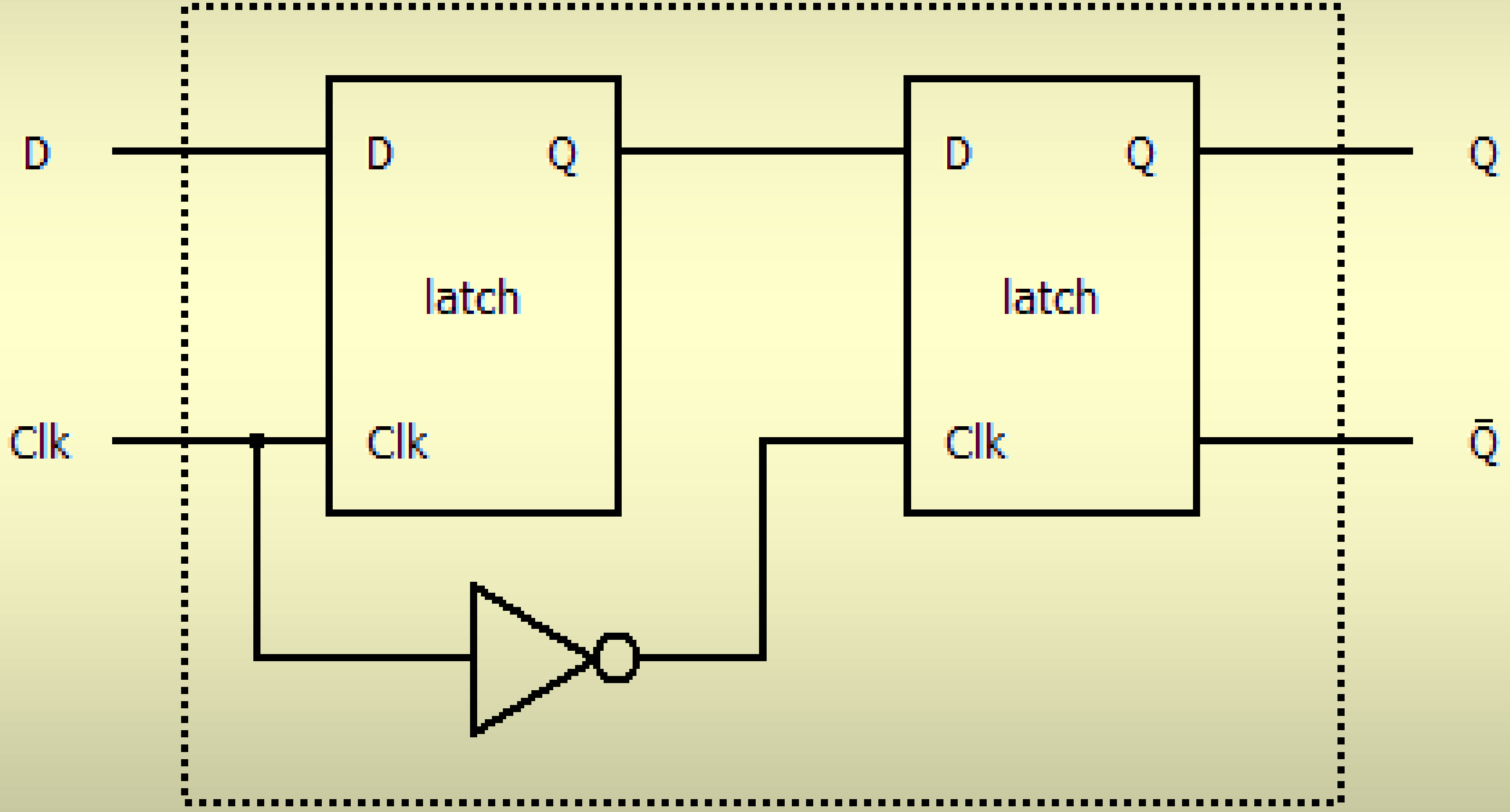
Flip-flop

- intrările sunt luate în considerare doar pe frontul crescător (sau descrescător) al semnalului de ceas
- cum se poate obţine un flip-flop
  - electronic - derivarea semnalului de ceas
  - utilizând circuite latch → circuite master-slave

Flip-flop master-slave D



Latch vs. flip-flop

- fiecare categorie are utilitatea sa
- circuitele flip-flop - utilizate pentru comanda sistemelor digitale
  - frontul semnalului de ceas este foarte scurt comparativ cu perioada ceasului
  - exact un pas în evoluţia sistemului într-o perioadă de ceas
- circuitele latch - sisteme asincrone

