Arhitectura sistemelor de calcul

- Prelegerea 10 -

2-DS (Automate)

Ruxandra F. Olimid

Facultatea de Matematică şi Informatică
Universitatea din Bucureşti

Cuprins

2-DS (Automate)

- Introducem încă un *ciclu* la sistemele cu 1 ciclu (1-DS) şi obţinem un grad de autonomie sporit al sistemelor digitale: starea sistemelor *2-DS* poate **evolua** parţial independent faţă de semnalul de intrare, spre deosebire de sistemele 1-DS care prezentau doar o autonomie parţială în sensul păstrării unei anumite stări

- Sistemele *2-DS* (cu 2 cicluri), numite şi *automate* evoluează în funcţie de 2 variabile: de **intrare** şi de **starea internă**

- 2-DS prezintă o autonomie internă bazată pe stări anterioare ale sistemului , deci spaţiul stărilor interne (ex. numărul de stări posibile) are influenţă asupra complexităţii sistemului

- Vom considera doar sisteme **sincrone**, în care orice ciclu se închide peste un 1-DS

2-DS (Automate)

- *Def.:* Un *automat* este un 5-tuplu $A = (Q, X, Y, \delta, \lambda)$, unde:
  - ✓ $Q \neq \emptyset$ este **mulţimea stărilor**
  - ✓ $X \neq \emptyset$ este **mulţimea variabilelor de intrare**
  - ✓ $Y \neq \emptyset$ este **mulţimea variabilelor de ieşire**
  - ✓ $\delta: Q \times X \to Q$ este **funcţia de tranziţie**
  - ✓ $\lambda: Q \times X \to Y$ sau $\lambda: Q \to Y$ este **funcţia de ieşire**

- δ defineşte cum se modifică starea internă, în funcţie de starea în care se găseşte sistemul la un moment dat şi de intrare

- λ defineşte ieşirea, care poate să depindă de intrare ($\lambda: Q \times X \to Y$ pentru *automatul de tip Mealy*) sau nu ($\lambda: Q \to Y$ pentru *automatul de tip Moore*) [info]

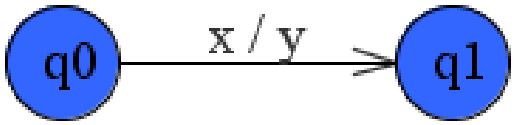
2-DS (Automate)

- Cele 2 tipuri de automate (Mealy şi Moore) sunt echivalente, abstracţie făcând eventual de primul caracter de ieşire

- În general vom lucra cu automate de tip Mealy

- Reprezentăm automatele sub forma unui graf orientat:
  - ✓ nodurile sunt stările (elemenete din Q)
  - ✓ arcele marchează perechi de intrare / ieşire (perechi din $X \times Y$) astfel:

$$\delta(q0, x) = q1$$
$$\lambda(q0, x) = y$$



[JFLAP]

2-DS (Automate)

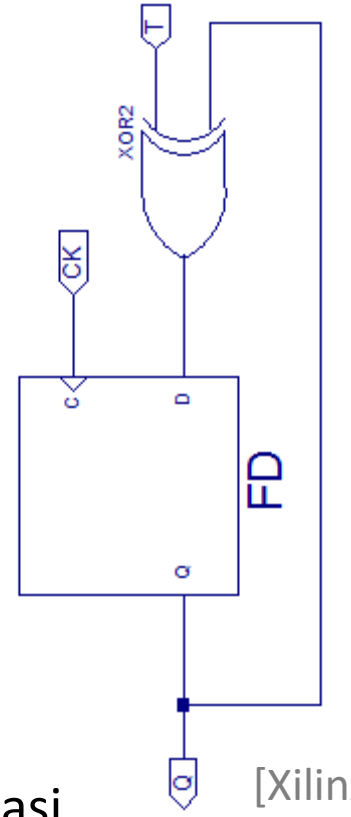
- Un *automat finit* este un automat cu un număr **finit** de stări (i.e. Q este o mulţime finită)

- La intrare se poate aplica un semnal (binar) infinit, dar şi în aceste condiţii automatul prezintă un număr finit (constant) de stări; δ prezintă o **definiţie nerecursivă**

- Pentru un *automat infinit* numărul de stări interne depinde (proporţional) de lungimea intrării. Deci dacă se aplică un semnal (binar) infinit la intrare, atunci numărul de stări devine infinit

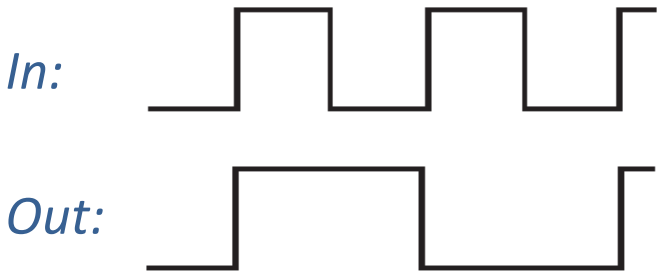
- Automatele defintie recursiv sunt mai simple, în timp ce automatele finite sunt mai complexe.

