CALCULATOARE NUMERICE

Calculatoare 2 2024-2025 Sem. 2

CURS 4 Sapt 4 20 martie 2025 12:00-14:00

serban@upit.ro

Regulament disciplina

Nota finala este formata din activitatile:

- Laborator 30%
- Lucrare control (midterm) 20%
- Examen 50%
- Bonus – prezenta activa curs 10%

Conditii pentru promovare:

- Nota de la laborator trebuie sa fie minim 5 (prezenta obligatorie la toate sedintele de laborator);
- Nota la lucrarea de control trebuie sa fie minim 5;
- Nota de la examen trebuie sa fie minim 5.

In cazul reluarii disciplinei intr-un alt an universitar, activitatile nepromovate trebuie parcurse din nou.

Continut disciplina

Structuri digitale folosite în calculatoare numerice

Memorii semiconductoare in calculatoare numerice

Memoria cache

Memoria virtuală

Adresarea memoriei in sistemele de calcul

Bibliografie

1. **David PATTERSON, John HENNESSY** *Computer Organization and Design The Hardware/Software Interface,* 5th ed., Morgan Kaufmann Elsevier 2012 (a se vedea şi traducerea în lb. română David A. Patterson, John L. Hennessy *Organizarea şi proiectarea calculatoarelor, Interfaţa hardware/software*; Editura All, Bucureşti, 2002);
2. John L. HENNESSY, David A. PATTERSON *Computer Architecture, A Quantitative Approach*, 5th ed., Morgan Kaufmann Publishers, Inc, San Francisco, 2012;
3. Bruce JACOB, Spencer NG, David WANG *Memory Systems Cache, DRAM, Disk* Morgan Kaufmann Elsevier 2008;
4. Miles MURDOCCA, Vincent HEURING *Principles of Computer Architecture*, Pretice Hill, 1999;
5. Jim HANDY The Cache Memory, 2nd ed., Academic Press Elsevier, 1998;
6. **Andrew S. TANENBAUM, Todd AUSTIN** *Structured Computer Organization*, 6th ed., Prentice-Hall, Inc., 2013 (a se vedea şi traducerea în lb. română Andrew S. Tanenbaum *Organizarea Structurată a Calculatoarelor*, Agora, Tg. Mureş, 2004);
7. **William STALLINGS** *Computer Organization and Architecture: Designing for Performance*, 9th edition, Prentice-Hall Inc., 2013;
8. Vincent HEURING, Harry JORDAN *Computer Systems Design and Architecture*, 2nd ed., Person Prentince Hill, 2007;
9. Sajjan G. SHIVA *Computer Organization, Design, and Architecture,* 4th ed., CRC Press, Taylor & Francis Group, Boca Raton, USA, 2008;
10. Carl HAMACHER, Zvonko VRANESIC, Safwat ZAKY, Naraig MANJIKIAN *Computer organization and embedded systems, 6th edition* McGraw Hill, 2012

AUTOMATE

Automat – o structura digitala care poate evolua intr-o multime de stari logice, cu trecerea dintr-o stare anterioara intr-o stare urmatoare, in conformitate cu un algoritm bine stabilit, tinand seama de valorile logice ale unor semnale de intrare si cu generarea unor semnale de iesire.

Un automat este un cvintuplu: A = (S, I, O, f, g) unde

S – multimea finita a starilor posibile

I – multimea finita a semnalelor de intrare

O – multimea finita a semnalelor de iesire

f – multimea functiilor de tranzitie care precizeaza starile viitoare in care ajunge automatul ca urmare a aplicarii unui semnal de intrare ($f : S \times I \rightarrow S$)

g – multimea functiilor de iesire care precizeaza semnalul de iesire generat in cazul aplicarii unui semnal de intrare ($g : S \times I \rightarrow O$)

Tipuri de automate: cu stari finite de tip Mealy, Moore, microprogramate

AUTOMATE MICROPROGRAMATE

Un Automat Microprogramat evolueaza pe baza unui Microprogram scris intr-o memorie (ROM) format din Microintrucțiuni. Fiecarei stari din evolutia automatului ii corespunde o microinstructiune prin care se rezolva mai multe sarcini: selectarea si testarea unui semnal de intrare specific starii respective, generarea corespunzatoare a unui semnal de iesire (comanda), pregatirea trecerii in starea urmatoare, corespunzatoare algoritmului implementat.

2 tipuri de automate microprogramate:

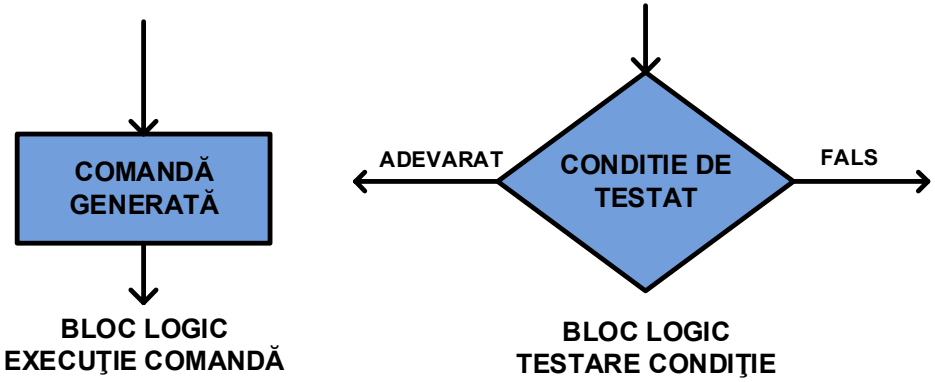
- cu format fix al microintrucțiunilor
- cu format variabil al microinstructiunilor