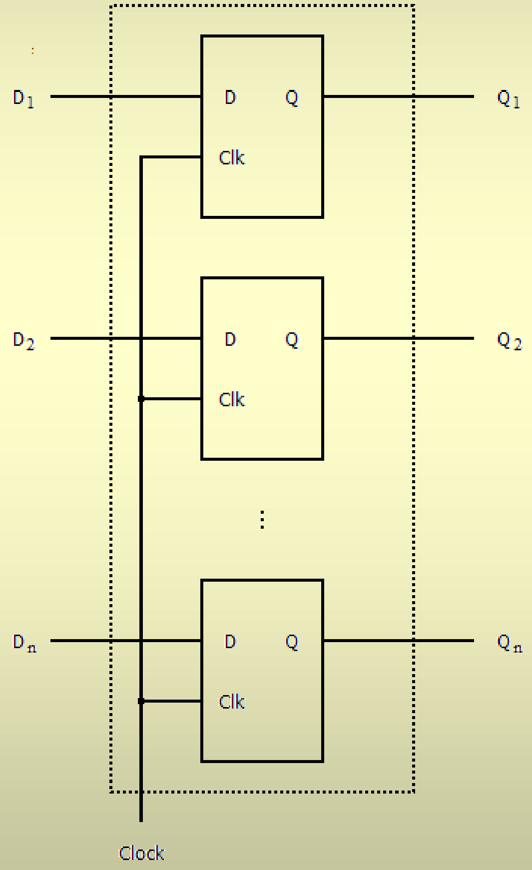
III.2. Circuite secvenţiale complexe

Regiştri

- un circuit bistabil controlează un singur bit
  - nu foarte util în practică
- putem utiliza mai mulţi bistabili simultan
  - toţi primind aceeași comandă
  - un asemenea circuit se numește registru
- tipuri de regiştri
  - paraleli
  - cu deplasare (seriali)

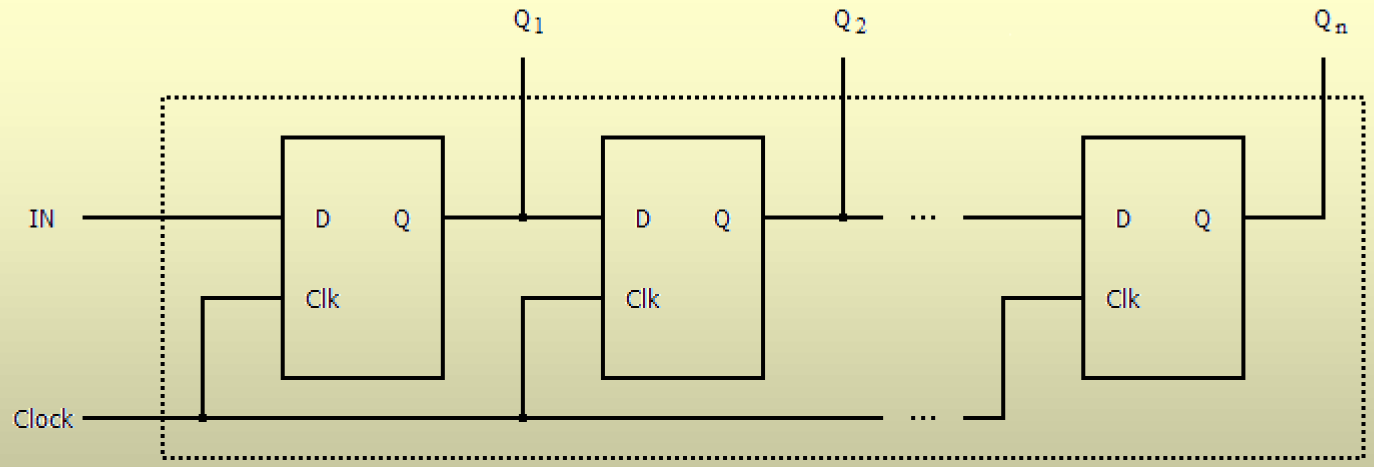
Registru paralel

- implementare cu bistabili D
  - pot fi latch sau flip-flop, după necesităţi
- aceeași comandă (ceas)
  - toţi bistabilii se modifică la aceleași momente
- extinderea bistabilului

