



Arhitectura sistemelor de calcul

- Prelegerea 10 -
2-DS (Automate)

Ruxandra F. Olimid

Facultatea de Matematică și Informatică
Universitatea din București

Cuprins

1. Definiție
2. T-Flip-Flops
3. JK-Flip-Flops
4. Circuite de înmulțire
5. Circuite de împărțire

2-DS (Automate)

- Introducem încă un *ciclu* la sistemele cu 1 ciclu (1-DS) și obținem un grad de autonomie sporit al sistemelor digitale: starea sistemelor *2-DS* poate **evolua** parțial independent față de semnalul de intrare, spre deosebire de sistemele 1-DS care prezentau doar o autonomie parțială în sensul păstrării unei anumite stări
- Sistemele *2-DS* (cu 2 cicluri), numite și *automate* evoluează în funcție de 2 variabile: de **intrare** și de **starea internă**
- 2-DS prezintă o autonomie internă bazată pe stări anterioare ale sistemului, deci spațiul stărilor interne (ex. numărul de stări posibile) are influență asupra complexității sistemului
- Vom considera doar sisteme **sincrone**, în care orice ciclu se încheie peste un 1-DS

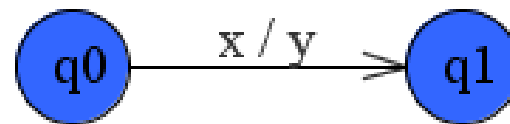
2-DS (Automate)

- *Def.:* Un *automat* este un 5-tuplu $A = (Q, X, Y, \delta, \lambda)$, unde:
 - ✓ $Q \neq \emptyset$ este **mulțimea stărilor**
 - ✓ $X \neq \emptyset$ este **mulțimea variabilelor de intrare**
 - ✓ $Y \neq \emptyset$ este **mulțimea variabilelor de ieșire**
 - ✓ $\delta: Q \times X \rightarrow Q$ este **funcția de tranziție**
 - ✓ $\lambda: Q \times X \rightarrow Y$ sau $\lambda: Q \rightarrow Y$ este **funcția de ieșire**
- δ definește cum se modifică starea internă, în funcție de starea în care se găsește sistemul la un moment dat și de intrare
- λ definește ieșirea, care poate să depindă de intrare ($\lambda: Q \times X \rightarrow Y$ pentru *automatul de tip Mealy*) sau nu ($\lambda: Q \rightarrow Y$ pentru *automatul de tip Moore*) [info](#)

2-DS (Automate)

- Cele 2 tipuri de automate (Mealy și Moore) sunt echivalente, abstracție făcând eventual de primul caracter de ieșire
- În general vom lucra cu automate de tip Mealy
- Reprezentăm automatele sub forma unui graf orientat:
 - ✓ nodurile sunt stările (elemente din Q)
 - ✓ arcele marchează perechi de intrare / ieșire (perechi din $X \times Y$) astfel:

$$\begin{aligned}\delta(q_0, x) &= q_1 \\ \lambda(q_0, x) &= y\end{aligned}$$



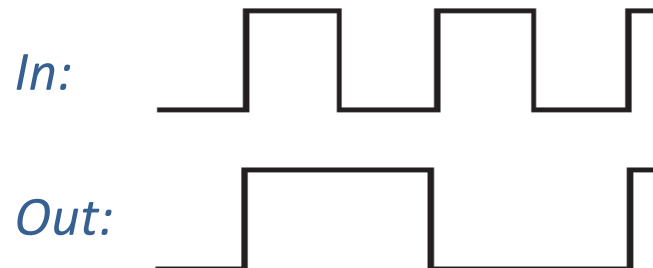
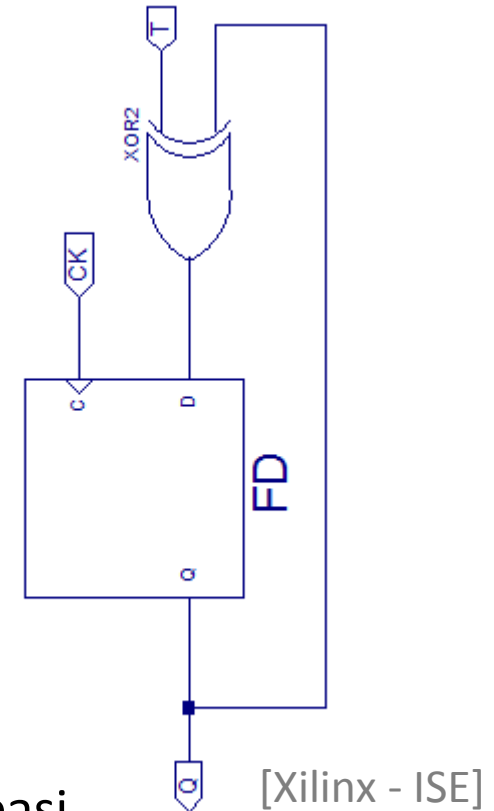
[JFLAP]