T-Flip-Flops



[Xilinx - ISE]

- *T-Flip-Flops* este se obţin prin închiderea cu o buclă a unui D-Flip-Flop şi a unei porţi XOR
- T-Flip-Flops au*:*
  - ✓ 2 intrări: *CK (intrarea de ceas)* şi *T (intrarea de date)*
  - ✓ 1 ieşire: *Q (starea sistemului)* (eventual 2 ieşiri, dacă se consideră şi $\bar{Q}$)
- Prezintă următoarea funcţionalitate:
  - ✓ Pentru $T = 0$, starea automatului rămâne aceeaşi
  - ✓ Pentru $T = 1$, starea automatului comută
- Se utilizează ca divizor de frecvenţă:



T-Flip-Flops

- Descrierea ca automat:

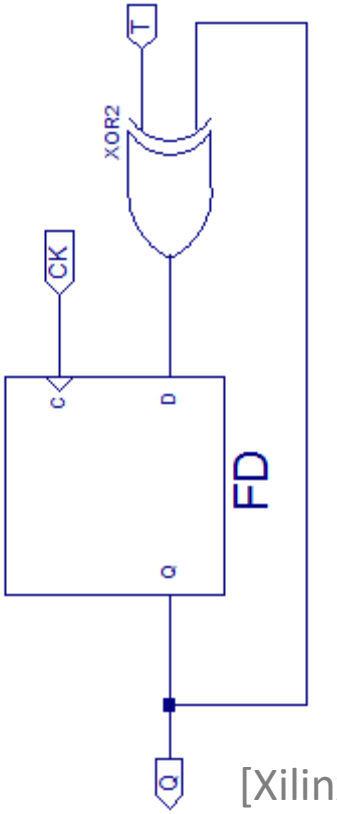
$$Q = X = Y = \{0,1\}$$

Q \ X	0	1
0	0	1
1	1	0

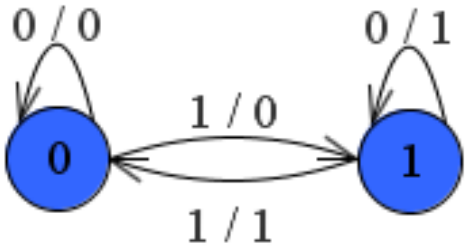
funcţia de tranziţie

Q \ X	0	1
0	0	0
1	1	1

funcţia de ieşire



[Xilinx - ISE]



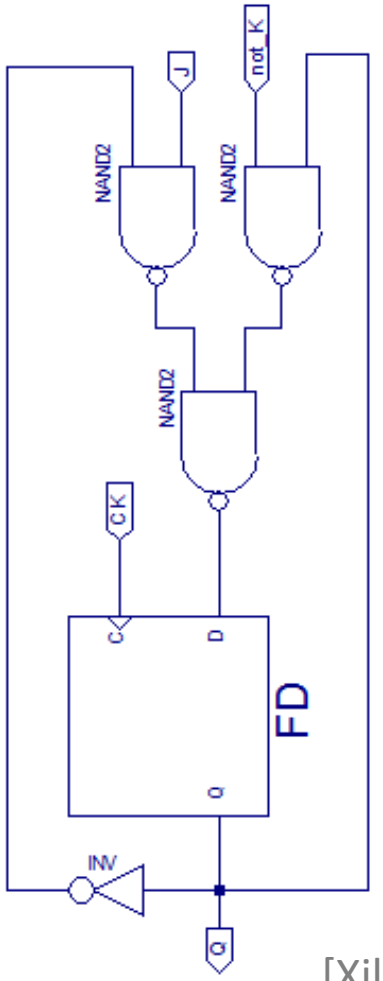
[JFLAP]

JK-Flip-Flops

- *JK-Flip-Flops* este se obţin prin închiderea cu o buclă a unui D-Flip-Flop şi câtorva porţi suplimentare

  [Poarta NOT nu este necesară, dacă considerăm DFF cu 2 ieşiri: Q şi $\bar{Q}$]

- T-Flip-Flops au*:*
  - ✓ 3 intrări: *CK (intrarea de ceas)* şi *J, K (intrările de date)*
  - ✓ 1 ieşire: Q *(starea sistemului)* (eventual 2 ieşiri, dacă se consideră şi $\bar{Q}$)



[Xilinx - ISE]

- Din schemă rezultă următoarea relaţie:

J	0	0	1	1
K	0	1	0	1
D	Q	0	1	$\bar{Q}$

JK-Flip-Flops

- Descrierea ca automat:

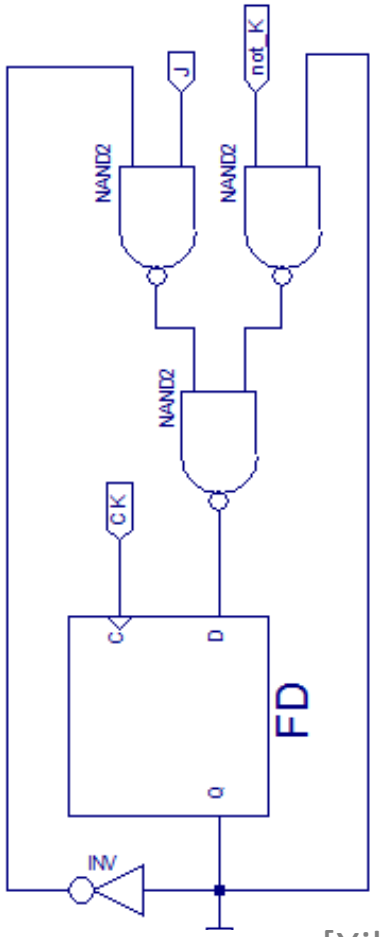
$$Q = Y = \{0,1\};\ X = \{0,1\}^2$$

Q \ X	00	01	10	11
0	0	0	1	1
1	1	0	1	0

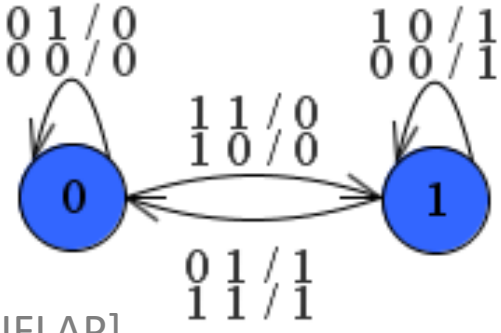
funcţia de tranziţie

Q \ X	00	01	10	11
0	0	0	0	0
1	1	1	1	1

funcţia de ieşire

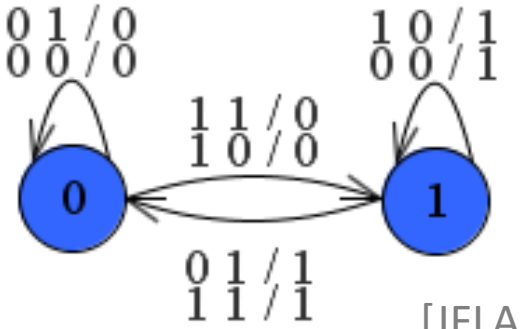


[Xilinx - ISE]



[JFLAP]

JK-Flip-Flops



[JFLAP]

- Prezintă următoarea funcţionalitate:
  - ✓ *(nop)* Pentru $\mathrm{J} = \mathrm{K} = 0$, starea automatului rămâne aceeaşi
  - ✓ *(reset)* Pentru $\mathrm{J} = 0, \mathrm{K} = 1$ starea automatului devine 0
  - ✓ *(set)* Pentru $\mathrm{J} = 1, \mathrm{K} = 0$ starea automatului devine 1
  - ✓ *(switch)* Pentru $\mathrm{J} = 1, \mathrm{K} = 1$ starea automatului comută

- Funcţionalitatea de switch necesită autonomie şi în funcţie de stare (specifică sistemelor 2-DS): sistemul cunoaşte starea anterioară ca să poată comuta

Circuite de înmulţire (32 biţi)

➢ Înmulţirea binară se realizează astfel:

```
   10101x
    1101
--------
   10101
  00000
 10101
10101
--------
100010001
```

Dacă lsb înmulţitorului este:

✓ 1, atunci produsul se iniţializează cu deînmulţitul

✓ 0, atunci produsul se iniţializează cu 0

Pentru fiecare bit al înmulţitorului, cu excepţia lsb:

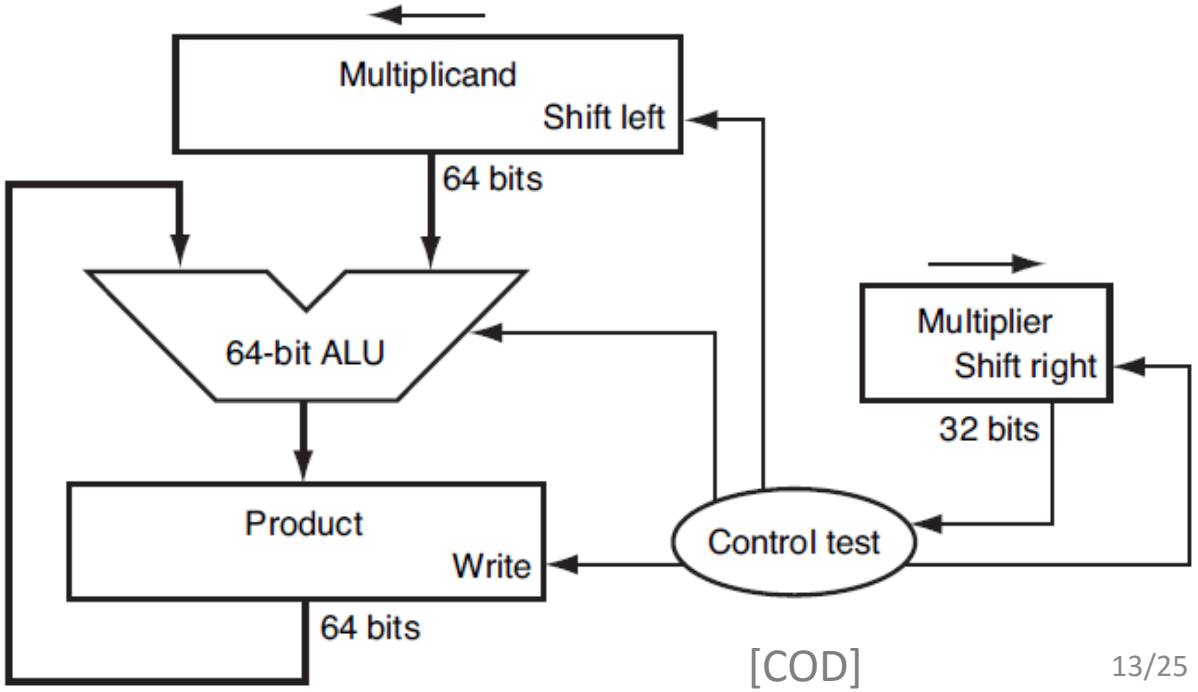
✓ se shiftează deînmulţitul la stânga cu o poziţie

