 de biți
  - cum facem?

ÎNMULȚIREA NUMERELOR ÎNTREGI

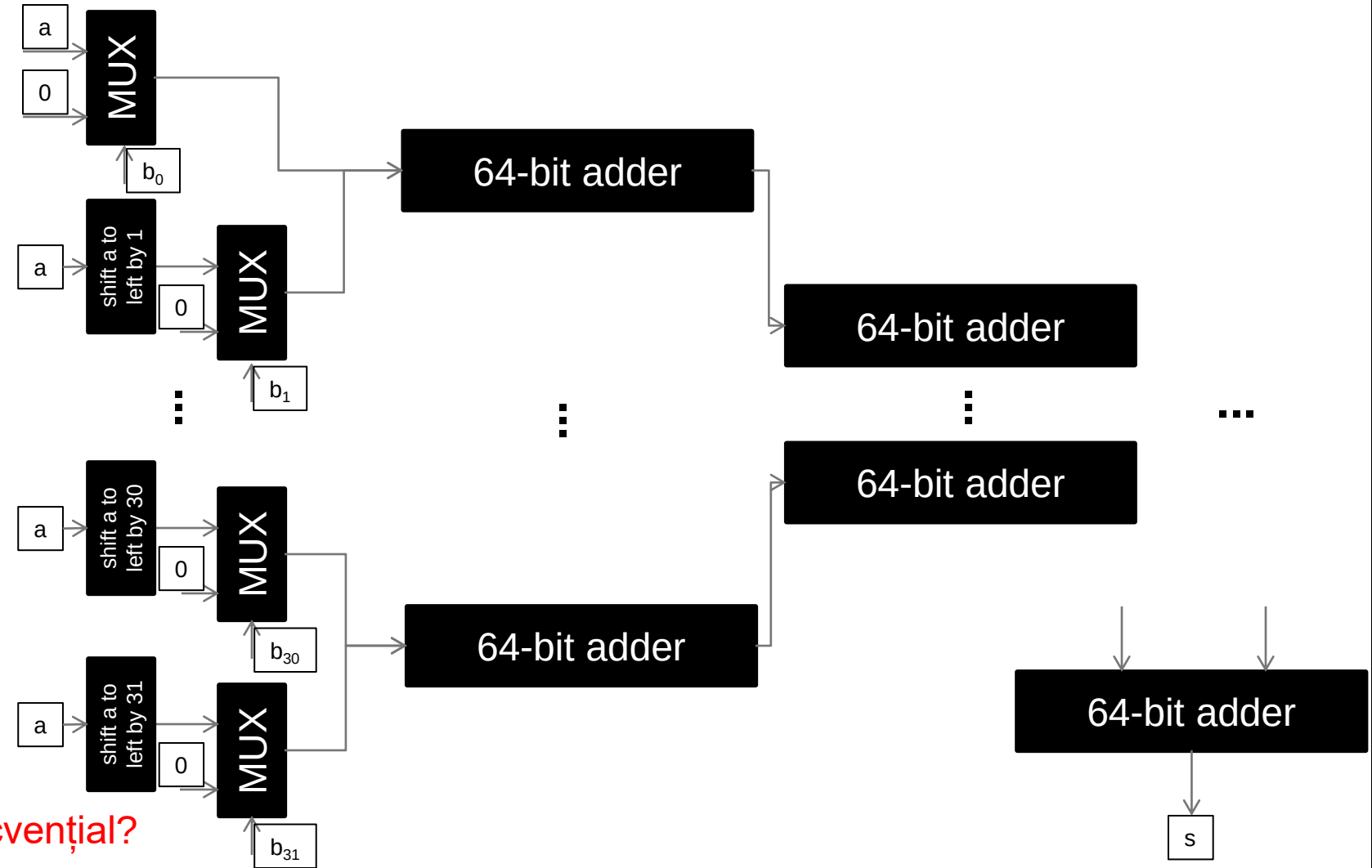
- **circuitul combinațional**
  - s = a x b, a și b sunt numere pe 32 de biți



ceva mai eficient?