End 13 mar 2025

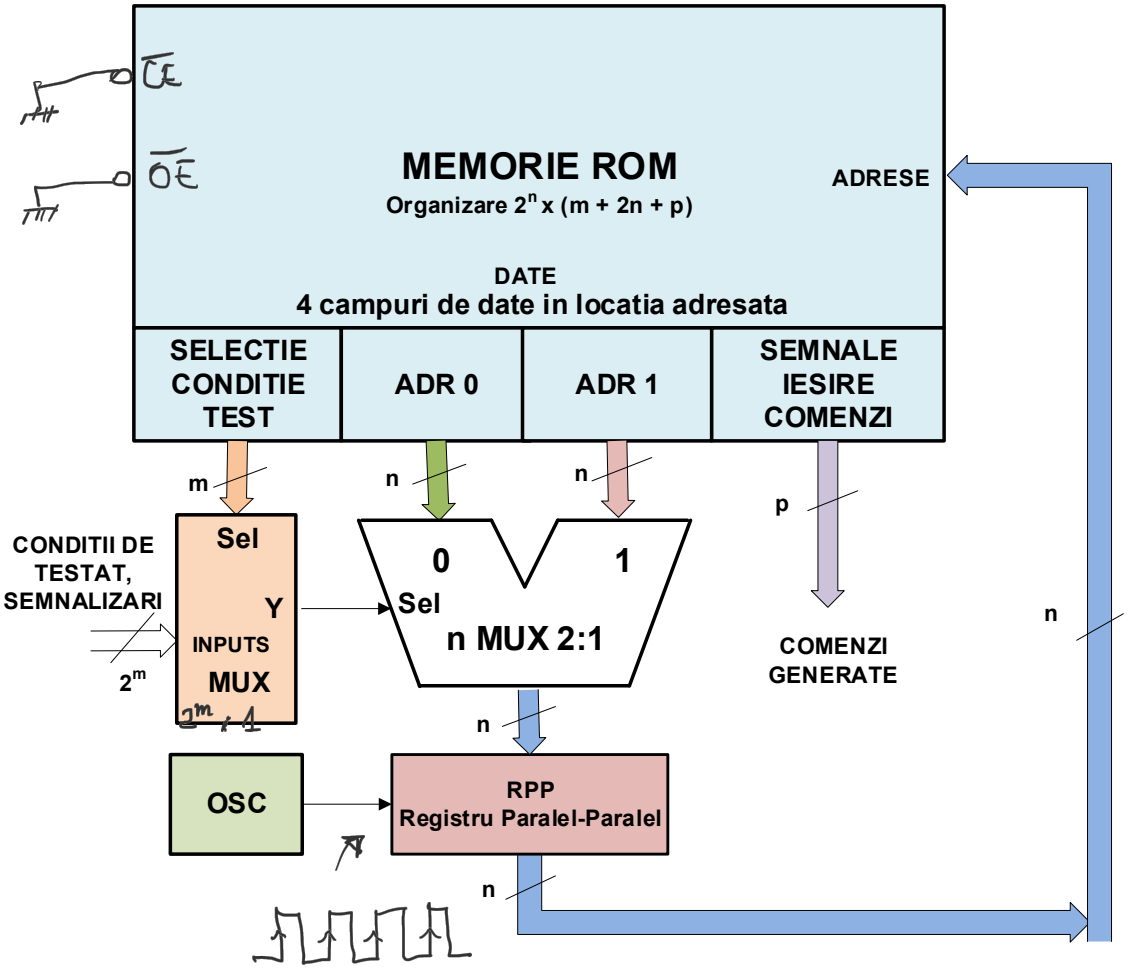
Functionarea unui automat se poate descrie prin mai multe forme:

- Limbaj natural;
- Diagrame de semnale;
- Diagrame de tranzitii a starilor si iesirilor;
- Scheme logice (organigrame).

Blocuri care apar in scheme logice (organigrame)



Automat microprogramat cu format fix al microinstrucțiunilor



Caracteristici ale automatului microprogramat cu format fix al microinstructiunilor

Fiecare locatie de memorie din memoria de microprograme corespunde unui bloc logic din organigrama. Fiecare bloc logic din organigrama reprezinta o stare in functionarea automatului microprogramat.

Continutul unei locatii de memorie a automatului este format din mai multe campuri de date, care pastreaza aceiasi dimensiune, indiferent de blocul logic corespondent din organigrama.