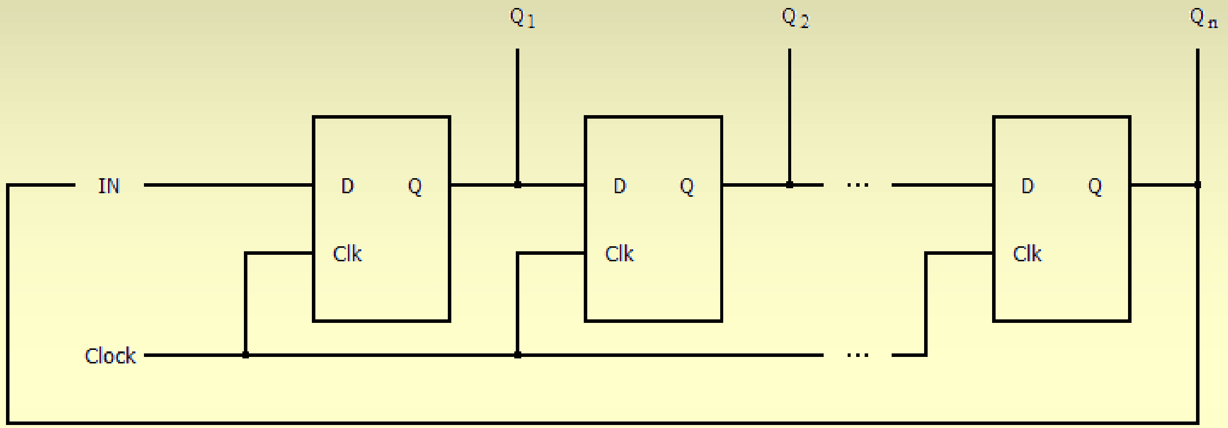


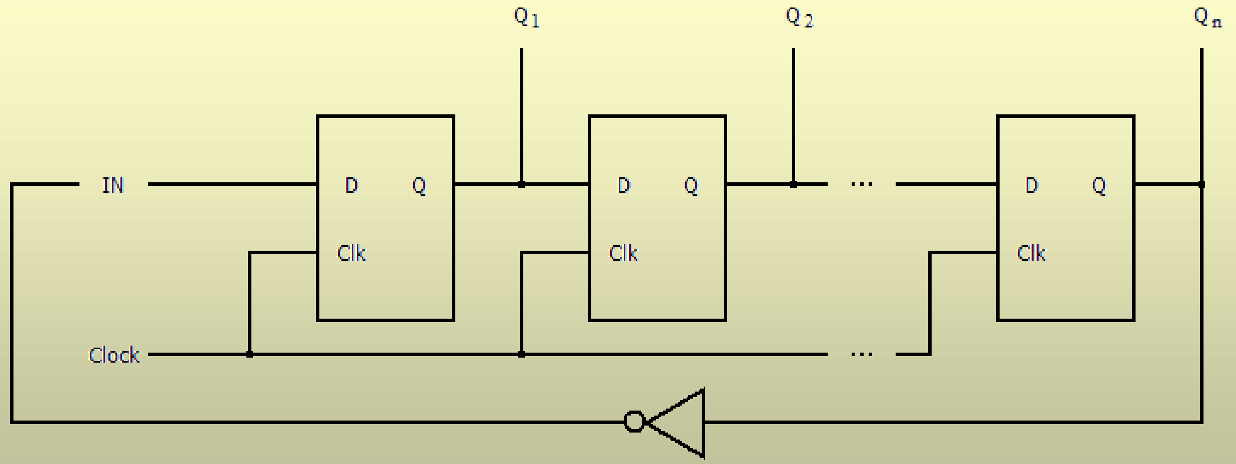
Registru cu deplasare (clasic)

- memorează ultimele *n* valori de la intrare
- poate fi implementat doar cu flip-flop
  - temă: de ce?



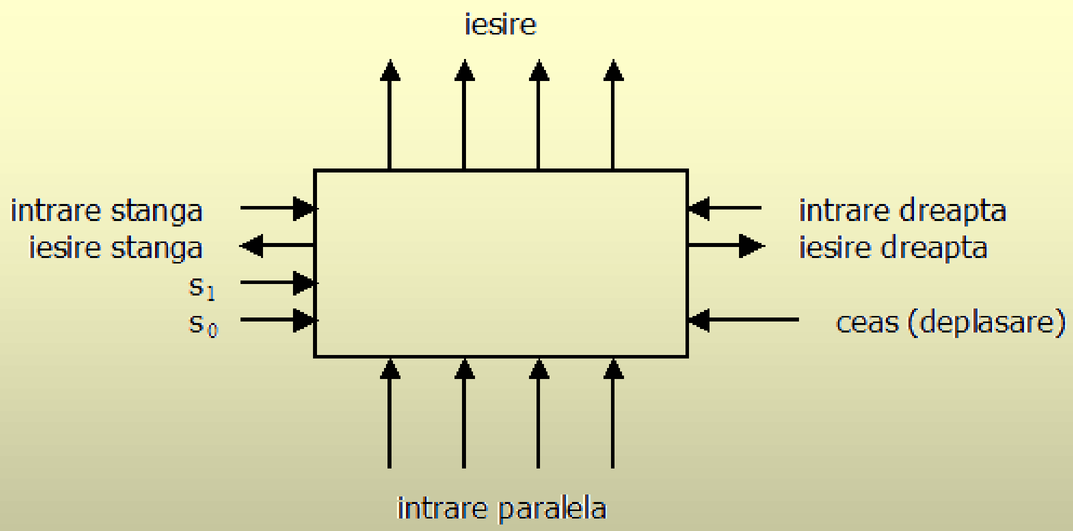
Alţi regiştri cu deplasare





Registru universal

- intrări și ieşiri seriale sau paralele
- deplasare spre stânga sau spre dreapta
- pot fi folosite cele care sunt necesare la un moment dat



s_0	s_1	funcţie
0	0	nemodificat
0	1	deplasare dreapta
1	0	deplasare stânga
1	1	încărcare paralelă

Proiectarea unui circuit secvenţial

- mașină cu număr finit de stări (automat)

1. stabilirea stărilor prin care trece circuitul
2. stabilirea tranziţiilor între stări
   - starea următoare și ieșirile în funcţie de intrări şi de starea curentă
3. codificarea stărilor
   - pe numărul de biți necesar
4. scrierea tabelului de adevăr pentru tranziții