ÎNMULȚIREA NUMERELOR ÎNTREGI

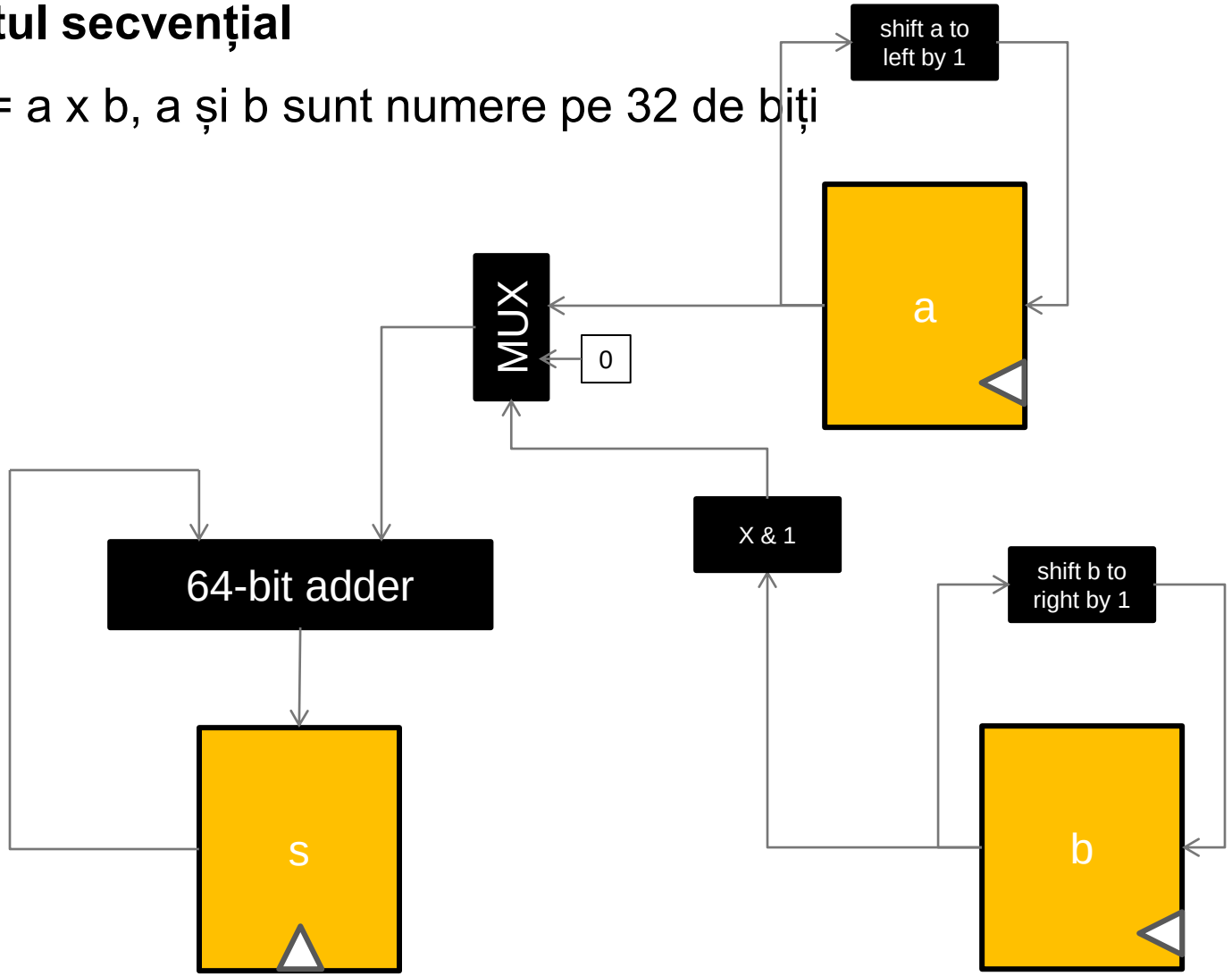
- **circuitul combinațional**
  - s = a x b, a și b sunt numere pe 32 de biți



ceva secvențial?

ÎNMULȚIREA NUMERELOR ÎNTREGI

- **circuitul secvențial**
  - s = a x b, a și b sunt numere pe 32 de biți



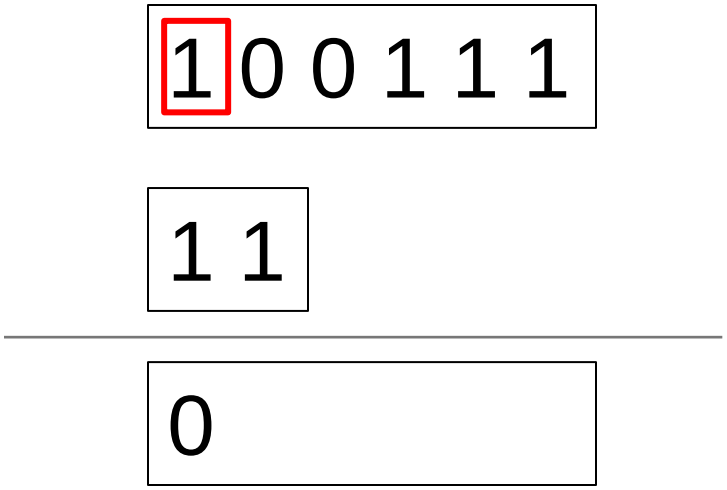
ÎMPĂRȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a ÷ b**