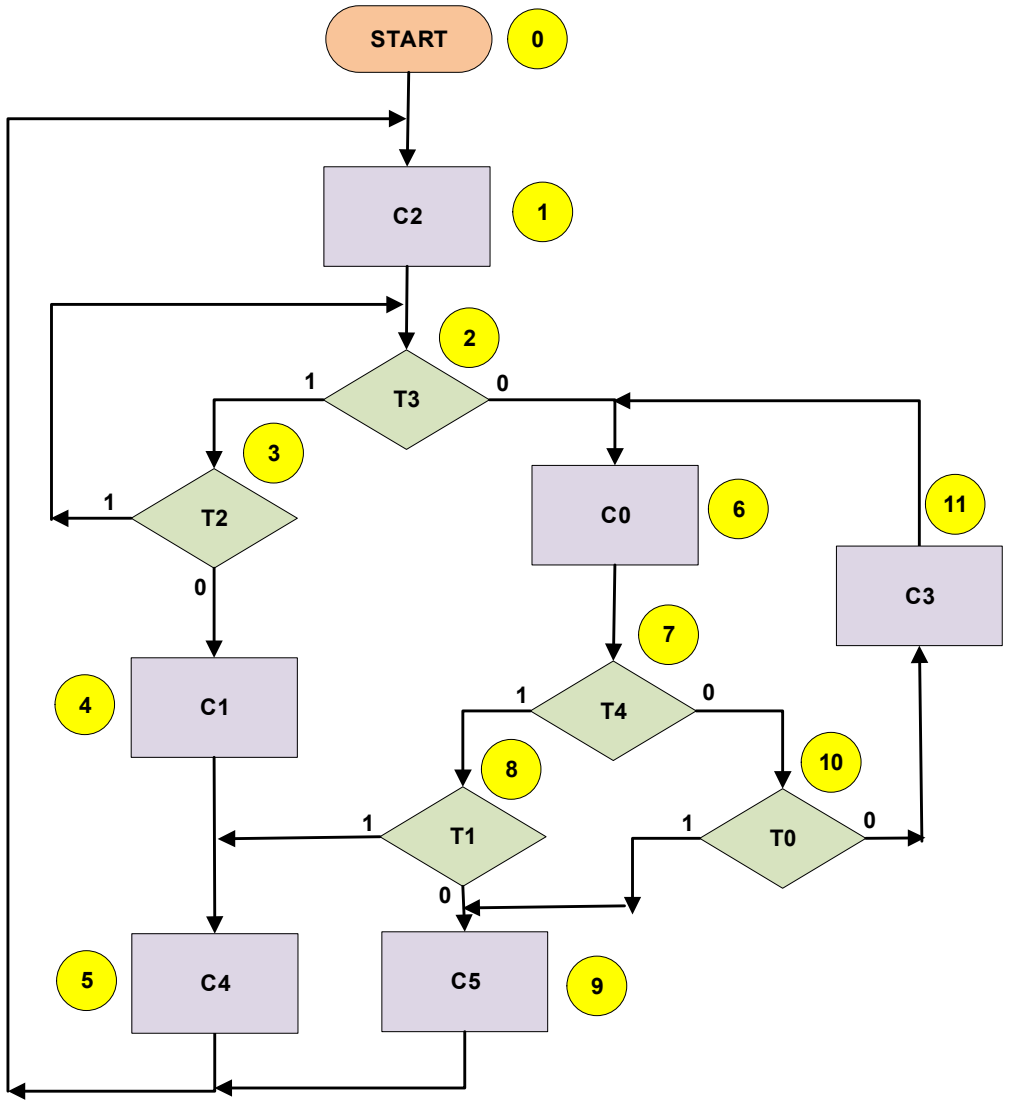
Semnificatia si dimensiunea fiecarui camp dintr-o locatie se pastreaza pentru toate blocurile logice din organigrama (format fix al microinstructiunii).

La blocurile logice de tip *Generare Comanda* (reprezentate prin dreptunghi in organigrama), campurile de date *ADR0* si *ADR1* coincid (genereaza o aceiasi adresa catre memoria ROM), iar continutul campului *Selectie Conditie Test* nu conteaza (don't care); campul de date *Semnale Iesire Comenzi* se completeaza conform organigramei.

La blocurile logice *Testare Conditie*, campurile de date *Selectie Conditie Test* , respectiv *ADR0* si *ADR1* se coreleaza cu organigrama; campul de date *Semnale Iesire Comenzi* se completeaza astfel incat comenzile de iesire sa nu fie actionate.

Aplicatie

Sa se realizeze automatul microprogramat cu format fix al microinstructiunilor care implementeaza urmatoarea schema logica (organigrama).



Implementarea ceruta de problema necesita raspunsul la 2 chestiuni:

- schema hardware a automatului; schema hardware este deja fixata, dar este nevoie de dimensionarea componentelor utilizate;
- continutul memoriei ROM din automat.

Metodologie de rezolvare:

1). Identificarea multimii semnalelor de intrare, dimensionarea MUX-ului care testeaza aceste semnale si alocarea lor pe intrarile MUX-ului de tip $2^m:1$;

2). Identificarea multimii semnalelor de iesire *(p)*;

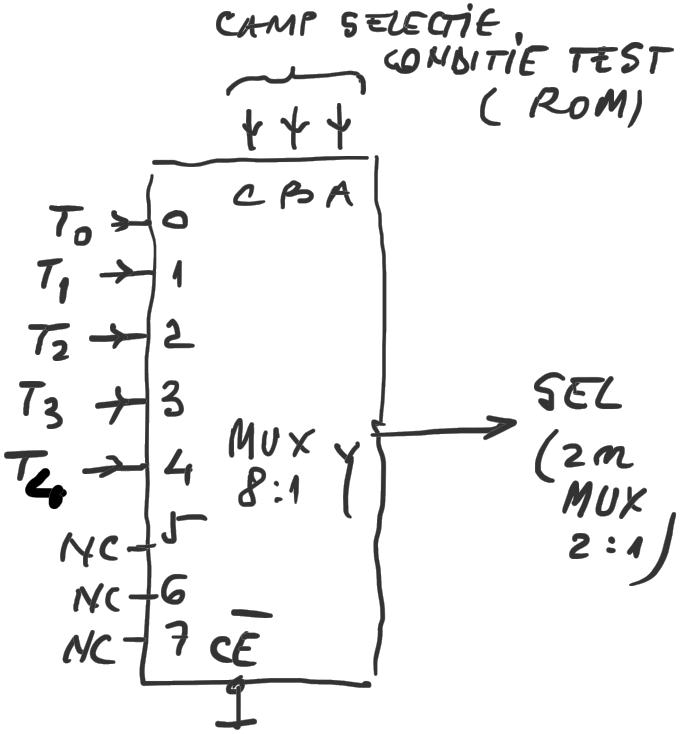
3). Identificarea multimii starilor prin care trece automatul (fiecarui bloc logic din organigrama ii este corespondenta o stare), de unde rezulta *n*, codificarea acestora, dimensionarea memoriei ROM $2^n \times (m + 2n + p)$ si a celor *nMUX2:1* si alocarea iesirilor memoriei ROM; implementarea cu memoriei cu circuite ROM tipizate;

4). Stabilirea continutului memoriei ROM.

