AUTOMATE SINCRONE

S.l. Ing. Vlad-Cristian Miclea

Universitatea Tehnica din Cluj-Napoca
Departamentul Calculatoare

CUPRINS

PLAN CURS

- Partea 1 – VHDL
  1. Limbajul VHDL – 1
  2. Limbajul VHDL – 2
  3. Limbajul VHDL – 3
- Partea 2 – Implementarea sistemelor numerice
  4. Microprogramare
  5. Partea 1 - Unitate de comanda – exemplu cuptor
  5. Partea 2 - Unitate de executie – exemplu cuptor
- Partea 3 – Automate
  6. Automate finite
  7. Stari
  8. **Automate sincrone**
  9. Automate asincrone
  10. Identificarea automatelor
  11. Automate fara pierderi
  12. Automate liniare
- Partea 4 – Probleme si discutii

CONTEXT

Cursurile trecute

- Automate finite
  - Abstractizarea circuitelor secventiale
  - Reprezentari
  - Clasificarea automatelor (Moore, Mealy)
- Stari ale automatelor
  - Reducerea numarului de stari
    - Paull-Unger
    - Complet vs incomplete definite
  - Codificarea eficienta a starilor

AUTOMATE SINCRONE

Introducere

- Automatele – abstractizari ale oricarui circuit
- Sincrone – toate schimbarile se intampla pe baza ceasului (CLK) – circuite secventiale clasice
- Nu permit intrari asincrone, nici tranzitii asincrone
- Exista mai multe modalitati de implementare
  - In functie de complexitatea circuitului
    - Trade-off intrari vs stari vs iesiri
  - **Obiectiv**: Reducerea numarului de resurse la implementarea automatului
    - Codificarea cat mai eficienta pentru nr de stari
    - Codificarea cat mai eficienta pentru nr de intrari/iesiri

AUTOMATE SINCRONE

Implementare cu memorii

- Memorii: ROM, RAM
  - Datele stocate la adresa curenta dau starea urmatoare
- Memoriile ROM:
  - Intrări = adrese – n biţi
  - Ieşiri = date
  - Număr locaţii adresate – 2^n
  - Lungime cuvânt memorie – m
  - Capacitate memorie – C = 2^n x m
  - Acces la date:
    - fixare adrese; generare CS; după trecerea timpului de acces la memorie datele apar pe liniile de date

METODE DE IMPLEMENTARE

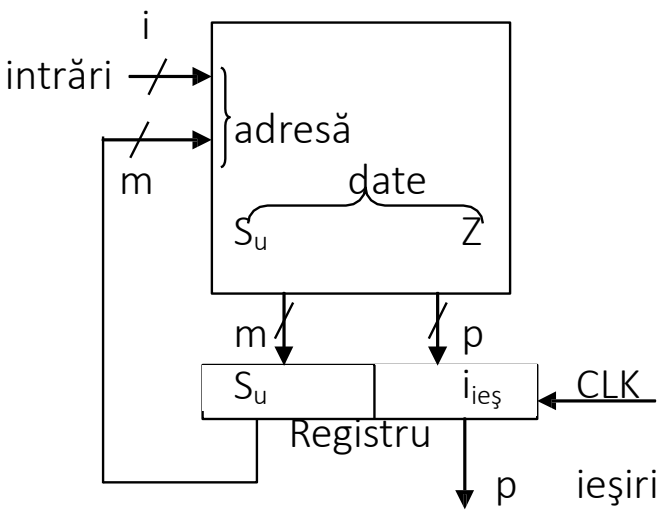
- Adresare pe arc
- Adresare pe stare
  - a. cu adresă pereche
  - b. cu adresă presupusă
  - c. cu format variabil

1. ADRESARE PE ARC

- Memoria ROM va fi folosita pentru generarea urmatoarei stari
- Adresa contine starea curenta si intrarile
  - Identifica unic generarea urmatoarei stari
- Datele contin starea urmatoare si iesirile
- “Adresare pe arc” – pt adresa se tine cont de valoarea varibilelor de intrare
- Varianta cea mai simplista

1. ADRESARE PE ARC

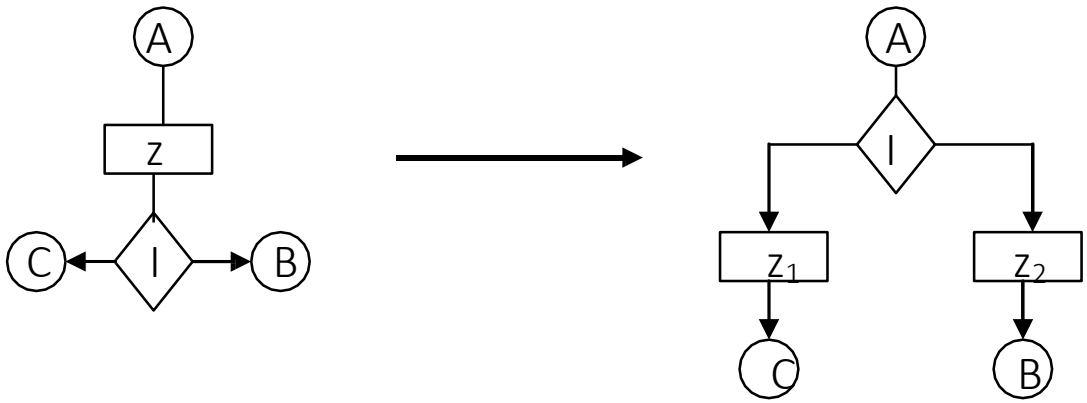
Schema bloc



- i = nr. intrări
- m = nr. de biţi pt. codificarea stării următoare S_u
- p = nr. de ieşiri distincte
- n= i+m = nr. biţi de adresă
- l = m+p = lungimea cuvântului de memorie
- C = 2^n x l = capacitatea memoriei
- Structură cuvânt de memorie
  - m biţi pentru codificarea stării următoare S_u
  - p biţi pentru ieşiri $i_{ieş}$