1 0 0 1 1 1
1 1

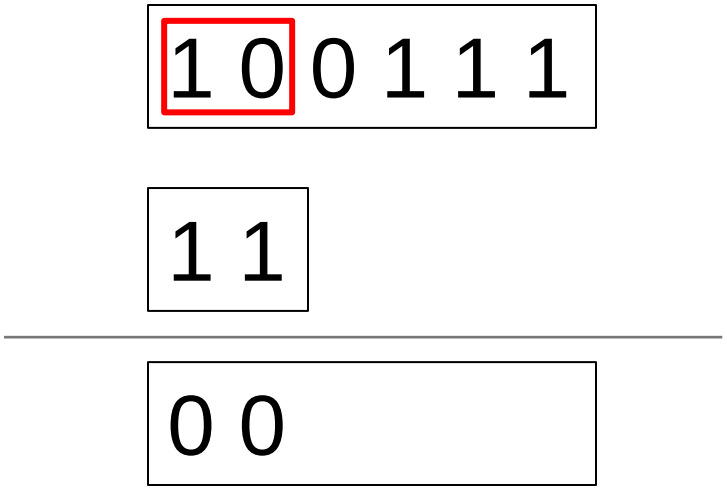
ÎMPĂRȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a ÷ b**



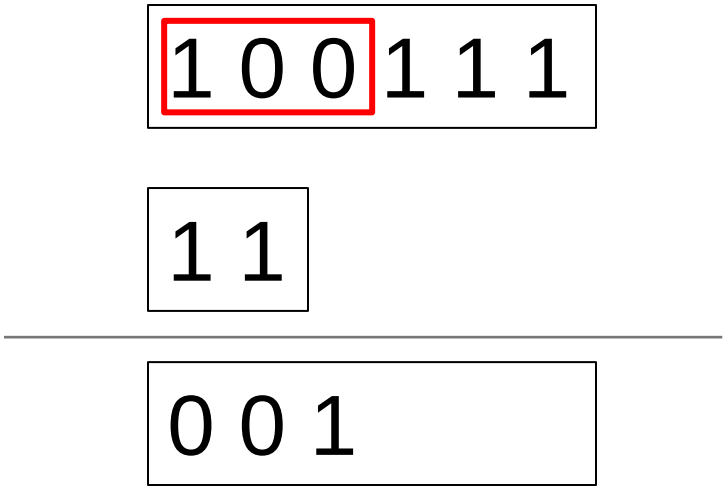
ÎMPĂRȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a ÷ b**



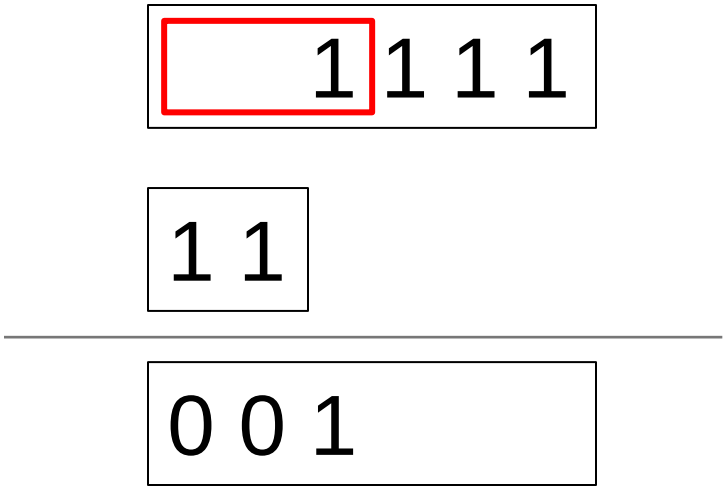
ÎMPĂRȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a ÷ b**