SOLUTIE

1). Sunt 5 semnale de intrare care apar in schema logica (*T0, T1, T2, T3, T4*) si care trebuie testate de automat. Rezulta necesitatea unui MUX cu 8 intrari, dintre care 5 vor fi folosite pentru semnalele mentionate anterior, iar 3 raman neconectate (NC).
La MUX-ul 8:1 vom avea 3 semnale de selectie (*m=3*) notate C, B si A. Alocam semnalele de intrare conform urmatorului tabel:

Intrare MUX 8:1	Semnal alocat	Selectie CBA
0	T0	000
1	T1	001
2	T2	010
3	T3	011
4	T4	100
5	NC	101
6	NC	110
7	NC	111



2). Semnalele de iesire care apar in schema logica sunt in numar de 6 (*C0, C1, C2, C3, C4, C5*). Rezulta *p=6*. Vom considera starea activa a semnelor de iesire ca fiind 1-logic.

3). Numarul de stari prin care trece automatul este de 12. O solutie de codificare a starilor este prezentata in schema logica. Numarul de stari este numarul de locatii din memorie folosite in implementare. Memoria ROM trebuie sa aiba minimal 16 locatii (cea mai mica putere intrega a lui 2 care acopera numarul de stari folosit). Rezulta *n=4* (numarul de linii de adrese al memoriei ROM; le notam *A3, A2, A1, A0*). Numarul de iesiri al memoriei ROM este: *m + 2n + p = 3 + 8 + 6 = 17* (linii de date, notate *D16 – D0*). Vor fi necesare *n = 4* MUX-uri de tip 2:1 (notate MUX3, MUX2, MUX1, MUX0 conform structurii HW a automatului).
Alocarea celor 17 linii de date, *D16-D0*, pentru memoria ROM este prezentata in tabelul:

	Selectie semnale intrare testate MUX8:1 (camp 3)			**Adresa ADR0 – adresa urmatoare pentru semnal testat cu valoarea 0-logic (camp 2)**				**Adresa ADR1 urmatoare pentru semnal testat cu valoarea 1-logic (camp 1)**				**Semnale de iesire generate Comenzi (camp 0)**					
Linie date ROM	**D16**	**D15**	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Semnal conectat	C	B	A	In 0 MUX 3	In 0 MUX 2	In 0 MUX 1	In 0 MUX 0	In 1 MUX 3	In 1 MUX 2	In 1 MUX 1	In 1 MUX 0	C5	C4	C3	C2	C1	C0

x - don't care

4). Stabilirea continutului memoriei ROM.

END 20 mar 2025

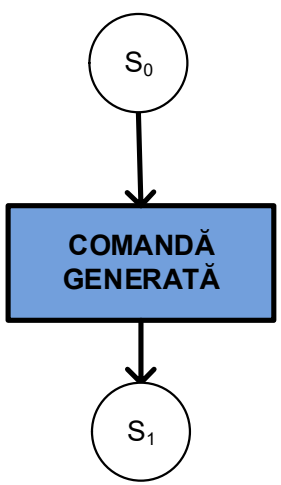
Se intocmeste tabelul urmator pentru continutul memoriei ROM:

STARE	ADRESE				DATE - camp 3 (Selectie test)			DATE - camp 2 (ADR0)				DATE - camp 1 (ADR1)				DATE - camp 0 (Comenzi)					
	A3	A2	A1	A0	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	x	x	x	0	0	0	1	0	0	0	1	0	0	0	0	0	0
1	0	0	0	1	x	x	x	0	0	1	0	0	0	1	0	0	0	0	1	0	0
2	0	0	1	0	0	1	1	0	1	1	0	0	0	1	1	0	0	0	0	0	0
3	0	0	1	1	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
4	0	1	0	0	x	x	x	0	1	0	1	0	1	0	1	0	0	0	0	1	0
5	0	1	0	1	x	x	x	0	0	0	1	0	0	0	1	0	1	0	0	0	0
6	0	1	1	0	x	x	x	0	1	1	1	0	1	1	1	0	0	0	0	0	1
7	0	1	1	1	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	1	1	0	0	1	0	1	0	1	0	0	0	0	0	0
9	1	0	0	1	x	x	x	0	0	0	1	0	0	0	1	1	0	0	0	0	0
10	1	0	1	0	0	0	0	1	0	1	1	1	0	0	1	0	0	0	0	0	0
11	1	0	1	1	x	x	x	0	1	1	0	0	1	1	0	0	0	1	0	0	0
12-15	y	y	y	y	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	0

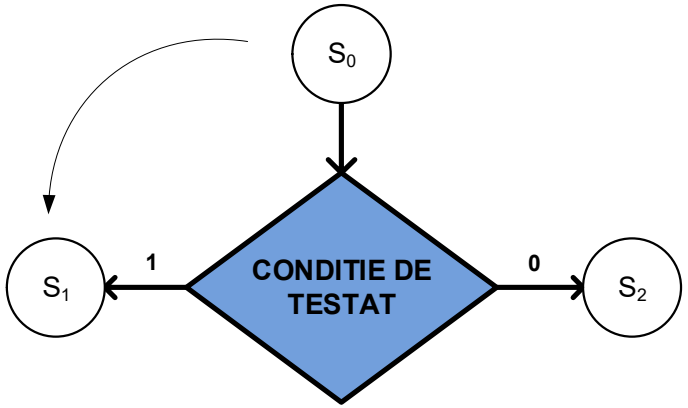
Automat microprogramat cu format variabil al microinstrucțiunilor

Automatul cu format variabil al microinstructiunilor urmareste optimizarea folosirii memoriei care intra in structura acestuia.