2-DS (Automate)

- Un *automat finit* este un automat cu un număr **finit** de stări (i.e. Q este o mulțime finită)
- La intrare se poate aplica un semnal (binar) infinit, dar și în aceste condiții automatul prezintă un număr finit (constant) de stări; δ prezintă o **definiție nerecursivă**
- Pentru un *automat infinit* numărul de stări interne depinde (proporțional) 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

- *T-Flip-Flops* este se obțin prin închiderea cu o buclă a unui D-Flip-Flop și a unei porți XOR
- T-Flip-Flops au:
 - ✓ 2 intrări: *CK (intrarea de ceas)* și *T (intrarea de date)*
 - ✓ 1 ieșire: *Q (starea sistemului)* (eventual 2 ieșiri, dacă se consideră și \bar{Q})
- Prezintă următoarea funcționalitate:
 - ✓ Pentru $T = 0$, starea automatului rămâne aceeași
 - ✓ Pentru $T = 1$, starea automatului comută
- Se utilizează ca divizor de frecvență:



T-Flip-Flops

- Descrierea ca automat:

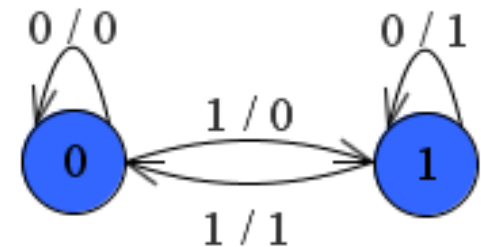
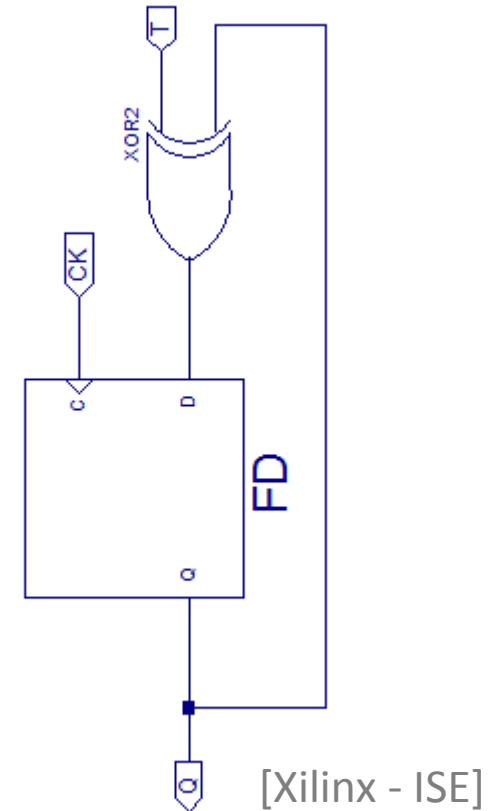
$$Q = X = Y = \{0,1\}$$

		X	
		0	1
Q	0	0	1
	1	1	0

funcția de tranziție

		X	
		0	1
Q	0	0	0
	1	1	1

funcția de ieșire



[JFLAP]

JK-Flip-Flops

- *JK-Flip-Flops* este se obțin prin închiderea cu o buclă a unui D-Flip-Flop și câtorva porți suplimentare

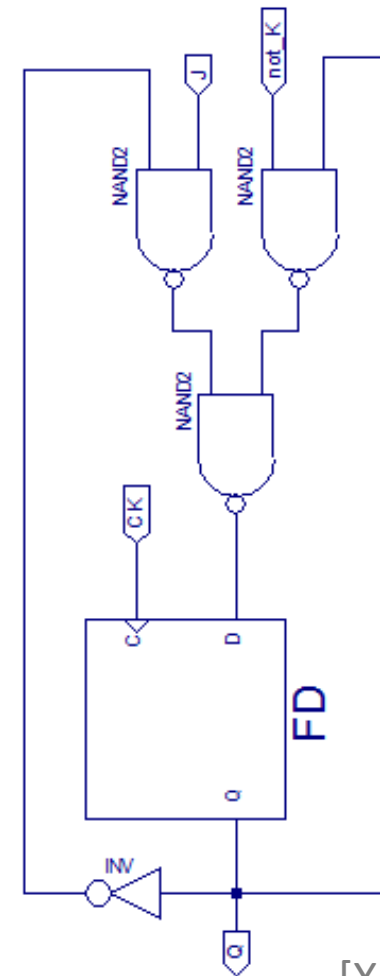
[Poarta NOT nu este necesară, dacă considerăm DFF cu 2 ieșiri: Q și \bar{Q}]

- T-Flip-Flops au:

- ✓ 3 intrări: *CK (intrarea de ceas)* și *J, K (intrările de date)*
- ✓ 1 ieșire: *Q (starea sistemului)* (eventual 2 ieșiri, dacă se consideră și \bar{Q})

- Din schemă rezultă următoarea relație:

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



[Xilinx - ISE]

JK-Flip-Flops

➤ Descrierea ca automat:

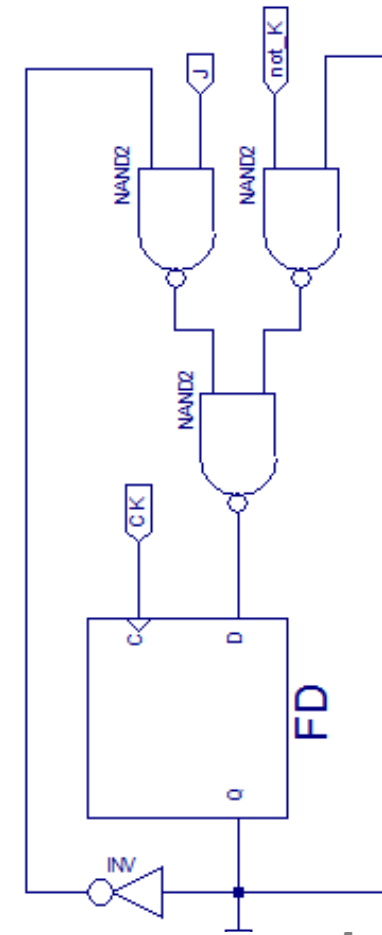
$$Q = Y = \{0,1\}; X = \{0,1\}^2$$

δ	X			
	00	01	10	11
0	0	0	1	1
1	1	0	1	0

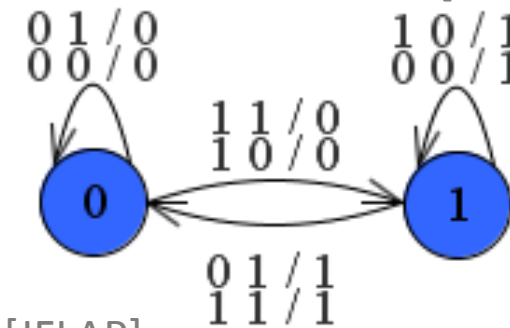
funcția de tranziție

λ	X			
	00	01	10	11
0	0	0	0	0
1	1	1	1	1

funcția de ieșire

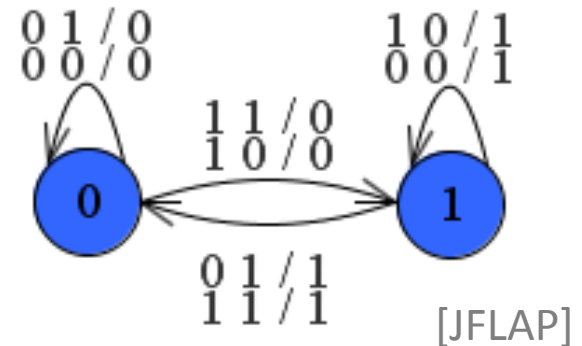


[Xilinx - ISE]



[JFLAP]

JK-Flip-Flops



- Prezintă următoarea funcționalitate:
 - ✓ *(nop)* Pentru $J = K = 0$, starea automatului rămâne aceeași
 - ✓ *(reset)* Pentru $J = 0, K = 1$ starea automatului devine 0
 - ✓ *(set)* Pentru $J = 1, K = 0$ starea automatului devine 1
 - ✓ *(switch)* Pentru $J = 1, K = 1$ starea automatului comută
- Funcționalitatea de switch necesită autonomie și în funcție de stare (specifică sistemelor 2-DS): sistemul cunoaște starea anterioară ca să poată comuta

Circuite de înmulțire (32 biți)

- Înmulțirea binară se realizează astfel:

Dacă lsb înmulțitorului este:

- ✓ 1, atunci produsul se inițializează cu deînmulțitul
- ✓ 0, atunci produsul se inițializează cu 0

Pentru fiecare bit al înmulțitorului, cu excepția lsb:

- ✓ se shiftează deînmulțitul la stânga cu o poziție
- ✓ dacă bitul este 1, se adaugă la rezultat deînmulțitul modificat conform pasului anterior