Proiectarea unui circuit secvenţial

5. minimizare
6. implementare
   - starea este memorată prin circuite bistabile
   - partea combinațională - conform minimizării
     - intrările părții combinaționale (starea curentă) se preiau de la ieșirile circuitelor bistabile și de la variabilele de intrare
     - ieșirile părții combinaționale (starea următoare) se aplică la intrările circuitelor bistabile

Numărătorul (contorul) binar

- reţine la fiecare moment un număr pe *n* biţi
- la fiecare "bătaie" a ceasului - incrementare
  - poate fi și decrementare
  - după valoarea maximă urmează din nou 0
  - nu are intrări, doar variabile de stare
    - care reţin de fapt numărul curent
  - ieșirile sunt identice cu variabilele de stare

Exemplu: n=4

starea curentă				starea următoare				starea curentă				starea următoare			
q_3	q_2	q_1	q_0	d_3	d_2	d_1	d_0	q_3	q_2	q_1	q_0	d_3	d_2	d_1	d_0
0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1
0	0	0	1	0	0	1	0	1	0	0	1	1	0	1	0
0	0	1	0	0	0	1	1	1	0	1	0	1	0	1	1
0	0	1	1	0	1	0	0	1	0	1	1	1	1	0	0
0	1	0	0	0	1	0	1	1	1	0	0	1	1	0	1
0	1	0	1	0	1	1	0	1	1	0	1	1	1	1	0
0	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0

Exemplu: *n*=4

- prin minimizare se obţin ecuaţiile

$$d_0 = \overline{q_0} = q_0 \oplus 1$$

$$d_1 = \overline{q_1} \cdot q_0 + q_1 \cdot \overline{q_0} = q_1 \oplus q_0$$

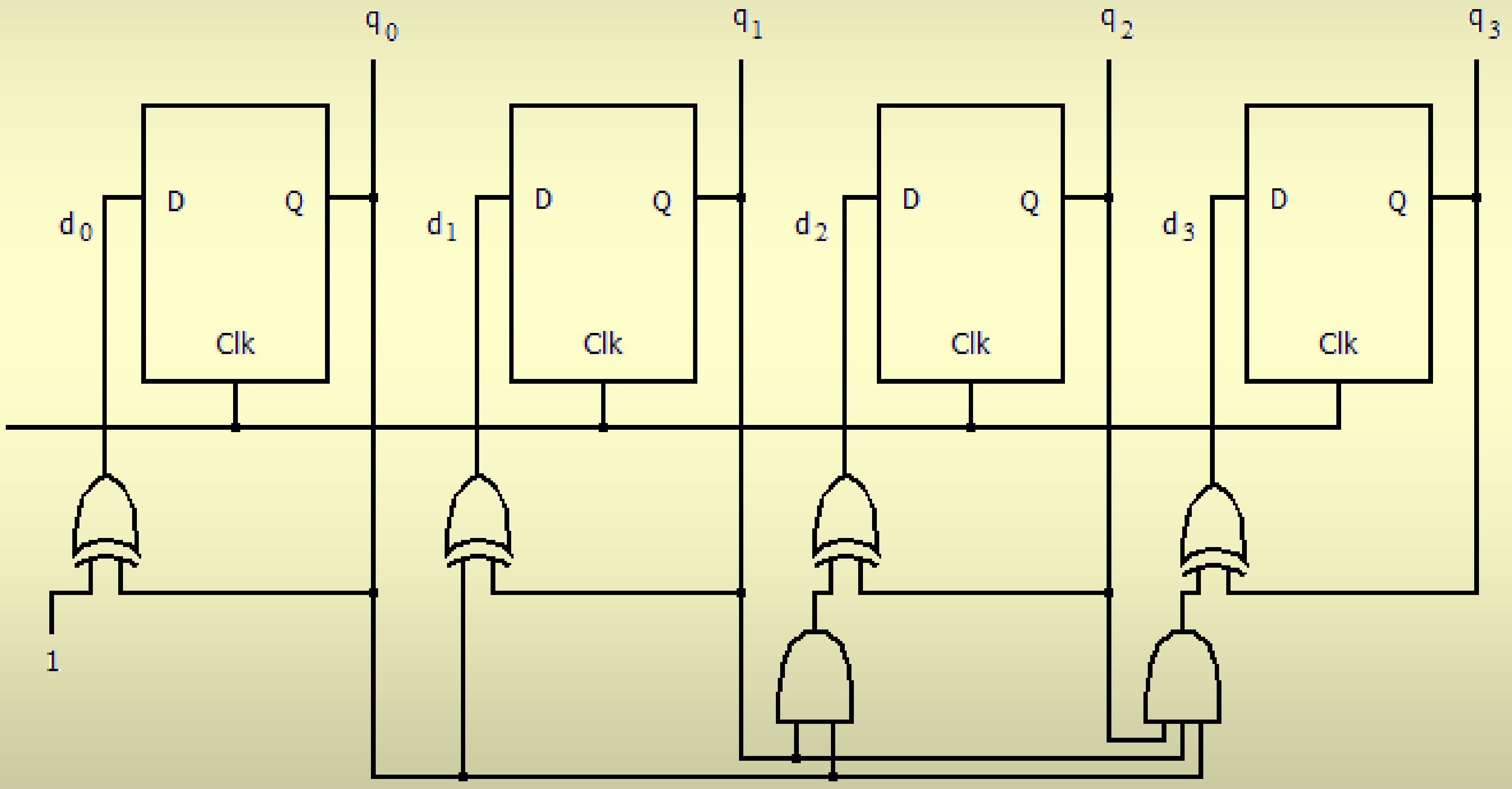
$$d_2 = \overline{q_2} \cdot q_1 \cdot q_0 + q_2 \cdot \overline{q_1} + q_2 \cdot \overline{q_0} = q_2 \oplus (q_1 \cdot q_0)$$