✓ dacă bitul este 1, se adaugă la rezultat deînmulţitul modificat conform pasului anterior

➢ Remarcaţi că produsul necesită n+m biţi, unde n este numărul de biţi ai deînmulţitului şi m este numărul de biţi al înmulţitorului!

Circuite de înmulţire (32 biţi)

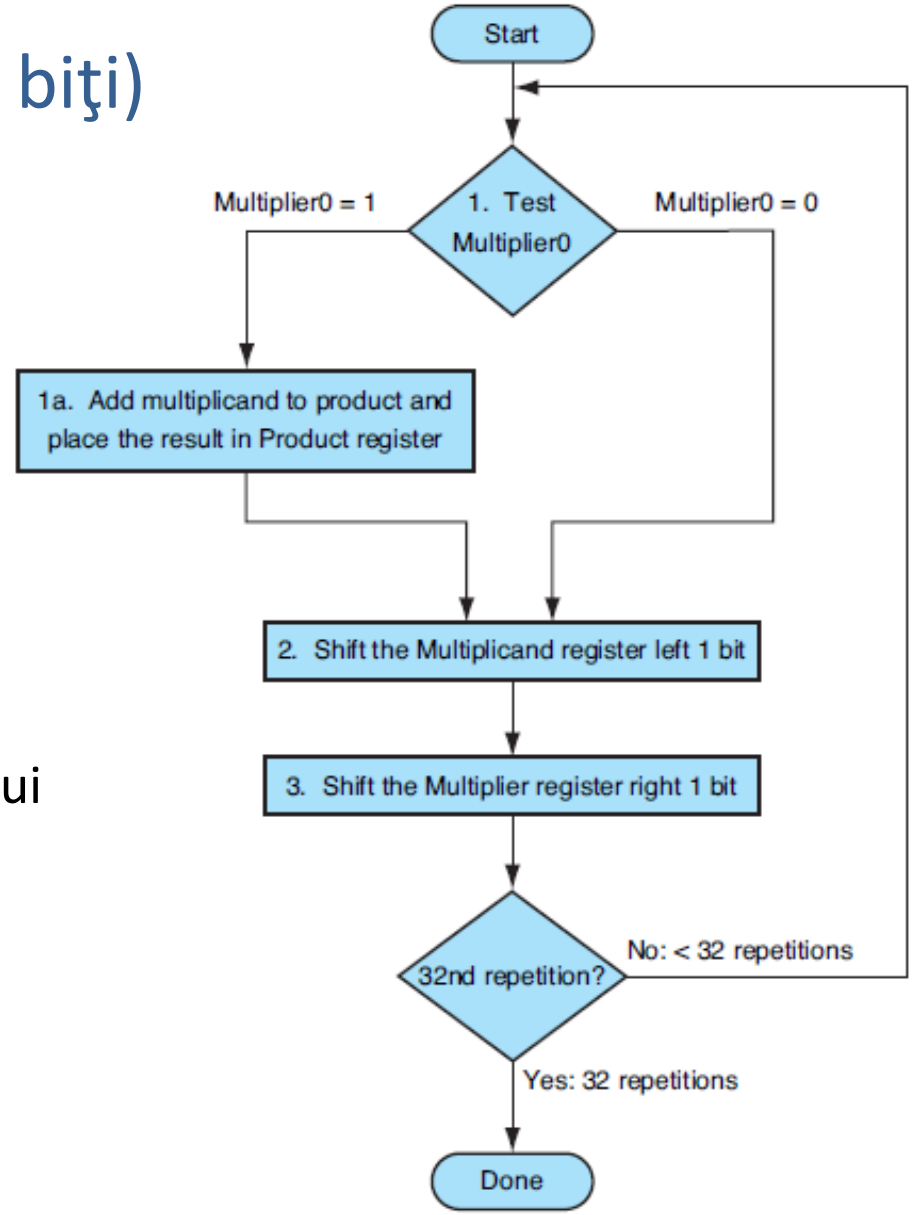
- Înmulţirea pe 32 de biţi necesită:
  - ✓ **2 regiştrii** de 32 de biţi pentru operanzi
  - ✓ **1 registru** de 64 biţi (32 + 32) pentru rezultat
  - ✓ **ALU** pe 64 biţi pentru calculul sumelor intermediare
  - ✓ **1 unitate de control** care dictează când se realizează adunarea, shiftările, scrierea în registrul în care se stochează produsul