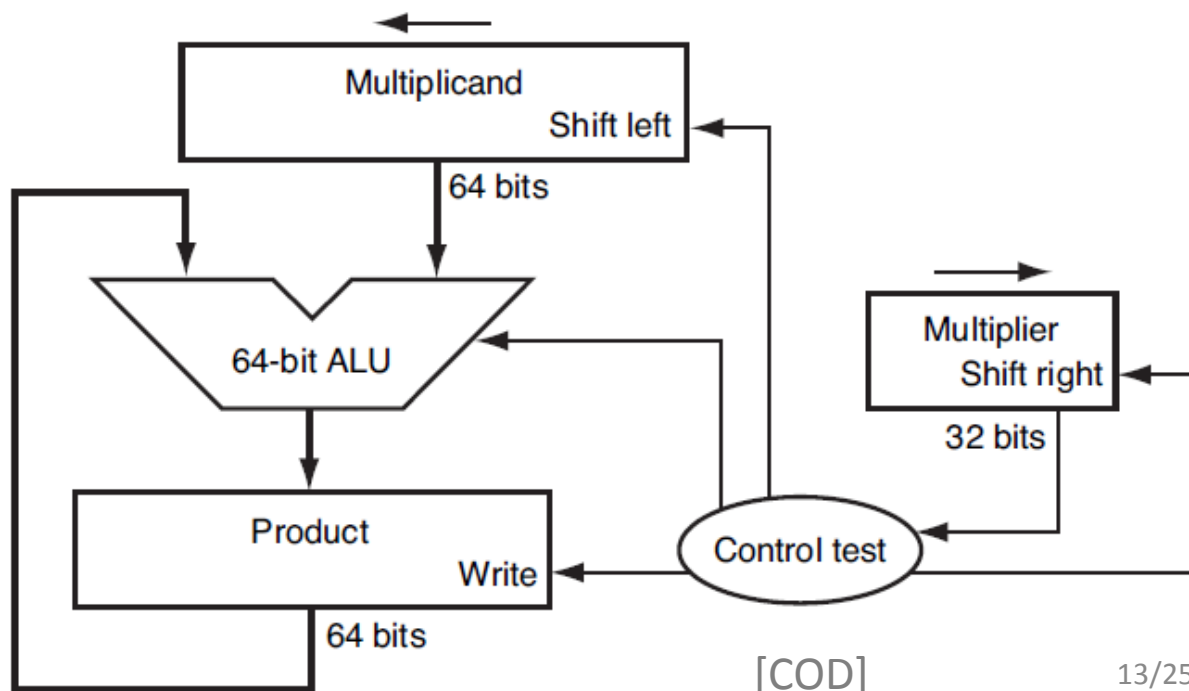
- Remarcați că produsul necesită $n+m$ biți, unde n este numărul de biți ai deînmulțitului și m este numărul de biți al înmulțitorului!

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

Circuite de înmulțire (32 biți)

➤ Înmulțirea pe 32 de biți necesită:

- ✓ **2 regiștrii** de 32 de biți pentru operanzi
- ✓ **1 registru** de 64 biți (32 + 32) pentru rezultat
- ✓ **ALU** pe 64 biți 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 înmulțire (32 biți)

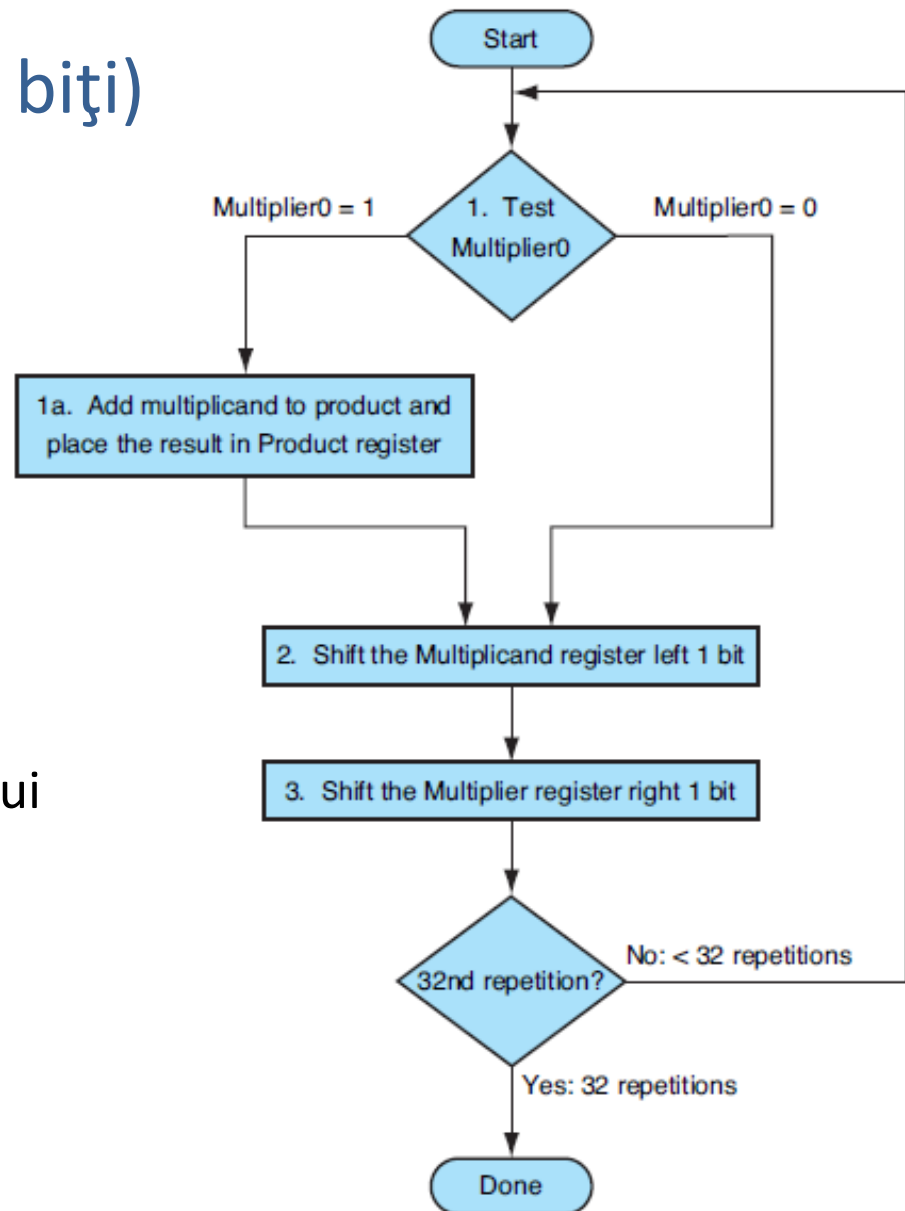
➤ Algoritmul este reprezentat alăturat în schemă logică