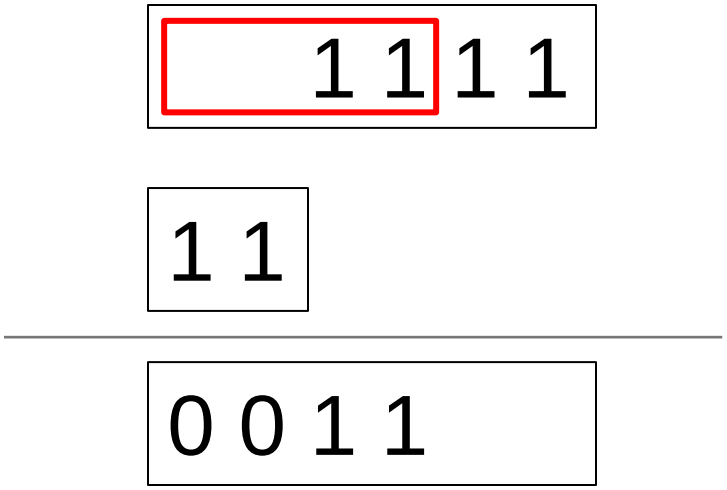
ÎMPĂRȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a ÷ b**



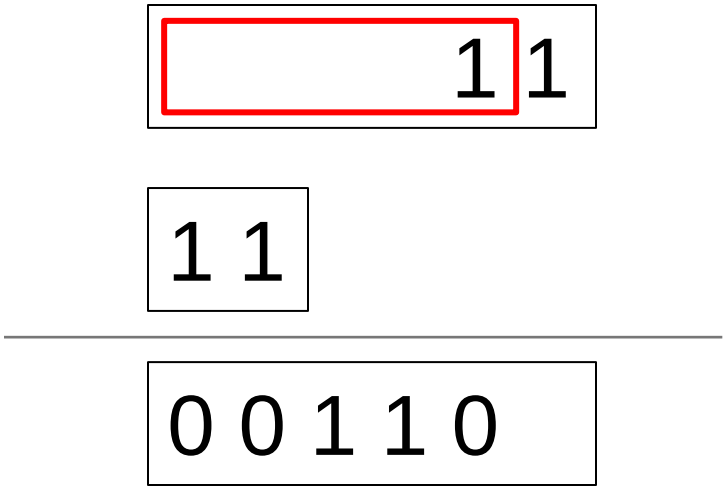
ÎMPĂRȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a ÷ b**



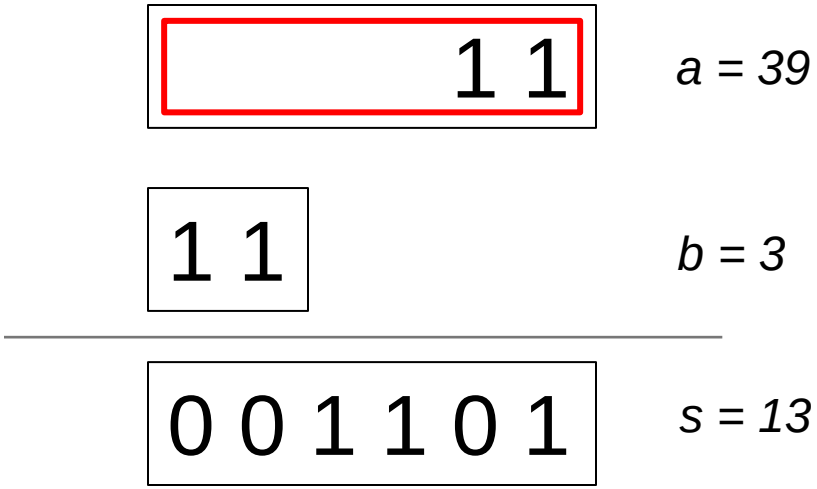
ÎMPĂRȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a ÷ b**



ÎMPĂRȚIREA NUMERELOR ÎNTREGI

- **exemplu, s = a ÷ b**



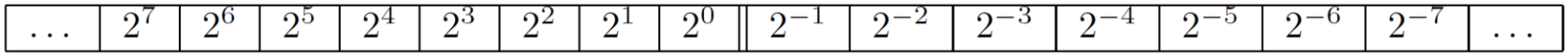
ÎMPĂRȚIREA NUMERELOR ÎNTREGI

- **s = a ÷ 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 x b + r
    - semnul lui r este semnul lui a
  - circuitul pentru împărțire nici nu vom încerca să îl facem
  - din cauza acestei complexități ridicate, compilatoarele și sistemele de calcul vor face tot posibilul pentru a evita o împărțire
  - vedem mai multe exemple la seminar ...