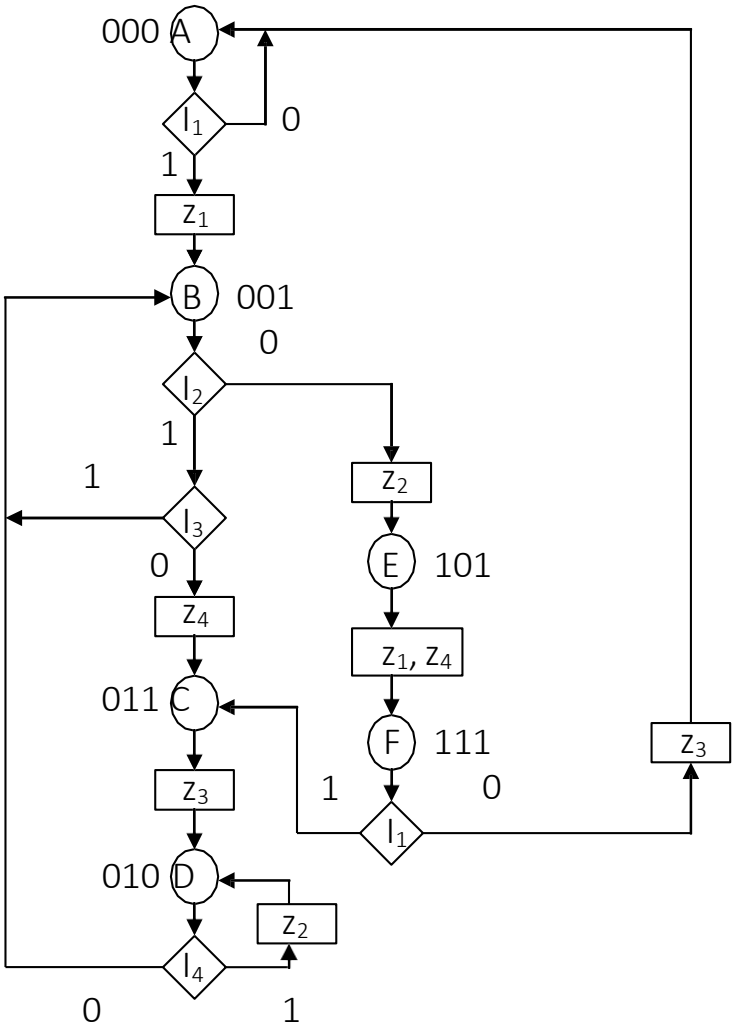
1. ADRESARE PE ARC

- Valabilă pentru automate de tip Mealy
  - starea următoare determinată de intrări şi de starea prezentă
- Pentru alte tipuri de automate
  - în organigramă - translatarea ieşirilor la capetele arcelor:



1. ADRESARE PE ARC

Exemplu



1. ADRESARE PE ARC

Exemplu

- Necesarul de memorie:
  - i = 4 intrări
  - m = 3 biţi pt. codificarea stării următoare
  - p = 4 ieşiri
  - n = i+m = 7 biţi de adrese
  - l = m+p = 7 biţi lungimea cuvântului de memorie
  - C = 2^n x l = 896 biţi – capacitatea memoriei
- Observaţie
  - nu este necesară codificarea adiacentă a stărilor (nu avem o implementare cu bistabile!)

1. ADRESARE PE ARC

Harta memoriei

- Adresele şi informaţia care va fi înscrisă în memorie

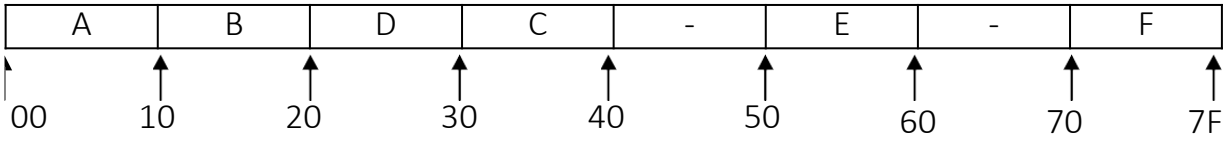
	Stări	Adrese							Date							
		Q_2	Q_1	Q_0	I_4	I_3	I_2	I_1	Q^t_2	Q^t_1	Q^t_0	z_4	z_3	z_2	z_1	
00	A	0	0	0	x	x	x	0	0	0	0	0	0	0	0	00
01		0	0	0	x	x	x	1	0	0	1	0	0	0	1	11
10	B	0	0	1	x	x	0	x	1	0	1	0	0	1	0	52
12		0	0	1	x	0	1	x	0	1	1	1	0	0	0	38
16		0	0	1	x	1	1	x	0	0	1	0	0	0	0	10
30	C	0	1	1	x	x	x	x	0	1	0	0	1	0	0	24
20	D	0	1	0	0	x	x	x	0	0	1	0	0	0	0	10
28		0	1	0	1	x	x	x	0	1	0	0	0	1	0	22
50	E	1	0	1	x	x	x	x	1	1	1	1	0	0	1	79
70	F	1	1	1	x	x	x	0	0	0	0	0	1	0	0	04
71		1	1	1	x	x	x	1	0	1	1	0	0	0	0	30
		Starea actuală			Intrări				Starea următoare			Ieşiri				