➤ Denumiri:

Multiplier = înmulțitor

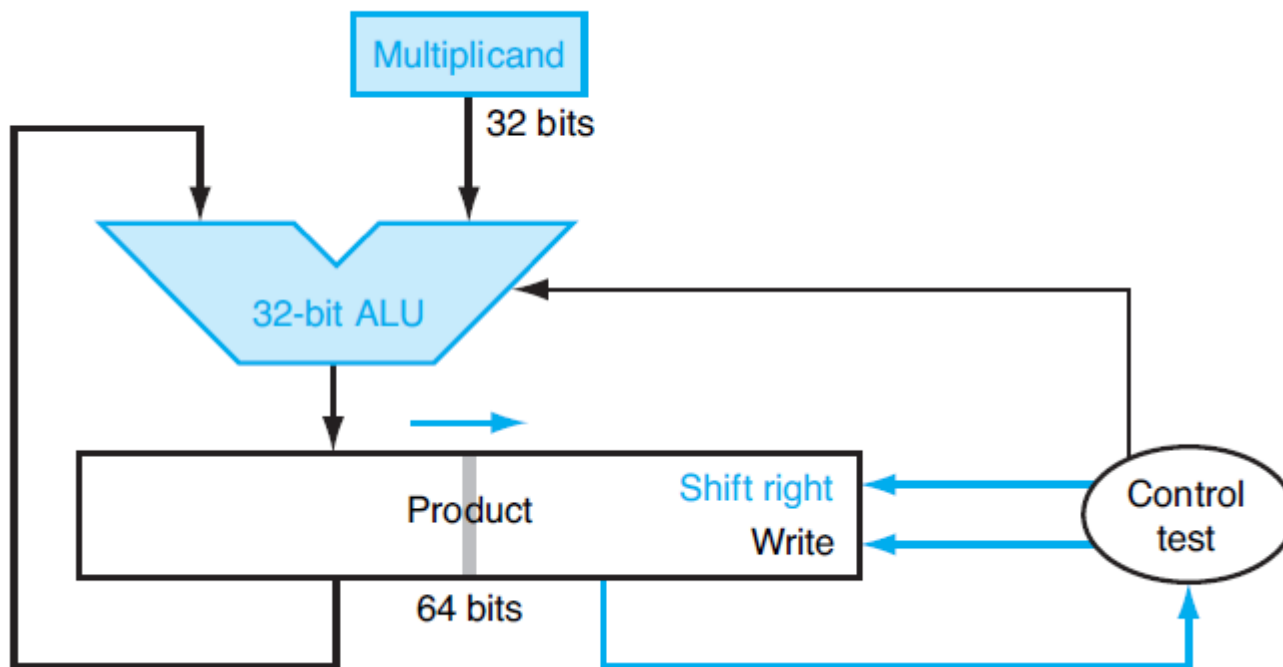
Multiplicand = deînmulțit

Multiplier0 = lsb al înmulțitorului



Circuite de înmulțire (32 biți)

- O metodă mai eficientă de înmulțire pe 32 de biți:



[COD]

Circuite de înmulțire (32 biți)

Observații:

- Se folosește un ALU pe 32 de biți pentru că la fiecare pas e necesară doar o adunare pe 32 de biți, dimensiunea înmulțitorului (restul biților rămân constanți sau eventual un bit se adună cu transportul); ieșirea din ALU e scrisă pe primii 32 de biți ai registrului Product; intrarea în ALU o constituie tot primii 32 de biți ai registrului Product
- Dispare registrul în care era stocat înmulțitorul, acesta fiind plasat în ultimii 32 de biți ai registrului Product
- Nu mai este necesară shiftarea la stânga a deînmulțitului, pentru că se shift-ează la dreapta registrul produs (rezultatele parțiale și înmulțitorul, pentru determinarea a câte un bit)

Circuite de împărțire (32 biți)

- Împărțirea binară se realizează astfel:

$$100010010 = 10101 \times \text{cat} + \text{rest}$$

- Pentru determinarea câtului se scade pe rând împărțitorul din deîmpărțit, adăugând 1 la cât pe poziția corespunzătoare sau 0 dacă nu se poate realiza scăderea și se mai coboară un bit

- Rezultatul final îl reprezintă restul

```
cat = 1101
100010010
-10101
```

```
-----
      11010
      -10101
```

```
-----
          1011
          10110
          -10101
```

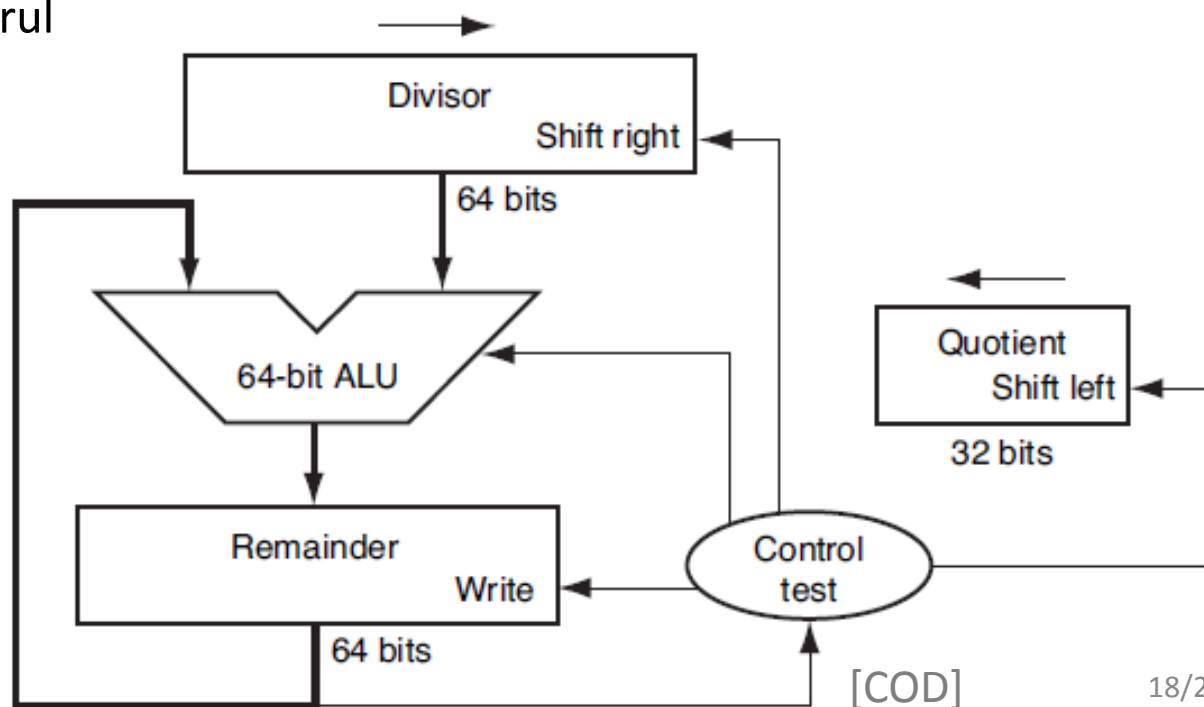
```
-----
                        1 = rest
```

Circuite de împărțire (32 biți)

➤ Înmulțirea pe 32 de biți necesită:

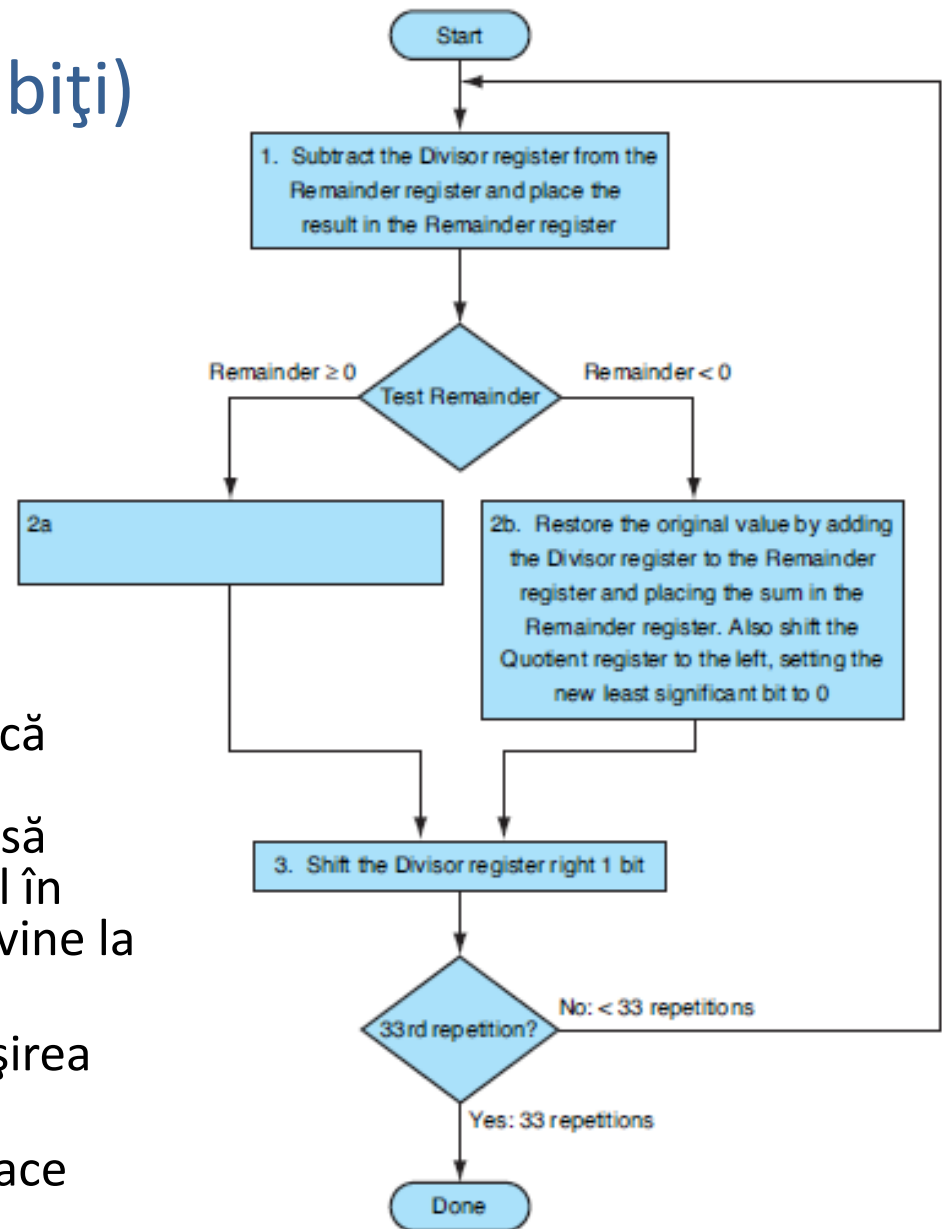
- ✓ **1 registru** de 32 de biți pentru cât
- ✓ **2 regiștrii** de 64 biți pentru împărțitor (inițial plasat în jumătatea stângă) și în care se va stoca la final restul
- ✓ **ALU** pe 64 biți pentru calculul diferențelor intermediare
- ✓ **1 unitate de control** care dictează când se realizează scăderea, shiftările, scrierea în registrul

în care se
stochează
restul



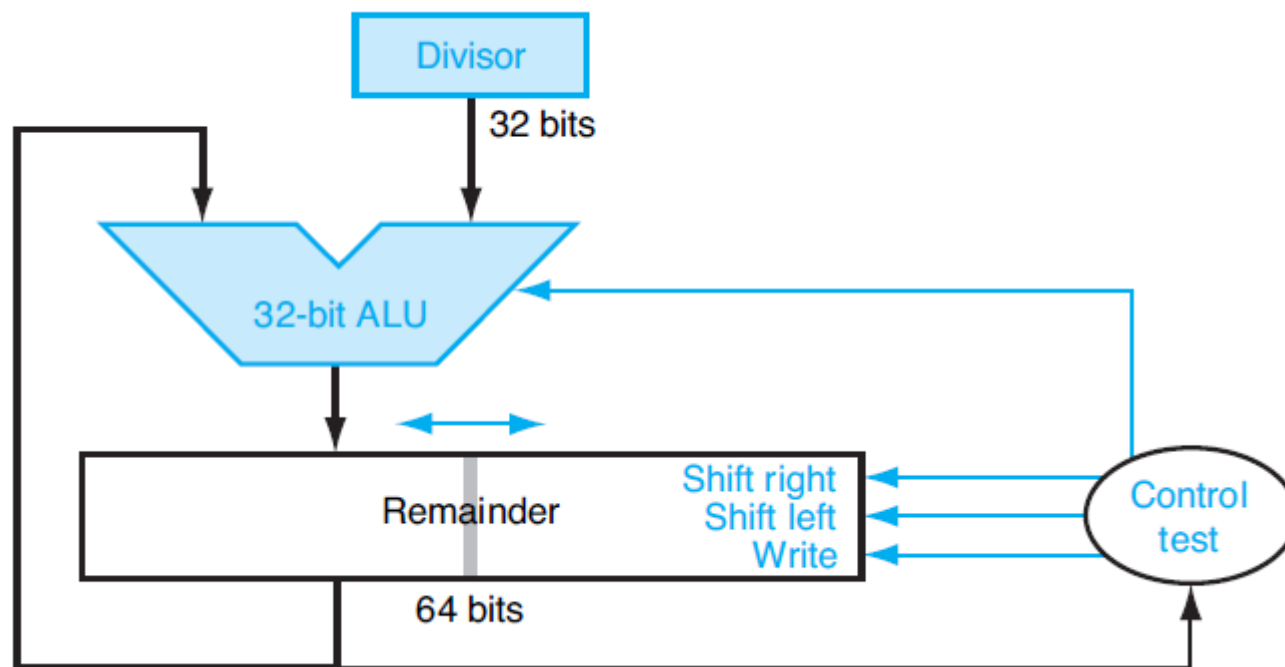
Circuite de împărțire (32 biți)