Optimizarea se bazeaza pe o codare specifica a starilor pe care un automat le parcurge in cadrul unei scheme logice (organigrama), functie de blocul logic traversat.
Astfel, la executia unui bloc logic de tip *Executie comanda*, se urmareste, pe cat este posibil, o codare a starilor care sa se obtina prin incrementare ($S_1 = S_0 + 1$), unde S_1 este starea urmatoare, iar S_0 este starea anterioara.
La executia unui bloc logic de tip *Testare conditie* codare va urmari ca pentru ramura de tip 1 - adevarat - starea urmatoare S_1 sa fie obtinuta din starea precedenta S_0 prin incrementare ($S_1 = S_0 + 1$), iar pentru ramura de tip 0 – fals, starea urmatoare S_2 nu este necesar sa aiba o legatura directa cu starea anterioara S_0.



BLOC LOGIC EXECUŢIE COMANDĂ



BLOC LOGIC TESTARE CONDIŢIE

Automat microprogramat cu format variabil al microinstrucțiunilor (cont)

Starile prin care trece automatul sunt de fapt adresele aplicate memoriei de microprograme. Fiecare stare este asociata unui bloc logic din organigrama. Continuturile adresate in fiecare stare se constituie in microinstructiunile prin care se efectueaza functionalitatile din blocurile logice ale organigramei (executia comenzilor, testarea semnalelor de intrare).

La automatul cu format fix al microinstructiunilor, adresarea memoriei de microprograme se face folosind un RPP (registru paralel-paralel) care permite doar functia de incarcare paralela, fiind actionat de semnalul de ceas in ritmul caruia se parcurge organigrama.

La automatul cu format variabil al microinstructiunilor, implementarea hardware se bazeaza pe utilizarea unui circuit prin care se adreseaza memoria de microprograme si care permite functia de incrementare, respectiv pe cea de incarcare paralela.

Un astfel de circuit este un numerator binar presetabil.

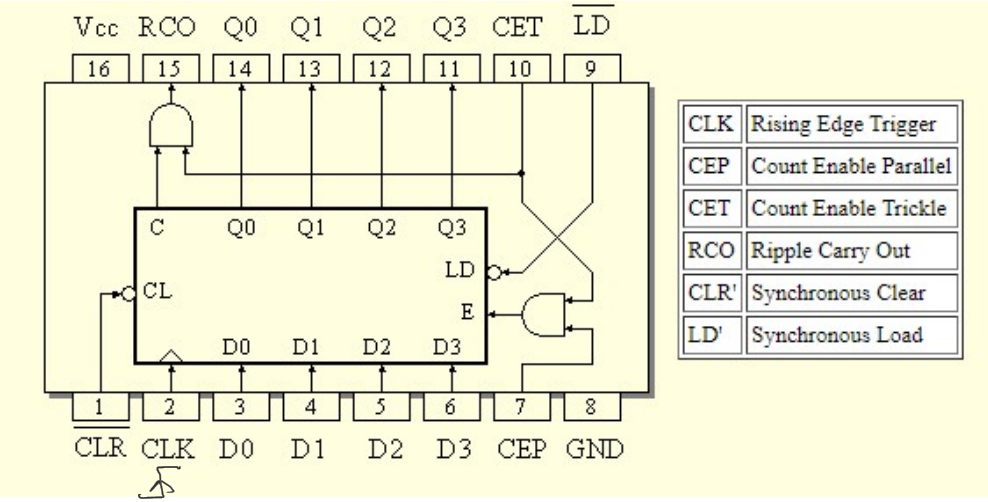
Acest circuit trebuie sa fie si sincron pentru ambele functii (numarare – incrementare, respectiv incarcare paralela), astfel incat starile prin care trece automatul sa fie sincrone cu semnalul de ceas aplicat numaratorului si in ritmul caruia se parcurge organigrama.

Dubla functionalitate a circuitului numarator presetabil permite optimizarea memoriei de microprograme a automatului, dar implica si necesitatea unei codari a starilor parcurse prin care sa se urmareasca parcurgerea acestora prin incrementare.

Exemple de circuite de tip numarator binar presetabil sincron

Circuitul 74163

Circuitul 74163 este un numarator binar pe 4 biti, cu functia de numarare catre inainte (incrementare), respectiv cu functia de incarcare paralela (presetare) sincrona, precum si cu functia de stergere sincrona.
Cele 3 functii mentionate se realizeaza pe frontal pozitiv al semnalului de ceas (CLK).