$$d_3 = \overline{q_3} \cdot q_2 \cdot q_1 \cdot q_0 + q_3 \cdot \overline{q_2} + q_3 \cdot \overline{q_1} + q_3 \cdot \overline{q_0} =$$

$$= q_3 \oplus (q_2 \cdot q_1 \cdot q_0)$$

  – implementarea stării - cu flip-flop-uri D

Implementare



Microprogramare (1)

- formă alternativă de implementare
  - starea este memorată tot de circuite bistabile
  - partea combinațională - cu ajutorul unei memorii ROM
  - intrările funcțiilor booleene se aplică la intrările de adresă ale circuitului
  - iar ieșirile funcțiilor booleene se colectează de la ieșirile de date

Microprogramare (2)

- implementarea părții combinaționale
  - se pornește tot de la tabelul de adevăr
  - în fiecare locație se scriu valorile dorite pentru ieșire
- avantaj - flexibilitate
  - orice modificare a automatului implică doar rescrierea conținutului memoriei ROM
- dezavantaj - viteză redusă
  - memoria ROM este mai lentă decât porțile

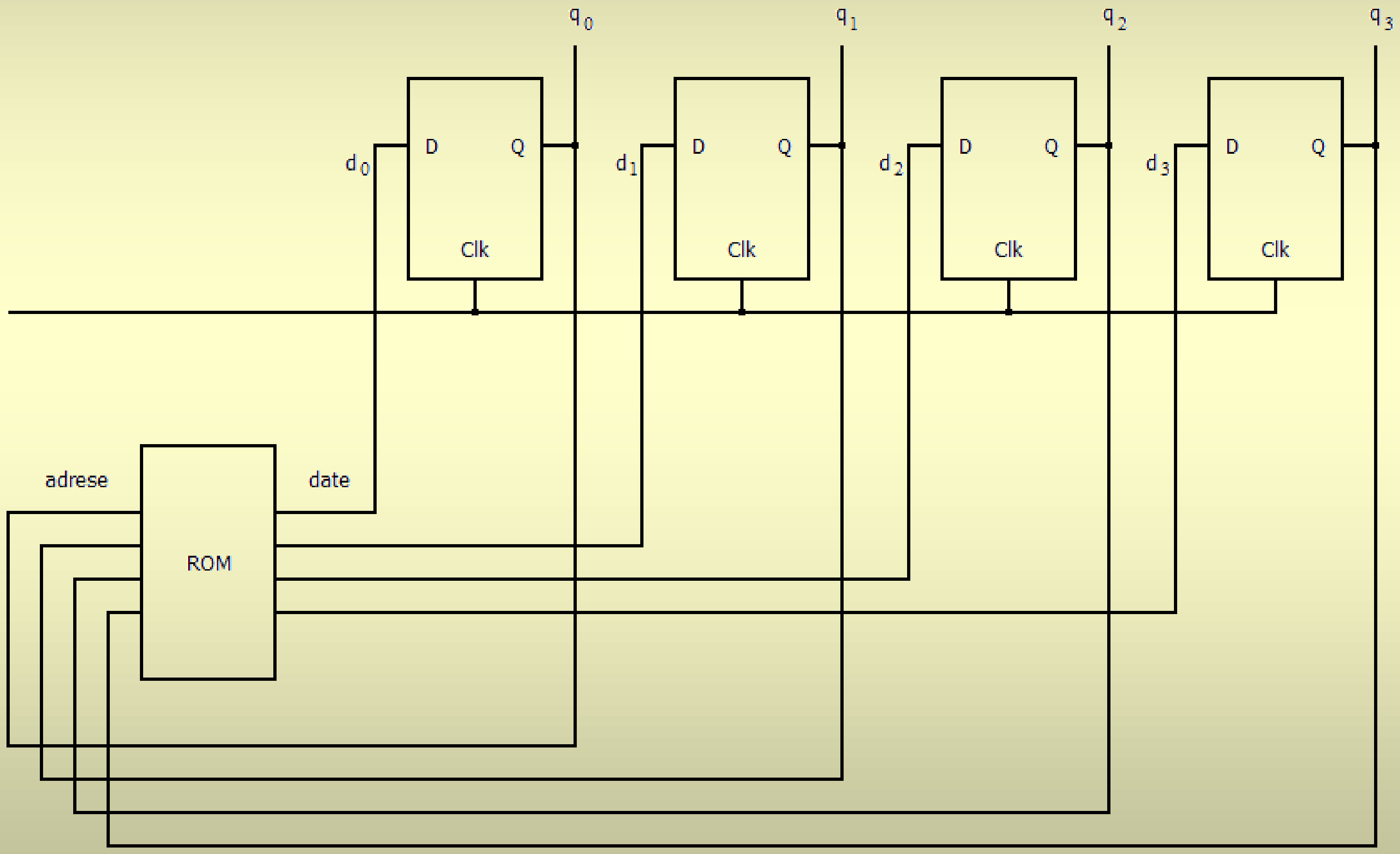
Același exemplu

- avem 16 (= 2^4) stări
  - codificate pe 4 biți de stare
- deci circuitul ROM va avea
  - 2^4 adrese → 4 biți de adresă
    - 16 locații
  - 4 biți de date → locații de 4 biți
  - în acest exemplu nu avem variabile de intrare și ieșiri ale sistemului care să se adauge la biții de stare

Conținutul memoriei ROM

adresă	valoare	adresă	valoare
0	0 0 0 1	8	1 0 0 1
1	0 0 1 0	9	1 0 1 0
2	0 0 1 1	10	1 0 1 1
3	0 1 0 0	11	1 1 0 0
4	0 1 0 1	12	1 1 0 1
5	0 1 1 0	13	1 1 1 0
6	0 1 1 1	14	1 1 1 1
7	1 0 0 0	15	0 0 0 0

Implementare



IV. Reprezentări interne

- reprezentări interne elementare
  - fac parte din arhitectura calculatorului
  - deci sunt implementate în hardware
  - accesibile direct programatorilor
- structuri de date mai complexe
  - pe baza reprezentărilor elementare
  - definite și accesibile programatorilor prin software

Reprezentări elementare

- date numerice
  - numere întregi, raționale
  - doar anumite submulțimi ale acestora
- date alfa-numerice
  - caractere etc.
- instrucțiuni
  - singurele specifice fiecărui sistem
  - deci nestandardizate și neportabile

Studiul reprezentărilor

- reprezentări numerice

  repr(n_1) *op* repr(n_2) = repr(n_1 *op* n_2) ???
- exemplu - dacă adunăm două variabile întregi, rezultatul va fi scris corect?