- Algoritmul este reprezentat alăturat în schemă logică
- Denumiri:
 - Divisor = împărțitor
 - Dividend = deîmpărțit
- Observații:
 - ✓ sistemul nu poate testa anterior dacă împărțitorul se cuprinde sau nu în valoarea curentă, așa încât trebuie să realizeze mereu diferența și în cazul în care obține un număr negativ se revine la situația anterioară
 - ✓ verificarea se realizează folosind ieșirea Zero din ALU
 - ✓ revenirea la situația anterioară se face prin adunarea împărțitorului



Circuite de împărțire (32 biți)

- O metodă mai eficientă de împărțire pe 32 de biți:



[COD]

Circuite de împărțire (32 biți)

Observații:

- Se folosește un ALU pe 32 de biți pentru că la fiecare pas e necesară doar o scădere pe 32 de biți, dimensiunea împărțitorului; ieșirea din ALU e scrisă pe primii 32 de biți ai registrului Remainder; intrarea în ALU o constituie tot primii 32 de biți ai registrului Remainder
- Dispare registrul în care era stocat câtul, acesta fiind plasat în ultimii 32 de biți ai registrului Remainder
- Registrul Divisor se reduce la 32 de biți și nu mai este necesară shiftarea la dreapta a acestuia, dar apare o shiftare la registrul Divisor

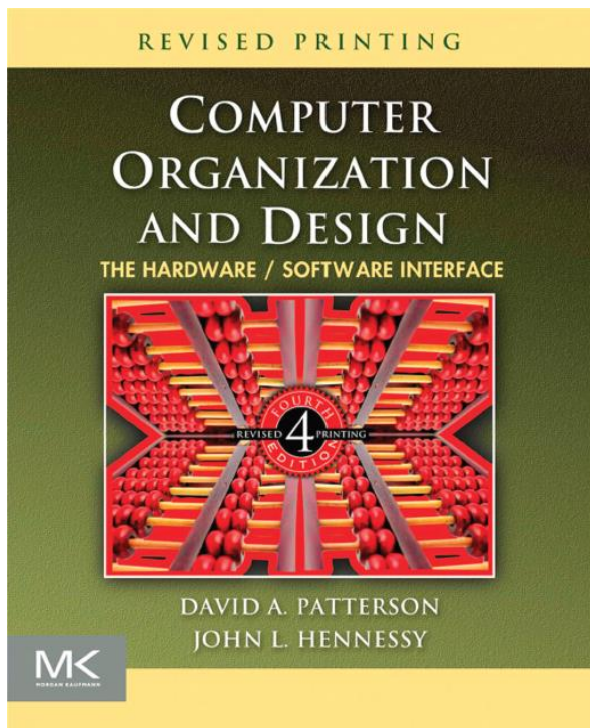
Circuite de înmulțire / împărțire (32 biți)

- Ambele construcții pot fi utilizate pentru calcule cu numere negative; semnul produsului, respectiv al câtului se obține imediat ca XOR între semnele celor 2 operanzi
- Ambele construcții sunt 2-DS pentru că închid un ciclu peste un registru (care este 1-DS)

Alte automate

- Exemple de automate interesante sunt:
 - ✓ *Stiva (FIFO)*
 - ✓ *Cooda (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 și Moore

➤ *George H. Mealy* (1927 - 2010)

- ✓ matematician și 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 și informatician american
- ✓ a introdus conceptul care îi poartă numele (automat Moore) în lucrarea *Gedanken-experiments on Sequential Machines* (1956)