CLK	Rising Edge Trigger
CEP	Count Enable Parallel
CET	Count Enable Trickle
RCO	Ripple Carry Out
CLR'	Synchronous Clear
LD'	Synchronous Load

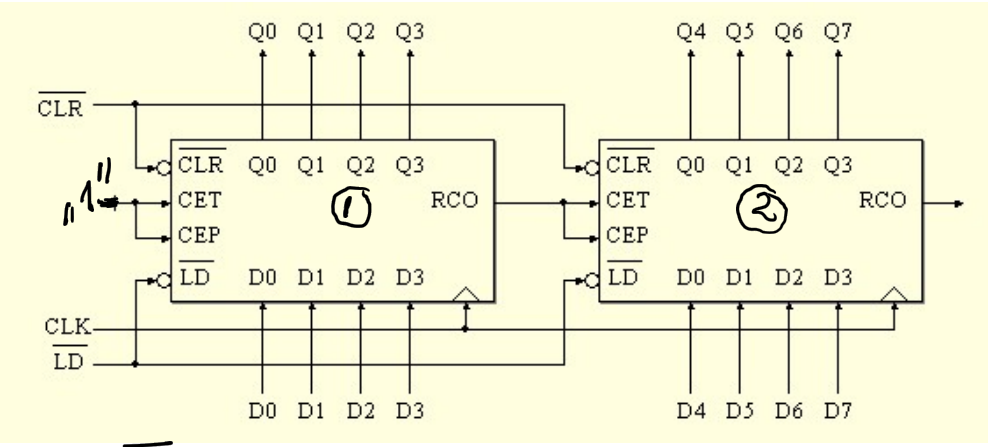
Din tabelul de adevar al circuitului se constata ca:

- functia de stergere are prioritatea maxima (se realizeaza la activarea semnalului LD, indiferent de starea semnalelor LD, respectiv CEP si CET);
- functia de presetare are urmatoarea prioritate (se realizeaza doar daca CLR este inactiv si indiferent de starea semnalelor CEP si CET);
- functia de numarare are prioritatea minimala (se realizeaza doar daca semnalele CLR si LD sunt inactive, respectiv daca CEP si CET sunt active)

Input					Output			Operation
CLR'	LD'	CLK	Enable		Q(t-1)	Q(t)	RCO	
			CEP	CET	Q3 - Q0	Q3 - Q0		
H	H	↑	H	H	< 15	Q(t-1) + 1	L	Count Up
H	H	↑	H	H	H H H L	H H H H	H	Count Up & Carry Out
L	X	↑	X	X	-	L L L L	L	Synchronous Clear
H	L	↑	X	X	-	D3 - D0	-	Load Data
H	H	-	L	H	H H H H	H H H H	H	Hold but Carry Out
H	H	-	X	L	-	-	L	Hold

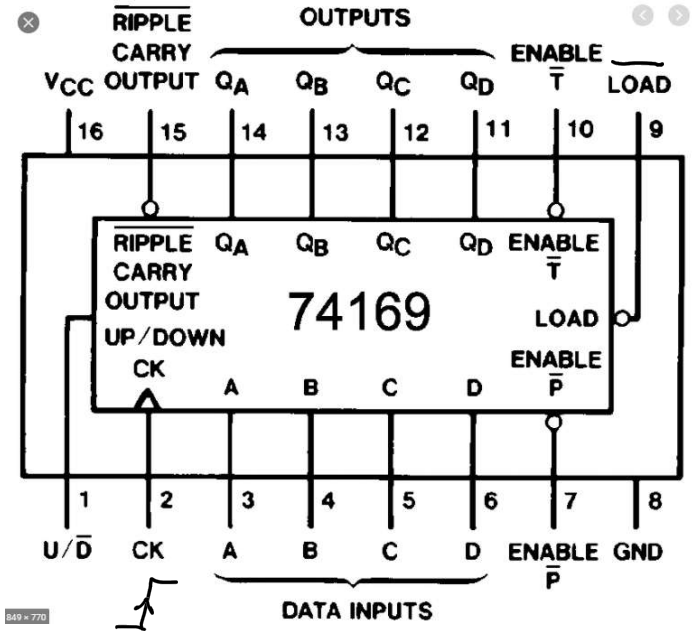
Circuitul 74163 permite realizarea de extinsii de numarare pe mai mult de 4 biti (multipli de 4 biti), care pe ansamblu au toate cele 3 functii mentionate sincrone (stergere – sau aducere la zero, incarcare paralela si numarare catre inainte).
Cel 3 functii se realizeaza pe frontul crescator al semnalului de ceas aplicat (CLK).
Se aplica acelasi semnal de ceas in mod comun circuitelor 74163 cu care se realizeaza extensia.

In figura se exemplifica o extensie pe 8 biti cu 2 circuite 74163.