1. ADRESARE PE ARC

Harta memoriei

- Dispunerea locaţiilor pentru stări

Adresă (hexa)	Conţinut (hexa)	Stare
00 – 0E (valori pare)	00	A
01 – 0F (valori impare)	11	A
10 – 11; 14 – 15; 18 – 19; 1C – 1D	52	B
12 – 13; 1A – 1B	38	B
16 – 17; 1E – 1F	10	B
20 – 27	10	D
28 – 2F	22	D
30 – 3F	24	C
40 – 4F	-	-
50 – 5F	79	E
60 – 6F	-	-
70 – 7E (valori pare)	04	F
71 – 7F (valori impare)	30	F



- Memoria este complet ineficient ocupată!!!

1. ADRESARE PE ARC

Observatii

- Metoda ar fi eficienta daca am avea putine intrari, dar multe iesiri
  - Putine adrese, mai multi biti pt date
- Memoria este ineficient ocupată
- Problema 1: Starile nu sunt adiacente, deci ocupa o zona mare de memorie
- Problema 2: Exista prea multe adrese generate
  - Capacitatea memoriei creste in principal datorita numarului mare de adrese (exponentiala)
  - Adresele contin si intrarile, si starea curenta
  - Nu e necesar sa folosim intrarile ca adresa!
- Tratarea ambelor probleme - reducerea numarului de adrese!

2. ADRESARE PE STARE

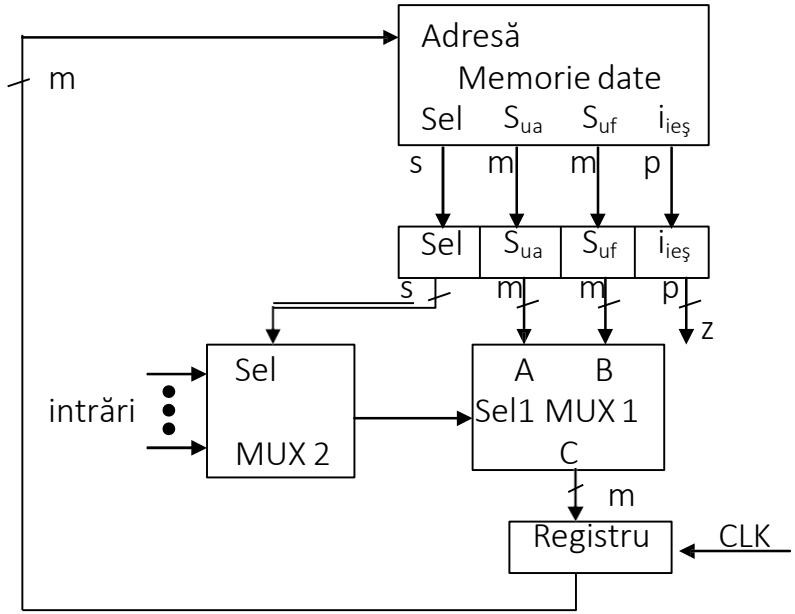
Alternativa

- Obiectiv - sa reducem capacitatea memoriei
  - Practic, nu avem nevoie de toate intrarile deodata; Putem sa le selectam doar cand e nevoie de ele
  - Putem sa **codificam intrarile** si sa transmitem doar semnale de selectie (mutam intrarile din “adresa” in “date”)
  - Putem sa nu tinem chair toate adresele urmatoare
- Exista mai multe moduri de a memora starea urmatoare si variabilele
  - La fiecare adresa tinem 2 posibilitati, in functie de intrari - **adresa pereche**
  - La fiecare adresa tinem doar alternativa (adresa cea mai probabila - e implicita) – **adresa presupusa**
  - La fiecare adresa tinem sau starea urmatoare, sau iesirile – **adresa cu format variabil**

2. ADRESARE PE STARE

a. Adresare pe stare cu adresă pereche

Schema bloc

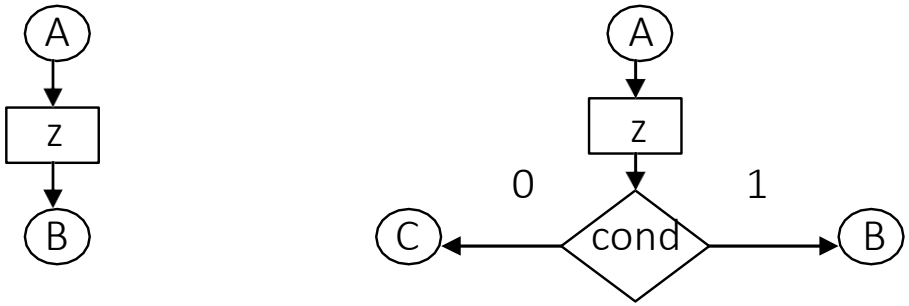


- Sel = selecţie – s biţi
- S_{ua} = starea următoare pentru intrări adevărate – m biţi
- S_{uf} = starea următoare pentru intrări false – m biţi
- $i_{ieş}$ = ieşiri – p biţi
- C= 2^m x (s+2m+p) = capacitatea memoriei