- erori de reprezentare
  – aproximări
  – depășiri

Transmiterea informațiilor

- între diverse medii
  - între calculatoare/sisteme
  - între componente ale aceluiași calculator/sistem
- pot apărea erori de transmisie
  - datorită perturbărilor/funcționării incorecte
  - semnal digital - unii biți sunt inversați
  - se dorește detectarea apariției acestor erori
  - și chiar corectarea lor, unde e posibil

Moduri de detectare/corectare

- adăugarea de biți suplimentari *redundanți*
- paritate - 1 bit suplimentar
  - permite detecția apariției unei erori (pe 1 bit)
  - paritate (im)pară: număr total (im)par de biți 1
- cod Hamming
  - 4 biţi de informaţie, 3 biţi suplimentari
  - permite detectarea/corecţia mai multor erori simultan

Exemplu paritate impară

- emițător
  - are de trimis valoarea $(110)_2$
  - 2 biți (par) pe 1, deci bitul suplimentar va fi 1
  - se va trimite $(1101)_2$
- receptor
  - primește șirul de biți
  - dacă numărul de biți pe 1 este par - eroare
  - altfel - elimină bitul de paritate și obține $(110)_2$

IV.1. Codificări alfanumerice

Codificări alfanumerice

- calculatorul nu poate reprezenta direct caractere
  - sau alte informații nenumerice: imagini etc.
- fiecărui caracter îi este asociat în mod unic un număr
  - este de fapt o codificare
  - codificarea poate fi la nivel hardware (reprezentare elementară) sau software

Standarde

- ASCII
  - fiecare caracter - 7 biți plus unul de paritate
- EBCDIC
  - fost concurent al ASCII
- ISO 8859-1
  - extinde codul ASCII: diacritice etc.
- Unicode, UCS
  - caractere non-latine

Codul ASCII

- literele mici au coduri consecutive
  - în ordinea dată de alfabetul englez
  - 'a' - 97; 'b' - 98; ...; 'z' - 122
- similar - literele mari (65, 66, ..., 90)
- similar - caracterele care afișează cifrele zecimale
  - atenție: codul pentru caracterul '0' este 48 (nu 0)
- comparații lexicografice - comparator binar

IV.2. Reprezentări interne numerice

Scrierea poziţională

- este tot o reprezentare
  - 397 nu este un număr, ci reprezentarea unui număr
- introdusă de indieni/arabi
- factor implicit ataşat fiecărei poziţii din reprezentare
- esenţială pentru arhitectura calculatoarelor
  - permite algoritmi eficienţi de calcul (adunare...)