Circuite de înmulţire (32 biţi)

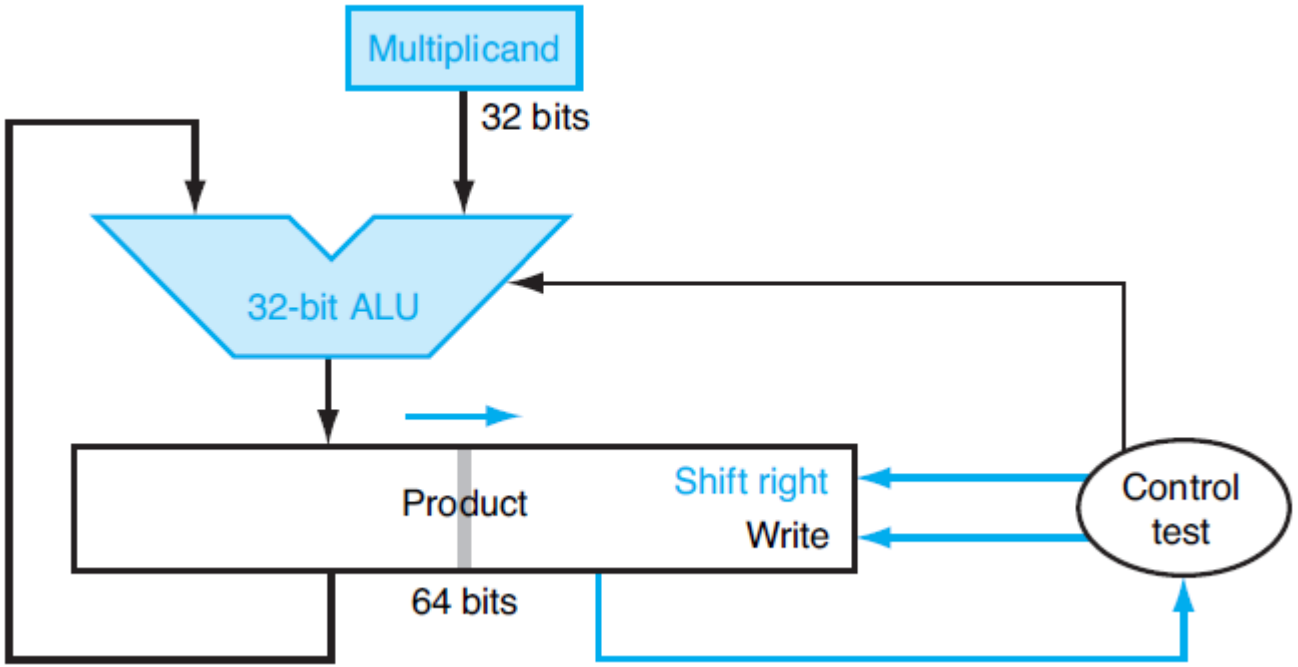
- Algoritmul este reprezentat alăturat în schemă logică

- Denumiri:
  - Multiplier = înmulţitor
  - Multiplicand = deînmulţit
  - Multiplier0 = lsb al înmulţitorului



Circuite de înmulţire (32 biţi)

- O metodă mai eficientă de înmulţire pe 32 de biţi:



[COD]

[Modificările faţă de constructia anterioară sunt marcate prin colorare]

Circuite de înmulţire (32 biţi)

Observaţii:

- Se foloseşte un ALU pe 32 de biţi pentru că la fiecare pas e necesară doar o adunare pe 32 de biţi, dimensiunea înmulţitorului (restul biţilor rămân constanţi sau eventual un bit se adună cu transportul); ieşirea din ALU e scrisă pe primii 32 de biţi ai registrului Product; intrarea în ALU o constituie tot primii 32 de biţi ai registrului Product

- Dispare registrul în care era stocat înmulţitorul, acesta fiind plasat în ultimii 32 de biţi ai registrului Product

- Nu mai este necesară shiftarea la dreapta a deînmulţitului, pentru că se schift-ează la stânga registrul produs (rezultatele parţiale şi înmulţitorul, pentru determinarea a câte un bit)

Circuite de împărţire (32 biţi)