2. ADRESARE PE STARE

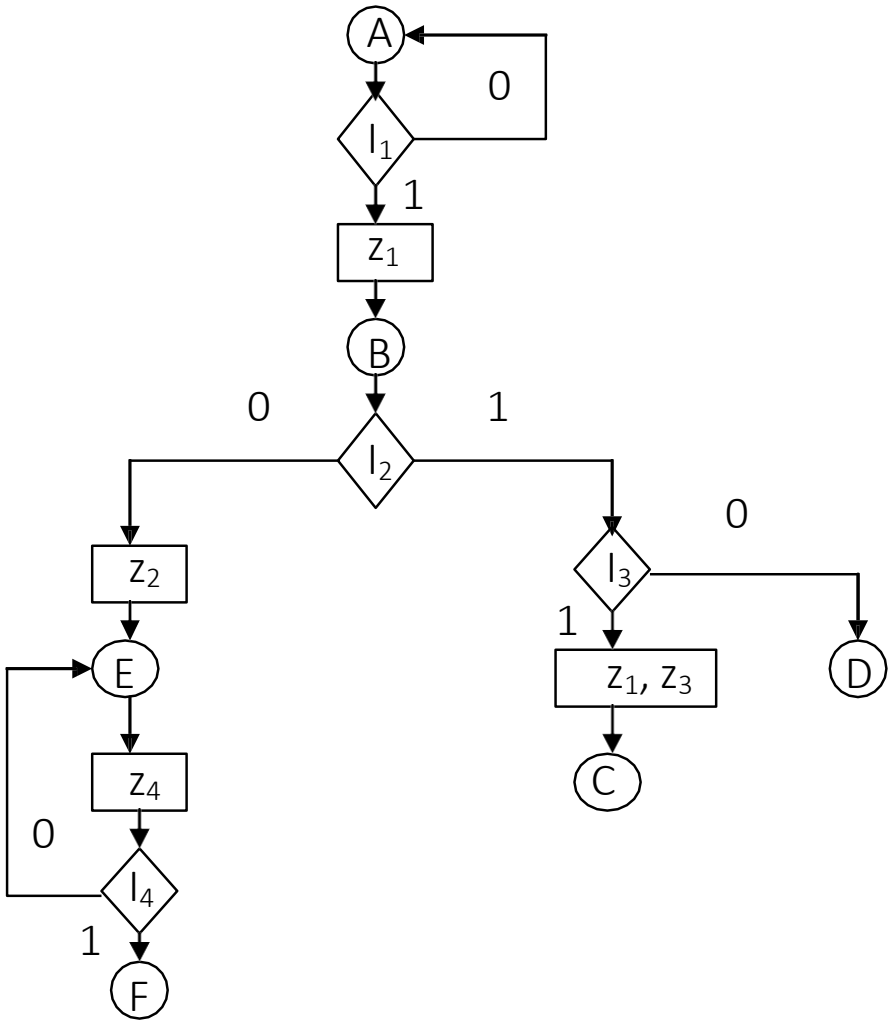
a. Adresare pe stare cu adresă pereche

- Valabilă pentru automate de tip Moore
  - ieşirile depind doar de variabilele de stare
- Pentru alte tipuri de automate
  - în organigramă – modificări pentru a obţine structuri doar de formele:



2. ADRESARE PE STARE

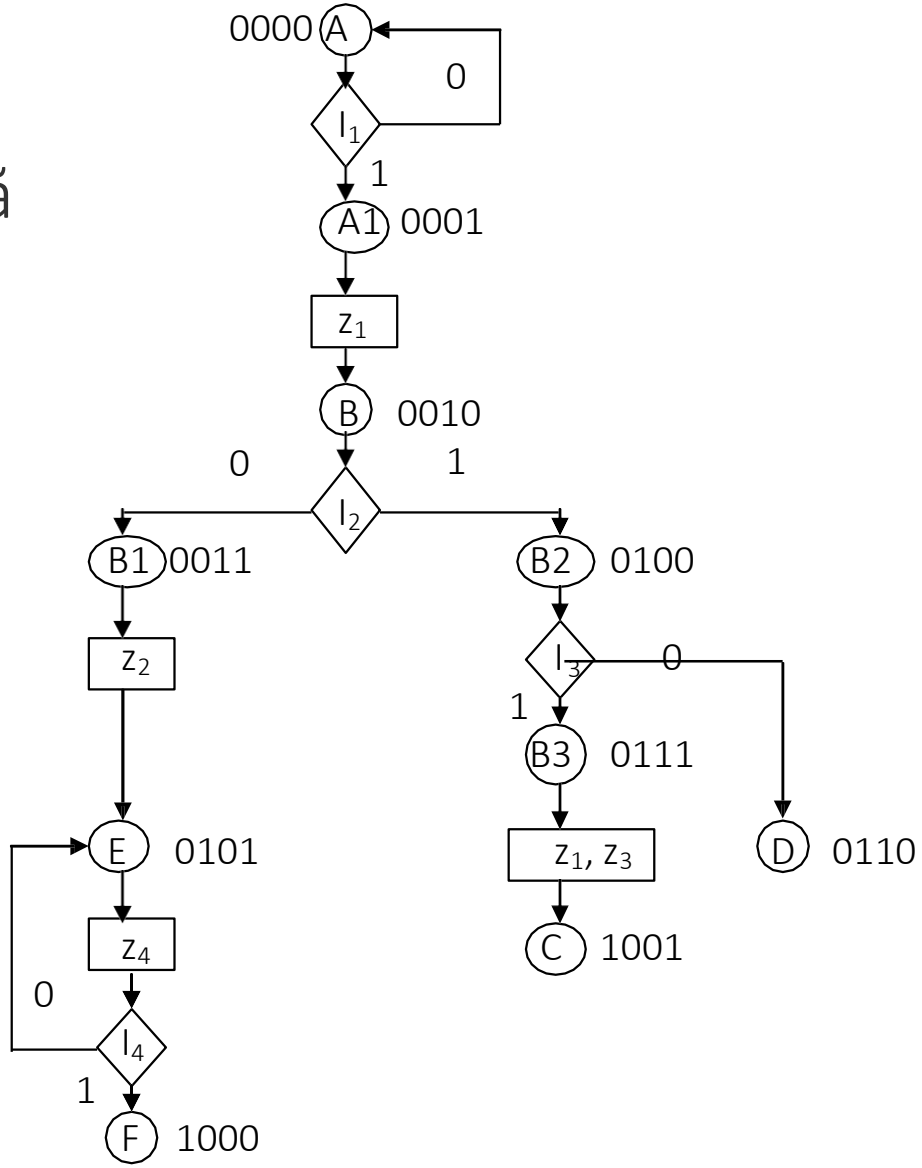
Exemplu



2. ADRESARE PE STARE

Exemplu

- Organigrama modificată
  - după A, pentru I_1=1
    - se introduce starea A_1
  - după B, pentru I_2=0
    - se introduce B_1
  - după B, pentru I_2=1
    - se introduce B_2
  - după B_2, pentru I_3= 1
    - se introduce B_3



2. ADRESARE PE STARE

Exemplu

- Codificarea (arbitrară) a stărilor şi condiţiile de test pentru intrări

A	0000	0
A1	0001	1
B	0010	2
B1	0011	3
B2	0100	4
D	0110	6
B3	0111	7
C	1001	9
E	0101	5
F	1000	8

	S_1S_0	
I_1	00	0
I_2	01	1
I_3	10	2
I_4	11	3

2. ADRESARE PE STARE

a. Adresare pe stare cu adresă pereche

- Capacitatea memoriei: C= 2^m x (s+2m+p) = 224 biţi
- Harta memoriei

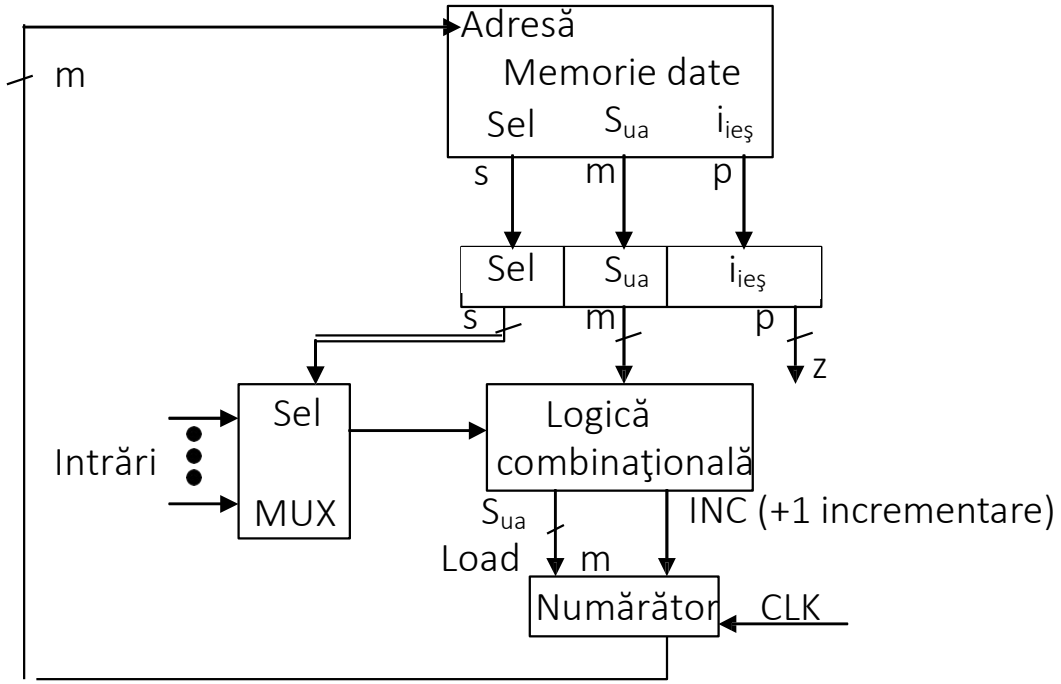
	Adresă stare prezentă	Sel (cond. de test)	S_{ua}	S_{uf}	z_4, z_3, z_2, z_1 $i_{ieş}$
A	0 (0000)	00 (I_1)	1 (0001)	0 (0000)	0 (0000)
A1	1 (0001)	xx	2 (0010)	2 (0010)	1 (0001)
B	2 (0010)	01 (I_2)	4 (0100)	3 (0011)	0 (0000)
B1	3 (0011)	xx	5 (0101)	5 (0101)	2 (0010)
B2	4 (0100)	10 (I_3)	7 (0111)	6 (0110)	0 (0000)
E	5 (0101)	11 (I_4)	8 (1000)	5 (0101)	8 (1000)
D	6 (0110)	În funcţie de dezvoltarea ulterioară			
B3	7 (0111)	xx	9 (1001)	9 (1001)	5 (0101)
F	8 (1000)	În funcţie de dezvoltarea ulterioară			
C	9 (1001)	În funcţie de dezvoltarea ulterioară			