Baze de numeraţie

- orice număr natural d>1
- mulţimea cifrelor în baza d: {0,1,…,d-1}
- calculatorul lucrează în baza d=2
  - tehnic: cel mai uşor de implementat fizic 2 cifre
  - teoretic: baza 2 se "potrivește" cu logica booleană
    - ca simboli şi ca operaţii
    - operațiile se pot implementa prin funcții booleene

Limitele reprezentărilor

- în practică, numărul de cifre este finit
- exemplu - numere întregi fără semn
  - pe 1 octet: $0 \div 2^8-1$ (= 255)
  - pe 2 octeți: $0 \div 2^{16}-1$ (= 65535)
  - pe 4 octeți: $0 \div 2^{32}-1$ (= 4294967295)
- orice număr mai mare (sau mai mic) decât limitele nu va putea fi reprezentat corect

Scrierea poziţională

- fie baza $d \in N^{*}-\{1\}$
- și reprezentarea dată de șirul de cifre
  $a_{n-1}a_{n-2}\ldots a_1a_0a_{-1}\ldots a_{-m}$
- numărul corespunzător reprezentării este

$$\sum_{i=-m}^{n-1}(a_i \times d^i)_{(10)}$$

- d^i este factorul implicit asociat poziției i
  - inclusiv pentru puteri negative

Treceri dintr-o bază d în baza 10

- conform formulei anterioare
- virgula rămâne în același loc
- exemplu

$5E4,D_{(16)} = 5\times16^2 + 14 \times16^1 + 4 \times16^0 +13 \times16^{-1} = 20480 + 3584 + 64 + 0,8125 = 24128,8125_{(10)}$

Trecerea din baza 10 în baza d

Exemplu: $87{,}35_{(10)} = 1010111{,}01(0110)_{(2)}$

partea întreagă

87 / 2 =	43	rest	1
43 / 2 =	21	rest	1
21 / 2 =	10	rest	1
10 / 2 =	5	rest	0
5 / 2 =	2	rest	1
2 / 2 =	1	rest	0
1 / 2 =	0	rest	1

$87_{(10)} = 1010111_{(2)}$

(cifrele se scriu de jos în sus)

partea fracționară

$0{,}35 \times 2 = 0{,}7 + 0$

$0{,}7 \times 2 = 0{,}4 + 1$

$0{,}4 \times 2 = 0{,}8 + 0$

$0{,}8 \times 2 = 0{,}6 + 1$

$0{,}6 \times 2 = 0{,}2 + 1$

$0{,}2 \times 2 = 0{,}4 + 0$

$0{,}4 \times 2 = 0{,}8 + 0$

(perioadă)

$0{,}35_{(10)} = 0{,}01(0110)_{(2)}$

Conversii între baze

- o bază este o putere a celeilalte baze
  - $d_1 = d_2^k \Rightarrow$ fiecărei cifre în baza d_1 îi corespund exact k cifre în baza d_2
- ambele baze sunt puteri ale numărului *n*
  - conversia se poate face prin intermediul bazei *n*

$703{,}102_{(8)} = 111\ 000\ 011{,}001\ 000\ 010_{(2)} =$

$= 0001\ 1100\ 0011{,}0010\ 0001\ 0000_{(2)} =$

$= 1C3{,}21_{(16)}$

Aproximare și depășire

- o reprezentare are *n* cifre la partea întreagă și *m* cifre la partea fracționară
  - *n* și *m* sunt finite
- dacă numărul necesită mai mult de *n* cifre la partea întreagă, se produce depăşire
- dacă numărul necesită mai mult de *m* cifre la partea fracționară, apare o aproximare
  - de cel mult 2^{-m}

IV.3. Reprezentările BCD și în exces

Reprezentarea BCD

- numerele sunt reprezentate ca şiruri de cifre în baza 10
  - fiecare cifră este reprezentată pe 4 biţi
- utilitate
  - aplicații de tip business (financiar etc.)
  - afișaje în baza 10 (temperatură etc.)
- calculele sunt dificil de efectuat
  - adunare - nu se poate utiliza direct un sumator

Adunarea BCD (1)

5	=	0101	+		5	=	0101	+
3	=	0011			8	=	1000	
$8_{(10)}$	=	1000	$= 8_{BCD}$		$13_{(10)}$	=	1101	
						$\neq 13_{BCD} =$		
						$= 0001\ 0011$		