- Împărţirea binară se realizează astfel:

$$100010010 = 10101 \times cat + rest$$

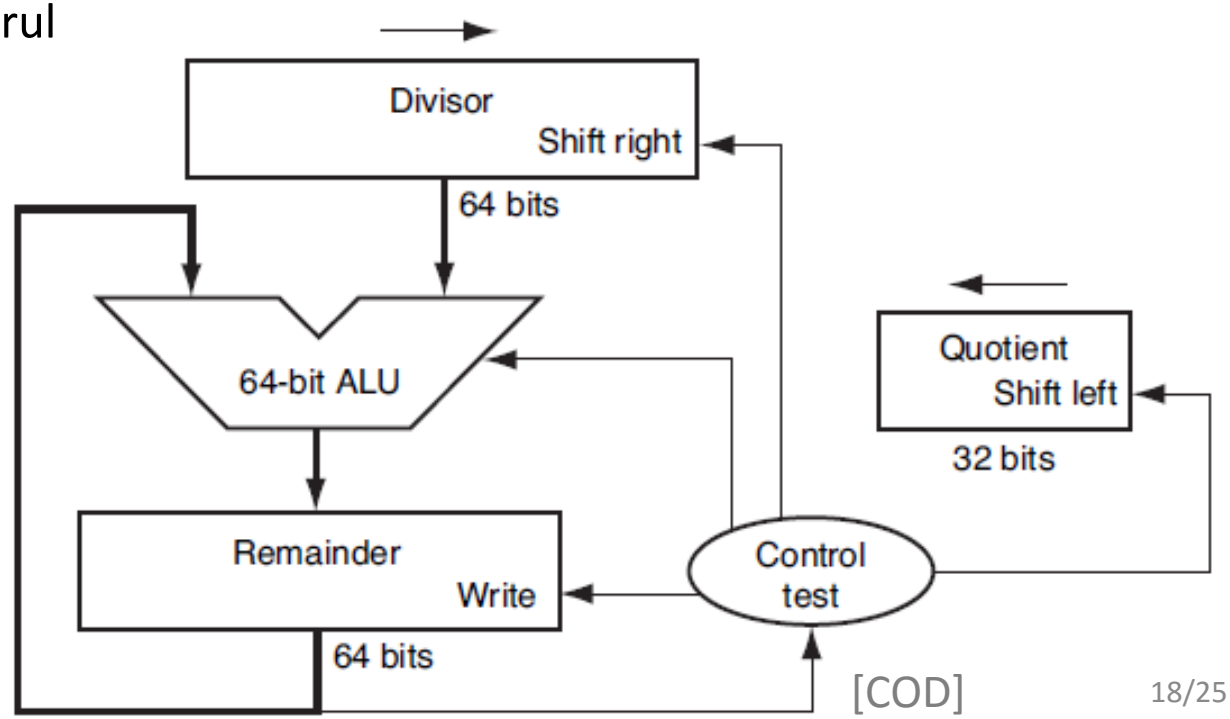
- Pentru determinarea câtului se scade pe rând împărţitorul din deîmpărţit, adăugând 1 la cât pe poziţia corespunzătoare sau 0 dacă nu se poate realiza scăderea şi se mai coboară un bit

- Rezultatul final îl reprezintă restul

```
cat = 1101
 100010010
 -10101
----------
   11010
  -10101
----------
     1011
     10110
    -10101
----------
          1 = rest
```

Circuite de împărţire (32 biţi)

- Înmulţirea pe 32 de biţi necesită:
  - ✓ **1 registru** de 32 de biţi pentru cât
  - ✓ **2 regiştrii** de 64 biţi pentru împărţitor (iniţial plasat în jumătatea stângă) şi în care se va stoca la final restul
  - ✓ **ALU** pe 64 biţi pentru calculul diferenţelor intermediare
  - ✓ **1 unitate de control** care dictează când se realizează scăderea, shiftările, scrierea în registrul în care se stochează restul