- Număr mai mic de locaţii de memorie, dar număr mai mare de biţi / locaţie

2. ADRESARE PE STARE

b. Adresare pe stare cu adresă presupusă

Schema bloc

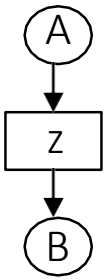


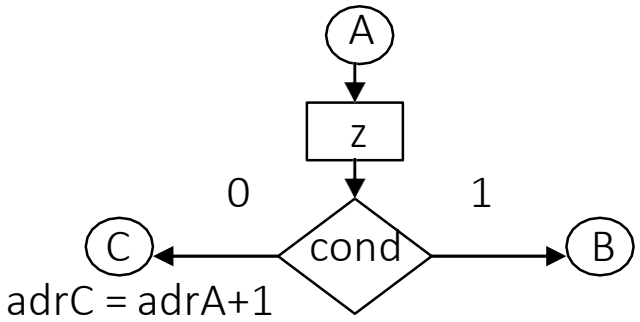
- Sel = selecţie – s biţi
- S_{ua} = starea următoare pentru intrări adevărate – m biţi
- $i_{ieş}$ = ieşiri – p biţi
- C= 2^m x (s+m+p) = capacitatea memoriei

2. ADRESARE PE STARE

b. Adresare pe stare cu adresă presupusă

- Structurile admise în organigramă au o condiţie în plus:
  - codul pentru S_{uf} = codul pentru $S_{actuală}$ +1
  - Una dintre ramuri, trebuie sa aiba adresa incrementata

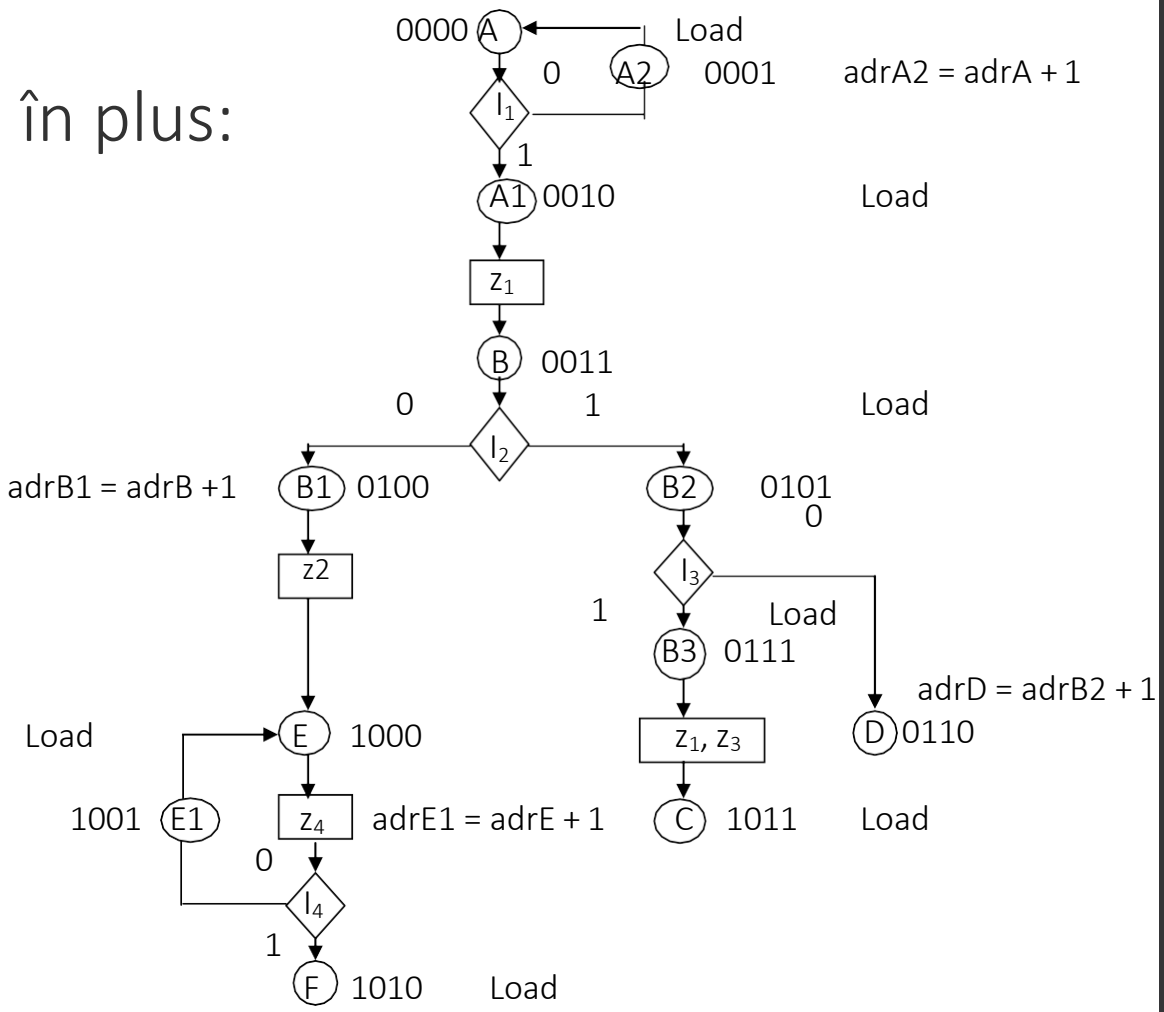




2. ADRESARE PE STARE

Exemplu

- Organigrama modificată în plus:
  - după A, pentru I_1=0
    - se introduce starea A_2
  - după E, pentru I_4=0
    - se introduce E_1