.

REPREZENTAREA ÎN VIRGULĂ MOBILĂ

- **am discutat la Seminar 0x00 despre reprezentarea în virgulă fixă**



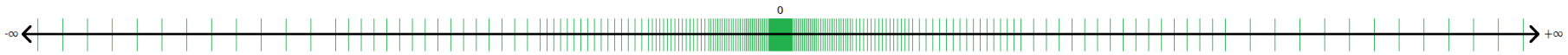
- **exemplu: 7.5 e scris ca 111.1**

- **care este problema cu acestă reprezentare?**
  - partea întreagă este separată de partea fracționară
  - fiecare are nevoie de un număr de biți prestabilit
  - asta poate să fie ineficient
  - vrem ca numărul de biți total să fie alocat “dinamic”, în funcție de numărul pe care trebuie să îl reprezentăm

REPREZENTAREA ÎN VIRGULĂ MOBILĂ

- **când trebuie să reprezentăm un număr real**
  - nu putem să avem precizie infinită
  - avem un număr finit de biți, deci putem să scriem biții în circuite
  - avem nevoie de precizie variabilă
  - putem avea precizie “infinită” dacă avem numere raționale (și vom salva separat numărătorul și numitor ca întregi)

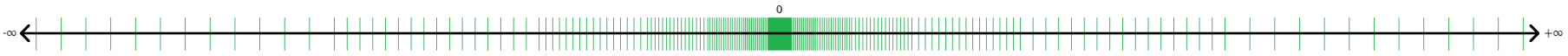
- **standardul: IEEE 754 Floating Point**
  - densitatea nu este uniformă pe linia reală



https://en.wikipedia.org/wiki/Floating-point_arithmetic

REPREZENTAREA ÎN VIRGULĂ MOBILĂ

- **standardul: IEEE 754 Floating Point**
  - densitatea nu este uniformă pe linia reală



- **sunt câteva consecințe**
  - (0.1 + 0.2) == 0.3 versus (0.2 + 0.3) == 0.5 (rontunjiri)
  - math.sqrt(3)*math.sqrt(3) == 3 versus 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
    - float(123456789101112) + 1.0 = 123456789101113.0
    - float(1234567891011121) + 1.0 = 1234567891011122.0
    - float(12345678910111213) + 1.0 = 1.2345678910111212e+16

testele sunt făcute în C și python cu variable care sunt single precision (FP 32 de biți), umătorul număr după 16777216 este 16777218
alt exemplu: după 167772160 vine 167772176 (cu cât cresc numerele cu atât distanțele între două consecutive sunt mai mari – ca în desenul de pe pagina anterioară)

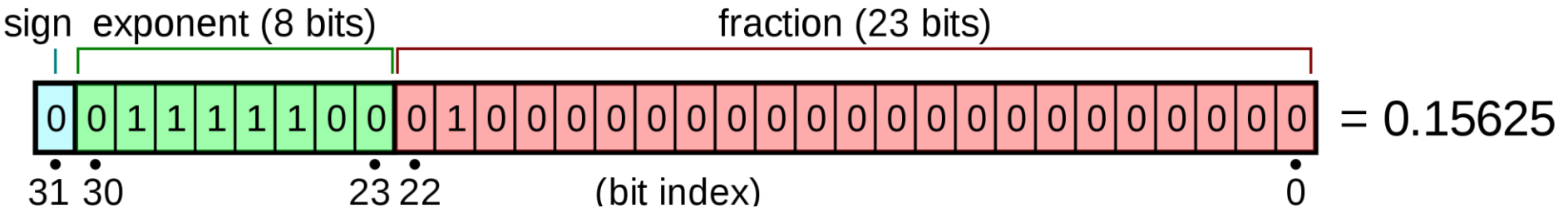
REPREZENTAREA ÎN VIRGULĂ MOBILĂ

- **reprezentarea științifică**
  - $12345 = 1.2345 \times 10^4$
  - $5024 = 5.024 \times 10^3$
  - $0.00925 = 9.25 \times 10^{-3}$
  - float(12345678910111213) + 1.0 = 1.2345678910111212e+16
  - $101010 = 1.01010 \times 2^5$
    - în sistemul binar, primul bit din reprezentare este mereu 1

.