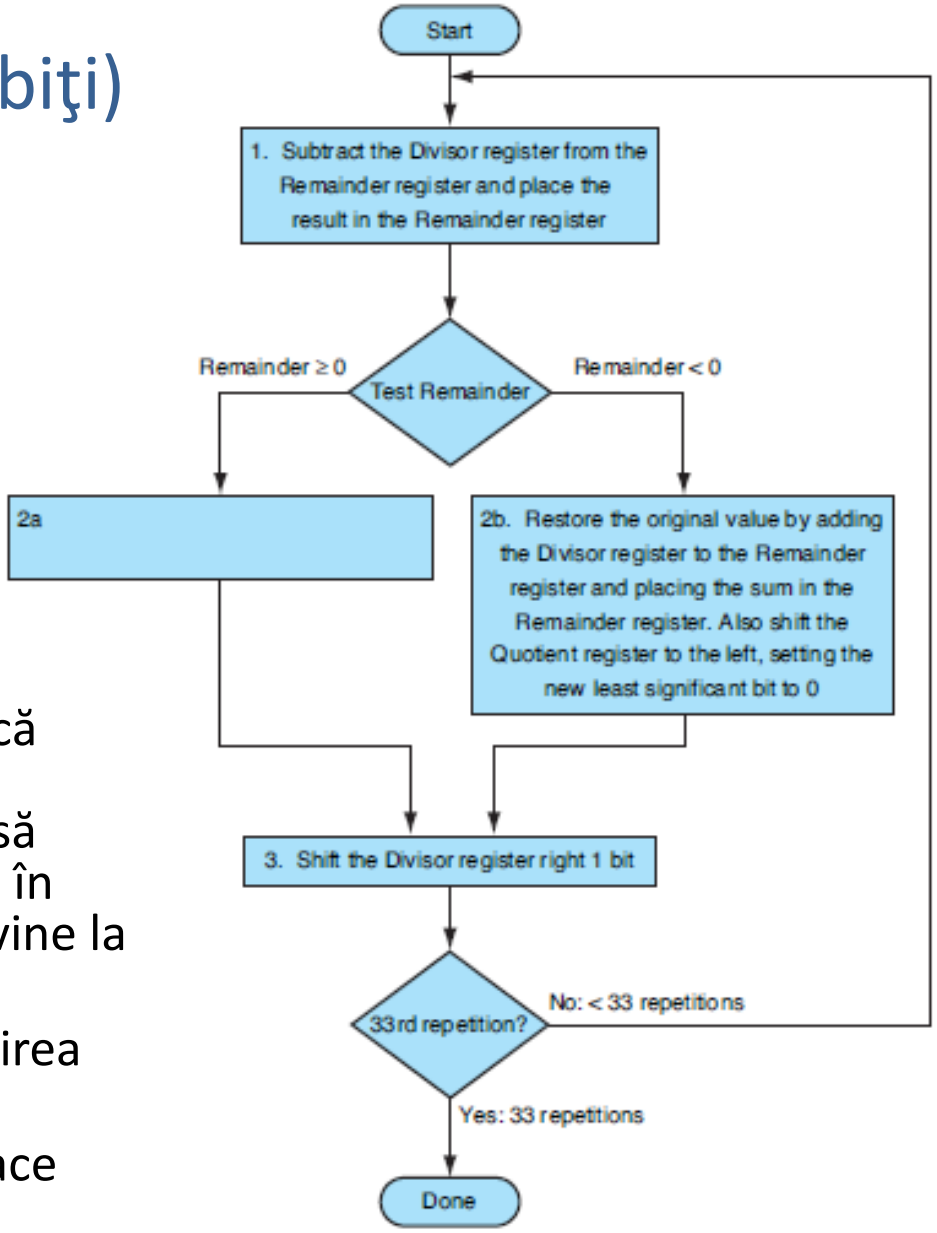


Circuite de împărţire (32 biţi)

- Algoritmul este reprezentat alăturat în schemă logică

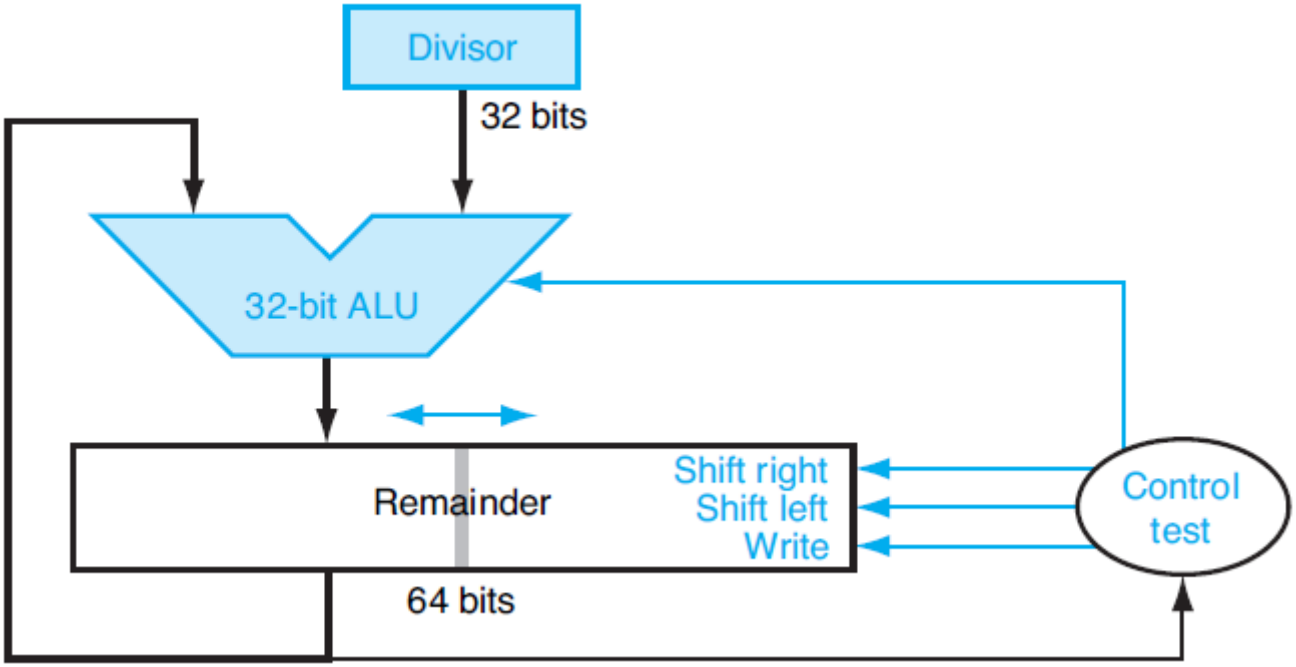
- Denumiri:
  - Divisor = împărţitor
  - Dividend = deîmpărţit