2. ADRESARE PE STARE

b. Adresare pe stare cu adresă presupusă

- Capacitatea memoriei: C= 2^m x (s+m+p) =160 biţi
- Harta memoriei

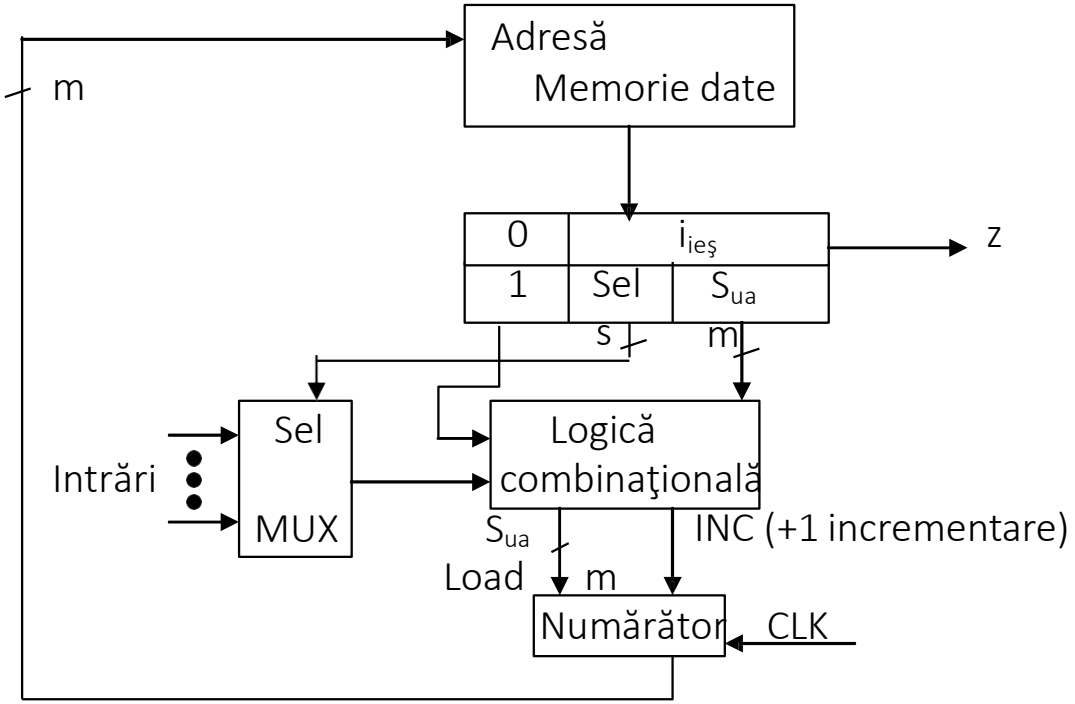
	Adresă stare prezentă	Sel (cond. de test)	S_{ua}	z_4, z_3, z_2, z_1 $i_{ieş}$
A	0 (0000)	00 (I_1)	2 (0010)	0 (0000)
A2	1 (0001)	xx	0 (0000)	0 (0000)
A1	2 (0010)	xx	3 (0011)	1 (0001)
B	3 (0011)	01 (I_2)	5 (0101)	0 (0000)
B1	4 (0100)	xx	8 (1000)	2 (0010)
B2	5 (0101)	10 (I_3)	7 (0111)	0 (0000)
D	6 (0110)	În funcţie de dezvoltarea ulterioară		
B3	7 (0111)	xx	B (1011)	5 (0101)
E	8 (1000)	11 (I_4)	A (1010)	8 (1000)
E1	9 (1001)	xx	8 (1000)	0 (0000)
F	A (1010)	În funcţie de dezvoltarea ulterioară		
C	B (1011)	În funcţie de dezvoltarea ulterioară		

- Lipseşte coloana pentru S_{uf}

2. ADRESARE PE STARE

c. Adresare pe stare cu format variabil

Schema bloc



- Sel = selecţie – s biţi
- S_{ua} = starea următoare pentru intrări adevărate – m biţi
- $i_{ieş}$ = ieşiri
- C= 2^m x (s+m+1) = capacitatea memoriei

2. ADRESARE PE STARE

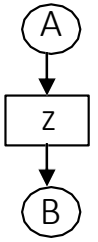
c. Adresare pe stare cu format variabil

- Avem 2 tipuri de cuvinte de memorie

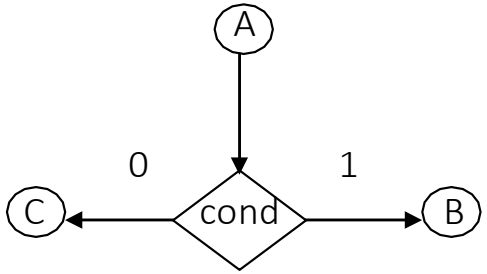
0	Ieşiri	
1	Sel	S_{ua}

format 0
format 1

- Primul bit al cuvântului de memorie stabileşte tipul
  - Nu mai putem avea iesiri, in stari unde sunt conditii
- Structurile admise în organigramă au condiţiile:
  - format 0 – codul pentru S_u = codul pentru $S_{actuală}$ +1
  - format 1 – codul pentru S_{uf} = codul pentru $S_{actuală}$ +1