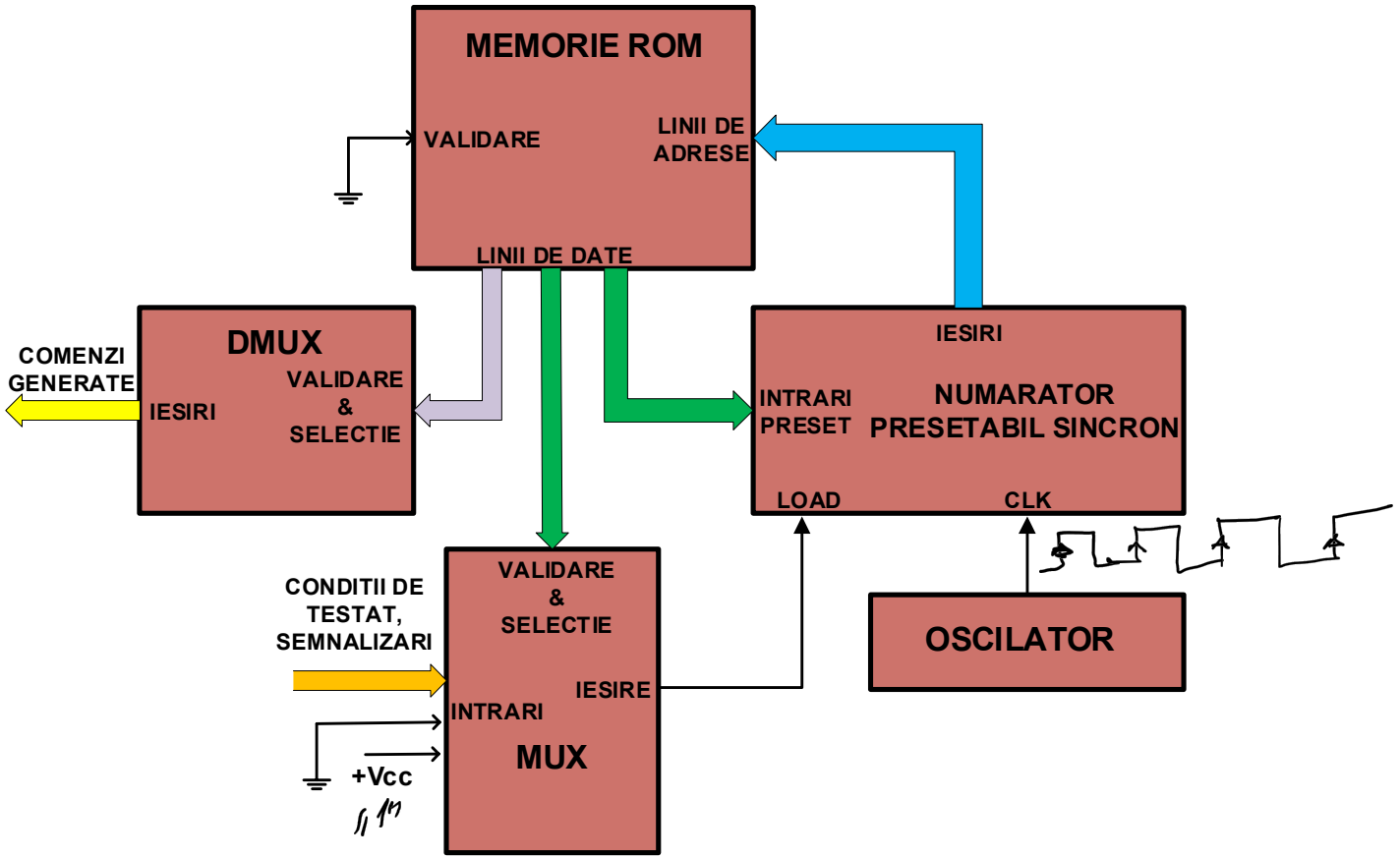


Circuitul 74169

Circuitul 74169 este un numarator binar pe 4 biti, cu functia de numarare catre inainte, respectiv inapoi (reversibil), cu functia de incarcare paralela (presetare) sincrona.
Cele 2 functii mentionate se realizeaza pe frontal pozitiv al semnalului de ceas (CLK).
Functia de incarcare paralela este prioritara fata de functia de numarare.
Circuitul poate realiza extensii sincrone cu functiile de numarare si incarcare paralela.
Circuitul nu dispune de functia de stergere (clear – sau aducere la zero).
Circuitul dispune de intrarea Up/Down (U/D) prin care se stabileste sensul de numarare.



Schema bloc a automatului microprogramat cu format variabil al microinstrucțiunilor



Explicatii pentru schema bloc a automatului microprogramat cu format variabil al microinstructiunilor

Fiecare bloc logic din organigrama are asociata o stare, identificata prin adresa locatiei respective de memorie.
Orice bloc logic din organigrama se executa prin folosirea continutului unei locatii de memorie (microinstructiune).
Cum in organigrame apar 2 tipuri de blocuri logice, pentru fiecare dintre acestea va corespunde cate un format al microinstructiunii prin care se executa, campurile de biti continuti fiind diferiti in fiecare format – de unde denumirea automatului.