- Observaţii:
  - ✓ sistemul nu poate testa anterior dacă împărţiotorul se cuprinde sau nu în valoarea curentă, aşa încât trebuie să realizeze mereu diferenţa şi în cazul în care obţine un număr negativ se revine la situaţia anterioară
  - ✓ verificarea se realizează folosind ieşirea Zero din ALU
  - ✓ revenirea la situaţia anterioară se face prin adunarea împărţitorului



Circuite de împărţire (32 biţi)

- O metodă mai eficientă de împărţire pe 32 de biţi:



[COD]

[Modificările faţă de constructia anterioară sunt marcate prin colorare]

Circuite de împărţire (32 biţi)

Observaţii:

- Se foloseşte un ALU pe 32 de biţi pentru că la fiecare pas e necesară doar o scădere pe 32 de biţi, dimensiunea împărţitorului; ieşirea din ALU e scrisă pe primii 32 de biţi ai registrului Remainder; intrarea în ALU o constituie tot primii 32 de biţi ai registrului Remainder

- Dispare registrul în care era stocat câtul, acesta fiind plasat în ultimii 32 de biţi ai registrului Remainder

- Registrul Divisor se reduce la 32 de biţi şi nu mai este necesară shiftarea la dreapta a acestuia, dar apare o shiftare la registrul Divisor

Circuite de înmulţire / împărţire (32 biţi)

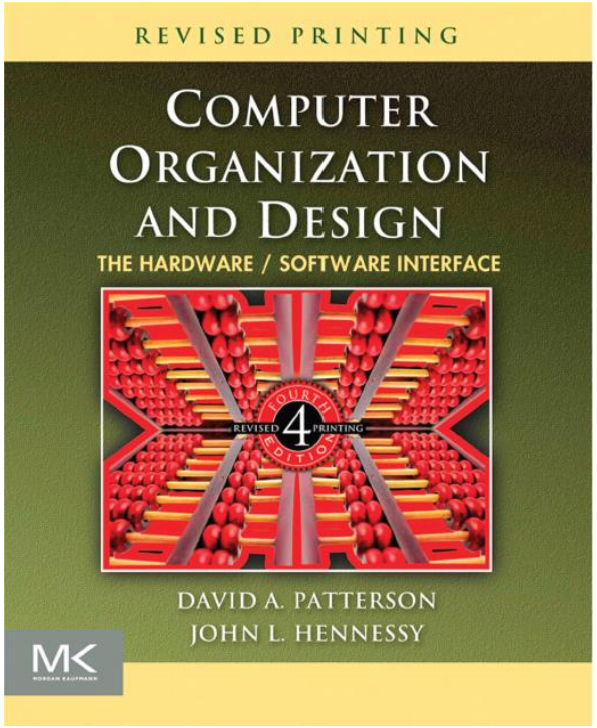
- Ambele construcţii pot fi utilizate pentru calcule cu numere negative; semnul produsului, respectiv al câtului se obţine imediat ca XOR între semnele celor 2 operanzi

- Ambele construcţii sunt 2-DS pentru că închid un ciclu peste un registru (care este 1-DS)

Alte automate

- Exemple de automate interesante sunt:
  - ✓ *Stiva (FIFO)*
  - ✓ *Coada (LIFO)*
  - ✓ *Alte circuite aritmetice:* sumatoare (mai eficiente), Countere (numărătoare), etc.

Referințe bibliografice



[AAT] A. Atanasiu, Arhitectura calculatorului

[COD] D. Patterson and J. Hennessy, Computer Organisation and Design

Schemele [Xilinx - ISE] au fost realizate folosind
http://www.xilinx.com/tools/projnav.htm

Grafurile [JFLAP] au fost realizate folosind
http://www.jflap.org/

Automatele Mealy şi Moore

- *George H. Mealy* (1927 - 2010)
  - ✓ matematician şi informatician american
  - ✓ a introdus conceptul care îi poartă numele (automat Mealy) în lucrarea *A Method for Synthesizing Sequential Circuits* (1955)

- *Edward F. Moore* (1925 - 2003)
  - ✓ matematician şi informatician american
  - ✓ a introdus conceptul care îi poartă numele (automat Moore) în lucrarea *Gedanken-experiments on Sequential Machines* (1956)

[inapoi]