adrB = adrA+1
format 0

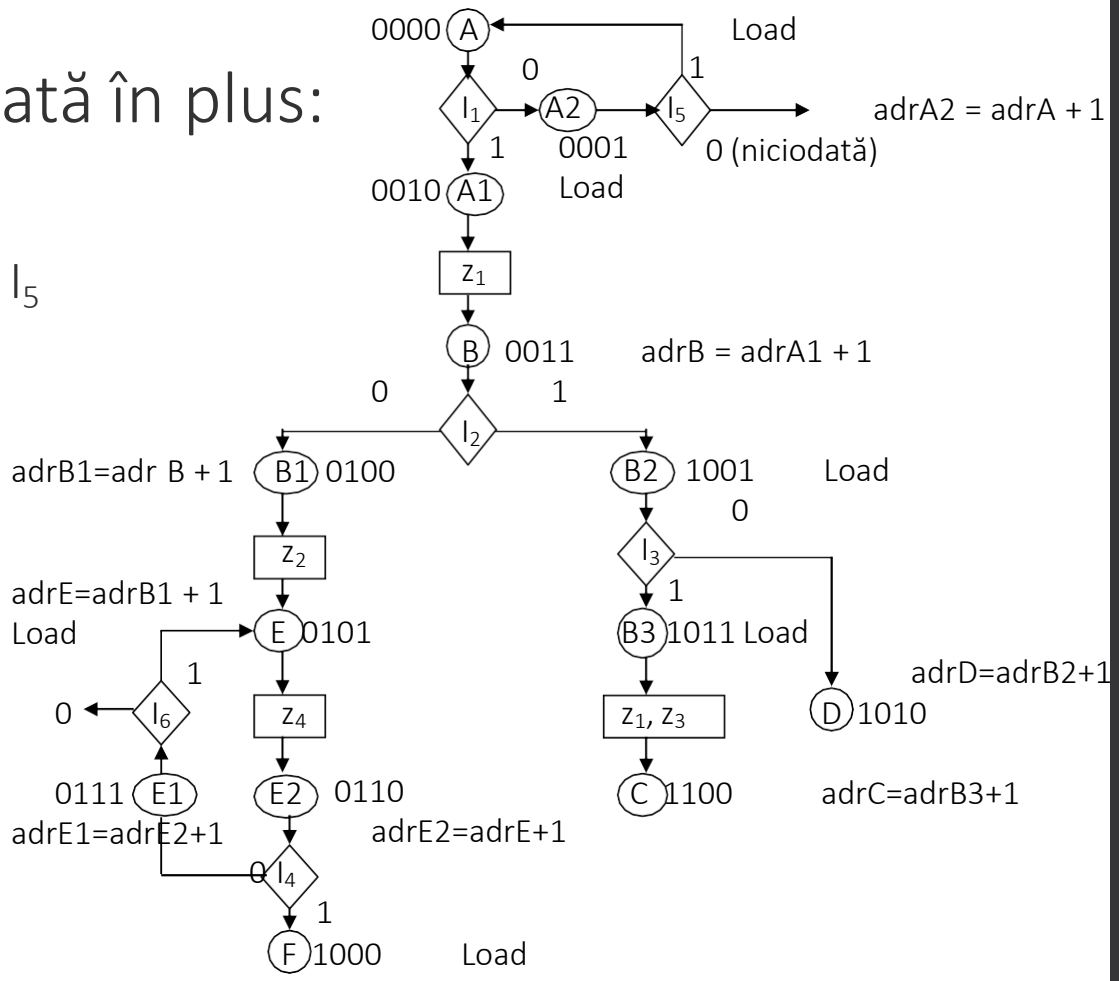


adrC = adrA+1
format 1

2. ADRESARE PE STARE

Exemplu

- Organigrama modificată în plus:
  - după starea A_2
    - se introduce intrarea I_5
  - după starea E
    - se introduce I_6
  - după ieşirea z_4
    - se introduce E_2
  - intrările I_5 şi I_6
    - iau doar valoarea 1



2. ADRESARE PE STARE

c. Adresare pe stare cu format variabil

- Capacitatea memoriei: C= 2^m x (s+m+1) =128 biţi
- Codificarea intrărilor – pe 3 biţi (3 linii de Sel la MUX)

	$S_2S_1S_0$
I_1	000
I_2	001
I_3	010
I_4	011
I_5	100
I_6	101

2. ADRESARE PE STARE

c. Adresare pe stare cu format variabil

- Harta memoriei

Stare	Adresă	Format	Sel	S_{ua}
			$z_4z_3z_2z_1$	
A	0 (0000)	1	000 (I_1)	0010
A2	1 (0001)	1	100 (I_5)	0000
A1	2 (0010)	0	0000001	
B	3 (0011)	1	001 (I_2)	1001
B1	4 (0100)	0	0000010	
E	5 (0101)	0	0001000	
E2	6 (0110)	1	011 (I_4)	1000
E1	7 (0111)	1	101 (I_6)	0101
F	8 (1000)	?	?	?
B2	9 (1001)	1	010 (I_3)	1011
D	0A (1010)	?	?	?
B3	0B (1011)	0	0000101	
C	0C (1100)	?	?	?

- Numărul biţilor pentru Sel + S_{ua} = numărul biţilor pentru ieşiri

Concluzii

- Automate sincrone
- Adresare pe arc
- Adresare pe stare
  - Pereche
  - Presupusa
  - Variabila
- Data viitoare – automate asincrone