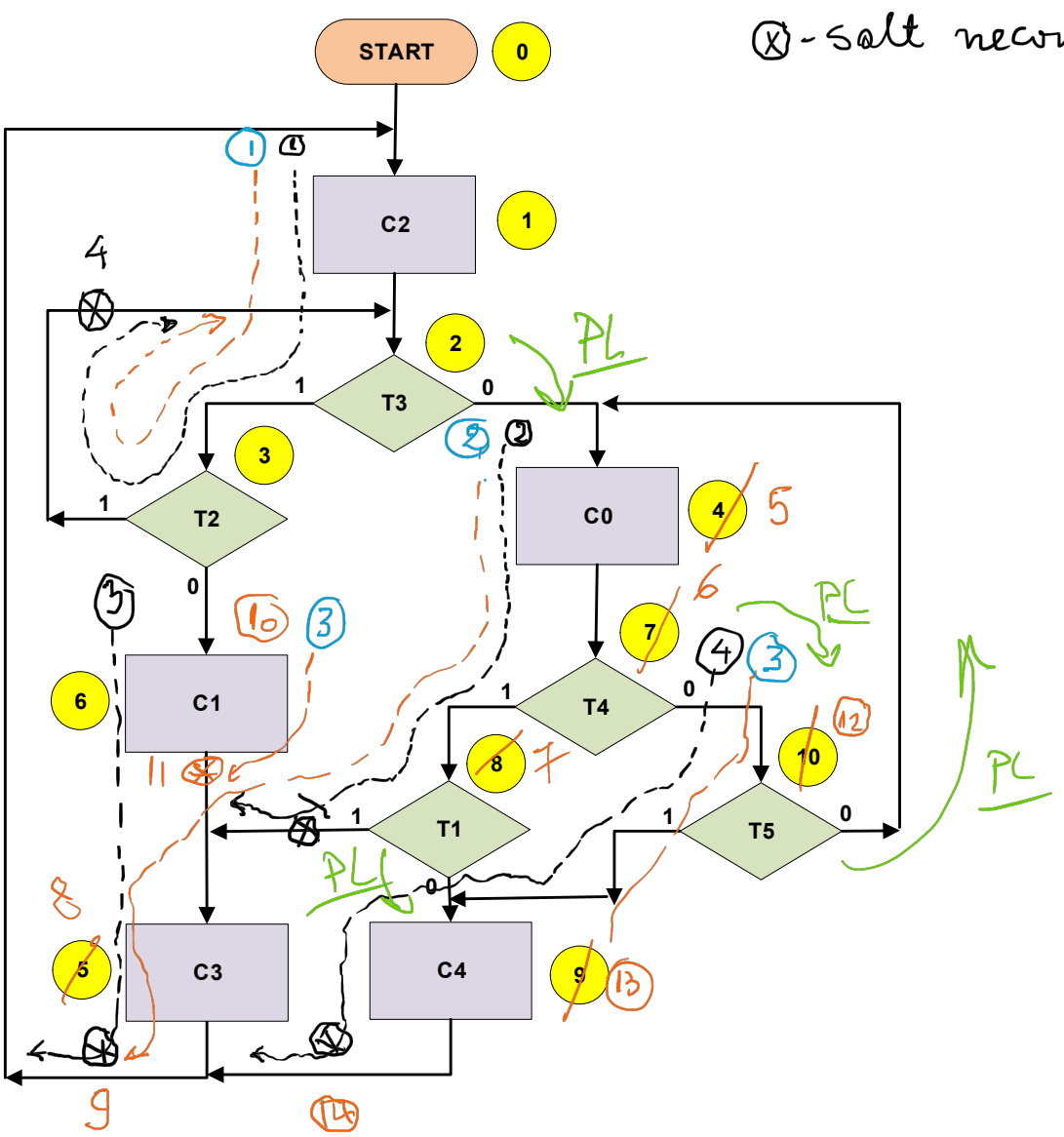
Microinstructiunea prin care se efectueaza blocurile logice de tip *Testare conditie*, contine 2 campuri de biti: *campul de biti care codeaza intrarea testata*, respectiv *campul de biti care reprezinta adresa de salt*, daca conditia testata indica efectuarea unui salt. Saltul se realizeaza in hardware utilizand functia de presetare a numaratorului. Campul de biti care codeaza intrarea testata este conectat la intrarile de selectie al unui MUX caruia i se aplica semnalele de ce trebuie testate. In plus, acest MUX prezinta 2 intrari suplimentare – una conectata permanent in 0-logic prin care se permite efectuarea de salturi neconditionate (prin functia de preset la numarator), iar o a 2-a conectata permanent in 1-logic care se utilizeaza cand trebuie efectuate blocuri logice de tip *Executie comanda* (prin functia de incrementare la numarator).
Dimensiunea *campului de biti care codeaza intrarea testata* este dat de relatia: $\log_2$(nr semnale de intrare testate+2).
Dimensiunea *campului care reprezinta adresa de salt* este dat de relatia: $\log_2$(nr stari prin care se trece in organigrama).

In microinstructiunea prin care se efectueaza blocurile logice de tip *Executie comanda*, se regaseste un *camp de biti care codeaza iesirile ce trebuie executate*, componenta hardware utilizata pentru executia comenzilor fiind un DMUX, respectiv un camp de biti prin care se activeaza DMUX-ul si, respectiv forteaza prin MUX efectuarea unei operatii de incrementare la nivelul numaratorului (selecteaza intrarea conectata in 1-logic a acestuia).

END 20 mar 2025

Problema

Sa se implementeze prin folosirea unui automat microprogramat cu format variabil al microinstructiunilor organigrama din figura.



Implementarea ceruta de problema necesita raspunsul la 2 chestiuni:

- schema hardware a automatului – este deja fixata, dar daca situatia impune, este necesara modificarea stucturii automatului si a dimensiunii componentelor utilizate; este nevoie de precizarea alocarii (asignarii) semnalelor de intrare si cele de iesire;
- continutul memoriei ROM din automat.

Metodologie de rezolvare:

1). Identificarea multimii semnalelor de intrare si alocarea lor pe intrarile MUX-ului; MUX-ul trebuie sa aiba suplimentar inca 2 intrari (una conectata la GND – pentru efectuarea salturilor neconditionate prin functia de incarcare paralela a numaratorului 74163, iar alta intrare conectata in 1-logic – pentru ca la efectuarea comenzilor, numaratorul 74163 sa efectueze functia de numarare);

2). Identificarea multimii semnalelor de iesire si alocarea lor pe iesirile DMUX-ului;

3). Identificarea multimii starilor prin care trece automatul (fiecarui bloc logic din organigrama ii este corespondenta o stare) si o prima codificare a acestora pe organigrama;

4). Recodificarea starilor prin care trece automatul, cu respectarea urmatoarelor reguli:

- identificarea traseelor din organigrama la parcurgerea carora numaratorul executa functia de numarare; toate blocurile logice din orgamigrama trebuie parcurse doar o singura data;
- identificarea locurilor din organigrama unde se impune efectuarea de salturi neconditionate si introducerea de stari suplimentare pe respectivele pozitii;
- renumerotarea starilor din organigrama pentru a tine seama de cele 2 reguli anterioare.

5). Stabilirea continutului memoriei ROM.

END 20 mar 2025