problemele apar atunci când suma cifrelor depăşeşte 9

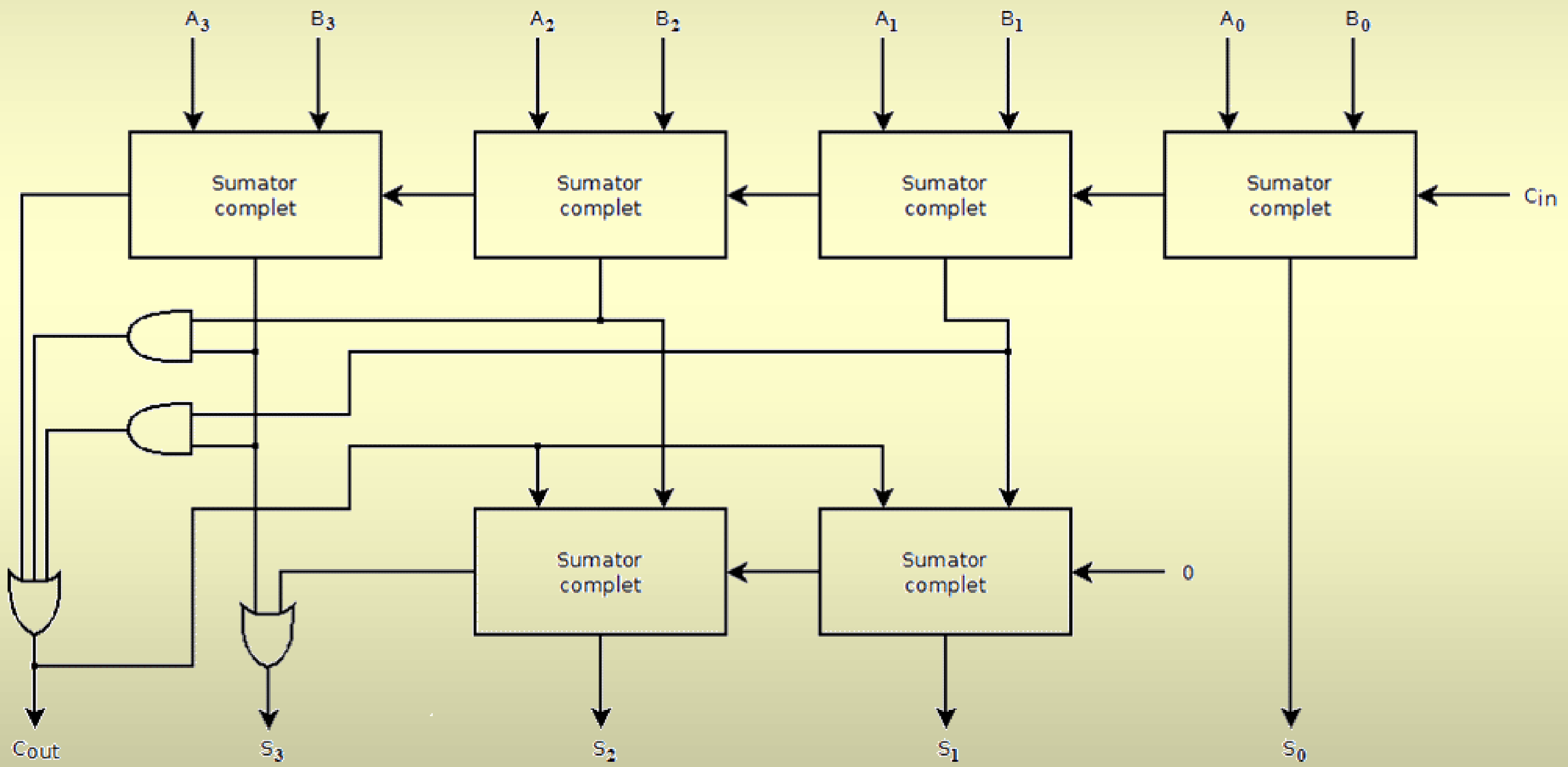
Adunarea BCD (2)

- soluţie
  - se adună 6 (0110) atunci când suma depăşeşte 9
- temă: de ce?

$$\begin{array}{rcrl} 5 & = & 0101 & + \\ 8 & = & 1000 & \\ \hline & & 1101 & + \\ 6 & = & 0110 & \\ \hline & & 1\ 0011 & = 13_{BCD} \end{array}$$

$$\begin{array}{rcrl} 9 & = & 1001 & + \\ 7 & = & 0111 & \\ \hline 16_{(10)} & = & 1\ 0000 & \neq 16_{BCD} \\ 6 & = & 0110 & \\ \hline & & 1\ 0110 & = 16_{BCD} \end{array}$$

Sumator BCD



A_3
B_3
A_2
B_2
A_1
B_1
A_0
B_0
Sumator complet
Sumator complet
Sumator complet
Sumator complet
Cin
Sumator complet
Sumator complet
0
Cout
S_3
S_2
S_1
S_0

Reprezentarea în exces

- pornește de la scrierea pozițională
  - numere pozitive
  - pe n biți, intervalul reprezentabil este $0 \div 2^n-1$
- reprezentarea Excess-k
  - pentru fiecare șir de biți, din valoarea care îi corespunde în scrierea pozițională se scade k
  - intervalul reprezentabil devine $-k \div 2^n-k-1$

Exemplu: Excess-5

Binar	Zecimal	Excess-5	Binar	Zecimal	Excess-5
0000	0	-5	1000	8	3
0001	1	-4	1001	9	4
0010	2	-3	1010	10	5
0011	3	-2	1011	11	6
0100	4	-1	1100	12	7
0101	5	0	1101	13	8
0110	6	1	1110	14	9
0111	7	2	1111	15	10