REPREZENTAREA ÎN VIRGULĂ MOBILĂ

- **standardul: IEEE 754 Floating Point**



- **$x = (-1)^s \ 1.mmmmmmmmmmmmmmmmmmmmmmm \ 2^{(eeeeeeee)_{10} - 127}$**

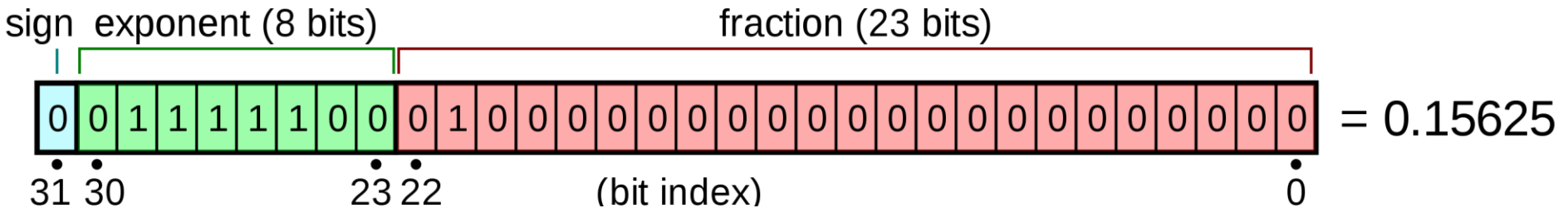
- **exemple:**
  - $0.15625 = (-1)^0 \ 1.0100...0 \ 2^{b01111100 - 127} = 1.25 \ 2^{-3} = 1.25/8$
  - alte exemple:
  - $(-1)^0 \ 1.1000...0 \ 2^{b01111100 - 127} = 1.5 \ 2^{-3} = 1.5/8 = 0.1875$
  - $1 = (-1)^0 \ 1.00000000000000000000000 \ 2^{b01111111-127}$
  - $-1 = (-1)^1 \ 1.00000000000000000000000 \ 2^{b01111111-127}$
  - $2 = (-1)^0 \ 1.00000000000000000000000 \ 2^{b10000000-127}$
  - $\infty = (-1)^0 \ 1.00000000000000000000000 \ 2^{b11111111-127}$
  - $-\infty = (-1)^1 \ 1.00000000000000000000000 \ 2^{b11111111-127}$

https://en.wikipedia.org/wiki/Floating-point_arithmetic

https://www.h-schmidt.net/FloatConverter/IEEE754.html

REPREZENTAREA ÎN VIRGULĂ MOBILĂ

- **standardul: IEEE 754 Floating Point**



- **$x = (-1)^s \ 1.mmmmmmmmmmmmmmmmmmmmmmm \ 2^{(eeeeeeee)_{10} - 127}$**

- **exemple:**
  - $\pi \approx 3.14159265 = (-1)^0 \ 1.10010010000111111011010 \ 2^{b10000000 - 127}$
  - $+0 = (-1)^0 \ \ 1.00000000000000000000000 \ 2^{00000000-127}$
  - $-0 = (-1)^1 \ \ 1.00000000000000000000000 \ 2^{00000000-127}$
  - signaling NaN: 0x7F800001 sau 0x7FBFFFFF sau între 0xFF800001 și 0xFFBFFFFF
  - quiet NaN: 0x7FC00000 sau 0x7FFFFFFF sau între 0xFFC00000 și 0xFFFFFFFF

mai mult exemple, la Seminarul 0x03

https://en.wikipedia.org/wiki/Floating-point_arithmetic

https://www.h-schmidt.net/FloatConverter/IEEE754.html

CE AM FĂCUT ASTĂZI

- logică secvențială + combinațională, un exemplu
- înmulțirea numerelor întregi binare
- împărțirea numerelor întregi binare
- reprezentarea numerelor în virgulă mobilă
- lucrul cu numerele în virgulă mobilă

DATA VIITOARE ...

- începem să discutăm despre arhitecturi de calcul

- începem să discutăm despre seturi de instrucțiuni

LECTURĂ SUPLIMENTARĂ

- **PH book**
  - 3.3 Multiplication
  - 3.4 Division
  - 3.5 Floating Point

- **Computerphile, Floating Point Numbers, https://www.youtube.com/watch?v=PZRI1IfStY0**
- **Computephile, Floating Point Numbers (Part1: Fp vs Fixed), https://www.youtube.com/watch?v=f4ekifyijlg**
- **Computephile, Floating Point Numbers (Part2: Fp Addition), https://www.youtube.com/watch?v=782QWNOD_Z0**

.

.